开发人员指南，版本 1

AWS IoT Greengrass

AWS IoT Greengrass: 开发人员指南，版本 1

Table of Contents

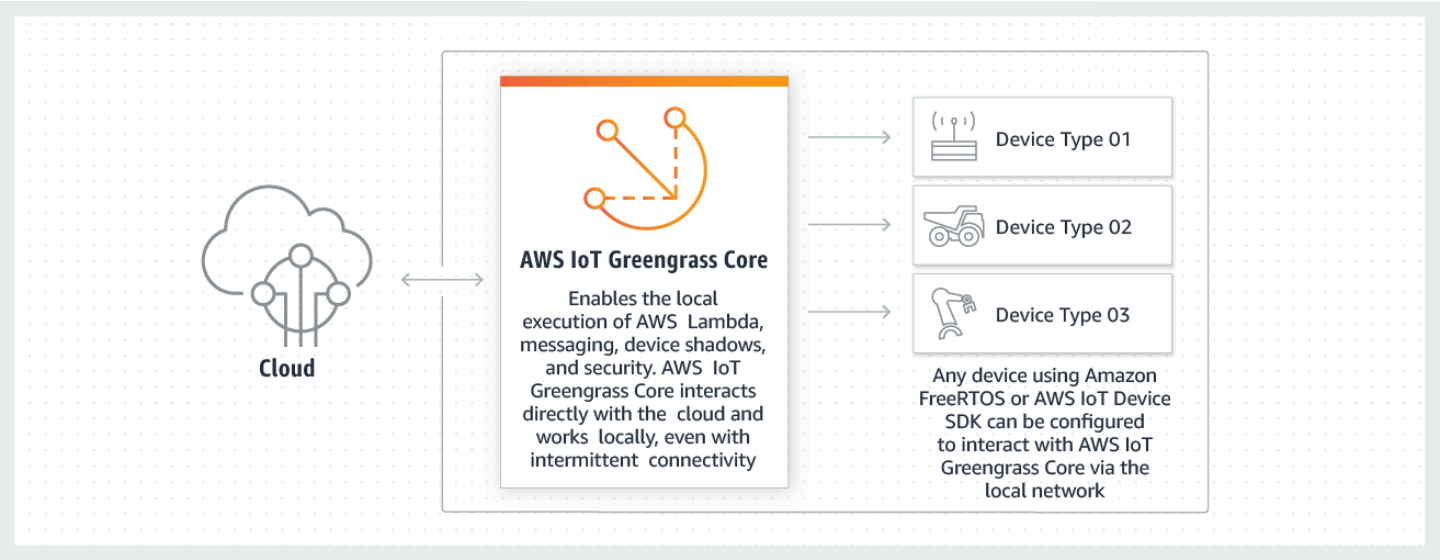
AWS IoT Greengrass Version 1 2023 年 6 月 30 日进入延长寿命阶段。有关更多信息，请参阅 AWS IoT Greengrass V1 维护策略。在此日期之后，将 AWS IoT Greengrass V1 不会发布提供功能、增强功能、错误修复或安全补丁的更新。在上面运行的设备 AWS IoT Greengrass V1 不会中断，将继续运行并连接到云端。我们强烈建议您迁移到 AWS IoT Greengrass Version 2，这样可以添加重要的新功能并支持其他平台。

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。

什么是 AWS IoT Greengrass？

AWS IoT Greengrass 是将云功能扩展到本地设备的软件。这使得设备可以更靠近信息源来收集和分析数据，自主响应本地事件，同时在本地网络上彼此安全地通信。本地设备还可以与 AWS IoT Core 安全通信并将 IoT 数据导出到 AWS Cloud。AWS IoT Greengrass 开发人员可以使用 AWS Lambda 函数和预构建的连接器来创建无服务器应用程序，这些应用程序将部署到设备上以进行本地执行。

下图演示了 AWS IoT Greengrass 的基本架构。



AWS IoT Greengrass 使客户能够构建 IoT 设备和应用程序逻辑。具体来说，AWS IoT Greengrass 对设备上运行的应用程序逻辑提供基于云的管理。在本地部署的 Lambda 函数和连接器通过本地事件以及来自云或其他来源的消息触发。

在 AWS IoT Greengrass 中，设备可在本地网络上安全地通信并互相交换消息而不必连接到云。AWS IoT Greengrass 提供了一个本地发布/订阅消息管理器，该管理器可在丢失连接的情况下智能地缓冲消息，使云的入站和出站消息得到保留。

AWS IoT Greengrass 保护用户数据：

- 通过安全的设备身份验证和授权。
- 通过本地网络中的安全连接。
- 在本地设备与云之间。

设备安全凭证在撤销之前一直在组中有效，即使到云的连接中断，设备可以继续在本地安全地进行通信。

AWS IoT Greengrass提供 Lambda 函数的安全 over-the-air 更新。

AWS IoT Greengrass 包含以下内容：

- 软件分发
  - AWS IoT Greengrass Core 软件
  - AWS IoT Greengrass Core 软件开发工具包
- 云服务
  - AWS IoT Greengrass API
- 功能
  - Lambda 运行时
  - 影子实施
  - 消息管理器
  - 组管理
  - 发现服务
  - O ver-the-air 更新代理
  - 流管理器
  - 本地资源访问
  - 本地机器学习推理
  - 本地 Secrets Manager
  - 内置集成了服务、协议和软件的连接器

主题

AWS IoT Greengrass Core 软件

AWS IoT Greengrass Core 软件提供了以下功能：

- 连接器和 Lambda 函数的部署和本地运行。
- 在本地处理数据流，并自动导出到 AWS Cloud。
- 使用托管订阅通过本地网络在设备、连接器和 Lambda 函数之间进行的 MQTT 消息传递。
- 使用托管订阅在 AWS IoT 与设备、连接器和 Lambda 函数之间进行的 MQTT 消息传递。
- 使用设备身份验证和授权确保设备和 AWS Cloud 之间的安全连接。
- 设备的本地影子同步。影子可配置为与 AWS Cloud 同步。
- 对本地设备和卷资源的受控访问。
- 用于运行本地推理的云训练机器学习模型的部署。
- 使设备能够发现 Greengrass 核心设备的自动 IP 地址检测。
- 全新的或更新的组配置的集中部署。下载配置数据后，核心设备将自动重启。
- 对用户定义的 Lambda 函数进行安全 over-the-air (OTA) 软件更新。
- 本地密钥的安全、加密的存储以及连接器和 Lambda 函数进行的受控访问。

AWS IoT Greengrass 核心实例通过可创建和更新存储在云中的 AWS IoT Greengrass 组定义的 AWS IoT Greengrass API 进行配置。

AWS IoT Greengrass Core 软件版本

AWS IoT Greengrass 提供了多种选项用于安装 AWS IoT Greengrass Core 软件，包括 targ.gz 下载文件、快速启动脚本，以及在支持的 Debian 平台上安装 `apt`。有关更多信息，请参阅 the section called “安装 AWS IoT Greengrass Core 软件”。

以下选项卡介绍了 AWS IoT Greengrass Core 软件版本的新增功能和更改。

GGC v1.11

1.11.6

错误修复和改进：

- 改进了部署期间突然断电时的恢复能力。
- 修复了流管理器数据损坏可能阻止 AWS IoT Greengrass Core 软件启动的问题。

- 修复了在某些情况下新客户端设备无法连接到核心的问题。
- 修复了流管理器流名称无法包含 `.log` 的问题。

1.11.5

错误修复和改进：

- 常规性能改进和错误修复。

1.11.4

错误修复和改进：

- 修复了流管理器的一个问题，该问题造成无法升级到 AWS IoT Greengrass Core 软件 v1.11.3。如果您使用流管理器将数据导出到云，现在可以使用 OTA 更新将较早的 v1.x 版 AWS IoT Greengrass Core 软件升级到 v1.11.4。
- 常规性能改进和错误修复。

1.11.3

错误修复和改进：

- 修复了一个问题，该问题导致在 Ubuntu 设备上 Snap 中运行的 AWS IoT Greengrass Core 软件在设备突然断电后停止响应。
- 修复了一个问题，该问题导致向长时间生存的 Lambda 函数传送 MQTT 消息时出现延迟。
- 修复了一个问题，该问题导致 `maxWorkItemCount` 值设置为大于 `1024` 时无法正确发送 MQTT 消息。
- 修复了一个问题，该问题导致 OTA 更新代理忽略 `config.json` 的 `keepAlive` 属性中指定的 MQTT `KeepAlive` 周期。
- 常规性能改进和错误修复。

> **Important**
>
> 如果您使用流管理器将数据导出到云，请不要从较早的 v1.x 版 AWS IoT Greengrass Core 软件升级到 v1.11.3。如果是首次启用流管理器，我们强烈建议您首先安装最新版本的 AWS IoT Greengrass Core 软件。

1.11.1

错误修复和改进：

- 修复了导致流管理器内存使用量增大的问题。
- 修复了在 Greengrass 核心设备关闭时间超过 time-to-live 指定的 (TTL) 流数据0时段时，直播管理器会将直播的序列号重置为的问题。
- 修复了一个问题，该问题导致流管理器无法正确停止重新尝试将数据导出到 AWS Cloud。

1.11.0

新功能：

- Greengrass 核心上的遥测代理可以收集本地遥测数据并将其发布到 AWS Cloud。要检索遥测数据以供进一步处理，客户可以创建 Amazon EventBridge 规则并订阅目标。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。
- 本地 HTTP API 可以返回由 AWS IoT Greengrass 启动的本地工作线程进程的当前状态快照。有关更多信息，请参阅调用本地运行状况检查 API。
- 流管理器可以自动将数据导出到 Amazon S3 和 AWS IoT SiteWise。

  借助新的流管理器参数，您可以更新现有的流并暂停或恢复数据导出。
- 支持在核心上运行 Python 3.8.x Lambda 函数。
- `config.json` 中包含一个新的 `ggDaemonPort` 属性，可用于配置 Greengrass 核心 IPC 端口号。默认端口号为 8000。

  `config.json` 中包含一个新的 `systemComponentAuthTimeout` 属性，它可用于配置 Greengrass 核心 IPC 身份验证的超时时间。默认超时为 5000 毫秒。
- 每个 AWS IoT Greengrass 组的 AWS IoT 设备数量上限从 200 增加到了 2500。

  每个组的订阅数量上限从 1000 增加到了 10000。

  有关更多信息，请参阅 AWS IoT Greengrass 端点和配额。

错误修复和改进：

- 进行了常规优化，可以降低 Greengrass 服务进程的内存利用率。
- 新的运行时配置参数 (`mountAllBlockDevices`) 允许 Greengrass 在设置 OverlayFS 后使用绑定挂载将所有块设备挂载到容器中。此功能解决了 `/usr` 不在 `/` 层次结构下时导致 Greengrass 部署失败的问题。
- 修复了 `/tmp` 是符号链接时导致 AWS IoT Greengrass 核心失败的问题。
- 修复了一个问题，使 Greengrass 部署代理可从 `mlmodel_public` 文件夹中移除未使用的机器学习模型工件。

- 常规性能改进和错误修复。

Extended life versions

1.10.5

错误修复和改进：

- 常规性能改进和错误修复。

1.10.4

错误修复和改进：

- 修复了一个问题，该问题导致在 Ubuntu 设备上 Snap 中运行的 AWS IoT Greengrass Core 软件在设备突然断电后停止响应。
- 修复了一个问题，该问题导致向长时间生存的 Lambda 函数传送 MQTT 消息时出现延迟。
- 修复了一个问题，该问题导致 `maxWorkItemCount` 值设置为大于 `1024` 时无法正确发送 MQTT 消息。
- 修复了一个问题，该问题导致 OTA 更新代理忽略 `config.json` 的 `keepAlive` 属性中指定的 MQTT `KeepAlive` 周期。
- 常规性能改进和错误修复。

1.10.3

错误修复和改进：

- `config.json` 中包含一个新的 `systemComponentAuthTimeout` 属性，它可用于配置 Greengrass 核心 IPC 身份验证的超时时间。默认超时为 5000 毫秒。
- 修复了导致流管理器内存使用量增大的问题。

1.10.2

错误修复和改进：

- config.json 中包含一个新的 `mqttOperationTimeout` 属性，可用于在与 AWS IoT Core 连接的 MQTT 中设置发布、订阅和取消订阅操作的超时时间。
- 常规性能改进和错误修复。

1.10.1

错误修复和改进：

- 流管理器能够更灵活地应对文件数据损坏。
- 修复了导致使用 Linux 内核 5.1 及更高版本的设备上系统安装失败的问题。
- 常规性能改进和错误修复。

1.10.0

新功能：

- 用于本地处理数据流并自动将其导出到 AWS Cloud 的流管理器。此功能需要 Greengrass 核心设备上的 Java 8。有关更多信息，请参阅 管理数据流。
- 用于在核心设备上运行 Docker 应用程序的新 Greengrass Docker 应用程序部署连接器。有关更多信息，请参阅 the section called “Docker 应用程序部署”。
- 一种新的物联网 SiteWise 连接器，可将工业设备数据从OPC-UA服务器发送到中的资产属性。AWS IoT SiteWise有关更多信息，请参阅 the section called “物联网 SiteWise”。
- 在未进行容器化的情况下运行的 Lambda 函数可以访问 Greengrass 组中的机器学习资源。有关更多信息，请参阅 the section called “访问机器学习资源”。
- 支持与 AWS IoT 的 MQTT 持久性会话。有关更多信息，请参阅 the section called “与 AWS IoT Core 的 MQTT 持久性会话”。
- 本地 MQTT 流量可通过默认端口 8883 以外的端口传输。有关更多信息，请参阅 the section called “用于本地消息收发的 MQTT 端口”。
- AWS IoT Greengrass Core 软件开发工具包中包含新的 `queueFullPolicy` 选项，可用于通过 Lambda 函数可靠地发布消息。
- 支持在核心上运行 Node.js 12.x Lambda 函数。
- OpenSSL 1.1 可以配置集成了硬件安全的 O ver-the-air (OTA) 更新。
- 常规性能改进和错误修复。

1.9.4

错误修复和改进：

- 常规性能改进和错误修复。

1.9.3

新功能：

- 支持 Armv6l。AWS IoT GreengrassCore 软件 v1.9.3 或更高版本可在 Armv6l 架构上的 Raspbian 发行版中安装（例如，在 Raspberry Pi Zero 设备上）。

- OTA 在端口 443 上使用 ALPN 进行更新。使用端口 443 进行 MQTT 流量的 Greengrass 内核现在支持 (OTA) 软件更新。 over-the-air AWS IoT Greengrass使用应用层协议网络 (ALPN) TLS 扩展来启用这些连接。有关更多信息，请参阅AWS IoT Greengrass Core 软件的 OTA 更新和the section called “通过端口 443 或网络代理进行连接”。

错误修复和改进：

- 修复了 v1.9.0 中引入的错误，该错误导致 Python 2.7 Lambda 函数无法将二进制负载发送到其他 Lambda 函数。
- 常规性能改进和错误修复。

1.9.2

新功能：

- Support fo OpenWrtr. AWS IoT Greengrass 核心软件 v1.9.2 或更高版本可以安装在采用 Armv8 (aarch64) 和 armv7L 架构的 OpenWrt 发行版上。目前， OpenWrt 不支持 ML 推理。

1.9.1

错误修复和改进：

- 修复了 1.9.0 版本中从 `cloud` 删除主题含有通配符的消息时引入的错误。

1.9.0

新功能：

- 支持 Python 3.7 和 Node.js 8.10 Lambda 运行时。使用 Python 3.7 和 Node.js 8.10 运行时的 Lambda 函数现在可以在 AWS IoT Greengrass 核心上运行。（ AWS IoT Greengrass 仍然支持 Python 2.7 和 Node.js 6.10 运行时。 ）
- 优化了 MQTT 连接。Greengrass 核心与 AWS IoT Core 建立的连接数较少。对于基于连接数的收费，此更改可以降低运营成本。
- 适用于本地 MQTT 服务器的椭圆曲线 (EC) 密钥。除了 RSA 密钥之外，本地 MQTT 服务器还支持 EC 密钥。（ 无论密钥类型如何，MQTT 服务器证书都具有 SHA-256 RSA 签名。 ）有关更多信息，请参阅 the section called “安全委托人”。

错误修复和改进：

- 常规性能改进和错误修复。

1.8.4

修复了影子同步和设备证书管理器重新连接的问题。

常规性能改进和错误修复。

1.8.3

常规性能改进和错误修复。

1.8.2

常规性能改进和错误修复。

1.8.1

常规性能改进和错误修复。

1.8.0

新功能：

- 组中 Lambda 函数的可配置默认访问身份。这一组级别设置确定用于运行 Lambda 函数的默认权限。您可以设置用户 ID 和/或组 ID。各个 Lambda 函数可以覆盖其组的默认访问身份。有关更多信息，请参阅 the section called “为组中的 Lambda 函数设置默认访问身份”。
- 通过端口 443 的 HTTPS 流量。HTTPS 通信可以配置为通过端口 443（而不是默认端口 8443）进行传输。这补充了 AWS IoT Greengrass 对应用层协议网络 (ALPN) TLS 扩展的支持，并允许所有 Greengrass 消息收发流量（包括 MQTT 和 HTTPS）使用端口 443。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”。
- AWS IoT 连接可预测命名的客户端 ID。此更改将启用对于 AWS IoT Device Defender 和 AWS IoT 生命周期事件的支持，因此，您可以针对连接、断开连接、订阅以及取消订阅事件接收通知。借助于可预测的命名，还可以更轻松地围绕连接 ID 创建逻辑（例如，基于证书属性创建订阅策略模板）。有关更多信息，请参阅 the section called “与 AWS IoT 的 MQTT 连接的客户端 ID”。

错误修复和改进：

- 修复了影子同步和设备证书管理器重新连接的问题。
- 常规性能改进和错误修复。

1.7.1

新功能：

- Greengrass 连接器提供与本地基础设施、设备协议、AWS 以及其他云服务的内置集成。有关更多信息，请参阅 使用连接器与服务和协议集成。
- AWS IoT Greengrass 将 AWS Secrets Manager 扩展到核心设备，从而使密码、令牌和其他密钥可用于连接器和 Lambda 函数。传输和静态中的密钥均经过加密。有关更多信息，请参阅 将密钥部署到核心。

- 支持可信硬件根安全选项。有关更多信息，请参阅 the section called “硬件安全性集成”。
- 隔离和权限设置，允许 Lambda 函数在无 Greengrass 容器的情况下运行，并允许使用指定用户和组的权限。有关更多信息，请参阅 the section called “控制 Greengrass Lambda 函数执行”。
- 您可以通过将 Greengrass 组配置为在不进行容器化的情况下运行，来在 Docker 容器（在 Windows、macOS 或 Linux 上）中运行 AWS IoT Greengrass。有关更多信息，请参阅 the section called “在 Docker 容器中运行 AWS IoT Greengrass”。
- 端口 443 上使用应用程序层协议协商 (ALPN) 或网络代理连接的 MQTT 消息传递。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”。
- SageMaker Neo 深度学习运行时，它支持由 SageMaker Neo 深度学习编译器优化的机器学习模型。有关 Neo 深度学习运行时的信息，请参阅the section called “用于 ML 推理的运行时和库”。
- 支持在 Raspberry Pi 核心设备上使用 Raspbian Stretch (2018-06-27)。

错误修复和改进：

- 常规性能改进和错误修复。

此外，此版本还提供以下功能：

- AWS IoT Device Tester for AWS IoT Greengrass 可用来验证 CPU 架构、内核配置和 AWS IoT Greengrass 使用的驱动器。有关更多信息，请参阅 使用适用于 AWS IoT Greengrass V1 的 AWS IoT 设备测试器。
- AWS IoT Greengrass核心软件、Cor AWS IoT Greengrass e SDK 和 Mach AWS IoT Greengrass ine Learning SDK 包可通过亚马逊下载 CloudFront。有关更多信息，请参阅 the section called “AWS IoT Greengrass 下载”。

1.6.1

新功能：

- 在 Greengrass 核心上运行二进制代码的 Lambda 可执行文件。使用适用于 C 的全新 AWS IoT Greengrass Core 软件开发工具包以 C 和 C++ 编写 Lambda 可执行文件。有关更多信息，请参阅 the section called “Lambda 可执行文件”。
- 可选的本地存储消息缓存，可以在重新启动时永久保存这些消息。您可以为排队处理的 MQTT 消息配置存储设置。有关更多信息，请参阅 the section called “MQTT 消息队列”。
- 在核心设备断开时的可配置最大重新连接重试间隔。有关更多信息，请参阅the section called “AWS IoT Greengrass 核心配置文件”中的 `mqttMaxConnectionRetryInterval` 属性。
- 主机上 /proc 目录的本地资源访问权限。有关更多信息，请参阅 访问本地资源。

- 可配置的写入目录。可以将 AWS IoT Greengrass Core 软件部署到只读和读写位置中。有关更多信息，请参阅 the section called “配置写入目录”。

错误修复和改进：

- 提高了在 Greengrass 核心中以及设备和核心之间发布消息的性能。
- 减少了处理用户定义 Lambda 函数生成的日志所需的计算资源。

1.5.0

新功能：

- AWS IoT Greengrass 机器学习 (ML) 推理已正式发布。您可以在 AWS IoT Greengrass 设备上使用在云中构建和训练的模型执行本地 ML 推理。有关更多信息，请参阅 执行机器学习推理。
- 除了 JSON 以外，Greengrass Lambda 函数现在还支持将二进制数据作为输入负载。要使用该功能，您必须升级到 AWS IoT Greengrass Core 软件开发工具包版本 1.1.0，这可以从 AWS IoT Greengrass Core 软件开发工具包下载页面进行下载。

错误修复和改进：

- 降低了整体内存占用空间。
- 将消息发送到云的性能改进。
- 对下载代理、Device Certificate Manager 和 OTA 更新代理的性能和稳定性改进。
- 次要错误修复。

1.3.0

新功能：

- O ver-the-air (OTA) 更新代理能够处理云端部署的 Greengrass 更新任务。该代理是在新 `/greengrass/ota` 目录下找到的。有关更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新。
- 本地资源访问功能允许 Greengrass Lambda 函数访问本地资源，如外围设备和卷。有关更多信息，请参阅 使用 Lambda 函数和连接器访问本地资源。

1.1.0

新功能：

- 可以删除 Lambda 函数、订阅和配置以重置部署的 AWS IoT Greengrass 组。有关更多信息，请参阅 the section called “重置部署”。

- 除 Python 2.7 以外，还支持 Node.js 6.10 和 Java 8 Lambda 运行时。

从以前版本的 AWS IoT Greengrass Core 进行迁移：

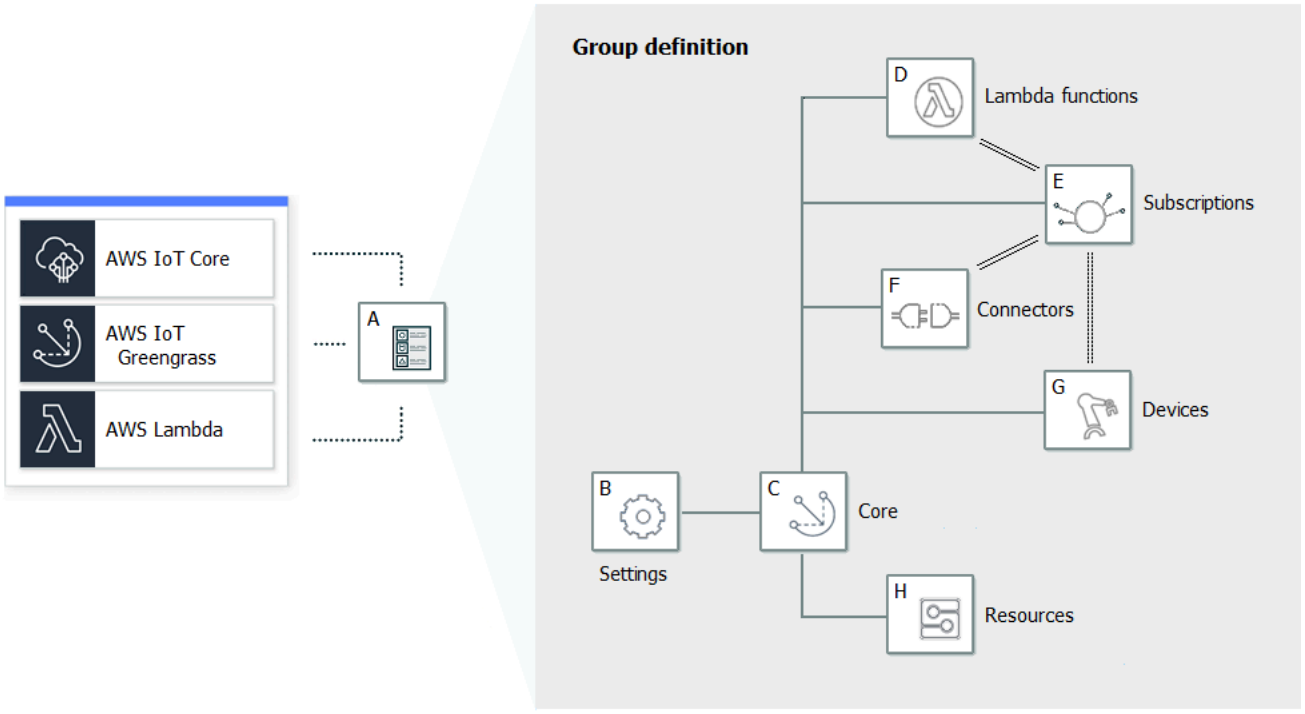
- 将证书从 `/greengrass/configuration/certs` 文件夹复制到 `/greengrass/certs`。
- 将 `/greengrass/configuration/config.json` 复制到 `/greengrass/config/config.json`。
- 运行 `/greengrass/ggc/core/greengrassd` 而非 `/greengrass/greengrassd`。
- 将组部署到新的核心。

1.0.0

初始版本

AWS IoT Greengrass 组

Greengrass 组是一系列设置和组件，例如 Greengrass 核心、设备和订阅。组用于定义交互范围。例如，一个组可能表示建筑物的一层、一辆卡车或整个采矿场所。下图显示了可以构成 Greengrass 组的组件。



在上图中：

A：Greengrass 组定义

有关组设置和组件的信息。

B：Greengrass 组设置

其中包括：

- Greengrass 组角色。
- 证书颁发机构和本地连接配置
- Greengrass 核心连接信息。
- 默认 Lambda 运行时环境。有关更多信息，请参阅 the section called “在组中设置 Lambda 函数的默认容器化”。
- CloudWatch 和本地日志配置。有关更多信息，请参阅 the section called “利用 AWS IoT Greengrass 日志进行监控”。

C：Greengrass 核心

表示 Greengrass 核心的 AWS IoT 事物（设备）。有关更多信息，请参阅 the section called “配置 AWS IoT Greengrass 核心”。

D：Lambda 函数定义

核心上本地运行的 Lambda 函数的列表，包含关联的配置数据。有关更多信息，请参阅 运行本地 Lambda 函数。

E：订阅定义

允许使用 MQTT 消息进行通信的订阅列表。订阅定义了：

- 消息源和消息目标。这些可能是客户端设备、Lambda 函数、连接器、AWS IoT Core 和本地影子服务。
- 用于筛选消息的主题。

有关更多信息，请参阅 the section called “MQTT 消息传递工作流中的托管订阅”。

F：连接器定义

核心上本地运行的连接器的列表，包含关联的配置数据。有关更多信息，请参阅 使用连接器与服务和协议集成。

G：设备定义

作为 Greengrass 组成员的 AWS IoT 事物（称为客户端设备或设备）以及关联的配置数据的列表。有关更多信息，请参阅 the section called “AWS IoT Greengrass 中的设备”。

H：资源定义

Greengrass 核心上的本地资源、机器学习资源和密钥资源的列表，包含关联的配置数据。有关更多信息，请参阅 访问本地资源、执行机器学习推理 和 将密钥部署到核心。

在部署后，Greengrass 组定义、Lambda 函数、连接器、资源和订阅表将复制到核心设备中。有关更多信息，请参阅 部署 AWS IoT Greengrass 组。

AWS IoT Greengrass 中的设备

Greengrass 组可以包含两种类型的 AWS IoT 设备：

Greengrass 核心

Greengrass 核心是运行 AWS IoT Greengrass Core 软件的设备，该软件允许核心直接与 AWS IoT Core 和 AWS IoT Greengrass 服务进行通信。核心具有自己的设备证书，用于在 AWS IoT Core 中进行身份验证。它有一个设备影子和一个 AWS IoT Core 注册表中的条目。Greengrass 核心运行本地 Lambda 运行时、部署代理和 IP 地址跟踪器（用于将 IP 地址信息发送到 AWS IoT Greengrass 服务以允许客户端设备自动发现其组和核心连接信息）。有关更多信息，请参阅 the section called “配置 AWS IoT Greengrass 核心”。

> **Note**
>
> 一个 Greengrass 组必须恰好包含一个核心。

客户端设备

客户端设备（也称为联网设备、Greengrass 设备或设备）是通过 MQTT 连接到 Greengrass 核心的设备。这些设备有其自己的设备证书（用于进行 AWS IoT Core身份验证）、设备影子以及在 AWS IoT Core 注册表中的项。客户端进设备可以运行 FreeRTOS 或使用 AWS IoT 设备开发工具包或 AWS IoT Greengrass 发现 API，以获取用于连接和验证同一 Greengrass 组中核心的发现信息。要了解如何使用 AWS IoT 控制台创建和配置 AWS IoT Greengrass 的设备，请参阅the section called “模块 4：在 AWS IoT Greengrass 组中与客户端设备交互”。或者，有关说明如何使用为其创建和配置客户端设备的示例AWS IoT Greengrass，请参阅《AWS CLI命令参考》create-device-definition中的。AWS CLI

在 Greengrass 组中，您可以创建订阅来允许客户端设备通过 MQTT 与组中的 Lambda 函数、连接器、其他设备以及与 AWS IoT Core 或和本地影子服务进行通信。MQTT 消息传递到核心。如果核

心设备丢失与云的连接，客户端设备可通过本地网络继续进行通信。客户端设备大小不一，既有基于微控制器的小型设备，也有大型设备。目前，一个 Greengrass 组最多可包含 2,500 个客户端设备。一个客户端设备最多可以是 10 个组的成员。

> **Note**
>
> OPC-UA 是一种用于工业通信的信息交换标准。要在 Greengrass 内核上实现对 OPC-UA 的支持，你可以使用物联网连接器。 SiteWise 该连接器可将工业设备数据从 OPC-UA 服务器发送到 AWS IoT SiteWise 中的资产属性。

下表显示了这些设备类型的相关性。

	Core	Device
Certificate	✓	✓
IoT Policy	✓	✓
IoT Thing	✓	✓
Device use	Gateway	Sensor and/or Actuator
Software	AWS IoT Greengrass Core Software	Amazon FreeRTOS / AWS IoT Device SDK
Group membership	✓	✓
Functions outside a Greengrass Group	✗	✓

AWS IoT Greengrass 核心设备在两个位置存储证书：

- */greengrass-root*/certs 中的核心设备证书。通常，核心设备证书命名为 *hash*.cert.pem（例如，86c84488a5.cert.pem）。当核心连接到 AWS IoT Core 和 AWS IoT Greengrass 服务时，AWS IoT 客户端将使用此证书进行相互身份验证。
- */greengrass-root*/ggc/var/state/server 中的 MQTT 服务器证书。MQTT 服务器证书名为 server.crt。此证书用于本地 MQTT 服务器（在 Greengrass 核心上）和 Greengrass 设备之间的相互身份验证。

> **Note**
>
> *greengrass-root* 表示在您的设备上安装 AWS IoT Greengrass Core 软件的路径。通常，这是 /greengrass 目录。

软件开发工具包

AWS 提供的以下软件开发工具包可与 AWS IoT Greengrass 搭配使用：

AWS 软件开发工具包

使用 AWS 软件开发工具包可构建与任何 AWS 服务（包括 Amazon S3、Amazon DynamoDB、AWS IoT 和 AWS IoT Greengrass 等）交互的应用程序。在 AWS IoT Greengrass 的上下文中，您可以使用已部署的 Lambda 函数中的 AWS 软件开发工具包对任何 AWS 服务进行直接调用。有关更多信息，请参阅 AWS SDK。

> **Note**
>
> AWS 软件开发工具包中提供的特定于 Greengrass 的操作也可以在 AWS IoT Greengrass API 和 AWS CLI 中使用。

AWS IoT Device 软件开发工具包

AWS IoT Device 软件开发工具包可帮助设备连接到 AWS IoT Core 或 AWS IoT Greengrass。有关更多信息，请参阅AWS IoT 开发人员指南中的 AWS IoT Device 软件开发工具包。

客户端设备可以使用任何 AWS IoT Device 软件开发工具包 v2 平台来发现 Greengrass 核心的连接信息。连接信息包括：

- 客户端设备所属的 Greengrass 组的 ID。

- 每个组中 Greengrass 核心的 IP 地址。这些也称为核心端点。
- 组 CA 证书，设备使用该证书与核心进行双向身份验证。有关更多信息，请参阅 the section called “设备连接工作流程”。

> **Note**
>
> 在 AWS IoT Device 软件开发工具包的 v1 中，只有 C++ 和 Python 平台提供内置的发现支持。

AWS IoT Greengrass Core 软件开发工具包

AWS IoT Greengrass Core 软件开发工具包允许 Lambda 函数与 Greengrass 核心交互，将消息发布到 AWS IoT，与本地影子服务交互，调用其他部署的 Lambda 函数，以及访问密钥资源。此软件开发工具包由 AWS IoT Greengrass 核心上运行的 Lambda 函数使用。有关更多信息，请参阅 AWS IoT Greengrass 核心开发工具包。

AWS IoT Greengrass 机器学习软件开发工具包

AWS IoT Greengrass 机器学习软件开发工具包允许 Lambda 函数 函数使用作为机器学习资源部署到 Greengrass 核心的机器学习模型。此软件开发工具包由 AWS IoT Greengrass 核心上运行的与本地推理服务交互的 Lambda 函数使用。有关更多信息，请参阅 AWS IoT Greengrass 机器学习开发工具包。

支持的平台和要求

以下选项卡列出了支持的平台和 AWS IoT Greengrass Core 软件的要求。

> **Note**
>
> 您可以从 AWS IoT Greengrass Core 软件下载页面下载 AWS IoT Greengrass Core 软件。

GGC v1.11

支持的平台:

- 架构：Armv7l
  - 操作系统：Linux

    - 操作系统：Linux (OpenWrt)
- 架构：Armv8 (AArch64)
    - 操作系统：Linux
    - 操作系统：Linux (OpenWrt)
- 架构：Armv6l
    - 操作系统：Linux
- 架构：x86_64
    - 操作系统：Linux
- Windows、macOS 和 Linux 平台可以在 Docker 容器中运行 AWS IoT Greengrass。有关更多信息，请参阅 the section called “在 Docker 容器中运行 AWS IoT Greengrass”。

要求：

- 可用于 AWS IoT Greengrass Core 软件的磁盘空间不少于 128 MB。如果您要使用 OTA 更新代理，则最小值为 400 MB。
- 分配给 AWS IoT Greengrass Core 软件的最少 128 MB 内存。启用流管理器后，最小 RAM 为 198 MB。

  > **Note**
  >
  > 如果您使用AWS IoT 控制台上的默认组创建选项来创建 Greengrass 组，则默认情况下会启用流管理器。

- Linux 内核版本：
    - 需要 Linux 内核版本 4.4 或更高版本才能支持 AWS IoT Greengrass 与容器一起运行。
    - 需要 Linux 内核版本 3.17 或更高版本才能支持不带容器运行 AWS IoT Greengrass。在此配置中，Greengrass 组的默认 Lambda 函数容器化必须设置为无容器。有关说明，请参阅the section called “在组中设置 Lambda 函数的默认容器化”。
- GNU C 库 (glibc) 版本 2.14 或更高版本。 OpenWrt 发行版需要 musl C 库版本 1.1.16 或更高版本。
- 设备上必须存在 `/var/run` 目录。
- `/dev/stdin`、`/dev/stdout` 和 `/dev/stderr` 文件必须是可用的。
- 必须在设备上启用硬链接和软链接保护。否则，AWS IoT Greengrass 只能使用 `-i` 标记在不安全的模式下运行。

- 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

    内核必须支持 cgroups。使用容器运行 AWS IoT Greengrass 需要满足以下要求：
    - 必须已启用并挂载 memory cgroup 以允许 AWS IoT Greengrass 设置 Lambda 函数的内存限制。
    - 如果使用具有本地资源访问权限的 Lambda 函数打开 AWS IoT Greengrass 核心设备上的文件，必须启用并安装 devices cgroup。
  - 其他：
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- 系统信任存储中必须存在 Amazon S3 和 AWS IoT 的根证书。
- 除了基本的 AWS IoT Greengrass Core 软件内存要求外，流管理器还需要 Java 8 运行时和至少 70 MB 的 RAM。如果您使用AWS IoT 控制台上的默认组创建选项，则默认情况下会启用流管理器。 OpenWrt 发行版不支持直播管理器。
- 为您想要在本地运行的 Lambda 函数所需的 AWS Lambda 运行时提供支持的库。所需的库必须安装在核心上并添加到 PATH 环境变量中。在同一核心上可以安装多个库。
  - 适用于使用 Python 3.8 运行时的函数的 Python 版本 3.8。
  - 适用于使用 Python 3.7 运行时的函数的 Python 版本 3.7。

- 适用于使用 Python 2.7 运行时的函数的 Python 版本 2.7。
- 适用于使用 Node.js 12.x 运行时的函数的 Node.js 版本 12.x。
- 适用于使用 Java 8 运行时的函数的 Java 版本 8 或更高版本。

> **Note**
>
> 官方不支持在 OpenWrt 发行版上运行 Java。但是，如果您的 OpenWrt 版本支持 Java，则可以在您的设备上运行用 Java 编写的 Lambda 函数。 OpenWrt

有关 AWS IoT Greengrass 对 Lambda 运行时的支持的更多信息，请参阅运行本地 Lambda 函数。

- (OTA) 更新代理需要以下 shell 命令over-the-air （ 不是 BusyBox 变体 ）：
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`
  - `umount`
  - `mv`
  - `gzip`
  - `mkdir`
  - `rm`
  - `ln`
  - `cut`
  - `cat`
  - `/bin/bash`

GGC v1.10

支持的平台:

- 架构：Armv7l
  - 操作系统：Linux
  - 操作系统：Linux (OpenWrt)
- 架构：Armv8 (AArch64)
  - 操作系统：Linux
  - 操作系统：Linux (OpenWrt)
- 架构：Armv6l
  - 操作系统：Linux
- 架构：x86_64
  - 操作系统：Linux
- Windows、macOS 和 Linux 平台可以在 Docker 容器中运行 AWS IoT Greengrass。有关更多信息，请参阅 the section called “在 Docker 容器中运行 AWS IoT Greengrass”。

要求：

- 可用于 AWS IoT Greengrass Core 软件的磁盘空间不少于 128 MB。如果您要使用 OTA 更新代理，则最小值为 400 MB。
- 分配给 AWS IoT Greengrass Core 软件的最少 128 MB 内存。启用流管理器后，最小 RAM 为 198 MB。

  > **Note**
  >
  > 如果您使用AWS IoT 控制台上的默认组创建选项来创建 Greengrass 组，则默认情况下会启用流管理器。

- Linux 内核版本：
  - 需要 Linux 内核版本 4.4 或更高版本才能支持 AWS IoT Greengrass 与容器一起运行。
  - 需要 Linux 内核版本 3.17 或更高版本才能支持不带容器运行 AWS IoT Greengrass。在此配置中，Greengrass 组的默认 Lambda 函数容器化必须设置为无容器。有关说明，请参阅the section called “在组中设置 Lambda 函数的默认容器化”。

- GNU C 库 (glibc) 版本 2.14 或更高版本。 OpenWrt 发行版需要 musl C 库版本 1.1.16 或更高版本。
- 设备上必须存在 `/var/run` 目录。
- `/dev/stdin`、`/dev/stdout` 和 `/dev/stderr` 文件必须是可用的。
- 必须在设备上启用硬链接和软链接保护。否则，AWS IoT Greengrass 只能使用 `-i` 标记在不安全的模式下运行。
- 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

    内核必须支持 cgroups。使用容器运行 AWS IoT Greengrass 需要满足以下要求：
    - 必须已启用并挂载 memory cgroup 以允许 AWS IoT Greengrass 设置 Lambda 函数的内存限制。
    - 如果使用具有本地资源访问权限的 Lambda 函数打开 AWS IoT Greengrass 核心设备上的文件，必须启用并安装 devices cgroup。
  - 其他：
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- 系统信任存储中必须存在 Amazon S3 和 AWS IoT 的根证书。

- 除了基本的 AWS IoT Greengrass Core 软件内存要求外，流管理器还需要 Java 8 运行时和至少 70 MB 的 RAM。如果您使用AWS IoT 控制台上的默认组创建选项，则默认情况下会启用流管理器。 OpenWrt 发行版不支持直播管理器。
- 为您想要在本地运行的 Lambda 函数所需的 AWS Lambda 运行时提供支持的库。所需的库必须安装在核心上并添加到 PATH 环境变量中。在同一核心上可以安装多个库。
  - 适用于使用 Python 3.7 运行时的函数的 Python 版本 3.7。
  - 适用于使用 Python 2.7 运行时的函数的 Python 版本 2.7。
  - 适用于使用 Node.js 12.x 运行时的函数的 Node.js 版本 12.x。
  - 适用于使用 Java 8 运行时的函数的 Java 版本 8 或更高版本。

    > **Note**
    >
    > 官方不支持在 OpenWrt 发行版上运行 Java。但是，如果您的 OpenWrt 版本支持 Java，则可以在您的设备上运行用 Java 编写的 Lambda 函数。 OpenWrt

    有关 AWS IoT Greengrass 对 Lambda 运行时的支持的更多信息，请参阅运行本地 Lambda 函数。
- (OTA) 更新代理需要以下 shell 命令over-the-air （ 不是 BusyBox 变体 ）：
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`
  - `umount`
  - `mv`
  - `gzip`
  - `mkdir`

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

GGC v1.9

支持的平台:

- 架构：Armv7l
  - 操作系统：Linux
  - 操作系统：Linux (OpenWrt)
- 架构：Armv8 (AArch64)
  - 操作系统：Linux
  - 操作系统：Linux (OpenWrt)
- 架构：Armv6l
  - 操作系统：Linux
- 架构：x86_64
  - 操作系统：Linux
- Windows、macOS 和 Linux 平台可以在 Docker 容器中运行 AWS IoT Greengrass。有关更多信息，请参阅 the section called “在 Docker 容器中运行 AWS IoT Greengrass”。

要求：

- 可用于 AWS IoT Greengrass Core 软件的磁盘空间不少于 128 MB。如果您要使用 OTA 更新代理，则最小值为 400 MB。
- 分配给 AWS IoT Greengrass Core 软件的最少 128 MB 内存。
- Linux 内核版本：
  - 需要 Linux 内核版本 4.4 或更高版本才能支持 AWS IoT Greengrass 与容器一起运行。
  - 需要 Linux 内核版本 3.17 或更高版本才能支持不带容器运行 AWS IoT Greengrass。在此配置中，Greengrass 组的默认 Lambda 函数容器化必须设置为无容器。有关说明，请参阅the section called “在组中设置 Lambda 函数的默认容器化”。

- GNU C 库 (glibc) 版本 2.14 或更高版本。 OpenWrt 发行版需要 musl C 库版本 1.1.16 或更高版本。
- 设备上必须存在 `/var/run` 目录。
- `/dev/stdin`、`/dev/stdout` 和 `/dev/stderr` 文件必须是可用的。
- 必须在设备上启用硬链接和软链接保护。否则，AWS IoT Greengrass 只能使用 `-i` 标记在不安全的模式下运行。
- 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

    内核必须支持 cgroups。使用容器运行 AWS IoT Greengrass 需要满足以下要求：
    - 必须已启用并挂载 memory cgroup 以允许 AWS IoT Greengrass 设置 Lambda 函数的内存限制。
    - 如果使用具有本地资源访问权限的 Lambda 函数打开 AWS IoT Greengrass 核心设备上的文件，必须启用并安装 devices cgroup。
  - 其他：
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- 系统信任存储中必须存在 Amazon S3 和 AWS IoT 的根证书。

- 为您想要在本地运行的 Lambda 函数所需的 AWS Lambda 运行时提供支持的库。所需的库必须安装在核心上并添加到 PATH 环境变量中。在同一核心上可以安装多个库。
  - 适用于使用 Python 2.7 运行时的函数的 Python 版本 2.7。
  - 适用于使用 Python 3.7 运行时的函数的 Python 版本 3.7。
  - 适用于使用 Node.js 6.10 运行时的函数的 Node.js 版本 6.10 或更高版本。
  - 适用于使用 Node.js 8.10 运行时的函数的 Node.js 版本 8.10 或更高版本。
  - 适用于使用 Java 8 运行时的函数的 Java 版本 8 或更高版本。

    > **Note**
    >
    > 官方不支持在 OpenWrt 发行版上运行 Java。但是，如果您的 OpenWrt 版本支持 Java，则可以在您的设备上运行用 Java 编写的 Lambda 函数。 OpenWrt

    有关 AWS IoT Greengrass 对 Lambda 运行时的支持的更多信息，请参阅运行本地 Lambda 函数。
- (OTA) 更新代理需要以下 shell 命令over-the-air （ 不是 BusyBox 变体 ）：
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`
  - `umount`
  - `mv`
  - `gzip`
  - `mkdir`
  - `rm`
  - `ln`

- `cut`
- `cat`

GGC v1.8

- 支持的平台:
  - 架构：Armv7l；操作系统：Linux
  - 架构：x86_64；操作系统：Linux
  - 架构：Armv8 (AArch64)；操作系统：Linux
  - Windows、macOS 和 Linux 平台可以在 Docker 容器中运行 AWS IoT Greengrass。有关更多信息，请参阅 the section called “在 Docker 容器中运行 AWS IoT Greengrass”。
  - Linux 平台可以使用 Greengrass 快照运行功能有限的 AWS IoT Greengrass 版本，该版本可通过 Snapcraft 获得。有关更多信息，请参阅 the section called “AWS IoT Greengrass Snap 软件”。
- 需要以下项目：
  - 可用于 AWS IoT Greengrass Core 软件的磁盘空间不少于 128 MB。如果您要使用 OTA 更新代理，则最小值为 400 MB。
  - 分配给 AWS IoT Greengrass Core 软件的最少 128 MB 内存。
  - Linux 内核版本：
    - 需要 Linux 内核版本 4.4 或更高版本才能支持 AWS IoT Greengrass 与容器一起运行。
    - 需要 Linux 内核版本 3.17 或更高版本才能支持不带容器运行 AWS IoT Greengrass。在此配置中，Greengrass 组的默认 Lambda 函数容器化必须设置为无容器。有关说明，请参阅the section called “在组中设置 Lambda 函数的默认容器化”。
  - GNU C 库 (glibc) 版本 2.14 或更高版本。
  - 设备上必须存在 `/var/run` 目录。
  - `/dev/stdin`、`/dev/stdout` 和 `/dev/stderr` 文件必须是可用的。
  - 必须在设备上启用硬链接和软链接保护。否则，AWS IoT Greengrass 只能使用 `-i` 标记在不安全的模式下运行。
  - 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

    内核必须支持 cgroups。使用容器运行 AWS IoT Greengrass 需要满足以下要求：
    - 必须已启用并挂载 memory cgroup 以允许 AWS IoT Greengrass 设置 Lambda 函数的内存限制。
    - 如果使用具有本地资源访问权限的 Lambda 函数打开 AWS IoT Greengrass 核心设备上的文件，必须启用并安装 devices cgroup。
  - 其他：
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- 系统信任存储中必须存在 Amazon S3 和 AWS IoT 的根证书。
- 有条件地需要以下项目：
  - 为您想要在本地运行的 Lambda 函数所需的 AWS Lambda 运行时提供支持的库。所需的库必须安装在核心上并添加到 PATH 环境变量中。在同一核心上可以安装多个库。
    - 适用于使用 Python 2.7 运行时的函数的 Python 版本 2.7。
    - 适用于使用 Node.js 6.10 运行时的函数的 Node.js 版本 6.10 或更高版本。
    - 适用于使用 Java 8 运行时的函数的 Java 版本 8 或更高版本。
  - (OTA) 更新代理需要以下 shell 命令over-the-air（不是 BusyBox 变体）：
    - `wget`
    - `realpath`
    - `tar`

- `readlink`
- `basename`
- `dirname`
- `pidof`
- `df`
- `grep`
- `umount`
- `mv`
- `gzip`
- `mkdir`
- `rm`
- `ln`
- `cut`
- `cat`

有关 AWS IoT Greengrass 配额（限值）的信息，请参阅Amazon Web Services 一般参考中的服务限额。

有关定价信息，请参阅 AWS IoT Greengrass 定价和 AWS IoT Core 定价。

AWS IoT Greengrass 下载

您可以使用以下信息来查找并下载可与 AWS IoT Greengrass 结合使用的软件。

主题

- AWS IoT Greengrass Core 软件
- AWS IoT Greengrass Snap 软件
- AWS IoT Greengrass Docker 软件
- AWS IoT Greengrass Core 软件开发工具包
- 支持的机器学习运行时和库
- AWS IoT Greengrass ML 软件开发工具包软件

AWS IoT Greengrass Core 软件

AWS IoT Greengrass Core 软件将 AWS 功能扩展到 AWS IoT Greengrass 核心设备，使本地设备可以在本地操作它们生成的数据。

v1.11

1.11.6

错误修复和改进：

- 改进了部署期间突然断电时的恢复能力。
- 修复了流管理器数据损坏可能阻止 AWS IoT Greengrass Core 软件启动的问题。
- 修复了在某些情况下新客户端设备无法连接到核心的问题。
- 修复了流管理器流名称无法包含 `.log` 的问题。

1.11.5

错误修复和改进：

- 常规性能改进和错误修复。

1.11.4

错误修复和改进：

- 修复了流管理器的一个问题，该问题造成无法升级到 AWS IoT Greengrass Core 软件 v1.11.3。如果您使用流管理器将数据导出到云，现在可以使用 OTA 更新将较早的 v1.x 版 AWS IoT Greengrass Core 软件升级到 v1.11.4。
- 常规性能改进和错误修复。

1.11.3

错误修复和改进：

- 修复了一个问题，该问题导致在 Ubuntu 设备上 Snap 中运行的 AWS IoT Greengrass Core 软件在设备突然断电后停止响应。
- 修复了一个问题，该问题导致向长时间生存的 Lambda 函数传送 MQTT 消息时出现延迟。
- 修复了一个问题，该问题导致 `maxWorkItemCount` 值设置为大于 `1024` 时无法正确发送 MQTT 消息。
- 修复了一个问题，该问题导致 OTA 更新代理忽略 `config.json` 的 `keepAlive` 属性中指定的 MQTT `KeepAlive` 周期。

- 常规性能改进和错误修复。

> **Important**
>
> 如果您使用流管理器将数据导出到云，请不要从较早的 v1.x 版 AWS IoT Greengrass Core 软件升级到 v1.11.3。如果是首次启用流管理器，我们强烈建议您首先安装最新版本的 AWS IoT Greengrass Core 软件。

1.11.1

错误修复和改进：

- 修复了导致流管理器内存使用量增大的问题。
- 修复了在 Greengrass 核心设备关闭时间超过 time-to-live 指定的 (TTL) 流数据0时段时，直播管理器会将直播的序列号重置为的问题。
- 修复了一个问题，该问题导致流管理器无法正确停止重新尝试将数据导出到 AWS Cloud。

1.11.0

新功能：

- Greengrass 核心上的遥测代理可以收集本地遥测数据并将其发布到 AWS Cloud。要检索遥测数据以供进一步处理，客户可以创建 Amazon EventBridge 规则并订阅目标。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。
- 本地 HTTP API 可以返回由 AWS IoT Greengrass 启动的本地工作线程进程的当前状态快照。有关更多信息，请参阅调用本地运行状况检查 API。
- 流管理器可以自动将数据导出到 Amazon S3 和 AWS IoT SiteWise。

  借助新的流管理器参数，您可以更新现有的流并暂停或恢复数据导出。
- 支持在核心上运行 Python 3.8.x Lambda 函数。
- `config.json` 中包含一个新的 `ggDaemonPort` 属性，可用于配置 Greengrass 核心 IPC 端口号。默认端口号为 8000。

  `config.json` 中包含一个新的 `systemComponentAuthTimeout` 属性，它可用于配置 Greengrass 核心 IPC 身份验证的超时时间。默认超时为 5000 毫秒。
- 每个 AWS IoT Greengrass 组的 AWS IoT 设备数量上限从 200 增加到了 2500。

  每个组的订阅数量上限从 1000 增加到了 10000。

有关更多信息，请参阅 AWS IoT Greengrass 端点和配额。

错误修复和改进：

- 进行了常规优化，可以降低 Greengrass 服务进程的内存利用率。
- 新的运行时配置参数 (`mountAllBlockDevices`) 允许 Greengrass 在设置 OverlayFS 后使用绑定挂载将所有块设备挂载到容器中。此功能解决了 `/usr` 不在 `/` 层次结构下时导致 Greengrass 部署失败的问题。
- 修复了 `/tmp` 是符号链接时导致 AWS IoT Greengrass 核心失败的问题。
- 修复了一个问题，使 Greengrass 部署代理可从 `mlmodel_public` 文件夹中移除未使用的机器学习模型工件。
- 常规性能改进和错误修复。

要在核心设备上安装 AWS IoT Greengrass Core 软件，请下载适用于您的架构和操作系统 (OS) 的软件包，然后按照《入门指南》中的步骤进行操作。

> **Tip**
>
> AWS IoT Greengrass 还提供了其他选项用于安装 AWS IoT Greengrass Core 软件。例如，您可以使用 Greengrass 设备设置来配置环境并安装最新版本的 AWS IoT Greengrass Core 软件。或者，在支持的 Debian 平台上，您可以使用 APT 软件包管理器 来安装或升级 AWS IoT Greengrass Core 软件。有关更多信息，请参阅 the section called “安装 AWS IoT Greengrass Core 软件”。

架构	操作系统	链接
Armv8 (AArch64)	Linux	下载
Armv8 (AArch64)	Linux (OpenWrt)	下载
Armv7l	Linux	下载
Armv7l	Linux (OpenWrt)	下载
Armv6l	Linux	下载

架构	操作系统	链接
x86_64	Linux	下载

Extended life versions

1.10.5

v1.10 中的新功能：

- 用于本地处理数据流并自动将其导出到 AWS Cloud 的流管理器。此功能需要 Greengrass 核心设备上的 Java 8。有关更多信息，请参阅 管理数据流。
- 用于在核心设备上运行 Docker 应用程序的新 Greengrass Docker 应用程序部署连接器。有关更多信息，请参阅 the section called “Docker 应用程序部署”。
- 一种新的物联网 SiteWise 连接器，可将工业设备数据从OPC-UA服务器发送到中的资产属性。AWS IoT SiteWise有关更多信息，请参阅 the section called “物联网 SiteWise”。
- 在未进行容器化的情况下运行的 Lambda 函数可以访问 Greengrass 组中的机器学习资源。有关更多信息，请参阅 the section called “访问机器学习资源”。
- 支持与 AWS IoT 的 MQTT 持久性会话。有关更多信息，请参阅 the section called “与 AWS IoT Core 的 MQTT 持久性会话”。
- 本地 MQTT 流量可通过默认端口 8883 以外的端口传输。有关更多信息，请参阅 the section called “用于本地消息收发的 MQTT 端口”。
- AWS IoT Greengrass Core 软件开发工具包中包含新的 `queueFullPolicy` 选项，可用于通过 Lambda 函数可靠地发布消息。
- 支持在核心上运行 Node.js 12.x Lambda 函数。

错误修复和改进：

- OpenSSL 1.1 可以配置集成了硬件安全的 O ver-the-air (OTA) 更新。
- 流管理器能够更灵活地应对文件数据损坏。
- 修复了导致使用 Linux 内核 5.1 及更高版本的设备上系统安装失败的问题。
- config.json 中包含一个新的 `mqttOperationTimeout` 属性，可用于在与 AWS IoT Core 连接的 MQTT 中设置发布、订阅和取消订阅操作的超时时间。
- 修复了导致流管理器内存使用量增大的问题。
- `config.json` 中包含一个新的 `systemComponentAuthTimeout` 属性，它可用于配置 Greengrass 核心 IPC 身份验证的超时时间。默认超时为 5000 毫秒。

- 修复了一个问题，该问题导致 OTA 更新代理忽略 `config.json` 的 `keepAlive` 属性中指定的 MQTT `KeepAlive` 周期。
- 修复了一个问题，该问题导致 `maxWorkItemCount` 值设置为大于 `1024` 时无法正确发送 MQTT 消息。
- 修复了一个问题，该问题导致向长时间生存的 Lambda 函数传送 MQTT 消息时出现延迟。
- 修复了一个问题，该问题导致在 Ubuntu 设备上 Snap 中运行的 AWS IoT Greengrass Core 软件在设备突然断电后停止响应。
- 常规性能改进和错误修复。

要在核心设备上安装 AWS IoT Greengrass Core 软件，请下载适用于您的架构和操作系统 (OS) 的软件包，然后按照《入门指南》中的步骤进行操作。

架构	操作系统	链接
Armv8 (AArch64)	Linux	下载
Armv8 (AArch64)	Linux (OpenWrt)	下载
Armv7l	Linux	下载
Armv7l	Linux (OpenWrt)	下载
Armv6l	Linux	下载
x86_64	Linux	下载

1.9.4

版本 1.9 中的新功能：

- 支持 Python 3.7 和 Node.js 8.10 Lambda 运行时。使用 Python 3.7 和 Node.js 8.10 运行时的 Lambda 函数现在可以在 AWS IoT Greengrass 核心上运行。（AWS IoT Greengrass 仍然支持 Python 2.7 和 Node.js 6.10 运行时。）
- 优化了 MQTT 连接。Greengrass 核心与 AWS IoT Core 建立的连接数较少。对于基于连接数的收费，此更改可以降低运营成本。
- 适用于本地 MQTT 服务器的椭圆曲线 (EC) 密钥。除了 RSA 密钥之外，本地 MQTT 服务器还支持 EC 密钥。（无论密钥类型如何，MQTT 服务器证书都具有 SHA-256 RSA 签名。）有关更多信息，请参阅 the section called “安全委托人”。

- Support fo OpenWrtr. AWS IoT Greengrass 核心软件 v1.9.2 或更高版本可以安装在采用 Armv8 (aarch64) 和 armv7L 架构的 OpenWrt 发行版上。目前， OpenWrt 不支持 ML 推理。
- 支持 Armv6l。AWS IoT GreengrassCore 软件 v1.9.3 或更高版本可在 Armv6l 架构上的 Raspbian 发行版中安装（例如，在 Raspberry Pi Zero 设备上）。
- OTA 在端口 443 上使用 ALPN 进行更新。使用端口 443 进行 MQTT 流量的 Greengrass 内核现在支持 (OTA) 软件更新。 over-the-air AWS IoT Greengrass使用应用层协议网络 (ALPN) TLS 扩展来启用这些连接。有关更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新和 the section called “通过端口 443 或网络代理进行连接”。

要在核心设备上安装 AWS IoT Greengrass Core 软件，请下载适用于您的架构和操作系统 (OS) 的软件包，然后按照《入门指南》中的步骤进行操作。

架构	操作系统	链接
Armv8 (AArch64)	Linux	下载
Armv8 (AArch64)	Linux (OpenWrt)	下载
Armv7l	Linux	下载
Armv7l	Linux (OpenWrt)	下载
Armv6l	Linux	下载
x86_64	Linux	下载

1.8.4

- 新功能：
  - 组中 Lambda 函数的可配置默认访问身份。这一组级别设置确定用于运行 Lambda 函数的默认权限。您可以设置用户 ID 和/或组 ID。各个 Lambda 函数可以覆盖其组的默认访问身份。有关更多信息，请参阅 the section called “为组中的 Lambda 函数设置默认访问身份”。
  - 通过端口 443 的 HTTPS 流量。HTTPS 通信可以配置为通过端口 443（而不是默认端口 8443）进行传输。这补充了 AWS IoT Greengrass 对应用层协议网络 (ALPN) TLS 扩展的支持，并允许所有 Greengrass 消息收发流量（包括 MQTT 和 HTTPS）使用端口 443。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”。

- AWS IoT 连接可预测命名的客户端 ID。此更改将启用对于 AWS IoT Device Defender 和 AWS IoT 生命周期事件的支持，因此，您可以针对连接、断开连接、订阅以及取消订阅事件接收通知。借助于可预测的命名，还可以更轻松地围绕连接 ID 创建逻辑（例如，基于证书属性创建订阅策略模板）。有关更多信息，请参阅 the section called “与 AWS IoT 的 MQTT 连接的客户端 ID”。

错误修复和改进：

- 修复了影子同步和设备证书管理器重新连接的问题。
- 常规性能改进和错误修复。

要在核心设备上安装 AWS IoT Greengrass Core 软件，请下载适用于您的架构和操作系统 (OS) 的软件包，然后按照《入门指南》中的步骤进行操作。

架构	操作系统	链接
Armv8 (AArch64)	Linux	下载
Armv7l	Linux	下载
x86_64	Linux	下载

下载此软件即表示您同意Greengrass Core 软件许可协议。

有关在设备上安装 AWS IoT Greengrass Core 软件的其他选项的信息，请参阅the section called “安装 AWS IoT Greengrass Core 软件”。

AWS IoT Greengrass Snap 软件

AWS IoT Greengrass Snap 1.11.x 使您能够在容器化环境中通过便捷的软件包以及所有必要的依赖项运行有限版本的 AWS IoT Greengrass。

> **Note**
>
> AWS IoT Greengrass Snap 适用于 AWS IoT Greengrass Core 软件 v1.11.x。AWS IoT Greengrass 不为 v1.10.x 提供 Snap。不受支持的版本不会收到错误修复或更新。
> AWS IoT Greengrass Snap 不支持连接器和机器学习 (ML) 推理。

有关更多信息，请参阅 the section called “在 Snap 中运行 AWS IoT Greengrass”。

AWS IoT Greengrass Docker 软件

AWS 提供 Dockerfile 和 Docker 映像，使您能够更轻松地在 Docker 容器中运行 AWS IoT Greengrass。

Dockerfile

Dockerfile 包含用于构建自定义 AWS IoT Greengrass 容器映像的源代码。可对映像进行修改，以在不同的平台架构上运行或减少映像的大小。有关说明，请参阅自述文件。

下载您的目标 AWS IoT Greengrass Core 软件版本。

v1.11

- 适用于 AWS IoT Greengrass v1.10.6 的 Dockerfile。

Extended life versions

v1.10

适用于 AWS IoT Greengrass v1.10.5 的 Dockerfile。

v1.9

适用于 AWS IoT Greengrass v1.9.4 的 Dockerfile。

v1.8

适用于 AWS IoT Greengrass v1.8.1 的 Dockerfile。

Docker 映像

Docker 映像在 Amazon Linux 2 (x86_64) 和 Alpine Linux（x86_64、Armv7l 或 AArch64）基本映像上安装了 AWS IoT Greengrass Core 软件和依赖项。您可以使用预构建的映像开始尝试 AWS IoT Greengrass。

> **Important**
>
> 2022 年 6 月 30 日，AWS IoT Greengrass 结束了对发布到 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 的 AWS IoT Greengrass Core 软件 v1.x Docker

镜像的维护。您可以继续从 Amazon ECR 和 Docker Hub 下载这些 Docker 映像，直至 2023 年 6 月 30（即维护结束 1 年后）为止。但在 2022 年 6 月 30 日维护结束后，AWS IoT Greengrass Core 软件 v1.x Docker 映像不再收到安全补丁或错误修复。如果运行的生产工作负载依赖于这些 Docker 映像，我们建议您使用提供的 Dockerfiles 构建自己的 Docker 映像。AWS IoT Greengrass有关更多信息，请参阅 AWS IoT Greengrass Version 1 维护策略。

从 Docker Hub 或 Amazon Elastic Container Registry (Amazon ECR) 中下载预构建映像。

- 若是 Docker Hub，请使用 *version* 标签下载特定版本的 Greengrass Docker 映像。要查找所有可用映像的标签，请查看 Docker Hub 上的 Tags 页面。
- 若是 Amazon ECR，请使用 `latest` 标签下载 Greengrass Docker 映像的最新可用版本。有关列出可用映像版本和从 Amazon ECR 下载映像的更多信息，请参阅在 Docker 容器中运行 AWS IoT Greengrass。

Warning

从AWS IoT Greengrass核心软件的 v1.11.6 开始，Greengrass Docker 镜像不再包含 Python 2.7，因为 Python 2.7 已于 2020 年推出，不再接收安全更新。 end-of-life 如果您选择更新这些 Docker 映像，我们建议您在将更新部署到生产设备之前，先验证应用程序是否可以使用新的 Docker 映像。如果使用 Greengrass Docker 映像的应用程序需要 Python 2.7，您可以修改 Greengrass Dockerfile，为您的应用程序包含 Python 2.7。

AWS IoT Greengrass 不为 AWS IoT Greengrass Core 软件 v1.11.1 提供 Docker 映像。

Note

默认情况下，`alpine-aarch64` 和 `alpine-armv7l` 映像只能在基于 ARM 的主机上运行。要在 x86 主机上运行这些映像，您可以安装 QEMU 并在主机上挂载 QEMU 库。例如：

```
docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
```

AWS IoT Greengrass Core 软件开发工具包

Lambda 函数使用 AWS IoT Greengrass Core 软件开发工具包与 AWS IoT Greengrass 核心进行本地交互。这允许已部署的 Lambda 函数执行：

- 与 AWS IoT Core 交换 MQTT 消息。
- 与 Greengrass 组中的连接器、客户端设备及其他 Lambda 函数交换 MQTT 消息。
- 与本地影子服务交互。
- 调用其他的本地 Lambda 函数。
- 访问密钥资源。
- 与流管理器交互。

从中下载适用于您的语言或平台的 AWS IoT Greengrass Core SDK GitHub。

- 适用于 Java 的 AWS IoT Greengrass Core 软件开发工具包
- 适用于 Node.js 的 AWS IoT Greengrass Core 软件开发工具包
- 适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包
- 适用于 C 的 AWS IoT Greengrass Core 软件开发工具包

有关更多信息，请参阅 AWS IoT Greengrass 核心开发工具包。

支持的机器学习运行时和库

要在 Greengrass 核心上执行推理，您必须为 ML 模型类型安装机器学习运行时或库。

AWS IoT Greengrass 支持以下 ML 模型类型。可以使用这些链接来查找有关如何为模型类型和设备平台安装运行时或库的信息。

- 深度学习运行时 (DLR)
- MXNet
- TensorFlow

机器学习示例

AWS IoT Greengrass 提供了可用于支持的 ML 运行时和库的示例。这些示例在 Greengrass Core 软件许可协议下发布。

Deep learning runtime (DLR)

下载适用于设备平台的示例：

- 适用于 Raspberry Pi 的 DLR 示例
- 适用于 NVIDIA Jetson TX2 的 DLR 示例
- 适用于 Intel Atom 的 DLR 示例

有关使用 DLR 示例的教程，请参阅the section called “如何配置优化的机器学习推理”。

MXNet

下载适用于设备平台的示例：

- 适用于 Raspberry Pi 的 MXNet 示例
- 适用于 NVIDIA Jetson TX2 的 MXNet 示例
- 适用于 Intel Atom 的 MXNet 示例

有关使用 MXNet 示例的教程，请参阅the section called “如何配置机器学习推理”。

TensorFlow

下载适用于您设备平台的 Tensorflow 示例。此示例适用于 Raspberry Pi、NVIDIA Jetson TX2 和 Intel Atom。

AWS IoT Greengrass ML 软件开发工具包软件

AWS IoT Greengrass 机器学习开发工具包 使您编写的 Lambda 函数能够使用本地机器学习模型，并将数据发送到 ML 反馈连接器以进行上传和发布。

v1.1.0

- Python 3.7

v1.0.0

- Python 2.7。

我们希望听到您的意见和建议

我们欢迎您提供反馈。要联系我们，请访问 AWS re:Post 并使用 AWS IoT Greengrass 标签。

安装 AWS IoT Greengrass Core 软件

AWS IoT Greengrass Core 软件将 AWS 功能扩展到 AWS IoT Greengrass 核心设备，使本地设备可以在本地操作它们生成的数据。

AWS IoT Greengrass 提供了多种选项用于安装 AWS IoT Greengrass Core 软件：

- 下载并解压缩 tar.gz 文件。
- 运行 Greengrass 设备设置脚本。
- 从 APT 存储库安装。

AWS IoT Greengrass 还提供了运行 AWS IoT Greengrass Core 软件的容器化环境。

- 在 Docker 容器中运行 AWS IoT Greengrass。
- 在 Snap 中运行 AWS IoT Greengrass。

下载并解压缩 AWS IoT Greengrass Core 软件包

选择适用于您的平台的 AWS IoT Greengrass Core 软件，下载 tar.gz 格式的文件并在设备上解压缩。您可以下载软件的最新版本。有关更多信息，请参阅 the section called “AWS IoT Greengrass Core 软件”。

运行 Greengrass 设备设置脚本

运行 Greengrass 设备设置以配置您的设备，安装最新的 AWS IoT Greengrass Core 软件版本，然后部署 Hello World Lambda 函数，几分钟就可以完成。有关更多信息，请参阅 the section called “快速入门：Greengrass 设备安装程序”。

从 APT 存储库安装 AWS IoT Greengrass Core 软件

> **Important**
>
> 自 2022 年 2 月 11 日起，您无法再从 APT 存储库安装或更新 AWS IoT Greengrass Core 软件。如果设备上已添添加 AWS IoT Greengrass 存储库，您必须从来源列表中移除存储库。从 APT 存储库运行该软件的设备将继续正常运转。我们建议您使用 tar 文件来更新 AWS IoT Greengrass Core 软件。

由 AWS IoT Greengrass 提供的 APT 存储库包括以下软件包：

- `aws-iot-greengrass-core`. 安装 AWS IoT Greengrass Core 软件。
- `aws-iot-greengrass-keyring`. 安装用于对 AWS IoT Greengrass 软件包存储库签名的 GnuPG (GPG) 密钥。

  下载此软件即表示您同意Greengrass Core 软件许可协议。

主题

- 使用 systemd 脚本管理 Greengrass 守护程序生命周期
- 使用 APT 存储库卸载 AWS IoT Greengrass Core 软件
- 删除 AWS IoT Greengrass Core 软件存储库来源

使用 systemd 脚本管理 Greengrass 守护程序生命周期

`aws-iot-greengrass-core` 软件包还会安装可用于管理 AWS IoT Greengrass Core 软件（守护程序）生命周期的 `systemd` 脚本。

- 要在引导过程中启动 Greengrass 守护程序，请执行以下操作：

```
systemctl enable greengrass.service
```

- 要启动 Greengrass 守护程序，请执行以下操作：

```
systemctl start greengrass.service
```

- 要停止 Greengrass 守护程序，请执行以下操作。

```
systemctl stop greengrass.service
```

- 要检查 Greengrass 守护程序的状态，请执行以下操作：

```
systemctl status greengrass.service
```

使用 APT 存储库卸载 AWS IoT Greengrass Core 软件

在卸载 AWS IoT Greengrass Core 软件时，您可以选择是要保留还是删除 AWS IoT Greengrass Core 软件的配置信息，例如设备证书、组信息和日志文件。

卸载 AWS IoT Greengrass Core 软件并保留配置信息

- 运行以下命令，以删除 AWS IoT Greengrass Core 软件包并将配置信息保留在 `/greengrass` 文件夹中。

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

卸载 AWS IoT Greengrass Core 软件并删除配置信息

1. 运行以下命令，以删除 AWS IoT Greengrass Core 软件包并从 `/greengrass folder` 中删除配置信息。

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

2. 从来源列表中删除 AWS IoT Greengrass Core 软件存储库。有关更多信息，请参阅 删除 AWS IoT Greengrass Core 软件存储库来源。

删除 AWS IoT Greengrass Core 软件存储库来源

不再需要从 APT 存储库安装或更新 AWS IoT Greengrass Core 软件时，您可以删除 AWS IoT Greengrass Core 软件存储库来源。2022 年 2 月 11 日之后，您必须将存储库从来源列表中移除，以免在运行 `apt update` 时出错。

从来源列表中删除 APT 存储库

- 运行以下命令，以从来源列表中删除 AWS IoT Greengrass Core 软件存储库。

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

在 Docker 容器中运行 AWS IoT Greengrass

AWS IoT Greengrass 提供 Dockerfile 和 Docker 映像，使您能够更轻松地在 Docker 容器中运行 AWS IoT Greengrass Core 软件。有关更多信息，请参阅 the section called “AWS IoT Greengrass Docker 软件”。

> **Note**
>
> 还可以在 Greengrass 核心设备上运行 Docker 应用程序。为此，应使用 Greengrass Docker 应用程序部署连接器。

在 Snap 中运行 AWS IoT Greengrass

AWS IoT Greengrass Snap 1.11.x 使您能够在容器化环境中通过便捷的软件包以及所有必要的依赖项运行有限版本的 AWS IoT Greengrass。

AWS IoT Greengrass 将于 2023 年 12 月 31 日结束对 AWS IoT Greengrass Core 软件版本 1.11.x Snap 的维护，这一消息已经发布在 snapcraft.io 上。当前运行 Snap 的设备将能够继续正常运转，直至另行通知。但在维护结束后，AWS IoT Greengrass Core Snap 将不会再收到安全补丁或错误修复。

Snap 概念

下方提供了一些基本的 Snap 概念，可帮助您了解如何使用 AWS IoT Greengrass Snap：

Channel

一个 Snap 组件，用于定义安装哪个版本的 snap 并跟踪其更新。Snap 会自动更新至当前频道的最新版本。

接口

一个 Snap 组件，用于授予对网络和用户文件等资源的访问权限。

要运行 AWS IoT Greengrass Spap，必须连接以下接口。请注意，必须首先连接 `greengrass-support-no-container`，并确保其始终保持连接。

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

其他接口是可选的。如果您的 Lambda 函数需要访问特定的资源，则可能需要连接到相应接口。

刷新

Snap 会自动更新。`snapd` 进程守护程序是 Snap 包管理器，默认为每天检查更新四次。每次更新检查称为一次刷新。当刷新发生时，进程守护程序会停止，接着 Snap 会更新，然后进程守护程序再重新启动。

有关更多信息，请参阅 Snapcraft 网站。

AWS IoT Greengrass Snap v1.11.x 新增功能

以下内容介绍了 AWS IoT Greengrass Snap 版本 1.11.x 中的新增和更改的功能。

- 此版本仅支持以 `snap_daemon` 用户，其呈现为用户 ID (UID) 和组 (GID) 584788 的形式。
- 此版本仅支持非容器化的 Lambda 函数。

> **Important**
>
> 由于非容器化的 Lambda 函数必须共享同一个用户 (snap_daemon)，因此 Lambda 函数彼此之间没有隔离。有关更多信息，请参阅使用组特定的配置控制 Greengrass Lambda 函数的执行。

- 此版本支持 C、C++、Java 8、Node.js 12.x、Python 2.7、Python 3.7 和 Python 3.8 运行时。

> **Note**
>
> 为了避免 Python 运行时冗余，Python 3.7 Lambda 函数实际上运行的是 Python 3.8 运行时。

AWS IoT Greengrass Snap 入门

以下过程可帮助您在设备上安装和配置 AWS IoT Greengrass Snap。

要求

要运行 AWS IoT Greengrass snap，您必须执行以下操作：

- 在支持的 Linux 发行版上运行 AWS IoT Greengrass Snap，例如 Ubuntu、Linux Mint、Debian 和 Fedora 等。
- 在您的设备安装 snapd 进程守护程序。snapd 进程守护程序包含 snap 工具，可用来管理您设备上的 Snap 环境。

有关支持的 Linux 发行版列表和安装说明，请参阅 Snap 文档中的安装 snapd。

安装和配置 AWS IoT Greengrass snap

以下教程介绍了如何在您的设备上安装和配置 AWS IoT Greengrass Snap。

> **Note**
>
> - 尽管本教程使用的是 Amazon EC2 实例（x86 t2.micro Ubuntu 20.04），但您可以使用 Raspberry Pi 等物理硬件运行 AWS IoT Greengrass snap。

- snapd 进程守护程序已预先安装在 Ubuntu 上。

1. 通过在设备的终端中运行以下命令来安装 core18 snap：

   ```
   sudo snap install core18
   ```

   core18 Snap 是一种基础 Snap，为运行时环境提供了常用的库。此 Snap 是基于 Ubuntu 18. 04 LTS 构建的。

2. 通过运行以下命令来升级 snapd：

   ```
   sudo snap install --channel=edge snapd; sudo snap refresh --channel=edge snapd
   ```

3. 运行 snap list 命令，以检查是否安装了 AWS IoT Greengrass Snap。

   以下示例响应显示 snapd 已经安装，但 aws-iot-greengrass 尚未安装。

   ```
   Name              Version               Rev    Tracking          Publisher   Notes
   amazon-ssm-agent  3.0.161.0             2996   latest/stable/…   aws#        classic
   core              16-2.48               10444  latest/stable     canonical#  core
   core18            20200929              1932   latest/stable     canonical#  base
   lxd               4.0.4                 18150  4.0/stable/…      canonical#  -
   snapd             2.48+git548.g929ccfb  10526  latest/edge       canonical#  snapd
   ```

4. 选择以下选项之一，以安装 AWS IoT Greengrass Snap 1.11.x。

   - 要安装 AWS IoT Greengrass snap，请运行下面的命令：

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

     响应示例：

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

   - 要从早期版本迁移到 v1.11.x 或更新为最新的可用补丁版本，请运行以下命令：

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

与其他 Snap 一样，AWS IoT Greengrass Snap 使用频道来管理次要版本。Snap 会自动更新至当前频道的最新可用版本。例如，如果您指定 `--channel=1.11.x`，则 AWS IoT Greengrass Snap 会更新到 v1.11.5。

您可以运行 `snap info aws-iot-greengrass` 命令来获取可供 AWS IoT Greengrass 使用的频道列表。

响应示例：

```
name:      aws-iot-greengrass
summary:   AWS supported software that extends cloud capabilities to local devices.
publisher: Amazon Web Services (aws#)
store-url: https://snapcraft.io/aws-iot-greengrass
contact:   https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass
license:   Proprietary
description: |
  AWS IoT Greengrass seamlessly extends AWS onto edge devices so they can act
 locally on the data
  they generate, while still using the cloud for management, analytics, and durable
 storage.
  AWS IoT Greenrgrass snap v1.11.0 enables you to run a limited version of AWS IoT
 Greengrass with
  all necessary dependencies in a containerized environment.
  The AWS IoT Greengrass snap doesn't support connectors and machine learning (ML)
 inference.
  By downloading this software you agree to the Greengrass Core Software License
 Agreement
  (https://s3-us-west-2.amazonaws.com/greengrass-release-license/greengrass-
license-v1.pdf).
  For more information, see Run AWS IoT Greengrass in a snap
  (https://docs.aws.amazon.com/greengrass/latest/developerguide/install-
ggc.html#gg-snap-support) in
  the AWS IoT Greengrass Developer.
  If you need help, try the AWS IoT Greengrass tag on AWS re:Post
  (https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass) or connect
 with an AWS IQ expert
  (https://iq.aws.amazon.com/services/aws/greengrass).
snap-id: SRDuhPJGj4XPxFNNZQKOTvURAp0wxKnd
channels:
  latest/stable:    1.11.3 2021-06-15 (59) 111MB -
  latest/candidate: 1.11.3 2021-06-14 (59) 111MB -
```

```
  latest/beta:      1.11.3 2021-06-14 (59) 111MB -
  latest/edge:      1.11.3 2021-06-14 (59) 111MB -
  1.11.x/stable:    1.11.3 2021-06-15 (59) 111MB -
  1.11.x/candidate: 1.11.3 2021-06-15 (59) 111MB -
  1.11.x/beta:      1.11.3 2021-06-15 (59) 111MB -
  1.11.x/edge:      1.11.3 2021-06-15 (59) 111MB -
```

5. 要访问 Lambda 函数所需的特定资源，您可以连接到其他接口。

   运行以下命令，以获取 AWS IoT Greengrass Snap 支持的接口列表：

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

   响应示例：

   ```
   Interface               Plug                                                Slot
               Notes
   camera                  aws-iot-greengrass:camera                           -
               -
   dvb                     aws-iot-greengrass:dvb                              -
               -
   gpio                    aws-iot-greengrass:gpio                             -
               -
   gpio-memory-control     aws-iot-greengrass:gpio-memory-control              -
               -
   greengrass-support      aws-iot-greengrass:greengrass-support-no-container
     :greengrass-support   -
   hardware-observe        aws-iot-greengrass:hardware-observe
     :hardware-observe     manual
   hardware-random-control aws-iot-greengrass:hardware-random-control          -
               -
   home                    aws-iot-greengrass:home-for-greengrassd             -
               -
   home                    aws-iot-greengrass:home-for-hooks                   :home
               manual
   hugepages-control       aws-iot-greengrass:hugepages-control
     :hugepages-control    manual
   i2c                     aws-iot-greengrass:i2c                              -
               -
   iio                     aws-iot-greengrass:iio                              -
               -
   joystick                aws-iot-greengrass:joystick                         -
               -
   ```

```
log-observe              aws-iot-greengrass:log-observe                    :log-
observe         manual
mount-observe            aws-iot-greengrass:mount-observe
  :mount-observe       manual
network                  aws-iot-greengrass:network
  :network             -
network-bind             aws-iot-greengrass:network-bind
  :network-bind        -
network-control          aws-iot-greengrass:network-control
  :network-control     -
opengl                   aws-iot-greengrass:opengl
  :opengl              -
optical-drive            aws-iot-greengrass:optical-drive
  :optical-drive       -
process-control          aws-iot-greengrass:process-control
  :process-control     -
raw-usb                  aws-iot-greengrass:raw-usb                        -
                -
removable-media          aws-iot-greengrass:removable-media                -
                -
serial-port              aws-iot-greengrass:serial-port                    -
                -
spi                      aws-iot-greengrass:spi                            -
                -
system-observe           aws-iot-greengrass:system-observe
  :system-observe      -
```

如果您在 Slot 列中看到连字符 (-)，则表示相应接口尚未连接。

6. 按照安装 AWS IoT Greengrass Core 软件的步骤操作，以创建 AWS IoT 事物、Greengrass 组、支持与 AWS IoT 进行安全通信的安全资源，以及 AWS IoT Greengrass Core 软件配置文件。配置文件 `config.json` 包含特定于 Greengrass 核心的配置，例如证书文件的位置和 AWS IoT 设备数据端点。

   > **Note**
   >
   > 如果您将该文件下载到其他设备，请按照此步骤操作，将文件传输到 AWS IoT Greengrass 核心设备。

7. 对于 AWS IoT Greengrass Snap，请务必更新 config.json 文件，如下所示：

   - 将 *`certificateId`* 的每个实例替换为证书和密钥文件名称中的证书 ID。

- 如果您下载的亚马逊根 CA 证书与亚马逊根 CA 1 的证书不同，请将 ca1.pem 的每个 *AmazonRootca1.pem* 实例替换为亚马逊根 CA 文件的名称。

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

8. 运行以下命令以添加您的 AWS IoT Greengrass 证书和配置文件：

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

部署 Lambda 函数

本节介绍如何将客户托管的 Lambda 函数部署到 AWS IoT Greengrass Snap 上。

> **Important**
>
> AWS IoT Greengrass Snap v1.11 仅支持非容器化的 Lambda 函数。

1. 运行以下命令，以启动 AWS IoT Greengrass 进程守护程序。

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

响应示例：

```
Started.
```

> **Note**
>
> 如果看到错误，您可以使用 `snap run` 命令来获取详细的错误消息。有关问故障排除的更多信息，请参阅错误：无法执行以下任务：-对 snap “” 的服务 ["greengrassd"] 运行服务命令 “start”（[start aws-iot-greengrass snap。 aws-iot-greengrass.greengrassd.service] 失败，退出状态 1：Job for snap。 aws-iot-greengrass.greengrassd.service 失败，因为控制进程退出时出现错误代码。 请参阅 “systemctl 状态快照”。 aws-iot-greengrass.greengrassd.service” 和 “journalctl-xe” 了解详情。 ）。

2. 运行以下命令，以确认进程守护程序正在运行：

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

响应示例：

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

3. 按照模块 3（第 1 部分）：AWS IoT Greengrass 上的 Lambda 函数中所述，开始创建和部署 Hello World Lambda 函数。但在部署 Lambda 函数之前，请先完成下面这一步。
4. 确保您的 Lambda 函数以 `snap_daemon` 用户身份在无容器模式下运行。要更新 Greengrass 组设置，请在 AWS IoT Greengrass 控制台中执行以下操作：

   a. 登录 AWS IoT Greengrass 控制台。

   b. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。

   c. 在 Greengrass 组下，选择目标组。

   d. 在组配置页面上的导航窗格中，选择 Lambda 函数选项卡。

   e. 在默认 Lambda 函数运行时环境下，选择编辑，然后执行以下操作：

i. 在默认系统用户和组中，选择其他用户 ID/组 ID，然后在系统用户 ID（数字）和系统组 ID（数字）中都输入 **584788**。
ii. 对于默认 Lambda 函数容器化，请选择无容器。
iii. 选择保存。

停止 AWS IoT Greengrass 进程守护程序

您可以使用 `snap stop` 命令来停止服务。

要停止 AWS IoT Greengrass 进程守护程序，请运行以下命令：

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

命令应返回 `Stopped.`。

要检查是否成功停止了 Snap，请运行以下命令：

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

响应示例：

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

卸载 AWS IoT Greengrass Snap

要卸载 AWS IoT Greengrass snap，请运行下面的命令：

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

响应示例：

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

AWS IoT Greengrass Snap 故障排除

以下信息可帮助您解决与 AWS IoT Greengrass snap 相关的问题。

收到权限被拒绝错误

解决方案：权限被拒绝错误的原因通常是缺少接口。如需缺失接口的列表和详细的故障排除信息，您可以使用 snappy-debug 工具。

运行以下命令来安装该工具。

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

响应示例：

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

在独立的终端会话中运行 sudo snappy-debug 命令。该操作将一直持续到出现权限被拒绝错误为止。

例如，如果 Lambda 函数尝试读取 $HOME 目录中的文件，您可能会得到以下响应：

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

此示例显示创建 /home/ubuntu/my-file.txt 文件导致了权限错误。它还建议您在 plugs 中添加 home。但是，此建议并不适用。home-for-greengrassd 和 home-for-hooks plug 仅被授予只读访问权限。

有关更多信息，请参阅 Snap 文档中的 snappy-debug Snap。

错误：无法执行以下任务：-对 snap “” 的服务 ["greengrassd"] 运行服务命令 “start” ([start aws-iot-greengrass snap。 aws-iot-greengrass.greengrassd.service] 失败，退出状态 1：Job for snap。 aws-iot-greengrass.greengrassd.service 失败，因为控制进程退出时出现错误代码。 请参阅 “systemctl 状态快照”。 aws-iot-greengrass.greengrassd.service” 和 “journalctl-xe” 了解详情。)

解决方案：当 `snap start aws-iot-greengrass` 命令无法启动 AWS IoT Greengrass 核心软件时，您可能会看到此错误。

