开发者指南，版本 2

AWS IoT Greengrass

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS IoT Greengrass: 开发者指南，版本 2

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商标和商业外观不得用于任何非 Amazon 的商品或服务，也不得以任何可能引起客户混淆、贬低或诋毁 Amazon 的方式使用。所有非 Amazon 拥有的其他商标均为各自所有者的财产，这些所有者可能附属于 Amazon、与 Amazon 有关联或由 Amazon 赞助，也可能不是如此。

Table of Contents

什么是 AWS IoT Greengrass？ 1
- 新功能 1
- 对于初次使用的用户 1
- 对于现有用户 2
- AWS IoT Greengrass 的工作原理 2
  - 重要概念 3
  - AWS IoT Greengrass 的特征 5
- 按操作系统划分的 Greengrass 功能兼容性 6

版本 2 中的新增内容 14
- AWS IoT Greengrass 酷睿 v2.12.4 软件更新 16
  - 公共组件更新 16
- AWS IoT Greengrass 酷睿 v2.12.3 软件更新 16
  - 公共组件更新 17
- AWS IoT Greengrass 酷睿 v2.12.2 软件更新 18
  - 公共组件更新 19
- AWS IoT Greengrass酷睿 v2.12.1 软件更新 20
  - 公共组件更新 20
- AWS IoT Greengrass酷睿 v2.12.0 软件更新 21
  - 公共组件更新 22
- AWS IoT Greengrass酷睿 v2.11.3 软件更新 22
  - 公共组件更新 23
- AWS IoT Greengrass酷睿 v2.11.2 软件更新 24
  - 公共组件更新 24
- AWS IoT Greengrass酷睿 v2.11.1 软件更新 25
  - 公共组件更新 25
- AWS IoT Greengrass酷睿 v2.11.0 软件更新 26
  - 公共组件更新 27
- AWS IoT Greengrass酷睿 v2.10.3 软件更新 28
  - 公共组件更新 28
- AWS IoT Greengrass酷睿 v2.10.2 软件更新 29
  - 公共组件更新 29
- AWS IoT Greengrass酷睿 v2.10.1 软件更新 31
  - 公共组件更新 31
- AWS IoT Greengrass酷睿 v2.10.0 软件更新 32

公共组件更新 ... 32
AWS IoT Greengrass酷睿 v2.9.6 软件更新 ... 33
公共组件更新 ... 34
AWS IoT Greengrass酷睿 v2.9.5 软件更新 ... 34
公共组件更新 ... 35
AWS IoT Greengrass酷睿 v2.9.4 软件更新 ... 35
公共组件更新 ... 36
AWS IoT Greengrass酷睿 v2.9.3 软件更新 ... 37
公共组件更新 ... 37
AWS IoT Greengrass酷睿 v2.9.2 软件更新 ... 38
公共组件更新 ... 38
AWS IoT Greengrass酷睿 v2.9.1 软件更新 ... 39
公共组件更新 ... 39
AWS IoT Greengrass酷睿 v2.9.0 软件更新 ... 40
公共组件更新 ... 41
AWS IoT Greengrass酷睿 v2.8.1 软件更新 ... 42
公共组件更新 ... 43
AWS IoT Greengrass酷睿 v2.8.0 软件更新 ... 43
公共组件更新 ... 44
AWS IoT Greengrass酷睿 v2.7.0 软件更新 ... 45
公共组件更新 ... 46
AWS IoT Greengrass酷睿 v2.6.0 软件更新 ... 47
公共组件更新 ... 48
AWS IoT Greengrass酷睿 v2.5.6 软件更新 ... 51
公共组件更新 ... 51
AWS IoT Greengrass酷睿 v2.5.5 软件更新 ... 52
公共组件更新 ... 52
AWS IoT Greengrass酷睿 v2.5.4 软件更新 ... 53
公共组件更新 ... 54
AWS IoT Greengrass酷睿 v2.5.3 软件更新 ... 54
公共组件更新 ... 55
AWS IoT Greengrass酷睿 v2.5.2 软件更新 ... 56
公共组件更新 ... 56
AWS IoT Greengrass酷睿 v2.5.1 软件更新 ... 57
公共组件更新 ... 57
AWS IoT Greengrass酷睿 v2.5.0 软件更新 ... 58

平台支持更新 59
公共组件更新 59
AWS IoT Greengrass酷睿 v2.4.0 软件更新 62
公共组件更新 63
AWS IoT Greengrass酷睿 v2.3.0 软件更新 65
公共组件更新 65
AWS IoT Greengrass酷睿 v2.2.0 软件更新 66
公共组件更新 67
AWS IoT Greengrass酷睿 v2.1.0 软件更新 69
平台支持更新 70
公共组件更新 71
AWS IoT Greengrass酷睿 v2.0.5 软件更新 76
公共组件更新 76
AWS IoT Greengrass酷睿 v2.0.4 软件更新 77
公共组件更新 77
从版本 1 迁移 79
我能否在 V2 上运行我的 V1 应用程序？ 79
迁移概述 79
V1 和 V2 之间的区别 80
验证 V1 核心设备能否运行 V2 软件 88
设置新的 V2 核心设备 88
第 1 步：在新设备上安装 Greengrass V2 88
步骤 2：创建和部署 V2 组件以迁移 V1 应用程序 89
第 3 步：测试您的 V2 应用程序 93
将 V1 核心设备升级到 V2 93
步骤 1：安装AWS IoT Greengrass核心软件 v2.x 93
第 2 步：将 Greengrass V2 组件部署到核心设备 96
开始使用 98
先决条件 99
步骤 1：设置 AWS 账户 100
注册 AWS 账户 100
创建管理用户 101
第 2 步：设置您的环境 102
步骤 3 安装AWS IoT Greengrass核心软件 107
安装AWS IoT Greengrass核心软件（控制台） 107
安装AWS IoT Greengrass核心软件 (CLI) 111

运行 Greengrass 软件 (Linux) 116
验证设备上是否安装了 Greengrass CLI 117
第 4 步：在设备上开发和测试组件 118
步骤 5：在AWS IoT Greengrass服务中创建您的组件 129
步骤 6：部署您的组件 140
后续步骤 144
设置 Greengrass 核心设备 145
支持的平台和要求 145
支持的平台 145
设备要求 146
Lambda 函数要求 149
Windows 设备的功能注意事项 150
设置一个 AWS 账户 151
安装 AWS IoT Greengrass Core 软件 152
使用自动配置进行安装 154
使用手动配置进行安装 167
使用队列配置进行安装 202
使用自定义配置进行安装 243
安装程序参数 258
运行AWS IoT Greengrass核心软件 262
检查 AWS IoT Greengrass Core 软件是否作为系统服务运行 263
将 AWS IoT Greengrass Core 软件作为系统服务运行 265
在没有系统服务的情况下运行 C AWS IoT Greengrass ore 软件 265
在 Docker AWS IoT Greengrass 中运行 266
支持的平台和要求 266
软件下载 267
选择如何配置 AWS 资源 267
从 Dockerfile 中生成AWS IoT Greengrass镜像 268
通过自动AWS IoT Greengrass配置在 Docker 中运行 273
使用手动AWS IoT Greengrass配置在 Docker 中运行 280
对 Docker 容器中的 AWS IoT Greengrass 执行问题排查 299
配置 AWS IoT Greengrass 核心软件 302
部署 Greengrass nucleus 组件 303
将 Greengrass 核心配置为系统服务 303
使用 JVM 选项控制内存分配 306
配置运行组件的用户 308

配置系统资源限制 ... 312
通过端口 443 或网络代理进行连接 ... 314
使用由私有 CA 签名的设备证书 ... 321
配置 MQTT 超时和缓存设置 ... 321
更新AWS IoT Greengrass核心软件 (OTA) ... 322
要求 ... 322
核心设备的注意事项 ... 322
Greengrass 核更新行为 ... 323
执行 OTA 更新 ... 324
卸载AWS IoT Greengrass核心软件 ... 324
教程 ... 328
开发一个可以延迟组件更新的组件 ... 328
先决条件 ... 329
第 1 步：安装 Greengrass 开发套件 CLI ... 330
第 2 步：开发可延迟更新的组件 ... 331
步骤 3：将组件发布到AWS IoT Greengrass服务 ... 339
步骤 4：在核心设备上部署和测试组件 ... 342
通过 MQTT 与本地物联网设备互动 ... 347
先决条件 ... 347
步骤 1：查看并更新核心设备AWS IoT政策 ... 348
步骤 2：启用客户端设备支持 ... 349
步骤 3：Connect 客户端设备 ... 354
步骤 4：开发可与客户端设备通信的组件 ... 357
步骤 5：开发与客户端设备影子交互的组件 ... 364
开始使用 SageMaker 边缘管理器 ... 388
先决条件 ... 389
在 SageMaker 边缘管理器中设置 ... 391
创建示例组件 ... 392
运行样本图像分类推理 ... 393
执行样本图像分类推断 ... 397
先决条件 ... 397
步骤 1：订阅默认通知主题 ... 398
步骤 2：部署 TensorFlow Lite 图像分类组件 ... 399
步骤 3：查看推理结果 ... 400
后续步骤 ... 402
对来自相机的图像执行样本图像分类推断 ... 402

先决条件 403
步骤 1：在设备上配置摄像头模块 404
第 2 步：验证您对默认通知主题的订阅 406
步骤 3：修改 TensorFlow Lite 图像分类组件配置并进行部署 406
步骤 4：查看推理结果 408
后续步骤 409
组件 410
AWS-提供的组件 410
Greengrass 核 418
客户端设备身份验证 447
CloudWatch 指标 503
AWS IoT Device Defender 525
磁盘后台处理程序 539
Docker 应用程序管理器 543
Kinesis Video Streams 的边缘连接器 551
Greengrass CLI 558
IP 探测器 569
Firehose 577
Lambda 启动器 592
Lambda 管理器 596
Lambda 运行时 604
旧版订阅路由器 606
本地调试控制台 616
日志管理器 630
机器学习组件 665
modbus-RTU 协议适配器 775
MQTT 网桥 804
MQTT 3.1.1 经纪商 (Moquette) 826
MQTT 5 经纪商 (EMQX) 832
Nucleus 遥测发射器 847
PKCS #11 提供商 858
秘密经理 865
安全隧道 874
影子经理 883
Amazon SNS 907
流管理器 922

- Systems Manager 代理 934
- 代币兑换服务 940
- 物联网 SiteWise OPC-UA 采集器 943
- 物联网 SiteWise OPC-UA 数据源模拟器 951
- 物联网 SiteWise 发行商 953
- 物联网 SiteWise 处理器 962

发布商支持的组件 973
- aishield.edge 973
- AI EdgeLabs 传感器 974
- Greengrass S3 Ingestor 974

社区组件 975

Greengrass 开发工具 978
- Greengrass 开发套件 CLI 979
- Greengrass 命令行界面 1006
- 使用 Greengrass 测试框架 1023

开发组件 1037
- 组件生命周期 1038
- 组件类型 1039
- 创建组件 1040
- 使用本地部署测试组件 1051
- 发布要部署的组件 1053
- 与AWS服务互动 1058
- 运行 Docker 容器 1062
- 食谱参考 1083
- 环境变量 1109

将组件部署到设备 1111
- 核心设备部署 1111
- 平台依赖关系解决方案 1111
- 组件依赖关系解析 1111
- 从事物组中移除设备 1112
- 部署 1113
- 部署选项 1113
- 创建部署 1115
- 创建子部署 1132
- 修改部署 1135
- 取消作业 1137

检查部署状态 ... 1138
日志记录和监控 ... 1142
监控工具 ... 1142
监控 Greengrass 日志 ... 1143
访问文件系统日志 ... 1143
访问 CloudWatch 日志 ... 1145
访问系统服务日志 ... 1147
启用记录到 CloudWatch 日志 ... 1149
为 AWS IoT Greengrass 配置日志记录 ... 1150
AWS CloudTrail日志 ... 1152
使用记录 API 调用 CloudTrail ... 1152
AWS IoT Greengrass V2 信息在 CloudTrail ... 1152
AWS IoT Greengrass 中的数据事件 CloudTrail ... 1153
AWS IoT Greengrass 中的管理事件 CloudTrail ... 1157
了解 AWS IoT Greengrass V2 日志文件条目 ... 1157
收集系统运行状况遥测数据 ... 1159
遥测指标 ... 1160
配置遥测代理设置 ... 1163
订阅遥测数据 EventBridge ... 1163
获取部署和组件运行状况通知 ... 1171
部署状态更改事件 ... 1172
组件状态更改事件 ... 1173
创建 EventBridge 规则的先决条件 ... 1175
配置设备运行状况通知（控制台） ... 1176
配置设备运行状况通知 (CLI) ... 1177
配置设备运行状况通知 (AWS CloudFormation) ... 1178
另请参阅 ... 1178
检查核心设备状态 ... 1178
检查核心设备的运行状况 ... 1179
检查核心设备组的运行状况 ... 1179
检查核心设备组件状态 ... 1180
运行 Lambda 函数 ... 1181
要求 ... 1181
配置 Lambda 函数生命周期 ... 1182
配置 Lambda 函数容器化 ... 1183
将 Lambda 函数作为组件导入（控制台） ... 1185

步骤 1：选择要导入的 Lambda 函数 1185
步骤 2：配置 Lambda 函数参数 1186
步骤 3：（可选）为 Lambda 函数指定支持的平台 1187
步骤 4：（可选）为 Lambda 函数指定组件依赖关系 1188
步骤 5：（可选）在容器中运行 Lambda 函数 1189
步骤 6：创建 Lambda 函数组件 1190
导入 Lambda 函数 (CLI) 1190
步骤 1：定义 Lambda 函数配置 1191
步骤 2：创建 Lambda 函数组件 1209
与 Greengrass 核、其他组件进行通信 AWS IoT Core 1212
IPC 客户端版本 1213
受支持的 SDK 1213
Connect 到 C AWS IoT Greengrass ore IPC 服务 1214
授权组件执行 IPC 操作 1220
授权策略中的通配符 1221
授权策略中的配方变量 1221
授权策略中的特殊字符 1221
授权策略示例 1222
订阅 IPC 事件直播 1226
定义订阅处理程序 1226
订阅处理程序示例 1228
IPC 最佳实践 1236
发布/订阅本地消息 1238
SDK 的最低版本 1238
授权 1239
PublishToTopic 1240
SubscribeToTopic 1249
示例 1261
发布/订阅 AWS IoT Core MQTT 消息 1283
SDK 的最低版本 1283
授权 1284
PublishToIotCore 1288
SubscribeToIotCore 1297
示例 1311
与组件生命周期交互 1319
SDK 的最低版本 1320

- 授权 1320
- UpdateState 1321
- SubscribeToComponentUpdates 1322
- DeferComponentUpdate 1323
- PauseComponent 1324
- ResumeComponent 1326

与组件配置交互 1327
- SDK 的最低版本 1327
- GetConfiguration 1328
- UpdateConfiguration 1329
- SubscribeToConfigurationUpdate 1330
- SubscribeToValidateConfigurationUpdates 1331
- SendConfigurationValidityReport 1332

检索秘密值 1333
- SDK 的最低版本 1333
- 授权 1333
- GetSecretValue 1334
- 示例 1340

与局部阴影互动 1346
- SDK 的最低版本 1346
- 授权 1347
- GetThingShadow 1358
- UpdateThingShadow 1365
- DeleteThingShadow 1373
- ListNamedShadowsForThing 1378

管理本地部署和组件 1385
- SDK 的最低版本 1386
- 授权 1387
- CreateLocalDeployment 1389
- ListLocalDeployments 1391
- GetLocalDeploymentStatus 1392
- ListComponents 1393
- GetComponentDetails 1394
- RestartComponent 1395
- StopComponent 1396
- CreateDebugPassword 1397

对客户端设备进行身份验证和授权 1398
SDK 的最低版本 1398
授权 1399
VerifyClientDeviceIdentity 1400
GetClientDeviceAuthToken 1401
AuthorizeClientDeviceAction 1402
SubscribeToCertificateUpdates 1403
与本地物联网设备互动 1405
客户机设备组件 1405
Connect 客户端设备与核心设备连接 1407
要求 1408
用于支持客户端设备的 Greengrass 组件 1419
配置云发现（控制台） 1421
配置云发现 (AWS CLI) 1421
关联客户端设备 1422
离线时对客户端进行身份验证 1424
管理核心设备端点 1425
选择一个 MQTT 经纪商 1430
连接到 MQTT 代理 1431
测试通信 1433
Greengrass 发现 RESTful API 1444
在客户端设备之间中继 MQTT 消息和 AWS IoT Core 1450
配置和部署 MQTT 网桥组件 1450
中继 MQTT 消息 1451
在组件中与客户端设备交互 1452
配置和部署 MQTT 网桥组件 1453
从客户端设备接收 MQTT 消息 1454
向客户端设备发送 MQTT 消息 1454
与客户端设备影子进行交互并进行同步 1455
先决条件 1455
启用影子管理器与客户端设备通信 1456
与组件中的客户端设备阴影交互 1458
将客户端设备阴影与同步 AWS IoT Core 1459
排查问题 1459
Greengrass 发现问题 1459
MQTT 连接问题 1465

与设备阴影互动 ... 1471
与组件中的阴影交互 ... 1471
检索和修改阴影状态 ... 1472
对阴影状态变化做出反应 ... 1472
将本地设备阴影与同步 AWS IoT Core ... 1473
先决条件 ... 1474
配置影子管理器组件 ... 1474
同步局部阴影 ... 1476
影子合并冲突行为 ... 1476
管理数据流 ... 1477
流管理工作流 ... 1477
要求 ... 1478
数据安全性 ... 1479
本地数据安全性 ... 1479
客户端身份验证 ... 1479
另请参阅 ... 1480
创建使用流管理器的自定义组件 ... 1480
定义使用流管理器的组件配方 ... 1480
在应用程序代码中Connect 流管理器 ... 1492
StreamManagerClient 用于处理直播 ... 1495
创建消息流 ... 1496
附加消息 ... 1499
读取消息 ... 1505
列出流 ... 1508
描述消息流 ... 1509
更新消息流 ... 1511
删除消息流 ... 1515
另请参阅 ... 1516
导出支持的云端目标的配置 ... 1517
配置流管理器 ... 1531
流管理器参数 ... 1531
另请参阅 ... 1533
执行机器学习推理 ... 1534
AWS IoT Greengrass ML 推理的工作原理 ... 1534
AWS IoT Greengrass版本 2 有什么不同？ ... 1535
要求 ... 1535

支持的模型源 ... 1536
支持的运行时 ... 1536
机器学习组件 ... 1536
使用 SageMaker 边缘管理器 ... 1541
工作方式 ... 1541
要求 ... 1542
开始使用 SageMaker 边缘管理器 ... 1544
使用 Lookout for Vision ... 1544
自定义您的机器学习组件 ... 1545
修改公共推理组件的配置 ... 1545
使用带有示例推理组件的自定义模型 ... 1547
创建自定义机器学习组件 ... 1550
创建自定义推理组件 ... 1553
排查问题 ... 1559
无法获取库 ... 1560
Cannot open shared object file ... 1561
Error: ModuleNotFoundError: No module named '<library>' ... 1561
未检测到支持 CUDA 的设备 ... 1562
没有这样的文件或目录 ... 1562
RuntimeError: module compiled against API version 0xf but this version of NumPy is <version> ... 1563
picamera.exc.PiCameraError: Camera is not enabled ... 1564
内存错误 ... 1564
磁盘空间错误 ... 1564
超时错误 ... 1564
使用管理核心设备 AWS Systems Manager ... 1565
安装 Systems Manager 代理 ... 1566
步骤 1：完成常规 Systems Manager 设置步骤 ... 1566
步骤 2：为 Systems Manager 创建 IAM 服务角色 ... 1566
步骤 3：为令牌交换角色添加权限 ... 1566
步骤 4：部署 Systems Manager 代理组件 ... 1571
步骤 5：使用 Systems Manager 验证核心设备的注册情况 ... 1573
卸载Systems Manager 代理 ... 1574
步骤 1：从Systems Manager 器注销核心设备 ... 1575
步骤 2：卸载Systems Manager 代理组件 ... 1575
第 3 步：卸载Systems Manager 代理软件 ... 1576

安全性 1577
数据保护 1578
数据加密 1579
硬件安全性集成 1580
设备身份验证和授权 1590
X.509 证书 1591
AWS IoT 策略 1592
更新核心设备的AWS IoT政策 1596
最低限度AWS IoT政策 1601
支持客户端设备的最低AWS IoT政策 1603
客户端设备的最低AWS IoT政策 1605
Identity and Access Management 1607
受众 1607
使用身份进行身份验证 1608
使用策略管理访问 1610
另请参阅 1612
AWS IoT Greengrass 如何与 IAM 协同工作 1612
基于身份的策略示例 1616
授权核心设备与AWS服务 1618
安装程序配置资源的最低 IAM 政策 1623
Greengrass 服务角色 1626
AWS 托管策略 1634
防止跨服务混淆代理 1640
排查身份和访问权限问题 1640
允许设备流量通过代理或防火墙 1642
基本操作的终端节点 1642
使用自动配置进行安装的终端节点 1645
AWS提供的组件的端点 1646
合规性验证 1646
故障恢复能力 1647
基础设施安全性 1648
配置和漏洞分析 1648
代码完整性 1649
VPC 端点 (AWS PrivateLink) 1650
AWS IoT Greengrass VPC 端点注意事项 1650
为 AWS IoT Greengrass 控制平面操作创建接口 VPC 端点 1651

为 AWS IoT Greengrass 创建 VPC 端点策略 1651
在 VPC 中操作AWS IoT Greengrass核心设备 1652
安全最佳实践 1656
授予可能的最低权限 1656
不要在 Greengrass 组件中对凭据进行硬编码 1656
不要记录敏感信息 1657
使设备时钟保持同步 1657
密码套件推荐 1657
另请参阅 1657
用AWS IoT Device Tester于 AWS IoT Greengrass V2 1658
AWS IoT Greengrass 资格认证套件 1658
自定义测试套件 1659
支持的版本 1659
适用 AWS IoT Greengrass 于 V2 的最新 IDT 版本 1659
不支持的 for V2 AWS IoT Device Tester 版本 AWS IoT Greengrass 1660
下载适用于 V2 的 AWS IoT Greengrass IDT 1665
手动下载 IDT 1665
以编程方式下载 IDT 1666
使用 IDT 运行 AWS IoT Greengrass 资格套件 1671
测试套件版本 1672
测试组描述 1672
先决条件 1675
配置设备以运行 IDT 测试 1695
配置 IDT 设置 1704
运行 AWS IoT Greengrass 资格认证套件 1715
了解结果和日志 1718
使用 IDT 开发和运行自己的测试套件 1721
下载适用于 AWS IoT Greengrass 的最新版本的 IDT 1675
测试套件创建工作流程 1722
教程：构建和运行示例 IDT 测试套件 1723
教程：开发一个简单的 IDT 测试套件 1728
创建 IDT 测试套件配置文件 1737
配置 IDT 测试管弦乐队 1744
配置 IDT 状态机 1750
创建 IDT 测试用例可执行文件 1772
使用 IDT 上下文 1778

为测试运行者配置设置 ... 1782
调试和运行自定义测试套件 ... 1792
查看 IDT 测试结果和日志 ... 1795
ID用 ... 1801
对 IDT 进行故障排除AWS IoT GreengrassV2 ... 1807
在哪里寻找错误 ... 1807
正在解决 IDTAWS IoT GreengrassV2 错误 ... 1808
的 Support AWS IoT Device Tester 政策 AWS IoT Greengrass ... 1814
基于 Greengrass 的物联网解决方案 ... 1815
欧洲科技大学 ... 1815
故障排除 ... 1816
查看 AWS IoT Greengrass 核心软件和组件日志 ... 1816
AWS IoT Greengrass 核心软件问题 ... 1816
无法设置核心设备 ... 1818
无法将 AWS IoT Greengrass Core 软件作为系统服务启动 ... 1818
无法将 nucleus 设置为系统服务 ... 1818
无法连接到 AWS IoT Core ... 1818
内存不足错误 ... 1819
无法安装 Greengrass CLI ... 1819
User root is not allowed to execute ... 1819
com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with ... 1819
Failed to map segment from shared object: operation not permitted ... 1820
无法设置 Windows 服务 ... 1820
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager ... 1821
com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime ... 1821
software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid ... 1821
software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy ... 1822
Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request ... 1823
Operation aws.greengrass#<operation> is not supported by Greengrass ... 1823
java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied) ... 1824

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist ... 1824
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn> . 1824
software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed ... 1825
java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi ... 1826
com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED ... 1826
Greengrass core device stuck on nucleus v2.12.3 ... 1826
AWS IoT Greengrass 云问题 ... 1828
An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null ... 1829
Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed} ... 1829
INACTIVE deployment status ... 1829
核心设备部署问题 ... 1829
Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact ... 1831
Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption. ... 1832
Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements> ... 1832
software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility ... 1833
com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component ... 1833

Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service ... 1834
Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration ... 1835
Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy ... 1835
Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration ... 1835
Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null) ... 1836
核心设备组件问题 ... 1836
Warn: '<command>' is not recognized as an internal or external command ... 1837
Python 脚本不记录消息 ... 1837
更改默认配置时组件配置不会更新 ... 1838
awsiot.greengrasscoreipc.model.UnauthorizedError ... 1839
com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>" ... 1840
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400) ... 1840
com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403) ... 1842
com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers ... 1842
Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>" ... 1842
copyFrom: <configurationPath> is already a container, not a leaf ... 1843
com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.' ... 1843
java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired. ... 1844
aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant ... 1845
核心设备 Lambda 函数组件问题 ... 1845

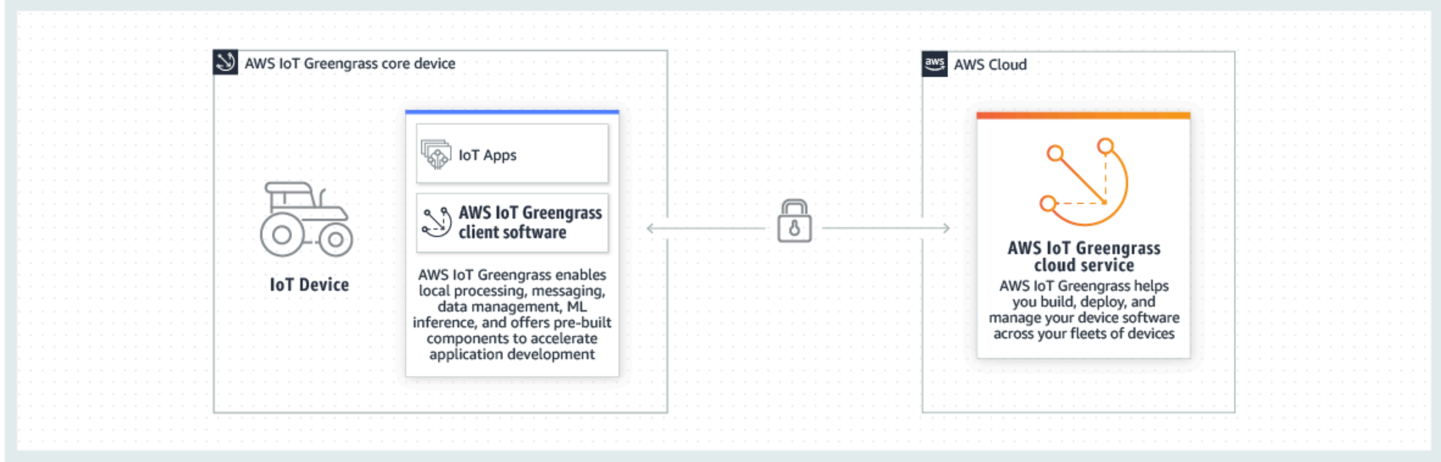
The following cgroup subsystems are not mounted: devices, memory ... 1846
ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn> ... 1846
组件版本已停产 ... 1846
Greengrass CLI 问题 ... 1847
java.lang.RuntimeException: Unable to create ipc client ... 1847
AWS CLI 问题 ... 1847
Error: Invalid choice: 'greengrassv2' ... 1848
详细的部署错误代码 ... 1848
权限错误 ... 1849
请求错误 ... 1851
组件配方错误 ... 1853
AWS组件错误、用户组件错误、组件错误 ... 1854
设备错误 ... 1855
依赖性错误 ... 1856
HTTP 错误 ... 1857
网络错误 ... 1858
核错误 ... 1858
服务器错误 ... 1859
云服务错误 ... 1860
通用错误 ... 1861
未知错误 ... 1861
详细的组件状态码 ... 1862
标记资源 ... 1865
在 AWS IoT Greengrass V2 中使用标签 ... 1865
用... 标记AWS Management Console ... 1865
使用AWS IoT Greengrass V2 API 进行标记 ... 1865
在 IAM 策略中使用标签 ... 1866
AWS CloudFormation 资源 ... 1868
AWS IoT Greengrass 和 AWS CloudFormation 模板 ... 1868
ComponentVersion 模板示例 ... 1868
部署模板示例 ... 1869
了解有关 AWS CloudFormation 的更多信息 ... 1870
开源软件 ... 1871
文档历史记录 ... 1872
AWS 词汇表 ... 1904

.... mcmv

什么是 AWS IoT Greengrass？

AWS IoT Greengrass是一款开源物联网 (IoT) 边缘运行时和云服务，可帮助您在设备上构建、部署和管理物联网应用程序。您可以使用AWS IoT Greengrass来构建软件，使您的设备能够根据其生成的数据进行本地操作、基于机器学习模型运行预测以及筛选和聚合设备数据。 AWS IoT Greengrass使您的设备能够在离数据生成地点更近的地方收集和分析数据，对本地事件做出自主反应，并与本地网络上的其他设备进行安全通信。Greengrass设备还可以安全地与物联网通信AWS IoT Core并将物联网数据导出到. AWS Cloud 您可以使用 AWS IoT Greengrass 通过预构建的软件模块（称为组件）来构建边缘应用程序，这些模块可以将您的边缘设备连接到 AWS 服务或第三方服务。您还可以使用 AWS IoT Greengrass Lambda 函数、Docker 容器、本机操作系统进程或您选择的自定义运行时来打包和运行您的软件。

以下示例显示AWS IoT Greengrass设备如何与交互。AWS Cloud



新功能

AWS IoT Greengrass V2引入了新功能和改进。以下内容包括有关版本 2 中提供的新功能的更多信息。

- 新增内容 AWS IoT Greengrass Version 2

对于首次使用的用户 AWS IoT Greengrass

如果您不熟悉AWS IoT Greengrass，我们建议您阅读以下部分：

- AWS IoT Greengrass 的工作原理

接下来，按照入门教程试用的基本功能AWS IoT Greengrass。在本教程中，你将在设备上安装 AWS IoT Greengrass Core 软件，开发 Hello World 组件，然后打包该组件进行部署。

对于的现有用户 AWS IoT Greengrass

对于当前的用户AWS IoT Greengrass V1，我们推荐以下主题来帮助您理解 Greengrass 版本 1 和 Greengrass 版本 2 之间的区别，并学习如何从版本 1 迁移到版本 2：

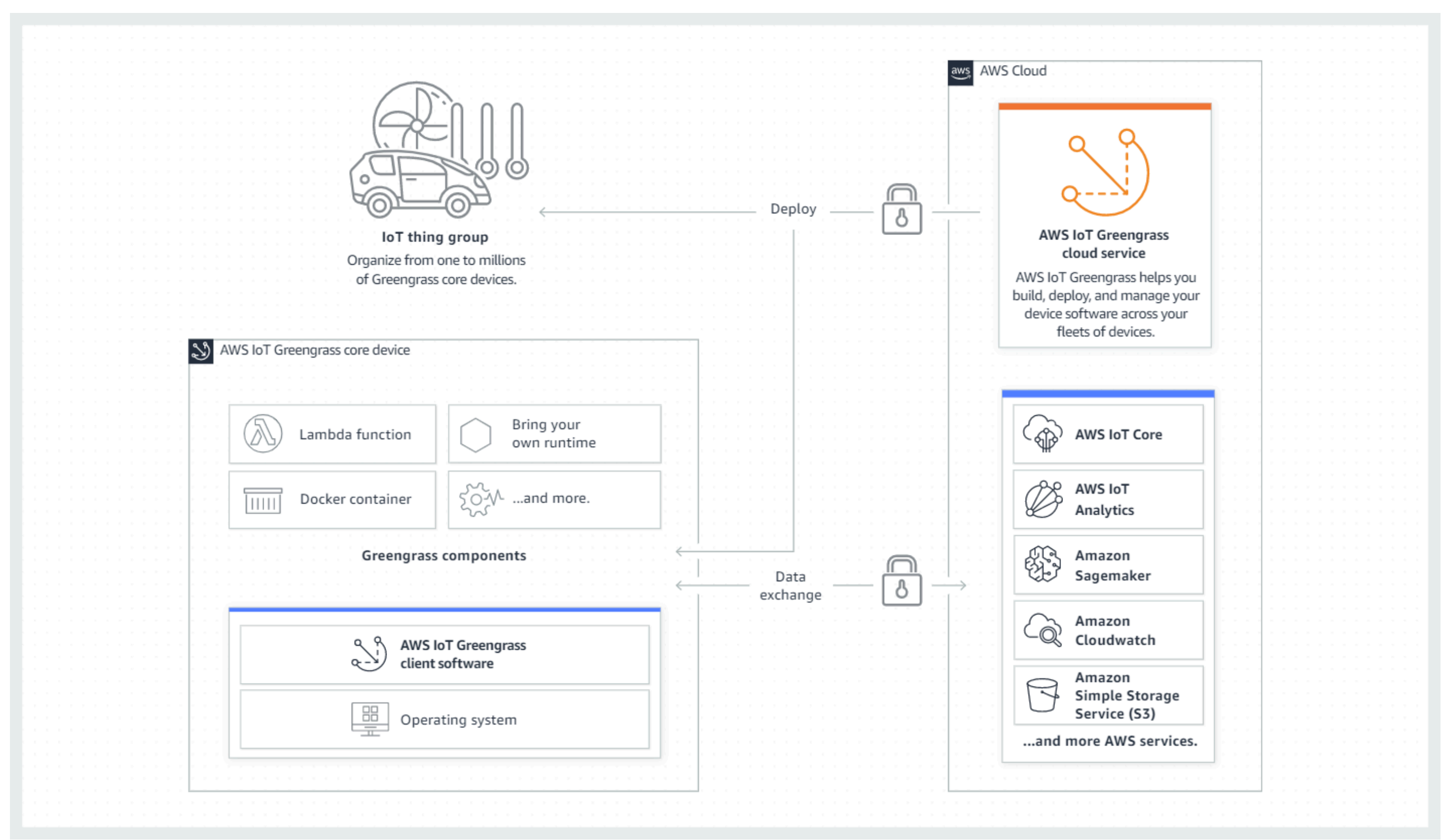
- 从AWS IoT Greengrass版本 1 迁移

AWS IoT Greengrass 的工作原理

AWS IoT Greengrass客户端软件，也称为AWS IoT Greengrass核心软件，可在Windows和基于Linux的发行版（例如Ubuntu或Raspberry Pi操作系统）上运行，适用于采用ARM或x86架构的设备。借AWS IoT Greengrass助，您可以对设备进行编程，使其根据其生成的数据进行本地操作，基于机器学习模型运行预测，以及筛选和聚合设备数据。 AWS IoT Greengrass允许在本地执行您选择的AWS Lambda函数、Docker 容器、本机操作系统进程或自定义运行时。

AWS IoT Greengrass提供称为组件的预建软件模块，可让您轻松扩展边缘设备功能。 AWS IoT Greengrass组件使您能够在边缘连接到AWS服务和第三方应用程序。在开发物联网应用程序之后，AWS IoT Greengrass您可以远程部署、配置和管理现场设备群中的这些应用程序。

以下示例显示AWS IoT Greengrass设备如何与AWS IoT Greengrass云服务以及中的其他AWS服务进行交互。AWS Cloud



AWS IoT Greengrass 的重要概念

以下是理解和使用的基本概念AWS IoT Greengrass：

AWS IoT东西

AWS IoT事物是特定设备或逻辑实体的表示。有关事物的信息存储在AWS IoT注册表中。

Greengrass 核心设备

运行 C AWS IoT Greengrass ore 软件的设备。Greengrass 核心设备是物联网的东西。AWS您可以将多个核心设备添加到AWS IoT事物组中，以创建和管理 Greengrass 核心设备组。有关更多信息，请参阅 设置AWS IoT Greengrass核心设备。

Greengrass 客户端设备

一种通过 MQTT 连接到 Greengrass 核心设备并与之通信的设备。Greengrass 客户端设备就是一回事。AWS IoT核心设备可以处理、筛选和聚合来自与其连接的客户端设备的数据。您可以将核心设备配置为在客户端设备、AWS IoT Core云服务和 Greengrass 组件之间中继 MQTT 消息。有关更多信息，请参阅 与本地物联网设备互动。

客户端设备可以运行 FreeRTOS，也可以使用AWS IoT Device SDK或 Greengrass 发现 API 来获取有关它们可以连接的核心设备的信息。

Greengrass 组件

部署在 Greengrass 核心设备上并在其上运行的软件模块。使用开发和部署的所有软件都建模AWS IoT Greengrass为一个组件。 AWS IoT Greengrass提供了预先构建的公共组件，这些组件提供了可在应用程序中使用的特性和功能。您还可以在本地设备或云端开发自己的自定义组件。开发自定义组件后，您可以使用AWS IoT Greengrass云服务将其部署到单个或多个核心设备。您可以创建自定义组件并将该组件部署到核心设备。当你这样做时，核心设备会下载以下资源来运行该组件：

- 配方：一个 JSON 或 YAML 文件，通过定义组件详细信息、配置和参数来描述软件模块。
- A@@ r tifact：定义将在您的设备上运行的软件的源代码、二进制文件或脚本。您可以从头开始创建工件，也可以使用 Lambda 函数、Docker 容器或自定义运行时创建组件。
- 依赖关系：组件之间的关系，使您可以强制自动更新或重新启动依赖组件。例如，您可以让一个依赖于加密组件的安全消息处理组件。这样可以确保对加密组件的任何更新都会自动更新并重新启动消息处理组件。

有关更多信息，请参阅 AWS-提供的组件和 开发AWS IoT Greengrass组件。

部署

发送组件并将所需组件配置应用于目标设备的过程，目标设备可以是单个 Greengrass 核心设备或一组 Greengrass 核心设备。部署会自动将任何更新的组件配置应用于目标，并包括定义为依赖关系的任何其他组件。您也可以克隆现有部署以创建使用相同组件但部署到不同目标的新部署。部署是连续的，这意味着您对部署的组件或组件配置所做的任何更新都会自动发送到所有目标目标。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。

AWS IoT Greengrass核心软件

您在核心设备上安装的所有AWS IoT Greengrass软件的集合。 AWS IoT Greengrass核心软件包括以下内容：

- Nucleus：这个必需的组件提供了AWS IoT Greengrass核心软件的最低功能。nucleus 管理其他组件的部署、编排和生命周期管理。它还便于在单个设备上进行本地AWS IoT Greengrass组件之间的通信。有关更多信息，请参阅 Greengrass 核。
- 可选组件：这些可配置组件由您的边缘设备提供，AWS IoT Greengrass并在您的边缘设备上启用其他功能。根据您的要求，您可以选择要部署到设备上的可选组件，例如数据流、本地机器学习推理或本地命令行界面。有关更多信息，请参阅 AWS-提供的组件。

您可以通过在设备上部署新版本的组件来升级 C AWS IoT Greengrass ore 软件。

AWS IoT Greengrass 的特征

AWS IoT Greengrass Version 2由以下元素组成：

- 软件发行版
  - Greengrass nucleus 组件，这是核心软件的最低安装量。AWS IoT Greengrass该组件管理 Greengrass 组件的部署、编排和生命周期管理。
  - 其他AWS可选提供的与服务、协议和软件集成的组件。
  - Greengrass 开发工具，可用于创建、测试、构建、发布和部署自定义 Greengrass 组件。
  - AWS IoT Device SDK，其中包含用于自定义 Greengrass 组件的进程间通信 (IPC) 库和用于客户端设备的 Greengrass 发现库。
  - Stream Manager SDK，可用于管理核心设备上的数据流。
- 云服务
  - AWS IoT Greengrass V2 API
  - AWS IoT Greengrass V2 控制台

AWS IoT Greengrass Core 软件

您可以使用在边缘设备上运行的 AWS IoT Greengrass Core 软件来执行以下操作：

- 在本地设备上处理数据流，并自动导出到AWS云端。有关更多信息，请参阅 管理 Greengrass 核心设备上的数据流。
- Support 支持AWS IoT和组件之间的 MQTT 消息传递。有关更多信息，请参阅 发布/订阅 AWS IoT Core MQTT 消息。
- 与通过 MQTT 连接和通信的本地设备进行交互。有关更多信息，请参阅 与本地物联网设备互动。
- Support 支持组件之间的本地发布和订阅消息。有关更多信息，请参阅 发布/订阅本地消息。
- 部署和调用组件和 Lambda 函数。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。
- 管理组件生命周期，例如支持安装和运行脚本。有关更多信息，请参阅 AWS IoT Greengrass组件配方参考。
- 对AWS IoT Greengrass核心软件和自定义组件执行安全 over-the-air (OTA) 软件更新。有关更多信息，请参阅 更新AWS IoT Greengrass核心软件 (OTA)和 将AWS IoT Greengrass组件部署到设备。
- 为本地机密提供安全、加密的存储以及组件的受控访问。有关更多信息，请参阅 秘密经理。
- 通过设备身份验证和授权，保护设备与AWS云端之间的连接。有关更多信息，请参阅 AWS IoT Greengrass 的设备身份验证和授权。

您可以AWS IoT Greengrass通过 API 配置和管理 Greengrass 核心设备，在那里您可以创建持续的软件部署。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。

某些功能仅在某些平台上受支持。有关更多信息，请参阅 按操作系统划分的 Greengrass 功能兼容性。

有关支持的平台、要求和下载的更多信息，请参阅设置AWS IoT Greengrass核心设备。

下载此软件即表示您同意Greengrass Core 软件许可协议。

按操作系统划分的 Greengrass 功能兼容性

AWS IoT Greengrass 支持运行各种操作系统的设备。某些功能仅在某些操作系统上受支持。使用下表了解每种支持的操作系统都有哪些功能可用。有关支持的操作系统、要求以及如何设置 Greengrass 核心设备的更多信息，请参阅。设置AWS IoT Greengrass核心设备

消息收发

功能	Linux	Windows
在 AWS IoT 和组件之间交换 MQTT 消息	是	是
在组件之间交换本地发布/订阅消息	是	是
通过 MQTT 与本地物联网设备互动	是	是
使用 modbus-RTU 组件与本地 Modbus-RTU 设备进行交互	是	否

安全性

功能	Linux	Windows
通过设备身份验证和授权实现安全连接	是	是
部署和访问来自的安全、加密的机密 AWS Secrets Manager	是	是
使用硬件安全模块 (HSM) 安全存储设备的私钥和证书	是	否
使用以下方法审核核心设备 AWS IoT Device Defender	是	是
使用 AWS 凭证与 AWS 服务进行交互	是	是

安装

功能	Linux	Windows
AWS IoT Greengrass 使用自动配置进行安装	是	是

功能	Linux	Windows
AWS IoT Greengrass 使用手动配置进行安装	是	是
AWS IoT Greengrass 使用 AWS IoT 队列配置进行安装	是	是
AWS IoT Greengrass 使用自定义配置插件进行安装	是	是
使用预先构建的 Docker 镜像 AWS IoT Greengrass 在 Docker 容器中运行	是	否

远程维护和更新

功能	Linux	Windows
执行安全 over-the-air（OTA）软件更新	是	是
使用管理核心设备 AWS Systems Manager	是	否

功能	Linux	Windows
通过 AWS IoT 安全隧道连接至核心设备	是	否

机器学习

功能	Linux	Windows
使用 Amazon SageMaker Edge Manager 执行机器学习推理	是	是
使用 Amazon Lookout for Vision 进行机器学习推理	是	否
使用 DLR 执行机器学习推理	是	是
使用执行机器学习推理 TensorFlow	是	是

组件特性

功能	Linux	Windows
部署和调用 Lambda 函数	是	否
在组件中运行 Docker 容器	是	是
使用流管理器处理和导出大容量数据流	是	是
使用生命周期脚本管理组件生命周期	是	是
与设备阴影互动	是	是
将日志上传到 Amazon CloudWatch 日志	是	是

功能	Linux	Windows
使用 CloudWatch 指标组件将数据上传到 Amazon CloudWatch 指标	是	是
使用 Amazon SNS 组件向亚马逊简单通知服务发布消息	是	否
使用流管理器将数据发布到 Amazon Data Firehose 传送流	是	是
使用 Firehose 组件将数据发布到亚马逊数据 Firehose 传送流	是	否
收集实时系统遥测指标并采取行动	是	是
为组件进程配置系统资源限制	是	否
暂停和恢复组件进程	是	否

功能	Linux	Windows
与 AWS IoT SiteWise 使用 AWS IoT SiteWise 组件进行集成	是	是
使用 Kinesis Video Streams 组件的边缘连接器将视频流发布到亚马逊 Kinesis Video Streams	是	否

组件开发

功能	Linux	Windows
在核心设备上本地开发组件	是	是
使用 AWS IoT Greengrass CLI 与核心设备交互	是	是
使用本地调试控制台与核心设备交互	是	是
在自定义组件中使用 AWS IoT Device SDK 适用于 Python 的	是	是

功能	Linux	Windows
在自定义组件中 AWS IoT Device SDK 使用 for C++	是	是
在自定义组件中使用 AWS IoT Device SDK 适用于 Java 的	是	是

设备认证

功能	Linux	Windows
AWS IoT Device Tester 用于验证 AWS IoT Greengrass V2 物联网设备	是	是

新增内容 AWS IoT Greengrass Version 2

AWS IoT Greengrass Version 2 是一个主要版本 AWS IoT Greengrass ，它引入了以下功能：

- 发布商支持的组件 — AWS IoT Greengrass 现在提供发布商支持的组件。这些组件由第三方供应商开发、提供和服务。有关更多信息，请参阅 发布商支持的组件。
- 在 VPC 中@@ 操作 Greengrass 设备 — 现在可以在 VPC 中操作 Greengrass 核心设备。这使您无需公共互联网访问即可在 VPC 中执行部署。有关更多信息，请参阅 在 VPC 中操作AWS IoT Greengrass核心设备。
- Greengrass 测试框架 (GTF) — GTF for 现已推出。 AWS IoT Greengrass Version 2 GTF 是支持 end-to-end 自动化的构建块的集合。它使 AWS IoT Greengrass Version 2 内部客户能够使用与服务团队相同的测试框架来验证软件更改、自动接受和质量保证。欲了解更多信息，请参阅 Github 上的 Greengrass 测试框架。
- PSA 认证 — n AWS IoT Greengrass ucleus 版本 2.7.0 及更高版本现已通过平台安全架构 (PSA) 认证。有关更多信息，请参阅AWS IoT Greengrass 是否经过 PSA 认证。

AWS IoT Greengrass 发行说明提供了有关 AWS IoT Greengrass 版本的详细信息，包括新功能、更新和改进以及常规修复。 AWS IoT Greengrass 有以下类型的版本：

- 的新功能发布 AWS IoT Greengrass
- AWS IoT Greengrass 核心软件更新

本节包含所有 AWS IoT Greengrass V2 发行说明，最新版本在前，并包括主要的功能更改和重要的错误修复。有关其他次要修复的信息，请参阅上的 aws-greengrass 组织。 GitHub

发布说明

- 发布： AWS IoT Greengrass 酷睿 v2.12.4 软件更新将于 2024 年 4 月 2 日发布
- 发布： AWS IoT Greengrass 酷睿 v2.12.3 软件更新将于 2024 年 3 月 27 日发布
- 发布： AWS IoT Greengrass 酷睿 v2.12.2 软件更新将于 2024 年 2 月 15 日发布
- 发布：AWS IoT Greengrass酷睿 v2.12.1 软件更新将于 2023 年 12 月 8 日发布
- 发布：AWS IoT Greengrass酷睿 v2.12.0 软件更新将于 2023 年 11 月 7 日发布
- 发布：AWS IoT Greengrass酷睿 v2.11.3 软件更新将于 2023 年 10 月 18 日发布
- 发布：AWS IoT Greengrass酷睿 v2.11.2 软件更新将于 2023 年 8 月 9 日发布
- 发布：AWS IoT Greengrass酷睿 v2.11.1 软件更新将于 2023 年 7 月 21 日发布

- 发布：AWS IoT Greengrass酷睿 v2.11.0 软件更新将于 2023 年 6 月 28 日发布
- 发布：AWS IoT Greengrass酷睿 v2.10.3 软件更新将于 2023 年 6 月 21 日发布
- 发布：AWS IoT Greengrass酷睿 v2.10.2 软件更新将于 2023 年 6 月 5 日发布
- 发布：AWS IoT Greengrass酷睿 v2.10.1 软件更新将于 2023 年 5 月 11 日发布
- 发布：AWS IoT Greengrass酷睿 v2.10.0 软件更新将于 2023 年 5 月 9 日发布
- 发布：AWS IoT Greengrass酷睿 v2.9.6 软件更新将于 2023 年 4 月 20 日发布
- 发布：AWS IoT Greengrass酷睿 v2.9.5 软件更新将于 2023 年 3 月 30 日发布
- 发布：AWS IoT Greengrass酷睿 v2.9.4 软件更新将于 2023 年 2 月 24 日发布
- 发布：AWS IoT Greengrass酷睿 v2.9.3 软件将于 2023 年 2 月 1 日更新
- 发布：AWS IoT Greengrass酷睿 v2.9.2 软件更新将于 2022 年 12 月 22 日发布
- 发布：AWS IoT Greengrass酷睿 v2.9.1 软件更新将于 2022 年 11 月 18 日发布
- 发布：AWS IoT Greengrass酷睿 v2.9.0 软件更新将于 2022 年 11 月 15 日发布
- 发布：AWS IoT Greengrass酷睿 v2.8.1 软件更新将于 2022 年 10 月 13 日发布
- 发布：AWS IoT Greengrass酷睿 v2.8.0 软件更新将于 2022 年 10 月 7 日发布
- 发布：AWS IoT Greengrass酷睿 v2.7.0 软件更新将于 2022 年 7 月 28 日发布
- 发布：AWS IoT Greengrass酷睿 v2.6.0 软件更新将于 2022 年 6 月 27 日发布
- 发布：AWS IoT Greengrass酷睿 v2.5.6 软件更新将于 2022 年 5 月 31 日发布
- 发布：AWS IoT Greengrass酷睿 v2.5.5 软件更新将于 2022 年 4 月 6 日发布
- 发布：AWS IoT Greengrass酷睿 v2.5.4 软件更新将于 2022 年 3 月 23 日发布
- 发布：AWS IoT Greengrass酷睿 v2.5.3 软件更新将于 2022 年 1 月 6 日发布
- 发布：AWS IoT Greengrass酷睿 v2.5.2 软件更新将于 2021 年 12 月 3 日发布
- 发布：AWS IoT Greengrass酷睿 v2.5.1 软件更新将于 2021 年 11 月 23 日发布
- 发布：AWS IoT Greengrass酷睿 v2.5.0 软件更新将于 2021 年 11 月 12 日发布
- 发布：AWS IoT Greengrass酷睿 v2.4.0 软件更新将于 2021 年 8 月 3 日发布
- 发布：AWS IoT Greengrass酷睿 v2.3.0 软件更新将于 2021 年 6 月 29 日发布
- 发布：AWS IoT Greengrass酷睿 v2.2.0 软件更新将于 2021 年 6 月 18 日发布
- 发布：AWS IoT Greengrass酷睿 v2.1.0 软件更新将于 2021 年 4 月 26 日发布
- 发布：AWS IoT Greengrass酷睿 v2.0.5 软件将于 2021 年 3 月 9 日更新
- 发布：AWS IoT Greengrass酷睿 v2.0.4 软件将于 2021 年 2 月 4 日更新

发布： AWS IoT Greengrass 酷睿 v2.12.4 软件更新将于 2024 年 4 月 2 日发布

此版本提供了 Greengrass nucleus 组件的 2.12.4 版本，并更新了提供的组件。 AWS

发布日期：2024 年 4 月 2 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的 AWS 包含新功能和更新功能的组件。

> **Important**
>
> 部署组件时， AWS IoT Greengrass 会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则 AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.12.4 版本已上市。 错误修复和改进 • 修复了在某些 Linux 设备上启动时核心进入死锁状态的问题。

发布： AWS IoT Greengrass 酷睿 v2.12.3 软件更新将于 2024 年 3 月 27 日发布

此版本提供了 Greengrass nucleus 组件的 2.12.3 版本，并更新了提供的组件。 AWS

发布日期：2024 年 3 月 27 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的组件 AWS ，其中包括新的和更新的功能。

Important

部署组件时， AWS IoT Greengrass 会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则 AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.12.3 版本已上市。 错误修复和改进 • 修复了在原子核重启后和组件恢复期间，原子核无法报告正确的组件状态的问题。 • 常规错误修复和性能改进。
影子经理	影子管理器组件的 2.3.7 版本可用。 错误修复和改进 修复了影子管理器在影子管理器同步期间定期记录`NullPointerException` 错误的问题。
舰队配置	AWS IoT 舰队配置插件版本 1.2.1 现已推出。

组件	详细信息
	错误修复和改进 修复了 Greengrass nucleus 启动期间舰队配置插件处于离线状态的问题。队列配置插件现在可以无限期地重试 MQTT 连接调用。
IP 探测器	磁盘后台处理程序组件的 2.1.9 版本可用。 错误修复和改进 将获取的 IP 步骤调整为仅发送调试日志级别的日志。
Moquette MQTT 3.1.1 代理组件	M oquette MQTT 3.1.1 经纪商组件的 2.3. 6 版现已推出。 错误修复和改进 常规错误修复和性能改进。
Lambda 管理器	Lambda 管理器组件已推出 2.3.3 版。 错误修复和改进 常规错误修复和性能改进。
本地调试控制台	本地调试控制台组件的 2.4.2 版本可用。 错误修复和改进 常规错误修复和性能改进。

发布： AWS IoT Greengrass 酷睿 v2.12.2 软件更新将于 2024 年 2 月 15 日发布

此版本提供了 Greengrass nucleus 组件的 2.12.2 版本，并更新了提供的组件。 AWS

发布日期：2024 年 2 月 15 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的组件 AWS ，其中包括新的和更新的功能。

Important

部署组件时， AWS IoT Greengrass 会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则 AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.12.2 版本已上市。 错误修复和改进 • 修复了旧日志未正确清理的问题。 • 常规错误修复和性能改进。
影子经理	影子管理器组件的 2.3.6 版本可用。 错误修复和改进 修复了设备离线时通过 AWS Cloud 更新删除的阴影属性在重新连接后继续存在于本地阴影中的问题。
Lambda 启动器	lambda 启动器组件的 2.0.13 版本现已推出。 错误修复和改进 常规错误修复和性能改进。
磁盘后台处理程序	磁盘后台处理程序组件的 1.0.3 版本可用。

组件	详细信息
	错误修复和改进 通过重复使用数据库连接来提高性能。

发布：AWS IoT Greengrass酷睿 v2.12.1 软件更新将于 2023 年 12 月 8 日发布

此版本提供了 Greengrass nucleus 组件的 2.12.1 版本，并更新了提供的组件。AWS

发布日期：2023 年 12 月 8 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的组件AWS，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关AWS IoT Greengrass核心软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.12.1 版本已上市。

组件	详细信息
	错误修复和改进 • 修复了核心可能会重复订阅部署主题的 MQTT，从而导致额外的日志记录和 MQTT 发布的问题。
客户端设备身份验证	客户端设备身份验证组件版本 2.4.5 已推出。 新功能 添加了对使用参数选择事物名称的通配符前缀的支持。selectionRule 错误修复和改进 修复了在某些情况下无法使用新的连接信息更新证书的问题。
磁盘后台处理程序	磁盘后台处理程序组件的 1.0.2 版本可用。 错误修复和改进 修复了在某些情况下无法保留 MQTT 消息格式字段的问题。
MQTT 网桥	磁盘后台处理程序组件的 2.3.1 版本可用。 错误修复和改进 修复了本地 MQTT 客户端进入断开连接循环的问题。
直播管理器	直播管理器组件已推出 2.1.12 版。 错误修复和改进 更新凭据的使用顺序，以便服务请求的首选 Greengrass 凭据。AWS

发布：AWS IoT Greengrass酷睿 v2.12.0 软件更新将于 2023 年 11 月 7 日发布

此版本提供了 Greengrass nucleus 组件的 2.12.0 版本，并更新了提供的组件。AWS

发布日期：2023 年 11 月 7 日

发布亮点

- 回滚时的 Bootstrap — AWS IoT Greengrass 现在提供了一个名为的 Greengrass 核心配置参数。BootstrapOnRollback此功能使您能够将引导生命周期步骤作为回滚部署的一部分来运行。

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的AWS包含新功能和更新功能的组件。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.12.0 版本已上市。 新功能 • 使您能够将引导生命周期步骤作为回滚部署的一部分运行。

发布：AWS IoT Greengrass酷睿 v2.11.3 软件更新将于 2023 年 10 月 18 日发布

此版本提供了 Greengrass nucleus 组件的 2.11.3 版本。

发布日期：2023 年 10 月 18 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的组件AWS，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.11.3 版本已上市。 错误修复和改进 • 修复了 nucleus 中的一个问题，即当组件的依赖关系失败时，它可能会不正确地启动组件。 新功能 • 添加可配置的 s3 端点类型。
Lambda 管理器	Lambda 管理器组件已推出 2.3.1 版。 错误修复和改进 • 调整某些错误的日志级别。
本地调试控制台	Lambda 管理器组件已推出 2.4.0 版。 新功能 • 添加直播管理器调试控制台。

组件	详细信息
日志管理器	日志管理器组件的 2.3.6 版本可用。 错误修复和改进 • 调整某些错误的日志级别。
影子经理	影子管理器组件的 2.3.4 版本现已推出。 错误修复和改进 • 增加了对空和空阴影状态文档的支持。

发布：AWS IoT Greengrass酷睿 v2.11.2 软件更新将于 2023 年 8 月 9 日发布

此版本提供了 Greengrass nucleus 组件的 2.11.2 版。

发布日期：2023 年 8 月 9 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的组件AWS，其中包括新的和更新的功能。

> **Important**
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.11.2 版本已上市。 错误修复和改进 • 修复了 nucleus MQTT 5 客户端中的一个问题，即在使用大量（> 50）订阅时，它可能会显示为离线。 • 为 docker 拨号 TCP 失败添加了重试功能。

发布：AWS IoT Greengrass酷睿 v2.11.1 软件更新将于 2023 年 7 月 21 日发布

此版本提供了 Greengrass nucleus 组件的 2.11.1 版本。

发布日期：2023 年 7 月 21 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的AWS包含新功能和更新的功能的组件。

> **Important**
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.11.1 版本已上市。 错误修复和改进 • 修复了在引导任务失败且部署元数据文件损坏时，nucleus 无法启动的问题。 • 修复了部署状态更新中未报告按需 Lambda 组件的问题。 • 增加了对重复授权策略 ID 的支持。
Lambda 管理器	Lambda 管理器已推出 2.2.11 版。 错误修复和改进 • 修复了 Lambda LegacySubscriptionRouter 配置更改时配置不会更新的问题。

发布：AWS IoT Greengrass酷睿 v2.11.0 软件更新将于 2023 年 6 月 28 日发布

此版本提供了 Greengrass nucleus 组件的 2.11.0 版本。

发布日期：2023 年 6 月 28 日

发布亮点

- 永久磁盘假脱机程序 — AWS IoT Greengrass 现在为从 Greengrass 核心设备后台处理的消息提供永久后台处理程序实现。AWS IoT Core此组件会将这些出站消息存储在磁盘上。有关更多信息，请参阅 磁盘后台处理程序。
- 本地部署改进-您现在可以取消本地部署、设置部署失败处理策略以及获取详细的部署状态。
- 日志速度改进-日志管理器组件的日志上传速度已提高。

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的组件AWS，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.11.0 版本已上市。 新功能 • 允许您取消本地部署。 • 使您能够为本地部署配置故障处理策略。 • 添加了对磁盘后台处理程序插件的支持。
Greengrass CLI	Greengrass CLI 的 2.11.0 版本现已推出。 新功能 • 允许您取消本地部署。 • 使您能够为本地部署配置故障处理策略。 • 改进了详细的部署状态报告。
磁盘后台处理程序	磁盘后台处理程序组件的 1.0.0 版本可用。 • 磁盘假脱机组件可永久存储从 Greengrass 核心设备发送到的消息。AWS IoT Core
日志管理器	日志管理器组件已推出 2.3.5 版。

组件	详细信息
	改进 提高了日志上传速度。

发布：AWS IoT Greengrass酷睿 v2.10.3 软件更新将于 2023 年 6 月 21 日发布

此版本提供了 Greengrass nucleus 组件的 2.10.3 版本。

发布日期：2023 年 6 月 21 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的组件AWS，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.10.3 版本已上市。

组件	详细信息
	错误修复和改进 • 修复了使用 PKCS #11 提供程序时 Greengrass 不订阅部署通知的问题。

发布：AWS IoT Greengrass酷睿 v2.10.2 软件更新将于 2023 年 6 月 5 日发布

此版本提供了 Greengrass nucleus 组件的 2.10.2 版本。

发布日期：2023 年 6 月 5 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的AWS包含新功能和更新的功能的组件。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.10.2 版本已上市。 错误修复和改进 • 允许对组件生命周期进行不区分大小写的解析。

组件	详细信息
	• 修复了未正确重新创建环境 PATH 变量的问题。 • 修复了组件的代理 URI 编码，包括带有特殊字符的用户名的流管理器。
客户端设备身份验证	客户端设备身份验证组件的 2.4.2 版本可用。 新功能 添加新的startupTimeoutSeconds 配置选项。
Lambda 管理器	Lambda 管理器组件已推出 2.2.9 版。 错误修复和改进 修复了由于时钟偏差导致端口号损坏的问题。
日志管理器	日志管理器组件的 2.3.4 版本可用。 错误修复和改进 • 增加了对将periodicUploadIntervalSec 参数设置为小数值的支持。最小值为 1 微秒。 • 修复了日志管理器不遵守 CloudWatchputLogEvents 限制的问题。
MQTT 3.1 交易商 (Moquette)	MQTT 3. 1 经纪商 (Moquette) 组件的 2.3. 3 版现已推出。 新功能 添加新的startupTimeoutSeconds 配置选项。
MQTT 桥接器	MQTT 桥接组件的 2.2.6 版本现已推出。 新功能 添加新的startupTimeoutSeconds 配置选项。
直播管理器	直播管理器组件已推出 2.1.7 版。 错误修复和改进 修复了直播管理器无法正确读取代理配置的问题。

发布：AWS IoT Greengrass酷睿 v2.10.1 软件更新将于 2023 年 5 月 11 日发布

此版本提供了 Greengrass nucleus 组件的 2.10.1 版本。

发布日期：2023 年 5 月 11 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的AWS包含新功能和更新功能的组件。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关AWS IoT Greengrass核心软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.10.1 版本已上市。 错误修复和改进 • 修复了可能导致某些 ARMv8 处理器（包括 Jetson Nano）在启动时崩溃的问题。 • Greengrass 不再关闭组件的标准，这会将行为恢复到 2.10.0 之前的行为
直播管理器	新的直播管理器已推出 2.1.6 版。

组件	详细信息
	错误修复和改进 修复了可能导致某些 ARMv8 处理器（包括 Jetson Nano）在启动时崩溃的问题。

发布：AWS IoT Greengrass酷睿 v2.10.0 软件更新将于 2023 年 5 月 9 日发布

此版本提供了 Greengrass nucleus 组件的 2.10.0 版本，并更新了提供的组件。AWS

发布日期：2023 年 5 月 9 日

发布亮点

- MQTT5 支持 — AWS IoT Greengrass 现在支持AWS IoT Core使用 MQTT5 发送和接收消息。有关更多信息，请参阅发布 AWS IoT Core MQTT 消息。

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的组件AWS，其中包括新的和更新的功能。

> **Important**
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

<table>
<tr><th>组件</th><th>详细信息</th></tr>
<tr><td>Greengrass 核</td><td>Greengrass nucleus 的 2.10.0 版本已上市。

新功能
添加对空正则表达式的<code>interpolateComponentConfiguration</code> 支持。Greengrass 现在可以从根配置对象中进行插值。
添加了对 MQTT5 的支持。
添加了无需扫描即可快速加载插件组件的机制。
让 Greengrass 能够通过删除未使用的 Docker 镜像来节省磁盘空间。
错误修复和改进
修复了回滚会使部署中的某些配置值保持不变的问题。
修复了 Greengrass nucleus 验证自定义非凭据和数据端点中的域序列的问题。AWS AWS
更新多组依赖关系解析以通过AWS Cloud协商重新解析所有组依赖关系，而不是锁定到活动版本。此更新还删除了部署错误代码<code>INSTALLED_COMPONENT_NOT_FOUND</code> 。
更新 Greengrass 核心，使其在本地已存在 Docker 镜像时跳过下载这些镜像。
更新 Greengrass 核心，以便在超时到期之前重新启动组件安装步骤。
其他小修复和改进。</td></tr>
<tr><td>影子经理</td><td>新的影子管理器已推出 2.3.2 版。

错误修复和改进

修复了本地影子数据库损坏时影子管理器进入<code>BROKEN</code>状态的问题。</td></tr>
</table>

发布：AWS IoT Greengrass酷睿 v2.9.6 软件更新将于 2023 年 4 月 20 日发布

此版本提供了 Greengrass nucleus 组件的 2.9.6 版本。

发布日期：2023 年 4 月 20 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的组件AWS，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.9.6 版本已上市。 错误修复和改进 • 修复了 Greengrass 部署失败并显示错误 LAUNCH_DIRECTORY_CORTED 以及随后的设备重启无法启动 Greengrass 的问题。当您在部署需要 Greengrass 重启的多个事物组之间移动 Greengrass 设备时，可能会发生此错误。

发布：AWS IoT Greengrass酷睿 v2.9.5 软件更新将于 2023 年 3 月 30 日发布

此版本提供了 Greengrass nucleus 组件的 2.9.5 版。

发布日期：2023 年 3 月 30 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的AWS包含新功能和更新功能的组件。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.9.5 版本已上市。 新功能 • 增加了对 Greengrass nucleus 软件签名验证的支持。 错误修复和改进 • 修复了本地配方元数据区域与 Greengrass nucleus 启动区域不匹配时部署失败的问题。当这种情况发生时，Greengrass 核现在会与云重新协商。 • 修复了 MQTT 消息后台处理程序已满且从不删除消息的问题。 • 其他小修复和改进。

发布：AWS IoT Greengrass酷睿 v2.9.4 软件更新将于 2023 年 2 月 24 日发布

此版本提供了 Greengrass nucleus 组件的 2.9.4 版本。

发布日期：2023 年 2 月 24 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的AWS包含新功能和更新功能的组件。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.9.4 版本已上市。 错误修复和改进 • 在丢弃 QOS 0 消息之前检查是否存在空消息。 • 如果作业状态详细信息值超过 1024 个字符的限制，则将其截断。 • 更新 Windows 的引导脚本以正确读取 Greengrass 根路径（如果该路径包含空格）。 • 更新订阅，AWS IoT Core以便在未发送订阅响应时丢弃客户端消息。 • 确保 nucleus 在主配置文件损坏或丢失时从备份文件加载其配置。

发布：AWS IoT Greengrass酷睿 v2.9.3 软件将于 2023 年 2 月 1 日更新

此版本提供了 Greengrass nucleus 组件的 2.9.3 版本。

发布日期：2023 年 2 月 1 日

发布详情

- 公共组件更新

公共组件更新

下表列出了由提供的AWS包含新功能和更新功能的组件。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.9.3 版本已上市。 错误修复和改进 • 确保 MQTT 客户端 ID 不重复。 • 增加了更强大的文件读取和写入功能，以避免损坏并从损坏中恢复。 • 重试 docker 镜像拉取特定的网络相关错误。 • 添加了 MQTT 连接`noProxyAddresses` 选项。

发布：AWS IoT Greengrass酷睿 v2.9.2 软件更新将于 2022 年 12 月 22 日发布

此版本提供了 Greengrass nucleus 组件的 2.9.2 版本。

发布日期：2022 年 12 月 22 日

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.9.2 版本已上市。 错误修复和改进 • 修复了配置`interpolateComponentConfiguration` 不适用于正在进行的部署的问题。 • 使用 OSHI 列出所有子进程。

发布：AWS IoT Greengrass酷睿 v2.9.1 软件更新将于 2022 年 11 月 18 日发布

此版本提供了 Greengrass nucleus 组件的 2.9.1 版，并更新了提供的组件。AWS

发布日期：2022 年 11 月 18 日

发布亮点

- 日志管理器-日志管理器现在可以处理和直接上传活动日志文件，而不必等待轮换新文件。这一改进显著减少了日志延迟。有关更多信息，请参阅日志管理器

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

> Important
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备，或者更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.9.1 版本已上市。 错误修复和改进 • 添加了在部署移除插件组件时 Greengrass 会重新启动的修复程序。

<table>
<tr><th>组件</th><th>详细信息</th></tr>
<tr><td>日志管理器</td><td>新的日志管理器已推出 2.3.0 版。

Note
当你升级到日志管理器 2.3.0 时，我们建议你升级到 Greengrass nucleus 2.9.1。

新功能
• 通过处理和直接上传活动日志文件而不是等待轮换新文件来减少日志延迟。
错误修复和改进
• 改进了在轮换具有唯一名称的文件时对日志轮换的支持。
• 其他小修复和改进。</td></tr>
</table>

发布：AWS IoT Greengrass酷睿 v2.9.0 软件更新将于 2022 年 11 月 15 日发布

此版本提供了 Greengrass nucleus 组件的 2.9.0 版本，并更新了提供的组件。AWS

发布日期：2022 年 11 月 15 日

发布亮点

- 离线身份验证-AWS IoT Greengrass 现在支持离线身份验证。您可以配置AWS IoT Greengrass核心设备，使客户端设备可以连接到核心设备，即使核心设备未连接到云端。有关更多信息，请参阅离线身份验证。
- 子部署-您现在可以创建子部署。您可以使用子部署来解决不成功的部署。每个子部署都可以在较小的设备子集上测试未成功部署的不同配置。有关更多信息，请参阅创建子部署。

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.9.0 版本已上市。 新功能 • 添加了创建子部署的功能，这些子部署可以重试使用较小的设备子集进行部署。此功能提供了一种更有效的方法来测试和解决不成功的部署。 错误修复和改进 • 改进了对没有useraddgroupadd、和的系统的支持usermod。 • 其他小修复和改进。
客户端设备身份验证	客户端设备身份验证组件已推出 2.3.0 版。 新功能 • 增加了对客户端设备离线身份验证的支持。借助此功能，当核心设备未连接到互联网时，客户端设备可以继续连接到核心设备。 • 增加了对客户提供的证书颁发机构 (CA) 的支持。您的核心设备使用客户提供的 CA 作为根证书来生成 MQTT 代理证书。
MQTT 5 经纪商 (EMQX)	该MQTT 5 经纪商 (EMQX)组件的 1.2.0 版本可用。

组件	详细信息
	新功能 添加对证书链的支持。
莫奎特 MQTT 经纪商	新的 M oquette MQTT 经纪商组件已推出 2.3.0 版。 新功能 添加对证书链的支持。
秘密经理	新的密钥管理器版本2.1.4现已推出。 错误修复和改进 修复了部署秘密管理器和 Greengrass 核心重启时缓存的机密被移除的问题。
直播管理器	新的直播管理器已推出 2.1.2 版。 错误修复和改进 修复了 Windows 操作系统上使用非英语语言的问题。

发布：AWS IoT Greengrass酷睿 v2.8.1 软件更新将于 2022 年 10 月 13 日发布

此版本提供了 Greengrass nucleus 组件的 2.8.1 版本。

发布日期：2022 年 10 月 13 日

Note

如果你使用的是 Greengrass nucleus 版本 2.8.0，我们强烈建议你升级到 Greengrass nucleus 版本 2.8.1。

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.8.1 版本已上市。 错误修复和改进 • 修复了 Greengrass API 错误导致部署错误代码未正确生成的问题。 • 修复了在部署期间组件达到某个状态时，舰队状态更新发送的ERRORED信息不准确的问题。 • 修复了 Greengrass 现有订阅超过 50 个时部署无法完成的问题。

发布：AWS IoT Greengrass酷睿 v2.8.0 软件更新将于 2022 年 10 月 7 日发布

此版本提供了 Greengrass nucleus 组件的 2.8.0 版和 MQTT 5 经纪商 (EMQX) 组件的 1.1.0 版。

发布日期：2022 年 10 月 7 日

发布亮点

- 部署错误代码 — Greengrass nucleus 现在会报告部署运行状况响应，其中包括无法完成组件部署时的详细错误代码。有关更多信息，请参阅 详细的部署错误代码。

- 组件错误状态 — Greengrass nucleus 现在会报告组件健康状态响应，其中包括组件进入或状态时的详细错误状态。BROKEN ERRORED有关更多信息，请参阅 详细的组件状态码。

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.8.0 版本已上市。 新功能 • 在将组件部署到核心设备时出现问题时，更新 Greengrass nucleus 以报告部署运行状况响应，其中包括详细的错误代码。有关更多信息，请参阅详细的部署错误代码。 • 更新 Greengrass nucleus 以报告组件健康状态响应，其中包括组件进入或状态时的详细错误代码。BROKEN ERRORED有关更多信息，请参阅 详细的组件状态码。 • 扩展状态消息字段以改善设备的云可用性信息。 • 提高了舰队状态服务的稳健性。

组件	详细信息
	错误修复和改进 • 允许损坏的组件在配置更改时重新安装。 • 修复了在 bootstrap 部署期间重启 nucleus 会导致部署失败的问题。 • 修复了 Windows 中根路径包含空格时安装失败的问题。 • 修复了在部署期间关闭的组件使用新版本的关闭脚本的问题。 • 各种关机改进。 • 其他小修复和改进。
MQTT 5 经纪商 (EMQX)	该MQTT 5 经纪商 (EMQX)组件的 1.1.0 版本可用。 新功能 • 增加了对 EMQX 配置的支持，包括代理选项和插件。 错误修复和改进 • 将 EMQX 更新至版本 4.4.9。

发布：AWS IoT Greengrass酷睿 v2.7.0 软件更新将于 2022 年 7 月 28 日发布

此版本提供了 Greengrass nucleus 组件的 2.7.0 版、流管理器组件的 2.1.0 版和 Lambda 管理器组件的 2.2.5 版。

发布日期：2022 年 7 月 28 日

发布亮点

- 流管理器遥测指标 — Stream manager 现在会自动将遥测指标发送到 Amazon EventBridge，因此您可以创建云应用程序来监控和分析您的核心设备上传的数据量。有关更多信息，请参阅 从AWS IoT Greengrass核心设备收集系统运行状况遥测数据。
- 自定义证书颁发机构 (CA)-现在支持由自定义证书 CA 签名的客户端证书（CA 未向其中注册）。AWS IoT有关更多信息，请参阅 使用由私有 CA 签名的设备证书。

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关AWS IoT Greengrass核心软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.7.0 版本已上市。 新功能 • 更新 Greengrass 核心，以便在核心设备应用本地部署时向AWS IoT Greengrass云端发送状态更新。 • 添加对由自定义证书颁发机构 (CA) 签名的客户端证书的支持，CA 未在其中注册AWS IoT。要使用此功能，可以将新的`greengrassDataPlaneEndpoint` 配置选项设置为`iotdata`。有关更多信息，请参阅 使用由私有 CA 签名的设备证书。 错误修复和改进 • 修复了在某些情况下，当原子核停止或重新启动时，Greengrass 核会回滚部署的问题。现在，在原子核重启后，原子核会恢复部署。 • 当您指定将软件设置为系统服务时，更新 Greengrass 安装程序以遵守`--start`该参数。 • 更新行为SubscribeToComponentUpdates以在核心更新组件的事件中设置部署 ID。 • 其他小修复和改进。
直播管理器	直播管理器组件已推出 2.1.0 版。

组件	详细信息
	新功能 • 更新此组件以自动向 Amazon EventBridge 发送遥测指标。有关更多信息，请参阅 从AWS IoT Greengrass核心设备收集系统运行状况遥测数据。 此功能需要 Greengrass nucleus 组件的 v2.7.0 或更高版本。 • Greengrass nucleus 版本 2.7.0 版本的版本已更新。
Lambda 管理器	Lambda 管理器组件已推出 2.2.5 版。 新功能 • 在您订阅本地发布/订阅消息的事件源中添加对 MQTT 主题通配符的支持。 此功能需要 Greengrass nucleus 组件的 v2.6.0 或更高版本。 • Greengrass nucleus 版本 2.7.0 版本的版本已更新。

发布：AWS IoT Greengrass酷睿 v2.6.0 软件更新将于 2022 年 6 月 27 日发布

此版本提供了 Greengrass nucleus 组件的 2.6.0 版、新AWS提供的组件以及对提供的组件的更新。AWS

发布日期：2022 年 6 月 27 日

发布亮点

- 本地发布/订阅主题中的通配符 — 订阅本地发布/订阅主题时，您现在可以使用 MQTT 通配符。有关更多信息，请参阅 发布/订阅本地消息和 SubscribeToTopic。
- 客户端设备影子支持-您现在可以在自定义组件中与客户端设备影子进行交互，并与客户端设备影子同步AWS IoT Core。有关更多信息，请参阅 与客户端设备影子进行交互并进行同步。
- 客户端设备支持本地 MQTT 5 — 您现在可以部署 EMQX MQTT 5 代理，以便在客户端设备和核心设备之间的通信中使用 MQTT 5 功能。有关更多信息，请参阅 MQTT 5 经纪商 (EMQX)和 Connect 客户端设备与核心设备连接。

- 组件配置中的配方变量-您现在可以在组件配置中使用特定的配方变量。在配方中定义组件的默认配置或在部署中配置组件时，可以使用这些配方变量。有关更多信息，请参阅 食谱变量和 在合并更新中使用配方变量。
- IPC 授权策略中的通配符-您现在可以使用*通配符匹配进程间通信 (IPC) 授权策略中的任意字符组合。此通配符使您能够在单个授权策略中允许访问多个资源。有关更多信息，请参阅 授权策略中的通配符。
- 管理本地部署和组件的 IPC 操作-您现在可以开发用于管理本地部署和查看组件详细信息的自定义组件。有关更多信息，请参阅 IPC：管理本地部署和组件。
- 对@@ 客户端设备进行身份验证和授权的 IPC 操作-您现在可以使用这些操作来创建自定义的本地代理组件。有关更多信息，请参阅 IPC：对客户端设备进行身份验证和授权。

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	Greengrass nucleus 的 2.6.0 版本已上市。 新功能 • 当您订阅本地发布/订阅主题时，添加对 MQTT 通配符的支持。有关更多信息，请参阅 发布/订阅本地消息和 SubscribeToTopic。

组件	详细信息
	• 在组件配置中添加对配方变量（*component_dependency_name* :configuration: *json_pointer* 配方变量除外）的支持。在配方中定义组件或在部署DefaultConfiguration 中配置组件时，可以使用这些配方变量。要启用此功能，请将interpolateComponentConfiguration配置选项设置为true。有关更多信息，请参阅 食谱变量和 在合并更新中使用配方变量。 • 在进程间通信 (IPC) * 授权策略中添加对通配符的完全支持。现在，您可以在资源字符串中*指定字符以匹配任意字符组合。有关更多信息，请参阅 授权策略中的通配符。 • 添加了对自定义组件的支持，以调用 Greengrass CLI 使用的 IPC 操作。您可以使用这些 IPC 操作来管理本地部署、查看组件详细信息以及生成用于登录本地调试控制台的密码。有关更多信息，请参阅 IPC：管理本地部署和组件。 错误修复和改进 • 修复了依赖组件在某些情况下在硬依赖项重新启动或更改状态时不会做出反应的问题。 • 改进了部署失败时核心设备向AWS IoT Greengrass云服务报告的错误消息。 • 修复了在某些情况下，当原子核重启时，Greengrass 核两次应用事物部署的问题。 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。
MQTT 5 经纪商 (EMQX)	新的 EMQX MQTT 5 代理组件已推出 1.0.0 版。 新功能 • 增加了对本地 EMQX MQTT 5 代理的支持。客户端设备可以连接到此 MQTT 代理，使用 MQTT 5 功能与核心设备通信。

组件	详细信息
影子经理	影子管理器组件已推出 2.2.0 版。 新功能 • 通过本地发布/订阅界面添加对本地影子服务的支持。现在，您可以就影子 MQTT 主题与本地发布/订阅消息代理进行通信，以获取、更新和删除核心设备上的阴影。此功能允许您使用 MQTT 网桥在客户端设备和本地发布/订阅接口之间中继有关影子主题的消息，从而将客户端设备连接到本地影子服务。 此功能需要 Greengrass nucleus 组件的 v2.6.0 或更高版本。要将客户端设备连接到本地影子服务，还必须使用 V2.2.0 或更高版本的 MQTT 桥接组件。 • 添加了可以配置为自定义方向的`direction` 选项，以便在本地阴影服务与之间同步阴影AWS Cloud。您可以配置此选项以减少带宽和与的连接 AWS Cloud。
客户端设备身份验证	客户端设备身份验证组件已推出 2.2.0 版。 新功能 • 添加了对自定义组件的支持，以调用进程间通信 (IPC) 操作来对客户端设备进行身份验证和授权。例如，您可以在自定义 MQTT 代理组件中使用这些操作。有关更多信息，请参阅 IPC：对客户端设备进行身份验证和授权。 • 添加`maxActiveAuthTokens` `cloudQueueSize` 、和`threadPoolSize` 选项，您可以配置这些选项以调整此组件的性能。
MQTT 桥接器	MQTT 桥接组件的 2.2.0 版本现已推出。 新功能 • 当您将本地发布/订阅指定为源消息代理时，添加对 MQTT 主题通配符（#和+）的支持。 此功能需要 Greengrass nucleus 组件的 v2.6.0 或更高版本。 • 添加`targetTopicPrefix` 选项，您可以指定该选项来配置 MQTT 网桥，使其在中继消息时向目标主题添加前缀。

组件	详细信息
Greengrass CLI	Greengrass CLI 的 2.6.0 版本现已推出。 新功能 • 添加了对自定义组件的支持，以调用 Greengrass CLI 使用的进程间通信 (IPC) 操作。您可以使用这些 IPC 操作来管理本地部署、查看组件详细信息以及生成用于登录本地调试控制台的密码。有关更多信息，请参阅 IPC：管理本地部署和组件。 错误修复和改进 • 其他小修复和改进。

发布：AWS IoT Greengrass酷睿 v2.5.6 软件更新将于 2022 年 5 月 31 日发布

此版本提供了 Greengrass nucleus 组件的 2.5.6 版和日志管理器组件的 2.2.4 版。

发布日期：2022 年 5 月 31 日

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

> **Important**
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.5.6 版本已上市。 新功能 • 增加了对使用 ECC 密钥的硬件安全模块的支持。您可以使用硬件安全模块 (HSM) 来安全地存储设备的私钥和证书。有关更多信息，请参阅 硬件安全性集成。 错误修复和改进 • 修复了在某些情况下部署安装脚本损坏的组件时，部署永远不会完成的问题。 • 提高了启动期间的性能。 • 其他小修复和改进。
日志管理器	日志管理器组件的 2.2.4 版本可用。 错误修复和改进 • 提高了处理无效配置时的稳定性。 • 其他小修复和改进。

发布：AWS IoT Greengrass酷睿 v2.5.5 软件更新将于 2022 年 4 月 6 日发布

此版本提供了 Greengrass nucleus 组件的 2.5.5 版。

发布日期：2022 年 4 月 6 日

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

> **Important**
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.5.5 版本已上市。 新功能 • 为组件添加`GG_ROOT_CA_PATH` 环境变量，这样您就可以在自定义组件中访问根证书颁发机构 (CA) 证书。 错误修复和改进 • 添加对使用非英语显示语言的 Windows 设备的支持。 • 更新 Greengrass nucleus 解析布尔安装程序参数的方式，因此您可以指定不带布尔值的布尔参数来指定值。`true`例如，您现在可以指定`--provision` 而不是使用自动资源配置`--provision true` 进行安装。 • 修复了在某些情况下核心设备在配置后未向AWS IoT Greengrass云服务报告其状态的问题。 • 其他小修复和改进。

发布：AWS IoT Greengrass酷睿 v2.5.4 软件更新将于 2022 年 3 月 23 日发布

此版本提供了 Greengrass nucleus 组件的 2.5.4 版和 Lambda 启动器组件的 2.0.10 版。

发布日期：2022 年 3 月 23 日

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.5.4 版本已上市。 错误修复和改进 • 常规错误修复和性能改进。
Lambda 启动器	Lambda 启动器组件已推出 2.0.10 版。 错误修复和改进 • 常规错误修复和性能改进。

发布：AWS IoT Greengrass酷睿 v2.5.3 软件更新将于 2022 年 1 月 6 日发布

此版本提供了 Greengrass nucleus 组件的 2.5.3 版和新的 PKCS #11 提供程序组件。

发布日期：2022 年 1 月 6 日

发布亮点

- 硬件安全集成 — 现在，您可以将 AWS IoT Greengrass Core 软件配置为使用安全存储在硬件安全模块 (HSM) 中的私钥和证书。有关更多信息，请参阅 硬件安全性集成。

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备，或者更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.5.3 版本已上市。 新功能 • 增加了对硬件安全集成的支持。您可以使用硬件安全模块 (HSM) 来安全地存储设备的私钥和证书。有关更多信息，请参阅 硬件安全性集成。 错误修复和改进 • 修复了在 nucleus 与建立 MQTT 连接时出现运行时异常的问题。AWS IoT Core
PKCS #11 提供商	PKCS #11 提供程序组件的 2.0.0 版本现已推出。

组件	详细信息
	新功能 • 增加了对硬件安全集成的支持。您可以使用硬件安全模块 (HSM) 来安全地存储设备的私钥和证书。有关更多信息，请参阅 硬件安全性集成。

发布：AWS IoT Greengrass酷睿 v2.5.2 软件更新将于 2021 年 12 月 3 日发布

此版本提供了 Greengrass nucleus 组件的 2.5.2 版。

发布日期：2021 年 12 月 3 日

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

> **Important**
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.5.2 版本已上市。

组件	详细信息
	错误修复和改进 • 修复了 Greengrass nucleus 更新后，Windows 服务在你停止或重启设备后无法重新启动的问题。
AWS IoT Device Defender	该AWS IoT Device Defender组件的 3.0.1 版本可用。 此版本的AWS IoT Device Defender组件需要的配置参数与 2.x 版本不同。如果您使用版本 2.x 的非默认配置，并且想要从 v2.x 升级到 v3.x，则必须更新该组件的配置。有关更多信息，请参阅AWS IoT Device Defender组件配置。 新功能 • 增加了对运行 Windows 的核心设备的支持。 • 将组件类型从 Lambda 组件更改为通用组件。此组件现在不再依赖旧版订阅路由器组件来创建订阅。 • 添加新的`UseInstaller` 配置参数，允许您选择禁用安装组件依赖项的安装脚本。

发布：AWS IoT Greengrass酷睿 v2.5.1 软件更新将于 2021 年 11 月 23 日发布

此版本提供了 Greengrass nucleus 组件的 2.5.1 版。

发布日期：2021 年 11 月 23 日

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

> **Important**
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会

自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.5.1 版本已上市。 错误修复和改进 • 在 Windows 上添加对 32 位版本的 Java 运行时环境 (JRE) 的支持。 • 更改AWS IoT策略未授予`greengrass:ListThingGroupsForCoreDevice` 权限的核心设备的事物组移除行为。使用此版本，部署会继续，并记录警告，并且在从事物组中移除核心设备时不会移除组件。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。 • 修复了 Greengrass 核心向 Greengrass 组件进程提供的系统环境变量的问题。现在，您可以重新启动组件，使其使用最新的系统环境变量。

发布：AWS IoT Greengrass酷睿 v2.5.0 软件更新将于 2021 年 11 月 12 日发布

此版本提供了 Greengrass nucleus 组件的 2.5.0 版、AWS新提供的组件以及对提供的组件的更新。AWS

发布日期：2021 年 11 月 12 日

发布亮点

- Windows 设备支持-您现在可以在运行 Windows 操作系统的设备上运行AWS IoT Greengrass核心软件。有关更多信息，请参阅 支持的平台和要求和 按操作系统划分的 Greengrass 功能兼容性。
- 新事物组移除行为-现在，您可以从事物组中移除核心设备，以便在下次部署到该设备时移除该事物组的组件。

> **Important**
>
> 由于此更改，核心设备的AWS IoT策略必须具有greengrass:ListThingGroupsForCoreDevice权限。如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则默认AWS IoT策略允许greengrass:*，其中包括此权限。有关更多信息，请参阅 AWS IoT Greengrass 的设备身份验证和授权。

- 硬件安全支持-您现在可以将AWS IoT Greengrass核心软件配置为使用硬件安全模块 (HSM)，这样您就可以安全地存储设备的私钥和证书。有关更多信息，请参阅 硬件安全性集成。
- HTTPS 代理支持-您现在可以将AWS IoT Greengrass核心软件配置为通过 HTTPS 代理进行连接。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。

发布详情

- 平台支持更新
- 公共组件更新

平台支持更新

平台	详细信息
Windows	AWS IoT Greengrass现在支持在以下版本的 Windows 上运行AWS IoT Greengrass核心软件： • Windows 10 • Windows Server 2019 有关更多信息，请参阅 支持的平台和要求和 按操作系统划分的 Greengrass 功能兼容性。

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.5.0 版本已上市。 新功能 • 增加了对运行 Windows 的核心设备的支持。 • 更改移除事物组的行为。使用此版本，您可以从事物组中移除核心设备，以便在下次部署中卸载该事物组的组件。 由于此更改，核心设备的AWS IoT策略必须具有greengrass:ListThingGroupsForCoreDevice 权限。如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则默认AWS IoT策略允许greengrass:* ，其中包括此权限。有关更多信息，请参阅 AWS IoT Greengrass 的设备身份验证和授权。 • 添加了对 HTTPS 代理配置的支持。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。 • 添加新的windowsUser 配置参数。您可以使用此参数来指定在 Windows 核心设备上运行组件时使用的默认用户。有关更多信息，请参阅 配置运行组件的用户。 • 添加了新的httpClient 配置选项，您可以使用这些选项来自定义 HTTP 请求超时时间，从而提高慢速网络的性能。有关更多信息，请参阅 HttpClient 配置参数。 错误修复和改进 • 修复了从组件重启核心设备的 bootstrap 生命周期选项。

组件	详细信息
	• 在配方变量中添加对连字符的支持。 • 修复了按需 Lambda 函数组件的 IPC 授权。 • 改进了日志消息并将非关键日志从级别更改INFO为DEBUG级别，因此日志更有用。 • 移除 Greengrass nucleus 在安装AWS IoT Greengrass具有自动配置功能的 Core 软件时创建的默认令牌交换角色的iot:DescribeCertificate 权限。Greengrass 核不使用此权限。 • 修复了一个问题，使自动配置脚本不需要该iam:GetPolicy 权限（如果iam:CreatePolicy 适用于同一策略）。 • 其他小修复和改进。
Greengrass CLI	G reengrass CLI 的 2.5.0 版本现已推出。 新功能 • 增加了对运行 Windows 的核心设备的支持。 • 添加新的AuthorizedWindowsGroups 配置参数，您可以指定该参数来授权系统组在 Windows 设备上使用 Greengrass CLI。 • 为本地部署添加windowsUser 参数。您可以使用此参数指定用于在 Windows 核心设备上运行组件的用户。

<table>
<tr><th>组件</th><th>详细信息</th></tr>
<tr><td>CloudWatch 指标</td><td>CloudWatch指标组件已推出 3.0.0 版。

此版本的 CloudWatch 指标组件需要的配置参数与版本 2.x 不同。如果您使用版本 2.x 的非默认配置，并且想要从 v2.x 升级到 v3.x，则必须更新该组件的配置。有关更多信息，请参阅CloudWatch指标组件配置。

新功能
• 增加了对运行 Windows 的核心设备的支持。
• 将组件类型从 Lambda 组件更改为通用组件。此组件现在不再依赖旧版订阅路由器组件来创建订阅。
• 添加新的<code>InputTopic</code> 配置参数以指定组件订阅以接收消息的主题。
• 添加新的<code>OutputTopic</code> 配置参数以指定组件向其发布状态响应的主题。
• 添加新的<code>PubSubToIoTCore</code> 配置参数以指定是否发布和订阅 AWS IoT Core MQTT 主题。
• 添加新的<code>UseInstaller</code> 配置参数，允许您选择禁用安装组件依赖项的安装脚本。
错误修复和改进

添加了对输入数据中重复时间戳的支持。</td></tr>
<tr><td>Lambda 管理器</td><td>Lambda 管理器组件已推出 2.2.0 版。

错误修复和改进
• 修复了 Lambda 函数在重启后无法写入日志的问题。
• 修复了当主题中有通配符时，传统订阅路由器会发送重复消息的问题。
• 修复了非固定 Lambda 函数无法使用中的 Greengrass 进程间通信 (IPC) 库的问题。AWS IoT Device SDK</td></tr>
</table>

发布：AWS IoT Greengrass酷睿 v2.4.0 软件更新将于 2021 年 8 月 3 日发布

此版本提供了 Greengrass nucleus 组件的 2.4.0 版、AWS新提供的组件以及对提供的组件的更新。AWS

发布日期：2021 年 8 月 3 日

发布亮点

- 系统资源限制 — Greengrass nucleus 组件现在支持系统资源限制。您可以配置每个组件的进程可以在核心设备上使用的最大 CPU 和 RAM 使用量。有关更多信息，请参阅 为组件配置系统资源限制。
- 暂停/恢复组件 — Greengrass nucleus 现在支持暂停和恢复组件。您可以使用进程间通信 (IPC) 库开发用于暂停和恢复其他组件进程的自定义组件。有关更多信息，请参阅 PauseComponent和 ResumeComponent。
- 使用@@ AWS IoT队列配置进行安装-使用新的AWS IoT队列配置插件在连接到的设备上安装AWS IoT Greengrass核心软件AWS IoT以配置所需AWS资源。设备使用索赔证书进行配置。您可以在制造过程中将索赔证书嵌入到设备上，这样每台设备都可以在上线后立即进行配置。有关更多信息，请参阅 安装具有 AWS IoT 队列配置功能的 C AWS IoT Greengrass ore 软件。
- 使用自定义配置进行安装-开发自定义配置插件，以便在设备上安装 AWS IoT Greengrass Core 软件时配置所需的AWS资源。您可以创建在安装过程中运行的 Java 应用程序，为您的自定义用例设置 Greengrass 核心设备。有关更多信息，请参阅 安装具有自定义资源配置功能的 C AWS IoT Greengrass ore 软件。

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

> **Important**
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.4.0 版本已上市。 新功能 • 增加了对系统资源限制的支持。您可以配置每个组件的进程可以在核心设备上使用的最大 CPU 和 RAM 使用量。有关更多信息，请参阅 为组件配置系统资源限制。 • 添加 IPC 操作以暂停和恢复组件。有关更多信息，请参阅 PauseComponent和 ResumeComponent。 • 添加了对配置插件的支持。您可以指定要在安装期间运行的 JAR 文件，以便为 Greengrass 核心设备配置所需的AWS资源。Greengrass 核心包含一个接口，您可以实现该接口来开发自定义配置插件。有关更多信息，请参阅 安装具有自定义资源配置功能的 C AWS IoT Greengrass ore 软件。 • 向 C AWS IoT Greengrass ore 软件安装程序添加可选`thing-name-policy`参数。在安装具有自动资源配置功能的 C AWS IoT Greengrass ore 软件时，您可以使用此选项来指定现有AWS IoT策略或自定义策略。 错误修复和改进 • 启动时更新日志配置。这修复了启动时未应用日志配置的问题。 • 在安装过程中，更新 nucleus 加载器符号链接以指向 Greengrass 根文件夹中的组件存储。此更新使您能够删除安装 C AWS IoT Greengrass ore 软件时下载的 JAR 文件和其他 Nucleus 工件。 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。
Greengrass CLI	G reengrass CLI 的 2.4.0 版本现已推出。 新功能 • 增加了对系统资源限制的支持。创建本地部署时，可以配置每个组件的进程可以在核心设备上使用的最大 CPU 和 RAM 使用量。有关更多信息，请参阅为组件配置系统资源限制和部署创建命令。
AWS IoT按索赔提供舰队	通过索赔插件配置AWS IoT舰队现已可用。有关更多信息，请参阅 安装具有 AWS IoT 队列配置功能的 C AWS IoT Greengrass ore 软件。

组件	详细信息
	新功能 • 增加了对安装带有AWS IoT队列配置的 AWS IoT Greengrass Core 软件的支持。在安装过程中，设备会AWS IoT连接到以配置所需的AWS资源并下载设备证书以用于常规操作。

发布：AWS IoT Greengrass酷睿 v2.3.0 软件更新将于 2021 年 6 月 29 日发布

此版本提供了 Greengrass nucleus 组件的 2.3.0 版本。

发布日期：2021 年 6 月 29 日

发布亮点

- 大型配置支持 — Greengrass nucleus 组件现在支持最大 10 MB 的部署文档。现在，您可以向 Greengrass 组件部署更大的配置更新。

> **Note**
>
> 要使用此功能，核心设备的AWS IoT策略必须允许该`greengrass:GetDeploymentConfiguration`权限。如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则您的核心设备的AWS IoT策略允许`greengrass:*`，其中包括此权限。有关更多信息，请参阅 AWS IoT Greengrass 的设备身份验证和授权。

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

⚠ Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.3.0 版本已上市。 新功能 • 添加对部署配置文档的支持，最大为 10 MB，从 7 KB（针对事物的部署）或 31 KB（针对事物组的部署）增加了。 要使用此功能，核心设备的AWS IoT策略必须允许该`greengrass:GetDeploymentConfiguration` 权限。如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则您的核心设备的AWS IoT策略允许`greengrass:*` ，其中包括此权限。有关更多信息，请参阅AWS IoT Greengrass 的设备身份验证和授权。 • 添加`iot:thingName` 配方变量。你可以使用这个配方变量来获取配方中AWS IoT核心设备的名称。有关更多信息，请参阅 食谱变量。 错误修复和改进 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。

发布：AWS IoT Greengrass酷睿 v2.2.0 软件更新将于 2021 年 6 月 18 日发布

此版本提供了 Greengrass nucleus 组件的 2.2.0 版、AWS新提供的组件以及对提供的组件的更新。AWS

发布日期：2021 年 6 月 18 日

发布亮点

- 客户端设备支持 AWS-新提供的客户端设备组件使您能够使用云发现将客户端设备连接到核心设备。您可以在 Greengrass 组件中将客户端设备与 Greengrass 组件中的客户端设备同步AWS IoT Core并与客户端设备进行交互。有关更多信息，请参阅 与本地物联网设备互动。
- 本地影子服务-新的影子管理器组件可在核心设备上启用本地影子服务。您可以使用中的 Greengrass 进程间通信 (IPC) 库，在离线时使用此影子服务与本地阴影进行交互。AWS IoT Device SDK您也可以使用影子管理器组件将本地阴影状态与同步AWS IoT Core。有关更多信息，请参阅 与设备阴影互动。

发布详情

- 公共组件更新

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.2.0 版本已上市。 新功能 • 为本地影子管理添加 IPC 操作。

组件	详细信息
	错误修复和改进 • 减小 JAR 文件的大小。 • 减少内存使用量。 • 修复了在某些情况下日志配置未更新的问题。 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。
影子经理	新的影子管理器组件已推出 2.0.0 版。 新功能 • 添加了对经典阴影和命名阴影的支持。 • 增加了对使用 IPC 进行本地阴影管理的支持。 • 添加了对与的阴影同步的支持AWS IoT Core。
客户端设备身份验证	新的客户端设备身份验证组件已推出 2.0.0 版。 新功能 • 增加了对 Greengrass 客户端设备的支持，这些设备是通过 MQTT 连接到核心设备的本地物联网设备。 • 增加了对客户端设备及其 MQTT 操作的身份验证和授权的支持。
Moquette MQTT	新的 M oquette MQTT 经纪商组件已推出 2.0.0 版。 新功能 • 添加对处理与客户端设备通信的本地 Moquette MQTT 代理的支持。
MQTT 桥接器	新的 MQTT 桥接组件已推出 2.0.0 版。 新功能 • 支持在本地 MQTT 代理、本地 Greengrass 发布/订阅代理和 MQTT 代理之间中继消息。AWS IoT Core

组件	详细信息
IP 探测器	新的 IP 探测器组件已推出 2.0.0 版。 新功能 • 增加了将核心设备的本地 MQTT 代理端点报告到AWS IoT Greengrass云服务以供客户端设备连接的支持。
日志管理器	日志管理器组件的 2.1.1 版本可用。 错误修复和改进 • 修复了在某些情况下系统日志配置未更新的问题。
DLR 物体检测	DLR 物体检测已推出 2.1.2 版。 错误修复和改进 • 修复了导致样本 DLR 对象检测推理结果中边界框不准确的图像缩放问题。
TensorFlow 精简版物体检测	精简版物体检测已推出 2.1.1 TensorFlow 版。 错误修复和改进 • 修复了导致样本 TensorFlow Lite 对象检测推理结果中边界框不准确的图像缩放问题。

发布：AWS IoT Greengrass酷睿 v2.1.0 软件更新将于 2021 年 4 月 26 日发布

此版本提供了 Greengrass nucleus 组件的 2.1.0 版，并更新了提供的组件。AWS

发布日期：2021 年 4 月 26 日

发布亮点

- Docker Hub 和亚马逊弹性容器注册表 (Amazon ECR) Registry 集成 — 新的 Docker 应用程序管理器组件使您可以从亚马逊 EC R 下载公共或私有映像。您还可以使用此组件从 Docker Hub 下载公共镜像，然后。AWS Marketplace有关更多信息，请参阅 运行 Docker 容器。

- AWS IoT Greengrass核心软件的 Dockerfile 和 Docker 镜像 — 你可以使用 Greengrass Docker 镜像在使用亚马逊 Linux 2 AWS IoT Greengrass 作为基本操作系统的 Docker 容器中运行。你也可以使用 AWS IoT Greengrass Dockerfile 来构建自己的 Greengrass 镜像。有关更多信息，请参阅 在 Docker 容器中运行 AWS IoT Greengrass 核心软件。
- 支持其他机器学习框架和平台-您可以部署示例机器学习推理组件，这些组件使用预训练的模型使用 TensorFlow Lite 2.5.0 和 DLR 1.6.0 执行样本图像分类和对象检测。此版本还扩展了对 Armv8 (aarch64) 设备的示例机器学习支持。有关更多信息，请参阅 执行机器学习推理。

发布详情

- 平台支持更新
- 公共组件更新

平台支持更新

平台	详细信息
Docker	的 Dockerfile 和 Docker 镜像现AWS IoT Greengrass已推出。 Dockerfile AWS IoT Greengrass提供了 Dockerfile 来构建在亚马逊 Linux 2 (x86_64) 基础映像上安装了AWS IoT Greengrass核心软件和依赖项的容器镜像。您可以修改 Dockerfile 中的基础映像，使其在不同的平台架构AWS IoT Greengrass上运行。 Docker 映像 AWS IoT Greengrass提供了一个预构建的 Docker 镜像，该镜像在亚马逊 Linux 2 (x86_64) 基础映像上安装了AWS IoT Greengrass核心软件和依赖项。 有关更多信息，请参阅 在 Docker 容器中运行 AWS IoT Greengrass 核心软件。

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

Important

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

<table>
<tr><th>组件</th><th>详细信息</th></tr>
<tr><td>Greengrass 核</td><td>
G reengrass nucleus 的 2.1.0 版本已上市。

新功能

支持从 Amazon ECR 中的私有存储库下载 Docker 镜像。
添加以下参数以自定义核心设备上的 MQTT 配置：

<code>maxInFlightPublishes</code> — 可以同时传输的未确认的 MQTT QoS 1 消息的最大数量。
<code>maxPublishRetry</code> — 重试发布失败的消息的最大次数。

添加<code>fleetstatusservice</code> 配置参数以配置核心设备向发布设备状态的时间间隔AWS Cloud。
其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。

错误修复和改进

修复了在 nucleus 重启时导致影子部署重复的问题。
修复了在遇到服务加载异常时导致 nucleus 崩溃的问题。
改进了组件依赖关系解决方案，使包含循环依赖关系的部署失败。
修复了如果插件组件之前已从核心设备中移除，则无法重新部署该组件的问题。

</td></tr>
</table>

组件	详细信息
	• 修复了导致将HOME环境变量设置为 Lambda 组件或以 root 身份运行的组件的*/greengrass/v2* /work目录的问题。现在，该HOME变量已正确设置为运行该组件的用户的主目录。 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。
Docker 应用程序管理器	新的 Docker 应用程序管理器组件已推出 2.0.0 版。 新功能 • 管理从 Amazon ECR 中的私有存储库下载图像的证书。 • 从 Amazon ECR、Docker Hub 和下载公共镜像。AWS Marketplace
Lambda 启动器	Lambda 启动器组件已推出 2.0.4 版。 错误修复和改进 • 修复了组件无法正确传递AddGroupOwner 到 Lambda 函数容器的问题。
旧版订阅路由器	现已推出旧版订阅路由器组件 2.1.0 版。 错误修复和改进 • 添加了对为和指定组件名称而不是 ARN source 的target支持。如果您为订阅指定组件名称，则无需在 Lambda 函数的版本每次更改时都重新配置订阅。
本地调试控制台	本地调试控制台组件已推出 2.1.0 版。 新功能 • 使用 HTTPS 保护您与本地调试控制台的连接。默认情况下，HTTPS 处于启用状态。 错误修复和改进 • 你可以在配置编辑器中关闭闪存栏消息。

组件	详细信息
日志管理器	日志管理器组件已推出 2.1.0 版。 错误修复和改进 • 使用适用于打印到标准输出 (stdout) `logFileDirectoryPath` 和`logFileRegex` 标准错误 (stderr) 的 Greengrass 组件的默认值。 • 将日志上传到 CloudWatch 日志时，通过配置的网络代理正确路由流量。 • 正确处理日志流名称中的冒号字符 (`:`)。 CloudWatch 日志日志流名称不支持冒号。 • 通过从日志流中删除事物组名称来简化日志流名称。 • 删除在正常行为期间打印的错误日志消息。
DLR 图像分类	DLR 图像分类组件的 2.1.1 版现已推出。 新功能 • 使用深度学习运行时 v1.6.0。 • 在 Armv8 (aarch64) 平台上添加对样本图像分类的支持。这扩展了对运行 NVIDIA Jetson 的 Greengrass 核心设备（例如 Jetson Nano）的机器学习支持。 • 启用摄像头集成以进行样本推理。使用新的`UseCamera` 配置参数启用示例推理代码，以访问您的 Greengrass 核心设备上的摄像头，并在本地对捕获的图像运行推理。 • 添加对发布推理结果的AWS Cloud支持。使用新的`PublishResultsOnTopic` 配置参数来指定要发布结果的主题。 • 添加新的`ImageDirectory` 配置参数，使您能够为要对其执行推理的图像指定自定义目录。 错误修复和改进 • 将推理结果写入组件日志文件，而不是单独的推理文件。 • 使用 C AWS IoT Greengrass ore 软件日志模块记录组件输出。 • 使用读AWS IoT Device SDK取组件配置并应用配置更改。

<table>
<tr><th>组件</th><th>详细信息</th></tr>
<tr><td>DLR 物体检测</td><td>DLR 物体检测组件已推出 2.1.1 版。

新功能
使用深度学习运行时 v1.6.0。
在 Armv8 (aarch64) 平台上添加对样本对象检测的支持。这扩展了对运行 NVIDIA Jetson 的 Greengrass 核心设备（例如 Jetson Nano）的机器学习支持。
启用摄像头集成以进行样本推理。使用新的<code>UseCamera</code> 配置参数启用示例推理代码，以访问您的 Greengrass 核心设备上的摄像头，并在本地对捕获的图像运行推理。
添加对发布推理结果的AWS Cloud支持。使用新的<code>PublishResultsOnTopic</code> 配置参数来指定要发布结果的主题。
添加新的<code>ImageDirectory</code> 配置参数，使您能够为要对其执行推理的图像指定自定义目录。
错误修复和改进
将推理结果写入组件日志文件，而不是单独的推理文件。
使用 C AWS IoT Greengrass ore 软件日志模块记录组件输出。
使用读AWS IoT Device SDK取组件配置并应用配置更改。</td></tr>
<tr><td>DLR 图像分类模型存储</td><td>DLR 图像分类模型存储组件的 2.1.1 版现已推出。

新功能
为 Armv8 (aarch64) 平台添加一个 ResNet -50 图像分类模型示例。这扩展了对运行 NVIDIA Jetson 的 Greengrass 核心设备（例如 Jetson Nano）的机器学习支持。</td></tr>
<tr><td>DLR 物体检测模型存储</td><td>DLR 对象检测模型存储组件的 2.1.1 版现已推出。

新功能
为 Armv8 (aarch64) 平台添加一个 YOLOv3 对象检测模型示例。这扩展了对运行 NVIDIA Jetson 的 Greengrass 核心设备（例如 Jetson Nano）的机器学习支持。</td></tr>
</table>

组件	详细信息
DLR 安装程序	DLR 组件的 1.6.1 版现已推出。 新功能 • 安装深度学习运行时 v1.6.0 及其依赖项。 • 添加对在 Armv8 (aarch64) 平台上安装 DLR 的支持。这扩展了对运行 NVIDIA Jetson 的 Greengrass 核心设备（例如 Jetson Nano）的机器学习支持。 错误修复和改进 • 在虚拟环境AWS IoT Device SDK中安装以读取组件配置并应用配置更改。 • 其他小错误修复和改进。
TensorFlow 精简版图像分类	新的TensorFlow 精简版图像分类组件已推出 2.1.0 版。 新功能 • 使用 TensorFlow Lite 添加对样本图像分类推断的支持。
TensorFlow 精简版物体检测	全新 TensorFlow Lite 对象检测组件已推出 2.1.0 版。 新功能 • 使用 TensorFlow Lite 添加对样本对象检测推断的支持。
TensorFlow 精简版图像分类模型存储	全新 TensorFlow Lite 图像分类模型存储组件已推出 2.1.0 版。 新功能 • 使用 Lite 为样本图像分类推断提供预训练的 MobileNet v1 量化模型。TensorFlow
TensorFlow 精简版物体检测模型存储	全新 TensorFlow Lite 对象检测模型存储组件已推出 2.1.0 版。 新功能 • 提供在 COCO 数据集中训练的预训练单枪检测 (SSD) MobileNet 模型，用于使用 TensorFlow Lite 进行样本对象检测推断。

组件	详细信息
TensorFlow 精简版	全新 TensorFlow Lite 组件已推出 2.5.0 版。 新功能 • 在 Armv7、Armv8 (aarch64) 和 x86_64 平台上的虚拟环境中安装 TensorFlow Lite v1.6.0 及其依赖项。

发布：AWS IoT Greengrass酷睿 v2.0.5 软件将于 2021 年 3 月 9 日更新

此版本提供了 Greengrass nucleus 组件的 2.0.5 版，并更新了提供的组件。AWS它修复了网络代理支持问题和中国地区Greengrass数据平面端点的问题。AWS

发布日期：2021 年 3 月 9 日

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

> **Important**
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.0.5 版本已上市。

组件	详细信息
	错误修复和改进 • 下载AWS提供的组件时，通过配置的网络代理正确路由流量。 • 在中国区域使用正确的 Greengrass 数据平面终端节点。AWS

发布：AWS IoT Greengrass酷睿 v2.0.4 软件将于 2021 年 2 月 4 日更新

此版本提供了 Greengrass nucleus 组件的 2.0.4 版本。它包括用于通过端口 443 配置 HTTPS 通信的新`greengrassDataPlanePort`参数，并修复了错误。现在，最低限度 IAM 策略要求`sts:GetCallerIdentity`使用`iam:GetPolicy`和运行AWS IoT Greengrass核心软件安装程序`--provision true`。

发布日期：2021 年 2 月 4 日

公共组件更新

下表列出了AWS由提供的组件，其中包括新的和更新的功能。

> **Important**
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

组件	详细信息
Greengrass 核	G reengrass nucleus 的 2.0.4 版本已上市。

组件	详细信息
	新功能 • 启用通过端口 443 的 HTTPS 流量。你可以使用 nucleus 组件版本 2.0.4 的新`greengrassDataPlanePort` 配置参数将 HTTPS 通信配置为通过端口 443 而不是默认端口 8443 传输。有关更多信息，请参阅 通过端口 443 配置 HTTPS。 • 添加工作路径配方变量。您可以使用此配方变量来获取组件工作文件夹的路径，您可以使用该路径在组件及其依赖项之间共享文件。有关更多信息，请参阅工作路径配方变量。 错误修复和改进 • 如果角色策略已经存在，则阻止创建令牌交换 AWS Identity and Access Management (IAM) 角色策略。 由于这一更改，安装程序现在需要`sts:GetCallerIdentity` 使用`iam:GetPolicy` 和`--provision true` 。有关更多信息，请参阅安装程序配置资源的最低 IAM 政策。 • 正确处理尚未成功注册的部署的取消问题。 • 更新配置以在回滚部署时删除带有较新时间戳的较旧条目。 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。

从AWS IoT Greengrass版本 1 迁移

AWS IoT Greengrass Version 2是AWS IoT Greengrass核心软件、API 和控制台的主要版本。 AWS IoT Greengrass V2引入了多项改进AWS IoT Greengrass V1，例如模块化应用程序、部署到大型设备群以及对其他平台的支持。

> **Note**
>
> 2023 年 6 月 30 日之后，将AWS IoT Greengrass Version 1不再收到功能更新、增强功能、错误修复或安全补丁。有关更多信息，请参阅 AWS IoT Greengrass V1维护策略。如果您使用AWS IoT Greengrass V1，我们强烈建议您迁移到AWS IoT Greengrass V2。

按照本指南中的说明从迁移AWS IoT Greengrass V1到AWS IoT Greengrass V2。

我能否在 V2 上运行我的 V1 应用程序？

大多数 V1 应用程序无需更改应用程序代码即可在 V2 核心设备上运行。如果您的 V1 应用程序使用以下功能，则将无法在 V2 上运行它们。

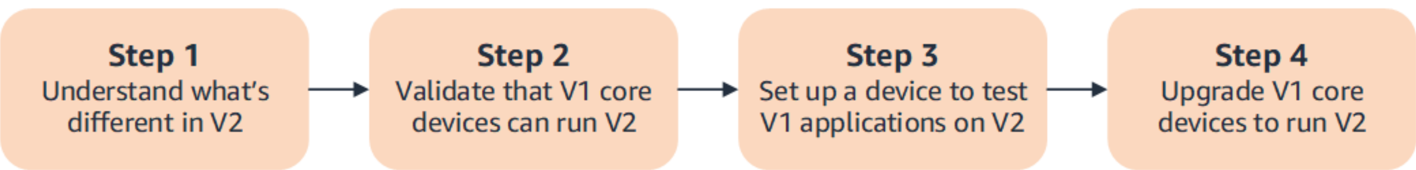
- C 和 C++ Lambda 函数运行时

如果您的 V1 应用程序使用以下任一功能，则必须修改应用程序代码才能使用 AWS IoT Device SDK V2 运行应用程序。AWS IoT Greengrass V2

- 与本地影子服务互动
- 将消息发布到本地连接的设备（Greengrass 设备）

迁移概述

简而言之，您可以使用以下步骤将核心设备从升级AWS IoT Greengrass V1到AWS IoT Greengrass V2。您遵循的确切步骤取决于您的环境的特定要求。



1. [了解 V1 和 V2 之间的区别]

   AWS IoT Greengrass V2为设备群和可部署软件引入了新的基本概念，V2 简化了 V1 中的几个概念。

   AWS IoT Greengrass V2云服务和AWS IoT Greengrass核心软件 v2.x 与AWS IoT Greengrass V1云服务和核心软件 v1.x 不向后兼容。AWS IoT Greengrass因此，AWS IoT Greengrass V1 over-the-air (OTA) 更新无法将核心设备从 V1 升级到 V2。
2. [验证 V1 核心设备是否可以运行 V2]

   验证 V1 核心设备是否可以运行 Cor AWS IoT Greengrass e 软件 v2.x 和功能。AWS IoT Greengrass V2 AWS IoT Greengrass V2具有与之不同的设备要求AWS IoT Greengrass V1。
3. [设置新设备以在 V2 上测试 V1 应用程序]

   为了最大限度地降低生产环境中设备的风险，请创建一台新设备来测试 V2 上的 V1 应用程序。安装AWS IoT Greengrass核心软件 v2.x 后，您可以创建和部署AWS IoT Greengrass V2组件来迁移和测试应用程序AWS IoT Greengrass V1。
4. [升级 V1 核心设备以运行 V2]

   升级现有的 V1 核心设备以运行AWS IoT Greengrass核心软件 v2.x 和组件。AWS IoT Greengrass V2要将设备队列从 V1 迁移到 V2，请对队列中的每台设备重复此步骤。

AWS IoT Greengrass V1 和之间的区别 AWS IoT Greengrass V2

AWS IoT Greengrass V2 为设备、舰队和可部署软件引入了新的基本概念。本节介绍了 V2 中不同的 V1 概念。

Greengrass 的概念和术语

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
应用程序代码	在中 AWS IoT Greengrass V1，Lambda 函数定义了在核心设备上运行的软件。在每个 Greengrass 组中，您可以定义该函数使用的订阅和本地资源。对于 AWS IoT Greengrass 核心软件在容器化 Lambda	在中 AWS IoT Greengrass V2，组件是在核心设备上运行的软件模块。 • 每个组件都有一个配方，用于定义组件的元数据、参数、依赖关系和脚本，以便

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
	运行时环境中运行的 Lambda 函数，您可以定义容器参数，例如内存限制。	在组件生命周期的每个步骤中运行。 • 该配方还定义了组件的工件，即二进制文件，例如脚本、编译后的代码和静态资源。 • 将组件部署到核心设备时，核心设备会下载组件配方和工件来运行该组件。 您可以将 V1 Lambda 函数作为在 Lambda 运行时环境中运行的组件导入。 AWS IoT Greengrass V2导入 Lambda 函数时，您需要为该函数指定订阅、本地资源和容器参数。有关更多信息，请参阅步骤 2：创建和部署 AWS IoT Greengrass V2 组件以迁移 AWS IoT Greengrass V1 应用程序。 有关如何创建自定义组件的更多信息，请参阅开发AWS IoT Greengrass组件。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass 群组和部署	在中 AWS IoT Greengrass V1，一个组定义了核心设备、该核心设备的设置和软件，以及可以连接到该核心设备的设备列表。 AWS IoT 您可以创建部署以将组的配置发送到核心设备。	在中 AWS IoT Greengrass V2，您可以使用部署来定义在核心设备上运行的软件组件和配置。 • 每个部署都以单个核心设备（即 AWS IoT 事物）或可包含多个核心设备 AWS IoT 的事物组为目标。 • 对事物组的部署是连续的，因此，当您将核心设备添加到事物组时，它会收到该组的软件配置。 有关更多信息，请参阅将AWS IoT Greengrass组件部署到设备。 在中 AWS IoT Greengrass V2，您还可以使用 Greengrass CLI 创建本地部署，以便在开发自定义软件组件的设备上对其进行测试。有关更多信息，请参阅创建 AWS IoT Greengrass 组件。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass 核心软件	在中 AWS IoT Greengrass V1， AWS IoT Greengrass Core 软件是一个包含该软件及其所有功能的单个软件包。安装 AWS IoT Greengrass 核心软件的边缘设备被称为 Greengrass 内核。	在中 AWS IoT Greengrass V2， AWS IoT Greengrass Core 软件是模块化的，因此您可以选择要安装的软件来控制内存占用。 • Greengrass nucleus 组件是核心软件所需的最低安装量。 AWS IoT Greengrass 安装原子核的边缘设备被称为 Greengrass 核心设备。 • nucleus 负责处理核心设备上其他组件的部署、编排和生命周期管理。 • 流管理器、密钥管理器和日志管理器等功能是只有在需要这些功能时才部署的组件。有关更多信息，请参阅AWS-提供的组件。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
连接器	在中 AWS IoT Greengrass V1，连接器是预先构建的模块，您可以将其部署到 AWS IoT Greengrass V1 核心设备上，以便与本地基础架构、设备协议和其他云服务进行交互。 AWS	在中 AWS IoT Greengrass V2， AWS 提供了 Greengrass 组件，这些组件实现了 V1 中连接器提供的功能。以下 AWS IoT Greengrass V2 组件提供 Greengrass V1 连接器功能： • CloudWatch 指标组件 • AWS IoT Device Defender 组件 • Firehose 组件 • modbus-RTU 协议适配器组件 • 亚马逊 SNS 组件 有关更多信息，请参阅AWS-提供的组件。

<table>
<tr><th>概念</th><th>AWS IoT Greengrass V1</th><th>AWS IoT Greengrass V2</th></tr>
<tr><td>连接的设备（Greengrass 设备）</td><td>在中 AWS IoT Greengrass V1，联网设备是 AWS IoT 指您添加到 Greengrass 组中，用于连接到该组中的核心设备并通过 MQTT 进行通信的设备。每次添加或移除连接的设备时，都必须部署该组。您可以使用订阅在连接的设备和核心设备上的应用程序之间中继消息。 AWS IoT Core</td><td>在中 AWS IoT Greengrass V2，连接的设备被称为 Greengrass 客户端设备。

• 您可以将客户端设备与核心设备关联以连接它们并通过 MQTT 进行通信。
• 要授权客户端设备连接，您需要定义可应用于客户端设备组的授权策略，因此无需创建部署即可添加或移除客户端设备。
• 要在客户端设备和 Greengrass 组件之间中继消息 AWS IoT Core，您可以配置可选的 MQTT 桥接组件。

在这两种模式中 AWS IoT Greengrass V1 AWS IoT Greengrass V2，设备都可以运行 FreeRTOS，也可以使用AWS IoT Device SDK或Greengrass 发现 API 来获取有关它们可以连接的核心设备的信息。Greengrass 发现 API 向后兼容，因此，如果您的客户端设备连接到 V1 核心设备，则无需更改其代码即可将它们连接到 V2 核心设备。

有关客户端设备的更多信息，请参阅与本地物联网设备互动。</td></tr>
</table>

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
本地资源	在中 AWS IoT Greengrass V1，可以将容器中运行的 Lambda 函数配置为访问核心设备文件系统上的卷和设备。这些文件系统资源称为本地资源。	在中 AWS IoT Greengrass V2，您可以运行 Lambda 函数、Docker 容器、本机操作系统进程或自定义运行时的组件。 • 将容器化 Lambda 函数作为组件导入时，必须指定该函数使用的本地资源。 • 非容器化的 Lambda 函数和非 Lambda 组件可以直接使用核心设备上的本地资源，因此您无需指定该组件使用的本地资源。
本地影子服务	在中 AWS IoT Greengrass V1，本地阴影服务默认处于启用状态，并且仅支持未命名的经典阴影。您可以在 Lambda 函数中使用 AWS IoT Greengrass 核心软件开发工具包与设备上的阴影进行交互。	在中 AWS IoT Greengrass V2，您可以通过部署影子管理器组件来启用本地影子服务。 • 您可以在 Lambda 函数和自定义组件中使用 AWS IoT Device SDK V2 与设备上的阴影进行交互。 • 本地影子服务支持命名阴影。 • 本地阴影服务允许您删除阴影并将已删除的阴影与同步 AWS IoT Core。 有关更多信息，请参阅与设备阴影互动。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2

订阅	在中 AWS IoT Greengrass V1，您可以为 Greengrass 组定义订阅，以指定 Lambda 函数、连接器、连接设备、 AWS IoT Core MQTT 代理和本地影子服务之间的通信渠道。订阅指定 Lambda 函数在何处接收事件消息以作为函数负载使用。	在中 AWS IoT Greengrass V2，您可以在不使用订阅的情况下指定通信渠道。
		
		• 组件管理自己的通信渠道，以便与本地发布/订阅消息、AWS IoT Core MQTT 消息和本地影子服务进行交互。
		  ◦ 要开发能够对来自其他组件或 AWS IoT Core MQTT 代理的消息做出反应的组件，您可以将进程间通信 (IPC) 接口用于本地发布/ 订阅消息和 MQTT 消息传递。AWS IoT Core
		  ◦ 要开发与本地影子服务交互的组件，您可以使用本地影子服务的 IPC 接口。
		  ◦ 在组件配置中，您可以定义授权策略来指定组件有权使用的主题和本地阴影。
		• 要配置客户端设备、本地发布/订阅代理和 MQTT 代理之间的通信渠道，您需要配置和部署 AWS IoT Core MQTT 桥接组件。MQTT 桥接组件使您能够与组件中的客户端设备进行交互，并在客户端设备和 AWS IoT Core之间中继消息。

概念	AWS IoT Greengrass V1	AWS IoT Greengrass V2
访问其他 AWS 服务	在中 AWS IoT Greengrass V1，您将一个名为群组角色的 AWS Identity and Access Management (IAM) 角色附加到 Greengrass 群组。群组角色定义了该群组核心设备上的 Lambda 函数和 AWS IoT Greengrass 功能所使用的访问权限。 AWS 服务	在中 AWS IoT Greengrass V2，您将 AWS IoT 角色别名附加到 Greengrass 核心设备。角色别名指向一个名为令牌交换角色的 IAM 角色。令牌交换角色定义了核心设备上的 Greengrass 组件用来访问的权限。 AWS 服务有关更多信息，请参阅 授权核心设备与AWS服务。

验证 V1 核心设备能否运行 V2 软件

AWS IoT Greengrass核心软件 v2.x 的要求与AWS IoT Greengrass核心软件 v1.x 的要求不同。在将 V1 核心设备升级到 V2 之前，请查看的设备要求AWS IoT Greengrass V2。 AWS IoT Greengrass V2 目前不支持使用 Yocto 项目迁移基于 Linux 的自定义系统。

您可以使用 AWS IoT Device Tester(IDT)AWS IoT Greengrass V2 来验证设备是否符合运行AWS IoT Greengrass Core 软件 v2.x 的要求。IDT 是一个可下载的测试框架，它在您的主机上运行并连接到要验证的设备。按照说明使用 IDT 运行AWS IoT Greengrass资格套件。配置 IDT 时，您可以选择验证设备是否支持可选功能，例如 Docker、机器学习 (ML)、数据流管理和硬件安全集成。

如果 IDT 报告 V1 核心设备的 V2 测试失败或错误，则无法将该设备从 V1 升级到 V2。

设置新的 V2 核心设备来测试 V1 应用程序

设置新的 AWS IoT Greengrass V2 核心设备来部署和测试为您的 AWS IoT Greengrass V1 应用程序AWS提供的组件和 AWS Lambda 功能。您还可以使用此 V2 核心设备开发和测试在核心设备上运行本机进程的其他自定义 Greengrass 组件。在 V2 核心设备上测试应用程序后，可以将现有的 V1 核心设备升级到 V2，然后部署提供 V1 功能的 V2 组件。

步骤 1：在新设备 AWS IoT Greengrass V2 上安装

在新设备上安装 AWS IoT Greengrass 核心软件 v2.x。您可以按照入门教程设置设备并学习如何开发和部署组件。本教程使用自动配置来快速设置设备。安装 AWS IoT Greengrass 核心软件 v2.x 时，请

指定用于部署 G reengrass CLI 的`--deploy-dev-tools`参数，这样您就可以直接在设备上开发、测试和调试组件。有关其他安装选项的更多信息，包括如何在代理服务器后面安装 AWS IoT Greengrass 核心软件或使用硬件安全模块 (HSM)，请参阅安装 AWS IoT Greengrass Core 软件。

（可选）启用对 Amazon CloudWatch 日志的登录

要让 V2 核心设备能够将日志上传到 Amazon CloudWatch Logs，您可以部署 AWS提供的日志管理器组件。您可以使用 CloudWatch 日志查看组件日志，因此无需访问核心设备的文件系统即可进行调试和故障排除。有关更多信息，请参阅监控AWS IoT Greengrass日志。

步骤 2：创建和部署 AWS IoT Greengrass V2 组件以迁移 AWS IoT Greengrass V1 应用程序

大多数 AWS IoT Greengrass V1 应用程序都可以在上运行 AWS IoT Greengrass V2。您可以将 Lambda 函数作为在上面运行的组件导入 AWS IoT Greengrass V2，也可以使用AWS提供与连接器相同功能的组件。 AWS IoT Greengrass

您还可以开发自定义组件来构建任何功能或运行时以在 Greengrass 核心设备上运行。有关如何在本地开发和测试组件的信息，请参阅创建 AWS IoT Greengrass 组件。

主题

- 导入 V1 Lambda 函数
- 使用 V1 连接器
- 运行 Docker 容器
- 运行机器学习推理
- 连接 V1 Greengrass 设备
- 启用本地影子服务
- 与集成 AWS IoT SiteWise

导入 V1 Lambda 函数

您可以将 Lambda 函数作为 AWS IoT Greengrass V2 组件导入。从以下方法中进行选择：

- 将 V1 Lambda 函数直接导入 Greengrass 组件。
- 更新你的 Lambda 函数以使用 v2 中的 Greengrass 库，然后将 Lambda 函数作为 Greengrass AWS IoT Device SDK 组件导入。

- 创建使用非 Lambda 代码和 AWS IoT Device SDK v2 的自定义组件，以实现与您的 Lambda 函数相同的功能。

如果您的 Lambda 函数使用流管理器或本地密钥等功能，则必须定义对打包这些功能的组件的 AWS依赖关系。部署 Lambda 函数组件时，部署还包括您定义为依赖项的每个功能的组件。在部署中，您可以配置参数，例如要将哪些密钥部署到核心设备。并非所有 V1 功能都需要在 V2 上的 Lambda 函数依赖组件。以下列表描述了如何在 V2 Lambda 函数组件中使用 V1 功能。

- 访问其他 AWS 服务

  如果您的 Lambda 函数使用 AWS 凭证向其他 AWS 服务发出请求，则核心设备的令牌交换角色必须允许核心设备执行 Lambda 函数使用的 AWS 操作。有关更多信息，请参阅授权核心设备与AWS服务。
- 直播管理器

  如果您的 Lambda 函数使用流管理器，请在导入函数时指定`aws.greengrass.StreamManager`为组件依赖关系。部署流管理器组件时，请指定要为目标核心设备设置的流管理器参数。核心设备的令牌交换角色必须允许核心设备访问您在直播管理器中使用的 AWS Cloud 目的地。有关更多信息，请参阅流管理器。
- 当地秘密

  如果您的 Lambda 函数使用本地密钥，请在导入函数时指定`aws.greengrass.SecretManager`为组件依赖关系。部署密钥管理器组件时，请指定要部署到目标核心设备的机密资源。核心设备的令牌交换角色必须允许核心设备检索要部署的秘密资源。有关更多信息，请参阅秘密经理。

  在部署 Lambda 函数组件时，请将其配置为具有 IPC 授权策略，该策略授予在 V2 中使用 GetSecretValue IPC 操作的权限。 AWS IoT Device SDK
- 局部阴影

  如果您的 Lambda 函数与本地阴影交互，则必须更新 Lambda 函数代码才能使用 V2。 AWS IoT Device SDK 在导入函数时`aws.greengrass.ShadowManager`，还必须指定为组件依赖关系。有关更多信息，请参阅与设备阴影互动。

  在部署 Lambda 函数组件时，请将其配置为具有 IPC 授权策略，该策略授予在 V2 中使用影子 IPC 操作的权限。 AWS IoT Device SDK
- 订阅
  - 如果您的 Lambda 函数订阅了来自云源的消息，请在导入该函数时将这些订阅指定为事件源。

  - 如果您的 Lambda 函数订阅了来自其他 Lambda 函数的消息，或者您的 Lambda 函数向或 AWS IoT Core 其他 Lambda 函数发布消息，请在部署 Lambda 函数时配置和部署旧版订阅路由器组件。部署旧版订阅路由器组件时，请指定 Lambda 函数使用的订阅。

    > **Note**
    >
    > 只有当您的 Lambda 函数使用 AWS IoT Greengrass 核心软件开发工具包中的`publish()`函数时，才需要使用旧版订阅路由器组件。如果您更新 Lambda 函数代码以使用 AWS IoT Device SDK V2 中的进程间通信 (IPC) 接口，则无需部署旧版订阅路由器组件。有关更多信息，请参阅以下进程间通信服务：
    >
    > - 发布/订阅本地消息
    > - 发布/订阅 AWS IoT Core MQTT 消息

  - 如果您的 Lambda 函数订阅了来自本地连接设备的消息，请在导入该函数时将这些订阅指定为事件源。您还必须配置和部署 MQTT 桥接组件，以便将来自所连接设备的消息中继到您指定为事件源的本地发布/订阅主题。
  - 如果您的 Lambda 函数向本地连接的设备发布消息，则必须更新 Lambda 函数代码以使用 AWS IoT Device SDK V2 发布本地发布/订阅消息。您还必须配置和部署 MQTT 网桥组件，以便将来自本地发布/订阅消息代理的消息中继到连接的设备。
- 本地卷和设备

  如果您的容器化 Lambda 函数访问本地卷或设备，请在导入 Lambda 函数时指定这些卷和设备。此功能不需要组件依赖关系。

有关更多信息，请参阅运行AWS Lambda函数。

使用 V1 连接器

您可以部署 AWS由提供的组件，这些组件的功能与某些 AWS IoT Greengrass 连接器相同。创建部署时，可以配置连接器的参数。

以下 AWS IoT Greengrass V2 组件提供 Greengrass V1 连接器功能：

- CloudWatch 指标组件
- AWS IoT Device Defender 组件
- Firehose 组件
- modbus-RTU 协议适配器组件

- 亚马逊 SNS 组件

运行 Docker 容器

AWS IoT Greengrass V2 不提供直接替换 V1 Docker 应用程序部署连接器的组件。但是，您可以使用 Docker 应用程序管理器组件下载 Docker 镜像，然后根据下载的镜像创建运行 Docker 容器的自定义组件。有关更多信息，请参阅 运行 Docker 容器 和 Docker 应用程序管理器。

运行机器学习推理

AWS IoT Greengrass V2 提供了一个 Amazon SageMaker Edge Manager 组件，用于安装 Amazon SageMaker Edge Manager 代理，并允许您在 Greengrass 核心设备上使用 SageMaker Neo 编译的模型作为模型组件。 AWS IoT Greengrass V2 还提供了在您的设备上安装深度学习运行时和TensorFlow 精简版的组件。您可以使用相应的 DLR 和 L TensorFlow ite 模型以及推理组件来执行样本图像分类和物体检测推理。要使用其他机器学习框架，例如 MxNet 和 TensorFlow，您可以开发自己的使用这些框架的自定义组件。

连接 V1 Greengrass 设备

中连接的设备 AWS IoT Greengrass V1 在中称为客户端设备 AWS IoT Greengrass V2。 AWS IoT Greengrass V2 对客户端设备的支持向后兼容 AWS IoT Greengrass V1，因此您可以将 V1 客户端设备连接到 V2 核心设备，而无需更改其应用程序代码。要使客户端设备能够连接到 V2 核心设备，请部署支持客户端设备的 Greengrass 组件，并将客户端设备与核心设备关联。要在客户端设备、 AWS IoT Core 云服务和 Greengrass 组件（包括 Lambda 函数）之间中继消息，请部署和配置 MQTT 桥接组件。您可以部署 IP 检测器组件来自动检测连接信息，也可以手动管理端点。有关更多信息，请参阅与本地物联网设备互动。

启用本地影子服务

在中 AWS IoT Greengrass V2，本地影子服务由 AWS提供的影子管理器组件实现。 AWS IoT Greengrass V2 还包括对命名阴影的支持。要使您的组件能够与本地阴影交互并将阴影状态同步到 AWS IoT Core，请配置和部署影子管理器组件，并在组件代码中使用影子 IPC 操作。有关更多信息，请参阅与设备阴影互动。

与集成 AWS IoT SiteWise

如果您使用 V1 核心设备作为 AWS IoT SiteWise 网关，请按照说明将新 V2 核心设备设置为 AWS IoT SiteWise 网关。 AWS IoT SiteWise 提供了为您部署 AWS IoT SiteWise 组件的安装脚本。

步骤 3：测试您的 AWS IoT Greengrass V2 应用程序

在创建 V2 组件并将其部署到新的 V2 核心设备后，请验证您的应用程序是否符合您的期望。您可以查看设备的日志，查看组件的标准输出 (stdout) 和标准错误 (stderr) 消息。有关更多信息，请参阅监控 AWS IoT Greengrass日志。

如果您将 Greengrass CLI 部署到核心设备，则可以使用它来调试组件及其配置。有关更多信息，请参阅Greengrass CLI 命令。

验证您的应用程序在 V2 核心设备上运行后，您可以将应用程序的 Greengrass 组件部署到其他核心设备。如果您开发了运行本机进程或 Docker 容器的自定义组件，则必须先将这些组件发布到 AWS IoT Greengrass 服务，然后才能将其部署到其他核心设备。

将 Greengrass V1 核心设备升级到 Greengrass V2

验证您的应用程序和组件在AWS IoT Greengrass V2核心设备上运行后，可以在当前运行 v1.x 的设备（例如生产设备）上安装 C AWS IoT Greengrass ore 软件 v2.x。然后，部署 Greengrass V2 组件，在设备上运行你的 Greengrass 应用程序。

要将设备队列从 V1 升级到 V2，请为每台要升级的设备完成以下步骤。您可以使用事物组将 V2 组件部署到核心设备队列中。

> **Tip**
>
> 我们建议您创建一个脚本来自动执行设备队列的升级过程。如果您使用AWS Systems Manager管理队列，则可以使用 Systems Manager 在每台设备上运行该脚本，将您的队列从 V1 升级到 V2。
> 您可以联系您的 E AWS nterprise Support 代表，询问有关如何以最佳方式自动执行升级过程的问题。

步骤 1：安装AWS IoT Greengrass核心软件 v2.x

从以下选项中进行选择，在 V1 AWS IoT Greengrass 核心设备上安装 Core 软件 v2.x：

- 只需更少的步骤即可升级

  要以更少的步骤进行升级，可以在安装 v2.x 软件之前卸载 v1.x 软件。

- 升级时停机时间最短

  要在最短的停机时间内进行升级，您可以同时安装两个版本的 AWS IoT Greengrass Core 软件。安装AWS IoT Greengrass核心软件 v2.x 并验证 V2 应用程序是否正常运行后，即可卸载 C AWS IoT Greengrass ore 软件 v1.x。在选择此选项之前，请考虑同时运行两个版本的AWS IoT Greengrass酷睿软件所需的额外内存。

在安装 v2.x 之前卸载 C AWS IoT Greengrass ore v1.x

如果要按顺序升级，请在设备上安装 v2.x 之前卸载 C AWS IoT Greengrass ore 软件 v1.x。

卸载AWS IoT Greengrass核心软件 v1.x

1. 如果AWS IoT Greengrass核心软件 v1.x 作为服务运行，则必须停止、禁用和删除该服务。

   a. 停止正在运行的AWS IoT Greengrass核心软件 v1.x 服务。

   ```
   sudo systemctl stop greengrass
   ```

   b. 等到服务停止。您可以使用`list`命令来检查服务的状态。

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

   c. 禁用该服务。

   ```
   sudo systemctl disable greengrass
   ```

   d. 删除该服务。

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

2. 如果 C AWS IoT Greengrass ore 软件 v1.x 未作为服务运行，请使用以下命令停止守护程序。将 *greengrass-root ### Greengrass #*文件夹的名称。默认位置是 `/greengrass`。

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

3. （可选）将 Greengrass 根文件夹以及自定义写入文件夹（如果适用）备份到设备上的其他文件夹。

   a. 使用以下命令将当前 Greengrass 根文件夹复制到其他文件夹，然后删除该根文件夹。

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

b. 使用以下命令将写入文件夹移至其他文件夹，然后移除写入文件夹。

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

然后，您可以使用的安装说明在您的设备上安装该软件。AWS IoT Greengrass V2

> **Tip**
>
> 要在将核心设备从 V1 迁移到 V2 时重复使用其身份，请按照说明通过手动配置安装AWS IoT Greengrass核心软件。首先从设备中删除 V1 核心软件，然后重复使用 V1 核心设备的AWS IoT东西和证书，并更新证书的AWS IoT策略以授予 v2.x 软件所需的权限。

在已经运行 v1.x 的设备上安装AWS IoT Greengrass酷睿软件 v2.x

如果您在已经运行AWS IoT Greengrass酷睿软件 v1.x 的设备上安装 C AWS IoT Greengrass ore v2.x 软件，请记住以下几点：

- V2 核心设备AWS IoT的事物名称必须是唯一的。不要使用与 V1 核心设备相同的名称。
- 用于AWS IoT Greengrass核心软件 v2.x 的端口必须与用于 v1.x 的端口不同。
  - 将 V1 流管理器配置为使用 8088 以外的端口。有关更多信息，请参阅配置直播管理器。
  - 将 V1 MQTT 代理配置为使用 8883 以外的端口。有关更多信息，请参阅为本地消息配置 MQTT 端口。
- AWS IoT Greengrass V2不提供重命名 Greengrass 系统服务的选项。如果您将 Greengrass 作为系统服务运行，则必须执行以下操作之一，以避免系统服务名称发生冲突：
  - 在安装 v2.x 之前，请重命名 v1.x 的 Greengrass 服务。
  - 在没有系统服务的情况下安装 C AWS IoT Greengrass ore 软件 v2.x，然后手动将该软件配置为系统服务，其名称不是。`greengrass`

  重命名 v1.x 版的 Greengrass 服务

  1. 停止AWS IoT Greengrass核心软件 v1.x 服务。

```
sudo systemctl stop greengrass
```

2. 等待服务停止。该服务可能需要几分钟才能停止。您可以使用`list-units`命令来检查服务是否已停止。

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

3. 禁用该服务。

```
sudo systemctl disable greengrass
```

4. 重命名服务。

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

5. 重新加载服务并启动它。

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

然后，您可以使用的安装说明在您的设备上安装该软件。AWS IoT Greengrass V2

> **Tip**
>
> 要在将核心设备从 V1 迁移到 V2 时重复使用其身份，请按照说明通过手动配置安装AWS IoT Greengrass核心软件。首先从设备中删除 V1 核心软件，然后重复使用 V1 核心设备的AWS IoT东西和证书，并更新证书的AWS IoT策略以授予 v2.x 软件所需的权限。

步骤 2：将AWS IoT Greengrass V2组件部署到核心设备

在设备上安装 C AWS IoT Greengrass ore 软件 v2.x 后，创建包含以下资源的部署。要将组件部署到由相似设备组成的队列，请为包含这些设备的事物组创建部署。

- 您通过 V1 Lambda 函数创建的 Lambda 函数组件。有关更多信息，请参阅 运行AWS Lambda函数。

- 如果您使用 V1 订阅，则为旧版订阅路由器组件。
- 如果你使用直播管理器，则是直播管理器组件。有关更多信息，请参阅 管理 Greengrass 核心设备上的数据流。
- 如果你使用本地密钥，则是密钥管理器组件。
- 如果您使用 V1 连接器，则AWS为提供的连接器组件。
- 如果你使用 Docker 容器，那就是 Docker 应用程序管理器组件。有关更多信息，请参阅 运行 Docker 容器。
- 如果您使用机器学习推理，则为机器学习提供支持的组件。有关更多信息，请参阅 执行机器学习推理。
- 如果您使用连接的设备，则客户端设备的组件将支持。您还必须启用客户端设备支持，并将客户端设备与核心设备关联。有关更多信息，请参阅 与本地物联网设备互动。
- 如果您使用设备影子，则为影子管理器组件。有关更多信息，请参阅 与设备阴影互动。
- 如果您将日志从 Greengrass 核心设备上传到日志管理器组件 CloudWatch Amazon Logs。有关更多信息，请参阅 监控AWS IoT Greengrass日志。
- 如果您与集成AWS IoT SiteWise，请按照说明将 V2 核心设备设置为AWS IoT SiteWise网关。 AWS IoT SiteWise提供了为您部署AWS IoT SiteWise组件的安装脚本。
- 您为实现自定义功能而开发的用户定义组件。

有关创建和修改部署的信息，请参阅将AWS IoT Greengrass组件部署到设备。

教程：AWS IoT Greengrass V2 入门

您可以完成本入门教程来学习的基本功能AWS IoT Greengrass V2。在本教程中，您将执行以下操作：

1. 在 Linux 设备（例如 Raspberry Pi）或 Windows 设备上安装和配置AWS IoT Greengrass核心软件。该设备是 Greengrass 的核心设备。
2. 在你的 Greengrass 核心设备上开发一个 Hello World 组件。组件是在 Greengrass 核心设备上运行的软件模块。
3. 将该组件上传到AWS IoT Greengrass V2AWS Cloud。
4. 将该组件从部署AWS Cloud到您的 Greengrass 核心设备上。

> **Note**
>
> 本教程介绍如何设置开发环境和探索的功能AWS IoT Greengrass。有关如何设置和配置生产设备的更多信息，请参阅以下内容：
>
> - 设置AWS IoT Greengrass核心设备
> - 安装 AWS IoT Greengrass Core 软件

在本教程中，预计花费 20 到 30 分钟。

主题

- 先决条件
- 步骤 1：设置 AWS 账户
- 第 2 步：设置您的环境
- 步骤 3 安装AWS IoT Greengrass核心软件
- 第 4 步：在设备上开发和测试组件
- 步骤 5：在AWS IoT Greengrass服务中创建您的组件
- 步骤 6：部署您的组件
- 后续步骤

先决条件

要完成本入门教程，您需要以下条件：

- AWS 账户。如果没有，请参阅步骤 1：设置 AWS 账户。
- 使用支持AWS 区域的AWS IoT Greengrass V2。有关支持的区域列表，请参见 AWS 一般参考 中的 AWS IoT Greengrass V2 端点和配额。
- 具有管理员权限的 AWS Identity and Access Management (IAM) 用户。
- 设置为 Greengrass 核心设备的设备，例如装有 Raspberry Pi 操作系统（以前称为 Raspbian）的 Raspberry Pi 或 Windows 10 设备。您必须拥有此设备的管理员权限，或者能够获得管理员权限，例如通过sudo。此设备必须连接互联网。

  您也可以选择使用符合要求的其他设备来安装和运行 AWS IoT Greengrass Core 软件。有关更多信息，请参阅 支持的平台和要求。

  如果您的开发计算机满足这些要求，则可以在本教程中将其设置为 Greengrass 核心设备。
- 为设备上的所有用户安装了 Python 3.5 或更高版本，并已添加到PATH环境变量中。在 Windows 上，你还必须为所有用户安装适用于 Windows 的 Python 启动器。

> **Important**
>
> 在 Windows 中，默认情况下不会为所有用户安装 Python。安装 Python 时，必须对安装进行自定义，将其配置为AWS IoT Greengrass核心软件运行 Python 脚本。例如，如果您使用图形化 Python 安装程序，请执行以下操作：
>
> 1. 选择 “为所有用户安装启动器（推荐）”。
> 2. 选择Customize installation。
> 3. 选择Next。
> 4. 选择 Install for all users。
> 5. 选择 Add Python to environment variables。
> 6. 选择安装。
>
> 有关更多信息，请参阅 Python 3 文档中的在 Windows 上使用 Python。

- AWS Command Line Interface(AWS CLI) 在您的开发计算机和设备上安装并配置了凭据。请务必使用相同的AWS 区域方法AWS CLI在开发计算机和设备上进行配置。要AWS IoT Greengrass V2与一起使用AWS CLI，您必须拥有以下版本之一或更高版本：
  - AWS CLIV1 最低版本：v1.18.197
  - AWS CLIV2 最低版本：v2.1.11

Tip

你可以运行以下命令来检查你拥有AWS CLI的版本。

```
aws --version
```

有关更多信息，请参阅《AWS Command Line Interface用户指南》AWS CLI中的 “安装、更新AWS CLI和卸载” 和 “配置”。

Note

如果您使用 32 位 ARM 设备，例如带有 32 位操作系统的树莓派，请安装 AWS CLI V1。AWS CLIV2 不适用于 32 位 ARM 设备。有关更多信息，请参阅安装、更新和卸载AWS CLI 版本 1。

步骤 1：设置 AWS 账户

注册 AWS 账户

如果您还没有 AWS 账户，请完成以下步骤来创建一个。

注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

当您注册 AWS 账户时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为管理用户分配管理访问权限，并且只使用根用户执行需要根用户访问权限的任务。

注册过程完成后，AWS 会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建管理用户

注册 AWS 账户 后，保护您的 AWS 账户根用户，启用 AWS IAM Identity Center，创建一个管理用户，以避免使用根用户执行日常任务。

保护您的 AWS 账户根用户

1. 选择根用户并输入您的 AWS 账户电子邮件地址，以账户拥有者身份登录 AWS Management Console。在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的以根用户身份登录。

2. 对您的根用户启用多重身份验证（MFA）。

   有关说明，请参阅《IAM 用户指南》中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建管理用户

1. 启用 IAM Identity Center

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，为管理用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《AWS IAM Identity Center 用户指南》中的使用默认 IAM Identity Center 目录 配置用户访问权限。

作为管理用户登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

要获取使用 IAM Identity Center 用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的登录 AWS 访问门户。

第 2 步：设置您的环境

按照本节中的步骤设置要用作AWS IoT Greengrass核心设备的 Linux 或 Windows 设备。

设置 Linux 设备（树莓派）

这些步骤假设你在树莓派操作系统中使用树莓派。如果您使用其他设备或操作系统，请查阅设备的相关文档。

要设置 Raspberry Pi AWS IoT Greengrass V2

1. 在 Raspberry Pi 上启用 SSH 以远程连接到树莓派。有关更多信息，请参阅 Raspberry Pi 文档中的 SSH（安全外壳）。
2. 找到 Raspberry Pi 的 IP 地址，通过 SSH 与之连接。为此，你可以在 Raspberry Pi 上运行以下命令。

   ```
   hostname -I
   ```

3. 使用 SSH 连接到你的 Raspberry Pi。

   在您的开发计算机上，运行以下命令。将*###*替换为要登录的用户名，并*pi-ip-address*替换为在上一步中找到的 IP 地址。

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

   Important

   如果你的开发计算机使用的是早期版本的 Windows，你可能没有这个ssh命令，或者你可能有ssh但无法连接到你的 Raspberry Pi。要连接你的 Raspberry Pi，你可以安装和配置 PuTTY，这是一款免费的开源 SSH 客户端。要连接你的 Raspberr y Pi，请查阅 Putty 文档。

4. 安装 Java 运行时，AWS IoT Greengrass核心软件需要运行该运行时。在 Raspberry Pi 上，使用以下命令安装 Java 11。

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

安装完成后，运行以下命令以验证 Java 是否在你的 Raspberry Pi 上运行。

```
java -version
```

该命令会打印设备上运行的 Java 版本。输出可能与以下示例类似。

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

提示：在 Raspberry Pi 上设置内核参数

如果您的设备是 Raspberry Pi，则可以完成以下步骤来查看和更新其 Linux 内核参数：

1. 打开 /boot/cmdline.txt文件。此文件指定树莓派启动时要应用的 Linux 内核参数。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 打开文件。

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

2. 验证该/boot/cmdline.txt文件是否包含以下内核参数。该systemd.unified_cgroup_hierarchy=0参数指定使用 cgroups v1 而不是 cgroups v2。

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

   如果/boot/cmdline.txt文件不包含这些参数，或者包含这些具有不同值的参数，请更新文件以包含这些参数和值。

3. 如果您更新了/boot/cmdline.txt文件，请重新启动 Raspberry Pi 以应用更改。

   ```
   sudo reboot
   ```

设置 Linux 设备（其他）

设置 Linux 设备用于 AWS IoT Greengrass V2

1. 安装 Java 运行时，AWS IoT Greengrass核心软件需要运行该运行时。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。以下命令向您展示了如何在您的设备上安装 OpenJDK。

   - 对于基于 Debian 或基于 Ubuntua 的发行版：

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

   - 对于基于 Red Hat 的发行版：

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

   - 对于 Amazon Linux 2：

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

   - 对于亚马逊 Linux 2023：

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

   安装完成后，运行以下命令以验证 Java 是否在您的 Linux 设备上运行。

   ```
   java -version
   ```

   该命令会打印设备上运行的 Java 版本。例如，在基于 Debian 的发行版上，输出可能与以下示例类似。

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

2. （可选）创建在设备上运行组件的默认系统用户和组。您也可以选择让AWS IoT Greengrass核心软件安装程序在安装过程中使用安装程序参数创建此用户和组。`--component-default-user`有关更多信息，请参阅 安装程序参数。

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

3. 验证运行 AWS IoT Greengrass Core 软件的用户（通常`root`）是否有权`sudo`与任何用户和任何组一起运行。

   a. 运行以下命令打开该`/etc/sudoers`文件。

   ```
   sudo visudo
   ```

   b. 验证用户的权限是否如以下示例所示。

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

4. （可选）要运行容器化 Lambda 函数，必须启用 c groups v1，并且必须启用并挂载内存和设备 cgroups。如果您不打算运行容器化 Lambda 函数，则可以跳过此步骤。

   要启用这些 cgroups 选项，请使用以下 Linux 内核参数启动设备。

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

   有关查看和设置设备内核参数的信息，请参阅操作系统和启动加载程序的文档。按照说明永久设置内核参数。

5. 按照中的要求列表所示，在您的设备上安装所有其他必需的依赖项设备要求。

设置 Windows 设备

要将 Windows 设备设置为 AWS IoT Greengrass V2

1. 安装 Java 运行时，AWS IoT Greengrass核心软件需要运行该运行时。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。

2. 检查 PATH 系统变量上是否有 Java 可用，如果没有，请添加它。该 LocalSystem 帐户运行 AWS IoT Greengrass Core 软件，因此您必须将 Java 添加到 PATH 系统变量中，而不是用户的 PATH 用户变量。执行以下操作：

   a. 按下 Windows 键打开开始菜单。

   b. 键入**`environment variables`**以从 “开始” 菜单中搜索系统选项。

c. 在开始菜单搜索结果中，选择编辑系统环境变量以打开系统属性窗口。

d. 选择环境变量... 打开 “环境变量” 窗口。

e. 在 “系统变量” 下，选择 “路径”，然后选择 “编辑”。在 “编辑环境变量” 窗口中，可以在单独的行上查看每个路径。

f. 检查 Java 安装bin文件夹的路径是否存在。路径可能与以下示例类似。

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

g. 如果路径中缺少 Java 安装bin的文件夹，请选择 “新建” 将其添加，然后选择 “确定”。

3. 以管理员身份打开 Windows 命令提示符 (cmd.exe)。

4. 在 Windows 设备上的 LocalSystem 帐户中创建默认用户。将*##*替换为安全密码。

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

> **Tip**
>
> 根据你的 Windows 配置，用户的密码可能会设置为在将来的某个日期过期。为确保您的 Greengrass 应用程序继续运行，请跟踪密码何时过期，并在密码过期之前对其进行更新。您也可以将用户的密码设置为永不过期。
>
> - 要检查用户及其密码何时过期，请运行以下命令。
>
> ```
> net user ggc_user | findstr /C:expires
> ```
>
> - 要将用户的密码设置为永不过期，请运行以下命令。
>
> ```
> wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
> ```
>
> - 如果你使用的是已弃用该wmic命令的 Windows 10 或更高版本，请运行以下 PowerShell 命令。
>
> ```
> Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>  'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
> ```

5. 从微软下载该PsExec实用程序并将其安装到设备上。

6. 使用该 PsExec 实用程序将默认用户的用户名和密码存储在 LocalSystem 账户的凭据管理器实例中。将*##*替换为您之前设置的用户密码。

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

如果PsExec License Agreement打开，Accept请选择同意许可并运行命令。

> **Note**
> 在 Windows 设备上，该 LocalSystem 帐户运行 Greengrass 核心，您必须使用 PsExec 该实用程序在帐户中存储默认用户信息。 LocalSystem 使用凭据管理器应用程序将此信息存储在当前登录用户的 Windows 帐户中，而不是 LocalSystem 帐户中。

步骤 3 安装AWS IoT Greengrass核心软件

按照本节中的步骤将 Raspberry Pi 设置为可用于本地开发的AWS IoT Greengrass核心设备。在本节中，您将下载并运行安装程序，该安装程序执行以下操作，为您的设备配置AWS IoT Greengrass Core 软件：

- 安装 Greengrass 核心组件。nucleus 是必备组件，是在设备上运行AWS IoT Greengrass Core 软件的最低要求。有关更多信息，请参阅 Greengrass ights 控制组件。
- 将您的设备注册为一AWS IoT件事物，并下载允许您的设备连接的数字证书AWS。有关更多信息，请参阅AWS IoT Greengrass 的设备身份验证和授权：
- 将设备AWS IoT的事物添加到事物组，即一组或一组AWS IoT事物。事物组使您能够管理 Greengrass 核心设备队列。将软件组件部署到设备时，可以选择部署到单个设备或设备组。有关更多信息，请参阅《AWS IoT Core开发者指南》AWS IoT中的 “使用管理设备”。
- 创建 IAM 角色，允许您的 Greengrass 核心设备与AWS服务进行交互。默认情况下，此角色允许您的设备与 Amazon Logs 进行交互AWS IoT并将日志发送到 AmazonCloudWatch Logs。有关更多信息，请参阅授权核心设备与AWS服务：
- 安装AWS IoT Greengrass命令行接口 (`greengrass-cli`)，您可以使用它来测试在核心设备上开发的自定义组件。有关更多信息，请参阅Greengrass 命令行界面：

安装AWS IoT Greengrass核心软件（控制台）

1. 登录 AWS IoT Greengrass 控制台。
2. 在 Greengrass 入门下，选择设置一台核心设备。

3. 在 “步骤 1：注册 Greengrass 核心设备” 下，在 “核心设备名称” 中，输入 Greengrass 核心设备的名称。AWS IoT如果该东西不存在，则安装程序会创建它。
4. 在步骤 2：添加到事物组以应用持续部署下，对于事物组，选择要向其添加核心设备AWS IoT的事物组。

   - 如果您选择 “输入新组名”，则在事物组名称中，输入要创建的新组的名称。安装程序会为您创建新组。
   - 如果您选择 “选择现有群组”，则在 “事物组名称” 中，选择要使用的现有群组。
   - 如果您选择 “无组”，则安装程序不会将核心设备添加到事物组。
5. 在 “步骤 3：安装 Greengrass Core 软件” 下，完成以下步骤。

   a. 选择核心设备的操作系统：Linux 或 Windows。
   b. 向设备提供您的AWS证书，以便安装程序可以为您的核心设备配置AWS IoT和 IAM 资源。为了提高安全性，我们建议您为 IAM 角色获取临时证书，该证书仅允许配置所需的最低权限。有关更多信息，请参阅 安装程序配置资源的最低 IAM 政策。

      > **Note**
      >
      > 安装程序不会保存或存储您的凭据。

      在您的设备上，执行以下任一操作以检索凭证并将其提供给 AWS IoT Greengrass Core 软件安装程序：

      - （推荐）使用来自的临时凭证 AWS IAM Identity Center

        i. 提供来自 IAM 身份中心的访问密钥 ID、私有访问密钥和会话令牌。有关更多信息，请参阅 IAM Identity Center 用户指南中获取和刷新临时证书中的手动刷新凭证。
        ii. 运行以下命令为AWS IoT Greengrass核心软件提供凭据。

            Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

- 使用 IAM 角色提供的临时安全证书：

  i. 提供您代入的 IAM 角色的访问密钥 ID、私有访问密钥和会话令牌。有关如何检索这些证书的更多信息，请参阅 IAM 用户指南中的申请临时安全证书。

  ii. 运行以下命令为AWS IoT Greengrass核心软件提供凭据。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

- 使用 IAM 用户的长期证书：

i. 为您的 IAM 用户提供访问密钥 ID 和私有访问密钥。您可以创建用于预配置的 IAM 用户，稍后再将其删除。有关向用户提供的 IAM 策略，请参阅安装程序配置资源的最低 IAM 政策。有关如何检索长期证书的更多信息，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

ii. 运行以下命令为AWS IoT Greengrass核心软件提供凭据。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

iii. （可选）如果您创建了一个 IAM 用户来配置您的 Greengrass 设备，请删除该用户。

iv. （可选）如果您使用了现有 IAM 用户的访问密钥 ID 和私有访问密钥，请更新该用户的密钥，使其不再有效。有关更多信息，请参阅AWS Identity and Access Management用户指南中的更新访问密钥。

c. 在 “运行安装程序” 下，完成以下步骤。

i. 在 “下载安装程序” 下，选择 “复制”，然后在核心设备上运行复制的命令。此命令下载最新版本的 AWS IoT Greengrass Core 软件并将其解压缩到您的设备上。

ii. 在 “运行安装程序” 下，选择 “复制”，然后在核心设备上运行复制的命令。此命令使用您之前指定的事物和事物组名称来运行AWS IoT Greengrass核心软件安装程序并为核心设备设置AWS资源。AWS IoT

此命令还执行以下操作：

- 将 AWS IoT Greengrass Core 软件设置为启动时运行的系统服务。在 Linux 设备上，这需要 Systemd 初始化系统。

> **Important**
>
> 在 Windows 核心设备上，必须将AWS IoT Greengrass核心软件设置为系统服务。

- 部署 AWS IoT GreengrassCLI 组件，这是一种命令行工具，可让您在核心设备上开发自定义 Greengrass 组件。
- 指定使用`ggc_user`系统用户在核心设备上运行软件组件。在 Linux 设备上，此命令还指定使用`ggc_group`系统组，安装程序会为您创建系统用户和组。

运行此命令时，您应该会看到以下消息，表明安装程序成功了。

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

> **Note**
>
> 如果您有 Linux 设备但没有 systemd，则安装程序不会将该软件设置为系统服务，也不会看到将 nucleus 设置为系统服务的成功消息。

安装AWS IoT Greengrass核心软件 (CLI)

安装和配置 C AWS IoT Greengrass ore 软件

1. 在 Greengrass 核心设备上，运行以下命令切换到主目录。

   Linux or Unix

   ```
   cd ~
   ```

Windows Command Prompt (CMD)

```
cd %USERPROFILE%
```

PowerShell

```
cd ~
```

2. 在您的核心设备上，将 AWS IoT Greengrass Core 软件下载到名为的文件中`greengrass-nucleus-latest.zip`。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   下载此软件即表示您同意Greengrass Core 软件许可协议。

3. 将 AWS IoT Greengrass Core 软件解压缩到设备上的某个文件夹。*GreengrassInstaller*替换为要使用的文件夹。

   Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

4. 运行以下命令以启动 AWS IoT Greengrass Core 软件安装程序。此命令执行以下操作：

   - 创建核心设备运行所需的AWS资源。
   - 将 AWS IoT Greengrass Core 软件设置为启动时运行的系统服务。在 Linux 设备上，这需要 Systemd 初始化系统。

   > **Important**
   >
   > 在 Windows 核心设备上，必须将AWS IoT Greengrass核心软件设置为系统服务。

   - 部署 AWS IoT GreengrassCLI 组件，这是一种命令行工具，可让您在核心设备上开发自定义 Greengrass 组件。
   - 指定使用`ggc_user`系统用户在核心设备上运行软件组件。在 Linux 设备上，此命令还指定使用`ggc_group`系统组，安装程序会为您创建系统用户和组。

   按如下方式替换命令中的参数值。

   a. */greengrass/v2*或 *C:\greengrass\v2*：用于安装 C AWS IoT Greengrass ore 软件的根文件夹的路径。

   b. *GreengrassInstaller*。您解压缩AWS IoT Greengrass核心软件安装程序的文件夹的路径。

   c. *##*。AWS 区域在其中查找或创建资源。

   d. *MyGreengrassCore*。你的 Green AWS IoT grass 核心设备的名称。如果该东西不存在，则安装程序会创建它。安装程序下载证书以进行身份AWS IoT验证。有关更多信息，请参阅 AWS IoT Greengrass 的设备身份验证和授权。

> **Note**
>
> 事物名称不能包含冒号 (:) 字符。

e. *MyGreengrassCoreGroup*。你的 Greengrass 核心设备AWS IoT的事物组名称。如果事物组不存在，则安装程序会创建该组并将其添加到其中。如果事物组存在且部署处于活动状态，则核心设备将下载并运行部署指定的软件。

> **Note**
>
> 事物组名称不能包含冒号 (:) 字符。

f. *GreenGras ThingPolicy* sv2IoT。允许 Greengrass 核心设备与和通信的AWS IoT策略名称。AWS IoT AWS IoT Greengrass如果该AWS IoT策略不存在，则安装程序会使用此名称创建允许AWS IoT策略。您可以根据自己的用例限制此策略的权限。有关更多信息，请参阅 [AWS IoT Greengrass V2核心设备的最低AWS IoT政策]。

g. *GreenGras TokenExchangeRole* sV2。允许 Greengrass 核心设备获得临时证书的 IAM 角色的名称。AWS如果该角色不存在，则安装程序会创建该角色并创建并附加名为的策略*GreengrassV2TokenExchangeRole*Access。有关更多信息，请参阅 [授权核心设备与AWS服务]。

h. *GreengrassCoreTokenExchangeRoleAlias*。IAM 角色的别名，允许 Greengrass 核心设备稍后获得临时证书。如果角色别名不存在，则安装程序会创建该别名并将其指向您指定的 IAM 角色。有关更多信息，请参阅 [授权核心设备与AWS服务]。

Linux or Unix

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

```
  --deploy-dev-tools true
```

Windows Command Prompt (CMD)

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

PowerShell

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

> **Note**
>
> 如果您在内存有限的设备AWS IoT Greengrass上运行，则可以控制AWS IoT Greengrass 酷睿软件使用的内存量。要控制内存分配，您可以在 nucleus 组件的`jvmOptions`配置参数中设置 JVM 堆大小选项。有关更多信息，请参阅 使用 JVM 选项控制内存分配。

运行此命令时，您应该会看到以下消息，表明安装程序成功了。

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

Note

如果您有 Linux 设备但没有 systemd，则安装程序不会将该软件设置为系统服务，也不会看到将 nucleus 设置为系统服务的成功消息。

（可选）运行 Greengrass 软件 (Linux)

如果您将软件作为系统服务安装，则安装程序会为您运行该软件。否则，您必须运行该软件。要查看安装程序是否将软件设置为系统服务，请在安装程序输出中查找以下行。

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

如果您没有看到此消息，请执行以下操作来运行该软件：

1. 运行以下命令来运行该软件。

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

   如果软件成功启动，则会打印以下消息。

   ```
   Launched Nucleus successfully.
   ```

2. 必须让当前的命令外壳保持打开状态才能保持 AWS IoT Greengrass Core 软件的运行。如果您使用 SSH 连接到核心设备，请在开发计算机上运行以下命令以打开第二个 SSH 会话，您可以使用该会话在核心设备上运行其他命令。将*###*替换为要登录的用户名，然后*pi-ip-address*替换为设备的 IP 地址。

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

有关如何与 Greengrass 系统服务交互的更多信息，请参阅。将 Greengrass 核心配置为系统服务

验证设备上是否安装了 Greengrass CLI

Greengrass CLI 最多可能需要一分钟才能部署。运行以下命令以检查部署状态。*MyGreengrassCore*替换为核心设备的名称。

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

`coreDeviceExecutionStatus`表示核心设备的部署状态。当状态为时`SUCCEEDED`，运行以下命令以验证 Greengrass CLI 是否已安装并运行。*/greengrass/v2*替换为根文件夹的路径。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

该命令输出 Greengrass CLI 的帮助信息。如果`greengrass-cli`未找到，则部署可能无法安装 Greengrass CLI。有关更多信息，请参阅 故障排除 AWS IoT Greengrass V2。

您也可以运行以下命令将 AWS IoT Greengrass CLI 手动部署到您的设备。

- 将*##*替换为您 AWS 区域 使用的区域。请务必使用与 AWS 区域 在设备 AWS CLI 上配置时使用的相同。
- 将*## ID #### AWS ## # ID*。
- *MyGreengrassCore*替换为核心设备的名称。

Linux, macOS, or Unix

```
aws greengrassv2 create-deployment \
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
```

```
  --components '{
    "aws.greengrass.Cli": {
      "componentVersion": "2.12.3"
    }
  }'
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --components "{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.12.3\"}}"
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --components '{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.12.3\"}}'
```

> **Tip**
>
> 您可以在PATH环境变量中添加 */greengrass/v2*/bin (Linux) 或 *C:\greengrass\v2*\bin (Windows)，以便在没有绝对路径`greengrass-cli`的情况下运行。

AWS IoT GreengrassCore 软件和本地开发工具在您的设备上运行。接下来，你可以在你的设备上开发 Hello WorldAWS IoT Greengrass 组件。

第 4 步：在设备上开发和测试组件

组件是在 AWS IoT Greengrass 核心设备上运行的软件模块。组件使您能够将复杂应用程序作为离散的构建块来创建和管理这些应用程序，您可以将这些应用程序从一个 Greengrass 核心设备重复使用到另一个 Greengrass 核心设备。每个组件都由配方和工件组成。

- 食谱

  每个组件都包含一个用于定义其元数据的配方文件。该配方还指定了组件的配置参数、组件依赖关系、生命周期和平台兼容性。组件生命周期定义了安装、运行和关闭组件的命令。有关更多信息，请参阅AWS IoT Greengrass组件配方参考。

你可以用 JSON 或 YAML 格式定义配方。

- 构件

  组件可以有任意数量的工件，即组件二进制文件。构件可以包括脚本、编译后的代码、静态资源以及组件消耗的任何其他文件。组件还可以使用组件依赖项中的工件。

借助 AWS IoT Greengrass，您可以使用 Greengrass CLI 在 Greengrass 核心设备上本地开发和测试组件，而无需与云端交互。 AWS 完成本地组件后，您可以使用组件配方和工件在 AWS 云端的 AWS IoT Greengrass 服务中创建该组件，然后将其部署到所有 Greengrass 核心设备上。有关组件的更多信息，请参阅开发AWS IoT Greengrass组件。

在本节中，您将学习如何在核心设备上本地创建和运行基本的 Hello World 组件。

在您的设备上开发 Hello World 组件

1. 为您的组件创建一个文件夹，其中包含用于存放配方和工件的子文件夹。在 Greengrass 核心设备上运行以下命令来创建这些文件夹并切换到组件文件夹。将 *~/greengrassv2 # %USERPROFILE%\ greengrassv2 ###############*径。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

2. 使用文本编辑器创建用于定义组件元数据、参数、依赖关系、生命周期和平台功能的配方文件。在配方文件名中包含组件版本，这样您就可以确定哪个配方反映了哪个组件版本。你可以为你的食谱选择 YAML 或 JSON 格式。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

JSON

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

YAML

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

> **Note**
>
> AWS IoT Greengrass 使用组件的语义版本。语义版本遵循 major.minor.patch 编号系统。例如，版本1.0.0表示组件的第一个主要版本。有关更多信息，请参阅语义版本规范。

3. 将以下食谱粘贴到文件中。

JSON

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

```
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
  ]
}
```

YAML

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

此食谱的`ComponentConfiguration`部分定义了一个默认为的参数`world`。`Message`该`Manifests`部分定义了一个清单，它是一组适用于平台的生命周期指令和工件。例如，您可以定义多个清单，为不同的平台指定不同的安装说明。在清单中，该`Lifecycle`部分指示 Greengrass 核心设备以参数值作为参数运行 Hello World 脚本。`Message`

4. 运行以下命令为组件工件创建文件夹。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

> **Important**
>
> 必须使用以下格式作为对象文件夹路径。包括您在配方中指定的组件名称和版本。
>
> ```
> artifacts/componentName/componentVersion/
> ```

5. 使用文本编辑器为你的 Hello World 组件创建 Python 脚本构件文件。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

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

   将以下 Python 脚本复制并粘贴到文件中。

   ```
   import sys

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

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

6. 使用本地 AWS IoT Greengrass CLI 来管理 Greengrass 核心设备上的组件。

运行以下命令将组件部署到内 AWS IoT Greengrass 核。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为你的 AWS IoT Greengrass V2 根文件夹，将 *~/greengrassv2 # %USER PROFILE%\ greengrassv2* 替换为你的组件开发文件夹。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

此命令添加了在中使用配方`recipes`和中的 Python 脚本的组件`artifacts`。该`--merge`选项添加或更新您指定的组件和版本。

7. AWS IoT Greengrass Core 软件将组件进程中的 stdout 保存到文件夹中的日志文件中。`logs`运行以下命令以验证 Hello World 组件是否运行并打印消息。

Linux or Unix

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

Windows Command Prompt (CMD)

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

该type命令将文件内容写入终端。多次运行此命令以观察文件中的更改。

PowerShell

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

您应该会看到类似于以下示例的消息。

```
Hello, world!
```

Note

如果文件不存在，则本地部署可能尚未完成。如果文件在 15 秒内不存在，则部署可能失败。例如，如果您的食谱无效，则可能会发生这种情况。运行以下命令查看 AWS IoT Greengrass 核心日志文件。此文件包含来自 Greengrass 核心设备部署服务的日志。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

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

该type命令将文件内容写入终端。多次运行此命令以观察文件中的更改。

PowerShell

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

8. 修改本地组件以迭代和测试您的代码。hello_world.py在文本编辑器中打开，然后在第 4 行添加以下代码以编辑 AWS IoT Greengrass 核心记录的消息。

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

现在，该hello_world.py脚本应该包含以下内容。

```
import sys

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

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

9. 运行以下命令，使用所做的更改更新组件。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   此命令使用最新的 Hello World 工件更新`com.example.HelloWorld`组件。

10. 运行以下命令以重新启动组件。重新启动组件时，核心设备将使用最新的更改。

    Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

11. 再次检查日志，验证 Hello World 组件是否打印了新消息。

Linux or Unix

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

Windows Command Prompt (CMD)

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

该type命令将文件内容写入终端。多次运行此命令以观察文件中的更改。

PowerShell

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

您应该会看到类似于以下示例的消息。

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

12. 您可以更新组件的配置参数以测试不同的配置。部署组件时，可以指定配置更新，该更新定义了如何在核心设备上修改组件的配置。您可以指定要重置为默认值的配置值以及要合并到核心设备上的新配置值。有关更多信息，请参阅更新组件配置。

执行以下操作：

a. 使用文本编辑器创建名hello-world-config-update.json为的文件以包含配置更新

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

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

b. 将以下 JSON 对象复制并粘贴到文件中。此 JSON 对象定义了配置更新，该更新`friend`将值合并到`Message`参数以更新其值。此配置更新未指定任何要重置的值。您无需重置`Message`参数，因为合并更新会替换现有值。

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

c. 运行以下命令将配置更新部署到 Hello World 组件。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

d. 再次检查日志，验证 Hello World 组件是否输出了新消息。

Linux or Unix

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

Windows Command Prompt (CMD)

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

该type命令将文件内容写入终端。多次运行此命令以观察文件中的更改。

PowerShell

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

您应该会看到类似于以下示例的消息。

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

13. 测试完组件后，将其从核心设备中移除。运行以下命令。

    Linux or Unix

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

    Windows Command Prompt (CMD)

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

    PowerShell

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

    > **Important**
    >
    > 将组件上传到核心设备后，需要执行此步骤才能将其部署回核心设备 AWS IoT Greengrass。否则，部署会因版本兼容性错误而失败，因为本地部署指定了不同的组件版本。

运行以下命令并确认该com.example.HelloWorld组件未出现在设备上的组件列表中。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

你的 Hello World 组件已完成，你现在可以将其上传到 AWS IoT Greengrass 云服务了。然后，您可以将该组件部署到 Greengrass 核心设备上。

步骤 5：在AWS IoT Greengrass服务中创建您的组件

在核心设备上开发完组件后，可以将其上传到中的AWS IoT Greengrass服务AWS Cloud。您也可以直接在AWS IoT Greengrass控制台中创建组件。 AWS IoT Greengrass提供托管组件的组件管理服务，以便您可以将它们部署到单个设备或设备群中。要将组件上传到AWS IoT Greengrass服务，请完成以下步骤：

- 将组件项目上传到 S3 存储桶。
- 将每个工件的亚马逊简单存储服务 (Amazon S3) URI 添加到组件配方中。
- AWS IoT Greengrass从组件配方中创建组件。

在本节中，您将在 Greengrass 核心设备上完成这些步骤，将 Hello World 组件上传到该服务。AWS IoT Greengrass

在AWS IoT Greengrass（控制台）中创建您的组件

1. 在您的AWS账户中使用一个 S3 存储桶来托管AWS IoT Greengrass组件项目。当您将组件部署到核心设备时，设备会从存储桶中下载该组件的构件。

您可以使用现有的 S3 存储桶，也可以创建新的存储桶。

a. 在 Amazon S3 控制台的存储桶下，选择创建存储桶。
b. 在存储桶名称中，输入唯一的存储桶名称。例如，您可以使用 **greengrass-component-artifacts-*region*-*123456789012***。将 *123456789012* 替换为您在本AWS教程中AWS区域使用的账户 ID 和*##*。
c. 对于AWS区域，请选择您用于本教程的AWS区域。
d. 请选择创建存储桶。
e. 在 Buc kets 下，选择您创建的存储桶，然后将`hello_world.py`脚本上传到存储桶中的`artifacts/com.example.HelloWorld/1.0.0`文件夹。有关将对象上传到 S3 存储桶的信息，请参阅 Amazon 简单存储服务用户指南中的上传对象。
f. 将`hello_world.py`对象的 S3 URI 复制到 S3 存储桶中。此 URI 应与以下示例类似。将 *DOC-EXAMPLE-BUCKET ### S3 ###*的名称。

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

2. 允许核心设备访问 S3 存储桶中的组件工件。

每台核心设备都有一个核心设备 IAM 角色，允许其与云进行交互AWS IoT并将日志发送到AWS云端。默认情况下，此设备角色不允许访问 S3 存储桶，因此您必须创建并附加允许核心设备从 S3 存储桶检索组件工件的策略。

如果您的设备角色已允许访问 S3 存储桶，则可以跳过此步骤。否则，请创建允许访问的 IAM 策略并将其附加到该角色，如下所示：

a. 在 IAM 控制台导航菜单中，选择策略，然后选择创建策略。
b. 在 JSON 选项卡中，将占位符内容替换为以下策略。将 *DOC-EXAMPLE-B* UCKET 替换为包含要下载的组件工件的 S3 存储桶的名称。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
```

```
    }
  ]
}
```

c. 请选择 Next（下一步）。

d. 在 “策略详细信息” 部分中，在 “名称” 中输入**MyGreengrassV2ComponentArtifactPolicy**。

e. 选择创建策略。

f. 在 IAM 控制台导航菜单中，选择角色，然后选择核心设备的角色名称。您在安装 C AWS IoT Greengrass ore 软件时指定了此角色名称。如果您未指定名称，则默认为`GreengrassV2TokenExchangeRole`。

g. 在 “权限” 下，选择添加权限，然后选择附加策略。

h. 在添加权限页面上，选中您创建的`MyGreengrassV2ComponentArtifactPolicy`策略旁边的复选框，然后选择添加权限。

3. 使用组件配方在AWS IoT Greengrass控制台中创建组件。

a. 在AWS IoT Greengrass控制台导航菜单中，选择组件，然后选择创建组件。

b. 在 “组件信息” 下，选择 “以 JSON 格式输入配方”。占位符配方应与以下示例类似。

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

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

c. 将每个`Artifacts`部分中的占位符 URI 替换为`hello_world.py`对象的 S3 URI。

d. 选择创建组件。

e. 在 com.example 上。 HelloWorld组件页面上，验证组件的状态是否为可部署。

在 AWS IoT Greengrass (AWS CLI) 中创建你的组件

上传你的 Hello World 组件

1. 使用您的中的 S3 存储桶AWS 账户来托管AWS IoT Greengrass组件项目。当您将组件部署到核心设备时，设备会从存储桶中下载该组件的构件。

   您可以使用现有的 S3 存储桶，也可以运行以下命令来创建存储桶。此命令使用您的 AWS 账户 ID 创建一个存储桶AWS 区域，并形成一个唯一的存储桶名称。将 *123456789012* 替换为你在本教 AWS 账户程中使用AWS 区域的 ID 和*##*。

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

如果请求成功，该命令将输出以下信息。

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

2. 允许核心设备访问 S3 存储桶中的组件工件。

   每台核心设备都有一个核心设备 IAM 角色，允许其与核心设备交互AWS IoT并向其发送日志AWS Cloud。默认情况下，此设备角色不允许访问 S3 存储桶，因此您必须创建并附加允许核心设备从 S3 存储桶检索组件工件的策略。

   如果核心设备的角色已经允许访问 S3 存储桶，则可以跳过此步骤。否则，请创建允许访问的 IAM 策略并将其附加到该角色，如下所示：

   a. 创建一个名为的文件，`component-artifact-policy.json`并将以下 JSON 复制到该文件中。此策略允许访问 S3 存储桶中的所有文件。将 *DOC-EXAMPLE-BUCKET ### S3 ###*的名称。

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "s3:GetObject"
         ],
         "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
       }
     ]
   }
   ```

   b. 运行以下命令，根据中的策略文档创建策略`component-artifact-policy.json`。

   Linux or Unix

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

   Windows Command Prompt (CMD)

   ```
   aws iam create-policy ^
   ```

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

PowerShell

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

从输出中的策略元数据中复制策略 Amazon 资源名称 (ARN)。在下一步中，您将使用此 ARN 将此策略附加到核心设备角色。

c. 运行以下命令将策略附加到核心设备角色。将 *GreenGrassV2 TokenExchangeRole* 替换为核心设备的角色名称。您在安装 C AWS IoT Greengrass ore 软件时指定了此角色名称。将策略 ARN 替换为上一步中的 ARN。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

如果命令没有输出，则表示成功了。核心设备现在可以访问您上传到此 S3 存储桶的项目。

3. 将 Hello World Python 脚本构件上传到 S3 存储桶。

运行以下命令将脚本上传到AWS IoT Greengrass核心上存在脚本的存储桶中的相同路径。将 *DOC-EXAMPLE-BUCKET ### S3 ###*的名称。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

`upload:`如果请求成功，该命令将输出以开头的行。

4. 将构件的 Amazon S3 URI 添加到组件配方中。

   Amazon S3 URI 由存储桶名称和存储桶中项目对象的路径组成。您的脚本构件的 Amazon S3 URI 就是您在上一步中将构件上传到的 URI。此 URI 应与以下示例类似。将 *DOC-EXAMPLE-BUCKET ### S3 ###*的名称。

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

   要将构件添加到配方中，请添加一个`Artifacts`包含带有 Amazon S3 URI 的结构的列表。

   JSON

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

```
  }
]
```

在文本编辑器中打开配方文件。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

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

将神器添加到配方中。您的配方文件应与以下示例类似。

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

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

YAML

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

在文本编辑器中打开配方文件。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

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

将神器添加到配方中。您的配方文件应与以下示例类似。

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
```

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

5. AWS IoT Greengrass从配方中创建组件资源。运行以下命令根据配方创建组件，该配方以二进制文件形式提供。

   JSON

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

   YAML

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

   如果请求成功，则响应类似于以下示例。

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

arn从输出中复制，以便在下一步中检查组件的状态。

Note

你还可以在AWS IoT Greengrass控制台的 “组件” 页面上看到你的 Hello World 组件。

6. 验证组件是否已创建并已准备好部署。创建组件时，其状态为REQUESTED。然后，AWS IoT Greengrass验证该组件是否可部署。您可以运行以下命令来查询组件状态并验证您的组件是否可部署。将arn替换为上一步中的 ARN。

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

如果组件通过验证，则响应表明组件状态为DEPLOYABLE。

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

你的 Hello World 组件现已在中可用AWS IoT Greengrass。您可以将其部署回此 Greengrass 核心设备或其他核心设备。

步骤 6：部署您的组件

使用AWS IoT Greengrass，您可以将组件部署到单个设备或设备组。部署组件时，AWS IoT Greengrass会在每台目标设备上安装并运行该组件的软件。您可以为每个组件指定要部署的组件以及要部署的配置更新。您还可以控制部署部署如何部署到部署目标设备。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。

在本节中，您将你的 Hello World 组件部署回你的 Greengrass 核心设备。

部署您的组件（控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择组件。
2. 在 “组件” 页面的 “我的组件” 选项卡上，选择com.example.HelloWorld。
3. 在 com.example.HelloWorld 页面上，选择部署。
4. 从 “添加到部署” 中，选择 “创建新部署”，然后选择 “下一步”。
5. 在指定目标页面中，执行以下操作：
   a. 在名称框中，输入 **`Deployment for MyGreengrassCore`**。
   b. 对于部署目标，选择核心设备和AWS IoT核心设备的名称。本教程中的默认值为*`MyGreengrassCore`*。
   c. 请选择 Next（下一步）。
6. 在 “选择组件” 页面的 “我的组件” 下，确认已选择该com.example.HelloWorld组件，然后选择 “下一步”。
7. 在配置组件页面上 com.example.HelloWorld，选择并执行以下操作：
   a. 选择配置组件。
   b. 在配置更新下的要合并的配置中，输入以下配置。

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

   此配置更新将此部署中设备的 Hello World `Message` 参数设置为。`universe`

   c. 选择确认。
   d. 请选择 Next（下一步）。
8. 在配置高级设置页面上，保留默认配置设置，然后选择下一步。

9. 在 Review（检查）页上，选择 Deploy（部署）。
10. 验证部署是否成功完成。完成部署可能需要数分钟。查看 Hello World 日志以验证更改。在你的 Greengrass 核心设备上运行以下命令。

    Linux or Unix

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

    Windows Command Prompt (CMD)

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

    PowerShell

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

    您应该会看到类似于以下示例的消息。

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

    Note

    如果日志消息未更改，则表示部署失败或未到达核心设备。如果您的核心设备未连接到互联网或无权从 S3 存储桶中检索项目，则可能会发生这种情况。在核心设备上运行以下命令以查看AWS IoT Greengrass核心软件日志文件。此文件包含来自 Greengrass 核心设备部署服务的日志。

    Linux or Unix

    ```
    sudo tail -f /greengrass/v2/logs/greengrass.log
    ```

    Windows Command Prompt (CMD)

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

    该`type`命令将文件内容写入终端。多次运行此命令以观察文件中的更改。

PowerShell

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

有关更多信息，请参阅 故障排除 AWS IoT Greengrass V2。

部署您的组件 (AWS CLI)

部署你的 Hello World 组件

1. 在您的开发计算机上，创建一个名为的文件，hello-world-deployment.json并将以下 JSON 复制到该文件中。此文件定义了要部署的组件和配置。

```
{
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"Message\":\"universe\"}"
      }
    }
  }
}
```

   此配置文件指定部署您在之前1.0.0的过程中开发和发布的 Hello World 组件的版本。configurationUpdate指定将组件配置合并为 JSON 编码的字符串。此配置更新将此部署中设备的 Hello World Message 参数设置为。universe

2. 运行以下命令将该组件部署到您的 Greengrass 核心设备上。您可以部署到事物（即单个设备）或事物组（即设备组）。*MyGreengrassCore*替换为AWS IoT核心设备的名称。

   Linux or Unix

```
aws greengrassv2 create-deployment \
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
  --cli-input-json file://hello-world-deployment.json
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --cli-input-json file://hello-world-deployment.json
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --cli-input-json file://hello-world-deployment.json
```

该命令输出类似于以下示例的响应。

```
{
  "deploymentId": "deb69c37-314a-4369-a6a1-3dff9fce73a9",
  "iotJobId": "b5d92151-6348-4941-8603-bdbfb3e02b75",
  "iotJobArn": "arn:aws:iot:region:account-id:job/b5d92151-6348-4941-8603-
bdbfb3e02b75"
}
```

3. 验证部署是否成功完成。完成部署可能需要数分钟。查看 Hello World 日志以验证更改。在你的 Greengrass 核心设备上运行以下命令。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   您应该会看到类似于以下示例的消息。

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

> **Note**
>
> 如果日志消息未更改，则表示部署失败或未到达核心设备。如果您的核心设备未连接到互联网或无权从 S3 存储桶中检索项目，则可能会发生这种情况。在核心设备上运行以下命令以查看AWS IoT Greengrass核心软件日志文件。此文件包含来自 Greengrass 核心设备部署服务的日志。
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\\logs\\greengrass.log
> ```
>
> 该type命令将文件内容写入终端。多次运行此命令以观察文件中的更改。
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
> ```
>
> 有关更多信息，请参阅 故障排除 AWS IoT Greengrass V2。

后续步骤

你已经完成了本教程。AWS IoT GreengrassCore 软件和你的 Hello World 组件在你的设备上运行。此外，您的 Hello World 组件可在AWS IoT Greengrass云服务中部署到其他设备。有关本教程中探讨主题的更多信息，请参阅下文：

- 创建 AWS IoT Greengrass 组件
- 发布组件以部署到您的核心设备
- 将AWS IoT Greengrass组件部署到设备

设置AWS IoT Greengrass核心设备

完成本节中的任务以安装、配置和运行AWS IoT Greengrass核心软件。

> **Note**
>
> 本节介绍AWS IoT Greengrass核心软件的高级安装和配置。如果您是首次使用AWS IoT Greengrass V2，我们建议您先完成入门教程，设置核心设备并探索的AWS IoT Greengrass功能。

支持的平台和要求

在开始之前，请确保满足以下要求才能安装和运行 AWS IoT Greengrass Core 软件。

> **Tip**
>
> 您可以在AWS合作伙伴设备目录AWS IoT Greengrass V2中搜索符合条件的设备。

主题

- 支持的平台
- 设备要求
- Lambda 函数要求

支持的平台

AWS IoT Greengrass正式支持运行以下平台的设备。平台未包含在此列表中的设备可能可以运行，但只能在这些指定平台上AWS IoT Greengrass进行测试。

Linux

架构：

- Armv7l

- Armv8 (AArch64)
- x86_64

Windows

架构：

- x86_64

版本：

- Windows 10
- Windows 11
- Windows Server 2019
- Windows Server 2022

> **Note**
>
> Windows 设备目前不支持某些AWS IoT Greengrass功能。有关更多信息，请参阅 按操作系统划分的 Greengrass 功能兼容性和 Windows 设备的功能注意事项。

Linux 平台也可以AWS IoT Greengrass V2在 Docker 容器中运行。有关更多信息，请参阅 在 Docker 容器中运行 AWS IoT Greengrass 核心软件。

要构建基于 Linux 的自定义操作系统，可以在项目AWS IoT Greengrass V2中使用 BitBake 配方。`meta-aws`该`meta-aws`项目提供了一些方法，可用于在使用Yocto Project构建AWS框架OpenEmbedded和Yocto Project构建框架构建的嵌入式Linux 系统中构建边缘软件功能。Yocto Project 是一个开源协作项目，无论硬件架构如何，它都能帮助您为嵌入式应用程序构建基于 Linux 的自定义系统。在您的设备上AWS IoT Greengrass V2安装、配置和自动运行 AWS IoT Greengrass Core 软件的 BitBake 秘诀。

设备要求

设备必须满足以下要求才能安装和运行 AWS IoT Greengrass Core 软件 v2.x。

Note

您可以使用AWS IoT Device TesterAWS IoT Greengrass来验证您的设备是否可以运行 AWS IoT Greengrass Core 软件并与通信AWS Cloud。有关更多信息，请参阅 用AWS IoT Device Tester于 AWS IoT Greengrass V2。

Linux

- 使用支持AWS 区域的AWS IoT Greengrass V2。有关支持的区域列表，请参见 AWS 一般参考中的 AWS IoT Greengrass V2 端点和配额。
- 至少 256 MB 磁盘空间可供AWS IoT Greengrass酷睿软件使用。此要求不包括部署到核心设备的组件。
- 分配给AWS IoT Greengrass核心软件的最少 96 MB 内存。此要求不包括在核心设备上运行的组件。有关更多信息，请参阅 使用 JVM 选项控制内存分配。
- Java 运行时环境 (JRE) 版本 8 或更高版本。Java 必须在设备上的 PATH 环境变量上可用。要使用 Java 开发自定义组件，您必须安装 Java 开发工具包 (JDK)。我们建议您使用 Amazon Corretto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。
- GNU C 库 (glibc) 版本 2.25 或更高版本。
- 您必须以 root 用户身份运行 AWS IoT Greengrass Core 软件。例如`sudo`，使用。
- 运行 AWS IoT Greengrass Core 软件（例如`root`）的 root 用户必须具有`sudo`与任何用户和任何组一起运行的权限。该`/etc/sudoers`文件必须授予该用户以其他组`sudo`身份运行的权限。中用户的权限`/etc/sudoers`应类似于以下示例。

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

- 核心设备必须能够对一组端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。
- 必须使用`exec`权限装入该`/tmp`目录。
- 以下所有 shell 命令：
  - `ps -ax -o pid,ppid`
  - `sudo`
  - `sh`
  - `kill`
  - `cp`

  - chmod
  - rm
  - ln
  - echo
  - exit
  - id
  - uname
  - grep
- 您的设备可能还需要以下可选的 shell 命令：
  - （可选）systemctl。此命令用于将 AWS IoT Greengrass Core 软件设置为系统服务。
  - （可选）useraddgroupadd、和usermod。这些命令用于设置ggc_user系统用户和ggc_group系统组。
  - （可选）mkfifo。此命令用于将 Lambda 函数作为组件运行。
- 要为组件进程配置系统资源限制，您的设备必须运行 Linux 内核版本 2.6.24 或更高版本。
- 要运行 Lambda 函数，您的设备必须满足其他要求。有关更多信息，请参阅 Lambda 函数要求。

Windows

- 使用支持AWS 区域的AWS IoT Greengrass V2。有关支持的区域列表，请参见 AWS 一般参考中的 AWS IoT Greengrass V2 端点和配额。
- 至少 256 MB 磁盘空间可供AWS IoT Greengrass酷睿软件使用。此要求不包括部署到核心设备的组件。
- 分配给AWS IoT Greengrass核心软件的最低 160 MB 内存。此要求不包括在核心设备上运行的组件。有关更多信息，请参阅 使用 JVM 选项控制内存分配。
- Java 运行时环境 (JRE) 版本 8 或更高版本。Java 必须在设备上的 PATH 系统变量上可用。要使用 Java 开发自定义组件，您必须安装 Java 开发工具包 (JDK)。我们建议您使用 Amazon Corretto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。

  > **Note**
  >
  > 要使用 Greengrass nucleus 的 2.5.0 版本，必须使用 64 位版本的 Java 运行时环境 (JRE)。Greengrass nucleus 版本 2.5.1 支持 32 位和 64 位 JRE。

- 安装 AWS IoT Greengrass Core 软件的用户必须是管理员。
- 您必须将 AWS IoT Greengrass Core 软件作为系统服务进行安装。指定`--setup-system-service true`何时安装软件。
- 每个运行组件进程的用户都必须存在于 LocalSystem 账户中，并且该用户的名字和密码必须位于该 LocalSystem 账户的 Credential Manager 实例中。当你按照说明安装 C AWS IoT Greengrass ore 软件时，你可以设置这个用户。
- 核心设备必须能够对一组端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Lambda 函数要求

您的设备必须满足以下要求才能运行 Lambda 函数：

- 基于 Linux 的操作系统。
- 您的设备必须有 `mkfifo` shell 命令。
- 您的设备必须运行 Lambda 函数所需的编程语言库。您必须在设备上安装所需的库并将其添加到PATH环境变量中。Greengrass 支持所有 Lambda 支持的 Python、Node.js 和 Java 运行时版本。Greengrass 不对已弃用的 Lambda 运行时版本施加任何额外限制。有关 AWS IoT Greengrass 对 Lambda 运行时的支持的更多信息，请参阅运行AWS Lambda函数。
- 要运行容器化 Lambda 函数，您的设备必须满足以下要求：
  - Linux 内核 4.4 或更高版本。
  - 内核必须支持 c groups v1，并且必须启用并挂载以下 cgroups：
    - 用于为AWS IoT Greengrass容器化 Lambda 函数设置内存限制的内存 cgroup。
    - 设备 cg roup 用于容器化 Lambda 函数来访问系统设备或卷。

  AWS IoT Greengrass核心软件不支持 cgroups v2。

  要满足此要求，请使用以下 Linux 内核参数启动设备。

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

  > **Tip**
  > 在 Raspberry Pi 上，编辑`/boot/cmdline.txt`文件以设置设备的内核参数。

- 您必须在设备上启用以下 Linux 内核配置：
  - 命名空间：
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Cgroups：
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG
  - 其他：
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM

Tip

查看 Linux 发行版的文档，了解如何验证和设置 Linux 内核参数。您也可以使用AWS IoT Device TesterAWS IoT Greengrass来验证您的设备是否满足这些要求。有关更多信息，请参阅 用AWS IoT Device Tester于 AWS IoT Greengrass V2。

Windows 设备的功能注意事项

Windows 设备目前不支持某些AWS IoT Greengrass功能。查看功能差异以确认 Windows 设备是否满足您的要求。有关更多信息，请参阅 按操作系统划分的 Greengrass 功能兼容性。

设置一个 AWS 账户

如果您还没有 AWS 账户，请完成以下步骤来创建一个。

注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册 AWS 账户时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为管理用户分配管理访问权限，并且只使用根用户执行需要根用户访问权限的任务。

要创建管理员用户，请选择以下选项之一。

选择一种方法来管理您的管理员	目的	方式	您也可以
在 IAM Identity Center 中 （建议）	使用短期凭证访问 AWS。 这符合安全最佳实操。有关最佳实践的信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。	有关说明，请参阅《AWS IAM Identity Center 用户指南》中的入门。	按照《AWS Command Line Interface 用户指南》中的配置 AWS CLI 以使用 AWS IAM Identity Center，配置编程式访问。
在 IAM 中 （不推荐使用）	使用长期凭证访问 AWS。	按照《IAM 用户指南》中的创建您的首个 IAM 管理员用户和组的说明操作。	按照《IAM 用户指南》中的管理 IAM 用户的访问密钥，配置编程式访问。

安装 AWS IoT Greengrass Core 软件

AWS IoT Greengrass扩展AWS到边缘设备，以便它们可以根据自己生成的数据采取行动，同时将其AWS Cloud用于管理、分析和持久存储。在边缘设备上安装 AWS IoT Greengrass Core 软件以AWS IoT Greengrass与之集成，AWS Cloud.

> **Important**
>
> 在下载并安装 AWS IoT Greengrass Core 软件之前，请检查您的核心设备是否满足安装和运行C AWS IoT Greengrass ore 软件 v2.0 的要求。

AWS IoT Greengrass核心软件包括一个安装程序，可将您的设备设置为 Greengrass 核心设备。运行安装程序时，您可以配置选项，例如根文件夹和AWS 区域要使用的。您可以选择让安装程序为您创建必需资源AWS IoT和 IAM 资源。您也可以选择部署本地开发工具来配置用于自定义组件开发的设备。

C AWS IoT Greengrass ore 软件需要以下资源AWS IoT和 IAM 资源才能连接到AWS Cloud并运行：

- AWS IoT 事物。当您将设备注册为AWS IoT事物时，该设备可以使用数字证书进行身份验证AWS。此证书允许设备与AWS IoT和通信AWS IoT Greengrass。有关更多信息，请参阅 AWS IoT Greengrass 的设备身份验证和授权。
- （可选）AWS IoT事物组。您可以使用事物组来管理 Greengrass 核心设备群。将软件组件部署到设备时，可以选择部署到单个设备或设备组。您可以将设备添加到事物组，以将该事物组的软件组件部署到该设备上。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。
- IAM 角色。Greengrass 核心设备使用凭证提供程序授权AWS IoT Core对具有 IAM 角色的服务的调用AWS。此角色允许您的设备与AWS IoT亚马逊简单存储服务 (Amazon S3) Service 进行交互，向亚马逊 CloudWatch 日志发送日志，以及从亚马逊简单存储服务 (Amazon S3) 下载自定义组件项目。有关更多信息，请参阅 授权核心设备与AWS服务。
- AWS IoT角色别名。Greengrass 核心设备使用角色别名来标识要使用的 IAM 角色。角色别名允许您更改 IAM 角色，但设备配置保持不变。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的授权直接调用AWS服务。

选择以下选项之一，在您的AWS IoT Greengrass核心设备上安装 Core 软件。

- 快速安装

  选择此选项，只需尽可能少的步骤即可设置 Greengrass 核心设备。安装程序会为您创建所需的资源AWS IoT和 IAM 资源。此选项要求您向安装程序提供AWS凭据才能在中创建资源AWS 账户。

您不能使用此选项在防火墙或网络代理后面安装。如果您的设备位于防火墙或网络代理之后，请考虑手动安装。

有关更多信息，请参阅 安装具有自动资源配置功能的 AWS IoT Greengrass Core 软件。

- 手动安装

  选择此选项可手动创建所需的AWS资源或安装在防火墙或网络代理后面。通过使用手动安装，您无需向安装程序授予在中创建资源的权限AWS 账户，因为您可以创建所需的资源AWS IoT和IAM 资源。您也可以将设备配置为通过端口 443 或通过网络代理进行连接。您还可以将 AWS IoT Greengrass Core 软件配置为使用存储在硬件安全模块 (HSM)、可信平台模块 (TPM) 或其他加密元素中的私钥和证书。

  有关更多信息，请参阅 使用手动资源配置来安装 AWS IoT Greengrass Core 软件。

- 使用AWS IoT队列配置进行安装

  选择此选项可根据AWS IoT队列配置模板创建所需的AWS资源。您可以选择此选项在车队中创建类似的设备，或者制造客户稍后激活的设备，例如车辆或智能家居设备。设备使用声明证书来验证和配置AWS资源，包括设备AWS Cloud用于连接以进行正常操作的 X.509 客户端证书。在制造过程中，您可以将索赔证书嵌入或闪存到设备的硬件中，也可以使用相同的索赔证书和密钥来配置多个设备。您也可以将设备配置为通过端口 443 或通过网络代理进行连接。

  有关更多信息，请参阅 安装具有 AWS IoT 队列配置功能的 C AWS IoT Greengrass ore 软件。

- 使用自定义配置进行安装

  选择此选项可开发预置所需AWS资源的自定义 Java 应用程序。如果您创建自己的 X.509 客户端证书，或者想要更好地控制配置过程，则可以选择此选项。 AWS IoT Greengrass提供了一个接口，您可以实现该接口，以便在自定义配置应用程序和AWS IoT Greengrass核心软件安装程序之间交换信息。

  有关更多信息，请参阅 安装具有自定义资源配置功能的 C AWS IoT Greengrass ore 软件。

AWS IoT Greengrass 还提供了运行 AWS IoT Greengrass Core 软件的容器化环境。你可以使用 Dockerfile 在 Dock er 容AWS IoT Greengrass器中运行。

主题

- 安装具有自动资源配置功能的 AWS IoT Greengrass Core 软件
- 使用手动资源配置来安装 AWS IoT Greengrass Core 软件

- 安装具有 AWS IoT 队列配置功能的 C AWS IoT Greengrass ore 软件
- 安装具有自定义资源配置功能的 C AWS IoT Greengrass ore 软件
- 安装程序参数

安装具有自动资源配置功能的 AWS IoT Greengrass Core 软件

AWS IoT Greengrass核心软件包括一个安装程序，可将您的设备设置为 Greengrass 核心设备。要快速设置设备，安装程序可以预配置核心设备运行所需AWS IoT的事物、事物组、IAM AWS IoT 角色和角色别名。AWS IoT安装程序还可以将本地开发工具部署到核心设备，因此您可以使用该设备开发和测试自定义软件组件。安装程序需要AWS凭据才能配置这些资源和创建部署。

如果您无法向设备提供AWS凭据，则可以配置核心设备运行所需的AWS资源。您也可以将开发工具部署到核心设备上，用作开发设备。这使您能够在运行安装程序时向设备提供更少的权限。有关更多信息，请参阅 使用手动资源配置来安装 AWS IoT Greengrass Core 软件。

Important

在下载AWS IoT Greengrass核心软件之前，请检查您的核心设备是否满足安装和运行AWS IoT Greengrass酷睿软件 v2.0 的要求。

主题

- 设置设备环境
- 为设备提供AWS凭证
- 下载AWS IoT Greengrass核心软件
- 安装 AWS IoT Greengrass Core 软件

设置设备环境

按照本节中的步骤设置要用作AWS IoT Greengrass核心设备的 Linux 或 Windows 设备。

设置 Linux 设备

设置 Linux 设备用于 AWS IoT Greengrass V2

1. 安装 Java 运行时，AWS IoT Greengrass核心软件需要运行该运行时。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。以下命令向您展示了如何在您的设备上安装 OpenJDK。

   - 对于基于 Debian 或基于 Ubuntua 的发行版：

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

   - 对于基于 Red Hat 的发行版：

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

   - 对于 Amazon Linux 2：

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

   - 对于亚马逊 Linux 2023：

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

   安装完成后，运行以下命令以验证 Java 是否在您的 Linux 设备上运行。

   ```
   java -version
   ```

   该命令会打印设备上运行的 Java 版本。例如，在基于 Debian 的发行版上，输出可能与以下示例类似。

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

2. （可选）创建在设备上运行组件的默认系统用户和组。您也可以选择让AWS IoT Greengrass核心软件安装程序在安装过程中使用安装程序参数创建此用户和组。`--component-default-user`有关更多信息，请参阅 安装程序参数。

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

3. 验证运行 AWS IoT Greengrass Core 软件的用户（通常`root`）是否有权`sudo`与任何用户和任何组一起运行。

   a. 运行以下命令打开该`/etc/sudoers`文件。

   ```
   sudo visudo
   ```

   b. 验证用户的权限是否如以下示例所示。

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

4. （可选）要运行容器化 Lambda 函数，必须启用 c groups v1，并且必须启用并挂载内存和设备 cgroups。如果您不打算运行容器化 Lambda 函数，则可以跳过此步骤。

   要启用这些 cgroups 选项，请使用以下 Linux 内核参数启动设备。

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

   有关查看和设置设备内核参数的信息，请参阅操作系统和启动加载程序的文档。按照说明永久设置内核参数。

5. 按照中的要求列表所示，在您的设备上安装所有其他必需的依赖项设备要求。

设置 Windows 设备

> **Note**
>
> 此功能适用于 Greengrass nucleus 组件的 2.5.0 及更高版本。

要将 Windows 设备设置为 AWS IoT Greengrass V2

1. 安装 Java 运行时，AWS IoT Greengrass核心软件需要运行该运行时。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。

2. 检查 PATH 系统变量上是否有 Java 可用，如果没有，请添加它。该 LocalSystem 帐户运行 AWS IoT Greengrass Core 软件，因此您必须将 Java 添加到 PATH 系统变量中，而不是用户的 PATH 用户变量。执行以下操作：

   a. 按下 Windows 键打开开始菜单。

   b. 键入**environment variables**以从 “开始” 菜单中搜索系统选项。

   c. 在开始菜单搜索结果中，选择编辑系统环境变量以打开系统属性窗口。

   d. 选择环境变量... 打开 “环境变量” 窗口。

   e. 在 “系统变量” 下，选择 “路径”，然后选择 “编辑”。在 “编辑环境变量” 窗口中，可以在单独的行上查看每个路径。

   f. 检查 Java 安装bin文件夹的路径是否存在。路径可能与以下示例类似。

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

   g. 如果路径中缺少 Java 安装bin的文件夹，请选择 “新建” 将其添加，然后选择 “确定”。

3. 以管理员身份打开 Windows 命令提示符 (cmd.exe)。

4. 在 Windows 设备上的 LocalSystem 帐户中创建默认用户。将*##*替换为安全密码。

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

   > **Tip**
   >
   > 根据你的 Windows 配置，用户的密码可能会设置为在将来的某个日期过期。为确保您的 Greengrass 应用程序继续运行，请跟踪密码何时过期，并在密码过期之前对其进行更新。您也可以将用户的密码设置为永不过期。
   >
   > - 要检查用户及其密码何时过期，请运行以下命令。
   >
   >   ```
   >   net user ggc_user | findstr /C:expires
   >   ```
   >
   > - 要将用户的密码设置为永不过期，请运行以下命令。
   >
   >   ```
   >   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
   >   ```
   >
   > - 如果你使用的是已弃用该wmic命令的 Windows 10 或更高版本，请运行以下 PowerShell 命令。

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

5. 从微软下载该PsExec实用程序并将其安装到设备上。
6. 使用该 PsExec 实用程序将默认用户的用户名和密码存储在 LocalSystem 账户的凭据管理器实例中。将##替换为您之前设置的用户密码。

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

如果PsExec License Agreement打开，Accept请选择同意许可并运行命令。

Note

在 Windows 设备上，该 LocalSystem 帐户运行 Greengrass 核心，您必须使用 PsExec 该实用程序在帐户中存储默认用户信息。 LocalSystem 使用凭据管理器应用程序将此信息存储在当前登录用户的 Windows 帐户中，而不是 LocalSystem 帐户中。

为设备提供AWS凭证

向设备提供您的AWS凭据，以便安装程序可以配置所需的AWS资源。有关所需权限的更多信息，请参阅 安装程序配置资源的最低 IAM 政策。

为设备提供AWS凭证

- 向设备提供您的AWS证书，以便安装程序可以为您的核心设备配置AWS IoT和 IAM 资源。为了提高安全性，我们建议您为 IAM 角色获取临时证书，该证书仅允许配置所需的最低权限。有关更多信息，请参阅 安装程序配置资源的最低 IAM 政策。

Note

安装程序不会保存或存储您的凭据。

在您的设备上，执行以下任一操作以检索凭证并将其提供给 AWS IoT Greengrass Core 软件安装程序：

- （推荐）使用来自的临时凭证 AWS IAM Identity Center

a. 提供来自 IAM 身份中心的访问密钥 ID、私有访问密钥和会话令牌。有关更多信息，请参阅 IAM Identity Center 用户指南中获取和刷新临时证书中的手动刷新凭证。

b. 运行以下命令为AWS IoT Greengrass核心软件提供凭据。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

- 使用 IAM 角色提供的临时安全证书：

  a. 提供您代入的 IAM 角色的访问密钥 ID、私有访问密钥和会话令牌。有关如何检索这些证书的更多信息，请参阅 IAM 用户指南中的申请临时安全证书。

  b. 运行以下命令为AWS IoT Greengrass核心软件提供凭据。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

- 使用 IAM 用户的长期证书：

  a. 为您的 IAM 用户提供访问密钥 ID 和私有访问密钥。您可以创建用于预配置的 IAM 用户，稍后再将其删除。有关向用户提供的 IAM 策略，请参阅安装程序配置资源的最低 IAM 政策。有关如何检索长期证书的更多信息，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

  b. 运行以下命令为AWS IoT Greengrass核心软件提供凭据。

  Linux or Unix

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

  Windows Command Prompt (CMD)

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

  PowerShell

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

  c. （可选）如果您创建了一个 IAM 用户来配置您的 Greengrass 设备，请删除该用户。

  d. （可选）如果您使用了现有 IAM 用户的访问密钥 ID 和私有访问密钥，请更新该用户的密钥，使其不再有效。有关更多信息，请参阅AWS Identity and Access Management用户指南中的更新访问密钥。

下载AWS IoT Greengrass核心软件

您可以从以下位置下载最新版本的 AWS IoT Greengrass Core 软件：

- https://d2s8p88vqu9w66.cloudfront.net/releases/ greengrass-nucleus-latest .zip

> **Note**
>
> 您可以从以下位置下载特定版本的 AWS IoT Greengrass Core 软件。将*##*替换为要下载的版本。
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

下载AWS IoT Greengrass核心软件

1. 在您的核心设备上，将 AWS IoT Greengrass Core 软件下载到名为的文件中`greengrass-nucleus-latest.zip`。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   下载此软件即表示您同意Greengrass Core 软件许可协议。

2. （可选）验证 Greengrass nucleus 软件签名

   > **Note**
   >
   > 此功能在 Greengrass nucleus 版本 2.9.5 及更高版本中可用。

   a. 使用以下命令验证你的 Greengrass 核工件的签名：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

根据您安装的 JDK 版本，文件名可能有所不同。*jdk17.0.6_10*替换为您安装的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

根据您安装的 JDK 版本，文件名可能有所不同。*jdk17.0.6_10*替换为您安装的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. 该`jarsigner`调用会产生指示验证结果的输出。

   i. 如果 Greengrass nucleus zip 文件已签名，则输出将包含以下语句：

   ```
   jar verified.
   ```

   ii. 如果 Greengrass nucleus zip 文件未签名，则输出将包含以下语句：

   ```
   jar is unsigned.
   ```

c. 如果您提供了 Jarsigner `-certs` 选项以及`-verify`和`-verbose`选项，则输出还包括详细的签名者证书信息。

3. 将 AWS IoT Greengrass Core 软件解压缩到设备上的某个文件夹。*GreengrassInstaller*替换为要使用的文件夹。

   Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

4. （可选）运行以下命令以查看 AWS IoT Greengrass Core 软件的版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> 如果您安装了 v2.4.0 之前的 Greengrass nucleus 版本，则在安装 Core 软件后请勿删除此文件夹。AWS IoT GreengrassC AWS IoT Greengrass ore 软件使用此文件夹中的文件来运行。如果您下载的是最新版本的软件，则需要安装 v2.4.0 或更高版本，并且可以在安装 C AWS IoT Greengrass ore 软件后删除此文件夹。

安装 AWS IoT Greengrass Core 软件

使用指定执行以下操作的参数运行安装程序：

- 创建核心设备运行所需的AWS资源。
- 指定使用`ggc_user`系统用户在核心设备上运行软件组件。在 Linux 设备上，此命令还指定使用`ggc_group`系统组，安装程序会为您创建系统用户和组。
- 将 AWS IoT Greengrass Core 软件设置为启动时运行的系统服务。在 Linux 设备上，这需要 Systemd 初始化系统。

> **Important**
>
> 在 Windows 核心设备上，必须将AWS IoT Greengrass核心软件设置为系统服务。

要使用本地开发工具设置开发设备，请指定`--deploy-dev-tools true`参数。安装完成后，部署本地开发工具最多可能需要一分钟。

有关您可以指定的参数的更多信息，请参阅安装程序参数。

> **Note**
>
> 如果您在内存有限的设备AWS IoT Greengrass上运行，则可以控制AWS IoT Greengrass酷睿软件使用的内存量。要控制内存分配，您可以在 nucleus 组件的`jvmOptions`配置参数中设置 JVM 堆大小选项。有关更多信息，请参阅 使用 JVM 选项控制内存分配。

安装 AWS IoT Greengrass Core 软件

1. 运行AWS IoT Greengrass核心安装程序。按如下方式替换命令中的参数值。

   a. */greengrass/v2*或 *C:\greengrass\v2*：用于安装AWS IoT Greengrass核心软件的根文件夹的路径。

   b. *GreengrassInstaller*。您解压缩AWS IoT Greengrass核心软件安装程序的文件夹的路径。

   c. *##*。AWS 区域在其中查找或创建资源。

   d. *MyGreengrassCore*。你的 Green AWS IoT grass 核心设备的名称。如果该东西不存在，则安装程序会创建它。安装程序下载证书以进行身份AWS IoT验证。有关更多信息，请参阅AWS IoT Greengrass 的设备身份验证和授权。

   > **Note**
   >
   > 事物名称不能包含冒号 (`:`) 字符。

   e. *MyGreengrassCoreGroup*。你的 Greengrass 核心设备AWS IoT的事物组名称。如果事物组不存在，则安装程序会创建该组并将其添加到其中。如果事物组存在且部署处于活动状态，则核心设备将下载并运行部署指定的软件。

   > **Note**
   >
   > 事物组名称不能包含冒号 (`:`) 字符。

f. *GreenGras ThingPolicy* sv2IoT。允许 Greengrass 核心设备与和通信的AWS IoT策略名称。AWS IoT AWS IoT Greengrass如果该AWS IoT策略不存在，则安装程序会使用此名称创建允许AWS IoT策略。您可以根据自己的用例限制此策略的权限。有关更多信息，请参阅 AWS IoT Greengrass V2核心设备的最低AWS IoT政策。

g. *GreenGras TokenExchangeRole* sV2。允许 Greengrass 核心设备获得临时证书的 IAM 角色的名称。AWS如果该角色不存在，则安装程序会创建该角色并创建并附加名为的策略*GreengrassV2TokenExchangeRole*Access。有关更多信息，请参阅 授权核心设备与AWS服务。

h. *GreengrassCoreTokenExchangeRoleAlias*。IAM 角色的别名，允许 Greengrass 核心设备稍后获得临时证书。如果角色别名不存在，则安装程序会创建该别名并将其指向您指定的 IAM 角色。有关更多信息，请参阅 授权核心设备与AWS服务。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

> **Important**
>
> 在 Windows 核心设备上，`--setup-system-service true`必须指定将AWS IoT Greengrass核心软件设置为系统服务。

如果安装成功，安装程序会打印以下消息：

- 如果您指定`--provision`，则安装程序将在成功配置资源时打印`Successfully configured Nucleus with provisioned resource details`出来。
- 如果您指定`--deploy-dev-tools`，则安装程序将在成功创建部署时打印`Configured Nucleus to deploy aws.greengrass.Cli component`出来。
- 如果您指定`--setup-system-service true`，则安装程序在将软件设置为服务并运行时会打印`Successfully set up Nucleus as a system service`出来。
- 如果您未指定`--setup-system-service true`，则安装程序会打印`Launched Nucleus successfully`成功并运行软件。

2. 如果您安装了 Greengrass 核 v2.0.4 或更高版本，请跳过此步骤。如果您下载的是最新版本的软件，则说明您安装了 v2.0.4 或更高版本。

   运行以下命令为AWS IoT Greengrass核心软件根文件夹设置所需的文件权限。*`/greengrass/v2`*替换为您在安装命令中指定的根文件夹，将 *`/greengrass`* 替换为根文件夹的父文件夹。

```
sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
```

如果您将 AWS IoT Greengrass Core 软件作为系统服务安装，则安装程序会为您运行该软件。否则，必须手动运行该软件。有关更多信息，请参阅 运行AWS IoT Greengrass核心软件。

> **Note**
> 默认情况下，安装程序创建的 IAM 角色不允许访问 S3 存储桶中的组件项目。要在 Amazon S3 中部署定义构件的自定义组件，您必须向该角色添加权限以允许您的核心设备检索组件工件。有关更多信息，请参阅 允许访问 S3 存储桶以获取组件项目。
> 如果您还没有用于组件工件的 S3 存储桶，则可以在创建存储桶后添加这些权限。

有关如何配置和使用软件的更多信息AWS IoT Greengrass，请参阅以下内容：

- 配置 AWS IoT Greengrass 核心软件
- 开发AWS IoT Greengrass组件
- 将AWS IoT Greengrass组件部署到设备
- Greengrass 命令行界面

使用手动资源配置来安装 AWS IoT Greengrass Core 软件

AWS IoT Greengrass 核心软件包括一个安装程序，可将您的设备设置为 Greengrass 核心设备。要手动设置设备，您可以创建设备要使用的必需资源 AWS IoT 和 IAM 资源。如果您手动创建这些资源，则无需向安装程序提供 AWS 凭据。

手动安装 AWS IoT Greengrass Core 软件时，也可以将设备配置为使用网络代理或通过端口 443 AWS 进行连接。例如，如果您的设备在防火墙或网络代理后面运行，则可能需要指定这些配置选项。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。

您也可以通过 PKCS #11 接口将 AWS IoT Greengrass 核心软件配置为使用硬件安全模块 (HSM)。此功能使您能够安全地存储私钥和证书文件，这样它们就不会在软件中暴露或复制。您可以将私钥和证书存储在硬件模块上，例如 HSM、可信平台模块 (TPM) 或其他加密元素。此功能仅在 Linux 设备上可用。有关硬件安全性及其使用要求的更多信息，请参阅硬件安全性集成。

Important

在下载 AWS IoT Greengrass 酷睿软件之前，请检查您的核心设备是否满足安装和运行 AWS IoT Greengrass 酷睿软件 v2.0 的要求。

主题

- 检索 AWS IoT 端点
- 创建一 AWS IoT 件事
- 创建事物证书
- 配置事物证书
- 创建代币交换角色
- 将证书下载到设备
- 设置设备环境
- 下载 AWS IoT Greengrass 核心软件
- 安装 AWS IoT Greengrass 核心软件

检索 AWS IoT 端点

获取您的终 AWS IoT 端节点 AWS 账户，然后将其保存以备后用。您的设备使用这些端点进行连接 AWS IoT。执行以下操作：

1. 获取您的 AWS IoT 数据端点 AWS 账户。

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

如果请求成功，则响应类似于以下示例。

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. 获取您的 AWS IoT 凭证端点 AWS 账户。

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

如果请求成功，则响应类似于以下示例。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

创建一 AWS IoT 件事

AWS IoT 事物代表连接到的设备和逻辑实体 AWS IoT。Greengrass 的核心设备就是东西。 AWS IoT 当您将设备注册为 AWS IoT 事物时，该设备可以使用数字证书进行身份验证 AWS。

在本节中，您将创建一个代表您的设备的 AWS IoT 东西。

创建 AWS IoT 事物

1. 为你的设备创建 AWS IoT 一件东西。在您的开发计算机上，运行以下命令。

   - *MyGreengrassCore*替换为要使用的事物名称。这个名字也是你的 Greengrass 核心设备的名字。

   > Note
   >
   > 事物名称不能包含冒号 (:) 字符。

   ```
   aws iot create-thing --thing-name MyGreengrassCore
   ```

   如果请求成功，则响应类似于以下示例。

   ```
   {
     "thingName": "MyGreengrassCore",
     "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
     "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
   }
   ```

2. （可选）将 AWS IoT 事物添加到新的或现有的事物组。您可以使用事物组来管理 Greengrass 核心设备群。将软件组件部署到设备时，可以将单个设备或设备组作为目标。您可以将设备添加到已激活 Greengrass 部署的事物组，以将该事物组的软件组件部署到该设备上。执行以下操作：

   a. （可选）创建 AWS IoT 事物组。

      - *MyGreengrassCoreGroup*替换为要创建的事物组的名称。

      > **Note**
      >
      > 事物组名称不能包含冒号 (:) 字符。

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      如果请求成功，则响应类似于以下示例。

      ```
      {
        "thingGroupName": "MyGreengrassCoreGroup",
        "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
      MyGreengrassCoreGroup",
        "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
      }
      ```

   b. 将 AWS IoT 事物添加到事物组。

      - *MyGreengrassCore*用你的 AWS IoT 东西的名字替换。
      - *MyGreengrassCoreGroup*替换为事物组的名称。

      ```
      aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
      name MyGreengrassCoreGroup
      ```

      如果请求成功，则该命令没有任何输出。

创建事物证书

当您将设备注册为 AWS IoT 事物时，该设备可以使用数字证书进行身份验证 AWS。此证书允许设备与 AWS IoT 和通信 AWS IoT Greengrass。

在本节中，您将创建和下载设备可用于连接的证书 AWS。

如果要将 AWS IoT Greengrass Core 软件配置为使用硬件安全模块 (HSM) 来安全存储私钥和证书，请按照步骤在 HSM 中使用私钥创建证书。否则，请按照步骤在 AWS IoT 服务中创建证书和私钥。硬件安全功能仅在 Linux 设备上可用。有关硬件安全性及其使用要求的更多信息，请参阅硬件安全性集成。

在 AWS IoT 服务中创建证书和私钥

创建事物证书

1. 创建一个文件夹，用于下载 AWS IoT 事物的证书。

```
mkdir greengrass-v2-certs
```

2. 为该 AWS IoT 事物创建并下载证书。

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
 greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

如果请求成功，则响应类似于以下示例。


```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
```

```
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```


保存证书的 Amazon 资源名称 (ARN)，以便稍后用于配置证书。

使用 HSM 中的私钥创建证书

> **Note**
>
> 此功能可用于 Greengrass nucleus 组件的 v2.5.3 及更高版本。 AWS IoT Greengrass 目前不支持在 Windows 核心设备上使用此功能。

创建事物证书

1. 在核心设备上，在 HSM 中初始化 PKCS #11 令牌，然后生成私钥。私钥必须是密钥大小为 RSA-2048（或更大）的 RSA 密钥或 ECC 密钥。

   > **Note**
   >
   > 要使用带有 ECC 密钥的硬件安全模块，必须使用 Greengrass nucleus v2.5.6 或更高版本。
   >
   > 要使用硬件安全模块和密钥管理器，必须使用带有 RSA 密钥的硬件安全模块。

查看 HSM 的文档，了解如何初始化令牌并生成私钥。如果您的 HSM 支持对象 ID，请在生成私钥时指定对象 ID。保存您在初始化令牌并生成私钥时指定的插槽 ID、用户 PIN、对象标签、对象 ID（如果您的 HSM 使用一个）。稍后将事物证书导入 HSM 并配置 C AWS IoT Greengrass ore 软件时，您将使用这些值。

2. 使用私钥创建证书签名请求 (CSR)。 AWS IoT 使用此 CSR 为您在 HSM 中生成的私钥创建事物证书。有关如何使用私钥创建 CSR 的信息，请参阅您的 HSM 文档。CSR 是一个文件，例如`iotdevicekey.csr`。

3. 将 CSR 从设备复制到您的开发计算机。如果在开发计算机和设备上启用了 SSH 和 SCP，则可以在开发计算机上使用`scp`命令传输 CSR。*device-ip-address*替换为设备的 IP 地址，将 *~/iotdevicekey.csr ###### CSR* 文件的路径。

```
scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
```

4. 在您的开发计算机上，创建一个文件夹，用于下载该 AWS IoT 事物的证书。

```
mkdir greengrass-v2-certs
```

5. 使用 CSR 文件创建 AWS IoT 事物的证书并将其下载到您的开发计算机上。

```
aws iot create-certificate-from-csr --set-as-active --certificate-signing-
request=file://iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/
device.pem.crt
```

如果请求成功，则响应类似于以下示例。

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
```

```
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

保存证书的 ARN 以供日后配置证书时使用。

配置事物证书

将事物证书附加到您之前创建 AWS IoT 的事物，然后向证书添加 AWS IoT 策略以定义核心设备的 AWS IoT 权限。

配置事物的证书

1. 将证书附加到 AWS IoT 事物上。

   - *MyGreengrassCore*用你的 AWS IoT 东西的名字替换。
   - 将证书 Amazon 资源名称 (ARN) 替换为您在上一步中创建的证书的 ARN。

   ```
   aws iot attach-thing-principal --thing-name MyGreengrassCore
    --principal arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   如果请求成功，则该命令没有任何输出。

2. 创建并附加用于定义 Greengrass 核心设备 AWS IoT 权限的 AWS IoT 策略。以下策略允许访问所有 MQTT 主题和 Greengrass 操作，因此您的设备可以处理需要新 Greengrass 操作的自定义应用程序和未来的更改。您可以根据自己的用例限制此政策。有关更多信息，请参阅 AWS IoT Greengrass V2核心设备的最低AWS IoT政策。

   如果您之前设置过 Greengrass 核心设备，则可以附加 AWS IoT 其策略，而不必创建新的策略。

   执行以下操作：

a. 创建一个包含 Greengrass 核心设备所需的 AWS IoT 策略文档的文件。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano greengrass-v2-iot-policy.json
```

将以下 JSON 复制到文件中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "iot:Connect",
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

b. 根据 AWS IoT 策略文档创建策略。

- 将 *GreenGrassv2IoT* 替换为要ThingPolicy创建的策略的名称。

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
```

```
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
          \\"*\\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. 将 AWS IoT 策略附加到 AWS IoT 事物的证书上。

- 将 *GreenGrassv2IoT* 替换为要ThingPolicy附加的策略的名称。
- 用你的东西的证书的 ARN 替换目标 ARN。 AWS IoT

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

如果请求成功，则该命令没有任何输出。

创建代币交换角色

Greengrass 核心设备使用 IAM 服务角色（称为令牌交换角色）来授权对服务的调用。 AWS 设备使用 AWS IoT 证书提供程序来获取此角色的临时 AWS 证书，从而允许设备与 Amazon Logs 进行交互 AWS IoT、向 Amazon Logs 发送 CloudWatch 日志以及从 Amazon S3 下载自定义组件项目。有关更多信息，请参阅 [授权核心设备与AWS服务](#)。

您可以使用 AWS IoT 角色别名为 Greengrass 核心设备配置令牌交换角色。角色别名允许您更改设备的令牌交换角色，但设备配置保持不变。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的[授权直接调用 AWS 服务](#)。

在本节中，您将创建一个令牌交换 IAM 角色和一个指向该 AWS IoT 角色的角色别名。如果您已经设置了 Greengrass 核心设备，则可以使用其代币交换角色和角色别名，而不必创建新的代币交换角色和角色别名。然后，您将设备配置为 AWS IoT 使用该角色和别名。

创建代币交换 IAM 角色

1. 创建您的设备可用作令牌交换角色的 IAM 角色。执行以下操作：

   a. 创建包含令牌交换角色所需的信任策略文档的文件。

      例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

      ```
      nano device-role-trust-policy.json
      ```

      将以下 JSON 复制到文件中。

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "credentials.iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. 使用信任策略文档创建令牌交换角色。

      - 将 *GreenGrassV2 TokenExchangeRole* 替换为要创建的 IAM 角色的名称。

      ```
      aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file://device-role-trust-policy.json
      ```

如果请求成功，则响应类似于以下示例。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. 创建包含令牌交换角色所需的访问策略文档的文件。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano device-role-access-policy.json
```

将以下 JSON 复制到文件中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
```

```
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 此访问策略不允许访问 S3 存储桶中的组件项目。要在 Amazon S3 中部署定义构件的自定义组件，您必须向该角色添加权限以允许您的核心设备检索组件工件。有关更多信息，请参阅 [允许访问 S3 存储桶以获取组件项目]。
> 如果您还没有用于组件工件的 S3 存储桶，则可以在创建存储桶后添加这些权限。

d. 根据策略文档创建 IAM 策略。

- 将 *GreenGrassV2 TokenExchangeRoleAccess* 替换为要创建的 IAM 策略的名称。

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. 将 IAM 策略附加到令牌交换角色。

- 将 *GreenGrassV2* 替换为 IA TokenExchangeRole M 角色的名称。
- 将策略 ARN 替换为您在上一步中创建的 IAM 策略的 ARN。

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

如果请求成功，则该命令没有任何输出。

2. 创建 AWS IoT 指向代币交换角色的角色别名。

- *GreengrassCoreTokenExchangeRoleAlias*替换为要创建的角色别名的名称。
- 将角色 ARN 替换为您在上一步中创建的 IAM 角色的 ARN。

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

如果请求成功，则响应类似于以下示例。

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> 要创建角色别名，您必须有权将令牌交换 IAM 角色传递给 AWS IoT。如果您在尝试创建角色别名时收到错误消息，请检查您的 AWS 用户是否具有此权限。有关更多信息，请参阅《用户指南》中的授予AWS Identity and Access Management 用户向 AWS 服务传递角色的权限。

3. 创建并附加 AWS IoT 允许您的 Greengrass 核心设备使用角色别名担任令牌交换角色的策略。如果您之前设置过 Greengrass 核心设备，则可以附加其角色 AWS IoT 别名策略，而不必创建新的角色别名策略。执行以下操作：

   a. （可选）创建一个包含角色别名所需的 AWS IoT 策略文档的文件。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano greengrass-v2-iot-role-alias-policy.json
```

将以下 JSON 复制到文件中。

- 将资源 ARN 替换为角色别名的 ARN。

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. 根据 AWS IoT 策略文档创建策略。

- *GreengrassCoreTokenExchangeRoleAliasPolicy*替换为要创建的 AWS IoT 策略的名称。

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --policy-document file://greengrass-v2-iot-role-alias-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
```

```
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. 将 AWS IoT 策略附加到 AWS IoT 事物的证书上。

   - *GreengrassCoreTokenExchangeRoleAliasPolicy*替换为角色别名 AWS IoT 策略的名称。
   - 用你的东西的证书的 ARN 替换目标 ARN。 AWS IoT

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

   如果请求成功，则该命令没有任何输出。

将证书下载到设备

之前，您已将设备的证书下载到开发计算机上。在本节中，您将证书复制到您的核心设备，以便为设备设置用于连接的证书 AWS IoT。您还可以下载 Amazon 根证书颁发机构 (CA) 证书。如果您使用 HSM，也可以在本节中将证书文件导入 HSM。

- 如果您之前在 AWS IoT 服务中创建了事物证书和私钥，请按照步骤下载带有私钥和证书文件的证书。
- 如果您之前使用硬件安全模块 (HSM) 中的私钥创建了事物证书，请按照步骤在 HSM 中下载带有私钥和证书的证书。

下载带有私钥的证书和证书文件

将证书下载到设备

1. 将 AWS IoT 事物证书从开发计算机复制到设备。如果在开发计算机和设备上启用了 SSH 和 SCP，则可以在开发计算机上使用`scp`命令来传输证书。*device-ip-address*替换为设备的 IP 地址。

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. 在设备上创建 Greengrass 根文件夹。稍后，您将将 AWS IoT Greengrass Core 软件安装到此文件夹。

   Linux or Unix

   - */greengrass/v2*替换为要使用的文件夹。

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - 将 *C:\greengrass\v2* 替换为要使用的文件夹。

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - 将 *C:\greengrass\v2* 替换为要使用的文件夹。

   ```
   mkdir C:\greengrass\v2
   ```

3. （仅限 Linux）设置 Greengrass 根文件夹的父文件夹的权限。

   - 将 */greengrass* 替换为根文件夹的父文件夹。

   ```
   sudo chmod 755 /greengrass
   ```

4. 将 AWS IoT 事物证书复制到 Greengrass 根文件夹。

   Linux or Unix

   • */greengrass/v2*替换为 Greengrass 根文件夹。

   ```
   sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
   ```

   Windows Command Prompt

   • 将 *C:\greengrass\v2* 替换为要使用的文件夹。

   ```
   robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
   ```

   PowerShell

   • 将 *C:\greengrass\v2* 替换为要使用的文件夹。

   ```
   cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
   ```

5. 下载 Amazon 根证书颁发机构 (CA) 证书。 AWS IoT 默认情况下，证书与亚马逊的根 CA 证书相关联。

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
   \greengrass\v2\\AmazonRootCA1.pem
   ```

在 HSM 中下载带有私钥和证书的证书

此功能可用于 Greengrass nucleus 组件的 v2.5.3 及更高版本。 AWS IoT Greengrass 目前不支持在 Windows 核心设备上使用此功能。

将证书下载到设备

1. 将 AWS IoT 事物证书从开发计算机复制到设备。如果在开发计算机和设备上启用了 SSH 和 SCP，则可以在开发计算机上使用`scp`命令来传输证书。*device-ip-address*替换为设备的 IP 地址。

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. 在设备上创建 Greengrass 根文件夹。稍后，您将将 AWS IoT Greengrass Core 软件安装到此文件夹。

   Linux or Unix

   - */greengrass/v2*替换为要使用的文件夹。

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - 将 *C:\greengrass\v2* 替换为要使用的文件夹。

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - 将 *C:\greengrass\v2* 替换为要使用的文件夹。

   ```
   mkdir C:\greengrass\v2
   ```

3. （仅限 Linux）设置 Greengrass 根文件夹的父文件夹的权限。

- 将 */greengrass* 替换为根文件夹的父文件夹。

```
sudo chmod 755 /greengrass
```

4. 将事物证书文件导入 HSM。~/greengrass-v2-certs/device.pem.crt查看 HSM 的文档，了解如何将证书导入其中。使用之前在 HSM 中生成私钥时使用的相同令牌、插槽 ID、用户 PIN、对象标签和对象 ID（如果您的 HSM 使用一个）导入证书。

> **Note**
>
> 如果您之前生成了没有对象 ID 的私钥，并且证书具有对象 ID，请将私钥的对象 ID 设置为与证书相同的值。查看 HSM 的文档，了解如何为私钥对象设置对象 ID。

5. （可选）删除事物证书文件，使其仅存在于 HSM 中。

```
rm ~/greengrass-v2-certs/device.pem.crt
```

6. 下载 Amazon 根证书颁发机构 (CA) 证书。 AWS IoT 默认情况下，证书与亚马逊的根 CA 证书相关联。

Linux or Unix

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

设置设备环境

按照本节中的步骤设置要用作 AWS IoT Greengrass 核心设备的 Linux 或 Windows 设备。

设置 Linux 设备

设置 Linux 设备用于 AWS IoT Greengrass V2

1. 安装 Java 运行时， AWS IoT Greengrass 核心软件需要运行该运行时。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。以下命令向您展示了如何在您的设备上安装 OpenJDK。

   - 对于基于 Debian 或基于 Ubuntua 的发行版：

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

   - 对于基于 Red Hat 的发行版：

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

   - 对于 Amazon Linux 2：

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

   - 对于亚马逊 Linux 2023：

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

   安装完成后，运行以下命令以验证 Java 是否在您的 Linux 设备上运行。

   ```
   java -version
   ```

   该命令会打印设备上运行的 Java 版本。例如，在基于 Debian 的发行版上，输出可能与以下示例类似。

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

2. （可选）创建在设备上运行组件的默认系统用户和组。您也可以选择让 AWS IoT Greengrass 核心软件安装程序在安装过程中使用安装程序参数创建此用户和组。`--component-default-user`有关更多信息，请参阅 安装程序参数。

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

3. 验证运行 AWS IoT Greengrass Core 软件的用户（通常`root`）是否有权`sudo`与任何用户和任何组一起运行。

   a. 运行以下命令打开该`/etc/sudoers`文件。

   ```
   sudo visudo
   ```

   b. 验证用户的权限是否如以下示例所示。

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

4. （可选）要运行容器化 Lambda 函数，必须启用 c groups v1，并且必须启用并挂载内存和设备 cgroups。如果您不打算运行容器化 Lambda 函数，则可以跳过此步骤。

   要启用这些 cgroups 选项，请使用以下 Linux 内核参数启动设备。

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

   有关查看和设置设备内核参数的信息，请参阅操作系统和启动加载程序的文档。按照说明永久设置内核参数。

5. 按照中的要求列表所示，在您的设备上安装所有其他必需的依赖项设备要求。

设置 Windows 设备

Note

此功能适用于 Greengrass nucleus 组件的 2.5.0 及更高版本。

要将 Windows 设备设置为 AWS IoT Greengrass V2

1. 安装 Java 运行时， AWS IoT Greengrass 核心软件需要运行该运行时。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。
2. 检查 PATH 系统变量上是否有 Java 可用，如果没有，请添加它。该 LocalSystem 帐户运行 AWS IoT Greengrass Core 软件，因此您必须将 Java 添加到 PATH 系统变量中，而不是用户的 PATH 用户变量。执行以下操作：

    a. 按下 Windows 键打开开始菜单。
    b. 键入**environment variables**以从开始菜单中搜索系统选项。
    c. 在开始菜单搜索结果中，选择编辑系统环境变量以打开系统属性窗口。
    d. 选择环境变量... 打开 “环境变量” 窗口。
    e. 在 “系统变量” 下，选择 “路径”，然后选择 “编辑”。在 “编辑环境变量” 窗口中，可以在单独的行上查看每个路径。
    f. 检查 Java 安装bin文件夹的路径是否存在。路径可能与以下示例类似。

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

    g. 如果路径中缺少 Java 安装bin的文件夹，请选择 “新建” 将其添加，然后选择 “确定”。
3. 以管理员身份打开 Windows 命令提示符 (cmd.exe)。
4. 在 Windows 设备上的 LocalSystem 帐户中创建默认用户。将*##*替换为安全密码。

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

    Tip

    根据你的 Windows 配置，用户的密码可能会设置为在将来的某个日期过期。为确保您的 Greengrass 应用程序继续运行，请跟踪密码何时过期，并在密码过期之前对其进行更新。您也可以将用户的密码设置为永不过期。

    - 要检查用户及其密码何时过期，请运行以下命令。

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

    - 要将用户的密码设置为永不过期，请运行以下命令。

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

- 如果你使用的是已弃用该wmic命令的 Windows 10 或更高版本，请运行以下 PowerShell 命令。

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

5. 从微软下载该PsExec实用程序并将其安装到设备上。
6. 使用该 PsExec 实用程序将默认用户的用户名和密码存储在 LocalSystem 账户的凭据管理器实例中。将*##*替换为您之前设置的用户密码。

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

如果PsExec License Agreement打开，Accept请选择同意许可并运行命令。

> **Note**
>
> 在 Windows 设备上，该 LocalSystem 帐户运行 Greengrass 核心，您必须使用 PsExec 该实用程序在帐户中存储默认用户信息。 LocalSystem 使用凭据管理器应用程序将此信息存储在当前登录用户的 Windows 帐户中，而不是 LocalSystem 帐户中。

下载 AWS IoT Greengrass 核心软件

您可以从以下位置下载最新版本的 AWS IoT Greengrass Core 软件：

- https://d2s8p88vqu9w66.cloudfront.net/releases/ greengrass-nucleus-latest .zip

> **Note**
>
> 您可以从以下位置下载特定版本的 AWS IoT Greengrass Core 软件。将*##*替换为要下载的版本。
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

下载 AWS IoT Greengrass 核心软件

1. 在您的核心设备上，将 AWS IoT Greengrass Core 软件下载到名为的文件中`greengrass-nucleus-latest.zip`。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   下载此软件即表示您同意Greengrass Core 软件许可协议。

2. （可选）验证 Greengrass nucleus 软件签名

   > **Note**
   > 此功能在 Greengrass nucleus 版本 2.9.5 及更高版本中可用。

   a. 使用以下命令验证你的 Greengrass 核工件的签名：

      Linux or Unix

      ```
      jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
      ```

      Windows Command Prompt (CMD)

      根据您安装的 JDK 版本，文件名可能有所不同。*jdk17.0.6_10*替换为您安装的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

根据您安装的 JDK 版本，文件名可能有所不同。*jdk17.0.6_10*替换为您安装的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. 该`jarsigner`调用会产生指示验证结果的输出。

   i. 如果 Greengrass nucleus zip 文件已签名，则输出将包含以下语句：

```
jar verified.
```

   ii. 如果 Greengrass nucleus zip 文件未签名，则输出将包含以下语句：

```
jar is unsigned.
```

c. 如果您提供了 Jarsigner `-certs` 选项以及`-verify`和`-verbose`选项，则输出还包括详细的签名者证书信息。

3. 将 AWS IoT Greengrass Core 软件解压缩到设备上的某个文件夹。*GreengrassInstaller*替换为要使用的文件夹。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

4. （可选）运行以下命令以查看 AWS IoT Greengrass Core 软件的版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> 如果您安装了 v2.4.0 之前的 Greengrass nucleus 版本，则在安装 Core 软件后请勿删除此文件夹。 AWS IoT Greengrass C AWS IoT Greengrass ore 软件使用此文件夹中的文件运行。如果您下载的是最新版本的软件，则需要安装 v2.4.0 或更高版本，并且可以在安装 C AWS IoT Greengrass ore 软件后删除此文件夹。

安装 AWS IoT Greengrass 核心软件

使用指定以下操作的参数运行安装程序：

- 从指定使用您之前创建的 AWS 资源和证书的部分配置文件进行安装。 AWS IoT Greengrass Core 软件使用配置文件来指定设备上每个 Greengrass 组件的配置。安装程序会根据您提供的部分配置文件创建完整的配置文件。
- 指定使用`ggc_user`系统用户在核心设备上运行软件组件。在 Linux 设备上，此命令还指定使用`ggc_group`系统组，安装程序会为您创建系统用户和组。
- 将 AWS IoT Greengrass Core 软件设置为启动时运行的系统服务。在 Linux 设备上，这需要 Systemd 初始化系统。

> **Important**
>
> 在 Windows 核心设备上，必须将 AWS IoT Greengrass 核心软件设置为系统服务。

有关您可以指定的参数的更多信息，请参阅安装程序参数。

如果您在内存有限的设备 AWS IoT Greengrass 上运行，则可以控制 AWS IoT Greengrass 酷睿软件使用的内存量。要控制内存分配，您可以在 nucleus 组件的`jvmOptions`配置参数中设置 JVM 堆大小选项。有关更多信息，请参阅 使用 JVM 选项控制内存分配。

- 如果您之前在 AWS IoT 服务中创建了事物证书和私钥，请按照步骤安装带有私钥和证书文件的 AWS IoT Greengrass 核心软件。
- 如果您之前使用硬件安全模块 (HSM) 中的私钥创建了事物证书，请按照步骤在 HSM 中安装带有私钥和证书的 AWS IoT Greengrass Core 软件。

使用私钥和证书文件安装 C AWS IoT Greengrass ore 软件

安装 AWS IoT Greengrass 核心软件

1. 检查 AWS IoT Greengrass 核心软件的版本。

   - *GreengrassInstaller*替换为包含该软件的文件夹的路径。

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. 使用文本编辑器创建名为的配置文件`config.yaml`以提供给安装程序。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

   ```
   nano GreengrassInstaller/config.yaml
   ```

   将以下 YAML 内容复制到文件中。此部分配置文件指定了系统参数和 Greengrass 核参数。

   ```
   ---
   system:
     certificateFilePath: "/greengrass/v2/device.pem.crt"
     privateKeyPath: "/greengrass/v2/private.pem.key"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
     aws.greengrass.Nucleus:
   ```

```
    componentType: "NUCLEUS"
    version: "2.12.3"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
```

然后执行以下操作：

- 将的*/greengrass/v2*每个实例替换为 Greengrass 根文件夹。
- *MyGreengrassCore*替换为 AWS IoT 事物的名称。
- 将 *2.12.3* 替换为 AWS IoT Greengrass 核心软件版本。
- 将 *us-west-2* 替换为您创建资源的位置 AWS 区域 。
- *GreengrassCoreTokenExchangeRoleAlias*替换为令牌交换角色别名的名称。
- `iotDataEndpoint`用您的 AWS IoT 数据端点替换。
- 用您的`iotCredEndpoint` AWS IoT 凭证终端节点替换。

> **Note**
>
> 在此配置文件中，您可以自定义其他 nucleus 配置选项，例如要使用的端口和网络代理，如以下示例所示。有关更多信息，请参阅 Greengrass 核配置。
>
> ```
> ---
> system:
>   certificateFilePath: "/greengrass/v2/device.pem.crt"
>   privateKeyPath: "/greengrass/v2/private.pem.key"
>   rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
>   rootpath: "/greengrass/v2"
>   thingName: "MyGreengrassCore"
> services:
>   aws.greengrass.Nucleus:
>     componentType: "NUCLEUS"
>     version: "2.12.3"
>     configuration:
>       awsRegion: "us-west-2"
>       iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
> ```

```
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
```

3. 运行安装程序，`--init-config`并指定提供配置文件。

   - 用 Greengrass 根文件夹替换*/greengrass/v2*或 *C:\greengrass\v2*。
   - 将的*GreengrassInstaller*每个实例替换为解压安装程序所在的文件夹。

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
```

```
  --setup-system-service true
```

> **Important**
>
> 在 Windows 核心设备上，`--setup-system-service true`必须指定将 AWS IoT Greengrass 核心软件设置为系统服务。

如果您指定`--setup-system-service true`，则安装程序在将软件设置为系统服务并运行时会打印`Successfully set up Nucleus as a system service`出来。否则，如果安装程序成功安装了软件，则不会输出任何消息。

> **Note**
>
> 在没有`deploy-dev-tools`参数的情况下运行安装程序时，不能使用`--provision true`参数来部署本地开发工具。有关直接在您的设备上部署 Greengrass CLI 的信息，请参阅。Greengrass 命令行界面

4. 通过查看根文件夹中的文件来验证安装。

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   如果安装成功，则根文件夹包含多个文件夹，例如`configpackages`、和`logs`。

在 HSM 中安装带有私钥和证书的 C AWS IoT Greengrass ore 软件

Note

此功能可用于 Greengrass nucleus 组件的 v2.5.3 及更高版本。 AWS IoT Greengrass 目前不支持在 Windows 核心设备上使用此功能。

安装 AWS IoT Greengrass 核心软件

1. 检查 AWS IoT Greengrass 核心软件的版本。

   - *GreengrassInstaller*替换为包含该软件的文件夹的路径。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

2. 要使 AWS IoT Greengrass 核心软件能够在 HSM 中使用私钥和证书，请在安装 C AWS IoT Greengrass ore 软件时安装 PKCS #11 提供程序组件。PKCS #11 提供程序组件是一个可以在安装过程中配置的插件。您可以从以下位置下载最新版本的 PKCS #11 提供程序组件：

   - https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar

   将 PKCS #11 提供程序插件下载到名`aws.greengrass.crypto.Pkcs11Provider.jar`为的文件中。*GreengrassInstaller*替换为要使用的文件夹。

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
aws.greengrass.crypto.Pkcs11Provider.jar
```

   下载此软件即表示您同意Greengrass Core 软件许可协议。

3. 使用文本编辑器创建名为的配置文件`config.yaml`以提供给安装程序。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano GreengrassInstaller/config.yaml
```

将以下 YAML 内容复制到文件中。此部分配置文件指定了系统参数、Greengrass nucleus 参数和 PKCS #11 提供程序参数。

```
---
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.3"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"
```

然后执行以下操作：

- 将 PKCS #11 URI 中的每个 *iotdevicekey ##########*并导入证书的对象标签。
- 将的*/greengrass/v2*每个实例替换为 Greengrass 根文件夹。
- *MyGreengrassCore*替换为 AWS IoT 事物的名称。
- 将 *2.12.3* 替换为 AWS IoT Greengrass 核心软件版本。
- 将 *us-west-2* 替换为您创建资源的位置 AWS 区域 。
- *GreengrassCoreTokenExchangeRoleAlias*替换为令牌交换角色别名的名称。
- `iotDataEndpoint`用您的 AWS IoT 数据端点替换。
- 用您的`iotCredEndpoint` AWS IoT 凭证终端节点替换。

- 将aws.greengrass.crypto.Pkcs11Provider组件的配置参数替换为核心设备上 HSM 配置的值。

> **Note**
>
> 在此配置文件中，您可以自定义其他 nucleus 配置选项，例如要使用的端口和网络代理，如以下示例所示。有关更多信息，请参阅 Greengrass 核配置。
>
> ```
> ---
> system:
>   certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
>   privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
>   rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
>   rootpath: "/greengrass/v2"
>   thingName: "MyGreengrassCore"
> services:
>   aws.greengrass.Nucleus:
>     componentType: "NUCLEUS"
>     version: "2.12.3"
>     configuration:
>       awsRegion: "us-west-2"
>       iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
>       iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
>       iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
> west-2.amazonaws.com"
>       mqtt:
>         port: 443
>       greengrassDataPlanePort: 443
>       networkProxy:
>         noProxyAddresses: "http://192.168.0.1,www.example.com"
>         proxy:
>           url: "https://my-proxy-server:1100"
>           username: "Mary_Major"
>           password: "pass@word1357"
>   aws.greengrass.crypto.Pkcs11Provider:
>     configuration:
>       name: "softhsm_pkcs11"
>       library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
>       slot: 1
>       userPin: "1234"
> ```

4. 运行安装程序，`--init-config`并指定提供配置文件。

   - *`/greengrass/v2`*替换为 Greengrass 根文件夹。
   - 将的*`GreengrassInstaller`*每个实例替换为解压安装程序所在的文件夹。

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin ./GreengrassInstaller/aws.greengrass.crypto.Pkcs11Provider.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

> **Important**
>
> 在 Windows 核心设备上，`--setup-system-service true`必须指定将 AWS IoT Greengrass 核心软件设置为系统服务。

如果您指定`--setup-system-service true`，则安装程序在将软件设置为系统服务并运行时会打印`Successfully set up Nucleus as a system service`出来。否则，如果安装程序成功安装了软件，则不会输出任何消息。

> **Note**
>
> 在没有`deploy-dev-tools`参数的情况下运行安装程序时，不能使用`--provision true`参数来部署本地开发工具。有关直接在您的设备上部署 Greengrass CLI 的信息，请参阅。Greengrass 命令行界面

5. 通过查看根文件夹中的文件来验证安装。

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

如果安装成功，则根文件夹包含多个文件夹，例如configpackages、和logs。

如果您将 AWS IoT Greengrass Core 软件作为系统服务安装，则安装程序会为您运行该软件。否则，必须手动运行该软件。有关更多信息，请参阅 运行AWS IoT Greengrass核心软件。

有关如何配置和使用软件的更多信息 AWS IoT Greengrass，请参阅以下内容：

- 配置 AWS IoT Greengrass 核心软件
- 开发AWS IoT Greengrass组件
- 将AWS IoT Greengrass组件部署到设备
- Greengrass 命令行界面

安装具有 AWS IoT 队列配置功能的 C AWS IoT Greengrass ore 软件

此功能适用于 Greengrass nucleus 组件的 v2.4.0 及更高版本。

借助 AWS IoT 队列配置，您可以配置 AWS IoT 为在设备首次连接时生成 X.509 设备证书和私钥并将其安全地交付 AWS IoT 给设备。 AWS IoT 提供由 Amazon 根证书颁发机构 (CA) 签署的客户证书。您还可以通过配置 AWS IoT 来为使用队列配置的 Greengrass 核心设备指定事物组、事物类型和权限。您可以定义配置模板来定义如何配置 AWS IoT 每台设备。配置模板指定了置备时要为设备创建的事物、策略和证书资源。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的配置模板。

AWS IoT Greengrass 提供了一个 AWS IoT 队列配置插件，您可以使用该插件使用 AWS IoT 队列配置创建的 AWS 资源来安装 AWS IoT Greengrass 核心软件。舰队配置插件使用按声明配置。设备使用配置声明证书和私钥来获取可用于常规操作的唯一 X.509 设备证书和私钥。在制造过程中，您可以将索赔证书和私钥嵌入每台设备中，这样您的客户便可以在每台设备上线后激活设备。您可以为多台设备使用相同的索赔证书和私钥。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的按声明进行配置。

Note

舰队配置插件目前不支持在硬件安全模块 (HSM) 中存储私钥和证书文件。要使用 HSM，请使用手动配置来安装 AWS IoT Greengrass Core 软件。

要安装具有 AWS IoT 队列配置功能的 AWS IoT Greengrass Core 软件，您必须在中设置用于配置 Gre AWS IoT engrass 核心设备的资源。 AWS 账户 这些资源包括配置模板、申领证书和令牌交换 IAM 角色。创建这些资源后，您可以重复使用它们来配置队列中的多个核心设备。有关更多信息，请参阅 为 Greengrass 核心AWS IoT设备设置队列配置。

Important

在下载 AWS IoT Greengrass 酷睿软件之前，请检查您的核心设备是否满足安装和运行 AWS IoT Greengrass 酷睿软件 v2.0 的要求。

主题

- 先决条件
- 检索 AWS IoT 端点
- 将证书下载到设备
- 设置设备环境
- 下载 AWS IoT Greengrass 核心软件
- 下载 AWS IoT 舰队配置插件
- 安装 AWS IoT Greengrass 核心软件
- 为 Greengrass 核心AWS IoT设备设置队列配置
- 配置AWS IoT舰队配置插件
- AWS IoT 舰队配置插件更新日志

先决条件

要安装具有 AWS IoT 队列配置功能的 AWS IoT Greengrass Core 软件，必须先为 Greengrass 核心 AWS IoT 设备设置队列配置。完成这些步骤后，您可以使用队列配置在任意数量的设备上安装 AWS IoT Greengrass 核心软件。

检索 AWS IoT 端点

获取您的终 AWS IoT 端节点 AWS 账户，然后将其保存以备后用。您的设备使用这些端点进行连接 AWS IoT。执行以下操作：

1. 获取您的 AWS IoT 数据端点 AWS 账户。

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

   如果请求成功，则响应类似于以下示例。

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 获取您的 AWS IoT 凭证端点 AWS 账户。

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

   如果请求成功，则响应类似于以下示例。

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

将证书下载到设备

设备使用索赔证书和私钥来验证其 AWS 调配资源和获取 X.509 设备证书的请求。您可以在制造过程中将索赔证书和私钥嵌入到设备中，也可以在安装过程中将证书和密钥复制到设备。在本节中，您将索赔证书和私钥复制到设备上。您还可以将 Amazon 根证书颁发机构 (CA) 证书下载到设备上。

> **Important**
>
> 供应声称私钥应始终受到保护，包括在 Greengrass 核心设备上。我们建议您使用 Amazon CloudWatch 指标和日志来监控是否存在滥用迹象，例如未经授权使用索赔证书来配置设备。如果您发现滥用，请禁用配置声明证书，使其无法用于设备配置。有关更多信息，请参阅 AWS IoT Core 开发人员指南 中的 监控 AWS IoT。

为了帮助您更好地管理自己在您的设备中注册的设备数量和设备 AWS 账户，您可以在创建队列配置模板时指定预配置挂钩。预配置挂钩是一种验证设备在注册期间提供的模板参数的 AWS Lambda 功能。例如，您可以创建一个预配置挂钩，根据数据库检查设备 ID，以验证设备是否有权进行置备。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的预配置挂钩。

将领取证书下载到设备

1. 将索赔证书和私钥复制到设备上。如果在开发计算机和设备上启用了 SSH 和 SCP，则可以在开发计算机上使用`scp`命令来传输声明证书和私钥。以下示例命令将开发计算机`claim-certs`上名为的文件夹中的这些文件传输到设备。*device-ip-address*替换为设备的 IP 地址。

```
scp -r claim-certs/ device-ip-address:~
```