有关更多故障排除信息，请运行以下命令：

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

响应示例：

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

此示例显示 AWS IoT Greengrass 无法找到 `config.json` 文件。您可以检查配置和证书文件。

/var/snap//aws-iot-greengrasscurrent//ggc-write-directorypackages/1.11.5/rootfs/merged 不是绝对路径或者是符号链接。

解决方案：AWS IoT Greengrass Snap 仅支持非容器化的 Lambda 函数。确保您的 Lambda 函数在无容器模式下运行。有关更多信息，请参阅《AWS IoT Greengrass Version 1 开发人员指南》中的选择 Lambda 函数容器化时的注意事项。

在您运行 sudo snap refresh snapd 命令后，snapd 进程守护程序无法重新启动。

解决方案：按照 安装和配置 AWS IoT Greengrass snap 中的第 6 到 8 步操作，将 AWS IoT Greengrass 证书和配置文件添加到 AWS IoT Greengrass Snap 中。

将 AWS IoT Greengrass Core 软件安装存档

当您升级到新版本的 AWS IoT Greengrass 核心软件时，您可以将当前安装的版本存档。这将保留您当前的安装环境，因此，您可以在同一硬件上测试新的软件版本。这还使您能够出于任何原因轻松回滚到您的存档版本。

存档当前安装并安装新版本

1. 下载您要升级到的 AWS IoT Greengrass Core 软件安装程序包。

2. 将此程序包复制到目标核心设备。有关描述如何传输文件的说明，请参阅此步骤。

> **Note**
>
> 稍后，您将当前的证书、密钥和配置文件复制到新的安装。

在核心设备终端中，在以下步骤中运行命令。

3. 确保 Greengrass 守护进程在您的核心设备上停止。

   a. 要检查进程守护程序是否正在运行，请执行以下操作：

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

   如果输出包含 root 的 /greengrass/ggc/packages/*ggc-version*/bin/daemon 条目，则表示进程守护程序正在运行。

   > **Note**
   >
   > 编写此过程的前提是 AWS IoT Greengrass 核心软件已安装在 /greengrass 目录中。

   b. 停止 进程守护程序：

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

4. 将当前 Greengrass 根目录移至其他目录。

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

5. 将新的软件解压缩到核心设备上。替换命令中的 *os-architecture* 和 *version* 占位符。

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

6. 将存档的证书、密钥和配置文件复制到新的安装。

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

7. 启动守护程序：

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

现在，您可以进行一个组部署来测试新的安装。如果出现故障，您可以还原已存档的安装。

还原已存档的安装

1. 停止守护程序。
2. 删除新的 /greengrass 目录。
3. 将 /greengrass_backup 目录移回 /greengrass。
4. 启动进程守护程序。

配置 AWS IoT Greengrass 核心

AWS IoT Greengrass 核心是在边缘环境中充当集线器或网关的 AWS IoT 事物（设备）。与其他 AWS IoT 设备一样，核心位于注册表中，具有设备影子，并使用设备证书在 AWS IoT Core 和 AWS IoT Greengrass 中进行身份验证。核心设备运行 AWS IoT Greengrass Core 软件，以使其可以管理 Greengrass 组的本地进程，如通信、影子同步和令牌交换。

AWS IoT Greengrass Core 软件提供了以下功能：

- 连接器和 Lambda 函数的部署和本地运行。
- 在本地处理数据流，并自动导出到 AWS Cloud。
- 使用托管订阅通过本地网络在设备、连接器和 Lambda 函数之间进行的 MQTT 消息传递。
- 使用托管订阅在 AWS IoT 与设备、连接器和 Lambda 函数之间进行的 MQTT 消息传递。
- 使用设备身份验证和授权确保设备和 AWS Cloud 之间的安全连接。
- 设备的本地影子同步。影子可配置为与 AWS Cloud 同步。
- 对本地设备和卷资源的受控访问。
- 用于运行本地推理的云训练机器学习模型的部署。
- 使设备能够发现 Greengrass 核心设备的自动 IP 地址检测。
- 全新的或更新的组配置的集中部署。下载配置数据后，核心设备将自动重启。
- 对用户定义的 Lambda 函数进行安全 over-the-air (OTA) 软件更新。

- 本地密钥的安全、加密的存储以及连接器和 Lambda 函数进行的受控访问。

AWS IoT Greengrass 核心配置文件

AWS IoT Greengrass 核心软件的配置文件为 `config.json`。它位于 */greengrass-root*`/config` 目录中。

> **Note**
>
> *greengrass-root* 表示在您的设备上安装 AWS IoT Greengrass Core 软件的路径。通常，这是 `/greengrass` 目录。
> 如果使用 AWS IoT Greengrass 控制台中的默认组创建选项，则将 `config.json` 文件部署到处于工作状态的核心设备中。

您可以运行以下命令以查看该文件的内容：

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

下面是一个 `config.json` 示例文件。这是您从 AWS IoT Greengrass 控制台中创建核心时生成的版本。

GGC v1.11

```
{
    "coreThing": {
        "caPath": "root.ca.pem",
        "certPath": "hash.cert.pem",
        "keyPath": "hash.private.key",
        "thingArn": "arn:partition:iot:region:account-id:thing/core-thing-name",
        "iotHost": "host-prefix-ats.iot.region.amazonaws.com",
        "ggHost": "greengrass-ats.iot.region.amazonaws.com",
        "keepAlive": 600,
        "ggDaemonPort": 8000,
        "systemComponentAuthTimeout": 5000
    },
    "runtime": {
        "maxWorkItemCount": 1024,
        "maxConcurrentLimit": 25,
        "lruSize": 25,
```

```
        "mountAllBlockDevices": "no",
        "cgroup": {
            "useSystemd": "yes"
        }
    },
    "managedRespawn": false,
    "crypto": {
        "principals": {
            "SecretsManager": {
                "privateKeyPath": "file:///greengrass/certs/hash.private.key"
            },
            "IoTCertificate": {
                "privateKeyPath": "file:///greengrass/certs/hash.private.key",
                "certificatePath": "file:///greengrass/certs/hash.cert.pem"
            }
        },
        "caPath": "file:///greengrass/certs/root.ca.pem"
    },
    "writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
    "pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}
```

config.json 文件支持以下属性：

coreThing

Field	描述	注意
caPath	AWS IoT 根 CA 的相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。 **Note** 确保您的端点与证书类型对应。
certPath	核心设备证书相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。

Field	描述	注意
`keyPath`	核心私有密钥的相对于 */greengrass-root* / `certs` 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 `crypto` 对象存在时，该属性将被忽略。
`thingArn`	表示 AWS IoT Greengrass 核心设备的 AWS IoT 事物的 Amazon 资源名称 (ARN)。	核心的 ARN 可在 AWS IoT Greengrass 控制台中的核心下方找到，通过运行 `aws greengrass get-core-definition-version` CLI 命令也可找到。
`iotHost`	您的 AWS IoT 终端节点。	该端点可在 AWS IoT 控制台中的设置下方找到，通过运行 `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI 命令也可找到。 此命令将返回 Amazon Trust Services (ATS) 终端节点。有关更多信息，请参阅服务器身份验证文档。 **Note** 确保您的端点与证书类型对应。 确保您的 端点与 AWS 区域 对应。

Field	描述	注意
ggHost	您的 AWS IoT Greengrass 终端节点。	这是您的 iotHost 终端节点，其主机前缀将替换为 greengrass（例如，greengrass-ats.iot. *region*.amazonaws.com ）。使用与 iotHost 相同的 AWS 区域。 **Note** 确保您的端点与证书类型对应。 确保您的 端点与 AWS 区域 对应。
iotMqttPort	可选。MQTT 与 AWS IoT 进行通信时所使用的端口号。	有效值为 8883 或 443。默认值为 8883。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
iotHttpPort	可选。用于创建到 AWS IoT 的 HTTPS 连接的端口号。	有效值为 8443 或 443。默认值为 8443。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
ggMqttPort	可选。用于通过本地网络进行 MQTT 通信的端口号。	有效值为 1024 到 65535。默认值为 8883。有关更多信息，请参阅 the section called “用于本地消息收发的 MQTT 端口”。

Field	描述	注意
ggHttpPort	可选。用于创建到 AWS IoT Greengrass 服务的 HTTPS 连接的端口号。	有效值为 8443 或 443。默认值为 8443。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
keepAlive	可选。MQTT KeepAlive 周期 (以秒为单位)。	有效范围在 30 到 1200 秒之间。默认值为 600。
networkProxy	可选。一个对象，它定义要连接到的代理服务器。	代理服务器可以是 HTTP，也可以是 HTTPS。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
mqttOperationTimeout	可选。允许 Greengrass 核心在与 AWS IoT Core 的 MQTT 连接中完成发布、订阅或取消订阅操作的时间（以秒为单位）。	默认值为 5。最小值为 5。
ggDaemonPort	可选。Greengrass 核心 IPC 端口号。	AWS IoT Greengrass v1.11.0 或更高版本中提供了此属性。 有效值在 1024 到 65535 之间。原定设置值为 8000。
systemComponentAuthTimeout	可选。允许 Greengrass core IPC 完成身份验证的时间（以毫秒为单位）。	AWS IoT Greengrass v1.11.0 或更高版本中提供了此属性。 有效值在 500 到 5000 之间。默认值是 5000。

runtime

Field	描述	注意
`maxWorkItem##`	可选。Greengrass 守护程序一次可以处理的最大工作项数。超过此限制的请求将被忽略。 工作项队列由系统组件、用户定义的 Lambda 函数和连接器共享。	默认值是 1024。最大值受设备硬件限制。 增加此值会增加 AWS IoT Greengrass 使用的内存。如果您希望核心接收大量 MQTT 消息流量，则可以增加此值。
`maxConcurrentLimit`	可选。Greengrass 守护程序可以拥有的并发未固定 Lambda 工作线程的最大数量。您可以指定一个不同的整数来覆盖此参数。	默认值为 25。最小值由 `lruSize` 定义。
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`mountAllBlock##`	Optional. Enables AWS IoT Greengrass to use bind mounts to mount all block devices into a container after setting up the OverlayFS.	AWS IoT Greengrass v1.11.0 或更高版本中提供了此属性。 有效值为 `yes` 和 `no`。默认值为 `no`。 如果您的 `/usr` 目录不在 `/` 层次结构之下，请将此值设置为 `yes`。
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		

Field	描述	注意
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are # or #. Run the `check_ggc_dependencies` script in 模块 1 to see if your device uses `systemd`.

crypto

`crypto` 包含一些属性，这些属性通过 PKCS # 11 和本地密钥存储支持硬件安全模块 (HSM) 上的私有密钥存储。有关更多信息，请参阅 the section called “安全委托人”、the section called “硬件安全性集成” 和 将密钥部署到核心。支持 HSM 上或文件系统中的私有密钥存储的配置。

Field	描述	注意
`caPath`	AWS IoT 根 CA 的绝对路径。	必须为以下格式的文件 URI：`file:///absolute/path/to/file` 。 **Note** 确保您的终端节点与证书类型对应。
`PKCS11`		
`OpenSSLEngine`	可选。OpenSSL 引擎 `.so` 文件（用于在 OpenSSL 上启用 PKCS # 11 支持）的绝对路径。	必须是文件系统上的文件的路径。 如果您使用 Greengrass OTA 更新代理来实现硬件安全性，则此属性是必需的。有关更多信息，请参阅the section called “配置 OTA 更新”。
`P11Provider`	PKCS#11 实施的 libdl-loadable 库的绝对路径。	必须是文件系统上的文件的路径。

Field	描述	注意
`slotLabel`	用于标识硬件模块的槽标签。	必须符合 PKCS # 11 标签规范。
`slotUserPin`	用于对模块的 Greengrass 核心进行身份验证的用户 PIN。	必须具有足够的权限才能使用配置的私有密钥执行 C_Sign。
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`###### privateKeyPath`	核心私有密钥的路径。	对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。
`IoTCertificate .certificatePath`	核心设备证书的绝对路径。	必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。
`MQTT ServerCertificate`	可选。核心为充当 MQTT 服务器或网关而将其证书结合使用的私有密钥。	

Field	描述	注意
`MQTT ServerCertificate # privateKeyPath`	本地 MQTT 服务器私有密钥的路径。	使用此值为本地 MQTT 服务器指定您自己的私有密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`*。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。 如果忽略此属性，AWS IoT Greengrass 会根据轮换设置对密钥进行轮换。如果指定，客户将负责对密钥进行轮换。
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 将密钥部署到核心.	
`SecretsManager.privateKeyPath`	本地 Secrets Manager 私有密钥的路径。	仅支持 RSA 密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`*。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。必须使用 PKCS#1 v1.5 填充机制生成私有密钥。

此外，还支持以下配置属性：

Field	描述	注意
mqttMaxConnectionRetryInterval	可选。在 MQTT 连接断开时的最大连接重试间隔（以秒为单位）。	将该值指定为无符号整数。默认值为 60。
managedRespawn	可选。指示 OTA 代理在更新之前需要运行自定义代码。	有效值为 true 或 false。有关更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新。
writeDirectory	可选。AWS IoT Greengrass 在其中创建所有读/写资源的写入目录。	有关更多信息，请参阅 为 AWS IoT Greengrass 配置写入目录。
pidFileDirectory	可选。AWS IoT Greengrass 将其进程 ID (PID) 存储在此目录下。	默认值为 /var/run。

Extended life versions

以下版本的 AWS IoT Greengrass Core 软件处于延长使用寿命阶段。此信息仅供参考。

GGC v1.10

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

```
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

config.json 文件支持以下属性：

coreThing

Field	描述	注意
caPath	AWS IoT 根 CA 的相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。 **Note** 确保您的端点与证书类型对应。
certPath	核心设备证书相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。

Field	描述	注意
`keyPath`	核心私有密钥的相对于 */greengrass-root* / `certs` 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 `crypto` 对象存在时，该属性将被忽略。
`thingArn`	表示 AWS IoT Greengrass 核心设备的 AWS IoT 事物的 Amazon 资源名称 (ARN)。	核心的 ARN 可在 AWS IoT Greengrass 控制台中的核心下方找到，通过运行 `aws greengrass get-core-definition-version` CLI 命令也可找到。
`iotHost`	您的 AWS IoT 终端节点。	该端点可在 AWS IoT 控制台中的设置下方找到，通过运行 `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI 命令也可找到。 此命令将返回 Amazon Trust Services (ATS) 终端节点。有关更多信息，请参阅服务器身份验证文档。 **Note** 确保您的端点与证书类型对应。 确保您的 端点与 AWS 区域 对应。

Field	描述	注意
`ggHost`	您的 AWS IoT Greengrass 终端节点。	这是您的 `iotHost` 终端节点，其主机前缀将替换为 `greengrass`（例如，`greengrass-ats.iot.`*`region`*`.amazonaws.com`）。使用与 `iotHost` 相同的 AWS 区域。 **Note** 确保您的端点与证书类型对应。 确保您的 端点与 AWS 区域 对应。
`iotMqttPort`	可选。MQTT 与 AWS IoT 进行通信时所使用的端口号。	有效值为 8883 或 443。默认值为 8883。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`iotHttpPort`	可选。用于创建到 AWS IoT 的 HTTPS 连接的端口号。	有效值为 8443 或 443。默认值为 8443。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`ggMqttPort`	可选。用于通过本地网络进行 MQTT 通信的端口号。	有效值为 1024 到 65535。默认值为 8883。有关更多信息，请参阅 the section called “用于本地消息收发的 MQTT 端口”。

Field	描述	注意
`ggHttpPort`	可选。用于创建到 AWS IoT Greengrass 服务的 HTTPS 连接的端口号。	有效值为 8443 或 443。默认值为 8443。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`keepAlive`	可选。MQTT `KeepAlive` 周期 (以秒为单位)。	有效范围在 30 到 1200 秒之间。默认值为 `600`。
`networkProxy`	可选。一个对象，它定义要连接到的代理服务器。	代理服务器可以是 HTTP，也可以是 HTTPS。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`mqttOperationTimeout`	可选。允许 Greengrass 核心在与 AWS IoT Core 的 MQTT 连接中完成发布、订阅或取消订阅操作的时间（以秒为单位）。	此属性包含在 AWS IoT Greengrass v1.10.2 及更高版本中。 默认值为 5。最小值为 5。

runtime

Field	描述	注意
`maxWorkItem##`	可选。Greengrass 守护程序一次可以处理的最大工作项数。超过此限制的请求将被忽略。 工作项队列由系统组件、用户定义的 Lambda 函数和连接器共享。	默认值是 1024。最大值受设备硬件限制。 增加此值会增加 AWS IoT Greengrass 使用的内存。如果您希望核心接收大量 MQTT 消息流量，则可以增加此值。
`maxConcurrentLimit`	可选。Greengrass 守护程序可以拥有的并发未固定 Lambda 工作线程的最大数	默认值为 25。最小值由 `lruSize` 定义。

Field	描述	注意
	量。您可以指定一个不同的整数来覆盖此参数。	
`lruSize`	Optional. Defines the minimum value for `maxConcurrentLimit` .	The default value is 25.
`postStartHealthCheckTimeout`	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are # or #. Run the `check_ggc_dependencies` script in 模块 1 to see if your device uses `systemd`.

crypto

`crypto` 包含一些属性，这些属性通过 PKCS # 11 和本地密钥存储支持硬件安全模块 (HSM) 上的私有密钥存储。有关更多信息，请参阅 the section called “安全委托人”、the section called “硬件安全性集成” 和 将密钥部署到核心。支持 HSM 上或文件系统中的私有密钥存储的配置。

Field	描述	注意
`caPath`	AWS IoT 根 CA 的绝对路径。	必须为以下格式的文件 URI：`file:///absolute/path/to/file` 。

Field	描述	注意
		Note 确保您的终端节点与证书类型对应。
PKCS11		
OpenSSLEngine	可选。OpenSSL 引擎 .so 文件（用于在 OpenSSL 上启用 PKCS # 11 支持）的绝对路径。	必须是文件系统上的文件的路径。 如果您使用 Greengrass OTA 更新代理来实现硬件安全性，则此属性是必需的。有关更多信息，请参阅the section called “配置 OTA 更新”。
P11Provider	PKCS#11 实施的 libdl-loadable 库的绝对路径。	必须是文件系统上的文件的路径。
slotLabel	用于标识硬件模块的槽标签。	必须符合 PKCS # 11 标签规范。
slotUserPin	用于对模块的 Greengrass 核心进行身份验证的用户 PIN。	必须具有足够的权限才能使用配置的私有密钥执行 C_Sign。
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	

Field	描述	注意
`###### privateKeyPath`	核心私有密钥的路径。	对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。
`IoTCertificate.certificatePath`	核心设备证书的绝对路径。	必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。
`MQTT ServerCertificate`	可选。核心为充当 MQTT 服务器或网关而将其证书结合使用的私有密钥。	
`MQTT ServerCertificate # privateKeyPath`	本地 MQTT 服务器私有密钥的路径。	使用此值为本地 MQTT 服务器指定您自己的私有密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。 如果忽略此属性，AWS IoT Greengrass 会根据轮换设置对密钥进行轮换。如果指定，客户将负责对密钥进行轮换。
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 将密钥部署到核心.	

Field	描述	注意
`SecretsManager .privateKeyPath`	本地 Secrets Manager 私有密钥的路径。	仅支持 RSA 密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。必须使用 PKCS#1 v1.5 填充机制生成私有密钥。

此外，还支持以下配置属性：

Field	描述	注意
`mqttMaxConnectionRetryInterval`	可选。在 MQTT 连接断开时的最大连接重试间隔（以秒为单位）。	将该值指定为无符号整数。默认值为 `60`。
`managedRespawn`	可选。指示 OTA 代理在更新之前需要运行自定义代码。	有效值为 `true` 或 `false`。有关更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新。
`writeDirectory`	可选。AWS IoT Greengrass 在其中创建所有读/写资源的写入目录。	有关更多信息，请参阅 为 AWS IoT Greengrass 配置写入目录。

GGC v1.0

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
```

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

config.json 文件支持以下属性：

coreThing

Field	描述	注意
caPath	AWS IoT 根 CA 的相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。 **Note** 确保您的端点与证书类型对应。

Field	描述	注意
`certPath`	核心设备证书相对于 */greengrass-root* / `certs` 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 `crypto` 对象存在时，该属性将被忽略。
`keyPath`	核心私有密钥的相对于 */greengrass-root* / `certs` 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 `crypto` 对象存在时，该属性将被忽略。
`thingArn`	表示 AWS IoT Greengrass 核心设备的 AWS IoT 事物的 Amazon 资源名称 (ARN)。	核心的 ARN 可在 AWS IoT Greengrass 控制台中的核心下方找到，通过运行 `aws greengrass get-core-definition-version` CLI 命令也可找到。
`iotHost`	您的 AWS IoT 终端节点。	该端点可在 AWS IoT 控制台中的设置下方找到，通过运行 `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI 命令也可找到。 此命令将返回 Amazon Trust Services (ATS) 终端节点。有关更多信息，请参阅服务器身份验证文档。 **Note** 确保您的端点与证书类型对应。 确保您的 端点与 AWS 区域 对应。

Field	描述	注意
`ggHost`	您的 AWS IoT Greengrass 终端节点。	这是您的 `iotHost` 终端节点，其主机前缀将替换为 `greengrass`（例如，`greengrass-ats.iot.`*`region`*`.amazonaws.com`）。使用与 `iotHost` 相同的 AWS 区域。 **Note** 确保您的端点与证书类型对应。 确保您的 端点与 AWS 区域 对应。
`iotMqttPort`	可选。MQTT 与 AWS IoT 进行通信时所使用的端口号。	有效值为 8883 或 443。默认值为 8883。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`iotHttpPort`	可选。用于创建到 AWS IoT 的 HTTPS 连接的端口号。	有效值为 8443 或 443。默认值为 8443。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`ggHttpPort`	可选。用于创建到 AWS IoT Greengrass 服务的 HTTPS 连接的端口号。	有效值为 8443 或 443。默认值为 8443。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`keepAlive`	可选。MQTT `KeepAlive` 周期 (以秒为单位)。	有效范围在 30 到 1200 秒之间。默认值为 `600`。

Field	描述	注意
networkProxy	可选。一个对象，它定义要连接到的代理服务器。	代理服务器可以是 HTTP，也可以是 HTTPS。有关更多信息，请参阅通过端口 443 或网络代理进行连接。

runtime

Field	描述	注意
maxConcurrentLimit	可选。Greengrass 守护程序可以拥有的并发未固定 Lambda 工作线程的最大数量。您可以指定一个不同的整数来覆盖此参数。	默认值为 25。最小值由 lruSize 定义。
lruSize	Optional. Defines the minimum value for maxConcurrentLimit .	The default value is 25.
postStartHealthCheckTimeout	Optional. The time (in milliseconds) after starting that the Greengrass daemon waits for the health check to finish.	The default timeout is 30 seconds (30000 ms).
cgroup		
useSystemd	Indicates whether your device uses systemd.	Valid values are # or #. Run the check_ggc_dependencies script in 模块 1 to see if your device uses systemd.

crypto

在 v1.7.0 中新增了 `crypto` 对象。它引入了一些属性，这些属性通过 PKCS # 11 和本地密钥存储支持硬件安全模块 (HSM) 上的私有密钥存储。有关更多信息，请参阅 the section called “安全委托人”、the section called “硬件安全性集成” 和 将密钥部署到核心。支持 HSM 上或文件系统中的私有密钥存储的配置。

Field	描述	注意
`caPath`	AWS IoT 根 CA 的绝对路径。	必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 **Note** 确保您的终端节点与证书类型对应。
`PKCS11`		
`OpenSSLEngine`	可选。OpenSSL 引擎 `.so` 文件（用于在 OpenSSL 上启用 PKCS # 11 支持）的绝对路径。	必须是文件系统上的文件的路径。 如果您使用 Greengrass OTA 更新代理来实现硬件安全性，则此属性是必需的。有关更多信息，请参阅the section called “配置 OTA 更新”。
`P11Provider`	PKCS#11 实施的 libdl-loadable 库的绝对路径。	必须是文件系统上的文件的路径。
`slotLabel`	用于标识硬件模块的槽标签。	必须符合 PKCS # 11 标签规范。
`slotUserPin`	用于对模块的 Greengrass 核心进行身份验证的用户 PIN。	必须具有足够的权限才能使用配置的私有密钥执行 C_Sign。

Field	描述	注意
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`###### privateKeyPath`	核心私有密钥的路径。	对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。
`IoTCertificate .certificatePath`	核心设备证书的绝对路径。	必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。
`MQTT ServerCertificate`	可选。核心为充当 MQTT 服务器或网关而将其证书结合使用的私有密钥。	

Field	描述	注意
`MQTT ServerCertificate # privateKeyPath`	本地 MQTT 服务器私有密钥的路径。	使用此值为本地 MQTT 服务器指定您自己的私有密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。 如果忽略此属性，AWS IoT Greengrass 会根据轮换设置对密钥进行轮换。如果指定，客户将负责对密钥进行轮换。
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 将密钥部署到核心.	
`SecretsManager.privateKeyPath`	本地 Secrets Manager 私有密钥的路径。	仅支持 RSA 密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。必须使用 PKCS#1 v1.5 填充机制生成私有密钥。

此外，还支持以下配置属性。

字段	描述	注意
mqttMaxConnectionRetryInterval	可选。在 MQTT 连接断开时的最大连接重试间隔（以秒为单位）。	将该值指定为无符号整数。默认值为 60。
managedRespawn	可选。指示 OTA 代理在更新之前需要运行自定义代码。	有效值为 true 或 false。有关更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新。
writeDirectory	可选。AWS IoT Greengrass 在其中创建所有读/写资源的写入目录。	有关更多信息，请参阅 为 AWS IoT Greengrass 配置写入目录。

GGC v1.8

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

```
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

config.json 文件支持以下属性。

coreThing

Field	描述	注意
caPath	AWS IoT 根 CA 的相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。 **Note** 确保您的端点与证书类型对应。
certPath	核心设备证书相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。
keyPath	核心私有密钥的相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。
thingArn	表示 AWS IoT Greengrass 核心设备的 AWS IoT 事物的 Amazon 资源名称 (ARN)。	核心的 ARN 可在 AWS IoT Greengrass 控制台中的核心下方找到，通过运行 aws greengrass get-core-definition-version CLI 命令也可找到。
iotHost	您的 AWS IoT 终端节点。	该端点可在 AWS IoT 控制台中的设置下方找到，通过

Field	描述	注意
		运行 `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI 命令也可找到。 此命令将返回 Amazon Trust Services (ATS) 终端节点。有关更多信息，请参阅服务器身份验证文档。 **Note** 确保您的端点与证书类型对应。确保您的端点与 AWS 区域 对应。
`ggHost`	您的 AWS IoT Greengrass 终端节点。	这是您的 `iotHost` 终端节点，其主机前缀将替换为 `greengrass`（例如，`greengrass-ats.iot.region.amazonaws.com`）。使用与 `iotHost` 相同的 AWS 区域。 **Note** 确保您的端点与证书类型对应。确保您的端点与 AWS 区域 对应。

Field	描述	注意
`iotMqttPort`	可选。MQTT 与 AWS IoT 进行通信时所使用的端口号。	有效值为 8883 或 443。默认值为 8883。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`iotHttpPort`	可选。用于创建到 AWS IoT 的 HTTPS 连接的端口号。	有效值为 8443 或 443。默认值为 8443。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`ggHttpPort`	可选。用于创建到 AWS IoT Greengrass 服务的 HTTPS 连接的端口号。	有效值为 8443 或 443。默认值为 8443。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`keepAlive`	可选。MQTT `KeepAlive` 周期 (以秒为单位)。	有效范围在 30 到 1200 秒之间。默认值为 `600`。
`networkProxy`	可选。一个对象，它定义要连接到的代理服务器。	代理服务器可以是 HTTP，也可以是 HTTPS。有关更多信息，请参阅通过端口 443 或网络代理进行连接。

runtime

Field	描述	注意
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are # or #. Run the `check_ggc_dependencies` script in 模块 1 to see if your device uses `systemd`.

crypto

在 v1.7.0 中新增了 `crypto` 对象。它引入了一些属性，这些属性通过 PKCS # 11 和本地密钥存储支持硬件安全模块 (HSM) 上的私有密钥存储。有关更多信息，请参阅 the section called “安全委托人”、the section called “硬件安全性集成” 和 将密钥部署到核心。支持 HSM 上或文件系统中的私有密钥存储的配置。

Field	描述	注意
`caPath`	AWS IoT 根 CA 的绝对路径。	必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 **Note** 确保您的终端节点与证书类型对应。
`PKCS11`		
`OpenSSLEngine`	可选。OpenSSL 引擎 `.so` 文件（用于在 OpenSSL 上启用 PKCS # 11 支持）的绝对路径。	必须是文件系统上的文件的路径。 如果您使用 Greengrass OTA 更新代理来实现硬件安全性，则此属性是必需的。有关更多信息，请参阅the section called “配置 OTA 更新”。
`P11Provider`	PKCS#11 实施的 libdl-loadable 库的绝对路径。	必须是文件系统上的文件的路径。
`slotLabel`	用于标识硬件模块的槽标签。	必须符合 PKCS # 11 标签规范。

Field	描述	注意
`slotUserPin`	用于对模块的 Greengrass 核心进行身份验证的用户 PIN。	必须具有足够的权限才能使用配置的私有密钥执行 C_Sign。
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`###### privateKeyPath`	核心私有密钥的路径。	对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。
`IoTCertificate.certificatePath`	核心设备证书的绝对路径。	必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。
`MQTT ServerCertificate`	可选。核心为充当 MQTT 服务器或网关而将其证书结合使用的私有密钥。	

Field	描述	注意
`MQTT ServerCertificate # privateKeyPath`	本地 MQTT 服务器私有密钥的路径。	使用此值为本地 MQTT 服务器指定您自己的私有密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。 如果忽略此属性，AWS IoT Greengrass 会根据轮换设置对密钥进行轮换。如果指定，客户将负责对密钥进行轮换。
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 将密钥部署到核心.	
`SecretsManager.privateKeyPath`	本地 Secrets Manager 私有密钥的路径。	仅支持 RSA 密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。必须使用 PKCS#1 v1.5 填充机制生成私有密钥。

此外，还支持以下配置属性：

Field	描述	注意
mqttMaxConnectionRetryInterval	可选。在 MQTT 连接断开时的最大连接重试间隔（以秒为单位）。	将该值指定为无符号整数。默认值为 60。
managedRespawn	可选。指示 OTA 代理在更新之前需要运行自定义代码。	有效值为 true 或 false。有关更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新。
writeDirectory	可选。AWS IoT Greengrass 在其中创建所有读/写资源的写入目录。	有关更多信息，请参阅 为 AWS IoT Greengrass 配置写入目录。

GGC v1.7

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

```
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

config.json 文件支持以下属性：

coreThing

Field	描述	注意
caPath	AWS IoT 根 CA 的相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。 **Note** 确保您的端点与证书类型对应。
certPath	核心设备证书相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。
keyPath	核心私有密钥的相对于 */greengrass-root* / certs 目录的路径。	为了向后兼容 1.7.0 之前的版本。当 crypto 对象存在时，该属性将被忽略。
thingArn	表示 AWS IoT Greengrass 核心设备的 AWS IoT 事物的 Amazon 资源名称 (ARN)。	核心的 ARN 可在 AWS IoT Greengrass 控制台中的核心下方找到，通过运行 aws greengrass get-core-definition-version CLI 命令也可找到。
iotHost	您的 AWS IoT 终端节点。	该端点可在 AWS IoT 控制台中的设置下方找到，通过

Field	描述	注意
		运行 `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI 命令也可找到。 此命令将返回 Amazon Trust Services (ATS) 终端节点。有关更多信息，请参阅服务器身份验证文档。 **Note** 确保您的端点与证书类型对应。确保您的端点与 AWS 区域 对应。
`ggHost`	您的 AWS IoT Greengrass 终端节点。	这是您的 `iotHost` 终端节点，其主机前缀将替换为 `greengrass`（例如，`greengrass-ats.iot.region.amazonaws.com`）。使用与 `iotHost` 相同的 AWS 区域。 **Note** 确保您的端点与证书类型对应。确保您的端点与 AWS 区域 对应。

Field	描述	注意
`iotMqttPort`	可选。MQTT 与 AWS IoT 进行通信时所使用的端口号。	有效值为 8883 或 443。默认值为 8883。有关更多信息，请参阅 通过端口 443 或网络代理进行连接。
`keepAlive`	可选。MQTT `KeepAlive` 周期 (以秒为单位)。	有效范围在 30 到 1200 秒之间。默认值为 `600`。
`networkProxy`	可选。一个对象，它定义要连接到的代理服务器。	代理服务器可以是 HTTP，也可以是 HTTPS。有关更多信息，请参阅通过端口 443 或网络代理进行连接。

runtime

Field	描述	注意
`cgroup`		
`useSystemd`	Indicates whether your device uses `systemd`.	Valid values are # or #. Run the `check_ggc_dependencies` script in 模块 1 to see if your device uses `systemd`.

crypto

`crypto` 对象（在版本 1.7.0 中添加）引入了一些属性，这些属性通过 PKCS # 11 和本地密钥存储支持硬件安全模块 (HSM) 上的私有密钥存储。有关更多信息，请参阅 the section called “硬件安全性集成”和 将密钥部署到核心。支持 HSM 上或文件系统中的私有密钥存储的配置。

Field	描述	注意
`caPath`	AWS IoT 根 CA 的绝对路径。	必须为以下格式的文件 URI：`file:///absolute/path/to/file` 。 **Note** 确保您的终端节点与证书类型对应。
`PKCS11`		
`OpenSSLEngine`	可选。OpenSSL 引擎 `.so` 文件（用于在 OpenSSL 上启用 PKCS # 11 支持）的绝对路径。	必须是文件系统上的文件的路径。 如果您使用 Greengrass OTA 更新代理来实现硬件安全性，则此属性是必需的。有关更多信息，请参阅the section called “配置 OTA 更新”。
`P11Provider`	PKCS#11 实施的 libdl-loadable 库的绝对路径。	必须是文件系统上的文件的路径。
`slotLabel`	用于标识硬件模块的槽标签。	必须符合 PKCS # 11 标签规范。
`slotUserPin`	用于对模块的 Greengrass 核心进行身份验证的用户 PIN。	必须具有足够的权限才能使用配置的私有密钥执行 C_Sign。
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	

Field	描述	注意
`###### privateKeyPath`	核心私有密钥的路径。	对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。
`IoTCertificate.certificatePath`	核心设备证书的绝对路径。	必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。
`MQTT ServerCertificate`	可选。核心为充当 MQTT 服务器或网关而将其证书结合使用的私有密钥。	
`MQTT ServerCertificate # privateKeyPath`	本地 MQTT 服务器私有密钥的路径。	使用此值为本地 MQTT 服务器指定您自己的私有密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。 如果忽略此属性，AWS IoT Greengrass 会根据轮换设置对密钥进行轮换。如果指定，客户将负责对密钥进行轮换。
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 将密钥部署到核心.	

Field	描述	注意
`SecretsManager .privateKeyPath`	本地 Secrets Manager 私有密钥的路径。	仅支持 RSA 密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。必须使用 PKCS#1 v1.5 填充机制生成私有密钥。

此外，还支持以下配置属性：

Field	描述	注意
`mqttMaxConnectionRetryInterval`	可选。在 MQTT 连接断开时的最大连接重试间隔（以秒为单位）。	将该值指定为无符号整数。默认值为 60。
`managedRespawn`	可选。指示 OTA 代理在更新之前需要运行自定义代码。	有效值为 `true` 或 `false`。有关更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新。
`writeDirectory`	可选。AWS IoT Greengrass 在其中创建所有读/写资源的写入目录。	有关更多信息，请参阅 为 AWS IoT Greengrass 配置写入目录。

GGC v1.6

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
```

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

> **Note**
>
> 如果使用 AWS IoT Greengrass 控制台中的默认组创建选项，则将 config.json 文件部署到处于工作状态的核心设备中，该文件指定了默认配置。

config.json 文件支持以下属性：

字段	描述	注意
caPath	AWS IoT 根 CA 的相对于 */greengrass-root* / certs 目录的路径。	将该文件保存在 */greengrass-root* / certs 中。
certPath	AWS IoT Greengrass 核心证书相对于 */greengrass-root* /certs 目录的路径。	将该文件保存在 */greengrass-root* / certs 中。
keyPath	AWS IoT Greengrass 核心私有密钥的相对于 */greengrass-root* / certs 目录的路径。	将该文件保存在 */greengrass-root* / certs 中。

字段	描述	注意
thingArn	表示 AWS IoT Greengrass 核心设备的 AWS IoT 事物的 Amazon 资源名称 (ARN)。	核心的 ARN 可在 AWS IoT Greengrass 控制台中的核心下方找到，通过运行 aws greengrass get-core-definition-version CLI 命令也可找到。
iotHost	您的 AWS IoT 终端节点。	该属性可在 AWS IoT 控制台中的设置下方找到，通过运行 aws iot describe-endpoint CLI 命令也可找到。
ggHost	您的 AWS IoT Greengrass 终端节点。	该值使用 greengrass.iot. *region*.amazonaws.com 格式。使用与 iotHost 相同的区域。
keepAlive	MQTT KeepAlive 周期 (以秒为单位)。	这是可选值。默认值为 600。
mqttMaxConnectionRetryInterval	在 MQTT 连接断开时的最大连接重试间隔（以秒为单位）。	将该值指定为无符号整数。这是可选值。默认值为 60。
useSystemd	指示您的设备是否使用 systemd。	有效值为 yes 或 no。运行check_ggc_dependencies 模块 1 中的 脚本以查看您的设备是否使用 systemd。
managedRespawn	这是一项可选的 over-the-air (OTA) 更新功能，它表明 OTA 代理需要在更新之前运行自定义代码。	有效值为 true 或 false。有关更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新。

字段	描述	注意
writeDirectory	AWS IoT Greengrass 在其中创建所有读/写资源的写入目录。	这是可选值。有关更多信息，请参阅 为 AWS IoT Greengrass 配置写入目录。

GGC v1.5

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

config.json 文件位于 */greengrass-root*/config 中并包含以下参数：

字段	描述	注意
caPath	AWS IoT 根 CA 的相对于 */greengrass-root* / certs 文件夹的路径。	将此文件保存在 */greengrass-root* / certs 文件夹下。
certPath	AWS IoT Greengrass 核心证书相对于 */greengrass-root* /certs 文件夹的路径。	将此文件保存在 */greengrass-root* / certs 文件夹下。

字段	描述	注意
keyPath	AWS IoT Greengrass 核心私有密钥相对于 /*greengrass-root* / certs 文件夹的路径。	将此文件保存在 /*greengrass-root* / certs 文件夹下。
thingArn	表示 AWS IoT Greengrass 核心设备的 AWS IoT 事物的 Amazon 资源名称 (ARN)。	核心的 ARN 可在 AWS IoT Greengrass 控制台中的核心下方找到，通过运行 aws greengrass get-core-definition-version CLI 命令也可找到。
iotHost	您的 AWS IoT 终端节点。	该属性可在 AWS IoT 控制台中的 设置 下方找到，通过运行 aws iot describe-endpoint 命令也可找到。
ggHost	您的 AWS IoT Greengrass 终端节点。	该值使用 greengrass.iot. *region*.amazonaws.com 格式。使用与 iotHost 相同的区域。
keepAlive	MQTT KeepAlive 周期 (以秒为单位)。	这是可选值。默认值为 600 秒。
useSystemd	指示您的设备是否使用 systemd。	有效值为 yes 或 no。运行check_ggc_dependencies 模块 1 中的 脚本以查看您的设备是否使用 systemd。
managedRespawn	这是一项可选的 over-the-air (OTA) 更新功能，它表明 OTA 代理需要在更新之前运行自定义代码。	有关更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新。

GGC v1.3

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

config.json 文件位于 */greengrass-root*/config 中并包含以下参数：

字段	描述	注意
caPath	AWS IoT 根 CA 的相对于 */greengrass-root* /certs 文件夹的路径。	将此文件保存在 */greengrass-root* /certs 文件夹下。
certPath	AWS IoT Greengrass 核心证书相对于 */greengrass-root* /certs 文件夹的路径。	将此文件保存在 */greengrass-root* /certs 文件夹下。
keyPath	AWS IoT Greengrass 核心私有密钥相对于 */greengrass-root* /certs 文件夹的路径。	将此文件保存在 */greengrass-root* /certs 文件夹下。

字段	描述	注意
thingArn	表示 AWS IoT Greengrass 核心的 AWS IoT 事物的 Amazon 资源名称 (ARN)。	您可以在 AWS IoT Greengrass 控制台中您的 AWS IoT 事物的定义下方找到该值。
iotHost	您的 AWS IoT 终端节点。	您可以在 AWS IoT 控制台中的设置下方找到该值。
ggHost	您的 AWS IoT Greengrass 终端节点。	您可以在 AWS IoT 控制台中的设置下方找到该值，并且前面附加有 greengrass. 。
keepAlive	MQTT KeepAlive 周期 (以秒为单位)。	这是可选值。默认值为 600 秒。
useSystemd	二进制标志 (如果您的设备使用了 systemd)。	值为 yes 或 no。使用模块 1 中的依赖项脚本中可以查看您的设备是否使用了 systemd。
managedRespawn	这是一项可选的 over-the-air (OTA) 更新功能，它表明 OTA 代理需要在更新之前运行自定义代码。	有关更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新。

GGC v1.1

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
```

```
    },
    "runtime": {
        "cgroup": {
            "useSystemd": "yes|no"
        }
    }
}
```

config.json 文件位于 */greengrass-root*/config 中并包含以下参数：

字段	描述	注意
caPath	AWS IoT 根 CA 的相对于 */greengrass-root* /certs 文件夹的路径。	将此文件保存在 */greengrass-root* /certs 文件夹下。
certPath	AWS IoT Greengrass 核心证书相对于 */greengrass-root* /certs 文件夹的路径。	将此文件保存在 */greengrass-root* /certs 文件夹下。
keyPath	AWS IoT Greengrass 核心私有密钥相对于 */greengrass-root* /certs 文件夹的路径。	将此文件保存在 */greengrass-root* /certs 文件夹下。
thingArn	表示 AWS IoT Greengrass 核心的 AWS IoT 事物的 Amazon 资源名称 (ARN)。	您可以在 AWS IoT Greengrass 控制台中您的 AWS IoT 事物的定义下方找到该值。
iotHost	您的 AWS IoT 终端节点。	您可以在 AWS IoT 控制台中的设置下方找到该值。
ggHost	您的 AWS IoT Greengrass 终端节点。	您可以在 AWS IoT 控制台中的设置下方找到该值，并且前面附加有 greengrass. 。

字段	描述	注意
keepAlive	MQTT KeepAlive 周期 (以秒为单位)。	这是可选值。默认值为 600 秒。
useSystemd	二进制标志 (如果您的设备使用了 systemd)。	值为 yes 或 no。使用模块 1 中的依赖项脚本中可以查看您的设备是否使用了 systemd。

GGC v1.0

在 AWS IoT Greengrass 核心 v1.0 中，config.json 部署到 *greengrass-root*/configuration。

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

config.json 文件位于 /*greengrass-root*/configuration 中并包含以下参数：

字段	描述	注意
caPath	AWS IoT 根 CA 的相对于 /*greengrass-root* /	将此文件保存在 /*greengrass-root* /

字段	描述	注意
	configuration/certs 文件夹的路径。	configuration/certs 文件夹下。
certPath	AWS IoT Greengrass 核心证书相对于 /*greengrass-root* /configuration/certs 文件夹的路径。	将此文件保存在 /*greengrass-root* /configuration/certs 文件夹下。
keyPath	AWS IoT Greengrass 核心私有密钥相对于 /*greengrass-root* /configuration/certs 文件夹的路径。	将此文件保存在 /*greengrass-root* /configuration/certs 文件夹下。
thingArn	表示 AWS IoT Greengrass 核心的 AWS IoT 事物的 Amazon 资源名称 (ARN)。	您可以在 AWS IoT Greengrass 控制台中您的 AWS IoT 事物的定义下方找到该值。
iotHost	您的 AWS IoT 终端节点。	您可以在 AWS IoT 控制台中的设置下方找到该值。
ggHost	您的 AWS IoT Greengrass 终端节点。	您可以在 AWS IoT 控制台中的设置下方找到该值，并且前面附加有 greengrass. 。
keepAlive	MQTT KeepAlive 周期 (以秒为单位)。	这是可选值。默认值为 600 秒。
useSystemd	二进制标志 (如果您的设备使用了 systemd)。	值为 yes 或 no。使用模块 1 中的依赖项脚本中可以查看您的设备是否使用了 systemd。

服务端点必须与根 CA 证书类型匹配

AWS IoT Core 和 AWS IoT Greengrass 终端节点必须与设备上的根 CA 证书的证书类型相对应。如果终端节点和证书类型不匹配，则设备与 AWS IoT Core 或 AWS IoT Greengrass 之间的身份验证尝试将失败。有关更多信息，请参阅《AWS IoT 开发人员指南》中的服务器身份验证。

如果您的设备使用 Amazon Trust Services (ATS) 根 CA 证书（首选方法），则设备还必须使用 ATS 端点执行设备管理和发现数据层面操作。ATS 终端节点包括 `ats` 分段，如 AWS IoT Core 终端节点的以下语法所示。

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

> **Note**
>
> 为了向后兼容，AWS IoT Greengrass目前在某些版本中支持传统的 VeriSign 根 CA 证书和端 AWS 区域点。如果您使用的是旧版 VeriSign根 CA 证书，我们建议您创建 ATS 端点并改用 ATS 根 CA 证书。否则，请确保使用相应的旧版端点。有关更多信息，请参阅《Amazon Web Services 一般参考》中的支持的旧版端点。

config.json 中的终端节点

在 Greengrass 核心设备上，终端节点是在 `config.json` 文件中的 `coreThing` 对象中指定的。`iotHost` 属性表示 AWS IoT Core 终端节点。`ggHost` 属性表示 AWS IoT Greengrass 终端节点。在以下示例片段中，这些属性指定了 ATS 终端节点。

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

AWS IoT Core 终端节点

可以通过适当的 `--endpoint-type` 参数运行 aws iot describe-endpoint CLI 命令来获取您的 AWS IoT Core 终端节点。

- 要返回 ATS 签名的终端节点，请运行：

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

- 要返回旧版 VeriSign 签名端点，请运行：

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

AWS IoT Greengrass 终端节点

您的 AWS IoT Greengrass 端点是您的 `iotHost` 端点，其主机前缀将替换为 greengrass。例如，ATS 签名的终端节点是 `greengrass-ats.iot.region.amazonaws.com`。这使用与 AWS IoT Core 终端节点相同的区域。

通过端口 443 或网络代理进行连接

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

Greengrass 核心使用 MQTT 消息收发协议和 TLS 客户端身份验证与 AWS IoT Core 进行通信。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经由 HTTP 代理连接到 Internet。

为了在此类情况下实现通信，AWS IoT Greengrass 允许以下配置：

- 在端口 443 上进行 MQTT 通信及 TLS 客户端身份验证。如果网络允许连接到端口 443，则可以将核心配置为使用端口 443（而非默认端口 8883）进行 MQTT 通信。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。

  AWS IoT Greengrass 使用应用层协议网络 (ALPN) TLS 扩展来启用该连接。与默认配置一样，端口 443 上的基于 TLS 的 MQTT 使用基于证书的客户端身份验证。

  当配置为使用与端口 443 的直接连接时，内核支持 over-the-air (OTA) AWS IoT Greengrass 软件更新。需要 AWS IoT Greengrass Core v1.10 或更高版本才能支持无线 (OTA) 更新。
- 通过端口 443 的 HTTPS 通信。AWS IoT Greengrass 默认情况下通过端口 8443 发送 HTTPS 通信，但您可以将其配置为使用端口 443。
- 通过网络代理连接。您可以配置一个网络代理服务器来充当连接到 Greengrass 核心的媒介。仅支持基本身份验证以及 HTTP 和 HTTPS 代理。

代理配置通过 `http_proxy`、`https_proxy` 和 `no_proxy` 环境变量传递给用户定义的 Lambda 函数。用户定义的 Lambda 函数必须使用这些传入的设置才能通过代理进行连接。Lambda 函数用于建立连接的公用库（例如 boto3 或 cURL 和 python `requests` 程序包）通常默认使用这些环境变量。如果 Lambda 函数还指定了这些相同的环境变量，则 AWS IoT Greengrass 不会覆盖它们。

> **Important**
>
> 配置为使用网络代理的 Greengrass 核心不支持 OTA 更新。

通过端口 443 配置 MQTT

此功能需要 AWS IoT Greengrass Core v1.7 或更高版本。

此过程使 Greengrass 核心能够使用端口 443 与 AWS IoT Core 进行 MQTT 消息传递。

1. 运行以下命令以停止 Greengrass 进程守护程序：

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

2. 打开 *greengrass-root*/config/config.json 以作为 su 用户进行编辑。
3. 在 `coreThing` 对象中，添加 `iotMqttPort` 属性，并将该属性的值设置为 **443**，如以下示例所示。

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

4. 启动守护程序。

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

通过端口 443 配置 HTTPS

此功能需要 AWS IoT Greengrass Core v1.8 或更高版本。

该过程将核心配置为使用 443 进行 HTTPS 通信。

1. 运行以下命令以停止 Greengrass 进程守护程序：

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

2. 打开 *greengrass-root*/config/config.json 以作为 su 用户进行编辑。
3. 在 coreThing 对象中，添加 iotHttpPort 和 ggHttpPort 属性，如以下示例所示。

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

4. 启动进程守护程序。

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

配置网络代理

此功能需要 AWS IoT Greengrass Core v1.7 或更高版本。

该过程允许 AWS IoT Greengrass 通过 HTTP 或 HTTPS 网络代理连接到 Internet。

1. 运行以下命令以停止 Greengrass 进程守护程序：

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

2. 打开 *greengrass-root*/config/config.json 以作为 su 用户进行编辑。
3. 在 coreThing 对象中，添加 networkProxy 对象，如以下示例所示。

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

4. 启动进程守护程序。

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

networkProxy 对象

使用 networkProxy 对象指定有关网络代理的信息。此对象具有以下属性。

字段	描述
noProxyAddresses	可选。不使用代理的 IP 地址或主机名的逗号分隔列表。

proxy	要连接的代理。代理具有以下属性。
	• url。代理服务器的 URL，格式为 scheme://userinfo@host:port ：
	  • scheme。方案。必须为 http 或 https。
	  • userinfo。可选。用户名和密码信息。如果指定，则忽略 username 和 password 字段。
	  • host。代理服务器的主机名或 IP 地址。
	  • port。可选。端口号。如果未指定，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时所使用的用户名。
	• password。可选。对代理服务器进行身份验证时所使用的密码。

允许终端节点

Greengrass 设备和 AWS IoT Core 或 AWS IoT Greengrass 之间的通信必须经过身份验证。此身份验证基于注册的 X.509 设备证书和加密密钥。要允许经身份验证的请求通过代理而无需其他加密，请允许以下终端节点。

Endpoint	端口	描述
greengrass. *region*.amazonaws.com	443	用于进行组管理的控制层面操作。
prefix-ats.iot. *region*.amazonaws.com 或者 *prefix*.iot.*region*.amazonaws.com	MQTT：8883 或 443 HTTPS：8443 或 443	用于设备管理的数据平面操作，例如影子同步。 允许仅使用其中一个端点或同时使用这两个端点，具体取决于您的核心设备和客户端设备是使用 Amazon Trust Services（首选）根 CA 证书，还是使用旧式根 CA 证书，或者是同时使用两者。有关更多信息，请参阅 the section called “服务端点必须与证书类型匹配”。
greengrass-ats.iot. *region*.amazonaws.com	8443 或 443	用于设备发现操作。

Endpoint	端口	描述
或者 `greengrass.iot.` *`region`*`.amazonaws.com`		允许仅使用其中一个端点或同时使用这两个端点，具体取决于您的核心设备和客户端设备是使用 Amazon Trust Services（首选）根 CA 证书，还是使用旧式根 CA 证书，或者是同时使用两者。有关更多信息，请参阅 the section called “服务端点必须与证书类型匹配”。 **Note** 在端口 443 上连接的客户端必须实现应用层

Endpoint	端口	描述
		协议协商 (ALPN) TLS 扩展，并且必须在 ProtocolNameList 中作为 ProtocolName 传递 x-amzn-http-tp-ca。更多信息，请参阅 AWS IoT 开发人员指南中的协议。

Endpoint	端口	描述
`*.s3.amazonaws.com`	443	用于部署操作和 over-the-air 更新。该格式包括 * 字符，因为终端节点前缀由内部控制，并且可能随时更改。
`logs.region.amazonaws.com`	443	如果 Greengrass 组配置为写入日志，则为必填项。CloudWatch

为 AWS IoT Greengrass 配置写入目录

此功能适用于 AWS IoT Greengrass Core v1.6 及更高版本。

默认情况下，AWS IoT Greengrass Core 软件部署在单个根目录中，AWS IoT Greengrass 在其中执行所有读取和写入操作。不过，您可以配置 AWS IoT Greengrass 以使用单独的目录执行所有写入操作，包括创建目录和文件。在这种情况下，AWS IoT Greengrass 使用两个顶级目录：

- *greengrass-root* 目录，您可以将其保留为读写目录，或者选择将其指定为只读目录。它包含应在运行时保持不变的 AWS IoT Greengrass 核心软件和其他关键组件，如证书和 `config.json`。
- 指定的写入目录。它包含可写内容，如日志、状态信息以及部署的用户定义的 Lambda 函数。

该配置将生成以下目录结构。

Greengrass 根目录

```
greengrass-root/
|-- certs/
```

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

写入目录

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

配置写入目录

1. 运行以下命令以停止 AWS IoT Greengrass 守护程序：

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

2. 打开 *greengrass-root*/config/config.json 以作为 su 用户进行编辑。
3. 添加 writeDirectory 以作为参数并指定目标目录路径，如以下示例所示。

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

> **Note**
>
> 您可以按照所需的频率更新 writeDirectory 设置。在更新该设置后，AWS IoT Greengrass 在下次启动时使用新指定的写入目录，但不会从以前的写入目录中迁移内容。

4. 现已配置您的写入目录，您可以选择将 *greengrass-root* 目录指定为只读目录。有关说明，请参阅将 Greengrass 根目录指定为只读目录。

   否则，启动 AWS IoT Greengrass 守护程序：

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

将 Greengrass 根目录指定为只读目录

只有在要将 Greengrass 根目录指定为只读目录时，才需要使用这些步骤。在开始之前，必须配置写入目录。

1. 授予对所需目录的访问权限：

   a. 为 `config.json` 所有者授予读写权限。

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

   b. 将 ggc_user 指定为 certs 和 system Lambda 目录的所有者。

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

   > **Note**
   >
   > 默认情况下，ggc_user 和 ggc_group 账户用于运行系统 Lambda 函数。如果您已配置组级别的默认访问身份来使用不同的账户，则应改为向该用户 (UID) 和组 (GID) 授予权限。

2. 使用所需的方法将 *greengrass-root* 目录指定为只读目录。

   > **Note**
   >
   > 要将 *greengrass-root* 目录指定为只读目录，一种方法是以只读方式挂载该目录。但是，要将 over-the-air (OTA) 更新应用于已装载目录中的AWS IoT Greengrass核心软件，必须先卸载该目录，然后在更新后重新挂载。您可以将这些 `umount` 和 `mount` 操作添加到 `ota_pre_update` 和 `ota_post_update` 脚本中。有关 OTA 更新的更多信息，请参阅the section called “Greengrass OTA 更新代理”和the section called “使用 OTA 更新的托管 Respawn”。

3. 启动守护程序。

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

   如果未正确设置步骤 1 中的权限，则不会启动守护程序。

配置 MQTT 设置

在 AWS IoT Greengrass 环境中，本地客户端设备、Lambda 函数、连接器和系统组件既可彼此通信，也可与 AWS IoT Core 通信。所有通信都经过核心，核心可管理负责授权实体之间的 MQTT 通信的订阅。

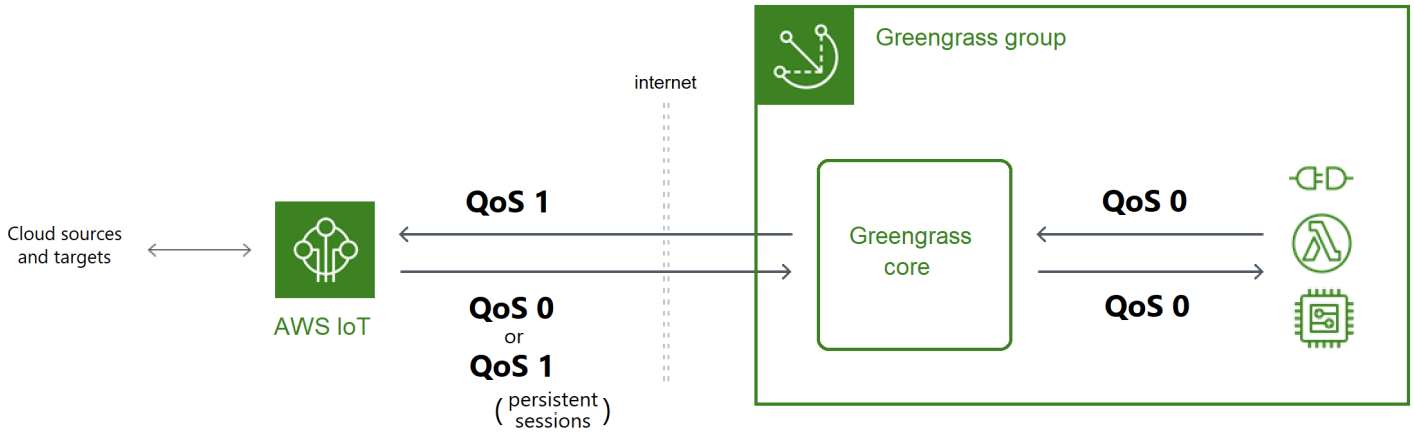
有关可为 AWS IoT Greengrass 配置 MQTT 设置的信息，请参阅以下各节：

- the section called “消息服务质量”
- the section called “MQTT 消息队列”
- the section called “与 AWS IoT Core 的 MQTT 持久性会话”
- the section called “与 AWS IoT 的 MQTT 连接的客户端 ID”
- 用于本地消息收发的 MQTT 端口
- the section called “在与 AWS Cloud的 MQTT 连接中的发布、订阅和取消订阅操作的超时”

> **Note**
>
> OPC-UA 是一种用于工业通信的信息交换标准。要在 Greengrass 内核上实现对 OPC-UA 的支持，你可以使用物联网连接器。 SiteWise 该连接器可将工业设备数据从 OPC-UA 服务器发送到 AWS IoT SiteWise 中的资产属性。

消息服务质量

AWS IoT Greengrass 支持 0 级或 1 级服务质量 (QoS)，具体取决于您的配置以及通信目标和方向。Greengrass 核心同时作为客户端（用于与 AWS IoT Core 通信）和消息代理（用于本地网络上的通信）。



有关 MQTT 和 QoS 的更多信息，请参阅 MQTT 网站上的入门。

与 AWS Cloud 的通信

- 出站消息使用 QoS 1

  核心使用 QoS 1 向 AWS Cloud 目标发送消息。AWS IoT Greengrass 使用 MQTT 消息队列来处理这些消息。如果 AWS IoT 未确认消息传输，则稍后会尝试在后台处理消息。如果队列已满，则消息无法重试。消息传输确认可以帮助最大限度减少间歇性连接造成的数据丢失。

  由于 AWS IoT 的出站消息使用 QoS 1，因此 Greengrass 核心可以发送消息的最大速率取决于核心与 AWS IoT 之间的延迟。每次核心发送消息时，它都会等到 AWS IoT 确认消息后再发送下一条消息。例如，如果核心与其 AWS 区域 之间的往返时间 为 50 毫秒，则核心每秒最多可以发送 20 条消息。选择核心连接的 AWS 区域 时，请考虑该行为。要将大容量 IoT 数据摄取到 AWS Cloud，您可以使用流管理器。

  有关 MQTT 消息队列的更多信息，包括如何配置可以保存发往 AWS Cloud 目标的消息的本地存储缓存，请参阅the section called “MQTT 消息队列”。

- 入站消息使用 QoS 0（默认）或 QoS 1

  默认情况下，核心以 QoS 0 订阅来自 AWS Cloud 源的消息。如果您启用持久会话，则核心将以 QoS 1 进行订阅。这有助于最大限度地减少间歇性连接造成的数据丢失。要管理这些订阅的 QoS，请在本地后台处理程序系统组件上配置持久性设置。

  有关更多信息，包括如何使核心能够与 AWS Cloud 目标建立持久会话，请参阅 the section called “与 AWS IoT Core 的 MQTT 持久性会话”。

与本地目标通信

所有本地通信都使用 QoS 0。核心尝试向本地目标发送消息，该目标可以是 Greengrass Lambda 函数、连接器或客户端设备。核心不会存储消息，也不会确认交付。消息可置于组件之间的任何位置。

> **Note**
>
> 虽然 Lambda 函数之间的直接通信不使用 MQTT 消息收发，但行为是相同的。

云目标的 MQTT 消息队列

发送到 AWS Cloud 目标的 MQTT 消息将排入队列以等待处理。排队的消息按照先进先出 (FIFO) 顺序进行处理。在处理消息并发布到 AWS IoT Core 后，将从队列中删除消息。

默认情况下，Greengrass 核心将发送到 AWS Cloud 目标的未处理消息存储在内存中。您可以配置核心以改为在本地存储缓存中存储未处理的消息。与内存中存储不同，本地存储缓存可以在核心重新启动时永久保存消息（例如，在组部署或设备重启后），因此，AWS IoT Greengrass 可以继续处理这些消息。您也可以配置存储大小。

> **Warning**
>
> Greengrass 核心在失去连接时可能会将重复的 MQTT 消息加入队列中，这是因为 Greengrass 核心会在 MQTT 客户机检测到其脱机之前重试一次发布操作。要避免云目标的 MQTT 消息重复，请将核心的 `keepAlive` 值配置为小于其 `mqttOperationTimeout` 值的一半。有关更多信息，请参阅 AWS IoT Greengrass 核心配置文件。

AWS IoT Greengrass 使用后台处理程序系统组件（`GGCloudSpooler` Lambda 函数）来管理消息队列。您可以使用以下 `GGCloudSpooler` 环境变量来配置存储设置。

- GG_CONFIG_STORAGE_TYPE。消息队列位置。有效值有：
  - `FileSystem`。将未处理的消息存储在物理核心设备磁盘上的本地存储缓存中。在核心重新启动时，将保留排队的消息以进行处理。在处理后，将删除消息。
  - `Memory` (默认值)。在内存中存储未处理的消息。在核心重新启动时，排队的消息将丢失。

    该选项是针对具有受限硬件功能的设备优化的。在使用该配置时，我们建议您部署组或在服务中断最少的情况下重新启动设备。

- GG_CONFIG_MAX_SIZE_BYTES。存储大小（以字节为单位）。该值可以是任何大于或等于 262144 (256 KB) 的非负整数；较小的大小将导致 AWS IoT Greengrass Core 软件无法启动。默认大小为 2.5 MB。在达到大小限制时，最早的排队消息将替换为新消息。

> **Note**
>
> 此功能适用于 AWS IoT Greengrass Core v1.6 及更高版本。更低版本使用内存中存储，并且队列大小为 2.5 MB。您无法为早期版本配置存储设置。

在本地存储中缓存消息

您可以配置 AWS IoT Greengrass 以将消息缓存到文件系统中，以便在核心重新启动时永久保存这些消息。为此，您需要部署一个函数定义版本，其中 `GGCloudSpooler` 函数将存储类型设置为 `FileSystem`。您必须使用 AWS IoT Greengrass API 配置本地存储缓存。您无法在控制台中执行该操作。

以下过程使用 `create-function-definition-version` CLI 命令配置后台处理程序，以便将排队的消息保存到文件系统中。它还配置 2.6 MB 队列大小。

1. 获取目标 Greengrass 组和组版本的 ID。此过程假定这是最新的组和组版本。以下查询将返回最近创建的组。

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

   或者，您也可以按名称查询。系统不要求组名称是唯一的，所以可能会返回多个组。

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

   > **Note**
   >
   > 您还可以在 AWS IoT 控制台中找到这些值。组 ID 显示在组的设置页面上。组版本 ID 显示在组的部署选项卡上。

2. 从输出中的目标组复制 `Id` 和 `LatestVersion` 值。
3. 获取最新的组版本。

- 将 *group-id* 替换为复制的 Id。
- 将 *latest-group-version-id* 替换为复制的 LatestVersion。

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

4. 从输出中的 Definition 对象复制 CoreDefinitionVersionArn 以及所有其他组组件（FunctionDefinitionVersionArn 除外）的 ARN。在创建新的组版本时，将使用这些值。
5. 在输出的 FunctionDefinitionVersionArn 中，复制函数定义的 ID。该 ID 是 ARN 中的 functions 段后面的 GUID，如以下示例所示。

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

> **Note**
>
> 或者，您可以通过运行 create-function-definition 命令以创建一个函数定义，然后从输出中复制该 ID。

6. 在函数定义中添加一个函数定义版本。
   - *function-definition-id*替换Id为你为函数定义复制的。
   - *arbitrary-function-id*替换为函数的名称，例如**spooler-function**。
   - 将要在该版本中包含的任何 Lambda 函数添加到 functions 数组中。您可以使用 get-function-definition-version 命令从现有的函数定义版本中获取 Greengrass Lambda 函数。

> **Warning**
>
> 确保为 GG_CONFIG_MAX_SIZE_BYTES 指定大于或等于 262144 的值。较小的大小将导致 AWS IoT Greengrass 核心软件无法启动。

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

> **Note**
>
> 如果您之前设置 GG_CONFIG_SUBSCRIPTION_QUALITY 环境变量以支持与 AWS IoT Core 的持久性会话，请将其包含在此函数实例中。

7. 从输出中复制函数定义版本的 Arn。
8. 创建一个包含系统 Lambda 函数的组版本。

   - 将 *group-id* 替换为组的 Id。
   - *core-definition-version-arn*替换为您从最新群组版本中复制的版本。CoreDefinitionVersionArn
   - *function-definition-version-arn*替换为你Arn为新函数定义版本复制的函数。
   - 替换从最新组版本中复制的其他组组件（例如 SubscriptionDefinitionVersionArn 或 DeviceDefinitionVersionArn）的 ARN。
   - 删除任何未使用的参数。例如，如果组版本不包含任何资源，请删除 --resource-definition-version-arn。

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

9. 从输出中复制 Version。这是新组版本的 ID。

10. 用新组版本替换组。

    - 使用为组复制的 *##* group-idId。
    - *group-version-id*替换为你Version为新群组版本复制的版本。

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

要更新存储设置，您需要使用 AWS IoT Greengrass API 创建新的函数定义版本，其中包含具有更新的配置的 GGCloudSpooler 函数。接下来，将函数定义版本添加到新的组版本（以及其他组组件）中，然后部署该组版本。如果要还原默认配置，您可以部署不包含 GGCloudSpooler 函数的函数定义版本。

控制台中不显示这个系统 Lambda 函数。不过，在将函数添加到最新组版本后，它将包含在从控制台指定的部署中（除非您使用 API 替换或删除该函数）。

与 AWS IoT Core 的 MQTT 持久性会话

此功能适用于 AWS IoT Greengrass Core v1.10 及更高版本。

Greengrass 核心可以与 AWS IoT 消息代理建立持久性会话。持久会话是一种持续连接，允许核心接收在核心离线时发送的消息。核心是连接中的客户端。

在持久性会话中，AWS IoT 消息代理可保存核心在连接期间所做的所有订阅。如果核心断开连接，则 AWS IoT 消息代理会存储未确认的消息以及作为 QoS 1 发布并发往本地目标（例如 Lambda 函数和客户端设备）的新消息。当核心重新连接时，将恢复持久性会话，AWS IoT 消息代理以每秒 10 条消息的最大速率将存储的消息发送到核心。永久会话的默认到期时间为 1 小时，从消息代理检测到核心断开连接时开始算起。有关更多信息，请参阅AWS IoT 开发人员指南中的 MQTT 持久性会话。

AWS IoT Greengrass 使用后台处理程序系统组件（Lambda GGCloudSpooler 函数）创建以 AWS IoT 作为源的订阅。您可以使用以下 GGCloudSpooler 环境变量来配置持久性会话。

- GG_CONFIG_SUBSCRIPTION_QUALITY。以 AWS IoT 为源的订阅的质量。有效值有：
  - AtMostOnce (默认值)。禁用持久性会话。订阅使用 QoS 0。
  - AtLeastOncePersistent。启用持久性会话。将 CONNECT 消息中的 cleanSession 标志设置为 0，并以 QoS 1 进行订阅。

如果核心收到的消息是以 QoS 1 发布，则会保证消息送达 Greengrass 守护进程的内存中工作队列。在将消息添加到队列后，核心会确认消息。从队列到本地目标（例如 Greengrass Lambda 函数、连接器或设备）的后续通信将以 QoS 0 发送。AWS IoT Greengrass 不能保证一定交付给本地目标。

> **Note**
>
> 您可以使用 C maxWorkItemount 配置属性来控制工作项队列的大小。例如，如果您的工作负载需要大量的 MQTT 流量，则可以增加队列大小。

启用持久会话后，核心至少会打开一个额外的连接，用于与 AWS IoT 交换 MQTT 消息。有关更多信息，请参阅 the section called “与 AWS IoT 的 MQTT 连接的客户端 ID”。

配置 MQTT 持久性会话

您可以配置 AWS IoT Greengrass 以使用与 AWS IoT Core 的持久性会话。为此，您需要部署一个函数定义版本，其中 `GGCloudSpooler` 函数将订阅质量设置为 `AtLeastOncePersistent`。此设置适用于具有 AWS IoT Core (`cloud`) 作为源的所有订阅。您可以使用 AWS IoT Greengrass API 来配置持久会话。您无法在控制台中执行该操作。

以下过程使用 `create-function-definition-version` CLI 命令将后台处理程序配置为使用持久会话。在该过程中，我们假定您要更新现有组的最新组版本的配置。

1. 获取目标 Greengrass 组和组版本的 ID。此过程假定这是最新的组和组版本。以下查询将返回最近创建的组。

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

   或者，您也可以按名称查询。系统不要求组名称是唯一的，所以可能会返回多个组。

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

> **Note**
>
> 您还可以在 AWS IoT 控制台中找到这些值。组 ID 显示在组的设置页面上。组版本 ID 显示在组的部署选项卡上。

2. 从输出中的目标组复制 `Id` 和 `LatestVersion` 值。
3. 获取最新的组版本。

   - 将 *`group-id`* 替换为复制的 `Id`。
   - 将 *`latest-group-version-id`* 替换为复制的 `LatestVersion`。

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

4. 从输出中的 `Definition` 对象复制 `CoreDefinitionVersionArn` 以及所有其他组组件（`FunctionDefinitionVersionArn` 除外）的 ARN。在创建新的组版本时，将使用这些值。
5. 在输出的 `FunctionDefinitionVersionArn` 中，复制函数定义的 ID。该 ID 是 ARN 中的 `functions` 段后面的 GUID，如以下示例所示。

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

> **Note**
>
> 或者，您可以通过运行 `create-function-definition` 命令以创建一个函数定义，然后从输出中复制该 ID。

6. 在函数定义中添加一个函数定义版本。

   - *`function-definition-id`*替换`Id`为你为函数定义复制的。
   - *`arbitrary-function-id`*替换为函数的名称，例如**`spooler-function`**。
   - 将要在该版本中包含的任何 Lambda 函数添加到 `functions` 数组中。您可以使用 `get-function-definition-version` 命令从现有的函数定义版本中获取 Greengrass Lambda 函数。

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

> **Note**
>
> 如果您之前设置了 GG_CONFIG_STORAGE_TYPE 或 GG_CONFIG_MAX_SIZE_BYTES 环境变量来定义存储设置，请将其包含在该函数实例中。

7. 从输出中复制函数定义版本的 Arn。
8. 创建一个包含系统 Lambda 函数的组版本。

   - 将 *group-id* 替换为组的 Id。
   - *core-definition-version-arn*替换为您从最新群组版本中复制的版本。CoreDefinitionVersionArn
   - *function-definition-version-arn*替换为你Arn为新函数定义版本复制的函数。
   - 替换从最新组版本中复制的其他组组件（例如 SubscriptionDefinitionVersionArn 或 DeviceDefinitionVersionArn）的 ARN。
   - 删除任何未使用的参数。例如，如果组版本不包含任何资源，请删除 --resource-definition-version-arn。

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

9. 从输出中复制 Version。这是新组版本的 ID。

10. 用新组版本替换组。

    - 使用为组复制的 *##* group-idId。
    - *group-version-id*替换为你Version为新群组版本复制的版本。

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

11. （可选）增加核心配置文件中的 C maxWorkItemount 属性。这可以帮助核心处理增加的 MQTT 流量以及与本地目标的通信。

要使用这些配置更改来更新核心，您需要使用 AWS IoT Greengrass API 创建新的函数定义版本，其中包含具有更新配置的 GGCloudSpooler 函数。接下来，将函数定义版本添加到新的组版本（以及其他组组件）中，然后部署该组版本。如果要还原默认配置，您可以创建不包含 GGCloudSpooler 函数的函数定义版本。

控制台中不显示这个系统 Lambda 函数。不过，在将函数添加到最新组版本后，它将包含在从控制台指定的部署中（除非您使用 API 替换或删除该函数）。

与 AWS IoT 的 MQTT 连接的客户端 ID

此功能适用于 AWS IoT Greengrass Core v1.8 及更高版本。

Greengrass 核心与 AWS IoT Core 建立 MQTT 连接，以便执行影子同步和证书管理等操作。对于这些连接，核心会根据核心事物名称生成可预测的客户端 ID。可预测的客户端 ID 可用于监控、审计和定价功能，包括 AWS IoT Device Defender 和 AWS IoT 生命周期事件。您还可以围绕可预测的客户端 ID 创建逻辑（例如，基于证书属性的订阅策略模板）。

GGC v1.9 and later

两个 Greengrass 系统组件可以与 AWS IoT Core 建立 MQTT 连接。这些组件使用以下模式为连接生成客户端 ID。

操作	客户端 ID 模式
部署	*core-thing-name*

操作	客户端 ID 模式
	例如：`MyCoreThing` 将该客户端 ID 用于连接、断开连接、订阅和取消订阅生命周期事件通知。
订阅	*`core-thing-name`* `-c`*`n`* 例如：`MyCoreThing-c01` *`n`* 是一个从 00 开始的整数，每出现一个新连接，这个整数就会递增，最大值为 250。连接的数量由组中与 AWS IoT Core 同步其影子状态的设备的数量（每组最大数量为 2,500）以及组中使用 `cloud` 作为其源的订阅的数量（每组最大数量为 10,000）来决定。 后台处理程序系统组件与 AWS IoT Core 连接以便与云源或目标交换订阅消息。后台处理程序还充当在 AWS IoT Core 与本地影子服务和设备证书管理器之间交换消息的代理。

要计算每组的 MQTT 连接数，请使用以下公式：

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

其中，

- 部署代理的连接数 = 1。
- 订阅的连接数 = `(2 subscriptions for supporting certificate generation + number of MQTT topics in AWS IoT Core + number of device shadows synced) / 50`。
  - 其中，`50` = 每个连接 AWS IoT Core 可以支持的最大订阅数。

> **Note**
>
> 如果您通过 AWS IoT Core 为订阅启用持久会话，则核心至少会打开一个额外的连接以在持久会话中使用。系统组件不支持持久会话，因此它们无法共享该连接。

为了减少 MQTT 连接的数量并帮助降低成本，您可以使用本地 Lambda 函数在边缘聚合数据。然后，您将聚合数据发送到 AWS Cloud。因此，您可以减少在 AWS IoT Core 中使用的 MQTT 主题数量。有关更多信息，请参阅AWS IoT Greengrass 定价。

GGC v1.8

几个 Greengrass 系统组件可以与 AWS IoT Core 建立 MQTT 连接。这些组件使用以下模式为连接生成客户端 ID。

操作	客户端 ID 模式
部署	*core-thing-name* 例如：MyCoreThing 将该客户端 ID 用于连接、断开连接、订阅和取消订阅生命周期事件通知。
MQTT 与 AWS IoT Core 的消息交换	*core-thing-name* -spr 例如：MyCoreThing-spr
影子同步	*core-thing-name* -s*nn* 例如：MyCoreThing-s01 *nn* 是一个整数，从 00 开始，随着每个新连接的增量递增，最大值为 03。连接数由与 AWS IoT Core 同步影子状态的设备数量（每个组最多 200 个设备）确定（每个连接最大 50 个订阅）。
设备证书管理	*core-thing-name* -dcm 例如：MyCoreThing-dcm

> **Note**
>
> 如果同时连接中使用了重复客户端 ID，则可能会导致在连接和断开连接之间无限循环。如果另一个设备硬编码为使用核心设备名称作为连接中的客户端 ID，则可能发生这种情况。有关更多信息，请参阅此问题排查步骤。

Greengrass 设备还与 AWS IoT Device Management 的实例集索引服务完全集成。这样您便可以根据云中的设备属性、影子状态和连接状态对设备进行索引和搜索。例如，Greengrass 设备至少建立一个使用事物名称作为客户端 ID 的连接，因此，您可以使用设备连接索引来发现哪些 Greengrass 设备当前已与 AWS IoT Core 连接或断开连接。有关更多信息，请参阅AWS IoT FleetWise 开发人员指南中的队列。

为本地消息收发配置 MQTT 端口

此功能需要 AWS IoT Greengrass Core v1.10 或更高版本。

Greengrass 核心充当本地 Lambda 函数、连接器和客户端设备之间的 MQTT 消息传递的本地消息代理。默认情况下，核心使用端口 8883 来传输本地网络上的 MQTT 流量。您可能需要更改端口，以避免与在端口 8883 上运行的其他软件发生冲突。

配置核心用于传输本地 MQTT 流量的端口号

1. 运行以下命令以停止 Greengrass 进程守护程序：

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

2. 打开 *greengrass-root*/config/config.json 以作为 su 用户进行编辑。

3. 在 coreThing 对象中，添加 ggMqttPort 属性并将值设置为要使用的端口号。有效值为 1024 到 65535。以下示例将端口号设置为 9000。

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

```
        "ggMqttPort" : 9000,
        "keepAlive" : 600
    },
    ...
}
```

4. 启动进程守护程序。

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

5. 如果为核心启用了自动 IP 检测，则配置已完成。

   如果未启用自动 IP 检测，则必须更新核心的连接信息。这允许客户端设备在发现操作期间接收正确的端口号，以获取核心连接信息。您可以使用 AWS IoT 控制台或 AWS IoT Greengrass API 来更新核心连接信息。在此过程中，您只需要更新端口号。核心的本地 IP 地址保持不变。

   更新核心的连接信息（控制台）

   1. 在组配置页面上，选择 Greengrass 核心。
   2. 在核心详细信息页面上，选择 MQTT 代理端点选项卡。
   3. 选择管理端点选项卡，然后选择添加端点。
   4. 输入您当前的本地 IP 地址和新的端口号。以下示例设置 IP 地址 192.168.1.8 的端口号 9000。
   5. 移除过时的端点，然后选择更新

   更新核心的连接信息 (API)

   - 使用 UpdateConnectivityInfo 操作。以下示例使用 AWS CLI 中的 update-connectivity-info 为 IP 地址 192.168.1.8 设置端口号 9000。

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

> **Note**
>
> 还可以配置核心用于与 AWS IoT Core 进行 MQTT 消息传递的端口。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”。

在与 AWS Cloud的 MQTT 连接中的发布、订阅和取消订阅操作的超时

AWS IoT Greengrass v1.10.2 或更高版本中提供了此功能。

您可以配置允许 Greengrass 核心在与 AWS IoT Core 的 MQTT 连接中完成发布、订阅或取消订阅操作的时间（以秒为单位）。如果因带宽限制或高延迟导致操作超时，您可能需要调整此设置。要在 config.json 文件中配置此设置，请在 `coreThing` 对象中添加或更改 `mqttOperationTimeout` 属性。例如：

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

默认超时时间为 5 秒。最小超时值为 5 秒。

激活自动 IP 检测

您可以配置 AWS IoT Greengrass 以使 Greengrass 组中的客户端设备能够自动发现 Greengrass 核心。启用后，内核将监视其 IP 地址的变化。如果地址发生变化，核心将发布更新的地址列表。与核心位于同一 Greengrass 组中的客户端设备可以使用这些地址。

> **Note**
>
> 客户端设备的 AWS IoT 策略必须授予 `greengrass:Discover` 允许设备检索核心的连接信息的 权限。有关策略语句的更多信息，请参阅the section called “发现授权”。

要从 AWS IoT Greengrass 控制台启用此功能，请在首次部署 Greengrass 组时选择自动检测。您也可以在组配置页面上启用或禁用此功能，方法是选择 Lambda 函数选项卡，然后选择 IP 检测器。如果选择自动检测和覆盖 MQTT 代理端点，则会启用自动 IP 检测。

要使用 AWS IoT Greengrass API 管理自动发现，您必须配置 `IPDetector` 系统 Lambda 函数。以下过程说明如何使用 create-function-definition-versionCLI 命令配置 Greengrass 内核的自动发现。

1. 获取目标 Greengrass 组和组版本的 ID。此过程假定这是最新的组和组版本。以下查询将返回最近创建的组。

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

   或者，您也可以按名称查询。系统不要求组名称是唯一的，所以可能会返回多个组。

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

   > **Note**
   >
   > 您还可以在 AWS IoT 控制台中找到这些值。组 ID 显示在组的设置页面上。组版本 ID 显示在组的部署选项卡上。

2. 从输出中的目标组复制 `Id` 和 `LatestVersion` 值。
3. 获取最新的组版本。

   - 将 *group-id* 替换为复制的 `Id`。
   - 将 *latest-group-version-id* 替换为复制的 `LatestVersion`。

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

4. 从输出中的 `Definition` 对象复制 `CoreDefinitionVersionArn` 以及所有其他组组件（`FunctionDefinitionVersionArn` 除外）的 ARN。在创建新的组版本时，将使用这些值。
5. 从输出中的 `FunctionDefinitionVersionArn`，复制函数定义的 ID 和函数定义版本：

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

> **Note**
>
> 您可以选择运行 create-function-definition 命令以创建一个函数定义，然后从输出中复制该 ID。

6. 使用 get-function-definition-version 命令可获取当前定义状态。使用*function-definition-id*您复制的作为函数定义。例如，*4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*。

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

记录下列出的函数配置。在创建新的函数定义版本时，您需要包括这些函数配置，以防止丢失当前的定义设置。

7. 在函数定义中添加一个函数定义版本。

   - *function-definition-id*替换Id为你为函数定义复制的。例如，*4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*。
   - *arbitrary-function-id*替换为函数的名称，例如**auto-detection-function**。
   - 将要包含在此版本中的所有 Lambda 函数都添加到 functions 数组中，例如上一步中列出的任何函数。

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

8. 从输出中复制函数定义版本的 Arn。
9. 创建一个包含系统 Lambda 函数的组版本。

- 将 *group-id* 替换为组的 Id。
- *core-definition-version-arn*替换为您从最新群组版本中复制的版本。CoreDefinitionVersionArn
- *function-definition-version-arn*替换为你Arn为新函数定义版本复制的函数。
- 替换从最新组版本中复制的其他组组件（例如 SubscriptionDefinitionVersionArn 或 DeviceDefinitionVersionArn）的 ARN。
- 删除任何未使用的参数。例如，如果组版本不包含任何资源，请删除 --resource-definition-version-arn。

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

10. 从输出中复制 Version。这是新组版本的 ID。
11. 用新组版本替换组。

    - 使用为组复制的 ## group-idId。
    - *group-version-id*替换为你Version为新群组版本复制的版本。

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

如果要手动输入 Greengrass 核心的 IP 地址，您可以使用其他不包含 IPDetector 函数的函数定义来完成此教程。这将阻止检测函数定位并自动输入您的 Greengrass 核心 IP 地址。

控制台中不显示这个系统 Lambda 函数。在将函数添加到最新组版本后，除非您使用 API 替换或删除该函数，否则它将包含在从控制台指定的部署中。

配置初始化系统以启动 Greengrass 守护程序

最佳做法是，将初始化系统设置为在引导期间启动 Greengrass 守护程序，尤其是在管理大量设备时。

> **Note**
>
> 如果您已经使用 `apt` 安装了 AWS IoT Greengrass Core 软件，则可以使用 systemd 脚本启用“引导时启动”。有关更多信息，请参阅 the section called “使用 systemd 脚本管理 Greengrass 守护程序生命周期”。

具有不同类型的初始化系统（如 initd、systemd 和 SystemV），并且它们使用类似的配置参数。以下示例是用于 systemd 的服务文件。`Type` 参数设置为 `forking`，因为 greengrassd（用于启动 Greengrass）为 Greengrass 守护程序进程创建分支，而 `Restart` 参数设置为 `on-failure` 以指示 systemd 在 Greengrass 进入故障状态时重新启动 Greengrass。

> **Note**
>
> 要查看您的设备是否使用 systemd，请按照模块 1 中所述运行 `check_ggc_dependencies` 脚本。然后，要使用 systemd，请确保将 `useSystemd``config.json` 中的 参数设置为 `yes`。

```
[Unit]
Description=Greengrass Daemon

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

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

另请参阅

- 什么是 AWS IoT Greengrass？
- the section called “支持的平台和要求”
- 入门 AWS IoT Greengrass
- the section called “组对象模型概述”
- the section called “硬件安全性集成”

AWS IoT Greengrass Version 1 维护策略

使用此 AWS IoT Greengrass V1 维护策略了解 AWS IoT Greengrass V1 服务和 AWS IoT Greengrass Core 软件 v1.x 的不同维护和更新级别。

主题

AWS IoT Greengrass 版本控制方案

AWS IoT Greengrass 对 AWS IoT Greengrass Core 软件使用语义版本控制。语义版本遵循 major.minor.patch 编号系统。如果功能和 API 更改与之前的主要版本不兼容，则主版本号将递增。如果版本新增了向后兼容功能，则次要版本号将递增。如果是安全性补丁或错误修复，则补丁版本号将递增。自 AWS IoT Greengrass 的第一个主要版本 v1.0.0 发布以来，已经发布了 11 个次要版本的 AWS IoT Greengrass Core 软件 v1.x，目前最新版本是 v1.11.6。我们建议您将 AWS IoT Greengrass Core 软件更新为最新可用版本，以利用新功能、增强功能和错误修复。

2020 年 12 月，AWS IoT Greengrass 发布了其第一个主要版本更新。该更新包括 AWS IoT Greengrass Core 软件的 AWS IoT Greengrass V2 服务和版本 2.0.3。对于新应用程序，我们强烈建议您使用 AWS IoT Greengrass Version 2 和 AWS IoT Greengrass Core 软件 v2.x。版本 2 包含新功能，包括 V1 的所有关键功能，并支持其他平台，还支持持续部署到大量设备。有关更多信息，请参阅什么是 AWS IoT Greengrass V2？。

AWS IoT Greengrass Core 软件主要版本的生命周期阶段

AWS IoT Greengrass Core 软件的每个主要版本都有以下三个连续的生命周期阶段。每个生命周期阶段在初始发布日期之后的一段时间内提供不同级别的维护。

- 发布阶段 — AWS IoT Greengrass 可能会发布以下更新：
  - 次要版本更新，为现有功能提供新功能或增强功能
  - 补丁版本更新，提供安全性补丁和错误修复
- 维护阶段 — AWS IoT Greengrass 可能会发布补丁版本更新，提供安全性补丁和错误修复。AWS IoT Greengrass在维护阶段不会发布新功能，也不会发布现有功能的增强功能。
- 生命周期延长阶段 — AWS IoT Greengrass 不会发布可提供新功能、增强现有功能、安全补丁或错误修复的更新。但是，AWS Cloud 端点和 API 操作将保持可用状态，并根据 AWS IoT Greengrass 服务等级协议运行。运行 AWS IoT Greengrass Core 软件 v1.x 的设备可以继续连接到 AWS Cloud 并正常运行。

AWS IoT Greengrass 主要版本的延长生命周期结束后，AWS Cloud 端点和 API 操作将被弃用且不再可用。运行 AWS IoT Greengrass Core 软件 v1.x 的设备将无法连接到 AWS Cloud 服务，因此无法运行。

AWS IoT Greengrass Core 软件的维护政策

AWS IoT Greengrass Core 软件 v1.x 于 2023 年 6 月 30 日进入延长使用寿命阶段。在此日期之后，AWS IoT Greengrass Core 软件 v1.x 将一直处于延长使用寿命阶段，直至另行通知。

AWS IoT Greengrass Core 软件 v2.x 目前处于发布阶段，并将一直处于发布阶段，直至另行通知。AWS IoT Greengrass 将继续为 AWS IoT Greengrass Core 软件 v2.x 增加新功能和增强功能。例如，AWS IoT Greengrass 在 AWS IoT Greengrass Core 软件的 2.5.0 版本中发布了 Windows 支持。AWS IoT Greengrass 会在发布之日起至少 1 年内，发布适用于 AWS IoT Greengrass Core v2.x 所有次要版本的安全补丁和错误修复。有关更多信息，请参阅 AWS IoT Greengrass V2 中的新增功能。

维护阶段时间表

2023 年 6 月 30 日，AWS IoT Greengrass Core 软件 v1.11.x 的维护阶段结束。2022 年 3 月 31 日，AWS IoT Greengrass Core 软件 v1.10.x 的维护阶段结束。某些 AWS IoT Greengrass Core 软件 v1.x 版本工件和功能的维护阶段将在这些日期之前结束。有关更多信息，请参阅维护结束时间表。

如果您有 AWS Support 计划，则 AWS IoT Greengrass Core 软件 v1.x 的维护阶段不会影响您的 AWS Support 计划。即使在维护阶段结束后，您也可以继续提交 AWS Support 票证。如果您有任何疑问或疑虑，请联系您的 AWS Support 联系人，或者使用 AWS IoT Greengrass 标签在 AWS re:Post 提出问题。

弃用时间表

目前，没有计划停止支持 AWS IoT Greengrass Core 软件 v1.x。AWS IoT Greengrass V1 端点和 API 操作将保持可用，直至另行通知。AWS IoT Greengrass Core 软件 v1.11.6 已于 2023 年 6 月 30 日进入延长使用寿命阶段。在此阶段，运行 AWS IoT Greengrass Core 软件 v1.x 的设备可继续连接到 AWS IoT Greengrass V1 服务以运行，直至另行通知。

如果 AWS IoT Greengrass V1 以后要停止提供支持，则 AWS IoT Greengrass 将停止支持之前提前 12 个月发出通知。这将帮助您计划将自己的应用程序更新为使用 AWS IoT Greengrass V2 和 AWS IoT Greengrass Core 软件 v2.x。有关如何将应用程序更新到 V2 的更多信息，请参阅从 AWS IoT Greengrass V1 更新到 V2。

Greengrass 核心设备上 AWS Lambda 功能的支持政策

AWS IoT Greengrass 使您能够在 IoT 设备上运行 AWS Lambda 函数。AWS Lambda 提供了支持策略和时间表，用于确定 AWS IoT Greengrass 中对 Lambda 运行时的支持。在 Lambda 运行时达到支持阶段结束后，AWS IoT Greengrass 也将终止对该运行时的支持。有关更多信息，请参阅 AWS Lambda 开发人员指南中的运行时支持策略。

当 Lambda 运行时终止支持时，您无法创建或更新使用该运行时的 Lambda 函数。但是，您可以继续将这些 Lambda 函数部署到 Greengrass 核心设备，也可以继续调用已部署的 Lambda 函数。本政策也适用于 AWS IoT Greengrass V2。

适用于 AWS IoT Greengrass V1 的 AWS IoT Device Tester 的支持策略

AWS IoT Device Tester (IDT) for AWS IoT Greengrass V1 使您能够验证您的 AWS IoT Greengrass 设备并确定设备是否有资格被包含在 AWS Partner 设备目录中。自 2022 年 4 月 4 日起，AWS IoT Device Tester (IDT) for AWS IoT Greengrass V1 不再生成带签名的资格报告。您无法再通过 AWS 设备资格认证计划来确定新的 AWS IoT Greengrass V1 设备是否有资格被列入 AWS Partner 设备目录中。虽然您无法获得 Greengrass V1 设备的资格认证，但可以继续使用 IDT for AWS IoT Greengrass V1 来测试您的 Greengrass V1 设备。我们建议您使用适用于 AWS IoT Greengrass V2 的 IDT 来进行资格认证并在 AWS Partner 设备目录中列示 Greengrass 设备。有关更多信息，请参阅适用于 AWS IoT Greengrass V1 的 AWS IoT Device Tester 的支持策略。

维护结束时间表

下表列出了 AWS IoT Greengrass Core v1.x 工件和功能的维护结束日期。如果您对维护计划或政策有疑问，请联系 AWS 支持部门。

构件或功能	维护结束日期
Greengrass APT 存储库安装	2022 年 2 月 11 日
ML 图像分类连接器	2022 年 3 月 31 日
ML 对象检测连接器	2022 年 3 月 31 日
ML 反馈连接器	2022 年 3 月 31 日
AWS IoT Analytics 连接器	2022 年 3 月 31 日
Twilio 通知连接器	2022 年 3 月 31 日
Splunk 集成连接器	2022 年 3 月 31 日
串行流连接器	2022 年 3 月 31 日
ServiceNow MetricBase 集成连接器	2022 年 3 月 31 日
Raspberry Pi GPIO 连接器	2022 年 3 月 31 日
AWS IoT Greengrass Core 软件 v1.10.x	2022 年 3 月 31 日
AWS IoT Greengrass Core 软件 v1.x Docker 映像	2022 年 6 月 30 日
AWS IoT Greengrass Core 软件 v1.11.x	2023 年 6 月 30 日
AWS IoT Greengrass Core 软件 v1.11.x Snap	2023 年 12 月 31 日

AWS IoT Greengrass Core 软件 v1.x Docker 映像的维护已结束

2022 年 6 月 30 日，AWS IoT Greengrass 结束了对发布到 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 的 AWS IoT Greengrass Core 软件 v1.x Docker 映像的维护。您可以

继续从 Amazon ECR 和 Docker Hub 下载这些 Docker 映像，直至 2023 年 6 月 30（即维护结束 1 年后）为止。但在 2022 年 6 月 30 日维护结束后，AWS IoT Greengrass Core 软件 v1.x Docker 映像不再收到安全补丁或错误修复。如果运行的生产工作负载依赖于这些 Docker 映像，我们建议您使用提供的 Dockerfiles 构建自己的 Docker 映像。AWS IoT Greengrass有关更多信息，请参阅AWS IoT Greengrass Docker 软件。

AWS IoT Greengrass Core 软件 v1.x APT 存储库的维护已结束

2022 年 2 月 11 日，AWS IoT Greengrass 终止了对于从 APT 存储库安装 AWS IoT Greengrass Core 软件 v1.x 选项的维护。APT 存储库已于上述日期移除，因此您无法再使用 APT 存储库来更新 AWS IoT Greengrass Core 软件，也无法在新设备上安装 AWS IoT Greengrass Core 软件。在您已经添加了 AWS IoT Greengrass 存储库的设备上，必须从来源列表中移除存储库。我们建议您使用 tar 文件来更新 AWS IoT Greengrass Core 软件 v1.x。

AWS IoT Greengrass Core 软件 v1.11.x Snap 的维护已结束

AWS IoT Greengrass 将于 2023 年 12 月 31 日结束对于 AWS IoT Greengrass Core 软件版本 1.11.x Snap 的维护，这一消息已经发布在 snapcraft.io 上。当前运行 Snap 的设备将能够继续正常运转，直至另行通知。但在维护结束后，AWS IoT Greengrass Core Snap 将不会再收到安全补丁或错误修复。

入门 AWS IoT Greengrass

本入门教程包括几个模块，旨在向您展示 AWS IoT Greengrass 基础知识并帮助您开始使用 AWS IoT Greengrass。本教程涵盖如下基本概念：

- 配置 AWS IoT Greengrass 核心和组。
- 在边缘运行 AWS Lambda 函数的部署过程。
- 将称为客户端设备的设备连接到 AWS IoT 核 AWS IoT Greengrass 心。
- 创建订阅以允许本地 Lambda 函数、客户端设备和之间的 MQTT 通信。 AWS IoT

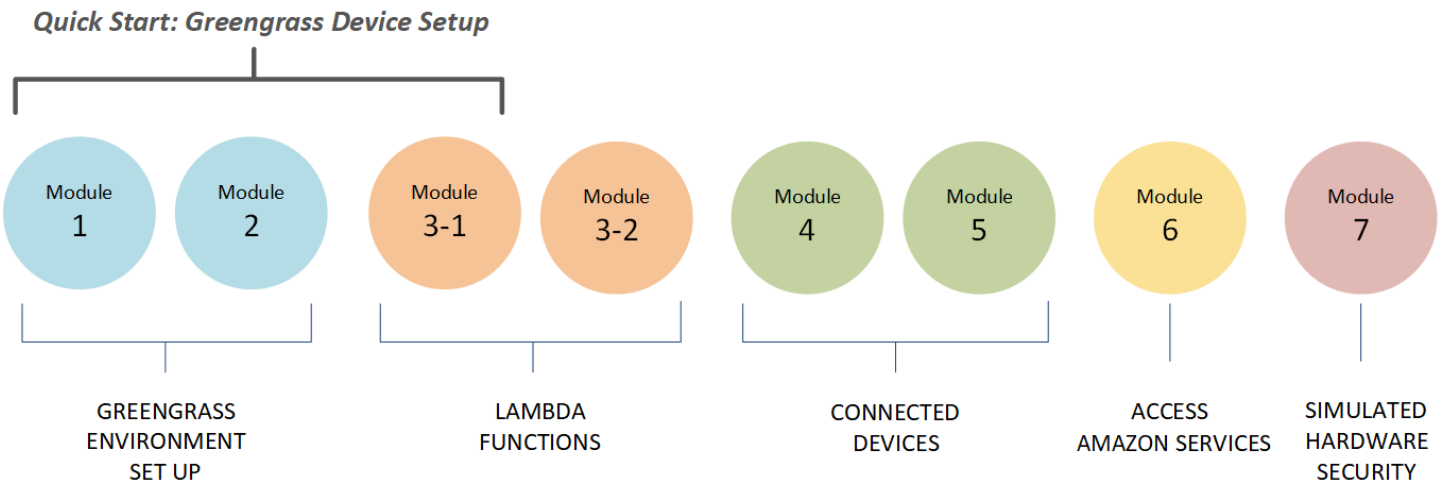
选择如何开始使用 AWS IoT Greengrass

您可以选择如何使用本教程设置您的核心设备：

- 在核心设备上运行 Greengrass 设备安装程序，只需几分钟即可完成从 AWS IoT Greengrass 安装依赖项到测试 Hello World Lambda 函数的过程。此脚本重现模块 1 到模块 3-1 中的步骤。

  - 或 -

- 逐步执行模块 1 到模块 3-1 中的步骤，以更仔细地检查 Greengrass 的要求和流程。这些步骤包括设置您的核心设备、创建和配置包含 Hello World Lambda 函数的 Greengrass 群组，以及部署 Greengrass 群组。通常，这需要一两个小时才能完成。



Quick Start（快速入门）

Greengrass 设备安装程序可配置您的核心设备和 Greengrass 资源。脚本：

- 安装 AWS IoT Greengrass 依赖关系。
- 下载根 CA 证书及核心设备证书和密钥。
- 在您的设备上下载、安装和配置 AWS IoT Greengrass Core 软件。
- 在核心设备上启动 Greengrass 守护进程。
- 如果需要，创建或更新 Greengrass 服务角色。
- 创建 Greengrass 组和 Greengrass 核心。
- （可选）创建 Hello World Lambda 函数、订阅和本地日志记录配置。
- （可选）部署 Greengrass 组。

模块 1 和模块 2

模块 1 和模块 2 描述了如何设置您的环境。（或者，使用 Greengrass 设备安装程序来为您运行这些模块。）

- 为 Greengrass 配置核心设备。
- 运行依赖项检查程序脚本。
- 创建 Greengrass 组和 Greengrass 核心。
- 从 tar.gz 文件下载并安装最新的 AWS IoT Greengrass 酷睿软件。
- 在核心设备上启动 Greengrass 守护进程。

> **Note**
>
> AWS IoT Greengrass 还提供了安装 AWS IoT Greengrass 核心软件的其他选项，包括在支持的 Debian 平台上`apt`安装。有关更多信息，请参阅 the section called “安装 AWS IoT Greengrass Core 软件”。

模块 3-1 和 3-2

模块 3-1 和模块 3-2 描述了如何使用本地 Lambda 函数。（或者，使用 Greengrass 设备安装程序来为您运行模块 3-1。）

- 在中创建 Hello World Lambda 函数。 AWS Lambda
- 将 Lambda 函数添加到您的 Greengrass 组。
- 创建允许在 Lambda 函数和之间进行 MQTT 通信的订阅。 AWS IoT
- 为 Greengrass 系统组件和 Lambda 函数配置本地日志记录。
- 部署程序包含 Lambda 函数和订阅的 Greengrass 组。
- 将来自本地 Lambda 函数的消息发送到。 AWS IoT
- 从中调用本地 Lambda 函数。 AWS IoT
- 测试按需和长时间运行的函数。

模块 4 和 5

模块 4 展示了客户端设备如何连接到核心并相互通信。

模块 5 展示了客户端设备如何使用阴影来控制状态。

- 注册和配置 AWS IoT 设备（由命令行终端表示）。
- 安装 AWS IoT Device SDK 适用于 Python 的。客户端设备使用它来发现 Greengrass 核心。
- 将客户端设备添加到您的 Greengrass 组。
- 创建允许 MQTT 通信的订阅。
- 部署程序包含客户端设备的 Greengrass 组。
- 测试 device-to-device 通信。
- 测试阴影状态更新。

模块 6

模块 6 展示了 Lambda 函数如何访问 AWS Cloud。

- 创建允许访问 Amazon DynamoDB 资源的 Greengrass 组角色。
- 将 Lambda 函数添加到 Greengrass 组。此函数使用适用于 Python 的 AWS 软件开发工具包与 DynamoDB 进行交互。
- 创建允许 MQTT 通信的订阅。
- 测试与 DynamoDB 的交互。

模块 7

模块 7 展示了如何配置模拟硬件安全模块 (HSM) 以便与 Greengrass 核心一起使用。

> **Important**
>
> 此高级模块仅用于实验和初始测试。它不适用于任何种类的生产用途。

- 安装和配置基于软件的 HSM 和私钥。
- 配置 Greengrass 核心以使用硬件安全性。
- 测试硬件安全性配置。

要求

要完成本教程，您需要：

- Mac、Windows PC 或类似 UNIX 的系统。
- 一个 AWS 账户。如果没有，请参阅the section called “创建一个 AWS 账户”。
- 使用支持的 AWS 区域 AWS IoT Greengrass。有关支持的区域列表 AWS IoT Greengrass，请参阅中的AWS 终端节点和配额AWS 一般参考。

> **Note**
>
> 记下你的 ， AWS 区域 并确保在本教程中始终如一地使用它。如果您在教程 AWS 区域 中切换，则在完成这些步骤时可能会遇到问题。

- 一台 Raspberry PI 4 Model B 或 Raspberry PI 3 Model B/B+（带有一张 8 GB microSD 卡）或一个 Amazon EC2 实例。由于理想情况下 AWS IoT Greengrass 应与物理硬件结合使用，因此，我们建议您使用 Raspberry Pi。

> **Note**
>
> 运行以下命令可获取 Raspberry Pi 的型号：
>
> ```
> cat /proc/cpuinfo
> ```
>
> 在列表底部附近，找到并记下 `Revision` 属性的值，然后查阅 Which Pi have I got? 表。例如，如果 `Revision` 的值为 `a02082`，查阅该表可知该 Pi 为 3 Model B。
> 请运行下面的命令，以确定您的 Raspberry Pi 的体系结构：
>
> ```
> uname -m
> ```
>
> 在本教程中，此结果应该大于或等于 `armv71`。

- 基本熟悉 Python。

尽管本教程旨在在 Raspberry Pi AWS IoT Greengrass 上运行，但 AWS IoT Greengrass 也支持其他平台。有关更多信息，请参阅 the section called “支持的平台和要求”。

创建一个 AWS 账户

如果您没有 AWS 账户，请按照以下步骤创建并激活 AWS 账户：

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 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。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。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，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后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。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。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. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

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《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 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。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

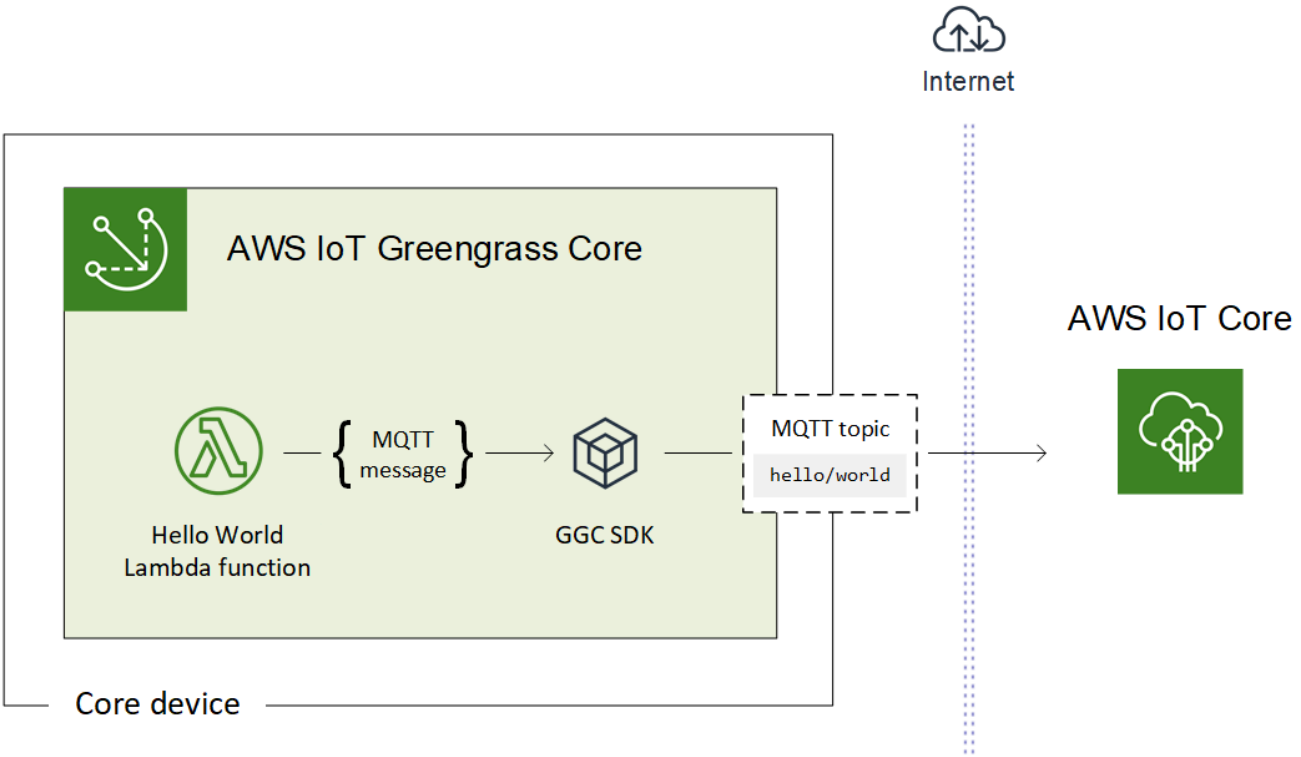
   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

> **Important**
>
> 在本教程中，我们假定您的 IAM 用户账户具有管理员访问权限。

快速入门：Greengrass 设备安装程序

Greengrass 设备安装程序是一个脚本，可在几分钟内安装好核心设备，让您快速开始使用 AWS IoT Greengrass。使用此脚本可以：

1. 配置您的设备并安装 AWS IoT Greengrass 核心软件。
2. 配置基于云的资源。
3. 使用 Hello World Lambda 函数有选择地部署 Greengrass 组，该函数会将 MQTT 消息从 AWS IoT Greengrass 核心发送到 AWS IoT。这将设置如下图所示的 Greengrass 环境。



要求

Greengrass 设备安装程序具有以下要求：

- 您的核心设备必须使用支持的平台。设备必须安装了适当的软件包管理器： `apt`、`yum` 或 `opkg`。
- 运行脚本的 Linux 用户必须具有以 `sudo` 身份运行的权限。
- 您必须提供 AWS 账户 凭证 有关更多信息，请参阅the section called “提供 AWS 账户 凭证”。

> **Note**
>
> Greengrass 设备安装程序会在设备上安装最新版本的 AWS IoT Greengrass 核心软件。安装 AWS IoT Greengrass 核心软件即表示您同意 Greengrass 核心软件许可协议。

运行 Greengrass 设备安装程序

您只需几个步骤即可运行 Greengrass 设备安装程序。提供 AWS 账户 证书后，脚本会预置您的 Greengrass 核心设备，并在几分钟内部署 Greengrass 组。在目标设备上的终端窗口中运行以下命令。

> **Note**
>
> 以下步骤介绍如何在交互模式下运行脚本，这会提示您输入或接受每个输入值。有关如何以无提示方式运行脚本的信息，请参阅 the section called “在无提示模式下运行 Greengrass 设备安装程序”。

1. 提供凭证。在此过程中，我们假定您提供临时安全凭证作为环境变量。

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

> **Note**
>
> 如果您在 Raspbian 或 OpenWrt 平台上运行 Greengrass 设备安装程序，请复制这些命令。重新启动设备后，必须再次提供它们。

2. 下载并启动脚本。您可以使用 `curl` 或 `wget` 下载脚本。

   `wget`:

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

   `curl`:

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

3. 在命令提示符下继续输入值。您可按 Enter 键使用默认值，也可键入自定义值，然后按 Enter。

   该脚本将如下所示的状态消息写入终端。

   ```
   ############## Greengrass Device Setup v1.0.0 ##############
   [GreengrassDeviceSetup] The Greengrass Device Setup bootstrap log is available at: /tmp/greengrass-device-setup-bootstrap-1575933831.log
   [GreengrassDeviceSetup] Using package management tool: yum...
   [GreengrassDeviceSetup] Using runtime: python3.7...
   [GreengrassDeviceSetup] Installing a dedicated pip for Greengrass Device Setup...
   [GreengrassDeviceSetup] Validating and installing required dependencies...
   [GreengrassDeviceSetup] The Greengrass Device Setup configuration is complete. Starting the Greengrass environment setup...
   [GreengrassDeviceSetup] Forwarding command-line parameters: bootstrap-greengrass-interactive

   [GreengrassDeviceSetup] Validating the device environment...
   [GreengrassDeviceSetup] Validation of the device environment is complete.

   [GreengrassDeviceSetup] Running the Greengrass environment setup...
   [GreengrassDeviceSetup] The Greengrass environment setup is complete.

   [GreengrassDeviceSetup] Configuring cloud-based Greengrass group management...
   [GreengrassDeviceSetup] The Greengrass group configuration is complete.

   [GreengrassDeviceSetup] Preparing the Greengrass core software...
   [GreengrassDeviceSetup] The Greengrass core software is running.

   [GreengrassDeviceSetup] Configuring the group deployment...
   [GreengrassDeviceSetup] The group deployment is complete.
   ```

4. 如果您的核心设备正在运行 Raspbian 或 OpenWrt，请在提示时重新启动设备，提供凭证，然后重新启动脚本。

a. 当提示重新启动设备时，请运行以下命令之一。

对于 Raspbian 平台：

```
sudo reboot
```

对于 OpenWrt 平台：

```
reboot
```

b. 设备重新启动后，打开终端并提供您的凭证作为环境变量。

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

c. 重新启动脚本。

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

d. 当提示是使用上一个会话中的输入值还是启动新安装时，输入 `yes` 以重复使用输入值。

> **Note**
>
> 在需要重新启动的平台上，上一个会话中的输入值（凭证除外）将临时存储在 `GreengrassDeviceSetup.config.info` 文件中。

安装完成后，终端会显示如下所示的状态消息。

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

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


Setup information:

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

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

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

5. 检查该脚本使用您提供的输入值配置的新 Greengrass 组。

   a. 在您的计算机上登录 AWS Management Console 并打开 AWS IoT 控制台。

   > **Note**
   >
   > 确保在控制台中选择的 AWS 区域 与您用于配置 Greengrass 环境的区域相同。默认情况下，区域为美国西部（俄勒冈州）。

   b. 在导航窗格中，展开 Greengrass 设备，然后选择组 (V1) 以查找新创建的组。

6. 如果您包含了 Hello World Lambda 函数，则 Greengrass 设备安装程序会将 Greengrass 组部署到您的核心设备。要测试 Lambda 函数，或者要了解如何从组中移除 Lambda 函数的信息，请继续学习入门教程模块 3-1 中的 the section called “验证 Lambda 函数是否在核心设备上运行”。

   > **Note**
   >
   > 确保在控制台中选择的 AWS 区域 与您用于配置 Greengrass 环境的区域相同。默认情况下，区域为美国西部（俄勒冈州）。

   如果您没有包含 Hello World Lambda 函数，可以创建自己的 Lambda 函数或尝试其他 Greengrass 功能。例如，您可以将 Docker 应用程序部署连接器添加到组，并使用它将 Docker 容器部署到您的核心设备。

排查问题

您可以使用以下信息帮助解决 AWS IoT Greengrass 设备安装的问题。

错误：找不到 Python (python3.7)。正在尝试安装...

解决方案：在操作 Amazon EC2 实例时，您可能会看到此错误。如果 `/usr/bin/python3.7` 文件夹中没有安装 Python，则会发生此错误。要解决这个错误，请在安装 Python 后将其移到正确的目录中：

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

其他故障排除

要解决AWS IoT Greengrass 设备安装程序时遇到的其他问题，您可以在日志文件中查找调试信息：

- 有关 Greengrass 设备安装程序配置的问题，请查阅 `/tmp/greengrass-device-setup-bootstrap-`*`epoch-timestamp`*`.log` 文件。
- 有关 Greengrass 组或核心环境安装的问题，请查阅与 `gg-device-setup-latest.sh` 目录相同或指定位置中的 `GreengrassDeviceSetup-`*`date`*`-`*`time`*`.log` 文件。

有关问题排查的更多帮助，请参阅 排查问题 或查看 AWS re:Post 的 AWS IoT Greengrass 标签。

Greengrass 设备安装程序配置选项

您可以配置 Greengrass 设备安装程序以访问您的 AWS 资源并设置您的 Greengrass 环境。

提供 AWS 账户 凭证

Greengrass 设备安装程序使用您的 AWS 账户 凭证访问您的 AWS 资源。它支持 IAM 用户的长期凭证或 IAM 角色的临时安全凭证。

首先，获取您的凭证。

- 要使用长期凭证，请为 IAM 用户提供访问密钥 ID 和秘密访问密钥。有关创建访问长期凭证密钥的信息，请参阅 IAM 用户指南 中的管理 IAM 用户的访问密钥。

- 要使用临时安全凭证（推荐），请提供假定的 IAM 角色的访问密钥 ID、秘密访问密钥和会话令牌。有关从 AWS STS `assume-role` 命令提取临时安全凭证的信息，请参阅 IAM 用户指南中的将临时安全凭证用于 AWS CLI。

> **Note**
>
> 在本教程中，我们假定 IAM 用户或 IAM 角色具有管理员访问权限。

然后，通过以下两种方式之一为 Greengrass 设备设置提供您的凭证：

- 作为环境变量。在启动脚本之前设置 `AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY` 和 `AWS_SESSION_TOKEN`（如果需要）环境变量，如the section called “运行 Greengrass 设备安装程序”中的步骤 1 所示。

- 作为输入值。启动脚本后，直接在终端中输入访问密钥 ID、秘密访问密钥和会话令牌（如果需要）值。

Greengrass 设备安装程序不会保存或存储您的凭证。

提供输入值

在交互模式下，Greengrass 设备安装程序会提示您提供输入值。您可按 Enter 键使用默认值，也可键入自定义值，然后按 Enter。在无提示模式下，您可以在启动脚本后提供输入值。

输入值

AWS访问密钥 ID

长期或临时安全凭证中的访问密钥 ID。仅当您不提供凭证作为环境变量时，才将此选项指定为输入值。有关更多信息，请参阅the section called “提供 AWS 账户 凭证”。

无提示模式的选项名称：`--aws-access-key-id`

AWS私有访问密钥

长期或临时安全凭证中的秘密访问密钥。仅当您不提供凭证作为环境变量时，才将此选项指定为输入值。有关更多信息，请参阅the section called “提供 AWS 账户 凭证”。

无提示模式的选项名称：`--aws-secret-access-key`

AWS会话令牌

临时安全凭证中的会话令牌。仅当您不提供凭证作为环境变量时，才将此选项指定为输入值。有关更多信息，请参阅the section called “提供 AWS 账户 凭证”。

无提示模式的选项名称：`--aws-session-token`

AWS 区域

您想要创建 Greengrass 组的 AWS 区域。有关支持的 AWS 区域，请参见 Amazon Web Services 一般参考 中的 AWS IoT Greengrass。

默认值：`us-west-2`

无提示模式的选项名称：`--region`

Group name

Greengrass 组的名称。

默认值：`GreengrassDeviceSetup_Group_`*`guid`*

无提示模式的选项名称：`--group-name`

核心名称

Greengrass 核心的名称。核心是运行 AWS IoT Greengrass 核心软件的 AWS IoT 设备（事物）。核心被添加到 AWS IoT 注册表和 Greengrass 组。如果您提供了名称，则该名称在 AWS 账户和 AWS 区域中必须是唯一的。

默认值：`GreengrassDeviceSetup_Core_`*`guid`*

无提示模式的选项名称：`--core-name`

AWS IoT Greengrass 核心软件安装路径

设备文件系统中您想要在安装 AWS IoT Greengrass 核心软件的位置。

默认值：`/`

无提示模式的选项名称：`--ggc-root-path`

Hello World Lambda 函数

指示是否在 Greengrass 组中包含 Hello World Lambda 函数。该函数每五秒钟向 `hello/world` 主题发布一条 MQTT 消息。

该脚本在 AWS Lambda 中创建并发布此用户定义的 Lambda 函数，并将其添加到您的 Greengrass 组中。该脚本还会在组中创建一个订阅，该订阅允许函数向 AWS IoT 发送 MQTT 消息。

> **Note**
>
> 这是一个 Python 3.7 Lambda 函数。如果设备上未安装 Python 3.7 且脚本无法安装，则脚本会在终端中输出一条错误消息。要将该 Lambda 函数包含在组中，必须手动安装 Python 3.7 并重新启动脚本。要创建不包含该 Lambda 函数的 Greengrass 组，请重新启动脚本，并在提示是否包含该函数时输入 no。

默认值：no

无提示模式的选项名称：`--hello-world-lambda` - 此选项没有值。如果要创建函数，请将其包含在命令中。

部署超时

Greengrass 设备安装程序停止检查 Greengrass 组部署状态之前的秒数。仅当组包含 Hello World Lambda 函数时才使用此功能。否则，不会部署该组。

部署时间取决于您的网速。如果网速较慢，您可以增加此值。

默认值：`180`

无提示模式的选项名称：`--deployment-timeout`

日志路径

包含 Greengrass 组和核心设置操作相关信息的日志文件的位置。使用此日志可对 Greengrass 组和核心设置过程中遇到的部署和其他问题进行故障排除。

默认值：`./`

无提示模式的选项名称：`--log-path`

详细程度

指示在脚本运行时是否在终端中打印详细的日志信息。您可以使用此信息对设备安装程序进行故障排除。

默认值：no

无提示模式的选项名称：`--verbose` - 此选项没有值。如果要打印详细日志信息，请将其包含在命令中。

在无提示模式下运行 Greengrass 设备安装程序

您可以在无提示模式下运行 Greengrass 设备安装程序，以便脚本不会提示您输入任何值。要在无提示模式下运行，请在启动脚本后指定 `bootstrap-greengrass` 模式和输入值。如果要使用输入值的默认值，则可以省略输入值。

此过程取决于您是在启动脚本之前提供 AWS 账户 凭证作为环境变量，还是在启动脚本后提供账户凭证作为输入值。

提供凭证作为环境变量

1. 提供您的凭证作为环境变量。以下示例导出临时凭证，其中包括会话令牌。

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

> **Note**
>
> 如果您在 Raspbian 或 OpenWrt 平台上运行 Greengrass 设备安装程序，请复制这些命令。重新启动设备后，必须再次提供它们。

2. 下载并启动脚本。根据需要提供输入值。例如：

   - 要使用所有默认值，请执行以下操作：

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

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

- 要指定自定义值，请执行以下操作：

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
greengrass
--region us-east-1
--group-name Custom_Group_Name
--core-name Custom_Core_Name
--ggc-root-path /custom/ggc/root/path
--deployment-timeout 300
--log-path /customized/log/path
--hello-world-lambda
--verbose
```

> **Note**
>
> 要使用 `curl` 下载脚本，请在命令中将 `wget -q -O` 替换为 `curl`。

3. 如果您的核心设备正在运行 Raspbian 或 OpenWrt，请在提示时重新启动设备，提供凭证，然后重新启动脚本。

   a. 当提示重新启动设备时，请运行以下命令之一。

   对于 Raspbian 平台：

   ```
   sudo reboot
   ```

   对于 OpenWrt 平台：

   ```
   reboot
   ```

   b. 设备重新启动后，打开终端并提供您的凭证作为环境变量。

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

```
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

c. 重新启动脚本。

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

d. 当提示是使用上一个会话中的输入值还是启动新安装时，输入 yes 以重复使用输入值。

> **Note**
>
> 在需要重新启动的平台上，上一个会话中的输入值（凭证除外）将临时存储在 GreengrassDeviceSetup.config.info 文件中。

安装完成后，终端会显示如下所示的状态消息。

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

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


Setup information:

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

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

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

4. 如果您包含了 Hello World Lambda 函数，则 Greengrass 设备安装程序会将 Greengrass 组部署到您的核心设备。要测试 Lambda 函数，或者要了解如何从组中移除 Lambda 函数的信息，请继续学习入门教程模块 3-1 中的 the section called “验证 Lambda 函数是否在核心设备上运行”。

> **Note**
>
> 确保在控制台中选择的 AWS 区域 与您用于配置 Greengrass 环境的区域相同。默认情况下，区域为美国西部（俄勒冈州）。

如果您没有包含 Hello World Lambda 函数，可以创建自己的 Lambda 函数或尝试其他 Greengrass 功能。例如，您可以将 Docker 应用程序部署连接器添加到组，并使用它将 Docker 容器部署到您的核心设备。

提供凭证作为输入值

1. 下载并启动脚本。提供您的凭证和您要指定的任何其他输入值。以下示例说明如何提供临时凭证，其中包括会话令牌。

   - 要使用所有默认值，请执行以下操作：

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --aws-access-key-id AKIAIOSFODNN7EXAMPLE
   --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
   ```

   - 要指定自定义值，请执行以下操作：

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --aws-access-key-id AKIAIOSFODNN7EXAMPLE
   --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
   --region us-east-1
   --group-name Custom_Group_Name
   --core-name Custom_Core_Name
   --ggc-root-path /custom/ggc/root/path
   --deployment-timeout 300
   --log-path /customized/log/path
   --hello-world-lambda
   --verbose
   ```

> **Note**
>
> 如果您在 Raspbian 或 OpenWrt 平台上运行 Greengrass 设备安装程序，请复制您的凭证。重新启动设备后，必须再次提供它们。
> 要使用 `curl` 下载脚本，请在命令中将 `wget -q -O` 替换为 `curl`。

2. 如果您的核心设备正在运行 Raspbian 或 OpenWrt，请在提示时重新启动设备，提供凭证，然后重新启动脚本。

   a. 当提示重新启动设备时，请运行以下命令之一。

      对于 Raspbian 平台：

      ```
      sudo reboot
      ```

      对于 OpenWrt 平台：

      ```
      reboot
      ```

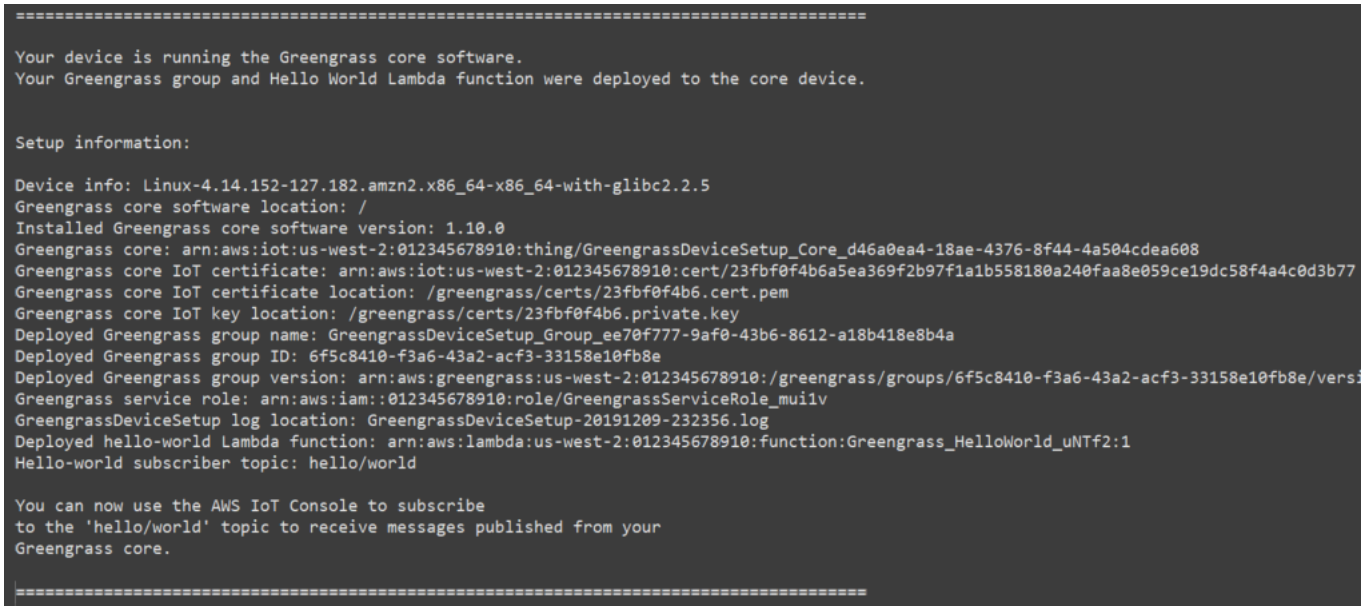
   b. 重新启动脚本。您必须在命令中包含您的凭证，但不包括其他输入值。例如：

      ```
      sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
      --aws-access-key-id AKIAIOSFODNN7EXAMPLE
      --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
      --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
      ```

   c. 当提示是使用上一个会话中的输入值还是启动新安装时，输入 `yes` 以重复使用输入值。

      > **Note**
      >
      > 在需要重新启动的平台上，上一个会话中的输入值（凭证除外）将临时存储在 `GreengrassDeviceSetup.config.info` 文件中。

   安装完成后，终端会显示如下所示的状态消息。



3. 如果您包含了 Hello World Lambda 函数，则 Greengrass 设备安装程序会将 Greengrass 组部署到您的核心设备。要测试 Lambda 函数，或者要了解如何从组中移除 Lambda 函数的信息，请继续学习入门教程模块 3-1 中的 the section called “验证 Lambda 函数是否在核心设备上运行”。

> **Note**
>
> 确保在控制台中选择的 AWS 区域 与您用于配置 Greengrass 环境的区域相同。默认情况下，区域为美国西部（俄勒冈州）。

如果您没有包含 Hello World Lambda 函数，可以创建自己的 Lambda 函数或尝试其他 Greengrass 功能。例如，您可以将 Docker 应用程序部署连接器添加到组，并使用它将 Docker 容器部署到您的核心设备。

模块 1：Greengrass 的环境设置

本模块将说明如何准备好开箱即用的 Raspberry Pi、Amazon EC2 实例或其他设备以作为您的 AWS IoT Greengrass 核心设备供 AWS IoT Greengrass 使用。

> **Tip**
>
> 或者，要使用为您设置核心设备的脚本，请参阅the section called “快速入门：Greengrass 设备安装程序”。

本模块应该需要不到 30 分钟即可完成。

在开始之前，请阅读本教程的要求。然后，按照下列主题之一中的设置说明操作。仅选择适用于核心设备类型的主题。

主题

- 设置 Raspberry Pi
- 设置 Amazon EC2 实例
- 设置其他设备

> **Note**
>
> 要了解如何使用预先构建的 Docker 容器中运行的 AWS IoT Greengrass，请参阅the section called “在 Docker 容器中运行 AWS IoT Greengrass”。

设置 Raspberry Pi

按照本主题中的步骤设置 Raspberry Pi 用作一个 AWS IoT Greengrass 核心。

> **Tip**
>
> AWS IoT Greengrass 还提供了其他选项用于安装 AWS IoT Greengrass Core 软件。例如，您可以使用 Greengrass 设备设置来配置环境并安装最新版本的 AWS IoT Greengrass Core 软件。或者，在支持的 Debian 平台上，您可以使用 APT 软件包管理器 来安装或升级 AWS IoT Greengrass Core 软件。有关更多信息，请参阅the section called “安装 AWS IoT Greengrass Core 软件”。

如果您是首次设置 Raspberry Pi，则必须执行所有这些步骤。否则，可以跳至步骤 9。不过，我们建议您使用步骤 2 中推荐的操作系统为您的 Raspberry Pi 重新创建镜像。

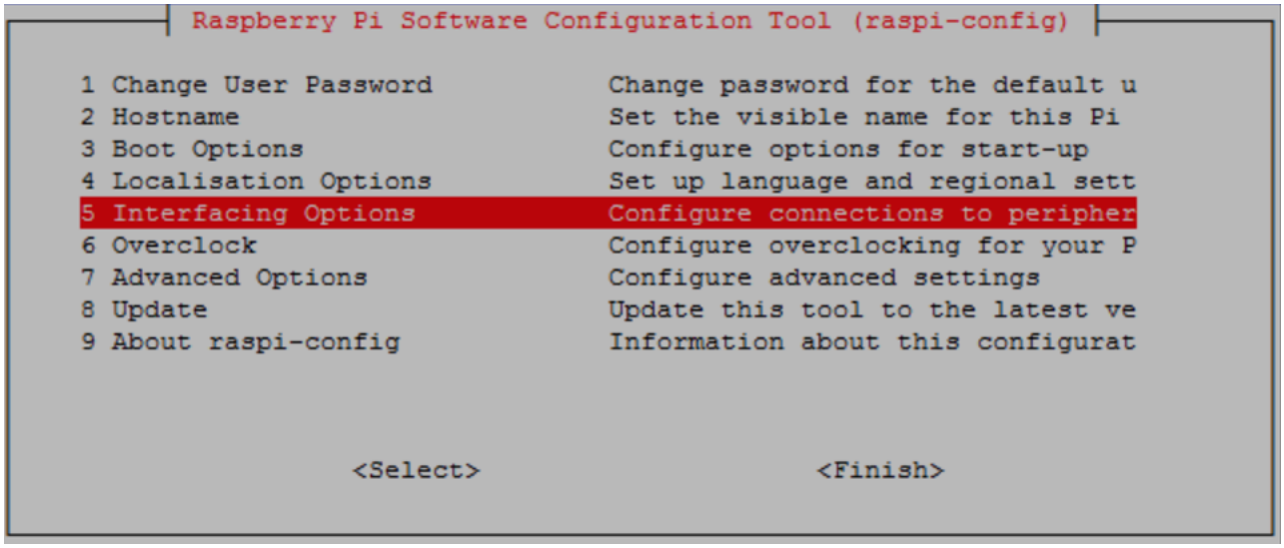
1. 下载并安装 SD 卡格式化程序，例如 SD Memory Card Formatter。将 SD 卡插入到您的计算机中。启动程序并选择已插入 SD 卡的驱动器。可以对 SD 卡执行快速格式化。
2. 下载 zip 文件格式的 Raspbian Buster 操作系统。
3. 使用 SD 卡写入工具（例如 Etcher），按照此工具的说明将下载的 zip 文件传输到 SD 上。由于操作系统映像很大，因此，该步骤可能需要一些时间。从您的计算机中弹出 SD 卡，然后将 microSD 卡插入您的 Raspberry Pi 上。
4. 对于第一次启动，我们建议您将 Raspberry Pi 连接到监视器 (通过 HDMI)、键盘和鼠标。接下来，将您的 Pi 连接到微型 USB 电源，然后 Raspbian 操作系统将启动。
5. 您可能需要配置 Pi 的键盘布局，然后再继续。为此，请选择右上方的 Raspberry 图标，再选择 Preferences (首选项)，然后选择 Mouse and Keyboard Settings (鼠标和键盘设置)。接下来，在 Keyboard (键盘) 选项卡上，选择 Keyboard Layout (键盘布局)，然后选择适当的键盘形式。
6. 接下来，通过 Wi-Fi 网络将 Raspberry Pi 连接到 Internet 或者通过以太网电缆执行该操作。

   > **Note**
   >
   > 将 Raspberry Pi 连接到计算机所连接的相同 网络，并确保计算机和 Raspberry Pi 能够访问 Internet，然后再继续。如果在工作环境中或者位于防火墙后面，可能需要将 Pi 和计算机连接到来宾网络，以便两个设备在同一网络上。但是，这种方法可能会断开计算机与本地网络资源（例如 Intranet）的连接。一个解决方案是将 Pi 连接到来宾 Wi-Fi 网络，将计算机连接到来宾 Wi-Fi 网络并且 通过以太网电缆连接到本地网络。在此配置中，计算机应该能够通过来宾 Wi-Fi 网络连接到 Raspberry Pi，并通过以太网电缆连接到本地网络资源。

7. 您必须设置 Pi 上的 SSH 才能远程连接到它。在您的 Raspberry Pi 上，打开终端窗口并运行以下命令：

   ```
   sudo raspi-config
   ```

   您将看到以下内容：



向下滚动并选择 Interfacing Options，然后选择 P2 SSH。系统提示时，请选择 Yes。（使用 Tab 键，然后按 Enter）。SSH 现在应该已启用。选择 OK（确定）。使用 Tab 键选择 Finish (完成)，然后按 Enter。如果 Raspberry Pi 未自动重启，请运行以下命令：

```
sudo reboot
```

8. 在您的 Raspberry Pi 上，在终端中运行以下命令：

```
hostname -I
```

这将返回您的 Raspberry Pi 的 IP 地址。

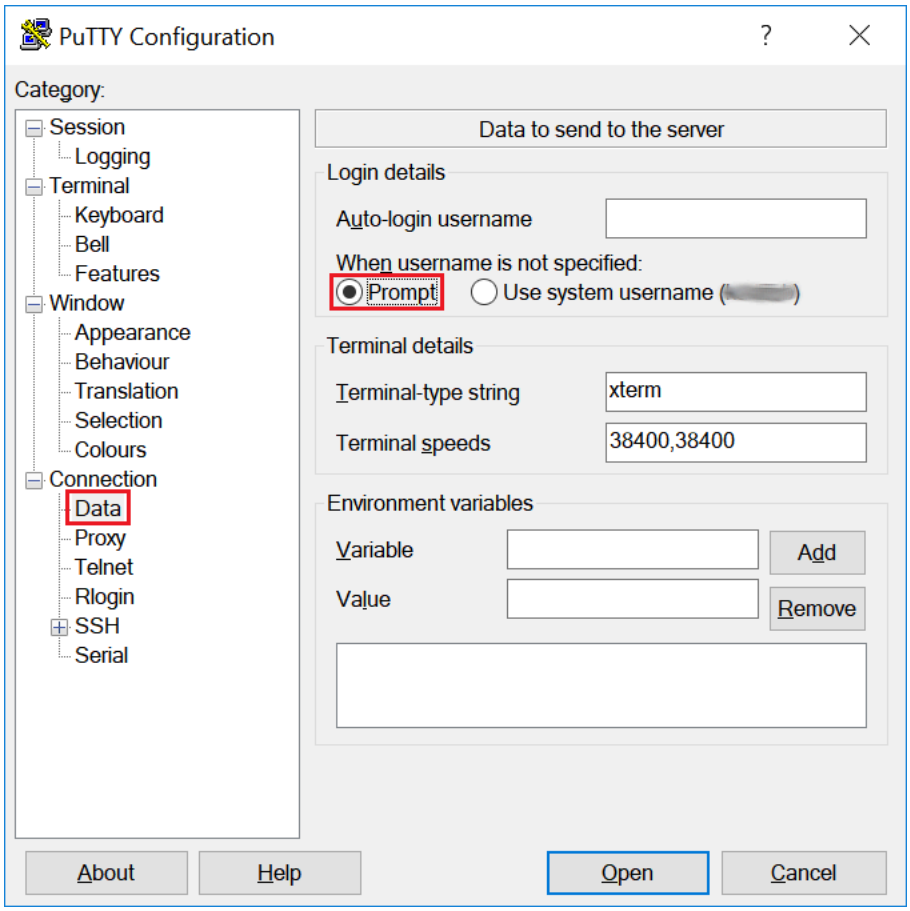
> **Note**
>
> 对于下面的内容，如果收到 ECDSA 密钥指纹消息 (`Are you sure you want to continue connecting (yes/no)?`)，请输入 `yes`。Raspberry Pi 的默认密码是 **`raspberry`**。

如果使用的是 macOS，请打开终端窗口并输入：

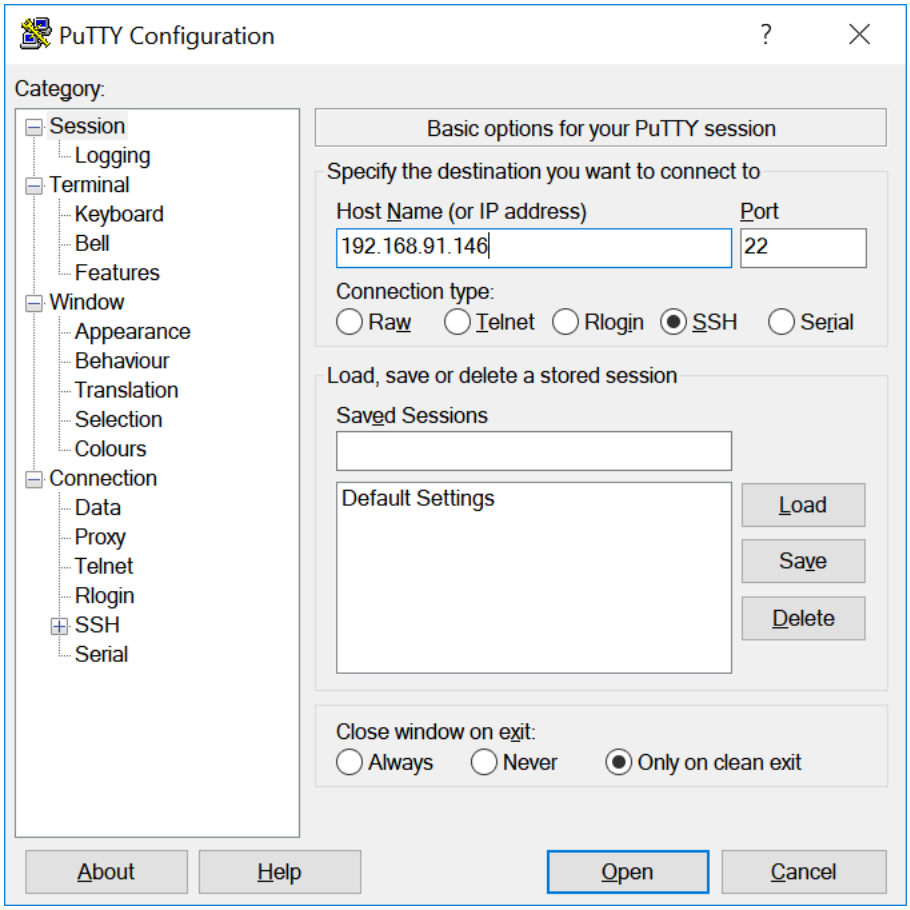
```
ssh pi@IP-address
```

`IP-address` 是通过使用 `hostname -I` 命令获取的 Raspberry Pi 的 IP 地址。

如果您使用的是 Windows，则需要安装和配置 PuTTY。展开 Connection (连接)，选择 Data (数据)，并确保已选择 Prompt (提示)：

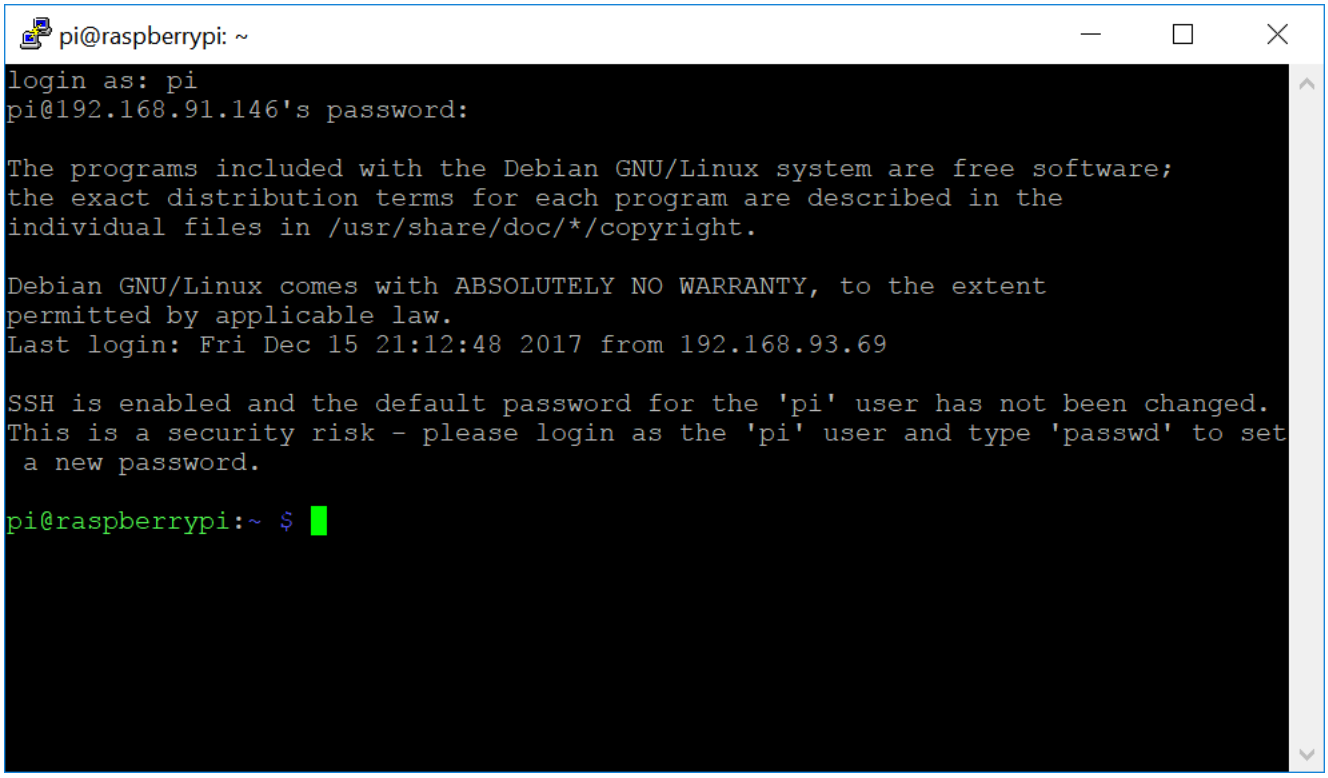


接下来，选择 Session (会话)，输入 Raspberry Pi 的 IP 地址，然后使用默认设置选择 Open (打开)。



如果显示 PuTTY 安全警报，请选择 Yes (是)。

默认 Raspberry Pi 登录名和密码分别为 **`pi`** 和 **`raspberry`**。



> **Note**
>
> 如果计算机使用 VPN 连接到远程网络，则可能难以使用 SSH 从计算机连接到 Raspberry Pi。

9. 现在，您已准备好为 AWS IoT Greengrass 设置 Raspberry Pi。首先，请从本地 Raspberry Pi 终端窗口或 SSH 终端窗口运行以下命令：

> **Tip**
>
> AWS IoT Greengrass 还提供了其他选项用于安装 AWS IoT Greengrass Core 软件。例如，您可以使用 Greengrass 设备设置来配置环境并安装最新版本的 AWS IoT Greengrass Core 软件。或者，在支持的 Debian 平台上，您可以使用 APT 软件包管理器来安装或升级 AWS IoT Greengrass Core 软件。有关更多信息，请参阅the section called “安装 AWS IoT Greengrass Core 软件”。

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

10. 为了提高 Pi 设备上的安全性，请在启动时在操作系统上启用硬链接和软链接 (symlink) 保护。

    a. 导航到 98-rpi.conf 文件。

    ```
    cd /etc/sysctl.d
    ls
    ```

    > **Note**
    >
    > 如果您没有看到 98-rpi.conf 文件，请按照 README.sysctl 文件中的说明操作。

    b. 使用文本编辑器（如 Leafpad、GNU nano 或 vi）将以下两行添加到文件的末尾。您可能需要使用 sudo 命令以根身份进行编辑（例如，sudo nano 98-rpi.conf）。

    ```
    fs.protected_hardlinks = 1
    fs.protected_symlinks = 1
    ```

    c. 重启 Pi。

    ```
    sudo reboot
    ```

    在约 1 分钟后，使用 SSH 连接到 Pi，然后运行以下命令来确认更改：

    ```
    sudo sysctl -a 2> /dev/null | grep fs.protected
    ```

    您现在会看到 fs.protected_hardlinks = 1 和 fs.protected_symlinks = 1。

11. 编辑命令行启动文件，以启用并装载内存控制组。这将允许 AWS IoT Greengrass 设置 Lambda 函数的内存限制。Cgroup 还需要在默认的容器化模式下运行 AWS IoT Greengrass。

    a. 导航到您的 boot 目录。

    ```
    cd /boot/
    ```

    b. 使用文本编辑器打开 cmdline.txt。将以下内容附加到现有行的末尾，而不是作为新行。您可能需要使用 sudo 命令以根身份进行编辑（例如，sudo nano cmdline.txt）。

    ```
    cgroup_enable=memory cgroup_memory=1
    ```

c. 现在重启 Pi。

```
sudo reboot
```

您的 Raspberry Pi 现在应该已为 AWS IoT Greengrass 做好准备。

12. 可选。安装流管理器所需的 Java 8 运行时。此教程不使用流管理器，但它将使用默认情况下启用流管理器的 Default Group creation (默认组创建) 工作流。在部署组之前，使用此命令在核心设备上安装 Java 8 运行时或禁用流管理器。模块 3 中提供了有关禁用流管理器的说明。

```
sudo apt install openjdk-8-jdk
```

13. 要确保您具有所需的所有依赖项，请从 GitHub AWS IoT Greengrass 示例存储库下载并运行 Greengrass 依赖项检查程序。这些命令将在 `Downloads` 目录中解压和运行依赖项检查程序脚本。

> **Note**
>
> 如果您运行的是 Raspbian 内核版本 5.4.51，则依赖项检查器可能会失败。此版本无法正确安装内存 cgroup。这可能会导致在容器模式下运行的 Lambda 函数失败。
> 有关更新内核的更多信息，请参阅 Raspberry Pi 论坛中的内核升级后未加载的 Cgroup。

```
cd /home/pi/Downloads
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

出现 `more` 时，按 Spacebar 键以显示另一屏文本。

> **Important**
>
> 本教程需要 Python 3.7 运行时才能运行本地 Lambda 函数。启用流管理器后，还需要 Java 8 运行时。如果 `check_ggc_dependencies` 脚本生成提示缺少这些必备运行时的警告，请确保先安装它们，然后再继续。您可以忽略提示缺少其他可选运行时的警告。

要获得有关 modprobe 命令的信息，请在终端中运行 man modprobe。

您的 Raspberry Pi 配置已完成。继续浏览 the section called “模块 2：安装 AWS IoT Greengrass Core 软件”。

设置 Amazon EC2 实例

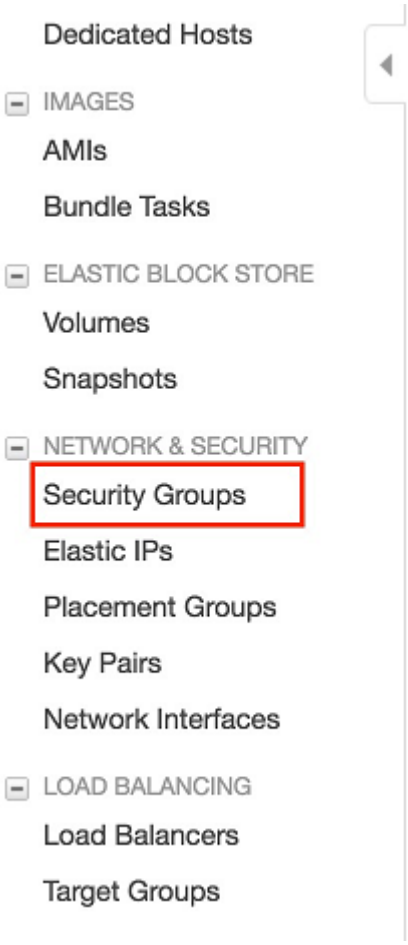
按照本主题中的步骤设置 Amazon EC2 实例用作您的 AWS IoT Greengrass 核心。

> **Tip**
>
> 或者，要使用脚本来设置您的环境并为您安装 AWS IoT Greengrass Core 软件，请参阅the section called “快速入门：Greengrass 设备安装程序”。

尽管您可以使用 Amazon EC2 实例完成本教程，但 AWS IoT Greengrass 最好与物理硬件一起使用。我们建议您尽可能设置 Raspberry Pi，而不是使用 Amazon EC2 实例。如果您使用的是 Raspberry Pi，则无需按照本主题中的步骤操作。

1. 登录 AWS Management Console 并使用 Amazon Linux AMI 启动一个 Amazon EC2 实例。有关 Amazon EC2 实例的信息，请参阅 Amazon EC2 入门指南。
2. 在您的 Amazon EC2 实例运行后，启用端口 8883 以允许传入的 MQTT 通信，以便其他设备可以与核心连接。 AWS IoT Greengrass

   a. 在 Amazon EC2 控制台的导航窗格中，选择 安全组。



b. 选择您刚刚启动的实例的安全组，然后选择入站规则选项卡。

c. 选择编辑入站规则。

   要启用端口 8883，您应向安全组添加自定义 TCP 规则。有关更多信息，请参阅 Amazon EC2 用户指南中的向安全组添加规则。

d. 在编辑入站规则页面上，选择添加规则，输入以下设置，然后选择保存。

   - 对于 Type（类型），选择 Custom TCP Rule（自定义 TCP 规则）。
   - 对于端口范围，输入 **8883**。
   - 对于 Source（源），请选择 Anywhere（任何位置）。
   - 对于描述，输入 **MQTT Communications**。

3. 连接到 Amazon EC2 实例。

   a. 在导航窗格中，选择 Instances (实例)，选择您的实例，然后选择 Connect (连接)。

b. 按照 Connect To Your Instance (连接到您的实例) 页面上的说明操作，使用 SSH 和您的私有密钥文件连接到您的实例。

您可以对 Windows 使用 PuTTY，对 macOS 使用终端。有关更多信息，请参阅 Amazon EC2 用户指南中的连接到您的 Linux 实例。

您现在已准备好为 AWS IoT Greengrass设置 Amazon EC2 实例。

4. 连接到 Amazon EC2 实例后，创建 `ggc_user` 和 `ggc_group` 账户：

```
sudo adduser --system ggc_user
sudo groupadd --system ggc_group
```

> **Note**
>
> 如果 `adduser` 命令在系统上不可用，请使用以下命令。
>
> ```
> sudo useradd --system ggc_user
> ```

5. 为了提高安全性，请确保在启动时在 Amazon EC2 实例的操作系统上启用硬链接和软链接 (symlink) 保护。

> **Note**
>
> 启用硬链接和软链接保护的步骤因操作系统而异。请参阅您的发行版的文档。

a. 运行以下命令以检查是否启用了硬链接和软链接保护：

```
sudo sysctl -a | grep fs.protected
```

如果硬链接和软链接设置为 1，则您的保护已正确启用。继续执行步骤 6。

> **Note**
>
> 软链接以 `fs.protected_symlinks` 表示。

b. 如果硬链接和软链接未设置为 1，请启用这些保护。导航到您的系统配置文件。

```
cd /etc/sysctl.d
ls
```

c. 使用您最喜欢的文本编辑器（Leafpad、GNU nano 或 vi）将以下两行添加到系统配置文件的末尾。在 Amazon Linux 1 上，这是 `00-defaults.conf` 文件。在 Amazon Linux 2 上，这是 `99-amazon.conf` 文件。您可能需要更改权限（使用 `chmod` 命令）以对此文件进行写入，或使用 `sudo` 命令以根身份进行编辑（例如，`sudo nano 00-defaults.conf`）。

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

d. 重启 Amazon EC2 实例。

```
sudo reboot
```

数分钟后，使用 SSH 连接到您的实例，然后运行以下命令来确认更改。

```
sudo sysctl -a | grep fs.protected
```

您应该看到硬链接和软链接都设置为 1。

6. 提取并运行以下脚本以挂载 Linux 控制组 (cgroups)。这 AWS IoT Greengrass 允许为 Lambda 函数设置内存限制。Cgroup 还需要 AWS IoT Greengrass 在默认的容器化模式下运行。

```
curl https://raw.githubusercontent.com/tianon/cgroupfs-
mount/951c38ee8d802330454bdede20d85ec1c0f8d312/cgroupfs-mount > cgroupfs-mount.sh
chmod +x cgroupfs-mount.sh
sudo bash ./cgroupfs-mount.sh
```

您的 Amazon EC2 实例现在应该已为 AWS IoT Greengrass做好准备。

7. 可选。安装流管理器所需的 Java 8 运行时。此教程不使用流管理器，但它将使用默认情况下启用流管理器的 Default Group creation (默认组创建) 工作流。在部署组之前，使用此命令在核心设备上安装 Java 8 运行时或禁用流管理器。模块 3 中提供了有关禁用流管理器的说明。

- 对于基于 Debian 的发行版：

```
sudo apt install openjdk-8-jdk
```

- 对于基于 Red Hat 的发行版：

```
sudo yum install java-1.8.0-openjdk
```

8. 为确保您拥有所有必需的依赖项，请从示例存储库下载并运行 Greengrass 依赖项检查器。AWS IoT Greengrass GitHub 这些命令将在您的 Amazon EC2 实例中下载、解压并运行依赖项检查程序脚本。

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo ./check_ggc_dependencies | more
```

> **Important**
>
> 本教程需要 Python 3.7 运行时才能运行本地 Lambda 函数。启用流管理器后，还需要 Java 8 运行时。如果 `check_ggc_dependencies` 脚本生成提示缺少这些必备运行时的警告，请确保先安装它们，然后再继续。您可以忽略提示缺少其他可选运行时的警告。

您的 Amazon EC2 实例配置已完成。继续the section called “模块 2：安装 AWS IoT Greengrass Core 软件”。

设置其他设备

按照本主题中的步骤设置设备（不包括 Raspberry Pi）用作您的 AWS IoT Greengrass 核心。

> **Tip**
>
> 或者，要使用用于设置环境并为您安装 AWS IoT Greengrass 核心软件的脚本，请参阅the section called “快速入门：Greengrass 设备安装程序”。

如果您刚开始使用 AWS IoT Greengrass，我们建议您使用 Raspberry Pi 或 Amazon EC2 实例作为您的核心设备，并且按照适合您的设备的设置步骤进行操作。

如果你打算使用 Yocto Project 构建基于 Linux 的自定义系统，你可以使用 `meta-aws` 项目中的 AWS IoT Greengrass Bitbake 配方。此配方还可以帮助您开发支持嵌入式应用程序 AWS 边缘软件的软件平台。Bitbake 版本可在您的设备上安装、配置并自动运行 AWS IoT Greengrass Core 软件。

Yocto 项目

一个开源协作项目，可帮助您为嵌入式应用程序构建基于 Linux 的自定义系统，无论硬件架构如何。有关更多信息，请参阅 Yocto 项目。

`meta-aws`

一个提供 Yocto 配方的 AWS 托管项目。您可以使用这些配方在基于 Linux 的系统中开发使用 OpenEmbedded 和 Yocto 项目构建的 AWS 边缘软件。有关此社区支持的功能的更多信息，请参阅 GitHub 上的 `meta-aws` 项目。

`meta-aws-demos`

包含 `meta-aws` 项目演示的 AWS 托管项目。有关集成过程的更多示例，请参阅 GitHub 上的 `meta-aws-demos` 项目。

要使用不同的设备或受支持的平台，请按照本主题中的步骤操作。

1. 如果您的核心设备是 NVIDIA Jetson 设备，您必须先使用 JetPack 4.3 安装程序刷写该固件。如果要配置不同的设备，请跳至步骤 2。

   > **Note**
   >
   > 您使用的 JetPack 安装程序版本基于目标 CUDA 工具包版本。以下说明使用 JetPack 4.3 和 CUDA Toolkit 10.0。有关使用适合您设备的版本的信息，请参阅 NVIDIA 文档中的如何安装 Jetpack。

   a. 在运行 Ubuntu 16.04 或更高版本的物理桌面上，使用 JetPack 4.3 安装程序切换该固件，如 NVIDIA 文档的下载并安装 JetPack (4.3) 中所示。

      按照安装程序中的说明将所有软件包和依赖项安装在 Jetson 板上，后者必须通过 Micro-B 电缆连接到桌面。

   b. 以正常模式重启您的面板，然后将一个显示屏连接到面板。

> **Note**
>
> 使用 SSH 连接到 Jetson 板时，请使用默认用户名 (**nvidia**) 和默认密码 (**nvidia**)。

2. 运行以下命令以创建用户 `ggc_user` 和组 `ggc_group`。根据核心设备上安装的分配，运行的命令将有所不同。

   - 如果核心设备正在运行 OpenWrt，请运行以下命令：

   ```
   opkg install shadow-useradd
   opkg install shadow-groupadd
   useradd --system ggc_user
   groupadd --system ggc_group
   ```

   - 否则，请运行以下命令：

   ```
   sudo adduser --system ggc_user
   sudo addgroup --system ggc_group
   ```

   > **Note**
   >
   > 如果 `addgroup` 命令在系统上不可用，请使用以下命令。
   >
   > ```
   > sudo groupadd --system ggc_group
   > ```

3. 可选。安装流管理器所需的 Java 8 运行时。此教程不使用流管理器，但它将使用默认情况下启用流管理器的 Default Group creation (默认组创建) 工作流。在部署组之前，使用此命令在核心设备上安装 Java 8 运行时或禁用流管理器。模块 3 中提供了有关禁用流管理器的说明。

   - 对于基于 Debian 或基于 Ubuntua 的发行版：

   ```
   sudo apt install openjdk-8-jdk
   ```

   - 对于基于 Red Hat 的发行版：

   ```
   sudo yum install java-1.8.0-openjdk
   ```

4. 要确保您具有所需的所有依赖项，请从 GitHub AWS IoT Greengrass 示例存储库下载并运行 Greengrass 依赖项检查程序。这些命令将解压和运行依赖项检查程序脚本。

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo ./check_ggc_dependencies | more
```

> **Note**
>
> `check_ggc_dependencies` 脚本在 AWS IoT Greengrass 支持的平台上运行并需要特定的 Linux 系统命令。有关更多信息，请参阅依赖项检查程序的自述文件。

5. 按照依赖项检查程序输出的指示，在设备上安装所有必需的依赖项。对于缺少的内核级别依赖项，您可能需要重新编译内核。要安装 Linux 控制组 (`cgroups`)，您可以运行 cgroupfs-mount 脚本。这将允许 AWS IoT Greengrass 设置 Lambda 函数的内存限制。Cgroup 还需要在默认的容器化模式下运行 AWS IoT Greengrass。

   如果没有错误在输出中显示，AWS IoT Greengrass 应该能够在您的设备上成功运行。

   > **Important**
   >
   > 本教程需要 Python 3.7 运行时才能运行本地 Lambda 函数。启用流管理器后，还需要 Java 8 运行时。如果 `check_ggc_dependencies` 脚本生成提示缺少这些必备运行时的警告，请确保先安装它们，然后再继续。您可以忽略提示缺少其他可选运行时的警告。

   有关 AWS IoT Greengrass 要求和依赖项的列表，请参阅the section called “支持的平台和要求”。

模块 2：安装 AWS IoT Greengrass Core 软件

本模块向您介绍如何在您选择的设备上安装 AWS IoT Greengrass 核心软件。在此模块中，您首先创建一个 Greengrass 组和核心。然后，您在核心设备上下载、配置和启动该软件。有关 AWS IoT Greengrass Core 软件功能的更多信息，请参阅the section called “配置 AWS IoT Greengrass 核心”。

开始之前，请确保您已为所选设备完成模块 1 中的设置步骤。

> **Tip**
>
> AWS IoT Greengrass 还提供了其他选项用于安装 AWS IoT Greengrass Core 软件。例如，您可以使用 Greengrass 设备设置来配置环境并安装最新版本的 AWS IoT Greengrass Core 软件。或者，在支持的 Debian 平台上，您可以使用 APT 软件包管理器 来安装或升级 AWS IoT Greengrass Core 软件。有关更多信息，请参阅the section called “安装 AWS IoT Greengrass Core 软件”。

本模块应该需要不到 30 分钟即可完成。

主题

- 预配置 AWS IoT 事物以用作 Greengrass 核心
- 为核心创建一个 AWS IoT Greengrass 组
- 在核心设备上安装并运行 AWS IoT Greengrass

预配置 AWS IoT 事物以用作 Greengrass 核心

Greengrass 核心是运行 AWS IoT Greengrass Core 软件来管理本地 IoT 进程的设备。要设置 Greengrass 核心，您需要创建 AWS IoT 事物 来表示连接到 AWS IoT 的设备或逻辑实体 将设备注册为 AWS IoT 事物后，该设备可以使用允许其访问 AWS IoT 的数字证书和密钥。您可以使用 AWS IoT 策略来允许设备与 AWS IoT 和 AWS IoT Greengrass 服务通信。

在此部分中，您要将设备注册为 AWS IoT 事物，以用作 Greengrass 核心。

创建 AWS IoT 事物

1. 导航到 AWS IoT 控制台。
2. 在管理下，展开所有设备，然后选择事物。
3. 在 Things（事物）页面上，选择 Create things（创建事物）。
4. 在 创建事物页面上，选择 创建单个事物，然后选择 下一步。
5. 在 指定事物属性 页面上，执行以下操作：
   a. 在事物名称中，输入一个名称来表示您的设备，例如 **`MyGreengrassV1Core`**。
   b. 选择 Next（下一步）。
6. 在配置设备证书页面上，选择下一步。

7. 在 将策略附加到证书 页面上，执行下列操作之一：

   - 选择授予核心所需权限的现有策略，然后选择创建事物。

     这将打开一个模态，供您下载设备用于连接到 AWS Cloud 和核心的证书和密钥。

   - 创建并附加一个新策略，以授予核心设备权限。执行以下操作：

     a. 选择 Create policy（创建策略）。

        此时将在新选项卡中打开创建策略页面。

     b. 在创建策略页面上，执行以下操作：

        i. 在策略名称中，输入一个名称来描述该策略，例如 **GreengrassV1CorePolicy**。

        ii. 在策略语句选项卡的策略文档下，选择 JSON。

        iii. 输入以下策略文档。此策略允许核心与 AWS IoT Core 服务通信，与设备影子交互以及与 AWS IoT Greengrass 服务通信。有关如何根据您的使用场景限制此策略的访问权限的信息，请参阅AWS IoT Greengrass 核心设备的最低 AWS IoT 策略。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "*"
```

```
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. 选择 Create（创建）以创建策略。

c. 返回到打开了将策略附加到证书页面的浏览器标签页中。执行以下操作：

i. 在策略列表中，选择您创建的策略，例如 GreengrassV1CorePolicy。

如果没有看到策略，请选择刷新按钮。

ii. 选择 Create thing（创建事物）。

这将打开一个模态，供您下载核心用于连接到 AWS IoT 的证书和密钥。

8. 返回到打开了将策略附加到证书页面的浏览器标签页中。执行以下操作：

a. 在策略列表中，选择您创建的策略，例如 GreengrassV1CorePolicy。

如果没有看到策略，请选择刷新按钮。

b. 选择 Create thing（创建事物）。

这将打开一个模态，供您下载核心用于连接到 AWS IoT 的证书和密钥。

9. 在下载证书和密钥模态中，下载设备的证书。

> **Important**
> 请先下载安全资源，然后再选择 完成。

执行以下操作：

a. 在设备证书中，选择下载以下载该设备证书。

b. 在公钥文件中，选择下载以下载该证书的公钥。

c. 在私钥文件中，选择下载以下载该证书的私钥文件。

d. 查看《AWS IoT 开发人员指南》中的服务器身份验证，然后选择相应的根 CA 证书。我们建议您使用 Amazon Trust Services (ATS) 端点和 ATS 根 CA 证书。在根 CA 证书下，针对根 CA 证书选择下载。

e. 选择完成。

记下设备证书和密钥文件名中常见的证书 ID。稍后您将需要用到它。

为核心创建一个 AWS IoT Greengrass 组

AWS IoT Greengrass 组包含设置和有关其组件的其他信息，如客户端设备、Lambda 函数和连接器。一个组定义内核的配置，包括其组件如何相互交互。

在本部分中，您将为核心创建群组。

> **Tip**
>
> 有关使用 AWS IoT Greengrass API 来创建和部署组的示例，请参阅 GitHub 中的 gg_group_setup 存储库。

为核心创建群组

1. 导航到 AWS IoT 控制台。
2. 在管理下，展开 Greengrass 设备，然后选择组 (V1)。

> **Note**
>
> 如果您未看到 Greengrass 设备菜单，请更改为支持 AWS IoT Greengrass V1 的 AWS 区域。有关支持的区域列表，请参见 AWS 一般参考 中的 AWS IoT Greengrass V1 端点和配额。您必须在AWS IoT Greengrass V1可用的区域中为您的核心创建 AWS IoT 事物。

3. 在 Greengrass 组页面上，选择创建组。
4. 在创建 Greengrass 组页上，执行以下操作：

a. 对于 Greengrass 组名，请输入描述组的名称，例如 **`MyGreengrassGroup`**。

b. 对于 Greengrass 核心，请选择您之前创建的 AWS IoT 事物，例如 MyGreengrassV1Core。

控制台会自动为您选择事物的设备证书。

c. 选择创建组。

在核心设备上安装并运行 AWS IoT Greengrass

> **Note**
>
> 虽然本教程提供了有关在 Raspberry Pi 上启动 AWS IoT Greengrass Core 软件的说明，但您可以使用任何受支持的设备。

本节中，您将在核心设备上配置、安装和运行 AWS IoT Greengrass Core 软件。

要安装并运行 AWS IoT Greengrass

1. 从本指南的 AWS IoT Greengrass Core 软件部分中，下载 AWS IoT Greengrass Core 软件安装包。选择最适合核心设备的 CPU 架构、发行版和操作系统的安装包。
   - 对于 Raspberry Pi，请下载适用于 Armv7l 架构和 Linux 操作系统的软件包。
   - 对于 Amazon EC2 实例，请下载适用于 x86_64 架构和 Linux 操作系统的安装包。
   - 对于 NVIDIA Jetson TX2，请下载适用于 Armv8 (AArch64) 架构和 Linux 操作系统的安装包。
   - 对于 Intel Atom，请下载适用于 x86_64 架构和 Linux 操作系统的安装包。
2. 在之前的步骤中，您已将五个文件下载到计算机：
   - `greengrass-`*`OS-architecture`*`-1.11.6.tar.gz` – 此压缩文件包含在核心设备上运行的 AWS IoT Greengrass Core 软件。
   - *`certificateId`*`-certificate.pem.crt` – 设备证书文件。
   - *`certificateId`*`-public.pem.key` – 设备证书的公钥文件。
   - *`certificateId`*`-private.pem.key` – 设备证书的私钥文件。
   - `AmazonRootCA1.pem` – Amazon 根证书颁发机构 (CA) 文件。

   在此步骤中，您将计算机中的这些文件传输到核心设备。执行以下操作：

   a. 如果您不知道 Greengrass 核心设备的 IP 地址，请在核心设备上打开终端并运行以下命令。

> **Note**
>
> 此命令可能无法为某些设备返回正确的 IP 地址。请参阅设备的文档以检索设备 IP 地址。

```
hostname -I
```

b. 将计算机中的这些文件传输到核心设备。文件传输步骤会根据计算机的操作系统而有所不同。选择您的操作系统，了解说明如何将文件传输到 Raspberry Pi 设备的步骤。

> **Note**
>
> 对于 Raspberry Pi，默认用户名为 **pi**，默认密码为 **raspberry**。
> 对于 NVIDIA Jetson TX2，默认用户名为 **nvidia**，默认密码为 **nvidia**。

Windows

要将压缩文件从计算机传输到 Raspberry Pi 核心设备，请使用 WinSCP 或 PuTTY pscp 命令等工具。要使用 pscp 命令，请在计算机上打开命令提示符窗口并运行以下命令：

```
cd path-to-downloaded-files
pscp -pw Pi-password greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-certificate.pem.crt pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-public.pem.key pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-private.pem.key pi@IP-address:/home/pi
pscp -pw Pi-password AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> 此命令中的版本号必须与您的 AWS IoT Greengrass Core 软件包的版本一致。

macOS

要将压缩文件从 Mac 传输到 Raspberry Pi 核心设备，请在计算机上打开终端窗口，并运行以下命令。*path-to-downloaded-files* 通常为 ~/Downloads。

> **Note**
>
> 可能会提示您输入两个密码。如果是这样的话，第一个密码用于 Mac 的 sudo 命令，第二个是 Raspberry Pi 的密码。

```
cd path-to-downloaded-files
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi
scp certificateId-public.pem.key pi@IP-address:/home/pi
scp certificateId-private.pem.key pi@IP-address:/home/pi
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> 此命令中的版本号必须与您的 AWS IoT Greengrass Core 软件包的版本一致。

UNIX-like system

要将压缩文件从您的计算机传输到 Raspberry Pi 核心设备，请在您的计算机上打开一个终端窗口并运行以下命令：

```
cd path-to-downloaded-files
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi
scp certificateId-public.pem.key pi@IP-address:/home/pi
scp certificateId-private.pem.key pi@IP-address:/home/pi
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> 此命令中的版本号必须与您的 AWS IoT Greengrass Core 软件包的版本一致。

Raspberry Pi web browser

如果您使用了 Raspberry Pi 的 Web 浏览器来下载压缩文件，文件应该位于 Pi 的 ~/Downloads 文件夹（例如 /home/pi/Downloads）中。否则，压缩文件应位于 Pi 的 ~ 文件夹（例如 /home/pi）中。

3. 在 Greengrass 核心设备上，打开终端，然后导航到包含 AWS IoT Greengrass Core 软件和证书的文件夹。将 *path-to-transferred-files* 替换为核心设备上传输文件的路径。例如，在 Raspberry Pi 上运行 `cd /home/pi`。

```
cd path-to-transferred-files
```

4. 将 AWS IoT Greengrass Core 软件解压缩到核心设备上。运行以下命令解压缩传输到核心设备的软件档案。此命令使用 `-C /` 参数在核心设备的根文件夹中创建 `/greengrass` 文件夹。

```
sudo tar -xzvf greengrass-OS-architecture-1.11.6.tar.gz -C /
```

> **Note**
>
> 此命令中的版本号必须与您的 AWS IoT Greengrass Core 软件包的版本一致。

5. 将证书和密钥移至 AWS IoT Greengrass Core 软件文件夹。运行以下命令为证书创建文件夹，并将证书和密钥移到该文件夹。将 *path-to-transferred-files* 替换为您在核心设备上传输文件的路径，并将 *certificateId* 替换为文件名中的证书 ID。例如，在 Raspberry Pi 上，将 *path-to-transferred-files* 替换为 **/home/pi**

```
sudo mv path-to-transferred-files/certificateId-certificate.pem.crt /greengrass/
certs
sudo mv path-to-transferred-files/certificateId-public.pem.key /greengrass/certs
sudo mv path-to-transferred-files/certificateId-private.pem.key /greengrass/certs
sudo mv path-to-transferred-files/AmazonRootCA1.pem /greengrass/certs
```

6. AWS IoT Greengrass Core 软件使用为软件指定参数的配置文件。此配置文件指定证书文件的文件路径和要使用的 AWS Cloud 端点。在此步骤中，您将为核心创建 AWS IoT Greengrass Core 软件配置文件。执行以下操作：

   a. 获取核心 AWS IoT 事物的 Amazon 资源名称（ARN）。执行以下操作：

      i. 在AWS IoT控制台中的管理、Greengrass 设备下，选择组 (V1)。

      ii. 在 Greengrass 组页面上，选择您之前创建的组。

      iii. 在概述下，选择 Greengrass 核心。

      iv. 在核心详细信息页面上，复制 AWS IoT 事物 ARN，然后将其保存以在 AWS IoT Greengrass Core 配置文件中使用。

   b. 获取您的 AWS 账户 在当前区域中的 AWS IoT 设备数据端点。设备使用此端点连接至 AWS 作为 AWS IoT 事物。执行以下操作：

      i. 在 AWS IoT 控制台中，选择设置。

      ii. 在设备数据端点下，复制端点，然后将其保存以在 AWS IoT Greengrass Core 配置文件中使用。

   c. 创建 AWS IoT Greengrass Core 软件配置文件。例如，您可以运行以下命令来使用 GNU nano 来创建文件。

      ```
      sudo nano /greengrass/config/config.json
      ```

      将文件的内容替换为以下 JSON 文档。

      ```
      {
        "coreThing" : {
          "caPath": "AmazonRootCA1.pem",
          "certPath": "certificateId-certificate.pem.crt",
          "keyPath": "certificateId-private.pem.key",
          "thingArn": "arn:aws:iot:region:account-id:thing/MyGreengrassV1Core",
          "iotHost": "device-data-prefix-ats.iot.region.amazonaws.com",
          "ggHost": "greengrass-ats.iot.region.amazonaws.com",
          "keepAlive": 600
        },
        "runtime": {
          "cgroup": {
            "useSystemd": "yes"
          }
        },
      ```