2. 在设备上创建 Greengrass 根文件夹。稍后，您将将 AWS IoT Greengrass Core 软件安装到此文件夹。

   Linux or Unix

   - */greengrass/v2*替换为要使用的文件夹。

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - 将 *C:\greengrass\v2* 替换为要使用的文件夹。

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - 将 *C:\greengrass\v2* 替换为要使用的文件夹。

   ```
   mkdir C:\greengrass\v2
   ```

3. （仅限 Linux）设置 Greengrass 根文件夹的父文件夹的权限。

- 将 */greengrass* 替换为根文件夹的父文件夹。

```
sudo chmod 755 /greengrass
```

4. 将索赔证书移至 Greengrass 根文件夹。

- 用 Greengrass 根文件夹替换*/greengrass/v2*或 *C:\greengrass\v2*。

Linux or Unix

```
sudo mv ~/claim-certs /greengrass/v2
```

Windows Command Prompt (CMD)

```
move %USERPROFILE%\claim-certs C:\greengrass\v2
```

PowerShell

```
mv -Path ~\claim-certs -Destination C:\greengrass\v2
```

5. 下载 Amazon 根证书颁发机构 (CA) 证书。 AWS IoT 默认情况下，证书与亚马逊的根 CA 证书相关联。

Linux or Unix

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

设置设备环境

按照本节中的步骤设置要用作 AWS IoT Greengrass 核心设备的 Linux 或 Windows 设备。

设置 Linux 设备

设置 Linux 设备用于 AWS IoT Greengrass V2

1. 安装 Java 运行时， AWS IoT Greengrass 核心软件需要运行该运行时。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。以下命令向您展示了如何在您的设备上安装 OpenJDK。

   - 对于基于 Debian 或基于 Ubuntua 的发行版：

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

   - 对于基于 Red Hat 的发行版：

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

   - 对于 Amazon Linux 2：

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

   - 对于亚马逊 Linux 2023：

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

   安装完成后，运行以下命令以验证 Java 是否在您的 Linux 设备上运行。

   ```
   java -version
   ```

   该命令会打印设备上运行的 Java 版本。例如，在基于 Debian 的发行版上，输出可能与以下示例类似。

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

2. （可选）创建在设备上运行组件的默认系统用户和组。您也可以选择让 AWS IoT Greengrass 核心软件安装程序在安装过程中使用安装程序参数创建此用户和组。--component-default-user有关更多信息，请参阅 安装程序参数。

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

3. 验证运行 AWS IoT Greengrass Core 软件的用户（通常root）是否有权sudo与任何用户和任何组一起运行。

   a. 运行以下命令打开该/etc/sudoers文件。

   ```
   sudo visudo
   ```

   b. 验证用户的权限是否如以下示例所示。

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

4. （可选）要运行容器化 Lambda 函数，必须启用 c groups v1，并且必须启用并挂载内存和设备 cgroups。如果您不打算运行容器化 Lambda 函数，则可以跳过此步骤。

   要启用这些 cgroups 选项，请使用以下 Linux 内核参数启动设备。

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

   有关查看和设置设备内核参数的信息，请参阅操作系统和启动加载程序的文档。按照说明永久设置内核参数。

5. 按照中的要求列表所示，在您的设备上安装所有其他必需的依赖项设备要求。

设置 Windows 设备

> **Note**
> 此功能适用于 Greengrass nucleus 组件的 2.5.0 及更高版本。

要将 Windows 设备设置为 AWS IoT Greengrass V2

1. 安装 Java 运行时， AWS IoT Greengrass 核心软件需要运行该运行时。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。
2. 检查 PATH 系统变量上是否有 Java 可用，如果没有，请添加它。该 LocalSystem 帐户运行 AWS IoT Greengrass Core 软件，因此您必须将 Java 添加到 PATH 系统变量中，而不是用户的 PATH 用户变量。执行以下操作：

   a. 按下 Windows 键打开开始菜单。
   b. 键入**environment variables**以从 “开始” 菜单中搜索系统选项。
   c. 在开始菜单搜索结果中，选择编辑系统环境变量以打开系统属性窗口。
   d. 选择环境变量... 打开 “环境变量” 窗口。
   e. 在 “系统变量” 下，选择 “路径”，然后选择 “编辑”。在 “编辑环境变量” 窗口中，可以在单独的行上查看每个路径。
   f. 检查 Java 安装bin文件夹的路径是否存在。路径可能与以下示例类似。

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

   g. 如果路径中缺少 Java 安装bin的文件夹，请选择 “新建” 将其添加，然后选择 “确定”。
3. 以管理员身份打开 Windows 命令提示符 (cmd.exe)。
4. 在 Windows 设备上的 LocalSystem 帐户中创建默认用户。将*##*替换为安全密码。

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

> **Tip**
>
> 根据你的 Windows 配置，用户的密码可能会设置为在将来的某个日期过期。为确保您的 Greengrass 应用程序继续运行，请跟踪密码何时过期，并在密码过期之前对其进行更新。您也可以将用户的密码设置为永不过期。
>
> - 要检查用户及其密码何时过期，请运行以下命令。
>
> ```
> net user ggc_user | findstr /C:expires
> ```
>
> - 要将用户的密码设置为永不过期，请运行以下命令。
>
> ```
> wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
> ```

- 如果你使用的是已弃用该wmic命令的 Windows 10 或更高版本，请运行以下 PowerShell 命令。

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

5. 从微软下载该PsExec实用程序并将其安装到设备上。
6. 使用该 PsExec 实用程序将默认用户的用户名和密码存储在 LocalSystem 账户的凭据管理器实例中。将*##*替换为您之前设置的用户密码。

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

如果PsExec License Agreement打开，Accept请选择同意许可并运行命令。

> **Note**
>
> 在 Windows 设备上，该 LocalSystem 帐户运行 Greengrass 核心，您必须使用 PsExec 该实用程序在帐户中存储默认用户信息。 LocalSystem 使用凭据管理器应用程序将此信息存储在当前登录用户的 Windows 帐户中，而不是 LocalSystem 帐户中。

下载 AWS IoT Greengrass 核心软件

您可以从以下位置下载最新版本的 AWS IoT Greengrass Core 软件：

- https://d2s8p88vqu9w66.cloudfront.net/releases/ greengrass-nucleus-latest .zip

> **Note**
>
> 您可以从以下位置下载特定版本的 AWS IoT Greengrass Core 软件。将*##*替换为要下载的版本。
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

下载 AWS IoT Greengrass 核心软件

1. 在您的核心设备上，将 AWS IoT Greengrass Core 软件下载到名为的文件中`greengrass-nucleus-latest.zip`。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   下载此软件即表示您同意Greengrass Core 软件许可协议。

2. （可选）验证 Greengrass nucleus 软件签名

   > Note
   >
   > 此功能在 Greengrass nucleus 版本 2.9.5 及更高版本中可用。

   a. 使用以下命令来验证你的 Greengrass 核工件的签名：

      Linux or Unix

      ```
      jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
      ```

      Windows Command Prompt (CMD)

      根据您安装的 JDK 版本，文件名可能有所不同。*`jdk17.0.6_10`*替换为您安装的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

根据您安装的 JDK 版本，文件名可能有所不同。*jdk17.0.6_10*替换为您安装的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. 该`jarsigner`调用会产生指示验证结果的输出。

i. 如果 Greengrass nucleus zip 文件已签名，则输出将包含以下语句：

```
jar verified.
```

ii. 如果 Greengrass nucleus zip 文件未签名，则输出将包含以下语句：

```
jar is unsigned.
```

c. 如果您提供了 Jarsigner `-certs` 选项以及`-verify`和`-verbose`选项，则输出还包括详细的签名者证书信息。

3. 将 AWS IoT Greengrass Core 软件解压缩到设备上的某个文件夹。*GreengrassInstaller*替换为要使用的文件夹。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

4. （可选）运行以下命令以查看 AWS IoT Greengrass Core 软件的版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> 如果您安装了 v2.4.0 之前的 Greengrass nucleus 版本，则在安装 Core 软件后请勿删除此文件夹。 AWS IoT Greengrass C AWS IoT Greengrass ore 软件使用此文件夹中的文件运行。如果您下载的是最新版本的软件，则需要安装 v2.4.0 或更高版本，并且可以在安装 C AWS IoT Greengrass ore 软件后删除此文件夹。

下载 AWS IoT 舰队配置插件

您可以从以下位置下载最新版本的 AWS IoT 舰队配置插件：

- https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim / fleetprovisioningbyclaim-latest.jar

> **Note**
>
> 您可以从以下位置下载特定版本的 AWS IoT 舰队配置插件。将##替换为要下载的版本。有关每个版本的舰队配置插件的更多信息，请参阅AWS IoT 舰队配置插件更新日志。
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
> FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
> ```

舰队配置插件是开源的。要查看其源代码，请参阅上的AWS IoT 舰队配置插件 GitHub。

下载 AWS IoT 舰队配置插件

- 在您的设备上，将 AWS IoT 队列配置插件下载到名为的文件中`aws.greengrass.FleetProvisioningByClaim.jar`。*GreengrassInstaller*替换为要使用的文件夹。

  Linux or Unix

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  Windows Command Prompt (CMD)

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  PowerShell

  ```
  iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar -OutFile GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  下载此软件即表示您同意Greengrass Core 软件许可协议。

安装 AWS IoT Greengrass 核心软件

使用指定以下操作的参数运行安装程序：

- 从部分配置文件进行安装，该文件指定使用队列配置插件来配置 AWS 资源。 AWS IoT Greengrass Core 软件使用配置文件来指定设备上每个 Greengrass 组件的配置。安装程序根据您提供的部分配置文件和舰队配置插件创建的 AWS 资源创建完整的配置文件。
- 指定使用`ggc_user`系统用户在核心设备上运行软件组件。在 Linux 设备上，此命令还指定使用`ggc_group`系统组，安装程序会为您创建系统用户和组。
- 将 AWS IoT Greengrass Core 软件设置为启动时运行的系统服务。在 Linux 设备上，这需要 Systemd 初始化系统。

Important

在 Windows 核心设备上，必须将 AWS IoT Greengrass 核心软件设置为系统服务。

有关您可以指定的参数的更多信息，请参阅安装程序参数。

如果您在内存有限的设备 AWS IoT Greengrass 上运行，则可以控制 AWS IoT Greengrass 酷睿软件使用的内存量。要控制内存分配，您可以在 nucleus 组件的`jvmOptions`配置参数中设置 JVM 堆大小选项。有关更多信息，请参阅 使用 JVM 选项控制内存分配。

安装 AWS IoT Greengrass 核心软件

1. 检查 AWS IoT Greengrass 核心软件的版本。

   - *GreengrassInstaller*替换为包含该软件的文件夹的路径。

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. 使用文本编辑器创建名为的配置文件`config.yaml`以提供给安装程序。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

   ```
   nano GreengrassInstaller/config.yaml
   ```

   将以下 YAML 内容复制到文件中。此部分配置文件指定了舰队配置插件的参数。有关您可以指定的选项的更多信息，请参阅配置AWS IoT舰队配置插件。

   Linux or Unix

   ```
   ---
   services:
     aws.greengrass.Nucleus:
       version: "2.12.3"
     aws.greengrass.FleetProvisioningByClaim:
   ```

```
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.3"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

然后执行以下操作：

- 将 *2.12.3* 替换为 AWS IoT Greengrass 核心软件版本。

- 用 Greengrass 根文件夹替换*/greengrass/v2*或 *C:\greengrass\v2* 的每个实例。

> **Note**
>
> 在 Windows 设备上，必须将路径分隔符指定为双反斜杠 (\\)，例如。`C:\\greengrass\\v2`

- 将 *us-west-2* 替 AWS 换为您创建配置模板和其他资源的区域。
- `iotDataEndpoint`用您的 AWS IoT 数据端点替换。
- 用您的`iotCredentialEndpoint` AWS IoT 凭证终端节点替换。
- *GreengrassCoreTokenExchangeRoleAlias*替换为令牌交换角色别名的名称。
- *GreengrassFleetProvisioningTemplate*替换为队列配置模板的名称。
- 将`claimCertificatePath`替换为设备上申领证书的路径。
- 将`claimCertificatePrivateKeyPath`替换为设备上申领证书私钥的路径。
- 将模板参数 (`templateParameters`) 替换为用于配置设备的值。此示例指的是定义`ThingName`和`ThingGroupName`参数的示例模板。

> **Note**
>
> 在此配置文件中，您可以自定义其他配置选项，例如要使用的端口和网络代理，如以下示例所示。有关更多信息，请参阅 Greengrass 核配置。
>
> Linux or Unix
>
> ```
> ---
> services:
>   aws.greengrass.Nucleus:
>     version: "2.12.3"
>     configuration:
>       mqtt:
>         port: 443
>       greengrassDataPlanePort: 443
>       networkProxy:
>         noProxyAddresses: "http://192.168.0.1,www.example.com"
>         proxy:
>           url: "http://my-proxy-server:1100"
>           username: "Mary_Major"
>           password: "pass@word1357"
> ```

```
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.3"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
```

```
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

要使用 HTTPS 代理，必须使用队列配置插件的 1.1.0 或更高版本。您还必须指定`rootCaPath`下方`system`，如以下示例所示。

Linux or Unix

```
---
system:
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
services:
  ...
```

Windows

```
---
system:
  rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
services:
  ...
```

3. 运行安装程序。指定`--trusted-plugin`提供队列配置插件，`--init-config`并指定提供配置文件。

- */greengrass/v2*替换为 Greengrass 根文件夹。
- 将的*GreengrassInstaller*每个实例替换为解压安装程序所在的文件夹。

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

> **Important**
>
> 在 Windows 核心设备上，`--setup-system-service true`必须指定将 AWS IoT Greengrass 核心软件设置为系统服务。

如果您指定`--setup-system-service true`，则安装程序在将软件设置为系统服务并运行时会打印`Successfully set up Nucleus as a system service`出来。否则，如果安装程序成功安装了软件，则不会输出任何消息。

> **Note**
>
> 在没有`deploy-dev-tools`参数的情况下运行安装程序时，不能使用`--provision true`参数来部署本地开发工具。有关直接在您的设备上部署 Greengrass CLI 的信息，请参阅。Greengrass 命令行界面

4. 通过查看根文件夹中的文件来验证安装。

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   如果安装成功，则根文件夹包含多个文件夹，例如`config``packages`、和`logs`。

如果您将 AWS IoT Greengrass Core 软件作为系统服务安装，则安装程序会为您运行该软件。否则，必须手动运行该软件。有关更多信息，请参阅 运行AWS IoT Greengrass核心软件。

有关如何配置和使用软件的更多信息 AWS IoT Greengrass，请参阅以下内容：

- 配置 AWS IoT Greengrass 核心软件
- 开发AWS IoT Greengrass组件
- 将AWS IoT Greengrass组件部署到设备
- Greengrass 命令行界面

为 Greengrass 核心AWS IoT设备设置队列配置

要安装具有队列配置功能的 AWS IoT Greengrass Core 软件，必须先在中设置以下资源AWS 账户。这些资源使设备能够在 Greengrass 核心设备上注册AWS IoT并作为 Greengrass 核心设备运行。只需按照本节中的步骤操作一次，即可在中创建和配置这些资源AWS 账户。

- 令牌交换 IAM 角色，核心设备使用该角色来授权对AWS服务的调用。
- 指向代币交换AWS IoT角色的角色别名。
- （可选）一项AWS IoT策略，核心设备使用该策略来授权对AWS IoT和AWS IoT Greengrass服务的调用。此AWS IoT策略必须`iot:AssumeRoleWithCertificate`允许指向令牌交换AWS IoT角色的角色别名的权限。

  您可以对队列中的所有核心设备使用单一AWS IoT策略，也可以配置队列配置模板为每台核心设备创建AWS IoT策略。
- AWS IoT舰队配置模板。此模板必须指定以下内容：
  - AWS IoT事物资源。您可以指定现有事物组列表，以便在每台设备联机时将组件部署到每台设备。
  - 一种AWS IoT策略资源。此资源可以定义以下属性之一：
    - 现有AWS IoT策略的名称。如果选择此选项，则根据此模板创建的核心设备将使用相同的AWS IoT策略，并且您可以像队列一样管理它们的权限。
    - 一份AWS IoT政策文件。如果选择此选项，则根据此模板创建的每台核心设备都使用唯一的AWS IoT策略，并且您可以管理每台核心设备的权限。
  - AWS IoT证书资源。此证书资源必须使用`AWS::IoT::Certificate::Id`参数将证书附加到核心设备。有关更多信息，请参阅《AWS IoT开发人员指南》中的 J ust-in-time 配置。
- 队列AWS IoT配置模板的配置声明证书和私钥。您可以在制造过程中将此证书和私钥嵌入到设备中，这样设备就可以在上线时自行注册和配置。

> **Important**
>
> 供应声称私钥应始终受到保护，包括在 Greengrass 核心设备上。我们建议您使用 Amazon CloudWatch 指标和日志来监控是否存在滥用迹象，例如未经授权使用索赔证书来配置设备。如果您发现滥用，请禁用配置声明证书，使其无法用于设备配置。有关更多信息，请参阅 AWS IoT Core 开发人员指南 中的 监控AWS IoT。
> 为了帮助您更好地管理自己在您的设备中注册的设备数量和设备AWS 账户，您可以在创建队列配置模板时指定预配置挂钩。预配置挂钩是一种验证设备在注册期间提供的模板参数的 AWS Lambda功能。例如，您可以创建一个预配置挂钩，根据数据库检查设备 ID，以验证

设备是否有权进行置备。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的预置挂钩。

- 您附加到配置声明证书的AWS IoT策略，允许设备注册和使用队列配置模板。

主题

- 创建代币交换角色
- 创建 AWS IoT 策略
- 创建舰队配置模板
- 创建配置声明证书和私钥

创建代币交换角色

Greengrass 核心设备使用 IAM 服务角色（称为令牌交换角色）来授权对服务的调用。AWS设备使用AWS IoT证书提供程序来获取此角色的临时AWS证书，从而允许设备与 Amazon Logs 进行交互AWS IoT、向 Amazon Logs 发送 CloudWatch 日志以及从 Amazon S3 下载自定义组件项目。有关更多信息，请参阅 授权核心设备与AWS服务。

您可以使用AWS IoT角色别名为 Greengrass 核心设备配置令牌交换角色。角色别名允许您更改设备的令牌交换角色，但设备配置保持不变。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的授权直接调用AWS服务。

在本节中，您将创建一个令牌交换 IAM 角色和一个指向该AWS IoT角色的角色别名。如果您已经设置了 Greengrass 核心设备，则可以使用其代币交换角色和角色别名，而不必创建新的代币交换角色和角色别名。

创建代币交换 IAM 角色

1. 创建您的设备可用作令牌交换角色的 IAM 角色。执行以下操作：

   a. 创建包含令牌交换角色所需的信任策略文档的文件。

      例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

      ```
      nano device-role-trust-policy.json
      ```

      将以下 JSON 复制到文件中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 使用信任策略文档创建令牌交换角色。

• 将 *GreenGrassV2 TokenExchangeRole* 替换为要创建的 IAM 角色的名称。

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-policy-document file://device-role-trust-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
```

```
}
```

c. 创建包含令牌交换角色所需的访问策略文档的文件。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano device-role-access-policy.json
```

将以下 JSON 复制到文件中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 此访问策略不允许访问 S3 存储桶中的组件项目。要在 Amazon S3 中部署定义构件的自定义组件，您必须向该角色添加权限以允许您的核心设备检索组件工件。有关更多信息，请参阅 允许访问 S3 存储桶以获取组件项目。
> 如果您还没有用于组件工件的 S3 存储桶，则可以在创建存储桶后添加这些权限。

d. 根据策略文档创建 IAM 策略。

- 将 *GreenGrassV2 TokenExchangeRoleAccess* 替换为要创建的 IAM 策略的名称。

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. 将 IAM 策略附加到令牌交换角色。

- 将 *GreenGrassV2* 替换为 IA TokenExchangeRole M 角色的名称。
- 将策略 ARN 替换为您在上一步中创建的 IAM 策略的 ARN。

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

如果请求成功，则该命令没有任何输出。

2. 创建AWS IoT指向代币交换角色的角色别名。

- *GreengrassCoreTokenExchangeRoleAlias*替换为要创建的角色别名的名称。
- 将角色 ARN 替换为您在上一步中创建的 IAM 角色的 ARN。

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

如果请求成功，则响应类似于以下示例。

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> 要创建角色别名，您必须有权将令牌交换 IAM 角色传递给AWS IoT。如果您在尝试创建角色别名时收到错误消息，请检查您的AWS用户是否具有此权限。有关更多信息，请参阅《用户指南》中的授予AWS Identity and Access Management用户向AWS服务传递角色的权限。

创建 AWS IoT 策略

将设备注册为AWS IoT事物后，该设备可以使用数字证书进行身份验证AWS。该证书包含一个或多个AWS IoT策略，这些策略定义了设备可以与证书一起使用的权限。这些策略允许设备与AWS IoT和通信AWS IoT Greengrass。

通过AWS IoT队列配置，设备可以AWS IoT连接到以创建和下载设备证书。在下一节中创建的队列配置模板中，您可以指定是将相同的AWS IoT策略AWS IoT附加到所有设备的证书，还是为每台设备创建新策略。

在本节中，您将创建一个AWS IoT附加到所有设备证书的AWS IoT策略。通过这种方法，您可以将所有设备的权限作为一个队列进行管理。如果您想为每台设备创建新AWS IoT策略，则可以跳过本节，在定义队列模板时参考其中的策略。

创建 AWS IoT策略

- 创建AWS IoT策略来定义您的 Greengrass 核心设备队列的AWS IoT权限。以下策略允许访问所有 MQTT 主题和 Greengrass 操作，因此您的设备可以处理需要新 Greengrass 操作的自定义应用程序和未来的更改。此策略还允许该`iot:AssumeRoleWithCertificate`权限，允许您的设备使

用您在上一节中创建的令牌交换角色。您可以根据自己的用例限制此政策。有关更多信息，请参阅AWS IoT Greengrass V2核心设备的最低AWS IoT政策。

执行以下操作：

a. 创建一个包含 Greengrass 核心设备所需的AWS IoT策略文档的文件。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano greengrass-v2-iot-policy.json
```

将以下 JSON 复制到文件中。

- 将`iot:AssumeRoleWithCertificate`资源替换为您在上一节中创建的AWS IoT角色别名的 ARN。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "iot:Connect",
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. 根据AWS IoT策略文档创建策略。

- 将 *GreenGrassv2IoT* 替换为要ThingPolicy创建的策略的名称。

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Subscribe\",
          \"iot:Receive\",
          \"iot:Connect\",
          \"greengrass:*\"
        ],
        \"Resource\": [
          \"*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:AssumeRoleWithCertificate\",
        \"Resource\": \"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

创建舰队配置模板

AWS IoT舰队配置模板定义了如何配置AWS IoT内容、策略和证书。要使用队列配置插件配置 Greengrass 核心设备，必须创建一个指定以下内容的模板：

- AWS IoT事物资源。您可以指定现有事物组列表，以便在每台设备联机时将组件部署到每台设备。
- 一种AWS IoT策略资源。此资源可以定义以下属性之一：
  - 现有AWS IoT策略的名称。如果选择此选项，则根据此模板创建的核心设备将使用相同的AWS IoT 策略，并且您可以像队列一样管理它们的权限。
  - 一份AWS IoT政策文件。如果选择此选项，则根据此模板创建的每台核心设备都使用唯一的AWS IoT策略，并且您可以管理每台核心设备的权限。
- AWS IoT证书资源。此证书资源必须使用`AWS::IoT::Certificate::Id`参数将证书附加到核心设备。有关更多信息，请参阅《AWS IoT开发人员指南》中的 J ust-in-time 配置。

在模板中，您可以指定将AWS IoT事物添加到现有事物组列表中。当核心设备首次连接到AWS IoT Greengrass时，它会收到其所属的每个事物组的 Greengrass 部署。您可以使用事物组在每台设备上线后立即将最新的软件部署到每台设备上。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。

在配置设备AWS 账户时，该AWS IoT服务需要权限才能在您中创建和更新AWS IoT资源。要授予AWS IoT服务访问权限，您需要创建一个 IAM 角色并在创建模板时提供该角色。 AWS IoT提供了托管策略 AWSIoTThingsRegistration，允许访问配置设备时AWS IoT可能使用的所有权限。您可以使用此托管策略，也可以创建自定义策略，根据您的用例缩小托管策略中的权限范围。

在本节中，您将创建一个允许AWS IoT为设备配置资源的 IAM 角色，并创建使用该 IAM 角色的队列配置模板。

创建舰队配置模板

1. 创建一个 IAM 角色，该角色AWS IoT可以代入在您的中配置资源AWS 账户。执行以下操作：

   a. 创建一个包含允许AWS IoT担任该角色的信任策略文档的文件。

      例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

      ```
      nano aws-iot-trust-policy.json
      ```

      将以下 JSON 复制到文件中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 使用信任策略文档创建 IAM 角色。

- *GreengrassFleetProvisioningRole*替换为要创建的 IAM 角色的名称。

```
aws iam create-role --role-name GreengrassFleetProvisioningRole --assume-role-policy-document file://aws-iot-trust-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassFleetProvisioningRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassFleetProvisioningRole",
    "CreateDate": "2021-07-26T00:15:12+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
```

```
  }
}
```

c. 查看AWSIoTThingsRegistration策略，该策略允许访问在配置设备时AWS IoT可能使用的所有权限。您可以使用此托管策略，也可以创建自定义策略来为您的用例定义范围缩小权限的权限。如果您选择创建自定义策略，请立即创建。

d. 将 IAM 策略附加到队列配置角色。

- 将 *GreengrassFleetProvisioningRole* 替换为 IAM 角色的名称。
- 如果您在上一步中创建了自定义策略，请将策略 ARN 替换为要使用的 IAM 策略的 ARN。

```
aws iam attach-role-policy --role-name GreengrassFleetProvisioningRole --
policy-arn arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration
```

如果请求成功，则该命令没有任何输出。

2. （可选）创建预配置挂钩，该AWS Lambda功能用于验证设备在注册期间提供的模板参数。您可以使用预配置挂钩来更好地控制您的AWS 账户机载哪些设备以及有多少设备。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的预置挂钩。

3. 创建舰队配置模板。执行以下操作：

a. 创建包含配置模板文档的文件。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano greengrass-fleet-provisioning-template.json
```

撰写配置模板文档。您可以从以下示例配置模板开始，该模板指定创建具有以下属性的AWS IoT事物：

- 事物的名称是您在`ThingName`模板参数中指定的值。
- 事物是您在`ThingGroupName`模板参数中指定的事物组的成员。事物组必须存在于您的中AWS 账户。
- 事物的证书上`GreengrassV2IoTThingPolicy`附有名为的AWS IoT策略。

有关更多信息，请参阅《AWS IoT Core开发人员指南》中的配置模板。

```
{
```

```
  "Parameters": {
    "ThingName": {
      "Type": "String"
    },
    "ThingGroupName": {
      "Type": "String"
    },
    "AWS::IoT::Certificate::Id": {
      "Type": "String"
    }
  },
  "Resources": {
    "MyThing": {
      "OverrideSettings": {
        "AttributePayload": "REPLACE",
        "ThingGroups": "REPLACE",
        "ThingTypeName": "REPLACE"
      },
      "Properties": {
        "AttributePayload": {},
        "ThingGroups": [
          {
            "Ref": "ThingGroupName"
          }
        ],
        "ThingName": {
          "Ref": "ThingName"
        }
      },
      "Type": "AWS::IoT::Thing"
    },
    "MyPolicy": {
      "Properties": {
        "PolicyName": "GreengrassV2IoTThingPolicy"
      },
      "Type": "AWS::IoT::Policy"
    },
    "MyCertificate": {
      "Properties": {
        "CertificateId": {
          "Ref": "AWS::IoT::Certificate::Id"
        },
        "Status": "Active"
      },
```

```
        "Type": "AWS::IoT::Certificate"
      }
    }
  }
```

> **Note**
>
> *MyThingMyPolicy*、和*MyCertificate*是任意名称，用于标识队列置备模板中的每个资源规范。 AWS IoT不会在根据模板创建的资源中使用这些名称。您可以使用这些名称或将其替换为有助于识别模板中每种资源的值。

b. 根据配置模板文档创建队列配置模板。

- *GreengrassFleetProvisioningTemplate*替换为要创建的模板的名称。
- 将模板描述替换为模板的描述。
- 将配置角色 ARN 替换为您之前创建的角色的 ARN。

Linux or Unix

```
aws iot create-provisioning-template \
  --template-name GreengrassFleetProvisioningTemplate \
  --description "A provisioning template for Greengrass core devices." \
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" \
  --template-body file://greengrass-fleet-provisioning-template.json \
  --enabled
```

Windows Command Prompt (CMD)

```
aws iot create-provisioning-template ^
  --template-name GreengrassFleetProvisioningTemplate ^
  --description "A provisioning template for Greengrass core devices." ^
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" ^
  --template-body file://greengrass-fleet-provisioning-template.json ^
  --enabled
```

PowerShell

```
aws iot create-provisioning-template `
  --template-name GreengrassFleetProvisioningTemplate `
  --description "A provisioning template for Greengrass core devices." `
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" `
  --template-body file://greengrass-fleet-provisioning-template.json `
  --enabled
```

> **Note**
>
> 如果您创建了预配置挂钩，请使用参数指定预置挂钩的 Lambda 函数的 ARN。`--pre-provisioning-hook`
>
> ```
> --pre-provisioning-hook targetArn=arn:aws:lambda:us-
> west-2:123456789012:function:GreengrassPreProvisioningHook
> ```

如果请求成功，则响应类似于以下示例。

```
{
    "templateArn": "arn:aws:iot:us-west-2:123456789012:provisioningtemplate/
GreengrassFleetProvisioningTemplate",
    "templateName": "GreengrassFleetProvisioningTemplate",
    "defaultVersionId": 1
}
```

创建配置声明证书和私钥

声明证书是 X.509 证书，允许设备注册为AWS IoT事物并检索用于常规操作的唯一 X.509 设备证书。创建索赔证书后，您可以附加一个AWS IoT策略，允许设备使用该策略来创建唯一的设备证书并使用队列配置模板进行配置。拥有声明证书的设备只能使用您在AWS IoT策略中允许的配置模板进行配置。

在本节中，您将创建申领证书并对其进行配置，以供设备与您在上一节中创建的队列配置模板一起使用。

Important

供应声称私钥应始终受到保护，包括在 Greengrass 核心设备上。我们建议您使用 Amazon CloudWatch 指标和日志来监控是否存在滥用迹象，例如未经授权使用索赔证书来配置设备。如果您发现滥用，请禁用配置声明证书，使其无法用于设备配置。有关更多信息，请参阅 AWS IoT Core 开发人员指南 中的 监控AWS IoT。

为了帮助您更好地管理自己在您的设备中注册的设备数量和设备AWS 账户，您可以在创建队列配置模板时指定预配置挂钩。预配置挂钩是一种验证设备在注册期间提供的模板参数的AWS Lambda功能。例如，您可以创建一个预配置挂钩，根据数据库检查设备 ID，以验证设备是否有权进行置备。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的预置挂钩。

创建置备申请证书和私钥

1. 创建一个文件夹，用于下载索赔证书和私钥。

```
mkdir claim-certs
```

2. 创建并保存用于置备的证书和私钥。 AWS IoT提供由 Amazon 根证书颁发机构 (CA) 签署的客户证书。

Linux or Unix

```
aws iot create-keys-and-certificate \
  --certificate-pem-outfile "claim-certs/claim.pem.crt" \
  --public-key-outfile "claim-certs/claim.public.pem.key" \
  --private-key-outfile "claim-certs/claim.private.pem.key" \
  --set-as-active
```

Windows Command Prompt (CMD)

```
aws iot create-keys-and-certificate ^
  --certificate-pem-outfile "claim-certs/claim.pem.crt" ^
  --public-key-outfile "claim-certs/claim.public.pem.key" ^
  --private-key-outfile "claim-certs/claim.private.pem.key" ^
  --set-as-active
```

PowerShell

```
aws iot create-keys-and-certificate `
```

```
  --certificate-pem-outfile "claim-certs/claim.pem.crt" `
  --public-key-outfile "claim-certs/claim.public.pem.key" `
  --private-key-outfile "claim-certs/claim.private.pem.key" `
  --set-as-active
```

如果请求成功，则响应将包含有关证书的信息。保存证书的 ARN 以备日后使用。

3. 创建并附加一项AWS IoT策略，允许设备使用该证书创建唯一的设备证书，并使用队列配置模板进行配置。以下策略允许访问设备配置 MQTT API。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的[设备配置 MQTT API](#)。

   执行以下操作：

   a. 创建一个包含 Greengrass 核心设备所需的AWS IoT策略文档的文件。

      例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

      ```
      nano greengrass-provisioning-claim-iot-policy.json
      ```

      将以下 JSON 复制到文件中。

      - 将每个*##*实例替换为您设置队列配置AWS 区域的实例。
      - 用您AWS 账户的 ID 替换每个*## ID* 实例。
      - 将的*GreengrassFleetProvisioningTemplate*每个实例替换为您在上一节中创建的队列置备模板的名称。

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": "iot:Connect",
            "Resource": "*"
          },
          {
            "Effect": "Allow",
            "Action": [
              "iot:Publish",
              "iot:Receive"
            ],
      ```

```
      "Resource": [
        "arn:aws:iot:region:account-id:topic/$aws/certificates/create/*",
        "arn:aws:iot:region:account-id:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:region:account-id:topicfilter/$aws/certificates/create/*",
        "arn:aws:iot:region:account-id:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    }
  ]
}
```

b. 根据AWS IoT策略文档创建策略。

- *GreengrassProvisioningClaimPolicy*替换为要创建的策略的名称。

```
aws iot create-policy --policy-name GreengrassProvisioningClaimPolicy --policy-
document file://greengrass-provisioning-claim-iot-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "policyName": "GreengrassProvisioningClaimPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassProvisioningClaimPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Connect\",
        \"Resource\": \"*\"
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": [
```

```
            \"iot:Publish\",
            \"iot:Receive\"
          ],
          \"Resource\": [
            \"arn:aws:iot:region:account-id:topic/$aws/certificates/create/*\",
            \"arn:aws:iot:region:account-id:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
          ]
        },
        {
          \"Effect\": \"Allow\",
          \"Action\": \"iot:Subscribe\",
          \"Resource\": [
            \"arn:aws:iot:region:account-id:topicfilter/$aws/certificates/create/
*\",
            \"arn:aws:iot:region:account-id:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
          ]
        }
      ]
  }",
  "policyVersionId": "1"
}
```

4. 将AWS IoT策略附加到置备声明证书。

   - *GreengrassProvisioningClaimPolicy*替换为要附加的策略的名称。
   - 将目标 ARN 替换为配置声明证书的 ARN。

```
aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --
target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

   如果请求成功，则该命令没有任何输出。

现在，您有了配置声明证书和私钥，设备可以使用这些证书和私钥注册AWS IoT并配置自己为 Greengrass 核心设备。您可以在制造过程中将索赔证书和私钥嵌入设备中，或者在安装 C AWS IoT Greengrass ore 软件之前将证书和密钥复制到设备。有关更多信息，请参阅 安装具有 AWS IoT 队列配置功能的 C AWS IoT Greengrass ore 软件。

配置AWS IoT舰队配置插件

AWS IoT队列配置插件提供了以下配置参数，当你安装带有队列配置的 AWS IoT Greengrass Core 软件时，你可以自定义这些参数。

`rootPath`

用作AWS IoT Greengrass核心软件根目录的文件夹的路径。

`awsRegion`

舰队配置插件用于配置AWS资源的。AWS 区域

`iotDataEndpoint`

您的AWS IoT数据端点AWS 账户。

`iotCredentialEndpoint`

您的AWS IoT凭证端点AWS 账户。

`iotRoleAlias`

指向令牌交换 IAM 角色的角色别名。AWS IoTAWS IoT凭证提供者扮演此角色是为了允许 Greengrass 核心设备与服务进行交互。AWS有关更多信息，请参阅 授权核心设备与AWS服务。

`provisioningTemplate`

用于配置AWS资源的AWS IoT队列配置模板。此模板必须指定以下内容：

- AWS IoT事物资源。您可以指定现有事物组列表，以便在每台设备联机时将组件部署到每台设备。
- AWS IoT策略资源。此资源可以定义以下属性之一：
  - 现有AWS IoT策略的名称。如果选择此选项，则根据此模板创建的核心设备将使用相同的AWS IoT策略，并且您可以像队列一样管理它们的权限。
  - 一份AWS IoT政策文件。如果选择此选项，则根据此模板创建的每台核心设备都使用唯一的AWS IoT策略，并且您可以管理每台核心设备的权限。
- AWS IoT证书资源。此证书资源必须使用`AWS::IoT::Certificate::Id`参数将证书附加到核心设备。有关更多信息，请参阅《AWS IoT开发人员指南》中的 J ust-in-time 配置。

有关更多信息，请参阅《AWS IoT Core开发人员指南》中的配置模板。

`claimCertificatePath`

您在中指定的预配模板的置备声明证书的路径`provisioningTemplate`。有关更多信息，请参阅《AWS IoT Core API 参考》中的 CreateProvisioningClaim。

`claimCertificatePrivateKeyPath`

您在中指定的配置模板的置备声明证书私钥的路径`provisioningTemplate`。有关更多信息，请参阅《AWS IoT Core API 参考》中的 CreateProvisioningClaim。

> **Important**
>
> 供应声称私钥应始终受到保护，包括在 Greengrass 核心设备上。我们建议您使用 Amazon CloudWatch 指标和日志来监控是否存在滥用迹象，例如未经授权使用索赔证书来配置设备。如果您发现滥用，请禁用配置声明证书，使其无法用于设备配置。有关更多信息，请参阅 AWS IoT Core 开发人员指南 中的 监控AWS IoT。
> 为了帮助您更好地管理自己在您的设备中注册的设备数量和设备AWS 账户，您可以在创建队列配置模板时指定预配置挂钩。预配置挂钩是一种验证设备在注册期间提供的模板参数的AWS Lambda功能。例如，您可以创建一个预配置挂钩，根据数据库检查设备 ID，以验证设备是否有权进行置备。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的预置挂钩。

`rootCaPath`

Amazon 根证书颁发机构 (CA) 证书的路径。

`templateParameters`

（可选）要提供给队列置备模板的参数映射。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的预配模板参数部分。

`deviceId`

（可选）当队列配置插件与其创建 MQTT 连接时，用作客户端 ID 的设备标识符。AWS IoT

默认：随机的 UUID。

`mqttPort`

（可选）用于 MQTT 连接的端口。

默认值：8883

`proxyUrl`

（可选）格式的代理服务器的 URL `scheme://userinfo@host:port`。要使用 HTTPS 代理，必须使用队列配置插件的 1.1.0 或更高版本。

- scheme— 方案，必须是http或https。

> **Important**
>
> Greengrass 核心设备必须运行 Greengrass nucleus v2.5.0 或更高版本才能使用 HTTPS 代理。
> 如果您配置 HTTPS 代理，则必须将代理服务器 CA 证书添加到核心设备的 Amazon 根 CA 证书中。有关更多信息，请参阅 使核心设备能够信任 HTTPS 代理。

- userinfo—（可选）用户名和密码信息。如果您在中指定此信息url，Greengrass 核心设备将忽略和字段。username password
- host— 代理服务器的主机名或 IP 地址。
- port—（可选）端口号。如果您未指定端口，则 Greengrass 核心设备将使用以下默认值：
  - http— 80
  - https— 443

proxyUserName

（可选）对代理服务器进行身份验证的用户名。

proxyPassword

（可选）对代理服务器进行身份验证的用户名。

CSRPath

（可选）证书签名请求 (CSR) 文件的路径，用于从 CSR 创建设备证书。有关更多信息，请参阅AWS IoT Core开发者指南中的按声明进行配置。

csrPrivateKey路径

（可选，如果已声明，csrPath则为必填项）用于生成 CSR 的私钥的路径。私钥必须已用于生成 CSR。有关更多信息，请参阅AWS IoT Core开发者指南中的按声明进行配置。

AWS IoT 舰队配置插件更新日志

下表描述了 claim plugin (aws.greengrass.FleetProvisioningByClaim) 在每个版本的 AWS IoT 队列配置中发生的变化。

版本	更改
1.2.1	错误修复和改进 • 修复了 Greengrass nucleus 启动期间舰队配置插件处于离线状态的问题。队列配置插件现在可以无限期地重试 MQTT 连接调用。
1.2.0	错误修复和改进 • 添加了对通过证书签名请求进行设备配置的支持，其中包含可配置的私钥路径。 • 次要修复和改进。
1.1.0	错误修复和改进 • 在 Windows 设备上配置插件时，添加了对其他文件路径格式的支持。 • 添加对 HTTPS 网络代理配置的支持。有关更多信息，请参阅 通过端口 443 或网络代理进行连接 和 使核心设备能够信任 HTTPS 代理。
1.0.0	初始版本。

安装具有自定义资源配置功能的 C AWS IoT Greengrass ore 软件

此功能适用于 Greengrass nucleus 组件的 v2.4.0 及更高版本。

AWS IoT Greengrass Core 软件安装程序提供了 Java 接口，您可以在配置所需 AWS 资源的自定义插件中实现该接口。您可以开发配置插件以使用自定义 X.509 客户端证书或运行其他安装过程不支持的复杂配置步骤。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的创建自己的客户端证书。

要在安装 C AWS IoT Greengrass ore 软件时运行自定义配置插件，需要创建提供给安装程序的 JAR 文件。安装程序运行插件，插件返回一个配置来定义 Greengrass 核心设备的 AWS 资源。安装程序使用此信息在设备上配置 AWS IoT Greengrass Core 软件。有关更多信息，请参阅 开发自定义配置插件。

> **Important**
>
> 在下载 AWS IoT Greengrass 酷睿软件之前，请检查您的核心设备是否满足安装和运行 AWS IoT Greengrass 酷睿软件 v2.0 的要求。

主题

- 先决条件
- 设置设备环境
- 下载 AWS IoT Greengrass 核心软件
- 安装 AWS IoT Greengrass 核心软件
- 开发自定义配置插件

先决条件

要使用自定义配置安装 AWS IoT Greengrass 核心软件，您必须具备以下条件：

- 用于实现的自定义置备插件的 JAR 文件`DeviceIdentityInterface`。自定义配置插件必须返回每个系统和 nucleus 配置参数的值。否则，您必须在安装过程中在配置文件中提供这些值。有关更多信息，请参阅 开发自定义配置插件。

设置设备环境

按照本节中的步骤设置要用作 AWS IoT Greengrass 核心设备的 Linux 或 Windows 设备。

设置 Linux 设备

设置 Linux 设备用于 AWS IoT Greengrass V2

1. 安装 Java 运行时， AWS IoT Greengrass 核心软件需要运行该运行时。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。以下命令向您展示了如何在您的设备上安装 OpenJDK。

   - 对于基于 Debian 或基于 Ubuntua 的发行版：

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

   - 对于基于 Red Hat 的发行版：

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

   - 对于 Amazon Linux 2：

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

- 对于亚马逊 Linux 2023：

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

安装完成后，运行以下命令以验证 Java 是否在您的 Linux 设备上运行。

```
java -version
```

该命令会打印设备上运行的 Java 版本。例如，在基于 Debian 的发行版上，输出可能与以下示例类似。

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

2. （可选）创建在设备上运行组件的默认系统用户和组。您也可以选择让 AWS IoT Greengrass 核心软件安装程序在安装过程中使用安装程序参数创建此用户和组。`--component-default-user`有关更多信息，请参阅 安装程序参数。

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

3. 验证运行 AWS IoT Greengrass Core 软件的用户（通常`root`）是否有权`sudo`与任何用户和任何组一起运行。

   a. 运行以下命令打开该`/etc/sudoers`文件。

   ```
   sudo visudo
   ```

   b. 验证用户的权限是否如以下示例所示。

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

4. （可选）要运行容器化 Lambda 函数，必须启用 c groups v1，并且必须启用并挂载内存和设备 cgroups。如果您不打算运行容器化 Lambda 函数，则可以跳过此步骤。

要启用这些 cgroups 选项，请使用以下 Linux 内核参数启动设备。

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

有关查看和设置设备内核参数的信息，请参阅操作系统和启动加载程序的文档。按照说明永久设置内核参数。

5. 按照中的要求列表所示，在您的设备上安装所有其他必需的依赖项设备要求。

设置 Windows 设备

Note

此功能适用于 Greengrass nucleus 组件的 2.5.0 及更高版本。

要将 Windows 设备设置为 AWS IoT Greengrass V2

1. 安装 Java 运行时， AWS IoT Greengrass 核心软件需要运行该运行时。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。
2. 检查 PATH 系统变量上是否有 Java 可用，如果没有，请添加它。该 LocalSystem 帐户运行 AWS IoT Greengrass Core 软件，因此您必须将 Java 添加到 PATH 系统变量中，而不是用户的 PATH 用户变量。执行以下操作：
    a. 按下 Windows 键打开开始菜单。
    b. 键入**environment variables**以从 “开始” 菜单中搜索系统选项。
    c. 在开始菜单搜索结果中，选择编辑系统环境变量以打开系统属性窗口。
    d. 选择环境变量... 打开 “环境变量” 窗口。
    e. 在 “系统变量” 下，选择 “路径”，然后选择 “编辑”。在 “编辑环境变量” 窗口中，可以在单独的行上查看每个路径。
    f. 检查 Java 安装bin文件夹的路径是否存在。路径可能与以下示例类似。

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

    g. 如果路径中缺少 Java 安装bin的文件夹，请选择 “新建” 将其添加，然后选择 “确定”。
3. 以管理员身份打开 Windows 命令提示符 (cmd.exe)。
4. 在 Windows 设备上的 LocalSystem 帐户中创建默认用户。将##替换为安全密码。

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

Tip

根据你的 Windows 配置，用户的密码可能会设置为在将来的某个日期过期。为确保您的 Greengrass 应用程序继续运行，请跟踪密码何时过期，并在密码过期之前对其进行更新。您也可以将用户的密码设置为永不过期。

- 要检查用户及其密码何时过期，请运行以下命令。

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

- 要将用户的密码设置为永不过期，请运行以下命令。

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

- 如果你使用的是已弃用该wmic命令的 Windows 10 或更高版本，请运行以下 PowerShell 命令。

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

5. 从微软下载该PsExec实用程序并将其安装到设备上。
6. 使用该 PsExec 实用程序将默认用户的用户名和密码存储在 LocalSystem 账户的凭据管理器实例中。将*##*替换为您之前设置的用户密码。

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

如果PsExec License Agreement打开，Accept请选择同意许可并运行命令。

Note

在 Windows 设备上，该 LocalSystem 帐户运行 Greengrass 核心，您必须使用 PsExec 该实用程序在帐户中存储默认用户信息。 LocalSystem 使用凭据管理器应用程序将此信息存储在当前登录用户的 Windows 帐户中，而不是 LocalSystem 帐户中。

下载 AWS IoT Greengrass 核心软件

您可以从以下位置下载最新版本的 AWS IoT Greengrass Core 软件：

- https://d2s8p88vqu9w66.cloudfront.net/releases/ greengrass-nucleus-latest .zip

> **Note**
>
> 您可以从以下位置下载特定版本的 AWS IoT Greengrass Core 软件。将*##*替换为要下载的版本。
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

下载 AWS IoT Greengrass 核心软件

1. 在您的核心设备上，将 AWS IoT Greengrass Core 软件下载到名为的文件中`greengrass-nucleus-latest.zip`。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   下载此软件即表示您同意Greengrass Core 软件许可协议。

2. （可选）验证 Greengrass nucleus 软件签名

> **Note**
>
> 此功能在 Greengrass nucleus 版本 2.9.5 及更高版本中可用。

a. 使用以下命令验证你的 Greengrass 核工件的签名：

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

根据您安装的 JDK 版本，文件名可能有所不同。*jdk17.0.6_10*替换为您安装的 JDK 版本。

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

根据您安装的 JDK 版本，文件名可能有所不同。*jdk17.0.6_10*替换为您安装的 JDK 版本。

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -verify -certs -verbose greengrass-nucleus-latest.zip
```

b. 该`jarsigner`调用会产生指示验证结果的输出。

i. 如果 Greengrass nucleus zip 文件已签名，则输出将包含以下语句：

```
jar verified.
```

ii. 如果 Greengrass nucleus zip 文件未签名，则输出将包含以下语句：

```
jar is unsigned.
```

c. 如果您提供了 Jarsigner `-certs` 选项以及`-verify`和`-verbose`选项，则输出还包括详细的签名者证书信息。

3. 将 AWS IoT Greengrass Core 软件解压缩到设备上的某个文件夹。*GreengrassInstaller*替换为要使用的文件夹。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

4. （可选）运行以下命令以查看 AWS IoT Greengrass Core 软件的版本。

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> 如果您安装了 v2.4.0 之前的 Greengrass nucleus 版本，则在安装 Core 软件后请勿删除此文件夹。 AWS IoT Greengrass C AWS IoT Greengrass ore 软件使用此文件夹中的文件来运行。如果您下载的是最新版本的软件，则需要安装 v2.4.0 或更高版本，并且可以在安装 C AWS IoT Greengrass ore 软件后删除此文件夹。

安装 AWS IoT Greengrass 核心软件

使用指定以下操作的参数运行安装程序：

- 从部分配置文件进行安装，该文件指定使用您的自定义配置插件来配置 AWS 资源。 AWS IoT Greengrass Core 软件使用配置文件来指定设备上每个 Greengrass 组件的配置。安装程序根据您提供的部分配置文件和自定义配置插件创建的 AWS 资源创建完整的配置文件。

- 指定使用`ggc_user`系统用户在核心设备上运行软件组件。在 Linux 设备上，此命令还指定使用`ggc_group`系统组，安装程序会为您创建系统用户和组。
- 将 AWS IoT Greengrass Core 软件设置为启动时运行的系统服务。在 Linux 设备上，这需要 Systemd 初始化系统。

Important

在 Windows 核心设备上，必须将 AWS IoT Greengrass 核心软件设置为系统服务。

有关您可以指定的参数的更多信息，请参阅安装程序参数。

Note

如果您在内存有限的设备 AWS IoT Greengrass 上运行，则可以控制 AWS IoT Greengrass 酷睿软件使用的内存量。要控制内存分配，您可以在 nucleus 组件的`jvmOptions`配置参数中设置 JVM 堆大小选项。有关更多信息，请参阅 使用 JVM 选项控制内存分配。

安装 AWS IoT Greengrass 核心软件 (Linux)

1. 检查 AWS IoT Greengrass 核心软件的版本。

   - *GreengrassInstaller*替换为包含该软件的文件夹的路径。

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. 使用文本编辑器创建名为的配置文件`config.yaml`以提供给安装程序。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

   ```
   nano GreengrassInstaller/config.yaml
   ```

   将以下 YAML 内容复制到文件中。

   ```
   ---
   system:
     rootpath: "/greengrass/v2"
   ```

```
  # The following values are optional. Return them from the provisioning plugin or
 set them here.
  # certificateFilePath: ""
  # privateKeyPath: ""
  # rootCaPath: ""
  # thingName: ""
services:
  aws.greengrass.Nucleus:
    version: "2.12.3"
    configuration:
      # The following values are optional. Return them from the provisioning plugin
 or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

然后执行以下操作：

- 将 *2.12.3* 替换为 AWS IoT Greengrass 核心软件版本。
- 将的*/greengrass/v2*每个实例替换为 Greengrass 根文件夹。
- （可选）指定系统和核配置值。如果您的配置插件不提供这些值，则必须设置这些值。
- （可选）指定要提供给您的配置插件的配置参数。

> **Note**
>
> 在此配置文件中，您可以自定义其他配置选项，例如要使用的端口和网络代理，如以下示例所示。有关更多信息，请参阅 Greengrass 核配置。
>
> ```
> ---
> system:
>   rootpath: "/greengrass/v2"
>   # The following values are optional. Return them from the provisioning
>  plugin or set them here.
>   # certificateFilePath: ""
>   # privateKeyPath: ""
> ```

```
  # rootCaPath: ""
  # thingName: ""
services:
  aws.greengrass.Nucleus:
    version: "2.12.3"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
      # The following values are optional. Return them from the provisioning
 plugin or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

3. 运行安装程序。指定`--trusted-plugin`提供您的自定义配置插件，`--init-config`并指定提供配置文件。

   - 用 Greengrass 根文件夹替换*/greengrass/v2*或 *C:\greengrass\v2*。
   - 将的*GreengrassInstaller*每个实例替换为解压安装程序所在的文件夹。
   - 将自定义配置插件 JAR 文件的路径替换为插件 JAR 文件的路径。

   Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
```

```
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

Important

在 Windows 核心设备上，`--setup-system-service true`必须指定将 AWS IoT Greengrass 核心软件设置为系统服务。

如果您指定`--setup-system-service true`，则安装程序在将软件设置为系统服务并运行时会打印`Successfully set up Nucleus as a system service`出来。否则，如果安装程序成功安装了软件，则不会输出任何消息。

Note

在没有`deploy-dev-tools`参数的情况下运行安装程序时，不能使用`--provision true`参数来部署本地开发工具。有关直接在您的设备上部署 Greengrass CLI 的信息，请参阅。[Greengrass 命令行界面](#)

4. 通过查看根文件夹中的文件来验证安装。

Linux or Unix

```
ls /greengrass/v2
```

Windows Command Prompt (CMD)

```
dir C:\greengrass\v2
```

PowerShell

```
ls C:\greengrass\v2
```

如果安装成功，则根文件夹包含多个文件夹，例如`config``packages`、和`logs`。

如果您将 AWS IoT Greengrass Core 软件作为系统服务安装，则安装程序会为您运行该软件。否则，必须手动运行该软件。有关更多信息，请参阅 运行AWS IoT Greengrass核心软件。

有关如何配置和使用软件的更多信息 AWS IoT Greengrass，请参阅以下内容：

- 配置 AWS IoT Greengrass 核心软件
- 开发AWS IoT Greengrass组件
- 将AWS IoT Greengrass组件部署到设备
- Greengrass 命令行界面

开发自定义配置插件

要开发自定义配置插件，请创建一个 Java 类来实现`com.aws.greengrass.provisioning.DeviceIdentityInterface`界面。您可以在项目中包含 Greengrass Nucleus JAR 文件来访问此接口及其类。此接口定义了一种输入插件配置并输出置备配置的方法。配置配置定义了系统的配置和Greengrass Core 组件. 这些区域有：AWS IoT Greengrass核心软件安装程序使用此配置配置来配置AWS IoT Greengrass设备上的核心软件。

开发自定义配置插件后，将其构建为 JAR 文件，您可以提供给AWS IoT Greengrass核心软件安装程序在安装过程中运行插件。安装程序在安装程序使用的同一 JVM 中运行自定义配置插件，因此您可以创建仅包含插件代码的 JAR。

这些区域有：AWS IoT队列预置插件实现DeviceIdentityInterface在安装期间使用队列预置。队列配置插件是开源的，因此您可以浏览其源代码以查看如何使用 Provisioning 插件界面的示例。有关更多信息，请参阅 。AWS IoT队列预置插件（位于 GitHub 上）。

主题

- 要求
- 实现 DeviceIdentityInterface 界面

要求

要开发自定义配置插件，必须创建符合以下要求的 Java 类：

- 使用`com.aws.greengrass`包裹，或者内的包裹`com.aws.greengrass`程序包。
- 有没有任何参数的构造函数。
- 实现`DeviceIdentityInterface`界面。有关更多信息，请参阅 实现 DeviceIdentityInterface 界面。

实现 DeviceIdentityInterface 界面

使用`com.aws.greengrass.provisioning.DeviceIdentityInterface`在自定义插件中的界面中，将 Greengrass Nucleus 作为项目的依赖项添加。

使用 DeviceIdentityInterface 在自定义配置插件项目中

- 您可以将 Greengrass Nucleus JAR 文件添加为库，或者添加 Greengrass 核心作为 Maven 依赖项。请执行下列操作之一：
  - 要将 Greengrass Nucleus JAR 文件添加为库，请下载AWS IoT Greengrass核心软件，其中包含 Greengrass 核心 JAR。您可以下载最新版本的AWS IoT Greengrass来自以下位置的核心软件：
    - https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

    您可以找到 Greengrass Core JAR 文件（`Greengrass.jar`) 在`lib`ZIP 文件中的文件夹。将此 JAR 文件添加到项目中。

- 要在 Maven 项目中消耗 Greengrass 核心，请在nucleus中的神器com.aws.greengrass组中)。您还必须添加greengrass-common存储库，因为 Greengrass 核心在 Maven 中央存储库中不可用。

```
<project ...>
    ...
    <repositories>
        <repository>
            <id>greengrass-common</id>
            <name>greengrass common</name>
            <url>https://d2jrmugq4soldf.cloudfront.net/snapshots</url>
        </repository>
    </repositories>
    ...
    <dependencies>
        <dependency>
            <groupId>com.aws.greengrass</groupId>
            <artifactId>nucleus</artifactId>
            <version>2.5.0-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

接口设备 Identity

这些区域有：com.aws.greengrass.provisioning.DeviceIdentityInterface界面具有以下形态。

> **Note**
>
> 您还可以在com.aws.greengrass.配置程序包的Greengrass Core 源代码（位于 GitHub 上）。

```
public interface com.aws.greengrass.provisioning.DeviceIdentityInterface {
    ProvisionConfiguration updateIdentityConfiguration(ProvisionContext context)
            throws RetryableProvisioningException, InterruptedException;

    // Return the name of the plugin.
    String name();
```

```
}

com.aws.greengrass.provisioning.ProvisionConfiguration {
    SystemConfiguration systemConfiguration;
    NucleusConfiguration nucleusConfiguration
}

com.aws.greengrass.provisioning.ProvisionConfiguration.SystemConfiguration {
    String certificateFilePath;
    String privateKeyPath;
    String rootCAPath;
    String thingName;
}

com.aws.greengrass.provisioning.ProvisionConfiguration.NucleusConfiguration {
    String awsRegion;
    String iotCredentialsEndpoint;
    String iotDataEndpoint;
    String iotRoleAlias;
}

com.aws.greengrass.provisioning.ProvisioningContext {
    Map<String, Object> parameterMap;
    String provisioningPolicy;  // The policy is always "PROVISION_IF_NOT_PROVISIONED".
}

com.aws.greengrass.provisioning.exceptions.RetryableProvisioningException {}
```

中的每个配置值`SystemConfiguration`和`NucleusConfiguration`需要安装AWS IoT Greengrass核心软件，但你可以返回`null`. 如果你的自定义配置插件返回`null`对于任何配置值，在创建`config.yaml`要提供给AWS IoT Greengrass核心软件安装程序。如果您的自定义配置插件为您也在中定义的选项返回非空值`config.yaml`，然后安装程序将替换中的值`config.yaml`使用插件返回的值。

安装程序参数

AWS IoT GreengrassCore 软件包括一个安装程序，用于设置软件并配置 Greengrass 核心设备运行所需的AWS资源。安装程序包含以下参数，您可以指定这些参数来配置安装：

`-h`, `--help`

（可选）显示安装程序的帮助信息。

`--version`

（可选）显示AWS IoT Greengrass核心软件的版本。

`-Droot`

（可选）用作AWS IoT Greengrass核心软件根目录的文件夹路径。

> **Note**
>
> 此参数设置 JVM 属性，因此您必须在运行安装程序之前-`jar`指定该属性。例如，指定 `java -Droot="/greengrass/v2" -jar /path/to/Greengrass.jar`。

默认值：

- Linux：`~/.greengrass`
- Windows：`%USERPROFILE%/.greengrass`

`-ar`, `--aws-region`

AWS IoT Greengrass核心软件用来检索或创建其所需AWS资源的。AWS 区域

`-p`, `--provision`

（可选）您可以将此设备注册为AWS IoT事物，并配置核心设备所需的AWS资源。如果您指定`true`，则AWS IoT Greengrass核心软件会预置一个AWS IoT事物、（可选）一个AWS IoT事物组、一个 IAM 角色和一个AWS IoT角色别名。

默认值：`false`

`-tn`, `--thing-name`

（可选）您注册为该核心设备的设备名称。AWS IoT如果你的名字中不存在这个名字AWS 账户，那么 AWS IoT Greengrass Core 软件就会创建它。

> **Note**
>
> 事物名称不能包含冒号 (`:`) 字符。

必须指定`--provision true`才能应用此参数。

默认：`GreengrassV2IotThing_`加上一个随机的 UUID。

`-tgn`, `--thing-group-name`

（可选）您在其中添加此核心设备AWS IoT的事AWS IoT物组的名称。如果部署以该事物组为目标，则该核心设备在连接到时会收到该部署AWS IoT Greengrass。如果您中不存在具有此名称的事物组AWS 账户，则 AWS IoT Greengrass Core 软件会创建它。

> **Note**
>
> 事物组名称不能包含冒号 (`:`) 字符。

必须指定`--provision true`才能应用此参数。

`-tpn`, `--thing-policy-name`

此功能适用于 Greengrass nucleus 组件的 v2.4.0 及更高版本。

（可选）要附加到该核心设备AWS IoT的事物证书的AWS IoT策略名称。如果您的AWS IoT策略中不存在带有此名称的策略AWS 账户，则由 AWS IoT Greengrass Core 软件创建该策略。

默认情况下，AWS IoT GreengrassCore 软件会创建宽松AWS IoT策略。您可以缩小此策略的范围，也可以创建自定义策略来限制用例的权限。有关更多信息，请参阅 AWS IoT Greengrass V2核心设备的最低AWS IoT政策。

必须指定`--provision true`才能应用此参数。

默认值：`GreengrassV2IoTThingPolicy`

`-trn`, `--tes-role-name`

（可选）用于获取允许核心设备与AWS服务交互的AWS证书的 IAM 角色的名称。如果您的角色中不存在具有此名称的角色AWS 账户，则AWS IoT Greengrass核心软件会使用`GreengrassV2TokenExchangeRoleAccess`策略创建该角色。此角色无权访问托管组件工件的 S3 存储桶。因此，在创建组件时，必须为工件的 S3 存储桶和对象添加权限。有关更多信息，请参阅 授权核心设备与AWS服务。

必须指定`--provision true`才能应用此参数。

默认值：`GreengrassV2TokenExchangeRole`

`-tra`, `--tes-role-alias-name`

（可选）指向为该AWS IoT核心设备提供AWS证书的 IAM 角色的角色别名的名称。如果您的中不存在具有此名称的角色别名AWS 账户，则 AWS IoT Greengrass Core 软件会创建该别名并将其指向您指定的 IAM 角色。

必须指定`--provision true`才能应用此参数。

默认值：`GreengrassV2TokenExchangeRoleAlias`

`-ss`, `--setup-system-service`

（可选）您可以将 AWS IoT Greengrass Core 软件设置为在该设备启动时运行的系统服务。系统服务名称是`greengrass`。有关更多信息，请参阅 将 Greengrass 核心配置为系统服务。

在 Linux 操作系统上，此参数要求设备上有 systemd 初始化系统。

> **Important**
>
> 在 Windows 核心设备上，必须将AWS IoT Greengrass核心软件设置为系统服务。

默认值：`false`

`-u`, `--component-default-user`

C AWS IoT Greengrass ore 软件用于运行组件的用户的名称或 ID。例如，您可以指定 **`ggc_user`**。在 Windows 操作系统上运行安装程序时，必须使用此值。

在 Linux 操作系统上，您也可以选择指定组。指定用冒号分隔的用户和组。例如，**`ggc_user:ggc_group`**。

以下其他注意事项适用于 Linux 操作系统：

- 如果您以 root 身份运行，则默认组件用户是在配置文件中定义的用户。如果配置文件未定义用户，则默认为`ggc_user:ggc_group`。如果存在`ggc_user`或`ggc_group`不存在，则由软件创建它们。
- 如果您以非 root 用户身份运行，则 AWS IoT Greengrass Core 软件将使用该用户来运行组件。
- 如果您未指定组，则 AWS IoT Greengrass Core 软件将使用系统用户的主组。

有关更多信息，请参阅 配置运行组件的用户。

-d, `--deploy-dev-tools`

（可选）您可以将 Greengrass CLI 组件下载并部署到该核心设备上。您可以使用此工具在此核心设备上开发和调试组件。

> **Important**
>
> 我们建议您仅在开发环境中使用此组件，而不是在生产环境中使用。此组件提供对生产环境中通常不需要的信息和操作的访问。遵循最低权限原则，将此组件仅部署到需要的核心设备。

必须指定`--provision true`才能应用此参数。

默认值：`false`

`-init`, `--init-config`

（可选）用于安装AWS IoT Greengrass核心软件的配置文件的路径。例如，您可以使用此选项来设置具有特定 nucleus 配置的新核心设备。

> **Important**
>
> 您指定的配置文件与核心设备上的现有配置文件合并。这包括核心设备上的组件和组件配置。我们建议配置文件仅列出您正在尝试更改的配置。

`-tp`, `--trusted-plugin`

（可选）要作为可信插件加载的 JAR 文件的路径。使用此选项提供配置插件 JAR 文件，例如使用队列配置或自定义配置进行安装，或者使用私钥和证书安装在硬件安全模块中。

`-s`, `--start`

（可选）您可以在安装AWS IoT Greengrass核心软件后启动它，也可以选择配置资源。

默认值：`true`

运行AWS IoT Greengrass核心软件

安装 AWS IoT Greengrass Core 软件后，运行它以将您的设备连接到AWS IoT Greengrass。

在安装 AWS IoT Greengrass Core 软件时，可以指定是否将其作为系统服务与 systemd 一起安装。如果您选择此选项，则安装程序会为您运行软件，并将其配置为在设备启动时运行。

Important

在 Windows 核心设备上，必须将AWS IoT Greengrass核心软件设置为系统服务。

主题

- 检查 AWS IoT Greengrass Core 软件是否作为系统服务运行
- 将 AWS IoT Greengrass Core 软件作为系统服务运行
- 在没有系统服务的情况下运行 C AWS IoT Greengrass ore 软件

检查 AWS IoT Greengrass Core 软件是否作为系统服务运行

安装 AWS IoT Greengrass Core 软件时，可以指定将AWS IoT Greengrass核心软件安装为系统服务的`--setup-system-service true`参数。Linux 设备需要 systemd 初始化系统才能将AWS IoT Greengrass核心软件设置为系统服务。如果您使用此选项，则安装程序会为您运行该软件，并将其配置为在设备启动时运行。如果安装程序成功将 C AWS IoT Greengrass ore 软件安装为系统服务，则会输出以下消息。

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

如果您之前安装了 AWS IoT Greengrass Core 软件但没有安装程序输出，则可以检查该软件是否作为系统服务安装。

检查 AWS IoT Greengrass Core 软件是否作为系统服务安装

- 运行以下命令来检查 Greengrass 系统服务的状态。

  Linux or Unix (systemd)

  ```
  sudo systemctl status greengrass.service
  ```

  如果 C AWS IoT Greengrass ore 软件作为系统服务安装并处于活动状态，则响应类似于以下示例。

  ```
  # greengrass.service - Greengrass Core
  ```

```
   Loaded: loaded (/etc/systemd/system/greengrass.service; enabled; vendor
 preset: disabled)
   Active: active (running) since Thu 2021-02-11 01:33:44 UTC; 4 days ago
 Main PID: 16107 (sh)
   CGroup: /system.slice/greengrass.service
           ##16107 /bin/sh /greengrass/v2/alts/current/distro/bin/loader
           ##16111 java -Dlog.store=FILE -Droot=/greengrass/v2 -jar /greengrass/
v2/alts/current/distro/lib/Greengrass...
```

如果greengrass.service未找到systemctl或未找到，则 AWS IoT Greengrass Core 软件不会作为系统服务安装。要运行该软件，请参阅在没有系统服务的情况下运行 C AWS IoT Greengrass ore 软件。

Windows Command Prompt (CMD)

```
sc query greengrass
```

如果 C AWS IoT Greengrass ore 软件作为 Windows 服务安装并处于活动状态，则响应类似于以下示例。

```
SERVICE_NAME: greengrass
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

PowerShell

```
Get-Service greengrass
```

如果 C AWS IoT Greengrass ore 软件作为 Windows 服务安装并处于活动状态，则响应类似于以下示例。

```
Status   Name               DisplayName
------   ----               -----------
Running  greengrass         greengrass
```

将 AWS IoT Greengrass Core 软件作为系统服务运行

如果将 AWS IoT Greengrass Core 软件作为系统服务安装，则可以使用系统服务管理器启动、停止和管理该软件。有关更多信息，请参阅 将 Greengrass 核心配置为系统服务。

运行AWS IoT Greengrass核心软件

- 运行以下命令以启动 AWS IoT Greengrass Core 软件。

  Linux or Unix (systemd)

  ```
  sudo systemctl start greengrass.service
  ```

  Windows Command Prompt (CMD)

  ```
  sc start greengrass
  ```

  PowerShell

  ```
  Start-Service greengrass
  ```

在没有系统服务的情况下运行 C AWS IoT Greengrass ore 软件

在 Linux 核心设备上，如果AWS IoT Greengrass核心软件未作为系统服务安装，则可以运行该软件的加载器脚本来运行该软件。

在没有系统服务的情况下运行 AWS IoT Greengrass Core 软件

- 运行以下命令以启动 AWS IoT Greengrass Core 软件。如果您在终端中运行此命令，则必须保持终端会话处于打开状态，以保持 AWS IoT Greengrass Core 软件的运行。
  - 将*/greengrass/v2*或 *C:\greengrass\v2* 替换为你使用的 Greengrass 根文件夹。

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

  如果软件成功启动，则会打印以下消息。

```
Launched Nucleus successfully.
```

在 Docker 容器中运行 AWS IoT Greengrass 核心软件

AWS IoT Greengrass 可以配置为在 Docker 容器中运行。Docker 是一个平台，可为您提供构建、运行、测试和部署基于 Linux 容器的应用程序的工具。运行 AWS IoT Greengrass Docker 镜像时，您可以选择是否向 Docker 容器提供 AWS 凭据，并允许 C AWS IoT Greengrass ore 软件安装程序自动配置 Greengrass 核心设备运行 AWS 所需的资源。如果您不想提供 AWS 凭证，则可以手动配置 AWS 资源并在 Docker 容器中运行 C AWS IoT Greengrass ore 软件。

主题

- 支持的平台和要求
- AWS IoT Greengrass Docker 软件下载
- 选择如何配置 AWS 资源
- 从 Dockerfile 构建AWS IoT Greengrass容器镜像
- AWS IoT Greengrass在具有自动资源配置功能的 Docker 容器中运行
- AWS IoT Greengrass在 Docker 容器中运行，手动配置资源
- 对 Docker 容器中的 AWS IoT Greengrass 执行问题排查

支持的平台和要求

主机必须满足以下最低要求才能在 Docker 容器中安装和运行 C AWS IoT Greengrass ore 软件：

- 基于Linux的操作系统，可连接互联网。
- Docker Engin e 版本 18.09 或更高版本。
- （可选）Docker Compose 版本 1.22 或更高版本。只有当你想使用 Docker Compose CLI 来运行 Docker 镜像时，才需要 Docker Compose。

要在 Docker 容器内运行 Lambda 函数组件，必须配置容器以满足其他要求。有关更多信息，请参阅Lambda 函数要求。

在处理模式下运行组件

AWS IoT Greengrass 不支持在 Docker 容器内的隔离运行时环境中运行 Lambda 函数或 AWS提供的组件。 AWS IoT Greengrass 您必须在没有任何隔离的情况下在进程模式下运行这些组件。

配置 Lambda 函数组件时，请将隔离模式设置为 “无容器”。有关更多信息，请参阅运行AWS Lambda 函数。

部署以下任何 AWS提供的组件时，请更新要将`containerMode`参数设置为的每个组件的配置。`NoContainer`有关配置更新的更多信息，请参阅更新组件配置。

- CloudWatch 指标
- Device Defender
- Firehose
- modbus-RTU 协议适配器
- Amazon SNS

AWS IoT Greengrass Docker 软件下载

AWS IoT Greengrass 提供了 Dockerfile 来构建在亚马逊 Linux 2 (x86_64) 基础映像上安装了 AWS IoT Greengrass 核心软件和依赖项的容器镜像。您可以修改 Dockerfile 中的基础映像，使其在不同的平台架构 AWS IoT Greengrass 上运行。

从中下载 Dockerfile 软件包。GitHub

Dockerfile 使用的是旧版本的 Greengrass。你应该更新文件以使用你想要的 Greengrass 版本。有关从 Dockerfile 构建 AWS IoT Greengrass 容器镜像的信息，请参阅。从 Dockerfile 构建AWS IoT Greengrass容器镜像

选择如何配置 AWS 资源

在 Docker 容器中安装 AWS IoT Greengrass 核心软件时，您可以选择是自动配置 Greengrass 核心设备运行所需的 AWS 资源，还是使用手动配置的资源。

- 自动资源预置-当您首次运行 AWS IoT Greengrass 容器映像时，安装程序会预置 AWS IoT 事物、事物组、IAM AWS IoT 角色和角色别名。 AWS IoT 安装程序还可以将本地开发工具部署到核心设备，因此您可以使用该设备开发和测试自定义软件组件。要自动配置这些资源，您必须向 Docker 镜像提供 AWS 凭证作为环境变量。

要使用自动配置，必须设置 Docker 环境变量`PROVISION=true`并挂载凭证文件以向容器提供您的 AWS 凭据。

- 手动资源配置-如果您不想为容器提供 AWS 凭证，则可以在运行 AWS IoT Greengrass 容器映像之前手动配置 AWS 资源。您必须创建配置文件，以便向 Docker 容器中的 C AWS IoT Greengrass ore 软件安装程序提供有关这些资源的信息。

  要使用手动配置，必须设置 Docker 环境变量`PROVISION=false`。手动配置是默认选项。

有关更多信息，请参阅 从 Dockerfile 构建AWS IoT Greengrass容器镜像。

从 Dockerfile 构建AWS IoT Greengrass容器镜像

AWS提供了一个 Dockerfile，你可以下载并使用它来在 Docker AWS IoT Greengrass 容器中运行核心软件。Dockerfiles 包含用于构建AWS IoT Greengrass容器镜像的源代码。

在构建AWS IoT Greengrass容器镜像之前，必须配置 Dockerfile 以选择要安装的AWS IoT Greengrass 核心软件版本。您还可以配置环境变量以选择在安装过程中如何配置资源，并自定义其他安装选项。本节介绍如何从 Dockerfile 配置和构建 AWS IoT Greengrass Docker 镜像。

下载 Dockerfile 软件包

你可以从以下网址下载 AWS IoT Greengrass Dockerfile 软件包： GitHub

AWS Greengrass Docker 存储库

下载软件包后，将内容解压缩到计算机上的`*download-directory/*aws-greengrass-docker-*nucleus-version*`文件夹。Dockerfile 使用的是旧版本的 Greengrass。你应该更新文件以使用你想要的 Greengrass 版本。

指定AWS IoT Greengrass核心软件版本

在 Dockerfile 中使用以下构建参数来指定要在 Doc AWS IoT Greengrass ker 镜像中使用的AWS IoT Greengrass核心软件的版本。默认情况下，Dockerfile 使用最新版本的AWS IoT Greengrass核心软件。

`GREENGRASS_RELEASE_VERSION`

AWS IoT Greengrass核心软件的版本。默认情况下，Dockerfile 会下载 Greengrass 核心的最新可用版本。将该值设置为要下载的 nucleus 版本。

设置环境变量

环境变量使您可以自定义 C AWS IoT Greengrass ore 软件在 Docker 容器中的安装方式。您可以通过多种方式为 AWS IoT Greengrass Docker 镜像设置环境变量。

- 要使用相同的环境变量来创建多个映像，请直接在 Dockerfile 中设置环境变量。
- 如果您使用启动容器，请`docker run`将环境变量作为命令中的参数传递，或者在环境变量文件中设置环境变量，然后将该文件作为参数传递。有关在 Docker 中设置环境变量的更多信息，请参阅 Docker 文档中的环境变量。
- 如果您使用`docker-compose up`启动容器，请在环境变量文件中设置环境变量，然后将该文件作为参数传递。有关在 Compose 中设置环境变量的更多信息，请参阅 Docker 文档。

您可以为 AWS IoT Greengrass Docker 镜像配置以下环境变量。

> **Note**
>
> 不要修改 Dockerfile 中的`TINI_KILL_PROCESS_GROUP`变量。此变量允许转发`SIGTERM`到 PID 组中的所有 PID，以便在 Docker 容器停止时AWS IoT Greengrass核心软件可以正确关闭。

`GGC_ROOT_PATH`

（可选）容器内用作AWS IoT Greengrass核心软件根目录的文件夹的路径。

默认值：`/greengrass/v2`

`PROVISION`

（可选）确定AWS IoT Greengrass核心是否提供AWS资源。

- 如果您指定`true`，AWS IoT GreengrassCore 软件会将容器镜像注册为AWS IoT事物，并配置 Greengrass 核心设备AWS所需的资源。AWS IoT Greengrass核心软件预置AWS IoT事物、（可选）AWS IoT事物组、IAM 角色和AWS IoT角色别名。有关更多信息，请参阅 AWS IoT Greengrass在具有自动资源配置功能的 Docker 容器中运行。
- 如果您指定`false`，则必须创建一个配置文件以提供给 AWS IoT Greengrass Core 安装程序，该文件指定使用您手动创建的AWS资源和证书。有关更多信息，请参阅 AWS IoT Greengrass在 Docker 容器中运行，手动配置资源。

默认值：`false`

AWS_REGION

（可选）AWS IoT Greengrass核心软件用于检索或创建所需AWS资源的。AWS 区域

默认值：us-east-1。

THING_NAME

（可选）您注册为该核心设备的设备名称。AWS IoT如果你的名字中不存在带有这个名字的东西 AWS 账户，AWS IoT GreengrassCore 软件就会创建它。

必须指定PROVISION=true才能应用此参数。

默认：GreengrassV2IotThing_加上一个随机的 UUID。

THING_GROUP_NAME

（可选）在其中添加此核心设备AWS IoT的事物组的名称。AWS IoT如果部署以该事物组为目标，则该设备和该组中的其他核心设备在连接到时会收到该部署AWS IoT Greengrass。如果您中不存在具有此名称的事物组AWS 账户，则 AWS IoT Greengrass Core 软件会创建它。

必须指定PROVISION=true才能应用此参数。

TES_ROLE_NAME

（可选）用于获取允许 Greengrass 核心设备与服务交互的AWS证书的 IAM 角色名称。AWS 如果您的角色中不存在具有此名称的角色AWS 账户，则AWS IoT Greengrass核心软件会使用GreengrassV2TokenExchangeRoleAccess策略创建该角色。此角色无权访问托管组件工件的 S3 存储桶。因此，在创建组件时，必须为工件的 S3 存储桶和对象添加权限。有关更多信息，请参阅 授权核心设备与AWS服务。

默认值：GreengrassV2TokenExchangeRole

TES_ROLE_ALIAS_NAME

（可选）指向为 Gre AWS IoT engrass 核心设备提供AWS证书的 IAM 角色的角色别名的名称。如果您的中不存在具有此名称的角色别名AWS 账户，则 AWS IoT Greengrass Core 软件会创建该别名并将其指向您指定的 IAM 角色。

默认值：GreengrassV2TokenExchangeRoleAlias

COMPONENT_DEFAULT_USER

（可选）AWS IoT Greengrass核心软件用于运行组件的系统用户和组的名称或 ID。指定用户和群组，用冒号分隔。组是可选的。举例来说，可以指定 **ggc_user:ggc_group** 或 **ggc_user**。

- 如果您以 root 身份运行，则默认为配置文件定义的用户和组。如果配置文件未定义用户和群组，则默认为ggc_user:ggc_group。如果存在ggc_user或ggc_group不存在，则由软件创建它们。
- 如果您以非 root 用户身份运行，则 AWS IoT Greengrass Core 软件将使用该用户来运行组件。
- 如果您未指定组，则 AWS IoT Greengrass Core 软件将使用系统用户的主组。

有关更多信息，请参阅 配置运行组件的用户。

DEPLOY_DEV_TOOLS

定义是否在容器镜像中下载和部署 Greengrass CLI 组件。您可以使用 Greengrass CLI 在本地开发和调试组件。

> **Important**
>
> 我们建议您仅在开发环境中使用此组件，而不是在生产环境中使用。此组件提供对生产环境中通常不需要的信息和操作的访问。遵循最低权限原则，将此组件仅部署到需要的核心设备。

默认值：false

INIT_CONFIG

（可选）用于安装AWS IoT Greengrass核心软件的配置文件的路径。例如，您可以使用此选项来设置具有特定核心配置的新 Greengrass 核心设备，或者指定手动配置的资源。必须将配置文件装载到在此参数中指定的路径上。

TRUSTED_PLUGIN

此功能适用于 Greengrass nucleus 组件的 v2.4.0 及更高版本。

（可选）要作为可信插件加载的 JAR 文件的路径。使用此选项提供配置插件 JAR 文件，例如使用队列配置或自定义配置进行安装。

THING_POLICY_NAME

此功能适用于 Greengrass nucleus 组件的 v2.4.0 及更高版本。

（可选）要附加到该核心设备AWS IoT的事物证书的AWS IoT策略名称。如果您的 C AWS IoT Greengrass ore 软件中不存在带有此名称的AWS IoT策略，则会创建AWS 账户该策略。

必须指定PROVISION=true才能应用此参数。

Note

默认情况下，AWS IoT GreengrassCore 软件会创建宽松AWS IoT策略。您可以缩小此策略的范围，也可以创建自定义策略来限制用例的权限。有关更多信息，请参阅 AWS IoT Greengrass V2核心设备的最低AWS IoT政策。

指定要安装的依赖关系

AWS IoT GreengrassDockerfile 中的 RUN 指令为运行AWS IoT Greengrass核心软件安装程序做好了容器环境的准备。您可以自定义在 C AWS IoT Greengrass ore 软件安装程序在 Docker 容器中运行之前安装的依赖项。

构建AWS IoT Greengrass镜像

使用 AWS IoT Greengrass Dockerfile 构建AWS IoT Greengrass容器镜像。你可以使用 Docker CLI 或 Docker Compose CLI 来构建镜像并启动容器。你也可以使用 Docker CLI 来构建镜像，然后使用 Docker Compose 从该镜像启动你的容器。

Docker

1. 在主机上，运行以下命令切换到包含已配置的 Dockerfile 的目录。

```
cd download-directory/aws-greengrass-docker-nucleus-version
```

2. 运行以下命令从 Dockerfile 构建AWS IoT Greengrass容器镜像。

```
sudo docker build -t "platform/aws-iot-greengrass:nucleus-version" ./
```

Docker Compose

1. 在主机上，运行以下命令切换到包含 Dockerfile 和 Compose 文件的目录。

```
cd download-directory/aws-greengrass-docker-nucleus-version
```

2. 运行以下命令以使用 Compose 文件构建AWS IoT Greengrass容器镜像。

```
docker-compose -f docker-compose.yml build
```

您已成功创建AWS IoT Greengrass容器镜像。Docker 镜像安装了AWS IoT Greengrass核心软件。现在，您可以在 Docker 容器中运行AWS IoT Greengrass核心软件。

AWS IoT Greengrass在具有自动资源配置功能的 Docker 容器中运行

本教程向您展示如何使用自动配置的AWS资源和本地开发工具在 Docker 容器中安装和运行 C AWS IoT Greengrass ore 软件。您可以使用此开发环境来探索 Docker 容器中的AWS IoT Greengrass功能。该软件需要AWS凭据才能配置这些资源和部署本地开发工具。

如果您无法向容器提供AWS凭证，则可以预配置核心设备运行所需的AWS资源。您也可以将开发工具部署到核心设备以用作开发设备。这使您能够在运行容器时向设备提供更少的权限。有关更多信息，请参阅 AWS IoT Greengrass在 Docker 容器中运行，手动配置资源。

先决条件

要完成本教程，您需要以下内容。

- AWS 账户。如果没有，请参阅设置一个 AWS 账户。
- 有权为 AWS Greengrass 核心设备配置AWS IoT和 IAM 资源的 IAM 用户。C AWS IoT Greengrass ore 软件安装程序使用您的AWS凭据自动配置这些资源。有关自动配置资源的最低 IAM 策略的信息，请参阅安装程序配置资源的最低 IAM 政策。
- 一张 AWS IoT Greengrass Docker 镜像。你可以从 AWS IoT Greengrass Dockerfile 中生成镜像。
- 运行 Docker 容器的主机必须满足以下要求：
  - 基于Linux的操作系统，可连接互联网。
  - Docker Engin e 版本 18.09 或更高版本。
  - （可选）Docker Compose 版本 1.22 或更高版本。只有当你想使用 Docker Compose CLI 来运行 Docker 镜像时，才需要 Docker Compose。

配置 AWS凭证

在此步骤中，您将在主机上创建一个包含您的AWS安全凭据的凭据文件。运行 AWS IoT Greengrass Docker 镜像时，必须将包含此凭证文件的文件夹挂载到 Docker 容器`/root/.aws/`中。AWS IoT Greengrass安装程序使用这些凭据在中配置资源AWS 账户。有关安装程序自动配置资源所需的最低 IAM 策略的信息，请参阅安装程序配置资源的最低 IAM 政策。

1. 检索以下内容之一。

- IAM 用户的长期证书。有关如何检索长期证书的信息，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。
- （推荐）IAM 角色的临时证书。有关如何检索临时证书的信息，请参阅 IAM 用户指南AWS CLI中的使用临时安全证书。

2. 创建一个用于存放凭证文件的文件夹。

```
mkdir ./greengrass-v2-credentials
```

3. 使用文本编辑器在./greengrass-v2-credentials文件夹credentials中创建名为的配置文件。

   例如，你可以运行以下命令来使用 GNU nano 来创建credentials文件。

```
nano ./greengrass-v2-credentials/credentials
```

4. 按以下格式将您的AWS凭证添加到credentials文件中。

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token
 = AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

   仅aws_session_token适用于临时证书。

> **Important**
>
> 启动AWS IoT Greengrass容器后，将凭据文件从主机中删除。如果您不删除凭证文件，则您的AWS凭证将保持挂载在容器内。有关更多信息，请参阅 在容器中运行 AWS IoT Greengrass Core 软件。

创建环境文件

本教程使用环境文件来设置环境变量，这些变量将传递给 Docker 容器内的 C AWS IoT Greengrass ore 软件安装程序。您还可以在docker run命令中使用-e或--env参数在 Docker 容器中设置环境变量，也可以在docker-compose.yml文件中的某个environment块中设置变量。

1. 使用文本编辑器创建名为的环境文件`.env`。

   例如，在基于 Linux 的系统上，您可以运行以下命令以使用 GNU nano 在当前目录`.env`中创建。

   ```
   nano .env
   ```

2. 将以下内容复制到文件中。

   ```
   GGC_ROOT_PATH=/greengrass/v2
   AWS_REGION=region
   PROVISION=true
   THING_NAME=MyGreengrassCore
   THING_GROUP_NAME=MyGreengrassCoreGroup
   TES_ROLE_NAME=GreengrassV2TokenExchangeRole
   TES_ROLE_ALIAS_NAME=GreengrassCoreTokenExchangeRoleAlias
   COMPONENT_DEFAULT_USER=ggc_user:ggc_group
   ```

   然后，替换以下值。

   - */greengrass/v2*。 要用于安装的 Greengrass 根文件夹。您可以使用`GGC_ROOT`环境变量来设置此值。
   - *##*。您创建资源AWS 区域的位置。
   - *MyGreengrassCore*。AWS IoT 事物的名称。如果该东西不存在，则安装程序会创建它。安装程序下载证书以进行身份AWS IoT验证。
   - *MyGreengrassCoreGroup*。AWS IoT事物组的名称。如果事物组不存在，则安装程序会创建该组并将其添加到其中。如果事物组存在且部署处于活动状态，则核心设备将下载并运行部署指定的软件。
   - *GreenGras TokenExchangeRole* sV2。替换为允许 Greengrass 核心设备获取临时证书的 IAM 令牌交换角色的名称。AWS如果该角色不存在，则安装程序会创建该角色并创建并附加名为 *GreenGr TokenExchangeRole* assV2 Access 的策略。有关更多信息，请参阅 [授权核心设备与AWS服务](#)。
   - *GreengrassCoreTokenExchangeRoleAlias*。代币交换角色别名。如果角色别名不存在，安装程序会创建它并将其指向您指定的 IAM 令牌交换角色。有关更多信息，请参阅

Note

您可以将DEPLOY_DEV_TOOLS环境变量设置为true以部署 Greengrass CLI 组件，这样您就可以在 Docker 容器内开发自定义组件。我们建议您仅在开发环境中使用此组件，而不是在生产环境中使用。此组件提供对生产环境中通常不需要的信息和操作的访问。遵循最低权限原则，将此组件仅部署到需要的核心设备。

在容器中运行 AWS IoT Greengrass Core 软件

本教程向您展示如何启动在 Docker 容器中构建的 Docker 镜像。你可以使用 Docker CLI 或 Docker Compose CLI 在 Docker AWS IoT Greengrass 容器中运行核心软件镜像。

Docker

1. 运行以下命令启动 Docker 容器。

```
docker run --rm --init -it --name docker-image \
 -v path/to/greengrass-v2-credentials:/root/.aws/:ro \
 --env-file .env \
 -p 8883 \
 your-container-image:version
```

此示例命令使用以下参数进行 docker 运行：

- `--rm`。 当容器退出时将其清理。
- `--init`。 在容器中使用初始化进程。

  Note

  当你停止 Docker 容器时，需要使用该`--init`参数才能关闭 C AWS IoT Greengrass ore 软件。

- `-it`。 （可选）作为交互式进程在前台运行 Docker 容器。你可以将其替换为以分离模式运行 Docker 容器的-d参数。有关更多信息，请参阅 Docker 文档中的分离与前台。
- `--name`。 运行名为的容器 `aws-iot-greengrass`

- `-v`。 将卷挂载到 Docker 容器中，使配置文件和证书文件可供在容器内AWS IoT Greengrass运行。
- `--env-file`。（可选）指定环境文件以设置将传递给 Docker 容器内的 C AWS IoT Greengrass ore 软件安装程序的环境变量。只有在创建环境文件来设置环境变量时，才需要此参数。如果您没有创建环境文件，则可以直接在 Docker 运行命令中使用`--env`参数设置环境变量。
- `-p`。（可选）将 8883 容器端口发布到主机。如果您想通过 MQTT 进行连接和通信，则需要使用此参数，因为 MQTT 流量AWS IoT Greengrass使用端口 8883。要打开其他端口，请使用其他`-p`参数。

> **Note**
>
> 要以更高的安全性运行 Docker 容器，您可以使用`--cap-drop`和`--cap-add`参数选择性地为容器启用 Linux 功能。有关更多信息，请参阅 Docker 文档中的运行时权限和 Linux 功能。

2. 从主机设备`./greengrass-v2-credentials`上移除凭证。

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> 您之所以删除这些凭证，是因为它们提供了核心设备仅在设置期间才需要的广泛权限。如果您不删除这些凭证，Greengrass 组件和容器中运行的其他进程就可以访问它们。如果您需要向 Greengrass 组件提供AWS凭证，请使用令牌交换服务。有关更多信息，请参阅 与AWS服务互动。

Docker Compose

1. 使用文本编辑器创建名为的 Docker Compose 文件。`docker-compose.yml`

   例如，在基于 Linux 的系统上，您可以运行以下命令以使用 GNU nano 在当前目录`docker-compose.yml`中创建。

```
nano docker-compose.yml
```

> **Note**
>
> 您也可以从中下载和使用AWS提供的 Compose 文件的最新版本。GitHub

2. 将以下内容添加到 Compose 文件中。您的文件应类似于以下示例。将 *docker ##*替换为你的 Docker 镜像的名称。

```
version: '3.7'

services:
  greengrass:
    init: true
    container_name: aws-iot-greengrass
    image: docker-image
    volumes:
      - ./greengrass-v2-credentials:/root/.aws/:ro
    env_file: .env
    ports:
      - "8883:8883"
```

此示例 Compose 文件中的以下参数是可选的：

- `ports`— 将 8883 个容器端口发布到主机。如果您想通过 MQTT 进行连接和通信，则需要使用此参数，因为 MQTT 流量AWS IoT Greengrass使用端口 8883。
- `env_file`— 指定环境文件以设置将传递给 Docker 容器内的 C AWS IoT Greengrass ore 软件安装程序的环境变量。只有在创建环境文件来设置环境变量时，才需要此参数。如果您没有创建环境文件，则可以使用环境参数直接在 Compose 文件中设置变量。

> **Note**
>
> 要以更高的安全性运行 Docker 容器，您可以在 Compose 文件`cap_add`中使用`cap_drop`和来选择性地为容器启用 Linux 功能。有关更多信息，请参阅 Docker 文档中的运行时权限和 Linux 功能。

3. 运行以下命令启动 Docker 容器。

```
docker-compose -f docker-compose.yml up
```

4. 从主机设备./greengrass-v2-credentials上移除凭证。

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
> 您之所以删除这些凭证，是因为它们提供了核心设备仅在设置期间才需要的广泛权限。如果您不删除这些凭证，Greengrass 组件和容器中运行的其他进程就可以访问它们。如果您需要向 Greengrass 组件提供AWS凭证，请使用令牌交换服务。有关更多信息，请参阅与AWS服务互动。

后续步骤

AWS IoT Greengrass核心软件现在在 Docker 容器中运行。运行以下命令以检索当前正在运行的容器的容器 ID。

```
docker ps
```

然后，您可以运行以下命令来访问容器并浏览容器内运行的 AWS IoT Greengrass Core 软件。

```
docker exec -it container-id /bin/bash
```

有关创建简单组件的信息，请参见第 4 步：在设备上开发和测试组件中的 教程：AWS IoT Greengrass V2 入门

> **Note**
> 当你使用docker exec在 Docker 容器内运行命令时，这些命令不会记录在 Docker 日志中。要将您的命令记录在 Docker 日志中，请将交互式外壳附加到 Docker 容器。有关更多信息，请参阅 将交互式外壳附加到 Docker 容器。

C AWS IoT Greengrass ore 日志文件被调用greengrass.log，位于中*/greengrass/v2*/logs。组件日志文件也位于同一目录中。要将 Greengrass 日志复制到主机上的临时目录，请运行以下命令：

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

如果您想在容器退出或移除后保留日志，我们建议您仅将该目录绑定到主机上的临时日志*/greengrass/v2*/logs目录，而不是挂载整个 Greengrass 目录。有关更多信息，请参阅 在 Docker 容器之外保存 Greengrass 日志。

要停止正在运行的 AWS IoT Greengrass Docker 容器，请运行`docker stop`或`docker-compose -f docker-compose.yml stop`。此操作发送`SIGTERM`到 Greengrass 进程，并关闭容器中启动的所有关联进程。Docker 容器使用`docker-init`可执行文件作为进程 PID 1 进行初始化，这有助于删除所有剩余的僵尸进程。有关更多信息，请参阅 Docker 文档中的指定初始化进程。

有关在 Docker 容器AWS IoT Greengrass中运行时遇到的问题疑难解答的信息，请参阅对 Docker 容器中的 AWS IoT Greengrass 执行问题排查。

AWS IoT Greengrass在 Docker 容器中运行，手动配置资源

本教程向您展示如何使用手动配置的资源在 Docker 容器中安装和运行 C AWS IoT Greengrass ore 软件。AWS

主题

- 先决条件
- 检索AWS IoT端点
- 创建 AWS IoT 事物。
- 创建事物证书
- 配置事物证书
- 创建代币交换角色
- 将证书下载到设备
- 创建配置文件
- 创建环境文件
- 在容器中运行 AWS IoT Greengrass Core 软件
- 后续步骤

先决条件

要完成本教程，您需要：

- AWS 账户。如果没有，请参阅设置一个 AWS 账户。
- 一张 AWS IoT Greengrass Docker 镜像。你可以从 AWS IoT Greengrass Dockerfile 中生成镜像。

- 运行 Docker 容器的主机必须满足以下要求：
  - 基于Linux的操作系统，可连接互联网。
  - Docker Engin e 版本 18.09 或更高版本。
  - （可选）Docker Compose 版本 1.22 或更高版本。只有当你想使用 Docker Compose CLI 来运行 Docker 镜像时，才需要 Docker Compose。

检索AWS IoT端点

获取您的终AWS IoT端节点AWS 账户，然后将其保存以备后用。您的设备使用这些端点进行连接AWS IoT。执行以下操作：

1. 获取您的AWS IoT数据端点AWS 账户。

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

   如果请求成功，则响应类似于以下示例。

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 获取您的AWS IoT凭证端点AWS 账户。

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

   如果请求成功，则响应类似于以下示例。

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

创建 AWS IoT 事物。

AWS IoT事物代表连接到的设备和逻辑实体AWS IoT。Greengrass 的核心设备就是东西。AWS IoT当您将设备注册为AWS IoT事物时，该设备可以使用数字证书进行身份验证AWS。

在本节中，您将创建一个代表您的设备的AWS IoT东西。

创建 AWS IoT 事物

1. 为你的设备创建AWS IoT一件东西。在开发计算机上，运行以下命令。

   • *MyGreengrassCore*替换为要使用的事物名称。这个名字也是你的 Greengrass 核心设备的名字。

   > Note
   >
   > 事物名称不能包含冒号 (:) 字符。

   ```
   aws iot create-thing --thing-name MyGreengrassCore
   ```

   如果请求成功，则响应类似于以下示例。

   ```
   {
     "thingName": "MyGreengrassCore",
     "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
     "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
   }
   ```

2. （可选）将AWS IoT事物添加到新的或现有的事物组。您可以使用事物组来管理 Greengrass 核心设备群。将软件组件部署到设备时，可以将单个设备或设备组作为目标。您可以将设备添加到已激活 Greengrass 部署的事物组，以将该事物组的软件组件部署到该设备上。执行以下操作：

   a. （可选）创建AWS IoT事物组。

      • *MyGreengrassCoreGroup*替换为要创建的事物组的名称。

      > Note
      >
      > 事物组名称不能包含冒号 (:) 字符。

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      如果请求成功，则响应类似于以下示例。

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. 将AWS IoT事物添加到事物组。

- *MyGreengrassCore*用你的AWS IoT东西的名字替换。
- *MyGreengrassCoreGroup*替换为事物组的名称。

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

如果请求成功，则该命令没有任何输出。

创建事物证书

当您将设备注册为AWS IoT事物时，该设备可以使用数字证书进行身份验证AWS。此证书允许设备与AWS IoT和通信AWS IoT Greengrass。

在本节中，您将创建和下载设备可用于连接的证书AWS。

创建事物证书

1. 创建一个文件夹，用于下载AWS IoT事物的证书。

```
mkdir greengrass-v2-certs
```

2. 为该AWS IoT事物创建并下载证书。

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
 greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

如果请求成功，则响应类似于以下示例。

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```


保存证书的 Amazon 资源名称 (ARN)，以便稍后用于配置证书。

配置事物证书

将事物证书附加到您之前创建AWS IoT的事物，然后向证书添加AWS IoT策略以定义核心设备的AWS IoT权限。

配置事物的证书

1. 将证书附加到AWS IoT事物上。

   - *MyGreengrassCore*用你的AWS IoT东西的名字替换。
   - 将证书 Amazon 资源名称 (ARN) 替换为您在上一步中创建的证书的 ARN。

   ```
   aws iot attach-thing-principal --thing-name MyGreengrassCore
    --principal arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   如果请求成功，则该命令没有任何输出。

2. 创建并附加用于定义 Greengrass 核心设备AWS IoT权限的AWS IoT策略。以下策略允许访问所有 MQTT 主题和 Greengrass 操作，因此您的设备可以处理需要新 Greengrass 操作的自定义应用程序和未来的更改。您可以根据自己的用例限制此政策。有关更多信息，请参阅 AWS IoT Greengrass V2核心设备的最低AWS IoT政策。

   如果您之前设置过 Greengrass 核心设备，则可以附加AWS IoT其策略，而不必创建新的策略。

   执行以下操作：

   a. 创建一个包含 Greengrass 核心设备所需的AWS IoT策略文档的文件。

      例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

      ```
      nano greengrass-v2-iot-policy.json
      ```

      将以下 JSON 复制到文件中。

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

```
        "Action": [
          "iot:Publish",
          "iot:Subscribe",
          "iot:Receive",
          "iot:Connect",
          "greengrass:*"
        ],
        "Resource": [
          "*"
        ]
      }
    ]
}
```

b. 根据AWS IoT策略文档创建策略。

- 将 *GreenGrassv2IoT* 替换为要ThingPolicy创建的策略的名称。

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
          \\"*\\"
        ]
```

```
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. 将AWS IoT策略附加到AWS IoT事物的证书上。

- 将 *GreenGrassv2IoT* 替换为要ThingPolicy附加的策略的名称。
- 用你的东西的证书的 ARN 替换目标 ARN。AWS IoT

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

如果请求成功，则该命令没有任何输出。

创建代币交换角色

Greengrass 核心设备使用 IAM 服务角色（称为令牌交换角色）来授权对服务的调用。AWS设备使用AWS IoT证书提供者获取此角色的临时AWS证书，从而允许设备与 Amazon Logs 进行交互AWS IoT、向 Amazon Logs 发送 CloudWatch 日志以及从 Amazon S3 下载自定义组件项目。有关更多信息，请参阅 授权核心设备与AWS服务。

您可以使用AWS IoT角色别名为 Greengrass 核心设备配置令牌交换角色。角色别名允许您更改设备的令牌交换角色，但设备配置保持不变。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的授权直接调用AWS服务。

在本节中，您将创建一个令牌交换 IAM 角色和一个指向该AWS IoT角色的角色别名。如果您已经设置了 Greengrass 核心设备，则可以使用其代币交换角色和角色别名，而不必创建新的代币交换角色和角色别名。然后，您将设备配置为AWS IoT使用该角色和别名。

创建代币交换 IAM 角色

1. 创建您的设备可用作令牌交换角色的 IAM 角色。执行以下操作：

   a. 创建包含令牌交换角色所需的信任策略文档的文件。

      例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano device-role-trust-policy.json
```

将以下 JSON 复制到文件中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 使用信任策略文档创建令牌交换角色。

- 将 *GreenGrassV2 TokenExchangeRole* 替换为要创建的 IAM 角色的名称。

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
```

```
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
    }
```

c. 创建包含令牌交换角色所需的访问策略文档的文件。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano device-role-access-policy.json
```

将以下 JSON 复制到文件中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 此访问策略不允许访问 S3 存储桶中的组件项目。要在 Amazon S3 中部署定义构件的自定义组件，您必须向该角色添加权限以允许您的核心设备检索组件工件。有关更多信息，请参阅 允许访问 S3 存储桶以获取组件项目。
> 如果您还没有用于组件工件的 S3 存储桶，则可以在创建存储桶后添加这些权限。

d. 根据策略文档创建 IAM 策略。

- 将 *GreenGrassV2 TokenExchangeRoleAccess* 替换为要创建的 IAM 策略的名称。

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. 将 IAM 策略附加到令牌交换角色。

- 将 *GreenGrassV2* 替换为 IA TokenExchangeRole M 角色的名称。
- 将策略 ARN 替换为您在上一步中创建的 IAM 策略的 ARN。

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

如果请求成功，则该命令没有任何输出。

2. 创建AWS IoT指向代币交换角色的角色别名。

- *GreengrassCoreTokenExchangeRoleAlias*替换为要创建的角色别名的名称。
- 将角色 ARN 替换为您在上一步中创建的 IAM 角色的 ARN。

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

如果请求成功，则响应类似于以下示例。

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> 要创建角色别名，您必须有权将令牌交换 IAM 角色传递给AWS IoT。如果您在尝试创建角色别名时收到错误消息，请检查您的AWS用户是否具有此权限。有关更多信息，请参阅《用户指南》中的授予AWS Identity and Access Management用户向AWS服务传递角色的权限。

3. 创建并附加AWS IoT允许您的 Greengrass 核心设备使用角色别名担任令牌交换角色的策略。如果您之前设置过 Greengrass 核心设备，则可以附加其角色AWS IoT别名策略，而不必创建新的角色别名策略。执行以下操作：

   a. （可选）创建一个包含角色别名所需的AWS IoT策略文档的文件。

      例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

      ```
      nano greengrass-v2-iot-role-alias-policy.json
      ```

      将以下 JSON 复制到文件中。

      - 将资源 ARN 替换为角色别名的 ARN。

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

```
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. 根据AWS IoT策略文档创建策略。

- *GreengrassCoreTokenExchangeRoleAliasPolicy*替换为要创建的AWS IoT策略的名称。

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --policy-document file://greengrass-v2-iot-role-alias-policy.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. 将AWS IoT策略附加到AWS IoT事物的证书上。

- *GreengrassCoreTokenExchangeRoleAliasPolicy*替换为角色别名AWS IoT策略的名称。
- 用你的东西的证书的 ARN 替换目标 ARN。AWS IoT

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

如果请求成功，则该命令没有任何输出。

将证书下载到设备

之前，您已将设备的证书下载到开发计算机上。在本节中，您将下载亚马逊根证书颁发机构 (CA) 证书。然后，如果您计划在 Docker 中与开发计算机不同的计算机上运行 C AWS IoT Greengrass ore 软件，则可以将证书复制到该主机。C AWS IoT Greengrass ore 软件使用这些证书连接到AWS IoT云服务。

将证书下载到设备

1. 在您的开发计算机上，下载 Amazon 根证书颁发机构 (CA) 证书。 AWS IoT默认情况下，证书与亚马逊的根 CA 证书相关联。

   Linux or Unix

   ```
   sudo curl -o ./greengrass-v2-certs/AmazonRootCA1.pem https://
   www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o .\greengrass-v2-certs\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile .
   \greengrass-v2-certs\AmazonRootCA1.pem
   ```

2. 如果您计划在 Docker 中与开发计算机不同的设备上运行 C AWS IoT Greengrass ore 软件，请将证书复制到主机。如果在开发计算机和主机上启用了 SSH 和 SCP，则可以在开发计算机上使用`scp`命令来传输证书。*`device-ip-address`*替换为主机的 IP 地址。

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

创建配置文件

1. 在主机上，创建一个用于存放配置文件的文件夹。

```
mkdir ./greengrass-v2-config
```

2. 使用文本编辑器在`./greengrass-v2-config`文件夹`config.yaml`中创建名为的配置文件。

   例如，你可以运行以下命令来使用 GNU nano 来创建。`config.yaml`

```
nano ./greengrass-v2-config/config.yaml
```

3. 将以下 YAML 内容复制到文件中。此部分配置文件指定了系统参数和 Greengrass 核参数。

```
---
system:
  certificateFilePath: "/tmp/certs/device.pem.crt"
  privateKeyPath: "/tmp/certs/private.pem.key"
  rootCaPath: "/tmp/certs/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "nucleus-version"
    configuration:
      awsRegion: "region"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.region.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.region.amazonaws.com"
```

   然后，替换以下值：

   - *`/tmp/certs`*#Docker 容器中的目录，您在启动容器时将下载的证书挂载到该目录。
   - *`/greengrass/v2`*。 要用于安装的 Greengrass 根文件夹。您可以使用`GGC_ROOT`环境变量来设置此值。
   - *`MyGreengrassCore`*。AWS IoT 事物的名称。

- *######*要安装的AWS IoT Greengrass核心软件的版本。此值必须与您下载的 Docker 镜像或 Dockerfile 的版本相匹配。如果你下载了带有标签的 Greengrass Docker 镜像，`latest`**`docker inspect`** ***`image-id`***请使用查看镜像版本。
- *##*。您创建AWS IoT资源AWS 区域的位置。您还必须在环境文件中为`AWS_REGION`环境变量指定相同的值。
- *`GreengrassCoreTokenExchangeRoleAlias`*。代币交换角色别名。
- *`device-data-prefix`*。您的AWS IoT数据端点的前缀。
- *`device-credentials-prefix`*。您的AWS IoT凭证端点的前缀。

创建环境文件

本教程使用环境文件来设置环境变量，这些变量将传递给 Docker 容器内的 C AWS IoT Greengrass ore 软件安装程序。您还可以在`docker run`命令中使用`-e`或`--env`参数在 Docker 容器中设置环境变量，也可以在`docker-compose.yml`文件中的某个`environment`块中设置变量。

1. 使用文本编辑器创建名为的环境文件`.env`。

   例如，在基于 Linux 的系统上，您可以运行以下命令以使用 GNU nano 在当前目录`.env`中创建。

   ```
   nano .env
   ```

2. 将以下内容复制到文件中。

   ```
   GGC_ROOT_PATH=/greengrass/v2
   AWS_REGION=region
   PROVISION=false
   COMPONENT_DEFAULT_USER=ggc_user:ggc_group
   INIT_CONFIG=/tmp/config/config.yaml
   ```

   然后，替换以下值。

   - *`/greengrass/v2`*。 用于安装 C AWS IoT Greengrass ore 软件的根文件夹路径。
   - *##*。您创建AWS IoT资源AWS 区域的位置。您必须在配置文件中为`awsRegion`配置参数指定相同的值。
   - *`/tmp/config/#`*启动 Docker 容器时用于挂载配置文件的文件夹。

Note

您可以将DEPLOY_DEV_TOOLS环境变量设置为true以部署 Greengrass CLI 组件，这样您就可以在 Docker 容器内开发自定义组件。我们建议您仅在开发环境中使用此组件，而不是在生产环境中使用。此组件提供对生产环境中通常不需要的信息和操作的访问。遵循最低权限原则，将此组件仅部署到需要的核心设备。

在容器中运行 AWS IoT Greengrass Core 软件

本教程向您展示如何启动在 Docker 容器中构建的 Docker 镜像。你可以使用 Docker CLI 或 Docker Compose CLI 在 Docker AWS IoT Greengrass 容器中运行核心软件镜像。

Docker

- 本教程向你展示了如何启动你在 Docker 容器中构建的 Docker 镜像。

```
docker run --rm --init -it --name docker-image \
 -v path/to/greengrass-v2-config:/tmp/config/:ro \
 -v path/to/greengrass-v2-certs:/tmp/certs:ro \
 --env-file .env \
 -p 8883 \
 your-container-image:version
```

此示例命令使用以下参数进行 docker 运行：

- --rm。 当容器退出时将其清理。
- --init。 在容器中使用初始化进程。

Note

当你停止 Docker 容器时，需要使用该--init参数才能关闭 C AWS IoT Greengrass ore 软件。

- -it。 （可选）作为交互式进程在前台运行 Docker 容器。你可以将其替换为以分离模式运行 Docker 容器的-d参数。有关更多信息，请参阅 Docker 文档中的分离与前台。
- --name。 运行名为的容器 aws-iot-greengrass

- `-v`。 将卷挂载到 Docker 容器中，使配置文件和证书文件可供在容器内AWS IoT Greengrass运行。
- `--env-file`。 （可选）指定环境文件以设置将传递给 Docker 容器内的 C AWS IoT Greengrass ore 软件安装程序的环境变量。只有在创建环境文件来设置环境变量时，才需要此参数。如果您没有创建环境文件，则可以直接在 Docker 运行命令中使用`--env`参数设置环境变量。
- `-p`。 （可选）将 8883 容器端口发布到主机。如果您想通过 MQTT 进行连接和通信，则需要使用此参数，因为 MQTT 流量AWS IoT Greengrass使用端口 8883。要打开其他端口，请使用其他`-p`参数。

> **Note**
>
> 要以更高的安全性运行 Docker 容器，您可以使用`--cap-drop`和`--cap-add`参数选择性地为容器启用 Linux 功能。有关更多信息，请参阅 Docker 文档中的运行时权限和 Linux 功能。

Docker Compose

1. 使用文本编辑器创建名为的 Docker Compose 文件。`docker-compose.yml`

   例如，在基于 Linux 的系统上，您可以运行以下命令以使用 GNU nano 在当前目录`docker-compose.yml`中创建。

```
nano docker-compose.yml
```

> **Note**
>
> 您也可以从中下载和使用AWS提供的 Compose 文件的最新版本。GitHub

2. 将以下内容添加到 Compose 文件中。您的文件应类似于以下示例。将:版本*`your-container-name### Dock`* er 镜像的名称。

```
version: '3.7'

services:
  greengrass:
```

```
    init: true
    build:
      context: .
    container_name: aws-iot-greengrass
    image: your-container-name:version
    volumes:
      - /path/to/greengrass-v2-config:/tmp/config/:ro
      - /path/to/greengrass-v2-certs:/tmp/certs:ro
    env_file: .env
    ports:
      - "8883:8883"
```

此示例 Compose 文件中的以下参数是可选的：

- `ports`— 将 8883 个容器端口发布到主机。如果您想通过 MQTT 进行连接和通信，则需要使用此参数，因为 MQTT 流量AWS IoT Greengrass使用端口 8883。
- `env_file`— 指定环境文件以设置将传递给 Docker 容器内的 C AWS IoT Greengrass ore 软件安装程序的环境变量。只有在创建环境文件来设置环境变量时，才需要此参数。如果您没有创建环境文件，则可以使用环境参数直接在 Compose 文件中设置变量。

> **Note**
>
> 要以更高的安全性运行 Docker 容器，您可以在 Compose 文件`cap_add`中使用`cap_drop`和来选择性地为容器启用 Linux 功能。有关更多信息，请参阅 Docker 文档中的运行时权限和 Linux 功能。

3. 运行以下命令启动容器。

```
docker-compose -f docker-compose.yml up
```

后续步骤

AWS IoT Greengrass核心软件现在在 Docker 容器中运行。运行以下命令以检索当前正在运行的容器的容器 ID。

```
docker ps
```

然后，您可以运行以下命令来访问容器并浏览容器内运行的 AWS IoT Greengrass Core 软件。

```
docker exec -it container-id /bin/bash
```

有关创建简单组件的信息，请参见第 4 步：在设备上开发和测试组件中的 教程：AWS IoT Greengrass V2 入门

> **Note**
>
> 当你使用`docker exec`在 Docker 容器内运行命令时，这些命令不会记录在 Docker 日志中。要将您的命令记录在 Docker 日志中，请将交互式外壳附加到 Docker 容器。有关更多信息，请参阅 将交互式外壳附加到 Docker 容器。

C AWS IoT Greengrass ore 日志文件被调用`greengrass.log`，位于中*/greengrass/v2*`/logs`。组件日志文件也位于同一目录中。要将 Greengrass 日志复制到主机上的临时目录，请运行以下命令：

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

如果您想在容器退出或移除后保留日志，我们建议您仅将该目录绑定到主机上的临时日志*/greengrass/v2*`/logs`目录，而不是挂载整个 Greengrass 目录。有关更多信息，请参阅 在 Docker 容器之外保存 Greengrass 日志。

要停止正在运行的 AWS IoT Greengrass Docker 容器，请运行`docker stop`或`docker-compose -f docker-compose.yml stop`。此操作发送`SIGTERM`到 Greengrass 进程，并关闭容器中启动的所有关联进程。Docker 容器使用`docker-init`可执行文件作为进程 PID 1 进行初始化，这有助于删除所有剩余的僵尸进程。有关更多信息，请参阅 Docker 文档中的指定初始化进程。

有关在 Docker 容器AWS IoT Greengrass中运行时遇到的问题疑难解答的信息，请参阅对 Docker 容器中的 AWS IoT Greengrass 执行问题排查。

对 Docker 容器中的 AWS IoT Greengrass 执行问题排查

使用以下信息来帮助您解决在 Docker 容器AWS IoT Greengrass中运行的问题，并调试 Docker 容器AWS IoT Greengrass中的问题。

主题

- 对运行 Docker 容器时出现的问题进行故障排除
- 在 Docker 容器中调试 AWS IoT Greengrass

对运行 Docker 容器时出现的问题进行故障排除

使用以下信息可帮助解决与在 Docker 容器中运行 AWS IoT Greengrass 相关的问题。

主题

- 错误：无法从非 TTY 设备执行交互式登录
- 错误：未知选项：-no-include-email
- 错误：防火墙阻止 Windows 和容器之间的文件共享。
- 错误：调用 GetAuthorizationToken 操作时出现错误 (AccessDeniedException)：用户：arn: aws: iam:: ac count-id: user/ <user-name>无权在资源上执行：ecr：* GetAuthorizationToken
- 错误：您已达到拉取速率上限

错误：无法从非 TTY 设备执行交互式登录

当您运行`aws ecr get-login-password`命令时，可能会出现此错误。确保您已安装最新的 AWS CLI 版本 2 或版本 1。建议您使用 AWS CLI 版本 2。有关更多信息，请参阅AWS Command Line Interface《用户指南》中的安装 AWS CLI。

错误：未知选项：-no-include-email

当您运行`aws ecr get-login`命令时，可能会出现此错误。确保您已安装最新的 AWS CLI 版本（例如，运行：`pip install awscli --upgrade --user`）。有关更多信息，请参阅AWS Command Line Interface用户指南中的在 Microsoft Windows 上安装 AWS Command Line Interface。

错误：防火墙阻止 Windows 和容器之间的文件共享。

在 Windows 计算机上运行 Docker 时，你可能会收到此错误或一条`Firewall Detected`消息。如果您登录虚拟私有网络 (VPN) 并且网络设置阻止挂载共享驱动器，也会出现此错误。在这种情况下，请关闭 VPN 并重新运行 Docker 容器。

错误：调用 GetAuthorizationToken 操作时出现错误 (AccessDeniedException)：用户：arn: aws: iam:: ac ***`count-id: user/`*** <user-name>无权在资源上执行：ecr：* GetAuthorizationToken

如果您没有足够权限来访问 Amazon ECR 存储库，则运行 `aws ecr get-login-password` 命令时可能会收到此错误。有关更多信息，请参阅Amazon ECR 用户指南中的 Amazon ECR 存储库策略示例和访问 One Amazon ECR 存储库。

错误：您已达到拉取速率上限

Docker Hub 限制了匿名用户和 Free Docker Hub 用户可以发出的拉取请求的数量。如果您超过匿名或免费用户拉取请求的速率限制，则会收到以下错误之一：

```
ERROR: toomanyrequests: Too Many Requests.
```

```
You have reached your pull rate limit.
```

要解决这些错误，您可以等待几个小时再尝试另一个拉取请求。如果您计划持续提交大量拉取请求，请访问 Docker Hub 网站，了解有关速率限制以及身份验证和升级 Docker 帐户的选项的信息。

在 Docker 容器中调试 AWS IoT Greengrass

要调试 Docker 容器的问题，您可以保留 Greengrass 运行时日志或将交互式 shell 附加到 Docker 容器。

在 Docker 容器之外保存 Greengrass 日志

停止AWS IoT Greengrass容器后，您可以使用以下`docker cp` 命令将 Greengrass 日志从 Docker 容器复制到临时日志目录。

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

要在容器退出或移除后仍保留日志，您必须在绑定挂载目录后运行 AWS IoT Greengrass Docker 容器。*/greengrass/v2*`/logs`

要绑定挂载*/greengrass/v2*`/logs`目录，请在运行新的 D AWS IoT Greengrass ocker 容器时执行以下任一操作。

- 包含`-v` */tmp/logs*:*/greengrass/v2*`/logs:ro`在你的`docker run`命令中。

  在运行`docker-compose up`命令之前，修改 Compose 文件中的`volumes`块以包含以下行。

```
volumes:
 - /tmp/logs:/greengrass/v2/logs:ro
```

然后，当在 Docker 容器内运行时，你可以在主机*/tmp/logs*上查看日志，查看 Greengrass 日志 AWS IoT Greengrass。

有关运行 Greengrass Docker 容器的信息，请参阅和 使用手动AWS IoT Greengrass配置在 Docker 中运行 通过自动AWS IoT Greengrass配置在 Docker 中运行

将交互式外壳附加到 Docker 容器

当你使用`docker exec`在 Docker 容器内运行命令时，这些命令不会在 Docker 日志中捕获。在 Docker 日志中记录命令可以帮助你调查 Greengrass Docker 容器的状态。请执行以下操作之一：

- 在单独的终端中运行以下命令，将终端的标准输入、输出和错误附加到正在运行的容器。这使您能够从当前终端查看和控制 Docker 容器。

```
docker attach container-id
```

- 在单独的终端中运行以下命令。这使您能够在交互模式下运行命令，即使未连接容器也是如此。

```
docker exec -it container-id sh -c "command > /proc/1/fd/1"
```

有关一般AWS IoT Greengrass故障排除，请参阅故障排除。

配置 AWS IoT Greengrass 核心软件

C AWS IoT Greengrass ore 软件提供了可用于配置软件的选项。您可以创建部署以在每台 AWS IoT Greengrass 核心设备上配置核心软件。

主题

- 部署 Greengrass nucleus 组件
- 将 Greengrass 核心配置为系统服务
- 使用 JVM 选项控制内存分配
- 配置运行组件的用户
- 为组件配置系统资源限制
- 通过端口 443 或网络代理进行连接
- 使用由私有 CA 签名的设备证书
- 配置 MQTT 超时和缓存设置

部署 Greengrass nucleus 组件

AWS IoT Greengrass 将 C AWS IoT Greengrass ore 软件作为组件提供，您可以将其部署到 Greengrass 核心设备上。您可以创建一个部署，将相同的配置应用于多个 Greengrass 核心设备。有关更多信息，请参阅 Greengrass 核 和 更新AWS IoT Greengrass核心软件 (OTA)。

将 Greengrass 核心配置为系统服务

要执行以下操作，您必须在设备的初始化系统中将 C AWS IoT Greengrass ore 软件配置为系统服务：

- 设备启动时启动 AWS IoT Greengrass Core 软件。如果您管理大量设备，这是一种很好的做法。
- 安装并运行插件组件。 AWS提供的几个组件是插件组件，这使它们能够直接与 Greengrass 核接口。有关组件类型的更多信息，请参阅 组件类型。
- 对核心设备的 AWS IoT Greengrass 核心软件应用 over-the-air (OTA) 更新。有关更多信息，请参阅 更新AWS IoT Greengrass核心软件 (OTA)。
- 允许组件在部署将组件更新到新版本或更新某些配置参数时重新启动 AWS IoT Greengrass 核心软件或核心设备。有关更多信息，请参阅引导生命周期步骤。

在 Windows 核心设备上，必须将 AWS IoT Greengrass 核心软件设置为系统服务。

主题

- 将核心配置为系统服务 (Linux)
- 将 nucleus 配置为系统服务 (Windows)

将核心配置为系统服务 (Linux)

Linux 设备支持不同的初始化系统，例如 initd、systemd 和 systemV。在安装 AWS IoT Greengrass Core 软件时，您可以使用`--setup-system-service true`参数将 nucleus 作为系统服务启动，并将其配置为在设备启动时启动。安装程序使用 systemd 将 AWS IoT Greengrass 核心软件配置为系统服务。

您也可以手动将 nucleus 配置为作为系统服务运行。以下示例是用于 systemd 的服务文件。

```
[Unit]
```

```
Description=Greengrass Core

[Service]
Type=simple
PIDFile=/greengrass/v2/alts/loader.pid
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

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

配置系统服务后，您可以运行以下命令来配置启动设备以及启动或停止 AWS IoT Greengrass Core 软件。

- 检查服务的状态 (systemd)

```
sudo systemctl status greengrass.service
```

- 使原子核能够在设备启动时启动。

```
sudo systemctl enable greengrass.service
```

- 在设备启动时阻止原子核启动。

```
sudo systemctl disable greengrass.service
```

- 启动 AWS IoT Greengrass 核心软件。

```
sudo systemctl start greengrass.service
```

- 停止 C AWS IoT Greengrass ore 软件。

```
sudo systemctl stop greengrass.service
```

将 nucleus 配置为系统服务 (Windows)

在安装 AWS IoT Greengrass Core 软件时，您可以使用`--setup-system-service true`参数将 nucleus 作为 Windows 服务启动，并将其配置为在设备启动时启动。

配置服务后，您可以运行以下命令来配置启动设备以及启动或停止 AWS IoT Greengrass Core 软件。必须运行命令提示符或以管理员 PowerShell 身份运行这些命令。

Windows Command Prompt (CMD)

- 检查服务状态

```
sc query "greengrass"
```

- 使原子核能够在设备启动时启动。

```
sc config "greengrass" start=auto
```

- 在设备启动时阻止原子核启动。

```
sc config "greengrass" start=disabled
```

- 启动 AWS IoT Greengrass 核心软件。

```
sc start "greengrass"
```

- 停止 C AWS IoT Greengrass ore 软件。

```
sc stop "greengrass"
```

> **Note**
> 在 Windows 设备上， AWS IoT Greengrass 酷睿软件在关闭 Greengrass 组件进程时会忽略此关闭信号。如果您运行此命令时， AWS IoT Greengrass Core 软件忽略了关机信号，请等待几秒钟，然后重试。

PowerShell

- 检查服务状态

```
Get-Service -Name "greengrass"
```

- 使原子核能够在设备启动时启动。

```
Set-Service -Name "greengrass" -Status stopped -StartupType automatic
```

- 在设备启动时阻止原子核启动。

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

- 启动 AWS IoT Greengrass 核心软件。

```
Start-Service -Name "greengrass"
```

- 停止 C AWS IoT Greengrass ore 软件。

```
Stop-Service -Name "greengrass"
```

> **Note**
>
> 在 Windows 设备上， AWS IoT Greengrass 酷睿软件在关闭 Greengrass 组件进程时会忽略此关闭信号。如果您运行此命令时， AWS IoT Greengrass Core 软件忽略了关机信号，请等待几秒钟，然后重试。

使用 JVM 选项控制内存分配

如果您在内存有限的设备 AWS IoT Greengrass 上运行，则可以使用 Java 虚拟机 (JVM) 选项来控制最大堆大小、垃圾收集模式和编译器选项，这些选项控制 AWS IoT Greengrass 核心软件使用的内存量。JVM 中的堆大小决定了在垃圾收集发生之前或应用程序耗尽内存之前，应用程序可以使用的内存量。最大堆大小指定了在占用大量内存的活动期间扩展堆时，JVM 可分配的最大内存量。

要控制内存分配，请创建新部署或修改包含 nucleus 组件的现有部署，然后在 nuc leus 组件配置的`jvmOptions`配置参数中指定您的 JVM 选项。

根据您的要求，您可以在减少内存分配或最小内存分配的情况下运行 C AWS IoT Greengrass ore 软件。

减少内存分配

要在减少内存分配的情况下运行 AWS IoT Greengrass Core 软件，我们建议您使用以下示例配置合并更新在 nucleus 配置中设置 JVM 选项：

```
{
  "jvmOptions": "-Xmx64m -XX:+UseSerialGC -XX:TieredStopAtLevel=1"
}
```

最小内存分配

要以最少的内存分配运行 AWS IoT Greengrass Core 软件，我们建议您使用以下示例配置合并更新在 nucleus 配置中设置 JVM 选项：

```
{
  "jvmOptions": "-Xmx32m -XX:+UseSerialGC -Xint"
}
```

这些示例配置合并更新使用以下 JVM 选项：

`-Xmx`*NN*`m`

设置最大 JVM 堆大小。

要减少内存分配，请使用`-Xmx64m`作为起始值，将堆大小限制为 64 MB。要获得最小内存分配，请使用`-Xmx32m`作为起始值，将堆大小限制为 32 MB。

您可以根据实际需求增加或减少该`-Xmx`值；但是，我们强烈建议您不要将最大堆大小设置为 16 MB 以下。如果最大堆大小对于您的环境来说太低，则 AWS IoT Greengrass Core 软件可能会因为内存不足而遇到意外错误。

`-XX:+UseSerialGC`

指定对 JVM 堆空间使用串行垃圾收集。与其他 JVM 垃圾收集实现相比，串行垃圾收集器速度较慢，但占用的内存更少。

`-XX:TieredStopAtLevel=1`

指示 JVM 使用一次 Java just-in-time (JIT) 编译器。由于 JIT 编译后的代码占用设备内存中的空间，因此多次使用 JIT 编译器比单次编译消耗更多的内存。

`-Xint`

指示 JVM 不要使用 just-in-time (JIT) 编译器。相反，JVM 在仅解释模式下运行。此模式比运行 JIT 编译后的代码慢；但是，编译后的代码不占用任何内存空间。

有关创建配置合并更新的信息，请参阅更新组件配置。

配置运行组件的用户

AWS IoT Greengrass Core 软件可以以不同于运行该软件的系统用户和组的身份运行组件进程。这可以提高安全性，因为您可以以 root 用户或管理员用户身份运行 C AWS IoT Greengrass ore 软件，而无需将这些权限授予在核心设备上运行的组件。

下表显示了 AWS IoT Greengrass 核心软件可以以您指定的用户身份运行哪些类型的组件。有关更多信息，请参阅 组件类型。

组件类型	配置组件用户
核	否
插件	否
通用	是
Lambda（非容器化）	是
Lambda（集装箱化）	是

必须先创建组件用户，然后才能在部署配置中指定该用户。在基于 Windows 的设备上，您还必须将用户的用户名和密码存储在账户的凭据管理器实例中。 LocalSystem 有关更多信息，请参阅 在 Windows 设备上设置组件用户。

在基于 Linux 的设备上配置组件用户时，也可以选择指定组。您可以按以下格式指定由冒号 (`:`) 分隔的用户和组：*user*:*group*。如果您未指定群组，则 AWS IoT Greengrass Core 软件将默认为该用户的主群组。您可以使用名称或 ID 来识别用户和组。

在基于 Linux 的设备上，您还可以以不存在的系统用户（也称为未知用户）的身份运行组件，以提高安全性。Linux 进程可以向同一用户运行的任何其他进程发出信号。未知用户不会运行其他进程，因此您可以以未知用户身份运行组件，以防止组件向核心设备上的其他组件发出信号。要以未知用户身份运行组件，请指定核心设备上不存在的用户 ID。您也可以指定不存在的群组 ID 以未知群组的身份运行。

您可以为每个组件和每个核心设备配置用户。

- 为组件进行配置

  您可以将每个组件配置为使用该组件特定的用户运行。创建部署时，可以在该组件的`runWith`配置中为每个组件指定用户。如果您配置组件， AWS IoT Greengrass Core 软件将以指定用户身份运行组件。否则，它将默认以您为核心设备配置的默认用户身份运行组件。有关在部署配置中指定组件用户的更多信息，请参阅中的`runWith`配置参数创建部署。
- 为核心设备配置默认用户

  您可以配置一个默认用户， AWS IoT Greengrass Core 软件使用该用户来运行组件。当 AWS IoT Greengrass Core 软件运行某个组件时，它会检查您是否为该组件指定了用户，并使用它来运行该组件。如果组件未指定用户，则 AWS IoT Greengrass Core 软件将以您为核心设备配置的默认用户身份运行该组件。有关更多信息，请参阅 配置默认组件用户。

> **Note**
>
> 在基于 Windows 的设备上，必须至少指定一个默认用户才能运行组件。
>
> 在基于 Linux 的设备上，如果您未将用户配置为运行组件，则需要考虑以下注意事项：
>
> - 如果您以 root 用户身份运行 AWS IoT Greengrass Core 软件，则该软件将无法运行组件。如果您以 root 身份运行，则必须指定默认用户来运行组件。
> - 如果您以非 root 用户身份运行 AWS IoT Greengrass Core 软件，则该软件将以该用户身份运行组件。

主题

- 在 Windows 设备上设置组件用户
- 配置默认组件用户

在 Windows 设备上设置组件用户

在基于 Windows 的设备上设置组件用户

1. 在设备上的 LocalSystem 帐户中创建组件用户。

```
net user /add component-user password
```

2. 使用 Microsoft 的 PsExec 实用程序将组件用户的用户名和密码存储在 LocalSystem 帐户的凭据管理器实例中。

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

> **Note**
>
> 在基于 Windows 的设备上，该 LocalSystem 帐户运行 Greengrass 核，您必须使用该 PsExec 实用程序将组件用户信息存储在帐户中。 LocalSystem使用凭据管理器应用程序将此信息存储在当前登录用户的 Windows 帐户中，而不是 LocalSystem帐户中。

配置默认组件用户

您可以使用部署在核心设备上配置默认用户。在此部署中，您将更新 n ucleus 组件配置。

> **Note**
>
> 在安装 C AWS IoT Greengrass ore 软件时，也可以使用`--component-default-user`选项设置默认用户。有关更多信息，请参阅 安装 AWS IoT Greengrass Core 软件。

创建一个部署，为该`aws.greengrass.Nucleus`组件指定以下配置更新。

Linux

```
{
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Windows

```
{
  "runWithDefault": {
    "windowsUser": "ggc_user"
  }
}
```

> **Note**
>
> 您指定的用户必须存在，并且该用户的用户名和密码必须存储在您的 Windows 设备上该 LocalSystem 帐户的凭据管理器实例中。有关更多信息，请参阅 在 Windows 设备上设置组件用户。

以下示例定义了配置`ggc_user`为默认用户和`ggc_group`默认组的基于 Linux 的设备的部署。`merge`配置更新需要序列化的 JSON 对象。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"posixUser\":\"ggc_user:ggc_group\"}}"
      }
    }
  }
}
```

为组件配置系统资源限制

Note

此功能适用于 Greengrass nucleus 组件的 v2.4.0 及更高版本。 AWS IoT Greengrass 目前不支持在 Windows 核心设备上使用此功能。

您可以配置每个组件的进程可以在核心设备上使用的最大 CPU 和 RAM 使用量。

下表显示了支持系统资源限制的组件类型。有关更多信息，请参阅 组件类型。

组件类型	配置系统资源限制
核	否
插件	否
通用	是
Lambda（非容器化）	是

组件类型	配置系统资源限制
Lambda（集装箱化）	否

⚠ Important

在 Docker 容器中运行 C AWS IoT Greengrass ore 软件时，不支持系统资源限制。

您可以为每个组件和每个核心设备配置系统资源限制。

- 为组件进行配置

  您可以为每个组件配置特定于该组件的系统资源限制。创建部署时，可以为部署中的每个组件指定系统资源限制。如果组件支持系统资源限制，则 AWS IoT Greengrass Core 软件会将限制应用于该组件的进程。如果您没有为组件指定系统资源限制，则 AWS IoT Greengrass Core 软件将使用您为核心设备配置的任何默认值。有关更多信息，请参阅 创建部署。
- 为核心设备配置默认值

  您可以配置 C AWS IoT Greengrass ore 软件应用于支持这些限制的组件的默认系统资源限制。当 AWS IoT Greengrass 核心软件运行某个组件时，它会应用您为该组件指定的系统资源限制。如果该组件未指定系统资源限制，则 AWS IoT Greengrass Core 软件将应用您为核心设备配置的默认系统资源限制。如果您未指定默认的系统资源限制，则默认情况下， AWS IoT Greengrass Core 软件不会应用任何系统资源限制。有关更多信息，请参阅 配置默认的系统资源限制。

配置默认的系统资源限制

您可以部署 Greengrass nucleus 组件来配置核心设备的默认系统资源限制。要配置默认的系统资源限制，请创建一个为`aws.greengrass.Nucleus`组件指定以下配置更新的部署。

```
{
  "runWithDefault": {
    "systemResourceLimits": {
      "cpu": cpuTimeLimit,
```

```
        "memory": memoryLimitInKb
      }
    }
}
```

以下示例定义了一个部署，该部署将 CPU 时间限制配置为2，相当于具有 4 个 CPU 内核的设备上的 50% 使用率。此示例还将内存使用量配置为 100 MB。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"systemResourceLimits\":\"cpus\":2,\"memory
\":102400}}}"
      }
    }
  }
}
```

通过端口 443 或网络代理进行连接

AWS IoT Greengrass 核心设备 AWS IoT Core 使用带有 TLS 客户端身份验证的 MQTT 消息协议与之通信。按照惯例，基于 TLS 的 MQTT 使用端口 8883。但是，作为一项安全措施，限制性环境可能会将入站和出站流量限制到一个较小的 TCP 端口范围。例如，企业防火墙可能会为 HTTPS 流量打开端口 443，但关闭不常用协议使用的其他端口，例如用于 MQTT 流量的端口 8883。其他限制性环境可能要求所有流量在连接到互联网之前都要通过代理。

> **Note**
>
> 运行 Greengrass 核心组件 v2.0.3 及更早版本的 Greengrass 核心设备使用端口 8443 连接到数据平面端点。 AWS IoT Greengrass 这些设备必须能够通过端口 8443 连接到此端点。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

为了在这些情况下启用通信， AWS IoT Greengrass 提供了以下配置选项：

- 通过端口 443 进行 MQTT 通信。如果您的网络允许连接到端口 443，则可以将 Greengrass 核心设备配置为使用端口 443 来处理 MQTT 流量，而不是默认端口 8883。这可以是与端口 443 的直接连

接，也可以是通过网络代理服务器的连接。与使用基于证书的客户端身份验证的默认配置不同，端口 443 上的 MQTT 使用设备服务角色进行身份验证。

有关更多信息，请参阅 通过端口 443 配置 MQTT。

- 通过端口 443 进行 HTTPS 通信。默认情况下， AWS IoT Greengrass 核心软件通过端口 8443 发送 HTTPS 流量，但您可以将其配置为使用端口 443。 AWS IoT Greengrass 使用应用层协议网络 (ALPN) TLS 扩展来启用此连接。与默认配置一样，端口 443 上的 HTTPS 使用基于证书的客户端身份验证。

> **Important**
> 要使用 ALPN 并通过端口 443 启用 HTTPS 通信，您的核心设备必须运行 Java 8 更新 252 或更高版本。Java 版本 9 及更高版本的所有更新也支持 ALPN。

有关更多信息，请参阅 通过端口 443 配置 HTTPS。

- 通过网络代理连接。您可以将网络代理服务器配置为连接到 Greengrass 核心设备的中介。 AWS IoT Greengrass 支持 HTTP 和 HTTPS 代理的基本身份验证。

Greengrass 核心设备必须运行 Greengrass nucleus v2.5.0 或更高版本才能使用 HTTPS 代理。

C AWS IoT Greengrass ore 软件通过`ALL_PROXY`、`HTTP_PROXYHTTPS_PROXY`、和`NO_PROXY`环境变量将代理配置传递给组件。组件必须使用这些设置才能通过代理进行连接。组件使用常用库（例如 boto3、curl 和 python `requests` 包），默认情况下，这些库通常使用这些环境变量来建立连接。如果组件还指定了这些环境变量，则 AWS IoT Greengrass 不会覆盖它们。

有关更多信息，请参阅 配置网络代理。

通过端口 443 配置 MQTT

您可以在现有核心设备上通过端口 443 配置 MQTT，也可以在新的核心设备上安装 AWS IoT Greengrass Core 软件时配置 MQTT。

主题

- 在现有核心设备上通过端口 443 配置 MQTT
- 在安装过程中通过端口 443 配置 MQTT

在现有核心设备上通过端口 443 配置 MQTT

您可以使用部署在单核设备或一组核心设备上通过端口 443 配置 MQTT。在此部署中，您将更新 nucleus 组件配置。当您更新其mqtt配置时，原子核会重新启动。

要通过端口 443 配置 MQTT，请创建一个为组件指定以下配置更新的部署。aws.greengrass.Nucleus

```
{
  "mqtt": {
    "port": 443
  }
}
```

以下示例定义了通过端口 443 配置 MQTT 的部署。merge配置更新需要序列化的 JSON 对象。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"mqtt\":{\"port\":443}}"
      }
    }
  }
}
```

在安装过程中通过端口 443 配置 MQTT

在核心设备上安装 AWS IoT Greengrass 核心软件时，可以通过端口 443 配置 MQTT。使用--init-config安装程序参数通过端口 443 配置 MQTT。在使用手动配置、队列配置或自定义配置进行安装时，您可以指定此参数。

通过端口 443 配置 HTTPS

此功能需要 Greengrass 核 v2.0.4 或更高版本。

您可以在现有核心设备上或在新的核心设备上安装 AWS IoT Greengrass 核心软件时通过端口 443 配置 HTTPS。

主题

- 在现有核心设备上通过端口 443 配置 HTTPS
- 在安装过程中通过端口 443 配置 HTTPS

在现有核心设备上通过端口 443 配置 HTTPS

您可以使用部署在单核设备或一组核心设备上通过端口 443 配置 HTTPS。在此部署中，您将更新 nucleus 组件配置。

要通过端口 443 配置 HTTPS，请创建一个为`aws.greengrass.Nucleus`组件指定以下配置更新的部署。

```
{
  "greengrassDataPlanePort": 443
}
```

以下示例定义了通过端口 443 配置 HTTPS 的部署。`merge`配置更新需要序列化的 JSON 对象。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"greengrassDataPlanePort\":443}"
      }
    }
  }
}
```

在安装过程中通过端口 443 配置 HTTPS

在核心设备上安装 AWS IoT Greengrass 核心软件时，可以通过端口 443 配置 HTTPS。使用`--init-config`安装程序参数通过端口 443 配置 HTTPS。在使用手动配置、队列配置或自定义配置进行安装时，您可以指定此参数。

配置网络代理

按照本节中的步骤将 Greengrass 核心设备配置为通过 HTTP 或 HTTPS 网络代理连接到互联网。有关核心设备使用的端点和端口的更多信息，请参阅允许设备流量通过代理或防火墙。

⚠ Important

如果您的核心设备运行的版本低于 v2.4.0 的 Greengrass nucleus，则您的设备角色必须允许以下权限才能使用网络代理：

- `iot:Connect`
- `iot:Publish`
- `iot:Receive`
- `iot:Subscribe`

这是必要的，因为设备使用来自令牌交换服务的 AWS 凭据来验证与 MQTT 的连接。 AWS IoT 设备使用 MQTT 接收和安装来自的部署 AWS Cloud，因此，除非您对其角色定义这些权限，否则您的设备将无法运行。设备通常使用 X.509 证书对 MQTT 连接进行身份验证，但是设备在使用代理时无法使用此证书进行身份验证。

有关如何配置设备角色的更多信息，请参阅授权核心设备与AWS服务。

主题

- 在现有核心设备上配置网络代理
- 在安装过程中配置网络代理
- 使核心设备能够信任 HTTPS 代理
- 网络代理对象

在现有核心设备上配置网络代理

您可以使用部署在单核设备或一组核心设备上配置网络代理。在此部署中，您将更新 n ucleus 组件配置。当您更新其`networkProxy`配置时，原子核会重新启动。

要配置网络代理，请为合并以下配置更新的`aws.greengrass.Nucleus`组件创建部署。此配置更新包含 NetworkProxy 对象。

```
{
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "https://my-proxy-server:1100"
    }
```

```
  }
}
```

以下示例定义了配置网络代理的部署。merge配置更新需要序列化的 JSON 对象。

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"networkProxy\":{\"noProxyAddresses\":
\"http://192.168.0.1,www.example.com\",\"proxy\":{\"url\":\"https://my-proxy-
server:1100\",\"username\":\"Mary_Major\",\"password\":\"pass@word1357\"}}}"
      }
    }
  }
}
```

在安装过程中配置网络代理

在 AWS IoT Greengrass 核心设备上安装 Core 软件时，可以配置网络代理。使用--init-config安装程序参数配置网络代理。在使用手动配置、队列配置或自定义配置进行安装时，您可以指定此参数。

使核心设备能够信任 HTTPS 代理

将核心设备配置为使用 HTTPS 代理时，必须将代理服务器证书链添加到核心设备中，使其能够信任 HTTPS 代理。否则，核心设备在尝试通过代理路由流量时可能会遇到错误。将代理服务器 CA 证书添加到核心设备的 Amazon 根 CA 证书文件中。

使核心设备能够信任 HTTPS 代理

1. 在核心设备上找到 Amazon 根 CA 证书文件。

   - 如果您安装了具有自动配置功能的 C AWS IoT Greengrass ore 软件，则 Amazon 根 CA 证书文件位于中*/greengrass/v2*/rootCA.pem。
   - 如果您使用手动或队列配置安装 AWS IoT Greengrass 核心软件，则 Amazon 根 CA 证书文件可能存在于*/greengrass/v2*/AmazonRootCA1.pem。

   如果这些地点不存在 Amazon 根 CA 证书，请查看system.rootCaPath房产*/greengrass/v2*/config/effectiveConfig.yaml以查找其位置。

2. 将代理服务器 CA 证书文件的内容添加到 Amazon 根 CA 证书文件中。

   以下示例显示了添加到 Amazon 根 CA 证书文件中的代理服务器 CA 证书。

```
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

网络代理对象

使用 `networkProxy` 对象指定有关网络代理的信息。该对象包含以下信息：

`noProxyAddresses`

（可选）以逗号分隔的 IP 地址或主机名列表，这些地址或主机名不受代理限制。

`proxy`

要连接的代理。该对象包含以下信息：

`url`

格式为代理服务器的 URL `scheme://userinfo@host:port`。

- `scheme`— 方案，必须是`http`或`https`。

Important

Greengrass 核心设备必须运行 Greengrass nucleus v2.5.0 或更高版本才能使用 HTTPS 代理。

> 如果您配置 HTTPS 代理，则必须将代理服务器 CA 证书添加到核心设备的 Amazon 根 CA 证书中。有关更多信息，请参阅 使核心设备能够信任 HTTPS 代理。

- `userinfo`—（可选）用户名和密码信息。如果您在中指定此信息`url`，Greengrass 核心设备将忽略和字段。`username` `password`
- `host`— 代理服务器的主机名或 IP 地址。
- `port`—（可选）端口号。如果您未指定端口，则 Greengrass 核心设备将使用以下默认值：
  - `http`— 80
  - `https`— 443

`username`

（可选）对代理服务器进行身份验证的用户名。

`password`

（可选）用于验证代理服务器的密码。

使用由私有 CA 签名的设备证书

如果您使用的是自定义私有证书颁发机构 (CA)，则必须将 Greengrass 核心设置为。**`greengrassDataPlaneEndpoint`** **`iotdata`**可以在部署或安装期间使用安装**`--init-config`**程序参数设置此选项。

您可以自定义设备连接的 Greengrass 数据平面端点。您可以将此配置选项设置为，将 Greengrass 数据平面终端节点设置为与物联网数据端点相同的端点，您可以使用指定该端点。**`iotdata`** **`iotDataEndpoint`**

配置 MQTT 超时和缓存设置

在 AWS IoT Greengrass 环境中，组件可以使用 MQTT 进行 AWS IoT Core通信。 AWS IoT Greengrass 核心软件管理组件的 MQTT 消息。当核心设备失去与的连接时 AWS Cloud，软件会缓存 MQTT 消息，以便稍后在连接恢复后重试。您可以配置诸如消息超时和缓存大小之类的设置。有关更多信息，请参阅 Greengrass nucleus 组件的`mqtt`和`mqtt.spooler`配置参数。

AWS IoT Core 对其 MQTT 消息代理施加服务配额。这些配额可能适用于您在核心设备和之间发送的消息 AWS IoT Core。有关更多信息，请参阅中的AWS IoT Core 消息代理服务配额AWS 一般参考。

更新AWS IoT Greengrass核心软件 (OTA)

AWS IoT Greengrass核心软件包括 Greengrass nucleus 组件和其他可选组件，您可以将这些组件部署到设备上以 over-the-air 执行软件的 (OTA) 更新。此功能内置在 AWS IoT Greengrass Core 软件中。

OTA 更新可以更高效地执行以下操作：

- 修复安全漏洞。
- 解决软件稳定性问题。
- 部署新的或改进的功能。

主题

- 要求
- 核心设备的注意事项
- Greengrass 核更新行为
- 执行 OTA 更新

要求

以下要求适用于部署AWS IoT Greengrass核心软件的 OTA 更新：

- Greengrass 核心设备必须连接到才能AWS Cloud接收部署。
- 必须正确配置 Greengrass 核心设备并配置证书和密钥，以便使用和进行身份验证。AWS IoT Core AWS IoT Greengrass
- C AWS IoT Greengrass ore 软件必须作为系统服务进行设置和运行。如果您从 JAR 文件运行核心，OTA 更新将不起作用。`Greengrass.jar`有关更多信息，请参阅 将 Greengrass 核心配置为系统服务。

核心设备的注意事项

在执行 OTA 更新之前，请注意对您更新的核心设备及其连接的客户端设备的影响：

- Greengrass 原子核关闭了。

- 核心设备上运行的所有组件也将关闭。如果这些组件写入本地资源，则除非正确关闭，否则它们可能会使这些资源处于不正确的状态。组件可以使用进程间通信来告诉 nucleus 组件推迟更新，直到它们清理所使用的资源。
- 当 nucleus 组件关闭时，核心设备会失去与AWS Cloud和本地设备的连接。核心设备在关闭时不会路由来自客户端设备的消息。
- 作为组件运行的长期 Lambda 函数会丢失其动态状态信息并丢弃所有待处理的工作。

Greengrass 核更新行为

部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。

当 Greengrass nucleus 组件的版本发生变化时，Core 软件（包括核心和设备AWS IoT Greengrass上的所有其他组件）将重新启动以应用更改。由于更新 nucleus 组件时会对核心设备产生影响，因此您可能需要控制何时将新 nucleus 补丁版本部署到您的设备。为此，您必须直接在部署中包含 Greengrass nucleus 组件。直接包含组件意味着您在部署配置中包含该组件的特定版本，并且不依赖组件依赖项将该组件部署到您的设备上。有关在组件配方中定义依赖关系的更多信息，请参阅食谱格式。

根据您的操作和部署配置，查看下表，了解 Greengrass nucleus 组件的更新行为。

操作	部署配置	Nucleus 更新行为
将新设备添加到现有部署所针对的事物组中，无需修改部署。	该部署不直接包括 Greengrass 核。 部署直接包括至少一个AWS由提供的组件，或者包括依赖于提供的组件或 Greengr AWS ass 核心的自定义组件。	在新设备上，安装符合所有组件依赖要求的最新补丁版本的 nucleus。 在现有设备上，不更新已安装的 nucleus 版本。
将新设备添加到现有部署所针对的事物组中，无需修改部署。	该部署直接包括特定版本的 Greengrass 核。	在新设备上，安装指定的 nucleus 版本。 在现有设备上，不更新已安装的 nucleus 版本。

操作	部署配置	Nucleus 更新行为
创建新部署或修改现有部署。	该部署不直接包括 Greengrass 核。 部署直接包括至少一个AWS由提供的组件，或者包括依赖于提供的组件或 Greengr AWS ass 核心的自定义组件。	在所有目标设备上，安装符合所有组件依赖关系要求的 nucleus 的最新补丁版本，包括在您添加到目标事物组的任何新设备上。
创建新部署或修改现有部署。	该部署直接包括特定版本的 Greengrass 核。	在所有目标设备上，安装指定的 nucleus 版本，包括您添加到目标事物组的所有新设备。

执行 OTA 更新

要执行 OTA 更新，请创建一个包含 n ucleus 组件和要安装的版本的部署。

卸载AWS IoT Greengrass核心软件

您可以卸载 AWS IoT Greengrass Core 软件，将其从不想用作 Greengrass 核心设备的设备中删除。您也可以使用这些步骤来清理失败的安装。

卸载 C AWS IoT Greengrass ore 软件

1. 如果将软件作为系统服务运行，则必须停止、禁用和删除该服务。根据您的操作系统运行以下命令。

   Linux

   1. 停止 服务。

      ```
      sudo systemctl stop greengrass.service
      ```

   2. 禁用该服务。

      ```
      sudo systemctl disable greengrass.service
      ```

3. 移除该服务。

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

4. 确认服务已删除。

```
sudo systemctl daemon-reload && sudo systemctl reset-failed
```

Windows (Command Prompt)

> **Note**
>
> 必须以管理员身份运行命令提示符才能运行这些命令。

1. 停止 服务。

```
sc stop "greengrass"
```

2. 禁用该服务。

```
sc config "greengrass" start=disabled
```

3. 移除该服务。

```
sc delete "greengrass"
```

4. 重启设备。

Windows (PowerShell)

> **Note**
>
> 您必须以管理员 PowerShell 身份运行才能运行这些命令。

1. 停止 服务。

```
Stop-Service -Name "greengrass"
```

2. 禁用该服务。

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

3. 移除该服务。

   • 对于 PowerShell 6.0 及更高版本：

   ```
   Remove-Service -Name "greengrass" -Confirm:$false -Verbose
   ```

   • 对于 6.0 之前的 PowerShell 版本：

   ```
   Get-Item HKLM:\SYSTEM\CurrentControlSet\Services\greengrass | Remove-Item
    -Force -Verbose
   ```

4. 重启设备。

2. 从设备中移除根文件夹。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为根文件夹的路径。

   Linux

   ```
   sudo rm -rf /greengrass/v2
   ```

   Windows (Command Prompt)

   ```
   rmdir /s /q C:\greengrass\v2
   ```

   Windows (PowerShell)

   ```
   cmd.exe /c "rmdir /s /q C:\greengrass\v2"
   ```

3. 从AWS IoT Greengrass服务中删除核心设备。此步骤将从中移除核心设备的状态信息AWS Cloud。如果您计划将AWS IoT Greengrass核心软件重新安装到同名的核心设备上，请务必完成此步骤。

   • 要从AWS IoT Greengrass控制台中删除核心设备，请执行以下操作：

     a. 导航到 AWS IoT Greengrass 控制台。

     b. 选择核心设备。

c. 选择要删除的核心设备。

d. 选择删除。

e. 在确认模式中，选择 “删除”。

- 要使用删除核心设备AWS Command Line Interface，请使用DeleteCoreDevice操作。运行以下命令，并*MyGreengrassCore*替换为核心设备的名称。

```
aws greengrassv2 delete-core-device --core-device-thing-name MyGreengrassCore
```

AWS IoT Greengrass V2 教程

您可以完成以下教程来了解AWS IoT Greengrass V2其功能。

主题

- 教程：开发一个可以延迟组件更新的 Greengrass 组件
- 教程：通过 MQTT 与本地物联网设备交互
- 教程： SageMaker 边缘管理器入门
- 教程：使用 TensorFlow Lite 执行样本图像分类推断
- 教程：使用 TensorFlow Lite 对来自相机的图像执行样本图像分类推断

教程：开发一个可以延迟组件更新的 Greengrass 组件

您可以完成本教程来开发延迟 over-the-air 部署更新的组件。在将更新部署到设备时，您可能需要根据条件延迟更新，例如：

- 该设备的电池电量不足。
- 设备正在运行无法中断的进程或作业。
- 该设备的互联网连接有限或昂贵。

> Note
>
> 组件是在AWS IoT Greengrass核心设备上运行的软件模块。组件使您能够将复杂的应用程序作为离散的构建块来创建和管理，您可以将这些应用程序从一个 Greengrass 核心设备重复使用到另一个 Greengrass 核心设备。

在本教程中，您将执行以下操作：

1. 在开发计算机上安装 Greengrass 开发套件 CLI (GDK CLI)。GDK CLI 提供的功能可帮助您开发自定义 Greengrass 组件。
2. 开发一个 Hello World 组件，用于在核心设备的电池电量低于阈值时推迟组件更新。此组件使用 SubscribeToComponentUpdatesIPC 操作订阅更新通知。当它收到通知时，它会检查电池电量是否低于可自定义的阈值。如果电池电量低于阈值，它会使用 DeferComponentUpdateIPC 操作将更新推迟 30 秒。您可以使用 GDK CLI 在开发计算机上开发此组件。

Note

该组件从您在核心设备上创建的文件中读取电池电量，以模仿真实的电池，因此您可以在没有电池的核心设备上完成本教程。

3. 将该组件发布到AWS IoT Greengrass服务。
4. 将该组件从部署AWS Cloud到 Greengrass 核心设备上进行测试。然后，您可以修改核心设备上的虚拟电池电量，并创建其他部署，以查看当电池电量不足时，核心设备如何推迟更新。

在本教程中，预计花费 20-30 分钟。

先决条件

要完成本教程，您需要：

- AWS 账户。如果没有，请参阅设置一个 AWS 账户。
- 具有管理员权限的 AWS Identity and Access Management (IAM) 用户。
- 一款具有互联网连接的 Greengrass 核心设备。有关如何设置核心设备的更多信息，请参阅设置AWS IoT Greengrass核心设备。
  - 在核心设备上为所有用户安装了 Python 3.6 或更高版本，并已添加到PATH环境变量中。在 Windows 上，你还必须为所有用户安装适用于 Windows 的 Python 启动器。

Important

在 Windows 中，默认情况下不会为所有用户安装 Python。安装 Python 时，必须对安装进行自定义，将其配置为AWS IoT Greengrass核心软件运行 Python 脚本。例如，如果您使用图形化 Python 安装程序，请执行以下操作：

1. 选择 “为所有用户安装启动器（推荐）”。
2. 选择Customize installation。
3. 选择Next。
4. 选择 Install for all users。
5. 选择 Add Python to environment variables。
6. 选择安装。

有关更多信息，请参阅 Python 3 文档中的在 Windows 上使用 Python。

- 一台具有互联网连接的 Windows、macOS 或 Unix 类似 Unix 的开发计算机。
  - 您的开发计算机上安装了 Python 3.6 或更高版本。
  - Git 已安装在您的开发计算机上。
  - AWS Command Line Interface(AWS CLI) 已在开发计算机上安装并使用凭据进行配置。有关更多信息，请参阅《AWS Command Line Interface用户指南》AWS CLI中的 "安装、更新AWS CLI和卸载" 和 "配置"。

Note

如果您使用树莓派或其他 32 位 ARM 设备，请安装 AWS CLI V1。 AWS CLIV2 不适用于 32 位 ARM 设备。有关更多信息，请参阅安装、更新和卸载AWS CLI版本 1。

第 1 步：安装 Greengrass 开发套件 CLI

Greengrass 开发套件 CLI (GDK CLI) 提供的功能可帮助您开发自定义 Greengras s 组件。您可以使用 GDK CLI 来创建、构建和发布自定义组件。

如果您尚未在开发计算机上安装 GDK CLI，请完成以下步骤进行安装。

安装最新版本的 GDK CLI

1. 在您的开发计算机上，运行以下命令从其GitHub存储库中安装最新版本的 GDK CLI。

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
```

2. 运行以下命令以验证 GDK CLI 是否成功安装。

```
gdk --help
```

如果找不到该gdk命令，请将其文件夹添加到 PATH。

- 在 Linux 设备上，添加/home/*MyUser*/.local/bin到 PATH，然后*MyUser*用您的用户名替换。

- 在 Windows 设备上，添加*PythonPath*\\Scripts到 PATH，然后*PythonPath*替换为设备上 Python 文件夹的路径。

第 2 步：开发可延迟更新的组件

在本节中，您将在 Python 中开发一个 Hello World 组件，当核心设备的电池电量低于您在部署组件时配置的阈值时，该组件会延迟组件更新。在此组件中，您将使用AWS IoT Device SDK适用于 Python 的 v2 中的进程间通信 (IPC) 接口。当核心设备收到部署时，您可以使用SubscribeToComponentUpdatesIPC 操作来接收通知。然后，您可以根据设备的电池电量使用DeferComponentUpdateIPC 操作来推迟或确认更新。

开发可延迟更新的 Hello World 组件

1. 在开发计算机上，为组件源代码创建一个文件夹。

```
mkdir com.example.BatteryAwareHelloWorld
cd com.example.BatteryAwareHelloWorld
```

2. 使用文本编辑器创建名为的文件`gdk-config.json`。GDK CLI 从名为 GDK CLI 的配置文件中读取数据`gdk-config.json`，以生成和发布组件。此配置文件存在于组件文件夹的根目录中。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano gdk-config.json
```

   将以下 JSON 复制到文件中。

   - 用你的名字替换 *Amazon*。
   - 将 *us-west-2* 替换为核心AWS 区域设备的运行位置。GDK CLI 将在此AWS 区域发布该组件。
   - *greengrass-component-artifacts*替换为要使用的 S3 存储桶前缀。当您使用 GDK CLI 发布组件时，GDK CLI 会使用以下格式将组件的项目上传到 S3 存储桶，该存储桶的名称由此值AWS 区域、和您的 AWS 账户 ID 组成：。*bucketPrefix*-*region*-*accountId*

     例如，如果您指定**greengrass-component-artifacts**和，且您的 AWS 账户 ID 为 **us-west-2123456789012**，则 GDK CLI 将使用名为`greengrass-component-artifacts-us-west-2-123456789012`的 S3 存储桶。

```
{
  "component": {
    "com.example.BatteryAwareHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
      "publish": {
        "region": "us-west-2",
        "bucket": "greengrass-component-artifacts"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

配置文件指定了以下内容：

- GDK CLI 将 Greengrass 组件发布到云服务时要使用的版本。AWS IoT Greengrass NEXT_PATCH指定在AWS IoT Greengrass云服务中可用的最新版本之后选择下一个补丁版本。如果该组件在AWS IoT Greengrass云服务中还没有版本，则 GDK CLI 会使用`1.0.0`。
- 组件的编译系统。当您使用编`zip`译系统时，GDK CLI 会将组件的源代码打包成一个 ZIP 文件，该文件将成为该组件的单个工件。
- GDK CLI AWS 区域 在那里发布 Greengrass 组件。
- GDK CLI 上传组件工件的 S3 存储桶的前缀。

3. 使用文本编辑器在名为的文件中创建组件源代码`main.py`。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

   ```
   nano main.py
   ```

   将以下 Python 代码复制到文件中。

   ```
   import json
   import os
   import sys
   import time
   ```

```
import traceback

from pathlib import Path

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

HELLO_WORLD_PRINT_INTERVAL = 15  # Seconds
DEFER_COMPONENT_UPDATE_INTERVAL = 30 * 1000  # Milliseconds


class BatteryAwareHelloWorldPrinter():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2, battery_file_path:
 Path, battery_threshold: float):
        self.battery_file_path = battery_file_path
        self.battery_threshold = battery_threshold
        self.ipc_client = ipc_client
        self.subscription_operation = None

    def on_component_update_event(self, event):
        try:
            if event.pre_update_event is not None:
                if self.is_battery_below_threshold():
                    self.defer_update(event.pre_update_event.deployment_id)
                    print('Deferred update for deployment %s' %
                          event.pre_update_event.deployment_id)
                else:
                    self.acknowledge_update(
                        event.pre_update_event.deployment_id)
                    print('Acknowledged update for deployment %s' %
                          event.pre_update_event.deployment_id)
            elif event.post_update_event is not None:
                print('Applied update for deployment')
        except:
            traceback.print_exc()

    def subscribe_to_component_updates(self):
        if self.subscription_operation == None:
            # SubscribeToComponentUpdates returns a tuple with the response and the
 operation.
            _, self.subscription_operation =
 self.ipc_client.subscribe_to_component_updates(
                on_stream_event=self.on_component_update_event)

    def close_subscription(self):
```

```
        if self.subscription_operation is not None:
            self.subscription_operation.close()
            self.subscription_operation = None

    def defer_update(self, deployment_id):
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id,
 recheck_after_ms=DEFER_COMPONENT_UPDATE_INTERVAL)

    def acknowledge_update(self, deployment_id):
        # Specify recheck_after_ms=0 to acknowledge a component update.
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id, recheck_after_ms=0)

    def is_battery_below_threshold(self):
        return self.get_battery_level() < self.battery_threshold

    def get_battery_level(self):
        # Read the battery level from the virtual battery level file.
        with self.battery_file_path.open('r') as f:
            data = json.load(f)
            return float(data['battery_level'])

    def print_message(self):
        message = 'Hello, World!'
        if self.is_battery_below_threshold():
            message += ' Battery level (%d) is below threshold (%d), so the
 component will defer updates' % (
                self.get_battery_level(), self.battery_threshold)
        else:
            message += ' Battery level (%d) is above threshold (%d), so the
 component will acknowledge updates' % (
                self.get_battery_level(), self.battery_threshold)
        print(message)


def main():
    # Read the battery threshold and virtual battery file path from command-line
 args.
    args = sys.argv[1:]
    battery_threshold = float(args[0])
    battery_file_path = Path(args[1])
    print('Reading battery level from %s and deferring updates when below %d' % (
        str(battery_file_path), battery_threshold))
```

```
    try:
        # Create an IPC client and a Hello World printer that defers component
 updates.
        ipc_client = GreengrassCoreIPCClientV2()
        hello_world_printer = BatteryAwareHelloWorldPrinter(
            ipc_client, battery_file_path, battery_threshold)
        hello_world_printer.subscribe_to_component_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                hello_world_printer.print_message()
                time.sleep(HELLO_WORLD_PRINT_INTERVAL)
        except InterruptedError:
            print('Subscription interrupted')
        hello_world_printer.close_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


if __name__ == '__main__':
    main()
```

这个 Python 应用程序执行以下操作：

- 从您稍后将在核心设备上创建的虚拟电池电量文件中读取核心设备的电池电量。这个虚拟电池电量文件模仿真实的电池，因此你可以在没有电池的核心设备上完成本教程。
- 读取电池电量阈值和虚拟电池电量文件路径的命令行参数。组件配方根据配置参数设置这些命令行参数，因此您可以在部署组件时自定义这些值。
- 使用AWS IoT Device SDK适用于 Python 的 v2 中的 IPC 客户端 V 2 与核心软件进行通信。AWS IoT Greengrass与最初的 IPC 客户端相比，IPC 客户端 V2 减少了在自定义组件中使用 IPC 所需编写的代码量。
- 使用 SubscribeToComponentUpdatesIPC 操作订阅更新通知。C AWS IoT Greengrass ore 软件在每次部署之前和之后都会发送通知。每次收到通知时，该组件都会调用以下函数。如果通知是针对即将部署的，则组件会检查电池电量是否低于阈值。如果电池电量低于阈值，则组件会使用 DeferComponentUpdateIPC 操作将更新推迟 30 秒。否则，如果电池电量不低于阈值，则组件会确认更新，因此更新可以继续进行。

```
def on_component_update_event(self, event):
    try:
        if event.pre_update_event is not None:
            if self.is_battery_below_threshold():
                self.defer_update(event.pre_update_event.deployment_id)
                print('Deferred update for deployment %s' %
                      event.pre_update_event.deployment_id)
            else:
                self.acknowledge_update(
                    event.pre_update_event.deployment_id)
                print('Acknowledged update for deployment %s' %
                      event.pre_update_event.deployment_id)
        elif event.post_update_event is not None:
            print('Applied update for deployment')
    except:
        traceback.print_exc()
```

> **Note**
>
> AWS IoT GreengrassCore 软件不发送本地部署的更新通知，因此您可以使用AWS IoT Greengrass云服务部署此组件来对其进行测试。

4. 使用文本编辑器在名为`recipe.json`或的文件中创建组件配方`recipe.yaml`。组件配方定义了组件的元数据、默认配置参数和平台特定的生命周期脚本。

   JSON

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

   ```
   nano recipe.json
   ```

   将以下 JSON 复制到文件中。

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "COMPONENT_NAME",
     "ComponentVersion": "COMPONENT_VERSION",
     "ComponentDescription": "This Hello World component defers updates when the
    battery level is below a threshold.",
     "ComponentPublisher": "COMPONENT_AUTHOR",
   ```

```
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "BatteryThreshold": 50,
      "LinuxBatteryFilePath": "/home/ggc_user/virtual_battery.json",
      "WindowsBatteryFilePath": "C:\\Users\\ggc_user\\virtual_battery.json"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk --upgrade",
        "run": "python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/LinuxBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk --upgrade",
        "run": "py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/WindowsBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    }
```

```
  ]
}
```

YAML

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano recipe.yaml
```

将以下 YAML 复制到文件中。

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "COMPONENT_NAME"
ComponentVersion: "COMPONENT_VERSION"
ComponentDescription: "This Hello World component defers updates when the
 battery level is below a threshold."
ComponentPublisher: "COMPONENT_AUTHOR"
ComponentConfiguration:
  DefaultConfiguration:
    BatteryThreshold: 50
    LinuxBatteryFilePath: "/home/ggc_user/virtual_battery.json"
    WindowsBatteryFilePath: "C:\\Users\\ggc_user\\virtual_battery.json"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk --upgrade
      run: python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/LinuxBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk --upgrade
      run: py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/WindowsBatteryFilePath}"
```

```
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
```

此配方指定了以下内容：

- 电池阈值、Linux 核心设备上的虚拟电池文件路径以及 Windows 核心设备上的虚拟电池文件路径的默认配置参数。
- 安装适用于 Python 的最新版本 AWS IoT Device SDK v2 的`install`生命周期。
- 在中运行 Python 应用程序的`run`生命周期`main.py`。
- 占位符，例如`COMPONENT_NAME`和`COMPONENT_VERSION`，其中 GDK CLI 在构建组件配方时会替换信息。

有关组件配方的更多信息，请参阅AWS IoT Greengrass组件配方参考。

步骤 3：将组件发布到AWS IoT Greengrass服务

在本节中，你将 Hello World 组件发布到AWS IoT Greengrass云服务。在AWS IoT Greengrass云服务中提供组件后，您可以将其部署到核心设备。您可以使用 GDK CLI 将组件从开发计算机发布到AWS IoT Greengrass云服务。GDK CLI 会为您上传组件的配方和工件。

将 Hello World 组件发布到该AWS IoT Greengrass服务

1. 运行以下命令使用 GDK CLI 构建组件。组件构建命令基于 GDK CLI 配置文件创建配方和工件。在此过程中，GDK CLI 会创建一个包含组件源代码的 ZIP 文件。

   ```
   gdk component build
   ```

   您应该会看到类似于以下示例的消息。

   ```
   [2022-04-28 11:20:16] INFO - Getting project configuration from gdk-config.json
   [2022-04-28 11:20:16] INFO - Found component recipe file 'recipe.yaml' in the
    project directory.
   [2022-04-28 11:20:16] INFO - Building the component
    'com.example.BatteryAwareHelloWorld' with the given project configuration.
   [2022-04-28 11:20:16] INFO - Using 'zip' build system to build the component.
   ```

```
[2022-04-28 11:20:16] WARNING - This component is identified as using 'zip' build
 system. If this is incorrect, please exit and specify custom build command in the
 'gdk-config.json'.
[2022-04-28 11:20:16] INFO - Zipping source code files of the component.
[2022-04-28 11:20:16] INFO - Copying over the build artifacts to the greengrass
 component artifacts build folder.
[2022-04-28 11:20:16] INFO - Updating artifact URIs in the recipe.
[2022-04-28 11:20:16] INFO - Creating component recipe in 'C:\Users\finthomp
\greengrassv2\com.example.BatteryAwareHelloWorld\greengrass-build\recipes'.
```

2. 运行以下命令将组件发布到AWS IoT Greengrass云服务。组件发布命令将组件的 ZIP 文件项目上传到 S3 存储桶。然后，它会在组件配方中更新 ZIP 文件的 S3 URI，并将配方上传到AWS IoT Greengrass服务。在此过程中，GDK CLI 会检查AWS IoT Greengrass云服务中已有哪个版本的 Hello World 组件，因此它可以选择该版本之后的下一个补丁版本。如果该组件尚不存在，GDK CLI 将使用版本`1.0.0`。

```
gdk component publish
```

您应该会看到类似于以下示例的消息。输出会告诉您 GDK CLI 创建的组件的版本。

```
[2022-04-28 11:20:29] INFO - Getting project configuration from gdk-config.json
[2022-04-28 11:20:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2022-04-28 11:20:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2022-04-28 11:20:30] INFO - No private version of the component
 'com.example.BatteryAwareHelloWorld' exist in the account. Using '1.0.0' as the
 next version to create.
[2022-04-28 11:20:30] INFO - Publishing the component
 'com.example.BatteryAwareHelloWorld' with the given project configuration.
[2022-04-28 11:20:30] INFO - Uploading the component built artifacts to s3 bucket.
[2022-04-28 11:20:30] INFO - Uploading component artifacts to S3
 bucket: greengrass-component-artifacts-us-west-2-123456789012. If this is your
 first time using this bucket, add the 's3:GetObject' permission to each core
 device's token exchange role to allow it to download the component artifacts. For
 more information, see https://docs.aws.amazon.com/greengrass/v2/developerguide/
device-service-role.html.
[2022-04-28 11:20:30] INFO - Not creating an artifacts bucket as it already exists.
[2022-04-28 11:20:30] INFO - Updating the component recipe
 com.example.BatteryAwareHelloWorld-1.0.0.
[2022-04-28 11:20:31] INFO - Creating a new greengrass component
 com.example.BatteryAwareHelloWorld-1.0.0
```

```
[2022-04-28 11:20:31] INFO - Created private version '1.0.0' of the component in
 the account.'com.example.BatteryAwareHelloWorld'.
```

3. 从输出中复制 S3 存储桶名称。您稍后使用存储桶名称来允许核心设备从该存储桶下载组件工件。
4. （可选）在AWS IoT Greengrass控制台中查看组件以验证其是否成功上传。执行以下操作：
   a. 在AWS IoT Greengrass控制台导航菜单中，选择组件。
   b. 在 “组件” 页面上，选择 “我的组件” 选项卡，然后选择com.example.BatteryAwareHelloWorld。

      在此页面上，您可以看到组件的配方以及有关该组件的其他信息。
5. 允许核心设备访问 S3 存储桶中的组件工件。

   每台核心设备都有一个核心设备 IAM 角色，允许其与云进行交互AWS IoT并将日志发送到AWS云端。默认情况下，此设备角色不允许访问 S3 存储桶，因此您必须创建并附加允许核心设备从 S3 存储桶检索组件工件的策略。

   如果您的设备角色已允许访问 S3 存储桶，则可以跳过此步骤。否则，请创建允许访问的 IAM 策略并将其附加到该角色，如下所示：
   a. 在 IAM 控制台导航菜单中，选择策略，然后选择创建策略。
   b. 在 JSON 选项卡中，将占位符内容替换为以下策略。将 *greengrass-component-artifacts-us-west-2-123456789012* 替换为 GDK CLI 上传组件工件的 S3 存储桶的名称。

      例如，如果您在 GDK CLI 配置文件**us-west-2**中指定了**greengrass-component-artifacts**和，而您的 AWS 账户 ID 为**123456789012**，则 GDK CLI 将使用名为的 S3 存储桶。greengrass-component-artifacts-us-west-2-123456789012

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::greengrass-component-artifacts-us-
west-2-123456789012/*"
    }
```

```
    ]
}
```

c. 请选择 Next（下一步）。
d. 在 “策略详细信息” 部分的 “名称” 中，输入**MyGreengrassV2ComponentArtifactPolicy**。
e. 选择创建策略。
f. 在 IAM 控制台导航菜单中，选择角色，然后选择核心设备的角色名称。您在安装 C AWS IoT Greengrass ore 软件时指定了此角色名称。如果您未指定名称，则默认为GreengrassV2TokenExchangeRole。
g. 在权限下，选择添加权限，然后选择附加策略。
h. 在添加权限页面上，选中您创建的MyGreengrassV2ComponentArtifactPolicy策略旁边的复选框，然后选择添加权限。

步骤 4：在核心设备上部署和测试组件

在本节中，您将组件部署到核心设备以测试其功能。在核心设备上，您可以创建虚拟电池电量文件来模仿真实的电池。然后，您可以创建其他部署并观察核心设备上的组件日志文件，以查看组件延迟和确认更新。

部署和测试延迟更新的 Hello World 组件

1. 使用文本编辑器创建虚拟电池电量文件。此文件模仿真实电池。

   - 在 Linux 核心设备上，创建一个名为的文件/home/ggc_user/virtual_battery.json。使用sudo权限运行文本编辑器。
   - 在 Windows 核心设备上，创建一个名为的文件C:\Users\ggc_user\virtual_battery.json。以管理员身份运行文本编辑器。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

   ```
   sudo nano /home/ggc_user/virtual_battery.json
   ```

   将以下 JSON 复制到文件中。

   ```
   {
     "battery_level": 50
   ```

```
}
```

2. 将 Hello World 组件部署到核心设备。执行以下操作：

   a. 在[AWS IoT Greengrass控制台](https://console.aws.amazon.com/greengrass)导航菜单中，选择组件。
   b. 在 “组件” 页面上，选择 “我的组件” 选项卡，然后选择com.example.BatteryAwareHelloWorld。
   c. 在 com.example.BatteryAwareHelloWorld 页面上，选择部署。
   d. 从 “添加到部署” 中，选择要修改的现有部署，或者选择创建新部署，然后选择 “下一步”。
   e. 如果您选择创建新部署，请为部署选择目标核心设备或事物组。在 “指定目标” 页面的 “部署目标” 下，选择核心设备或事物组，然后选择下一步。
   f. 在 “选择组件” 页面上，确认已选择该com.example.BatteryAwareHelloWorld组件，然后选择 “下一步”。
   g. 在 “配置组件” 页面上 com.example.BatteryAwareHelloWorld，选择，然后执行以下操作：

      i. 选择配置组件。
      ii. 在 “配置” com.example.BatteryAwareHelloWorld 模式的 “配置更新” 下，在 “要合并的配置” 中，输入以下配置更新。

      ```
      {
        "BatteryThreshold": 70
      }
      ```

      iii. 选择 “确认” 关闭模式，然后选择 “下一步”。
   h. 在 “确认高级设置” 页面的 “部署策略” 部分的 “组件更新策略” 下，确认已选中 “通知组件”。创建新部署时，默认情况下会选择 “通知组件”。
   i. 在 Review（检查）页上，选择 Deploy（部署）。

      部署最多可能需要一分钟才能完成。

3. AWS IoT GreengrassCore 软件将组件进程中的 stdout 保存到文件夹中的日志文件中。`logs`运行以下命令以验证 Hello World 组件是否运行并打印状态消息。

   Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

您应该会看到类似于以下示例的消息。

```
Hello, World! Battery level (50) is below threshold (70), so the component will
 defer updates.
```

Note

如果文件不存在，则部署可能尚未完成。如果文件在 30 秒内不存在，则部署可能失败。例如，如果核心设备无权从 S3 存储桶下载组件的项目，则可能会发生这种情况。运行以下命令查看AWS IoT Greengrass核心软件日志文件。此文件包含来自 Greengrass 核心设备部署服务的日志。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

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

该type命令将文件内容写入终端。多次运行此命令以观察文件中的更改。

PowerShell

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

4. 在核心设备上创建新的部署，以验证该组件是否推迟了更新。执行以下操作：

   a. 在AWS IoT Greengrass控制台导航菜单中，选择部署。

b. 选择您之前创建或修改的部署。

c. 在部署页面上，选择修订。

d. 在 “修订部署” 模式中，选择 “修订部署”。

e. 在每个步骤中选择 “下一步”，然后选择 “部署”。

5. 运行以下命令以再次查看组件的日志，并验证它是否推迟了更新。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

您应该会看到类似于以下示例的消息。该组件将更新推迟 30 秒，因此该组件会重复打印此消息。

```
Deferred update for deployment 50722a95-a05f-4e2a-9414-da80103269aa.
```

6. 使用文本编辑器编辑虚拟电池电量文件并将电池电量更改为高于阈值的值，这样部署就可以继续进行。

• 在 Linux 核心设备上，编辑名为的文件/home/ggc_user/virtual_battery.json。使用sudo权限运行文本编辑器。

• 在 Windows 核心设备上，编辑名为的文件C:\Users\ggc_user\virtual_battery.json。以管理员身份运行文本编辑器。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
sudo nano /home/ggc_user/virtual_battery.json
```

将电池电量更改为80。

```
{
  "battery_level": 80
}
```

7. 运行以下命令以再次查看组件的日志，并验证它是否确认更新。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   您应该会看到与以下示例类似的消息。

   ```
   Hello, World! Battery level (80) is above threshold (70), so the component will
    acknowledge updates.
   Acknowledged update for deployment f9499eb2-4a40-40a7-86c1-c89887d859f1.
   ```

你已经完成了本教程。Hello World 组件会根据核心设备的电池电量来推迟或确认更新。有关本教程所探讨的主题的更多信息，请参阅以下内容：

- 开发AWS IoT Greengrass组件
- 将AWS IoT Greengrass组件部署到设备
- 使用AWS IoT Device SDK与 Greengrass 原子核、其他组件进行通信 AWS IoT Core
- AWS IoT Greengrass开发套件命令行界面

教程：通过 MQTT 与本地物联网设备交互

您可以完成本教程，将核心设备配置为与通过 MQTT 连接到核心设备的本地 IoT 设备（称为客户端设备）进行交互。在本教程中，AWS IoT您将配置为使用云发现作为客户端设备连接到核心设备。配置云发现时，客户端设备可以向AWS IoT Greengrass云服务发送请求以发现核心设备。来自的响应AWS IoT Greengrass包括您配置客户端设备以发现的核心设备的连接信息和证书。然后，客户端设备可以使用此信息连接到可用的核心设备，在那里它可以通过 MQTT 进行通信。

在本教程中，您将执行以下操作：

1. 如有必要，请查看并更新核心设备的权限。
2. 将客户端设备与核心设备关联，以便他们可以使用云发现来发现核心设备。
3. 将 Greengrass 组件部署到核心设备以启用客户端设备支持。
4. 将客户端设备连接到核心设备并测试与AWS IoT Core云服务的通信。
5. 开发一个与客户端设备通信的自定义 Greengrass 组件。
6. 开发一个与客户端设备的AWS IoT设备影子交互的自定义组件。

本教程使用单核设备和单个客户端设备。您也可以按照教程连接和测试多个客户端设备。

在本教程中，预计花费 30-60 分钟。

先决条件

要完成本教程，您需要：

- AWS 账户。如果没有，请参阅设置一个 AWS 账户。
- 具有管理员权限的 AWS Identity and Access Management (IAM) 用户。
- 一款 Greengrass 核心设备。有关如何设置核心设备的更多信息，请参阅设置AWS IoT Greengrass 核心设备。
  - 核心设备必须运行 Greengrass nucleus v2.6.0 或更高版本。此版本包括对本地发布/订阅通信中的通配符的支持以及对客户端设备影子的支持。

Note

客户端设备支持需要 Greengrass nucleus v2.0 或更高版本。但是，本教程探讨了较新的功能，例如在本地发布/订阅中支持 MQTT 通配符以及支持客户端设备影子。这些功能需要 Greengrass nucleus v2.6.0 或更高版本。

- 核心设备必须与客户端设备位于同一个网络上才能连接。
- （可选）要完成开发自定义 Greengrass 组件的模块，核心设备必须运行 Greengrass CLI。有关更多信息，请参阅 安装 Greengrass CLI。

- 在本教程中，可以作为客户端设备进行连接。AWS IoT有关更多信息，请参阅《AWS IoT Core开发人员指南》中的创建AWS IoT资源。
  - 客户端设备的AWS IoT策略必须允许该`greengrass:Discover`权限。有关更多信息，请参阅 客户端设备的最低AWS IoT政策。
  - 客户端设备必须与核心设备位于同一个网络上。
  - 客户端设备必须运行 Python 3。
  - 客户端设备必须运行 Git。

步骤 1：查看并更新核心设备AWS IoT政策

要支持客户端设备，核心设备的AWS IoT策略必须允许以下权限：

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`—（可选）使用 IP 检测器组件需要此权限，该组件将核心设备的网络连接信息报告给AWS IoT Greengrass云服务。

有关核心设备的这些权限和AWS IoT策略的更多信息，请参阅数据层面操作的 AWS IoT 策略和支持客户端设备的最低AWS IoT政策。

在本节中，您将查看核心设备的AWS IoT策略并添加缺少的所有必需权限。如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则您的核心设备具有允许访问所有AWS IoT Greengrass操

作的AWS IoT策略（`greengrass:*`）。在这种情况下，只有当您计划将影子管理器组件配置为与其同步设备影子时，才必须更新AWS IoT策略AWS IoT Core。否则，你可以跳过本节。

查看和更新核心设备的AWS IoT政策

1. 在AWS IoT Greengrass控制台导航菜单中，选择核心设备。
2. 在核心设备页面上，选择要更新的核心设备。
3. 在核心设备详细信息页面上，选择指向核心设备的 Thin g 的链接。此链接可在AWS IoT控制台中打开事物详细信息页面。
4. 在事物详细信息页面上，选择证书。
5. 在 “证书” 选项卡中，选择事物的有效证书。
6. 在证书详细信息页面上，选择策略。
7. 在 “策略” 选项卡中，选择要查看和更新的AWS IoT策略。您可以向附加到核心设备活动证书的任何策略添加所需的权限。

> **Note**
>
> 如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则有两个AWS IoT策略。我们建议您选择名为GreengrassV2IoTThingPolicy的策略（如果存在）。默认情况下，使用快速安装程序创建的核心设备使用此策略名称。如果您向此策略添加权限，则也将这些权限授予使用此策略的其他核心设备。

8. 在策略概述中，选择编辑活动版本。
9. 查看策略以获取所需权限，然后添加缺少的所有必需权限。
10. 要将新的策略版本设置为活动版本，请在策略版本状态下，选择将编辑后的版本设置为该策略的活动版本。
11. 选择另存为新版本。

步骤 2：启用客户端设备支持

要使客户端设备使用云发现连接到核心设备，必须关联这些设备。当您将客户端设备与核心设备关联时，可以使该客户端设备检索核心设备的 IP 地址和证书以用于连接。

要使客户端设备能够安全地连接到核心设备并与 Greengrass 组件通信，请将以下 Greengrass 组件AWS IoT Core部署到核心设备：

- 客户端设备身份验证 (`aws.greengrass.clientdevices.Auth`)

  部署客户端设备身份验证组件以对客户端设备进行身份验证并授权客户端设备操作。这个组件允许你的AWS IoT东西连接到核心设备。

  此组件需要一些配置才能使用。您必须指定客户端设备组以及每个组有权执行的操作，例如通过 MQTT 进行连接和通信。有关更多信息，请参阅客户端设备身份验证组件配置。
- MQTT 3.1.1 经纪商 (Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  部署 Moquette MQTT 代理组件来运行轻量级 MQTT 代理。Moquette MQTT 代理符合 MQTT 3.1.1，包括对 QoS 0、QoS 1、QoS 2、保留消息、最后遗嘱消息和永久订阅的本地支持。

  您无需配置此组件即可使用它。但是，您可以配置此组件运行 MQTT 代理的端口。默认情况下，它使用端口 8883。
- MQTT 网桥 (`aws.greengrass.clientdevices.mqtt.Bridge`)

  （可选）部署 MQTT 桥接组件，以便在客户端设备（本地 MQTT）、本地发布/订阅和 MQTT 之间中继消息。AWS IoT Core将此组件配置为与 Greengrass 组件中的客户端设备同步AWS IoT Core并与客户端设备进行交互。

  此组件需要配置才能使用。您必须指定此组件中继消息的主题映射。有关更多信息，请参阅 MQTT 网桥组件配置。
- IP 探测器 (`aws.greengrass.clientdevices.IPDetector`)

  （可选）部署 IP 检测器组件，自动向AWS IoT Greengrass云服务报告核心设备的 MQTT 代理端点。如果您的网络设置很复杂，例如路由器将 MQTT 代理端口转发到核心设备的网络设置，则无法使用此组件。

  您无需配置此组件即可使用它。

在本节中，您将使用AWS IoT Greengrass控制台关联客户端设备并将客户端设备组件部署到核心设备。

启用客户端设备支持

1. 导航到 AWS IoT Greengrass 控制台。
2. 在左侧导航菜单中，选择核心设备。
3. 在核心设备页面上，选择要在其中启用客户端设备支持的核心设备。

4. 在核心设备详细信息页面上，选择客户端设备选项卡。
5. 在 “客户端设备” 选项卡上，选择 “配置云发现”。

   将打开 “配置核心设备” 发现页面。在此页面上，您可以将客户端设备与核心设备关联并部署客户端设备组件。本页在步骤 1：选择目标核心设备中为您选择核心设备。

   > **Note**
   >
   > 您也可以使用此页为事物组配置核心设备发现。如果选择此选项，则可以将客户端设备组件部署到事物组中的所有核心设备。但是，如果选择此选项，则必须在创建部署后手动将客户端设备与每台核心设备关联起来。在本教程中，您将配置单核设备。

6. 在步骤 2：关联客户端设备中，将客户端设备的设备AWS IoT与核心设备相关联。这使客户端设备能够使用云发现来检索核心设备的连接信息和证书。执行以下操作：

   a. 选择 “关联客户端设备”。
   b. 在 “将客户端设备与核心设备关联” 模式中，输入要关联AWS IoT的事物的名称。
   c. 选择添加。
   d. 选择关联。

7. 在步骤 3：配置和部署 Greengrass 组件中，部署组件以启用客户端设备支持。如果目标核心设备以前部署过，则此页面将修改该部署。否则，此页面将为核心设备创建新的部署。要配置和部署客户端设备组件，请执行以下操作：

   a. 核心设备必须运行 Greengrass nucleus v2.6.0 或更高版本才能完成本教程。如果核心设备运行的是早期版本，请执行以下操作：

      i. 选中该框以部署组aws.greengrass.Nucleus件。
      ii. 对于组aws.greengrass.Nucleus件，选择编辑配置。
      iii. 对于组件版本，请选择版本 2.6.0 或更高版本。
      iv. 选择确认。

      > **Note**
      >
      > 如果您将 Greengrass nucleus 从较早的次要版本升级，并且核心设备AWS运行依赖于该核的组件，则还必须将提供的组件更新到较新的版本。AWS在本教程的后面部

分中查看部署时，可以配置这些组件的版本。有关更多信息，请参阅 更新AWS IoT Greengrass核心软件 (OTA)。

b. 对于组aws.greengrass.clientdevices.Auth件，选择编辑配置。

c. 在客户端设备身份验证组件的编辑配置模式中，配置授权策略，允许客户端设备在核心设备上发布和订阅 MQTT 代理。执行以下操作：

   i. 在 “配置” 下的 “合并代码配置” 块中，输入以下配置，其中包含客户端设备授权策略。每个设备组授权策略都指定了一组操作以及客户端设备可以用来执行这些操作的资源。

      • 此策略允许名称MyClientDevice以开头的客户端设备就所有 MQTT 主题进行连接和通信。将 *MyClientDevice** 替换为要作为客户端设备连接AWS IoT的事物的名称。您也可以使用*通配符指定与客户端设备名称相匹配的名称。通*配符必须位于名称的末尾。

        如果您要连接第二台客户端设备，请将 *MyOtherClientDevice** 替换为该客户端设备的名称或与该客户端设备名称匹配的通配符模式。否则，您可以删除或保留选择规则的这一部分，该部分允许名称匹配的客户端设备MyOtherClientDevice*进行连接和通信。

      • 此策略还使用OR运算符来允许名称MyOtherClientDevice以开头的客户端设备就所有 MQTT 主题进行连接和通信。您可以删除选择规则中的此子句，也可以对其进行修改以匹配要连接的客户端设备。

      • 此政策允许客户端设备发布和订阅所有 MQTT 主题。要遵循最佳安全实践，请将mqtt:publish和mqtt:subscribe操作限制在客户端设备用于通信的最少一组话题上。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice* OR
 thingName: MyOtherClientDevice*",
        "policyName": "MyClientDevicePolicy"
      }
    },
    "policies": {
      "MyClientDevicePolicy": {
```

```
          "AllowConnect": {
            "statementDescription": "Allow client devices to connect.",
            "operations": [
              "mqtt:connect"
            ],
            "resources": [
              "*"
            ]
          },
          "AllowPublish": {
            "statementDescription": "Allow client devices to publish to all
 topics.",
            "operations": [
              "mqtt:publish"
            ],
            "resources": [
              "*"
            ]
          },
          "AllowSubscribe": {
            "statementDescription": "Allow client devices to subscribe to all
 topics.",
            "operations": [
              "mqtt:subscribe"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  }
```

有关更多信息，请参阅客户端设备身份验证组件配置。

ii. 选择确认。

d. 对于组aws.greengrass.clientdevices.mqtt.Bridge件，选择编辑配置。

e. 在 MQTT 网桥组件的编辑配置模式中，配置主题映射，将 MQTT 消息从客户端设备中继到。AWS IoT Core执行以下操作：

i. 在 “配置” 下的 “合并代码配置” 块中，输入以下配置。此配置指定将`clients/+/hello/world`主题过滤器上的 MQTT 消息从客户端设备中继到AWS IoT Core云服务。例如，此主题筛选条件与`clients/MyClientDevice1/hello/world`主题匹配。

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

有关更多信息，请参阅 MQTT 网桥组件配置。

ii. 选择确认。

8. 选择 “查看并部署” 以查看此页为您创建的部署。
9. 如果您之前未在此区域设置过 Greengrass 服务角色，则控制台会打开一个模式来为您设置服务角色。客户端设备身份验证组件使用此服务角色来验证客户端设备的身份，IP 检测器组件使用此服务角色来管理核心设备的连接信息。选择 Grant permissions（授予权限）。
10. 在 “查看” 页面上，选择 “部署” 以开始部署到核心设备。
11. 要验证部署是否成功，请检查部署状态并检查核心设备上的日志。要查看核心设备上的部署状态，可以在部署概述中选择 Targ et。有关更多信息，请参阅下列内容：
    - 检查部署状态
    - 监控AWS IoT Greengrass日志

步骤 3：Connect 客户端设备

客户端设备可以使用AWS IoT Device SDK来发现、连接核心设备并与之通信。客户端设备必须是一个AWS IoT东西。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的创建事物对象。

在本节中，您将安装适用于 Python 的 AWS IoT Device SDK v2 并从中运行 Greengrass 发现示例应用程序。AWS IoT Device SDK

AWS IoT Device SDK还提供其他编程语言版本。本教程使用适用于 Python 的 AWS IoT Device SDK v2，但你可以根据自己的用例探索其他 SDK。有关更多信息，请参阅AWS IoT Core 开发人员指南中的 AWS IoT Device 软件开发工具包。

将客户端设备连接到核心设备

1. 下载适用于 Python 的 AWS IoT Device SDK v2 并将其安装到要作为客户端设备连接的设备上。AWS IoT

   在客户端设备上，执行以下操作：

   a. 克隆适用于 Python 的 AWS IoT Device SDK v2 版本库进行下载。

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. 安装适用于 Python 的 AWS IoT Device SDK v2。

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. 在 python 版 AWS IoT Device SDK v2 中切换到示例文件夹。

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

3. 运行示例 Greengrass 发现应用程序。此应用程序需要指定客户端设备事物名称、要使用的 MQTT 主题和消息以及用于验证和保护连接的证书的参数。以下示例向`clients/MyClientDevice1/hello/world`主题发送了 Hello World 消息。

   本主题与您配置 MQTT 网桥以将消息中继到AWS IoT Core之前的主题相匹配。

   - 将 *MyClientDevice1* 替换为客户端设备的事物名称。
   - 将 *~/certs/ AmazonRoot ca1.pem #*换为客户端设备上亚马逊根 CA 证书的路径。
   - 将 *~/certs/device.pem.crt ########*上设备证书的路径。
   - 将 *~/certs/private.pem.key ###########*文件的路径。

- 将 *us-east-1* 替换为您的客户端设备和核心设备运行的AWS区域。

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

发现示例应用程序发送消息 10 次并断开连接。它还订阅发布消息的同一主题。如果输出显示应用程序收到了有关该主题的 MQTT 消息，则客户端设备可以成功地与核心设备通信。

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...
```

```
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

如果应用程序改为输出错误，请参阅 Greengrass 发现问题疑难解答。

您还可以查看核心设备上的 Greengrass 日志，以验证客户端设备是否成功连接和发送消息。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

4. 验证 MQTT 网桥是否将来自客户端设备的消息中继到。AWS IoT Core您可以在AWS IoT Core控制台中使用 MQTT 测试客户端订阅 MQTT 主题筛选器。执行以下操作：

   a. 导航到 AWS IoT 控制台。
   b. 在左侧导航菜单的测试下，选择 MQTT 测试客户端。
   c. 在 “订阅主题” 选项卡上，在 “主题筛选器” 中，输入`clients/+/hello/world`订阅来自核心设备的客户端设备消息。
   d. 选择订阅。
   e. 再次在客户端设备上运行发布/订阅应用程序。

      MQTT 测试客户端显示您从客户端设备发送的与该主题过滤器匹配的主题的消息。

步骤 4：开发可与客户端设备通信的组件

您可以开发与客户端设备通信的 Greengrass 组件。组件使用进程间通信 (IPC) 和本地发布/订阅接口在核心设备上进行通信。要与客户端设备交互，请将 MQTT 桥接组件配置为在客户端设备和本地发布/订阅接口之间中继消息。

在本节中，您将更新 MQTT 桥接组件，以将来自客户端设备的消息中继到本地发布/订阅接口。然后，你开发一个订阅这些消息并在收到消息时打印这些消息的组件。

开发可与客户端设备通信的组件

1. 修改核心设备的部署，并配置 MQTT 网桥组件以将来自客户端设备的消息中继到本地发布/订阅。执行以下操作：

   a. 导航到 AWS IoT Greengrass 控制台。

b. 在左侧导航菜单中，选择核心设备。

c. 在核心设备页面上，选择本教程中使用的核心设备。

d. 在核心设备详细信息页面上，选择客户端设备选项卡。

e. 在 “客户端设备” 选项卡上，选择 “配置云发现”。

   将打开 “配置核心设备” 发现页面。在此页面上，您可以更改或配置部署到核心设备的客户端设备组件。

f. 在步骤 3 中，对于组aws.greengrass.clientdevices.mqtt.Bridge件，选择编辑配置。

g. 在 MQTT 网桥组件的编辑配置模式中，配置主题映射，将 MQTT 消息从客户端设备中继到本地发布/订阅接口。执行以下操作：

   i. 在 “配置” 下的 “合并代码配置” 块中，输入以下配置。此配置指定将与主题过滤器匹配的主题上的 MQTT 消息从客户端设备中继到AWS IoT Core云服务和`clients/+/hello/world`本地 Greengrass 发布/订阅代理。

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

   有关更多信息，请参阅 MQTT 网桥组件配置。

   ii. 选择确认。

h. 选择 “查看并部署” 以查看此页为您创建的部署。

i. 在 “查看” 页面上，选择 “部署” 以开始部署到核心设备。

j. 要验证部署是否成功，请检查部署状态并检查核心设备上的日志。要查看核心设备上的部署状态，可以在部署概述中选择 Targ et。有关更多信息，请参阅下列内容：

   • 检查部署状态

   - 监控AWS IoT Greengrass日志

2. 开发和部署一个 Greengrass 组件，用于订阅来自客户端设备的 Hello World 消息。执行以下操作：

   a. 在核心设备上为配方和工件创建文件夹。

      Linux or Unix

      ```
      mkdir recipes
      mkdir -p artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0
      ```

      Windows Command Prompt (CMD)

      ```
      mkdir recipes
      mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
      ```

      PowerShell

      ```
      mkdir recipes
      mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
      ```

      > **Important**
      >
      > 必须使用以下格式作为对象文件夹路径。包括您在配方中指定的组件名称和版本。
      >
      > ```
      > artifacts/componentName/componentVersion/
      > ```

   b. 使用文本编辑器创建包含以下内容的组件配方。此配方指定安装适用于 Python 的 AWS IoT Device SDK v2 并运行订阅主题并打印消息的脚本。

      例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

      ```
      nano recipes/com.example.clientdevices.MyHelloWorldSubscriber-1.0.0.json
      ```

      将以下配方复制到文件中。

      ```
      {
      ```

```
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.clientdevices.MyHelloWorldSubscriber",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to Hello World messages
 from client devices.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MyHelloWorldSubscriber:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/hello_world_subscriber.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/hello_world_subscriber.py"
      }
    }
  ]
```

```
}
```

c. 使用文本编辑器创建名`hello_world_subscriber.py`为以下内容的 Python 脚本构件。此应用程序使用发布/订阅 IPC 服务来订阅`clients/+/hello/world`主题并打印它收到的消息。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0/
hello_world_subscriber.py
```

将以下 Python 代码复制到文件中。

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

CLIENT_DEVICE_HELLO_WORLD_TOPIC = 'clients/+/hello/world'
TIMEOUT = 10


def on_hello_world_message(event):
    try:
        message = str(event.binary_message.message, 'utf-8')
        print('Received new message: %s' % message)
    except:
        traceback.print_exc()


try:
    ipc_client = GreengrassCoreIPCClientV2()

    # SubscribeToTopic returns a tuple with the response and the operation.
    _, operation = ipc_client.subscribe_to_topic(
        topic=CLIENT_DEVICE_HELLO_WORLD_TOPIC,
 on_stream_event=on_hello_world_message)
    print('Successfully subscribed to topic: %s' %
          CLIENT_DEVICE_HELLO_WORLD_TOPIC)

    # Keep the main thread alive, or the process will exit.
```

```
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')

    operation.close()
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Note

此组件使用AWS IoT Device SDK适用于 Python 的 v2 中的 IPC 客户端 V2 与核心软件进行通信。AWS IoT Greengrass与最初的 IPC 客户端相比，IPC 客户端 V2 减少了在自定义组件中使用 IPC 所需编写的代码量。

d. 使用 Greengrass CLI 部署该组件。

Linux or Unix

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

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

PowerShell

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

3. 查看组件日志，验证组件是否成功安装并订阅了该主题。

   Linux or Unix

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

   PowerShell

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

   您可以保持日志源处于打开状态，以验证核心设备是否收到消息。

4. 在客户端设备上，再次运行示例 Greengrass 发现应用程序，向核心设备发送消息。

   ```
   python3 basic_discovery.py \\
     --thing_name MyClientDevice1 \\
     --topic 'clients/MyClientDevice1/hello/world' \\
     --message 'Hello World!' \\
     --ca_file ~/certs/AmazonRootCA1.pem \\
     --cert ~/certs/device.pem.crt \\
     --key ~/certs/private.pem.key \\
     --region us-east-1 \\
     --verbosity Warn
   ```

5. 再次查看组件日志，以验证该组件是否接收并打印了来自客户端设备的消息。

   Linux or Unix

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

   PowerShell

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

步骤 5：开发与客户端设备影子交互的组件

您可以开发与客户端设备的设备影子交互的 Greengrass 组件。AWS IoT影子是一个 JSON 文档，用于存储诸如客户端设备AWS IoT之类的当前或所需状态信息。即使客户端设备未连接到，自定义组件也可以访问客户端设备的影子以AWS IoT管理其状态。每AWS IoT件事物都有一个未命名的阴影，你也可以为每个事物创建多个命名的阴影。

在本节中，您将部署影子管理器组件来管理核心设备上的阴影。您还可以更新 MQTT 网桥组件，以便在客户端设备和影子管理器组件之间中继影子消息。然后，开发一个用于更新客户端设备影子的组件，并在客户端设备上运行一个示例应用程序，以响应来自该组件的影子更新。该组件代表智能灯光管理应用程序，其中核心设备管理作为客户端设备连接到它的智能灯的颜色状态。

开发与客户端设备影子交互的组件

1. 修改核心设备的部署以部署影子管理器组件，并配置 MQTT 网桥组件以在客户端设备和影子管理器通信的本地发布/订阅之间中继影子消息。执行以下操作：

   a. 导航到 AWS IoT Greengrass 控制台。
   b. 在左侧导航菜单中，选择核心设备。
   c. 在核心设备页面上，选择本教程中使用的核心设备。
   d. 在核心设备详细信息页面上，选择客户端设备选项卡。
   e. 在 “客户端设备” 选项卡上，选择 “配置云发现”。

      将打开 “配置核心设备” 发现页面。在此页面上，您可以更改或配置部署到核心设备的客户端设备组件。
   f. 在步骤 3 中，对于组aws.greengrass.clientdevices.mqtt.Bridge件，选择编辑配置。
   g. 在 MQTT bridge 组件的编辑配置模式中，配置主题映射，以便在客户端设备和本地发布/订阅接口之间中继设备影子主题上的 MQTT 消息。您还要确认部署指定了兼容的 MQTT 桥接版本。客户端设备影子支持需要 MQTT 网桥 v2.2.0 或更高版本。执行以下操作：

      i. 对于组件版本，请选择版本 2.2.0 或更高版本。
      ii. 在 “配置” 下的 “合并代码配置” 块中，输入以下配置。此配置指定在影子主题上中继 MQTT 消息。

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
```

```
        "source": "LocalMqtt",
        "target": "IotCore"
      },
      "HelloWorldPubsubMapping": {
        "topic": "clients/+/hello/world",
        "source": "LocalMqtt",
        "target": "Pubsub"
      },
      "ShadowsLocalMqttToPubsub": {
        "topic": "$aws/things/+/shadow/#",
        "source": "LocalMqtt",
        "target": "Pubsub"
      },
      "ShadowsPubsubToLocalMqtt": {
        "topic": "$aws/things/+/shadow/#",
        "source": "Pubsub",
        "target": "LocalMqtt"
      }
    }
  }
```

有关更多信息，请参阅 MQTT 网桥组件配置。

iii. 选择确认。

h. 在步骤 3 中，选择要部署的aws.greengrass.ShadowManager组件。

i. 选择 “查看并部署” 以查看此页为您创建的部署。

j. 在 “查看” 页面上，选择 “部署” 以开始部署到核心设备。

k. 要验证部署是否成功，请检查部署状态并检查核心设备上的日志。要查看核心设备上的部署状态，可以在部署概述中选择 Targ et。有关更多信息，请参阅下列内容：

- 检查部署状态
- 监控AWS IoT Greengrass日志

2. 开发和部署用于管理智能轻型客户端设备的 Greengrass 组件。执行以下操作：

a. 在核心设备上为组件的工件创建一个文件夹。

Linux or Unix

```
mkdir -p artifacts/com.example.clientdevices.MySmartLightManager/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

PowerShell

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

Important

必须使用以下格式作为对象文件夹路径。包括您在配方中指定的组件名称和版本。

```
artifacts/componentName/componentVersion/
```

b. 使用文本编辑器创建包含以下内容的组件配方。此配方指定安装适用于 Python 的 AWS IoT Device SDK v2 并运行一个脚本，该脚本与智能灯光客户端设备的阴影交互以管理其颜色。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano recipes/com.example.clientdevices.MySmartLightManager-1.0.0.json
```

将以下配方复制到文件中。

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.clientdevices.MySmartLightManager",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that interacts with smart light client
 devices.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.Nucleus": {
      "VersionRequirement": "^2.6.0"
    },
    "aws.greengrass.ShadowManager": {
      "VersionRequirement": "^2.2.0"
    },
    "aws.greengrass.clientdevices.mqtt.Bridge": {
```

```
      "VersionRequirement": "^2.2.0"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "smartLightDeviceNames": [],
      "accessControl": {
        "aws.greengrass.ShadowManager": {
          "com.example.clientdevices.MySmartLightManager:shadow:1": {
            "policyDescription": "Allows access to client devices' unnamed
 shadows",
            "operations": [
              "aws.greengrass#GetThingShadow",
              "aws.greengrass#UpdateThingShadow"
            ],
            "resources": [
              "$aws/things/MyClientDevice*/shadow"
            ]
          }
        },
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MySmartLightManager:pubsub:1": {
            "policyDescription": "Allows access to client devices' unnamed
 shadow updates",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "$aws/things/+/shadow/update/accepted"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/smart_light_manager.py"
      }
```

```
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/smart_light_manager.py"
      }
    }
  ]
}
```

c. 使用文本编辑器创建名`smart_light_manager.py`为以下内容的 Python 脚本构件。此应用程序使用影子 IPC 服务来获取和更新客户端设备影子，使用本地发布/订阅 IPC 服务来接收报告的影子更新。

例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano artifacts/com.example.clientdevices.MySmartLightManager/1.0.0/
smart_light_manager.py
```

将以下 Python 代码复制到文件中。

```
import json
import random
import sys
import time
import traceback
from uuid import uuid4

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import ResourceNotFoundError

SHADOW_COLOR_PROPERTY = 'color'
CONFIGURATION_CLIENT_DEVICE_NAMES = 'smartLightDeviceNames'
COLORS = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
SHADOW_UPDATE_TOPIC = '$aws/things/+/shadow/update/accepted'
SET_COLOR_INTERVAL = 15



class SmartLightDevice():
```

```
    def __init__(self, client_device_name: str, reported_color: str = None):
        self.name = client_device_name
        self.reported_color = reported_color
        self.desired_color = None


class SmartLightDeviceManager():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2):
        self.ipc_client = ipc_client
        self.devices = {}
        self.client_tokens = set()
        self.shadow_update_accepted_subscription_operation = None
        self.client_device_names_configuration_subscription_operation = None
        self.update_smart_light_device_list()

    def update_smart_light_device_list(self):
        # Update the device list from the component configuration.
        response = self.ipc_client.get_configuration(
            key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES])
        # Identify the difference between the configuration and the currently
 tracked devices.
        current_device_names = self.devices.keys()
        updated_device_names =
 response.value[CONFIGURATION_CLIENT_DEVICE_NAMES]
        added_device_names = set(updated_device_names) -
 set(current_device_names)
        removed_device_names = set(current_device_names) -
 set(updated_device_names)
        # Stop tracking any smart light devices that are no longer in the
 configuration.
        for name in removed_device_names:
            print('Removing %s from smart light device manager' % name)
            self.devices.pop(name)
        # Start tracking any new smart light devices that are in the
 configuration.
        for name in added_device_names:
            print('Adding %s to smart light device manager' % name)
            device = SmartLightDevice(name)
            device.reported_color = self.get_device_reported_color(device)
            self.devices[name] = device
            print('Current color for %s is %s' % (name,
 device.reported_color))

    def get_device_reported_color(self, smart_light_device):
```

```
        try:
            response = self.ipc_client.get_thing_shadow(
                thing_name=smart_light_device.name, shadow_name='')
            shadow = json.loads(str(response.payload, 'utf-8'))
            if 'reported' in shadow['state']:
                return shadow['state']['reported'].get(SHADOW_COLOR_PROPERTY)
            return None
        except ResourceNotFoundError:
            return None

    def request_device_color_change(self, smart_light_device, color):
        # Generate and track a client token for the request.
        client_token = str(uuid4())
        self.client_tokens.add(client_token)
        # Create a shadow payload, which must be a blob.
        payload_json = {
            'state': {
                'desired': {
                    SHADOW_COLOR_PROPERTY: color
                }
            },
            'clientToken': client_token
        }
        payload = bytes(json.dumps(payload_json), 'utf-8')
        self.ipc_client.update_thing_shadow(
            thing_name=smart_light_device.name, shadow_name='',
payload=payload)
        smart_light_device.desired_color = color

    def subscribe_to_shadow_update_accepted_events(self):
        if self.shadow_update_accepted_subscription_operation == None:
            # SubscribeToTopic returns a tuple with the response and the
operation.
            _, self.shadow_update_accepted_subscription_operation =
self.ipc_client.subscribe_to_topic(
                topic=SHADOW_UPDATE_TOPIC,
on_stream_event=self.on_shadow_update_accepted_event)
            print('Successfully subscribed to shadow update accepted topic')

    def close_shadow_update_accepted_subscription(self):
        if self.shadow_update_accepted_subscription_operation is not None:
            self.shadow_update_accepted_subscription_operation.close()

    def on_shadow_update_accepted_event(self, event):
```

```
        try:
            message = str(event.binary_message.message, 'utf-8')
            accepted_payload = json.loads(message)
            # Check for reported states from smart light devices and ignore
desired states from components.
            if 'reported' in accepted_payload['state']:
                # Process this update only if it uses a client token created by
this component.
                client_token = accepted_payload.get('clientToken')
                if client_token is not None and client_token in
self.client_tokens:
                    self.client_tokens.remove(client_token)
                    shadow_state = accepted_payload['state']['reported']
                    if SHADOW_COLOR_PROPERTY in shadow_state:
                        reported_color = shadow_state[SHADOW_COLOR_PROPERTY]
                        topic = event.binary_message.context.topic
                        client_device_name = topic.split('/')[2]
                        if client_device_name in self.devices:
                            # Set the reported color for the smart light
device.
                            self.devices[client_device_name].reported_color =
reported_color
                            print(
                                'Received shadow update confirmation from
client device: %s' % client_device_name)
                    else:
                        print("Shadow update doesn't specify color")
        except:
            traceback.print_exc()

    def subscribe_to_client_device_name_configuration_updates(self):
        if self.client_device_names_configuration_subscription_operation ==
None:
            # SubscribeToConfigurationUpdate returns a tuple with the response
and the operation.
            _, self.client_device_names_configuration_subscription_operation =
self.ipc_client.subscribe_to_configuration_update(
                key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES],
on_stream_event=self.on_client_device_names_configuration_update_event)
            print(
                'Successfully subscribed to configuration updates for smart
light device names')

    def close_client_device_names_configuration_subscription(self):
```

```
        if self.client_device_names_configuration_subscription_operation is not
 None:

 self.client_device_names_configuration_subscription_operation.close()

    def on_client_device_names_configuration_update_event(self, event):
        try:
            if CONFIGURATION_CLIENT_DEVICE_NAMES in
 event.configuration_update_event.key_path:
                print('Received configuration update for list of client
 devices')
                self.update_smart_light_device_list()
        except:
            traceback.print_exc()


def choose_random_color():
    return random.choice(COLORS)

def main():
    try:
        # Create an IPC client and a smart light device manager.
        ipc_client = GreengrassCoreIPCClientV2()
        smart_light_manager = SmartLightDeviceManager(ipc_client)
        smart_light_manager.subscribe_to_shadow_update_accepted_events()

 smart_light_manager.subscribe_to_client_device_name_configuration_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                # Set each smart light device to a random color at a regular
 interval.
                for device_name in smart_light_manager.devices:
                    device = smart_light_manager.devices[device_name]
                    desired_color = choose_random_color()
                    print('Chose random color (%s) for %s' %
                          (desired_color, device_name))
                    if desired_color == device.desired_color:
                        print('Desired color for %s is already %s' %
                              (device_name, desired_color))
                    elif desired_color == device.reported_color:
                        print('Reported color for %s is already %s' %
                              (device_name, desired_color))
                    else:
```

```
                                smart_light_manager.request_device_color_change(
                                    device, desired_color)
                                print('Requested color change for %s to %s' %
                                      (device_name, desired_color))
                        time.sleep(SET_COLOR_INTERVAL)
                except InterruptedError:
                    print('Application interrupted')
                smart_light_manager.close_shadow_update_accepted_subscription()

 smart_light_manager.close_client_device_names_configuration_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


if __name__ == '__main__':
    main()
```

这个 Python 应用程序执行以下操作：

- 读取组件的配置以获取要管理的智能轻型客户端设备列表。
- 使用 SubscribeToConfigurationUpdate IPC 操作订阅配置更新通知。每次组件的配置发生变化时，C AWS IoT Greengrass ore 软件都会发送通知。当组件收到配置更新通知时，它会更新其管理的智能轻型客户端设备列表。
- 获取每台智能灯光客户端设备的阴影以获取其初始颜色状态。
- 每隔 15 秒将每台智能灯客户端设备的颜色设置为随机颜色。该组件更新客户端设备的事物阴影以更改其颜色。此操作通过 MQTT 向客户端设备发送阴影增量事件。
- 使用 IPC 操作在本地发布/订阅接口上订阅影子更新已接受的SubscribeToTopic消息。该组件接收这些消息以跟踪每台智能灯客户端设备的颜色。当智能轻型客户端设备收到影子更新时，它会发送 MQTT 消息以确认已收到更新。MQTT 网桥将此消息中继到本地发布/订阅接口。

d. 使用 Greengrass CLI 部署该组件。部署此组件时，需要指定由其管理影子的客户端设备列表。`smartLightDeviceNames`将 *MyClientDevice1* 替换为客户端设备的事物名称。

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir recipes \
```

```
  --artifactDir artifacts \
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" \
  --update-config '{
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
        ]
      }
    }
  }'
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" ^
  --update-config '{"com.example.clientdevices.MySmartLightManager":
{"MERGE":{"smartLightDeviceNames":["MyClientDevice1"]}}}'
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" `
  --update-config '{
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
        ]
      }
    }
  }'
```

3. 查看组件日志，验证组件是否成功安装和运行。

Linux or Unix

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

PowerShell

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

该组件发送更改智能灯客户端设备颜色的请求。影子管理器接收请求并设置影子的desired状态。但是，智能灯客户端设备尚未运行，因此阴影的reported状态不会改变。该组件的日志包含以下消息。

```
2022-07-07T03:49:24.908Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
 for MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.912Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout.
 Requested color change for MyClientDevice1 to blue.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

您可以将日志源保持打开状态，以查看组件何时打印消息。

4. 下载并运行使用 Greengrass 发现功能并订阅设备影子更新的示例应用程序。在客户端设备上，执行以下操作：

   a. 在 python 版 AWS IoT Device SDK v2 中切换到示例文件夹。此示例应用程序使用示例文件夹中的命令行解析模块。

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

   b. 使用文本编辑器创建名为的 Python 脚本basic_discovery_shadow.py，其内容如下。此应用程序使用 Greengrass 发现和阴影来保持客户端设备和核心设备之间的属性同步。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

```
nano basic_discovery_shadow.py
```

将以下 Python 代码复制到文件中。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0.

from awscrt import io
from awscrt import mqtt
from awsiot import iotshadow
from awsiot.greengrass_discovery import DiscoveryClient
from awsiot import mqtt_connection_builder
from concurrent.futures import Future
import sys
import threading
import traceback
from uuid import uuid4

# Parse arguments
import utils.command_line_utils;
cmdUtils = utils.command_line_utils.CommandLineUtils("Basic Discovery -
 Greengrass discovery example with device shadows.")
cmdUtils.add_common_mqtt_commands()
cmdUtils.add_common_topic_message_commands()
cmdUtils.add_common_logging_commands()
cmdUtils.register_command("key", "<path>", "Path to your key in PEM format.",
 True, str)
cmdUtils.register_command("cert", "<path>", "Path to your client certificate in
 PEM format.", True, str)
cmdUtils.remove_command("endpoint")
cmdUtils.register_command("thing_name", "<str>", "The name assigned to your IoT
 Thing", required=True)
cmdUtils.register_command("region", "<str>", "The region to connect through.",
 required=True)
cmdUtils.register_command("shadow_property", "<str>", "The name of the shadow
 property you want to change (optional, default='color'", default="color")
# Needs to be called so the command utils parse the commands
cmdUtils.get_args()

# Using globals to simplify sample code
is_sample_done = threading.Event()
mqtt_connection = None
```

```
shadow_thing_name = cmdUtils.get_command_required("thing_name")
shadow_property = cmdUtils.get_command("shadow_property")

SHADOW_VALUE_DEFAULT = "off"

class LockedData:
    def __init__(self):
        self.lock = threading.Lock()
        self.shadow_value = None
        self.disconnect_called = False
        self.request_tokens = set()

locked_data = LockedData()


def on_connection_interupted(connection, error, **kwargs):
    print('connection interrupted with error {}'.format(error))


def on_connection_resumed(connection, return_code, session_present, **kwargs):
    print('connection resumed with return code {}, session present
 {}'.format(return_code, session_present))

# Try IoT endpoints until we find one that works
def try_iot_endpoints():
    for gg_group in discover_response.gg_groups:
        for gg_core in gg_group.cores:
            for connectivity_info in gg_core.connectivity:
                try:
                    print('Trying core {} at host {} port
 {}'.format(gg_core.thing_arn, connectivity_info.host_address,
 connectivity_info.port))
                    mqtt_connection = mqtt_connection_builder.mtls_from_path(
                        endpoint=connectivity_info.host_address,
                        port=connectivity_info.port,
                        cert_filepath=cmdUtils.get_command_required("cert"),
                        pri_key_filepath=cmdUtils.get_command_required("key"),

 ca_bytes=gg_group.certificate_authorities[0].encode('utf-8'),
                        on_connection_interrupted=on_connection_interupted,
                        on_connection_resumed=on_connection_resumed,
                        client_id=cmdUtils.get_command_required("thing_name"),
                        clean_session=False,
                        keep_alive_secs=30)
```

```
                    connect_future = mqtt_connection.connect()
                    connect_future.result()
                    print('Connected!')
                    return mqtt_connection

                except Exception as e:
                    print('Connection failed with exception {}'.format(e))
                    continue

    exit('All connection attempts failed')

# Function for gracefully quitting this sample
def exit(msg_or_exception):
    if isinstance(msg_or_exception, Exception):
        print("Exiting sample due to exception.")
        traceback.print_exception(msg_or_exception.__class__, msg_or_exception,
 sys.exc_info()[2])
    else:
        print("Exiting sample:", msg_or_exception)

    with locked_data.lock:
        if not locked_data.disconnect_called:
            print("Disconnecting...")
            locked_data.disconnect_called = True
            future = mqtt_connection.disconnect()
            future.add_done_callback(on_disconnected)

def on_disconnected(disconnect_future):
    # type: (Future) -> None
    print("Disconnected.")

    # Signal that sample is finished
    is_sample_done.set()

def on_get_shadow_accepted(response):
    # type: (iotshadow.GetShadowResponse) -> None
    try:
        with locked_data.lock:
            # check that this is a response to a request from this session
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return
```

```
            print("Finished getting initial shadow state.")
            if locked_data.shadow_value is not None:
                print("  Ignoring initial query because a delta event has
 already been received.")
                return

        if response.state:
            if response.state.delta:
                value = response.state.delta.get(shadow_property)
                if value:
                    print("  Shadow contains delta value '{}'.".format(value))
                    change_shadow_value(value)
                    return

            if response.state.reported:
                value = response.state.reported.get(shadow_property)
                if value:
                    print("  Shadow contains reported value
 '{}'.".format(value))

 set_local_value_due_to_initial_query(response.state.reported[shadow_property])
                    return

        print("  Shadow document lacks '{}' property. Setting
 defaults...".format(shadow_property))
        change_shadow_value(SHADOW_VALUE_DEFAULT)
        return

    except Exception as e:
        exit(e)

def on_get_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        if error.code == 404:
            print("Thing has no shadow document. Creating with defaults...")
```

```
            change_shadow_value(SHADOW_VALUE_DEFAULT)
        else:
            exit("Get request was rejected. code:{} message:'{}'".format(
                error.code, error.message))

    except Exception as e:
        exit(e)

def on_shadow_delta_updated(delta):
    # type: (iotshadow.ShadowDeltaUpdatedEvent) -> None
    try:
        print("Received shadow delta event.")
        if delta.state and (shadow_property in delta.state):
            value = delta.state[shadow_property]
            if value is None:
                print("  Delta reports that '{}' was deleted. Resetting
 defaults...".format(shadow_property))
                change_shadow_value(SHADOW_VALUE_DEFAULT)
                return
            else:
                print("  Delta reports that desired value is '{}'. Changing
 local value...".format(value))
                if (delta.client_token is not None):
                    print ("  ClientToken is: " + delta.client_token)
                change_shadow_value(value, delta.client_token)
        else:
            print("  Delta did not report a change in
 '{}'".format(shadow_property))

    except Exception as e:
        exit(e)

def on_publish_update_shadow(future):
    #type: (Future) -> None
    try:
        future.result()
        print("Update request published.")
    except Exception as e:
        print("Failed to publish update request.")
        exit(e)

def on_update_shadow_accepted(response):
    # type: (iotshadow.UpdateShadowResponse) -> None
    try:
```

```
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

        try:
            if response.state.reported != None:
                if shadow_property in response.state.reported:
                    print("Finished updating reported shadow value to
 '{}'.".format(response.state.reported[shadow_property])) # type: ignore
                else:
                    print ("Could not find shadow property with name:
 '{}'.".format(shadow_property)) # type: ignore
            else:
                print("Shadow states cleared.") # when the shadow states are
 cleared, reported and desired are set to None
        except:
            exit("Updated shadow is missing the target property")

    except Exception as e:
        exit(e)

def on_update_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        exit("Update request was rejected. code:{} message:'{}'".format(
            error.code, error.message))

    except Exception as e:
        exit(e)

def set_local_value_due_to_initial_query(reported_value):
    with locked_data.lock:
        locked_data.shadow_value = reported_value
```

```
def change_shadow_value(value, token=None):
    with locked_data.lock:
        if locked_data.shadow_value == value:
            print("Local value is already '{}'.".format(value))
            return

        print("Changed local shadow value to '{}'.".format(value))
        locked_data.shadow_value = value

        print("Updating reported shadow value to '{}'...".format(value))

        reuse_token = token is not None
        # use a unique token so we can correlate this "request" message to
        # any "response" messages received on the /accepted and /rejected
 topics
        if not reuse_token:
            token = str(uuid4())

        # if the value is "clear shadow" then send a UpdateShadowRequest with
 None
        # for both reported and desired to clear the shadow document
 completely.
        if value == "clear_shadow":
            tmp_state = iotshadow.ShadowState(reported=None, desired=None,
 reported_is_nullable=True, desired_is_nullable=True)
            request = iotshadow.UpdateShadowRequest(
                thing_name=shadow_thing_name,
                state=tmp_state,
                client_token=token,
            )
        # Otherwise, send a normal update request
        else:
            # if the value is "none" then set it to a Python none object to
            # clear the individual shadow property
            if value == "none":
                value = None

            request = iotshadow.UpdateShadowRequest(
            thing_name=shadow_thing_name,
            state=iotshadow.ShadowState(
                reported={ shadow_property: value }
                ),
                client_token=token,
            )
```

```
        future = shadow_client.publish_update_shadow(request,
 mqtt.QoS.AT_LEAST_ONCE)

        if not reuse_token:
            locked_data.request_tokens.add(token)

        future.add_done_callback(on_publish_update_shadow)


if __name__ == '__main__':
    tls_options =
 io.TlsContextOptions.create_client_with_mtls_from_path(cmdUtils.get_command_required("
 cmdUtils.get_command_required("key"))
    if cmdUtils.get_command(cmdUtils.m_cmd_ca_file):
        tls_options.override_default_trust_store_from_path(None,
 cmdUtils.get_command(cmdUtils.m_cmd_ca_file))
    tls_context = io.ClientTlsContext(tls_options)

    socket_options = io.SocketOptions()

    print('Performing greengrass discovery...')
    discovery_client =
 DiscoveryClient(io.ClientBootstrap.get_or_create_static_default(),
 socket_options, tls_context, cmdUtils.get_command_required("region"))
    resp_future =
 discovery_client.discover(cmdUtils.get_command_required("thing_name"))
    discover_response = resp_future.result()

    print(discover_response)
    if cmdUtils.get_command("print_discover_resp_only"):
        exit(0)

    mqtt_connection = try_iot_endpoints()
    shadow_client = iotshadow.IotShadowClient(mqtt_connection)

    try:
        # Subscribe to necessary topics.
        # Note that is **is** important to wait for "accepted/rejected"
 subscriptions
        # to succeed before publishing the corresponding "request".
        print("Subscribing to Update responses...")
        update_accepted_subscribed_future, _ =
 shadow_client.subscribe_to_update_shadow_accepted(
```

```
request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_accepted)

        update_rejected_subscribed_future, _ =
shadow_client.subscribe_to_update_shadow_rejected(

request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_rejected)

        # Wait for subscriptions to succeed
        update_accepted_subscribed_future.result()
        update_rejected_subscribed_future.result()

        print("Subscribing to Get responses...")
        get_accepted_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_accepted(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_accepted)

        get_rejected_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_rejected(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_rejected)

        # Wait for subscriptions to succeed
        get_accepted_subscribed_future.result()
        get_rejected_subscribed_future.result()

        print("Subscribing to Delta events...")
        delta_subscribed_future, _ =
shadow_client.subscribe_to_shadow_delta_updated_events(

request=iotshadow.ShadowDeltaUpdatedSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_shadow_delta_updated)

        # Wait for subscription to succeed
```

```
        delta_subscribed_future.result()

        # The rest of the sample runs asynchronously.

        # Issue request for shadow's current state.
        # The response will be received by the on_get_accepted() callback
        print("Requesting current shadow state...")

        with locked_data.lock:
            # use a unique token so we can correlate this "request" message to
            # any "response" messages received on the /accepted and /rejected
topics
            token = str(uuid4())

            publish_get_future = shadow_client.publish_get_shadow(

request=iotshadow.GetShadowRequest(thing_name=shadow_thing_name,
client_token=token),
                qos=mqtt.QoS.AT_LEAST_ONCE)

            locked_data.request_tokens.add(token)

        # Ensure that publish succeeds
        publish_get_future.result()

    except Exception as e:
        exit(e)

    # Wait for the sample to finish (user types 'quit', or an error occurs)
    is_sample_done.wait()
```

这个 Python 应用程序执行以下操作：

- 使用 Greengrass 发现功能来发现并连接到核心设备。
- 从核心设备请求影子文档以获取属性的初始状态。
- 订阅 shadow delta 事件，当属性的值与其`desired``reported`值不同时，核心设备会发送这些事件。当应用程序收到 shadow delta 事件时，它会更改属性的值并向核心设备发送更新以将新值设置为其`reported`值。

该应用程序结合了 Greengrass 的发现和 v2 中的阴影样本。AWS IoT Device SDK

c. 运行示例应用程序。此应用程序需要指定客户端设备事物名称、要使用的影子属性以及用于验证和保护连接的证书的参数。

   - 将 *MyClientDevice1* 替换为客户端设备的事物名称。
   - 将 *~/certs/ AmazonRoot ca1.pem #*换为客户端设备上亚马逊根 CA 证书的路径。
   - 将 *~/certs/device.pem.crt ########*上设备证书的路径。
   - 将 *~/certs/private.pem.key ###########*文件的路径。
   - 将 *us-east-1* 替换为您的客户端设备和核心设备运行的AWS区域。

```
python3 basic_discovery_shadow.py \
  --thing_name MyClientDevice1 \
  --shadow_property color \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbosity Warn
```

示例应用程序订阅影子主题并等待接收来自核心设备的阴影增量事件。如果输出表明应用程序接收并响应 shadow delta 事件，则客户端设备可以成功地与核心设备上的影子进行交互。

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGG
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----
\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Subscribing to Update responses...
Subscribing to Get responses...
Subscribing to Delta events...
Requesting current shadow state...
Received shadow delta event.
  Delta reports that desired value is 'purple'. Changing local value...
```

```
  ClientToken is: 3dce4d3f-e336-41ac-aa4f-7882725f0033
Changed local shadow value to 'purple'.
Updating reported shadow value to 'purple'...
Update request published.
```

如果应用程序改为输出错误，请参阅 Greengrass 发现问题疑难解答。

您还可以查看核心设备上的 Greengrass 日志，以验证客户端设备是否成功连接和发送消息。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

5. 再次查看组件日志，以验证该组件是否收到来自智能轻型客户端设备的阴影更新确认。

Linux or Unix

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

PowerShell

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

该组件记录消息以确认智能轻型客户端设备已更改其颜色。

```
2022-07-07T03:49:24.908Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
 for MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.912Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout.
 Requested color change for MyClientDevice1 to blue.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.959Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Received
 shadow update confirmation from client device: MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

客户端设备的影子在核心设备和客户端设备之间同步。但是，核心设备不会将客户端设备的影子与同步AWS IoT Core。例如，您可以将影子与同步，AWS IoT Core以查看或修改队列中所有设备的状态。有关如何配置阴影管理器组件以与之同步阴影的更多信息AWS IoT Core，请参阅将本地设备阴影与同步 AWS IoT Core。

你已经完成了本教程。客户端设备连接到核心设备，向和 Greengrass 组件发送 MQTT 消息，AWS IoT Core并接收来自核心设备的影子更新。有关本教程所涵盖主题的更多信息，请参阅以下内容：

- 关联客户端设备
- 管理核心设备端点
- 测试客户端设备通信
- Greengrass 发现 RESTful API
- 在客户端设备之间中继 MQTT 消息和 AWS IoT Core
- 在组件中与客户端设备交互
- 与设备阴影互动
- 与客户端设备影子进行交互并进行同步

教程： SageMaker 边缘管理器入门

Important

SageMaker Edge Manager 将于 2024 年 4 月 26 日停产。有关继续将模型部署到边缘设备的更多信息，请参阅 Edg SageMaker e Manager 生命周期终止。

Amazon SageMaker Edge Manager 是一款在边缘设备上运行的软件代理。 SageMaker Edge Manager 为边缘设备提供模型管理，因此您可以直接在 Greengrass 核心设备上打包和使用 Amazon SageMaker Neo 编译的模型。通过使用 SageMaker Edge Manager，您还可以对核心设备的模型输入和输出数据进行采样，并将这些数据发送到AWS Cloud进行监控和分析。有关 SageMaker Edge Manager 如何在 Greengrass 核心设备上工作的更多信息，请参阅。在 Green SageMaker grass 核心设备上使用亚马逊 Edge Manager

本教程向您展示了如何在现有核心设备上使用 SageMaker 边缘管理器以及AWS提供的示例组件。这些示例组件使用 SageMaker Edge Manager 组件作为依赖项来部署 Edge Manager 代理，并使用使用 Neo 编译的预训练模型执行推理。 SageMaker 有关 SageMaker 边缘管理器代理的更多信息，请参阅《亚马逊 SageMaker 开发者指南》中的SageMaker 边缘管理器。

要在现有 Green SageMaker grass 核心设备上设置和使用 Edge Manager 代理AWS，请提供可用于创建以下示例推理和建模组件的示例代码。

- 图像分类
  - `com.greengrass.SageMakerEdgeManager.ImageClassification`
  - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- 物体检测
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection`
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

本教程向您展示如何部署示例组件和 SageMaker Edge Manager 代理。

主题

- 先决条件
- 在 Edge Manager 中设置你的 Greengrass 核心设备 SageMaker
- 创建示例组件
- 运行样本图像分类推理

先决条件

要完成本教程，您必须满足以下先决条件：

- 在亚马逊 Linux 2、基于 Debian 的 Linux 平台（x86_64 或 Armv8）或 Windows（x86_64）上运行的 Greengrass 核心设备。如果没有，请参阅教程：AWS IoT Greengrass V2 入门。
- 安装在核心设备上的 Pyth@@ on 3.6 或更高版本，包括`pip`适用于你的 Python 版本。
- 安装在核心设备上的 OpenGL API GLX 运行时 `libgl1-mesa-glx` ()。
- 具有管理员权限的 AWS Identity and Access Management (IAM) 用户。
- 符合以下要求且支持互联网的 Windows、Mac 或类似 Unix 的开发计算机：
  - 已安装 Python 3.6 或更高版本。

  - AWS CLI使用您的 IAM 管理员用户证书安装和配置。有关更多信息，请参阅安装AWS CLI和配置 AWS CLI。
- 以下 S3 存储桶AWS 账户与AWS 区域您的 Greengrass 核心设备相同：
  - 一个 S3 存储桶，用于存储示例推理和模型组件中包含的项目。本教程使用 *DOC-EXAMPLE-BUCKET1* 来引用此存储桶。
  - 与 SageMaker 边缘设备队列关联的 S3 存储桶。 SageMaker Edge Manager 需要一个 S3 存储桶来创建边缘设备队列，并存储在设备上运行推理的示例数据。本教程使用 *DOC-EXAMPLE-BUCKET2* 来引用此存储桶。

  有关创建 S3 存储桶的信息，请参阅 Amazon S3 入门。
- Greengrass 设备角色配置如下：
  - 一种允许`credentials.iot.amazonaws.com`和代`sagemaker.amazonaws.com`入角色的信任关系，如以下 IAM 策略示例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

  - I AmazonSageMakerEdgeDeviceFleetPolicyAM 托管策略。
  - I AmazonSageMakerFullAccessAM 托管策略。
  - 包含您的组件工件的 S3 存储桶的`s3:GetObject`操作，如以下 IAM 策略示例所示。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
  {
    "Action": [
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
    ],
    "Effect": "Allow"
  }
  ]
}
```

在 Edge Manager 中设置你的 Greengrass 核心设备 SageMaker

边缘管理器中的 SageMaker 边缘设备队列是按逻辑分组的设备的集合。要将 SageMaker Edge Manager 与一起使用AWS IoT Greengrass，必须创建一个边缘设备队列，该队列使用的AWS IoT角色别名与部署边缘管理器代理的 Greengrass 核心设备相同。 SageMaker 然后，您必须将核心设备注册为该队列的一部分。

主题

- 创建边缘设备队列
- 注册你的 Greengrass 核心设备

创建边缘设备队列

创建边缘设备队列（控制台）

1. 在亚马逊 SageMaker 控制台中，选择边缘管理器，然后选择边缘设备队列。
2. 在设备队列页面上，选择创建设备队列。
3. 在 “设备队列属性” 下，执行以下操作：

   - 在设备队列名称中，输入设备队列的名称。
   - 对于 IAM 角色，请输入您在设置 Greengrass 核心设备时指定的AWS IoT角色别名的亚马逊资源名称 (ARN)。
   - 禁用 “创建 IAM 角色别名” 开关。

4. 请选择 Next（下一步）。
5. 在输出配置下，对于 S3 存储桶 UR I，输入要与设备队列关联的 S3 存储桶的 URI。
6. 选择提交。

注册你的 Greengrass 核心设备

将您的 Greengrass 核心设备注册为边缘设备（主机）

1. 在亚马逊 SageMaker 控制台中，选择边缘管理器，然后选择边缘设备。
2. 在设备页面上，选择注册设备。
3. 在 “设备属性” 下的 “设备队列名称” 中，输入您创建的设备队列的名称，然后选择 “下一步”。
4. 请选择 Next（下一步）。
5. 在 “设备来源” 下，在 “设备名称” 中，输入 Greengrass 核心设备AWS IoT的事物名称。
6. 选择提交。

创建示例组件

为了帮助您开始使用 SageMaker Edge Manager 组件，AWS提供了一个AWS Cloud用于创建示例推理和模型组件的 Python GitHub 脚本，并将它们上传到。在开发计算机上完成以下步骤。

创建示例组件

1. 将AWS IoT Greengrass组件示例存储库下载 GitHub 到您的开发计算机上。
2. 导航到已下载/machine-learning/sagemaker-edge-manager的文件夹。

```
cd download-directory/machine-learning/sagemaker-edge-manager
```

3. 运行以下命令来创建示例组件并将其上传到AWS Cloud。

```
python3 create_components.py -r region -b DOC-EXAMPLE-BUCKET
```

将*##*替换为您创建 Greengrass 核心设备AWS 区域的位置，*# DOC-EXAMPLE-BUCKET1 ##*为用于存储组件工件的 S3 存储桶的名称。

> Note
>
> 默认情况下，该脚本会为图像分类和物体检测推断创建示例组件。要仅为特定类型的推理创建组件，请指定`-i ImageClassification | ObjectDetection`参数。

现在已在中创建了与 SageMaker Edge Manager 一起使用的示例推理和模型组件。AWS 账户要在AWS IoT Greengrass控制台中查看示例组件，请选择 “组件”，然后在 “我的组件” 下搜索以下组件：

- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

运行样本图像分类推理

要使用AWS提供的示例组件和 SageMaker Edge Manager 代理运行图像分类推理，必须将这些组件部署到核心设备。部署这些组件会下载 SageMaker Neo 编译的预训练的 Resnet-50 模型，并在您的设备上安装 SageMaker Edge Manager 代理。边 SageMaker 缘管理器代理加载模型并发布有关该`gg/sageMakerEdgeManager/image-classification`主题的推理结果。要查看这些推理结果，请使用AWS IoT控制台中的 AWS IoT MQTT 客户端订阅此主题。

主题

- 订阅通知主题
- 部署示例组件
- 查看推理结果

订阅通知主题

在此步骤中，您将在AWS IoT控制台中配置 AWS IoT MQTT 客户端，以查看示例推理组件发布的 MQTT 消息。默认情况下，该组件会发布有关该`gg/sageMakerEdgeManager/image-classification`主题的推理结果。在将组件部署到 Greengrass 核心设备之前，请订阅此主题，以查看组件首次运行时的推理结果。

订阅默认通知主题

1. 在AWS IoT控制台导航菜单中，选择测试，MQTT 测试客户端。
2. 在 “订阅主题” 下的 “主题名称” 框中输入**gg/sageMakerEdgeManager/image-classification**。
3. 选择订阅。

部署示例组件

在此步骤中，您将配置以下组件并将其部署到核心设备：

- `aws.greengrass.SageMakerEdgeManager`
- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

部署组件（控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择部署，然后为要修改的目标设备选择部署。
2. 在部署页面上，选择修订，然后选择修订部署。
3. 在指定目标页面，选择下一步。
4. 在选择用户页面上，执行以下操作：

   a. 在 “我的组件” 下，选择以下组件：

      - `com.greengrass.SageMakerEdgeManager.ImageClassification`
      - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

   b. 在 “公共组件” 下，关闭 “仅显示选定的组件” 开关，然后选择该`aws.greengrass.SageMakerEdgeManager`组件。

   c. 请选择 Next（下一步）。
5. 在配置组件页面上，选择`aws.greengrass.SageMakerEdgeManager`组件并执行以下操作。

   a. 选择配置组件。

   b. 在配置更新下的要合并的配置中，输入以下配置。

      ```
      {
      ```

```
    "DeviceFleetName": "device-fleet-name",
    "BucketName": "DOC-EXAMPLE-BUCKET"
}
```

*device-fleet-name*替换为您创建的边缘设备队列的名称，将 *DOC-EXAMPLE-B* UCKET 替换为与您的设备队列关联的 S3 存储桶的名称。

c. 选择确认，然后选择下一步。

6. 在配置高级设置页面上，保留默认配置设置，然后选择下一步。
7. 在 “审阅” 页面上，选择 “部署”

部署组件 (AWS CLI)

1. 在开发计算机上，创建一个`deployment.json`文件来定义 SageMaker Edge Manager 组件的部署配置。此文件应类似于以下示例。

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.SageMakerEdgeManager": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
        "merge": "{\"DeviceFleetName\":\"device-fleet-name\",\"BucketName\":\"DOC-
EXAMPLE-BUCKET2\"}"
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification.Model": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
  }
}
```

- 在 `targetArn` 字段中，按以下格式将 *targetArn* 替换为部署目标的事物或事物组的 Amazon 资源名称 (ARN)：

- 事物：arn:aws:iot:*region*:*account-id*:thing/*thingName*
- 事物组：arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*
- 在merge字段中，*device-fleet-name*替换为您创建的边缘设备队列的名称。然后，将 *DOC-EXAMPLE-BUCKET2* 替换为与您的设备队列关联的 S3 存储桶的名称。
- 将每个组件的组件版本替换为最新的可用版本。

2. 运行以下命令以在设备上部署组件：

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

完成部署可能需要数分钟。在下一步中，检查组件日志，以验证部署是否成功完成并查看推理结果。

查看推理结果

部署组件后，您可以在 Greengrass 核心设备的组件日志和控制台的 MQTT 客户端中AWS IoT查看推理结果。AWS IoT要订阅组件发布推理结果的主题，请参阅订阅通知主题。

- AWS IoTMQTT 客户端-要查看推理组件在默认通知主题上发布的结果，请完成以下步骤：

  1. 在AWS IoT控制台导航菜单中，选择测试，MQTT 测试客户端。
  2. 在 “订阅” 下，选择**gg/sageMakerEdgeManager/image-classification**。

- 组件日志-要在组件日志中查看推理结果，请在 Greengrass 核心设备上运行以下命令。

```
sudo tail -f /greengrass/v2/logs/
com.greengrass.SageMakerEdgeManager.ImageClassification.log
```

如果您在组件日志或 MQTT 客户端中看不到推理结果，则表示部署失败或未到达核心设备。如果您的核心设备未连接到互联网或没有运行该组件的正确权限，则可能会发生这种情况。在核心设备上运行以下命令以查看AWS IoT Greengrass核心软件日志文件。此文件包含来自 Greengrass 核心设备部署服务的日志。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

有关更多信息，请参阅 对机器学习推理进行故障排除。

教程：使用 TensorFlow Lite 执行样本图像分类推断

本教程向您展示如何使用 TensorFlow Lite 图像分类推理组件在 Greengrass 核心设备上执行样本图像分类推理。该组件包括以下组件依赖关系：

- TensorFlow 精简版图像分类模型存储组件
- TensorFlow 精简版运行时组件

部署此组件时，它会下载预先训练的 MobileNet v1 模型并安装 TensorFlow Lite 运行时及其依赖项。该组件发布有关该`ml/tflite/image-classification`主题的推理结果。要查看这些推理结果，请使用AWS IoT控制台中的 AWS IoT MQTT 客户端订阅此主题。

在本教程中，您将部署示例推理组件，以便对提供的AWS IoT Greengrass示例图像执行图像分类。完成本教程后，您可以完成本教程教程：使用 TensorFlow Lite 对来自相机的图像执行样本图像分类推断，该教程向您展示了如何修改示例推理组件，以便在 Greengrass 核心设备上对来自本地摄像机的图像进行图像分类。

有关在 Greengrass 设备上进行机器学习的更多信息，请参阅。执行机器学习推理

主题

- 先决条件
- 步骤 1：订阅默认通知主题
- 步骤 2：部署 TensorFlow Lite 图像分类组件
- 步骤 3：查看推理结果
- 后续步骤

先决条件

要完成本教程，您需要：

- 一款 Linux Greengrass 核心设备。如果没有，请参阅教程：AWS IoT Greengrass V2 入门。核心设备必须满足以下要求：
  - 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。

- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

       ```
       sudo raspi-config
       ```

    2. 选择接口选项。
    3. 选择旧版相机以启用旧版相机堆栈。
    4. 重启 Raspberry Pi。

步骤 1：订阅默认通知主题

在此步骤中，您将在AWS IoT控制台中配置 AWS IoT MQTT 客户端，以查看 TensorFlow 精简版图像分类组件发布的 MQTT 消息。默认情况下，该组件会发布有关该`ml/tflite/image-classification`主题的推理结果。在将组件部署到 Greengrass 核心设备之前，请订阅此主题，以查看组件首次运行时的推理结果。

订阅默认通知主题

1. 在AWS IoT控制台导航菜单中，选择测试，MQTT 测试客户端。
2. 在 “订阅主题” 下的 “主题名称” 框中输入**`ml/tflite/image-classification`**。
3. 选择订阅。

步骤 2：部署 TensorFlow Lite 图像分类组件

在此步骤中，您将 TensorFlow 精简版图像分类组件部署到您的核心设备：

部署 TensorFlow Lite 图像分类组件（控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择组件。
2. 在组件页面的公有组件选项卡上，选择 `aws.greengrass.TensorFlowLiteImageClassification`。
3. 在 aws.greengrass.TensorFlowLiteImageClassification 页面上，选择部署。
4. 从添加到部署中，选择以下选项之一：
   a. 要将此组件合并到目标设备上的现有部署，请选择添加到现有部署，然后选择要修改的部署。
   b. 要在目标设备上创建新部署，请选择创建新部署。如果您的设备上已有部署，选择此步骤将替换现有部署。
5. 在指定目标页面中，执行以下操作：
   a. 在部署信息下，输入或修改部署的友好名称。
   b. 在部署目标下，选择部署目标，然后选择下一步。如果您正在修改现有部署，则无法更改部署目标。
6. 在 “选择组件” 页面的 “公共组件” 下，确认已选择该`aws.greengrass.TensorFlowLiteImageClassification`组件，然后选择 “下一步”。
7. 在配置组件页面上，保留默认配置设置，然后选择下一步。
8. 在配置高级设置页面上，保留默认配置设置，然后选择下一步。
9. 在 “审阅” 页面上，选择 “部署”

部署 TensorFlow Lite 图像分类组件 (AWS CLI)

1. 创建一个`deployment.json`文件来定义 TensorFlow Lite 图像分类组件的部署配置。此文件应如下所示：

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
```

```
        "configurationUpdate": {
        }
      }
    }
}
```

- 在 targetArn 字段中，按以下格式将 *targetArn* 替换为部署目标的事物或事物组的 Amazon 资源名称 (ARN)：
  - 事物：arn:aws:iot:*region*:*account-id*:thing/*thingName*
  - 事物组：arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*
- 本教程使用组件版本 2.1.0。在aws.greengrass.TensorFlowLiteObjectDetection组件对象中，替换 *2.1.0* 以使用不同版本的 TensorFlow Lite 对象检测组件。

2. 运行以下命令在设备上部署 TensorFlow Lite 图像分类组件：

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

完成部署可能需要数分钟。在下一步中，检查组件日志，以验证部署是否成功完成并查看推理结果。

步骤 3：查看推理结果

部署组件后，您可以在 Greengrass 核心设备的组件日志和控制台的 MQTT 客户端中AWS IoT查看推理结果。AWS IoT要订阅组件发布推理结果的主题，请参阅步骤 1：订阅默认通知主题。

- AWS IoTMQTT 客户端-要查看推理组件在默认通知主题上发布的结果，请完成以下步骤：

  1. 在AWS IoT控制台导航菜单中，选择测试，MQTT 测试客户端。
  2. 在 “订阅” 下，选择**ml/tflite/image-classification**。

     您应该会看到类似于以下示例的消息。

```
{
  "timestamp": "2021-01-01 00:00:00.000000",
  "inference-type": "image-classification",
  "inference-description": "Top 5 predictions with score 0.3 or above ",
  "inference-results": [
    {
      "Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
 concolor",
```

```
        "Score": "0.5882352941176471"
      },
      {
        "Label": "Persian cat",
        "Score": "0.5882352941176471"
      },
      {
        "Label": "tiger cat",
        "Score": "0.5882352941176471"
      },
      {
        "Label": "dalmatian, coach dog, carriage dog",
        "Score": "0.5607843137254902"
      },
      {
        "Label": "malamute, malemute, Alaskan malamute",
        "Score": "0.5450980392156862"
      }
    ]
}
```

- 组件日志-要在组件日志中查看推理结果，请在 Greengrass 核心设备上运行以下命令。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

您应该看到与以下示例类似的结果。

```
2021-01-01 00:00:00.000000 [INFO] (Copier)
 aws.greengrass.TensorFlowLiteImageClassification: stdout. Publishing results to the
 IoT core....
 {scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
 serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}

2021-01-01 00:00:00.000000 [INFO] (Copier)
 aws.greengrass.TensorFlowLiteImageClassification: stdout. {"timestamp":
 "2021-01-01 00:00:00.000000", "inference-type": "image-classification", "inference-
description": "Top 5 predictions with score 0.3 or above ", "inference-results":
 [{"Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
 concolor", "Score": "0.5882352941176471"}, {"Label": "Persian cat", "Score":
 "0.5882352941176471"}, {"Label": "tiger cat", "Score": "0.5882352941176471"},
 {"Label": "dalmatian, coach dog, carriage dog", "Score": "0.5607843137254902"},
 {"Label": "malamute, malemute, Alaskan malamute", "Score": "0.5450980392156862"}]}.
```

```
{scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}
```

如果您在组件日志或 MQTT 客户端中看不到推理结果，则表示部署失败或未到达核心设备。如果您的核心设备未连接到互联网或没有运行该组件的正确权限，则可能会发生这种情况。在核心设备上运行以下命令以查看AWS IoT Greengrass核心软件日志文件。此文件包含来自 Greengrass 核心设备部署服务的日志。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

有关更多信息，请参阅对机器学习推理进行故障排除。

后续步骤

如果您的 Greengrass 核心设备支持相机接口，则可以教程：使用 TensorFlow Lite 对来自相机的图像执行样本图像分类推断完成，其中向您展示了如何修改示例推理组件以对来自相机的图像进行图像分类。

要进一步探索 TensorFlow Lite 图像分类推理示例组件的配置，请尝试以下操作：

- 修改`InferenceInterval`配置参数以更改推理代码的运行频率。
- 修改推理组件`ImageDirectory`配置中的`ImageName`和配置参数，以指定用于推理的自定义图像。

有关自定义公共组件配置或创建自定义机器学习组件的信息，请参阅自定义您的机器学习组件。

教程：使用 TensorFlow Lite 对来自相机的图像执行样本图像分类推断

本教程向您展示如何使用 TensorFlow Lite 图像分类推理组件在 Greengrass 核心设备上对来自本地摄像机的图像执行样本图像分类推断。该组件包括以下组件依赖关系：

- TensorFlow 精简版图像分类模型存储组件
- TensorFlow 精简版运行时组件

Note

本教程访问了 Raspberry Pi 或 NVIDIA Jetson Nano 设备的摄像头模块，但AWS IoT Greengrass支持 armv7L、Armv8 或 x86_64 平台上的其他设备。要为其他设备设置摄像头，请查阅设备的相关文档。

有关在 Greengrass 设备上进行机器学习的更多信息，请参阅。执行机器学习推理

主题

- 先决条件
- 步骤 1：在设备上配置摄像头模块
- 第 2 步：验证您对默认通知主题的订阅
- 步骤 3：修改 TensorFlow Lite 图像分类组件配置并进行部署
- 步骤 4：查看推理结果
- 后续步骤

先决条件

要完成本教程，必须先完成本教程教程：使用 TensorFlow Lite 执行样本图像分类推断。

您需要以下项目：

- 一款带有摄像头接口的 Linux Greengrass 核心设备。本教程将在以下支持的设备之一上访问摄像头模块：
  - Raspberry Pi 运行树莓派操作系统（以前称为 Raspbian）
  - NVIDIA 杰森纳米

  有关设置 Greengrass 核心设备的信息，请参阅。教程：AWS IoT Greengrass V2 入门

  核心设备必须满足以下要求：
  - 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
  - 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令来安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上进行升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

       ```
       sudo raspi-config
       ```

    2. 选择 “接口选项”。
    3. 选择 “旧版相机” 以启用旧版相机堆栈。
    4. 重启 Raspberry Pi。

- 对于 Raspberry Pi 或 NVIDIA Jetson Nano 设备，树莓派摄像头模块 V2-800 万像素，1080 p。要了解如何设置摄像机，请参阅 Raspberry Pi 文档中的连接摄像机。

步骤 1：在设备上配置摄像头模块

在此步骤中，您将为设备安装并启用摄像头模块。在设备上运行以下命令。

Raspberry Pi (Armv7l)

1. 安装摄像头模块的`picamera`接口。运行以下命令安装本教程所需的摄像头模块和其他 Python 库。

   ```
   sudo apt-get install -y python3-picamera
   ```

2. 验证是否成功安装了 Picamera。

   ```
   sudo -u ggc_user bash -c 'python3 -c "import picamera"'
   ```

如果输出未包含错误，则表示验证成功。

> **Note**
>
> 如果设备上安装的 Python 可执行文件是python3.7，请使用python3.7而不是python3作为本教程中的命令。确保 pip 安装映射到正确的 python3.7 或 python3 版本以避免依赖项错误。

3. 重新启动设备。

```
sudo reboot
```

4. 打开 Raspberry Pi 配置工具。

```
sudo raspi-config
```

5. 使用箭头键打开接口选项并启用摄像机接口。如果出现提示，请允许设备重新启动。
6. 运行以下命令来测试摄像机设置。

```
raspistill -v -o test.jpg
```

这将在 Raspberry Pi 上打开一个预览窗口，将名为 test.jpg 的图片保存到您的当前目录，并在 Raspberry Pi 终端中显示有关摄像机的信息。

7. 运行以下命令创建符号链接，使推理组件能够从运行时组件创建的虚拟环境中访问您的摄像机。

```
sudo ln -s /usr/lib/python3/dist-packages/picamera "MLRootPath/
greengrass_ml_tflite_venv/lib/python3.7/site-packages"
```

本教程中 *ML RootPath* 的默认值为*/greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml。此位置中的greengrass_ml_tflite_venv文件夹是在中首次部署推理组件时创建的。教程：使用 TensorFlow Lite 执行样本图像分类推断

Jetson Nano (Armv8)

1. 运行以下命令来测试摄像机设置。

```
gst-launch-1.0 nvarguscamerasrc num-buffers=1 ! "video/x-raw(memory:NVMM),
 width=1920, height=1080, format=NV12, framerate=30/1" ! nvjpegenc ! filesink
 location=test.jpg
```

这会捕获名为当前目录的图像并将其保存test.jpg到您的当前目录。

2. （可选）重新启动设备。如果您在上一步中运行gst-launch命令时遇到问题，则重新启动设备可能会解决这些问题。

```
sudo reboot
```

> **Note**
>
> 对于 Armv8 (aarch64) 设备，例如 Jetson Nano，您无需创建符号链接即可使推理组件能够从运行时组件创建的虚拟环境访问摄像机。

第 2 步：验证您对默认通知主题的订阅

在中教程：使用 TensorFlow Lite 执行样本图像分类推断，您在AWS IoT控制台中将 AWS IoT MQTT 客户端配置为观看 TensorFlow Lite 图像分类组件针对该主题发布的 MQTT 消息。ml/tflite/image-classification在AWS IoT控制台中，验证此订阅是否存在。如果没有，请在将组件部署步骤 1：订阅默认通知主题到 Greengrass 核心设备之前，按照中的步骤订阅此主题。

步骤 3：修改 TensorFlow Lite 图像分类组件配置并进行部署

在此步骤中，您将配置 TensorFlow Lite 图像分类组件并将其部署到您的核心设备：

配置和部署 TensorFlow Lite 图像分类组件（控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择组件。
2. 在组件页面的公有组件选项卡上，选择 aws.greengrass.TensorFlowLiteImageClassification。
3. 在 aws.greengrass.TensorFlowLiteImageClassification 页面上，选择部署。
4. 从 “添加到部署” 中，选择以下选项之一：

    a. 要将此组件合并到目标设备上的现有部署，请选择添加到现有部署，然后选择要修改的部署。

   b. 要在目标设备上创建新部署，请选择创建新部署。如果您的设备上已有部署，选择此步骤将替换现有部署。

5. 在指定目标页面中，执行以下操作：

   a. 在部署信息下，输入或修改部署的友好名称。

   b. 在部署目标下，选择部署目标，然后选择下一步。如果您正在修改现有部署，则无法更改部署目标。

6. 在 “选择组件” 页面的 “公共组件” 下，确认已选择该`aws.greengrass.TensorFlowLiteImageClassification`组件，然后选择 “下一步”。

7. 在 “配置组件” 页面上，执行以下操作：

   a. 选择推理组件，然后选择配置组件。

   b. 在 “配置更新” 下，在 “要合并的配置” 框中输入以下配置更新。

   ```
   {
     "InferenceInterval": "60",
     "UseCamera": "true"
   }
   ```

   通过此配置更新，该组件将访问您设备上的摄像头模块，并对摄像机拍摄的图像进行推理。推理代码每 60 秒运行一次。

   c. 选择确认，然后选择下一步。

8. 在配置高级设置页面上，保留默认配置设置，然后选择下一步。

9. 在 “审阅” 页面上，选择 “部署”

配置和部署 TensorFlow Lite 图像分类组件 (AWS CLI)

1. 创建一个`deployment.json`文件来定义 TensorFlow Lite 图像分类组件的部署配置。此文件应如下所示：

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
        "InferenceInterval": "60",
```

```
          "UseCamera": "true"
        }
      }
    }
}
```

- 在 targetArn 字段中，按以下格式将 *targetArn* 替换为部署目标的事物或事物组的 Amazon 资源名称 (ARN)：
  - 事物：arn:aws:iot:*region*:*account-id*:thing/*thingName*
  - 事物组：arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*
- 本教程使用组件版本 2.1.0。在aws.greengrass.TensorFlowLiteImageClassification组件对象中，替换 *2.1.0* 以使用不同版本的 TensorFlow Lite 图像分类组件。

通过此配置更新，该组件将访问您设备上的摄像头模块，并对摄像机拍摄的图像进行推理。推理代码每 60 秒运行一次。替换以下值

2. 运行以下命令在设备上部署 TensorFlow Lite 图像分类组件：

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

完成部署可能需要数分钟。在下一步中，检查组件日志，以验证部署是否成功完成并查看推理结果。

步骤 4：查看推理结果

部署组件后，您可以在 Greengrass 核心设备的组件日志和控制台的 MQTT 客户端中AWS IoT查看推理结果。AWS IoT要订阅组件发布推理结果的主题，请参阅第 2 步：验证您对默认通知主题的订阅。

- AWS IoTMQTT 客户端-要查看推理组件在默认通知主题上发布的结果，请完成以下步骤：
  1. 在AWS IoT控制台导航菜单中，选择测试，MQTT 测试客户端。
  2. 在 “订阅” 下，选择**ml/tflite/image-classification**。
- 组件日志-要在组件日志中查看推理结果，请在 Greengrass 核心设备上运行以下命令。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

如果您在组件日志或 MQTT 客户端中看不到推理结果，则表示部署失败或未到达核心设备。如果您的核心设备未连接到互联网或没有运行该组件所需的权限，则可能会发生这种情况。在您的核心设备上运行以下命令以查看AWS IoT Greengrass核心软件日志文件。此文件包含来自 Greengrass 核心设备部署服务的日志。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

有关更多信息，请参阅对机器学习推理进行故障排除。

后续步骤

本教程向您展示如何使用 TensorFlow 精简版图像分类组件以及自定义配置选项对相机拍摄的图像进行样本图像分类。

有关自定义公共组件配置或创建自定义机器学习组件的更多信息，请参阅自定义您的机器学习组件。

组件

AWS IoT Greengrass组件是您部署到 Greengrass 核心设备的软件模块。组件可以代表应用程序、运行时安装程序、库或您将在设备上运行的任何代码。您可以定义依赖于其他组件的组件。例如，您可以定义一个安装 Python 的组件，然后将该组件定义为运行 Python 应用程序的组件的依赖项。当您将组件部署到设备队列时，Greengrass 仅部署您的设备所需的软件模块。

主题

- AWS-提供的组件
- 发布商支持的组件
- 社区组件
- AWS IoT Greengrass开发工具
- 开发AWS IoT Greengrass组件
- 将AWS IoT Greengrass组件部署到设备

AWS-提供的组件

AWS IoT Greengrass 提供并维护可部署到设备上的预建组件。这些组件包括功能（例如流管理器）、 AWS IoT Greengrass V1 连接器（例如 CloudWatch 指标）和本地开发工具（例如 AWS IoT Greengrass CLI）。您可以将这些组件部署到设备上以实现其独立功能，也可以将它们用作自定义 Greengrass 组件中的依赖项。

> **Note**
>
> AWS提供的几个组件依赖于 Greengrass 核的特定次要版本。由于这种依赖关系，当你将 Greengrass nucleus 更新到新的次要版本时，你需要更新这些组件。有关每个组件所依赖的原子核的特定版本的信息，请参阅相应的组件主题。有关更新原子核的更多信息，请参见更新 AWS IoT Greengrass核心软件 (OTA)。

当组件的组件类型同时为泛型和 Lambda 时，该组件的当前版本为泛型类型，而该组件的先前版本为 Lambda 类型。

组件	描述	组件类型	支持的操作系统	开源
Greengrass 核	AWS IoT Greengrass 核心软件的核心。使用此组件在核心设备上配置和更新软件。	核	Linux、Windows	是
客户端设备身份验证	使本地 IoT 设备（称为客户端设备）能够连接到核心设备。	插件	Linux、Windows	是
CloudWatch 指标	向 Amazon 发布自定义指标 CloudWatch。	通用、Lambda	Linux、Windows	是
AWS IoT Device Defender	通知管理员有关 Greengrass 核心设备状态的变化，以识别异常行为。	通用、Lambda	Linux、Windows	是
磁盘后台处理程序	为从 Greengrass 核心设备后台处理的消息启用永久存储选项。 AWS IoT Core此组件会将这些出站消息存储在磁盘上。	插件	Linux、Windows	是

组件	描述	组件类型	支持的操作系统	开源
Docker 应用程序管理器	AWS IoT Greengrass 允许从 Docker Hub 和亚马逊弹性容器注册表 (Amazon ECR) Container Registry 下载 Docker 镜像。	通用	Linux、Windows	否
Kinesis Video Streams 的边缘连接器	读取来自本地摄像机的视频源，将直播发布到 Kinesis Video Streams，并使用在 Grafana 仪表板中显示直播。 AWS IoT TwinMaker	通用	Linux	否
Greengrass CLI	提供命令行界面，可用于创建本地部署并与 Greengrass 核心设备及其组件进行交互。	插件	Linux、Windows	是

组件	描述	组件类型	支持的操作系统	开源
IP 探测器	向报告 MQTT 代理连接信息 AWS IoT Greengrass，以便客户户端设备可以发现如何连接。	插件	Linux、Windows	是
Firehose	通过 Amazon Data Firehose 将数据发布到中的目的地。AWS Cloud	Lambda	Linux	否
Lambda 启动器	处理 Lambda 函数的进程和环境配置。	通用	Linux	否
Lambda 管理器	处理 Lambda 函数的进程间通信和扩展。	插件	Linux	否
Lambda 运行时	为每个 Lambda 运行时提供构件。	通用	Linux	否
旧版订阅路由器	管理在 V1 上 AWS IoT Greengrass 运行的 Lambda 函数的订阅。	通用	Linux	否

组件	描述	组件类型	支持的操作系统	开源
本地调试控制台	提供本地控制台，可用于调试和管理 Greengrass 核心设备及其组件。	插件	Linux、Windows	是
日志管理器	在 Greengrass 核心设备上收集和上传日志。	插件	Linux、Windows	是
机器学习组件	提供机器学习模型和示例推理代码，可用于在 Greengrass 核心设备上执行机器学习推理。	请参阅 机器学习组件。		
modbus-RTU 协议适配器	轮询来自本地 Modbus RTU 设备的信息。	Lambda	Linux	否
Nucleus 遥测发射器	将从核心收集的系统运行状况遥测数据发布到本地主题或 AWS IoT Core MQTT 主题。	插件	Linux、Windows	是

组件	描述	组件类型	支持的操作系统	开源
MQTT 网桥	在客户端设备、本地 AWS IoT Greengrass 发布/订阅和之间中继 MQTT 消息。AWS IoT Core	插件	Linux、Windows	是
MQTT 3.1.1 经纪商 (Moquette)	运行处理客户端设备和核心设备之间消息的 MQTT 3.1.1 代理。	插件	Linux、Windows	是
MQTT 5 经纪商 (EMQX)	运行处理客户端设备和核心设备之间消息的 MQTT 5 代理。	通用	Linux、Windows	否
PKCS #11 提供商	允许 Greengrass 组件访问您安全存储在硬件安全模块 (HSM) 中的私钥和证书。	插件	Linux	是

组件	描述	组件类型	支持的操作系统	开源
秘密经理	部署来自机密的 AWS Secrets Manager 机密，以便您可以在 Greengrass 核心设备的自定义组件中安全地使用密码（例如密码）。	插件	Linux、Windows	是
安全隧道	启用 AWS IoT 安全的隧道连接，可用于与受限防火墙后面的 Greengrass 核心设备建立双向通信。	通用	Linux	否
影子经理	启用与核心设备上的阴影交互。它管理影子文档存储以及本地卷影状态与 Dev AWS IoT ice Shadow 服务的同步。	插件	Linux、Windows	是

组件	描述	组件类型	支持的操作系统	开源
Amazon SNS	向 Amazon SNS 主题发布消息。	Lambda	Linux	否
流管理器	将大量数据从本地源流式传输到。 AWS Cloud	通用	Linux、Windows	否
Systems Manager 代理	使用管理核心设备 AWS Systems Manager，使您能够修补设备、运行命令等。	通用	Linux	否
代币兑换服务	提供可用于与 AWS 服务交互的 AWS 凭证。	通用	Linux、Windows	否
物联网 SiteWise OPC-UA 采集器	从 OPC-UA 服务器收集数据。	通用	Linux、Windows	否
物联网 SiteWise OPC-UA 数据源模拟器	运行生成样本数据的本地 OPC-UA 服务器。	通用	Linux、Windows	否
物联网 SiteWise 发行商	将数据发布到 AWS 云端。	通用	Linux、Windows	否

组件	描述	组件类型	支持的操作系统	开源
物联网 SiteWise 处理器	处理 Greengrass 核心设备上的数据。	通用	Linux、Windows	否

Greengrass 核

Greengrass nucleus 组件 `aws.greengrass.Nucleus` () 是必备组件，也是在设备上运行 AWS IoT Greengrass 核心软件的最低要求。您可以将此组件配置为远程自定义和更新您的 AWS IoT Greengrass Core 软件。部署此组件可在核心设备上配置代理、设备角色和 AWS IoT 事物配置等设置。

> **Important**
>
> 当 nucleus 组件的版本发生变化时，或者当你更改某些配置参数时， AWS IoT Greengrass Core 软件（包括 nucleus 和设备上的所有其他组件）会重新启动以应用更改。
>
> 部署组件时， AWS IoT Greengrass 会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则 AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

主题

- 版本
- 操作系统
- 要求
- 依赖项
- 下载和安装
- 配置

- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x
- 2.8.x
- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

有关更多信息，请参阅 支持的平台。

要求

设备必须满足某些要求才能安装和运行 Greengrass nucleus 和 Core 软件。 AWS IoT Greengrass 有关更多信息，请参阅 设备要求。

支持 Greengrass 核心组件在 VPC 中运行。要在 VPC 中部署此组件，需要满足以下条件。

- Greengrass 核心组件必须连接到、 AWS IoT data凭证 AWS IoT 和 Amazon S3。

依赖项

Greengrass 核心不包含任何组件依赖关系。但是，一些 AWS提供的组件将核作为依赖项包含在内。有关更多信息，请参阅 AWS-提供的组件。

有关组件依赖关系的更多信息，请参阅组件配方参考。

下载和安装

你可以下载安装程序，在你的设备上设置 Greengrass nucleus 组件。此安装程序将您的设备设置为 Greengrass 核心设备。您可以执行两种类型的安装：一种是为您创建所需 AWS 资源的快速安装，另一种是手动安装，您可以自己创建 AWS 资源。有关更多信息，请参阅 安装 AWS IoT Greengrass Core 软件。

你也可以按照教程安装 Greengrass 核心并探索 Greengrass 组件的开发。有关更多信息，请参阅 教程：AWS IoT Greengrass V2 入门。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。某些参数要求 AWS IoT Greengrass Core软件重新启动才能生效。有关为何以及如何配置此组件的更多信息，请参阅配置 AWS IoT Greengrass 核心软件。

`iotRoleAlias`

指向令牌交换 IAM 角色的角色别名。 AWS IoT AWS IoT 凭证提供者扮演此角色是为了允许 Greengrass 核心设备与服务进行交互。 AWS 有关更多信息，请参阅 授权核心设备与AWS服务。

当你使用`--provision true`选项运行 AWS IoT Greengrass Core 软件时，该软件会预置一个角色别名并在 nucleus 组件中设置其值。

`interpolateComponentConfiguration`

（可选）您可以启用 Greengrass nucleus，以便在组件配置中插入组件配方变量并合并配置更新。我们建议您将此选项设置为，`true`以便核心设备可以运行在其配置中使用配方变量的 Greengrass 组件。

此功能适用于该组件的 2.6.0 及更高版本。

默认：false

networkProxy

（可选）用于所有连接的网络代理。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。

Important

当您部署对此配置参数的更改时， AWS IoT Greengrass Core 软件会重新启动以使更改生效。

该对象包含以下信息：

noProxyAddresses

（可选）以逗号分隔的 IP 地址或主机名列表，这些地址或主机名不受代理限制。

proxy

要连接的代理。该对象包含以下信息：

url

格式为代理服务器的 URL scheme://userinfo@host:port。

- scheme— 方案，必须是http或https。

Important

Greengrass 核心设备必须运行 Greengrass nucleus v2.5.0 或更高版本才能使用 HTTPS 代理。

如果您配置 HTTPS 代理，则必须将代理服务器 CA 证书添加到核心设备的 Amazon 根 CA 证书中。有关更多信息，请参阅 使核心设备能够信任 HTTPS 代理。

- userinfo—（可选）用户名和密码信息。如果您在中指定此信息url，Greengrass 核心设备将忽略和字段。username password
- host— 代理服务器的主机名或 IP 地址。
- port—（可选）端口号。如果您未指定端口，则 Greengrass 核心设备将使用以下默认值：

- http— 80
- https— 443

username

（可选）对代理服务器进行身份验证的用户名。

password

（可选）对代理服务器进行身份验证的密码。

mqtt

（可选）Greengrass 核心设备的 MQTT 配置。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。

> **Important**
> 当您部署对此配置参数的更改时， AWS IoT Greengrass Core 软件会重新启动以使更改生效。

该对象包含以下信息：

port

（可选）用于 MQTT 连接的端口。

默认：8883

keepAliveTimeoutMs

（可选）客户端为保持 MQTT 连接活动而发送的每PING条消息之间的间隔时间（以毫秒为单位）。此值必须大于pingTimeoutMs。

默认值：60000（60 秒）

pingTimeoutMs

（可选）客户端等待从服务器接收PINGACK消息的时间（以毫秒为单位）。如果等待时间超过超时时间，核心设备将关闭并重新打开 MQTT 连接。此值必须小于keepAliveTimeoutMs。

默认值：30000（30 秒）

`operationTimeoutMs`

（可选）客户端等待 MQTT 操作（例如或）完成的时间（以毫秒为单位）。CONNECT PUBLISH此选项不适用于 MQTT PING 或保持活动状态的消息。

默认值：30000（30 秒）

`maxInFlightPublishes`

（可选）可以同时传输的未确认的 MQTT QoS 1 消息的最大数量。

此功能适用于该组件的 v2.1.0 及更高版本。

默认：5

有效范围：最大值为 100

`maxMessageSizeInBytes`

（可选）MQTT 消息的最大大小。如果一条消息超过这个大小，Greengrass 核心会以错误拒绝该消息。

此功能适用于该组件的 v2.1.0 及更高版本。

默认值：131072(128 KB)

有效范围：最大值为 2621440 (2.5 MB)

`maxPublishRetry`

（可选）重试发布失败的消息的最大次数。您可以指定-1重试次数不限。

此功能适用于该组件的 v2.1.0 及更高版本。

默认：100

`spooler`

（可选）Greengrass 核心设备的 MQTT 后台处理程序配置。该对象包含以下信息：

`storageType`

用于存储消息的存储类型。如果设置storageType为Disk，则pluginName可以配置。您可指定 Memory 或 Disk。

此功能适用于 Greengrass nucleus 组件的 v2.11.0 及更高版本。

> **Important**
>
> 如果 MQTT 后台处理程序设置storageType为，Disk并且您想将 Greengrass nucleus 从 2.11.x 版本降级到早期版本，则必须将配置更改回到。MemoryGreengrass nucleus 版本 2.10.x 及更早版本支持的唯一配置是。storageType Memory不遵循此指导可能会导致后台处理程序中断。这将导致您的 Greengrass 核心设备无法向发送 MQTT 消息。 AWS Cloud

默认：`Memory`

`pluginName`

（可选）插件组件名称。只有设置为时，才会使用storageType此组件Disk。此选项默认为aws.greengrass.DiskSpooler并将使用 GreenGr 磁盘后台处理程序 ass 提供的选项。

此功能适用于 Greengrass nucleus 组件的 v2.11.0 及更高版本。

默认："aws.greengrass.DiskSpooler"

`maxSizeInBytes`

（可选）核心设备在内存中存储未处理的 MQTT 消息的最大缓存大小。如果缓存已满，则新消息将被拒绝。

默认值：2621440(2.5 MB)

`keepQos0WhenOffline`

（可选）您可以对核心设备离线时收到的 MQTT QoS 0 消息进行后台处理。如果将此选项设置为true，则核心设备会缓冲离线时无法发送的 QoS 0 消息。如果将此选项设置为false，则核心设备会丢弃这些消息。除非线轴已满，否则核心设备总是假脱机 QoS 1 消息。

默认：false

`version`

（可选）MQTT 的版本。您可指定 mqtt3 或 mqtt5。

此功能适用于 Greengrass nucleus 组件的 v2.10.0 及更高版本。

默认：`mqtt5`

`receiveMaximum`

（可选）代理可以发送的未确认的 QoS1 数据包的最大数量。

此功能适用于 Greengrass nucleus 组件的 v2.10.0 及更高版本。

默认：`100`

`sessionExpirySeconds`

（可选）您可以从 IoT Core 请求持续会话的时间（以秒为单位）。默认值为支持的最长时间 AWS IoT Core。

此功能适用于 Greengrass nucleus 组件的 v2.10.0 及更高版本。

默认：`604800 (7 days)`

`minimumReconnectDelaySeconds`

（可选）重新连接行为的选项。MQTT 重新连接的最短时间（以秒为单位）。

此功能适用于 Greengrass nucleus 组件的 v2.10.0 及更高版本。

默认：`1`

`maximumReconnectDelaySeconds`

（可选）重新连接行为的选项。MQTT 重新连接的最大时间（以秒为单位）。

此功能适用于 Greengrass nucleus 组件的 v2.10.0 及更高版本。

默认：`120`

`minimumConnectedTimeBeforeRetryResetSeconds`

（可选）重新连接行为的选项。在将重试延迟重置回最小值之前，连接必须处于活动状态的时间（以秒为单位）。

此功能适用于 Greengrass nucleus 组件的 v2.10.0 及更高版本。

默认：`30`

`jvmOptions`

（可选）用于运行 AWS IoT Greengrass 核心软件的 JVM 选项。有关运行 C AWS IoT Greengrass ore 软件时推荐的 JVM 选项的信息，请参阅使用 JVM 选项控制内存分配。

Important

当您部署对此配置参数的更改时， AWS IoT Greengrass Core 软件会重新启动以使更改生效。

`iotDataEndpoint`

您的 AWS IoT 数据端点 AWS 账户。

当你使用该`--provision true`选项运行 AWS IoT Greengrass Core 软件时，该软件会从 AWS IoT 中获取您的数据和凭据端点，并将其设置在 nucleus 组件中。

`iotCredEndpoint`

您的 AWS IoT 凭证终端节点 AWS 账户。

当你使用该`--provision true`选项运行 AWS IoT Greengrass Core 软件时，该软件会从 AWS IoT 中获取您的数据和凭据端点，并将其设置在 nucleus 组件中。

`greengrassDataPlaneEndpoint`

此功能在该组件的 2.7.0 及更高版本中可用。

有关更多信息，请参阅 使用由私有 CA 签名的设备证书。

`greengrassDataPlanePort`

此功能在该组件的 v2.0.4 及更高版本中可用。

（可选）用于数据平面连接的端口。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。

Important

您必须指定设备可以进行出站连接的端口。如果您指定被屏蔽的端口，则设备将无法连接 AWS IoT Greengrass 以接收部署。

从以下选项中进行选择：

- 443
- 8443

默认：8443

`awsRegion`

AWS 区域 要使用的。

`runWithDefault`

用于运行组件的系统用户。

Important

当您部署对此配置参数的更改时， AWS IoT Greengrass Core 软件会重新启动以使更改生效。

该对象包含以下信息：

`posixUser`

核心设备用于运行通用组件和 Lambda 组件的系统用户的名称或 ID，以及可选的系统组。使用以下格式指定由半角冒号（:）分隔的用户和组：`user:group`。组是可选的。如果您未指定群组，则 AWS IoT Greengrass Core 软件将使用该用户的主群组。举例来说，可以指定 `ggc_user` 或 `ggc_user:ggc_group`。有关更多信息，请参阅 配置运行组件的用户。

当你使用`--component-default-user` *`ggc_user`*:*`ggc_group`*选项运行 AWS IoT Greengrass Core 软件安装程序时，软件会在 nucleus 组件中设置此参数。

`windowsUser`

此功能在该组件的 v2.5.0 及更高版本中可用。

用于在 Windows 核心设备上运行此组件的 Windows 用户的名称。用户必须存在于每台 Windows 核心设备上，其用户名和密码必须存储在 LocalSystem 账户的凭据管理器实例中。有关更多信息，请参阅 配置运行组件的用户。

当你使用`--component-default-user` *`ggc_user`*选项运行 AWS IoT Greengrass Core 软件安装程序时，软件会在 nucleus 组件中设置此参数。

`systemResourceLimits`

此功能在该组件的 v2.4.0 及更高版本中可用。 AWS IoT Greengrass 目前不支持在 Windows 核心设备上使用此功能。

默认情况下，适用于通用和非容器化 Lambda 组件进程的系统资源限制。在创建部署时，您可以覆盖各个组件的系统资源限制。有关更多信息，请参阅 为组件配置系统资源限制。

该对象包含以下信息：

`cpus`

每个组件的进程可以在核心设备上使用的最大 CPU 时间。核心设备的总 CPU 时间等于 CPU 核心的设备数量。例如，在具有 4 个 CPU 内核的核心设备上，您可以将此值设置为，2将每个组件的进程使用率限制为每个 CPU 内核的 50%。在具有 1 个 CPU 内核的设备上，您可以将此值设置为，`0.25`将每个组件的进程的 CPU 使用率限制在 25% 以内。如果将此值设置为大于 CPU 内核数的数字，则 AWS IoT Greengrass Core 软件不会限制组件的 CPU 使用率。

`memory`

每个组件的进程可以在核心设备上使用的最大 RAM 量（以千字节为单位）。

`s3EndpointType`

（可选）S3 端点类型。此参数仅对美国东部（弗吉尼亚北部）(`us-east-1`) 区域生效。从任何其他区域设置此参数将被忽略。从以下选项中进行选择：

- `REGIONAL`— S3 客户端和预签名 URL 使用区域终端节点。
- `GLOBAL`— S3 客户端和预签名 URL 使用传统端点。

默认：`GLOBAL`

`logging`

（可选）核心设备的日志配置。有关如何配置和使用 Greengrass 日志的更多信息，请参阅。监控 AWS IoT Greengrass日志

该对象包含以下信息：

`level`

（可选）要输出的最低日志消息级别。

从以下日志级别中进行选择，此处按级别顺序列出：

- DEBUG
- INFO
- WARN
- ERROR

默认：INFO

format

（可选）日志的数据格式。从以下选项中进行选择：

- TEXT— 如果您想以文本形式查看日志，请选择此选项。
- JSON— 如果您想使用 Greengrass CLI 日志命令查看日志或以编程方式与日志交互，请选择此选项。

默认：TEXT

outputType

（可选）日志的输出类型。从以下选项中进行选择：

- FILE— C AWS IoT Greengrass ore 软件将日志输出到您在中指定的目录中的文件outputDirectory。
- CONSOLE— C AWS IoT Greengrass ore 软件将日志打印到stdout。选择此选项可在核心设备打印日志时查看日志。

默认：FILE

fileSizeKB

（可选）每个日志文件的最大大小（以千字节为单位）。日志文件超过此最大文件大小后，AWS IoT Greengrass Core 软件会创建一个新的日志文件。

此参数仅在您FILE为指定时适用outputType。

默认：1024

totalLogsSizeKB

（可选）每个组件（包括 Greengrass 核）的最大日志文件总大小（以千字节为单位）。Greengrass nucleus 的日志文件还包括来自插件组件的日志。当组件的日志文件总大小超过此最大大小后，AWS IoT Greengrass Core 软件会删除该组件最旧的日志文件。

此参数等同于日志管理器组件的磁盘空间限制参数 (diskSpaceLimit)，您可以为 Greengrass 核（系统）和每个组件指定该参数。 AWS IoT Greengrass Core 软件使用两个值中的最小值作为 Greengrass 核和每个组件的最大总日志大小。

此参数仅在您FILE为指定时适用outputType。

默认：10240

outputDirectory

（可选）日志文件的输出目录。

此参数仅在您FILE为指定时适用outputType。

默认：*/greengrass/v2*/logs， AWS IoT Greengrass 根文件夹在*/greengrass/v2*哪里。

fleetstatus

此参数在该组件的 v2.1.0 及更高版本中可用。

（可选）核心设备的队列状态配置。

该对象包含以下信息：

periodicStatusPublishIntervalSeconds

（可选）核心设备向发布设备状态的间隔时间（以秒为单位） AWS Cloud。

最短：86400（24 小时）

默认值：86400（24 小时）

telemetry

（可选）核心设备的系统运行状况遥测配置。有关遥测指标以及如何对遥测数据采取行动的更多信息，请参阅。从AWS IoT Greengrass核心设备收集系统运行状况遥测数据

该对象包含以下信息：

enabled

（可选）您可以启用或禁用遥测。

默认：true

`periodicAggregateMetricsIntervalSeconds`

（可选）核心设备汇总指标的时间间隔（以秒为单位）。

如果将此值设置为低于支持的最小值，则 nucleus 将改用默认值。

最低：3600

默认：3600

`periodicPublishMetricsIntervalSeconds`

（可选）核心设备向发布遥测指标的间隔时间（以秒为单位）。 AWS Cloud

如果将此值设置为低于支持的最小值，则 nucleus 将改用默认值。

最低：86400

默认：86400

`deploymentPollingFrequencySeconds`

（可选）轮询部署通知的时间段（以秒为单位）。

默认：15

`componentStoreMaxSizeBytes`

（可选）组件存储在磁盘上的最大大小，其中包括组件配方和工件。

默认值：10000000000(10 GB)

`platformOverride`

（可选）标识核心设备平台的属性字典。使用它来定义自定义平台属性，组件配方可以使用这些属性来识别组件的正确生命周期和工件。例如，您可以定义硬件功能属性以仅部署最少的构件集以供组件运行。有关更多信息，请参阅组件配方中的清单平台参数。

您也可以使用此参数来覆盖核心设备的`architecture`和平台属性。`os`

`httpClient`

此参数在该组件的 v2.5.0 及更高版本中可用。

（可选）核心设备的 HTTP 客户端配置。这些配置选项适用于该组件发出的所有 HTTP 请求。如果核心设备在较慢的网络上运行，则可以增加这些超时持续时间以防止 HTTP 请求超时。

该对象包含以下信息：

`connectionTimeoutMs`

（可选）连接请求超时之前等待连接打开的时间（以毫秒为单位）。

默认：2000（2 秒）

`socketTimeoutMs`

（可选）在连接超时之前等待通过打开的连接传输数据的时间（以毫秒为单位）。

默认值：30000（30 秒）

Example 示例：配置合并更新

```
{
  "iotRoleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "http://my-proxy-server:1100",
      "username": "Mary_Major",
      "password": "pass@word1357"
    }
  },
  "mqtt": {
    "port": 443
  },
  "greengrassDataPlanePort": 443,
  "jvmOptions": "-Xmx64m",
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.12.4	错误修复和改进 • 修复了在某些 Linux 设备上启动时核心进入死锁状态的问题。
2.12.3	**Warning** 此版本不再可用。此版本的改进将在此组件的更高版本中提供。 错误修复和改进 • 修复了在原子核重启后和组件恢复期间，原子核无法报告正确的组件状态的问题。 • 常规错误修复和性能改进。

版本	更改
2.12.2	错误修复和改进 • 修复了旧日志未正确清理的问题。 • 常规错误修复和性能改进。
2.12.1	错误修复和改进 • 修复了核心可能会重复订阅部署主题的 MQTT，从而导致额外的日志记录和 MQTT 发布的问题。
2.12.0	新功能 • 使您能够在回滚部署中运行引导生命周期步骤。
2.11.3	错误修复和改进 • 修复了 nucleus 中的一个问题，即当组件的依赖关系失败时，它可能会不正确地启动组件。 新功能 • 添加可配置的 s3 端点类型。
2.11.2	错误修复和改进 • 修复了 nucleus MQTT 5 客户端中的一个问题，即在使用大量（> 50）订阅时，它可能会显示为离线。 • 为 docker 拨号 TCP 失败添加了重试功能。
2.11.1	错误修复和改进 • 修复了在引导任务失败且部署元数据文件损坏时，nucleus 无法启动的问题。 • 修复了部署状态更新中未报告按需 Lambda 组件的问题。 • 增加了对重复授权策略 ID 的支持。
2.11.0	新功能 • 允许您取消本地部署。 • 使您能够为本地部署配置故障处理策略。 • 添加了对磁盘后台处理程序插件的支持。

版本	更改
2.10.3	错误修复和改进 • 修复了使用 PKCS #11 提供程序时 Greengrass 不订阅部署通知的问题。
2.10.2	错误修复和改进 • 允许对组件生命周期进行不区分大小写的解析。 • 修复了未正确重新创建环境 PATH 变量的问题。 • 修复了组件的代理 URI 编码，包括带有特殊字符的用户名的流管理器。
2.10.1	错误修复和改进 • 修复了可能导致某些 ARMv8 处理器（包括 Jetson Nano）在启动时崩溃的问题。 • Greengrass 不再关闭组件的标准，这会将行为恢复到 2.10.0 之前的行为
2.10.0	新功能 • 添加对空正则表达式的`interpolateComponentConfiguration` 支持。Greengrass 现在可以从根配置对象中进行插值。 • 添加了对 MQTT5 的支持。 • 添加了无需扫描即可快速加载插件组件的机制。 • 让 Greengrass 能够通过删除未使用的 Docker 镜像来节省磁盘空间。 错误修复和改进 • 修复了回滚会使部署中的某些配置值保持不变的问题。 • 修复了 Greengrass nucleus 验证自定义非凭据和数据端点中的域序列的问题。 AWS AWS • 更新多组依赖关系解析以通过 AWS Cloud 协商重新解析所有组依赖关系，而不是锁定到活动版本。此更新还删除了部署错误代码`INSTALLED_COMPONENT_NOT_FOUND` 。 • 更新 Greengrass 核心，使其在本地已存在 Docker 镜像时跳过下载这些镜像。 • 更新 Greengrass 核心，以便在超时到期之前重新启动组件安装步骤。 • 其他小修复和改进。

版本	更改
2.9.6	错误修复和改进 • 修复了 Greengrass 部署失败并显示错误 LAUNCH_DIRECTORY_CORTED 以及随后的设备重启无法启动 Greengrass 的问题。当您在部署需要 Greengrass 重启的多个事物组之间移动 Greengrass 设备时，可能会发生此错误。
2.9.5	新功能 • 增加了对 Greengrass nucleus 软件签名验证的支持。 错误修复和改进 • 修复了本地配方元数据区域与 Greengrass nucleus 启动区域不匹配时部署失败的问题。当这种情况发生时，Greengrass 核现在会与云重新协商。 • 修复了 MQTT 消息后台处理程序已满且从不删除消息的问题。 • 其他小修复和改进。
2.9.4	错误修复和改进 • 在丢弃 QOS 0 消息之前检查是否存在空消息。 • 如果作业状态详细信息值超过 1024 个字符的限制，则将其截断。 • 更新 Windows 的引导脚本以正确读取 Greengrass 根路径（如果该路径包含空格）。 • 更新订阅， AWS IoT Core 以便在未发送订阅响应时丢弃客户端消息。 • 确保 nucleus 在主配置文件损坏或丢失时从备份文件加载其配置。
2.9.3	错误修复和改进 • 确保 MQTT 客户端 ID 不重复。 • 增加了更强大的文件读取和写入功能，以避免损坏并从损坏中恢复。 • 重试 docker 镜像拉取特定的网络相关错误。 • 添加了 MQTT 连接`noProxyAddresses` 选项。

版本	更改
2.9.2	错误修复和改进 • 修复了配置interpolateComponentConfiguration 不适用于正在进行的部署的问题。 • 使用 OSHI 列出所有子进程。
2.9.1	错误修复和改进 • 添加了在部署移除插件组件时 Greengrass 会重新启动的修复程序。
2.9.0	新功能 • 添加了创建子部署的功能，这些子部署可以重试使用较小的设备子集进行部署。此功能提供了一种更有效的方法来测试和解决不成功的部署。 错误修复和改进 • 改进了对没有useraddgroupadd、和的系统的支持usermod。 • 其他小修复和改进。
2.8.1	错误修复和改进 • 修复了 Greengrass API 错误导致部署错误代码未正确生成的问题。 • 修复了在部署期间组件达到某个状态时，舰队状态更新发送的ERRORED信息不准确的问题。 • 修复了 Greengrass 现有订阅超过 50 个时部署无法完成的问题。

版本	更改
2.8.0	新功能 • 在将组件部署到核心设备时出现问题时，更新 Greengrass nucleus 以报告部署运行状况响应，其中包括详细的错误代码。有关更多信息，请参阅详细的部署错误代码。 • 更新 Greengrass nucleus 以报告组件健康状态响应，其中包括组件进入或状态时的详细错误代码。BROKEN ERRORED有关更多信息，请参阅 详细的组件状态码。 • 扩展状态消息字段以改善设备的云可用性信息。 • 提高了舰队状态服务的稳健性。 错误修复和改进 • 允许损坏的组件在配置更改时重新安装。 • 修复了在 bootstrap 部署期间重启 nucleus 会导致部署失败的问题。 • 修复了 Windows 中根路径包含空格时安装失败的问题。 • 修复了在部署期间关闭的组件使用新版本的关闭脚本的问题。 • 各种关机改进。 • 其他小修复和改进。

版本	更改
2.7.0	新功能 • 更新 Greengrass 核心，以便在核心设备应用本地部署时向 AWS IoT Greengrass 云端发送状态更新。 • 添加对由自定义证书颁发机构 (CA) 签名的客户端证书的支持，CA 未在其中注册 AWS IoT。要使用此功能，可以将新的`greengrassDataPlaneEndpoint` 配置选项设置为`iotdata`。有关更多信息，请参阅 使用由私有 CA 签名的设备证书。 错误修复和改进 • 修复了在某些情况下，当原子核停止或重新启动时，Greengrass 核会回滚部署的问题。现在，在原子核重启后，原子核会恢复部署。 • 当您指定将软件设置为系统服务时，更新 Greengrass 安装程序以遵守`--start`该参数。 • 更新行为SubscribeToComponentUpdates以在核心更新组件的事件中设置部署 ID。 • 其他小修复和改进。

版本	更改
2.6.0	新功能 • 当您订阅本地发布/订阅主题时，添加对 MQTT 通配符的支持。有关更多信息，请参阅 发布/订阅本地消息 和 SubscribeToTopic。 • 在组件配置中添加对配方变量（*component_dependency_name* :configuration: *json_pointer* 配方变量除外）的支持。在配方中定义组件或在部署DefaultConfiguration 中配置组件时，可以使用这些配方变量。要启用此功能，请将interpolateComponentConfiguration配置选项设置为true。有关更多信息，请参阅 食谱变量 和 在合并更新中使用配方变量。 • 在进程间通信 (IPC) * 授权策略中添加对通配符的完全支持。现在，您可以在资源字符串中*指定字符以匹配任意字符组合。有关更多信息，请参阅 授权策略中的通配符。 • 添加了对自定义组件的支持，以调用 Greengrass CLI 使用的 IPC 操作。您可以使用这些 IPC 操作来管理本地部署、查看组件详细信息以及生成用于登录本地调试控制台的密码。有关更多信息，请参阅 IPC：管理本地部署和组件。 错误修复和改进 • 修复了依赖组件在某些情况下在硬依赖项重新启动或更改状态时不会做出反应的问题。 • 改进了部署失败时核心设备向 AWS IoT Greengrass 云服务报告的错误消息。 • 修复了在某些情况下，当原子核重启时，Greengrass 核两次应用事物部署的问题。 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。

版本	更改
2.5.6	新功能 • 增加了对使用 ECC 密钥的硬件安全模块的支持。您可以使用硬件安全模块 (HSM) 来安全存储设备的私钥和证书。有关更多信息，请参阅 硬件安全性集成。 错误修复和改进 • 修复了在某些情况下部署安装脚本损坏的组件时，部署永远不会完成的问题。 • 提高了启动期间的性能。 • 其他小修复和改进。
2.5.5	新功能 • 为组件添加GG_ROOT_CA_PATH 环境变量，这样您就可以在自定义组件中访问根证书颁发机构 (CA) 证书。 错误修复和改进 • 添加对使用非英语显示语言的 Windows 设备的支持。 • 更新 Greengrass nucleus 解析布尔安装程序参数的方式，因此您可以指定不带布尔值的布尔参数来指定值。true例如，您现在可以指定--provision 而不是使用自动资源配置--provision true 进行安装。 • 修复了在某些情况下核心设备在配置后未向 AWS IoT Greengrass 云服务报告其状态的问题。 • 其他小修复和改进。
2.5.4	错误修复和改进 • 常规错误修复和性能改进。
2.5.3	新功能 • 增加了对硬件安全集成的支持。您可以使用硬件安全模块 (HSM) 来安全存储设备的私钥和证书。有关更多信息，请参阅 硬件安全性集成。 错误修复和改进 • 修复了在 nucleus 与建立 MQTT 连接时出现运行时异常的问题。 AWS IoT Core

版本	更改
2.5.2	错误修复和改进 • 修复了 Greengrass nucleus 更新后，Windows 服务在你停止或重启设备后无法重新启动的问题。
2.5.1	⚠ Warning 此版本不再可用。此版本的改进将在此组件的更高版本中提供。 错误修复和改进 • 在 Windows 上添加对 32 位版本的 Java 运行时环境 (JRE) 的支持。 • 更改 AWS IoT 策略未授予`greengrass:ListThingGroupsForCoreDevice` 权限的核心设备的事物组移除行为。在此版本中，部署会继续，记录警告，并且在从事物组中移除核心设备时不会移除组件。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。 • 修复了 Greengrass 核心向 Greengrass 组件进程提供的系统环境变量的问题。现在，您可以重新启动组件，使其使用最新的系统环境变量。

版本	更改
2.5.0	新功能 • 增加了对运行 Windows 的核心设备的支持。 • 更改移除事物组的行为。使用此版本，您可以从事物组中移除核心设备，以便在下次部署中卸载该事物组的组件。 由于此更改，核心设备的 AWS IoT 策略必须具有greengrass:ListThingGroupsForCoreDevice 权限。如果您使用AWS IoT Greengrass 核心软件安装程序来配置资源，则默认 AWS IoT 策略允许greengrass:* ，其中包括此权限。有关更多信息，请参阅 AWS IoT Greengrass 的设备身份验证和授权。 • 添加了对 HTTPS 代理配置的支持。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。 • 添加新的windowsUser 配置参数。您可以使用此参数指定用于在 Windows 核心设备上运行组件的默认用户。有关更多信息，请参阅 配置运行组件的用户。 • 添加了新的httpClient 配置选项，您可以使用这些选项来自定义 HTTP 请求超时时间，从而提高慢速网络的性能。有关更多信息，请参阅 HttpClient 配置参数。 错误修复和改进 • 修复了从组件重启核心设备的 bootstrap 生命周期选项。 • 在配方变量中添加对连字符的支持。 • 修复了按需 Lambda 函数组件的 IPC 授权。 • 改进了日志消息并将非关键日志从级别更改INFO为DEBUG级别，因此日志更有用。 • 移除 Greengrass nucleus 在安装 AWS IoT Greengrass 具有自动配置功能的 Core 软件时创建的默认令牌交换角色的iot:DescribeCertificate 权限。Greengrass 核不使用此权限。 • 修复了一个问题，使自动配置脚本不需要该iam:GetPolicy 权限（如果iam:CreatePolicy 适用于同一策略）。 • 其他小修复和改进。

版本	更改
2.4.0	新功能 • 增加了对系统资源限制的支持。您可以配置每个组件的进程可以在核心设备上使用的最大 CPU 和 RAM 使用量。有关更多信息，请参阅 为组件配置系统资源限制。 • 添加 IPC 操作以暂停和恢复组件。有关更多信息，请参阅 PauseComponent 和 ResumeComponent。 • 添加了对配置插件的支持。您可以指定在安装期间运行的 JAR 文件，以便为 Greengrass 核心设备配置所需的 AWS 资源。Greengrass 核心包含一个接口，您可以实现该接口来开发自定义配置插件。有关更多信息，请参阅 安装具有自定义资源配置功能的 C AWS IoT Greengrass ore 软件。 • 向 C AWS IoT Greengrass ore 软件安装程序添加可选`thing-name-policy`参数。在安装具有自动资源配置功能的 C AWS IoT Greengrass ore 软件时，您可以使用此选项来指定现有 AWS IoT 策略或自定义策略。 错误修复和改进 • 启动时更新日志配置。这修复了启动时未应用日志配置的问题。 • 在安装过程中，更新 nucleus 加载器符号链接以指向 Greengrass 根文件夹中的组件存储。此更新使您能够删除安装 C AWS IoT Greengrass ore 软件时下载的 JAR 文件和其他 Nucleus 工件。 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。

版本	更改
2.3.0	新功能 • 添加对部署配置文档的支持，最大为 10 MB，从 7 KB（针对事物的部署）或 31 KB（针对事物组的部署）增加了。 要使用此功能，核心设备的 AWS IoT 策略必须允许该`greengrass:GetDeploymentConfiguration` 权限。如果您使用AWS IoT Greengrass 核心软件安装程序来配置资源，则您的核心设备的 AWS IoT 策略允许`greengrass:*` ，其中包括此权限。有关更多信息，请参阅 AWS IoT Greengrass 的设备身份验证和授权。 • 添加`iot:thingName` 配方变量。你可以使用这个配方变量来获取配方中 AWS IoT 核心设备的名称。有关更多信息，请参阅 食谱变量。 错误修复和改进 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。
2.2.0	新功能 • 为本地影子管理添加 IPC 操作。 错误修复和改进 • 减小 JAR 文件的大小。 • 减少内存使用量。 • 修复了在某些情况下日志配置未更新的问题。 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。

版本	更改

2.1.0	新功能
	• 支持从 Amazon ECR 中的私有存储库下载 Docker 镜像。
	• 添加以下参数以自定义核心设备上的 MQTT 配置：
	  • `maxInFlightPublishes` — 可以同时传输的未确认的 MQTT QoS 1 消息的最大数量。
	  • `maxPublishRetry` — 重试发布失败的消息的最大次数。
	• 添加`fleetstatusservice` 配置参数以配置核心设备向发布设备状态的时间间隔 AWS Cloud。
	• 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。
	错误修复和改进
	• 修复了在 nucleus 重启时导致影子部署重复的问题。
	• 修复了在遇到服务加载异常时导致 nucleus 崩溃的问题。
	• 改进了组件依赖关系解决方案，使包含循环依赖关系的部署失败。
	• 修复了如果插件组件之前已从核心设备中移除，则无法重新部署该组件的问题。
	• 修复了导致将`HOME`环境变量设置为 Lambda 组件或以 root 身份运行的组件的`/greengrass/v2 /work`目录的问题。现在，该`HOME`变量已正确设置为运行该组件的用户的主目录。
	• 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。

2.0.5	错误修复和改进 • 下载 AWS提供的组件时，通过配置的网络代理正确路由流量。 • 在中国区域使用正确的 Greengrass 数据平面终端节点。 AWS

版本	更改
2.0.4	新功能 • 启用通过端口 443 的 HTTPS 流量。你可以使用 nucleus 组件版本 2.0.4 的新`greengrassDataPlanePort` 配置参数将 HTTPS 通信配置为通过端口 443 而不是默认端口 8443 传输。有关更多信息，请参阅 通过端口 443 配置 HTTPS。 • 添加工作路径配方变量。您可以使用此配方变量来获取组件工作文件夹的路径，您可以使用该路径在组件及其依赖项之间共享文件。有关更多信息，请参阅工作路径配方变量。 错误修复和改进 • 如果角色策略已经存在，则阻止创建令牌交换 AWS Identity and Access Management (IAM) 角色策略。 由于这一更改，安装程序现在需要`sts:GetCallerIdentity` 使用`iam:GetPolicy` 和`--provision true` 。有关更多信息，请参阅安装程序配置资源的最低 IAM 政策。 • 正确处理尚未成功注册的部署的取消问题。 • 更新配置以在回滚部署时删除带有较新时间戳的较旧条目。 • 其他小修复和改进。有关更多信息，请参阅上的版本 GitHub。
2.0.3	初始版本。

客户端设备身份验证

客户端设备身份验证组件 (`aws.greengrass.clientdevices.Auth`) 对客户端设备进行身份验证并授权客户端设备操作。

Note

客户端设备是连接到 Greengrass 核心设备以发送 MQTT 消息和数据进行处理的本地物联网设备。有关更多信息，请参阅 与本地物联网设备互动。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

> **Note**
>
> 客户端设备身份验证版本 2.3.0 已停产。我们强烈建议您升级到客户端设备身份验证版本 2.3.1 或更高版本。

此组件有以下版本：

- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- Greengrass 服务角色必须与您的关联并允许 AWS 账户 该权限。`iot:DescribeCertificate`
- 核心设备的 AWS IoT 策略必须允许以下权限：
  - `greengrass:GetConnectivityInfo`，其中资源包括运行该组件的核心设备的 ARN
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`，其中资源包括连接到核心设备的每台客户端设备的 Amazon 资源名称 (ARN)
  - `greengrass:VerifyClientDeviceIdentity`
  - `greengrass:PutCertificateAuthorities`
  - `iot:Publish`，其中资源包括以下 MQTT 主题的 ARN：
    - `$aws/things/coreDeviceThingName*-gci/shadow/get`
  - `iot:Subscribe`，其中资源包括以下 MQTT 主题过滤器的 ARN：
    - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
    - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`
  - `iot:Receive`，其中资源包括以下 MQTT 主题的 ARN：
    - `$aws/things/coreDeviceThingName*-gci/shadow/update/delta`
    - `$aws/things/coreDeviceThingName*-gci/shadow/get/accepted`

  有关更多信息，请参阅 数据层面操作的 AWS IoT 策略 和 支持客户端设备的最低AWS IoT政策。
- （可选）要使用离线身份验证， AWS IoT Greengrass 服务使用的 AWS Identity and Access Management (IAM) 角色必须包含以下权限：
  - `greengrass:ListClientDevicesAssociatedWithCoreDevice`使核心设备能够列出客户机进行离线身份验证。
- 支持在 VPC 中运行客户端设备身份验证组件。要在 VPC 中部署此组件，需要满足以下条件。
  - 客户端设备身份验证组件必须连接到 AWS IoT data、 AWS IoT 凭证和 Amazon S3。

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
iot.*region*.amazonaws.com	443	支持	用于获取有关 AWS IoT 事物证书的信息。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.4.4

下表列出了此组件版本 2.4.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.6.0 <2.13.0	软性

2.4.3

下表列出了此组件版本 2.4.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.6.0 <2.12.0	软性

2.4.1 and 2.4.2

下表列出了此组件版本 2.4.1 和 2.4.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.6.0 <2.11.0	软性

2.3.0 – 2.4.0

下表列出了此组件版本 2.3.0 到 2.4.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.6.0 <2.10.0	软性

2.3.0

下表列出了此组件版本 2.3.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.6.0 <2.10.0	软性

2.2.3

下表列出了此组件版本 2.2.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.6.0 <=2.9.0	软性

2.2.2

下表列出了此组件版本 2.2.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.6.0 <=2.8.0	软性

2.2.1

下表列出了此组件版本 2.2.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.6.0 <2.8.0	软性

2.2.0

下表列出了此组件版本 2.2.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.6.0 <2.7.0	软性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.7.0	软性

2.0.4

下表列出了此组件版本 2.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.6.0	软性

2.0.2 and 2.0.3

下表列出了此组件版本 2.0.2 和 2.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.5.0	软性

2.0.1

下表列出了此组件版本 2.0.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.4.0	软性

2.0.0

下表列出了此组件版本 2.0.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.3.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

> **Note**
>
> 订阅权限是在客户向本地 MQTT 代理发出订阅请求期间进行评估的。如果客户现有的订阅权限被撤销，则该客户端将无法再订阅某个主题。但是，它将继续接收来自任何以前订阅的主题的消息。为防止这种行为，应在撤销订阅权限后重新启动本地 MQTT 代理，以强制重新授权客户端。
> 对于 MQTT 5 代理 (EMQX) 组件，请更新`restartIdentifier`配置以重启 MQTT 5 代理。有关更多信息，请参阅 MQTT 5 代理组件配置。

对于 MQTT 3.1.1 代理 (Moquette) 组件，当服务器证书更改迫使客户端重新授权时，它默认每周重新启动一次。您可以通过更改核心设备的连接信息（IP 地址）来强制重启，也可以通过部署移除代理组件然后稍后重新部署来强制重启。

v2.4.5

`deviceGroups`

设备组是指有权与核心设备连接和通信的客户端设备组。使用选择规则来识别客户端设备组，并定义为每个设备组指定权限的客户端设备授权策略。

该对象包含以下信息：

`formatVersion`

此配置对象的格式版本。

从以下选项中进行选择：

- `2021-03-05`

`definitions`

此核心设备的设备组。每个定义都指定了用于评估客户端设备是否为该组成员的选择规则。每个定义还指定了要应用于与选择规则相匹配的客户端设备的权限策略。如果一台客户端设备是多个设备组的成员，则该设备的权限由每个组的权限策略组成。

该对象包含以下信息：

`groupNameKey`

该设备组的名称。*`groupNameKey`*替换为可帮助您识别此设备组的名称。

该对象包含以下信息：

`selectionRule`

用于指定哪些客户端设备是该设备组成员的查询。当客户端设备连接时，核心设备会评估此选择规则，以确定该客户端设备是否为该设备组的成员。如果客户端设备是成员，则核心设备将使用该设备组的策略来授权客户端设备的操作。

每条选择规则至少包含一个选择规则子句，即可以匹配客户端设备的单个表达式查询。选择规则使用的查询语法与 AWS IoT 舰队索引相同。有关选择规则语法的更多信息，请参阅《AWS IoT Core 开发人员指南》中的AWS IoT 舰队索引查询语法。

使用*通配符将多个客户端设备与一个选择规则子句进行匹配。您可以在事物名称的开头和结尾使用此通配符来匹配名称以您指定的字符串开头或结尾的客户端设备。您也可以使用此通配符来匹配所有客户端设备。

> **Note**
>
> 要选择包含冒号字符 (:) 的值，请使用反斜杠字符 (\) 对冒号进行转义。在 JSON 等格式中，必须对反斜杠字符进行转义，因此在冒号字符之前输入两个反斜杠字符。例如，指定`thingName: MyTeam\\:ClientDevice1`选择名称为的事物`MyTeam:ClientDevice1`。

您可以指定以下选择器：

- `thingName`— 客户端设备的 AWS IoT 名称。

Example 选择规则示例

以下选择规则匹配名称为`MyClientDevice1`或的客户端设备`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 选择规则示例（使用通配符）

以下选择规则匹配名称以开头的客户端设备`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 选择规则示例（使用通配符）

以下选择规则匹配名称以结尾的客户端设备`MyClientDevice`。

```
thingName: *MyClientDevice
```

Example 选择规则示例（匹配所有设备）

以下选择规则匹配所有客户端设备。

```
thingName: *
```

policyName

适用于该设备组中的客户端设备的权限策略。指定您在policies对象中定义的策略的名称。

policies

连接到核心设备的客户端设备的客户端设备授权策略。每项授权策略都指定了一组操作以及客户端设备可以在其中执行这些操作的资源。

该对象包含以下信息：

policyNameKey

此授权策略的名称。*policyNameKey*替换为可帮助您识别此授权策略的名称。您可以使用此策略名称来定义哪个策略适用于设备组。

该对象包含以下信息：

statementNameKey

本政策声明的名称。*statementNameKey*替换为可帮助您识别此政策声明的名称。

该对象包含以下信息：

operations

允许使用此策略中的资源的操作列表。

您可以包括以下任何操作：

- mqtt:connect— 授予连接核心设备的权限。客户端设备必须具有此权限才能连接到核心设备。

  此操作支持以下资源：

  - mqtt:clientId:*deviceClientId*— 根据客户端设备用于连接核心设备的 MQTT 代理的客户端 ID 限制访问。*deviceClientId*替换为要使用的客户端 ID。
- mqtt:publish— 授予向主题发布 MQTT 消息的权限。

  此操作支持以下资源：

  - mqtt:topic:*mqttTopic*— 根据客户端设备发布消息的 MQTT 主题限制访问。将 *mqttTop* ic 替换为要使用的主题。

    此资源不支持 MQTT 主题通配符。

- `mqtt:subscribe`— 授予订阅 MQTT 主题过滤器以接收消息的权限。

  此操作支持以下资源：

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 根据客户端设备可以订阅消息的 MQTT 主题限制访问权限。*`mqttTopicFilter`*替换为要使用的主题筛选器。

    此资源支持+和 # MQTT 主题通配符。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的 MQTT 主题。

    客户端设备可以订阅您允许的确切主题过滤器。例如，如果您允许客户端设备订阅`mqtt:topicfilter:client/+/status`资源，则客户端设备可以订阅，`client/+/status`但不能订阅`client/client1/status`。

您可以指定`*`通配符以允许访问所有操作。

`resources`

允许在此策略中进行操作的资源列表。指定与此策略中的操作相对应的资源。例如，您可以在指定`mqtt:publish`操作的策略中指定 MQTT 主题资源列表 (`mqtt:topic:`*`mqttTopic`*)。

您可以指定`*`通配符以允许访问所有资源。您不能使用`*`通配符来匹配部分资源标识符。例如，您可以指定**`"resources": "*"`**，但不能指定**`"resources": "mqtt:clientId:*"`**。

`statementDescription`

（可选）此政策声明的描述。

`certificates`

（可选）此核心设备的证书配置选项。该对象包含以下信息：

`serverCertificateValiditySeconds`

（可选）本地 MQTT 服务器证书过期的时间（以秒为单位）。您可以配置此选项以自定义客户端设备断开连接并重新连接到核心设备的频率。

此组件会在本地 MQTT 服务器证书到期前 24 小时对其进行轮换。MQTT 代理（例如 Moquett e MQTT 代理组件）会生成新证书并重新启动。发生这种情况时，所有连接到该核心设备的客户端设备都将断开连接。客户机设备可以在短时间后重新连接到核心设备。

默认：`604800`（7 天）

最小值：172800（2 天）

最大值：864000（10 天）

`performance`

（可选）此核心设备的性能配置选项。该对象包含以下信息：

`maxActiveAuthTokens`

（可选）活动客户端设备授权令牌的最大数量。您可以增加此数字，使更多的客户端设备能够连接到单个核心设备，而无需重新对其进行身份验证。

默认：2500

`cloudRequestQueueSize`

（可选）在此组件拒绝 AWS Cloud 请求之前要排队的最大请求数。

默认：100

`maxConcurrentCloudRequests`

（可选）要发送到的最大并发请求数 AWS Cloud。您可以增加此数字，以提高连接大量客户端设备的核心设备的身份验证性能。

默认：1

`certificateAuthority`

（可选）证书颁发机构配置选项，用您自己的中间证书颁发机构替换核心设备中间颁发机构。

> **Note**
>
> 如果您将 Greengrass 核心设备配置为自定义证书颁发机构 (CA)，并使用相同的 CA 颁发客户端设备证书，则 Greengrass 会绕过对客户端设备 MQTT 操作的授权策略检查。客户端设备身份验证组件完全信任使用由其配置为使用的 CA 签名的证书的客户端。要在使用自定义 CA 时限制此行为，请使用其他 CA 或中间 CA 创建和签署客户端设备，然后调整`certificateUri`和`certificateChainUri`字段以指向正确的中间 CA。

此对象包含以下信息。

证书网址

证书的位置。它可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书的 URI。

`certificateChainUri`

核心设备 CA 的证书链的位置。这应该是返回到您的根 CA 的完整证书链。它可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书链的 URI。

`privateKeyUri`

核心设备私钥的位置。这可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书私钥的 URI。

`security`

（可选）此核心设备的安全配置选项。此对象包含以下信息。

`clientDeviceTrustDurationMinutes`

在要求客户端设备重新进行身份验证之前，客户端设备的身份验证信息可以被信任的持续时间（以分钟为单位）。默认值是 1。

`metrics`

（可选）此核心设备的指标选项。只有在客户端设备身份验证出现错误时，才会显示错误指标。该对象包含以下信息：

`disableMetrics`

如果该`disableMetrics`字段设置为`true`，则客户端设备身份验证将不会收集指标。

默认：`false`

`aggregatePeriodSeconds`

以秒为单位的聚合周期，它决定了客户端设备身份验证聚合指标并将其发送到遥测代理的频率。这不会改变指标发布的频率，因为遥测代理仍然每天发布一次。

默认：`3600`

startupTimeoutSeconds

（可选）组件启动的最长时间（以秒为单位）。`BROKEN`如果超过此超时时间，则组件的状态将更改为。

默认：120

Example 示例：配置合并更新（使用限制性策略）

以下示例配置指定允许名称以开头的MyClientDevice客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
```

```
          ]
        }
      }
    }
  }
}
```

Example 示例：配置合并更新（使用许可策略）

以下示例配置指定允许所有客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.2 - v2.4.4

`deviceGroups`

设备组是指有权与核心设备连接和通信的客户端设备组。使用选择规则来识别客户端设备组，并定义为每个设备组指定权限的客户端设备授权策略。

该对象包含以下信息：

`formatVersion`

此配置对象的格式版本。

从以下选项中进行选择：

- `2021-03-05`

`definitions`

此核心设备的设备组。每个定义都指定了用于评估客户端设备是否为该组成员的选择规则。每个定义还指定了要应用于与选择规则相匹配的客户端设备的权限策略。如果一台客户端设备是多个设备组的成员，则该设备的权限由每个组的权限策略组成。

该对象包含以下信息：

`groupNameKey`

该设备组的名称。*`groupNameKey`*替换为可帮助您识别此设备组的名称。

该对象包含以下信息：

`selectionRule`

用于指定哪些客户端设备是该设备组成员的查询。当客户端设备连接时，核心设备会评估此选择规则，以确定该客户端设备是否为该设备组的成员。如果客户端设备是成员，则核心设备将使用该设备组的策略来授权客户端设备的操作。

每条选择规则至少包含一个选择规则子句，即可以匹配客户端设备的单个表达式查询。选择规则使用的查询语法与 AWS IoT 舰队索引相同。有关选择规则语法的更多信息，请参阅《AWS IoT Core 开发人员指南》中的AWS IoT 舰队索引查询语法。

使用`*`通配符将多个客户端设备与一个选择规则子句进行匹配。您可以在事物名称末尾使用此通配符来匹配名称以您指定的字符串开头的客户端设备。您也可以使用此通配符来匹配所有客户端设备。

> **Note**
>
> 要选择包含冒号字符 (`:`) 的值，请使用反斜杠字符 (`\\`) 对冒号进行转义。在 JSON 等格式中，必须对反斜杠字符进行转义，因此在冒号字符之前输入两个反斜杠字符。例如，指定`thingName: MyTeam\\\\:ClientDevice1`选择名称为的事物`MyTeam:ClientDevice1`。

您可以指定以下选择器：

- `thingName`— 客户端设备的 AWS IoT 名称。

Example 选择规则示例

以下选择规则匹配名称为`MyClientDevice1`或的客户端设备`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 选择规则示例（使用通配符）

以下选择规则匹配名称以开头的客户端设备`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 选择规则示例（匹配所有设备）

以下选择规则匹配所有客户端设备。

```
thingName: *
```

`policyName`

适用于该设备组中的客户端设备的权限策略。指定您在`policies`对象中定义的策略的名称。

`policies`

连接到核心设备的客户端设备的客户端设备授权策略。每项授权策略都指定了一组操作以及客户端设备可以在其中执行这些操作的资源。

该对象包含以下信息：

`policyNameKey`

此授权策略的名称。*`policyNameKey`*替换为可帮助您识别此授权策略的名称。您可以使用此策略名称来定义哪个策略适用于设备组。

该对象包含以下信息：

`statementNameKey`

本政策声明的名称。*`statementNameKey`*替换为可帮助您识别此政策声明的名称。

该对象包含以下信息：

`operations`

允许使用此策略中的资源的操作列表。

您可以包括以下任何操作：

- `mqtt:connect`— 授予连接核心设备的权限。客户端设备必须具有此权限才能连接到核心设备。

  此操作支持以下资源：

  - `mqtt:clientId:`*`deviceClientId`*— 根据客户端设备用于连接核心设备的 MQTT 代理的客户端 ID 限制访问。*`deviceClientId`*替换为要使用的客户端 ID。
- `mqtt:publish`— 授予向主题发布 MQTT 消息的权限。

  此操作支持以下资源：

  - `mqtt:topic:`*`mqttTopic`*— 根据客户端设备发布消息的 MQTT 主题限制访问。将 *`mqttTop`* ic 替换为要使用的主题。

    此资源不支持 MQTT 主题通配符。
- `mqtt:subscribe`— 授予订阅 MQTT 主题过滤器以接收消息的权限。

  此操作支持以下资源：

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 根据客户端设备可以订阅消息的 MQTT 主题限制访问权限。*`mqttTopicFilter`*替换为要使用的主题筛选器。

    此资源支持+和 # MQTT 主题通配符。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的 MQTT 主题。

    客户端设备可以订阅您允许的确切主题过滤器。例如，如果您允许客户端设备订阅`mqtt:topicfilter:client/+/status`资源，则客户端设备可以订阅，`client/+/status`但不能订阅`client/client1/status`。

您可以指定`*`通配符以允许访问所有操作。

`resources`

允许在此策略中进行操作的资源列表。指定与此策略中的操作相对应的资源。例如，您可以在指定mqtt:publish操作的策略中指定 MQTT 主题资源列表 (mqtt:topic:*mqttTopic*)。

您可以指定*通配符以允许访问所有资源。您不能使用*通配符来匹配部分资源标识符。例如，您可以指定**"resources": "*"**，但不能指定**"resources": "mqtt:clientId:*"**。

`statementDescription`

（可选）此政策声明的描述。

`certificates`

（可选）此核心设备的证书配置选项。该对象包含以下信息：

`serverCertificateValiditySeconds`

（可选）本地 MQTT 服务器证书过期的时间（以秒为单位）。您可以配置此选项以自定义客户端设备断开连接并重新连接到核心设备的频率。

此组件会在本地 MQTT 服务器证书到期前 24 小时对其进行轮换。MQTT 代理（例如 Moquett e MQTT 代理组件）会生成新证书并重新启动。发生这种情况时，所有连接到该核心设备的客户端设备都将断开连接。客户机设备可以在短时间后重新连接到核心设备。

默认：604800（7 天）

最小值：172800（2 天）

最大值：864000（10 天）

`performance`

（可选）此核心设备的性能配置选项。该对象包含以下信息：

`maxActiveAuthTokens`

（可选）活动客户端设备授权令牌的最大数量。您可以增加此数字，使更多的客户端设备能够连接到单个核心设备，而无需重新对其进行身份验证。

默认：2500

`cloudRequestQueueSize`

（可选）在此组件拒绝 AWS Cloud 请求之前要排队的最大请求数。

默认：100

maxConcurrentCloudRequests

（可选）要发送到的最大并发请求数 AWS Cloud。您可以增加此数字，以提高连接大量客户端设备的核心设备的身份验证性能。

默认：1

certificateAuthority

（可选）证书颁发机构配置选项，用您自己的中间证书颁发机构替换核心设备中间颁发机构。

> **Note**
>
> 如果您将 Greengrass 核心设备配置为自定义证书颁发机构 (CA)，并使用相同的 CA 颁发客户端设备证书，则 Greengrass 会绕过对客户端设备 MQTT 操作的授权策略检查。客户端设备身份验证组件完全信任使用由其配置为使用的 CA 签名的证书的客户端。要在使用自定义 CA 时限制此行为，请使用其他 CA 或中间 CA 创建和签署客户端设备，然后调整`certificateUri`和`certificateChainUri`字段以指向正确的中间 CA。

此对象包含以下信息。

证书网址

证书的位置。它可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书的 URI。

certificateChainUri

核心设备 CA 的证书链的位置。这应该是返回到您的根 CA 的完整证书链。它可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书链的 URI。

privateKeyUri

核心设备私钥的位置。这可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书私钥的 URI。

security

（可选）此核心设备的安全配置选项。此对象包含以下信息。

`clientDeviceTrustDurationMinutes`

在要求客户端设备重新进行身份验证之前，客户端设备的身份验证信息可以被信任的持续时间（以分钟为单位）。默认值是 1。

`metrics`

（可选）此核心设备的指标选项。只有在客户端设备身份验证出现错误时，才会显示错误指标。该对象包含以下信息：

`disableMetrics`

如果该`disableMetrics`字段设置为`true`，则客户端设备身份验证将不会收集指标。

默认：`false`

`aggregatePeriodSeconds`

以秒为单位的聚合周期，它决定了客户端设备身份验证聚合指标并将其发送到遥测代理的频率。这不会改变指标发布的频率，因为遥测代理仍然每天发布一次。

默认：`3600`

startupTimeoutSeconds

（可选）组件启动的最长时间（以秒为单位）。`BROKEN`如果超过此超时时间，则组件的状态将更改为。

默认：`120`

Example 示例：配置合并更新（使用限制性策略）

以下示例配置指定允许名称以开头的`MyClientDevice`客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
```

```
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 示例：配置合并更新（使用许可策略）

以下示例配置指定允许所有客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
```

```
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.0 - v2.4.1

`deviceGroups`

设备组是指有权与核心设备连接和通信的客户端设备组。使用选择规则来识别客户端设备组，并定义为每个设备组指定权限的客户端设备授权策略。

该对象包含以下信息：

`formatVersion`

此配置对象的格式版本。

从以下选项中进行选择：

- `2021-03-05`

`definitions`

此核心设备的设备组。每个定义都指定了用于评估客户端设备是否为该组成员的选择规则。每个定义还指定了要应用于与选择规则相匹配的客户端设备的权限策略。如果一台客户端设备是多个设备组的成员，则该设备的权限由每个组的权限策略组成。

该对象包含以下信息：

groupNameKey

该设备组的名称。*groupNameKey*替换为可帮助您识别此设备组的名称。

该对象包含以下信息：

`selectionRule`

用于指定哪些客户端设备是该设备组成员的查询。当客户端设备连接时，核心设备会评估此选择规则，以确定该客户端设备是否为该设备组的成员。如果客户端设备是成员，则核心设备使用该设备组的策略来授权客户端设备的操作。

每条选择规则至少包含一个选择规则子句，即可以匹配客户端设备的单个表达式查询。选择规则使用的查询语法与 AWS IoT 舰队索引相同。有关选择规则语法的更多信息，请参阅《AWS IoT Core 开发人员指南》中的AWS IoT 舰队索引查询语法。

使用*通配符将多个客户端设备与一个选择规则子句进行匹配。您可以在事物名称末尾使用此通配符来匹配名称以您指定的字符串开头的客户端设备。您也可以使用此通配符来匹配所有客户端设备。

> **Note**
>
> 要选择包含冒号字符 (:) 的值，请使用反斜杠字符 (\\) 对冒号进行转义。在 JSON 等格式中，必须对反斜杠字符进行转义，因此在冒号字符之前输入两个反斜杠字符。例如，指定`thingName: MyTeam\\\\:ClientDevice1`选择名称为的事物`MyTeam:ClientDevice1`。

您可以指定以下选择器：

- `thingName`— 客户端设备的 AWS IoT 名称。

Example 选择规则示例

以下选择规则匹配名称为`MyClientDevice1`或的客户端设备`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 选择规则示例（使用通配符）

以下选择规则匹配名称以开头的客户端设备`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 选择规则示例（匹配所有设备）

以下选择规则匹配所有客户端设备。

```
thingName: *
```

`policyName`

适用于该设备组中的客户端设备的权限策略。指定您在`policies`对象中定义的策略的名称。

`policies`

连接到核心设备的客户端设备的客户端设备授权策略。每项授权策略都指定了一组操作以及客户端设备可以在其中执行这些操作的资源。

该对象包含以下信息：

`policyNameKey`

此授权策略的名称。*`policyNameKey`*替换为可帮助您识别此授权策略的名称。您可以使用此策略名称来定义哪个策略适用于设备组。

该对象包含以下信息：

`statementNameKey`

本政策声明的名称。*`statementNameKey`*替换为可帮助您识别此政策声明的名称。

该对象包含以下信息：

`operations`

允许使用此策略中的资源的操作列表。

您可以包括以下任何操作：

- `mqtt:connect`— 授予连接核心设备的权限。客户端设备必须具有此权限才能连接到核心设备。

此操作支持以下资源：

- `mqtt:clientId:`*`deviceClientId`*— 根据客户端设备用于连接核心设备的 MQTT 代理的客户端 ID 限制访问。*`deviceClientId`*替换为要使用的客户端 ID。

- `mqtt:publish`— 授予向主题发布 MQTT 消息的权限。

  此操作支持以下资源：

  - `mqtt:topic:`*`mqttTopic`*— 根据客户端设备发布消息的 MQTT 主题限制访问。将 *`mqttTop`* ic 替换为要使用的主题。

    此资源不支持 MQTT 主题通配符。

- `mqtt:subscribe`— 授予订阅 MQTT 主题过滤器以接收消息的权限。

  此操作支持以下资源：

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 根据客户端设备可以订阅消息的 MQTT 主题限制访问权限。*`mqttTopicFilter`*替换为要使用的主题筛选器。

    此资源支持+和 # MQTT 主题通配符。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的 MQTT 主题。

    客户端设备可以订阅您允许的确切主题过滤器。例如，如果您允许客户端设备订阅`mqtt:topicfilter:client/+/status`资源，则客户端设备可以订阅，`client/+/status`但不能订阅`client/client1/status`。

您可以指定`*`通配符以允许访问所有操作。

`resources`

允许在此策略中进行操作的资源列表。指定与此策略中的操作相对应的资源。例如，您可以在指定`mqtt:publish`操作的策略中指定 MQTT 主题资源列表 (`mqtt:topic:`*`mqttTopic`*)。

您可以指定`*`通配符以允许访问所有资源。您不能使用`*`通配符来匹配部分资源标识符。例如，您可以指定**`"resources": "*"`**，但不能指定**`"resources": "mqtt:clientId:*"`**。

`statementDescription`

（可选）此政策声明的描述。

`certificates`

（可选）此核心设备的证书配置选项。该对象包含以下信息：

`serverCertificateValiditySeconds`

（可选）本地 MQTT 服务器证书过期的时间（以秒为单位）。您可以配置此选项以自定义客户端设备断开连接并重新连接到核心设备的频率。

此组件会在本地 MQTT 服务器证书到期前 24 小时对其进行轮换。MQTT 代理（例如 Moquett e MQTT 代理组件）会生成新证书并重新启动。发生这种情况时，所有连接到该核心设备的客户端设备都将断开连接。客户机设备可以在短时间后重新连接到核心设备。

默认：`604800`（7 天）

最小值：`172800`（2 天）

最大值：`864000`（10 天）

`performance`

（可选）此核心设备的性能配置选项。该对象包含以下信息：

`maxActiveAuthTokens`

（可选）活动客户端设备授权令牌的最大数量。您可以增加此数字，使更多的客户端设备能够连接到单个核心设备，而无需重新对其进行身份验证。

默认：`2500`

`cloudRequestQueueSize`

（可选）在此组件拒绝 AWS Cloud 请求之前要排队的最大请求数。

默认：`100`

`maxConcurrentCloudRequests`

（可选）要发送到的最大并发请求数 AWS Cloud。您可以增加此数字，以提高连接大量客户端设备的核心设备的身份验证性能。

默认：`1`

`certificateAuthority`

（可选）证书颁发机构配置选项，用您自己的中间证书颁发机构替换核心设备中间颁发机构。此对象包含以下信息。

该对象包含以下信息：

证书网址

证书的位置。它可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书的 URI。

`certificateChainUri`

核心设备 CA 的证书链的位置。这应该是返回到您的根 CA 的完整证书链。它可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书链的 URI。

`privateKeyUri`

核心设备私钥的位置。这可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书私钥的 URI。

`security`

（可选）此核心设备的安全配置选项。此对象包含以下信息。

`clientDeviceTrustDurationMinutes`

在要求客户端设备重新进行身份验证之前，客户端设备的身份验证信息可以被信任的持续时间（以分钟为单位）。默认值是 1。

`metrics`

（可选）此核心设备的指标选项。只有在客户端设备身份验证出现错误时，才会显示错误指标。该对象包含以下信息：

`disableMetrics`

如果该`disableMetrics`字段设置为`true`，则客户端设备身份验证将不会收集指标。

默认：`false`

`aggregatePeriodSeconds`

以秒为单位的聚合周期，它决定了客户端设备身份验证聚合指标并将其发送到遥测代理的频率。这不会改变指标发布的频率，因为遥测代理仍然每天发布一次。

默认：`3600`

Example 示例：配置合并更新（使用限制性策略）

以下示例配置指定允许名称以开头的`MyClientDevice`客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
```

```
      }
    }
  }
```

Example 示例：配置合并更新（使用许可策略）

以下示例配置指定允许所有客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.3.x

`deviceGroups`

设备组是指有权与核心设备连接和通信的客户端设备组。使用选择规则来识别客户端设备组，并定义为每个设备组指定权限的客户端设备授权策略。

该对象包含以下信息：

`formatVersion`

此配置对象的格式版本。

从以下选项中进行选择：

- `2021-03-05`

`definitions`

此核心设备的设备组。每个定义都指定了用于评估客户端设备是否为该组成员的选择规则。每个定义还指定了要应用于与选择规则相匹配的客户端设备的权限策略。如果一台客户端设备是多个设备组的成员，则该设备的权限由每个组的权限策略组成。

该对象包含以下信息：

`groupNameKey`

该设备组的名称。*`groupNameKey`*替换为可帮助您识别此设备组的名称。

该对象包含以下信息：

`selectionRule`

用于指定哪些客户端设备是该设备组成员的查询。当客户端设备连接时，核心设备会评估此选择规则，以确定该客户端设备是否为该设备组的成员。如果客户端设备是成员，则核心设备使用该设备组的策略来授权客户端设备的操作。

每条选择规则至少包含一个选择规则子句，即可以匹配客户端设备的单个表达式查询。选择规则使用的查询语法与 AWS IoT 舰队索引相同。有关选择规则语法的更多信息，请参阅《AWS IoT Core 开发人员指南》中的AWS IoT 舰队索引查询语法。

使用`*`通配符将多个客户端设备与一个选择规则子句进行匹配。您可以在事物名称末尾使用此通配符来匹配名称以您指定的字符串开头的客户端设备。您也可以使用此通配符来匹配所有客户端设备。

> **Note**
>
> 要选择包含冒号字符 (`:`) 的值，请使用反斜杠字符 (`\\`) 对冒号进行转义。在 JSON 等格式中，必须对反斜杠字符进行转义，因此在冒号字符之前输入两个反斜杠字符。例如，指定`thingName: MyTeam\\\\:ClientDevice1`选择名称为的事物`MyTeam:ClientDevice1`。

您可以指定以下选择器：

- `thingName`— 客户端设备的 AWS IoT 名称。

Example 选择规则示例

以下选择规则匹配名称为`MyClientDevice1`或的客户端设备`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 选择规则示例（使用通配符）

以下选择规则匹配名称以开头的客户端设备`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 选择规则示例（匹配所有设备）

以下选择规则匹配所有客户端设备。

```
thingName: *
```

`policyName`

适用于该设备组中的客户端设备的权限策略。指定您在`policies`对象中定义的策略的名称。

`policies`

连接到核心设备的客户端设备的客户端设备授权策略。每项授权策略都指定了一组操作以及客户端设备可以在其中执行这些操作的资源。

该对象包含以下信息：

`policyNameKey`

此授权策略的名称。*`policyNameKey`*替换为可帮助您识别此授权策略的名称。您可以使用此策略名称来定义哪个策略适用于设备组。

该对象包含以下信息：

`statementNameKey`

本政策声明的名称。*`statementNameKey`*替换为可帮助您识别此政策声明的名称。

该对象包含以下信息：

`operations`

允许使用此策略中的资源的操作列表。

您可以包括以下任何操作：

- `mqtt:connect`— 授予连接核心设备的权限。客户端设备必须具有此权限才能连接到核心设备。

  此操作支持以下资源：

  - `mqtt:clientId:`*`deviceClientId`*— 根据客户端设备用于连接核心设备的 MQTT 代理的客户端 ID 限制访问。*`deviceClientId`*替换为要使用的客户端 ID。
- `mqtt:publish`— 授予向主题发布 MQTT 消息的权限。

  此操作支持以下资源：

  - `mqtt:topic:`*`mqttTopic`*— 根据客户端设备发布消息的 MQTT 主题限制访问。将 *`mqttTop`* ic 替换为要使用的主题。

    此资源不支持 MQTT 主题通配符。
- `mqtt:subscribe`— 授予订阅 MQTT 主题过滤器以接收消息的权限。

  此操作支持以下资源：

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 根据客户端设备可以订阅消息的 MQTT 主题限制访问权限。*`mqttTopicFilter`*替换为要使用的主题筛选器。

    此资源支持+和 # MQTT 主题通配符。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的 MQTT 主题。

    客户端设备可以订阅您允许的确切主题过滤器。例如，如果您允许客户端设备订阅`mqtt:topicfilter:client/+/status`资源，则客户端设备可以订阅，`client/+/status`但不能订阅`client/client1/status`。

您可以指定`*`通配符以允许访问所有操作。

`resources`

允许在此策略中进行操作的资源列表。指定与此策略中的操作相对应的资源。例如，您可以在指定`mqtt:publish`操作的策略中指定 MQTT 主题资源列表 (`mqtt:topic:`*`mqttTopic`*)。

您可以指定*通配符以允许访问所有资源。您不能使用*通配符来匹配部分资源标识符。例如，您可以指定**"resources": "*"**，但不能指定**"resources": "mqtt:clientId:*"**。

`statementDescription`

（可选）此政策声明的描述。

`certificates`

（可选）此核心设备的证书配置选项。该对象包含以下信息：

`serverCertificateValiditySeconds`

（可选）本地 MQTT 服务器证书过期的时间（以秒为单位）。您可以配置此选项以自定义客户端设备断开连接并重新连接到核心设备的频率。

此组件会在本地 MQTT 服务器证书到期前 24 小时对其进行轮换。MQTT 代理（例如 Moquett e MQTT 代理组件）会生成新证书并重新启动。发生这种情况时，所有连接到该核心设备的客户端设备都将断开连接。客户机设备可以在短时间后重新连接到核心设备。

默认：604800（7 天）

最小值：172800（2 天）

最大值：864000（10 天）

`performance`

（可选）此核心设备的性能配置选项。该对象包含以下信息：

`maxActiveAuthTokens`

（可选）活动客户端设备授权令牌的最大数量。您可以增加此数字，使更多的客户端设备能够连接到单核设备，而无需重新进行身份验证。

默认：2500

`cloudRequestQueueSize`

（可选）在此组件拒绝 AWS Cloud 请求之前要排队的最大请求数。

默认：100

`maxConcurrentCloudRequests`

（可选）要发送到的最大并发请求数 AWS Cloud。您可以增加此数字，以提高连接大量客户端设备的核心设备的身份验证性能。

默认：1

`certificateAuthority`

（可选）证书颁发机构配置选项，用您自己的中间证书颁发机构替换核心设备中间颁发机构。此对象包含以下信息。

证书网址

证书的位置。它可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书的 URI。

`certificateChainUri`

核心设备 CA 的证书链的位置。这应该是返回到您的根 CA 的完整证书链。它可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书链的 URI。

`privateKeyUri`

核心设备私钥的位置。这可以是文件系统 URI，也可以是指向存储在硬件安全模块中的证书私钥的 URI。

`security`

（可选）此核心设备的安全配置选项。此对象包含以下信息。

`clientDeviceTrustDurationMinutes`

在要求客户端设备向核心设备重新进行身份验证之前，可以信任客户端设备的身份验证信息的时长（以分钟为单位）。默认值是 1。

Example 示例：配置合并更新（使用限制性策略）

以下示例配置指定允许名称以开头的`MyClientDevice`客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
```

```
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 示例：配置合并更新（使用许可策略）

以下示例配置指定允许所有客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
```

```
          "selectionRule": "thingName: *",
          "policyName": "MyPermissivePolicy"
        }
      },
      "policies": {
        "MyPermissivePolicy": {
          "AllowAll": {
            "statementDescription": "Allow client devices to perform all actions.",
            "operations": [
              "*"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  }
```

v2.2.x

`deviceGroups`

设备组是指有权与核心设备连接和通信的客户端设备组。使用选择规则来识别客户端设备组，并定义为每个设备组指定权限的客户端设备授权策略。

该对象包含以下信息：

`formatVersion`

此配置对象的格式版本。

从以下选项中进行选择：

- `2021-03-05`

`definitions`

此核心设备的设备组。每个定义都指定了用于评估客户端设备是否为该组成员的选择规则。每个定义还指定了要应用于与选择规则相匹配的客户端设备的权限策略。如果一台客户端设备是多个设备组的成员，则该设备的权限由每个组的权限策略组成。

该对象包含以下信息：

groupNameKey

该设备组的名称。*groupNameKey*替换为可帮助您识别此设备组的名称。

该对象包含以下信息：

`selectionRule`

用于指定哪些客户端设备是该设备组成员的查询。当客户端设备连接时，核心设备会评估此选择规则，以确定该客户端设备是否为该设备组的成员。如果客户端设备是成员，则核心设备使用该设备组的策略来授权客户端设备的操作。

每条选择规则至少包含一个选择规则子句，即可以匹配客户端设备的单个表达式查询。选择规则使用的查询语法与 AWS IoT 舰队索引相同。有关选择规则语法的更多信息，请参阅《AWS IoT Core 开发人员指南》中的AWS IoT 舰队索引查询语法。

使用*通配符将多个客户端设备与一个选择规则子句进行匹配。您可以在事物名称末尾使用此通配符来匹配名称以您指定的字符串开头的客户端设备。您也可以使用此通配符来匹配所有客户端设备。

> **Note**
>
> 要选择包含冒号字符 (:) 的值，请使用反斜杠字符 (\\) 对冒号进行转义。在 JSON 等格式中，必须对反斜杠字符进行转义，因此在冒号字符之前输入两个反斜杠字符。例如，指定`thingName: MyTeam\\\\:ClientDevice1`选择名称为的事物`MyTeam:ClientDevice1`。

您可以指定以下选择器：

- `thingName`— 客户端设备的 AWS IoT 名称。

Example 选择规则示例

以下选择规则匹配名称为`MyClientDevice1`或的客户端设备`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 选择规则示例（使用通配符）

以下选择规则匹配名称以开头的客户端设备`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 选择规则示例（匹配所有设备）

以下选择规则匹配所有客户端设备。

```
thingName: *
```

`policyName`

适用于该设备组中的客户端设备的权限策略。指定您在`policies`对象中定义的策略的名称。

`policies`

连接到核心设备的客户端设备的客户端设备授权策略。每项授权策略都指定了一组操作以及客户端设备可以在其中执行这些操作的资源。

该对象包含以下信息：

`policyNameKey`

此授权策略的名称。*`policyNameKey`*替换为可帮助您识别此授权策略的名称。您可以使用此策略名称来定义哪个策略适用于设备组。

该对象包含以下信息：

`statementNameKey`

本政策声明的名称。*`statementNameKey`*替换为可帮助您识别此政策声明的名称。

该对象包含以下信息：

`operations`

允许使用此策略中的资源的操作列表。

您可以包括以下任何操作：

- `mqtt:connect`— 授予连接核心设备的权限。客户端设备必须具有此权限才能连接到核心设备。

此操作支持以下资源：

- mqtt:clientId:*deviceClientId*— 根据客户端设备用于连接核心设备的 MQTT 代理的客户端 ID 限制访问。*deviceClientId*替换为要使用的客户端 ID。

- mqtt:publish— 授予向主题发布 MQTT 消息的权限。

  此操作支持以下资源：

  - mqtt:topic:*mqttTopic*— 根据客户端设备发布消息的 MQTT 主题限制访问。将 *mqttTop* ic 替换为要使用的主题。

    此资源不支持 MQTT 主题通配符。

- mqtt:subscribe— 授予订阅 MQTT 主题过滤器以接收消息的权限。

  此操作支持以下资源：

  - mqtt:topicfilter:*mqttTopicFilter*— 根据客户端设备可以订阅消息的 MQTT 主题限制访问权限。*mqttTopicFilter*替换为要使用的主题筛选器。

    此资源支持+和 # MQTT 主题通配符。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的 MQTT 主题。

    客户端设备可以订阅您允许的确切主题过滤器。例如，如果您允许客户端设备订阅mqtt:topicfilter:client/+/status资源，则客户端设备可以订阅，client/+/status但不能订阅client/client1/status。

您可以指定*通配符以允许访问所有操作。

resources

允许在此策略中进行操作的资源列表。指定与此策略中的操作相对应的资源。例如，您可以在指定mqtt:publish操作的策略中指定 MQTT 主题资源列表 (mqtt:topic:*mqttTopic*)。

您可以指定*通配符以允许访问所有资源。您不能使用*通配符来匹配部分资源标识符。例如，您可以指定**"resources": "*"**，但不能指定**"resources": "mqtt:clientId:*"**。

statementDescription

（可选）此政策声明的描述。

`certificates`

（可选）此核心设备的证书配置选项。该对象包含以下信息：

`serverCertificateValiditySeconds`

（可选）本地 MQTT 服务器证书过期的时间（以秒为单位）。您可以配置此选项以自定义客户端设备断开连接并重新连接到核心设备的频率。

此组件会在本地 MQTT 服务器证书到期前 24 小时对其进行轮换。MQTT 代理（例如 Moquett e MQTT 代理组件）会生成新证书并重新启动。发生这种情况时，所有连接到该核心设备的客户端设备都将断开连接。客户机设备可以在短时间后重新连接到核心设备。

默认：`604800`（7 天）

最小值：`172800`（2 天）

最大值：`864000`（10 天）

`performance`

（可选）此核心设备的性能配置选项。该对象包含以下信息：

`maxActiveAuthTokens`

（可选）活动客户端设备授权令牌的最大数量。您可以增加此数字，使更多的客户端设备能够连接到单核设备，而无需重新进行身份验证。

默认：`2500`

`cloudRequestQueueSize`

（可选）在此组件拒绝 AWS Cloud 请求之前要排队的最大请求数。

默认：`100`

`maxConcurrentCloudRequests`

（可选）要发送到的最大并发请求数 AWS Cloud。您可以增加此数字，以提高连接大量客户端设备的核心设备的身份验证性能。

默认：`1`

Example 示例：配置合并更新（使用限制性策略）

以下示例配置指定允许名称以开头的`MyClientDevice`客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 示例：配置合并更新（使用许可策略）

以下示例配置指定允许所有客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.1.x

`deviceGroups`

设备组是指有权与核心设备连接和通信的客户端设备组。使用选择规则来识别客户端设备组，并定义为每个设备组指定权限的客户端设备授权策略。

该对象包含以下信息：

`formatVersion`

此配置对象的格式版本。

从以下选项中进行选择：

- `2021-03-05`

`definitions`

此核心设备的设备组。每个定义都指定了用于评估客户端设备是否为该组成员的选择规则。每个定义还指定了要应用于与选择规则相匹配的客户端设备的权限策略。如果一台客户端设备是多个设备组的成员，则该设备的权限由每个组的权限策略组成。

该对象包含以下信息：

`groupNameKey`

该设备组的名称。*`groupNameKey`*替换为可帮助您识别此设备组的名称。

该对象包含以下信息：

`selectionRule`

用于指定哪些客户端设备是该设备组成员的查询。当客户端设备连接时，核心设备会评估此选择规则，以确定该客户端设备是否为该设备组的成员。如果客户端设备是成员，则核心设备使用该设备组的策略来授权客户端设备的操作。

每条选择规则至少包含一个选择规则子句，即可以匹配客户端设备的单个表达式查询。选择规则使用的查询语法与 AWS IoT 舰队索引相同。有关选择规则语法的更多信息，请参阅《AWS IoT Core 开发人员指南》中的AWS IoT 舰队索引查询语法。

使用*通配符将多个客户端设备与一个选择规则子句进行匹配。您可以在事物名称末尾使用此通配符来匹配名称以您指定的字符串开头的客户端设备。您也可以使用此通配符来匹配所有客户端设备。

> **Note**
>
> 要选择包含冒号字符 (:) 的值，请使用反斜杠字符 (\\) 对冒号进行转义。在 JSON 等格式中，必须对反斜杠字符进行转义，因此在冒号字符之前输入两个反斜杠字符。例如，指定`thingName: MyTeam\\\\:ClientDevice1`选择名称为的事物`MyTeam:ClientDevice1`。

您可以指定以下选择器：

- `thingName`— 客户端设备的 AWS IoT 名称。

Example 选择规则示例

以下选择规则匹配名称为`MyClientDevice1`或的客户端设备`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 选择规则示例（使用通配符）

以下选择规则匹配名称以开头的客户端设备`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 选择规则示例（匹配所有设备）

以下选择规则匹配所有客户端设备。

```
thingName: *
```

`policyName`

适用于该设备组中的客户端设备的权限策略。指定您在`policies`对象中定义的策略的名称。

`policies`

连接到核心设备的客户端设备的客户端设备授权策略。每项授权策略都指定了一组操作以及客户端设备可以在其中执行这些操作的资源。

该对象包含以下信息：

`policyNameKey`

此授权策略的名称。*`policyNameKey`*替换为可帮助您识别此授权策略的名称。您可以使用此策略名称来定义哪个策略适用于设备组。

该对象包含以下信息：

`statementNameKey`

本政策声明的名称。*`statementNameKey`*替换为可帮助您识别此政策声明的名称。

该对象包含以下信息：

`operations`

允许使用此策略中的资源的操作列表。

您可以包括以下任何操作：

- `mqtt:connect`— 授予连接核心设备的权限。客户端设备必须具有此权限才能连接到核心设备。

  此操作支持以下资源：

  - `mqtt:clientId:`*`deviceClientId`*— 根据客户端设备用于连接核心设备的 MQTT 代理的客户端 ID 限制访问。*`deviceClientId`*替换为要使用的客户端 ID。

- `mqtt:publish`— 授予向主题发布 MQTT 消息的权限。

  此操作支持以下资源：

  - `mqtt:topic:`*`mqttTopic`*— 根据客户端设备发布消息的 MQTT 主题限制访问。将 *`mqttTop`* ic 替换为要使用的主题。

    此资源不支持 MQTT 主题通配符。

- `mqtt:subscribe`— 授予订阅 MQTT 主题过滤器以接收消息的权限。

  此操作支持以下资源：

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 根据客户端设备可以订阅消息的 MQTT 主题限制访问权限。*`mqttTopicFilter`*替换为要使用的主题筛选器。

    此资源支持+和 # MQTT 主题通配符。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的 [MQTT 主题]。

    客户端设备可以订阅您允许的确切主题过滤器。例如，如果您允许客户端设备订阅`mqtt:topicfilter:client/+/status`资源，则客户端设备可以订阅，`client/+/status`但不能订阅`client/client1/status`。

您可以指定`*`通配符以允许访问所有操作。

`resources`

允许在此策略中进行操作的资源列表。指定与此策略中的操作相对应的资源。例如，您可以在指定`mqtt:publish`操作的策略中指定 MQTT 主题资源列表 (`mqtt:topic:`*`mqttTopic`*)。

您可以指定`*`通配符以允许访问所有资源。您不能使用`*`通配符来匹配部分资源标识符。例如，您可以指定**`"resources": "*"`**，但不能指定**`"resources": "mqtt:clientId:*"`**。

`statementDescription`

（可选）此政策声明的描述。

`certificates`

（可选）此核心设备的证书配置选项。该对象包含以下信息：

`serverCertificateValiditySeconds`

（可选）本地 MQTT 服务器证书过期的时间（以秒为单位）。您可以配置此选项以自定义客户端设备断开连接并重新连接到核心设备的频率。

此组件会在本地 MQTT 服务器证书到期前 24 小时对其进行轮换。MQTT 代理（例如 Moquett e MQTT 代理组件）会生成新证书并重新启动。发生这种情况时，所有连接到该核心设备的客户端设备都将断开连接。客户机设备可以在短时间后重新连接到核心设备。

默认：`604800`（7 天）

最小值：`172800`（2 天）

最大值：`864000`（10 天）

Example 示例：配置合并更新（使用限制性策略）

以下示例配置指定允许名称以开头的`MyClientDevice`客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
```

```
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 示例：配置合并更新（使用许可策略）

以下示例配置指定允许所有客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
```

```
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.0.x

deviceGroups

设备组是指有权与核心设备连接和通信的客户端设备组。使用选择规则来识别客户端设备组，并定义为每个设备组指定权限的客户端设备授权策略。

该对象包含以下信息：

formatVersion

此配置对象的格式版本。

从以下选项中进行选择：

- 2021-03-05

definitions

此核心设备的设备组。每个定义都指定了用于评估客户端设备是否为该组成员的选择规则。每个定义还指定了要应用于与选择规则相匹配的客户端设备的权限策略。如果一台客户端设备是多个设备组的成员，则该设备的权限由每个组的权限策略组成。

该对象包含以下信息：

groupNameKey

该设备组的名称。*groupNameKey*替换为可帮助您识别此设备组的名称。

该对象包含以下信息：

`selectionRule`

用于指定哪些客户端设备是该设备组成员的查询。当客户端设备连接时，核心设备会评估此选择规则，以确定该客户端设备是否为该设备组的成员。如果客户端设备是成员，则核心设备使用该设备组的策略来授权客户端设备的操作。

每条选择规则至少包含一个选择规则子句，即可以匹配客户端设备的单个表达式查询。选择规则使用的查询语法与 AWS IoT 舰队索引相同。有关选择规则语法的更多信息，请参阅《AWS IoT Core 开发人员指南》中的AWS IoT 舰队索引查询语法。

使用*通配符将多个客户端设备与一个选择规则子句进行匹配。您可以在事物名称末尾使用此通配符来匹配名称以您指定的字符串开头的客户端设备。您也可以使用此通配符来匹配所有客户端设备。

> **Note**
>
> 要选择包含冒号字符 (:) 的值，请使用反斜杠字符 (\\) 对冒号进行转义。在 JSON 等格式中，必须对反斜杠字符进行转义，因此在冒号字符之前输入两个反斜杠字符。例如，指定`thingName: MyTeam\\\\:ClientDevice1`选择名称为的事物`MyTeam:ClientDevice1`。

您可以指定以下选择器：

- `thingName`— 客户端设备的 AWS IoT 名称。

Example 选择规则示例

以下选择规则匹配名称为`MyClientDevice1`或的客户端设备`MyClientDevice2`。

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example 选择规则示例（使用通配符）

以下选择规则匹配名称以开头的客户端设备`MyClientDevice`。

```
thingName: MyClientDevice*
```

Example 选择规则示例（匹配所有设备）

以下选择规则匹配所有客户端设备。

```
thingName: *
```

policyName

适用于该设备组中的客户端设备的权限策略。指定您在policies对象中定义的策略的名称。

policies

连接到核心设备的客户端设备的客户端设备的客户端设备授权策略。每项授权策略都指定了一组操作以及客户端设备可以在其中执行这些操作的资源。

该对象包含以下信息：

policyNameKey

此授权策略的名称。*policyNameKey*替换为可帮助您识别此授权策略的名称。您可以使用此策略名称来定义哪个策略适用于设备组。

该对象包含以下信息：

statementNameKey

本政策声明的名称。*statementNameKey*替换为可帮助您识别此政策声明的名称。

该对象包含以下信息：

operations

允许使用此策略中的资源的操作列表。

您可以包括以下任何操作：

- mqtt:connect— 授予连接核心设备的权限。客户端设备必须具有此权限才能连接到核心设备。

  此操作支持以下资源：

  - mqtt:clientId:*deviceClientId*— 根据客户端设备用于连接核心设备的 MQTT 代理的客户端 ID 限制访问。*deviceClientId*替换为要使用的客户端 ID。
- mqtt:publish— 授予向主题发布 MQTT 消息的权限。

  此操作支持以下资源：

  - mqtt:topic:*mqttTopic*— 根据客户端设备发布消息的 MQTT 主题限制访问。将 *mqttTop* ic 替换为要使用的主题。

此资源不支持 MQTT 主题通配符。

- `mqtt:subscribe`— 授予订阅 MQTT 主题过滤器以接收消息的权限。

  此操作支持以下资源：

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— 根据客户端设备可以订阅消息的 MQTT 主题限制访问权限。*`mqttTopicFilter`*替换为要使用的主题筛选器。

    此资源支持+和 # MQTT 主题通配符。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的 MQTT 主题。

    客户端设备可以订阅您允许的确切主题过滤器。例如，如果您允许客户端设备订阅`mqtt:topicfilter:client/+/status`资源，则客户端设备可以订阅，`client/+/status`但不能订阅`client/client1/status`。

您可以指定`*`通配符以允许访问所有操作。

`resources`

允许在此策略中进行操作的资源列表。指定与此策略中的操作相对应的资源。例如，您可以在指定`mqtt:publish`操作的策略中指定 MQTT 主题资源列表 (`mqtt:topic:`*`mqttTopic`*)。

您可以指定`*`通配符以允许访问所有资源。您不能使用`*`通配符来匹配部分资源标识符。例如，您可以指定**`"resources": "*"`**，但不能指定**`"resources": "mqtt:clientId:*"`**。

`statementDescription`

（可选）此政策声明的描述。

Example 示例：配置合并更新（使用限制性策略）

以下示例配置指定允许名称以开头的`MyClientDevice`客户端设备连接和发布/订阅所有主题。

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
```

```
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example 示例：配置合并更新（使用许可策略）

以下示例配置指定允许所有客户端设备连接和发布/订阅所有主题。

```
{
```

```
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.4.5	新功能 添加了对使用参数选择事物名称的通配符前缀的支持。selectionRule 错误修复和改进 修复了在某些情况下无法使用新的连接信息更新证书的问题。
2.4.4	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.4.3	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.4.2	新功能 添加新的startupTimeoutSeconds 配置选项。
2.4.1	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.4.0	新功能 • 添加了对客户端设备身份验证的支持，以发布将由遥测代理发布的操作指标。 错误修复和改进 • 修复了客户端设备身份验证需要 10 秒以上的时间来验证客户端设备身份的问题。

版本	更改
	• 其他小修复和改进。
2.3.2	错误修复和改进 • 添加了对缓存主机名信息的支持，以便组件在离线时重新启动时正确生成证书主题。
2.3.1	错误修复和改进 • 修复了内存泄漏问题。
2.3.0	⚠ Warning 此版本不再可用。此版本的改进将在此组件的更高版本中提供。 新功能 • 增加了对客户端设备的离线身份验证的支持，以便当核心设备未连接到 Internet 时，它们可以继续连接到核心设备。 • 添加对客户提供的证书颁发机构的支持，核心设备将其用作生成 MQTT 代理证书的根证书。
2.2.3	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.2.2	错误修复和改进 • 修复了在某些情况下，本地 MQTT 服务器证书的轮换频率高于预期的问题。
2.2.1	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.2.0	新功能 • 添加了对自定义组件的支持，以调用进程间通信 (IPC) 操作来对客户端设备进行身份验证和授权。例如，您可以在自定义 MQTT 代理组件中使用这些操作。有关更多信息，请参阅 IPC：对客户端设备进行身份验证和授权。 • 添加`maxActiveAuthTokens` `cloudQueueSize` 、和`threadPoolSize` 选项，您可以配置这些选项以调整此组件的性能。

版本	更改
2.1.0	新功能 • 添加可配置为自定义 MQTT 代理服务器证书何时到期的`serverCertificateValiditySeconds` 选项。您可以将服务器证书配置为在 2 到 10 天后过期。 错误修复和改进 • 修复了此组件如何处理配置重置更新的问题。 • 修复了在某些情况下，本地 MQTT 服务器证书的轮换频率高于预期的问题。 要应用此修复程序，您还必须使用 M oquette MQ TT 代理组件的 v2.1.0 或更高版本。 • 改进了此组件在轮换证书时记录的消息。 • Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.0.4	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.3	错误修复和改进 • 现在，如果您轮换核心设备的私钥，凭据会刷新。 • 更新以使日志消息更加清晰。
2.0.2	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.1	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.0	初始版本。

CloudWatch 指标

亚马逊 CloudWatch 指标组件 (`aws.greengrass.Cloudwatch`) 将来自Greengrass核心设备的自定义指标发布到亚马逊。 CloudWatch该组件使组件能够发布 CloudWatch 指标，您可以使用这些指标来监控和分析 Greengrass 核心设备的环境。有关更多信息，请参阅亚马逊 CloudWatch 用户指南中的使用亚马逊 CloudWatch指标。

要使用此组件发布 CloudWatch 指标，请向该组件订阅的主题发布一条消息。默认情况下，此组件订阅`cloudwatch/metric/put`本地发布/ 订阅主题。部署此组件时，您可以指定其他主题，包括 AWS IoT Core MQTT 主题。

该组件对位于同一命名空间中的指标进行批处理，并定期将其发布到 CloudWatch 。

Note

此组件提供的功能与 AWS IoT Greengrass V1 中的 CloudWatch 指标连接器类似。有关更多信息，请参阅 AWS IoT Greengrass V1 开发者指南中的CloudWatch 指标连接器。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 输入数据
- 输出数据
- 许可证
- 本地日志文件
- 更改日志
- 另请参阅

版本

此组件有以下版本：

- 3.1.x
- 3.0.x
- 2.1.x
- 2.0.x

有关每个版本组件变更的信息，请参阅变更日志。

类型

v3.x

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

v2.x

此组件是一个 Lambda 组件 () `aws.greengrass.lambda`。Greengrass 核心使用 Lambda 启动器组件运行此组件的 Lambda 函数。

有关更多信息，请参阅组件类型。

操作系统

v3.x

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

v2.x

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

3.x

- Python 版本 3.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- Greengrass 设备角色必须允许`cloudwatch:PutMetricData`该操作，如以下示例 IAM 策略所示。

```
{
```

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

有关更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的亚马逊 CloudWatch权限参考。

2.x

- 您的核心设备必须满足运行 Lambda 函数的要求。如果您希望核心设备运行容器化的 Lambda 函数，则该设备必须满足要求。有关更多信息，请参阅Lambda 函数要求。
- Python 版本 3.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- Greengrass 设备角色必须允许`cloudwatch:PutMetricData`该操作，如以下示例 IAM 策略所示。

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

有关更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的亚马逊 CloudWatch权限参考。

- 要接收此组件的输出数据，在部署此组件时，必须合并旧版订阅路由器组件 (`aws.greengrass.LegacySubscriptionRouter`) 的以下配置更新。此配置指定此组件发布响应的主题。

Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-cloudwatch": {
      "id": "aws-greengrass-cloudwatch",
      "source": "component:aws.greengrass.Cloudwatch",
      "subject": "cloudwatch/metric/put/status",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-cloudwatch": {
      "id": "aws-greengrass-cloudwatch",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
cloudwatch:version",
      "subject": "cloudwatch/metric/put/status",
      "target": "cloud"
    }
  }
}
```

- 将*##*替换为您 AWS 区域 使用的区域。
- 将*##*替换为该组件运行的 Lambda 函数的版本。要查找 Lambda 函数版本，您必须查看要部署的此组件版本的配方。在AWS IoT Greengrass 控制台中打开此组件的详细信息页面，查找 Lambda 函数键值对。此键值对包含 Lambda 函数的名称和版本。

> **Important**
>
> 每次部署此组件时，都必须更新旧版订阅路由器上的 Lambda 函数版本。这样可以确保您为部署的组件版本使用正确的 Lambda 函数版本。

有关更多信息，请参阅创建部署。

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
monitoring. *region*.amazonaws.com	443	是	上传 CloudWatch 指标。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

3.0.0 - 3.1.0

下表列出了此组件的 3.0.0 到 3.1.0 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <3.0.0	软性
代币兑换服务	>=0.0.0	硬性

2.1.2 and 2.1.3

下表列出了此组件版本 2.1.2 和 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	硬性
Lambda 启动器	^2.0.0	硬性

依赖关系	兼容版本	依赖关系类型
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.8 - 2.1.0

下表列出了此组件版本 2.0.8 到 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.7

下表列出了此组件版本 2.0.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.6

下表列出了此组件版本 2.0.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.5

下表列出了此组件版本 2.0.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.4

下表列出了此组件版本 2.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.3

下表列出了此组件版本 2.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	硬性
Lambda 启动器	>=1.0.0	硬性
Lambda 运行时	>=1.0.0	软性
代币兑换服务	>=1.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

v3.x

`PublishInterval`

（可选）在组件发布给定命名空间的批处理指标之前等待的最大秒数。要将组件配置为在收到指标时发布指标（这意味着不进行批处理），请指定`0`。

该组件在同一命名空间中收到 20 个指标之后或在您指定的间隔之后发布到。 CloudWatch

> **Note**
>
> 该组件未指定事件的发布顺序。

此值最多可为 900 秒。

默认值：10 秒

`MaxMetricsToRetain`

（可选）在组件用较新的指标替换所有命名空间的指标之前，要在内存中保存的最大指标数。

当核心设备没有连接到互联网时，此限制适用，因此组件会缓冲指标以供日后发布。当缓冲区已满时，该组件会将最旧的指标替换为较新的指标。给定命名空间中的指标仅替换同一命名空间中的指标。

> **Note**
>
> 如果组件的主机进程中断，则该组件不会保存指标。例如，这可能发生在部署期间或核心设备重启时。

此值必须至少为 2,000 个指标。

默认值：5,000 个指标

`InputTopic`

（可选）组件订阅以接收消息的主题。如果您`true`为指定`PubSubToIoTCore`，则可以在本主题中使用 MQTT 通配符（+ 和 #）。

默认：`cloudwatch/metric/put`

`OutputTopic`

（可选）组件向其发布状态响应的主题。

默认：`cloudwatch/metric/put/status`

`PubSubToIoTCore`

（可选）定义是否发布和订阅 AWS IoT Core MQTT 主题的字符串值。支持的值为 `true` 和 `false`。

默认：`false`

`UseInstaller`

（可选）布尔值，用于定义是否使用此组件中的安装程序脚本来安装此组件的 SDK 依赖项。

`false`如果您想使用自定义脚本来安装依赖项，或者想要在预构建的 Linux 映像中包含运行时依赖关系，请将此值设置为。要使用此组件，必须安装以下库（包括所有依赖项），并使其可供默认 Greengrass 系统用户使用。

- AWS IoT Device SDK 适用于 Python 的 v2
- AWS SDK for Python (Boto3)

默认：`true`

`PublishRegion`

（可选） AWS 区域 要向其发布 CloudWatch 指标的。此值将覆盖核心设备的默认区域。只有跨区域指标才需要此参数。

`accessControl`

（可选）包含授权策略的对象，该策略允许组件发布和订阅指定主题。如果您为`InputTopic`和指定了自定义值`OutputTopic`，则必须更新此对象中的资源值。

默认值：

```
{
  "aws.greengrass.ipc.pubsub": {
    "aws.greengrass.Cloudwatch:pubsub:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
```

```
    "aws.greengrass.Cloudwatch:pubsub:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  },
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.Cloudwatch:mqttproxy:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
    "aws.greengrass.Cloudwatch:mqttproxy:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  }
}
```

Example 示例：配置合并更新

```
{
  "PublishInterval": 0,
  "PubSubToIoTCore": true
}
```

v2.x

Note

此组件的默认配置包括 Lambda 函数参数。我们建议您仅编辑以下参数，以便在您的设备上配置此组件。

`lambdaParams`

包含此组件的 Lambda 函数参数的对象。该对象包含以下信息：

`EnvironmentVariables`

一个包含 Lambda 函数参数的对象。该对象包含以下信息：

`PUBLISH_INTERVAL`

（可选）在组件发布给定命名空间的批处理指标之前等待的最大秒数。要将组件配置为在收到指标时发布指标（这意味着不进行批处理），请指定`0`。

该组件在同一命名空间中收到 20 个指标之后或在您指定的间隔之后发布到。CloudWatch

Note

该组件不保证事件的发布顺序。

此值最多可为 900 秒。

默认值：10 秒

`MAX_METRICS_TO_RETAIN`

（可选）在组件用较新的指标替换所有命名空间的指标之前，要在内存中保存的最大指标数。

当核心设备没有连接到互联网时，此限制适用，因此组件会缓冲指标以供日后发布。当缓冲区已满时，该组件会将最旧的指标替换为较新的指标。给定命名空间中的指标仅替换同一命名空间中的指标。

> **Note**
>
> 如果组件的主机进程中断，则该组件不会保存指标。例如，这可能发生在部署期间或核心设备重启时。

此值必须至少为 2,000 个指标。

默认值：5,000 个指标

`PUBLISH_REGION`

（可选）AWS 区域 要向其发布 CloudWatch 指标的。此值将覆盖核心设备的默认区域。只有跨区域指标才需要此参数。

`containerMode`

（可选）此组件的容器化模式。从以下选项中进行选择：

- `NoContainer`— 该组件不在隔离的运行时环境中运行。
- `GreengrassContainer`— 该组件在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

默认：`GreengrassContainer`

`containerParams`

（可选）包含此组件容器参数的对象。如果您为指定`GreengrassContainer`，则该组件将使用这些参数`containerMode`。

该对象包含以下信息：

`memorySize`

（可选）要分配给组件的内存量（以千字节为单位）。

默认为 64 MB (65,535 KB)。

`pubsubTopics`

（可选）一个对象，其中包含组件订阅以接收消息的主题。您可以指定每个主题以及该组件是订阅来自的 MQTT 主题 AWS IoT Core 还是本地发布/订阅主题。

该对象包含以下信息：

`0`— 这是字符串形式的数组索引。

包含以下信息的对象：

`type`

（可选）此组件用于订阅消息的发布/订阅消息的类型。从以下选项中进行选择：

- PUB_SUB – 订阅本地发布/订阅消息。如果选择此选项，则主题不能包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅本地消息。
- `IOT_CORE`— 订阅 AWS IoT Core MQTT 消息。如果选择此选项，则主题可以包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

默认：PUB_SUB

`topic`

（可选）组件订阅以接收消息的主题。如果您`IotCore`为指定`type`，则可以在本主题中使用 MQTT 通配符（+和#）。

Example 示例：配置合并更新（容器模式）

```
{
  "containerMode": "GreengrassContainer"
}
```

Example 示例：配置合并更新（无容器模式）

```
{
  "containerMode": "NoContainer"
}
```

输入数据

此组件接受有关以下主题的指标并将这些指标发布到 CloudWatch。默认情况下，此组件订阅本地发布/订阅消息。有关如何从您的自定义组件向该组件发布消息的更多信息，请参阅发布/订阅本地消息。

从组件版本 v3.0.0 开始，您可以选择通过将配置参数设置为，将此组件配置为订阅 MQTT 主题。PubSubToIoTCore true有关在自定义组件中向 MQTT 主题发布消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

默认主题：cloudwatch/metric/put

该消息接受以下属性。输入消息必须采用 JSON 格式。

request

此消息中的指标。

请求对象包含要发布到 CloudWatch 的指标数据。指标值必须符合PutMetricData操作规范。

类型：其中object包含以下信息：

namespace

此请求中指标数据的用户定义命名空间。 CloudWatch 使用命名空间作为指标数据点的容器。

> **Note**
>
> 不能指定以保留字符串 AWS/ 开头的命名空间。

类型：string

有效模式：[^:].*

metricData

指标的数据。

类型：其中object包含以下信息：

metricName

指标的名称。

类型：string

value

指标的值。

Note

CloudWatch 拒绝太小或太大的值。该值必须介于8.515920e-109和1.174271e+108（基数 10）或2e-360和2e360（基数 2）之间。 CloudWatch 不支持特殊值NaN，例如+Infinity、和-Infinity。

类型：double

dimensions

（可选）指标的维度。维度提供有关指标及其数据的附加信息。指标最多可定义 10 个维度。

该组件自动包含一个名为的维度coreName，其中值是核心设备的名称。

类型：array每个对象都包含以下信息：

name

（可选）维度名称。

类型：string

value

（可选）维度值。

类型：string

timestamp

（可选）接收指标数据的时间，以 Unix 纪元时间为单位的秒表示。

默认为组件收到消息的时间。

类型：double

Note

如果您在版本 2.0.3 和 2.0.7 之间使用此组件，我们建议您在从单个来源发送多个指标时分别检索每个指标的时间戳。不要使用变量来存储时间戳。

`unit`

（可选）指标的单位。

类型：`string`

有效值：`Seconds`、`Microseconds`、`Milliseconds`、`Bytes`、`Kilobytes`、`Megabytes`、`Gigaby`
`Second`、`Kilobytes/Second`、`Megabytes/Second`、`Gigabytes/Second`、`Terabytes/Second`、`Bits/Second`、`Kilobits/Second`、`Megabits/Second`、`Gigabits/Second`、`Terabits/Second`、`Count/Second`、`None`

默认值为 None。

> **Note**
>
> 适用于 CloudWatch `PutMetricData` API 的所有配额都适用于您使用此组件发布的指标。以下配额尤其重要：
>
> - API 有效负载上限为 40 KB
> - 每个 API 请求的 20 个指标
> - `PutMetricData` API 的每秒 150 个事务 (TPS)
>
> 有关更多信息，请参阅《CloudWatch 用户指南》中的CloudWatch 服务配额。

Example 示例输入

```
{
  "request": {
    "namespace": "Greengrass",
    "metricData": {
      "metricName": "latency",
      "dimensions": [
        {
          "name": "hostname",
          "value": "test_hostname"
        }
      ],
```

```
      "timestamp": 1539027324,
      "value": 123.0,
      "unit": "Seconds"
    }
  }
}
```

输出数据

默认情况下，此组件将响应作为以下本地发布/订阅主题的输出数据发布。有关如何在您的自定义组件中订阅有关此主题的消息的更多信息，请参阅发布/订阅本地消息。

您可以选择将配置参数设置为，将此组件PubSubToIoTCore配置为发布到 MQTT 主题。true有关在自定义组件中订阅有关 MQTT 主题的消息的更多信息，请参阅。发布/订阅 AWS IoT Core MQTT 消息

> **Note**
>
> 默认情况下，组件版本 2.0.x 将响应作为 MQTT 主题的输出数据发布。您必须在传统订阅路由器组件的配置subject中将主题指定为。

默认主题：cloudwatch/metric/put/status

Example 示例输出：成功

响应包括指标数据的命名空间和 CloudWatch 响应中的RequestId字段。

```
{
  "response": {
    "cloudwatch_rid": "70573243-d723-11e8-b095-75ff2EXAMPLE",
    "namespace": "Greengrass",
    "status": "success"
  }
}
```

Example 示例输出：失败

```
{
  "response" : {
    "namespace": "Greengrass",
```

```
    "error": "InvalidInputException",
    "error_message": "cw metric is invalid",
    "status": "fail"
  }
}
```

> **Note**
>
> 如果组件检测到可以重试的错误（例如连接错误），则会在下一批中重试发布。

许可证

此组件包括以下第三方软件/许可：

- AWS SDK for Python (Boto3)/Apache 许可证 2.0
- botocore/Apache 许可证 2.0
- dateutil/PSF 许可证
- docutils/BSD 许可证，GNU 通用公共许可证 (GPL)，Python 软件基金会许可证，公共领域
- jmespath/MIT 许可证
- s3transfer/Apache 许可证 2.0
- urllib3/MIT 许可证

该组件是根据 Greengrass 核心软件许可协议发布的。

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.greengrass.Cloudwatch.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.Cloudwatch.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

v3.x

版本	更改
3.1.0	错误修复和改进 • 添加对 HTTPS 网络代理配置的支持。有关更多信息，请参阅 通过端口 443 或网络代理进行连接 和 使核心设备能够信任 HTTPS 代理。
3.0.0	此版本的 CloudWatch 指标组件需要的配置参数与版本 2.x 不同。如果您使用版本 2.x 的非默认配置，并且想要从 v2.x 升级到 v3.x，则必须更新该组件的配置。有关更多信息，请参阅CloudWatch指标组件配置。 新功能 • 增加了对运行 Windows 的核心设备的支持。 • 将组件类型从 Lambda 组件更改为通用组件。此组件现在不再依赖旧版订阅路由器组件来创建订阅。 • 添加新的`InputTopic` 配置参数以指定组件订阅以接收消息的主题。 • 添加新的`OutputTopic` 配置参数以指定组件向其发布状态响应的主题。

版本	更改
	• 添加新的`PubSubToIoTCore` 配置参数以指定是否发布和订阅 AWS IoT Core MQTT 主题。 • 添加新的`UseInstaller` 配置参数，允许您选择禁用安装组件依赖项的安装脚本。 错误修复和改进 增加了对输入数据中重复时间戳的支持。

v2.x

版本	更改
2.1.3	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.0	新功能 • 添加对 HTTPS 网络代理配置的支持。有关更多信息，请参阅 通过端口 443 或网络代理进行连接 和 使核心设备能够信任 HTTPS 代理。
2.0.8	错误修复和改进 • 增加了对输入数据中重复时间戳的支持。 • Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.7	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.6	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.5	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.0.4	Greengrass nucleus 版本 2.1.0 版本的版本已更新。
2.0.3	初始版本。

另请参阅

- 使用亚马逊 CloudWatch 用户指南中的亚马逊 CloudWatch指标
- PutMetricData在 Amazon CloudWatch API 参考中

AWS IoT Device Defender

组 AWS IoT Device Defender 件 (`aws.greengrass.DeviceDefender`) 会通知管理员有关 Greengrass 核心设备状态的变化。这有助于识别可能指示受损设备的异常行为。有关更多信息，请参阅 AWS IoT Core 开发人员指南中的 AWS IoT Device Defender。

该组件读取核心设备上的系统指标。然后，它将指标发布到 AWS IoT Device Defender。有关如何阅读和解释此组件报告的指标的更多信息，请参阅AWS IoT Core 开发人员指南中的设备指标文档规范。

> **Note**
>
> 此组件提供的功能与中的 Device Defender 连接器类似 AWS IoT Greengrass V1。有关更多信息，请参阅《AWS IoT Greengrass V1 开发者指南》中的 “设备防御者连接器”。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 输入数据
- 输出数据
- 本地日志文件
- 许可证
- 更改日志

版本

此组件有以下版本：

- 3.1.x
- 3.0.x
- 2.0.x

有关每个版本组件变更的信息，请参阅变更日志。

类型

v3.x

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

v2.x

此组件是一个 Lambda 组件 () `aws.greengrass.lambda`。Greengrass 核心使用 Lambda 启动器组件运行此组件的 Lambda 函数。

有关更多信息，请参阅组件类型。

操作系统

v3.x

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

v2.x

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

v3.x

- Python 版本 3.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- AWS IoT Device Defender 配置为使用 “检测” 功能来监视违规行为。有关更多信息，请参阅 AWS IoT Core 开发人员指南中的检测。

v2.x

- 您的核心设备必须满足运行 Lambda 函数的要求。如果您希望核心设备运行容器化的 Lambda 函数，则该设备必须满足要求。有关更多信息，请参阅Lambda 函数要求。
- Python 版本 3.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- AWS IoT Device Defender 配置为使用 “检测” 功能来监视违规行为。有关更多信息，请参阅 AWS IoT Core 开发人员指南中的检测。
- 安装在核心设备上的 psutil 库。版本 5.7.0 是经验证可与该组件配合使用的最新版本。
- 安装在核心设备上的 cbor 库。版本 1.0.0 是经验证可与该组件配合使用的最新版本。
- 要接收此组件的输出数据，在部署此组件时，必须合并旧版订阅路由器组件 (`aws.greengrass.LegacySubscriptionRouter`) 的以下配置更新。此配置指定此组件发布响应的主题。

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-device-defender": {
        "id": "aws-greengrass-device-defender",
        "source": "component:aws.greengrass.DeviceDefender",
        "subject": "$aws/things/+/defender/metrics/json",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-device-defender": {
        "id": "aws-greengrass-device-defender",
  ```

```
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-device-defender:version",
        "subject": "$aws/things/+/defender/metrics/json",
        "target": "cloud"
      }
    }
}
```

- 将##替换为您 AWS 区域 使用的区域。
- 将##替换为该组件运行的 Lambda 函数的版本。要查找 Lambda 函数版本，您必须查看要部署的此组件版本的配方。在AWS IoT Greengrass 控制台中打开此组件的详细信息页面，查找 Lambda 函数键值对。此键值对包含 Lambda 函数的名称和版本。

> **Important**
>
> 每次部署此组件时，都必须更新旧版订阅路由器上的 Lambda 函数版本。这样可以确保您为部署的组件版本使用正确的 Lambda 函数版本。

有关更多信息，请参阅创建部署。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

3.1.1

下表列出了此组件版本 3.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <3.0.0	软性
代币兑换服务	>=0.0.0	硬性

3.0.0 - 3.0.2

下表列出了此组件的 3.0.0 到 3.0.2 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <3.0.0	软性
代币兑换服务	>=0.0.0	硬性

2.0.10 and 2.0.11

下表列出了此组件版本 2.0.10 和 2.0.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.9

下表列出了此组件版本 2.0.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.8

下表列出了此组件版本 2.0.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.7

下表列出了此组件版本 2.0.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.6

下表列出了此组件版本 2.0.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性

依赖关系	兼容版本	依赖关系类型
代币兑换服务	^2.0.0	硬性

2.0.5

下表列出了此组件版本 2.0.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.4

下表列出了此组件版本 2.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.3

下表列出了此组件版本 2.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	硬性
Lambda 启动器	>=1.0.0	硬性
Lambda 运行时	>=1.0.0	软性
代币兑换服务	>=1.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

v3.x

`PublishRetryCount`

重试发布的次数。此功能在 3.1.1 版本中可用。

最小值为 0。

最大值为 72。

默认：5

`SampleIntervalSeconds`

（可选）组件收集和报告指标的每个周期之间的时间（以秒为单位）。

最小值为 300 秒 (5 分钟)。

默认值：300 秒

`UseInstaller`

（可选）布尔值，用于定义是否使用此组件中的安装程序脚本来安装此组件的依赖项。

`false`如果您想使用自定义脚本来安装依赖项，或者想要在预构建的 Linux 映像中包含运行时依赖关系，请将此值设置为。要使用此组件，必须安装以下库（包括所有依赖项），并使其可供默认 Greengrass 系统用户使用。

- AWS IoT Device SDK 适用于 Python 的 v2
- cbor 图书馆。版本 1.0.0 是经验证可与该组件配合使用的最新版本。
- psutil 库。版本 5.7.0 是经验证可与该组件配合使用的最新版本。

> **Note**
>
> 如果您在配置为使用 HTTPS 代理的核心设备上使用此组件的 3.0.0 或 3.0.1 版，则必须将此值设置为。`false`在此组件的这些版本中，安装程序脚本不支持在 HTTPS 代理后面的操作。

默认：`true`

v2.x

> **Note**
>
> 此组件的默认配置包括 Lambda 函数参数。我们建议您仅编辑以下参数，以便在您的设备上配置此组件。

`lambdaParams`

包含此组件的 Lambda 函数参数的对象。该对象包含以下信息：

`EnvironmentVariables`

一个包含 Lambda 函数参数的对象。该对象包含以下信息：

`PROCFS_PATH`

（可选）`/proc`文件夹的路径。

- 要在容器中运行此组件，请使用默认值`/host-proc`。默认情况下，该组件在容器中运行。
- 要在无容器模式下运行此组件，请`/proc`为此参数指定。

默认值：`/host-proc`。这是该组件在容器中安装`/proc`文件夹的默认路径。

Note

此组件对该文件夹具有只读访问权限。

`SAMPLE_INTERVAL_SECONDS`

（可选）组件收集和报告指标的每个周期之间的时间（以秒为单位）。

最小值为 300 秒 (5 分钟)。

默认值：300 秒

`containerMode`

（可选）此组件的容器化模式。从以下选项中进行选择：

- `GreengrassContainer`— 组件在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。
- `NoContainer`— 该组件不在隔离的运行时环境中运行。

  如果指定此选项，则必须`/proc`为`PROCFS_PATH`环境变量参数指定。

默认：`GreengrassContainer`

`containerParams`

（可选）包含此组件容器参数的对象。如果您为指定`GreengrassContainer`，则组件将使用这些参数`containerMode`。

该对象包含以下信息：

`memorySize`

（可选）要分配给组件的内存量（以千字节为单位）。

默认为 50,000 KB。

`pubsubTopics`

（可选）一个对象，其中包含组件订阅以接收消息的主题。您可以指定每个主题以及该组件是订阅来自的 MQTT 主题 AWS IoT Core 还是本地发布/订阅主题。

该对象包含以下信息：

`0`— 这是字符串形式的数组索引。

包含以下信息的对象：

`type`

（可选）此组件用于订阅消息的发布/订阅消息的类型。从以下选项中进行选择：

- PUB_SUB – 订阅本地发布/订阅消息。如果选择此选项，则主题不能包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅本地消息。
- `IOT_CORE`— 订阅 AWS IoT Core MQTT 消息。如果选择此选项，则主题可以包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

默认：PUB_SUB

`topic`

（可选）组件订阅以接收消息的主题。如果您`IotCore`为指定`type`，则可以在本主题中使用 MQTT 通配符（`+`和`#`）。

Example 示例：配置合并更新（容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/host_proc"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example 示例：配置合并更新（无容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/proc"
    }
  },
  "containerMode": "NoContainer"
```

```
}
```

输入数据

此组件不接受消息作为输入数据。

输出数据

此组件将安全指标发布到以下保留主题 AWS IoT Device Defender。在发布指标时 *coreDeviceName*，此组件将替换为核心设备的名称。

主题 (AWS IoT Core MQTT)：`$aws/things/coreDeviceName/defender/metrics/json`

Example 示例输出


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


有关此组件报告的指标的更多信息，请参阅《AWS IoT Core 开发人员指南》中的设备指标文档规范。

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.greengrass.DeviceDefender.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DeviceDefender.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log -Tail 10 -Wait
  ```

许可证

该组件是根据 Greengrass 核心软件许可协议发布的。

更改日志

下表描述了该组件的每个版本中的更改。

v3.x

版本	更改
3.1.1	错误修复和改进 • 添加网络中断后连接无法恢复时客户端连接的重试次数。 • 为发布指标添加可配置的重试功能。
3.1.0	错误修复和改进 • 添加对 HTTPS 网络代理配置的支持。有关更多信息，请参阅 通过端口 443 或网络代理进行连接 和 使核心设备能够信任 HTTPS 代理。

版本	更改
3.0.1	修复了组件如何计算指标增量值的问题。
3.0.0	Warning 此版本不再可用。此版本的改进将在此组件的更高版本中提供。 初始版本。

v2.x

版本	更改
2.0.11	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.0.10	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.0.9	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.0.8	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.7	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.6	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.5	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.0.4	Greengrass nucleus 版本 2.1.0 版本的版本已更新。
2.0.3	初始版本。

磁盘后台处理程序

磁盘假脱机组件 (`aws.greengrass.DiskSpooler`) 为从 Greengrass 核心设备后台处理的消息提供了永久存储选项。 AWS IoT Core此组件会将这些出站消息存储在磁盘上。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 使用量
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 1.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅监控AWS IoT Greengrass日志。

有关更多信息，请参阅组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- storageType应设置Disk为使用此组件。你可以在 Greengrass 核配置中进行设置。
- maxSizeInBytes不得将其配置为大于设备上的可用空间。你可以在 Greengrass 核配置中进行设置。
- 支持在 VPC 中运行磁盘后台处理程序组件。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

1.0.1 – 1.0.3

下表列出了此组件的 1.0.1 到 1.0.3 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.11.0 <2.13.0	硬性

1.0.0

下表列出了此组件版本 1.0.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.11.0 <2.12.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

使用量

要使用磁盘后台处理程序组件，aws.greengrass.DiskSpooler必须部署。

要配置和使用此组件，必须将设置pluginName为aws.greengrass.DiskSpooler。

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
1.0.3	错误修复和改进 通过重复使用数据库连接来提高性能。
1.0.2	错误修复和改进 修复了在某些情况下无法保留 MQTT 消息格式字段的问题。

版本	更改
1.0.1	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
1.0.0	初始版本。

Docker 应用程序管理器

Docker 应用程序管理器组件 (`aws.greengrass.DockerApplicationManager`) 允许AWS IoT Greengrass从托管在亚马逊 Elastic Container Registry (Amazon ECR) 上的公共映像注册表和私有注册表下载 Docker 镜像。它还允许自动管理证书AWS IoT Greengrass，以便安全地从 Amazon ECR 中的私有存储库下载图像。

开发运行 Docker 容器的自定义组件时，请将 Docker 应用程序管理器作为依赖项包含在组件中下载指定为构件的 Docker 镜像。有关更多信息，请参阅 运行 Docker 容器。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志
- 另请参阅

版本

此组件有以下版本：

- 2.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 安装在 Greengr@@ ass 核心设备上的 Docker E ngine 1.9.1 或更高版本。版本 20.10 是经验证可与AWS IoT Greengrass核心软件配合使用的最新版本。在部署运行 Docker 容器的组件之前，必须直接在核心设备上安装 Docker。
- 在部署此组件之前，Docker 守护程序已启动并在核心设备上运行。
- 存储在以下支持的图像源之一中的 Docker 镜像：
  - Amazon Elastic Container Registry (Amazon ECR) 中的公共和私有图像存储库
  - 公共 Docker Hub 存储库
  - 公共 Docker 可信注册表
- Docker 镜像作为工件包含在您的自定义 Docker 容器组件中。使用以下 URI 格式来指定您的 Docker 镜像：
  - 亚马逊 ECR 的私有图片：`docker:account-id.dkr.ecr.region.amazonaws.com/repository/image[:tag|@digest]`
  - 公开 Amazon ECR 图片：`docker:public.ecr.aws/repository/image[:tag|@digest]`
  - 公共 Docker Hub 镜像：`docker:name[:tag|@digest]`

  有关更多信息，请参阅 运行 Docker 容器。

Note

如果您没有在映像的工件 URI 中指定图像标签或图像摘要，则在部署自定义 Docker 容器组件时，Docker 应用程序管理器会提取该镜像的最新可用版本。为确保所有核心设备都运行相同版本的图像，我们建议您在构件 URI 中包含图像标签或图像摘要。

- 运行 Docker 容器组件的系统用户必须具有根或管理员权限，或者您必须将 Docker 配置为以非根用户或非管理员用户身份运行。
  - 在 Linux 设备上，您可以将用户添加到`docker`群组中，无需用户即可调用`docker`命令`sudo`。
  - 在 Windows 设备上，无需管理员权限即可将用户添加到`docker-users`群组中以调用`docker`命令。

  Linux or Unix

  要将或用于运行 Docker 容器组件的非 root 用户添加到`ggc_user``docker`群组，请运行以下命令。

  ```
  sudo usermod -aG docker ggc_user
  ```

  有关更多信息，请参阅以非 root 用户身份管理 Docker。

  Windows Command Prompt (CMD)

  要将或您用来运行 Docker 容器组件的用户添加到`ggc_user`该`docker-users`组，请以管理员身份运行以下命令。

  ```
  net localgroup docker-users ggc_user /add
  ```

  Windows PowerShell

  要将或您用来运行 Docker 容器组件的用户添加到`ggc_user`该`docker-users`组，请以管理员身份运行以下命令。

  ```
  Add-LocalGroupMember -Group docker-users -Member ggc_user
  ```

- 如果您将 AWS IoT Greengrass Core 软件配置为使用网络代理，则必须将 Docker 配置为使用相同的代理服务器。
- 如果您的 Docker 镜像存储在 Amazon ECR 私有注册表中，则必须将令牌交换服务组件作为依赖项包含在 Docker 容器组件中。此外，Greengrass 设备角色

必须允许、ecr:GetDownloadUrlForLayer和操作ecr:BatchGetImage，如ecr:GetAuthorizationToken以下示例 IAM 策略所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- 支持 docker 应用程序管理器组件在 VPC 中运行。要在 VPC 中部署此组件，需要满足以下条件。
  - docker 应用程序管理器组件必须具有连接才能下载镜像。例如，如果您使用 ECR，则必须连接到以下端点。
    - `*.dkr.ecr.region.amazonaws.com`（VPC 终端节点`com.amazonaws.region.ecr.dkr`）
    - `api.ecr.region.amazonaws.com`（VPC 终端节点`com.amazonaws.region.ecr.api`）

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
`ecr.region.amazonaws.com`	443	否	如果您从亚马逊 ECR 下载 Docker 镜

Endpoint	端口	必需	描述
			像，则为必填项。
hub.docker.com registry.hub.docker.com/v1	443	否	如果您从 Docker Hub 下载 Docker 镜像，则为必填项。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.0.11

下表列出了此组件版本 2.0.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.13.0	软性

2.0.10

下表列出了此组件版本 2.0.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.12.0	软性

2.0.9

下表列出了此组件版本 2.0.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.11.0	软性

2.0.8

下表列出了此组件版本 2.0.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.10.0	软性

2.0.7

下表列出了此组件版本 2.0.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.9.0	软性

2.0.6

下表列出了此组件版本 2.0.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.8.0	软性

2.0.5

下表列出了此组件版本 2.0.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.7.0	软性

2.0.4

下表列出了此组件版本 2.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.6.0	软性

2.0.3

下表列出了此组件版本 2.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.5.0	软性

2.0.2

下表列出了此组件版本 2.0.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.4.0	软性

2.0.1

下表列出了此组件版本 2.0.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.3.0	软性

2.0.0

下表列出了此组件版本 2.0.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.2.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件没有任何配置参数。

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.0.11	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.0.10	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.0.9	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.0.8	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.0.7	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.0.6	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.0.5	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.0.4	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.2	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.1	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.0.0	初始版本。

另请参阅

- 运行 Docker 容器

Kinesis Video Streams 的边缘连接器

Kinesis Video Streams 组件 `aws.iot.EdgeConnectorForKVS` () 的边缘连接器读取来自本地摄像机的视频源并将直播发布到 Kinesis Video Streams。您可以将此组件配置为使用实时流协议 (RTSP) 从互联网协议 (IP) 摄像机读取视频源。然后，您可以在 Amazon Managed Grafana 或本地 Grafana 服务器中设置控制面板，以监控视频流并与之交互。

您可以将此组件与集成 AWS IoT TwinMaker ，以在 Grafana 仪表板中显示和控制视频流。AWS IoT TwinMaker 是一项使您能够构建物理系统的可操作数字双胞胎的AWS服务。您可以使用AWS IoT TwinMaker 可视化来自传感器、摄像头和企业应用程序的数据，以便跟踪实际工厂、建筑物或工业厂房。您还可以使用这些数据来监控操作、诊断错误和修复错误。有关更多信息，请参阅AWS IoT TwinMaker 《用户指南》中的什么是AWS IoT TwinMaker？。

该组件将其配置存储在中AWS IoT SiteWise，这是一项对工业数据进行建模和存储的AWS服务。在中AWS IoT SiteWise，资产表示诸如设备、设备或其他对象组之类的对象。要配置和使用此组件，您需要为每台 Greengrass 核心设备以及连接到每台核心设备的每台 IP 摄像机创建AWS IoT SiteWise资产。每项资产都有您可以配置的属性来控制直播、按需上传和本地缓存等功能。要为每台摄像机指定网址，请在中创建一个AWS Secrets Manager包含摄像机网址的密钥。如果摄像机需要身份验证，则还需要在 URL 中指定用户名和密码。然后，您可以在 IP 摄像机的资产属性中指定该机密。

此组件将每台摄像机的视频流上传到 Kinesis 视频流。您可以在每台摄像机的AWS IoT SiteWise资产配置中指定目标 Kinesis 视频流的名称。如果 Kinesis 视频流不存在，则此组件会为您创建。

AWS IoT TwinMaker 提供了一个脚本，你可以运行该脚本来创建这些AWS IoT SiteWise资产和 Secrets Manager 密钥。有关如何创建这些资源以及如何安装、配置和使用此组件的更多信息，请参阅《AWS IoT TwinMaker 用户指南》中的AWS IoT TwinMaker 视频集成。

> **Note**
>
> Kinesis Video Streams Video Streams 组件的边缘连接器仅在以下版本中AWS 区域可用：
>
> - 美国东部（弗吉尼亚州北部）
> - 美国西部（俄勒冈州）
> - 欧洲地区（法兰克福）
> - 欧洲地区（爱尔兰）
> - 亚太地区（新加坡）

主题

- 版本
- 类型
- 操作系统
- 要求

- 依赖项
- 配置
- 许可证
- 使用量
- 本地日志文件
- 更改日志
- 另请参阅

版本

此组件有以下版本：

- 1.0.x

类型

此组件是一个通用组件 (aws.greengrass.generic)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

- 您只能将此组件部署到单核设备，因为每个核心设备的组件配置必须是唯一的。您无法将此组件部署到核心设备组。
- GStreamer 1.18.4 或更高版本安装在核心设备上。有关更多信息，请参阅安装 gStreamer。

  在带有的设备上apt，您可以运行以下命令来安装 GStreamer。

  ```
  sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
   gstreamer1.0-plugins-base-apps
  sudo apt install -y gstreamer1.0-libav
  ```

```
sudo apt install -y gstreamer1.0-plugins-bad gstreamer1.0-plugins-good gstreamer1.0-
plugins-ugly gstreamer1.0-tools
```

- 每台核心设备的AWS IoT SiteWise资产。此AWS IoT SiteWise资产代表核心设备。有关如何创建此资产的更多信息，请参阅《AWS IoT TwinMaker 用户指南》中的AWS IoT TwinMaker 视频集成。
- 与每个核心设备相连的每台 IP 摄像机的AWS IoT SiteWise资产。这些AWS IoT SiteWise资产代表向每台核心设备传输视频的摄像机。每台摄像机的资产都必须与连接到摄像机的核心设备的资产相关联。相机资产具有属性，您可以配置这些属性来指定 Kinesis 视频流、身份验证密钥和视频流参数。有关如何创建和配置摄像机资产的更多信息，请参阅《AWS IoT TwinMaker 用户指南》中的AWS IoT TwinMaker 视频集成。
- 每台 IP 摄像机的AWS Secrets Manager秘密。这个密钥必须定义一个键值对，其中密钥是`RTSPStreamUrl`，值是摄像机的网址。如果摄像机需要身份验证，请在此 URL 中包含用户名和密码。在创建此组件所需的资源时，您可以使用脚本来创建密钥。有关更多信息，请参阅《AWS IoT TwinMaker 用户指南》中的AWS IoT TwinMaker 视频集成。

  您还可以使用 Secrets Manager 控制台和 API 来创建其他密钥。有关更多信息，请参阅《AWS Secrets Manager用户指南》中的创建密钥。
- Greengrass 代币交换角色必须允许以下、和 AWS IoT SiteWise Kinesis Video Streams 操作，如 AWS Secrets Manager以下示例 IAM 策略所示。

> **Note**
>
> 此示例策略允许设备获取名为**`IPCamera1Url`**和的密钥的值**`IPCamera2Url`**。在配置每个 IP 摄像机时，您需要指定一个包含该摄像机 URL 的密钥。如果摄像机需要身份验证，则还需要在 URL 中指定用户名和密码。核心设备的令牌交换角色必须允许访问每个要连接的 IP 摄像机的密钥。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:region:account-id:secret:IPCamera1Url",
        "arn:aws:secretsmanager:region:account-id:secret:IPCamera2Url"
```

```
        ]
      },
      {
        "Action": [
          "iotsitewise:BatchPutAssetPropertyValue",
          "iotsitewise:DescribeAsset",
          "iotsitewise:DescribeAssetModel",
          "iotsitewise:DescribeAssetProperty",
          "iotsitewise:GetAssetPropertyValue",
          "iotsitewise:ListAssetRelationships",
          "iotsitewise:ListAssets",
          "iotsitewise:ListAssociatedAssets",
          "kinesisvideo:CreateStream",
          "kinesisvideo:DescribeStream",
          "kinesisvideo:GetDataEndpoint",
          "kinesisvideo:PutMedia",
          "kinesisvideo:TagStream"
        ],
        "Effect": "Allow",
        "Resource": [
          "*"
        ]
      }
    ]
  }
```

> **Note**
>
> 如果您使用客户管理的密AWS Key Management Service钥来加密机密，则设备角色还必须允许该`kms:Decrypt`操作。

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
`kinesisvideo. region.amazonaws.com`	443	是	将数据上传到 Kinesis

Endpoint	端口	必需	描述
			Video Streams。
`data.iotsitewise.`*`region`*`.amazonaws.com`	443	是	将视频流元数据发布到AWS IoT SiteWise。
`secretsmanager.`*`region`*`.amazonaws.com`	443	是	将摄像头 URL 密钥下载到核心设备。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

下表列出了此组件的 1.0.0 到 1.0.4 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
代币兑换服务	>=2.0.3	硬性
直播管理器	>=2.0.9	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`SiteWiseAssetIdForHub`

代表此核心设备的AWS IoT SiteWise资产的 ID。有关如何创建此资产并使用它与该组件交互的更多信息，请参阅《AWS IoT TwinMaker 用户指南》中的AWS IoT TwinMaker 视频集成。

Example 示例：配置合并更新

```
{
  "SiteWiseAssetIdForHub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

许可证

此组件包括以下第三方软件/许可：

- Quartz Job 调度器 /Apache 许可证 2.0
- gStreamer 1.x 的 Java 绑定/G NU 宽松通用公共许可证 v3.0

使用量

要配置此组件并与之交互，可以在代表核心设备及其连接的 IP 摄像机的AWS IoT SiteWise资产上设置属性。您还可以通过以下方式在 Grafana 仪表板中可视化视频流并与之交互。AWS IoT TwinMaker有关更多信息，请参阅《AWS IoT TwinMaker 用户指南》中的AWS IoT TwinMaker 视频集成。

本地日志文件

此组件使用以下日志文件。

```
/greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。*/greengrass/v2*替换为AWS IoT Greengrass根文件夹的路径。

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
1.0.4	错误修复和改进 • 修复了导致实时上传停止的问题。
1.0.3	常规错误修复和性能改进。
1.0.1	常规错误修复和性能改进。
1.0.0	初始版本。

另请参阅

- AWS IoT TwinMaker 用户指南 中的什么是 AWS IoT TwinMaker？
- AWS IoT TwinMaker 《AWS IoT TwinMaker 用户指南》中的@@ 视频集成
- AWS IoT SiteWise 用户指南 中的什么是 AWS IoT SiteWise？
- 更新《AWS IoT SiteWise用户指南》中的属性值
- AWS Secrets Manager 用户指南 中的什么是 AWS Secrets Manager？
- 在《AWS Secrets Manager用户指南》中@@ 创建和管理密钥

Greengrass CLI

Greengrass CLI 组件 `aws.greengrass.Cli` () 提供了一个本地命令行界面，您可以在核心设备上使用该界面在本地开发和调试组件。例如，Greengrass CLI 允许您在核心设备上创建本地部署和重启组件。

您可以在安装 C AWS IoT Greengrass ore 软件时安装此组件。有关更多信息，请参阅 教程：AWS IoT Greengrass V2 入门。

Important

我们建议您仅在开发环境中使用此组件，而不是在生产环境中使用。此组件提供对生产环境中通常不需要的信息和操作的访问。遵循最低权限原则，将此组件仅部署到需要的核心设备。

安装此组件后，运行以下命令以查看其帮助文档。安装此组件后，它会在*/greengrass/v2*/bin文件夹greengrass-cli中添加一个符号链接。您可以从此路径运行 Greengrass CLI，也可以将其添加到PATH您的环境变量中以在没有绝对路径的情况下运行。greengrass-cli

Linux or Unix

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

Windows

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

例如，以下命令会重新启动名com.example.HelloWorld为的组件。

Linux or Unix

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

Windows

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

有关更多信息，请参阅 Greengrass 命令行界面。

主题

- 版本
- 类型
- 操作系统

- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x
- 2.8.x
- 2.7.x
- 2.6.x
- 2.5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 您必须获得授权才能使用 Greengrass CLI 与核心软件进行交互。 AWS IoT Greengrass 要使用 Greengrass CLI，请执行以下任一操作：
  - 使用运行 C AWS IoT Greengrass ore 软件的系统用户。
  - 使用具有 root 权限或管理员权限的用户。在 Linux 核心设备上sudo，您可以使用获取根权限。
  - 部署组件时，请使用您在AuthorizedPosixGroups或AuthorizedWindowsGroups配置参数中指定的组中的系统用户。有关更多信息，请参阅 Greengrass CLI 组件配置。
- 支持 Greengrass CLI 组件在 VPC 中运行。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.12.0 – 2.12.4

下表列出了此组件版本 2.12.0 到 2.12.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.12.0 <2.13.0	软性

2.11.0 – 2.11.3

下表列出了此组件的 2.11.0 到 2.11.3 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.11.0 <2.12.0	软性

2.10.0 – 2.10.3

下表列出了此组件的 2.10.0 到 2.10.3 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.11.0	软性

2.9.0 – 2.9.6

下表列出了此组件的 2.9.0 到 2.9.6 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.10.0	软性

2.8.0 – 2.8.1

下表列出了此组件版本 2.8.0 和 2.8.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.9.0	软性

2.7.0

下表列出了此组件版本 2.7.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.8.0	软性

2.6.0

下表列出了此组件版本 2.6.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.7.0	软性

2.5.0 – 2.5.6

下表列出了此组件的 2.5.0 到 2.5.6 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.6.0	软性

2.4.0

下表列出了此组件版本 2.4.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.5.0	软性

2.3.0

下表列出了此组件版本 2.3.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.4.0	软性

2.2.0

下表列出了此组件版本 2.2.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.3.0	软性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.2.0	软性

2.0.x

下表列出了此组件版本 2.0.x 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.1.0	软性

Note

Greengrass nucleus 的最低兼容版本对应于 Greengrass CLI 组件的补丁版本。

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

2.5.x

`AuthorizedPosixGroups`

（可选）包含以逗号分隔的系统组列表的字符串。您授权这些系统组使用 Greengrass CLI 与核心软件进行交互。 AWS IoT Greengrass 您可以指定群组名称或群组 ID。例

如，`group1,1002,group3`授权三个系统组（`group11002`、和`group3`）使用 Greengrass CLI。

如果您未指定要授权的任何群组，则可以以 root 用户 `sudo` () 或运行核心软件的系统用户身份使用 Greengrass CLI。 AWS IoT Greengrass

`AuthorizedWindowsGroups`

（可选）包含以逗号分隔的系统组列表的字符串。您授权这些系统组使用 Greengrass CLI 与核心软件进行交互。 AWS IoT Greengrass 您可以指定群组名称或群组 ID。例如，`group1,1002,group3`授权三个系统组（`group11002`、和`group3`）使用 Greengrass CLI。

如果您未指定要授权的任何群组，则可以以管理员或运行 Core 软件的系统用户身份使用 Greengrass CLI。 AWS IoT Greengrass

Example 示例：配置合并更新

以下示例配置指定授权三个 POSIX 系统组（`group11002`、和`group3`）和两个 Windows 用户组（`Device Operators`和`QA Engineers`）使用 Greengrass CLI。

```
{
  "AuthorizedPosixGroups": "group1,1002,group3",
  "AuthorizedWindowsGroups": "Device Operators,QA Engineers"
}
```

2.4.x - 2.0.x

`AuthorizedPosixGroups`

（可选）包含以逗号分隔的系统组列表的字符串。您授权这些系统组使用 Greengrass CLI 与核心软件进行交互。 AWS IoT Greengrass 您可以指定群组名称或群组 ID。例如，`group1,1002,group3`授权三个系统组（`group11002`、和`group3`）使用 Greengrass CLI。

如果您未指定要授权的任何群组，则可以以 root 用户 `sudo` () 或运行核心软件的系统用户身份使用 Greengrass CLI。 AWS IoT Greengrass

Example 示例：配置合并更新

以下示例配置指定授权三个系统组（`group11002`、和`group3`）使用 Greengrass CLI。

```
{
  "AuthorizedPosixGroups": "group1,1002,group3"
}
```

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.12.4	Greengrass nucleus 版本 2.12.4 版本的版本已更新。

版本	更改
2.12.3	**Warning** 此版本不再可用。此版本的改进将在此组件的更高版本中提供。 Greengrass nucleus 版本 2.12.3 版本的版本已更新。
2.12.2	Greengrass nucleus 版本 2.12.2 版本的版本已更新。
2.12.1	Greengrass nucleus 版本 2.12.1 版本的版本已更新。
2.12.0	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.11.3	Greengrass nucleus 版本 2.11.3 版本的版本已更新。
2.11.2	Greengrass nucleus 版本 2.11.2 版本的版本已更新。
2.11.1	Greengrass nucleus 版本 2.11.1 版本的版本已更新。
2.11.0	新功能 • 允许您取消本地部署。 • 使您能够为本地部署配置故障处理策略。 • 改进了详细的部署状态报告。
2.10.3	Greengrass nucleus 版本 2.10.3 版本的版本已更新。
2.10.2	Greengrass nucleus 版本 2.10.2 版本的版本已更新。
2.10.1	Greengrass nucleus 版本 2.10.1 版本的版本已更新。
2.10.0	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.9.6	Greengrass nucleus 版本 2.9.6 版本的版本已更新。
2.9.5	Greengrass nucleus 版本 2.9.5 版本的版本已更新。
2.9.4	Greengrass nucleus 版本 2.9.4 版本的版本已更新。

版本	更改
2.9.3	Greengrass nucleus 版本 2.9.3 版本的版本已更新。
2.9.2	Greengrass nucleus 版本 2.9.2 版本的版本已更新。
2.9.1	Greengrass nucleus 版本 2.9.1 版本的版本已更新。
2.9.0	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.8.1	Greengrass nucleus 版本 2.8.1 版本的版本已更新。
2.8.0	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.7.0	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.6.0	新功能 • 添加了对自定义组件的支持，以调用 Greengrass CLI 使用的进程间通信 (IPC) 操作。您可以使用这些 IPC 操作来管理本地部署、查看组件详细信息以及生成用于登录本地调试控制台的密码。有关更多信息，请参阅 IPC：管理本地部署和组件。 错误修复和改进 • 其他小修复和改进。
2.5.6	Greengrass nucleus 版本 2.5.6 版本的版本已更新。
2.5.5	Greengrass nucleus 版本 2.5.5 版本的版本已更新。
2.5.4	Greengrass nucleus 版本 2.5.4 版本的版本已更新。
2.5.3	Greengrass nucleus 版本 2.5.3 版本的版本已更新。
2.5.2	Greengrass nucleus 版本 2.5.2 版本的版本已更新。
2.5.1	Greengrass nucleus 版本 2.5.1 版本的版本已更新。

版本	更改
2.5.0	新功能 • 增加了对运行 Windows 的核心设备的支持。 • 添加新的`AuthorizedWindowsGroups` 配置参数，您可以指定该参数来授权系统组在 Windows 设备上使用 Greengrass CLI。 • 为本地部署添加`windowsUser` 参数。您可以使用此参数指定用于在 Windows 核心设备上运行组件的用户。
2.4.0	新功能 • 增加了对系统资源限制的支持。创建本地部署时，可以配置每个组件的进程可以在核心设备上使用的最大 CPU 和 RAM 使用量。有关更多信息，请参阅为组件配置系统资源限制和部署创建命令。
2.3.0	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.2.0	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.1.0	Greengrass nucleus 版本 2.1.0 版本的版本已更新。
2.0.5	Greengrass nucleus 版本 2.0.5 版本的版本已更新。
2.0.4	Greengrass nucleus 版本 2.0.4 版本的版本已更新。
2.0.3	初始版本。

IP 探测器

IP 检测器组件 (`aws.greengrass.clientdevices.IPDetector`) 执行以下操作：

- 监控 Greengrass 核心设备的网络连接信息。此信息包括核心设备的网络端点和 MQTT 代理运行的端口。
- 更新 AWS IoT Greengrass 云服务中核心设备的连接信息。

客户设备可以使用 Greengrass 云发现来检索相关核心设备的连接信息。然后，客户端设备可以尝试连接到每台核心设备，直到它们成功连接。

Note

客户端设备是连接到 Greengrass 核心设备以发送 MQTT 消息和数据进行处理的本地物联网设备。有关更多信息，请参阅 与本地物联网设备互动。

IP 探测器组件将核心设备的现有连接信息替换为其检测到的信息。由于此组件会删除现有信息，因此您可以使用 IP 检测器组件，也可以手动管理连接信息。

Note

IP 检测器组件仅检测 IPv4 地址。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x
- 2.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- Greengrass 服务角色必须与 AWS 账户 您的关联并允许和权限。`iot:GetThingShadow` `iot:UpdateThingShadow`
- 核心设备的 AWS IoT 策略必须允许该`greengrass:UpdateConnectivityInfo`权限。有关更多信息，请参阅 数据层面操作的 AWS IoT 策略 和 支持客户端设备的最低AWS IoT政策。
- 如果您将核心设备的 MQTT 代理组件配置为使用默认端口 8883 以外的端口，则必须使用 IP 检测器 v2.1.0 或更高版本。将其配置为报告代理运行的端口。
- 如果您的网络设置很复杂，IP 检测器组件可能无法识别客户端设备可以连接到核心设备的端点。如果 IP 检测器组件无法管理端点，则必须改为手动管理核心设备端点。例如，如果核心设备位于将 MQTT 代理端口转发给它的路由器后面，则必须将路由器的 IP 地址指定为核心设备的终端节点。有关更多信息，请参阅 管理核心设备端点。
- 支持 IP 检测器组件在 VPC 中运行。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.8 – 2.1.9

下表列出了此组件版本 2.1.8 和 2.1.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.13.0	软性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.12.0	软性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.11.0	软性

2.1.5

下表列出了此组件版本 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.10.0	软性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.9.0	软性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.8.0	软性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.7.0	软性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.6.0	软性

2.1.0 and 2.0.2

下表列出了此组件版本 2.1.0 和 2.0.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.5.0	软性

2.0.1

下表列出了此组件版本 2.0.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.4.0	软性

2.0.0

下表列出了此组件版本 2.0.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.3.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

2.1.x

`defaultPort`

（可选）此组件检测到 IP 地址时要报告的 MQTT 代理端口。如果您将 MQTT 代理配置为使用与默认端口 8883 不同的端口，则必须指定此参数。

默认：`8883`

`includeIPv4LoopbackAddrs`

（可选）您可以启用此选项来检测和报告 IPv4 环回地址。这些是 IP 地址，例如`localhost`，设备可以与自己通信的地方。在核心设备和客户端设备在同一系统上运行的测试环境中使用此选项。

默认：`false`

`includeIPv4LinkLocalAddrs`

（可选）您可以启用此选项来检测和报告 IPv4 链路本地地址。如果核心设备的网络没有动态主机配置协议 (DHCP) 或静态分配的 IP 地址，请使用此选项。

默认：false

2.0.x

includeIPv4LoopbackAddrs

（可选）您可以启用此选项来检测和报告 IPv4 环回地址。这些是 IP 地址，例如localhost，设备可以与自己通信的地方。在核心设备和客户端设备在同一系统上运行的测试环境中使用此选项。

默认：false

includeIPv4LinkLocalAddrs

（可选）您可以启用此选项来检测和报告 IPv4 链路本地地址。如果核心设备的网络没有动态主机配置协议 (DHCP) 或静态分配的 IP 地址，请使用此选项。

默认：false

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.9	错误修复和改进 • 将获取的 IP 步骤调整为仅发送调试日志级别的日志。
2.1.8	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.7	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.5	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.4	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.2	错误修复和改进 • 改进了此组件在某些情况下记录的错误消息。 • Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.1.0	改进 • 添加`defaultPort` 参数，使您能够使用非默认 MQTT 代理端口。 • 更新以使日志消息更加清晰。
2.0.2	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.1	Greengrass nucleus 版本 2.3.0 版本的版本已更新。

版本	更改
2.0.0	初始版本。

Firehose

Firehose 组件 (`aws.greengrass.KinesisFirehose`) 通过亚马逊数据 Firehose 传输流将数据发布到目的地，例如亚马逊 S3、亚马逊 Redshift 和亚马逊服务。 OpenSearch 有关更多信息，请参阅什么是 Amazon Data Firehose？ 在《亚马逊数据 Firehose 开发者指南》中。

要使用此组件发布到 Kinesis 交付流，请向该组件订阅的主题发布消息。默认情况下，此组件订阅`kinesisfirehose/message`和`kinesisfirehose/message/binary/#`本地发布/订阅主题。部署此组件时，您可以指定其他主题，包括 AWS IoT Core MQTT 主题。

> **Note**
> 此组件提供的功能与 V1 中的 Firehose 连接器类似。 AWS IoT Greengrass 有关更多信息，请参阅《AWS IoT Greengrass V1 开发者指南》中的 Firehose 连接器。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 输入数据
- 输出数据
- 本地日志文件
- 许可证
- 更改日志
- 另请参阅

版本

此组件有以下版本：

- 2.1.x
- 2.0.x

类型

此组件是一个 Lambda 组件 () `aws.greengrass.lambda`。Greengrass 核心使用 Lambda 启动器组件运行此组件的 Lambda 函数。

有关更多信息，请参阅组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

- 您的核心设备必须满足运行 Lambda 函数的要求。如果您希望核心设备运行容器化的 Lambda 函数，则该设备必须满足要求。有关更多信息，请参阅Lambda 函数要求。
- Python 版本 3.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- Greengrass 设备角色必须允许`firehose:PutRecordBatch`和操作，如`firehose:PutRecord`以下示例 IAM 策略所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:firehose:region:account-id:deliverystream/stream-name"
```

```
      ]
    }
  ]
}
```

您可以动态覆盖此组件的输入消息负载中的默认传送流。如果您的应用程序使用此功能，则 IAM 策略必须将所有目标流作为资源包括在内。您可以授予对资源的具体或条件访问权限（例如，通过使用通配符 * 命名方案）。

- 要接收此组件的输出数据，在部署此组件时，必须合并旧版订阅路由器组件 (`aws.greengrass.LegacySubscriptionRouter`) 的以下配置更新。此配置指定此组件发布响应的主题。

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-kinesisfirehose": {
        "id": "aws-greengrass-kinesisfirehose",
        "source": "component:aws.greengrass.KinesisFirehose",
        "subject": "kinesisfirehose/message/status",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-kinesisfirehose": {
        "id": "aws-greengrass-kinesisfirehose",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
  kinesisfirehose:version",
        "subject": "kinesisfirehose/message/status",
        "target": "cloud"
      }
    }
  }
  ```

  - 将*##*替换为您 AWS 区域 使用的区域。

  - 将*##*替换为该组件运行的 Lambda 函数的版本。要查找 Lambda 函数版本，您必须查看要部署的此组件版本的配方。在AWS IoT Greengrass 控制台中打开此组件的详细信息页面，查找 Lambda 函数键值对。此键值对包含 Lambda 函数的名称和版本。

> **Important**
> 每次部署此组件时，都必须更新旧版订阅路由器上的 Lambda 函数版本。这样可以确保您为部署的组件版本使用正确的 Lambda 函数版本。

有关更多信息，请参阅创建部署。

- 支持在 VPC 中运行 Firehose 组件。要在 VPC 中部署此组件，需要满足以下条件。
  - Firehose 组件必须连接到 `firehose.region.amazonaws.com` VPC 终端节点为。`com.amazonaws.region.kinesis-firehose`

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
`firehose.`*`region`*`.amazonaws.com`	443	是	将数据上传到 Firehose。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.5

下表列出了此组件版本 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性

依赖关系	兼容版本	依赖关系类型
代币兑换服务	^2.0.0	硬性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.8 - 2.1.0

下表列出了此组件版本 2.0.8 和 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.7

下表列出了此组件版本 2.0.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.6

下表列出了此组件版本 2.0.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.5

下表列出了此组件版本 2.0.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.4

下表列出了此组件版本 2.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.3

下表列出了此组件版本 2.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	硬性
Lambda 启动器	>=1.0.0	硬性
Lambda 运行时	>=1.0.0	软性
代币兑换服务	>=1.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

> **Note**
> 此组件的默认配置包括 Lambda 函数参数。我们建议您仅编辑以下参数，以便在您的设备上配置此组件。

lambdaParams

包含此组件的 Lambda 函数参数的对象。该对象包含以下信息：

EnvironmentVariables

一个包含 Lambda 函数参数的对象。该对象包含以下信息：

DEFAULT_DELIVERY_STREAM_ARN

组件发送数据的默认 Firehose 传输流的 ARN。您可以使用输入消息负载中的delivery_stream_arn属性覆盖目标流。

> **Note**
>
> 核心设备角色必须允许对所有目标传送流执行所需的操作。有关更多信息，请参阅要求。

PUBLISH_INTERVAL

（可选）组件将批处理数据发布到 Firehose 之前等待的最大秒数。要将组件配置为在收到指标时发布指标（这意味着不进行批处理），请指定0。

此值最多可为 900 秒。

默认值：10 秒

DELIVERY_STREAM_QUEUE_SIZE

（可选）在组件拒绝同一传输流的新记录之前，要在内存中保留的最大记录数。

此值必须至少为 2,000 条记录。

默认值：5,000 条记录

containerMode

（可选）此组件的容器化模式。从以下选项中进行选择：

- NoContainer— 该组件不在隔离的运行时环境中运行。
- GreengrassContainer— 该组件在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

默认：GreengrassContainer

`containerParams`

（可选）包含此组件容器参数的对象。如果您为指定`GreengrassContainer`，则该组件将使用这些参数`containerMode`。

该对象包含以下信息：

`memorySize`

（可选）要分配给组件的内存量（以千字节为单位）。

默认为 64 MB (65,535 KB)。

`pubsubTopics`

（可选）一个对象，其中包含组件订阅以接收消息的主题。您可以指定每个主题以及该组件是订阅来自的 MQTT 主题 AWS IoT Core 还是本地发布/订阅主题。

该对象包含以下信息：

`0`— 这是字符串形式的数组索引。

包含以下信息的对象：

`type`

（可选）此组件用于订阅消息的发布/订阅消息的类型。从以下选项中进行选择：

- PUB_SUB – 订阅本地发布/订阅消息。如果选择此选项，则主题不能包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅本地消息。
- `IOT_CORE`— 订阅 AWS IoT Core MQTT 消息。如果选择此选项，则主题可以包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

默认：PUB_SUB

`topic`

（可选）组件订阅以接收消息的主题。如果您`IotCore`为指定`type`，则可以在本主题中使用 MQTT 通配符（+和#）。

Example 示例：配置合并更新（容器模式）

```
{
```

```
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example 示例：配置合并更新（无容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "NoContainer"
}
```

输入数据

此组件接受以下主题的直播内容，并将内容发送到目标传送流。该组件接受两种类型的输入数据：

- JSON 数据，位于 `kinesisfirehose/message` 主题中。
- 二进制数据，位于 `kinesisfirehose/message/binary/#` 主题中。

JSON 数据的默认主题（本地发布/订阅）：`kinesisfirehose/message`

该消息接受以下属性。输入消息必须采用 JSON 格式。

`request`

要发送到传输流和目标传输流（如果不同于默认流）的数据。

类型：其中`object`包含以下信息：

`data`

要发送到传输流的数据。

类型：`string`

`delivery_stream_arn`

（可选）目标 Firehose 传送流的 ARN。指定此属性可覆盖默认的传送流。

类型：`string`

`id`

请求的任意 ID。使用此属性将输入请求映射到输出响应。当您指定此属性时，组件会将响应对象中的`id`属性设置为该值。

类型：`string`

Example 示例输入

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-id:deliverystream/
stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

二进制数据的默认主题（本地发布/订阅）：`kinesisfirehose/message/binary/#`

使用此主题发送包含二进制数据的消息。该组件不解析二进制数据。该组件按原样传输数据。

要将输入请求映射到输出响应，请将消息主题中的 # 通配符替换为任意请求 ID。例如，如果您将消息发布到 `kinesisfirehose/message/binary/request123`，响应对象中的 `id` 属性将设置为 `request123`。

如果您不希望将请求映射到响应，可以将消息发布到 `kinesisfirehose/message/binary/`。请务必包含尾部的斜杠 () `/`。

输出数据

默认情况下，此组件将响应作为以下 MQTT 主题的输出数据发布。您必须在传统订阅路由器组件的配置`subject`中将此主题指定为。有关如何在您的自定义组件中订阅有关此主题的消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

默认主题 (AWS IoT Core MQTT)：`kinesisfirehose/message/status`

Example 示例输出

响应将包含批次中发送的每条数据记录的状态。

```
{
  "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**
> 如果组件检测到可以重试的错误（例如连接错误），则会在下一批中重试发布。

本地日志文件

此组件使用以下日志文件。

```
/greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。*/greengrass/v2*替换为 AWS IoT Greengrass 根文件夹的路径。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

许可证

此组件包括以下第三方软件/许可：

- AWS SDK for Python (Boto3)/Apache 许可证 2.0
- botocore/Apache 许可证 2.0
- dateutil/PSF 许可证
- docutils/BSD 许可证，GNU 通用公共许可证 (GPL)，Python 软件基金会许可证，公共领域
- jmespath/MIT 许可证
- s3transfer/Apache 许可证 2.0
- urllib3/MIT 许可证

该组件是根据 Greengrass 核心软件许可协议发布的。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.7	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.5	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.4	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.6.0 版本的版本已更新。

版本	更改
2.1.0	新功能 • 添加对 HTTPS 网络代理配置的支持。有关更多信息，请参阅 通过端口 443 或网络代理进行连接 和 使核心设备能够信任 HTTPS 代理。
2.0.8	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.7	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.6	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.5	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.0.4	Greengrass nucleus 版本 2.1.0 版本的版本已更新。
2.0.3	初始版本。

另请参阅

- 什么是亚马逊 Data Firehose？ 在 Amazon Data Firehos e 开发者指南中

Lambda 启动器

Lambda 启动器组件 (`aws.greengrass.LambdaLauncher`) 启动和停止 AWS IoT Greengrass 核心 AWS Lambda 设备上的功能。此组件还会设置任何容器化并以您指定的用户身份运行进程。

> **Note**
>
> 将 Lambda 函数组件部署到核心设备时，部署还包括此组件。有关更多信息，请参阅运行AWS Lambda函数。

主题

- 版本
- 类型
- 操作系统

- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

- 您的核心设备必须满足运行 Lambda 函数的要求。如果您希望核心设备运行容器化的 Lambda 函数，则该设备必须满足要求。有关更多信息，请参阅Lambda 函数要求。
- 支持在 VPC 中运行 Lambda 启动器组件。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.0.11 – 2.0.13

下表列出了该组件 2.0.11 到 2.0.13 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Lambda 管理器	>=2.0.0 <2.4.0	硬性

2.0.9 – 2.0.10

下表列出了此组件版本 2.0.9 到 2.0.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Lambda 管理器	>=2.0.0 <2.3.0	硬性

2.0.4 - 2.0.8

下表列出了此组件版本 2.0.4 到 2.0.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Lambda 管理器	>=2.0.0 <2.2.0	硬性

2.0.3

下表列出了此组件版本 2.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Lambda 管理器	>=2.0.3 <2.1.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件没有任何配置参数。

本地日志文件

此组件使用以下日志文件。

```
/greengrass/v2/logs/lambdaFunctionComponentName.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。*/greengrass/v2*替换为 AWS IoT Greengrass 根文件夹的路径，并将*lambdaFunctionComponent##*替换为该组件启动的 Lambda 函数组件的名称。

  ```
  sudo tail -f /greengrass/v2/logs/lambdaFunctionComponentName.log
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.0.13	错误修复和改进 常规错误修复和性能改进。
2.0.12	错误修复和改进 修复了如果前一个进程未正确停止，Lambda 启动器可能会抛出错误的问题。
2.0.11	支持 Lambda 管理器 2.3.0。
2.0.10	错误修复和改进 • 常规错误修复和性能改进。
2.0.9	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.8	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.7	Greengrass nucleus 版本 2.3.0 版本的版本已更新。

版本	更改
2.0.6	常规性能改进和错误修复。
2.0.4	错误修复和改进 • 修复了组件无法正确传递`AddGroupOwner` 到 Lambda 函数容器的问题。
2.0.3	初始版本。

Lambda 管理器

Lambda 管理器组件 (`aws.greengrass.LambdaManager`) 管理在 Greengrass 核心设备上运行的 AWS Lambda 函数的工作项和进程间通信。

> **Note**
> 将 Lambda 函数组件部署到核心设备时，部署还包括此组件。有关更多信息，请参阅 运行 AWS Lambda函数。

主题

- 版本
- 操作系统
- 类型
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

操作系统

此组件只能安装在 Linux 核心设备上。

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

要求

此组件具有以下要求：

- 您的核心设备必须满足运行 Lambda 函数的要求。如果您希望核心设备运行容器化的 Lambda 函数，则该设备必须满足执行此操作的要求。有关更多信息，请参阅 Lambda 函数要求。
- 支持在 VPC 中运行 Lambda 管理器组件。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.3.2 and 2.3.3

下表列出了此组件版本 2.3.2 和 2.3.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性

2.2.10 and 2.3.1

下表列出了此组件版本 2.2.10 和 2.3.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性

2.2.8 and 2.2.9

下表列出了此组件版本 2.2.8 和 2.2.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性

2.2.7

下表列出了此组件版本 2.2.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性

2.2.6

下表列出了此组件版本 2.2.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性

2.2.5

下表列出了此组件版本 2.2.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性

2.2.4

下表列出了此组件版本 2.2.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性

2.2.1 - 2.2.3

下表列出了此组件版本 2.2.1 到 2.2.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性

2.2.0

下表列出了此组件版本 2.2.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.6.0	软性

2.1.3 and 2.1.4

下表列出了此组件版本 2.1.3 和 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性

2.0.x

下表列出了此组件版本 2.0.x 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`logHandlerMode`

> Note
>
> 仅适用于 lambda 管理器版本 2.3.0+

用于选择要使用的 Lambda 日志管理器的实现。将该值设置为可使用更少`optimized`的线程来读取 lambda 日志。

`getResultTimeoutInSecond`

（可选）Lambda 函数在超时之前可以运行的最大时间（以秒为单位）。

默认：`60`

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

```
/greengrass/v2/logs/greengrass.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。*/greengrass/v2*替换为 AWS IoT Greengrass 根文件夹的路径。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.3.3	错误修复和改进 • 常规错误修复和性能改进。
2.3.2	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.3.1	错误修复和改进 • 调整某些错误的日志级别。
2.3.0	新功能 • 日志处理程序经过优化，可减少 CPU 负载。通过将配置选项`logHandlerMode` 设置为，即可使用此功能`optimized` 。 错误修复和改进 • 不再记录完整的堆栈跟踪`WorkQueueFullException` ，从而改善了日志和性能。 • 将 lambda 关闭超时从 15 秒设置为 300 秒，以防止关机超时。 • 修复了更改配置后按需 lambda 可能无法重新启动的问题。
2.2.11	错误修复和改进 • 修复了 Lambda LegacySubscriptionRouter 配置更改时配置不会更新的问题。
2.2.10	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.2.9	错误修复和改进 修复了由于时钟偏差导致端口号损坏的问题。
2.2.8	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.2.7	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.2.6	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.2.5	新功能 • 在您订阅本地发布/订阅消息的事件源中添加对 MQTT 主题通配符的支持。

版本	更改
	此功能需要 Greengrass nucleus 组件的 v2.6.0 或更高版本。 • Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.2.4	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.2.3	错误修复和改进 • 修复了 Lambda 函数的多个实例共享一个 cgroup 的问题。此组件使用 cgroup 来管理 Lambda 函数的资源使用情况。
2.2.2	错误修复和改进 • 修复了固定的 Lambda 函数组件在某些情况下意外重启的问题。
2.2.1	错误修复和改进 • 更改此组件的 Greengrass nucleus 依赖版本约束以修复依赖关系解析问题。
2.2.0	错误修复和改进 • 修复了 Lambda 函数在重启后无法写入日志的问题。 • 修复了当主题中有通配符时，传统订阅路由器会发送重复消息的问题。 • 修复了非固定 Lambda 函数无法使用中的 Greengrass 进程间通信 (IPC) 库的问题。 AWS IoT Device SDK
2.1.4	错误修复和改进 • 修复了导致使用 NodeJS 运行时的 Lambda 函数仅处理一条消息的问题。 • Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.1.0	Greengrass nucleus 版本 2.1.0 版本的版本已更新。

版本	更改
2.0.3	初始版本。

Lambda 运行时

Lambda 运行时组件 (`aws.greengrass.LambdaRuntimes`) 提供了 Greengrass 核心设备用来运行函数的运行时。AWS Lambda

将 Lambda 函数组件部署到核心设备时，部署还包括此组件。有关更多信息，请参阅 运行 AWS Lambda函数。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

- 您的核心设备必须满足运行 Lambda 函数的要求。如果您希望核心设备运行容器化的 Lambda 函数，则该设备必须满足要求。有关更多信息，请参阅 Lambda 函数要求。
- 支持在 VPC 中运行 Lambda 运行时组件。

依赖项

这个组件没有任何依赖关系。

配置

此组件没有任何配置参数。

本地日志文件

此组件不输出日志。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.0.8	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.7	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.6	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.5	Greengrass nucleus 版本 2.2.0 版本的版本已更新。

版本	更改
2.0.4	Greengrass nucleus 版本 2.1.0 版本的版本已更新。
2.0.3	初始版本。

旧版订阅路由器

传统订阅路由器 (`aws.greengrass.LegacySubscriptionRouter`) 管理 Greengrass 核心设备上的订阅。订阅是 AWS IoT Greengrass V1 的一项功能，它定义了 Lambda 函数可用于在核心设备上进行 MQTT 消息传送的主题。有关更多信息，请参阅 AWS IoT GreengrassV1 开发人员指南中的 MQTT 消息传递工作流程中的托管订阅。

您可以使用此组件启用对使用 C AWS IoT Greengrass ore SDK 的连接器组件和 Lambda 函数组件的订阅。

> **Note**
>
> 只有当您的 Lambda 函数使用AWS IoT Greengrass核心软件开发工具包中的`publish()`函数时，才需要使用旧版订阅路由器组件。如果您更新 Lambda 函数代码以使用 AWS IoT Device SDK V2 中的进程间通信 (IPC) 接口，则无需部署旧版订阅路由器组件。有关更多信息，请参阅以下进程间通信服务：
>
> - 发布/订阅本地消息
> - 发布/订阅 AWS IoT Core MQTT 消息

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件

- 更改日志

版本

此组件有以下版本：

- 2.1.x
- 2.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

- 支持在 VPC 中运行传统订阅路由器。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.11

下表列出了此组件版本 2.1.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性

2.1.10

下表列出了此组件版本 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性

2.1.9

下表列出了此组件版本 2.1.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性

2.1.8

下表列出了此组件版本 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性

2.1.5

下表列出了此组件版本 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性

2.0.3

下表列出了此组件版本 2.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

v2.1.x

`subscriptions`

（可选）要在核心设备上启用的订阅。这是一个对象，其中每个键都是一个唯一的 ID，每个值都是一个定义该连接器订阅的对象。部署使用 Core SDK 的 V1 连接器组件或 Lambda 函数时，必须配置订阅。AWS IoT Greengrass

每个订阅对象都包含以下信息：

`id`

此订阅的唯一 ID。此 ID 必须与此订阅对象的密钥匹配。

`source`

Lambda 函数，它使用AWS IoT Greengrass核心软件开发工具包针对您在中指定的主题发布 MQTT 消息。`subject`指定下列项之一：

- 核心设备上的 Lambda 函数组件的名称。使用`component:`前缀指定组件名称，例如**`component:com.example.HelloWorldLambda`**。
- 核心设备上的 Lambda 函数的亚马逊资源名称 (ARN)。

> **Important**
>
> 如果 Lambda 函数的版本发生变化，则必须使用该函数的新版本配置订阅。否则，在版本与订阅匹配之前，此组件不会路由消息。
>
> 您必须指定包含要导入的函数版本的 Amazon 资源名称 (ARN)。您不能使用像 `$LATEST` 这样的版本别名。

要部署 V1 连接器组件的订阅，请指定该组件的名称或连接器组件的 Lambda 函数的 ARN。

`subject`

源和目标可以发布和接收消息的 MQTT 主题或主题筛选器。此值支持`+`和`#`主题通配符。

`target`

接收有关您在中指定的主题的 MQTT 消息的目标。`subject`订阅指定该`source`函数向核心设备上的 Lambda 函数发布AWS IoT Core或发布 MQTT 消息。指定下列项之一：

- `cloud`。 该`source`函数将 MQTT 消息发布到。AWS IoT Core
- 核心设备上的 Lambda 函数组件的名称。使用`component:`前缀指定组件名称，例如**`component:com.example.HelloWorldLambda`**。
- 核心设备上的 Lambda 函数的亚马逊资源名称 (ARN)。

> **Important**
>
> 如果 Lambda 函数的版本发生变化，则必须使用该函数的新版本配置订阅。否则，在版本与订阅匹配之前，此组件不会路由消息。

您必须指定包含要导入的函数版本的 Amazon 资源名称 (ARN)。您不能使用像 $LATEST 这样的版本别名。

默认：无订阅

Example 配置更新示例（定义订阅AWS IoT Core）

以下示例指定 com.example.HelloWorldLambda Lambda 函数组件向AWS IoT Core发布有关该主题的 MQTT 消息。hello/world

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_cloud": {
      "id": "Greengrass_HelloWorld_to_cloud",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "cloud"
    }
  }
}
```

Example 配置更新示例（定义对另一个 Lambda 函数的订阅）

以下示例指定 com.example.HelloWorldLambda Lambda 函数组件将 MQTT 消息发布到主题上的 Lamb com.example.MessageRelay da 函数组件。hello/world

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_MessageRelay": {
      "id": "Greengrass_HelloWorld_to_MessageRelay",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "component:com.example.MessageRelay"
    }
  }
}
```

v2.0.x

`subscriptions`

（可选）要在核心设备上启用的订阅。这是一个对象，其中每个键都是一个唯一的 ID，每个值都是一个定义该连接器订阅的对象。部署使用 Core SDK 的 V1 连接器组件或 Lambda 函数时，必须配置订阅。AWS IoT Greengrass

每个订阅对象都包含以下信息：

`id`

此订阅的唯一 ID。此 ID 必须与此订阅对象的密钥匹配。

`source`

Lambda 函数，它使用AWS IoT Greengrass核心软件开发工具包针对您在中指定的主题发布 MQTT 消息。`subject`指定以下内容：

- 核心设备上的 Lambda 函数的亚马逊资源名称 (ARN)。

Important

如果 Lambda 函数的版本发生变化，则必须使用该函数的新版本配置订阅。否则，在版本与订阅匹配之前，此组件不会路由消息。
您必须指定包含要导入的函数版本的 Amazon 资源名称 (ARN)。您不能使用像 `$LATEST` 这样的版本别名。

要部署 V1 连接器组件的订阅，请指定连接器组件的 Lambda 函数的 ARN。

`subject`

源和目标可以发布和接收消息的 MQTT 主题或主题筛选器。此值支持`+`和`#`主题通配符。

`target`

接收有关您在中指定的主题的 MQTT 消息的目标。`subject`订阅指定该`source`函数向核心设备上的 Lambda 函数发布AWS IoT Core或发布 MQTT 消息。指定下列项之一：

- `cloud`。 该`source`函数将 MQTT 消息发布到。AWS IoT Core
- 核心设备上的 Lambda 函数的亚马逊资源名称 (ARN)。

> **Important**
>
> 如果 Lambda 函数的版本发生变化，则必须使用该函数的新版本配置订阅。否则，在版本与订阅匹配之前，此组件不会路由消息。
>
> 您必须指定包含要导入的函数版本的 Amazon 资源名称 (ARN)。您不能使用像 `$LATEST` 这样的版本别名。

默认：无订阅

Example 配置更新示例（定义订阅AWS IoT Core）

以下示例指定`Greengrass_HelloWorld`函数向该`hello/world`主题发布 MQTT 消息。AWS IoT Core

```
"subscriptions": {
  "Greengrass_HelloWorld_to_cloud": {
    "id": "Greengrass_HelloWorld_to_cloud",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "cloud"
  }
}
```

Example 配置更新示例（定义对另一个 Lambda 函数的订阅）

以下示例指定该`Greengrass_HelloWorld`函数向`hello/world`主题发布 MQTT 消息。`Greengrass_MessageRelay`

```
"subscriptions": {
  "Greengrass_HelloWorld_to_MessageRelay": {
    "id": "Greengrass_HelloWorld_to_MessageRelay",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_MessageRelay:5"
  }
}
```

本地日志文件

此组件不输出日志。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.11	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.10	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.9	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.8	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.7	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.5	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.4	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.1.0	错误修复和改进 • 添加了对为和指定组件名称而不是 ARN `source` 的`target`支持。如果您为订阅指定组件名称，则无需在 Lambda 函数的版本每次更改时都重新配置订阅。
2.0.3	初始版本。

本地调试控制台

本地调试控制台组件 (`aws.greengrass.LocalDebugConsole`) 提供了一个本地仪表板，用于显示有关 AWS IoT Greengrass 核心设备及其组件的信息。您可以使用此仪表板来调试核心设备和管理本地组件。

> **Important**
>
> 我们建议您仅在开发环境中使用此组件，而不是在生产环境中使用。此组件提供对生产环境中通常不需要的信息和操作的访问。遵循最低权限原则，将此组件仅部署到需要的核心设备。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 使用量
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 您使用用户名和密码登录控制面板。用户名（即）是为您提供的。debug您必须使用 AWS IoT Greengrass CLI 创建临时密码，以便通过核心设备上的控制面板对您进行身份验证。您必须能够使用 AWS IoT Greengrass CLI 才能使用本地调试控制台。有关更多信息，请参阅 Greengr ass CLI 要求。有关如何生成密码和登录的更多信息，请参阅本地调试控制台组件用法。
- 支持在 VPC 中运行本地调试控制台组件。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.4.1 – 2.4.2

下表列出了此组件版本 2.4.1 到 2.4.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.10.0 <2.13.0	硬性
Greengrass CLI	>=2.10.0 <2.13.0	硬性

2.4.0

下表列出了此组件版本 2.4.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.10.0 <2.12.0	硬性
Greengrass CLI	>=2.10.0 <2.12.0	硬性

2.3.0 and 2.3.1

下表列出了此组件版本 2.3.0 和 2.3.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.10.0 <2.12.0	硬性
Greengrass CLI	>=2.10.0 <2.12.0	硬性

2.2.9

下表列出了此组件版本 2.2.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.12.0	硬性
Greengrass CLI	>=2.1.0 <2.12.0	硬性

2.2.8

下表列出了此组件版本 2.2.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.11.0	硬性
Greengrass CLI	>=2.1.0 <2.11.0	硬性

2.2.7

下表列出了此组件版本 2.2.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.10.0	硬性
Greengrass CLI	>=2.1.0 <2.10.0	硬性

2.2.6

下表列出了此组件版本 2.2.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.9.0	硬性
Greengrass CLI	>=2.1.0 <2.9.0	硬性

2.2.5

下表列出了此组件版本 2.2.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.8.0	硬性

依赖关系	兼容版本	依赖关系类型
Greengrass CLI	>=2.1.0 <2.8.0	硬性

2.2.4

下表列出了此组件版本 2.2.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.7.0	硬性
Greengrass CLI	>=2.1.0 <2.7.0	硬性

2.2.3

下表列出了此组件版本 2.2.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.6.0	硬性
Greengrass CLI	>=2.1.0 <2.6.0	硬性

2.2.2

下表列出了此组件版本 2.2.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.5.0	硬性
Greengrass CLI	>=2.1.0 <2.5.0	硬性

2.2.1

下表列出了此组件版本 2.2.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.4.0	硬性
Greengrass CLI	>=2.1.0 <2.4.0	硬性

2.2.0

下表列出了此组件版本 2.2.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.3.0	硬性
Greengrass CLI	>=2.1.0 <2.3.0	硬性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.2.0	硬性
Greengrass CLI	>=2.1.0 <2.2.0	硬性

2.0.x

下表列出了此组件版本 2.0.x 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	软性
Greengrass CLI	>=2.0.3 <2.1.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

v2.1.x - v2.4.x

`httpsEnabled`

（可选）您可以为本地调试控制台启用 HTTPS 通信。如果启用 HTTPS 通信，则本地调试控制台会创建自签名证书。Web 浏览器会为使用自签名证书的网站显示安全警告，因此您必须手动验证证书。然后，你可以绕过警告。有关更多信息，请参阅 使用量。

默认值：`true`

`port`

（可选）提供本地调试控制台的端口。

默认：`1441`

`websocketPort`

（可选）用于本地调试控制台的 websocket 端口。

默认：`1442`

`bindHostname`

（可选）用于本地调试控制台的主机名。

如果您在 Docker 容器中运行 C AWS IoT Greengrass ore 软件，请将此参数设置为`0.0.0.0`，这样您就可以在 Docker 容器之外打开本地调试控制台。

默认：`localhost`

Example 示例：配置合并更新

以下示例配置指定在非默认端口上打开本地调试控制台并禁用 HTTPS。

```
{
  "httpsEnabled": false,
  "port": "10441",
  "websocketPort": "10442"
```

```
}
```

v2.0.x

`port`

（可选）提供本地调试控制台的端口。

默认：`1441`

`websocketPort`

（可选）用于本地调试控制台的 websocket 端口。

默认：`1442`

`bindHostname`

（可选）用于本地调试控制台的主机名。

如果您在 Docker 容器中运行 C AWS IoT Greengrass ore 软件，请将此参数设置为`0.0.0.0`，这样您就可以在 Docker 容器之外打开本地调试控制台。

默认：`localhost`

Example 示例：配置合并更新

以下示例配置指定在非默认端口上打开本地调试控制台。

```
{
  "port": "10441",
  "websocketPort": "10442"
}
```

使用量

要使用本地调试控制台，请通过 Greengrass CLI 创建会话。创建会话时，Greengrass CLI 会提供一个用户名和临时密码，您可以使用这些用户名和临时密码登录本地调试控制台。

按照以下说明在核心设备或开发计算机上打开本地调试控制台。

v2.1.x - v2.4.x

在 2.1.0 及更高版本中，本地调试控制台默认使用 HTTPS。启用 HTTPS 后，本地调试控制台会创建自签名证书来保护连接。由于此自签名证书，当您打开本地调试控制台时，您的 Web 浏览器会显示安全警告。使用 Greengrass CLI 创建会话时，输出包括证书的指纹，因此您可以验证证书是否合法以及连接是否安全。

您可以禁用 HTTPS。有关更多信息，请参阅本地调试控制台配置。

打开本地调试控制台

1. （可选）要在开发计算机上查看本地调试控制台，可以通过 SSH 转发控制台的端口。但是，您必须先在核心设备的 SSH 配置文件中启用该`AllowTcpForwarding`选项。默认情况下，此选项处于启用状态。在开发计算机上运行以下命令，以在开发计算机`localhost:1441`上查看仪表板。

   ```
   ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
   ```

   > **Note**
   >
   > 您可以从1441和更改默认端口1442。有关更多信息，请参阅本地调试控制台配置。

2. 创建会话以使用本地调试控制台。创建会话时，会生成一个用于进行身份验证的密码。本地调试控制台需要密码才能提高安全性，因为您可以使用此组件在核心设备上查看重要信息并执行操作。如果您在组件配置中启用 HTTPS，则本地调试控制台还会创建证书来保护连接。默认情况下，HTTPS 处于启用状态。

   使用 C AWS IoT Greengrass LI 创建会话。此命令生成一个 43 个字符的随机密码，该密码将在 8 小时后过期。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass V2 根文件夹的路径。

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli get-debug-password
   ```

   Windows

   ```
   C:\greengrass\v2\bin\greengrass-cli get-debug-password
   ```

如果您已将本地调试控制台配置为使用 HTTPS，则命令输出类似于以下示例。打开本地调试控制台时，您可以使用证书指纹来验证连接是否安全。

```
Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is
 self-signed so you will need to bypass your web browser's security warnings to
 open the console.
Before you bypass the security warning, verify that the certificate fingerprint
 matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67
 96 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

调试视图组件创建一个持续 8 小时的会话。之后，必须生成新密码才能再次查看本地调试控制台。

3. 打开并登录控制面板。在 Greengrass 核心设备上查看控制面板，如果您通过 SSH 转发端口，则可以在开发计算机上查看控制面板。请执行以下操作之一：

   - 如果您在本地调试控制台中启用了 HTTPS（默认设置），请执行以下操作：

     a. 在核心设备`https://localhost:1441`上打开，如果通过 SSH 转发端口，则在开发计算机上打开。

        您的浏览器可能会显示有关安全证书无效的安全警告。

     b. 如果您的浏览器显示安全警告，请验证证书是否合法并绕过安全警告。执行以下操作：

        i. 找到证书的 SHA-256 或 SHA-1 指纹，并验证它是否与该`get-debug-password`命令之前打印的 SHA-256 或 SHA-1 指纹相匹配。您的浏览器可能会提供一个或两个指纹。要查看证书及其指纹，请查阅浏览器的文档。在某些浏览器中，证书指纹被称为指纹。

           Note

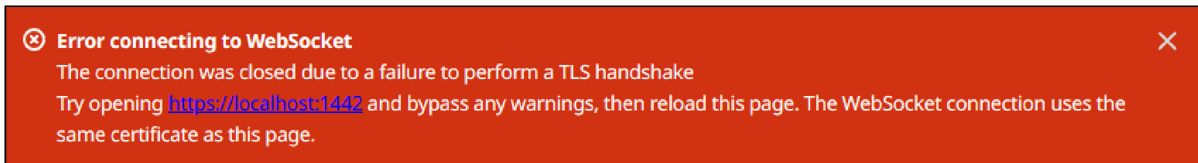
           如果证书指纹不匹配，请转Step 2至创建新会话。如果证书指纹仍然不匹配，则您的连接可能不安全。

ii. 如果证书指纹匹配，请绕过浏览器的安全警告，打开本地调试控制台。请查阅浏览器的文档以绕过浏览器安全警告。

c. 使用`get-debug-password`命令之前打印的用户名和密码登录网站。

本地调试控制台打开。

d. 如果本地调试控制台显示错误提示由于 TLS 握手失败 WebSocket 而无法连接，则必须绕过该 URL 的自签名安全警告。 WebSocket



执行以下操作：

i. `https://localhost:1442`在打开本地调试控制台的同一浏览器中打开。

ii. 验证证书并绕过安全警告。

绕过警告后，您的浏览器可能会显示 HTTP 404 页面。

iii. `https://localhost:1441`再次打开。

本地调试控制台显示有关核心设备的信息。

- 如果您在本地调试控制台中禁用了 HTTPS，请执行以下操作：

a. 在核心设备`http://localhost:1441`上打开，或者如果您通过 SSH 转发端口，则在开发计算机上将其打开。

b. 使用该`get-debug-password`命令之前打印的用户名和密码登录网站。

本地调试控制台打开。

v2.0.x

打开本地调试控制台

1. （可选）要在开发计算机上查看本地调试控制台，可以通过 SSH 转发控制台的端口。但是，您必须先在核心设备的 SSH 配置文件中启用该`AllowTcpForwarding`选项。默认情况下，此选项处于启用状态。在开发计算机上运行以下命令，以在开发计算机`localhost:1441`上查看仪表板。

```
ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
```

> **Note**
>
> 您可以从1441和更改默认端口1442。有关更多信息，请参阅本地调试控制台配置。

2. 创建会话以使用本地调试控制台。创建会话时，会生成一个用于进行身份验证的密码。本地调试控制台需要密码才能提高安全性，因为您可以使用此组件在核心设备上查看重要信息并执行操作。

   使用 C AWS IoT Greengrass LI 创建会话。此命令生成一个 43 个字符的随机密码，该密码将在 8 小时后过期。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass V2 根文件夹的路径。

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli get-debug-password
   ```

   Windows

   ```
   C:\greengrass\v2\bin\greengrass-cli get-debug-password
   ```

   命令输出类似于以下示例。

   ```
   Username: debug
   Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
   Password will expire at: 2021-04-01T17:01:43.921999931-07:00
   ```

   调试视图组件创建一个持续 4 小时的会话，然后您必须生成新密码才能再次查看本地调试控制台。

3. 在核心设备`http://localhost:1441`上打开，或者如果您通过 SSH 转发端口，则在开发计算机上将其打开。

4. 使用该`get-debug-password`命令之前打印的用户名和密码登录网站。

   本地调试控制台打开。

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.4.2	错误修复和改进 • 常规错误修复和性能改进。
2.4.1	Greengrass nucleus 版本 2.12.0 版本的版本已更新。

版本	更改
2.4.0	新功能 • 添加直播管理器调试控制台。
2.3.1	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.3.0	Greengrass nucleus 版本 2.10.0 版本的版本已更新。 新功能 • 包括 PubSub 和 AWS IoT Core MQTT 调试客户端。
2.2.7	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.2.6	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.2.5	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.2.4	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.2.3	错误修复和改进 • 修复了当组件无法解密保存 SSL 私钥的密钥库时无法启动的问题。 • Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.2.2	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.2.1	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.2.0	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.1.0	新功能 • 使用 HTTPS 保护您与本地调试控制台的连接。默认情况下，HTTPS 处于启用状态。 错误修复和改进 • 你可以在配置编辑器中关闭闪存栏消息。
2.0.3	初始版本。

日志管理器

日志管理器组件 (`aws.greengrass.LogManager`) 将日志从AWS IoT Greengrass核心设备上传到 Amazon CloudWatch 日志。你可以上传来自 Greengrass 核心、其他 Greengrass 组件以及其他不是 Greengrass 组件的应用程序和服务的日志。有关如何监控日志和本地文件系统上的 CloudWatch 日志的更多信息，请参阅监控AWS IoT Greengrass日志。

使用日志管理器组件写入日志时，需要考虑以下注意事项： CloudWatch

- 日志延迟

Note

我们建议您升级到日志管理器版本 2.3.0，该版本可减少轮换和活动日志文件的日志延迟。当你升级到日志管理器 2.3.0 时，我们建议你同时升级到 Greengrass nucleus 2.9.1。

日志管理器组件版本 2.2.8（及更早版本）仅处理和上传轮换日志文件中的日志。默认情况下，AWS IoT GreengrassCore 软件每小时或在 1,024 KB 之后轮换一次日志文件。因此，只有在 C AWS IoT Greengrass ore 软件或 Greengrass 组件写入价值超过 1,024 KB 的日志之后，日志管理器组件才会上传日志。您可以配置较低的日志文件大小限制，以使日志文件更频繁地轮换。这会导致日志管理器组件更频繁地将 CloudWatch 日志上传到日志。

日志管理器组件版本 2.3.0（及更高版本）处理并上传所有日志。当您写入新日志时，日志管理器版本 2.3.0（及更高版本）会处理并直接上传该活动日志文件，而不是等待其轮换。这意味着您可以在 5 分钟或更短的时间内查看新日志。

日志管理器组件会定期上传新日志。默认情况下，日志管理器组件每 5 分钟上传一次新日志。您可以配置较低的上传间隔，以便日志管理器组件通过配置来更频繁地将 CloudWatch 日志上传到日志。`periodicUploadIntervalSec`有关如何配置此周期间隔的更多信息，请参阅配置。

日志可以近乎实时地从同一 Greengrass 文件系统上传。如果您需要实时观察日志，请考虑使用文件系统日志。

Note

如果您使用不同的文件系统写入日志，则日志管理器会恢复到日志管理器组件版本 2.2.8 及更早版本中的行为。有关访问文件系统日志的信息，请参阅访问文件系统日志。

- 时钟偏差

  日志管理器组件使用标准的签名版本 4 签名流程来创建对 CloudWatch 日志的 API 请求。如果核心设备上的系统时间不同步超过 15 分钟，则 CloudWatch Logs 会拒绝请求。有关更多信息，请参阅《AWS 一般参考》中的签名版本 4 签名流程。

有关此组件向其上传日志的日志组和日志流的信息，请参阅使用量。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 使用量
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- Greengrass 设备角色必须允许logs:CreateLogGroup、、logs:DescribeLogStreams和操作logs:CreateLogStreamlogs:PutLogEvents，如以下示例 IAM 策略所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

> Note
>
> 默认情况下，您在安装 Core 软件时创建的 Greengrass 设备角色包含AWS IoT Greengrass 此示例策略中的权限。

有关更多信息，请参阅 A ma CloudWatch zon Logs 用户指南中的使用基于身份的 CloudWatch 日志策略（IAM 策略）。

- 支持在 VPC 中运行日志管理器组件。要在 VPC 中部署此组件，需要满足以下条件。
  - 日志管理器组件必须连接到 `logs.region.amazonaws.com` VPC 终端节点为`com.amazonaws.us-east-1.logs`。

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
`logs.*region*.amazonaws.com`	443	否	如果将日志写入日CloudWatch 志，则为必填项。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.3.7

下表列出了此组件版本 2.3.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.13.0	软性

2.3.5 and 2.3.6

下表列出了此组件版本 2.3.5 和 2.3.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.12.0	软性

2.3.3 – 2.3.4

下表列出了此组件版本 2.3.3 到 2.3.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.11.0	软性

2.2.8 – 2.3.2

下表列出了此组件版本 2.2.8 到 2.3.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.10.0	软性

2.2.7

下表列出了此组件版本 2.2.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.9.0	软性

2.2.6

下表列出了此组件版本 2.2.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.8.0	软性

2.2.5

下表列出了此组件版本 2.2.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.7.0	软性

2.2.1 - 2.2.4

下表列出了此组件版本 2.2.1-2.2.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.6.0	软性

2.1.3 and 2.2.0

下表列出了此组件版本 2.1.3 和 2.2.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.5.0	软性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.4.0	软性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.3.0	软性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.1.0 <2.2.0	软性

2.0.x

下表列出了此组件版本 2.0.x 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

v2.3.6 – v2.3.7

`logsUploaderConfiguration`

（可选）日志管理器组件上传的日志配置。该对象包含以下信息：

`systemLogsConfiguration`

（可选）AWS IoT Greengrass核心软件系统日志的配置，其中包括来自 Greengrass 核心的日志和插件组件。指定此配置以使日志管理器组件能够管理系统日志。该对象包含以下信息：

`uploadToCloudWatch`

（可选）您可以将系统日志上传到 CloudWatch 日志。

默认值：`false`

`minimumLogLevel`

（可选）要上传的最低日志消息级别。只有当您将 Greengrass nucleus 组件配置为输出 JSON 格式的日志时，此最低级别才适用。要启用 JSON 格式日志，请指定`JSON`日志格式参数 (`logging.format`)。

从以下日志级别中进行选择，此处按级别顺序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

默认值：`INFO`

`diskSpaceLimit`

（可选）Greengrass 系统日志文件的最大总大小，以您在中指定的单位为单位。`diskSpaceLimitUnit`在 Greengrass 系统日志文件的总大小超过此最大总大小后，核心软件会删除最旧的 Greengrass 系统日志文件。AWS IoT Greengrass

此参数等同于 Greengrass nucleus 组件的日志大小限制参数 (`totalLogsSizeKB`)。AWS IoT GreengrassCore 软件使用两个值中的最小值作为 Greengrass 系统日志的最大总大小。

`diskSpaceLimitUnit`

（可选）的单位`diskSpaceLimit`。从以下选项中进行选择：

- `KB`— 千字节
- `MB`— 兆字节
- `GB`— 千兆字节

默认值：`KB`

`deleteLogFileAfterCloudUpload`

（可选）在日志管理器组件将日志上传到 CloudWatch 日志后，您可以删除日志文件。

默认值：`false`

`componentLogsConfigurationMap`

（可选）核心设备上组件的日志配置地图。此映射中的每个`componentName`对象都定义了组件或应用程序的日志配置。日志管理器组件将这些组件日志上传到日 CloudWatch 志。

> **Important**
>
> 我们强烈建议每个组件使用一个配置密钥。您只应将一组文件作为目标，这些文件只有一个在使用时正在写入的日志文件`logFileRegex`。不遵循此建议可能会导致重复的日志上传到 CloudWatch。如果您使用单个正则表达式定位多个活动日志文件，我们建议您升级到日志管理器 v2.3.1 或更高版本，并考虑使用示例配置更改配置。

> **Note**
>
> 如果您要从 v2.2.0 之前的日志管理器版本升级，则可以继续使用该`componentLogsConfiguration`列表，而不是。`componentLogsConfigurationMap`但是，我们强烈建议您使用地图格式，以便您可以使用合并和重置更新来修改特定组件的配置。有关该`componentLogsConfiguration`参数的信息，请参阅此组件 v2.1.x 的配置参数。

`componentName`

此日志配置的*`componentName`*组件或应用程序的日志配置。您可以指定 Greengrass 组件的名称或其他值来标识此日志组。

每个对象都包含以下信息：

`minimumLogLevel`

（可选）要上传的最低日志消息级别。仅当此组件的日志使用特定的 JSON 格式时，此最低级别才适用，您可以在AWS IoT Greengrass日志模块存储库中找到该格式 GitHub。

从以下日志级别中进行选择，此处按级别顺序列出：

- `DEBUG`
- `INFO`

- WARN
- ERROR

默认值：INFO

diskSpaceLimit

（可选）此组件所有日志文件的最大总大小，以您在中指定的单位为单位diskSpaceLimitUnit。此组件日志文件的总大小超过此最大总大小后，AWS IoT GreengrassCore 软件将删除该组件最旧的日志文件。

此参数与 Greengrass nucleus totalLogsSizeKB 组件的日志大小限制参数 () 有关。AWS IoT GreengrassCore 软件使用两个值中的最小值作为该组件的最大总日志大小。

diskSpaceLimitUnit

（可选）的单位diskSpaceLimit。从以下选项中进行选择：

- KB— 千字节
- MB— 兆字节
- GB— 千兆字节

默认值：KB

logFileDirectoryPath

（可选）包含此组件日志文件的文件夹的路径。

对于打印到标准输出 (stdout) 和标准错误 (stderr) 的 Greengrass 组件，您无需指定此参数。

默认值：*/greengrass/v2*/logs。

logFileRegex

（可选）一个正则表达式，用于指定组件或应用程序使用的日志文件名格式。日志管理器组件使用此正则表达式来识别位于的文件夹中的日志文件logFileDirectoryPath。

对于打印到标准输出 (stdout) 和标准错误 (stderr) 的 Greengrass 组件，您无需指定此参数。

如果您的组件或应用程序轮换日志文件，请指定与轮换的日志文件名相匹配的正则表达式。例如，您可以指定**hello_world\\\\w*.log**上传 Hello World 应用程序的日

志。该\\\\w*模式匹配零个或多个单词字符，其中包括字母数字字符和下划线。此正则表达式匹配名称中带有和不带时间戳的日志文件。在此示例中，日志管理器上传以下日志文件：

- hello_world.log— Hello World 应用程序的最新日志文件。
- hello_world_2020_12_15_17_0.log— Hello World 应用程序的旧日志文件。

默认:*componentName*\\\\w*.log，其中 *componentN* ame 是此日志配置的组件的名称。

deleteLogFileAfterCloudUpload

（可选）在日志管理器组件将日志上传到 CloudWatch 日志后，您可以删除日志文件。

默认值：false

multiLineStartPattern

（可选）一个正则表达式，用于标识新行上的日志消息何时为新日志消息。如果正则表达式与新行不匹配，则日志管理器组件会将新行附加到上一行的日志消息中。

默认情况下，日志管理器组件会检查该行是否以空格字符（例如制表符或空格）开头。如果不是，则日志管理器会将该行作为新的日志消息处理。否则，它会将该行附加到当前日志消息中。此行为可确保日志管理器组件不会拆分跨多行的消息，例如堆栈跟踪。

periodicUploadIntervalSec

（可选）日志管理器组件检查要上传的新日志文件的时间段（以秒为单位）。

默认：300（5 分钟）

最小值：0.000001（1 微秒）

deprecatedVersionSupport

表示日志管理器是否应使用日志管理器 v2.3.5 中引入的日志速度改进。将值设置为false以使用这些改进。

如果将此值设置为从日志管理器 v2.3.1 或更早版本升级false时，可能会上传重复的日志条目。

默认值为 true。

Example 示例：配置合并更新

以下示例配置指定将系统日志和`com.example.HelloWorld`组件日志上传到日 CloudWatch 志。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300",
  "deprecatedVersionSupport": "false"
}
```

Example 示例：使用日志管理器 v2.3.1 上传多个活动日志文件的配置

如果您想将多个活动日志文件作为目标，则推荐使用以下示例配置。此示例配置指定了您要上传到哪些活动日志文件 CloudWatch。使用此配置示例配置还将上传与匹配的所有旋转文件。`logFileRegex`日志管理器 v2.3.1 支持此示例配置。

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
```

```
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.3.x

`logsUploaderConfiguration`

（可选）日志管理器组件上传的日志配置。该对象包含以下信息：

`systemLogsConfiguration`

（可选）AWS IoT Greengrass核心软件系统日志的配置，其中包括来自 Greengrass 核心的日志和插件组件。指定此配置以使日志管理器组件能够管理系统日志。该对象包含以下信息：

`uploadToCloudWatch`

（可选）您可以将系统日志上传到 CloudWatch 日志。

默认值：`false`

`minimumLogLevel`

（可选）要上传的最低日志消息级别。只有当您将 Greengrass nucleus 组件配置为输出 JSON 格式的日志时，此最低级别才适用。要启用 JSON 格式日志，请指定`JSON`日志格式参数 (`logging.format`)。

从以下日志级别中进行选择，此处按级别顺序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

默认值：`INFO`

`diskSpaceLimit`

（可选）Greengrass 系统日志文件的最大总大小，以您在中指定的单位为单位。`diskSpaceLimitUnit`在 Greengrass 系统日志文件的总大小超过此最大总大小后，核心软件会删除最旧的 Greengrass 系统日志文件。AWS IoT Greengrass

此参数等同于 Greengrass nucleus 组件的日志大小限制参数 (totalLogsSizeKB)。AWS IoT GreengrassCore 软件使用两个值中的最小值作为 Greengrass 系统日志的最大总大小。

`diskSpaceLimitUnit`

（可选）的单位diskSpaceLimit。从以下选项中进行选择：

- KB— 千字节
- MB— 兆字节
- GB— 千兆字节

默认值：KB

`deleteLogFileAfterCloudUpload`

（可选）在日志管理器组件将日志上传到 CloudWatch 日志后，您可以删除日志文件。

默认值：false

`componentLogsConfigurationMap`

（可选）核心设备上组件的日志配置地图。此映射中的每个componentName对象都定义了组件或应用程序的日志配置。日志管理器组件将这些组件日志上传到日 CloudWatch 志。

> **Important**
>
> 我们强烈建议每个组件使用一个配置密钥。您只应将一组文件作为目标，这些文件只有一个在使用时正在写入的日志文件logFileRegex。不遵循此建议可能会导致重复的日志上传到 CloudWatch。如果您使用单个正则表达式定位多个活动日志文件，我们建议您升级到日志管理器 v2.3.1，并考虑使用示例配置更改配置。

> **Note**
>
> 如果您要从 v2.2.0 之前的日志管理器版本升级，则可以继续使用该componentLogsConfiguration列表，而不是。componentLogsConfigurationMap但是，我们强烈建议您使用地图格式，以便您可以使用合并和重置更新来修改特定组件的配置。有关该componentLogsConfiguration参数的信息，请参阅此组件 v2.1.x 的配置参数。

componentName

此日志配置的*componentName*组件或应用程序的日志配置。您可以指定 Greengrass 组件的名称或其他值来标识此日志组。

每个对象都包含以下信息：

`minimumLogLevel`

（可选）要上传的最低日志消息级别。仅当此组件的日志使用特定的 JSON 格式时，此最低级别才适用，您可以在AWS IoT Greengrass日志模块存储库中找到该格式 GitHub。

从以下日志级别中进行选择，此处按级别顺序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

默认值：`INFO`

`diskSpaceLimit`

（可选）此组件所有日志文件的最大总大小，以您在中指定的单位为单位`diskSpaceLimitUnit`。此组件日志文件的总大小超过此最大总大小后，AWS IoT GreengrassCore 软件将删除该组件最旧的日志文件。

此参数与 Greengrass nucleus `totalLogsSizeKB` 组件的日志大小限制参数 () 有关。AWS IoT GreengrassCore 软件使用两个值中的最小值作为该组件的最大总日志大小。

`diskSpaceLimitUnit`

（可选）的单位`diskSpaceLimit`。从以下选项中进行选择：

- `KB`— 千字节
- `MB`— 兆字节
- `GB`— 千兆字节

默认值：`KB`

`logFileDirectoryPath`

（可选）包含此组件日志文件的文件夹的路径。

对于打印到标准输出 (stdout) 和标准错误 (stderr) 的 Greengrass 组件，您无需指定此参数。

默认值：`/greengrass/v2/logs`。

`logFileRegex`

（可选）一个正则表达式，用于指定组件或应用程序使用的日志文件名格式。日志管理器组件使用此正则表达式来识别位于的文件夹中的日志文件`logFileDirectoryPath`。

对于打印到标准输出 (stdout) 和标准错误 (stderr) 的 Greengrass 组件，您无需指定此参数。

如果您的组件或应用程序轮换日志文件，请指定与轮换的日志文件名相匹配的正则表达式。例如，您可以指定**`hello_world\\\\w*.log`**上传 Hello World 应用程序的日志。该`\\\\w*`模式匹配零个或多个单词字符，其中包括字母数字字符和下划线。此正则表达式匹配名称中带有和不带时间戳的日志文件。在此示例中，日志管理器上传以下日志文件：

- `hello_world.log`— Hello World 应用程序的最新日志文件。
- `hello_world_2020_12_15_17_0.log`— Hello World 应用程序的旧日志文件。

默认:*`componentName`*`\\\\w*.log`，其中 *`componentN`*ame 是此日志配置的组件的名称。

`deleteLogFileAfterCloudUpload`

（可选）在日志管理器组件将日志上传到 CloudWatch 日志后，您可以删除日志文件。

默认值：`false`

`multiLineStartPattern`

（可选）一个正则表达式，用于标识新行上的日志消息何时为新日志消息。如果正则表达式与新行不匹配，则日志管理器组件会将新行附加到上一行的日志消息中。

默认情况下，日志管理器组件会检查该行是否以空格字符（例如制表符或空格）开头。如果不是，则日志管理器会将该行作为新的日志消息处理。否则，它会将该行附

加到当前日志消息中。此行为可确保日志管理器组件不会拆分跨多行的消息，例如堆栈跟踪。

`periodicUploadIntervalSec`

（可选）日志管理器组件检查要上传的新日志文件的时间段（以秒为单位）。

默认：300（5 分钟）

最小值：0.000001（1 微秒）

Example 示例：配置合并更新

以下示例配置指定将系统日志和`com.example.HelloWorld`组件日志上传到日 CloudWatch 志。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

Example 示例：使用日志管理器 v2.3.1 上传多个活动日志文件的配置

如果您想将多个活动日志文件作为目标，则推荐使用以下示例配置。此示例配置指定了您要上传到哪些活动日志文件 CloudWatch。使用此配置示例配置还将上传与匹配的所有旋转文件。`logFileRegex`日志管理器 v2.3.1 支持此示例配置。

```
{
```

```
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.2.x

`logsUploaderConfiguration`

（可选）日志管理器组件上传的日志配置。该对象包含以下信息：

`systemLogsConfiguration`

（可选）AWS IoT Greengrass核心软件系统日志的配置，其中包括来自 Greengrass 核心的日志和插件组件。指定此配置以使日志管理器组件能够管理系统日志。该对象包含以下信息：

`uploadToCloudWatch`

（可选）您可以将系统日志上传到 CloudWatch 日志。

默认值：`false`

`minimumLogLevel`

（可选）要上传的最低日志消息级别。只有当您将 Greengrass nucleus 组件配置为输出 JSON 格式的日志时，此最低级别才适用。要启用 JSON 格式日志，请指定JSON日志格式参数 (`logging.format`)。

从以下日志级别中进行选择，此处按级别顺序列出：

- `DEBUG`
- `INFO`
- `WARN`

- ERROR

默认值：INFO

diskSpaceLimit

（可选）Greengrass 系统日志文件的最大总大小，以您在中指定的单位为单位。diskSpaceLimitUnit在 Greengrass 系统日志文件的总大小超过此最大总大小后，核心软件会删除最旧的 Greengrass 系统日志文件。AWS IoT Greengrass

此参数等同于 Greengrass nucleus 组件的日志大小限制参数 (totalLogsSizeKB)。AWS IoT GreengrassCore 软件使用两个值中的最小值作为 Greengrass 系统日志的最大总大小。

diskSpaceLimitUnit

（可选）的单位diskSpaceLimit。从以下选项中进行选择：

- KB— 千字节
- MB— 兆字节
- GB— 千兆字节

默认值：KB

deleteLogFileAfterCloudUpload

（可选）在日志管理器组件将日志上传到 CloudWatch 日志后，您可以删除日志文件。

默认值：false

componentLogsConfigurationMap

（可选）核心设备上组件的日志配置地图。此映射中的每个componentName对象都定义了组件或应用程序的日志配置。日志管理器组件将这些组件日志上传到日 CloudWatch 志。

> **Note**
>
> 如果您要从 v2.2.0 之前的日志管理器版本升级，则可以继续使用该componentLogsConfiguration列表，而不是。componentLogsConfigurationMap但是，我们强烈建议您使用地图格式，以便您可以使用合并和重置更新来修改特定组件的配置。有关该componentLogsConfiguration参数的信息，请参阅此组件 v2.1.x 的配置参数。

componentName

此日志配置的*componentName*组件或应用程序的日志配置。您可以指定 Greengrass 组件的名称或其他值来标识此日志组。

每个对象都包含以下信息：

`minimumLogLevel`

（可选）要上传的最低日志消息级别。仅当此组件的日志使用特定的 JSON 格式时，此最低级别才适用，您可以在AWS IoT Greengrass日志模块存储库中找到该格式 GitHub。

从以下日志级别中进行选择，此处按级别顺序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

默认值：`INFO`

`diskSpaceLimit`

（可选）此组件所有日志文件的最大总大小，以您在中指定的单位为单位`diskSpaceLimitUnit`。此组件日志文件的总大小超过此最大总大小后，AWS IoT GreengrassCore 软件将删除该组件最旧的日志文件。

此参数与 Greengrass nucleus `totalLogsSizeKB` 组件的日志大小限制参数 () 有关。AWS IoT GreengrassCore 软件使用两个值中的最小值作为该组件的最大总日志大小。

`diskSpaceLimitUnit`

（可选）的单位`diskSpaceLimit`。从以下选项中进行选择：

- `KB`— 千字节
- `MB`— 兆字节
- `GB`— 千兆字节

默认值：`KB`

`logFileDirectoryPath`

（可选）包含此组件日志文件的文件夹的路径。

对于打印到标准输出 (stdout) 和标准错误 (stderr) 的 Greengrass 组件，您无需指定此参数。

默认值：*/greengrass/v2*/logs。

`logFileRegex`

（可选）一个正则表达式，用于指定组件或应用程序使用的日志文件名格式。日志管理器组件使用此正则表达式来识别位于的文件夹中的日志文件`logFileDirectoryPath`。

对于打印到标准输出 (stdout) 和标准错误 (stderr) 的 Greengrass 组件，您无需指定此参数。

如果您的组件或应用程序轮换日志文件，请指定与轮换的日志文件名相匹配的正则表达式。例如，您可以指定**`hello_world\\\\w*.log`**上传 Hello World 应用程序的日志。该`\\\\w*`模式匹配零个或多个单词字符，其中包括字母数字字符和下划线。此正则表达式匹配名称中带有和不带时间戳的日志文件。在此示例中，日志管理器上传以下日志文件：

- `hello_world.log`— Hello World 应用程序的最新日志文件。
- `hello_world_2020_12_15_17_0.log`— Hello World 应用程序的旧日志文件。

默认:*componentName*`\\\\w*.log`，其中 *componentN* ame 是此日志配置的组件的名称。

`deleteLogFileAfterCloudUpload`

（可选）在日志管理器组件将日志上传到 CloudWatch 日志后，您可以删除日志文件。

默认值：`false`

`multiLineStartPattern`

（可选）一个正则表达式，用于标识新行上的日志消息何时为新日志消息。如果正则表达式与新行不匹配，则日志管理器组件会将新行附加到上一行的日志消息中。

默认情况下，日志管理器组件会检查该行是否以空格字符（例如制表符或空格）开头。如果不是，则日志管理器会将该行作为新的日志消息处理。否则，它会将该行附加到当前日志消息中。此行为可确保日志管理器组件不会拆分跨多行的消息，例如堆栈跟踪。

`periodicUploadIntervalSec`

（可选）日志管理器组件检查要上传的新日志文件的时间段（以秒为单位）。

默认：300（5 分钟）

最小值：0.000001（1 微秒）

Example 示例：配置合并更新

以下示例配置指定将系统日志和`com.example.HelloWorld`组件日志上传到日 CloudWatch 志。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.1.x

`logsUploaderConfiguration`

（可选）日志管理器组件上传的日志配置。该对象包含以下信息：

`systemLogsConfiguration`

（可选）AWS IoT Greengrass核心软件系统日志的配置，其中包括来自 Greengrass 核心的日志和插件组件。指定此配置以使日志管理器组件能够管理系统日志。该对象包含以下信息：

`uploadToCloudWatch`

（可选）您可以将系统日志上传到 CloudWatch 日志。

默认值：`false`

`minimumLogLevel`

（可选）要上传的最低日志消息级别。只有当您将 Greengrass nucleus 组件配置为输出 JSON 格式的日志时，此最低级别才适用。要启用 JSON 格式日志，请指定`JSON`日志格式参数 (`logging.format`)。

从以下日志级别中进行选择，此处按级别顺序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

默认值：`INFO`

`diskSpaceLimit`

（可选）Greengrass 系统日志文件的最大总大小，以您在中指定的单位为单位。`diskSpaceLimitUnit`在 Greengrass 系统日志文件的总大小超过此最大总大小后，核心软件会删除最旧的 Greengrass 系统日志文件。AWS IoT Greengrass

此参数等同于 Greengrass nucleus 组件的日志大小限制参数 (`totalLogsSizeKB`)。AWS IoT GreengrassCore 软件使用两个值中的最小值作为 Greengrass 系统日志的最大总大小。

`diskSpaceLimitUnit`

（可选）的单位`diskSpaceLimit`。从以下选项中进行选择：

- `KB`— 千字节
- `MB`— 兆字节
- `GB`— 千兆字节

默认值：`KB`

`deleteLogFileAfterCloudUpload`

（可选）在日志管理器组件将日志上传到 CloudWatch 日志后，您可以删除日志文件。

默认值：`false`

`componentLogsConfiguration`

（可选）核心设备上组件的日志配置列表。此列表中的每个配置都定义了组件或应用程序的日志配置。日志管理器组件将这些组件日志上传到 CloudWatch Logs

每个对象都包含以下信息：

`componentName`

此日志配置的组件或应用程序的名称。您可以指定 Greengrass 组件的名称或其他值来标识此日志组。

`minimumLogLevel`

（可选）要上传的最低日志消息级别。仅当此组件的日志使用特定的 JSON 格式时，此最低级别才适用，您可以在AWS IoT Greengrass日志模块存储库中找到该格式 GitHub。

从以下日志级别中进行选择，此处按级别顺序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

默认值：`INFO`

`diskSpaceLimit`

（可选）此组件所有日志文件的最大总大小，以您在中指定的单位为单位`diskSpaceLimitUnit`。此组件日志文件的总大小超过此最大总大小后，AWS IoT GreengrassCore 软件将删除该组件最旧的日志文件。

此参数与 Greengrass nucleus `totalLogsSizeKB` 组件的日志大小限制参数 () 有关。AWS IoT GreengrassCore 软件使用两个值中的最小值作为该组件的最大总日志大小。

`diskSpaceLimitUnit`

（可选）的单位`diskSpaceLimit`。从以下选项中进行选择：

- `KB`— 千字节
- `MB`— 兆字节

- GB— 千兆字节

默认值：KB

logFileDirectoryPath

（可选）包含此组件日志文件的文件夹的路径。

对于打印到标准输出 (stdout) 和标准错误 (stderr) 的 Greengrass 组件，您无需指定此参数。

默认值：*/greengrass/v2*/logs。

logFileRegex

（可选）一个正则表达式，用于指定组件或应用程序使用的日志文件名格式。日志管理器组件使用此正则表达式来识别位于的文件夹中的日志文件logFileDirectoryPath。

对于打印到标准输出 (stdout) 和标准错误 (stderr) 的 Greengrass 组件，您无需指定此参数。

如果您的组件或应用程序轮换日志文件，请指定与轮换的日志文件名相匹配的正则表达式。例如，您可以指定**hello_world\\\\w*.log**上传 Hello World 应用程序的日志。该\\\\w*模式匹配零个或多个单词字符，其中包括字母数字字符和下划线。此正则表达式匹配名称中带有和不带时间戳的日志文件。在此示例中，日志管理器上传以下日志文件：

- hello_world.log— Hello World 应用程序的最新日志文件。
- hello_world_2020_12_15_17_0.log— Hello World 应用程序的旧日志文件。

默认:*componentName*\\\\w*.log，其中 *componentN* ame 是此日志配置的组件的名称。

deleteLogFileAfterCloudUpload

（可选）在日志管理器组件将日志上传到 CloudWatch 日志后，您可以删除日志文件。

默认值：false

multiLineStartPattern

（可选）一个正则表达式，用于标识新行上的日志消息何时为新日志消息。如果正则表达式与新行不匹配，则日志管理器组件会将新行附加到上一行的日志消息中。

默认情况下，日志管理器组件会检查该行是否以空格字符（例如制表符或空格）开头。如果不是，则日志管理器会将该行作为新的日志消息处理。否则，它会将该行附加到当前日志消息中。此行为可确保日志管理器组件不会拆分跨多行的消息，例如堆栈跟踪。

`periodicUploadIntervalSec`

（可选）日志管理器组件检查要上传的新日志文件的时间段（以秒为单位）。

默认：300（5 分钟）

最小值：`0.000001`（1 微秒）

Example 示例：配置合并更新

以下示例配置指定将系统日志和`com.example.HelloWorld`组件日志上传到日 CloudWatch 志。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.0.x

`logsUploaderConfiguration`

（可选）日志管理器组件上传的日志配置。该对象包含以下信息：

`systemLogsConfiguration`

（可选）AWS IoT GreengrassCore 软件系统日志的配置。指定此配置以使日志管理器组件能够管理系统日志。该对象包含以下信息：

`uploadToCloudWatch`

（可选）您可以将系统日志上传到 CloudWatch 日志。

默认值：`false`

`minimumLogLevel`

（可选）要上传的最低日志消息级别。只有当您将 Greengrass nucleus 组件配置为输出 JSON 格式的日志时，此最低级别才适用。要启用 JSON 格式日志，请指定`JSON`日志格式参数 (`logging.format`)。

从以下日志级别中进行选择，此处按级别顺序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

默认值：`INFO`

`diskSpaceLimit`

（可选）Greengrass 系统日志文件的最大总大小，以您在中指定的单位为单位。`diskSpaceLimitUnit`在 Greengrass 系统日志文件的总大小超过此最大总大小后，核心软件会删除最旧的 Greengrass 系统日志文件。AWS IoT Greengrass

此参数等同于 Greengrass nucleus 组件的日志大小限制参数 (`totalLogsSizeKB`)。AWS IoT GreengrassCore 软件使用两个值中的最小值作为 Greengrass 系统日志的最大总大小。

`diskSpaceLimitUnit`

（可选）的单位`diskSpaceLimit`。从以下选项中进行选择：

- `KB`— 千字节
- `MB`— 兆字节
- `GB`— 千兆字节

默认值：`KB`

`deleteLogFileAfterCloudUpload`

（可选）在日志管理器组件将日志上传到 CloudWatch 日志后，您可以删除日志文件。

默认值：`false`

`componentLogsConfiguration`

（可选）核心设备上组件的日志配置列表。此列表中的每个配置都定义了组件或应用程序的日志配置。日志管理器组件将这些组件日志上传到 CloudWatch Logs

每个对象都包含以下信息：

`componentName`

此日志配置的组件或应用程序的名称。您可以指定 Greengrass 组件的名称或其他值来标识此日志组。

`minimumLogLevel`

（可选）要上传的最低日志消息级别。仅当此组件的日志使用特定的 JSON 格式时，此最低级别才适用，您可以在AWS IoT Greengrass日志模块存储库中找到该格式 GitHub。

从以下日志级别中进行选择，此处按级别顺序列出：

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

默认值：`INFO`

`diskSpaceLimit`

（可选）此组件所有日志文件的最大总大小，以您在中指定的单位为单位`diskSpaceLimitUnit`。此组件日志文件的总大小超过此最大总大小后，AWS IoT GreengrassCore 软件将删除该组件最旧的日志文件。

此参数与 Greengrass nucleus `totalLogsSizeKB` 组件的日志大小限制参数 () 有关。AWS IoT GreengrassCore 软件使用两个值中的最小值作为该组件的最大总日志大小。

`diskSpaceLimitUnit`

（可选）的单位`diskSpaceLimit`。从以下选项中进行选择：

- KB— 千字节
- MB— 兆字节
- GB— 千兆字节

默认值：KB

`logFileDirectoryPath`

包含此组件日志文件的文件夹的路径。

要上传 Greengrass 组件的日志，***/greengrass/v2*/logs**请指定并替换*/greengrass/v2*为您的 Greengrass 根文件夹。

`logFileRegex`

一种正则表达式，用于指定组件或应用程序使用的日志文件名格式。日志管理器组件使用此正则表达式来识别位于的文件夹中的日志文件`logFileDirectoryPath`。

要上传 Greengrass 组件的日志，请指定与轮换后的日志文件名相匹配的正则表达式。例如，您可以指定**`com.example.HelloWorld\\w*.log`**上传 Hello World 组件的日志。该`\\w*`模式匹配零个或多个单词字符，其中包括字母数字字符和下划线。此正则表达式匹配名称中带有和不带时间戳的日志文件。在此示例中，日志管理器上传以下日志文件：

- `com.example.HelloWorld.log`— Hello World 组件的最新日志文件。
- `com.example.HelloWorld_2020_12_15_17_0.log`— Hello World 组件的旧日志文件。Greengrass nucleus 在日志文件中添加了一个旋转的时间戳。

`deleteLogFileAfterCloudUpload`

（可选）在日志管理器组件将日志上传到 CloudWatch 日志后，您可以删除日志文件。

默认值：`false`

`multiLineStartPattern`

（可选）一个正则表达式，用于标识新行上的日志消息何时为新日志消息。如果正则表达式与新行不匹配，则日志管理器组件会将新行附加到上一行的日志消息中。

默认情况下，日志管理器组件会检查该行是否以空格字符（例如制表符或空格）开头。如果不是，则日志管理器会将该行作为新的日志消息处理。否则，它会将该行附加到当前日志消息中。此行为可确保日志管理器组件不会拆分跨多行的消息，例如堆栈跟踪。

`periodicUploadIntervalSec`

（可选）日志管理器组件检查要上传的新日志文件的时间段（以秒为单位）。

默认：300（5 分钟）

最小值：0.000001（1 微秒）

Example 示例：配置合并更新

以下示例配置指定将系统日志和com.example.HelloWorld组件日志上传到日 CloudWatch 志。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "logFileDirectoryPath": "/greengrass/v2/logs",
        "logFileRegex": "com.example.HelloWorld\\w*.log",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

使用量

日志管理器组件上传到以下日志组和日志流。

2.1.0 and later

日志组名称

```
/aws/greengrass/componentType/region/componentName
```

日志组名称使用以下变量：

- componentType— 组件的类型，可以是以下类型之一：
  - GreengrassSystemComponent— 此日志组包括核心和插件组件的日志，它们与 Greengrass 核心在同一 JVM 中运行。该组件是 Greengrass 核的一部分。
  - UserComponent— 此日志组包括设备上通用组件、Lambda 组件和其他应用程序的日志。该组件不是 Greengrass 核的一部分。

  有关更多信息，请参阅 组件类型。
- region— 核心设备使用的AWS区域。
- componentName— 组件的名称。对于系统日志，此值为System。

日志流名称

```
/date/thing/thingName
```

日志流名称使用以下变量：

- date— 日志的日期，例如2020/12/15。日志管理器组件使用该yyyy/MM/dd格式。
- thingName— 核心设备的名称。

Note

如果事物名称包含冒号 (:)，则日志管理器会将冒号替换为加号 (+)。

2.0.x

日志组名称

```
/aws/greengrass/componentType/region/componentName
```

日志组名称使用以下变量：

- componentType— 组件的类型，可以是以下类型之一：
  - GreengrassSystemComponent— 该组件是 Greengrass 核的一部分。
  - UserComponent— 该组件不是 Greengrass 核的一部分。日志管理器将这种类型用于设备上的 Greengrass 组件和其他应用程序。
- region— 核心设备使用的AWS区域。

- componentName— 组件的名称。对于系统日志，此值为System。

日志流名称

```
/date/deploymentTargets/thingName
```

日志流名称使用以下变量：

- date— 日志的日期，例如2020/12/15。日志管理器组件使用该yyyy/MM/dd格式。
- deploymentTargets— 部署的内容包括组件。日志管理器组件用斜线分隔每个目标。如果在本地部署后组件在核心设备上运行，则此值为LOCAL_DEPLOYMENT。

  举一个例子，你有一个名为的核心设备MyGreengrassCore，而核心设备有两个部署：

  - 以核心设备为目标的部署，MyGreengrassCore。
  - 一种以名为的事物组为目标的部署MyGreengrassCoreGroup，该组包含核心设备。

  这个核心设备是thing/MyGreengrassCore/thinggroup/MyGreengrassCoreGroup。deploymentTargets
- thingName— 核心设备的名称。

日志条目的格式。

Greengrass 核心以字符串或 JSON 格式写入日志文件。对于系统日志，您可以通过设置logging条目的format字段来控制格式。你可以在 Greengrass nucleus 组件的配置文件中找到该logging条目。有关更多信息，请参阅 Greengrass 核配置。

文本格式为自由格式，接受任何字符串。以下舰队状态服务消息是字符串格式日志的示例：

```
2023-03-26T18:18:27.271Z [INFO] (pool-1-thread-2)
com.aws.greengrass.status.FleetStatusService: fss-status-update-published.
Status update published to FSS. {trigger=CADENCE, serviceName=FleetStatusService,
currentState=RUNNING}
```

如果您想使用 Greengrass CLI 日志命令查看日志或以编程方式与日志进行交互，则应使用 JSON 格式。以下示例概述了 JSON 形状：

```
{
   "loggerName": <string>,
   "level": <"DEBUG" | "INFO" | "ERROR" | "TRACE" | "WARN">,
```

```
    "eventType": <string, optional>,
    "cause": <string, optional>,
    "contexts": {},
    "thread": <string>,
    "message": <string>,
    "timestamp": <epoch time> # Needs to be epoch time
}
```

要控制组件日志的输出，您可以使用minimumLogLevel配置选项。要使用此选项，您的组件必须以 JSON 格式写入其日志条目。您应使用与系统日志文件相同的格式。

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.3.7	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.3.6	错误修复和改进 • 调整某些错误的日志级别。
2.3.5	改进 提高了日志上传速度。 Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.3.4	错误修复和改进 • 增加了对将`periodicUploadIntervalSec` 参数设置为小数值的支持。最小值为 1 微秒。 • 修复了日志管理器不遵守 CloudWatch`putLogEvents` 限制的问题。
2.3.3	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.3.2	错误修复和改进 • 改进了空间管理，因此日志文件在上传之前不会被删除。 • 修复了缓存管理问题。 • 其他小错误修复和改进。
2.3.1	错误修复和改进 • 修复了具有多个活动日志文件的目标文件组将重复条目上传到 CloudWatch的问题。 • 其他小错误修复和改进。
2.3.0	**Note** 当你升级到日志管理器 2.3.0 时，我们建议你升级到 Greengrass nucleus 2.9.1。

版本	更改
	新功能 通过处理和直接上传活动日志文件而不是等待轮换新文件来减少日志延迟。 错误修复和改进 • 改进了在轮换具有唯一名称的文件时对日志轮换的支持。 • 其他小错误修复和改进。
2.2.8	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.2.7	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.2.6	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.2.5	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.2.4	错误修复和改进 • 提高了处理无效配置时的稳定性。 • 其他小修复和改进。
2.2.3	错误修复和改进 • 提高了组件重启或遇到错误的某些情况下的稳定性。 • 修复了在某些情况下无法上传大型日志消息和大型日志文件的问题。 • 修复了此组件如何处理配置重置更新的问题。 • 修复了`nulldiskSpaceLimit` 配置值导致组件无法部署的问题。
2.2.2	错误修复和改进 • 添加对大于 256 KB 的日志消息的支持。日志管理器组件将这些大型日志消息拆分为多条具有相同日志事件时间戳的消息。
2.2.1	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.2.0	新特征 • 添加`componentLogsConfigurationMap` 配置参数以支持组件日志配置的映射格式。地图中的每个`componentName` 对象都定义了组件或应用程序的日志配置。

版本	更改
2.1.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.1.1	错误修复和改进 • 修复了在某些情况下系统日志配置未更新的问题。
2.1.0	错误修复和改进 • 使用适用于打印到标准输出 (stdout) `logFileDirectoryPath` 和`logFileRegex` 标准错误 (stderr) 的 Greengrass 组件的默认值。 • 将日志上传到 CloudWatch 日志时，通过配置的网络代理正确路由流量。 • 正确处理日志流名称中的冒号字符 (`:`)。 CloudWatch 日志日志流名称不支持冒号。 • 通过从日志流中删除事物组名称来简化日志流名称。 • 删除在正常行为期间打印的错误日志消息。
2.0.x	初始版本。

机器学习组件

AWS IoT Greengrass提供了以下机器学习组件，您可以将这些组件部署到支持的设备上，以便使用在 Amazon 中训练的模型 SageMaker 或存储在 Amazon S3 中的您自己的预训练模型来执行机器学习推理。

AWS提供以下类别的机器学习组件：

- 模型组件-包含作为 Greengrass 工件的机器学习模型。
- 运行时组件-包含用于在 Greengrass 核心设备上安装机器学习框架及其依赖关系的脚本。
- 推理组件-包含推理代码并包括组件依赖项，用于安装机器学习框架和下载预训练的机器学习模型。

您可以使用AWS提供的机器学习组件中的示例推理代码和预训练模型，使用 DLR 和 Lite 执行图像分类和目标检测。 TensorFlow 要使用存储在 Amazon S3 中的您自己的模型执行自定义机器学习推理，或者使用其他机器学习框架，您可以使用这些公共组件的配方作为模板来创建自定义机器学习组件。有关更多信息，请参阅 自定义您的机器学习组件。

AWS IoT Greengrass还包括一个AWS提供的组件，用于管理 Greengrass 核心设备上 SageMaker 边缘管理器代理的安装和生命周期。借助 SageMaker Edge Manager，您可以直接在核心设备上使用 Amazon SageMaker Neo 编译的模型。有关更多信息，请参阅 在 Green SageMaker grass 核心设备上使用亚马逊 Edge Manager。

下表列出了中可用的机器学习组件AWS IoT Greengrass。

Note

AWS提供的几个组件依赖于 Greengrass 核的特定次要版本。由于这种依赖关系，当你将 Greengrass nucleus 更新到新的次要版本时，你需要更新这些组件。有关每个组件所依赖的原子核的特定版本的信息，请参阅相应的组件主题。有关更新原子核的更多信息，请参见更新 AWS IoT Greengrass核心软件 (OTA)。

当组件的组件类型同时为泛型和 Lambda 时，该组件的当前版本为泛型类型，而该组件的先前版本为 Lambda 类型。

组件	描述	组件类型	支持的操作系统	开源
Lookout for Vision Edge Agent	在 Greengrass 核心设备上部署 Amazon Lookout for Vision 运行时，因此您可以使用计算机视觉来查找工业产品中的缺陷。	通用	Linux	否
SageMaker 边缘管理器	在 Green SageMaker grass 核心设备上部署 Amazon Edge	通用	Linux、Windows	否

组件	描述	组件类型	支持的操作系统	开源
	Manager 代理。			
DLR 图像分类	推理组件，使用 DLR 图像分类模型存储和 DLR 运行时组件作为依赖项，在支持的设备上安装 DLR、下载样本图像分类模型和执行图像分类推理。	通用	Linux、Windows	否
DLR 物体检测	推理组件，使用 DLR 对象检测模型存储和 DLR 运行时组件作为依赖项，用于在支持的设备上安装 DLR、下载示例对象检测模型和执行对象检测推理。	通用	Linux、Windows	否
DLR 图像分类模型存储	包含作为 Greengrass 伪影的样本 ResNet -50 图像分类模型的模型组件。	通用	Linux、Windows	否

组件	描述	组件类型	支持的操作系统	开源
DLR 物体检测模型存储	包含样本 YOLOv3 对象检测模型的模型组件，例如 Greengrass 工件。	通用	Linux、Windows	否
DLR 运行时	包含安装脚本的运行时组件，该脚本用于在 Greengrass 核心设备上安装 DLR 及其依赖关系。	通用	Linux、Windows	否
TensorFlow 精简版图像分类	推理组件，使用 TensorFlow Lite 图像分类模型存储和 TensorFlow Lite 运行时组件作为依赖项，用于在支持的设备上安装 TensorFlow Lite、下载样本图像分类模型和执行图像分类推理。	通用	Linux、Windows	否

组件	描述	组件类型	支持的操作系统	开源
TensorFlow 精简版物体检测	推理组件，使用 TensorFlow Lite 对象检测模型存储和 TensorFlow Lite 运行时组件作为依赖项，用于在支持的设备上安装 TensorFlow Lite、下载示例对象检测模型和执行对象检测推理。	通用	Linux、Windows	否
TensorFlow 精简版图像分类模型存储	包含作为 Greengrass 工件的示例 MobileNet v1 模型的模型组件。	通用	Linux、Windows	否
TensorFlow 精简版物体检测模型存储	模型组件，其中包含作为 Greengrass 工件的样本单枪检测 (SSD) MobileNet 模型。	通用	Linux、Windows	否

组件	描述	组件类型	支持的操作系统	开源
TensorFlow 精简版运行时	包含用于安装 TensorFlow Lite 的安装脚本及其对 Greengrass 核心设备的依赖关系的运行时组件。	通用	Linux、Windows	否

Lookout for Vision Edge Agent

Lookout for Vision Edge Agent 组件 `aws.iot.lookoutvision.EdgeAgent` () 安装本地亚马逊 Lookout for Vision 运行时服务器，该服务器使用计算机视觉来查找工业产品中的视觉缺陷。

要使用此组件，请创建并部署 Lookout for Vision 机器学习模型组件。这些机器学习模型通过查找用于训练模型的图像中的模式来预测图像中是否存在异常。然后，您可以开发和部署自定义 Greengrass 组件（称为客户端应用程序组件），这些组件为该运行时组件提供图像和视频流，以便使用机器学习模型检测异常。

你可以使用 Lookout for Vision Edge Agent API 与其他 Greengrass 组件中的该组件进行交互。此 API 是使用 gRPC 实现的，gRPC 是一种用于进行远程过程调用的协议。有关更多信息，请参阅亚马逊 Lookout f or Vision 开发者指南中的编写客户端应用程序组件和 Lookout for Visi on Edge Agent API 参考。

有关如何使用此组件的更多信息，请参阅以下内容：

- 在 Greengrass 核心设备上使用 Amazon Lookout for Vision fo
- 什么是 Amazon Lookout for Vision？ 在亚马逊 Lookout for Vision 开发者指南中
- 在亚马逊 Lookout for Vision 开发者指南中创建 Lookout for Vision 模型。
- 在《亚马逊 Lookout for Vision 开发者指南》中的边缘设备上使用 Lookout for Vision 模型。

> Note
>
> Lookout for Vision Edge Agent 组件仅在以下版本中 AWS 区域可用：

- 美国东部（俄亥俄）
- 美国东部（弗吉尼亚州北部）
- US West（Oregon）
- 欧洲地区（法兰克福）
- 欧洲（爱尔兰）
- 亚太地区（东京)
- 亚太地区 (首尔)

主题

- 版本
- Type
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 1.2.x
- 1.1.x
- 1.0.x
- 0.1.x

Type

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

- Greengrass 核心设备必须使用 Armv8 (aarch64) 或 x86_64 架构。
- 如果你使用此组件的 1.0.0 或更高版本，则安装在 Greengrass 核心设备上的 Py th on 3.8 或 Python 3.9（包括`pip`）。

  如果你使用此组件的 0.1.x 版本，则安装在 Greengrass 核心设备上的 Python 3.7（包括`pip`）。

  > Important
  >
  > 该设备必须具有这些 Python 版本中的一个。此组件不支持更高版本的 Python。

- 要使用图形处理单元 (GPU) 推理，核心设备必须满足以下要求。在此组件的 1.1.0 及更高版本中，GPU 推断是可选的。
  - 支持 CUDA 的图形处理单元 (GPU)。有关更多信息，请参阅 CUD A 工具包文档中的验证您拥有支持 CUD A 的 GPU。
  - Cudnn、CUDA 和 Tensorrt 安装在 Greengrass 核心设备上。
    - 在 NVIDIA Jetson 设备上，例如 Jetson Nano 或 Jetson Xavier，cuda 和 Tensorrt 都安装了 NVIDIA。 JetPack您无需进行任何更改。此组件支持 JetPack 4.4、JetPack 4. 5、JetPack 4.5.1 和 JetPack4.6. 1。

      > Important
      >
      > 您必须安装其中一个版本而不是另一个版本。 JetPack Lookout for Vision 服务为这些平台编译计算机视觉模型。 JetPack

    - 在配备采用 NVIDIA Ampere 微架构（或 GPU 的计算容量为 8.0）的 GPU 的 x86 设备上，请执行以下操作：
      - 按照 N VIDIA cuDNN 安装指南中的说明安装 cu DNN。
      - 按照适用于 Linux 的 NVIDIA CUDA 安装指南中的说明安装 CUDA 版本 11.2。
      - 按照 NVIDIA Tensorrt 文档中的说明安装 TensorRT 版本 8.2.0。

  - 在 x86 设备上，GPU 的架构早于 Ampere 的 NVIDIA 架构（或者 GPU 的计算容量低于 8.0），请执行以下操作：
    - 按照 N VIDIA cuDNN 安装指南中的说明安装 cu DNN。
    - 按照适用于 Linux 的 NVIDIA CUDA 安装指南中的说明安装 CUDA 版本 10.2。
    - 按照 NVIDIA Tensorrt 文档中的说明安装 TensorRT 版本 7.1.3 或更高版本，但早于 8.0.0 版本。
- 运行此组件的系统用户必须是有权访问设备上的 GPU 的系统组的成员。该组的名称因操作系统而异。请查阅您的操作系统和 GPU 的文档，以确定该系统组的名称。

  例如，在 NVIDIA Jetson 设备上`video`，该组的名称为，您可以运行以下命令将系统用户添加到该组。将 *ggc_user* 替换为要添加的用户名。

  ```
  sudo usermod -aG video ggc_user
  ```

依赖项

这个组件没有任何依赖关系。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`Socket`

（可选）运行 Edge 代理的文件套接字。Lookout for Vision 模型组件使用此文件套接字与 Edge Agent 通信。如果更改此参数，则在部署 Lookout for Vision 模型组件时必须指定相同的值。

默认：`unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock`

本地日志文件

此组件使用以下日志文件。

```
/greengrass/v2/logs/aws.iot.lookoutvision.EdgeAgent.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。*/greengrass/v2*替换为 AWS IoT Greengrass 根文件夹的路径。

```
sudo tail -f /greengrass/v2/logs/aws.iot.lookoutvision.EdgeAgent.log
```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
1.2.0	常规错误修复和性能改进。
1.1.9	常规错误修复和性能改进。
1.1.8	常规错误修复和性能改进。
1.1.7	新功能 • 在 Lookout for Vision Edge Agent 虚拟环境中安装`opencv-python-headless` 软件包。 错误修复和改进 • 改进了置信度分数的计算。 • 将热图模型掩码的大小调整为原始文件大小。 • 常规错误修复和性能改进。
1.1.6	新功能 为`DetectAnomalies` 结果添加了新值。 • `anomaly_score` — 介于 0.0 和 1.0 之间的数字，表示图像的异常程度。 • `anomaly_threshold` — 在模型训练期间设置的阈值，用于确定异常图像和普通图像之间的边界。 常规错误修复和性能改进。

版本	更改
flink-client	新功能 增加了对 OpenCV 的支持，以便在可用时调整图像大小。当 OpenCV 不可用时，边缘代理会使用 Pillow。 错误修复和改进 常规错误修复和性能改进。
1.1.3	常规错误修复和性能改进。
1.1.1	常规错误修复和性能改进。
1.1.0	新功能 • 增加了对图像分割模型的支持，该模型可以识别图像中的异常。 • 增加了对 CPU 推断的支持，因此你可以在没有 GPU 的核心设备上使用 Lookout for Vision 模型。 错误修复和改进 • 常规错误修复和性能改进。
1.0.0	此版本的 Lookout for Vision Edge Agent 组件需要与 0.1.x 版本不同的 Python 版本。如果要从 v0.1.x 升级到 1.x 版本，则必须升级核心设备上的 Python 安装。 错误修复和改进 • 常规错误修复和性能改进。
0.1.37	常规错误修复和性能改进。
0.1.36	初始版本。

SageMaker 边缘管理器

Important

SageMaker Edge Manager 将于 2024 年 4 月 26 日停产。有关继续将模型部署到边缘设备的更多信息，请参阅 Edg SageMaker e Manager 生命周期终止。

Amazon SageMaker Edge Manager 组件 (`aws.greengrass.SageMakerEdgeManager`) 安装 SageMaker 边缘管理器代理二进制文件。

SageMaker Edge Manager 为边缘设备提供模型管理，因此您可以优化、保护、监控和维护边缘设备队列上的机器学习模型。边 SageMaker 缘管理器组件在您的核心设备上安装和管理边 SageMaker 缘管理器代理的生命周期。您还可以使用 SageMaker Edge Manager 在 Greengrass 核心设备上打包和使用 SageMaker Neo 编译的模型作为模型组件。有关在核心设备上使用 SageMaker Edge Manager 代理的更多信息，请参阅在 Green SageMaker grass 核心设备上使用亚马逊 Edge Manager。

SageMaker Edge Manager 组件 v1.3.x 安装边缘管理器代理二进制 v1.20220822.836f3023。有关 Edge Manager 代理二进制版本的更多信息，请参阅边缘管理器代理。

Note

边 SageMaker 缘管理器组件仅在以下版本中可用AWS 区域：

- 美国东部（俄亥俄）
- 美国东部（弗吉尼亚州北部）
- 美国西部（俄勒冈州）
- 欧洲（法兰克福）
- 欧洲（爱尔兰）
- 亚太地区（东京）

主题

- 版本
- 类型
- 操作系统
- 要求

- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 1.3.x
- 1.2.x
- 1.1.x
- 1.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 在亚马逊 Linux 2、基于 Debian 的 Linux 平台（x86_64 或 Armv8）或 Windows（x86_64）上运行的 Greengrass 核心设备。如果没有，请参阅教程：AWS IoT Greengrass V2 入门。
- 安装在核心设备上的 Pyth@@ on 3.6 或更高版本，包括`pip`适用于你的 Python 版本。
- Greengrass 设备角色配置如下：
  - 一种允许`credentials.iot.amazonaws.com`和代`sagemaker.amazonaws.com`入角色的信任关系，如以下 IAM 策略示例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- I AmazonSageMakerEdgeDeviceFleetPolicyAM 托管策略。
- `s3:PutObject`操作，如以下 IAM 策略示例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- 与您的 Greengrass 核心设备AWS 区域相同AWS 账户且创建的 Amazon S3 存储桶。 SageMaker Edge Manager 需要一个 S3 存储桶来创建边缘设备队列，并存储在设备上运行推理的示例数据。有关创建 S3 存储桶的信息，请参阅 Amazon S3 入门。

- 使用与 Greengrass 核心设备相同的AWS IoT角色别名的 SageMaker 边缘设备队列。有关更多信息，请参阅 创建边缘设备队列。
- 您的 Greengrass 核心设备在您的 Edge 设备群中注册为边缘设备。 SageMaker 边缘设备名称必须与核心设备AWS IoT的事物名称相匹配。有关更多信息，请参阅 注册你的 Greengrass 核心设备。

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
`edge.sagemaker.` *`region`* `.amazonaws.com`	443	是	检查设备注册状态并将指标发送到 SageMaker。
`*.s3.amazonaws.com`	443	是	将捕获数据上传到您指定的 S3 存储桶。 您可以将*替换为上传数据的每个存储桶的名称。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

1.3.5

下表列出了此组件版本 1.3.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性
代币兑换服务	>=0.0.0	硬性

1.3.4

下表列出了此组件版本 1.3.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性
代币兑换服务	>=0.0.0	硬性

1.3.3

下表列出了此组件版本 1.3.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性
代币兑换服务	>=0.0.0	硬性

1.3.2

下表列出了此组件版本 1.3.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性

依赖关系	兼容版本	依赖关系类型
代币兑换服务	>=0.0.0	硬性

1.3.1

下表列出了此组件版本 1.3.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性
代币兑换服务	>=0.0.0	硬性

1.1.1 - 1.3.0

下表列出了此组件版本 1.1.1-1.3.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性
代币兑换服务	>=0.0.0	硬性

1.1.0

下表列出了此组件版本 1.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性
代币兑换服务	>=0.0.0	硬性

1.0.3

下表列出了此组件版本 1.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性
代币兑换服务	>=0.0.0	硬性

1.0.1 and 1.0.2

下表列出了此组件版本 1.0.1 和 1.0.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性
代币兑换服务	>=0.0.0	硬性

1.0.0

下表列出了此组件版本 1.0.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性
代币兑换服务	>=0.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

> **Note**
>
> 本节介绍您在组件中设置的配置参数。有关相应 SageMaker 边缘管理器配置的更多信息，请参阅《亚马逊 SageMaker 开发者指南》中的 Edge Manager 代理。

`DeviceFleetName`

包含你的 Green SageMaker grass 核心设备的 Edge Manager 设备队列的名称。

部署此组件时，必须在配置更新中为此参数指定一个值。

`BucketName`

将捕获的推理数据上传到的 S3 存储桶的名称。存储桶名称必须包含字符串`sagemaker`。

如果设置`CaptureDataDestination`为`Cloud`，或者设置`CaptureDataPeriodicUpload`为`true`，则在部署此组件时，必须在配置更新中为此参数指定一个值。

> **Note**
>
> 捕获数据是一项用于将推理输入、推理结果和其他推理数据上传到 S3 存储桶或本地目录以供将来分析的 SageMaker 功能。有关在 SageMaker Edge Manager 中使用捕获数据的更多信息，请参阅 Amazon SageMaker 开发者指南中的管理模型。

`CaptureDataBatchSize`

（可选）代理处理的一批捕获数据请求的大小。此值必须小于您在中指定的缓冲区大小`CaptureDataBufferSize`。我们建议不要超过缓冲区大小的一半。

当缓冲区中的请求数量达到该数量时，或者经过`CapturePushPeriodSeconds`间隔时（以先发生者为准），代理会处理请求批处理。`CaptureDataBatchSize`

默认值：`10`

`CaptureDataBufferSize`

（可选）存储在缓冲区中的最大捕获数据请求数。

默认值：`30`

`CaptureDataDestination`

（可选）存储捕获数据的目的地。此参数可以具有以下值：

- `Cloud`— 将捕获的数据上传到您在中指定的 S3 存储桶。`BucketName`

- Disk— 将捕获的数据写入组件的工作目录。

如果您指定Disk，也可以通过将设置CaptureDataPeriodicUpload为来选择定期将捕获的数据上传到您的 S3 存储桶true。

默认值：Cloud

CaptureDataPeriodicUpload

（可选）指定是否定期上传捕获数据的字符串值。支持的值为 true 和 false。

true如果设置为，则将此参数设置CaptureDataDestination为Disk，并且还希望代理定期将捕获的数据上传到您的 S3 存储桶。

默认值：false

CaptureDataPeriodicUploadPeriodSeconds

（可选）SageMaker 边缘管理器代理将捕获的数据上传到 S3 存储桶的时间间隔（以秒为单位）。如果设置CaptureDataPeriodicUpload为，则使用此参数true。

默认值：8

CaptureDataPushPeriodSeconds

（可选）SageMaker Edge Manager 代理处理来自缓冲区的一批捕获数据请求的时间间隔（以秒为单位）。

当缓冲区中的请求数量达到该数量时，或者经过CaptureDataPushPeriodSeconds间隔时（以先发生者为准），代理会处理请求批处理。CaptureDataBatchSize

默认值：4

CaptureDataBase64EmbedLimit

（可选）SageMaker Edge Manager 代理上传的捕获数据的最大大小（以字节为单位）。

默认值：3072

FolderPrefix

（可选）代理将捕获的数据写入的文件夹的名称。如果设置CaptureDataDestination为Disk，则代理将在指定的目录中创建文件

夹`CaptureDataDiskPath`。如果您设置`CaptureDataDestination`为`Cloud`，或者设置为`CaptureDataPeriodicUploadtrue`，则代理将在您的 S3 存储桶中创建文件夹。

默认值：`sme-capture`

`CaptureDataDiskPath`

此功能在 v1.1.0 及更高版本的 SageMaker Edge Manager 组件中可用。

（可选）代理创建捕获的数据文件夹的文件夹路径。如果设置`CaptureDataDestination`为`Disk`，则代理将在此目录中创建捕获的数据文件夹。如果您未指定此值，代理将在组件的工作目录中创建捕获的数据文件夹。使用`FolderPrefix`参数指定捕获的数据文件夹的名称。

默认值：`/greengrass/v2/work/aws.greengrass.SageMakerEdgeManager/capture`

`LocalDataRootPath`

此功能在 v1.2.0 及更高版本的 SageMaker Edge Manager 组件中可用。

（可选）此组件在核心设备上存储以下数据的路径：

- 设置`DbEnable`为时，运行时数据的本地数据库`true`。
- SageMaker 新编译的模型，当您设置`DeploymentEnable`为时，此组件会自动下载这些模型。`true`

默认值：`/greengrass/v2/work/aws.greengrass.SageMakerEdgeManager`

`DbEnable`

（可选）您可以启用此组件以将运行时数据存储在本地数据库中，以便在组件出现故障或设备断电时保留数据。

此数据库需要在核心设备的文件系统上有 5 MB 的存储空间。

默认值：`false`

`DeploymentEnable`

此功能在 v1.2.0 及更高版本的 SageMaker Edge Manager 组件中可用。

（可选）您可以启用此组件以自动检索您上传到 Amazon S3 的 SageMaker Neo 编译模型。将新模型上传到 Amazon S3 后，使用 SageMaker Studio 或 SageMaker API 将新模型部署到该核心设备。启用此功能后，无需创建部署即可将新模型AWS IoT Greengrass部署到核心设备。

> ⚠ Important
>
> 要使用此功能，必须将设置`DbEnable`为`true`。此功能使用本地数据库来跟踪从中检索的 AWS Cloud模型。

默认值：`false`

`DeploymentPollInterval`

此功能在 v1.2.0 及更高版本的 SageMaker Edge Manager 组件中可用。

（可选）此组件检查要下载的新模型的间隔时间（以分钟为单位）。当您设置为时，此选项适`DeploymentEnable`用`true`。

默认：`1440`（1 天）

`DLRBackendOptions`

此功能在 v1.2.0 及更高版本的 SageMaker Edge Manager 组件中可用。

（可选）要在此组件使用的 DLR 运行时中设置的 DLR 运行时标志。您可以设置以下标志：

- `TVM_TENSORRT_CACHE_DIR`— 启用 Tensorrt 模型缓存。指定具有读/写权限的现有文件夹的绝对路径。
- `TVM_TENSORRT_CACHE_DISK_SIZE_MB`— 分配 Tensorrt 模型缓存文件夹的上限。当目录大小超过此限制时，使用最少的缓存引擎将被删除。默认值为 512 MB。

例如，您可以将此参数设置为以下值以启用 Tensorrt 模型缓存并将缓存大小限制为 800 MB。

```
TVM_TENSORRT_CACHE_DIR=/data/secured_folder/trt/cache;
 TVM_TENSORRT_CACHE_DISK_SIZE_MB=800
```

`SagemakerEdgeLogVerbose`

（可选）指定是否启用调试日志记录的字符串值。支持的值为 `true` 和 `false`。

默认值：`false`

`UnixSocketName`

（可选） SageMaker Edge Manager 套接字文件描述符在核心设备上的位置。

默认值：`/tmp/aws.greengrass.SageMakerEdgeManager.sock`

Example 示例：配置合并更新

以下示例配置指定核心设备是的一部分，*MyEdgeDeviceFleet*并且代理将捕获数据写入设备和 S3 存储桶。此配置还启用调试日志记录。