```
  "managedRespawn": false,
  "crypto": {
    "caPath": "file:///greengrass/certs/AmazonRootCA1.pem",
    "principals": {
      "SecretsManager": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-
private.pem.key"
      },
      "IoTCertificate": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-
private.pem.key",
        "certificatePath": "file:///greengrass/certs/certificateId-
certificate.pem.crt"
      }
    }
  }
}
```

然后执行以下操作：

- 如果您下载的 Amazon 根 CA 证书与 Amazon 根 CA 1 不同，请将 *AmazonRootCA1.pem* 的每个实例替换为 Amazon 根 CA 文件的名称。
- 将 *certificateId* 的每个实例替换为证书和密钥文件名称中的证书 ID。
- 将 *arn:aws:iot:region:account-id:thing/MyGreengrassV1Core* 替换为您之前保存的核心事物的 ARN。
- 将 *MyGreengrassV1core* 替换为您的核心事物的名称。
- 将 *device-data-prefix-ats.iot.region.amazonaws.com* 替换为您之前保存的 AWS IoT 设备数据端点。
- 将*##*替换为您的 AWS 区域。

有关在配置文件中可指定的配置选项详细信息，请参阅 AWS IoT Greengrass 核心配置文件。

7. 确保您的核心设备已连接到 Internet。然后，在核心设备上启动 AWS IoT Greengrass。

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

您应看到 `Greengrass successfully started` 消息。记录 PID。

> **Note**
>
> 要将核心设备设置为在系统引导时启动 AWS IoT Greengrass，请参阅 the section called “在系统引导时启动 Greengrass”。

您可以运行以下命令来确认 AWS IoT Greengrass 核心软件（Greengrass 守护程序）是否正常工作。将 *PID-number* 替换为您的 PID：

```
ps aux | grep PID-number
```

您应看到 PID 的条目，其中包含指向正在运行的 Greengrass 守护程序的路径（例如，`/greengrass/ggc/packages/1.11.6/bin/daemon`）。如果您在启动 AWS IoT Greengrass 时遇到问题，请查看 排查问题。

模块 3（第 1 部分）：AWS IoT Greengrass 的 Lambda 函数

本模块介绍了如何创建和部署从 AWS IoT Greengrass 核心设备发送 MQTT 消息的 Lambda 函数。本模块介绍了 Lambda 函数配置、用于允许 MQTT 消息传递的订阅和对核心设备的部署。

模块 3（第 2 部分）讲述了在 AWS IoT Greengrass 核心上运行的按需和长时间生存的 Lambda 函数之间的差异。

在开始之前，请确保您已完成模块 1 和模块 2 并且有正在运行的 AWS IoT Greengrass 核心设备。

> **Tip**
>
> 或者，要使用为您设置核心设备的脚本，请参阅the section called “快速入门：Greengrass 设备安装程序”。该脚本还可以创建和部署此模块中使用的 Lambda 函数。

本模块应该需要大约 30 分钟才能完成。

主题

- 创建并打包 Lambda 函数
- 为 AWS IoT Greengrass 配置 Lambda 函数
- 将云配置部署到 Greengrass 核心设备

- 验证 Lambda 函数是否在核心设备上运行

创建并打包 Lambda 函数

此模块中的 Python Lambda 函数示例使用适用于 Python 的 AWS IoT Greengrass 核心开发工具包发布 MQTT 消息。

在此步骤中，您：

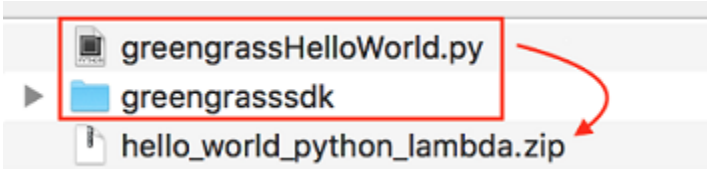
- 将适用于 Python 的 AWS IoT Greengrass 核心开发工具包下载到您的计算机（而不是 AWS IoT Greengrass 核心设备）。
- 创建包含函数代码和依赖项的 Lambda 函数部署程序包。
- 使用 Lambda 控制台创建 Lambda 函数和上传部署程序包。
- 发布 Lambda 函数的版本并创建指向该版本的别名。

要完成本模块，必须在核心设备上安装 Python 3.7。

1. 从 AWS IoT Greengrass Core 软件开发工具包下载页面，将适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包下载到您的计算机上。
2. 解压缩下载的程序包以获取 Lambda 函数代码和软件开发工具包。

   本模块中的 Lambda 函数使用：

   - `examples\HelloWorld` 中的 `greengrassHelloWorld.py` 文件。这是您的 Lambda 函数代码。该函数每 5 秒将两条可能的消息之一发布到 `hello/world` 主题一次。
   - `greengrasssdk` 文件夹。这是开发工具包。
3. 将 `greengrasssdk` 文件夹复制到包含 `greengrassHelloWorld.py` 的 `HelloWorld` 文件夹中。
4. 要创建 Lambda 函数部署程序包，请将 `greengrassHelloWorld.py` 文件和 `greengrasssdk` 文件夹保存到名为 `hello_world_python_lambda.zip` 的压缩 zip 文件。py 文件和 `greengrasssdk` 文件夹必须位于该目录的根目录中。



在类 UNIX 系统（包括 Mac 终端）上，您可以使用以下命令打包文件和文件夹：

```
zip -r hello_world_python_lambda.zip greengrasssdk greengrassHelloWorld.py
```

> **Note**
>
> 根据您的发行版，可能需要先安装 `zip`（例如，通过运行 `sudo apt-get install zip`）。您的发行版的安装命令可能不同。

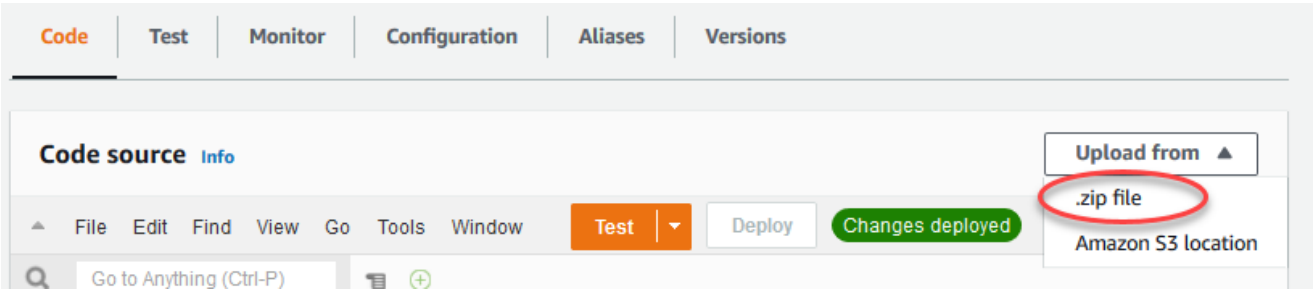
现在您已准备好创建您的 Lambda 函数和上传部署程序包。

5. 打开 Lambda 控制台，选择创建函数。
6. 选择从头开始编写。
7. 将您的函数命名为 **`Greengrass_HelloWorld`** 并设置其余字段，如下所示：

   - 对于 Runtime (运行时)，选择 Python 3.7。
   - 对于权限，请保留默认设置。这将创建一个授予基本 Lambda 权限的执行角色。此角色未由 AWS IoT Greengrass 使用。

   选择 Create function（创建函数）。
8. 上传 Lambda 函数部署软件包：

   a. 在代码选项卡上的代码源下，选择上传自。从下拉列表中选择 .zip 文件。



   b. 选择上传，然后选择您的 `hello_world_python_lambda.zip` 部署包。然后，选择 Save (保存)。

   c. 在函数的代码选项卡中，在运行时设置下选择编辑，然后输入以下值。

      - 对于 Runtime (运行时)，选择 Python 3.7。

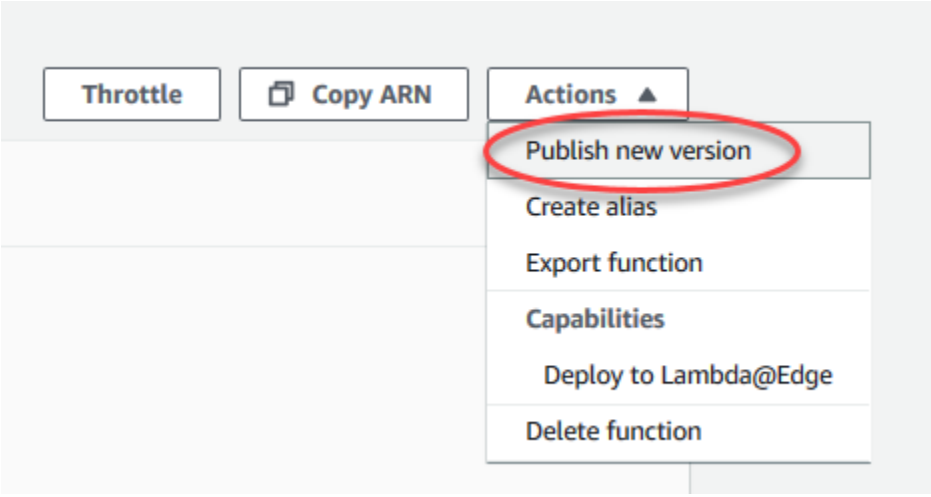
- 对于 Handler (处理程序)，输入 **`greengrassHelloWorld.function_handler`**。

d. 选择 Save（保存）。

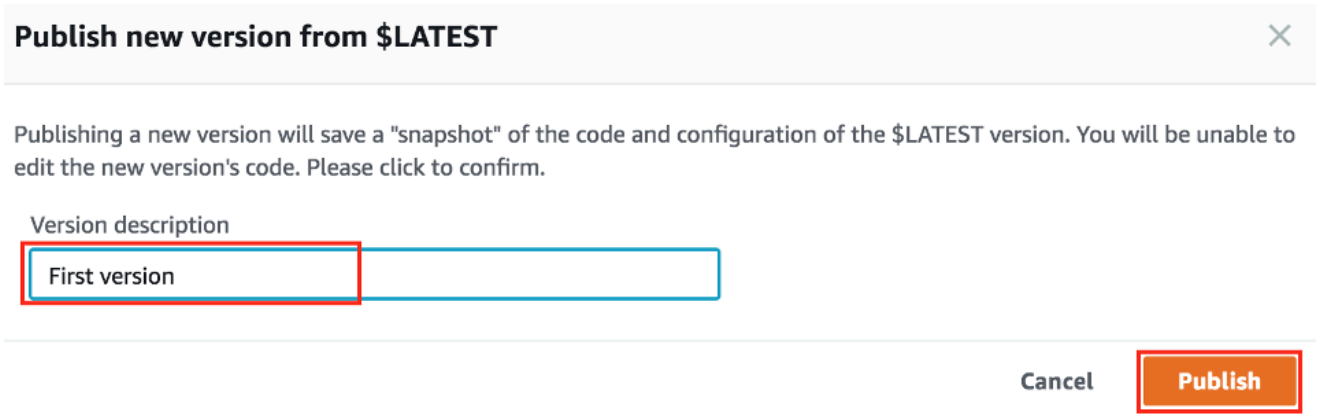
> **Note**
>
> AWS Lambda 控制台上的测试按钮不适用于此功能。AWS IoT Greengrass Core 软件开发工具包不包含在 AWS Lambda 控制台中独立运行 Greengrass Lambda 函数所需的模块。这些模块（例如 `greengrass_common`）是在函数部署到您的 Greengrass 核心之后提供给它们的。

9. 发布 Lambda 函数：

a. 在页面顶部的操作菜单上，选择发布新版本。



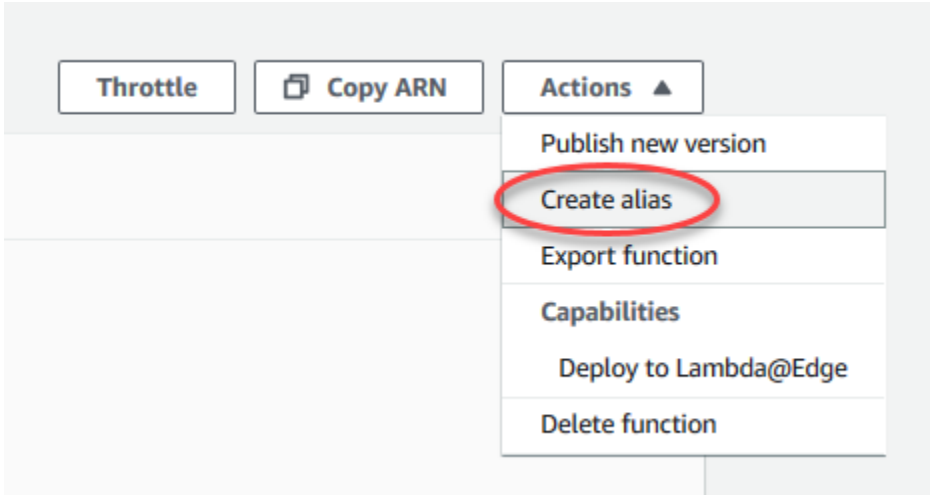
b. 对于 Version description (版本描述)，输入 **`First version`**，然后选择 Publish (发布)。



10. 为 Lambda 函数版本创建别名：

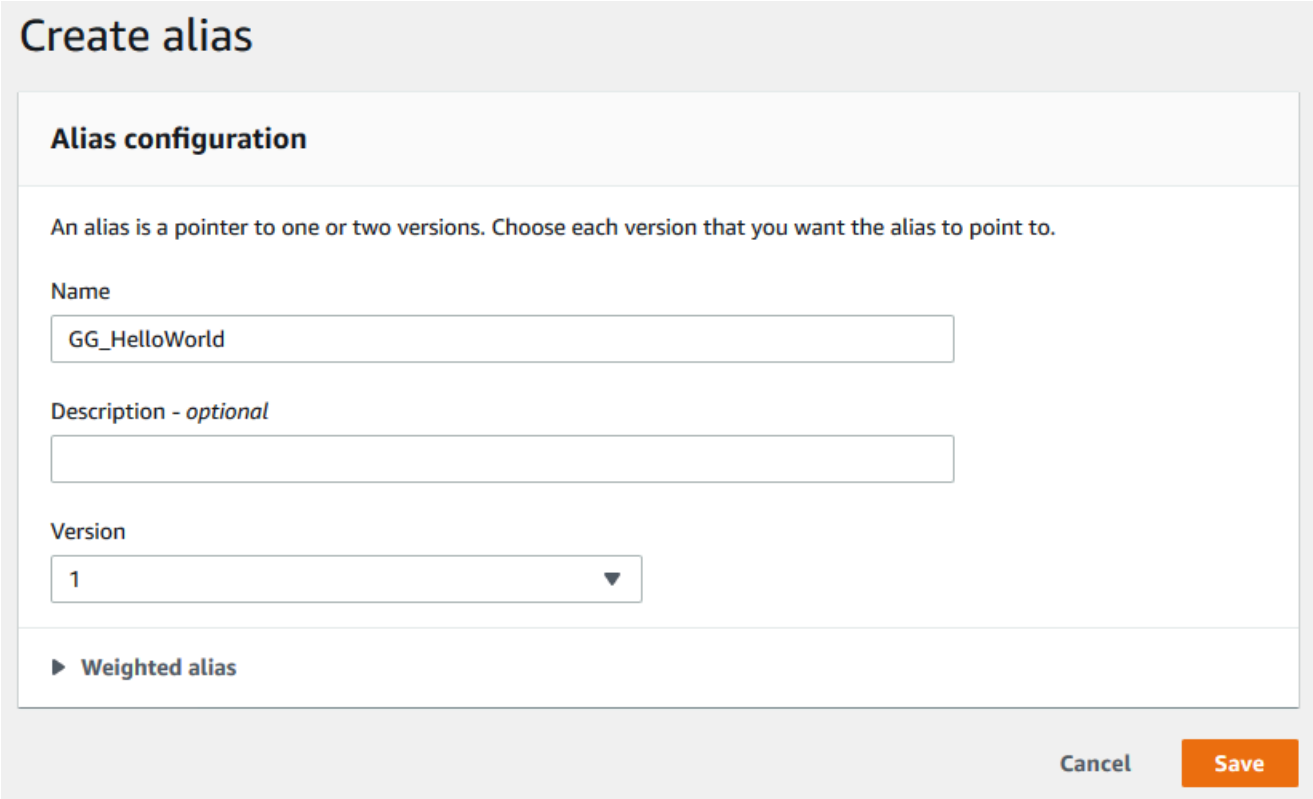
> **Note**
>
> Greengrass 组可以按别名（推荐）或版本引用 Lambda 函数。使用别名，您可以更轻松地管理代码更新，因为您在更新函数代码时，不必更改订阅表或组定义。相反，您只需将别名指向新的函数版本。

a. 从页面顶部的操作菜单中，选择创建别名。



b. 将别名命名为 **`GG_HelloWorld`**，将版本设置为 **`1`**（对应于您刚刚发布的版本），然后选择保存。

> **Note**
>
> AWS IoT Greengrass 不支持 $LATEST 版本的 Lambda 别名。



为 AWS IoT Greengrass 配置 Lambda 函数

现在，您已准备好为 AWS IoT Greengrass 配置您的 Lambda 函数。

在此步骤中，您：

- 使用 AWS IoT 控制台将 Lambda 函数添加到 Greengrass 组。
- 为 Lambda 函数配置特定于组的设置。
- 向该组添加订阅，允许 Lambda 函数向 AWS IoT 发布 MQTT 消息。
- 配置该组的本地日志设置。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 在 Greengrass 组下，选择您在模块 2 中创建的组。
3. 在组配置页面上，选择 Lambda 函数选项卡，然后向下滚动到我的 Lambda 函数部分，然后选择添加 Lambda 函数。
4. 选择您在上一步中创建的 Lambda 函数的名称（Greengrass_HelloWorld，而非别名）。
5. 对于版本，选择 别名：GG_HelloWorld。
6. 在 Lambda 函数配置部分中，进行以下更改：
   - 将系统用户和组设置为使用组默认值。
   - 将 Lambda 函数容器化设置为 使用组默认值。
   - 将超时设置为 25 秒。此 Lambda 函数在每次调用前会休眠 5 秒。
   - 对于已固定，选择 True。

   > **Note**
   >
   > 长时间生存（或固定）的 Lambda 函数在 AWS IoT Greengrass 启动后自动启动并在自己的容器中保持运行。这与按需 Lambda 函数相反，后者在调用时启动，并在没有要运行的任务时停止。有关更多信息，请参阅the section called “生命周期配置”。

7. 选择 添加 Lambda 函数以保存您的更改。有关 Lambda 函数属性的信息，请参阅 the section called “控制 Greengrass Lambda 函数执行”。

   接下来，创建一个允许 Lambda 函数 向 AWS IoT Core 发送 MQTT 消息的订阅。

   Greengrass Lambda 函数可以与以下对象交换 MQTT 消息：

   - Greengrass 组中的设备。
   - 组中的连接器。
   - 组中的其他 Lambda 函数。
   - AWS IoT Core.
   - 本地影子服务。有关更多信息，请参阅the section called “模块 5：与设备影子交互”。

该组使用订阅来控制这些实体可以如何互相通信。订阅提供可预测的交互和一层安全性。

订阅由源、目标和主题组成。源是消息的发起方，目标是消息的目的地。主题允许您筛选从源发送到目标的数据。源或目标可以是 Greengrass 设备、Lambda 函数、连接器、设备影子或 AWS IoT Core。

> **Note**
>
> 订阅是定向的，也就是消息流为特定方向：从源流到目标。要允许双向通信，您必须设置两个订阅。

> **Note**
>
> 目前，订阅主题筛选器不允许在一个主题中使用多个 + 字符。主题筛选器只允许在主题的结尾有一个 # 字符。

`Greengrass_HelloWorld` Lambda 函数只将消息发送到 AWS IoT Core 中的 `hello/world` 主题，因此，您只需要创建一个从 Lambda 函数到 AWS IoT Core 的订阅。您可以在下一步中创建此订阅。

8. 在组配置页面中，选择订阅选项卡，然后选择添加订阅。

   有关向您展示如何使用 AWS CLI 创建订阅的示例，请参阅《AWS CLI 命令参考》中的 create-subscription-definition。

9. 在源类型中，选择 Lambda 函数，对于源，选择 Greengrass_HelloWorld。
10. 对于目标类型，选择服务，对于目标，选择 IoT 云。
11. 对于主题筛选条件字段中，输入 **`hello/world`**，然后选择订阅。
12. 配置组的日志记录设置。在本教程中，您将配置 AWS IoT Greengrass 系统组件和用户定义的 Lambda 函数，以将日志写入核心设备的文件系统。

    a. 在组配置页面上，选择日志选项卡。

    b. 在本地日志配置部分，选择 编辑。

    c. 在编辑本地日志配置对话框中，保留日志级别和存储大小的默认值，然后选择保存。

您可以使用日志解决运行本教程时可能遇到的任何问题。在排查问题时，您可以暂时将日志记录级别更改为调试。有关更多信息，请参阅the section called “访问文件系统日志”。

13. 如果核心设备上未安装 Java 8 运行时，您必须安装它或禁用流管理器。

> **Note**
>
> 此教程不使用流管理器，但它将使用默认情况下启用流管理器的 Default Group creation (默认组创建) 工作流。如果已启用流管理器，但未安装 Java 8，则组部署将失败。有关更多信息，请参阅流管理器要求。

要禁用流管理器，请执行以下操作：

a. 在组设置页面上，选择 Lambda 函数选项卡。

b. 在系统 Lambda 函数部分下，选择流管理器，然后选择编辑。

c. 选择禁用，然后选择保存。

将云配置部署到 Greengrass 核心设备

1. 确保您的 Greengrass 核心设备已连接到 Internet。例如，尝试成功导航到网页。

2. 确保 Greengrass 守护进程正在您的核心设备上运行。在您的核心设备终端中运行以下命令来检查进程守护程序是否正在运行并启动它（如果需要）。

   a. 要检查守护程序是否正在运行，请执行以下操作：

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

   如果输出包含 `root` 的 `/greengrass/ggc/packages/1.11.6/bin/daemon` 条目，则表示守护程序正在运行。

   b. 启动进程守护程序：

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

现在您已准备好将 Lambda 函数和订阅配置部署到您的 Greengrass 核心设备。

3. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
4. 在 Greengrass 组下，选择您在模块 2 中创建的组。
5. 在组配置页面上，选择部署。
6. 在 Lambda 函数选项卡的系统 Lambda 函数部分中，选择 IP 检测器。
7. 选择编辑，然后选择自动检测和覆盖 MQTT 代理端点。这使得设备可以自动获取核心的连接信息，例如 IP 地址、DNS 和端口号。建议使用自动检测，不过 AWS IoT Greengrass 也支持手动指定的终端节点。只有在首次部署组时，系统才会提示您选择发现方法。

第一次部署可能需要几分钟。当部署完成后，您应该在部署页面上的状态列中看到已成功完成：

> **Note**
>
> 部署状态也显示在页面标题上的组名称下方。

有关问题排查帮助，请参阅排查问题。

验证 Lambda 函数是否在核心设备上运行

1. 在 AWS IoT 控制台导航窗格中的测试下方，选择 MQTT 测试客户端。
2. 选择 订阅主题选项卡。
3. 在主题筛选器中输入 **`hello/world`**，然后展开其他配置。
4. 输入在以下字段中列出的信息：
   - 对于服务质量，选择 0。
   - 对于 MQTT payload display (MQTT 负载显示)，选择 Display payloads as strings (以字符串形式显示负载)。
5. 选择 Subscribe。

假定 Lambda 函数正在设备上运行，它将向 `hello/world` 主题发布消息，如下所示：

虽然 Lambda 函数继续向 `hello/world` 主题发送 MQTT 消息，但不要停止 AWS IoT Greengrass 守护程序。其余模块是在假定它正在运行的情况下编写的。

您可以从组中删除函数和订阅：

- 在组配置页面的 Lambda 函数选项卡下，选择要移除的 Lambda 函数，然后选择移除。
- 在组配置页面中的订阅选项卡下，选择订阅，然后选择删除。

在下一个组部署期间，将会从核心中删除函数和订阅。

模块 3（第 2 部分）：AWS IoT Greengrass 的 Lambda 函数

此模块讲述了在 AWS IoT Greengrass 核心上运行的按需和长时间生存的 Lambda 函数之间的差异。

在开始之前，请运行 Greengrass 设备安装 脚本或确保您已完成模块 1、模块 2、和模块 3（第 1 部分）。

本模块应该需要大约 30 分钟才能完成。

主题

- 创建和打包 Lambda 函数
- 为 AWS IoT Greengrass 配置长时间生存的 Lambda 函数
- 测试长时间生存的 Lambda 函数
- 按需测试 Lambda 函数

创建和打包 Lambda 函数

在此步骤中，您：

- 创建包含函数代码和依赖项的 Lambda 函数部署程序包。
- 使用 Lambda 控制台创建 Lambda 和上传部署包。
- 发布 Lambda 函数的版本并创建指向该版本的别名。

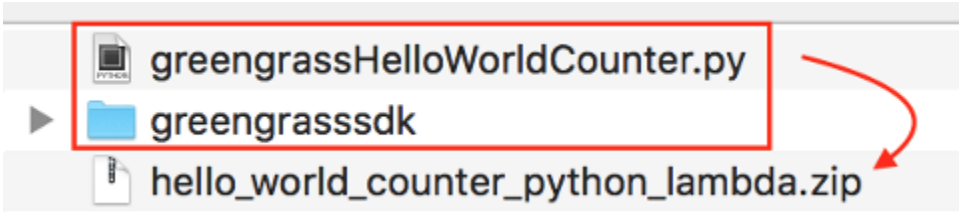
1. 在计算机上，转到您在模块 3-1 的 the section called “创建并打包 Lambda 函数” 中下载并解压缩的适用于 Python 的 AWS IoT Greengrass Core 软件包开发工具。

   本模块中的 Lambda 函数使用：

   - `examples\HelloWorldCounter` 中的 `greengrassHelloWorldCounter.py` 文件。这是您的 Lambda 函数代码。
   - `greengrasssdk` 文件夹。这是软件开发工具包。
2. 创建 Lambda 函数部署程序包：

   a. 将 `greengrasssdk` 文件夹复制到包含 `greengrassHelloWorldCounter.py` 的 `HelloWorldCounter` 文件夹中。

   b. 将 `greengrassHelloWorldCounter.py` 和 `greengrasssdk` 文件夹保存到名为 `hello_world_counter_python_lambda.zip` 的 `zip` 文件。`py` 文件和 `greengrasssdk` 文件夹必须位于该目录的根目录中。

   在已安装 `zip` 的类 UNIX 系统（包括 Mac 终端）上，您可以使用以下命令打包文件和文件夹：

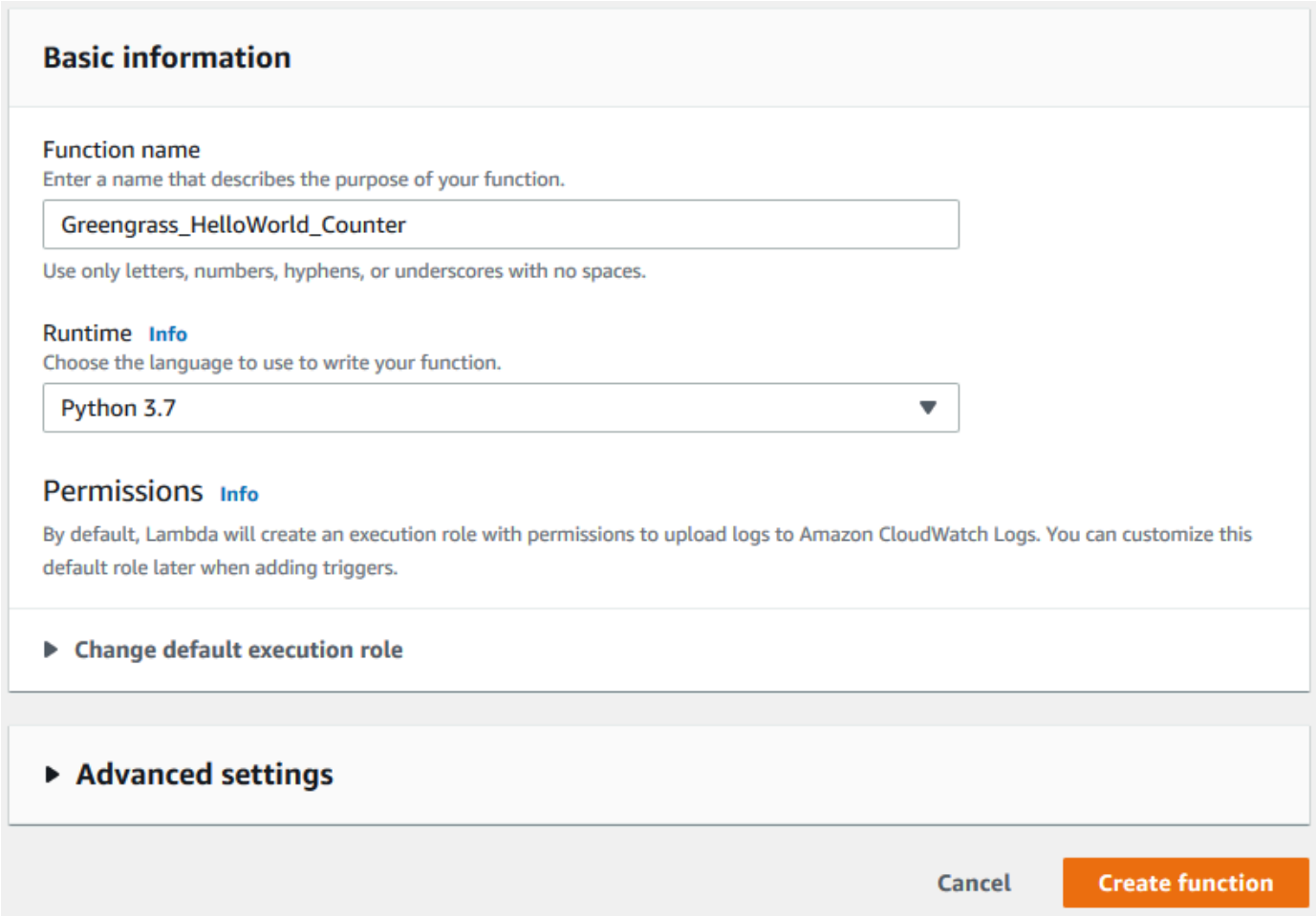
   ```
   zip -r hello_world_counter_python_lambda.zip greengrasssdk
    greengrassHelloWorldCounter.py
   ```

现在您已准备好创建您的 Lambda 函数和上传部署包。

3. 打开 Lambda 控制台，选择创建函数。
4. 选择从头开始创作。
5. 将您的函数命名为 **`Greengrass_HelloWorld_Counter`** 并设置其余字段，如下所示：

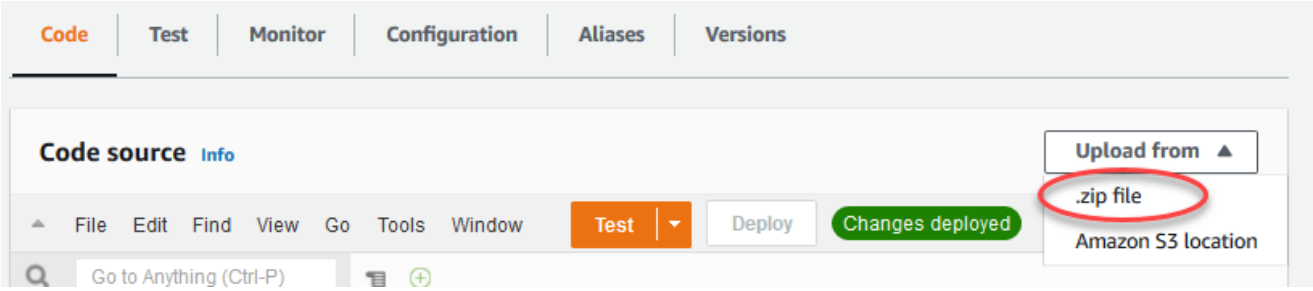
   - 对于运行时系统，选择 Python 3.7。
   - 对于权限，请保留默认设置。这将创建一个授予基本 Lambda 权限的执行角色。此角色不由 AWS IoT Greengrass 使用。或者，您可以重复使用在模块 3-1 中创建的角色。

   选择创建函数。



6. 上传您的 Lambda 函数部署软件包。

   a. 在代码选项卡上的代码源下，选择上传自。从下拉列表中选择 .zip 文件。



b. 选择上传，然后选择您的 `hello_world_counter_python_lambda.zip` 部署包。然后，选择保存。

c. 在函数的代码选项卡中，在运行时设置下选择编辑，然后输入以下值。

- 对于运行时系统，选择 Python 3.7。
- 对于处理程序，输入 **`greengrassHelloWorldCounter.function_handler`**。

d. 选择 Save（保存）。

> **Note**
>
> AWS Lambda 控制台上的测试按钮不适用于此功能。AWS IoT Greengrass Core 软件开发工具包不包含在 AWS Lambda 控制台中独立运行 Greengrass Lambda 函数所需的模块。这些模块（例如 `greengrass_common`）是在函数部署到您的 Greengrass 核心之后提供给它们的。

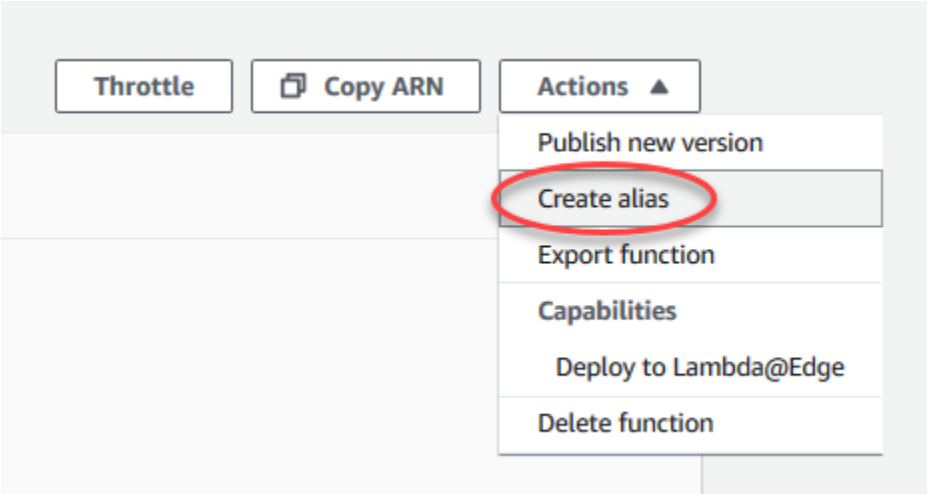
7. 发布函数的第一个版本：

a. 在页面顶部的操作菜单上，选择发布新版本。对于版本描述，输入 **`First version`**。

b. 选择发布。

8. 为函数版本创建别名。

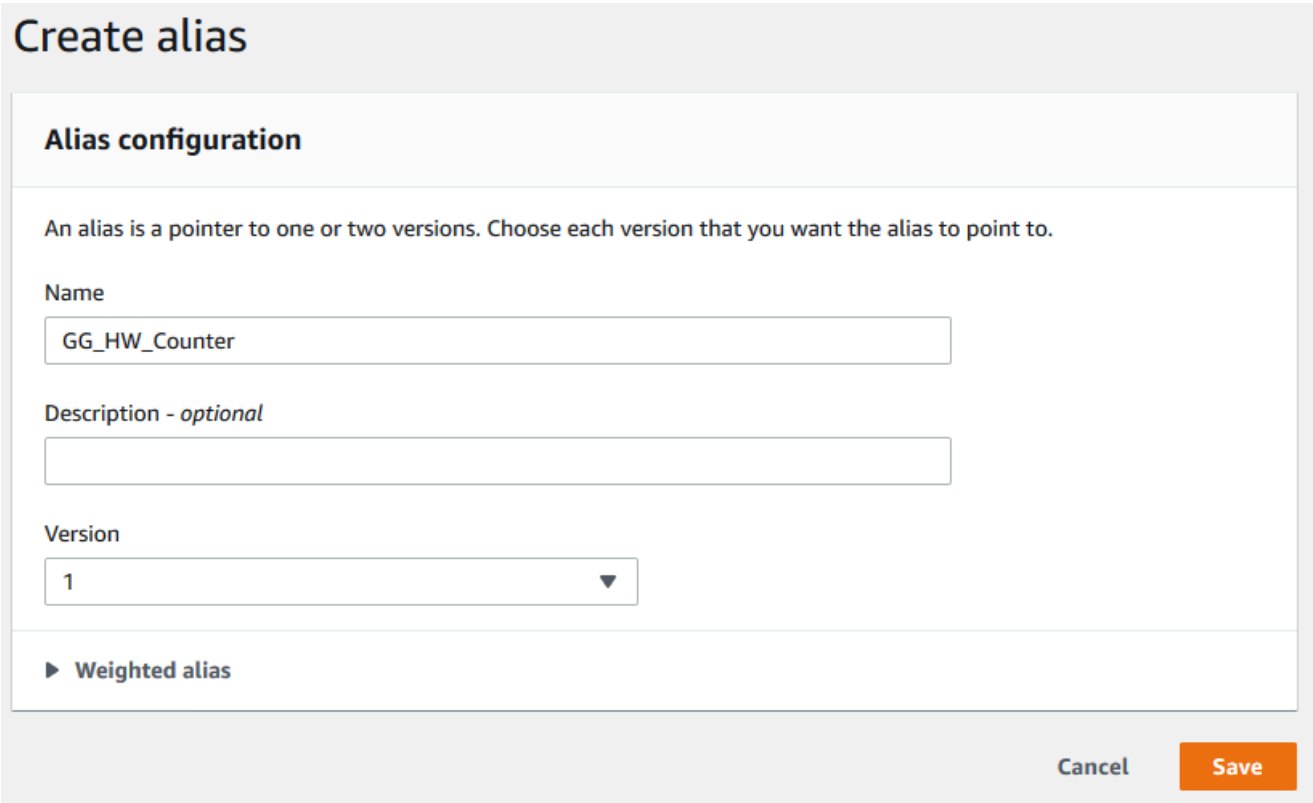
a. 从页面顶部的操作菜单中，选择创建别名。



b. 对于名称，输入 **GG_HW_Counter**。

c. 对于版本，选择 1。

d. 选择 Save（保存）。



别名可为您的Lambda 函数创建一个单一实体，可供 Greengrass 设备订阅。这样，每次修改函数后，您就不必使用新的 Lambda 函数版本号更新订阅。

为 AWS IoT Greengrass 配置长时间生存的 Lambda 函数

现在，您已准备好为 AWS IoT Greengrass 配置您的 Lambda 函数。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 在 Greengrass 组下，选择您在模块 2 中创建的组。
3. 在组配置页面上，选择 Lambda 函数选项卡，然后在我的 Lambda 函数下选择添加。
4. 对于 Lambda 函数，选择 Greengrass_HelloWorld_Counter。
5. 对于 Lambda 函数版本，选择您发布的版本的别名。
6. 对于超时（秒），输入 **25**。此 Lambda 函数在每次调用前会休眠 20 秒。
7. 对于已固定，选择 True。
8. 保留所有其他字段的默认值，然后选择添加 Lambda 函数。

测试长时间生存的 Lambda 函数

长时间生存的 Lambda 函数在 AWS IoT Greengrass 核心启动时自动启动（并在单个容器/沙盒中运行）。保留在函数处理程序外部定义的所有变量和预处理逻辑，用于函数处理程序的每次调用。函数处理程序的多次调用将排队，直到执行完前面的调用。

此模块中使用的 `greengrassHelloWorldCounter.py` 代码定义了函数处理程序外部的 `my_counter` 变量。

> **Note**
>
> 您可以在 AWS Lambda 控制台或 GitHub 上的适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包中查看该代码。

在该步骤中，您将创建允许 Lambda 函数和 AWS IoT 交换 MQTT 消息的订阅。然后，部署组并测试函数。

1. 在组配置页面上，选择 订阅，然后选择添加。
2. 在 源类型 下，选择 Lambda 函数，然后选择 Greengrass_HelloWorld_Counter。
3. 在目标类型下，选择服务，选择 IoT 云。
4. 对于 Topic filter (主题筛选条件)，输入 **`hello/world/counter`**。

5. 选择 Create subscription（创建订阅）。

   此单个订阅只有一个方向：从 `Greengrass_HelloWorld_Counter` Lambda 函数到 AWS IoT。要从云中调用（或触发）此 Lambda 函数，您必须创建反方向的订阅。

6. 按照步骤 1 至 5 添加另一个订阅，它使用以下值。此订阅允许 Lambda 函数从 AWS IoT 接收消息。当您从调用该函数的 AWS IoT 控制台发送消息时，您可以使用此订阅。

   - 对于源，选择服务，然后选择 IoT 云。
   - 对于目标，选择 Lambda 函数，然后选择 Greengrass_HelloWorld_Counter。
   - 对于主题筛选条件，输入 **`hello/world/counter/trigger`**。

   本主题筛选条件中使用了 `/trigger` 扩展，因为您创建了两个订阅，不希望它们互相干扰。

7. 确保 Greengrass 进程守护程序正在运行，如 将云配置部署到核心设备 中所述。
8. 在组配置页面上，选择部署。
9. 在您的部署完成后，返回到 AWS IoT 控制台主页并选择 测试。
10. 配置以下字段：

    - 对于 Subscription topic (订阅主题)，输入 **`hello/world/counter`**。
    - 对于服务质量，选择 0。
    - 对于 MQTT payload display (MQTT 负载显示)，选择 Display payloads as strings (以字符串形式显示负载)。

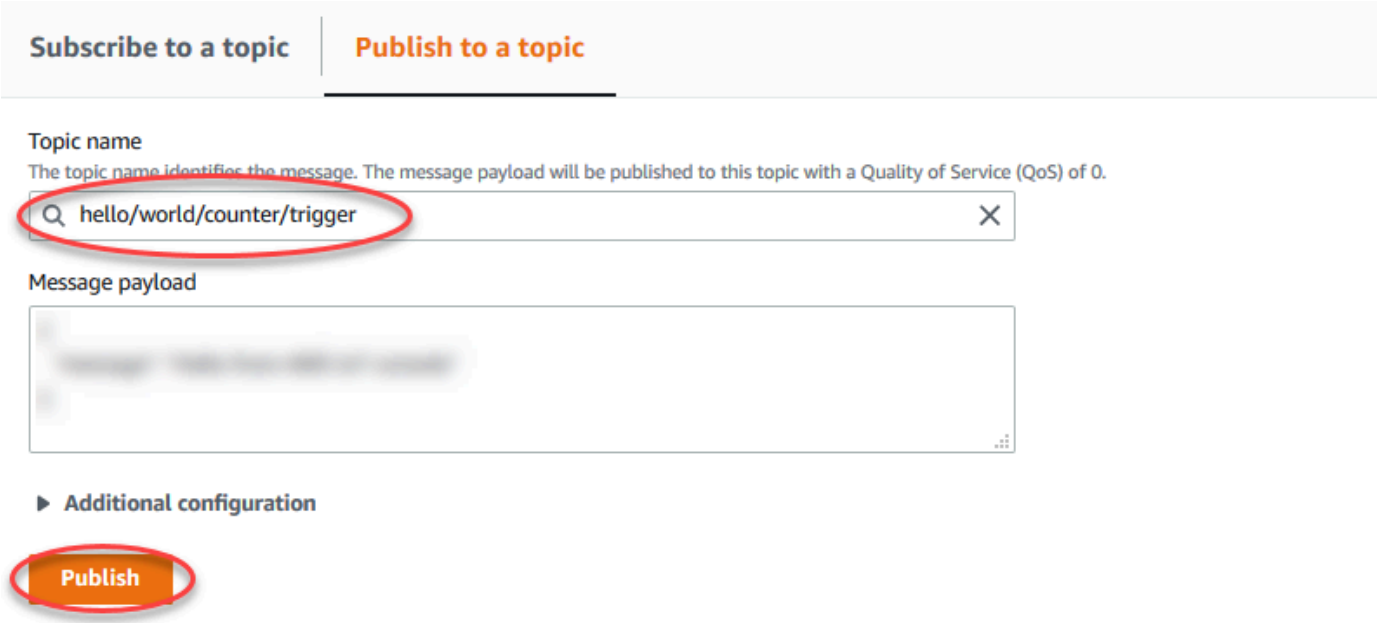
11. 选择 Subscribe。

    与此模块的第 1 部分不同，在您订阅 `hello/world/counter` 之后，您不应看到任何消息。这是因为发布到 `hello/world/counter` 主题的 `greengrassHelloWorldCounter.py` 代码位于函数处理程序中，该函数处理程序仅在调用函数时才运行。

    在此模块中，您配置了 `Greengrass_HelloWorld_Counter` Lambda 函数，当其收到以 `hello/world/counter/trigger` 为主题的 MQTT 消息时将被调用。

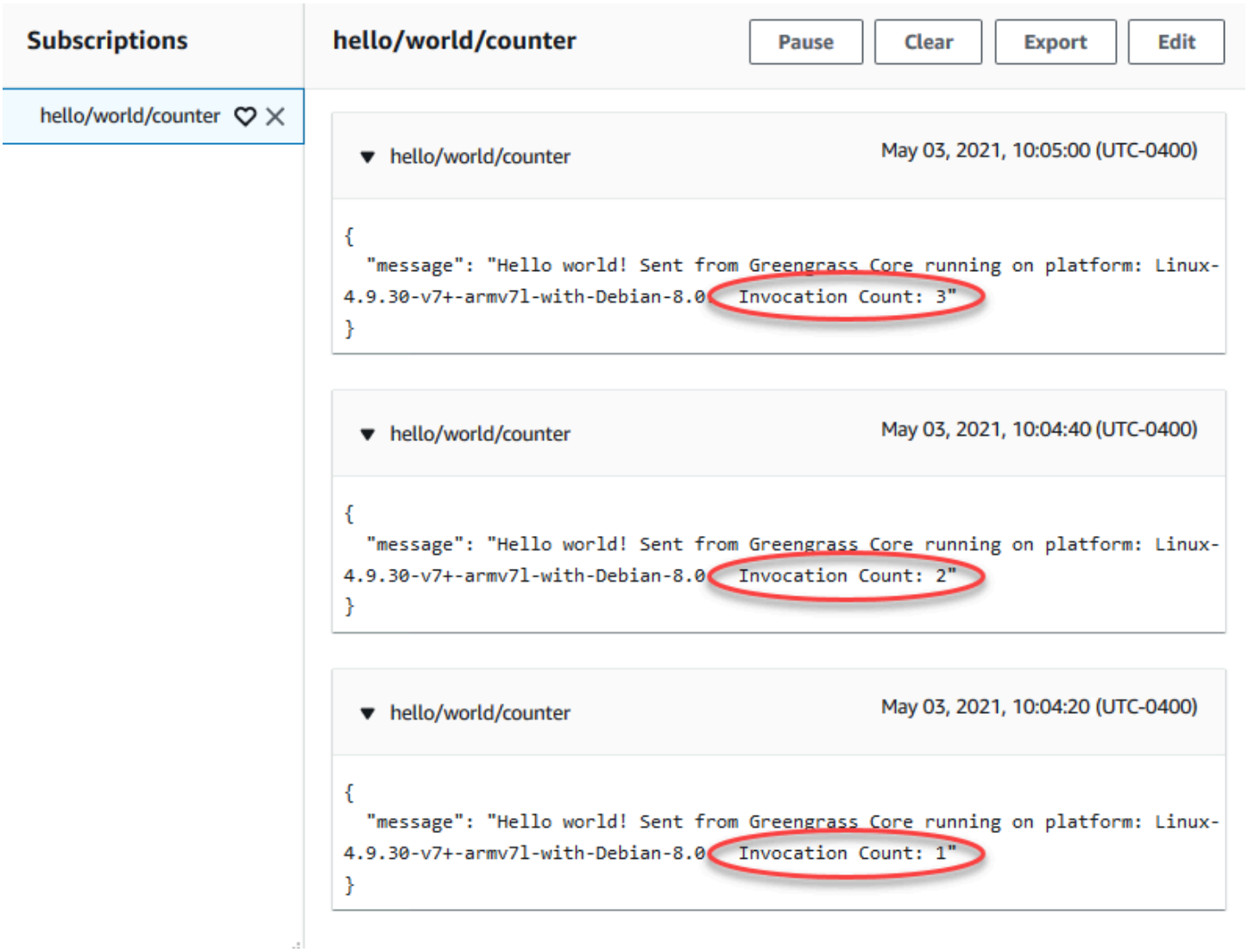
    Greengrass_HelloWorld_Counter 到 IoT Cloud 的订阅允许该函数以 `hello/world/counter` 为主题向 AWS IoT 发送消息。IoT 云到 Greengrass_HelloWorld_Counter 的订阅允许 AWS IoT 以 `hello/world/counter/trigger` 为主题向该函数发送消息。

12. 要测试较长的生命周期，请通过向 `hello/world/counter/trigger` 主题发布消息来调用 Lambda 函数。您可以使用默认消息。



> **Note**
>
> 该 `Greengrass_HelloWorld_Counter` 函数会忽略所接收消息的内容。它只运行 `function_handler` 中的代码，该代码会向 `hello/world/counter` 主题发送消息。您可以从 GitHub 的 适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包查看此代码。

每次将消息发布到 `hello/world/counter/trigger` 主题时，`my_counter` 变量都会递增。此调用计数显示在从 Lambda 函数发送的消息中。由于函数处理程序包含 20 秒休眠周期 (`time.sleep(20)`)，反复触发处理程序队列将需要排队等候来自 AWS IoT Greengrass 核心的响应。



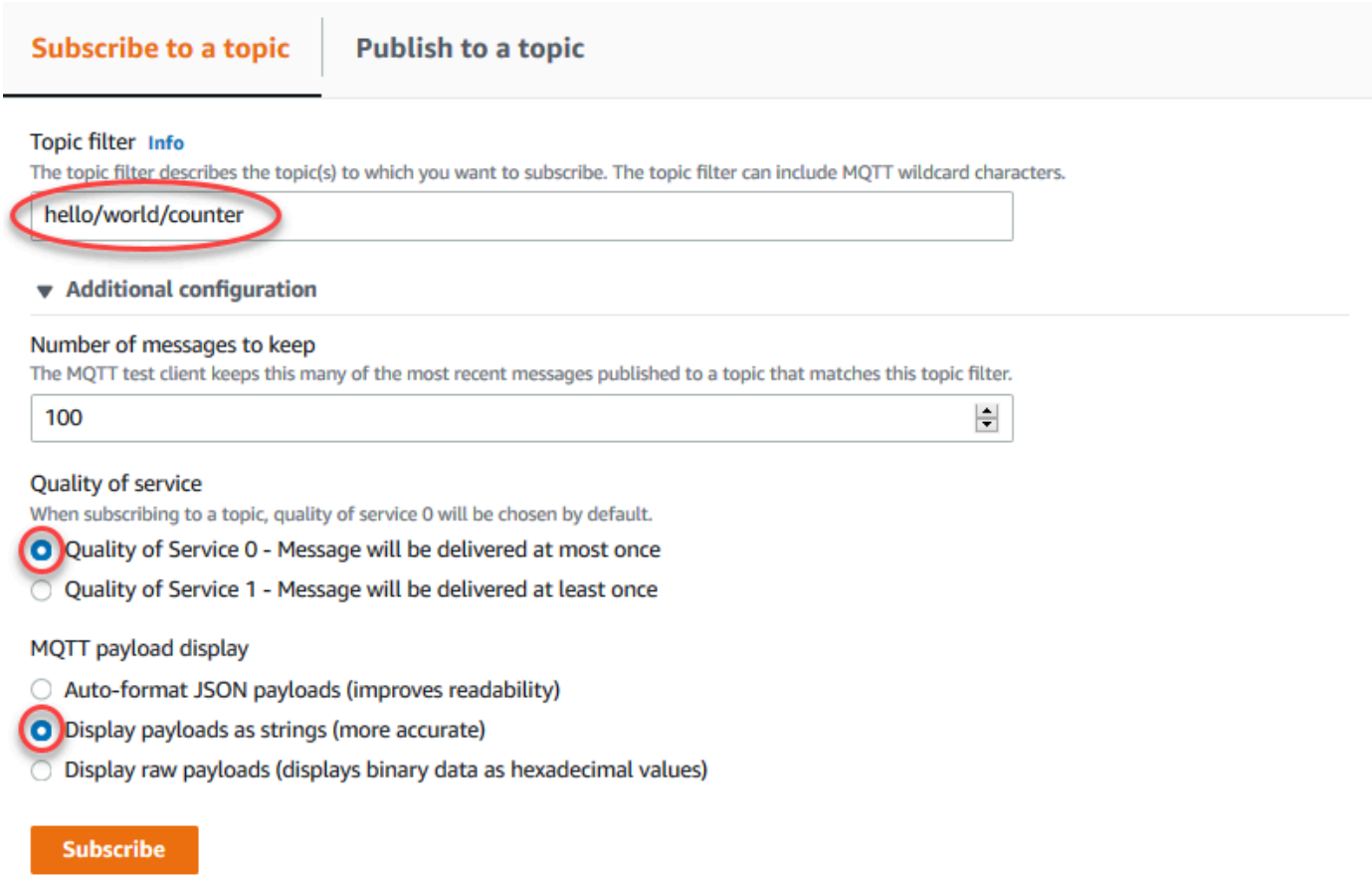
按需测试 Lambda 函数

按需 Lambda 函数在功能上与基于云的 AWS Lambda 函数类似。按需 Lambda 函数的多次调用可以并行运行。Lambda 函数调用创建单独的容器以处理调用，或者在资源允许时重复使用现有的容器。在创建容器时，不会保留在函数处理程序外部定义的任何变量或预处理。

1. 在组配置页面上，选择Lambda 函数选项卡。
2. 在我的 Lambda 函数下，选择 Greengrass_HelloWorld_Counter Lambda 函数。
3. 在 Greengrass_HelloWorld_Counter 详细信息页上，选择编辑。
4. 在已固定中，选择 False，然后选择保存。
5. 在组配置页面上，选择部署。
6. 在您的部署完成后，返回到 AWS IoT 控制台主页并选择 测试。

7. 配置以下字段：

   - 对于 Subscription topic (订阅主题)，输入 **`hello/world/counter`**。
   - 对于服务质量，选择 0。
   - 对于 MQTT payload display (MQTT 负载显示)，选择 Display payloads as strings (以字符串形式显示负载)。

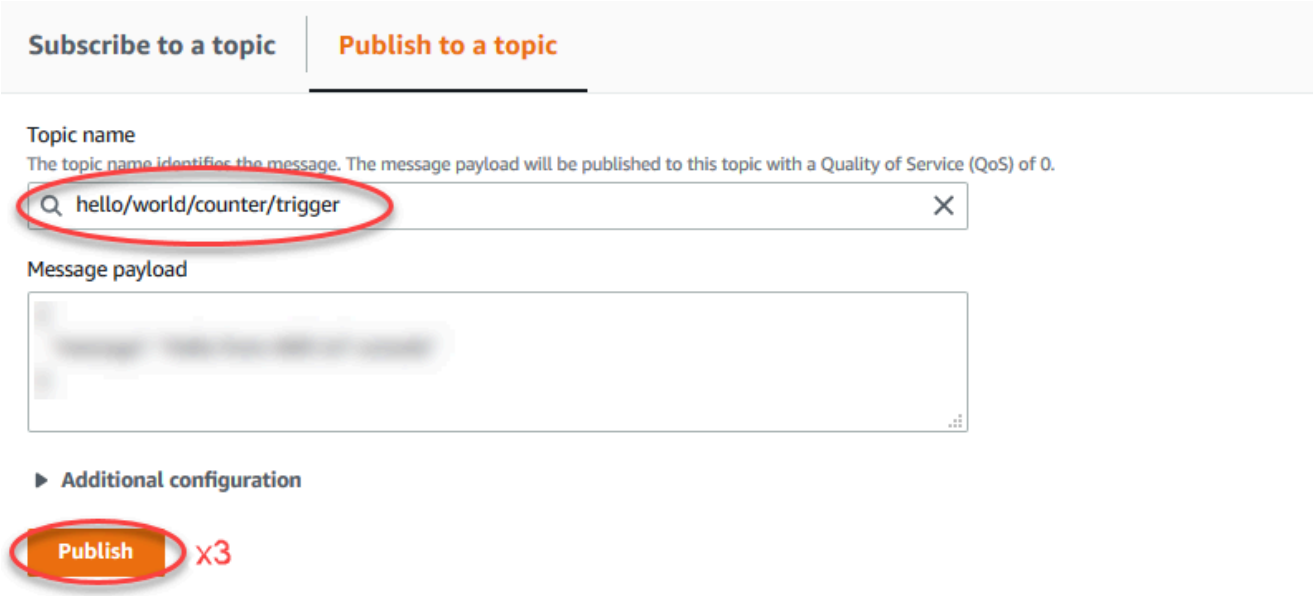


8. 选择 Subscribe。

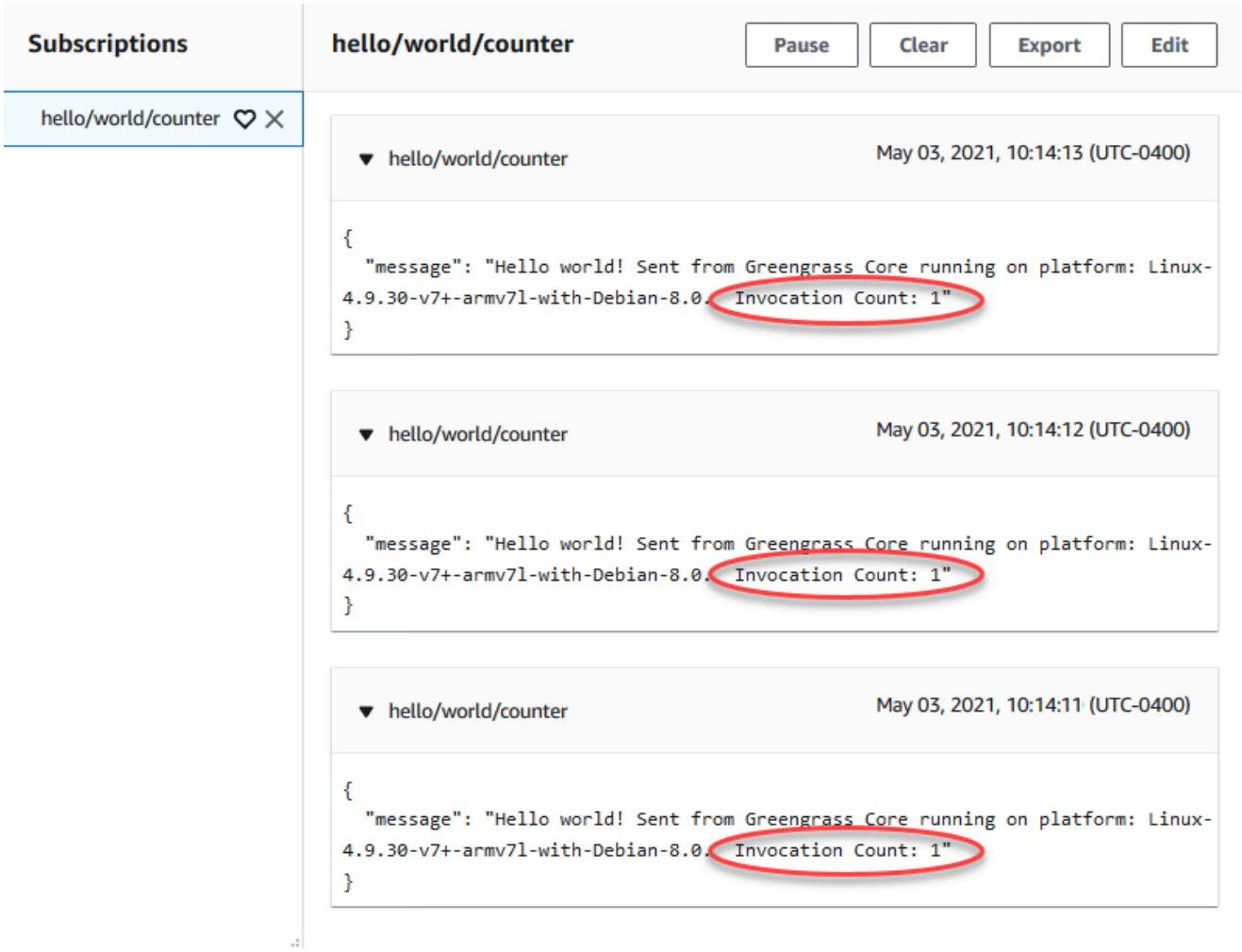
> **Note**
>
> 在您订阅之后，您不应看到任何消息。

9. 要测试按需生命周期，请通过向 `hello/world/counter/trigger` 主题发布消息来调用该函数。您可以使用默认消息。

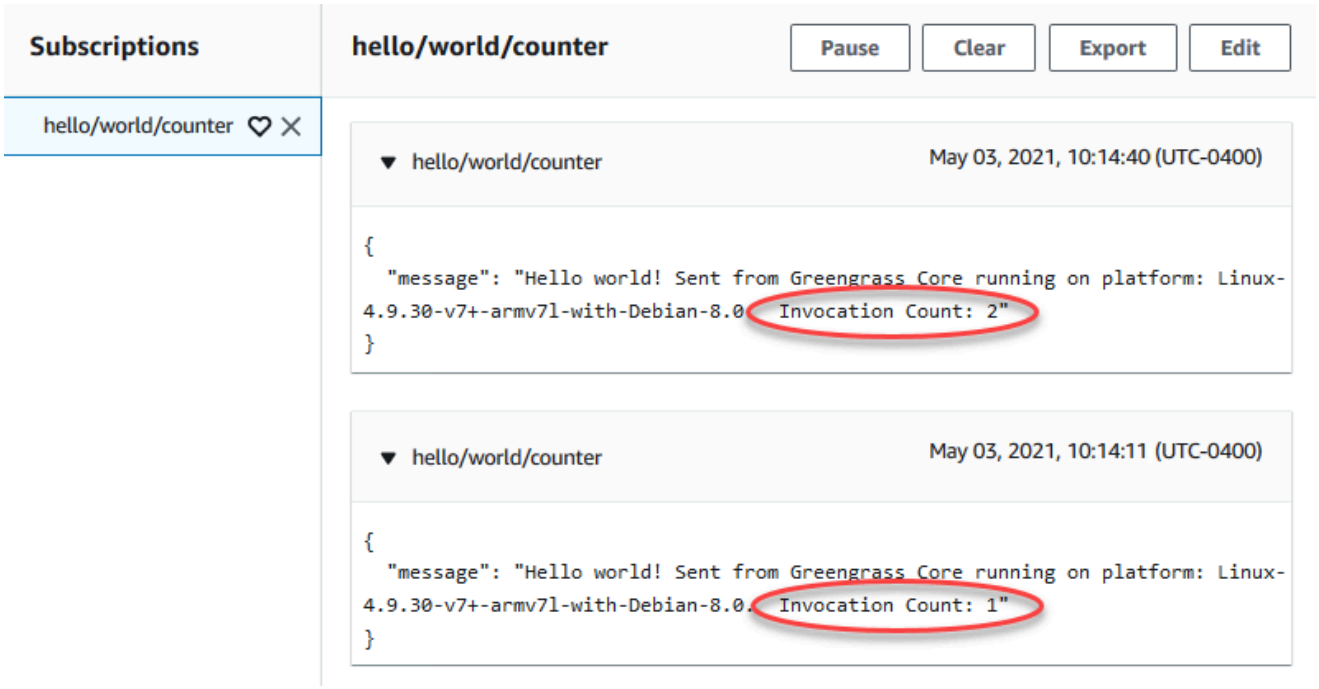
   a. 快速选择 发布 三次，每次都在五秒钟内按下按钮。



每次发布都会调用函数处理程序并为每次调用创建一个容器。在您三次触发该函数时，不会为触发递增调用计数，因为每个按需 Lambda 函数都有自己的容器/沙盒。



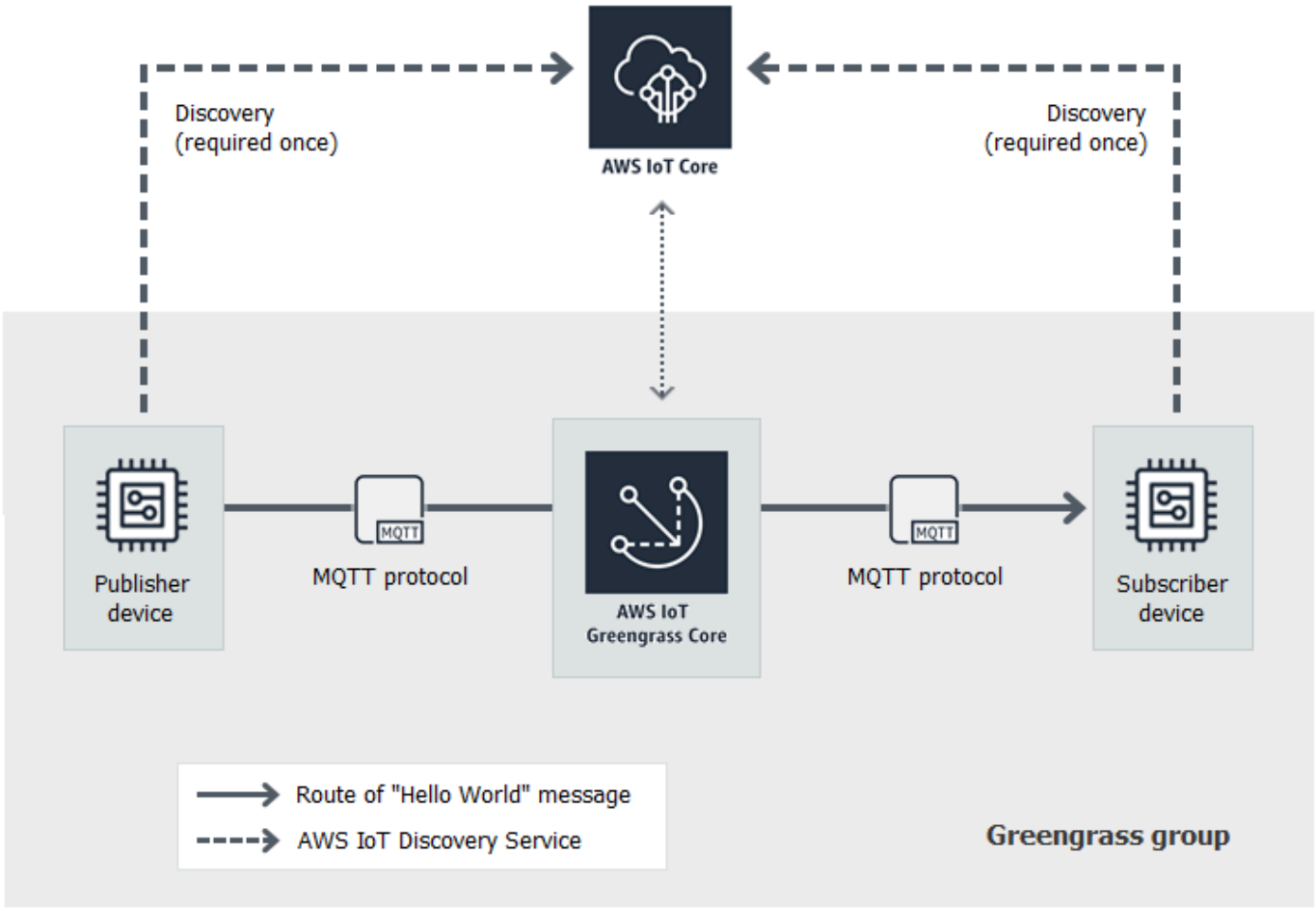
b. 等待大约 30 秒，然后选择 Publish to topic (发布到主题)。调用计数应递增到 2。这说明重复使用了从之前调用创建的容器，并且存储了函数处理程序外部的预处理变量。



您现在应该了解了可在 AWS IoT Greengrass 核心上运行的两种类型的 Lambda 函数。下一模块（模块 4）将说明本地 IoT 设备如何在 AWS IoT Greengrass 组中进行交互。

模块 4：在 AWS IoT Greengrass 组中与客户端设备交互

本模块向您展示本地物联网设备（称为客户端设备或设备）如何连接到 AWS IoT Greengrass 核心设备并与之通信。连接到 AWS IoT Greengrass 核心的客户端设备是 AWS IoT Greengrass 组的一部分，可以参与 AWS IoT Greengrass 编程范例。在此模块中，一个客户端设备将向 Greengrass 组中的另一个客户端设备发送“Hello World”消息：



在开始之前，请运行 Greengrass 设备安装程序脚本或完成模块 1 和模块 2。此模块将创建两个模拟客户端设备。您无需其他组件或设备。

本模块应该需要不到 30 分钟即可完成。

主题

- 在 AWS IoT Greengrass 组中创建客户端设备
- 配置订阅
- 安装适用于 Python 的 AWS IoT Device SDK
- 测试通信

在 AWS IoT Greengrass 组中创建客户端设备

在此步骤中，您将两个客户端设备添加到您的 Greengrass 组。此过程包括注册设备作为 AWS IoT 事物以及配置证书和密钥以允许它们连接到 AWS IoT Greengrass。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择目标组。
3. 在组配置页面中，选择客户端设备，然后选择关联。
4. 在将客户端设备与此组关联模态中，选择新建 AWS IoT 事物。

   此时将在新选项卡中打开创建事物页面。
5. 在 创建事物页面上，选择 创建单个事物，然后选择 下一步。
6. 在指定事物属性页面上，将此客户端设备注册为 **`HelloWorld_Publisher`**，然后选择下一步。
7. 在配置设备证书页面上，选择下一步。
8. 在 将策略附加到证书 页面上，执行下列操作之一：
   - 选择授予客户端设备所需权限的现有策略，然后选择创建事物。

     这将打开一个模态，供您下载设备用于连接到 AWS Cloud 和核心的证书和密钥。
   - 创建并附加一个新策略，以授予客户端权限。执行以下操作：

     a. 选择 Create policy（创建策略）。

        此时将在新选项卡中打开创建策略页面。

     b. 在创建策略页面上，执行以下操作：

        i. 在策略名称中，输入一个名称来描述该策略，例如 **`GreengrassV1ClientDevicePolicy`**。

        ii. 在策略语句选项卡的策略文档下，选择 JSON。

        iii. 输入以下策略文档。此策略允许客户端设备发现 Greengrass 核心并就所有 MQTT 主题进行通信。有关如何限制此策略访问权限的信息，请参阅[AWS IoT Greengrass 的设备身份验证和授权]。

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

```
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. 选择 Create（创建）以创建策略。

c. 返回到打开了将策略附加到证书页面的浏览器标签页中。执行以下操作：

i. 在策略列表中，选择您创建的策略，例如 GreengrassV1ClientDevicePolicy。

如果没有看到策略，请选择刷新按钮。

ii. 选择 Create thing（创建事物）。

这将打开一个模态，供您下载设备用于连接到 AWS Cloud 和核心的证书和密钥。

9. 在下载证书和密钥模态中，下载设备的证书。

> **Important**
> 请先下载安全资源，然后再选择 完成。

执行以下操作：

a. 在设备证书中，选择下载以下载该设备证书。

b. 在公钥文件中，选择下载以下载该证书的公钥。

c. 在私钥文件中，选择下载以下载该证书的私钥文件。

d. 查看《AWS IoT 开发人员指南》中的服务器身份验证，然后选择相应的根 CA 证书。我们建议您使用 Amazon Trust Services (ATS) 端点和 ATS 根 CA 证书。在根 CA 证书下，针对根 CA 证书选择下载。

e. 选择完成。

记下设备证书和密钥文件名中常见的证书 ID。稍后您将需要用到它。

10. 返回到打开了将客户端设备与此组关联模态的浏览器标签页。执行以下操作：

a. 在 AWS IoT 事物名称中，选择您创建的 HelloWorld_Publisher 事物。

如果没有看到事物，请选择刷新按钮。

b. 选择 Associate。

11. 重复步骤第 到 10 步，以在组中添加第二个客户端设备。

将此客户端设备命名为 **`HelloWorld_Subscriber`**。将客户端设备的证书和密钥下载到您的计算机中。同样，记下 HelloWorld_Subscriber 设备的文件名中的常见证书 ID。

现在，您的 Greengrass 组中应该有两个客户端设备：

- HelloWorld_Publisher
- HelloWorld_Subscriber

12. 在您的计算机上创建一个文件夹来保存这些客户端设备的安全凭证。将证书和密钥复制到此文件夹。

配置订阅

在本步骤中，您将使 HelloWorld_Publisher 客户端设备能够向 HelloWorld_Subscriber 客户端设备发送 MQTT 消息。

1. 在组配置页面中，选择 订阅选项卡，然后选择 添加。
2. 在创建订阅页中，执行以下操作以配置订阅：

a. 对于来源类型，选择客户端设备，然后选择 HelloWorld_Publisher。

b. 在目标类型下面，选择客户端设备，然后选择 HelloWorld_Subscriber。

c. 对于 Topic filter (主题筛选条件)，输入 **hello/world/pubsub**。

> **Note**
>
> 您可以删除前面的模块中的订阅。在组的订阅页面上，选择要删除的订阅，然后选择删除。

d. 选择 Create subscription（创建订阅）。

3. 确保启用了自动检测，以便 Greengrass 核心可以发布其 IP 地址的列表。客户端设备使用此信息来发现核心。执行以下操作：

   a. 在组配置页面上，选择Lambda 函数选项卡。

   b. 在系统 Lambda 函数下，选择 IP 检测器，然后选择编辑。

   c. 在编辑 IP 检测器设置中，选择自动检测和覆盖 MQTT 代理端点，然后选择保存。

4. 确保 Greengrass 进程守护程序正在运行，如 将云配置部署到核心设备 中所述。

5. 在组配置页面上，选择部署。

部署状态显示在页面标题上的组名称下方。要查看部署详细信息，请选择部署选项卡。

安装适用于 Python 的 AWS IoT Device SDK

客户端设备可以使用适用于 Python 的 AWS IoT Device SDK 来与 AWS IoT 和 AWS IoT Greengrass 核心设备通信（使用 Python 编程语言）。有关更多信息（包括要求），请参阅 GitHub 上的适用于 Python 的 AWS IoT Device SDK 自述文件。

在此步骤中，您将安装软件开发工具包并获取计算机上模拟客户端设备使用的 `basicDiscovery.py` 示例函数。

1. 要将软件开发工具包与所有必需的组件一起安装到您的计算机上，请选择您的操作系统：

   Windows

   1. 打开提升的命令提示符并运行以下命令：

      ```
      python --version
      ```

如果没有返回版本信息，或者 Python 2 的版本号小于 2.7 或 Python 3 的版本号小于 3.3，则按照下载 Python 中的说明安装 Python 2.7+ 或 Python 3.3+。有关更多信息，请参阅在 Windows 上使用 Python。

2. 将适用于 Python 的 AWS IoT Device SDK下载为 `zip` 文件并将其解压缩到您计算机上的适当位置。

   请记下其中包含 `setup.py` 文件的解压缩 `aws-iot-device-sdk-python-master` 文件夹的文件路径：在下一个步骤中，此文件路径将由 *`path-to-SDK-folder`* 指示。

3. 从提升的命令提示符运行以下命令：

```
cd path-to-SDK-folder
python setup.py install
```

macOS

1. 打开终端窗口，并运行以下命令：

```
python --version
```

   如果没有返回版本信息，或者 Python 2 的版本号小于 2.7 或 Python 3 的版本号小于 3.3，则按照下载 Python 中的说明安装 Python 2.7+ 或 Python 3.3+。有关更多信息，请参阅在 Macintosh 上使用 Python。

2. 在终端窗口中，运行以下命令以确定 OpenSSL 版本：

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

   记下 OpenSSL 版本值。

> **Note**
>
> 如果运行的是 Python 3，请使用 print(ssl.OPENSSL_VERSION)。

要关闭 Python shell，请运行以下命令：

```
>>>exit()
```

如果 OpenSSL 版本为 1.0.1 或更高版本，请跳至步骤 c。否则，请按照以下步骤操作：

- 在终端窗口中，运行以下命令以确定计算机正在使用简单 Python 版本管理：

  ```
  which pyenv
  ```

如果返回文件路径，则选择使用 **pyenv** 选项卡。如果未返回任何内容，则选择未使用 **pyenv** 选项卡。

Using pyenv

1. 请参阅用于 Mac OS X 的 Python 版本（或类似内容），以确定最新的稳定 Python 版本。在以下示例中，该值由 *latest-Python-version* 指示。
2. 在终端窗口中，运行以下命令：

   ```
   pyenv install latest-Python-version
   pyenv global latest-Python-version
   ```

   例如，如果 Python 2 的最新版本是 2.7.14，则这些命令为：

   ```
   pyenv install 2.7.14
   pyenv global 2.7.14
   ```

3. 关闭并重新打开终端窗口，然后运行以下命令：

   ```
   python
   >>>import ssl
   >>>print ssl.OPENSSL_VERSION
   ```

   OpenSSL 版本应至少为 1.0.1。如果版本低于 1.0.1，则更新失败。检查 pyenv install 和 pyenv global 命令中使用的 Python 版本值，然后重试。

4. 运行以下命令以退出 Python shell：

   ```
   exit()
   ```

Not using pyenv

1. 从终端窗口中运行以下命令以确定 brew 是否已安装：

```
which brew
```

如果未返回文件路径，请安装 brew，如下所示：

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
Homebrew/install/master/install)"
```

> **Note**
>
> 按照安装提示进行操作。下载 Xcode 命令行工具可能需要一些时间。

2. 运行以下命令：

```
brew update
brew install openssl
brew install python@2
```

适用于 Python 的 AWS IoT Device SDK 需要使用 Python 可执行文件编译的 OpenSSL 版本 1.0.1（或更高版本）。brew install python 命令会安装满足此要求的 python2 可执行文件。该 python2 可执行文件安装在 /usr/local/bin 目录中，该目录应作为 PATH 环境变量的一部分。要确认这一点，请运行以下命令：

```
python2 --version
```

如果提供了 python2 版本信息，请跳到下一步。否则，通过将以下行附加到您的 Shell 配置文件，永久地将 /usr/local/bin 路径添加到您的 PATH 环境变量：

```
export PATH="/usr/local/bin:$PATH"
```

例如，如果您使用的是 .bash_profile 或您还没有 Shell 配置文件，则从终端窗口中运行以下命令：

```
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
```

接下来，对您的 Shell 配置文件执行 source 命令并确认 `python2 --version` 提供版本信息。例如，如果您正在使用 `.bash_profile`，则运行以下命令：

```
source ~/.bash_profile
python2 --version
```

应返回 python2 版本信息。

3. 将以下行附加到您的 Shell 配置文件：

```
alias python="python2"
```

例如，如果您使用的是 `.bash_profile` 或您还没有 Shell 配置文件，则运行以下命令：

```
echo 'alias python="python2"' >> ~/.bash_profile
```

4. 接下来，对您的 Shell 配置文件执行 source 命令。例如，如果您正在使用 `.bash_profile`，则运行以下命令：

```
source ~/.bash_profile
```

调用 python 命令将运行 Python 可执行文件 (python2)，其中包含所需的 OpenSSL 版本。

5. 运行以下命令：

```
python
 import ssl
 print ssl.OPENSSL_VERSION
```

OpenSSL 版本应为 1.0.1 或更高版本。

6. 要退出 Python shell，请运行以下命令：

```
 exit()
```

3. 运行以下命令，安装 适用于 Python 的 AWS IoT Device SDK：

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

UNIX-like system

1. 从您的 终端窗口中运行以下命令：

```
python --version
```

如果没有返回版本信息，或者 Python 2 的版本号小于 2.7 或 Python 3 的版本号小于 3.3，则按照下载 Python 中的说明安装 Python 2.7+ 或 Python 3.3+。有关更多信息，请参阅在 Unix 平台上使用 Python。

2. 在终端中，运行以下命令以确定 OpenSSL 版本：

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

记下 OpenSSL 版本值。

> **Note**
>
> 如果运行的是 Python 3，请使用 print(ssl.OPENSSL_VERSION)。

要关闭 Python shell，请运行以下命令：

```
exit()
```

如果 OpenSSL 版本为 1.0.1 或更高版本，请跳至下一步。否则，运行命令以便为您的发行版更新 OpenSSL（例如，`sudo yum update openssl`、`sudo apt-get update` 等）。

通过运行以下命令来确认 OpenSSL 版本为 1.0.1 或更高版本：

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
>>>exit()
```

3. 运行以下命令，安装 适用于 Python 的 AWS IoT Device SDK：

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

2. 安装 适用于 Python 的 AWS IoT Device SDK 后，请导航到 `samples` 文件夹并打开 `greengrass` 文件夹。

   在本教程中，您将复制 `basicDiscovery.py` 示例函数，该函数会使用您在 the section called “在 AWS IoT Greengrass 组中创建客户端设备” 中下载的证书和密钥。

3. 将 `basicDiscovery.py` 复制到包含 HelloWorld_Publisher 和 HelloWorld_Subscriber 设备证书和密钥的文件夹。

测试通信

1. 确保您的计算机和 AWS IoT Greengrass 核心设备使用相同的网络连接到互联网。

   a. 在 AWS IoT Greengrass 核心设备上，运行以下命令以查找其 IP 地址。

   ```
   hostname -I
   ```

   b. 在计算机上，使用核心的 IP 地址运行以下命令。可以使用 Ctrl + C 停止 ping 命令。

   ```
   ping IP-address
   ```

   类似于以下内容的输出表示计算机和 AWS IoT Greengrass 核心设备之间成功通信（丢包 0%）：

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

> **Note**
>
> 如果您无法 ping 正在运行的 EC2 实例 AWS IoT Greengrass，请确保该实例的入站安全组规则允许 ECHO 请求消息的 ICMP 流量。有关更多信息，请参阅 Amazon EC2 用户指南中的向安全组添加规则。
> 在 Windows 主机上，在具有高级安全性应用程序的 Windows 防火墙中，您可能还需要启用一个允许入站回显请求的入站规则（例如，File and Printer Sharing (Echo Request - ICMPv4-In) (文件和打印机共享 (回显请求 - ICMPv4-In)) ），或创建一个。

2. 获取您的 AWS IoT 终端节点。

   a. 从AWS IoT 控制台导航窗格中，选择设置。

   b. 在设备数据端点下，记下 端点的值。您可以使用此值在以下步骤中替换命令中的 *AWS_IOT_ENDPOINT* 占位符。

> **Note**
>
> 确保您的终端节点与证书类型对应。

3. 在您的计算机（不是 AWS IoT Greengrass 核心设备）上，打开两个命令行（终端或命令提示符）窗口。一个窗口代表 HelloWorld _Publisher 客户端设备，另一个窗口代表 HelloWorld _Subscriber 客户端设备。

   执行后，`basicDiscovery.py`尝试收集有关 AWS IoT Greengrass 核心端点位置的信息。在客户端设备发现并成功连接到核心后，将会存储此信息。这将允许未来的消息传递和操作可以在本地执行（无需 Internet 连接）。

> **Note**
>
> 用于 MQTT 连接的客户端 ID 必须与客户端设备的事物名称相匹配。basicDiscovery.py 脚本将 MQTT 连接的客户端 ID 设置为您在运行脚本时指定的事物名称。
> 从包含 basicDiscovery.py 文件的文件夹运行以下命令，以查看详细的脚本使用信息：
>
> ```
> python basicDiscovery.py --help
> ```

4. 在 HelloWorld _Publisher 客户端设备窗口中，运行以下命令。

   - 将 *path-to-certs-folder* 替换为包含证书、密钥和 basicDiscovery.py 的文件夹的路径。
   - 将 *AWS_IOT_ENDPOINT* 替换为您的端点。
   - 将两个*###CertId*实例替换为您的 HelloWorld _Publisher 客户端设备文件名中的证书 ID。

   ```
   cd path-to-certs-folder
   python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
    --cert publisherCertId-certificate.pem.crt --key publisherCertId-private.pem.key
    --thingName HelloWorld_Publisher --topic 'hello/world/pubsub' --mode publish --
   message 'Hello, World! Sent from HelloWorld_Publisher'
   ```

   您应该看到类似于以下内容的输出，其中包括诸如 `Published topic 'hello/world/pubsub': {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}` 的条目。

   > **Note**
   >
   > 如果脚本返回 `error: unrecognized arguments` 消息，请针对 `--topic` 和 `--message` 参数将单引号更改为双引号，并再次运行该命令。
   > 要排查连接问题，您可以尝试使用手动 IP 检测。

```
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}

2017-11-13 21:12:26,296 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:26,297 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:27,301 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}

2017-11-13 21:12:27,302 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:27,303 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:28,305 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}

2017-11-13 21:12:28,306 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
2017-11-13 21:12:28,307 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [puback] event
2017-11-13 21:12:29,310 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 3}
```

5. 在 HelloWorld _Subscriber 客户端设备窗口中，运行以下命令。

   - 将 *path-to-certs-folder* 替换为包含证书、密钥和 basicDiscovery.py 的文件夹的路径。
   - 将 *AWS_IOT_ENDPOINT* 替换为您的端点。
   - 将这两个*###CertId*实例替换为您的 HelloWorld _Subscriber 客户端设备文件名中的证书 ID。

   ```
   cd path-to-certs-folder
   python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
   cert subscriberCertId-certificate.pem.crt --key subscriberCertId-private.pem.key --
   thingName HelloWorld_Subscriber --topic 'hello/world/pubsub' --mode subscribe
   ```

   您应该看到以下输出，其中包括诸如 `Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}` 的条目。

   ```
   Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}

   2017-11-13 21:12:27,435 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
   2017-11-13 21:12:27,435 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
   2017-11-13 21:12:27,436 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
   Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}

   2017-11-13 21:12:28,320 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
   2017-11-13 21:12:28,324 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
   2017-11-13 21:12:28,324 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
   Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}

   2017-11-13 21:12:29,547 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
   2017-11-13 21:12:29,552 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
   2017-11-13 21:12:29,552 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
   ```