```
{
    "DeviceFleetName": "MyEdgeDeviceFleet",
    "BucketName": "DOC-EXAMPLE-BUCKET",
    "CaptureDataDestination": "Disk",
    "CaptureDataPeriodicUpload": "true",
    "SagemakerEdgeLogVerbose": "true"
}
```

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log -Tail
   10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
1.3.5	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
1.3.4	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
1.3.3	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
1.3.2	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
1.3.1	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
1.3.0	新功能 • 增加了对 Tensorrt 缓存磁盘大小管理的支持。 • 在 DLR BackendOptions 参数中添加可选`TVM_TENSORRT_CACHE_DISK_SIZE_MB` 标志，以设置磁盘上缓存模型的大小限制。 改进 • 提供改进的预测并发性。这有助于更好地使用设备加速器引擎，例如 GPU。
1.2.0	新功能 • 添加对该组件的支持，以自动检索您上传到 Amazon S3 的 SageMaker Neo 编译模型。启用此功能后，无需创建部署即可将新模型AWS IoT Greengrass部署到核心设备。 • 添加了对备份数据库的支持，该组件使用该数据库来保存运行时数据，以防组件出现故障或设备断电。 • 支持您在配置此组件时配置 DLR 运行时标志。
1.1.1	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
1.1.0	新功能 • 增加了对运行亚马逊 Linux 2 的 Greengrass 核心设备的支持。 • 添加新的`CaptureDataDiskPath` 配置参数。您可以使用此参数来指定设备上捕获的数据文件夹的路径。

版本	更改
	错误修复和改进 • Greengrass nucleus 版本 2.5.0 版本的版本已更新。
1.0.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
1.0.2	错误修复和改进 更新组件生命周期中的安装脚本。在部署此组件之前，您的核心设备现在必须在设备上安装 Python 3.6 或更高版本（包括`pip`您的 Python 版本）。
1.0.1	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
1.0.0	初始版本。

DLR 图像分类

DLR 图像分类组件 (`aws.greengrass.DLRImageClassification`) 包含示例推理代码，用于使用深度学习运行时和 resnet-50 模型执行图像分类推理。此组件使用变体DLR 图像分类模型存储和DLR 运行时组件作为依赖项来下载 DLR 和示例模型。

要将此推理组件与自定义训练的 DLR 模型一起使用，请创建依赖模型存储组件的自定义版本。要使用自己的自定义推理代码，您可以使用此组件的配方作为模板来创建自定义推理组件。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x
- 2.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令来安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上进行升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

启用旧版相机堆栈

1. 运行以下命令打开 Raspberry Pi 配置工具。

```
sudo raspi-config
```

2. 选择 “接口选项”。
3. 选择旧版相机以启用旧版相机堆栈。
4. 重启 Raspberry Pi。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.13

下表列出了此组件版本 2.1.13 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性
DLR 图像分类模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.12

下表列出了此组件版本 2.1.12 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性
DLR 图像分类模型存储	~2.1.0	硬性

依赖关系	兼容版本	依赖关系类型
德国航空航天中心	~1.6.0	硬性

2.1.11

下表列出了此组件版本 2.1.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性
DLR 图像分类模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.10

下表列出了此组件版本 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性
DLR 图像分类模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.9

下表列出了此组件版本 2.1.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性
DLR 图像分类模型存储	~2.1.0	硬性

依赖关系	兼容版本	依赖关系类型
德国航空航天中心	~1.6.0	硬性

2.1.8

下表列出了此组件版本 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性
DLR 图像分类模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性
DLR 图像分类模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性
DLR 图像分类模型存储	~2.1.0	硬性

依赖关系	兼容版本	依赖关系类型
德国航空航天中心	~1.6.0	硬性

2.1.4 - 2.1.5

下表列出了此组件版本 2.1.4 到 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性
DLR 图像分类模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性
DLR 图像分类模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性
DLR 图像分类模型存储	~2.1.0	硬性

依赖关系	兼容版本	依赖关系类型
德国航空航天中心	~1.6.0	硬性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性
DLR 图像分类模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.0.x

下表列出了此组件版本 2.0.x 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	~2.0.0	软性
DLR 图像分类模型存储	~2.0.0	硬性
DLR	~1.3.0	软性

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

2.1.x

`accessControl`

（可选）包含授权策略的对象，该策略允许组件向默认通知主题发布消息。

默认值：

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/dlr/image-classification"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

（可选）您要发布推理结果的主题。如果您修改此值，则还必须修改`accessControl`参数`resources`中的值以匹配您的自定义主题名称。

默认值：`ml/dlr/image-classification`

`Accelerator`

你要使用的加速器。支持的值为 `cpu` 和 `gpu`。

依赖模型组件中的示例模型仅支持 CPU 加速。要在不同的自定义模型中使用 GPU 加速，请创建一个自定义模型组件来覆盖公共模型组件。

默认值：`cpu`

`ImageDirectory`

（可选）设备上推理组件读取图像的文件夹路径。您可以将此值修改为设备上任何您拥有读/写访问权限的位置。

默认值：`/greengrass/v2/packages/artifacts-unarchived/component-name/image_classification/sample_images/`

> **Note**
>
> 如果将的值设置`UseCamera`为`true`，则忽略此配置参数。

`ImageName`

（可选）推理组件用作预测输入的图像的名称。该组件将在中指定的文件夹中查找图像ImageDirectory。默认情况下，该组件使用默认图像目录中的示例图像。 AWS IoT Greengrass支持以下图像格式：jpeg、jpgpng、和npy。

默认值：cat.jpeg

Note

如果将的值设置UseCamera为true，则忽略此配置参数。

`InferenceInterval`

（可选）推理代码每次预测之间的时间（以秒为单位）。示例推理代码无限期运行，并在指定的时间间隔内重复其预测。例如，如果您想使用摄像机拍摄的图像进行实时预测，则可以将其更改为较短的间隔。

默认值：3600

`ModelResourceKey`

（可选）在依赖的公共模型组件中使用的模型。仅当使用自定义组件覆盖公共模型组件时，才修改此参数。

默认值：

```
{
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "windows": "DLR-resnet50-win-cpu-ImageClassification"
}
```

`UseCamera`

（可选）字符串值，用于定义是否使用连接到 Greengrass 核心设备的摄像机的图像。支持的值为 true 和 false。

当您将此值设置为时true，示例推理代码将访问设备上的摄像头，并在本地对捕获的图像运行推理。ImageName和ImageDirectory参数的值将被忽略。确保运行此组件的用户对相机存储捕获图像的位置具有读/写访问权限。

默认值：false

Note

当您查看此组件的配方时，UseCamera配置参数不会出现在默认配置中。但是，在部署组件时，可以在配置合并更新中修改此参数的值。

如果设置为 UseCameratrue，则还必须创建符号链接，以使推理组件能够从运行时组件创建的虚拟环境中访问您的摄像头。有关使用带有示例推理组件的摄像头的更多信息，请参阅更新组件配置。

2.0.x

MLRootPath

（可选）Linux 核心设备上推理组件读取图像和写入推理结果的文件夹路径。您可以将此值修改为设备上运行此组件的用户具有读/写访问权限的任何位置。

默认值：*/greengrass/v2*/work/variant.DLR/greengrass_ml

默认值：*/greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml

Accelerator

你要使用的加速器。支持的值为 cpu 和 gpu。

依赖模型组件中的示例模型仅支持 CPU 加速。要在不同的自定义模型中使用 GPU 加速，请创建一个自定义模型组件来覆盖公共模型组件。

默认值：cpu

ImageName

（可选）推理组件用作预测输入的图像的名称。该组件将在中指定的文件夹中查找图像ImageDirectory。默认位置是*MLRootPath*/images。 AWS IoT Greengrass支持以下图像格式：jpeg、jpgpng、和npy。

默认值：cat.jpeg

InferenceInterval

（可选）推理代码每次预测之间的时间（以秒为单位）。示例推理代码无限期运行，并在指定的时间间隔内重复其预测。例如，如果您想使用摄像机拍摄的图像进行实时预测，则可以将其更改为较短的间隔。

默认值：3600

`ModelResourceKey`

（可选）在依赖的公共模型组件中使用的模型。仅当使用自定义组件覆盖公共模型组件时，才修改此参数。

默认值：

```
armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
```

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将 */greengrass/v2* 或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log -
  Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

<table>
<tr><th>版本</th><th>更改</th></tr>
<tr><td>2.1.13</td><td>Greengrass nucleus 版本 2.12.0 版本的版本已更新。</td></tr>
<tr><td>2.1.12</td><td>Greengrass nucleus 版本 2.11.0 版本的版本已更新。</td></tr>
<tr><td>2.1.11</td><td>Greengrass nucleus 版本 2.10.0 版本的版本已更新。</td></tr>
<tr><td>2.1.10</td><td>Greengrass nucleus 版本 2.9.0 版本的版本已更新。</td></tr>
<tr><td>2.1.9</td><td>Greengrass nucleus 版本 2.8.0 版本的版本已更新。</td></tr>
<tr><td>2.1.8</td><td>Greengrass nucleus 版本 2.7.0 版本的版本已更新。</td></tr>
<tr><td>2.1.7</td><td>Greengrass nucleus 版本 2.6.0 版本的版本已更新。</td></tr>
<tr><td>2.1.6</td><td>Greengrass nucleus 版本 2.5.0 版本的版本已更新。</td></tr>
<tr><td>2.1.5</td><td>组件全部发布AWS 区域。</td></tr>
<tr><td>2.1.4</td><td>Greengrass nucleus 版本 2.4.0 版本的版本已更新。

此版本在欧洲（伦敦）不可用（<code>eu-west-2</code> ）。</td></tr>
<tr><td>2.1.3</td><td>Greengrass nucleus 版本 2.3.0 版本的版本已更新。</td></tr>
<tr><td>2.1.2</td><td>Greengrass nucleus 版本 2.2.0 版本的版本已更新。</td></tr>
<tr><td>2.1.1</td><td>新功能
使用深度学习运行时 v1.6.0。
在 Armv8 (aarch64) 平台上添加对样本图像分类的支持。这扩展了对运行 NVIDIA Jetson 的 Greengrass 核心设备（例如 Jetson Nano）的机器学习支持。
启用摄像头集成以进行样本推理。使用新的<code>UseCamera</code> 配置参数启用示例推理代码，以访问您的 Greengrass 核心设备上的摄像头，并在本地对捕获的图像运行推理。</td></tr>
</table>

版本	更改
	• 添加对发布推理结果的AWS Cloud支持。使用新的PublishResultsOnTopic 配置参数来指定要发布结果的主题。 • 添加新的ImageDirectory 配置参数，使您能够为要对其执行推理的图像指定自定义目录。 错误修复和改进 • 将推理结果写入组件日志文件，而不是单独的推理文件。 • 使用 C AWS IoT Greengrass ore 软件日志模块记录组件输出。 • 使用读AWS IoT Device SDK取组件配置并应用配置更改。
2.0.4	初始版本。

DLR 物体检测

DLR 对象检测组件 (aws.greengrass.DLRObjectDetection) 包含示例推理代码，用于使用深度学习运行时和样本预训练模型执行对象检测推理。此组件使用变体DLR 物体检测模型存储和DLR 运行时组件作为依赖项来下载 DLR 和示例模型。

要将此推理组件与自定义训练的 DLR 模型一起使用，请创建依赖模型存储组件的自定义版本。要使用自己的自定义推理代码，您可以使用此组件的配方作为模板来创建自定义推理组件。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x
- 2.0.x

类型

此组件是一个通用组件 (aws.greengrass.generic)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上进行升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

```
sudo raspi-config
```

2. 选择接口选项。
3. 选择旧版相机以启用旧版相机堆栈。
4. 重启 Raspberry Pi。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.13

下表列出了此组件版本 2.1.13 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.12

下表列出了此组件版本 2.1.12 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.11

下表列出了此组件版本 2.1.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.10

下表列出了此组件版本 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.9

下表列出了此组件版本 2.1.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.8

下表列出了此组件版本 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.4 - 2.1.5

下表列出了此组件版本 2.1.4 到 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性
DLR 目标检测模型存储	~2.1.0	硬性
德国航空航天中心	~1.6.0	硬性

2.0.x

下表列出了此组件版本 2.0.x 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	~2.0.0	软性
DLR 目标检测模型存储	~2.0.0	硬性
DLR	~1.3.0	软性

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

2.1.x

`accessControl`

（可选）包含授权策略的对象，该策略允许组件向默认通知主题发布消息。

默认值：

```
{
    "aws.greengrass.ipc.mqttproxy": {
        "aws.greengrass.DLRObjectDetection:mqttproxy:1": {
            "policyDescription": "Allows access to publish via topic ml/dlr/object-
detection.",
            "operations": [
                "aws.greengrass#PublishToIoTCore"
            ],
```

```
            "resources": [
              "ml/dlr/object-detection"
            ]
          }
        }
      }
```

PublishResultsOnTopic

（可选）您要发布推理结果的主题。如果您修改此值，则还必须修改accessControl参数resources中的值以匹配您的自定义主题名称。

默认值：ml/dlr/object-detection

Accelerator

你要使用的加速器。支持的值为 cpu 和 gpu。

依赖模型组件中的示例模型仅支持 CPU 加速。要在不同的自定义模型中使用 GPU 加速，请创建一个自定义模型组件来覆盖公共模型组件。

默认值：cpu

ImageDirectory

（可选）设备上推理组件读取图像的文件夹路径。您可以将此值修改为设备上您有权读/写访问的任何位置。

默认值：*/greengrass/v2*/packages/artifacts-unarchived/*component-name*/object_detection/sample_images/

> **Note**
>
> 如果将的值设置UseCamera为true，则忽略此配置参数。

ImageName

（可选）推理组件用作预测输入的图像的名称。该组件将在中指定的文件夹中查找图像ImageDirectory。默认情况下，该组件使用默认图像目录中的示例图像。 AWS IoT Greengrass支持以下图像格式：jpeg、jpgpng、和npy。

默认值：objects.jpg

Note

如果将的值设置UseCamera为true，则忽略此配置参数。

InferenceInterval

（可选）推理代码每次预测之间的时间（以秒为单位）。示例推理代码无限期运行，并在指定的时间间隔内重复其预测。例如，如果您想使用摄像机拍摄的图像进行实时预测，则可以将其更改为较短的间隔。

默认值：3600

ModelResourceKey

（可选）在依赖的公共模型组件中使用的模型。仅当使用自定义组件覆盖公共模型组件时，才修改此参数。

默认值：

```
{
    "armv7l": "DLR-yolo3-armv7l-cpu-ObjectDetection",
    "aarch64": "DLR-yolo3-aarch64-gpu-ObjectDetection",
    "x86_64": "DLR-yolo3-x86_64-cpu-ObjectDetection",
    "windows": "DLR-resnet50-win-cpu-ObjectDetection"
}
```

UseCamera

（可选）字符串值，用于定义是否使用连接到 Greengrass 核心设备的摄像机的图像。支持的值为 true 和 false。

将此值设置为时true，示例推理代码将访问设备上的摄像头，并在本地对捕获的图像运行推理。ImageName和ImageDirectory参数的值将被忽略。确保运行此组件的用户对相机存储捕获图像的位置具有读/写访问权限。

默认值：false

Note

当您查看此组件的配方时，UseCamera配置参数不会出现在默认配置中。但是，在部署组件时，可以在配置合并更新中修改此参数的值。

如果设置为 UseCameratrue，则还必须创建符号链接，以使推理组件能够从运行时组件创建的虚拟环境中访问您的摄像头。有关使用带有示例推理组件的摄像头的更多信息，请参阅更新组件配置。

2.0.x

`MLRootPath`

（可选）Linux 核心设备上推理组件读取图像和写入推理结果的文件夹路径。您可以将此值修改为设备上运行此组件的用户具有读/写访问权限的任何位置。

默认值：*/greengrass/v2*/work/variant.DLR/greengrass_ml

默认值：*/greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml

`Accelerator`

请勿修改。目前，加速器唯一支持的值是cpu，因为依赖模型组件中的模型仅针对 CPU 加速器进行编译。

`ImageName`

（可选）推理组件用作预测输入的图像的名称。该组件将在中指定的文件夹中查找图像ImageDirectory。默认位置是*MLRootPath*/images。 AWS IoT Greengrass支持以下图像格式：jpeg、jpgpng、和npy。

默认值：objects.jpg

`InferenceInterval`

（可选）推理代码每次预测之间的时间（以秒为单位）。示例推理代码无限期运行，并在指定的时间间隔内重复其预测。例如，如果您想使用摄像机拍摄的图像进行实时预测，则可以将其更改为较短的间隔。

默认值：3600

`ModelResourceKey`

（可选）在依赖的公共模型组件中使用的模型。仅当使用自定义组件覆盖公共模型组件时，才修改此参数。

默认值：

```
{
```

```
    armv7l: "DLR-yolo3-armv7l-cpu-ObjectDetection",
    x86_64: "DLR-yolo3-x86_64-cpu-ObjectDetection"
}
```

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log -Tail 10
   -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.13	Greengrass nucleus 版本 2.12.0 版本的版本已更新。

版本	更改
2.1.12	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.11	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.10	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.9	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.8	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.7	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.1.5	组件全部发布AWS 区域。
2.1.4	Greengrass nucleus 版本 2.4.0 版本的版本已更新。 此版本在欧洲（伦敦）不可用（`eu-west-2` ）。
2.1.3	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.1.2	错误修复和改进 • 修复了导致样本 DLR 对象检测推理结果中边界框不准确的图像缩放问题。

版本	更改
2.1.1	新功能 • 使用深度学习运行时 v1.6.0。 • 在 Armv8 (aarch64) 平台上添加对样本对象检测的支持。这扩展了对运行 NVIDIA Jetson 的 Greengrass 核心设备（例如 Jetson Nano）的机器学习支持。 • 启用摄像头集成以进行样本推理。使用新的`UseCamera` 配置参数启用示例推理代码，以访问您的 Greengrass 核心设备上的摄像头，并在本地对捕获的图像运行推理。 • 添加对发布推理结果的AWS Cloud支持。使用新的`PublishResultsOnTopic` 配置参数来指定要发布结果的主题。 • 添加新的`ImageDirectory` 配置参数，使您能够为要对其执行推理的图像指定自定义目录。 错误修复和改进 • 将推理结果写入组件日志文件，而不是单独的推理文件。 • 使用AWS IoT Greengrass核心软件日志模块记录组件输出。 • 使用读AWS IoT Device SDK取组件配置并应用配置更改。
2.0.4	初始版本。

DLR 图像分类模型存储

DLR 图像分类模型存储库是一个机器学习模型组件，其中包含预训练的 ResNet -50 个模型作为 Greengrass 工件。此组件中使用的预训练模型从 GluonCV 模型库中获取，并使用 SageMaker Neo Deep Learnin g Runtime 进行编译。

D LR 图像分类推理组件使用此组件作为模型源的依赖项。要使用自定义训练的 DLR 模型，请创建此模型组件的自定义版本，并将您的自定义模型作为组件构件包括在内。您可以使用此组件的配方作为模板来创建自定义模型组件。

Note

DLR 图像分类模型存储组件的名称因版本而异。版本 2.1.x 及更高版本的组件名称为。`variant.DLR.ImageClassification.ModelStore`版本 2.0.x 的组件名称为。`variant.ImageClassification.ModelStore`

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x () `variant.DLR.ImageClassification.ModelStore`
- 2.0.x () `variant.ImageClassification.ModelStore`

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux

- Windows

要求

此组件具有以下要求：

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

       ```
       sudo raspi-config
       ```

    2. 选择接口选项。
    3. 选择旧版相机以启用旧版相机堆栈。
    4. 重启 Raspberry Pi。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.12

下表列出了此组件版本 2.1.12 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性

2.1.11

下表列出了此组件版本 2.1.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性

2.1.10

下表列出了此组件版本 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性

2.1.9

下表列出了此组件版本 2.1.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性

2.1.8

下表列出了此组件版本 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性

2.1.5

下表列出了此组件版本 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性

2.0.x

下表列出了此组件版本 2.0.x 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	~2.0.0	软性

配置

此组件没有任何配置参数。

本地日志文件

此组件不输出日志。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.12	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.11	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.10	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.9	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.8	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.7	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.5	新功能 • 为 Windows 核心设备添加示例图像分类模型。 • Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.1.4	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.1.1	新功能 • 为 Armv8 (aarch64) 平台添加一个 ResNet -50 图像分类模型示例。这扩展了对运行 NVIDIA Jetson 的 Greengrass 核心设备（例如 Jetson Nano）的机器学习支持。
2.0.4	初始版本。

DLR 物体检测模型存储

DLR 对象检测模型存储库是一个机器学习模型组件，其中包含作为 Greengrass 工件的预训练的 YOLOv3 模型。此组件中使用的示例模型从 GluonCV 模型库中获取，并使用 SageMaker Neo Dee p Learnin g Runtime 进行编译。

D LR 对象检测推理组件使用此组件作为模型源的依赖项。要使用自定义训练的 DLR 模型，请创建此模型组件的自定义版本，并将您的自定义模型作为组件构件包括在内。您可以使用此组件的配方作为模板来创建自定义模型组件。

> **Note**
>
> DLR 对象检测模型存储组件的名称因其版本而异。版本 2.1.x 及更高版本的组件名称为。`variant.DLR.ObjectDetection.ModelStore`版本 2.0.x 的组件名称为。`variant.ObjectDetection.ModelStore`

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x
- 2.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上进行升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

       ```
       sudo raspi-config
       ```

    2. 选择接口选项。
    3. 选择旧版相机以启用旧版相机堆栈。
    4. 重启 Raspberry Pi。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.13

下表列出了此组件版本 2.1.13 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性

2.1.12

下表列出了此组件版本 2.1.12 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性

2.1.11

下表列出了此组件版本 2.1.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性

2.1.10

下表列出了此组件版本 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性

2.1.9

下表列出了此组件版本 2.1.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性

2.1.8

下表列出了此组件版本 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性

2.1.5 and 2.1.6

下表列出了此组件版本 2.1.5 和 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性

2.0.x

下表列出了此组件版本 2.0.x 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	~2.0.0	软性

配置

此组件没有任何配置参数。

本地日志文件

此组件不输出日志。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.13	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.12	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.11	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.10	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.9	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.8	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.7	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.6	添加 CPU 型号以修复 Armv8 (aarch64) 设备上的问题。
2.1.5	新功能 • 为 Windows 核心设备添加示例对象检测模型。 错误修复和改进 • Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.1.4	Greengrass nucleus 版本 2.4.0 版本的版本已更新。

版本	更改
2.1.3	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.1.1	新功能 • 为 Armv8 (aarch64) 平台添加一个 YOLOv3 对象检测模型示例。这扩展了对运行 NVIDIA Jetson 的 Greengrass 核心设备（例如 Jetson Nano）的机器学习支持。
2.0.4	初始版本。

DLR 运行时

DLR 运行时组件 (`variant.DLR`) 包含一个脚本，用于在设备上的虚拟环境中安装深度学习运行时 (DLR) 及其依赖关系。DLR 图像分类和DLR 物体检测组件使用此组件作为安装 DLR 的依赖项。组件版本 1.6.x 安装了 DLR v1.6.0，组件版本 1.3.x 安装了 DLR v1.3.0。

要使用不同的运行时，您可以使用此组件的配方作为模板来创建自定义机器学习组件。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 使用量
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 1.6.x
- 1.3.x

类型

此组件是一个通用组件 (aws.greengrass.generic)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

```
sudo raspi-config
```

2. 选择接口选项。
3. 选择旧版相机以启用旧版相机堆栈。
4. 重启 Raspberry Pi。

端点和端口

默认情况下，此组件使用安装程序脚本使用`apt`、`yum``brew`、和`pip`命令安装软件包，具体取决于核心设备使用的平台。此组件必须能够对各种软件包索引和存储库执行出站请求才能运行安装程序脚本。要允许此组件的出站流量通过代理或防火墙，您必须确定软件包索引的端点以及核心设备要连接安装的存储库。

在确定此组件的安装脚本所需的端点时，请考虑以下几点：

- 端点取决于核心设备的平台。例如，运行 Ubuntu 的核心设备使用`apt`而不是`yum`或。`brew`此外，使用相同软件包索引的设备可能具有不同的来源列表，因此它们可能会从不同的存储库中检索软件包。
- 使用相同软件包索引的多台设备之间的端点可能有所不同，因为每台设备都有自己的源列表，用于定义在哪里检索软件包。
- 端点可能会随着时间的推移而发生变化。每个包索引都提供您下载软件包的存储库的 URL，软件包的所有者可以更改软件包索引提供的网址。

有关此组件安装的依赖项以及如何禁用安装程序脚本的更多信息，请参阅UseInstaller配置参数。

有关基本操作所需的终端节点和端口的更多信息，请参阅允许设备流量通过代理或防火墙。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

1.6.11 and 1.6.12

下表列出了此组件版本 1.6.11 和 1.6.12 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <3.0.0	软性

1.6.10

下表列出了此组件版本 1.6.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性

1.6.9

下表列出了此组件版本 1.6.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性

1.6.8

下表列出了此组件版本 1.6.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性

1.6.6 and 1.6.7

下表列出了此组件版本 1.6.6 和 1.6.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性

1.6.4 and 1.6.5

下表列出了此组件版本 1.6.4 和 1.6.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性

1.6.3

下表列出了此组件版本 1.6.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性

1.6.2

下表列出了此组件版本 1.6.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性

1.6.1

下表列出了此组件版本 1.6.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性

1.3.x

下表列出了此组件版本 1.3.x 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	~2.0.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`MLRootPath`

（可选）Linux 核心设备上推理组件读取图像和写入推理结果的文件夹路径。您可以将此值修改为设备上运行此组件的用户具有读/写访问权限的任何位置。

默认值：`/greengrass/v2/work/variant.DLR/greengrass_ml`

`WindowsMLRootPath`

此功能在本组件的 1.6.6 及更高版本中可用。

（可选）Windows 核心设备上推理组件读取图像和写入推理结果的文件夹路径。您可以将此值修改为设备上运行此组件的用户具有读/写访问权限的任何位置。

默认值：`C:\greengrass\v2\\work\\variant.DLR\\greengrass_ml`

`UseInstaller`

（可选）字符串值，用于定义是否使用此组件中的安装程序脚本来安装 DLR 及其依赖项。支持的值为 `true` 和 `false`。

`false`如果要使用自定义脚本安装 DLR，或者要在预构建的 Linux 映像中包含运行时依赖关系，请将此值设置为。要将此组件与AWS提供的 DLR 推理组件一起使用，请安装以下库（包括所有依赖项），并将其提供给运行 ML 组件的系统用户（例如`ggc_user`）。

- Python 3.7 或更高版本，包括pip适用于你的 Python 版本。
- 深度学习运行时 v1. 6.0
- NumPy.
- OpenCV Python。

- AWS IoT Device SDK适用于 Python 的 v2。
- AWS通用运行时 (CRT) Python。
- Picamera（仅适用于 Raspberry Pi 设备）。
- awscam模块（用于AWS DeepLens设备）。
- libGL（适用于 Linux 设备）

默认值：true

使用量

使用UseInstaller配置参数设置为的此组件在您的设备上安装 DLR 及其依赖项。true该组件在您的设备上设置一个虚拟环境，其中包括 OpenCV 和 DLR 所需的 NumPy 库。

> **Note**
>
> 此组件中的安装程序脚本还会安装在设备上配置虚拟环境和使用已安装的机器学习框架所需的额外系统库的最新版本。这可能会升级您设备上的现有系统库。查看下表，了解此组件为每个支持的操作系统安装的库列表。如果要自定义此安装过程，请将UseInstaller配置参数设置为false，然后开发自己的安装程序脚本。

平台	设备系统上安装的库	安装在虚拟环境中的库
Armv7l	build-essential，cmake，ca-certificates，git	setuptools，wheel
Amazon Linux 2	mesa-libGL	无
Ubuntu	wget	无

部署推理组件时，此运行时组件会首先验证您的设备是否已安装 DLR 及其依赖项，如果没有，则会为您安装。

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/variant.DLR.log
```

Windows

```
C:\greengrass\v2\logs\variant.DLR.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/variant.DLR.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\variant.DLR.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
1.6.12	错误修复和改进 • 修复了 Windows 操作系统用户的安装脚本。
1.6.11	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
1.6.10	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
1.6.9	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
1.6.8	Greengrass nucleus 版本 2.6.0 版本的版本已更新。

版本	更改
1.6.7	错误修复和改进 • 更新`UseInstaller` 安装脚本以安装 libGL，该脚本在某些 Linux 平台上默认不可用。 • 更新`UseInstaller` 安装脚本，使其始终在此组件的虚拟环境中使用 Python 3.9。此更改有助于确保与其他库的兼容性。
1.6.6	新功能 • 增加了对运行 Windows 的核心设备的支持。 • 添加新的`WindowsMLRootPath` 配置参数，您可以使用该参数在 Windows 核心设备上配置推理结果文件夹。
1.6.5	新功能 • 添加新的`UseInstaller` 配置参数，您可以使用该参数禁用此组件中的安装脚本。
1.6.4	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
1.6.3	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
1.6.2	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
1.6.1	新功能 • 安装深度学习运行时 v1.6.0 及其依赖项。 • 添加对在 Armv8 (aarch64) 平台上安装 DLR 的支持。这扩展了对运行 NVIDIA Jetson 的 Greengrass 核心设备（例如 Jetson Nano）的机器学习支持。 错误修复和改进 • 在虚拟环境AWS IoT Device SDK中安装以读取组件配置并应用配置更改。 • 其他小错误修复和改进。
1.3.2	初始版本。安装 DLR v1.3.0。

TensorFlow 精简版图像分类

TensorFlow Lite 图像分类组件 (`aws.greengrass.TensorFlowLiteImageClassification`) 包含示例推理代码，用于使用 TensorFlow Lite 运行时和样本预训练的 MobileNet 1.0 量化模型执行图像分类推理。此组件使用变体TensorFlow 精简版图像分类模型存储和TensorFlow 精简版运行时组件作为依赖项来下载 TensorFlow Lite 运行时和示例模型。

要将此推理组件与自定义训练的 TensorFlow Lite 模型一起使用，请创建依赖模型存储组件的自定义版本。要使用自己的自定义推理代码，您可以使用此组件的配方作为模板来创建自定义推理组件。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux

- Windows

要求

此组件具有以下要求：

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上进行升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

       ```
       sudo raspi-config
       ```

    2. 选择接口选项。
    3. 选择旧版相机以启用旧版相机堆栈。
    4. 重启 Raspberry Pi。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.11

下表列出了此组件版本 2.1.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.10

下表列出了此组件版本 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.9

下表列出了此组件版本 2.1.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.8

下表列出了此组件版本 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.5

下表列出了此组件版本 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`accessControl`

（可选）包含授权策略的对象，该策略允许组件向默认通知主题发布消息。

默认值：

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/tflite/image-classification"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

（可选）您要发布推理结果的主题。如果您修改此值，则还必须修改`accessControl`参数`resources`中的值以匹配您的自定义主题名称。

默认值：`ml/tflite/image-classification`

`Accelerator`

你要使用的加速器。支持的值为 cpu 和 gpu。

依赖模型组件中的示例模型仅支持 CPU 加速。要在不同的自定义模型中使用 GPU 加速，请创建一个自定义模型组件来覆盖公共模型组件。

默认值：cpu

`ImageDirectory`

（可选）设备上推理组件读取图像的文件夹路径。您可以将此值修改为设备上任何您拥有读/写访问权限的位置。

默认值：*/greengrass/v2*/packages/artifacts-unarchived/*component-name*/image_classification/sample_images/

> **Note**
> 如果将的值设置UseCamera为true，则忽略此配置参数。

ImageName

（可选）推理组件用作预测输入的图像的名称。该组件将在中指定的文件夹中查找图像ImageDirectory。默认情况下，该组件使用默认图像目录中的示例图像。 AWS IoT Greengrass支持以下图像格式：jpeg、jpgpng、和npy。

默认值：cat.jpeg

> **Note**
> 如果将的值设置UseCamera为true，则忽略此配置参数。

InferenceInterval

（可选）推理代码每次预测之间的时间（以秒为单位）。示例推理代码无限期运行，并在指定的时间间隔内重复其预测。例如，如果您想使用摄像机拍摄的图像进行实时预测，则可以将其更改为较短的间隔。

默认值：3600

ModelResourceKey

（可选）在依赖的公共模型组件中使用的模型。仅当使用自定义组件覆盖公共模型组件时，才修改此参数。

默认值：

```
{
    "model": "TensorFlowLite-Mobilenet"
}
```

`UseCamera`

（可选）字符串值，用于定义是否使用连接到 Greengrass 核心设备的摄像机的图像。支持的值为 `true` 和 `false`。

当您将此值设置为时`true`，示例推理代码将访问设备上的摄像头，并在本地对捕获的图像运行推理。`ImageName`和`ImageDirectory`参数的值将被忽略。确保运行此组件的用户对相机存储捕获图像的位置具有读/写访问权限。

默认值：`false`

当您查看此组件的配方时，`UseCamera`配置参数不会出现在默认配置中。但是，在部署组件时，可以在配置合并更新中修改此参数的值。
如果设置为 `UseCameratrue`，则还必须创建符号链接，以使推理组件能够从运行时组件创建的虚拟环境中访问您的摄像头。有关使用带有示例推理组件的摄像头的更多信息，请参阅更新组件配置。

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteImageClassification.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

Linux

```
sudo tail -f /greengrass/v2/logs/
aws.greengrass.TensorFlowLiteImageClassification.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs
\aws.greengrass.TensorFlowLiteImageClassification.log -Tail 10 -Wait
```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.11	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.10	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.9	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.8	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.7	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.5	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.4	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.1.0	初始版本。

TensorFlow 精简版物体检测

TensorFlow Lite 对象检测组件 (`aws.greengrass.TensorFlowLiteObjectDetection`) 包含使用 L TensorFlow it e 执行对象检测推理的示例推理代码和预训练的单枪检测 (SSD) MobileNet 1.0 模型示例。此组件使用变体TensorFlow 精简版物体检测模型存储和TensorFlow 精简版运行时组件作为依赖项来下载 TensorFlow Lite 和示例模型。

要将此推理组件与自定义训练的 TensorFlow Lite 模型一起使用，您可以创建依赖模型存储组件的自定义版本。要使用您自己的自定义推理代码，请使用此组件的配方作为模板来创建自定义推理组件。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux

- Windows

要求

此组件具有以下要求：

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上进行升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

       ```
       sudo raspi-config
       ```

    2. 选择接口选项。
    3. 选择旧版相机以启用旧版相机堆栈。
    4. 重启 Raspberry Pi。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.11

下表列出了此组件版本 2.1.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.10

下表列出了此组件版本 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.9

下表列出了此组件版本 2.1.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.8

下表列出了此组件版本 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.5

下表列出了此组件版本 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性
TensorFlow 精简版图像分类模型存储	>=2.1.0 <2.2.0	硬性
TensorFlow 精简版	>=2.5.0 <2.6.0	硬性

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`accessControl`

（可选）包含授权策略的对象，该策略允许组件向默认通知主题发布消息。

默认值：

```
{
    "aws.greengrass.ipc.mqttproxy": {
        "aws.greengrass.TensorFlowLiteObjectDetection:mqttproxy:1": {
            "policyDescription": "Allows access to publish via topic ml/tflite/object-
detection.",
            "operations": [
                "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
                "ml/tflite/object-detection"
            ]
        }
    }
}
```

`PublishResultsOnTopic`

（可选）您要发布推理结果的主题。如果您修改此值，则还必须修改`accessControl`参数`resources`中的值以匹配您的自定义主题名称。

默认值：`ml/tflite/object-detection`

`Accelerator`

你要使用的加速器。支持的值为 cpu 和 gpu。

依赖模型组件中的示例模型仅支持 CPU 加速。要在不同的自定义模型中使用 GPU 加速，请创建一个自定义模型组件来覆盖公共模型组件。

默认值：cpu

`ImageDirectory`

（可选）设备上推理组件读取图像的文件夹路径。您可以将此值修改为设备上任何您拥有读/写访问权限的位置。

默认值：*/greengrass/v2*/packages/artifacts-unarchived/*component-name*/object_detection/sample_images/

> **Note**
> 如果将的值设置UseCamera为true，则忽略此配置参数。

ImageName

（可选）推理组件用作预测输入的图像的名称。该组件将在中指定的文件夹中查找图像ImageDirectory。默认情况下，该组件使用默认图像目录中的示例图像。 AWS IoT Greengrass支持以下图像格式：jpeg、jpgpng、和npy。

默认值：objects.jpg

> **Note**
> 如果将的值设置UseCamera为true，则忽略此配置参数。

InferenceInterval

（可选）推理代码每次预测之间的时间（以秒为单位）。示例推理代码无限期运行，并在指定的时间间隔内重复其预测。例如，如果您想使用摄像机拍摄的图像进行实时预测，则可以将其更改为较短的间隔。

默认值：3600

ModelResourceKey

（可选）在依赖的公共模型组件中使用的模型。仅当使用自定义组件覆盖公共模型组件时，才修改此参数。

默认值：

```
{
    "model": "TensorFlowLite-SSD"
}
```

`UseCamera`

（可选）字符串值，用于定义是否使用连接到 Greengrass 核心设备的摄像机的图像。支持的值为 `true` 和 `false`。

将此值设置为时`true`，示例推理代码将访问设备上的摄像头，并在本地对捕获的图像运行推理。`ImageName`和`ImageDirectory`参数的值将被忽略。确保运行此组件的用户对相机存储捕获图像的位置具有读/写访问权限。

默认值：`false`

> **Note**
>
> 当您查看此组件的配方时，`UseCamera`配置参数不会出现在默认配置中。但是，在部署组件时，可以在配置合并更新中修改此参数的值。
> 如果设置为 `UseCamera` `true`，则还必须创建符号链接，以使推理组件能够从运行时组件创建的虚拟环境中访问您的摄像头。有关使用带有示例推理组件的摄像头的更多信息，请参阅更新组件配置。

> **Note**
>
> 当您查看此组件的配方时，`UseCamera`配置参数不会出现在默认配置中。但是，在部署组件时，可以在配置合并更新中修改此参数的值。
> 如果设置为 `UseCamera` `true`，则还必须创建符号链接，以使推理组件能够从运行时组件创建的虚拟环境中访问您的摄像头。有关使用带有示例推理组件的摄像头的更多信息，请参阅更新组件配置。

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteObjectDetection.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.greengrass.TensorFlowLiteObjectDetection.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.greengrass.TensorFlowLiteObjectDetection.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.11	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.10	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.9	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.8	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.7	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.5	Greengrass nucleus 版本 2.6.0 版本的版本已更新。

版本	更改
2.1.4	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.1.1	错误修复和改进 • 修复了导致样本 TensorFlow Lite 对象检测推理结果中边界框不准确的图像缩放问题。
2.1.0	初始版本。

TensorFlow 精简版图像分类模型存储

TensorFlow 精简版图像分类模型存储 (`variant.TensorFlowLite.ImageClassification.ModelStore`) 是一个机器学习模型组件，其中包含一个作为 Greengrass 工件的预训练的 MobileNet v1 模型。此组件中使用的示例模型是从 TensorFlowHub 获取的，并使用 TensorFlow Lite 实现。

TensorFlow 精简版图像分类推理组件使用此组件作为模型源的依赖项。要使用自定义训练的 TensorFlow Lite 模型，请创建此模型组件的自定义版本，并将您的自定义模型作为组件构件包括在内。您可以使用此组件的配方作为模板来创建自定义模型组件。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令来安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上进行升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

启用旧版相机堆栈

1. 运行以下命令打开 Raspberry Pi 配置工具。

```
sudo raspi-config
```

2. 选择 “接口选项”。
3. 选择旧版相机以启用旧版相机堆栈。
4. 重启 Raspberry Pi。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.11

下表列出了此组件版本 2.1.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性

2.1.10

下表列出了此组件版本 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性

2.1.9

下表列出了此组件版本 2.1.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性

2.1.8

下表列出了此组件版本 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性

2.1.5

下表列出了此组件版本 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性

配置

此组件没有任何配置参数。

本地日志文件

此组件不输出日志。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.11	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.10	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.9	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.8	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.7	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.5	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.4	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.1.0	初始版本。

TensorFlow 精简版物体检测模型存储

TensorFlow Lite 对象检测模型存储 (variant.TensorFlowLite.ObjectDetection.ModelStore) 是一个机器学习模型组件，其中包含作为 Greengrass 工件的预训练单枪检测 (SSD) MobileNet 模型。此组件中使用的示例模型是从 TensorFlow Hub 获取的，并使用 TensorFlow Lite 实现。

L TensorFlow ite 对象检测推理组件使用此组件作为模型源的依赖项。要使用自定义训练的 TensorFlow Lite 模型，请创建此模型组件的自定义版本，并将您的自定义模型作为组件构件包括在内。您可以使用此组件的配方作为模板来创建自定义模型组件。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x

类型

此组件是一个通用组件 (aws.greengrass.generic)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux

- Windows

要求

此组件具有以下要求：

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上进行升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

       ```
       sudo raspi-config
       ```

    2. 选择接口选项。
    3. 选择旧版相机以启用旧版相机堆栈。
    4. 重启 Raspberry Pi。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.11

下表列出了此组件版本 2.1.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性

2.1.10

下表列出了此组件版本 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性

2.1.9

下表列出了此组件版本 2.1.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性

2.1.8

下表列出了此组件版本 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性

2.1.5

下表列出了此组件版本 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性

配置

此组件没有任何配置参数。

本地日志文件

此组件不输出日志。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.11	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.10	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.9	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.8	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.7	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.5	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.4	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.1.0	初始版本。

TensorFlow 精简版运行时

TensorFlow Lite 运行时组件 (`variant.TensorFlowLite`) 包含一个脚本，用于在设备上的虚拟环境中安装 TensorFlow Lite 版本 2.5.0 及其依赖项。TensorFlow Lite 图像分类和 TensorFlow Lite 对象检测组件使用此运行时组件作为安装 TensorFlow Lite 的依赖项。

> **Note**
>
> TensorFlow 精简版运行时组件 v2.5.6 及更高版本会重新安装 TensorFlow Lite 运行时及其依赖项的现有安装。重新安装有助于确保核心设备运行兼容版本的 TensorFlow Lite 及其依赖项。

要使用不同的运行时，您可以使用此组件的配方作为模板来创建自定义机器学习组件。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 使用量
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.5.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。

- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上进行升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

       ```
       sudo raspi-config
       ```

    2. 选择接口选项。
    3. 选择旧版相机以启用旧版相机堆栈。
    4. 重启 Raspberry Pi。

端点和端口

默认情况下，此组件使用安装程序脚本使用`apt`、`yumbrew`、和`pip`命令安装软件包，具体取决于核心设备使用的平台。此组件必须能够对各种软件包索引和存储库执行出站请求才能运行安装程序脚本。要允许此组件的出站流量通过代理或防火墙，您必须确定软件包索引的端点以及核心设备要连接安装的存储库。

在确定此组件的安装脚本所需的端点时，请考虑以下几点：

- 端点取决于核心设备的平台。例如，运行 Ubuntu 的核心设备使用`apt`而不是`yum`或。`brew`此外，使用相同软件包索引的设备可能具有不同的来源列表，因此它们可能会从不同的存储库中检索软件包。
- 使用相同软件包索引的多台设备之间的端点可能有所不同，因为每台设备都有自己的源列表，用于定义在哪里检索软件包。

- 端点可能会随着时间的推移而发生变化。每个包索引都提供您下载软件包的存储库的 URL，软件包的所有者可以更改软件包索引提供的网址。

有关此组件安装的依赖项以及如何禁用安装程序脚本的更多信息，请参阅UseInstaller配置参数。

有关基本操作所需的终端节点和端口的更多信息，请参阅允许设备流量通过代理或防火墙。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.5.14

下表列出了此组件版本 2.5.14 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性

2.5.13

下表列出了此组件版本 2.5.13 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性

2.5.12

下表列出了此组件版本 2.5.12 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性

2.5.11

下表列出了此组件版本 2.5.11 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性

2.5.10

下表列出了此组件版本 2.5.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性

2.5.9

下表列出了此组件版本 2.5.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性

2.5.8

下表列出了此组件版本 2.5.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性

2.5.5 - 2.5.7

下表列出了此组件版本 2.5.5 到 2.5.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性

2.5.3 and 2.5.4

下表列出了此组件版本 2.5.3 和 2.5.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性

2.5.2

下表列出了此组件版本 2.5.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性

2.5.1

下表列出了此组件版本 2.5.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性

2.5.0

下表列出了此组件版本 2.5.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

MLRootPath

（可选）Linux 核心设备上推理组件读取图像和写入推理结果的文件夹路径。您可以将此值修改为设备上运行此组件的用户具有读/写访问权限的任何位置。

默认值：*/greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml

WindowsMLRootPath

此功能在本组件的 1.6.6 及更高版本中可用。

（可选）Windows 核心设备上推理组件读取图像和写入推理结果的文件夹路径。您可以将此值修改为设备上运行此组件的用户具有读/写访问权限的任何位置。

默认值：*C:\greengrass\v2*\\work\\variant.DLR\\greengrass_ml

UseInstaller

（可选）字符串值，用于定义是否使用此组件中的安装程序脚本来安装 TensorFlow Lite 及其依赖项。支持的值为 true 和 false。

false如果要使用自定义脚本进行 TensorFlow 精简版安装，或者想要在预构建的 Linux 映像中包含运行时依赖关系，请将此值设置为。要将此组件与AWS提供的 TensorFlow Lite 推理组件一起使用，请安装以下库（包括所有依赖项），并将其提供给运行 ML 组件的系统用户（例如ggc_user）。

- Python 3.8 或更高版本，包括pip适用于你的 Python 版本
- TensorFlow 精简版 v2. 5.0
- NumPy
- OpenCV-Pyt
- AWS IoT Device SDK适用于 Python 的 v2
- AWS公共运行时 (CRT) Python
- Picamera（适用于 Raspberry Pi 设备）
- awscam模块（用于AWS DeepLens设备）

- libGL（适用于 Linux 设备）

  默认值：`true`

使用量

使用此组件，将`UseInstaller`配置参数设置`true`为，在您的设备上安装 TensorFlow Lite 及其依赖项。该组件在您的设备上设置了一个虚拟环境，其中包括精简版所需的 TensorFlow OpenCV和 NumPy 库。

> **Note**
>
> 此组件中的安装程序脚本还会安装在设备上配置虚拟环境和使用已安装的机器学习框架所需的额外系统库的最新版本。这可能会升级您设备上的现有系统库。查看下表，了解此组件为每个支持的操作系统安装的库列表。如果要自定义此安装过程，请将`UseInstaller`配置参数设置为`false`，然后开发自己的安装程序脚本。

平台	设备系统上安装的库	安装在虚拟环境中的库
Armv7l	`build-essential`，`cmake`，`ca-certificates`，`git`	`setuptools`，`wheel`
Amazon Linux 2	`mesa-libGL`	无
Ubuntu	`wget`	无

部署推理组件时，此运行时组件会首先验证您的设备是否已安装 TensorFlow Lite 及其依赖项。如果不是，则运行时组件会为您安装它们。

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/variant.TensorFlowLite.log
```

Windows

```
C:\greengrass\v2\logs\variant.TensorFlowLite.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/variant.TensorFlowLite.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\variant.TensorFlowLite.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.5.14	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.5.13	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.5.12	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.5.11	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.5.10	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.5.9	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.5.8	Greengrass nucleus 版本 2.6.0 版本的版本已更新。

版本	更改
2.5.7	错误修复和改进 • 更新`UseInstaller` 安装脚本以安装 libGL，该脚本在某些 Linux 平台上默认不可用。 • 更新`UseInstaller` 安装脚本，使其始终在此组件的虚拟环境中使用 Python 3.9。此更改有助于确保与其他库的兼容性。
2.5.6	错误修复和改进 • 更新此组件以安装 TensorFlow Lite 2.5.0 (`tflite-runtime-2.5.0.post1`) 的最新补丁，因此您可以在 Python 3.9 中使用此组件。如果此组件无法安装该修补程序，则`tflite-runtime-2.5.0` 改为安装该组件。 • 更新此组件以重新安装现有的 TensorFlow Lite 及其依赖项。此更改有助于确保核心设备运行兼容版本的 TensorFlow Lite 及其依赖项。
2.5.5	新功能 • 增加了对运行 Windows 的核心设备的支持。 • 添加新的`WindowsMLRootPath` 配置参数，您可以使用该参数在 Windows 核心设备上配置推理结果文件夹。
2.5.4	新功能 • 添加新的`UseInstaller` 配置参数，允许您禁用此组件中的安装脚本。
2.5.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.5.2	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.5.1	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.5.0	初始版本。

modbus-RTU 协议适配器

Modbus-RTU 协议适配器组件 (`aws.greengrass.Modbus`) 轮询来自本地 Modbus RTU 设备的信息。

要使用此组件从本地 Modbus RTU 设备请求信息，请向该组件订阅的主题发布一条消息。在消息中，指定要发送到设备的 Modbus RTU 请求。然后，该组件发布一个包含 Modbus RTU 请求结果的响应。

> **Note**
>
> 该组件提供的功能与 V1 中的 Modbus RTU 协议适配器 AWS IoT Greengrass 连接器类似。有关更多信息，请参阅《AWS IoT Greengrass V1 开发人员指南》中的 Modbus RTU 协议适配器连接器。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 输入数据
- 输出数据
- Modbus RTU 请求和响应
- 本地日志文件
- 许可证
- 更改日志

版本

此组件有以下版本：

- 2.1.x
- 2.0.x

类型

此组件是一个 Lambda 组件 () `aws.greengrass.lambda`。Greengrass 核心使用 Lambda 启动器组件运行此组件的 Lambda 函数。

有关更多信息，请参阅组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

- 您的核心设备必须满足运行 Lambda 函数的要求。如果您希望核心设备运行容器化的 Lambda 函数，则该设备必须满足要求。有关更多信息，请参阅Lambda 函数要求。
- Python 版本 3.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- AWS IoT Greengrass 核心设备和 Modbus 设备之间的物理连接。核心设备必须通过串行端口（例如 USB 端口）与 Modbus RTU 网络进行物理连接。
- 要接收此组件的输出数据，在部署此组件时，必须合并旧版订阅路由器组件 (`aws.greengrass.LegacySubscriptionRouter`) 的以下配置更新。此配置指定此组件发布响应的主题。

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-modbus": {
        "id": "aws-greengrass-modbus",
        "source": "component:aws.greengrass.Modbus",
        "subject": "modbus/adapter/response",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-modbus": {
        "id": "aws-greengrass-modbus",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
  modbus:version",
        "subject": "modbus/adapter/response",
  ```

```
        "target": "cloud"
      }
    }
  }
```

- 将##替换为您 AWS 区域 使用的区域。
- 将##替换为该组件运行的 Lambda 函数的版本。要查找 Lambda 函数版本，必须查看要部署的此组件版本的配方。在AWS IoT Greengrass 控制台中打开此组件的详细信息页面，查找 Lambda 函数键值对。此键值对包含 Lambda 函数的名称和版本。

> **Important**
>
> 每次部署此组件时，都必须更新旧版订阅路由器上的 Lambda 函数版本。这样可以确保您为部署的组件版本使用正确的 Lambda 函数版本。

有关更多信息，请参阅创建部署。

- 支持 Modbus-RTU 协议适配器在 VPC 中运行。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.8

下表列出了此组件版本 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.4 and 2.1.5

下表列出了此组件版本 2.1.4 和 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性

依赖关系	兼容版本	依赖关系类型
代币兑换服务	^2.0.0	硬性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.8 and 2.1.0

下表列出了此组件版本 2.0.8 和 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.7

下表列出了此组件版本 2.0.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.6

下表列出了此组件版本 2.0.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.5

下表列出了此组件版本 2.0.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.4

下表列出了此组件版本 2.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性

依赖关系	兼容版本	依赖关系类型
代币兑换服务	^2.0.0	硬性

2.0.3

下表列出了此组件版本 2.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	硬性
Lambda 启动器	>=1.0.0	硬性
Lambda 运行时	>=1.0.0	软性
代币兑换服务	>=1.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

Note

此组件的默认配置包括 Lambda 函数参数。我们建议您仅编辑以下参数，以便在您的设备上配置此组件。

v2.1.x

`lambdaParams`

包含此组件的 Lambda 函数参数的对象。该对象包含以下信息：

`EnvironmentVariables`

一个包含 Lambda 函数参数的对象。该对象包含以下信息：

`ModbusLocalPort`

核心设备上物理 Modbus 串行端口的绝对路径，例如`/dev/ttyS2`。

要在容器中运行此组件，必须将此路径定义为该组件可以访问的系统设备（中`containerParams.devices`）。默认情况下，此组件在容器中运行。

> **Note**
>
> 此组件必须具有对设备的读/写访问权限。

`ModbusBaudRate`

（可选）一个字符串值，它指定与本地 Modbus TCP 设备进行串行通信的波特率。

默认：`9600`

`ModbusByteSize`

（可选）一个字符串值，用于指定与本地 Modbus TCP 设备进行串行通信时字节的大小。选择5、67、或8位。

默认：`8`

`ModbusParity`

（可选）奇偶校验模式，用于验证与本地 Modbus TCP 设备的串行通信中的数据完整性。

- E— 通过均匀的奇偶校验来验证数据的完整性。
- O— 使用奇数奇偶校验来验证数据的完整性。
- N— 不要验证数据的完整性。

默认：`N`

`ModbusStopBits`

（可选）一个字符串值，它指定在与本地 Modbus TCP 设备进行串行通信时表示字节结束的位数。

默认：`1`

`containerMode`

（可选）此组件的容器化模式。从以下选项中进行选择：

- `GreengrassContainer`— 该组件在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

  如果指定此选项，则必须指定系统设备（在`containerParams.devices`）中以允许容器访问 Modbus 设备。
- `NoContainer`— 该组件不在隔离的运行时环境中运行。

默认：`GreengrassContainer`

`containerParams`

（可选）包含此组件容器参数的对象。如果您为指定`GreengrassContainer`，则该组件将使用这些参数`containerMode`。

该对象包含以下信息：

`memorySize`

（可选）要分配给组件的内存量（以千字节为单位）。

默认为 512 MB (525,312 KB)。

`devices`

（可选）一个对象，它指定组件可以在容器中访问的系统设备。

> **Important**
> 要在容器中运行此组件，必须指定在`ModbusLocalPort`环境变量中配置的系统设备。

该对象包含以下信息：

`0`— 这是字符串形式的数组索引。

包含以下信息的对象：

`path`

核心设备上系统设备的路径。该值必须与您为配置的值相同`ModbusLocalPort`。

`permission`

（可选）从容器访问系统设备的权限。此值必须为`rw`，表示该组件具有对系统设备的读/写访问权限。

默认：`rw`

`addGroupOwner`

（可选）是否将运行该组件的系统组添加为系统设备的所有者。

默认：`true`

`pubsubTopics`

（可选）一个对象，其中包含组件订阅以接收消息的主题。您可以指定每个主题以及该组件是订阅来自的 MQTT 主题 AWS IoT Core 还是本地发布/订阅主题。

该对象包含以下信息：

`0`— 这是字符串形式的数组索引。

包含以下信息的对象：

`type`

（可选）此组件用于订阅消息的发布/订阅消息的类型。从以下选项中进行选择：

- PUB_SUB – 订阅本地发布/订阅消息。如果选择此选项，则主题不能包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅本地消息。
- `IOT_CORE`— 订阅 AWS IoT Core MQTT 消息。如果选择此选项，则主题可以包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

默认：PUB_SUB

`topic`

（可选）组件订阅以接收消息的主题。如果您`IotCore`为指定`type`，则可以在本主题中使用 MQTT 通配符（`+`和`#`）。

Example 示例：配置合并更新（容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
```

```
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example 示例：配置合并更新（无容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

v2.0.x

`lambdaParams`

包含此组件的 Lambda 函数参数的对象。该对象包含以下信息：

`EnvironmentVariables`

一个包含 Lambda 函数参数的对象。该对象包含以下信息：

`ModbusLocalPort`

核心设备上物理 Modbus 串行端口的绝对路径，例如`/dev/ttyS2`。

要在容器中运行此组件，必须将此路径定义为该组件可以访问的系统设备（中`containerParams.devices`）。默认情况下，此组件在容器中运行。

> **Note**
>
> 此组件必须具有对设备的读/写访问权限。

containerMode

（可选）此组件的容器化模式。从以下选项中进行选择：

- GreengrassContainer— 该组件在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

  如果指定此选项，则必须指定系统设备（在containerParams.devices）中以允许容器访问 Modbus 设备。
- NoContainer— 该组件不在隔离的运行时环境中运行。

默认：GreengrassContainer

containerParams

（可选）包含此组件容器参数的对象。如果您为指定GreengrassContainer，则该组件将使用这些参数containerMode。

该对象包含以下信息：

memorySize

（可选）要分配给组件的内存量（以千字节为单位）。

默认为 512 MB (525,312 KB)。

devices

（可选）一个对象，它指定组件可以在容器中访问的系统设备。

Important

要在容器中运行此组件，必须指定在ModbusLocalPort环境变量中配置的系统设备。

该对象包含以下信息：

0— 这是字符串形式的数组索引。

包含以下信息的对象：

path

核心设备上系统设备的路径。该值必须与您为配置的值相同ModbusLocalPort。

`permission`

（可选）从容器访问系统设备的权限。此值必须为`rw`，表示该组件具有对系统设备的读/写访问权限。

默认：`rw`

`addGroupOwner`

（可选）是否将运行该组件的系统组添加为系统设备的所有者。

默认：`true`

`pubsubTopics`

（可选）一个对象，其中包含组件订阅以接收消息的主题。您可以指定每个主题以及该组件是订阅来自的 MQTT 主题 AWS IoT Core 还是本地发布/订阅主题。

该对象包含以下信息：

`0`— 这是字符串形式的数组索引。

包含以下信息的对象：

`type`

（可选）此组件用于订阅消息的发布/订阅消息的类型。从以下选项中进行选择：

- PUB_SUB – 订阅本地发布/订阅消息。如果选择此选项，则主题不能包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅本地消息。
- `IOT_CORE`— 订阅 AWS IoT Core MQTT 消息。如果选择此选项，则主题可以包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

默认：PUB_SUB

`topic`

（可选）组件订阅以接收消息的主题。如果您`IotCore`为指定`type`，则可以在本主题中使用 MQTT 通配符（+和#）。

Example 示例：配置合并更新（容器模式）

```
{
```

```
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example 示例：配置合并更新（无容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

输入数据

该组件接受以下主题上的 Modbus RTU 请求参数，并向设备发送 Modbus RTU 请求。默认情况下，此组件订阅本地发布/订阅消息。有关如何从您的自定义组件向该组件发布消息的更多信息，请参阅发布/订阅本地消息。

默认主题（本地发布/订阅）：`modbus/adapter/request`

该消息接受以下属性。输入消息必须采用 JSON 格式。

`request`

要发送的 Modbus RTU 请求的参数。

请求消息的形状取决于它所代表的 Modbus RTU 请求的类型。所有请求都需要以下属性。

类型：其中`object`包含以下信息：

`operation`

要运行的操作的名称。例如，指定在 `ReadCoilsRequest` Modbus RTU 设备上读取线圈。有关支持的操作的更多信息，请参阅Modbus RTU 请求和响应。

类型：`string`

`device`

请求的目标设备。

此值必须是介于`0`和之间的整数`247`。

类型：`integer`

要包含在请求中的其他参数取决于操作。此组件处理循环冗余校验 (CRC)，以验证您的数据请求。

> **Note**
>
> 如果您请求包含`address`属性，则必须将其值指定为整数。例如，`"address": 1`。

`id`

请求的任意 ID。使用此属性将输入请求映射到输出响应。指定此属性时，组件会将响应对象中的`id`属性设置为该值。

类型：`string`

Example 示例输入：读取线圈请求

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "MyRequest"
```

```
}
```

输出数据

默认情况下，此组件将响应作为以下 MQTT 主题的输出数据发布。您必须在传统订阅路由器组件的配置subject中将此主题指定为。有关如何在自定义组件中订阅有关此主题的消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

默认主题 (AWS IoT Core MQTT)：modbus/adapter/response

响应消息的形状取决于请求操作和响应状态。有关示例，请参阅示例请求和响应。

每个响应均包括以下属性：

`response`

来自 Modbus RTU 设备的响应。

类型：其中object包含以下信息：

`status`

请求的状态。状态可以是以下值之一：

- Success— 请求有效，组件将请求发送到 Modbus RTU 网络，Modbus RTU 网络返回了响应。
- Exception— 请求有效，组件将请求发送到 Modbus RTU 网络，Modbus RTU 网络返回异常。有关更多信息，请参阅响应状态：异常。
- No Response— 请求无效，组件在将请求发送到 Modbus RTU 网络之前捕获了错误。有关更多信息，请参阅响应状态：无响应。

`operation`

组件请求的操作。

`device`

组件发送请求的设备。

`payload`

来自 Modbus RTU 设备的响应。如果status是No Response，则此对象仅包含带有错误描述的error属性（例如[Input/Output] No Response received from the remote unit）。

`id`

请求的 ID，您可以使用它来识别哪个响应与哪个请求相对应。

> **Note**
> 写入操作的响应只不过是对请求的回显。尽管写入响应不包含有意义的信息，但最好检查响应的状态以查看请求是成功还是失败。

Example 示例输出：成功

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "MyRequest"
}
```

Example 示例输出：失败

```
{
  "response" : {
    "status" : "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id" : "MyRequest"
}
```

有关更多示例，请参阅示例请求和响应。

Modbus RTU 请求和响应

此连接器接受 Modbus RTU 请求参数作为输入数据，并发布响应作为输出数据。

支持以下常见操作。

请求中的操作名称	响应中的函数代码
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16
MaskWriteRegisterRequest	22
ReadWriteMultipleRegistersRequest	23

示例请求和响应

以下是受支持操作的示例请求和响应。

读取线圈

请求示例：

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
```

```
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

响应示例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

读取离散输入

请求示例：

```
{
  "request": {
    "operation": "ReadDiscreteInputsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

响应示例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadDiscreteInputsRequest",
```

```
    "payload": {
      "function_code": 2,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

读取持仓寄存器

请求示例：

```
{
  "request": {
    "operation": "ReadHoldingRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

响应示例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadHoldingRegistersRequest",
    "payload": {
      "function_code": 3,
      "registers": [20,30]
    }
  },
  "id" : "TestRequest"
}
```

读取输入寄存器

请求示例：

```
{
  "request": {
```

```
    "operation": "ReadInputRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

写单线圈

请求示例：

```
{
  "request": {
    "operation": "WriteSingleCoilRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

响应示例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteSingleCoilRequest",
    "payload": {
      "function_code": 5,
      "address": 1,
      "value": true
    }
  },
  "id" : "TestRequest"
}
```

写入单个寄存器

请求示例：

```
{
```

```
  "request": {
    "operation": "WriteSingleRegisterRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

写入多个线圈

请求示例：

```
{
  "request": {
    "operation": "WriteMultipleCoilsRequest",
    "device": 1,
    "address": 1,
    "values": [1,0,0,1]
  },
  "id": "TestRequest"
}
```

响应示例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleCoilsRequest",
    "payload": {
      "function_code": 15,
      "address": 1,
      "count": 4
    }
  },
  "id" : "TestRequest"
}
```

写入多个寄存器

请求示例：

```
{
  "request": {
    "operation": "WriteMultipleRegistersRequest",
    "device": 1,
    "address": 1,
    "values": [20,30,10]
  },
  "id": "TestRequest"
}
```

响应示例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
      "address": 1,
      "count": 3
    }
  },
  "id" : "TestRequest"
}
```

掩码写入寄存器

请求示例：

```
{
  "request": {
    "operation": "MaskWriteRegisterRequest",
    "device": 1,
    "address": 1,
    "and_mask": 175,
    "or_mask": 1
  },
  "id": "TestRequest"
}
```

响应示例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "MaskWriteRegisterRequest",
    "payload": {
      "function_code": 22,
      "and_mask": 0,
      "or_mask": 8
    }
  },
  "id" : "TestRequest"
}
```

读写多个寄存器

请求示例：

```
{
  "request": {
    "operation": "ReadWriteMultipleRegistersRequest",
    "device": 1,
    "read_address": 1,
    "read_count": 2,
    "write_address": 3,
    "write_registers": [20,30,40]
  },
  "id": "TestRequest"
}
```

响应示例：

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadWriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
      "registers": [10,20,10,20]
    }
  },
```

```
  "id" : "TestRequest"
}
```

> **Note**
>
> 响应包括组件读取的寄存器。

响应状态：异常

当请求格式有效但请求未成功完成时，可能会发生异常。在此情况下，响应将包含以下信息：

- `status` 设置为 `Exception`。
- `function_code` 等于请求的函数代码 + 128。
- `exception_code` 包含异常代码。有关更多信息，请参阅 Modbus 异常代码。

示例：

```
{
  "response": {
    "status": "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id": "TestRequest"
}
```

响应状态：无响应

该连接器对 Modbus 请求执行验证检查。例如，它会检查无效格式和缺失字段。如果验证失败，则连接器不会发送请求。而是会返回一个包含以下信息的响应：

- `status` 设置为 `No Response`。
- `error` 包含错误原因。

- `error_message` 包含错误消息。

示例：

```
{
  "response": {
    "status": "fail",
    "error_message": "Invalid address field. Expected <type 'int'>, got <type 'str'>",
    "error": "No Response",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "error": "Invalid address field. Expected Expected <type 'int'>, got <type
 'str'>"
    }
  },
  "id": "TestRequest"
}
```

如果请求目标是不存在的设备，或者 Modbus RTU 网络不起作用，您可能会获得采用“无响应”格式的 `ModbusIOException`。

```
{
  "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"
}
```

本地日志文件

此组件使用以下日志文件。

```
/greengrass/v2/logs/aws.greengrass.Modbus.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。*/greengrass/v2*替换为 AWS IoT Greengrass 根文件夹的路径。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.Modbus.log
```

许可证

此组件包括以下第三方软件/许可：

- pymodbus /B SD 许可证
- pyserial /BSD 许可证

该组件是根据 Greengrass 核心软件许可协议发布的。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.8	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.7	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.5	错误修复和改进 • 修复了ReadDiscreteInput 操作中的一个问题。
2.1.4	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.6.0 版本的版本已更新。

版本	更改
2.1.0	新功能 • 添加ModbusBaudRate 、ModbusByteSize 、和ModbusStopBits 选项ModbusParity ，您可以指定这些选项来配置与 Modbus RTU 设备的串行通信。
2.0.8	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.7	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.6	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.5	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.0.4	Greengrass nucleus 版本 2.1.0 版本的版本已更新。
2.0.3	初始版本。

MQTT 网桥

MQTT 桥接组件 (`aws.greengrass.clientdevices.mqtt.Bridge`) 在客户端设备、本地 Greengrass 发布/订阅和之间中继 MQTT 消息。AWS IoT Core您可以使用此组件在自定义组件中处理来自客户端设备的 MQTT 消息，并将客户端设备与同AWS Cloud步。

> Note
>
> 客户端设备是连接到 Greengrass 核心设备以发送 MQTT 消息和数据进行处理的本地物联网设备。有关更多信息，请参阅 与本地物联网设备互动。

您可以使用此组件在以下消息代理之间中继消息：

- 本地 MQTT — 本地 MQTT 代理处理客户端设备和核心设备之间的消息。
- 本地发布/订阅 — 本地 Greengrass 消息代理处理核心设备上组件之间的消息。有关如何在 Greengrass 组件中与这些消息交互的更多信息，请参阅。发布/订阅本地消息

- AWS IoT Core— AWS IoT Core MQTT 代理处理物联网设备和AWS Cloud目标之间的消息。有关如何在 Greengrass 组件中与这些消息交互的更多信息，请参阅。发布/订阅 AWS IoT Core MQTT 消息

> **Note**
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 如果您将核心设备的 MQTT 代理组件配置为使用默认端口 8883 以外的端口，则必须使用 MQTT 网桥 v2.1.0 或更高版本。将其配置为在代理运行的端口上进行连接。
- 支持在 VPC 中运行 MQTT 网桥组件。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.3.0

下表列出了此组件版本 2.3.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.5.0	硬性

2.2.5 and 2.2.6

下表列出了此组件版本 2.2.5 和 2.2.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.5.0	硬性

2.2.3 and 2.2.4

下表列出了此组件版本 2.2.3 和 2.2.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.4.0	硬性

2.2.0 – 2.2.2

下表列出了此组件版本 2.2.0 到 2.2.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.3.0	硬性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.0.0 <2.2.0	硬性

2.0.0 to 2.1.0

下表列出了此组件的 2.0.0 到 2.1.0 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.0.0 <2.1.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

2.3.0

`mqttTopicMapping`

您要桥接的主题映射。此组件订阅有关源主题的消息，并将其收到的消息发布到目标主题。每个主题映射都定义了主题、来源类型和目标类型。

该对象包含以下信息：

`topicMappingNameKey`

此主题映射的名称。将 *`topicMappingNameKey`* 替换为可帮助您识别此主题映射的名称。

该对象包含以下信息：

`topic`

用于在源代理和目标代理之间架起桥梁的主题或主题筛选器。

您可以使用+和 # MQTT 主题通配符来中继与主题过滤器匹配的所有主题的消息。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的 MQTT 主题。

> Note
>
> 要在Pubsub源代理中使用 MQTT 主题通配符，必须使用 Greengrass 核心组件的 v2.6.0 或更高版本。

`targetTopicPrefix`

此组件中继消息时要添加到目标主题的前缀。

`source`

源消息代理。从以下选项中进行选择：

- `LocalMqtt`— 客户端设备通信的本地 MQTT 代理。

- `Pubsub`— 本地 Greengrass 发布/订阅消息代理。
- `IotCore`— AWS IoT Core MQTT 消息代理。

> Note
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

`source`并且`target`必须不同。

`target`

目标消息代理。从以下选项中进行选择：

- `LocalMqtt`— 客户端设备通信的本地 MQTT 代理。
- `Pubsub`— 本地 Greengrass 发布/订阅消息代理。
- `IotCore`— AWS IoT Core MQTT 消息代理。

> Note
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

`source`并且`target`必须不同。

mqtt5 RouteOptions

（可选）提供用于配置主题映射的选项，以便将消息从源主题与目标主题桥接起来。

该对象包含以下信息：

`mqtt5 RouteOptionsNameKey`

主题映射的路径选项的名称。将 *`mqtt5 RouteOptionsNameKey`* 替换为字段中定义的匹配*`topicMappingName##`*。`mqttTopicMapping`

该对象包含以下信息：

noLocal

（可选）启用后，网桥不会转发有关网桥本身发布的主题的消息。使用它来防止循环，如下所示：

```
{
    "mqtt5RouteOptions": {
        "toIoTCore": {
            "noLocal": true
        }
    },
    "mqttTopicMapping": {
        "toIoTCore": {
            "topic": "device",
            "source": "LocalMqtt",
            "target": "IotCore"
        },
        "toLocal": {
            "topic": "device",
            "source": "IotCore",
            "target": "LocalMqtt"
        }
    }
}
```

`noLocal`仅支持为的路`source`由`LocalMqtt`。

默认值：false

retainAsPublished

（可选）启用后，网桥转发的消息与向该路由的代理发布的消息具有相同的`retain`标志。

`retainAsPublished`仅支持为的路`source`由`LocalMqtt`。

默认值：false

mqtt

（可选）用于与本地代理通信的 MQTT 协议设置。

version

（可选）网桥用于与本地代理通信的 MQTT 协议版本。必须与在 nucleus 配置中选择的 MQTT 版本相同。

请从以下内容中选择：

- `mqtt3`
- `mqtt5`

当`mqttTopicMapping`对象的`source`或`target`字段设置为时，必须部署 MQTT 代理。`LocalMqtt`如果选择该`mqtt5`选项，则必须使用MQTT 5 经纪商 (EMQX)。

默认值：`mqtt3`

ackTimeoutSeconds

（可选）在操作失败之前等待 PUBACK、SUBACK 或 UNSUBACK 数据包的时间间隔。

默认值：60

connAckTimeout女士

（可选）在关闭连接之前等待 CONNACK 数据包的时间间隔。

默认值：20000（20 秒）

pingTimeoutMs

（可选）网桥等待收到来自本地代理的 PINGACK 消息的时间（以毫秒为单位）。如果等待时间超过超时时间，网桥将关闭，然后重新打开 MQTT 连接。此值必须小于`keepAliveTimeoutSeconds`。

默认值：30000（30 秒）

keepAliveTimeout秒

（可选）网桥为保持 MQTT 连接活动而发送的每条 PING 消息之间的间隔时间（以秒为单位）。此值必须大于`pingTimeoutMs`。

默认值：60

maxReconnectDelay女士

（可选）MQTT 重新连接的最大时间（以秒为单位）。

默认值：30000（30 秒）

minReconnectDelay女士

（可选）MQTT 重新连接的最短时间（以秒为单位）。

接收最大值

（可选）网桥可以发送的未确认的 QoS1 数据包的最大数量。

默认值：100

maximumPacketSize

客户端将接受的 MQTT 数据包的最大字节数。

默认：空（无限制）

sessionExpiryInterval

（可选）您可以请求在网桥和本地代理之间持续会话的时间（以秒为单位）。

默认值：4294967295（会话永不过期）

`brokerUri`

（可选）本地 MQTT 代理的 URI。如果您将 MQTT 代理配置为使用与默认端口 8883 不同的端口，则必须指定此参数。使用以下格式，并将*##*替换为 MQTT 代理运行的端口：`ssl://localhost:`*`port`*。

默认值：`ssl://localhost:8883`

startupTimeoutSeconds

（可选）组件启动的最长时间（以秒为单位）。`BROKEN`如果超过此超时时间，则组件的状态将更改为。

默认值：`120`

Example 示例：配置合并更新

以下示例配置更新指定了以下内容：

- 将来自客户端设备的消息中继到与AWS IoT Core主题过滤器匹配`clients/+/hello/world`的主题。

- 将来自客户端设备的消息中继到与主题过滤器匹配的clients/+/detections主题的本地发布/订阅，并在目标主题中添加events/input/前缀。生成的目标主题与events/input/clients/+/detections主题筛选条件相匹配。
- 将来自客户端设备的消息中继到AWS IoT Core与主题过滤器匹配clients/+/status的主题，并在目标主题中添加$aws/rules/StatusUpdateRule/前缀。此示例使用 B asic Ingest 将这些消息直接中继StatusUpdateRule到名为的AWS IoT规则以降低成本。

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example 示例：配置 MQTT 5

以下示例配置更新了以下内容：

- 允许网桥与本地代理一起使用 MQTT 5 协议。
- 为ClientDeviceHelloWorld主题映射配置 MQTT 保留为已发布设置。

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
```

```
        "topic": "clients/+/hello/world",
        "source": "LocalMqtt",
        "target": "IotCore"
      }
    },
    "mqtt5RouteOptions": {
      "ClientDeviceHelloWorld": {
        "retainAsPublished": true
      }
    },
    "mqtt": {
      "version": "mqtt5"
    }
  }
```

2.2.6

`mqttTopicMapping`

您要桥接的主题映射。此组件订阅有关源主题的消息，并将其收到的消息发布到目标主题。每个主题映射都定义了主题、来源类型和目标类型。

该对象包含以下信息：

`topicMappingNameKey`

此主题映射的名称。将 *`topicMappingNameKey`* 替换为可帮助您识别此主题映射的名称。

该对象包含以下信息：

`topic`

用于在源代理和目标代理之间架起桥梁的主题或主题筛选器。

您可以使用+和 # MQTT 主题通配符来中继与主题过滤器匹配的所有主题的消息。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的 MQTT 主题。

> **Note**
>
> 要在Pubsub源代理中使用 MQTT 主题通配符，必须使用 Greengrass 核心组件的 v2.6.0 或更高版本。

`targetTopicPrefix`

此组件中继消息时要添加到目标主题的前缀。

`source`

源消息代理。从以下选项中进行选择：

- `LocalMqtt`— 客户端设备通信的本地 MQTT 代理。
- `Pubsub`— 本地 Greengrass 发布/订阅消息代理。
- `IotCore`— AWS IoT Core MQTT 消息代理。

> **Note**
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

`source`并且`target`必须不同。

`target`

目标消息代理。从以下选项中进行选择：

- `LocalMqtt`— 客户端设备通信的本地 MQTT 代理。
- `Pubsub`— 本地 Greengrass 发布/订阅消息代理。
- `IotCore`— AWS IoT Core MQTT 消息代理。

> **Note**
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

`source`并且`target`必须不同。

brokerUri

（可选）本地 MQTT 代理的 URI。如果您将 MQTT 代理配置为使用与默认端口 8883 不同的端口，则必须指定此参数。使用以下格式，并将##替换为 MQTT 代理运行的端口：ssl://localhost:*port*。

默认值：ssl://localhost:8883

startupTimeoutSeconds

（可选）组件启动的最长时间（以秒为单位）。BROKEN如果超过此超时时间，则组件的状态将更改为。

默认值：120

Example 示例：配置合并更新

以下示例配置更新指定了以下内容：

- 将来自客户端设备的消息中继到与AWS IoT Core主题过滤器匹配clients/+/hello/world的主题。
- 将来自客户端设备的消息中继到与主题过滤器匹配的clients/+/detections主题的本地发布/订阅，并在目标主题中添加events/input/前缀。生成的目标主题与events/input/clients/+/detections主题筛选条件相匹配。
- 将来自客户端设备的消息中继到AWS IoT Core与主题过滤器匹配clients/+/status的主题，并在目标主题中添加$aws/rules/StatusUpdateRule/前缀。此示例使用 B asic Ingest 将这些消息直接中继StatusUpdateRule到名为的AWS IoT规则以降低成本。

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
```

```
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.2.0 - 2.2.5

`mqttTopicMapping`

您要桥接的主题映射。此组件订阅有关源主题的消息，并将其收到的消息发布到目标主题。每个主题映射都定义了主题、来源类型和目标类型。

该对象包含以下信息：

topicMappingNameKey

此主题映射的名称。将 *topicMappingNameKey* 替换为可帮助您识别此主题映射的名称。

该对象包含以下信息：

`topic`

用于在源代理和目标代理之间架起桥梁的主题或主题筛选器。

您可以使用+和 # MQTT 主题通配符来中继与主题过滤器匹配的所有主题的消息。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的 MQTT 主题。

> **Note**
>
> 要在Pubsub源代理中使用 MQTT 主题通配符，必须使用 Greengrass 核心组件的 v2.6.0 或更高版本。

`targetTopicPrefix`

此组件中继消息时要添加到目标主题的前缀。

`source`

源消息代理。从以下选项中进行选择：

- `LocalMqtt`— 客户端设备通信的本地 MQTT 代理。
- `Pubsub`— 本地 Greengrass 发布/订阅消息代理。
- `IotCore`— AWS IoT Core MQTT 消息代理。

> **Note**
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

`source`并且`target`必须不同。

`target`

目标消息代理。从以下选项中进行选择：

- `LocalMqtt`— 客户端设备通信的本地 MQTT 代理。
- `Pubsub`— 本地 Greengrass 发布/订阅消息代理。
- `IotCore`— AWS IoT Core MQTT 消息代理。

> **Note**
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

`source`并且`target`必须不同。

`brokerUri`

（可选）本地 MQTT 代理的 URI。如果您将 MQTT 代理配置为使用与默认端口 8883 不同的端口，则必须指定此参数。使用以下格式，并将*##*替换为 MQTT 代理运行的端口：`ssl://localhost:port`。

默认值：`ssl://localhost:8883`

Example 示例：配置合并更新

以下示例配置更新指定了以下内容：

- 将来自客户端设备的消息中继到与AWS IoT Core主题过滤器匹配clients/+/hello/world的主题。
- 将来自客户端设备的消息中继到与主题过滤器匹配的clients/+/detections主题的本地发布/订阅，并在目标主题中添加events/input/前缀。生成的目标主题与events/input/clients/+/detections主题筛选条件相匹配。
- 将来自客户端设备的消息中继到AWS IoT Core与主题过滤器匹配clients/+/status的主题，并在目标主题中添加$aws/rules/StatusUpdateRule/前缀。此示例使用 B asic Ingest 将这些消息直接中继StatusUpdateRule到名为的AWS IoT规则以降低成本。

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.1.x

mqttTopicMapping

您要桥接的主题映射。此组件订阅有关源主题的消息，并将其收到的消息发布到目标主题。每个主题映射都定义了主题、来源类型和目标类型。

该对象包含以下信息：

topicMappingNameKey

此主题映射的名称。将 *topicMappingNameKey* 替换为可帮助您识别此主题映射的名称。

该对象包含以下信息：

`topic`

用于在源代理和目标代理之间架起桥梁的主题或主题筛选器。

如果您指定`LocalMqtt`或`IotCore`源代理，则可以使用`+`和 `#` MQTT 主题通配符来中继与主题筛选条件匹配的所有主题的消息。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的 [MQTT 主题]。

`source`

源消息代理。从以下选项中进行选择：

- `LocalMqtt`— 客户端设备通信的本地 MQTT 代理。
- `Pubsub`— 本地 Greengrass 发布/订阅消息代理。
- `IotCore`— AWS IoT Core MQTT 消息代理。

> **Note**
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅[配置 MQTT 超时和缓存设置]。

`source`并且`target`必须不同。

`target`

目标消息代理。从以下选项中进行选择：

- `LocalMqtt`— 客户端设备通信的本地 MQTT 代理。
- `Pubsub`— 本地 Greengrass 发布/订阅消息代理。
- `IotCore`— AWS IoT Core MQTT 消息代理。

> **Note**
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

`source`并且`target`必须不同。

`brokerUri`

（可选）本地 MQTT 代理的 URI。如果您将 MQTT 代理配置为使用与默认端口 8883 不同的端口，则必须指定此参数。使用以下格式，并将*##*替换为 MQTT 代理运行的端口：`ssl://localhost:port`。

默认值：`ssl://localhost:8883`

Example 示例：配置合并更新

以下示例配置更新指定将来自客户端设备的消息中继到`clients/MyClientDevice1/hello/world`和`clients/MyClientDevice2/hello/world`主题AWS IoT Core上。

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.0.x

`mqttTopicMapping`

您要桥接的主题映射。此组件订阅有关源主题的消息，并将其收到的消息发布到目标主题。每个主题映射都定义了主题、来源类型和目标类型。

该对象包含以下信息：

`topicMappingNameKey`

此主题映射的名称。将 *`topicMappingNameKey`* 替换为可帮助您识别此主题映射的名称。

该对象包含以下信息：

`topic`

用于在源代理和目标代理之间架起桥梁的主题或主题筛选器。

如果您指定`LocalMqtt`或`IotCore`源代理，则可以使用`+`和 `#` MQTT 主题通配符来中继与主题筛选条件匹配的所有主题的消息。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的 MQTT 主题。

`source`

源消息代理。从以下选项中进行选择：

- `LocalMqtt`— 客户端设备通信的本地 MQTT 代理。
- `Pubsub`— 本地 Greengrass 发布/订阅消息代理。
- `IotCore`— AWS IoT Core MQTT 消息代理。

> **Note**
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

`source`并且`target`必须不同。

`target`

目标消息代理。从以下选项中进行选择：

- `LocalMqtt`— 客户端设备通信的本地 MQTT 代理。
- `Pubsub`— 本地 Greengrass 发布/订阅消息代理。
- `IotCore`— AWS IoT Core MQTT 消息代理。

> **Note**
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

`source`并且`target`必须不同。

Example 示例：配置合并更新

以下示例配置更新指定将来自客户端设备的消息中继到`clients/MyClientDevice1/hello/world`和`clients/MyClientDevice2/hello/world`主题AWS IoT Core上。

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.3.1	错误修复和改进 修复了本地 MQTT 客户端进入断开连接循环的问题。
2.3.0	新功能 为本地 MQTT 源AWS IoT Core和本地 MQTT 源之间的桥接添加了 MQTT5 支持。

版本	更改
2.2.6	新功能 添加新的`startupTimeoutSeconds` 配置选项。
2.2.5	版本已针对客户端设备身份验证版本 2.4.0 版本进行了更新。
2.2.4	G reengrass 客户端设备身份验证版本 2.3.0 版本的版本已更新。
2.2.3	此版本包含错误修复和改进。
2.2.2	错误修复和改进 • 日志调整。
2.2.1	错误修复和改进 修复了可能导致 MQTT 桥无法订阅 MQTT 主题的问题。
2.2.0	新功能 • 当您将本地发布/订阅指定为源消息代理时，添加对 MQTT 主题通配符（#和+）的支持。 此功能需要 Greengrass nucleus 组件的 v2.6.0 或更高版本。 • 添加`targetTopicPrefix` 选项，您可以指定该选项来配置 MQTT 网桥，使其在中继消息时向目标主题添加前缀。
2.1.1	错误修复和改进 • 修复了此组件如何处理配置重置更新的问题。 • 降低证书轮换时 MQTT 客户端断开连接的频率。
2.1.0	新功能 • 添加`brokerUri` 参数，使您能够使用非默认 MQTT 代理端口。
2.0.1	此版本包括错误修复和改进。
2.0.0	初始版本。

MQTT 3.1.1 经纪商 (Moquette)

Moquette MQTT 代理组件 (`aws.greengrass.clientdevices.mqtt.Moquette`) 处理客户端设备和 Greengrass 核心设备之间的 MQTT 消息。该组件提供了 M oquette MQTT 代理的修改版本。部署此 MQTT 代理来运行轻量级 MQTT 代理。有关如何选择 MQTT 代理的更多信息，请参阅选择一个 MQTT 经纪商。

该交易商实现了 MQTT 3.1.1 协议。它包括对 QoS 0、QoS 1、QoS 2 保留消息、最后遗嘱消息和持久会话的支持。

> **Note**
>
> 客户端设备是连接到 Greengrass 核心设备以发送 MQTT 消息和数据进行处理的本地物联网设备。有关更多信息，请参阅 与本地物联网设备互动。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 核心设备必须能够接受 MQTT 代理运行的端口上的连接。默认情况下，此组件在端口 8883 上运行 MQTT 代理。配置此组件时，可以指定不同的端口。

  如果您指定其他端口，并且使用 MQTT 网桥组件将 MQTT 消息中继给其他代理，则必须使用 MQTT 网桥 v2.1.0 或更高版本。将其配置为使用 MQTT 代理运行的端口。

  如果您指定其他端口，并使用 IP 检测器组件管理 MQTT 代理端点，则必须使用 IP 检测器 v2.1.0 或更高版本。将其配置为报告 MQTT 代理运行的端口。
- 支持 Moquette MQTT 代理组件在 VPC 中运行。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.3.2 – 2.3.6

下表列出了此组件版本 2.3.2 到 2.3.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.5.0	硬性

2.3.0 and 2.3.1

下表列出了此组件版本 2.3.0 和 2.3.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.4.0	硬性

2.2.0

下表列出了此组件版本 2.2.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.3.0	硬性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.0.0 <2.2.0	硬性

2.0.0 - 2.0.2

下表列出了此组件的 2.0.0 到 2.0.2 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.0.0 <2.1.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`moquette`

（可选）要使用的 M oquette MQTT 代理配置。您可以在此组件中配置 Moqeutte 配置选项的子集。有关更多信息，请参阅 Moquette 配置文件中的内联注释。

该对象包含以下信息：

`ssl_port`

（可选）MQTT 代理运行的端口。

> **Note**
>
> 如果您指定其他端口，并且使用 MQTT 网桥组件将 MQTT 消息中继给其他代理，则必须使用 MQTT 网桥 v2.1.0 或更高版本。将其配置为使用 MQTT 代理运行的端口。
> 如果您指定其他端口，并使用 IP 检测器组件管理 MQTT 代理端点，则必须使用 IP 检测器 v2.1.0 或更高版本。将其配置为报告 MQTT 代理运行的端口。

默认：`8883`

`host`

（可选）MQTT 代理绑定的接口。例如，您可以更改此参数，使 MQTT 代理仅绑定到特定的本地网络。

默认：`0.0.0.0`（绑定到所有网络接口）

startupTimeoutSeconds

（可选）组件启动的最长时间（以秒为单位）。`BROKEN`如果超过此超时时间，则组件的状态将更改为。

默认：120

Example 示例：配置合并更新

以下示例配置指定在端口 443 上运行 MQTT 代理。

```
{
  "moquette": {
    "ssl_port": "443"
  }
}
```

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.3.6	错误修复和改进 • 常规错误修复和性能改进。
2.3.5	错误修复和改进 • 将 Moquette 更新至版本 0.17。
2.3.4	错误修复和改进 • 修复了由于客户端 ID 重复而导致客户端在发送或接收消息时可能遇到无效会话错误的问题。此问题导致客户端会话关闭。
2.3.3	新功能 添加了一个新的`startupTimeoutSeconds` 配置选项。
2.3.2	版本已针对客户端设备身份验证版本 2.4.0 版本进行了更新。
2.3.1	错误修复和改进 • 修复了由于会话无效而导致客户端在尝试重新连接后可能断开连接的争用情况。
2.3.0	添加对证书链的支持。
2.2.0	版本已针对客户端设备身份验证版本 2.2.0 版本进行了更新。
2.1.0	错误修复和改进 • 将此组件更新为使用 Moquett e 版本 0.16，该版本提高了性能并包括其他几项改进。 • 修复了在某些情况下本地 MQTT 服务器证书的轮换频率高于预期的问题。 要应用此修复程序，您还必须使用 v2.1.0 或更高版本的客户端设备身份验证组件。

版本	更改
2.0.2	错误修复和改进 • 将 MQTT 消息的最大大小从 8,092 字节增加到 128 千字节。有效的 MQTT 消息负载限制略低，因为消息大小限制包括消息头。 • 在ssl_port参数中添加对整数值的支持。
2.0.1	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.0	初始版本。

MQTT 5 经纪商 (EMQX)

EMQX MQTT 代理组件 (aws.greengrass.clientdevices.mqtt.EMQX) 处理客户端设备和 Greengrass 核心设备之间的 MQTT 消息。此组件提供了 EMQX MQTT 5. 0 代理的修改版本。部署此 MQTT 代理，以便在客户端设备和核心设备之间的通信中使用 MQTT 5 功能。有关如何选择 MQTT 代理的更多信息，请参阅选择一个 MQTT 经纪商。

该交易商实现了 MQTT 5.0 协议。它包括对会话和消息过期间隔、用户属性、共享订阅、主题别名等的支持。MQTT 5 向后兼容 MQTT 3.1.1，因此，如果您运行 M oquette MQTT 3.1.1 代理，则可以将其替换为 EMQX MQTT 5 代理，客户端设备可以继续照常连接和运行。

> **Note**
>
> 客户端设备是连接到 Greengrass 核心设备以发送 MQTT 消息和数据进行处理的本地物联网设备。有关更多信息，请参阅 与本地物联网设备互动。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置

- 本地日志文件
- 许可证
- 更改日志

版本

此组件有以下版本：

- 2.0.x
- 1.2.x
- 1.1.x
- 1.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 核心设备必须能够接受 MQTT 代理运行的端口上的连接。默认情况下，此组件在端口 8883 上运行 MQTT 代理。配置此组件时，可以指定不同的端口。

  如果您指定其他端口，并且使用 MQTT 网桥组件将 MQTT 消息中继给其他代理，则必须使用 MQTT 网桥 v2.1.0 或更高版本。将其配置为使用 MQTT 代理运行的端口。

  如果您指定其他端口，并使用 IP 检测器组件管理 MQTT 代理端点，则必须使用 IP 检测器 v2.1.0 或更高版本。将其配置为报告 MQTT 代理运行的端口。

- 在 Linux 核心设备上，在核心设备上安装并配置 Docker：
  - 安装在 Greengr@@ ass 核心设备上的 Docker E ngine 1.9.1 或更高版本。版本 20.10 是经验证可与 AWS IoT Greengrass 核心软件配合使用的最新版本。在部署运行 Docker 容器的组件之前，必须直接在核心设备上安装 Docker。
  - 在部署此组件之前，Docker 守护程序已启动并在核心设备上运行。
  - 运行此组件的系统用户必须具有 root 或管理员权限。或者，您可以以组中的系统用户身份运行此`docker`组件，并将此组件的`requiresPrivileges`选项配置为在没有权限的情况下运行 EQMX MQTT 代理。`false`
- EMQX MQTT 代理组件支持在 VPC 中运行。
- 该平台不支持 EMQX MQTT 代理组件。`armv7`

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.0.0

下表列出了此组件版本 2.0.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.5.0	硬性

1.2.2 – 1.2.3

下表列出了此组件版本 1.2.2 到 1.2.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.5.0	硬性

1.2.0 and 1.2.1

下表列出了此组件版本 1.2.0 和 1.2.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.4.0	硬性

1.0.0 and 1.1.0

下表列出了此组件版本 1.0.0 和 1.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
客户端设备身份验证	>=2.2.0 <2.3.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

2.0.0

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

Important

如果您使用版本 2 的 MQTT 5 代理 (EMQX) 组件，则必须更新您的配置文件。版本 1 配置文件不适用于版本 2。

emqxConfig

（可选）要使用的 EMQX MQTT 代理配置。您可以在此组件中设置 EMQX 配置选项。

当你使用 EMQX 代理时，Greengrass 使用默认配置。除非您使用此字段对其进行修改，否则将使用此配置。

修改以下配置设置会导致 EMQX 代理组件重新启动。其他配置更改无需重新启动组件即可生效。

- emqxConfig/cluster
- emqxConfig/node
- emqxConfig/rpc

> **Note**
>
> aws.greengrass.clientdevices.mqtt.EMQX允许您配置对安全敏感的选项。这些包括 TLS 设置、身份验证和授权提供商。我们推荐使用双向 TLS 身份验证和 Greengrass 客户端设备身份验证提供程序的默认配置。

Example 示例：默认配置

以下示例显示了为 MQTT 5 (EMQX) 代理设置的默认值。您可以使用emqxConfig配置设置覆盖这些设置。

```
{
  "authorization": {
    "no_match": "deny",
    "sources": []
  },
  "node": {
    "cookie": "<placeholder>"
  },
  "listeners": {
     "ssl": {
       "default": {
         "ssl_options": {
           "keyfile": "{work:path}\\data\\key.pem",
           "certfile": "{work:path}\\data\\cert.pem",
           "cacertfile": null,
           "verify": "verify_peer",
           "versions": ["tlsv1.3", "tlsv1.2"],
           "fail_if_no_peer_cert": true
         }
       }
     },
     "tcp": {
       "default": {
         "enabled": false
       }
     },
```

```
      "ws": {
        "default": {
          "enabled": false
        }
      },
      "wss": {
        "default": {
          "enabled": false
        }
      }
   },
   "plugins": {
     "states": [{"name_vsn": "gg-1.0.0", "enable": true}],
     "install_dir": "plugins"
   }
 }
```

身份验证模式

（可选）为经纪人设置授权提供商。可以是以下任一值：

- enabled—（默认）使用 Greengrass 身份验证和授权提供程序。
- bypass_on_failure— 使用 Greengrass 身份验证提供程序，如果 Greengrass 拒绝身份验证或授权，则使用 EMQX 提供程序链中剩余的所有身份验证提供程序。
- bypass— Greengrass 提供程序已禁用。身份验证和授权由 EMQX 提供商链处理。

requiresPrivilege

（可选）在 Linux 核心设备上，您可以指定在没有根权限或管理员权限的情况下运行 EMQX MQTT 代理。如果将此选项设置为false，则运行此组件的系统用户必须是该docker组的成员。

默认：true

startupTimeoutSeconds

（可选）EMQX MQTT 代理启动的最长时间（以秒为单位）。BROKEN如果超过此超时时间，则组件的状态将更改为。

默认：90

ipcTimeoutSeconds

（可选）组件等待 Greengrass 核响应进程间通信 (IPC) 请求的最长时间（以秒为单位）。如果此组件在检查客户端设备是否获得授权时报告超时错误，请增加此数字。

默认：5

`crtLogLevel`

（可选） AWS 公共运行时 (CRT) 库的日志级别。

默认为 EMQX MQTT 代理日志级别（`log.level`中）。`emqx`

`restartIdentifier`

（可选）配置此选项以重启 EMQX MQTT 代理。当此配置值更改时，此组件会重新启动 MQTT 代理。您可以使用此选项强制客户端设备断开连接。

`dockerOptions`

（可选）仅在 Linux 操作系统上配置此选项以向 Docker 命令行添加参数。例如，要映射其他端口，请使用 -p Docker 参数：

```
"-p 1883:1883"
```

Example 示例：将 v1.x 配置文件更新为 v2.x

以下示例显示了将 v1.x 配置文件更新到 2.x 版本所需的更改。

版本 1.x 配置文件：

```
{
    "emqx": {
        "listener.ssl.external": "443",
        "listener.ssl.external.max_connections": "1024000",
        "listener.ssl.external.max_conn_rate": "500",
        "listener.ssl.external.rate_limit": "50KB,5s",
        "listener.ssl.external.handshake_timeout": "15s",
        "log.level": "warning"
    },
    "mergeConfigurationFiles": {
        "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
    }
}
```

v2 的等效配置文件：

```
{
    "emqxConfig": {
        "listeners": {
            "ssl": {
                "default": {
                    "bind": "8883",
                    "max_connections": "1024000",
                    "max_conn_rate": "500",
                    "handshake_timeout": "15s"
                }
            }
        },
        "log": {
            "console": {
              "enable": true,
              "level": "warning"
            }
        }
    },
    "authMode": "enabled"
}
```

没有与`listener.ssl.external.rate_limit`配置条目对应的内容。`use_greengrass_managed_certificates`配置选项已被删除。

Example 示例：为代理设置新端口

以下示例将 MQTT 代理运行的端口从默认的 8883 更改为端口 1234。如果您使用的是 Linux，请包含该`dockerOptions`字段。

```
{
  "emqxConfig": {
    "listeners": {
      "ssl": {
        "default": {
          "bind": 1234
        }
      }
    }
  },
  "dockerOptions": "-p 1234:1234"
}
```

Example 示例：调整 MQTT 代理的日志级别

以下示例将 MQTT 代理的日志级别更改为。debug您可以从以下日志级别中进行选择：

- debug
- info
- notice
- warning
- error
- critical
- alert
- emergency

默认日志级别为warning。

```
{
  "emqxConfig": {
    "log": {
      "console": {
         "level": "debug"
      }
    }
  }
}
```

Example 示例：启用 EMQX 控制面板

以下示例启用 EMQX 控制面板，以便您可以监控和管理您的经纪商。如果您使用的是 Linux，请包含该dockerOptions字段。

```
{
  "emqxConfig": {
    "dashboard": {
      "listeners": {
        "http": {
          "bind": 18083
        }
      }
```

```
    }
  },
  "dockerOptions": "-p 18083:18083"
}
```

1.0.0 - 1.2.2

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`emqx`

（可选）要使用的 EMQX MQTT 代理配置。您可以在此组件中配置 EMQX 配置选项的子集。

该对象包含以下信息：

`listener.ssl.external`

（可选）MQTT 代理运行的端口。

> **Note**
>
> 如果您指定其他端口，并且使用 MQTT 网桥组件将 MQTT 消息中继给其他代理，则必须使用 MQTT 网桥 v2.1.0 或更高版本。将其配置为使用 MQTT 代理运行的端口。
>
> 如果您指定其他端口，并使用 IP 检测器组件管理 MQTT 代理端点，则必须使用 IP 检测器 v2.1.0 或更高版本。将其配置为报告 MQTT 代理运行的端口。

默认：`8883`

`listener.ssl.external.max_connections`

（可选）MQTT 代理支持的最大并发连接数。

默认：`1024000`

`listener.ssl.external.max_conn_rate`

（可选）MQTT 代理每秒可以接收的最大新连接数。

默认：`500`

`listener.ssl.external.rate_limit`

（可选）与 MQTT 代理的所有连接的带宽限制。按以下格式指定该带宽的带宽和持续时间，用逗号 (`,`) 分隔：`bandwidth,duration`。例如，您可以指定`50KB,5s`将 MQTT 代理限制为每 5 秒 50 千字节 (KB) 的数据。

`listener.ssl.external.handshake_timeout`

（可选）MQTT 代理等待完成对新连接进行身份验证的时间。

默认：`15s`

`mqtt.max_packet_size`

（可选）MQTT 消息的最大大小。

默认值：`268435455`（256 MB 减去 1）

`log.level`

（可选）MQTT 代理的日志级别。从以下选项中进行选择：

- `debug`
- `info`
- `notice`
- `warning`
- `error`
- `critical`
- `alert`
- `emergency`

默认日志级别为`warning`。

`requiresPrivilege`

（可选）在 Linux 核心设备上，您可以指定在没有根权限或管理员权限的情况下运行 EMQX MQTT 代理。如果将此选项设置为`false`，则运行此组件的系统用户必须是该`docker`组的成员。

默认：`true`

`startupTimeoutSeconds`

（可选）EMQX MQTT 代理启动的最长时间（以秒为单位）。BROKEN如果超过此超时时间，则组件的状态将更改为。

默认：90

`ipcTimeoutSeconds`

（可选）组件等待 Greengrass 核响应进程间通信 (IPC) 请求的最长时间（以秒为单位）。如果此组件在检查客户端设备是否获得授权时报告超时错误，请增加此数字。

默认：5

`crtLogLevel`

（可选） AWS 公共运行时 (CRT) 库的日志级别。

默认为 EMQX MQTT 代理日志级别（`log.level`中）。emqx

`restartIdentifier`

（可选）配置此选项以重启 EMQX MQTT 代理。当此配置值更改时，此组件会重新启动 MQTT 代理。您可以使用此选项强制客户端设备断开连接。

`dockerOptions`

（可选）仅在 Linux 操作系统上配置此选项以向 Docker 命令行添加参数。例如，要映射其他端口，请使用 -p Docker 参数：

```
"-p 1883:1883"
```

`mergeConfigurationFiles`

（可选）配置此选项以添加或覆盖指定 EMQX 配置文件中的默认值。有关配置文件及其格式的信息，请参阅 EMQX 4.0 文档中的配置。您指定的值将附加到配置文件中。

以下示例更新了该`etc/emqx.conf`文件。

```
"mergeConfigurationFiles": {
    "etc/emqx.conf": "broker.sys_interval=30s\nbroker.sys_heartbeat=10s"
},
```

除了 EMQX 支持的配置文件外，Greengrass 还支持一个名为 EMQX 配置 Greengrass 身份验证插件的文件。etc/plugins/aws_greengrass_emqx_auth.conf有两个支持的选项，auth_mode和use_greengrass_managed_certificates。要使用其他身份验证提供商，请将该auth_mode选项设置为以下选项之一：

- enabled—（默认）使用 Greengrass 身份验证和授权提供程序。
- bypass_on_failure— 使用 Greengrass 身份验证提供程序，如果 Greengrass 拒绝身份验证或授权，则使用 EMQX 提供程序链中剩余的所有身份验证提供程序。
- bypass— Greengrass 提供程序已禁用。然后，身份验证和授权由 EMQX 提供商链处理。

如果use_greengrass_managed_certificates为true，则此选项表示 Greengrass 管理代理 TLS 证书。如果false，则表示您通过其他来源提供证书。

以下示例更新了etc/plugins/aws_greengrass_emqx_auth.conf配置文件中的默认值。

```
"mergeConfigurationFiles": {
    "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
  },
```

> **Note**
>
> aws.greengrass.clientdevices.mqtt.EMQX允许您配置对安全敏感的选项。这些包括 TLS 设置、身份验证和授权提供商。推荐的配置是使用双向 TLS 身份验证和 Greengrass 客户端设备身份验证提供程序的默认配置。

replaceConfigurationFiles

（可选）配置此选项以替换指定的 EMQX 配置文件。您指定的值将替换整个现有配置文件。您无法在本节中指定该etc/emqx.conf文件。必须使用mergeConfigurationFile才能修改etc/emqx.conf。

Example 示例：配置合并更新

以下示例配置指定在端口 443 上运行 MQTT 代理。

```
{
```

```
  "emqx": {
    "listener.ssl.external": "443",
    "listener.ssl.external.max_connections": "1024000",
    "listener.ssl.external.max_conn_rate": "500",
    "listener.ssl.external.rate_limit": "50KB,5s",
    "listener.ssl.external.handshake_timeout": "15s",
    "log.level": "warning"
  },
  "requiresPrivilege": "true",
  "startupTimeoutSeconds": "90",
  "ipcTimeoutSeconds": "5"
}
```

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将 *`/greengrass/v2`* 或 *`C:\greengrass\v2`* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log -Tail 10 -Wait
  ```

许可证

在 Windows 操作系统上，该软件包括根据微软软件许可条款分发的代码——微软 Visual Studio 社区 2022。下载此软件即表示您同意该代码的许可条款。

该组件是根据 Greengrass 核心软件许可协议发布的。

更改日志

下表描述了该组件的每个版本中的更改。

v2.x

版本	更改
2.0.0	此版本的 MQTT 5 代理 (EMQX) 需要的配置参数与 1.x 版本不同。如果您使用版本 1.x 的非默认配置，则必须更新 2.x 版的组件配置。有关更多信息，请参阅 配置。 新功能 • 将 MQTT 代理升级到 EMQX 5.1.1。 • 无需重新启动组件即可更改代理配置。 更新 • 添加一个新的emqxConfig 配置字段，用于替换emqxmergeConfigurationFiles 、和replaceConfigurationFiles 配置字段。

v1.x

版本	更改
1.2.3	错误修复和改进 • 修复了先前通过断开连接并重新验证客户端进行身份验证后，客户端无法与 EMQX 交互的问题。
1.2.2	版本已针对客户端设备身份验证版本 2.4.0 版本进行了更新。

版本	更改
1.2.1	错误修复和改进 • 修复了如果尚不存在 Visual C++ 可再发行版，则该组件无法在 Windows 上启动的问题。 • 将 EMQX 更新至版本 4.4.14。
1.2.0	添加对证书链的支持。
1.1.0	新功能 • 增加了对 EMQX 配置的支持，包括代理选项和插件。 错误修复和改进 • 将 EMQX 更新至版本 4.4.9。
1.0.1	修复了 TLS 握手期间导致某些 MQTT 客户端无法连接的问题。
1.0.0	初始版本。

Nucleus 遥测发射器

nucleus 遥测发射器组件 (`aws.greengrass.telemetry.NucleusEmitter`) 收集系统健康遥测数据，并将其持续发布到本地主题和 MQTT 主题。AWS IoT Core此组件使您能够在 Greengrass 核心设备上收集实时系统遥测数据。有关向亚马逊发布系统遥测数据的 Greengrass 遥测代理的信息，请参阅。 EventBridge 从AWS IoT Greengrass核心设备收集系统运行状况遥测数据

默认情况下，nucleus 遥测发射器组件每 60 秒将遥测数据发布到以下本地发布/订阅主题。

```
$local/greengrass/telemetry
```

默认情况下，nucleus 遥测发射器组件不会发布到 AWS IoT Core MQTT 主题中。您可以将此组件配置为在部署 AWS IoT Core MQTT 主题时将其发布到 MQTT 主题。使用 MQTT 主题向发布数据需视AWS IoT Core定价而定。AWS Cloud

AWS IoT Greengrass提供了多个社区组件，可帮助您使用 InfluxDB 和 Grafana 在核心设备上本地分析和可视化遥测数据。这些组件使用来自原子核发射器组件的遥测数据。有关更多信息，请参阅 InfluxDB 发布者组件的自述文件。

主题

- 版本
- 类型
- 操作系统
- 依赖项
- 配置
- 输出数据
- 使用量
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 1.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组

件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

1.0.8

下表列出了此组件版本 1.0.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.4.0 <2.13.0	硬性

1.0.7

下表列出了此组件版本 1.0.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.4.0 <2.12.0	硬性

1.0.6

下表列出了此组件版本 1.0.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.4.0 <2.11.0	硬性

1.0.5

下表列出了此组件版本 1.0.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.4.0 <2.10.0	硬性

1.0.4

下表列出了此组件版本 1.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.4.0 <2.9.0	硬性

1.0.3

下表列出了此组件版本 1.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.4.0 <2.8.0	硬性

1.0.2

下表列出了此组件版本 1.0.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.4.0 <2.7.0	硬性

1.0.1

下表列出了此组件版本 1.0.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.4.0 <2.6.0	硬性

1.0.0

下表列出了此组件版本 1.0.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.4.0 <2.5.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`pubSubPublish`

（可选）定义是否向`$local/greengrass/telemetry`主题发布遥测数据。支持的值为 `true` 和 `false`。

默认值：`true`

`mqttTopic`

（可选）此组件向其发布遥测数据的 AWS IoT Core MQTT 主题。

将此值设置为要向其发布遥测数据的 AWS IoT Core MQTT 主题。当此值为空时，nucleus 发射器不会将遥测数据发布到。AWS Cloud

> **Note**
> 使用 MQTT 主题向发布数据需视AWS IoT Core定价而定。AWS Cloud

默认值：`""`

`telemetryPublishIntervalMs`

（可选）组件发布遥测数据的间隔时间（以毫秒为单位）。如果将此值设置为低于支持的最小值，则组件将改用最小值。

> **Note**
> 较短的发布间隔会导致核心设备上的 CPU 使用率更高。我们建议您从默认发布间隔开始，然后根据设备的 CPU 使用率进行调整。

最低：500

默认值：60000

Example 示例：配置合并更新

以下示例显示了一个配置合并更新示例，该更新允许每 5 秒向`$local/greengrass/telemetry`主题和 `greengrass/myTelemetry` AWS IoT Core MQTT 主题发布遥测数据。

```
{
  "pubSubPublish": "true",
  "mqttTopic": "greengrass/myTelemetry",
  "telemetryPublishIntervalMs": 5000
}
```

输出数据

此组件将遥测指标作为 JSON 数组发布以下主题。

本地话题：`$local/greengrass/telemetry`

您也可以选择将遥测指标发布到 AWS IoT Core MQTT 主题。有关主题的更多信息，请参阅《AWS IoT Core开发人员指南》中的 MQTT 主题。

Example 示例数据

```
[
  {
    "A": "Average",
    "N": "CpuUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Percent",
    "V": 26.21981271562346
  },
  {
    "A": "Count",
    "N": "TotalNumberOfFDs",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Count",
    "V": 7316
  },
```

```
  {
    "A": "Count",
    "N": "SystemMemUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Megabytes",
    "V": 10098
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStarting",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsInstalled",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStateless",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStopping",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsBroken",
    "NS": "GreengrassComponents",
```

```
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsRunning",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 7
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsErrored",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsNew",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsFinished",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 2
  }
]
```


输出数组包含具有以下属性的指标列表：

A

指标的聚合类型。

对于该CpuUsage指标，此属性设置为，Average因为该指标的发布值是自上次发布事件以来的平均 CPU 使用量。

对于所有其他指标，nucleus 发射器不会聚合指标值，并且此属性设置为。Count

N

指标的名称。

NS

指标命名空间。

TS

收集数据的时间戳。

U

指标值的单位。

V

指标值。

Nucleus 发射器发布以下指标：

名称	描述
系统	
SystemMemUsage	Greengrass 核心设备上所有应用程序（包括操作系统）当前使用的内存量。
CpuUsage	Greengrass 核心设备上所有应用程序（包括操作系统）当前使用的 CPU 量。
TotalNumberOfFDs	Greengrass 核心设备操作系统存储的文件描述符的数量。一个文件描述符可以唯一地标识一个打开的文件。

名称	描述	
Greengrass 核		
`NumberOfComponentsRunning`	在 Greengrass 核心设备上运行的组件数量。	
`NumberOfComponentsErrored`	Greengrass 核心设备上处于错误状态的组件数量。	
`NumberOfComponentsInstalled`	安装在 Greengrass 核心设备上的组件数量。	
`NumberOfComponentsStarting`	在 Greengrass 核心设备上启动的组件数量。	
`NumberOfComponentsNew`	Greengrass 核心设备上新增的组件数量。	
`NumberOfComponentsStopping`	在 Greengrass 核心设备上停止运行的组件数量。	
`NumberOfComponentsFinished`	在 Greengrass 核心设备上完成的组件数量。	
`NumberOfComponentsBroken`	Greengrass 核心设备上损坏的组件数量。	
`NumberOfComponentsStateless`	Greengrass 核心设备上处于无状态的组件数量。	

使用量

要使用系统运行状况遥测数据，您可以创建自定义组件，订阅 nucleus 发射器发布遥测数据的主题，并根据需要对这些数据做出反应。由于 nucleus 发射器组件提供了将遥测数据发布到本地主题的选项，因此您可以订阅该主题，并使用已发布的数据在核心设备上进行本地操作。然后，即使核心设备与云的连接有限，它也可以对遥测数据做出反应。

例如，您可以配置一个组件，该组件监听`$local/greengrass/telemetry`主题中的遥测数据，并将数据发送到流管理器组件，以便将您的数据流式传输到。AWS Cloud有关创建此类组件的更多信息，请参见发布/订阅本地消息和创建使用流管理器的自定义组件。

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
1.0.8	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
1.0.7	Greengrass nucleus 版本 2.11.0 版本的版本已更新。

版本	更改
1.0.6	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
1.0.5	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
1.0.4	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
1.0.3	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
1.0.2	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
1.0.1	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
1.0.0	初始版本。

PKCS #11 提供商

PKCS #11 提供程序组件 (`aws.greengrass.crypto.Pkcs11Provider`) 使您可以通过 P KCS #11 接口将AWS IoT Greengrass核心软件配置为使用硬件安全模块 (HSM)。此组件使您能够安全地存储证书和私钥文件，这样它们就不会在软件中暴露或复制。有关更多信息，请参阅 硬件安全性集成。

要配置在 HSM 中存储其证书和私钥的 Greengrass 核心设备，您必须在安装 Core 软件时将此组件指定为配置插件。AWS IoT Greengrass有关更多信息，请参阅 使用手动资源配置来安装 AWS IoT Greengrass Core 软件。

AWS IoT Greengrass将此组件作为 JAR 文件提供，您可以下载该文件以在安装过程中将其指定为预配插件。您可以通过以下 URL 下载该组件 JAR 文件的最新版本：https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置

- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

- 一种硬件安全模块，支持 PKCS #1 v1.5 签名方案和密钥大小为 RSA-2048（或更大）或 ECC 密钥的 RSA 密钥。

> Note
>
> 要使用带有 ECC 密钥的硬件安全模块，必须使用 Greengrass nucleus v2.5.6 或更高版本。要使用硬件安全模块和密钥管理器，必须使用带有 RSA 密钥的硬件安全模块。

- 一个 PKCS #11 提供程序库，AWS IoT Greengrass核心软件可以在运行时加载该库（使用 libdl）来调用 PKCS #11 函数。PKCS #11 提供程序库必须实现以下 PKCS #11 API 操作：
  - `C_Initialize`
  - `C_Finalize`

  - C_GetSlotList
  - C_GetSlotInfo
  - C_GetTokenInfo
  - C_OpenSession
  - C_GetSessionInfo
  - C_CloseSession
  - C_Login
  - C_Logout
  - C_GetAttributeValue
  - C_FindObjectsInit
  - C_FindObjects
  - C_FindObjectsFinal
  - C_DecryptInit
  - C_Decrypt
  - C_DecryptUpdate
  - C_DecryptFinal
  - C_SignInit
  - C_Sign
  - C_SignUpdate
  - C_SignFinal
  - C_GetMechanismList
  - C_GetMechanismInfo
  - C_GetInfo
  - C_GetFunctionList
- 硬件模块必须可按槽标签解析，如 PKCS#11 规范所定义。
- 如果 HSM 支持对象 ID，则必须将私钥和证书存储在 HSM 的同一个插槽中，并且它们必须使用相同的对象标签和对象 ID。
- 证书和私钥必须可通过对象标签进行解析。
- 私钥必须具有以下权限：

  - sign

  - decrypt
- （可选）要使用密钥管理器组件，必须使用 2.1.0 或更高版本，并且私钥必须具有以下权限：
  - unwrap
  - wrap
- （可选）如果您使用 TPM2 库并将 Greengrass 核心作为服务运行，则必须提供一个包含 PKCS #11 存储位置的环境变量。以下示例是带有所需环境变量的 systemd 服务文件：

```
[Unit]
Description=Greengrass Core
After=network.target

[Service]
Type=simple
PIDFile=/var/run/greengrass.pid
Environment=TPM2_PKCS11_STORE=/path/to/store/directory
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

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

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.0.7

下表列出了此组件版本 2.0.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.3 <2.13.0	软性

2.0.6

下表列出了此组件版本 2.0.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.3 <2.12.0	软性

2.0.5

下表列出了此组件版本 2.0.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.3 <2.11.0	软性

2.0.4

下表列出了此组件版本 2.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.3 <2.10.0	软性

2.0.3

下表列出了此组件版本 2.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.3 <2.9.0	软性

2.0.2

下表列出了此组件版本 2.0.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.3 <2.8.0	软性

2.0.1

下表列出了此组件版本 2.0.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.3 <2.7.0	软性

2.0.0

下表列出了此组件版本 2.0.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.3 <2.6.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`name`

PKCS #11 配置的名称。

`library`

AWS IoT Greengrass核心软件可以使用 libdl 加载的 PKCS #11 实现库的绝对文件路径。

`slot`

包含私钥和设备证书的插槽的 ID。此值不同于插槽索引或插槽标签。

`userPin`

用于访问插槽的用户 PIN。

Example 示例：配置合并更新

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.0.7	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.0.6	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.0.5	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.0.4	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.0.3	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.0.2	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.0.1	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.0.0	初始版本。

秘密经理

密钥管理器组件 (`aws.greengrass.SecretManager`) 将机密部署AWS Secrets Manager到 Greengrass 核心设备。使用此组件可在 Greengrass 核心设备的自定义组件中安全地使用密码（如密码）。有关 Secrets Manager 的更多信息，请参阅 AWS Secrets Manager 用户指南中的什么是 AWS Secrets Manager？。

要在您的自定义 Greengrass 组件中访问此组件的秘密，请使用中的操作。GetSecretValueAWS IoT Device SDK有关更多信息，请参阅 使用AWS IoT Device SDK与 Greengrass 原子核、其他组件进行通信 AWS IoT Core和 检索秘密值。

此组件对核心设备上的机密进行加密，以确保您的凭据和密码的安全，直到您需要使用它们为止。它使用核心设备的私钥来加密和解密机密。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项

- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x
- 2.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- Greengrass 设备角色必须允许`secretsmanager:GetSecretValue`该操作，如以下示例 IAM 策略所示。

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

```
        "Action": [
          "secretsmanager:GetSecretValue"
        ],
        "Effect": "Allow",
        "Resource": [
          "arn:aws:secretsmanager:region:123456789012:secret:MySecret"
        ]
      }
    ]
  }
```

> **Note**
>
> 如果您使用客户管理的密AWS Key Management Service钥来加密机密，则设备角色也必须允许该kms:Decrypt操作。

有关 Secrets Manager 的 IAM 策略的更多信息，请参阅AWS Secrets Manager用户指南中的以下内容：

- 的身份验证和访问控制 AWS Secrets Manager
- 您可以在 IAM 策略或密钥策略中使用的操作、资源和上下文密钥 AWS Secrets Manager

- 自定义组件必须定义授权策略，aws.greengrass#GetSecretValue允许获取您存储在此组件中的机密。在此授权策略中，您可以限制组件对特定机密的访问权限。有关更多信息，请参阅密钥管理器 IPC 授权。
- （可选）如果您将核心设备的私钥和证书存储在硬件安全模块 (HSM) 中，则 HSM 必须支持 RSA 密钥，私钥必须具有unwrap权限，公钥必须具有权限。wrap

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
secretsmanager.*region*.amazonaws.com	443	是	将密钥下载到核心设备。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.13.0	软性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.12.0	软性

2.1.5

下表列出了此组件版本 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.11.0	软性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.10.0	软性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.9.0	软性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.8.0	软性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.7.0	软性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.6.0	软性

2.0.9

下表列出了此组件版本 2.0.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性

2.0.8

下表列出了此组件版本 2.0.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性

2.0.7

下表列出了此组件版本 2.0.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性

2.0.6

下表列出了此组件版本 2.0.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性

2.0.4 and 2.0.5

下表列出了此组件版本 2.0.4 和 2.0.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`cloudSecrets`

要部署到核心设备的 Secrets Manager 密钥列表。您可以指定标签来定义要部署的每个密钥的哪些版本。如果您未指定版本，则此组件将部署附有暂存标签`AWSCURRENT`的版本。有关更多信息，请参阅《AWS Secrets Manager用户指南》中的暂存标签。

密钥管理器组件在本地缓存机密。如果 Secrets Manager 中的密钥值发生变化，则此组件不会自动检索新值。要更新本地副本，请为密钥指定一个新标签，然后将此组件配置为检索由新标签标识的密钥。

每个对象都包含以下信息：

`arn`

要部署的秘密的 ARN。密钥的 ARN 可以是完整的 ARN，也可以是部分 ARN。我们建议您指定完整的 ARN，而不是部分 ARN。有关更多信息，请参阅从部分 ARN 中查找密钥。以下是完整 ARN 和部分 ARN 的示例：

- 完整的 ARN：`arn:aws:secretsmanager:us-east-2:111122223333:secret:`*`SecretName`*`-abcdef`
- 部分 ARN：`arn:aws:secretsmanager:us-east-2:111122223333:secret:`*`SecretName`*

`labels`

（可选）用于标识要部署到核心设备的密钥版本的标签列表。

每个标签必须是一个字符串。

Example 示例：配置合并更新

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-abcdef"
    }
  ]
}
```

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.7	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.6	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.5	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.4	错误修复和改进 修复了部署秘密管理器和 Greengrass 核心重启时缓存的机密被移除的问题。 Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.0	新功能 • 增加了对硬件安全集成的支持。密钥管理器组件可以使用存储在硬件安全模块 (HSM) 中的私钥来加密和解密机密。有关更多信息，请参阅 硬件安全性集成。 错误修复和改进 • Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.9	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.8	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.7	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.0.6	Greengrass nucleus 版本 2.1.0 版本的版本已更新。
2.0.5	改进 • 添加对AWS中国地区和AWS GovCloud (US)地区的支持。
2.0.4	初始版本。

安全隧道

使用该aws.greengrass.SecureTunneling组件，您可以与位于受限防火墙后面的 Greengrass 核心设备建立安全的双向通信。

例如，假设您在防火墙后面有一台 Greengrass 核心设备，该设备禁止所有传入连接。安全隧道使用 MQTT 将访问令牌传输到设备，然后使用 WebSockets 通过防火墙与设备建立 SSH 连接。使用此 AWS IoT托管隧道，您可以打开设备所需的 SSH 连接。有关使用AWS IoT安全隧道连接远程设备的更多信息，请参阅《开发人员指南》中的AWS IoT安全隧道。AWS IoT

此组件订阅$aws/things/*greengrass-core-device*/tunnels/notify主题上的 AWS IoT Core MQTT 消息代理以接收安全隧道通知。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 许可证
- 使用量
- 另请参阅
- 更改日志

版本

此组件有以下版本：

- 1.0.x

类型

此组件是一个通用组件 (aws.greengrass.generic)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

架构：

- armv71
- Armv8 (AArch64)
- x86_64

要求

此组件具有以下要求：

- 安全隧道组件至少有 32 MB 的可用磁盘空间。此要求不包括运行在同一设备上的 Greengrass 核心软件或其他组件。
- 安全隧道组件至少有 16 MB 的可用内存。此要求不包括运行在同一设备上的 Greengrass 核心软件或其他组件。有关更多信息，请参阅 使用 JVM 选项控制内存分配。
- 安全隧道组件版本 1.0.12 及更高版本需要 GNU C 库 (glibc) 版本 2.25 或更高版本，Linux 内核为 3.2 或更高版本。不支持超过长期支持期限的操作系统和库的版本。您应该使用具有长期支持的操作系统和库。
- 操作系统和 Java 运行时都必须安装为 64 位。
- Python 3.5 或更高版本安装在 Greengrass 核心设备上，并已添加到 PATH 环境变量中。
- `libcrypto.so.1.1`安装在 Greengrass 核心设备上并添加到 PATH 环境变量中。
- 在 Greengrass 核心设备的端口 443 上打开出站流量。
- 开启对要用来与 Greengrass 核心设备通信的通信服务的支持。例如，要打开与设备的 SSH 连接，必须在该设备上打开 SSH。

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
`data.tunneling.iot.region.amazonaws.com`	443	是	建立安全隧道。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

1.0.18

下表列出了此组件版本 1.0.18 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性

1.0.16 – 1.0.17

下表列出了此组件的 1.0.16 到 1.0.17 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性

1.0.14 – 1.0.15

下表列出了此组件的 1.0.14 到 1.0.15 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性

1.0.11 – 1.0.13

下表列出了此组件版本 1.0.11-1.0.13 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性

1.0.10

下表列出了此组件版本 1.0.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性

1.0.9

下表列出了此组件版本 1.0.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性

1.0.8

下表列出了此组件版本 1.0.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性

1.0.5 - 1.0.7

下表列出了此组件的 1.0.5 到 1.0.7 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性

1.0.4

下表列出了此组件版本 1.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性

1.0.3

下表列出了此组件版本 1.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性

1.0.2

下表列出了此组件版本 1.0.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性

1.0.1

下表列出了此组件版本 1.0.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性

1.0.0

下表列出了此组件版本 1.0.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`OS_DIST_INFO`

（可选）核心设备的操作系统。默认情况下，该组件会尝试自动识别核心设备上运行的操作系统。如果组件无法使用默认值启动，请使用此值来指定操作系统。有关此组件支持的操作系统的列表，请参见设备要求。

该值可以是以下值之一：`auto`、`ubuntu`、`amzn2`、`raspberrypi`。

默认值：`auto`

`accessControl`

（可选）包含授权策略的对象，该策略允许组件订阅安全隧道通知主题。

> **Note**
>
> 如果您的部署以事物组为目标，请勿修改此配置参数。如果您的部署以单个核心设备为目标，并且您想将其订阅限制为该设备的主题，请指定核心设备的事物名称。在设备授权策略的`resources`值中，将 MQTT 主题通配符替换为设备的事物名称。

```
{
  "aws.greengrass.ipc.mqttproxy": {
    "aws.iot.SecureTunneling:mqttproxy:1": {
      "policyDescription": "Access to tunnel notification pubsub topic",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
```

```
      ],
      "resources": [
        "$aws/things/+/tunnels/notify"
      ]
    }
  }
}
```

Example 示例：配置合并更新

以下示例配置指定允许此组件在运行 Ubuntu 的名为**MyGreengrassCore**的核心设备上打开安全隧道。

```
{
  "OS_DIST_INFO": "ubuntu",
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.iot.SecureTunneling:mqttproxy:1": {
        "policyDescription": "Access to tunnel notification pubsub topic",
        "operations": [
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "$aws/things/MyGreengrassCore/tunnels/notify"
        ]
      }
    }
  }
}
```

本地日志文件

此组件使用以下日志文件。

```
/greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。*/greengrass/v2*替换为AWS IoT Greengrass根文件夹的路径。

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

许可证

此组件包括以下第三方软件/许可：

- AWS IoT设备客户端 /Apache 许可证 2.0
- AWS IoT Device SDK for Java/Apache 许可证 2.0
- gson /Apache 许可证 2.0
- log4j /Apache 许可证 2.0
- slf4j /Apache 许可证 2.0

使用量

要在设备上使用安全隧道组件，请执行以下操作：

1. 将安全隧道组件部署到您的设备上。
2. 打开AWS IoT控制台。从左侧菜单中选择 “远程操作”，然后选择 “安全隧道”。
3. 创建一条通往你的 Greengrass 设备的隧道。
4. 下载源访问令牌。
5. 使用带有源访问令牌的本地代理连接到您的目的地。有关更多信息，请参阅《AWS IoT开发人员指南》中的如何使用本地代理。

另请参阅

- AWS IoT《开发人员指南》AWS IoT中的@@ 安全隧道
- 如何使用AWS IoT开发人员指南中的本地代理

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
1.0.18	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
1.0.17	错误修复和改进 • 修复了阻碍用户创建隧道的线程清理问题。现在，该组件将在线程接收到 CloseTunnel 信号后或隧道在 12 小时后过期时对其进行清理。
1.0.16	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2016-09-01	错误修复和改进 • 修复了设备上没有主目录的用户的启动问题。现在，安全隧道组件无需为影子文档创建目录即可启动。
1.0.14	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
1.0.13	错误修复和改进 • 修复了孤立客户端进程阻止多个隧道瞄准设备的问题。
1.0.12	错误修复和改进 • 在 Raspberry Pi 操作系统上运行时，增加了对 x86_64 (AMD64) 和 armv8 (Aarch64) 的支持。
2016-09-01	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2016-09-01	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
1.0.9	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
1.0.8	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
1.0.7	错误修复和改进 • 修复了通过 SCP 传输大文件时组件断开连接的问题。
1.0.6	此版本包含错误修复。
1.0.5	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
1.0.4	Greengrass nucleus 版本 2.4.0 版本的版本已更新。

版本	更改
1.0.3	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
1.0.2	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
1.0.1	Greengrass nucleus 版本 2.1.0 版本的版本已更新。
1.0.0	初始版本。

影子经理

影子管理器组件 (`aws.greengrass.ShadowManager`) 可在核心设备上启用本地影子服务。本地阴影服务允许组件使用进程间通信与本地阴影进行交互。影子管理器组件管理本地卷影文档的存储，还处理本地卷影状态与 Dev AWS IoT ice Shadow 服务的同步。

有关 Greengrass 核心设备如何与阴影交互的更多信息，请参阅。与设备阴影互动

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

类型

此组件是一个插件组件 (`aws.greengrass.plugin`)。Greengrass 核心在与核心相同的 Java 虚拟机 (JVM) 中运行此组件。当您在核心设备上更改此组件的版本时，nucleus 会重新启动。

该组件使用与 Greengrass 核相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- （可选）要将影子同步到 Device Shadow 服务，Greengrass 核心设备的策略必须允许以下影 AWS IoT 子策略操作： AWS IoT AWS IoT Core
  - `iot:GetThingShadow`
  - `iot:UpdateThingShadow`
  - `iot:DeleteThingShadow`

  有关这些 AWS IoT Core 政策的更多信息，请参阅《AWS IoT 开发人员指南》中的AWS IoT Core 策略操作。

  有关最低 AWS IoT 政策的更多信息，请参阅 AWS IoT Greengrass V2核心设备的最低AWS IoT政策
- 支持在 VPC 中运行影子管理器组件。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.3.5 – 2.3.7

下表列出了此组件版本 2.3.5 到 2.3.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.13.0	软性

2.3.3 and 2.3.4

下表列出了此组件版本 2.3.3 和 2.3.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.12.0	软性

2.3.2

下表列出了此组件版本 2.3.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.11.0	软性

2.3.0 and 2.3.1

下表列出了此组件版本 2.3.0 和 2.3.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.5.0 <2.10.0	软性

2.2.3 and 2.2.4

下表列出了此组件版本 2.2.3 和 2.2.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <3.0.0	软性

2.2.2

下表列出了此组件版本 2.2.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.9.0	软性

2.2.1

下表列出了此组件版本 2.2.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.8.0	软性

2.1.1 and 2.2.0

下表列出了此组件版本 2.1.1 和 2.2.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.7.0	软性

2.0.5 - 2.1.0

下表列出了此组件的 2.0.5 到 2.1.0 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.6.0	软性

2.0.3 and 2.0.4

下表列出了此组件版本 2.0.3 和 2.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.5.0	软性

2.0.1 and 2.0.2

下表列出了此组件版本 2.0.1 和 2.0.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.4.0	软性

2.0.0

下表列出了此组件版本 2.0.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.2.0 <2.3.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

2.3.x

`strategy`

（可选）此组件用于在 AWS IoT Core 和核心设备之间同步阴影的策略。

此对象包含以下信息。

`type`

（可选）此组件用于在 AWS IoT Core 和核心设备之间同步阴影的策略类型。从以下选项中进行选择：

- `realTime`— AWS IoT Core 每次阴影更新时都同步阴影。
- `periodic`— 按照您使用 AWS IoT Core `delay`配置参数指定的固定间隔将阴影与同步。

默认：`realTime`

`delay`

（可选）指定同步策略时，此组件与 AWS IoT Core之同步阴影的时间间隔（以秒为`periodic`单位）。

> **Note**
>
> 如果您指定`periodic`同步策略，则此参数是必需的。

`synchronize`

（可选）决定阴影如何与同步的同步设置。 AWS Cloud

> **Note**
>
> 必须使用此属性创建配置更新，才能将阴影与同步 AWS Cloud。

此对象包含以下信息。

`coreThing`

（可选）要同步的核心设备影子。此对象包含以下信息。

`classic`

（可选）默认情况下，影子管理器会将核心设备的经典影子的本地状态与同 AWS Cloud 步。如果您不想同步经典设备影子，请将其设置为`false`。

默认：`true`

`namedShadows`

（可选）要同步的已命名核心设备影子列表。必须指定阴影的确切名称。

Warning

该 AWS IoT Greengrass 服务使用`AWSManagedGreengrassV2Deployment`命名的影子来管理针对单个核心设备的部署。这个名为 shadow 的保留供 AWS IoT Greengrass 服务使用。请勿更新或删除这个名为 shadow 的影子。

`shadowDocumentsMap`

（可选）要同步的其他设备影子。使用此配置参数可以更轻松地指定影子文档。我们建议您使用此参数代替`shadowDocuments`对象。

Note

如果指定`shadowDocumentsMap`对象，则不得指定`shadowDocuments`对象。

每个对象都包含以下信息：

thingName

此影子配置的 t*hingName* 的影子配置。

`classic`

（可选）如果您不想同步设备的经典设备影子，请将其设置为`false`。`thingName`

`namedShadows`

要同步的已命名阴影列表。必须指定阴影的确切名称。

`shadowDocuments`

（可选）要同步的其他设备影子列表。我们建议您改用`shadowDocumentsMap`参数。

Note

如果指定`shadowDocuments`对象，则不得指定`shadowDocumentsMap`对象。

此列表中的每个对象都包含以下信息。

thingName

要同步阴影的设备的事物名称。

classic

（可选）如果您不想同步设备的经典设备影子，请将其设置为false。thingName

默认：true

namedShadows

（可选）您要同步的已命名设备影子列表。必须指定阴影的确切名称。

direction

（可选）在本地阴影服务与之间同步阴影的方向 AWS Cloud。您可以配置此选项以减少带宽和与的连接 AWS Cloud。从以下选项中进行选择：

- betweenDeviceAndCloud— 同步本地影子服务与之间的阴影 AWS Cloud。
- deviceToCloud— 将卷影更新从本地影子服务发送到 AWS Cloud，并忽略来自的影子更新 AWS Cloud。
- cloudToDevice— 接收来自的影子更新 AWS Cloud，不要将来自本地影子服务的影子更新发送到 AWS Cloud。

默认：BETWEEN_DEVICE_AND_CLOUD

rateLimits

（可选）确定影子服务请求速率限制的设置。

此对象包含以下信息。

maxOutboundSyncUpdatesPerSecond

（可选）设备每秒传输的最大同步请求数。

默认：100 个请求/秒

maxTotalLocalRequestsRate

（可选）每秒发送到核心设备的最大本地 IPC 请求数。

默认值：200 个请求/秒

maxLocalRequestsPerSecondPerThing

（可选）每秒为每个连接的物联网事物发送的最大本地 IPC 请求数。

默认：每件事每秒 20 个请求

Note

这些速率限制参数定义了本地影子服务每秒的最大请求数。Dev AWS IoT ice Shadow 服务每秒的最大请求数取决于您的 AWS 区域。有关更多信息，请参阅中的 Dev AWS IoT ice Shadow 服务 API 的限制Amazon Web Services 一般参考。

`shadowDocumentSizeLimitBytes`

（可选）每个 JSON 状态文档允许用于本地阴影的最大大小。

如果增加此值，则还必须增加云阴影的 JSON 状态文档的资源限制。有关更多信息，请参阅中的 Dev AWS IoT ice Shadow 服务 API 的限制Amazon Web Services 一般参考。

默认值：8192 字节

最大值：30720 字节

Example 示例：配置合并更新

以下示例显示了配置合并更新示例，其中包含影子管理器组件的所有可用配置参数。

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
```

```
            }
        },
        "direction":"betweenDeviceAndCloud"
    },
    "rateLimits":{
        "maxOutboundSyncUpdatesPerSecond":100,
        "maxTotalLocalRequestsRate":200,
        "maxLocalRequestsPerSecondPerThing":20
    },
    "shadowDocumentSizeLimitBytes":8192
}
```

2.2.x

`strategy`

（可选）此组件用于在 AWS IoT Core 和核心设备之间同步阴影的策略。

此对象包含以下信息。

`type`

（可选）此组件用于在 AWS IoT Core 和核心设备之间同步阴影的策略类型。从以下选项中进行选择：

- `realTime`— AWS IoT Core 每次阴影更新时都同步阴影。
- `periodic`— 按照您使用 AWS IoT Core `delay`配置参数指定的固定间隔将阴影与同步。

默认：`realTime`

`delay`

（可选）指定同步策略时，此组件与 AWS IoT Core之同步阴影的时间间隔（以秒为`periodic`单位）。

> **Note**
>
> 如果您指定`periodic`同步策略，则此参数是必需的。

`synchronize`

（可选）决定阴影如何与同步的同步设置。 AWS Cloud

Note

必须使用此属性创建配置更新，才能将阴影与同步 AWS Cloud。

此对象包含以下信息。

`coreThing`

（可选）要同步的核心设备影子。此对象包含以下信息。

`classic`

（可选）默认情况下，影子管理器会将核心设备的经典影子的本地状态与同 AWS Cloud 步。如果您不想同步经典设备影子，请将其设置为`false`。

默认：`true`

`namedShadows`

（可选）要同步的已命名核心设备影子列表。必须指定阴影的确切名称。

Warning

该 AWS IoT Greengrass 服务使用`AWSManagedGreengrassV2Deployment`命名的影子来管理针对单个核心设备的部署。这个名为 shadow 的保留供 AWS IoT Greengrass 服务使用。请勿更新或删除这个名为 shadow 的影子。

`shadowDocumentsMap`

（可选）要同步的其他设备影子。使用此配置参数可以更轻松地指定影子文档。我们建议您使用此参数代替`shadowDocuments`对象。

Note

如果指定`shadowDocumentsMap`对象，则不得指定`shadowDocuments`对象。

每个对象都包含以下信息：

`thingName`

此影子配置的 t*`hingName`* 的影子配置。

`classic`

（可选）如果您不想同步设备的经典设备影子，请将其设置为`false`。`thingName`

`namedShadows`

要同步的已命名阴影列表。必须指定阴影的确切名称。

`shadowDocuments`

（可选）要同步的其他设备影子列表。我们建议您改用`shadowDocumentsMap`参数。

> **Note**
>
> 如果指定`shadowDocuments`对象，则不得指定`shadowDocumentsMap`对象。

此列表中的每个对象都包含以下信息。

`thingName`

要同步阴影的设备的事物名称。

`classic`

（可选）如果您不想同步设备的经典设备影子，请将其设置为`false`。`thingName`

默认：`true`

`namedShadows`

（可选）您要同步的已命名设备影子列表。必须指定阴影的确切名称。

`direction`

（可选）在本地阴影服务与之间同步阴影的方向 AWS Cloud。您可以配置此选项以减少带宽和与的连接 AWS Cloud。从以下选项中进行选择：

- `betweenDeviceAndCloud`— 同步本地影子服务与之间的阴影 AWS Cloud。
- `deviceToCloud`— 将卷影更新从本地影子服务发送到 AWS Cloud，并忽略来自的影子更新 AWS Cloud。
- `cloudToDevice`— 接收来自的影子更新 AWS Cloud，不要将来自本地影子服务的影子更新发送到 AWS Cloud。

默认：`BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

（可选）确定影子服务请求速率限制的设置。

此对象包含以下信息。

`maxOutboundSyncUpdatesPerSecond`

（可选）设备每秒传输的最大同步请求数。

默认：100 个请求/秒

`maxTotalLocalRequestsRate`

（可选）每秒发送到核心设备的最大本地 IPC 请求数。

默认值：200 个请求/秒

`maxLocalRequestsPerSecondPerThing`

（可选）每秒为每个连接的物联网事物发送的最大本地 IPC 请求数。

默认：每件事每秒 20 个请求

> **Note**
>
> 这些速率限制参数定义了本地影子服务每秒的最大请求数。Dev AWS IoT ice Shadow 服务每秒的最大请求数取决于您的 AWS 区域。有关更多信息，请参阅中的 Dev AWS IoT ice Shadow 服务 API 的限制Amazon Web Services 一般参考。

`shadowDocumentSizeLimitBytes`

（可选）每个 JSON 状态文档允许用于本地阴影的最大大小。

如果增加此值，则还必须增加云阴影的 JSON 状态文档的资源限制。有关更多信息，请参阅中的 Dev AWS IoT ice Shadow 服务 API 的限制Amazon Web Services 一般参考。

默认值：8192 字节

最大值：30720 字节

Example 示例：配置合并更新

以下示例显示了配置合并更新示例，其中包含影子管理器组件的所有可用配置参数。

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.1.x

`strategy`

（可选）此组件用于在 AWS IoT Core 和核心设备之间同步阴影的策略。

此对象包含以下信息。

`type`

（可选）此组件用于在 AWS IoT Core 和核心设备之间同步阴影的策略类型。从以下选项中进行选择：

- `realTime`— AWS IoT Core 每次阴影更新时都同步阴影。
- `periodic`— 按照您使用 AWS IoT Core `delay`配置参数指定的固定间隔将阴影与同步。

默认：`realTime`

`delay`

（可选）指定同步策略时，此组件与 AWS IoT Core之同步阴影的时间间隔（以秒为`periodic`单位）。

> **Note**
>
> 如果您指定`periodic`同步策略，则此参数是必需的。

`synchronize`

（可选）决定阴影如何与同步的同步设置。 AWS Cloud

> **Note**
>
> 必须使用此属性创建配置更新，才能将阴影与同步 AWS Cloud。

此对象包含以下信息。

`coreThing`

（可选）要同步的核心设备影子。此对象包含以下信息。

`classic`

（可选）默认情况下，影子管理器会将核心设备的经典影子的本地状态与同 AWS Cloud 步。如果您不想同步经典设备影子，请将其设置为`false`。

默认：`true`

`namedShadows`

（可选）要同步的已命名核心设备影子列表。必须指定阴影的确切名称。

> **Warning**
>
> 该 AWS IoT Greengrass 服务使用`AWSManagedGreengrassV2Deployment`命名的影子来管理针对单个核心设备的部署。这个名为 shadow 的保留供 AWS IoT Greengrass 服务使用。请勿更新或删除这个名为 shadow 的影子。

`shadowDocumentsMap`

（可选）要同步的其他设备影子。使用此配置参数可以更轻松地指定影子文档。我们建议您使用此参数代替`shadowDocuments`对象。

Note

如果指定`shadowDocumentsMap`对象，则不得指定`shadowDocuments`对象。

每个对象都包含以下信息：

`thingName`

此影子配置的 t*`hingName`* 的影子配置。

`classic`

（可选）如果您不想同步设备的经典设备影子，请将其设置为`false`。`thingName`

`namedShadows`

要同步的已命名阴影列表。必须指定阴影的确切名称。

`shadowDocuments`

（可选）要同步的其他设备影子列表。我们建议您改用`shadowDocumentsMap`参数。

Note

如果指定`shadowDocuments`对象，则不得指定`shadowDocumentsMap`对象。

此列表中的每个对象都包含以下信息。

`thingName`

要同步阴影的设备的事物名称。

`classic`

（可选）如果您不想同步设备的经典设备影子，请将其设置为`false`。`thingName`

默认：`true`

namedShadows

（可选）要同步的已命名设备影子列表。必须指定阴影的确切名称。

rateLimits

（可选）确定影子服务请求速率限制的设置。

此对象包含以下信息。

maxOutboundSyncUpdatesPerSecond

（可选）设备每秒传输的最大同步请求数。

默认：100 个请求/秒

maxTotalLocalRequestsRate

（可选）每秒发送到核心设备的最大本地 IPC 请求数。

默认值：200 个请求/秒

maxLocalRequestsPerSecondPerThing

（可选）每秒为每个连接的物联网事物发送的最大本地 IPC 请求数。

默认：每件事每秒 20 个请求

> **Note**
>
> 这些速率限制参数定义了本地影子服务每秒的最大请求数。Dev AWS IoT ice Shadow 服务每秒的最大请求数取决于您的 AWS 区域。有关更多信息，请参阅中的 Dev AWS IoT ice Shadow 服务 API 的限制Amazon Web Services 一般参考。

shadowDocumentSizeLimitBytes

（可选）每个 JSON 状态文档允许用于本地阴影的最大大小。

如果增加此值，则还必须增加云阴影的 JSON 状态文档的资源限制。有关更多信息，请参阅中的 Dev AWS IoT ice Shadow 服务 API 的限制Amazon Web Services 一般参考。

默认值：8192 字节

最大值：30720 字节

Example 示例：配置合并更新

以下示例显示了配置合并更新示例，其中包含影子管理器组件的所有可用配置参数。

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.0.x

`synchronize`

（可选）决定阴影如何与同步的同步设置。 AWS Cloud

> **Note**
>
> 必须使用此属性创建配置更新，才能将阴影与同步 AWS Cloud。

此对象包含以下信息。

`coreThing`

（可选）要同步的核心设备影子。此对象包含以下信息。

`classic`

（可选）默认情况下，影子管理器会将核心设备的经典影子的本地状态与同 AWS Cloud 步。如果您不想同步经典设备影子，请将其设置为`false`。

默认：`true`

`namedShadows`

（可选）要同步的已命名核心设备影子列表。必须指定阴影的确切名称。

Warning

该 AWS IoT Greengrass 服务使用`AWSManagedGreengrassV2Deployment`命名的影子来管理针对单个核心设备的部署。这个名为 shadow 的保留供 AWS IoT Greengrass 服务使用。请勿更新或删除这个名为 shadow 的影子。

`shadowDocumentsMap`

（可选）要同步的其他设备影子。使用此配置参数可以更轻松地指定影子文档。我们建议您使用此参数代替`shadowDocuments`对象。

Note

如果指定`shadowDocumentsMap`对象，则不得指定`shadowDocuments`对象。

每个对象都包含以下信息：

`thingName`

此影子配置的 t*hingName* 的影子配置。

`classic`

（可选）如果您不想同步设备的经典设备影子，请将其设置为`false`。`thingName`

namedShadows

要同步的已命名阴影列表。必须指定阴影的确切名称。

shadowDocuments

（可选）要同步的其他设备影子列表。我们建议您改用shadowDocumentsMap参数。

> **Note**
> 如果指定shadowDocuments对象，则不得指定shadowDocumentsMap对象。

此列表中的每个对象都包含以下信息。

thingName

要同步阴影的设备的事物名称。

classic

（可选）如果您不想同步设备的经典设备影子，请将其设置为false。thingName

默认：true

namedShadows

（可选）要同步的已命名设备影子列表。必须指定阴影的确切名称。

rateLimits

（可选）确定影子服务请求速率限制的设置。

此对象包含以下信息。

maxOutboundSyncUpdatesPerSecond

（可选）设备每秒传输的最大同步请求数。

默认：100 个请求/秒

maxTotalLocalRequestsRate

（可选）每秒发送到核心设备的最大本地 IPC 请求数。

默认值：200 个请求/秒

`maxLocalRequestsPerSecondPerThing`

（可选）每秒为每个连接的物联网事物发送的最大本地 IPC 请求数。

默认：每件事每秒 20 个请求

> **Note**
>
> 这些速率限制参数定义了本地影子服务每秒的最大请求数。Dev AWS IoT ice Shadow 服务每秒的最大请求数取决于您的 AWS 区域。有关更多信息，请参阅中的 Dev AWS IoT ice Shadow 服务 API 的限制Amazon Web Services 一般参考。

`shadowDocumentSizeLimitBytes`

（可选）每个 JSON 状态文档允许用于本地阴影的最大大小。

如果增加此值，则还必须增加云阴影的 JSON 状态文档的资源限制。有关更多信息，请参阅中的 Dev AWS IoT ice Shadow 服务 API 的限制Amazon Web Services 一般参考。

默认值：8192 字节

最大值：30720 字节

Example 示例：配置合并更新

以下示例显示了配置合并更新示例，其中包含影子管理器组件的所有可用配置参数。

```
{
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadowA",
        "MyCoreShadowB"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
```

```
          "namedShadows": [
            "MyShadowA",
            "MyShadowB"
          ]
        },
        {
          "thingName": "MyDevice2",
          "classic": true,
          "namedShadows": []
        }
      ]
    },
    "rateLimits": {
      "maxOutboundSyncUpdatesPerSecond": 100,
      "maxTotalLocalRequestsRate": 200,
      "maxLocalRequestsPerSecondPerThing": 20
    },
    "shadowDocumentSizeLimitBytes": 8192
}
```

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.3.7	错误修复和改进 • 修复了影子管理器在影子管理器同步期间定期记录NullPointerException 错误的问题。
2.3.6	错误修复和改进 • 修复了设备离线时通过 AWS Cloud 更新删除的阴影属性在重新连接后继续存在于本地阴影中的问题。
2.3.5	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.3.4	错误修复和改进 • 增加了对空和空阴影状态文档的支持。
2.3.3	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.3.2	错误修复和改进 • 修复了本地影子数据库损坏时影子管理器进入BROKEN状态的问题。 • Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.3.1	错误修复和改进 • 修复了可能导致云影更新无法同步的情况。 • 修复了对命名阴影同步配置的更改仅适用于一个命名阴影的问题。
2.3.0	错误修复和改进 • 修复了 Greengrass 设备私钥存储在硬件安全模块中时，阴影可能无法同步的问题。

版本	更改
2.2.4	错误修复和改进 • 修复了更新本地影子文档时阴影大小验证与云不一致的问题。 • 修复了部署在配置节点`RESET`上执行时，影子管理器会停止监听配置更新的问题。
2.2.3	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.2.2	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.2.1	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.2.0	新功能 • 通过本地发布/订阅界面添加对本地影子服务的支持。现在，您可以就影子 MQTT 主题与本地发布/订阅消息代理进行通信，以获取、更新和删除核心设备上的阴影。此功能允许您使用 MQTT 网桥在客户端设备和本地发布/订阅接口之间中继有关影子主题的消息，从而将客户端设备连接到本地影子服务。 此功能需要 Greengrass nucleus 组件的 v2.6.0 或更高版本。要将客户端设备连接到本地影子服务，还必须使用 V2.2.0 或更高版本的 MQTT 桥接组件。 • 添加了可以配置为自定义方向的`direction` 选项，以便在本地阴影服务与之间同步阴影 AWS Cloud。您可以配置此选项以减少带宽和与的连接 AWS Cloud。
2.1.1	错误修复和改进 • 修复了 JSON 设备影子状态文档`desired`和`reported`部分的最大深度为 4 级而不是 5 级的问题。 • Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.0	新功能 • 增加了对定期阴影同步间隔的支持，因此您可以配置核心设备以减少带宽使用量和费用。
2.0.6	此版本包含错误修复和改进。

版本	更改
2.0.5	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.4	错误修复和改进 • 修复了导致影子管理器删除之前删除的所有阴影的新创建版本的问题。 • 更新 `DeleteThingShadow` IPC 操作以在调用时递增影子版本。
2.0.3	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.2	错误修复和改进 • 修复了在同步来自 AWS IoT Core的阴影状态时导致影子管理器无法识别该`delta`属性的问题。 • 修复了有时会导致影子同步请求合并错误的问题。
2.0.1	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.0	初始版本。

Amazon SNS

亚马逊 SNS 组件 (`aws.greengrass.SNS`) 向亚马逊简单通知服务 (Amazon SNS) Simple Notification Service 主题发布消息。您可以使用此组件将事件从 Greengrass 核心设备发送到 Web 服务器、电子邮件地址和其他消息订阅者。有关更多信息，请参阅《Amazon Simple Notification Service 开发者指南》中的什么是 Amazon SNS？。

要使用此组件发布到 Amazon SNS 主题，请向该组件订阅的主题发布一条消息。默认情况下，此组件订阅`sns/message`本地发布/ 订阅主题。部署此组件时，您可以指定其他主题，包括 AWS IoT Core MQTT 主题。

在您的自定义组件中，您可能需要实现筛选或格式化逻辑来处理来自其他来源的消息，然后再将其发布到此组件。这使您能够将消息处理逻辑集中在单个组件上。

> **Note**
>
> 此组件提供的功能与 V1 中的 Amazon SNS 连接器类似。 AWS IoT Greengrass 有关更多信息，请参阅《AWS IoT Greengrass V1 开发者指南》中的 Amazon SNS 连接器。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 输入数据
- 输出数据
- 本地日志文件
- 许可证
- 更改日志

版本

此组件有以下版本：

- 2.1.x
- 2.0.x

类型

此组件是一个 Lambda 组件 () `aws.greengrass.lambda`。Greengrass 核心使用 Lambda 启动器组件运行此组件的 Lambda 函数。

有关更多信息，请参阅组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

- 您的核心设备必须满足运行 Lambda 函数的要求。如果您希望核心设备运行容器化的 Lambda 函数，则该设备必须满足要求。有关更多信息，请参阅Lambda 函数要求。
- Python 版本 3.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- Amazon SNS 主题。有关更多信息，请参阅 Amazon Simple Notification Service 开发人员指南中的创建 Amazon SNS 主题。
- Greengrass 设备角色必须允许`sns:Publish`该操作，如以下示例 IAM 策略所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:region:account-id:topic-name"
      ]
    }
  ]
}
```

  您可以动态覆盖此组件的输入消息负载中的默认主题。如果您的应用程序使用此功能，则 IAM 策略必须包含所有目标主题作为资源。您可以授予对资源的具体或条件访问权限（例如，通过使用通配符 * 命名方案）。

- 要接收此组件的输出数据，在部署此组件时，必须合并旧版订阅路由器组件 (`aws.greengrass.LegacySubscriptionRouter`) 的以下配置更新。此配置指定此组件发布响应的主题。

  Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-sns": {
      "id": "aws-greengrass-sns",
      "source": "component:aws.greengrass.SNS",
      "subject": "sns/message/status",
      "target": "cloud"
    }
  }
```

```
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-sns": {
      "id": "aws-greengrass-sns",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-sns:version",
      "subject": "sns/message/status",
      "target": "cloud"
    }
  }
}
```

- 将*##*替换为您 AWS 区域 使用的区域。
- 将*##*替换为该组件运行的 Lambda 函数的版本。要查找 Lambda 函数版本，您必须查看要部署的此组件版本的配方。在AWS IoT Greengrass 控制台中打开此组件的详细信息页面，然后查找 Lambda 函数键值对。此键值对包含 Lambda 函数的名称和版本。

> **Important**
>
> 每次部署此组件时，都必须更新旧版订阅路由器上的 Lambda 函数版本。这样可以确保您为部署的组件版本使用正确的 Lambda 函数版本。

有关更多信息，请参阅创建部署。

- 支持在 VPC 中运行 Amazon SNS 组件。要在 VPC 中部署此组件，需要满足以下条件。
  - Amazon SNS 组件必须连接到 `sns.region.amazonaws.com` VPC 终端节点为。`com.amazonaws.us-east-1.sns`

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
`sns.region.amazonaws.com`	443	是	向 Amazon SNS 发布消息。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.7

下表列出了此组件版本 2.1.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.6

下表列出了此组件版本 2.1.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性

依赖关系	兼容版本	依赖关系类型
代币兑换服务	^2.0.0	硬性

2.1.5

下表列出了此组件版本 2.1.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.4

下表列出了此组件版本 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.3

下表列出了此组件版本 2.1.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.2

下表列出了此组件版本 2.1.2 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.8 - 2.1.0

下表列出了此组件版本 2.0.8 和 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.7

下表列出了此组件版本 2.0.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.6

下表列出了此组件版本 2.0.6 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性

依赖关系	兼容版本	依赖关系类型
代币兑换服务	^2.0.0	硬性

2.0.5

下表列出了此组件版本 2.0.5 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.4

下表列出了此组件版本 2.0.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	硬性
Lambda 启动器	^2.0.0	硬性
Lambda 运行时	^2.0.0	软性
代币兑换服务	^2.0.0	硬性

2.0.3

下表列出了此组件版本 2.0.3 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	硬性
Lambda 启动器	>=1.0.0	硬性
Lambda 运行时	>=1.0.0	软性
代币兑换服务	>=1.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

Note

此组件的默认配置包括 Lambda 函数参数。我们建议您仅编辑以下参数，以便在您的设备上配置此组件。

`lambdaParams`

包含此组件的 Lambda 函数参数的对象。该对象包含以下信息：

`EnvironmentVariables`

一个包含 Lambda 函数参数的对象。该对象包含以下信息：

`DEFAULT_SNS_ARN`

此组件发布消息的默认 Amazon SNS 主题的 ARN。您可以使用输入消息负载中的`sns_topic_arn`属性覆盖目标主题。

`containerMode`

（可选）此组件的容器化模式。从以下选项中进行选择：

- `NoContainer`— 该组件不在隔离的运行时环境中运行。
- `GreengrassContainer`— 该组件在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

默认：GreengrassContainer

containerParams

（可选）包含此组件容器参数的对象。如果您为指定GreengrassContainer，则该组件将使用这些参数containerMode。

该对象包含以下信息：

memorySize

（可选）要分配给组件的内存量（以千字节为单位）。

默认为 512 MB (525,312 KB)。

pubsubTopics

（可选）一个对象，其中包含组件订阅以接收消息的主题。您可以指定每个主题以及该组件是订阅来自的 MQTT 主题 AWS IoT Core 还是本地发布/订阅主题。

该对象包含以下信息：

0— 这是字符串形式的数组索引。

包含以下信息的对象：

type

（可选）此组件用于订阅消息的发布/订阅消息的类型。从以下选项中进行选择：

- PUB_SUB – 订阅本地发布/订阅消息。如果选择此选项，则主题不能包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅本地消息。
- IOT_CORE— 订阅 AWS IoT Core MQTT 消息。如果选择此选项，则主题可以包含 MQTT 通配符。有关在指定此选项时如何从自定义组件发送消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

默认：PUB_SUB

topic

（可选）组件订阅以接收消息的主题。如果您IotCore为指定type，则可以在本主题中使用 MQTT 通配符（+和#）。

Example 示例：配置合并更新（容器模式）

```
{
```

```
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example 示例：配置合并更新（无容器模式）

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "NoContainer"
}
```

输入数据

此组件接受有关以下主题的消息，并将消息按原样发布到目标 Amazon SNS 主题。默认情况下，此组件订阅本地发布/订阅消息。有关如何从您的自定义组件向该组件发布消息的更多信息，请参阅发布/订阅本地消息。

默认主题（本地发布/订阅）：`sns/message`

该消息接受以下属性。输入消息必须采用 JSON 格式。

`request`

有关要发送至 Amazon SNS 主题的消息的信息。

类型：其中`object`包含以下信息：

`message`

以字符串形式显示的消息内容。

要发送 JSON 对象，请将其序列化为字符串，然后`json`为该`message_structure`属性指定。

类型：`string`

`subject`

（可选）消息的主题。

类型：`string`

主题可以是 ASCII 文本，最多 100 个字符。它必须以字母、数字或标点符号开头。它不能包含换行符或控制字符。

`sns_topic_arn`

（可选）此组件发布消息的 Amazon SNS 主题的 ARN。指定此属性可覆盖默认的 Amazon SNS 主题。

类型：`string`

`message_structure`

（可选）消息的结构。指定`json`在`content`属性中发送您序列化为字符串的 JSON 消息。

类型：`string`

有效值：`json`

`id`

请求的任意 ID。使用此属性将输入请求映射到输出响应。当您指定此属性时，组件会将响应对象中的`id`属性设置为该值。

类型：`string`

> **Note**
>
> 邮件大小最大可以为 256 KB。

Example 示例输入：字符串消息

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
```

```
  },
  "id": "request123"
}
```

Example 示例输入：JSON 消息

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
    "message_structure": "json"
  },
  "id": "request123"
}
```

输出数据

默认情况下，此组件将响应作为以下 MQTT 主题的输出数据发布。您必须在传统订阅路由器组件的配置`subject`中将此主题指定为。有关如何在您的自定义组件中订阅有关此主题的消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

默认主题 (AWS IoT Core MQTT)：`sns/message/status`

Example 示例输出：成功

```
{
  "response": {
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
    "status": "success"
  },
  "id": "request123"
}
```

Example 示例输出：失败

```
{
  "response" : {
    "error": "InvalidInputException",
    "error_message": "SNS Topic Arn is invalid",
    "status": "fail"
  },
  "id": "request123"
```

```
}
```

本地日志文件

此组件使用以下日志文件。

```
/greengrass/v2/logs/aws.greengrass.SNS.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。*/greengrass/v2*替换为 AWS IoT Greengrass 根文件夹的路径。

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SNS.log
  ```

许可证

此组件包括以下第三方软件/许可：

- AWS SDK for Python (Boto3)/Apache 许可证 2.0
- botocore/Apache 许可证 2.0
- dateutil/PSF 许可证
- docutils/BSD 许可证，GNU 通用公共许可证 (GPL)，Python 软件基金会许可证，公共领域
- jmespath/MIT 许可证
- s3transfer/Apache 许可证 2.0
- urllib3/MIT 许可证

该组件是根据 Greengrass 核心软件许可协议发布的。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.7	Greengrass nucleus 版本 2.12.0 版本的版本已更新。

版本	更改
2.1.6	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.5	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.4	Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.3	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.2	Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.6.0 版本的版本已更新。
2.1.0	新功能 • 添加对 HTTPS 网络代理配置的支持。有关更多信息，请参阅 通过端口 443 或网络代理进行连接 和 使核心设备能够信任 HTTPS 代理。
2.0.8	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.7	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.6	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.5	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.0.4	Greengrass nucleus 版本 2.1.0 版本的版本已更新。
2.0.3	初始版本。

流管理器

流管理器组件 (`aws.greengrass.StreamManager`) 使您能够处理要 AWS Cloud 从 Greengrass 核心设备传输到的数据流。

有关如何在自定义组件中配置和使用流管理器的更多信息，请参阅管理 Greengrass 核心设备上的数据流。

主题

- 版本

- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.1.x
- 2.0.x

> **Note**
>
> 如果您使用流管理器将数据导出到云端，则无法将流管理器组件的 2.0.7 版本升级到 v2.0.8 和 v2.0.11 之间的版本。如果您是首次部署流管理器，我们强烈建议您部署最新版本的流管理器组件。

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- 令牌交换角色必须允许访问您在直播管理器中使用的 AWS Cloud 目的地。有关更多信息，请参阅：
  - the section called "AWS IoT Analytics 通道"
  - the section called "Amazon Kinesis data streams"
  - the section called "AWS IoT SiteWise 资产属性"
  - the section called "Amazon S3 对象"
- 支持在 VPC 中运行流管理器组件。要在 VPC 中部署此组件，需要满足以下条件。
  - 流管理器组件必须连接到您向其发布数据的 AWS 服务。
    - 亚马逊 S3：`com.amazonaws.region.s3`
    - 亚马逊 Kinesis Data Streams：`com.amazonaws.region.kinesis-streams`
    - AWS IoT SiteWise: `com.amazonaws.region.iotsitewise.data`
  - 如果您将数据发布到`us-east-1`该区域的 Amazon S3，则默认情况下，此组件将尝试使用 S3 全局终端节点；但是，此终端节点无法通过 Amazon S3 VPC 接口终端节点使用。有关更多信息，请参阅 Amazon S3 AWS PrivateLink 的限制和限制。要解决此问题，您可以从以下选项中进行选择。
    - 通过提供`AWS_S3_US_EAST_1_REGIONAL_ENDPOINT=regional`环境变量，将流管理器组件配置为使用该`us-east-1`区域中的区域 S3 端点。
    - 创建 Amazon S3 网关 VPC 终端节点，而不是亚马逊 S3 接口 VPC 终端节点。S3 网关终端节点支持访问 S3 全局终端节点。有关更多信息，请参阅创建网关终端节点。

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
`iotanalytics. region.amazonaws.com`	443	否	如果您向发布数据，则为必填项 AWS IoT Analytics。

Endpoint	端口	必需	描述
kinesis.*region*.amazonaws.com	443	否	如果您向 Firehose 发布数据，则为必填项。
data.iotsitewise. *region*.amazonaws.com	443	否	如果您向发布数据，则为必填项 AWS IoT SiteWise。
*.s3.amazonaws.com	443	否	如果您将数据发布到 S3 存储桶，则为必填项。 您可以将*替换为发布数据的每个存储桶的名称。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

2.1.11

下表列出了此组件版本 2.1.11 到 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.13.0	软性
代币兑换服务	>=0.0.0	硬性

2.1.9 – 2.1.10

下表列出了此组件版本 2.1.9 到 2.1.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.12.0	软性
代币兑换服务	>=0.0.0	硬性

2.1.5 – 2.1.8

下表列出了此组件版本 2.1.5 到 2.1.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.11.0	软性
代币兑换服务	>=0.0.0	硬性

2.1.2 – 2.1.4

下表列出了此组件版本 2.1.2 到 2.1.4 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.10.0	软性
代币兑换服务	>=0.0.0	硬性

2.1.1

下表列出了此组件版本 2.1.1 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.9.0	软性
代币兑换服务	>=0.0.0	硬性

2.1.0

下表列出了此组件版本 2.1.0 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.8.0	软性
代币兑换服务	>=0.0.0	硬性

2.0.15

下表列出了此组件版本 2.0.15 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.7.0	软性
代币兑换服务	>=0.0.0	硬性

2.0.13 and 2.0.14

下表列出了此组件版本 2.0.13 和 2.0.14 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.6.0	软性

依赖关系	兼容版本	依赖关系类型
代币兑换服务	>=0.0.0	硬性

2.0.11 and 2.0.12

下表列出了此组件版本 2.0.11 和 2.0.12 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.5.0	软性
代币兑换服务	>=0.0.0	硬性

2.0.10

下表列出了此组件版本 2.0.10 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.4.0	软性
代币兑换服务	>=0.0.0	硬性

2.0.9

下表列出了此组件版本 2.0.9 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.3.0	软性
代币兑换服务	>=0.0.0	硬性

2.0.8

下表列出了此组件版本 2.0.8 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.0 <2.2.0	软性
代币兑换服务	>=0.0.0	硬性

2.0.7

下表列出了此组件版本 2.0.7 的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.0.3 <2.1.0	软性
代币兑换服务	>=0.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`STREAM_MANAGER_STORE_ROOT_DIR`

（可选）用于存储直播的本地目录的绝对路径。此值必须以正斜杠开头（例如，`/data`）。

您必须指定现有文件夹，并且运行流管理器组件的系统用户必须具有读取和写入该文件夹的权限。例如，您可以运行以下命令来创建和配置文件夹`/var/greengrass/streams`，将其指定为流管理器根文件夹。这些命令允许默认系统用户读取和写入此文件夹。`ggc_user`

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

默认：*/greengrass/v2*`/work/aws.greengrass.StreamManager`

`STREAM_MANAGER_SERVER_PORT`

（可选）用于与流管理器通信的本地端口号。

您可以指定`0`使用随机可用端口。

默认：`8088`

`STREAM_MANAGER_AUTHENTICATE_CLIENT`

（可选）您可以强制要求客户端在与直播管理器交互之前进行身份验证。直播管理器 SDK 控制客户端和直播管理器之间的交互。此参数决定哪些客户端可以调用 Stream Manager SDK 来处理直播。有关更多信息，请参阅直播管理器客户端身份验证。

如果您指定`true`，则流管理器 SDK 仅允许 Greengrass 组件作为客户端。

如果您指定`false`，Stream Manager SDK 允许核心设备上的所有进程成为客户端。

默认：`true`

`STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

（可选）流管理器可用于导出数据的平均最大带宽（以千比特每秒为单位）。

默认值：无限制

`STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

（可选）流管理器可用于导出数据的最大活动线程数。

最佳大小取决于您的硬件、流的量和计划的导出流数量。如果导出速度较慢，您可以调整此设置以找出适合您的硬件和业务案例的最佳大小。核心设备硬件的 CPU 和内存是限制因素。首先，您可以尝试将此值设置为等于设备上的处理器核心数。

请注意，不要设置大于硬件可以支持的大小。每个流都消耗硬件资源，因此请尽量限制受限设备上的导出流数量。

默认：5 个线程

`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

（可选）在 Amazon S3 的分段上传中分段的最小大小（以字节为单位）。流管理器使用此设置和输入文件的大小来确定如何对多部分 PUT 请求中的数据进行批处理。

> **Note**
>
> 流管理器使用 streams `sizeThresholdForMultipartUploadBytes` 属性来确定是以单段上传还是分段上传的形式导出到 Amazon S3。 AWS IoT Greengrass 组件可以在创建导出到 Amazon S3 的流时设置此阈值。

默认值：5242880(5 MB)。这也是最小值。

LOG_LEVEL

（可选）组件的日志级别。从以下日志级别中进行选择，此处按级别顺序列出：

- TRACE
- DEBUG
- INFO
- WARN
- ERROR

默认：INFO

JVM_ARGS

（可选）启动时要传递给流管理器的自定义 Java 虚拟机参数。用空格分隔多个参数。

仅当您必须覆盖 JVM 使用的默认设置时才使用此参数。例如，如果计划导出大量的流，则可能需要增加默认堆大小。

Example 示例：配置合并更新

以下示例配置指定使用非默认端口。

```
{
  "STREAM_MANAGER_SERVER_PORT": "18088"
}
```

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.greengrass.StreamManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.StreamManager.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.StreamManager.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.StreamManager.log -Tail 10 -
  Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.1.12	错误修复和改进 更新凭据的使用顺序，以便服务请求的首选 Greengrass 凭据。 AWS
2.1.11	Greengrass nucleus 版本 2.12.0 版本的版本已更新。
2.1.10	错误修复和改进 修复了 HTTPS 代理配置不信任 Greengrass 证书颁发机构 (CA) 证书链的问题。
2.1.9	Greengrass nucleus 版本 2.11.0 版本的版本已更新。
2.1.8	错误修复和改进 修复了流管理器无限重试 SiteWise 导出失败的问题。`InvalidRequestException`

版本	更改
2.1.7	错误修复和改进 修复了直播管理器无法正确读取代理配置的问题。
2.1.6	错误修复和改进 修复了可能导致某些 ARMv8 处理器（包括 Jetson Nano）在启动时崩溃的问题。
2.1.5	Greengrass nucleus 版本 2.10.0 版本的版本已更新。
2.1.4	错误修复和改进 • 修复了在单个批量中具有相同时间戳的相同属性资产的条目`ConflictingOperationException` 从 SiteWise API 返回的问题，该问题会导致流管理器不断重试。 • 将默认连接超时从 3 秒更新为 1 分钟。
2.1.3	错误修复和改进 修复了以 SYSTEM 用户身份运行时在 Windows 操作系统上出现的启动问题。
2.1.2	错误修复和改进 • 修复了 Windows 操作系统上使用非英语语言的问题。 • Greengrass nucleus 版本 2.9.0 版本的版本已更新。
2.1.1	Greengrass nucleus 版本 2.8.0 版本的版本已更新。
2.1.0	新功能 • 更新此组件以自动向 Amazon EventBridge 发送遥测指标。有关更多信息，请参阅从AWS IoT Greengrass核心设备收集系统运行状况遥测数据。 此功能需要 Greengrass nucleus 组件的 v2.7.0 或更高版本。 • Greengrass nucleus 版本 2.7.0 版本的版本已更新。
2.0.15	Greengrass nucleus 版本 2.6.0 版本的版本已更新。

版本	更改
2.0.14	此版本包含错误修复和改进。
2.0.13	Greengrass nucleus 版本 2.5.0 版本的版本已更新。
2.0.12	错误修复和改进 修复了无法将直播管理器 v2.0.7 升级到 v2.0.8 和 v2.0.11 之间的版本的问题。如果您使用流管理器将数据导出到云端，则现在可以升级到 v2.0.12。
2.0.11	Greengrass nucleus 版本 2.4.0 版本的版本已更新。
2.0.10	Greengrass nucleus 版本 2.3.0 版本的版本已更新。
2.0.9	Greengrass nucleus 版本 2.2.0 版本的版本已更新。
2.0.8	Greengrass nucleus 版本 2.1.0 版本的版本已更新。
2.0.7	初始版本。

Systems Manager 代理

AWS Systems Manager 代理组件 (`aws.greengrass.SystemsManagerAgent`) 会安装 Systems Manager 代理，因此您可以使用 Systems Manager 管理核心设备。Systems Manager 是一项可用于查看和控制基础设施的 AWS 服务 AWS，包括 Amazon EC2 实例、本地服务器和虚拟机 (VM) 以及边缘设备。Systems Manager 使您能够查看操作数据、自动执行操作任务以及维护安全性和合规性。有关更多信息，请参阅什么是 AWS Systems Manager？ 以及《AWS Systems Manager 用户指南》中的 “关于 Systems Manager 代理”。

Systems Manager 的工具和功能称为功能。Greengrass 核心设备支持 Systems Manager 的所有功能。有关这些功能以及如何使用 Systems Manager 管理核心设备的更多信息，请参阅《AWS Systems Manager 用户指南》中的 Systems Manager 功能。

主题

- 版本
- 类型
- 操作系统

- 要求
- 依赖项
- 配置
- 本地日志文件
- 另请参阅
- 更改日志

版本

此组件有以下版本：

- 1.1.x
- 1.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件只能安装在 Linux 核心设备上。

要求

此组件具有以下要求：

- 一款在 64 位 Linux 平台上运行的 Greengrass 核心设备：Armv8 (aarch64) 或 x86_64。
- 您必须拥有 Systems Manager 可以担任的 AWS Identity and Access Management (IAM) 服务角色。此角色必须包含 AmazonSSM ManagedInstanceCore 托管策略或定义等效权限的自定义策略。有关更多信息，请参阅AWS Systems Manager 用户指南中的为边缘设备创建 IAM 服务角色。

  部署此组件时，必须为`SSMRegistrationRole`配置参数指定此角色的名称。
- Greengrass 设备角色必须允许和操作。`ssm:AddTagsToResource` `ssm:RegisterManagedInstance`设备角色还必须允许满足先前要求的 IAM 服务角色执行`iam:PassRole`操作。以下示例 IAM 策略授予这些权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
ec2messages. *region*.amazonaws.com	443	支持	与中的 Systems Manager 服务进行通信 AWS Cloud。
ssm.*region*.amazonaws.com	443	支持	将核心设备注册为

Endpoint	端口	必需	描述
			Systems Manager 托管节点。
ssmmessages. *region*.amazonaws.com	443	支持	在中与会话管理器（Systems Manager 的一项功能）进行通信 AWS Cloud。

有关更多信息，请参阅《用户指南》中的 “参考：ec2messages、ssmessages 和其他 API 调用”。AWS Systems Manager

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

下表列出了此组件的 1.0.0 到 1.2.4 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
代币兑换服务	^2.0.0	软性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件提供以下配置参数，您可以在部署该组件时对其进行自定义。

`SSMRegistrationRole`

Systems Manager 可以担任的 IAM 服务角色，其中包括 AmazonSSM ManagedInstanceCore 托管策略或定义等效权限的自定义策略。有关更多信息，请参阅AWS Systems Manager 用户指南中的为边缘设备创建 IAM 服务角色。

`SSMOverrideExistingRegistration`

（可选）如果核心设备已经运行通过混合激活注册的 Systems Manager 代理，则可以覆盖该设备的现有 Systems Manager 代理注册。将此选项设置为，`true`使用此组件提供的 Systems Manager 代理将核心设备注册为托管节点。

> **Note**
>
> 此选项仅适用于通过混合激活注册的设备。如果核心设备在安装了 Systems Manager 代理并配置了实例配置文件角色的 Amazon EC2 实例上运行，则 Amazon EC2 实例的现有托管节点 ID 以开头`i-`。安装 Systems Manager 代理组件时，Systems Manager 代理会注册一个新的托管节点，该节点的 ID 以`mi-`而不是开头`i-`。然后，您可以使用 ID `mi-` 以开头的托管节点通过 Systems Manager 管理核心设备。

默认：`false`

`SSMResourceTags`

（可选）要添加到此组件为核心设备创建的 Systems Manager 托管节点的标签。您可以使用这些标签通过 Systems Manager 管理一组核心设备。例如，您可以在所有带有您指定标签的设备上运行命令。

指定一个列表，其中每个标签都是一个带有`Key`和的对象`Value`。例如，以下值`SSMResourceTags`指示此组件在核心设备的托管节点**`richard-roe`**上将**`Owner`**标签设置为。

```
[
  {
    "Key": "Owner",
    "Value": "richard-roe"
  }
]
```

如果托管节点已经存在并且存在，`SSMOverrideExistingRegistration`则`false`此组件将忽略这些标签。

Example 示例：配置合并更新

以下示例配置指定使用名为的服务角色`SSMServiceRole`以允许核心设备注册并与 Systems Manager 通信。

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false,
  "SSMResourceTags": [
    {
      "Key": "Owner",
      "Value": "richard-roe"
    },
    {
      "Key": "Team",
      "Value": "solar"
    }
  ]
}
```

本地日志文件

Systems Manager 代理软件将日志写入 Greengrass 根文件夹之外的文件夹。有关更多信息，请参阅《AWS Systems Manager 用户指南》中的 “查看 Systems Manager 代理日志”。

Systems Manager 代理组件使用 shell 脚本安装、启动和停止 Systems Manager 代理。您可以在以下日志文件中找到这些脚本的输出。

```
/greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。*/greengrass/v2*替换为 AWS IoT Greengrass 根文件夹的路径。

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
  ```

另请参阅

- 通过以下方式管理 Greengrass 核心设备 AWS Systems Manager

- AWS Systems Manager 用户指南 中的什么是 AWS Systems Manager？
- 关于《AWS Systems Manager 用户指南》中的 Systems Manager 代理

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
1.2.4	错误修复和改进 更新此组件以获取代理版本 3.2.2303.0。
1.2.3	错误修复和改进 • 在 Greengrass 上使用 snap 添加代理组件安装的重试次数。 • 更新代理组件的配置，使其仅使用 Greengrass 中的 Onprem 身份。 • 仅当安装的代理版本与 Greengrass SSM 代理组件的版本不匹配时，才更新此组件以更新代理。
1.1.0	此版本包含错误修复和改进。
1.0.0	初始版本。

代币兑换服务

令牌交换服务组件 (`aws.greengrass.TokenExchangeService`) 提供了可用于与自定义组件中的AWS服务进行交互的AWS凭证。

令牌交换服务将亚马逊弹性容器服务 (Amazon ECS) 容器实例作为本地服务器运行。此本地服务器使用您在 Greengras s 核心核心组件中配置的AWS IoT角色别名连接到AWS IoT凭证提供程序。该组件提供了两个环境变量，`AWS_CONTAINER_CREDENTIALS_FULL_URI`和`AWS_CONTAINER_AUTHORIZATION_TOKEN`。`AWS_CONTAINER_CREDENTIALS_FULL_URI`定义此本地服务器的 URI。当组件创建 S AWS DK 客户端时，客户端会识别此 URI 环境变量，并使用中的令牌`AWS_CONTAINER_AUTHORIZATION_TOKEN`连接到令牌交换服务并检索AWS凭证。这允许 Greengrass 核心设备调用服务操作。AWS有关如何在自定义组件中使用此组件的更多信息，请参阅与AWS服务互动。

> **Important**
>
> 2016 年 7 月 13 日，AWSSDK 中增加了以这种方式获取AWS凭证的支持。您的组件必须使用在该日期或之后创建的 AWS SDK 版本。有关更多信息，请参阅《亚马逊弹性容器服务开发者指南》中的使用支持的AWS软件开发工具包。

主题

- 版本
- 类型
- 操作系统
- 依赖项
- 配置
- 本地日志文件
- 更改日志

版本

此组件有以下版本：

- 2.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

依赖项

这个组件没有任何依赖关系。

配置

此组件没有任何配置参数。

本地日志文件

该组件使用与 Greengrass nucleus 组件相同的日志文件。

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

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

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.0.3	初始版本。

物联网 SiteWise OPC-UA 采集器

物联网 SiteWise OPC-UA 收集器组件 (`aws.iot.SiteWiseEdgeCollectorOpcua`) 使 AWS IoT SiteWise 网关能够从本地 OPC-UA 服务器收集数据。

使用此组件， AWS IoT SiteWise 网关可以连接到多台 OPC-UA 服务器。有关 AWS IoT SiteWise 网关的更多信息，请参阅《AWS IoT SiteWise 用户指南》中的在边缘使用 AWS IoT SiteWise。

主题

- 版本
- Type
- 操作系统
- 要求
- 依赖项
- 配置
- 输入数据
- 输出数据
- 本地日志文件
- 故障排除和调试
- 许可证
- 更改日志
- 另请参阅

版本

此组件有以下版本：

- 2.4.x

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Type

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- Greengrass 核心设备必须在以下平台之一上运行：
  - OS：Ubuntu 18.04 或更高版本

    架构：x86_64 (AMD64) 或 ARMv8 (Aarch64)
  - OS：Red Hat Enterprise Linux (RHEL) 8

    架构：x86_64 (AMD64) 或 ARMv8 (Aarch64)
  - OS：Amazon Linux 2

    架构：x86_64 (AMD64) 或 ARMv8 (Aarch64)
  - OS：Debian 11

    架构：x86_64 (AMD64) 或 ARMv8 (Aarch64)
  - 操作系统：Windows Server 2019 或更高版本

    架构：x86_64 (AMD64)

- Greengrass 核心设备必须允许与 OPC-UA 服务器的出站网络连接。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

下表列出了该组件所有版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.3.0 <3.0 <3.0.0	硬性
流管理器	>2.0.10<3.0.0	硬性
秘密经理	>=2.0.8 <3.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件没有任何配置参数。

您可以使用 AWS IoT SiteWise 控制台或 API 来配置 IoT SiteWise OPC-UA 收集器组件。有关更多信息，请参阅《AWS IoT SiteWise 用户指南》中的步骤 4：添加数据源-可选。

输入数据

此组件仅接受以下格式的数据，所有其他格式的数据都将被忽略并丢弃。下表将 OPC UA 数据类型映射到其 SiteWise等效类型。

SiteWise 数据类型	OPC UA 数据类型	描述
STRING	String	最大长度为 1024 字节的字符串。

SiteWise 数据类型	OPC UA 数据类型	描述
	Guid XmlElement	
INTEGER	SByte Byte Int16 UInt16 Int32 UInt32* Int64*	一个有符号的 32 位整数，范围为。-2,147,483,648 to 2,147,483,647
DOUBLE	UInt32* Int64* Float Double	一个浮点数，其范围为-10^100 to 10^100和IEEE 754双精度。
BOOLEAN	Boolean	true 或 false。

* 对于 OPC UA 数据类型UInt32和Int64，INTEGER如果能够表示其值，则其 SiteWise 数据类型将为，否则将 SiteWise DOUBLE是。

输出数据

此组件将BatchPutAssetPropertyValue消息写入 AWS IoT Greengrass 流管理器。有关更多信息，请参阅《AWS IoT SiteWise API 参考》中的 BatchPutAssetPropertyValue。

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log -Tail
   10 -Wait
  ```

故障排除和调试

此组件包括一个新的事件日志，可帮助客户识别和修复问题。日志文件与本地日志文件分开，位于以下位置。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgeCollectorOpcua/logs/
IotSiteWiseOpcUaCollectorEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgeCollectorOpcua\logs
\IotSiteWiseOpcUaCollectorEvents.log
```

此日志包含详细信息和故障排除说明。故障排除信息与诊断一起提供，并附有如何解决问题的说明，有时还会提供指向更多信息的链接。诊断信息包括以下内容：

- 严重性级别
- Timestamp
- 其他活动特定信息

Example 示例日志

```
dataSourceConnectionSuccess:
  Summary: Successfully connected to OpcUa server
  Level: INFO
  Timestamp: '2023-06-15T21:04:16.303Z'
  Description: Successfully connected to the data source.
  AssociatedMetrics:
  - Name: FetchedDataStreams
    Description: The number of fetched data streams for this data source
    Value: 1.0
    Namespace: IoTSiteWise
    Dimensions:
    - Name: SourceName
      Value: SourceName{value=OPC-UA Server}
    - Name: ThingName
      Value: test-core
  AssociatedData:
  - Name: DataSourceTrace
    Description: Name of the data source
    Data:
    - OPC-UA Server
  - Name: EndpointUri
    Description: The endpoint to which the connection was attempted.
    Data:
    - '"opc.tcp://10.0.0.1:1234"'
```

许可证

该组件是根据 Greengrass 核心软件许可协议发布的。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
2.4.2	错误修复和改进 • 修复了 OPC UA 服务器发现期间可能多次发现节点的问题。 • 修复了快照功能，以确保每个快照数据点的时间戳都是新的。
2.4.1	错误修复和改进 • 修复了与代理支持有关的问题。 • 修复了线程清理失败并导致数据阻塞的问题。
2.4.0	新功能 • 添加事件日志，便于识别和修复问题。 错误修复和改进 • 修复了 OPC-UA 客户端在连接到使用 OPC-UA 规范版本 1.05 的 OPC-UA 服务器时导致证书错误的问题。
2.3.0	新功能 • 在 Linux 上添加对 Greengrass n ucleu s HTTP 代理配置的支持。 错误修复和改进 • 修复了一个安全问题 (CVE-2019-19135)。

2.2.0	新功能
	• 增加了对在 Linux ARMv8 架构上安装数据收集包的支持。
	• Linux armv8 的最低要求：
	  • 内存：4 GB
	  • CPU：ARM Cortex-A72 或同等规格
	错误修复和改进
	• 改进了节点发现过程中的指标记录。
	• 改进了对不支持的数据类型的处理。
	• 改进了数据流错误的记录。

2.1.3	新功能 • 添加对 Windows Server 2019 或更高版本的支持。

版本	更改
	错误修复和改进 • 改进了在不支持的设备上部署此组件时的错误消息。

2.1.1	新功能
	• 增加了对配置以下订阅属性的支持：
	  • DataChangeTrigger-您可以定义启动数据变更警报的条件。
	  • QueueSize-OPC-UA 服务器上特定指标的队列深度，其中监控项目的通知已排队。
	  • PublishingIntervalMilliseconds-创建订阅时指定的发布周期间隔（以毫秒为单位）。
	  • SnapshotFrequencyMilliseconds -您可以配置快照频率超时设置，以确保 AWS IoT SiteWise Edge 摄取稳定的数据流。
	• 此版本支持采集BAD质量数据，并根据以下数据质量筛选数据：
	  • UNCERTAIN 质量数据
	  • BAD质量数据
	错误修复和改进
	• 对买家指标的改进。
	• 修复了在连接到启用加密的服务器时有时会出现问题的安全编码。
	• 修复了属性组更新失败的问题。

2.0.3	错误修复和改进。
2.0.2	错误修复并改进了与 edge 同步的资产优先级。
2.0.1	初始版本。

另请参阅

- 什么是 AWS IoT SiteWise？ 在《AWS IoT SiteWise 用户指南》中。

物联网 SiteWise OPC-UA 数据源模拟器

IoT SiteWise OPC-UA 数据源模拟器组件 (`aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator`) 启动生成示例数据的本地 OPC-UA 服务器。使用此 OPC-UA 服务器模拟网关上物联网 SiteWise OPC-UA 收集器组件读取的数据源。AWS IoT SiteWise然后，您可以使用此示例数据探索AWS IoT SiteWise要素。有关AWS IoT SiteWise网关的更多信息，请参阅《AWS IoT SiteWise用户指南》中的在边缘使用AWS IoT SiteWise。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 本地日志文件
- 更改日志
- 另请参阅

版本

此组件有以下版本：

- 1.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- Greengrass 核心设备必须能够使用本地主机上的端口 4840。此组件的本地 OPC-UA 服务器在此端口上运行。

依赖项

部署组件时，AWS IoT Greengrass还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

下表列出了该组件所有版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.3.0 <3.0 <3.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件没有任何配置参数。

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log -Tail 10 -Wait
  ```

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
1.0.0	初始版本。 添加对 Windows Server 2016 或更高版本的支持。

另请参阅

- 什么是AWS IoT SiteWise？ 在《AWS IoT SiteWise用户指南》中。

物联网 SiteWise 发行商

物联网 SiteWise 发布者组件 (`aws.iot.SiteWiseEdgePublisher`) 使 AWS IoT SiteWise 网关能够将数据从边缘导出到 AWS Cloud。

有关 AWS IoT SiteWise 网关的更多信息，请参阅《AWS IoT SiteWise 用户指南》中的在边缘使用 AWS IoT SiteWise。

主题

- 版本
- 类型
- 操作系统
- 要求
- 依赖项
- 配置
- 输入数据
- 本地日志文件
- 故障排除和调试
- 许可证
- 更改日志
- 另请参阅

版本

此组件有以下版本：

- 3.1.x
- 3.0.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- Greengrass 核心设备必须在以下平台之一上运行：
  - OS：Ubuntu 18.04 或更高版本

    架构：x86_64 (AMD64) 或 ARMv8 (Aarch64)
  - OS：Red Hat Enterprise Linux (RHEL) 8

    架构：x86_64 (AMD64) 或 ARMv8 (Aarch64)
  - OS：Amazon Linux 2

    架构：x86_64 (AMD64) 或 ARMv8 (Aarch64)
  - OS：Debian 11

    架构：x86_64 (AMD64) 或 ARMv8 (Aarch64)
  - 操作系统：Windows Server 2019 或更高版本

    架构：x86_64 (AMD64)
- Greengrass 核心设备必须连接到互联网。
- Greengrass 核心设备必须获得授权才能执行该操作。`iotsitewise:BatchPutAssetPropertyValue`有关更多信息，请参阅授权核心设备与 AWS 服务交互。

  Example 权限策略

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

```
}
```

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
`data.iotsitewise.region.amazonaws.com`	443	支持	将数据发布到 AWS IoT SiteWise。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

下表列出了此组件的 2.0.x 到 2.2.x 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
Greengrass 核	>=2.3.0<3.0.0	硬性
流管理器	>=2.0.10<3.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件没有任何配置参数。

您可以使用 AWS IoT SiteWise 控制台或 API 来配置 IoT SiteWise 发布者组件。有关更多信息，请参阅《AWS IoT SiteWise 用户指南》中的步骤 3：配置发布者-可选。

输入数据

该组件从 AWS IoT Greengrass 流管理器读取`PutAssetPropertyValueEntry`消息。有关更多信息，请参阅 AWS IoT SiteWise API 参考[PutAssetPropertyValueEntry](#)中的。

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log -Tail 10 -
  Wait
  ```

故障排除和调试

此组件包括新的事件日志，可帮助客户识别和修复问题。日志文件与本地日志文件分开，位于以下位置。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/
IotSiteWisePublisherEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgePublisher\logs
\IotSiteWisePublisherEvents.log
```

此日志包含详细信息和故障排除说明。故障排除信息与诊断一起提供，并附有如何解决问题的说明，有时还提供指向更多信息的链接。诊断信息包括以下内容：

- 严重性级别
- Timestamp
- 其他活动特定信息

Example 示例日志

```
accountBeingThrottled:
  Summary: Data upload speed slowed due to quota limits
  Level: WARN
  Timestamp: '2023-06-09T21:30:24.654Z'
  Description: The IoT SiteWise Publisher is limited to the "Rate of data points
 ingested"
    quota for a customers account. See the associated documentation and associated
    metric for the number of requests that were limited for more information. Note
    that this may be temporary and not require any change, although if the issue
 continues
    you may need to request an increase for the mentioned quota.
  FurtherInformation:
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/quotas.html
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/troubleshooting-
gateway.html#gateway-issue-data-streams
  AssociatedMetrics:
  - Name: TotalErrorCount
    Description: The total number of errors of this type that occurred.
    Value: 327724.0
  AssociatedData:
  - Name: AggregatePropertyAliases
```

```
    Description: The aggregated property aliases of the throttled data.
    FileLocation: /greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/./logs/data/
AggregatePropertyAliases_1686346224654.log
```

许可证

该组件是根据 Greengrass 核心软件许可协议发布的。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
3.1.2	错误修复和改进 • 修复了 3.1.1 版本中引入的 CPU 使用率过高的问题。
3.1.1	错误修复和改进 • 添加额外的日志记录，用于在发生错误时识别受影响的数据别名。 • 在本地强制执行 AWS IoT SiteWise API 对采集数据的年限的限制。 • 修复了当有多个 Amazon S3 目标时，Publisher 会混淆 StreamManager 直播检查点的问题。 • 修复了发布者如何从 StreamManager 直播中读取数据的性能瓶颈。
3.1.0	新功能 • 增加了对将数据作为 parquet 文件发布到 Amazon S3 的支持。 • 增加了对 AWS IoT SiteWise 缓冲摄取的支持。
3.0.0	错误修复和改进 • 修复了与代理支持有关的问题。 新功能 • 支持从 MQTT 代理提取数据。
2.4.1	错误修复和改进 • 允许组件与 Java Corretto 11 版本 11.0.20.8.1 及更高版本一起使用。与 Java Corretto 版本 11.0.20.8.1 一起使用时，组件版本 2.4.0 和 2.3.3 会显示"`Could not find or load main class`"错误消息。

版本	更改
2.4.0	新功能 • 添加新的事件日志，便于识别和修复问题。 错误修复和改进 • 改进了发布者检查点恢复。
2.3.3	错误修复和改进 • 提高了支持高吞吐量的能力。
2.3.2	错误修复和改进 • 修复了下载发布器配置时对 HTTP 代理的支持。

2.3.1	新功能
	• 增加了对在 Linux ARMv8 架构上安装数据收集包的支持。
	• Linux armv8 的最低要求：
	  • 内存：4 GB
	  • CPU：ARM Cortex-A72 或同等规格

2.2.3	错误修复和改进 • 删除对不在可重检异常列表中的通用异常的重试。
2.2.2	错误修复和改进 • 重新引入了 AWS IoT SiteWise 通过 HTTP 代理服务器上传数据的支持。
2.2.1	Note 此版本不支持 HTTP 代理配置。2.2.2 及更高版本重新引入了对此功能的支持。 新功能 • 为该组件添加支持，以便在将数据上传到时切换压缩 AWS IoT SiteWise。

<table>
<tr><th>版本</th><th>更改</th></tr>
<tr><td>2.2.0</td><td> Note
此版本不支持 HTTP 代理配置。2.2.2 及更高版本重新引入了对此功能的支持。

新功能
更新此组件以在将数据发送到 AWS IoT SiteWise 服务之前对其进行压缩。
在大多数情况下，与该组件的先前版本相比，此更改可将带宽使用量减少75%。
在大多数情况下，此更改会将 CPU 使用率增加多达 5%。在处理大量数据的网关上，此更改可将 CPU 使用率增加多达 15%。
此更改不会影响 AWS IoT SiteWise 服务费或服务配额的使用。
添加对 Windows Server 2019 或更高版本的支持。
错误修复和改进
修复了在检查点文件损坏时阻止此组件启动的问题。</td></tr>
<tr><td>2.1.4</td><td>错误修复和改进
修复了与 Java 版本 8 的兼容性。</td></tr>
<tr><td>2.1.3</td><td> Warning
此版本不再可用，但美国东部（俄亥俄州）、加拿大（中部）和 AWS GovCloud （美国东部）地区除外。此组件版本需要 Java 版本 11 或更高版本才能运行。此版本的改进将在此组件的更高版本中提供。

错误修复和改进
改进了在不支持的设备上部署此组件时的错误消息。
更新以记录数据上传失败时的错误。</td></tr>
</table>

版本	更改
2.1.2	错误修复和改进 • 更新以在数据过期后立即调用过期的数据导出功能。
2.1.1	错误修复和改进。
2.1.0	新功能 • 增加了对先将最新数据发布到云端的支持。 • 增加了对不将过期数据发布到云端的支持。 • 增加了对在本地存储过期数据的支持。 错误修复和改进 • 减少磁盘 I/O 和相应的延迟。
2.0.2	错误修复和改进。
2.0.1	初始版本。

另请参阅

- 什么是 AWS IoT SiteWise？ 在《AWS IoT SiteWise 用户指南》中。

物联网 SiteWise 处理器

物联网 SiteWise 处理器组件 (`aws.iot.SiteWiseEdgeProcessor`) 使 AWS IoT SiteWise 网关能够在边缘处理数据。

借助此组件， AWS IoT SiteWise 网关可以使用资产模型和资产来处理网关设备上的数据。有关 AWS IoT SiteWise 网关的更多信息，请参阅《AWS IoT SiteWise 用户指南》中的在边缘使用 AWS IoT SiteWise。

主题

- 版本
- 类型
- 操作系统
- 要求

- 依赖项
- 配置
- 本地日志文件
- 许可证
- 更改日志
- 另请参阅

版本

此组件有以下版本：

- 3.2.x
- 3.1.x
- 3.0.x
- 2.2.x
- 2.1.x
- 2.0.x

类型

此组件是一个通用组件 (`aws.greengrass.generic`)。Greengrass 核心运行组件的生命周期脚本。

有关更多信息，请参阅 组件类型。

操作系统

此组件可以安装在运行以下操作系统的核心设备上：

- Linux
- Windows

要求

此组件具有以下要求：

- Greengrass 核心设备必须在以下平台之一上运行：

  - 操作系统：Ubuntu 20.04 或 18.04

    架构：x86_64 (AMD64)
  - OS：Red Hat Enterprise Linux (RHEL) 8

    架构：x86_64 (AMD64)
  - OS：Amazon Linux 2

    架构：x86_64 (AMD64)
  - 操作系统：Windows Server 2019 或更高版本

    架构：x86_64 (AMD64)
- Greengrass 核心设备必须允许端口 443 上的入站流量。
- Greengrass 核心设备必须允许端口 443 和 8883 上的出站流量。
- 以下端口已保留供以下用户使用 AWS IoT SiteWise：80、443、3001、4569、4572、8000、8081、8082、8084、8085、8086、8445、9000、9500、11080 和 50010。使用预留端口通信可能导致连接终止。

  > **Note**
  >
  > 只有此组件的 2.0.15 及更高版本需要端口 8087。

- Greengrass 设备角色必须拥有允许您在设备上使用网关的 AWS IoT SiteWise 权限。 AWS IoT Greengrass V2 有关更多信息，请参阅《AWS IoT SiteWise 用户指南》中的要求。

端点和端口

除了基本操作所需的端点和端口外，此组件还必须能够对以下端点和端口执行出站请求。有关更多信息，请参阅 允许设备流量通过代理或防火墙。

Endpoint	端口	必需	描述
`model.iotsitewise.` *`region`*`.amazonaws.com`	443	支持	获取有关您的 AWS IoT SiteWise 资产和资产

Endpoint	端口	必需	描述
			模型的信息。
edge.iotsitewise. *region*.amazonaws.com	443	支持	获取有关核心设备 AWS IoT SiteWise 网关配置的信息。
ecr.*region*.amazonaws.com	443	支持	从亚马逊弹性容器器注册表下载 AWS IoT SiteWise Edge 网关 Docker 镜像。
iot.*region*.amazonaws.com	443	支持	获取您的设备终端节点 AWS 账户。
sts.*region*.amazonaws.com	443	支持	获取您的身份证 AWS 账户。
monitor.iotsitewise. *region*.amazonaws.com	443	不支持	如果您访问核心设备上的 AWS IoT SiteWise Monitor 门户，则为必填项。

依赖项

部署组件时， AWS IoT Greengrass 还会部署其依赖项的兼容版本。这意味着您必须满足组件及其所有依赖项的要求才能成功部署该组件。本节列出了此组件已发布版本的依赖关系以及定义每个依赖项的组件版本的语义版本限制。您还可以在AWS IoT Greengrass 控制台中查看组件每个版本的依赖关系。在组件详细信息页面上，查找 “依赖关系” 列表。

下表列出了此组件的 2.0.x 到 2.1.x 版本的依赖关系。

依赖关系	兼容版本	依赖关系类型
代币兑换服务	>=2.0.3 <3.0.0	硬性
流管理器	>=2.0.10 <3.0.0	硬性
Greengrass CLI	>=2.3.0 <3.0 <3.0.0	硬性

有关组件依赖关系的更多信息，请参阅组件配方参考。

配置

此组件没有任何配置参数。

本地日志文件

此组件使用以下日志文件。

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log
```

查看此组件的日志

- 在核心设备上运行以下命令以实时查看此组件的日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为 AWS IoT Greengrass 根文件夹的路径。

Linux

```
sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log -Tail 10 -
Wait
```

许可证

此组件包括以下第三方软件/许可：

- Apache-2.0
- 麻省理工学院
- BSD-2 条款
- BSD-3 条款
- CDDL-1.0
- CDDL-1.1
- ISC
- Zlib
- GPL-3.0-with-GCC-Exception
- 公共领域
- Python-2.0
- Unicode-dfs-2015
- BSD-1 条款
- OpenSSL
- EPL-1.0
- EPL-2.0
- GPL-2.0-with-classpath-exception
- MPL-2.0
- CC0-1.0

- JSON

该组件是根据 Greengrass 核心软件许可协议发布的。

更改日志

下表描述了该组件的每个版本中的更改。

版本	更改
3.2.0	性能改进 • 优化 API 服务以减少内存占用量，并减少安装所需的磁盘空间 • 这使整个组件的初始内存使用量减少了 2 GB（现在启动时使用 7.5 GB 的内存，但仍建议使用 16 GB 的内存），并将整个组件的下载大小减少 500 MB（现在需要下载 1.4 GB）。 新功能 • `GetAssetPropertyValueAggregates` API 现在支持边缘的 15 分钟聚合窗口。 • 此组件无需再有端口 8081 和 8082 即可正常运行。 Note AWS IoT SiteWise 数据平面 API（例如）的本地端点已从更改`http://localhost:8081` 为`http://localhost:11080/data` 。`get-asset-property-value` AWS IoT SiteWise 控制平面 API（例如）的本地端点已从更改`http://localhost:11080` 为`http://localhost:11080/control` 。`list-asset-models` AWS 始终建议您使用 SiteWise Edge 网关 HTTPS 终端节点。这些端点没有改变。 错误修复和改进 • 现在，如果上一次同步中断，则从同步 AWS IoT SiteWise 会将资源转换为有效状态。这将修复强制重启后某些资源损坏的问题。 • 修复了一种罕见的情况，即如果在同步期间修改资源，则资源可能会在边缘损坏。现在，如果检测到这种情况，同步将失败，并且将在下次同步时重试该资源。

版本	更改
	• 修复了可能允许外部调用 API 的 HTTP 端点的问题。现在只有 HTTPS 可以用来调用本地环回地址之外的 API。 • `ListAssets` API 现在显示存储在边缘的资产的资产层次结构。 • 修复了数据处理包无法在 Windows 上重新启动、升级或降级的问题。 • 修复了 Windows 操作系统数据处理包中的一个错误，该错误导致客户无法使用凭据连接 MQTT 代理。
3.1.3	错误修复和改进 • 修复了当某些资源实际出现故障时，数据处理包错误地报告同步成功的问题。 • 允许多个资源使用相同的名称，前提是它们的父项不相同。
3.1.1	错误修复和改进 • 修复了由于时区不匹配而导致的 Sigv4 请求失败的问题。 • 修复了转换和指标属性在重新启动后依赖属性时停止计算的问题。 • 启用对自定义 Stream Manager 端口配置的支持。 • 修复了同步到边缘的属性可能会停止更新的问题。
3.1.0	错误修复和改进 • 修复了 `ListAssetModels` API 无法生成下一个令牌的问题。
3.0.0	新功能 • 支持从 MQTT 代理提取数据。

<table>
<tr><th>版本</th><th>更改</th></tr>
<tr><td>2.2.1</td><td>错误修复和改进
调整同步过程，使控制平面数据存储与云的运行方式更加一致。这会稍微影响升级。

Note
在 2.2.1 或更高版本上同步的控制平面数据将与之前的版本不兼容。要降级到以前的版本，你需要完成全新安装。这不会影响升级，在先前版本上同步的数据将适用于版本 2.2.1。
对 AWS 证书链进行了其他修改，以确定 AWS IoT Greengrass V2 证书的优先级。</td></tr>
<tr><td>2.1.37</td><td>错误修复和改进
弃用 dependency-routing-service 流程并将其功能移到流程中，以减少通信 property-state-service 进程的资源消耗。
将 <code>get-asset-property-value-history</code> API 的最大结果限制增加到 20,000，以匹配使用的限制 AWS IoT SiteWise。
修复了未指定最大结果限制时，未在 <code>get-asset-property-value-history</code> API 的分页结果中提供下一个标记的问题。</td></tr>
<tr><td>2.1.35</td><td>错误修复和改进
修改 AWS 凭证链以确定 AWS IoT Greengrass 凭证的优先级。
修复了作为 Thing 群组 AWS IoT 的一部分进行部署时的帐号检测问题。</td></tr>
<tr><td>2.1.34</td><td>错误修复和改进
在 Linux 上调整指标/转换计算以使用多线程。为了兼容，Windows 继续运行单线程计算。
修复了某些计算窗口缺少指标计算的问题。</td></tr>
<tr><td>2.1.33</td><td>错误修复和改进
修复了向 Greengrass 控制台报告错误状态的问题。</td></tr>
</table>

版本	更改
2.1.32	错误修复和改进 • 添加对自定义用户名和群组的支持。
2.1.31	错误修复和改进 • 增加了对在中建模的数据计算时间加权平均值和时间加权标准差的支持。AWS IoT SiteWise
2.1.29	错误修复和改进 • 增加了对边缘资源过滤功能的支持。
2.1.28	错误修复和改进 • 优化资源同步，使大量资产能够从边缘同步 AWS Cloud 到边缘。
2.1.24	错误修复和改进 • 修复了在第二次同步资源时导致仪表板消失的问题。
2.1.23	错误修复和改进 • 增加了`aws.iot.SiteWiseEdgeProcessor` 安装过程的超时时间，以避免因特网连接速度慢时安装失败。 • 优化了资源同步，提高了云端和边缘之间的同步效率。
2.1.21	Warning 从 2.0.x 升级到 2.1.x 将导致本地数据丢失。 新功能 • 添加对 Windows Server 2019 或更高版本的支持。 • 移除基于 Linux 的操作系统的 docker。
2.0.16	此版本包含错误修复和改进。

版本	更改
2.0.15	错误修复和改进 • 将此组件用于资源同步 API 操作的端口从 8085 更改为 8087。因此，此组件现在需要端口 8087 才可用。此组件仍需要端口 8085 才可用。 • 更新 AWS OpsHub 身份验证以在登录期间拒绝未经授权的用户，而不是在用户尝试调用 API 操作时拒绝未经授权的用户。
2.0.14	此版本包含错误修复和改进。
2.0.13	错误修复和改进 • 修复了一个问题，当此组件向 Amazon CloudWatch 指标报告数据时，它现在可以正确指示哪些数据未建模。
2.0.9	错误修复和改进 • 提高了在核心设备上创建和更新 AWS IoT SiteWise 资源的可靠性。 • 添加其他本地 API 操作，您可以使用这些操作来监控核心设备上安装了哪些组件、每个组件的版本以及每个组件的状态。您可以在核心设备上的 for AWS IoT SiteWise 应用程序的 “设置” 选项卡上查看此信息。 AWS OpsHub • 为该组件运行的 Docker 容器添加健康状态。您可以运行`docker ps`命令来查看容器的运行状况。
2.0.7	错误修复和改进 • 修复了对在核心设备上查看 AWS IoT SiteWise Monitor 传送门的支持。
2.0.6	错误修复和改进 • 修复了 AWS IoT SiteWise `statetime()` 此组件在核心设备上计算的`earliest()` 、和`latest()`函数。
2.0.5	错误修复和改进 • 添加对该组件在核心设备上计算的变换中的 AWS IoT SiteWise `pretrigger()` 函数的支持。 • 更改此组件存储用于身份验证的轻型目录访问协议 (LDAP) 配置的路径。
2.0.2	初始版本。

另请参阅

- 什么是 AWS IoT SiteWise？ 在《AWS IoT SiteWise 用户指南》中。

发布商支持的组件

> 发布商支持的组件处于预览版中AWS IoT Greengrass，可能会发生变化。不支持这些组件AWS。如果每个组件存在任何问题，您必须联系发行商。

Greengrass Publisher 支持的组件由第三方组件供应商开发、提供和服务。第三方组件供应商要么来自AWS Partner设备目录、AWS英雄，要么来自社区供应商。您可以直接联系第三方组件供应商购买此目录中的组件。

Greengrass Publisher 支持的组件包括以下内容：

主题

- aishield.edge
- AI EdgeLabs 传感器
- Greengrass S3 Ingestor

aishield.edge

该组件由 AisHield 开发并提供支持，由博世提供支持。使用 Aishield.edge 提高你的人工智能安全性。该组件旨在向边缘设备无缝部署基于威胁的、量身定制的防御措施，从而保护您的设备免受人工智能攻击。

此组件具有以下优点：

- 从 AiShield AI Security 的漏洞分析无缝过渡到内部的强化边缘防御 AWS
- 轻松在多个边缘设备上部署量身定制的防御
- 针对不同的 AI 设置量身定制的广泛保护，支持各种模型类型和框架
- 通过无缝集成Amazon SageMaker和 Greengrass 工作流程，随时了解最新情况
- 通过将数据直接中继到，获得对潜在威胁的即时见解 AWS IoT Core
- 来自 AisHield AI Security 的 Marketplace 上用于边缘防御部署的有凝聚力的人工智能安全途径 AWS

此组件必须在以下平台上运行：

- 操作系统：Linux

如果您有兴趣购买此组件，请联系博世软件和数字解决方案：`<AIShield.Contact@bosch.com>`。

AI EdgeLabs 传感器

此组件由 AI 开发并支持 EdgeLabs。AI EdgeLabs Sensor 是一款基于容器的应用程序，包含基于 AI 的威胁检测和防御功能。AI Sensor 封装在 Greengrass 组件中，并作为独立容器与其他 Greengrass 组件一起部署在核心设备上。

当前的组件是一个基于容器的代理，可以持续验证网络通信，在边缘主机或物联网网关上运行的软件中查找威胁模式。该组件使用 eBPF、进程带宽的行为验证和基于主机的配置。该组件的主要功能基于 NDR/IPS 和 EDR 功能。

此组件具有以下优点：

- 基于人工智能的威胁检测，针对网络攻击和恶意软件 (EDR/NDR)
- 基于 AI 的自动事件响应 (IPS)
- 主机本地威胁情报，将外部数据传输量降至最低
- 使用 Docker 和 Greengrass 进行轻量级部署

此组件必须在以下平台之一上运行：

- 操作系统：Linux

如果你有兴趣购买此组件，请联系 AI EdgeLabs：`<contact@edgelabs.ai>`。

Greengrass S3 Ingestor

该组件由内森·格洛弗开发并得到其支持。Greengrass S3 Ingestor 组件专为与直播管理器组件一起使用而设计。此组件从流管理器获取以行分隔的 JSON 消息流，然后将其批处理到 GZIP 文件中。该组件支持将数据高效地摄取到 Amazon S3 中以供进一步处理或存储。此组件不支持实时向发送数据。AWS Cloud

此组件必须在以下平台之一上运行：

- 操作系统：Linux
- 操作系统：Windows

如果你有兴趣购买此组件，请联系 Nathan Glover：`<nathan@glovers.id.au>`。

社区组件

Greengrass 软件目录是 Greengrass 社区开发的 Greengrass 组件的索引。您可以从该目录中下载、修改和部署组件来创建 Greengrass 应用程序。你可以通过以下链接查看目录：https://github.com/ aws-greengrass/ aws-greengrass-software-catalog。

每个组件都有一个可供您浏览的公共 GitHub 存储库。请查看 Greengrass 软件目录，查找社区 GitHub 组件的完整列表。例如，此目录包括以下组件：

- Amazon Kinesis Video Streams

  该组件从使用实时流媒体协议 (RTSP) 的本地摄像机接收音频和视频流。然后，该组件将音频和视频流上传到 Amazon Kinesis Video Streams。
- 蓝牙物联网网关

  此组件使用支持与低功耗蓝牙 (LE) 设备通信的BluePy库来创建蓝牙 LE 客户端接口。
- 证书轮换器

  该组件提供了一种在您的队列中大规模轮换 AWS IoT Greengrass 核心设备证书和私钥的方法。
- 容器化安全隧道

  该组件提供了一个 Docker 容器，用于安全隧道传输，其中包含所有依赖项和匹配库，包含一个不依赖特定主机操作系统的可重复使用的配方。
- Grafana

  此组件使您能够在 Greengrass 核心设备上托管 Grafana 服务器。您可以使用 Grafana 仪表板来可视化和管理核心设备上的数据。
- 适用于亚马逊 Lookout for Vision 的 gStreamer

  该组件提供了一个 gStreamer 插件，因此你可以在自定义 gStreamer 管道中执行 Lookout for Vision 异常检测。
- 家庭助理

该组件使客户能够使用 Home Assist ant 对智能家居设备进行本地控制。它提供与边缘和云端 AWS 服务的集成，以提供扩展Home Assistant的家庭自动化解决方案。

- InfluxdbGrafana 仪表板

  该组件提供了设置 InfluxDB 和 Grafana 组件的一键式体验。它将 InfluxDB 连接到 Grafana，并自动设置本地 Grafana 仪表板，该仪表板可以实时呈现遥测数据。 AWS IoT Greengrass

- InfluxDB

  该组件在 Greengrass 核心设备上提供了 InfluxDB 时间序列数据库。您可以使用此组件来处理来自物联网传感器的数据、实时分析数据以及监控边缘操作。

- InfluxDB 发行商

  该组件将 AWS IoT Greengrass 系统运行状况遥测数据从 Nucleus 发射器插件中继到 Influ xDB。该组件还可以将自定义遥测数据转发到InfluxDB。

- 物联网发布订阅框架

  该框架提供了应用架构、模板代码和可部署的示例，可帮助 AWS IoT Greengrass 使用 v2 自定义组件提高分布式事件驱动的 IoT pubsub 应用程序的代码质量。有关更多信息，请参阅 创建 AWS IoT Greengrass 组件。

- Jupyter 实验室

  此组件部署 JupyterLab 到 AWS IoT Greengrass 核心设备。Jupyter 环境可以访问由设置的流程和环境变量资源 AWS IoT Greengrass，从而简化了测试和开发用 Python 编写的组件的过程。

- 本地 Web 服务器

  此组件使您能够在 Greengrass 核心设备上创建本地 Web 用户界面。例如，您可以创建本地 Web 用户界面，使您能够配置设备和应用程序设置或监控设备。

- LoRaWaN 协议适配器

  该组件从使用 LoRaWa N 协议（一种低功耗广域网 (LPWAN) 协议）的本地无线设备摄取数据。该组件使您无需与云端通信即可在本地分析和处理数据。

- Modbus

  该组件使用 ModbustCP 协议从本地设备收集数据，并将其发布到选定的数据流。

- Node-red

此组件使用 NPM 在 AWS IoT Greengrass 核心设备上安装 Node-RED。该组件依赖于 Node-red 身份验证组件，该组件必须明确部署和配置。你可以使用适用于 Greengrass 的 Node-RED CLI 将节点红色流程部署到设备。 AWS IoT Greengrass

- Node-red Docker

  此组件使用官方的 Node-red Docker 容器在 AWS IoT Greengrass 核心设备上安装 Node-red。该组件依赖于 Node-red 身份验证组件，该组件必须明确部署和配置。你可以使用适用于 Greengrass 的 Node-RED CLI 将节点红色流程部署到设备。 AWS IoT Greengrass

- Node-red 身份验证

  此组件配置用户名和密码以保护在核心设备上运行的 Node-RED 实例。 AWS IoT Greengrass

- OpenThread边境路由器

  此组件部署 OpenThread 边界路由器 Docker 容器。该组件有助于组成包含 Thread 边界路由器的 Matter 设备。

- OSI Pi 流媒体数据连接器

  该组件提供从 OSI Pi 数据存档到现代数据架构的流式实时数据采集。 AWS它集成到通过 AWS IoT PubSub 消息传递进行集中管理的 OSI Pi 资产框架。

- Parsec 提供商

  该组件使 AWS IoT Greengrass 设备能够使用云原生计算基金会 (CNCF) 的开源 Parsec 项目集成硬件安全解决方案。

- PostgreSQL 数据库

  该组件为边缘的 Postgre SQ L 关系数据库提供支持。客户可以使用此组件在 docker 容器中配置和管理本地 PostgreSQL 实例。

- S3 文件上传器

  此组件监控目录中的新文件，将其上传到亚马逊简单存储服务 (Amazon S3) Service，然后在成功上传后将其删除。

- Secrets Manager 客户端

  此组件提供了一个 CLI 工具，其他需要在配方生命周期脚本中从 Secrets Manager 组件检索密钥的组件可以使用该工具。

- TES 路由到集装箱

此组件在 AWS IoT Greengrass 设备上配置 nftables 或 iptables，以便它可以将该组件与容器一起使用。代币兑换服务

- WebRTC

  该组件从连接到 AWS IoT Greengrass 核心设备的 RTSP 摄像机接收音频和视频流。然后，该组件通过亚马逊 Kinesis Video Streams 将音频和视频流转换为 peer-to-peer通信或中继。

要请求功能或报告错误，请在存储库中为该组件打开一个 GitHub 问题。 AWS 不为社区组件提供支持。有关更多信息，请参阅每个组件存储库中的CONTRIBUTING.md文件。

AWS提供的几个组件也是开源的。有关更多信息，请参阅 开源AWS IoT Greengrass核心软件。

AWS IoT Greengrass开发工具

使用AWS IoT Greengrass开发工具创建、测试、构建、发布和部署自定义 Greengrass 组件。

- Greengrass 开发套件 CLI

  使用本地AWS IoT Greengrass开发环境中的开发套件命令行界面 (GDK CLI)，从 Greengr ass 软件目录中的模板和社区组件创建组件。您可以使用 GDK CLI 来构建该组件，并将该组件作为私有组件发布到您的AWS IoT GreengrassAWS 账户服务中。

- Greengrass 命令行界面

  使用 Greengrass 核心设备上的 Greengrass 命令行界面 (Greengrass CLI) 来部署和调试 Greengrass 组件。Greengrass CLI 是一个可以部署到核心设备的组件，用于创建本地部署、查看已安装组件的详细信息以及浏览日志文件。

- 本地调试控制台

  使用 Greengrass 核心设备上的本地调试控制台，使用本地仪表板 Web 界面部署和调试 Greengrass 组件。本地调试控制台是一个组件，您可以将其部署到核心设备上，以创建本地部署并查看有关已安装组件的详细信息。

AWS IoT Greengrass还提供了以下可在自定义 Greengrass 组件中使用的软件开发工具包：

- AWS IoT Device SDK，其中包含进程间通信 (IPC) 库。有关更多信息，请参阅 使用AWS IoT Device SDK与 Greengrass 原子核、其他组件进行通信 AWS IoT Core。

- 流管理器 SDK，可用于将数据流传输到AWS Cloud。有关更多信息，请参阅 管理 Greengrass 核心设备上的数据流。

主题

- AWS IoT Greengrass开发套件命令行界面
- Greengrass 命令行界面
- 使用 AWS IoT Greengrass 测试框架

AWS IoT Greengrass开发套件命令行界面

AWS IoT Greengrass开发套件命令行界面 (GDK CLI) 提供的功能可帮助您开发自定义 Gre engrass 组件。您可以使用 GDK CLI 来创建、构建和发布自定义组件。使用 GDK CLI 创建组件存储库时，可以从 Gre engrass 软件目录中的模板或社区组件开始。然后，您可以选择将文件打包为 ZIP 存档、使用 Maven 或 Gradle 构建脚本或运行自定义构建命令的构建系统。创建组件后，您可以使用 GDK CLI 将其发布到AWS IoT Greengrass服务，这样您就可以使用AWS IoT Greengrass控制台或 API 将该组件部署到您的 Greengrass 核心设备上。

在没有 GDK CLI 的情况下开发 Greengrass 组件时，每次创建组件的新版本时，都必须更新组件配方文件中的版本和工件 URI。当您使用 GDK CLI 时，它可以在您每次发布组件的新版本时自动为您更新版本和构件 URI。

GDK CLI 是开源的，可在上使用。 GitHub您可以自定义和扩展 GDK CLI 以满足您的组件开发需求。我们邀请您在 GitHub 仓库中打开议题和拉取请求。你可以通过以下链接找到 GDK CLI 的源代码：https://github.com/aws-greengrass/ aws-greengrass-gdk-cli。

先决条件

要安装和使用 Greengrass 开发套件 CLI，你需要以下内容：

- AWS 账户。如果没有，请参阅设置一个 AWS 账户。
- 一台具有互联网连接的 Windows、macOS 或 Unix 类似 Unix 的开发计算机。
- 对于 GDK CLI 版本 1.1.0 或更高版本，您的开发计算机上安装了 Python 3.6 或更高版本。

  对于安装在开发计算机上的 GDK CLI 版本 1.0.0、Python 3.8 或更高版本。
- Git 已安装在您的开发计算机上。

- AWS Command Line Interface(AWS CLI) 已在开发计算机上安装并使用凭据进行配置。有关更多信息，请参阅《AWS Command Line Interface用户指南》AWS CLI中的 “安装、更新AWS CLI和卸载” 和 “配置”。

> **Note**
>
> 如果您使用树莓派或其他 32 位 ARM 设备，请安装 AWS CLI V1。 AWS CLIV2 不适用于 32 位 ARM 设备。有关更多信息，请参阅安装、更新和卸载AWS CLI版本 1。

- 要使用 GDK CLI 向AWS IoT Greengrass服务发布组件，您必须具有以下权限：
  - `s3:CreateBucket`
  - `s3:GetBucketLocation`
  - `s3:PutObject`
  - `greengrass:CreateComponentVersion`
  - `greengrass:ListComponentVersions`
- 要使用 GDK CLI 构建其构件存在于 S3 存储桶而不是本地文件系统中的组件，您必须具有以下权限：
  - `s3:ListBucket`

  此功能适用于 GDK CLI 版本 1.1.0 及更高版本。

更改日志

下表描述了 GDK CLI 的每个版本中的更改。有关更多信息，请参阅上的 GDK CLI 版本页面。 GitHub

版本	更改
1.6.2	错误修复和改进 • 修复了 Windows gradlew.bat 由于相对路径而无法运行的问题。 • 对日志、测试和打包进行了细微改进。
1.6.1	错误修复和改进 • 为 CLI 参数解析添加了安全补丁。 • 使 GDK 能够将最新的 Greengrass 测试框架 (GTF) 版本名称作为默认 GTF 版本。

版本	更改
	• 允许 GDK 推荐使用旧版 GTF 的客户更新到最新版本。
1.6.0	新功能 • 在和命令期间添加了针对 Greengrass 配方架构的配方验证检查。component build component publish此更新可帮助开发人员在组件创建过程的早期识别其组件配方中的可操作问题。 • 向模板添加可通过test-e2e init命令下拉的置信度测试套件。该置信度测试套件包括八个通用测试，这些测试可用于和扩展以满足基本的组件测试需求。 错误修复和改进 • 将命令使用的默认 Greengrass 测试框架 (GTF) 版本更新为版本 1.2.0。test-e2e
1.5.0	错误修复和改进 如果build_system 是，则更新excludes构建选项识别的模式zip。此版本现在可以识别根据通配符匹配路径名的全局模式。这允许自定义指定要从哪些目录中排除。
1.4.0	新功能 • 添加新config命令，启动交互式提示以修改现有 GDK 配置文件中的字段。 • 在继续操作之前，修改gdk component build和gdk component publish命令以验证配方大小是否在 Greengrass 要求范围内（<=16000 字节）。 错误修复和改进 • 当配方语法错误导致构建无法完成时，在gdk component build 命令的输出中添加额外的日志记录。 • 由于开放测试框架已otf-options 重命名otf-version 为 Greengrass 测试框架，因此将gtf-options 和gtf-version 分别重命名为和。

版本	更改
1.3.0	新功能 • 添加新test-e2e命令以支持使用开放 end-to-end 测试框架测试组件。 • 添加了一个新的配置选项zip_name，以支持 zip 编译系统中可配置的 zip 文件名。 • 将 GDK 配置文件中的region属性设为可选。 错误修复和改进 • 修复了使用参数初始化 GDK 项目时，即使指定的模板或存储库不存在也会创建新目录的问题。--name
1.2.3	错误修复和改进 • 修复了由于错误处理不正确而导致存储桶创建失败的问题。 • 修复了组件配方中的列表结构被移除的问题。
1.2.2	错误修复和改进 • 配方密钥不再区分大小写。 • 在创建新存储桶之前，添加检查以确定存储桶中是否存在AWS 区域并可供用户访问。要求用户拥有GetBucketLocation 权限。 • 修复了 GDK CLI 配置文件中excludes关键字的问题。
1.2.1	错误修复和改进 • 接受gdk-config.json 文件中区域配置条目AWS 区域中的加拿大（中部ca-central-1 ）()。 • 修复了publish命令的 --region GDK CLI 参数存在的问题。

版本	更改
1.2.0	新功能 • 将该options条目添加到 GDK CLI 配置文件中的配置中。buildexcludes在使用编zip译系统时，支持在 options zip 工件中排除某些文件。 • 添加gradlew编译系统以使用 Gradle Wrapper 来构建组件。 • 为构建选项添加了对 Kotlin DSL gradle 构建文件的支持。 • 在 GDK CLI publish 配置文件中为配置添加一个options条目。支持file_upload_args 下方，options以便在将文件上传到 Amazon S3 时提供额外的参数。 错误修复和改进 • 修复了 Gradle 版本在运行构建命令之前未清理的问题。 • 修复了生成命令失败时构建未退出的问题。 • 改进了gdk component list命令的输出格式。

<table>
<tr><th>版本</th><th>更改</th></tr>
<tr><td>1.1.0</td><td>新功能
添加对 Gradle 编译系统的支持。
在 Windows 设备上添加对 Maven 编译系统的支持。
将--bucket参数添加到组件发布命令中。您可以使用此参数指定 GDK CLI 上传组件工件的确切存储桶。
将--name参数添加到组件 init 命令中。您可以使用此选项指定 GDK CLI 初始化组件的文件夹。
添加对 S3 存储桶中存在但不存在于本地组件构建文件夹中的组件工件的支持。您可以使用此功能来降低大型组件工件（例如机器学习模型）的带宽成本。
错误修复和改进
更新组件 publish 命令以在发布组件之前检查组件是否已构建。如果未生成组件，则此命令现在会为您构建组件。
修复了当 ZIP 文件名包含大写字母时，zip 编译系统无法在 Windows 设备上构建的问题。
改进了运行低于 3.8 的 Python 版本的设备的日志消息格式并将默认日志级别更改为。INFO
将 Python 的最低版本要求更改为 Python 3.6。</td></tr>
<tr><td>1.0.0</td><td>初始版本。</td></tr>
</table>

安装或更新AWS IoT Greengrass开发套件命令行界面

AWS IoT Greengrass开发套件命令行接口 (GDK CLI) 基于 Python 构建，因此pip您可以使用它安装在开发计算机上。

Tip

你也可以在 Python 虚拟环境（例如 ven v）中安装 GDK CLI。有关更多信息，请参阅 Python 3 文档中的虚拟环境和包。

安装或更新 GDK CLI

1. 运行以下命令从其GitHub存储库中安装最新版本的 GDK CLI。

```
python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
```

> **Note**
>
> 要安装特定版本的 GDK CLI，请将 *VersionTag* 替换为要安装的版本标签。您可以在 GDK CLI 的GitHub存储库中查看 GDK CLI 的版本标签。
>
> ```
> python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@versionTag
> ```

2. 运行以下命令以验证 GDK CLI 是否成功安装。

```
gdk --help
```

如果找不到该gdk命令，请将其文件夹添加到 PATH。

- 在 Linux 设备上，添加/home/*MyUser*/.local/bin到 PATH，然后*MyUser*用您的用户名替换。
- 在 Windows 设备上，添加*PythonPath*\\Scripts到 PATH，然后*PythonPath*替换为设备上 Python 文件夹的路径。

现在，您可以使用 GDK CLI 来创建、构建和发布 Greengrass 组件。有关如何使用 GDK CLI 的更多信息，请参阅AWS IoT Greengrass开发套件命令行界面命令。

AWS IoT Greengrass开发套件命令行界面命令

AWS IoT Greengrass开发套件命令行接口 (GDK CLI) 提供了一个命令行界面，可用于在开发计算机上创建、构建和发布 Greengrass 组件。GDK CLI 命令使用以下格式。

```
gdk <command> <subcommand> [arguments]
```

安装 GDK CLI 时，安装程序会添加gdk到 PATH 中，因此您可以从命令行运行 GDK CLI。

可以在任何命令中使用以下参数：

- 使用`-h`或`--help`获取有关 GDK CLI 命令的信息。
- 使用`-v`或`--version`查看安装了哪个版本的 GDK CLI。
- 使用`-d`或`--debug`输出可用于调试 GDK CLI 的详细日志。

本节介绍了 GDK CLI 命令并提供了每个命令的示例。每个命令的提要都显示了其参数及其用法。可选参数显示在方括号中。

可用命令

- 组件
- config
- test-e2e

组件

使用AWS IoT Greengrass开发套件`component`命令行界面 (GDK CLI) 中的命令创建、构建和发布自定义 Greengrass 组件。

子命令

- init
- build
- publish
- list

init

从组件模板或社区组件初始化 Greengrass 组件文件夹。

GDK CLI 从 Greengrass 软件目录中检索社区组件，并从组件模板存储库中检索组件模板。AWS IoT Greengrass GitHub

Note

如果您使用 GDK CLI v1.0.0，则必须在空文件夹中运行此命令。GDK CLI 会将模板或社区组件下载到当前文件夹。

如果您使用 GDK CLI v1.1.0 或更高版本，则可以指定--name参数来指定 GDK CLI 下载模板或社区组件的文件夹。如果使用此参数，请指定一个不存在的文件夹。GDK CLI 会为您创建文件夹。如果您未指定此参数，GDK CLI 将使用当前文件夹，该文件夹必须为空。

如果组件使用 zip 编译系统，则 GDK CLI 会将组件文件夹中的某些文件压缩到与组件文件夹同名的 zip 文件中。例如，如果组件文件夹的名称为HelloWorld，则 GDK CLI 会创建一个名HelloWorld.zip为的 zip 文件。在组件配方中，zip 工件名称必须与组件文件夹的名称相匹配。如果您在 Windows 设备上使用 GDK CLI 版本 1.0.0，则组件文件夹和 zip 文件名必须仅包含小写字母。

如果将使用 zip 构建系统的模板或社区组件初始化为与模板或组件名称不同的文件夹，则必须在组件配方中更改 zip 工件名称。更新Artifacts和Lifecycle定义，使 zip 文件名与组件文件夹的名称相匹配。以下示例突出显示了Artifacts和Lifecycle定义中的 zip 文件名。

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
```

```
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

摘要

```
$ gdk component init
    [--language]
    [--template]
    [--repository]
    [--name]
```

参数（从组件模板初始化）

- `-l`，`--language`— 用于指定模板的编程语言。

  必须指定`--repository`或`--language`和`--template`。
- `-t`，`--template`— 用于本地组件项目的组件模板。要查看可用模板，请使用 l is t 命令。

  必须指定`--repository`或`--language`和`--template`。
- `-n`，`--name`—（可选）GDK CLI 初始化组件的本地文件夹的名称。指定一个不存在的文件夹。GDK CLI 会为您创建文件夹。

  此功能适用于 GDK CLI 版本 1.1.0 及更高版本。

参数（从社区组件初始化）

- `-r`，`--repository`— 要签到本地文件夹的社区组件。要查看可用的社区组件，请使用 l is t 命令。

  必须指定`--repository`或`--language`和`--template`。
- `-n`，`--name`—（可选）GDK CLI 初始化组件的本地文件夹的名称。指定一个不存在的文件夹。GDK CLI 会为您创建文件夹。

  此功能适用于 GDK CLI 版本 1.1.0 及更高版本。

输出

以下示例显示了运行此命令从 Python Hello World 模板初始化组件文件夹时生成的输出。

```
$ gdk component init -l python -t HelloWorld
[2021-11-29 12:51:40] INFO - Initializing the project directory with a python
 component template - 'HelloWorld'.
[2021-11-29 12:51:40] INFO - Fetching the component template 'HelloWorld-python'
 from Greengrass Software Catalog.
```

以下示例显示了运行此命令从社区组件初始化组件文件夹时生成的输出。

```
$ gdk component init -r aws-greengrass-labs-database-influxdb
[2022-01-24 15:44:33] INFO - Initializing the project directory with a component
 from repository catalog - 'aws-greengrass-labs-database-influxdb'.
[2022-01-24 15:44:33] INFO - Fetching the component repository 'aws-greengrass-labs-
database-influxdb' from Greengrass Software Catalog.
```

build

将组件的源代码构建成可以发布到AWS IoT Greengrass服务的配方和工件。GDK CLI 运行您在 GDK CLI 配置文件中指定的构建系统。`gdk-config.json`您必须在`gdk-config.json`文件所在的同一个文件夹中运行此命令。

运行此命令时，GDK CLI 会在组件文件夹的`greengrass-build`文件夹中创建配方和工件。GDK CLI 将配方保存在`greengrass-build/recipes`文件夹中，并将工件保存在`greengrass-build/artifacts/componentName/componentVersion`文件夹中。

如果您使用 GDK CLI v1.1.0 或更高版本，则组件配方可以指定存在于 S3 存储桶中但不存在于本地组件构建文件夹中的项目。在开发具有大型工件的组件（例如机器学习模型）时，可以使用此功能来减少带宽使用量。

构建组件后，您可以执行以下操作之一在 Greengrass 核心设备上对其进行测试：

- 如果您在与运行 C AWS IoT Greengrass ore 软件的设备不同的设备上进行开发，则必须发布该组件才能将其部署到 Greengrass 核心设备上。将组件发布到AWS IoT Greengrass服务，然后将其部署到 Greengrass 核心设备。有关更多信息，请参阅 p ublish 命令和创建部署。
- 如果您在运行 AWS IoT Greengrass Core 软件的同一台设备上进行开发，则可以将组件发布到AWS IoT Greengrass服务进行部署，也可以创建本地部署来安装和运行该组件。要创建本地部署，请使用 Greengrass CLI。有关更多信息，请参阅 Greengrass 命令行界面和 使用本地部署测试AWS

IoT Greengrass组件。创建本地部署时，请指定greengrass-build/recipes为配方文件夹，指定greengrass-build/artifacts为构件文件夹。

摘要

```
$ gdk component build
```

Arguments (参数)

无

输出

以下示例显示了运行此命令时生成的输出。

```
$ gdk component build
[2021-11-29 13:18:49] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:18:49] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:18:49] INFO - Building the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:18:49] INFO - Using 'zip' build system to build the component.
[2021-11-29 13:18:49] WARNING - This component is identified as using 'zip' build
 system. If this is incorrect, please exit and specify custom build command in the
 'gdk-config.json'.
[2021-11-29 13:18:49] INFO - Zipping source code files of the component.
[2021-11-29 13:18:49] INFO - Copying over the build artifacts to the greengrass
 component artifacts build folder.
[2021-11-29 13:18:49] INFO - Updating artifact URIs in the recipe.
[2021-11-29 13:18:49] INFO - Creating component recipe in 'C:\Users\MyUser\Documents
\greengrass-components\python\HelloWorld\greengrass-build\recipes'.
```

publish

将此组件发布到AWS IoT Greengrass服务。此命令将构建工件上传到 S3 存储桶，更新配方中的构件 URI，并根据配方创建新版本的组件。GDK CLI 使用您在 GDK CLI 配置文件中指定的 S3 存储桶和AWS区域。gdk-config.json您必须在gdk-config.json文件所在的同一个文件夹中运行此命令。

如果您使用 GDK CLI v1.1.0 或更高版本，则可以指定--bucket参数来指定 GDK CLI 上传组件工件的 S3 存储桶。*##########GDK CLI #######bucket-region-accountId############gdk-*

config.json#####accountID ### ID# AWS 账户如果存储桶不存在，GDK CLI 会创建该存储桶。

如果您使用 GDK CLI v1.2.0 或更高版本，则可以使用参数覆盖 GDK CLI 配置文件中AWS 区域指定的内容。`--region`您也可以使用`--options`参数指定其他选项。有关可用选项的列表，请参阅Greengrass 开发套件 CLI 配置文件。

运行此命令时，GDK CLI 会发布包含您在配方中指定的版本的组件。如果您指定`NEXT_PATCH`，GDK CLI 将使用尚不存在的下一个补丁版本。语义版本使用主调。 未成年人。 补丁编号系统。有关更多信息，请参阅语义版本规范。

> **Note**
>
> 如果您使用 GDK CLI v1.1.0 或更高版本，则在运行此命令时，GDK CLI 会检查组件是否已构建。如果组件未构建，GDK CLI 将在发布组件之前构建该组件。

摘要

```
$ gdk component publish
    [--bucket] [--region] [--options]
```

Arguments (参数)

- `-b`，`--bucket`—（可选）指定 GDK CLI 在其中发布组件工件的 S3 存储桶的名称。

  ##########GDK CLI #######bucket-region-accountId############gdk-config.json#####accountID ### ID# AWS 账户如果存储桶不存在，GDK CLI 会创建该存储桶。

  如果存储桶不存在，GDK CLI 会创建该存储桶。

  此功能适用于 GDK CLI 版本 1.1.0 及更高版本。
- `-r`，`--region`—（可选）在创建组件时AWS 区域指定 to 的名称。此参数覆盖 GDK CLI 配置中的区域名称。

  此功能适用于 GDK CLI 版本 1.2.0 及更高版本。
- `-o`，`--options`（可选）指定用于发布组件的选项列表。参数必须是有效的 JSON 字符串或包含发布选项的 JSON 文件的文件路径。此参数覆盖 GDK CLI 配置中的选项。

  此功能适用于 GDK CLI 版本 1.2.0 及更高版本。

输出

以下示例显示了运行此命令时生成的输出。

```
$ gdk component publish
[2021-11-29 13:45:29] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:45:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:45:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2021-11-29 13:45:30] INFO - Publishing the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:45:30] INFO - No private version of the component
 'com.example.PythonHelloWorld' exist in the account. Using '1.0.0' as the next
 version to create.
[2021-11-29 13:45:30] INFO - Uploading the component built artifacts to s3 bucket.
[2021-11-29 13:45:30] INFO - Uploading component artifacts to S3 bucket: {bucket}.
 If this is your first time using this bucket, add the 's3:GetObject' permission
 to each core device's token exchange role to allow it to download the component
 artifacts. For more information, see https://docs.aws.amazon.com/greengrass/v2/
developerguide/device-service-role.html.
[2021-11-29 13:45:30] INFO - Not creating an artifacts bucket as it already exists.
[2021-11-29 13:45:30] INFO - Updating the component recipe
 com.example.PythonHelloWorld-1.0.0.
[2021-11-29 13:45:30] INFO - Creating a new greengrass component
 com.example.PythonHelloWorld-1.0.0
[2021-11-29 13:45:30] INFO - Created private version '1.0.0' of the component in the
 account.'com.example.PythonHelloWorld'.
```

list

检索可用组件模板和社区组件的列表。

GDK CLI 从 Greengrass 软件目录中检索社区组件，并从组件模板存储库中检索组件模板。AWS IoT Greengrass GitHub

您可以将此命令的输出传递给 init 命令，以便从模板和社区组件初始化组件存储库。

摘要

```
$ gdk component list
    [--template]
```

```
    [--repository]
```

Arguments (参数)

- `-t`，`--template`—（可选）指定此参数可列出可用的组件模板。此命令按格式输出每个模板的名称和语言*name*-*language*。例如，在中`HelloWorld-python`，模板名称为`HelloWorld`，语言为`python`。
- `-r`，`--repository`—（可选）指定此参数可列出可用的社区组件存储库。

输出

以下示例显示了运行此命令时生成的输出。

```
$ gdk component list --template
[2021-11-29 12:29:04] INFO - Listing all the available component templates from
 Greengrass Software Catalog.
[2021-11-29 12:29:04] INFO - Found '2' component templates to display.
1. HelloWorld-python
2. HelloWorld-java
```

config

使用AWS IoT Greengrass开发套件`config`命令行界面 (GDK CLI) 中的命令修改配置文件中 GDK 的配置。`gdk-config.json`

子命令

- [update](#)

update

启动交互式提示以修改现有 GDK 配置文件中的字段。

摘要

```
$ gdk config update
    [--component]
```

Arguments (参数)

- `-c`，`--component`— 更新文件中与组件相关的字段。`gdk-config.json`这个参数是必需的，因为它是唯一的选择。

输出

以下示例显示了运行此命令来配置组件时生成的输出。

```
$ gdk config update --component
Current value of the REQUIRED component_name is (default:
 com.example.PythonHelloWorld):
Current value of the REQUIRED author is (default: author):
Current value of the REQUIRED version is (default: NEXT_PATCH):
Do you want to change the build configurations? (y/n)
Do you want to change the publish configurations? (y/n)
[2023-09-26 10:19:48] INFO - Config file has been updated. Exiting...
```

test-e2e

使用AWS IoT Greengrass开发套件`test-e2e`命令行界面 (GDK CLI) 中的命令在 GDK 项目中初始化、生成和 end-to-end 运行测试模块。

子命令

- init
- build
- run

init

使用使用 Greengrass 测试框架 (GTF) 的测试模块初始化现有的 GDK CLI 项目。

默认情况下，GDK CLI 会从上的 “AWS IoT Greengrass组件模板” 存储库中检索 maven 模块模板。GitHub这个 maven 模块附带了对 `aws-greengrass-testing-standalone` JAR 文件的依赖关系。

此命令在 GDK 项目`gg-e2e-tests`内部创建一个名为的新目录。如果测试模块目录已经存在且不为空，则命令将不执行任何操作退出。此`gg-e2e-tests`文件夹包含 Maven 项目中结构的 Cucumber 功能和步骤定义。

默认情况下，此命令将尝试使用最新版本的 GTF。

摘要

```
$ gdk test-e2e init
```

```
    [--gtf-version]
```

Arguments (参数)

- `-ov`，`--gtf-version`—（可选）与 GDK 项目中的 end-to-end 测试模块配合使用的 GTF 版本。此值必须是发行版中的 GTF 版本之一。此参数覆盖 GDK CLI 配置`gtf_version`中的。

输出

以下示例显示了运行此命令以使用测试模块初始化 GDK 项目时生成的输出。

```
$ gdk test-e2e init
[2023-12-06 12:20:28] INFO - Using the GTF version provided in the GDK test config
 1.2.0
[2023-12-06 12:20:28] INFO - Downloading the E2E testing template from GitHub into
 gg-e2e-tests directory...
```

build

Note

在生成 end-to-end 测试模块gdk component build之前，必须通过运行来构建组件。

构建 end-to-end 测试模块。GDK CLI 使用您在 GDK CLI 配置文件中的属性下指定的构建系统来构建测试模块。`gdk-config.json` `test-e2e`您必须在`gdk-config.json`文件所在的同一个文件夹中运行此命令。

默认情况下，GDK CLI 使用 maven 编译系统来构建测试模块。需要使用 Maven 才能运行该`gdk test-e2e build`命令。

如果测试功能文件包含诸如`GDK_COMPONENT_NAME`和之类的变量需要插值，则必须在构建测试模块gdk-component-build之前通过运行`GDK_COMPONENT_RECIPE_FILE`来构建组件。

运行此命令时，GDK CLI 会插入 GDK 项目配置中的所有变量，并生成`gg-e2e-tests`模块以生成最终的测试 JAR 文件。

摘要

```
$ gdk test-e2e build
```

Arguments (参数)

无

输出

以下示例显示了运行此命令时生成的输出。

```
$ gdk test-e2e build
[2023-07-20 15:36:48] INFO - Updating feature file: file:///path/to//
HelloWorld/greengrass-build/gg-e2e-tests/src/main/resources/greengrass/features/
component.feature
[2023-07-20 15:36:48] INFO - Creating the E2E testing recipe file:///path/to/
HelloWorld/greengrass-build/recipes/e2e_test_recipe.yaml
[2023-07-20 15:36:48] INFO - Building the E2E testing module
[2023-07-20 15:36:48] INFO - Running the build command 'mvn package'
.........
```

run

使用 GDK 配置文件中的测试选项运行测试模块。

Note

在运行测试gdk test-e2e build之前，必须通过运行来构建 end-to-end 测试模块。

摘要

```
$ gdk test-e2e run
    [--gtf-options]
```

Arguments (参数)

- `-oo`，`--gtf-options`—（可选）指定用于运行 end-to-end 测试的选项列表。参数必须是有效的 JSON 字符串或包含 GTF 选项的 JSON 文件的文件路径。配置文件中提供的选项与命令参数中提供的选项合并。如果两个地方都存在一个选项，则参数中的一个优先于配置文件中的选项。

  如果此命令中未指定该`tags`选项，GDK 将使用`Sample`标记。如果`ggc-archive`未指定，GDK 将下载最新版本的 Greengrass 核心档案。

输出

以下示例显示了运行此命令时生成的输出。

```
$ gdk test-e2e run
[2023-07-20 16:35:53] INFO - Downloading latest nucleus archive from url https://
d2s8p88vqu9w66.cloudfront.net/releases/greengrass-latest.zip
[2023-07-20 16:35:57] INFO - Running test jar with command java -jar /path/to/
greengrass-build/gg-e2e-tests/target/uat-features-1.0.0.jar —ggc-archive=/path/to/
aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-nucleus-latest.zip —
tags=Sample

16:35:59.693 [] [] [] [INFO]
 com.aws.greengrass.testing.modules.GreengrassContextModule - Extracting /path/
to/workplace/aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-
nucleus-latest.zip into /var/folders/7g/ltzcb_3s77nbtmkzfb6brwv40000gr/T/gg-
testing-7718418114158172636/greengrass
16:36:00.534 [gtf-1.1.0-SNAPSHOT] [] [] [INFO]
 com.aws.greengrass.testing.features.LoggerSteps - GTF Version is gtf-1.1.0-SNAPSHOT
.......
```

Greengrass 开发套件 CLI 配置文件

AWS IoT Greengrass开发套件命令行接口 (GDK CLI) 从`gdk-config.json`名为的配置文件中读取数据以生成和发布组件。此配置文件必须存在于组件存储库的根目录中。您可以使用 GDK CLI init 命令使用此配置文件初始化组件存储库。

主题

- GDK CLI 配置文件格式
- GDK CLI 配置文件示例

GDK CLI 配置文件格式

在为组件定义 GDK CLI 配置文件时，需要以 JSON 格式指定以下信息。

`gdk_version`

与此组件兼容的 GDK CLI 的最低版本。此值必须是发行版中的 GDK CLI 版本之一。

`component`

此组件的配置。

`componentName`

`author`

组件的作者或发布者。

`version`

组件版本。指定下列项之一：

- `NEXT_PATCH`— 当您选择此选项时，GDK CLI 将在您发布组件时设置版本。GDK CLI 会查询AWS IoT Greengrass服务以识别该组件的最新发布版本。然后，它将版本设置为该版本之后的下一个补丁版本。如果您之前没有发布过该组件，GDK CLI 将使用版本`1.0.0`。

  如果选择此选项，则无法使用 Greengrass CLI 在本地部署该组件并将其测试到运行 Core 软件的本地开发计算机上。AWS IoT Greengrass要启用本地部署，必须改为指定语义版本。
- 语义版本，例如。**`1.0.0`**语义版本使用主调。 未成年人。 补丁编号系统。有关更多信息，请参阅语义版本规范。

  如果您在 Greengrass 核心设备上开发要部署和测试该组件的组件，请选择此选项。要使用 Greengr ass CLI 创建本地部署，必须使用特定版本构建组件。

`build`

用于将此组件的源代码构建为工件的配置。该对象包含以下信息：

`build_system`

要使用的构建系统。从以下选项中进行选择：

- `zip`— 将组件的文件夹打包为 ZIP 文件以定义为该组件的唯一构件。为以下类型的组件选择此选项：
  - 使用解释型编程语言的组件，例如 Python 或 JavaScript。
  - 打包除代码之外的文件的组件，例如机器学习模型或其他资源。

  GDK CLI 将组件的文件夹压缩为与组件文件夹同名的 zip 文件。例如，如果组件文件夹的名称为`HelloWorld`，则 GDK CLI 会创建一个名`HelloWorld.zip`为的 zip 文件。

Note

如果您在 Windows 设备上使用 GDK CLI 版本 1.0.0，则组件文件夹和 zip 文件名必须仅包含小写字母。

当 GDK CLI 将组件的文件夹压缩为 zip 文件时，它会跳过以下文件：

- gdk-config.json 文件
- 配方文件（recipe.json或recipe.yaml）
- 生成文件夹，例如 greengrass-build

- maven— 运行mvn clean package命令将组件的源代码构建为工件。对于使用 Mave n 的组件（例如 Java 组件），请选择此选项。

  在 Windows 设备上，此功能适用于 GDK CLI v1.1.0 及更高版本。

- gradle— 运行gradle build命令将组件的源代码构建为工件。对于使用 Gradle 的组件，请选择此选项。此功能适用于 GDK CLI 版本 1.1.0 及更高版本。

  编gradle译系统支持 Kotlin DSL 作为构建文件。此功能适用于 GDK CLI 版本 1.2.0 及更高版本。

- gradlew— 运行gradlew命令将组件的源代码构建为工件。对于使用 Gradle 包装器的组件，请选择此选项。

  此功能适用于 GDK CLI 版本 1.2.0 及更高版本。

- custom— 运行自定义命令将组件的源代码构建为配方和工件。在custom_build_command参数中指定自定义命令。

custom_build_command

（可选）要为自定义编译系统运行的自定义编译命令。如果为指定，则必须指定custom此参数build_system。

Important

此命令必须在组件文件夹的以下文件夹中创建配方和工件。当您运行组件构建命令时，GDK CLI 会为您创建这些文件夹。

- 食谱文件夹：greengrass-build/recipes

- 工件文件夹：greengrass-build/artifacts/*componentName*/*componentVersion*

  将 *componentName* 替换为组件名称，并将 *componentVersion* *######* *#*或。NEXT_PATCH

您可以指定单个字符串或字符串列表，其中每个字符串都是命令中的一个单词。例如，要为 C++ 组件运行自定义生成命令，可以指定**cmake --build build --config Release**或**["cmake", "--build", "build", "--config", "Release"]**。

要查看自定义编译系统的示例，请参阅aws.greengrass.labs.LocalWebServer community component上的 GitHub。

options

（可选）在组件构建过程中使用的其他配置选项。

此功能适用于 GDK CLI 版本 1.2.0 及更高版本。

excludes

glob 模式列表，用于定义构建 zip 文件时要从组件目录中排除哪些文件。仅在为时build_system有效zip。

> **Note**
>
> 在 GDK CLI 版本 1.4.0 及更早版本中，任何与排除列表中的条目相匹配的文件都将从组件的所有子目录中排除。要在 GDK CLI 版本 1.5.0 及更高版本中实现相同的行为，请在排除列表中的现有条目前**/面加上。例如，*.txt将仅从目录中排除文本文件；**/*.txt将从所有目录和子目录中排除文本文件。
> 在 GDK CLI 版本 1.5.0 及更高版本中，当在 GDK 配置文件中定义时excludes，您可能会在组件构建期间看到警告。要禁用此警告，请将环境变量设置GDK_EXCLUDES_WARN_IGNORE为true。

GDK CLI 始终从 zip 文件中排除以下文件：

- gdk-config.json 文件
- 配方文件（recipe.json或recipe.yaml）

- 生成文件夹，例如 greengrass-build

默认情况下，不包括以下文件。但是，您可以使用该excludes选项控制将哪些文件排除在外。

- 任何以前缀 “test” (test*) 开头的文件夹
- 所有隐藏的文件
- node_modules 文件夹

如果您指定该excludes选项，GDK CLI 将仅排除您使用该excludes选项设置的文件。如果您未指定该excludes选项，GDK CLI 将排除前面提到的默认文件和文件夹。

zip_name

在生成过程中创建 zip 工件时使用的 zip 文件名。仅在为时build_system有效zip。如果为build_system空，则使用组件名称作为 zip 文件名。

publish

用于将此组件发布到AWS IoT Greengrass服务的配置。

如果您使用 GDK CLI v1.1.0 或更高版本，则可以指定--bucket参数来指定 GDK CLI 上传组件工件的 S3 存储桶。*##########GDK CLI #######bucket-region-accountId# ###########gdk-config.json#####accountID ### ID#* AWS 账户如果存储桶不存在，GDK CLI 会创建该存储桶。

该对象包含以下信息：

bucket

用于托管组件工件的 S3 存储桶名称。

region

GDK CLI 发布此组件AWS 区域的位置。

如果您使用的是 GDK CLI 1.3.0 或更高版本，则此属性是可选的。

options

（可选）创建组件版本期间使用的其他配置选项。

此功能适用于 GDK CLI 版本 1.2.0 及更高版本。

file_upload_args

一种 JSON 结构，包含在将文件上传到存储桶时发送到 Amazon S3 的参数，例如元数据和加密机制。有关允许的参数的列表，请参阅 Boto3 文档中的S3Transfer类。 。

test-e2e

（可选） end-to-end 测试组件期间要使用的配置。此功能适用于 GDK CLI 版本 1.3.0 及更高版本。

build

build_system— 要使用的构建系统。默认选项是maven。从以下选项中进行选择：

- maven— 运行mvn package命令以生成测试模块。选择此选项来构建使用 Maven 的测试模块。
- gradle— 运行gradle build命令以生成测试模块。为使用 Gradle 的测试模块选择此选项。

gtf_version

（可选）使用 GTF 初始化 GDK 项目时，用作测试模块依赖项的 Greengrass 测试框架 (GTF) 版本。 end-to-end 此值必须是发行版中的 GTF 版本之一。默认为 GTF 版本 1.1.0。

gtf_options

（可选） end-to-end 测试组件期间使用的其他配置选项。

以下列表包括您可以在 GTF 版本 1.1.0 中使用的选项。

- additional-plugins—（可选）其他黄瓜插件
- aws-region— 针对特定的区域AWS服务端点。默认为 S AWS DK 发现的内容。
- credentials-path— 可选的AWS配置文件凭据路径。默认为在主机环境中发现的凭据。
- credentials-path-rotation— AWS 证书的可选轮换持续时间。默认为 15 分钟或PT15M。
- csr-path— 用于生成设备证书的 CSR 路径。
- device-mode— 正在测试的目标设备。默认为本地设备。
- env-stage— 瞄准 Greengrass 的部署环境。默认为生产。
- existing-device-cert-arn— 要用作 Greengrass 设备证书的现有证书的 arn。

- feature-path— 包含其他功能文件的文件或目录。默认为不使用其他功能文件。
- gg-cli-version— 覆盖 Greengrass CLI 的版本。默认为中找到的值ggc.version。
- gg-component-bucket— 存放 Greengrass 组件的现有 Amazon S3 存储桶的名称。
- gg-component-overrides— Greengrass 组件重写列表。
- gg-persist— 测试运行后要保留的测试元素列表。默认行为是不保留任何内容。可接受的值为：aws.resourcesinstalled.software、和generated.files。
- gg-runtime— 影响测试与测试资源交互方式的值列表。这些值取代参数。gg.persist如果默认值为空，则假设所有测试资源都由测试用例管理，包括已安装的 Greengrass 运行时。可接受的值为：aws.resourcesinstalled.software、和generated.files。
- ggc-archive— 存档的 Greengrass 核组件的路径。
- ggc-install-root— 用于安装 Greengrass nucleus 组件的目录。默认为 test.temp.path 和测试运行文件夹。
- ggc-log-level— 为测试运行设置 Greengrass nucleus 日志级别。默认为 “信息”。
- ggc-tes-rolename— AWS IoT Greengrass Core 将担任的访问AWS服务的 IAM 角色。如果不存在具有给定名称的角色，则将创建一个默认访问策略。
- ggc-trusted-plugins— 需要添加到 Greengrass 的可信插件的路径（主机上）的逗号分隔列表。要提供 DUT 本身的路径，请在路径前面加上 'dut: '
- ggc-user-name— Greengrass 核的 user: group posixUser 值。默认为当前登录的用户名。
- ggc-version— 覆盖正在运行的 Greengrass nucleus 组件的版本。默认为 ggc.archive 中找到的值。
- log-level— 测试运行的日志级别。默认为 “信息”。
- parallel-config— 将批次索引和批次数设置为 JSON 字符串。批次索引的默认值为 0，批次数为 1。
- proxy-url— 将所有测试配置为通过此 URL 路由流量。
- tags— 仅运行功能标签。可以与 '&' 相交
- test-id-prefix— 适用于所有测试特定资源（包括AWS资源名称和标签）的通用前缀。默认为 “gg” 前缀。
- test-log-path— 将包含整个测试运行结果的目录。默认为 “测试结果”。
- test-results-json— 用于确定生成的 Cucumber JSON 报告是否已写入磁盘的标志。默认值为 true。
- test-results-log— 用于确定控制台输出是否生成的写入磁盘的标志。默认值为 false。

- `test-results-xml`— 用于确定生成的 JUnit XML 报告是否已写入磁盘的标志。默认值为 true。
- `test-temp-path`— 生成本地测试工件的目录。默认为以 gg-testing 为前缀的随机临时目录。
- `timeout-multiplier`— 为所有测试超时提供乘数。默认值为 1.0。

GDK CLI 配置文件示例

您可以参考以下 GDK CLI 配置文件示例来帮助您配置 Greengrass 组件环境。

你好世界 (Python)

以下 GDK CLI 配置文件支持运行 Python 脚本的 Hello World 组件。此配置文件使用编`zip`译系统将组件的 Python 脚本打包为 ZIP 文件，GDK CLI 将其作为构件上传。

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip",
        "options": {
           "excludes": [".*"]
        }
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
    },
  "test-e2e":{
    "build":{
        "build_system": "maven"
```

```
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
         "tags": "Sample"
     }
  },
  "gdk_version": "1.6.1"
  }
}
```