关闭 HelloWorld_Publisher 窗口，以阻止 HelloWorld_Subscriber 窗口中累积消息。

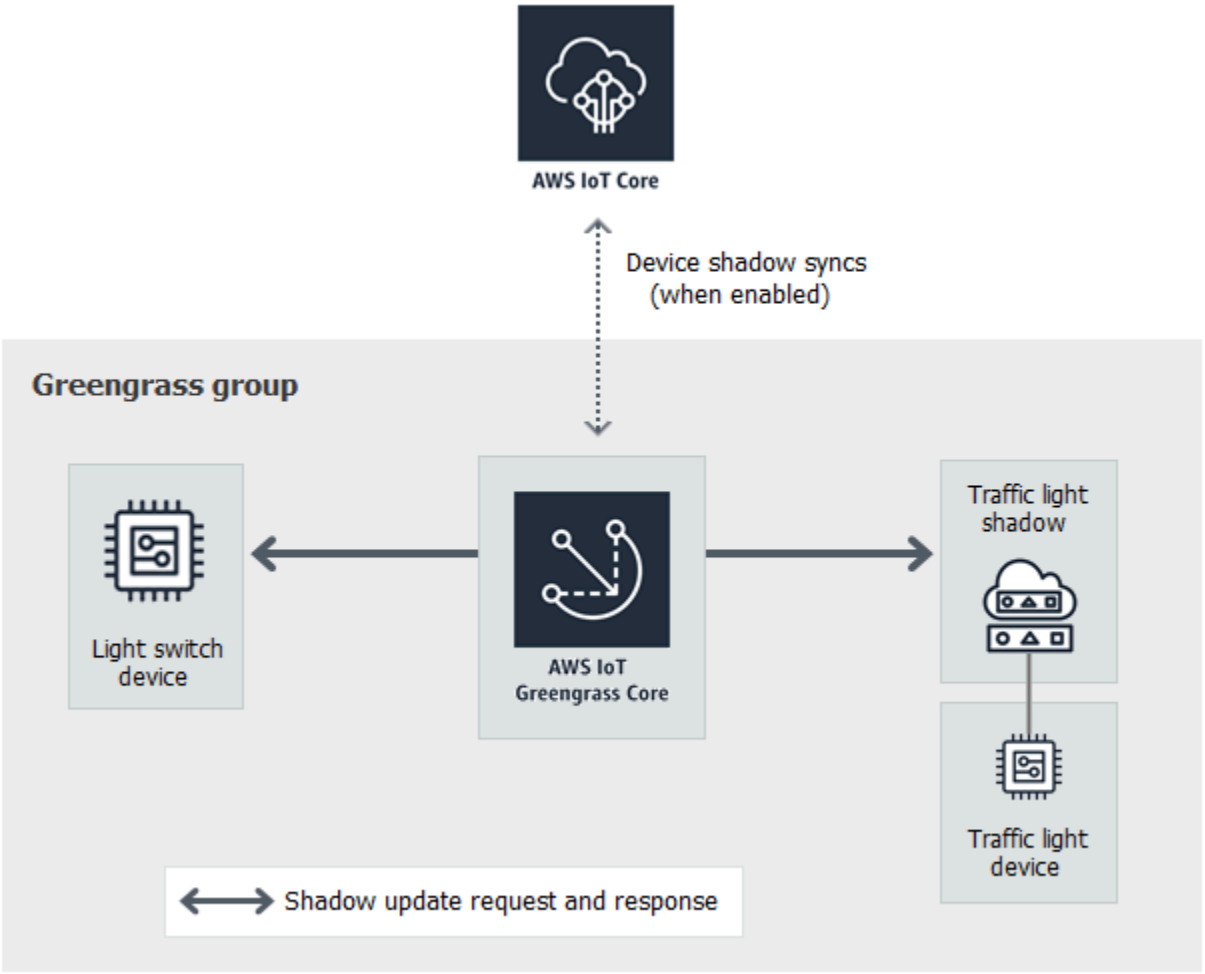
在公司网络上进行测试可能会干扰与核心的连接。作为解决方法，您可以手动输入端点。这样可以确保`basicDiscovery.py`脚本连接到 AWS IoT Greengrass 核心设备的正确 IP 地址。

手动输入端点

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择群组 (V1)。
2. 在 Greengrass 组下，选择您的组。
3. 配置核心，以手动管理 MQTT 代理端点。执行以下操作：
   a. 在组配置页面上，选择Lambda 函数选项卡。
   b. 在系统 Lambda 函数下，选择 IP 检测器，然后选择编辑。
   c. 在编辑 IP 检测器设置中，选择手动管理 MQTT 代理端点，然后选择保存。
4. 输入核心的 MQTT 代理端点。执行以下操作：
   a. 在概述下，选择 Greengrass 核心。
   b. 在 MQTT 代理端点下，选择管理端点。
   c. 选择添加端点，确保只有一个端点值。此值必须是 AWS IoT Greengrass 核心设备端口 8883 的 IP 地址端点（例如`192.168.1.4`）。
   d. 选择更新。

模块 5：与设备影子交互

本高级模块演示客户端设备如何与 AWS IoT Greengrass 组中的 AWS IoT 设备影子交互。影子 是用于存储事物的当前或所需状态信息的 JSON 文档。在本模块中，您将探索一个客户端设备 (`GG_Switch`) 如何修改另一个客户端设备 (`GG_TrafficLight`) 的状态，以及这些状态如何同步到 AWS IoT Greengrass 云：



在开始之前，请运行 Greengrass 设备安装程序脚本，或确保您已完成模块 1 和模块 2。您还应该了解如何将设备连接到 AWS IoT Greengrass 核心（模块 4）。您无需其他组件或设备。

本模块应该需要大约 30 分钟才能完成。

主题

- 配置设备和订阅
- 下载必需的文件
- 测试通信（禁用了设备同步）
- 测试通信 (启用了设备同步)

配置设备和订阅

当 AWS IoT Greengrass 核心连接到 Internet 时，影子可以同步到 AWS IoT。在此模块中，您首先使用本地影子而不同步到云。然后，您启用云同步。

每个客户端设备都有自己的影子。有关更多信息，请参阅 AWS IoT 开发人员指南中的适用于 AWS IoT 的 Device Shadow 服务。

1. 在组配置页面上，选择客户端设备选项卡。
2. 在客户端设备选项卡中，在 AWS IoT Greengrass 组中添加两台新的客户端设备。有关此过程的详细步骤，请参阅the section called “在 AWS IoT Greengrass 组中创建客户端设备”。
   - 将客户端设备命名为 **GG_Switch** 和 **GG_TrafficLight**。
   - 生成并下载两个客户端设备的安全资源。
   - 记下客户端设备的安全资源的文件名中的证书 ID。稍后会用到这些值。
3. 在您的计算机上创建一个文件夹来保存这些客户端设备的安全凭证。将证书和密钥复制到此文件夹。
4. 确保客户端设备设置为使用本地影子，而不与 AWS Cloud 同步。如果不是，请选择客户端设备，选择同步影子，然后选择禁用与云端的影子同步。
5. 将下表中的订阅添加到您的组中。例如，要创建第一个订阅，请执行以下操作：
   a. 在组配置页面中，选择 订阅选项卡，然后选择 添加。
   b. 对于来源类型，选择客户端设备，然后选择 GG_Switch。
   c. 对于目标类型，选择服务，然后选择本地影子服务。
   d. 对于 Topic filter (主题筛选条件)，输入 **$aws/things/GG_TrafficLight/shadow/update**。
   e. 选择 Create subscription（创建订阅）。

   主题的输入方式必须与表中所示完全相同。尽管可以使用通配符来整合一些订阅，但我们不建议这种做法。有关更多信息，请参阅 AWS IoT 开发人员指南中的影子 MQTT 主题。

源	目标	主题	注意
GG_Switch	本地影子服务	$aws/things/GG_TrafficLight/shadow/update	GG_Switch 发送更新请求来更新主题。
本地影子服务	GG_Switch	$aws/things/GG_TrafficLight/shadow/update/accepted	GG_Switch 需要知道更新请求是否被接受。
本地影子服务	GG_Switch	$aws/things/GG_TrafficLight/shadow/update/rejected	GG_Switch 需要知道更新请求是否被拒绝。
GG_TrafficLight	本地影子服务	$aws/things/GG_TrafficLight/shadow/update	GG_TrafficLight 将其状态的更新发送到更新主题。
本地影子服务	GG_TrafficLight	$aws/things/GG_TrafficLight/shadow/update/delta	本地影子服务通过增量主题将收到的更新发送到 GG_TrafficLight。
本地影子服务	GG_TrafficLight	$aws/things/GG_TrafficLight/shadow/update/accepted	GG_TrafficLight 需要知道其状态更新是否被接受。
本地影子服务	GG_TrafficLight	$aws/things/GG_TrafficLight/shadow/update/rejected	GG_TrafficLight 需要知道其状态更新是否被拒绝。

新订阅会显示在 订阅 选项卡上。

> **Note**
>
> 有关 $ 符号的信息，请参阅保留主题。

6. 确保启用了自动检测，以便 Greengrass 核心可以发布其 IP 地址的列表。客户端设备使用此信息来发现核心。执行以下操作：

   a. 在组配置页面上，选择Lambda 函数选项卡。

   b. 在系统 Lambda 函数下，选择 IP 检测器，然后选择编辑。

   c. 在编辑 IP 检测器设置中，选择自动检测和覆盖 MQTT 代理端点，然后选择保存。

7. 确保 Greengrass 进程守护程序正在运行，如 将云配置部署到核心设备 中所述。

8. 在组配置页面上，选择部署。

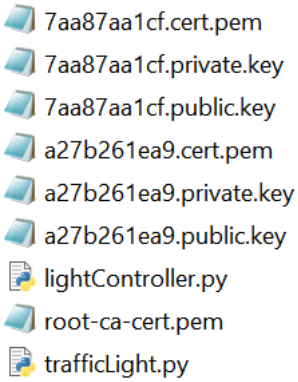
下载必需的文件

1. 如果您还没有这样做，请安装适用于 Python 的 AWS IoT Device SDK。有关说明，请参阅the section called “安装适用于 Python 的 AWS IoT Device SDK”中的步骤 1。

   该开发工具包供客户端设备用于与 AWS IoT 以及与 AWS IoT Greengrass 核心设备通信。

2. 从 GitHub 上的 TrafficLight 示例文件夹中，将 `lightController.py` 和 `trafficLight.py` 文件下载到您的计算机上。将它们保存在包含 GG_Switch 和 GG_TrafficLight 客户端设备证书和密钥的文件夹中。

   `lightController.py` 脚本对应于 GG_Switch 客户端设备，而 `trafficLight.py` 脚本对应于 GG_TrafficLight 客户端设备。

> **Note**
>
> 为方便起见，Python 示例文件存储在适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包存储库中，但它们不使用 AWS IoT Greengrass Core 软件开发工具包。

测试通信（禁用了设备同步）

1. 确保您的计算机和 AWS IoT Greengrass 核心设备使用相同的网络连接到互联网。

   a. 在 AWS IoT Greengrass 核心设备上，运行以下命令以查找其 IP 地址。

   ```
   hostname -I
   ```

   b. 在计算机上，使用核心的 IP 地址运行以下命令。可以使用 Ctrl + C 停止 ping 命令。

   ```
   ping IP-address
   ```

   类似于以下内容的输出表示计算机和 AWS IoT Greengrass 核心设备之间成功通信（丢包 0%）：

   ```
   $ping 176.32.103.205
   PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
   64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
   64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
   64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
   64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
   64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
   64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
   ^C
   --- 176.32.103.205 ping statistics ---
   6 packets transmitted, 6 received, 0% packet loss, time 5549ms
   rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
   ```

   > **Note**
   >
   > 如果您无法 ping 正在运行的 EC2 实例 AWS IoT Greengrass，请确保该实例的入站安全组规则允许 ECHO 请求消息的 ICMP 流量。有关更多信息，请参阅 Amazon EC2 用户指南中的向安全组添加规则。
   >
   > 在 Windows 主机上，在具有高级安全性应用程序的 Windows 防火墙中，您可能还需要启用一个允许入站回显请求的入站规则（例如，File and Printer Sharing (Echo Request - ICMPv4-In) (文件和打印机共享 (回显请求 - ICMPv4-In)) ），或创建一个。

2. 获取您的 AWS IoT 终端节点。

   a. 从AWS IoT 控制台导航窗格中，选择设置。

b. 在设备数据端点下，记下 端点的值。您可以使用此值在以下步骤中替换命令中的 *AWS_IOT_ENDPOINT* 占位符。

> **Note**
>
> 确保您的终端节点与证书类型对应。

3. 在您的计算机（不是 AWS IoT Greengrass 核心设备）上，打开两个命令行（终端或命令提示符）窗口。一个窗口代表 GG_Switch 客户端设备，另一个窗口代表 GG_ TrafficLight 客户端设备。

   a. 从 GG_Switch 客户端设备窗口中，运行以下命令。

   - 将 *path-to-certs-folder* 替换为包含证书、密钥和 Python 文件的文件夹的路径。
   - 将 *AWS_IOT_ENDPOINT* 替换为您的端点。
   - 将这两个*###CertId*实例替换为您的 GG_Switch 客户端设备文件名中的证书 ID。

```
cd path-to-certs-folder
python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA
 AmazonRootCA1.pem --cert switchCertId-certificate.pem.crt --key switchCertId-
private.pem.key --thingName GG_TrafficLight --clientId GG_Switch
```

   b. 在 GG_ TrafficLight 客户端设备窗口中，运行以下命令。

   - 将 *path-to-certs-folder* 替换为包含证书、密钥和 Python 文件的文件夹的路径。
   - 将 *AWS_IOT_ENDPOINT* 替换为您的端点。
   - 将这两个*##CertId*实例替换为您的 GG_ TrafficLight 客户端设备文件名中的证书 ID。

```
cd path-to-certs-folder
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
 --cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --
thingName GG_TrafficLight --clientId GG_TrafficLight
```

   每隔 20 秒，开关会将影子状态更新为 G、Y 和 R，并且灯会显示新状态，如下文所示。

   GG_Switch 输出：

```
{"state":{"desired":{"property":"R"}}}
2018-12-20 12:23:01,446 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~~~~~~~~Shadow Update Accepted~~~~~~~~~~~~~~~
Update request with token: 3b22e27c-930d-4c6a-8562-9f86088249f4 accepted!
property: R
~~~~~~~~~~~~~~~~~~~~~~~~~~
```

GG_ TrafficLight 输出：

```
++++++++ Received Shadow Delta ++++++++++
{u'state': {u'property': u'R'}, u'metadata': {u'property': {u'timestamp': 1545337381}}, u'version': 33, u'clientToken':
u'3b22e27c-930d-4c6a-8562-9f86088249f4'}
property: R
version: 33
++++++++++++++++++++++


Light changed to: R
{"state":{"reported":{"property":"R"}}}
2018-12-20 12:23:01,539 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~~~~~~~~ Shadow Update Accepted ~~~~~~~~~~~~~~~
Update request with token: f552109f-c1c2-4ae6-a841-8443506eefcb accepted!
property: R
~~~~~~~~~~~~~~~~~~~~~~~~~~
```

首次执行时，每个客户端设备脚本都会运行 AWS IoT Greengrass 发现服务以连接到 AWS IoT Greengrass 核心（通过 Internet）。在客户机设备发现并成功连接到 AWS IoT Greengrass 核心之后，可以在本地执行 future 操作。

> **Note**
>
> `lightController.py` 和 `trafficLight.py` 脚本将连接信息存储在 `groupCA` 文件夹中，该文件夹与脚本在同一文件夹中创建。如果您收到连接错误，请确保 `ggc-host` 文件中的 IP 地址与您在此步骤中为核心配置的单个 IP 地址端点匹配。

4. 在 AWS IoT 控制台中，选择您的 AWS IoT Greengrass 群组，选择客户端设备选项卡，然后选择 GG_ TrafficLight 以打开客户端设备 AWS IoT 的事物详细信息页面。
5. 选择设备影子选项卡。在 GG_Switch 更改状态后，该影子不应有任何更新。这是因为 GG_ 设置 TrafficLight 为禁用与云的阴影同步。
6. 在 GG_Switch (`lightController.py`) 客户端设备窗口中按 Ctrl + C。您应该看到 GG_ TrafficLight (`trafficLight.py`) 窗口停止接收状态更改消息。

   保持这些窗口打开，以便您可以在下一部分中运行这些命令。

测试通信 (启用了设备同步)

对于此测试，您将 GG_TrafficLight 设备影子配置为与 AWS IoT 同步。您运行与上一个测试中相同的命令，但这次当 GG_Switch 发送更新请求时，云中的影子状态会更新。

1. 在 AWS IoT 控制台中，选择您的 AWS IoT Greengrass 组，然后选择客户端设备选项卡。
2. 选择 GG_TrafficLight 设备，选择同步影子，然后选择启用与云的影子同步。

   您应该会收到设备影子同步数据已更新的通知。
3. 在组配置页面上，选择部署。
4. 在两个命令行窗口中，针对 GG_Switch 和 GG_TrafficLight 客户端设备运行上一个测试的命令。
5. 现在，请检查 AWS IoT 控制台中的影子状态。选择您的AWS IoT Greengrass 组，选择客户端设备选项卡，选择 GG_TrafficLight，选择设备影子选项卡，然后选择经典影子。

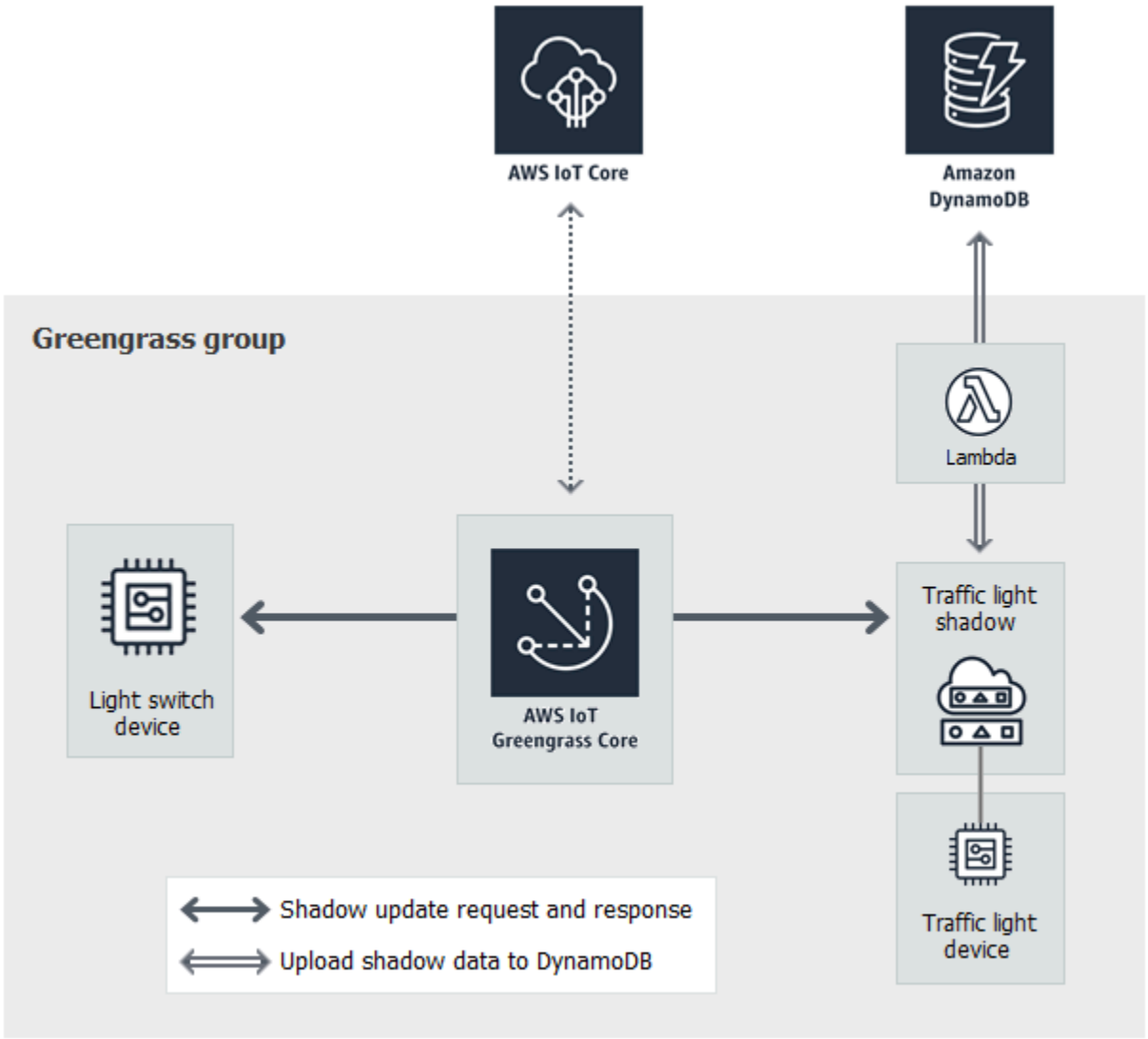
   因为已启用 GG_TrafficLight 影子到 AWS IoT 的同步，所以每当 GG_Switch 发送更新时，云中的影子状态应该会更新。此功能可用于将客户端设备的状态公开给 AWS IoT。

   > **Note**
   >
   > 如果需要，您可以通过查看 AWS IoT Greengrass Core 日志（具体而言是 `runtime.log`）来排查问题：
   >
   > ```
   > cd /greengrass/ggc/var/log
   > sudo cat system/runtime.log | more
   > ```
   >
   > 您也可以查看 `GGShadowSyncManager.log` 和 `GGShadowService.log`。有关更多信息，请参阅排查问题。

保持客户端设备和订阅的设置。您将在下一模块中使用它们。您还会运行相同的命令。

模块 6：访问其他 AWS 服务

本高级模块演示 AWS IoT Greengrass 核心如何与云中的其他 AWS 服务交互。它在模块 5 中的交通信号灯示例基础上构建，并添加一个 Lambda 函数来处理影子状态并将摘要上传到 Amazon DynamoDB 表。



在开始之前，请运行 Greengrass 设备安装程序脚本，或确保您已完成模块 1 和模块 2。您还应该完成模块 5。您无需其他组件或设备。

本模块应该需要大约 30 分钟才能完成。

> **Note**
>
> 本模块将在 DynamoDB 中创建并更新表。尽管大多数操作比较小并且在 Amazon Web Services Free Tier 内执行，但执行本模块中的某些步骤可能会在您的账户中产生费用。有关定价的信息，请参阅 DynamoDB 定价文档。

主题

配置组角色

组角色是您创建并附加到 Greengrass 组的 IAM 角色。此角色包含部署的 Lambda 函数（和其他 AWS IoT Greengrass 功能）用来访问 AWS 服务的权限。有关更多信息，请参阅the section called “Greengrass 组角色”。

您可以使用以下高级步骤在 IAM 控制台中创建组角色。

1. 创建允许或拒绝对一个或多个资源执行操作的策略。
2. 创建使用 Greengrass 服务作为受信任实体的角色。
3. 将策略附加到角色。

然后，在 AWS IoT 控制台中，您将角色添加到 Greengrass 组。

> **Note**
>
> 一个 Greengrass 组有一个组角色。如果要添加权限，可以编辑附加的策略或附加更多策略。

对于此教程，您将创建权限策略，该策略允许对 Amazon DynamoDB 表执行描述、创建和更新操作。然后，您可以将该策略附加到新角色并将该角色与您的 Greengrass 组关联。

首先，创建一个客户托管策略，该策略授予这个模块中的 Lambda 函数要求的权限。

1. 在 IAM 控制台的导航窗格中，选择 Policies，然后选择 Create policy。
2. 在 JSON 选项卡中，将占位符内容替换为以下策略。此模块中的 Lambda 函数使用这些权限来创建和更新名为 `CarStats` 的 DynamoDB 表。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PermissionsForModule6",
            "Effect": "Allow",
            "Action": [
                "dynamodb:DescribeTable",
                "dynamodb:CreateTable",
                "dynamodb:PutItem"
            ],
            "Resource": "arn:aws:dynamodb:*:*:table/CarStats"
        }
    ]
}
```

3. 依次选择 Next: Tags（下一步：标签）和 Next: Review（下一步：查看）。本教程中未使用标签。
4. 对于 Name (名称)，输入 **greengrass_CarStats_Table**，然后选择 Create policy (创建策略)。

   接下来，创建一个使用新策略的角色。
5. 在导航窗格中，选择 Roles (角色)，然后选择 Create role (创建角色)。
6. 在可信实体类型下，选择 AWS 服务。
7. 在用例、其他 AWS 服务的用例下面，选择 Greengrass，选择 Greengrass，然后选择下一步。
8. 在权限策略下，选择新的 **greengrass_CarStats_Table** 策略，然后选择下一步。
9. 对于角色名称，输入 **Greengrass_Group_Role**。
10. 对于描述，输入 **Greengrass group role for connectors and user-defined Lambda functions**。
11. 选择 Create role（创建角色）。

    现在，将该角色添加到 Greengrass 组。
12. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
13. 在 Greengrass 组下，选择您的组。
14. 选择 设置，然后选择 关联角色。
15. 从您的角色列表中选择 Greengrass_Group_Role，然后选择关联角色。

创建并配置 Lambda 函数

在本步骤中，您将创建一个 Lambda 函数，该函数将跟踪通过交通信号灯的汽车的数量。每当 `GG_TrafficLight` 影子状态更改为 `G` 时，Lambda 函数都将模拟通过的随机数量的汽车（从 1 到 20）。每当 `G` 灯第三次改变，Lambda 函数都会将基本统计数据（如最小值和最大值）发送到 DynamoDB 表。

1. 在您的计算机上，创建一个名为 `car_aggregator` 的文件夹。
2. 从 GitHub 上的 TrafficLight 示例文件夹中，将 `carAggregator.py` 文件下载到 `car_aggregator` 文件夹。这是您的 Lambda 函数代码。

   > **Note**
   >
   > 为方便起见，此 Python 示例文件存储在 AWS IoT Greengrass 核心开发工具包存储库中，但它不使用 AWS IoT Greengrass 核心开发工具包。

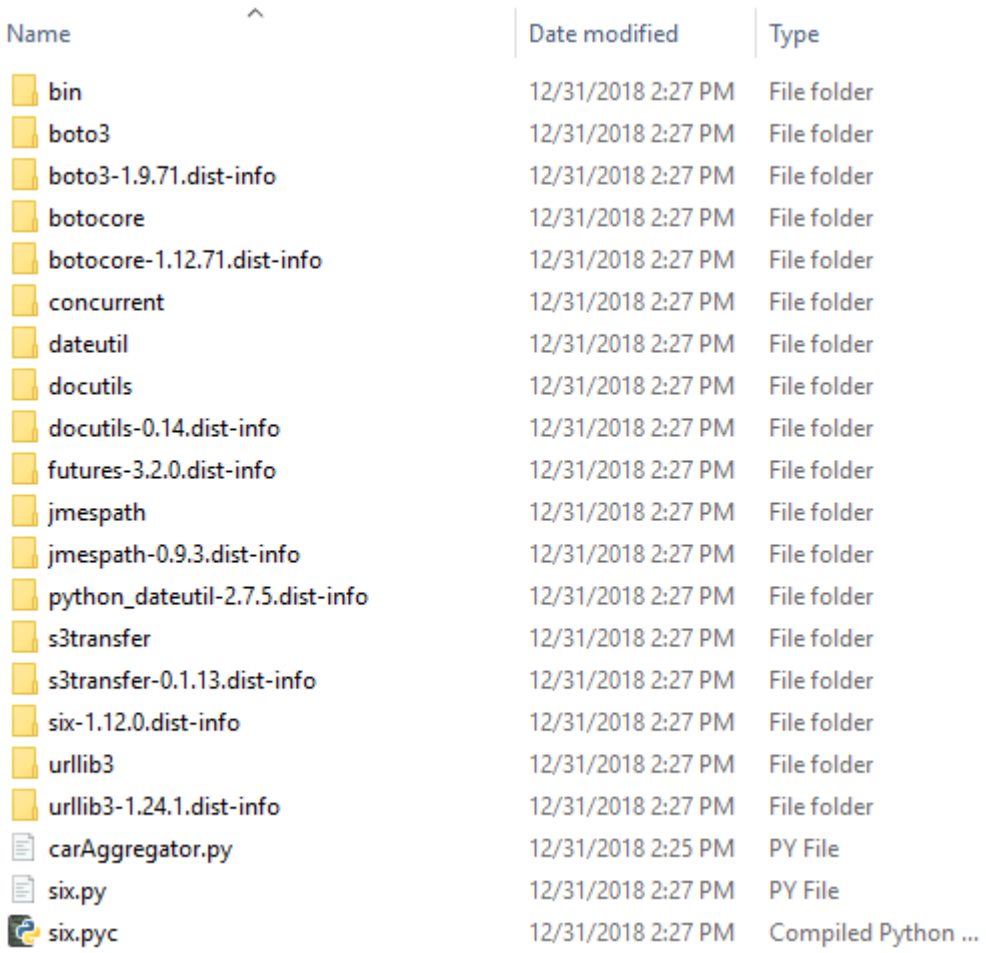
3. 如果您不在美国东部（弗吉尼亚北部）区域工作，请打开 `carAggregator.py`，并将以下行中的 `region_name` 更改为当前在 AWS IoT 控制台中选择的 AWS 区域。有关支持的 AWS 区域，请参见 Amazon Web Services 一般参考 中的 AWS IoT Greengrass。

   ```
   dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
   ```

4. 在命令行窗口中运行以下命令，以将 AWS SDK for Python (Boto3) 软件包及其依赖项安装到 `car_aggregator` 文件夹中。Greengrass Lambda 函数使用 AWS 开发工具包访问其他 AWS 服务。（对于 Windows，请使用提升的命令提示符。）

   ```
   pip install boto3 -t path-to-car_aggregator-folder
   ```

   这会显示类似于下面的目录列表：

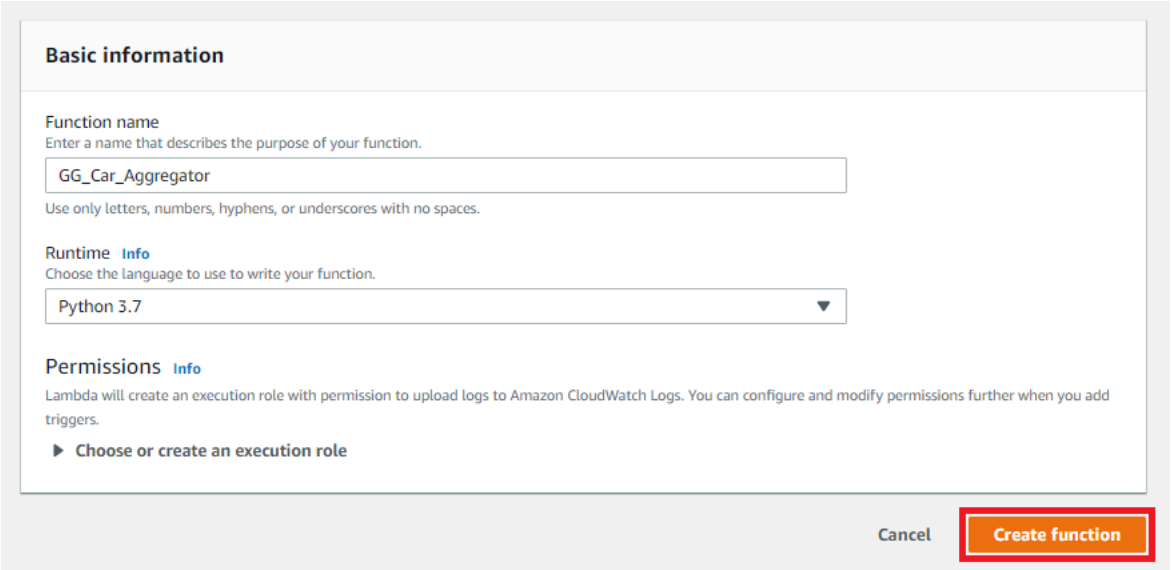


Name	Date modified	Type
bin	12/31/2018 2:27 PM	File folder
boto3	12/31/2018 2:27 PM	File folder
boto3-1.9.71.dist-info	12/31/2018 2:27 PM	File folder
botocore	12/31/2018 2:27 PM	File folder
botocore-1.12.71.dist-info	12/31/2018 2:27 PM	File folder
concurrent	12/31/2018 2:27 PM	File folder
dateutil	12/31/2018 2:27 PM	File folder
docutils	12/31/2018 2:27 PM	File folder
docutils-0.14.dist-info	12/31/2018 2:27 PM	File folder
futures-3.2.0.dist-info	12/31/2018 2:27 PM	File folder
jmespath	12/31/2018 2:27 PM	File folder
jmespath-0.9.3.dist-info	12/31/2018 2:27 PM	File folder
python_dateutil-2.7.5.dist-info	12/31/2018 2:27 PM	File folder
s3transfer	12/31/2018 2:27 PM	File folder
s3transfer-0.1.13.dist-info	12/31/2018 2:27 PM	File folder
six-1.12.0.dist-info	12/31/2018 2:27 PM	File folder
urllib3	12/31/2018 2:27 PM	File folder
urllib3-1.24.1.dist-info	12/31/2018 2:27 PM	File folder
carAggregator.py	12/31/2018 2:25 PM	PY File
six.py	12/31/2018 2:27 PM	PY File
six.pyc	12/31/2018 2:27 PM	Compiled Python ...

5. 将 `car_aggregator.zip` 文件夹的内容压缩到一个名为 `car_aggregator.zip` 的文件中。（压缩文件夹的内容，而不是文件夹。）此即 Lambda 函数部署程序包。
6. 在 Lambda 控制台中，创建一个名为 **`GG_Car_Aggregator`** 的函数，然后按下面所示设置其余字段：

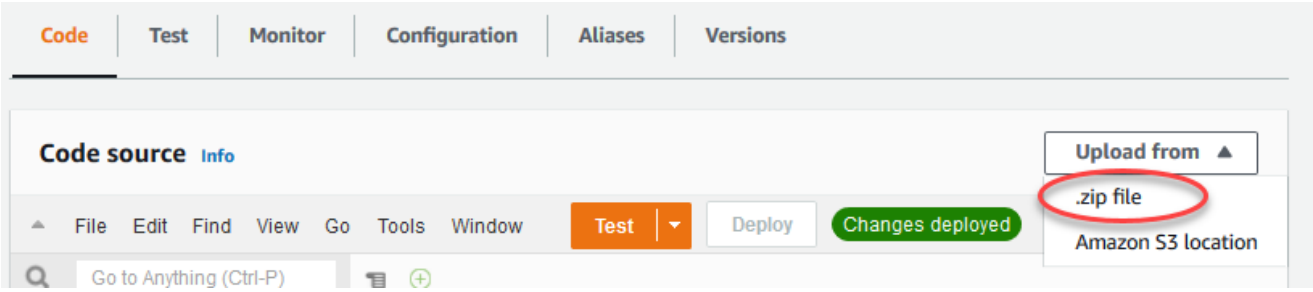
   - 对于 Runtime (运行时)，选择 Python 3.7。
   - 对于权限，请保留默认设置。这将创建一个授予基本 Lambda 权限的执行角色。此角色未由 AWS IoT Greengrass 使用。

   选择 Create function（创建函数）。



7. 上传 Lambda 函数部署软件包：

   a. 在代码选项卡上的代码源下，选择上传自。从下拉列表中选择 .zip 文件。



   b. 选择上传，然后选择您的 `car_aggregator.zip` 部署包。然后，选择 Save (保存)。

   c. 在函数的代码选项卡中，在运行时设置下选择编辑，然后输入以下值。

      - 对于 Runtime (运行时)，选择 Python 3.7。
      - 对于 Handler (处理程序)，输入 **`carAggregator.function_handler`**。

   d. 选择 Save（保存）。

8. 发布 Lambda 函数，然后创建一个名为 **`GG_CarAggregator`** 的别名。有关分步说明，请参阅模块 3（第 1 部分）中的发布 Lambda 函数和创建别名步骤。

9. 在 AWS IoT 控制台中，将刚才创建的 Lambda 函数添加到 AWS IoT Greengrass 组：

   a. 在组配置页面上，选择 Lambda 函数，然后在我的 Lambda 函数下选择添加。

   b. 对于 Lambda 函数，请选择 gg_car_Aggregator。

   c. 对于 Lambda 函数版本，选择您发布的版本的别名。

   d. 对于 Memory limit (内存限制)，输入 **`64 MB`**。

e. 对于已固定，选择 True。

f. 选择添加 Lambda 函数。

> **Note**
>
> 您可以删除先前模块中的其他 Lambda 函数。

配置订阅

在本步骤中，您将创建一个订阅，使得 GG_TrafficLight 影子能够将更新后的状态信息发送到 GG_Car_Aggregator Lambda 函数。此订阅会添加到您在模块 5 中创建的订阅中，这些订阅在此模块中都是必需的。

1. 在组配置页面中，选择 订阅选项卡，然后选择 添加。
2. 在创建事件订阅页中，执行以下操作：

   a. 对于源类型，选择服务，然后选择本地影子服务。

   b. 对于目标类型，选择 Lambda 函数，然后选择 GG_Car_Aggregator。

   c. 对于 Topic filter (主题筛选条件)，输入 **`$aws/things/GG_TrafficLight/shadow/update/documents`**。

   d. 选择 Create subscription（创建订阅）。

   此模块需要新订阅和您在模块 5 中创建的订阅。
3. 确保 Greengrass 进程守护程序正在运行，如 将云配置部署到核心设备 中所述。
4. 在组配置页面上，选择部署。

测试通信

1. 在您的计算机上，打开两个命令行窗口。就像在模块 5 中一样，一个窗口用于 GG_Switch 客户端设备，另一个窗口用于 GG_TrafficLight 客户端设备。您可以使用它们运行在模块 5 中运行的相同命令。

   为 GG_Switch 客户端设备运行以下命令：

```
cd path-to-certs-folder
python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
 --cert switchCertId-certificate.pem.crt --key switchCertId-private.pem.key --
thingName GG_TrafficLight --clientId GG_Switch
```

为 GG_TrafficLight 客户端设备运行以下命令：

```
cd path-to-certs-folder
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --thingName
 GG_TrafficLight --clientId GG_TrafficLight
```

每隔 20 秒，开关会将影子状态更新为 G、Y 和 R，并且灯会显示新状态。

2. 在每次第 3 个绿灯时（每 3 分钟），都会触发 Lambda 函数的函数处理程序，并会创建新的 DynamoDB 记录。在 `lightController.py` 和 `trafficLight.py` 运行了 3 分钟后，转到 AWS Management Console，并打开 DynamoDB 控制台。
3. 在 AWS 区域菜单中，选择美国东部（弗吉尼亚州北部）区域。这是 `GG_Car_Aggregator` 函数在其中创建表的区域。
4. 在导航窗格中，选择表，然后选择 CarStats 表。
5. 选择查看项目，以查看表中的条目。

   您应该看到包含有关通过的汽车数的基本统计数据的条目 (每隔 3 分钟创建一个条目)。您可能需要选择刷新按钮来查看表的更新。
6. 如果测试不成功，您可以在 Greengrass 日志中查找故障排除信息。

   a. 切换到根用户并导航到 `log` 目录。访问 AWS IoT Greengrass 日志需要根权限。

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

   b. 检查 `runtime.log` 有无错误。

   ```
   cat system/runtime.log | grep 'ERROR'
   ```

   c. 检查 Lambda 函数生成的日志。

   ```
   cat user/region/account-id/GG_Car_Aggregator.log
   ```

`lightController.py` 和 `trafficLight.py` 脚本将连接信息存储在 `groupCA` 文件夹中，该文件夹与脚本在同一文件夹中创建。如果您收到连接错误，请确保 `ggc-host` 文件中的 IP 地址与您在此步骤中为核心配置的单个 IP 地址端点匹配。

有关更多信息，请参阅排查问题。

本基础教程到此结束。您现在应该了解了 AWS IoT Greengrass 编程模型及其基本概念，包括 AWS IoT Greengrass 核心、组、订阅、客户端设备以及在边缘站点运行的 Lambda 函数的部署过程。

您可以删除 DynamoDB 表以及 Greengrass Lambda 函数和订阅。要停止 AWS IoT Greengrass 核心设备和 AWS IoT 云之间的通信，请在核心设备上打开终端并运行以下命令之一：

- 关闭 AWS IoT Greengrass 核心设备：

```
sudo halt
```

- 停止 AWS IoT Greengrass 进程守护程序：

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

模块 7：模拟硬件安全集成

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

此高级模块向您展示如何配置模拟硬件安全模块 (HSM) 以便与 Greengrass 核心一起使用。该配置使用 SoftHSM，这是一个使用 PKCS#11 应用程序编程接口 (API) 的纯软件实现。此模块的目的是允许您设置一个环境，您可以在其中学习 PKCS#11 API 的纯软件实现并针对其进行初始测试。它仅用于学习和初始测试，不适用于任何类型的生产用途。

您可以通过此配置尝试使用与 PKCS＃11 兼容的服务来存储您的私有密钥。有关纯软件实施的更多信息，请参阅 SoftHSM。有关在 AWS IoT Greengrass 核心上集成硬件安全性的更多信息，包括一般要求，请参阅the section called “硬件安全性集成”。

> **Important**
>
> 此模块仅用于实验目的。我们强烈建议不要在生产环境中使用 SoftHSM，因为它可能产生一种安全性得到增强的错觉。实际上，生成的配置不具备任何实际安全性优势。存储在 SoftHSM 中的密钥并不比 Greengrass 环境中任何其他密钥存储方式更加安全。
> 本模块的目的是让您了解 PKCS#11 规范，并在将来计划使用基于硬件的实际 HSM 时对软件进行初始测试。
> 在用于生产用途之前，您必须单独全面地测试未来的硬件实现，因为 SoftHSM 中提供的 PKCS#11 实现与基于硬件的实现之间可能存在差异。

如果在使用支持的硬件安全模块方面需要帮助，请联系您的 AWS 企业支持代表。

在开始之前，请运行 Greengrass 设备安装 脚本，或确保您已完成入门教程的模块 1 和模块 2。在本模块中，我们假设您的核心已经配置并与 AWS 通信。本模块应该需要大约 30 分钟才能完成。

安装 SoftHSM 软件

在本步骤中，您将安装 SoftHSM 和 pkcs11 工具，这些工具用于管理您的 SoftHSM 实例。

- 在 AWS IoT Greengrass 核心设备上的终端中，运行以下命令：

  ```
  sudo apt-get install softhsm2 libsofthsm2-dev pkcs11-dump
  ```

  有关这些程序包的更多信息，请参阅安装 softhsm2、安装 libsofthsm2-dev 和安装 pkcs11-dump。

  > **Note**
  >
  > 如果在系统上使用此命令时遇到问题，请参阅 GitHub 上的 SoftHSM 版本 2。该网站提供了更多安装信息，包括如何从源代码构建。

配置 SoftHSM

在此步骤中，您将配置 SoftHSM。

1. 切换到根用户。

```
sudo su
```

2. 使用手册页面查找系统范围的 softhsm2.conf 位置。一个常见位置是 /etc/softhsm/softhsm2.conf，但在某些系统上，位置可能有所不同。

```
man softhsm2.conf
```

3. 在系统范围的位置中为 softhsm2 配置文件创建目录。在本例中，我们假设位置为 /etc/softhsm/softhsm2.conf。

```
mkdir -p /etc/softhsm
```

4. 在 /greengrass 目录中创建令牌目录。

> **Note**
>
> 如果跳过此步骤，softhsm2-util 将报告 ERROR: Could not initialize the library。

```
mkdir -p /greengrass/softhsm2/tokens
```

5. 配置令牌目录。

```
echo "directories.tokendir = /greengrass/softhsm2/tokens" > /etc/softhsm/
softhsm2.conf
```

6. 配置基于文件的后端。

```
echo "objectstore.backend = file" >> /etc/softhsm/softhsm2.conf
```

> **Note**
>
> 这些配置设置仅用于实验目的。要查看所有配置选项，请打开配置文件的手册页面。
>
> ```
> man softhsm2.conf
> ```

将私有密钥导入 SoftHSM

在此步骤中，初始化 SoftHSM 令牌，转换私有密钥格式，然后导入私有密钥。

1. 初始化 SoftHSM 令牌。

```
softhsm2-util --init-token --slot 0 --label greengrass --so-pin 12345 --pin 1234
```

> **Note**
>
> 如果出现提示，请输入 SO PIN 12345 和用户 PIN 1234。AWS IoT Greengrass 不使用 SO（主管）PIN，因此您可以使用任何值。
> 如果您收到 `CKR_SLOT_ID_INVALID: Slot 0 does not exist` 错误，请尝试执行以下命令：
>
> ```
> softhsm2-util --init-token --free --label greengrass --so-pin 12345 --pin
>  1234
> ```

2. 将私有密钥转换成 SoftHSM 导入工具可以使用的格式。在本教程中，您将转换从“入门”教程模块 2 中的默认组创建选项中获得的私钥。

```
openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in hash.private.key -
out hash.private.pem
```

3. 将私有密钥导入 SoftHSM。只运行以下命令之一，具体取决于您的 softhsm2-util 版本。

   Raspbian softhsm2-util v2.2.0 语法

```
softhsm2-util --import hash.private.pem --token greengrass --label iotkey --id
 0000 --pin 12340
```

   Ubuntu softhsm2-util v2.0.0 语法

```
softhsm2-util --import hash.private.pem --slot 0 --label iotkey --id 0000 --pin
 1234
```

   此命令将插槽标识为 `0`，并将键标签定义为 `iotkey`。您将在下一节中使用这些值。

导入私有密钥后，您可以选择将其从 /greengrass/certs 目录中删除。确保将根 CA 和设备证书保留在目录中。

配置 Greengrass 核心以使用 SoftHSM

在本步骤中，您将修改 Greengrass 核心配置文件以使用 SoftHSM。

1. 在您的系统上找到 SoftHSM 提供程序库 (libsofthsm2.so) 的路径：

   a. 获取库的已安装程序包的列表。

   ```
   sudo dpkg -L libsofthsm2
   ```

   libsofthsm2.so 文件位于 softhsm 目录中。

   b. 复制文件的完整路径（例如 /usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so）。您稍后会使用此值。

2. 停止 Greengrass 守护程序。

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

3. 打开 Greengrass 配置文件。这是位于 /greengrass/config 目录中的 config.json 文件。

   > **Note**
   >
   > 本过程中的示例假设 config.json 文件使用“入门”教程模块 2 中的默认组创建选项生成的格式。

4. 在 crypto.principals 对象中，插入以下 MQTT 服务器证书对象。在需要的位置添加逗号以创建有效的 JSON 文件。

   ```
   "MQTTServerCertificate": {
     "privateKeyPath": "path-to-private-key"
   }
   ```

5. 在 crypto 对象中，插入以下 PKCS11 对象。在需要的位置添加逗号以创建有效的 JSON 文件。

   ```
   "PKCS11": {
     "P11Provider": "/path-to-pkcs11-provider-so",
     "slotLabel": "crypto-token-name",
   ```

```
    "slotUserPin": "crypto-token-user-pin"
  }
```

您的文件应如下所示：

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix.iot.region.amazonaws.com",
    "ggHost" : "greengrass.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto": {
    "PKCS11": {
      "P11Provider": "/path-to-pkcs11-provider-so",
      "slotLabel": "crypto-token-name",
      "slotUserPin": "crypto-token-user-pin"
    },
    "principals" : {
      "MQTTServerCertificate": {
        "privateKeyPath": "path-to-private-key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      },
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

> **Note**
>
> 要将无线 (OTA) 更新与硬件安全性结合使用，PKCS11 对象还必须包含 OpenSSLEngine 属性。有关更多信息，请参阅the section called “配置 OTA 更新”。

6. 编辑 crypto 数据：

   a. 配置 PKCS11 对象。

      - 对于 P11Provider，输入 libsofthsm2.so 的完整路径。
      - 对于 slotLabel，输入 greengrass。
      - 对于 slotUserPin，输入 1234。

   b. 在 principals 对象中配置私有密钥路径。请勿编辑 certificatePath 属性。

      - 对于 privateKeyPath 属性，请输入以下 RFC 7512 PKCS#11 路径（指定键的标签）。对 IoTCertificate、SecretsManager 和 MQTTServerCertificate 委托人执行此操作。

        ```
        pkcs11:object=iotkey;type=private
        ```

   c. 检查 crypto 对象。如下所示：

      ```
      "crypto": {
        "PKCS11": {
          "P11Provider": "/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so",
          "slotLabel": "greengrass",
          "slotUserPin": "1234"
        },
        "principals": {
          "MQTTServerCertificate": {
            "privateKeyPath": "pkcs11:object=iotkey;type=private"
          },
          "SecretsManager": {
            "privateKeyPath": "pkcs11:object=iotkey;type=private"
          },
          "IoTCertificate": {
            "certificatePath": "file://certs/core.crt",
            "privateKeyPath": "pkcs11:object=iotkey;type=private"
          }
        },
      ```

```
    "caPath": "file://certs/root.ca.pem"
  }
```

7. 从 coreThing 对象中删除 caPath、certPath 和 keyPath。如下所示：

```
"coreThing" : {
  "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
  "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
  "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
  "keepAlive" : 600
}
```

> **Note**
>
> 在本教程中，您将为所有委托人指定相同的私有密钥。有关为本地 MQTT 服务器选择私有密钥的更多信息，请参阅性能。有关本地 Secrets Manager 的更多信息，请参阅将密钥部署到核心。

测试配置

- 启动 Greengrass 守护程序。

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

如果守护程序成功启动，则您的核心已正确配置。

现在已准备就绪，可以了解 PKCS#11 规范并使用 SoftHSM 实现提供的 PKCS#11 API 进行初始测试。

> **Important**
>
> 重申一遍，必须注意此模块仅用于学习和测试，这一点非常重要。它不会实际改善您的 Greengrass 环境的安全状况。
> 该模块的目的是使您能够开始学习和测试，以便为将来使用真正的基于硬件的 HSM 做准备。那时，您必须在开始任何生产用途之前，针对基于硬件的 HSM 单独全面地测试您的软件，因为 SoftHSM 中提供的 PKCS＃11 实施与基于硬件的实施之间可能存在差异。

另请参阅

- PKCS #11 加密令牌接口使用指南 2.40 版本。编辑者：John Leiseboer 和 Robert Griffin。2014 年 11 月 16 日。OASIS 委员会注意 02。http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html。最新版本：http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html。
- RFC 7512

AWS IoT Greengrass Core 软件的 OTA 更新

AWS IoT Greengrass Core 软件包中包含可执行 AWS IoT Greengrass 软件无线 (OTA) 更新的更新代理。您可以使用 OTA 更新在一个或多个核心上安装最新版本的 AWS IoT Greengrass Core 软件或 OTA 更新代理软件。通过 OTA 更新，您的核心设备无需物理连接。

我们建议您尽可能使用 OTA 更新。这提供了一种可用于跟踪更新状态和更新历史记录的机制。如果更新失败，则 OTA 更新代理回滚到以前的软件版本。

> **Note**
>
> 使用 `apt` 安装 AWS IoT Greengrass Core 软件时，不支持 OTA 更新。对于这些安装，建议您使用 `apt` 升级软件。有关更多信息，请参阅the section called “从 APT 存储库安装”。

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。

此功能与 AWS IoT 作业集成。

要求

以下要求适用于 AWS IoT Greengrass 软件的 OTA 更新。

- Greengrass 核心的本地存储必须至少有 400 MB 的可用磁盘空间。OTA 更新代理的需求是 AWS IoT Greengrass Core 软件运行时使用需求的大约三倍。有关更多信息，请参阅 Amazon Web Services 一般参考 中 Greengrass 核心的服务限额。
- Greengrass Core 必须具有到 AWS Cloud 的连接。
- 必须使用证书和密钥正确配置和预置 Greengrass Core 以便对 AWS IoT Core 和 AWS IoT Greengrass 进行身份验证。有关更多信息，请参阅the section called “X.509 证书”。
- Greengrass Core 不能配置为使用网络代理。

> **Note**
>
> 从 AWS IoT Greengrass v1.9.3 开始，将 MQTT 流量配置为使用端口 443 而不是默认端口 8883 的核心支持 OTA 更新。但是，OTA 更新代理不支持通过网络代理进行更新。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”。

- 无法在包含 AWS IoT Greengrass Core 软件的分区中启用可信引导。

> **Note**
>
> 您可以在启用了可信引导的分区上安装和运行 AWS IoT Greengrass Core 软件，但这不支持 OTA 更新。

- AWS IoT Greengrass 必须具有对包含 AWS IoT Greengrass Core 软件的分区的读/写权限。
- 如果您使用 init 系统来管理 Greengrass Core，则必须配置 OTA 更新以与 init 系统集成。有关更多信息，请参阅the section called “与初始化系统集成”。
- 您必须创建用于对 AWS IoT Greengrass 软件更新构件的 Amazon S3 URL 进行预签名的角色。此签署人角色允许 AWS IoT Core 代表您访问存储在 Amazon S3 中的软件更新构件。有关更多信息，请参阅the section called “OTA 更新的 IAM 权限”。

OTA 更新的 IAM 权限

当 AWS IoT Greengrass 发布 AWS IoT Greengrass Core 软件的新版本时，AWS IoT Greengrass 更新存储在用于 OTA 更新的 Amazon S3 中的软件构件。

您的 AWS 账户 必须包含可用于访问这些构件的 Amazon S3 URL 签署人角色。该角色必须具有允许对目标 AWS 区域 中存储桶执行 `s3:GetObject` 操作的权限策略。该角色还必须具有信任策略，允许 `iot.amazonaws.com` 作为可信实体代入角色。

权限策略

对于角色权限，您可以使用 AWS 托管策略或创建自定义策略。

- 使用 AWS 托管策略

  GreengrassOTAUpdateArtifactAccess 托管策略由 AWS IoT Greengrass 提供。如果您希望允许访问 AWS IoT Greengrass 现在和将来支持的所有 Amazon Web Services 区域，请使用此策略。

- 创建自定义策略

  如果您要明确指定部署核心的 Amazon Web Services 区域，则应创建自定义策略。以下示例策略允许访问六个区域中的 AWS IoT Greengrass 软件更新。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToGreengrassOTAUpdateArtifacts",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::us-east-1-greengrass-updates/*",
                "arn:aws:s3:::us-west-2-greengrass-updates/*",
                "arn:aws:s3:::ap-northeast-1-greengrass-updates/*",
                "arn:aws:s3:::ap-southeast-2-greengrass-updates/*",
                "arn:aws:s3:::eu-central-1-greengrass-updates/*",
                "arn:aws:s3:::eu-west-1-greengrass-updates/*"
            ]
        }
    ]
}
```

信任策略

附加到角色的信任策略必须允许 `sts:AssumeRole` 操作并将 `iot.amazonaws.com` 定义为委托人。这允许 AWS IoT Core 代入该角色作为可信实体。以下是示例策略文档：

```
{
     "Version": "2012-10-17",
     "Statement": [
         {
             "Sid": "AllowIotToAssumeRole",
             "Action": "sts:AssumeRole",
             "Principal": {
                  "Service": "iot.amazonaws.com"
             },
             "Effect": "Allow"
        }
    ]
```

```
}
```

此外，启动 OTA 更新的用户必须具有使用 `greengrass:CreateSoftwareUpdateJob` 和 `iot:CreateJob` 的权限，并可以使用 `iam:PassRole` 来传递签署人角色的权限。以下为 IAM 策略示例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "greengrass:CreateSoftwareUpdateJob"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJob"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn-of-s3-url-signer-role"
        }
    ]
}
```

注意事项

在对 Greengrass 核心软件启动 OTA 更新之前，要注意该操作对 Greengrass 组中设备的影响，不仅影响核心设备，还会影响在本地连接至该核心的客户端设备：

- 核心在更新期间会关闭。

- 在核心上运行的任何 Lambda 函数都将关闭。如果这些函数写入到本地资源，除非正常关闭，否则它们可能会导致这些资源的状态不正确。
- 在核心的停机时间内，与 AWS Cloud 的所有连接都将丢失。客户端设备通过核心路由的消息将丢失。
- 凭证缓存会丢失。
- 容纳 Lambda 函数的待处理工作的队列将丢失。
- 长时间生存的 Lambda 函数将丢失其动态状态信息，并且将丢弃所有待处理工作。

OTA 更新期间保留以下状态信息：

- Core 配置文件
- Greengrass 组配置
- 本地影子
- Greengrass 日志
- OTA 更新代理日志

Greengrass OTA 更新代理

Greengrass OTA 更新代理是设备上的软件组件，可处理在云中创建和部署的更新任务。OTA 更新代理在与 AWS IoT Greengrass Core 软件相同的软件包中分发。代理位于 */greengrass-root*/ota/ota_agent/ggc-ota。它将日志写入到 /var/log/greengrass/ota/ggc_ota.txt。

> **Note**
>
> */greengrass-root* 表示在您的设备上安装 AWS IoT Greengrass 核心软件的路径。通常，这是 /greengrass 目录。

您可以通过手动执行二进制文件或通过作为 init 脚本的一部分（如 systemd 服务文件）集成来启动 OTA 更新代理。如果您手动执行二进制文件，则应以 root 用户身份运行。在启动时，OTA 更新代理从 AWS IoT Core 侦听 AWS IoT Greengrass 软件更新作业并按顺序执行它们。OTA 更新代理将忽略所有其他 AWS IoT 作业类型。

以下摘录显示了用于启动、停止和重新启动 OTA 更新代理的系统服务文件示例：

```
[Unit]
```

```
Description=Greengrass OTA Daemon

[Service]
Type=forking
Restart=on-failure
ExecStart=/greengrass/ota/ota_agent/ggc-ota

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

作为更新目标的核心不能运行 OTA 更新代理的两个实例。这样做会导致两个代理处理相同的任务，因此产生冲突。

与初始化系统集成

在 OTA 更新期间，OTA 更新代理将重新启动核心设备上的二进制文件。如果二进制文件正在运行，在更新期间，当 init 系统监控 AWS IoT Greengrass Core 软件或代理的状态时，这可能会导致冲突。为了帮助将 OTA 更新机制与您的 init 监控策略集成，您可以编写在更新前后运行的 shell 脚本。例如，您可以使用 `ggc_pre_update.sh` 脚本，在设备关闭之前备份数据或停止进程。

要告诉 OTA 更新代理运行这些脚本，您必须在 config.json 文件中包含 `"managedRespawn" : true` 标志。以下摘录显示了此设置：

```
{
   "coreThing": {
       …
   },
   "runtime": {
       …
   },
   "managedRespawn": true
       …
}
```

使用 OTA 更新的托管 Respawn

以下要求适用于 `managedRespawn` 设置为 `true` 的 OTA 更新：

- */greengrass-root*`/usr/scripts` 目录中必须存在以下 shell 脚本：
  - `ggc_pre_update.sh`

  - `ggc_post_update.sh`
  - `ota_pre_update.sh`
  - `ota_post_update.sh`
- 脚本必须返回成功的返回代码。
- 脚本必须由 root 拥有，并且只能由 root 执行。
- `ggc_pre_update.sh` 脚本必须停止 Greengrass 进程守护程序。
- `ggc_post_update.sh` 脚本必须启动 Greengrass 进程守护程序。

> **Note**
>
> 由于 OTA 更新代理管理自己的进程，因此 `ota_pre_update.sh` 和 `ota_post_update.sh` 脚本无需停止或启动 OTA 服务。

OTA 更新代理运行来自 */greengrass-root*`/usr/scripts` 的脚本。该目录树应如下所示：

```
<greengrass_root>
|-- certs
|-- config
|   |-- config.json
|-- ggc
|-- usr/scripts
|   |-- ggc_pre_update.sh
|   |-- ggc_post_update.sh
|   |-- ota_pre_update.sh
|   |-- ota_post_update.sh
|-- ota
```

如果将 `managedRespawn` 设置为 `true`，则 OTA 更新代理在软件更新前后检查 */greengrass-root*`/usr/scripts` 目录中的这些脚本。如果脚本不存在，则更新失败。AWS IoT Greengrass 不会验证这些脚本的内容。作为最佳实践，请验证您的脚本是否正常运行，并针对错误发出相应的退出代码。

对于 AWS IoT Greengrass Core 软件的 OTA 更新：

- 开始更新之前，代理运行 `ggc_pre_update.sh` 脚本。使用此脚本执行需要在 OTA 更新代理启动 AWS IoT Greengrass Core 软件更新之前运行的命令，例如备份数据或停止任何正在运行的进程。以下示例是一个用于停止 Greengrass 进程守护程序的简单脚本。

```
#!/bin/bash
set -euo pipefail
systemctl stop greengrass
```

- 完成更新后，代理运行 `ggc_post_update.sh` 脚本。使用此脚本执行在 OTA 更新代理启动 AWS IoT Greengrass Core 软件更新后需要运行的命令，例如重新启动进程。以下示例是一个启动 Greengrass 进程守护程序的简单脚本。

```
#!/bin/bash
set -euo pipefail
systemctl start greengrass
```

对于 OTA 更新代理的 OTA 更新，请执行以下操作：

- 开始更新之前，代理运行 `ota_pre_update.sh` 脚本。使用此脚本执行需要在 OTA 更新代理更新自身之前运行的命令，例如备份数据或停止任何正在运行的进程。
- 完成更新后，代理运行 `ota_post_update.sh` 脚本。使用此脚本执行需要在 OTA 更新代理更新自身后运行的命令，例如重新启动进程。

> **Note**
>
> 如果将 `managedRespawn` 设置为 `false`，则 OTA 更新代理不运行脚本。

创建 OTA 更新

按照以下步骤在一个或多个核心上执行 AWS IoT Greengrass 软件的 OTA 更新：

1. 确保您的核心满足 OTA 更新的要求。

   > **Note**
   >
   > 如果您将 init 系统配置为管理 AWS IoT Greengrass Core 软件或 OTA 更新代理，请验证核心上的以下内容：
   >
   > - config.json 文件指定 `"managedRespawn" : true`。
   > - */greengrass-root*/usr/scripts 目录包含以下脚本：

- ggc_pre_update.sh
- ggc_post_update.sh
- ota_pre_update.sh
- ota_post_update.sh

有关更多信息，请参阅the section called “与初始化系统集成”。

2. 在核心设备终端中，启动 OTA 更新代理。

```
cd /greengrass-root/ota/ota_agent
sudo ./ggc-ota
```

> **Note**
>
> *greengrass-root* 表示在您的设备上安装 AWS IoT Greengrass 核心软件的路径。通常，这是 /greengrass 目录。

不要在核心上启动 OTA 更新代理的多个实例，因为这可能会导致冲突。

3. 使用 AWS IoT Greengrass API 创建软件更新作业。

   a. 调用 CreateSoftwareUpdateJob API。在此示例过程中，我们使用 AWS CLI 命令。

   以下命令创建一个作业，用于更新核心上的 AWS IoT Greengrass Core 软件。替换示例值，然后运行命令。

   Linux or macOS terminal

```
aws greengrass create-software-update-job \
--update-targets-architecture x86_64 \
--update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] \
--update-targets-operating-system ubuntu \
--software-to-update core \
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
--update-agent-log-level WARN \
--amzn-client-token myClientToken1
```

Windows command prompt

```
aws greengrass create-software-update-job ^
--update-targets-architecture x86_64 ^
--update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] ^
--update-targets-operating-system ubuntu ^
--software-to-update core ^
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole ^
--update-agent-log-level WARN ^
--amzn-client-token myClientToken1
```

此命令将返回以下响应。

```
{
    "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "IotJobArn": "arn:aws:iot:region:123456789012:job/
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "PlatformSoftwareVersion": "1.10.1"
}
```

b. 从响应复制 `IoTJobId`。

c. 在 AWS IoT Core API 中调用 DescribeJob 以查看任务状态。将示例值替换为您的作业 ID，然后运行命令。

```
aws iot describe-job --job-id GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-
a1da0EXAMPLE
```

该命令返回一个响应对象，其中包含有关作业的信息，包括 `status` 和 `jobProcessDetails`。

```
{
    "job": {
        "jobArn": "arn:aws:iot:region:123456789012:job/
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
        "jobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
        "targetSelection": "SNAPSHOT",
        "status": "IN_PROGRESS",
        "targets": [
            "arn:aws:iot:region:123456789012:thing/myCoreDevice"
```

```
        ],
        "description": "This job was created by Greengrass to update the
 Greengrass Cores in the targets with version 1.10.1 of the core software
 running on x86_64 architecture.",
        "presignedUrlConfig": {
            "roleArn": "arn:aws::iam::123456789012:role/myS3UrlSignerRole",
            "expiresInSec": 3600
        },
        "jobExecutionsRolloutConfig": {},
        "createdAt": 1588718249.079,
        "lastUpdatedAt": 1588718253.419,
        "jobProcessDetails": {
            "numberOfCanceledThings": 0,
            "numberOfSucceededThings": 0,
            "numberOfFailedThings": 0,
            "numberOfRejectedThings": 0,
            "numberOfQueuedThings": 1,
            "numberOfInProgressThings": 0,
            "numberOfRemovedThings": 0,
            "numberOfTimedOutThings": 0
        },
        "timeoutConfig": {}
    }
}
```

有关问题排查帮助，请参阅排查问题。

CreateSoftwareUpdateJob API

您可以使用 `CreateSoftwareUpdateJob` API 更新核心设备上的 AWS IoT Greengrass Core 软件或 OTA 更新代理软件。此 API 创建一个 AWS IoT 快照作业，在更新可用时通知设备。调用 `CreateSoftwareUpdateJob` 后，您可以使用其他 AWS IoT 作业命令来跟踪软件更新。有关更多信息，请参阅 AWS IoT 开发人员指南中的作业。

以下示例介绍如何使用 AWS CLI 创建用于更新核心设备上的 AWS IoT Greengrass Core 软件的作业：

```
aws greengrass create-software-update-job \
--update-targets-architecture x86_64 \
--update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] \
--update-targets-operating-system ubuntu \
```

```
--software-to-update core \
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
--update-agent-log-level WARN \
--amzn-client-token myClientToken1
```

`create-software-update-job` 命令会返回一个 JSON 响应，其中包含作业 ID、作业 ARN 和更新所安装的软件版本：

```
{
    "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "IotJobArn": "arn:aws:iot:region:123456789012:job/GreengrassUpdateJob_c3bd7f36-
ee80-4d42-8321-a1da0EXAMPLE",
    "PlatformSoftwareVersion": "1.9.2"
}
```

有关演示如何使用 `create-software-update-job` 更新核心设备的步骤，请参阅the section called “创建 OTA 更新”。

`create-software-update-job` 命令包含以下参数：

`--update-targets-architecture`

核心设备的架构。

有效值：`armv7l`、`armv6l`、`x86_64` 或 `aarch64`

`--update-targets`

要更新的核心。该列表可以包含单个核心的 ARN 和其成员为核心的事物组的 ARN。有关事物组的更多信息，请参阅AWS IoT开发人员指南中的静态事物组。

`--update-targets-operating-system`

核心设备的操作系统。

有效值：`ubuntu`、`amazon_linux`、`raspbian` 或 `openwrt`

`--software-to-update`

指定应该更新核心的软件还是 OTA 更新代理软件。

有效值：`core` 或 `ota_agent`

`--s3-url-signer-role`

用于预签名 Amazon S3 URL（链接至 AWS IoT Greengrass 软件更新构件）的 IAM 角色的 ARN。角色的附加权限策略必须允许对目标 AWS 区域 中的存储桶执行 `s3:GetObject` 操作。角色还必须允许 `iot.amazonaws.com` 代入角色作为可信实体。有关更多信息，请参阅the section called “OTA 更新的 IAM 权限”。

`--amzn-client-token`

（可选）用于进行幂等请求的客户端令牌。请提供一个唯一令牌，以防止因内部重试而创建重复更新。

`--update-agent-log-level`

（可选）OTA 更新代理生成的日志语句的日志记录级别。默认为 `ERROR`。

有效值：`NONE`、`TRACE`、`DEBUG`、`VERBOSE`、`INFO`、`WARN`、`ERROR` 或 `FATAL`

> **Note**
>
> `CreateSoftwareUpdateJob` 仅接受对以下受支持的架构和操作系统组合的请求：
>
> - ubuntu/x86_64
> - ubuntu/aarch64
> - amazon_linux/x86_64
> - raspbian/armv7l
> - raspbian/armv6l
> - openwrt/aarch64
> - openwrt/armv7l

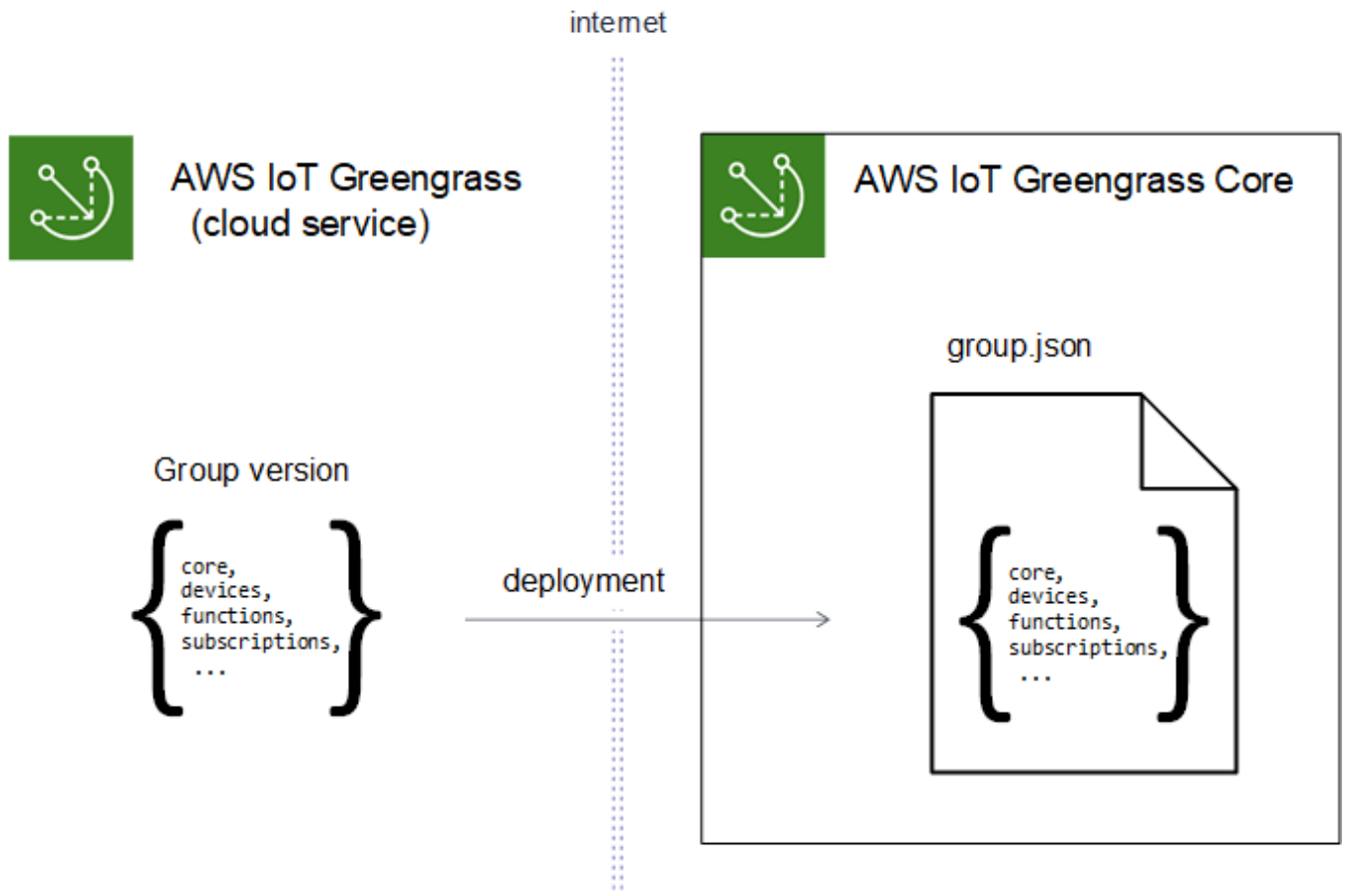
将 AWS IoT Greengrass 组部署到 AWS IoT Greengrass 核心

使用 AWS IoT Greengrass 组用于组织边缘环境中的实体。您也可以使用组来控制组中的实体如何相互交互并与 AWS Cloud 交互。例如，只针对本地运行部署组中的 Lambda 函数，仅组中的设备可以使用本地 MQTT 服务器进行通信。

一个组必须包含一个核心，它是一个运行 AWS IoT Greengrass 核心软件的 AWS IoT 设备。网关作为边缘核心，可在边缘环境中提供 AWS IoT Core 功能。根据您的业务需求，还可以向组中添加以下实体：

- 客户端设备。在 AWS IoT 注册表中表示为事物。这些设备必须运行 FreeRTOS 或使用 AWS IoT 设备开发工具包或AWS IoT Greengrass发现 API 来获取核心的连接信息。只有作为组成员的客户端设备才可以连接到核心。
- Lambda 函数。在核心上运行代码的用户定义的无服务器应用程序。Lambda 函数是在 AWS Lambda 中编写并从 Greengrass 组引用的。有关更多信息，请参阅 运行本地 Lambda 函数。
- 连接器。在核心上运行代码的预定义无服务器应用程序。连接器可提供与本地基础设施、设备协议、AWS 以及其他云服务的内置集成。有关更多信息，请参阅 使用连接器与服务和协议集成。
- 订阅。定义有权进行 MQTT 通信的发布者、订阅者和 MQTT 主题（或对象）。
- 资源。引用本地设备和卷、机器学习模型和密钥（用于通过 Greengrass Lambda 函数和连接器进行访问控制）。
- 日志。AWS IoT Greengrass 系统组件和 Lambda 函数的日志记录配置。有关更多信息，请参阅 the section called “利用 AWS IoT Greengrass 日志进行监控”。

您可以管理 AWS Cloud 中的 Greengrass 组，然后将其部署到核心。部署会将组配置复制到核心设备上的 `group.json` 文件。此文件位于 *`greengrass-root`*`/ggc/deployments/group` 中：



> **Note**
>
> 在部署期间，核心设备上的 Greengrass 守护程序进程停止，然后重新启动。

从 AWS IoT 控制台中部署组

您可以从 AWS IoT 控制台中的组配置页面中部署组并管理其部署。

> **Note**
>
> 要在控制台中打开此页面，请依次选择 Greengrass 设备和组 (V1)，然后在 Greengrass 组下选择您的组。

部署当前版本的组

- 在组配置页面上，选择 部署。

查看组的部署历史记录

组的部署历史记录包括每次部署尝试的日期和时间、组版本和状态。

1. 在组配置页面中，选择部署选项卡。
2. 要查看有关部署的更多信息（包括错误消息），请在 AWS IoT 控制台的 Greengrass 设备下选择部署。

重新部署组部署

如果当前部署失败，您可能希望重新部署一个部署，或者恢复为不同的组版本。

1. 在 AWS IoT 控制台中，选择 Greengrass 设备，然后选择 组 (V1)。
2. 选择 部署 选项卡。
3. 选择要重新部署的部署，然后选择重新部署。

重置组部署

您可能希望重置组部署以移动或删除组或删除部署信息。有关更多信息，请参阅 the section called “重置部署”。

1. 在 AWS IoT 控制台中，选择 Greengrass 设备，然后选择 组 (V1)。
2. 选择 部署 选项卡。
3. 选择要重置的部署，然后选择重置部署。

使用 AWS IoT Greengrass API 部署组

AWS IoT Greengrass API 提供了以下操作，可用于部署 AWS IoT Greengrass 组和管理组部署。您可以从 AWS CLI、AWS IoT Greengrass API 或 AWS 开发工具包中调用这些操作。

操作	描述
CreateDeployment	创建 `NewDeployment` 或 `Redeployment` 部署。

操作	描述
	如果当前部署失败，您可能希望重新部署一个部署。或者，您可能希望重新部署，以恢复到另一个组版本。
GetDeploymentStatus	返回部署的状态：`Building`、`InProgress` 、`Success` 或 `Failure`。 您可以将 Amazon EventBridge 事件配置为接收部署通知。有关更多信息，请参阅 the section called “获取部署通知”。
ListDeployments	返回组的部署历史记录。
ResetDeployments	重置组的部署。 您可能希望重置组部署以移动或删除组或删除部署信息。有关更多信息，请参阅 the section called “重置部署”。

> **Note**
>
> 有关批量部署操作的更多信息，请参阅 the section called “创建批量部署”。

获取组 ID

组 ID 通常用于 API 操作中。您可以使用ListGroups操作从群组列表中查找目标群组的 ID。例如，在 AWS CLI 中，使用 `list-groups` 命令。

```
aws greengrass list-groups
```

您还可以包含用于筛选结果的 `query` 选项。例如：

- 要获取最近创建的组，请执行以下操作：

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

- 按名称获取组：

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

系统不要求组名称是唯一的，所以可能会返回多个组。

以下为 `list-groups` 响应示例。每个组的信息都包含组 ID（在 `Id` 属性中）和最新组版本 ID（在 `LatestVersion` 属性中）。要获取群组的其他版本 ID，请使用群组 ID 和ListGroupVersions。

> **Note**
>
> 您还可以在 AWS IoT 控制台中找到这些值。组 ID 显示在组的设置页面上。组版本 ID 显示在组的部署选项卡上。

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
```

```
        ...
    ]
}
```

如果您不指定 AWS 区域，AWS CLI 命令将使用您的配置文件中的默认区域。要返回不同区域中的组，请包含##选项。例如：

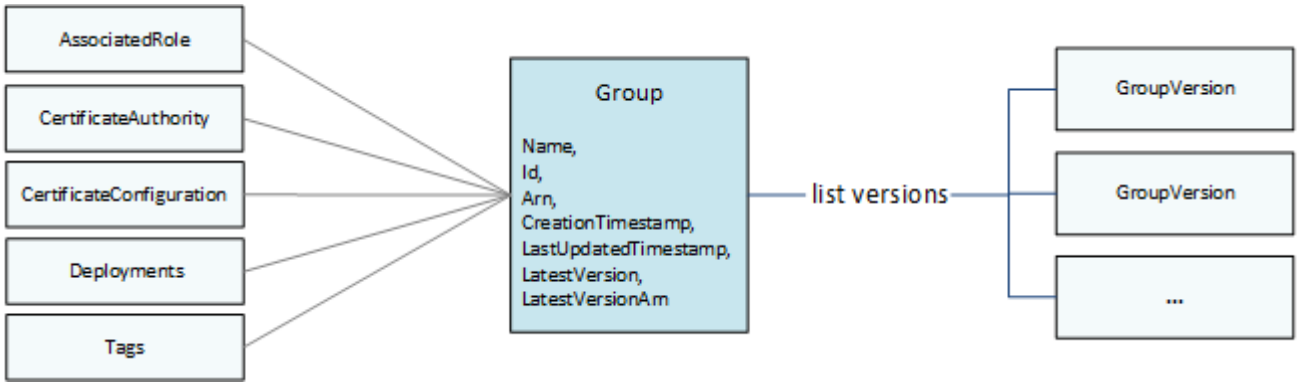
```
aws greengrass list-groups --region us-east-1
```

AWS IoT Greengrass 组对象模型概述

当使用 AWS IoT Greengrass API 编程时，了解 Greengrass 组对象模型很有帮助。

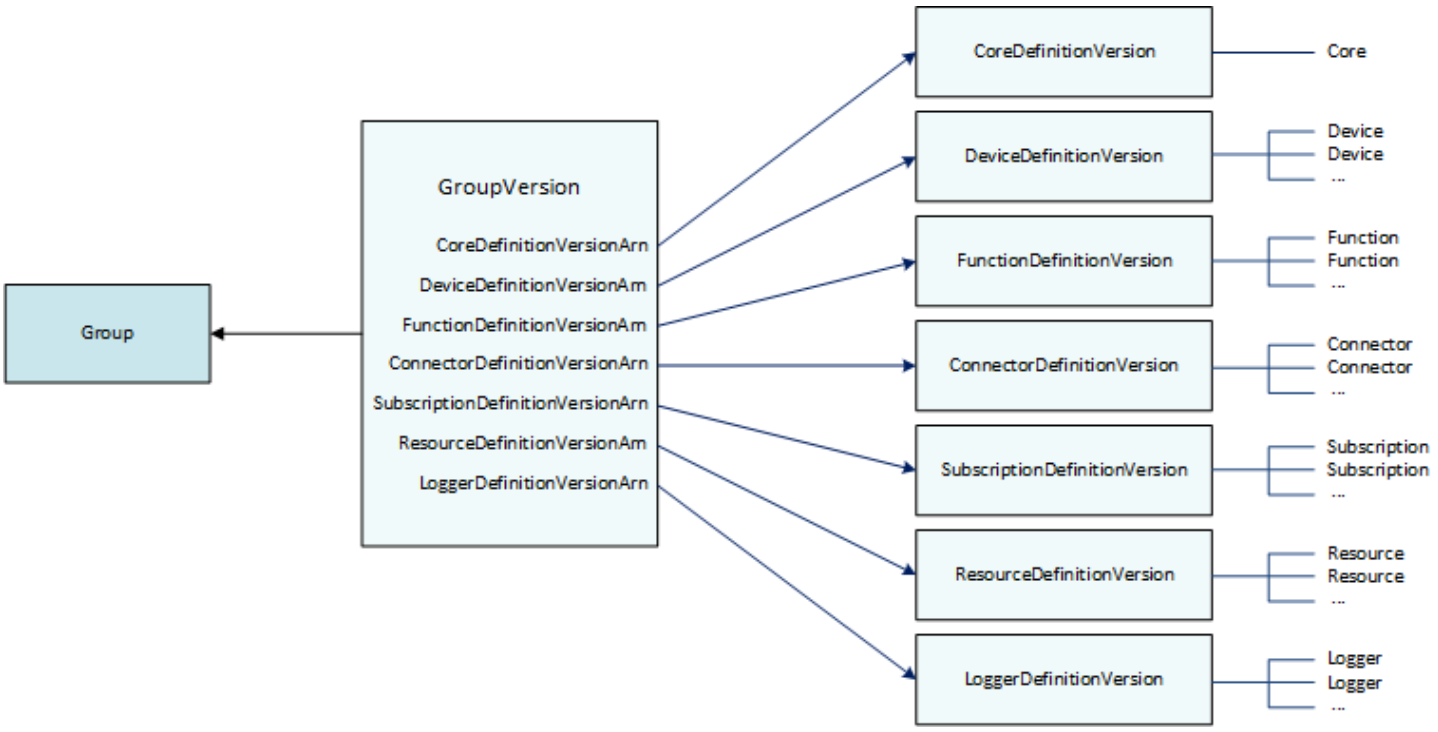
组

在 AWS IoT Greengrass API 中，顶级 `Group` 对象包含元数据和 `GroupVersion` 对象的列表。`GroupVersion` 对象通过 ID 与 `Group` 关联。



组版本

`GroupVersion` 对象可定义组成员资格。每个 `GroupVersion` 通过 ARN 引用一个 `CoreDefinitionVersion` 和其他组件版本。这些引用可确定组中要包括哪些实体。



例如，要在组中包含三个 Lambda 函数、一个设备和两个订阅，`GroupVersion` 引用：

- 包含所需核心的 `CoreDefinitionVersion`。
- 包含三个函数的 `FunctionDefinitionVersion`。
- 包含客户端设备的 `DeviceDefinitionVersion`。
- 包含两个订阅的 `SubscriptionDefinitionVersion`。

部署到核心设备的 `GroupVersion` 决定了本地环境中的可用实体及其它们如何进行交互。

组组件

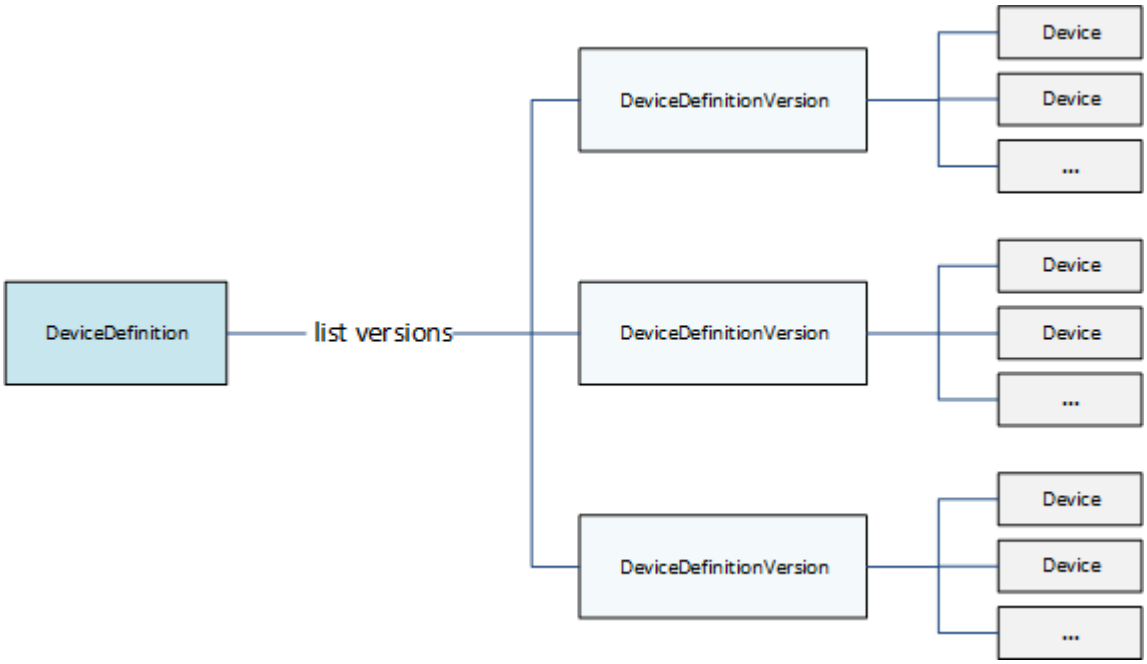
添加到组中的组件具有三级层次结构：

- 引用给定类型DefinitionVersion对象列表的定义。例如，`DeviceDefinition` 引用 `DeviceDefinitionVersion` 对象的列表。
- DefinitionVersion包含一组给定类型的实体的类型。例如，`DeviceDefinitionVersion` 包含 `Device` 对象的列表。
- 用于定义其属性和行为的各个实体。例如，`Device` 用于定义 AWS IoT 注册表中相应客户端设备的 ARN、其设备证书的 ARN，以及其本地影子是否自动与云同步。

  您可以向组中添加以下类型的实体：

- 连接器
- 核心实例
- 设备
- 函数
- Logger
- 资源
- 订阅

以下示例 DeviceDefinition 会引用三个 DeviceDefinitionVersion 对象，每个都包含多个 Device 对象。一个组中一次只使用一个 DeviceDefinitionVersion。



更新组

在 AWS IoT Greengrass API 中，您可以使用多个版本更新组的配置。版本是不可变的，因此要添加、移除或更改组组件，必须创建包含新的或更新的实DefinitionVersion体的对象。

您可以将新DefinitionVersions对象与新的或现有的定义对象相关联。例如，您可以使用 CreateFunctionDefinition 操作来创建 FunctionDefinition（其中包含作为初始版本的 FunctionDefinitionVersion），也可以使用 CreateFunctionDefinitionVersion 操作并引用现有 FunctionDefinition。

创建组组件后，您将创建一个GroupVersion包含要包含在组中的所有DefinitionVersion对象。然后，部署 GroupVersion。

要部署 GroupVersion，则必须引用包含一个 Core 的 CoreDefinitionVersion。所有引用的实体必须是组的成员。此外，Greengrass 服务角色必须与正在部署 GroupVersion 的 AWS 区域 中的 AWS 账户 关联。

> **Note**
>
> API 中的 Update 操作用于更改 Group 或组件 Definition 对象的名称。

更新引用 AWS 资源的实体

Greengrass Lambda 函数和密钥资源会定义 Greengrass 特定属性，并引用相应的 AWS 资源。要更新这些实体，您可以更改相应的 AWS 资源，而不是 Greengrass 对象。例如，Lambda 函数会引用 AWS Lambda 中的一个函数，并且定义特定于 Greengrass 组的生命周期和其他属性。

- 要更新 Lambda 函数代码或打包依赖项，请在 AWS Lambda 中进行更改。在检索下一组部署的过程中，会从 AWS Lambda 中检索这些更改并将其复制到您的本地环境中。
- 要更新 Greengrass 特定的属性，您可以创建一个包含已更新 Function 属性的 FunctionDefinitionVersion。

> **Note**
>
> Greengrass Lambda 函数可以通过别名 ARN 或版本 ARN 引用 Lambda 函数。如果您引用别名 ARN（推荐），则在 AWS Lambda 中发布新的函数版本时不需要更新 FunctionDefinitionVersion（或 SubscriptionDefinitionVersion）。有关更多信息，请参阅 the section called “按别名或版本引用函数”。

另请参阅

- the section called “获取部署通知”
- the section called “重置部署”
- the section called “创建批量部署”
- 排查部署问题

- AWS IoT Greengrass Version 1 API Reference
- AWS CLI 命令参考中的AWS IoT Greengrass 命令

获取部署通知

利用 Amazon EventBridge 事件规则，您可以获取关于 Greengrass 组部署状态更改的通知。EventBridge 提供近乎实时的系统事件流，这些系统事件描述了 AWS 资源的更改。AWS IoT Greengrass 会将这些事件发送到 EventBridge 至少一次。这意味着 AWS IoT Greengrass 可能会发送给定事件的多个副本来确保传输成功。此外，事件侦听器可能无法按事件的发生顺序接收事件。

> **Note**
>
> Amazon EventBridge 是一种事件总线服务，您可以用其轻松地将应用程序与来自各种来源的数据相连接，例如 Greengrass 核心设备和部署通知。有关更多信息，请参阅 Amazon EventBridge 用户指南中的什么是 Amazon EventBridge？。

组部署状态改变时 AWS IoT Greengrass 会触发一个事件。您可以创建一个 EventBridge 规则，用于在发生所有状态转换或转换为您指定的状态时运行。当部署进入将启动规则的状态时，EventBridge 会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。

部署进入以下状态时 AWS IoT Greengrass 会触发一个事件：`Building`、`InProgress`、`Success` 和 `Failure`。

> **Note**
>
> 目前不支持对批量部署操作状态的监控。但是，AWS IoT Greengrass 会为构成批量部署的各个组部署触发状态更改事件。

组部署状态更改事件

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 Deployment Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "time":"2018-03-22T00:38:11Z",
    "region":"us-west-2",
    "resources":[],
    "detail":{
        "group-id": "284dcd4e-24bc-4c8c-a770-EXAMPLEf03b8",
        "deployment-id": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
        "deployment-type": "NewDeployment|Redeployment|ResetDeployment|
ForceResetDeployment",
        "status": "Building|InProgress|Success|Failure"
    }
}
```