你好 World (Java)

以下 GDK CLI 配置文件支持运行 Java 应用程序的 Hello World 组件。此配置文件使用编maven译系统将组件的 Java 源代码打包到 JAR 文件中，GDK CLI 将其作为构件上传。

```
{
  "component": {
    "com.example.JavaHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "maven"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
  },
  "test-e2e":{
    "build":{
        "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
         "tags": "Sample"
     }
```

```
  },
  "gdk_version": "1.6.1"
  }
}
```

社区组件

Greengrass 软件目录中的几个社区组件使用 GD K CLI。您可以浏览这些组件存储库中的 GDK CLI 配置文件。

查看社区组件的 GDK CLI 配置文件

1. 运行以下命令列出使用 GDK CLI 的社区组件。

   ```
   gdk component list --repository
   ```

   响应列出了使用 GDK CLI 的每个社区组件的 GitHub 存储库名称。每个存储库都存在于`awslabs`组织中。

   ```
   [2022-02-22 17:27:31] INFO - Listing all the available component repositories from
    Greengrass Software Catalog.
   [2022-02-22 17:27:31] INFO - Found '6' component repositories to display.
   1. aws-greengrass-labs-database-influxdb
   2. aws-greengrass-labs-telemetry-influxdbpublisher
   3. aws-greengrass-labs-dashboard-grafana
   4. aws-greengrass-labs-dashboard-influxdb-grafana
   5. aws-greengrass-labs-local-web-server
   6. aws-greengrass-labs-lookoutvision-gstreamer
   ```

2. 通过以下 URL 打开社区组件的 GitHub 存储库。*community-component-name*替换为上一步中的社区组件的名称。

   ```
   https://github.com/awslabs/community-component-name
   ```

Greengrass 命令行界面

Greengrass 命令行界面 (CLI) 允许您在设备上AWS IoT Greengrass与 Core 进行交互，以便在本地开发组件和调试问题。例如，您可以使用 Greengrass CLI 创建本地部署并在核心设备上重启组件。

部署 Greengrass CLI 组件 `aws.greengrass.Cli` ()，在核心设备上安装 Greengrass CLI。

> **Important**
>
> 我们建议您仅在开发环境中使用此组件，而不是在生产环境中使用。此组件提供对生产环境中通常不需要的信息和操作的访问。遵循最低权限原则，将此组件仅部署到需要的核心设备。

主题

- 安装 Greengrass CLI
- Greengrass CLI 命令

安装 Greengrass CLI

你可以通过以下方式之一安装 Greengrass CLI：

- 首次在设备上设置 AWS IoT Greengrass Core 软件时，请使用该`--deploy-dev-tools`参数。您还必须指定`--provision true`才能应用此参数。
- 在您的设备上部署 Greengrass CLI 组件 `aws.greengrass.Cli` ()。

本节介绍部署 Greengrass CLI 组件的步骤。有关在初始设置期间安装 Greengrass CLI 的信息，请参阅。教程：AWS IoT Greengrass V2 入门

先决条件

要部署 Greengrass CLI 组件，必须满足以下要求：

- AWS IoT Greengrass 在您的核心设备上安装并配置了核心软件。有关更多信息，请参阅 教程：AWS IoT Greengrass V2 入门。
- 要使用部 AWS CLI 署 Greengrass CLI，您必须已安装并配置了。 AWS CLI有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的配置 AWS CLI。
- 您必须获得授权才能使用 Greengrass CLI 与核心软件进行交互。 AWS IoT Greengrass 要使用 Greengrass CLI，请执行以下任一操作：
  - 使用运行 C AWS IoT Greengrass ore 软件的系统用户。
  - 使用具有 root 权限或管理员权限的用户。在 Linux 核心设备上`sudo`，您可以使用获取根权限。
  - 部署组件时，请使用您在`AuthorizedPosixGroups`或`AuthorizedWindowsGroups`配置参数中指定的组中的系统用户。有关更多信息，请参阅 Greengrass CLI 组件配置。

部署 Greengrass CLI 组件

完成以下步骤，将 Greengrass CLI 组件部署到您的核心设备：

部署 Greengrass CLI 组件（控制台）

1. 登录 AWS IoT Greengrass 控制台。
2. 在导航菜单中，选择组件。
3. 在组件页面的公有组件选项卡上，选择 `aws.greengrass.Cli`。
4. 在 aws.greengrass.Cli 页面上，选择部署。
5. 从 “添加到部署” 中，选择 “创建新部署”。
6. 在指定目标页面的部署目标下的目标名称列表中，选择要部署到的 Greengrass 组，然后选择下一步。
7. 在 “选择组件” 页面上，确认已选择该aws.greengrass.Cli组件，然后选择 “下一步”。
8. 在配置组件页面上，保留默认配置设置，然后选择下一步。
9. 在 “配置高级设置” 页面上，保留默认配置设置，然后选择 “下一步”。
10. 在 “审阅” 页面上，单击 “部署”

部署 Greengrass CLI 组件 ()AWS CLI

1. 在您的设备上，创建一个`deployment.json`文件来定义 Greengrass CLI 组件的部署配置。此文件应如下所示：

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.Cli": {
      "componentVersion": "2.12.3",
      "configurationUpdate": {
        "merge": "{\"AuthorizedPosixGroups\":\"<group1>,<group2>,...,<groupN>\",
\"AuthorizedWindowsGroups\":\"<group1>,<group2>,...,<groupN>\"}"
      }
    }
  }
}
```

- 在 `target` 字段中，按以下格式将 *`targetArn`* 替换为部署目标的事物或事物组的 Amazon 资源名称（ARN）：

    - 事物：arn:aws:iot:*region*:*account-id*:thing/*thingName*
    - 事物组：arn:aws:iot:*region*:*account-id*:thinggroup/*thingGroupName*
- 在aws.greengrass.Cli组件对象中，按如下方式指定值：

  version

  Greengrass CLI 组件的版本。

  configurationUpdate.AuthorizedPosixGroups

  （可选）包含以逗号分隔的系统组列表的字符串。您授权这些系统组使用 Greengrass CLI 与核心软件进行交互。 AWS IoT Greengrass 您可以指定群组名称或群组 ID。例如，group1,1002,group3授权三个系统组（group11002、和group3）使用 Greengrass CLI。

  如果您未指定要授权的任何群组，则可以以 root 用户 sudo () 或运行核心软件的系统用户身份使用 Greengrass CLI。 AWS IoT Greengrass

  configurationUpdate.AuthorizedWindowsGroups

  （可选）包含以逗号分隔的系统组列表的字符串。您授权这些系统组使用 Greengrass CLI 与核心软件进行交互。 AWS IoT Greengrass 您可以指定群组名称或群组 ID。例如，group1,1002,group3授权三个系统组（group11002、和group3）使用 Greengrass CLI。

  如果您未指定要授权的任何群组，则可以以管理员或运行 Core 软件的系统用户身份使用 Greengrass CLI。 AWS IoT Greengrass

2. 运行以下命令在设备上部署 Greengrass CLI 组件：

```
$ aws greengrassv2 create-deployment --cli-input-json file://path/to/deployment.json
```

在安装过程中，该组件会在设备上的*/greengrass/v2*/bin文件夹greengrass-cli中添加一个符号链接，然后您可以从此路径运行 Greengrass CLI。要在没有绝对路径的情况下运行 Greengrass CLI，*/greengrass/v2*/bin请将您的文件夹添加到 PATH 变量中。要验证 Greengrass CLI 的安装，请运行以下命令：

Linux or Unix

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

Windows

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

您应看到以下输出：

```
Usage: greengrass-cli [-hV] [--ggcRootPath=<ggcRootPath>] [COMMAND]
Greengrass command line interface

      --ggcRootPath=<ggcRootPath>
                    The AWS IoT Greengrass V2 root directory.
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
  help                Show help information for a command.
  component           Retrieve component information and stop or restart
                        components.
  deployment          Create local deployments and retrieve deployment status.
  logs                Analyze Greengrass logs.
  get-debug-password  Generate a password for use with the HTTP debug view
                        component.
```

如果`greengrass-cli`未找到，则部署可能无法安装 Greengrass CLI。有关更多信息，请参阅 故障排除 AWS IoT Greengrass V2。

Greengrass CLI 命令

Greengrass CLI 提供了一个命令行界面，用于在本地与核心设备进行交互。AWS IoT GreengrassGreengrass CLI 命令使用以下格式。

```
$ greengrass-cli <command> <subcommand> [arguments]
```

默认情况下，该文件夹中的`greengrass-cli`可执行文件与该*/greengrass/v2*/bin/文件夹中运行的 AWS IoT Greengrass Core 软件版本进行*/greengrass/v2*交互。如果您调用未放置在此位置的可执行文件，或者您想在其他位置与AWS IoT Greengrass核心软件进行交互，则必须使用以下方法之一来明确指定要与之交互的AWS IoT Greengrass核心软件的根路径：

- 将 GGC_ROOT_PATH 环境变量设置为 */greengrass/v2*。
- 将--ggcRootPath */greengrass/v2*参数添加到您的命令中，如以下示例所示。

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

可以在任何命令中使用以下参数：

- --help用于获取有关特定 Greengrass CLI 命令的信息。
- --version用于获取有关 Greengrass CLI 版本的信息。

本节介绍了 Greengrass CLI 命令并提供了这些命令的示例。每个命令的提要都显示了其参数及其用法。可选参数显示在方括号中。

可用命令

- 组件
- 部署
- 日志
- get-debug-password

组件

使用component命令与核心设备上的本地组件进行交互。

子命令

- details
- 列表
- 重新启动
- stop

details

检索一个组件的版本、状态和配置。

摘要

```
greengrass-cli component details --name <component-name>
```

Arguments (参数)

--name，-n。 组件名称。

输出

以下示例显示了运行此命令时生成的输出。

```
$ sudo greengrass-cli component details --name MyComponent

Component Name: MyComponent
Version: 1.0.0
State: RUNNING
Configuration: null
```

列表

检索设备上安装的每个组件的名称、版本、状态和配置。

摘要

```
greengrass-cli component list
```

Arguments (参数)

无

输出

以下示例显示了运行此命令时生成的输出。

```
$ sudo greengrass-cli component list

Components currently running in Greengrass:
Component Name: FleetStatusService
Version: 0.0.0
State: RUNNING
Configuration: {"periodicUpdateIntervalSec":86400.0}
```

```
Component Name: UpdateSystemPolicyService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Nucleus
Version: 2.0.0
State: FINISHED
Configuration: {"awsRegion":"region","runWithDefault":
{"posixUser":"ggc_user:ggc_group"},"telemetry":{}}
Component Name: DeploymentService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: TelemetryAgent
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Cli
Version: 2.0.0
State: RUNNING
Configuration: {"AuthorizedPosixGroups":"ggc_user"}
```

重新启动

重启组件。

摘要

```
greengrass-cli component restart --names <component-name>,...
```

Arguments (参数)

--names，-n。 组件名称。至少需要一个组件名称。您可以指定其他组件名称，用逗号分隔每个名称。

输出

无

stop

停止运行组件。

摘要

```
greengrass-cli component stop --names <component-name>,...
```

Arguments (参数)

`--names`，`-n`。 组件名称。至少需要一个组件名称。如果需要，您可以指定其他组件名称，用逗号分隔每个名称。

输出

无

部署

使用`deployment`命令与核心设备上的本地组件进行交互。

要监控本地部署的进度，请使用`status`子命令。您无法使用控制台监控本地部署的进度。

子命令

- 创建
- 取消
- list
- status

创建

使用指定的组件配方、构件和运行时参数创建或更新本地部署。

摘要

```
greengrass-cli deployment create
    --recipeDir path/to/component/recipe
    [--artifactDir path/to/artifact/folder ]
    [--update-config {component-configuration}]
    [--groupId <thing-group>]
    [--merge "<component-name>=<component-version>"]...
    [--runWith "<component-name>:posixUser=<user-name>[:<group-name>]"]...
    [--systemLimits "{component-system-resource-limits}]"]...
```

```
    [--remove <component-name>,...]
    [--failure-handling-policy <policy name[ROLLBACK, DO_NOTHING]>]
```

Arguments (参数)

- `--recipeDir`，`-r`。 包含组件配方文件的文件夹的完整路径。
- `--artifactDir`，`-a`。 包含要包含在部署中的构件文件的文件夹的完整路径。构件文件夹必须包含以下目录结构：

```
/path/to/artifact/folder/<component-name>/<component-version>/<artifacts>
```

- `--update-config`，`-c`。 部署的配置参数，以 JSON 字符串或 JSON 文件形式提供。JSON 字符串应采用以下格式：

```
{ \
  "componentName": { \
    "MERGE": {"config-key": "config-value"}, \
    "RESET": ["path/to/reset/"] \
  } \
}
```

  MERGE和区RESET分大小写并且必须为大写字母。

- `--groupId`，`-g`。 部署的目标事物组。
- `--merge`，`-m`。 要添加或更新的目标组件的名称和版本。必须以格式提供组件信息*<component>*=*<version>*。为每个要指定的其他组件使用单独的参数。如果需要，可使用`--runWith`参数提供`posixUser``posixGroup`、和用于运行组件`windowsUser`的信息。
- `--runWith`。 `posixUser``posixGroup`、和用于运行通用组件或 Lambda 组件`windowsUser`的信息。您必须按以下格式提供此信息*<component>*:{posixUser|windowsUser}=*<user>*[:*<=posixGroup>*]。例如，您可以指定**HelloWorld:posixUser=ggc_user:ggc_group**或**HelloWorld:windowsUser=ggc_user**。为每个其他选项使用单独的参数进行指定。

  有关更多信息，请参阅 配置运行组件的用户。

- `--systemLimits`。 适用于核心设备上的通用和非容器化 Lambda 组件进程的系统资源限制。您可以配置每个组件的进程可以使用的最大 CPU 和 RAM 使用量。指定序列化的 JSON 对象或 JSON 文件的文件路径。JSON 对象必须采用以下格式。

```
{  \
  "componentName": { \
```

```
      "cpus": cpuTimeLimit, \
      "memory": memoryLimitInKb \
    } \
  }
```

您可以为每个组件配置以下系统资源限制：

- cpus— 此组件的进程可以在核心设备上使用的最大 CPU 时间。核心设备的总 CPU 时间等于 CPU 核心的设备数量。例如，在具有 4 个 CPU 内核的核心设备上，您可以将此值设置为，2将该组件的进程使用率限制为每个 CPU 内核的 50%。在具有 1 个 CPU 内核的设备上，您可以将此值设置为，0.25将此组件的进程使用率限制在 25% 以内。如果将此值设置为大于 CPU 内核数的数字，则 AWS IoT Greengrass Core 软件不会限制组件的 CPU 使用率。
- memory— 此组件的进程可以在核心设备上使用的最大 RAM 量（以千字节为单位）。

有关更多信息，请参阅 为组件配置系统资源限制。

此功能适用于 Linux 核心设备上的 Greengrass n ucleus 组件和 Greengrass CLI 的 v2.4.0 及更高版本。 AWS IoT Greengrass 目前不支持在 Windows 核心设备上使用此功能。

- --remove。 要从本地部署中移除的目标组件的名称。要移除已从云部署中合并的组件，必须按以下格式提供目标事物组的群组 ID：

  Greengrass nucleus v2.4.0 and later

  ```
  --remove <component-name> --groupId <group-name>
  ```

  Earlier than v2.4.0

  ```
  --remove <component-name> --groupId thinggroup/<group-name>
  ```

- --failure-handling-policy。 定义部署失败时采取的操作。您可以指定两个操作：
  - ROLLBACK –
  - DO_NOTHING –

  此功能适用于 v2.11.0 及更高版本。Greengrass 核

输出

以下示例显示了运行此命令时生成的输出。

```
$ sudo greengrass-cli deployment create \
    --merge MyApp1=1.0.0 \
```

```
    --merge MyApp2=1.0.0 --runWith MyApp2:posixUser=ggc_user \
    --remove MyApp3 \
    --recipeDir recipes/ \
    --artifactDir artifacts/

Local deployment has been submitted! Deployment Id: 44d89f46-1a29-4044-
ad89-5151213dfcbc
```

取消

取消指定的部署。

摘要

```
greengrass-cli deployment cancel
    -i <deployment-id>
```

参数

`-i`。 要取消的部署的唯一标识符。部署 ID 将在`create`命令的输出中返回。

输出

- 无

list

检索最近 10 个本地部署的状态。

摘要

```
greengrass-cli deployment list
```

Arguments (参数)

无

输出

以下示例显示了运行此命令时生成的输出。根据您的部署状态，输出显示以下状态值之一：`IN_PROGRESSSUCCEEDED`、或`FAILED`。

```
$ sudo greengrass-cli deployment list

44d89f46-1a29-4044-ad89-5151213dfcbc: SUCCEEDED
Created on: 6/27/23 11:05 AM
```

status

检索特定部署的状态。

摘要

```
greengrass-cli deployment status -i <deployment-id>
```

Arguments (参数)

`-i`。 部署的 ID。

输出

以下示例显示了运行此命令时生成的输出。根据您的部署状态，输出显示以下状态值之一：`IN_PROGRESS``SUCCEEDED`、或`FAILED`。

```
$ sudo greengrass-cli deployment status -i 44d89f46-1a29-4044-ad89-5151213dfcbc

44d89f46-1a29-4044-ad89-5151213dfcbc: FAILED
Created on: 6/27/23 11:05 AM
Detailed Status: <Detailed deployment status>
Deployment Error Stack: List of error codes
Deployment Error Types: List of error types
Failure Cause: Cause
```

日志

使用该`logs`命令分析核心设备上的 Greengrass 日志。

子命令

- get
- 列表关键词

- list-log-files

get

收集、筛选和可视化 Greengrass 日志文件。此命令仅支持 JSON 格式的日志文件。您可以在 nucleus 配置中指定日志格式。

摘要

```
greengrass-cli logs get
    [--log-dir path/to/a/log/folder]
    [--log-file path/to/a/log/file]
    [--follow true | false ]
    [--filter <filter> ]
    [--time-window <start-time>,<end-time> ]
    [--verbose ]
    [--no-color ]
    [--before <value> ]
    [--after <value> ]
    [--syslog ]
    [--max-long-queue-size <value> ]
```

Arguments (参数)

- `--log-dir`，`-ld`。 用于检查日志文件的目录路径，例如***/greengrass/v2*/logs**。请勿与`--syslog`。为每个要指定的其他目录使用单独的参数。必须使用`--log-dir`或中的至少一个`--log-file`。您也可以在单个命令中同时使用这两个参数。
- `--log-file`，`-lf`。 您要使用的日志目录的路径。为每个要指定的其他目录使用单独的参数。必须使用`--log-dir`或中的至少一个`--log-file`。您也可以在单个命令中同时使用这两个参数。
- `--follow`，`-fol`。 显示发生的日志更新。Greengrass CLI 继续运行并从指定的日志中读取数据。如果您指定时间窗口，则 Greengrass CLI 会在所有时间窗口结束后停止监控日志。
- `--filter`，`-f`。 用作过滤器的关键字、正则表达式或键值对。以字符串、正则表达式或键值对的形式提供此值。为每个要指定的其他过滤器使用单独的参数。

  评估后，在单个参数中指定的多个过滤器由 OR 运算符分隔，在其他参数中指定的过滤器与 AND 运算符组合在一起。例如，如果您的命令包含`--filter "installed" --filter "name=alpha,name=beta"`，则 Greengrass CLI 将筛选并显示同时包含`installed`关键字和值为或的键name的日志消息。`alpha` `beta`

- --time-window，-t。 显示日志信息的时间窗口。您可以同时使用精确的时间戳和相对偏移量。您必须按以下格式提供此信息*<begin-time>*,*<end-time>*。如果您未指定开始时间或结束时间，则该选项的值默认为当前系统日期和时间。为每个额外的时间窗口使用单独的参数进行指定。

  Greengrass CLI 支持以下格式的时间戳：

  - yyyy-MM-DD，例如，2020-06-30。使用此格式时，时间默认为 00:00:00。

    yyyyMMDD，例如，20200630。使用此格式时，时间默认为 00:00:00。

    HH:mm:ss，例如，15:30:45。使用此格式时，该日期默认为当前系统日期。

    HH:mm:ssSSS，例如，15:30:45。使用此格式时，该日期默认为当前系统日期。

    YYYY-MM-DD'T'HH:mm:ss'Z'，例如，2020-06-30T15:30:45Z。

    YYYY-MM-DD'T'HH:mm:ss，例如，2020-06-30T15:30:45。

    yyyy-MM-dd'T'HH:mm:ss.SSS，例如，2020-06-30T15:30:45.250。

  相对偏移量指定与当前系统时间相比的时间段偏移。Greengrass CLI 支持以下相对偏移量格式:. +|-[<*value*>h|hr|hours][*value*m|min|minutes][*value*]s|sec|seconds

  例如，以下参数用于指定当前时间之前的 1 小时到 2 小时 15 分钟之间的时间窗口--time-window -2h15min,-1hr。

- --verbose。 显示日志消息中的所有字段。请勿与--syslog。
- --no-color，-nc。 移除颜色编码。日志消息的默认颜色编码使用粗体红色文本。仅支持类似 Unix 的终端，因为它使用 ANSI 转义序列。
- --before，-b。 在匹配的日志条目之前要显示的行数。默认值为 0。
- --after，-a。 匹配的日志条目后要显示的行数。默认值为 0。
- --syslog。 使用 RFC3164 定义的系统日志协议处理所有日志文件。请勿与--log-dir和一起使用--verbose。系统日志协议使用以下格式:. "<$Priority>$Timestamp $Host $Logger ($Class): $Message" 如果您未指定日志文件，则 Greengrass CLI 会从以下位置/var/log/messages读取日志消息：、或。/var/log/syslog /var/log/system.log

  AWS IoT Greengrass目前不支持在 Windows 核心设备上使用此功能。

- --max-log-queue-size，-m。 分配给内存的最大日志条目数。使用此选项可以优化内存使用情况。默认值为 100。

输出

以下示例显示了运行此命令时生成的输出。

```
$ sudo greengrass-cli logs get --verbose \
    --log-file /greengrass/v2/logs/greengrass.log \
    --filter deployment,serviceName=DeploymentService \
    --filter level=INFO \
    --time-window 2020-12-08T01:11:17,2020-12-08T01:11:22

2020-12-08T01:11:17.615Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.DeploymentService: Current deployment finished.
 {DeploymentId=44d89f46-1a29-4044-ad89-5151213dfcbc, serviceName=DeploymentService,
 currentState=RUNNING}
2020-12-08T01:11:17.675Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.IotJobsHelper: Updating status of persisted
 deployment. {Status=SUCCEEDED, StatusDetails={detailed-deployment-
status=SUCCESSFUL}, ThingName=MyThing, JobId=22d89f46-1a29-4044-ad89-5151213dfcbc
```

列表关键词

显示可用于筛选日志文件的建议关键字。

摘要

```
greengrass-cli logs list-keywords [arguments]
```

Arguments (参数)

无

输出

以下示例显示了运行此命令时产生的输出。

```
$ sudo greengrass-cli logs list-keywords

Here is a list of suggested keywords for Greengrass log:
level=$str
thread=$str
loggerName=$str
```

```
eventType=$str
serviceName=$str
error=$str
```

```
$ sudo greengrass-cli logs list-keywords --syslog

Here is a list of suggested keywords for syslog:
priority=$int
host=$str
logger=$str
class=$str
```

list-log-files

显示位于指定目录中的日志文件。

摘要

```
greengrass-cli logs list-log-files [arguments]
```

Arguments (参数)

`--log-dir`，`-ld`。 用于检查日志文件的目录路径。

输出

以下示例显示了运行此命令时生成的输出。

```
$ sudo greengrass-cli logs list-log-files -ld /greengrass/v2/logs/

/greengrass/v2/logs/aws.greengrass.Nucleus.log
/greengrass/v2/logs/main.log
/greengrass/v2/logs/greengrass.log
Total 3 files found.
```

get-debug-password

使用`get-debug-password`命令打印随机生成的密码本地调试控制台组件(`aws.greengrass.LocalDebugConsole`)。密码在生成 8 小时后过期。

摘要

```
greengrass-cli get-debug-password
```

Arguments (参数)

None（无）

输出

以下示例显示了运行此命令时生成的输出。

```
$ sudo greengrass-cli get-debug-password

Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is self-
signed so you will need to bypass your web browser's security warnings to open the
 console.
Before you bypass the security warning, verify that the certificate fingerprint
 matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67 96
 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

使用 AWS IoT Greengrass 测试框架

Greengrass 测试框架 (GTF) 是一系列从客户角度支持自动化的构建模块。 end-to-endGTF 使用 Cuc u mber 作为功能驱动程序。 AWS IoT Greengrass 使用相同的构造块来验证不同设备上的软件更改。欲了解更多信息，请参阅 Github 上的 Greengrass 测试框架。

GTF 是使用 Cucumber（一种用于运行自动测试的工具）实现的，以鼓励组件的行为驱动开发 (BDD)。在 Cucumber 中，该系统的功能在一种名为的特殊文件类型中概述`feature`。每个功能都以一种人类可读的格式描述，称为场景，这些规范可以转换为自动测试。每个场景都被概述为一系列步骤，这些步骤使用一种名为Gherkin的特定领域语言来定义被测系统的交互和结果。使用一种名为步骤定义的方法将 Gherkin 步骤链接到编程代码，该方法将规范与测试流程硬连线。GTF 中的步骤定义是用 Java 实现的。

主题

- 工作方式

- 更改日志
- Greengrass 测试框架配置选项
- 教程：使用 Greengrass end-to-end 测试框架和 Greengrass 开发套件运行测试
- 教程：使用置信度测试套件中的置信度测试

工作方式

AWS IoT Greengrass 将 GTF 作为由多个 Java 模块组成的独立 JAR 分发。要使用 GTF end-to-end 测试组件，必须在 Java 项目中实现测试。将测试可用 Jar 作为依赖项添加到 Java 项目中，可以使用 GTF 的现有功能，并通过编写自己的自定义测试用例对其进行扩展。要运行自定义测试用例，您可以构建 Java 项目并使用中所述的配置选项运行目标 JAR Greengrass 测试框架配置选项。

GTF 独立版 JAR

Greengrass 使用 Cloudfront 作为 Maven 存储库来托管不同版本的 GTF 独立 JAR。有关 GTF 版本的完整列表，请参阅 GTF 版本。

GTF 独立 JAR 包括以下模块。它不仅限于这些模块。您可以在项目中单独选择每个依赖项，也可以将所有这些依赖项同时包含在测试的独立 JAR 文件中。

- `aws-greengrass-testing-resources`：此模块为在测试过程中管理 AWS 资源的生命周期提供了抽象。你可以用它来使用`ResourceSpec`抽象来定义你的自定义 AWS 资源，这样 GTF 就可以为你创建和删除这些资源。
- `aws-greengrass-testing-platform`：此模块为测试生命周期中的被测设备提供平台级抽象。它包含用于与独立于平台的操作系统交互的 API，并可用于模拟在设备外壳中运行的命令。
- `aws-greengrass-testing-components`：此模块由用于测试 Greengrass 核心功能（例如部署、IPC 和其他功能）的示例组件组成。
- `aws-greengrass-testing-features`：此模块由可重复使用的常用步骤及其定义组成，用于在 Greengrass 环境中进行测试。

主题

- 更改日志
- Greengrass 测试框架配置选项
- 教程：使用 Greengrass end-to-end 测试框架和 Greengrass 开发套件运行测试
- 教程：使用置信度测试套件中的置信度测试

更改日志

下表描述了每个版本的 GTF 中的更改。如需了解更多信息，请参阅上的 GTF 版本页面。 GitHub

版本	更改
1.2.0	新功能 • 添加与网络相关的步骤，以便在测试期间配置 MQTT 和互联网网络连接。 • 添加系统指标步骤以监控设备 RAM 和 CPU 使用情况。 错误修复和改进 • Greengrass CLI 本地部署步骤会重试，直到成功为止。 • 测试可以优雅地阻止 Greengrass 核，而不是杀死它。 • 增加了改进，即 GTF 轮询 AWS IoT 凭证端点，直到可以检索事物和角色别名的凭证。 • 修复了缺失的工件和配方目录。此版本还修复了缺失的组件版本。 • 修复了在 docker 镜像清理 docker 镜像期间，如果 docker 镜像不存在，GTF 会失败的问题。 • 将 CURRENT 关键字添加为组件的版本。
1.1.0	新功能 • 添加了安装带配置的自定义组件的功能。这需要自定义组件的配方。 • 添加了使用自定义配置更新本地部署的功能。 错误修复和改进 • 修复了日志上下文 GTF 版本不一致的问题。
1.0.0	初始版本。

Greengrass 测试框架配置选项

GTF 配置选项

Greengrass 测试框架 (GTF) 允许您在启动期间配置某些参数 end-to-end 用于协调测试流程的测试流程。您可以将这些配置选项指定为 GTF 独立 JAR 的 CLI 参数。

GTF 1.1.0 及更高版本提供了以下配置选项。

- `additional-plugins`—（可选）其他 Cumber 插件
- `aws-region`— 针对特定的区域终端节点AWS服务。默认为AWSSDK 发现了。
- `credentials-path`— 可选AWS配置文件凭据路径。默认为在主机环境中发现的凭据。
- `credentials-path-rotation`— 可选的轮换持续时间AWS证书。默认为 15 分钟或`PT15M`。
- `csr-path`— 用于生成设备证书的 CSR 路径。
- `device-mode`— 正在测试的目标设备。默认为本地设备。
- `env-stage`— 瞄准 Greengrass 的部署环境。默认为生产。
- `existing-device-cert-arn`— 要用作 Greengrass 设备证书的现有证书的 arn。
- `feature-path`— 包含其他功能文件的文件或目录。默认为不使用其他功能文件。
- `gg-cli-version`— 覆盖 Greengrass CLI 的版本。默认为中找到的值`ggc.version`。
- `gg-component-bucket`— 存放 Greengrass 组件的现有 Amazon S3 存储桶的名称。
- `gg-component-overrides`— Greengrass 组件重写列表。
- `gg-persist`— 测试运行后要保留的测试元素列表。默认行为是不保留任何内容。可接受的值为：`aws.resources`，`installed.software`，以及`generated.files`。
- `gg-runtime`— 影响测试与测试资源交互方式的值列表。这些值取代了`gg.persist`参数。如果默认值为空，则假设所有测试资源都由测试用例管理，包括已安装的 Greengrass 运行时。可接受的值为：`aws.resources`，`installed.software`，以及`generated.files`。
- `ggc-archive`— 存档的 Greengrass nucleus 组件的路径。
- `ggc-install-root`— 用于安装 Greengrass nucleus 组件的目录。默认为 test.temp.path 和测试运行文件夹。
- `ggc-log-level`— 设置测试运行的 Greengrass nucleus 日志级别。默认为 “信息”。
- `ggc-tes-rolename`— 那个 IAM 角色AWS IoT GreengrassCore 将假设可以访问AWS服务。如果不存在具有给定名称的角色，则将创建一个默认访问策略。
- `ggc-trusted-plugins`— 需要添加到 Greengrass 的可信插件的路径（主机上）的逗号分隔列表。要提供 DUT 本身的路径，请在路径前面加上 'dut: '
- `ggc-user-name`— Greengrass 核的 user: group posixUser 值。默认为当前登录的用户名。
- `ggc-version`— 覆盖正在运行的 Greengrass nucleus 组件的版本。默认为 ggc.archive 中找到的值。
- `log-level`— 测试运行的日志级别。默认为 “信息”。

- `parallel-config`— 将批次索引和批次数设置为 JSON 字符串。批次索引的默认值为 0，批次数为 1。
- `proxy-url`— 将所有测试配置为通过此 URL 路由流量。
- `tags`— 仅运行功能标签。可以与 '&' 相交
- `test-id-prefix`— 适用于所有测试特定资源的通用前缀，包括AWS资源名称和标签。默认为 “gg” 前缀。
- `test-log-path`— 将包含整个测试运行结果的目录。默认为 “测试结果”。
- `test-results-json`— 用于确定生成的 Cucumber JSON 报告是否已生成写入磁盘的标志。默认值为 true。
- `test-results-log`— 用于确定控制台输出是否生成的写入磁盘的标志。默认值为 false。
- `test-results-xml`— 用于确定生成的 JUnit XML 报告是否已写入磁盘的标志。默认值为 true。
- `test-temp-path`— 生成本地测试工件的目录。默认为以 gg-testing 为前缀的随机临时目录。
- `timeout-multiplier`— 为所有测试超时提供乘数。默认值为 1.0。

教程：使用 Greengrass end-to-end 测试框架和 Greengrass 开发套件运行测试

AWS IoT Greengrass测试框架 (GTF) 和 Greengrass 开发套件 (GDK) 为开发人员提供了运行测试的方法。 end-to-end 完成本教程后，您可以使用组件初始化 GDK 项目，使用 end-to-end 测试模块初始化 GDK 项目，并生成自定义测试用例。生成自定义测试用例后，就可以运行测试了。

在本教程中，您将执行以下操作：

1. 使用组件初始化 GDK 项目。
2. 使用 end-to-end 测试模块初始化 GDK 项目。
3. 生成自定义测试用例。
4. 为新的测试用例添加标签。
5. 生成测试 JAR。
6. 运行 测试。

主题

- 先决条件
- 步骤 1：使用组件初始化 GDK 项目
- 步骤 2：使用 end-to-end测试模块初始化 GDK 项目

- 第 3 步：构建自定义测试用例
- 第 4 步：为新测试用例添加标签
- 第 5 步：构建测试 JAR
- 第 6 步：运行测试
- 示例：生成自定义测试用例

先决条件

要完成本教程，您需要：

- GDK 版本 1.3.0 或更高版本
- Java
- Maven
- Git

步骤 1：使用组件初始化 GDK 项目

- 使用 GDK 项目初始化一个空文件夹。运行以下命令下载用 Python 实现的HelloWorld组件。

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  此命令在当前目录HelloWorld中创建一个名为的新目录。

步骤 2：使用 end-to-end测试模块初始化 GDK 项目

- GDK 允许您下载由功能和步骤实现组成的测试模块模板。运行以下命令打开HelloWorld目录并使用测试模块初始化现有的 GDK 项目。

  ```
  cd HelloWorld
  gdk test-e2e init
  ```

  此命令将在该目录gg-e2e-tests中创建一个名为的新HelloWorld目录。这个测试目录是一个 Maven 项目，它依赖于 Greengrass 测试独立 JAR。

第 3 步：构建自定义测试用例

编写自定义测试用例大致包括两个步骤：创建包含测试场景的功能文件和实现步骤定义。有关构建自定义测试用例的示例，请参阅示例：生成自定义测试用例。使用以下步骤来构建您的自定义测试用例：

1. 使用测试场景创建要素文件

   一项功能通常描述正在测试的软件的特定功能。在 Cucumber 中，每个功能都指定为一个单独的特征文件，其中包含标题、详细描述以及一个或多个称为场景的特定案例示例。每个场景都由标题、详细描述和一系列定义互动和预期结果的步骤组成。场景以结构化格式编写，使用 “给定”、“何时” 和 “然后” 关键字。

2. 实现步骤定义

   步骤定义用通俗易懂的语言将 Gherkin 步骤链接到编程代码。当 Cucumber 在场景中识别 Gherkin 步骤时，它将寻找要运行的匹配步骤定义。

第 4 步：为新测试用例添加标签

- 您可以为功能和场景分配标签以组织测试过程。您可以使用标签对场景的子集进行分类，也可以有条件地选择要运行的挂钩。功能和场景可以有多个标签，用空格隔开。

  在这个例子中，我们使用的是组`HelloWorld`件。

  在功能文件中，在标签`@HelloWorld`旁边添加一个名为的新`@Sample`标签。

  ```
  @Sample @HelloWorld
  Scenario: As a developer, I can create a component and deploy it on my device
  ....
  ```

第 5 步：构建测试 JAR

1. 生成组件。在生成测试模块之前，必须先生成组件。

   ```
   gdk component build
   ```

2. 使用以下命令生成测试模块。此命令将在该`greengrass-build`文件夹中生成测试 JAR。

   ```
   gdk test-e2e build
   ```

第 6 步：运行测试

当您运行自定义测试用例时，GTF 会自动执行测试的生命周期，并管理在测试期间创建的资源。它首先将待测设备 (DUT) 配置为一个AWS IoT东西，然后在其上安装 Greengrass 核心软件。然后，它将HelloWorld使用该路径中指定的配方创建一个名为的新组件。然后，该HelloWorld组件通过 Greengrass 事物部署部署到核心设备上。然后将对其进行验证，以确定部署是否成功。如果部署成功，部署状态将COMPLETED在 3 分钟内更改为。

1. 转到项目目录中的gdk-config.json文件，将带有HelloWorld标签的测试作为目标。使用以下命令更新test-e2e密钥。

```
"test-e2e":{
  "gtf_options" : {
       "tags":"HelloWorld"
   }
}
```

2. 在运行测试之前，必须向主机设备提供AWS凭据。在测试过程中，GTF 使用这些凭证来管理AWS资源。确保您提供的角色有权自动执行测试中包含的必要操作。

   运行以下命令以提供AWS凭据。

   - Linux or Unix

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

     Windows Command Prompt (CMD)

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

     PowerShell

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

3. 使用以下命令运行测试。

```
gdk test-e2e run
```

此命令在文件夹中下载最新版本的 Greengrass nucleus `greengrass-build` 并使用它运行测试。此命令还仅针对带有`HelloWorld`标签的场景，并为这些场景生成报告。您将看到在此测试期间创建的AWS资源在测试结束时被丢弃。

示例：生成自定义测试用例

Example

GDK 项目中下载的测试模块由一个示例功能和一个步骤实现文件组成。

在以下示例中，我们创建了一个功能文件，用于测试 Greengrass 软件的事物部署功能。我们使用通过 Greengr AWS Cloud ass 部署组件的场景部分测试了此功能的功能。这是一系列步骤，可帮助我们了解此用例的互动和预期结果。

1. 创建要素文件

   导航到当前目录中的`gg-e2e-tests/src/main/resources/greengrass/features`文件夹。您可以找到与以下示例类似`component.feature`的示例。

   在此功能文件中，您可以测试 Greengrass 软件的事物部署功能。您可以使用通过 Greengrass 云部署组件的场景来部分测试此功能的功能。该场景是一系列步骤，有助于了解此用例的互动和预期结果。

```
Feature: Testing features of Greengrassv2 component

Background:
    Given my device is registered as a Thing
    And my device is running Greengrass

@Sample
Scenario: As a developer, I can create a component and deploy it on my device
    When I create a Greengrass deployment with components
        HelloWorld | /path/to/recipe/file
    And I deploy the Greengrass deployment configuration
    Then the Greengrass deployment is COMPLETED on the device after 180 seconds
    And I call my custom step
```

   GTF 包含以下所有步骤的步骤定义，但名为:`And I call my custom step`的步骤除外。

2. 实现步骤定义

GTF 独立 JAR 包含所有步骤的步骤定义，只有一个步骤除外:`And I call my custom step`. 你可以在测试模块中实现此步骤。

导航到测试文件的源代码。您可以使用以下命令使用步骤定义来链接您的自定义步骤。

```
@And("I call my custom step")
public void customStep() {
    System.out.println("My custom step was called ");
}
```

教程：使用置信度测试套件中的置信度测试

AWS IoT Greengrass测试框架 (GTF) 和 Greengrass 开发套件 (GDK) 为开发人员提供了运行测试的方法。 end-to-end 完成本教程后，您可以使用组件初始化 GDK 项目，使用 end-to-end 测试模块初始化 GDK 项目，以及使用置信度测试套件中的置信度测试。生成自定义测试用例后，就可以运行测试了。

置信度测试是 Greengrass 提供的通用测试，用于验证基本组件行为。可以修改或扩展这些测试以适应更具体的组件需求。

在本教程中，我们将使用一个 HelloWorld 组件。如果您使用的是其他组件，请用您的 HelloWorld 组件替换该组件。

在本教程中，您将执行以下操作：

1. 使用组件初始化 GDK 项目。
2. 使用 end-to-end 测试模块初始化 GDK 项目。
3. 使用置信度测试套件中的测试。
4. 为新的测试用例添加标签。
5. 生成测试 JAR。
6. 运行 测试。

主题

- 先决条件
- 步骤 1：使用组件初始化 GDK 项目

- 步骤 2：使用 end-to-end测试模块初始化 GDK 项目
- 步骤 3：使用置信度测试套件中的测试
- 第 4 步：为新测试用例添加标签
- 第 5 步：构建测试 JAR
- 步骤 6：运行测试
- 示例：使用置信度测试

先决条件

要完成本教程，您需要：

- GDK 版本 1.6.0 或更高版本
- Java
- Maven
- Git

步骤 1：使用组件初始化 GDK 项目

- 使用 GDK 项目初始化一个空文件夹。运行以下命令下载用 Python 实现的`HelloWorld`组件。

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  此命令在当前目录`HelloWorld`中创建一个名为的新目录。

步骤 2：使用 end-to-end测试模块初始化 GDK 项目

- GDK 允许您下载由功能和步骤实现组成的测试模块模板。运行以下命令打开`HelloWorld`目录并使用测试模块初始化现有的 GDK 项目。

  ```
  cd HelloWorld
  gdk test-e2e init
  ```

  此命令在目录`gg-e2e-tests`中创建一个名为的新`HelloWorld`目录。这个测试目录是一个 Maven 项目，它依赖于 Greengrass 测试独立 JAR。

步骤 3：使用置信度测试套件中的测试

编写置信度测试用例包括使用提供的功能文件，并在需要时修改场景。有关使用置信度测试的示例，请参阅示例：生成自定义测试用例。使用以下步骤使用置信度测试：

- 使用提供的功能文件。

  导航到当前目录中的`gg-e2e-tests/src/main/resources/greengrass/features`文件夹。打开示例`confidenceTest.feature`文件以使用置信度测试。

第 4 步：为新测试用例添加标签

- 您可以为功能和场景分配标签以组织测试过程。您可以使用标签对场景的子集进行分类，也可以有条件地选择要运行的挂钩。功能和场景可以有多个标签，用空格隔开。

  在这个例子中，我们使用的是组`HelloWorld`件。

  每个场景都标有`@ConfidenceTest`。如果您只想运行测试套件的子集，请更改或添加标签。每个测试场景都在每个置信度测试的顶部进行描述。该场景是一系列步骤，有助于了解每个测试用例的交互作用和预期结果。您可以通过添加自己的步骤或修改现有步骤来扩展这些测试。

```
@ConfidenceTest
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
 is working as expected
....
```

第 5 步：构建测试 JAR

1. 生成组件。在生成测试模块之前，必须先生成组件。

```
gdk component build
```

2. 使用以下命令生成测试模块。此命令将在该`greengrass-build`文件夹中生成测试 JAR。

```
gdk test-e2e build
```

步骤 6：运行测试

当您运行置信度测试时，GTF 会自动执行测试的生命周期并管理测试期间创建的资源。它首先将待测设备 (DUT) 配置为一个AWS IoT东西，然后在其上安装 Greengrass 核心软件。然后，它将HelloWorld使用该路径中指定的配方创建一个名为的新组件。然后，该HelloWorld组件通过 Greengrass 事物部署部署到核心设备上。然后将验证部署是否成功。如果部署成功，部署状态将COMPLETED在 3 分钟内更改为。

1. 转到项目目录中的gdk-config.json文件，使用ConfidenceTest标签或步骤 4 中指定的任何标签 yo8u 来定位测试。使用以下命令更新test-e2e密钥。

```
"test-e2e":{
  "gtf_options" : {
       "tags":"ConfidenceTest"
   }
}
```

2. 在运行测试之前，必须向主机设备提供AWS凭据。在测试过程中，GTF 使用这些凭证来管理AWS资源。确保您提供的角色有权自动执行测试中包含的必要操作。

   运行以下命令以提供AWS凭据。

   - Linux or Unix

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

     Windows Command Prompt (CMD)

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

     PowerShell

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

3. 使用以下命令运行测试。

```
gdk test-e2e run
```

此命令在文件夹中下载最新版本的 Greengrass nucleus `greengrass-build` 并使用它运行测试。此命令还仅针对带有`ConfidenceTest`标签的场景，并为这些场景生成报告。您将看到在此测试期间创建的AWS资源在测试结束时被丢弃。

示例：使用置信度测试

Example

GDK 项目中下载的测试模块由提供的功能文件组成。

在以下示例中，我们使用功能文件来测试 Greengrass 软件的事物部署功能。我们使用通过 Greengr AWS Cloud ass 部署组件的场景部分测试了此功能的功能。这是一系列步骤，可帮助我们了解此用例的互动和预期结果。

- 使用提供的功能文件。

  导航到当前目录中的`gg-e2e-tests/src/main/resources/greengrass/features`文件夹。您可以找到与以下示例类似`confidenceTest.feature`的示例。

```
Feature: Confidence Test Suite

Background:
    Given my device is registered as a Thing
    And my device is running Greengrass

@ConfidenceTest
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
 is working as expected
    When I create a Greengrass deployment with components
      | GDK_COMPONENT_NAME | GDK_COMPONENT_RECIPE_FILE |
      | aws.greengrass.Cli | LATEST                    |
    And I deploy the Greengrass deployment configuration
    Then the Greengrass deployment is COMPLETED on the device after 180 seconds
    # Update component state accordingly. Possible states: {RUNNING, FINISHED,
 BROKEN, STOPPING}
    And I verify the GDK_COMPONENT_NAME component is RUNNING using the greengrass-
cli
```

每个测试场景都在每个置信度测试的顶部进行描述。该场景是一系列步骤，有助于了解每个测试用例的交互作用和预期结果。您可以通过添加自己的步骤或修改现有步骤来扩展这些测试。每个场景都包含可帮助您进行这些调整的注释。

开发AWS IoT Greengrass组件

您可以在 Greengrass 核心设备上开发和测试组件。因此，您无需与交互即可创建和迭代AWS IoT Greengrass软件。AWS Cloud完成组件版本后，可以将其上传到AWS IoT Greengrass云端，这样您和您的团队就可以将该组件部署到队列中的其他设备上。有关如何部署组件的更多信息，请参阅将AWS IoT Greengrass组件部署到设备。

每个组件都由配方和工件组成。

- 食谱

  每个组件都包含一个用于定义其元数据的配方文件。该配方还指定了组件的配置参数、组件依赖关系、生命周期和平台兼容性。组件生命周期定义了安装、运行和关闭组件的命令。有关更多信息，请参阅 AWS IoT Greengrass组件配方参考。

  你可以用 JSON 或 YAML 格式定义配方。
- 构件

  组件可以有任意数量的工件，即组件二进制文件。构件可以包括脚本、编译后的代码、静态资源以及组件消耗的任何其他文件。组件还可以使用组件依赖项中的工件。

AWS IoT Greengrass提供了预先构建的组件，您可以在应用程序中使用这些组件并部署到您的设备上。例如，您可以使用流管理器组件将数据上传到各种AWS服务，也可以使用 CloudWatch 指标组件将自定义指标发布到 Amazon CloudWatch。有关更多信息，请参阅 AWS-提供的组件。

AWS IoT Greengrass策划了一份名为 Greengrass 软件目录的 Greengrass 组件索引。该目录追踪了 Greengrass 社区开发的 Greengrass 组件。您可以从该目录中下载、修改和部署组件来创建 Greengrass 应用程序。有关更多信息，请参阅 社区组件。

AWS IoT GreengrassCore 软件以系统用户和组的身份运行您在核心设备上配置的组件，例如`ggc_user`和`ggc_group`。这意味着组件拥有该系统用户的权限。如果您使用没有主目录的系统用户，则组件不能使用使用主目录的运行命令或代码。例如，这意味着你不能使用`pip install some-library --user`命令来安装 Python 软件包。如果您按照入门教程设置核心设备，则您的系统用户没有主目录。有关如何配置运行组件的用户和组的更多信息，请参阅配置运行组件的用户。

AWS IoT Greengrass使用组件的语义版本。语义版本遵循 major.minor.patch 编号系统。例如，版本1.0.0表示组件的第一个主要版本。有关更多信息，请参阅语义版本规范。

主题

- 组件生命周期
- 组件类型
- 创建 AWS IoT Greengrass 组件
- 使用本地部署测试AWS IoT Greengrass组件
- 发布组件以部署到您的核心设备
- 与AWS服务互动
- 运行 Docker 容器
- AWS IoT Greengrass组件配方参考
- 组件环境变量引用

组件生命周期

组件生命周期定义了AWS IoT Greengrass核心软件用于安装和运行组件的阶段。每个阶段都定义一个脚本和其他信息，用于指定组件的行为。例如，在安装组件时，AWS IoT GreengrassCore 软件会运行该组件的Install生命周期脚本。核心设备上的组件具有以下生命周期状态：

- NEW— 组件的配方和工件已加载到核心设备上，但未安装该组件。组件进入此状态后，它会运行其安装脚本。
- INSTALLED— 组件安装在核心设备上。组件在运行安装脚本后进入此状态。
- STARTING— 组件正在核心设备上启动。组件在运行其启动脚本时会进入此状态。如果启动成功，则组件进入RUNNING状态。
- RUNNING— 该组件正在核心设备上运行。当组件运行其运行脚本或启动脚本中有活跃的后台进程时，它就会进入此状态。
- FINISHED— 组件成功运行并完成运行。
- STOPPING— 组件正在停止。组件在运行其关闭脚本时会进入此状态。

- ERRORED— 组件遇到了错误。当组件进入此状态时，它会运行其恢复脚本。然后，组件重新启动以尝试恢复正常使用。如果组件三次进入ERRORED状态但未成功运行，则该组件变为BROKEN。
- BROKEN— 该组件多次遇到错误并且无法恢复。必须重新部署该组件才能对其进行修复。

组件类型

组件类型指定 AWS IoT Greengrass Core 软件如何运行组件。组件可以有以下类型：

- Nucleus () `aws.greengrass.nucleus`

  Greengrass 核心是提供核心软件最低功能的组件。AWS IoT Greengrass有关更多信息，请参阅 Greengrass 核。
- 插件 (`aws.greengrass.plugin`)

  Greengrass nucleus 在与核心相同的 Java 虚拟机 (JVM) 中运行插件组件。当你在核心设备上更改插件组件的版本时，nucleus 会重新启动。要安装和运行插件组件，必须将 Greengrass nucleus 配置为作为系统服务运行。有关更多信息，请参阅 将 Greengrass 核心配置为系统服务。

  提供的AWS几个组件是插件组件，这使它们能够直接与 Greengrass 核接口。插件组件使用与 Greengrass 核心相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。
- 通用 (`aws.greengrass.generic`)

  如果通用组件定义了生命周期，Greengrass nucleus 就会运行该组件的生命周期脚本。

  此类型是自定义组件的默认类型。
- Lambda () `aws.greengrass.lambda`

  Greengrass nucleus 使用 Lambda 启动器组件运行 Lambda 函数组件。

  当您通过 Lambda 函数创建组件时，该组件具有这种类型。有关更多信息，请参阅 运行AWS Lambda函数。

> **Note**
>
> 我们不建议您在配方中指定组件类型。 AWS IoT Greengrass在创建组件时为您设置类型。

创建 AWS IoT Greengrass 组件

您可以在本地开发计算机或 Greengrass 核心设备上开发自定义 AWS IoT Greengrass 组件。 AWS IoT Greengrass 提供了AWS IoT Greengrass 开发套件命令行界面 (GDK CLI)，可帮助您根据预定义的组件模板和社区组件创建、生成和发布组件。您还可以运行内置的 shell 命令来创建、生成和发布组件。从以下选项中进行选择以创建自定义 Greengrass 组件：

- 使用 Greengrass 开发套件 CLI

  使用 GDK CLI 在本地开发计算机上开发组件。GDK CLI 构建组件源代码并将其打包为配方和工件，您可以将其作为私有组件发布到该 AWS IoT Greengrass 服务。您可以将 GDK CLI 配置为在发布组件时自动更新组件的版本和工件 URI，因此无需每次都更新配方。要使用 GDK CLI 开发组件，可以从 Gre engrass 软件目录中的模板或社区组件开始。有关更多信息，请参阅AWS IoT Greengrass开发套件命令行界面。
- 运行内置 shell 命令

  您可以运行内置的 shell 命令在本地开发计算机或 Greengrass 核心设备上开发组件。您可以使用 shell 命令将组件源代码复制或构建到工件中。每次创建组件的新版本时，都必须使用新的组件版本创建或更新配方。将组件发布到 AWS IoT Greengrass 服务时，必须更新配方中每个组件工件的 URI。

主题

- 创建组件 (GDK CLI)
- 创建组件（外壳命令）

创建组件 (GDK CLI)

按照本节中的说明使用 GDK CLI 创建和构建组件。

开发 Greengrass 组件 (GDK CLI)

1. 如果尚未安装 GDK CLI，请在开发计算机上安装 GDK CLI。有关更多信息，请参阅安装或更新 AWS IoT Greengrass开发套件命令行界面。
2. 切换到要在其中创建组件文件夹的文件夹。

   Linux or Unix

   ```
   mkdir ~/greengrassv2
   ```

```
cd ~/greengrassv2
```

Windows Command Prompt (CMD)

```
mkdir %USERPROFILE%\greengrassv2
cd %USERPROFILE%\greengrassv2
```

PowerShell

```
mkdir ~/greengrassv2
cd ~/greengrassv2
```

3. 选择要下载的组件模板或社区组件。GDK CLI 下载模板或社区组件，因此您可以从功能示例开始。使用组件列表命令检索可用模板或社区组件的列表。

   - 要列出组件模板，请运行以下命令。响应中的每一行都包含模板的名称和编程语言。

     ```
     gdk component list --template
     ```

   - 要列出社区组件，请运行以下命令。

     ```
     gdk component list --repository
     ```

4. 创建并更改为 GDK CLI 下载模板或社区组件的组件文件夹。*HelloWorld*替换为组件的名称或其他可帮助您识别此组件文件夹的名称。

   Linux or Unix

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

   PowerShell

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

5. 将模板或社区组件下载到当前文件夹。使用组件 init 命令。

   - 要使用模板创建组件文件夹，请运行以下命令。*HelloWorld*替换为模板的名称，并将 *python* 替换为编程语言的名称。

     ```
     gdk component init --template HelloWorld --language python
     ```

   - 要从社区组件创建组件文件夹，请运行以下命令。*ComponentName*替换为社区组件的名称。

     ```
     gdk component init --repository ComponentName
     ```

   > **Note**
   >
   > 如果您使用 GDK CLI v1.0.0，则必须在空文件夹中运行此命令。GDK CLI 会将模板或社区组件下载到当前文件夹。
   > 如果您使用 GDK CLI v1.1.0 或更高版本，则可以指定`--name`参数来指定 GDK CLI 下载模板或社区组件的文件夹。如果使用此参数，请指定一个不存在的文件夹。GDK CLI 会为您创建文件夹。如果您未指定此参数，GDK CLI 将使用当前文件夹，该文件夹必须为空。

6. GDK CLI 从名为 GDK CLI 的配置文件中读取数据`gdk-config.json`，以构建和发布组件。此配置文件存在于组件文件夹的根目录中。上一步将为您创建此文件。在此步骤中，您将`gdk-config.json`使用有关组件的信息进行更新。执行以下操作：

   a. 在文本编辑器中打开 `gdk-config.json`。
   b. （可选）更改组件的名称。组件名称是`component`对象中的关键。
   c. 更改组件的作者。
   d. （可选）更改组件的版本。指定下列项之一：

      - `NEXT_PATCH`— 当您选择此选项时，GDK CLI 将在您发布组件时设置版本。GDK CLI 会查询 AWS IoT Greengrass 服务以识别该组件的最新发布版本。然后，它将版本设置为该版本之后的下一个补丁版本。如果您之前没有发布过该组件，GDK CLI 将使用版本`1.0.0`。

        如果选择此选项，则无法使用 Greengrass CLI 在本地部署该组件并将其测试到运行 Core 软件的本地开发计算机上。 AWS IoT Greengrass 要启用本地部署，必须改为指定语义版本。
      - 语义版本，例如。**`1.0.0`**语义版本使用主调。 未成年人。 补丁编号系统。有关更多信息，请参阅语义版本规范。

如果您在 Greengrass 核心设备上开发要部署和测试该组件的组件，请选择此选项。要使用 Greengr ass CLI 创建本地部署，必须使用特定版本构建组件。

e. （可选）更改组件的编译配置。构建配置定义了 GDK CLI 如何将组件的源代码构建为工件。从以下选项中进行选择build_system：

- zip— 将组件的文件夹打包为 ZIP 文件以定义为该组件的唯一构件。为以下类型的组件选择此选项：
  - 使用解释型编程语言的组件，例如 Python 或 JavaScript。
  - 打包除代码之外的文件的组件，例如机器学习模型或其他资源。

  GDK CLI 将组件的文件夹压缩为与组件文件夹同名的 zip 文件。例如，如果组件文件夹的名称为HelloWorld，则 GDK CLI 会创建一个名HelloWorld.zip为的 zip 文件。

  > **Note**
  >
  > 如果您在 Windows 设备上使用 GDK CLI 版本 1.0.0，则组件文件夹和 zip 文件名必须仅包含小写字母。

  当 GDK CLI 将组件的文件夹压缩为 zip 文件时，它会跳过以下文件：
  - gdk-config.json 文件
  - 配方文件（recipe.json或recipe.yaml）
  - 生成文件夹，例如 greengrass-build
- maven— 运行mvn clean package命令将组件的源代码构建为工件。对于使用 Mave n 的组件（例如 Java 组件），请选择此选项。

  在 Windows 设备上，此功能适用于 GDK CLI v1.1.0 及更高版本。
- gradle— 运行gradle build命令将组件的源代码构建为工件。对于使用 Gradle 的组件，请选择此选项。此功能适用于 GDK CLI 版本 1.1.0 及更高版本。

  编gradle译系统支持 Kotlin DSL 作为构建文件。此功能适用于 GDK CLI 版本 1.2.0 及更高版本。
- gradlew— 运行gradlew命令将组件的源代码构建为工件。对于使用 Gradle 包装器的组件，请选择此选项。

  此功能适用于 GDK CLI 版本 1.2.0 及更高版本。

- custom— 运行自定义命令将组件的源代码构建为配方和工件。在custom_build_command参数中指定自定义命令。

f. 如果您custom为指定build_system，请将custom_build_command添加到build对象中。在中custom_build_command，指定单个字符串或字符串列表，其中每个字符串都是命令中的一个单词。例如，要为 C++ 组件运行自定义生成命令，可以指定**["cmake", "--build", "build", "--config", "Release"]**。

g. 如果您使用 GDK CLI v1.1.0 或更高版本，则可以指定--bucket参数来指定 GDK CLI 上传组件工件的 S3 存储桶。*##########GDK CLI #######bucket-region-accountId## ##########gdk-config.json#####accountID ### ID#* AWS 账户 如果存储桶不存在，GDK CLI 会创建该存储桶。

   更改组件的发布配置。执行以下操作：

   i. 指定用于托管组件工件的 S3 存储桶的名称。

   ii. 指定 GDK CLI 发布组件 AWS 区域 的位置。

完成此步骤后，gdk-config.json文件可能与以下示例类似。

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

7. 更新名为recipe.yaml或的组件配方文件recipe.json。执行以下操作：

a. 如果您下载了使用编zip译系统的模板或社区组件，请检查 zip 工件名称是否与组件文件夹的名称相匹配。GDK CLI 将组件文件夹压缩为与组件文件夹同名的 zip 文件。该

配方在组件工件列表和使用 zip 构件中文件的生命周期脚本中包含 zip 工件名称。更新Artifacts和Lifecycle定义，使 zip 文件名与组件文件夹的名称相匹配。以下部分配方示例突出显示了Artifacts和Lifecycle定义中的 zip 文件名。

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

b. （可选）更新组件描述、默认配置、构件、生命周期脚本和平台支持。有关更多信息，请参阅AWS IoT Greengrass组件配方参考。

完成此步骤后，配方文件可能与以下示例类似。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "{COMPONENT_NAME}",
  "ComponentVersion": "{COMPONENT_VERSION}",
  "ComponentDescription": "This is a simple Hello World component written in
 Python.",
  "ComponentPublisher": "{COMPONENT_AUTHOR}",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "World"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: "2020-01-25"
```

```
ComponentName: "{COMPONENT_NAME}"
ComponentVersion: "{COMPONENT_VERSION}"
ComponentDescription: "This is a simple Hello World component written in
 Python."
ComponentPublisher: "{COMPONENT_AUTHOR}"
ComponentConfiguration:
  DefaultConfiguration:
    Message: "World"
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

8. 开发和构建 Greengrass 组件。组件构建命令会在组件文件夹的`greengrass-build`文件夹中生成配方和工件。运行以下命令。

```
gdk component build
```

当您准备好测试组件时，请使用 GDK CLI 将其发布到 AWS IoT Greengrass 服务。然后，您可以将该组件部署到 Greengrass 核心设备上。有关更多信息，请参阅发布组件以部署到您的核心设备。

创建组件（外壳命令）

按照本节中的说明创建包含多个组件源代码和构件的配方和构件文件夹。

开发 Greengrass 组件（外壳命令）

1. 为您的组件创建一个文件夹，其中包含用于存放配方和工件的子文件夹。在 Greengrass 核心设备上运行以下命令来创建这些文件夹并切换到组件文件夹。将 *~/greengrassv2 # %USERPROFILE%\ greengrassv2 ###############*径。

   Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

2. 使用文本编辑器创建用于定义组件元数据、参数、依赖关系、生命周期和平台功能的配方文件。在配方文件名中包含组件版本，这样您就可以确定哪个配方反映了哪个组件版本。你可以为你的食谱选择 YAML 或 JSON 格式。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 来创建文件。

   JSON

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

   YAML

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

> Note
>
> AWS IoT Greengrass 使用组件的语义版本。语义版本遵循 major.minor.patch 编号系统。例如，版本`1.0.0`表示组件的第一个主要版本。有关更多信息，请参阅语义版本规范。

3. 为您的组件定义配方。有关更多信息，请参阅AWS IoT Greengrass组件配方参考。

   您的食谱可能与以下 Hello World 示例食谱类似。

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.HelloWorld",
   ```

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

YAML

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

```
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

此配方运行一个 Hello World Python 脚本，该脚本可能与以下示例脚本类似。

```
import sys

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

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

4. 为要开发的组件版本创建文件夹。我们建议您为每个组件版本的构件使用单独的文件夹，以便可以识别每个组件版本的构件。运行以下命令。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   Important

   必须使用以下格式作为构件文件夹路径。包括您在配方中指定的组件名称和版本。

```
artifacts/componentName/componentVersion/
```

5. 在上一步中创建的文件夹中为您的组件创建构件。构件可以包括软件、图像和组件使用的任何其他二进制文件。

   组件准备就绪后，测试您的组件。

使用本地部署测试AWS IoT Greengrass组件

如果您在核心设备上开发 Greengrass 组件，则可以创建本地部署来安装和测试它。按照本节中的步骤创建本地部署。

如果您在另一台计算机（例如本地开发计算机）上开发组件，则无法创建本地部署。而是将该组件发布到AWS IoT Greengrass服务，这样您就可以将其部署到 Greengrass 核心设备上进行测试。有关更多信息，请参阅 发布组件以部署到您的核心设备和 将AWS IoT Greengrass组件部署到设备。

在 Greengrass 核心设备上测试组件

1. 核心设备记录诸如组件更新之类的事件。您可以查看此日志文件来发现组件中的错误，并对其进行故障排除，例如配方无效。此日志文件还显示您的组件打印到标准输出 (stdout) 的消息。我们建议您在核心设备上再打开一个终端会话，以便实时观察新的日志消息。打开新的终端会话（例如通过SSH），然后运行以下命令查看日志。*/greengrass/v2*替换为AWS IoT Greengrass根文件夹的路径。

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/greengrass.log
   ```

   PowerShell

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

   您还可以查看组件的日志文件。

Linux or Unix

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

PowerShell

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

2. 在最初的终端会话中，运行以下命令使用您的组件更新核心设备。*/greengrass/v2*替换为AWS IoT Greengrass根文件夹的路径，将 *~/greengrassv2* 替换为本地开发文件夹的路径。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   > **Note**
   >
   > 您也可以使用`greengrass-cli deployment create`命令来设置组件配置参数的值。有关更多信息，请参阅 创建。

3. 使用`greengrass-cli deployment status`命令监视组件的部署进度。

Unix or Linux

```
sudo /greengrass/v2/bin/greengrass-cli deployment status \
  -i deployment-id
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment status ^
  -i deployment-id
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment status `
  -i deployment-id
```

4. 在组件在 Greengrass 核心设备上运行时对其进行测试。完成此版本的组件后，可以将其上传到AWS IoT Greengrass服务。然后，可以将组件部署到其他核心设备。有关更多信息，请参阅 发布组件以部署到您的核心设备。

发布组件以部署到您的核心设备

在构建或完成组件版本后，可以将其发布到AWS IoT Greengrass服务。然后，你可以将其部署到Greengrass 核心设备上。

如果您使用 G reengrass 开发套件 CLI (GDK CLI) 开发和构建组件，则可以使用 GDK CLI 将该组件发布到。AWS Cloud否则，请使用内置的 shell 命令和AWS CLI来发布组件。

您还可以使用AWS CloudFormation从模板创建组件和其他AWS资源。有关更多信息，请参阅什么是AWS CloudFormation？ 以及AWS::GreengrassV2::ComponentVersion在《AWS CloudFormation用户指南》中。

主题

- 发布组件 (GDK CLI)
- 发布组件（外壳命令）

发布组件 (GDK CLI)

按照本节中的说明使用 GDK CLI 发布组件。GDK CLI 将构建项目上传到 S3 存储桶，更新配方中的构件 URI，并根据配方创建组件。您可以在 GDK CLI 配置文件中指定要使用的 S3 存储桶和区域。

如果您使用 GDK CLI v1.1.0 或更高版本，则可以指定`--bucket`参数来指定 GDK CLI 上传组件工件的 S3 存储桶。*##########GDK CLI #######bucket-region-accountId###########gdk-config.json#####accountID ### ID#* AWS 账户如果存储桶不存在，GDK CLI 会创建该存储桶。

> **Important**
>
> 默认情况下，核心设备角色不允许访问 S3 存储桶。如果这是您首次使用此 S3 存储桶，则必须向该角色添加权限，以允许核心设备从此 S3 存储桶中检索组件工件。有关更多信息，请参阅 允许访问 S3 存储桶以获取组件项目。

发布 Greengrass 组件 (GDK CLI)

1. 在命令提示符或终端中打开组件文件夹。
2. 如果你还没有，请构建 Greengrass 组件。组件构建命令会在组件文件夹的`greengrass-build`文件夹中生成配方和工件。运行以下命令。

```
gdk component build
```

3. 将该组件发布到AWS Cloud。组件发布命令将组件的构件上传到 Amazon S3，并使用每个构件的 URI 更新组件的配方。然后，它会在AWS IoT Greengrass服务中创建组件。

> **Note**
>
> AWS IoT Greengrass在创建组件时计算每个工件的摘要。这意味着在创建组件后，您无法修改 S3 存储桶中的项目文件。如果这样做，则包含此组件的部署将失败，因为文件摘要不匹配。如果修改构件文件，则必须创建该组件的新版本。

如果您在 GDK CLI 配置文件中`NEXT_PATCH`为组件版本指定，GDK CLI 将使用服务中尚不存在的下一个补丁版本。AWS IoT Greengrass

运行以下命令。

```
gdk component publish
```

输出会告诉您 GDK CLI 创建的组件的版本。

发布组件后，您可以将该组件部署到核心设备。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。

发布组件（外壳命令）

使用以下步骤使用 shell 命令和 AWS Command Line Interface (AWS CLI) 发布组件。发布组件时，需要执行以下操作：

1. 将组件项目发布到 S3 存储桶。
2. 将每个工件的 Amazon S3 URI 添加到组件配方中。
3. AWS IoT Greengrass从组件配方中创建组件版本。

> **Note**
> 您上传的每个组件版本都必须是唯一的。请务必上传正确的组件版本，因为上传后无法对其进行编辑。

您可以按照以下步骤从开发计算机或 Greengrass 核心设备发布组件。

发布组件（shell 命令）

1. 如果组件使用AWS IoT Greengrass服务中存在的版本，则必须更改该组件的版本。在文本编辑器中打开食谱，增加版本并保存文件。选择反映您对组件所做更改的新版本。

   > **Note**
   > AWS IoT Greengrass使用组件的语义版本。语义版本遵循 major.minor.patch 编号系统。例如，版本`1.0.0`表示组件的第一个主要版本。有关更多信息，请参阅语义版本规范。

2. 如果您的组件有工件，请执行以下操作：

   a. 将组件的构件发布到您的 S3 存储桶中AWS 账户。

Tip

我们建议您将组件名称和版本包含在 S3 存储桶中工件的路径中。此命名方案可以帮助您维护先前版本的组件所使用的构件，这样您就可以继续支持以前的组件版本。

运行以下命令将项目文件发布到 S3 存储桶。*# DOC-EXAMPLE-BUCKET ############# artifacts/com.example# HelloWorld/1.0.0/artifact.py*，其中包含工件文件的路径。

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

Important

默认情况下，核心设备角色不允许访问 S3 存储桶。如果这是您首次使用此 S3 存储桶，则必须向该角色添加权限，以允许核心设备从此 S3 存储桶中检索组件工件。有关更多信息，请参阅 允许访问 S3 存储桶以获取组件项目。

b. 如果组件配方中不存在名`Artifacts`为的列表，请将其添加到该列表中。该`Artifacts`列表显示在每个清单中，其中定义了组件在其支持的每个平台上的要求（或该组件对所有平台的默认要求）。

c. 将每个构件添加到构件列表中，或更新现有构件的 URI。Amazon S3 URI 由存储桶名称和存储桶中项目对象的路径组成。您的工件的 Amazon S3 URI 应与以下示例类似。

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

完成这些步骤后，您的食谱应有一个如下所示的`Artifacts`列表。

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        ...
```

```
        },
        "Artifacts": [
          {
            "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/MyGreengrassComponent/1.0.0/
artifact.py",
            "Unarchive": "NONE"
          }
        ]
      }
    ]
}
```

Note

您可以为 ZIP 构件添加`"Unarchive": "ZIP"`选项，以将 AWS IoT Greengrass Core 软件配置为在组件部署时解压缩工件。

YAML

```
...
Manifests:
  - Lifecycle:
      ...
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/MyGreengrassComponent/1.0.0/
artifact.py
        Unarchive: NONE
```

Note

您可以使用该`Unarchive: ZIP`选项将 AWS IoT Greengrass Core 软件配置为在部署组件时解压缩 ZIP 工件。有关如何在组件中使用 ZIP 构件的更多信息，请参阅 artifacts: decompress edPath 配方变量。

有关配方的更多信息，请参阅AWS IoT Greengrass组件配方参考。

3. 使用AWS IoT Greengrass控制台从配方文件创建组件。

运行以下命令从配方文件创建组件。此命令创建组件，并将其作为私有AWS IoT Greengrass组件发布到您的中AWS 账户。将 p *ath/to/recipeFile ########*路径。

```
aws greengrassv2 create-component-version --inline-recipe fileb://path/to/
recipeFile
```

复制响应arn中的，以便在下一步中检查组件的状态。

> **Note**
>
> AWS IoT Greengrass在创建组件时计算每个工件的摘要。这意味着在创建组件后，您无法修改 S3 存储桶中的项目文件。如果这样做，则包含此组件的部署将失败，因为文件摘要不匹配。如果修改构件文件，则必须创建该组件的新版本。

4. AWS IoT Greengrass服务中的每个组件都有一个状态。运行以下命令以确认您在此过程中发布的组件版本的状态。替换 *com.example# HelloWorld*以及 *1.0.0*，其中包含要查询的组件版本。将arn替换为上一步中的 ARN。

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

该操作返回一个包含组件元数据的响应。元数据包含一个包含组件状态和任何错误（如果适用）的status对象。

当组件状态为时DEPLOYABLE，您可以将组件部署到设备上。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。

与AWS服务互动

Greengrass 核心设备使用 X.509 证书通过 TLS 双向身份验证协议进行连接。AWS IoT Core这些证书允许设备在AWS IoT没有AWS凭证的情况下进行交互，证书通常包括访问密钥 ID 和私有访问密钥。其他AWS服务需要AWS凭据而不是 X.509 证书才能在服务端点调用 API 操作。 AWS IoT Core有一个凭据提供程序，允许设备使用其 X.509 证书对请求进行身份验证。AWSAWS IoT凭证提供者使用 X.509 证书对设备进行身份验证，并以临时的有限AWS权限安全令牌的形式颁发证书。设备可以使用此令牌对任何AWS请求进行签名和身份验证。这样就无需在 Greengrass 核心设备上存储AWS凭证。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的授权直接调用AWS服务。

要从 AWS IoT Greengrass 获取证书，核心设备使用指向 IAM 角色AWS IoT的角色别名。此 IAM 角色称为令牌交换角色。在安装 C AWS IoT Greengrass ore 软件时，您可以创建角色别名和令牌交换角色。要指定核心设备使用的角色别名，请配置的`iotRoleAlias`参数Greengrass 核。

AWS IoT凭证提供商代表您扮演令牌交换角色，为核心设备提供AWS凭证。您可以将适当的 IAM 策略附加到此角色，以允许您的核心设备访问您的AWS资源，例如 S3 存储桶中的组件工件。有关如何配置令牌交换角色的更多信息，请参阅授权核心设备与AWS服务。

Greengrass 核心设备将凭据AWS存储在内存中，默认情况下，凭证会在一小时后过期。如果 AWS IoT Greengrass Core 软件重新启动，则必须重新获取凭据。您可以使用该UpdateRoleAlias操作来配置凭证的有效期限。

AWS IoT Greengrass提供了一个公共组件，即令牌交换服务组件，您可以将其定义为自定义组件中的依赖项以与AWS服务进行交互。令牌交换服务为您的组件提供了一个环境变量`AWS_CONTAINER_CREDENTIALS_FULL_URI`，该变量定义了提供AWS凭据的本地服务器的 URI。创建 S AWS DK 客户端时，客户端会检查此环境变量并连接到本地服务器以检索AWS凭证，并使用这些凭证签署 API 请求。这样，您就可以使用 AWS SDK 和其他工具来调用组件中的AWS服务。有关更多信息，请参阅 代币兑换服务。

> **Important**
>
> 2016 年 7 月 13 日，AWSSDK 中增加了以这种方式获取AWS凭证的支持。您的组件必须使用在该日期或之后创建的 AWS SDK 版本。有关更多信息，请参阅《亚马逊弹性容器服务开发者指南》中的使用支持的AWS软件开发工具包。

要在自定义组件中获取AWS凭据，请在组件配方中定义`aws.greengrass.TokenExchangeService`为依赖项。以下示例配方定义了一个组件，该组件用于安装 boto3 并运行一个 Python 脚本，该脚本使用来自令牌交换服务的AWS证书列出 Amazon S3 存储桶。

> **Note**
>
> 要运行此示例组件，您的设备必须具有`s3:ListAllMyBuckets`权限。有关更多信息，请参阅授权核心设备与AWS服务。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses the token exchange service to list
 S3 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "run": "python3 -u {artifacts:path}/list_s3_buckets.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "run": "py -3 -u {artifacts:path}/list_s3_buckets.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.ListS3Buckets
ComponentVersion: '1.0.0'
```

```
ComponentDescription: A component that uses the token exchange service to list S3
 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: '^2.0.0'
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        pip3 install --user boto3
      run: |-
        python3 -u {artifacts:path}/list_s3_buckets.py
  - Platform:
      os: windows
    Lifecycle:
      install:
        pip3 install --user boto3
      run: |-
        py -3 -u {artifacts:path}/list_s3_buckets.py
```

此示例组件运行以下 Python 脚本`list_s3_buckets.py`，其中列出了 Amazon S3 存储桶。

```
import boto3
import os

try:
    print("Creating boto3 S3 client...")
    s3 = boto3.client('s3')
    print("Successfully created boto3 S3 client")
except Exception as e:
    print("Failed to create boto3 s3 client. Error: " + str(e))
    exit(1)

try:
    print("Listing S3 buckets...")
    response = s3.list_buckets()
    for bucket in response['Buckets']:
        print(f'\t{bucket["Name"]}')
    print("Successfully listed S3 buckets")
```

```
except Exception as e:
    print("Failed to list S3 buckets. Error: " + str(e))
    exit(1)
```

运行 Docker 容器

您可以将AWS IoT Greengrass组件配置为使用存储在以下位置的映像运行 Docker 容器：

- Amazon Elastic Container Registry (Amazon ECR) 中的公有和私有图像存储库
- 公共 Docker Hub 存储库
- 公共 Docker 可信注册表
- S3 存储桶

在您的自定义组件中，将 Docker 镜像 URI 作为工件包括在内，以检索镜像并在核心设备上运行。对于 Amazon ECR 和 Docker Hub 镜像，您可以使用 Docker 应用程序管理器组件下载映像并管理私有 Amazon ECR 存储库的凭证。

主题

- 要求
- 在 Amazon ECR 或 Docker Hub 中使用公共镜像运行 Docker 容器
- 使用亚马逊 ECR 中的私有镜像运行 Docker 容器
- 使用亚马逊 S3 中的镜像运行 Docker 容器
- 在 Docker 容器组件中使用进程间通信
- 在 Docker 容器组件中使用AWS凭证 (Linux)
- 在 Docker 容器组件中使用流管理器 (Linux)

要求

要在组件中运行 Docker 容器，你需要以下内容：

- 一款 Greengrass 核心设备。如果没有，请参阅教程：AWS IoT Greengrass V2 入门。
- 安装在 Greengr@@ ass 核心设备上的 Docker E ngine 1.9.1 或更高版本。版本 20.10 是经验证可与AWS IoT Greengrass核心软件配合使用的最新版本。在部署运行 Docker 容器的组件之前，必须直接在核心设备上安装 Docker。

Tip

您还可以将核心设备配置为在组件安装时安装 Docker Engine。例如，以下安装脚本在加载 Docker 镜像之前安装 Docker 引擎。此安装脚本适用于基于 Debian 的 Linux 发行版，例如 Ubuntu。如果您使用此命令将组件配置为安装 Docker Engine，则可能需要在生命周期脚本true中将其设置RequiresPrivilege为才能运行安装和docker命令。有关更多信息，请参阅 AWS IoT Greengrass组件配方参考。

```
apt-get install docker-ce docker-ce-cli containerd.io && docker load -i
 {artifacts:path}/hello-world.tar
```

- 运行 Docker 容器组件的系统用户必须具有根或管理员权限，或者您必须将 Docker 配置为以非根用户或非管理员用户身份运行。
  - 在 Linux 设备上，您可以将用户添加到docker群组中，无需用户即可调用docker命令sudo。
  - 在 Windows 设备上，无需管理员权限即可将用户添加到docker-users群组中以调用docker命令。

  Linux or Unix

  要将或用于运行 Docker 容器组件的非 root 用户添加到ggc_userdocker群组，请运行以下命令。

  ```
  sudo usermod -aG docker ggc_user
  ```

  有关更多信息，请参阅以非 root 用户身份管理 Docker。

  Windows Command Prompt (CMD)

  要将或您用来运行 Docker 容器组件的用户添加到ggc_userdocker-users群组，请以管理员身份运行以下命令。

  ```
  net localgroup docker-users ggc_user /add
  ```

  Windows PowerShell

  要将或您用来运行 Docker 容器组件的用户添加到ggc_userdocker-users群组，请以管理员身份运行以下命令。

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- 作为卷安装在 Docker 容器中的 Docker 容器组件访问的文件。
- 如果您将 AWS IoT Greengrass Core 软件配置为使用网络代理，则必须将 Docker 配置为使用相同的代理服务器。

除了这些要求外，如果这些要求适用于您的环境，则还必须满足以下要求：

- 要使用 Docker Compose 创建和启动 Docker 容器，请在 Greengrass 核心设备上安装 Docker Compose，然后将你的 Docker Compose 文件上传到 S3 存储桶。您必须将 Compose 文件存储在 AWS 账户与AWS 区域组件相同的 S3 存储桶中。有关在自定义组件中使用该docker-compose up命令的示例，请参阅在 Amazon ECR 或 Docker Hub 中使用公共镜像运行 Docker 容器。
- 如果您在网络代理AWS IoT Greengrass后面运行，请将 Docker 守护程序配置为使用代理服务器。
- 如果您的 Docker 镜像存储在 Amazon ECR 或 Docker Hub 中，请将 Docker 组件管理器组件作为依赖项包含在 Docker 容器组件中。在部署组件之前，必须在核心设备上启动 Docker 守护程序。

  此外，还要将图像 URI 作为组件工件包括在内。图片 URI 必须采用以下示例docker:*registry*/*image*[*:tag|@digest*]所示的格式：

  - 亚马逊 ECR 的私有图片：docker:*account-id*.dkr.ecr.*region*.amazonaws.com/*repository*/*image*[*:tag|@digest*]
  - 公开 Amazon ECR 图片：docker:public.ecr.aws/*repository*/*image*[*:tag|@digest*]
  - 公共 Docker Hub 镜像：docker:*name*[*:tag|@digest*]

  有关使用存储在公共存储库中的镜像运行 Docker 容器的更多信息，请参阅在 Amazon ECR 或 Docker Hub 中使用公共镜像运行 Docker 容器。
- 如果您的 Docker 镜像存储在 Amazon ECR 私有存储库中，则必须将令牌交换服务组件作为依赖项包含在 Docker 容器组件中。此外，Greengrass 设备角色必须允许、ecr:GetDownloadUrlForLayer和操作ecr:BatchGetImage，如ecr:GetAuthorizationToken以下示例 IAM 策略所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
```

```
          "ecr:BatchGetImage",
          "ecr:GetDownloadUrlForLayer"
        ],
        "Resource": [
          "*"
        ],
        "Effect": "Allow"
      }
    ]
  }
```

有关使用存储在 Amazon ECR 私有存储库中的映像运行 Docker 容器的信息，请参阅。使用亚马逊 ECR 中的私有镜像运行 Docker 容器

- 要使用存储在 Amazon ECR 私有存储库中的 Docker 映像，私有存储库必须与核心设备位于同一 AWS 区域位置。
- 如果您的 Docker 映像或 Compose 文件存储在 S3 存储桶中，则 Greengrass 设备角色必须允许核心设备将图像下载为组件工件，如以下示例 IAM 策略所示。`s3:GetObject`

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

有关使用存储在 Amazon S3 中的映像运行 Docker 容器的信息，请参阅使用亚马逊 S3 中的镜像运行 Docker 容器。

- 要在 Docker 容器组件中使用进程间通信 (IPC)、AWS凭据或流管理器，必须在运行 Docker 容器时指定其他选项。有关更多信息，请参阅下列内容：
  - 在 Docker 容器组件中使用进程间通信
  - 在 Docker 容器组件中使用AWS凭证 (Linux)

- 在 Docker 容器组件中使用流管理器 (Linux)

在 Amazon ECR 或 Docker Hub 中使用公共镜像运行 Docker 容器

本节介绍如何创建自定义组件，该组件使用 Docker Compose 从存储在 Amazon ECR 和 Docker Hub 的 Docker 镜像中运行 Docker 容器。

使用 Docker Compose 运行 Docker 容器

1. 创建 Docker Compose 文件并将其上传到亚马逊 S3 存储桶。确保 Greengrass 设备角色允许设备访问 C `s3:GetObject` ompose 文件。以下示例中显示的示例撰写文件包括来自亚马逊 ECR 的亚马逊 CloudWatch 代理镜像和来自 Docker Hub 的 MySQL 镜像。

```
version: "3"
services:
  cloudwatchagent:
    image: "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
  mysql:
    image: "mysql:8.0"
```

2. 在AWS IoT Greengrass核心设备上创建自定义组件。以下示例中显示的示例配方具有以下属性：

   - Docker 应用程序管理器组件作为依赖项。该组件AWS IoT Greengrass允许从公共 Amazon ECR 和 Docker Hub 存储库下载映像。
   - 一个组件工件，用于在公共 Amazon ECR 存储库中指定 Docker 镜像。
   - 一种组件工件，用于在公共 Docker Hub 存储库中指定 Docker 镜像。
   - 一个组件工件，它指定 Docker Compose 文件，该文件包含你要运行的 Docker 镜像的容器。
   - 一个生命周期运行脚本，它使用 docker-compose up 从指定的映像创建和启动容器。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyDockerComposeComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses Docker Compose to run images
 from public Amazon ECR and Docker Hub.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
```

```
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "run": "docker-compose -f {artifacts:path}/docker-compose.yaml up"
      },
      "Artifacts": [
        {
          "URI": "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-
agent:latest"
        },
        {
          "URI": "docker:mysql:8.0"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/folder/docker-compose.yaml"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyDockerComposeComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that uses Docker Compose to run images from
 public Amazon ECR and Docker Hub.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
```

```
    Lifecycle:
        run: docker-compose -f {artifacts:path}/docker-compose.yaml up
    Artifacts:
      - URI: "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
      - URI: "docker:mysql:8.0"
      - URI: "s3://DOC-EXAMPLE-BUCKET/folder/docker-compose.yaml"
```

> **Note**
>
> 要在 Docker 容器组件中使用进程间通信 (IPC)、AWS凭据或流管理器，必须在运行 Docker 容器时指定其他选项。有关更多信息，请参阅下列内容：
>
> - 在 Docker 容器组件中使用进程间通信
> - 在 Docker 容器组件中使用AWS凭证 (Linux)
> - 在 Docker 容器组件中使用流管理器 (Linux)

3. 测试组件以验证其是否按预期工作。

> **Important**
>
> 在部署组件之前，必须安装并启动 Docker 守护程序。

在本地部署组件后，您可以运行 docker 容器 ls 命令来验证您的容器是否运行。

```
docker container ls
```

4. 组件准备就绪后，将该组件上传AWS IoT Greengrass到以部署到其他核心设备。有关更多信息，请参阅 发布组件以部署到您的核心设备。

使用亚马逊 ECR 中的私有镜像运行 Docker 容器

本节介绍如何使用存储在 Amazon ECR 私有存储库中的 Docker 映像创建运行 Docker 容器的自定义组件。

运行 Docker 容器

1. 在AWS IoT Greengrass核心设备上创建自定义组件。使用以下示例配方，它具有以下属性：

- Docker 应用程序管理器组件作为依赖项。此组件AWS IoT Greengrass允许管理从私有存储库下载图像的凭据。
- 作为依赖项的令牌交换服务组件。此组件允许检索AWS凭证AWS IoT Greengrass以与 Amazon ECR 进行交互。
- 一个组件工件，用于在私有 Amazon ECR 存储库中指定 Docker 镜像。
- 生命周期运行脚本，使用 docker run 从镜像创建和启动容器。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyPrivateDockerComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that runs a Docker container from a
 private Amazon ECR image.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
    },
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "run": "docker run account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
      },
      "Artifacts": [
        {
          "URI": "docker:account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
        }
      ]
    }
```

```
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyPrivateDockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from a private
 Amazon ECR image.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
        run: docker run account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]
    Artifacts:
      - URI: "docker:account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]"
```

> **Note**
>
> 要在 Docker 容器组件中使用进程间通信 (IPC)、AWS凭据或流管理器，必须在运行 Docker 容器时指定其他选项。有关更多信息，请参阅下列内容：
>
> - 在 Docker 容器组件中使用进程间通信
> - 在 Docker 容器组件中使用AWS凭证 (Linux)
> - 在 Docker 容器组件中使用流管理器 (Linux)

2. 测试组件以验证其是否按预期工作。

Important

在部署组件之前，必须安装并启动 Docker 守护程序。

在本地部署组件后，您可以运行 docker 容器 ls 命令来验证您的容器是否运行。

```
docker container ls
```

3. 将组件上传到AWS IoT Greengrass以部署到其他核心设备。有关更多信息，请参阅 发布组件以部署到您的核心设备。

使用亚马逊 S3 中的镜像运行 Docker 容器

本节介绍如何通过存储在亚马逊 S3 中的 Docker 镜像在组件中运行 Docker 容器。

通过亚马逊 S3 中的镜像在组件中运行 Docker 容器

1. 运行 docker save 命令创建 Docker 容器的备份。您可以将此备份作为组件工件提供来运行容器AWS IoT Greengrass。将 *hello-wor* ld 替换为图像的名称，将 *hello-world.tar* 替换为要创建的存档文件的名称。

```
docker save hello-world > artifacts/com.example.MyDockerComponent/1.0.0/hello-
world.tar
```

2. 在AWS IoT Greengrass核心设备上创建自定义组件。使用以下示例配方，它具有以下属性：

   - 一种生命周期安装脚本，它使用 docker 加载从存档中加载 Docker 镜像。
   - 生命周期运行脚本，使用 docker run 从镜像创建和启动容器。该--rm选项会在容器退出时对其进行清理。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyS3DockerComponent",
  "ComponentVersion": "1.0.0",
```

```
  "ComponentDescription": "A component that runs a Docker container from an
 image in an S3 bucket.",
  "ComponentPublisher": "Amazon",
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": {
          "Script": "docker load -i {artifacts:path}/hello-world.tar"
        },
        "run": {
          "Script": "docker run --rm hello-world"
        }
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      run:
        Script: docker run --rm hello-world
```

> **Note**
>
> 要在 Docker 容器组件中使用进程间通信 (IPC)、AWS凭据或流管理器，必须在运行 Docker 容器时指定其他选项。有关更多信息，请参阅下列内容：
>
> - 在 Docker 容器组件中使用进程间通信
> - 在 Docker 容器组件中使用AWS凭证 (Linux)
> - 在 Docker 容器组件中使用流管理器 (Linux)

3. 测试组件以验证其是否按预期工作。

   在本地部署组件后，您可以运行 docker 容器 ls 命令来验证您的容器是否运行。

   ```
   docker container ls
   ```

4. 组件准备就绪后，将 Docker 映像存档上传到 S3 存储桶，然后将其 URI 添加到组件配方中。然后，您可以将组件上传到AWS IoT Greengrass以部署到其他核心设备。有关更多信息，请参阅 发布组件以部署到您的核心设备。

   完成后，组件配方应类似于以下示例。

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyS3DockerComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that runs a Docker container from an
    image in an S3 bucket.",
     "ComponentPublisher": "Amazon",
     "Manifests": [
       {
         "Platform": {
           "os": "linux"
         },
         "Lifecycle": {
           "install": {
             "Script": "docker load -i {artifacts:path}/hello-world.tar"
           },
           "run": {
   ```

```
          "Script": "docker run --rm hello-world"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      run:
        Script: docker run --rm hello-world
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar
```

在 Docker 容器组件中使用进程间通信

您可以使用中的 Greengrass 进程间通信 (IPC) 库与 Greengrass 核、AWS IoT Device SDK其他 Greengrass 组件以及进行通信。AWS IoT Core有关更多信息，请参阅 使用AWS IoT Device SDK与 Greengrass 原子核、其他组件进行通信 AWS IoT Core。

要在 Docker 容器组件中使用 IPC，必须使用以下参数运行 Docker 容器：

- 将 IPC 插槽安装在容器中。Greengrass nucleus 在环境变量中提供 IPC 套接字文件路径。`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`
- 将`SVCUID`和`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`环境变量设置为 Greengrass 核为组件提供的值。您的组件使用这些环境变量来验证与 Greengrass 核的连接。

Example 示例配方：将 MQTT 消息发布到 AWS IoT Core (Python)

以下配方定义了一个将 MQTT 消息发布到的 Docker 容器组件示例。AWS IoT Core该配方具有以下属性：

- 一种授权策略 (`accessControl`)，允许组件向其发布AWS IoT Core有关所有主题的 MQTT 消息。有关更多信息，请参阅授权组件执行 IPC 操作和 AWS IoT CoreMQTT IPC 授权。
- 一个组件工件，它将 Docker 镜像指定为 Amazon S3 中的 TAR 存档。
- 从 TAR 存档中加载 Docker 镜像的生命周期安装脚本。
- 基于镜像运行 Docker 容器的生命周期运行脚本。D ocker 运行命令具有以下参数：
  - `-v`参数将 Greengrass IPC 套接字安装在容器中。
  - 前两个`-e`参数在 Docker 容器中设置所需的环境变量。
  - 其他`-e`参数设置了本示例使用的环境变量。
  - 该`--rm`参数会在容器退出时清理容器。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.PublishToIoTCore",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses interprocess communication to publish an MQTT
 message to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "topic": "test/topic/java",
      "message": "Hello, World!",
      "qos": "1",
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.python.docker.PublishToIoTCore:pubsub:1": {
```

```
            "policyDescription": "Allows access to publish to IoT Core on all
 topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/publish-to-iot-core.tar",
        "run": "docker run -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e SVCUID -e
 AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e MQTT_TOPIC=
\"{configuration:/topic}\" -e MQTT_MESSAGE=\"{configuration:/message}\" -e MQTT_QOS=
\"{configuration:/qos}\" --rm publish-to-iot-core"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.PublishToIoTCore
ComponentVersion: 1.0.0
ComponentDescription: Uses interprocess communication to publish an MQTT message to
 IoT Core.
```

```
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    topic: 'test/topic/java'
    message: 'Hello, World!'
    qos: '1'
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        'com.example.python.docker.PublishToIoTCore:pubsub:1':
          policyDescription: Allows access to publish to IoT Core on all topics.
          operations:
            - 'aws.greengrass#PublishToIoTCore'
          resources:
            - '*'
Manifests:
  - Platform:
      os: all
    Lifecycle:
      install: 'docker load -i {artifacts:path}/publish-to-iot-core.tar'
      run: |
        docker run \
          -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e SVCUID \
          -e AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e MQTT_TOPIC="{configuration:/topic}" \
          -e MQTT_MESSAGE="{configuration:/message}" \
          -e MQTT_QOS="{configuration:/qos}" \
          --rm publish-to-iot-core
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar
```

在 Docker 容器组件中使用AWS凭证 (Linux)

您可以使用令牌交换服务组件与 Greengrass 组件中的AWS服务进行交互。该组件使用本地容器服务器提供来自核心设备的令牌交换角色的AWS凭证。有关更多信息，请参阅 与AWS服务互动。

> **Note**
>
> 本节中的示例仅适用于 Linux 核心设备。

要在 Docker 容器组件中使用来自令牌交换服务的AWS凭证，必须使用以下参数运行 Docker 容器：

- 使用`--network=host`参数提供对主机网络的访问权限。此选项使 Docker 容器能够连接到本地令牌交换服务以检索AWS凭据。这个参数仅适用于适用于 Linux 的 Docker。

> **Warning**
>
> 此选项允许容器访问主机上的所有本地网络接口，因此，与在没有主机网络访问权限的情况下运行 Docker 容器相比，此选项不那么安全。在开发和运行使用此选项的 Docker 容器组件时，请考虑这一点。有关更多信息，请参阅 Docker 文档中的网络：主机。

- 将`AWS_CONTAINER_CREDENTIALS_FULL_URI`和`AWS_CONTAINER_AUTHORIZATION_TOKEN`环境变量设置为 Greengrass 核为组件提供的值。 AWSSDK 使用这些环境变量来检索AWS证书。

Example 示例配方：在 Docker 容器组件中列出 S3 存储桶 (Python)

以下配方定义了一个示例 Docker 容器组件，该组件列出了你的 S3 存储桶。AWS 账户该配方具有以下属性：

- 作为依赖项的令牌交换服务组件。这种依赖关系使组件能够检索AWS凭证以与其他AWS服务进行交互。
- 一个组件工件，它将 Docker 镜像指定为 Amazon S3 中的 tar 存档。
- 从 TAR 存档中加载 Docker 镜像的生命周期安装脚本。
- 基于镜像运行 Docker 容器的生命周期运行脚本。D ocker 运行命令具有以下参数：
  - 该`--network=host`参数提供容器对主机网络的访问权限，因此容器可以连接到令牌交换服务。
  - 该`-e`参数在 Docker 容器中设置所需的环境变量。
  - 该`--rm`参数会在容器退出时清理容器。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses the token exchange service to lists your S3
 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
```

```
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/list-s3-buckets.tar",
        "run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN -e
 AWS_CONTAINER_CREDENTIALS_FULL_URI --rm list-s3-buckets"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.ListS3Buckets
ComponentVersion: 1.0.0
ComponentDescription: Uses the token exchange service to lists your S3 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/list-s3-buckets.tar'
      run: |
        docker run \
```

```
            --network=host \
            -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
            -e AWS_CONTAINER_CREDENTIALS_FULL_URI \
            --rm list-s3-buckets
      Artifacts:
        - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar
```

在 Docker 容器组件中使用流管理器 (Linux)

您可以使用流管理器组件来管理 Greengrass 组件中的数据流。此组件使您能够处理数据流并将大量物联网数据传输到. AWS Cloud AWS IoT Greengrass提供了用于与流管理器组件交互的流管理器 SDK。有关更多信息，请参阅 管理 Greengrass 核心设备上的数据流。

> **Note**
>
> 本节中的示例仅适用于 Linux 核心设备。

要在 Docker 容器组件中使用流管理器 SDK，必须使用以下参数运行 Docker 容器：

- 使用`--network=host`参数提供对主机网络的访问权限。此选项使 Docker 容器能够通过本地 TLS 连接与流管理器组件进行交互。这个参数仅适用于适用于 Linux 的 Docker

  > **Warning**
  >
  > 此选项允许容器访问主机上的所有本地网络接口，因此，与在没有主机网络访问权限的情况下运行 Docker 容器相比，此选项不那么安全。在开发和运行使用此选项的 Docker 容器组件时，请考虑这一点。有关更多信息，请参阅 Docker 文档中的网络：主机。

- 如果您将流管理器组件配置为需要身份验证（这是默认行为），请将`AWS_CONTAINER_CREDENTIALS_FULL_URI`环境变量设置为 Greengrass nucleus 为组件提供的值。有关更多信息，请参阅直播管理器配置。
- 如果将流管理器组件配置为使用非默认端口，请使用进程间通信 (IPC) 从流管理器组件配置中获取该端口。要使用 IPC，您必须使用其他选项运行 Docker 容器。有关更多信息，请参阅下列内容：
  - 在应用程序代码中Connect 流管理器
  - 在 Docker 容器组件中使用进程间通信

Example 示例配方：将文件流式传输到 Docker 容器组件中的 S3 存储桶 (Python)

以下配方定义了一个示例 Docker 容器组件，该组件用于创建文件并将其流式传输到 S3 存储桶。该配方具有以下属性：

- 作为依赖项的流管理器组件。这种依赖关系使该组件能够使用流管理器 SDK 与流管理器组件进行交互。
- 一个组件工件，它将 Docker 镜像指定为 Amazon S3 中的 TAR 存档。
- 从 TAR 存档中加载 Docker 镜像的生命周期安装脚本。
- 基于镜像运行 Docker 容器的生命周期运行脚本。D ocker 运行命令具有以下参数：
  - 该--network=host参数提供容器对主机网络的访问权限，因此容器可以连接到流管理器组件。
  - 第一个-e参数在 Docker 容器中设置所需的AWS_CONTAINER_AUTHORIZATION_TOKEN环境变量。
  - 其他-e参数设置了本示例使用的环境变量。
  - -v参数将组件的工作文件夹挂载到容器中。此示例在工作文件夹中创建一个文件，以便使用流管理器将该文件上传到 Amazon S3。
  - 该--rm参数会在容器退出时清理容器。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.StreamFileToS3",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Creates a text file and uses stream manager to stream the
 file to S3.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "bucketName": ""
    }
  },
  "Manifests": [
```

```
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/stream-file-to-s3.tar",
        "run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN
 -e BUCKET_NAME=\"{configuration:/bucketName}\" -e WORK_PATH=\"{work:path}\" -v
 {work:path}:{work:path} --rm stream-file-to-s3"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.StreamFileToS3
ComponentVersion: 1.0.0
ComponentDescription: Creates a text file and uses stream manager to stream the file
 to S3.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    bucketName: ''
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/stream-file-to-s3.tar'
      run: |
        docker run \
          --network=host \
```

```
            -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
            -e BUCKET_NAME="{configuration:/bucketName}" \
            -e WORK_PATH="{work:path}" \
            -v {work:path}:{work:path} \
            --rm stream-file-to-s3
      Artifacts:
        - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar
```

AWS IoT Greengrass组件配方参考

组件配方是一个定义组件的详细信息、依赖关系、工件和生命周期的文件。例如，组件生命周期指定要运行的安装、运行和关闭组件的命令。内AWS IoT Greengrass核使用您在配方中定义的生命周期来安装和运行组件。部署组件时，该AWS IoT Greengrass服务使用配方来识别要部署到核心设备的依赖项和工件。

在配方中，您可以为组件支持的每个平台定义唯一的依赖关系和生命周期。您可以使用此功能将组件部署到具有不同要求的多个平台的设备上。您还可以使用它来AWS IoT Greengrass防止在不支持组件的设备上安装该组件。

每个食谱都包含一份清单清单。每个清单都指定了一组平台要求以及平台满足这些要求的核心设备要使用的生命周期和工件。核心设备使用第一份清单，其中包含该设备满足的平台要求。指定没有任何平台要求的清单，以匹配任何核心设备。

您还可以指定清单中没有的全局生命周期。在全球生命周期中，您可以使用选择键来标识生命周期的子部分。然后，您可以在清单中指定这些选择密钥，以便在清单的生命周期之外使用全局生命周期中的这些部分。仅当清单未定义生命周期时，核心设备才会使用清单的选择密钥。您可以使用清单中的`all`选择来匹配全局生命周期的各个部分，而无需选择键。

在AWS IoT Greengrass核心软件选择与核心设备匹配的清单后，它会执行以下操作来确定要使用的生命周期步骤：

- 如果所选清单定义了生命周期，则核心设备将使用该生命周期。
- 如果所选清单未定义生命周期，则核心设备将使用全局生命周期。核心设备执行以下操作来确定要使用全球生命周期的哪些部分：
  - 如果清单定义了选择密钥，则核心设备将使用全局生命周期中包含清单选择密钥的部分。
  - 如果清单未定义选择密钥，则核心设备将使用全局生命周期中没有选择密钥的部分。此行为等同于定义`all`选择内容的清单。

> **Important**
>
> 核心设备必须满足至少一个清单的平台要求才能安装该组件。如果没有清单与核心设备匹配，则AWS IoT Greengrass核心软件不会安装该组件，部署将失败。

你可以用 JSON 或 YAML 格式定义配方。食谱示例部分包括每种格式的食谱。

主题

- 食谱验证
- 食谱格式
- 食谱变量
- 食谱示例

食谱验证

Greengrass 在创建组件版本时会验证 JSON 或 YAML 组件配方。此配方验证会检查您的 JSON 或 YAML 组件配方中是否存在常见错误，以防止出现潜在的部署问题。验证会检查配方中是否存在常见错误（例如，缺少逗号、大括号和字段），并确保配方格式正确。

如果您收到食谱验证错误消息，请检查您的食谱中是否有任何缺少逗号、大括号或字段。查看食谱格式，确认您没有遗漏任何字段。

食谱格式

在为组件定义配方时，需要在配方文档中指定以下信息。同样的结构适用于 YAML 和 JSON 格式的配方。

`RecipeFormatVersion`

食谱的模板版本。选择以下选项：

- `2020-01-25`

`ComponentName`

此配方定义的组件的名称。在每个区域中，组件名称必须是唯一的AWS 账户。

> **提示**
>
> - 使用反向域名格式以避免公司内部的域名冲突。例如，如果您的公司拥有一个太阳能项目，`example.com`并且您从事该项目，则可以命名您的 Hello World 组件`com.example.solar.HelloWorld`。这有助于避免公司内部的组件名称冲突。
> - 避免在组件名称中使用`aws.greengrass`前缀。 AWS IoT Greengrass将此前缀用于它提供的公共组件。如果您选择与公共组件相同的名称，则您的组件将替换该组件。然后，在部署依赖于该公共组件的组件时，AWS IoT Greengrass提供您的组件而不是公共组件。此功能使您可以覆盖公共组件的行为，但如果您不打算覆盖公共组件，它也可能会破坏其他组件。

`ComponentVersion`

组件版本。主要、次要和补丁值的最大值为 999999。

> **Note**
>
> AWS IoT Greengrass使用组件的语义版本。语义版本遵循 major.minor.patch 编号系统。例如，版本`1.0.0`表示组件的第一个主要版本。有关更多信息，请参阅语义版本规范。

`ComponentDescription`

（可选）组件的描述。

`ComponentPublisher`

组件的发布者或作者。

`ComponentConfiguration`

（可选）定义组件配置或参数的对象。您可以定义默认配置，然后在部署组件时，可以指定要提供给组件的配置对象。组件配置支持嵌套参数和结构。该对象包含以下信息：

`DefaultConfiguration`

定义组件默认配置的对象。您可以定义此对象的结构。

Note

AWS IoT Greengrass使用 JSON 作为配置值。JSON 指定了数字类型，但不区分整数和浮点数。因此，配置值可能会转换为浮点数。AWS IoT Greengrass为确保您的组件使用正确的数据类型，我们建议您将数字配置值定义为字符串。然后，让你的组件将它们解析为整数或浮点数。这样可以确保您的配置值在配置和核心设备上具有相同的类型。

`ComponentDependencies`

（可选）一个对象字典，每个对象都定义了组件的组件依赖关系。每个对象的密钥标识组件依赖关系的名称。 AWS IoT Greengrass安装组件时安装组件依赖关系。 AWS IoT Greengrass等待依赖关系启动后再启动组件。每个对象都包含以下信息：

`VersionRequirement`

npm 风格的语义版本约束，用于定义此依赖项的兼容组件版本。您可以指定一个版本或一系列版本。有关更多信息，请参阅 npm 语义版本计算器。

`DependencyType`

（可选）此依赖项的类型。从以下选项中进行选择。

- `SOFT` – 如果依赖项更改状态，组件不会重新启动。
- `HARD` – 如果依赖项更改状态，组件将会重新启动。

默认值为 `HARD`。

`ComponentType`

（可选）组件的类型。

Note

我们不建议您在配方中指定组件类型。 AWS IoT Greengrass在创建组件时为您设置类型。

该类型可以是以下类型之一：

- `aws.greengrass.generic`— 该组件运行命令或提供工件。
- `aws.greengrass.lambda`— 该组件使用 Lambda 启动器组件运行 Lambda 函数。该`ComponentSource`参数指定此组件运行的 Lambda 函数的 ARN。

我们不建议您使用此选项，因为它是在您通过 Lambda 函数创建组件AWS IoT Greengrass时设置的。有关更多信息，请参阅 运行AWS Lambda函数。

- `aws.greengrass.plugin`— 该组件与 Greengrass 核心在同一 Java 虚拟机 (JVM) 中运行。如果您部署或重启插件组件，Greengrass 核心将重新启动。

  插件组件使用与 Greengrass 核心相同的日志文件。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

  我们不建议您在组件配方中使用此选项，因为它适用于AWS由 Java 编写的、直接与 Greengrass 核心接口的组件。有关哪些公共组件是插件的更多信息，请参阅AWS-提供的组件。
- `aws.greengrass.nucleus`— 原子核成分。有关更多信息，请参阅 Greengrass 核。

  我们不建议您在组件配方中使用此选项。它适用于Greengrass nucleus组件，该组件提供了核心软件的最低功能。AWS IoT Greengrass

`aws.greengrass.generic`当您使用配方创建组件时，或者从 Lambda 函数创建组件`aws.greengrass.lambda`时，默认为。

有关更多信息，请参阅 组件类型。

`ComponentSource`

（可选）组件运行的 Lambda 函数的 ARN。

我们不建议您在配方中指定组件来源。 AWS IoT Greengrass当您通过 Lambda 函数创建组件时，为您设置此参数。有关更多信息，请参阅 运行AWS Lambda函数。

`Manifests`

对象列表，每个对象都定义了组件的生命周期、参数和平台要求。如果核心设备符合多个清单的平台要求，则AWS IoT Greengrass使用与核心设备匹配的第一个清单。为确保核心设备使用正确的清单，请先定义具有更严格平台要求的清单。适用于所有平台的清单必须是列表中的最后一个清单。

> **Important**
> 核心设备必须满足至少一个清单的平台要求才能安装该组件。如果没有清单与核心设备匹配，则AWS IoT Greengrass核心软件不会安装该组件，部署将失败。

每个对象都包含以下信息：

Name

（可选）此清单定义的平台的友好名称。

如果省略此参数，则会从平台AWS IoT Greengrass创建名称，os然后。architecture

Platform

（可选）一个对象，用于定义此清单所适用的平台。省略此参数可定义适用于所有平台的清单。

此对象指定有关核心设备运行平台的键值对。部署此组件时，AWS IoT GreengrassCore 软件会将这些键值对与核心设备上的平台属性进行比较。C AWS IoT Greengrass ore 软件始终定义os和architecture，它可能会定义其他属性。在部署 Greengrass nucleus 组件时，您可以为核心设备指定自定义平台属性。有关更多信息，请参阅 Gre engrass nucleus 组件的平台覆盖参数。

对于每个键值对，您可以指定以下值之一：

- 一个精确的值，例如linux或windows。精确值必须以字母或数字开头。
- *，它与任何值匹配。当值不存在时，这也会匹配。
- Java 风格的正则表达式，例如。/windows|linux/正则表达式必须以斜杠字符 (/) 开头和结尾。例如，正则表达式/.+/匹配任何非空值。

该对象包含以下信息：

os

（可选）此清单支持的平台的操作系统的名称。常用平台包括以下值：

- linux
- windows
- darwin (macOS)

architecture

（可选）此清单支持的平台的处理器架构。常见架构包括以下值：

- amd64
- arm
- aarch64
- x86

architecture.detail

（可选）此清单支持的平台的处理器架构详细信息。常见的架构细节包括以下值：

- arm61
- arm71
- arm81

key

（可选）您为此清单定义的平台属性。将*##*替换为平台属性的名称。AWS IoT GreengrassCore 软件将此平台属性与您在 Greengrass nucleus 组件配置中指定的键值对进行匹配。有关更多信息，请参阅 Gre engrass nucleus 组件的平台覆盖参数。

> Tip
>
> 使用反向域名格式以避免公司内部的域名冲突。例如，如果您的公司拥有一个广播项目，example.com而您从事广播项目，则可以命名自定义平台属性com.example.radio.RadioModule。这有助于避免公司内部的平台属性名称冲突。

例如，您可以定义平台属性com.example.radio.RadioModule，根据核心设备上可用的无线电模块来指定不同的清单。每个清单可以包含适用于不同硬件配置的不同构件，因此您可以将最少的软件集部署到核心设备。

`Lifecycle`

一个对象或字符串，用于定义如何在此清单定义的平台上安装和运行组件。您还可以定义适用于所有平台的全球生命周期。只有在要使用的清单未指定生命周期时，核心设备才会使用全局生命周期。

> Note
>
> 你可以在清单中定义这个生命周期。您在此处指定的生命周期步骤仅适用于此清单所定义的平台。您还可以定义适用于所有平台的全球生命周期。

此对象或字符串包含以下信息：

`Setenv`

（可选）提供给所有生命周期脚本的环境变量字典。您可以在每个生命周期脚本Setenv中使用覆盖这些环境变量。

`install`

（可选）定义组件安装时要运行的脚本的对象或字符串。每次软件启动时，AWS IoT GreengrassCore 软件也会运行此生命周期步骤。

如果`install`脚本以成功代码退出，则组件将进入`INSTALLED`状态。

此对象或字符串包含以下信息：

`Script`

要运行的脚本。

`RequiresPrivilege`

（可选）您可以使用 root 权限运行脚本。如果将此选项设置为`true`，则 AWS IoT Greengrass Core 软件将以 root 用户身份运行此生命周期脚本，而不是以您配置为运行此组件的系统用户的身份运行此生命周期脚本。默认值为 `false`。

`Skipif`

（可选）用于确定是否运行脚本的检查。您可以定义以检查路径上是否有可执行文件或文件是否存在。如果输出为真，则 AWS IoT Greengrass Core 软件将跳过该步骤。选择以下支票之一：

- `onpath` *`runnable`*— 检查系统路径上是否有可运行对象。例如，如果 Python 3 可用，则使用**`onpath python3`**跳过此生命周期步骤。
- `exists` *`file`*— 检查文件是否存在。例如，如果`/tmp/my-configuration.db`存在此生命周期步骤，则使用**`exists /tmp/my-configuration.db`**可跳过此生命周期步骤。

`Timeout`

（可选）在AWS IoT Greengrass核心软件终止进程之前，脚本可以运行的最大时间（以秒为单位）。

默认值：120 秒

`Setenv`

（可选）要提供给脚本的环境变量字典。这些环境变量会覆盖您在中提供的变量`Lifecycle.Setenv`。

`run`

（可选）定义组件启动时要运行的脚本的对象或字符串。

当此生命周期步骤运行时，组件进入RUNNING状态。如果run脚本以成功代码退出，则组件将进入STOPPING状态。如果指定了shutdown脚本，则它会运行；否则组件进入FINISHED状态。

依赖此组件的组件将在此生命周期步骤运行时启动。要运行后台进程，例如依赖组件使用的服务，请改用startup生命周期步骤。

当您部署具有run生命周期的组件时，核心设备可以在此生命周期脚本运行后立即报告部署已完成。因此，即使run生命周期脚本在运行后不久就失败了，部署也可以完成并成功。如果您希望部署状态取决于组件启动脚本的结果，请改用startup生命周期步骤。

> **Note**
>
> 您只能定义一个startup或run生命周期。

此对象或字符串包含以下信息：

`Script`

要运行的脚本。

`RequiresPrivilege`

（可选）您可以使用 root 权限运行脚本。如果将此选项设置为`true`，则 AWS IoT Greengrass Core 软件将以 root 用户身份运行此生命周期脚本，而不是以您配置为运行此组件的系统用户的身份运行此生命周期脚本。默认值为 `false`。

`Skipif`

（可选）用于确定是否运行脚本的检查。您可以定义以检查路径上是否有可执行文件或文件是否存在。如果输出为真，则 AWS IoT Greengrass Core 软件将跳过该步骤。选择以下支票之一：

- `onpath` *`runnable`*— 检查系统路径上是否有可运行对象。例如，如果 Python 3 可用，则使用**`onpath python3`**跳过此生命周期步骤。
- `exists` *`file`*— 检查文件是否存在。例如，如果`/tmp/my-configuration.db`存在此生命周期步骤，则使用**`exists /tmp/my-configuration.db`**可跳过此生命周期步骤。

`Timeout`

（可选）在AWS IoT Greengrass核心软件终止进程之前，脚本可以运行的最大时间（以秒为单位）。

默认情况下，此生命周期步骤不会超时。如果省略此超时，run脚本将一直运行直到退出。

Setenv

（可选）要提供给脚本的环境变量字典。这些环境变量会覆盖您在中提供的变量Lifecycle.Setenv。

startup

（可选）定义组件启动时要运行的后台进程的对象或字符串。

startup用于运行必须成功退出或将组件状态更新为RUNNING才能启动依赖组件的命令。使用 UpdateStateIPC 操作将组件的状态设置为RUNNING或ERRORED当组件启动未退出的脚本时。例如，您可以定义一个用于启动 MySQL 进程的startup步骤/etc/init.d/mysqld start。

当此生命周期步骤运行时，组件进入STARTING状态。如果startup脚本以成功代码退出，则组件将进入RUNNING状态。然后，可以启动依赖组件。

当您部署具有startup生命周期的组件时，核心设备可以在此生命周期脚本退出或报告其状态后将部署报告为已完成。换句话说，部署的状态是IN_PROGRESS直到所有组件的启动脚本退出或报告状态为止。

Note

您只能定义一个startup或run生命周期。

此对象或字符串包含以下信息：

Script

要运行的脚本。

RequiresPrivilege

（可选）您可以使用 root 权限运行脚本。如果将此选项设置为true，则 AWS IoT Greengrass Core 软件将以 root 用户身份运行此生命周期脚本，而不是以您配置为运行此组件的系统用户的身份运行此生命周期脚本。默认值为 false。

`Skipif`

（可选）用于确定是否运行脚本的检查。您可以定义以检查路径上是否有可执行文件或文件是否存在。如果输出为真，则 AWS IoT Greengrass Core 软件将跳过该步骤。选择以下支票之一：

- `onpath` *`runnable`*— 检查系统路径上是否有可运行对象。例如，如果 Python 3 可用，则使用**`onpath python3`**跳过此生命周期步骤。
- `exists` *`file`*— 检查文件是否存在。例如，如果`/tmp/my-configuration.db`存在此生命周期步骤，则使用**`exists /tmp/my-configuration.db`**可跳过此生命周期步骤。

`Timeout`

（可选）在AWS IoT Greengrass核心软件终止进程之前，脚本可以运行的最大时间（以秒为单位）。

默认值：120 秒

`Setenv`

（可选）要提供给脚本的环境变量字典。这些环境变量会覆盖您在中提供的变量`Lifecycle.Setenv`。

`shutdown`

（可选）定义组件关闭时要运行的脚本的对象或字符串。使用关闭生命周期来执行要在组件处于`STOPPING`状态时运行的代码。关闭生命周期可用于停止`startup`或`run`脚本启动的进程。

如果您在中启动后台进程`startup`，请使用该`shutdown`步骤在组件关闭时停止该进程。例如，您可以定义一个停止 MySQL 进程的`shutdown`步骤`/etc/init.d/mysqld stop`。

该`shutdown`脚本在组件进入`STOPPING`状态后运行。如果脚本成功完成，则组件将进入`FINISHED`状态。

此对象或字符串包含以下信息：

`Script`

要运行的脚本。

`RequiresPrivilege`

（可选）您可以使用 root 权限运行脚本。如果将此选项设置为`true`，则 AWS IoT Greengrass Core 软件将以 root 用户身份运行此生命周期脚本，而不是以您配置为运行此组件的系统用户的身份运行此生命周期脚本。默认值为 `false`。

`Skipif`

（可选）用于确定是否运行脚本的检查。您可以定义以检查路径上是否有可执行文件或文件是否存在。如果输出为真，则 AWS IoT Greengrass Core 软件将跳过该步骤。选择以下支票之一：

- `onpath` *`runnable`*— 检查系统路径上是否有可运行对象。例如，如果 Python 3 可用，则使用**`onpath python3`**跳过此生命周期步骤。
- `exists` *`file`*— 检查文件是否存在。例如，如果`/tmp/my-configuration.db`存在此生命周期步骤，则使用**`exists /tmp/my-configuration.db`**可跳过此生命周期步骤。

`Timeout`

（可选）在AWS IoT Greengrass核心软件终止进程之前，脚本可以运行的最大时间（以秒为单位）。

默认值：15 秒。

`Setenv`

（可选）要提供给脚本的环境变量字典。这些环境变量会覆盖您在中提供的变量`Lifecycle.Setenv`。

`recover`

（可选）一个对象或字符串，用于定义在组件遇到错误时要运行的脚本。

此步骤在组件进入`ERRORED`状态时运行。如果组件变为`ERRORED`三次但未成功恢复，则该组件将变为`BROKEN`状态。要修复组`BROKEN`件，必须重新部署它。

此对象或字符串包含以下信息：

`Script`

要运行的脚本。

`RequiresPrivilege`

（可选）您可以使用 root 权限运行脚本。如果将此选项设置为`true`，则 AWS IoT Greengrass Core 软件将以 root 用户身份运行此生命周期脚本，而不是以您配置为运行此组件的系统用户的身份运行此生命周期脚本。默认值为 `false`。

`Skipif`

（可选）用于确定是否运行脚本的检查。您可以定义以检查路径上是否有可执行文件或文件是否存在。如果输出为真，则 AWS IoT Greengrass Core 软件将跳过该步骤。选择以下支票之一：

- `onpath` *`runnable`*— 检查系统路径上是否有可运行对象。例如，如果 Python 3 可用，则使用**`onpath python3`**跳过此生命周期步骤。
- `exists` *`file`*— 检查文件是否存在。例如，如果`/tmp/my-configuration.db`存在此生命周期步骤，则使用**`exists /tmp/my-configuration.db`**可跳过此生命周期步骤。

`Timeout`

（可选）在AWS IoT Greengrass核心软件终止进程之前，脚本可以运行的最大时间（以秒为单位）。

默认值：60 秒。

`Setenv`

（可选）要提供给脚本的环境变量字典。这些环境变量会覆盖您在中提供的变量`Lifecycle.Setenv`。

`bootstrap`

（可选）定义要求AWS IoT Greengrass核心软件或核心设备重新启动的脚本的对象或字符串。例如，这使您可以开发一个在安装操作系统更新或运行时更新后执行重启的组件。

> **Note**
>
> 要安装不需要重启AWS IoT Greengrass核心软件或设备的更新或依赖项，请使用安装生命周期。

在以下情况下，当AWS IoT Greengrass核心软件部署组件时，此生命周期步骤在安装生命周期步骤之前运行：

- 该组件首次部署到核心设备。
- 组件版本发生变化。
- 由于组件配置更新，引导脚本会发生变化。

AWS IoT Greengrass核心软件完成部署中包含引导步骤的所有组件的引导步骤后，软件将重新启动。

> **Important**
>
> 必须将AWS IoT Greengrass核心软件配置为系统服务才能重新启动AWS IoT Greengrass核心软件或核心设备。如果您未将 AWS IoT Greengrass Core 软件配置为系统服务，则该软件将无法重新启动。有关更多信息，请参阅 将 Greengrass 核心配置为系统服务。

此对象或字符串包含以下信息：

`BootstrapOnRollback`

> **Note**
>
> 启用此功能后，`BootstrapOnRollback`只有在失败的目标部署中已完成或尝试运行引导生命周期步骤的组件才会运行。此功能适用于 Greengrass nucleus 版本 2.12.0 及更高版本。

（可选）您可以将引导生命周期步骤作为回滚部署的一部分来运行。如果将此选项设置为`true`，则将在回滚部署中定义的引导生命周期步骤运行。部署失败时，先前版本的组件引导程序生命周期将在回滚部署期间再次运行。

默认值为 `false`。

`Script`

要运行的脚本。此脚本的退出代码定义了重启指令。使用以下退出代码：

- `0`— 不要重启AWS IoT Greengrass核心软件或核心设备。在所有组件启动后，AWS IoT GreengrassCore 软件仍会重新启动。
- `100`— 请求重新启动 AWS IoT Greengrass Core 软件。

- 101— 请求重启核心设备。

  退出代码 100 到 199 是为特殊行为保留的。其他退出代码代表脚本错误。

`RequiresPrivilege`

（可选）您可以使用 root 权限运行脚本。如果将此选项设置为`true`，则 AWS IoT Greengrass Core 软件将以 root 用户身份运行此生命周期脚本，而不是以您配置为运行此组件的系统用户的身份运行此生命周期脚本。默认值为 `false`。

`Timeout`

（可选）在AWS IoT Greengrass核心软件终止进程之前，脚本可以运行的最大时间（以秒为单位）。

默认值：120 秒

`Setenv`

（可选）要提供给脚本的环境变量字典。这些环境变量会覆盖您在中提供的变量`Lifecycle.Setenv`。

`Selections`

（可选）选择键列表，用于指定要为此清单运行的全局生命周期的各个部分。在全局生命周期中，您可以使用任何级别的选择键来定义生命周期步骤，以选择生命周期的子部分。然后，核心设备使用与该清单中的选择键相匹配的部分。有关更多信息，请参阅全球生命周期示例。

> **Important**
>
> 仅当此清单未定义生命周期时，核心设备才会使用全局生命周期中的选项。

您可以指定`all`选择密钥来运行全局生命周期中没有选择键的部分。

`Artifacts`

（可选）对象列表，每个对象都为该清单所定义的平台上的组件定义了一个二进制构件。例如，您可以将代码或图像定义为工件。

部署组件时，AWS IoT GreengrassCore 软件会将构件下载到核心设备上的文件夹。您也可以将构件定义为软件下载后提取的存档文件。

您可以使用配方变量来获取核心设备上安装工件的文件夹的路径。

- 普通文件 — 使用 artifacts: path 配方变量获取包含工件的文件夹的路径。例如，在配方{artifacts:path}/my_script.py中指定以获取包含 URI 的工件的路径s3://*DOC-EXAMPLE-BUCKET*/path/to/my_script.py。
- 提取的档案 — 使用 art ifacts: decompressedPath 配方变量获取包含已提取存档工件的文件夹的路径。C AWS IoT Greengrass ore 软件将每个存档提取到与存档同名的文件夹中。例如，在配方{artifacts:decompressedPath}/my_archive/my_script.py中指定以获取包含 URI 的存档构件my_script.py中的路径s3://*DOC-EXAMPLE-BUCKET*/path/to/my_archive.zip。

> **Note**
>
> 在本地核心设备上开发带有存档构件的组件时，可能没有该构件的 URI。要使用提取构件的Unarchive选项测试组件，请指定一个文件名与存档构件文件名匹配的 URI。您可以指定要将存档项目上传到的 URI，也可以指定新的占位符 URI。例如，要在本地部署期间提取my_archive.zip对象，可以指定s3://*DOC-EXAMPLE-BUCKET*/my_archive.zip。

每个对象都包含以下信息：

URI

S3 存储桶中工件的 URI。安装组件时，AWS IoT GreengrassCore 软件会从此 URI 中获取工件，除非该构件已存在于设备上。每个项目在每个清单中都必须有一个唯一的文件名。

Unarchive

（可选）要解压缩的档案类型。从以下选项中进行选择：

- NONE— 该文件不是要解压的存档。C AWS IoT Greengrass ore 软件将构件安装到核心设备上的文件夹。你可以使用 artifacts: path 配方变量来获取此文件夹的路径。
- ZIP— 该文件是一个 ZIP 存档。C AWS IoT Greengrass ore 软件将存档提取到与存档同名的文件夹。你可以使用 artifacts: decompressedPath 配方变量来获取包含此文件夹的文件夹的路径。

默认值为 NONE。

Permission

（可选）一个对象，用于定义要为此构件文件设置的访问权限。您可以设置读取权限和执行权限。

Note

您无法设置写入权限，因为AWS IoT Greengrass核心软件不允许组件编辑构件文件夹中的构件文件。要编辑组件中的构件文件，请将其复制到其他位置或发布并部署新的构件文件。

如果您将工件定义为要解压缩的存档，则 AWS IoT Greengrass Core 软件会对其从存档中解压缩的文件设置这些访问权限。C AWS IoT Greengrass ore 软件将文件夹的访问权限设置ALL为Read和Execute。这允许组件查看文件夹中已解压缩的文件。要为存档中的单个文件设置权限，可以在安装生命周期脚本中设置权限。

该对象包含以下信息：

`Read`

（可选）要为此构件文件设置的读取权限。要允许其他组件访问此构件，例如依赖于此组件的组件，请指定ALL。从以下选项中进行选择：

- NONE— 文件不可读。
- OWNER— 您配置为运行此组件的系统用户可以读取该文件。
- ALL— 所有用户都可以读取该文件。

默认值为 OWNER。

`Execute`

（可选）要为此构件文件设置的运行权限。该Execute权限意味着Read权限。例如，如果您ALL为指定Execute，则所有用户都可以读取和运行此构件文件。

从以下选项中进行选择：

- NONE— 该文件无法运行。
- OWNER— 该文件可由您配置为运行该组件的系统用户运行。
- ALL— 该文件可供所有用户运行。

默认值为 NONE。

`Digest`

（只读）工件的加密摘要哈希。创建组件时，AWS IoT Greengrass使用哈希算法计算构件文件的哈希值。然后，在部署组件时，Greengrass nucleus 会计算已下载工件的哈希值，并将

哈希值与该摘要进行比较，以便在安装之前验证工件。如果哈希值与摘要不匹配，则部署失败。

如果您设置了此参数，则AWS IoT Greengrass会替换您在创建组件时设置的值。

`Algorithm`

（只读）AWS IoT Greengrass用于计算工件摘要哈希值的哈希算法。

如果您设置了此参数，则AWS IoT Greengrass会替换您在创建组件时设置的值。

`Lifecycle`

一个对象，用于定义如何安装和运行组件。只有在要使用的清单未指定生命周期时，核心设备才会使用全局生命周期。

> **Note**
>
> 你可以在清单之外定义这个生命周期。您还可以定义适用于与该清单匹配的平台的清单生命周期。

在全局生命周期中，您可以指定针对您在每个清单中指定的某些选择键运行的生命周期。选择键是字符串，用于标识要为每个清单运行的全局生命周期的各个部分。

`all`选择键是任何没有选择键的部分的默认值。这意味着您可以在清单中指定`all`选择密钥来运行全局生命周期的各个部分，而无需选择键。您无需在全局生命周期中指定`all`选择密钥。

如果清单未定义生命周期或选择密钥，则核心设备将默认使用该`all`选择。这意味着在这种情况下，核心设备使用全局生命周期中不使用选择键的部分。

此对象包含的信息与清单生命周期相同，但您可以在任何级别指定选择键来选择生命周期的子部分。

> **Tip**
>
> 我们建议您对每个选择键仅使用小写字母，以避免选择键和生命周期密钥之间发生冲突。生命周期密钥以大写字母开头。

Example 带有顶级选择键的全局生命周期示例

```
Lifecycle:
```

```
  key1:
    install:
      Skipif: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script: command3
```

Example 带有底层选择键的全局生命周期示例

```
Lifecycle:
  install:
    Script:
      key1: command1
      key2: command2
      all: command3
```

Example 具有多个选择键级别的全局生命周期示例

```
Lifecycle:
  key1:
    install:
      Skipif: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script:
        key3: command3
        key4: command4
        all: command5
```

食谱变量

配方变量显示来自当前组件和核的信息，供您在配方中使用。例如，您可以使用配方变量将组件配置参数传递给在生命周期脚本中运行的应用程序。

您可以在组件配方的以下部分中使用配方变量：

- 生命周期定义。
- 组件配置定义，前提是您使用 Greengrass nucleus v2.6.0 或更高版本并将配置选项设置为。interpolateComponentConfiguration`true`在部署组件配置更新时，也可以使用配方变量。

配方变量使用`{recipe_variable}`语法。大括号表示配方变量。

AWS IoT Greengrass支持以下配方变量：

`*component_dependency_name*:configuration:*json_pointer*`

此配方定义的组件或该组件所依赖的组件的配置参数值。

您可以使用此变量为在组件生命周期中运行的脚本提供参数。

> **Note**
>
> AWS IoT Greengrass仅在组件生命周期定义中支持此配方变量。

此配方变量具有以下输入：

- `component_dependency_name`—（可选）要查询的组件依赖项的名称。省略此区段可查询此配方定义的组件。您只能指定直接依赖关系。
- `json_pointer`— 指向要评估的配置值的 JSON 指针。JSON 指针以正斜杠/开头。要标识嵌套组件配置中的值，请使用正斜杠 (/) 分隔配置中每个级别的键。您可以使用数字作为键来指定列表中的索引。有关更多信息，请参阅 JSON 指针规范。

  AWS IoT GreengrassCore 使用 JSON 指针获取 YAML 格式的食谱。

JSON 指针可以引用以下节点类型：

- 一个值节点。 AWS IoT GreengrassCore 将配方变量替换为该值的字符串表示形式。空值转换为`null`字符串。
- 一个对象节点。 AWS IoT GreengrassCore 将配方变量替换为该对象的序列化 JSON 字符串表示形式。
- 没有节点。 AWS IoT GreengrassCore 不会替换配方变量。

例如，{configuration:/Message}配方变量检索组件配置中Message键的值。{com.example.MyComponentDependency:configuration:/server/port}配方变量在组件依赖关系的server配置对象port中检索的值。

component_dependency_name:artifacts:path

此配方定义的组件或该组件所依赖的组件的工件根路径。

安装组件时，会将该组件的构件AWS IoT Greengrass复制到此变量公开的文件夹。例如，您可以使用此变量来标识要在组件生命周期中运行的脚本的位置。

此路径下的文件夹为只读文件夹。要修改构件文件，请将文件复制到其他位置，例如当前工作目录（$PWD或.）。然后，修改那里的文件。

要从组件依赖项中读取或运行工件，则该构件Read或Execute权限必须是ALL。有关更多信息，请参阅您在组件配方中定义的构件权限。

此配方变量具有以下输入：

- component_dependency_name—（可选）要查询的组件依赖项的名称。省略此区段可查询此配方定义的组件。您只能指定直接依赖关系。

component_dependency_name:artifacts:decompressedPath

此配方定义的组件或该组件所依赖的组件的解压缩存档工件的根路径。

安装组件后，将该组件的存档工件AWS IoT Greengrass解压缩到此变量公开的文件夹。例如，您可以使用此变量来标识要在组件生命周期中运行的脚本的位置。

每个工件都会解压缩到解压缩路径内的一个文件夹，该文件夹与构件同名，减去其扩展名。例如，名为的 ZIP 工件models.zip解压缩到该{artifacts:decompressedPath}/models文件夹。

此路径下的文件夹为只读文件夹。要修改构件文件，请将文件复制到其他位置，例如当前工作目录（$PWD或.）。然后，修改那里的文件。

要从组件依赖项中读取或运行工件，则该构件Read或Execute权限必须是ALL。有关更多信息，请参阅您在组件配方中定义的构件权限。

此配方变量具有以下输入：

- component_dependency_name—（可选）要查询的组件依赖项的名称。省略此区段可查询此配方定义的组件。您只能指定直接依赖关系。

component_dependency_name:work:path

此功能适用于 2.0.4 及更高版本的 Greengrass nucleus 组件。

此配方定义的组件或该组件所依赖的组件的工作路径。从组件的上下文中运行时，此配方变量的值等同于$PWD环境变量和 pwd 命令的输出。

您可以使用此配方变量在组件和依赖项之间共享文件。

此路径下的文件夹可由此配方定义的组件以及以同一用户和组身份运行的其他组件进行读写操作。

此配方变量具有以下输入：

- component_dependency_name—（可选）要查询的组件依赖项的名称。省略此区段可查询此配方定义的组件。您只能指定直接依赖关系。

kernel:rootPath

AWS IoT Greengrass核心根路径。

iot:thingName

此功能适用于 2.3.0 及更高版本的 Greengrass nucleus 组件。

核心设备的名称AWS IoT。

食谱示例

您可以参考以下配方示例，以帮助您为组件创建配方。

AWS IoT Greengrass策划了一份名为 Greengrass 软件目录的 Greengrass 组件索引。该目录追踪了 Greengrass 社区开发的 Greengrass 组件。您可以从该目录中下载、修改和部署组件来创建 Greengrass 应用程序。有关更多信息，请参阅 社区组件。

主题

- 你好 World 组件配方
- Python 运行时组件示例
- 指定多个字段的组件配方

你好 World 组件配方

以下配方描述了一个运行 Python 脚本的 Hello World 组件。该组件支持所有平台，并接受作为Message参数AWS IoT Greengrass传递给 Python 脚本的参数。这是入门教程中 Hello World 组件的配方。

JSON

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

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
```

```
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Python 运行时组件示例

以下配方描述了一个安装 Python 的组件。此组件支持 64 位 Linux 设备。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PythonRuntime",
  "ComponentDescription": "Installs Python 3.7",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "3.7.0",
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": "apt-get update\napt-get install python3.7"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
```

```
ComponentName: com.example.PythonRuntime
ComponentDescription: Installs Python 3.7
ComponentPublisher: Amazon
ComponentVersion: '3.7.0'
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install: |
        apt-get update
        apt-get install python3.7
```

指定多个字段的组件配方

以下组件配方使用多个配方字段。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.FooService",
  "ComponentDescription": "Complete recipe for AWS IoT Greengrass components",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "1.0.0",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "TestParam": "TestValue"
    }
  },
  "ComponentDependencies": {
    "BarService": {
      "VersionRequirement": "^1.1.0",
      "DependencyType": "SOFT"
    },
    "BazService": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
```

```
          "architecture": "amd64"
        },
        "Lifecycle": {
          "install": {
            "Skipif": "onpath git",
            "Script": "sudo apt-get install git"
          },
          "Setenv": {
            "environment_variable1": "variable_value1",
            "environment_variable2": "variable_value2"
          }
        },
        "Artifacts": [
          {
            "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world.zip",
            "Unarchive": "ZIP"
          },
          {
            "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world_linux.py"
          }
        ]
      },
      {
        "Lifecycle": {
          "install": {
            "Skipif": "onpath git",
            "Script": "sudo apt-get install git",
            "RequiresPrivilege": "true"
          }
        },
        "Artifacts": [
          {
            "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world.py"
          }
        ]
      }
    ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
```

```
ComponentName: com.example.FooService
ComponentDescription: Complete recipe for AWS IoT Greengrass components
ComponentPublisher: Amazon
ComponentVersion: 1.0.0
ComponentConfiguration:
  DefaultConfiguration:
    TestParam: TestValue
ComponentDependencies:
  BarService:
    VersionRequirement: ^1.1.0
    DependencyType: SOFT
  BazService:
    VersionRequirement: ^2.0.0
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install:
        Skipif: onpath git
        Script: sudo apt-get install git
      Setenv:
        environment_variable1: variable_value1
        environment_variable2: variable_value2
    Artifacts:
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world.zip'
        Unarchive: ZIP
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world_linux.py'
  - Lifecycle:
      install:
        Skipif: onpath git
        Script: sudo apt-get install git
        RequiresPrivilege: 'true'
    Artifacts:
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world.py'
```

组件环境变量引用

AWS IoT Greengrass核心软件在为组件运行生命周期脚本时设置环境变量。你可以在组件中获取这些环境变量来获得事物名称、和 GreengAWS 区域 rass nucleus 版本。该软件还设置您的组件使用进程间通信 SDK 和与AWS服务交互所需的环境变量。

您还可以为组件的生命周期脚本设置自定义环境变量。有关更多信息，请参阅 Setenv。

AWS IoT GreengrassCore 软件设置以下环境变量：

`AWS_IOT_THING_NAME`

代表这个 Greengrass 核心设备的AWS IoT东西的名字。

`AWS_REGION`

这款 Greengrass 核心设备的运行AWS 区域地点。

SAWS DK 使用此环境变量来标识要使用的默认区域。此变量等效于`AWS_DEFAULT_REGION`。

`AWS_DEFAULT_REGION`

这款 Greengrass 核心设备的运行AWS 区域地点。

AWS CLI使用此环境变量来标识要使用的默认区域。此变量等效于`AWS_REGION`。

`GGC_VERSION`

在这款 Greengrass 核心设备上运行的 Greengrass 核心组件的版本。

`GG_ROOT_CA_PATH`

此功能适用于 Greengrass nucleus 组件的 v2.5.5 及更高版本。

Greengrass 核心要使用的根证书颁发机构 (CA) 证书的路径。

`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`

组件用于与AWS IoT Greengrass Core 软件通信的 IPC 套接字路径。有关更多信息，请参阅使用AWS IoT Device SDK与 Greengrass 原子核、其他组件进行通信 AWS IoT Core：

`SVCUID`

组件用来连接到 IPC 套接字并与AWS IoT Greengrass Core 软件通信的秘密令牌。有关更多信息，请参阅使用AWS IoT Device SDK与 Greengrass 原子核、其他组件进行通信 AWS IoT Core：

`AWS_CONTAINER_AUTHORIZATION_TOKEN`

组件用来从令牌交换服务组件检索凭证的密钥令牌。

`AWS_CONTAINER_CREDENTIALS_FULL_URI`

组件请求从令牌交换服务组件检索凭证的 URI。

将AWS IoT Greengrass组件部署到设备

您可以使用AWS IoT Greengrass将组件部署到设备或设备组。您可以使用部署来定义发送到设备的组件和配置。 AWS IoT Greengrass部署到代表 Greengrass 核心设备的目标、AWS IoT事物或事物组。AWS IoT Greengrass使用AWS IoT Core作业部署到您的核心设备。您可以配置任务如何部署到您的设备。

核心设备部署

每台核心设备都运行该设备的部署组件。对同一目标的新部署会覆盖先前部署到该目标的部署。创建部署时，您可以定义要应用于核心设备现有软件的组件和配置。

修改目标的部署时，会将先前修订版中的组件替换为新修订版中的组件。例如，您将日志管理器和秘密经理组件部署到事物组`TestGroup`。然后，您为`TestGroup`其创建另一个部署，该部署仅指定密钥管理器组件。因此，该组中的核心设备不再运行日志管理器。

平台依赖关系解决方案

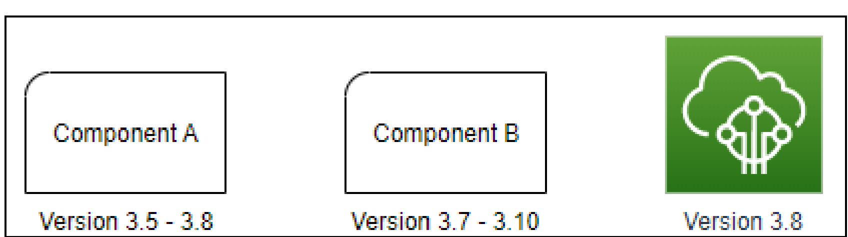
当核心设备收到部署时，它会进行检查以确保组件与核心设备兼容。例如，如果您将部署Firehose到 Windows 目标，则部署将失败。

组件依赖关系解析

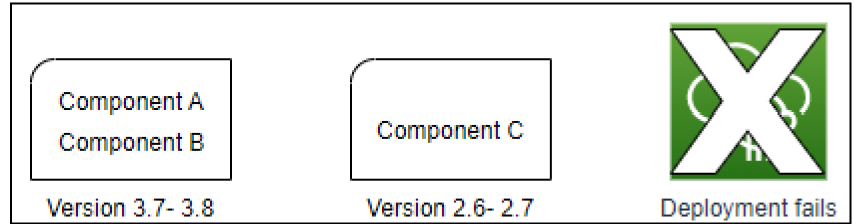
核心设备还会检查每个组件的依赖关系是否与将其他组件部署到该事物组的版本限制兼容。如果组件的版本限制重叠，Greengrass 将使用该组件的最高适用版本。例如：

- 将组件 A 部署到`TestGroup`。组件 A 取决于组件`com.example.PythonRuntime`版本 3.5-3.10。
- 然后，将组件 B 部署到`TestGroup`。组件 B 依赖于 3.7 到 3.8 `com.example.PythonRuntime` 版本的组件。

因此，核心设备`TestGroup`决定可以部署该`com.example.PythonRuntime`组件的 3.8 版，因为此版本是版本限制重叠的最高适用版本。



然后将组件 C 部署到`TestGroup`。组件 C 依赖于组件`com.example.PythonRuntime`版本 2.6-2.7。此部署失败，因为没有满足约束 2.6-2.7 和 3.7-3.8 的组件版本。



从事物组中移除设备

从事物组中移除核心设备时，组件部署行为取决于核心设备运行的 Greengrass 核心版本。

2.5.1 and later

从事物组中移除核心设备时，其行为取决于AWS IoT策略是否授予`greengrass:ListThingGroupsForCoreDevice`权限。有关此权限和核心设备AWS IoT策略的更多信息，请参阅AWS IoT Greengrass 的设备身份验证和授权。

- 如果AWS IoT策略授予此权限

  当您从事物组中移AWS IoT Greengrass除核心设备时，会在下次对该设备进行部署时移除该事物组的组件。如果设备上的某个组件包含在下次部署中，则该组件不会从设备中删除。

- 如果AWS IoT策略未授予此权限

  从事物组中移除核心设备时，AWS IoT Greengrass不会从设备中删除该事物组的组件。

  要从设备中移除组件，请使用 Greengrass CLI 的部署创建命令。使用`--remove`参数指定要移除的组件，并使用`--groupId`参数指定事物组。

2.5.0

当您从事物组中移AWS IoT Greengrass除核心设备时，会在下次对该设备进行部署时移除该事物组的组件。如果设备上的某个组件包含在下次部署中，则该组件不会从设备中删除。

此行为需要核心设备的AWS IoT策略授予`greengrass:ListThingGroupsForCoreDevice`权限。如果核心设备没有此权限，则该核心设备将无法应用部署。有关更多信息，请参阅 AWS IoT Greengrass 的设备身份验证和授权。

2.0.x - 2.4.x

从事物组中移除核心设备时，AWS IoT Greengrass不会从设备中删除该事物组的组件。

要从设备中移除组件，请使用 Greengrass CLI 的部署创建命令。使用`--remove`参数指定要移除的组件，并使用`--groupId`参数指定事物组。

部署

部署是持续的。创建部署时，AWS IoT Greengrass会将部署部署部署部署部署到在线的目标设备。如果目标设备未在线，则它将在下次连接时收到部署AWS IoT Greengrass。将核心设备添加到目标事物组时，AWS IoT Greengrass会向该设备发送该事物组的最新部署。

默认情况下，在核心设备部署组件之前，它会通知设备上的每个组件。Greengrass 组件可以响应推迟部署的通知。如果设备电池电量不足或正在运行无法中断的进程，则可能需要推迟部署。有关更多信息，请参阅 教程：开发一个可以延迟组件更新的 Greengrass 组件。创建部署时，您可以将其配置为在不通知组件的情况下进行部署。

每个目标事物或事物组一次只能有一个部署。这意味着，当您为目标创建部署时，将AWS IoT Greengrass不再部署该目标部署的先前版本。

部署选项

部署提供了多个选项，可让您控制哪些设备接收更新以及如何部署更新。创建部署时，可以配置以下选项：

- AWS IoT Greengrass组件

  定义要在目标设备上安装和运行的组件。 AWS IoT Greengrass组件是您在 Greengrass 核心设备上部署和运行的软件模块。只有当组件支持设备的平台时，设备才会接收组件。这样，即使目标设备在多个平台上运行，也可以部署到设备组。如果某个组件不支持该设备的平台，则该组件不会部署到设备上。

  您可以将自定义组件和AWS提供的组件部署到您的设备上。部署组件时，会AWS IoT Greengrass识别所有组件依赖关系并将其部署。有关更多信息，请参阅 开发AWS IoT Greengrass组件和 AWS-提供的组件。

  您可以为每个组件定义要部署的版本和配置更新。配置更新指定了如何在核心设备上修改组件的现有配置，或者如果核心设备上不存在该组件，则如何修改组件的默认配置。您可以指定要重置为默认值的配置值以及要合并到核心设备上的新配置值。当核心设备收到针对不同目标的部署，并且每个部署

都指定了兼容的组件版本时，核心设备会根据您创建部署的时间戳按顺序应用配置更新。有关更多信息，请参阅 更新组件配置。

> **Important**
>
> 部署组件时，AWS IoT Greengrass会安装该组件所有依赖项的最新支持版本。因此，如果您向事物组中添加新设备或更新针对这些设备的部署，则AWS提供的公共组件的新补丁版本可能会自动部署到您的核心设备上。某些自动更新（例如 nucleus 更新）可能会导致您的设备意外重启。
>
> 为防止设备上运行的组件出现意外更新，我们建议您在创建部署时直接包含该组件的首选版本。有关 C AWS IoT Greengrass ore 软件更新行为的更多信息，请参阅更新AWS IoT Greengrass核心软件 (OTA)。

- 部署策略

  定义何时可以安全部署配置以及部署失败时该怎么做。您可以指定是否等待组件报告它们可以更新。您还可以指定如果设备应用的部署失败，是否将其还原到其先前的配置。

- 停止配置

  定义何时以及如何停止部署。如果满足您定义的标准，则部署将停止并失败。例如，您可以将部署配置为在最少数量的设备收到部署后仍有一定百分比的设备未能应用该部署时停止。

- 推出配置

  定义部署部署到目标设备的速率。您可以配置具有最小和最大速率界限的指数速率提升。

- 超时配置

  定义每台设备应用部署的最长时间。如果设备超过了您指定的持续时间，则该设备将无法应用部署。

> **Important**
>
> 自定义组件可以在 S3 存储桶中定义项目。当AWS IoT Greengrass核心软件部署组件时，它会从中下载该组件的工件。AWS Cloud默认情况下，核心设备角色不允许访问 S3 存储桶。要部署在 S3 存储桶中定义对象的自定义组件，核心设备角色必须授予从该存储桶下载项目的权限。有关更多信息，请参阅 允许访问 S3 存储桶以获取组件项目。

主题

- 创建部署
- 创建子部署
- 修改部署
- 取消作业
- 检查部署状态

创建部署

您可以创建以事物或事物组为目标的部署。

创建部署时，需要配置要部署的软件组件以及部署任务如何部署到目标设备。您可以在提供给的 JSON 文件中定义部署AWS CLI。

部署目标决定了要在哪些设备上运行组件。要部署到一台核心设备，请指定一个事物。要部署到多台核心设备，请指定包含这些设备的事物组。有关如何配置事物组的更多信息，请参阅《AWS IoT开发人员指南》中的静态事物组和动态事物组。

按照本节中的步骤创建到目标的部署。有关如何在已部署的目标上更新软件组件的更多信息，请参阅修改部署。

> **Warning**
>
> 该CreateDeployment操作可以从核心设备上卸载组件。如果之前的部署中存在某个组件，而不是新部署中存在组件，则核心设备将卸载该组件。为避免卸载组件，请先使用该ListDeployments操作来检查部署目标是否已有部署。然后，在创建新部署时，使用该GetDeployment操作从现有部署开始。

创建部署 (AWS CLI)

1. 创建一个名为的文件`deployment.json`，然后将以下 JSON 对象复制到该文件中。将 *`targetArn`* 替换为部署目标的事物或事物组AWS IoT的 ARN。事物和事物组 ARN 的格式如下：

   - 事物：`arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
   - 事物组：`arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*

   ```
   {
   ```

```
  "targetArn": "targetArn"
}
```

2. 检查部署目标是否有要修改的现有部署。执行以下操作：

   a. 运行以下命令，列出部署目标的部署。将 *targetArn* 替换为目标事物或事物组的 AWS IoT ARN。

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   响应中包含目标最新部署的列表。如果响应为空，则表示目标没有现有部署，您可以跳至Step 3。否则，请复制响应中的 `deploymentId`，以便在下一步中使用。

   > **Note**
   >
   > 您还可以修改除目标最新版本之外的部署。指定`--history-filter ALL`参数以列出目标的所有部署。然后，复制要修改的部署的 ID。

   b. 运行以下命令，获取部署的详细信息。这些详细信息包括元数据、组件和作业配置。将 *deplomentId* 替换为上一步中的 ID。

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   响应包含部署的详细信息。

   c. 将上一个命令的响应中的以下任意键值对复制到。`deployment.json`您可以为新部署更改这些值。

   - `deploymentName`— 部署的名称。
   - `components`— 部署的组件。要卸载组件，请将其从该对象中移除。
   - `deploymentPolicies`— 部署的策略。
   - `iotJobConfiguration`— 部署的任务配置。
   - `tags`— 部署的标签。

3. （可选）定义部署的名称。将 deplo *ymentName #####*名称。

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName"
```

```
}
```

4. 添加每个组件以部署目标设备。为此，请向components对象添加键值对，其中键是组件名称，值是包含该组件详细信息的对象。为您添加的每个组件指定以下详细信息：

   - version— 要部署的组件版本。
   - configurationUpdate— 要部署的配置更新。更新是一项修补操作，用于修改组件在每台目标设备上的现有配置，或者修改组件的默认配置（如果目标设备上不存在该配置）。您可以指定以下配置更新：
     - 重置更新 (reset)-（可选）JSON 指针列表，用于定义要在目标设备上重置为默认值的配置值。AWS IoT Greengrass 核心软件会在其应用合并更新前应用重置更新。有关更多信息，请参阅 重置更新。
     - 合并更新 (merge)-（可选）一个 JSON 文档，用于定义要合并到目标设备的配置值。必须将 JSON 文档序列化为字符串。有关更多信息，请参阅 合并更新。
   - runWith—（可选）AWS IoT Greengrass核心软件用于在核心设备上运行此组件进程的系统进程选项。如果您省略了runWith对象中的参数，则 AWS IoT Greengrass Core 软件将使用您在Gre engrass nucleus 组件上配置的默认值。

     您可以指定以下任一选项：

     - posixUser— 用于在 Linux 核心设备上运行此组件的 POSIX 系统用户和（可选）组。用户和组（如果已指定）必须存在于每台 Linux 核心设备上。使用以下格式指定由半角冒号（:）分隔的用户和组：user:group。组是可选的。如果您未指定群组，则 AWS IoT Greengrass Core 软件将使用该用户的主群组。有关更多信息，请参阅 配置运行组件的用户。
     - windowsUser— 用于在 Windows 核心设备上运行此组件的 Windows 用户。用户必须存在于每台 Windows 核心设备上，其用户名和密码必须存储在 LocalSystem 账户的凭据管理器实例中。有关更多信息，请参阅 配置运行组件的用户。

       此功能适用于 Greengrass nucleus 组件的 2.5.0 及更高版本。
     - systemResourceLimits— 适用于此组件进程的系统资源限制。您可以将系统资源限制应用于通用和非容器化 Lambda 组件。有关更多信息，请参阅 为组件配置系统资源限制。

       您可以指定以下任一选项：

       - cpus— 此组件的进程可以在核心设备上使用的最大 CPU 时间。核心设备的总 CPU 时间等于 CPU 核心的设备数量。例如，在具有 4 个 CPU 内核的核心设备上，您可以将此值设置为，2将该组件的进程使用率限制为每个 CPU 内核的 50%。在具有 1 个 CPU 内核的设备上，您可以将此值设置为，0.25将此组件的进程使用率限制在 25% 以内。如果将此值设

置为大于 CPU 内核数的数字，则 AWS IoT Greengrass Core 软件不会限制组件的 CPU 使用率。

- memory— 此组件的进程可以在核心设备上使用的最大 RAM 量（以千字节为单位）。

此功能适用于 Greengrass nucleus 组件的 v2.4.0 及更高版本。 AWS IoT Greengrass目前不支持在 Windows 核心设备上使用此功能。

Example 基本配置更新示例

以下示例components对象指定部署一个需要名为的配置参数的组件pythonVersion。com.example.PythonRuntime

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.PythonRuntime": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"pythonVersion\":\"3.7\"}"
      }
    }
  }
}
```

Example 使用重置和合并更新的配置更新示例

以一个具有以下默认配置的工业仪表板组件为例。com.example.IndustrialDashboard

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
```

```
  "tags": []
}
```

以下配置更新指定了以下说明：

1. 将 HTTPS 设置重置为其默认值 (`true`)。
2. 将工业标签列表重置为空列表。
3. 合并标识两台锅炉温度和压力数据流的工业标签列表。

```
{
  "reset": [
    "/network/useHttps",
    "/tags"
  ],
  "merge": {
    "tags": [
      "/boiler/1/temperature",
      "/boiler/1/pressure",
      "/boiler/2/temperature",
      "/boiler/2/pressure"
    ]
  }
}
```

以下示例`components`对象指定部署此工业仪表板组件和配置更新。

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
```

```
      }
    }
  }
```

5. （可选）为部署定义部署策略。您可以配置核心设备何时可以安全地应用部署，或者在核心设备无法应用部署时该怎么做。为此，请向中添加一个deploymentPolicies对象deployment.json，然后执行以下任一操作：

   1. （可选）指定组件更新策略（componentUpdatePolicy）。此策略定义部署是否允许组件将更新推迟到准备好更新之前。例如，组件可能需要清理资源或完成关键操作，然后才能重新启动以应用更新。该策略还定义了组件响应更新通知所需的时间。

      此策略是一个具有以下参数的对象：

      - action—（可选）是否通知组件并等待它们在准备更新时报告。从以下选项中进行选择：
        - NOTIFY_COMPONENTS – 部署会在其停止并更新该组件前通知每个组件。组件可以使用 SubscribeToComponentUpdates IPC 操作来接收这些通知。
        - SKIP_NOTIFY_COMPONENTS – 部署不会通知组件或等待它们可安全更新。

        默认值为 NOTIFY_COMPONENTS。
      - timeoutInSeconds每个组件通过 DeferComponentUpdate IPC 操作响应更新通知所需的时间（以秒为单位）。如果组件在这段时间内没有响应，则会在核心设备上继续部署。

        默认为 60 秒。

   2. （可选）指定配置验证策略（configurationValidationPolicy）。此策略定义了每个组件需要多长时间才能验证部署中的配置更新。组件可以使用 SubscribeToValidateConfigurationUpdates IPC 操作订阅自己的配置更新的通知。然后，组件可以使用 SendConfigurationValidityReport IPC 操作来告诉 C AWS IoT Greengrass ore 软件配置更新是否有效。如果配置更新无效，则部署将失败。

      此策略是一个具有以下参数的对象：

      - timeoutInSeconds（可选）每个组件验证配置更新所需的时间（以秒为单位）。如果组件在这段时间内没有响应，则会在核心设备上继续部署。

        默认为 30 秒。

   3. （可选）指定故障处理策略（failureHandlingPolicy）。此策略是一个字符串，用于定义部署失败时是否回滚设备。从以下选项中进行选择：

      - ROLLBACK— 如果在核心设备上部署失败，则AWS IoT Greengrass核心软件会将该核心设备回滚到以前的配置。

- DO_NOTHING— 如果在核心设备上部署失败，则AWS IoT Greengrass核心软件将保留新的配置。如果新配置无效，这可能会导致组件损坏。

默认值为 ROLLBACK。

您在中的部署deployment.json可能与以下示例类似：

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  }
}
```

6. （可选）定义部署如何停止、推出或超时。 AWS IoT Greengrass使用AWS IoT Core作业将部署发送到核心设备，因此这些选项与AWS IoT Core作业的配置选项相同。有关更多信息，请参阅《AWS IoT开发者指南》中的 Job 部署和中止配置。

   要定义作业选项，请向中添加iotJobConfiguration对象deployment.json。然后，定义要配置的选项。

您在中的部署deployment.json可能与以下示例类似：

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  },
  "iotJobConfiguration": {
    "abortConfig": {
      "criteriaList": [
        {
          "action": "CANCEL",
          "failureType": "ALL",
          "minNumberOfExecutedThings": 100,
          "thresholdPercentage": 5
        }
      ]
    },
    "jobExecutionsRolloutConfig": {
      "exponentialRate": {
        "baseRatePerMinute": 5,
        "incrementFactor": 2,
```

```
          "rateIncreaseCriteria": {
            "numberOfNotifiedThings": 10,
            "numberOfSucceededThings": 5
          }
        },
        "maximumPerMinute": 50
      },
      "timeoutConfig":  {
        "inProgressTimeoutInMinutes": 5
      }
    }
  }
```

7. （可选）为部署添加标签 (tags)。有关更多信息，请参阅 标记 AWS IoT Greengrass Version 2 资源。
8. 运行以下命令从中创建部署deployment.json。

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

   响应中deploymentId包含用于标识此部署的。您可以使用部署 ID 来检查部署的状态。有关更多信息，请参阅 检查部署状态。

更新组件配置

组件配置是定义每个组件参数的 JSON 对象。每个组件的配方都定义了其默认配置，当您将组件部署到核心设备时，您可以修改这些配置。

创建部署时，可以指定要应用于每个组件的配置更新。配置更新是补丁操作，这意味着更新会修改核心设备上存在的组件配置。如果核心设备没有该组件，则配置更新会修改并应用该部署的默认配置。

配置更新定义了重置更新和合并更新。重置更新定义了要将哪些配置值重置为默认值或移除哪些配置值。合并更新定义要为组件设置的新配置值。部署配置更新时， AWS IoT Greengrass Core 软件会在合并更新之前运行重置更新。

组件可以验证您部署的配置更新。当部署更改其配置时，该组件订阅后会收到通知，并且它可以拒绝其不支持的配置。有关更多信息，请参阅 与组件配置交互。

主题

- 重置更新
- 合并更新

- 示例

重置更新

重置更新定义了核心设备上要将哪些配置值重置为默认值。如果配置值没有默认值，则重置更新会将该值从组件的配置中删除。这可以帮助您修复因配置无效而中断的组件。

使用 JSON 指针列表来定义要重置哪些配置值。JSON 指针以正斜杠/开头。要标识嵌套组件配置中的值，请使用正斜杠 (/) 分隔配置中每个级别的键。有关更多信息，请参阅 JSON 指针规范。

> **Note**
>
> 您只能将整个列表重置为其默认值。您不能使用重置更新来重置列表中的单个元素。

要将组件的整个配置重置为其默认值，请指定一个空字符串作为重置更新。

```
"reset": [""]
```

合并更新

合并更新定义要插入核心组件配置的配置值。合并更新是一个 JSON 对象， AWS IoT Greengrass 核心软件会在重置您在重置更新中指定的路径中重置值后合并该对象。使用 AWS CLI 或 AWS 软件开发工具包时，必须将此 JSON 对象序列化为字符串。

您可以合并组件默认配置中不存在的键值对。您也可以合并类型与具有相同键的值不同的键值对。新值将替换旧值。这意味着您可以更改配置对象的结构。

您可以合并空值和空字符串、列表和对象。

> **Note**
>
> 不能将合并更新用于在列表中插入元素或将元素追加到列表中。您可以替换整个列表，也可以定义一个对象，其中每个元素都有一个唯一的键。
>
> AWS IoT Greengrass 使用 JSON 作为配置值。JSON 指定了数字类型，但不区分整数和浮点数。因此，配置值可能会转换为浮点数。 AWS IoT Greengrass为确保您的组件使用正确的数据类型，我们建议您将数字配置值定义为字符串。然后，让你的组件将它们解析为整数或浮点数。这样可以确保您的配置值在配置和核心设备上具有相同的类型。

在合并更新中使用配方变量

此功能适用于 Greengrass nucleus 组件的 2.6.0 及更高版本。

如果您将 Greengrass n interpolateComponentConfigurationucleus 的配置选项设置为，则可以在合并更新中使用配方变量true以外的配*component_dependency_name*:configuration:*json_pointer*方变量。例如，您可以在合并更新中使用{iot:thingName}配方变量，将核心设备 AWS IoT 的事物名称包含在组件配置值（例如进程间通信 (IPC) 授权策略）中。

示例

以下示例演示了具有以下默认配置的仪表板组件的配置更新。此示例组件显示有关工业设备的信息。

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

工业仪表板组件配方

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IndustrialDashboard",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Displays information about industrial equipment.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "name": null,
      "mode": "REQUEST",
      "network": {
```

```
        "useHttps": true,
        "port": {
          "http": 80,
          "https": 443
        },
      },
      "tags": []
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/industrial_dashboard.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/industrial_dashboard.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IndustrialDashboard
ComponentVersion: '1.0.0'
ComponentDescription: Displays information about industrial equipment.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    name: null
    mode: REQUEST
    network:
      useHttps: true
```

```
      port:
        http: 80
        https: 443
    tags: []
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/industrial_dashboard.py
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/industrial_dashboard.py
```

Example 示例 1：合并更新

您可以创建应用以下配置更新的部署，该更新指定合并更新，但不指定重置更新。此配置更新告诉组件在 HTTP 端口 8080 上显示仪表板，其中包含来自两个锅炉的数据。

Console

要合并的配置

```
{
  "name": "Factory 2A",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

AWS CLI

以下命令创建对核心设备的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

该dashboard-deployment.json文件包含以下 JSON 文档。

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"Factory 2A\",\"network\":{\"useHttps\":false,\"port
\":{\"http\":8080}},\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

以下 Greengrass CLI 命令在核心设备上创建本地部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration.json
```

该dashboard-configuration.json文件包含以下 JSON 文档。

```
{
  "com.example.IndustrialDashboard": {
    "MERGE": {
      "name": "Factory 2A",
      "network": {
        "useHttps": false,
```

```
        "port": {
          "http": 8080
        }
      },
      "tags": [
        "/boiler/1/temperature",
        "/boiler/1/pressure",
        "/boiler/2/temperature",
        "/boiler/2/pressure"
      ]
    }
  }
}
```

此更新后，仪表板组件具有以下配置。

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

Example 示例 2：重置和合并更新

然后，您创建一个应用以下配置更新的部署，该更新指定了重置更新和合并更新。这些更新指定在默认的 HTTPS 端口上显示仪表板，其中包含来自不同锅炉的数据。这些更新修改了上一个示例中配置更新所产生的配置。

Console

重置路径

```
[
  "/network/useHttps",
  "/tags"
]
```

要合并的配置

```
{
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure"
  ]
}
```

AWS CLI

以下命令创建对核心设备的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-
deployment2.json
```

该dashboard-deployment2.json文件包含以下 JSON 文档。

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
```

```
        "merge": "{\"tags\":[\"/boiler/3/temperature\",\"/boiler/3/pressure\",\"/
boiler/4/temperature\",\"/boiler/4/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

以下 Greengrass CLI 命令在核心设备上创建本地部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration2.json
```

该`dashboard-configuration2.json`文件包含以下 JSON 文档。

```
{
  "com.example.IndustrialDashboard": {
    "RESET": [
      "/network/useHttps",
      "/tags"
    ],
    "MERGE": {
      "tags": [
        "/boiler/3/temperature",
        "/boiler/3/pressure",
        "/boiler/4/temperature",
        "/boiler/4/pressure"
      ]
    }
  }
}
```

此更新后，仪表板组件具有以下配置。

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
```

```
  "network": {
    "useHttps": true,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure",
  ]
}
```

创建子部署

> **Note**
>
> 子部署功能在 Greengrass nucleus 版本 2.9.0 及更高版本上可用。无法使用早期组件版本的 Greengrass nucleus 将配置部署到子部署。

子部署是一种针对父部署中较小一部分设备的部署。您可以使用子部署将配置部署到较小的设备子集。您还可以创建子部署，以便在父部署中的一台或多台设备失败时重试失败的父部署。使用此功能，您可以选择在该父部署中失败的设备并创建子部署来测试配置，直到子部署成功。子部署成功后，您可以将该配置重新部署到父部署。

按照本节中的步骤创建子部署并检查其状态。有关如何创建部署的更多信息，请参阅创建部署。

创建子部署 () AWS CLI

1. 运行以下命令以检索事物组的最新部署。将命令中的 ARN 替换为要查询的事物组的 ARN。设置`--history-filter`**`LATEST_ONLY`**为可查看该事物组的最新部署。

   ```
   aws greengrassv2 list-deployments --target-arn arn:aws:iot:region:account-id:thinggroup/thingGroupName --history-filter LATEST_ONLY
   ```

2. 将响应`deploymentId`中的命令复制到list-deployments命令中，以便在下一步中使用。
3. 运行以下命令以检索部署的状态。*`deploymentId`*替换为要查询的部署的 ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

4. 将响应iotJobId中的命令复制到get-deployment命令中，以便在接下来的步骤中使用。
5. 运行以下命令以检索指定作业的任务执行列表。将 *jobId* 替换为iotJobId上一步中的。将*##*替换为您要筛选的状态。您可以筛选具有以下状态的结果：

   - QUEUED
   - IN_PROGRESS
   - SUCCEEDED
   - FAILED
   - TIMED_OUT
   - REJECTED
   - REMOVED
   - CANCELED

   ```
   aws iot list-job-executions-for-job --job-id jobID --status status
   ```

6. 为子部署创建新AWS IoT事物组或使用现有事物组。然后，向该AWS IoT事物组中添加一个事物。您可以使用事物组来管理 Greengrass 核心设备群。将软件组件部署到设备时，可以将单个设备或设备组作为目标。您可以通过激活 Greengrass 部署将设备添加到事物组。添加后，您可以将该事物组的软件组件部署到该设备上。

   要创建新事物组并将您的设备添加到该群组，请执行以下操作：

   a. 创建AWS IoT事物组。*MyGreengrassCoreGroup*替换为新事物组的名称。不能在事物组名称中使用冒号 (:)。

      Note

      如果将子部署的事物组与一个事物组一起使用parentTargetArn，则无法在其他父队列中重复使用该事物组。如果事物组已被用于为另一个队列创建子部署，则 API 将返回错误。

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

如果请求成功，则响应类似于以下示例：

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. 将已配置的 Greengrass 核心添加到您的事物组中。使用这些参数运行以下命令：

- *MyGreengrassCore*替换为已配置的 Greengrass 核心的名称。
- *MyGreengrassCoreGroup*替换为事物组的名称。

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

如果请求成功，则该命令没有任何输出。

7. 创建一个名为的文件`deployment.json`，然后将以下 JSON 对象复制到该文件中。将 *targetArn* 替换为子部署目标的事物组AWS IoT的 ARN。子部署目标只能是事物组。事物组 ARN 的格式如下：

   - 事物组 — `arn:aws:iot:region:account-id:thinggroup/thingGroupName`

```
{
  "targetArn": "targetArn"
}
```

8. 再次运行以下命令以获取原始部署的详细信息。这些详细信息包括元数据、组件和作业配置。将 de *plomentId* 替换为来自的 ID。Step 1您可以使用此部署配置来配置您的子部署，并根据需要进行更改。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

响应包含部署的详细信息。将get-deployment命令响应中的以下任意键值对复制到。`deployment.json`您可以更改子部署的这些值。有关此命令的详细信息的更多信息，请参阅GetDeployment。

- components— 部署的组件。要卸载组件，请将其从该对象中移除。
- deploymentName— 部署的名称。
- deploymentPolicies— 部署的策略。
- iotJobConfiguration— 部署的任务配置。
- parentTargetArn— 父部署的目标。
- tags— 部署的标签。

9. 运行以下命令从deployment.json中创建子部署。将 *subdeploymentName ######*的名称。

```
aws greengrassv2 create-deployment --deployment-name subdeploymentName --cli-input-json file://deployment.json
```

   响应中包含标识deploymentId此子部署的。您可以使用部署 ID 来检查部署的状态。有关更多信息，请参阅检查部署状态。

10. 如果子部署成功，则可以使用其配置来修改父部署。复制您在deployment.json上一步中使用的。将 JSON 文件targetArn中的替换为父部署的 ARN，然后运行以下命令使用此新配置创建父部署。

   Note

   如果您创建父队列的新部署修订版，它将替换该父队列的所有部署修订版和子部署。有关更多信息，请参阅修改部署。

```
aws greengrassv2 create-deployment --cli-input-json file://deployment.json
```

   响应中deploymentId包含用于标识此部署的。您可以使用部署 ID 来检查部署的状态。有关更多信息，请参阅 检查部署状态。

修改部署

每个目标事物或事物组一次只能有一个活动部署。当您为已部署的目标创建部署时，新部署中的软件组件将替换先前部署中的软件组件。如果新部署未定义先前部署所定义的组件，则AWS IoT Greengrass 核心软件将从目标核心设备中删除该组件。您可以修改现有部署，这样就不会将核心设备上运行的组件从先前的部署中移至目标。

要修改部署，您需要创建一个从先前部署中存在的相同组件和配置开始的部署。您使用该CreateDeployment操作，该操作与用于创建部署的操作相同。

修改部署 (AWS CLI)

1. 运行以下命令，列出部署目标的部署。将 *targetArn* 替换为目标事物或事物组的 AWS IoT ARN。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

   响应中包含目标最新部署的列表。复制响应deploymentId中的内容，以便在下一步中使用。

> **Note**
>
> 您还可以修改除目标最新版本之外的部署。指定--history-filter ALL参数以列出目标的所有部署。然后，复制要修改的部署的 ID。

2. 运行以下命令，获取部署的详细信息。这些详细信息包括元数据、组件和作业配置。将 *deplomentId* 替换为上一步中的 ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

   响应包含部署的详细信息。

3. 创建一个名为 deployment.json 的文件，并将上一命令的响应复制到该文件中。
4. 从 deployment.json 中的 JSON 对象中删除以下键/值对：

   - deploymentId
   - revisionId
   - iotJobId
   - iotJobArn
   - creationTimestamp
   - isLatestForTarget
   - deploymentStatus

   该CreateDeployment操作需要具有以下结构的有效负载。

```
{
  "targetArn": "String",
  "components": Map of components,
  "deploymentPolicies": DeploymentPolicies,
  "iotJobConfiguration": DeploymentIoTJobConfiguration,
  "tags": Map of tags
}
```

5. 在 `deployment.json` 中，执行以下任何操作：

   - 更改部署的名称 (`deploymentName`)。
   - 更改部署的组件 (`components`)。
   - 更改部署的策略 (`deploymentPolicies`)。
   - 更改部署的作业配置 (`iotJobConfiguration`)。
   - 更改部署的标签 (`tags`)。

   有关如何定义这些部署细节的更多信息，请参阅创建部署。

6. 运行以下命令从中创建部署`deployment.json`。

   ```
   aws greengrassv2 create-deployment --cli-input-json file://deployment.json
   ```

   响应中`deploymentId`包含用于标识此部署的。您可以使用部署 ID 来检查部署的状态。有关更多信息，请参阅 检查部署状态。

取消作业

您可以取消主动部署，以防止其软件组件安装在AWS IoT Greengrass核心设备上。如果您取消以事物组为目标的部署，则添加到该组的核心设备将无法获得持续部署。如果核心设备已经运行了部署，则取消部署时您不会更改该设备上的组件。您必须创建新的部署或修改部署，以修改在收到取消部署的核心设备上运行的组件。

取消部署 (AWS CLI)

1. 运行以下命令以查找目标的最新部署版本的 ID。最新修订版是唯一可以对目标处于活动状态的部署，因为在创建新修订版时，先前的部署会取消。*targetArn* 事物或事物组的 ARN 替换目标 AWS IoT事物或事物组的 ARN。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

响应包含一个列表，其中包含目标的最新部署。复制响应deploymentId中的内容，以便在下一个步骤中使用。

2. 运行以下命令取消部署。将d *eploymentId* 替换为上一步中的 ID。

```
aws greengrassv2 cancel-deployment --deployment-id deploymentId
```

如果操作成功，则部署状态将更改为CANCELED。

检查部署状态

您可以检查在其中创建的部署的状态AWS IoT Greengrass。您也可以检查部署至每台核心设备的AWS IoT作业的状态。部署处于活动状态时，AWS IoT任务的状态为IN_PROGRESS。创建部署的新修订版后，先前修订版的AWS IoT任务状态更改为CANCELLED。

主题

- 检查部署状态
- 检查设备部署状态

检查部署状态

您可以检查通过其目标或 ID 识别的部署的状态。

按目标检查部署状态 (AWS CLI)

- 运行以下命令以检索目标的更新部署的状态。将 *TargetArn* 替换为部署目标或AWS IoT事物组的 Amazon 资源名称（ARN）。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

响应包含一个列表，其中包含目标的最新部署。此部署对象包括部署状态。

通过 ID 检查部署状态 (AWS CLI)

- 运行以下命令以检索部署的状态。将 de *ploydeploymentId* 替换为要查询的部署的 ID。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

响应包含部署状态。

检查设备部署状态

您可以检查应用于单个核心设备的部署作业的状态。您也可以检查事物组部署的部署作业的状态。

检查核心设备的部署任务状态 (AWS CLI)

- 运行以下命令以检索核心设备的所有部署作业的状态。*coreDeviceName*替换为要查询的核心设备的名称。

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

  响应包含核心设备的部署任务列表。您可以通过任务的`deploymentId`或来确定要部署的任务`targetArn`。每个部署作业都包含核心设备上的作业的状态。

检查事物组的部署状态 (AWS CLI)

1. 运行以下命令以检索现有部署的 ID。将 *TargetArn* 替换为目标事物组的 ARN。

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   响应包含一个列表，其中包含目标的最新部署。复制响应`deploymentId`中的内容，以便在下一步中使用。

   > **Note**
   > 您也可以列出目标的最新部署以外的部署。指定`--history-filter ALL`参数以列出目标的所有部署。然后，复制要检查其状态的部署的 ID。

2. 运行以下命令以获取部署的详细信息。用上*####DeplogetId* 替换上一步中的ID。

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   响应包含有关部署信的信息。复制响应`iotJobId`中的内容，以便在后续步骤中使用。

3. 运行以下命令以描述部署时执行的核心设备的作业的情况。将*iotJobId*和*coreDeviceThing##*替换为上一步中的任务 ID 和要检查状态的核心设备。

```
aws iot describe-job-execution --job-id iotJobId --thing-name coreDeviceThingName
```

响应包含核心设备的部署任务执行状态和有关状态的详细信息。detailsMap包含以下信息：

- detailed-deployment-status— 部署结果状态，可能具有下列值之一：
  - SUCCESSFUL— 部署成功。
  - FAILED_NO_STATE_CHANGE— 当核心设备准备应用部署时，部署失败。
  - FAILED_ROLLBACK_NOT_REQUESTED— 部署失败，部署未指定回滚到以前的工作配置，因此核心设备可能无法正常运行。
  - FAILED_ROLLBACK_COMPLETE— 部署失败，核心设备成功回滚到以前的工作配置。
  - FAILED_UNABLE_TO_ROLLBACK— 部署失败，核心设备无法恢复到以前的工作配置，因此核心设备可能无法正常运行。

  如果部署失败，请检查deployment-failure-cause值和核心设备的日志文件以确定问题。有关如何访问核心设备的日志文件的更多信息，请参阅监控AWS IoT Greengrass日志。
- deployment-failure-cause— 一条错误消息，提供有关任务执行失败原因的更多详细信息。

响应与下面的示例相似。

```
{
  "execution": {
    "jobId": "2cc2698a-5175-48bb-adf2-1dd345606ebd",
    "status": "FAILED",
    "statusDetails": {
      "detailsMap": {
        "deployment-failure-cause": "No local or cloud component version
 satisfies the requirements. Check whether the version constraints conflict
 and that the component exists in your AWS ## with a version that matches the
 version constraints. If the version constraints conflict, revise deployments
 to resolve the conflict. Component com.example.HelloWorld version constraints:
 LOCAL_DEPLOYMENT requires =1.0.0, thinggroup/MyGreengrassCoreGroup requires
 =1.0.1.",
        "detailed-deployment-status": "FAILED_NO_STATE_CHANGE"
      }
```

```
    },
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
    "queuedAt": "2022-02-15T14:45:53.098000-08:00",
    "startedAt": "2022-02-15T14:46:05.670000-08:00",
    "lastUpdatedAt": "2022-02-15T14:46:20.892000-08:00",
    "executionNumber": 1,
    "versionNumber": 3
  }
}
```

AWS IoT Greengrass 中的日志记录和监控

监控是保持 AWS IoT Greengrass 和您的 AWS 解决方案的可靠性、可用性和性能的重要方面。您应从 AWS 解决方案的所有部分收集监控数据，以便更轻松地调试出现的多点故障。在开始监控 AWS IoT Greengrass 之前，您应该创建一个监控计划，其中包括以下问题的答案：

- 监控目的是什么？
- 您将监控哪些资源？
- 监控这些资源的频率如何？
- 您将使用哪些监控工具？
- 谁负责执行监控任务？
- 出现错误时应通知谁？

主题

- 监控工具
- 监控AWS IoT Greengrass日志
- 使用记录 AWS IoT Greengrass V2 API 调用 AWS CloudTrail
- 从AWS IoT Greengrass核心设备收集系统运行状况遥测数据
- 获取部署和组件运行状况通知
- 查看 Greengrass 核心设备状态

监控工具

AWS 为您提供了各种可用于监控 AWS IoT Greengrass 的工具。您可以配置其中的一些工具以便进行监控。一些工具需要手动干预。建议您尽可能实现监控任务自动化。

您可以使用以下自动化监控工具来监控 AWS IoT Greengrass 并报告问题：

- Amazon CloudWatch Logs — 监控、存储和访问来自AWS CloudTrail或其他来源的日志文件。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的监控日志文件。
- AWS CloudTrail日志监控-在账户之间共享日志文件，通过将 CloudTrail 日志文件发送到 “日志” 来实时监控 CloudWatch 日志文件，用 Java 编写日志处理应用程序，并验证您的日志文件在传送后是否未更改 CloudTrail。有关更多信息，请参阅《AWS CloudTrail用户指南》中的使用 CloudTrail 日志文件。

- Greengrass 系统运行状况遥测 – 订阅后可接收从 Greengrass 核心发送的遥测数据。有关更多信息，请参阅 the section called “收集系统运行状况遥测数据”。
- 设备运行状况通知使用 Amazon 创建事件 EventBridge ，以接收有关部署和组件的状态更新。有关更多信息，请参阅 获取部署和组件运行状况通知。
- 舰队状态服务 — 使用舰队状态 API 操作来检查核心设备及其 Greengrass 组件的状态。您还可以在 AWS IoT Greengrass控制台中查看舰队状态信息。有关更多信息，请参阅 查看 Greengrass 核心设备状态。

监控AWS IoT Greengrass日志

AWS IoT Greengrass 由云服务和 AWS IoT Greengrass 核心软件组成。AWS IoT Greengrass核心软件可以将日志写入 Amazon CloudWatch Logs 和核心设备的本地文件系统。在核心设备上运行的 Greengrass 组件也可以将日志写入日志和本地文件 CloudWatch 系统。您可以使用日志来监控事件和排查问题。所有 AWS IoT Greengrass 日志条目包含时间戳、日志级别和事件相关信息。

默认情况下，AWS IoT GreengrassCore 软件仅将日志写入本地文件系统。您可以实时查看文件系统日志，因此可以调试自己开发和部署的 Greengrass 组件。您还可以将核心设备配置为将日志写入日 CloudWatch 志，这样您就可以在不访问本地文件系统的情况下对核心设备进行故障排除。有关更多信息，请参阅 启用记录到 CloudWatch 日志。

主题

- 访问文件系统日志
- 访问 CloudWatch 日志
- 访问系统服务日志
- 启用记录到 CloudWatch 日志
- 为 AWS IoT Greengrass 配置日志记录
- AWS CloudTrail日志

访问文件系统日志

AWS IoT GreengrassCore 软件将日志存储在核心设备上的*/greengrass/v2*/logs文件夹中，其中*/greengrass/v2*是AWS IoT Greengrass根文件夹的路径。logs 文件夹具有以下结构。

```
/greengrass/v2
### logs
    ### greengrass.log
```

```
    ### greengrass_2021_09_14_15_0.log
    ### ComponentName.log
    ### ComponentName_2021_09_14_15_0.log
    ### main.log
```

- `greengrass.log`— AWS IoT Greengrass 核心软件日志文件。使用此日志文件查看有关组件和部署的实时信息。该日志文件包括 Greengrass nucleus（核心软件的核心）和插件组件（例如日志管理器和秘密管理器）的AWS IoT Greengrass日志。
- *ComponentName*`.log`— Greengrass 组件日志文件。使用组件日志文件查看有关在核心设备上运行的 Greengrass 组件的实时信息。通用组件和 Lambda 组件将标准输出 (stdout) 和标准错误 (stderr) 写入这些日志文件。
- `main.log`— 处理组件生命周期的`main`服务的日志文件。此日志文件将始终为空。

有关插件、通用组件和 Lambda 组件之间区别的更多信息，请参阅。组件类型

在使用文件系统日志时，请注意以下几点：

- root 用户权限

  您必须具有根权限才能读取文件系统上的 AWS IoT Greengrass 日志。
- 日志文件轮换

  AWS IoT GreengrassCore 软件每小时或在日志文件超过文件大小限制时轮换日志文件。轮换的日志文件的文件名中包含时间戳。例如，可以命名`greengrass_2021_09_14_15_0.log`轮换的 AWS IoT Greengrass Core 软件日志文件。默认文件大小限制为 1,024 KB (1 MB)。你可以在 Greengrass nucleus 组件上配置文件大小限制。
- 删除日志文件

  当 C AWS IoT Greengrass ore 软件日志文件或 Greengrass 组件日志文件（包括轮换的日志文件）的大小超过磁盘空间限制时，C AWS IoT Greengrass ore 软件会清理较早的日志文件。AWS IoT Greengrass核心软件日志和每个组件日志的默认磁盘空间限制为 10,240 KB (10 MB)。您可以在 Greengrass nucleus 组件或日志管理器组件上配置AWS IoT Greengrass核心软件日志磁盘空间限制。您可以在日志管理器组件上配置每个组件的日志磁盘空间限制。

查看AWS IoT Greengrass核心软件日志文件

- 运行以下命令以实时查看日志文件。*/greengrass/v2*替换为AWS IoT Greengrass根文件夹的路径。

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows Command Prompt (CMD)

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

该type命令将文件内容写入终端。多次运行此命令以观察文件中的更改。

PowerShell

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

查看组件的日志文件

- 运行以下命令以实时查看日志文件。将*/greengrass/v2*或 *C:\greengrass\v2* 替换为AWS IoT Greengrass根文件夹的路径，然后替换 *com.example# HelloWorld*使用组件的名称。

  Linux or Unix

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

  PowerShell

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

你也可以使用 Greengrass CLI 的logs命令来分析核心设备上的 Greengr ass 日志。要使用该logs命令，必须将 Greengrass nucleus 配置为输出 JSON 格式的日志文件。有关更多信息，请参阅 Greengrass 命令行界面和 日志。

访问 CloudWatch 日志

您可以部署日志管理器组件来配置核心设备以写入 CloudWatch 日志。有关更多信息，请参阅 启用记录到 CloudWatch 日志。然后，您可以在 Amazon CloudWatch 控制台的 “日志” 页面上或使用 CloudWatch 日志 API 查看日志。

日志组名称

```
/aws/greengrass/componentType/region/componentName
```

日志组名称使用以下变量：

- componentType— 组件的类型，可以是以下类型之一：
  - GreengrassSystemComponent— 此日志组包括核心和插件组件的日志，它们与 Greengrass 核心在同一 JVM 中运行。该组件是 Greengrass 核的一部分。
  - UserComponent— 此日志组包括设备上通用组件、Lambda 组件和其他应用程序的日志。该组件不是 Greengrass 核的一部分。

  有关更多信息，请参阅 组件类型。
- region— 核心设备使用的AWS区域。
- componentName— 组件的名称。对于系统日志，此值为System。

日志流名称

```
/date/thing/thingName
```

日志流名称使用以下变量：

- date— 日志的日期，例如2020/12/15。日志管理器组件使用该yyyy/MM/dd格式。
- thingName— 核心设备的名称。

> Note
>
> 如果事物名称包含冒号 (:)，则日志管理器会将冒号替换为加号 (+)。

使用日志管理器组件写入日志时，需要考虑以下注意事项： CloudWatch

- 日志延迟

> Note
>
> 我们建议您升级到日志管理器版本 2.3.0，该版本可减少轮换和活动日志文件的日志延迟。当你升级到日志管理器 2.3.0 时，我们建议你同时升级到 Greengrass nucleus 2.9.1。

日志管理器组件版本 2.2.8（及更早版本）仅处理和上传轮换日志文件中的日志。默认情况下，AWS IoT GreengrassCore 软件每小时或在 1,024 KB 之后轮换一次日志文件。因此，只有在 C AWS IoT Greengrass ore 软件或 Greengrass 组件写入价值超过 1,024 KB 的日志之后，日志管理器组件才会上传日志。您可以配置较低的日志文件大小限制，以使日志文件更频繁地轮换。这会导致日志管理器组件更频繁地将 CloudWatch 日志上传到日志。

日志管理器组件版本 2.3.0（及更高版本）处理并上传所有日志。当您写入新日志时，日志管理器版本 2.3.0（及更高版本）会处理并直接上传该活动日志文件，而不是等待其轮换。这意味着您可以在 5 分钟或更短的时间内查看新日志。

日志管理器组件会定期上传新日志。默认情况下，日志管理器组件每 5 分钟上传一次新日志。您可以配置较低的上传间隔，以便日志管理器组件通过配置来更频繁地将 CloudWatch 日志上传到日志。`periodicUploadIntervalSec`有关如何配置此周期间隔的更多信息，请参阅配置。

日志可以近乎实时地从同一 Greengrass 文件系统上传。如果您需要实时观察日志，请考虑使用文件系统日志。

> **Note**
>
> 如果您使用不同的文件系统写入日志，则日志管理器会恢复到日志管理器组件版本 2.2.8 及更早版本中的行为。有关访问文件系统日志的信息，请参阅访问文件系统日志。

- 时钟偏差

  日志管理器组件使用标准的签名版本 4 签名流程来创建对 CloudWatch 日志的 API 请求。如果核心设备上的系统时间不同步超过 15 分钟，则 CloudWatch Logs 会拒绝请求。有关更多信息，请参阅《AWS 一般参考》中的签名版本 4 签名流程。

访问系统服务日志

如果您将 AWS IoT Greengrass Core 软件配置为系统服务，则可以查看系统服务日志以解决问题，例如软件无法启动。

查看系统服务日志 (CLI)

1. 运行以下命令查看AWS IoT Greengrass核心软件系统服务日志。

Linux or Unix (systemd)

```
sudo journalctl -u greengrass.service
```

Windows Command Prompt (CMD)

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

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.wrapper.log
```

2. 在 Windows 设备上，AWS IoT GreengrassCore 软件会为系统服务错误创建单独的日志文件。运行以下命令查看系统服务错误日志。

   Windows Command Prompt (CMD)

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

   PowerShell

   ```
   gc C:\greengrass\v2\logs\greengrass.err.log
   ```

在 Windows 设备上，您还可以使用事件查看器应用程序来查看系统服务日志。

查看 Windows 服务日志（事件查看器）

1. 打开事件查看器应用程序。
2. 选择 Windows 日志将其展开。
3. 选择应用程序以查看应用程序服务日志。
4. 查找并打开来源为的事件日志greengrass。

启用记录到 CloudWatch 日志

您可以部署日志管理器组件来配置核心设备以将日志写入日 CloudWatch 志。您可以为AWS IoT Greengrass核心软件 CloudWatch 日志启用日志，也可以为特定 Greengrass 组件启用 CloudWatch 日志。

> **Note**
>
> Greengrass 核心设备的令牌交换角色必须允许核心设备写入 CloudWatch 日志，如以下示例 IAM 策略所示。如果您安装了具有自动资源配置功能的 AWS IoT Greengrass Core 软件，则您的核心设备具有这些权限。
>
> ```
> {
>   "Version": "2012-10-17",
>   "Statement": [
>     {
>       "Action": [
>         "logs:CreateLogGroup",
>         "logs:CreateLogStream",
>         "logs:PutLogEvents",
>         "logs:DescribeLogStreams"
>       ],
>       "Effect": "Allow",
>       "Resource": "arn:aws:logs:*:*:*"
>     }
>   ]
> }
> ```

要将核心设备配置为将AWS IoT Greengrass核心软件日志写入 CloudWatch 日志，请创建一个部署，为该`aws.greengrass.LogManager`组件指定设置`uploadToCloudWatchtrue`为的配置更新。AWS IoT Greengrass核心软件日志包括 Greengrass 核心和插件组件的日志。

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true"
    }
  }
}
```

要将核心设备配置为将 Greengrass 组件的日志写入日志，CloudWatch 请创建一个指定配置更新的部署，将该组件添加到组件日志配置列表中。当您将组件添加到此列表时，日志管理器组件会将其日志写入 CloudWatch 日志。组件日志包括通用组件和 Lambda 组件的日志。

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {

      }
    }
  }
}
```

部署日志管理器组件时，还可以配置磁盘空间限制，以及核心设备是否在将日志文件写入 CloudWatch 日志后将其删除。有关更多信息，请参阅 为 AWS IoT Greengrass 配置日志记录。

为 AWS IoT Greengrass 配置日志记录

您可以配置以下选项来自定义 Greengrass 核心设备的日志记录。要配置这些选项，请创建一个部署，以指定 Greengrass 核心或日志管理器组件的配置更新。

- 将日志写入 CloudWatch 日志

  要对核心设备进行远程故障排除，您可以将核心设备配置为将AWS IoT Greengrass核心软件和组件日志写入 CloudWatch 日志。为此，请部署和配置日志管理器组件。有关更多信息，请参阅 启用记录到 CloudWatch 日志。
- 删除上传的日志文件

  为了减少磁盘空间使用量，您可以将核心设备配置为在将日志文件写入日志后删除 CloudWatch 日志文件。有关更多信息，请参阅日志管理器组件的`deleteLogFileAfterCloudUpload`参数，您可以为AWS IoT Greengrass核心软件日志和组件日志指定该参数。
- 日志磁盘空间限制

  要限制磁盘空间的使用，可以在核心设备上为每个日志（包括其轮换的日志文件）配置最大磁盘空间。例如，您可以为`greengrass.log`和轮换`greengrass.log`的文件配置最大组合磁盘空间。有关更多信息，请参阅 Greengrass nucleus 组件`logging.totalLogsSizeKB`的参数和日志管理器组件`diskSpaceLimit`的参数，您可以AWS IoT Greengrass为核心软件日志和组件日志指定这些参数。

- 日志文件大小限制

  您可以为每个日志文件配置最大文件大小。日志文件超过此文件大小限制后，AWS IoT GreengrassCore 软件会创建一个新的日志文件。日志管理器组件版本 2.28（及更早版本）仅将轮换的日志文件写入 CloudWatch 日志，因此您可以指定较低的文件大小限制，以便更频繁地将日志写入 CloudWatch 日志。日志管理器组件版本 2.3.0（及更高版本）会处理和上传所有日志，而不是等待轮换。有关更多信息，请参阅 Greengrass nucleus 组件的日志文件大小限制参数 ()。`logging.fileSizeKB`

- 最低日志级别

  您可以配置 Greengrass nucleus 组件写入文件系统日志的最低日志级别。例如，您可以指定`DEBUG`级别日志来帮助进行故障排除，也可以指定`ERROR`级别日志以减少核心设备创建的日志量。有关更多信息，请参阅 Greengrass nucleus 组件的日志级别参数 ()。 `logging.level`

  您还可以配置日志管理器组件写入日志的最低 CloudWatch 日志级别。例如，您可以指定更高的日志级别以降低日志成本。有关更多信息，请参阅日志管理器组件的`minimumLogLevel`参数，您可以为AWS IoT Greengrass核心软件日志和组件日志指定该参数。

- 检查要写入日志的日志的时间间隔 CloudWatch

  要增加或减少日志管理器组件将日志写入日志的频率，您可以配置它检查要写入的新日志文件的间隔。 CloudWatch 例如，与默认的 5 分钟间隔相比，您可以指定更短的时间间隔来查看 CloudWatch 日志中的日志。您可以指定更高的间隔以降低成本，因为日志管理器组件会将日志文件批处理成更少的请求。有关更多信息，请参阅日志管理器组件的上传间隔参数 (`periodicUploadIntervalSec`)。

- 日志格式

  您可以选择 C AWS IoT Greengrass ore 软件是以文本格式还是 JSON 格式写入日志。如果您阅读日志，请选择文本格式；如果您使用应用程序读取或解析日志，则选择 JSON 格式。有关更多信息，请参阅 Greengrass nucleus 组件的日志格式参数 ()。 `logging.format`

- 本地文件系统日志文件夹

  您可以在核心设备上将日志文件夹从更改*/greengrass/v2*/logs为另一个文件夹。有关更多信息，请参阅 Greengrass nucleus 组件的输出目录参数 ()。 `logging.outputDirectory`

AWS CloudTrail日志

AWS IoT Greengrass与AWS CloudTrail一项服务集成，该服务提供用户、角色或AWS 服务中的操作记录AWS IoT Greengrass。有关更多信息，请参阅 使用记录 AWS IoT Greengrass V2 API 调用 AWS CloudTrail。

使用记录 AWS IoT Greengrass V2 API 调用 AWS CloudTrail

AWS IoT Greengrass V2 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在中执行的操作的记录 AWS IoT Greengrass Version 2。 CloudTrail 将所有 API 调用捕获 AWS IoT Greengrass 为事件。捕获的调用包括来自 AWS IoT Greengrass 控制台的调用和对 AWS IoT Greengrass API 操作的代码调用。

如果您创建了跟踪，则可以启用向 S3 存储桶持续传输事件，包括的事件 AWS IoT Greengrass。CloudTrail 如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 "事件历史记录" 中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向哪个请求发出 AWS IoT Greengrass、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

有关的更多信息 CloudTrail，请参阅《AWS CloudTrail 用户指南》。

AWS IoT Greengrass V2 信息在 CloudTrail

CloudTrail 在您创建账户 AWS 账户 时已在您的账户上启用。当活动发生在中时 AWS IoT Greengrass，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在 AWS 账户中查看、搜索和下载最新事件。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录您的 AWS 账户事件（包括的事件） AWS IoT Greengrass，请创建跟踪。跟踪允许 CloudTrail 将日志文件传送到 S3 存储桶。默认情况下，当您在控制台中创建跟踪时，该跟踪将应用于所有 AWS 区域跟踪。跟踪记录 AWS 分区中所有区域的事件，并将日志文件传送到您指定的 S3 存储桶。此外，您可以配置其他 AWS 服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- 创建跟踪记录概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个地区的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

所有 AWS IoT Greengrass V2 操作均由《API 参考》记录 CloudTrail 并记录在《AWS IoT Greengrass V2 API 参考》中。例如，调用`CreateDeployment`和`CancelDeployment`操作会在 CloudTrail 日志文件中生成条目。`CreateComponentVersion`

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息有助于您确定以下内容：

- 请求是使用根证书还是 AWS Identity and Access Management (IAM) 用户凭证发出。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

AWS IoT Greengrass 中的数据事件 CloudTrail

数据事件提供有关在资源上或在资源中执行的资源操作的信息（例如，获取组件版本或部署配置）。这些也称为数据层面操作。数据事件通常是高容量活动。默认情况下， CloudTrail 不记录数据事件。CloudTrail 事件历史记录不记录数据事件。

记录数据事件将收取额外费用。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail 定价。

您可以使用 CloudTrail 控制台或 CloudTrail API 操作记录 AWS IoT Greengrass 资源类型的数据事件。 AWS CLI本节中的表格显示了可用的资源类型 AWS IoT Greengrass。

- 要使用 CloudTrail 控制台记录数据事件，请创建跟踪或事件数据存储以记录数据事件，或者更新现有的跟踪或事件数据存储以记录数据事件。
  1. 选择数据事件以记录数据事件。
  2. 从数据事件类型列表中，选择要为其记录数据事件的资源类型。
  3. 选择要使用的日志选择器模板。您可以记录资源类型的所有数据事件、记录所有`readOnly`事件、记录所有`writeOnly`事件，或者创建自定义日志选择器模板来筛选`readOnlyeventName`、和`resources.ARN`字段。
- 要使用记录数据事件 AWS CLI，请将`--advanced-event-selectors`参数配置为将`eventCategory`字段设置为等于`Data`并将该`resources.type`字段设置为资源类型值（参见表）。您可以添加条件来筛选`readOnlyeventName`、和`resources.ARN`字段的值。
  - 要配置记录数据事件的跟踪，请运行put-event-selectors命令。有关更多信息，请参阅使用记录跟踪的数据事件 AWS CLI。

- 要将事件数据存储配置为记录数据事件，请运行create-event-data-store命令创建新的事件数据存储以记录数据事件，或者运行update-event-data-store命令来更新现有的事件数据存储。有关更多信息，请参阅使用记录事件数据存储的数据事件 AWS CLI。

下表列出了 AWS IoT Greengrass 资源类型。数据事件类型（控制台）列显示要从控制 CloudTrail 台上的数据事件类型列表中选择的值。res ources.type 值列显示该`resources.type`值，您将在使用或 API 配置高级事件选择器时指定该值。 AWS CLI CloudTrail “记录到的数据 API CloudTrail” 列显示了 CloudTrail针对该资源类型记录的 API 调用。

数据事件类型（控制台）	resources.type 值	数据 API 已登录到 CloudTrail
物联网证书	`AWS::IoT::Certificate`	• VerifyClientDeviceIdentity • VerifyClientDeviceIoTCertificateAssociation
物联网 Greengrass 组件版本	`AWS::GreengrassV2::ComponentVersion`	• ResolveComponentCandidates
物联网 Greengrass 部署	`AWS::GreengrassV2::Deployment`	• GetDeploymentConfiguration
物联网的东西	`AWS::IoT::Thing`	• ListThingGroupsForCoreDevices • PutCertificateAuthorities • VerifyClientDeviceIoTCertificateAssociation

> **Note**
>
> Greengrass 不会记录被拒绝访问的事件。

您可以将高级事件选择器配置为在 `eventName`、`readOnly` 和 `resources.ARN` 字段上进行筛选，从而仅记录那些对您很重要的事件。

添加筛选条件`eventName`以包含或排除特定的数据 API。

有关这些字段的更多信息，请参阅AdvancedFieldSelector。

以下示例说明如何使用配置高级选择器。 AWS CLI用您自己的信息替换*TrailName*和*##*。

Example — 记录物联网事物的数据事件

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log all thing data events",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] }
            ]
        }
]'
```

Example — 筛选特定的物联网事物 API

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log IoT Greengrass PutCertificateAuthorities API calls",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] },
                { "Field": "eventName", "Equals": ["PutCertificateAuthorities"] }
            ]
    }
]'
```

Example — 记录所有 Greengrass 数据事件

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
        "Name": "Log all certificate data events",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
```

```
                "Equals": [
                    "Data"
                ]
            },
            {
                "Field": "resources.type",
                "Equals": [
                    "AWS::IoT::Certificate"
                ]
            }
        ]
    },
    {
        "Name": "Log all component version data events",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
                "Equals": [
                    "Data"
                ]
            },
            {
                "Field": "resources.type",
                "Equals": [
                    "AWS::GreengrassV2::ComponentVersion"
                ]
            }
        ]
    },
    {
        "Name": "Log all deployment version",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
                "Equals": [
                    "Data"
                ]
            },
            {
                "Field": "resources.type",
                "Equals": [
                    "AWS::GreengrassV2::Deployment"
                ]
            }
```

```
        ]
    },
    {
        “Name”: “Log all thing data events”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::IoT::Thing”
                ]
            }
        ]
    }
]’
```

AWS IoT Greengrass 中的管理事件 CloudTrail

管理事件提供有关对您 AWS 账户中的资源执行的管理操作的信息。这些也称为控制层面操作。默认情况下， CloudTrail 记录管理事件。

AWS IoT Greengrass 将所有 AWS IoT Greengrass 控制平面操作记录为管理事件。有关 AWS IoT Greengrass 登录到的 AWS IoT Greengrass 控制平面操作的列表 CloudTrail，请参阅 AWS IoT Greengrass API 参考版本 2。

了解 AWS IoT Greengrass V2 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件表示来自任何源的单个请求。它包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

以下示例显示了演示该`CreateDeployment`操作的 CloudTrail 日志条目。

```
{
    "eventVersion": "1.08",
    "userIdentity": {
```

```
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Administrator",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Administrator"
    },
    "eventTime": "2021-01-06T02:38:05Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "CreateDeployment",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "aws-cli/2.1.9 Python/3.7.9 Windows/10 exe/AMD64 prompt/off command/
greengrassv2.create-deployment",
    "requestParameters": {
        "deploymentPolicies": {
            "failureHandlingPolicy": "DO_NOTHING",
            "componentUpdatePolicy": {
                "timeoutInSeconds": 60,
                "action": "NOTIFY_COMPONENTS"
            },
            "configurationValidationPolicy": {
                "timeoutInSeconds": 60
            }
        },
        "deploymentName": "Deployment for MyGreengrassCoreGroup",
        "components": {
            "aws.greengrass.Cli": {
                "componentVersion": "2.0.3"
            }
        },
        "iotJobConfiguration": {},
        "targetArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup"
    },
    "responseElements": {
        "iotJobArn": "arn:aws:iot:us-west-2:123456789012:job/fdfeba1d-ac6d-44ef-
ab28-54f684ea578d",
        "iotJobId": "fdfeba1d-ac6d-44ef-ab28-54f684ea578d",
        "deploymentId": "4196dddc-0a21-4c54-a985-66a525f6946e"
    },
    "requestID": "311b9529-4aad-42ac-8408-c06c6fec79a9",
    "eventID": "c0f3aa2c-af22-48c1-8161-bad4a2ab1841",
    "readOnly": false,
```

```
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123456789012"
}
```

从AWS IoT Greengrass核心设备收集系统运行状况遥测数据

系统运行状况遥测数据是一种诊断数据，可以帮助您监控 Greengrass 核心设备上关键操作的性能。您可以创建项目和应用程序，以检索、分析、转换和报告来自边缘设备的遥测数据。工艺工程师等领域专家可以使用这些应用程序来深入了解实例集的运行状况。

您可以使用以下方法从 Greengrass 核心设备收集遥测数据：

- Nucleus 遥测发射器组件 — Greengras s 核心设备aws.greengrass.telemetry.NucleusEmitter上的 nucleus 遥测发射器组件 () 默认向主题发布遥测数据。$local/greengrass/telemetry即使您的设备与云的连接有限，您也可以使用发布到本主题的数据在核心设备上进行本地操作。或者，您也可以将组件配置为将遥测数据发布到您选择的 M AWS IoT Core QTT 主题。

  必须将 nucleus 发射器组件部署到核心设备才能发布遥测数据。向本地主题发布遥测数据不会产生任何费用。但是，使用 MQTT 主题向发布数据AWS Cloud需视AWS IoT Core定价而定。

  AWS IoT Greengrass提供了多个社区组件，可帮助您使用 InfluxDB 和 Grafana 在核心设备上本地分析和可视化遥测数据。这些组件使用来自原子核发射器组件的遥测数据。有关更多信息，请参阅 InfluxDB 发布者组件的自述文件。

- 遥测代理 — Greengrass 核心设备上的遥测代理收集本地遥测数据并将其发布到亚马逊，无需任何客户互动。 EventBridge 核心设备会尽最大努力向其 EventBridge 发布遥测数据。例如，核心设备在离线时可能无法传输遥测数据。

  所有 Greengrass 核心设备默认启用遥测代理功能。设置 Greengrass 核心设备后，您就会自动开始接收数据。除了您的数据链路费用外，从核心设备传输到的数据AWS IoT Core也是免费的。这是因为代理会发布到 AWS 保留主题。但是，根据您的使用场景，您在接收或处理数据时可能会产生费用。

Note

Amazon EventBridge 是一项事件总线服务，可用于将应用程序与来自各种来源（例如 Greengrass 核心设备）的数据连接起来。有关更多信息，请参阅什么是亚马逊 EventBridge？ 在《亚马逊 EventBridge 用户指南》中。

为确保 AWS IoT Greengrass Core 软件正常运行，请AWS IoT Greengrass使用数据进行开发和质量改进。此功能还有助于为新的和增强的边缘功能提供信息。 AWS IoT Greengrass将遥测数据最多保留七天。

本节介绍如何配置和使用遥测代理。有关配置 nucleus 遥测发射器组件的信息，请参见。Nucleus 遥测发射器

主题

- 遥测指标
- 配置遥测代理设置
- 订阅遥测数据 EventBridge

遥测指标

下表描述了遥测代理发布的指标。

名称	描述	
系统		
`SystemMemUsage`	Greengrass 核心设备上所有应用程序（包括操作系统）当前使用的内存量。	
`CpuUsage`	Greengrass 核心设备上所有应用程序（包括操作系统）当前使用的 CPU 量。	
`TotalNumberOfFDs`	Greengrass 核心设备操作系统存储的文件描述符的数量。一	

名称	描述
	个文件描述符可以唯一地标识一个打开的文件。
Greengrass 核	
NumberOfComponents Running	在 Greengrass 核心设备上运行的组件数量。
NumberOfComponents Errored	Greengrass 核心设备上处于错误状态的组件数量。
NumberOfComponents Installed	安装在 Greengrass 核心设备上的组件数量。
NumberOfComponents Starting	在 Greengrass 核心设备上启动的组件数量。
NumberOfComponents New	Greengrass 核心设备上新增的组件数量。
NumberOfComponents Stopping	在 Greengrass 核心设备上停止运行的组件数量。
NumberOfComponents Finished	在 Greengrass 核心设备上完成的组件数量。
NumberOfComponents Broken	Greengrass 核心设备上损坏的组件数量。
NumberOfComponents Stateless	Greengrass 核心设备上处于无状态的组件数量。
客户端设备身份验证 — 此功能需要客户端设备身份验证组件的 v2.4.0 或更高版本。	
VerifyClientDevice Identity.Success	验证客户端设备身份是否成功的次数。

名称	描述	
`VerifyClientDeviceIdentity.Failure`	验证客户端设备身份失败的次数。	
`AuthorizeClientDeviceActions.Success`	客户端设备被授权完成请求的操作的次数。	
`AuthorizeClientDeviceActions.Failure`	客户端设备无权完成请求的操作的次数。	
`GetClientDeviceAuthToken.Success`	客户端设备成功通过身份验证的次数。	
`GetClientDeviceAuthToken.Failure`	无法对客户端设备进行身份验证的次数。	
`SubscribeToCertificateUpdates.Success`	成功订阅证书更新的次数。	
`SubscribeToCertificateUpdates.Failure`	尝试订阅证书更新的失败次数。	
`ServiceError`	客户端设备身份验证中未处理的内部错误数量。	
直播管理器 — 此功能需要 Greengrass 核心组件的 v2.7.0 或更高版本。		
`BytesAppended`	附加到流管理器的数据的字节数。	
`BytesUploadedToIoTAnalytics`	流管理器导出到 AWS IoT Analytics 中频道的数据的字节数。	

名称	描述
BytesUploadedToKinesis	流管理器导出到 Amazon Kinesis Data Streams 中流的数据的字节数。
BytesUploadedToIoTSiteWise	流管理器导出到 AWS IoT SiteWise 中资源属性的数据的字节数。
BytesUploadedToS3	流管理器导出到 Amazon S3 中对象的数据的字节数。

配置遥测代理设置

遥测代理使用以下默认设置：

- 遥测代理每小时汇总一次遥测数据。
- 遥测代理每 24 小时发布一次遥测消息。

遥测代理使用 MQTT 协议发布服务质量 (QoS) 级别为 0 的数据，这意味着它不会确认交付或重试发布尝试。遥测消息与其他发往 AWS IoT Core 的订阅消息共享 MQTT 连接。

除了数据链路成本外，从核心到核心的数据传输AWS IoT Core也是免费的。这是因为代理会发布到 AWS 保留主题。但是，根据您的使用场景，您在接收或处理数据时可能会产生费用。

您可以为每台 Greengrass 核心设备启用或禁用遥测代理功能。您还可以配置核心设备聚合和发布数据的间隔。要配置遥测，请在部署 Greengr ass nucleus 组件时自定义遥测配置参数。

订阅遥测数据 EventBridge

您可以在 Amazon 中创建规则 EventBridge ，定义如何处理遥测代理在 Greengrass 核心设备上发布的遥测数据。 EventBridge 收到数据后，它会调用您的规则中定义的目标操作。例如，您可以创建事件规则来发送通知、存储事件信息、采取纠正措施或调用其他事件。

遥测事件

遥测事件使用以下格式。

```
{
  "version": "0",
  "id": "a09d303e-2f6e-3d3c-a693-8e33f4fe3955",
  "detail-type": "Greengrass Telemetry Data",
  "source": "aws.greengrass",
  "account": "123456789012",
  "time": "2020-11-30T20:45:53Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "ThingName": "MyGreengrassCore",
    "Schema": "2020-07-30",
    "ADP": [
      {
        "TS": 1602186483234,
        "NS": "SystemMetrics",
        "M": [
          {
            "N": "TotalNumberOfFDs",
            "Sum": 6447.0,
            "U": "Count"
          },
          {
            "N": "CpuUsage",
            "Sum": 15.458333333333332,
            "U": "Percent"
          },
          {
            "N": "SystemMemUsage",
            "Sum": 10201.0,
            "U": "Megabytes"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "GreengrassComponents",
        "M": [
          {
            "N": "NumberOfComponentsStopping",
            "Sum": 0.0,
            "U": "Count"
          },
```

```
        {
          "N": "NumberOfComponentsStarting",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsBroken",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsFinished",
          "Sum": 1.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsInstalled",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsRunning",
          "Sum": 7.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsNew",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsErrored",
          "Sum": 0.0,
          "U": "Count"
        },
        {
          "N": "NumberOfComponentsStateless",
          "Sum": 0.0,
          "U": "Count"
        }
      ]
    },
    {
      "TS": 1602186483234,
```

```
        "NS": "aws.greengrass.ClientDeviceAuth",
        "M": [
          {
            "N": "VerifyClientDeviceIdentity.Success",
            "Sum": 3.0,
            "U": "Count"
          },
          {
            "N": "VerifyClientDeviceIdentity.Failure",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "AuthorizeClientDeviceActions.Success",
            "Sum": 20.0,
            "U": "Count"
          },
          {
            "N": "AuthorizeClientDeviceActions.Failure",
            "Sum": 5.0,
            "U": "Count"
          },
          {
            "N": "GetClientDeviceAuthToken.Success",
            "Sum": 5.0,
            "U": "Count"
          },
          {
            "N": "GetClientDeviceAuthToken.Failure",
            "Sum": 2.0,
            "U": "Count"
          },
          {
            "N": "SubscribeToCertificateUpdates.Success",
            "Sum": 10.0,
            "U": "Count"
          },
          {
            "N": "SubscribeToCertificateUpdates.Failure",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "ServiceError",
```

```
            "Sum": 3.0,
            "U": "Count"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "aws.greengrass.StreamManager",
        "M": [
          {
            "N": "BytesAppended",
            "Sum": 157745524.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTAnalytics",
            "Sum": 149012.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToKinesis",
            "Sum": 12192.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTSiteWise",
            "Sum": 13321.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToS3",
            "Sum": 12213.0,
            "U": "Bytes"
          }
        ]
      }
    ]
  }
}
```


ADP 数组包含具有以下属性的聚合数据点的列表：

`TS`

收集数据的时间戳。

`NS`

指标命名空间。

`M`

指标列表。一个指标包含以下属性：

`N`

指标的名称。

`Sum`

此遥测事件中指标值的总和。

`U`

指标值的单位。

有关每个指标的更多信息，请参阅遥测指标。

创建 EventBridge规则的先决条件

在为创建 EventBridge 规则之前AWS IoT Greengrass，应执行以下操作：

- 熟悉中的事件、规则和目标。 EventBridge
- 创建和配置您的 EventBridge 规则调用的目标。规则可以调用多种类型的目标，例如 Amazon Kinesis 流、AWS Lambda 函数、Amazon SNS 主题和 Amazon SQS 队列。

  您的 EventBridge 规则和关联目标必须位于您创建 Greengrass 资源AWS 区域的地方。有关更多信息，请参阅 AWS 一般参考 中的服务端点和配额。

有关更多信息，请参阅什么是亚马逊 EventBridge？ 以及《亚马逊 EventBridge 用户指南》 EventBridge中的 “亚马逊入门”。

创建事件规则以获取遥测数据（控制台）

使用以下步骤AWS Management Console来创建用于接收 Greengrass 核心设备发布的遥测数据的 EventBridge 规则。这样，Web 服务器、电子邮件地址和其他主题订阅者就可以响应事件。有关更多

信息，请参阅 Amazon EventBridge 用户指南中的创建针对来自AWS资源的事件触发的 EventBridge 规则。

1. 打开 A mazon EventBridge 控制台，然后选择创建规则。
2. 在 Name and description (名称和描述) 下，输入规则的名称和描述。
3. 在 Define pattern (定义模式) 下，配置规则模式。

   a. 选择 Event pattern (事件模式)。
   b. 选择 Pre-defined pattern by service (服务预定义的模式)。
   c. 对于 Service provider (服务提供商)，选择 AWS。
   d. 对于 Service name (服务名称)，选择 Greengrass。
   e. 对于事件类型，请选择 Greengrass 遥测数据。
4. 在 Select event bus (选择事件总线) 下，保留默认事件总线选项。
5. 在 Select targets (选择目标) 下，配置您的目标。以下示例使用 Amazon SQS 队列，但您可以配置其他目标类型。

   a. 对于 “目标”，选择 SQS 队列。
   b. 对于 Queue*，选择您的目标队列。
6. 在 Tags - optional (标签 - 可选) 下，定义规则的标签或将字段留空。
7. 选择创建。

创建事件规则以获取遥测数据 (CLI)

使用以下步骤AWS CLI来创建用于接收 Greengrass 核心设备发布的遥测数据的 EventBridge 规则。这样，Web 服务器、电子邮件地址和其他主题订阅者就可以响应事件。

1. 创建 规则。

   - 将*####*替换为核心设备的事物名称。

   Linux or Unix

   ```
   aws events put-rule \
     --name MyGreengrassTelemetryEventRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
   \": [\"thing-name\"]}}"
   ```

Windows Command Prompt (CMD)

```
aws events put-rule ^
  --name MyGreengrassTelemetryEventRule ^
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
\": [\"thing-name\"]}}"
```

PowerShell

```
aws events put-rule `
  --name MyGreengrassTelemetryEventRule `
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
\": [\"thing-name\"]}}"
```

模式中省略的属性将被忽略。

2. 将主题添加为规则目标。以下示例使用 Amazon SQS，但您也可配置其他目标类型。

   - 将 *queue-arn* 替换为 Amazon SQS 队列的 ARN。

Linux or Unix

```
aws events put-targets \
  --rule MyGreengrassTelemetryEventRule \
  --targets "Id"="1","Arn"="queue-arn"
```

Windows Command Prompt (CMD)

```
aws events put-targets ^
  --rule MyGreengrassTelemetryEventRule ^
  --targets "Id"="1","Arn"="queue-arn"
```

PowerShell

```
aws events put-targets `
  --rule MyGreengrassTelemetryEventRule `
  --targets "Id"="1","Arn"="queue-arn"
```

Note

要允许 Amazon EventBridge 调用您的目标队列，您必须在主题中添加基于资源的策略。有关更多信息，请参阅亚马逊 EventBridge 用户指南中的亚马逊 SQS 权限。

有关更多信息，请参阅 Amazon EventBridge 用户指南 EventBridge中的事件和事件模式。

获取部署和组件运行状况通知

亚马逊 EventBridge 事件规则向您提供有关您的设备收到的 Greengrass 部署以及设备上已安装组件的状态变更的通知。 EventBridge 提供描述 AWS 资源变化的近乎实时的系统事件流。 AWS IoT Greengrass 尽最大努力将 EventBridge 这些事件发送到。这意味着， AWS IoT Greengrass 尝试将所有事件发送到， EventBridge 但在极少数情况下，事件可能无法传送。此外， AWS IoT Greengrass 可能会发送给定事件的多个副本，这意味着您的事件侦听器可能不会按照事件发生的顺序接收事件。

Note

Amazon EventBridge 是一项事件总线服务，您可以使用它来连接应用程序与来自各种来源的数据，例如 Greengrass 核心设备以及部署和组件通知。有关更多信息，请参阅什么是亚马逊 EventBridge？ 在《亚马逊 EventBridge 用户指南》中。

主题

- 部署状态更改事件
- 组件状态更改事件
- 创建 EventBridge 规则的先决条件
- 配置设备运行状况通知（控制台）
- 配置设备运行状况通知 (CLI)
- 配置设备运行状况通知 (AWS CloudFormation)
- 另请参阅

部署状态更改事件

AWS IoT Greengrass 当部署进入以下状态时会发出一个事件：FAILED、SUCCEEDED和。COMPLETED您可以创建一条 EventBridge 规则，该规则适用于所有状态转换或向指定状态的过渡。当部署进入启动规则的状态时， EventBridge 调用规则中定义的目标操作。这样，您就可以发送通知、捕获事件信息、采取纠正措施或启动其他事件以响应状态更改。例如，您可以为以下使用案例创建规则：

- 启动部署后操作，例如下载资产和通知人员。
- 在部署成功或失败时发送通知。
- 发布关于部署事件的自定义指标。

部署状态更改的事件采用以下格式：


```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Effective Deployment Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail":{
        "deploymentId": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
        "coreDeviceExecutionStatus": "FAILED|SUCCEEDED|COMPLETED",
        "statusDetails": {
          "errorStack": ["DEPLOYMENT_FAILURE", "ARTIFACT_DOWNLOAD_ERROR", "S3_ERROR",
 "S3_ACCESS_DENIED", "S3_HEAD_OBJECT_ACCESS_DENIED"],
          "errorTypes": ["DEPENDENCY_ERROR", "PERMISSION_ERROR"],
        },
        "reason": "S3_HEAD_OBJECT_ACCESS_DENIED: FAILED_NO_STATE_CHANGE: Failed to
 download artifact name: 's3://pentest27/nucleus/281/aws.greengrass.nucleus.zip' for
 component aws.greengrass.Nucleus-2.8.1, reason: S3 HeadObject returns 403 Access
 Denied. Ensure the IAM role associated with the core device has a policy granting
 s3:GetObject. null (Service: S3, Status Code: 403, Request ID: HR94ZNT2161DAR58,
 Extended Request ID: wTX4DDI+qigQt3uzwl9rlnQiYlBgwvPm/KJFWeFAn9t1mnGXTms/
luLCYANgq08RIH+x2H+hEKc=)"
      }
```

```
}
```

您可以创建规则和事件，以更新您的部署状态。当部署以、、OR 的形式完成时 FAILEDSUCCEEDED，就会启动事件COMPLETED。如果在核心设备上部署失败，您将收到详细的响应，说明部署失败的原因。有关部署错误代码的更多信息，请参阅详细的部署错误代码。

部署状态

- FAILED. 部署失败。
- SUCCEEDED。 以事物组为目标的部署成功完成。
- COMPLETED。 针对某件事的部署成功完成。

可能是事件重复或者顺序颠倒。要确定事件的顺序，请使用 time 属性。

有关errorStacks和中错误代码的完整列表errorTypes，请参阅详细的部署错误代码和详细的组件状态码。

组件状态更改事件

对于 2.12.2 及更早 AWS IoT Greengrass 版本，当组件进入以下状态时，Greengrass 会发出一个事件：和。ERRORED BROKEN对于 Greengrass nucleus 版本 2.12.3 及更高版本，当组件进入以下状态时，Greengrass 会发出一个事件：、、和。ERRORED BROKEN RUNNING FINISHED部署完成后，Greengrass 还会发出一个事件。您可以创建一条 EventBridge 规则，该规则适用于所有状态转换或向指定状态的过渡。当已安装的组件进入启动规则的状态时，将 EventBridge 调用规则中定义的目标操作。这样，您就可以发送通知、捕获事件信息、采取纠正措施或启动其他事件以响应状态更改。

组件状态更改的事件使用以下格式：

Greengrass nucleus v2.12.2 and earlier

<title>组件状态：ERRORED或 BROKEN</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
```

```
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
          }
        ]
    }
}
```

Greengrass nucleus v2.12.3 and later

<title>组件状态：ERRORED或 BROKEN</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
          }
        ]
    }
```

```
}
```

<title>组件状态：RUNNING或 FINISHED</title>


```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "RUNNING|FINISHED",
            "lifecycleStateDetails": null
          }
        ]
    }
}
```


您可以创建规则和事件，以更新已安装组件的状态。当组件在设备上改变状态时，就会启动一个事件。您将收到一份详细的回复，其中解释了组件出错或损坏的原因。您还将收到一个指示失败原因的状态码。有关组件状态代码的更多信息，请参阅详细的组件状态码。

创建 EventBridge 规则的先决条件

在为创建 EventBridge 规则之前 AWS IoT Greengrass，请执行以下操作：

- 熟悉中的事件、规则和目标。 EventBridge
- 创建和配置您的 EventBridge 规则调用的目标。规则可以调用许多类型的目标，包括：
  - Amazon Simple Notification Service (Amazon SNS)
  - AWS Lambda 函数
  - Amazon Kinesis Video Streams

- Amazon Simple Queue Service（Amazon SQS）队列

有关更多信息，请参阅什么是亚马逊 EventBridge？ 以及《亚马逊 EventBridge 用户指南》EventBridge中的 “亚马逊入门”。

配置设备运行状况通知（控制台）

使用以下步骤创建 EventBridge 规则，以便在群组的部署状态发生变化时发布 Amazon SNS 主题。这样，Web 服务器、电子邮件地址和其他主题订阅者就可以响应事件。有关更多信息，请参阅 Amazon EventBridge 用户指南中的创建针对来自 AWS 资源的事件触发的 EventBridge 规则。

1. 打开亚马逊 EventBridge控制台。
2. 在导航窗格中，选择规则。
3. 选择创建规则。
4. 为规则输入名称和描述。

   规则不能与同一区域中的另一个规则和同一事件总线上的名称相同。
5. 对于事件总线，请选择要与此规则关联的事件总线。如果您希望此规则对来自您自己的账户的匹配事件触发，请选择 AWS 默认事件总线。当你账户中的某项 AWS 服务发出事件时，它总是会进入你账户的默认事件总线。
6. 对于规则类型，选择具有事件模式的规则。
7. 选择下一步。
8. 对于事件源，选择 AWS 事件。
9. 在事件模式中，选择 AWS 服务。
10. 对于AWS 服务，选择 Greengrass。
11. 对于事件类型，请从以下选项中进行选择：

    - 对于部署事件，请选择 Greengrass V2 有效部署状态更改。
    - 对于组件事件，请选择 Greengrass V2 已安装组件状态更改。
12. 选择 Next（下一步）。
13. 对于目标类型，选择AWS 服务。
14. 在 选择目标 下，配置您的目标。此示例使用了 Amazon SNS 主题，而您可以配置其他目标类型来发送通知。

    a. 对于 Target (目标)，选择 SNS topic (SNS 主题)。

b. 对于 Topic (主题)，请选择您的目标主题。

c. 选择下一步。

15. 选择下一步。
16. 查看规则详细信息并选择创建规则。

配置设备运行状况通知 (CLI)

使用以下步骤创建在发生 Greengrass 状态更改事件时发布 Amazon SNS 主题的 EventBridge 规则。这样，Web 服务器、电子邮件地址和其他主题订阅者就可以响应事件。

1. 创建 规则。

   • 用于部署状态更改事件。

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
    [\"Greengrass V2 Effective Deployment Status Change\"]}"
   ```

   • 用于组件状态更改事件。

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
    [\"Greengrass V2 Installed Component Status Change\"]}"
   ```

   模式中省略的属性将被忽略。

2. 将主题添加为规则目标。

   • 将 *topic-arn* 替换为 Amazon SNS 主题的 ARN。

   ```
   aws events put-targets \
     --rule TestRule \
     --targets "Id"="1","Arn"="topic-arn"
   ```

Note

要允许 Amazon EventBridge 调用您的目标主题，您必须在主题中添加基于资源的策略。有关更多信息，请参阅《亚马逊 EventBridge 用户指南》中的 Ama zon SNS 权限。

有关更多信息，请参阅 Amazon EventBridge 用户指南 EventBridge中的事件和事件模式。

配置设备运行状况通知 (AWS CloudFormation)

使用 AWS CloudFormation 模板创建 EventBridge 规则，发送有关 Greengrass 群组部署状态变更的通知。有关更多信息，请参阅AWS CloudFormation 用户指南中的 Amazon EventBridge 资源类型参考。

另请参阅

- 检查设备部署状态
- 什么是亚马逊 EventBridge？ 在《亚马逊 EventBridge 用户指南》中

查看 Greengrass 核心设备状态

Greengrass 核心设备向其报告其软件组件的状态。 AWS IoT Greengrass您可以查看每台设备的运行状况摘要，也可以查看每台设备上每个组件的状态。

核心设备的健康状态如下：

- HEALTHY— AWS IoT Greengrass 核心软件和所有组件在核心设备上正常运行。
- UNHEALTHY— AWS IoT Greengrass 核心设备上的核心软件或组件处于错误状态。

Note

AWS IoT Greengrass 依靠各个设备向发送状态更新 AWS Cloud。如果 AWS IoT Greengrass Core 软件未在设备上运行，或者如果设备未连接到 AWS Cloud，则该设备报告的状态可能无法反映其当前状态。状态时间戳表示上次更新设备状态的时间。

核心设备在以下时间发送状态更新：

- AWS IoT Greengrass 核心软件何时启动

- 当核心设备收到来自的部署时 AWS Cloud
- 对于 Greengrass nucleus 2.12.2 及更早版本，当核心设备上任何组件的状态变为或时，核心设备会发送状态更新 ERRORED BROKEN
- 对于 Greengrass nucleus 2.12.3 及更高版本，当核心设备上任何组件的状态变为、、或时，核心设备会发送状态更新 ERRORED BROKEN RUNNING FINISHED
- 按您可以配置的固定间隔，默认为 24 小时

对于 C AWS IoT Greengrass ore v2.7.0 及更高版本，当进行本地部署和云部署时，核心设备会发送状态更新

主题

- 检查核心设备的运行状况
- 检查核心设备组的运行状况
- 检查核心设备组件状态

检查核心设备的运行状况

您可以检查各个核心设备的状态。

检查核心设备的状态 (AWS CLI)

- 运行以下命令以检索设备的状态。*coreDeviceName*替换为要查询的核心设备的名称。

```
aws greengrassv2 get-core-device --core-device-thing-name coreDeviceName
```

  响应包含有关核心设备的信息，包括其状态。

检查核心设备组的运行状况

您可以检查一组核心设备（事物组）的状态。

检查一组设备的状态 (AWS CLI)

- 运行以下命令以检索多个核心设备的状态。将命令中的 ARN 替换为要查询的事物组的 ARN。

```
aws greengrassv2 list-core-devices --thing-group-arn "arn:aws:iot:region:account-id:thinggroup/thingGroupName"
```

响应包含事物组中的核心设备列表。列表中的每个条目都包含核心设备的状态。

检查核心设备组件状态

您可以检查核心设备上软件组件的状态，例如生命周期状态。有关组件生命周期状态的更多信息，请参阅开发AWS IoT Greengrass组件。

检查核心设备上组件的状态 (AWS CLI)

- 运行以下命令以检索核心设备上组件的状态。*coreDeviceName*替换为要查询的核心设备的名称。

  ```
  aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
  ```

  响应包含在核心设备上运行的组件列表。列表中的每个条目都包含组件的生命周期状态，包括数据状态的最新程度以及 Greengrass 核心设备上次向云端发送包含特定组件的消息的时间。响应还将包括将该组件引入Greengrass核心设备的最新部署来源。

  Note

  此命令检索 Greengrass 核心设备运行的组件的分页列表。默认情况下，此列表不包括作为其他组件的依赖项部署的组件。您可以通过将topologyFilter参数设置为，在响应中包含依赖关系ALL。

运行AWS Lambda函数

Note

AWS IoT Greengrass目前不支持在 Windows 核心设备上使用此功能。

您可以将AWS Lambda函数作为在AWS IoT Greengrass核心设备上运行的组件导入。在以下情况下，您可能需要这样做：

- 您的 Lambda 函数中有要部署到核心设备的应用程序代码。
- 您有想要在AWS IoT Greengrass V2核心设备上运行的 AWS IoT Greengrass V1 应用程序。有关更多信息，请参阅 步骤 2：创建和部署 AWS IoT Greengrass V2 组件以迁移 AWS IoT Greengrass V1 应用程序。

Lambda 函数包括对以下组件的依赖关系。在导入函数时，您无需将这些组件定义为依赖项。当您部署 Lambda 函数组件时，部署包括这些 Lambda 组件依赖项。

- Lambda 启动器组件 (`aws.greengrass.LambdaLauncher`) 处理进程和环境配置。
- Lambda 管理器组件 (`aws.greengrass.LambdaManager`) 处理进程间通信和扩展。
- Lambda 运行时组件 (`aws.greengrass.LambdaRuntimes`) 为每个支持的 Lambda 运行时提供构件。

主题

- 要求
- 配置 Lambda 函数生命周期
- 配置 Lambda 函数容器化
- 将 Lambda 函数作为组件导入（控制台）
- 将 Lambda 函数作为组件导入 () AWS CLI

要求

您的核心设备和 Lambda 函数必须满足以下要求才能在AWS IoT Greengrass核心软件上运行这些函数：

- 您的核心设备必须满足运行 Lambda 函数的要求。如果您希望核心设备运行容器化的 Lambda 函数，则该设备必须满足要求。有关更多信息，请参阅 Lambda 函数要求。
- 您必须在核心设备上安装 Lambda 函数使用的编程语言。

Tip

您可以创建一个安装编程语言的组件，然后将该组件指定为 Lambda 函数组件的依赖项。Greengrass 支持所有 Lambda 支持的 Python、Node.js 和 Java 运行时版本。Greengrass 不对已弃用的 Lambda 运行时版本施加任何额外限制。您可以在上运行使用这些已弃用运行时的 Lambda 函数AWS IoT Greengrass，但不能在中创建它们。AWS Lambda有关 AWS IoT Greengrass 对 Lambda 运行时的支持的更多信息，请参阅运行AWS Lambda函数。

配置 Lambda 函数生命周期

Greengrass Lambda 函数生命周期确定函数何时启动以及它如何创建和使用容器。生命周期还决定了 AWS IoT Greengrass核心软件如何保留函数处理程序之外的变量和预处理逻辑。

AWS IoT Greengrass支持按需（默认）和长生命周期：

- 按需函数在调用时启动，在没有任务要运行时停止。除非现有容器可供重复使用，否则每次调用该函数都会创建一个单独的容器（也称为沙箱）来处理调用。任何容器都可能处理您发送到函数的数据。

  按需函数的多次调用可以同时运行。

  创建新容器时，在函数处理程序之外定义的变量和预处理逻辑不会被保留。
- 长寿命（或固定）功能从 AWS IoT Greengrass Core 软件启动时开始，并在单个容器中运行。同一个容器处理您发送到该函数的所有数据。

  多个调用将排队，直到 C AWS IoT Greengrass ore 软件运行较早的调用。

  每次调用处理程序时，都会保留您在函数处理程序之外定义的变量和预处理逻辑。

  当您需要在没有任何初始输入的情况下开始工作时，请使用寿命长的 Lambda 函数。例如，长寿命函数可以加载并开始处理机器学习模型，以便在函数接收设备数据时准备就绪。

Note

长寿命函数的超时时间与其处理程序的每次调用相关联。如果要调用无限期运行的代码，则必须在处理程序之外启动它。确保处理程序之外没有可能阻止函数初始化的阻塞代码。

除非AWS IoT Greengrass核心软件停止（例如在部署或重启期间），否则这些功能才会运行。如果函数遇到未捕获的异常、超出其内存限制或进入错误状态（例如处理程序超时），则这些函数将无法运行。

有关容器重用的更多信息，请参阅 AWSCompute 博客AWS Lambda中的了解容器重用。

配置 Lambda 函数容器化

默认情况下，Lambda 函数在容器AWS IoT Greengrass内运行。Greengrass 容器在您的函数和主机之间提供隔离。这种隔离提高了主机和容器中函数的安全性。

我们建议您在 Greengrass 容器中运行 Lambda 函数，除非您的用例要求它们在不进行容器化的情况下运行。通过在 Greengrass 容器中运行 Lambda 函数，您可以更好地控制如何限制对资源的访问。

在以下情况下，您可以运行不带容器化的 Lambda 函数：

- 你想AWS IoT Greengrass在不支持容器模式的设备上运行。例如，如果您想使用特殊的 Linux 发行版，或者使用已过时的早期内核版本。
- 您需要在另一个有自己的 OverlayFS 的容器环境中运行 Lambda 函数，但在 Greengrass 容器中运行时遇到了 OverlayFS 冲突。
- 您需要访问本地资源，其路径在部署时无法确定，或者其路径在部署后可能会发生变化。此资源的一个例子是可插拔设备。
- 你有一个早期的应用程序是作为进程编写的，当你在 Greengrass 容器中运行它时，你会遇到问题。

容器化差异

容器化	注意事项
Greengrass 容器	• 在 Greengrass 容器中运行 Lambda 函数时，所有 AWS IoT Greengrass 功能均可用。 • 在 Greengrass 容器中运行的 Lambda 函数无权访问其他 Lambda 函数的已部署代码，即使

容器化	注意事项
	它们在同一个系统组中运行也是如此。换句话说，您的 Lambda 函数在运行时彼此之间的隔离程度有所提高。 • 由于 AWS IoT Greengrass Core 软件在 Lambda 函数所在的容器中运行所有子进程，因此当 Lambda 函数停止时，子进程就会停止。

无容器	• 以下功能不适用于非容器化的 Lambda 函数：
	  • Lambda 函数内存限制。
	  • 本地设备和卷资源。您必须使用这些资源在核心设备上的文件路径而不是作为 Lambda 函数资源来访问这些资源。
	• 如果您的非容器化 Lambda 函数访问机器学习资源，则必须标识资源所有者并设置对资源（而不是 Lambda 函数）的访问权限。
	• 非容器化的 Lambda 函数对在同一系统组中运行的其他 Lambda 函数的已部署代码具有只读访问权限。

如果您在部署 Lambda 函数时更改其容器化，则该函数可能无法按预期运行。如果 Lambda 函数使用的本地资源在新的容器化设置下不再可用，则部署将失败。

- 当您将 Lambda 函数从在 Greengrass 容器中运行更改为不使用容器化运行时，该函数的内存限制将被丢弃。您必须直接访问文件系统，而不是使用附加的本地资源。在部署 Lambda 函数之前，必须移除所有附加的资源。
- 将 Lambda 函数从在不进行容器化的情况下运行更改为在容器中运行时，Lambda 函数将失去对文件系统的直接访问权限。必须为每个函数定义内存限制或接受默认的 16 MB 内存限制。在部署每个 Lambda 函数时，您可以为其配置这些设置。

要更改 Lambda 函数组件的容器化设置，请在部署该组件时将`containerMode`配置参数的值设置为以下选项之一。

- `NoContainer`— 该组件不在隔离的运行时环境中运行。

- `GreengrassContainer`— 该组件在AWS IoT Greengrass容器内的隔离运行时环境中运行。

有关如何部署和配置组件的更多信息，请参阅将AWS IoT Greengrass组件部署到设备和更新组件配置。

将 Lambda 函数作为组件导入（控制台）

使用AWS IoT Greengrass控制台创建 Lambda 函数组件时，需要导入现有AWS Lambda函数，然后将其配置为创建在 Greengrass 设备上运行的组件。

在开始之前，请查看在 Greengr ass 设备上运行 Lambda 函数的要求。

任务

- 步骤 1：选择要导入的 Lambda 函数
- 步骤 2：配置 Lambda 函数参数
- 步骤 3：（可选）为 Lambda 函数指定支持的平台
- 步骤 4：（可选）为 Lambda 函数指定组件依赖关系
- 步骤 5：（可选）在容器中运行 Lambda 函数
- 步骤 6：创建 Lambda 函数组件

步骤 1：选择要导入的 Lambda 函数

1. 在AWS IoT Greengrass控制台导航菜单中，选择组件。
2. 在组件页面上，选择创建组件。
3. 在创建组件页面的组件信息下，选择导入 Lambda 函数。
4. 在 Lambda 函数中，搜索并选择要导入的 Lambda 函数。

   AWS IoT Greengrass创建名为 Lambda 函数的组件。
5. 在 Lambda 函数版本中，选择要导入的版本。你不能像这样选择 Lambda 别名。`$LATEST`

   AWS IoT Greengrass使用 Lambda 函数的版本作为有效的语义版本创建组件。例如，如果您的函数版本是 `3`，则组件版本将变为 `3.0.0`。

步骤 2：配置 Lambda 函数参数

在创建组件页面的 Lambda 函数配置下，配置用于运行 Lambda 函数的以下参数。

1. （可选）添加 Lambda 函数订阅的工作消息的事件源列表。您可以指定事件源来订阅此函数以订阅本地发布/订阅消息和 AWS IoT Core MQTT 消息。Lambda 函数在收到来自事件源的消息时被调用。

   > **Note**
   >
   > 要将此函数订阅来自其他 Lambda 函数或组件的消息，请在部署此 Lambda 函数组件时部署旧版订阅路由器组件。部署旧版订阅路由器组件时，请指定 Lambda 函数使用的订阅。

   在 “事件源” 下，执行以下操作以添加事件源：

   a. 对于您添加的每个事件源，请指定以下选项：

      - 主题-订阅消息的主题。
      - 类型-事件源的类型。从以下选项中进行选择：
        - 本地发布/订阅-订阅本地发布/订阅消息。

          如果您使用 Greengrass nucleus v2.6.0 或更高版本以及 Lambda manager v2.2.5 或更高版本，则在指定此类型时，可以在主题中使用 MQTT 主题通配符（和）。**+ #**
        - AWS IoT CoreMQTT — 订阅 AWS IoT Core MQTT 消息。

          当您指定此类型时，可以在主题中使用 MQTT 主题通配符（+和#）。

   b. 要添加其他事件源，请选择添加事件源并重复上一步操作。要移除事件源，请选择要移除的事件源旁边的 “移除”。

2. 对于超时（秒），输入非固定 Lambda 函数在超时之前可以运行的最大时间（秒）。默认值为 3 秒。

3. 对于已固定，请选择是否固定 Lambda 函数组件。默认值为 True。

   - 固定的（或长寿命的）Lambda 函数在启动时AWS IoT Greengrass启动，并在自己的容器中继续运行。
   - 非固定（或按需）Lambda 函数仅在收到工作项目时启动，并在指定的最大空闲时间内保持空闲状态后退出。如果该函数有多个工作项，AWS IoT Greengrass Core 软件将创建该函数的多个实例。

4. （可选）在 “其他参数” 下，设置以下 Lambda 函数参数。

    - 状态超时（秒）-Lambda 函数组件向 Lambda 管理器组件发送状态更新的间隔（以秒为单位）。此参数仅适用于固定函数。默认值为 60 秒。
    - 最大队列大小-Lambda 函数组件的消息队列的最大大小。AWS IoT GreengrassCore 软件将消息存储在 FIFO（先进先出）队列中，直到它可以运行 Lambda 函数来使用每条消息。默认值为 1,000 条消息。
    - 最大实例数-非固定 Lambda 函数可以同时运行的最大实例数。默认值为 100 个实例。
    - 最大空闲时间（秒）-在AWS IoT Greengrass核心软件停止其进程之前，非固定 Lambda 函数可以空闲的最大时间（以秒为单位）。默认值为 60 秒。
    - 编码类型-Lambda 函数支持的负载类型。从以下选项中进行选择：
        - JSON
        - 二进制

      默认值为 JSON。

5. （可选）指定在 Lambda 函数运行时要传递给 Lambda 函数的命令行参数列表。

    a. 在 “其他参数” 的 “处理参数” 下，选择 “添加参数”。

    b. 对于您添加的每个参数，输入要传递给该函数的参数。

    c. 要删除参数，请选择要删除的参数旁边的 “删除”。

6. （可选）指定 Lambda 函数在运行时可用的环境变量。环境变量使您无需更改函数代码即可存储和更新配置设置。

    a. 在 “其他参数” 的 “环境变量” 下，选择 “添加环境变量”。

    b. 对于您添加的每个环境变量，请指定以下选项：

        - Key-变量名。
        - 值-此变量的默认值。

    c. 要移除环境变量，请选择要移除的环境变量旁边的移除。

步骤 3：（可选）为 Lambda 函数指定支持的平台

所有核心设备都具有操作系统和架构的属性。当您部署 Lambda 函数组件时，AWS IoT GreengrassCore 软件会将您指定的平台值与核心设备上的平台属性进行比较，以确定该设备是否支持 Lambda 函数。

Note

在将 Greengrass nucleus 组件部署到核心设备时，也可以指定自定义平台属性。有关更多信息，请参阅 Gre engrass nucleus 组件的平台覆盖参数。

在 Lambda 函数配置下的其他参数 “平台” 下，执行以下操作以指定此 Lambda 函数支持的平台。

1. 对于每个平台，请指定以下选项：
   - 操作系统-平台操作系统的名称。目前，唯一支持的值是 `linux`。
   - 架构-平台的处理器架构。支持的值为：
     - `amd64`
     - `arm`
     - `aarch64`
     - `x86`
2. 要添加其他平台，请选择添加平台并重复上一步操作。要移除支持的平台，请选择要移除的平台旁边的 “移除”。

步骤 4：（可选）为 Lambda 函数指定组件依赖关系

组件依赖关系AWS可识别您的函数使用的其他由提供的组件或自定义组件。当您部署 Lambda 函数组件时，部署中会包含这些依赖关系供您的函数运行。

Important

要导入您创建的在 AWS IoT Greengrass V1 上运行的 Lambda 函数，必须为函数使用的功能（例如机密、本地阴影和流管理器）定义各个组件依赖关系。将这些组件定义为硬依赖关系，这样 Lambda 函数组件在依赖关系状态发生变化时会重新启动。有关更多信息，请参阅 导入 V1 Lambda 函数。

在 Lambda 函数配置 “其他参数”、“组件依赖关系” 下，完成以下步骤，为 Lambda 函数指定组件依赖关系。

1. 选择 “添加依赖关系”。
2. 对于您添加的每个组件依赖关系，请指定以下选项：

- 组件名称-组件名称。例如，输入**aws.greengrass.StreamManager**以包含直播管理器组件。
- 版本要求 — npm 风格的语义版本约束，用于标识此组件依赖关系的兼容版本。您可以指定单个版本或一系列版本。例如，输入**^1.0.0**以指定此 Lambda 函数依赖于流管理器组件的第一个主要版本中的任何版本。有关语义版本限制的更多信息，请参阅 npm sem ver 计算器。
- 类型-依赖关系的类型。从以下选项中进行选择：
  - 困难 — 如果依赖关系状态发生变化，Lambda 函数组件将重新启动。这是默认选择。
  - S@@ oft — 如果依赖关系更改状态，Lambda 函数组件不会重新启动。

3. 要移除组件依赖关系，请选择组件依赖关系旁边的 “删除”

步骤 5：（可选）在容器中运行 Lambda 函数

默认情况下，Lambda 函数在AWS IoT Greengrass核心软件内部的隔离运行时环境中运行。您也可以选择将 Lambda 函数作为不带任何隔离的进程运行（即在 “无容器” 模式下）。

在 Linux 进程配置下，对于隔离模式，从以下选项中进行选择，为您的 Lambda 函数选择容器化：

- Greengrass 容器 — Lambda 函数在容器中运行。这是默认选择。
- 无容器 — Lambda 函数作为进程运行，没有任何隔离。

如果您在容器中运行 Lambda 函数，请完成以下步骤来配置 Lambda 函数的流程配置。

1. 配置容器可用的内存量和系统资源，例如卷和设备。

   在 “容器参数” 下，执行以下操作。

   a. 在内存大小中，输入要分配给容器的内存大小。您可以以 MB 或 K B 为单位指定内存大小。
   b. 对于只读的 sys 文件夹，请选择容器是否可以读取设备`/sys`文件夹中的信息。默认值为 False。

2. （可选）配置容器化 Lambda 函数可以访问的本地卷。定义卷时，AWS IoT Greengrass Core 软件会将源文件挂载到容器内的目标。

   a. 在 “卷” 下，选择 “添加卷”。
   b. 对于您添加的每个卷，请指定以下选项：

      - 物理卷-核心设备上源文件夹的路径。

- 逻辑卷-容器中目标文件夹的路径。
- 权限-（可选）从容器访问源文件夹的权限。从以下选项中进行选择：
  - 只读-Lambda 函数对源文件夹具有只读访问权限。这是默认选择。
  - 读写-Lambda 函数具有对源文件夹的读/写访问权限。
- 添加群组所有者 —（可选）是否将运行 Lambda 函数组件的系统群组添加为源文件夹的所有者。默认值为 False。

c. 要移除卷，请选择要移除的卷旁边的 “移除”。

3. （可选）配置容器化 Lambda 函数可以访问的本地系统设备。

   a. 在 “设备” 下，选择 “添加设备”。

   b. 对于您添加的每台设备，请指定以下选项：

   - 挂载路径-核心设备上系统设备的路径。
   - 权限-（可选）从容器访问系统设备的权限。从以下选项中进行选择：
     - 只读-Lambda 函数对系统设备具有只读访问权限。这是默认选择。
     - 读写-Lambda 函数具有对源文件夹的读/写访问权限。
   - 添加群组所有者 —（可选）是否将运行 Lambda 函数组件的系统组添加为系统设备的所有者。默认值为 False。

步骤 6：创建 Lambda 函数组件

为 Lambda 函数组件配置设置后，选择创建以完成新组件的创建。

要在核心设备上运行 Lambda 函数，您可以将新组件部署到核心设备。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。

将 Lambda 函数作为组件导入 () AWS CLI

使用CreateComponentVersion操作从 Lambda 函数创建组件。当您调用此操作时，请指定导`lambdaFunction`入 Lambda 函数。

任务

- 步骤 1：定义 Lambda 函数配置
- 步骤 2：创建 Lambda 函数组件

步骤 1：定义 Lambda 函数配置

1. 创建一个名为的文件`lambda-function-component.json`，然后将以下 JSON 对象复制到该文件中。将替换为`lambdaArn`要导入的 Lambda 函数的 ARN。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1"
  }
}
```

> **Important**
>
> 您必须指定包含要导入的函数版本的 ARN。您不能使用像 `$LATEST` 这样的版本别名。

2. （可选）指定组件的名称 (`componentName`)。如果省略此参数，则使用 Lambda 函数的名称AWS IoT Greengrass创建组件。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda"
  }
}
```

3. （可选）指定组件的版本 (`componentVersion`)。如果省略此参数，则使用 Lambda 函数版本作为有效语义版本AWS IoT Greengrass创建组件。例如，如果您的函数版本是 `3`，则组件版本将变为 `3.0.0`。

> **Note**
>
> 您上传的每个组件版本都必须是唯一的。请务必上传正确的组件版本，因为上传后无法对其进行编辑。
>
> AWS IoT Greengrass使用组件的语义版本。语义版本遵循 major.minor.patch 编号系统。例如，版本`1.0.0`表示组件的第一个主要版本。有关更多信息，请参阅[语义版本规范](https://semver.org/)。

```
{
  "lambdaFunction": {
```

```
      "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
      "componentName": "com.example.HelloWorldLambda",
      "componentVersion": "1.0.0"
    }
  }
```

4. （可选）指定此 Lambda 函数支持的平台。每个平台都包含用于标识平台的属性地图。所有核心设备都具有操作系统 (os) 和架构 (architecture) 的属性。C AWS IoT Greengrass ore 软件可能会添加其他平台属性。在将 Greengrass nucleus 组件部署到核心设备时，也可以指定自定义平台属性。执行以下操作：

   a. 向中的 Lambda 函数添加平台列表 (componentPlatforms)。lambda-function-component.json

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
       "componentName": "com.example.HelloWorldLambda",
       "componentVersion": "1.0.0",
       "componentPlatforms": [

       ]
     }
   }
   ```

   b. 将每个支持的平台添加到列表中。每个平台都有一个便name于识别的平台和一个属性地图。以下示例指定此函数支持运行 Linux 的 x86 设备。

   ```
   {
     "name": "Linux x86",
     "attributes": {
       "os": "linux",
       "architecture": "x86"
     }
   }
   ```

   您lambda-function-component.json可能包含类似于以下示例的文档。

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
       "componentName": "com.example.HelloWorldLambda",
   ```

```
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ]
  }
}
```

5. （可选）为您的 Lambda 函数指定组件依赖关系。当您部署 Lambda 函数组件时，部署中会包含这些依赖项，供您的函数运行。

> **Important**
>
> 要导入您创建的在 AWS IoT Greengrass V1 上运行的 Lambda 函数，必须为函数使用的功能（例如机密、本地阴影和流管理器）定义各个组件依赖关系。将这些组件定义为硬依赖关系，以便在依赖关系状态发生变化时，您的 Lambda 函数组件会重新启动。有关更多信息，请参阅 导入 V1 Lambda 函数。

执行以下操作：

a. 将组件依赖关系映射 (`componentDependencies`) 添加到中的 Lambda 函数。`lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
```

```
    ],
    "componentDependencies": {

    }
  }
}
```

b. 将每个组件依赖关系添加到地图中。将组件名称指定为键，并使用以下参数指定对象：

- `versionRequirement`— npm 样式的语义版本约束，用于标识组件依赖关系的兼容版本。您可以指定单个版本或一系列版本。有关语义版本限制的更多信息，请参阅 npm semver 计算器。
- `dependencyType`—（可选）依赖项的类型。请从以下内容中选择：
  - `SOFT`— 如果依赖关系更改状态，Lambda 函数组件不会重新启动。
  - `HARD`— 如果依赖关系状态发生变化，Lambda 函数组件将重新启动。

  默认值为 `HARD`。

以下示例指定此 Lambda 函数依赖于流管理器组件的第一个主要版本中的任何版本。当流管理器重新启动或更新时，Lambda 函数组件会重新启动。

```
{
  "aws.greengrass.StreamManager": {
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
  }
}
```

您`lambda-function-component.json`可能包含类似于以下示例的文档。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
```

```
            "architecture": "x86"
          }
        }
      ],
      "componentDependencies": {
        "aws.greengrass.StreamManager": {
          "versionRequirement": "^1.0.0",
          "dependencyType": "HARD"
        }
      }
    }
}
```

6. （可选）配置用于运行该函数的 Lambda 函数参数。您可以配置诸如环境变量、消息事件源、超时和容器设置之类的选项。执行以下操作：

   a. 将 Lambda 参数对象 (`componentLambdaParameters`) 添加到中的 Lambda 函数中。`lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {

    }
  }
```

```
}
```

b. （可选）指定 Lambda 函数订阅的工作消息的事件源。您可以指定事件源来订阅此函数以订阅本地发布/订阅消息和 AWS IoT Core MQTT 消息。Lambda 函数在收到来自事件源的消息时被调用。

> **Note**
>
> 要将此函数订阅来自其他 Lambda 函数或组件的消息，请在部署此 Lambda 函数组件时部署旧版订阅路由器组件。部署旧版订阅路由器组件时，请指定 Lambda 函数使用的订阅。

执行以下操作：

i. 将事件源列表 (eventSources) 添加到 Lambda 函数参数中。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [

      ]
    }
```

```
  }
}
```

ii. 将每个事件源添加到列表中。每个事件源都有以下参数：

- topic— 订阅消息的主题。
- type— 事件源的类型。从以下选项中进行选择：
  - PUB_SUB – 订阅本地发布/订阅消息。

    如果您使用 Greengrass nucleus v2.6.0 或更高版本以及 Lambda manager v2.2.5 或更高版本，则可以在指定此类型时在中使用 MQTT 主题通配符（和）。+ # topic
  - IOT_CORE – 订阅 AWS IoT Core MQTT 消息。

    当您指定此类型topic时，可以在中使用 MQTT 主题通配符（+和#）。

以下示例在与主题筛选条件匹配的主题上订阅 AWS IoT Core MQTT。hello/world/+

```
{
  "topic": "hello/world/+",
  "type": "IOT_CORE"
}
```

lambda-function-component.json可能与以下示例类似。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
```

```
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ]
    }
  }
}
```

c. （可选）在 Lambda 函数参数对象中指定以下任意参数：

- `environmentVariables`— Lambda 函数在运行时可用的环境变量映射。
- `execArgs`— Lambda 函数运行时要传递给它的参数列表。
- `inputPayloadEncodingType`— Lambda 函数支持的负载类型。从以下选项中进行选择：
  - `json`
  - `binary`

  默认值：`json`
- `pinned`— Lambda 函数是否已固定。默认值为 `true`。
  - 固定的（或长寿命的）Lambda 函数在启动时AWS IoT Greengrass启动，并在自己的容器中继续运行。
  - 非固定（或按需）Lambda 函数仅在收到工作项目时启动，并在指定的最大空闲时间内保持空闲状态后退出。如果该函数有多个工作项，AWS IoT Greengrass Core 软件将创建该函数的多个实例。

  `maxIdleTimeInSeconds`用于设置函数的最大空闲时间。
- `timeoutInSeconds`— Lambda 函数在超时之前可以运行的最大时间（以秒为单位）。默认值为 3 秒。
- `statusTimeoutInSeconds`— Lambda 函数组件向 Lambda 管理器组件发送状态更新的间隔（以秒为单位）。此参数仅适用于固定函数。默认值为 60 秒。

- `maxIdleTimeInSeconds`— 在AWS IoT Greengrass核心软件停止其进程之前，非固定 Lambda 函数可以空闲的最大时间（以秒为单位）。默认值为 60 秒。
- `maxInstancesCount`— 非固定 Lambda 函数可以同时运行的最大实例数。默认值为 100 个实例。
- `maxQueueSize`— Lambda 函数组件的消息队列的最大大小。AWS IoT Greengrass核心软件将消息存储在 FIFO (first-in-first-out) 队列中，直到它可以运行 Lambda 函数来使用每条消息。默认值为 1,000 条消息。

您`lambda-function-component.json`可能包含类似于以下示例的文档。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
```

```
          "-d"
        ],
        "inputPayloadEncodingType": "json",
        "pinned": true,
        "timeoutInSeconds": 120,
        "statusTimeoutInSeconds": 30,
        "maxIdleTimeInSeconds": 30,
        "maxInstancesCount": 50,
        "maxQueueSize": 500
      }
    }
}
```

d. （可选）为 Lambda 函数配置容器设置。默认情况下，Lambda 函数在AWS IoT Greengrass 核心软件内部的隔离运行时环境中运行。您也可以选择将 Lambda 函数作为不进行任何隔离的进程运行。如果您在容器中运行 Lambda 函数，则需要配置容器的内存大小以及 Lambda 函数可用的系统资源。执行以下操作：

i. 将 Linux 进程参数对象 (`linuxProcessParams`) 添加到中的 Lambda 参数对象。`lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
```

```
          {
            "topic": "hello/world/+",
            "type": "IOT_CORE"
          }
        ],
        "environmentVariables": {
          "LIMIT": "300"
        },
        "execArgs": [
          "-d"
        ],
        "inputPayloadEncodingType": "json",
        "pinned": true,
        "timeoutInSeconds": 120,
        "statusTimeoutInSeconds": 30,
        "maxIdleTimeInSeconds": 30,
        "maxInstancesCount": 50,
        "maxQueueSize": 500,
        "linuxProcessParams": {

        }
      }
    }
}
```

ii. （可选）指定 Lambda 函数是否在容器中运行。将`isolationMode`参数添加到流程参数对象，然后从以下选项中进行选择：

- `GreengrassContainer`— Lambda 函数在容器中运行。
- `NoContainer`— Lambda 函数作为一个没有任何隔离的进程运行。

默认值为 `GreengrassContainer`。

iii. （可选）如果您在容器中运行 Lambda 函数，则可以配置内存量和系统资源，例如卷和设备，以供容器使用。执行以下操作：

A. 将容器参数对象 (`containerParams`) 添加到中的 Linux 进程参数对象`lambda-function-component.json`。

```
{
  "lambdaFunction": {
```

```
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {

        }
      }
```

```
      }
    }
  }
```

B. （可选）添加memorySizeInKB参数以指定容器的内存大小。默认值为 16,384 KB (16 MB)。

C. （可选）添加mountROSysfs参数以指定容器是否可以读取设备/sys文件夹中的信息。默认值为 false。

D. （可选）配置容器化 Lambda 函数可以访问的本地卷。定义卷时，AWS IoT Greengrass Core 软件会将源文件挂载到容器内的目标。执行以下操作：

   I. 将卷列表 (volumes) 添加到容器参数中。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
```

```
          "LIMIT": "300"
        },
        "execArgs": [
          "-d"
        ],
        "inputPayloadEncodingType": "json",
        "pinned": true,
        "timeoutInSeconds": 120,
        "statusTimeoutInSeconds": 30,
        "maxIdleTimeInSeconds": 30,
        "maxInstancesCount": 50,
        "maxQueueSize": 500,
        "linuxProcessParams": {
          "containerParams": {
            "memorySizeInKB": 32768,
            "mountROSysfs": true,
            "volumes": [

            ]
          }
        }
      }
    }
}
```

II. 将每个卷添加到列表中。每个卷都有以下参数：

- `sourcePath`— 核心设备上源文件夹的路径。
- `destinationPath`— 容器中目标文件夹的路径。
- `permission`—（可选）从容器访问源文件夹的权限。从以下选项中进行选择：
  - `ro`— Lambda 函数对源文件夹具有只读访问权限。
  - `rw`— Lambda 函数具有对源文件夹的读写访问权限。

  默认值为 `ro`。
- `addGroupOwner`—（可选）是否将运行 Lambda 函数组件的系统组添加为源文件夹的所有者。默认值为 `false`。

您`lambda-function-component.json`可能包含类似于以下示例的文档。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
```

```
            "memorySizeInKB": 32768,
            "mountROSysfs": true,
            "volumes": [
              {
                "sourcePath": "/var/data/src",
                "destinationPath": "/var/data/dest",
                "permission": "rw",
                "addGroupOwner": true
              }
            ]
          }
        }
      }
    }
}
```

E. （可选）配置容器化 Lambda 函数可以访问的本地系统设备。执行以下操作：

I. 将系统设备列表 (devices) 添加到容器参数中。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
```

```
            "topic": "hello/world/+",
            "type": "IOT_CORE"
          }
        ],
        "environmentVariables": {
          "LIMIT": "300"
        },
        "execArgs": [
          "-d"
        ],
        "inputPayloadEncodingType": "json",
        "pinned": true,
        "timeoutInSeconds": 120,
        "statusTimeoutInSeconds": 30,
        "maxIdleTimeInSeconds": 30,
        "maxInstancesCount": 50,
        "maxQueueSize": 500,
        "linuxProcessParams": {
          "containerParams": {
            "memorySizeInKB": 32768,
            "mountROSysfs": true,
            "volumes": [
              {
                "sourcePath": "/var/data/src",
                "destinationPath": "/var/data/dest",
                "permission": "rw",
                "addGroupOwner": true
              }
            ],
            "devices": [

            ]
          }
        }
      }
    }
}
```

II. 将每台系统设备添加到列表中。每台系统设备都有以下参数：

- path— 核心设备上系统设备的路径。
- permission—（可选）从容器访问系统设备的权限。从以下选项中进行选择：

  - `ro`— Lambda 函数对系统设备具有只读访问权限。
  - `rw`— Lambda 函数具有对系统设备的读写访问权限。

  默认值为 `ro`。
- `addGroupOwner`—（可选）是否将运行 Lambda 函数组件的系统组添加为系统设备的所有者。默认值为 `false`。

您`lambda-function-component.json`可能包含类似于以下示例的文档。

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
```

```
          "-d"
        ],
        "inputPayloadEncodingType": "json",
        "pinned": true,
        "timeoutInSeconds": 120,
        "statusTimeoutInSeconds": 30,
        "maxIdleTimeInSeconds": 30,
        "maxInstancesCount": 50,
        "maxQueueSize": 500,
        "linuxProcessParams": {
          "containerParams": {
            "memorySizeInKB": 32768,
            "mountROSysfs": true,
            "volumes": [
              {
                "sourcePath": "/var/data/src",
                "destinationPath": "/var/data/dest",
                "permission": "rw",
                "addGroupOwner": true
              }
            ],
            "devices": [
              {
                "path": "/dev/sda3",
                "permission": "rw",
                "addGroupOwner": true
              }
            ]
          }
        }
      }
    }
}
```

7. （可选）为组件添加标签 (`tags`)。有关更多信息，请参阅 标记 AWS IoT Greengrass Version 2 资源。

步骤 2：创建 Lambda 函数组件

1. 运行以下命令从中创建 Lambda 函数组件。`lambda-function-component.json`

```
aws greengrassv2 create-component-version --cli-input-json file://lambda-function-component.json
```

如果请求成功，则响应类似于以下示例。

```
{
  "arn":
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorldLambda:versions:1
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "Mon Dec 15 20:56:34 UTC 2020",
  "status": {
    "componentState": "REQUESTED",
    "message": "NONE",
    "errors": {}
  }
}
```

arn从输出中复制，以便在下一步中检查组件的状态。

2. 创建组件时，其状态为REQUESTED。然后，AWS IoT Greengrass验证该组件是否可部署。您可以运行以下命令来查询组件状态并验证您的组件是否可部署。arn用上一步中的 ARN 替换。

```
aws greengrassv2 describe-component \
  --arn "arn:aws:greengrass:region:account-id:components:com.example.HelloWorldLambda:versions:1.0.0"
```

如果组件通过验证，则响应表明组件状态为DEPLOYABLE。

```
{
  "arn": "arn:aws:greengrass:region:account-id:components:com.example.HelloWorldLambda:versions:1.0.0",
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-12-15T20:56:34.376000-08:00",
  "publisher": "AWS Lambda",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
```

```
  "platforms": [
    {
      "name": "Linux x86",
      "attributes": {
        "architecture": "x86",
        "os": "linux"
      }
    }
  ]
}
```

组件完成后DEPLOYABLE，您可以将 Lambda 函数部署到您的核心设备。有关更多信息，请参阅将AWS IoT Greengrass组件部署到设备。

使用AWS IoT Device SDK与 Greengrass 原子核、其他组件进行通信 AWS IoT Core

在核心设备上运行的组件可以使用中的AWS IoT Greengrass核心进程间通信 (IPC) 库与AWS IoT Greengrass核心和其他 Gre AWS IoT Device SDK engrass 组件进行通信。要开发和运行使用 IPC 的自定义组件，必须使用连接AWS IoT Device SDK到 C AWS IoT Greengrass ore IPC 服务并执行 IPC 操作。

IPC 接口支持两种类型的操作：

- 请求/响应

  组件向 IPC 服务发送请求并收到包含请求结果的响应。

- 订阅

  组件向 IPC 服务发送订阅请求，并期望事件消息流作为响应。组件提供了一个订阅处理程序，用于处理事件消息、错误和流关闭。AWS IoT Device SDK包括一个处理程序接口，其中包含每个 IPC 操作的正确响应和事件类型。有关更多信息，请参阅 订阅 IPC 事件直播。

主题

- IPC 客户端版本
- 支持用于进程间通信的 SDK
- Connect 到 C AWS IoT Greengrass ore IPC 服务
- 授权组件执行 IPC 操作
- 订阅 IPC 事件直播
- IPC 最佳实践
- 发布/订阅本地消息
- 发布/订阅 AWS IoT Core MQTT 消息
- 与组件生命周期交互
- 与组件配置交互
- 检索秘密值
- 与局部阴影互动
- 管理本地部署和组件

- 对客户端设备进行身份验证和授权

IPC 客户端版本

在更高版本的 Java 和 Python 软件开发工具包中，AWS IoT Greengrass提供了 IPC 客户端的改进版本，名为 IPC 客户端 V2。IPC 客户端 V2：

- 减少使用 IPC 操作所需编写的代码量，并有助于避免 IPC 客户端 V1 可能出现的常见错误。
- 在单独的线程中调用订阅处理程序回调，因此您现在可以在订阅处理程序回调中运行阻塞代码，包括其他 IPC 函数调用。IPC 客户端 V1 使用相同的线程与 IPC 服务器通信并调用订阅处理程序回调。
- 允许您使用 Lambda 表达式 (Java) 或函数 (Python) 调用订阅操作。IPC 客户端 V1 要求您定义订阅处理程序类。
- 提供每个 IPC 操作的同步和异步版本。IPC 客户端 V1 仅提供每个操作的异步版本。

我们建议您使用 IPC 客户端 V2 来利用这些改进。但是，本文档和一些在线内容中的许多示例仅演示如何使用 IPC 客户端 V1。您可以使用以下示例和教程来查看使用 IPC 客户端 V2 的示例组件：

- PublishToTopic例子
- SubscribeToTopic例子
- 教程：开发一个可以延迟组件更新的 Greengrass 组件
- 教程：通过 MQTT 与本地物联网设备交互

目前，AWS IoT Device SDK适用于 C++ 的 v2 仅支持 IPC 客户端 V1。

支持用于进程间通信的 SDK

C AWS IoT Greengrass ore IPC 库包含在以下AWS IoT Device SDK版本中。

SDK	最低版本	使用量	
AWS IoT Device SDK 适用于 Java v2	v1.6.0	请参阅 用AWS IoT Device SDK于 Java v2（IPC 客户端 V2）。	

SDK	最低版本	使用量
AWS IoT Device SDK 适用于 Python v2	v1.9.0	请参阅 用AWS IoT Device SDK于 Python v2（IPC 客户端 V2）。
AWS IoT Device SDK 适用于 C++ v2	v1.17.0	请参阅 用AWS IoT Device SDK于 C++ v2。
AWS IoT Device SDK 适用于 JavaScript v2	v1.12.0	请参阅 用AWS IoT Device SDK于 JavaScript v2（IPC 客户端 V1）。

Connect 到 C AWS IoT Greengrass ore IPC 服务

要在自定义组件中使用进程间通信，必须创建与 C AWS IoT Greengrass ore 软件运行的 IPC 服务器套接字的连接。完成以下任务，下载并使用您选择AWS IoT Device SDK的语言。

用AWS IoT Device SDK于 Java v2（IPC 客户端 V2）

要使用AWS IoT Device SDK适用于 Java v2（IPC 客户端 V2）

1. 下载AWS IoT Device SDK适用于 Java 版本 2（1.6.0 或更高版本）的。
2. 执行以下任一操作以在组件中运行您的自定义代码：
   - 将您的组件构建为包含的 JAR 文件AWS IoT Device SDK，然后在组件配方中运行此 JAR 文件。
   - 将 AWS IoT Device SDK JAR 定义为组件工件，并在组件配方中运行应用程序时将该构件添加到类路径中。
3. 使用以下代码创建 IPC 客户端。

```
try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
    // Use client.
} catch (Exception e) {
```

```
    LOGGER.log(Level.SEVERE, "Exception occurred when using IPC.", e);
    System.exit(1);
}
```

用AWS IoT Device SDK于 Python v2（IPC 客户端 V2）

要使用AWS IoT Device SDK适用于 Python v2（IPC 客户端 V2）

1. 下载AWS IoT Device SDK适用于 Python 的（1.9.0 或更高版本）。
2. 将 SDK 的安装步骤添加到组件配方中的安装生命周期中。
3. 创建与AWS IoT Greengrass核心 IPC 服务的连接。使用以下代码创建 IPC 客户端。

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

try:
    ipc_client = GreengrassCoreIPCClientV2()
    # Use IPC client.
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

用AWS IoT Device SDK于 C++ v2

要构建 C++ 版 AWS IoT Device SDK v2，设备必须具有以下工具：

- C++ 11 或更高版本
- cMake 3.1 或更高版本
- 以下编译器之一：
  - GCC 4.8 或更高版本
  - Clang 3.9 或更高版本
  - MSVC 2015 或更高版本

要使用AWS IoT Device SDK适用于 C++ v2 的

1. 下载AWS IoT Device SDK适用于 C++ v2（1.17.0 或更高版本）的。

2. 按照自述文件中的安装说明从源代码构建 C++ v2 版。AWS IoT Device SDK
3. 在你的 C++ 构建工具中，链接你在上一步中构建的 Greengrass IPC 库`AWS::GreengrassIpc-cpp`。以下`CMakeLists.txt`示例将 Greengrass IPC 库链接到您使用 CMake 构建的项目。

```
cmake_minimum_required(VERSION 3.1)
project (greengrassv2_pubsub_subscriber)

file(GLOB MAIN_SRC
        "*.h"
        "*.cpp"
        )
add_executable(${PROJECT_NAME} ${MAIN_SRC})

set_target_properties(${PROJECT_NAME} PROPERTIES
        LINKER_LANGUAGE CXX
        CXX_STANDARD 11)
find_package(aws-crt-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(EventstreamRpc-cpp PATHS ~/sdk-cpp-workspace/build)
find_package(GreengrassIpc-cpp PATHS ~/sdk-cpp-workspace/build)
target_link_libraries(${PROJECT_NAME} AWS::GreengrassIpc-cpp)
```

4. 在组件代码中，创建与 C AWS IoT Greengrass ore IPC 服务的连接以创建 IPC 客户端 () `Aws::Greengrass::GreengrassCoreIpcClient`。必须定义一个处理 IPC 连接、断开连接和错误事件的 IPC 连接生命周期处理程序。以下示例创建了一个 IPC 客户端和一个 IPC 连接生命周期处理程序，它们在 IPC 客户端连接、断开连接和遇到错误时进行打印。

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() <<
 std::endl;
```

```
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    // Create the IPC client.
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    // Use the IPC client to create an operation request.

    // Activate the operation request.
    auto activate = operation.Activate(request, nullptr);
    activate.wait();

    // Wait for Greengrass Core to respond to the request.
    auto responseFuture = operation.GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
        exit(-1);
    }

    // Check the result of the request.
    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully published to topic: " << topic << std::endl;
    } else {
```

```
        // An error occurred.
        std::cout << "Failed to publish to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    return 0;
}
```

5. 要在组件中运行自定义代码，请将代码构建为二进制构件，然后在组件配方中运行二进制构件。将构件的Execute权限设置为OWNER，以使AWS IoT Greengrass核心软件能够运行二进制工件。

   您的组件配方Manifests部分可能与以下示例类似。

   JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pubsub_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
...
Manifests:
  - Lifecycle:
      run: {artifacts:path}/greengrassv2_pubsub_subscriber
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber
        Permission:
          Execute: OWNER
```

用AWS IoT Device SDK于 JavaScript v2（IPC 客户端 V1）

要编译AWS IoT Device SDK适用于 JavaScript v2 的，以便与 NodeJS 一起使用，设备必须具有以下工具：

- NodeJS 10.0 或更高版本
  - 运行`node -v`以检查 Node 版本。
- cMake 3.1 或更高版本

要使用AWS IoT Device SDK适用于 JavaScript v2 的（IPC 客户端 V1）

1. 下载AWS IoT Device SDK适用于 JavaScript v2（v 1.12.10 或更高版本）的。
2. 按照自述文件中的安装说明从源代码构建 JavaScript v2 版。AWS IoT Device SDK
3. 创建与AWS IoT Greengrass核心 IPC 服务的连接。完成以下步骤创建 IPC 客户端并建立连接。
4. 使用以下代码创建 IPC 客户端。

```
import * as greengrascoreipc from 'aws-iot-device-sdk-v2';

let client = greengrascoreipc.createClient();
```

5. 使用以下代码建立从您的组件到 Greengrass 核的连接。

```
await client.connect();
```

授权组件执行 IPC 操作

要允许您的自定义组件使用某些 IPC 操作，您必须定义允许该组件对某些资源执行操作的授权策略。每个授权策略都定义了该策略允许的操作列表和资源列表。例如，发布/订阅消息 IPC 服务定义了主题资源的发布和订阅操作。您可以使用*通配符来允许访问所有操作或所有资源。

您可以使用accessControl配置参数定义授权策略，可以在组件配方中或部署组件时设置该参数。该accessControl对象将 IPC 服务标识符映射到授权策略列表。您可以为每个 IPC 服务定义多个授权策略来控制访问权限。每个授权策略都有一个策略 ID，它在所有组件中必须是唯一的。

> **Tip**
>
> 要创建唯一的策略 ID，可以组合组件名称、IPC 服务名称和计数器。例如，名为的组件com.example.HelloWorld可以定义两个具有以下 ID 的发布/订阅授权策略：
>
> - com.example.HelloWorld:pubsub:1
> - com.example.HelloWorld:pubsub:2

授权策略使用以下格式。此对象是accessControl配置参数。

JSON

```
{
  "IPC service identifier": {
    "policyId": {
      "policyDescription": "description",
      "operations": [
        "operation1",
        "operation2"
      ],
      "resources": [
        "resource1",
        "resource2"
      ]
    }
  }
}
```

YAML

```
IPC service identifier:
  policyId:
    policyDescription: description
    operations:
      - operation1
      - operation2
    resources:
      - resource1
      - resource2
```

授权策略中的通配符

您可以在 IPC 授权策略resources元素中使用*通配符来允许访问单个授权策略中的多个资源。

- 在所有版本的 Greengrass nucleus 中，您可以指定*单个角色作为资源以允许访问所有资源。
- 在 Greengrass nucleus v2.6.0 及更高版本中，您可以在资源中指定字符以匹配任意字符*组合。例如，您可以指定factory/1/devices/Thermostat*/status允许访问工厂中所有恒温器设备的状态主题，其中每台设备的名称都以 Thermostat “开头”。

在为 AWS IoT Core MQTT IPC 服务定义授权策略时，也可以使用 MQTT 通配符（+和#）来匹配多个资源。有关更多信息，请参阅 MQTT IPC 授权策略中的 AWS IoT Core MQTT 通配符。

授权策略中的配方变量

如果你使用 Greengrass nucleus v2.6.0 或更高版本，并且将 Greeng interpolateComponentConfigurationrass nucleus 的配置选项设置为，则可以在授权策略中使用配方变量。true{iot:thingName}当您需要包含核心设备名称的授权策略（例如 MQTT 主题或设备影子）时，可以使用此配方变量为一组核心设备配置单个授权策略。例如，您可以允许组件访问以下资源以进行影子 IPC 操作。

```
$aws/things/{iot:thingName}/shadow/
```

授权策略中的特殊字符

要在授权策略中指定文字*或?字符，必须使用转义序列。以下转义序列指示 AWS IoT Greengrass Core 软件使用字面值而不是字符的特殊含义。例如，该*字符是与任意字符组合匹配的通配符。

字面字符	转义序列	注意事项
*	${*}	
?	${?}	AWS IoT Greengrass目前不支持与任何单个字符匹配的?通配符。
$	${$}	使用此转义序列来匹配包含的资源${。例如，要匹配名为的资源${resourceName} ，必须指定${$}{resourceName} 。否则，要匹配包含的资源$，您可以使用文字$，例如允许访问以开头$aws的主题。

授权策略示例

您可以参考以下授权策略示例，以帮助您为组件配置授权策略。

Example 带有授权策略的组件配方示例

以下示例组件配方包括一个定义授权策略的accessControl对象。此策略授权该com.example.HelloWorld组件发布到该test/topic主题。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
```

```
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/HelloWorld.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        "com.example.HelloWorld:pubsub:1":
          policyDescription: Allows access to publish to test/topic.
          operations:
            - "aws.greengrass#PublishToTopic"
          resources:
            - "test/topic"
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/HelloWorld.jar
```

Example 使用授权策略更新组件配置示例

以下部署中的示例配置更新指定使用定义授权策略的accessControl对象来配置组件。此策略授权该com.example.HelloWorld组件发布到该test/topic主题。

Console

要合并的配置

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.HelloWorld:pubsub:1": {
        "policyDescription": "Allows access to publish to test/topic.",
        "operations": [
          "aws.greengrass#PublishToTopic"
        ],
        "resources": [
          "test/topic"
        ]
      }
    }
  }
}
```

AWS CLI

以下命令创建对核心设备的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://hello-world-
deployment.json
```

该hello-world-deployment.json文件包含以下 JSON 文档。

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
```

```
        "merge": "{\"accessControl\":{\"aws.greengrass.ipc.pubsub\":
{\"com.example.HelloWorld:pubsub:1\":{\"policyDescription\":\"Allows access to
 publish to test/topic.\",\"operations\":[\"aws.greengrass#PublishToTopic\"],
\"resources\":[\"test/topic\"]}}}}}"
      }
    }
  }
}
```

Greengrass CLI

以下 Greengrass CLI 命令在核心设备上创建本地部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config hello-world-configuration.json
```

该hello-world-configuration.json文件包含以下 JSON 文档。

```
{
  "com.example.HelloWorld": {
    "MERGE": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  }
}
```

订阅 IPC 事件直播

您可以使用 IPC 操作在 Greengrass 核心设备上订阅事件流。要使用订阅操作，请定义订阅处理程序并创建对 IPC 服务的请求。然后，每次核心设备将事件消息流式传输到您的组件时，IPC 客户端都会运行订阅处理程序的函数。

您可以关闭订阅以停止处理事件消息。为此，请在用于打开订阅的订阅操作对象上调用 `closeStream()` `Close()` (Java)、`close()` (Python) 或 (C++)。

C AWS IoT Greengrass ore IPC 服务支持以下订阅操作：

- SubscribeToTopic
- SubscribeToIotCore
- SubscribeToComponentUpdates
- SubscribeToConfigurationUpdate
- SubscribeToValidateConfigurationUpdates

主题

- 定义订阅处理程序
- 订阅处理程序示例

定义订阅处理程序

要定义订阅处理程序，请定义处理事件消息、错误和流关闭的回调函数。如果使用 IPC 客户端 V1，则必须在类中定义这些函数。如果您使用 IPC 客户端 V2（在 Java 和 Python 软件开发工具包的更高版本中可用），则无需创建订阅处理程序类即可定义这些函数。

Java

如果您使用 IPC 客户端 V1，则必须实现通用`software.amazon.awssdk.eventstreamrpc.StreamResponseHandler<StreamEventType>`接口。*StreamEventType*是订阅操作的事件消息的类型。定义以下函数来处理事件消息、错误和流关闭。

如果您使用 IPC 客户端 V2，则可以在订阅处理程序类之外定义这些函数或使用 lambda 表达式。

`void onStreamEvent(StreamEventType event)`

IPC 客户端在收到事件消息（例如 MQTT 消息或组件更新通知）时调用的回调。

`boolean onStreamError(Throwable error)`

IPC 客户端在直播错误发生时调用的回调。

如果出现错误，则返回 true 则关闭订阅流，或返回 false 以保持直播的打开状态。

`void onStreamClosed()`

IPC 客户端在直播关闭时调用的回调。

Python

如果您使用 IPC 客户端 V1，则必须扩展与订阅操作相对应的流响应处理程序类。AWS IoT Device SDK包括每个订阅操作的订阅处理程序类。*StreamEventType*是订阅操作的事件消息的类型。定义以下函数来处理事件消息、错误和流关闭。

如果您使用 IPC 客户端 V2，则可以在订阅处理程序类之外定义这些函数或使用 lambda 表达式。

`def on_stream_event(self, event: StreamEventType) -> None`

IPC 客户端在收到事件消息（例如 MQTT 消息或组件更新通知）时调用的回调。

`def on_stream_error(self, error: Exception) -> bool`

IPC 客户端在直播错误发生时调用的回调。

如果出现错误，则返回 true 则关闭订阅流，或返回 false 以保持直播的打开状态。

`def on_stream_closed(self) -> None`

IPC 客户端在直播关闭时调用的回调。

C++

实现一个从与订阅操作对应的流响应处理程序类派生的类。AWS IoT Device SDK包括每个订阅操作的订阅处理程序基类。*StreamEventType*是订阅操作的事件消息的类型。定义以下函数来处理事件消息、错误和流关闭。

`void OnStreamEvent(StreamEventType *event)`

IPC 客户端在收到事件消息（例如 MQTT 消息或组件更新通知）时调用的回调。

`bool OnStreamError(OperationError *error)`

IPC 客户端在直播错误发生时调用的回调。

如果出现错误，则返回 true 则关闭订阅流，或返回 false 以保持直播的打开状态。

`void OnStreamClosed()`

IPC 客户端在直播关闭时调用的回调。

JavaScript

实现一个从与订阅操作对应的流响应处理程序类派生的类。AWS IoT Device SDK包括每个订阅操作的订阅处理程序基类。*StreamEventType*是订阅操作的事件消息的类型。定义以下函数来处理事件消息、错误和流关闭。

`on(event: 'ended', listener: StreamingOperationEndedListener)`

IPC 客户端在直播关闭时调用的回调。

`on(event: 'streamError', listener: StreamingRpcErrorListener)`

IPC 客户端在直播错误发生时调用的回调。

如果出现错误，则返回 true 则关闭订阅流，或返回 false 以保持直播的打开状态。

`on(event: 'message', listener: (message: InboundMessageType) => void)`

IPC 客户端在收到事件消息（例如 MQTT 消息或组件更新通知）时调用的回调。

订阅处理程序示例

以下示例演示如何使用SubscribeToTopic操作和订阅处理程序来订阅本地发布/订阅消息。

Java (IPC client V2)

Example 示例：订阅本地发布/订阅消息

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;
```

```
import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }
```

```
    public static void onStreamEvent(SubscriptionResponseMessage
 subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
 subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
 StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
 message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
 response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
```

Python (IPC client V2)

Example 示例：订阅本地发布/订阅消息

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)
```

```
def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
```

```
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


if __name__ == '__main__':
    main()
```

C++

Example 示例：订阅本地发布/订阅消息

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
```

```
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
```

```
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example 示例：订阅本地发布/订阅消息

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options

            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })
```

```
            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

IPC 最佳实践

在自定义组件中使用 IPC 的最佳实践在 IPC 客户端 V1 和 IPC 客户端 V2 之间有所不同。请遵循您所使用的 IPC 客户端版本的最佳实践。

IPC client V2

IPC 客户端 V2 在单独的线程中运行回调函数，因此与 IPC 客户端 V1 相比，在使用 IPC 和编写订阅处理函数时，需要遵循的准则较少。

- 重复使用一个 IPC 客户端

  创建 IPC 客户端后，请将其保持打开状态并重复用于所有 IPC 操作。创建多个客户端会消耗额外的资源，并可能导致资源泄漏。

- 处理异常

  IPC 客户端 V2 在订阅处理函数中记录未捕获的异常。您应该在处理函数中捕获异常，以处理代码中发生的错误。

IPC client V1

IPC 客户端 V1 使用与 IPC 服务器通信并调用订阅处理程序的单线程。在编写订阅处理函数时，必须考虑这种同步行为。

- 重复使用一个 IPC 客户端

  创建 IPC 客户端后，请将其保持打开状态并重复用于所有 IPC 操作。创建多个客户端会消耗额外的资源，并可能导致资源泄漏。

- 异步运行阻塞代码

  当线程被阻塞时，IPC 客户端 V1 无法发送新请求或处理新的事件消息。你应该在从处理函数运行的单独线程中运行阻塞代码。阻塞代码包括`sleep`调用、持续运行的循环以及需要一段时间才能完成的同步 I/O 请求。

- 异步发送新的 IPC 请求

  IPC 客户端 V1 无法从订阅处理函数中发送新请求，因为如果您等待响应，该请求会阻塞处理器函数。您应该在从处理程序函数运行的单独线程中发送 IPC 请求。

- 处理异常

  IPC 客户端 V1 不处理订阅处理函数中未捕获的异常。如果您的处理函数抛出异常，则订阅将关闭，并且该异常不会出现在您的组件日志中。您应该在处理程序函数中捕获异常，以保持订阅处于打开状态，并记录代码中发生的错误。

发布/订阅本地消息

发布/订阅 (pubsub) 消息使您能够向主题发送和接收消息。组件可以向主题发布消息，将消息发送给其他组件。然后，订阅该主题的组件可以对它们收到的消息进行操作。

Note

您不能使用此发布/订阅 IPC 服务来发布或订阅 MQTT。AWS IoT Core有关如何使用 AWS IoT Core MQTT 交换消息的更多信息，请参阅发布/订阅 AWS IoT Core MQTT 消息。

主题

- SDK 的最低版本
- 授权
- PublishToTopic
- SubscribeToTopic
- 示例

SDK 的最低版本

下表列出了在向本地主题发布和订阅消息时必须使用的最低版本。AWS IoT Device SDK

SDK	最低版本	
AWS IoT Device SDK适用于 Java v2	v1.2.10	
AWS IoT Device SDK适用于 Python v2	v1.5.3	
AWS IoT Device SDK适用于 C ++ v2	v1.17.0	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

授权

要在自定义组件中使用本地发布/订阅消息，必须定义允许您的组件向主题发送和接收消息的授权策略。有关定义授权策略的信息，请参阅授权组件执行 IPC 操作。

发布/订阅消息的授权策略具有以下属性。

IPC 服务标识符：`aws.greengrass.ipc.pubsub`

操作	描述	资源
`aws.greengrass#PublishToTopic`	允许组件向您指定的主题发布消息。	主题字符串，例如`test/topic` 。*使用匹配主题中的任意字符组合。 此主题字符串不支持 MQTT 主题通配符（#和）。+
`aws.greengrass#SubscribeToTopic`	允许组件订阅您指定的主题的消息。	主题字符串，例如`test/topic` 。*使用匹配主题中的任意字符组合。 在 Greengrass nucleus v2.6.0 及更高版本中，你可以订阅包含 MQTT 主题通配符（和）的主题。# +此主题字符串支持 MQTT 主题通配符作为文字字符。例如，如果组件的授权策略授予访问权限`test/topic/#` ，则该组件可以订阅`test/topic/#` ，但无法订阅`test/topic/filter` 。
`*`	允许组件发布和订阅您指定的主题的消息。	主题字符串，例如`test/topic` 。*使用匹配主题中的任意字符组合。 在 Greengrass nucleus v2.6.0 及更高版本中，你可以订阅包

操作	描述	资源
		含 MQTT 主题通配符（和）的主题。# +此主题字符串支持 MQTT 主题通配符作为文字字符。例如，如果组件的授权策略授予访问权限test/topic/# ，则该组件可以订阅 test/topic/# ，但无法订阅test/topic/filter 。

授权策略示例

您可以参考以下授权策略示例，以帮助您为组件配置授权策略。

Example 授权策略示例

以下示例授权策略允许组件发布和订阅所有主题。

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyLocalPubSubComponent:pubsub:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToTopic",
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

PublishToTopic

向主题发布消息。

请求

此操作的请求具有以下参数：

`topic`

要向其发布消息的主题。

`publishMessage`（Python:`publish_message`）

要发布的消息。此对象包含以下信息。`PublishMessage`必须指定`jsonMessage`和之一`binaryMessage`。

`jsonMessage`（Python:`json_message`）

（可选）一条 JSON 消息。此对象包含以下信息：`JsonMessage`

`message`

作为对象的 JSON 消息。

`context`

消息的上下文，例如发布消息的主题。

此功能适用于 Greengrass nucleus 组件的 2.6.0 及更高版本。下表列出了访问消息上下文时必须使用的最低版本。AWS IoT Device SDK

SDK	最低版本	
AWS IoT Device SDK适用于 Java v2	v1.9.3	
AWS IoT Device SDK适用于 Python v2	v1.11.3	
AWS IoT Device SDK适用于 C++ v2	v1.18.4	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

Note

C AWS IoT Greengrass ore 软件在`PublishToTopic`和`SubscribeToTopic`操作中使用相同的消息对象。当您订阅时，AWS IoT GreengrassCore 软件会在消息中设置此上下文对象，并在您发布的消息中忽略此上下文对象。

此对象包含以下信息：`MessageContext`

`topic`

发布消息的主题。

`binaryMessage`（Python:`binary_message`）

（可选）二进制消息。此对象包含以下信息：`BinaryMessage`

`message`

以 blob 形式呈现的二进制消息。

`context`

消息的上下文，例如发布消息的主题。

此功能适用于 Greengrass nucleus 组件的 2.6.0 及更高版本。下表列出了访问消息上下文时必须使用的最低版本。AWS IoT Device SDK

SDK	最低版本
AWS IoT Device SDK适用于 Java v2	v1.9.3
AWS IoT Device SDK适用于 Python v2	v1.11.3
AWS IoT Device SDK适用于 C++ v2	v1.18.4
AWS IoT Device SDK for JavaScript v2	v1.12.0

Note

C AWS IoT Greengrass ore 软件在PublishToTopic和SubscribeToTopic操作中使用相同的消息对象。当您订阅时，AWS IoT GreengrassCore 软件会在消息中设置此上下文对象，并在您发布的消息中忽略此上下文对象。

此对象包含以下信息：`MessageContext`

`topic`

发布消息的主题。

响应

此操作在其响应中未提供任何信息。

示例

以下示例演示了如何在自定义组件代码中调用此操作。

Java (IPC client V2)

Example 示例：发布二进制消息

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.BinaryMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;

import java.nio.charset.StandardCharsets;

public class PublishToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
```

```
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            PublishToTopicV2.publishBinaryMessageToTopic(ipcClient, topic,
 message);
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static PublishToTopicResponse publishBinaryMessageToTopic(
            GreengrassCoreIPCClientV2 ipcClient, String topic, String message)
 throws InterruptedException {
        BinaryMessage binaryMessage =
                new
 BinaryMessage().withMessage(message.getBytes(StandardCharsets.UTF_8));
        PublishMessage publishMessage = new
 PublishMessage().withBinaryMessage(binaryMessage);
        PublishToTopicRequest publishToTopicRequest =
                new
 PublishToTopicRequest().withTopic(topic).withPublishMessage(publishMessage);
        return ipcClient.publishToTopic(publishToTopicRequest);
    }
}
```

Python (IPC client V2)

Example 示例：发布二进制消息

```
import sys
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    PublishMessage,
    BinaryMessage
)
```

```
def main():
    args = sys.argv[1:]
    topic = args[0]
    message = args[1]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        publish_binary_message_to_topic(ipc_client, topic, message)
        print('Successfully published to topic: ' + topic)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def publish_binary_message_to_topic(ipc_client, topic, message):
    binary_message = BinaryMessage(message=bytes(message, 'utf-8'))
    publish_message = PublishMessage(binary_message=binary_message)
    return ipc_client.publish_to_topic(topic=topic,
 publish_message=publish_message)


if __name__ == '__main__':
    main()
```

C++

Example 示例：发布二进制消息

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }
```

```
    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    String message("Hello, World!");
    int timeout = 10;

    PublishToTopicRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    BinaryMessage binaryMessage;
    binaryMessage.SetMessage(messageData);
    PublishMessage publishMessage;
    publishMessage.SetBinaryMessage(binaryMessage);
    request.SetTopic(topic);
    request.SetPublishMessage(publishMessage);

    auto operation = ipcClient.NewPublishToTopic();
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
```

```
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }
    return 0;
}
```

JavaScript

Example 示例：发布二进制消息

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {BinaryMessage, PublishMessage, PublishToTopicRequest} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";

class PublishToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;
    private readonly messageString : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.messageString = "<define_your_message_string>";
        this.publishToTopic().then(r => console.log("Started workflow"));
    }

    private async publishToTopic() {
        try {
            this.ipcClient = await getIpcClient();
```

```
            const binaryMessage : BinaryMessage = {
                message: this.messageString
            }

            const publishMessage : PublishMessage = {
                binaryMessage: binaryMessage
            }

            const request : PublishToTopicRequest = {
                topic: this.topic,
                publishMessage: publishMessage
            }

            this.ipcClient.publishToTopic(request).finally(() =>
 console.log(`Published message ${publishMessage.binaryMessage?.message} to topic`))

        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToTopic = new PublishToTopic();
```

SubscribeToTopic

订阅有关某个主题的消息。

此操作是一种订阅操作，您可以在其中订阅事件消息流。要使用此操作，请定义一个流响应处理程序，其中包含处理事件消息、错误和流关闭的函数。有关更多信息，请参阅 订阅 IPC 事件直播。

事件消息类型：`SubscriptionResponseMessage`

请求

此操作的请求具有以下参数：

`topic`

要订阅的主题。

Note

在 Greengrass nucleus v2.6.0 及更高版本中，本主题支持 MQTT 主题通配符（和）。# +

`receiveMode`（Python:`receive_mode`）

（可选）指定组件是否从自身接收消息的行为。您可以更改此行为以允许组件根据自己的消息进行操作。默认行为取决于主题是否包含 MQTT 通配符。从以下选项中进行选择：

- `RECEIVE_ALL_MESSAGES`— 接收与该主题匹配的所有消息，包括来自订阅组件的消息。

  当您订阅不包含 MQTT 通配符的主题时，此模式是默认选项。
- `RECEIVE_MESSAGES_FROM_OTHERS`— 接收与该主题匹配的所有消息，但来自订阅组件的消息除外。

  当您订阅包含 MQTT 通配符的主题时，此模式是默认选项。

此功能适用于 Greengrass nucleus 组件的 2.6.0 及更高版本。下表列出了在设置接收模式时AWS IoT Device SDK必须使用的最低版本。

SDK	最低版本	
AWS IoT Device SDK适用于 Java v2	v1.9.3	

SDK	最低版本
AWS IoT Device SDK适用于 Python v2	v1.11.3
AWS IoT Device SDK适用于 C++ v2	v1.18.4
AWS IoT Device SDK适用于 JavaScript v2	v1.12.0

响应

此操作的响应包含以下信息：

`messages`

消息流。此对象包含以下信息。`SubscriptionResponseMessage`每条消息都包含`jsonMessage`或`binaryMessage`。

`jsonMessage`（Python:`json_message`）

（可选）一条 JSON 消息。此对象包含以下信息：`JsonMessage`

`message`

作为对象的 JSON 消息。

`context`

消息的上下文，例如发布消息的主题。

此功能适用于 Greengrass nucleus 组件的 2.6.0 及更高版本。下表列出了访问消息上下文时必须使用的最低版本。AWS IoT Device SDK

SDK	最低版本
AWS IoT Device SDK适用于 Java v2	v1.9.3
AWS IoT Device SDK适用于 Python v2	v1.11.3

SDK	最低版本	
AWS IoT Device SDK适用于 C++ v2	v1.18.4	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

> **Note**
>
> C AWS IoT Greengrass ore 软件在`PublishToTopic`和`SubscribeToTopic`操作中使用相同的消息对象。当您订阅时，AWS IoT GreengrassCore 软件会在消息中设置此上下文对象，并在您发布的消息中忽略此上下文对象。

此对象包含以下信息：`MessageContext`

`topic`

发布消息的主题。

`binaryMessage`（Python:`binary_message`）

（可选）二进制消息。此对象包含以下信息：`BinaryMessage`

`message`

以 blob 形式呈现的二进制消息。

`context`

消息的上下文，例如发布消息的主题。

此功能适用于 Greengrass nucleus 组件的 2.6.0 及更高版本。下表列出了访问消息上下文时必须使用的最低版本。AWS IoT Device SDK

SDK	最低版本	
AWS IoT Device SDK适用于 Java v2	v1.9.3	

SDK	最低版本
AWS IoT Device SDK适用于 Python v2	v1.11.3
AWS IoT Device SDK适用于 C++ v2	v1.18.4
AWS IoT Device SDK for JavaScript v2	v1.12.0

Note

C AWS IoT Greengrass ore 软件在PublishToTopic和SubscribeToTopic操作中使用相同的消息对象。当您订阅时，AWS IoT GreengrassCore 软件会在消息中设置此上下文对象，并在您发布的消息中忽略此上下文对象。

此对象包含以下信息：MessageContext

topic

发布消息的主题。

topicName（Python:topic_name）

消息发布到的主题。

Note

目前未使用此属性。在 Greengrass nucleus v2.6.0 及更高版本中，你可以从中获取值以获取(jsonMessage|binaryMessage).context.topic消息发布的主题SubscriptionResponseMessage。

示例

以下示例演示了如何在自定义组件代码中调用此操作。

Java (IPC client V2)

Example 示例：订阅本地发布/订阅消息

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
```

```
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static void onStreamEvent(SubscriptionResponseMessage
 subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
 subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
 StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
 message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
 response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
```

Python (IPC client V2)

Example 示例：订阅本地发布/订阅消息

```
import sys
```

```
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)
```

```
def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


if __name__ == '__main__':
    main()
```

C++

Example 示例：订阅本地发布/订阅消息

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
```

```
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
```

```
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }
```

```
    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example 示例：订阅本地发布/订阅消息

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options

            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
```

```
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

示例

使用以下示例来学习如何在组件中使用发布/订阅 IPC 服务。

发布/订阅发布者示例（Java、IPC 客户端 V1）

以下示例配方允许该组件发布到所有主题。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherJava:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/PubSubPublisher.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubPublisherJava:pubsub:1':
          policyDescription: Allows access to publish to all topics.
          operations:
            - 'aws.greengrass#PublishToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/PubSubPublisher.jar
```

以下示例 Java 应用程序演示如何使用发布/订阅 IPC 服务将消息发布到其他组件。

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubPublisher {
```

```
    public static void main(String[] args) {
        String message = "Hello from the pub/sub publisher (Java).";
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
GreengrassCoreIPCClient(eventStreamRPCConnection);

            while (true) {
                PublishToTopicRequest publishRequest = new PublishToTopicRequest();
                PublishMessage publishMessage = new PublishMessage();
                BinaryMessage binaryMessage = new BinaryMessage();
                binaryMessage.setMessage(message.getBytes(StandardCharsets.UTF_8));
                publishMessage.setBinaryMessage(binaryMessage);
                publishRequest.setPublishMessage(publishMessage);
                publishRequest.setTopic(topic);
                CompletableFuture<PublishToTopicResponse> futureResponse = ipcClient
                        .publishToTopic(publishRequest,
Optional.empty()).getResponse();

                try {
                    futureResponse.get(10, TimeUnit.SECONDS);
                    System.out.println("Successfully published to topic: " + topic);
                } catch (TimeoutException e) {
                    System.err.println("Timeout occurred while publishing to topic: " +
topic);
                } catch (ExecutionException e) {
                    if (e.getCause() instanceof UnauthorizedError) {
                        System.err.println("Unauthorized error while publishing to
topic: " + topic);
                    } else {
                        System.err.println("Execution exception while publishing to
topic: " + topic);
                    }
                    throw e;
                }
                Thread.sleep(5000);
            }
        } catch (InterruptedException e) {
            System.out.println("Publisher interrupted.");
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
```

```
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

发布/订阅订阅者示例（Java、IPC 客户端 V1）

以下示例配方允许该组件订阅所有主题。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberJava:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/PubSubSubscriber.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubSubscriberJava:pubsub:1':
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - 'aws.greengrass#SubscribeToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/PubSubSubscriber.jar
```

以下示例 Java 应用程序演示如何使用发布/订阅 IPC 服务将消息订阅到其他组件。

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.SubscriptionResponseMessage;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
```

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubSubscriber {

    public static void main(String[] args) {
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            SubscribeToTopicRequest subscribeRequest = new SubscribeToTopicRequest();
            subscribeRequest.setTopic(topic);
            SubscribeToTopicResponseHandler operationResponseHandler = ipcClient
                    .subscribeToTopic(subscribeRequest, Optional.of(new
 SubscribeResponseHandler()));
            CompletableFuture<SubscribeToTopicResponse> futureResponse =
 operationResponseHandler.getResponse();

            try {
                futureResponse.get(10, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
 topic);
                throw e;
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to topic:
 " + topic);
                } else {
                    System.err.println("Execution exception while subscribing to topic:
 " + topic);
                }
                throw e;
            }

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
```

```
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    private static class SubscribeResponseHandler implements
 StreamResponseHandler<SubscriptionResponseMessage> {

        @Override
        public void onStreamEvent(SubscriptionResponseMessage
 subscriptionResponseMessage) {
            try {
                String message = new
 String(subscriptionResponseMessage.getBinaryMessage()
                        .getMessage(), StandardCharsets.UTF_8);
                System.out.println("Received new message: " + message);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        @Override
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false; // Return true to close stream, false to keep stream open.
        }

        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to topic stream closed.");
        }
    }
}
```

发布/订阅发布者示例（Python、IPC 客户端 V1）

以下示例配方允许该组件发布到所有主题。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherPython:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/pubsub_publisher.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/pubsub_publisher.py"
      }
    }
```

```
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherPython
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherPython:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/pubsub_publisher.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/pubsub_publisher.py
```

以下示例 Python 应用程序演示了如何使用发布/订阅 IPC 服务将消息发布到其他组件。

```
import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
```

```
    PublishToTopicRequest,
    PublishMessage,
    BinaryMessage,
    UnauthorizedError
)


topic = "test/topic/python"
message = "Hello from the pub/sub publisher (Python)."
TIMEOUT = 10

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    while True:
        request = PublishToTopicRequest()
        request.topic = topic
        publish_message = PublishMessage()
        publish_message.binary_message = BinaryMessage()
        publish_message.binary_message.message = bytes(message, "utf-8")
        request.publish_message = publish_message
        operation = ipc_client.new_publish_to_topic()
        operation.activate(request)
        future_response = operation.get_response()

        try:
            future_response.result(TIMEOUT)
            print('Successfully published to topic: ' + topic)
        except concurrent.futures.TimeoutError:
            print('Timeout occurred while publishing to topic: ' + topic,
 file=sys.stderr)
        except UnauthorizedError as e:
            print('Unauthorized error while publishing to topic: ' + topic,
 file=sys.stderr)
            raise e
        except Exception as e:
            print('Exception while publishing to topic: ' + topic, file=sys.stderr)
            raise e
        time.sleep(5)
except InterruptedError:
    print('Publisher interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
```

```
    exit(1)
```

发布/订阅订阅者示例（Python、IPC 客户端 V1）

以下示例配方允许该组件订阅所有主题。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberPython:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/pubsub_subscriber.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
```

```
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/pubsub_subscriber.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberPython
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberPython:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/pubsub_subscriber.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/pubsub_subscriber.py
```

以下示例 Python 应用程序演示了如何使用发布/订阅 IPC 服务来订阅其他组件的消息。

```
import concurrent.futures
```

```
import sys
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    SubscribeToTopicRequest,
    SubscriptionResponseMessage,
    UnauthorizedError
)

topic = "test/topic/python"
TIMEOUT = 10


class StreamHandler(client.SubscribeToTopicStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: SubscriptionResponseMessage) -> None:
        try:
            message = str(event.binary_message.message, "utf-8")
            print("Received new message: " + message)
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        print("Received a stream error.", file=sys.stderr)
        traceback.print_exc()
        return False  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        print('Subscribe to topic stream closed.')


try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = SubscribeToTopicRequest()
    request.topic = topic
    handler = StreamHandler()
    operation = ipc_client.new_subscribe_to_topic(handler)
    operation.activate(request)
```

```
    future_response = operation.get_response()

    try:
        future_response.result(TIMEOUT)
        print('Successfully subscribed to topic: ' + topic)
    except concurrent.futures.TimeoutError as e:
        print('Timeout occurred while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except UnauthorizedError as e:
        print('Unauthorized error while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while subscribing to topic: ' + topic, file=sys.stderr)
        raise e

    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

发布/订阅发布者示例 (C++)

以下示例配方允许该组件发布到所有主题。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
```

```
          "com.example.PubSubPublisherCpp:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pubsub_publisher"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
```

```
        com.example.PubSubPublisherCpp:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_pubsub_publisher"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.PubSubPublisherCpp/1.0.0/
greengrassv2_pubsub_publisher
        Permission:
          Execute: OWNER
```

以下示例 C++ 应用程序演示了如何使用发布/订阅 IPC 服务将消息发布到其他组件。

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
```

```
    String message("Hello from the pub/sub publisher (C++).");
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToTopicRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        BinaryMessage binaryMessage;
        binaryMessage.SetMessage(messageData);
        PublishMessage publishMessage;
        publishMessage.SetBinaryMessage(binaryMessage);
        request.SetTopic(topic);
        request.SetPublishMessage(publishMessage);

        auto operation = ipcClient.NewPublishToTopic();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
```

```
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

发布/订阅订阅者示例 (C++)

以下示例配方允许该组件订阅所有主题。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberCpp:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
```

```
        }
      }
    },
    "Manifests": [
      {
        "Lifecycle": {
          "run": "{artifacts:path}/greengrassv2_pub_sub_subscriber"
        },
        "Artifacts": [
          {
            "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber",
            "Permission": {
              "Execute": "OWNER"
            }
          }
        ]
      }
    ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberCpp:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_pub_sub_subscriber"
    Artifacts:
```

```
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber
        Permission:
          Execute: OWNER
```

以下示例 C++ 应用程序演示了如何使用发布/订阅 IPC 服务来订阅其他组件的消息。

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                std::cout << "Received new message: " << messageString << std::endl;
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    std::cout << "Received new message: " << messageString <<
 std::endl;
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
```

```
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to topic stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }
```

```
    SubscribeToTopicRequest request;
    request.SetTopic(topic);
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

发布/订阅 AWS IoT Core MQTT 消息

AWS IoT Core MQTT 消息 IPC 服务允许您发送和接收 MQTT 消息。 AWS IoT Core组件可以向其他来源发布消息 AWS IoT Core 和订阅主题，以处理来自其他来源的 MQTT 消息。有关 MQTT AWS IoT Core 实现的更多信息，请参阅《AWS IoT Core 开发人员指南》中的 MQTT。

> Note
>
> 此 MQTT 消息 IPC 服务允许您与交换消息。 AWS IoT Core有关如何在组件之间交换消息的更多信息，请参阅发布/订阅本地消息。

主题

- SDK 的最低版本
- 授权
- PublishToIotCore
- SubscribeToIotCore
- 示例

SDK 的最低版本

下表列出了在发布和订阅 MQTT 消息时必须使用的最低版本。 AWS IoT Device SDK AWS IoT Core

SDK	最低版本	
AWS IoT Device SDK 适用于 Java v2	v1.2.10	
AWS IoT Device SDK 适用于 Python v2	v1.5.3	
AWS IoT Device SDK 适用于 C++ v2	v1.17.0	
AWS IoT Device SDK 适用于 JavaScript v2	v1.12.0	

授权

要在自定义组件中使用 AWS IoT Core MQTT 消息传递，您必须定义授权策略，允许您的组件发送和接收有关主题的消息。有关定义授权策略的信息，请参阅授权组件执行 IPC 操作。

AWS IoT Core MQTT 消息传递的授权策略具有以下属性。

IPC 服务标识符：`aws.greengrass.ipc.mqttproxy`

操作	描述	资源
`aws.greengrass#PublishToIoTCore`	允许组件在您指定的 MQTT 主题 AWS IoT Core 上向其发布消息。	一个主题字符串，例如或`test/topic`，`*`用于允许访问所有主题。您可以使用 MQTT 主题通配符（`#`和`+`）来匹配多个资源。
`aws.greengrass#SubscribeToIoTCore`	允许组件订阅来自您指定 AWS IoT Core 主题的消息。	一个主题字符串，例如或`test/topic`，`*`用于允许访问所有主题。您可以使用 MQTT 主题通配符（`#`和`+`）来匹配多个资源。
`*`	允许组件发布和订阅您指定的主题的 AWS IoT Core MQTT 消息。	一个主题字符串，例如或`test/topic`，`*`用于允许访问所有主题。您可以使用 MQTT 主题通配符（`#`和`+`）来匹配多个资源。

MQTT 授权策略中的 M AWS IoT Core QTT 通配符

您可以在 MQTT IPC 授权策略中使用 AWS IoT Core MQTT 通配符。组件可以发布和订阅与您在授权策略中允许的主题筛选条件相匹配的主题。例如，如果组件的授权策略授予访问权限`test/topic/#`，则该组件可以订阅`test/topic/#`，也可以发布和订阅`test/topic/filter`。

AWS IoT Core MQTT 授权策略中的配方变量

如果您使用 Greengrass 核心的 v2.6.0 或更高版本，则可以在授权策略中使用配方变量。{iot:thingName}此功能使您可以为一组核心设备配置单一授权策略，其中每台核心设备只能访问包含自己名称的主题。例如，您可以允许组件访问以下主题资源。

```
devices/{iot:thingName}/messages
```

有关更多信息，请参阅 食谱变量 和 在合并更新中使用配方变量。

授权策略示例

您可以参考以下授权策略示例，以帮助您为组件配置授权策略。

Example 具有无限制访问权限的授权策略示例

以下示例授权策略允许组件发布和订阅所有主题。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    com.example.MyIoTCorePubSubComponent:mqttproxy:1:
```

```
      policyDescription: Allows access to publish/subscribe to all topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - "*"
```

Example 限制访问权限的授权策略示例

以下示例授权策略允许组件发布和订阅名为`factory/1/events`和的两个主题`factory/1/actions`。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to factory 1
 topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/actions",
          "factory/1/events"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to factory 1 topics.
      operations:
        - aws.greengrass#PublishToIoTCore
```

```
          - aws.greengrass#SubscribeToIoTCore
        resources:
          - factory/1/actions
          - factory/1/events
```

Example 一组核心设备的授权策略示例

Important

此示例使用了 Greengrass nucleus 组件的 v2.6.0 及更高版本中可用的功能。Greengrass nucleus v2.6.0 增加了对大多数配方变量的支持，例如组件配置中的变量。{iot:thingName}

以下示例授权策略允许组件发布和订阅包含运行该组件的核心设备名称的主题。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/devices/{iot:thingName}/controls"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
```

```
        policyDescription: Allows access to publish/subscribe to all topics.
        operations:
          - aws.greengrass#PublishToIoTCore
          - aws.greengrass#SubscribeToIoTCore
        resources:
          - factory/1/devices/{iot:thingName}/controls
```

PublishToIotCore

向 AWS IoT Core 发布有关某个主题的 MQTT 消息。

当您向发布 MQTT 消息时 AWS IoT Core，有每秒 100 个交易的配额。如果超过此配额，则消息将排队等候在 Greengrass 设备上处理。还有每秒 512 Kb 的数据配额，整个账户的配额为每秒 20,000 次发布（有些 AWS 区域有 2,000 次）。有关 MQTT 消息代理限制的更多信息 AWS IoT Core，请参阅AWS IoT Core 消息代理和协议限制和配额。

如果您超过这些配额，Greengrass 设备会将发布消息限制为。 AWS IoT Core消息存储在内存中的后台处理程序中。默认情况下，分配给后台处理程序的内存为 2.5 Mb。如果后台处理程序已满，则新消息将被拒绝。您可以增加后台处理程序的大小。有关更多信息，请参阅 Greengrass 核文档中的 配置。为避免填满后台处理程序并需要增加分配的内存，请将发布请求限制为每秒不超过 100 个请求。

当您的应用程序需要以更高的速率或更大的速率发送消息时，可以考虑使用向 Kinesis Data Streams 发送消息。流管理器流管理器组件旨在将大量数据传输到。 AWS Cloud有关更多信息，请参阅 管理 Greengrass 核心设备上的数据流。

请求

此操作的请求具有以下参数：

`topicName`（Python:`topic_name`）

要向其发布消息的主题。

`qos`

要使用的 MQTT 服务质量。这个枚举QOS具有以下值：

- `AT_MOST_ONCE`— QoS 0。MQTT 消息最多只能传送一次。
- `AT_LEAST_ONCE`— QoS 1. MQTT 消息至少传送一次。

`payload`

（可选）以 blob 形式显示的消息负载。

使用 MQTT 5 Greengrass 核 时，以下功能可用于 v2.10.0 及更高版本。当您使用 MQTT 3.1.1 时，这些功能将被忽略。下表列出了访问这些功能必须使用的 AWS IoT 设备 SDK 的最低版本。

SDK	最低版本
AWS IoT Device SDK for Python v2	v1.15.0
AWS IoT Device SDK for Java v2	v1.13.0
AWS IoT Device SDK for C++ v2	v1.24.0
AWS IoT Device SDK for JavaScript v2	v1.13.0

`payloadFormat`

（可选）消息负载的格式。如果未设置`payloadFormat`，则假定类型为`BYTES`。枚举具有以下值：

- `BYTES`— 有效载荷的内容是二进制 blob。
- `UTF8`— 有效载荷的内容是一个 UTF8 字符串。

`retain`

（可选）指示是否在发布`true`时将 MQTT 保留选项设置为。

`userProperties`

（可选）要发送的应用程序特定`UserProperty`对象的列表。该`UserProperty`对象的定义如下：

```
UserProperty:
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

（可选）消息过期并被服务器删除之前的秒数。如果未设置此值，则消息不会过期。

`correlationData`

（可选）添加到请求中的信息，可用于将请求与响应关联起来。

`responseTopic`

（可选）应用于响应消息的主题。

`contentType`

（可选）消息内容类型的应用程序特定标识符。

响应

此操作在其响应中未提供任何信息。

示例

以下示例演示了如何在自定义组件代码中调用此操作。

Java (IPC client V2)

Example 示例：发布消息

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import java.nio.charset.StandardCharsets;

public class PublishToIoTCore {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
 GreengrassCoreIPCClientV2.builder().build()) {
            ipcClientV2.publishToIoTCore(new PublishToIoTCoreRequest()
                    .withTopicName(topic)
                    .withPayload(message.getBytes(StandardCharsets.UTF_8))
                    .withQos(qos));
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
```

```
}
```

Python (IPC client V2)

Example 示例：发布消息

Note

此示例假设您使用的是 Python v2 版本的 1.5.4 或更高版本。 AWS IoT Device SDK

```
import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'
payload = 'Hello, World'

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp = ipc_client.publish_to_iot_core(topic_name=topic, qos=qos, payload=payload)
ipc_client.close()
```

Java (IPC client V1)

Example 示例：发布消息

Note

此示例使用一个IPCUtils类来创建与 C AWS IoT Greengrass ore IPC 服务的连接。有关更多信息，请参阅 Connect 到 C AWS IoT Greengrass ore IPC 服务。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.PublishToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreResponse;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
```

```
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PublishToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            PublishToIoTCoreResponseHandler responseHandler =
                    PublishToIoTCore.publishBinaryMessageToTopic(ipcClient, topic,
 message, qos);
            CompletableFuture<PublishToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully published to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while publishing to topic: " +
 topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
```

```
        }
    }

    public static PublishToIoTCoreResponseHandler
 publishBinaryMessageToTopic(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 topic, String message, QOS qos) {
        PublishToIoTCoreRequest publishToIoTCoreRequest = new
 PublishToIoTCoreRequest();
        publishToIoTCoreRequest.setTopicName(topic);

 publishToIoTCoreRequest.setPayload(message.getBytes(StandardCharsets.UTF_8));
        publishToIoTCoreRequest.setQos(qos);
        return greengrassCoreIPCClient.publishToIoTCore(publishToIoTCoreRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example 示例：发布消息

> **Note**
>
> 此示例假设您使用的是 Python v2 版本的 1.5.4 或更高版本。 AWS IoT Device SDK

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    QOS,
    PublishToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

topic = "my/topic"
message = "Hello, World"
qos = QOS.AT_LEAST_ONCE

request = PublishToIoTCoreRequest()
```

```
request.topic_name = topic
request.payload = bytes(message, "utf-8")
request.qos = qos
operation = ipc_client.new_publish_to_iot_core()
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)
```

C++

Example 示例：发布消息

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
```

```
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String message("Hello, World!");
    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    PublishToIoTCoreRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    request.SetTopicName(topic);
    request.SetPayload(messageData);
    request.SetQos(qos);

    auto operation = ipcClient.NewPublishToIoTCore();
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }

    return 0;
}
```

JavaScript

Example 示例：发布消息

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {QOS, PublishToIoTCoreRequest} from "aws-iot-device-sdk-v2/dist/
greengrasscoreipc/model";

class PublishToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.publishToIoTCore().then(r => console.log("Started workflow"));
    }

    private async publishToIoTCore() {
        try {
            const request: PublishToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            await this.ipcClient.publishToIoTCore(request);
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
```

```
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToIoTCore = new PublishToIoTCore();
```

SubscribeToIotCore

通过主题或主题筛选器订阅 MQTT 消息。 AWS IoT Core 当组件的生命周期结束时，C AWS IoT Greengrass ore 软件会删除订阅。

此操作是一种订阅操作，您可以在其中订阅事件消息流。要使用此操作，请定义一个流响应处理程序，其中包含处理事件消息、错误和流关闭的函数。有关更多信息，请参阅 订阅 IPC 事件直播。

事件消息类型：`IoTCoreMessage`

请求

此操作的请求具有以下参数：

`topicName`（Python:`topic_name`）

要订阅的主题。您可以使用 MQTT 主题通配符（#和+）来订阅多个主题。

`qos`

要使用的 MQTT 服务质量。这个枚举`QOS`具有以下值：

- `AT_MOST_ONCE`— QoS 0。MQTT 消息最多只能传送一次。
- `AT_LEAST_ONCE`— QoS 1. MQTT 消息至少传送一次。

响应

此操作的响应包含以下信息：

`messages`

MQTT 消息流。此对象包含以下信息：`IoTCoreMessage`

`message`

MQTT 消息。此对象包含以下信息：`MQTTMessage`

`topicName`（Python:`topic_name`）

消息发布到的主题。

`payload`

（可选）以 blob 形式显示的消息负载。

使用 MQTT 5 Greengrass 核 时，以下功能可用于 v2.10.0 及更高版本。当您使用 MQTT 3.1.1 时，这些功能将被忽略。下表列出了访问这些功能必须使用的 AWS IoT 设备 SDK 的最低版本。

SDK	最低版本
AWS IoT Device SDK for Python v2	v1.15.0
AWS IoT Device SDK for Java v2	v1.13.0
AWS IoT Device SDK for C++ v2	v1.24.0
AWS IoT Device SDK for JavaScript v2	v1.13.0

`payloadFormat`

（可选）消息负载的格式。如果未设置`payloadFormat`，则假定类型为`BYTES`。枚举具有以下值：

- `BYTES`— 有效载荷的内容是二进制 blob。
- `UTF8`— 有效载荷的内容是一个 UTF8 字符串。

`retain`

（可选）指示是否在发布`true`时将 MQTT 保留选项设置为。

`userProperties`

（可选）要发送的应用程序特定UserProperty对象的列表。该UserProperty对象的定义如下：

```
UserProperty:
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

（可选）消息过期并被服务器删除之前的秒数。如果未设置此值，则消息不会过期。

`correlationData`

（可选）添加到请求中的信息，可用于将请求与响应关联起来。

`responseTopic`

（可选）应用于响应消息的主题。

`contentType`

（可选）消息内容类型的应用程序特定标识符。

示例

以下示例演示了如何在自定义组件代码中调用此操作。

Java (IPC client V2)

Example 示例：订阅消息

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.IoTCoreMessage;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreResponse;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.function.Consumer;
```

```
import java.util.function.Function;


public class SubscribeToIoTCore {

    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);

        Consumer<IoTCoreMessage> onStreamEvent = ioTCoreMessage ->
                System.out.printf("Received new message on topic %s: %s%n",
                        ioTCoreMessage.getMessage().getTopicName(),
                        new String(ioTCoreMessage.getMessage().getPayload(),
 StandardCharsets.UTF_8));

        Optional<Function<Throwable, Boolean>> onStreamError =
                Optional.of(e -> {
                    System.err.println("Received a stream error.");
                    e.printStackTrace();
                    return false;
                });

        Optional<Runnable> onStreamClosed = Optional.of(() ->
                System.out.println("Subscribe to IoT Core stream closed."));

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToIoTCoreRequest request = new SubscribeToIoTCoreRequest()
                    .withTopicName(topic)
                    .withQos(qos);

            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToIoTCoreResponse,
 SubscribeToIoTCoreResponseHandler>
                    streamingResponse = ipcClientV2.subscribeToIoTCore(request,
 onStreamEvent, onStreamError, onStreamClosed);

            streamingResponse.getResponse();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            while (true) {
                Thread.sleep(10000);
            }
```

```
            // To stop subscribing, close the stream.
            streamingResponse.getHandler().closeStream();
        } catch (InterruptedException e) {
            System.out.println("Subscribe interrupted.");
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Python (IPC client V2)

Example 示例：订阅消息

Note

此示例假设您使用的是 Python v2 版本的 1.5.4 或更高版本。 AWS IoT Device SDK

```
import threading
import traceback

import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'

def on_stream_event(event):
    try:
        topic_name = event.message.topic_name
        message = str(event.message.payload, 'utf-8')
        print(f'Received new message on topic {topic_name}:  {message}')
    except:
        traceback.print_exc()

def on_stream_error(error):
    # Return True to close stream, False to keep stream open.
    return True

def on_stream_closed():
```

```
    pass

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp, operation = ipc_client.subscribe_to_iot_core(
    topic_name=topic,
    qos=qos,
    on_stream_event=on_stream_event,
    on_stream_error=on_stream_error,
    on_stream_closed=on_stream_closed
)

# Keep the main thread alive, or the process will exit.
event = threading.Event()
event.wait()

# To stop subscribing, close the operation stream.
operation.close()
ipc_client.close()
```

Java (IPC client V1)

Example 示例：订阅消息

Note

此示例使用一个IPCUtils类来创建与 C AWS IoT Greengrass ore IPC 服务的连接。有关更多信息，请参阅 Connect 到 C AWS IoT Greengrass ore IPC 服务。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
```

```
import java.util.concurrent.TimeoutException;

public class SubscribeToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            StreamResponseHandler<IoTCoreMessage> streamResponseHandler =
                    new SubscriptionResponseHandler();
            SubscribeToIoTCoreResponseHandler responseHandler =
                    SubscribeToIoTCore.subscribeToIoTCore(ipcClient, topic, qos,
                            streamResponseHandler);
            CompletableFuture<SubscribeToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
 topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to
 topic: " + topic);
                } else {
                    throw e;
                }
            }

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
```

```
            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static SubscribeToIoTCoreResponseHandler
subscribeToIoTCore(GreengrassCoreIPCClient greengrassCoreIPCClient, String topic,
QOS qos, StreamResponseHandler<IoTCoreMessage> streamResponseHandler) {
        SubscribeToIoTCoreRequest subscribeToIoTCoreRequest = new
SubscribeToIoTCoreRequest();
        subscribeToIoTCoreRequest.setTopicName(topic);
        subscribeToIoTCoreRequest.setQos(qos);
        return
greengrassCoreIPCClient.subscribeToIoTCore(subscribeToIoTCoreRequest,
                Optional.of(streamResponseHandler));
    }

    public static class SubscriptionResponseHandler implements
StreamResponseHandler<IoTCoreMessage> {

        @Override
        public void onStreamEvent(IoTCoreMessage ioTCoreMessage) {
            try {
                String topic = ioTCoreMessage.getMessage().getTopicName();
                String message = new
String(ioTCoreMessage.getMessage().getPayload(),
                        StandardCharsets.UTF_8);
                System.out.printf("Received new message on topic %s: %s%n", topic,
message);
            } catch (Exception e) {
                System.err.println("Exception occurred while processing subscription
response " +
                        "message.");
                e.printStackTrace();
            }
        }

        @Override
```

```
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false;
        }

        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to IoT Core stream closed.");
        }
    }
}
```

Python (IPC client V1)

Example 示例：订阅消息

此示例假设您使用的是 Python v2 版本的 1.5.4 或更高版本。 AWS IoT Device SDK

```
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    IoTCoreMessage,
    QOS,
    SubscribeToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

class StreamHandler(client.SubscribeToIoTCoreStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: IoTCoreMessage) -> None:
        try:
```

```
            message = str(event.message.payload, "utf-8")
            topic_name = event.message.topic_name
            # Handle message.
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        # Handle error.
        return True  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        # Handle close.
        pass


topic = "my/topic"
qos = QOS.AT_MOST_ONCE

request = SubscribeToIoTCoreRequest()
request.topic_name = topic
request.qos = qos
handler = StreamHandler()
operation = ipc_client.new_subscribe_to_iot_core(handler)
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)

# Keep the main thread alive, or the process will exit.
while True:
    time.sleep(10)

# To stop subscribing, close the operation stream.
operation.close()
```

C++

Example 示例：订阅消息

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
```

```
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}

    private:
        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string topicName =
 message.value().GetTopicName().value().c_str();
                // Handle message.
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};
```

```
int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
```

```
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example 示例：订阅消息

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {IoTCoreMessage, QOS, SubscribeToIoTCoreRequest} from "aws-iot-device-sdk-v2/
dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToIoTCore().then(r => console.log("Started workflow"));
    }

    private async subscribeToIoTCore() {
        try {
            const request: SubscribeToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            const streamingOperation = this.ipcClient.subscribeToIoTCore(request);
```

```
            streamingOperation.on('message', (message: IoTCoreMessage) => {
                // parse the message depending on your use cases, e.g.
                if (message.message && message.message.payload) {
                    const receivedMessage = message.message.payload.toString();
                }
            });

            streamingOperation.on('streamError', (error : RpcError) => {
                // define your own error handling logic
            });

            streamingOperation.on('ended', () => {
                // define your own logic
            });

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
```

```
const subscribeToIoTCore = new SubscribeToIoTCore();
```

示例

使用以下示例来学习如何在组件中使用 AWS IoT Core MQTT IPC 服务。

示例 AWS IoT Core MQTT 发布器 (C++)

以下示例配方允许该组件发布到所有主题。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes MQTT messages to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCorePublisherCpp:mqttproxy:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_iotcore_publisher"
      },
      "Artifacts": [
        {
```

```
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCorePublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes MQTT messages to IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCorePublisherCpp:mqttproxy:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_iotcore_publisher"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher
        Permission:
          Execute: OWNER
```

以下示例 C++ 应用程序演示了如何使用 AWS IoT Core MQTT IPC 服务向发布消息。 AWS IoT Core

```
#include <iostream>
```

```
#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the Greengrass IPC MQTT publisher (C++).");
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToIoTCoreRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
```

```
        request.SetTopicName(topic);
        request.SetPayload(messageData);
        request.SetQos(qos);

        auto operation = ipcClient.NewPublishToIoTCore();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

示例 AWS IoT Core MQTT 订阅者 (C++)

以下示例配方允许该组件订阅所有主题。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to MQTT messages from IoT
 Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCoreSubscriberCpp:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_iotcore_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCoreSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to MQTT messages from IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCoreSubscriberCpp:mqttproxy:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_iotcore_subscriber"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber
        Permission:
          Execute: OWNER
```

以下示例 C++ 应用程序演示了如何使用 AWS IoT Core MQTT IPC 服务订阅来自的消息。 AWS IoT Core

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}
```

```
    private:

        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string messageTopic =
 message.value().GetTopicName().value().c_str();
                std::cout << "Received new message on topic: " << messageTopic <<
 std::endl;
                std::cout << "Message: " << messageString << std::endl;
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to IoT Core stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
```

```
};

int main() {
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
```

```
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

与组件生命周期交互

使用组件生命周期 IPC 服务可以：

- 更新核心设备上的组件状态。
- 订阅组件状态更新。
- 防止 nucleus 在部署期间停止组件以应用更新。
- 暂停和恢复组件进程。

主题

- SDK 的最低版本
- 授权
- UpdateState
- SubscribeToComponentUpdates
- DeferComponentUpdate
- PauseComponent
- ResumeComponent

SDK 的最低版本

下表列出了在与组件生命周期交互时必须使用的最低版本。AWS IoT Device SDK

SDK	最低版本	
AWS IoT Device SDK适用于 Java v2	v1.2.10	
AWS IoT Device SDK适用于 Python v2	v1.5.3	
AWS IoT Device SDK适用于 C ++ v2	v1.17.0	
AWS IoT Device SDK适用于 JavaScript v2	v1.12.0	

授权

要暂停或恢复自定义组件中的其他组件，必须定义允许您的组件管理其他组件的授权策略。有关定义授权策略的信息，请参阅授权组件执行 IPC 操作。

组件生命周期管理的授权策略具有以下属性。

IPC 服务标识符：`aws.greengrass.ipc.lifecycle`

操作	描述	资源
`aws.greengrass#PauseComponent`	允许组件暂停您指定的组件。	组件名称，或者*用于允许访问所有组件。
`aws.greengrass#ResumeComponent`	允许组件恢复您指定的组件。	组件名称，或者*用于允许访问所有组件。
`*`	允许组件暂停和恢复您指定的组件。	组件名称，或者*用于允许访问所有组件。

授权策略示例

您可以参考以下授权策略示例，以帮助您为组件配置授权策略。

Example 授权策略示例

以下示例授权策略允许组件暂停和恢复所有组件。

```
{
  "accessControl": {
    "aws.greengrass.ipc.lifecycle": {
      "com.example.MyLocalLifecycleComponent:lifecycle:1": {
        "policyDescription": "Allows access to pause/resume all components.",
        "operations": [
          "aws.greengrass#PauseComponent",
          "aws.greengrass#ResumeComponent"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

UpdateState

更新核心设备上组件的状态。

请求

此操作的请求具有以下参数：

`state`

要设置的状态。这个枚举`LifecycleState`具有以下值：

- `RUNNING`
- `ERRORED`

响应

此操作在其响应中未提供任何信息。

SubscribeToComponentUpdates

订阅即可在 AWS IoT Greengrass Core 软件更新组件之前接收通知。该通知指定了在更新过程中核是否会重新启动。

只有当部署的组件更新策略指定通知组件时，nucleus 才会发送更新通知。默认行为是通知组件。有关更多信息，请参阅创建部署以及您在调用CreateDeployment操作时可以提供的DeploymentComponentUpdatePolicy对象。

Important

本地部署不会在更新之前通知组件。

此操作是一种订阅操作，您可以在其中订阅事件消息流。要使用此操作，请定义一个流响应处理程序，其中包含处理事件消息、错误和流关闭的函数。有关更多信息，请参阅 订阅 IPC 事件直播。

事件消息类型：`ComponentUpdatePolicyEvents`

Tip

你可以按照教程学习如何开发一个有条件地推迟组件更新的组件。有关更多信息，请参阅 教程：开发一个可以延迟组件更新的 Greengrass 组件。

请求

此操作的请求没有任何参数。

响应

此操作的响应包含以下信息：

`messages`

通知消息流。此对象包含以下信息：`ComponentUpdatePolicyEvents`

`preUpdateEvent`（Python:`pre_update_event`）

（可选）指示 nucleus 想要更新组件的事件。您可以通过DeferComponentUpdate操作进行响应，以确认更新或推迟更新，直到您的组件准备好重启为止。此对象包含以下信息：`PreComponentUpdateEvent`

`deploymentId`（Python:`deployment_id`）

更新组件的AWS IoT Greengrass部署的 ID。

`isGgcRestarting`（Python:`is_ggc_restarting`）

Nucleus 是否需要重启才能应用更新。

`postUpdateEvent`（Python:`post_update_event`）

（可选）指示 nucleus 更新组件的事件。此对象包含以下信息：`PostComponentUpdateEvent`

`deploymentId`（Python:`deployment_id`）

更新组件的AWS IoT Greengrass部署的 ID。

Note

此功能需要 Greengrass nucleus 组件的 v2.7.0 或更高版本。

DeferComponentUpdate

确认或推迟您发现的组件更新SubscribeToComponentUpdates。您可以指定在组件是否准备好让组件更新继续进行之前等待的时间。您也可以使用此操作告诉 nucleus 您的组件已准备好进行更新。

如果某个组件未响应组件更新通知，则 nucleus 会等待您在部署组件更新策略中指定的时间。超时之后，核心继续部署。默认的组件更新超时时间为 60 秒。有关更多信息，请参阅创建部署以及您在调用CreateDeployment操作时可以提供的DeploymentComponentUpdatePolicy对象。

Tip

你可以按照教程学习如何开发一个有条件地推迟组件更新的组件。有关更多信息，请参阅 教程：开发一个可以延迟组件更新的 Greengrass 组件。

请求

此操作的请求具有以下参数：

`deploymentId`（Python:`deployment_id`）

要推迟的AWS IoT Greengrass部署的 ID。

`message`

（可选）要推迟更新的组件的名称。

默认为发出请求的组件的名称。

`recheckAfterMs`（Python:`recheck_after_ms`）

延迟更新的时间（以毫秒为单位）。原子核会等待这段时间，然后发送另一个你可以用它`PreComponentUpdateEvent`来发现。SubscribeToComponentUpdates

指定`0`以确认更新。这会告诉 nucleus 您的组件已准备好进行更新。

默认为零毫秒，这意味着确认更新。

响应

此操作在其响应中未提供任何信息。

PauseComponent

此功能适用于 Greengrass nucleus 组件的 v2.4.0 及更高版本。 AWS IoT Greengrass目前不支持在 Windows 核心设备上使用此功能。

在核心设备上暂停组件的进程。要恢复组件，请使用ResumeComponent操作。

只能暂停通用组件。如果您尝试暂停任何其他类型的组件，则此操作会抛出。`InvalidRequestError`

> **Note**
>
> 此操作无法暂停容器化进程，例如 Docker 容器。要暂停和恢复 Docker 容器，你可以使用 docker 暂停和 docker 取消暂停命令。

此操作不会暂停组件依赖项或依赖于您暂停的组件的组件。当你暂停依赖于另一个组件的组件时，请考虑这种行为，因为依赖组件在暂停依赖关系时可能会遇到问题。

当您重启或关闭已暂停的组件时（例如通过部署），Greengrass nucleus 会恢复该组件并运行其关闭生命周期。有关重启组件的更多信息，请参阅RestartComponent。

Important

要使用此操作，您必须定义授权策略来授予使用此操作的权限。有关更多信息，请参阅 授权。

SDK 的最低版本

下表列出了暂停和恢复组件时AWS IoT Device SDK必须使用的最低版本。

SDK	最低版本
AWS IoT Device SDK适用于 Java v2	v1.4.3
AWS IoT Device SDK适用于 Python v2	v1.6.2
AWS IoT Device SDK适用于 C ++ v2	v1.13.1
AWS IoT Device SDK适用于 JavaScript v2	v1.12.0

请求

此操作的请求具有以下参数：

`componentName`（Python:`component_name`）

要暂停的组件的名称，它必须是通用组件。有关更多信息，请参阅 组件类型。

响应

此操作在其响应中未提供任何信息。

ResumeComponent

此功能适用于 Greengrass nucleus 组件的 v2.4.0 及更高版本。 AWS IoT Greengrass目前不支持在 Windows 核心设备上使用此功能。

在核心设备上恢复组件的进程。要暂停组件，请使用PauseComponent操作。

您只能恢复已暂停的组件。如果您尝试恢复未暂停的组件，则此操作会抛出。`InvalidRequestError`

Important

要使用此操作，您必须定义授权策略来授予执行此操作的权限。有关更多信息，请参阅 授权。

SDK 的最低版本

下表列出了暂停和恢复组件时AWS IoT Device SDK必须使用的最低版本。

SDK	最低版本
AWS IoT Device SDK适用于 Java v2	v1.4.3
AWS IoT Device SDK适用于 Python v2	v1.6.2
AWS IoT Device SDK适用于 C ++ v2	v1.13.1
AWS IoT Device SDK适用于 JavaScript v2	v1.12.0

请求

此操作的请求具有以下参数：

`componentName`（Python:`component_name`）

要恢复的组件的名称。

响应

此操作在其响应中未提供任何信息。

与组件配置交互

组件配置 IPC 服务允许您执行以下操作：

- 获取和设置组件配置参数。
- 订阅组件配置更新。
- 在 nucleus 应用组件配置更新之前对其进行验证。

主题

- SDK 的最低版本
- GetConfiguration
- UpdateConfiguration
- SubscribeToConfigurationUpdate
- SubscribeToValidateConfigurationUpdates
- SendConfigurationValidityReport

SDK 的最低版本

下表列出了与组件配置交互时AWS IoT Device SDK必须使用的最低版本。

SDK	最低版本
AWS IoT Device SDK适用于 Java v2	v1.2.10
AWS IoT Device SDK适用于 Python v2	v1.5.3
AWS IoT Device SDK适用于 C ++ v2	v1.17.0

SDK	最低版本	
AWS IoT Device SDK适用于 JavaScript v2	v1.12.0	

GetConfiguration

获取核心设备上某个组件的配置值。您可以指定要获取配置值的密钥路径。

请求

此操作的请求具有以下参数：

`componentName`（Python:`component_name`）

（可选）组件的名称。

默认为发出请求的组件的名称。

`keyPath`（Python:`key_path`）

配置值的关键路径。指定一个列表，其中每个条目都是配置对象中单个级别的关键。例如，`["mqtt", "port"]`在以下配置`port`中指定获取的值。

```
{
  "mqtt": {
    "port": 443
  }
}
```

要获取组件的完整配置，请指定一个空列表。

响应

此操作的响应包含以下信息：

`componentName`（Python:`component_name`）

组件名称。

`value`

请求的配置作为对象。

UpdateConfiguration

更新核心设备上此组件的配置值。

请求

此操作的请求具有以下参数：

`keyPath`（Python:`key_path`）

（可选）要更新的容器节点（对象）的密钥路径。指定一个列表，其中每个条目都是配置对象中单个级别的关键。例如，在以下配置中指定`{ "port": 443 }`要设置值的`port`密钥路径`["mqtt"]`和合并值。

```
{
  "mqtt": {
    "port": 443
  }
}
```

密钥路径必须在配置中指定容器节点（对象）。如果组件的配置中不存在该节点，则此操作会创建该节点并将其值设置为中的对象`valueToMerge`。

默认为配置对象的根目录。

`timestamp`

当前 Unix 纪元时间（以毫秒为单位）。此操作使用此时间戳来解析密钥的并发更新。如果组件配置中的密钥的时间戳大于请求中的时间戳，则请求将失败。

`valueToMerge`（Python:`value_to_merge`）

要在中指定的位置合并的配置对象`keyPath`。有关更多信息，请参阅 更新组件配置。

响应

此操作在其响应中未提供任何信息。

SubscribeToConfigurationUpdate

订阅即可在组件配置更新时接收通知。当您订阅密钥时，当该密钥的任何子项更新时，您都会收到通知。

此操作是一种订阅操作，您可以在其中订阅事件消息流。要使用此操作，请定义一个流响应处理程序，其中包含处理事件消息、错误和流关闭的函数。有关更多信息，请参阅 订阅 IPC 事件直播。

事件消息类型：`ConfigurationUpdateEvents`

请求

此操作的请求具有以下参数：

`componentName`（Python:`component_name`）

（可选）组件的名称。

默认为发出请求的组件的名称。

`keyPath`（Python:`key_path`）

要订阅的配置值的关键路径。指定一个列表，其中每个条目都是配置对象中单个级别的关键。例如，`["mqtt", "port"]`在以下配置`port`中指定获取的值。

```
{
  "mqtt": {
    "port": 443
  }
}
```

要订阅组件配置中所有值的更新，请指定一个空列表。

响应

此操作的响应包含以下信息：

`messages`

通知消息流。此对象包含以下信息：`ConfigurationUpdateEvents`

`configurationUpdateEvent`（Python:`configuration_update_event`）

配置更新事件。此对象包含以下信息：`ConfigurationUpdateEvent`

`componentName`（Python:`component_name`）

组件名称。

`keyPath`（Python:`key_path`）

已更新的配置值的密钥路径。

SubscribeToValidateConfigurationUpdates

订阅即可在此组件的配置更新之前接收通知。这样，组件就可以验证自己配置的更新。使用该SendConfigurationValidityReport操作告诉核子核配置是否有效。

Important

本地部署不会将更新通知组件。

此操作是一种订阅操作，您可以在其中订阅事件消息流。要使用此操作，请定义一个流响应处理程序，其中包含处理事件消息、错误和流关闭的函数。有关更多信息，请参阅 订阅 IPC 事件直播。

事件消息类型：`ValidateConfigurationUpdateEvents`

请求

此操作的请求没有任何参数。

响应

此操作的响应包含以下信息：

`messages`

通知消息流。此对象包含以下信息：`ValidateConfigurationUpdateEvents`

`validateConfigurationUpdateEvent`（Python:`validate_configuration_update_event`）

配置更新事件。此对象包含以下信息：`ValidateConfigurationUpdateEvent`

`deploymentId`（Python:`deployment_id`）

更新组件的AWS IoT Greengrass部署的 ID。

`configuration`

包含新配置的对象。

SendConfigurationValidityReport

告诉 nucleus 对该组件的配置更新是否有效。如果您告诉 nucleus 新配置无效，则部署将失败。使用SubscribeToValidateConfigurationUpdates操作订阅以验证配置更新。

如果组件未响应验证配置更新通知，则 nucleus 会等待您在部署的配置验证策略中指定的时间。超时之后，核心继续部署。默认的组件验证超时时间为 20 秒。有关更多信息，请参阅创建部署以及您在调用CreateDeployment操作时可以提供的DeploymentConfigurationValidationPolicy对象。

请求

此操作的请求具有以下参数：

`configurationValidityReport`（Python:`configuration_validity_report`）

该报告告诉核心，配置更新是否有效。此对象包含以下信息：`ConfigurationValidityReport`

`status`

有效性状态。这个枚举`ConfigurationValidityStatus`具有以下值：

- `ACCEPTED`— 配置有效，原子核可以将其应用于该组件。
- `REJECTED`— 配置无效，部署失败。

`deploymentId`（Python:`deployment_id`）

请求配置更新的AWS IoT Greengrass部署的 ID。

`message`

（可选）一条消息，用于报告配置无效的原因。

响应

此操作在其响应中未提供任何信息。

检索秘密值

使用密钥管理器 IPC 服务从核心设备上的密钥中检索机密值。您可以使用密钥管理器组件将加密密钥部署到核心设备。然后，您可以使用 IPC 操作来解密密密钥并在自定义组件中使用其值。

主题

- SDK 的最低版本
- 授权
- GetSecretValue
- 示例

SDK 的最低版本

下表列出了从核心设备上的密钥中检索密钥值时必须使用的最低版本。AWS IoT Device SDK

SDK	最低版本	
AWS IoT Device SDK适用于 Java v2	v1.2.10	
AWS IoT Device SDK适用于 Python v2	v1.5.3	
AWS IoT Device SDK适用于 C ++ v2	v1.17.0	
AWS IoT Device SDK适用于 JavaScript v2	v1.12.0	

授权

要在自定义组件中使用密钥管理器，必须定义授权策略，允许您的组件获取存储在核心设备上的密钥的值。有关定义授权策略的信息，请参阅授权组件执行 IPC 操作。

密钥管理器的授权策略具有以下属性。

IPC 服务标识符：`aws.greengrass.SecretManager`

操作	描述	资源
`aws.greengrass#GetSecretValue` 或 `*`	允许组件获取在核心设备上加密的机密的值。	Secrets Manager 密钥 ARN，或者*用于允许访问所有密钥。

授权策略示例

您可以参考以下授权策略示例，以帮助您为组件配置授权策略。

Example 授权策略示例

以下示例授权策略允许组件获取核心设备上任何密钥的值。

> **Note**
> 我们建议在生产环境中缩小授权策略的范围，以便组件仅检索其使用的密钥。部署组件时，您可以将*通配符更改为机密 ARN 列表。

```
{
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.MySecretComponent:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

GetSecretValue

获取您存储在核心设备上的密钥的值。

此操作与 Secrets Manager 操作类似，您可以使用该操作来获取中密钥的值AWS Cloud。有关更多信息，请参阅《AWS Secrets Manager API 参考》中的 GetSecretValue。

请求

此操作的请求具有以下参数：

`secretId`（Python:`secret_id`）

要获取的秘密的名称。您可以指定密钥的 Amazon 资源名称 (ARN) 或友好名称。

`versionId`（Python:`version_id`）

（可选）要获取的版本的 ID。

您可指定 `versionId` 或 `versionStage`。

如果未指定`versionId`或`versionStage`，则此操作默认为带有`AWSCURRENT`标签的版本。

`versionStage`（Python:`version_stage`）

（可选）要获取的版本的暂存标签。

您可指定 `versionId` 或 `versionStage`。

如果未指定`versionId`或`versionStage`，则此操作默认为带有`AWSCURRENT`标签的版本。

响应

此操作的响应包含以下信息：

`secretId`（Python:`secret_id`）

密钥的 ID。

`versionId`（Python:`version_id`）

此版本的密钥的 ID。

`versionStage`（Python:`version_stage`）

此版本机密所附的暂存标签列表。

`secretValue`（Python:`secret_value`）

这个版本的秘密的价值。此对象包含以下信息。`SecretValue`

`secretString`（Python:`secret_string`）

您以字符串形式提供给 Secrets Manager 的受保护机密信息的解密部分。

`secretBinary`（Python:`secret_binary`）

（可选）您以字节数组形式作为二进制数据提供给 Secrets Manager 的受保护机密信息的解密部分。此属性包含以 base64 编码字符串形式存在的二进制数据。

如果您在 Secrets Manager 控制台中创建了密钥，则不会使用此属性。

示例

以下示例演示了如何在自定义组件代码中调用此操作。

Java (IPC client V1)

Example 示例：获取密钥值

> **Note**
>
> 此示例使用一个`IPCUtils`类来创建与 C AWS IoT Greengrass ore IPC 服务的连接。有关更多信息，请参阅 Connect 到 C AWS IoT Greengrass ore IPC 服务。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetSecretValueResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueRequest;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetSecretValue {
```

```
    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String secretArn = args[0];
        String versionStage = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetSecretValueResponseHandler responseHandler =
                    GetSecretValue.getSecretValue(ipcClient, secretArn,
 versionStage);
            CompletableFuture<GetSecretValueResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                GetSecretValueResponse response =
 futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                response.getSecretValue().postFromJson();
                String secretString = response.getSecretValue().getSecretString();
                System.out.println("Successfully retrieved secret value: " +
 secretString);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while retrieving secret: " +
 secretArn);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while retrieving secret:
 " + secretArn);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetSecretValueResponseHandler
 getSecretValue(GreengrassCoreIPCClient greengrassCoreIPCClient, String secretArn,
 String versionStage) {
```

```
        GetSecretValueRequest getSecretValueRequest = new GetSecretValueRequest();
        getSecretValueRequest.setSecretId(secretArn);
        getSecretValueRequest.setVersionStage(versionStage);
        return greengrassCoreIPCClient.getSecretValue(getSecretValueRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example 示例：获取密钥值

Note

此示例假设您使用的是 Python v2 版本的 1.5.4 或更高版本。AWS IoT Device SDK

```
import json

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

secret_id = 'arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef'
TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

request = GetSecretValueRequest()
request.secret_id = secret_id
request.version_stage = 'AWSCURRENT'
operation = ipc_client.new_get_secret_value()
operation.activate(request)
future_response = operation.get_response()
response = future_response.result(TIMEOUT)
secret_json = json.loads(response.secret_value.secret_string)
# Handle secret value.
```

JavaScript

Example 示例：获取密钥值

```
import {
    GetSecretValueRequest,
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";

class GetSecretValue {
    private readonly secretId : string;
    private readonly versionStage : string;
    private ipcClient : greengrasscoreipc.Client

    constructor() {
        this.secretId = "<define_your_own_secretId>"
        this.versionStage = "<define_your_own_versionStage>"

        this.getSecretValue().then(r => console.log("Started workflow"));
    }

    private async getSecretValue() {
        try {
            this.ipcClient = await getIpcClient();

            const getSecretValueRequest : GetSecretValueRequest = {
                secretId: this.secretId,
                versionStage: this.versionStage,
            };

            const result = await
 this.ipcClient.getSecretValue(getSecretValueRequest);
            const secretString = result.secretValue.secretString;
            console.log("Successfully retrieved secret value: " + secretString)
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
```

```
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const getSecretValue = new GetSecretValue();
```

示例

使用以下示例来学习如何在组件中使用密钥管理器 IPC 服务。

示例：打印密钥（Python、IPC 客户端 V1）

此示例组件打印您部署到核心设备的密钥的值。

> **Important**
>
> 此示例组件打印密钥的值，因此仅将其与存储测试数据的密钥一起使用。不要使用此组件来打印存储重要信息的密钥的值。

主题

- 配方
- 构件
- 使用量

配方

以下示例配方定义了一个秘密 ARN 配置参数，并允许该组件获取核心设备上任何密钥的值。

Note

我们建议在生产环境中缩小授权策略的范围，以便组件仅检索其使用的密钥。部署组件时，您可以将*通配符更改为机密 ARN 列表。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PrintSecret",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Prints the value of an AWS Secrets Manager secret.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.SecretManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "SecretArn": "",
      "accessControl": {
        "aws.greengrass.SecretManager": {
          "com.example.PrintSecret:secrets:1": {
            "policyDescription": "Allows access to a secret.",
            "operations": [
              "aws.greengrass#GetSecretValue"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
```

```
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PrintSecret
ComponentVersion: 1.0.0
ComponentDescription: Prints the value of a Secrets Manager secret.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.SecretManager:
    VersionRequirement: "^2.0.0"
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    SecretArn: ''
    accessControl:
      aws.greengrass.SecretManager:
        com.example.PrintSecret:secrets:1:
          policyDescription: Allows access to a secret.
          operations:
            - aws.greengrass#GetSecretValue
          resources:
            - "*"
Manifests:
```

```
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
```

构件

以下示例 Python 应用程序演示了如何使用密钥管理器 IPC 服务在核心设备上获取密钥的值。

```
import concurrent.futures
import json
import sys
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

TIMEOUT = 10

if len(sys.argv) == 1:
    print('Provide SecretArn in the component configuration.', file=sys.stdout)
    exit(1)

secret_id = sys.argv[1]

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = GetSecretValueRequest()
    request.secret_id = secret_id
    operation = ipc_client.new_get_secret_value()
    operation.activate(request)
    future_response = operation.get_response()
```

```
    try:
        response = future_response.result(TIMEOUT)
        secret_json = json.loads(response.secret_value.secret_string)
        print('Successfully got secret: ' + secret_id)
        print('Secret value: ' + str(secret_json))
    except concurrent.futures.TimeoutError:
        print('Timeout occurred while getting secret: ' + secret_id, file=sys.stderr)
    except UnauthorizedError as e:
        print('Unauthorized error while getting secret: ' + secret_id,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while getting secret: ' + secret_id, file=sys.stderr)
        raise e
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

使用量

您可以将此示例组件与密钥管理器组件一起使用，在核心设备上部署和打印密钥的值。

创建、部署和打印测试密钥

1. 使用测试数据创建 Secrets Manager 密钥。

   Linux or Unix

   ```
   aws secretsmanager create-secret \
     --name MyTestGreengrassSecret \
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   Windows Command Prompt (CMD)

   ```
   aws secretsmanager create-secret ^
     --name MyTestGreengrassSecret ^
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

PowerShell

```
aws secretsmanager create-secret `
  --name MyTestGreengrassSecret `
  --secret-string '{"my-secret-key": "my-secret-value"}'
```

保存密钥的 ARN，以便在以下步骤中使用。

有关更多信息，请参阅《AWS Secrets Manager用户指南》中的创建密钥。

2. 使用以下配置合并更新部署密钥管理器组件 (aws.greengrass.SecretManager)。指定您之前创建的密钥的 ARN。

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
    }
  ]
}
```

有关更多信息，请参阅将AWS IoT Greengrass组件部署到设备或 Greengr ass CLI 部署命令。

3. 使用以下配置合并更新创建和部署本节中的示例组件。指定您之前创建的密钥的 ARN。

```
{
  "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret",
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.PrintSecret:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
        ]
      }
```

```
      }
    }
  }
```

有关更多信息，请参阅 创建 AWS IoT Greengrass 组件。

4. 查看AWS IoT Greengrass核心软件日志以验证部署是否成功，并查看`com.example.PrintSecret`组件日志以查看打印的密钥值。有关更多信息，请参阅 监控 AWS IoT Greengrass日志。

与局部阴影互动

使用 shadow IPC 服务与设备上的本地阴影进行交互。您选择与之交互的设备可以是您的核心设备或连接的客户端设备。

要使用这些 IPC 操作，请将影子管理器组件作为依赖项包含在自定义组件中。然后，您可以在自定义组件中使用 IPC 操作，通过阴影管理器与设备上的本地阴影进行交互。要使自定义组件能够对本地阴影状态的变化做出反应，您还可以使用发布/订阅 IPC 服务来订阅影子事件。有关使用发布/订阅服务的更多信息，请参阅。发布/订阅本地消息

> Note
>
> 要使核心设备能够与客户端设备影子进行交互，您还必须配置和部署 MQTT 桥接组件。有关更多信息，请参阅启用影子管理器以与客户端设备通信。

主题

- SDK 的最低版本
- 授权
- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

SDK 的最低版本

下表列出了与本地阴影交互时必须使用的最低版本。AWS IoT Device SDK

SDK	最低版本	
AWS IoT Device SDK适用于 Java v2	v1.4.0	
AWS IoT Device SDK适用于 Python v2	v1.6.0	
AWS IoT Device SDK适用于 C ++ v2	v1.17.0	
AWS IoT Device SDK适用于 JavaScript v2	v1.12.0	

授权

要在自定义组件中使用影子 IPC 服务，必须定义允许您的组件与阴影交互的授权策略。有关定义授权策略的信息，请参阅授权组件执行 IPC 操作。

影子交互的授权策略具有以下属性。

IPC 服务标识符：`aws.greengrass.ShadowManager`

操作	描述	资源
`aws.greengrass#GetThingShadow`	允许组件检索事物的影子。	以下字符串之一： • `$aws/things/`*thingName*`/shadow/`，以允许访问经典设备影子。 • `$aws/things/`*thingName*`/shadow/name/`*shadowName*，以允许访问已命名的影子。 • `*`以允许访问所有阴影。

操作	描述	资源
aws.greengrass#UpdateThingShadow	允许组件更新事物的阴影。	以下字符串之一： • $aws/things/ *thingName* /shadow/，以允许访问经典设备影子。 • $aws/things/ *thingName* /shadow/name/ *shadowName* ，以允许访问已命名的影子。 • *以允许访问所有阴影。
aws.greengrass#DeleteThingShadow	允许组件删除事物的阴影。	以下字符串之一： • $aws/things/ *thingName* /shadow/，以允许访问经典设备影子 • $aws/things/ *thingName* /shadow/name/ *shadowName* ，以允许访问已命名的影子 • *，以允许访问所有阴影。
aws.greengrass#ListNamedShadowsForThing	允许组件检索事物的命名阴影列表。	允许访问事物以列出其阴影的事物名称字符串。 用于*允许访问所有内容。

IPC 服务标识符：`aws.greengrass.ipc.pubsub`

操作	描述	资源
aws.greengrass#SubscribeToTopic	允许组件订阅您指定的主题的消息。	以下主题字符串之一： • *shadowTopicPrefix* /get/accepted • *shadowTopicPrefix* /get/rejected • *shadowTopicPrefix* /delete/accepted • *shadowTopicPrefix* /delete/rejected • *shadowTopicPrefix* /update/accepted • *shadowTopicPrefix* /update/delta • *shadowTopicPrefix* /update/rejected 主题前缀的值*shadowTopicPrefix* 取决于阴影的类型： • 经典阴影：$aws/things/ *thingName* /shadow • 命名为阴影：$aws/things/ *thingName* /shadow/name/ *shadowName* 用于*允许访问所有主题。 在 Greengrass nucleus v2.6.0 及更高版本中，你可以订阅包含 MQTT 主题通配符（和）的

操作	描述	资源
		主题。# +此主题字符串支持 MQTT 主题通配符作为文字字符。例如，如果组件的授权策略授予访问权限test/topic/# ，则该组件可以订阅 test/topic/# ，但无法订阅test/topic/filter 。

本地影子授权策略中的配方变量

如果您使用 Greengrass 核心的 v2.6.0 或更高版本，并且将 Greengrass nucleus 的配置选项设置为，则可以在授权策略中使用配方变量interpolateComponentConfiguration。true{iot:thingName}此功能使您可以为一组核心设备配置单个授权策略，其中每个核心设备只能访问自己的影子。例如，您可以允许组件访问以下资源以进行影子 IPC 操作。

```
$aws/things/{iot:thingName}/shadow/
```

授权策略示例

您可以参考以下授权策略示例，以帮助您为组件配置授权策略。

Example 示例：允许一组核心设备与本地阴影进行交互

Important

此示例使用了 Greengrass nucleus 组件的 v2.6.0 及更高版本中可用的功能。Greengrass nucleus v2.6.0 增加了对大多数配方变量的支持，例如组件配置中的变量。{iot:thingName}要启用此功能，请将 Green interpolateComponentConfigurationgrass 核的配置选项设置为。true有关适用于所有版本的 Greengrass nucleus 的示例，请参阅单核设备的授权策略示例。

以下示例授权策略允许该组件com.example.MyShadowInteractionComponent与经典设备影子以及运行该组件的核心设备的命名影子myNamedShadow进行交互。此策略还允许此组件接收有关这些影子的本地主题的消息。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "{iot:thingName}"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/get/accepted",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/{iot:thingName}/shadow
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - '{iot:thingName}'
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/get/accepted
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted
```

Example 示例：允许一组核心设备与客户端设备影子进行交互

Important

此功能需要 Greengrass nucleus v2.6.0 或更高版本、影子管理器 v2.2.0 或更高版本以及 MQTT bridge v2.2.0 或更高版本。您必须配置 MQTT 网桥才能使影子管理器能够与客户端设备通信。

以下示例授权策略允许该组件与名称`com.example.MyShadowInteractionComponent`以开头的客户端设备的所有设备影子进行交互`MyClientDevice`。

Note

要使核心设备能够与客户端设备影子进行交互，您还必须配置和部署 MQTT 桥接组件。有关更多信息，请参阅启用影子管理器以与客户端设备通信。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyClientDevice*/shadow",
          "$aws/things/MyClientDevice*/shadow/name/*"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "MyClientDevice*"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
```

```
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyClientDevice*/shadow
        - $aws/things/MyClientDevice*/shadow/name/*
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyClientDevice*
```

Example 示例：允许单核设备与本地阴影交互

以下示例授权策略允许该组件com.example.MyShadowInteractionComponent与设备的经典设备影子和命名的影子myNamedShadow进行交互MyThingName。此策略还允许此组件接收有关这些影子的本地主题的消息。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow",
          "$aws/things/MyThingName/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
```

```
        "resources": [
          "MyThingName"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/get/accepted",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyThingName/shadow
        - $aws/things/MyThingName/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyThingName
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
```

```
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/get/accepted
        - $aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted
```

Example 示例：允许一组核心设备对本地阴影状态变化做出反应

Important

此示例使用了 Greengrass nucleus 组件的 v2.6.0 及更高版本中可用的功能。Greengrass nucleus v2.6.0 增加了对大多数配方变量的支持，例如组件配置中的变量。{iot:thingName}要启用此功能，请将 Green interpolateComponentConfigurationgrass 核的配置选项设置为。true有关适用于所有版本的 Greengrass nucleus 的示例，请参阅单核设备的授权策略示例。

以下示例访问控制策略com.example.MyShadowReactiveComponent允许自定义用户在运行该组件的每台核心设备上接收myNamedShadow有关经典设备影子和命名影子的/update/delta主题的消息。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/update/delta",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/update/delta
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta
```

Example 示例：允许单核设备对本地阴影状态变化做出反应

以下示例访问控制策略`com.example.MyShadowReactiveComponent`允许自定义用户接收有关经典设备影子`/update/delta`主题和设备命名影子的`myNamedShadow`消息`MyThingName`。

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/update/delta",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
```

```
        policyDescription: Allows access to shadow pubsub topics
        operations:
          - 'aws.greengrass#SubscribeToTopic'
        resources:
          - $aws/things/MyThingName/shadow/update/delta
          - $aws/things/MyThingName/shadow/name/myNamedShadow/update/delta
```

GetThingShadow

获取指定事物的阴影。

请求

此操作的请求具有以下参数：

`thingName`（Python:`thing_name`）

事物的名称。

类型：`string`

`shadowName`（Python:`shadow_name`）

影子的名称。要指定事物的经典阴影，请将此参数设置为空字符串 ("")。

Warning

该AWS IoT Greengrass服务使用`AWSManagedGreengrassV2Deployment`命名的影子来管理针对单个核心设备的部署。这个名为 shadow 的保留供AWS IoT Greengrass服务使用。请勿更新或删除这个名为 shadow 的影子。

类型：`string`

响应

此操作的响应包含以下信息：

`payload`

响应状态文档为 blob。

类型：其中object包含以下信息：

state

状态信息。

此对象包含以下信息。

desired

设备中请求更新的状态属性和值。

类型：map键值对

reported

设备报告的状态属性和值。

类型：map键值对

delta

所需状态和报告的状态属性和值之间的差异。仅当desired和reported状态不同时，才会出现此属性。

类型：map键值对

metadata

desired和reported部分中每个属性的时间戳，以便您可以确定何时更新状态。

类型：string

timestamp

生成响应的时代日期和时间。

类型：integer

clientToken（Python:clientToken）

用于匹配请求和相应响应的令牌

类型：string

version

本地影子文档的版本。

类型：integer

错误

此操作可能会返回以下错误。

`InvalidArgumentsError`

本地影子服务无法验证请求参数。如果请求包含格式错误的 JSON 或不支持的字符，则可能会发生这种情况。

`ResourceNotFoundError`

找不到请求的本地影子文档。

`ServiceError`

发生了内部服务错误，或者对 IPC 服务的请求数超过了影子管理器组件中`maxLocalRequestsPerSecondPerThing`和`maxTotalLocalRequestsRate`配置参数中指定的限制。

`UnauthorizedError`

该组件的授权策略不包括此操作所需的权限。

示例

以下示例演示了如何在自定义组件代码中调用此操作。

Java (IPC client V1)

Example 示例：获取事物影子

此示例使用一个`IPCUtils`类来创建与 C AWS IoT Greengrass ore IPC 服务的连接。有关更多信息，请参阅 Connect 到 C AWS IoT Greengrass ore IPC 服务。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowRequest;
```

```
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetThingShadowResponseHandler responseHandler =
                    GetThingShadow.getThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<GetThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                GetThingShadowResponse response =
 futureResponse.get(TIMEOUT_SECONDS,
                        TimeUnit.SECONDS);
                String shadowPayload = new String(response.getPayload(),
 StandardCharsets.UTF_8);
                System.out.printf("Successfully got shadow %s/%s: %s%n", thingName,
 shadowName,
                        shadowPayload);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while getting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
```

```
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while getting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.printf("Unable to find shadow to get: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetThingShadowResponseHandler
 getThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String thingName,
 String shadowName) {
        GetThingShadowRequest getThingShadowRequest = new GetThingShadowRequest();
        getThingShadowRequest.setThingName(thingName);
        getThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.getThingShadow(getThingShadowRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example 示例：获取事物影子

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import GetThingShadowRequest

TIMEOUT = 10

def sample_get_thing_shadow_request(thingName, shadowName):
    try:
```

```
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the GetThingShadow request
        get_thing_shadow_request = GetThingShadowRequest()
        get_thing_shadow_request.thing_name = thingName
        get_thing_shadow_request.shadow_name = shadowName

        # retrieve the GetThingShadow response after sending the request to the IPC
 server
        op = ipc_client.new_get_thing_shadow()
        op.activate(get_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
        ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example 示例：获取事物影子

```
import {
    GetThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class GetThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
```

```
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleGetThingShadowOperation(this.thingName,
                this.shadowName);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleGetThingShadowOperation(
        thingName: string,
        shadowName: string
    ) {
        const request: GetThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        };
        const response = await this.ipcClient.getThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use casesâ
        throw err
    }
}
```

```
const startScript = new GetThingShadow();
```

UpdateThingShadow

更新指定事物的阴影。如果阴影不存在，则会创建一个阴影。

请求

此操作的请求具有以下参数：

`thingName`（Python:`thing_name`）

事物的名称。

类型：`string`

`shadowName`（Python:`shadow_name`）

影子的名称。要指定事物的经典阴影，请将此参数设置为空字符串 ("")。

> **Warning**
>
> 该AWS IoT Greengrass服务使用`AWSManagedGreengrassV2Deployment`命名的影子来管理针对单个核心设备的部署。这个名为 shadow 的保留供AWS IoT Greengrass服务使用。请勿更新或删除这个名为 shadow 的影子。

类型：`string`

`payload`

请求状态文档为 blob。

类型：其中`object`包含以下信息：

`state`

要更新的状态信息。此 IPC 操作仅影响指定的字段。

此对象包含以下信息。通常，您将在同一个请求中使用`desiredreported`属性或属性，但不能同时使用这两个属性。

`desired`

设备中请求更新的状态属性和值。

类型：map键值对

reported

设备报告的状态属性和值。

类型：map键值对

clientToken（Python:client_token）

（可选）用于匹配客户端令牌的请求和相应响应的令牌。

类型：string

version

（可选）要更新的本地影子文档的版本。仅当指定的版本与其最新版本匹配时，影子服务才会处理更新。

类型：integer

响应

此操作的响应包含以下信息：

payload

响应状态文档为 blob。

类型：其中object包含以下信息：

state

状态信息。

此对象包含以下信息。

desired

设备中请求更新的状态属性和值。

类型：map键值对

reported

设备报告的状态属性和值。

类型：map键值对

delta

设备报告的状态属性和值。

类型：map键值对

metadata

desired和reported部分中每个属性的时间戳，以便您可以确定何时更新状态。

类型：string

timestamp

生成响应的时代日期和时间。

类型：integer

clientToken（Python:client_token）

用于匹配请求和相应响应的令牌。

类型：string

version

更新完成后的本地影子文档的版本。

类型：integer

错误

此操作可能会返回以下错误。

ConflictError

本地影子服务在更新操作期间遇到了版本冲突。当请求负载中的版本与最新的可用本地影子文档中的版本不匹配时，就会发生这种情况。

InvalidArgumentsError

本地影子服务无法验证请求参数。如果请求包含格式错误的 JSON 或不支持的字符，则可能会发生这种情况。

有效的payload具有以下属性：

- 该state节点存在，并且是一个包含desired或reported状态信息的对象。

- desired和reported节点要么是对象，要么为空。这些对象中至少有一个必须包含有效的状态信息。
- desired和reported对象的深度不能超过八个节点。
- 该clientToken值的长度不能超过 64 个字符。
- 该version值必须等于1或更高。

ServiceError

发生了内部服务错误，或者对 IPC 服务的请求数超过了影子管理器组件中maxLocalRequestsPerSecondPerThing和maxTotalLocalRequestsRate配置参数中指定的限制。

UnauthorizedError

该组件的授权策略不包括此操作所需的权限。

示例

以下示例演示了如何在自定义组件代码中调用此操作。

Java (IPC client V1)

Example 示例：更新事物影子

此示例使用一个IPCUtils类来创建与 C AWS IoT Greengrass ore IPC 服务的连接。有关更多信息，请参阅 Connect 到 C AWS IoT Greengrass ore IPC 服务。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.UpdateThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowResponse;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
```

```
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class UpdateThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        byte[] shadowPayload = args[2].getBytes(StandardCharsets.UTF_8);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            UpdateThingShadowResponseHandler responseHandler =
                    UpdateThingShadow.updateThingShadow(ipcClient, thingName,
 shadowName,
                            shadowPayload);
            CompletableFuture<UpdateThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully updated shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while updating shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while updating shadow: %s/
%s%n",
                            thingName, shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
```

```
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static UpdateThingShadowResponseHandler
 updateThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName, byte[] shadowPayload) {
        UpdateThingShadowRequest updateThingShadowRequest = new
 UpdateThingShadowRequest();
        updateThingShadowRequest.setThingName(thingName);
        updateThingShadowRequest.setShadowName(shadowName);
        updateThingShadowRequest.setPayload(shadowPayload);
        return greengrassCoreIPCClient.updateThingShadow(updateThingShadowRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example 示例：更新事物影子

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import UpdateThingShadowRequest

TIMEOUT = 10

def sample_update_thing_shadow_request(thingName, shadowName, payload):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the UpdateThingShadow request
        update_thing_shadow_request = UpdateThingShadowRequest()
        update_thing_shadow_request.thing_name = thingName
        update_thing_shadow_request.shadow_name = shadowName
        update_thing_shadow_request.payload = payload

        # retrieve the UpdateThingShadow response after sending the request to the
 IPC server
```

```
        op = ipc_client.new_update_thing_shadow()
        op.activate(update_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ConflictError | UnauthorizedError | ServiceError
```

JavaScript

Example 示例：更新事物影子

```
import {
    UpdateThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class UpdateThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;
    private shadowDocumentStr: string;

    constructor() {
        // Define args parameters here

        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.shadowDocumentStr = "<define_your_own_payload>";

        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
```

```
        try {
            await this.handleUpdateThingShadowOperation(
                this.thingName,
                this.shadowName,
                this.shadowDocumentStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleUpdateThingShadowOperation(
        thingName: string,
        shadowName: string,
        payloadStr: string
    ) {
        const request: UpdateThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName,
            payload: payloadStr
        }
        // make the UpdateThingShadow request
        const response = await this.ipcClient.updateThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new UpdateThingShadow();
```

DeleteThingShadow

删除指定事物的影子。

从影子管理器 v2.0.4 开始，删除阴影会增加版本号。例如，当您在版本 1 中删除阴影`MyThingShadow`时，已删除阴影的版本为 2。如果您随后使用该名称重新创建阴影`MyThingShadow`，则该阴影的版本为 3。

请求

此操作的请求具有以下参数：

`thingName`（Python:`thing_name`）

事物的名称。

类型：`string`

`shadowName`（Python:`shadow_name`）

影子的名称。要指定事物的经典阴影，请将此参数设置为空字符串 ("")。

Warning

该AWS IoT Greengrass服务使用`AWSManagedGreengrassV2Deployment`命名的影子来管理针对单个核心设备的部署。这个名为 shadow 的保留供AWS IoT Greengrass服务使用。请勿更新或删除这个名为 shadow 的影子。

类型：`string`

响应

此操作的响应包含以下信息：

`payload`

一个空的响应状态文档。

错误

此操作可能会返回以下错误。

`InvalidArgumentsError`

本地影子服务无法验证请求参数。如果请求包含格式错误的 JSON 或不支持的字符，则可能会发生这种情况。

`ResourceNotFoundError`

找不到请求的本地影子文档。

`ServiceError`

发生了内部服务错误，或者对 IPC 服务的请求数超过了影子管理器组件中`maxLocalRequestsPerSecondPerThing`和`maxTotalLocalRequestsRate`配置参数中指定的限制。

`UnauthorizedError`

该组件的授权策略不包括此操作所需的权限。

示例

以下示例演示了如何在自定义组件代码中调用此操作。

Java (IPC client V1)

Example 示例：删除事物影子

Note

此示例使用一个`IPCUtils`类来创建与 C AWS IoT Greengrass ore IPC 服务的连接。有关更多信息，请参阅 Connect 到 C AWS IoT Greengrass ore IPC 服务。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.DeleteThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
```

```
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class DeleteThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            DeleteThingShadowResponseHandler responseHandler =
                    DeleteThingShadow.deleteThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<DeleteThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully deleted shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while deleting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while deleting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.printf("Unable to find shadow to delete: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
```

```
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static DeleteThingShadowResponseHandler
 deleteThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName) {
        DeleteThingShadowRequest deleteThingShadowRequest = new
 DeleteThingShadowRequest();
        deleteThingShadowRequest.setThingName(thingName);
        deleteThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.deleteThingShadow(deleteThingShadowRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example 示例：删除事物影子

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import DeleteThingShadowRequest

TIMEOUT = 10

def sample_delete_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the DeleteThingShadow request
        delete_thing_shadow_request = DeleteThingShadowRequest()
        delete_thing_shadow_request.thing_name = thingName
        delete_thing_shadow_request.shadow_name = shadowName
```

```
        # retrieve the DeleteThingShadow response after sending the request to the
 IPC server
        op = ipc_client.new_delete_thing_shadow()
        op.activate(delete_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example 示例：删除事物影子

```
import {
    DeleteThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class DeleteThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
```

```
            await this.handleDeleteThingShadowOperation(this.thingName,
 this.shadowName)
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleDeleteThingShadowOperation(thingName: string, shadowName: string) {
        const request: DeleteThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        }
        // make the DeleteThingShadow request
        const response = await this.ipcClient.deleteThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new DeleteThingShadow();
```

ListNamedShadowsForThing

列出指定事物的已命名阴影。

请求

此操作的请求具有以下参数：

thingName（Python:thing_name）

事物的名称。

类型：string

pageSize（Python:page_size）

（可选）每次调用中要返回的影子名称的数量。

类型：integer

默认值：25

最大值：100

nextToken（Python:next_token）

（可选）用于检索下一组结果的标记。该值在分页结果中返回，并在返回下一页的调用中使用。

类型：string

响应

此操作的响应包含以下信息：

results

影子名称列表。

类型：array

timestamp

（可选）生成响应的日期和时间。

类型：integer

nextToken（Python:next_token）

（可选）分页请求中使用的令牌值，用于检索序列中的下一页。当没有更多影子名称可供返回时，此标记不存在。

类型：string

Note

如果请求的页面大小与响应中的影子名称数量完全匹配，则存在此标记；但是，使用时，它会返回一个空列表。

错误

此操作可能会返回以下错误。

`InvalidArgumentsError`

本地影子服务无法验证请求参数。如果请求包含格式错误的 JSON 或不支持的字符，则可能会发生这种情况。

`ResourceNotFoundError`

找不到请求的本地影子文档。

`ServiceError`

发生了内部服务错误，或者对 IPC 服务的请求数超过了影子管理器组件中`maxLocalRequestsPerSecondPerThing`和`maxTotalLocalRequestsRate`配置参数中指定的限制。

`UnauthorizedError`

该组件的授权策略不包括此操作所需的权限。

示例

以下示例演示了如何在自定义组件代码中调用此操作。

Java (IPC client V1)

Example 示例：列出事物的名字命名为 shadows

Note

此示例使用一个`IPCUtils`类来创建与 C AWS IoT Greengrass ore IPC 服务的连接。有关更多信息，请参阅 Connect 到 C AWS IoT Greengrass ore IPC 服务。

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import
 software.amazon.awssdk.aws.greengrass.ListNamedShadowsForThingResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingRequest;
import
 software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class ListNamedShadowsForThing {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            List<String> namedShadows = new ArrayList<>();
            String nextToken = null;
            try {
                // Send additional requests until there's no pagination token in the
 response.
                do {
                    ListNamedShadowsForThingResponseHandler responseHandler =

 ListNamedShadowsForThing.listNamedShadowsForThing(ipcClient, thingName,
                                    nextToken, 25);
```

```
                    CompletableFuture<ListNamedShadowsForThingResponse>
 futureResponse =
                            responseHandler.getResponse();
                    ListNamedShadowsForThingResponse response =
                            futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                    List<String> responseNamedShadows = response.getResults();
                    namedShadows.addAll(responseNamedShadows);
                    nextToken = response.getNextToken();
                } while (nextToken != null);
                System.out.printf("Successfully got named shadows for thing %s: %s
%n", thingName,
                        String.join(",", namedShadows));
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while listing named shadows for
 thing: " + thingName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while listing named
 shadows for " +
                            "thing: " + thingName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.println("Unable to find thing to list named shadows:
 " + thingName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static ListNamedShadowsForThingResponseHandler
 listNamedShadowsForThing(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String nextToken, int pageSize) {
        ListNamedShadowsForThingRequest listNamedShadowsForThingRequest =
                new ListNamedShadowsForThingRequest();
        listNamedShadowsForThingRequest.setThingName(thingName);
        listNamedShadowsForThingRequest.setNextToken(nextToken);
        listNamedShadowsForThingRequest.setPageSize(pageSize);
```

```
        return
 greengrassCoreIPCClient.listNamedShadowsForThing(listNamedShadowsForThingRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example 示例：列出事物的名字命名为 shadows

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import ListNamedShadowsForThingRequest

TIMEOUT = 10

def sample_list_named_shadows_for_thing_request(thingName, nextToken, pageSize):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the ListNamedShadowsForThingRequest request
        list_named_shadows_for_thing_request = ListNamedShadowsForThingRequest()
        list_named_shadows_for_thing_request.thing_name = thingName
        list_named_shadows_for_thing_request.next_token = nextToken
        list_named_shadows_for_thing_request.page_size = pageSize

        # retrieve the ListNamedShadowsForThingRequest response after sending the
 request to the IPC server
        op = ipc_client.new_list_named_shadows_for_thing()
        op.activate(list_named_shadows_for_thing_request)
        fut = op.get_response()

        list_result = fut.result(TIMEOUT)

        # additional returned fields
        timestamp = list_result.timestamp
        next_token = result.next_token
        named_shadow_list = list_result.results

        return named_shadow_list, next_token, timestamp

    except InvalidArgumentsError as e:
```

```
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example 示例：列出事物的名字命名为 shadows

```
import {
    ListNamedShadowsForThingRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class listNamedShadowsForThing {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private pageSizeStr: string;
    private nextToken: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.pageSizeStr = "<define_your_own_pageSize>";
        this.nextToken = "<define_your_own_token>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleListNamedShadowsForThingOperation(this.thingName,
                this.nextToken, this.pageSizeStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }
```

```
    async handleListNamedShadowsForThingOperation(
        thingName: string,
        nextToken: string,
        pageSizeStr: string
    ) {
        let request: ListNamedShadowsForThingRequest = {
            thingName: thingName,
            nextToken: nextToken,
        };
        if (pageSizeStr) {
            request.pageSize = parseInt(pageSizeStr);
        }
        // make the ListNamedShadowsForThing request
        const response = await this.ipcClient.listNamedShadowsForThing(request);
        const shadowNames = response.results;
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new listNamedShadowsForThing();
```

管理本地部署和组件

Note

此功能适用于 Greengrass nucleus 组件的 2.6.0 及更高版本。

使用 Greengrass CLI IPC 服务来管理核心设备上的本地部署和 Greengrass 组件。

要使用这些 IPC 操作，请将 Greengrass CLI 组件的 2.6.0 或更高版本作为依赖项包含在自定义组件中。然后，您可以在自定义组件中使用 IPC 操作来执行以下操作：

- 创建本地部署以修改和配置核心设备上的 Greengrass 组件。
- 重启并停止核心设备上的 Greengrass 组件。
- 生成可用于登录本地调试控制台的密码。

主题

- SDK 的最低版本
- 授权
- CreateLocalDeployment
- ListLocalDeployments
- GetLocalDeploymentStatus
- ListComponents
- GetComponentDetails
- RestartComponent
- StopComponent
- CreateDebugPassword

SDK 的最低版本

下表列出了与 Greengras AWS IoT Device SDK s CLI IPC 服务交互时必须使用的最低版本。

SDK	最低版本
AWS IoT Device SDK适用于 Java v2	v1.2.10
AWS IoT Device SDK适用于 Python v2	v1.5.3
AWS IoT Device SDK适用于 C ++ v2	v1.17.0

SDK	最低版本	
AWS IoT Device SDK适用于 JavaScript v2	v1.12.0	

授权

要在自定义组件中使用 Greengrass CLI IPC 服务，必须定义允许您的组件管理本地部署和组件的授权策略。有关定义授权策略的信息，请参阅授权组件执行 IPC 操作。

Greengrass CLI 的授权策略具有以下属性。

IPC 服务标识符：aws.greengrass.Cli

操作	描述	资源
aws.greengrass#CreateLocalDeployment	允许组件在核心设备上创建本地部署。	*
aws.greengrass#ListLocalDeployments	允许组件列出核心设备上的本地部署。	*
aws.greengrass#GetLocalDeploymentStatus	允许组件获取核心设备上本地部署的状态。	本地部署 ID，或者*用于允许访问所有本地部署。
aws.greengrass#ListComponents	允许组件列出核心设备上的组件。	*
aws.greengrass#GetComponentDetails	允许组件获取有关核心设备上某个组件的详细信息。	组件名称，例如或com.example.HelloWorld ，*用于允许访问所有组件。
aws.greengrass#RestartComponent	允许组件在核心设备上重启组件。	组件名称，例如或com.example.HelloWorld ，*用于允许访问所有组件。

操作	描述	资源
aws.greengrass#StopComponent	允许组件停止核心设备上的组件。	组件名称，例如或com.example.HelloWorld ，*用于允许访问所有组件。
aws.greengrass#CreateDebugPassword	允许组件生成用于登录本地调试控制台组件的密码。	*

Example 授权策略示例

以下示例授权策略允许组件创建本地部署、查看所有本地部署和组件，以及重新启动和停止名为的组件com.example.HelloWorld。

```
{
  "accessControl": {
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:1": {
        "policyDescription": "Allows access to create local deployments and view
 deployments and components.",
        "operations": [
          "aws.greengrass#CreateLocalDeployment",
          "aws.greengrass#ListLocalDeployments",
          "aws.greengrass#GetLocalDeploymentStatus",
          "aws.greengrass#ListComponents",
          "aws.greengrass#GetComponentDetails"
        ],
        "resources": [
          "*"
        ]
      }
    },
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:2": {
        "policyDescription": "Allows access to restart and stop the Hello World
 component.",
        "operations": [
          "aws.greengrass#RestartComponent",
          "aws.greengrass#StopComponent"
        ],
        "resources": [
```

```
            "com.example.HelloWorld"
          ]
        }
      }
    }
  }
```

CreateLocalDeployment

使用指定的组件配方、构件和运行时参数创建或更新本地部署。

此操作提供的功能与 Greengrass CLI 中的部署创建命令相同。

请求

此操作的请求具有以下参数：

`recipeDirectoryPath`（Python:`recipe_directory_path`）

（可选）包含组件配方文件的文件夹的绝对路径。

`artifactDirectoryPath`（Python:`artifact_directory_path`）

（可选）包含要包含在部署中的对象文件的文件夹的绝对路径。构件文件夹必须包含以下文件夹结构：

```
/path/to/artifact/folder/component-name/component-version/artifacts
```

`rootComponentVersionsToAdd`（Python:`root_component_versions_to_add`）

（可选）要安装在核心设备上的组件版本。此对象是一个包含以下键值对的地图：`ComponentToVersionMap`

`key`

组件名称。

`value`

组件版本。

`rootComponentsToRemove`（Python:`root_components_to_remove`）

（可选）要从核心设备上卸载的组件。指定一个列表，其中每个条目都是组件的名称。

componentToConfiguration（Python:component_to_configuration）

（可选）部署中每个组件的配置更新。此对象是一个包含以下键值对的地图：ComponentToConfiguration

key

组件名称。

value

组件的配置更新 JSON 对象。JSON 对象必须采用以下格式。

```
{
  "MERGE": {
    "config-key": "config-value"
  },
  "RESET": [
    "path/to/reset/"
  ]
}
```

有关配置更新的更多信息，请参阅更新组件配置。

componentToRunWithInfo（Python:component_to_run_with_info）

（可选）部署中每个组件的运行时配置。此配置包括拥有每个组件进程的系统用户以及适用于每个组件的系统限制。此对象是一个包含以下键值对的地图：ComponentToRunWithInfo

key

组件名称。

value

组件的运行时配置。如果省略运行时配置参数，则 Core 软件将使用您在 Gre engr ass AWS IoT Greengrass 核心上配置的默认值。此对象包含以下信息：RunWithInfo

posixUser（Python:posix_user）

（可选）用于在 Linux 核心设备上运行此组件的 POSIX 系统用户和（可选）组。用户和组（如果已指定）必须存在于每台 Linux 核心设备上。使用以下格式指定由半角冒号（:）分隔的用户和组：user:group。组是可选的。如果您未指定群组，则 AWS IoT Greengrass Core 软件将使用该用户的主群组。有关更多信息，请参阅配置运行组件的用户。

windowsUser（Python:windows_user）

（可选）用于在 Windows 核心设备上运行此组件的 Windows 用户。用户必须存在于每台 Windows 核心设备上，其用户名和密码必须存储在 LocalSystem 账户的凭据管理器实例中。有关更多信息，请参阅配置运行组件的用户。

systemResourceLimits（Python:system_resource_limits）

（可选）适用于此组件进程的系统资源限制。您可以将系统资源限制应用于通用和非容器化 Lambda 组件。有关更多信息，请参阅为组件配置系统资源限制。

AWS IoT Greengrass目前不支持在 Windows 核心设备上使用此功能。

此对象包含以下信息：SystemResourceLimits

cpus

（可选）此组件的进程可以在核心设备上使用的最大 CPU 时间。核心设备的总 CPU 时间等于 CPU 核心的设备数量。例如，在具有 4 个 CPU 内核的核心设备上，您可以将此值设置为，2将该组件的进程使用率限制为每个 CPU 内核的 50%。在具有 1 个 CPU 内核的设备上，您可以将此值设置为，0.25将此组件的进程使用率限制在 25% 以内。如果将此值设置为大于 CPU 内核数的数字，则 AWS IoT Greengrass Core 软件不会限制组件的 CPU 使用率。

memory

（可选）此组件的进程可以在核心设备上使用的最大 RAM 量（以千字节为单位）。

groupName（Python:group_name）

（可选）此部署要定位的事物组的名称。

响应

此操作的响应包含以下信息：

deploymentId（Python:deployment_id）

请求创建的本地部署的 ID。

ListLocalDeployments

获取最近 10 个本地部署的状态。

此操作提供的功能与 Greengrass CLI 中的部署列表命令相同。

请求

此操作的请求没有任何参数。

响应

此操作的响应包含以下信息：

`localDeployments`（Python:`local_deployments`）

本地部署列表。此列表中的每个对象都是一个`LocalDeployment`对象，其中包含以下信息：

`deploymentId`（Python:`deployment_id`）

本地部署的 ID。

`status`

本地部署的状态。这个枚举`DeploymentStatus`具有以下值：

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

GetLocalDeploymentStatus

获取本地部署的状态。

此操作提供的功能与 Greengrass CLI 中的部署状态命令相同。

请求

此操作的请求具有以下参数：

`deploymentId`（Python:`deployment_id`）

要获取的本地部署的 ID。

响应

此操作的响应包含以下信息：

`deployment`

本地部署。此对象包含以下信息：`LocalDeployment`

`deploymentId`（Python:`deployment_id`）

本地部署的 ID。

`status`

本地部署的状态。这个枚举`DeploymentStatus`具有以下值：

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

ListComponents

获取核心设备上每个根组件的名称、版本、状态和配置。根组件是您在部署中指定的组件。此响应不包括作为其他组件的依赖项安装的组件。

此操作提供的功能与 Greengrass CLI 中的组件列表命令相同。

请求

此操作的请求没有任何参数。

响应

此操作的响应包含以下信息：

`components`

核心设备上的根组件列表。此列表中的每个对象都是一个`ComponentDetails`对象，其中包含以下信息：

`componentName`（Python:`component_name`）

组件名称。

`version`

组件版本。

`state`

组件的状态。此状态可以是以下状态之一：

- `BROKEN`
- `ERRORED`
- `FINISHED`
- `INSTALLED`
- `NEW`
- `RUNNING`
- `STARTING`
- `STOPPING`

`configuration`

组件的配置为 JSON 对象。

GetComponentDetails

获取核心设备上组件的版本、状态和配置。

此操作提供的功能与 Greengrass CLI 中的组件详细信息命令相同。

请求

此操作的请求具有以下参数：

`componentName`（Python:`component_name`）

要获取的组件的名称。

响应

此操作的响应包含以下信息：

`componentDetails`（Python:`component_details`）

组件的详细信息。此对象包含以下信息：`ComponentDetails`

`componentName`（Python:`component_name`）

组件名称。

`version`

组件版本。

`state`

组件的状态。此状态可以是以下状态之一：

- `BROKEN`
- `ERRORED`
- `FINISHED`
- `INSTALLED`
- `NEW`
- `RUNNING`
- `STARTING`
- `STOPPING`

`configuration`

组件的配置为 JSON 对象。

RestartComponent

在核心设备上重新启动组件。

虽然您可以重新启动任何组件，但我们建议您仅重新启动通用组件。

此操作提供的功能与 Greengrass CLI 中的组件重启命令相同。

请求

此操作的请求具有以下参数：

`componentName`（Python:`component_name`）

组件名称。

响应

此操作的响应包含以下信息：

`restartStatus`（Python:`restart_status`）

重启请求的状态。请求状态可以是以下状态之一：

- `SUCCEEDED`
- `FAILED`

`message`

一条消息，说明如果请求失败，则组件无法重新启动的原因。

StopComponent

停止核心设备上组件的进程。

Note

虽然您可以停止任何组件，但我们建议您只停止通用组件。

此操作提供的功能与 Greengrass CLI 中的组件停止命令相同。

请求

此操作的请求具有以下参数：

`componentName`（Python:`component_name`）

组件名称。

响应

此操作的响应包含以下信息：

`stopStatus`（Python:`stop_status`）

停止请求的状态。请求状态可以是以下状态之一：

- `SUCCEEDED`
- `FAILED`

`message`

一条消息，说明如果请求失败，组件为何无法停止。

CreateDebugPassword

生成一个随机密码，您可以使用该密码登录本地调试控制台组件。密码将在生成 8 小时后过期。

此操作提供的功能与 Greengrass CLI 中的get-debug-password 命令相同。

请求

此操作的请求没有任何参数。

响应

此操作的响应包含以下信息：

`username`

用于登录的用户名。

`password`

用于登录的密码。

`passwordExpiration`（Python:`password_expiration`）

密码的到期时间。

`certificateSHA256Hash`（Python:`certificate_sha256_hash`）

启用 HTTPS 时本地调试控制台使用的自签名证书的 SHA-256 指纹。打开本地调试控制台时，使用此指纹验证证书是否合法且连接安全。

`certificateSHA1Hash`（Python:`certificate_sha1_hash`）

启用 HTTPS 时本地调试控制台使用的自签名证书的 SHA-1 指纹。打开本地调试控制台时，使用此指纹验证证书是否合法且连接安全。

对客户端设备进行身份验证和授权

Note

此功能适用于 Greengrass nucleus 组件的 2.6.0 及更高版本。

使用客户端设备身份验证 IPC 服务开发自定义本地代理组件，本地物联网设备（例如客户端设备）可以在其中进行连接。

要使用这些 IPC 操作，请将客户端设备身份验证组件的 2.2.0 或更高版本作为依赖项包含在您的自定义组件中。然后，您可以在自定义组件中使用 IPC 操作来执行以下操作：

- 验证连接到核心设备的客户端设备的身份。
- 为客户端设备创建连接核心设备的会话。
- 验证客户端设备是否有权执行某项操作。
- 当核心设备的服务器证书轮换时，会收到通知。

主题

- SDK 的最低版本
- 授权
- VerifyClientDeviceIdentity
- GetClientDeviceAuthToken
- AuthorizeClientDeviceAction
- SubscribeToCertificateUpdates

SDK 的最低版本

下表列出了与客户端设备身份验证 IPC 服务交互时必须使用的最低版本。AWS IoT Device SDK

SDK	最低版本	
AWS IoT Device SDK适用于 Java v2	v1.9.3	
AWS IoT Device SDK适用于 Python v2	v1.11.3	
AWS IoT Device SDK适用于 C ++ v2	v1.18.3	
AWS IoT Device SDK适用于 JavaScript v2	v1.12.0	

授权

要在自定义组件中使用客户端设备身份验证 IPC 服务，必须定义允许您的组件执行这些操作的授权策略。有关定义授权策略的信息，请参阅授权组件执行 IPC 操作。

客户端设备身份验证和授权的授权策略具有以下属性。

IPC 服务标识符：`aws.greengrass.clientdevices.Auth`

操作	描述	资源
`aws.greengrass#VerifyClientDeviceIdentity`	允许组件验证客户端设备的身份。	*
`aws.greengrass#GetClientDeviceAuthToken`	允许组件验证客户端设备的凭据并为该客户端设备创建会话。	*
`aws.greengrass#AuthorizeClientDeviceAction`	允许组件验证客户端设备是否有权执行某项操作。	*

操作	描述	资源
aws.greengrass#SubscribeToCertificateUpdates	允许组件在核心设备的服务器证书轮换时接收通知。	*
*	允许组件执行所有客户端设备身份验证 IPC 服务操作。	*

授权策略示例

您可以参考以下授权策略示例，以帮助您为组件配置授权策略。

Example 授权策略示例

以下示例授权策略允许组件执行所有客户端设备身份验证 IPC 操作。

```
{
  "accessControl": {
    "aws.greengrass.clientdevices.Auth": {
      "com.example.MyLocalBrokerComponent:clientdevices:1": {
        "policyDescription": "Allows access to authenticate and authorize client
 devices.",
        "operations": [
          "aws.greengrass#VerifyClientDeviceIdentity",
          "aws.greengrass#GetClientDeviceAuthToken",
          "aws.greengrass#AuthorizeClientDeviceAction",
          "aws.greengrass#SubscribeToCertificateUpdates"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

VerifyClientDeviceIdentity

验证客户端设备的身份。此操作验证客户端设备是否有效AWS IoT。

请求

此操作的请求具有以下参数：

`credential`

客户端设备的凭证。此对象包含以下信息：`ClientDeviceCredential`

`clientDeviceCertificate`（Python:`client_device_certificate`）

客户端设备的 X.509 设备证书。

响应

此操作的响应包含以下信息：

`isValidClientDevice`（Python:`is_valid_client_device`）

客户端设备的身份是否有效。

GetClientDeviceAuthToken

验证客户端设备的凭证并为该客户端设备创建会话。此操作会返回一个会话令牌，您可以在后续请求中使用该令牌来授权客户端设备操作。

要成功连接客户端设备，客户端设备身份验证组件必须为该客户端设备使用的客户端 ID 授予`mqtt:connect`权限。

请求

此操作的请求具有以下参数：

`credential`

客户端设备的凭证。此对象包含以下信息：`CredentialDocument`

`mqttCredential`（Python:`mqtt_credential`）

客户端设备的 MQTT 凭证。指定客户端设备用于连接的客户端 ID 和证书。此对象包含以下信息：`MQTTCredential`

`clientId`（Python:`client_id`）

用于连接的客户端 ID。

`certificatePem`（Python:`certificate_pem`）

用于连接的 X.509 设备证书。

`username`

Note

目前未使用此属性。

`password`

Note

目前未使用此属性。

响应

此操作的响应包含以下信息：

`clientDeviceAuthToken`（Python:`client_device_auth_token`）

客户端设备的会话令牌。您可以在后续请求中使用此会话令牌来授权此客户端设备的操作。

AuthorizeClientDeviceAction

验证客户端设备是否有权对资源执行操作。客户端设备授权策略指定客户端设备在连接到核心设备时可以执行的权限。在配置客户端设备身份验证组件时，您可以定义客户端设备授权策略。

请求

此操作的请求具有以下参数：

`clientDeviceAuthToken`（Python:`client_device_auth_token`）

客户端设备的会话令牌。

`operation`

要授权的操作。

`resource`

客户端设备执行操作的资源。

响应

此操作的响应包含以下信息：

`isAuthorized`（Python:`is_authorized`）

客户端设备是否有权对资源执行操作。

SubscribeToCertificateUpdates

订阅即可在每次轮换时接收核心设备的新服务器证书。服务器证书轮换时，代理必须使用新的服务器证书重新加载。

默认情况下，客户端设备身份验证组件每 7 天轮换一次服务器证书。您可以将轮换间隔配置为 2 到 10 天之间。

此操作是一种订阅操作，您可以在其中订阅事件消息流。要使用此操作，请定义一个流响应处理程序，其中包含处理事件消息、错误和流关闭的函数。有关更多信息，请参阅 订阅 IPC 事件直播。

事件消息类型：`CertificateUpdateEvent`

请求

此操作的请求具有以下参数：

`certificateOptions`（Python:`certificate_options`）

要订阅的证书更新的类型。此对象包含以下信息：`CertificateOptions`

`certificateType`（Python:`certificate_type`）

要订阅的更新的证书类型。选择以下选项：

- `SERVER`

响应

此操作的响应包含以下信息：

`messages`

消息流。此对象包含以下信息：`CertificateUpdateEvent`

`certificateUpdate`（Python:`certificate_update`）

有关新证书的信息。此对象包含以下信息：`CertificateUpdate`

`certificate`

证书。

`privateKey`（Python:`private_key`）

证书的私钥。

`publicKey`（Python:`public_key`）

证书的公钥。

`caCertificates`（Python:`ca_certificates`）

证书的 CA 证书链中的证书颁发机构 (CA) 证书列表。

与本地物联网设备互动

客户端设备是通过 MQTT 连接到 Greengrass 核心设备并与之通信的本地物联网设备。您可以将客户端设备连接到核心设备以执行以下操作：

- 在 Greengrass 组件中与 MQTT 消息进行交互。
- 在客户端设备和之间中继消息和数据AWS IoT Core。
- 在 Greengrass 组件中与客户端设备阴影进行交互。
- 将客户端设备的阴影与同步AWS IoT Core。

要连接到核心设备，客户端设备可以使用云发现。客户端设备连接到AWS IoT Greengrass云服务以检索有关它们可以连接的核心设备的信息。然后，他们可以连接到核心设备来处理消息并将数据与AWS IoT Core云服务同步。

你可以按照一个教程来介绍如何配置核心设备以与AWS IoT事物进行连接和通信。本教程还探讨了如何开发可与客户端设备交互的自定义 Greengrass 组件。有关更多信息，请参阅 教程：通过 MQTT 与本地物联网设备交互。

主题

- AWS-提供的客户端设备组件
- Connect 客户端设备与核心设备连接
- 在客户端设备之间中继 MQTT 消息和 AWS IoT Core
- 在组件中与客户端设备交互
- 与客户端设备影子进行交互并进行同步
- 对客户端设备进行故障排除

AWS-提供的客户端设备组件

AWS IoT Greengrass提供了以下公共组件，您可以将其部署到核心设备。这些组件使客户端设备能够与核心设备连接和通信。

Note

AWS提供的几个组件依赖于 Greengrass 核的特定次要版本。由于这种依赖关系，当你将 Greengrass nucleus 更新到新的次要版本时，你需要更新这些组件。有关每个组件所依赖的

原子核的特定版本的信息，请参阅相应的组件主题。有关更新原子核的更多信息，请参见更新 AWS IoT Greengrass核心软件 (OTA)。

当组件的组件类型同时为泛型和 Lambda 时，该组件的当前版本为泛型类型，而该组件的先前版本为 Lambda 类型。

组件	描述	组件类型	支持的操作系统	开源
客户端设备身份验证	使本地 IoT 设备（称为客户端设备）能够连接到核心设备。	插件	Linux、Windows	是
IP 探测器	向报告 MQTT 代理连接信息AWS IoT Greengrass，以便客户端设备可以发现如何连接。	插件	Linux、Windows	是
MQTT 网桥	在客户端设备、本地AWS IoT Greengrass发布/订阅和之间中继 MQTT 消息。AWS IoT Core	插件	Linux、Windows	是
MQTT 3.1.1 经纪商 (Moquette)	运行处理客户端设备和核心设备之间消息	插件	Linux、Windows	是

组件	描述	组件类型	支持的操作系统	开源
	的 MQTT 3.1.1 代理。			
MQTT 5 经纪商 (EMQX)	运行处理客户端设备和核心设备之间消息的 MQTT 5 代理。	通用	Linux、Windows	否
影子经理	启用与核心设备上的阴影交互。它管理影子文档存储以及本地卷影状态与 Dev AWS IoT ice Shadow 服务的同步。	插件	Linux、Windows	是

Connect 客户端设备与核心设备连接

您可以将云发现配置为将客户端设备连接到核心设备。配置云发现时，客户端设备可以连接到AWS IoT Greengrass云服务，以检索有关其可以连接的核心设备的信息。然后，客户端设备可以尝试连接到每台核心设备，直到它们成功连接。

要使用云发现，您必须执行以下操作：

- 将客户端设备与它们可以连接的核心设备相关联。
- 指定客户端设备可以连接到每个核心设备的 MQTT 代理端点。
- 将组件部署到支持客户端设备的核心设备。

  您也可以部署可选组件来执行以下操作：

  - 在客户端设备、Greengrass 组件和云服务之间中继消息。AWS IoT Core
  - 自动为您管理核心设备 MQTT 代理端点。

- 管理本地客户端设备阴影并将阴影与AWS IoT Core云服务同步。

您还必须查看并更新核心设备的AWS IoT策略，以验证其是否具有连接客户端设备所需的权限。有关更多信息，请参阅 要求。

配置云发现后，您可以测试客户端设备和核心设备之间的通信。有关更多信息，请参阅 测试客户端设备通信。

主题

- 要求
- 用于支持客户端设备的 Greengrass 组件
- 配置云发现（控制台）
- 配置云发现 (AWS CLI)
- 关联客户端设备
- 离线时对客户端进行身份验证
- 管理核心设备端点
- 选择一个 MQTT 经纪商
- 使用 MQTT 代理将客户端设备连接到AWS IoT Greengrass核心设备
- 测试客户端设备通信
- Greengrass 发现 RESTful API

要求

要将客户端设备连接到核心设备，必须具备以下条件：

- 核心设备必须运行 Greengrass nucleus v2.0 或更高版本。
- 在核心设备运行的区域，AWS IoT Greengrass与AWS 账户您关联的 Greengrass 服务角色。AWS 有关更多信息，请参阅 配置 Greengrass 服务角色。
- 核心设备的AWS IoT策略必须允许以下权限：
  - `greengrass:PutCertificateAuthorities`
  - `greengrass:VerifyClientDeviceIdentity`
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
  - `greengrass:GetConnectivityInfo`

- greengrass:UpdateConnectivityInfo—（可选）使用 IP 检测器组件需要此权限，该组件将核心设备的网络连接信息报告给AWS IoT Greengrass云服务。
- iot:GetThingShadowiot:UpdateThingShadow、和 iot:DeleteThingShadow —（可选）使用影子管理器组件与客户端设备影子同步需要这些权限AWS IoT Core。此功能需要 Greengrass nucleus v2.6.0 或更高版本、影子管理器 v2.2.0 或更高版本以及 MQTT bridge v2.2.0 或更高版本。

有关更多信息，请参阅 配置AWS IoT事物策略。

> **Note**
>
> 如果您在安装AWS IoT Greengrass酷睿软件时使用默认AWS IoT策略，则核心设备具有允许访问所有AWS IoT Greengrass操作的AWS IoT策略（greengrass:*）。

- AWS IoT可以作为客户端设备连接的东西。有关更多信息，请参阅《AWS IoT Core开发人员指南》中的创建AWS IoT资源。
- 客户端设备必须使用客户端 ID 进行连接。客户端 ID 是一个事物名称。不接受其他客户户ID。
- 每台客户端设备的AWS IoT策略都必须允许该greengrass:Discover权限。有关更多信息，请参阅 客户端设备的最低AWS IoT政策。

主题

- 配置 Greengrass 服务角色
- 配置AWS IoT事物策略

配置 Greengrass 服务角色

Greengrass 服务角色是一种 AWS Identity and Access Management (IAM) 服务角色，用于向 AWS IoT Greengrass 授权代表您访问 AWS 服务中的资源。此角色AWS IoT Greengrass使验证客户端设备的身份和管理核心设备的连接信息成为可能。

如果您之前未在该区域设置 Greengrass 服务角色，则必须将该区域的 Green grass 服务角色与您的服务角色相关联。AWS IoT Greengrass AWS 账户

在AWS IoT Greengrass控制台中使用配置核心设备发现页面时，请为您AWS IoT Greengrass设置 Greengrass 服务角色。否则，您可以使用AWS IoT控制台或 AWS IoT Greengrass API 进行手动设置。

在本节中，您将检查 Greengrass 服务角色是否已设置。如果未设置，则可以在该区域为您创建一个新的 Greengrass 服务角色AWS IoT Greengrass进行关联。AWS 账户

配置 Greengrass 服务角色（控制台）

1. 检查该区域中是否与您的 Greengrass 服务角色相关AWS IoT Greengrass联。AWS 账户执行以下操作：

    a. 导航到 AWS IoT 控制台。

    b. 在导航窗格中，选择设置。

    c. 在 Greengrass 服务角色部分，找到当前服务角色以查看是否关联了 Greengrass 服务角色。

       如果您关联了 Greengrass 服务角色，则可以满足使用 IP 检测器组件的要求。跳至配置AWS IoT事物策略。

2. 如果该区域中没有AWS IoT Greengrass与您的 Greengrass 服务角色关联，AWS 账户请创建一个 Greengrass 服务角色并将其关联。执行以下操作：

    a. 导航到 IAM 控制台。

    b. 选择角色。

    c. 选择创建角色。

    d. 在创建角色页面上，执行以下操作：

        i. 在 “可信实体类型” 下，选择AWS 服务。

        ii. 在 “用例” 下，选择 “其他用例”，选择 GreengrassAWS 服务，选择 Greengrass。此选项指定添加为可以AWS IoT Greengrass担任此角色的可信实体。

        iii. 请选择 Next（下一步）。

        iv. 在 “权限策略” 下，选择AWSGreengrassResourceAccessRolePolicy要附加到角色的。

        v. 请选择 Next（下一步）。

        vi. 在角色名称中，输入角色的名称，例如**Greengrass_ServiceRole**。

        vii. 选择创建角色。

    e. 导航到 AWS IoT 控制台。

    f. 在导航窗格中，选择设置。

    g. 在 Greengrass 服务角色部分，选择附加角色。

    h. 在更新 Greengrass 服务角色模式中，选择您创建的 IAM 角色，然后选择附加角色。

配置 Greengrass 服务角色 () AWS CLI

1. 检查该区域中是否与您的 Greengrass 服务角色相关AWS IoT Greengrass联。AWS 账户

```
aws greengrassv2 get-service-role-for-account
```

如果关联了 Greengrass 服务角色，则该操作会返回包含有关该角色的信息的响应。

如果您关联了 Greengrass 服务角色，则可以满足使用 IP 检测器组件的要求。跳至配置AWS IoT 事物策略。

2. 如果该区域中没有AWS IoT Greengrass与您的 Greengrass 服务角色关联，AWS 账户请创建一个 Greengrass 服务角色并将其关联。执行以下操作：

a. 使用允许 AWS IoT Greengrass 代入该角色的信任策略创建角色。此示例将创建一个名为 `Greengrass_ServiceRole` 的角色，但您也可以使用其他名称。我们建议您在信任策略中加入 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键，以帮助防止出现混淆代理人安全问题。条件上下文键可限制访问权限，仅允许来自指定账户和 Greengrass 工作空间的请求。有关混淆代理人问题的更多信息，请参阅 防止跨服务混淆代理。

Linux or Unix

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
```

```
}'
```

Windows Command Prompt (CMD)

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document "{\\"Version\\":\\"2012-10-17\\",\\"Statement\\":[{\\"Effect\
\":\\"Allow\\",\\"Principal\\":{\\"Service\\":\\"greengrass.amazonaws.com
\\"},\\"Action\\":\\"sts:AssumeRole\\",\\"Condition\\":{\\"ArnLike\\":
{\\"aws:SourceArn\\":\\"arn:aws:greengrass:region:account-id:*\\"},\
\"StringEquals\\":{\\"aws:SourceAccount\\":\\"account-id\\"}}}]}"
```

PowerShell

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

b. 从输出中的角色元数据复制角色 ARN。使用该 ARN 将角色与您的账户关联。

c. 将 `AWSGreengrassResourceAccessRolePolicy` 策略附加到该角色。

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

d. 将 Greengrass 服务角色与您的服务角色相关联。AWS IoT Greengrass AWS 账户将 *role-arn* 替换为服务角色的 ARN。

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn
```

如果操作成功，则返回以下响应。

```
{
  "associatedAt": "timestamp"
}
```

配置AWS IoT事物策略

核心设备使用 X.509 设备证书来授权连接。AWS您可以将AWS IoT策略附加到设备证书以定义核心设备的权限。有关更多信息，请参阅 数据层面操作的 AWS IoT 策略和 支持客户端设备的最低AWS IoT 政策。

要将客户端设备连接到核心设备，核心设备的AWS IoT策略必须允许以下权限：

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`—（可选）使用 IP 检测器组件需要此权限，该组件将核心设备的网络连接信息报告给AWS IoT Greengrass云服务。
- `iot:GetThingShadowiot:UpdateThingShadow`、和 `iot:DeleteThingShadow` —（可选）使用影子管理器组件与客户端设备影子同步需要这些权限AWS IoT Core。此功能需要 Greengrass nucleus v2.6.0 或更高版本、影子管理器 v2.2.0 或更高版本以及 MQTT bridge v2.2.0 或更高版本。

在本节中，您将查看核心设备的AWS IoT策略并添加缺少的所有必需权限。如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则您的核心设备具有允许访问所有AWS IoT Greengrass操作的AWS IoT策略（`greengrass:*`）。在这种情况下，只有当您计划部署影子管理器组件来同步设备影子时，才必须更新AWS IoT策略AWS IoT Core。否则，你可以跳过本节。

配置AWS IoT事物策略（控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择核心设备。
2. 在核心设备页面上，选择要更新的核心设备。
3. 在核心设备详细信息页面上，选择指向核心设备的 Thin g 的链接。此链接可在AWS IoT控制台中打开事物详细信息页面。
4. 在事物详细信息页面上，选择证书。
5. 在 “证书” 选项卡中，选择事物的有效证书。
6. 在证书详细信息页面上，选择策略。
7. 在 “策略” 选项卡中，选择要查看和更新的AWS IoT策略。您可以向附加到核心设备活动证书的任何策略添加所需的权限。

   > **Note**
   >
   > 如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则有两个AWS IoT策略。我们建议您选择名为GreengrassV2IoTThingPolicy的策略（如果存在）。默认情况下，使用快速安装程序创建的核心设备使用此策略名称。如果您向此策略添加权限，则也将这些权限授予使用此策略的其他核心设备。

8. 在策略概述中，选择编辑活动版本。
9. 查看策略以获取所需权限，然后添加缺少的所有必需权限。
   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
   - `greengrass:GetConnectivityInfo`
   - `greengrass:UpdateConnectivityInfo`—（可选）使用 IP 检测器组件需要此权限，该组件将核心设备的网络连接信息报告给AWS IoT Greengrass云服务。
   - `iot:GetThingShadowiot:UpdateThingShadow`、和 `iot:DeleteThingShadow` —（可选）使用影子管理器组件与客户端设备影子同步需要这些权限AWS IoT Core。此功能需要 Greengrass nucleus v2.6.0 或更高版本、影子管理器 v2.2.0 或更高版本以及 MQTT bridge v2.2.0 或更高版本。
10. （可选）要允许核心设备与其同步阴影AWS IoT Core，请在策略中添加以下声明。如果您计划与客户端设备影子进行交互，但不打算与之同步AWS IoT Core，请跳过此步骤。将*##*和*## ID* 替换为您使用的地区和您的AWS 账户号码。

- 此示例语句允许访问所有事物的设备影子。要遵循最佳安全实践，您可以限制仅访问核心设备和连接到核心设备的客户端设备。有关更多信息，请参阅 支持客户端设备的最低AWS IoT政策。

```
{
  "Effect": "Allow",
  "Action": [
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
    "iot:DeleteThingShadow"
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/*"
  ]
}
```

添加此语句后，策略文档可能与以下示例类似。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "greengrass:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:thing/*"
      ]
```

```
    }
  ]
}
```

11. 要将新的策略版本设置为活动版本，请在策略版本状态下，选择将编辑后的版本设置为该策略的活动版本。
12. 选择另存为新版本。

配置AWS IoT事物策略 (AWS CLI)

1. 列出核心设备的原理AWS IoT。事物主体可以是 X.509 设备证书或其他标识。运行以下命令，并*MyGreengrassCore*替换为核心设备的名称。

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   该操作返回一个响应，其中列出了核心设备的事物主体。

   ```
   {
       "principals": [
           "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
       ]
   }
   ```

2. 识别核心设备的活动证书。运行以下命令，将 cert *i* ficateID 替换为上一步中每个证书的 ID，直到找到活动证书。证书 ID 是证书 ARN 末尾的十六进制字符串。该`--query`参数指定仅输出证书的状态。

   ```
   aws iot describe-certificate --certificate-id certificateId --query
    'certificateDescription.status'
   ```

   该操作以字符串形式返回证书状态。例如，如果证书处于活动状态，则此操作会输出`"ACTIVE"`。
3. 列出附加到证书的AWS IoT策略。运行以下命令，并将证书 ARN 替换为证书的 ARN。

   ```
   aws iot list-principal-policies --principal arn:aws:iot:us-
   west-2:123456789012:cert/certificateId
   ```

   该操作返回一个响应，其中列出了附加到证书的AWS IoT策略。

   ```
   {
   ```

```
    "policies": [
        {
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
}
```

4. 选择要查看和更新的策略。

> **Note**
>
> 如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则有两个AWS IoT策略。我们建议您选择名为GreengrassV2IoTThingPolicy的策略（如果存在）。默认情况下，使用快速安装程序创建的核心设备使用此策略名称。如果您向此策略添加权限，则也将这些权限授予使用此策略的其他核心设备。

5. 获取保单文件。运行以下命令，将 *GreenGrassv2IoT #####ThingPolicy###*。

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

该操作会返回一个响应，其中包含策略的文档和有关该策略的其他信息。策略文档是一个序列化为字符串的 JSON 对象。

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
```

```
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. 使用在线转换器或其他工具将策略文档字符串转换为 JSON 对象，然后将其保存到名为的文件中`iot-policy.json`。

   例如，如果您安装了 jq 工具，则可以运行以下命令来获取策略文档，将其转换为 JSON 对象，然后将策略文档另存为 JSON 对象。

   ```
   aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
    'policyDocument' | jq fromjson >> iot-policy.json
   ```

7. 查看策略以获取所需权限，然后添加缺少的所有必需权限。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 打开文件。

   ```
   nano iot-policy.json
   ```

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
   - `greengrass:GetConnectivityInfo`
   - `greengrass:UpdateConnectivityInfo`—（可选）使用 IP 检测器组件需要此权限，该组件将核心设备的网络连接信息报告给AWS IoT Greengrass云服务。
   - `iot:GetThingShadowiot:UpdateThingShadow`、和 `iot:DeleteThingShadow` —（可选）使用影子管理器组件与客户端设备影子同步需要这些权限AWS IoT Core。此功能需

要 Greengrass nucleus v2.6.0 或更高版本、影子管理器 v2.2.0 或更高版本以及 MQTT bridge v2.2.0 或更高版本。

8. 将更改另存为新版本的策略。运行以下命令，将 *GreenGrassv2IoT #####ThingPolicy###*。

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
document file://iot-policy.json --set-as-default
```

如果操作成功，则返回类似于以下示例的响应。

```
{
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
\\t\\t\\"iot:Connect\\",\
\\t\\t\\"iot:Publish\\",\
\\t\\t\\"iot:Subscribe\\",\
\\t\\t\\"iot:Receive\\",\
\\t\\t\\"greengrass:*\\"\
      ],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

用于支持客户端设备的 Greengrass 组件

Important

核心设备必须运行 Greengrass nucleus v2.0 或更高版本才能支持客户端设备。

要使客户端设备能够与核心设备连接和通信，您需要将以下 Greengrass 组件部署到核心设备：

- 客户端设备身份验证 (aws.greengrass.clientdevices.Auth)

  部署客户端设备身份验证组件以对客户端设备进行身份验证并授权客户端设备操作。这个组件允许你的AWS IoT东西连接到核心设备。

  此组件需要一些配置才能使用。您必须指定客户端设备组以及每个组有权执行的操作，例如通过 MQTT 进行连接和通信。有关更多信息，请参阅客户端设备身份验证组件配置。
- MQTT 3.1.1 经纪商 (Moquette) (aws.greengrass.clientdevices.mqtt.Moquette)

  部署 Moquette MQTT 代理组件来运行轻量级 MQTT 代理。Moquette MQTT 代理符合 MQTT 3.1.1，包括对 QoS 0、QoS 1、QoS 2、保留消息、最后遗嘱消息和永久订阅的本地支持。

  您无需配置此组件即可使用它。但是，您可以配置此组件运行 MQTT 代理的端口。默认情况下，它使用端口 8883。
- MQTT 5 经纪商 (EMQX) (aws.greengrass.clientdevices.mqtt.EMQX)

  > Note
  >
  > 要使用 EMQX MQTT 5 代理，必须使用 Greengrass n ucleus v2.6.0 或更高版本以及客户端设备身份验证 v2. 2.0 或更高版本。

  部署 EMQX MQTT 代理组件，以便在客户端设备和核心设备之间的通信中使用 MQTT 5.0 功能。EMQX MQTT 代理符合 MQTT 5.0，包括对会话和消息过期间隔、用户属性、共享订阅、主题别名等的支持。

  您无需配置此组件即可使用它。但是，您可以配置此组件运行 MQTT 代理的端口。默认情况下，它使用端口 8883。
- MQTT 网桥 (aws.greengrass.clientdevices.mqtt.Bridge)

  （可选）部署 MQTT 桥接组件，以便在客户端设备（本地 MQTT）、本地发布/订阅和 MQTT 之间中继消息。AWS IoT Core将此组件配置为与 Greengrass 组件中的客户端设备同步AWS IoT Core并与客户端设备进行交互。

  此组件需要配置才能使用。您必须指定此组件中继消息的主题映射。有关更多信息，请参阅 MQTT 网桥组件配置。
- IP 探测器 (aws.greengrass.clientdevices.IPDetector)

（可选）部署 IP 检测器组件，自动向AWS IoT Greengrass云服务报告核心设备的 MQTT 代理端点。如果您的网络设置很复杂，例如路由器将 MQTT 代理端口转发到核心设备的网络设置，则无法使用此组件。

您无需配置此组件即可使用它。

- 影子经理 (aws.greengrass.ShadowManager)

> **Note**
>
> 要管理客户端设备影子，必须使用 Greengrass nucleus v2.6.0 或更高版本、影子管理器 v2.2.0 或更高版本以及 MQTT bridge v2.2.0 或更高版本。

（可选）部署影子管理器组件以管理核心设备上的客户端设备影子。Greengrass 组件可以获取、更新和删除客户端设备影子，以便与客户端设备进行交互。您也可以将影子管理器组件配置为将客户端设备影子与AWS IoT Core云服务同步。

要将此组件用于客户端设备影子，必须将 MQTT 桥接组件配置为在客户端设备和影子管理器之间中继消息，后者使用本地发布/订阅。否则，此组件不需要配置即可使用，但需要配置才能同步设备影子。

> **Note**
>
> 我们建议您仅部署一个 MQTT 代理组件。MQTT 网桥和 IP 检测器组件一次只能使用一个 MQTT 代理组件。如果您部署多个 MQTT 代理组件，则必须将它们配置为使用不同的端口。

配置云发现（控制台）

您可以使用AWS IoT Greengrass控制台关联客户端设备、管理核心设备端点和部署组件以启用客户端设备支持。有关更多信息，请参阅 步骤 2：启用客户端设备支持。

配置云发现 (AWS CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 关联客户端设备、管理核心设备端点和部署组件以启用客户端设备支持。有关更多信息，请参阅下列内容：

- 管理客户端设备关联 (AWS CLI)
- 管理核心设备端点
- AWS-提供的客户端设备组件
- 创建部署

关联客户端设备

要使用云发现，请将客户端设备与核心设备关联，以便它们可以发现核心设备。然后，他们可以使用 Greengrass 发现 API 来检索关联核心设备的连接信息和证书。

同样，取消客户端设备与核心设备的关联以阻止它们发现核心设备。

主题

- 管理客户端设备关联（控制台）
- 管理客户端设备关联 (AWS CLI)
- 管理客户端设备关联 (API)

管理客户端设备关联（控制台）

您可以使用AWS IoT Greengrass控制台查看、添加和删除客户端设备关联。

查看核心设备的客户端设备关联（控制台）

1. 导航到 AWS IoT Greengrass 控制台。
2. 选择核心设备。
3. 选择托管管理的核心设备。
4. 在核心设备的详细信息页面上，请选择客户端设备选项卡。
5. 在 “关联的客户端设备” 部分，您可以看到哪些客户端设备（AWS IoT事物）与核心设备关联。

将客户端设备与核心设备关联（控制台）

1. 导航到 AWS IoT Greengrass 控制台。
2. 选择核心设备。
3. 选择托管管理的核心设备。
4. 在核心设备的详细信息页面上，请选择客户端设备选项卡。

5. 在关联客户端设备部分，选择关联客户端设备。
6. 在将客户端设备与核心设备关联模式中，对要关联的每台客户端设备执行以下操作：

   a. 输入要关联为客户端设备AWS IoT的事物的名称。
   b. 选择添加。
7. 选择关联。

   您关联的客户端设备现在可使用 Greengrass 发现 API 来发现此核心设备。

取消客户端设备与核心设备的关联（控制台）

1. 导航到 AWS IoT Greengrass 控制台。
2. 选择核心设备。
3. 选择托管管理的核心设备。
4. 在核心设备的详细信息页面上，请选择客户端设备选项卡。
5. 在 “关联的客户端设备” 部分，选择要取消关联的每台客户端设备。
6. 选择取消关联。
7. 在确认模式中，选择取消关联。

   您取消关联的客户端设备无法再使用 Greengrass 发现 API 来发现此核心设备。

管理客户端设备关联 (AWS CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 来管理核心设备的客户端设备关联。

查看核心设备的客户端设备关联 (AWS CLI)

- 使用以下命令：list-client-devices-associated-with-core-device。

将客户端设备与核心设备关联 (AWS CLI)

- 使用以下命令：batch-associate-client-device-with-core-device。

取消客户端设备与核心设备的关联 () AWS CLI

- 使用以下命令：batch-disassociate-client-device-from-core-device。

管理客户端设备关联 (API)

您可以使用 AWS API 来管理核心设备的客户端设备关联。

查看核心设备的客户端设备关联 (AWSAPI)

- 使用以下操作：ListClientDevicesAssociatedWithCoreDevice。

将客户端设备与核心设备关联 (AWSAPI)

- 使用以下操作：BatchAssociateClientDeviceWithCoreDevice。

取消客户端设备与核心设备的关联 (AWSAPI)

- 使用以下操作：BatchDisassociateClientDeviceFromCoreDevice。

离线时对客户端进行身份验证

通过离线身份验证，您可以配置您的 AWS IoT Greengrass 核心设备，以便即使核心设备未连接到云端，客户端设备也可以连接到核心设备。当您使用离线身份验证时，您的 Greengrass 设备可以在部分离线环境中继续工作。

要对连接到云端的客户端设备使用离线身份验证，您需要满足以下条件：

- 部署了客户端设备身份验证组件的 AWS IoT Greengrass 核心设备。必须使用 2.3.0 或更高版本进行离线身份验证。
- 在客户端设备初始连接期间，核心设备的云连接。

存储客户凭证

当客户端设备首次连接到核心设备时，核心设备会调用该 AWS IoT Greengrass 服务。当被调用时，Greengrass 会将客户端设备的注册作为事物进行验证。 AWS IoT 它还会验证设备是否具有有效的证书。然后，核心设备将这些信息存储在本地。

设备下次连接时，Greengrass 核心设备会尝试使用服务验证客户端设备。 AWS IoT Greengrass 如果无法连接 AWS IoT Greengrass，则核心设备将使用其本地存储的设备信息来验证客户端设备。

您可以配置 Greengrass 核心设备存储凭据的时间长度。您可以通过在客户端设备身份验证组件配置中设置`clientDeviceTrustDurationMinutes`配置选项，将超时时间从一分钟设置为 2,147,483,647

分钟。默认值为一分钟，这实际上会关闭离线身份验证。在设置此超时时，我们建议您考虑您的安全需求。您还应该考虑核心设备在与云端断开连接时预计能运行多长时间。

核心设备会三次更新其凭据存储：

1. 当设备首次连接到核心设备时。
2. 如果核心设备已连接到云端，则当客户端设备重新连接到核心设备时。
3. 如果核心设备已连接到云端，则每天刷新一次整个凭据存储。

当 Greengrass 核心设备刷新其凭据存储区时，它会使用该操作。ListClientDevicesAssociatedWithCoreDeviceGreengrass 仅刷新此操作返回的设备。要将客户端设备与核心设备关联，请参阅关联客户端设备。

要使用该`ListClientDevicesAssociatedWithCoreDevice`操作，您必须向与运行的关联的 AWS Identity and Access Management (IAM) 角色添加操作权限 AWS IoT Greengrass。 AWS 账户有关更多信息，请参阅 授权核心设备与AWS服务。

管理核心设备端点

当您使用云发现时，您可以将核心设备的 MQTT 代理端点存储在AWS IoT Greengrass云服务中。客户端设备连接到AWS IoT Greengrass以检索这些端点及其关联核心设备的其他信息。

对于每台核心设备，您可以自动或手动管理端点。

- 使用 IP 探测器自动管理端点

  如果您的网络设置不复杂，例如客户端设备与核心设备位于同一网络上，则可以部署 IP 检测器组件来自动为您管理核心设备端点。例如，如果核心设备位于将 MQTT 代理端口转发到核心设备的路由器后面，则无法使用 IP 检测器组件。

  如果您部署到事物组，IP 检测器组件也很有用，因为它可以管理事物组中所有核心设备的端点。有关更多信息，请参阅 使用 IP 探测器自动管理端点。

- 手动管理端点

  如果您无法使用 IP 检测器组件，则必须手动管理核心设备端点。您可以使用控制台或 API 更新这些终端节点。有关更多信息，请参阅 手动管理端点。

主题

- 使用 IP 探测器自动管理端点
- 手动管理端点

使用 IP 探测器自动管理端点

如果您的网络设置很简单，例如客户端设备与核心设备位于同一网络上，则可以部署 IP 检测器组件来执行以下操作：

- 监控 Greengrass 核心设备的本地网络连接信息。此信息包括核心设备的网络端点和 MQTT 代理运行的端口。
- 向AWS IoT Greengrass云服务报告核心设备的连接信息。

IP 检测器组件会覆盖您手动设置的端点。

> **Important**
>
> 核心设备的AWS IoT策略必须允许使用 IP 检测器组件的`greengrass:UpdateConnectivityInfo`权限。有关更多信息，请参阅 数据层面操作的 AWS IoT 策略和 配置AWS IoT事物策略。

您可以执行以下任一操作来部署 IP 检测器组件：

- 使用控制台中的配置发现页面。有关更多信息，请参阅 配置云发现（控制台）。
- 创建和修改部署以包括 IP 检测器。您可以使用控制台或 AWS API 来管理部署。AWS CLI有关更多信息，请参阅 创建部署。

部署 IP 检测器组件（控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择组件。
2. 在 “组件” 页上，选择 “公共组件” 选项卡，然后选择aws.greengrass.clientdevices.IPDetector。
3. 在 aws.greengrass.clientdevices.IPDetector 页面上，选择部署。
4. 从 “添加到部署” 中，选择要修改的现有部署，或选择创建新部署，然后选择 “下一步”。
5. 如果您选择创建新部署，请为部署选择目标核心设备或事物组。在 “指定目标” 页面的 “部署目标” 下，选择核心设备或事物组，然后选择下一步。

6. 在 “选择组件” 页面上，确认已选择该aws.greengrass.clientdevices.IPDetector组件，然后选择 “下一步”。
7. 在 “配置组件” 页面上 aws.greengrass.clientdevices.IPDetector，选择，然后执行以下操作：

   a. 选择配置组件。
   b. 在 “配置” aws.greengrass.clientdevices.IPDetector 模式的 “配置更新” 下，在 “要合并的配置” 中，您可以输入配置更新来配置 IP 检测器组件。您可以指定以下任一配置选项：

      - `defaultPort`—（可选）此组件检测到 IP 地址时要报告的 MQTT 代理端口。如果您将 MQTT 代理配置为使用与默认端口 8883 不同的端口，则必须指定此参数。
      - `includeIPv4LoopbackAddrs`—（可选）您可以启用此选项来检测和报告 IPv4 环回地址。这些是 IP 地址，例如`localhost`，设备可以与自己通信的地方。在核心设备和客户端设备在同一系统上运行的测试环境中使用此选项。
      - `includeIPv4LinkLocalAddrs`—（可选）您可以启用此选项来检测和报告 IPv4 链路本地地址。如果核心设备的网络没有动态主机配置协议 (DHCP) 或静态分配的 IP 地址，请使用此选项。

      配置更新可能与以下示例类似。

      ```
      {
        "defaultPort": "8883",
        "includeIPv4LoopbackAddrs": false,
        "includeIPv4LinkLocalAddrs": false
      }
      ```
   c. 选择 “确认” 关闭模式，然后选择 “下一步”。
8. 在配置高级设置页面上，保留默认配置设置，然后选择下一步。
9. 在 Review（检查）页上，选择 Deploy（部署）。

   部署最多可能需要一分钟才能完成。

部署 IP 检测器组件 (AWS CLI)

要部署 IP 检测器组件，请创建包含`aws.greengrass.clientdevices.IPDetector`在`components`对象中的部署文档，并指定该组件的配置更新。按照中的创建部署说明创建新部署或修改现有部署。

在创建部署文档时，您可以指定以下任一选项来配置 IP 检测器组件：

- defaultPort—（可选）此组件检测到 IP 地址时要报告的 MQTT 代理端口。如果您将 MQTT 代理配置为使用与默认端口 8883 不同的端口，则必须指定此参数。
- includeIPv4LoopbackAddrs—（可选）您可以启用此选项来检测和报告 IPv4 环回地址。这些是 IP 地址，例如localhost，设备可以与自己通信的地方。在核心设备和客户端设备在同一系统上运行的测试环境中使用此选项。
- includeIPv4LinkLocalAddrs—（可选）您可以启用此选项来检测和报告 IPv4 链路本地地址。如果核心设备的网络没有动态主机配置协议 (DHCP) 或静态分配的 IP 地址，请使用此选项。

以下示例部分部署文档指定将端口 8883 报告为 MQTT 代理端口。

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

手动管理端点

您可以手动管理核心设备的 MQTT 代理端点。

每个 MQTT 代理端点都有以下信息：

端点 (HostAddress)

客户端设备可以连接到核心设备上的 MQTT 代理的 IP 地址或 DNS 地址。

端口 (PortNumber)

MQTT 代理在核心设备上运行的端口。

您可以在 M oquette MQTT 代理组件上配置此端口，该组件默认使用端口 8883。

元数据 (Metadata)

向连接到此端点的客户端设备提供的其他元数据。

主题

- 管理终端节点（控制台）
- 管理端点 (AWS CLI)
- 管理终端节点 (API)

管理终端节点（控制台）

您可以使用AWS IoT Greengrass控制台查看、更新和删除核心设备的端点。

管理核心设备的端点（控制台）

1. 导航到 AWS IoT Greengrass 控制台。
2. 选择核心设备。
3. 选择托管管理的核心设备。
4. 在核心设备的详细信息页面上，请选择客户端设备选项卡。
5. 在 MQTT 代理端点部分，您可以看到核心设备的 MQTT 代理端点。选择管理端点。
6. 在管理终端节点模式中，添加或删除核心设备的 MQTT 代理端点。
7. 选择更新。

管理端点 (AWS CLI)

您可以使用 AWS Command Line Interface (AWS CLI) 来管理核心设备的端点。

> **Note**
>
> 由于中的AWS IoT Greengrass V2客户端设备支持向后兼容AWS IoT Greengrass V1，因此您可以使用AWS IoT Greengrass V2或 AWS IoT Greengrass V1 API 操作来管理核心设备端点。

获取核心设备的端点 (AWS CLI)

- 使用以下任一命令：
  - greengrassv2： get-connectivity-info
  - greengrass: get-connectivity-info

更新核心设备的端点 (AWS CLI)

- 使用以下任一命令：
  - greengrassv2： update-connectivity-info
  - greengrass: update-connectivity-info

管理终端节点 (API)

您可以使用 AWS API 来管理核心设备的端点。

> **Note**
> 由于中的AWS IoT Greengrass V2客户端设备支持向后兼容AWS IoT Greengrass V1，因此您可以使用AWS IoT Greengrass V2或 AWS IoT Greengrass V1 API 操作来管理核心设备端点。

获取核心设备的端点 (AWSAPI)

- 使用以下任一操作：
  - V2: GetConnectivityInfo
  - V1: GetConnectivityInfo

更新核心设备的端点 (AWSAPI)

- 使用以下任一操作：
  - V2: UpdateConnectivityInfo
  - V1: UpdateConnectivityInfo

选择一个 MQTT 经纪商

AWS IoT Greengrass提供选项供您选择在核心设备上运行哪个本地 MQTT 代理。客户端设备连接到在核心设备上运行的 MQTT 代理，因此请选择与您要连接的客户端设备兼容的 MQTT 代理。

Note

我们建议您仅部署一个 MQTT 代理组件。MQTT 网桥和 IP 检测器组件一次只能使用一个 MQTT 代理组件。如果您部署多个 MQTT 代理组件，则必须将它们配置为使用不同的端口。

您可以从以下 MQTT 经纪商中进行选择：

- MQTT 3.1.1 经纪商（Mo quette）— `aws.greengrass.clientdevices.mqtt.Moquette`

  对于符合 MQTT 3.1.1 标准的轻量级 MQTT 代理，请选择此选项。AWS IoT CoreMQTT 代理AWS IoT Device SDK也符合 MQTT 3.1.1 标准，因此您可以使用这些功能来创建在您的设备上使用 MQTT 3.1.1 的应用程序。AWS Cloud

- MQTT 5 经纪商 (EMQX) — `aws.greengrass.clientdevices.mqtt.EMQX`

  选择此选项可在核心设备和客户端设备之间的通信中使用 MQTT 5 功能。该组件使用的资源比 Moquette MQTT 3.1.1 代理还要多，在 Linux 核心设备上，它需要 Docker。

  MQTT 5 与 MQTT 3.1.1 向后兼容，因此您可以将使用 MQTT 3.1.1 的客户端设备连接到该代理。如果您运行 Moquette MQTT 3.1.1 代理，则可以将其替换为 EMQX MQTT 5 代理，客户端设备可以继续照常连接和运行。

- 实现自定义代理

  选择此选项可创建用于与客户端设备通信的自定义本地代理组件。您可以创建使用 MQTT 以外的协议的自定义本地代理。 AWS IoT Greengrass提供了一个组件 SDK，可用于对客户端设备进行身份验证和授权。有关更多信息，请参阅 使用AWS IoT Device SDK与 Greengrass 原子核、其他组件进行通信 AWS IoT Core 和 对客户端设备进行身份验证和授权。

使用 MQTT 代理将客户端设备连接到AWS IoT Greengrass核心设备

当您在AWS IoT Greengrass Core 设备上使用 MQTT 代理时，设备会使用该设备独有的核心设备证书颁发机构 (CA) 向代理颁发证书，以便与客户端建立双向 TLS 连接。

AWS IoT Greengrass将自动生成核心设备 CA，也可以提供您的 CA。连接客户端设备身份验证组件 AWS IoT Greengrass时，会使用核心设备 CA 进行注册。自动生成的核心设备 CA 是永久性的，只要配置了客户端设备身份验证组件，设备就会继续使用相同的 CA。

当 MQTT 代理启动时，它会请求证书。客户端设备身份验证组件使用核心设备 CA 颁发 X.509 证书。当代理启动、证书过期或连接信息（例如 IP 地址）发生变化时，证书会轮换。有关更多信息，请参阅在本地 MQTT 代理上轮换证书：

要将客户端连接到 MQTT 代理，您以下以下以下参数：

- 客户端设备必须具有AWS IoT Greengrass核心设备 CA。您可以通过云发现或手动提供 CA 来获得此 CA。有关更多信息，请参阅使用您自己的证书颁发机构：
- 核心设备（CA）颁发的代理证书中的核心设备的完全限定域名（FQDN）或 IP 地址。您可以使用IP 探测器组件或手动配置 IP 地址来确保这一点。有关更多信息，请参阅管理核心设备端点：
- 客户端设备身份验证组件必须授予客户端设备连接到 Greengrass 核心设备的权限。有关更多信息，请参阅客户端设备身份验证：

使用您自己的证书颁发机构

如果您的客户端设备无法访问云以发现您的核心设备，则可以提供核心设备证书颁发机构 (CA)。您的 Greengrass 核心设备使用核心设备 CA 为您的 MQTT 代理颁发证书。配置核心设备并向客户端设备预置其 CA 后，您的客户端设备即可连接到终端节点并使用核心设备 CA（自己提供的 CA 或自动生成）验证 TLS 握手。

要将客户端设备身份验证组件配置为使用您的核心设备 CA，请在部署组件时设置`certificateAuthority`配置参数。您提供以下细节在配置期间提供以下详细信息：

- 核心设备 CA 证书的位置。
- 核心设备 CA 证书的私钥。
- （可选）如果核心设备 CA 是中间 CA，则证书链到根证书。

如果您提供核心设备 CA，则将 CAAWS IoT Greengrass 注册到云中。

您可以将证书存储在硬件安全模块或文件系统中。以下示例显示了使用 HSM/TPM 存储的中间 CA 的`certificateAuthority`配置。请注意，证书链只能存储在磁盘上。

```
"certificateAuthority": {
    "certificateUri": "pkcs11:object=CustomerIntermediateCA;type=cert",
    "privateKeyUri": "pkcs11:object=CustomerIntermediateCA;type=private"
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

在此示例中，certificateAuthority配置参数将客户端设备身份验证组件配置为使用文件系统中的中间 CA：

```
"certificateAuthority": {
    "certificateUri": "file:///home/ec2-user/creds/intermediateCA.pem",
    "privateKeyUri": "file:///home/ec2-user/creds/intermediateCA.privateKey.pem",
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

要将设备连接到AWS IoT Greengrass Core 设备，请执行以下操作：

1. 使用您组织的根 CA 为 Greengrass 核心设备创建中间证书颁发机构（CA）。我们建议您使用中间 CA 作为最佳安全实践。
2. 向 Greengrass 核心设备提供您的根 CA 的中间 CA 证书、私钥和证书链。有关更多信息，请参阅客户端设备身份验证：中间 CA 成为 Greengrass 核心设备的核心设备 CA，该设备将 CA 注册到AWS IoT Greengrass。
3. 将客户端设备注册为AWS IoT事物。有关更多信息，请参阅AWS IoT Core开发者指南中的创建事物对象。将私有密钥、公有密钥、设备证书和根 CA 证书添加到您的客户端设备中。如何添加信息取决于您的设备和软件。

配置设备后，您可以使用证书和公钥链连接到 Greengrass 核心设备。您的软件负责查找核心设备端点。您可以手动为核心设备设置终端节点。有关更多信息，请参阅 手动管理端点。

测试客户端设备通信

客户端设备可以使用AWS IoT Device SDK来发现、连接核心设备并与之通信。您可以使用中的 Greengrass 发现客户端AWS IoT Device SDK来使用 Greengras s 发现 API，它会返回有关客户端设备可以连接的核心设备的信息。API 响应包括要连接的 MQTT 代理端点和用于验证每台核心设备身份的证书。然后，客户端设备可以尝试每个端点，直到它成功连接到核心设备。

客户端设备只能发现您与之关联的核心设备。在测试客户端设备与核心设备之间的通信之前，必须将客户端设备与核心设备相关联。有关更多信息，请参阅 关联客户端设备。

Greengrass 发现 API 会返回您指定的核心设备 MQTT 代理端点。您可以使用 IP 检测器组件为您管理这些端点，也可以手动管理每台核心设备的端点。有关更多信息，请参阅 管理核心设备端点。

Note

要使用 Greengrass 发现 API，客户端设备必须拥有该权限。`greengrass:Discover`有关更多信息，请参阅 客户端设备的最低AWS IoT政策。

AWS IoT Device SDK有多种编程语言版本。有关更多信息，请参阅AWS IoT Core 开发人员指南中的AWS IoT Device 软件开发工具包。

主题

- 测试通信 (Python)
- 测试通信 (C++)
- 测试通信 (JavaScript)
- 测试通信 (Java)

测试通信 (Python)

在本节中，您将使用适用于 Python 的 AWS IoT Device SDK v2 中的 Greengrass 发现示例来测试客户端设备与核心设备之间的通信。

Important

要使用适用于 Python 的 AWS IoT Device SDK v2，设备必须运行 Python 3.6 或更高版本。

测试通信（适用于 Python 的 AWS IoT Device SDK v2）

1. 下载适用于 Python 的 AWS IoT Device SDK v2 并将其安装到要作为客户端设备连接的设备上。AWS IoT

   在客户端设备上，执行以下操作：

   a. 克隆适用于 Python 的 AWS IoT Device SDK v2 版本库进行下载。

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. 安装适用于 Python 的 AWS IoT Device SDK v2。

```
python3 -m pip install --user ./aws-iot-device-sdk-python-v2
```

2. 在 python 版 AWS IoT Device SDK v2 中切换到示例文件夹。

```
cd aws-iot-device-sdk-python-v2/samples
```

3. 运行示例 Greengrass 发现应用程序。此应用程序需要指定客户端设备事物名称、要使用的 MQTT 主题和消息以及用于验证和保护连接的证书的参数。以下示例向该clients/*MyClientDevice1*/hello/world主题发送了一条 Hello World 消息。

- 将 *MyClientDevice1* 替换为客户端设备的事物名称。
- 将 *~/certs/ AmazonRoot ca1.pem #*换为客户端设备上亚马逊根 CA 证书的路径。
- 将 *~/certs/device.pem.crt ########*上设备证书的路径。
- 将 *~/certs/private.pem.key ##########*文件的路径。
- 将 *us-east-1* 替换为您的客户端设备和核心设备运行的AWS区域。

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

发现示例应用程序发送消息 10 次并断开连接。它还订阅发布消息的同一主题。如果输出显示应用程序收到了有关该主题的 MQTT 消息，则客户端设备可以成功地与核心设备通信。

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
```

```
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

如果应用程序改为输出错误，请参阅 Greengrass 发现问题疑难解答。

您还可以查看核心设备上的 Greengrass 日志，以验证客户端设备是否成功连接和发送消息。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

测试通信 (C++)

在本节中，您将使用适用于 C++ 的 v2 中的 AWS IoT Device SDK Greengrass 发现示例来测试客户端设备与核心设备之间的通信。

要构建 C++ 版 AWS IoT Device SDK v2，设备必须具有以下工具：

- C++ 11 或更高版本
- cMake 3.1 或更高版本
- 以下编译器之一：
  - GCC 4.8 或更高版本

- Clang 3.9 或更高版本
- MSVC 2015 或更高版本

测试通信（适用于 C++ 的 AWS IoT Device SDK v2）

1. 下载并构建 C++ 版 AWS IoT Device SDK v2 以AWS IoT将其作为客户端设备进行连接。

   在客户端设备上，执行以下操作：

   a. 为 AWS IoT Device SDK v2 for C++ 工作区创建一个文件夹，然后更改为该文件夹。

   ```
   cd
   mkdir iot-device-sdk-cpp
   cd iot-device-sdk-cpp
   ```

   b. 克隆适用于 C++ 的 AWS IoT Device SDK v2 存储库进行下载。该`--recursive`标志指定下载子模块。

   ```
   git clone --recursive https://github.com/aws/aws-iot-device-sdk-cpp-v2.git
   ```

   c. 为 AWS IoT Device SDK v2 for C++ 编译输出创建一个文件夹，然后更改为该文件夹。

   ```
   mkdir aws-iot-device-sdk-cpp-v2-build
   cd aws-iot-device-sdk-cpp-v2-build
   ```

   d. 为 C++ 构建 AWS IoT Device SDK v2。

   ```
   cmake -DCMAKE_INSTALL_PREFIX="~/iot-device-sdk-cpp" -
   DCMAKE_BUILD_TYPE="Release" ../aws-iot-device-sdk-cpp-v2
   cmake --build . --target install
   ```

2. 在 v2 中构建 C++ 版 Greengrass 发现示例应用程序。AWS IoT Device SDK执行以下操作：

   a. 切换到 C++ 版 v2 中的 Greengrass 发现示例文件夹。AWS IoT Device SDK

   ```
   cd ../aws-iot-device-sdk-cpp-v2/samples/greengrass/basic_discovery
   ```

   b. 为 Greengrass 发现示例构建输出创建一个文件夹，然后更改为该文件夹。

   ```
   mkdir build
   ```

```
cd build
```

c. 构建 Greengrass 发现示例应用程序。

```
cmake -DCMAKE_PREFIX_PATH="~/iot-device-sdk-cpp" -
DCMAKE_BUILD_TYPE="Release" ..
cmake --build . --config "Release"
```

3. 运行示例 Greengrass 发现应用程序。此应用程序需要指定客户端设备事物名称、要使用的 MQTT 主题以及用于验证和保护连接的证书的参数。以下示例订阅了该`clients/MyClientDevice1/hello/world`主题，并将您在命令行中输入的消息发布到同一主题。

   - 将 *MyClientDevice1* 替换为客户端设备的事物名称。
   - 将 *~/certs/ AmazonRoot ca1.pem #*换为客户端设备上亚马逊根 CA 证书的路径。
   - 将 *~/certs/device.pem.crt ########*上设备证书的路径。
   - 将 *~/certs/private.pem.key ##########*文件的路径。
   - 将 *us-east-1* 替换为您的客户端设备和核心设备运行的AWS区域。

```
./basic-discovery \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1
```

discovery 示例应用程序订阅主题并提示您输入要发布的消息。

```
Connecting to group greengrassV2-coreDevice-MyGreengrassCore with thing arn
 arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Connected to group greengrassV2-coreDevice-MyGreengrassCore, using connection to
 203.0.113.0:8883
Successfully subscribed to clients/MyClientDevice1/hello/world
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
```

如果应用程序改为输出错误，请参阅 Greengrass 发现问题疑难解答。

4. 输入一条消息，例如**`Hello World!`**。

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
Hello World!
```

如果输出显示应用程序收到了有关该主题的 MQTT 消息，则客户端设备可以成功地与核心设备通信。

```
Operation on packetId 2 Succeeded
Publish received on topic clients/MyClientDevice1/hello/world
Message:
Hello World!
```

您还可以查看核心设备上的 Greengrass 日志，以验证客户端设备是否成功连接和发送消息。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

测试通信 (JavaScript)

在本节中，您将使用 v2 JavaScript 中的 AWS IoT Device SDK Greengrass 发现示例来测试客户端设备与核心设备之间的通信。

> **Important**
>
> 要将 AWS IoT Device SDK v2 用于 JavaScript，设备必须运行 Node v10.0 或更高版本。

测试通信（AWS IoT Device SDKv2 适用于 JavaScript）

1. 下载并安装该设备的 AWS IoT Device SDK JavaScript v2 以作为客户端设备进行连接。AWS IoT

   在客户端设备上，执行以下操作：

   a. 克隆 AWS IoT Device SDK v2 以供 JavaScript 存储库下载。

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
   ```

   b. 安装适用的 AWS IoT Device SDK v2。 JavaScript

   ```
   cd aws-iot-device-sdk-js-v2
   ```

```
npm install
```

2. 切换到 v2 中的 Greengrass 发现示例文件夹。AWS IoT Device SDK JavaScript

```
cd samples/node/basic_discovery
```

3. 安装 Greengrass 发现示例应用程序。

```
npm install
```

4. 运行示例 Greengrass 发现应用程序。此应用程序需要指定客户端设备事物名称、要使用的 MQTT 主题和消息以及用于验证和保护连接的证书的参数。以下示例向该`clients/MyClientDevice1/hello/world`主题发送了一条 Hello World 消息。

   - 将 *MyClientDevice1* 替换为客户端设备的事物名称。
   - 将 *~/certs/ AmazonRoot ca1.pem #*换为客户端设备上亚马逊根 CA 证书的路径。
   - 将 *~/certs/device.pem.crt ########*上设备证书的路径。
   - 将 *~/certs/private.pem.key ##########*文件的路径。
   - 将 *us-east-1* 替换为您的客户端设备和核心设备运行的AWS区域。

```
node dist/index.js \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --message 'Hello World!' \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbose warn
```

发现示例应用程序发送消息 10 次并断开连接。它还订阅发布消息的同一主题。如果输出显示应用程序收到了有关该主题的 MQTT 消息，则客户端设备可以成功地与核心设备通信。

```
Discovery Response:
{"gg_groups":[{"gg_group_id":"greengrassV2-coreDevice-
MyGreengrassCore","cores":[{"thing_arn":"arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore","connectivity":
[{"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}]}],"certificat
["-----BEGIN CERTIFICATE-----\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n"]}]}
```

```
Trying
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
[WARN] [2021-06-12T00:46:45Z] [00007f90c0e8d700] [socket] - id=0x7f90b8018710
 fd=26: setsockopt() for NO_SIGNAL failed with errno 92. If you are having SIGPIPE
 signals thrown, you may want to install a signal trap in your application layer.
Connected to
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":1}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":2}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":3}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":4}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":5}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":6}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":7}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":8}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":9}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":10}
Complete!
```

如果应用程序改为输出错误，请参阅 Greengrass 发现问题疑难解答。

您还可以查看核心设备上的 Greengrass 日志，以验证客户端设备是否成功连接和发送消息。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

测试通信 (Java)

在本节中，您将使用适用于 Java 的 v2 中的 AWS IoT Device SDK Greengrass 发现示例来测试客户端设备与核心设备之间的通信。

> **Important**
>
> 要构建 Java 版 AWS IoT Device SDK v2，设备必须具有以下工具：
>
> - Java 8 或更高版本，`JAVA_HOME`指向 Java 文件夹。
> - Apache Maven

测试通信（适用于 Java 的 AWS IoT Device SDK v2）

1. 下载并编译适用于 Java 的 AWS IoT Device SDKAWS IoT v2，将其作为客户端设备进行连接。

   在客户端设备上，执行以下操作：

   a. 克隆适用于 Java 的 AWS IoT Device SDK v2 存储库进行下载。

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-java-v2.git
   ```

   b. 切换到适用于 Java 的 AWS IoT Device SDK v2 文件夹。

   c. 编译适用于 Java 的 AWS IoT Device SDK v2。

   ```
   cd aws-iot-device-sdk-java-v2
   mvn versions:use-latest-versions -Dincludes="software.amazon.awssdk.crt*"
   mvn clean install
   ```

2. 运行示例 Greengrass 发现应用程序。此应用程序需要指定客户端设备事物名称、要使用的 MQTT 主题以及用于验证和保护连接的证书的参数。以下示例订阅了该`clients/MyClientDevice1/hello/world`主题，并将您在命令行中输入的消息发布到同一主题。

   - 将 *MyClientDevice1* 的两个实例替换为客户端设备的事物名称。
   - 将 *$home/certs/ AmazonRoot ca1.pem #*换为客户端设备上亚马逊根 CA 证书的路径。
   - 将 *$HOME/certs/device.pem.crt ########*上设备证书的路径。
   - 将 *$HOME/certs/Private.pem.key ###########*文件的路径。
   - 将 *us-east-1* 替换为客户端设备和核心设备的运行AWS 区域位置。

```
DISCOVERY_SAMPLE_ARGS="--thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --ca_file $HOME/certs/AmazonRootCA1.pem \
  --cert $HOME/certs/device.pem.crt \
  --key $HOME/certs/private.pem.key \
  --region us-east-1"

mvn exec:java -pl samples/Greengrass \
  -Dexec.mainClass=greengrass.BasicDiscovery \
  -Dexec.args="$DISCOVERY_SAMPLE_ARGS"
```

discovery 示例应用程序订阅主题并提示您输入要发布的消息。

```
Connecting to group ID greengrassV2-coreDevice-MyGreengrassCore, with thing
 arn arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Started a clean session
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
```

如果应用程序改为输出错误，请参阅 Greengrass 发现问题疑难解答。

3. 输入一条消息，例如**`Hello World!`**。

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
Hello World!
```

如果输出显示应用程序收到了有关该主题的 MQTT 消息，则客户端设备可以成功地与核心设备通信。

```
Message received on topic clients/MyClientDevice1/hello/world: Hello World!
```

您还可以查看核心设备上的 Greengrass 日志，以验证客户端设备是否成功连接和发送消息。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

Greengrass 发现 RESTful API

AWS IoT Greengrass提供了 `Discover` API 操作，客户端设备可以使用该操作来识别 Greengrass 核心设备可以连接的位置。客户端设备使用此数据平面操作来检索连接到 Greengrass 核心设备所需的信息，您可以将它们与 API 操作相关联。BatchAssociateClientDeviceWithCoreDevice当客户端设备上线时，它可以连接到AWS IoT Greengrass云服务并使用发现 API 来查找：

- 每台关联的 Greengrass 核心设备的 IP 地址和端口。
- 核心设备 CA 证书，客户端设备可以使用该证书对 Greengrass 核心设备进行身份验证。

> **Note**
>
> 客户端设备还可以使用中的发现客户端AWS IoT Device SDK来发现 Greengrass 核心设备的连接信息。发现客户端使用发现 API。有关更多信息，请参阅下列内容：
>
> - 测试客户端设备通信
> - 开发者@@ 指南中的 Greengrass Discovery RES Tful API。AWS IoT Greengrass Version 1

要使用此 API 操作，请向 Greengrass 数据平面端点上的发现 API 发送 HTTP 请求。此 API 端点采用以下格式。

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

有关AWS IoT Greengrass发现 API 支持的终端节点列表AWS 区域和终端节点列表，请参阅中的AWS IoT Greengrass V2终端节点和配额AWS 一般参考。此 API 操作仅在 Greengrass 数据平面端点上可用。用于管理组件和部署的控制平面端点与数据平面端点不同。

> **Note**
>
> AWS IoT Greengrass V1和的发现 API 相同AWS IoT Greengrass V2。如果您有连接到AWS IoT Greengrass V1核心的客户端设备，则无需更改客户端设备上的代码即可将它们连接到AWS IoT Greengrass V2核心设备。有关更多信息，请参阅《开发者指南》中的 Greengrass Discovery RES Tful API。AWS IoT Greengrass Version 1

主题

- 发现身份验证和授权
- 请求
- 响应
- 使用 curl 测试发现 API

发现身份验证和授权

要使用发现 API 检索连接信息，客户端设备必须使用 TLS 双向身份验证和 X.509 客户端证书进行身份验证。有关更多信息，请参阅《开发人员指南》AWS IoT Core中的 X.509 客户端证书。

客户端设备还必须具有执行该greengrass:Discover操作的权限。以下示例AWS IoT策略允许名MyClientDevice1为AWS IoT的事物自行执行Discover。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:Discover",
      "Resource": [
        "arn:aws:iot:us-west-2:123456789012:thing/MyClientDevice1"
      ]
    }
  ]
}
```

> **Important**
>
> 核心设备或 Greengrass 数据平面操作的AWS IoT策略不支持@@ 事物策略变量 (iot:Connection.Thing.*)。相反，您可以使用通配符来匹配多个名称相似的设备。例如，您可以指定MyGreengrassDevice*匹配MyGreengrassDevice1MyGreengrassDevice2、等。

有关更多信息，请参阅《AWS IoT Core开发者指南》中的AWS IoT Core政策。

请求

该请求包含标准的 HTTP 标头，并被发送到 Greengrass 发现端点，如以下示例所示。

端口号取决于核心设备是配置为通过端口 8443 还是端口 443 发送 HTTPS 流量。有关更多信息，请参阅 the section called “通过端口 443 或网络代理进行连接”。

Note

这些示例使用亚马逊信任服务 (ATS) 终端节点，该终端节点适用于推荐的 ATS 根 CA 证书。终端节点必须与 CA 证书类型匹配。

端口 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/
thing/thing-name
```

端口 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/
thing/thing-name
```

Note

在端口 443 上连接的客户端必须实现应用层协议协商 (ALPN) TLS 扩展，并且必须在 `ProtocolNameList` 中作为 `ProtocolName` 传递 `x-amzn-http-ca`。更多信息，请参阅 AWS IoT 开发人员指南中的协议。

响应

成功后，响应标头包含 HTTP 200 状态码，响应正文包含发现响应文档。

Note

由于AWS IoT Greengrass V2使用与相同的发现 APIAWS IoT Greengrass V1，因此响应会根据AWS IoT Greengrass V1概念（例如 Greengrass 群组）来组织信息。响应中包含

Greengrass 群组列表。在中AWS IoT Greengrass V2，每台核心设备都位于自己的组中，该组中仅包含该核心设备及其连接信息。

示例发现响应文档

以下文档显示了与一个 Greengrass 核心设备关联的客户端设备的响应。核心设备有一个端点和一个 CA 证书。

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

以下文档显示了与两台核心设备关联的客户端设备的响应。核心设备有多个端点和多个组 CA 证书。

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
```

```
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-connection-1-description"
            },
            {
              "id": "core-device-01-connection-id-2",
              "hostAddress": "core-device-01-address-2",
              "portNumber": core-device-01-port-2,
              "metadata": "core-device-01-connection-2-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-02-thing-name",
      "Cores": [
        {
          "thingArn":"core-device-02-thing-arn",
          "Connectivity" : [
            {
              "id": "core-device-02-connection-id",
              "hostAddress": "core-device-02-address",
              "portNumber": core-device-02-port,
              "metadata": "core-device-02-connection-1-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
```

```
}
```

使用 curl 测试发现 API

如果您已cURL安装，则可以测试发现 API。以下示例指定客户端设备的证书，用于对发往 Greengrass 发现 API 端点的请求进行身份验证。

```
curl -i \
  --cert 1a23bc4d56.cert.pem \
  --key 1a23bc4d56.private.key \
  https://greengrass-ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/
thing/MyClientDevice1
```

> **Note**
>
> `-i`参数指定输出 HTTP 响应标头。您可以使用此选项来帮助识别错误。

如果请求成功，则此命令将输出类似于以下示例的响应。

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-MyGreengrassCore",
      "Cores": [
        {
          "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
          "Connectivity": [
            {
              "Id": "AUTOIP_192.168.1.4_1",
              "HostAddress": "192.168.1.5",
              "PortNumber": 8883,
              "Metadata": ""
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----\ncert-contents\n-----END CERTIFICATE-----\n"
      ]
    }
```

```
  ]
}
```

如果命令输出错误，请参阅 Greengrass 发现问题疑难解答。

在客户端设备之间中继 MQTT 消息和 AWS IoT Core

您可以在客户端设备和之间中继 MQTT 消息和其他数据。AWS IoT Core客户端设备连接到在核心设备上运行的 MQTT 代理组件。默认情况下，核心设备不会在客户端设备和AWS IoT Core之间中继 MQTT 消息或数据。默认情况下，客户端设备只能通过 MQTT 相互通信。

要在客户端设备和之间中继 MQTT 消息AWS IoT Core，请将 MQTT 网桥组件配置为执行以下操作：

- 将来自客户端设备的消息中继到AWS IoT Core。
- AWS IoT Core将来自客户端设备的消息中继。

> Note
>
> 即使客户端设备使用 QoS 0 发布和订阅AWS IoT Core本地 MQTT 代理，MQTT 桥也使用 QoS 1 来发布和订阅。因此，当您将来自本地 MQTT 代理上的客户端设备的 MQTT 消息中继到时，您可能会观察到额外的延迟。AWS IoT Core有关核心设备上的 MQTT 配置的更多信息，请参阅配置 MQTT 超时和缓存设置。

主题

- 配置和部署 MQTT 网桥组件
- 中继 MQTT 消息

配置和部署 MQTT 网桥组件

MQTT bridge 组件使用主题映射列表，每个主题映射都指定消息源和消息目的地。要在客户端设备和之间中继消息AWS IoT Core，请部署 MQTT 桥接组件，并在组件配置中指定每个源和目标主题。

要将 MQTT 桥接组件部署到一台或一组核心设备，请创建包含该`aws.greengrass.clientdevices.mqtt.Bridge`组件的部署。在部署的 MQTT 网桥组件配置中指定主题映射。`mqttTopicMapping`

以下示例定义了一个部署，该部署将 MQTT 桥接组件配置为将与主题过滤器匹配的主题上的消息从客户端设备中继到。clients/+/hello/world AWS IoT Coremerge配置更新需要序列化的 JSON 对象。有关更多信息，请参阅 更新组件配置。

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCore": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"mqttTopicMapping\":{\"HelloWorldIotCore\":{\"topic\":\"clients/
+/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"IotCore\"}}}"
      }
    }
    ...
  }
}
```

中继 MQTT 消息

要在客户端设备和之间中继 MQTT 消息AWS IoT Core，请配置和部署 MQTT Bridge 组件并指定要中继的主题。

Example 示例：将有关某个主题的消息从客户端设备中继到 AWS IoT Core

以下 MQTT bridge 组件配置指定将与主题筛选条件匹配的clients/+/hello/world/event主题上的消息从客户端设备转发到。AWS IoT Core

```
{
  "mqttTopicMapping": {
    "HelloWorldEvent": {
      "topic": "clients/+/hello/world/event",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example 示例：将有关某个主题的消息从客户端设备中继AWS IoT Core出来

以下 MQTT bridge 组件配置指定将与主题筛选条件匹配的`clients/+/hello/world/event/response`主题上的消息中继AWS IoT Core到客户端设备。

```
{
  "mqttTopicMapping": {
    "HelloWorldEventConfirmation": {
      "topic": "clients/+/hello/world/event/response",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

在组件中与客户端设备交互

您可以开发与连接到核心设备的客户端设备交互的自定义 Greengrass 组件。例如，您可以开发执行以下操作的组件：

- 处理来自客户端设备的 MQTT 消息并将数据发送到AWS Cloud目的地。
- 向客户端设备发送 MQTT 消息以启动操作。

客户端设备通过核心设备上运行的 MQTT 代理组件连接到核心设备并与之通信。默认情况下，客户端设备只能通过 MQTT 相互通信，而 Greengrass 组件无法接收这些 MQTT 消息或向客户端设备发送消息。

Greengrass 组件使用本地发布/订阅接口在核心设备上进行通信。要在 Greengrass 组件中与客户端设备通信，请将 MQTT 网桥组件配置为执行以下操作：

- 将来自客户端设备的 MQTT 消息中继到本地发布/订阅。
- 将来自本地发布/订阅的 MQTT 消息中继到客户端设备。

您还可以在 Greengrass 组件中与客户端设备阴影进行交互。有关更多信息，请参阅 与客户端设备影子进行交互并进行同步。

主题

- 配置和部署 MQTT 网桥组件
- 从客户端设备接收 MQTT 消息
- 向客户端设备发送 MQTT 消息

配置和部署 MQTT 网桥组件

MQTT bridge 组件使用主题映射列表，每个主题映射都指定消息源和消息目的地。要与客户端设备通信，请部署 MQTT 网桥组件，并在组件配置中指定每个源和目标主题。

要将 MQTT 桥接组件部署到一台或一组核心设备，请创建包含该aws.greengrass.clientdevices.mqtt.Bridge组件的部署。在部署的 MQTT 网桥组件配置中指定主题映射。mqttTopicMapping

以下示例定义了一个部署，该部署将 MQTT 桥接组件配置为将clients/MyClientDevice1/hello/world主题从客户端设备中继到本地发布/订阅代理。merge配置更新需要序列化的 JSON 对象。有关更多信息，请参阅 更新组件配置。

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldPubsub": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

AWS CLI

```
{
```

```
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "\"mqttTopicMapping\":{\"HelloWorldPubsub\":{\"topic\":\"clients/
MyClientDevice1/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"Pubsub\"}}}"
      }
    }
    ...
  }
}
```

您可以使用 MQTT 主题通配符来中继与主题筛选条件匹配的主题上的消息。如果您使用 MQTT bridge v2.2.0 或更高版本，则当源代理为本地发布/订阅时，可以在主题过滤器中使用 MQTT 主题通配符。有关更多信息，请参阅 MQTT 网桥组件配置。

从客户端设备接收 MQTT 消息

您可以订阅您为 MQTT bridge 组件配置的本地发布/订阅主题，以接收来自客户端设备的消息。

使用自定义组件从客户端设备接收 MQTT 消息

1. 配置和部署 MQTT 桥接组件，将来自客户端设备发布的 MQTT 主题的消息中继到本地发布/订阅主题。
2. 使用本地发布/订阅 IPC 接口订阅 MQTT 网桥中继消息的主题。有关更多信息，请参阅 发布/订阅本地消息和 SubscribeToTopic。

Connect and test 客户端设备教程包括一个部分，您可以在其中开发一个用于订阅来自客户端设备的消息的组件。有关更多信息，请参阅 步骤 4：开发可与客户端设备通信的组件。

向客户端设备发送 MQTT 消息

您可以发布到您为 MQTT bridge 组件配置的本地发布/订阅主题，以便向客户端设备发送消息。

使用自定义组件向客户端设备发布 MQTT 消息

1. 配置和部署 MQTT 桥接组件，将来自本地发布/订阅主题的消息中继到客户端设备订阅的 MQTT 主题。

2. 使用本地发布/订阅 IPC 接口发布到 MQTT 网桥中继消息的主题。有关更多信息，请参阅 发布/订阅本地消息 和 PublishToTopic。

与客户端设备影子进行交互并进行同步

您可以使用阴影管理器组件来管理本地阴影，包括客户端设备阴影。您可以使用影子管理器执行以下操作：

- 在 Greengrass 组件中与客户端设备阴影进行交互。
- 将客户端设备阴影与同步AWS IoT Core。

> Note
>
> 默认情况下，阴影管理器组件不与AWS IoT Core阴影同步。您必须配置影子管理器组件以指定要同步的客户端设备影子。

主题

- 先决条件
- 启用影子管理器与客户端设备通信
- 与组件中的客户端设备阴影交互
- 将客户端设备阴影与同步 AWS IoT Core

先决条件

要与客户端设备影子交互并与之同步客户端设备影子AWS IoT Core，核心设备必须满足以下要求：

- 除了支持客户端设备的 Greengrass 组件外，核心设备还必须运行以下组件：
  - Greengrass nucleus v2.6.0 或更高版本
  - 影子管理器 v2.2.0 或更高版本
  - MQTT bridge v2.0 或更高版本
- 必须将客户端设备身份验证组件配置为允许客户端设备就设备影子主题进行通信。

启用影子管理器与客户端设备通信

默认情况下，影子管理器组件不管理客户端设备影子。要启用此功能，您必须在客户端设备和影子管理器组件之间中继 MQTT 消息。客户端设备使用 MQTT 消息来接收和发送设备影子更新。影子管理器组件订阅本地 Greengrass 发布/订阅接口，因此您可以将 MQTT 桥接组件配置为在设备影子主题上中继 MQTT 消息。

MQTT bridge 组件使用主题映射列表，每个主题映射都指定消息源和消息目的地。要使影子管理器组件能够管理客户端设备影子，请部署 MQTT 桥接组件，并为客户端设备影子指定影子主题。您必须将网桥配置为在本地 MQTT 和本地发布/订阅之间双向中继消息。

要将 MQTT 桥接组件部署到一台或一组核心设备，请创建包含该aws.greengrass.clientdevices.mqtt.Bridge组件的部署。在部署的 MQTT 网桥组件配置中指定主题映射。mqttTopicMapping

使用以下示例配置 MQTT 网桥组件，以启用客户端设备与影子管理器组件之间的通信。

> **Note**
>
> 您可以在AWS IoT Greengrass控制台中使用这些配置示例。如果您使用 AWS IoT Greengrass API，则merge配置更新需要序列化的 JSON 对象，因此必须将以下 JSON 对象序列化为字符串。有关更多信息，请参阅 更新组件配置。

Example 示例：管理所有客户端设备影子

以下 MQTT 桥接配置示例使影子管理器能够管理所有客户端设备的所有阴影。

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
```

```
  }
}
```

Example 示例：管理客户端设备的阴影

以下 MQTT 桥接配置示例使影子管理器能够管理名MyClientDevice为的客户端设备的所有阴影。

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example 示例：管理所有客户端设备的已命名影子

以下 MQTT 网桥配置示例使影子管理器能够管理DeviceConfiguration为所有客户端设备命名的影子。

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example 示例：管理所有客户端设备的未命名阴影

以下 MQTT 桥接配置示例使影子管理器能够管理所有客户端设备的未命名阴影，但未命名的阴影。

```
{
  "mqttTopicMapping": {
    "DeleteShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/delete",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "DeleteShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/delete/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "GetShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/get",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "GetShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/get/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "UpdateShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/update",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "UpdateShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/update/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

与组件中的客户端设备阴影交互

您可以开发使用本地影子服务读取和修改客户端设备的本地影子文档的自定义组件。有关更多信息，请参阅 与组件中的阴影交互。

将客户端设备阴影与同步 AWS IoT Core

您可以将影子管理器组件配置为与本地客户端设备影子状态同步AWS IoT Core。有关更多信息，请参阅 将本地设备阴影与同步 AWS IoT Core。

对客户端设备进行故障排除

使用本节中的故障排除信息和解决方案来帮助解决 Greengrass 客户端设备和客户端设备组件的问题。

主题

- Greengrass 发现问题
- MQTT 连接问题

Greengrass 发现问题

使用以下信息对 Greengrass 发现问题进行故障排除。当客户端设备使用 Greengrass 发现 API 来识别它们可以连接的 Greengrass 核心设备时，可能会出现这些问题。

主题

- Greengrass 发现问题 (HTTP API)
- Greengrass 发现问题（适用于 Python 的 v2）AWS IoT Device SDK
- Greengrass 发现问题（C++ 版 v2）AWS IoT Device SDK
- Greengrass 发现问题 (v2 适用于) AWS IoT Device SDK JavaScript
- Greengrass 发现问题（适用于 Java 的 v2）AWS IoT Device SDK

Greengrass 发现问题 (HTTP API)

使用以下信息对 Greengrass 发现问题进行故障排除。如果您使用 curl 测试发现 API，则可能会看到这些错误。

主题

- curl: (52) Empty reply from server
- HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}
- HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

curl: (52) Empty reply from server

如果您在请求中指定了非活动AWS IoT证书，则可能会看到此错误。

检查客户端设备是否附有证书，以及证书是否处于活动状态。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的将事物或策略附加到客户端证书和激活或停用客户端证书。

HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}

如果客户端设备无权自行调用`greengrass:Discover`，则可能会看到此错误。

检查客户端设备的证书是否有允许的政策`greengrass:Discover`。您不能使用`Resource`部分中的事物策略变量 (`iot:Connection.Thing.*`) 来获得此权限。有关更多信息，请参阅 发现身份验证和授权。

HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

在以下情况下，您可能会看到此错误：

- 客户端设备未与任何 Greengrass 核心设备或群组关联。AWS IoT Greengrass V1
- 客户端设备关联的 Greengrass 核心设备AWS IoT Greengrass V1或组都没有 MQTT 代理端点。
- 客户端设备关联的 Greengrass 核心设备均不运行客户端设备身份验证组件。

检查客户端设备是否与您要连接的核心设备相关联。然后，检查核心设备是否运行客户端设备身份验证组件，并且至少有一个 MQTT 代理端点。有关更多信息，请参阅下列内容：

- 关联客户端设备
- 管理核心设备端点
- 配置云发现（控制台）

Greengrass 发现问题（适用于 Python 的 v2）AWS IoT Device SDK

使用以下信息来解决适用于 Python 的 v2 中的 Greengrass 发现问题。AWS IoT Device SDK

主题

- awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.
- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)

- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.

如果您在请求中指定了非活动AWS IoT证书，则可能会看到此错误。

检查客户端设备是否附有证书，以及证书是否处于活动状态。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的将事物或策略附加到客户端证书和激活或停用客户端证书。

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)

如果客户端设备无权自行调用`greengrass:Discover`，则可能会看到此错误。

检查客户端设备的证书是否有允许的政策`greengrass:Discover`。您不能使用`Resource`部分中的事物策略变量 (`iot:Connection.Thing.*`) 来获得此权限。有关更多信息，请参阅 发现身份验证和授权。

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

在以下情况下，您可能会看到此错误：

- 客户端设备未与任何 Greengrass 核心设备或群组关联。AWS IoT Greengrass V1
- 客户端设备关联的 Greengrass 核心设备AWS IoT Greengrass V1或组都没有 MQTT 代理端点。
- 客户端设备关联的 Greengrass 核心设备均不运行客户端设备身份验证组件。

检查客户端设备是否与您要连接的核心设备相关联。然后，检查核心设备是否运行客户端设备身份验证组件，并且至少有一个 MQTT 代理端点。有关更多信息，请参阅下列内容：

- 关联客户端设备
- 管理核心设备端点
- 配置云发现（控制台）

Greengrass 发现问题（C++ 版 v2）AWS IoT Device SDK

使用以下信息来解决适用于 C++ 的 v2 中的 Greengrass 发现问题。AWS IoT Device SDK

主题

- aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

如果您在请求中指定了非活动AWS IoT证书，则可能会看到此错误。

检查客户端设备是否附有证书，以及证书是否处于活动状态。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的将事物或策略附加到客户端证书和激活或停用客户端证书。

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)

如果客户端设备无权自行调用`greengrass:Discover`，则可能会看到此错误。

检查客户端设备的证书是否有允许的政策`greengrass:Discover`。您不能使用`Resource`部分中的事物策略变量 (`iot:Connection.Thing.*`) 来获得此权限。有关更多信息，请参阅 发现身份验证和授权。

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

在以下情况下，您可能会看到此错误：

- 客户端设备未与任何 Greengrass 核心设备或群组关联。AWS IoT Greengrass V1
- 客户端设备关联的 Greengrass 核心设备AWS IoT Greengrass V1或组都没有 MQTT 代理端点。
- 客户端设备关联的 Greengrass 核心设备均不运行客户端设备身份验证组件。

检查客户端设备是否与您要连接的核心设备相关联。然后，检查核心设备是否运行客户端设备身份验证组件，并且至少有一个 MQTT 代理端点。有关更多信息，请参阅下列内容：

- 关联客户端设备
- 管理核心设备端点
- 配置云发现（控制台）

Greengrass 发现问题 (v2 适用于) AWS IoT Device SDK JavaScript

使用以下信息对 v2 中的 Greengrass 发现问题进行故障排除。AWS IoT Device SDK JavaScript

主题

- Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- Error: Discovery failed (headers: [object Object]) { response_code: 403 }
- Error: Discovery failed (headers: [object Object]) { response_code: 404 }
- Error: Discovery failed (headers: [object Object])

Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

如果您在请求中指定了非活动AWS IoT证书，则可能会看到此错误。

检查客户端设备是否附有证书，以及证书是否处于活动状态。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的将事物或策略附加到客户端证书和激活或停用客户端证书。

Error: Discovery failed (headers: [object Object]) { response_code: 403 }

如果客户端设备无权自行调用`greengrass:Discover`，则可能会看到此错误。

检查客户端设备的证书是否有允许的政策`greengrass:Discover`。您不能使用`Resource`部分中的事物策略变量 (`iot:Connection.Thing.*`) 来获得此权限。有关更多信息，请参阅 发现身份验证和授权。

Error: Discovery failed (headers: [object Object]) { response_code: 404 }

在以下情况下，您可能会看到此错误：

- 客户端设备未与任何 Greengrass 核心设备或群组关联。AWS IoT Greengrass V1
- 客户端设备关联的 Greengrass 核心设备AWS IoT Greengrass V1或组都没有 MQTT 代理端点。
- 客户端设备关联的 Greengrass 核心设备均不运行客户端设备身份验证组件。

检查客户端设备是否与您要连接的核心设备相关联。然后，检查核心设备是否运行客户端设备身份验证组件，并且至少有一个 MQTT 代理端点。有关更多信息，请参阅下列内容：

- 关联客户端设备
- 管理核心设备端点
- 配置云发现（控制台）

Error: Discovery failed (headers: [object Object])

运行 Greengrass 发现示例时，您可能会看到此错误（没有 HTTP 响应代码）。出现此错误的原因可能有多种。

- 如果客户端设备无权自行调用`greengrass:Discover`，则可能会看到此错误。

  检查客户端设备的证书是否有允许的政策`greengrass:Discover`。您不能使用`Resource`部分中的事物策略变量 (`iot:Connection.Thing.*`) 来获得此权限。有关更多信息，请参阅 发现身份验证和授权。
- 在以下情况下，您可能会看到此错误：
  - 客户端设备未与任何 Greengrass 核心设备或群组关联。AWS IoT Greengrass V1
  - 客户端设备关联的 Greengrass 核心设备AWS IoT Greengrass V1或组都没有 MQTT 代理端点。
  - 客户端设备关联的 Greengrass 核心设备均不运行客户端设备身份验证组件。

  检查客户端设备是否与您要连接的核心设备相关联。然后，检查核心设备是否运行客户端设备身份验证组件，并且至少有一个 MQTT 代理端点。有关更多信息，请参阅下列内容：
  - 关联客户端设备
  - 管理核心设备端点
  - 配置云发现（控制台）

Greengrass 发现问题（适用于 Java 的 v2）AWS IoT Device SDK

使用以下信息来解决适用于 Java 的 v2 中的 Greengrass 发现问题。AWS IoT Device SDK

主题

- software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)
- java.lang.RuntimeException: Error x-amzn-ErrorType(403)
- java.lang.RuntimeException: Error x-amzn-ErrorType(404)

software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)

如果您在请求中指定了非活动AWS IoT证书，则可能会看到此错误。

检查客户端设备是否附有证书，以及证书是否处于活动状态。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的将事物或策略附加到客户端证书和激活或停用客户端证书。

java.lang.RuntimeException: Error x-amzn-ErrorType(403)

如果客户端设备无权自行调用`greengrass:Discover`，则可能会看到此错误。

检查客户端设备的证书是否有允许的政策`greengrass:Discover`。您不能使用`Resource`部分中的事物策略变量 (`iot:Connection.Thing.*`) 来获得此权限。有关更多信息，请参阅 发现身份验证和授权。

java.lang.RuntimeException: Error x-amzn-ErrorType(404)

在以下情况下，您可能会看到此错误：

- 客户端设备未与任何 Greengrass 核心设备或群组关联。AWS IoT Greengrass V1
- 客户端设备关联的 Greengrass 核心设备AWS IoT Greengrass V1或组都没有 MQTT 代理端点。
- 客户端设备关联的 Greengrass 核心设备均不运行客户端设备身份验证组件。

检查客户端设备是否与您要连接的核心设备相关联。然后，检查核心设备是否运行客户端设备身份验证组件，并且至少有一个 MQTT 代理端点。有关更多信息，请参阅下列内容：

- 关联客户端设备
- 管理核心设备端点
- 配置云发现（控制台）

MQTT 连接问题

使用以下信息对客户端设备的 MQTT 连接问题进行故障排除。当客户端设备尝试通过 MQTT 连接到核心设备时，可能会出现这些问题。

主题

- io.moquette.broker.Authorizator: Client does not have read permissions on the topic

- MQTT 连接问题 (Python)
- MQTT 连接问题 (C++)
- MQTT 连接问题 (Java)
- MQTT 连接问题 () JavaScript

io.moquette.broker.Authorizator: Client does not have read permissions on the topic

当客户端设备尝试订阅它没有权限的 MQTT 主题时，您可能会在 Greengrass 日志中看到此错误。错误消息包含主题。

检查客户端设备身份验证组件的配置是否包括以下内容：

- 与客户端设备匹配的设备组。
- 该设备组的客户端设备授权策略，用于授予该主题的`mqtt:subscribe`权限。

有关如何部署和配置客户端设备身份验证组件的更多信息，请参阅以下内容：

- 配置云发现（控制台）
- 客户端设备身份验证
- 创建部署

MQTT 连接问题 (Python)

使用适用于 Pyt hon 的 AWS IoT Device SDK v2 时，使用以下信息来解决客户端设备的 MQTT 连接问题。

主题

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

如果客户端设备身份验证组件未定义授予客户端设备连接权限的客户端设备授权策略，则可能会看到此错误。

检查客户端设备身份验证组件的配置是否包括以下内容：

- 与客户端设备匹配的设备组。
- 该设备组的客户端设备授权策略，用于授予对客户端设备的`mqtt:connect`权限。

有关如何部署和配置客户端设备身份验证组件的更多信息，请参阅以下内容：

- 配置云发现（控制台）
- 客户端设备身份验证
- 创建部署

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果客户端设备身份验证组件未定义授予客户端设备连接权限的客户端设备授权策略，则可能会看到此错误。

检查客户端设备身份验证组件的配置是否包括以下内容：

- 与客户端设备匹配的设备组。
- 该设备组的客户端设备授权策略，用于授予对客户端设备的`mqtt:connect`权限。

有关如何部署和配置客户端设备身份验证组件的更多信息，请参阅以下内容：

- 配置云发现（控制台）
- 客户端设备身份验证
- 创建部署

MQTT 连接问题 (C++)

使用适用于 C++ 的 AWS IoT Device SDK v2 时，使用以下信息来解决客户端设备 MQTT 连接问题。

主题

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

如果客户端设备身份验证组件未定义授予客户端设备连接权限的客户端设备授权策略，则可能会看到此错误。

检查客户端设备身份验证组件的配置是否包括以下内容：

- 与客户端设备匹配的设备组。
- 该设备组的客户端设备授权策略，用于授予对客户端设备的`mqtt:connect`权限。

有关如何部署和配置客户端设备身份验证组件的更多信息，请参阅以下内容：

- 配置云发现（控制台）
- 客户端设备身份验证
- 创建部署

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果客户端设备身份验证组件未定义授予客户端设备连接权限的客户端设备授权策略，则可能会看到此错误。

检查客户端设备身份验证组件的配置是否包括以下内容：

- 与客户端设备匹配的设备组。
- 该设备组的客户端设备授权策略，用于授予对客户端设备的`mqtt:connect`权限。

有关如何部署和配置客户端设备身份验证组件的更多信息，请参阅以下内容：

- 配置云发现（控制台）
- 客户端设备身份验证
- 创建部署

MQTT 连接问题 (Java)

使用适用于 Java 的 AWS IoT Device SDK v2 时，使用以下信息来解决客户端设备 MQTT 连接问题。

主题

- software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred

- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred

如果客户端设备身份验证组件未定义授予客户端设备连接权限的客户端设备授权策略，则可能会看到此错误。

检查客户端设备身份验证组件的配置是否包括以下内容：

- 与客户端设备匹配的设备组。
- 该设备组的客户端设备授权策略，用于授予对客户端设备的`mqtt:connect`权限。

有关如何部署和配置客户端设备身份验证组件的更多信息，请参阅以下内容：

- 配置云发现（控制台）
- 客户端设备身份验证
- 创建部署

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果客户端设备身份验证组件未定义授予客户端设备连接权限的客户端设备授权策略，则可能会看到此错误。

检查客户端设备身份验证组件的配置是否包括以下内容：

- 与客户端设备匹配的设备组。
- 该设备组的客户端设备授权策略，用于授予对客户端设备的`mqtt:connect`权限。

有关如何部署和配置客户端设备身份验证组件的更多信息，请参阅以下内容：

- 配置云发现（控制台）
- 客户端设备身份验证
- 创建部署

MQTT 连接问题 () JavaScript

使用 AWS IoT Device SDKv2 时，使用以下信息来解决客户端设备 MQTT 连接问题。 JavaScript

主题

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

如果客户端设备身份验证组件未定义授予客户端设备连接权限的客户端设备授权策略，则可能会看到此错误。

检查客户端设备身份验证组件的配置是否包括以下内容：

- 与客户端设备匹配的设备组。
- 该设备组的客户端设备授权策略，用于授予对客户端设备的`mqtt:connect`权限。

有关如何部署和配置客户端设备身份验证组件的更多信息，请参阅以下内容：

- 配置云发现（控制台）
- 客户端设备身份验证
- 创建部署

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

如果客户端设备身份验证组件未定义授予客户端设备连接权限的客户端设备授权策略，则可能会看到此错误。

检查客户端设备身份验证组件的配置是否包括以下内容：

- 与客户端设备匹配的设备组。
- 该设备组的客户端设备授权策略，用于授予对客户端设备的`mqtt:connect`权限。

有关如何部署和配置客户端设备身份验证组件的更多信息，请参阅以下内容：

- 配置云发现（控制台）
- 客户端设备身份验证
- 创建部署

与设备阴影互动

Greengrass 核心设备可以使用组件与设备影子进行交互AWS IoT。影子是一个 JSON 文档，用于存储AWS IoT事物的当前或所需状态信息。无论设备是否已连接，阴影都可以使设备的状态可供其他AWS IoT Greengrass组件AWS IoT使用。每AWS IoT台设备都有自己的经典未命名影子。您也可以为每台设备创建多个已命名的阴影。

设备和服务可以通过使用 MQTT 和保留的 MQTT 影子主题、使用 Device Shadow RESTAPI 的 HTTP 以及 for 创建、更新和删除云阴影。AWS CLIAWS IoT

影子管理器组件使您的 Greengrass 组件能够使用本地影子服务和本地发布/订阅影子主题来创建、更新和删除本地阴影。影子管理器还管理核心设备上这些本地卷影文档的存储，并处理阴影状态信息与云阴影的同步。

您还可以使用影子管理器组件来管理连接到核心设备的客户端设备的本地阴影。要使影子管理器能够管理客户端设备影子，您需要将 MQTT 桥接组件配置为在本地 MQTT 代理和本地发布/订阅服务之间中继消息。有关更多信息，请参阅 与客户端设备影子进行交互并进行同步。

有关AWS IoT设备影子概念的更多信息，请参阅《AWS IoT开发者指南》中的 AWS IoTDevice Shadow 服务。

主题

- 与组件中的阴影交互
- 将本地设备阴影与同步 AWS IoT Core

与组件中的阴影交互

您可以开发自定义组件，包括 Lambda 函数组件，这些组件使用本地影子服务读取和修改本地影子文档和客户端设备影子文档。

自定义组件使用中的 C AWS IoT Greengrass ore IPC 库与本地影子服务进行交互。AWS IoT Device SDK影子管理器组件可在核心设备上启用本地影子服务。

要将影子管理器组件部署到 Greengrass 核心设备，请创建包含该组件的部署。aws.greengrass.ShadowManager

Note

默认情况下，部署影子管理器组件仅启用本地卷影操作。AWS IoT Greengrass要启用将核心设备阴影的阴影状态信息或客户端设备的任何阴影状态信息同步到中相应的云影文档AWS IoT Core，必须为包含`synchronize`参数的影子管理器组件创建配置更新。有关更多信息，请参阅 将本地设备阴影与同步 AWS IoT Core。

主题

- 检索和修改阴影状态
- 对阴影状态变化做出反应

检索和修改阴影状态

影子 IPC 操作检索和更新本地影子文档中的状态信息。影子管理器组件负责处理核心设备上这些影子文档的存储。

修改局部阴影状态

1. 在自定义组件的配方中添加授权策略，以允许该组件接收有关本地影子主题的消息。

   有关授权策略的示例，请参阅本地影子 IPC 授权策略示例。

2. 使用影子 IPC 操作来检索和修改影子状态信息。有关在组件代码中使用影子 IPC 操作的更多信息，请参阅与局部阴影互动。

Note

要使核心设备能够与客户端设备影子进行交互，您还必须配置和部署 MQTT 桥接组件。有关更多信息，请参阅启用影子管理器以与客户端设备通信。

对阴影状态变化做出反应

Greengrass 组件使用本地发布/订阅接口在核心设备上进行通信。要使自定义组件能够对影子状态变化做出反应，您可以订阅本地发布/订阅主题。这允许组件接收有关本地影子主题的消息，然后对这些消息进行操作。

本地影子主题使用与AWS IoT设备影子 MQTT 主题相同的格式。有关影子主题的更多信息，请参阅《AWS IoT开发者指南》中的 Device Shadow MQTT 主题。

对局部阴影状态变化做出反应

1. 在自定义组件的配方中添加访问控制策略，以允许该组件接收有关本地影子主题的消息。

   有关授权策略的示例，请参阅本地影子 IPC 授权策略示例。
2. 要在组件中启动自定义操作，请使用 `SubscribeToTopic` IPC 操作订阅要接收消息的影子主题。有关在组件代码中使用本地发布/订阅 IPC 操作的更多信息，请参阅。发布/订阅本地消息
3. 要调用 Lambda 函数，请使用事件源配置提供影子主题的名称并指定它是本地发布/订阅主题。有关创建 Lambda 函数组件的信息，请参阅。运行AWS Lambda函数

> **Note**
>
> 要使核心设备能够与客户端设备影子进行交互，您还必须配置和部署 MQTT 桥接组件。有关更多信息，请参阅启用影子管理器以与客户端设备通信。

将本地设备阴影与同步 AWS IoT Core

通过影子管理器组件AWS IoT Greengrass，可以将本地设备影子状态与同步AWS IoT Core。您必须修改影子管理器组件的配置以包含`synchronization`配置参数，并为您的设备指定AWS IoT事物名称以及要同步的阴影。

将阴影管理器配置为同步阴影时，它会同步指定阴影的所有状态更改，无论更改发生在本地卷影文档还是云影文档中。

您还可以指定阴影管理器组件是实时同步阴影还是定期同步阴影。默认情况下，影子管理器组件会实时同步阴影，因此每次更新发生AWS IoT Core时，核心设备都会发送和接收影子更新。您可以配置定期间隔以减少带宽使用量和费用。

主题

- 先决条件
- 配置影子管理器组件
- 同步局部阴影
- 影子合并冲突行为

先决条件

要与本地阴影同步AWS IoT Core，必须将 Greengrass 核心设备的策略配置为允许以下影AWS IoT子策略操作。AWS IoT Core

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

有关更多信息，请参阅下列内容：

- AWS IoT Core《AWS IoT开发者指南》中的@@ 策略操作
- AWS IoT Greengrass V2核心设备的最低AWS IoT政策
- 更新核心设备的AWS IoT政策

配置影子管理器组件

影子管理器需要一个阴影名称映射列表，才能将本地卷影文档中的阴影状态信息同步到中的AWS IoT Core云影文档。

要同步影子状态，请创建一个包含该`aws.greengrass.ShadowManager`组件的部署，并在部署的影子管理器`synchronize`配置的配置参数中指定要同步的阴影。

> **Note**
> 要使核心设备能够与客户端设备影子进行交互，您还必须配置和部署 MQTT 桥接组件。有关更多信息，请参阅启用影子管理器以与客户端设备通信。

以下示例配置更新指示影子管理器组件将以下阴影与AWS IoT Core同步：

- 核心设备的经典影子
- `MyCoreShadow`为核心设备命名的
- 名为物联网的经典影子 `MyDevice2`
- 命名的阴影`MyShadowA`，`MyShadowB`对于一个名为 IoT 的物联网事物 `MyDevice1`

此配置更新指定与阴影实时AWS IoT Core同步。如果您使用影子管理器 v2.1.0 或更高版本，则可以将影子管理器组件配置为定期同步阴影。要配置此功能，请将同步策略更改为periodic，然后为间隔指定delay以秒为单位。有关更多信息，请参阅影子管理器组件的策略配置参数。

此配置更新指定在AWS IoT Core和核心设备之间双向同步阴影。如果您使用阴影管理器 v2.2.0 或更高版本，则可以将阴影管理器组件配置为仅在一个方向上同步阴影。要配置此功能，请将同步更改direction为deviceToCloud或cloudToDevice。有关更多信息，请参阅影子管理器组件的方向配置参数。

```
{
  "strategy": {
    "type": "realTime"
  },
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadow"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": [ ]
      }
    ],
    "direction": "betweenDeviceAndCloud"
  }
}
```

同步局部阴影

当 Greengrass 核心设备连接到AWS IoT云端时，影子管理器会对您在组件配置中指定的阴影执行以下任务。行为取决于您指定的阴影同步方向配置选项。默认情况下，阴影管理器使用该betweenDeviceAndCloud选项来同步双向阴影。如果您使用阴影管理器 v2.2.0 或更高版本，则可以将核心设备配置为仅在一个方向上同步阴影，可以是cloudToDevice或。deviceToCloud

- 如果阴影同步方向配置为betweenDeviceAndCloud或cloudToDevice，则影子管理器将从中的云影文档中AWS IoT Core检索报告的状态信息。然后，它会更新本地存储的影子文档以同步设备状态。
- 如果阴影同步方向配置为betweenDeviceAndCloud或deviceToCloud，则影子管理器会将设备的当前状态发布到云影文档。

影子合并冲突行为

在某些情况下，例如当核心设备与互联网断开连接时，在影子管理器同步更改之前，本地影子服务和AWS IoT云中的影子可能会发生变化。因此，本地影子服务和AWS IoT云之间的期望状态和报告的状态会有所不同

当影子管理器同步阴影时，它会根据以下行为合并更改：

- 如果您使用低于 v2.2.0 的卷影管理器版本，或者指定了betweenDeviceAndCloud阴影同步方向，则以下行为适用：
  - 当卷影的所需状态下存在合并冲突时，影子管理器会用来自AWS IoT云的值覆盖本地卷影文档的冲突部分。
  - 当阴影的报告状态下存在合并冲突时，影子管理器会使用本地卷影文档中的值覆盖AWS IoT云中阴影的冲突部分。
- 当您指定deviceToCloud阴影同步方向时，阴影管理器会使用本地阴影文档中的值覆盖AWS IoT云中阴影的冲突部分。
- 当您指定cloudToDevice阴影同步方向时，阴影管理器会使用来自AWS IoT云的值覆盖本地卷影文档中冲突的部分。

管理 Greengrass 核心设备上的数据流

AWS IoT Greengrass流管理器可以更高效、更可靠地将大量物联网数据传输到. AWS Cloud 流管理器在将数据流导出到 AWS IoT Greengrass Core 之前先处理核心上的数据流AWS Cloud。Stream Manager 与常见的边缘场景集成，例如机器学习 (ML) 推理，在这种场景中，AWS IoT GreengrassCore 设备在将数据导出到AWS Cloud或本地存储目标之前处理和分析数据。

Stream Manager 提供了一个通用接口来简化自定义组件的开发，因此您无需构建自定义流管理功能。您的组件可以使用标准化机制来处理高容量流并管理本地数据保留策略。您可以为每个流定义存储类型、大小和数据保留策略，以控制流管理器处理和导出数据的方式。

直播管理器可在连接断断续续或连接受限的环境中运行。您可以定义带宽使用情况、超时行为以及 AWS IoT Greengrass内核在连接或断开连接时如何处理流数据。您还可以设置优先级以控制AWS IoT Greengrass核心向中导出流的顺序AWS Cloud。这使您能够比其他数据更快地处理关键数据。

您可以将流管理器配置为自动将数据导出到以AWS Cloud进行存储或进一步处理和分析。直播管理器支持导出到以下AWS Cloud目的地：

- 频道进来AWS IoT Analytics。 AWS IoT Analytics允许您对数据进行高级分析，以帮助做出业务决策和改进机器学习模型。有关更多信息，请参阅AWS IoT Analytics《用户指南》中的什么是AWS IoT Analytics？。
- 在 Amazon Kinesis Data Streams 中直播。您可以使用 Kinesis Data Streams 来聚合大量数据并将其加载到数据仓库 MapReduce 或集群中。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南 中的什么是 Amazon Kinesis Data Streams？。
- 中的资产属性AWS IoT SiteWise。 AWS IoT SiteWise允许您大规模收集、组织和分析来自工业设备的数据。有关更多信息，请参阅AWS IoT SiteWise《用户指南》中的什么是AWS IoT SiteWise？。
- 亚马逊简单存储服务 Amazon S3 中的对象。您可以使用 Amazon S3 存储和检索大量的数据。有关更多信息，请参阅什么是 Amazon S3？ 在《Amazon 简单存储服务开发者指南》中。

流管理工作流

您的物联网应用通过流管理器 SDK 与流管理器交互。

在简单的工作流程中，内AWS IoT Greengrass核上的组件消耗物联网数据，例如时间序列温度和压力指标。该组件可能会筛选或压缩数据，然后调用 Stream Manager SDK 将数据写入流管理器中的流。直播管理器可以根据您为直播定义的策略AWS Cloud自动将直播导出到。组件还可以将数据直接发送到本地数据库或存储库。

您的物联网应用程序可以包含多个用于读取或写入流的自定义组件。这些组件可以读取和写入流，以筛选、聚合和分析AWS IoT Greengrass核心设备上的数据。这使得在数据从核心传输到AWS Cloud或本地目的地之前，可以快速响应本地事件并提取有价值的信息。

首先，请将流管理器组件部署到您的AWS IoT Greengrass核心设备。在部署中，配置流管理器组件参数以定义适用于 Greengrass 核心设备上所有直播的设置。使用这些参数可以控制流管理器如何根据您的业务需求和环境限制存储、处理和导出流。

配置流管理器后，您可以创建和部署 IoT 应用程序。这些通常是在 Stream Manager SDK `StreamManagerClient` 中用于创建直播并与之交互的自定义组件。创建直播时，您可以定义每个流的策略，例如导出目的地、优先级和持久性。

要求

以下要求适用于使用流管理器：

- 除了AWS IoT Greengrass核心软件外，直播管理器还需要至少 70 MB 的内存。您的总内存需求取决于您的工作负载。
- AWS IoT Greengrass组件必须使用流管理器 SDK 才能与流管理器交互。直播管理器 SDK 有以下语言版本：
  - 适用于 Java 的直播管理器 SDK（v1.1.0 或更高版本）
  - 适用于 Node.js 的直播管理器 SDK（v1.1.0 或更高版本）
  - 适用于 Python 的直播管理器 SDK（v1.1.0 或更高版本）
- AWS IoT Greengrass组件必须在其配方中将流管理器组件 (`aws.greengrass.StreamManager`) 指定为依赖项才能使用流管理器。

> **Note**
>
> 如果您使用流管理器将数据导出到云端，则无法将流管理器组件的 2.0.7 版本升级到 v2.0.8 和 v2.0.11 之间的版本。如果您是首次部署流管理器，我们强烈建议您部署最新版本的流管理器组件。

- 如果您为直播定义AWS Cloud导出目标，则必须创建导出目标并以 G reengrass 设备角色授予访问权限。根据不同的目的地，也可能适用其他要求。有关更多信息，请参阅：
  - the section called “AWS IoT Analytics 通道”
  - the section called “Amazon Kinesis data streams”

- the section called “AWS IoT SiteWise 资产属性”
- the section called “Amazon S3 对象”

由您来负责维护这些 AWS Cloud 资源。

数据安全性

使用流管理器时，请注意以下安全注意事项。

本地数据安全性

AWS IoT Greengrass不加密核心设备上的静态数据或本地组件之间传输的流数据。

- 静态数据。流数据存储在本地存储在存储目录中。为了确保数据安全，AWS IoT Greengrass依赖文件权限和全盘加密（如果启用）。您可以使用可选的 STREAM_MANAGER_STORE_ROOT_DIR 参数指定存储目录。如果稍后将此参数更改为使用其他存储目录，AWS IoT Greengrass 不会删除以前的存储目录或其内容。
- 数据在本地传输。 AWS IoT Greengrass不加密数据源、AWS IoT Greengrass组件、流管理器 SDK 和流管理器之间本地传输的流数据。
- 传输到 AWS Cloud 的数据。流管理器导出到 AWS Cloud 的数据流使用带传输层安全性 (TLS) 的标准 AWS 服务客户端加密。

客户端身份验证

直播管理器客户端使用直播管理器 SDK 与直播管理器通信。启用客户端身份验证后，只有 Greengrass 组件可以在流管理器中与直播交互。禁用客户端身份验证后，在 Greengrass 核心设备上运行的任何进程都可以在流管理器中与流媒体交互。只有在您的业务案例需要时才应禁用身份验证。

您可以使用 STREAM_MANAGER_AUTHENTICATE_CLIENT 参数来设置客户端身份验证模式。在将流管理器组件部署到核心设备时，可以配置此参数。

	已启用	禁用
参数值	`true`（默认值和推荐值）	`false`
允许的客户端	核心设备上的 Greengrass 组件	核心设备上的 Greengrass 组件

	已启用	禁用
		Greengrass 核心设备上运行的其他进程

另请参阅

- the section called “配置流管理器”
- the section called “ StreamManagerClient 用于处理直播”
- the section called “导出支持的云端目标的配置”

创建使用流管理器的自定义组件

在自定义 Greengrass 组件中使用流管理器来存储、处理和导出 IoT 设备数据。使用本节中的过程和示例创建可与 Stream Manager 配合使用的组件配方、构件和应用程序。有关如何开发和测试组件的更多信息，请参阅创建 AWS IoT Greengrass 组件。

主题

- 定义使用流管理器的组件配方
- 在应用程序代码中Connect 流管理器

定义使用流管理器的组件配方

要在自定义组件中使用流管理器，必须将该`aws.greengrass.StreamManager`组件定义为依赖关系。您还必须提供直播管理器 SDK。完成以下任务，下载和使用您选择的语言的 Stream Manager SDK。

使用适用于 Java 的直播管理器 SDK

适用于 Java 的 Stream Manager SDK 以 JAR 文件形式提供，您可以使用它来编译组件。然后，您可以创建包含 Stream Manager SDK 的应用程序 JAR，将应用程序 JAR 定义为组件构件，并在组件生命周期中运行应用程序 JAR。

使用适用于 Java 的直播管理器 SDK

1. 下载适用于 Java 的直播管理器 SDK JAR 文件。

2. 执行以下操作之一，从 Java 应用程序和 Stream Manager SDK JAR 文件中创建组件构件：
   - 将您的应用程序构建为包含 Stream Manager SDK JAR 的 JAR 文件，然后在组件配方中运行此 JAR 文件。
   - 将流管理器 SDK JAR 定义为组件构件。在组件配方中运行应用程序时，将该工件添加到类路径中。

   您的组件配方可能类似于以下示例。此组件运行 StreamManagerS3.java 示例的修改版本，其中`StreamManagerS3.jar`包括 Stream Manager SDK JAR。

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Java",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/StreamManagerS3.jar"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar"
        }
      ]
    }
  ]
}
```

   YAML

```
---
```

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Java
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Lifecycle:
      run: java -jar {artifacts:path}/StreamManagerS3.jar
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar
```

有关如何开发和测试组件的更多信息，请参阅创建 AWS IoT Greengrass 组件。

使用适用于 Python 的直播管理器 SDK

适用于 Python 的 Stream Manager SDK 可作为源代码提供，您可以将其包含在组件中。创建 Stream Manager SDK 的 ZIP 文件，将 ZIP 文件定义为组件构件，然后在组件生命周期中安装 SDK 的要求。

使用适用于 Python 的直播管理器 SDK

1. 克隆或下载 aws-greengrass-stream-manager-sdk- python 存储库。

```
git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-
python.git
```

2. 创建一个包含该文件夹的 ZIP 文件，该`stream_manager`文件夹包含适用于 Python 的 Stream Manager SDK 的源代码。您可以将此 ZIP 文件作为组件构件提供，AWS IoT GreengrassCore 软件会在安装您的组件时解压缩。执行以下操作：

   a. 打开包含您在上一个步骤中克隆或下载的存储库的文件夹。

   ```
   cd aws-greengrass-stream-manager-sdk-python
   ```

   b. 将该`stream_manager`文件夹压缩到名为的 ZIP 文件中`stream_manager_sdk.zip`。

Linux or Unix

```
zip -rv stream_manager_sdk.zip stream_manager
```

Windows Command Prompt (CMD)

```
tar -acvf stream_manager_sdk.zip stream_manager
```

PowerShell

```
Compress-Archive stream_manager stream_manager_sdk.zip
```

c. 验证`stream_manager_sdk.zip`文件是否包含该`stream_manager`文件夹及其内容。运行以下命令列出 ZIP 文件的内容。

Linux or Unix

```
unzip -l stream_manager_sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream_manager_sdk.zip
```

该输出值应该类似于以下内容。

```
Archive:  aws-greengrass-stream-manager-sdk-python/stream_manager.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 20:45   stream_manager/
      913  02-24-2021 20:45   stream_manager/__init__.py
     9719  02-24-2021 20:45   stream_manager/utilinternal.py
     1412  02-24-2021 20:45   stream_manager/exceptions.py
     1004  02-24-2021 20:45   stream_manager/util.py
        0  02-24-2021 20:45   stream_manager/data/
   254463  02-24-2021 20:45   stream_manager/data/__init__.py
    26515  02-24-2021 20:45   stream_manager/streammanagerclient.py
---------                     -------
   294026                     8 files
```

3. 将 Stream Manager SDK 构件复制到组件的构件文件夹。除了 Stream Manager SDK 压缩文件外，您的组件还使用 SDK `requirements.txt` 的文件来安装 Stream Manager SDK 的依赖项。将 *~/greengrass-co* mponents 替换为用于本地开发的文件夹的路径。

   Linux or Unix

   ```
   cp {stream_manager_sdk.zip,requirements.txt} ~/greengrass-components/artifacts/
   com.example.StreamManagerS3Python/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0 stream_manager_sdk.zip
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0 requirements.txt
   ```

   PowerShell

   ```
   cp .\stream_manager_sdk.zip,.\requirements.txt ~\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0\
   ```

4. 创建您的组件配方。在配方中，执行以下操作：

   a. 将`stream_manager_sdk.zip`和定义`requirements.txt`为工件。

   b. 将您的 Python 应用程序定义为工件。

   c. 在安装生命周期中，从中安装 Stream Manager SDK 要求`requirements.txt`。

   d. 在运行生命周期中，将 Stream Manager SDK 附加到`PYTHONPATH`并运行您的 Python 应用程序。

   您的组件配方可能类似于以下示例。此组件运行 stream_manager_s3.py 示例。

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.StreamManagerS3Python",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "Uses stream manager to upload a file to an S3
    bucket.",
   ```

```
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "run": "export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk; python3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "run": "set \"PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk\" & py -3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
```

```
            "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
            "Unarchive": "ZIP"
          },
          {
            "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
          },
          {
            "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
          }
        ]
      }
    ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Python
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      run: |
        export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk
        python3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
```

```
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
  - Platform:
      os: windows
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      run: |
        set "PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk"
        py -3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
```

有关如何开发和测试组件的更多信息，请参阅创建 AWS IoT Greengrass 组件。

使用 Stream Manager SDK JavaScript

的 Stream Manager SDK 可作为源代码提供，您可以将其包含在组件中。JavaScript创建 Stream Manager SDK 的 ZIP 文件，将 ZIP 文件定义为组件构件，然后在组件生命周期中安装 SDK。

使用 Stream Manager SDK JavaScript

1. 克隆或下载 aws-greengrass-stream-manager-sdk-js 存储库。

   ```
   git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-js.git
   ```

2. 创建一个包含该文件夹的 ZIP 文件，该`aws-greengrass-stream-manager-sdk`文件夹包含用于 Stream Manager SDK 的源代码JavaScript。您可以将此 ZIP 文件作为组件构件提供，AWS IoT GreengrassCore 软件会在安装您的组件时解压缩。执行以下操作：

   a. 打开包含您在上一个步骤中克隆或下载的存储库的文件夹。

      ```
      cd aws-greengrass-stream-manager-sdk-js
      ```

b. 将该aws-greengrass-stream-manager-sdk文件夹压缩到名为的 ZIP 文件中stream-manager-sdk.zip。

Linux or Unix

```
zip -rv stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
```

Windows Command Prompt (CMD)

```
tar -acvf stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
```

PowerShell

```
Compress-Archive aws-greengrass-stream-manager-sdk stream-manager-sdk.zip
```

c. 验证stream-manager-sdk.zip文件是否包含该aws-greengrass-stream-manager-sdk文件夹及其内容。运行以下命令列出 ZIP 文件的内容。

Linux or Unix

```
unzip -l stream-manager-sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream-manager-sdk.zip
```

该输出值应该类似于以下内容。

```
Archive:  stream-manager-sdk.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/
      369  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/package.json
     1017  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/util.js
     8374  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/utilInternal.js
     1937  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/exceptions.js
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/
   353343  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/index.js
    22599  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/client.js
```

```
      216  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/index.js
---------                     -------
   387855                     9 files
```

3. 将 Stream Manager SDK 构件复制到组件的构件文件夹。将 *~/greengrass-co* mponents 替换为用于本地开发的文件夹的路径。

   Linux or Unix

   ```
   cp stream-manager-sdk.zip ~/greengrass-components/artifacts/
   com.example.StreamManagerS3JS/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3JS\1.0.0 stream-manager-sdk.zip
   ```

   PowerShell

   ```
   cp .\stream-manager-sdk.zip ~\greengrass-components\artifacts
   \com.example.StreamManagerS3JS\1.0.0\
   ```

4. 创建您的组件配方。在配方中，执行以下操作：

   a. 定义`stream-manager-sdk.zip`为神器。
   b. 将您的JavaScript应用程序定义为人工制品。
   c. 在安装生命周期中，从`stream-manager-sdk.zip`构件中安装 Stream Manager SDK。此`npm install`命令创建一个包含 Stream Manager SDK 及其依赖项`node_modules`的文件夹。
   d. 在运行生命周期中，将`node_modules`文件夹追加到`NODE_PATH`并运行您的JavaScript应用程序。

您的组件配方可能类似于以下示例。此组件运行 S StreamManager3 示例。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3JS",
```

```
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "run": "export NODE_PATH=$NODE_PATH:{work:path}/node_modules; node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "run": "set \"NODE_PATH=%NODE_PATH%;{work:path}/node_modules\" & node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
```

```
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3JS
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      run: |
        export NODE_PATH=$NODE_PATH:{work:path}/node_modules
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
  - Platform:
      os: windows
    Lifecycle:
```

```
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      run: |
        set "NODE_PATH=%NODE_PATH%;{work:path}/node_modules"
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
```

有关如何开发和测试组件的更多信息，请参阅创建 AWS IoT Greengrass 组件。

在应用程序代码中Connect 流管理器

要在应用程序中连接到流管理器，请`StreamManagerClient`从 Stream Manager SDK 创建一个实例。此客户端通过其默认端口 8088 或您指定的端口连接到流管理器组件。有关如何在创建安装安装安装`StreamManagerClient`安装安装的更多信息，请参阅 StreamManagerClient 用于处理直播。

Example 示例：使用默认端口Connect 直播管理器

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;

public class MyStreamManagerComponent {

    void connectToStreamManagerWithDefaultPort() {
        StreamManagerClient client = StreamManagerClientFactory.standard().build();

        // Use the client.
    }
}
```

Python

```
from stream_manager import (
    StreamManagerClient
)
```

```
def connect_to_stream_manager_with_default_port():
    client = StreamManagerClient()

    # Use the client.
```

JavaScript

```
const {
    StreamManagerClient
} = require('aws-greengrass-stream-manager-sdk');

function connectToStreamManagerWithDefaultPort() {
    const client = new StreamManagerClient();

    // Use the client.
}
```

Example 示例：使用非默认端口Connect 直播管理器

如果您为流管理器配置了非默认端口，则必须使用进程间通信从组件配置中检索端口。

> **Note**
>
> port配置参数包含您在部署流管理器STREAM_MANAGER_SERVER_PORT时指定的值。

Java

```
void connectToStreamManagerWithCustomPort() {
    EventStreamRPCConnection eventStreamRpcConnection =
 IPCUtils.getEventStreamRpcConnection();
    GreengrassCoreIPCClient greengrassCoreIPCClient = new
 GreengrassCoreIPCClient(eventStreamRpcConnection);
    List<String> keyPath = new ArrayList<>();
    keyPath.add("port");

    GetConfigurationRequest request = new GetConfigurationRequest();
    request.setComponentName("aws.greengrass.StreamManager");
    request.setKeyPath(keyPath);
    GetConfigurationResponse response =
```

```
            greengrassCoreIPCClient.getConfiguration(request,
 Optional.empty()).getResponse().get();
    String port = response.getValue().get("port").toString();
    System.out.print("Stream Manager is running on port: " + port);

    final StreamManagerClientConfig config = StreamManagerClientConfig.builder()

 .serverInfo(StreamManagerServerInfo.builder().port(Integer.parseInt(port)).build()).build()

    StreamManagerClient client =
 StreamManagerClientFactory.standard().withClientConfig(config).build();

    // Use the client.
}
```

Python

```
import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetConfigurationRequest
)
from stream_manager import (
    StreamManagerClient
)

TIMEOUT = 10

def connect_to_stream_manager_with_custom_port():
    # Use IPC to get the port from the stream manager component configuration.
    ipc_client = awsiot.greengrasscoreipc.connect()
    request = GetConfigurationRequest()
    request.component_name = "aws.greengrass.StreamManager"
    request.key_path = ["port"]
    operation = ipc_client.new_get_configuration()
    operation.activate(request)
    future_response = operation.get_response()
    response = future_response.result(TIMEOUT)
    stream_manager_port = str(response.value["port"])

    # Use port to create a stream manager client.
    stream_client = StreamManagerClient(port=stream_manager_port)

    # Use the client.
```

StreamManagerClient 用于处理直播

在 Greengrass 核心设备上运行的用户定义的 Greengrass 组件可以使用`StreamManagerClient`流管理器 SDK 中的对象在直播管理器中创建直播，然后与直播交互。当组件创建流时，它会定义流的AWS Cloud目的地、优先级以及其他导出和数据保留策略。要将数据发送到流管理器，组件会将数据附加到流中。如果为流定义了导出目标，流管理器会自动导出流。

> **Note**
>
> 通常，流管理器的客户端是用户定义的 Greengrass 组件。如果您的业务案例需要，您还可以允许在 Greengrass 内核（例如 Docker 容器）上运行的非组件进程与流管理器交互。有关更多信息，请参阅 the section called "客户端身份验证"。

本主题中的代码段向您展示客户端如何调用 `StreamManagerClient` 方式处理流。有关方法及其参数的实现详细信息，请使用指向每个代码片段后面列出的开发工具包参考的链接。

如果您在 Lambda 函数中使用流管理器，则您的 Lambda 函数应在函数处理程序之外进行实例化`StreamManagerClient`。如果在处理程序中进行实例化，该函数每次被调用时都会创建一个 `client` 并连接到流管理器。

> **Note**
>
> 如果在处理程序中实例化 `StreamManagerClient`，则必须在 `client` 完成其工作时显式调用 `close()` 方法。否则，`client` 会保持连接打开，并且另一个线程一直运行，直到脚本退出。

`StreamManagerClient` 支持以下操作：

- the section called "创建消息流"
- the section called "附加消息"
- the section called "读取消息"
- the section called "列出流"
- the section called "描述消息流"
- the section called "更新消息流"

- the section called “删除消息流”

创建消息流

要创建直播，用户定义的 Greengrass 组件会调用 create 方法并传入一个对象。MessageStreamDefinition此对象指定流的唯一名称，并定义当达到最大流大小时，流管理器应如何处理新数据。您可以使用 MessageStreamDefinition 及其数据类型（如 ExportDefinition、StrategyOnFull 和 Persistence）来定义其他流属性。其中包括：

- 自动导出的目标 AWS IoT Analytics、Kinesis Data Streams、AWS IoT SiteWise 和 Amazon S3 目标。有关更多信息，请参阅 the section called “导出支持的云端目标的配置”。
- 导出优先级。流管理器先导出优先级较高的流，然后导出优先级较低的流。
- AWS IoT Analytics、Kinesis Data Streams 和 AWS IoT SiteWise 目标的最大批处理大小和批处理间隔。当满足任一条件时，流管理器导出消息。
- T ime-to-live (TTL)。保证流数据可用于处理的时间量。您应确保数据可以在此时间段内使用。这不是删除策略。TTL 期限后可能不会立即删除数据。
- 流持久性。选择将流保存到文件系统，以便在核心重新启动期间保留数据或将流保存在内存中。
- 起始序列号。指定要在导出中用作起始消息的消息的序列号。

有关 MessageStreamDefinition 的更多信息，请参阅目标语言的开发工具包参考：

- MessageStreamDefinition在 Java 开发工具包中
- MessageStreamDefinition在 Node.js SDK 中
- MessageStreamDefinition在 Python 软件开发工具包中

> **Note**
>
> StreamManagerClient 还提供了一个可用于将流导出到 HTTP 服务器的目标。此目标仅用于测试目的。其不稳定，或不支持在生产环境中使用。

创建流后，您的 Greengrass 组件可以将消息附加到流中以发送数据以供导出，并从流中读取消息以进行本地处理。您创建的流数量取决于您的硬件功能和业务案例。一种策略是为 AWS IoT Analytics 或 Kinesis 数据流中的每个目标通道创建一个流（尽管您可以为一个流定义多个目标）。流具有持久的使用寿命。

要求

此操作具有以下要求：

- Stream Manager SDK 最低版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

示例

以下代码段创建一个名为 StreamName 的流。它定义了 MessageStreamDefinition 中的流属性和从属的数据类型。

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 AWS Cloud.
            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.
```

Python SDK 参考：创建消息流 | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    client.createMessageStream(
            new MessageStreamDefinition()
                    .withName("StreamName") // Required.
                    .withMaxSize(268435456L)    // Default is 256 MB.
                    .withStreamSegmentSize(16777216L)    // Default is 16 MB.
                    .withTimeToLiveMillis(null)    // By default, no TTL is enabled.
                    .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)    //
 Required.
                    .withPersistence(Persistence.File)    // Default is File.
                    .withFlushOnWrite(false)    // Default is false.
                    .withExportDefinition(    // Optional. Choose where/how the
 stream is exported to the AWS Cloud.
                            new ExportDefinition()
                                    .withKinesis(null)
                                    .withIotAnalytics(null)
                                    .withIotSitewise(null)
                                    .withS3(null)
                    )

    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 开发工具包参考：createMessageStream| 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 AWS Cloud.
          new ExportDefinition()
            .withKinesis(null)
            .withIotAnalytics(null)
            .withIotSiteWise(null)
            .withS3(null)
        )
    );
  } catch (e) {
    // Properly handle errors.
  }
});
client.onError((err) => {
  // Properly handle connection errors.
  // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 参考：createMessageStream| MessageStreamDefinition

有关配置导出目标的更多信息，请参阅the section called “导出支持的云端目标的配置”。

附加消息

要将数据发送到流管理器进行导出，Greengrass 组件会将数据附加到目标流。导出目标决定要传递给此方法的数据类型。

要求

此操作具有以下要求：

- Stream Manager SDK 最低版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

示例

AWS IoT Analytics 或 Kinesis Data Streams 导出目标

以下代码段将消息附加到名为 `StreamName` 的流。对于我们AWS IoT Analytics的 Kinesis Data Streams 目标，你的 Greengrass 组件会附加一大堆数据。

此代码段具有以下要求：

- Stream Manager SDK 最低版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

Python

```
client = StreamManagerClient()

try:
    sequence_number = client.append_message(stream_name="StreamName",
 data=b'Arbitrary bytes data')
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python 开发工具包参考：append_message

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
 array".getBytes());
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 开发工具包参考：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 开发工具包参考：appendMessage

AWS IoT SiteWise 导出目标

以下代码段将消息附加到名为 StreamName 的流。对于AWS IoT SiteWise目的地，您的 Greengrass 组件会附加一个序列化对象。PutAssetPropertyValueEntry有关更多信息，请参阅 the section called “导出到 AWS IoT SiteWise”。

> **Note**
>
> 当您将数据发送到 AWS IoT SiteWise 时，数据必须满足 BatchPutAssetPropertyValue 操作的要求。有关更多信息，请参阅《AWS IoT SiteWise API 参考》中的 BatchPutAssetPropertyValue。

此代码段具有以下要求：

- Stream Manager SDK 最低版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages and also needs timestamps
 not earlier
    # than 10 minutes in the past. Add some randomness to time and offset.