您可以创建适用于一个或多个组的规则。您可以按以下一种或多种部署类型和部署状态筛选规则：

部署类型

- `NewDeployment`. 组版本的第一次部署。
- `ReDeployment`. 组版本的重新部署。
- `ResetDeployment`. 删除存储在 AWS Cloud 和 AWS IoT Greengrass 核心中的部署信息。有关更多信息，请参阅the section called “重置部署”。
- `ForceResetDeployment`. 删除存储在 AWS Cloud 中的部署信息并报告成功，而无需等待核心进行响应。如果核心已连接或当下次连接时，会同时删除存储在核心中的部署信息。

部署状态

- `Building`。AWS IoT Greengrass 正在验证组配置并构建部署构件。
- `InProgress`. 正在 AWS IoT Greengrass 核心上进行部署。
- `Success`. 部署成功。
- `Failure`. 部署失败。

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。

> **Note**
>
> AWS IoT Greengrass 不使用 `resources` 属性，因此该属性始终为空。

创建 EventBridge 规则的先决条件

在为 AWS IoT Greengrass 创建 EventBridge 规则之前，请先执行以下操作：

- 熟悉 EventBridge 中的事件、规则和目标。
- 创建和配置由 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 Amazon EventBridge 用户指南中的什么是 Amazon EventBridge？和 Amazon EventBridge 入门。

配置部署通知（控制台）

使用以下步骤创建一个 EventBridge 规则，此规则会在一个组的部署状态更改时发布一个 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 Amazon EventBridge 控制台。
2. 在导航窗格中，选择 Rules (规则)。
3. 选择 Create rule (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 Event bus（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。当您账户中的某个 AWS 服务发出一个事件时，它始终会发送到您账户的默认事件总线。
6. 对于 Rule type（规则类型），选择 Rule with an event pattern（具有事件模式的规则）。

7. 选择 Next（下一步）。
8. 对于 Event source（事件源），选择 AWS services（服务）。
9. 在事件模式中，选择 AWS服务。
10. 对于 AWS 服务，选择 Greengrass。
11. 对于 Event type (事件类型)，选择 Greengrass Deployment Status Change (Greengrass 部署状态更改)。

> **Note**
>
> 通过 CloudTrail 进行的 AWS API 调用 事件类型基于 AWS IoT Greengrass 与 AWS CloudTrail 的集成。可以使用此选项创建由对 AWS IoT Greengrass API 进行的读取或写入调用启动的规则。有关更多信息，请参阅the section called “使用 AWS CloudTrail 记录 AWS IoT Greengrass API 调用”。

12. 选择将启动通知的部署状态。
    - 要接收所有状态更改事件的通知，请选择 Any state (任何状态)。
    - 要仅接收某些状态更改事件的通知，请选择 Specific state(s) (特定状态)，然后选择目标状态。
13. 选择将启动通知的部署类型。
    - 要接收所有部署类型的通知，请选择 Any state (任何状态)。
    - 要仅接收某些部署类型的通知，请选择 Specific state(s) (特定状态)，然后选择目标部署类型。
14. 选择 Next（下一步）。
15. 对于 Target types（目标类型），选择 AWS service（服务）。
16. 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 Next（下一步）。
17. 在 标签 下，定义规则的标签或将字段留空。
18. 选择 Next（下一步）。
19. 查看规则详细信息并选择 Create rule（创建规则）。

配置部署通知 (CLI)

使用以下步骤创建一个 EventBridge 规则，此规则会在一个组的部署状态更改时发布一个 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。

1. 创建 规则。

   - 将 *group-id* 替换为 AWS IoT Greengrass 组的 ID。

   ```
   aws events put-rule \
     --name TestRule \
     --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"group-id\":
    [\"group-id\"]}}"
   ```

   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，您必须将基于资源的策略添加到您的主题中。有关更多信息，请参阅 Amazon EventBridge 用户指南中的 Amazon SNS 权限。

有关更多信息，请参阅 Amazon EventBridge 用户指南中的 EventBridge 中的事件和事件模式。

配置部署通知 (AWS CloudFormation)

使用 AWS CloudFormation 模板可创建 EventBridge 规则，这些规则将发送有关 Greengrass 组部署状态更改的通知。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

- 部署 AWS IoT Greengrass 组
- Amazon EventBridge 用户指南中的什么是 Amazon EventBridge？。

重置部署

此功能适用于 AWS IoT Greengrass Core v1.1 及更高版本。

您可能希望重置组的部署以执行以下操作：

- 删除组，例如当您要将组的核心移至另一个组时，或者组的核心已重新制作了映像时。在删除组之前，您必须重置该组的部署，以便将核心与另一个 Greengrass 组结合使用。
- 将组的核心移到不同的组。
- 将组恢复到任何部署之前的状态。
- 从核心设备删除部署配置。
- 从核心设备或云中删除敏感数据。
- 将新的组配置部署到一个核心而不必将该核心替换成当前组中的另一个核心。

> **Note**
>
> AWS IoT Greengrass 核心软件 v1.0.0 中不支持重置部署功能。您不能删除已使用 v1.0.0 部署的组。

重置部署操作首先会清除存储在云中的指定组的所有部署信息。然后，它将指示该组的核心设备也清理其所有部署相关部署信息（Lambda 函数、用户日志、影子数据库和服务器证书，但不包括用户定义的 `config.json` 或 Greengrass 核心证书。）如果一个组目前的部署状态为 `In Progress` 或 `Building`，则无法为其启动部署重置。

从 AWS IoT 控制台中重置部署

您可以在 AWS IoT 控制台中的组配置页面中重置组部署。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择目标组。

3. 在 部署 选项卡中，选择 重置部署。
4. 在重置此 Greengrass 组的部署对话框中，键入 **confirm** 以表示同意，然后选择重置部署。

使用 AWS IoT Greengrass API 重置部署

您可以使用 AWS CLI、AWS IoT Greengrass API 或 AWS 开发工具包中的 `ResetDeployments` 操作来重置部署。本主题中的示例使用 CLI。

```
aws greengrass reset-deployments --group-id GroupId [--force]
```

`reset-deployments` CLI 命令的参数：

`--group-id`

组 ID。使用 `list-groups` 命令以获取此值。

`--force`

可选。如果组的核心设备丢失、被盗或损毁，则使用此参数。该选项可使重置部署过程在云中的所有部署信息都清理完后即报告成功，而不必等待核心设备响应。但是，如果核心设备处于或变为活动状态，它还会执行清理操作。

`reset-deployments` CLI 命令的输出如下所示：

```
{
    "DeploymentId": "4db95ef8-9309-4774-95a4-eea580b6ceef",
    "DeploymentArn": "arn:aws:greengrass:us-west-2:106511594199:/greengrass/groups/
b744ed45-a7df-4227-860a-8d4492caa412/deployments/4db95ef8-9309-4774-95a4-eea580b6ceef"
}
```

可以使用 `get-deployment-status` CLI 命令查看重置部署的状态：

```
aws greengrass get-deployment-status --deployment-id DeploymentId --group-id GroupId
```

`get-deployment-status` CLI 命令的参数：

`--deployment-id`

部署 ID。

`--group-id`

组 ID。

`get-deployment-status` CLI 命令的输出如下所示：

```
{
    "DeploymentStatus": "Success",
    "UpdatedAt": "2017-04-04T00:00:00.000Z"
}
```

当重置部署正在准备时，`DeploymentStatus` 设置为 `Building`。当重置部署已就绪而 AWS IoT Greengrass 核心未选择重置部署时，`DeploymentStatus` 为 `InProgress`。

如果重置操作失败，将在响应中返回错误信息。

另请参阅

- 部署 AWS IoT Greengrass 组
- ResetDeployments 在 AWS IoT Greengrass Version 1API 参考中
- GetDeploymentStatus在 AWS IoT Greengrass Version 1API 参考中

为组创建批量部署

您可以使用简单 API 调用一次部署大量 Greengrass 组。这些部署通过具有固定上限的自适应速率触发。

本教程介绍如何使用 AWS CLI 在 AWS IoT Greengrass 中创建和监控批量组部署。本教程中的批量部署示例包含多个组。您可以在您的实施中使用该示例根据需要添加任意数量的组。

本教程包含以下概括步骤：

1. 创建并上传批量部署输入文件
2. 创建并配置 IAM 执行角色用于批量部署
3. 允许您的执行角色访问您的 S3 存储桶
4. 部署组
5. 测试部署

先决条件

要完成此教程，需要：

- 一个或多个可部署的 Greengrass 组。有关创建 AWS IoT Greengrass 组和核心的更多信息，请参阅入门 AWS IoT Greengrass。
- 在您的计算机上安装并配置 AWS CLI。有关信息，请参阅 AWS CLI 用户指南。
- 在 与 AWS IoT Greengrass 所在的同一 AWS 区域 中创建的一个 S3 存储桶。有关信息，请参阅 Amazon Simple Storage Service 用户指南中的创建和配置 S3 存储桶。

> **Note**
>
> 目前，不支持启用了 SSE KMS 的存储桶。

步骤 1：创建并上传批量部署输入文件

在此步骤中，您将创建一个部署输入文件并将其上传到您的 Amazon S3 存储桶。此文件是一个行分隔的序列化 JSON 文件，包含有关批量部署中的每个组的信息。AWS IoT Greengrass 在初始化批量组部署时使用此信息代表您部署每个组。

1. 运行以下命令以获取要部署的每个组的 `groupId`。在批量部署输入文件中输入 `groupId`，让 AWS IoT Greengrass 可以识别要部署的每个组。

   > **Note**
   >
   > 您还可以在 AWS IoT 控制台中找到这些值。组 ID 显示在组的设置页面上。组版本 ID 显示在组的部署选项卡上。

   ```
   aws greengrass list-groups
   ```

   该响应包含有关您 AWS IoT Greengrass 账户中的每个组的信息：

   ```
   {
   ```

```
  "Groups": [
    {
      "Name": "string",
      "Id": "string",
      "Arn": "string",
      "LastUpdatedTimestamp": "string",
      "CreationTimestamp": "string",
      "LatestVersion": "string",
      "LatestVersionArn": "string"
    }
  ],
  "NextToken": "string"
}
```

运行以下命令以获取要部署的每个组的 groupVersionId。

```
list-group-versions --group-id groupId
```

该响应包含有关组中的所有版本的信息。记下要使用的组版本的 Version 值。

```
{
  "Versions": [
    {
      "Arn": "string",
      "Id": "string",
      "Version": "string",
      "CreationTimestamp": "string"
    }
  ],
  "NextToken": "string"
}
```

2. 在您的计算机终端或所选的编辑器中，从以下示例创建一个文件 *MyBulkDeploymentInputFile*。此文件包含有关要包含在批量部署中的每个 AWS IoT Greengrass 组的信息。虽然此示例定义多个组，但在本教程中，您的文件只能包含一个组。

> **Note**
>
> 此文件的大小必须小于 100 MB。

```
{"GroupId":"groupId1", "GroupVersionId":"groupVersionId1",
 "DeploymentType":"NewDeployment"}
{"GroupId":"groupId2", "GroupVersionId":"groupVersionId2",
 "DeploymentType":"NewDeployment"}
{"GroupId":"groupId3", "GroupVersionId":"groupVersionId3",
 "DeploymentType":"NewDeployment"}
...
```

每个记录（或行）包含一个组对象。每个组对象包含其相应 `GroupId`、`GroupVersionId` 和 `DeploymentType`。目前，AWS IoT Greengrass 仅支持 `NewDeployment` 批量部署类型。

保存并关闭该文件。记下该文件的位置。

3. 在您的终端中使用以下命令将输入文件上传到 Amazon S3 存储桶。将文件路径替换为该文件的位置和名称。有关信息，请参阅将对象添加到存储桶。

```
aws s3 cp path/MyBulkDeploymentInputFile s3://my-bucket/
```

步骤 2：创建并配置 IAM 执行角色

在此步骤中，您将使用 IAM 控制台创建一个独立的执行角色。然后，在该角色与 AWS IoT Greengrass 之间建立信任关系，并确保您的 IAM 用户对该执行角色具有 `PassRole` 权限。这使 AWS IoT Greengrass 可以担任该执行角色并代表您创建部署。

1. 使用以下策略创建一个执行角色。此策略文档允许 AWS IoT Greengrass 在代表您创建每个部署时访问批量部署输入文件。

   有关创建 IAM 角色及委派权限的更多信息，请参阅创建 IAM 角色。

```
{
   "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "greengrass:CreateDeployment",
            "Resource": [
                "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId1",
                "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId2",
                "arn:aws:greengrass:region:accountId:/greengrass/groups/groupId3",
                ...
            ]
        }
    ]
}
```

> **Note**
>
> 此策略必须在批量部署输入文件中包含要由 AWS IoT Greengrass 部署的每个组或组版本的资源。要允许访问所有组，请为 `Resource` 指定一个星号：
>
> ```
> "Resource": ["*"]
> ```

2. 修改该执行角色的信任关系以包含 AWS IoT Greengrass。这允许 AWS IoT Greengrass 使用该执行角色及向其附加的权限。有关信息，请参阅编辑现有角色的信任关系。

   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。有关混淆代理问题的更多信息，请参阅防止跨服务混淆代理。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
```

```
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        }
      }
    }
  ]
}
```

3. 为 IAM 用户赋予执行角色的 IAM PassRole 权限。此 IAM 用户是用于启动批量部署的用户。PassRole 权限允许您的 IAM 用户将该执行角色传递给 AWS IoT Greengrass 以供使用。有关更多信息，请参阅向用户授予将角色传递给 AWS 服务的权限。

   使用以下示例更新附加到您的执行角色的 IAM 策略。根据需要修改此示例。

```
 {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1508193814000",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:user/executionRoleArn"
            ]
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "greengrass.amazonaws.com"
                }
            }
        }
    ]
}
```

步骤 3：允许您的执行角色访问您的 S3 存储桶

要启动批量部署，您的执行角色必须能够从 Amazon S3 存储桶读取批量部署输入文件。将以下示例策略附加到 Amazon S3 存储桶，这样其 `GetObject` 权限便可供您的执行角色访问。

有关更多信息，请参阅[如何添加 S3 存储桶策略？]

```
{
    "Version": "2008-10-17",
    "Id": "examplePolicy",
    "Statement": [
        {
            "Sid": "Stmt1535408982966",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "executionRoleArn"
                ]
            },
            "Action": "s3:GetObject",
            "Resource":
            "arn:aws:s3:::my-bucket/objectKey"
        }
    ]
}
```

您可以在终端中使用以下命令检查存储桶的策略：

```
aws s3api get-bucket-policy --bucket my-bucket
```

> **Note**
>
> 您可以直接修改您的执行角色，改为向该角色授予对 Amazon S3 存储桶的 `GetObject` 权限。为此，请将以下示例策略附加到您的执行角色。
>
> ```
> {
>     "Version": "2012-10-17",
>     "Statement": [
>         {
> ```

```
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::my-bucket/objectKey"
        }
    ]
}
```

步骤 4：部署组

在此步骤中，您将对批量部署输入文件中配置的所有组版本启动一个批量部署操作。每个组版本的部署操作均为 NewDeploymentType 类型。

> **Note**
>
> 当同一账户中的其他批量部署仍在运行时，您无法调用 StartBulkDeployment。请求被拒绝。

1. 使用以下命令启动批量部署。

   我们建议您在每个 StartBulkDeployment 请求中包含 X-Amzn-Client-Token 令牌。这些请求在令牌和请求参数方面是幂等的。此令牌可以是唯一的、区分大小写的且最多不超过 64 个 ASCII 字符的任意字符串。

   ```
   aws greengrass start-bulk-deployment --cli-input-json "{
            "InputFileUri":"URI of file in S3 bucket",
            "ExecutionRoleArn":"ARN of execution role",
            "AmznClientToken":"your Amazon client token"
            }"
   ```

   该命令应生成成功状态代码 200，以及以下响应：

   ```
   {
     "bulkDeploymentId": UUID
   }
   ```

记下批量部署 ID。它可用于检查批量部署的状态。

> **Note**
>
> 虽然目前不支持批量部署操作，但您可以创建 Amazon EventBridge 事件规则以获取有关各个组部署状态更改的通知。有关更多信息，请参阅the section called “获取部署通知”。

2. 使用以下命令检查批量部署的状态。

```
aws greengrass get-bulk-deployment-status --bulk-deployment-id 1234567
```

除了 JSON 信息负载之外，该命令还应返回一个成功状态代码 200：

```
 {
  "BulkDeploymentStatus": Running,
  "Statistics": {
     "RecordsProcessed": integer,
     "InvalidInputRecords": integer,
     "RetryAttempts": integer
  },
  "CreatedAt": "string",
  "ErrorMessage": "string",
  "ErrorDetails": [
    {
      "DetailedErrorCode": "string",
      "DetailedErrorMessage": "string"
    }
  ]
}
```

BulkDeploymentStatus 包含批量执行的当前状态。该执行可具有六个不同的状态之一：

- Initializing. 批量部署请求已收到，并且执行正在准备启动。
- Running. 批量部署执行已启动。
- Completed. 批量部署执行已完成所有记录的处理。
- Stopping. 批量部署执行已收到停止命令，并将很快终止。当之前的部署处于 Stopping 状态时，您无法启动新的批量部署。

- Stopped. 批量部署执行已手动停止。
- Failed. 批量部署执行遇到了错误并终止。您可以在 ErrorDetails 字段中找到错误详细信息。

JSON 负载还包含有关批量部署进度的统计信息。您可以使用此信息确定多少个组已处理以及多少个组已失败。统计信息包含：

- RecordsProcessed：已尝试的组记录的数量。
- InvalidInputRecords：返回了不可重试错误的记录总数。例如，如果输入文件中的组记录使用无效格式或指定不存在的组版本，或者执行未授予部署组或组版本的权限，则可能会发生此错误。
- RetryAttempts：返回了可重试错误的部署尝试次数。例如，如果部署组的尝试返回一个限流错误，则将触发重试。一个组部署最多可重试五次。

如果批量部署执行失败，则此负载还包含一个 ErrorDetails 节，可用于排查问题。该节包含有关执行失败原因的信息。

您可以定期检查批量部署的状态，以确认它正在正常进行处理。在部署完成后，RecordsProcessed 应等于批量部署输入文件中的部署组的数量。这表示每个记录均已得到处理。

步骤 5：测试部署

使用 ListBulkDeployments 命令找到批量部署的 ID。

```
aws greengrass list-bulk-deployments
```

此命令会返回所有批量部署的列表（从新到旧），包括 BulkDeploymentId。

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": 1234567,
      "BulkDeploymentArn": "string",
```

```
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

现在，调用 ListBulkDeploymentDetailedReports 命令以收集有关每个部署的详细信息。

```
aws greengrass list-bulk-deployment-detailed-reports --bulk-deployment-id 1234567
```

该命令应返回一个成功状态代码 200 以及 JSON 信息负载：

```
{
  "BulkDeploymentResults": [
    {
      "DeploymentId": "string",
      "GroupVersionedArn": "string",
      "CreatedAt": "string",
      "DeploymentStatus": "string",
      "ErrorMessage": "string",
      "ErrorDetails": [
        {
          "DetailedErrorCode": "string",
          "DetailedErrorMessage": "string"
        }
      ]
    }
  ],
  "NextToken": "string"
}
```

此负载通常包含每个部署及其部署状态的分页列表（从新到旧）。此外，还包含批量部署执行失败时的更多信息。同样，列出的部署总数应等于批量部署输入文件中标识的组的数量。

返回的信息可能会有所有变化，直到部署处于最终状态（成功或失败）。您可以在此之前定期调用此命令。

批量部署问题排查

如果批量部署未成功，您可以尝试以下问题排查步骤。在您的终端中运行命令。

排查输入文件错误

如果批量部署输入文件中存在语法错误，则批量部署可能会失败。这会返回批量部署状态 Failed，并显示一条错误消息，指出第一个验证错误的行号。有四个可能的错误：

- ```
  InvalidInputFile: Missing GroupId at line number: line number
  ```

  此错误指出给定输入文件行无法注册指定的参数。可能缺少参数的是 GroupId 和 GroupVersionId。

- ```
  InvalidInputFile: Invalid deployment type at line number : line number. Only valid
   type is 'NewDeployment'.
  ```

  此错误指出给定输入文件行列出了无效的部署类型。此时，唯一受支持的部署类型为 NewDeployment。

- ```
  Line %s is too long in S3 File. Valid line is less than 256 chars.
  ```

  此错误指出给定输入文件行过长，必须缩短。

- ```
  Failed to parse input file at line number: line number
  ```

  此错误指出给定输入文件行被视为无效 json。

检查是否存在并发批量部署

当其他部署仍在运行或处于非最终状态时，您无法启动新的批量部署。这可能会导致 Concurrent Deployment Error。您可以使用 ListBulkDeployments 命令来验证批量部署当前是否正在运行。此命令列出您的批量部署（从新到旧）。

```
{
```

```
  "BulkDeployments": [
    {
      "BulkDeploymentId": BulkDeploymentId,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

使用第一个列出的批量部署的 BulkDeploymentId 运行 GetBulkDeploymentStatus 命令。如果最新批量部署处于正在运行状态（Initializing 或 Running），请使用以下命令停止批量部署。

```
aws greengrass stop-bulk-deployment --bulk-deployment-id BulkDeploymentId
```

此操作会生成 Stopping 状态，直到该部署的状态变为 Stopped。在该部署达到 Stopped 状态后，您便可启动一个新的批量部署。

检查 ErrorDetails

运行 GetBulkDeploymentStatus 命令以返回一个 JSON 负载，其中包含关于所有批量部署执行失败的详细信息。

```
  "Message": "string",
  "ErrorDetails": [
    {
      "DetailedErrorCode": "string",
      "DetailedErrorMessage": "string"
    }
  ]
```

如果退出时出现错误，则此调用返回的 ErrorDetails JSON 负载包含有关批量部署执行失败的更多信息。例如，400 系列中的错误状态代码指示输入参数或调用方依赖项中的输入错误。

检查 AWS IoT Greengrass 核心日志

您可以通过查看 AWS IoT Greengrass 核心日志来排查问题：使用以下命令查看 runtime.log：

```
cd /greengrass/ggc/var/log
sudo cat system/runtime.log | more
```

有关 AWS IoT Greengrass 日志记录的更多信息，请参阅 利用 AWS IoT Greengrass 日志进行监控。

另请参阅

有关更多信息，请参阅以下资源：

- 部署 AWS IoT Greengrass 组
- AWS CLI 命令参考 中的 Amazon S3 API 命令。
- AWS CLI 命令参考中的AWS IoT Greengrass 命令

在 AWS IoT Greengrass 核心上运行 Lambda 函数

AWS IoT Greengrass 为您在 AWS Lambda 中编写的用户定义代码提供了容器化 Lambda 运行时环境。部署到 AWS IoT Greengrass 核心的 Lambda 函数在核心的本地 Lambda 运行时环境中运行。本地事件、云中的消息和其他来源可能会触发本地 Lambda 函数，这会为客户端设备提供本地计算功能。例如，您可以使用 Greengrass Lambda 函数筛选设备数据，然后再将数据传输到云。

要将 Lambda 函数部署到核心，您需要将该函数添加到一个 Greengrass 组（通过引用现有的 Lambda 函数），为该函数配置组特定的设置，然后部署该组。如果该函数访问 AWS 服务，您还必须为 Greengrass 组角色添加任何所需的权限。

您可以配置确定 Lambda 函数运行方式的参数，包括权限、隔离、内存限制等。有关更多信息，请参阅 the section called “控制 Greengrass Lambda 函数执行”。

> **Note**
>
> 利用这些设置，还可以在 Docker 容器中运行 AWS IoT Greengrass。有关更多信息，请参阅 the section called “在 Docker 容器中运行 AWS IoT Greengrass”。

下表列出了受支持的 AWS Lambda 运行时以及它们可以在其上运行的 AWS IoT Greengrass 核心软件版本。

语言或平台	GGC 版本
Python 3.8	1.11
Python 3.7	1.9 或更高版本
Python 2.7 *	1.0 或更高版本
Java 8	1.1 或更高版本
Node.js 12.x *	1.10 或更高版本
Node.js 8.10 *	1.9 或更高版本
Node.js 6.10 *	1.1 或更高版本

语言或平台	GGC 版本
C、C++	1.6 或更高版本

* 您可以在支持的 AWS IoT Greengrass 版本上运行使用这些运行时的 Lambda 函数，但无法在 AWS Lambda 中创建它们。如果设备上的运行时与为该函数指定的 AWS Lambda 运行时不同，则可以使用 `FunctionDefintionVersion` 中 `FunctionRuntimeOverride` 来选择自己的运行时。有关更多信息，请参阅CreateFunctionDefinition。有关支持的运行时的更多信息，请参阅AWS Lambda 开发人员指南中的运行时支持策略。

Greengrass Lambda 函数的软件开发工具包

AWS 提供三个开发工具包可供 AWS IoT Greengrass 核心上运行的 Greengrass Lambda 函数使用。这些开发工具包在不同的软件包中，因此，函数可以同时使用它们。要在 Greengrass Lambda 函数中使用开发工具包，请将其包含在您上传到 AWS Lambda 的 Lambda 函数部署包中。

AWS IoT Greengrass 核心开发工具包

允许本地 Lambda 函数与核心交互以执行：

- 与 AWS IoT Core 交换 MQTT 消息。
- 与 Greengrass 组中的连接器、客户端设备及其他 Lambda 函数交换 MQTT 消息。
- 与本地影子服务交互。
- 调用其他的本地 Lambda 函数。
- 访问密钥资源。
- 与流管理器交互。

AWS IoT Greengrass在上提供以下语言和平台的 AWS IoT Greengrass Core SDK GitHub。

- 适用于 Java 的 AWS IoT Greengrass Core 软件开发工具包
- 适用于 Node.js 的 AWS IoT Greengrass Core 软件开发工具包
- 适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包
- 适用于 C 的 AWS IoT Greengrass Core 软件开发工具包

要在 Lambda 函数部署程序包中包含 AWS IoT Greengrass 核心开发工具包依赖项，请执行以下操作：

1. 下载与 Lambda 函数的运行时匹配的 AWS IoT Greengrass 核心开发工具包程序包的语言或平台。
2. 解压缩下载的程序包以获取软件开发工具包。软件开发工具包是 `greengrasssdk` 文件夹。
3. 将 `greengrasssdk` 包含在包含函数代码的 Lambda 函数部署程序包中。这是您在创建 Lambda 函数时上传到 AWS Lambda 的程序包。

StreamManagerClient

只有以下 AWS IoT Greengrass 核心开发工具包可用于流管理器操作：

- Java SDK（v1.4.0 或更高版本）
- Python SDK（v1.5.0 或更高版本）
- Node.js SDK（v1.6.0 或更高版本）

要使用 AWS IoT Greengrass Core SDK for Python 与流管理器进行交互，必须安装 Python 3.7 或更高版本。您还必须安装依赖项以包含在 Python Lambda 函数部署程序包中：

1. 导航到包含该 `requirements.txt` 文件的开发工具包目录。此文件列出了依赖项。
2. 安装开发工具包依赖项。例如，运行以下 `pip` 命令将它们安装在当前目录中：

```
pip install --target . -r requirements.txt
```

在核心设备上安装 AWS IoT Greengrass Core SDK for Python

如果您正在运行 Python Lambda 函数，则可以使用 `pip` 在核心设备上安装适用于 Python 的 AWS IoT Greengrass 核心开发工具包。然后，您可以在 Lambda 函数部署包中部署您的函数而不包括开发工具包。有关更多信息，请参阅 greengrasssdk。

此支持适用于具有大小限制的核心。我们建议您尽可能将开发工具包包含在 Lambda 函数部署程序包中。

AWS IoT Greengrass 机器学习开发工具包

允许本地 Lambda 函数使用作为 ML 资源部署到 Greengrass 核心的机器学习 (ML) 模型。对于作为连接器部署到核心的本地推理服务，Lambda 函数可以使用 SDK 来调用并与之交互。Lambda 函

数和 ML 连接器也可以使用软件开发工具包将数据发送到 ML 反馈连接器进行上传和发布。有关更多信息，包括使用此开发工具包的代码示例，请参阅the section called “ML 图像分类”、the section called “ML 对象检测”和the section called “机器学习反馈”。

下表列出了对于各开发工具包版本支持的语言或平台以及它们可以在其中运行的 AWS IoT Greengrass 核心软件的版本。

SDK 版本	语言或平台	必需的 GGC 版本	更改日志
1.1.0	Python 3.7 或 2.7	1.9.3 或更高版本	添加了 Python 3.7 支持和新的 `feedback` 客户端。
1.0.0	Python 2.7	1.7 或更高版本	首次发布。

有关下载信息，请参阅the section called “AWS IoT Greengrass ML 软件开发工具包软件”。

AWS SDK

允许本地 Lambda 函数对 AWS 服务进行直接调用，如 Amazon S3、DynamoDB、AWS IoT 和 AWS IoT Greengrass。要在 Greengrass Lambda 函数中使用 AWS 开发工具包，必须将其包含在部署软件包中。在使用同一程序包中的 AWS 开发工具包和 AWS IoT Greengrass 核心开发工具包时，请确保 Lambda 函数使用正确的命名空间。在核心处于脱机状态时，Greengrass Lambda 函数无法与云服务进行通信。

从入门资源中心下载 AWS 开发工具包。

有关创建部署程序包的更多信息，请参阅入门教程中的 the section called “创建并打包 Lambda 函数” 或 AWS Lambda 开发人员指南中的创建部署程序包。

迁移基于云的 Lambda 函数

AWS IoT Greengrass 核心开发工具包采用 AWS 开发工具包编程模型，从而轻松将为云开发的 Lambda 函数移植到在 AWS IoT Greengrass 核心上运行的 Lambda 函数。

例如，以下 Python Lambda 函数使用 AWS SDK for Python (Boto3) 将消息发布到云中的主题 `some/topic`：

```
import boto3

iot_client = boto3.client("iot-data")
response = iot_client.publish(
    topic="some/topic", qos=0, payload="Some payload".encode()
)
```

要为 AWS IoT Greengrass 核心移植该函数，请在 `import` 语句和 `client` 初始化中将 `boto3` 模块名称更改为 `greengrasssdk`，如以下示例所示：

```
import greengrasssdk

iot_client = greengrasssdk.client("iot-data")
iot_client.publish(topic="some/topic", qos=0, payload="Some payload".encode())
```

> **Note**
>
> AWS IoT Greengrass 核心开发工具包仅支持发送 QoS 为 0 的 MQTT 消息。有关更多信息，请参阅 the section called “消息服务质量”。

编程模型之间的相似性还使您能够在云中开发 Lambda 函数，然后只需要极少的工作量便可将其迁移到 AWS IoT Greengrass。Lambda 可执行文件无法在云中运行，因此，您无法在部署之前使用 AWS 开发工具包在云中对其进行测试。

按别名或版本引用 Lambda 函数

Greengrass 组可以按别名（推荐）或版本引用 Lambda 函数。使用别名，您可以更轻松地管理代码更新，因为您在更新函数代码时，不必更改订阅表或组定义。相反，您只需将别名指向新的函数版本。在组部署期间，别名将解析为版本号。在使用别名时，解析的版本将在部署时更新为别名指向的版本。

AWS IoT Greengrass 不支持 $LATEST 版本的 Lambda 别名。$LATEST 版本未绑定到发布的不可变函数版本，可以随时对其进行更改，这与 AWS IoT Greengrass 版本不可变原则相反。

使用代码更改更新 Greengrass Lambda 函数的常见做法是，在 Greengrass 组和订阅中使用名为 **`PRODUCTION`** 的别名。在将 Lambda 函数的新版本提升到生产环境时，请将别名指向最新的稳定版本，然后重新部署组。您也可以使用这种方法回滚到以前的版本。

使用组特定的配置控制 Greengrass Lambda 函数的执行

AWS IoT Greengrass 提供了基于云的 Greengrass Lambda 函数管理。尽管使用 AWS Lambda 来管理 Lambda 函数的代码和依赖项，但您可以配置 Lambda 函数在 Greengrass 组中运行时的行为方式。

组特定的配置设置

AWS IoT Greengrass 为 Greengrass Lambda 函数提供以下组特定的配置设置。

系统用户和组

用于运行 Lambda 函数的访问身份。默认情况下，Lambda 函数以该组的默认访问身份运行。通常情况下，这是标准 AWS IoT Greengrass 系统账户（ggc_user 和 ggc_group）。您可以更改设置，并选择具有运行 Lambda 函数所需权限的用户 ID 和组 ID。您可以覆盖 UID 和 GID 或只覆盖一个（如果将另一个字段留空）。通过此设置可以更精细地控制设备资源访问。建议使用适当的资源限制、文件权限和磁盘配额为使用其权限运行 Lambda 函数的用户和组配置 Greengrass 硬件。

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

> **Important**
>
> 除非绝对有必要，否则建议避免以根用户身份运行 Lambda 函数。以 root 身份运行会增加以下风险：
>
> - 意外更改的风险，如意外删除关键文件。
> - 恶意个人对您的数据和设备造成的风险。
> - 当 Docker 容器使用 `--net=host` 和 `UID=EUID=0` 运行时，就可以规避容器的风险
>
> 如果需要以根用户身份运行，必须更新 AWS IoT Greengrass 配置以启用它。有关更多信息，请参阅 the section called “以根用户身份运行 Lambda 函数”。

系统用户 ID（数字）

具有运行 Lambda 函数所需权限的用户的用户 ID。仅当选择 以另一个用户 ID/组 ID 身份运行时，此设置才可用。通过对 AWS IoT Greengrass 核心设备使用 getent passwd 命令，可以查找要用于运行 Lambda 函数的用户 ID。

如果您使用相同的 UID 在 Greengrass 核心设备上运行进程和 Lambda 函数，则您的 Greengrass 组角色可以授予进程临时凭据。这些进程可以在整个 Greengrass 核心部署中使用临时证书。

系统组 ID（数字）

具有运行 Lambda 函数所需权限的组的组 ID。仅当选择 以另一个用户 ID/组 ID 身份运行时，此设置才可用。通过对 AWS IoT Greengrass 核心设备使用 getent group 命令，可以查找要用于运行 Lambda 函数的组 ID。

Lambda 函数容器化

选择 Lambda 函数是否对组使用默认容器化运行，或者指定应始终对此 Lambda 函数使用的容器化。

Lambda 函数的容器化模式决定其隔离级别。

- 容器化 Lambda 函数在 Greengrass 容器模式下运行。Lambda 函数在 AWS IoT Greengrass 容器内的隔离运行时环境（或命名空间）中运行。
- 非容器化 Lambda 函数在 无容器 模式下运行。这些 Lambda 函数作为常规 Linux 进程运行，没有任何隔离。

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

建议在 Greengrass 容器中运行 Lambda 函数，除非您的使用案例要求它们在不进行容器化的情况下运行。当 Lambda 函数在 Greengrass 容器中运行时，可以使用附加的本地和设备资源并获得隔离和安全性提高的优势。在更改容器化之前，请参阅the section called “选择 Lambda 函数容器化时的注意事项”。

> **Note**
>
> 要在不启用设备内核命名空间和组的情况下运行，所有 Lambda 函数都必须在不进行容器化的情况下运行。通过为组设置默认容器化可以轻松完成此操作。有关信息，请参阅 the section called “在组中设置 Lambda 函数的默认容器化”。

内存限制

为函数分配的内存。默认值为 16 MB。

> **Note**
>
> 当您将 Lambda 函数更改为在无容器化的情况下运行时，内存限制设置将不可用。在没有容器化的情况下运行的 Lambda 函数没有内存限制。在将 Lambda 函数或组默认容器化设置更改为在不执行容器化的情况下运行时，将丢弃内存限制设置。

超时

在终止函数或请求之前经过的时间。默认值为 3 秒。

Pinned

Lambda 函数生命周期可以是按需或长时间生存。默认为“按需”。

在调用时，将在新的或重复使用的容器中启动按需 Lambda 函数。函数请求可以由任何可用的容器进行处理。长时间生存的或固定的 Lambda 函数在 AWS IoT Greengrass 启动后自动启动，并在自己的容器 (或沙盒) 中保持运行。函数的所有请求是由相同的容器处理的。有关更多信息，请参阅 the section called “生命周期配置”。

对 /sys 目录的只读访问权限

函数是否可以访问主机的 /sys 文件夹。当函数必须从 /sys 中读取设备信息时，请使用此设置。默认值为 false。

> **Note**
>
> 在不进行容器化的情况下运行 Lambda 函数时，此设置不可用。将 Lambda 函数更改为在不进行容器化的情况下运行时，将丢弃此设置的值。

编码类型

函数输入负载的预期编码类型：JSON 或二进制。默认值为 JSON。

从 AWS IoT Greengrass 核心软件 v1.5.0 和 AWS IoT Greengrass 核心开发工具包 v1.1.0 开始，将提供二进制编码类型支持。对于与设备数据交互的函数，接受二进制输入数据可能是非常有用的，因为设备的受限硬件功能通常导致很难或无法构造 JSON 数据类型。

> **Note**
>
> Lambda 可执行文件仅支持二进制编码类型，不支持 JSON。

进程参数

在 Lambda 函数运行时要传递给该函数的命令行参数。

环境变量

可动态将设置传递到函数代码和库的键值对。本地环境变量的工作方式与 AWS Lambda 函数环境变量相同，但可以在核心环境中使用。

资源访问策略

允许 Lambda 函数访问的最多 10 个本地资源、密钥资源和机器学习资源的列表，以及相应的 `read-only` 或 `read-write` 权限。在控制台中，这些关联资源列在资源选项卡的组配置页面上。

容器化模式会影响 Lambda 函数访问本地设备和卷资源以及机器学习资源的方式。

- 非容器化 Lambda 函数必须直接通过核心设备上的文件系统访问本地设备和卷资源。
- 要允许非容器化 Lambda 函数访问 Greengrass 组中的机器学习资源，必须在机器学习资源上设置资源所有者和访问权限属性。有关更多信息，请参阅 the section called “访问机器学习资源”。

有关使用 AWS IoT Greengrass API 为用户定义的 Lambda 函数设置特定于组的配置设置的信息，请参阅 CreateFunctionDefinitionAWS IoT Greengrass Version 1API 参考或命令参考create-function-definition中的。AWS CLI要将 Lambda 函数部署到 Greengrass 核心，请创建包含函数的函数定义版本，创建引用函数定义版本和其他组组件的组版本，然后部署组。

以根用户身份运行 Lambda 函数

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

必须先更新 AWS IoT Greengrass 配置以启用支持，然后才能以根用户身份运行一个或多个 Lambda 函数。默认情况下，不支持以根用户身份运行 Lambda 函数。如果尝试部署 Lambda 函数并以根用户身份运行它（UID 和 GID 为 0），并且没有更新 AWS IoT Greengrass 配置，部署将失败。运行时日志 (*greengrass_root*/ggc/var/log/system/runtime.log) 中会显示类似下面的错误：

```
lambda(s)
```

```
[list of function arns] are configured to run as root while Greengrass is not
 configured to run lambdas with root permissions
```

Important

除非绝对有必要，否则建议避免以根用户身份运行 Lambda 函数。以 root 身份运行会增加以下风险：

- 意外更改的风险，如意外删除关键文件。
- 恶意个人对您的数据和设备造成的风险。
- 当 Docker 容器使用 `--net=host` 和 `UID=EUID=0` 运行时，就可以规避容器的风险

允许 Lambda 函数以根用户身份运行

1. 在 AWS IoT Greengrass 设备上，导航到 *greengrass-root*/config 文件夹。

   Note

   默认情况下，*greengrass-root* 是 /greengrass 目录。

2. 编辑 config.json 文件以向 `runtime` 字段添加 `"allowFunctionsToRunAsRoot" : "yes"`。例如：

   ```
   {
     "coreThing" : {
       ...
     },
     "runtime" : {
       ...
       "allowFunctionsToRunAsRoot" : "yes"
     },
     ...
   }
   ```

3. 使用以下命令重启 AWS IoT Greengrass：

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

现在，可以将 Lambda 函数的用户 ID 和组 ID (UID/GID) 设置为 0，从而以根用户身份运行 Lambda 函数。

如果要禁止以根用户身份运行 Lambda 函数，可以将 `"allowFunctionsToRunAsRoot"` 的值更改为 `"no"` 并重启 AWS IoT Greengrass。

选择 Lambda 函数容器化时的注意事项

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

默认情况下，Lambda 函数在 AWS IoT Greengrass 容器内运行。容器在函数和主机之间提供隔离，从而为主机和容器内的函数提供更高的安全性。

建议在 Greengrass 容器中运行 Lambda 函数，除非您的使用案例要求它们在不进行容器化的情况下运行。通过在 Greengrass 容器内运行 Lambda 函数，可以更好地控制资源访问限制。

下面是在不进行容器化的情况下运行的一些示例使用案例：

- 您想要在不支持容器模式的设备上运行 AWS IoT Greengrass（例如，因为您使用的是特殊的 Linux 发行版或内核版本太旧）。
- 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。
- 您需要访问的本地资源的路径在部署时无法确定，或者其路径在部署后可能更改，如可插拔设备。
- 您有一个编写为进程的旧应用程序，在作为容器化的 Lambda 函数运行它时遇到了问题。

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。 • 即使以相同的组 ID 运行，在 Greengrass 容器中运行的 Lambda 函数也无权访问其他 Lambda 函数的部署代码。也就是说，Lambda 函数之间以更强的隔离运行。 • 由于在 AWS IoT Greengrass 容器中运行的 Lambda 函数的所有子进程都在与 Lambda 函

容器化	注意事项
	数相同的容器中运行，当 Lambda 函数终止时，子进程也会终止。

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。您必须直接访问核心设备上的这些资源，而不是将其作为 Greengrass 组的一部分进行访问。
	• 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。这需要 AWS IoT Greengrass Core 软件 v1.10 或更高版本。有关更多信息，请参阅 the section called “访问机器学习资源”。
	• Lambda 函数对使用相同组 ID 运行的其他 Lambda 函数的部署代码具有只读访问权限。
	• 在父 Lambda 函数终止时，在不同进程会话中生成子进程或使用重写的 SIGHUP（信号挂断）处理程序（例如使用 nohup 实用程序）的 Lambda 函数不会自动被 AWS IoT Greengrass 终止。

> **Note**
>
> Greengrass 组的默认容器化设置不适用于连接器。

更改 Lambda 函数的容器化会在部署时导致问题。如果为 Lambda 函数分配的本地资源在新容器化设置下不再可用，部署将失败。

- 将 Lambda 函数从在 Greengrass 容器中运行更改为在不进行容器化的情况下运行时，将丢弃该函数的内存限制。您必须直接访问文件系统，而不是使用附加的本地资源。必须在部署前删除所有附加的资源。

- 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。您必须为每个函数定义内存限制或接受默认设置 (16MB)。您必须在部署前为每个 Lambda 函数配置这些设置。

更改 Lambda 函数的容器化设置

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择包含要更改设置的 Lambda 函数的组。
3. 选择 Lambda 函数选项卡。
4. 在要更改的 Lambda 函数上，选择省略号 (...)，然后选择编辑配置。
5. 更改容器化设置。如果要将 Lambda 函数配置为在 Greengrass 容器中运行，还必须设置内存限制和对 /sys 目录的只读访问权限。
6. 选择保存，然后选择确认，保存对您的 Lambda 函数所做的更改。

这些更改将在部署组时生效。

您也可以在 AWS IoT GreengrassAPI 参考CreateFunctionDefinitionVersion中使用CreateFunctionDefinition和。更改容器化设置时还请务必更新其他参数。例如，从在 Greengrass 容器中运行 Lambda 函数更改为在不进行容器化的情况下运行时，请务必清除 `MemorySize` 参数。

确定 Greengrass 设备支持的隔离模式

可以使用 AWS IoT Greengrass 依赖项检查程序确定 Greengrass 设备支持哪些隔离模式（Greengrass 容器/无容器）。

运行 AWS IoT Greengrass 依赖项检查程序

1. 从GitHub存储库下载并运行AWS IoT Greengrass依赖关系检查器。

```
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

2. 在 `more` 出现处，按 Spacebar 键以显示另一页文本。

要获得有关 modprobe 命令的信息，请在终端中运行 man modprobe。

为组中的 Lambda 函数设置默认访问身份

此功能适用于 AWS IoT Greengrass Core v1.8 及更高版本。

要更多地控制对设备资源的访问权限，您可以配置用于运行组中的 Lambda 函数的默认访问身份。此设置确定了在核心设备上运行 Lambda 函数时提供给这些函数的默认权限。要覆盖组中各个函数的设置，您可以使用该函数的 Run as (运行身份) 属性。有关更多信息，请参阅运行身份。

此组级别设置也用于运行底层 AWS IoT Greengrass 核心软件。这由负责管理操作（如消息路由、本地影子同步和自动 IP 检测）的系统 Lambda 函数组成。

默认访问身份可配置为以标准 AWS IoT Greengrass 系统账户身份（ggc_user 和 ggc_group）运行，或者使用另一个用户或组的权限。建议使用适当的资源限制、文件权限和磁盘配额，为使用其权限运行用户定义的或系统 Lambda 函数的任何用户和组配置 Greengrass 硬件。

修改 AWS IoT Greengrass 组的默认访问身份

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择要更改其设置的组。
3. 选择 Lambda 函数选项卡，然后在默认 Lambda 函数运行时环境部分下选择编辑。
4. 在 编辑默认 Lambda 函数运行时环境)页面的默认系统用户和组下，选择其他用户 ID/组 ID。

   当您选择此选项时，系统用户 ID (数字) 和 系统组 ID (数字) 字段会显示。
5. 输入用户 ID 和/或组 ID。如果将字段留空，将使用相应的 Greengrass 系统账户（ggc_user 或 ggc_group）。

   - 对于 系统用户 ID (数字)，输入某个用户的用户 ID，该用户具有您在默认情况下要用来运行组中 Lambda 函数的权限。您可以在您的 AWS IoT Greengrass 设备上使用 getent passwd 命令查找用户 ID。
   - 对于 系统组 ID (数字)，输入某个组的组 ID，该组具有您在默认情况下要用来运行组中 Lambda 函数的权限。您可以在您的 AWS IoT Greengrass 设备上使用 getent group 命令查找组 ID。

> **Important**
>
> 以根用户身份运行会给您的数据和设备增加风险。请勿以根用户身份（UID/GID=0）运行，除非您的业务案例需要这样。有关更多信息，请参阅 the section called “以根用户身份运行 Lambda 函数”。

这些更改将在部署组时生效。

在组中设置 Lambda 函数的默认容器化

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

Greengrass 组的容器化设置决定了 Lambda 函数在组中的默认容器化。

- 在 Greengrass 容器模式下，默认情况下，Lambda 函数在 AWS IoT Greengrass 容器内的独立运行时环境中运行。
- 在无容器模式下，默认情况下，Lambda 函数将作为常规 Linux 进程运行。

您可以修改组设置以指定 Lambda 函数在组中的默认容器化。如果需要 Lambda 函数以不同于组默认设置的容器化方式运行，可以覆盖组中一个或多个 Lambda 函数的此设置。在更改容器化设置之前，请参阅the section called “选择 Lambda 函数容器化时的注意事项”。

> **Important**
>
> 如果要更改组的默认容器化设置，但有一个或多个函数使用不同的容器化，请先更改这些 Lambda 函数的设置，然后再更改组设置。如果先更改组容器化设置，将丢弃内存限制和对 /sys 目录的只读访问权限设置的值。

修改 AWS IoT Greengrass 组的容器化设置

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择要更改其设置的组。
3. 选择 Lambda 函数选项卡。
4. 在默认 Lambda 函数运行时环境下，选择编辑。

5. 在编辑默认 Lambda 函数运行时环境页面的默认 Lambda 函数容器化下面，更改容器化设置。
6. 选择保存。

这些更改将在部署组时生效。

Greengrass Lambda 函数的通信流

Greengrass Lambda 函数支持多种与 AWS IoT Greengrass 组的其他成员、本地服务和云服务（包括 AWS 服务）通信的方法。

使用 MQTT 消息进行通信

Lambda 函数可以使用由订阅控制的发布-订阅模式发送和接收 MQTT 消息。

此通信流能让 Lambda 函数与以下实体交换消息：

- 组中的客户端设备。
- 组中的连接器。
- 组中的其他 Lambda 函数。
- AWS IoT.
- 本地设备影子服务。

订阅会定义消息源、消息目标以及用于将消息从源路由到目标的主题。发布到 Lambda 函数的消息将传递到为该函数注册的处理程序。订阅实现了更高的安全性并提供可预测的交互。有关更多信息，请参阅 the section called “MQTT 消息传递工作流中的托管订阅”。

> **Note**
>
> 在核心处于脱机状态时，Greengrass Lambda 函数可以与客户端设备、连接器、其他函数和本地影子交换消息，但将到 AWS IoT 的消息排入队列。有关更多信息，请参阅 the section called “MQTT 消息队列”。

其他通信流

- 为了与核心设备上的本地设备和卷资源以及机器学习模型进行交互，Greengrass Lambda 函数使用平台特定的操作系统接口。例如，您可以在 Python 函数中使用 openos 模块中的 方法。要允许函

数访问某个资源，函数必须与该资源关联，或者为其授予了 `read-only` 或 `read-write` 权限。有关更多信息（包括 AWS IoT Greengrass 核心版本可用性），请参阅 访问本地资源 和 the section called “从 Lambda 函数代码访问机器学习资源”。

> **Note**
>
> 如果您在不进行容器化的情况下运行 Lambda 函数，则无法使用附加的本地设备和卷资源，并且必须直接访问这些资源。

- Lambda 函数可以使用 AWS IoT Greengrass 核心开发工具包中的 `Lambda` 客户端来调用 Greengrass 组中的其他 Lambda 函数。
- Lambda 函数可以使用 AWS 开发工具包与 AWS 服务进行通信。有关更多信息，请参阅 AWS 软件开发工具包。
- Lambda 函数可以使用第三方接口与外部云服务进行通信，类似于基于云的 Lambda 函数。

> **Note**
>
> 在核心处于脱机状态时，Lambda 函数无法与 AWS 或其他云服务进行通信。

检索输入 MQTT 主题（或主旨）

AWS IoT Greengrass 使用订阅来控制在客户端设备、Lambda 函数以及组中的连接器之间交换 MQTT 消息以及与 AWS IoT 或本地影子服务交换 MQTT 消息。订阅定义消息源、消息目标以及用于路由消息的 MQTT 主题。当目标是一个 Lambda 函数时，当源发布消息时将调用该函数的处理程序。有关更多信息，请参阅 the section called “使用 MQTT 消息进行通信”。

以下示例显示 Lambda 函数如何可以从传递给处理程序的 `context` 获取输入主题。具体方法是通过从上下文层次结构 (`context.client_context.custom['subject']`) 访问 `subject` 密钥。该示例还会解析输入 JSON 消息，然后发布解析的主题和消息。

> **Note**
>
> 在 AWS IoT Greengrass API 中，订阅的主题由 `subject` 属性表示。

```
import greengrasssdk
```

```
import logging

client = greengrasssdk.client('iot-data')

OUTPUT_TOPIC = 'test/topic_results'

def get_input_topic(context):
    try:
        topic = context.client_context.custom['subject']
    except Exception as e:
        logging.error('Topic could not be parsed. ' + repr(e))
    return topic

def get_input_message(event):
    try:
        message = event['test-key']
    except Exception as e:
        logging.error('Message could not be parsed. ' + repr(e))
    return message

def function_handler(event, context):
    try:
        input_topic = get_input_topic(context)
        input_message = get_input_message(event)
        response = 'Invoked on topic "%s" with message "%s"' % (input_topic,
 input_message)
        logging.info(response)
    except Exception as e:
        logging.error(e)

    client.publish(topic=OUTPUT_TOPIC, payload=response)

    return
```

要测试函数，请使用默认配置设置将其添加到组中。然后，添加以下订阅并部署该组。有关说明，请参阅 the section called “模块 3（第 1 部分）：AWS IoT Greengrass 的 Lambda 函数”。

主题筛选条件	此函数将接收的输入数据
test/input_message	输入消息
test/topic_results	输出消息

部署完成后，调用该函数。

1. 在 AWS IoT 控制台中打开 MQTT 测试客户端页面。
2. 要订阅 test/topic_results 主题，可选择订阅主题选项卡。
3. 要向 test/input_message 主题发布消息，可选择发布到主题选项卡。对于此示例，您必须在 JSON 消息中包含 test-key 属性。

   ```
   {
     "test-key": "Some string value"
   }
   ```

   如果成功，此函数会将输入主题和消息字符串发布到 test/topic_results 主题。

Greengrass Lambda 函数的生命周期配置

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。生命周期还确定如何保留位于函数处理程序外部的变量和预处理逻辑。

AWS IoT Greengrass 支持按需（默认）或长时间生存的生命周期：

- 按需函数在调用时启动，并在没有要执行的任务时停止。除非具有可重复使用的现有容器，否则，函数调用将创建单独的容器（或沙盒）以处理调用。发送到函数的数据可以由任何容器提取。

  按需 函数的多次调用可以并行运行。

  在创建新的容器时，不会保留在函数处理程序外部定义的变量和预处理逻辑。

- 长时间生存的（或固定的）函数在 AWS IoT Greengrass 核心启动时自动启动，并在单个容器中运行。发送到函数的所有数据由相同的容器提取。

  多个调用将排入队列，直到执行了以前的调用。

  将为处理程序的每次调用保留在函数处理程序外部定义的变量和预处理逻辑。

  如果需要在没有任何初始输入的情况下开始工作，长时间生存的 Lambda 函数是非常有用的。例如，在长时间存在的函数开始接收设备数据时，该函数可以加载并开始处理 ML 模型以使其准备就绪。

  > **Note**
  >
  > 请记住，长时间生存的函数具有与其处理程序调用关联的超时。如果要执行无限期运行的代码，您必须在处理程序外部启动该代码。确保在处理程序外部没有阻止代码而导致函数可能无法完成初始化。
  > 这些函数将运行，除非核心停止（例如，在组部署或设备重启期间）或函数进入错误状态（例如，处理程序超时、未捕获的异常或者当它超过其内存限制时）。

有关容器重复使用的更多信息，请参阅 AWS 计算博客上的了解 AWS Lambda 中的容器重复使用。

Lambda 可执行文件

此功能适用于 AWS IoT Greengrass Core v1.6 及更高版本。

Lambda 可执行文件是一种 Greengrass Lambda 函数，可用于在核心环境中运行二进制代码。这样，您就可以在本地执行设备特定的功能，并从较小的编译代码占用空间中受益。Lambda 可执行文件可以由事件调用，它调用其他函数以及访问本地资源。

Lambda 可执行文件仅支持二进制编码类型（不支持 JSON），但您可以像其他 Greengrass Lambda 函数一样在 Greengrass 组中管理这些函数并进行部署。不过，创建 Lambda 可执行文件的过程与创建 Python、Java 和 Node.js Lambda 函数不同：

- 您无法使用 AWS Lambda 控制台创建（或管理）Lambda 可执行文件。您只能使用 AWS Lambda API 创建 Lambda 可执行文件。
- 您将函数代码作为包含 适用于 C 的 AWS IoT Greengrass Core 软件开发工具包的编译可执行文件上传到 AWS Lambda。
- 您将可执行文件名称指定为函数处理程序。

Lambda 可执行文件必须在其函数代码中实施某些调用和编程模式。例如，`main` 方法必须：

- 调用 `gg_global_init` 以初始化 Greengrass 内部全局变量。必须先调用该函数，然后再创建任何线程以及调用任何其他 AWS IoT Greengrass 核心开发工具包函数。
- 调用 `gg_runtime_start` 以在 Greengrass Lambda 运行时环境中注册函数处理程序。必须在初始化时调用该函数。调用该函数将导致运行时环境使用当前线程。可选的 `GG_RT_OPT_ASYNC` 参数指示该函数不要阻止，而是为运行时环境创建新的线程。该函数使用 `SIGTERM` 处理程序。

以下片段是 simple_handler.c 代码示例中的`main`方法。 GitHub

```
int main() {
    gg_error err = GGE_SUCCESS;

    err = gg_global_init(0);
    if(err) {
        gg_log(GG_LOG_ERROR, "gg_global_init failed %d", err);
        goto cleanup;
    }

    gg_runtime_start(handler, 0);

cleanup:
    return -1;
}
```

有关要求、限制和其他实施细节的更多信息，请参阅适用于 C 的 AWS IoT Greengrass Core 软件开发工具包。

创建 Lambda 可执行文件

在编译代码以及开发工具包后，请使用 AWS Lambda API 创建 Lambda 函数并上传编译的可执行文件。

> **Note**
>
> 必须使用与 C89 兼容的编译器编译您的函数。

以下示例使用 create-function CLI 命令创建 Lambda 可执行文件。该命令指定：

- 处理程序的可执行文件名称。它必须是编译的可执行文件的确切名称。
- 包含编译的可执行文件的 `.zip` 文件的路径。
- 运行时环境的 `arn:aws:greengrass:::runtime/function/executable`。这是所有 Lambda 可执行文件的运行时环境。

> **Note**
>
> 对于 `role`，您可以指定任何 Lambda 执行角色的 ARN。AWS IoT Greengrass 不使用该角色，但需要使用该参数以创建函数。有关 Lambda 执行角色的更多信息，请参阅 AWS Lambda 开发人员指南中的 AWS Lambda 许可模型。

```
aws lambda create-function \
--region aws-region \
--function-name function-name \
--handler executable-name \
--role role-arn \
--zip-file fileb://file-name.zip \
--runtime arn:aws:greengrass:::runtime/function/executable
```

接下来，使用 AWS Lambda API 发布一个版本并创建别名。

- 使用 publish-version 发布一个函数版本。

```
aws lambda publish-version \
--function-name function-name \
```

```
--region aws-region
```

- 使用 create-alias 创建指向刚发布的版本的别名。在将 Lambda 函数添加到 Greengrass 组时，我们建议您按别名引用这些函数。

```
aws lambda create-alias \
--function-name function-name \
--name alias-name \
--function-version version-number \
--region aws-region
```

> **Note**
>
> AWS Lambda 控制台不显示 Lambda 可执行文件。要更新函数代码，您还必须使用 AWS Lambda API。

接下来，将 Lambda 可执行文件添加到一个 Greengrass 组，在组特定的设置中将其配置为接受二进制输入数据，然后部署该组。您可以在 AWS IoT Greengrass 控制台中或者使用 AWS IoT Greengrass API 执行此操作。

在 Docker 容器中运行 AWS IoT Greengrass

AWS IoT Greengrass 可以配置为在 Docker 容器中运行。

您可以通过 Amazon CloudFront（安装了 AWS IoT Greengrass 核心软件和依赖项）下载 Dockerfile。要修改 Docker 镜像以便在不同平台架构上运行或减少 Docker 镜像的大小，请参阅 Docker 程序包下载中的 `README` 文件。

为了帮助您开始体验 AWS IoT Greengrass，AWS 还提供了预构建的 Docker 映像，其中安装了 AWS IoT Greengrass Core 软件和依赖项。您可以从 Docker Hub 或 Amazon Elastic Container Registry (Amazon ECR) 下载映像。这些预构建的映像使用 Amazon Linux 2 (x86_64) 和 Alpine Linux（x86_64、Armv7l 或 AArch64）基本映像。

> **Important**
>
> 2022 年 6 月 30 日，AWS IoT Greengrass 结束了对发布到 Amazon Elastic Container Registry (Amazon ECR) 和 Docker Hub 的 AWS IoT Greengrass Core 软件 v1.x Docker 映像的维护。您可以继续从 Amazon ECR 和 Docker Hub 下载这些 Docker 映像，直至 2023

年 6 月 30（即维护结束 1 年后）为止。但在 2022 年 6 月 30 日维护结束后，AWS IoT Greengrass Core 软件 v1.x Docker 映像不再收到安全补丁或错误修复。如果运行的生产工作负载依赖于这些 Docker 映像，我们建议您使用提供的 Dockerfiles 构建自己的 Docker 映像。AWS IoT Greengrass有关更多信息，请参阅AWS IoT Greengrass Docker 软件。

本主题介绍如何从 Amazon ECR 下载 AWS IoT Greengrass Docker 映像并在 Windows、macOS 或 Linux (x86_64) 平台上运行它。本主题包含以下步骤：

1. 从 Amazon ECR 获取 AWS IoT Greengrass 容器镜像
2. 创建和配置 Greengrass 组和核心
3. 本地运行 AWS IoT Greengrass
4. 为组配置“No container (无容器)”容器化
5. 将 Lambda 函数部署到 Docker 容器
6. （可选）部署与 Docker 容器中的 Greengrass 交互的客户端设备

在 Docker 容器中运行 AWS IoT Greengrass 时，不支持以下功能：

- 在 Greengrass 容器模式下运行的连接器。要在 Docker 容器中运行连接器，该连接器必须在无容器模式下运行。要查找支持无容器模式的连接器，请参阅the section called “AWS 提供的 Greengrass 连接器”。其中一些连接器具有隔离模式参数，您必须将此参数设为无容器。
- 本地设备和卷资源。在 Docker 容器中运行的用户定义 Lambda 函数必须直接访问核心上的设备和卷。

当 Greengrass 组的 Lambda 运行时环境设置为 无容器时（此时需要在 Docker 容器中运行 AWS IoT Greengrass），这些功能不受支持。

先决条件

在开始本教程之前，必须执行以下操作。

- 您必须根据选择的 AWS Command Line Interface (AWS CLI) 版本，在主机上安装以下软件和版本。

  AWS CLI version 2

  - Docker 版本 18.09 或更高版本。早期版本也可能有效，但我们建议使用 18.09 或更高版本。
  - AWS CLI 2.0.0 版或更高版本。

- 要安装 AWS CLI 版本 2，请参阅安装 AWS CLI 版本 2。
- 要配置 AWS CLI，请参阅配置 AWS CLI。

> **Note**
>
> 要在 Windows 计算机上升级到更高的 AWS CLI 版本 2，您必须重复 MSI 安装过程。

AWS CLI version 1

- Docker 版本 18.09 或更高版本。早期版本也可能有效，但我们建议使用 18.09 或更高版本。
- Python 版本 3.6 或更高版本。
- pip 版本 18.1 或更高版本。
- AWS CLI 1.17.10 版或更高版本
  - 要安装 AWS CLI 版本 1，请参阅安装 AWS CLI 版本 1。
  - 要配置 AWS CLI，请参阅配置 AWS CLI。
  - 要升级到 AWS CLI 版本 1 的最新版本，请运行以下命令。

```
pip install awscli --upgrade --user
```

> **Note**
>
> 如果您在 Windows 上使用 MSI 安装AWS CLI版本 1，请注意以下几点：
>
> - 如果 AWS CLI 版本 1 安装过程中未能安装 botocore，请尝试使用 Python 和 pip 安装。
> - 要升级到更新的 AWS CLI 版本 1，您必须重复 MSI 安装过程。

- 要访问 Amazon Elastic Container Registry (Amazon ECR) 资源，您必须授予以下权限。
  - Amazon ECR 要求用户通过 AWS Identity and Access Management IAM 策略授予 `ecr:GetAuthorizationToken` 权限，然后才能对注册表进行身份验证并对 Amazon ECR 存储库推送或提取镜像。有关更多信息，请参阅 Amazon Elastic Container Registry 用户指南中的 Amazon ECR 存储库策略示例和访问一个 Amazon ECR 存储库。

步骤 1：从 Amazon ECR 获取 AWS IoT Greengrass 容器镜像

AWS 提供安装了 AWS IoT Greengrass Core 软件的 Docker 映像。

> **Warning**
>
> 从 AWS IoT Greengrass Core 软件的 v1.11.6 开始，Greengrass Docker 映像不再包含 Python 2.7，因为 Python 2.7 已于 2020 年停用，不再接收安全更新。如果您选择更新这些 Docker 映像，我们建议您在将更新部署到生产设备之前，先验证应用程序是否可以使用新的 Docker 映像。如果使用 Greengrass Docker 映像的应用程序需要 Python 2.7，您可以修改 Greengrass Dockerfile，为您的应用程序包含 Python 2.7。

有关演示如何从 Amazon ECR 拉取 `latest` 映像的步骤，请选择您的操作系统：

拉取容器镜像 (Linux)

在计算机终端中运行以下命令。

1. 登录 Amazon ECR 中的 AWS IoT Greengrass 注册表。

   ```
   aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
   password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
   ```

   如果成功，输出将打印 `Login Succeeded`。

2. 检索 AWS IoT Greengrass 容器镜像。

   ```
   docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   > **Note**
   >
   > `latest` 映像包含安装在 Amazon Linux 2 基本映像上的 AWS IoT Greengrass Core 软件的最新稳定版本。您还可以从存储库中拉取其他版本。要查找所有可用映像的标签，请查看 Docker Hub 上的 标签页面或使用 aws ecr list-images 命令。例如：
   >
   > ```
   > aws ecr list-images --region us-west-2 --registry-id 216483018798 --
   > repository-name aws-iot-greengrass
   > ```

3. 启用符号链接和硬链接保护。如果您尝试在容器中运行 AWS IoT Greengrass，则只能启用当前引导的设置。

> **Note**
>
> 您可能需要使用 sudo 才能运行这些命令。

- 要仅启用当前引导的设置，请执行以下操作：

```
echo 1 > /proc/sys/fs/protected_hardlinks
echo 1 > /proc/sys/fs/protected_symlinks
```

- 要使设置在重新启动后保持不变，请执行以下操作：

```
echo '# AWS IoT Greengrass' >> /etc/sysctl.conf
echo 'fs.protected_hardlinks = 1' >> /etc/sysctl.conf
echo 'fs.protected_symlinks = 1' >> /etc/sysctl.conf

sysctl -p
```

4. 启用 IPv4 网络转发，这是 AWS IoT Greengrass 云部署和 MQTT 通信在 Linux 上运行所必需的。在 `/etc/sysctl.conf` 文件中，将 `net.ipv4.ip_forward` 设置为 1，然后重新加载 `sysctls`。

```
sudo nano /etc/sysctl.conf
# set this net.ipv4.ip_forward = 1
sudo sysctl -p
```

> **Note**
>
> 您可以使用您选择的编辑器而不是 nano。

拉取容器镜像 (macOS)

在计算机终端中运行以下命令。

1. 登录 Amazon ECR 中的 AWS IoT Greengrass 注册表。

```
aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

如果成功，输出将打印 `Login Succeeded`。

2. 检索 AWS IoT Greengrass 容器镜像。

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

> **Note**
>
> `latest` 映像包含安装在 Amazon Linux 2 基本映像上的 AWS IoT Greengrass Core 软件的最新稳定版本。您还可以从存储库中拉取其他版本。要查找所有可用映像的标签，请查看 Docker Hub 上的 标签页面或使用 aws ecr list-images 命令。例如：
>
> ```
> aws ecr list-images --region us-west-2 --registry-id 216483018798 --
> repository-name aws-iot-greengrass
> ```

拉取容器镜像 (Windows)

在命令提示符中运行以下命令。Docker 桌面必须先处于运行状态，然后您才能在 Windows 上使用 Docker 命令。

1. 登录 Amazon ECR 中的 AWS IoT Greengrass 注册表。

```
aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

如果成功，输出将打印 `Login Succeeded`。

2. 检索 AWS IoT Greengrass 容器镜像。

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

> **Note**
>
> `latest` 映像包含安装在 Amazon Linux 2 基本映像上的 AWS IoT Greengrass Core 软件的最新稳定版本。您还可以从存储库中拉取其他版本。要查找所有可用映像的标签，请查看 Docker Hub 上的 标签页面或使用 aws ecr list-images 命令。例如：

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

步骤 2：创建和配置 Greengrass 组和核心

虽然 Docker 镜像已安装 AWS IoT Greengrass 核心软件，但您必须创建 Greengrass 组和核心。这包括下载证书和核心配置文件。

- 按 the section called “模块 2：安装 AWS IoT Greengrass Core 软件” 中的步骤操作。跳过下载和运行 AWS IoT Greengrass 核心软件的步骤。Docker 镜像中已设置软件及其运行时依赖项。

步骤 3：本地运行 AWS IoT Greengrass

配置您的组后，便能配置和启动核心。有关演示如何执行此操作的步骤，请选择您的操作系统：

本地运行 Greengrass (Linux)

在计算机终端中运行以下命令。

1. 为设备的安全资源创建一个文件夹，然后将证书和密钥移入该文件夹中。运行以下命令。将 *path-to-security-files* 替换为安全资源的路径，并将文件名称中的 *certificateId* 替换为证书 ID。

   ```
   mkdir /tmp/certs
   mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
   mv path-to-security-files/certificateId-public.pem.key /tmp/certs
   mv path-to-security-files/certificateId-private.pem.key /tmp/certs
   mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
   ```

2. 为设备配置创建一个文件夹，然后将 AWS IoT Greengrass Core 配置文件移入该文件夹中。运行以下命令。将 *path-to-config-file* 替换为指向配置文件的路径。

   ```
   mkdir /tmp/config
   mv path-to-config-file/config.json /tmp/config
   ```

3. 启动 AWS IoT Greengrass 并将证书和配置文件绑定挂载到 Docker 容器中。

   将 `/tmp` 替换为您解压证书和配置文件的路径。

```
docker run --rm --init -it --name aws-iot-greengrass \
--entrypoint /greengrass-entrypoint.sh \
-v /tmp/certs:/greengrass/certs \
-v /tmp/config:/greengrass/config \
-p 8883:8883 \
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

输出应类似于此示例：

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

本地运行 Greengrass (macOS)

在计算机终端中运行以下命令。

1. 为设备的安全资源创建一个文件夹，然后将证书和密钥移入该文件夹中。运行以下命令。将 *path-to-security-files* 替换为安全资源的路径，并将文件名称中的 *certificateId* 替换为证书 ID。

   ```
   mkdir /tmp/certs
   mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
   mv path-to-security-files/certificateId-public.pem.key /tmp/certs
   mv path-to-security-files/certificateId-private.pem.key /tmp/certs
   mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
   ```

2. 为设备配置创建一个文件夹，然后将 AWS IoT Greengrass Core 配置文件移入该文件夹中。运行以下命令。将 *path-to-config-file* 替换为指向配置文件的路径。

   ```
   mkdir /tmp/config
   mv path-to-config-file/config.json /tmp/config
   ```

3. 启动 AWS IoT Greengrass 并将证书和配置文件绑定挂载到 Docker 容器中。

   将 `/tmp` 替换为您解压证书和配置文件的路径。

```
docker run --rm --init -it --name aws-iot-greengrass \
--entrypoint /greengrass-entrypoint.sh \
-v /tmp/certs:/greengrass/certs \
-v /tmp/config:/greengrass/config \
-p 8883:8883 \
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

输出应类似于此示例：

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

本地运行 Greengrass (Windows)

1. 为设备的安全资源创建一个文件夹，然后将证书和密钥移入该文件夹中。在命令提示符中运行以下命令。将 *path-to-security-files* 替换为安全资源的路径，并将文件名称中的 *certificateId* 替换为证书 ID。

   ```
   mkdir C:\Users\%USERNAME%\Downloads\certs
   move path-to-security-files\certificateId-certificate.pem.crt C:\Users\%USERNAME%
   \Downloads\certs
   move path-to-security-files\certificateId-public.pem.key C:\Users\%USERNAME%
   \Downloads\certs
   move path-to-security-files\certificateId-private.pem.key C:\Users\%USERNAME%
   \Downloads\certs
   move path-to-security-files\AmazonRootCA1.pem C:\Users\%USERNAME%\Downloads\certs
   ```

2. 为设备配置创建一个文件夹，然后将 AWS IoT Greengrass Core 配置文件移入该文件夹中。在命令提示符中运行以下命令。将 *path-to-config-file* 替换为指向配置文件的路径。

   ```
   mkdir C:\Users\%USERNAME%\Downloads\config
   move path-to-config-file\config.json C:\Users\%USERNAME%\Downloads\config
   ```

3. 启动 AWS IoT Greengrass 并将证书和配置文件绑定挂载到 Docker 容器中。在命令提示符处运行以下命令。

```
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs
 -v c:/Users/%USERNAME%/Downloads/config:/greengrass/config -p 8883:8883
 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

当 Docker 提示您与 Docker 守护程序共享 `C:\` 驱动器时，允许它在 Docker 容器内绑定挂载 `C:\` 目录。有关更多信息，请参阅 Docker 文档中的共享驱动器。

输出应类似于此示例：

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

> **Note**
>
> 如果容器不打开 shell 并立即退出，则可以在启动映像时通过绑定挂载 Greengrass 运行时日志来调试问题。有关更多信息，请参阅the section called “在 Docker 容器之外保留 Greengrass 运行时日志”。

步骤 4：为 Greengrass 组配置“No container (无容器)”容器化

在 Docker 容器中运行 AWS IoT Greengrass 时，所有 Lambda 函数都必须在不进行容器化的情况下运行。在此步骤中，将组的默认容器化设置为 No container (无容器)。在首次部署组之前，您必须执行此操作。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择要更改其设置的组。
3. 选择 Lambda 函数选项卡。
4. 在默认 Lambda 函数运行时环境下，选择编辑。
5. 在编辑默认 Lambda 函数运行时环境中，更改 Lambda 函数容器化下的容器化设置。
6. 选择 Save（保存）。

这些更改将在部署组时生效。

有关更多信息，请参阅the section called “在组中设置 Lambda 函数的默认容器化”。

> **Note**
>
> 默认情况下，Lambda 函数使用组容器化设置。当 AWS IoT Greengrass 在 Docker 容器中运行时，如果您覆盖任何 Lambda 函数的无容器设置，则部署将失败。

步骤 5：将 Lambda 函数部署到 AWS IoT Greengrass Docker 容器

您可以将长期存在的 Lambda 函数部署到 Greengrass Docker 容器。

- 执行 the section called “模块 3（第 1 部分）：AWS IoT Greengrass 的 Lambda 函数” 中的步骤以将长期存在的 Hello World Lambda 函数部署到容器。

步骤 6：（可选）部署与 Docker 容器中运行的 Greengrass 交互的客户端设备

您还可以部署客户端设备，此设备在 Docker 容器中运行时将与 AWS IoT Greengrass 交互。

- 执行 the section called “模块 4：在 AWS IoT Greengrass 组中与客户端设备交互” 中的步骤以部署连接到核心并发送 MQTT 消息的客户端设备。

停止 AWS IoT Greengrass Docker 容器

要停止 AWS IoT Greengrass Docker 容器，请在您的终端或命令提示符处按 Ctrl+C。此操作将向 Greengrass 守护程序进程发送 `SIGTERM` 以停用 Greengrass 守护程序进程及其启动的所有 Lambda 进程。Docker 容器通过 `/dev/init` 进程初始化为 PID 1，这有助于删除任何剩余的僵尸进程。有关更多信息，请参阅 Docker 运行参考。

对 Docker 容器中的 AWS IoT Greengrass 执行问题排查

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。

错误：无法从非 TTY 设备执行交互式登录。

解决方案：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。

解决方案：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`）。如果您使用的是 Windows，并且您已使用 MSI 安装程序安装 CLI，则必须重复安装过程。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。

警告：IPv4 处于禁用状态。网络将不起作用。

解决方案：当在 Linux 计算机上运行 AWS IoT Greengrass 时，您可能会收到此警告或类似的消息。按照此步骤中所述进行操作来启用 IPv4 网络转发。AWS IoT Greengrass 云部署和 MQTT 通信在未启用 IPv4 转发时将不运行。有关更多信息，请参阅 Docker 文档中的在运行时配置具有命名空间的内核参数 (sysctls)。

错误：防火墙阻止 Windows 和容器之间的文件共享。

解决方案：在 Windows 计算机上运行 Docker 时，您可能会收到此错误或 `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 操作时发生错误 (AccessDeniedException): 用户 arn:aws:iam::<account-id>:user/<user-name> 无权对资源执行 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。

对于一般 AWS IoT Greengrass 问题排查帮助，请参阅 排查问题。

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 容器之外保留 Greengrass 运行时日志

绑定挂载 `/greengrass/ggc/var/log` 目录后，您可以运行 AWS IoT Greengrass Docker 容器。即使容器退出或被删除，日志仍然存在。

在 Linux 或 macOS 上

停止任何在主机上运行的 Greengrass Docker 容器，然后在终端中运行以下命令。这会绑定挂载 Greengrass `log` 目录并启动 Docker 镜像。

将 `/tmp` 替换为您解压证书和配置文件的路径。

```
docker run --rm --init -it --name aws-iot-greengrass \
      --entrypoint /greengrass-entrypoint.sh \
      -v /tmp/certs:/greengrass/certs \
      -v /tmp/config:/greengrass/config \
      -v /tmp/log:/greengrass/ggc/var/log \
      -p 8883:8883 \
      216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

然后，您可以在主机上的 `/tmp/log` 中检查日志，以查看 Greengrass 在 Docker 容器内运行时发生的情况。

在 Windows 上

停止任何在主机上运行的 Greengrass Docker 容器，然后在命令提示符中运行以下命令。这会绑定挂载 Greengrass `log` 目录并启动 Docker 镜像。

```
cd C:\Users\%USERNAME%\Downloads
mkdir log
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs -v c:/
Users/%USERNAME%/Downloads/config:/greengrass/config -v c:/Users/%USERNAME%/
Downloads/log:/greengrass/ggc/var/log -p 8883:8883 216483018798.dkr.ecr.us-
west-2.amazonaws.com/aws-iot-greengrass:latest
```

然后，您可以在主机上的 `C:/Users/%USERNAME%/Downloads/log` 中检查日志，以查看 Greengrass 在 Docker 容器内运行时发生的情况。

将交互式 Shell 附加到 Docker 容器

您可以将交互式 shell 附加到正在运行的 AWS IoT Greengrass Docker 容器。这可以帮助您调查 Greengrass Docker 容器的状态。

在 Linux 或 macOS 上

当 Greengrass Docker 容器正在运行时，在单独的终端中运行以下命令。

```
docker exec -it $(docker ps -a -q -f "name=aws-iot-greengrass") /bin/bash
```

在 Windows 上

当 Greengrass Docker 容器正在运行时，在单独的命令提示符中运行以下命令。

```
docker ps -a -q -f "name=aws-iot-greengrass"
```

将 *gg-container-id* 替换为上一个命令的 `container_id` 结果。

```
docker exec -it gg-container-id /bin/bash
```

使用 Lambda 函数和连接器访问本地资源

此功能适用于 AWS IoT Greengrass Core v1.3 及更高版本。

借助 AWS IoT Greengrass，您可以在云中编写 AWS Lambda 函数并配置连接器，并将其部署到核心设备以便在本地执行。在运行 Linux 的 Greengrass 核心上，这些本地部署的 Lambda 函数和连接器可以访问 Greengrass 核心设备上实际存在的本地资源。例如，要与通过 Modbus 或 CANbus 连接的设备通信，您可以启用 Lambda 函数以访问核心设备上的串行端口。要配置对本地资源的安全访问，您必须确保物理硬件以及 Greengrass 核心设备操作系统的安全性。

要开始访问本地资源，请查看以下教程：

- 如何使用 AWS 命令行界面配置本地资源访问
- 如何使用 AWS Management Console配置本地资源访问

支持的资源类型

您可以访问两种类型的本地资源：卷资源和设备资源。

卷资源

根文件系统上的文件或目录（/sys、/dev 或 /var 中的文件或目录除外）。其中包括：

- 用于在 Greengrass Lambda 函数中读取或写入信息的文件夹或文件（例如，/usr/lib/python2.x/site-packages/local）。
- 主机的 /proc 文件系统中的文件夹或文件（例如，/proc/net 或 /proc/stat）。在 v1.6 或更高版本中受到支持。有关其他要求，请参阅the section called “/proc 目录下的卷资源”。

> **Tip**
>
> 要将 /var、/var/run 和 /var/lib 目录配置为卷资源，请先将目录挂载到不同的文件夹，然后配置文件夹作为卷资源。

配置卷资源时，您可以指定源路径和目标路径。源路径是资源在主机上的绝对路径。目的地路径是资源在 Lambda 命名空间环境内的绝对路径。这是 Greengrass Lambda 函数或连接器在其中运行的容器。对目标路径的任何更改都会反映在主机文件系统上的源路径中。

> **Note**
>
> 目标路径中的文件仅在 Lambda 命名空间中可见。您在常规 Linux 命名空间中看不到它们。

设备资源

/dev 下的文件。设备资源只允许 /dev 下的字符设备或块存储设备。其中包括：

- 用于与通过串行端口连接的设备进行通信的串行端口（例如，/dev/ttyS0、/dev/ttyS1）。
- 用于连接 USB 外围设备的 USB（例如，/dev/ttyUSB0 或 /dev/bus/usb）。
- 用于通过 GPIO 连接的传感器和致动器的 GPIO（例如，/dev/gpiomem）。
- 用于使用板载 GPU 加快机器学习的 GPU（例如，/dev/nvidia0）。
- 用于捕获图像和视频的摄像机（例如，/dev/video0）。

> **Note**
>
> /dev/shm 是一个例外。只能将其配置为卷资源。/dev/shm 下的资源必须授予 rw 权限。

AWS IoT Greengrass 还支持用于执行机器学习推理的资源类型。有关更多信息，请参阅[执行机器学习推理]。

要求

以下要求适用于配置对本地资源的安全访问：

- 您必须使用 AWS IoT Greengrass Core v1.3 或更高版本。要为主机的 /proc 目录创建资源，您必须使用 v1.6 或更高版本。
- 必须在 Greengrass 核心设备上正确安装本地资源（包括任何所需的驱动程序和库），并且该资源在使用期间持续可用。
- 所需的资源操作以及对资源的访问不得要求根权限。
- 只有 `read` 或 `read and write` 权限可用。Lambda 函数无法对资源执行特权操作。
- 您必须提供 Greengrass 核心设备的操作系统上的本地资源的完整路径。
- 资源名称或 ID 的最大长度为 128 个字符，并且必须使用模式 `[a-zA-Z0-9:_-]+`。

/proc 目录下的卷资源

以下注意事项适用于主机的 /proc 目录中的卷资源。

- 您必须使用 AWS IoT Greengrass Core v1.6 或更高版本。
- 您可以允许 Lambda 函数进行只读访问，但不能进行读写访问。该访问级别是由 AWS IoT Greengrass 管理的。
- 您可能还需要授予操作系统组权限以在文件系统中启用读取访问。例如，假设源目录或文件具有 660 文件权限，这意味着仅组中的所有者或用户具有读取（和写入）访问权限。在这种情况下，您必须添加资源的操作系统组所有者权限。有关更多信息，请参阅the section called “组所有者文件访问权限”。
- 主机环境和 Lambda 命名空间均包含 /proc 目录，因此，请务必在指定目标路径时避免命名冲突。例如，如果 /proc 是源路径，您可以将 /host-proc 指定为目标路径（或“/proc”以外的任何路径名称）。

组所有者文件访问权限

AWS IoT Greengrass Lambda 函数进程通常以 `ggc_user` 和 `ggc_group` 身份运行。不过，您可以在本地资源定义中，向 Lambda 函数赋予额外的文件访问权限，如下所示：

- 要添加拥有资源的 Linux 组的权限，请使用 `GroupOwnerSetting#AutoAddGroupOwner` 参数或自动添加拥有资源的 Linux 组的操作系统组权限控制台选项。
- 要添加不同 Linux 组的权限，请使用 `GroupOwnerSetting#GroupOwner` 参数或 指定另一个系统组以添加文件系统权限控制台选项。如果 `GroupOwnerSetting#AutoAddGroupOwner` 为 true，则忽略 `GroupOwner` 值。

AWS IoT Greengrass Lambda 函数进程继承 `ggc_user`、`ggc_group` 和 Linux 组（如已添加）的所有文件系统权限。要使 Lambda 函数能够访问资源，Lambda 函数进程必须具有资源的所需权限。如有必要，您可以使用 `chmod(1)` 命令更改资源的权限。

另请参阅

- 《Amazon Web Services 一般参考》中的资源服务限额

如何使用 AWS 命令行界面配置本地资源访问

此功能适用于 AWS IoT Greengrass Core v1.3 及更高版本。

要使用本地资源，您必须在部署到 Greengrass 核心设备的组定义中添加资源定义。该组定义还必须包含一个 Lambda 函数定义，以便在其中向 Lambda 函数授予本地资源的访问权限。有关更多信息（包括要求和约束），请参阅使用 Lambda 函数和连接器访问本地资源。

本教程介绍了使用 AWS Command Line Interface (CLI) 创建本地资源并配置其访问权限的过程。要执行本教程中的步骤，您必须已创建一个 Greengrass 组，如入门 AWS IoT Greengrass 中所述。

有关使用 AWS Management Console的教程，请参阅如何使用 AWS Management Console配置本地资源访问。

创建本地资源

首先，您使用 `CreateResourceDefinition` 命令创建一个资源定义以指定要访问的资源。在此示例中，我们创建两个资源（`TestDirectory` 和 `TestCamera`）：

```
aws greengrass create-resource-definition  --cli-input-json '{
    "Name": "MyLocalVolumeResource",
    "InitialVersion": {
        "Resources": [
            {
                "Id": "data-volume",
                "Name": "TestDirectory",
                "ResourceDataContainer": {
                    "LocalVolumeResourceData": {
                        "SourcePath": "/src/LRAtest",
                        "DestinationPath": "/dest/LRAtest",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": true,
                            "GroupOwner": ""
                        }
                    }
                }
            },
            {
                "Id": "data-device",
                "Name": "TestCamera",
                "ResourceDataContainer": {
```

```
                    "LocalDeviceResourceData": {
                        "SourcePath": "/dev/video0",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": true,
                            "GroupOwner": ""
                        }
                    }
                }
            }
        ]
    }
}'
```

资源：Greengrass 组中的 `Resource` 对象列表。一个 Greengrass 组最多可包含 50 个资源。

Resource#Id：资源的唯一标识符。该 ID 用于在 Lambda 函数配置中引用资源。最大长度为 128 个字符。模式：[a-zA-Z0-9:_-]+。

Resource#Name：资源的名称。资源名称显示在 Greengrass 控制台中。最大长度为 128 个字符。模式：[a-zA-Z0-9:_-]+。

LocalDeviceResourceData# SourcePath: 设备资源的本地绝对路径。设备资源的源路径只能引用 `/dev` 中的字符设备或块设备。

LocalVolumeResourceData# SourcePath: Greengrass 核心设备上卷资源的本地绝对路径。此位置位于函数在其中运行的容器之外。卷资源类型的源路径不能以 `/sys` 开头。

LocalVolumeResourceData# DestinationPath：Lambda 环境中卷资源的绝对路径。此位置位于函数在其中运行的容器之内。

GroupOwnerSetting：允许您为 Lambda 流程配置其他群组权限。该字段是可选的。有关更多信息，请参阅 组所有者文件访问权限。

GroupOwnerSetting# AutoAddGroupOwner: 如果为真，Greengrass 会自动将该资源的指定 Linux 操作系统组所有者添加到 Lambda 进程权限中。因此，Lambda 进程具有添加的 Linux 组的文件访问权限。

GroupOwnerSetting# GroupOwner: 指定其权限已添加到 Lambda 进程的 Linux 操作系统组的名称。该字段是可选的。

`CreateResourceDefinition` 将返回一个资源定义版本 ARN。在更新组定义时，应使用该 ARN。

```
{
    "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/resources/ab14d0b5-116e-4951-a322-9cde24a30373/versions/a4d9b882-
d025-4760-9cfe-9d4fada5390d",
    "Name": "MyLocalVolumeResource",
    "LastUpdatedTimestamp": "2017-11-15T01:18:42.153Z",
    "LatestVersion": "a4d9b882-d025-4760-9cfe-9d4fada5390d",
    "CreationTimestamp": "2017-11-15T01:18:42.153Z",
    "Id": "ab14d0b5-116e-4951-a322-9cde24a30373",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/resources/
ab14d0b5-116e-4951-a322-9cde24a30373"
}
```

创建 Greengrass 函数

在创建资源后，请使用 CreateFunctionDefinition 命令创建 Greengrass 函数，并为该函数授予资源访问权限：

```
aws greengrass create-function-definition --cli-input-json '{
    "Name": "MyFunctionDefinition",
    "InitialVersion": {
        "Functions": [
            {
                "Id": "greengrassLraTest",
                "FunctionArn": "arn:aws:lambda:us-
west-2:012345678901:function:lraTest:1",
                "FunctionConfiguration": {
                    "Pinned": false,
                    "MemorySize": 16384,
                    "Timeout": 30,
                    "Environment": {
                        "ResourceAccessPolicies": [
                            {
                                "ResourceId": "data-volume",
                                "Permission": "rw"
                            },
                            {
                                "ResourceId": "data-device",
                                "Permission": "ro"
                            }
                        ],
                        "AccessSysfs": true
```

```
                    }
                }
            }
        ]
    }
}'
```

ResourceAccessPolicies：包含`resourceId`和，`permission`它们授予 Lambda 函数访问资源的权限。Lambda 函数最多可以访问 20 个资源。

ResourceAccessPolicy#Permission: 指定 Lambda 函数对资源拥有哪些权限。可用的选项为 `rw`（读/写）或 `ro`（只读）。

AccessSysfs: 如果为真，则 Lambda 进程可以对 Greengras `/sys` s 核心设备上的文件夹具有读取权限。这在 Greengrass Lambda 函数需要从 `/sys` 读取设备信息的情况下使用。

同样，`CreateFunctionDefinition` 返回一函数定义版本 ARN。应在组定义版本中使用该 ARN。

```
{
    "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad/versions/37f0d50e-ef50-4faf-
b125-ade8ed12336e",
    "Name": "MyFunctionDefinition",
    "LastUpdatedTimestamp": "2017-11-22T02:28:02.325Z",
    "LatestVersion": "37f0d50e-ef50-4faf-b125-ade8ed12336e",
    "CreationTimestamp": "2017-11-22T02:28:02.325Z",
    "Id": "3c9b1685-634f-4592-8dfd-7ae1183c28ad",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad"
}
```

将 Lambda 函数添加到组

最后，使用 `CreateGroupVersion` 将该函数添加到组中。例如：

```
aws greengrass create-group-version --group-id "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5" \
--resource-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/resources/db6bf40b-29d3-4c4e-9574-21ab7d74316c/versions/31d0010f-
e19a-4c4c-8098-68b79906fb87" \
--core-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/cores/adbf3475-f6f3-48e1-84d6-502f02729067/
versions/297c419a-9deb-46dd-8ccc-341fc670138b" \
```

```
--function-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/functions/d1123830-da38-4c4c-a4b7-e92eec7b6d3e/versions/a2e90400-
caae-4ffd-b23a-db1892a33c78" \
--subscription-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/subscriptions/7a8ef3d8-1de3-426c-9554-5b55a32fbcb6/
versions/470c858c-7eb3-4abd-9d48-230236bfbf6a"
```

> **Note**
>
> 要了解如何获取用于此命令的组 ID，请参阅the section called “获取组 ID”。

返回新组版本：

```
{
    "Arn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/groups/
b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5/versions/291917fb-ec54-4895-823e-27b52da25481",
    "Version": "291917fb-ec54-4895-823e-27b52da25481",
    "CreationTimestamp": "2017-11-22T01:47:22.487Z",
    "Id": "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5"
}
```

您的 Greengrass 群组现在包含 LRATest Lambda 函数，该函数可以访问两个资源：和。TestDirectory TestCamera

以下示例 Lambda 函数 (`lraTest.py`) 是使用 Python 编写的，它写入到本地卷资源：

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.

import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)
```

```
# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

这些命令由 Greengrass API 提供，用于创建和管理资源定义及资源定义版本：

- CreateResourceDefinition
- CreateResourceDefinitionVersion
- DeleteResourceDefinition
- GetResourceDefinition
- GetResourceDefinitionVersion
- ListResourceDefinitions
- ListResourceDefinitionVersions
- UpdateResourceDefinition

排查问题

- 问：为什么我的 Greengrass 组部署失败，错误类似于：

  ```
  group config is invalid:
      ggc_user or [ggc_group root tty] don't have ro permission on the file: /dev/tty0
  ```

  答：此错误表明该 Lambda 进程没有访问指定资源的权限。解决方案是更改资源的文件权限，以便 Lambda 可以访问该资源。（请参阅组所有者文件访问权限以了解详细信息。）

- 问：将 /var/run 配置为卷资源时，为什么 Lambda 函数无法启动，且 runtime.log 中包含错误消息：

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
 container.
container_linux.go:259: starting container process caused "process_linux.go:345:
container init caused \"rootfs_linux.go:62: mounting \\\"/var/run\\\" to rootfs \\\"/
greengrass/ggc/packages/1.3.0/rootfs_sys\\\" at \\\"/greengrass/ggc/packages/1.3.0/
rootfs_sys/run\\\"
caused \\\"invalid argument\\\"\""
```

  答：AWS IoT Greengrass 核心当前不支持将 /var、/var/run 和 /var/lib 配置为卷资源。一种解决办法是首先在不同的文件夹中挂载 /var、/var/run 或 /var/lib，然后将该文件夹配置为卷资源。

- 问：将 /dev/shm 配置为具有只读权限的卷资源时，为什么 Lambda 函数无法启动并在 runtime.log 中显示错误：

```
[ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
 container.
container_linux.go:259: starting container process caused "process_linux.go:345:
container init caused \"rootfs_linux.go:62: mounting \\\"/dev/shm\\\" to rootfs \\\"/
greengrass/ggc/packages/1.3.0/rootfs_sys\\\" at \\\"/greengrass/ggc/packages/1.3.0/
rootfs_sys/dev/shm\\\"
caused \\\"operation not permitted\\\"\""”
```

  答：/dev/shm 只能配置为读/写。将资源权限更改为 rw 以解决该问题。

如何使用 AWS Management Console配置本地资源访问

此功能适用于 AWS IoT Greengrass Core v1.3 及更高版本。

您可以配置 Lambda 函数以安全访问主机 Greengrass 核心设备上的本地资源。本地资源是指物理上存在于主机中的总线和外围设备，或主机操作系统上的文件系统卷。有关更多信息（包括要求和约束），请参阅使用 Lambda 函数和连接器访问本地资源。

本教程介绍如何使用 AWS Management Console 配置对 AWS IoT Greengrass 核心设备上本地资源的访问权限。它包含以下高级步骤：

1. 创建 Lambda 函数部署程序包

2. 创建并发布 Lambda 函数
3. 将 Lambda 函数添加到组
4. 将本地资源添加到组
5. 将订阅添加到组
6. 部署组

有关使用 AWS Command Line Interface的教程，请参阅如何使用 AWS 命令行界面配置本地资源访问。

先决条件

要完成此教程，需要：

- Greengrass 组和 Greengrass Core（v1.3 或更高版本）。要创建 Greengrass 组或核心，请参阅入门 AWS IoT Greengrass。
- Greengrass 核心设备上的以下目录：
  - /src/LRAtest
  - /dest/LRAtest

  这些目录的所有者组必须具有对目录的读写访问权限。您可以使用以下命令授予访问权限：

```
sudo chmod 0775 /src/LRAtest
```

步骤 1：创建 Lambda 函数部署程序包

在该步骤中，您将创建一个 Lambda 函数部署程序包，这是包括函数代码和依赖项的 ZIP 文件。您还可以下载 AWS IoT Greengrass 核心开发工具包，以包括在程序包中作为依赖项。

1. 在您的计算机上，将以下 Python 脚本复制到名为 `lraTest.py` 的本地文件。这是 Lambda 函数的应用程序逻辑。

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.
```

```
import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass
 Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

2. 从 AWS IoT Greengrass Core 软件开发工具包下载页面，将适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包下载到您的计算机上。
3. 解压缩下载的程序包以获取软件开发工具包。软件开发工具包是 `greengrasssdk` 文件夹。
4. 将以下项目压缩到名为 `lraTestLambda.zip` 的文件中：

   - `lraTest.py`. 应用程序逻辑。
   - `greengrasssdk`. 所有 Python Lambda 函数必需的库。

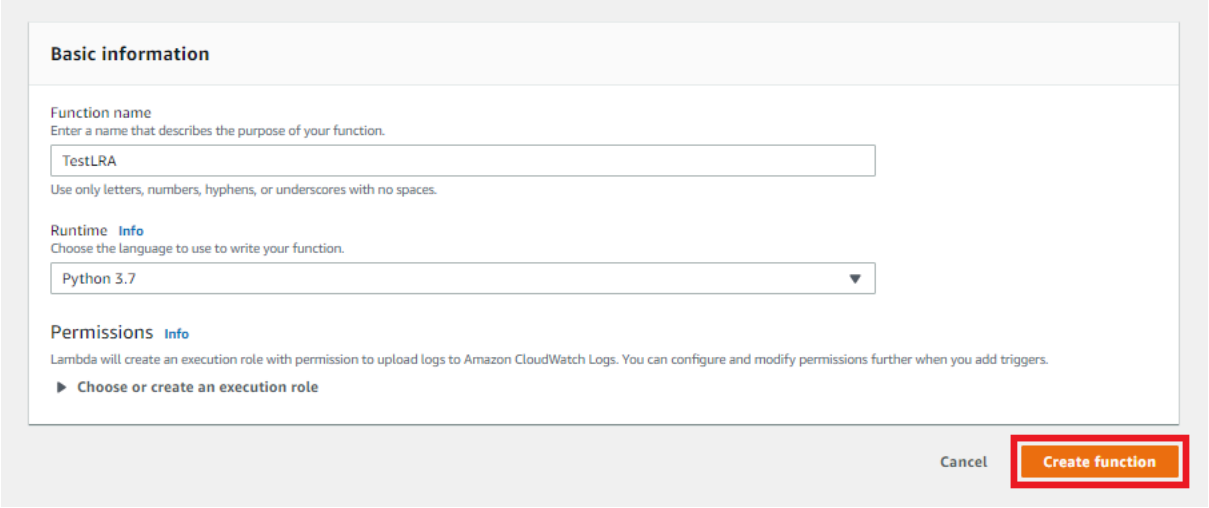
   `lraTestLambda.zip` 文件即 Lambda 函数部署程序包。现在您已准备好创建 Lambda 函数和上传部署程序包。

步骤 2：创建并发布 Lambda 函数

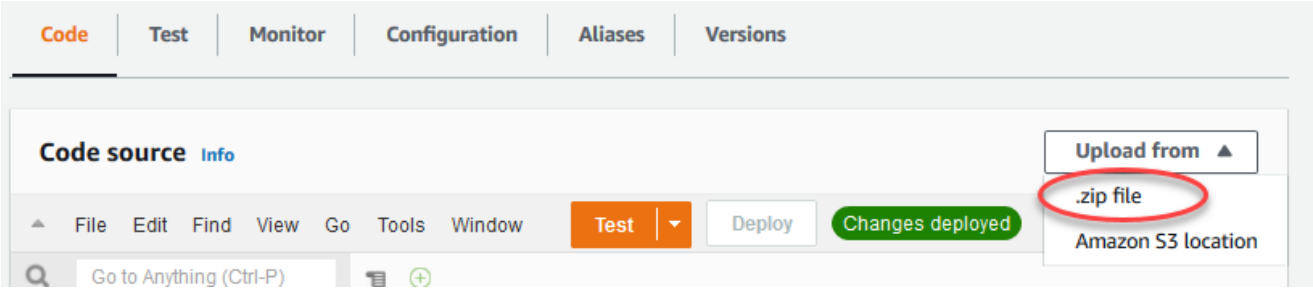
在该步骤中，您使用 AWS Lambda 控制台创建 Lambda 函数，然后将其配置为使用您的部署包。接着，发布函数版本并创建别名。

首先，创建 Lambda 函数。

1. 在 AWS Management Console中，选择 Services (服务)，然后打开 AWS Lambda 控制台。
2. 选择 函数。
3. 选择 创建函数，然后选择 从头开始创作。
4. 在 Basic information (基本信息) 部分中，使用以下值。
   a. 对于 Function name（函数名称），请输入 **`TestLRA`**。
   b. 对于 Runtime (运行时)，选择 Python 3.7。
   c. 对于权限，请保留默认设置。这将创建一个授予基本 Lambda 权限的执行角色。此角色未由 AWS IoT Greengrass 使用。
5. 选择 Create function（创建函数）。



6. 上传您的 Lambda 函数部署程序包并注册处理程序。
   a. 在代码选项卡上的代码源下，选择上传自。从下拉列表中选择 .zip 文件。



b. 选择上传，然后选择您的 `lraTestLambda.zip` 部署包。然后，选择 Save (保存)。

c. 在函数的代码选项卡中，在运行时设置下选择编辑，然后输入以下值。

- 对于 Runtime (运行时)，选择 Python 3.7。
- 对于 Handler (处理程序)，请输入 lraTest.function_handler。

d. 选择 Save（保存）。

> **Note**
>
> AWS Lambda 控制台上的测试按钮不适用于此功能。AWS IoT Greengrass Core 软件开发工具包不包含在 AWS Lambda 控制台中独立运行 Greengrass Lambda 函数所需的模块。这些模块（例如 `greengrass_common`）是在函数部署到您的 Greengrass 核心之后提供给它们的。

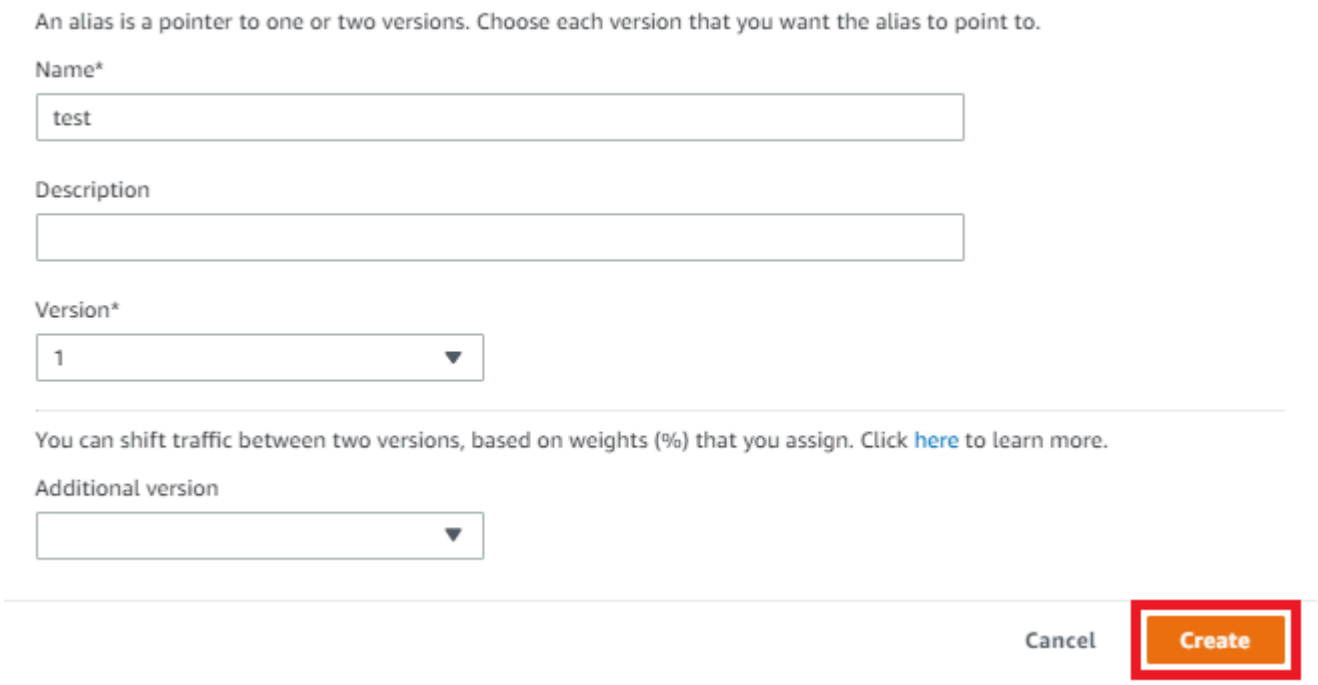
接下来，发布您的 Lambda 函数的第一个版本。然后，创建版本的别名。

Greengrass 组可以按别名（推荐）或版本引用 Lambda 函数。使用别名，您可以更轻松地管理代码更新，因为您在更新函数代码时，不必更改订阅表或组定义。相反，您只需将别名指向新的函数版本。

7. 在操作中，选择发布新版本。
8. 对于 Version description (版本描述)，输入 **`First version`**，然后选择 Publish (发布)。
9. 在 TestLRA: 1 配置页面上，从 Actions (操作) 中选择 Create alias (创建别名)。
10. 在 创建别名 页面上，对于 名称，输入 **`test`**。对于 Version (版本)，输入 1。

> **Note**
>
> AWS IoT Greengrass 不支持 $LATEST 版本的 Lambda 别名。

11. 选择 Create（创建）。



现在，您可以将 Lambda 函数添加到 Greengrass 组。

步骤 3：将 Lambda 函数添加到 Greengrass 组

在该步骤中，您将该函数添加到您的组并配置该函数的生命周期。

首先，将 Lambda 函数添加到 Greengrass 组。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择要在其中添加 Lambda 函数的 Greengrass 组。
3. 在组配置页面上，选择Lambda 函数选项卡。
4. 在我的 Lambda 函数部分下，选择添加。
5. 在 添加 Lambda 函数 页中，选择 Lambda 函数。Select **`TestLRA`**。
6. 选择 Lambda 函数版本。
7. 在 Lambda 函数配置部分中，选择系统用户和组和 Lambda 函数容器化。

接下来，配置 Lambda 函数的生命周期。

8. 对于 Timeout (超时)，选择 30 seconds (30 秒)。

> **Important**
>
> 使用本地资源的 Lambda 函数（如本过程中所述）必须在 Greengrass 容器中运行。否则，如果尝试部署该函数，则部署将失败。有关更多信息，请参阅容器化。

9. 在页面底部，选择添加 Lambda 函数。

步骤 4：将本地资源添加到 Greengrass 组

在该步骤中，您将本地卷资源添加到 Greengrass 组并为该函数授予对资源的读写访问权限。本地资源有一个群组级别的作用域。您可以给组中的任何 Lambda 函数授予权限以访问资源。

1. 在组配置页面上，选择资源选项卡。
2. 在本地资源部分下，选择添加。
3. 在 添加本地资源 页面上，使用以下值：

   a. 对于 Resource name (资源名称)，输入 **`testDirectory`**。

   b. 对于 Resource type (资源类型)，选择 Volume (卷)。

   c. 对于 本地设备路径，输入 **`/src/LRAtest`**。主机操作系统上必须存在该路径。

      本地设备路径是核心设备文件系统上的资源的绝对路径。此位置位于函数在其中运行的容器之外。该路径不能以 `/sys` 开头。

   d. 对于 Destination path (目的地路径)，输入 **`/dest/LRAtest`**。主机操作系统上必须存在该路径。

      目的地路径是 Lambda 命名空间中资源的绝对路径。此位置位于函数在其中运行的容器之内。

   e. 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限)。

      系统组所有者和文件访问权限选项可让您授予对 Lambda 进程的额外的文件访问权限。有关更多信息，请参阅组所有者文件访问权限。

4. 选择 Add resource (添加资源)。资源页面显示新的 testDirectory 资源。

步骤 5：将订阅添加到 Greengrass 组

在该步骤中，您将两个订阅添加到 Greengrass 组。这些订阅在 Lambda 函数与 AWS IoT 之间启用双向通信。

首先，为 Lambda 函数创建一个订阅，用于将消息发送到 AWS IoT。

1. 在组配置页面上，选择 订阅选项卡。
2. 选择 Add（添加）。
3. 在创建订阅页面中，按如下所述配置源和目标：

   a. 对于源类型，选择 Lambda 函数，然后选择 TestLRA。

   b. 对于目标类型，选择服务，然后选择 IoT 云。

   c. 对于主题筛选条件字段中，输入 **LRA/test**，然后选择订阅。
4. 订阅页面显示新订阅。

   接下来，配置订阅以从 AWS IoT 调用函数。
5. 在订阅页面，选择添加订阅。
6. 在选择您的源和目标页面，配置源和目标，如下所示：

   a. 在源类型中，选择 Lambda 函数，然后选择 IoT 云。

   b. 在目标类型中，选择服务，然后选择 TestLRA。

   c. 选择 Next（下一步）。
7. 在利用主题筛选您的数据页面上，对于主题筛选条件，输入 **invoke/LRAFunction**，然后选择下一步。
8. 选择 Finish (结束)。订阅页面显示两个订阅。

步骤 6：部署 AWS IoT Greengrass 组

在该步骤中，您将部署组定义的当前版本。

1. 确保 AWS IoT Greengrass 核心正在运行。根据需要在您的 Raspberry Pi 终端中运行以下命令。

   a. 要检查守护程序是否正在运行，请执行以下操作：

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

如果输出包含 root 的 /greengrass/ggc/packages/1.11.6/bin/daemon 条目，则表示守护程序正在运行。

> **Note**
>
> 路径中的版本取决于您的核心设备上安装的 AWS IoT Greengrass 核心软件版本。

b. 启动进程守护程序：

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

2. 在组配置页面上，选择部署。

> **Note**
>
> 如果在不进行容器化的情况下运行 Lambda 函数并且尝试访问附加的本地资源，部署将失败。

3. 如果出现提示，则在 Lambda 函数选项卡的系统 Lambda 函数下，选择 IP 检测器，然后选择编辑，然后选择自动检测。

这使得设备可以自动获取核心的连接信息，例如 IP 地址、DNS 和端口号。建议使用自动检测，不过 AWS IoT Greengrass 也支持手动指定的终端节点。只有在首次部署组时，系统才会提示您选择发现方法。

> **Note**
>
> 如果出现提示，请授予权限，以创建 Greengrass 服务角色并将其关联至当前 AWS 区域中的 AWS 账户。该角色将允许 AWS IoT Greengrass 访问您的 AWS 服务资源。

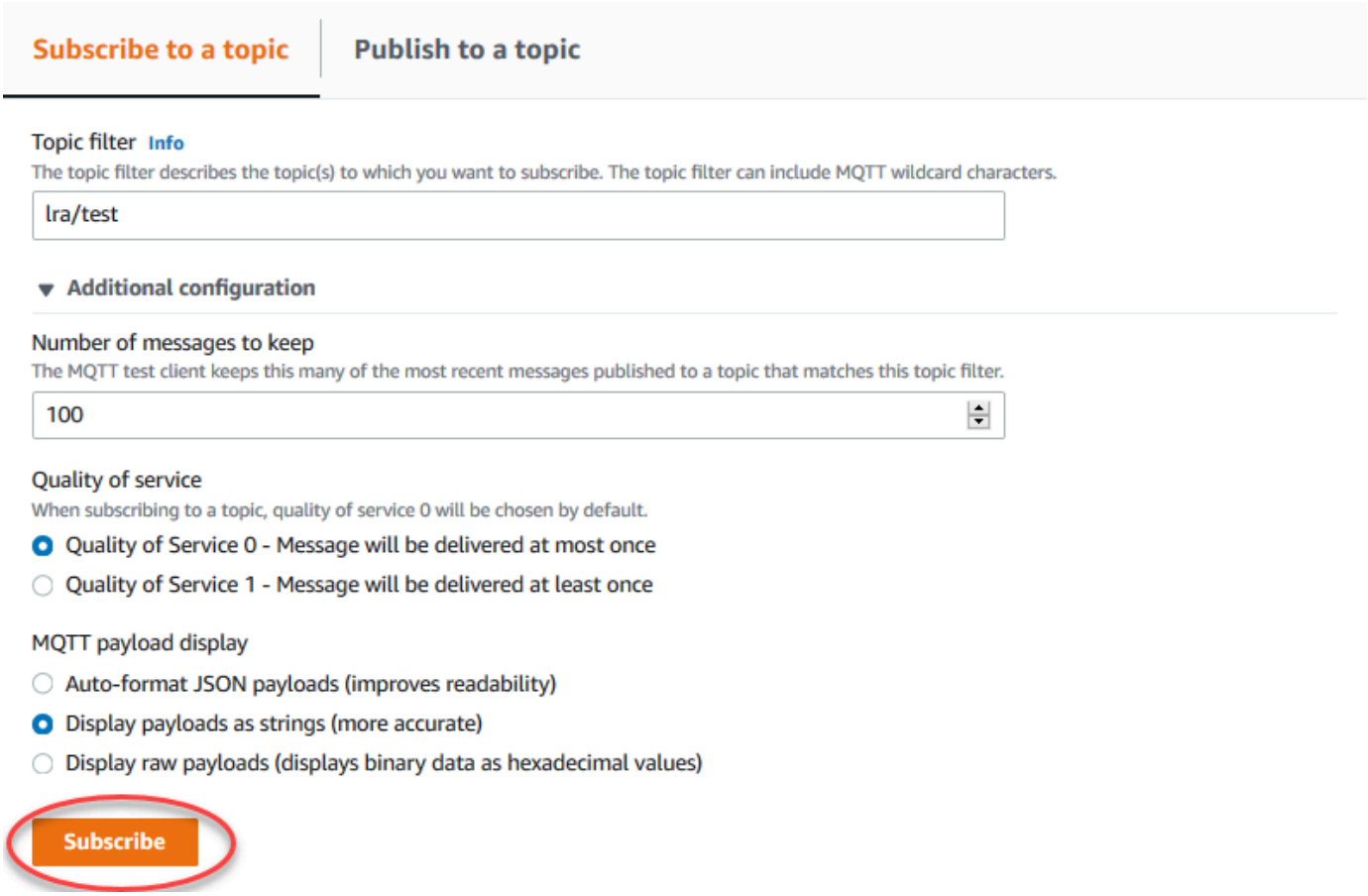
Deployments (部署) 页面显示了部署时间戳、版本 ID 和状态。完成后，部署状态为 已完成。

有关问题排查帮助，请参阅排查问题。

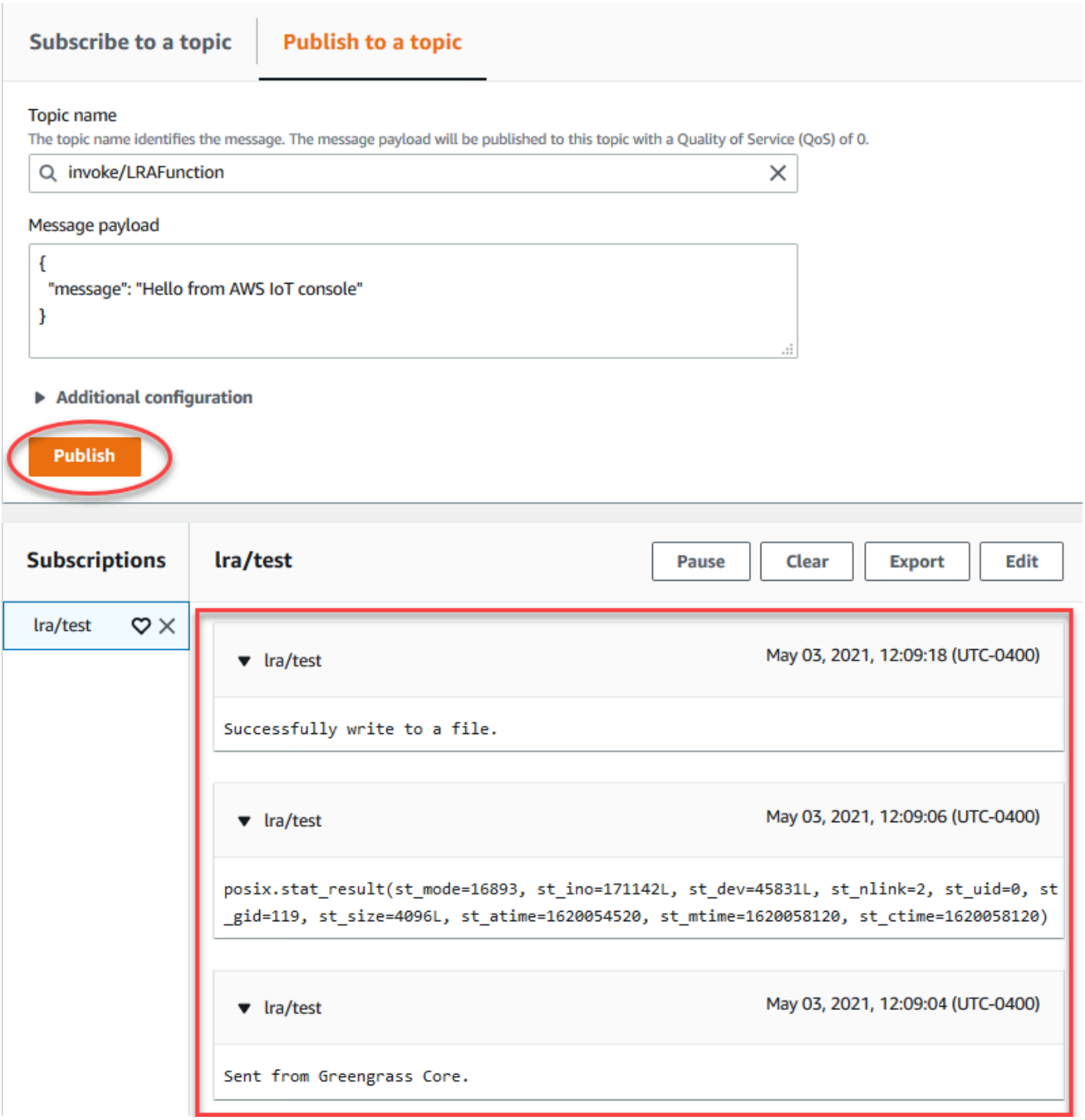
测试本地资源访问

现在，您可以验证是否正确配置了本地资源访问。要进行测试，您需要订阅 `LRA/test` 主题，并发布到 `invoke/LRAFunction` 主题。如果 Lambda 函数将预期的负载发送到 AWS IoT，则测试成功。

1. 在 AWS IoT 控制台导航菜单的测试下，选择 MQTT 测试客户端。
2. 在订阅主题下的主题筛选器中，输入 **`LRA/test`**。
3. 在其他信息下，对于 MQTT 负载显示，请选择将负载显示为字符串。
4. 选择 Subscribe。您的 Lambda 函数将发布到 LRA/test 主题。



5. 在发布到主题下的主题名称中输入 **`invoke/LRAFunction`**，然后选择发布以调用您的 Lambda 函数。如果页面显示函数的三个消息负载，则测试成功。



Lambda 函数创建的测试文件位于 Greengrass 核心设备上的 `/src/LRAtest` 目录中。尽管该 Lambda 函数写入到 `/dest/LRAtest` 目录中的文件，但该文件仅在 Lambda 命名空间中是可见的。您在常规 Linux 命名空间中看不到它。对目标路径的任何更改都会反映在文件系统上的源路径中。

有关问题排查帮助，请参阅排查问题。

执行机器学习推理

此功能适用于 AWS IoT Greengrass Core v1.6 或更高版本。

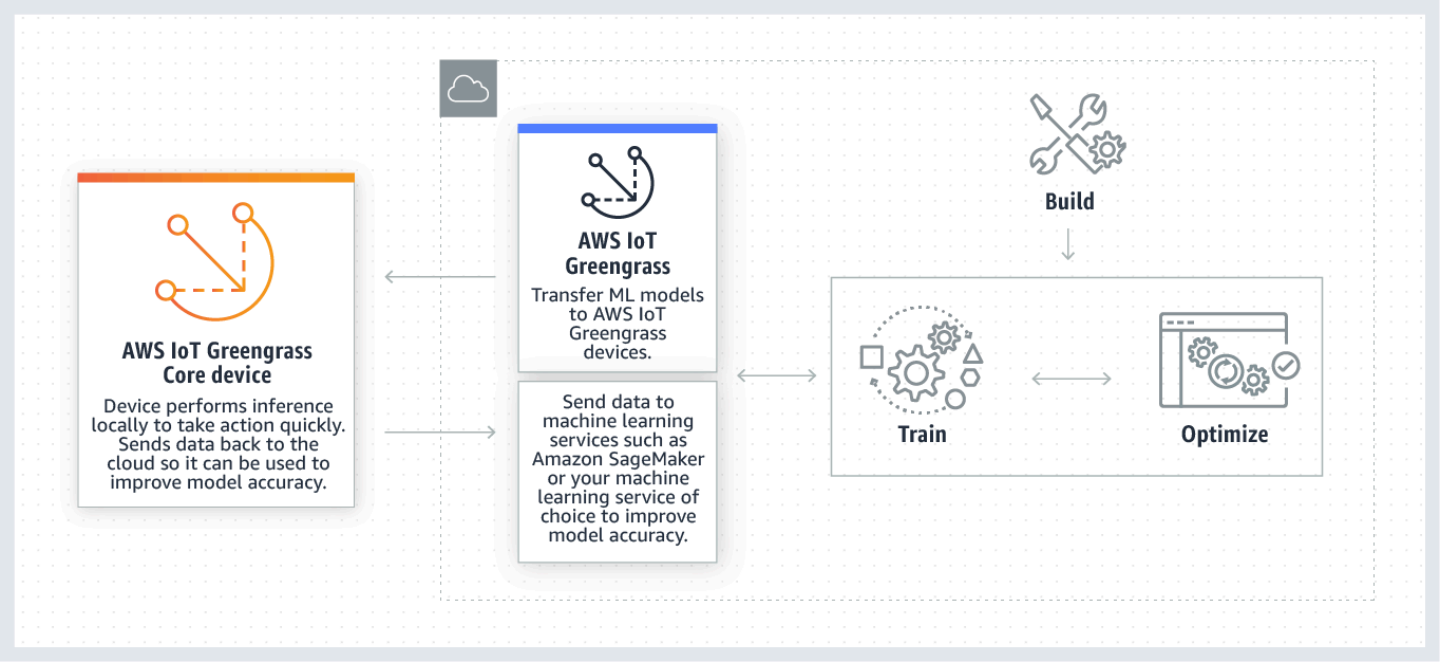
借助 AWS IoT Greengrass，您可以使用云训练模型对本地生成的数据在边缘站点执行机器学习 (ML) 推理。您可以从运行本地推理的低延迟和成本节省中受益，且仍然可以利用云计算在训练模型和复杂处理方面的强大功能。

要开始执行本地推理，请参阅the section called “如何配置机器学习推理”。

AWS IoT Greengrass ML 推理的工作原理

您可以在任何位置训练您的推理模型，在 Greengrass 组中将它们本地部署为机器学习资源，然后从 Greengrass Lambda 函数访问它们。例如，您可以在 SageMaker 中构建并训练深度学习模型，然后将其部署到 Greengrass 核心。然后，您的 Lambda 函数可以在连接的设备上使用本地模型来执行推理并将新训练数据发送回云中。

下图显示了 AWS IoT Greengrass ML 推理工作流。



AWS IoT Greengrass ML 推理简化了 ML 工作流的每个步骤，包括：

- 构建和部署机器学习框架原型。
- 访问云训练的模型并将其部署到 Greengrass 核心设备。
- 创建可以将硬件加速器 (如 GPU 和 FPGA) 作为本地资源访问的推理应用程序。

机器学习资源

机器学习资源表示部署到 AWS IoT Greengrass 核心的云训练推理模型。要部署机器学习资源，首先请将资源添加到 Greengrass 组，然后定义组中的 Lambda 函数如何才能访问它们。组部署期间，AWS IoT Greengrass 从云中检索源模型包，并将它们提取到 Lambda 运行时命名空间内的目录。然后，Greengrass Lambda 函数会使用本地部署的模型来执行推理。

要更新本地部署的模型，首先请更新与机器学习资源对应的源模型 (在云中)，然后部署组。部署期间，AWS IoT Greengrass 会检查更改的来源。如果检测到更改，则 AWS IoT Greengrass 会更新本地模型。

支持的模型源

AWS IoT Greengrass 支持机器学习资源的 SageMaker 和 Amazon S3 模型源。

以下要求适用于模型源：

- 存储您的 SageMaker 和 Amazon S3 模型源的 S3 存储桶不得使用 SSE-C 加密。对于使用服务器端加密的存储桶，AWS IoT Greengrass ML 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。
- 存储您的 SageMaker 和 Amazon S3 模型源的 S3 存储桶的名称不得包含句点 (`.`)。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。
- 服务级别 AWS 区域 支持必须同时可用于 AWS IoT Greengrass 和 SageMaker。当前，AWS IoT Greengrass 在以下区域中支持 SageMaker 模型：
  - 美国东部（俄亥俄州）
  - 美国东部（弗吉尼亚北部）
  - 美国西部（俄勒冈州）
  - Asia Pacific (Mumbai)
  - 亚太地区（首尔）
  - 亚太地区（新加坡）
  - 亚太地区（悉尼）
  - 亚太地区（东京）
  - 欧洲地区（法兰克福）
  - 欧洲地区（爱尔兰）

  - 欧洲地区（伦敦）
- AWS IoT Greengrass 必须具有对模型源的 `read` 权限，如以下部分中所述。

SageMaker

AWS IoT Greengrass 支持另存为 SageMaker 训练作业的模型。SageMaker 是一项完全托管的 ML 服务，可用于使用内置或自定义算法构建和训练模型。有关更多信息，请参阅 SageMaker 开发人员指南中的什么是 SageMaker？。

如果您通过创建存储桶 (存储桶名称包含 `sagemaker`) 来配置您的 SageMaker 环境，则 AWS IoT Greengrass 具有访问 SageMaker 训练作业的足够权限。`AWSGreengrassResourceAccessRolePolicy` 托管策略允许访问名称包含字符串 `sagemaker` 的存储桶。此策略将附加到 Greengrass 服务角色。

否则，您必须为 AWS IoT Greengrass 授予对存储您的训练作业的存储桶的 `read` 权限。为此，请在服务角色中嵌入以下内联策略。您可以列出多个存储桶 ARN。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket-name"
            ]
        }
    ]
}
```

Amazon S3

AWS IoT Greengrass 支持以 `tar.gz` 或 `.zip` 文件形式存储在 Amazon S3 中的模型。

要支持 AWS IoT Greengrass 访问在 Amazon S3 存储桶中存储的模型，您必须通过执行以下操作之一为 AWS IoT Greengrass 授予访问存储桶的 `read` 权限：

- 将您的模型存储在名称包含 `greengrass` 的存储桶中。

`AWSGreengrassResourceAccessRolePolicy` 托管策略允许访问名称包含字符串 `greengrass` 的存储桶。此策略将附加到 Greengrass 服务角色。

- 在 Greengrass 服务角色中嵌入一个内联策略。

  如果您的存储桶名称不包含 `greengrass`，请将以下内联策略添加到服务角色。您可以列出多个存储桶 ARN。

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "s3:GetObject"
              ],
              "Resource": [
                  "arn:aws:s3:::my-bucket-name"
              ]
          }
      ]
  }
  ```

  有关更多信息，请参阅 IAM 用户指南中的嵌入内联策略。

要求

以下要求适用于创建和使用机器学习资源：

- 您必须使用 AWS IoT Greengrass Core v1.6 或更高版本。
- 用户定义的 Lambda 函数可以对资源执行 `read` 或 `read and write` 操作。针对其他操作的权限不可用。附属 Lambda 函数的容器化模式决定如何设置访问权限。有关更多信息，请参阅the section called “访问机器学习资源”。
- 您必须提供核心设备的操作系统上的资源的完整路径。
- 资源名称或 ID 的最大长度为 128 个字符，并且必须使用模式 `[a-zA-Z0-9:_-]+`。

用于 ML 推理的运行时和库

您可以将以下 ML 运行时和库与 AWS IoT Greengrass 结合使用。

- Amazon SageMaker Neo 深度学习运行时
- Apache MXNet
- TensorFlow

这些运行时和库可以安装在 NVIDIA Jetson TX2、Intel Atom 和 Raspberry Pi 平台上。有关下载信息，请参阅the section called “支持的机器学习运行时和库”。您可以将它们直接安装到核心设备上。

请务必阅读以下有关兼容性和限制的信息。

SageMaker Neo 深度学习运行时

您可以使用 SageMaker Neo 深度学习运行时在 AWS IoT Greengrass 设备上执行具有优化机器学习模型的推理。使用 SageMaker Neo 深度学习编译器对这些模型进行优化，以提高机器学习推理预测的速度。有关 SageMaker 中模型优化的更多信息，请参阅 SageMaker Neo 文档。

> **Note**
>
> 目前，您只能在特定的 Amazon Web Services 区域使用 Neo 深度学习编译器来优化机器学习模型。但是，您可以将 Neo 深度学习运行时与每个 AWS 区域 中支持 AWS IoT Greengrass 内核的优化模型一起使用。有关信息，请参阅如何配置优化的机器学习推理

MXNet 版本控制

Apache MXNet 目前不确保向前兼容性，因此您使用框架的较高版本训练的模型可能无法在框架的较低版本中正常工作。为了避免模型训练和模型服务阶段之间的冲突，以及为了提供一致的端到端体验，请在这两个阶段使用相同的 MXNet 框架版本。

Raspberry Pi 上的 MXNet

访问本地 MXNet 模型的 Greengrass Lambda 函数必须设置以下环境变量：

```
MXNET_ENGINE_TYPE=NativeEngine
```

您可以在函数代码中设置该环境变量，或者将其添加到函数的组特定的配置。对于将该环境变量添加为配置设置的示例，请参阅此步骤。

> **Note**
>
> 对于 MXNet 框架的一般用途，如运行第三方代码示例，必须在 Raspberry Pi 上配置该环境变量。

对 Raspberry Pi 的 TensorFlow 模型服务限制

以下改进推理结果的建议基于我们在 Raspberry Pi 平台上使用 TensorFlow 32 位 Arm 库进行的测试。这些建议仅供高级用户参考，不包含任何类型的保证。

- 使用检查点格式训练的模型在服务之前应被“冻结”为协议缓冲格式。有关示例，请参阅 TensorFlow-Slim 图像分类模型库。
- 请勿在训练或推理代码中使用 TF-Estimator 和 TF-Slim 库。而应使用以下示例中所示的 `.pb` 文件模型加载模式。

```
graph = tf.Graph()
graph_def = tf.GraphDef()
graph_def.ParseFromString(pb_file.read())
with graph.as_default():
  tf.import_graph_def(graph_def)
```

> **Note**
>
> 有关 TensorFlow 的支持平台的更多信息，请参阅 TensorFlow 文档中的安装 TensorFlow。

从 Lambda 函数访问机器学习资源

用户定义的 Lambda 函数可以访问机器学习资源以在 AWS IoT Greengrass 核心上运行本地推断。机器学习资源由经过训练的模型和下载到核心设备的其他构件组成。

要允许 Lambda 函数访问核心上的机器学习资源，必须将该资源附加到 Lambda 函数并定义访问权限。关联（或附加）的 Lambda 函数的容器化模式决定了您如何执行此操作。

机器学习资源的访问权限

从 AWS IoT Greengrass 核心 v1.10.0 开始，您可以为机器学习资源定义资源所有者。资源所有者表示 AWS IoT Greengrass 用于下载资源构件的操作系统组和权限。如果未定义资源所有者，则下载的资源构件仅可供根用户访问。

- 如果非容器化 Lambda 函数访问机器学习资源，则必须定义资源所有者，因为容器中没有权限控制。非容器化 Lambda 函数可以继承资源所有者权限并使用它们访问资源。

- 如果只有容器化 Lambda 函数访问资源，我们建议您使用函数级别的权限，而不是定义资源所有者。

资源拥有者属性

资源所有者指定组所有者和组所有者权限。

组所有者。核心设备上现有 Linux 操作系统组的组 (GID) 的 ID。组的权限将添加到 Lambda 进程。具体而言，GID 将添加到 Lambda 函数的补充组 ID。

如果将 Greengrass 组中的 Lambda 函数配置为以与机器学习资源的资源所有者相同的操作系统组身份运行，则必须将该资源附加到该 Lambda 函数。否则，部署失败，因为此配置提供了隐式权限， Lambda 函数可以在未经 AWS IoT Greengrass 授权的情况下访问资源。如果 Lambda 函数以 root 身份运行，则会跳过部署验证检查 (UID=0)。

我们建议您使用 Greengrass 核心上的其他资源、 Lambda 函数或文件未使用的操作系统组。使用共享的操作系统组可为附加的 Lambda 函数提供比其所需更多的访问权限。如果使用共享的操作系统组，还必须将附加的 Lambda 函数附加到使用共享操作系统组的所有机器学习资源。否则，部署将失败。

组所有者权限。要添加到 Lambda 进程的只读或读取和写入权限。

非容器化 Lambda 函数必须继承对于资源的这些访问权限。容器化 Lambda 函数可以继承这些资源级权限或定义函数级权限。如果它们定义了函数级权限，则这些权限必须与资源级权限相同或更具限制性。

下表显示了受支持的访问权限配置。

GGC v1.10 or later

属性	如果只有容器化 Lambda 函数访问资源	如果任何非容器化 Lambda 函数访问资源
函数级属性		
权限（读/写）	除非资源定义了资源所有者，否则为必需的。如果定义了资源所有者，则函数级别的权限必须与资源所有者权限相同或更具限制性。 如果只有容器化 Lambda 函数访问资源，我们建议您不要定义资源所有者。	非容器化 Lambda 函数： 不支持。非容器化 Lambda 函数必须继承资源级权限。 容器化 Lambda 函数： 可选，但必须与资源级权限相同或更具限制性。
资源级属性		
资源拥有者	可选（不推荐）。	必需。
权限（读/写）	可选（不推荐）。	必需。

GGC v1.9 or earlier

属性	如果只有容器化 Lambda 函数访问资源	如果任何非容器化 Lambda 函数访问资源
函数级属性		
权限（读/写）	必需。	不支持。
资源级属性		
资源拥有者	不支持。	不支持。
权限（读/写）	不支持。	不支持。

> **Note**
>
> 当您使用 AWS IoT Greengrass API 配置 Lambda 函数和资源时，还需要函数级别 `ResourceId` 属性。`ResourceId` 属性将机器学习资源附加到 Lambda 函数。

定义 Lambda 函数的访问权限（控制台）

在 AWS IoT 控制台中，您可以在配置机器学习资源或将其附加到 Lambda 函数时定义访问权限。

容器化 Lambda 函数

如果只将容器化 Lambda 函数附加到机器学习资源：

- 选择 无系统组 作为机器学习资源的资源拥有者。当只有容器化 Lambda 函数访问机器学习资源时，这是建议使用的设置。否则，您可能会为附加的 Lambda 函数授予比其所需更多的访问权限。

非容器化 Lambda 函数（需要 GGC v1.10 或更高版本）

如果将任何非容器化 Lambda 函数附加到机器学习资源：

- 指定要用作机器学习资源的资源拥有者的系统组的 ID (GID)。选择 指定系统组和权限，然后输入 GID。您可以在您的核心设备上使用 `getent group` 命令查找系统组的 ID。

- 为系统组权限选择 只读访问权限或 读写访问权限。

定义 Lambda 函数的访问权限 (API)

在 AWS IoT Greengrass API 中，您可以在 Lambda 函数的 `ResourceAccessPolicy` 属性或机器学习资源的 `OwnerSetting` 属性中定义对此资源的权限。

容器化 Lambda 函数

如果只将容器化 Lambda 函数附加到机器学习资源：

- 对于容器化 Lambda 函数，请在 `ResourceAccessPolicies` 属性的 `Permission` 属性中定义访问权限。例如：

```
"Functions": [
    {
        "Id": "my-containerized-function",
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id",
                        "Permission": "ro-or-rw"
                    }
                ]
            },
            "MemorySize": 512,
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- 对于机器学习资源，请省略 OwnerSetting 属性。例如：

```
"Resources": [
    {
        "Id": "my-resource-id",
        "Name": "my-resource-name",
        "ResourceDataContainer": {
            "S3MachineLearningModelResourceData": {
                "DestinationPath": "/local-destination-path",
                "S3Uri": "s3://uri-to-resource-package"
            }
        }
    }
]
```

当只有容器化 Lambda 函数访问机器学习资源时，这是推荐的配置。否则，您可能会为附加的 Lambda 函数授予比其所需更多的访问权限。

非容器化 Lambda 函数（需要 GGC v1.10 或更高版本）

如果将任何非容器化 Lambda 函数附加到机器学习资源：

- 对于非容器化 Lambda 函数，请省略 `ResourceAccessPolicies` 中的 `Permission` 属性。此配置是必需的，并允许函数继承资源级权限。例如：

```
"Functions": [
    {
        "Id": "my-non-containerized-function",
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "Execution": {
                    "IsolationMode": "NoContainer",
                },
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id"
                    }
                ]
            },
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- 对于同时访问机器学习资源的容器化 Lambda 函数，请省略 `ResourceAccessPolicies` 中的 `Permission` 属性，或定义与资源级权限相同或更具限制性的权限。例如：

```
"Functions": [
    {
        "Id": "my-containerized-function",
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id",
                        "Permission": "ro-or-rw" // Optional, but cannot exceed the GroupPermission defined for the resource.
```

```
                    }
                ]
            },
            "MemorySize": 512,
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- 对于机器学习资源，请定义 OwnerSetting 属性，包括子 GroupOwner 和 GroupPermission 属性。例如：

```
"Resources": [
    {
        "Id": "my-resource-id",
        "Name": "my-resource-name",
        "ResourceDataContainer": {
            "S3MachineLearningModelResourceData": {
                "DestinationPath": "/local-destination-path",
                "S3Uri": "s3://uri-to-resource-package",
                "OwnerSetting": {
                    "GroupOwner": "os-group-id",
                    "GroupPermission": "ro-or-rw"
                }
            }
        }
    }
]
```

从 Lambda 函数代码访问机器学习资源

用户定义的 Lambda 函数使用特定于平台的操作系统接口以访问核心设备上的机器学习资源。

GGC v1.10 or later

对于容器化 Lambda 函数，资源安装在 Greengrass 容器内，并在为资源定义的本地目标路径中可用。对于非容器化 Lambda 函数，资源将符号链接到 Lambda 特定的工作目录并传递给 Lambda 进程中的 AWS_GG_RESOURCE_PREFIX 环境变量。

为获取机器学习资源的下载构件的路径，Lambda 函数将 AWS_GG_RESOURCE_PREFIX 环境变量附加到为资源定义的本地目标路径。对于容器化 Lambda 函数，返回的值是单个正斜杠 (/)。

```
resourcePath = os.getenv("AWS_GG_RESOURCE_PREFIX") + "/destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

GGC v1.9 or earlier

机器学习资源的下载构件位于为资源定义的本地目标路径中。只有容器化 Lambda 函数才能访问 AWS IoT Greengrass 核心 v1.9 及更早版本中的机器学习资源。

```
resourcePath = "/local-destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

您的模型加载实现取决于您的 ML 库。

排查问题

使用以下信息帮助解决访问机器学习资源的问题。

主题

- InvalidML ModelOwner - GroupOwnerSetting 在 ML 模型资源中提供，但不存在 GroupOwner 或 GroupPermission 不存在
- NoContainer 在附加 Machine Learning 资源时，函数无法配置权限。 <function-arn>指在资源 <resource-id><ro/rw> 访问策略中具有权限的机器学习资源。
- 函数<function-arn>指的是 Machine <resource-id>Learning 资源，但两者都缺少权限 ResourceAccessPolicy 和资源 OwnerSetting。
- 函数<function-arn>指的是具有<resource-id>\ “rw\” 权限的 Machine Learning 资源，而资源所有者设置 GroupPermission仅允许\ “ro\”。
- NoContainer 函数<function-arn>是指嵌套目标路径的资源。
- Lambda <function-arn> 通过共享同一组所有者 ID 获得对资源 <resource-id> 的访问权限

InvalidML ModelOwner - GroupOwnerSetting 在 ML 模型资源中提供，但不存在 GroupOwner 或 GroupPermission 不存在

解决方案：如果机器学习资源包含ResourceDownloadOwnerSetting对象，但未定义必需的`GroupOwner`或`GroupPermission`属性，则会收到此错误。要解决此问题，请定义缺失的属性。

NoContainer 在附加 Machine Learning 资源时，函数无法配置权限。 <function-arn>指在资源 <resource-id><ro/rw> 访问策略中具有权限的机器学习资源。

解决方案：如果非容器化 Lambda 函数指定了对机器学习资源的函数级权限，则会收到此错误。非容器化函数必须从在机器学习资源上定义的资源所有者权限继承权限。要解决此问题，请选择继承资源所有者权限（控制台）或从 Lambda 函数的资源访问策略 (API) 中删除权限。

函数<function-arn>指的是 Machine <resource-id>Learning 资源，但两者都缺少权限 ResourceAccessPolicy 和资源 OwnerSetting。

解决方案：如果未为附加的 Lambda 函数或资源配置对机器学习资源的权限，则会收到此错误。要解决此问题，请在 Lambda 函数的ResourceAccessPolicy属性或资源的OwnerSetting属性中配置权限。

函数<function-arn>指的是具有<resource-id>\ “rw\” 权限的 Machine Learning 资源，而资源所有者设置 GroupPermission仅允许\ “ro\”。

解决方案：如果为附加的 Lambda 函数定义的访问权限超过为机器学习资源定义的资源所有者权限，则会收到此错误。要解决此问题，请为 Lambda 函数设置限制更多的权限或为资源所有者设置限制较少的权限。

NoContainer 函数<function-arn>是指嵌套目标路径的资源。

解决方案：如果附加到非容器化 Lambda 函数的多个机器学习资源使用相同的目标路径或嵌套的目标路径，则会收到此错误。要解决此问题，请为资源指定单独的目标路径。

Lambda <function-arn> 通过共享同一组所有者 ID 获得对资源 <resource-id> 的访问权限

解决方案：如果将相同的操作系统组指定为 Lambda 函数的运行方式身份和机器学习资源的资源所有者，但该资源未附加到 Lambda 函数，您会 `runtime.log` 在中收到此错误。此配置为 Lambda 函数提供隐式权限，它可以使用这些权限来访问资源而无需 AWS IoT Greengrass 授权。

要解决此问题，请为其中一个属性使用不同的操作系统组，或将机器学习资源附加到 Lambda 函数。

另请参阅

- 执行机器学习推理
- the section called “如何配置机器学习推理”
- the section called “如何配置优化的机器学习推理”
- AWS IoT Greengrass Version 1 API Reference

如何使用 AWS Management Console配置机器学习推理

要按照本教程中的步骤操作，需要 AWS IoT Greengrass Core v1.10 或更高版本。

您可以使用本地生成的数据在 Greengrass 核心设备上本地执行机器学习 (ML) 推理。有关信息（包括要求和约束），请参阅执行机器学习推理。

本教程介绍如何使用 AWS Management Console 将 Greengrass 组配置为运行一个 Lambda 推理应用程序，该应用程序可在本地识别摄像机中的图像，而无需将数据发送至云。该推理应用程序将访问 Raspberry Pi 上的摄像机模块并使用开源 SqueezeNet 模型运行推理。

本教程包含以下概括步骤：

1. 配置 Raspberry Pi
2. 安装 MXNet 框架
3. 创建模型包
4. 创建并发布 Lambda 函数
5. 将 Lambda 函数添加到组
6. 将资源添加到组
7. 将订阅添加到组
8. 部署组
9. 测试应用程序

先决条件

要完成此教程，需要：

- Raspberry Pi 4 Model B 或 Raspberry Pi 3 Model B/B+，已设置并配置为与 AWS IoT Greengrass 结合使用。要使用 AWS IoT Greengrass 设置 Raspberry Pi，请运行 Greengrass 设备安装脚本，或确保您已完成 入门 AWS IoT Greengrass 的模块 1 和模块 2。

  > **Note**
  >
  > Raspberry Pi 可能需要一个 2.5A 的电源来运行通常用于图像分类的深度学习框架。额定值不足的电源可能会导致设备重新启动。

- Raspberry Pi 摄像机模块 V2 - 800 万像素，1080p。要了解关于如何设置摄像机的更多信息，请参阅 Raspberry Pi 文档中的连接摄像机。
- Greengrass 组和 Greengrass 核心。有关如何创建 Greengrass 组或核心的信息，请参阅 入门 AWS IoT Greengrass。

> **Note**
>
> 本教程使用的是 Raspberry Pi，但 AWS IoT Greengrass 支持其他平台，例如 Intel Atom 和 NVIDIA Jetson TX2。在 Jetson TX2 的示例中，可以使用静态图像，而不是从摄像机流式传输的图像。如果使用的是 Jetson TX2 示例，您可能需要安装 Python 3.6 而不是 Python 3.7。有关配置设备以便安装 AWS IoT Greengrass Core 软件的信息，请参阅 the section called “设置其他设备”。
>
> 对于 AWS IoT Greengrass 不支持的第三方平台，您必须在非容器化模式下运行 Lambda 函数。要在非容器化模式下运行，必须以 root 用户身份运行 Lambda 函数。有关更多信息，请参阅 the section called “选择 Lambda 函数容器化时的注意事项” 和 the section called “为组中的 Lambda 函数设置默认访问身份”：

步骤 1：配置 Raspberry Pi

在该步骤中，将安装 Raspbian 操作系统的更新，安装摄像机模块软件和 Python 依赖项，以及启用摄像机接口。

在您的 Raspberry Pi 终端中运行以下命令。

1. 安装 Raspbian 的更新。

   ```
   sudo apt-get update
   ```

```
sudo apt-get dist-upgrade
```

2. 安装适用于摄像机模块的 picamera 接口以及本教程所需的其他 Python 库。

   ```
   sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
   ```

   验证安装：

   - 确保您的 Python 3.7 安装包含 pip。

     ```
     python3 -m pip
     ```

     如果未安装 pip，请从 pip 网站 下载它，然后运行以下命令。

     ```
     python3 get-pip.py
     ```

   - 确保您的 Python 版本为 3.7 或更高版本。

     ```
     python3 --version
     ```

     如果输出列出了早期版本，请运行以下命令。

     ```
     sudo apt-get install -y python3.7-dev
     ```

   - 确保已成功安装 Setuptools 和 Picamera。

     ```
     sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
     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，请将 python3.7 而非 python3 用于本教程中的命令。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. 重启 Raspberry Pi。

```
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. 使用以下命令测试摄像机设置。

```
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。

步骤 2：安装 MXNet 框架

在此步骤中，在 Raspberry Pi 上安装 MXNet 库。

1. 远程登录您的 Raspberry Pi。

```
ssh pi@your-device-ip-address
```

2. 打开 MXNet 文档，再打开安装 MXNet，然后按照说明执行操作以在设备上安装 MXNet。

> **Note**
>
> 对于本教程，我们建议安装版本 1.5.0 并从源代码构建 MXNet，以避免设备冲突。

3. 安装 MXNet 后，请验证以下配置：
   - 确保 `ggc_user` 系统账户可以使用 MXNet 框架。

```
sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
```

   - 确保已安装 NumPy。

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

步骤 3：创建 MXNet 模型包

在此步骤中，创建包含要上传到 Amazon Simple Storage Service (Amazon S3) 的示例预训练 MXNet 模型的模型包。AWS IoT Greengrass 可以使用来自 Amazon S3 的模型包，前提是您使用的是 tar.gz 或 zip 格式。

1. 在您的计算机上，从the section called “机器学习示例”下载 Raspberry Pi 的 MXNet 示例。
2. 不要解压下载的 `mxnet-py3-armv7l.tar.gz` 文件。
3. 导航到 `squeezenet`目录。

   ```
   cd path-to-downloaded-sample/mxnet-py3-armv7l/models/squeezenet
   ```

   此目录中的 `squeezenet.zip` 文件是您的模型包。它包含图像分类模型的 SqueezeNet 开源模型构件。稍后，您可将此模型包上传到 Amazon S3。

步骤 4：创建并发布 Lambda 函数

在此步骤中，创建 Lambda 函数部署程序包和 Lambda 函数。接下来，发布函数版本并创建别名。

首先，创建 Lambda 函数部署程序包。

1. 在您的计算机上，导航到您在the section called “创建模型包”中提取的示例程序包中的 `examples` 目录。

   ```
   cd path-to-downloaded-sample/mxnet-py3-armv7l/examples
   ```

   `examples` 目录包含函数代码和依赖项。

   - `greengrassObjectClassification.py` 是本教程中使用的推理代码。可以将此代码用作模板来创建您自己的推理函数。
   - `greengrasssdk` 是适用于 Python 的 AWS IoT Greengrass 核心 SDK 的 1.5.0 版本。

   > **Note**
   >
   > 如果提供了新版本，您可以下载该版本并升级部署程序包中的开发工具包版本。有关更多信息，请参阅 GitHub 上的适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。

2. 将 `examples` 目录的内容压缩到一个名为 `greengrassObjectClassification.zip` 的文件中。这就是您的部署程序包。

```
zip -r greengrassObjectClassification.zip .
```

> **Note**
>
> 确保 `.py` 文件和依赖项位于该目录的根目录中。

下一步，创建 Lambda 函数。

3. 在 AWS IoT 控制台中，选择函数和创建函数。
4. 选择从头开始创作并使用以下值创建您的函数：

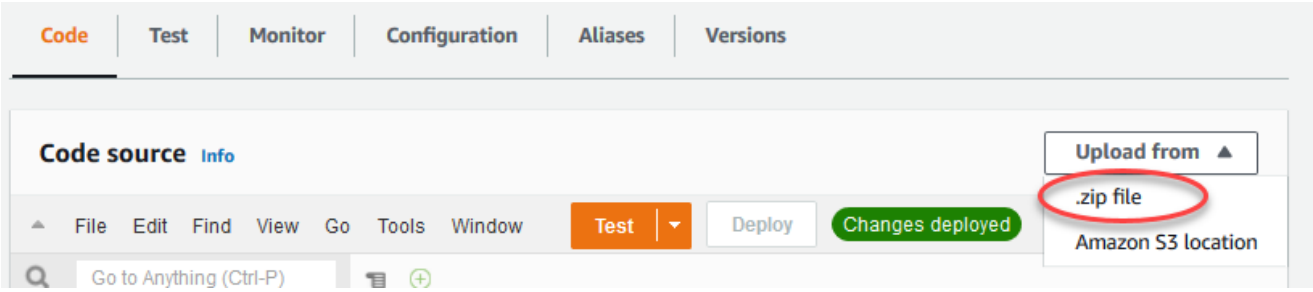
   - 对于 Function name（函数名称），请输入 **`greengrassObjectClassification`**。
   - 对于 Runtime (运行时)，选择 Python 3.7。

   对于权限，请保留默认设置。这将创建一个授予基本 Lambda 权限的执行角色。此角色未由 AWS IoT Greengrass 使用。
5. 选择 Create function（创建函数）。

现在，上传您的 Lambda 函数部署程序包并注册处理程序。

6. 选择您的 Lambda 函数并上传您的 Lambda 函数部署包。

   a. 在代码选项卡上的代码源下，选择上传自。从下拉列表中选择 .zip 文件。



b. 选择上传，然后选择您的 `greengrassObjectClassification.zip` 部署包。然后，选择 Save (保存)。

c. 在函数的代码选项卡中，在运行时设置下选择编辑，然后输入以下值。

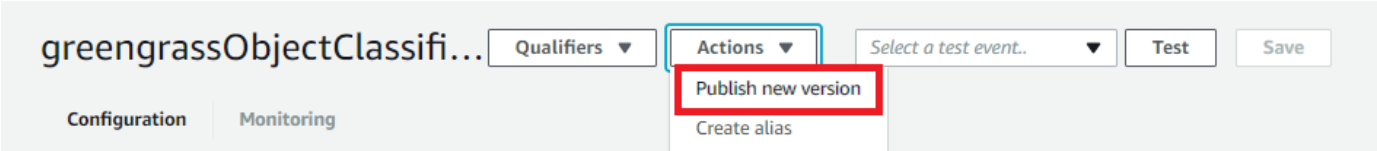
- 对于 Runtime (运行时)，选择 Python 3.7。
- 对于处理程序，输入 **`greengrassObjectClassification.function_handler`**。

选择 Save（保存）。

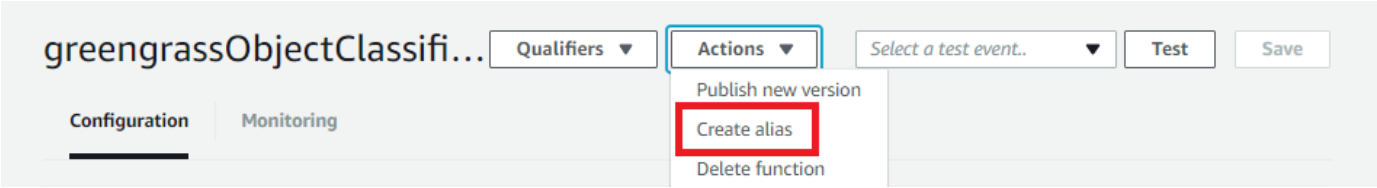
接下来，发布您的 Lambda 函数的第一个版本。然后，创建版本的别名。

> **Note**
>
> Greengrass 组可以按别名（推荐）或版本引用 Lambda 函数。使用别名，您可以更轻松地管理代码更新，因为您在更新函数代码时，不必更改订阅表或组定义。相反，您只需将别名指向新的函数版本。

7. 在 Actions 菜单上，选择 Publish new version。



8. 对于 Version description (版本描述)，输入 **`First version`**，然后选择 Publish (发布)。
9. 在 greengrassObjectClassification: 1 配置页面上，从操作菜单中选择创建别名。



10. 在创建新别名页面上，使用以下值：

- 对于 Name（名称），请输入 **`mlTest`**。
- 对于 Version (版本)，输入 **`1`**。

> **Note**
>
> AWS IoT Greengrass 不支持 $LATEST 版本的 Lambda 别名。

11. 选择 Save（保存）。

现在，将 Lambda 函数添加到 Greengrass 组。

步骤 5：将 Lambda 函数添加到 Greengrass 组

在该步骤中，将 Lambda 函数添加到该组，然后配置其生命周期和环境变量。

首先，将 Lambda 函数添加到 Greengrass 组。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 在组配置页面上，选择 Lambda 函数选项卡。
3. 在我的 Lambda 函数部分下，选择添加。
4. 对于 Lambda 函数，选择 greengrassObjectClassification。
5. 对于 Lambda 函数版本，请选择 Alias:mlTest。

接下来，配置 Lambda 函数的生命周期和环境变量。

6. 在 Lambda 函数配置部分中，进行以下更新。

> **Note**
>
> 我们建议您在不进行容器化的情况下运行 Lambda 函数，除非您的业务案例需要这样做。这有助于您在无需配置设备资源的前提下访问您的设备 GPU 和摄像头。如果您在没有容器化的情况下运行，则还必须为 AWS IoT Greengrass Lambda 函数授予根用户访问权限。

a. 在不进行容器化的情况下运行：

- 对于系统用户和群，请选择 **Another user ID/group ID**。在系统用户 ID 中，输入 **0**。在系统组 ID 中，输入 **0**。

  这将允许您的 Lambda 函数以根用户身份运行。有关以根用户身份运行的更多信息，请参阅 the section called “为组中的 Lambda 函数设置默认访问身份”。

  > **Tip**
  >
  > 您还必须更新 `config.json` 文件，从而能够为 Lambda 函数授予根用户访问权限。有关此步骤，请参阅 the section called “以根用户身份运行 Lambda 函数”。

- 对于 Lambda 函数容器化，请选择无容器。

  有关不进行容器化的情况下运行的更多信息，请参阅 the section called “选择 Lambda 函数容器化时的注意事项”。

- 对于 Timeout (超时)，输入 **10 seconds**。
- 对于已固定，选择 True。

  有关更多信息，请参阅the section called “生命周期配置”。

b. 改为在容器化模式下运行：

> **Note**
>
> 我们不建议以容器化模式运行，除非您的业务案例需要这样做。

- 对于系统用户和组，选择使用组默认值。
- 对于 Lambda 函数容器化，选择使用组默认值。
- 对于 Memory limit (内存限制)，输入 **96 MB**。
- 对于 Timeout (超时)，输入 **10 seconds**。
- 对于已固定，选择 True。

  有关更多信息，请参阅the section called “生命周期配置”。

7. 在 Environment variables (环境变量) 下，创建一个键/值对。键/值对是与 Raspberry Pi 上的 MXNet 模型进行交互的函数所必需的。

   对于键，使用 MXNET_ENGINE_TYPE。对于值，使用 NaiveEngine。

> **Note**
>
> 在您自己的用户定义的 Lambda 函数中，您可以选择在函数代码中设置环境变量。

8. 为所有其他属性保留默认值，然后选择添加 Lambda 函数。

步骤 6：将资源添加到 Greengrass 组

在该步骤中，将为摄像机模块和 ML 推理模型创建资源，并将该资源与 Lambda 函数关联。这使 Lambda 函数能够访问核心设备上的资源。

> **Note**
>
> 如果您在非容器化模式下运行，则无需配置这些设备资源，AWS IoT Greengrass 可直接访问您的设备 GPU 和摄像头。

首先，为摄像机创建两种本地设备资源：一种用于共享内存，另一种用于设备接口。有关本地资源访问的更多信息，请参阅使用 Lambda 函数和连接器访问本地资源。

1. 在组配置页面上，选择资源选项卡。
2. 在本地资源部分，选择添加本地资源。
3. 在 添加本地资源 页面上，使用以下值：

   - 对于 Resource name (资源名称)，输入 **`videoCoreSharedMemory`**。
   - 对于 Resource type (资源类型)，选择 Device (设备)。
   - 对于 本地设备路径，输入 **`/dev/vcsm`**。

     设备路径是设备资源的本地绝对路径。该路径只能引用 `/dev` 下的字符设备或块储存设备。
   - 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限。

     组所有者文件访问权限选项可让您授予对 Lambda 进程的额外的文件访问权限。有关更多信息，请参阅组所有者文件访问权限。
4. 接下来，您将为摄像机接口添加本地设备资源。
5. 选择添加本地资源。
6. 在 添加本地资源 页面上，使用以下值：

- 对于 Resource name (资源名称)，输入 **videoCoreInterface**。
- 对于 Resource type (资源类型)，选择 Device (设备)。
- 对于 本地设备路径，输入 **/dev/vchiq**。
- 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限。

7. 在页面底部，选择添加资源。

现在，将推理模型添加为机器学习资源。该步骤包含将 `squeezenet.zip` 模型包上传到 Amazon S3。

1. 在组的资源选项卡上，在机器学习部分，选择添加机器学习资源。
2. 在 添加机器学习资源 页面上，对于 资源名称，输入 **squeezenet_model**。
3. 对于模型来源，选择使用存储在 S3 中的模型，例如通过深度学习编译器优化的模型。
4. 对于 S3 URI，请输入 S3 桶的保存路径。
5. 选择 Browse S3（浏览 S3）。这会在 Amazon S3 控制台中打开一个新选项卡。
6. 在 Amazon S3 控制台选项卡上，将 `squeezenet.zip` 文件上传到 S3 桶。有关信息，请参阅 Amazon Simple Storage Service 用户指南中的如何将文件和文件夹上传到 S3 存储桶？。

   > **Note**
   >
   > 为了便于访问 S3 桶，桶名称必须包含字符串 **greengrass** 并且桶所在的区域必须与您用于 AWS IoT Greengrass 的区域相同。请选择唯一名称（如 **greengrass-bucket-*user-id*-*epoch-time***）。不要在存储桶名称中使用句点 (.)。

7. 在 AWS IoT Greengrass 控制台选项卡中，找到并选择 S3 存储桶。找到并上传 `squeezenet.zip` 文件，然后选择 Select (选择)。您可能需要选择刷新以更新可用存储桶和文件的列表。
8. 对于 Destination path (目的地路径)，输入 **/greengrass-machine-learning/mxnet/squeezenet**。

   这是 Lambda 运行时目标命名空间中本地模型的目标。当您部署该组时，AWS IoT Greengrass 将检索源模型包，然后将内容提取到指定的目录。本教程的示例 Lambda 函数已配置为使用该路径 (在 `model_path` 变量中)。
9. 在系统组所有者和文件访问权限下，选择不含系统组。

10. 选择 Add resource (添加资源)。

使用经过 SageMaker 训练的模型

本教程使用存储在 Amazon S3 中的模型，但您也可以轻松地使用 SageMaker 模型。AWS IoT Greengrass 控制台具有内置的 SageMaker 集成，因此您无需手动将这些模型上传到 Amazon S3。有关使用 SageMaker 模型的要求和限制，请参阅 the section called “支持的模型源”。

要使用 SageMaker 模型，请执行以下操作：

- 对于模型源，选择使用在 AWS SageMaker 中训练的模型，然后选择该模型的训练任务的名称。
- 对于本地路径，输入您的 Lambda 函数在其中查找该模型的目录的路径。

步骤 7：将订阅添加到 Greengrass 组

在该步骤中，将订阅添加到该组。此订阅使 Lambda 函数能够将预测结果发布到 MQTT 主题，从而将这些结果发送到 AWS IoT。

1. 在组配置页面中，选择订阅选项卡，然后选择添加订阅。
2. 在订阅详细信息页面中，按如下所述配置源和目标：
   a. 在源类型中，选择 Lambda 函数，然后选择 greengrassObjectClassification。
   b. 对于目标类型，选择服务，然后选择 IoT 云。
3. 在主题筛选条件字段中，输入 **`hello/world`**，然后选择订阅。

步骤 8：部署 Greengrass 组

在该步骤中，将最新版本的组定义部署到 Greengrass 核心设备。该定义包含您添加的 Lambda 函数、资源和订阅配置。

1. 确保 AWS IoT Greengrass 核心正在运行。根据需要在您的 Raspberry Pi 终端中运行以下命令。
   a. 要检查守护程序是否正在运行，请执行以下操作：

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

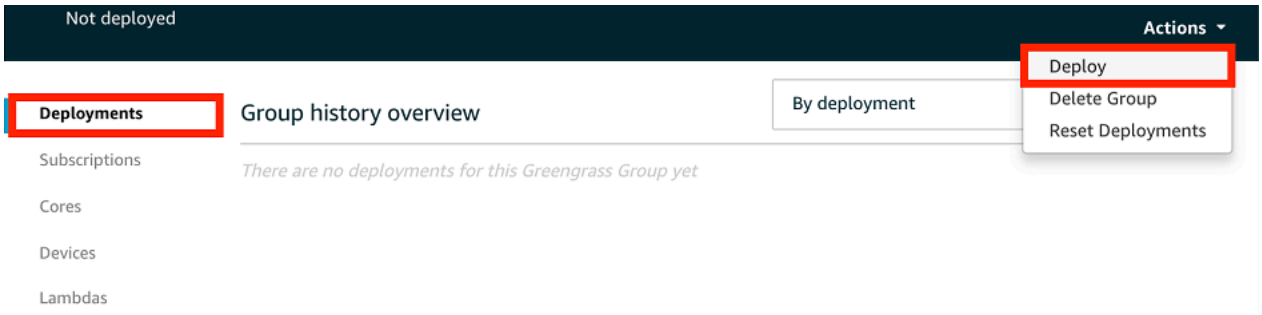
如果输出包含 root 的 /greengrass/ggc/packages/1.11.6/bin/daemon 条目，则表示守护程序正在运行。

> **Note**
>
> 路径中的版本取决于您的核心设备上安装的 AWS IoT Greengrass 核心软件版本。

b. 启动进程守护程序：

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

2. 在组配置页面上，选择部署。



3. 在 Lambda 函数选项卡的系统 Lambda 函数部分下，选择 IP 检测器，再选择编辑。
4. 在编辑 IP 检测器设置对话框中，选择自动检测和覆盖 MQTT 代理端点。
5. 选择 Save（保存）。

这使得设备可以自动获取核心的连接信息，例如 IP 地址、DNS 和端口号。建议使用自动检测，不过 AWS IoT Greengrass 也支持手动指定的终端节点。只有在首次部署组时，系统才会提示您选择发现方法。

> **Note**
>
> 如果出现提示，请授予权限，以创建 Greengrass 服务角色并将其关联至当前 AWS 区域中的 AWS 账户。该角色将允许 AWS IoT Greengrass 访问您的 AWS 服务资源。

Deployments (部署) 页面显示了部署时间戳、版本 ID 和状态。完成后，部署的状态应显示为 已完成。

有关部署的更多信息，请参阅 部署 AWS IoT Greengrass 组。有关问题排查帮助，请参阅排查问题。

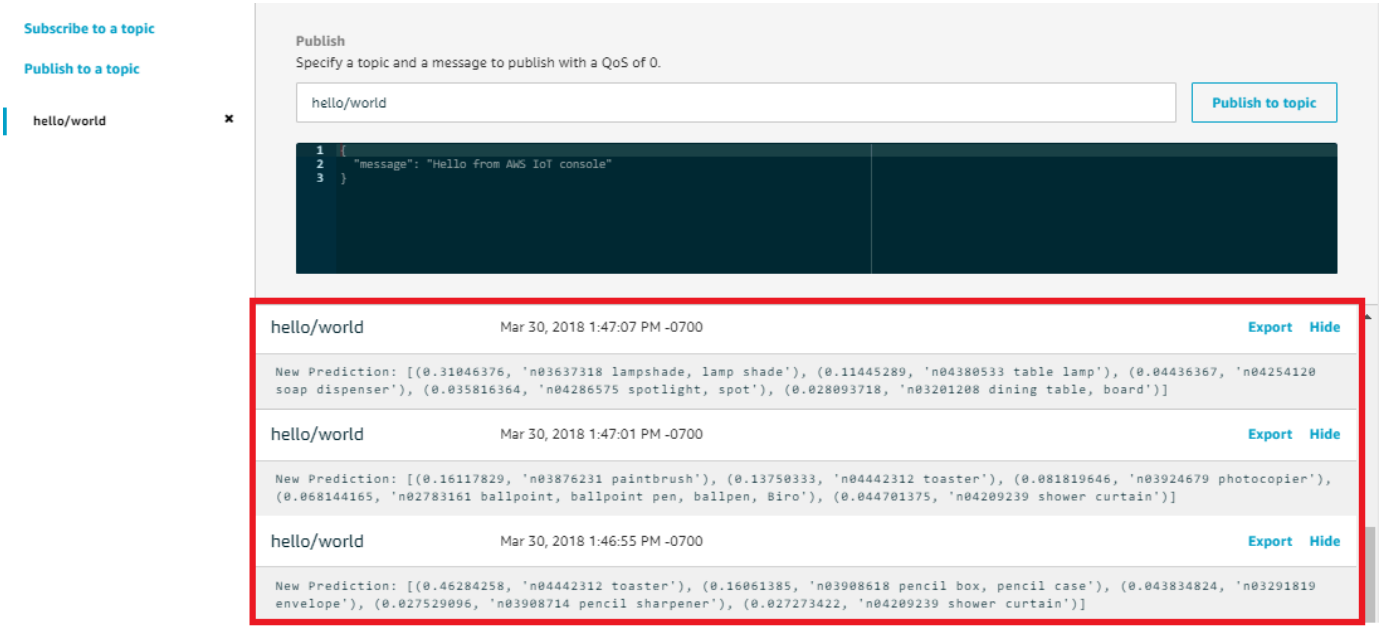
步骤 9：测试推理应用程序

现在，您可以验证是否正确配置了部署。要进行测试，您应订阅 `hello/world` 主题并查看由 Lambda 函数发布的预测结果。

> **Note**
>
> 如果将一个监视器连接到 Raspberry Pi，活动摄像机源将显示在预览窗口中。

1. 在 AWS IoT 控制台的测试下，选择 MQTT 测试客户端。
2. 对于 Subscriptions (订阅)，使用以下值：
   - 对于订阅主题，请使用 hello/world。
   - 在其他配置下，对于 MQTT 负载显示，请选择将负载显示为字符串。
3. 选择 Subscribe。

   如果测试成功，来自 Lambda 函数的消息将显示在页面底部。每条消息包含图像的前 5 个预测结果，这些结果使用以下格式：可能性、预测的类 ID 及对应的类名称。



AWS IoT Greengrass ML 推理问题排查

如果测试失败，您可以尝试以下问题排查步骤。在您的 Raspberry Pi 终端中运行以下命令。

检查错误日志

1. 切换到根用户并导航到 `log` 目录。访问 AWS IoT Greengrass 日志需要根权限。

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

2. 在 `system` 目录中，选中 `runtime.log` 或 `python_runtime.log`。

   在 `user/`*`region`*`/`*`account-id`* 目录中，选中 `greengrassObjectClassification.log`。

   有关更多信息，请参阅the section called “使用日志排查问题”。

runtime.log 中的“解压缩”错误

如果 `runtime.log` 包含类似于下面的错误，请确保您的 `tar.gz` 源模型包具有一个父目录。

```
Greengrass deployment error: unable to download the artifact model-arn: Error while
 processing.
Error while unpacking the file from /tmp/greengrass/artifacts/model-arn/path to /
greengrass/ggc/deployment/path/model-arn,
error: open /greengrass/ggc/deployment/path/model-arn/squeezenet/
squeezenet_v1.1-0000.params: no such file or directory
```

如果您的包没有包含模型文件的父目录，请使用以下命令重新打包模型：

```
tar -zcvf model.tar.gz ./model
```

例如：

```
#$ tar -zcvf test.tar.gz ./test
./test
./test/some.file
./test/some.file2
./test/some.file3
```

> **Note**
>
> 不要在此命令中包含尾随 /* 字符。

验证 Lambda 函数是否已成功部署

1. 列出 /lambda 目录中已部署的 Lambda 的内容。在运行命令前替换占位符值。

```
cd /greengrass/ggc/deployment/lambda/
arn:aws:lambda:region:account:function:function-name:function-version
ls -la
```

2. 验证该目录是否包含与您在 步骤 4：创建并发布 Lambda 函数 中上传的 greengrassObjectClassification.zip 部署程序包相同的内容。

   确保 .py 文件和依赖项位于该目录的根目录中。

验证推理模型是否已成功部署

1. 查找 Lambda 运行时进程的进程标识号 (PID)：

```
ps aux | grep 'lambda-function-name*'
```

   在输出中，PID 显示在 Lambda 运行时进程的行的第二列中。

2. 输入 Lambda 运行时命名空间。请确保在运行命令前替换占位符 *pid* 值。

> **Note**
>
> 此目录及其内容位于 Lambda 运行时命名空间中，因此它们不会显示在常规 Linux 命名空间中。

```
sudo nsenter -t pid -m /bin/bash
```

3. 列出您为 ML 资源指定的本地目录的内容。

```
cd /greengrass-machine-learning/mxnet/squeezenet/
ls -ls
```

您应该看到以下文件：

```
32 -rw-r--r-- 1 ggc_user ggc_group   31675 Nov 18 15:19 synset.txt
32 -rw-r--r-- 1 ggc_user ggc_group   28707 Nov 18 15:19 squeezenet_v1.1-symbol.json
4832 -rw-r--r-- 1 ggc_user ggc_group 4945062 Nov 18 15:19
 squeezenet_v1.1-0000.params
```

后续步骤

接下来，探索其他推理应用程序。AWS IoT Greengrass 提供了您可用于尝试执行本地推理的其他 Lambda 函数。您可以在您在the section called “安装 MXNet 框架”中下载的预编译库文件夹中查找示例程序包。

配置 Intel Atom

要在 Intel Atom 设备上运行本教程，您必须提供源图像、配置 Lambda 函数并添加其他本地设备资源。要使用 GPU 进行推理，请确保您的设备上已安装以下软件：

- OpenCL 版本 1.0 或更高版本
- Python 3.7 和 pip

> **Note**
>
> 如果您的设备是使用 Python 3.6 预构建的，则可改为创建指向 Python 3.7 的符号链接。有关更多信息，请参阅Step 2。

- NumPy
- 基于 Wheel 的 OpenCV

1. 下载 Lambda 函数的静态 PNG 或 JPG 图像以用于图像分类。该示例最适合小型图像文件。

将图像文件保存在包含 `greengrassObjectClassification.py` 文件的目录中（或保存在此目录的子目录中）。此目录位于您在 the section called “创建并发布 Lambda 函数” 中上传的 Lambda 函数部署程序包中。

> **Note**
>
> 如果您使用的是 AWS DeepLens，则可以使用板载摄像机或安装自己的摄像机来对捕获到的图像而非静态图像运行推理。但是，我们强烈建议您先用静态图像开始。
> 如果您使用的是摄像机，请确保 `awscam` APT 程序包已安装并且是最新的。有关更多信息，请参阅 AWS DeepLens 开发人员指南中的更新 AWS DeepLens 设备。

2. 如果您使用的不是 Python 3.7，请确保创建从 Python 3.x 到 Python 3.7 的符号链接。这会将您的设备配置为将 Python 3 与 AWS IoT Greengrass 结合使用。运行以下命令可查找您的 Python 安装：

```
which python3
```

运行以下命令可创建符号链接：

```
sudo ln -s path-to-python-3.x/python3.x path-to-python-3.7/python3.7
```