    # Note: To create a new asset property data, you should use the classes defined
 in the
    # greengrasssdk.stream_manager module.

    time_in_nanos = TimeInNanos(
        time_in_seconds=calendar.timegm(time.gmtime()) - random.randint(0, 60),
 offset_in_nanos=random.randint(0, 10000)
    )
    variant = Variant(double_value=random.random())
```

```
    asset = [AssetPropertyValue(value=variant, quality=Quality.GOOD,
 timestamp=time_in_nanos)]
    putAssetPropertyValueEntry =
 PutAssetPropertyValueEntry(entry_id=str(uuid.uuid4()),
 property_alias="PropertyAlias", property_values=asset)
    sequence_number = client.append_message(stream_name="StreamName",
 Util.validate_and_serialize_to_json_bytes(putAssetPropertyValueEntry))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 参考：追加消息 | PutAssetPropertyValueEntry

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    Random rand = new Random();
    // Note: To create a new asset property data, you should use the classes defined
 in the
    // com.amazonaws.greengrass.streammanager.model.sitewise package.
    List<AssetPropertyValue> entries = new ArrayList<>() ;

    // IoTSiteWise requires unique timestamps in all messages and also needs
 timestamps not earlier
    // than 10 minutes in the past. Add some randomness to time and offset.
    final int maxTimeRandomness = 60;
    final int maxOffsetRandomness = 10000;
    double randomValue = rand.nextDouble();
    TimeInNanos timestamp = new TimeInNanos()
            .withTimeInSeconds(Instant.now().getEpochSecond() -
 rand.nextInt(maxTimeRandomness))
            .withOffsetInNanos((long) (rand.nextInt(maxOffsetRandomness)));
    AssetPropertyValue entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);
    entries.add(entry);

    PutAssetPropertyValueEntry putAssetPropertyValueEntry = new
 PutAssetPropertyValueEntry()
```

```
            .withEntryId(UUID.randomUUID().toString())
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues(entries);
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 参考：[附加消息 | 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 SDK 参考：追加消息 | PutAssetPropertyValueEntry

Amazon S3 导出目标

以下代码段将导出任务附加到名为 `StreamName` 的流。对于亚马逊 S3 目的地，您的 Greengrass 组件会附加一个`S3ExportTaskDefinition`序列化对象，其中包含有关源输入文件和目标 Amazon S3 对象的信息。如果指定的对象不存在，流管理器会为您创建。有关更多信息，请参阅 the section called “导出到 Amazon S3”。

此代码段具有以下要求：

- Stream Manager SDK 最低版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

Python

```
client = StreamManagerClient()

try:
    # Append an Amazon S3 Task definition and print the sequence number.
    s3_export_task_definition = S3ExportTaskDefinition(input_url="URLToFile",
 bucket="BucketName", key="KeyName")
    sequence_number = client.append_message(stream_name="StreamName",
 Util.validate_and_serialize_to_json_bytes(s3_export_task_definition))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 参考：追加消息 | S3 ExportTaskDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
```

```
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

[Java SDK 参考：附加消息 | S3 ExportTaskDefinition]

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
     // Append an Amazon S3 export task definition and print the sequence number.
     const taskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(taskDefinition)));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

[Node.js SDK 参考：追加消息 | S3 ExportTaskDefinition]

读取消息

从流读取消息。

要求

此操作具有以下要求：

- Stream Manager SDK 最低版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

示例

以下代码段读取名为 StreamName 的流中的消息。read 方法接受一个可选 ReadMessagesOptions 对象，该对象指定要开始读取的序列号、要读取的最小数量和最大数量以及读取消息的超时。

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.
```

Python SDK 参考：阅读消息 | ReadMessagesOptions

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    List<Message> messages = client.readMessages("StreamName",
```

```
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                    // Try to read from sequence number 100 or greater. By default
 this is 0.
                    .withDesiredStartSequenceNumber(100L)
                    // Try to read 10 messages. If 10 messages are not available,
 then NotEnoughMessagesException is raised. By default, this is 1.
                    .withMinMessageCount(10L)
                    // Accept up to 100 messages. By default this is 1.
                    .withMaxMessageCount(100L)
                    // Try to wait at most 5 seconds for the min_messsage_count to
 be fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                    .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 开发工具包参考：[阅读消息 | 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 SDK 参考：[阅读消息 | ReadMessagesOptions]

列出流

在流管理器中获取流列表。

要求

此操作具有以下要求：

- Stream Manager SDK 最低版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

示例

以下代码段获取流管理器中的流列表（按名称）。

Python

```
client = StreamManagerClient()

try:
    stream_names = client.list_streams()
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python 开发工具包参考：[list_streams]

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 开发工具包参考：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 开发工具包参考：listStreams

描述消息流

获取有关流的元数据，包括流定义、大小和导出状态。

要求

此操作具有以下要求：

- Stream Manager SDK 最低版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

示例

以下代码段获取有关名为 `StreamName` 的流的元数据，包括流的定义、大小和导出程序状态。

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.
```

Python 开发工具包参考：describe_message_stream

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    MessageStreamInfo description = client.describeMessageStream("StreamName");
    String lastErrorMessage =
 description.getExportStatuses().get(0).getErrorMessage();
    if (lastErrorMessage != null && !lastErrorMessage.equals("")) {
        // The last export of export destination 0 failed with some error.
        // Here is the last sequence number that was successfully exported.
        description.getExportStatuses().get(0).getLastExportedSequenceNumber();
    }

    if (description.getStorageStatus().getNewestSequenceNumber() >
            description.getExportStatuses().get(0).getLastExportedSequenceNumber())
 {
        // The end of the stream is ahead of the last exported sequence number.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
```

```
}
```

Java 开发工具包参考：describeMessageStream

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const description = await client.describeMessageStream("StreamName");
        const lastErrorMessage = description.exportStatuses[0].errorMessage;
        if (lastErrorMessage) {
            // The last export of export destination 0 failed with some error.
            // Here is the last sequence number that was successfully exported.
            description.exportStatuses[0].lastExportedSequenceNumber;
        }

        if (description.storageStatus.newestSequenceNumber >
            description.exportStatuses[0].lastExportedSequenceNumber) {
            // The end of the stream is ahead of the last exported sequence number.
        }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 参考：describeMessageStream

更新消息流

更新现有流的属性。如果流创建后您的要求发生变化，则可能需要更新流。例如：

- 为 AWS Cloud 目标添加新的导出配置。
- 增加流的最大大小以更改数据的导出或保留方式。例如，将流大小与您在完整设置下的策略相结合，可能会导致数据在流管理器处理之前被删除或拒绝。
- 暂停然后恢复导出；例如，如果导出任务运行时间较长，而您想对上传数据进行配给。

您的 Greengrass 组件遵循以下高级流程来更新直播：

1. 获取流的描述。
2. 更新相应 MessageStreamDefinition 和从属对象的目标属性。
3. 传入更新后的 MessageStreamDefinition。请务必包含更新后的流的完整对象定义。未定义属性将恢复为默认值。

   可以指定要在导出中用作起始消息的消息的序列号。

要求

此操作具有以下要求：

- Stream Manager SDK 最低版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

示例

以下代码段更新名为 StreamName 的流。它会更新导出到 Kinesis Data Streams 的流的多个属性。

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.
```

Python 开发工具包参考：updateMessageStream| 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 AWS
 Cloud.
                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.
}
```

Java SDK 参考：更新消息流 | 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 AWS
 Cloud.
                    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 SDK 参考：updateMessageStream| MessageStreamDefinition

更新流时的限制

更新流时，有以下限制。除非在以下列表中注明，否则更新会立即生效。

- 无法更新流的持久性。要更改此行为，请删除该流然后创建一个流以定义新的持久性策略。

- 只有在以下条件下，您才能更新流的最大大小：
  - 最大大小必须大于或等于流的当前大小。要查找此信息，请描述流，然后检查返回的 `MessageStreamInfo` 对象的存储状态。
  - 最大大小必须大于或等于流的段大小。
- 可以将流的段大小更新为小于流的最大大小的值。更新的设置将应用于新的段。
- 生存时间（TTL）属性的更新将应用于新的追加操作。如果您减小此值，流管理器也可能会删除超过 TTL 的现有段。
- 对全属性策略的更新将应用于新的追加操作。如果您将策略设置为覆盖最旧的数据，则流管理器还可能根据新设置覆盖现有段。
- 写入时刷新属性的更新将应用于新消息。
- 导出配置的更新将应用于新的导出。更新请求必须包含您想要支持的所有导出配置。否则，流管理器会将其删除。
  - 更新导出配置时，请指定目标导出配置的标识符。
  - 要添加导出配置，请为新的导出配置指定唯一标识符。
  - 要删除导出配置，请省略导出配置。
- 要更新流中导出配置的起始序列号，必须指定一个小于最新序列号的值。要查找此信息，请描述流，然后检查返回的 `MessageStreamInfo` 对象的存储状态。

删除消息流

删除流。删除流时，流的所有存储数据将从磁盘中删除。

要求

此操作具有以下要求：

- Stream Manager SDK 最低版本：Python：1.1.0 | Java：1.1.0 | Node.js：1.1.0

示例

以下代码段删除名为 `StreamName` 的流。

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.
```

Python 开发工具包参考：deleteMessageStream

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 开发工具包参考：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 SDK 参考：deleteMessageStream

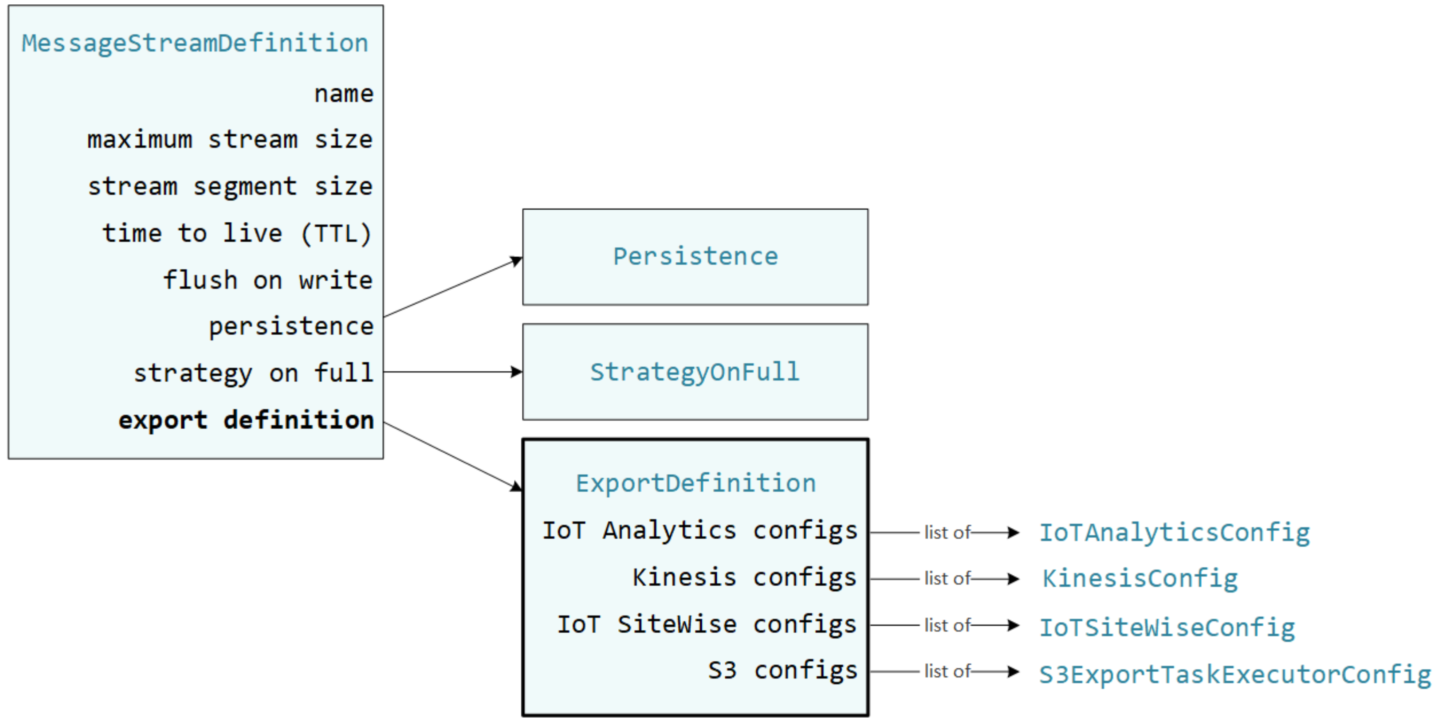
另请参阅

- 管理 Greengrass 核心设备上的数据流

- 配置 AWS IoT Greengrass 流管理器
- 导出支持的 AWS Cloud 目标的配置
- StreamManagerClient在直播管理器 SDK 参考中：
  - Python
  - Java
  - Node.js

导出支持的 AWS Cloud 目标的配置

直播管理器 SDK 中StreamManagerClient使用用户定义的 Greengrass 组件与直播管理器进行交互。当组件创建流或更新流时，它会传递一个表示流属性的MessageStreamDefinition对象，包括导出定义。ExportDefinition 对象包含为流定义的导出配置。流管理器使用这些导出配置来确定将流导出到何处以及如何导出。



您可以为一个流定义零个或多个导出配置，包括针对单个目标类型的多个导出配置。例如，您可以将一个流导出到两个 AWS IoT Analytics 通道和一个 Kinesis 数据流。

对于失败的导出尝试，流管理器会持续重试将数据导出到 AWS Cloud，间隔不超过五分钟。重试次数没有最大限制。

Note

`StreamManagerClient` 还提供了一个可用于将流导出到 HTTP 服务器的目标。此目标仅用于测试目的。其不稳定，或不支持在生产环境中使用。

受支持的 AWS Cloud 的目标

- AWS IoT Analytics 通道
- Amazon Kinesis data streams
- AWS IoT SiteWise 资产属性
- Amazon S3 对象

由您来负责维护这些 AWS Cloud 资源。

AWS IoT Analytics 通道

流管理器支持自动导出到 AWS IoT Analytics。AWS IoT Analytics 允许您对数据进行高级分析，以帮助做出业务决策和改进机器学习模型。有关更多信息，请参阅 AWS IoT Analytics 用户指南中的什么是 AWS IoT Analytics？。

在 Stream Manager SDK 中，您的 Greengrass 组件使用来定义`IoTAnalyticsConfig`此目标类型的导出配置。有关更多信息，请参阅目标语言的开发工具包参考：

- Python 软件开发工具包AnalyticsConfig中的@@ 物联网
- Java SDK AnalyticsConfig 中的@@ 物联网
- Node.js SDK AnalyticsConfig 中的@@ 物联网

要求

此导出目的地具有以下要求：

- 中的目标频道AWS IoT Analytics必须与 Greengrass 核心设备相同AWS 账户。AWS 区域
- 授权核心设备与AWS服务 必须允许对目标通道的 `iotanalytics:BatchPutMessage` 权限。例如：

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Resource": [
        "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
        "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
      ]
    }
  ]
}
```

您可以授予对资源的具体或条件访问权限（例如，通过使用通配符 * 命名方案）。有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

导出到 AWS IoT Analytics

要创建导出到的流AWS IoT Analytics，Greengrass 组件会创建一个包含一个或多个对象的导出定义的流。IoTAnalyticsConfig此对象定义导出设置，例如目标频道、批次大小、批次间隔和优先级。

当你的 Greengrass 组件从设备接收数据时，它们会将包含大量数据的消息附加到目标流。

然后，流管理器根据流的导出配置中定义的批处理设置和优先级导出数据。

Amazon Kinesis data streams

流管理器支持自动导出到 Amazon Kinesis Data Streams。Kinesis Data Streams 通常用于聚合大量数据并将其加载到数据仓库 MapReduce 或集群中。有关更多信息，请参阅 Amazon Kinesis 开发人员指南中的什么是 Amazon Kinesis Data Streams？。

在 Stream Manager SDK 中，您的 Greengrass 组件使用来定义KinesisConfig此目标类型的导出配置。有关更多信息，请参阅目标语言的开发工具包参考：

- KinesisConfig在 Python 软件开发工具包中
- KinesisConfig在 Java 开发工具包中
- KinesisConfig在 Node.js 软件开发工具包中

要求

此导出目的地具有以下要求：

- Kinesis Data Streams 中的目标流必须与 Greengrass 核心设备AWS 账户相同AWS 区域。
- 授权核心设备与AWS服务 必须允许对数据流的 `kinesis:PutRecords` 权限。例如：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:region:account-id:stream/stream_1_name",
        "arn:aws:kinesis:region:account-id:stream/stream_2_name"
      ]
    }
  ]
}
```

  您可以授予对资源的具体或条件访问权限（例如，通过使用通配符 * 命名方案）。有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

导出到 Kinesis Data Streams

要创建导出到 Kinesis Data Streams 的流，您的 Greengrass 组件会创建一个包含一个或多个对象的导出定义的流。`KinesisConfig`此对象定义导出设置，例如目标数据流、批次大小、批次间隔和优先级。

当你的 Greengrass 组件从设备接收数据时，它们会将包含大量数据的消息附加到目标流。然后，流管理器根据流的导出配置中定义的批处理设置和优先级导出数据。

流管理器会为上传到 Amazon Kinesis 的每条记录生成一个唯一的随机 UUID 作为分区键。

AWS IoT SiteWise 资产属性

流管理器支持自动导出到 AWS IoT SiteWise。AWS IoT SiteWise 可让您能够大规模收集、组织和分析来自工业设备的数据。有关更多信息，请参阅 AWS IoT SiteWise 用户指南中的什么是 AWS IoT SiteWise？。

在 Stream Manager SDK 中，您的 Greengrass 组件使用来定义`IoTSiteWiseConfig`此目标类型的导出配置。有关更多信息，请参阅目标语言的开发工具包参考：

- Python 软件开发工具包SiteWiseConfig中的@@ 物联网
- Java SDK SiteWiseConfig 中的@@ 物联网
- Node.js SDK SiteWiseConfig 中的@@ 物联网

> **Note**
>
> AWS还提供了AWS IoT SiteWise组件，这些组件提供了一个预先构建的解决方案，可用于流式传输来自 OPC-UA 来源的数据。有关更多信息，请参阅 物联网 SiteWise OPC-UA 采集器。

要求

此导出目的地具有以下要求：

- 中的目标资产属性AWS IoT SiteWise必须与 Greengrass 核心设备相同AWS 账户。AWS 区域

  > **Note**
  >
  > 有关AWS IoT SiteWise支持的列表，请参阅《AWS 区域AWS一般参考》中的AWS IoT SiteWise终端节点和配额。

- 授权核心设备与AWS服务 必须允许对目标资产属性的 `iotsitewise:BatchPutAssetPropertyValue` 权限。以下示例策略使用 `iotsitewise:assetHierarchyPath` 条件键来授予对目标根资产及其子项的访问权限。您可以从策略中删除 `Condition`，以允许访问您的所有 AWS IoT SiteWise 资产。

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

```
        "Effect": "Allow",
        "Action": "iotsitewise:BatchPutAssetPropertyValue",
        "Resource": "*",
        "Condition": {
          "StringLike": {
            "iotsitewise:assetHierarchyPath": [
              "/root node asset ID",
              "/root node asset ID/*"
            ]
          }
        }
      }
    ]
}
```

您可以授予对资源的具体或条件访问权限（例如，通过使用通配符 * 命名方案）。有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

有关重要的安全信息，请参阅《AWS IoT SiteWise用户指南》中的 BatchPutAssetPropertyValue 授权。

导出到 AWS IoT SiteWise

要创建导出到的流AWS IoT SiteWise，Greengrass 组件会创建一个包含一个或多个对象的导出定义的流。IoTSiteWiseConfig此对象定义导出设置，例如批次大小、批次间隔和优先级。

当你的 Greengrass 组件从设备接收资产属性数据时，它们会将包含这些数据的消息附加到目标流。消息是 JSON 序列化的 PutAssetPropertyValueEntry 对象，其中包含一个或多个资产属性的属性值。有关更多信息，请参阅为AWS IoT SiteWise导出目标追加消息。

> **Note**
>
> 当您将数据发送到 AWS IoT SiteWise 时，数据必须满足 BatchPutAssetPropertyValue 操作的要求。有关更多信息，请参阅《AWS IoT SiteWise API 参考》中的 BatchPutAssetPropertyValue。

然后，流管理器根据流的导出配置中定义的批处理设置和优先级导出数据。

您可以调整直播管理器设置和 Greengrass 组件逻辑来设计导出策略。例如：

- 对于近乎实时的导出，请设置较低的批量大小和间隔设置，并在收到数据时将数据追加到流中。
- 为了优化批处理、缓解带宽限制或最大限度地降低成本，Greengrass 组件可以在 timestamp-quality-value 将数据追加到流之前，为单个资产属性汇集接收到的 (TQV) 数据点。一种策略是在一条消息中批量输入多达 10 种不同的财产资产组合或属性别名，而不是为同一个属性发送多个条目。这有助于流管理器保持在AWS IoT SiteWise配额范围内。

Amazon S3 对象

流管理器支持自动导出到 Amazon S3。您可以使用 Amazon S3 存储和检索大量的数据。有关更多信息，请参阅 Amazon Simple Storage Service 开发人员指南中的什么是 Amazon S3？。

在 Stream Manager SDK 中，您的 Greengrass 组件使用来定义`S3ExportTaskExecutorConfig`此目标类型的导出配置。有关更多信息，请参阅目标语言的开发工具包参考：

- Python 软件开发工具包ExportTaskExecutorConfig中的 S@@ 3
- Java 开发工具包ExportTaskExecutorConfig中的 S3
- Node.js 软件开发工具包ExportTaskExecutorConfig中的 S3

要求

此导出目的地具有以下要求：

- 目标 Amazon S3 存储桶必须与 Greengrass 核心设备AWS 账户相同。
- 如果在 Greengrass 容器模式下运行的 Lambda 函数将输入文件写入输入文件目录，则必须将该目录作为具有写入权限的卷挂载到容器中。这样可以确保将文件写入根文件系统，并且对在容器外部运行的流管理器组件可见。
- 如果 Docker 容器组件将输入文件写入输入文件目录，则必须将该目录作为具有写入权限的卷挂载到容器中。这样可以确保将文件写入根文件系统，并且对在容器外部运行的流管理器组件可见。
- 授权核心设备与AWS服务 必须允许对目标存储桶具有以下权限。例如：

```
{
  "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/*"
        ]
      }
    ]
  }
```

您可以授予对资源的具体或条件访问权限（例如，通过使用通配符 * 命名方案）。有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

导出到 Amazon S3

要创建导出到 Amazon S3 的流，您的 Greengrass 组件使用`S3ExportTaskExecutorConfig`该对象来配置导出策略。该策略定义了导出设置，例如分段上传阈值和优先级。对于 Amazon S3 导出，流管理器会上传它从核心设备上的本地文件中读取的数据。要启动上传，您的 Greengrass 组件会将导出任务附加到目标流。导出任务包含有关输入文件和目标 Amazon S3 对象的信息。直播管理器按照任务附加到流中的顺序运行任务。

> **Note**
>
> 目标桶必须已存在于您的 AWS 账户 中。如果指定密钥的对象不存在，则流管理器会为您创建该对象。

Stream Manager 使用分段上传阈值属性、最小分段大小设置和输入文件的大小来确定如何上传数据。分段上传阈值必须大于或等于最小分段大小。如果要并行上传数据，则可以创建多个流。

指定您的目标 Amazon S3 对象的密钥可以在`!{timestamp:`*`value`*`}`占位符中包含有效的 Java DateTimeFormatter 字符串。您可以使用这些时间戳占位符根据输入文件数据的上传时间对 Amazon S3 中的数据进行分区。例如，以下键名解析为诸如 `my-key/2020/12/31/data.txt` 之类的值。

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

Note

如果要监控流的导出状态，请先创建一个状态流，然后将导出流配置为使用该状态流。有关更多信息，请参阅 the section called “监控导出任务”。

管理输入数据

您可以编写代码，供物联网应用用来管理输入数据的生命周期。以下示例工作流程显示了如何使用 Greengrass 组件来管理这些数据。

1. 本地进程从设备或外围设备接收数据，然后将数据写入核心设备上目录中的文件。这些是流管理器的输入文件。
2. Greengrass 组件会扫描该目录，并在创建新文件时将导出任务附加到目标流。该任务是一个 JSON 序列化 `S3ExportTaskDefinition` 对象，用于指定输入文件的 URL、目标 Amazon S3 存储桶和密钥以及可选的用户元数据。
3. 流管理器读取输入文件并按照附加任务的顺序将数据导出到 Amazon S3。目标桶必须已存在于您的 AWS 账户 中。如果指定密钥的对象不存在，则流管理器会为您创建该对象。
4. Greengrass 组件从状态流中读取消息以监控导出状态。导出任务完成后，Greengrass 组件可以删除相应的输入文件。有关更多信息，请参阅 the section called “监控导出任务”。

监控导出任务

您可以编写代码，让物联网应用程序监控您的 Amazon S3 导出状态。您的 Greengrass 组件必须创建状态流，然后配置导出流以将状态更新写入状态流。单个状态流可以接收来自导出到 Amazon S3 的多个流的状态更新。

首先，创建一个流以用作状态流。您可以为流配置大小和保留策略，以控制状态消息的生命周期。例如：

- 如果您不想存储状态消息，请将 `Persistence` 设置为 `Memory`。
- 将 `StrategyOnFull` 设置为 `OverwriteOldestData`，这样新的状态消息就不会丢失。

然后，创建或更新导出流以使用状态流。具体而言，设置流 `S3ExportTaskExecutorConfig` 导出配置的状态配置属性。此设置告诉流管理器将有关导出任务的状态消息写入状态流。在 `StatusConfig` 对象中，指定状态流的名称和详细程度。以下支持的值范围从最低 verbose (`ERROR`) 到最长 verbose (`TRACE`) 不等。默认值为 `INFO`。

- ERROR
- WARN
- INFO
- DEBUG
- TRACE

以下示例工作流程显示了 Greengrass 组件如何使用状态流来监控导出状态。

1. 如前面的工作流程所述，Greengrass 组件将导出任务附加到配置为将有关导出任务的状态消息写入状态流的流。附加操作返回一个表示任务 ID 的序列号。
2. Greengrass 组件按顺序从状态流读取消息，然后根据流名称和任务 ID 或消息上下文中的导出任务属性筛选消息。例如，Greengrass 组件可以按导出任务的输入文件 URL 进行筛选，该文件由消息上下文`S3ExportTaskDefinition`中的对象表示。

   以下状态代码指示导出任务已达到完成状态：

   - `Success`。上传已成功完成。
   - `Failure`。流管理器遇到错误，例如，指定的存储桶不存在。解决问题后，您可以再次将导出任务追加到流中。
   - `Canceled`。 由于流或导出定义已删除，或者任务的 time-to-live (TTL) 期限已过期，该任务已停止。

   > **Note**
   >
   > 该任务的状态也可能为 `InProgress` 或 `Warning`。当事件返回不影响任务执行的错误时，流管理器会发出警告。例如，如果无法清理部分上传，则会返回警告。
3. 导出任务完成后，Greengrass 组件可以删除相应的输入文件。

以下示例显示了 Greengrass 组件如何读取和处理状态消息。

Python

```
import time
from stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
```

```
    StatusLevel,
    StatusMessage,
    StreamManagerClient,
)
from stream_manager.util import Util

client = StreamManagerClient()

try:
    # Read the statuses from the export status stream
    is_file_uploaded_to_s3 = False
    while not is_file_uploaded_to_s3:
        try:
            messages_list = client.read_messages(
                "StatusStreamName", ReadMessagesOptions(min_message_count=1,
 read_timeout_millis=1000)
            )
            for message in messages_list:
                # Deserialize the status message first.
                status_message = Util.deserialize_json_bytes_to_obj(message.payload,
 StatusMessage)

                # Check the status of the status message. If the status is
 "Success",
                # the file was successfully uploaded to S3.
                # If the status was either "Failure" or "Cancelled", the server was
 unable to upload the file to S3.
                # We will print the message for why the upload to S3 failed from the
 status message.
                # If the status was "InProgress", the status indicates that the
 server has started uploading
                # the S3 task.
                if status_message.status == Status.Success:
                    logger.info("Successfully uploaded file at path " + file_url + "
 to S3.")
                    is_file_uploaded_to_s3 = True
                elif status_message.status == Status.Failure or
 status_message.status == Status.Canceled:
                    logger.info(
                        "Unable to upload file at path " + file_url + " to S3.
 Message: " + status_message.message
                    )
                    is_file_uploaded_to_s3 = True
            time.sleep(5)
```

```
        except StreamManagerException:
            logger.exception("Exception while running")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 参考：[阅读消息 | StatusMessage]

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.StreamManagerClientFactory;
import com.amazonaws.greengrass.streammanager.client.utils.ValidateAndSerialize;
import com.amazonaws.greengrass.streammanager.model.ReadMessagesOptions;
import com.amazonaws.greengrass.streammanager.model.Status;
import com.amazonaws.greengrass.streammanager.model.StatusConfig;
import com.amazonaws.greengrass.streammanager.model.StatusLevel;
import com.amazonaws.greengrass.streammanager.model.StatusMessage;

 try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    try {
        boolean isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                List<Message> messages = client.readMessages("StatusStreamName",
                    new
 ReadMessagesOptions().withMinMessageCount(1L).withReadTimeoutMillis(1000L));
                for (Message message : messages) {
                    // Deserialize the status message first.
                    StatusMessage statusMessage =
 ValidateAndSerialize.deserializeJsonBytesToObj(message.getPayload(),
 StatusMessage.class);
                    // Check the status of the status message. If the status is
 "Success", the file was successfully uploaded to S3.
                    // If the status was either "Failure" or "Canceled", the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
```

```
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (Status.Success.equals(statusMessage.getStatus())) {
                        System.out.println("Successfully uploaded file at path " +
 FILE_URL + " to S3.");
                        isS3UploadComplete = true;
                     } else if (Status.Failure.equals(statusMessage.getStatus()) ||
 Status.Canceled.equals(statusMessage.getStatus())) {
                        System.out.println(String.format("Unable to upload file at
 path %s to S3. Message %s",

 statusMessage.getStatusContext().getS3ExportTaskDefinition().getInputUrl(),
                            statusMessage.getMessage()));
                        sS3UploadComplete = true;
                    }
                }
            } catch (StreamManagerException ignored) {
            } finally {
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                Thread.sleep(5000);
            }
        } catch (e) {
        // Properly handle errors.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 开发工具包参考：阅读消息 | StatusMessage

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require(*'aws-greengrass-stream-manager-sdk'*);

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        let isS3UploadComplete = false;
        while (!isS3UploadComplete) {
```

```
            try {
                // Read the statuses from the export status stream
                const messages = await c.readMessages("StatusStreamName",
                    new ReadMessagesOptions()
                        .withMinMessageCount(1)
                        .withReadTimeoutMillis(1000));

                messages.forEach((message) => {
                    // Deserialize the status message first.
                    const statusMessage =
 util.deserializeJsonBytesToObj(message.payload, StatusMessage);
                    // Check the status of the status message. If the status is
 'Success', the file was successfully uploaded to S3.
                    // If the status was either 'Failure' or 'Cancelled', the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (statusMessage.status === Status.Success) {
                        console.log(`Successfully uploaded file at path ${FILE_URL}
 to S3.`);
                        isS3UploadComplete = true;
                    } else if (statusMessage.status === Status.Failure ||
 statusMessage.status === Status.Canceled) {
                        console.log(`Unable to upload file at path ${FILE_URL} to
 S3. Message: ${statusMessage.message}`);
                        isS3UploadComplete = true;
                    }
                });
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                await new Promise((r) => setTimeout(r, 5000));
            } catch (e) {
                // Ignored
            }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
```

```
});
```

Node.js SDK 参考：阅读消息 | StatusMessage

配置 AWS IoT Greengrass 流管理器

在 Greengrass 核心设备上，流管理器可以存储、处理和导出物联网设备数据。流管理器提供了用于配置运行时设置的参数。这些设置适用于 Greengrass 核心设备上的所有直播。部署组件时，您可以使用 AWS IoT Greengrass控制台或 API 来配置流管理器设置。更改将在部署完成后生效。

流管理器参数

流管理器提供了以下参数，您可以在将组件部署到核心设备时配置这些参数。所有参数都是可选的。

存储目录

参数名称: `STREAM_MANAGER_STORE_ROOT_DIR`

用于存储直播的本地文件夹的绝对路径。此值必须以正斜杠开头（例如，`/data`）。

您必须指定现有文件夹，并且运行流管理器组件的系统用户必须具有读取和写入该文件夹的权限。例如，您可以运行以下命令来创建和配置文件夹`/var/greengrass/streams`，将其指定为流管理器根文件夹。这些命令允许默认系统用户读取和写入此文件夹。`ggc_user`

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

有关保护流数据安全的信息，请参阅the section called “本地数据安全性”。

默认值：*`/greengrass/v2`*`/work/aws.greengrass.StreamManager`

服务器端口

参数名称: `STREAM_MANAGER_SERVER_PORT`

用于与流管理器通信的本地端口号。默认值为 `8088`。

您可以指定`0`使用随机可用端口。

验证客户端身份

参数名称: `STREAM_MANAGER_AUTHENTICATE_CLIENT`

指示客户端是否必须通过身份验证才能与流管理器交互。客户端和直播管理器之间的所有交互都由直播管理器 SDK 控制。此参数决定哪些客户端可以调用 Stream Manager SDK 来处理直播。有关更多信息，请参阅 the section called “客户端身份验证”。

有效值为 `true` 或 `false`。默认值为 `true`（推荐）。

- `true`。 仅允许 Greengrass 组件作为客户端。组件使用内部AWS IoT Greengrass核心协议通过流管理器 SDK 进行身份验证。
- `false`。 允许在 C AWS IoT Greengrass ore 上运行的任何进程成为客户端。`false`除非您的业务案例要求，否则请勿将该值设置为。例如，`false`仅当核心设备上的非组件进程必须直接与流管理器通信时才使用。

最大带宽

参数名称: `STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

可用于导出数据的平均最大带宽（以千位/秒为单位）。默认设置允许无限制使用可用带宽。

线程池大小

参数名称: `STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

可用于导出数据的最大活动线程数。默认值为 `5`。

最佳大小取决于您的硬件、流的量和计划的导出流数量。如果导出速度较慢，您可以调整此设置以找出适合您的硬件和业务案例的最佳大小。核心设备硬件的 CPU 和内存是限制因素。首先，您可以尝试将此值设置为等于设备上的处理器核心数。

请注意，不要设置大于硬件可以支持的大小。每个流都消耗硬件资源，因此请尽量限制受限设备上的导出流数量。

JVM 参数

参数名称: `JVM_ARGS`

在启动时传递给流管理器的自定义 Java 虚拟机参数。多个参数应该用空格分隔。

仅当您必须覆盖 JVM 使用的默认设置时才使用此参数。例如，如果计划导出大量的流，则可能需要增加默认堆大小。

Logging level (日志记录级别)

参数名称: `LOG_LEVEL`

组件的日志级别。从以下日志级别中进行选择，此处按级别顺序列出：

- `TRACE`
- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

默认值：`INFO`

分段上传的最小大小

参数名称: `STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

向 Amazon S3 进行分段上传的分段最小大小（以字节为单位）。流管理器使用此设置和输入文件的大小来确定如何对多部分 PUT 请求中的数据进行批处理。默认最小值为 `5242880` 字节 (5 MB)。

> **Note**
>
> 流管理器使用流的 `sizeThresholdForMultipartUploadBytes` 属性来确定是以单段上传还是分段上传的形式导出到 Amazon S3。用户定义的 Greengrass 组件在创建导出到 Amazon S3 的直播时会设置此阈值。默认阈值为 5 MB。

另请参阅

- 管理 Greengrass 核心设备上的数据流
- StreamManagerClient 用于处理直播
- 导出支持的 AWS Cloud 目标的配置

执行机器学习推理

借AWS IoT Greengrass助，您可以使用经过云训练的模型在边缘设备上对本地生成的数据执行机器学习 (ML) 推理。您可以从运行本地推理的低延迟和成本节省中受益，且仍然可以利用云计算在训练模型和复杂处理方面的强大功能。

AWS IoT Greengrass使执行推理所需的步骤更加高效。你可以在任何地方训练你的推理模型，并将它们作为机器学习组件部署到本地。例如，你可以在亚马逊中构建和训练深度学习模型， SageMaker 或者在亚马逊 Lookout for Vision 中构建和训练计算机视觉模型。然后，您可以将这些模型存储在 Amazon S3 存储桶中，这样您就可以将这些模型用作组件中的工件，以便在核心设备上执行推理。

主题

- AWS IoT Greengrass ML 推理的工作原理
- AWS IoT Greengrass版本 2 有什么不同？
- 要求
- 支持的模型源
- 支持的机器学习运行时
- AWS-提供的机器学习组件
- 在 Green SageMaker grass 核心设备上使用亚马逊 Edge Manager
- 在 Greengrass 核心设备上使用 Amazon Lookout for Vision fo
- 自定义您的机器学习组件
- 对机器学习推理进行故障排除

AWS IoT Greengrass ML 推理的工作原理

AWS提供了机器学习组件，您可以使用这些组件来创建一步部署以在设备上执行机器学习推理。您也可以将这些组件用作模板来创建自定义组件以满足您的特定要求。

AWS提供以下类别的机器学习组件：

- 模型组件-包含作为 Greengrass 工件的机器学习模型。
- 运行时组件-包含用于在 Greengrass 核心设备上安装机器学习框架及其依赖关系的脚本。
- 推理组件-包含推理代码并包括组件依赖项，用于安装机器学习框架和下载预训练的机器学习模型。

您为执行机器学习推理而创建的每个部署都至少包含一个组件，用于运行推理应用程序、安装机器学习框架和下载机器学习模型。要使用AWS提供的组件执行示例推理，您需要将推理组件部署到核心设备，该组件会自动包括相应的模型和运行时组件作为依赖项。要自定义部署，您可以将示例模型组件插入或替换为自定义模型组件，也可以将AWS提供的组件配方用作模板来创建自己的自定义推理、模型和运行时组件。

要使用自定义组件执行机器学习推理，请执行以下操作：

1. 创建模型组件。该组件包含要用于执行推理的机器学习模型。 AWS提供了预训练的 DLR 和 L TensorFlow ite 模型示例。要使用自定义模型，请创建自己的模型组件。
2. 创建运行时组件。该组件包含为模型安装机器学习运行时所需的脚本。 AWS为深度学习运行时 (DLR) 和TensorFlow 精简版提供了示例运行时组件。要将其他运行时与您的自定义模型和推理代码一起使用，请创建自己的运行时组件。
3. 创建推理组件。该组件包含您的推理代码，并包括您的模型和运行时组件作为依赖项。 AWS为使用 DLR 和 Lite 进行图像分类和目标检测提供示例推理组件。 TensorFlow 要执行其他类型的推理，或者要使用自定义模型和运行时，请创建自己的推理组件。
4. 部署推理组件。部署此组件时，AWS IoT Greengrass还会自动部署模型和运行时组件依赖关系。

要开始使用AWS提供的组件，请参阅the section called “执行样本图像分类推断”。

有关创建自定义机器学习组件的信息，请参阅自定义您的机器学习组件。

AWS IoT Greengrass版本 2 有什么不同？

AWS IoT Greengrass将机器学习的功能单元（例如模型、运行时和推理代码）整合到组件中，使您能够使用一步式流程安装机器学习运行时、下载经过训练的模型并在设备上执行推理。

通过使用AWS提供的机器学习组件，您可以灵活地开始使用示例推理代码和预训练模型执行机器学习推理。您可以插入自定义模型组件，将自己的自定义训练模型与提供的推理和运行时组件一起使用。AWS对于完全自定义的机器学习解决方案，您可以使用公共组件作为模板来创建自定义组件，并使用所需的任何运行时、模型或推理类型。

要求

要创建和使用机器学习组件，必须具备以下条件：

- 一款 Greengrass 核心设备。如果没有，请参阅教程：AWS IoT Greengrass V2 入门。
- 至少 500 MB 的本地存储空间才能使用AWS提供的示例机器学习组件。

支持的模型源

AWS IoT Greengrass支持使用存储在 Amazon S3 中的自定义训练机器学习模型。您还可以使用 Amazon SageMaker 边缘打包任务直接为您的 SageMaker Neo 编译模型创建模型组件。有关将 SageMaker 边缘管理器与配合使用的信息AWS IoT Greengrass，请参阅在 Green SageMaker grass 核心设备上使用亚马逊 Edge Manager。你也可以使用 Amazon Lookout for Vision 模型打包任务为 Lookout for Vision 模型创建模型组件。有关将 Lookout for Vision AWS IoT Greengrass 与配合使用的更多信息，在 Greengrass 核心设备上使用 Amazon Lookout for Vision fo请参阅。

包含您的模型的 S3 存储桶必须满足以下要求：

- 不得使用 SSE-C 对其进行加密。 对于使用服务器端加密的存储桶，AWS IoT Greengrass机器学习推理目前仅支持 SSE-S3 或 SSE-KMS 加密选项。有关服务器端加密选项的更多信息，请参阅 Amazon Simple Storage Service 用户指南中的使用服务器端加密保护数据。
- 它们的名字不得包含句点 (`.`)。有关更多信息，请参阅 Amazon Simple Storage Service 用户指南中的存储桶命名规则中有关通过 SSL 使用虚拟托管式存储桶的规则。
- 存储模型源的 S3 存储桶必须与您的机器学习组件相同AWS 账户。AWS 区域
- AWS IoT Greengrass必须拥有模型来源的`read`权限。AWS IoT Greengrass要允许访问 S3 存储桶，G reengrass 设备角色必须允许该操作。`s3:GetObject`有关设备角色的更多信息，请参阅授权核心设备与AWS服务。

支持的机器学习运行时

AWS IoT Greengrass允许您创建自定义组件，以便使用您选择的任何机器学习运行时对自定义训练模型执行机器学习推理。有关创建自定义机器学习组件的信息，请参阅自定义您的机器学习组件。

为了提高机器学习入门过程的效率，AWS IoT Greengrass提供了使用以下机器学习运行时的示例推理、模型和运行时组件：

- 深度学习运行时 (DLR) v1.6.0 和 v1.3.0
- TensorFlow 精简版 v2. 5.0

AWS-提供的机器学习组件

下表列出了AWS提供的用于机器学习的组件。

Note

AWS提供的几个组件依赖于 Greengrass 核的特定次要版本。由于这种依赖关系，当你将 Greengrass nucleus 更新到新的次要版本时，你需要更新这些组件。有关每个组件所依赖的特定原子核版本的信息，请参阅相应的组件主题。有关更新原子核的更多信息，请参见更新AWS IoT Greengrass核心软件 (OTA)。

组件	描述	组件类型	支持的操作系统	开源
Lookout for Vision Edge Agent	在 Greengrass 核心设备上部署 Amazon Lookout for Vision 运行时，因此您可以使用计算机视觉来查找工业产品中的缺陷。	通用	Linux	否
SageMaker 边缘管理器	在 Green SageMaker grass 核心设备上部署亚马逊 Edge Manager 代理。	通用	Linux、Windows	否
DLR 图像分类	推理组件，使用 DLR 图像分类模型存储和 DLR 运行时组件作为依赖项，在支持的设备上安装 DLR、下载样	通用	Linux、Windows	否

组件	描述	组件类型	支持的操作系统	开源
	本图像分类模型和执行图像分类推理。			
DLR 物体检测	推理组件，使用 DLR 对象检测模型存储和 DLR 运行时组件作为依赖项，用于在支持的设备上安装 DLR、下载示例对象检测模型和执行对象检测推理。	通用	Linux、Windows	否
DLR 图像分类模型存储	包含作为 Greengrass 伪影的样本 ResNet -50 图像分类模型的模型组件。	通用	Linux、Windows	否
DLR 物体检测模型存储	包含样本 YOLOv3 对象检测模型的模型组件，例如 Greengrass 工件。	通用	Linux、Windows	否

组件	描述	组件类型	支持的操作系统	开源
DLR 运行时	包含安装脚本的运行时组件，该脚本用于在 Greengrass 核心设备上安装 DLR 及其依赖关系。	通用	Linux、Windows	否
TensorFlow 精简版图像分类	推理组件，使用 TensorFlow Lite 图像分类模型存储和 TensorFlow Lite 运行时组件作为依赖项，用于在支持的设备上安装 TensorFlow Lite、下载样本图像分类模型和执行图像分类推理。	通用	Linux、Windows	否

组件	描述	组件类型	支持的操作系统	开源
TensorFlow 精简版物体检测	推理组件，使用 TensorFlow Lite 对象检测模型存储和 TensorFlow Lite 运行时组件作为依赖项，用于在支持的设备上安装 TensorFlow Lite、下载示例对象检测模型和执行对象检测推理。	通用	Linux、Windows	否
TensorFlow 精简版图像分类模型存储	包含作为 Greengrass 工件的示例 MobileNet v1 模型的模型组件。	通用	Linux、Windows	否
TensorFlow 精简版物体检测模型存储	模型组件，其中包含作为 Greengrass 工件的样本单枪检测 (SSD) MobileNet 模型。	通用	Linux、Windows	否

组件	描述	组件类型	支持的操作系统	开源
TensorFlow 精简版运行时	包含用于安装 TensorFlow Lite 的安装脚本及其对 Greengrass 核心设备的依赖关系的运行时组件。	通用	Linux、Windows	否

在 Green SageMaker grass 核心设备上使用亚马逊 Edge Manager

Important

SageMaker Edge Manager 将于 2024 年 4 月 26 日停产。有关继续将模型部署到边缘设备的更多信息，请参阅 Edg SageMaker e Manager 生命周期终止。

Amazon SageMaker Edge Manager 是一款在边缘设备上运行的软件代理。 SageMaker Edge Manager 为边缘设备提供模型管理，因此您可以直接在 Greengrass 核心设备上打包和使用 Amazon SageMaker Neo 编译的模型。通过使用 SageMaker Edge Manager，您还可以对核心设备的模型输入和输出数据进行采样，并将这些数据发送到AWS Cloud进行监控和分析。由于 E SageMaker dge Manager 使用 SageMaker Neo 来针对目标硬件优化模型，因此您无需直接在设备上安装 DLR 运行时。在 Greengrass 设备上 SageMaker ，Edge Manager 不会加载AWS IoT本地证书或直接调用AWS IoT凭据提供程序端点。相反， SageMaker Edge Manager 使用令牌交换服务从 TES 端点获取临时证书。

本节介绍 SageMaker 边缘管理器在 Greengrass 核心设备上的工作原理。

SageMaker 边缘管理器如何在 Greengrass 设备上运行

要将 SageMaker Edge Manager 代理部署到您的核心设备，请创建包含该`aws.greengrass.SageMakerEdgeManager`组件的部署。 AWS IoT Greengrass管

理设备上边缘管理器代理的安装和生命周期。当代理二进制文件有新版本可用时，请部署该`aws.greengrass.SageMakerEdgeManager`组件的更新版本以升级设备上安装的代理版本。

当您将 SageMaker Edge Manager 与一起使用时AWS IoT Greengrass，您的工作流程包括以下高级步骤：

1. 使用 SageMaker Neo 编译模型。
2. 使用 SageMaker 边缘打包作业打包您的 SageMaker Neo 编译模型。当你为模型运行边缘打包作业时，你可以选择用打包的模型作为工件来创建模型组件，然后部署到你的 Greengrass 核心设备上。
3. 创建自定义推理组件。您可以使用此推理组件与 Edge Manager 代理进行交互，以便在核心设备上执行推理。这些操作包括加载模型、调用预测请求以运行推理，以及在组件关闭时卸载模型。
4. 部署 SageMaker Edge Manager 组件、打包的模型组件和推理组件，以便在设备上的 SageMaker 推理引擎（边缘管理器代理）上运行您的模型。

有关创建与 Edge Manager 配 SageMaker 合使用的边缘打包任务和推理组件的更多信息，请参阅《亚马逊 SageMaker 开发者指南》AWS IoT Greengrass中的 “使用部署模型包和边缘管理器代理”。

本教程： SageMaker 边缘管理器入门教程使用提供的示例代码，向您展示了如何在现有 Green SageMaker grass 核心设备上设置和使用 Edge Manager 代理，您可以使用这些示例代码AWS来创建示例推理和建模组件。

在 Green SageMaker grass 核心设备上使用 Edge Manager 时，也可以使用捕获数据功能将示例数据上传到。AWS Cloud捕获数据是一项用于将推理输入、推理结果和其他推理数据上传到 S3 存储桶或本地目录以供将来分析的 SageMaker 功能。有关在 SageMaker Edge Manager 中使用捕获数据的更多信息，请参阅亚马逊 SageMaker 开发者指南中的管理模型。

要求

要在 Greengrass 核心设备上使用 SageMaker Edge Manager 代理，必须满足以下要求。

- 在亚马逊 Linux 2、基于 Debian 的 Linux 平台（x86_64 或 Armv8）或 Windows（x86_64）上运行的 Greengrass 核心设备。如果没有，请参阅教程：AWS IoT Greengrass V2 入门。
- 安装在核心设备上的 Pyth@@ on 3.6 或更高版本，包括`pip`适用于你的 Python 版本。
- Greengrass 设备角色配置如下：
  - 一种允许`credentials.iot.amazonaws.com`和代`sagemaker.amazonaws.com`入角色的信任关系，如以下 IAM 策略示例所示。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- I AmazonSageMakerEdgeDeviceFleetPolicyAM 托管策略。
- `s3:PutObject`操作，如以下 IAM 策略示例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- 与您的 Greengrass 核心设备AWS 区域相同AWS 账户且创建的 Amazon S3 存储桶。 SageMaker Edge Manager 需要一个 S3 存储桶来创建边缘设备队列，并存储在设备上运行推理的示例数据。有关创建 S3 存储桶的信息，请参阅 Amazon S3 入门。
- 使用与 Greengrass 核心设备相同的AWS IoT角色别名的 SageMaker 边缘设备队列。有关更多信息，请参阅 创建边缘设备队列。

- 您的 Greengrass 核心设备在您的 Edge 设备群中注册为边缘设备。 SageMaker 边缘设备名称必须与核心设备AWS IoT的事物名称相匹配。有关更多信息，请参阅 注册你的 Greengrass 核心设备。

开始使用 SageMaker 边缘管理器

您可以完成教程以开始使用 SageMaker 边缘管理器。本教程向您展示了如何在现有核心设备上开始使用 SageMaker Edge Manager 以及AWS提供的示例组件。这些示例组件使用 SageMaker Edge Manager 组件作为依赖项来部署 Edge Manager 代理，并使用使用 Neo 编译的预训练模型执行推理。SageMaker 有关更多信息，请参阅 教程： SageMaker 边缘管理器入门。

在 Greengrass 核心设备上使用 Amazon Lookout for Vision fo

> Note
>
> AWS IoT Greengrass目前在 Windows 核心设备上不支持此功能。

Amazon Lookout foAWS 服务 r Vision 可以用来发现工业产品中的视觉缺陷。它使用计算机视觉发现工业产品中缺失的组件、车辆或结构的损坏、生产线中的异常、印刷电路板中缺失的电容以及硅片或者质量至关重要的任何其他实物中的缺陷。有关更多信息，请参阅什么是 Amazon Lookout for Vision？在亚马逊 Lookout for Vision 开发者指南中。

你可以创建 Greengrass 应用程序，使用 Lookout 进行视觉推断来查找 Greengrass 核心设备上的视觉缺陷。将 Lookout for Vision 工作流程部署到 Greengrass 核心设备后，无需连接到中的 Lookout for Vision 服务即可执行计算机视觉AWS Cloud。要创建使用 Lookout for Vision 的 Greengrass 应用程序，您需要设置和部署以下 Greengrass 组件：

- Lookout for Vision 模型组件 — 包含视觉机器学习模型Lookout for Vision 器，如 Greengrass 工件。您可以使用 Lookout for Vision 控制台和 API 来生成打包预先训练的机器学习模型的模型组件。这些组件是您的私有 Greengrass 组件AWS 账户。有关更多信息，请参阅《亚马逊 Lookout for Vision 开发者指南》中的 “为视觉创建监视模式” 和 “打包视觉观察模型”。
- Lookout for Vision Edge Agent 组件 — 提供本地 Lookout for Vision 运行时服务器，该服务器使用计算机视觉使用您提供的机器学习模型检测异常。此组件是AWS提供的组件。有关更多信息，请参阅 Lookout for Vision Agent 组件。
- 注意 Vision 客户端应用程序组件 — 与 Lookout for Vision Edge Agent 组件交互以处理图像中是否存在异常。您可以开发自定义客户端应用程序组件，将图像和视频流发送到本地 Lookout for Vision

Edge Agent，并报告机器学习模型检测到的任何异常。有关更多信息，请参阅《亚马逊 Lookout f or Vision 开发者指南》中的编写客户端应用程序组件和 Lookout for Vision Edge Agent API 参考资料。

有关如何创建、配置和使用这些组件的更多信息，请参阅《亚马逊 Lookout for Vision 开发者指南》中的在边缘设备上使用 Lookout for Visi on 模型。

自定义您的机器学习组件

在中AWS IoT Greengrass，您可以配置示例机器学习组件，以使用推理、模型和运行时组件作为构建块，自定义在设备上执行机器学习推理的方式。 AWS IoT Greengrass还允许您灵活地使用示例组件作为模板并根据需要创建自己的自定义组件。您可以混合搭配这种模块化方法，通过以下方式自定义您的机器学习推理组件：

使用示例推理组件

- 在部署推理组件时修改其配置。
- 将自定义模型与示例推理组件一起使用，方法是将示例模型存储组件替换为自定义模型组件。您的自定义模型必须使用与示例模型相同的运行时进行训练。

使用自定义推理组件

- 通过添加公共模型组件和运行时组件作为自定义推理组件的依赖项，在示例模型和运行时中使用自定义推理代码。
- 创建和添加自定义模型组件或运行时组件作为自定义推理组件的依赖项。如果要使用自定义推理代码或AWS IoT Greengrass不提供示例组件的运行时，则必须使用自定义组件。

主题

- 修改公共推理组件的配置
- 使用带有示例推理组件的自定义模型
- 创建自定义机器学习组件
- 创建自定义推理组件

修改公共推理组件的配置

在AWS IoT Greengrass控制台中，组件页面显示该组件的默认配置。例如， TensorFlow 精简版图像分类组件的默认配置如下所示：

```
{
```

```
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/tflite/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/tflite/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "model": "TensorFlowLite-Mobilenet"
  }
}
```

部署公共推理组件时，您可以修改默认配置以自定义部署。有关每个公共推理组件的可用配置参数的信息，请参阅中的AWS-提供的机器学习组件组件主题。

本节介绍如何从AWS IoT Greengrass控制台部署修改后的组件。有关使用部署组件的信息AWS CLI，请参阅创建部署。

部署修改后的公共推理组件（控制台）

1. 登录 AWS IoT Greengrass 控制台。
2. 在导航菜单中，选择组件。
3. 在组件页面的公共组件选项卡上，选择要部署的组件。
4. 在组件页面上，选择部署。
5. 从 “添加到部署” 中，选择以下选项之一：
   a. 要将此组件合并到目标设备上的现有部署，请选择添加到现有部署，然后选择要修改的部署。
   b. 要在目标设备上创建新部署，请选择创建新部署。如果您的设备上已有部署，选择此步骤将替换现有部署。
6. 在指定目标页面中，执行以下操作：

   a. 在部署信息下，输入或修改部署的友好名称。
   b. 在部署目标下，选择部署目标，然后选择下一步。如果您正在修改现有部署，则无法更改部署目标。

7. 在 “选择组件” 页面上，在 “公共组件” 下，验证是否选择了经过修改的配置的推理组件，然后选择下一步。
8. 在配置组件页面上，执行以下操作：

   a. 选择推理组件，然后选择配置组件。
   b. 在配置更新下，输入要更新的配置值。例如，在 “要合并的配置” 框中输入以下配置更新，将推理间隔更改为 15 秒，并指示组件在文件夹custom.jpg中查找名为的/custom-ml-inference/images/图像。

   ```
   {
     "InferenceInterval": "15",
     "ImageName": "custom.jpg",
     "ImageDirectory": "/custom-ml-inference/images/"
   }
   ```

   要将组件的整个配置重置为其默认值，请在 “重置路径” 框""中指定一个空字符串。
   c. 选择确认，然后选择下一步。

9. 在 “配置高级设置” 页面上，保留默认配置设置，然后选择 “下一步”。
10. 在 “审阅” 页面上，选择 “部署”

使用带有示例推理组件的自定义模型

如果要将示例推理组件与自己的机器学习模型一起用于AWS IoT Greengrass提供示例运行时组件的运行时，则必须使用使用这些模型作为构件的组件来覆盖公共模型组件。简而言之，您需要完成以下步骤，将自定义模型与示例推理组件一起使用：

1. 创建使用 S3 存储桶中的自定义模型作为构件的模型组件。您的自定义模型必须使用与要替换的模型相同的运行时进行训练。
2. 修改推理组件中的ModelResourceKey配置参数以使用自定义模型。有关更新推理组件配置的信息，请参见 修改公共推理组件的配置

部署推理组件时，会AWS IoT Greengrass查找其组件依赖关系的最新版本。如果依赖的公共模型组件的更高自定义版本存在于同一个AWS 账户和AWS 区域中，则它会覆盖该组件。

创建自定义模型组件（控制台）

1. 将您的模型上传到 S3 存储桶。有关将模型上传到 S3 存储桶的信息，请参阅《亚马逊简单存储服务用户指南》中的 “使用 Amazon S3 存储桶”。

> **Note**
>
> 您必须将项目存储在与组件相同AWS 账户的 S3 存储桶中。AWS 区域AWS IoT Greengrass要允许访问这些工件，Greengrass 设备角色必须允许该操作。s3:GetObject有关设备角色的更多信息，请参阅授权核心设备与AWS服务。

2. 在AWS IoT Greengrass控制台导航菜单中，选择组件。
3. 检索公共模型商店组件的组件配方。
   a. 在组件页面的公共组件选项卡上，查找并选择要为其创建新版本的公共模型组件。例如，variant.DLR.ImageClassification.ModelStore。
   b. 在组件页面上，选择查看配方并复制显示的 JSON 配方。
4. 在 “组件” 页面的 “我的组件” 选项卡上，选择 “创建组件”。
5. 在创建组件页面的组件信息下，选择以 JSON 形式输入配方作为组件来源。
6. 在 “配方” 框中，粘贴您之前复制的组件配方。
7. 在配方中，更新以下值：
   - `ComponentVersion`：增加组件的次要版本。

     创建自定义组件以覆盖公共模型组件时，必须仅更新现有组件版本的次要版本。例如，如果公共组件版本为`2.1.0`，则可以使用版本创建自定义组件`2.1.1`。
   - `Manifests.Artifacts.Uri`：将每个 URI 值更新为您要使用的模型的 Amazon S3 URI。

> **Note**
>
> 请勿更改组件的名称。

8. 选择创建组件。

创建自定义模型组件 (AWS CLI)

1. 将您的模型上传到 S3 存储桶。有关将模型上传到 S3 存储桶的信息，请参阅《亚马逊简单存储服务用户指南》中的 “使用 Amazon S3 存储桶”。

> **Note**
>
> 您必须将项目存储在与组件相同AWS 账户的 S3 存储桶中。AWS 区域AWS IoT Greengrass要允许访问这些工件，Greengrass 设备角色必须允许该操作。s3:GetObject有关设备角色的更多信息，请参阅授权核心设备与AWS服务。

2. 运行以下命令以检索公共组件的组件配方。此命令将组件配方写入您在命令中提供的输出文件中。根据需要将检索到的 base64 编码字符串转换为 JSON 或 YAML。

   Linux, macOS, or Unix

```
aws greengrassv2 get-component \
    --arn <arn> \
    --recipe-output-format <recipe-format> \
    --query recipe \
    --output text | base64 --decode > <recipe-file>
```

   Windows Command Prompt (CMD)

```
aws greengrassv2 get-component ^
    --arn <arn> ^
    --recipe-output-format <recipe-format> ^
    --query recipe ^
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

   PowerShell

```
aws greengrassv2 get-component `
    --arn <arn> `
    --recipe-output-format <recipe-format> `
    --query recipe `
    --output text > <recipe-file>.base64
```

```
certutil -decode <recipe-file>.base64 <recipe-file>
```

3. 将配方文件的名称更新为*<component-name>*-*<component-version>*，其中组件版本是新组件的目标版本。例如，`variant.DLR.ImageClassification.ModelStore-2.1.1.yaml`。
4. 在配方中，更新以下值：

   - `ComponentVersion`：增加组件的次要版本。

     创建自定义组件以覆盖公共模型组件时，必须仅更新现有组件版本的次要版本。例如，如果公共组件版本为`2.1.0`，则可以使用版本创建自定义组件`2.1.1`。
   - `Manifests.Artifacts.Uri`：将每个 URI 值更新为您要使用的模型的 Amazon S3 URI。

   > **Note**
   >
   > 请勿更改组件的名称。

5. 运行以下命令，使用您检索和修改的配方创建新组件。

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/component/recipe
```

   > **Note**
   >
   > 此步骤在中的AWS IoT Greengrass服务中创建组件AWS Cloud。在将组件上传到云端之前，您可以使用 Greengrass CLI 在本地开发、测试和部署组件。有关更多信息，请参阅开发AWS IoT Greengrass组件。

有关创建组件的更多信息，请参阅开发AWS IoT Greengrass组件。

创建自定义机器学习组件

如果要使用自定义推理代码或AWS IoT Greengrass不提供示例组件的运行时，则必须创建自定义组件。您可以将自定义推理代码与AWS提供的示例机器学习模型和运行时结合使用，也可以使用自己的模型和运行时开发完全自定义的机器学习推理解决方案。如果您的模型使用的运行时AWS IoT Greengrass提供了示例运行时组件，则可以使用该运行时组件，并且您只需要为推理代码和要使用的模型创建自定义组件。

主题

- 检索公共组件的配方
- 检索示例组件工件
- 将组件项目上传到 S3 存储桶
- 创建自定义组件

检索公共组件的配方

您可以使用现有公共机器学习组件的配方作为模板来创建自定义组件。要查看最新版本公共组件的组件配方，请使用控制台或AWS CLI如下所示：

- 使用控制台

  1. 在 “组件” 页面的 “公共组件” 选项卡上，查找并选择公共组件。
  2. 在组件页面上，选择查看食谱。

- 使用 AWS CLI

  运行以下命令以检索公共变体组件的组件配方。此命令将组件配方写入您在命令中提供的 JSON 或 YAML 配方文件。

  Linux, macOS, or Unix

```
aws greengrassv2 get-component \
    --arn <arn> \
    --recipe-output-format <recipe-format> \
    --query recipe \
    --output text | base64 --decode > <recipe-file>
```

  Windows Command Prompt (CMD)

```
aws greengrassv2 get-component ^
    --arn <arn> ^
    --recipe-output-format <recipe-format> ^
    --query recipe ^
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

PowerShell

```
aws greengrassv2 get-component `
    --arn <arn> `
    --recipe-output-format <recipe-format> `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

按如下方式替换命令中的值：

- *<arn>*。 公共组件的亚马逊资源名称 (ARN)。
- *<recipe-format>*。 您要创建配方文件的格式。支持的值为 `JSON` 和 `YAML`。
- *<recipe-file>*。 格式中的配方名称*<component-name>*-*<component-version>*。

检索示例组件工件

您可以使用公共机器学习组件使用的构件作为模板来创建自定义组件工件，例如推理代码或运行时安装脚本。

要查看公共机器学习组件中包含的示例工件，请部署公共推理组件，然后在设备上的`/greengrass/v2/packages/artifacts-unarchived/component-name/component-version/`文件夹中查看这些工件。

将组件项目上传到 S3 存储桶

在创建自定义组件之前，必须将组件工件上传到 S3 存储桶并在组件配方中使用 S3 URI。例如，要在推理组件中使用自定义推理代码，请将代码上传到 S3 存储桶。然后，您可以将推理代码的 Amazon S3 URI 用作组件中的构件。

有关将内容上传到 S3 存储桶的信息，请参阅《亚马逊简单存储服务用户指南》中的 “使用 Amazon S3 存储桶”。

> **Note**
>
> 您必须将项目存储在与组件相同AWS 账户的 S3 存储桶中。AWS 区域AWS IoT Greengrass要允许访问这些工件，Greengrass 设备角色必须允许该操作。`s3:GetObject`有关设备角色的更多信息，请参阅授权核心设备与AWS服务。

创建自定义组件

您可以使用检索到的工件和配方来创建自定义机器学习组件。有关示例，请参阅创建自定义推理组件。

有关创建组件并将其部署到 Greengrass 设备的详细信息，请参阅和。开发AWS IoT Greengrass组件 将AWS IoT Greengrass组件部署到设备

创建自定义推理组件

本节介绍如何使用 DLR 图像分类组件作为模板创建自定义推理组件。

主题

- 将您的推理代码上传到 Amazon S3 存储桶
- 为推理组件创建配方
- 创建推理组件

将您的推理代码上传到 Amazon S3 存储桶

创建您的推理代码，然后将其上传到 S3 存储桶。有关将内容上传到 S3 存储桶的信息，请参阅《亚马逊简单存储服务用户指南》中的 “使用 Amazon S3 存储桶”。

您必须将项目存储在与组件相同AWS 账户的 S3 存储桶中。AWS 区域AWS IoT Greengrass要允许访问这些工件，Greengrass 设备角色必须允许该操作。`s3:GetObject`有关设备角色的更多信息，请参阅授权核心设备与AWS服务。

为推理组件创建配方

1. 运行以下命令以检索 DLR 图像分类组件的组件配方。此命令将组件配方写入您在命令中提供的 JSON 或 YAML 配方文件。

   Linux, macOS, or Unix

```
aws greengrassv2 get-component \
    --arn
 arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
 \
```

```
    --recipe-output-format JSON | YAML \
    --query recipe \
    --output text | base64 --decode > <recipe-file>
```

Windows Command Prompt (CMD)

```
aws greengrassv2 get-component ^
    --arn
 arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
 ^
    --recipe-output-format JSON | YAML ^
    --query recipe ^
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

PowerShell

```
aws greengrassv2 get-component `
    --arn
 arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
 `
    --recipe-output-format JSON | YAML `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

<recipe-file>用格式中的配方名称替换*<component-name>*-*<component-version>*。

2. 在配方中的`ComponentDependencies`对象中，根据要使用的模型和运行时组件，执行以下一项或多项操作：

   • 如果要使用 DLR 编译的模型，请保留 DLR 组件依赖关系。您也可以将其替换为对自定义运行时组件的依赖关系，如以下示例所示。

     运行时组件

     JSON

     ```
     {
         "<runtime-component>": {
     ```

```
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
<runtime-component>:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

- 保持 DLR 图像分类模型存储依赖关系以使用AWS提供的预训练的 ResNet -50 模型，或者对其进行修改以使用自定义模型组件。当你为公共模型组件添加依赖关系时，如果该组件的更高自定义版本存在于同一个AWS 账户和中AWS 区域，则推理组件将使用该自定义组件。指定模型组件依赖关系，如以下示例所示。

  公共模型组件

  JSON

```
{
    "variant.DLR.ImageClassification.ModelStore": {
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

  YAML

```
variant.DLR.ImageClassification.ModelStore:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

  自定义模型组件

  JSON

```
{
    "<custom-model-component>": {
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
```

```
}
```

YAML

```
<custom-model-component>:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

3. 在`ComponentConfiguration`对象中，添加此组件的默认配置。您可以稍后在部署组件时修改此配置。以下摘录显示了 DLR 图像分类组件的组件配置。

   例如，如果您使用自定义模型组件作为自定义推理组件的依赖项，请进行修改`ModelResourceKey`以提供您正在使用的模型的名称。

   JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.ImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/dlr/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/dlr/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification"
  }
}
```

YAML

```
accessControl:
    aws.greengrass.ipc.mqttproxy:
        'aws.greengrass.ImageClassification:mqttproxy:1':
            policyDescription: 'Allows access to publish via topic ml/dlr/image-
classification.'
            operations:
                - 'aws.greengrass#PublishToIoTCore'
            resources:
                - ml/dlr/image-classification
PublishResultsOnTopic: ml/dlr/image-classification
ImageName: cat.jpeg
InferenceInterval: 3600
ModelResourceKey:
    armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
    x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
    aarch64: "DLR-resnet50-aarch64-cpu-ImageClassification"
```

4. 在Manifests对象中，提供有关该组件部署到不同平台时使用的构件和配置的信息，以及成功运行该组件所需的任何其他信息。以下摘录显示了 DLR 图像分类组件中适用于 Linux 平台的Manifests对象配置。

JSON

```
{
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "arm"
      },
      "Name": "32-bit armv7l - Linux (raspberry pi)",
      "Artifacts": [
        {
          "URI": "s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Setenv": {
```

```
          "DLR_IC_MODEL_DIR":
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}",
          "DEFAULT_DLR_IC_IMAGE_DIR": "{artifacts:decompressedPath}/
image_classification/sample_images/"
        },
        "run": {
          "RequiresPrivilege": true,
          "script": ". {variant.DLR:configuration:/MLRootPath}/
greengrass_ml_dlr_venv/bin/activate\npython3 {artifacts:decompressedPath}/
image_classification/inference.py"
        }
      }
    }
  ]
}
```

YAML

```
Manifests:
  - Platform:
      os: linux
      architecture: arm
    Name: 32-bit armv7l - Linux (raspberry pi)
    Artifacts:
      - URI: s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip
        Unarchive: ZIP
    Lifecycle:
      Setenv:
        DLR_IC_MODEL_DIR:
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}"
        DEFAULT_DLR_IC_IMAGE_DIR: "{artifacts:decompressedPath}/
image_classification/sample_images/"
      run:
        RequiresPrivilege: true
        script: |-
          . {variant.DLR:configuration:/MLRootPath}/greengrass_ml_dlr_venv/bin/
activate
          python3 {artifacts:decompressedPath}/image_classification/inference.py
```

有关创建组件配方的详细信息，请参阅AWS IoT Greengrass组件配方参考。

创建推理组件

使用AWS IoT Greengrass控制台或AWS CLI使用您刚才定义的配方创建组件。创建组件后，您可以将其部署以在设备上执行推理。有关如何部署推理组件的示例，请参阅教程：使用 TensorFlow Lite 执行样本图像分类推断。

创建自定义推理组件（控制台）

1. 登录 AWS IoT Greengrass 控制台。
2. 在导航菜单中，选择组件。
3. 在 “组件” 页面的 “我的组件” 选项卡上，选择 “创建组件”。
4. 在 “创建组件” 页面的 “组件信息” 下，选择 “以 JSON 形式输入配方” 或 “以 YAML 形式输入配方” 作为组件来源。
5. 在食谱框中，输入您创建的自定义食谱。
6. 单击 “创建组件”。

创建自定义推理组件 () AWS CLI

运行以下命令，使用您创建的配方创建新的自定义组件。

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/recipe/file
```

Note

此步骤在中的AWS IoT Greengrass服务中创建组件AWS Cloud。在将组件上传到云端之前，您可以使用 Greengrass CLI 在本地开发、测试和部署组件。有关更多信息，请参阅 开发AWS IoT Greengrass组件。

对机器学习推理进行故障排除

使用本节中的故障排除信息和解决方案来帮助解决机器学习组件的问题。有关公共机器学习推理组件，请参阅以下组件日志中的错误消息：

Linux or Unix

- */greengrass/v2*/logs/aws.greengrass.DLRImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.DLRObjectDetection.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteObjectDetection.log

Windows

- *C:\greengrass\v2*\logs\aws.greengrass.DLRImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.DLRObjectDetection.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteObjectDetection.log

如果组件安装正确，则组件日志将包含用于推理的库的位置。

问题

- 无法获取库
- Cannot open shared object file
- Error: ModuleNotFoundError: No module named '<library>'
- 未检测到支持 CUDA 的设备
- 没有这样的文件或目录
- RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>
- picamera.exc.PiCameraError: Camera is not enabled
- 内存错误
- 磁盘空间错误
- 超时错误

无法获取库

在 Raspberry Pi 设备上部署期间，如果安装程序脚本无法下载所需的库，则会发生以下错误。

```
Err:2 http://raspbian.raspberrypi.org/raspbian buster/main armhf python3.7-dev armhf
 3.7.3-2+deb10u1
404 Not Found [IP: 93.93.128.193 80]
E: Failed to fetch http://raspbian.raspberrypi.org/raspbian/pool/main/p/python3.7/
libpython3.7-dev_3.7.3-2+deb10u1_armhf.deb 404 Not Found [IP: 93.93.128.193 80]
```

再次运行`sudo apt-get update`并部署您的组件。

Cannot open shared object file

在 Raspberry Pi 设备上部署`opencv-python`期间，当安装程序脚本无法下载所需的依赖项时，您可能会看到类似以下内容的错误。

```
ImportError: libopenjp2.so.7: cannot open shared object file: No such file or directory
```

运行以下命令手动安装的依赖项`opencv-python`：

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

Error: ModuleNotFoundError: No module named '<library>'

当 ML 运行时库`variant.DLR.log`或其依赖项未正确安装时，您可能会在 ML 运行时组件日志（或`variant.TensorFlowLite.log`）中看到此错误。在以下情况下可能会发生此错误：

- 如果您使用默认处于启用状态的`UseInstaller`选项，则此错误表示 ML 运行时组件无法安装运行时或其依赖项。执行以下操作：
  1. 配置 ML 运行时组件以禁用该`UseInstaller`选项。
  2. 安装 ML 运行时及其依赖关系，并使其可供运行 ML 组件的系统用户使用。有关更多信息，请参阅下列内容：
     - DLR 运行时选项 UseInstaller
     - TensorFlow精简版运行时 UseInstaller 选项
- 如果您不使用该`UseInstaller`选项，则此错误表示未为运行 ML 组件的系统用户安装 ML 运行时或其依赖项。执行以下操作：
  1. 检查是否为运行 ML 组件的系统用户安装了该库。将 *`ggc_user ########`*名称，并将 *`tflite_runtime ########`*的名称。

Linux or Unix

```
sudo -H -u ggc_user bash -c "python3 -c 'import tflite_runtime'"
```

Windows

```
runas /user:ggc_user "py -3 -c \"import tflite_runtime\""
```

2. 如果未安装该库，请为该用户安装该库。将 *ggc_user ########*名称，并将 *tflite_runtime* 替换为库的名称。

   Linux or Unix

   ```
   sudo -H -u ggc_user bash -c "python3 -m pip install --user tflite_runtime"
   ```

   Windows

   ```
   runas /user:ggc_user "py -3 -m pip install --user tflite_runtime"
   ```

   有关每个 ML 运行时的依赖关系的更多信息，请参阅以下内容：

   - DLR 运行时选项 UseInstaller
   - TensorFlow精简版运行时 UseInstaller 选项

3. 如果问题仍然存在，请为其他用户安装该库，以确认此设备是否可以安装该库。例如，该用户可以是您的用户、root 用户或管理员用户。如果您无法为任何用户成功安装该库，则您的设备可能不支持该库。请查阅库的文档，查看要求并解决安装问题。

未检测到支持 CUDA 的设备

使用 GPU 加速时，您可能会看到以下错误。运行以下命令为 Greengrass 用户启用 GPU 访问权限。

```
sudo usermod -a -G video ggc_user
```

没有这样的文件或目录

以下错误表明运行时组件无法正确设置虚拟环境：

- `MLRootPath/greengrass_ml_dlr_conda/bin/conda: No such file or directory`
- `MLRootPath/greengrass_ml_dlr_venv/bin/activate: No such file or directory`

- *MLRootPath*/greengrass_ml_tflite_conda/bin/conda: No such file or directory
- *MLRootPath*/greengrass_ml_tflite_venv/bin/activate: No such file or directory

检查日志，确保所有运行时依赖项均已正确安装。有关安装程序脚本安装的库的更多信息，请参阅以下主题：

- DLR 运行时
- TensorFlow 精简版运行时

默认情况下RootPath，*ML* 设置为*/greengrass/v2*/work/*component-name*/greengrass_ml。要更改此位置，请直接在部署中加入DLR 运行时或 TensorFlow 精简版运行时 runtime 组件，并在配置合并更新中为MLRootPath参数指定修改后的值。有关配置组件的更多信息，请参阅更新组件配置。

> **Note**
>
> 对于 DLR 组件 v1.3.x，您可以在推理组件的配置中设置MLRootPath参数，默认值为。$HOME/greengrass_ml

RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>

当你在运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 上运行机器学习推理时，你可能会看到以下错误。

```
RuntimeError: module compiled against API version 0xf but this version of numpy is 0xd
ImportError: numpy.core.multiarray failed to import
```

之所以出现此错误，是因为 Raspberry Pi OS Bullseye 包含的版本早 NumPy于 OpenCV 要求的版本。要修复此问题，请运行以下命令升级 NumPy 到最新版本。

```
pip3 install --upgrade numpy
```

picamera.exc.PiCameraError: Camera is not enabled

当你在运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 上运行机器学习推理时，你可能会看到以下错误。

```
picamera.exc.PiCameraError: Camera is not enabled. Try running 'sudo raspi-config' and
 ensure that the camera has been enabled.
```

之所以出现此错误，是因为 Raspberry Pi OS Bullseye 包含一个与 ML 组件不兼容的新相机堆栈。要修复此问题，请启用旧版相机堆栈。

启用旧版相机堆栈

1. 运行以下命令打开 Raspberry Pi 配置工具。

   ```
   sudo raspi-config
   ```
2. 选择接口选项。
3. 选择旧版相机以启用旧版相机堆栈。
4. 重启 Raspberry Pi。

内存错误

当设备没有足够的内存并且组件处理中断时，通常会发生以下错误。

- `stderr. Killed.`
- `exitCode=137`

我们建议至少有 500 MB 的内存来部署公共机器学习推理组件。

磁盘空间错误

该`no space left on device`错误通常发生在设备存储空间不足时。再次部署组件之前，请确保设备上有足够的可用磁盘空间。我们建议至少有 500 MB 的可用磁盘空间来部署公共机器学习推理组件。

超时错误

公共机器学习组件会下载大于 200 MB 的大型机器学习模型文件。如果在部署期间下载超时，请检查您的互联网连接速度并重试部署。

通过以下方式管理 Greengrass 核心设备 AWS Systems Manager

> **Note**
>
> AWS IoT Greengrass目前不支持在 Windows 核心设备上使用此功能。

Systems Manager 是一项可用于查看和控制基础设施的AWS服务AWS，包括 Amazon EC2 实例、本地服务器和虚拟机 (VM) 以及边缘设备。Systems Manager 使您能够查看操作数据、自动执行操作任务以及维护安全性和合规性。当你向 Systems Manager 注册一台计算机时，它被称为托管节点。有关更多信息，请参阅AWS Systems Manager《用户指南》中的什么是AWS Systems Manager？。

AWS Systems Manager代理（Systems Manager Agent）是可以安装在设备上的软件，用于让 Systems Manager 更新、管理和配置这些设备。要在 Greengrass 核心设备上安装 Systems Manager 代理，请部署 Systems Manager 代理组件。首次部署 Systems Manager 代理时，它会将核心设备注册为 Systems Manager 托管节点。Systems Manager 代理在设备上运行，以支持与中的 Systems Manager 服务进行通信AWS Cloud。有关如何安装和配置 Systems Manager 代理组件的更多信息，请参阅安装 AWS Systems Manager 代理。

Systems Manager 的工具和功能称为功能。Greengrass 核心设备支持 Systems Manager 的所有功能。有关这些功能以及如何使用 Systems Manager 管理核心设备的更多信息，请参阅《AWS Systems Manager用户指南》中的 Systems Manager 功能。

AWS Systems Manager为 Systems Manager 托管节点提供了标准实例层和高级实例层。如果您是首次使用 Systems Manager，则从标准实例层开始。在标准实例层中，每个托管节点最多可以注册 1,000 个AWS 区域。AWS 账户如果您需要在单个账户和区域中注册超过 1,000 个托管节点，或者需要使用会话管理器功能，请使用高级实例级别。有关更多信息，请参阅《AWS Systems Manager用户指南》中的配置实例层。

主题

- 安装 AWS Systems Manager 代理
- 卸载 AWS Systems Manager 代理

安装 AWS Systems Manager 代理

AWS Systems Manager代理（Systems Manager Agent）是您安装的亚马逊软件，用于让 Systems Manager 更新、管理和配置 Greengrass 核心设备、亚马逊 EC2 实例和其他资源。代理在中处理和运行来自 Systems Manager 服务的请求AWS Cloud。然后，代理将状态和运行时信息发送回 Systems Manager 服务。有关更多信息，请参阅《AWS Systems Manager用户指南》中的 “关于 Systems Manager 代理”。

AWS将 Systems Manager 代理作为 Greengrass 组件提供，您可以将其部署到 Greengrass 核心设备上，以便使用 Systems Manager 对其进行管理。Systems Manager Agent 组件安装 Systems Manager 代理软件，并将核心设备注册为系统管理器中的托管节点。按照本页上的步骤完成先决条件，将 Systems Manager 代理组件部署到核心设备或一组核心设备。

主题

- 步骤 1：完成常规 Systems Manager 设置步骤
- 步骤 2：为 Systems Manager 创建 IAM 服务角色
- 步骤 3：为令牌交换角色添加权限
- 步骤 4：部署 Systems Manager 代理组件
- 步骤 5：使用 Systems Manager 验证核心设备的注册情况

步骤 1：完成常规 Systems Manager 设置步骤

如果您尚未执行此操作，请完成的常规设置步骤AWS Systems Manager。有关更多信息，请参阅《AWS Systems Manager用户指南》中的 “完成一般 Systems Manager 设置步骤”。

步骤 2：为 Systems Manager 创建 IAM 服务角色

Systems Manager 代理使用 AWS Identity and Access Management (IAM) 服务角色与之通信AWS Systems Manager。Systems Manager 扮演此角色是为了在每台核心设备上启用 Systems Manager 功能。在部署核心设备时，Systems Manager Agent 组件还使用此角色将核心设备注册为 Systems Manager 托管节点。如果您尚未这样做，请创建 Systems Manager 服务角色以供系统管理器代理组件使用。有关更多信息，请参阅AWS Systems Manager用户指南中的为边缘设备创建 IAM 服务角色。

步骤 3：为令牌交换角色添加权限

Greengrass 核心设备使用 IAM 服务角色（称为令牌交换角色）与服务进行交互。AWS每台核心设备都有您在安装 Core 软件时创建的AWS IoT Greengrass令牌交换角色。许多 Greengrass 组件，例如

Systems Manager Agent，都需要对该角色的额外权限。Systems Manager 代理组件需要以下权限，其中包括使用您在中创建的角色的权限步骤 2：为 Systems Manager 创建 IAM 服务角色。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

如果您尚未这样做，请将这些权限添加到核心设备的令牌交换角色中，以允许 Systems Manager 代理运行。您可以向令牌交换角色添加新策略以授予此权限。

向令牌交换角色添加权限（控制台）

1. 在 IAM 控制台导航菜单中，选择角色。
2. 选择您在安装 C AWS IoT Greengrass ore 软件时设置为令牌交换角色的 IAM 角色。如果您在安装 C AWS IoT Greengrass ore 软件时没有为令牌交换角色指定名称，则它会创建一个名为的角色`GreengrassV2TokenExchangeRole`。
3. 在 “权限” 下，选择 “添加权限”，然后选择 “附加策略”。
4. 选择创建策略。创建策略页面将在新的浏览器选项卡中打开。
5. 在创建策略页面上，执行以下操作：

   a. 选择 JSON 以打开 JSON 编辑器。

b. 将下面的 策略粘贴到 JSON 编辑器中。将 *SSM ServiceRole* 替换为您在中创建的服务角色的[步骤 2：为 Systems Manager 创建 IAM 服务角色](#)名称。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

c. 选择下一步：标签。

d. 选择下一步：审核。

e. 输入策略的名称，例如 **GreengrassSSMAgentComponentPolicy**。

f. 选择创建策略。

g. 切换到上一个浏览器选项卡，您可以在其中打开令牌交换角色。

6. 在添加权限页面上，选择刷新按钮，然后选择您在上一步中创建的 Greengrass Systems Manager 代理策略。

7. 选择附加策略。

   使用此令牌交换角色的核心设备现在有权与 Systems Manager 服务进行交互。

为令牌交换角色添加权限 (AWS CLI)

添加授予使用 Systems Manager 权限的策略

1. 创建一个名为的文件，`ssm-agent-component-policy.json`并将以下 JSON 复制到该文件中。将 *SSM ServiceRole* 替换为您在中创建的服务角色的[步骤 2：为 Systems Manager 创建 IAM 服务角色](#)名称。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

2. 运行以下命令，根据中的策略文档创建策略`ssm-agent-component-policy.json`。

   Linux or Unix

```
aws iam create-policy \
  --policy-name GreengrassSSMAgentComponentPolicy \
  --policy-document file://ssm-agent-component-policy.json
```

   Windows Command Prompt (CMD)

```
aws iam create-policy ^
```

```
  --policy-name GreengrassSSMAgentComponentPolicy ^
  --policy-document file://ssm-agent-component-policy.json
```

PowerShell

```
aws iam create-policy `
  --policy-name GreengrassSSMAgentComponentPolicy `
  --policy-document file://ssm-agent-component-policy.json
```

从输出中的策略元数据中复制策略 Amazon 资源名称 (ARN)。在下一步中，您将使用此 ARN 将此策略附加到核心设备角色。

3. 运行以下命令将策略附加到令牌交换角色。

   • 将 *GreenGrassV2 TokenExchangeRole* 替换为您在安装 Core 软件时指定的令牌交换角色的名称。AWS IoT Greengrass如果您在安装 C AWS IoT Greengrass ore 软件时没有为令牌交换角色指定名称，则它会创建一个名为的角色`GreengrassV2TokenExchangeRole`。

   • 将策略 ARN 替换为上一步中的 ARN。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

如果命令没有输出，则表示成功了。使用此令牌交换角色的核心设备现在有权与 Systems Manager 服务进行交互。

步骤 4：部署 Systems Manager 代理组件

完成以下步骤以部署和配置 Systems Manager 代理组件。您可以将组件部署到单核设备或一组核心设备。

部署 Systems Manager 代理组件（控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择组件。
2. 在 “组件” 页上，选择 “公共组件” 选项卡，然后选择aws.greengrass.SystemsManagerAgent。
3. 在 aws.greengrass.SystemsManagerAgent 页面上，选择部署。
4. 从 “添加到部署” 中，选择要修改的现有部署，或者选择创建新部署，然后选择 “下一步”。
5. 如果您选择创建新部署，请为部署选择目标核心设备或事物组。在 “指定目标” 页面的 “部署目标” 下，选择核心设备或事物组，然后选择下一步。
6. 在 “选择组件” 页面上，确认已选择该aws.greengrass.SystemsManagerAgent组件，然后选择 “下一步”。
7. 在 “配置组件” 页面上 aws.greengrass.SystemsManagerAgent，选择，然后执行以下操作：

    a. 选择配置组件。

    b. 在 “配置” aws.greengrass.SystemsManagerAgent 模式的 “配置更新” 下，在 “要合并的配置” 中，输入以下配置更新。将 *SSM ServiceRole* 替换为您在中创建的服务角色的步骤 2：为 Systems Manager 创建 IAM 服务角色名称。

    ```
    {
      "SSMRegistrationRole": "SSMServiceRole",
      "SSMOverrideExistingRegistration": false
    }
    ```

    > **Note**
    >
    > 如果核心设备已经运行通过混合激活注册的 Systems Manager 代理，SSMOverrideExistingRegistration请更改为true。此参数指定当

Systems Manager 代理已经通过混合激活在设备上运行时，Systems Manager Agent 组件是否注册核心设备。

您还可以指定要添加到 Systems Manager 代理组件为核心设备创建的 Systems Manager 托管节点的标签 (SSMResourceTags)。有关更多信息，请参阅 Systems Manager 代理组件配置。

c. 选择 “确认” 关闭模式，然后选择 “下一步”。

8. 在配置高级设置页面上，保留默认配置设置，然后选择下一步。
9. 在 Review（检查）页上，选择 Deploy（部署）。

   部署最多可能需要一分钟才能完成。

部署 Systems Manager 代理组件 (AWS CLI)

要部署 Systems Manager 代理组件，请创建包含`aws.greengrass.SystemsManagerAgent`在`components`对象中的部署文档，并指定该组件的配置更新。按照中的创建部署说明创建新部署或修改现有部署。

以下示例部分部署文档指定使用名为的服务角色`SSMServiceRole`。将 *SSM ServiceRole* 替换为您在中创建的服务角色的步骤 2：为 Systems Manager 创建 IAM 服务角色名称。

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.SystemsManagerAgent": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"SSMRegistrationRole\":\"SSMServiceRole\",
\"SSMOverrideExistingRegistration\":false}"
      }
    }
  }
}
```

Note

如果核心设备已经运行通过混合激活注册的 Systems Manager 代理，`SSMOverrideExistingRegistration`请更改为`true`。此参数指定当 Systems

Manager 代理已经通过混合激活在设备上运行时，Systems Manager Agent 组件是否注册核心设备。
您还可以指定要添加到 Systems Manager 代理组件为核心设备创建的 Systems Manager 托管节点的标签 (`SSMResourceTags`)。有关更多信息，请参阅 Systems Manager 代理组件配置。

完成部署可能需要数分钟。您可以使用该AWS IoT Greengrass服务来检查部署状态，也可以检查AWS IoT Greengrass核心软件日志和 Systems Manager 代理组件日志，以验证 Systems Manager 代理是否成功运行。有关更多信息，请参阅下列内容：

- 检查部署状态
- 监控AWS IoT Greengrass日志
- 在《AWS Systems Manager用户指南》中@@ 查看 Systems Manager 代理日志

如果部署失败或 Systems Manager 代理无法运行，则可以对每台核心设备上的部署进行故障排除。有关更多信息，请参阅下列内容：

- 故障排除 AWS IoT Greengrass V2
- AWS Systems Manager用户指南中的 Systems Manager 代理疑难解答

步骤 5：使用 Systems Manager 验证核心设备的注册情况

当 Systems Manager 代理组件运行时，它会在 Systems Manager 中将核心设备注册为托管节点。您可以使用AWS IoT Greengrass控制台、Systems Manager 控制台和 Systems Manager API 来验证核心设备是否已注册为托管节点。在AWS控制台和 API 的某些部分中，托管节点也被称为实例。

验证核心设备注册（AWS IoT Greengrass控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择核心设备。
2. 选择要验证的核心设备。
3. 在核心设备的详细信息页面上，找到AWS Systems Manager实例属性。如果存在此属性并显示指向 Systems Manager 控制台的链接，则核心设备将被注册为托管节点。

   您还可以找到 AWS Systems Managerping 状态属性来检查核心设备上 Systems Manager 代理的状态。当状态为 “在线” 时，您可以使用 Systems Manager 管理核心设备。

验证核心设备注册（Systems Manager 控制台）

1. 在 Systems Manager 控制台导航菜单中，选择 Fleet Manager。
2. 在 “托管节点” 下，执行以下操作：
   a. 在 “来源” 类型所在的位置添加过滤器AWS::IoT::Thing。
   b. 添加一个过滤器，其中 S ource ID 是要验证的核心设备的名称。
3. 在 “托管节点” 表中找到核心设备。如果核心设备在表中，则将其注册为托管节点。

   您还可以找到 S ystems Manager 代理 ping 状态属性来检查核心设备上 Systems Manager 代理的状态。当状态为 “在线” 时，您可以使用 Systems Manager 管理核心设备。

验证核心设备注册 (AWS CLI)

- 使用DescribeInstanceInformation操作获取与您指定的筛选条件相匹配的托管节点列表。运行以下命令以验证核心设备是否已注册为托管节点。*MyGreengrassCore*替换为要验证的核心设备的名称。

  ```
  aws ssm describe-instance-information --filter
   Key=SourceIds,Values=MyGreengrassCore Key=SourceTypes,Values=AWS::IoT::Thing
  ```

  响应包含与过滤器匹配的托管节点列表。如果列表包含托管节点，则该核心设备将被注册为托管节点。您还可以在响应中找到有关核心设备托管节点的其他信息。如果`PingStatus`属性为`Online`，则可以使用 Systems Manager 管理核心设备。

验证核心设备已在 Systems Manager 中注册为托管节点后，您可以使用 Systems Manager 控制台和 API 来管理该核心设备。有关可用于管理 Greengrass 核心设备的 Systems Manager 功能的更多信息，请参阅《用户指南》中的 Systems Manager 功能。AWS Systems Manager

卸载 AWS Systems Manager 代理

如果您不想再管理 Greengrass 核心设备，则可以从Systems Manager 管理器中取消注册核心设备，然后从设备上卸载代AWS Systems Manager理（系统管理器代理）。AWS Systems Manager

您可随时再次重新注册核心设备。为此，请再次部署 Systems Manager 代理组件，该组件会在安装核心设备时向Systems Manager 注册。Systems Manager 将已取消注册的核心设备的命令历史记录存储 30 天。

主题

- 步骤 1：从Systems Manager 器注销核心设备
- 步骤 2：卸载Systems Manager 代理组件
- 第 3 步：卸载Systems Manager 代理软件

步骤 1：从Systems Manager 器注销核心设备

您可以使用 Systems Manager 控制台或者 API 取消注册核心设备。有关更多信息，请参阅AWS Systems Manager用户指南中的取消注册托管节点。

步骤 2：卸载Systems Manager 代理组件

取消注册核心设备后，从设备上卸载 S ystems Manager 代理组件。要从 Greengrass 核心设备中删除组件，请修改安装该组件的部署，然后从部署中删除该组件。当AWS IoT Greengrass核心设备的部署均未指定组件时，核心软件会卸载该组件。有关更多信息，请参阅将AWS IoT Greengrass组件部署到设备：

卸载Systems Manager 代理组件（控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择核心设备。
2. 选择要卸载 Systems Manager 代理组件的核心设备。
3. 在核心设备详细信息页面上，选择 Deploation（部署）
4. 选择将 Systems Manager 代理组件部署到核心设备的部署。
5. 在部署详细信息页面上，选择修订。
6. 在修订部署模式中，选择修改部署。
7. 在步骤 1：指定目标中，选择下一步。
8. 在步骤 2：选择组件中，清除对aws.greengrass.SystemsManagerAgent组件的选择，然后选择下一步。
9. 在步骤 3：配置组件中，选择下一步。
10. 在步骤 4：配置高级设置中，选择下一步。
11. 在步骤 5：查看中，选择部署。

卸载Systems Manager 代理组件 (CLI)

要卸载 Systems Manager 代理组件，请修改部署该组件的部署，然后将其从部署中删除。有关更多信息，请参阅修改部署：

部署可能需要几分钟的时间才能完成。您可以使用该AWS IoT Greengrass服务检查部署状态。有关更多信息，请参阅检查部署状态：

第 3 步：卸载Systems Manager 代理软件

删除Systems Manager 代理组件后，Systems Manager 器代理软件将继续在核心设备上运行。要删除 Systems Manager 代理软件，可以在核心设备上运行命令。有关更多信息，请参阅《AWS Systems Manager用户指南》中的 “从 Linux 实例中卸载 Systems Manager 代理”。

AWS IoT Greengrass 中的安全性

AWS 十分重视云安全性。作为 AWS 客户，您将从专为满足大多数安全敏感型企业的要求而打造的数据中心和网络架构中受益。

安全性是 AWS 和您的共同责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云的安全性 – AWS 负责保护在 AWS Cloud 中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。第三方审核员定期测试和验证我们的安全性的有效性，作为 AWS Compliance Programs 的一部分。要了解适用于 AWS IoT Greengrass 的合规性计划，请参阅合规性计划范围内的 AWS 服务。
- 云中的安全性——您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

使用 AWS IoT Greengrass 时，您还需要负责保护您的设备、本地网络连接和私钥。

此文档将帮助您了解如何在使用 AWS IoT Greengrass 时应用责任共担模型。以下主题说明如何配置 AWS IoT Greengrass 以实现您的安全性和合规性目标。您还会了解如何使用其他 AWS 服务以帮助您监控和保护 AWS IoT Greengrass 资源。

主题

- AWS IoT Greengrass 中的数据保护
- AWS IoT Greengrass 的设备身份验证和授权
- 适用于 AWS IoT Greengrass 的身份和访问管理
- 允许设备流量通过代理或防火墙
- AWS IoT Greengrass 的合规性验证
- AWS IoT Greengrass 中的故障恢复能力
- AWS IoT Greengrass 中的基础设施安全性
- AWS IoT Greengrass 中的配置和漏洞分析
- 中的代码完整性AWS IoT Greengrass V2
- AWS IoT Greengrass 和接口 VPC 端点 (AWS PrivateLink)
- AWS IoT Greengrass 的安全最佳实践

AWS IoT Greengrass 中的数据保护

AWS 责任共担模式适用于 AWS IoT Greengrass 中的数据保护。如该模式中所述，AWS 负责保护运行所有 AWS Cloud 的全球基础设施。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使用的 AWS 服务 的安全配置和管理任务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅 AWS 安全性博客 上的博客文章 AWS Shared Responsibility Model and GDPR。

出于数据保护目的，我们建议您保护 AWS 账户凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management（IAM）设置单个用户。这样，每个用户只获得履行其工作职责所需的权限。我们还建议您通过以下方式保护数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用 SSL/TLS 与 AWS 资源进行通信。我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用 AWS CloudTrail 设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 服务 中的所有默认安全控制。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的敏感数据。
- 如果您在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，请参阅《美国联邦信息处理标准（FIPS）第 140-2 版》。

我们强烈建议您切勿将机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括使用控制台、API、AWS CLI 或 AWS SDK 处理 AWS IoT Greengrass 或其他 AWS 服务时。在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，我们强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

有关保护 AWS IoT Greengrass 敏感信息的更多信息，请参阅the section called “不要记录敏感信息”。

有关数据保护的更多信息，请参阅AWS安全性博客 上的AWS责任共担模式和 GDPR 博客文章。

主题

- 数据加密
- 硬件安全性集成

数据加密

AWS IoT Greengrass 使用加密来保护传输（通过互联网或本地网络）中和静态（存储在 AWS Cloud 中）数据的安全。

AWS IoT Greengrass 环境中的设备通常会收集发送到 AWS 服务以供进一步处理的数据。有关其它 AWS 服务上的数据加密的更多信息，请参阅该服务的安全文档。

主题

- 传输中加密
- 静态加密
- Greengrass 核心设备的密钥管理

传输中加密

AWS IoT Greengrass有两种传输数据的通信模式：

- the section called “通过 Internet 传输的数据”. Greengrass 核心和AWS IoT Greengrass通过互联网进行的是加密的。
- the section called “核心设备上的数据”. Greengrass 核心设备上的组件之间的通信未加密。

通过 Internet 传输的数据

AWS IoT Greengrass 使用传输层安全性 (TLS) 来加密通过 Internet 进行的所有通信。发送到的所有数据AWS Cloud通过 TLS 连接发送使用 MQTT 或 HTTPS 协议，因此默认情况下是安全的。AWS IoT Greengrass使用AWS IoT传输安全模型。有关更多信息，请参阅 。传输安全中的AWS IoT Core开发人员指南.

核心设备上的数据

AWS IoT Greengrass 不会对 Greengrass 核心设备上本地交换的数据进行加密，因为数据不会离开设备。这包括用户定义的组件之间的通信，AWS IoT设备 SDK 和公共组件，例如流媒体管理器。

静态加密

AWS IoT Greengrass 存储您的数据：

- the section called “中的静态数据AWS Cloud”. 此数据已加密。
- the section called “Greengrass 核心上的静态数据”. 此数据未加密（密钥的本地副本除外）。

中的静态数据AWS Cloud

AWS IoT Greengrass加密存储在中的客户数据AWS Cloud. 此数据使用由 AWS IoT Greengrass 管理的 AWS KMS 密钥进行保护。

Greengrass 核心上的静态数据

AWS IoT Greengrass 依赖于 Unix 文件权限和全磁盘加密（如果启用）来保护核心上的静态数据。您负责确保文件系统和设备的安全性。

但是，AWS IoT Greengrass 会对从 AWS Secrets Manager 检索到的密钥的本地副本进行加密。有关更多信息，请参阅 。密钥管理器组件。

Greengrass 核心设备的密钥管理

客户有责任保证在 Greengrass 核心设备上安全存储加密（公共和私有）密钥。AWS IoT Greengrass 在以下情况下使用公钥和私钥：

- IoT 客户端密钥与 IoT 证书一起使用，以便在 Greengrass 核心连接 AWS IoT Core 时对传输层安全性 (TLS) 握手进行身份验证。有关更多信息，请参阅 the section called “设备身份验证和授权”。

Note

密钥和证书也称为核心私钥和核心设备证书。

Greengrass 核心设备支持使用文件系统权限或使用文件系统权限或硬件安全模块. 如果您使用基于文件系统的私钥，您需要负责在核心设备上安全存储这些私钥。

硬件安全性集成

Note

此功能适用于 Greengrass nucleus 组件的 2.5.3 及更高版本。 AWS IoT Greengrass目前不支持在 Windows 核心设备上使用此功能。

您可以通过 PKCS #11 接口将AWS IoT Greengrass核心软件配置为使用硬件安全模块 (HSM)。此功能使您能够安全地存储设备的私钥和证书，这样它们就不会在软件中暴露或复制。您可以将私钥和证书存储在 HSM 或可信平台模块 (TPM) 等硬件模块上。

AWS IoT GreengrassCore 软件使用私钥和 X.509 证书来验证与和服务的连接。AWS IoT AWS IoT Greengrass密钥管理器组件使用此私钥来安全地加密和解密您部署到 Greengrass 核心设备的机密。将核心设备配置为使用 HSM 时，这些组件使用您存储在 HSM 中的私钥和证书。

M oquette MQTT 代理组件还存储其本地 MQTT 服务器证书的私钥。此组件将设备文件系统上的私钥存储在组件的工作文件夹中。目前，AWS IoT Greengrass不支持将此私钥或证书存储在 HSM 中。

Tip

在AWS合作伙伴设备目录中搜索支持此功能的设备。

主题

- 要求
- 硬件安全最佳实践
- 安装具有硬件安全性的 AWS IoT Greengrass Core 软件
- 在现有核心设备上配置硬件安全
- 使用不支持 PKCS #11 的硬件
- 另请参阅

要求

要在 Greengrass 核心设备上使用 HSM，您必须满足以下要求：

- Greengrass nucleus v2.5.3 或更高版本安装在核心设备上。在AWS IoT Greengrass核心设备上安装酷睿软件时，可以选择兼容版本。
- 安装在核心设备上的 PKCS #11 提供程序组件。在AWS IoT Greengrass核心设备上安装 Core 软件时，可以下载并安装此组件。
- 一种硬件安全模块，支持 PKCS #1 v1.5 签名方案和密钥大小为 RSA-2048（或更大）或 ECC 密钥的 RSA 密钥。

Note

要使用带有 ECC 密钥的硬件安全模块，必须使用 Greengrass nucleus v2.5.6 或更高版本。要使用硬件安全模块和密钥管理器，必须使用带有 RSA 密钥的硬件安全模块。

- 一个 PKCS #11 提供程序库，AWS IoT Greengrass核心软件可以在运行时加载该库（使用 libdl）来调用 PKCS #11 函数。PKCS #11 提供程序库必须实现以下 PKCS #11 API 操作：
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`
  - `C_OpenSession`
  - `C_GetSessionInfo`
  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`
  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`
  - `C_FindObjectsFinal`
  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`
  - `C_Sign`
  - `C_SignUpdate`
  - `C_SignFinal`
  - `C_GetMechanismList`
  - `C_GetMechanismInfo`
  - `C_GetInfo`
  - `C_GetFunctionList`
- 硬件模块必须可按槽标签解析，如 PKCS#11 规范所定义。
- 如果 HSM 支持对象 ID，则必须将私钥和证书存储在 HSM 的同一个插槽中，并且它们必须使用相同的对象标签和对象 ID。

- 证书和私钥必须可通过对象标签进行解析。
- 私钥必须具有以下权限：
  - `sign`
  - `decrypt`
- （可选）要使用密钥管理器组件，必须使用 2.1.0 或更高版本，并且私钥必须具有以下权限：
  - `unwrap`
  - `wrap`

硬件安全最佳实践

在 Greengrass 核心设备上配置硬件安全时，请考虑以下最佳实践。

- 使用内部硬件随机数字生成器直接在 HSM 上生成私有密钥。这种方法比导入您在其他地方生成的私钥更安全，因为私钥保留在 HSM 中。
- 将私钥配置为不可变并禁止导出。
- 使用 HSM 硬件供应商推荐的配置工具，使用受硬件保护的私钥生成证书签名请求 (CSR)，然后使用 AWS IoT控制台或 API 生成客户端证书。

> **Note**
>
> 在 HSM 上生成私钥时，轮换密钥的安全最佳实践不适用。

安装具有硬件安全性的 AWS IoT Greengrass Core 软件

安装 AWS IoT Greengrass Core 软件时，可以将其配置为使用在 HSM 中生成的私钥。这种方法遵循安全最佳实践，在 HSM 中生成私钥，因此私钥保留在 HSM 中。

要安装具有硬件安全性的AWS IoT Greengrass核心软件，请执行以下操作：

1. 在 HSM 中生成私钥。
2. 使用私钥创建证书签名请求 (CSR)。
3. 从 CSR 创建证书。您可以创建由其他根证书颁发机构 (CA) 签名AWS IoT或由其他根证书颁发机构 (CA) 签名的证书。有关如何使用其他根 CA 的更多信息，请参阅《AWS IoT Core开发人员指南》中的创建自己的客户端证书。

4. 下载AWS IoT证书并将其导入 HSM。
5. 从指定使用 PKCS #11 提供程序组件以及 HSM 中的私钥和证书的配置文件中安装 C AWS IoT Greengrass ore 软件。

您可以选择以下安装选项之一来安装具有硬件安全性的 AWS IoT Greengrass Core 软件：

- 手动安装

  选择此选项可手动创建所需的AWS资源并配置硬件安全。有关更多信息，请参阅 使用手动资源配置来安装 AWS IoT Greengrass Core 软件。
- 使用自定义配置进行安装

  选择此选项可开发自定义 Java 应用程序，该应用程序可自动创建所需AWS资源并配置硬件安全。有关更多信息，请参阅 安装具有自定义资源配置功能的 C AWS IoT Greengrass ore 软件。

当前，在使用自动资源配置或AWS IoT队列配置进行安装时，AWS IoT Greengrass不支持安装具有硬件安全性的AWS IoT Greengrass核心软件。

在现有核心设备上配置硬件安全

您可以将核心设备的私钥和证书导入 HSM 以配置硬件安全。

> 注意事项
>
> - 您必须对核心设备的文件系统具有 root 用户访问权限。
> - 在此过程中，您将关闭 AWS IoT Greengrass Core 软件，因此在配置硬件安全时，核心设备处于离线状态且不可用。

要在现有核心设备上配置硬件安全，请执行以下操作：

1. 初始化 HSM。
2. 将 PKCS #11 提供程序组件部署到核心设备。
3. 停止AWS IoT Greengrass核心软件。
4. 将核心设备的私钥和证书导入 HSM。
5. 更新AWS IoT Greengrass核心软件的配置文件以使用 HSM 中的私钥和证书。

6. 启动AWS IoT Greengrass核心软件。

步骤 1：初始化硬件安全模块

完成以下步骤，在核心设备上初始化 HSM。

初始化硬件安全模块

- 在 HSM 中初始化 PKCS #11 令牌，然后保存该令牌的插槽 ID 和用户 PIN。查看 HSM 的文档，了解如何初始化令牌。稍后在部署和配置 PKCS #11 提供程序组件时，您将使用插槽 ID 和用户 PIN。

步骤 2：部署 PKCS #11 提供程序组件

完成以下步骤以部署和配置 PKCS #11 提供程序组件。您可以将该组件部署到一个或多个核心设备上。

部署 PKCS #11 提供程序组件（控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择组件。
2. 在 “组件” 页上，选择 “公共组件” 选项卡，然后选择aws.greengrass.crypto.Pkcs11Provider。
3. 在 aws.greengrass.crypto.Pkcs11Provider 页面上，选择部署。
4. 从 “添加到部署” 中，选择要修改的现有部署，或选择创建新部署，然后选择 “下一步”。
5. 如果您选择创建新部署，请为部署选择目标核心设备或事物组。在 “指定目标” 页面的 “部署目标” 下，选择核心设备或事物组，然后选择下一步。
6. 在 “选择组件” 页上的 “公共组件” 下，选择 aws.greengrass.crypto.Pkcs11Provider，然后选择 “下一步”。
7. 在 “配置组件” 页面上 aws.greengrass.crypto.Pkcs11Provider，选择，然后执行以下操作：

    a. 选择配置组件。
    b. 在 “配置” aws.greengrass.crypto.Pkcs11Provider 模式的 “配置更新” 下，在 “要合并的配置” 中，输入以下配置更新。使用目标核心设备的值更新以下配置参数。指定之前初始化 PKCS #11 令牌的插槽 ID 和用户 PIN。稍后，您将私钥和证书导入 HSM 中的此插槽。

        name

            PKCS #11 配置的名称。

library

AWS IoT Greengrass核心软件可以使用 libdl 加载的 PKCS #11 实现库的绝对文件路径。

slot

包含私钥和设备证书的插槽的 ID。此值不同于插槽索引或插槽标签。

userPin

用于访问插槽的用户 PIN。

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

c. 选择 “确认” 关闭模式，然后选择 “下一步”。

8. 在配置高级设置页面上，保留默认配置设置，然后选择下一步。
9. 在 Review（检查）页上，选择 Deploy（部署）。

   部署最多可能需要一分钟才能完成。

部署 PKCS #11 提供程序组件 () AWS CLI

要部署 PKCS #11 提供程序组件，请创建包含`aws.greengrass.crypto.Pkcs11Provider`在`components`对象中的部署文档，并指定该组件的配置更新。按照中的创建部署说明创建新部署或修改现有部署。

以下示例部分部署文档指定部署和配置 PKCS #11 提供程序组件。使用目标核心设备的值更新以下配置参数。保存插槽 ID 和用户 PIN，以便以后在将私钥和证书导入 HSM 时使用。

name

PKCS #11 配置的名称。

library

AWS IoT Greengrass核心软件可以使用 libdl 加载的 PKCS #11 实现库的绝对文件路径。

`slot`

包含私钥和设备证书的插槽的 ID。此值不同于插槽索引或插槽标签。

`userPin`

用于访问插槽的用户 PIN。

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.crypto.Pkcs11Provider": {
      "componentVersion": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"softhsm_pkcs11\",\"library\":\"/usr/lib/softhsm/
libsofthsm2.so\",\"slot\":1,\"userPin\":\"1234\"}"
      }
    }
  }
}
```

完成部署可能需要数分钟。您可以使用该AWS IoT Greengrass服务来检查部署状态。您可以查看AWS IoT Greengrass核心软件日志，以验证 PKCS #11 提供程序组件是否成功部署。有关更多信息，请参阅下列内容：

- 检查部署状态
- 监控AWS IoT Greengrass日志

如果部署失败，您可以对每台核心设备上的部署进行故障排除。有关更多信息，请参阅 故障排除 AWS IoT Greengrass V2。

步骤 3：更新核心设备上的配置

C AWS IoT Greengrass ore 软件使用指定设备运行方式的配置文件。此配置文件包括在哪里可以找到设备用于连接的私钥和证书AWS Cloud。完成以下步骤，将核心设备的私钥和证书导入 HSM，并更新配置文件以使用 HSM。

更新核心设备上的配置以使用硬件安全

1. 停止AWS IoT Greengrass核心软件。如果您使用 systemd 将AWS IoT Greengrass核心软件配置为系统服务，则可以运行以下命令来停止该软件。

```
sudo systemctl stop greengrass.service
```

2. 查找核心设备的私钥和证书文件。

   - 如果您安装了具有自动配置或队列配置功能的 C AWS IoT Greengrass ore 软件，则私钥存在于*/greengrass/v2*/privKey.key，证书存在于*/greengrass/v2*/thingCert.crt。
   - 如果您安装了手动配置的 AWS IoT Greengrass Core 软件，则*/greengrass/v2*/private.pem.key默认情况下私钥存在，证书*/greengrass/v2*/device.pem.crt默认存在。

   您也可以查看system.privateKeyPath和system.certificateFilePath属性*/greengrass/v2*/config/effectiveConfig.yaml以查找这些文件的位置。

3. 将私钥和证书导入 HSM。查看 HSM 的文档，了解如何将私钥和证书导入其中。使用之前初始化 PKCS #11 令牌的插槽 ID 和用户 PIN 导入私钥和证书。私钥和证书必须使用相同的对象标签和对象 ID。保存您在导入每个文件时指定的对象标签。稍后在更新 AWS IoT Greengrass Core 软件配置以使用 HSM 中的私钥和证书时，您将使用此标签。

4. 更新AWS IoT Greengrass核心配置以使用 HSM 中的私钥和证书。要更新配置，您可以修改AWS IoT Greengrass核心配置文件，然后使用更新的配置文件运行AWS IoT Greengrass核心软件以应用新的配置。

   执行以下操作：

   a. 创建AWS IoT Greengrass核心配置文件的备份。如果您在配置硬件安全时遇到问题，则可以使用此备份来恢复核心设备。

   ```
   sudo cp /greengrass/v2/config/effectiveConfig.yaml ~/ggc-config-backup.yaml
   ```

b. 在文本编辑器中打开AWS IoT Greengrass核心配置文件。例如，你可以运行以下命令来使用 GNU nano 来编辑文件。*/greengrass/v2*替换为 Greengrass 根文件夹的路径。

```
sudo nano /greengrass/v2/config/effectiveConfig.yaml
```

c. 将的值替换为 HSM 中私钥的 PKCS #11 URI。`system.privateKeyPath`将 *iotdevicek* ey 替换为之前导入私钥和证书的对象标签。

```
pkcs11:object=iotdevicekey;type=private
```

d. 将的值替换为 HSM 中证书的 PKCS #11 URI。`system.certificateFilePath`将 *iotdevicek* ey 替换为之前导入私钥和证书的对象标签。

```
pkcs11:object=iotdevicekey;type=cert
```

完成这些步骤后，AWS IoT GreengrassCore 配置文件中的`system`属性应与以下示例类似。

```
system:
  certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
  privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
  rootCaPath: "/greengrass/v2/rootCA.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
```

5. 在更新的`effectiveConfig.yaml`文件中应用配置。`Greengrass.jar`使用`--init-config`参数运行以在中应用配置`effectiveConfig.yaml`。*/greengrass/v2*替换为 Greengrass 根文件夹的路径。

```
sudo java -Droot="/greengrass/v2" \
  -jar /greengrass/v2/alts/current/distro/lib/Greengrass.jar \
  --start false \
  --init-config /greengrass/v2/config/effectiveConfig.yaml
```

6. 启动AWS IoT Greengrass核心软件。如果您使用 systemd 将AWS IoT Greengrass核心软件配置为系统服务，则可以运行以下命令来启动该软件。

```
sudo systemctl start greengrass.service
```

有关更多信息，请参阅 运行AWS IoT Greengrass核心软件。

7. 查看 AWS IoT Greengrass Core 软件日志，验证软件是否启动并连接到AWS Cloud。C AWS IoT Greengrass ore 软件使用私钥和证书连接到AWS IoT和AWS IoT Greengrass服务。

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

   以下信息级别的日志消息表明 AWS IoT Greengrass Core 软件已成功连接到AWS IoT和AWS IoT Greengrass服务。

```
2021-12-06T22:47:53.702Z [INFO] (Thread-3)
 com.aws.greengrass.mqttclient.AwsIotMqttClient: Successfully connected to AWS IoT
 Core. {clientId=MyGreengrassCore5, sessionPresent=false}
```

8. （可选）在验证AWS IoT Greengrass核心软件是否可以使用 HSM 中的私钥和证书后，从设备的文件系统中删除私钥和证书文件。运行以下命令，并将文件路径替换为私钥和证书文件的路径。

```
sudo rm /greengrass/v2/privKey.key
sudo rm /greengrass/v2/thingCert.crt
```

使用不支持 PKCS #11 的硬件

PKCS#11 库通常由硬件供应商提供或是开源软件。例如，对于符合标准的硬件（如 TPM1.2），可能会使用现有的开源软件。但是，如果您的硬件没有相应的 PKCS #11 库实现，或者您想编写自定义 PKCS #11 提供程序，请联系您的 Amazon Web Services Enterprise Support 代表提出与集成相关的问题。

另请参阅

- PKCS #11 加密令牌接口使用指南版本 2.4.0
- RFC 7512
- PKCS #1：RSA 加密版本 1.5

AWS IoT Greengrass 的设备身份验证和授权

AWS IoT Greengrass 环境中的设备使用 X.509 证书进行身份验证，使用 AWS IoT 策略进行授权。证书和策略可让设备在彼此之间以及与 AWS IoT Core 和 AWS IoT Greengrass 之间安全地相互连接。

X.509 证书属于数字证书，它按照 X.509 公有密钥基础设施标准将公有密钥与证书所含的身份相关联。X.509 证书由一家名为证书颁发机构 (CA) 的可信实体颁发。CA 持有一个或多个名为 CA 证书的特殊证书，它使用这种证书来颁发 X.509 证书。只有证书颁发机构才有权访问 CA 证书。

AWS IoT 策略定义允许 AWS IoT 设备执行的操作集。具体而言，它们允许和拒绝访问 AWS IoT Core 和 AWS IoT Greengrass 数据层面操作，如发布 MQTT 消息和检索设备阴影。

所有设备都需要在 AWS IoT Core 注册表中有条目，并且需要带附加 AWS IoT 策略的已激活的 X.509 证书。设备分为两类：

- Greengrass 核心设备

  Greengrass 核心设备使用证书AWS IoT和策略来连接和。AWS IoT Core AWS IoT Greengrass证书和策略还允许AWS IoT Greengrass将组件和配置部署到核心设备。
- 客户端设备

  MQTT 客户端设备使用证书和策略来连接AWS IoT Core和AWS IoT Greengrass服务。这使客户端设备能够使用AWS IoT Greengrass云发现来查找并连接到 Greengrass 核心设备。客户端设备使用相同的证书连接到AWS IoT Core云服务和核心设备。客户端设备还使用发现信息与核心设备进行双向身份验证。有关更多信息，请参阅 与本地物联网设备互动。

X.509 证书

核心设备与客户端设备之间以及设备和/或之间的通信AWS IoT Greengrass必须经过身份验证。AWS IoT Core此双向身份验证基于注册的 X.509 设备证书和加密密钥。

在 AWS IoT Greengrass 环境中，设备对以下传输层安全性 (TLS) 连接使用带有公钥和私钥的证书：

- Greengrass 核心设备上的AWS IoT客户端组件，用于连接和通过互联网进行连接AWS IoT Core。AWS IoT Greengrass
- 通过互联网连接以AWS IoT Greengrass发现核心设备的客户端设备。
- Greengrass 核心上的 MQTT 代理组件通过本地网络连接到群组中的 Greengrass 设备。

AWS IoT Greengrass核心设备将证书存储在 Greengrass 根文件夹中。

证书颁发机构 (CA) 证书

Greengrass 核心设备和客户端设备会下载用于对和服务进行身份验证的根 CA 证书。AWS IoT Core AWS IoT Greengrass我们建议您使用 Amazon Trust Services (ATS) 根 CA 证书，如 Amazon Root CA 1。有关更多信息，请参阅 AWS IoT Core 开发人员指南中的服务器身份验证的 CA 证书。

客户端设备还会下载 Greengrass 核心设备 CA 证书。在相互身份验证期间，他们使用此证书来验证核心设备上的 MQTT 服务器证书。

在本地 MQTT 代理上轮换证书

启用客户端设备支持后，Greengrass 核心设备会生成本地 MQTT 服务器证书，客户端设备使用该证书进行相互身份验证。此证书由核心设备 CA 证书签名，核心设备将其存储在AWS IoT Greengrass云中。客户端设备在发现核心设备时会检索核心设备 CA 证书。当他们连接到核心设备时，他们使用核心设备 CA 证书来验证核心设备的 MQTT 服务器证书。核心设备 CA 证书将在 5 年后过期。

默认情况下，MQTT 服务器证书每 7 天过期一次，您可以将此期限配置为 2 到 10 天之间。此时间期限基于安全最佳实践。这种轮换有助于缓解攻击者窃取 MQTT 服务器证书和私钥来冒充 Greengrass 核心设备的威胁。

Greengrass 核心设备会在 MQTT 服务器证书到期前 24 小时对其进行轮换。Greengrass 核心设备生成新证书并重新启动本地 MQTT 代理。发生这种情况时，所有连接到 Greengrass 核心设备的客户端设备都将断开连接。客户端设备可以在短时间后重新连接到 Greengrass 核心设备。

数据层面操作的 AWS IoT 策略

使用AWS IoT策略来授权访问AWS IoT Core和AWS IoT Greengrass数据平面。AWS IoT Core数据平面为设备、用户和应用程序提供操作。这些操作包括连接AWS IoT Core和订阅主题的功能。AWS IoT Greengrass数据平面为 Greengrass 设备提供操作。有关更多信息，请参阅 AWS IoT Greengrass V2 策略操作。这些操作包括解析组件依赖关系和下载公共组件工件的能力。

AWS IoT 策略是一个与 IAM 策略 类似的 JSON 文档。它包含一个或多个策略语句，用于指定以下属性：

- `Effect`。访问模式，可以是 `Allow` 或 `Deny`。
- `Action`. 策略允许或拒绝的操作的列表。
- `Resource`. 允许或拒绝对其执行操作的资源的列表。

AWS IoT 策略支持 * 作为通配符，并将 MQTT 通配符（+和#）视为文字字符串。有关 * 通配符的更多信息，请参阅《AWS Identity and Access Management 用户指南》中的在资源 ARN 中使用通配符。

有关更多信息，请参阅 AWS IoT Core 开发人员指南中的 AWS IoT策略和 AWS IoT 策略操作。

> **Important**
>
> 核心设备或 Greengrass 数据平面操作的AWS IoT策略不支持@@ 事物策略变量 (`iot:Connection.Thing.*`)。相反，您可以使用通配符来匹配多个名称相似的设备。例如，您可以指定`MyGreengrassDevice*`匹配`MyGreengrassDevice1``MyGreengrassDevice2`、等。

> **Note**
>
> AWS IoT Core 允许您将 AWS IoT 策略附加到事物组，以定义设备组的权限。事物组策略不允许访问 AWS IoT Greengrass 数据面板操作。要允许事物访问 AWS IoT Greengrass 数据面板操作，请将权限添加到您附加到事物证书的 AWS IoT 策略中。

AWS IoT Greengrass V2 策略操作

AWS IoT Greengrass V2定义了 Greengrass 核心设备和客户端设备可以在策略中使用的以下策略操作。AWS IoT要为策略操作指定资源，您可以使用该资源的 Amazon 资源名称 (ARN)。

核心设备操作

`greengrass:GetComponentVersionArtifact`

授予获取预签名 URL 的权限，以下载公共组件工件或 Lambda 组件工件。

当核心设备收到指定公共组件的部署或包含工件的 Lambda 时，将评估此权限。如果核心设备已经有工件，则它不会再次下载该工件。

资源类型：`componentVersion`

资源 ARN 格式：`arn:aws:greengrass:`*`region`*`:`*`account-id`*`:components:`*`component-name`*`:versions:`*`component-version`*

`greengrass:ResolveComponentCandidates`

授予识别满足部署组件、版本和平台要求的组件列表的权限。如果需求冲突，或者不存在符合要求的组件，则此操作将返回错误，并且设备上的部署将失败。

当核心设备收到指定组件的部署时，将评估此权限。

资源类型：无

资源 ARN 格式：*

`greengrass:GetDeploymentConfiguration`

授予获取用于下载大型部署文档的预签名 URL 的权限。

当核心设备收到的部署指定的部署文档大于 7 KB（如果部署以事物为目标）或 31 KB（如果部署以事物组为目标）时，将评估此权限。部署文档包括组件配置、部署策略和部署元数据。有关更多信息，请参阅 将AWS IoT Greengrass组件部署到设备。

此功能适用于 2.3.0 及更高版本的 Greengrass nucleus 组件。

资源类型：无

资源 ARN 格式：*

`greengrass:ListThingGroupsForCoreDevice`

授予获取核心设备的事物组层次结构的权限。

当核心设备收到来自的部署时，将检查此权限AWS IoT Greengrass。核心设备使用此操作来确定自上次部署以来是否已从事物组中移除。如果核心设备已从事物组中移除，并且该事物组是部署到核心设备的目标，则该核心设备将移除该部署安装的组件。

Greengrass nucleus 组件的 2.5.0 及更高版本使用此功能。

资源类型：`thing`（核心设备）

资源 ARN 格式：`arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:VerifyClientDeviceIdentity`

授予验证连接到核心设备的客户端设备身份的权限。

当核心设备运行客户端设备身份验证组件并收到来自客户端设备的 MQTT 连接时，将评估此权限。客户端设备出示其AWS IoT设备证书。然后，核心设备将设备证书发送到AWS IoT Greengrass云服务以验证客户端设备的身份。有关更多信息，请参阅 与本地物联网设备互动。

资源类型：无

资源 ARN 格式：*

`greengrass:VerifyClientDeviceIoTCertificateAssociation`

授予验证客户端设备是否与AWS IoT证书关联的权限。

当核心设备运行客户端设备身份验证组件并授权客户端设备通过 MQT T 进行连接时，将评估此权限。有关更多信息，请参阅 与本地物联网设备互动。

> **Note**
>
> 要使核心设备使用此操作，Greengrass 服务角色必须与您的关联并允许AWS 账户该权限。`iot:DescribeCertificate`

资源类型：`thing`（客户端设备）

资源 ARN 格式：`arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

`greengrass:PutCertificateAuthorities`

授予上传证书颁发机构 (CA) 证书的权限，客户端设备可以下载这些证书来验证核心设备。

当核心设备安装并运行客户端设备身份验证组件时，将评估此权限。此组件创建本地证书颁发机构，并使用此操作上传其 CA 证书。当客户端设备使用 D iscover 操作查找可以连接的核心设备时，它们会下载这些 CA 证书。当客户端设备连接到核心设备上的 MQTT 代理时，它们使用这些 CA 证书来验证核心设备的身份。有关更多信息，请参阅 与本地物联网设备互动。

资源类型：无

ARN 格式：*

`greengrass:GetConnectivityInfo`

授予获取核心设备连接信息的权限。此信息描述了客户端设备如何连接到核心设备。

当核心设备安装并运行客户端设备身份验证组件时，将评估此权限。此组件使用连接信息生成有效的 CA 证书，以便通过PutCertificateAuthories操作上传到AWS IoT Greengrass云服务。客户端设备使用这些 CA 证书来验证核心设备的身份。有关更多信息，请参阅 与本地物联网设备互动。

您也可以在AWS IoT Greengrass控制平面上使用此操作来查看核心设备的连接信息。有关更多信息，请参阅《AWS IoT Greengrass V1 API 参考》中的 GetConnectivityInfo。

资源类型：`thing`（核心设备）

资源 ARN 格式：`arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:UpdateConnectivityInfo`

授予更新核心设备连接信息的权限。此信息描述了客户端设备如何连接到核心设备。

当核心设备运行 IP 检测器组件时，将评估此权限。此组件标识客户端设备连接到本地网络上的核心设备所需的信息。然后，此组件使用此操作将连接信息上传到AWS IoT Greengrass云服务，这样客户端设备就可以通过 D iscover 操作检索此信息。有关更多信息，请参阅 与本地物联网设备互动。

您也可以在AWS IoT Greengrass控制平面上使用此操作来手动更新核心设备的连接信息。有关更多信息，请参阅《AWS IoT Greengrass V1 API 参考》中的 UpdateConnectivityInfo。

资源类型：`thing`（核心设备）

资源 ARN 格式：`arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

客户端设备操作

`greengrass:Discover`

授予发现客户端设备可以连接的核心设备的连接信息的权限。此信息描述了客户端设备如何连接到核心设备。通过使用该BatchAssociateClientDeviceWithCoreDevice操作，客户端设备只能发现您与之关联的核心设备。有关更多信息，请参阅 与本地物联网设备互动。

资源类型：`thing`（客户端设备）

资源 ARN 格式：`arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

更新核心设备的AWS IoT政策

您可以使用AWS IoT Greengrass和AWS IoT控制台或 AWS IoT API 来查看和更新核心设备的AWS IoT政策。

> **Note**
>
> 如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则您的核心设备具有允许访问所有AWS IoT Greengrass操作的AWS IoT策略（`greengrass:*`）。您可以按照以下步骤将访问权限限制为仅访问核心设备使用的操作。

查看和更新核心设备的AWS IoT政策（控制台）

1. 在AWS IoT Greengrass控制台导航菜单中，选择核心设备。
2. 在核心设备页面上，选择要更新的核心设备。
3. 在核心设备详细信息页面上，选择指向核心设备的 Thin g 的链接。此链接可在AWS IoT控制台中打开事物详细信息页面。
4. 在事物详细信息页面上，选择证书。
5. 在 “证书” 选项卡中，选择事物的有效证书。
6. 在证书详细信息页面上，选择策略。
7. 在 “策略” 选项卡中，选择要查看和更新的AWS IoT策略。您可以向附加到核心设备活动证书的任何策略添加所需的权限。

   > **Note**
   >
   > 如果您使用AWS IoT Greengrass核心软件安装程序来配置资源，则有两个AWS IoT策略。我们建议您选择名为GreengrassV2IoTThingPolicy的策略（如果存在）。默认情况下，使用快速安装程序创建的核心设备使用此策略名称。如果您向此策略添加权限，则也将这些权限授予使用此策略的其他核心设备。

8. 在策略概述中，选择编辑活动版本。
9. 查看策略，并根据需要添加、删除或编辑权限。
10. 要将新的策略版本设置为活动版本，请在策略版本状态下，选择将编辑后的版本设置为该策略的活动版本。
11. 选择另存为新版本。

查看并更新核心设备的AWS IoT政策 (AWS CLI)

1. 列出核心设备的原理AWS IoT。事物主体可以是 X.509 设备证书或其他标识。运行以下命令，并*MyGreengrassCore*替换为核心设备的名称。

```
aws iot list-thing-principals --thing-name MyGreengrassCore
```

该操作返回一个响应，其中列出了核心设备的事物主体。

```
{
    "principals": [
        "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
    ]
}
```

2. 识别核心设备的活动证书。运行以下命令，将 cert *i* ficateID 替换为上一步中每个证书的 ID，直到找到活动证书。证书 ID 是证书 ARN 末尾的十六进制字符串。该`--query`参数指定仅输出证书的状态。

```
aws iot describe-certificate --certificate-id certificateId --query
 'certificateDescription.status'
```

该操作以字符串形式返回证书状态。例如，如果证书处于活动状态，则此操作会输出`"ACTIVE"`。

3. 列出附加到证书的AWS IoT策略。运行以下命令，并将证书 ARN 替换为证书的 ARN。

```
aws iot list-principal-policies --principal arn:aws:iot:us-
west-2:123456789012:cert/certificateId
```

该操作会返回一个响应，其中列出了附加到证书的AWS IoT策略。

```
{
    "policies": [
        {
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
```

```
}
```

4. 选择要查看和更新的策略。

> **Note**
>
> 如果您使用[AWS IoT Greengrass核心软件安装程序来配置资源](#)，则有两个AWS IoT策略。我们建议您选择名为GreengrassV2IoTThingPolicy的策略（如果存在）。默认情况下，使用快速安装程序创建的核心设备使用此策略名称。如果您向此策略添加权限，则也将这些权限授予使用此策略的其他核心设备。

5. 获取保单文件。运行以下命令，将 *GreenGrassv2IoT #####ThingPolicy###*。

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

该操作会返回一个响应，其中包含策略的文档和有关该策略的其他信息。策略文档是一个序列化为字符串的 JSON 对象。

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
```

```
    "generationId":
  "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. 使用在线转换器或其他工具将策略文档字符串转换为 JSON 对象，然后将其保存到名为的文件中`iot-policy.json`。

   例如，如果您安装了 [jq](https://stedolan.github.io/jq/) 工具，则可以运行以下命令来获取策略文档，将其转换为 JSON 对象，然后将策略文档另存为 JSON 对象。

   ```
   aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
    'policyDocument' | jq fromjson >> iot-policy.json
   ```

7. 查看策略文档，并根据需要添加、删除或编辑权限。

   例如，在基于 Linux 的系统上，你可以运行以下命令来使用 GNU nano 打开文件。

   ```
   nano iot-policy.json
   ```

   完成后，策略文档可能与核心设备的最低AWS IoT策略类似。

8. 将更改保存为新版本的策略。运行以下命令，将 *GreenGrassv2IoT #####ThingPolicy###*。

   ```
   aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
   document file://iot-policy.json --set-as-default
   ```

   如果操作成功，则返回类似于以下示例的响应。

   ```
   {
       "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
   GreengrassV2IoTThingPolicy",
       "policyDocument": "{\
     \\"Version\\": \\"2012-10-17\\",\
     \\"Statement\\": [\
       {\
         \\"Effect\\": \\"Allow\\",\
         \\"Action\\": [\
   \\t\\t\\"iot:Connect\\",\
   \\t\\t\\"iot:Publish\\",\
   \\t\\t\\"iot:Subscribe\\",\
   \\t\\t\\"iot:Receive\\",\
   ```

```
\\t\\t\\"greengrass:*\\"\
      ],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

AWS IoT Greengrass V2核心设备的最低AWS IoT政策

> **Important**
>
> Greengrass nucleus 组件的更高版本需要对最低策略的额外权限。AWS IoT您可能需要更新核心设备的AWS IoT策略才能授予其他权限。
>
> - 从事物组中移除核心设备时，运行 Greengrass nucleus v2.5.0 及更高版本greengrass:ListThingGroupsForCoreDevice的核心设备使用卸载组件的权限。
> - 运行 Greengrass nucleus v2.3.0 及更高版本的核心设备使用greengrass:GetDeploymentConfiguration该权限来支持大型部署配置文档。

以下示例策略包含为支持核心设备的基本 Greengrass 功能所需的一组最少操作。

- 该Connect策略在核心设备事物名称后面包含*通配符（例如*core-device-thing-name**）。核心设备使用相同的设备证书进行多个并行订阅AWS IoT Core，但是连接中的客户端 ID 可能与核心设备的事物名称不完全匹配。在前 50 个订阅之后，核心设备将*core-device-thing-name*#*number*用作客户端 ID，其中每*number*增加 50 个订阅就会递增。例如，当名为的核心设备MyCoreDevice创建 150 个并发订阅时，它将使用以下客户端 ID：
  - 订阅 1 到 50：MyCoreDevice
  - 订阅 51 到 100：MyCoreDevice#2
  - 订阅 101 到 150：MyCoreDevice#3

  通配符允许核心设备在使用这些带有后缀的客户端 ID 时进行连接。
- 该策略列出了核心设备可将消息发布到、订阅和从中接收消息的 MQTT 主题和主题筛选条件（包括用于影子状态的主题）。要支持 Greengrass 组件和客户端设备之间的AWS IoT Core消息交换，请

指定要允许的主题和主题过滤器。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的发布/订阅策略示例。

- 该政策允许向以下主题发布遥测数据。

```
$aws/things/core-device-thing-name/greengrass/health/json
```

对于禁用遥测功能的核心设备，您可以移除此权限。有关更多信息，请参阅 从AWS IoT Greengrass 核心设备收集系统运行状况遥测数据。

- 该策略授予通过角色别名担任 IAM AWS IoT 角色的权限。核心设备使用此角色（称为令牌交换角色）来获取可用于对AWS请求进行身份验证的AWS凭证。有关更多信息，请参阅 授权核心设备与AWS服务。

安装 AWS IoT Greengrass Core 软件时，您可以创建并附加仅包含此权限的第二个AWS IoT策略。如果您在核心设备的主AWS IoT策略中包含此权限，则可以分离和删除其他AWS IoT策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:region:account-id:client/core-device-thing-name*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive",
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/greengrass/health/json",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/greengrassv2/health/json",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/jobs/*",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/shadow/*"
```

```
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name/jobs/*",
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:region:account-id:rolealias/token-exchange-role-
alias-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetComponentVersionArtifact",
                "greengrass:ResolveComponentCandidates",
                "greengrass:GetDeploymentConfiguration",
                "greengrass:ListThingGroupsForCoreDevice"
            ],
            "Resource": "*"
        }
    ]
}
```

支持客户端设备的最低AWS IoT政策

以下示例策略包括支持在核心设备上与客户端设备交互所需的最低操作集。要支持客户端设备，除了基本操作的最低AWS IoT策略外，核心设备还必须具有此AWS IoT策略中的权限。

- 该政策允许核心设备更新自己的连接信息。仅当将 IP 检测器组件部署到核心设备时，才需要此权限 (`greengrass:UpdateConnectivityInfo`)。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/get"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name-gci/shadow/update/delta",
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/update/delta",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:PutCertificateAuthorities",
                "greengrass:VerifyClientDeviceIdentity"
            ],
            "Resource": "*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:VerifyClientDeviceIoTCertificateAssociation"
            ],
            "Resource": "arn:aws:iot:region:account-id:thing/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-device-thing-name"
            ]
        }
    ]
}
```

客户端设备的最低AWS IoT政策

以下示例策略包括客户端设备发现其通过 MQTT 进行连接和通信的核心设备所需的最低操作集。客户端设备的AWS IoT策略必须包括允许设备发现其关联的 Greengrass 核心设备的连接信息的`greengrass:Discover`操作。在该`Resource`部分中，指定客户端设备的亚马逊资源名称 (ARN)，而不是 Greengrass 核心设备的 ARN。

- 该策略允许就所有 MQTT 主题进行通信。要遵循最佳安全实践，请将`iot:Publish``iot:Subscribe`、和`iot:Receive`权限限制为客户端设备在您的用例中所需的最少一组主题。
- 该政策允许事物发现所有AWS IoT事物的核心设备。要遵循最佳安全实践，请将`greengrass:Discover`权限限制为客户端设备的AWS IoT东西或与一组AWS IoT内容匹配的通配符。

> **Important**
>
> 核心设备或 Greengrass 数据平面操作的AWS IoT策略不支持@@ 事物策略变量 (`iot:Connection.Thing.*`)。相反，您可以使用通配符来匹配多个名称相似的设备。例如，您可以指定`MyGreengrassDevice*`匹配`MyGreengrassDevice1``MyGreengrassDevice2`、等。

- 客户端设备的AWS IoT策略通常不需要、或iot:DeleteThingShadow操作的权限 iot:GetThingShadowiot:UpdateThingShadow，因为 Greengrass 核心设备负责处理客户端设备的影子同步操作。要使核心设备能够处理客户端设备影子，请检查核心设备的AWS IoT策略是否允许这些操作，以及该Resource部分是否包含客户端设备的 ARN。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/*"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:Discover"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/*"
            ]
        }
    ]
}
```

适用于 AWS IoT Greengrass 的身份和访问管理

AWS Identity and Access Management (IAM) 是一项 AWS 服务，可以帮助管理员安全地控制对 AWS 资源的访问。IAM 管理员控制谁可以通过身份验证（登录）和授权（具有权限）来使用 AWS IoT Greengrass 资源。IAM 是一项无需额外费用即可使用的 AWS 服务。

> **Note**
>
> 本主题介绍 IAM 的概念和功能。有关 AWS IoT Greengrass 支持的 IAM 功能的信息，请参阅 the section called “AWS IoT Greengrass 如何与 IAM 协同工作”。

受众

使用 AWS Identity and Access Management (IAM) 的方式因您可以在 AWS IoT Greengrass 中执行的操作而异。

服务用户 - 如果使用 AWS IoT Greengrass 服务来完成任务，则您的管理员会为您提供所需的凭证和权限。当您使用更多 AWS IoT Greengrass 特征来完成工作时，您可能需要额外权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 AWS IoT Greengrass 中的特征，请参阅排查 AWS IoT Greengrass 的身份和访问权限问题。

服务管理员 - 如果您在公司负责管理 AWS IoT Greengrass 资源，则您可能具有 AWS IoT Greengrass 的完全访问权限。您有责任确定您的服务用户应访问哪些 AWS IoT Greengrass 特征和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 AWS IoT Greengrass 搭配使用的更多信息，请参阅AWS IoT Greengrass 如何与 IAM 协同工作。

IAM 管理员 - 如果您是 IAM 管理员，您可能希望了解如何编写策略以管理对 AWS IoT Greengrass 的访问权限的详细信息。要查看您可在 IAM 中使用的 AWS IoT Greengrass 基于身份的策略示例，请参阅 适用于 AWS IoT Greengrass 的基于身份的策略示例。

使用身份进行身份验证

身份验证是使用身份凭证登录 AWS 的方法。您必须作为 AWS 账户根用户、IAM 用户或通过分派 IAM 角色进行身份验证（登录到 AWS）。

您可以使用通过身份源提供的凭证以联合身份登录到 AWS。AWS IAM Identity Center（IAM Identity Center）用户、您公司的单点登录身份验证以及您的 Google 或 Facebook 凭证都是联合身份的示例。当您以联合身份登录时，管理员以前使用 IAM 角色设置了身份联合验证。当您使用联合身份验证访问 AWS 时，您就是在间接分派角色。

根据用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录到 AWS 的更多信息，请参阅《AWS 登录 用户指南》中的如何登录到您的 AWS 账户。

如果您以编程方式访问 AWS，则 AWS 将提供软件开发工具包（SDK）和命令行界面（CLI），以便使用您的凭证以加密方式签署您的请求。如果您不使用 AWS 工具，则必须自行对请求签名。有关使用推荐的方法自行签署请求的更多信息，请参阅《IAM 用户指南》中的签署 AWS API 请求。

无论使用何种身份验证方法，您都可能需要提供其它安全信息。例如，AWS 建议您使用多重身份验证（MFA）来提高账户的安全性。要了解更多信息，请参阅《AWS IAM Identity Center 用户指南》中的多重身份验证和《IAM 用户指南》中的在 AWS 中使用多重身份验证 (MFA)。

AWS 账户 根用户

创建 AWS 账户 时，最初使用的是一个对账户中所有 AWS 服务 和资源拥有完全访问权限的登录身份。此身份称为 AWS 账户根用户，使用您创建账户时所用的电子邮件地址和密码登录，即可获得该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

IAM 用户和组

IAM 用户是 AWS 账户 内对某个人员或应用程序具有特定权限的一个身份。在可能的情况下，建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的对于需要长期凭证的使用场景定期轮换访问密钥。

IAM 组是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人分派。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

IAM 角色是 AWS 账户 中具有特定权限的身份。它类似于 IAM 用户，但与特定人员不关联。您可以通过切换角色，在 AWS Management Console 中暂时分派 IAM 角色。您可以调用 AWS CLI 或 AWS API 操作或使用自定义网址以分派角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 - 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的权限集。
- 临时 IAM 用户权限 – IAM 用户或角色可分派 IAM 角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）访问您的账户中的资源。角色是授予跨账户存取权限的主要方式。但是，对于某些 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为座席）。要了解用于跨账户存取的角色和基于资源的策略之间的差别，请参阅《IAM 用户指南》中的 IAM 角色与基于资源的策略有何不同。
- 跨服务访问 – 某些 AWS 服务使用其它 AWS 服务中的功能。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Amazon S3 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话：当您使用 IAM 用户或角色在 AWS 中执行操作时，您将被视为主体。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中启动另一个操作。FAS 使用主体调用 AWS 服务 的权限，结合请求的 AWS 服务，向下游服务发出请求。只有在服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详细信息，请参阅转发访问会话。
  - 服务角色 - 服务角色是服务代表您在您的账户中执行操作而分派的 IAM 角色。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

  - 服务相关角色 - 服务相关角色是与 AWS 服务关联的一种服务角色。服务可以分派代表您执行操作的角色。服务相关角色显示在您的 AWS 账户中，并由该服务拥有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 Amazon EC2 上运行的应用程序 - 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时凭证。这优先于在 EC2 实例中存储访问密钥。要将 AWS 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，您可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅《IAM 用户指南》中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您将创建策略并将其附加到 AWS 身份或资源，以控制 AWS 中的访问。策略是 AWS 中的对象；在与身份或资源相关联时，策略定义它们的权限。在主体（用户、根用户或角色会话）发出请求时，AWS 将评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略在 AWS 中存储为 JSON 文档。有关 JSON 策略文档的结构和内容的更多信息，请参阅《IAM 用户指南》中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM 策略，然后用户就可以代入角色。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设有一个允许 `iam:GetRole` 操作的策略。具有该策略的用户可以从 AWS Management Console、AWS CLI 或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅《IAM 用户指南》中的 创建 IAM 策略。

基于身份的策略可以进一步归类为内联策略或托管式策略。内联策略直接嵌入单个用户、组或角色中。托管式策略是可以附加到AWS 账户中的多个用户、组和角色的独立策略。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅《IAM 用户指南》中的在托管式策略与内联策略之间进行选择。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定主体可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定主体。主体可以包括账户、用户、角色、联合用户或 AWS 服务。

基于资源的策略是位于该服务中的内联策略。您不能在基于资源的策略中使用来自 IAM 的 AWS 托管策略。

访问控制列表（ACL）

访问控制列表（ACL）控制哪些主体（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3、AWS WAF 和 Amazon VPC 是支持 ACL 的服务示例。要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》 中的访问控制列表（ACL）概览。

其他策略类型

AWS 支持额外的、不太常用的策略类型。这些策略类型可以设置更常用的策略类型授予的最大权限。

- 权限边界 – 权限边界是一个高级功能，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可以为实体设置权限边界。这些结果权限是实体基于身份的策略及其权限边界的交集。在 `Principal` 字段中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅《IAM 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略（SCP）– SCP 是 JSON 策略，指定了组织或组织单位（OU）在 AWS Organizations 中的最大权限。AWS Organizations 服务可以分组和集中管理您的企业拥有的多个 AWS 账户 账户。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略（SCP）。SCP 限制成员账户中实体（包括每个 AWS 账户根用户）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 - 会话策略是当您以编程方式为角色或联合用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅《IAM 用户指南》中的会话策略。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解 AWS 如何确定在涉及多种策略类型时是否允许请求，请参阅《IAM 用户指南》中的策略评估逻辑。

另请参阅

- the section called “AWS IoT Greengrass 如何与 IAM 协同工作”
- the section called “基于身份的策略示例”
- the section called “排查身份和访问权限问题”

AWS IoT Greengrass 如何与 IAM 协同工作

在使用 IAM 管理对的访问之前AWS IoT Greengrass，您应了解您可以使用的 IAM 功能AWS IoT Greengrass。

IAM 功能	Greengrass 支持吗？
具有资源级权限的基于身份的策略	是
基于资源的策略	否
访问控制列表 (ACL)	否
基于标签的授权	是
临时凭证	是
服务相关角色	否
服务角色	是

要大致了解其他AWS服务如何与 IAM 一起使用，请参阅 IAM 用户指南中的与 IAM 一起使用的AWS服务。

适用于 AWS IoT Greengrass 的基于身份的策略

使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。AWS IoT Greengrass 支持特定的操作、资源和条件密钥。要了解您在策略中使用的所有元素，请参阅 IAM 用户指南中的 IAM JSON 策略元素参考。

操作

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体 可以对什么资源执行操作，以及在什么 条件 下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行相关操作的权限。

AWS IoT Greengrass 的策略操作在操作前使用 `greengrass:` 前缀。例如，要允许某人使用`ListCoreDevices` API 操作列出其中的核心设备AWS 账户，您应将`greengrass:ListCoreDevices`操作纳入其策略中。策略语句必须包括 `Action` 或 `NotAction` 元素。AWS IoT Greengrass 定义了自己的一组操作，这些操作描述了可使用该服务执行的任务。

要在单个语句中指定多项操作，请将操作列在方括号 (`[]`) 中，并使用逗号将它们隔开，如下所示：

```
"Action": [
  "greengrass:action1",
  "greengrass:action2",
  "greengrass:action3"
]
```

您可以使用通配符 (*) 指定多个操作。例如，要指定以单词 `List` 开头的所有操作，包括以下操作：

```
"Action": "greengrass:List*"
```

> **Note**
>
> 我们建议您避免使用通配符来指定服务的所有可用操作。最佳做法是，您应在策略中授予最低特权和范围狭窄的权限。有关更多信息，请参阅the section called “授予可能的最低权限”：

有关AWS IoT Greengrass操作的完整列表，请参阅 IAM 用户指南AWS IoT Greengrass中定义的操作。

资源

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件 下执行。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称（ARN）指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符（*）指示语句应用于所有资源。

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

下表包含可在策略语句的 Resource 元素中使用的 AWS IoT Greengrass 资源 ARN。有关支持的资源级AWS IoT Greengrass操作权限的映射，请参阅 IAM 用户指南AWS IoT Greengrass中定义的操作。

某些 AWS IoT Greengrass 操作（例如，某些列表操作）不能在特定资源上执行。在这种情况下，您必须单独使用通配符。

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

要在语句中指定多个资源 ARN，请将它们列在方括号 ([]) 之间并用逗号分隔，如下所示：

```
"Resource": [
  "resource-arn1",
  "resource-arn2",
  "resource-arn3"
]
```

有关 ARN 格式的更多信息，请参阅中的亚马逊资源名称 (ARN) 和AWS服务命名空间Amazon Web Services 一般参考。

条件键

管理员可以使用 AWS JSON 策略来指定谁有权访问什么内容。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则 AWS 使用逻辑 OR 运算来评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM policy 元素：变量和标签。

AWS支持全局条件键和特定于服务的条件键。要查看所有 AWS 全局条件键，请参阅《IAM 用户指南》中的 AWS 全局条件上下文键。

示例

要查看 AWS IoT Greengrass 基于身份的策略的示例，请参阅 the section called “基于身份的策略示例”。

AWS IoT Greengrass 的基于资源的策略

AWS IoT Greengrass 不支持基于资源的策略。

访问控制列表 (ACL)

AWS IoT Greengrass 不支持 ACL。

基于 AWS IoT Greengrass 标签的授权

您可以将标签附加到支持的 AWS IoT Greengrass 资源，或将请求中的标签传递到 AWS IoT Greengrass。要基于标签控制访问，您需要在策略的 Condue 元素中使用`aws:ResourceTag/${TagKey}aws:RequestTag/${TagKey}`、或`aws:TagKeys`条件键提供标签信息。有关更多信息，请参阅标记资源：

的 IAM 角色AWS IoT Greengrass

IAM 角色是 AWS 账户中具有特定权限的实体。

将临时凭证用于 AWS IoT Greengrass

临时凭证用于联合身份登录，担任 IAM 角色或担任角色或担任角色或担任角色或担任角色使用。您可以通过调用AWS STS API 操作（例如AssumeRole或）获取临时安全凭证GetFederationToken。

在 Greengrass 核心上，设备角色的临时证书可供Greengrass组件使用。如果您的组件使用AWS SDK，则无需添加逻辑即可获取证书，因为AWS SDK 会为您执行此操作。

服务相关角色

AWS IoT Greengrass 不支持服务相关角色。

服务角色

此功能允许服务代表您担任服务角色。此角色允许服务访问其它服务中的资源以代表您完成操作。服务角色显示在您的 IAM 账户中，并归该账户所有。这意味着，IAM 管理员可以更改该角色的权限。但是，这样做可能会中断服务的功能。

AWS IoT Greengrass核心设备使用服务角色允许 Greengrass 组件和 Lambda 函数代表您访问您的某些AWS资源。有关更多信息，请参阅the section called “授权核心设备与AWS服务”：

AWS IoT Greengrass 使用服务角色代表您访问某些 AWS 资源。有关更多信息，请参阅 Greengrass 服务角色。

适用于 AWS IoT Greengrass 的基于身份的策略示例

预设情况下，IAM 用户和角色没有创建或修改 AWS IoT Greengrass 资源的权限。它们还无法使用 AWS Management Console、AWS CLI 或 AWS API 执行任务。IAM 管理员必须创建 IAM policy，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的 IAM 用户或组。

策略最佳实践

基于身份的策略确定某个人是否可以创建、访问或删除您账户中的 AWS IoT Greengrass 资源。这些操作可能会使 AWS 账户 产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- AWS 托管策略及转向最低权限许可入门 - 要开始向用户和工作负载授予权限，请使用 AWS 托管策略来为许多常见使用场景授予权限。您可以在 AWS 账户 中找到这些策略。我们建议通过定义特定于您的使用场景的 AWS 客户管理型策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略或工作职能的 AWS 托管策略。

- 应用最低权限 – 在使用 IAM policy 设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM policy 中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果通过特定 AWS 服务（例如 AWS CloudFormation）使用服务操作，您还可以使用条件来授予对服务操作的访问权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件。
- 使用 IAM Access Analyzer 验证您的 IAM policy，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM policy语言 (JSON) 和 IAM 最佳实践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。
- 需要多重身份验证（MFA）：如果您的 AWS 账户 需要 IAM 用户或根用户，请启用 MFA 来提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实践的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。

策略示例

以下客户定义的策略示例可为常见方案授予权限。

示例

- 允许用户查看他们自己的权限

要了解如何使用这些示例 JSON 策略文档创建 IAM 基于身份的策略，请参阅 IAM 用户指南中的在 JSON 选项卡上创建策略。

允许用户查看他们自己的权限

该示例说明了您如何创建策略，以允许 IAM 用户查看附加到其用户身份的内联和托管式策略。此策略包括在控制台上完成此操作或者以编程方式使用 AWS CLI 或 AWS API 所需的权限。

```
{
    "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": "*"
        }
    ]
}
```

授权核心设备与AWS服务

AWS IoT Greengrass核心设备使用AWS IoT Core授权调用的凭证提供商AWS服务。这些区域有：AWS IoT Core凭据提供程序允许设备使用其 X.509 证书作为唯一的设备身份进行身份验证AWS请求。这样，您就不必存储AWS您的访问密钥 ID 和秘密访问密钥AWS IoT Greengrass核心设备。有关更多信息，请参阅 。授予直接拨打AWS服务中的AWS IoT Core开发人员指南.

当你运行AWS IoT Greengrass核心软件，你可以选择配置AWS核心设备需要的资源。这包括AWS Identity and Access Management您的核心设备通过AWS IoT Core凭证提供程序。使用`--provision true`参数来配置允许核心设备临时获得的角色和策略AWS凭证。这个参数还配置了AWS IoT指向此 IAM 角色的角色别名。您可以指定 IAM 角色的名称，AWS IoT要使用的角色别名。如果你指定`--provision true`如果没有这些其他名称参数，Greengrass 核心设备将创建并使用以下默认资源：

- IAM 角色：`GreengrassV2TokenExchangeRole`

此角色的策略名为GreengrassV2TokenExchangeRoleAccess以及允许的信任关系credentials.iot.amazonaws.com来代入该角色。该策略包括核心设备的最低权限。

> **Important**
>
> 此策略不包括访问 S3 存储桶中的文件。您必须向角色添加权限，以允许核心设备从 S3 存储桶中检索组件项目。有关更多信息，请参阅 允许访问 S3 存储桶以获取组件项目。

- AWS IoT角色别名：GreengrassV2TokenExchangeRoleAlias

  此角色别名是指 IAM 角色。

有关更多信息，请参阅 步骤 3 安装AWS IoT Greengrass核心软件。

您还可以为现有核心设备设置角色别名。为此，请配置iotRoleAlias的配置参数Greengrass 核心组件.

你可以临时购买AWS为此 IAM 角色执行凭证AWS自定义组件中的操作。有关更多信息，请参阅 与AWS服务互动。

主题

- 核心设备的服务角色权限
- 允许访问 S3 存储桶以获取组件项目

核心设备的服务角色权限

此角色允许以下服务代入该角色：

- credentials.iot.amazonaws.com

如果您将AWS IoT Greengrass核心软件创建此角色时，它使用以下权限策略允许核心设备连接并将日志发送到AWS. 策略的名称默认为 IAM 角色的名称以结尾Access. 例如，如果您使用默认的 IAM 角色名称，则此策略的名称为GreengrassV2TokenExchangeRoleAccess.

Greengrass nucleus v2.5.0 and later

```
{
  "Version": "2012-10-17",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

v2.4.x

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Earlier than v2.4.0

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

```
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

允许访问 S3 存储桶以获取组件项目

默认核心设备角色不允许核心设备访问 S3 存储桶。要部署 S3 存储桶中包含工件的组件，必须添加`s3:GetObject`允许核心设备下载组件工件的权限。您可以向核心设备角色添加新策略以授予此权限。

添加允许访问 Amazon S3 中组件工件的策略

1. 创建一个名为的文件`component-artifact-policy.json`然后将以下 JSON 复制到该文件中。此策略允许访问 S3 存储桶中的所有文件。Replace*DOC-###*以及允许核心设备访问的 S3 存储桶的名称。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
  ]
}
```

2. 运行以下命令以从中的策略文档创建策略：`component-artifact-policy.json`.

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

   PowerShell

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

   从输出中的策略元数据复制策略 Amazon 资源名称 (ARN)。您可以在下一步中使用此 ARN 将此策略附加到核心设备角色。

3. 运行以下命令以将策略附加到核心设备角色。Replace*GreenGrassv2Token###Role*以及运行时指定的角色的名称AWS IoT Greengrass核心软件。然后，将策略 ARN 替换为上一步中的 ARN。

   Linux or Unix

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

   Windows Command Prompt (CMD)

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

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

PowerShell

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

如果命令没有输出，则它成功，您的核心设备可以访问您上传到此 S3 存储桶的工件。

安装程序配置资源的最低 IAM 政策

安装 AWS IoT Greengrass Core 软件时，您可以为设备预置所需的 AWS 资源， AWS IoT 例如设备和 IAM 角色。您也可以将本地开发工具部署到设备。安装程序需要 AWS 凭据才能在中执行这些操作 AWS 账户。有关更多信息，请参阅 安装 AWS IoT Greengrass Core 软件。

以下示例策略包括安装程序配置这些资源所需的最低操作集。如果您为安装程序指定`--provision`参数，则需要这些权限。将*## ID* 替换为您的 AWS 账户 ID，并将 *GreenGrassV2TokenExchangeRole* 替换为您在安装程序参数中指定的令牌交换角色的名称。`--tes-role-name`

> Note
>
> 只有在为安装程序指定`--deploy-dev-tools`参数时，才需要`DeployDevTools`策略声明。

Greengrass nucleus v2.5.0 and later

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
```

```
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:role/GreengrassV2TokenExchangeRole",
                "arn:aws:iam::account-
id:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

Earlier than v2.5.0

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:role/GreengrassV2TokenExchangeRole",
                "arn:aws:iam::account-
id:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
```

```
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
        }
    ]
}
```

Greengrass 服务角色

Greengrass 服务角色是一种 AWS Identity and Access Management (IAM) 服务角色，用于向 AWS IoT Greengrass 授权代表您访问 AWS 服务中的资源。此角色AWS IoT Greengrass使验证客户端设备的身份和管理核心设备的连接信息成为可能。

AWS IoT Greengrass V1还使用此角色来执行基本任务。有关更多信息，请参阅《开发者指南》中的 Greengrass 服务角色。AWS IoT Greengrass V1

要允许 AWS IoT Greengrass 访问您的资源，Greengrass 服务角色必须与您的 AWS 账户关联并指定 AWS IoT Greengrass 作为可信实体。该角色必须包含AWSGreengrassResourceAccessRolePolicy托管策略或自定义策略，该策略定义了您所使用的AWS IoT Greengrass功能的等效权限。 AWS维护此

策略，该策略定义了AWS IoT Greengrass用于访问您的AWS资源的权限集。有关更多信息，请参阅 AWS托管策略：AWSGreengrassResourceAccessRolePolicy。

你可以重复使用相同的 Greengrass 服务角色AWS 区域，但你必须在每个使用的地方将其与你的账户关联。AWS 区域 AWS IoT Greengrass如果当前未配置服务角色AWS 区域，则核心设备将无法验证客户端设备，也无法更新连接信息。

以下各节介绍如何使用或创建和管理 Greengrass 服务角色。AWS Management Console AWS CLI

主题

- 管理 Greengrass 服务角色（控制台）
- 管理 Greengrass 服务角色 (CLI)
- 另请参阅

> **Note**
>
> 除了授权服务级别访问的服务角色外，您还可以向 Greengrass 核心设备分配令牌交换角色。令牌交换角色是一个单独的 IAM 角色，用于控制核心设备上的 Greengrass 组件和 Lambda 函数如何访问服务。AWS有关更多信息，请参阅 授权核心设备与AWS服务。

管理 Greengrass 服务角色（控制台）

AWS IoT 控制台可让您轻松管理 Greengrass 服务角色。例如，当您为核心设备配置客户端设备发现时，控制台会检查您AWS 账户是否已连接到当前的 Greengrass 服务角色。AWS 区域如果没有，则控制台可以为您创建和配置服务角色。有关更多信息，请参阅 the section called “创建 Greengrass 服务角色”。

您可以使用 控制台执行以下角色管理任务：

主题

- 查找您的 Greengrass 服务角色（控制台）
- 创建 Greengrass 服务角色（控制台）
- 更改 Greengrass 服务角色（控制台）
- 移除 Greengrass 服务角色（控制台）

Note

登录到控制台的用户必须有权查看、创建或更改服务角色。

查找您的 Greengrass 服务角色（控制台）

使用以下步骤查找当前中AWS IoT Greengrass使用的服务角色AWS 区域。

1. 导航到 AWS IoT 控制台。
2. 在导航窗格中，选择设置。
3. 滚动到 Greengrass 服务角色部分以查看您的服务角色及其策略。

   如果您没有看到服务角色，则控制台可以为您创建或配置一个服务角色。有关更多信息，请参阅创建 Greengrass 服务角色。

创建 Greengrass 服务角色（控制台）

控制台可以为您创建和配置默认 Greengrass 服务角色。此角色具有以下属性：

属性	值
名称	`Greengrass_ServiceRole`
可信任的实体	`AWS service: greengrass`
Policy	AWSGreengrassResourceAccessRolePolicy

Note

如果您使用AWS IoT Greengrass V1设备设置脚本创建此角色，则角色名称为`GreengrassServiceRole_`*`random-string`*。

当你为核心设备配置客户端设备发现时，控制台会检查当前的 Greengrass 服务角色是否与你的关联。AWS 账户 AWS 区域如果未关联，控制台会提示您允许 AWS IoT Greengrass 代表您对 AWS 服务进行读写操作。

如果您授予权限，控制台会检查您的 AWS 账户 中是否存在名为 `Greengrass_ServiceRole` 的角色。

- 如果该角色存在，则控制台会将该服务角色附加到当前 AWS 区域 中的 AWS 账户。
- 如果该角色不存在，则控制台会创建默认 Greengrass 服务角色并将其附加到当前 AWS 区域 中的 AWS 账户。

> **Note**
>
> 如果要使用自定义角色策略创建服务角色，请使用 IAM 控制台创建或修改角色。有关更多信息，请参阅 IAM 用户指南中的创建向 AWS 服务委派权限的角色或修改角色。确保该角色针对您使用的功能和资源授予和 `AWSGreengrassResourceAccessRolePolicy` 托管策略同等的权限。我们建议您在信任策略中加入 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键，以帮助防止出现混淆代理人安全问题。条件上下文键可限制访问权限，仅允许来自指定账户和 Greengrass 工作空间的请求。有关混淆代理人问题的更多信息，请参阅 防止跨服务混淆代理。
> 如果您创建了服务角色，请返回AWS IoT控制台并将该角色附加到您的AWS 账户。你可以在 “设置” 页面上的 Greengrass 服务角色下执行此操作。

更改 Greengrass 服务角色（控制台）

使用以下过程选择其他 Greengrass 服务角色以附加到控制台中当前所选 AWS 区域 中您的 AWS 账户。

1. 导航到 AWS IoT 控制台。
2. 在导航窗格中，选择设置。
3. 在 Greengrass 服务角色下，选择 更改角色。

   更新 Greengrass 服务角色对话框打开，其中显示了您的 AWS 账户 中的哪些 IAM 角色将 AWS IoT Greengrass 定义为可信实体。
4. 选择要附加的 Greengrass 服务角色。
5. 选择附加角色。

移除 Greengrass 服务角色（控制台）

使用以下步骤将当前的 Greengrass 服务角色与您的账户分离。AWS AWS 区域这将撤销 AWS IoT Greengrass 访问当前 AWS 区域 中 AWS 服务的权限。

> **Important**
> 移除服务角色可能会中断有效操作。

1. 导航到 AWS IoT 控制台。
2. 在导航窗格中，选择设置。
3. 在 Greengrass 服务角色 下，选择 移除角色。
4. 在确认对话框中，选择 Detach (分离)。

> **Note**
> 如果您不再需要该角色，则可在 IAM 控制台中将其删除。有关更多信息，请参阅《IAM 用户指南》中的删除角色或实例配置文件。
> 其他角色可能允许 AWS IoT Greengrass 访问您的资源。要查找允许 AWS IoT Greengrass 代表您使用权限的所有角色，请在 IAM 控制台的角色页面上的可信实体列中查找包含 AWS 服务: greengrass 的角色。

管理 Greengrass 服务角色 (CLI)

在以下步骤中，我们假设AWS Command Line Interface已安装并配置为使用您的AWS 账户。有关更多信息，请参阅《AWS Command Line Interface用户指南》AWS CLI中的 "安装、更新AWS CLI和卸载" 和 "配置"。

您可以使用 AWS CLI 执行以下角色管理任务：

主题

- 获取 Greengrass 服务角色 (CLI)
- 创建 Greengrass 服务角色 (CLI)
- 删除 Greengrass 服务角色 (CLI)

获取 Greengrass 服务角色 (CLI)

使用以下过程了解 Greengrass 服务角色是否已与您在 AWS 区域 中的 AWS 账户 关联。

- 获取服务角色。将 *##* 替换为您的 AWS 区域（例如，us-west-2）。

```
aws greengrassv2 get-service-role-for-account --region region
```

  如果 Greengrass 服务角色已与您的账户关联，则请求将返回以下角色元数据。

```
{
  "associatedAt": "timestamp",
  "roleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

  如果请求未返回角色元数据，则您必须在中创建服务角色（如果不存在）并将其与您的账户关联 AWS 区域。

创建 Greengrass 服务角色 (CLI)

使用以下步骤创建角色，并将其与您的 AWS 账户 关联。

使用 IAM 创建服务角色

1. 使用允许 AWS IoT Greengrass 代入该角色的信任策略创建角色。此示例将创建一个名为 `Greengrass_ServiceRole` 的角色，但您也可以使用其他名称。我们建议您在信任策略中加入 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键，以帮助防止出现混淆代理人安全问题。条件上下文键可限制访问权限，仅允许来自指定账户和 Greengrass 工作空间的请求。有关混淆代理人问题的更多信息，请参阅 防止跨服务混淆代理。

   Linux or Unix

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
```

```
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

Windows Command Prompt (CMD)

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-
policy-document "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect
\":\"Allow\",\"Principal\":{\"Service\":\"greengrass.amazonaws.com\"},
\"Action\":\"sts:AssumeRole\",\"Condition\":{\"ArnLike\":{\"aws:SourceArn
\":\"arn:aws:greengrass:region:account-id:*\"},\"StringEquals\":
{\"aws:SourceAccount\":\"account-id\"}}}]}"
```

PowerShell

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
```

```
    }
  ]
}'
```

2. 从输出中的角色元数据复制角色 ARN。使用该 ARN 将角色与您的账户关联。
3. 将 AWSGreengrassResourceAccessRolePolicy 策略附加到该角色。

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

将服务角色与您的 AWS 账户 关联

- 将角色与您的账户关联。将 *role-arn* 替换为服务角色 ARN，并将*##*替换为您的 AWS 区域（例如，us-west-2）。

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn --
region region
```

如果成功，请求将返回以下响应。

```
{
  "associatedAt": "timestamp"
}
```

删除 Greengrass 服务角色 (CLI)

使用以下步骤解除 Greengrass 服务角色与您的 AWS 账户 的关联。

- 取消服务角色与您的账户的关联。将 *##* 替换为您的 AWS 区域（例如，us-west-2）。

```
aws greengrassv2 disassociate-service-role-from-account --region region
```

如果成功，将返回以下响应。

```
{
  "disassociatedAt": "timestamp"
}
```

Note

如果您未在任何 AWS 区域 中使用该服务角色，则应将其删除。先使用 delete-role-policy 从角色中移除 AWSGreengrassResourceAccessRolePolicy 托管策略，然后使用 delete-role 删除角色。有关更多信息，请参阅《IAM 用户指南》中的删除角色或实例配置文件。

另请参阅

- IAM 用户指南中的创建向 AWS 服务委派权限的角色。
- 《IAM 用户指南》中的修改角色
- IAM 用户指南中的删除角色或实例配置文件。
- AWS CLI 命令参考中的 AWS IoT Greengrass 命令
  - associate-service-role-to-账户
  - disassociate-service-role-from-账户
  - get-service-role-for-账户
- AWS CLI 命令参考中的 IAM 命令
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

AWS适用于 AWS IoT Greengrass 的托管策略

AWS 托管式策略是由 AWS 创建和管理的独立策略。AWS 托管式策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住，AWS 托管式策略可能不会为您的特定使用场景授予最低权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新在 AWS 托管式策略中定义的权限，则更新会影响该策略所附加到的所有主体身份（用户、组和角色）。当新的 AWS 服务启动或新的 API 操作可用于现有服务时，AWS 最有可能更新 AWS 托管式策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

主题

- AWS托管策略：AWSGreengrassFullAccess
- AWS托管策略：AWSGreengrassReadOnlyAccess
- AWS托管策略：AWSGreengrassResourceAccessRolePolicy
- 对 AWS 托管式策略的 AWS IoT Greengrass 更新

AWS托管策略：AWSGreengrassFullAccess

您可以将 `AWSGreengrassFullAccess` 策略附加得到 IAM 身份。

此策略授予管理权限，允许委托人完全访问所有内容AWS IoT Greengrass行动。

权限详细信息

此策略包含以下权限：

- `greengrass`— 允许校长完全访问所有内容AWS IoT Greengrass行动。

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

AWS托管策略：AWSGreengrassReadOnlyAccess

您可以将 `AWSGreengrassReadOnlyAccess` 策略附加得到 IAM 身份。

此策略授予只读权限，允许委托人查看但不能修改中的信息AWS IoT Greengrass。例如，拥有这些权限的主用户可以查看部署到 Greengrass 核心设备的组件列表，但不能创建部署来更改在该设备上运行的组件。

权限详细信息

此策略包含以下权限：

- greengrass— 允许校长执行返回项目列表或项目详细信息的操作。这包括以开头的 API 操作List要么Get。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:List*",
                "greengrass:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS托管策略：AWSGreengrassResourceAccessRolePolicy

你可以附上AWSGreengrassResourceAccessRolePolicy针对您的 IAM 实体的政策。AWS IoT Greengrass还将此策略附加到允许的服务角色AWS IoT Greengrass代表您执行操作。有关更多信息，请参阅Greengrass 服务角色：

此政策授予的管理权限允许AWS IoT Greengrass执行基本任务，例如检索您的 Lambda 函数、管理AWS IoT设备影子，以及验证 Greengrass 客户端设备。

权限详细信息

此策略包含以下权限。

- greengrass— 管理 Greengrass 资源。
- iot（*Shadow）— 管理AWS IoT名称中包含以下特殊标识符的阴影。这些权限是必需的，因此AWS IoT Greengrass可以与核心设备通信。
  - *-gci—AWS IoT Greengrass使用此影子存储核心设备连接信息，以便客户端设备可以发现并连接到核心设备。

  - *-gcm—AWS IoT Greengrass V1使用此影子通知核心设备 Greengrass 组的证书颁发机构 (CA) 证书已轮换。
  - *-gda—AWS IoT Greengrass V1使用此影子将部署通知核心设备。
  - GG_*— 未使用。
- iot（DescribeThing和DescribeCertificate）— 检索有关的信息AWS IoT东西和证书。这些权限是必需的，因此AWS IoT Greengrass可以验证连接到核心设备的客户端设备。有关更多信息，请参阅与本地物联网设备互动：
- lambda— 检索有关的信息AWS Lambda函数。此权限是必需的，因此AWS IoT Greengrass V1可以将 Lambda 函数部署到 Greengrass 内核。有关更多信息，请参见在上运行 Lambda 函数AWS IoT Greengrass核心在里面AWS IoT Greengrass V1开发者指南。
- secretsmanager— 检索的值AWS Secrets Manager名字开头的秘密greengrass-。此权限是必需的，因此AWS IoT Greengrass V1可以将密钥管理器机密部署到 Greengrass 内核。有关更多信息，请参见将机密部署到AWS IoT Greengrass核心在里面AWS IoT Greengrass V1开发者指南。
- s3— 从名称包含的 S3 存储桶中检索文件对象greengrass要么sagemaker。这些权限是必需的，因此AWS IoT Greengrass V1可以部署存储在 S3 存储桶中的机器学习资源。有关更多信息，请参见机器学习资源在里面AWS IoT Greengrass V1开发者指南。
- sagemaker— 检索有关亚马逊的信息SageMaker机器学习推理模型。此权限是必需的，因此AWS IoT Greengrass V1可以将机器学习模型部署到 Greengrass 内核。有关更多信息，请参见进行机器学习推断在里面AWS IoT Greengrass V1开发者指南。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGreengrassAccessToShadows",
            "Action": [
                "iot:DeleteThingShadow",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:*:*:thing/GG_*",
                "arn:aws:iot:*:*:thing/*-gcm",
                "arn:aws:iot:*:*:thing/*-gda",
                "arn:aws:iot:*:*:thing/*-gci"
            ]
```

```
        },
        {
            "Sid": "AllowGreengrassToDescribeThings",
            "Action": [
                "iot:DescribeThing"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:thing/*"
        },
        {
            "Sid": "AllowGreengrassToDescribeCertificates",
            "Action": [
                "iot:DescribeCertificate"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:cert/*"
        },
        {
            "Sid": "AllowGreengrassToCallGreengrassServices",
            "Action": [
                "greengrass:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetLambdaFunctions",
            "Action": [
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetGreengrassSecrets",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:secretsmanager:*:*:secret:greengrass-*"
        },
        {
            "Sid": "AllowGreengrassAccessToS3Objects",
```

```
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::*Greengrass*",
                "arn:aws:s3:::*GreenGrass*",
                "arn:aws:s3:::*greengrass*",
                "arn:aws:s3:::*Sagemaker*",
                "arn:aws:s3:::*SageMaker*",
                "arn:aws:s3:::*sagemaker*"
            ]
        },
        {
            "Sid": "AllowGreengrassAccessToS3BucketLocation",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassAccessToSageMakerTrainingJobs",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:sagemaker:*:*:training-job/*"
            ]
        }
    ]
}
```

对 AWS 托管式策略的 AWS IoT Greengrass 更新

您可以查看有关更新的详细信息AWS的托管策略AWS IoT Greengrass从该服务开始跟踪这些更改时起。如需有关此页面变更的自动提醒，请订阅上的 RSS feedAWS IoT Greengrass V2文档历史页面。

更改	说明	日期
AWS IoT Greengrass 已开启跟踪更改	AWS IoT Greengrass 为其 AWS 托管式策略开启了跟踪更改。	2021 年 7 月 2 日

防止跨服务混淆代理

混淆代理问题是一个安全性问题，即不具有操作执行权限的实体可能会迫使具有更高权限的实体执行该操作。在 AWS 中，跨服务模拟可能会导致混淆代理问题。一个服务（呼叫服务) 调用另一项服务（所谓的服务)时，可能会发生跨服务模拟。可以操纵调用服务，使用其权限以在其他情况下该服务不应有访问权限的方式对另一个客户的资源进行操作。为防止这种情况，AWS 提供可帮助您保护所有服务的数据的工具，而这些服务中的服务委托人有权限访问账户中的资源。

我们建议在资源策略中使用 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键，以限制 AWS IoT Greengrass 为其他服务提供的资源访问权限。如果使用两个全局条件上下文键，在同一策略语句中使用时，`aws:SourceAccount` 值和 `aws:SourceArn` 值中的账户必须使用相同的账户 ID。

的值`aws:SourceArn`Greengrass 是与`sts:AssumeRole`请求.

防范混淆代理问题最有效的方法是使用 `aws:SourceArn` 全局条件上下文键和资源的完整 ARN。如果不知道资源的完整 ARN，或者正在指定多个资源，请针对 ARN 未知部分使用带有通配符 (*) 的 `aws:SourceArn` 全局上下文条件键。例如，`arn:aws:greengrass::account-id:*`。

有关使用`aws:SourceArn`和`aws:SourceAccount`全局条件上下文键，请参见创建 Greengrass 服务角色.

排查 AWS IoT Greengrass 的身份和访问权限问题

使用以下信息可帮助您诊断和修复在使用 AWS IoT Greengrass 和 IAM 时可能遇到的常见问题。

问题

- 我无权在 AWS IoT Greengrass 中执行操作
- 我无权执行 iam:PassRole
- 我是管理员并希望允许其他人访问 AWS IoT Greengrass
- 我希望允许我的 AWS 账户以外的人访问我的 AWS IoT Greengrass 资源

有关一般故障排除帮助，请参见故障排除.

我无权在 AWS IoT Greengrass 中执行操作

如果您收到错误消息，提示您无权执行操作，必须联系您的管理员寻求帮助。您的管理员是指为您提供用户名和密码的那个人。

以下示例错误发生在mateojacksonIAM 用户尝试查看有关核心设备的详细信息，但没有greengrass:GetCoreDevice权限。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: greengrass:GetCoreDevice on resource: arn:aws:greengrass:us-
west-2:123456789012:coreDevices/MyGreengrassCore
```

在这种情况下，Mateo 请求他的管理员更新其策略，以允许他使用 greengrass:GetCoreDevice 操作访问 arn:aws:greengrass:us-west-2:123456789012:coreDevices/MyGreengrassCore 资源。

以下是在使用时可能遇到的一般 IAM 问题AWS IoT Greengrass.

我无权执行 iam:PassRole

如果您收到一个错误，表明您无权执行 iam:PassRole 操作，则必须更新策略以允许您将角色传递给 AWS IoT Greengrass。

有些 AWS 服务允许您将现有角色传递到该服务，而不是创建新服务角色或服务相关角色。为此，您必须具有将角色传递到服务的权限。

当名为 marymajor 的 IAM 用户尝试使用控制台在 AWS IoT Greengrass 中执行操作时，会发生以下示例错误。但是，服务必须具有服务角色所授予的权限才可执行此操作。Mary 不具有将角色传递到服务的权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

在这种情况下，必须更新 Mary 的策略以允许她执行 iam:PassRole 操作。

如果您需要帮助，请联系您的 AWS 管理员。管理员是向您提供登录凭证的人。

我是管理员并希望允许其他人访问 AWS IoT Greengrass

要允许其他人访问 AWS IoT Greengrass，您必须为需要访问权限的人员或应用程序创建一个 IAM 实体（用户或角色）。它们将使用该实体的凭证访问 AWS。然后，您必须将策略附加到实体，以便在 AWS IoT Greengrass 中向其授予正确的权限。

要立即开始使用，请参阅 IAM 用户指南中的创建您的第一个 IAM 委派用户和组。

我希望允许我的 AWS 账户以外的人访问我的 AWS IoT Greengrass 资源

您可以创建一个 IAM 角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的AWS 资源。您可以指定谁值得信赖，可以带入角色。有关更多信息，请参阅在另一个中向 IAM 用户提供访问权限AWS 账户您拥有的和向向向访问提供访问权限AWS 账户由第三方拥有在里面IAM 用户指南.

AWS IoT Greengrass 不支持根据基于资源的策略或访问控制列表 (ACL) 跨账户访问。

允许设备流量通过代理或防火墙

Greengrass 核心设备和 Greengrass 组件执行对服务和其他网站的出站请求。AWS作为一项安全措施，您可以将出站流量限制在较小的端点和端口范围内。您可以使用以下有关终端节点和端口的信息来限制通过代理、防火墙或 Amazon VPC 安全组的设备流量。有关如何将核心设备配置为使用代理的更多信息，请参阅通过端口 443 或网络代理进行连接。

主题

- 基本操作的终端节点
- 使用自动配置进行安装的终端节点
- AWS提供的组件的端点

基本操作的终端节点

Greengrass 核心设备使用以下端点和端口进行基本操作。

检索AWS IoT端点

获取您的终AWS IoT端节点AWS 账户，然后将其保存以备后用。您的设备使用这些端点进行连接AWS IoT。执行以下操作：

1. 获取您的AWS IoT数据端点AWS 账户。

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

如果请求成功，则响应类似于以下示例。

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. 获取您的AWS IoT凭证端点AWS 账户。

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

如果请求成功，则响应类似于以下示例。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

Endpoint	端口	必需	描述
`greengrass-ats.iot. region.amazonaws.com`	8443 或 443	是	用于数据平面操作，例如安装部署和使用客户端设备。
`device-data-prefix -ats.iot. region.amazonaws.com`	MQTT：8883 或 443 HTTPS：8443 或 443	是	用于设备管理的数据平面操作，例如 MQTT 通信和与的影子同 AWS IoT Core步。

Endpoint	端口	必需	描述
device-credentials-prefix .credentials.iot. *region*.amazonaws.com	443	是	用于获取AWS证书，核心设备使用这些证书从Amazon S3 下载组件工件并执行其他操作。有关更多信息，请参阅 授权核心设备与AWS服务。
*.s3.amazonaws.com *.s3.*region*.amazonaws.com	443	是	用于部署。此格式包括*字符，因为端点前缀由内部控制，并且可能随时更改。

Endpoint	端口	必需	描述
data.iot. *region*.amazonaws.com	443	否	如果核心设备运行的版本早于 v2.4.0 的 Greengrass 核心，并且配置为使用网络代理，则为必填项。在代理服务器后面，核心设备使用此端点与 AWS IoT Core MQTT 通信。有关更多信息，请参阅 配置网络代理。

使用自动配置进行安装的终端节点

当您安装AWS IoT Greengrass具有自动资源配置功能的核心软件时，Greengrass 核心设备使用以下端点和端口。

Endpoint	端口	必需	描述
iot.*region*.amazonaws.com	443	是	用于创建AWS IoT资源和检索有关现有AWS IoT资源的信息。

Endpoint	端口	必需	描述
`iam.amazonaws.com`	443	是	用于创建 IAM 资源和检索有关现有 IAM 资源的信息。
`sts.`*`region`*`.amazonaws.com`	443	是	用来获取你的身份证AWS 账户。
`greengrass.` *`region`*`.amazonaws.com`	443	否	如果您使用`--deploy-dev-tools` 参数将 Greengrass CLI 组件部署到核心设备，则为必填项。

AWS提供的组件的端点

Greengrass 核心设备使用其他端点，具体取决于它们运行的软件组件。您可以在本开发者指南中每个组件页面的 “需求” 部分中找到每个AWS提供的组件所需的端点。有关更多信息，请参阅 AWS-提供的组件。

AWS IoT Greengrass 的合规性验证

要了解某个 AWS 服务 是否在特定合规性计划范围内，请参阅合规性计划范围内的 AWS 服务，然后选择您感兴趣的合规性计划。有关常规信息，请参阅AWS 合规性计划。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅在 AWS Artifact 中下载报告、。

您使用 AWS 服务 的合规性责任取决于数据的敏感性、贵公司的合规性目标以及适用的法律法规。AWS 提供以下资源来帮助满足合规性：

- 安全性与合规性快速入门指南 - 这些部署指南讨论了架构注意事项，并提供了在 AWS 上部署以安全性和合规性为重点的基准环境的步骤。
- Amazon Web Services 上的 HIPAA 安全性和合规性架构设计 - 该白皮书介绍了公司如何使用 AWS 创建符合 HIPAA 标准的应用程序。

> **Note**
>
> 并非所有 AWS 服务 都符合 HIPAA 要求。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS 合规性资源 - 此业务手册和指南集合可能适用于您的行业和位置。
- AWS 客户合规指南：从合规角度了解责任共担模式。这些指南总结了保护 AWS 服务 的最佳实践，并将指南映射到跨多个框架的安全控制，包括美国国家标准与技术研究院（NIST）、支付卡行业安全标准委员会（PCI）和国际标准化组织（ISO）。
- AWS Config 开发人员指南中的使用规则评估资源 - 此 AWS Config 服务评测您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS Security Hub - 此 AWS 服务 向您提供 AWS 中安全状态的全面视图。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实操。有关受支持服务及控制的列表，请参阅 Security Hub 控制参考。
- AWS Audit Manager – 此 AWS 服务 可帮助您持续审计您的 AWS 使用情况，以简化管理风险以及与相关法规和行业标准的合规性的方式。

AWS IoT Greengrass 中的故障恢复能力

这些区域有：AWS全球基础设施围绕 Amazon Web Services 区域和可用区构建。EANAWS 区域提供多个在物理上独立且隔离的可用区，这些可用区与延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

如需更多信息，请参阅 AWS 全球基础设施。

除了 AWS 全球基础设施之外，AWS IoT Greengrass 还提供了多种功能，以帮助支持您的数据弹性和备份需求。

- 您可以将 Greengrass 核心设备配置为将日志写入本地文件系统和 CloudWatch 日志。如果核心设备丢失连接，它可以继续在文件系统上记录消息。当它重新连接时，它会将日志消息写入 CloudWatch 日志。有关更多信息，请参阅 监控AWS IoT Greengrass日志。
- 如果核心设备在部署过程中断电源，则在AWS IoT Greengrass核心软件再次启动。
- 如果核心设备丢失互联网连接，Greengrass 客户端设备可通过本地网络继续进行通信。
- 你可以创作可以读取的 Greengrass 组件流管理器流式传输数据并将数据发送到本地存储目标。

AWS IoT Greengrass 中的基础设施安全性

作为一项托管式服务，AWS IoT Greengrass 由 Amazon Web Services：安全流程概览白皮书中所述的 AWS 全球网络安全程序提供保护。

您可以使用 AWS 发布的 API 调用通过网络访问 AWS IoT Greengrass。客户端必须支持传输层安全性 (TLS) 1.2 或更高版本。建议使用 TLS 1.3 或更高版本。客户端还必须支持具有完全向前保密 (PFS) 的密码套件，例如 Ephemeral Diffie-Hellman (DHE) 或 Elliptic Curve Ephemeral Diffie-Hellman (ECDHE)。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

必须使用访问密钥 ID 以及与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service (AWS STS) 生成临时安全凭证来对请求进行签名。

在AWS IoT Greengrass环境中，设备使用 X.509 证书和加密密钥来连接和进行身份验证。AWS Cloud 有关更多信息，请参阅 the section called “设备身份验证和授权”。

AWS IoT Greengrass 中的配置和漏洞分析

IoT 环境可能由大量具有不同功能、长期存在且地理位置分散的设备组成。这些特性导致设备设置复杂且容易出错。由于设备的计算能力、内存和存储功能通常有限，因而限制了在设备本身上对加密和其它形式的安全功能的使用。此外，设备经常使用具有已知漏洞的软件。这些因素不仅令 IoT 设备成为吸引黑客的目标，而且导致难以持续保护设备安全。

AWS IoT Device Defender 提供了识别安全问题和最佳实践偏离情况的工具，从而解决了这些难题。您可以使用 AWS IoT Device Defender 分析、审计和监控连接设备，以检测异常行为并降低安全风险。AWS IoT Device Defender 可以审计设备，以确保其遵守安全最佳实践，并检测设备上的异常行

为。这使您能够跨多个设备实施一致的安全策略，并且能够在设备遭到破坏时快速响应。有关更多信息，请参阅以下主题：

- 这些区域有：Device Defender 组件
- AWS IoT Device Defender（在 AWS IoT Core 开发人员指南中）。

在 AWS IoT Greengrass 环境中，您应注意以下事项：

- 您有责任保护物理设备、设备上的文件系统和本地网络的安全。
- AWS IoT Greengrass不会对用户定义的 Greengrass 组件强制实施网络隔离，无论它们是否在 Greengrass 容器中运行。因此，Greengrass 组件可以通过网络与系统内外运行的任何其他进程进行通信。

中的代码完整性AWS IoT Greengrass V2

AWS IoT Greengrass从中部署软件组件AWS Cloud转到运行AWS IoT GreengrassCore 软件。这些软件组件包括AWS-提供的组件和自定义组件你上传到你的AWS 账户. 每个组件都由一个配方组成。配方定义了组件的元数据和任意数量的工件，它们是组件二进制文件，例如编译的代码和静态资源。Amazon S3 中存储组件。

在开发和部署 Greengrass 组件时，您可以按照以下基本步骤操作，这些基本步骤可以在AWS 账户在你的设备上：

1. 创建工件并将其上传到 S3 存储桶。
2. 从中的配方和工件中创建组件AWS IoT Greengrass服务，它计算加密哈希每个神器。
3. 将组件部署到 Greengrass 核心设备，这些设备下载并验证每个工件的完整性。

AWS负责在将工件上传到 S3 存储桶后维护工件的完整性，包括在将组件部署到 Greengrass 核心设备时。在将工件上传到 S3 存储桶之前，您有责任保护软件工件。您还有责任确保对您中的资源的访问权限AWS 账户，包括您在其中上传组件工件的 S3 存储桶。

> **Note**
>
> Amazon S3 提供了一项名为 S3 对象锁定的功能，您可以使用该功能防止 S3 存储桶中的组件工件发生更改AWS 账户. 您可以使用 S3 对象锁定在删除或覆盖组件项目。有关更多信息，请参阅 。使用 S3 对象锁定中的Amazon Simple Storage Service 用户指南.

何时AWS发布公共组件，当你上传自定义组件时，AWS IoT Greengrass计算每个组件工件的加密摘要。AWS IoT Greengrass更新组件配方以包括每个工件的摘要和用于计算该摘要的哈希算法。本摘要保证了工件的完整性，因为如果工件在AWS Cloud或者在下载期间，它的文件摘要不匹配AWS IoT Greengrass存储在组件配方中。有关更多信息，请参阅 。组件配方参考中的工件.

将组件部署到核心设备时，AWS IoT Greengrass核心软件下载配方定义的组件配方和每个组件工件。这些区域有：AWS IoT Greengrass核心软件计算每个下载的工件文件的摘要，并将其与配方中的工件摘要进行比较。如果摘要不匹配，则部署会失败，并AWS IoT Greengrass核心软件从设备的文件系统中删除下载的工件。有关如何连接核心设备和AWS IoT Greengrass是安全的，请参阅传输中加密.

您有责任保护核心设备文件系统上的组件工件文件。这些区域有：AWS IoT Greengrass核心软件将工件保存到packagesGreengrass 根文件夹中的文件夹。您可以使用AWS IoT Device Defender以分析、审计和监控核心设备。有关更多信息，请参阅AWS IoT Greengrass 中的配置和漏洞分析。

AWS IoT Greengrass 和接口 VPC 端点 (AWS PrivateLink)

您可以通过创建接口 VPC 端点在 VPC 和 AWS IoT Greengrass 控制面板之间建立私有连接。您可以使用此端点管理AWS IoT Greengrass服务中的组件、部署和核心设备。接口端点由 AWS PrivateLink 提供支持，该技术支持您通过私密方式访问 AWS IoT Greengrass API，而无需互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。VPC 中的实例即使没有公有 IP 地址也可与 AWS IoT Greengrass API 进行通信。VPC 和 AWS IoT Greengrass 之间的流量不会脱离 Amazon 网络。

每个接口端点均由子网中的一个或多个弹性网络接口表示。

有关更多信息，请参阅 Amazon VPC 用户指南中的接口 VPC 端点 (AWS PrivateLink)。

主题

- AWS IoT Greengrass VPC 端点注意事项
- 为 AWS IoT Greengrass 控制平面操作创建接口 VPC 端点
- 为 AWS IoT Greengrass 创建 VPC 端点策略
- 在 VPC 中操作AWS IoT Greengrass核心设备

AWS IoT Greengrass VPC 端点注意事项

请先查看 Amazon VPC 用户指南中的接口端点属性和限制，然后再为 AWS IoT Greengrass 设置 VPC 接口端点。此外，请了解以下注意事项：

- AWS IoT Greengrass 支持从 VPC 调用它的所有控制面板 API 操作。控制平面包括诸如CreateDeployment和之类的操作ListEffectiveDeployments。控制平面不包括诸如ResolveComponentCandidates和 D iscover 之类的操作，它们是数据平面操作。
- AWS IoT Greengrass 的 VPC 端点目前在 AWS 中国区域不受支持。

为 AWS IoT Greengrass 控制平面操作创建接口 VPC 端点

您可以使用 Amazon VPC 控制台或AWS Command Line Interface（AWS CLI）为 AWS IoT Greengrass 控制面板创建 VPC 端点。有关更多信息，请参阅《Amazon VPC 用户指南》中的创建接口端点

使用以下服务名称为 AWS IoT Greengrass 创建 VPC 端点：

- com.amazonaws.*region*.greengrass

如果为端点启用私有 DNS，则可以使用其默认 DNS 名称作为区域，向 AWS IoT Greengrass 发送 API 请求，例如 `greengrass.us-east-1.amazonaws.com`。默认情况下将启用私有 DNS。

有关更多信息，请参阅《Amazon VPC 用户指南》中的通过接口端点访问服务。

为 AWS IoT Greengrass 创建 VPC 端点策略

您可以为 VPC 端点附加控制对 AWS IoT Greengrass 控制面板操作的访问的端点策略。该策略指定以下信息：

- 可执行操作的主体。
- 主体可以执行的操作。
- 主体可以对其执行操作的资源。

有关更多信息，请参阅《Amazon VPC 用户指南》中的使用 VPC 端点控制对服务的访问权限。

Example 示例：AWS IoT Greengrass 操作的 VPC 端点策略

下面是用于 AWS IoT Greengrass 的端点策略示例。当附加到端点时，此策略会向所有资源上的所有主体授予对列出的 AWS IoT Greengrass 操作的访问权限。

```
{
    "Statement": [
        {
```

```
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "greengrass:ListEffectiveDeployments"
            ],
            "Resource": "*"
        }
    ]
}
```

在 VPC 中操作AWS IoT Greengrass核心设备

无需公共互联网访问即可操作 Greengrass 核心设备并在 VPC 中执行部署。您必须至少使用相应的 DNS 别名设置以下 VPC 终端节点。有关如何创建和使用 VPC 终端节点的更多信息，请参阅 Amazon VPC 用户指南中的创建 VPC 终端节点。

> **Note**
>
> 对于和AWS IoT凭证，用于自动创建 DNS 记录的 VPC 功能已禁用。AWS IoT data要连接这些终端节点，必须手动创建私有 DNS 记录。有关更多信息，请参阅接口终端节点的私有 DNS。有关 AWS IoT Core VPC 限制的更多信息，请参阅 V PC 终端节点的限制。

先决条件

- 您必须使用手动配置步骤安装 C AWS IoT Greengrass ore 软件。有关更多信息，请参阅 使用手动资源配置来安装 AWS IoT Greengrass Core 软件。

限制

- 中国地区不支持在 VPC 中运行 Greengrass 核心设备，以及。AWS GovCloud (US) Regions
- 有关限制AWS IoT data和AWS IoT凭证提供者 VPC 终端节点的更多信息，请参阅限制。

将您的 Greengrass 核心设备设置为在 VPC 中运行

1. 获取您的终AWS IoT端节点AWS 账户，然后将其保存以备后用。您的设备使用这些端点进行连接 AWS IoT。执行以下操作：

a. 获取您的AWS IoT数据端点AWS 账户。

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

如果请求成功，则响应类似于以下示例。

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

b. 获取您的AWS IoT凭证端点AWS 账户。

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

如果请求成功，则响应类似于以下示例。

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

2. 为AWS IoT data和AWS IoT证书终端节点创建 Amazon VPC 接口：

a. 导航到 VPC 端点控制台，在左侧菜单的虚拟私有云下，选择端点，然后选择创建端点。

b. 在创建端点页面上，指定以下信息。

- 为 Service category（服务类别）选择AWS 服务。
- 对于 Service Name（服务名称），通过输入关键字 `iot` 进行搜索。在显示的 `iot` 服务列表中，请选择端点。

  如果您为 AWS IoT Core 数据平面创建 VPC 端点，请选择您所在区域的 AWS IoT Core 数据平面 API 端点。终端节点的格式为 `com.amazonaws.region.iot.data`。

  如果您为 AWS IoT Core 凭证提供商创建 VPC 端点，请选择您所在区域的 AWS IoT Core 凭证提供商端点。终端节点的格式为 `com.amazonaws.region.iot.credentials`。

> **Note**
>
> 中国区域的 AWS IoT Core 数据平面的服务名称将采用 `cn.com.amazonaws.region.iot.data` 格式。中国区域不支持为 AWS IoT Core 凭证提供商创建 VPC 端点。

- 对于 VPC 和 Subnets（子网），选择要在其中创建端点的 VPC 和要在其中创建端点网络的可用区 (AZ)。
- 对于 Enable DNS name（启用 DNS 名称），请确保未选择 Enable for this endpoint（为此终端节点启用）。AWS IoT Core 数据平面和 AWS IoT Core 凭证提供商都不支持私有 DNS 名称。
- 对于 Security group（安全组），选择要与端点网络接口关联的安全组。
- 您可以选择添加或删除标签。标签是用于与端点关联的名称-值对。

c. 要创建 VPC 终端节点，请选择 Create endpoint（创建端点）。

3. 创建 AWS PrivateLink 端点后，在端点的详细信息选项卡下，您将看到 DNS 名称列表。您可以使用在本部分中创建的这些 DNS 名称中的一个来配置私有托管区域。
4. 创建 Amazon S3 终端节点。有关更多信息，请参阅为 Amazon S3 创建 VPC 终端节点。
5. 如果您使用的是AWS提供的 Greengrass 组件，则可能需要其他端点和配置。要查看端点要求，请从AWS提供的组件列表中选择组件，然后查看 “需求” 部分。例如，日志管理器组件要求要求此组件必须能够执行对端点的出站请求`logs.region.amazonaws.com`。

   如果您使用的是自己的组件，则可能需要查看依赖关系并进行其他测试，以确定是否需要任何其他端点。
6. 在 Greengrass 核配置中，必须设置为。`greengrassDataPlaneEndpoint` **`iotdata`**有关更多信息，请参阅 Greengrass 核配置。
7. 如果您`us-east-1`在该区域，请在 Greengrass nucleus 配置**`REGIONAL`**中将配置参数`s3EndpointType`设置为。此功能适用于 Greengrass nucleus 版本 2.11.3 或更高版本。

Example 示例：组件配置

```
{
"aws.greengrass.Nucleus": {
   "configuration": {
      "awsRegion": "us-east-1",
      "iotCredEndpoint": "xxxxxx.credentials.iot.region.amazonaws.com",
```

```
      "iotDataEndpoint": "xxxxxx-ats.iot.region.amazonaws.com",
      "greengrassDataPlaneEndpoint": "iotdata",
      "s3EndpointType": "REGIONAL"
      ...
     }
   }
}
```

下表提供了有关相应的自定义私有 DNS 别名的信息。

服务	VPC 端点服务名称	VPC 终端节点类型	自定义私有 DNS 别名	注意事项
AWS IoT data	com.amazonaws. *region*.iot.c	接口	*prefix*-ats.iot. *region*. s.com	私有 DNS 记录应与您账户的AWS IoT data终端节点相匹配：aws iot describe-endpoint --endpoint-type iot:Data-ATS 。
AWS IoT 凭证	com.amazonaws. *region*.iot.credentials	接口	*prefix*.cr als.iot. s.com	私有 DNS 记录应与您的账户AWS IoT凭证端点匹配：aws

服务	VPC 端点服务名称	VPC 终端节点类型	自定义私有 DNS 别名	注意事项
				iot describe-endpoint --endpoint-type iot:CredentialProvider 。
Amazon S3	com.amazonaws. *region*.s3	接口		DNS 记录是自动创建的。

AWS IoT Greengrass 的安全最佳实践

本主题包含 AWS IoT Greengrass 的安全最佳实践。

授予可能的最低权限

以非特权用户身份运行组件，遵循最低权限原则。除非绝对必要，否则组件不应以 root 身份运行。

在 IAM 角色中使用最低权限集。限制使用*的通配符Action和Resource您的 IAM 策略中的属性。而是在可能的情况下声明一组有限的操作和资源。有关最低权限和其他策略最佳实践的更多信息，请参阅the section called “策略最佳实践”。

最低权限最佳做法也适用于AWS IoT你附加到你的 Greengrass 核心的政策。

不要在 Greengrass 组件中对凭据进行硬编码

不要在用户定义的 Greengrass 组件中对凭据进行硬编码。为了更好地保护您的凭证：

- 与之互动AWS服务，在中定义特定操作和资源的权限Greengrass 核心设备服务角色。
- 使用秘密管理器组件来存储您的凭证。或者，如果该函数使用AWSSDK，使用来自默认凭证提供商链的凭证。

不要记录敏感信息

您应该禁止记录凭证和其他个人身份信息 (PII)。即使访问核心设备上的本地日志需要 root 权限和访问权限，我们也建议您实施以下安全措施 CloudWatch 日志需要 IAM 权限。

- 不要在 MQTT 主题路径中使用敏感信息。
- 不要在 AWS IoT Core 注册表中的设备（事物）名称、类型和属性中使用敏感信息。
- 不要在用户定义的 Greengrass 组件或 Lambda 函数中记录敏感信息。
- 不要在 Greengrass 资源的名称和 ID 中使用敏感信息：
  - 核心设备
  - 组件
  - 部署
  - 日志记录程序

使设备时钟保持同步

请务必确保您的设备上有准确的时间。X.509 证书具有到期日期和时间。设备上的时钟用于验证服务器证书是否仍有效。设备时钟可能会在一段时间后出现偏差，或者电池可能会放电。

有关更多信息，请参阅保持设备的时钟同步中的最佳实践AWS IoT Core开发者指南。

密码套件推荐

Greengrass 默认选择设备上可用的最新 TLS 密码套件。考虑在设备上禁用传统密码套件的使用。例如，CBC 密码套件。

有关更多信息，请参阅Java 密码学配置。

另请参阅

- 中的安全最佳实践AWS IoT Core在AWS IoT开发者指南
- 工业物联网解决方案的十大安全黄金法则在开启物联网AWS官方博客

用AWS IoT Device Tester于 AWS IoT Greengrass V2

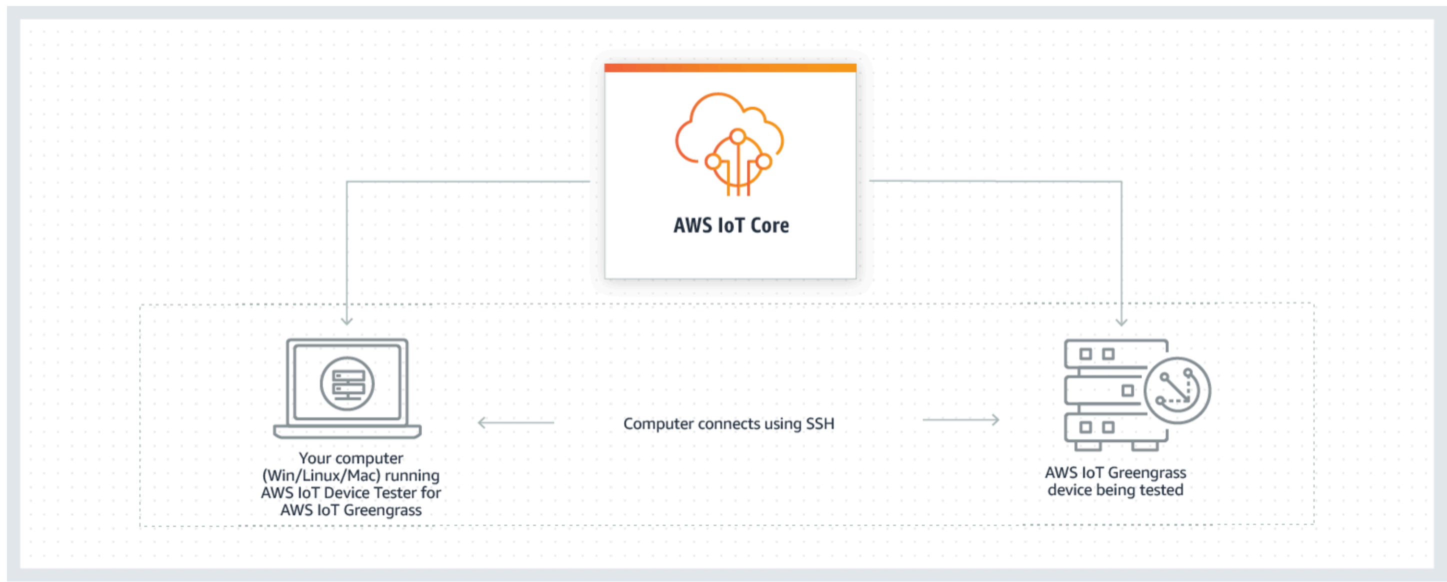
AWS IoT Device Tester (IDT) 是一个可下载的测试框架，可供您用于验证 IoT 设备。您可以使用 IDT AWS IoT Greengrass 来运行AWS IoT Greengrass资格套件，并为您的设备创建和运行自定义测试套件。

适用于 AWS IoT Greengrass 的 IDT 在连接到要测试的设备的主机（Windows、macOS 或 Linux）上运行。它运行测试并聚合结果。它还提供命令行界面来管理测试过程。

AWS IoT Greengrass 资格认证套件

用AWS IoT Device Tester于 AWS IoT Greengrass V2 来验证 C AWS IoT Greengrass ore 软件是否在您的硬件上运行并且可以与通信。AWS Cloud它还使用执行 end-to-end 测试AWS IoT Core。例如，它会验证您的设备是否可以部署组件并对其进行升级。

如果要将硬件添加到 AWS Partner 设备目录中，请运行 AWS IoT Greengrass 资格套件以生成可以提交到 AWS IoT 的测试报告。有关更多信息，请参阅 AWS 设备资格认证计划。



IDT fo AWS IoT Greengrass r V2 使用测试套件和测试组的概念组织测试。

- 测试套件是一组测试组，用于验证设备运行的是否为特定版本的 AWS IoT Greengrass。
- 测试组是与特定功能（例如组件部署）相关的一组单独测试。

有关更多信息，请参阅 使用 IDT 运行 AWS IoT Greengrass 资格套件。

自定义测试套件

从 IDT v4.0.1 开始，IDT for AWS IoT Greengrass V2 将标准化配置设置和结果格式与测试套件环境相结合，使您能够为设备和设备软件开发自定义测试套件。您可以添加自定义测试来用于自己的内部验证，也可以将其提供给客户进行设备验证。

测试编写者如何配置自定义测试套件决定了运行自定义测试套件需要的设置配置。有关更多信息，请参阅 使用 IDT 开发和运行自己的测试套件。

支持的 f AWS IoT Device Tester or AWS IoT Greengrass V2 版本

本主题列出了 AWS IoT Greengrass V2 版 IDT 支持的版本。作为最佳实践，我们建议您使用支持目标版本 AWS IoT Greengrass V2 的最新版本 IDT for V2。 AWS IoT Greengrass 的新版本 AWS IoT Greengrass 可能需要您下载适用于 AWS IoT Greengrass V2 的 IDT 的新版本。如果 IDT for AWS IoT Greengrass V2 与您正在使用的版本不兼容，则在开始测试运行时 AWS IoT Greengrass 会收到通知。

下载软件即表示您同意AWS IoT Device Tester 许可协议。

Note

IDT 不支持由多个用户从共享位置（如 NFS 目录或 Windows 网络共享文件夹）运行。建议您将 IDT 包解压缩到本地驱动器，并在本地工作站上运行 IDT 二进制文件。

适用 AWS IoT Greengrass 于 V2 的最新 IDT 版本

您可以将此版本的 IDT for AWS IoT Greengrass V2 与此处列出的 AWS IoT Greengrass 版本一起使用。

IDT v4.9.3 适用于 AWS IoT Greengrass

支持的 AWS IoT Greengrass 版本：

- Greengrass nucleus v2.12.0、v2.11. 0、v2.10.0 和 v2.9.5

IDT 软件下载：

- 带有适用于 Linux 的测试套件 GGV2Q_2.5.3 的 IDT v4.9.3
- 带测试套件 GGV2Q_2.5.3 的 IDT v4.9.3 适用于 macOS

- 带有适用于 Windows 的测试套件 GGV2Q_2.5.3 的 IDT v4.9.3

发行说明：

- 修复了从 Windows 主机测试 Linux 设备时组件测试中的一个问题，反之亦然。
- 从`localcomponent`测试组中移除`component`测试用例。此测试用例不再是资格认证所必需的。

附加注释：

- 如果你的设备使用 HSM，而你使用的是 nucleus 2.10.x，请迁移到 Greengrass nucleus 版本 2.11.0 或更高版本。

测试套件版本：

`GGV2Q_2.5.3`

- 2024.04.05 发布

不支持的 for V2 AWS IoT Device Tester 版本 AWS IoT Greengrass

本主题列出了不支持的 V2 版 IDT 版本。 AWS IoT Greengrass 不受支持的版本不会收到错误修复或更新。有关更多信息，请参阅 the section called “的 Support AWS IoT Device Tester 政策 AWS IoT Greengrass”。

IDT v4.9.2 适用于 AWS IoT Greengrass

发行说明：

- 修复了由于 Java 8 被弃用而导致 Lambda 测试套件失败的问题。

测试套件版本：

`GGV2Q_2.5.2`

- 2024.03.18 发布

IDT v4.9.1 适用于 AWS IoT Greengrass

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 酷睿软件版本 2.12.0、2.11.0、2.10.0 和 2.9.5 的设备。
- 次要错误修复。

测试套件版本：

`GGV2Q_2.5.1`

- 2023.10.05 发布

IDT v4.7.0 适用于 AWS IoT Greengrass

支持的 AWS IoT Greengrass 版本：

- Greengrass nucleus v2.11.0、v2.10.0 和 v2.9.5

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 酷睿软件版本 2.11.0、2.10.0 和 2.9.5 的设备。
- 添加了对在 Parameter Stor AWS Systems Manager e 中存储 IDT 用户数据值的支持，并使用占位符语法将其提取到配置中。
- 次要错误修复。

测试套件版本：

`GGV2Q_2.5.0`

- 2022.12.13 发布

IDT v4.5.11 适用于 AWS IoT Greengrass

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 酷睿软件版本 2.9.1、2.9.0、2.8.1、2.8.0、2.7.0 和 2.6.0 的设备。
- 增加了对在核心设备上测试 PreInstalled Greengrass 的支持。
- 次要错误修复。

测试套件版本：

`GGV2Q_2.4.1`

- 2022.10.13 发布

IDT v4.5.8 适用于 AWS IoT Greengrass

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 酷睿软件版本 2.7.0、2.6.0 和 2.5.6 的设备。
- 使您能够在核心设备上使用 PreInstalled Greengrass 进行测试。
- 次要错误修复。

测试套件版本：

`GGV2Q_2.4.0`

- 2022.08.12 已发布

IDT v4.5.3 适用于 AWS IoT Greengrass

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 酷睿软件版本 2.7.0、2.6.0、2.5.6、2.5.5、2.5.4 和 2.5.3 的设备。
- 更新 DockerApplicationManager 测试以使用基于 ECR 的 docker 镜像。
- 次要错误修复。

测试套件版本：

`GGV2Q_2.3.1`

- 2022.04.15 已发布

IDT v4.5.1 适用于 AWS IoT Greengrass

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 核心软件 v2.5.3 的设备。
- 增加了对使用硬件安全模块 (HSM) 存储酷睿软件使用的私钥和证书的基于 Linux 的设备的验证和鉴定支持。 AWS IoT Greengrass
- 实现用于配置自定义测试套件的新 IDT 测试编排工具。有关更多信息，请参阅 配置 IDT 测试管弦乐队。
- 修复了其他小错误。

测试套件版本：

`GGV2Q_2.3.0`

- 2022.01.11 已发布

IDT v4.4.1 适用于 AWS IoT Greengrass

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 酷睿软件 v2.5.2 的设备。
- 增加了对使用用户定义的 IAM 角色作为被测设备假设的与 AWS 资源交互的令牌交换角色的支持。

  您可以在`userdata.json`文件中指定 IAM 角色。如果您指定自定义角色，IDT 将在测试运行期间使用该角色而不是创建默认的令牌交换角色。
- 修复了其他小错误。

测试套件版本：

`GGV2Q_2.2.1`

- 2021.12.12 发布

IDT v4.4.0 适用于 AWS IoT Greengrass

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 酷睿软件 v2.5.0 的设备。
- 增加了对在 Windows 上运行 AWS IoT Greengrass 酷睿软件的设备进行验证和认证的支持。
- 支持使用公钥验证进行安全 shell (SSH) 设备连接。
- 利用安全最佳实践改进 IDT 权限 IAM 策略。
- 修复了其他小错误。

测试套件版本：

`GGV2Q_2.1.0`

- 2021.11.19 发布

IDT v4.2.0 适用于 AWS IoT Greengrass

发行说明：

- 包括支持在运行 C AWS IoT Greengrass ore 软件 v2.2.0 及更高版本的设备上认证以下功能：
  - Docker — 验证设备是否可以从亚马逊弹性容器注册表 (Amazon ECR) Container Registry 下载 Docker 容器镜像。
  - [机器学习-验证设备是否可以使用深度学习运行时或 Lite ML 框架执行机器学习 (ML) 推理。TensorFlow]
  - 流管理器-验证设备是否可以下载、安装和运行流管理器。 AWS IoT Greengrass
- 使您能够验证和鉴定运行 AWS IoT Greengrass 酷睿软件 v2.4.0、v2.3.0、v2.0 和 v2.1.0 的设备。
- 将每个测试用例的测试日志分组到*<device-tester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>*目录内单独的 *< test-case-id >* 文件夹中。
- 修复了其他小错误。

测试套件版本：

`GGV2Q_2.0.1`

- 2021.08.31 发布

IDT v4.1.0 适用于 AWS IoT Greengrass

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 酷睿软件 v2.3.0、v2.2.0、v2.1.0 和 v2.0.5 的设备。

- 删除了指定`GreengrassNucleusVersion`和`GreengrassCLIVersion`属性的要求，从而改进了`userdata.json`配置。
- 包括对 AWS IoT Greengrass 核心软件 v2.1.0 及更高版本的 Lambda 和 MQTT 功能认证的支持。现在，您可以使用适用于 AWS IoT Greengrass V2 的 IDT 来验证您的核心设备是否可以运行 Lambda 函数，以及该设备是否可以发布和订阅 MQTT 主题。 AWS IoT Core
- 提高了日志记录功能。
- 修复了其他小错误。

测试套件版本：

`GGV2Q_1.1.1`

- 2021.06.18 已发布

IDT v4.0.2 适用于 AWS IoT Greengrass

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 核心软件 v2.1.0 的设备。
- 增加了对 AWS IoT Greengrass 核心软件 v2.1.0 及更高版本的 Lambda 和 MQTT 功能认证的支持。现在，您可以使用适用于 AWS IoT Greengrass V2 的 IDT 来验证您的核心设备是否可以运行 Lambda 函数，以及该设备是否可以发布和订阅 MQTT 主题。 AWS IoT Core
- 提高了日志记录功能。
- 修复了其他小错误。

测试套件版本：

`GGV2Q_1.1.1`

- 2021.05.05 已发布

IDT v4.0.1 适用于 AWS IoT Greengrass

发行说明：

- 使您能够验证和鉴定运行 AWS IoT Greengrass 版本 2 软件的设备。
- 使您可以使用 f AWS IoT Device Tester or 开发和运行自定义测试套件 AWS IoT Greengrass。有关更多信息，请参阅 使用 IDT 开发和运行自己的测试套件。
- 提供适用于 macOS 和 Windows 的代码签名 IDT 应用程序。在 macOS 上，你可能需要为 IDT 授予安全例外。有关更多信息，请参阅 macOS 上的安全异常。

测试套件版本：

`GGV2Q_1.0.0`

- 2020.12.22 发布

- 除非您将features数组value中的相应测试设置为，否则测试套件仅运行资格所需的测试yes。

下载适用于 V2 的 AWS IoT Greengrass IDT

本主题介绍了 AWS IoT Greengrass V2 AWS IoT Device Tester 的下载选项。您可以使用以下软件下载链接之一，也可以按照说明以编程方式下载 IDT。

主题

- 手动下载 IDT
- 以编程方式下载 IDT

下载软件即表示您同意AWS IoT Device Tester 许可协议。

> **Note**
>
> IDT 不支持由多个用户从共享位置（如 NFS 目录或 Windows 网络共享文件夹）运行。建议您将 IDT 包解压缩到本地驱动器，并在本地工作站上运行 IDT 二进制文件。

手动下载 IDT

本主题列出了 AWS IoT Greengrass V2 版 IDT 支持的版本。作为最佳实践，我们建议您使用支持目标版本 AWS IoT Greengrass V2 的最新版本 IDT for V2。 AWS IoT Greengrass 的新版本 AWS IoT Greengrass 可能需要您下载适用于 AWS IoT Greengrass V2 的 IDT 的新版本。如果 IDT for AWS IoT Greengrass V2 与您正在使用的版本不兼容，则在开始测试运行时 AWS IoT Greengrass 会收到通知。

IDT v4.9.3 适用于 AWS IoT Greengrass

支持的 AWS IoT Greengrass 版本：

- Greengrass nucleus v2.12.0、v2.11. 0、v2.10.0 和 v2.9.5

IDT 软件下载：

- 带有适用于 Linux 的测试套件 GGV2Q_2.5.3 的 IDT v4.9.3
- 带测试套件 GGV2Q_2.5.3 的 IDT v4.9.3 适用于 macOS
- 带有适用于 Windows 的测试套件 GGV2Q_2.5.3 的 IDT v4.9.3

发行说明：

- 修复了从 Windows 主机测试 Linux 设备时组件测试中的一个问题，反之亦然。
- 从`localcomponent`测试组中移除`component`测试用例。此测试用例不再是资格认证所必需的。

附加注释：

- 如果你的设备使用 HSM，而你使用的是 nucleus 2.10.x，请迁移到 Greengrass nucleus 版本 2.11.0 或更高版本。

测试套件版本：

`GGV2Q_2.5.3`

- 2024.04.05 发布

以编程方式下载 IDT

IDT 提供了一个 API 操作，您可以使用该操作来检索 URL，也可以在其中以编程方式下载 IDT。您还可以使用此 API 操作来检查是否具有最新版本的 IDT。此 API 操作具有以下端点。

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

要调用此 API 操作，您必须具有执行 **`iot-device-tester:LatestIdt`** 操作的权限。包括您的 AWS 签名并`iot-device-tester`用作服务名称。

API 请求

HostOs — 主机的操作系统。从以下选项中进行选择：

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

TestSuiteType — 测试套件的类型。选择以下选项：

`GGV2`— V2 的 IDT AWS IoT Greengrass

ProductVersion

（可选）Greengrass 核的版本。该服务返回该版本的 Greengrass nucleus 的最新兼容版本的 IDT。如果不指定此选项，则该服务将返回 IDT 的最新版本。

API 响应

API 响应采用以下格式。DownloadURL 包括一个 zip 文件。

```
{
    "Success": True or False,
    "Message": Message,
    "LatestBk": {
        "Version": The version of the IDT binary,
        "TestSuiteVersion": The version of the test suite,
        "DownloadURL": The URL to download the IDT Bundle, valid for one hour
    }
 }
```

示例

您可以参考以下示例以编程方式下载 IDT。这些示例使用您在 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY 环境变量中存储的凭证。要遵循最佳安全实践，请勿在代码中存储您的凭证。

Example 示例：使用 cURL 版本 7.75.0 或更高版本下载（Mac 和 Linux）

如果您的 cURL 版本为 7.75.0 或更高版本，则可以使用 aws-sigv4 标记对 API 请求进行签名。该示例使用 jq 来解析响应中的下载 URL。

> **Warning**
>
> 该aws-sigv4标志要求 curl GET 请求的查询参数按HostOs/ProductVersion/TestSuiteType或HostOs/TestSuiteType的顺序排列。如果参数顺序不符合要求，则会导致从 API Gateway 获得不匹配的规范字符串签名错误。
> 如果包含可选参数ProductVersion，则必须使用 AWS IoT Greengrass V2 的支持版本中所述 AWS IoT Device Tester 的支持的产品版本。

- 将 us-we *st-2* 替换为你的。 AWS 区域有关区域代码的列表，请参阅区域端点。
- 将 *linux* 替换为主机的操作系统。
- 将 *2.5.3* 替换为你的 n AWS IoT Greengrass ucleus 版本。

```
url=$(curl --request GET "https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&ProductVersion=2.5.3&TestSuiteType=GGV2" \
--user $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY \
--aws-sigv4 "aws:amz:us-west-2:iot-device-tester" \
| jq -r '.LatestBk["DownloadURL"]')

curl $url --output devicetester.zip
```

Example 示例：使用早期版本的 cURL 下载（Mac 和 Linux）

您可以将以下 curl 命令与您签名并计算的 AWS 签名一起使用。有关如何签名和计算签名的更多信息，请参阅 AWS 签名 AWS API 请求。

- 将 *linux* 替换为主机的操作系统。
- 将 *Timestamp* 替换为日期和时间，例如 **20220210T004606Z**。
- 将 *Data* 替换为日期，例如 **20220210**。
- 替换*AWSRegion*为你的 AWS 区域。有关区域代码的列表，请参阅区域端点。
- *AWSSignature*替换为您生成的AWS 签名。

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&TestSuiteType=GGV2' \
--header 'X-Amz-Date: Timestamp \
--header 'Authorization: AWS4-HMAC-SHA256 Credential=$AWS_ACCESS_KEY_ID/Date/AWSRegion/
iot-device-tester/aws4_request, SignedHeaders=host;x-amz-date, Signature=AWSSignature'
```

Example 示例：使用 Python 脚本下载

此示例使用 Python 请求库。此示例改编自AWS 通用参考中的 Python 示例，用于签署 AWS API 请求。

- 将 *us-west-2* 替换为您的区域。有关区域代码的列表，请参阅区域端点。
- 将 *linux* 替换为主机的操作系统。

```
# Copyright 2010-2022 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# This file is licensed under the Apache License, Version 2.0 (the "License").
```

```
# You may not use this file except in compliance with the License. A copy of the
#License is located at
#
# http://aws.amazon.com/apache2.0/
#
# This file is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS
# OF ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
# This version makes a GET request and passes the signature
# in the Authorization header.
import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests
# ************* REQUEST VALUES *************
method = 'GET'
service = 'iot-device-tester'
host = 'download.devicetester.iotdevicesecosystem.amazonaws.com'
region = 'us-west-2'
endpoint = 'https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt'
request_parameters = 'HostOs=linux&TestSuiteType=GGV2'

# Key derivation functions. See:
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-
examples-python
def sign(key, msg):
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

def getSignatureKey(key, dateStamp, regionName, serviceName):
    kDate = sign(('AWS4' + key).encode('utf-8'), dateStamp)
    kRegion = sign(kDate, regionName)
    kService = sign(kRegion, serviceName)
    kSigning = sign(kService, 'aws4_request')
    return kSigning

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

# Create a date for headers and the credential string
```

```
t = datetime.datetime.utcnow()
amzdate = t.strftime('%Y%m%dT%H%M%SZ')
datestamp = t.strftime('%Y%m%d') # Date w/o time, used in credential scope

# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must
# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
 variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
# Note: The request can include any headers; canonical_headers and
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
```

```
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()

# ************* TASK 4: ADD SIGNING INFORMATION TO THE REQUEST *************
# The signing information can be either in a query string value or in
# a header named Authorization. This code shows how to use a header.
# Create authorization header and add to request headers
authorization_header = algorithm + ' ' + 'Credential=' + access_key + '/' +
 credential_scope + ', ' +  'SignedHeaders=' + signed_headers + ', ' + 'Signature=' +
 signature
# The request can include any headers, but MUST include "host", "x-amz-date",
# and (for this scenario) "Authorization". "host" and "x-amz-date" must
# be included in the canonical_headers and signed_headers, as noted
# earlier. Order here is not significant.
# Python note: The 'host' header is added automatically by the Python 'requests'
 library.
headers = {'x-amz-date':amzdate, 'Authorization':authorization_header}

# ************* SEND THE REQUEST *************
request_url = endpoint + '?' + canonical_querystring
print('\nBEGIN REQUEST++++++++++++++++++++++++++++++++++++')
print('Request URL = ' + request_url)
response = requests.get(request_url, headers=headers)
print('\nRESPONSE++++++++++++++++++++++++++++++++++++')
print('Response code: %d\n' % response.status_code)
print(response.text)

download_url = response.json()["LatestBk"]["DownloadURL"]
r = requests.get(download_url)
open('devicetester.zip', 'wb').write(r.content)
```

使用 IDT 运行 AWS IoT Greengrass 资格套件

您可以使用 AWS IoT Device Tester AWS IoT Greengrass V2 来验证 C AWS IoT Greengrass ore 软件是否在您的硬件上运行并且可以与通信。 AWS Cloud它还使用执行 end-to-end 测试 AWS IoT Core。例如，它会验证您的设备是否可以部署组件并对其进行升级。

除了测试设备外，IDT f AWS IoT Greengrass or V2 还会在中创建资源（例如， AWS IoT 事物、群组等）， AWS 账户 以简化认证流程。

要创建这些资源，适用于 AWS IoT Greengrass V2 的 IDT 使用`config.json`文件中配置的 AWS 凭据代表您进行 API 调用。这些资源将在测试过程的不同时间进行预置。

当您使用 IDT f AWS IoT Greengrass or V2 运行 AWS IoT Greengrass 资格套件时，它会执行以下步骤：

1. 加载和验证您的设备和凭证配置。
2. 使用所需的本地资源和云资源执行选定测试。
3. 清除本地资源和云资源。
4. 生成测试报告，指明您的主板是否已通过资格认证所需的测试。

测试套件版本

IDT for AWS IoT Greengrass V2 将测试组织到测试套件和测试组中。

- 测试套件是一组测试组，用于验证设备运行的是否为特定版本的 AWS IoT Greengrass。
- 测试组是与特定功能（例如组件部署）相关的一组单独测试。

例如`GGV2Q_1.0.0`，测试套件使用某种`major.minor.patch`格式进行版本控制。当你下载 IDT 时，该软件包包含最新的 Greengrass 资格套件版本。

> Important
>
> 不受支持的测试套件版本进行的测试对于设备资格认证无效。IDT 不会为不受支持的版本打印资格认证报告。有关更多信息，请参阅 the section called "的 Support AWS IoT Device Tester 政策 AWS IoT Greengrass"。
> 您可以运行列`list-supported-products`出当前版本的 IDT 支持的版本 AWS IoT Greengrass 和测试套件。

测试组描述

核心资格必备测试组

这些测试组必须使您的 AWS IoT Greengrass V2 设备符合 AWS Partner 设备目录的资格。

核心依赖关系

验证设备是否满足 C AWS IoT Greengrass ore 软件的所有软件和硬件要求。该测试组包括以下测试用例：

Java 版本

检查被测设备上是否安装了所需的 Java 版本。 AWS IoT Greengrass 需要 Java 8 或更高版本。

PreTest 验证

检查设备是否满足运行测试的软件要求。

- 对于基于 Linux 的设备，此测试将检查设备是否可以运行以下 Linux 命令：

  `chmod`, `cp`, `echo`, `grep`, `kill`, `ln`, `mkinfo`, `ps`, `rm`, `sh`, `uname`

- 对于基于 Windows 的设备，此测试会检查设备是否安装了以下 Microsoft 软件：

  Powershell v5.1 或更高版本、.NET v4.6.1 或更高版本、V isual C++ 2017 或更高版本、实用 PsExec

版本检查器

检查所 AWS IoT Greengrass 提供的版本是否与您正在使用的 AWS IoT 设备测试器版本兼容。

组件

验证设备是否可以部署组件并对其进行升级。该测试组包括以下测试：

云组件

验证云组件的设备功能。

本地组件

验证本地组件的设备功能。

Lambda

此测试不适用于基于 Windows 的设备。

验证设备是否可以部署使用 Java 运行时的 Lambda 函数组件，以及 Lambda 函数是否可以 AWS IoT Core 使用 MQTT 主题作为工作消息的事件源。

MQTT

验证设备是否可以订阅和发布 AWS IoT Core MQTT 主题。

可选测试组

Note

这些测试组是可选的，仅用于符合条件的基于 Linux 的 Greengrass 核心设备。如果您选择符合可选测试的资格，则您的设备将在 AWS Partner 设备目录中列出其他功能。

Docker 依赖关系

验证设备是否满足使用 AWS提供的 Docker 应用程序管理器 () `aws.greengrass.DockerApplicationManager` 组件所需的所有技术依赖项。

Docker 应用程序管理员资格

验证设备是否可以从亚马逊 ECR 下载 Docker 容器镜像。

Machine Learning

验证设备是否满足使用 AWS提供的机器学习 (ML) 组件所需的所有技术依赖项。

Machine Learning 推理测试

验证设备是否可以使用深度学习运行时和 L TensorFlow ite ML 框架执行 ML 推理。

直播管理器依赖关系

验证设备是否可以下载、安装和运行AWS IoT Greengrass 直播管理器。

硬件安全性集成 (HSI)

Note

该测试仅在 IDT v4.5.1 及更高版本中适用于基于 Linux 的设备。 AWS IoT Greengrass 目前不支持 Windows 设备的硬件安全集成。

验证设备是否可以使用存储在硬件安全模块 (HSM) 中的私钥和证书对与和 AWS IoT Greengrass 服务的连接进行身份验证。 AWS IoT 该测试还验证了提供的 PKCS #11 AWS提供者组件是否可以使用供应商提供的 PKCS #11 库与 HSM 接口。有关更多信息，请参阅 硬件安全性集成。

运行 AWS IoT Greengrass 资格套件的先决条件

本节介绍使用 AWS IoT Device Tester (IDT) 的先决条件。AWS IoT Greengrass

下载最新版本的 fo AWS IoT Device Tester r AWS IoT Greengrass

下载最新版本的 IDT 并将该软件解压缩到文件系统上您拥有读/写权限的位置 (*< device-tester-extract-location >*)。

> **Note**
>
> IDT 不支持由多个用户从共享位置（如 NFS 目录或 Windows 网络共享文件夹）运行。建议您将 IDT 包解压缩到本地驱动器，并在本地工作站上运行 IDT 二进制文件。
>
> Windows 的路径长度限制为 260 个字符。如果您使用的是 Windows，请将 IDT 提取到根目录（如 `C:\` 或 `D:\`）以使路径长度不超过 260 个字符的限制。

下载该AWS IoT Greengrass软件

IDT fo AWS IoT Greengrass r V2 会测试您的设备是否与特定版本的兼容。AWS IoT Greengrass运行以下命令将 AWS IoT Greengrass Core 软件下载到名为的文件中`aws.greengrass.nucleus.zip`。将*##*替换为您的 IDT 版本所支持的 nucleus 组件版本。

Linux or Unix

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

Windows Command Prompt (CMD)

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

PowerShell

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

将下载`aws.greengrass.nucleus.zip`的文件放在*<device-tester-extract-location>*/`products/`文件夹中。

> **Note**
>
> 对于相同的操作系统和架构，请勿在此目录中放置多个文件。

创建和配置 AWS 账户

在 AWS IoT Greengrass V2 中使用AWS IoT Device Tester之前，必须执行以下步骤：

1. 设置一个 AWS 账户。如果您已有 AWS 账户，请跳至步骤 2。
2. 为 IDT 配置权限。

这些账户权限允许 IDT 代表您访问AWS服务和创建AWS资源，例如AWS IoT事物和AWS IoT Greengrass组件。

要创建这些资源，适用于 AWS IoT Greengrass V2 的 IDT 使用`config.json`文件中配置的AWS凭据代表您进行 API 调用。这些资源将在测试过程的不同时间进行预置。

> **Note**
>
> 尽管大多数测试都符合AWS免费套餐资格，但在注册时必须提供信用卡AWS 账户。有关更多信息，请参阅我的账户使用的是免费套餐，为什么还需要提供付款方式？

第 1 步：设置 AWS 账户

在此步骤中，将创建并配置 AWS 账户。如果您已有 AWS 账户，请跳至 the section called “步骤 2：为 IDT 配置权限”。

如果您还没有 AWS 账户，请完成以下步骤来创建一个。

注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

当您注册 AWS 账户时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为管理用户分配管理访问权限，并且只使用根用户执行需要根用户访问权限的任务。

要创建管理员用户，请选择以下选项之一。

选择一种方法来管理您的管理员	目的	方式	您也可以
在 IAM Identity Center 中 （建议）	使用短期凭证访问 AWS。 这符合安全最佳实操。有关最佳实践的信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。	有关说明，请参阅《AWS IAM Identity Center 用户指南》中的入门。	按照《AWS Command Line Interface 用户指南》中的配置 AWS CLI 以使用 AWS IAM Identity Center，配置编程式访问。
在 IAM 中 （不推荐使用）	使用长期凭证访问 AWS。	按照《IAM 用户指南》中的创建您的首个 IAM 管理员用户和组的说明操作。	按照《IAM 用户指南》中的管理 IAM 用户的访问密钥，配置编程式访问。

步骤 2：为 IDT 配置权限

在此步骤中，配置 IDT for AWS IoT Greengrass V2 用于运行测试和收集 IDT 使用情况数据的权限。您可以使用AWS Management Console或 AWS Command Line Interface(AWS CLI) 为 IDT 创建 IAM 策略和测试用户，然后将策略附加到该用户。如果您已经为 IDT 创建了测试用户，请跳至。配置设备以运行 IDT 测试

为 IDT 配置权限（控制台）

1. 登录 IAM 控制台。
2. 创建客户托管策略，该策略授权创建具有特定权限的角色。

   a. 在导航窗格中，选择 Policies (策略)，然后选择 Create policy (创建策略)。
   b. 如果您未使用 PreInstalled，请在 JSON 选项卡上将占位符内容替换为以下策略。如果您正在使用 PreInstalled，请继续执行以下步骤。

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
```

```
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
```

```
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
```

```
          "greengrass:*",
          "iot:GetThingShadow",
          "iot:UpdateThingShadow",
          "iot:ListThings",
          "iot:DescribeEndpoint",
          "iot:CreateKeysAndCertificate"
        ],
        "Resource":"*"
      },
      {
        "Sid":"iamResourcesUpdate",
        "Effect":"Allow",
        "Action":[
          "iam:CreateRole",
          "iam:DeleteRole",
          "iam:CreatePolicy",
          "iam:DeletePolicy",
          "iam:AttachRolePolicy",
          "iam:DetachRolePolicy",
          "iam:TagRole",
          "iam:TagPolicy",
          "iam:GetPolicy",
          "iam:ListAttachedRolePolicies",
          "iam:ListEntitiesForPolicy"
        ],
        "Resource":[
          "arn:aws:iam::*:role/idt-*",
          "arn:aws:iam::*:policy/idt-*"
        ]
      }
    ]
}
```

c. 如果您使用的是 PreInstalled，请在 JSON 选项卡上将占位符内容替换为以下策略。确保你：

- 将`iotResources`语句中的 *ThingName* 和 *ThingGrou* p 替换为在受测设备 (DUT) 上安装 Greengrass 期间创建的事物名称和事物组，以添加权限。
- *######## roleAliasResourcesPass Role # roleAlias # #passRoleForResources## DUT ### Greengrass ########*

```
{
    "Version":"2012-10-17",
```

```
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
```

```
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
```

```
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
```

```
          "iam:CreateRole",
          "iam:DeleteRole",
          "iam:CreatePolicy",
          "iam:DeletePolicy",
          "iam:AttachRolePolicy",
          "iam:DetachRolePolicy",
          "iam:TagRole",
          "iam:TagPolicy",
          "iam:GetPolicy",
          "iam:ListAttachedRolePolicies",
          "iam:ListEntitiesForPolicy"
        ],
        "Resource":[
          "arn:aws:iam::*:role/idt-*",
          "arn:aws:iam::*:policy/idt-*"
        ]
      }
    ]
}
```

> **Note**
>
> 如果您想使用自定义 IAM 角色作为受测设备的令牌交换角色，请务必更新策略中的`roleAliasResources`声明和`passRoleForResources`声明以允许您的自定义 IAM 角色资源。

d. 选择查看策略。

e. 对于 Name (名称)，请输入 **IDTGreengrassIAMPermissions**。在 Summary (摘要) 下，查看策略授予的权限。

f. 选择创建策略。

3. 创建 IAM 用户并附加适用于 AWS IoT Greengrass 的 IDT 所需的权限。

   a. 创建 IAM 用户。按照 IAM 用户指南的创建 IAM 用户（控制台）中的步骤 1 到 5 操作。

   b. 将权限附加到您的 IAM 用户：

      i. 在 Set permissions (设置权限) 页面上，选择 Attach existing policies to user directly (直接附加现有策略到用户)。

      ii. 搜索您在上一步中创建的 IDTGreengrassIAMPermissions 策略。选中复选框。

   c. 选择下一步：标签。

d. 选择 Next: Review (下一步：审核) 以查看您的选择摘要。

e. 选择创建用户。

f. 要查看用户的访问密钥（访问密钥 ID 和秘密访问密钥），请选择密码和访问密钥旁边的 Show (显示)。要保存访问密钥，请选择Download.csv (下载 .csv)，然后将文件保存到安全位置。稍后您可以使用此信息配置 AWS 凭证文件。

4. 下一步：配置物理设备。

为 IDT 配置权限 (AWS CLI)

1. 在您的计算机上，安装并配置 AWS CLI（如果尚未安装）。按照《AWS Command Line Interface 用户指南》中安装 AWS CLI 的步骤来操作。

> Note
>
> AWS CLI 是一个开源工具，您可以使用此工具通过命令行 Shell 与 AWS 服务进行交互。

2. 创建用于授予管理 IDT 和 AWS IoT Greengrass 角色的权限的客户托管策略。

a. 如果您未使用 PreInstalled，请打开文本编辑器并将以下策略内容保存到 JSON 文件中。如果您正在使用 PreInstalled，请继续执行以下步骤。

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
```

```
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
```

```
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
```

```
        "Effect":"Allow",
        "Action":[
          "iot-device-tester:SendMetrics",
          "iot-device-tester:SupportedVersion",
          "iot-device-tester:LatestIdt",
          "iot-device-tester:CheckVersion",
          "iot-device-tester:DownloadTestSuite"
        ],
        "Resource":"*"
      },
      {
        "Sid":"genericResources",
        "Effect":"Allow",
        "Action":[
          "greengrass:*",
          "iot:GetThingShadow",
          "iot:UpdateThingShadow",
          "iot:ListThings",
          "iot:DescribeEndpoint",
          "iot:CreateKeysAndCertificate"
        ],
        "Resource":"*"
      },
      {
        "Sid":"iamResourcesUpdate",
        "Effect":"Allow",
        "Action":[
          "iam:CreateRole",
          "iam:DeleteRole",
          "iam:CreatePolicy",
          "iam:DeletePolicy",
          "iam:AttachRolePolicy",
          "iam:DetachRolePolicy",
          "iam:TagRole",
          "iam:TagPolicy",
          "iam:GetPolicy",
          "iam:ListAttachedRolePolicies",
          "iam:ListEntitiesForPolicy"
        ],
        "Resource":[
          "arn:aws:iam::*:role/idt-*",
          "arn:aws:iam::*:policy/idt-*"
        ]
      }
```

```
  ]
}
```

b. 如果您正在使用 PreInstalled，请打开文本编辑器并将以下策略内容保存到 JSON 文件中。确保你：

- 在被*### (DUT) ### Greengrass #####iotResources##### ThingNam e # ThingGro* up 以添加权限。
- *######## roleAliasResourcesPass Role # roleAlias # #passRoleForResources## DUT ### Greengrass ########*

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
```

```
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
```

```
    "Effect":"Allow",
    "Action":[
      "s3:GetObject",
      "s3:PutObject",
      "s3:DeleteObjectVersion",
      "s3:DeleteObject",
      "s3:CreateBucket",
      "s3:ListBucket",
      "s3:ListBucketVersions",
      "s3:DeleteBucket",
      "s3:PutObjectTagging",
      "s3:PutBucketTagging"
    ],
    "Resource":"arn:aws:s3::*:idt-*"
  },
  {
    "Sid":"roleAliasResources",
    "Effect":"Allow",
    "Action":[
      "iot:CreateRoleAlias",
      "iot:DescribeRoleAlias",
      "iot:DeleteRoleAlias",
      "iot:TagResource",
      "iam:GetRole"
    ],
    "Resource":[
      "arn:aws:iot:*:*:rolealias/roleAlias",
      "arn:aws:iam::*:role/idt-*"
    ]
  },
  {
    "Sid":"idtExecuteAndCollectMetrics",
    "Effect":"Allow",
    "Action":[
      "iot-device-tester:SendMetrics",
      "iot-device-tester:SupportedVersion",
      "iot-device-tester:LatestIdt",
      "iot-device-tester:CheckVersion",
      "iot-device-tester:DownloadTestSuite"
    ],
    "Resource":"*"
  },
  {
    "Sid":"genericResources",
```

```
        "Effect":"Allow",
        "Action":[
          "greengrass:*",
          "iot:GetThingShadow",
          "iot:UpdateThingShadow",
          "iot:ListThings",
          "iot:DescribeEndpoint",
          "iot:CreateKeysAndCertificate"
        ],
        "Resource":"*"
      },
      {
        "Sid":"iamResourcesUpdate",
        "Effect":"Allow",
        "Action":[
          "iam:CreateRole",
          "iam:DeleteRole",
          "iam:CreatePolicy",
          "iam:DeletePolicy",
          "iam:AttachRolePolicy",
          "iam:DetachRolePolicy",
          "iam:TagRole",
          "iam:TagPolicy",
          "iam:GetPolicy",
          "iam:ListAttachedRolePolicies",
          "iam:ListEntitiesForPolicy"
        ],
        "Resource":[
          "arn:aws:iam::*:role/idt-*",
          "arn:aws:iam::*:policy/idt-*"
        ]
      }
    ]
}
```

Note

如果您想使用自定义 IAM 角色作为受测设备的令牌交换角色，请务必更新策略中的`roleAliasResources`声明和`passRoleForResources`声明以允许您的自定义 IAM 角色资源。

c. 运行以下命令创建名为的客户托管策略`IDTGreengrassIAMPermissions`。*policy.json*替换为您在上一步中创建的 JSON 文件的完整路径。

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-document file://policy.json
```

3. 创建 IAM 用户并附加适用于 AWS IoT Greengrass 的 IDT 所需的权限。

   a. 创建 IAM 用户。在此示例设置中，用户被命名为 `IDTGreengrassUser`。

   ```
   aws iam create-user --user-name IDTGreengrassUser
   ```

   b. 将您在步骤 2 中创建的 `IDTGreengrassIAMPermissions` 策略附加到您的 IAM 用户。将命令中的 *<account-id>* 替换为您的 AWS 账户 ID。

   ```
   aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
    arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
   ```

4. 为用户创建私密访问密钥。

   ```
   aws iam create-access-key --user-name IDTGreengrassUser
   ```

   将输出存储在安全位置。稍后您可以使用此信息配置 AWS 凭证文件。

5. 下一步：配置物理设备。

AWS IoT Device Tester 权限

以下策略描述了AWS IoT Device Tester权限。

AWS IoT Device Tester版本检查和自动更新功能需要这些权限。

- `iot-device-tester:SupportedVersion`

  授予 AWS IoT Device Tester 获取受支持产品、测试套件和 IDT 版本列表的权限。

- `iot-device-tester:LatestIdt`

  授予 AWS IoT Device Tester 获取可供下载的最新 IDT 版本的权限。

- `iot-device-tester:CheckVersion`

授予 AWS IoT Device Tester 检查 IDT、测试套件和产品的版本兼容性的权限。

- `iot-device-tester:DownloadTestSuite`

  授AWS IoT Device Tester予下载测试套件更新的权限。

AWS IoT Device Tester还使用以下权限来报告可选指标：

- `iot-device-tester:SendMetrics`

  授AWS予收集AWS IoT Device Tester内部使用情况指标的权限。如果省略此权限，则不会收集这些指标。

配置设备以运行 IDT 测试

要让 IDT 运行设备资格测试，您必须将主机配置为访问您的设备，并在您的设备上配置用户权限。

在主机上安装 Java

从 IDT v4.2.0 开始，可选的资格测试AWS IoT Greengrass要求运行 Java。

你可以使用 Java 版本 8 或更高版本。我们建议您使用 Amazon Cor retto 或 OpenJDK 长期支持版本。需要版本 8 或更高版本。

配置主机以访问所测试设备

IDT 在主机上运行，并且必须能够使用 SSH 连接到您的设备。有两个选项允许 IDT 获得对所测试设备的 SSH 访问权限：

1. 按照此处的说明创建一个 SSH 密钥对并授权您的密钥，以便登录所测试设备而无需指定密码。
2. 在 `device.json` 文件中为每个设备提供用户名和密码。有关更多信息，请参阅 配置 device.json。

您可以使用任何 SSL 实施创建 SSH 密钥。以下说明介绍如何使用 SSH-KEYGEN 或 PuTTYgen（对于 Windows）。如果您使用的是另一个 SSL 实施，请参阅该实施的文档。

IDT 使用 SSH 密钥对所测试设备进行身份验证。

使用 SSH-KEYGEN 创建 SSH 密钥

1. 创建 SSH 密钥。

您可以使用 Open SSH ssh-keygen 命令创建 SSH 密钥对。如果您的主机上已有一个 SSH 密钥对，则最佳做法是专门为 IDT 创建一个 SSH 密钥对。这样，完成测试后，如果没有输入密码，主机将无法再连接到设备。它还使您能够仅向需要访问远程设备的人员授予访问权限。

Note

Windows 没有安装 SSH 客户端。有关在 Windows 上安装 SSH 客户端的信息，请参阅下载 SSH 客户端软件。

ssh-keygen 命令会提示您输入要存储密钥对的名称和路径。默认情况下，密钥对文件的名称为 `id_rsa`（私有密钥）和 `id_rsa.pub`（公有密钥）。在 macOS 和 Linux 上，这些文件的默认位置为 `~/.ssh/`。在 Windows 上，默认位置为 `C:\Users\<user-name>\.ssh`。

根据提示，输入密钥短语来保护您的 SSH 密钥。有关更多信息，请参阅生成新的 SSH 密钥。

2. 向所测试设备添加经过授权的 SSH 密钥。

   IDT 必须使用您的 SSH 私有密钥登录所测试设备。要授权 SSH 私有密钥以登录所测试设备，请在主机上使用 ssh-copy-id 命令。此命令会将您的公有密钥添加到所测试设备上的 `~/.ssh/authorized_keys` 文件中。例如：

   $ ssh-copy-id ***<remote-ssh-user>***@***<remote-device-ip>***

   哪里*remote-ssh-user*是用于登录被测设备的用户名，以及*remote-device-ip*用于运行测试的被测设备的 IP 地址。例如：

   ssh-copy-id pi@192.168.1.5

   系统提示时，输入在 ssh-copy-id 命令中指定的用户名所对应的密码。

   ssh-copy-id 公有密钥的名称为 `id_rsa.pub` 并且存储在默认位置（macOS 和 Linux 上的位置为 `~/.ssh/`，Windows 上的位置为 `C:\Users\<user-name>\.ssh`）。如果公有密钥采用其他名称或存储在其他位置，则必须使用 -i 选项与 ssh-copy-id 指定 SSH 公有密钥的完全限定路径（例如，ssh-copy-id -i ~/my/path/myKey.pub）。有关创建 SSH 密钥和复制公有密钥的更多信息，请参阅 SSH-COPY-ID。

使用 PuTTYgen 创建 SSH 密钥（仅限 Windows）

1. 确保您在所测试的设备上安装了 OpenSSH 服务器和客户端。有关更多信息，请参阅 OpenSSH。
2. 在所测试的设备上安装 PuTTYgen。
3. 打开 PuTTYgen。
4. 选择 Generate (生成)，然后在框中移动鼠标光标以生成私有密钥。
5. 从 Conversions (转换) 菜单中，选择 Export OpenSSH key (导出 OpenSSH 密钥)，然后使用 `.pem` 文件扩展名保存私有密钥 。
6. 将公有密钥添加到所测试设备上的 `/home/<user>/.ssh/authorized_keys` 文件中。
   a. 从 PuTTYgen 窗口复制公有密钥文本。
   b. 使用 PuTTY 在所测试设备上创建会话。
      i. 从命令提示符或 Windows Powershell 窗口中，运行以下命令：

      ```
      C:/<path-to-putty>/putty.exe -ssh <user>@<dut-ip-address>
      ```
      ii. 在系统提示时，输入您的设备的密码。
      iii. 使用 vi 或其他文本编辑器将公有密钥附加到所测试设备上的 `/home/<user>/.ssh/authorized_keys` 文件中。
7. 使用您的用户名、IP 地址以及您刚刚为每个所测试的设备保存在主机上的私钥文件的路径更新 `device.json` 文件。有关更多信息，请参阅 the section called “配置 device.json”。确保提供私有密钥的完整路径和文件名，并使用正斜杠（“/”）。例如，对于 Windows 路径 `C:\DT\privatekey.pem`，请在 `device.json` 文件中使用 `C:/DT/privatekey.pem`。

为 Windows 设备配置用户凭证

要获得基于 Windows 的设备的资格，您必须在被测设备上的LocalSystem 帐户中为以下用户配置用户凭据：

- 默认 Greengrass 用户 ()。`ggc_user`
- 您用来连接到被测设备的用户。您可以在`device.json`文件中配置此用户。

您必须在被测设备上的 LocalSystem 账户中创建每个用户，然后将该用户的用户名和密码存储在 LocalSystem 账户的 Credential Manager 实例中。

在 Windows 设备上配置用户

1. 以管理员身份打开 Windows 命令提示符 (`cmd.exe`)。
2. 在 Windows 设备上的 LocalSystem 帐户中创建用户。为要创建的每个用户运行以下命令。*#### Greengrass ############* ggc_user将*##*替换为安全密码。

   ```
   net user /add user-name password
   ```

3. 从微软下载该PsExec实用程序并将其安装到设备上。
4. 使用该 PsExec 实用程序将默认用户的用户名和密码存储在 LocalSystem 账户的凭据管理器实例中。

   为要在凭据管理器中配置的每个用户运行以下命令。*#### Greengrass ############* ggc_user将*##*替换为您之前设置的用户密码。

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

   如果PsExec License Agreement打开，Accept请选择同意许可并运行命令。

   Note

   在 Windows 设备上，该 LocalSystem 帐户运行 Greengrass 核心，您必须使用 PsExec 该实用程序在帐户中存储用户信息。 LocalSystem使用凭据管理器应用程序将此信息存储在当前登录用户的 Windows 帐户中，而不是 LocalSystem帐户中。

在您的设备上配置用户权限

IDT 将对所测试设备中的各种目录和文件执行操作。其中一些操作需要提升的权限（使用 sudo）。要自动执行这些操作，AWS IoT GreengrassV2 版 IDT 必须能够在不被提示输入密码的情况下使用 sudo 运行命令。

请在所测试设备上执行以下步骤，以允许在不提示输入密码的情况下进行 sudo 访问。

Note

`username` 是指 IDT 用来访问所测试设备的 SSH 用户。

将用户添加到 sudo 组

1. 在所测试设备上，运行 `sudo usermod -aG sudo <username>`。
2. 注销，然后重新登录，以使更改生效。
3. 要验证您的用户名是否已成功添加，请运行 sudo echo test。如果系统未提示您输入密码，则说明已正确配置您的用户。
4. 打开 `/etc/sudoers` 文件，并将以下行添加到文件末尾：

```
<ssh-username> ALL=(ALL) NOPASSWD: ALL
```

配置自定义令牌交换角色

您可以选择使用自定义 IAM 角色作为被测设备假设的与AWS资源交互的令牌交换角色。有关创建 IAM 角色的信息，请参阅《IAM 用户指南》中的创建 IAM 角色。

您必须满足以下要求才能允许 IDT 使用您的自定义 IAM 角色。我们强烈建议您仅向该角色添加最低要求的策略操作。

- 必须更新 userdata.json 配置文件才能将参数设置为。`GreengrassV2TokenExchangeRole` `true`
- 必须使用以下最低信任策略配置自定义 IAM 角色：

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":[
               "credentials.iot.amazonaws.com",
               "lambda.amazonaws.com",
               "sagemaker.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

- 必须使用以下最低权限策略配置自定义 IAM 角色：

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iot:DescribeCertificate",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:DescribeLogStreams",
            "iot:Connect",
            "iot:Publish",
            "iot:Subscribe",
            "iot:Receive",
            "iot:ListThingPrincipals",
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:PutObject",
            "s3:AbortMultipartUpload",
            "s3:ListMultipartUploadParts"
         ],
         "Resource":"*"
      }
   ]
}
```

- 自定义 IAM 角色的名称必须与您在测试用户的 IAM 权限中指定的 IAM 角色资源相匹配。默认情况下，测试用户策略允许访问角色名称中`idt-`带有前缀的 IAM 角色。如果您的 IAM 角色名称不使用此前缀，请将`arn:aws:iam::*:role/custom-iam-role-name`资源添加到测试用户策略的`passRoleForResources`语`roleAliasResources`句和语句中，如以下示例所示：

Example **passRoleForResources** statement

```
{
   "Sid":"passRoleForResources",
   "Effect":"Allow",
   "Action":"iam:PassRole",
   "Resource":"arn:aws:iam::*:role/custom-iam-role-name",
```

```
    "Condition":{
       "StringEquals":{
          "iam:PassedToService":[
             "iot.amazonaws.com",
             "lambda.amazonaws.com",
             "greengrass.amazonaws.com"
          ]
       }
    }
}
```

Example **roleAliasResources** statement

```
{
    "Sid":"roleAliasResources",
    "Effect":"Allow",
    "Action":[
       "iot:CreateRoleAlias",
       "iot:DescribeRoleAlias",
       "iot:DeleteRoleAlias",
       "iot:TagResource",
       "iam:GetRole"
    ],
    "Resource":[
       "arn:aws:iot:*:*:rolealias/idt-*",
       "arn:aws:iam::*:role/custom-iam-role-name"
    ]
}
```

配置设备以测试可选功能

本节介绍针对可选 Docker 和机器学习 (ML) 功能运行 IDT 测试的设备要求。只有在您想要测试这些功能时，才必须确保您的设备满足这些要求。否则，请继续查看the section called “配置 IDT 设置”。

主题

- 码头工人资格要求
- 机器学习资格要求
- HSM 资格要求

码头工人资格要求

IDT f AWS IoT Greengrass or V2 提供 Docker 资格测试，以验证您的设备是否可以使用AWS提供的 Docker 应用程序管理器组件下载您可以使用自定义 Dock er 容器组件运行的 Docker 容器镜像。有关创建自定义 Docker 组件的信息，请参阅运行 Docker 容器。

要运行 Docker 资格测试，您的被测设备必须满足以下要求才能部署 Docker 应用程序管理器组件。

- 安装在 Greengr@@ ass 核心设备上的 Docker E ngine 1.9.1 或更高版本。版本 20.10 是经验证可与AWS IoT Greengrass核心软件配合使用的最新版本。在部署运行 Docker 容器的组件之前，必须直接在核心设备上安装 Docker。
- 在部署此组件之前，Docker 守护程序已启动并在核心设备上运行。
- 运行 Docker 容器组件的系统用户必须具有根或管理员权限，或者您必须将 Docker 配置为以非根用户或非管理员用户身份运行。
  - 在 Linux 设备上，您可以将用户添加到docker群组中，无需用户即可调用docker命令sudo。
  - 在 Windows 设备上，无需管理员权限即可将用户添加到docker-users群组中以调用docker命令。

  Linux or Unix

    要将或用于运行 Docker 容器组件的非 root 用户添加到ggc_userdocker群组，请运行以下命令。

    ```
    sudo usermod -aG docker ggc_user
    ```

    有关更多信息，请参阅以非 root 用户身份管理 Docker。

  Windows Command Prompt (CMD)

    要将或您用来运行 Docker 容器组件的用户添加到ggc_userdocker-users群组，请以管理员身份运行以下命令。

    ```
    net localgroup docker-users ggc_user /add
    ```

  Windows PowerShell

    要将或您用来运行 Docker 容器组件的用户添加到ggc_userdocker-users群组，请以管理员身份运行以下命令。

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

机器学习资格要求

IDT f AWS IoT Greengrass or V2 提供机器学习资格测试，以验证您的设备是否可以使用AWS提供的机器学习组件，使用深度学习运行时或 TensorFlow Lite ML 框架在本地执行机器学习推理。有关在 Greengrass 设备上运行 ML 推理的更多信息，请参阅。执行机器学习推理

要运行机器学习资格测试，您的被测设备必须满足以下要求才能部署机器学习组件。

- 在运行亚马逊 Linux 2 或 Ubuntu 18.04 的 Greengrass 核心设备上，设备上安装了 GNU C 库 (glibc) 2.27 或更高版本。
- 在 armv7L 设备上，例如 Raspberry Pi，设备上安装了 OpenCV-Python 的依赖关系。运行以下命令来安装依赖项。

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- 运行 Raspberry Pi OS Bullseye 的 Raspberry Pi 设备必须满足以下要求：
  - NumPy 设备上安装了 1.22.4 或更高版本。Raspberry Pi OS Bullseye 包含的早期版本 NumPy，因此你可以运行以下命令在设备 NumPy 上进行升级。

    ```
    pip3 install --upgrade numpy
    ```

  - 设备上已启用旧版相机堆栈。Raspberry Pi OS Bullseye 包含一个新的相机堆栈，该堆栈默认处于启用状态且不兼容，因此您必须启用旧版相机堆栈。

    启用旧版相机堆栈

    1. 运行以下命令打开 Raspberry Pi 配置工具。

       ```
       sudo raspi-config
       ```

    2. 选择 “接口选项”。
    3. 选择 “旧版相机” 以启用旧版相机堆栈。
    4. 重启 Raspberry Pi。

HSM 资格要求

AWS IoT Greengrass提供 PKCS #11 提供程序组件，用于与设备上的 PKCS 硬件安全模块 (HSM) 集成。HSM 设置取决于您的设备和您选择的 HSM 模块。只要提供了 IDT 配置设置中记录的预期 HSM 配置，IDT 就会获得运行此可选功能资格测试所需的信息。

配置 IDT 设置以运行 AWS IoT Greengrass 资格认证套件

在运行测试之前，必须配置主机上的 AWS 凭证和设备的设置。

在 config.json 中配置AWS凭据

您必须在 *<device_tester_extract_location>*`/configs/config.json` 文件中配置 IAM 用户凭证。使用中创建的 AWS IoT Greengrass V2 版 IDT 用户的凭证。the section called “创建和配置 AWS 账户”您可以采用以下两种方法之一来指定凭证：

- 在凭证文件中
- 作为环境变量

通过凭证文件配置 AWS 凭证

IDT 使用与 AWS CLI 相同的凭证文件。有关更多信息，请参阅配置和凭证文件。

凭证文件的位置因您使用的操作系统而异：

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

按以下格式将 AWS 凭证添加到 `credentials` 文件中：

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

要将 IDT f AWS IoT Greengrass or V2 配置为使用`credentials`文件中的AWS凭据，请按如下方式编辑您的`config.json`文件：

```
{
```

```
  "awsRegion": "region",
  "auth": {
    "method": "file",
    "credentials": {
      "profile": "default"
    }
  }
}
```

Note

如果您未使用 default AWS 配置文件，请确保在 config.json 文件中更改配置文件名。有关更多信息，请参阅命名配置文件。

通过环境变量配置 AWS 凭证

环境变量是由操作系统维护且由系统命令使用的变量。如果您关闭 SSH 会话，则不会保存它们。适用于 AWS IoT Greengrass V2 的 IDT 可以使用AWS_ACCESS_KEY_ID和AWS_SECRET_ACCESS_KEY环境变量来存储您的AWS证书。

要在 Linux、macOS 或 Unix 上设置这些变量，请使用 export：

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

要在 Windows 上设置这些变量，请使用 set：

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

要配置 IDT 以使用环境变量，请编辑 config.json 文件中的 auth 部分。示例如下：

```
{
  "awsRegion": "region",
  "auth": {
    "method": "environment"
  }
}
```

配置 device.json

除了AWS凭证外，IDT fo AWS IoT Greengrass r V2 还需要有关运行测试的设备的信息。示例信息包括 IP 地址、登录信息、操作系统和 CPU 架构。

您必须使用位于 *<device_tester_extract_location>*/configs/device.json 中的 device.json 模板提供此信息：

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "ml",
        "value": "dlr | tensorflowlite | dlr,tensorflowlite | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "hsm | no"
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "operatingSystem": "Linux | Windows",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "publicKeyPath": "<public-key-path>",
```

```
            "auth": {
              "method": "pki | password",
              "credentials": {
                "user": "<user-name>",
                "privKeyPath": "/path/to/private/key",
                "password": "<password>"
              }
            }
          }
        }
      ]
    }
]
```

> **Note**
>
> 只有当 `method` 设置为 `pki` 时才指定 `privKeyPath`。
>
> 只有当 `method` 设置为 `password` 时才指定 `password`。

所有包含值的属性都是必需的，如下所述：

`id`

一个用户定义的字母数字 ID，用于唯一地标识称作设备池的设备集合。属于池的设备必须具有相同的硬件。运行一组测试时，池中的设备将用于对工作负载进行并行化处理。多个设备用于运行不同测试。

`sku`

唯一标识所测试设备的字母数字值。该 SKU 用于跟踪符合条件的主板。

> **Note**
>
> 如果您想在设备目录中列出您的AWS Partner设备，则在此处指定的 SKU 必须与您在发布过程中使用的 SKU 相匹配。

`features`

包含设备支持的功能的数组。所有功能都是必需的。

arch

测试运行验证的支持的操作系统架构。有效值为：

- x86_64
- armv6l
- armv7l
- aarch64

ml

验证设备是否满足使用AWS提供的机器学习 (ML) 组件所需的所有技术依赖项。

启用此功能还可以验证设备是否可以使用深度学习运行时和 TensorFlow Lite ML 框架执行机器学习推理。

有效值是dlr和tensorflowlite、或的任意组合no。

docker

验证设备是否满足使用AWS提供的 Docker 应用程序管理器 () aws.greengrass.DockerApplicationManager 组件所需的所有技术依赖项。

启用此功能还可以验证设备是否可以从 Amazon ECR 下载 Docker 容器映像。

有效值是yes或的任意组合no。

streamManagement

验证设备是否可以下载、安装和运行AWS IoT Greengrass直播管理器。

有效值是yes或的任意组合no。

hsi

验证设备是否可以使用存储在硬件安全模块 (HSM) 中的私钥和证书对与和AWS IoT Greengrass服务的连接进行身份验证。AWS IoT该测试还验证了提供的 PKCS #11 AWS 提供者组件是否可以使用供应商提供的 PKCS #11 库与 HSM 接口。有关更多信息，请参阅 硬件安全性集成。

有效值为 hsm 或 no。

> **Note**
>
> IDT v4.2.0 及更高版本支持测试mldocker、和功能。streamManagement如果您不想测试这些功能，请将相应的值设置为no。

Note

测试hsi仅在 IDT v4.5.1 及更高版本中可用。

`devices.id`

用户定义的测试的设备的唯一标识符。

`devices.operatingSystem`

设备操作系统。支持的值为 `Linux` 和 `Windows`。

`connectivity.protocol`

用于与此设备通信的通信协议。当前，唯一支持的值`ssh`适用于物理设备。

`connectivity.ip`

测试的设备 IP 地址。

此属性仅在 `connectivity.protocol` 设置为 `ssh` 时适用。

`connectivity.port`

可选。用于 SSH 连接的端口号。

默认值为 22。

此属性仅在 `connectivity.protocol` 设置为 `ssh` 时适用。

`connectivity.publicKeyPath`

可选。用于验证待测设备连接的公有密钥的完整路径。

如果指定 `publicKeyPath`，IDT 会在与待测设备建立 SSH 连接时验证设备的公有密钥。如果未指定此值，IDT 将创建 SSH 连接，但不验证设备的公有密钥。

我们强烈建议您指定公有密钥的路径，并使用安全的方法来获取此公有密钥。对于基于标准命令行的 SSH 客户端，known_hosts 文件中提供了公有密钥。如果您指定单独的公有密钥文件，则该文件必须使用与 known_hosts 文件相同的格式，即 *`ip-address key-type public-key`*。如果有多个条目具有相同 IP 地址，则 IDT 使用的密钥类型的条目必须位于文件中的其他条目之前。

`connectivity.auth`

连接的身份验证信息。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.auth.method

用于通过给定的连接协议访问设备的身份验证方法。

支持的值为：

- pki
- password

connectivity.auth.credentials

用于身份验证的凭证。

connectivity.auth.credentials.password

该密码用于登录到正在测试的设备。

此值仅在 connectivity.auth.method 设置为 password 时适用。

connectivity.auth.credentials.privKeyPath

用于登录所测试设备的私有密钥的完整路径。

此值仅在 connectivity.auth.method 设置为 pki 时适用。

connectivity.auth.credentials.user

用于登录所测试设备的用户名。

配置 userdata.json

IDT fo AWS IoT Greengrass r V2 还需要有关测试工件和AWS IoT Greengrass软件位置的更多信息。

您必须使用位于 *<device_tester_extract_location>*/configs/userdata.json 中的 userdata.json 模板提供此信息：

```
{
    "TempResourcesDirOnDevice": "/path/to/temp/folder",
    "InstallationDirRootOnDevice": "/path/to/installation/folder",
    "GreengrassNucleusZip": "/path/to/aws.greengrass.nucleus.zip",
    "PreInstalled": "yes/no",
    "GreengrassV2TokenExchangeRole": "custom-iam-role-name",
 "hsm": {
        "greengrassPkcsPluginJar": "/path/to/aws.greengrass.crypto.Pkcs11Provider-
latest.jar",
```

```
        "pkcs11ProviderLibrary": "/path/to/pkcs11-vendor-library",
        "slotId": "slot-id",
        "slotLabel": "slot-label",
        "slotUserPin": "slot-pin",
        "keyLabel": "key-label",
        "preloadedCertificateArn": "certificate-arn"
        "rootCA": "path/to/root-ca"
    }
}
```

所有包含值的属性都是必需的，如下所述：

`TempResourcesDirOnDevice`

被测设备上用于存储测试工件的临时文件夹的完整路径。确保不需要 sudo 权限即可写入此目录。

> **Note**
>
> IDT 在完成测试运行后会删除该文件夹的内容。

`InstallationDirRootOnDevice`

设备上要安装的文件夹的完整路径AWS IoT Greengrass。对于 PreInstalled Greengrass 来说，这是 Greengrass 安装目录的路径。

您必须为此文件夹设置所需的文件权限。对安装路径中的每个文件夹运行以下命令。

```
sudo chmod 755 folder-name
```

`GreengrassNucleusZip`

主机上 Greengrass nucleus ZIP `greengrass-nucleus-latest.zip` () 文件的完整路径。使用 PreInstalled Greengrass 进行测试时不需要此字段。

> **Note**
>
> 有关适用于 IDT 的 Greengrass nucleus 支持版本的信息，请参阅。AWS IoT Greengrass [适用 AWS IoT Greengrass 于 V2 的最新 IDT 版本要下载最新的 Greengrass 软件，请参阅下载该软件。AWS IoT Greengrass]

`PreInstalled`

此功能仅适用于 Linux 设备上的 IDT v4.5.8 及更高版本。

（可选）当值为 "#" 时，IDT 将假定中提到的路径是InstallationDirRootOnDevice安装 Greengrass 的目录。

有关如何在您的设备上安装 Greengrass 的更多信息，请参阅。安装具有自动资源配置功能的 AWS IoT Greengrass Core 软件如果使用手动配置进行安装，请在手动创建AWS IoT 事物时包含 "将事物添加到新的或现有AWS IoT的事物组" 步骤。IDT 假设事物和事物组是在安装设置期间创建的。确保这些值反映在effectiveConfig.yaml文件中。IDT 正在检查effectiveConfig.yaml下<InstallationDirRootOnDevice>/config/effectiveConfig.yaml方的文件。

要使用 HSM 运行测试，请确保在中effectiveConfig.yaml更新了该aws.greengrass.crypto.Pkcs11Provider字段。

`GreengrassV2TokenExchangeRole`

（可选）您要用作被测设备假定与AWS资源交互的令牌交换角色的自定义 IAM 角色。

> **Note**
>
> IDT 使用此自定义 IAM 角色，而不是在测试运行期间创建默认的令牌交换角色。如果您使用自定义角色，则可以更新测试用户的 IAM 权限，以排除允许该用户创建和删除 IAM 角色和策略的iamResourcesUpdate语句。

有关创建自定义 IAM 角色作为令牌交换角色的更多信息，请参阅配置自定义令牌交换角色。

`hsm`

此功能适用于 IDT v4.5.1 及更高版本。

（可选）用于使用AWS IoT Greengrass硬件安全模块 (HSM) 进行测试的配置信息。否则，应忽略 hsm 属性。有关更多信息，请参阅 硬件安全性集成。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

> **Warning**
>
> 如果硬件安全模块在 IDT 与其他系统之间共享，则 HSM 配置可能被视为敏感数据。在这种情况下，您可以通过将这些配置值存储在 Paramet AWS er Stor SecureString e 参数中并

配置 IDT 使其在测试执行期间获取它们，从而避免以明文形式保护这些配置值。有关更多信息，请参阅 ???。

hsm.greengrassPkcsPluginJar

您下载到 IDT 主机的 PKCS #11 提供程序组件的完整路径。 AWS IoT Greengrass将此组件作为 JAR 文件提供，您可以下载该文件以在安装过程中将其指定为预配插件。您可以通过以下 URL 下载该组件 JAR 文件的最新版本：https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar。

hsm.pkcs11ProviderLibrary

PKCS #11 库的完整路径，该库由硬件安全模块 (HSM) 供应商提供，用于与 HSM 交互。

hsm.slotId

用于识别要加载密钥和证书的 HSM 插槽的插槽 ID。

hsm.slotLabel

用于识别要加载密钥和证书的 HSM 插槽的插槽标签。

hsm.slotUserPin

IDT 用来向 HSM 验证AWS IoT Greengrass核心软件的用户 PIN。

> **Note**
>
> 作为安全最佳实践，请勿在生产设备上使用相同的用户 PIN。

hsm.keyLabel

用于标识硬件模块中的键的标签。密钥和证书必须使用相同的密钥标签。

hsm.preloadedCertificateArn

云端上传的设备证书的亚马逊资源名称 (ARN)。AWS IoT

您之前必须使用 HSM 中的密钥生成此证书，将其导入您的 HSM，然后将其上传到云端。AWS IoT有关生成和导入证书的信息，请参阅您的 HSM 文档。

您必须将证书上传到您在 config.js on 中提供的相同账户和区域。 。有关将证书上传到的更多信息AWS IoT，请参阅《AWS IoT开发人员指南》中的手动注册客户端证书。

hsm.rootCAPath

（可选）IDT 主机上指向签署证书的根证书颁发机构 (CA) 的完整路径。如果您在 HSM 中创建的证书不是由 Amazon 根 CA 签名的，则需要这样做。

从AWS参数存储中获取配置

AWS IoT设备测试器 (IDT) 包含一项可选功能，用于从 S AWSystems Manager 参数存储区获取配置值。 AWS参数存储允许对配置进行安全和加密的存储。配置后，IDT 可以从 Paramet AWS er Store 获取参数，而不是将参数以纯文本形式存储在文件中。userdata.json这对于任何应加密存储的敏感数据都很有用，例如：密码、PIN 和其他机密。

1. 要使用此功能，您必须更新创建 IDT 用户时使用的权限，以允许对 IDT 配置为使用的参数进行 GetParameter 操作。以下是可以添加到 IDT 用户的权限声明的示例。有关更多信息，请参阅AWS Systems Manager用户指南。

```
{
        "Sid":"parameterStoreResources",
        "Effect": "Allow",
         "Action": [
             "ssm:GetParameter"
         ],
         "Resource": "arn:aws:ssm:*:*:parameter/IDT*"
}
```

   上述权限配置为允许使用通配符获取名称以IDT开头的所有参数。*您应该根据自己的需要对其进行自定义，以便 IDT 可以根据您正在使用的参数的命名来获取任何已配置的参数。

2. 你需要将配置值存储在 P AWS aramater Store 中。这可以通过AWS控制台或 AWS CLI 完成。AWS参数存储允许您选择加密或未加密的存储。要存储机密、密码和 PIN 等敏感值，应使用加密选项，该选项的参数类型为 SecureString。要使用 AWS CLI 上传参数，可以使用以下命令：

```
aws ssm put-parameter --name IDT-example-name --value IDT-example-value --type
 SecureString
```

   您可以使用以下命令验证参数是否已存储。（可选）使用该--with-decryption标志获取解密后的参数。SecureString

```
aws ssm get-parameter --name IDT-example-name
```

使用 AWS CLI 将上传当前 CLI 用户AWS所在区域中的参数，IDT 将从中config.json配置的区域获取参数。要AWS通过 CLI 查看您所在的地区，请使用以下命令：

```
aws configure get region
```

3. 在AWS云端拥有配置值后，您可以更新 IDT 配置中的任何值以从AWS云端获取。为此，您可以在表单{{AWS.Parameter.parameter_name}}的 IDT 配置中使用占位符从 Parameter Store 中按该名称获取AWS参数。

   例如，假设您要在 HSM IDT-example-name 配置中使用步骤 2 中的参数作为 HSM keyLabel。为此，您可以按userdata.json如下方式更新您的：

   ```
   "hsm": {
           "keyLabel": "{{AWS.Parameter.IDT-example-name}}",
           [...]
       }
   ```

   IDT 将在运行时获取步骤 2 中设置的此参数IDT-example-value的值。此配置与设置类似，"keyLabel": "IDT-example-value"但相反，该值以加密方式存储在AWS云中。

运行 AWS IoT Greengrass 资格认证套件

设置所需的配置后，就可以开始测试了。完整测试套件的运行时取决于您的硬件。作为参考，在 Raspberry Pi 3B 上完成完整的测试套件大约需要 30 分钟。

使用以下run-suite命令运行一套测试。

```
devicetester_[linux | mac | win]_x86-64 run-suite  \\
    --suite-id suite-id  \\
    --group-id group-id  \\
    --pool-id your-device-pool \\
    --test-id test-id  \\
    --update-idt y|n  \\
    --userdata userdata.json
```

所有选项都是可选的。例如，pool-id如果您的device.json文件中仅定义了一个设备池，即一组相同的设备，则可以省略。或者，如果您要在 tests 文件夹中运行最新的测试套件版本，则可以忽略 suite-id。

Note

如果在线提供了更新的测试套件版本，IDT 会提示您。有关更多信息，请参阅 the section called “测试套件版本”。

运行资格套件的命令示例

以下命令行示例向您展示了如何运行设备池的资格测试。有关 `run-suite` 和其他 IDT 命令的更多信息，请参阅 the section called “IDT 命令”。

使用以下命令运行指定测试套件中的所有测试组。该`list-suites`命令列出了该`tests`文件夹中的测试套件。

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --pool-id <pool-id> \
    --userdata userdata.json
```

使用以下命令在测试套件中运行特定的测试组。该`list-groups`命令列出了测试套件中的测试组。

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --group-id <group-id> \
    --pool-id <pool-id> \
    --userdata userdata.json
```

使用以下命令在测试组中运行特定的测试用例。

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --group-id <group-id> \
    --test-id <test-id> \
    --userdata userdata.json
```

使用以下命令在一个测试组中运行多个测试用例。

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --group-id <group-id> \
    --test-id <test-id1>,<test-id2>
```

```
    --userdata userdata.json
```

使用以下命令列出测试组中的所有测试用例。

```
devicetester_[linux | mac | win]_x86-64 list-test-cases --group-id <group-id>
```

我们建议您运行完整的资格测试套件，该套件按正确的顺序运行测试组依赖关系。如果您选择运行特定的测试组，我们建议您先运行依赖项检查器测试组，以确保在运行相关测试组之前安装了所有 Greengrass 依赖项。例如：

- 在运行核心资格测试组之前运行 `coredependencies`。

AWS IoT GreengrassV2 命令的同义词

IDT 命令位于 *<device-tester-extract-location>*/bin 目录中。要运行测试套件，请按以下格式提供命令：

`help`

列出有关指定命令的信息。

`list-groups`

列出给定测试套件中的组。

`list-suites`

列出可用的测试套件。

`list-supported-products`

列出受支持的产品（在本例中为 AWS IoT Greengrass 版本）和当前 IDT 版本的测试套件版本。

`list-test-cases`

列出给定测试组中的测试用例。支持以下选项：

- `group-id`。要搜索的测试组。此选项是必需的，必须指定单个组。

`run-suite`

对某个设备池运行一组测试。以下是一些受支持的选项：

- `suite-id`. 要运行的测试套件版本。如果未指定，IDT 将使用 `tests` 文件夹中的最新版本。

- group-id。要以逗号分隔的列表形式运行的测试组。如果未指定，IDT 将根据中配置的设置在测试套件中device.json运行所有相应的测试组。IDT 不会根据您配置的设置运行设备不支持的任何测试组，即使这些测试组是在group-id列表中指定的。
- test-id。要以逗号分隔的列表形式运行的测试用例。指定后，group-id 必须指定单个组。
- pool-id. 要测试的设备池。如果您在 device.json 文件中定义了多个设备池，则必须指定一个池。
- stop-on-first-failure。 将 IDT 配置为在第一次出现故障时停止运行。当您要调试指定的测试组group-id时，请将此选项与配合使用。在运行完整测试套件以生成资格认证报告时，请勿使用此选项。
- update-idt。 设置更新 IDT 的提示的响应。如果 IDT 检测到有更新的版本，则Y响应将停止测试执行。N响应会继续执行测试。
- userdata。 包含有关测试工件路径的信息的userdata.json文件的完整路径。此选项是该run-suite命令所必需的。*#userdata.json###### devicetester_extract_location /devicetester_ggv2_ [win|mac|linux] /configs/ ####*

有关 run-suite 选项的更多信息，请使用 help 选项：

```
devicetester_[linux | mac | win]_x86-64 run-suite -h
```

了解结果和日志

本节介绍如何查看和解释 IDT 结果报告和日志。

要解决错误，请参阅对 IDT 进行故障排除AWS IoT GreengrassV2。

查看结果

在运行时，IDT 会将错误写入控制台、日志文件和测试报告中。IDT 在完成资格测试套件后，会生成两个测试报告。这些报告位于*<device-tester-extract-location>*/results/*<execution-id>*/。这两份报告都记录了运行资格测试套件的结果。

awsiotdevicetester_report.xml这是您提交的资格测试报告，AWS用于在设备目录中列出您的AWS Partner设备。该报告包含以下元素：

- IDT 版本。
- 所测试的 AWS IoT Greengrass 版本。
- device.json 文件中指定的 SKU 和设备池名称。

- device.json 文件中指定的设备池的功能。
- 测试结果的摘要汇总。
- 根据设备功能（例如本地资源访问、影子和 MQTT）进行测试的库对测试结果进行细分。

GGV2Q_Result.xml 报告采用 JUnit XML 格式。您可以将它集成到持续集成和开发平台，例如 Jenkins、Bamboo 等。该报告包含以下元素：

- 测试结果的摘要汇总。
- 按照已测试的 AWS IoT Greengrass 功能细分的测试结果。

解析AWS IoT Device Tester结果

awsiotdevicetester_report.xml 或 awsiotdevicetester_report.xml 中的报告部分列出了运行的测试以及结果。

第一个 XML 标签<testsuites>包含测试运行的摘要。例如：

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

<testsuites> 标签中使用的属性

name

测试套件的名称。

time

运行资格认证套件所花时间（以秒为单位）。

tests

运行的测试数量。

failures

已运行但未通过的测试数。

errors

IDT 无法运行的测试数量。

`disabled`

忽略此属性。不会使用。

`awsiotdevicetester_report.xml` 文件包含一个 `<awsproduct>` 标签，其中包含有关正测试的产品以及在运行测试套件后验证的产品功能的信息。

`<awsproduct>` 标签中使用的属性

`name`

所测试的产品的名称。

`version`

所测试的产品的版本。

`features`

验证的功能。标记为 `required` 的功能需要提交您的主板信息以供资格审核。以下代码段演示了此信息在 `awsiotdevicetester_report.xml` 文件中的显示方式。

```
<name="aws-iot-greengrass-v2-core" value="supported" type="required"></feature>
```

如果没有针对所需功能的测试失败或错误，则设备满足运行 AWS IoT Greengrass 的技术要求并可以与 AWS IoT 服务互操作。如果您想在设备目录中列出您的AWS Partner设备，则可以使用此报告作为资格证据。

如果出现测试失败或错误，则可以通过检查 `<testsuites>` XML 标签来确定失败的测试。`<testsuites>` 标签内的 `<testsuite>` XML 标签显示了测试组的测试结果摘要。例如：

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

其格式与 `<testsuites>` 标签类似，但包含一个未使用并可忽略的 `skipped` 属性。在每个`<testsuite>` XML`<testcase>` 标签中，都有为测试组运行的每项测试的标签。例如：

```
<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>>
```

<testcase> 标签中使用的属性

name

测试的名称。

attempts

IDT 运行测试用例的次数。

当测试失败或出现错误时，将会在 <testcase> 标签中添加包含用于故障排除的信息的 <failure> 或 <error> 标签。例如：

```
<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>
```

查看日志

IDT 从运行的测试中生成日志*<devicetester-extract-location>*/results/*<execution-id>*/logs。它会生成两组日志：

test_manager.log

从的测试管理器组件生成的日志AWS IoT Device Tester（例如，与配置、测试排序和报告生成相关的日志）。

<test-case-id>.log (for example, lambdaDeploymentTest.log)

测试组内测试用例的日志，包括来自被测设备的日志。从 IDT v4.2.0 开始，IDT 将每个测试用例的测试日志分组到*<devicetester-extract-location>*/results/*<execution-id>*/logs/*<test-group-id>/*目录中单独的 *<test-case-id >* 文件夹中。

使用 IDT 开发和运行自己的测试套件

从 IDT v4.0.1 开始，IDT for AWS IoT Greengrass V2 将标准化配置设置和结果格式与测试套件环境相结合，使您可以为设备和设备软件开发自定义测试套件。您可以添加自定义测试来用于自己的内部验证，也可以将其提供给客户进行设备验证。

使用 IDT 开发和运行自定义测试套件，如下所示：

开发自定义测试套件

- 使用自定义测试逻辑为要测试的 Greengrass 设备创建测试套件。
- 向 IDT 提供您的自定义测试套件以供测试运行者使用。包括有关测试套件的特定设置配置的信息。

运行自定义测试套件

- 设置要测试的设备。
- 根据要使用的测试套件的要求实现设置配置。
- 使用 IDT 运行您的自定义测试套件。
- 查看 IDT 运行的测试的测试结果和执行日志。

下载最新版本的 fo AWS IoT Device Tester r AWS IoT Greengrass

下载最新版本的 IDT 并将该软件解压缩到文件系统上您拥有读/写权限的位置 (*< device-tester-extract-location >*)。

> **Note**
>
> IDT 不支持由多个用户从共享位置（如 NFS 目录或 Windows 网络共享文件夹）运行。建议您将 IDT 包解压缩到本地驱动器，并在本地工作站上运行 IDT 二进制文件。
> Windows 的路径长度限制为 260 个字符。如果您使用的是 Windows，请将 IDT 提取到根目录（如 `C:\` 或 `D:\`）以使路径长度不超过 260 个字符的限制。

测试套件创建工作流程

测试套件由三种类型的文件组成：

- 为 IDT 提供有关如何运行测试套件的信息的配置文件。
- 测试 IDT 用来运行测试用例的可执行文件。
- 运行测试所需的其他文件。

完成以下基本步骤来创建自定义 IDT 测试：

1. 为测试套件创建配置文件。
2. 创建包含测试套件测试逻辑的测试用例可执行文件。

3. 验证并记录测试运行器运行测试套件所需的配置信息。
4. 验证 IDT 能否按预期运行您的测试套件并生成测试结果。

要快速构建示例自定义套件并运行它，请按照 教程：构建和运行示例 IDT 测试套件 中的说明进行操作。

要开始使用 Python 创建自定义测试套件，请参阅教程：开发一个简单的 IDT 测试套件。

教程：构建和运行示例 IDT 测试套件

AWS IoT Device Tester 下载内容包括示例测试套件的源代码。您可以完成本教程来构建和运行示例测试套件，以了解如何使用 IDT AWS IoT Greengrass 来运行自定义测试套件。

在本教程中，您将完成以下步骤：

1. 构建示例测试套件
2. 使用 IDT 运行示例测试套件

先决条件

要完成本教程，您需要：

- 主机要求
  - 最新版本的 AWS IoT Device Tester
  - Python 3.7 或更高版本

    要检查您计算机安装的 Python 版本，请运行以下命令：

    ```
    python3 --version
    ```

    在 Windows 上，如果运行此命令时返回错误，则可改用 `python --version`。如果返回的版本号为 3.7 或更高版本，则可通过在 Powershell 终端中运行以下命令将 `python3` 设置为 `python` 命令的别名。

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    如果没有返回版本信息，或者版本号小于 3.7，则按照下载 Python 中的说明安装 Python 3.7+。有关更多信息，请参阅 Python 文档。

  - [urllib3](https://urllib3.readthedocs.io/en/latest/)

    要验证 `urllib3` 是否已正确安装，请运行以下命令：

    ```
    python3 -c 'import urllib3'
    ```

    如果未安装 `urllib3`，请运行以下命令进行安装：

    ```
    python3 -m pip install urllib3
    ```

- 设备要求
  - 一种运行 Linux 操作系统的设备，其网络连接到与您主机相同的网络。

    我们建议您使用搭载 Raspberry Pi 操作系统的 Raspberry Pi。请确保您设置 Raspberry Pi 上的 SSH 才能远程连接到它。

配置 IDT 的设备信息

配置您的设备信息，以便 IDT 运行测试。您必须使用以下信息，更新位于 *<device-tester-extract-location>*`/configs` 文件夹中的 `device.json` 模板。

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

```
        }
      }
    ]
  }
]
```

在 devices 对象中，提供以下信息：

id

专属于您设备的用户定义唯一标识符。

connectivity.ip

您设备的 IP 地址。

connectivity.port

可选。用于通过 SSH 连接到您的设备的端口号。

connectivity.auth

连接的身份验证信息。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.auth.method

用于通过给定的连接协议访问设备的身份验证方法。

支持的值为：

- pki
- password

connectivity.auth.credentials

用于身份验证的凭证。

connectivity.auth.credentials.user

用于登录您的设备的用户名。

connectivity.auth.credentials.privKeyPath

用于登录您设备的私有密钥的完整路径。

此值仅在 connectivity.auth.method 设置为 pki 时适用。

`devices.connectivity.auth.credentials.password`

该密码用于登录到您的设备。

此值仅在 `connectivity.auth.method` 设置为 `password` 时适用。

> **Note**
>
> 只有当 `method` 设置为 `pki` 时才指定 `privKeyPath`。
>
> 只有当 `method` 设置为 `password` 时才指定 `password`。

构建示例测试套件

<device-tester-extract-location>/samples/python 文件夹包含示例配置文件、源代码和 IDT 客户端软件开发工具包，您可以使用提供的构建脚本将其组合成一个测试套件。以下目录树显示了这些示例文件的位置：

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

要构建测试套件，请在主机上运行以下命令：

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
```

这将在该 *<device-tester-extract-location>*/tests 文件夹下的 IDTSampleSuitePython_1.0.0 文件夹中创建示例测试套件。查看IDTSampleSuitePython_1.0.0文件夹中的文件以了解示例测试套件的结构，并查看测试用例可执行文件和测试配置 JSON 文件的各种示例。

> **Note**
>
> 示例测试套件包含 python 源代码。请勿在测试套件代码中包含敏感信息。

下一步：使用 IDT 运行您创建的示例测试套件。

使用 IDT 运行示例测试套件

要运行示例测试套件，请在主机上运行以下命令：

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT 会运行示例测试套件，并将结果流式传输到控制台。测试运行完毕后，您会看到以下信息：

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

排查问题

使用以下信息，以帮助解决在完成本教程时遇到的任何问题。

测试用例未成功运行

如果测试运行失败，IDT 会将错误日志流式传输到控制台，以帮助您对测试运行进行故障排除。请确保满足本教程的所有先决条件。

无法连接到被测设备

请验证以下内容：

- 您的 `device.json` 文件包含正确的 IP 地址、端口和身份验证信息。
- 您可以通过 SSH 从主机连接到您的设备。

教程：开发一个简单的 IDT 测试套件

测试套件结合了以下内容：

- 包含测试逻辑的测试可执行文件
- 描述测试套件的配置文件

本教程向您展示如何使用 AWS IoT Greengrass IDT 来开发包含单个测试用例的 Python 测试套件。在本教程中，您将完成以下步骤：

1. 创建测试套件目录
2. 创建配置文件
3. 创建测试用例可执行文件
4. 运行测试套件

先决条件

要完成本教程，您需要：

- 主机要求
  - 最新版本的 AWS IoT Device Tester
  - Python 3.7 或更高版本

    要检查您计算机安装的 Python 版本，请运行以下命令：

    ```
    python3 --version
    ```

    在 Windows 上，如果运行此命令时返回错误，则可改用 `python --version`。如果返回的版本号为 3.7 或更高版本，则可通过在 Powershell 终端中运行以下命令将 `python3` 设置为 `python` 命令的别名。

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    如果没有返回版本信息，或者版本号小于 3.7，则按照下载 Python 中的说明安装 Python 3.7+。有关更多信息，请参阅 Python 文档。
  - urllib3

    要验证 `urllib3` 是否已正确安装，请运行以下命令：

    ```
    python3 -c 'import urllib3'
    ```

    如果未安装 `urllib3`，请运行以下命令进行安装：

    ```
    python3 -m pip install urllib3
    ```
- 设备要求
  - 一种运行 Linux 操作系统的设备，其网络连接到与您主机相同的网络。

    我们建议您使用搭载 Raspberry Pi 操作系统的 Raspberry Pi。请确保您设置 Raspberry Pi 上的 SSH 才能远程连接到它。

创建测试套件目录

IDT 在逻辑上将测试用例分成每个测试套件中的测试组。每个测试用例都必须位于测试组中。在本教程中，创建一个名为 `MyTestSuite_1.0.0` 的文件夹，并在此文件夹中创建以下目录树：

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

创建配置文件

您的测试套件必须包含以下必需的配置文件：

必需配置文件

`suite.json`

包含有关测试套件的信息。请参阅配置 suite.json。

`group.json`

包含有关测试组的信息。您必须为测试套件中的每个测试组创建一个 `group.json` 文件。请参阅配置 group.json。

`test.json`

包含有关测试用例的信息。您必须为测试套件中的每个测试用例创建一个 `test.json` 文件。请参阅配置 test.json。

1. 在 `MyTestSuite_1.0.0/suite` 文件夹中，创建以下文件夹结构的 `suite.json`：

```
{
    "id": "MyTestSuite_1.0.0",
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. 在 `MyTestSuite_1.0.0/myTestGroup` 文件夹中，创建以下文件夹结构的 `group.json`：

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. 在 MyTestSuite_1.0.0/myTestGroup/myTestCase 文件夹中，创建以下文件夹结构的 test.json：

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

您的 MyTestSuite_1.0.0 文件夹应类似于以下内容：

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

获取 IDT 客户端 SDK

您可以使用 IDT 客户端 SDK 让 IDT 与被测设备进行交互并报告测试结果。在本教程中，您将使用 Python 版本的软件开发工具包。

从 *<device-tester-extract-location>*/sdks/python/ 文件夹，将 idt_client 文件夹复制到您的 MyTestSuite_1.0.0/suite/myTestGroup/myTestCase 文件夹。

要验证 SDK 是否复制，可以运行以下命令。

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

创建测试用例可执行文件

测试用例可执行文件包含要运行的测试逻辑。一个测试套件可以包含多个测试用例可执行文件。在本教程中，您将只创建一个测试用例可执行文件。

1. 创建测试套件文件。

   在 MyTestSuite_1.0.0/suite/myTestGroup/myTestCase 文件夹中，创建使用以下内容的 myTestCase.py 文件：

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

   if __name__ == "__main__":
       main()
   ```

2. 使用客户端 SDK 函数将以下测试逻辑添加到您的 myTestCase.py 文件中：

   a. 在被测设备上运行 SSH 命令。

      ```
      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. 将测试结果发送给 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()
```

配置 IDT 的设备信息

配置您的设备信息，以便 IDT 运行测试。您必须使用以下信息，更新位于 *<device-tester-extract-location>*/configs 文件夹中的 device.json 模板。

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

在 devices 对象中，提供以下信息：

id

专属于您设备的用户定义唯一标识符。

connectivity.ip

您设备的 IP 地址。

connectivity.port

可选。用于通过 SSH 连接到您的设备的端口号。

`connectivity.auth`

连接的身份验证信息。

此属性仅在 `connectivity.protocol` 设置为 `ssh` 时适用。

`connectivity.auth.method`

用于通过给定的连接协议访问设备的身份验证方法。

支持的值为：

- `pki`
- `password`

`connectivity.auth.credentials`

用于身份验证的凭证。

`connectivity.auth.credentials.user`

用于登录您的设备的用户名。

`connectivity.auth.credentials.privKeyPath`

用于登录您设备的私有密钥的完整路径。

此值仅在 `connectivity.auth.method` 设置为 `pki` 时适用。

`devices.connectivity.auth.credentials.password`

该密码用于登录到您的设备。

此值仅在 `connectivity.auth.method` 设置为 `password` 时适用。

> **Note**
> 只有当 `method` 设置为 `pki` 时才指定 `privKeyPath`。
> 只有当 `method` 设置为 `password` 时才指定 `password`。

运行测试套件

创建测试套件后，您需要确保其按预期运行。要使用现有设备池运行测试套件，请完成以下步骤。

1. 将您的 MyTestSuite_1.0.0 文件夹复制到 *<device-tester-extract-location>*/tests。
2. 运行以下命令：

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT 会运行您的测试套件，并将结果流式传输到控制台。测试运行完毕后，您会看到以下信息：

```
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
```

排查问题

使用以下信息，以帮助解决在完成本教程时遇到的任何问题。

测试用例未成功运行

如果测试运行失败，IDT 会将错误日志流式传输到控制台，以帮助您对测试运行进行故障排除。检查错误日志之前，请验证以下内容：

- 如本步骤所述，IDT 客户端 SDK 位于正确的文件夹中。
- 您已满足本教程的所有先决条件。

无法连接到被测设备

请验证以下内容：

- 您的 `device.json` 文件包含正确的 IP 地址、端口和身份验证信息。
- 您可以通过 SSH 从主机连接到您的设备。

创建 IDT 测试套件配置文件

本节介绍创建配置文件时使用的格式，您在编写自定义测试套件时包含了这些文件。

必需配置文件

`suite.json`

包含有关测试套件的信息。请参阅配置 suite.json。

`group.json`

包含有关测试组的信息。您必须为测试套件中的每个测试组创建一个 `group.json` 文件。请参阅配置 group.json。

`test.json`

包含有关测试用例的信息。您必须为测试套件中的每个测试用例创建一个 `test.json` 文件。请参阅配置 test.json。

可选配置文件

`test_orchestrator.yaml` 或 `state_machine.json`

定义 IDT 运行测试套件时运行测试的方式。请参阅配置 test_orchestrator.yaml。

> **Note**
>
> 从 IDT v4.5.1 开始，您可以使用该`test_orchestrator.yaml`文件来定义测试工作流程。在以前版本的 IDT 中，请使用 `state_machine.json` 文件。有关状态机的信息，请参阅[配置 IDT 状态机](#)。

`userdata_schema.json`

定义测试运行器可以在其设置配置中包含的[`userdata.json`文件](#)架构。`userdata.json` 文件用于存储运行测试所需但 `device.json` 文件中不存在的任何其他配置信息。请参阅[配置 userdata_schema.json](#)。

配置文件放置在您的 *<custom-test-suite-folder>* 中，如下所示。

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

配置 suite.json

`suite.json` 文件设置环境变量并确定运行测试套件是否需要用户数据。使用以下模板来配置您的 *<custom-test-suite-folder>*`/suite/suite.json` 文件：

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
```

```
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

`id`

测试套件的唯一用户定义 ID。`id` 的值必须与 `suite.json` 文件所在的测试套件文件夹的名称相匹配。套件名称和套件版本还必须满足以下要求：

- *<suite-name>* 不可以包含下划线。
- *<suite-version>* 表示为 *x.x.x*，其中 x 为数字。

ID 显示在 IDT 生成的测试报告中。

`title`

此测试套件正在测试的产品或功能的用户定义名称。该名称显示在 IDT CLI 中供测试运行器使用。

`details`

测试套件用途的简短描述。

`userDataRequired`

定义测试运行器是否需要在 `userdata.json` 文件中包含自定义信息。如果将此值设置为 `true`，还必须将`userdata_schema.json` 文件包含在测试套件文件夹中。

`environmentVariables`

可选。一组要为此测试套件设置的环境变量。

`environmentVariables.key`

环境变量的名称。

`environmentVariables.value`

环境变量的值。

配置 group.json

`group.json` 文件定义测试组是必需的还是可选的。使用以下模板来配置您的 *<custom-test-suite-folder>*/suite/*<test-group>*/group.json 文件：

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

包含值的所有字段都为必填字段，如下所述：

`id`

测试组的唯一用户定义 ID。的值id必须与group.json文件所在的测试组文件夹的名称相匹配，并且不能包含下划线 (_)。ID 用于 IDT 生成的测试报告中。

`title`

测试组的描述性名称。该名称显示在 IDT CLI 中供测试运行器使用。

`details`

测试组用途的简短描述。

`optional`

可选。设置为 `true` 以便在 IDT 完成运行所需测试后，将此测试组显示为可选组。默认值为 `false`。

配置 test.json

`test.json` 文件确定测试用例的可执行文件和测试用例使用的环境变量。有关创建测试用例可执行文件的更多信息，请参阅 创建 IDT 测试用例可执行文件。

使用以下模板来配置您的 *<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.json 文件：

```
{
    "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>",
        }
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

`id`

测试用例的唯一用户定义 ID。的值`id`必须与`test.json`文件所在的测试用例文件夹的名称相匹配，并且不能包含下划线 (_)。ID 用于 IDT 生成的测试报告。

`title`

测试用例的描述性名称。该名称显示在 IDT CLI 中供测试运行器使用。

`details`

测试用例用途的简短描述。

`requireDUT`

可选。如果需要设备才能运行此测试，则设置为 `true`，否则设置为 `false`。默认值为 `true`。测试运行器将在其 `device.json` 文件中配置将用来运行测试的设备。

`requiredResources`

可选。一个阵列，提供有关运行此测试所需资源设备的信息。

`requiredResources.name`

运行此测试时为资源设备提供的唯一名称。

`requiredResources.features`

一系列用户定义的资源设备功能。

`requiredResources.features.name`

功能的名称。您要使用此设备的设备功能。此名称与 `resource.json` 文件中测试运行器提供的功能名称相匹配。

`requiredResources.features.version`

可选。功能的版本。此值与 `resource.json` 文件中测试运行器提供的功能版本相匹配。如果未提供版本，则不检查该功能。如果功能不需要版本号，请将此字段留为空白。

`requiredResources.features.jobSlots`

可选。此功能可以支持的同时测试的数量。默认值为 `1`。如果您希望 IDT 使用不同的设备来实现各项功能，我们建议您将此值设置为 `1`。

`execution.timeout`

IDT 等待测试完成运行的时间长度（以毫秒为单位）。有关设置该值的更多信息，请参阅 创建 IDT 测试用例可执行文件。

`execution.os`

要运行的测试用例可执行文件基于运行 IDT 的主机操作系统。支持的值有`linux`、`mac`和`win`。

`execution.os.cmd`

要在指定操作系统上运行的测试用例可执行文件的路径。此位置必须位于系统路径中。

`execution.os.args`

可选。为运行测试用例可执行文件而提供的参数。

`environmentVariables`

可选。一组要为此测试用例设置的环境变量。

`environmentVariables.key`

环境变量的名称。

`environmentVariables.value`

环境变量的值。

> **Note**
>
> 如果在 `test.json` 文件和 `suite.json` 文件中指定相同的环境变量，则 `test.json` 文件中的值优先。

配置 test_orchestrator.yaml

测试编排工具是一种控制测试套件执行流程的构造。它决定测试套件的起始状态，根据用户定义的规则管理状态转换，并继续在这些状态之间进行转换，直到达到结束状态。

如果您的测试套件不包含用户定义的测试编排工具，IDT 将为您生成一个测试编排工具。

默认测试编排工具执行以下功能：

- 使测试运行器能够选择和运行特定的测试组，而不是整个测试套件。
- 如果未选择特定的测试组，则按随机顺序运行测试套件中的每个测试组。
- 生成报告并打印控制台摘要，其中显示每个测试组和测试用例的测试结果。

有关 IDT 测试编排工具工作原理的更多信息，请参阅[配置 IDT 测试管弦乐队](#)。

配置 userdata_schema.json

`userdata_schema.json` 文件确定了测试运行器提供用户数据的架构。如果您的测试套件需要 `device.json` 文件中不存在的信息，则需要用户数据。例如，您的测试可能需要 Wi-Fi 网络凭证、特定的开放端口或用户必须提供的证书。此信息可提供给 IDT，作为用户在其 *<device-tester-extract-location>*`/config` 文件夹中创建的输入参数，称为 `userdata`，其值是一个 `userdata.json` 文件。`userdata.json` 文件的格式取决于您在测试套件中包含的 `userdata_schema.json` 文件。

要指示测试运行器必须提供 `userdata.json` 文件：

1. 在 `suite.json` 文件中设置 `userDataRequired` 为 `true`。
2. 在您的 *<custom-test-suite-folder>* 中，创建一个 `userdata_schema.json` 文件。
3. 编辑 `userdata_schema.json` 文件以创建有效的 IETF 草稿 v4 JSON 架构。

当 IDT 运行您的测试套件时，它会自动读取架构并使用它来验证测试运行器提供的 `userdata.json` 文件。如果有效，则 `userdata.json` 文件的内容在 IDT 上下文和测试编排工具上下文中均可用。

配置 IDT 测试管弦乐队

从 IDT v4.5.1 开始，IDT 包括一个新的测试编排编排组件。测试管弦乐队是一个 IDT 组件，用于控制测试套件执行流程，并在 IDT 完成运行所有测试后生成测试报告。测试管弦乐队根据用户定义的规则确定测试选择和运行测试的顺序。

如果你的测试套件不包含用户定义的测试管弦乐队，IDT 将为你生成测试管弦乐队。

默认的测试管弦乐队执行以下功能：

- 让测试运行者能够选择和运行特定的测试组，而不是整个测试套件。
- 如果未选择特定的测试组，则以随机顺序运行测试套件中的每个测试组。
- 生成报告并打印显示每个测试组和测试用例的测试结果的控制台摘要。

测试管弦乐队取代了 IDT 测试管弦乐队。我们强烈建议您使用测试管弦乐队来开发测试套件，而不是 IDT 测试管弦乐队。测试管弦乐队提供了以下改进的功能：

- 与 IDT 状态机使用的命令格式相比，使用声明式格式。这使您能够具体说明你想运行哪些测试以及什么时候你想运行它们.

- 管理特定的组处理、报告生成、错误处理和结果跟踪所以你不是必需的以手动管理这些操作。
- 使用 YAML 格式，默认情况下支持注释。
- 需要百分之八十比测试管弦乐队更少的磁盘空间来定义相同的工作流程。
- 添加测试前验证，以验证您的工作流程定义不包含错误的测试 ID 或循环依赖关系。

测试管弦乐队格式

您可以使用以下模板配置自己的模板。*<custom-test-suite-folder>*/suite/test_orchestrator.yamlfile:

```
Aliases:
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
        - test-descriptor
    OneOfTests:
        - test-descriptor
    IsRequired: boolean
```

包含值的所有字段都为必填字段，如下所述：

`Aliases`

可选。映射到上下文表达式的自定义字符串 别名允许您生成友好名称在测试管弦乐队配置中识别上下文表达式。如果您正在创建复杂的上下文表达式或多个位置使用的表达式，这将特别有用。

您可以使用上下文表达式来存储允许您访问其他 IDT 配置中的数据的上下文查询。有关更多信息，请参阅 在上下文中访问数据。

Example 示例

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

可选。条件列表以及满足每个条件时运行的相应测试用例。每个条件可以有多个测试用例；但是，您只能将给定的测试用例分配给一个条件。

默认情况下，IDT 会运行任何未分配给此列表中条件的测试用例。如果您未指定此部分，IDT 将运行测试套件中的所有测试组。

中的每个项目`ConditionalTests`list 包含以下参数：

`Condition`

一个计算结果为布尔值值。如果评估值为 true，IDT 将运行在`Tests`参数。

`Tests`

测试描述符列表。

每个测试描述符使用测试组 ID 和一个或多个测试用例 ID 来标识要从特定测试组运行的单个测试。测试描述符使用以下格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example 示例

以下示例使用可以将其定义为的通用上下文表达式`Aliases`.

```
ConditionalTests:
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
    - Condition: "{{$aliases.Condition2}}"
      Tests:
```

```
            - GroupId: D
      - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
        Tests:
            - GroupId: C
```

根据定义的条件，IDT 选择测试组如下：

- 如果`Condition1`是的，IDT 在测试组 A、B 和 C 中运行测试
- 如果`Condition2`是的，IDT 在测试组 C 和 D 中运行测试

`Order`

可选。运行测试的顺序。您可以在测试组级别指定测试顺序。如果您没有指定此部分，IDT 将按随机顺序运行所有适用的测试组。的价值`Order`是组描述符列表的列表。你没有列出的任何测试组`Order`，可以与列出的任何其他测试组 parallel 运行。

每个组描述符列表都包含一个或多个组描述符，并标识每个描述符中指定的组的运行顺序。您可以使用以下格式定义单个组描述符：

- *`group-id`*— 现有测试组的组 ID。
- [*`group-id`*, *`group-id`*]— 可以按相对于彼此的任意顺序运行的测试组列表。
- `"*"`— 通配符。这相当于当前组描述符列表中尚未指定的所有测试组的列表。

的值`Order`此外，还必须满足以下要求：

- 您在组描述符中指定的测试组 ID 必须存在于测试套件中。
- 每个组描述符列表必须包含至少一个测试组。
- 每个组描述符列表必须包含唯一的组 ID。您不能在单个组描述符中重复测试组 ID。
- 一个组描述符列表最多可以包含一个通配符组描述符。通配符组描述符必须是列表中的第一个或最后一个项目。

Example 示例

对于包含测试组 A、B、C、D 和 E 的测试套件，以下示例列表显示了指定 IDT 应首先运行测试组 A，然后运行测试组 B，然后按任意顺序运行测试组 C、D 和 E 的不同方法。

- ```
  Order:
      - - A
        - B
        - [C, D, E]
  ```
  ```

- ```
  Order:
      - - A
        - B
        - "*"
  ```
- ```
  Order:
      - - A
        - B
  
      - - B
        - C
  
      - - B
        - D
  
      - - B
        - E
  ```

`Features`

可选。您希望 IDT 添加到`awsiotdevicetester_report.xml`文件。如果您没有指定此部分，IDT 将不会向报告中添加任何商品功能。

产品功能是关于设备可能满足的特定标准的用户定义信息。例如，MQTT 产品功能可以指定设备正确发布 MQTT 消息。In`awsiotdevicetester_report.xml`，产品功能设置为`supported`、`not-supported`，或者根据指定的测试是否通过了自定义的用户定义值。

中的每个项目`Features`list 包含以下参数：

`Name`

功能的名称。

`Value`

可选。您想在报告中使用的自定义值，而不是`supported`. 如果未指定此值，则基于 IDT 将功能值设置为`supported`要么`not-supported`根据测试结果。如果您在不同的条件下测试同一功能，则可以在`Features`列表，IDT 连接受支持的条件的功能值。有关更多信息，请参阅 。

`Condition`

一个计算结果为布尔值值。如果评估值为 true，IDT 会在测试套件运行完成后将该功能添加到测试报告中。如果评估值为 false，则报告中不包括测试。

`Tests`

可选。测试描述符列表。此列表中指定的所有测试都必须通过，才能支持该功能。

此列表中的每个测试描述符使用测试组 ID 和一个或多个测试用例 ID 来标识要从特定测试组运行的单个测试。测试描述符使用以下格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

您必须指定`Tests`要么`OneOfTests`对于中的每个功能Featureslist。

`OneOfTests`

可选。测试描述符列表。必须通过此列表中指定的至少一项测试才能支持该功能。

此列表中的每个测试描述符使用测试组 ID 和一个或多个测试用例 ID 来标识要从特定测试组运行的单个测试。测试描述符使用以下格式：

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

您必须指定`Tests`要么`OneOfTests`对于中的每个功能Featureslist。

`IsRequired`

用于定义测试报告中是否需要该功能的布尔值。原设定值为 `false`。

Example

测试编排编排编排上下文

测试管弦乐队上下文是只读 JSON 文档，其中包含测试管弦乐队在执行期间可用的数据。测试管弦乐队上下文只能从测试管弦乐队访问，其中包含确定测试流程的信息。例如，您可以使用测试运行者在`userdata.json`文件来确定是否需要运行特定测试。

测试编排编排程序上下文使用以下格式：

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
```

```
            <userdata-json-content>
        },
        "config": {
            <config-json-content>
        }
}
```

`pool`

有关为测试运行选择的设备池的信息。对于选定的设备池，此信息将从中定义的相应顶级设备池阵列元素中检索`device.json`文件。

`userData`

中的信息`userdata.json`文件。

`config`

中的信息`config.json`文件。

您可以使用 jsonPath 表示法查询上下文。状态定义中 jsonPath 查询的语法为`{{query}}`. 当您从测试管弦乐队上下文访问数据时，请确保每个值的计算结果为字符串、数字或布尔值.

有关使用 JSONPath 表示法访问上下文中的数据的更多信息，请参阅使用 IDT 上下文.

配置 IDT 状态机

> **Important**
>
> 从 IDT v4.5.1 开始，此状态机已弃用。强烈建议您使用新的测试管弦乐队。有关更多信息，请参阅 配置 IDT 测试管弦乐队。

状态机是控制测试套件执行流程的构造。它确定测试套件的起始状态，根据用户定义的规则管理状态转换，并继续通过这些状态转换直到达结束状态。

如果您的测试套件不包含用户定义的状态机，IDT 将为您生成状态机。默认状态机执行以下功能：

- 让测试运行者能够选择和运行特定的测试组，而不是整个测试套件。
- 如果未选择特定的测试组，则以随机顺序运行测试套件中的每个测试组。
- 生成报告并打印显示每个测试组和测试用例的测试结果的控制台摘要。

IDT 测试套件的状态机必须满足以下条件：

- 每个州都对应于 IDT 要采取的操作，例如运行测试组或产品报告文件。
- 过渡到状态将执行与状态关联的操作。
- 每个州都定义了下一个状态的过渡规则。
- 终止状态必须为`Succeed`要么`Fail`.

状态机格式

您可以使用以下模板配置自己的模板。*<custom-test-suite-folder>*`/suite/state_machine.json`file:

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

包含值的所有字段都为必填字段，如下所述：

`Comment`

状态机的描述。

`StartAt`

IDT 开始运行测试套件的状态名称。的价值`StartAt`必须设置为中列出的状态之一。`States`对象。

`States`

将用户定义的状态名称映射到有效的 IDT 状态的对象。每个州。*state ##*对象包含映射到的有效状态的定义*state ##*.

这些区域有：`States`对象必须包含`Succeed`和`Fail`状态。有关有效状态的信息，请参阅。有效的状态和州定义.

有效的状态和州定义

本节介绍可在 IDT 状态机中使用的所有有效状态的状态的状态定义。以下某些状态支持测试用例级别的配置。但是，我们建议您在测试组级别而不是测试用例级别配置状态转换规则，除非绝对必要。

州定义

- RunTask
- Choice
- Parallel
- 添加产品功能
- 报告
- 日志消息
- 选择组
- Fail
- Succeed

RunTask

这些区域有：`RunTask`状态将从测试套件中定义的测试组中运行测试用例。

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

包含值的所有字段都为必填字段，如下所述：

`Next`

在当前状态下执行操作后，要转换到的状态的名称。

`TestGroup`

可选。要运行的测试组的 ID。如果未指定此值，则 IDT 将运行测试运行者选择的测试组。

`TestCases`

可选。中指定的组中的测试用例 ID 数组`TestGroup`. 基于的价值`TestGroup`和`TestCases`，IDT 确定测试执行行为如下：

- 当这两者时`TestGroup`和`TestCases`被指定，IDT 从测试组运行指定的测试用例。
- 何时`TestCases`已指定但`TestGroup`未指定，IDT 运行指定的测试用例。
- 何时`TestGroup`已指定，但`TestCases`未指定，IDT 将运行指定测试组中的所有测试用例。
- 当两者都不`TestGroup`要么`TestCases`，IDT 将运行测试运行者从 IDT CLI 中选择的测试组中的所有测试用例。要为测试跑步者启用小组选择，您必须同时包括两者`RunTask`和`Choice`在你的状态`state_machine.json`文件。有关其工作方式的示例，请参阅。示例状态机：运行用户选择的测试组.

  有关为测试运行者启用 IDT CLI 命令的更多信息，请参阅。the section called "启用 IDT CLI 命令".

`ResultVar`

要与测试运行结果一起设置的上下文变量的名称。如果您没有为指定值，请勿指定此值。`TestGroup`. IDT 设置您在中定义的变量的值`ResultVar`到`true`要么`false`根据以下内容：

- 如果变量名称为形式*`text_text`*`_passed`，则该值设置为第一个测试组中的所有测试是通过还是已跳过。
- 在所有其他情况下，该值设置为所有测试组中的所有测试是通过还是被跳过。

通常情况下，您将使用`RunTask`状态指定测试组 ID 而不指定单个测试用例 ID，以便 IDT 将运行指定测试组中的所有测试用例。由此状态运行的所有测试用例均以随机顺序 parallel 运行。但是，如果所有测试用例都需要一台设备运行，并且只有一台设备可用，则测试用例将按顺序运行。

错误处理

如果任何指定的测试组或测试用例 ID 无效，则此状态将发出`RunTaskError`执行错误。如果状态遇到执行错误，那么它还会设置`hasExecutionError`状态机上下文中的变量`true`.

Choice

这些区域有：Choice状态允许您根据用户定义的条件动态设置要转换到的下一个状态。

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

`Default`

在中没有定义任何表达式时，要转换为的默认状态。`Choices`可以评估`true`.

`FallthroughOnError`

可选。指定状态在计算表达式时遇到错误时的行为。设置为`true`如果评估导致错误，则想跳过表达式。如果没有匹配表达式，则状态机转换为`Default`状态。如果`FallthroughOnError`值未指定，默认为`false`.

`Choices`

用于确定在当前状态下执行操作后转换到哪个状态的表达式和状态数组。

`Choices.Expression`

计算结果为布尔值的表达式字符串。如果表达式的计算结果为`true`，然后状态机转换为中定义的状态。`Choices.Next`. 表达式字符串从状态机上下文中检索值，然后对它们执行操作以得出布尔值。有关访问状态机上下文的信息，请参阅状态机上下文.

`Choices.Next`

在中定义的表达式时，要转换为的状态的名称。`Choices.Expression`评估为`true`.

错误处理

这些区域有：Choice在以下情况下，状态可能需要错误处理：

- 选择表达式中的某些变量在状态机上下文中不存在。
- 表达式的结果不是布尔值。
- JSON 查找的结果不是字符串、数字或布尔值。

您不能使用`Catch`阻止处理此状态下的错误。如果要在状态机遇到错误时停止执行状态机，则必须设置`FallthroughOnError`到`false`. 但是，我们建议您设置`FallthroughOnError`到`true`，并且根据您的使用情形，执行以下操作之一：

- 如果您正在访问的变量在某些情况下预计不存在，则使用`Default`以及其他`Choices`块以指定下一个状态。
- 如果你正在访问的变量应该始终存在，那么将`Default`状态`Fail`.

Parallel

这些区域有：`Parallel`状态允许你彼此并行定义和运 parallel 新的状态机。

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

`Next`

在当前状态下执行操作后，要转换到的状态的名称。

`Branches`

要运行的状态机定义数组。每个状态机定义必须包含自己的定义`StartAt`、`Succeed`, 和`Fail`状态。此数组中的状态机定义不能引用自己定义之外的状态。

> **Note**
>
> 由于每台分支状态机共享相同的状态机上下文，因此在一个分支中设置变量然后从另一个分支读取这些变量可能会导致意外行为。

这些区域有：Parallel状态只有在运行所有分支状态机后才会移动到下一个状态。需要设备的每个状态都将等到设备可用为止。如果有多台设备可用，则此状态并行运 parallel 来自多个组的测试用例。如果没有足够的设备可用，则测试用例将按顺序运行。由于测试用例在并行运行时以随机顺序运 parallel，因此可能会使用不同的设备来运行同一测试组的测试。

错误处理

确保分支状态机和父状态机都过渡到Fail状态来处理执行错误。

由于分支状态机不会将执行错误传输到父状态机，因此不能使用Catch阻止来处理分支状态机中的执行错误。请改用hasExecutionErrors共享状态机上下文中的值。有关其工作方式的示例，请参阅。示例状态机：parallel 行运行两个测试组.

添加产品功能

这些区域有：AddProductFeatures状态允许您将产品功能添加到awsiotdevicetester_report.xmlIDT 生成的文件。

产品功能是关于设备可能满足的特定标准的用户定义信息。例如，MQTT产品功能可以指定设备正确发布 MQTT 消息。在报告中，产品功能设置为supported、not-supported，或者根据指定的测试是否通过的自定义值。

> **Note**
>
> 这些区域有：AddProductFeatures状态本身不会生成报告。此状态必须过渡到Reportstate以生成报告。

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
```

```
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

Next

在当前状态下执行操作后，要转换到的状态的名称。

Features

一系列产品功能将在awsiotdevicetester_report.xml文件。

Feature

功能的名称

FeatureValue

可选。要在报表中使用的自定义值，而不是supported. 如果未指定此值，则根据测试结果，要素值将设置为supported要么not-supported.

如果你使用自定义值FeatureValue，您可以使用不同的条件测试同一功能，并且 IDT 连接受支持的条件的功能值。例如，以下摘录显示了MyFeature具有两个单独的功能值的功能：

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
```

```
},
...
```

如果两个测试组都通过，则要素值将设置为`first-feature-supported, second-feature-supported`.

`Groups`

可选。测试组 ID 的数组。每个指定测试组中的所有测试都必须通过，才能支持该功能。

`OneOfGroups`

可选。测试组 ID 的数组。必须通过至少一个指定测试组中的所有测试才能支持该功能。

`TestCases`

可选。测试用例 ID 的数组。如果指定此值，则以下条件适用：

- 必须通过所有指定的测试用例才能支持该功能。
- `Groups`必须只包含一个测试组 ID。
- `OneOfGroups`必须指定不能指定。

`IsRequired`

可选。设置为`false`在报告中将此功能标记为可选功能。原设定值为 `true`。

`ExecutionMethods`

可选。一组与`protocol`中指定的值`device.json`文件。如果指定了此值，则测试运行者必须指定`protocol`值，该值与此数组中的其中一个值相匹配，以便在报表中包括该功能。如果未指定此值，则该功能将始终包含在报告中。

使用`AddProductFeatures`状态，你必须设置的值`ResultVar`中的`RunTask`状态为以下值之一：

- 如果您指定了单个测试用例 ID，那么请设置`ResultVar`到*group-id_test-id*_passed.
- 如果你没有指定单个测试用例 ID，那么设置`ResultVar`到*group-id*_passed.

这些区域有：`AddProductFeatures`按以下方式检查测试结果：

- 如果您没有指定任何测试用例 ID，则每个测试组的结果将根据*group-id*_passed状态机上下文中的变量。

- 如果您确实指定了测试用例 ID，则每个测试的结果将根据*group-id_test-id*_passed状态机上下文中的变量。

错误处理

如果在此状态下提供的组 ID 不是有效的组 ID，则此状态将导致AddProductFeaturesError执行错误。如果状态遇到执行错误，那么它还会设置hasExecutionErrors状态机上下文中的变量true.

报告

这些区域有：Report状态机*suite-name*_Report.xml和awsiotdevicetester_report.xml文件。此状态还会将报告流式传输到控制台。

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

包含值的所有字段都为必填字段，如下所述：

Next

在当前状态下执行操作后，要转换到的状态的名称。

你应该始终过渡到Report状态接近测试执行流程结束，以便测试运行者可以查看测试结果。通常，此状态之后的下一个状态是Succeed.

错误处理

如果此状态在生成报告时遇到问题，则会发出ReportError执行错误。

日志消息

这些区域有：LogMessage状态机test_manager.log将日志消息传输到控制台。

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

包含值的所有字段都为必填字段，如下所述：

`Next`

在当前状态下执行操作后，要转换到的状态的名称。

`Level`

创建日志消息的错误级别。如果指定的级别无效，则此状态将生成错误消息并将其丢弃。

`Message`

要记录的消息。

选择组

这些区域有：`SelectGroup`状态会更新状态机上下文以指示选择了哪些组。此状态设置的值将被任何后续的任何人使用`Choice`状态。

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

包含值的所有字段都为必填字段，如下所述：

`Next`

在当前状态下执行操作后，要转换到的状态的名称。

`TestGroups`

将标记为选定的测试组的数组。对于此阵列中的每个测试组 ID，*`group-id`*`_selected`变量设置为`true`在上下文中。确保您提供了有效的测试组 ID，因为 IDT 不会验证指定的组是否存在。

Fail

这些区域有：`Fail`状态表示状态机未正确执行。这是状态机的结束状态，每个状态机定义都必须包含此状态。

```
{
    "Type": "Fail"
}
```

Succeed

这些区域有：Succeed状态表示状态机正确执行。这是状态机的结束状态，每个状态机定义都必须包含此状态。

```
{
    "Type": "Succeed"
}
```

状态机上下文

状态机上下文是一个只读 JSON 文档，其中包含在执行过程中状态机可用的数据。状态机上下文只能从状态机访问，其中包含确定测试流程的信息。例如，您可以使用测试运行者在userdata.json文件来确定是否需要运行特定测试。

状态机上下文采用以下格式：

```
{
    "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`

有关为测试运行选择的设备池的信息。对于选定的设备池，此信息将从中定义的相应顶级设备池阵列元素中检索`device.json`文件。

`userData`

中的信息`userdata.json`文件。

`config`

信息固定`config.json`文件。

`suiteFailed`

该值设置为`false`状态机启动的时间。如果测试组在`RunTask`状态，那么该值被设置为`true`在状态机执行的剩余时间内。

`specificTestGroups`

如果测试运行者选择要运行的特定测试组而不是整个测试套件，则会创建此密钥并包含特定测试组 ID 的列表。

`specificTestCases`

如果测试运行者选择要运行的特定测试用例而不是整个测试套件，则会创建此密钥并包含特定测试用例 ID 的列表。

`hasExecutionErrors`

状态机启动时不退出。如果任何状态遇到执行错误，则创建此变量并将其设置为`true`在状态机执行的剩余时间内。

您可以使用 jsonPath 表示法查询上下文。状态定义中的 jsonPath 查询的语法为`{{$.query}}`. 您可以在某些州内使用 jsonPath 查询作为占位符字符串。IDT 将占位符字符串替换为上下文中已评估的 jsonPath 查询的值。您可以对以下值使用占位符：

- 这些区域有：`TestCases`中的值`RunTask`状态。
- 这些区域有：`Expression`值`Choice`状态。

在访问状态机上下文中的数据时，请确保满足以下条件：

- 您的 JSON 路径必须以开头`$.`
- 每个值必须计算为字符串、数字或布尔值。

有关使用 JSONPath 表示法访问上下文中的数据的更多信息，请参阅。使用 IDT 上下文.

执行错误

执行错误是状态机定义中的错误，状态机在执行状态时遇到的错误。IDT 将有关每个错误的信息记录在test_manager.log将日志消息传输到控制台。

您可以使用以下方法处理执行错误：

- 添加Catch街区在州定义中。
- 检查hasExecutionErrors值在状态机上下文中。

抓取

使用Catch中，将以下内容添加到您的州定义中：

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

包含值的所有字段都为必填字段，如下所述：

`Catch.ErrorEquals`

要 catch 的错误类型的数组。如果执行错误与指定的值之一匹配，则状态机转换为中指定的状态。Catch.Next. 有关它产生的错误类型的信息，请参阅每个状态定义。

`Catch.Next`

如果当前状态遇到与中指定的值之一匹配的执行错误，则要转换到的下一个状态Catch.ErrorEquals.

捕获块按顺序处理，直到一次匹配。如果没有错误与 Catch 块中列出的错误匹配，则状态机将继续执行。由于执行错误是不正确的状态定义造成的，因此我们建议您在遇到执行错误时转换为 “失败” 状态。

有执行错误

当某些州遇到执行错误时，除了发出错误之外，它们还设置hasExecutionError值true在状态机上下文中。您可以使用此值来检测何时发生错误，然后使用Choice状态以将状态机转换为Fail状态。

该方法具有以下特征。

- 状态机不会以分配给的任何值开始hasExecutionError，而且在特定状态设置之前，此值才可用。这意味着您必须明确设置FallthroughOnError到false(对于)Choice状态，访问此值以防止状态机在没有发生执行错误时停止。
- 一旦它被设置为true、hasExecutionError永远不会设置为 false 或从上下文中删除。这意味着该值只有在第一次设置为时才有用true，对于后续的所有状态，它并不能提供有意义的价值。
- 这些区域有：hasExecutionError值与中的所有分支状态机共享Parallel状态，这可能会导致意外的结果，具体取决于访问它的顺序。

由于这些特征，如果你可以改用 Catch 块，我们建议你不要使用此方法。

示例状态机

本章节提供了状态机配置示例。

示例

- 示例状态机：运行单个测试组
- 示例状态机：运行用户选择的测试组
- 示例状态机：运行具有产品功能的单个测试组
- 示例状态机：parallel 行运行两个测试组

示例状态机：运行单个测试组

状态机：

- 使用 id 运行测试组GroupA，必须在套件中存在group.json文件。
- 检查执行错误并过渡到Fail如果找到了。
- 生成报告并过渡到Succeed如果没有错误，以及Fail否则为。

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

示例状态机：运行用户选择的测试组

状态机：

- 检查测试运行者是否选择了特定的测试组。状态机不会检查特定的测试用例，因为测试运行者不能在不选择测试组的情况下选择测试用例。
- 如果选择测试组：

- 在选定的测试组内运行测试用例。为此，状态机不会在RunTask状态。
- 运行所有测试并退出后生成报告。
- 如果未选择测试组：
  - 在测试组中运行测试GroupA.
  - 生成报告并退出。

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

示例状态机：运行具有产品功能的单个测试组

状态机：

- 运行测试组GroupA.
- 检查执行错误并过渡到Fail如果找到了。
- 添加FeatureThatDependsOnGroupA功能awsiotdevicetester_report.xmlfile:
  - 如果GroupA通行证时，要素设置为supported.
  - 该功能在报告中未标记为可选项。
- 生成报告并过渡到Succeed如果没有错误，以及Fail否则

```
{
```

```
    "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"
        }
    }
}
```

示例状态机：parallel 行运行两个测试组

状态机：

- 运行GroupA和GroupB并行测试 parallel。这些区域有：ResultVar存储在上下文中的变量RunTask分支状态机中的状态可供AddProductFeatures状态。
- 检查执行错误并过渡到Fail如果找到了。此状态机不使用Catch阻止因为该方法无法检测到分支状态机中的执行错误。
- 将功能添加到awsiotdevicetester_report.xml基于通过的组的文件
  - 如果GroupA通行证时，要素设置为supported.
  - 该功能在报告中未标记为可选项。
- 生成报告并过渡到Succeed如果没有错误，以及Fail否则

如果在设备池中配置了两台设备，则两者都配置GroupA和GroupB可以同时运行。但是，如果任何一个GroupA要么GroupB其中有多个测试，那么两台设备都可以分配给这些测试。如果只配置了一台设备，测试组将按顺序运行。

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

创建 IDT 测试用例可执行文件

您可以通过以下方式创建测试用例可执行文件并将其放置在测试套件文件夹中：

- 对于使用`test.json`文件中的参数或环境变量来确定要运行哪些测试的测试套件，您可以为整个测试套件创建单个测试用例可执行文件，也可以为测试套件中的每个测试组创建一个测试可执行文件。
- 对于要根据指定命令运行特定测试的测试套件，您可以为测试套件中的每个测试用例创建一个可执行的测试用例。

作为测试编写者，您可以确定哪种方法适合您的用例，并相应地构建您的测试用例可执行文件。确保在每个`test.json`文件中提供正确的测试用例可执行文件路径，并且指定的可执行文件正确运行。

当所有设备都准备好运行测试用例时，IDT 会读取以下文件：

- 所选测试`test.json`用例的决定了要启动的进程和要设置的环境变量。
- 测试套件`suite.json`的决定了要设置的环境变量。

IDT 根据`test.json`文件中指定的命令和参数启动所需的测试可执行进程，并将所需的环境变量传递给该进程。

使用 IDT 客户端 SDK

IDT Client SDK 允许您使用 API 命令来简化在测试可执行文件中编写测试逻辑的方式，您可以使用这些命令与 IDT 和被测设备进行交互。IDT 目前提供以下 SDK：

- 适用于Python的 IDT Client
- 适用于 Go 的 IDT Client DK
- 适用于 Java 的 IDT Client S

这些 SDK 位于该*<device-tester-extract-location>*/sdks文件夹中。创建新的测试用例可执行文件时，必须将要使用的 SDK 复制到包含测试用例可执行文件的文件夹，并在代码中引用 SDK。本节简要描述了可以在测试用例可执行文件中使用的可用 API 命令。

本节内容

- 设备互动
- IDT 互动
- 主持人互动

设备互动

以下命令使您无需实现任何额外的设备交互和连接管理功能，即可与被测设备通信。

`ExecuteOnDevice`

允许测试套件在支持 SSH 或 Docker shell 连接的设备上运行 shell 命令。

`CopyToDevice`

允许测试套件将本地文件从运行 IDT 的主机复制到支持 SSH 或 Docker shell 连接的设备上的指定位置。

`ReadFromDevice`

允许测试套件从支持 UART 连接的设备的串行端口读取。

> **Note**
>
> 由于 IDT 不管理使用上下文中的设备访问信息与设备的直接连接，因此我们建议在测试用例可执行文件中使用这些设备交互 API 命令。但是，如果这些命令不符合您的测试用例要求，则可以从 IDT 上下文中检索设备访问信息，并使用这些信息从测试套件直接连接到设备。
>
> 要建立直接连接，请分别在`device.connectivity`和`resource.devices.connectivity`字段中检索被测设备和资源设备的信息。有关使用 IDT 上下文的更多信息，请参阅使用 IDT 上下文。

IDT 互动

以下命令使您的测试套件能够与 IDT 通信。

`PollForNotifications`

允许测试套件检查来自 IDT 的通知。

`GetContextValue` 和 `GetContextString`

允许测试套件从 IDT 上下文检索值。有关更多信息，请参阅使用 IDT 上下文：

`SendResult`

允许测试套件向 IDT 报告测试用例结果。必须在测试套件中每个测试用例结束时调用此命令。

主持人互动

以下命令使您的测试套件能够与主机通信。

`PollForNotifications`

允许测试套件检查来自 IDT 的通知。

`GetContextValue` 和 `GetContextString`

允许测试套件从 IDT 上下文检索值。有关更多信息，请参阅使用 IDT 上下文：

`ExecuteOnHost`

允许测试套件在本地计算机上运行命令，并让 IDT 管理测试用例可执行生命周期。

启用 IDT CLI 命令

`run-suite`命令 IDT CLI 提供了多个选项，允许测试运行器自定义测试执行。要允许测试运行者使用这些选项来运行您的自定义测试套件，您需要实现对 IDT CLI 的支持。如果您不实现支持，测试运行器仍然可以运行测试，但某些 CLI 选项将无法正常运行。为了提供理想的客户体验，我们建议您在 IDT CLI 中实现对`run-suite`命令以下参数的支持：

`timeout-multiplier`

指定一个大于 1.0 的值，该值将应用于运行测试时的所有超时。

测试运行者可以使用此参数来延长他们想要运行的测试用例的超时时间。当测试运行器在其`run-suite`命令中指定此参数时，IDT 会使用它来计算 IDT_TEST_TIMEOUT 环境变量的值并在 IDT 上下文中设置该`config.timeoutMultiplier`字段。要支持这个论点，您必须执行以下操作：

- 与其直接使用`test.json`文件中的超时值，不如读取 IDT_TEST_TIMEOUT 环境变量以获得正确计算的超时值。
- 从 IDT 上下文中检索该`config.timeoutMultiplier`值并将其应用于长时间运行的超时。

有关因超时事件而提前退出的更多信息，请参阅指定退出行为。

`stop-on-first-failure`

指定 IDT 在遇到失败时应停止运行所有测试。

当测试运行器在其`run-suite`命令中指定此参数时，IDT 将在遇到故障后立即停止运行测试。但是，如果测试用例parallel 运行，则可能导致意想不到的结果。要实现支持，请确保如果 IDT 遇到此事件，您的测试逻辑会指示所有正在运行的测试用例停止，清理临时资源并将测试结果报告给 IDT。有关在失败时尽早退出的更多信息，请参阅指定退出行为。

`group-id` 和 `test-id`

指定 IDT 应仅运行选定的测试组或测试用例。

测试运行者可以在`run-suite`命令中使用这些参数来指定以下测试执行行为：

- 在指定的测试组内运行所有测试。
- 在指定的测试组中运行一系列测试。

要支持这些论点，您的测试套件的测试编排器必须在测试编排器中包含一组特定的`RunTask`和`Choice`状态。如果您没有使用自定义状态机，则默认 IDT test orchestrator 会包含您所需的状态，您无需采取其他操作。但是，如果您使用的是自定义测试编排器，则可以将其用示例状态机：运行用户选择的测试组作示例，在测试编排器中添加所需的状态。

有关 IDT CLI 命令的更多信息，请参阅调试和运行自定义测试套件。

写入事件日志

在测试运行时，您可以向控制台发送数据`stderr`，`stdout`并将事件日志和错误消息写入控制台。有关控制台消息格式的信息，请参阅控制台消息格式。

当 IDT 完成测试套件的运行后，该信息也可以在位于该`*<devicetester-extract-location>*/results/*<execution-id>*/logs`文件夹`test_manager.log`的文件中找到。

您可以将每个测试用例配置为将其测试运行的日志（包括来自被测设备的日志）写入位于该`*<device-tester-extract-location>*/results/*execution-id*/logs`文件夹中的`*<group-*`

*id>*_*<test-id>*文件中。为此，请使用testData.logFilePath查询从 IDT 上下文中检索日志文件的路径，在该路径上创建一个文件，然后将所需的内容写入其中。IDT 会根据正在运行的测试用例自动更新路径。如果您选择不为测试用例创建日志文件，则不会为该测试用例生成任何文件。

您也可以设置文本可执行文件，以便根据需要在*<device-tester-extract-location>*/logs文件夹中创建其他日志文件。我们建议您为日志文件名指定唯一前缀，这样您的文件就不会被覆盖。

向 IDT 报告结果

IDT 将测试结果写入awsiotdevicetester_report.xml和*suite-name*_report.xml文件。这些报告文件位于*<device-tester-extract-location>*/results/*<execution-id>*/。这两份报告都捕获了测试套件执行的结果。有关 IDT 用于这些报告的架构的更多信息，请参见查看 IDT 测试结果和日志

要填充*suite-name*_report.xml文件内容，必须在测试执行完成之前使用SendResult命令将测试结果报告给 IDT。如果 IDT 无法找到测试结果，则会为测试用例发出错误。以下 Python 摘录显示了将测试结果发送到 IDT 的命令：

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

如果您不通过 API 报告结果，IDT 会在测试工件文件夹中查找测试结果。此文件夹的路径存储在testData.testArtifactsPath IDT 上下文中的文件中。在此文件夹中，IDT 使用它找到的第一个按字母顺序排序的 XML 文件作为测试结果。

如果您的测试逻辑生成 JUnit XML 结果，则可以将测试结果写入工件文件夹中的 XML 文件中，直接将结果提供给 IDT，而不是解析结果然后使用 API 将其提交给 IDT。

如果您使用此方法，请确保您的测试逻辑准确地总结了测试结果，并将结果文件格式化为与*suite-name*_report.xml文件相同的格式。IDT 不会对您提供的数据进行任何验证，但以下情况除外：

- IDT 会忽略testsuites标签的所有属性。相反，它根据其他报告的测试组结果计算标签属性。
- 其中必须至少存在一个testsuite标签testsuites。

由于 IDT 对所有测试用例使用相同的构件文件夹，并且不会在两次测试运行之间删除结果文件，因此，如果 IDT 读取了错误的文件，此方法也可能导致错误报告。我们建议您在所有测试用例中为生成的 XML 结果文件使用相同的名称，以覆盖每个测试用例的结果，并确保 IDT 可以使用正确的结果。尽管您可以在测试套件中使用混合方法进行报告，即对某些测试用例使用 XML 结果文件，为其他测试用例通过 API 提交结果，但我们不建议使用这种方法。

指定退出行为

将您的文本可执行文件配置为始终以退出代码为 0 退出，即使测试用例报告失败或错误结果也是如此。仅使用非零退出代码来表示测试用例未运行，或者测试用例可执行文件无法向 IDT 传达任何结果。当 IDT 收到非零的退出代码时，它标志着测试用例遇到了阻止其运行的错误。

在以下事件中，IDT 可能会请求或期望测试用例在完成之前停止运行。使用此信息配置您的测试用例可执行文件，以检测测试用例中的每个事件：

超时

当测试用例的运行时间超过`test.json`文件中指定的超时值时发生。如果测试运行器使用`timeout-multiplier`参数指定超时乘数，则 IDT 使用该乘数计算超时值。

要检测此事件，请使用 IDT_TEST_TIMEOUT 环境变量。当测试运行器启动测试时，IDT 将 IDT_TEST_TIMEOUT 环境变量的值设置为计算出的超时值（以秒为单位），并将该变量传递给测试用例可执行文件。您可以读取变量值来设置适当的计时器。

打断

在测试运行器中断 IDT 时发生。例如，按Ctrl+C。

由于终端将信号传播到所有子进程，因此您只需在测试用例中配置信号处理程序即可检测中断信号。

或者，您可以定期轮询 API 以检查`PollForNotifications` API 响应中`CancellationRequested`布尔值的值。当 IDT 收到中断信号时，它会将`CancellationRequested`布尔值设置为`true`。

第一次失败时停止

当与当前测试用例parallel 运行的测试用例失败且测试运行器使用`stop-on-first-failure`参数指定 IDT 在遇到任何失败时应停止时发生。

要检测此事件，您可以定期轮询 API 以检查`PollForNotifications` API 响应中`CancellationRequested`布尔值的值。当 IDT 遇到故障并配置为在第一次失败时停止时，它将`CancellationRequested`布尔值设置为`true`。

当其中任何一个事件发生时，IDT 会等待 5 分钟，直到所有当前正在运行的测试用例完成运行。如果所有正在运行的测试用例均未在 5 分钟内退出，IDT 会强制其每个进程停止。如果 IDT 在进程结束之前没有收到测试结果，它会将测试用例标记为已超时。作为最佳实践，您应确保您的测试用例在遇到其中一个事件时执行以下操作：

1. 停止运行正常的测试逻辑。
2. 清理所有临时资源，例如被测设备上的测试工件。
3. 向 IDT 报告测试结果，例如测试失败或错误。
4. 退出。

使用 IDT 上下文

当 IDT 运行测试套件时，测试套件可以访问一组数据，这些数据可用于确定每个测试的运行方式。这些数据称为 IDT 上下文。例如，测试运行者在`userdata.json`文件可供 IDT 上下文中的测试套件使用。

IDT 上下文可以被视为只读 JSON 文档。测试套件可以使用标准的 JSON 数据类型（如对象、数组、数字等）从上下文中检索数据并将数据写入上下文。

上下文架构

IDT 上下文使用以下格式：

```
{
    "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`

来自`config.json`文件. 这些区域有：config字段还包含以下附加字段：

`config.timeoutMultiplier`

测试套件使用的任何超时值的乘数。该值由 IDT CLI 中的测试运行器指定。原设定值为 1。

`device`

有关为测试运行选择的设备的信息。此信息等效于devices中的数组元素`device.json`文件对于所选设备。

`devicePool`

有关为测试运行选择的设备池的信息。此信息等同于中定义的顶级设备池阵列元素device.json所选设备池的文件。

`resource`

有关资源设备的信息resource.json文件。

`resource.devices`

此信息等效于devices数组在resource.json文件。EADdevices元素包括以下附加字段：

`resource.device.name`

资源设备的名称。此值设置为requiredResource.name中的值test.json文件。

`testData.awsCredentials`

这些区域有：AWS测试用于连接到AWS云。此信息来自config.json文件。

`testData.logFilePath`

测试用例将日志消息写入的日志文件的路径。如果此文件不存在，则测试套件将创建它。

`userData`

测试运行者在[userdata.json文件](#).

在上下文中访问数据

您可以使用 JSON 文件和文本可执行文本中的 jsonPath 表示法查询上下文`GetContextValue`和`GetContextString`API。JsonPath 字符串访问 IDT 上下文的语法如下所示：

- In`suite.json`和`test.json`，您使用`{{query}}`. 也就是说，不要使用根元素`$.`开始你的表情。
- In`test_orchestrator.yaml`，您使用`{{query}}`.

  如果你使用已弃用的状态机，那么`state_machine.json`，您使用`{{$.query}}`.
- 在 API 命令中，你使用`query`要么`{{$.query}}`，取决于命令。有关更多信息，请参阅开发工具包中的内联文档。

下表介绍了典型 JSONPath 表达式中的运算符：

Operator	Description
`$`	The root element. Because the top-level context value for IDT is an object, you will typically use `$.` to start your queries.
`.childname`	Accesses the child element with name ### from an object. If applied to an array, yields a new array with this operator applied to each element. The element name is case sensitive. For example, the query to access the `awsRegion` value in the `config` object is `$.config.aws ##` .
`[##:##]`	Filters elements from an array, retrieving items beginning from the ## index and going up to the ## index, both inclusive.

Operator	Description
`[index1#index 2#...#indexN]`	Filters elements from an array, retrieving items from only the specified indices.
`[# (expr)]`	Filters elements from an array using the `expr` expression. This expression must evaluate to a boolean value.

要创建过滤器表达式，请使用以下语法：

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

在此语法中：

- `jsonpath`是一个使用标准 JSON 语法的 jsonPath。
- `value`是使用标准 JSON 语法的任何自定义值。
- `operator`是以下操作员之一：
  - <（小于）
  - <=（小于或等于）
  - ==（等于）

    如果表达式中的 jsonPath 或值是数组、布尔值或对象值，那么这是唯一可以使用的受支持的二进制运算符。
  - >=（大于或等于）
  - >（大于）
  - =~（正则表达式匹配）。要在过滤器表达式中使用此运算符，表达式左侧的 jsonPath 或值必须计算为字符串，右侧必须是跟随[RE2 语法](https://github.com/google/re2/wiki/Syntax).

你可以使用表格 {{{*##*} 作为占位符字符串在`args`和`environmentVariables`字段`test.json`文件和`environmentVariables`字段`suite.json`文件。IDT 执行上下文查找，并使用查询的评估值填充字段。例如，在`suite.json`文件中，您可以使用占位符字符串来指定随每个测试用例而更改的环境变量值，IDT 将使用每个测试用例的正确值填充环境变量。但是，当你在中使用占位符字符串时`test.json`和`suite.json`文件，请注意以下几点：

- 你必须每次出现devicePool在你的查询中都是小写字母。也就是说，使用devicepool相反。
- 对于数组，你只能使用字符串数组。此外，阵列使用非标准item1, item2,...,itemN格式的日期和时间。如果数组只包含一个元素，那么它将被序列化为item，使其与字符串字段无法区分。
- 不能使用占位符从上下文中检索对象。

出于这些考虑因素，我们建议尽可能使用 API 访问测试逻辑中的上下文，而不是中的占位符字符串test.json和suite.json文件。但是，在某些情况下，使用 jsonPath 占位符检索单个字符串以设置为环境变量可能更方便。

为测试运行者配置设置

要运行自定义测试套件，测试运行器必须根据他们要运行的测试套件配置设置。设置是根据位于该 *<device-tester-extract-location>*/configs/ 文件夹中的配置文件模板指定的。如果需要，测试运行器还必须设置 IDT 用于连接 AWS 云的 AWS 凭证。

作为测试编写者，您需要配置这些文件来调试您的测试套件。您必须向测试运行器提供说明，以便他们可以根据需要配置以下设置来运行您的测试套件。

配置 device.json

device.json 文件包含有关运行测试的设备的信息（例如，IP 地址、登录信息、操作系统和 CPU 架构）。

测试运行器可以使用位于 *<device-tester-extract-location>*/configs/ 文件夹中的以下模板 device.json 文件来提供此信息。

```
[
    {
        "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>",
                }
            }
        ]
    }
]
```

包含值的所有字段都为必填字段，如下所述：

`id`

一个用户定义的字母数字 ID，用于唯一地标识称作设备池的设备集合。属于池的设备必须具有相同的硬件。运行一组测试时，池中的设备将用于对工作负载进行并行化处理。多个设备用于运行不同测试。

`sku`

唯一标识所测试设备的字母数字值。该 SKU 用于跟踪符合条件的设备。

Note

如果需要在 AWS Partner 设备目录中列出您的主板，在此处指定的 SKU 必须与在列表过程中使用的 SKU 相匹配。

`features`

可选。包含设备支持的功能的数组。设备功能是您在测试套件中配置的用户定义值。您必须向测试运行器提供有关要包含在 `device.json` 文件中的功能名称和值的信息。例如，如果您想测试一台可充当其他设备的 MQTT 服务器的设备，则可以配置测试逻辑来验证名为 `MQTT_QOS` 的功能的特定支持级别。测试运行器提供此功能名称，并将该功能值设置为其设备支持的 QOS 级别。您可以通过查询从 IDT 上下文中检索所提供的信息，也可以通过`devicePool.features`查询从测试协调器上下文中检索所提供的信息。`pool.features`

`features.name`

特征的名称。

`features.value`

支持的功能值。

`features.configs`

该功能的配置设置（如果需要）。

`features.config.name`

配置设置的名称。

`features.config.value`

支持的设置值。

`devices`

池中待测试的设备阵列。至少需要选择一个设备。

`devices.id`

用户定义的测试的设备的唯一标识符。

`connectivity.protocol`

用于与此设备通信的通信协议。池中的每台设备都必须使用相同的协议。

目前，唯一支持的值，对于物理设备为 ssh 和 uart，对于 Docker 容器为 docker。

connectivity.ip

测试的设备 IP 地址。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.port

可选。用于 SSH 连接的端口号。

默认值为 22。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.auth

连接的身份验证信息。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.auth.method

用于通过给定的连接协议访问设备的身份验证方法。

支持的值为：

- pki
- password

connectivity.auth.credentials

用于身份验证的凭证。

connectivity.auth.credentials.password

该密码用于登录到正在测试的设备。

此值仅在 connectivity.auth.method 设置为 password 时适用。

connectivity.auth.credentials.privKeyPath

用于登录所测试设备的私有密钥的完整路径。

此值仅在 connectivity.auth.method 设置为 pki 时适用。

connectivity.auth.credentials.user

用于登录所测试设备的用户名。

connectivity.serialPort

可选。设备所连接的串行端口。

此属性仅在 connectivity.protocol 设置为 uart 时适用。

connectivity.containerId

所测试的 Docker 容器的容器 ID 或名称。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.containerUser

可选。容器内用户对用户的名称。默认值为 Dockerfile 中提供的用户。

默认值为 22。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

> **Note**
>
> 要检查测试运行者是否为测试配置了错误的设备连接，您可以pool.Devices[0].Connectivity.Protocol从测试协调器上下文中进行检索，并将其与状态下的预期值进行比较。Choice如果使用的协议不正确，则使用 LogMessage 状态打印一条消息并过渡到 Fail 状态。
> 或者，您可以使用错误处理代码来报告错误设备类型的测试失败。

（可选）配置 userdata.json

userdata.json 文件包含测试套件所需但 device.json 文件中未指定的任何其他信息。此文件的格式取决于测试套件中定义的userdata_scheme.json文件。如果您是测试编写者，请务必将此信息提供给将运行您编写的测试套件的用户。

（可选）配置 resource.json

resource.json 文件包含有关将用作资源设备的所有设备的信息。资源设备是测试被测设备的某些功能所需的设备。例如，要测试设备的蓝牙功能，您可以使用资源设备来测试您的设备能否成功连接

到该设备。资源设备是可选的，您可以根据需要任意数量的资源设备。作为测试编写者，您可以使用 test.json 文件来定义测试所需的资源设备功能。然后，测试运行器使用 resource.json 文件提供具有所需功能的资源设备池。请务必将此信息提供给将运行您编写的测试套件的用户。

测试运行器可以使用位于 *<device-tester-extract-location>*/configs/ 文件夹中的以下模板 resource.json 文件来提供此信息。

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-version>",
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
```

```
                }
            }
        ]
    }
]
```

包含值的所有字段都为必填字段，如下所述：

`id`

一个用户定义的字母数字 ID，用于唯一地标识称作设备池的设备集合。属于池的设备必须具有相同的硬件。运行一组测试时，池中的设备将用于对工作负载进行并行化处理。多个设备用于运行不同测试。

`features`

可选。包含设备支持的功能的数组。此字段中所需的信息在测试套件的 test.json 文件中定义，用于确定要运行哪些测试以及如何运行这些测试。如果测试套件不需要任何功能，则此字段不是必填字段。

`features.name`

功能的名称。

`features.version`

功能版本。

`features.jobSlots`

用于指明可以同时使用该设备的测试次数的设置。默认值为 `1`。

`devices`

池中待测试的设备阵列。至少需要选择一个设备。

`devices.id`

用户定义的测试的设备的唯一标识符。

`connectivity.protocol`

用于与此设备通信的通信协议。池中的每台设备都必须使用相同的协议。

目前，唯一支持的值，对于物理设备为 `ssh` 和 `uart`，对于 Docker 容器为 `docker`。

`connectivity.ip`

测试的设备 IP 地址。

此属性仅在 `connectivity.protocol` 设置为 `ssh` 时适用。

`connectivity.port`

可选。用于 SSH 连接的端口号。

默认值为 22。

此属性仅在 `connectivity.protocol` 设置为 `ssh` 时适用。

`connectivity.auth`

连接的身份验证信息。

此属性仅在 `connectivity.protocol` 设置为 `ssh` 时适用。

`connectivity.auth.method`

用于通过给定的连接协议访问设备的身份验证方法。

支持的值为：

- `pki`
- `password`

`connectivity.auth.credentials`

用于身份验证的凭证。

`connectivity.auth.credentials.password`

该密码用于登录到正在测试的设备。

此值仅在 `connectivity.auth.method` 设置为 `password` 时适用。

`connectivity.auth.credentials.privKeyPath`

用于登录所测试设备的私有密钥的完整路径。

此值仅在 `connectivity.auth.method` 设置为 `pki` 时适用。

`connectivity.auth.credentials.user`

用于登录所测试设备的用户名。

`connectivity.serialPort`

可选。设备所连接的串行端口。

此属性仅在 connectivity.protocol 设置为 uart 时适用。

connectivity.containerId

所测试的 Docker 容器的容器 ID 或名称。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

connectivity.containerUser

可选。容器内用户对用户的名称。默认值为 Dockerfile 中提供的用户。

默认值为 22。

此属性仅在 connectivity.protocol 设置为 ssh 时适用。

（可选）配置 config.json

config.json 文件包含 IDT 的配置信息。通常，测试运行器无需修改此文件，除非提供他们的 IDT AWS 用户凭证和 AWS 区域（可选）。如果提供了具有所需权限的 AWS 凭证，则 AWS IoT Device Tester 会收集使用情况指标并将其提交给 AWS。这是一项可选功能，用来改进 IDT 功能。有关更多信息，请参阅 ID用。

测试运行器可以通过以下方式之一配置 AWS 凭证：

- 凭证文件

  IDT 使用与 AWS CLI 相同的凭证文件。有关更多信息，请参阅配置和凭证文件。

  凭证文件的位置因您使用的操作系统而异：

  - macOS、Linux：~/.aws/credentials
  - Windows：C:\Users*UserName*\.aws\credentials
- 环境变量

  环境变量是由操作系统维护且由系统命令使用的变量。在 SSH 会话期间定义的变量在该会话关闭后不可用。IDT 可使用 AWS_ACCESS_KEY_ID 和 AWS_SECRET_ACCESS_KEY 环境变量来存储 AWS 凭证

  要在 Linux、macOS 或 Unix 上设置这些变量，请使用 export：

```
export AWS_ACCESS_KEY_ID=<your_access_key_id>
```

```
export AWS_SECRET_ACCESS_KEY=<your_secret_access_key>
```

要在 Windows 上设置这些变量，请使用 set：

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

要配置 IDT 的 AWS 凭证，测试运行器需要编辑 *<device-tester-extract-location>*/configs/ 文件夹中 config.json 文件中的 auth 部分。

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

包含值的所有字段都为必填字段，如下所示：

> **Note**
>
> 此文件中的所有路径都是相对于 *< device-tester-extract-location >* 定义的。

log.location

< device-tester-extract-location > 中日志文件夹的路径。

`configFiles.root`

包含配置文件的文件夹的路径。

`configFiles.device`

`device.json` 文件的路径。

`testPath`

包含测试套件的文件夹的路径。

`reportPath`

IDT 运行测试套件后将包含测试结果的文件夹的路径。

`awsRegion`

可选。测试套件将使用的 AWS 区域。如果未设置，则测试套件将使用每个测试套件中指定的默认区域。

`auth.method`

IDT 用于检索 AWS 凭证的方法。支持的值是用于从凭证文件中检索凭证的 `file`，以及使用环境变量检索凭证的 `environment`。

`auth.credentials.profile`

要从凭证文件中使用的凭证配置文件。此属性仅在 `auth.method` 设置为 `file` 时适用。

调试和运行自定义测试套件

设置所需的配置后，IDT 就可以运行您的测试套件了。完整测试套件的运行时取决于硬件和测试套件的组成。作为参考，在 Raspberry Pi 3B 上完成整套 AWS IoT Greengrass 资格测试大约需要 30 分钟。

在编写测试套件时，您可以使用 IDT 在调试模式下运行测试套件，以便在运行代码之前检查代码或将其提供给测试运行者。

在调试模式下运行 IDT

由于测试套件依赖于 IDT 来与设备交互、提供上下文和接收结果，因此如果没有任何 IDT 交互，您就无法在 IDE 中简单地调试测试套件。为此，IDT CLI 提供了 `debug-test-suite` 命令，供您用于在调试模式下运行 IDT。运行以下命令以查看 `debug-test-suite` 的可用选项：

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

当您在调试模式下运行 IDT 时，IDT 实际上并不启动测试套件或运行编排工具；相反，它会与您的 IDE 交互以响应在 IDE 中运行的测试套件所发出的请求，并将日志输出到控制台。IDT 不会超时，而会等待退出，直到手动中断。在调试模式下，IDT 也不运行测试编排工具，也不会生成任何报告文件。要调试测试套件，您必须使用 IDE 来提供 IDT 通常从配置 JSON 文件中获得的一些信息。务必提供以下信息：

- 每个测试的环境变量和参数。IDT 不会从 `test.json` 或 `suite.json` 中读取此信息。
- 用于选择资源设备的参数。IDT 不会从 `test.json` 中读取此信息。

要调试您的测试套件，请完成以下步骤：

1. 创建运行测试套件所需的设置配置文件。例如，如果您的测试套件需要 `device.json`、`resource.json` 和 `user data.json`，请确保根据需要来配置所有测试套件。
2. 运行以下命令将 IDT 置于调试模式，然后选择运行测试需要的所有设备。

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   运行此命令后，IDT 会等待来自测试套件的请求，然后响应这些请求。IDT 还会生成 IDT 客户端软件开发工具包案例处理所需的环境变量。
3. 在您的 IDE 中，使用 `run` 或 `debug` 配置来执行以下操作：

   a. 设置 IDT 生成的环境变量的值。
   b. 设置您在 `test.json` 和 `suite.json` 文件中指定的任何环境变量或参数的值。
   c. 根据需要设置断点。
4. 在 IDE 中运行测试套件。

   您可以根据需要多次调试和重新运行测试套件。IDT 在调试模式下不会超时。
5. 完成调试后，请中断 IDT 以退出调试模式。

用于运行测试的 IDT CLI 命令

以下小节介绍了 IDT CLI 命令。

IDT v4.0.0

`help`

列出有关指定命令的信息。

`list-groups`

列出给定测试套件中的组。

`list-suites`

列出可用的测试套件。

`list-supported-products`

列出您的 IDT 版本支持的产品（本例中为 AWS IoT Greengrass 版本），以及当前 IDT 版本适用的 AWS IoT Greengrass 资格测试套件版本。

`list-test-cases`

列出给定测试组中的测试用例。支持以下选项：

- `group-id`。要搜索的测试组。此选项是必需的，必须指定单个组。

`run-suite`

对某个设备池运行一组测试。以下是一些常用的选项：

- `suite-id`。要运行的测试套件版本。如果未指定，IDT 将使用 `tests` 文件夹中的最新版本。
- `group-id`。要以逗号分隔的列表形式运行的测试组。如果未指定，IDT 将运行测试套件中的所有测试组。
- `test-id`。要以逗号分隔的列表形式运行的测试用例。指定后，`group-id` 必须指定单个组。
- `pool-id`。要测试的设备池。如果您在 `device.json` 文件中定义了多个设备池，则测试运行器必须指定一个池。
- `timeout-multiplier`。将 IDT 配置为使用用户定义的乘数来修改 `test.json` 文件中为测试指定的测试执行超时。
- `stop-on-first-failure`。将 IDT 配置为在第一次失败时停止执行。应将此选项与 `group-id` 结合使用来调试指定的测试组。
- `userdata`。设置包含运行测试套件所需用户数据信息的文件。只有在测试套件的 `suite.json` 文件中 `userdataRequired` 被设置为 true 时，才需要这样做。

有关 run-suite 选项的更多信息，请使用 help 选项：

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

debug-test-suite

在调试模式下运行测试套件。有关更多信息，请参阅 在调试模式下运行 IDT。

查看 IDT 测试结果和日志

本节介绍 IDT 生成控制台日志和测试报告的格式。

控制台消息格式

AWS IoT Device Tester在控制台启动测试套件时，使用标准格式将消息打印到控制台。以下摘录显示了 IDT 生成的控制台消息的示例。

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

大多数控制台消息包含以下字段：

time

记录的事件的完整 ISO 8601 时间戳。

level

记录事件的消息级别。通常，记录的消息级别是其中之一info、warn，或者error. IDT 发行fatal要么panic如果遇到导致其提前退出的预期事件，则会发送消息。

msg

记录的消息。

executionId

当前 IDT 流程的唯一 ID 字符串。此 ID 用于区分个别 IDT 运行。

从测试套件生成的控制台消息提供了有关被测设备以及 IDT 运行的测试套件、测试组和测试用例的其他信息。以下摘录显示了从测试套件生成的控制台消息的示例。

```
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
```

控制台消息的测试套件特定部分包含以下字段：

`suiteId`

当前运行的测试套件的名称。

`groupId`

当前运行的测试组的 ID。

`testCaseId`

当前正在运行的测试用例的 ID。

`deviceId`

当前测试用例正在使用的受测设备的 ID。

要在 IDT 完成测试运行时向控制台打印测试摘要，您必须包含Reportstate在你的测试管弦乐队里。测试摘要包含有关测试套件的信息、运行的每个组的测试结果以及生成的日志和报告文件的位置。以下示例显示了测试摘要消息。

```
========== Test Summary ==========
Execution Time:     5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
----------------------------------
Test Groups:
    GroupA:         PASSED
    GroupB:         FAILED
----------------------------------
Failed Tests:
    Group Name: GroupB
        Test Name: TestB1
            Reason: Something bad happened
----------------------------------
Path to IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT Device Tester报告架构

awsiotdevicetester_report.xml是一份包含以下信息的签名报告：

- IDT 版本。
- 测试套件版本。
- 用于签署报告的报告签名和密钥。
- 中指定的设备 SKU 以及设备池名称device.json文件。
- 测试的产品版本和设备功能。
- 测试结果的摘要汇总。此信息与中包含的信息相同*suite-name*_report.xml文件。

```
<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>
```

awsiotdevicetester_report.xml 文件包含一个 <awsproduct> 标签，其中包含有关正测试的产品以及在运行测试套件后验证的产品功能的信息。

<awsproduct> 标签中使用的属性

name

所测试的产品的名称。

version

所测试的产品的版本。

features

验证的功能。标记为required是测试套件验证设备所必需的。以下代码段演示了此信息在 awsiotdevicetester_report.xml 文件中的显示方式。

```
<feature name="ssh" value="supported" type="required"></feature>
```

标记为optional是验证所必需的。以下代码段显示了可选功能。

```
<feature name="hsi" value="supported" type="optional"></feature>

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

测试套件报告架构

*suite-name*_Result.xml 报告采用 JUnit XML 格式。您可以将它集成到持续集成和开发平台，例如 Jenkins、Bamboo 等。此报告包含测试结果的摘要汇总。

```
<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>
```

两者中的报告部分awsiotdevicetester_report.xml要么*suite-name*_report.xml列出了已经运行的测试以及结果。

第一个 XML 标签 <testsuites> 包含测试执行情况的摘要。例如：

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

<testsuites> 标签中使用的属性

name

　测试套件的名称。

time

　运行测试套件所用的时间（以秒为单位）。

tests

　执行的测试数。

failures

已运行但未通过的测试数。

errors

IDT 无法执行的测试数。

disabled

此属性未使用，可以忽略。

如果出现测试失败或错误，则可以通过检查 <testsuites> XML 标签来确定失败的测试。<testsuites> 标签内的 <testsuite> XML 标签显示了测试组的测试结果摘要。例如：

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

其格式与 <testsuites> 标签类似，但包含一个未使用并可忽略的 skipped 属性。在每个 <testsuite> XML 标签内部，对于一个测试组，所执行的每个测试都有 <testcase> 标签。例如：

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

<testcase> 标签中使用的属性

name

测试的名称。

attempts

IDT 执行测试用例的次数。

当测试失败或出现错误时，将会在 <testcase> 标签中添加包含用于故障排除的信息的 <failure> 或 <error> 标签。例如：

```
<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>
```

ID用

如果您提供具有所需权限的AWS证书，则AWS IoT Device Tester收集使用指标并将其提交给AWS。这是一项可选功能，用于改进 IDT 功能。IDT 收集以下信息：

- 用于运行 IDT 的AWS 账户 ID
- 用于运行测试的 IDTAWS CLI 命令
- 正在运行的测试套件
- *<device-tester-extract-location >* 文件夹中的测试套件
- 设备池中配置的设备数量
- 测试用例名称和运行时间
- 测试结果信息，例如测试是否通过、失败、遇到错误或被跳过
- 产品功能已测试
- IDT 退出行为，例如意外退出或提前退出

IDT 发送的所有信息也会记录到该*<device-tester-extract-location>*/results/*<execution-id>*/文件夹中的`metrics.log`文件中。您可以查看日志文件以查看测试运行期间收集的信息。只有当您选择收集使用情况指标时，才会生成此文件。

要禁用指标收集，您无需采取其他措施。只需不存储您的AWS凭证，如果您存储了AWS证书，也不要将`config.json`文件配置为访问它们。

配置 AWS凭证

如果您还没有AWS 账户，您必须创建一个。如果您已经拥有AWS 账户，则只需为账户配置所需的权限，以允许 IDT 代表您向其AWS发送使用指标。

第 1 步：创建AWS 账户

在此步骤中，创建和配置AWS 账户。如果您已经有AWS 账户，请跳至the section called “步骤 2：为 IDT 配置权限”。

如果您还没有 AWS 账户，请完成以下步骤来创建一个。

注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

在注册时，您将接到一通电话，要求您使用电话键盘输入一个验证码。

当您注册 AWS 账户 时，系统将会创建一个 AWS 账户根用户。根用户有权访问该账户中的所有 AWS 服务和资源。作为安全最佳实践，请 为管理用户分配管理访问权限，并且只使用根用户执行 需要根用户访问权限的任务。

要创建管理员用户，请选择以下选项之一。

选择一种方法来管理您的管理员	目的	方式	您也可以
在 IAM Identity Center 中 （建议）	使用短期凭证访问 AWS。 这符合安全最佳实践。有关最佳实践的信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实践。	有关说明，请参阅《AWS IAM Identity Center 用户指南》中的入门。	按照《AWS Command Line Interface 用户指南》中的配置 AWS CLI 以使用 AWS IAM Identity Center，配置编程式访问。
在 IAM 中 （不推荐使用）	使用长期凭证访问 AWS。	按照《IAM 用户指南》中的创建您的首个 IAM 管理员用户和组的说明操作。	按照《IAM 用户指南》中的管理 IAM 用户的访问密钥，配置编程式访问。

步骤 2：为 IDT 配置权限

在此步骤中，配置 IDT 用于运行测试和收集 IDT 使用数据的权限。您可以使用AWS Management Console或AWS Command Line Interface (AWS CLI) 为 IDT 创建 IAM 策略和用户，然后将策略附加到该用户。

- 为 IDT 配置权限（控制台）

- [为 IDT 配置权限 (AWS CLI)]

为 IDT 配置权限（控制台）

请按照以下步骤使用控制台为适用于 AWS IoT Greengrass 的 IDT 配置权限。

1. 登录 IAM 控制台。
2. 创建客户托管策略，该策略授权创建具有特定权限的角色。

    a. 在导航窗格中，选择 Policies (策略)，然后选择 Create policy (创建策略)。
    b. 在 JSON 选项卡中，将占位符内容替换为以下策略。

    ```
    {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "iot-device-tester:SendMetrics"
                ],
                "Resource": "*"
            }
        ]
    }
    ```

    c. 选择Review policy（查看策略）。
    d. 对于 Name (名称)，请输入 **IDTUsageMetricsIAMPermissions**。在 Summary (摘要) 下，查看策略授予的权限。
    e. 选择 Create policy（创建策略）。
3. 创建 IAM

    a. 创建 IAM 用户。按照 IAM 用户指南中创建 IAM 用户（控制台）中的步骤 1 到 5 进行操作。如果您已创建 IAM
    b. 将权附加到您的 IM 用户：

        i. 在 Set permissions (设置权限) 页面上，选择 Attach existing policies to user directly (直接附加现有策略到用户)。
        ii. 搜索您在上一步中创建的 IDTUsageMetrics IAm entlyimess。选中复选框。
    c. 请选择下一步：标签。

d. 选择 Next: Review (下一步：审核) 以查看您的选择摘要。

e. 选择 Create user。

f. 要查看用户的访问密钥（访问密钥 ID 和秘密访问密钥），请选择密码和访问密钥旁边的 Show (显示)。要保存访问密钥，请选择Download.csv (下载 .csv)，然后将文件保存到安全位置。稍后您可以使用此信息来配置您的AWS证书文件。

为 IDT 配置权限 (AWS CLI)

请按照以下步骤使用 AWS CLI 为适用于 AWS IoT Greengrass 的 IDT 配置权限。

1. 在您的计算机上，安装并配置 AWS CLI（如果尚未安装）。按照《AWS Command Line Interface 用户指南》中的 "安装"AWS CLI 中的步骤进行操作。

> **Note**
>
> AWS CLI是一种开源工具，允许在命令行 Shell 中使用与AWS服务进行交互。

2. 创建以下客户管理策略，授予管理 IDT 和AWS IoT Greengrass角色的权限。

Linux or Unix

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document
                                          '{\"Version\": \"2012-10-17\",
 \"Statement\": [{\"Effect\": \"Allow\", \"Action\": [\"iot-device-
tester:SendMetrics\"], \"Resource": \"*\"}]}'
```

> **Note**
>
> 此步骤包含一个 Windows 命令提示符示例，因为它使用的 JSON 语法与 Linux、macOS 或 Unix 终端命令不同。

PowerShell

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}'
```

3. 创建 IMAWS IoT Greengrass

   a. 创建 IAM 用户。

   ```
   aws iam create-user --user-name user-name
   ```

   b. 将您创建的`IDTUsageMetricsIAMPermissions`策略附加到您的 IAM 用户。将*###*替换为您的 IAM 用户名，并将<account-id>命令中的用户名替换为您的 IDAWS 账户。

```
aws iam attach-user-policy --user-name user-name --policy-arn
 arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

4. 为用户创建私密访问密钥。

```
aws iam create-access-key --user-name user-name
```

将输出存储在安全位置。稍后您可以使用此信息来配置您的AWS证书文件。

向 IDT 提供AWS凭证

要允许 IDT 访问您的AWS证书并将指标提交给AWS，请执行以下操作：

1. 将 IAM 用户的AWS证书存储为环境变量或存储在证书文件中：

   a. 要使用环境变量，请运行以下命令。

      Linux or Unix

      ```
      export AWS_ACCESS_KEY_ID=access-key
      export AWS_SECRET_ACCESS_KEY=secret-access-key
      ```

      Windows Command Prompt (CMD)

      ```
      set AWS_ACCESS_KEY_ID=access-key
      set AWS_SECRET_ACCESS_KEY=secret-access-key
      ```

      PowerShell

      ```
      $env:AWS_ACCESS_KEY_ID="access-key"
      $env:AWS_SECRET_ACCESS_KEY="secret-access-key"
      ```

   b. 要使用凭证文件，请将以下信息添加到`~/.aws/credentials`文件中。

      ```
      [profile-name]
      aws_access_key_id=access-key
      aws_secret_access_key=secret-access-key
      ```

2. 配置`config.json`文件`auth`部分。有关更多信息，请参阅 [（可选）配置 config.json](#)。

对 IDT 进行故障排除AWS IoT GreengrassV2

同义词AWS IoT GreengrassV2 根据错误的类型将错误写入不同位置。IDT 将错误写入控制台、日志文件和测试报告。

在哪里寻找错误

测试运行时，控制台上会显示高级错误，当所有测试完成后，会显示失败测试的摘要。`awsiotdevicetester_report.xml`包含导致测试失败的所有错误的摘要。IDT 将每次测试运行的日志文件存储在带有测试执行的 UUID 的目录中，测试运行期间显示在控制台上。

IDT 测试日志目录是*<device-tester-extract-location>*/results/*<execution-id>*/logs/。此目录包含表格中显示的以下文件。这对于调试非常有用。

文件	描述
`test_manager.log`	测试运行时写入控制台的日志。此文件末尾的结果摘要包括了哪些测试失败的列表。 此文件中的警告和错误日志可以为您提供有关失败的一些信息。
test-group-id /*test-case-id* /*test-name* .log	测试组中特定测试的详细日志。对于部署 Greengrass 组件的测试，测试用例日志文件被称为`greengrass-test-run.log` 。
test-group-id /*test-case-id* / greengrass.log	的详细日志AWS IoT Greengrass核心软件。IDT 在运行安装测试时会从被测设备复制此文件AWS IoT Greengrass设备上的核心软件。有关此日志文件中消息的更多信息，请参阅故障排除 AWS IoT Greengrass V2。
test-group-id /*test-case-id*/*component-name* .log	测试运行期间部署的 Greengrass 组件的详细日志。IDT 在运行部署特定组件的测试时，会从被测设备复制组件日志文件。每个组件日志文件的名称对应于已部署组件的名称。有关此日志文件中消息的更多信息，请参阅故障排除 AWS IoT Greengrass V2。

正在解决 IDTAWS IoT GreengrassV2 错误

在你运行 IDT 之前AWS IoT Greengrass，获取正确的配置文件。如果您收到解析和配置错误，则第一步是找到并使用适合您环境的配置模板。

如果仍有问题，请参阅以下调试过程。

主题

- 别名解析错误
- 冲突错误
- 无法启动测试错误
- Docker 资格镜像存在错误
- 读取凭证失败
- Guice 错误与 PreInstalled Greengrass
- 签名异常无效
- 机器学习资格错误
- 开放测试框架 (OTF) 部署失败
- 解析错误
- 权限被拒绝错误
- 资格报告生成错误
- 缺少必需参数错误
- macOS 上的安全异常
- SSH 连接错误
- 直播管理员资格错误
- 超时错误
- 版本检查错误

别名解析错误

运行自定义测试套件时，您可能会在控制台和中看到以下错误`test_manager.log`。

```
Couldn't resolve placeholders: couldn't do a json lookup: index out of range
```

当 IDT 测试协调器中配置的别名无法正确解析或者配置文件中不存在已解析的值时，就会发生此错误。要解决此错误，请确保您的device.json和userdata.json 包含您的测试套件所需的正确信息。有关所需配置的信息AWS IoT Greengrass资格，请参阅配置 IDT 设置以运行 AWS IoT Greengrass 资格认证套件。

冲突错误

运行时可能会看到以下错误AWS IoT Greengrass在多台设备上同时使用资格套件。

```
ConflictException: Component [com.example.IDTHelloWorld : 1.0.0] for account [account-id] already exists with state: [DEPLOYABLE] { RespMetadata: { StatusCode: 409,
 RequestID: “id” }, Message_: “Component [com.example.IDTHelloWorld : 1.0.0] for
 account [account-id] already exists with state: [DEPLOYABLE]” }
```

尚不支持并行测试执行AWS IoT Greengrass资格套件。按顺序为每台设备运行资格套件。

无法启动测试错误

您可能会遇到错误，这些错误指向测试尝试开始时发生的故障。有几种可能的原因，因此，请执行以下操作：

- 确保执行命令中的池名称确实存在。IDT 直接引用您的 pool 名称device.json文件。
- 确保池中的设备具有正确的配置参数。

Docker 资格镜像存在错误

Docker 应用程序管理器资格测试使用amazon/amazon-ec2-metadata-mockAmazon ECR 中的容器镜像，用于验证被测设备。

如果镜像已存在于被测设备上的 Docker 容器中，则可能会收到以下错误。

```
The Docker image amazon/amazon-ec2-metadata-mock:version already exists on the device.
```

如果你之前下载了这张图片并运行了amazon/amazon-ec2-metadata-mock设备上的容器，在运行资格测试之前，请务必从被测设备中移除此映像。

读取凭证失败

在测试 Windows 设备时，你可能会遇到Failed to read credential里面有错误greengrass.log如果未在该设备的凭据管理器中设置您用来连接被测设备的用户，则为该文件。

要解决此错误，请在被测设备的凭据管理器中配置 IDT 用户的用户名和密码。

有关更多信息，请参阅为 Windows 设备配置用户凭证：

Guice 错误与 PreInstalled Greengrass

使用 IDT 运行时 PreInstalled Greengrass，如果你遇到的错误是`Guice`要么`ErrorInCustomProvider`，检查文件是否`userdata.json`有`InstalledDirRootOnDevice`设置为 Greengrass 安装文件夹。IDT 正在检查文件`effectiveConfig.yaml`下`<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`。

有关更多信息，请参阅为 Windows 设备配置用户凭证：

签名异常无效

在运行 Lambda 资格测试时，您可能会遇到`invalidsignatureexception`如果您的 IDT 主机遇到网络访问问题，则会出错。重置路由器并再次运行测试。

机器学习资格错误

在运行机器学习 (ML) 资格测试时，如果您的设备不满足，则可能会遇到资格认证失败要求部署AWS-提供了 ML 组件。要对 ML 资格错误进行故障排除，请执行以下操作：

- 在组件日志中查找测试运行期间部署的组件的错误详细信息。组件日志位于`<device-tester-extract-location>/results/<execution-id>/logs/<test-group-id>`目录。
- 添加`-Dgg.persist=installed.software`争论`test.json`失败的测试用例的文件。的`test.json`文件件位于`<device-tester-extract-location>/tests/GGV2Q_version directory.`

开放测试框架 (OTF) 部署失败

如果 OTF 测试未能完成部署，则可能的原因可能是为的父文件夹设置了权限`TempResourcesDirOnDevice`和`InstallationDirRootOnDevice`。要正确设置此文件夹的权限，请运行以下命令。替换*folder-name*使用父文件夹的名称。

```
sudo chmod 755 folder-name
```

解析错误

JSON 配置中的错别字可能导致解析错误。大部分情况下，问题是因 JSON 文件中漏掉括号、逗号或引号所导致。IDT 执行 JSON 验证并输入调试信息。它输出发生错误的行、行号以及语法错误的列号。这些信息应该足以帮助您修复错误，但是如果您仍然找不到错误，则可以在IDE、Atom或Sublime等文本编辑器中或通过jsonLint之类的在线工具手动进行验证。

权限被拒绝错误

IDT 将对所测试设备中的各种目录和文件执行操作。其中一些操作需要根用户访问权限。要自动执行这些操作，IDT 必须能够在不键入密码的情况下使用 sudo 运行命令。

请按照以下步骤操作，以允许在不键入密码的情况下进行 sudo 访问。

> **Note**
>
> `user` 和 `username` 是指 IDT 用来访问所测试设备的 SSH 用户。

1. 使用 sudo usermod -aG sudo ***<ssh-username>*** 将 SSH 用户添加到 sudo 组。
2. 注销，然后重新登录，以使更改生效。
3. 打开 `/etc/sudoers` 文件，并将以下行添加到文件末尾：`<ssh-username> ALL=(ALL) NOPASSWD: ALL`

   > **Note**
   >
   > 作为最佳实践，我们建议您在编辑 `/etc/sudoers` 时使用 sudo visudo。

资格报告生成错误

IDT 支持四种最新版本*major*.*minor*的版本AWS IoT GreengrassV2 资格套件 (GGV2Q)，用于生成资格报告，您可以将其提交给AWS Partner Network将您的设备包含在AWS Partner设备目录。早期版本的资格认证套件不生成资格报告。

如果您对支持政策有疑问，请联系AWS Support。

缺少必需参数错误

当 IDT 添加新功能时，它可能会对配置文件进行更改。使用旧配置文件可能会破坏您的配置。如果出现这种情况，`/results/<execution-id>/logs` 下的 `<test_case_id>.log` 文件明确列出了所有缺少的参数。IDT 还会验证您的 JSON 配置文件架构，以验证您使用的是最新的支持版本。

macOS 上的安全异常

当你在 macOS 主机上运行 IDT 时，它会阻止 IDT 的运行。要运行 IDT，请为作为 IDT 运行时功能一部分的可执行文件授予安全例外。当您在主机上看到警告消息显示时，请对每个适用的可执行文件执行以下操作：

向 IDT 可执行文件授予安全例外

1. 在 macOS 电脑上，在苹果菜单上，打开系统偏好设置。
2. 选择安全与隐私，然后在普通的选项卡上，选择锁定图标以更改安全设置。
3. 如果被封锁`devicetester_mac_x86-64`，寻找消息`"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.`然后选择无论如何都允许。
4. 恢复 IDT 测试，直到完成所有涉及的可执行文件。

SSH 连接错误

当 IDT 无法连接到被测设备时，它会将连接失败记录在`/results/<execution-id>/logs/<test-case-id>.log`。SSH 消息出现在该日志文件的顶部，因为连接到被测设备是 IDT 执行的第一批操作之一。

大多数 Windows 配置使用 PuTTY 终端应用程序连接到 Linux 主机。此应用程序要求您将标准 PEM 私钥文件转换为名为 PPK 的 Windows 专有格式。如果你在你的 SSH 中配置`device.json`文件，使用 PEM 文件。如果您使用 PPK 文件，IDT 将无法与 PPK 文件创建 SSH 连接AWS IoT Greengrass设备且无法运行测试。

从 IDT v4.4.0 开始，如果您尚未在被测设备上启用 SFTP，则可能会在日志文件中看到以下错误。

```
SSH connection failed with EOF
```

要解决此错误，请在您的设备上启用 SFTP。

直播管理员资格错误

运行直播管理员资格测试时，您可能会在中看到以下错误com.aws.StreamManagerExport.log文件。

```
Failed to upload data to S3
```

当直播管理器使用时，可能会发生此错误AWS中的凭证~/root/.aws/credentials在您的设备上归档，而不是使用 IDT 导出到被测设备的环境凭证。要防止出现此问题，请删除credentials在您的设备上归档，然后重新运行资格测试。

超时错误

您可以通过指定应用于每个测试超时默认值的超时乘数来增加每个测试的超时时间。为此标志配置的任何值都必须大于或等于 1.0。

要使用超时乘数，请在运行测试时使用标志 --timeout-multiplier。例如：

```
./devicetester_linux run-suite --suite-id GGV2Q_1.0.0 --pool-id DevicePool1 --timeout-multiplier 2.5
```

有关更多信息，请运行 run-suite --help。

由于配置问题而无法完成 IDT 测试用例时，会出现一些超时错误。您无法通过增加超时乘数来解决这些错误。使用测试运行的日志来解决底层的配置问题。

- 如果 MQTT 或 Lambda 组件日志包含Access denied错误，您的 Greengrass 安装文件夹可能没有正确的文件权限。对您在安装路径中定义的每个文件夹运行以下命令userdata.json文件。

  ```
  sudo chmod 755 folder-name
  ```

- 如果 Greengrass 日志显示 Greengrass CLI 部署尚未完成，请执行以下操作：
  - 验证一下bash安装在被测设备上。
  - 如果你的userdata.json文件包含GreengrassCliVersion配置参数，将其删除。在 IDT v4.1.0 及更高版本中，此参数已被弃用。有关更多信息，请参阅配置 userdata.json：
- 如果 Lambda 部署测试失败并显示错误消息 “验证 Lambda 发布：超时”，并且您在测试日志文件中收到错误 (idt-gg2-lambda-function-idt-*<resource-id>*.log) 上面写着Error: Could not find or load main class com.amazonaws.greengrass.runtime.LambdaRuntime.，请执行以下操作：

- 验证使用了什么文件夹`InstallationDirRootOnDevice`在`userdata.json`文件。
- 确保在您的设备上设置了正确的用户权限。有关更多详细信息，请参阅在您的设备上配置用户权限。

版本检查错误

IDT 在以下情况下会发出以下错误AWSIDT 用户的用户证书没有所需的 IAM 权限。

```
Failed to check version compatibility
```

的AWS没有所需的 IAM 权限的用户。

的 Support AWS IoT Device Tester 政策 AWS IoT Greengrass

AWS IoT Device Tester f AWS IoT Greengrass or 是一款测试自动化工具，用于验证和确认您的 AWS IoT Greengrass 设备是否包含在AWS Partner 设备目录中。我们建议您使用最新版本的 AWS IoT Greengrass 和 AWS IoT Device Tester 来测试或鉴定您的设备。

的每个支持的版本 AWS IoT Device Tester 都至少有一个版本可用 AWS IoT Greengrass。有关支持的版本 AWS IoT Greengrass，请参阅 Greengrass nucleus 版本。有关支持的版本 AWS IoT Device Tester，请参阅支持的 f AWS IoT Device Tester or AWS IoT Greengrass V2 版本。

您还可以使用 AWS IoT Greengrass 和的任何支持的版本 AWS IoT Device Tester 来测试或鉴定您的设备。尽管您可以继续使用不支持的版本 AWS IoT Device Tester，但这些版本不会收到错误修复或更新。如果您对支持政策有疑问，请联系AWS Support。

基于 Greengrass 的物联网解决方案

Eurotech 的 Everyware GreenEdge 处于预览版AWS IoT Greengrass，可能会发生变化。AWS 不支持此解决方案。如果此设备有任何问题，您必须联系 Eurotech。

AWS IoT Greengrass提供合作伙伴提供的解决方案，以优化您安装 Greengrass 的体验。以下是与 Euro AWS tech 合作提供的解决方案。该解决方案预装了 AWS IoT Greengrass Core 边缘运行时和其他功能。

欧洲科技大学

AWS已与 Eurotech 合作，为正在寻找预装了 C AWS IoT Greengrass ore 软件的设备的客户提供物联网解决方案。Eurotech 的 Everyware GreenEdge 是一款物联网边缘软件，经过预先配置和预认证。AWS该解决方案融合了Greengrass和Eurotech Everyware软件框架（ESF）的功能，通过协议适配器为客户提供广泛的南向连接，例如：Modbus、OPC-UA Client/Server、S7、TwinCat、J1939、DNP3 Master/Outstation 等。使用此解决方案，您还可以将数据发送到AWS Cloud并连接到所有北行AWS服务（例如、、、Amazon S3 和 Amazon Kinesis Video Streams Amazon Kinesis Video Streams）。AWS IoT Core AWS IoT SiteWise AWS IoT Analytics该解决方案与Eurotech的设备管理解决方案Everyware Cloud相结合，引入了一种新颖的零接触配置服务，可简化设备上线和大规模部署。

有关 Eurotech 的更多信息，请参阅 Eu ro tech。

故障排除 AWS IoT Greengrass V2

使用本节中的疑难解答信息和解决方案来帮助解决问题 AWS IoT Greengrass Version 2。

主题

- 查看 AWS IoT Greengrass 核心软件和组件日志
- AWS IoT Greengrass 核心软件问题
- AWS IoT Greengrass 云问题
- 核心设备部署问题
- 核心设备组件问题
- 核心设备 Lambda 函数组件问题
- 组件版本已停产
- Greengrass 命令行界面问题
- AWS Command Line Interface 问题
- 详细的部署错误代码
- 详细的组件状态码

查看 AWS IoT Greengrass 核心软件和组件日志

AWS IoT Greengrass Core 软件将日志写入本地文件系统，您可以使用该文件系统查看有关核心设备的实时信息。您还可以将核心设备配置为将日志写入日 CloudWatch 志，这样您就可以远程排除核心设备的故障。这些日志可以帮助您识别组件、部署和核心设备的问题。有关更多信息，请参阅 监控AWS IoT Greengrass日志。

AWS IoT Greengrass 核心软件问题

对 AWS IoT Greengrass 核心软件问题进行故障排除。

主题

- 无法设置核心设备
- 无法将 AWS IoT Greengrass Core 软件作为系统服务启动
- 无法将 nucleus 设置为系统服务
- 无法连接到 AWS IoT Core

- 内存不足错误
- 无法安装 Greengrass CLI
- User root is not allowed to execute
- com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with
- Failed to map segment from shared object: operation not permitted
- 无法设置 Windows 服务
- com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
- com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime
- software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid
- software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy
- Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request
- Operation aws.greengrass#<operation> is not supported by Greengrass
- java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed
- java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED
- Greengrass core device stuck on nucleus v2.12.3

无法设置核心设备

如果 AWS IoT Greengrass Core 软件安装程序失败而您无法设置核心设备，则可能需要卸载该软件并重试。有关更多信息，请参阅 卸载AWS IoT Greengrass核心软件。

无法将 AWS IoT Greengrass Core 软件作为系统服务启动

如果 AWS IoT Greengrass Core 软件无法启动，请检查系统服务日志以确定问题。一个常见的问题是 Java 在 PATH 环境变量 (Linux) 或 PATH 系统变量 (Windows) 上不可用。

无法将 nucleus 设置为系统服务

当 AWS IoT Greengrass Core 软件安装程序无法设置 AWS IoT Greengrass 为系统服务时，您可能会看到此错误。在 Linux 设备上，如果核心设备没有 systemd 初始化系统，则通常会发生此错误。即使安装程序未能设置系统服务，安装程序也可以成功设置 AWS IoT Greengrass 核心软件。

请执行以下操作之一：

- 将 C AWS IoT Greengrass ore 软件配置为系统服务并运行。必须将软件配置为系统服务才能使用的所有功能 AWS IoT Greengrass。您可以安装 systemd 或使用不同的初始化系统。有关更多信息，请参阅 将 Greengrass 核心配置为系统服务。
- 在没有系统服务的情况下运行 C AWS IoT Greengrass ore 软件。您可以使用安装程序在 Greengrass 根文件夹中设置的加载程序脚本来运行该软件。有关更多信息，请参阅 在没有系统服务的情况下运行 C AWS IoT Greengrass ore 软件。

无法连接到 AWS IoT Core

例如，当 AWS IoT Greengrass Core 软件无法连接 AWS IoT Core 以检索部署任务时，您可能会看到此错误。执行以下操作：

- 检查您的核心设备是否可以连接到互联网，然后 AWS IoT Core. 有关您的设备连接的 AWS IoT Core 终端节点的更多信息，请参阅配置 AWS IoT Greengrass 核心软件。
- 检查您的核心设备是否 AWS IoT 使用了允许`iot:Connect`、`iot:Publishiot:Receive`、和`iot:Subscribe`权限的证书。
- 如果您的核心设备使用网络代理，请检查您的核心设备是否具有设备角色以及其角色是否允许`iot:Connectiot:Publish`、`iot:Receive`、和`iot:Subscribe`权限。

内存不足错误

如果您的设备内存不足，无法在 Java 堆中分配对象，则通常会发生此错误。在内存有限的设备上，您可能需要指定最大堆大小来控制内存分配。有关更多信息，请参阅 使用 JVM 选项控制内存分配。

无法安装 Greengrass CLI

当你在 C AWS IoT Greengrass ore 的安装命令中使用--deploy-dev-tools参数时，你可能会看到以下控制台消息。

```
Thing group exists, it could have existing deployment and devices, hence NOT creating
 deployment for Greengrass first party dev tools, please manually create a deployment
 if you wish to
```

当您的核心设备是现有部署的事物组的成员而未安装 Greengrass CLI 组件时，就会发生这种情况。如果您看到此消息，则可以手动将 Greengrass CLI 组件 aws.greengrass.Cli () 部署到设备上以安装 Greengrass CLI。有关更多信息，请参阅 安装 Greengrass CLI。

User root is not allowed to execute

当运行 C AWS IoT Greengrass ore 软件的用户（通常root）无权sudo与任何用户和任何组一起运行时，您可能会看到此错误。对于默认ggc_user系统用户，此错误如下所示：

```
Sorry, user root is not allowed to execute <command> as ggc_user:ggc_group.
```

检查您的/etc/sudoers文件是否允许用户以其他群组的sudo身份运行。中用户的权限/etc/sudoers应类似于以下示例。

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

com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with

当核心设备尝试运行组件，而 Greengrass nucleus 未指定用于运行组件的默认系统用户时，您可能会看到此错误。

要修复此问题，请配置 Greengrass nucleus 以指定运行组件的默认系统用户。有关更多信息，请参阅 配置运行组件的用户 和 配置默认组件用户。

Failed to map segment from shared object: operation not permitted

当 AWS IoT Greengrass Core 软件无法启动时，您可能会看到此错误，因为该/tmp文件夹已安装noexec权限。默认情况下，AWS 公共运行时 (CRT) 库使用该/tmp文件夹。

请执行以下操作之一：

- 运行以下命令以重新安装具有exec权限/tmp的文件夹，然后重试。

```
sudo mount -o remount,exec /tmp
```

- 如果你运行 Greengrass nucleus v2.5.0 或更高版本，则可以设置 JVM 选项来更改 CRT 库使用的文件夹。 AWS 您可以在部署中或安装核心软件时在 Greengrass 核心组件配置中指定jvmOptions参数。 AWS IoT Greengrass 将 */path/to/use ### AWS CRT ######*文件夹的路径。

```
{
  "jvmOptions": "-Daws.crt.lib.dir=\"/path/to/use\""
}
```

无法设置 Windows 服务

如果你在微软 Windows 2016 设备上安装 AWS IoT Greengrass 酷睿软件，你可能会看到这个错误。Windows 2016 不支持 AWS IoT Greengrass 核心软件，有关支持的操作系统的列表，请参阅支持的平台。

如果你必须使用 Windows 2016，你可以执行以下操作：

1. 解压缩下载的 AWS IoT Greengrass Core 安装档案
2. 在Greengrass目录中打开bin/greengrass.xml.template文件。
3. 将<autoRefresh>标签添加到文件末尾的</service>标签前面。

```
  </log>
  <autoRefresh>false</autoRefresh>
</service>
```

com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager

在没有根证书颁发机构 (CA) 文件的情况下安装 C AWS IoT Greengrass ore 软件时，您可能会看到此错误。

```
2022-06-05T10:00:39.556Z [INFO] (main) com.aws.greengrass.lifecyclemanager.Kernel:
 service-loaded. {serviceName=DeploymentService}
2022-06-05T10:00:39.943Z [WARN] (main)
 com.aws.greengrass.componentmanager.ClientConfigurationUtils: configure-greengrass-
mutual-auth. Error during configure greengrass client mutual auth. {}
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
```

检查您是否在提供给安装程序的配置文件中使用`rootCaPath`参数指定了有效的根 CA 文件。有关更多信息，请参阅 安装 AWS IoT Greengrass Core 软件。

com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime

当核心设备无法连接以订阅部署任务通知时 AWS IoT Core ，您可能会看到此警告消息。执行以下操作：

- 检查核心设备是否已连接到互联网并且可以访问您配置 AWS IoT 的数据端点。有关核心设备使用的端点的更多信息，请参阅允许设备流量通过代理或防火墙。
- 查看 Greengrass 日志中是否有其他能揭示其他根本原因的错误。

software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid

当您安装具有自动配置功能的 AWS IoT Greengrass Core 软件时，您可能会看到此错误，而安装程序使用的 AWS 会话令牌无效。执行以下操作：

- 如果您使用临时安全证书，请检查会话令牌是否正确，以及是否正在复制和粘贴完整的会话令牌。
- 如果您使用长期安全证书，请检查设备是否没有您之前使用临时证书时的会话令牌。执行以下操作：

  1. 运行以下命令取消设置会话令牌环境变量。

Linux or Unix

```
unset AWS_SESSION_TOKEN
```

Windows Command Prompt (CMD)

```
set AWS_SESSION_TOKEN=
```

PowerShell

```
Remove-Item Env:\AWS_SESSION_TOKEN
```

2. 检查 AWS 凭证文件是否包含会话令牌aws_session_token。~/.aws/credentials如果是，请从文件中删除该行。

```
aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
+FvwqnKwRcOIfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

您也可以在不提供 AWS 凭据的情况下安装 AWS IoT Greengrass Core 软件。有关更多信息，请参阅使用手动资源配置来安装 AWS IoT Greengrass Core 软件或安装具有 AWS IoT 队列配置功能的 C AWS IoT Greengrass ore 软件。

software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy

在安装具有自动配置功能的 AWS IoT Greengrass Core 软件时，您可能会看到此错误，而安装程序使用的 AWS 凭据不具有所需权限。有关所需权限的更多信息，请参阅安装程序配置资源的最低 IAM 政策。

检查凭证的 IAM 身份的权限，并向 IAM 身份授予缺失的所有必需权限。

Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request

当你使用影子管理器组件同步设备阴影时，你可能会看到这个错误 AWS IoT Core。HTTP 403 状态码表示出现此错误是因为核心设备的 AWS IoT 策略未授予呼叫GetThingShadow权限。

```
com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute
 cloud shadow get request. {thing name=MyGreengrassCore, shadow name=MyShadow}
2021-07-14T21:09:02.456Z [ERROR] (pool-2-thread-109)
 com.aws.greengrass.shadowmanager.sync.SyncHandler: sync. Skipping sync request. {thing
 name=MyGreengrassCore, shadow name=MyShadow}
com.aws.greengrass.shadowmanager.exception.SkipSyncRequestException:
 software.amazon.awssdk.services.iotdataplane.model.IotDataPlaneException:
 null (Service: IotDataPlane, Status Code: 403, Request ID:
 f6e713ba-1b01-414c-7b78-5beb3f3ad8f6, Extended Request ID: null)
```

要与本地阴影同步 AWS IoT Core，核心设备的 AWS IoT 策略必须授予以下权限：

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

检查核心设备的 AWS IoT 策略，并添加缺少的所有必需权限。有关更多信息，请参阅下列内容：

- AWS IoT Core 《AWS IoT 开发者指南》中的@@ 策略操作
- 更新核心设备的AWS IoT政策

Operation aws.greengrass#<operation> is not supported by Greengrass

当您在自定义 Greengrass 组件中使用进程间通信 (IPC) 操作并且核心设备上未安装所需的组件时，您可能会看到此错误 AWS。

要解决此问题，请在组件配方中添加所需的组件作为依赖项，这样 AWS IoT Greengrass Core 软件就会在您部署组件时安装所需的组件。

- 检索秘密值 — `aws.greengrass.SecretManager`
- 与局部阴影互动 — `aws.greengrass.ShadowManager`

- 管理本地部署和组件 — `aws.greengrass.Cli` v2.6.0 或更高版本
- 对@@ 客户端设备进行身份验证和授权 — `aws.greengrass.clientdevices.Auth` v2.2.0 或更高版本

java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)

当您将流管理器配置为使用不存在或权限正确的根文件夹时，您可能会在流管理器日志文件 (`aws.greengrass.StreamManager.log`) 中看到此错误。有关如何配置此文件夹的更多信息，请参阅直播管理器配置。

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist

当 PKCS #11 提供程序组件无法找到或加载您在将 C AWS IoT Greengrass ore 软件配置为使用硬件安全模块 (HSM) 时指定的私钥或证书时，就会发生此错误。执行以下操作：

- 使用配置 AWS IoT Greengrass 核心软件使用的插槽、用户 PIN 和对象标签，检查私钥和证书是否存储在 HSM 中。
- 检查私钥和证书在 HSM 中是否使用相同的对象标签。
- 如果您的 HSM 支持对象 ID，请检查私钥和证书在 HSM 中是否使用相同的对象 ID。

查看 HSM 的文档，了解如何在 HSM 中查询有关安全令牌的详细信息。如果您需要更改安全令牌的插槽、对象标签或对象 ID，请查看 HSM 的文档以了解如何操作。

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>

当您使用密钥管理器组件部署密钥时，可能会发生此错误。 AWS Secrets Manager 如果核心设备的令牌交换 IAM 角色未授予获取密钥的权限，则部署将失败，并且 Greengrass 日志中包含此错误。

授权核心设备下载密钥

1. 为核心设备的令牌交换角色添加`secretsmanager:GetSecretValue`权限。以下示例策略声明授予获取密钥值的权限。

```
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager:GetSecretValue"
    ],
    "Resource": [
        "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
abcdef"
    ]
}
```

有关更多信息，请参阅 授权核心设备与AWS服务。

2. 将部署重新应用于核心设备。请执行以下操作之一：

   - 在不做任何更改的情况下修改部署。当核心设备收到修改后的部署时，它会尝试再次下载密钥。有关更多信息，请参阅 修改部署。
   - 重新启动 C AWS IoT Greengrass ore 软件以重试部署。有关更多信息，请参阅 运行AWS IoT Greengrass核心软件。

   如果密钥管理器成功下载密钥，则部署成功。

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed

当您使用密钥管理器组件部署由密钥加密的 AWS Secrets Manager 密 AWS Key Management Service 钥时，可能会发生此错误。如果核心设备的令牌交换 IAM 角色未授予解密密密钥的权限，则部署将失败，并且 Greengrass 日志中包含此错误。

要解决此问题，请向核心设备的令牌交换角色添加`kms:Decrypt`权限。有关更多信息，请参阅下列内容：

- 《用户指南》中的@@ 秘密加密和解密AWS Secrets Manager
- 授权核心设备与AWS服务

java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi

当你尝试安装具有硬件安全性的 AWS IoT Greengrass Core软件并且使用不支持硬件安全集成的早期 Greengrass nucleus版本时，你可能会看到这个错误。要使用硬件安全集成，必须使用 Greengrass nucleus v2.5.3 或更高版本。

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED

当你将 C AWS IoT Greengrass ore 作为系统服务运行时，使用 TPM2 库时，你可能会看到这个错误。

此错误表示您需要添加一个环境变量，该变量在 C AWS IoT Greengrass ore systemd 服务文件中提供 PKCS #11 存储的位置。

有关更多信息，请参阅PKCS #11 提供商组件文档的 “要求” 部分。

Greengrass core device stuck on nucleus v2.12.3

如果您的 Greengrass 核心设备无法从 nucleus 版本 2.12.3 修改您的部署，则可能需要下载该文件并将其替换为 Greengrass nucleus 版本 2.12.2。`Greengrass.jar`执行以下操作：

1. 在你的 Greengrass 核心设备上，运行以下命令停止 Greengrass Core 软件。

   Linux or Unix

   ```
   sudo systemctl stop greengrass
   ```

   Windows Command Prompt (CMD)

   ```
   sc stop "greengrass"
   ```

   PowerShell

   ```
   Stop-Service -Name "greengrass"
   ```

2. 在您的核心设备上，将 AWS IoT Greengrass 软件下载到名为的文件中`greengrass-2.12.2.zip`。

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >
 greengrass-2.12.2.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >
 greengrass-2.12.2.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip -
OutFile greengrass-2.12.2.zip
```

3. 将 AWS IoT Greengrass Core 软件解压缩到设备上的某个文件夹。*GreengrassInstaller*替换为要使用的文件夹。

   Linux or Unix

   ```
   unzip greengrass-2.12.2.zip -d GreengrassInstaller && rm greengrass-2.12.2.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-2.12.2.zip -
   C GreengrassInstaller && del greengrass-2.12.2.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-2.12.2.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-2.12.2.zip
   ```

4. 运行以下命令，使用 nucleus 版本 2.12.2 Greengrass JAR 文件覆盖 nucleus 版本 2.12.3 Greengrass JAR 文件。

Linux or Unix

```
sudo cp ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/
artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib
```

Windows Command Prompt (CMD)

```
robocopy ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/
artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib /E
```

PowerShell

```
cp -Path ./GreengrassInstaller/lib/Greengrass.jar -Destination /
greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/
aws.greengrass.nucleus/lib
```

5. 运行以下命令启动 Greengrass Core 软件。

   Linux or Unix

   ```
   sudo systemctl start greengrass
   ```

   Windows Command Prompt (CMD)

   ```
   sc start "greengrass"
   ```

   PowerShell

   ```
   Start-Service -Name "greengrass"
   ```

AWS IoT Greengrass 云问题

使用以下信息对 AWS IoT Greengrass 控制台和 API 的问题进行故障排除。每个条目都对应一条错误消息，您在执行操作时可能会看到该消息。

An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null

当您通过 AWS IoT Greengrass 控制台或通过CreateComponentVersion操作创建组件版本时，您可能会看到此错误。

此错误表明您的食谱不是有效的 JSON 或 YAML。请检查配方的语法，修复所有语法问题，然后重试。您可以使用在线 JSON 或 YAML 语法检查器来识别配方中的语法问题。

Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}

当您通过 AWS IoT Greengrass 控制台或通过CreateComponentVersion操作创建组件版本时，您可能会看到此错误。此错误表示组件配方中的 S3 工件无效。

执行以下操作：

- 检查 S3 存储桶是否与您创建组件的 AWS 区域 位置相同。 AWS IoT Greengrass 不支持跨区域请求组件工件。
- 检查项目 URI 是否为有效的 S3 对象 URL，并检查该 S3 对象 URL 中是否存在该项目。
- 检查您是否 AWS 账户 有权通过其 S3 对象 URL 访问该项目。

INACTIVE deployment status

在没有必需的依赖 AWS IoT 策略的情况下调用 ListDeploymentsAPI 时，您可能会获得INACTIVE部署状态。您必须拥有必要的权限才能获得准确的部署状态。您可以通过查看由定义的操作 AWS IoT Greengrass V2并遵循所需的权限来找到相关操作ListDeployments。如果没有所需的依赖 AWS IoT 权限，您仍然可以看到部署状态，但可能会看到不准确的部署状态INACTIVE。

核心设备部署问题

对 Greengrass 核心设备上的部署问题进行故障排除。每个条目都对应一条日志消息，您可能会在核心设备上看到该消息。

主题

- Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact
- Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.
- Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>
- software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility
- com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component
- Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service
- Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration
- Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy
- Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration
- Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact

当核心设备应用部署时，当 AWS IoT Greengrass 核心软件无法下载组件工件时，您可能会看到此错误。由于此错误，部署失败。

当您收到此错误时，日志中还会包含可用于识别特定问题的堆栈跟踪。以下每个条目都对应一条消息，您可能会在`Failed to download artifact`错误消息的堆栈跟踪中看到该消息。

主题

- software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)
- software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)

在以下情况下，该PackageDownloadException 错误可能包括此堆栈跟踪：

- 组件工件在组件配方中指定的 S3 对象 URL 中不可用。检查您是否已将项目上传到 S3 存储桶，以及项目 URI 是否与存储桶中项目的 S3 对象 URL 相匹配。
- 核心设备的令牌交换角色不允许 AWS IoT Greengrass 核心软件从您在组件配方中指定的 S3 对象 URL 下载组件工件。检查令牌交换角色是否允许使用`s3:GetObject`对象可用的 S3 对象 URL。

software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

当核心设备无权调用时，PackageDownloadException 错误可能包括此堆栈跟踪`s3:GetBucketLocation`。错误消息还包括以下消息。

```
reason: Failed to determine S3 bucket location
```

检查核心设备的令牌交换角色是否允许`s3:GetBucketLocation`项目可用的 S3 存储桶。

Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchE Integrity check for downloaded artifact failed. Probably due to file corruption.

当核心设备应用部署时，当 AWS IoT Greengrass 核心软件无法下载组件工件时，您可能会看到此错误。部署失败，因为下载的构件文件的校验和与创建组件时 AWS IoT Greengrass 计算的校验和不匹配。

执行以下操作：

- 检查托管项目文件的 S3 存储桶中是否发生了更改。如果自创建组件以来文件发生了变化，请将其恢复到核心设备期望的先前版本。如果您无法将文件恢复到其先前版本，或者想要使用该文件的新版本，请使用构件文件创建该组件的新版本。
- 检查您的核心设备的互联网连接。如果工件文件在下载时损坏，则可能会发生此错误。创建新部署，然后重试。

Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVers Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>

当核心设备找不到满足该核心设备部署要求的组件版本时，您可能会看到此错误。核心设备在 AWS IoT Greengrass 服务中和本地设备上检查组件。错误消息包括每个部署的目标以及该部署对组件的版本要求。部署目标可以是事物、事物组或`LOCAL_DEPLOYMENT`，它代表核心设备上的本地部署。

在以下情况下可能会出现此问题：

- 核心设备是多个部署的目标，这些部署的组件版本要求相互冲突。例如，核心设备可能是包含一个`com.example.HelloWorld`组件的多个部署的目标，其中一个部署需要版本 1.0.0，另一个需要版本 1.0.1。不可能有一个同时满足这两个要求的组件，因此部署失败了。
- 该组件版本不存在于 AWS IoT Greengrass 服务中或本地设备上。例如，该组件可能已被删除。
- 有符合版本要求的组件版本，但没有一个版本与核心设备的平台兼容。
- 核心设备的 AWS IoT 策略不授予`greengrass:ResolveComponentCandidates`权限。在错误日志`Status Code: 403`中查找以识别此问题。要解决此问题，请

将`greengrass:ResolveComponentCandidates`权限添加到核心设备的 AWS IoT 策略中。有关更多信息，请参阅 AWS IoT Greengrass V2核心设备的最低AWS IoT政策。

要解决此问题，请修改部署以包括兼容的组件版本或删除不兼容的组件版本。有关如何修改云部署的更多信息，请参阅修改部署。有关如何修改本地部署的更多信息，请参阅 AWS IoT Greengrass CLI 部署创建命令。

software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility

当您将组件部署到核心设备并且该组件未列出与核心设备平台兼容的平台时，您可能会看到此错误。请执行以下操作之一：

- 如果该组件是自定义 Greengrass 组件，则可以更新该组件以使其与核心设备兼容。添加与核心设备平台匹配的新清单，或更新现有清单以匹配核心设备的平台。有关更多信息，请参阅 AWS IoT Greengrass组件配方参考。
- 如果组件由提供 AWS，请检查该组件的另一个版本是否与核心设备兼容。如果没有版本兼容，请AWS re:Post使用AWS IoT Greengrass 标签联系我们，或者联系我们AWS Support。

com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component

部署依赖于 Greengrass 核心的组件，并且核心设备运行的 Greengrass 核心版本早于可用的最新次要版本时，您可能会看到此错误。出现此错误的原因是 AWS IoT Greengrass Core 软件会尝试自动将组件更新到最新的兼容版本。但是， AWS IoT Greengrass Core 软件会阻止 Greengrass 核更新到新的次要版本，因为提供的 AWS几个组件依赖于 Greengrass 核的特定次要版本。有关更多信息，请参阅 Greengrass 核更新行为。

您必须修改部署以指定要使用的 Greengrass 核心版本。请执行以下操作之一：

- 修改部署以指定核心设备当前运行的 Greengrass 核心版本。
- 修改部署以指定 Greengrass 核的后续次要版本。如果选择此选项，则还必须更新所有依赖于 Greengr AWS ass 核的特定次要版本的组件的版本。有关更多信息，请参阅 AWS-提供的组件。

Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service

当你将 Greengrass 设备从一个事物组移到另一个事物组，然后再移回需要重启 Greengrass 的部署的原始组时，你可能会看到这个错误。

要解决此问题，请重新创建设备的启动目录。我们还强烈建议升级到 Greengrass 核心的 2.9.6 或更高版本。

以下是用于重新创建启动目录的 Linux 脚本。将脚本保存在名为的文件中`fix_directory.sh`。

```
#!/bin/bash

set -e

GG_ROOT=$1
GG_VERSION=$2

CURRENT="$GG_ROOT/alts/current"

if [ ! -L "$CURRENT" ]; then
  mkdir -p $GG_ROOT/alts/directory_fix
  echo "Relinking $GG_ROOT/alts/directory_fix to $CURRENT"
  ln -sf $GG_ROOT/alts/directory_fix $CURRENT
fi

TARGET=$(readlink $CURRENT)

if [[ ! -d "$TARGET" ]]; then
  echo "Creating directory: $TARGET"
  mkdir -p "$TARGET"
fi

DISTRO_LINK="$TARGET/distro"
DISTRO="$GG_ROOT/packages/artifacts-unarchived/aws.greengrass.Nucleus/$GG_VERSION/
aws.greengrass.nucleus/"
echo "Relinking Nucleus artifacts to $DISTRO_LINK"
ln -sf $DISTRO $DISTRO_LINK
```

要运行脚本，请执行以下命令：

```
[root@ip-172-31-27-165 ~]# ./fix_directory.sh /greengrass/v2 2.9.5
Relinking /greengrass/v2/alts/directory_fix to /greengrass/v2/alts/current
Relinking Nucleus artifacts to /greengrass/v2/alts/directory_fix/distro
```

Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration

当核心设备收到大型部署文档时，您可能会看到此错误，该文档是大于 7 KB（针对以事物为目标的部署）或 31 KB（针对事物组的部署）的部署文档。要检索大型部署文档，核心设备的 AWS IoT 策略必须允许该greengrass:GetDeploymentConfiguration权限。当核心设备没有此权限时，可能会发生此错误。发生此错误时，部署将无限期重试，其状态为 “进行中” ()。IN_PROGRESS

要解决此问题，请将greengrass:GetDeploymentConfiguration权限添加到核心设备的 AWS IoT 策略中。有关更多信息，请参阅 更新核心设备的AWS IoT政策。

Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy

当核心设备收到部署并且核心设备的 AWS IoT 策略不允许该greengrass:ListThingGroupsForCoreDevice权限时，您可能会看到此警告。创建部署时，核心设备使用此权限来识别其事物组，并移除已从中移除核心设备的所有事物组的组件。如果核心设备运行 Greengrass nucleus v2.5.0，则部署将失败。如果核心设备运行 Greengrass nucleus v2.5.1 或更高版本，则部署会继续进行，但不会移除组件。有关事物组移除行为的更多信息，请参阅将AWS IoT Greengrass组件部署到设备。

要更新核心设备的行为以移除要从中移除核心设备的事物组的组件，请将greengrass:ListThingGroupsForCoreDevice权限添加到核心设备的 AWS IoT 策略中。有关更多信息，请参阅 更新核心设备的AWS IoT政策。

Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration

您可能会看到此信息消息多次打印而不会出现错误，因为核心设备会在DEBUG日志级别记录错误。当核心设备收到大型部署文档时，可能会出现此问题。出现此问题时，部署将无限期重试，其状态为 “进行中” ()。IN_PROGRESS有关如何解决此问题的更多信息，请参阅此疑难解答条目。

Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataE null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

当数据平面 API 没有`iot:Connect`权限时，你可能会看到此错误。如果您没有正确的保单，您将收到`GreengrassV2DataException: 403`. 要创建权限策略，请按照以下说明进行操作：创建 AWS IoT 策略。

核心设备组件问题

对核心设备上的 Greengrass 组件问题进行故障排除。

主题

- Warn: '<command>' is not recognized as an internal or external command
- Python 脚本不记录消息
- 更改默认配置时组件配置不会更新
- awsiot.greengrasscoreipc.model.UnauthorizedError
- com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)
- com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers
- Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"
- copyFrom: <configurationPath> is already a container, not a leaf
- com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'
- java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.
- aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

Warn: '<command>' is not recognized as an internal or external command

当 Core 软件无法在组件的生命周期脚本中运行命令时 AWS IoT Greengrass ，您可能会在 Greengrass 组件的日志中看到此错误。由于此错误，组件的状态变BROKEN为。如果运行组件（例如）的系统用户在 P ATH 的文件夹中找不到该命令的可执行文件，则可能会发生此错误。ggc_user

在 Windows 设备上，检查包含可执行文件的文件夹是否位于运行该组件PATH的系统用户所在的文件夹中。如果中缺少它PATH，请执行以下任一操作：

- 将可执行文件的文件夹添加到PATH系统变量中，该变量可供所有用户使用。然后，重新启动该组件。

  如果您运行 Greengrass nucleus 2.5.0，则在更新PATH系统变量后，必须重新启动 Core 软件才能运行更新后的 AWS IoT Greengrass 组件。PATH如果 AWS IoT Greengrass Core 软件在您重新启动软件PATH后未使用更新的版本，请重新启动设备并重试。有关更多信息，请参阅 运行AWS IoT Greengrass核心软件。
- 将可执行文件的文件夹添加到运行该PATH组件的系统用户的用户变量中。

Python 脚本不记录消息

Greengrass 核心设备收集可用于识别组件问题的日志。如果您的 Python 脚本stdout和stderr消息未出现在组件日志中，则可能需要在 Python 中为这些标准输出流刷新缓冲区或禁用缓冲。执行以下任一操作：

- 使用 -u 参数运行 Python 以禁用和上的缓冲。stdout stderr

  Linux or Unix

  ```
  python3 -u hello_world.py
  ```

  Windows

  ```
  py -3 -u hello_world.py
  ```
- 在组件的配方中使用 Setenv 将 PYTHONUNBUFFERED 环境变量设置为非空字符串。此环境变量禁用和上的缓冲。stdout stderr
- 刷新stdout或stderr流的缓冲区。请执行以下操作之一：
  - 打印时刷新消息。

```
import sys

print('Hello, error!', file=sys.stderr, flush=True)
```

- 打印后刷新消息。在刷新直播之前，你可以发送多条消息。

```
import sys

print('Hello, error!', file=sys.stderr)
sys.stderr.flush()
```

有关如何验证 Python 脚本是否输出日志消息的更多信息，请参阅监控AWS IoT Greengrass日志。

更改默认配置时组件配置不会更新

当您更改组件配方DefaultConfiguration中的时，新的默认配置不会在部署期间取代该组件的现有配置。要应用新的默认配置，必须将组件的配置重置为其默认设置。部署组件时，请指定一个空字符串作为重置更新。

Console

重置路径

```
[""]
```

AWS CLI

以下命令创建对核心设备的部署。

```
aws greengrassv2 create-deployment --cli-input-json file://reset-configuration-deployment.json
```

该reset-configuration-deployment.json文件包含以下 JSON 文档。

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
```

```
      "com.example.HelloWorld": {
        "componentVersion": "1.0.0",
        "configurationUpdate": {,
          "reset": [""]
        }
      }
    }
  }
```

Greengrass CLI

以下 Greengrass CLI 命令在核心设备上创建本地部署。

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config reset-configuration-deployment.json
```

该`reset-configuration-deployment.json`文件包含以下 JSON 文档。

```
{
  "com.example.HelloWorld": {
    "RESET": [""]
  }
}
```

awsiot.greengrasscoreipc.model.UnauthorizedError

当 Greengrass 组件无权对资源执行 IPC 操作时，您可能会在 Greengrass 组件的日志中看到此错误。要授予组件调用 IPC 操作的权限，请在组件的配置中定义 IPC 授权策略。有关更多信息，请参阅 授权组件执行 IPC 操作。

> **Tip**
>
> 如果您更改组件配方`DefaultConfiguration`中的，则必须将该组件的配置重置为其新的默认配置。部署组件时，请指定一个空字符串作为重置更新。有关更多信息，请参阅 更改默认配置时组件配置不会更新。

com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"

如果多个 IPC 授权策略（包括核心设备上的所有组件）使用相同的策略 ID，则可能会看到此错误。

请检查组件的 IPC 授权策略，修复所有重复项，然后重试。要创建唯一的策略 ID，我们建议您将组件名称、IPC 服务名称和计数器组合在一起。有关更多信息，请参阅 授权组件执行 IPC 操作。

> Tip
>
> 如果您更改组件配方DefaultConfiguration中的，则必须将该组件的配置重置为其新的默认配置。部署组件时，请指定一个空字符串作为重置更新。有关更多信息，请参阅 更改默认配置时组件配置不会更新。

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)

当核心设备无法从令牌交换服务获取 AWS 凭据时，您可能会看到此错误。HTTP 400 状态代码表示之所以发生此错误，是因为核心设备的令牌交换 IAM 角色不存在或不存在允许 AWS IoT 凭证提供者担任该角色的信任关系。

执行以下操作：

1. 确定核心设备使用的令牌交换角色。错误消息包括核心设备的 AWS IoT 角色别名，该别名指向令牌交换角色。在开发计算机上运行以下命令，并*MyGreengrassCoreTokenExchangeRoleAlias*替换为错误消息中 AWS IoT 角色别名的名称。

   ```
   aws iot describe-role-alias --role-alias MyGreengrassCoreTokenExchangeRoleAlias
   ```

   响应包括令牌交换 IAM 角色的亚马逊资源名称 (ARN)。

   ```
   {
     "roleAliasDescription": {
       "roleAlias": "MyGreengrassCoreTokenExchangeRoleAlias",
       "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
   MyGreengrassCoreTokenExchangeRoleAlias",
   ```

```
    "roleArn": "arn:aws:iam::123456789012:role/MyGreengrassV2TokenExchangeRole",
    "owner": "123456789012",
    "credentialDurationSeconds": 3600,
    "creationDate": "2021-02-05T16:46:18.042000-08:00",
    "lastModifiedDate": "2021-02-05T16:46:18.042000-08:00"
  }
}
```

2. 检查该角色是否存在。运行以下命令，将 *MyGreengrassV2 TokenExchangeRole* 替换为令牌交换角色的名称。

   ```
   aws iam get-role --role-name MyGreengrassV2TokenExchangeRole
   ```

   如果命令返回`NoSuchEntity`错误，则该角色不存在，您必须创建它。有关如何创建和配置此角色的更多信息，请参阅授权核心设备与AWS服务。

3. 检查该角色是否具有允许 AWS IoT 凭证提供者担任该角色的信任关系。上一步的响应包含一个`AssumeRolePolicyDocument`，它定义了角色的信任关系。该角色必须定义一种允许担任该角色`credentials.iot.amazonaws.com`的信任关系。此文档应与以下示例类似。

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "credentials.iot.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
       }
     ]
   }
   ```

   如果角色的信任关系不`credentials.iot.amazonaws.com`允许代替，则必须将此信任关系添加到角色中。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的修改角色。

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)

当核心设备无法从令牌交换服务获取 AWS 凭据时，您可能会看到此错误。HTTP 403 状态代码表示出现此错误是因为核心设备的 AWS IoT 策略未授予核心设备 AWS IoT 角色别名的`iot:AssumeRoleWithCertificate`权限。

查看核心设备的 AWS IoT 策略，然后为核心设备的 AWS IoT 角色别名添加`iot:AssumeRoleWithCertificate`权限。错误消息包括核心设备的当前 AWS IoT 角色别名。有关此权限以及如何更新核心设备 AWS IoT 策略的更多信息，请参阅AWS IoT Greengrass V2核心设备的最低AWS IoT政策和更新核心设备的AWS IoT政策。

com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers

当组件尝试请求 AWS 凭据但无法连接到令牌交换服务时，您可能会看到此错误。

执行以下操作：

- 检查该组件是否声明了对令牌交换服务组件的依赖关系`aws.greengrass.TokenExchangeService`。如果没有，请添加依赖关系并重新部署该组件。
- 如果该组件在 docker 中运行，请确保根据应用正确的网络设置和环境变量。在 Docker 容器组件中使用AWS凭证 (Linux)
- 如果组件是用 NodeJS 编写的，请设置 dns。 setDefaultResult订购到**`ipv4first`**。
- 检查是否有`/etc/hosts`以开头`::1`并包含的条目`localhost`。删除该条目以查看它是否导致组件通过错误的地址连接到令牌交换服务。

Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"

当组件未运行令牌交换服务并且某个组件尝试请求 AWS 凭据时，您可能会看到此错误。

执行以下操作：

- 检查该组件是否声明了对令牌交换服务组件的依赖关系`aws.greengrass.TokenExchangeService`。如果没有，请添加依赖关系并重新部署该组件。

- 检查组件是否在其install生命周期中使用 AWS 凭证。 AWS IoT Greengrass 并不能保证代币交换服务在install生命周期中的可用性。更新组件以将使用 AWS 凭据的代码移至startup或run生命周期，然后重新部署该组件。

copyFrom: <configurationPath> is already a container, not a leaf

当您将配置值从容器类型（列表或对象）更改为非容器类型（字符串、数字或布尔值）时，可能会看到此错误。执行以下操作：

1. 检查组件的配方，看看其默认配置是将该配置值设置为列表还是对象。如果是，请删除或更改该配置值。
2. 创建部署以将该配置值重置为其默认值。有关更多信息，请参阅 创建部署 和 更新组件配置。

然后，您可以将该配置值设置为字符串、数字或布尔值。

com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'

当 Docker 应用程序管理器组件尝试从亚马逊 Elastic Container Registry (Amazon ECR) 的私有存储库下载 Dock er 镜像时，你可能会在 Greengrass 核心日志中看到这个错误。如果您使用 wincred Docker 凭证助手 ()docker-credential-wincred，则会发生此错误。因此，Amazon ECR 无法存储登录凭证。

采取以下操作之一：

- 如果您不使用 wincred Docker 凭证助手，请从核心设备中删除该docker-credential-wincred程序。
- 如果您使用 wincred Docker 凭证助手，请执行以下操作：
  1. 在核心设备上重命名该docker-credential-wincred程序。wincred替换为 Windows Docker 凭据助手的新名称。例如，您可以将其重命名为docker-credential-wincredreal。
  2. 更新 Docker 配置文件 (.docker/config.json) 中的credsStore选项，以使用 Windows Docker 凭据助手的新名称。例如，如果您将程序重命名为docker-credential-wincredreal，则将该credsStore选项更新为wincredreal。

```
{
  "credsStore": "wincredreal"
}
```

java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.

当运行组件进程（例如）的系统用户密码已过期时，您可能会在 Windows 核心设备上看到此错误。ggc_user因此， AWS IoT Greengrass Core 软件无法以该系统用户的身份运行组件进程。

更新 Greengrass 系统用户的密码

1. 以管理员身份运行以下命令来设置用户的密码。将 *ggc_user* 替换为系统用户，将*########*的密码。

   ```
   net user ggc_user password
   ```

2. 使用该[PsExec 实用程序](https://docs.microsoft.com/en-us/sysinternals/downloads/psexec)将用户的新密码存储在 LocalSystem 账户的凭据管理器实例中。将*##*替换为您设置的用户密码。

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

> **Tip**
>
> 根据你的 Windows 配置，用户的密码可能会设置为在将来的某个日期过期。为确保您的 Greengrass 应用程序继续运行，请跟踪密码何时过期，并在密码过期之前对其进行更新。您也可以将用户的密码设置为永不过期。
>
> - 要检查用户及其密码何时过期，请运行以下命令。
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - 要将用户的密码设置为永不过期，请运行以下命令。
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```

- 如果你使用的是已弃用该wmic命令的 Windows 10 或更高版本，请运行以下 PowerShell 命令。

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

aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

将流管理器 v2.0.7 升级到 v2.0.8 和 v2.0.11 之间的版本时，如果流管理器组件无法启动，则可能会在流管理器组件的日志中看到以下错误。

```
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}
```

如果您部署了流管理器 v2.0.7 并且想要升级到更高版本，则必须直接升级到流管理器 v2.0.12。有关直播管理器组件的更多信息，请参阅流管理器。

核心设备 Lambda 函数组件问题

对核心设备上的 Lambda 函数组件问题进行故障排除。

主题

- The following cgroup subsystems are not mounted: devices, memory
- ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

The following cgroup subsystems are not mounted: devices, memory

在以下情况下，当您运行容器化 Lambda 函数时，您可能会看到此错误：

- 核心设备没有为内存或设备 cgroup 启用 cgroup v1。
- 核心设备启用了 cgroups v2。Greengrass Lambda 函数需要 cgroups v1，cgroups v1 和 v2 是互斥的。

要启用 cgroups v1，请使用以下 Linux 内核参数启动设备。

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

Tip

在 Raspberry Pi 上，编辑`/boot/cmdline.txt`文件以设置设备的内核参数。

ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

当您在 V2 核心设备上运行使用 AWS IoT Greengrass Core SDK 的 V1 Lambda 函数，而没有在旧版订阅路由器组件中指定订阅时，您可能会看到此错误。要解决此问题，请部署和配置旧版订阅路由器以指定所需的订阅。有关更多信息，请参阅 导入 V1 Lambda 函数。

组件版本已停产

当核心设备上的某个组件版本停产时，你可能会在个人健康控制面板 (PHD) 上看到一条通知。组件版本将在停产后的 60 分钟内将此通知发送给您的 PHD。

要查看需要修改哪些部署，请使用以下方法执行以下操作 AWS Command Line Interface：

1. 运行以下命令以获取您的核心设备列表。

```
aws greengrassv2 list-core-devices
```

2. 运行以下命令以检索步骤 1 中每台核心设备上组件的状态。*coreDeviceName*替换为要查询的每台核心设备的名称。

```
aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
```

3. 收集安装了前面步骤中已停产组件版本的核心设备。
4. 运行以下命令以检索步骤 3 中每台核心设备的所有部署任务的状态。*coreDeviceName*替换为要查询的核心设备的名称。

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

   响应包含核心设备的部署任务列表。您可以修改部署以选择其他组件版本。有关如何修改部署的更多信息，请参阅修订部署。

Greengrass 命令行界面问题

使用 Greengr ass CLI 对问题进行故障排除。

主题

- java.lang.RuntimeException: Unable to create ipc client

java.lang.RuntimeException: Unable to create ipc client

当您运行 Greengrass CLI 命令并指定与安装核心软件的根文件夹不同的根文件夹时，您可能会看到此错误。 AWS IoT Greengrass

执行以下任一操作来设置根路径，并*/greengrass/v2*替换为 AWS IoT Greengrass Core 软件安装路径：

- 将 GGC_ROOT_PATH 环境变量设置为 */greengrass/v2*。
- 将--ggcRootPath */greengrass/v2*参数添加到您的命令中，如以下示例所示。

```
greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
```

AWS Command Line Interface 问题

对 AWS CLI 的问题进行故障排除 AWS IoT Greengrass V2。

主题

- Error: Invalid choice: 'greengrassv2'

Error: Invalid choice: 'greengrassv2'

当你使用运行 AWS IoT Greengrass V2 命令时 AWS CLI（例如，`aws greengrassv2 list-core-devices`），你可能会看到这个错误。

此错误表示您的版本 AWS CLI 不支持 AWS IoT Greengrass V2。要 AWS IoT Greengrass V2 与一起使用 AWS CLI，您必须拥有以下版本之一或更高版本：

- AWS CLI V1 最低版本：v1.18.197
- AWS CLI V2 最低版本：v2.1.11

> **Tip**
>
> 你可以运行以下命令来检查你拥有 AWS CLI 的版本。
>
> ```
> aws --version
> ```

要解决此问题，请将更新 AWS CLI 到支持的更高版本 AWS IoT Greengrass V2。有关更多信息，请参阅《AWS Command Line Interface 用户指南》中的安装、更新和卸载 AWS CLI。

详细的部署错误代码

在使用 Greengrass nucleus 2.8.0 或更高版本时，使用这些部分中的错误代码和解决方案来帮助解决组件部署问题。

Greengrass 核心按从最不具体的代码到最具体的可用代码的层次结构报告部署错误。您可以使用此层次结构来帮助查明部署错误的原因。例如，以下是可能的错误层次结构：

- 部署失败
  - 神器_下载_错误
    - IO_ERROR
      - 磁盘空间_关键

错误代码分为几种类型。每种类型代表一类可能发生的错误。AWS IoT Greengrass在控制台、API 和中报告这些错误类型AWS CLI。可能有多种错误类型，具体取决于错误层次结构中报告的错误。对于前面的示例，返回的错误类型是DEVICE_ERROR。

类型有：

- 权限错误— 访问需要权限的操作被拒绝。
- 请求_错误— 由于部署文档中的问题，出现错误。
- 组件_食谱_错误— 由于组件配方中的问题而发生错误。
- AWS_COMPONENT_错误— 启动或删除时出错AWS提供的组件。
- 用户_COMPONENT_错误— 启动或删除用户组件时出错。
- 组件错误— 启动或移除组件时出现错误，但是 Greengrass 核无法确定该组件是否为AWS提供的组件或用户组件。
- 设备错误— 本地 I/O 出现错误或其他设备错误。
- 依赖关系_错误— 部署未能从 Amazon S3 下载构件或从 ECR 注册表提取映像。
- HTTP_错误— HTTP 请求出现错误。
- 网络错误— 设备网络出现错误。
- NUCLES_ER— Greengrass 核找不到组件或找不到活跃的核版本。
- 服务器错误— 服务器在响应请求时返回了 500 错误。
- 云服务错误— 出现错误AWS IoT Greengrass云服务。
- 未知错误— 组件引发了未经检查的异常。

本节中的许多错误在中报告了其他信息AWS IoT Greengrass核心日志。这些日志存储在核心设备的本地文件系统上。有的日志AWS IoT Greengrass核心软件和每个单独的组件。有关访问日志的信息，请参见访问文件系统日志。

权限错误

访问_被拒绝

在以下情况下你可能会遇到这个错误AWS服务操作返回 403 错误，因为权限设置不正确。有关详细信息，请查看更具体的错误代码。

获取部署_配置_访问_被拒绝

在以下情况下你可能会遇到这个错误AWS IoT策略不允许调用`GetDeploymentConfiguration`操作。添加`greengrass::GetDeploymentConfiguration`核心设备策略的权限。

GET_COMPONENT_VERSION_ARTIFACT_ACCESS_

当核心设备出现时，你可能会遇到这个错误AWS IoT政策不允许`greengrass:GetComponentVersionArtifact`许可。将权限添加到核心设备的策略中。

RESOLVE_COMPONENT_CANDIDATES_ACCESS_

当核心设备出现时，你可能会遇到这个错误AWS IoT政策不允许`greengrass:ResolveComponentCandidates`许可。将权限添加到核心设备的策略中。

GET_ECR_CREDENTIAL_ERROR

当部署无法使用 ECR 中的私有注册表进行身份验证时，您可能会遇到此错误。检查日志中是否存在特定错误，然后再次尝试部署。

USER_NOT_AUTHORIZED_FOR_DOCKER

当 Greengrass 用户无权使用 Docker 时，你可能会遇到这个错误。确保您以 root 用户身份运行 Greengrass 或者已将该用户添加到`docker`组。然后再次尝试部署。

S3_ACCESS_DENED

当 Amazon S3 操作返回 403 错误时，您可能会遇到此错误。有关详细信息，请查看任何其他错误代码或日志。

S3_HEAD_OBJECT_ACCESS_DENE

要么当设备的代币交换角色不允许时，你可能会遇到这个错误AWS IoT Greengrass用于从组件配方中指定或组件构件不可用的 S3 对象 URL 下载组件构件的核心软件。检查代币交换角色是否允许`s3:GetObject`用于表示构件可用且存在构件的 S3 对象 URL。

S3_GET_BUCKET_LOCATION_ACCESS_DENIED

当设备的代币交换角色不允许时，你可能会遇到这个错误`s3:GetBucketLocation`对象可用的 Amazon S3 存储桶的权限。检查设备是否允许权限，然后再次尝试部署。

S3_GET_OBJECT_ACCESS_DENED

要么当设备的代币交换角色不允许时，你可能会遇到这个错误AWS IoT Greengrass用于从组件配方中指定或组件构件不可用的 S3 对象 URL 下载组件构件的核心软件。检查代币交换角色是否允许`s3:GetObject`用于表示构件可用且存在构件的 S3 对象 URL。

请求错误

NUCLES_缺失_所需能力

当部署中的 nucleus 版本无法执行所请求的操作（例如下载大型配置或设置 Linux 资源限制）时，您可能会遇到此错误。使用支持该操作的核心版本重试部署。

MULTIPLE_NUCLE_SORVED_

当部署尝试部署多个 nucleus 组件时，你可能会遇到这个错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

组件_CIRCULAR_DEPENDENCY_错误

当部署中的两个组件相互依赖时，你可能会遇到这个错误。修改组件设置，使部署中的组件不相互依赖。

未授权_NUCLES_MINOR_VERSION_UPDATE

当部署中的某个组件需要 nucleus 次要版本更新，但在部署中未指定该版本时，您可能会遇到此错误。这有助于减少依赖于不同版本的组件的意外次要版本更新。在部署中包括新的次要核心版本。

缺少_DOCKER_应用程序管理器

当你部署 Docker 组件而不部署 Docker 应用程序管理器时，你可能会遇到这个错误。确保您的部署包含 Docker 应用程序管理器。

缺少代币交换服务

当部署要在不部署令牌交换服务的情况下从私有 ECR 注册表下载 Docker 镜像构件时，你可能会遇到这个错误。确保您的部署包括令牌交换服务。

组件_版本_要求_未满足

当存在版本约束冲突或组件版本不存在时，您可能会遇到此错误。有关更多信息，请参阅Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>：

TROTTLING_ERROR

在以下情况下你可能会遇到这个错误AWS服务操作超过了速率配额。重试部署。

有冲突的请求

在以下情况下你可能会遇到这个错误AWS服务操作返回 409 错误，因为您的部署正在尝试同时执行多个操作。重试部署。

未找到资源

在以下情况下你可能会遇到这个错误AWS服务操作返回 404 错误，因为找不到资源。检查日志中是否有缺失的资源。

使用_CONFIG_运行_无效

在以下情况下你可能会遇到这个错误posixUser，posixGroup，或windowsUser为运行该组件而指定的信息无效。检查用户是否有效，然后重试部署。

不支持的区域

当为部署指定的区域不受支持时，您可能会遇到此错误AWS IoT Greengrass。检查区域，然后重试部署。

IOT_CRED_ENDPOINT_NOT_VALID

在以下情况下你可能会遇到这个错误AWS IoT配置中指定的凭证端点无效。检查终端节点，然后重试您的请求。

IOT_DATA_ENDPOINT_NOAT_VALID

在以下情况下你可能会遇到这个错误AWS IoT配置中指定的数据端点无效。检查终端节点，然后重试您的请求。

S3_HEAD_OBJECT_SOURCE_NOT_FOUN

当组件构件在组件配方中指定的 S3 对象 URL 上不可用时，您可能会遇到此错误。检查您是否将构件上传到 S3 存储桶，以及该对象 URI 是否与存储桶中该对象的 S3 对象 URL 相匹配。

S3_GET_BUCKET_LOCATION_RESOURCE_NOT_FOUND

当找不到 Amazon S3 存储桶时，你可能会遇到这个错误。检查存储桶是否存在，然后重试部署。

S3_GET_OBJECT_SOURCE_NOT_FOUND

当组件构件在组件配方中指定的 S3 对象 URL 上不可用时，您可能会遇到此错误。检查您是否将构件上传到 S3 存储桶，以及该对象 URI 是否与存储桶中该对象的 S3 对象 URL 相匹配。

IO_MAPPING_错误

解析部署文档或配方时出现 I/O 错误时，可能会出现此错误。有关详细信息，请查看任何其他错误代码或日志。

组件配方错误

RECIPE_PARSE_ERROR

当由于配方结构有错误而无法解析部署配方时，你可能会遇到这个错误。检查配方格式是否正确，然后重试部署。

RECIPE_METADATA_PARSE_ERROR

当无法解析从云端下载的部署配方元数据时，你可能会遇到这个错误。联系 AWS Support。

ARTIFACT_URI_NOT_VALID

当配方中的工件 URI 格式不正确时，你可能会遇到这个错误。检查日志中是否有无效的 URI，更新配方中的 URI，然后再次尝试部署。

S3_ARTIFACT_URI_NOT_VALID

当配方中工件的 Amazon S3 URI 无效时，你可能会遇到这个错误。检查日志中是否有无效的 URI，更新配方中的 URI，然后再次尝试部署。

DOCKER_ARTIFACT_URI_NOT_VALID

当配方中工件的 Docker URI 无效时，你可能会遇到这个错误。检查日志中是否有无效的 URI，更新配方中的 URI，然后再次尝试部署。

EMPTY_ARTIFACT_URI

如果未在配方中指定工件的 URI，则可能会出现此错误。检查日志中是否存在缺少 URI 的构件，更新配方中的 URI，然后再次尝试部署。

清空神器方案

当没有为构件定义 URI 架构时，你可能会遇到这个错误。检查日志中是否有无效的 URI，更新配方中的 URI，然后再次尝试部署。

不支持的神器方案

当正在运行的 nucleus 版本不支持 URI 架构时，你可能会遇到这个错误。要么是 URI 无效，要么你需要更新 nucleus 版本。如果 URI 无效，请检查日志中是否存在无效的 URI，更新配方中的 URI，然后再次尝试部署。

食谱_缺失_清单

当清单部分未包含在食谱中时，你可能会遇到这个错误。将清单添加到配方中，然后重试部署。

RECIPE_MISSING_ARTIFACT_HASH

如果在没有哈希算法的配方中指定了非本地构件，则可能会出现此错误。将算法添加到构件中，然后重试请求。

ARTIFACT_CHECKSUM_MIS

当下载的工件的摘要与配方中指定的摘要不同时，你可能会遇到这个错误。确保配方包含正确的摘要，然后再次尝试部署。有关更多信息，请参阅 Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.。

组件_依赖关系_无效

当部署配方中指定的依赖项类型无效时，你可能会遇到这个错误。检查食谱，然后重试您的请求。

CONFIG_INTERPOLATE_错误

插值配方变量时可能会出现此错误。有关详细信息，请查看日志。

IO_MAPPING_错误

解析部署文档或配方时出现 I/O 错误时，可能会出现此错误。有关详细信息，请查看任何其他错误代码或日志。

AWS组件错误、用户组件错误、组件错误

当组件出现问题时，将返回以下错误代码。报告的实际错误类型取决于引发错误的特定组件。如果 Greengrass 核将该组件识别为由提供的组件AWS IoT Greengrass，它会返回`AWS_COMPONENT_ERROR`。如果该组件被识别为用户组件，则 Greengrass 核将返回`USER_COMPONENT_ERROR`。如果格林格拉斯核无法分辨出来，它就会返回`COMPONENT_ERROR`。

组件_更新_错误

当组件在部署期间未更新时，你可能会遇到这个错误。检查任何其他错误代码或查看日志，看看是什么原因导致了错误。

组件_BROKEN

当组件在部署期间损坏时，你可能会遇到这个错误。检查组件日志以了解错误详细信息，然后再次尝试部署。

删除_组件_错误

当 nucleus 无法在部署期间删除组件时，你可能会遇到这个错误。检查日志以了解错误详细信息，然后再次尝试部署。

组件_BOOTSTRAP_TIMEOUT

当组件的引导任务花费的时间超过配置的超时时间时，你可能会遇到这个错误。延长超时时间或减少引导任务的执行时间，然后再次尝试部署。

COMPONENT_BOOTSTRAP_ERROR

当组件的引导任务出现错误时，你可能会遇到这个错误。检查日志以了解错误详细信息，然后再次尝试部署。

组件配置无效

当 nucleus 无法验证组件的已部署配置时，你可能会遇到这个错误。检查日志以了解错误详细信息，然后再次尝试部署。

设备错误

IO_WRITE_ERROR

写入文件时可能会出现此错误。有关详细信息，请查看日志。

IO_READ_ERROR

读取文件时可能会出现此错误。有关详细信息，请查看日志。

磁盘空间_关键

当磁盘空间不足以完成部署请求时，您可能会遇到此错误。你必须有至少 20 Mb 的可用空间，或者足以容纳更大的神器。释放一些磁盘空间，然后重试部署。

IO_FILE_ATTRIBUTE_错误

当无法从文件系统检索现有文件大小时，你可能会遇到这个错误。有关详细信息，请查看日志。

设置权限错误

当无法在已下载的构件或构件目录上设置权限时，可能会出现此错误。有关详细信息，请查看日志。

IO_UNZIP_ERROR

当无法解压缩工件时，你可能会遇到这个错误。有关详细信息，请查看日志。

本地食谱_未找到

当找不到配方文件的本地副本时，你可能会遇到这个错误。再次尝试部署。

本地食谱_已损坏

当食谱的本地副本自下载以来已更改时，您可能会遇到此错误。删除配方的现有副本，然后重试部署。

本地_RECIPE_METADATA_NOT_FOUND

当找不到配方元数据文件的本地副本时，你可能会遇到这个错误。再次尝试部署。

启动_目录_损坏

当目录用于启动 Greengrass nucleus 时，你可能会遇到这个错误 (`/greengrass/v2/alts/current`) 自上次启动原子核以来已经过修改。重启核心，然后重试部署。

哈希算法_不可用

当设备的 Java 发行版不支持所需的哈希算法或者组件配方中指定的哈希算法无效时，你可能会遇到这个错误。

DEVIC_CONFIG_NOT_VALID_FOR_ARTIFACT_DOWNOA

当设备配置中出现错误导致部署无法从 Amazon S3 或 Greengrass 云下载构件时，您可能会遇到此错误。检查日志中是否存在特定的配置错误，然后重试部署。

依赖性错误

DOCKER_ERROR

提取 Docker 镜像时可能会出现此错误。有关详细信息，请查看任何其他错误代码或日志。

DOCKER_SERVICE_S

当 Greengrass 无法登录 Docker 注册表时，你可能会遇到这个错误。检查日志中是否存在特定错误，然后再次尝试部署。

DOCKER_LOGIN_ERROR

当登录 Docker 时出现意外错误时，你可能会遇到这个错误。检查日志中是否存在特定错误，然后再次尝试部署。

DOCKER_PUL_ERROR

当从注册表中提取 Docker 镜像时出现意外错误时，你可能会遇到这个错误。检查日志中是否存在特定错误，然后再次尝试部署。

DOCKER_IMAGE_NOVALD

当请求的 Docker 镜像不存在时，你可能会遇到这个错误。检查日志中是否存在特定错误，然后重试部署。

DOCKER_IMAGE_QUERY_ERROR

当查询 Docker 以获取可用图像时出现意外故障时，您可能会遇到此错误。检查日志中是否存在特定错误，然后重试部署。

S3_错误

在下载 Amazon S3 构件时，你可能会遇到这个错误。有关详细信息，请查看任何其他错误代码或日志。

S3_RESOURCE_NOT_FOUND

当 Amazon S3 操作返回 404 错误时，您可能会遇到此错误。有关详细信息，请查看任何其他错误代码或日志。

S3_BAD_REQUEST

当 Amazon S3 操作返回 400 错误时，您可能会遇到此错误。检查日志中是否存在特定错误，然后重试请求。

HTTP 错误

HTTP_请求_错误

当发出 HTTP 请求时出现错误时，你可能会遇到这个错误。检查日志中是否存在特定错误。

下载_部署_文档_错误

当下载部署文档时出现 HTTP 错误时，您可能会遇到此错误。检查日志中是否存在特定 HTTP 错误。

GET_GREENGRASS_ARTIFACT_SIZE_错误

当获取公共组件构件的大小时出现 HTTP 错误时，你可能会遇到这个错误。检查日志中是否存在特定 HTTP 错误。

下载_GREENGRASS_ARTIFACT_ERROR

当下载公共组件构件时出现 HTTP 错误时，你可能会遇到这个错误。检查日志中是否存在特定 HTTP 错误。

网络错误

网络错误

部署期间出现连接问题时，您可能会遇到此错误。检查设备与 Internet 的连接，然后重试部署。

核错误

请求不正确

在以下情况下你可能会遇到这个错误AWS云操作返回 400 错误。查看日志，看看是哪个 API 导致了错误，然后查看 nucleus 软件更新页面以查看问题是否已在更高版本的 nucleus 中得到纠正，或者联系我们AWS Support。

NUCLES_VERSION_NOT

当核心设备找不到活跃核的版本时，你可能会遇到这个错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

核重启失败

在任何需要重启核心的部署中，如果核没有重启，你可能会遇到这个错误。查看加载器日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

已安装_组件_未找到

当 nucleus 找不到已安装的组件时，你可能会遇到这个错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

部署_文档_无效

当设备收到无效的部署文档时，您可能会遇到此错误。检查任何其他错误代码或查看日志，看看是什么原因导致了错误。

清空部署请求

当设备收到空部署请求时，您可能会遇到此错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

部署_DOCUMENT_PARSE_错误

当部署请求格式与预期格式不匹配时，您可能会遇到此错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们 AWS Support。

组件_METADATA_NOT_VALID_IN_DEPLOYMENT

当部署请求包含无效的组件元数据时，您可能会遇到此错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们 AWS Support。

启动_目录_损坏

当你将 Greengrass 设备从一个事物组移到另一个事物组，然后移回原始组，部署需要 Greengrass 重新启动时，你可能会遇到这个错误。要解决该错误，请在设备上重新创建 Greengrass 的启动目录。

有关更多信息，请参阅Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service：

服务器错误

服务器错误

在以下情况下你可能会遇到这个错误AWS服务操作返回 500 错误，因为服务现在无法处理请求。稍后重试部署。

S3_SERVER_错误

当 Amazon S3 操作返回 500 错误时，您可能会遇到此错误。有关详细信息，请查看任何其他错误代码或日志。

云服务错误

RESOLVE_COMPONENT_CANDIDATES_BAD_RES

当 Greengrass 云服务向 Greengrass 云服务发送不兼容的响应时，你可能会遇到这个错误`ResolveComponentCandidates`操作。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

已超过部署文档大小

当请求的部署文档超过最大大小配额时，您可能会遇到此错误。减小部署文档的大小，然后重试部署。

GREENGRASS_ARTIFACT_SIZE_NOT_

当 Greengrass 无法获得公共组件工件的大小时，你可能会遇到这个错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

部署_文档_无效

当设备收到无效的部署文档时，您可能会遇到此错误。检查任何其他错误代码或查看日志，看看是什么原因导致了错误。

清空部署请求

当设备收到空部署请求时，您可能会遇到此错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

部署_DOCUMENT_PARSE_错误

当部署请求格式与预期格式不匹配时，您可能会遇到此错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

组件_METADATA_NOT_VALID_IN_DEPLOYMENT

当部署请求包含无效的组件元数据时，您可能会遇到此错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

通用错误

这些一般错误没有相关的错误类型。

部署_中断

当由于 nucleus 关闭或其他外部事件而无法完成部署时，您可能会遇到此错误。有关详细信息，请查看任何其他错误代码或日志。

神器_下载_错误

当下载构件时出现问题时，你可能会遇到这个错误。有关详细信息，请查看任何其他错误代码或日志。

没有可用的组件版本

当云端或本地不存在组件版本时，或者存在依赖关系解决冲突时，你可能会遇到这个错误。有关详细信息，请查看任何其他错误代码或日志。

组件_PACKAGE_LOADING_错误

处理下载的构件时出错，你可能会遇到这个错误。有关详细信息，请查看任何其他错误代码或日志。

CLOUD_API_错误

当调用时出现错误时，你可能会遇到这个错误AWS服务 API。有关详细信息，请查看任何其他错误代码或日志。

IO_ERROR

部署期间出现 I/O 错误时，可能会出现此错误。有关详细信息，请查看任何其他错误代码或日志。

组件_更新_错误

当组件在部署期间未更新时，你可能会遇到这个错误。检查任何其他错误代码或查看日志，看看是什么原因导致了错误。

未知错误

部署失败

当由于引发未检查的异常而导致部署失败时，您可能会遇到此错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

部署类型_无效

当部署类型无效时，你可能会遇到这个错误。查看日志以查看导致错误的原因，然后查看 nucleus 软件更新页面以查看问题在更高版本的 nucleus 中是否已得到纠正，或者联系我们AWS Support。

详细的组件状态码

在使用 Greengrass nucleus 2.8.0 或更高版本时，使用这些部分中的状态代码和解决方案来帮助解决组件问题。

本主题中的许多状态在AWS IoT Greengrass核心日志中报告了其他信息。这些日志存储在核心设备的本地文件系统上。每个组件都有日志。有关访问日志的信息，请参阅访问文件系统日志。

安装错误

当运行安装脚本时出现错误时，你可能会遇到这种情况。错误代码在组件日志中报告。检查安装脚本是否有错误，然后再次部署您的组件。

安装_配置_无效

当由于食谱的`Install`部分无效而无法完成组件的安装时，你可能会遇到这个错误。检查食谱的安装部分是否有错误，然后再次尝试部署。

安装_IO_错误

当安装组件时出现 I/O 错误时，你可能会遇到这个问题。有关错误的详细信息，请检查组件错误日志。

安装_MISSING_DEFAULT_RUNWITH

当AWS IoT Greengrass无法确定安装组件时要使用的用户或组时，可能会出现此错误。检查并确保安装指南的`runWith`部分包含有效的用户或群组。

安装超时

如果安装脚本未在配置的超时时间内完成，则可能会出现此错误。要么延长食谱`Install`部分中指定的`Timeout`时间段，要么修改安装脚本以在配置的超时时间内完成。

STARTUP_ERROR

当运行启动脚本时出现错误时，你可能会遇到这种情况。错误代码在组件日志中报告。检查安装脚本是否有错误，然后再次部署您的组件。

启动_配置_无效

当由于食谱的Startup部分无效而无法完成组件的安装时，你可能会遇到这个错误。检查食谱的启动部分是否有错误，然后再次尝试部署。

STARTUP_IO_ERROR

当组件启动期间出现 I/O 错误时，你可能会遇到这种情况。有关错误的详细信息，请检查组件错误日志。

STARTUP_MISSING_DEFAULT_RUNWITH

当AWS IoT Greengrass无法确定运行组件时要使用的用户或组时，可能会出现此错误。检查并确保启动食谱的runWith部分包含有效的用户或群组。

STARTUP_TIMEOUT

当启动脚本未在配置的超时时间内完成时，您可能会遇到此错误。要么延长食谱Startup部分中指定的Timeout时间段，要么修改启动脚本以在配置的超时时间内完成。

RUN_ERROR

当运行组件脚本时出现错误时，你可能会遇到这个问题。错误代码在组件日志中报告。检查运行脚本是否有错误，然后再次部署您的组件。

RUN_MISSING_DEFAUT_RUNWITH

当AWS IoT Greengrass无法确定运行组件时要使用的用户或组时，可能会出现此错误。检查并确保您的运行食谱的runWith部分包含有效的用户或群组。

运行_配置_无效

当由于配方的Run部分无效而无法运行组件时，你可能会遇到这个错误。检查食谱的运行部分是否有错误，然后再次尝试部署。

RUN_IO_ERROR

当组件运行时出现 I/O 错误时，你可能会遇到这种情况。有关错误的详细信息，请检查组件错误日志。

运行_TIMEOUT

当运行脚本未在配置的超时时间内完成时，可能会出现此错误。要么延长食谱Run部分中指定的Timeout时间段，要么修改运行脚本以在配置的超时时间内完成。

关机_错误

当关闭组件脚本时出现错误时，你可能会遇到这种情况。错误代码在组件日志中报告。检查关机脚本是否有错误，然后再次部署您的组件。

关机_超时

当关机脚本未在配置的超时时间内完成时，您可能会遇到此错误。要么延长食谱Shutdown部分中指定的Timeout时间段，要么修改运行脚本以在配置的超时时间内完成。

标记 AWS IoT Greengrass Version 2 资源

利用标签，您可以在 AWS IoT Greengrass 中组织和管理您的资源。您可以使用标签为资源分配元数据，也可以在 IAM 策略中使用标签来定义对资源的有条件访问权限。

目前，AWS IoT 账单组或成本分配报告不支持 Greengrass 资源标签。

在 AWS IoT Greengrass V2 中使用标签

可以使用标签按用途、拥有者、环境或使用案例的任何其他分类，对 AWS IoT Greengrass 资源进行分类。如果您具有很多相同类型的资源，标签可帮助您更轻松地识别特定资源。

每个标签都包含您定义的一个键和一个可选值。例如，您可以为核心设备定义一组标签，以跟踪拥有这些设备的客户。我们建议您为每类资源创建一组可满足您的需求的标签键。通过使用一组连续的标签键，可以更轻松地管理资源。

用... 标记AWS Management Console

中的标签编辑器AWS Management Console提供了一种集中而统一的方法，以创建和管理所有AWS服务的资源的标签。有关更多信息，请参阅 Resource Editor AWS Resource Groups用户指南中的标签编辑器。

使用AWS IoT Greengrass V2 API 进行标记

您还可以使用AWS IoT Greengrass V2 API 处理标签。在创建标签之前，请注意标签限制。有关更多信息，请参阅中的标签命名和使用惯例AWS 一般参考。

- 要在创建资源时添加标签，请在资源的 `tags` 属性中定义这些标签。
- 要将标签添加到现有资源或更新标签值，请使用TagResource操作。
- 要从资源删除标签，请使用UntagResource操作。
- 要检索与资源关联的标签，请使用ListTagsForResource操作，或者描述资源并检查其`tags`属性。

下表列出了您可以使用AWS IoT Greengrass V2 API 标记的资源及其相应的`Create`和/`Describe`或`Get`操作。

可标记的 AWS IoT Greengrass V2 资源

资源	创建操作	描述或获取操作信息
核心设备	无。在设备上运行AWS IoT Greengrass Core 软件以创建核心设备。	GetCoreDevice
组件	CreateComponentVersion	DescribeComponent, GetComponent
部署	CreateDeployment	GetDeployment

使用以下操作可查看和管理支持标记的资源的标签：

- TagResource— 向资源添加标签，或更新现有标签的值。
- ListTagsForResource— 列出资源的标签。
- UntagResource— 删除资源的标签。

您可以随时为资源添加或删除标签。要更改标签键的值，请将标签添加到定义相同的键和新值的资源。新值取代了先前的值。您可以将值设为空的字符串，但不能将值设为空值。

在删除一项资源时，与该资源关联的标签也将被删除。

在 IAM 策略中使用标签

在 IAM 策略中，您可以使用资源标签控制用户访问和权限。例如，策略可以允许用户仅创建那些具有特定标签的资源。策略还可以限制用户创建或修改具有特定标签的资源。

> Note
>
> 如果您使用标签来允许或拒绝用户对资源的访问，则应拒绝用户对相同资源添加或删除这些标签的能力。否则，用户会通过修改资源标签来绕过您的限制并获得资源访问权限。

您可以在策略语句的Condition元素（也称为Condition块）中使用以下条件上下文键和值。

`greengrassv2:ResourceTag/tag-key: tag-value`

允许或拒绝带特定标签的资源上的操作。

`aws:RequestTag/tag-key: tag-value`

要求在创建或修改可标记资源时使用或不使用特定标签。

`aws:TagKeys: [tag-key, ...]`

在创建或修改可标记资源时，要求使用或不使用一组特定的标签密钥。

> **Note**
>
> IAM 策略中的条件上下文密钥和值仅适用于将可标记资源作为必需参数的操作。例如，您可以为设置基于标签的条件访问权限ListCoreDevices。

有关更多信息，请参阅 IAM 用户指南中的使用AWS资源标签控制资源访问权限和 IAM JSON 策略参考。

使用 AWS CloudFormation 创建 AWS IoT Greengrass 资源

AWS IoT Greengrass 与 AWS CloudFormation 集成，后者是一项服务，可帮助您对 AWS 资源进行建模和设置，这样您只需花较少的时间来创建和管理资源与基础设施。您创建了一个描述所有内容的模板AWS你想要的资源（例如组件版本和部署），以及AWS CloudFormation为您预置和配置这些资源。

在您使用 AWS CloudFormation 时，可重复使用您的模板来不断地重复设置您的 AWS IoT Greengrass 资源。描述您的资源一次，然后在多个 AWS 账户 和区域中反复预置相同的资源。

AWS IoT Greengrass 和 AWS CloudFormation 模板

要为 AWS IoT Greengrass 和相关服务设置和配置资源，您必须了解 AWS CloudFormation 模板。模板是 JSON 或 YAML 格式的文本文件。这些模板描述要在 AWS CloudFormation 堆栈中调配的资源。如果您不熟悉 JSON 或 YAML，可以在 AWS CloudFormation Designer 的帮助下开始使用 AWS CloudFormation 模板。有关更多信息，请参阅 AWS CloudFormation 用户指南中的什么是 AWS CloudFormation Designer？。

AWS IoT Greengrass支持在中创建组件版本和部署AWS CloudFormation. 有关更多信息（包括组件版本和部署的 JSON 和 YAML 模板示例），请参阅 JSON 和 YAML 模板示例AWS IoT Greengrass资源类型参考在里面AWS CloudFormation用户指南.

ComponentVersion 模板示例

以下是简单组件版本的 YAML 模板。为便于阅读，JSON 配方包含换行符。

```
Parameters:
  ComponentVersion:
    Type: String
Resources:
  TestSimpleComponentVersion:
    Type: AWS::GreengrassV2::ComponentVersion
    Properties:
      InlineRecipe: !Sub
        - "{\n
           \"RecipeFormatVersion\": \"2020-01-25\",\n
           \"ComponentName\": \"component1\",\n
           \"ComponentVersion\": \"${ComponentVersion}\",\n
           \"ComponentType\": \"aws.greengrass.generic\",\n
           \"ComponentDescription\": \"This\",\n
```

```
            \"ComponentPublisher\": \"You\",\n
            \"Manifests\": [\n
            {\n
              \"Platform\": {\n
              \"os\": \"darwin\"\n
            },\n
            \"Lifecycle\": {},\n
            \"Artifacts\": []\n
           },\n
           {\n
             \"Lifecycle\": {},\n
             \"Artifacts\": []\n
           }\n
          ],\n
          \"Lifecycle\": {\n
          \"install\": {\n
            \"script\": \"yuminstallpython\"\n
          }\n
         }\n
        }"
        - { ComponentVersion: !Ref ComponentVersion }
```

部署模板示例

以下是定义部署简单模板的 YAML 文件。

```
Parameters:
  ComponentVersion:
    Type: String
  TargetArn:
    Type: String
Resources:
  TestDeployment:
    Type: AWS::GreengrassV2::Deployment
    Properties:
      Components:
        component1:
          ComponentVersion: !Ref ComponentVersion
      TargetArn: !Ref TargetArn
      DeploymentName: CloudFormationIntegrationTest
      DeploymentPolicies:
        FailureHandlingPolicy: DO_NOTHING
        ComponentUpdatePolicy:
```

```
            TimeoutInSeconds: 5000
            Action: SKIP_NOTIFY_COMPONENTS
          ConfigurationValidationPolicy:
            TimeoutInSeconds: 30000
Outputs:
  TestDeploymentArn:
    Value: !Sub
      - arn:${AWS::Partition}:greengrass:${AWS::Region}:${AWS::AccountId}:deployments:
${DeploymentId}
      - DeploymentId: !GetAtt TestDeployment.DeploymentId
```

了解有关 AWS CloudFormation 的更多信息

要了解有关 AWS CloudFormation 的更多信息，请参阅以下资源：

- AWS CloudFormation
- AWS CloudFormation 用户指南
- AWS CloudFormation API 参考
- AWS CloudFormation 命令行界面用户指南

开源AWS IoT Greengrass核心软件

边AWS IoT Greengrass Version 2缘运行时（nucleus）和AWS IoT Greengrass核心软件的其他组件都是开源的。这意味着您可以查看代码以解决与应用程序的交互问题。您还可以自定义和扩展AWS IoT Greengrass核心软件，以满足您的特定软件和硬件需求。

有关AWS IoT Greengrass核心软件开源存储库的信息，请参阅上的 aws-greengrass 组织。 GitHub您对开源软件的使用受相应 GitHub 存储库中的开源许可证的约束。

您对不受开源许可约束的AWS IoT Greengrass核心软件和组件的使用受 AWS Greengrass 核心软件许可证的约束。

《AWS IoT Greengrass V2 开发者指南》的文档历史记录

下表描述了此版本的文档 AWS IoT Greengrass Version 2。

- API 版本：2020-11-30

变更	说明	日期
AWS IoT Device Tester v4.9.3 和 GGV2Q v2.5.3 已发布	适用于 AWS IoT Greengrass V2 的 IDT 4.9.3 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.5.3，并支持 Greengrass nucleus 版本 2.12.0、2.11.0、2.10.0、2.9.5。	2024 年 4 月 5 日
Greengrass CLI v2.12.4 已发布	Greengrass CLI 组件 v2.12.4 已上线。	2024 年 4 月 2 日
AWS IoT Greengrass 酷睿 v2.12.4 软件更新	此版本本提供了 Greengrass nucleus 组件的 2.12.4 版本，并更新了提供的组件。 AWS	2024 年 4 月 2 日
影子管理器 v2.3.7 已发布	影子管理器 v2.3.7 已上线。此版本修复了影子管理器在同步影子管理器期间定期记录`NullPointerException` 错误的问题。	2024 年 3 月 27 日
Moquette MQTT 3.1.1 经纪商 v2.3.6 已发布	Moquette MQTT 3.1.1 经纪商组件 v2.3.6 已上线。此版本包括常规错误修复和改进。	2024 年 3 月 27 日
本地调试控制台 v2.4.2 发布	本地调试控制台组件 v2.4.2 可用。此版本包括常规错误修复和改进。	2024 年 3 月 27 日

Lambda 管理器 v2.3.3 已发布	Lambda 管理器组件 v2.3.3 已上线。此版本包括常规错误修复和改进。	2024 年 3 月 27 日
IP 探测器 v2.1.9 已发布	IP 探测器组件 v2.1.9 现已推出。此版本将获取的 IP 步骤调整为仅发送调试日志级别的日志。	2024 年 3 月 27 日
AWS IoT 舰队配置插件 v1.2.1 发布	AWS IoT 舰队配置插件 v1.2.1 已可用。此版本修复了 Greengrass nucleus 启动期间舰队配置插件处于离线状态的问题。队列配置插件现在可以无限期地重试 MQTT 连接调用。	2024 年 3 月 27 日
AWS IoT Greengrass 酷睿 v2.12.3 软件更新	此版本提供了 Greengrass nucleus 组件的 2.12.3 版本，并更新了提供的组件。 AWS	2024 年 3 月 27 日
Greengrass CLI v2.12.3 已发布	Greengrass CLI 组件 v2.12.3 已上线。	2024 年 3 月 25 日
AWS IoT Device Tester v4.9.2 和 GGV2Q v2.5.2 已发布	适用于 AWS IoT Greengras s V2 的 IDT 4.9.2 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.5.2，并支持 Greengrass nucleus 版本 2.12.0、2.11.0、2.10 .0、2.9.5。	2024年3月18日
Lookout for Vision 边缘代理 v1.2.0 已发布	Lookout for Vision 边缘代理 v1.2.0 已上市。	2024 年 3 月 11 日

AWS IoT Greengrass 酷睿 v2.12.2 软件更新	此版本提供了 Greengrass nucleus 组件的 2.12.2 版，并更新了提供的组件。 AWS	2024 年 2 月 15 日
影子管理器 v2.3.6 已发布	影子管理器 v2.3.6 已上线。此版本修复了设备离线时通过 AWS Cloud 更新删除的阴影属性在重新连接后继续存在于本地阴影中的问题。	2024年2月14日
Lambda 启动器 v2.0.13 已发布	Lambda 启动器组件已推出 2.0.13 版本。此版本包括常规错误修复和改进。	2024年2月14日
磁盘后台处理程序 v1.0.3 已发布	磁盘后台处理程序组件 v1.0.3 已可用。此版本通过重复使用数据库连接来提高性能。	2024年2月14日
Lookout for Vision 边缘代理 v1.1.9 已发布	Lookout for Vision 边缘代理 v1.1.9 已上市。	2024 年 1 月 17 日
Greengrass 开发套件 CLI v1.6.2	Greengrass 开发套件 CLI 的 1.6.2 版本现已推出。此版本修复了 Windows gradlew.bat 由于相对路径而无法运行的问题。此版本还包含其他改进。	2024 年 1 月 16 日
新 CloudTrail 数据事件	现在，您可以记录 AWS CloudTrail 数据事件以获取有关资源操作的信息，例如获取组件或部署配置。使用这些事件深入了解 Greengrass 设备的运行情况。	2023 年 12 月 20 日
Lookout for Vision 边缘代理 v1.1.8 已发布	Lookout for Vision 边缘代理 v1.1.8 已上市。	2023 年 12 月 12 日

直播管理器 v2.1.12 已发布	直播管理器 v2.1.12 现已推出。此版本更改了 Greengrass 用于为服务调用选择一组凭据的顺序。 AWS	2023 年 12 月 8 日
MQTT bridge v2.3.1 已发布	MQTT bridge v2.3.1 已上线。此版本修复了本地 MQTT 客户端进入断开连接循环的罕见问题。	2023 年 12 月 8 日
磁盘后台处理程序 v1.0.2 已发布	磁盘后台处理程序组件 v1.0.2 已可用。此版本修复了在某些情况下无法保留 MQTT 消息格式字段的问题。	2023 年 12 月 8 日
客户端设备身份验证组件 v2.4.5 发布	客户端设备身份验证组件 v2.4.5 可用。此版本增加了对选择规则中事物名称末尾的通配符的支持，并修复了在某些情况下无法使用新的连接信息更新证书的问题。	2023 年 12 月 8 日
AWS IoT Greengrass 酷睿 v2.12.1 软件更新	此版本提供了 Greengrass nucleus 组件的 2.12.1 版本，并更新了提供的组件。 AWS	2023 年 12 月 8 日
Greengrass 开发套件 CLI v1.6.1	Greengrass 开发套件 CLI 的 1.6.1 版现已推出。此版本包含错误修复和改进。	2023 年 12 月 6 日
配方验证	添加了配方验证功能，该功能将在创建组件版本时验证组件配方。	2023 年 11 月 16 日
发布商支持的组件	AWS IoT Greengrass 现在提供发布商支持的组件。这些组件由第三方供应商开发、提供和服务。	2023 年 11 月 16 日

Greengrass 测试框架 v1.2.0 发布	Greengrass 测试框架 v1.2.0 已上线。	2023 年 11 月 15 日
Greengrass 开发套件 CLI v1.6.0	Greengrass 开发套件 CLI 的 1.6.0 版本现已推出。此版本在和命令期间添加了针对 Greengrass 配方架构的配方验证检查。`component build` `component publish`此更新可帮助开发人员在组件创建过程的早期识别其组件配方中的可操作问题。此版本还在模板中添加了可信度测试套件，可以通过`test-e2e init`命令将其下拉。该置信度测试套件包括八个通用测试，这些测试可用于和扩展以满足基本的组件测试需求。	2023 年 11 月 15 日
AWS IoT Device Tester v4.9.1 支持 Greengrass nucleus 版本 2.12.0	适用于 AWS IoT Greengrass V2 的 IDT 4.9.1 版本现在支持 Greengrass nucleus 版本 2.12.0。	2023 年 11 月 7 日
AWS IoT Greengrass 酷睿 v2.12.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.12.0 版，并更新了提供的组件。 AWS	2023 年 11 月 7 日
在 VPC 中操作 Greengrass 核心设备	可以在 VPC 中操作 Greengrass 核心设备。此功能使您无需访问公共互联网即可在 VPC 中执行部署。	2023 年 11 月 3 日
Greengrass CLI v2.12.0 已发布	Greengrass CLI 组件 v2.12.0 已上线。	2023 年 10 月 30 日

直播管理器 v2.1.10 已发布	直播管理器 v2.1.10 现已推出。此版本修复了 HTTPS 代理配置不信任 Greengrass CA 证书链的问题。	2023 年 10 月 26 日
Lambda 启动器 v2.0.12 已发布	Lambda 启动器组件已推出 2.0.12 版。此版本修复了如果之前的进程未正确停止，Lambda 启动器可能会抛出错误的问题。	2023 年 10 月 26 日
Greengrass 开发套件 CLI v1.5.0	Greengrass 开发套件 CLI 的 1.5.0 版本现已推出。此版本更新了 `excludes` build 选项识别的模式（如果`build_system` 是）`zip`。此版本现在可以识别根据通配符匹配路径名的全局模式。这允许自定义指定要从哪些目录中排除。	2023 年 10 月 26 日
Lookout for Vision 边缘代理 v1.1.7 已发布	Lookout for Vision 边缘代理 v1.1.7 已上线。	2023 年 10 月 24 日
影子管理器 v2.3.4 已发布	影子管理器 v2.3.4 现已推出。此版本增加了对空和空阴影状态文档的支持。	2023 年 10 月 18 日
日志管理器 v2.3.6 已发布	日志管理器组件 v2.3.6 已可用。	2023 年 10 月 18 日
本地调试控制台 v2.4.0 发布	本地调试控制台组件 v2.4.0 可用。	2023 年 10 月 18 日
Lambda 管理器 v2.3.1 已发布	Lambda 管理器组件 v2.3.1 已上线。	2023 年 10 月 18 日
Greengrass CLI v2.11.3 已发布	Greengrass CLI 组件 v2.11.3 已上线。	2023 年 10 月 18 日

AWS IoT Greengrass 酷睿 v2.11.3 软件更新	此版本提供了 Greengrass nucleus 组件的 2.11.3 版，并更新了提供的组件。 AWS	2023 年 10 月 18 日
安全隧道 v1.0.17 已发布	安全隧道 v1.0.17 已可用。	2023 年 10 月 4 日
Greengrass 开发套件 CLI v1.4.0	Greengrass 开发套件 CLI 的 1.4.0 版本现已推出。此版本添加了一个新`config`命令，该命令启动交互式提示以修改现有 GDK 配置文件中的字段。在继续操作之前，此版本还修改了`gdk component build` 和`gdk component publish`命令，以验证配方大小是否在 Greengrass 要求范围内（<=16000 字节）。	2023 年 10 月 2 日
Moquette MQTT 3.1.1 经纪商 v2.3.5 已发布	Moquette MQTT 3.1.1 经纪商组件 v2.3.5 已上线。此版本将 Moquette 更新到版本 0.17。	2023 年 9 月 28 日
MQTT bridge v2.3.0 已发布	MQTT bridge v2.3.0 已上线。此版本增加了对与本地 MQTT 源 AWS IoT Core 之间的桥接的 MQTT 5 支持。	2023 年 9 月 28 日
Lookout for Vision 边缘代理 v1.1.6 已发布	Lookout for Vision 边缘代理 v1.1.6 已上市。	2023 年 9 月 27 日
Lambda 管理器 v2.3.0 已发布	Lambda 管理器组件 v2.3.0 已上线。	2023 年 9 月 15 日
Lambda 启动器 v2.0.11 已发布	Lambda 启动器组件已推出 2.0.11 版。此版本支持 Lambda Manager 2.3.0。	2023 年 9 月 15 日

Moquette MQTT 3.1.1 经纪商 v2.3.4 已发布	Moquette MQTT 3.1.1 经纪商组件 v2.3.4 已上线。	2023 年 9 月 1 日
Greengrass 测试框架	GTF 是支持 end-to-end 自动化的构建块的集合。它使 AWS IoT Greengrass Version 2 内部客户能够使用与服务团队相同的测试框架来验证软件更改、自动接受和质量保证。	2023 年 8 月 11 日
AWS IoT Greengrass 酷睿 v2.11.2 软件更新	此版本提供了 Greengrass nucleus 组件的 2.11.2 版，并更新了提供的组件。 AWS	2023 年 8 月 9 日
Greengrass 开发套件 CLI v1.3.0	Greengrass 开发套件 CLI 的 1.3.0 版现已推出。此版本添加了一个新`test-e2e`命令来支持使用开放 end-to-end 测试框架测试组件。	2023 年 7 月 21 日
AWS IoT Greengrass 酷睿 v2.11.1 软件更新	此版本提供了 Greengrass nucleus 组件的 2.11.1 版，并更新了提供的组件。 AWS	2023 年 7 月 21 日
Disk spooler v1.0.0 已发布	磁盘后台处理程序组件 v1.0.0 可用。	2023 年 6 月 28 日
AWS IoT Greengrass 酷睿 v2.11.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.11.0 版本，并更新了提供的组件。 AWS	2023 年 6 月 28 日
AWS IoT Greengrass 酷睿 v2.10.3 软件更新	此版本提供了 Greengrass nucleus 组件的 2.10.3 版，并更新了提供的组件。 AWS	2023 年 6 月 21 日
AWS IoT Greengrass 酷睿 v2.10.2 软件更新	此版本提供了 Greengrass nucleus 组件的 2.10.2 版，并更新了提供的组件。 AWS	2023 年 6 月 5 日

AWS IoT Greengrass 酷睿 v2.10.1 软件更新	此版本提供了 Greengrass nucleus 组件的 2.10.1 版本，并更新了提供的组件。 AWS	2023 年 5 月 11 日
AWS IoT Greengrass 酷睿 v2.10.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.10.0 版本，并更新了提供的组件。 AWS	2023 年 5 月 9 日
SageMaker 边缘管理器已停产	亚马逊 SageMaker Edge Manager 组件将于 2024 年 4 月 26 日停产。	2023 年 4 月 28 日
AWS IoT Greengrass 酷睿 v2.9.6 软件更新	此版本提供了 Greengrass nucleus 组件的 2.9.6 版本，并更新了提供的组件。 AWS	2023 年 4 月 20 日
日志管理器 v2.3.2 已发布	日志管理器组件 v2.3.2 已可用。	2023 年 4 月 19 日
直播管理器 v2.1.4 已发布	直播管理器 v2.1.4 现已推出。此版本修复了一个问题，即单个批量中具有相同时间戳的相同属性资产的条目`ConflictingOperationException` 从 SiteWise API 返回，这会导致流管理器不断重试。此版本还将默认连接超时时间从 3 秒更新为 1 分钟。	2023 年 4 月 13 日
Greengrass 开发套件 CLI v1.2.3	Greengrass 开发套件 CLI 的 1.2.3 版本现已推出。此版本包含错误修复。	2023 年 4 月 13 日

客户端设备身份验证组件 v2.4.0 发布	客户端设备身份验证组件 v2.4.0 可用。此版本增加了对客户端设备身份验证的支持，以发布可在 Greengrass 客户端设备仪表板上显示的操作指标。	2023 年 4 月 10 日
Greengrass 开发套件 CLI v1.2.2	Greengrass 开发套件 CLI 的 1.2.2 版本现已推出。此版本包含改进和错误修复。	2023 年 4 月 7 日
AWS IoT Greengrass 酷睿 v2.9.5 软件更新	此版本提供了 Greengrass nucleus 组件的 2.9.5 版，并更新了提供的组件。 AWS	2023 年 3 月 30 日
直播管理器 v2.1.3 已发布	直播管理器 v2.1.3 现已推出。此版本修复了以 SYSTEM 用户身份运行时在 Windows 操作系统上出现的启动问题。	2023 年 3 月 7 日
modbus-RTU 协议适配器 v2.1.5 已发布	modbus-RTU 协议适配器组件 v2.1.5 已上线。此版本修复了`ReadDiscreteInput` 操作中的一个问题。	2023 年 3 月 7 日
客户端设备身份验证组件 v2.3.2 发布	客户端设备身份验证组件 v2.3.2 可用。此版本增加了对缓存主机名信息的支持，以便该组件在脱机时重新启动时正确生成证书主题。	2023 年 3 月 7 日
AWS IoT Device Tester v4.7.0 支持 Greengrass nucleus 版本 2.9.4	适用于 AWS IoT Greengrass V2 的 IDT 4.7.0 版本现在支持 Greengrass nucleus 版本 2.9.4。	2023 年 3 月 2 日
Greengrass 命令行界面 v1.2.0 发布	Greengrass 命令行界面 v1.2.0 已上线。	2023 年 2 月 28 日

AWS IoT Greengrass 酷睿 v2.9.4 软件更新	此版本提供了 Greengrass nucleus 组件的 2.9.4 版，并更新了提供的组件。 AWS	2023 年 2 月 24 日
影子管理器 v2.3.1 已发布	影子管理器 v2.3.1 现已推出。此版本修复了可能导致云影更新无法同步的情况。此版本还修复了命名阴影同步配置的更改仅适用于一个命名阴影的问题。	2023 年 2 月 21 日
AWS IoT Device Tester v4.7.0 支持 Greengrass nucleus 版本 2.9.3	适用于 AWS IoT Greengrass V2 的 IDT 4.7.0 版本现在支持 Greengrass nucleus 版本 2.9.3。	2023 年 2 月 9 日
IAM 最佳实践已更新	更新了指南，使其符合 IAM 最佳实践。有关更多信息，请参阅 IAM 安全最佳实践。	2023 年 2 月 3 日
AWS IoT Greengrass 酷睿 v2.9.3 软件更新	此版本提供了 Greengrass nucleus 组件的 2.9.3 版，并更新了提供的组件。 AWS	2023 年 2 月 1 日
日志管理器 v2.3.1 已发布	日志管理器 v2.3.1 已可用。	2023 年 1 月 27 日
AWS IoT Device Tester v4.7.0 支持 Greengrass nucleus 版本 2.9.2	适用于 AWS IoT Greengrass V2 的 IDT 4.7.0 版本现在支持 Greengrass nucleus 版本 2.9.2。	2023 年 1 月 3 日
Shadow Manager v2.3.0 发布	影子管理器 v2.3.0 已上线。此版本修复了设备将 Greengrass 设备私钥存储在硬件安全模块中时可能无法同步阴影的问题。	2022 年 12 月 29 日

AWS IoT 舰队配置插件 v1.2.0 发布	AWS IoT 舰队配置插件 v1.2.0 已推出。此版本增加了对通过带有可配置私钥路径的证书签名请求进行设备配置的支持。	2022 年 12 月 22 日
AWS IoT Greengrass 酷睿 v2.9.2 软件更新	此版本提供了 Greengrass nucleus 组件的 2.9.2 版，并更新了提供的组件。 AWS	2022 年 12 月 22 日
AWS IoT Device Tester v4.7.0 和 GGV2Q v2.5.0 已发布	适用于 AWS IoT Greengrass V2 的 IDT 4.7.0 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.5.0，支持 Greengrass nucleus 版本 2.9.1、2.9.0、2.8.1、2.8.0、2.7.0、2.7.0 和 2.6.0。	2022 年 12 月 13 日
影子管理器 v2.2.4 已发布	修复了更新本地影子文档时阴影大小验证与云不一致的问题。这还修复了部署在配置节点RESET上执行时，影子管理器会停止监听配置更新的问题。	2022 年 12 月 8 日
Lookout for Vision Edge Agent 1.1.1 已发布	Lookout for Vision Edge Agent 组件 v1.1.1 已上线。	2022 年 12 月 5 日
日志管理器 v2.3.0 已发布	日志管理器组件 v2.3.0 已可用。	2022 年 11 月 18 日
AWS IoT Device Tester v4.5.11 支持 Greengrass nucleus 版本 2.9.1	适用于 AWS IoT Greengrass V2 的 IDT 4.5.11 版本现在支持 Greengrass nucleus 版本 2.9.1。	2022 年 11 月 18 日

AWS IoT Greengrass 酷睿 v2.9.1 软件更新	此版本提供了 Greengrass nucleus 组件的 2.9.1 版，并更新了提供的组件。 AWS	2022 年 11 月 18 日
AWS IoT Device Tester v4.5.11 支持 Greengrass nucleus 版本 2.9.0	适用于 AWS IoT Greengrass V2 的 IDT 4.5.11 版本现在支持 Greengrass nucleus 版本 2.9.0。	2022 年 11 月 17 日
直播管理器 v2.1.2 已发布	直播管理器 v2.1.2 现已推出。此版本修复了 Windows 操作系统上使用非英语语言的问题。	2022 年 11 月 15 日
AWS IoT Greengrass 酷睿 v2.9.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.9.0 版，并更新了提供的组件。 AWS	2022 年 11 月 15 日
AWS IoT Device Tester v4.5.11 支持 Greengrass nucleus 版本 2.8.1	适用于 AWS IoT Greengrass V2 的 IDT 4.5.11 版本现在支持 Greengrass nucleus 版本 2.8.1。	2022 年 10 月 19 日
AWS IoT Device Tester v4.5.11 和 GGV2Q v2.4.1 已发布	适用于 AWS IoT Greengrass V2 的 IDT 4.5.11 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.4.1，并支持 Greengrass nucleus 版本 2.8.0、2.7.0 和 2.6.0。	2022 年 10 月 13 日
AWS IoT Greengrass 酷睿 v2.8.1 软件更新	此版本提供了 Greengrass nucleus 组件的 2.8.1 版，并更新了提供的组件。 AWS	2022 年 10 月 13 日
AWS IoT Greengrass 酷睿 v2.8.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.8.0 版，并更新了提供的组件。 AWS	2022 年 10 月 7 日

增加了对部署的 AWS CloudFormation 支持	AWS CloudFormation 现在支持将 AWS IoT Greengrass 部署作为资源。	2022 年 10 月 6 日
SageMaker 边缘管理器 v1.3.0 发布	亚马逊 SageMaker Edge Manager 组件 v1.3.0 已上线。此版本增加了对该组件的支持，用于设置 TensorRT 模型缓存的磁盘大小，并改进了预测并发性，以更好地利用 GPU 等设备加速器引擎。	2022 年 9 月 1 日
使用进程间通信 (IPC) 客户端 V2	添加了有关 IPC 客户端 V2 的信息，它减少了使用 IPC 操作所需编写的代码量，并有助于避免 IPC 客户端 V1 可能出现的常见错误。	2022 年 8 月 12 日
AWS IoT Device Tester v4.5.8 和 GGV2Q v2.4.0 已发布	适用于 AWS IoT Greengrass V2 的 IDT 4.5.8 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.4.0，并支持 Greengrass nucleus 版本 2.7.0、2.6.0 和 2.5.6。	2022 年 8 月 12 日
SageMaker 边缘管理器 v1.2.0 发布	亚马逊 SageMaker Edge Manager 组件 v1.2.0 已上市。此版本增加了对该组件的支持，可自动检索您上传到 Amazon S3 的 SageMaker Neo 编译模型，因此您无需创建 AWS IoT Greengrass 部署即可部署新模型。	2022 年 8 月 3 日

AWS IoT Device Tester v4.5.3 支持 Greengrass nucleus 版本 2.7.0	适用于 AWS IoT Greengrass V2 的 IDT 4.5.3 版本现在支持 Greengrass nucleus 版本 2.7.0。	2022 年 8 月 1 日
直播管理器 v2.1.0 已发布	直播管理器 v2.1.0 现已推出。此版本支持您将遥测指标发送到 Amazon EventBridge。	2022 年 7 月 28 日
AWS IoT Greengrass 酷睿 v2.7.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.7.0 版本，并更新了提供的组件。 AWS它包括支持您将遥测指标发送到 Amazon EventBridge。	2022 年 7 月 28 日
物联网 SiteWise 发行商 v2.2.0 发布	物联网 SiteWise 发布者组件 v2.2.0 现已推出。此版本更新了组件，以便在将数据发送到 AWS IoT SiteWise 服务之前对其进行压缩，从而将带宽使用量减少多达 75%。	2022 年 7 月 19 日
教程：开发一个与客户端设备影子交互的组件	在 “教程：通过 MQTT 与本地 IoT 设备交互” 中添加了一个新模块，您可以按照该模块学习如何开发与客户端设备影子交互的组件。	2022 年 7 月 18 日
选择本地 MQTT 经纪商	添加了有关如何选择客户端设备连接到核心设备的本地 MQTT 代理的信息。	2022 年 7 月 18 日
AWS IoT Device Tester v4.5.3 支持 Greengrass nucleus 版本 2.6.0	适用于 AWS IoT Greengrass V2 的 IDT 4.5.3 版本现在支持 Greengrass nucleus 版本 2.6.0。	2022 年 6 月 29 日

AWS IoT Greengrass 酷睿 v2.6.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.6.0 版，并更新了提供的组件。 AWS它包括对客户端设备影子的支持和对客户端设备的本地 MQTT 5 代理的支持。它还支持本地发布/订阅主题中的通配符、组件配置中的配方变量以及 IPC 授权策略中的通配符。这些功能使您能够更轻松地开发和配置部署到核心设备队列的组件。此版本还支持使用管理本地部署的 IPC 操作的组件和核心设备上的组件。	2022 年 6 月 27 日
客户端设备组件更新	客户端设备身份验证 v2.1.0、MQTT Broker (Moquette) v2.1.0、MQTT bridg e v2.1.1 和 IP 探测器 v2.1.2 现已推出。此版本改进了证书轮换，提高了 MQTT 代理性能，并修复了这些组件如何处理配置重置更新的问题。	2022 年 6 月 14 日
AWS IoT Device Tester v4.5.3 支持 Greengrass nucleus 版本 2.5.6	适用于 AWS IoT Greengrass V2 的 IDT 4.5.3 版本现在支持 Greengrass nucleus 版本 2.5.6。	2022 年 6 月 1 日
AWS IoT Greengrass 酷睿 v2.5.6 软件更新	此版本提供了 Greengrass nucleus 组件的 2.5.6 版，并更新了提供的组件。 AWS它包括对带有 ECC 密钥的硬件安全模块的支持。它还包括其他错误修复和改进。	2022 年 5 月 31 日

AWS IoT 舰队配置插件 v1.1.0 发布	AWS IoT 舰队配置插件 v1.1.0 已推出。当您在 Windows 设备上配置插件时，此版本增加了对其他文件路径格式的支持。	2022 年 5 月 12 日
新的 Lambda 运行时已发布	增加了对新 Lambda 运行时的支持：Python 3.9、Java 11 和 NodeJS 14。	2022 年 5 月 10 日
开发一个可以延迟组件更新的 Greengrass 组件	添加了一个教程，您可以按照该教程来学习如何开发可延迟部署组件更新的 Greengrass 组件。例如，当设备电池电量不足或运行无法中断的进程时，您可能需要延迟更新。	2022 年 5 月 4 日
CloudWatch 指标 v3.1.0 和 AWS IoT Device Defender v3.1.0 已发布	CloudWatch 指标组件 v3.1.0 和 AWS IoT Device Defender 组件 v3.1.0 可用。这些版本增加了对 HTTPS 网络代理配置的支持。有关更多信息，请参阅通过端口 443 或通过网络代理连接和启用核心设备信任 HTTPS 代理。	2022 年 4 月 27 日
迁移自 AWS IoT Greengrass Version 1	添加了从 AWS IoT Greengrass V1 迁移到时可以遵循的指南 AWS IoT Greengrass V2。	2022 年 4 月 26 日

AWS IoT Device Tester v4.5.3 更新了 GGV2Q v2.3.1，支持的版本中添加了带有 GGV2Q v2.3.0 的 IDT v4.5.1	带有 AWS IoT Greengrass V AWS IoT Greengrass 2 资格套件 (GGV2Q) v2.3.1 的 IDT 版本 4.5.3 已更新，包括对 Greengrass nucleus 版本 2.5.5、2.5.4 和 2.5.3 的支持。此更新还包括以 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.3.0 作为支持版本的 IDT 4.5.1。带有 AWS IoT Greengrass V2 资格套件的 IDT 4.5.1 (GGV2Q) v2.3.0 支持 Greengrass nucleus 版本 2.5.3。	2022 年 4 月 25 日
modbus-RTU 协议适配器 v2.1.0 发布	modbus-RTU 协议适配器组件 v2.1.0 已上线。此版本添加了新的参数，您可以指定这些参数来配置与 Modbus RTU 设备的串行通信。	2022 年 4 月 20 日
CloudWatch 指标 v2.1.0、Firehose v2.1.0 和亚马逊 SNS v2.1.0 发布	CloudWatch 指标组件 v2.1.0、Firehose 组件 v2.1.0 和亚马逊 SNS 组件 v2.1.0 现已推出。这些版本增加了对 HTTPS 网络代理配置的支持。有关更多信息，请参阅通过端口 443 或通过网络代理连接和启用核心设备信任 HTTPS 代理。	2022 年 4 月 19 日

AWS IoT Device Tester v4.5.3 已发布 GGV2Q v2.3.1	适用于 AWS IoT Greengrass V2 的 IDT 4.5.3 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.3.1，并支持 Greengrass nucleus 版本 2.5.5。	2022 年 4 月 15 日
AWS IoT Greengrass 酷睿 v2.5.5 软件更新	此版本提供了 Greengrass nucleus 组件的 2.5.5 版，并更新了提供的组件。 AWS它增加了对使用非英语显示语言的 Windows 设备的支持。它还修复了在某些情况下核心设备在配置后未向 AWS IoT Greengrass 云服务报告其状态的问题。	2022 年 4 月 6 日
AWS IoT Greengrass 酷睿 v2.5.4 软件更新	此版本提供了 Greengrass nucleus 组件的 2.5.4 版，并更新了提供的组件。 AWS它包括错误修复和改进。	2022 年 3 月 23 日
AWS IoT Device Tester 以编程方式下载	添加了有关如何以 AWS IoT Greengrass V2 编程方式下载 IDT 的信息。	2022 年 3 月 15 日
Greengrass 开发套件 CLI v1.1.0	Greengrass 开发套件 CLI 的 1.1.0 版现已推出。此版本为`component init`和`component publish`命令添加了新的参数。如果组件未构建，则此版本还会更新构建该组件的`component publish`命令。	2022 年 2 月 24 日

Shadow Manager v2.1.0 发布	影子管理器组件 v2.1.0 已推出。此版本增加了配置组件与阴影同步的时间间隔的 AWS IoT Core选项。例如，您可以指定更长的间隔以减少带宽使用量和费用。	2022 年 2 月 3 日
适用于 AWS IoT Greengrass 核心软件 v2.5.3 的 Dockerfile 和 Docker 镜像	适用于 AWS IoT Greengrass 核心软件 v2.5.3 的 Dockerfile 和 Docker 镜像现已推出。	2022 年 1 月 12 日
AWS IoT Device Tester v4.5.1，GGV2Q v2.3.0 已发布	适用于 AWS IoT Greengrass V2 的 IDT 4.5.1 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.3.0，并支持验证和鉴定基于 Linux 的设备，这些设备使用硬件安全模块 (HSM) 来存储酷睿软件使用的私钥和证书。 AWS IoT Greengrass	2022 年 1 月 11 日
AWS IoT Greengrass 酷睿 v2.5.3 软件更新	此版本提供了 Greengrass nucleus 组件的 2.5.3 版，并更新了提供的组件。 AWS它支持您将 AWS IoT Greengrass 核心软件配置为使用安全存储在硬件安全模块 (HSM) 中的私钥和证书。	2022 年 1 月 6 日
适用于 AWS IoT Greengrass 核心软件 v2.5.2 的 Dockerfile 和 Docker 镜像	适用于 AWS IoT Greengrass 核心软件 v2.5.2 的 Dockerfile 和 Docker 镜像现已推出。	2021 年 12 月 20 日

AWS IoT Device Tester v4.4.1 随着 GGV2Q v2.2.1 的发布	适用于 AWS IoT Greengrass V2 的 IDT 版本 4.4.1 现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.2.1，并支持 Greengrass nucleus 版本 2.5.2 进行设备认证。	2021 年 12 月 12 日
使用 Amazon Lookout for Vision 进行机器学习推理	添加了有关如何在 Greengrass 核心设备上使用 Lookout for Vision 执行机器学习推理的信息。Lookout for Vision 使用计算机视觉来发现工业产品中的视觉缺陷。	2021 年 12 月 8 日
AWS IoT Device Tester v4.4.1 随着 GGV2Q v2.2.0 的发布	适用于 AWS IoT Greengrass V2 的 IDT 版本 4.4.1 现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.2.0，并支持 Greengrass nucleus 版本 2.5.2 进行设备认证。	2021 年 12 月 6 日
AWS IoT Greengrass 酷睿 v2.5.2 软件更新	此版本提供了 Greengrass nucleus 组件的 2.5.2 版，并更新了提供的组件。 AWS它修复了 Greengrass 核心更新后出现的 Windows 服务问题。它还包括在 Windows 设备上对该 AWS IoT Device Defender 组件的支持。	2021 年 12 月 3 日

Kinesis Video Streams 组件的新边缘连接器	Kinesis Video Streams Video Streams 组件的边缘连接器版本 1.0.0 已上市。此 AWS提供的内容读取来自本地摄像机的视频源并将直播发布到 Kinesis Video Streams。此组件与集成 AWS IoT TwinMaker，使您能够在 Grafana 仪表板中查看和管理视频流和其他数据。	2021 年 11 月 30 日
使用管理 Greengrass 核心设备 AWS Systems Manager	添加了有关如何使用管理 Greengrass 核心设备的信息。 AWS Systems Manager Systems Manager 是一项 AWS 服务，使您能够查看操作数据、自动执行操作任务以及维护安全与合规性。	2021 年 11 月 29 日
Greengrass 开发套件 CLI	添加了有关 AWS IoT Greengrass 开发套件命令行接口 (GDK CLI) 的信息，该工具可以下载到本地开发计算机上，以帮助您开发自定义 Greengrass 组件。您可以使用 GDK CLI 来创建、构建和发布自定义组件。	2021 年 11 月 29 日
社区提供的 Greengrass 组件	添加了有关 Greengrass 软件目录的信息，该目录是 Greengrass 社区开发的 Greengrass 组件的索引。您可以从该目录中下载、修改和部署组件来创建 Greengrass 应用程序。	2021 年 11 月 29 日

AWS IoT Greengrass 酷睿 v2.5.1 软件更新	此版本提供了 Greengrass nucleus 组件的 2.5.1 版，并更新了提供的组件。 AWS它包括对 Windows 设备上的 32 位 Java 的支持。它还修复了新事物组删除行为和在 Windows 设备上加载系统环境变量的问题。	2021 年 11 月 23 日
AWS IoT Device Tester v4.0 和 GGV2Q v2.1.0 已发布	适用于 AWS IoT Greengrass V2 的 IDT 4.4.0 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.1.0，并支持对运行 Greengrass nucleus 版本 2.5.0 的基于 Windows 的 Greengrass 设备进行资格认证。	2021 年 11 月 19 日
AWS IoT Greengrass 酷睿 v2.5.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.5.0 版本，并更新了提供的组件。 AWS它包括对在 Windows 设备上运行 AWS IoT Greengrass 核心软件的支持。它还会更改事物组的移除行为并添加对 HTTPS 代理的支持。	2021 年 11 月 12 日
SageMaker 边缘管理器 v1.1.0 发布	亚马逊 SageMaker Edge Manager 组件 v1.1.0 已上线。此版本增加了对运行 Amazon Linux 2 的 Greengrass 核心设备的支持，并添加了一个新的配置参数来指定设备上捕获数据文件夹的位置。	2021 年 11 月 3 日

防止跨服务混淆代理更新	AWS IoT Greengrass V2 支持在 IAM 资源策略中使用aws:SourceArn 和aws:SourceAccount 全局条件上下文密钥来防止混淆副手问题。	2021 年 11 月 1 日
客户端设备组件更新	客户端设备身份验证 v2.0.3、IP 检测器 v2.1.0、MQTT bridge v2.1.0 和 MQTT Broker (Moquette) v2.0.2 现已推出。此版本增加了对非默认 MQTT 代理端口的全面支持，并包括其他错误修复和改进。	2021 年 10 月 28 日
影子管理器 v2.0.4 已发布	影子管理器组件 v2.0.4 已可用。此版本修复了导致影子管理器删除以前删除的所有影子的新创建版本的问题。从此版本开始，DeleteThingShadow IPC 操作会增加影子版本。	2021 年 10 月 20 日
日志管理器 v2.2.0 已发布	日志管理器组件 v2.2.0 可用。日志管理器现在支持使用配置映射来提供组件日志配置。	2021 年 10 月 20 日
Lambda 管理器 v2.1.4 已发布	Lambda 管理器组件 v2.1.4 已上线。此版本修复了导致使用 NodeJS 运行时的 Lambda 函数仅处理一条消息的问题。	2021 年 10 月 20 日
在 Docker 容器组件中使用进程间通信、 AWS 凭证和流管理器	添加了有关如何在自定义 Docker 容器组件中使用进程间通信 (IPC)、 AWS 凭据和流管理器的信息。	2021 年 10 月 19 日

新的 nucleus 遥测发射器组件	nucleus 遥测发射器组件的 1.0.0 版本现已推出。此 AWS 提供的组件收集系统运行状况遥测数据，并将其持续发布到本地主题和 MQTT 主题。AWS IoT Core	2021 年 9 月 30 日
允许设备流量通过代理或防火墙	添加了有关 Greengrass 核心设备使用的端点和端口的信息，因此您可以限制流量作为安全措施。	2021 年 9 月 16 日
AWS IoT Device Tester v4.2.0 和 GGV2Q v2.0.1 已发布	适用于 V2 的 IDT 4.2.0 版本已更新为 AWS IoT Greengrass V AWS IoT Greengrass 2 资格套件 (GGV2Q) v2.0.1。此版本支持 Greengrass nucleus 版本 2.4.0 进行设备认证。	2021 年 8 月 31 日
更新了机器学习安装程序组件	DLR 安装程序组件 v1.6.5 和 TensorFlow Lite 安装程序组件 v2.5.4 现已推出。这些组件版本包括新的`UseInstaller` 配置参数，允许您禁用默认安装脚本。	2021 年 8 月 30 日
嵌入式 Linux 支持 AWS IoT Greengrass	的 BitBake 配方 AWS IoT Greengrass V2 可在上的`meta-aws`项目中找到 GitHub。你可以使用这个配方通过 Yocto Project 来构建基于 Linux 的自定义操作系统。	2021 年 8 月 20 日

代码完整性	添加了有关如何 AWS IoT Greengrass V2 验证 Greengrass 核心设备从中下载的软件完整性的信息。 AWS Cloud	2021 年 8 月 19 日
VPC 端点 (AWS PrivateLink)	AWS IoT Greengrass 现在支持 AWS IoT Greengrass 控制平面的接口 VPC 终端节点 (AWS PrivateLink)。您可以在 VPC 和 AWS IoT Greengrass 控制平面之间建立私有连接。	2021 年 8 月 16 日
直播管理器 v2.0.12 已发布	直播管理器 v2.0.12 现已推出。此版本修复了导致无法从流管理器组件的 2.0.7 版本升级到 v2.0.8 和 v2.0.11 之间的版本的问题。	2021 年 8 月 10 日
适用于 AWS IoT Greengrass 核心软件 v2.4.0 的 Dockerfile 和 Docker 镜像	适用于 AWS IoT Greengrass 核心软件 v2.4.0 的 Dockerfile 和 Docker 镜像现已推出。	2021 年 8 月 9 日
AWS IoT Greengrass 酷睿 v2.4.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.4.0 版，并更新了提供的组件。 AWS它包括对组件系统资源限制、用于暂停和恢复组件的 IPC 操作以及配置插件的支持。	2021 年 8 月 3 日
新 AWS IoT SiteWise 组件	添加了以下 AWS提供的组件 AWS IoT SiteWise：IoT SiteWise OPC-UA 收集器、物联网 SiteWise发布器和物联网处理器。 SiteWise	2021 年 7 月 29 日

AWS IoT Device Tester v4.2.0 和 GGV2Q v2.0.0 已发布	适用于 AWS IoT Greengrass V2 的 IDT 4.2.0 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v2.0.0，并支持针对 Docker 组件、机器学习和流管理器的可选资格测试。	2021 年 7 月 14 日
AWS IoT Greengrass C++ v2 中 AWS IoT Device SDK 提供了核心 IPC 库	AWS IoT Device SDK 适用于 C++ v2 的 1.13.0 版本支持 C AWS IoT Greengrass ore IPC，因此你可以用 C++ 开发与核心软件交互的组件。 AWS IoT Greengrass	2021 年 7 月 14 日
SageMaker 边缘管理器组件 v1.0.2 发布	亚马逊 SageMaker Edge Manager 组件 v1.0.2 已上线。此版本更新了组件生命周期中的安装脚本。在部署此组件之前，您的核心设备现在必须在设备上安装 Python 3.6 或更高版本（包括`pip`您的 Python 版本）。	2021 年 7 月 12 日
适用于 AWS IoT Greengras s V2 的 Su AWS IoT Device Tester pport 更新	AWS IoT Greengrass V2 版本 4.1.0 的 IDT 现在支持使用 Greengrass nucleus 版本 2.3.0 进行设备认证。	2021 年 7 月 8 日
适用于 AWS IoT Greengrass 核心软件 v2.3.0 的 Dockerfile 和 Docker 镜像	适用于 AWS IoT Greengrass 核心软件 v2.3.0 的 Dockerfile 和 Docker 镜像现已推出。	2021 年 7 月 7 日
AWS 托管策略	添加了有关 AWS 托管策略的信息 AWS IoT Greengrass。	2021 年 7 月 2 日

新推荐的 JVM 选项	添加了有关用于控制 C AWS IoT Greengrass ore 软件内存分配的推荐的 JVM 选项的信息。	2021 年 6 月 30 日
AWS IoT Greengrass 酷睿 v2.3.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.3.0 版本，并更新了提供的组件。 AWS它包括对部署中的大型组件配置文档的支持。	2021 年 6 月 29 日
适用于 AWS IoT Greengrass 核心软件 v2.2.0 的 Dockerfile 和 Docker 镜像	适用于 AWS IoT Greengrass 核心软件 v2.2.0 的 Dockerfile 和 Docker 镜像现已上市。	2021 年 6 月 28 日
AWS IoT Device Tester v4.1.0，GGV2Q v1.1.1 已发布	适用于 AWS IoT Greengrass V2 的 IDT 4.1.0 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v1.1.1，并支持使用 Greengrass nucleus v2.2.0、v2.1.0 和 v2.0.5 进行设备认证。	2021 年 6 月 18 日
AWS IoT Greengrass 酷睿 v2.2.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.2.0 版，并更新了提供的组件。 AWS它包括一些组件，您可以部署这些组件来增加对客户端设备的支持并添加本地影子服务。	2021 年 6 月 18 日
Lambda 启动器 v2.0.6 已发布	Lambda 启动器组件已推出 2.0.6 版。此版本包括性能改进和错误修复。	2021 年 6 月 13 日

新的 SageMaker 边缘管理器组件已发布	亚马逊 SageMaker Edge Manager 组件的 1.0.0 版本可用于。 AWS IoT Greengrass 此组件在 Green SageMaker grass 核心设备上安装 Edge Manager 代理二进制文件。	2021 年 6 月 10 日
组件类型	在中添加了有关组件类型的信息 AWS IoT Greengras s。组件类型指定 AWS IoT Greengrass Core 软件如何运行组件。	2021 年 6 月 3 日
AWS IoT Device Tester v4.0.2，GGV2Q v1.1.0 已发布	适用于 AWS IoT Greengras s V2 的 IDT 4.0.2 版本现已推出。此版本包括 AWS IoT Greengrass V2 资格套件 (GGV2Q) v1.1.0，并支持使用 Greengrass nucleus v2.1.0 和 Greengrass CLI v2.1.0 进行设备认证。这还包括 MQTT 和 Lambda 所需的新测试组，以及其他小错误修复和改进。	2021 年 5 月 5 日
适用于 AWS IoT Greengrass 核心软件 v2.1.0 的 Dockerfile 和 Docker 镜像	适用于 AWS IoT Greengrass 核心软件 v2.1.0 的 Dockerfile 和 Docker 镜像现已推出。Do cker 镜像使您能够在使用亚马逊 Linux 2 作为基本操作系统的 Docker 容器中运行 AWS IoT Greengrass 核心软件。	2021 年 4 月 27 日

AWS IoT Greengrass 酷睿 v2.1.0 软件更新	此版本提供了 Greengrass nucleus 组件的 2.1.0 版，并更新了提供的组件。 AWS它包括一个可用于从私有 Amazon ECR 存储库下载 Docker 映像的新组件，以及用于使用 Lite 执行机器学习推断的新示例组件。 TensorFlow	2021 年 4 月 26 日
使用 Secrets Manager 的示例组件	添加了一个示例组件，用于打印部署到核心设备的 AWS Secrets Manager 密钥的值。	2021 年 4 月 8 日
Greengrass 核心设备的最低 AWS IoT 政策	添加了有关在核心设备上支持 Greengrass 基本功能所需的最低权限集的信息。	2021 年 4 月 2 日
订阅 IPC 事件直播	添加了有关如何使用进程间通信 (IPC) 操作在 Greengrass 核心设备上订阅事件流的信息。	2021 年 4 月 1 日
的 Suppor AWS IoT Device Tester t 更新 AWS IoT Greengrass	AWS IoT Greengrass V2 版本 4.0.1 的 IDT 现在支持使用 Greengrass nucleus 版本 2.0.5 和 Greengrass CLI 版本 2.0.5 进行设备认证。	2021 年 3 月 17 日
创建使用流管理器的自定义组件	添加了有关如何配置组件配方和构件以开发管理数据流的应用程序的信息。	2021 年 3 月 9 日
AWS IoT Greengrass 酷睿 v2.0.5 软件更新	此版本提供了 Greengrass nucleus 组件的 2.0.5 版，并更新了提供的组件。 AWS它修复了网络代理支持问题和中国地区Greengrass数据平面端点的问题。 AWS	2021 年 3 月 9 日

组件环境变量引用	添加了有关 AWS IoT Greengrass 核心软件为组件设置的环境变量的信息。你可以使用这些环境变量来获取事物名称 AWS 区域、和 Greengrass nucleus 版本。	2021 年 2 月 23 日
手动安装	添加了有关如何手动创建所需 AWS 资源或如何安装在防火墙或网络代理后面的信息。通过使用手动安装，您无需向安装程序授予在中创建资源的权限 AWS 账户，因为您可以创建所需的资源 AWS IoT 和 IAM 资源。您也可以将设备配置为通过端口 443 或通过网络代理进行连接。	2021 年 2 月 17 日
AWS IoT Greengrass Python v2 的 AWS IoT Device SDK 核心 IPC 库已更新	AWS IoT Device SDK 适用于 Python v2 的 1.5.4 版简化了连接 AWS IoT Greengrass 核心 IPC 服务所需的步骤。	2021 年 2 月 11 日
的 Suppor AWS IoT Device Tester t 更新 AWS IoT Greengrass	AWS IoT Greengrass V2 版本 4.0.1 的 IDT 现在支持使用 Greengrass nucleus 版本 2.0.4 和 Greengrass CLI 版本 2.0.4 进行设备认证。	2021 年 2 月 5 日
导入 Lambda 函数的新教程	添加了新的基于控制台的教程，用于将 Lambda 函数导入为在 Greengrass 核心设备上运行的组件。	2021 年 2 月 5 日

AWS IoT Greengrass 酷睿 v2.0.4 软件更新	此版本提供了 Greengrass nucleus 组件的 2.0.4 版本。它包括用于通过端口 443 配置 HTTPS 通信的新`greengrassDataPlanePort` 参数，并修复了错误。现在，最低限度 IAM 策略要求`sts:GetCallerIdentity` 使用`iam:GetPolicy` 和运行 AWS IoT Greengrass 核心软件安装程序`--provision true` 。	2021 年 2 月 4 日
发布了新的安全隧道组件	1.0.0 版的安全隧道组件可用于。 AWS IoT Greengrass此AWS提供的组件使用 AWS IoT 安全隧道与位于受限防火墙后面的 Greengrass 核心设备建立安全的双向通信。	2021 年 1 月 21 日
AWS IoT Device Tester 适用于 AWS IoT Greengrass v4.0.1 的版本已发布	适用于 AWS IoT Greengrass V2 的 IDT 4.0.1 版本现已推出。此版本允许您使用 IDT 开发和运行用于设备验证的自定义测试套件。这还包括适用于 macOS 和 Windows 的代码签名的 IDT 应用程序。	2020 年 12 月 22 日
的初始版本 AWS IoT Greengrass Version 2	AWS IoT Greengrass V2 是的新主要版本版本 AWS IoT Greengrass。此版本增加了多项功能，例如模块化软件组件和持续部署。这些功能使您可以更轻松地开发和管理边缘应用程序。	2020 年 12 月 15 日

AWS 词汇表

有关最新 AWS 术语，请参阅《AWS 词汇表 参考资料》中的AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。