重新启动设备。

3. 编辑 Lambda 函数的配置。按照the section called “将 Lambda 函数添加到组”中的程序进行操作。

> **Note**
>
> 我们建议您在不进行容器化的情况下运行 Lambda 函数，除非您的业务案例需要这样做。这有助于您在无需配置设备资源的前提下访问您的设备 GPU 和摄像头。如果您在没有容器化的情况下运行，则还必须为 AWS IoT Greengrass Lambda 函数授予根用户访问权限。

a. 在不进行容器化的情况下运行：

   - 对于系统用户和群，请选择 **Another user ID/group ID**。在系统用户 ID 中，输入 **0**。在系统组 ID 中，输入 **0**。

这将允许您的 Lambda 函数以根用户身份运行。有关以根用户身份运行的更多信息，请参阅 the section called “为组中的 Lambda 函数设置默认访问身份”。

> **Tip**
>
> 您还必须更新 `config.json` 文件，从而能够为 Lambda 函数授予根用户访问权限。有关此步骤，请参阅 the section called “以根用户身份运行 Lambda 函数”。

- 对于 Lambda 函数容器化，请选择无容器。

  有关不进行容器化的情况下运行的更多信息，请参阅 the section called “选择 Lambda 函数容器化时的注意事项”。
- 将 Timeout (超时) 值更新为 5 秒。这可确保请求不会过早超时。设置后，需要几分钟以运行推理。
- 对于已固定，选择 True。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。
- 对于 Lambda 生命周期，选择 Make this function long-lived and keep it running indefinitely (使此函数长时间生存，保持其无限期运行)。

b. 改为在容器化模式下运行：

> **Note**
>
> 我们不建议以容器化模式运行，除非您的业务案例需要这样做。

- 将 Timeout (超时) 值更新为 5 秒。这可确保请求不会过早超时。设置后，需要几分钟以运行推理。
- 对于已固定，选择 True。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。

4. 如果在容器化模式下运行，请添加所需的本地设备资源以授予对设备 GPU 的访问权限。

> **Note**
>
> 如果您在非容器化模式下运行，则无需配置设备资源，AWS IoT Greengrass 可直接访问您的设备 GPU。

a. 在组配置页面上，选择资源选项卡。

b. 选择添加本地资源。

c. 定义资源：

- 对于 Resource name (资源名称)，输入 **renderD128**。
- 对于 资源类型，选择 本地设备。
- 对于 Device path (设备路径)，输入 **/dev/dri/renderD128**。
- 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限。
- 对于 Lambda 函数从属关系，为您的 Lambda 函数授予读写访问权限。

配置 NVIDIA Jetson TX2

要在 NVIDIA Jetson TX2 上运行本教程，您需要提供源图像并配置 Lambda 函数。如果您使用的是 GPU，则还必须添加本地设备资源。

1. 确保已配置您的 Jetson 设备，以便安装 AWS IoT Greengrass Core 软件。有关配置设备的更多信息，请参阅the section called “设置其他设备”。
2. 打开 MXNet 文档，转至在 Jetson 上安装 MXNet，然后按照说明执行操作以在 Jetson 设备上安装 MXNet。

   > **Note**
   >
   > 如果要从源构建 MXNet，请按照说明执行操作来构建共享库。编辑 `config.mk` 文件中的以下设置以使用 Jetson TX2 设备：
   >
   > - 将 `-gencode arch=compute-62, code=sm_62` 添加到 `CUDA_ARCH` 设置。
   > - 打开 CUDA。
   >
   > ```
   > USE_CUDA = 1
   > ```

3. 下载 Lambda 函数的静态 PNG 或 JPG 图像以用于图像分类。该应用最适合小型图像文件。或者，您可以分析 Jetson 板上的摄像机来捕获源图像。

将图像文件保存在包含 `greengrassObjectClassification.py` 文件的目录中。您也可以将这些文件保存在此目录的子目录中。此目录位于您在 the section called “创建并发布 Lambda 函数” 中上传的 Lambda 函数部署程序包中。

4. 创建从 Python 3.7 到 Python 3.6 的符号链接以将 Python 3 与 AWS IoT Greengrass 结合使用。运行以下命令可查找您的 Python 安装：

```
which python3
```

运行以下命令可创建符号链接：

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

重新启动设备。

5. 确保 `ggc_user` 系统账户可以使用 MXNet 框架：

```
“sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
```

6. 编辑 Lambda 函数的配置。按照the section called “将 Lambda 函数添加到组”中的程序进行操作。

> **Note**
>
> 我们建议您在不进行容器化的情况下运行 Lambda 函数，除非您的业务案例需要这样做。这有助于您在无需配置设备资源的前提下访问您的设备 GPU 和摄像头。如果您在没有容器化的情况下运行，则还必须为 AWS IoT Greengrass Lambda 函数授予根用户访问权限。

a. 在不进行容器化的情况下运行：

- 对于系统用户和群，请选择 **Another user ID/group ID**。在系统用户 ID 中，输入 **0**。在系统组 ID 中，输入 **0**。

  这将允许您的 Lambda 函数以根用户身份运行。有关以根用户身份运行的更多信息，请参阅 the section called “为组中的 Lambda 函数设置默认访问身份”。

> **Tip**
>
> 您还必须更新 `config.json` 文件，从而能够为 Lambda 函数授予根用户访问权限。有关此步骤，请参阅 the section called “以根用户身份运行 Lambda 函数”。

- 对于 Lambda 函数容器化，请选择无容器。

  有关不进行容器化的情况下运行的更多信息，请参阅 the section called “选择 Lambda 函数容器化时的注意事项”。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。
- 在 环境变量下，将以下键/值对添加到 Lambda 函数。这会将 AWS IoT Greengrass 配置为使用 MXNet 框架。

键	Value
路径	/usr/local/cuda/bin:$PATH
MXNET_HOME	$HOME/mxnet/
PYTHONPATH	$MXNET_HOME/python:$PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

b. 改为在容器化模式下运行：

> **Note**
>
> 我们不建议以容器化模式运行，除非您的业务案例需要这样做。

- 提高内存限制值。使用 500 MB（对于 CPU）或至少使用 2000 MB（对于 GPU）。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。
- 在 环境变量下，将以下键/值对添加到 Lambda 函数。这会将 AWS IoT Greengrass 配置为使用 MXNet 框架。

键	Value
路径	/usr/local/cuda/bin:$PATH
MXNET_HOME	$HOME/mxnet/
PYTHONPATH	$MXNET_HOME/python:$PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

7. 如果在容器化模式下运行，请添加以下本地设备资源以授予对设备 GPU 的访问权限。按照the section called “将资源添加到组”中的程序进行操作。

> **Note**
>
> 如果您在非容器化模式下运行，则无需配置设备资源，AWS IoT Greengrass 可直接访问您的设备 GPU。

对于每个资源：

- 对于 Resource type (资源类型)，选择 Device (设备)。
- 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限。

名称	设备路径
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/nvhost-dbg-gpu

名称	设备路径
nvhost-prof-gpu	/dev/nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

8. 如果在容器化模式下运行，请添加以下本地卷资源以授予对设备摄像头的访问权限。按照the section called “将资源添加到组”中的程序进行操作。

> **Note**
>
> 如果您在非容器化模式下运行，则无需配置卷资源，AWS IoT Greengrass 可直接访问您的设备摄像头。

- 对于 Resource type (资源类型)，选择 Volume (卷)。
- 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限。

名称	源路径	目的地路径
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

如何使用 AWS Management Console配置优化的机器学习推理

要按照本教程中的步骤操作，您必须使用 AWS IoT Greengrass Core v1.10 或更高版本。

您可以使用 SageMaker Neo 深度学习编译器优化本机机器学习推理模型在 Tensorflow、Apache MXNet、PyTorch、ONNX 和 XGBoost 框架中的预测效率，以实现更少的内存占用量和更快的性能。然后可以下载优化的模型，安装 SageMaker Neo 深度学习运行时，并将其部署到您的 AWS IoT Greengrass 设备，以实现更快的推理。

本教程介绍如何使用 AWS Management Console 将 Greengrass 组配置为运行一个 Lambda 推理示例，该示例可在本地识别摄像机中的图像，而无需将数据发送至云。该推理示例访问 Raspberry Pi 上的摄像机模块。在本教程中，您将下载由 Resnet-50 训练并在 Neo 深度学习编译器中优化的预先打包的模型。然后使用该模型在您的 AWS IoT Greengrass 设备上执行本地图像分类。

本教程包含以下概括步骤：

1. 配置 Raspberry Pi
2. 安装 Neo 深度学习运行时
3. 创建推理 Lambda 函数。
4. 将 Lambda 函数添加到组
5. 将 Neo 优化的模型资源添加到组中
6. 将摄像机设备资源添加到组
7. 将订阅添加到组
8. 部署组
9. 测试示例

先决条件

要完成此教程，需要：

- Raspberry Pi 4 Model B 或 Raspberry Pi 3 Model B/B+，已设置并配置为与 AWS IoT Greengrass 结合使用。要使用 AWS IoT Greengrass 设置 Raspberry Pi，请运行 Greengrass 设备安装脚本，或确保您已完成 入门 AWS IoT Greengrass 的模块 1 和模块 2。

  > **Note**
  >
  > Raspberry Pi 可能需要一个 2.5A 的电源来运行通常用于图像分类的深度学习框架。额定值不足的电源可能会导致设备重新启动。

- Raspberry Pi 摄像机模块 V2 - 800 万像素，1080p。要了解如何设置摄像机，请参阅 Raspberry Pi 文档中的连接摄像机。
- Greengrass 组和 Greengrass 核心。要了解如何创建 Greengrass 组或核心，请参阅入门 AWS IoT Greengrass。

> **Note**
>
> 本教程使用的是 Raspberry Pi，但 AWS IoT Greengrass 支持其他平台，例如 Intel Atom 和 NVIDIA Jetson TX2。如果使用的是 Intel Atom 示例，您可能需要安装 Python 3.6 而不是 Python 3.7。有关配置设备以便安装 AWS IoT Greengrass Core 软件的信息，请参阅the section called “设置其他设备”。
>
> 对于 AWS IoT Greengrass 不支持的第三方平台，您必须在非容器化模式下运行 Lambda 函数。要在非容器化模式下运行，必须以 root 用户身份运行 Lambda 函数。有关更多信息，请参阅 the section called “选择 Lambda 函数容器化时的注意事项” 和 the section called “为组中的 Lambda 函数设置默认访问身份”：

步骤 1：配置 Raspberry Pi

在该步骤中，将安装 Raspbian 操作系统的更新，安装摄像机模块软件和 Python 依赖项，以及启用摄像机接口。

在您的 Raspberry Pi 终端中运行以下命令。

1. 安装 Raspbian 的更新。

   ```
   sudo apt-get update
   sudo apt-get dist-upgrade
   ```

2. 安装适用于摄像机模块的 `picamera` 接口以及本教程所需的其他 Python 库。

   ```
   sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
   ```

   验证安装：

   - 确保您的 Python 3.7 安装包含 pip。

     ```
     python3 -m pip
     ```

     如果未安装 pip，请从 pip 网站 下载它，然后运行以下命令。

     ```
     python3 get-pip.py
     ```

   - 确保您的 Python 版本为 3.7 或更高版本。

```
python3 --version
```

如果输出列出了早期版本，请运行以下命令。

```
sudo apt-get install -y python3.7-dev
```

- 确保已成功安装 Setuptools 和 Picamera。

```
sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
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`，请将 `python3.7` 而非 `python3` 用于本教程中的命令。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. 重启 Raspberry Pi。

```
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. 使用以下命令测试摄像机设置。

```
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。

步骤 2：安装 Amazon SageMaker Neo 深度学习运行时

在该步骤中，在 Raspberry Pi 上安装 Neo 深度学习运行时 (DLR)。

> **Note**
>
> 对于本教程，我们建议安装 1.1.0 版。

1. 远程登录您的 Raspberry Pi。

   ```
   ssh pi@your-device-ip-address
   ```

2. 打开 DLR 文档，打开安装 DLR，然后找到 Raspberry Pi 设备的 wheel URL。然后，按照说明执行操作，在您的设备上安装 DLR。例如，您可以使用 pip：

   ```
   pip3 install rasp3b-wheel-url
   ```

3. 安装 DLR 后，请验证以下配置：

   - 确保 `ggc_user` 系统账户可以使用 DLR 库。

     ```
     sudo -u ggc_user bash -c 'python3 -c "import dlr"'
     ```

   - 确保已安装 NumPy。

     ```
     sudo -u ggc_user bash -c 'python3 -c "import numpy"'
     ```

步骤 3：创建推理 Lambda 函数

在此步骤中，创建 Lambda 函数部署程序包和 Lambda 函数。接下来，发布函数版本并创建别名。

1. 在您的计算机上，从the section called “机器学习示例”下载 Raspberry Pi 的 DLR 示例。
2. 不要解压下载的 `dlr-py3-armv7l.tar.gz` 文件。

   ```
   cd path-to-downloaded-sample
   tar -xvzf dlr-py3-armv7l.tar.gz
   ```

   提取的示例程序包中的 `examples` 目录包含函数代码和依赖项。

- `inference.py` 是本教程中使用的推理代码。可以将此代码用作模板来创建您自己的推理函数。
- `greengrasssdk` 是适用于 Python 的 AWS IoT Greengrass 核心 SDK 的 1.5.0 版本。

> **Note**
>
> 如果提供了新版本，您可以下载该版本并升级部署程序包中的开发工具包版本。有关更多信息，请参阅 GitHub 上的适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。

3. 将 `examples` 目录的内容压缩到一个名为 `optimizedImageClassification.zip` 的文件中。这就是您的部署程序包。

```
cd path-to-downloaded-sample/dlr-py3-armv7l/examples
zip -r optimizedImageClassification.zip .
```

部署程序包包含您的函数代码和依赖项。其中包括的代码将调用 Neo 深度学习运行时 Python API 以使用 Neo 深度学习编译器模型执行推理。

> **Note**
>
> 确保 `.py` 文件和依赖项位于该目录的根目录中。

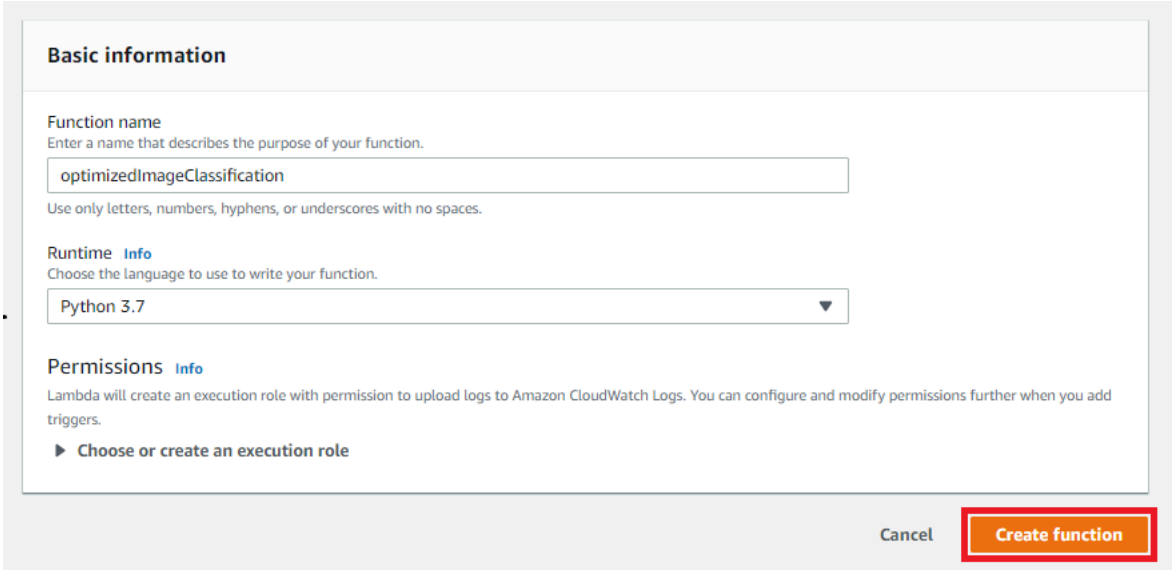
4. 现在，将 Lambda 函数添加到 Greengrass 组。

从 Lambda 控制台页面，选择函数，然后选择创建函数。

5. 选择从头开始创作并使用以下值创建您的函数：

- 对于 Function name（函数名称），请输入 **`optimizedImageClassification`**。
- 对于 Runtime (运行时)，选择 Python 3.7。

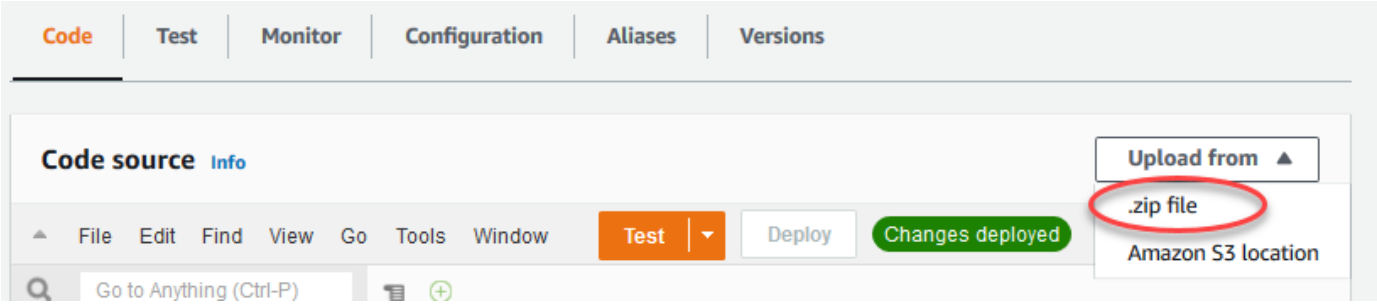
对于权限，请保留默认设置。这将创建一个授予基本 Lambda 权限的执行角色。此角色未由 AWS IoT Greengrass 使用。



6. 选择 Create function（创建函数）。

现在，上传您的 Lambda 函数部署程序包并注册处理程序。

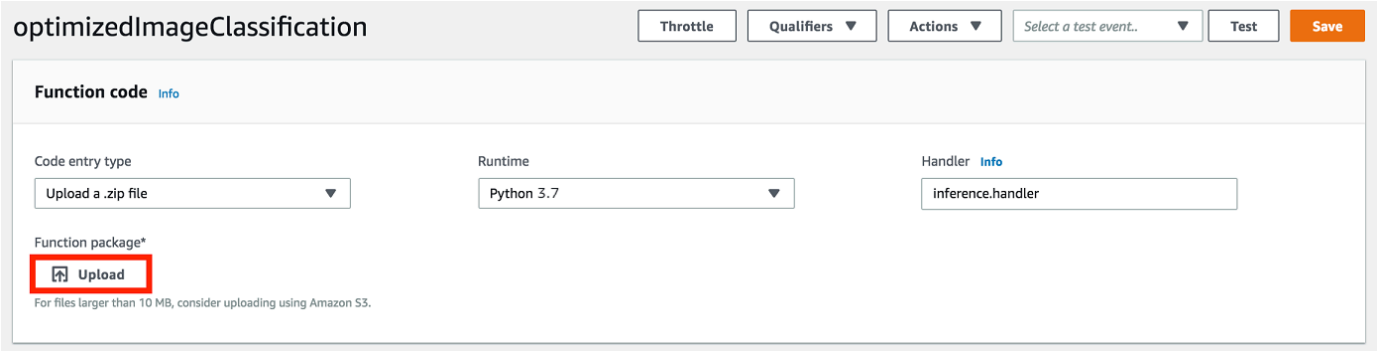
1. 在代码选项卡上的代码源下，选择上传自。从下拉列表中选择 .zip 文件。



2. 选择您的 `optimizedImageClassification.zip` 部署包，然后选择保存。
3. 在函数的代码选项卡中，在运行时设置下选择编辑，然后输入以下值。

   - 对于 Runtime (运行时)，选择 Python 3.7。
   - 对于处理程序，输入 **`inference.handler`**。

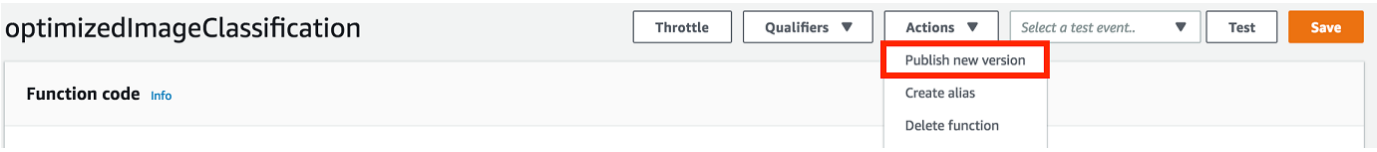
   选择 Save（保存）。



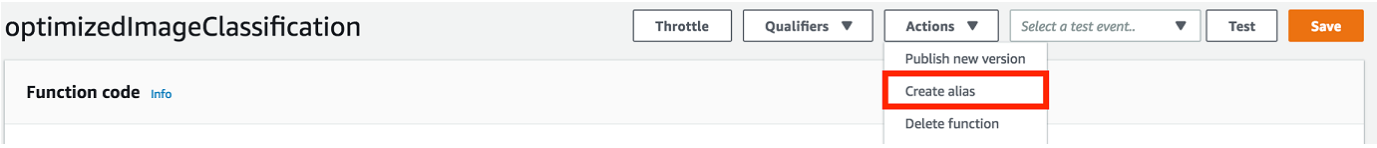
接下来，发布您的 Lambda 函数的第一个版本。然后，创建版本的别名。

> **Note**
>
> Greengrass 组可以按别名（推荐）或版本引用 Lambda 函数。使用别名，您可以更轻松地管理代码更新，因为您在更新函数代码时，不必更改订阅表或组定义。相反，您只需将别名指向新的函数版本。

1. 在 Actions 菜单上，选择 Publish new version。



2. 对于 Version description (版本描述)，输入 **`First version`**，然后选择 Publish (发布)。
3. 在 optimizedImageClassification: 1 配置页面上，从 Actions (操作) 菜单中选择 Create alias (创建别名)。



4. 在创建新别名页面上，使用以下值：
   - 对于 Name（名称），请输入 **`mlTestOpt`**。
   - 对于 Version (版本)，输入 **`1`**。

> **Note**
>
> AWS IoT Greengrass 不支持 $LATEST 版本的 Lambda 别名。

5. 选择 Create（创建）。

   现在，将 Lambda 函数添加到 Greengrass 组。

步骤 4：将 Lambda 函数添加到 Greengrass 组

在此步骤中，将 Lambda 函数添加到该组，然后配置其生命周期。

首先，将 Lambda 函数添加到 Greengrass 组。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 在组配置页面上，选择 Lambda 函数选项卡，然后选择添加。
3. 选择 Lambda 函数 并选择 optimizedImageClassification。
4. 在 Lambda 函数版本上，请选择您所发布的版本的别名。

接下来，配置 Lambda 函数的生命周期。

1. 在 Lambda 函数配置部分中，进行以下更新。

   > **Note**
   >
   > 我们建议您在不进行容器化的情况下运行 Lambda 函数，除非您的业务案例需要这样做。这有助于您在无需配置设备资源的前提下访问您的设备 GPU 和摄像头。如果您在没有容器化的情况下运行，则还必须为 AWS IoT Greengrass Lambda 函数授予根用户访问权限。

   a. 在不进行容器化的情况下运行：

      - 对于系统用户和群，请选择 **Another user ID/group ID**。在系统用户 ID 中，输入 **0**。在系统组 ID 中，输入 **0**。

这将允许您的 Lambda 函数以根用户身份运行。有关以根用户身份运行的更多信息，请参阅 the section called “为组中的 Lambda 函数设置默认访问身份”。

> **Tip**
>
> 您还必须更新 `config.json` 文件，从而能够为 Lambda 函数授予根用户访问权限。有关此步骤，请参阅 the section called “以根用户身份运行 Lambda 函数”。

- 对于 Lambda 函数容器化，请选择无容器。

  有关不进行容器化的情况下运行的更多信息，请参阅 the section called “选择 Lambda 函数容器化时的注意事项”。
- 对于 Timeout (超时)，输入 **`10 seconds`**。
- 对于已固定，选择 True。

  有关更多信息，请参阅the section called “生命周期配置”。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。

b. 改为在容器化模式下运行：

> **Note**
>
> 我们不建议以容器化模式运行，除非您的业务案例需要这样做。

- 对于系统用户和组，选择使用组默认值。
- 对于 Lambda 函数容器化，选择使用组默认值。
- 对于 Memory limit (内存限制)，输入 **`1024 MB`**。
- 对于 Timeout (超时)，输入 **`10 seconds`**。
- 对于已固定，选择 True。

  有关更多信息，请参阅the section called “生命周期配置”。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。

2. 选择 添加 Lambda 函数。

步骤 5：将 SageMaker Neo 优化的模型资源添加到 Greengrass 组中

在此步骤中，将为优化的 ML 推理模型创建一个资源并将其上传到 Amazon S3 存储桶。然后，在 AWS IoT Greengrass 控制台中找到 Amazon S3 上传的模型，并将新创建的资源与 Lambda 函数关联。这使该函数能够访问其在核心设备上的资源。

1. 在您的计算机上，导航到您在the section called “创建推理 Lambda 函数。”中提取的示例程序包中的 `resnet50` 目录。

   > **Note**
   >
   > 如果使用的是 NVIDIA Jetson 示例，则需要改用示例程序包中的 `resnet18` 目录。有关更多信息，请参阅the section called “配置 NVIDIA Jetson TX2”。

   ```
   cd path-to-downloaded-sample/dlr-py3-armv7l/models/resnet50
   ```

   该目录包含使用 Resnet-50 训练的图像分类模型的预编译模型项目。

2. 将 `resnet50` 目录中的文件压缩到一个名为 `resnet50.zip` 的文件中。

   ```
   zip -r resnet50.zip .
   ```

3. 在您的 AWS IoT Greengrass 组的组配置页面上，选择 资源选项卡。导航到 Machine Learning (机器学习) 部分，然后选择 Add machine learning resource (添加机器学习资源)。在 Create a machine learning resource (创建机器学习资源) 页面上，对于 Resource name (资源名称)，输入 **`resnet50_model`**。
4. 对于模型来源，选择使用存储在 S3 中的模型，例如通过深度学习编译器优化的模型。
5. 在 S3 URI 下，选择浏览 S3。

   > **Note**
   >
   > 当前，优化的 SageMaker 模型会自动存储在 Amazon S3 中。您可以使用此选项在 Amazon S3 存储桶中找到您的优化模型。有关 SageMaker 中模型优化的更多信息，请参阅 SageMaker Neo 文档。

6. 选择上传模型。

7. 在 Amazon S3 控制台选项卡中，将 zip 文件上传到 Amazon S3 存储桶。有关信息，请参阅 Amazon Simple Storage Service 用户指南中的如何将文件和文件夹上传到 S3 存储桶？。

   > **Note**
   >
   > 您的存储桶名称必须包含字符串 **greengrass**。请选择唯一名称（如 **greengrass-dlr-bucket-*user-id*-*epoch-time***）。不要在存储桶名称中使用句点 (.)。

8. 在 AWS IoT Greengrass 控制台选项卡中，找到并选择 Amazon S3 存储桶。找到并上传 `resnet50.zip` 文件，然后选择 Select (选择)。您可能需要刷新页面以更新可用存储桶和文件的列表。

9. 对于 目的地路径，输入 **/ml_model**。

   这是 Lambda 运行时目标命名空间中本地模型的目标。当您部署该组时，AWS IoT Greengrass 将检索源模型包，然后将内容提取到指定的目录。

   > **Note**
   >
   > 强烈建议使用为您的本地路径提供的准确路径。如果在该步骤中使用不同的本地模型目标路径，则会导致本教程中提供的一些问题排查命令不准确。如果使用不同的路径，则必须设置一个使用此处提供的准确路径的 `MODEL_PATH` 环境变量。有关环境变量的信息，请参阅 AWS Lambda 环境变量。

10. 如果在容器化模式下运行：

    a. 在系统组所有者和文件访问权限下，选择指定系统组和权限。

    b. 选择 只读访问权限，然后选择 添加资源。

步骤 6：将摄像机设备资源添加到 Greengrass 组

在此步骤中，为摄像机模块创建一个资源，并将该资源与 Lambda 函数关联。这使 Lambda 函数能够访问核心设备上的资源。

> **Note**
>
> 如果您在非容器化模式下运行，则无需配置此设备资源，AWS IoT Greengrass 可直接访问您的设备 GPU 和摄像头。

1. 在组配置页面上，选择资源选项卡。
2. 在本地资源选项卡上，选择添加本地资源。
3. 在 添加本地资源 页面上，使用以下值：
   - 对于 Resource name (资源名称)，输入 **`videoCoreSharedMemory`**。
   - 对于 Resource type (资源类型)，选择 Device (设备)。
   - 对于 本地设备路径，输入 **`/dev/vcsm`**。

     设备路径是设备资源的本地绝对路径。该路径只能引用 `/dev` 下的字符设备或块储存设备。
   - 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限。

     组所有者文件访问权限选项可让您授予对 Lambda 进程的额外的文件访问权限。有关更多信息，请参阅组所有者文件访问权限。
4. 在页面底部，选择添加资源。
5. 在资源选项卡中，选择添加并使用以下值创建另一个本地资源：
   - 对于 Resource name (资源名称)，输入 **`videoCoreInterface`**。
   - 对于 Resource type (资源类型)，选择 Device (设备)。
   - 对于 本地设备路径，输入 **`/dev/vchiq`**。
   - 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限。
6. 选择 Add resource (添加资源)。

步骤 7：将订阅添加到 Greengrass 组

在该步骤中，将订阅添加到组。这些订阅使 Lambda 函数能够将预测结果发布到 MQTT 主题，从而将这些结果发送到 AWS IoT。

1. 在组配置页面中，选择订阅选项卡，然后选择添加订阅。
2. 在创建订阅页面中，按如下所述配置源和目标：

a. 在 源类型中，选择 Lambda 函数，然后选择 optimizedImageClassification。

b. 对于目标类型，选择服务，然后选择 IoT 云。

c. 在主题筛选条件字段中，输入 **/resnet-50/predictions**，然后选择订阅。

3. 添加另一个订阅。选择订阅选项卡，选择添加订阅，然后按如下所示配置源和目标：

a. 在源类型中，选择服务，然后选择 IoT 云。

b. 在 目标类型中，选择 Lambda 函数，然后选择 optimizedImageClassification。

c. 在主题筛选条件字段中，输入 **/resnet-50/test**，然后选择订阅。

步骤 8：部署 Greengrass 组

在该步骤中，将最新版本的组定义部署到 Greengrass 核心设备。该定义包含您添加的 Lambda 函数、资源和订阅配置。

1. 确保 AWS IoT Greengrass 核心正在运行。根据需要在您的 Raspberry Pi 终端中运行以下命令。

a. 要检查守护程序是否正在运行，请执行以下操作：

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

如果输出包含 root 的 /greengrass/ggc/packages/*latest-core-version*/bin/daemon 条目，则表示守护程序正在运行。

b. 启动进程守护程序：

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

2. 在组配置页面上，选择部署。

3. 在 Lambda 函数选项卡上，选择 IP 检测器，然后选择编辑。

4. 在编辑 IP 检测器设置对话框中，选择自动检测和覆盖 MQTT 代理端点，然后选择保存。

这使得设备可以自动获取核心的连接信息，例如 IP 地址、DNS 和端口号。建议使用自动检测，不过 AWS IoT Greengrass 也支持手动指定的终端节点。只有在首次部署组时，系统才会提示您选择发现方法。

> **Note**
>
> 如果出现提示，请授予权限，以创建 Greengrass 服务角色并将其关联至当前 AWS 区域中的 AWS 账户。该角色将允许 AWS IoT Greengrass 访问您的 AWS 服务资源。

Deployments (部署) 页面显示了部署时间戳、版本 ID 和状态。完成后，部署的状态应显示为 已完成。

有关部署的更多信息，请参阅 部署 AWS IoT Greengrass 组。有关问题排查帮助，请参阅排查问题。

测试推理示例

现在，您可以验证是否正确配置了部署。要进行测试，您需要订阅 `/resnet-50/predictions` 主题，并向 `/resnet-50/test` 主题发布任何消息。这会触发 Lambda 函数使用您的 Raspberry Pi 拍照，并对捕获的图像执行推理。

> **Note**
>
> 如果使用的是 NVIDIA Jetson 示例，请确保改用 `resnet-18/predictions` 和 `resnet-18/test` 主题。

> **Note**
>
> 如果将一个监视器连接到 Raspberry Pi，活动摄像机源将显示在预览窗口中。

1. 在 AWS IoT 控制台主页的测试下，选择 MQTT 测试客户端。
2. 对于订阅，选择订阅主题。使用以下值。将剩余选项保留为默认值。
   - 对于 Subscription topic (订阅主题)，输入 **`/resnet-50/predictions`**。
   - 在其他配置下，对于 MQTT 负载显示，请选择将负载显示为字符串。
3. 选择 Subscribe。
4. 选择发布主题，输入 **`/resnet-50/test`** 作为主题名称，然后选择发布。

5. 如果测试成功，发布的消息会触发 Raspberry Pi 摄像机捕获图像。来自 Lambda 函数的消息将显示在页面底部。该消息包含图像的预测结果，采用的格式为：预测的类名称、可能性和峰值内存使用率。

配置 Intel Atom

要在 Intel Atom 设备上运行本教程，您必须提供源图像、配置 Lambda 函数并添加其他本地设备资源。要使用 GPU 进行推理，请确保您的设备上已安装以下软件：

- OpenCL 版本 1.0 或更高版本
- Python 3.7 和 pip
- NumPy
- 基于 Wheel 的 OpenCV

1. 下载 Lambda 函数的静态 PNG 或 JPG 图像以用于图像分类。该示例最适合小型图像文件。

   将图像文件保存在包含 `inference.py` 文件的目录中（或保存在此目录的子目录中）。此目录位于您在 the section called “创建推理 Lambda 函数。” 中上传的 Lambda 函数部署程序包中。

   > **Note**
   >
   > 如果您使用的是 AWS DeepLens，则可以使用板载摄像机或安装自己的摄像机来对捕获到的图像而非静态图像运行推理。但是，我们强烈建议您先用静态图像开始。
   > 如果您使用的是摄像机，请确保 `awscam` APT 程序包已安装并且是最新的。有关更多信息，请参阅 AWS DeepLens 开发人员指南中的更新 AWS DeepLens 设备。

2. 编辑 Lambda 函数的配置。按照the section called “将 Lambda 函数添加到组”中的程序进行操作。

   > **Note**
   >
   > 我们建议您在不进行容器化的情况下运行 Lambda 函数，除非您的业务案例需要这样做。这有助于您在无需配置设备资源的前提下访问您的设备 GPU 和摄像头。如果您在没有容器化的情况下运行，则还必须为 AWS IoT Greengrass Lambda 函数授予根用户访问权限。

   a. 在不进行容器化的情况下运行：

- 对于系统用户和群，请选择 **Another user ID/group ID**。在系统用户 ID 中，输入 **0**。在系统组 ID 中，输入 **0**。

  这将允许您的 Lambda 函数以根用户身份运行。有关以根用户身份运行的更多信息，请参阅 the section called “为组中的 Lambda 函数设置默认访问身份”。

  > **Tip**
  >
  > 您还必须更新 `config.json` 文件，从而能够为 Lambda 函数授予根用户访问权限。有关此步骤，请参阅 the section called “以根用户身份运行 Lambda 函数”。

- 对于 Lambda 函数容器化，请选择无容器。

  有关不进行容器化的情况下运行的更多信息，请参阅 the section called “选择 Lambda 函数容器化时的注意事项”。

- 将超时值增加到 2 分钟。这可确保请求不会过早超时。设置后，需要几分钟以运行推理。
- 对于已固定，选择 True。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。

b. 改为在容器化模式下运行：

> **Note**
>
> 我们不建议以容器化模式运行，除非您的业务案例需要这样做。

- 将内存限制值提高到 3000 MB。
- 将超时值增加到 2 分钟。这可确保请求不会过早超时。设置后，需要几分钟以运行推理。
- 对于已固定，选择 True。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。

3. 将 Neo 优化的模型资源添加到组中。上传您在the section called “创建推理 Lambda 函数。”中提取的示例程序包的 `resnet50` 目录中的模型资源。该目录包含使用 Resnet-50 训练的图像分类模型的预编译模型项目。按照the section called “将 Neo 优化的模型资源添加到组中”中的过程操作，并进行以下更新。

   - 将 `resnet50` 目录中的文件压缩到一个名为 `resnet50.zip` 的文件中。

- 在 Create a machine learning resource (创建机器学习资源) 页面上，对于 Resource name (资源名称)，输入 **`resnet50_model`**。
- 上传 `resnet50.zip` 文件。

4. 如果在容器化模式下运行，请添加所需的本地设备资源以授予对设备 GPU 的访问权限。

> **Note**
>
> 如果您在非容器化模式下运行，则无需配置设备资源，AWS IoT Greengrass 可直接访问您的设备 GPU。

   a. 在组配置页面上，选择资源选项卡。
   b. 在本地资源部分，选择添加本地资源。
   c. 定义资源：
      - 对于 Resource name (资源名称)，输入 **`renderD128`**。
      - 对于 Resource type (资源类型)，选择 Device (设备)。
      - 对于 本地设备路径，输入 **`/dev/dri/renderD128`**。
      - 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限。

配置 NVIDIA Jetson TX2

要在 NVIDIA Jetson TX2 上运行本教程，请提供源图像、配置 Lambda 函数并添加更多本地设备资源。

1. 确保已配置您的 Jetson 设备，以便安装 AWS IoT Greengrass Core 软件并使用 GPU 进行推理。有关配置设备的更多信息，请参阅the section called “设置其他设备”。要在 NVIDIA Jetson TX2 上使用 GPU 进行推理，在使用 Jetpack 4.3 获取板卡图像时，您必须在设备上安装 CUDA 10.0 和 cuDNN 7.0。
2. 下载 Lambda 函数的静态 PNG 或 JPG 图像以用于图像分类。该示例最适合小型图像文件。

   将图像文件保存在包含 `inference.py` 文件的目录中。您也可以将这些文件保存在此目录的子目录中。此目录位于您在 the section called “创建推理 Lambda 函数。” 中上传的 Lambda 函数部署程序包中。

> **Note**
>
> 或者，您也可以在 Jetson 板上安装摄像头来捕获源图像。但是，我们强烈建议您先用静态图像开始。

3. 编辑 Lambda 函数的配置。按照the section called “将 Lambda 函数添加到组”中的程序进行操作。

> **Note**
>
> 我们建议您在不进行容器化的情况下运行 Lambda 函数，除非您的业务案例需要这样做。这有助于您在无需配置设备资源的前提下访问您的设备 GPU 和摄像头。如果您在没有容器化的情况下运行，则还必须为 AWS IoT Greengrass Lambda 函数授予根用户访问权限。

a. 在不进行容器化的情况下运行：

- 对于运行方式，选择 **`Another user ID/group ID`**。对于 UID，请输入 **`0`**。对于 GUID，请输入 **`0`**。

  这将允许您的 Lambda 函数以根用户身份运行。有关以根用户身份运行的更多信息，请参阅 the section called “为组中的 Lambda 函数设置默认访问身份”。

  > **Tip**
  >
  > 您还必须更新 `config.json` 文件，从而能够为 Lambda 函数授予根用户访问权限。有关此步骤，请参阅 the section called “以根用户身份运行 Lambda 函数”。

- 对于 Lambda 函数容器化，请选择无容器。

  有关不进行容器化的情况下运行的更多信息，请参阅 the section called “选择 Lambda 函数容器化时的注意事项”。

- 将超时值增加到 5 分钟。这可确保请求不会过早超时。设置后，需要几分钟以运行推理。
- 对于已固定，选择 True。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。

b. 改为在容器化模式下运行：

> **Note**
>
> 我们不建议以容器化模式运行，除非您的业务案例需要这样做。

- 提高内存限制值。要在 GPU 模式下使用提供的模型，请使用至少 2000 MB。
- 将超时值增加到 5 分钟。这可确保请求不会过早超时。设置后，需要几分钟以运行推理。
- 对于已固定，选择 True。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。

4. 将 Neo 优化的模型资源添加到组中。上传您在the section called “创建推理 Lambda 函数。”中提取的示例程序包的 `resnet18` 目录中的模型资源。该目录包含使用 Resnet-18 训练的图像分类模型的预编译模型构件。按照the section called “将 Neo 优化的模型资源添加到组中”中的过程操作，并进行以下更新。

   - 将 `resnet18` 目录中的文件压缩到一个名为 `resnet18.zip` 的文件中。
   - 在 Create a machine learning resource (创建机器学习资源) 页面上，对于 Resource name (资源名称)，输入 **`resnet18_model`**。
   - 上传 `resnet18.zip` 文件。

5. 如果在容器化模式下运行，请添加所需的本地设备资源以授予对设备 GPU 的访问权限。

   > **Note**
   >
   > 如果您在非容器化模式下运行，则无需配置设备资源，AWS IoT Greengrass 可直接访问您的设备 GPU。

   a. 在组配置页面上，选择资源选项卡。
   b. 在本地资源部分，选择添加本地资源。
   c. 定义每个资源：

      - 对于资源名称和设备路径，请使用下表中的值。为表中的每个行创建一个设备资源。
      - 对于 Resource type (资源类型)，选择 Device (设备)。
      - 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限。

名称	设备路径
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/nvhost-dbg-gpu
nvhost-prof-gpu	/dev/nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vic
tegra_dc_ctrl	/dev/tegra_dc_ctrl

6. 如果在容器化模式下运行，请添加以下本地卷资源以授予对设备摄像头的访问权限。按照the section called “将 Neo 优化的模型资源添加到组中”中的程序进行操作。

> **Note**
>
> 如果您在非容器化模式下运行，则无需配置设备资源，AWS IoT Greengrass 可直接访问您的设备 摄像头。

- 对于 Resource type (资源类型)，选择 Volume (卷)。
- 在 系统组所有者和文件访问权限 下，选择 自动添加拥有资源的系统组的文件系统权限。

名称	源路径	目的地路径
shm	/dev/shm	/dev/shm
tmp	/tmp	/tmp

7. 更新组订阅以使用正确的目录。按照the section called “将订阅添加到组”中的过程操作，并进行以下更新。

   - 对于第一个主题筛选器，请输入 **/resnet-18/predictions**。
   - 对于第二个主题筛选器，请输入 **/resnet-18/test**。

8. 更新测试订阅以使用正确的目录。按照the section called “测试示例”中的过程操作，并进行以下更新。

   - 对于订阅，选择订阅主题。对于 Subscription topic (订阅主题)，输入 **/resnet-18/predictions**。
   - 在 /resnet-18/predictions 页面上，指定要发布到的 /resnet-18/test 主题。

AWS IoT Greengrass ML 推理问题排查

如果测试失败，您可以尝试以下问题排查步骤。在您的 Raspberry Pi 终端中运行以下命令。

检查错误日志

1. 切换到根用户并导航到 log 目录。访问 AWS IoT Greengrass 日志需要根权限。

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

2. 检查 runtime.log 是否存在任何错误。

   ```
   cat system/runtime.log | grep 'ERROR'
   ```

   您还可以在用户定义的 Lambda 函数日志中查找任何错误：

   ```
   cat user/your-region/your-account-id/lambda-function-name.log | grep 'ERROR'
   ```

   有关更多信息，请参阅the section called “使用日志排查问题”。

验证 Lambda 函数是否已成功部署

1. 列出 /lambda 目录中已部署的 Lambda 的内容。在运行命令前替换占位符值。

```
cd /greengrass/ggc/deployment/lambda/
arn:aws:lambda:region:account:function:function-name:function-version
ls -la
```

2. 验证该目录是否包含与您在 步骤 3：创建推理 Lambda 函数 中上传的 `optimizedImageClassification.zip` 部署程序包相同的内容。

   确保 `.py` 文件和依赖项位于该目录的根目录中。

验证推理模型是否已成功部署

1. 查找 Lambda 运行时进程的进程标识号 (PID)：

   ```
   ps aux | grep lambda-function-name
   ```

   在输出中，PID 显示在 Lambda 运行时进程的行的第二列中。

2. 输入 Lambda 运行时命名空间。请确保在运行命令前替换占位符 *pid* 值。

   > **Note**
   >
   > 此目录及其内容位于 Lambda 运行时命名空间中，因此它们不会显示在常规 Linux 命名空间中。

   ```
   sudo nsenter -t pid -m /bin/bash
   ```

3. 列出您为 ML 资源指定的本地目录的内容。

   > **Note**
   >
   > 如果您的 ML 资源路径不是 `ml_model`，则必须在此处替换。

   ```
   cd /ml_model
   ls -ls
   ```

您应该看到以下文件：

```
    56 -rw-r--r-- 1 ggc_user ggc_group     56703 Oct 29 20:07 model.json
196152 -rw-r--r-- 1 ggc_user ggc_group 200855043 Oct 29 20:08 model.params
   256 -rw-r--r-- 1 ggc_user ggc_group    261848 Oct 29 20:07 model.so
    32 -rw-r--r-- 1 ggc_user ggc_group     30564 Oct 29 20:08 synset.txt
```

Lambda 函数无法找到 **/dev/dri/renderD128**

如果 OpenCL 无法连接到需要的 GPU 设备，则可能会发生这种情况。您必须为 Lambda 函数创建必要设备的设备资源。

后续步骤

接下来，探索其他优化模型。有关信息，请参阅 SageMaker Neo 文档。

在 AWS IoT Greengrass 核心上管理数据流

AWS IoT Greengrass 流管理器可以更轻松、更可靠地将大容量 IoT 数据传输到 AWS Cloud。流管理器在本地处理数据流并自动将其导出到 AWS Cloud。此功能与常见的边缘方案（如机器学习 (ML) 推理）集成，在将数据导出到 AWS Cloud 或本地存储目标之前，将对数据进行本地处理和分析。

流管理器简化了应用程序开发。IoT 应用程序可以使用标准化机制来处理大容量流和管理本地数据保留策略，而不是构建自定义流管理功能。IoT 应用程序可以读取和写入流。它们可以在每个流的基础之上定义存储类型、大小和数据的保留策略，以控制流管理器处理和导出流的方式。

流管理器设计为在间歇性或有限连接的环境中工作。您可以定义带宽使用、超时行为以及当核心连接或断开连接时如何处理流数据。对于关键数据，您可以设置优先级以控制流导出到 AWS Cloud 的顺序。

您可以配置自动导出到 AWS Cloud 以便存储和进行进一步处理和分析。流管理器支持导出到以下 AWS Cloud 目标。

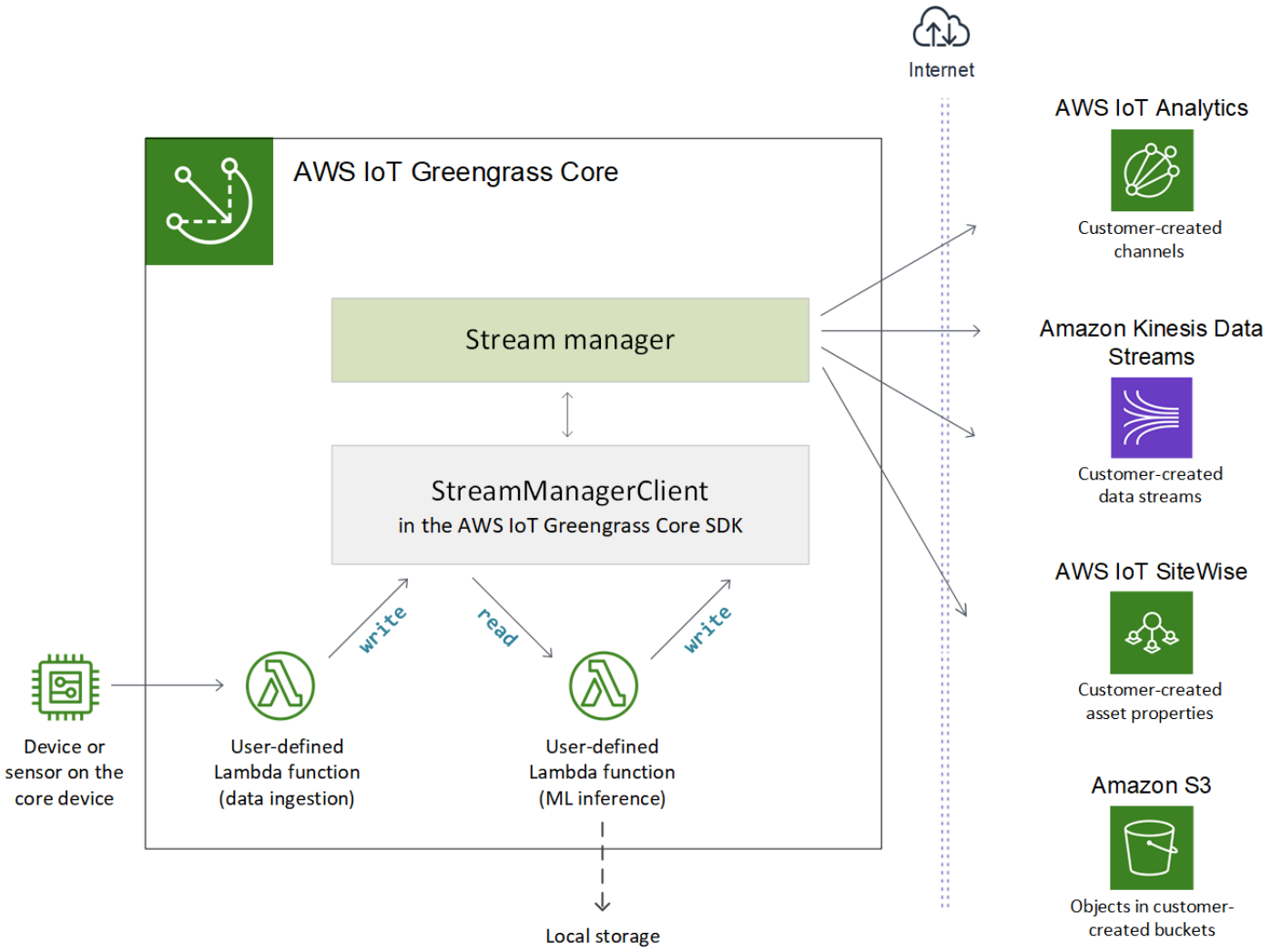
- AWS IoT Analytics 中的频道。AWS IoT Analytics 允许您对数据进行高级分析，以帮助做出业务决策和改进机器学习模型。有关更多信息，请参阅 AWS IoT Analytics 用户指南中的什么是 AWS IoT Analytics？。
- Kinesis Data Streams 中的流。Kinesis Data Streams 通常用于聚合大量数据并将其加载到数据仓库或 map-reduce 集群中。有关更多信息，请参阅 Amazon Kinesis 开发人员指南中的什么是 Amazon Kinesis Data Streams？。
- AWS IoT SiteWise 中的资产属性。AWS IoT SiteWise 允许您大规模收集、整理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？。
- 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 Amazon Simple Storage Service 开发人员指南中的什么是 Amazon S3？。

流管理工作流

您的 IoT 应用程序通过 AWS IoT Greengrass 核心开发工具包与流管理器交互。在简单工作流中，在 Greengrass 核心上运行的用户定义的 Lambda 函数会消耗 IoT 数据，如时间序列温度和压力指标。Lambda 函数可能会过滤或压缩数据，然后调用 AWS IoT Greengrass 核心开发工具包以将数据写入流管理器中的流。流管理器可以根据为流定义的策略自动将流导出到 AWS Cloud。用户定义的 Lambda 函数还可以将数据直接发送到本地数据库或存储库。

您的 IoT 应用程序可以包含多个用户定义的 Lambda 函数用于读取或写入流。这些本地 Lambda 函数可以读取和写入流，以便在本地过滤、聚合和分析数据。这样可以在数据从核心传输到云或本地目的地之前快速响应本地事件并提取有价值的信息。

下图显示了工作流程示例。



要使用流管理器，请首先配置流管理器参数来定义组级别的运行时设置，以应用到 Greengrass 核心上的所有流中。这些可自定义设置允许您根据业务需求和环境约束控制流管理器存储、处理和导出流的方式。有关更多信息，请参阅the section called “配置流管理器”。

配置流管理器后，您可以创建和部署 IoT 应用程序。这些通常是用户定义的 Lambda 函数，它们使用 AWS IoT Greengrass Core 软件包开发工具包中的 `StreamManagerClient` 来创建流并与之交互。在创建流期间，Lambda 函数会定义每个流的策略，例如导出目标、优先级和持久性。有关更多信息（包括 `StreamManagerClient` 操作的代码片段），请参阅the section called “使用 StreamManagerClient 处理流”。

有关配置简单工作流程的教程，请参阅the section called “导出数据流（控制台）”或the section called “导出数据流 (CLI)”。

要求

以下要求适用于使用流管理器：

- 您必须使用 AWS IoT Greengrass 核心软件版本 v1.10 或更高版本且启用了流管理器。有关更多信息，请参阅the section called “配置流管理器”。

  OpenWRT 发行版不支持流管理器。
- 核心设备上必须安装 Java 8 运行时 (JDK 8)。
  - 对于基于 Debian 的发行版（包括 Raspbian）或基于 Ubuntui 的发行版，运行以下命令：

    ```
    sudo apt install openjdk-8-jdk
    ```
  - 对于基于 Red Hat 的发行版（包括 Amazon Linux），请运行以下命令：

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    有关更多信息，请参阅 OpenJDK 文档中的如何下载并安装预先构建的 OpenJDK 程序包。

- 除了基础 AWS IoT Greengrass 核心软件之外，流管理器还需要至少 70 MB 的 RAM。您的总内存需求取决于您的工作负载。

- 用户定义的 Lambda 函数必须使用 AWS IoT Greengrass 核心开发工具包与流管理器交互。AWS IoT Greengrass 核心开发工具包有多种语言可用，但只有以下版本支持流管理器操作。
  - Java SDK（v1.4.0 或更高版本）
  - Python SDK（v1.5.0 或更高版本）
  - Node.js SDK（v1.6.0 或更高版本）

  下载与 Lambda 函数运行时对应的开发工具包版本，并将其包含在 Lambda 函数部署包中。

> **Note**
>
> 适用于 Python 的 AWS IoT Greengrass 核心开发工具包需要 Python 3.7 或更高版本，并且具有其他软件包依赖关系。有关更多信息，请参阅创建 Lambda 函数部署包（控制台）或创建 Lambda 函数部署包 (CLI)。

- 如果您为流定义 AWS Cloud 导出目标，则必须创建导出目标并授予在 Greengrass 组角色中的访问权限。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 不会在核心设备上的组件之间进行静态或本地传输的流数据加密。

- 静态数据。流数据存储在本地存储在 Greengrass 核心的存储目录中。为了保证数据安全，AWS IoT Greengrass 依赖 Unix 文件权限和全磁盘加密（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。

- 本地传输中的数据。AWS IoT Greengrass 不加密数据源、Lambda 函数、AWS IoT Greengrass 核心开发工具包和流管理器之间的本地传输流数据。

- 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。

有关更多信息，请参阅the section called “数据加密”。

客户端身份验证

流管理器客户端使用 AWS IoT Greengrass 核心开发工具包与流管理器进行通信。启用客户端身份验证后，只有 Greengrass 组中的 Lambda 函数才能与流管理器中的流交互。禁用客户端身份验证时，Greengrass 核心上运行的任何进程（如 Docker 容器）都可以与流管理器中的流进行交互。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。您可以从控制台或 AWS IoT Greengrass API 配置此参数。更改在部署组后生效。

	启用	已禁用
参数值	`true`（默认值和推荐值）	`false`
允许的客户端	Greengrass 组中的用户定义 Lambda 函数	Greengrass 组中的用户定义 Lambda 函数 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 “使用 StreamManagerClient 处理流”
- the section called “导出支持的 AWS Cloud 目标的配置”
- the section called “导出数据流（控制台）”
- the section called “导出数据流 (CLI)”

配置 AWS IoT Greengrass 流管理器

在 AWS IoT Greengrass 核心上，流管理器可以存储、处理和导出 IoT 设备数据。流管理器提供用于配置组级运行时设置的参数。这些设置适用于 Greengrass 核心上的所有流。您可以使用 AWS IoT 控制台或 AWS IoT Greengrass API 配置流管理器设置。更改在部署组后生效。

> **Note**
>
> 配置流管理器后，您可以创建和部署在 Greengrass 核心上运行并与流管理器交互的 IoT 应用程序。这些 IoT 应用程序通常是用户定义的 Lambda 函数。有关更多信息，请参阅the section called “使用 StreamManagerClient 处理流”。

流管理器参数

流管理器提供以下允许您定义组级别设置的参数。所有参数都是可选的。

存储目录

参数名称: `STREAM_MANAGER_STORE_ROOT_DIR`

用于存储流的本地目录的绝对路径。此值必须以正斜杠开头（例如，`/data`）。

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”。

AWS IoT Greengrass Core 最低版本：1.10.0

Server port

参数名称: `STREAM_MANAGER_SERVER_PORT`

用于与流管理器通信的本地端口号。默认为 `8088`。

AWS IoT Greengrass Core 最低版本：1.10.0

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。客户端和流管理器之间的所有交互都由 AWS IoT Greengrass 核心开发工具包控制。此参数确定哪些客户端可以调用 AWS IoT Greengrass 核心开发工具包来处理流。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 Lambda 函数作为客户端。Lambda 函数客户端使用内部 AWS IoT Greengrass 核心协议向 AWS IoT Greengrass Core SDK 进行身份验证。

- `false`. 允许在 AWS IoT Greengrass 核心上运行的任何进程成为客户端。除非您的业务案例需要，否则请勿设置为 `false`。例如，仅当核心设备上的非 Lambda 进程必须直接与流管理器（例如在核心上运行的 Docker 容器）通信时，才将此值设置为 `false`。

AWS IoT Greengrass Core 最低版本：1.10.0

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。

AWS IoT Greengrass Core 最低版本：1.10.0

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。每个流都会消耗硬件资源，因此您应该尝试限制受约束设备上的导出流的数量。

AWS IoT Greengrass Core 最低版本：1.10.0

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。

AWS IoT Greengrass Core 最低版本：1.10.0

只读输入文件目录

参数名称: `STREAM_MANAGER_READ_ONLY_DIRS`

以逗号分隔的列表，列出了根文件系统以外存储输入文件的目录的绝对路径。流管理器读取文件并将其上传到 Amazon S3，并将这些目录挂载为只读。有关更多关于导出至 Amazon S3 的信息，请参阅 the section called “Amazon S3 对象”。

仅当满足以下条件时才使用此参数：

- 导出到 Amazon S3 的流的输入文件目录位于以下位置之一：
  - 根文件系统以外的分区。
  - 在根文件系统中的 `/tmp` 下。
- Greengrass 组的默认容器化是 Greengrass 容器。

示例值: `/mnt/directory-1,/mnt/directory-2,/tmp`

AWS IoT Greengrass Core 最低版本：1.11.0

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 Lambda 函数在创建导出到 Amazon S3 的流时会设置此阈值。默认阈值为 5 MB。

AWS IoT Greengrass Core 最低版本：1.11.0

配置流管理器设置（控制台）

您可以使用 AWS IoT 控制台执行以下管理任务：

- 检查是否已启用流管理器
- 在组创建过程中启用或禁用流管理器
- 为现有组启用或禁用流管理器

- 更改流管理器设置

更改将在部署 Greengrass 组后生效。请参阅the section called “导出数据流（控制台）”，其中的教程演示了如何部署一个包含可与流管理器交互的 Lambda 函数的 Greengrass 组。

> **Note**
>
> 使用控制台启用流管理器并部署组时，流管理器的内存大小默认设置为 4194304 KB (4 GB)。建议您将内存大小设置为至少 128000 KB。

检查流管理器是否已启用（控制台）

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择目标组。
3. 选择 Lambda 函数选项卡。
4. 在系统 Lambda 函数下，选择流管理器，然后选择编辑。
5. 检查启用或禁用状态。还会显示所有配置的自定义流管理器设置。

在组创建过程中启用或禁用流管理器（控制台）

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择 Create Group（创建组）。您在下一页上的选择决定了如何为组配置流管理器。
3. 继续完成命名您的组，然后选择 Greengrass 核心页面。
4. 选择创建组。
5. 在组配置页面上，选择 Lambda 函数选项卡，选择流管理器，然后选择编辑。

   - 要使用默认设置启用流管理器，请选择 启用默认设置。

   - 要使用自定义设置启用流管理器，请选择 Customize settings (自定义设置)。

     1. 在配置流管理器页面上，选择启用自定义设置。

2. 在 Custom settings (自定义设置) 下，输入流管理器参数的值。有关更多信息，请参阅the section called “流管理器参数”。将字段留空以允许 AWS IoT Greengrass 使用其默认值。

- 要禁用直播管理器，请选择禁用。
  1. 在 Configure stream manager (配置流管理器) 页面上，选择 Disable (禁用)。

6. 选择 Save（保存）。
7. 继续浏览剩余页面以创建您的组。
8. 在客户端设备页面上，下载安全资源，查看信息，然后选择完成。

> **Note**
>
> 启用流管理器后，必须在核心设备上安装 Java 8 运行时，然后再部署组。

为现有组启用或禁用流管理器（控制台）

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择目标组。
3. 选择 Lambda 函数选项卡。
4. 在系统 Lambda 函数下，选择流管理器，然后选择编辑。
5. 检查启用或禁用状态。还会显示所有配置的自定义流管理器设置。

更改流管理器设置（控制台）

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择目标组。
3. 选择 Lambda 函数选项卡。
4. 在系统 Lambda 函数下，选择流管理器，然后选择编辑。
5. 检查启用或禁用状态。还会显示所有配置的自定义流管理器设置。

6. 选择 Save（保存）。

配置流管理器设置 (CLI)

在 AWS CLI 中，使用系统 `GGStreamManager` Lambda 函数配置流管理器。系统 Lambda 函数是 AWS IoT Greengrass 核心软件的组件。对于流管理器和其他一些系统 Lambda 函数，您可以通过管理 Greengrass 组中的相应 `Function` 和 `FunctionDefinitionVersion` 对象来配置 Greengrass 功能。有关更多信息，请参阅the section called “组对象模型概述”。

您可以使用 API 执行以下管理任务。本节中的示例说明了如何使用 AWS CLI，但您也可以直接调用 AWS IoT Greengrass API 或使用 AWS SDK。

- 检查是否已启用流管理器
- 启用、禁用或配置流管理器

更改在部署组后生效。请参阅 the section called “导出数据流 (CLI)”，其中的教程演示了如何部署一个包含可与流管理器交互的 Lambda 函数的 Greengrass 组。

> **Tip**
>
> 要查看核心设备流管理器是否启用并正在运行，您可以在设备上的终端中运行以下命令。
>
> ```
> ps aux | grep -i 'streammanager'
> ```

检查流管理器是否启用 (CLI)

如果部署的函数定义版本包含系统 `GGStreamManager` Lambda 函数，则启用了流管理器。要进行检查，请执行以下操作;

1. 获取目标 Greengrass 组和组版本的 ID。此过程假定这是最新的组和组版本。以下查询将返回最近创建的组。

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

或者，您也可以按名称查询。系统不要求组名称是唯一的，所以可能会返回多个组。

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

> **Note**
>
> 您还可以在 AWS IoT 控制台中找到这些值。组 ID 显示在组的设置页面上。组版本 ID 显示在组的部署选项卡上。

2. 从输出中的目标组复制 `Id` 和 `LatestVersion` 值。
3. 获取最新的组版本。

   - 将 *group-id* 替换为复制的 `Id`。
   - 将 *latest-group-version-id* 替换为复制的 `LatestVersion`。

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

4. 从输出中的 `FunctionDefinitionVersionArn`，获得函数定义的 ID 和函数定义版本。

   - 函数定义 ID 是 Amazon 资源名称（ARN）中 `functions` 段后面的 GUID。
   - 函数定义版本 ID 是 ARN 中 `versions` 段后面的 GUID。

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/function-definition-id/versions/function-definition-version-id
```

5. 获取函数定义版本。

   - 将 *function-definition-id* 替换为函数定义 ID。
   - 将 *function-definition-version-id* 替换为函数定义版本 ID。

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

如果输出中的 functions 数组包含 GGStreamManager 函数，则启用了流管理器。为函数定义的任何环境变量都表示流管理器的自定义设置。

启用、禁用或配置流管理器 (CLI)

在 AWS CLI 中，使用系统 GGStreamManager Lambda 函数配置流管理器。更改在部署组后生效。

- 要启用流管理器，请在函数定义版本的 functions 数组中包含 GGStreamManager。要配置自定义设置，请为相应的流管理器参数定义环境变量。
- 要禁用流管理器，请从函数定义版本的 functions 数组中删除 GGStreamManager。

带默认设置的流管理器

以下示例配置使用默认设置启用流管理器。它将任意函数 ID 设置为 streamManager。

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

> **Note**
>
> 对于 FunctionConfiguration 属性，您可能知道以下内容：
>
> - 在默认设置下，MemorySize 设置为 4194304 KB (4 GB)。您可随时更改此值。建议您将 MemorySize 设置为至少 128000 KB。
> - Pinned 必须设置为 true。
> - Timeout 是函数定义版本所需的，但 GGStreamManager 不使用它。

带自定义设置的流管理器

以下示例配置使用针对存储目录、服务器端口和线程池大小参数的自定义值来启用流管理器。

```
{
```

```
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "Environment": {
            "Variables": {
                "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                "STREAM_MANAGER_SERVER_PORT": "1234",
                "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
            }
        },
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

对于未指定为环境变量的流管理器参数，AWS IoT Greengrass 将使用默认值。

带有 Amazon S3 自定义导出设置的流管理器

以下示例配置为流管理器启用了上传目录的自定义值和最小分段上传大小参数。

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "Environment": {
            "Variables": {
                "STREAM_MANAGER_READ_ONLY_DIRS": "/mnt/directory-1,/mnt/
directory-2,/tmp",

 "STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES":
 "10485760"
            }
        },
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

启用、禁用或配置流管理器 (CLI)

1. 获取目标 Greengrass 组和组版本的 ID。此过程假定这是最新的组和组版本。以下查询将返回最近创建的组。

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

   或者，您也可以按名称查询。系统不要求组名称是唯一的，所以可能会返回多个组。

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

   > **Note**
   >
   > 您还可以在 AWS IoT 控制台中找到这些值。组 ID 显示在组的设置页面上。组版本 ID 显示在组的部署选项卡上。

2. 从输出中的目标组复制 `Id` 和 `LatestVersion` 值。
3. 获取最新的组版本。

   - 将 *group-id* 替换为复制的 `Id`。
   - 将 *latest-group-version-id* 替换为复制的 `LatestVersion`。

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

4. 从输出中复制 `CoreDefinitionVersionArn` 和所有其他版本的 ARN（`FunctionDefinitionVersionArn` 除外）。在创建组版本时，将使用这些值。
5. 在输出的 `FunctionDefinitionVersionArn` 中，复制函数定义的 ID。该 ID 是 ARN 中的 `functions` 段后面的 GUID，如以下示例所示。

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

> **Note**
>
> 或者，您可以通过运行 create-function-definition 命令以创建一个函数定义，然后从输出中复制该 ID。

6. 在函数定义中添加一个函数定义版本。

   - 使用为函数定义复制的 *## function-definition-id*Id。
   - 在 functions 数组中，包括要在 Greengrass 核心上提供的所有其他函数。您可以使用 get-function-definition-version 命令获取现有函数的列表。

   使用默认设置启用流管理器

   以下示例通过在 functions 数组中包含 GGStreamManager 函数来启用流管理器。此示例使用流管理器参数的默认值。

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "MemorySize":  4194304,
                "Pinned": true,
                "Timeout": 3
            },
            "Id": "streamManager"
        },
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
```

```
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> 示例中的 myLambdaFunction 函数表示用户定义的 Lambda 函数之一。

使用自定义设置启用流管理器

以下示例通过在 functions 数组中包含 GGStreamManager 函数来启用流管理器。除非要更改默认值，否则所有流管理器设置都是可选的。此示例演示如何使用环境变量来设置自定义值。

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "Environment": {
                    "Variables": {
                        "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                        "STREAM_MANAGER_SERVER_PORT": "1234",
                        "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
                    }
                },
                "MemorySize":  4194304,
                "Pinned": true,
                "Timeout": 3
            },
            "Id": "streamManager"
        },
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
```

```
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> 对于 FunctionConfiguration 属性，您可能知道以下内容：
>
> - 在默认设置下，MemorySize 设置为 4194304 KB (4 GB)。您可随时更改此值。建议您将 MemorySize 设置为至少 128000 KB。
> - Pinned 必须设置为 true。
> - Timeout 是函数定义版本所需的，但 GGStreamManager 不使用它。

禁用流管理器

以下示例省略了用于禁用流管理器的 GGStreamManager 函数。

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> 如果不需要部署任何 Lambda 函数，则可以完全省略函数定义版本。

7. 从输出中复制函数定义版本的 `Arn`。
8. 创建一个包含系统 Lambda 函数的组版本。

   - 将 *group-id* 替换为组的 `Id`。
   - 将 *core-definition-version-arn* 替换为从最新组版本中复制的 `CoreDefinitionVersionArn`。
   - 使用为新函数定义版本复制的 *##* function-definition-version-arn`Arn`。
   - 替换从最新组版本中复制的其他组组件（例如 `SubscriptionDefinitionVersionArn` 或 `DeviceDefinitionVersionArn`）的 ARN。
   - 删除任何未使用的参数。例如，如果组版本不包含任何资源，请删除 `--resource-definition-version-arn`。

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

9. 从输出中复制 `Version`。这是新组版本的 ID。
10. 用新组版本替换组。

    - 使用为组复制的 *##* group-id`Id`。
    - 使用为新组版本复制的 *##* group-version-id`Version`。

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

如果您想稍后再次编辑流管理器设置，请按照以下步骤进行操作。请确保创建一个函数定义版本，其中包括具有更新配置的 `GGStreamManager` 函数。组版本必须引用要部署到核心的所有组件版本 ARN。更改在部署组后生效。

另请参阅

- 管理数据流
- the section called “使用 StreamManagerClient 处理流”
- the section called “导出支持的 AWS Cloud 目标的配置”
- the section called “导出数据流（控制台）”
- the section called “导出数据流 (CLI)”

使用 StreamManagerClient 处理流

在 AWS IoT Greengrass 核心上运行的用户定义的 Lambda 函数可以使用 AWS IoT Greengrass Core SDK 中的 `StreamManagerClient` 对象在流管理器中创建流，然后便可以与流进行交互。当 Lambda 函数创建流时，它会定义该流的 AWS Cloud 目标、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，Lambda 函数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 Lambda 函数。如果您的业务案例需要它，您也可以允许在 Greengrass 核心上运行的非 Lambda 进程（例如 Docker 容器）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。有关使用完整 Python Lambda 函数的教程，请参阅 the section called “导出数据流（控制台）” 或 the section called “导出数据流 (CLI)”。

您的 Lambda 函数应该在函数处理程序之外实例化 `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

要创建流，用户定义的 Lambda 函数会调用 create 方法并传入一个 `MessageStreamDefinition` 对象。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 AWS Cloud 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。
- 生存时间 (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：

- Java SDK 中的 MessageStreamDefinition
- Node.js SDK 中的 MessageStreamDefinition
- Python SDK 中的 MessageStreamDefinition

> **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 Lambda 函数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：

- AWS IoT Greengrass Core 最低版本：1.10.0
- AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

> **Note**
>
> 要使用 AWS IoT SiteWise 或 Amazon S3 导出目标创建流，需要满足以下要求：
>
> - AWS IoT Greengrass Core 最低版本：1.11.0
> - AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.6.0 | Java：1.5.0 | Node.js：1.7.0

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 开发工具包参考：create_message_stream | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    client.createMessageStream(
            new MessageStreamDefinition()
                    .withName("StreamName") // Required.
                    .withMaxSize(268435456L)  // Default is 256 MB.
                    .withStreamSegmentSize(16777216L)  // Default is 16 MB.
                    .withTimeToLiveMillis(null)  // By default, no TTL is enabled.
                    .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)  //
 Required.
                    .withPersistence(Persistence.File)  // Default is File.
                    .withFlushOnWrite(false)  // Default is false.
```

```
                    .withExportDefinition(  // Optional. Choose where/how the stream
 is exported to the AWS Cloud.
                            new ExportDefinition()
                                    .withKinesis(null)
                                    .withIotAnalytics(null)
                                    .withIotSitewise(null)
                                    .withS3TaskExecutor(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)
                        .withS3TaskExecutor(null)
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
```

```
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js 开发工具包参考：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 AWS Cloud 目标的配置”。

附加消息

要将数据发送到流管理器进行导出，您的 Lambda 函数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：

- AWS IoT Greengrass Core 最低版本：1.10.0
- AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

> **Note**
>
> 要附加 AWS IoT SiteWise 或 Amazon S3 导出目标的消息，需要满足以下要求：
>
> - AWS IoT Greengrass Core 最低版本：1.11.0
> - AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.6.0 | Java：1.5.0 | Node.js：1.7.0

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 AWS IoT Analytics 或 Kinesis Data Streams 目标，您的 Lambda 函数会附加一个数据 BLOB。

此代码段具有以下要求：

- AWS IoT Greengrass Core 最低版本：1.10.0
- AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

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 =
 GreengrassClientBuilder.streamManagerClient().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 目标，您的 Lambda 函数会附加一个序列化 `PutAssetPropertyValueEntry` 对象。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：

- AWS IoT Greengrass Core 最低版本：1.11.0
- AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.6.0 | Java：1.5.0 | Node.js：1.7.0

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages. Add some randomness to
 time and offset.

    # Note: To create a new asset property data, you should use the classes defined
 in the
    # greengrasssdk.stream_manager module.
```

```
    time_in_nanos = TimeInNanos(
        time_in_seconds=calendar.timegm(time.gmtime()) - random.randint(0, 60),
 offset_in_nanos=random.randint(0, 10000)
    )
    variant = Variant(double_value=random.random())
    asset = [AssetPropertyValue(value=variant, quality=Quality.GOOD,
 timestamp=time_in_nanos)]
    putAssetPropertyValueEntry =
 PutAssetPropertyValueEntry(entry_id=str(uuid.uuid4()),
 property_alias="PropertyAlias", property_values=asset)
    sequence_number = client.append_message(stream_name="StreamName",
 data=Util.validate_and_serialize_to_json_bytes(putAssetPropertyValueEntry))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 参考：append_message | PutAssetPropertyValueEntry

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    Random rand = new Random();
    // Note: To create a new asset property data, you should use the classes defined
 in the
    // com.amazonaws.greengrass.streammanager.model.sitewise package.
    List<AssetPropertyValue> entries = new ArrayList<>() ;

    // IoTSiteWise requires unique timestamps in all messages. Add some randomness
 to time and offset.
    final int maxTimeRandomness = 60;
    final int maxOffsetRandomness = 10000;
    double randomValue = rand.nextDouble();
    TimeInNanos timestamp = new TimeInNanos()
            .withTimeInSeconds(Instant.now().getEpochSecond() -
 rand.nextInt(maxTimeRandomness))
            .withOffsetInNanos((long) (rand.nextInt(maxOffsetRandomness)));
    AssetPropertyValue entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
```

```
            .withTimestamp(timestamp);
    entries.add(entry);

    PutAssetPropertyValueEntry putAssetPropertyValueEntry = new
 PutAssetPropertyValueEntry()
            .withEntryId(UUID.randomUUID().toString())
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues(entries);
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 参考：appendMessage | PutAssetPropertyValueEntry

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const maxTimeRandomness = 60;
        const maxOffsetRandomness = 10000;
        const randomValue = Math.random();
        // Note: To create a new asset property data, you should use the classes
 defined in the
        // aws-greengrass-core-sdk StreamManager module.
        const timestamp = new TimeInNanos()
            .withTimeInSeconds(Math.round(Date.now() / 1000) -
 Math.floor(Math.random() - maxTimeRandomness))
            .withOffsetInNanos(Math.floor(Math.random() * maxOffsetRandomness));
        const entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);

        const putAssetPropertyValueEntry =  new PutAssetPropertyValueEntry()
            .withEntryId(`${ENTRY_ID_PREFIX}${i}`)
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues([entry]);
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
    } catch (e) {
        // Properly handle errors.
```

```
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 参考：appendMessage | 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 Amazon S3 目标，您的 Lambda 函数会附加一个序列化 S3ExportTaskDefinition 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：

- AWS IoT Greengrass Core 最低版本：1.11.0
- AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.6.0 | Java：1.5.0 | Node.js：1.7.0

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",
 data=Util.validate_and_serialize_to_json_bytes(s3_export_task_definition))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python SDK 参考：append_message | S3ExportTaskDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java SDK 参考：appendMessage | S3ExportTaskDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
     // Append an Amazon S3 export task definition and print the sequence number.
     const taskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(taskDefinition)));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js SDK 参考：appendMessage | S3ExportTaskDefinition

读取消息

从流读取消息。

要求

此操作具有以下要求：

- AWS IoT Greengrass Core 最低版本：1.10.0
- AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

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 开发工具包参考：read_messages | ReadMessagesOptions

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    List<Message> messages = client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                    // Try to read from sequence number 100 or greater. By default
 this is 0.
                    .withDesiredStartSequenceNumber(100L)
                    // Try to read 10 messages. If 10 messages are not available,
 then NotEnoughMessagesException is raised. By default, this is 1.
                    .withMinMessageCount(10L)
                    // Accept up to 100 messages. By default this is 1.
                    .withMaxMessageCount(100L)
                    // Try to wait at most 5 seconds for the min_messsage_count to
 be fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                    .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 开发工具包参考：readMessages | ReadMessagesOptions

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messages = await client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
```

```
                // Try to read from sequence number 100 or greater. By default this
 is 0.
                .withDesiredStartSequenceNumber(100)
                // Try to read 10 messages. If 10 messages are not available, then
 NotEnoughMessagesException is thrown. By default, this is 1.
                .withMinMessageCount(10)
                // Accept up to 100 messages. By default this is 1.
                .withMaxMessageCount(100)
                // Try to wait at most 5 seconds for the minMessageCount to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                .withReadTimeoutMillis(5 * 1000)
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js 开发工具包参考：readMessages | 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：

- AWS IoT Greengrass Core 最低版本：1.10.0
- AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

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 =
 GreengrassClientBuilder.streamManagerClient().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：

- AWS IoT Greengrass Core 最低版本：1.10.0
- AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

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 =
 GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo description = client.describeMessageStream("StreamName");
    String lastErrorMessage =
 description.getExportStatuses().get(0).getErrorMessage();
    if (lastErrorMessage != null && !lastErrorMessage.equals("")) {
        // The last export of export destination 0 failed with some error.
        // Here is the last sequence number that was successfully exported.
        description.getExportStatuses().get(0).getLastExportedSequenceNumber();
    }

    if (description.getStorageStatus().getNewestSequenceNumber() >
            description.getExportStatuses().get(0).getLastExportedSequenceNumber())
 {
        // The end of the stream is ahead of the last exported sequence number.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

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 开发工具包参考：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 Lambda 函数遵循以下高级流程来更新流：

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：

- AWS IoT Greengrass Core 最低版本：1.11.0
- AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.6.0 | Java：1.5.0 | Node.js：1.7.0

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 SDK 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 开发工具包参考：update_message_stream | MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messageStreamInfo = await c.describeMessageStream(STREAM_NAME);
        await client.updateMessageStream(
            messageStreamInfo.definition
                // Max Size update should be greater than initial Max Size defined
 in Create Message Stream request
                .withMaxSize(536870912)  // Default is 256 MB. Updating Max Size to
 512 MB.
                .withStreamSegmentSize(33554432)  // Default is 16 MB. Updating
 Segment Size to 32 MB.
                .withTimeToLiveMillis(3600000)  // By default, no TTL is enabled.
 Update TTL to 1 hour.
                .withStrategyOnFull(StrategyOnFull.RejectNewData)  // Required.
 Updating Strategy on full to reject new data.
                .withPersistence(Persistence.Memory)  // Default is File. Update the
 persistence to Memory
                .withFlushOnWrite(true)  // Default is false. Updating to true.
                .withExportDefinition(
```

```
                    // Optional. Choose where/how the stream is exported to the 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：

- AWS IoT Greengrass Core 最低版本：1.10.0
- AWS IoT Greengrass 核心软件开发工具包最低版本：Python：1.5.0 | Java：1.4.0 | Node.js：1.6.0

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 =
 GreengrassClientBuilder.streamManagerClient().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.
});
```

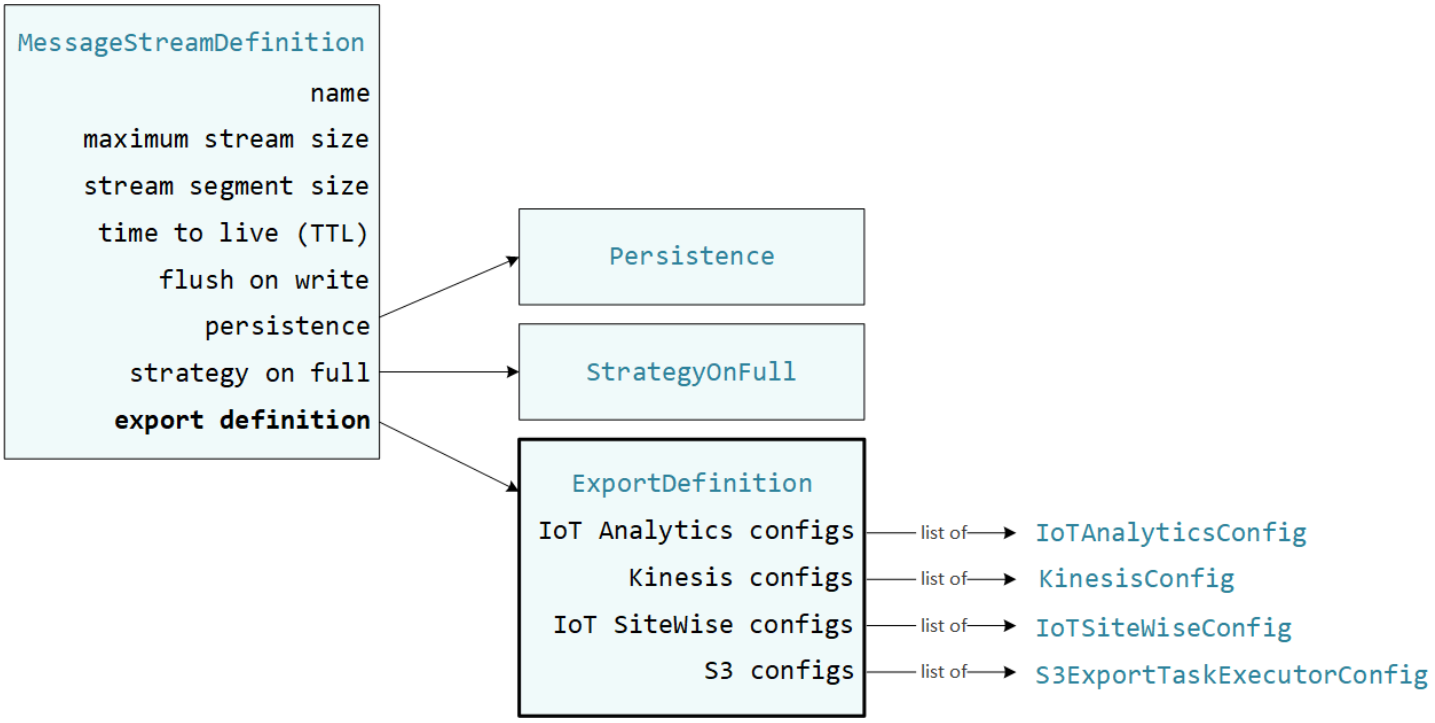
Node.js 开发工具包参考：deleteMessageStream

另请参阅

- 管理数据流
- the section called “配置流管理器”
- the section called “导出支持的 AWS Cloud 目标的配置”
- the section called “导出数据流（控制台）”
- the section called “导出数据流 (CLI)”
- AWS IoT Greengrass Core 软件开发工具包中的 `StreamManagerClient`：
  - Python
  - Java
  - Node.js

导出支持的 AWS Cloud 目标的配置

用户定义的 Lambda 函数使用 AWS IoT Greengrass 核心开发工具包中的 `StreamManagerClient` 与流管理器交互。当 Lambda 函数创建流或更新流时，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 AWS IoT Greengrass 核心软件开发工具包中，您的 Lambda 函数使用 `IoTAnalyticsConfig` 来定义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 开发工具包中的 IoTAnalyticsConfig
- Java 开发工具包中的 IoTAnalyticsConfig
- Node.js 开发工具包中的 IoTAnalyticsConfig

要求

此导出目的地具有以下要求：

- AWS IoT Analytics 中的目标通道必须与 Greengrass 组处于相同的 AWS 账户 和 AWS 区域 中。
- the section called “Greengrass 组角色” 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 AWS IoT Analytics 的流，您的 Lambda 函数会创建一个包含一个或多个 `IoTAnalyticsConfig` 对象的导出定义的流。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。

当您的 Lambda 函数从设备接收数据时，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或 map-reduce 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 AWS IoT Greengrass 核心软件开发工具包中，您的 Lambda 函数使用 `KinesisConfig` 来定义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 开发工具包中的 KinesisConfig
- Java 开发工具包中的 KinesisConf
- Node.js 开发工具包中的 KinesisConfig

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 组处于相同的 AWS 账户 和 AWS 区域 中。
- the section called “Greengrass 组角色” 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 Lambda 函数会创建一个包含一个或多个 `KinesisConfig` 对象的导出定义的流。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。

当您的 Lambda 函数从设备接收数据时，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 AWS IoT Greengrass 核心软件开发工具包中，您的 Lambda 函数使用 `IoTSiteWiseConfig` 来定义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 开发工具包中的 IoTSiteWiseConfig

- Java 开发工具包中的IoTSiteWiseConfig
- Node.js 开发工具包中的 IoTSiteWiseConfig

> **Note**
>
> AWS 还提供 the section called “物联网 SiteWise”，这是一个预先构建的解决方案，可以配合 OPC-UA 源使用。

要求

此导出目的地具有以下要求：

- AWS IoT SiteWise 中的目标资产属性必须与 Greengrass 组处于相同的 AWS 账户 和 AWS 区域中。

  > **Note**
  >
  > 有关 AWS IoT SiteWise 支持的区域的名称的列表，请参阅 AWS 一般参考中的 AWS IoT SiteWise 终端节点和配额。

- the section called “Greengrass 组角色” 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 batchputAssetPropertyPortyValue 授权。

导出到 AWS IoT SiteWise

要创建导出到 AWS IoT SiteWise 的流，您的 Lambda 函数会创建一个包含一个或多个 `IoTSiteWiseConfig` 对象的导出定义的流。此对象定义导出设置，例如批次大小、批次间隔和优先级。

当您的 Lambda 函数从设备接收资产属性数据时，它们会将包含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。

您可以调整流管理器设置和 Lambda 函数逻辑来设计您的导出策略。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，您的 Lambda 函数可以在将数据追加到流之前，将单个资产属性收到的时间戳质量值 (TQV) 数据点汇集在一起。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 AWS IoT Greengrass 核心软件开发工具包中，您的 Lambda 函数使用 `S3ExportTaskExecutorConfig` 来定义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 开发工具包中的 S3ExportTaskExecutorConfig
- Java 开发工具包中的 S3ExportTaskExecutorConfig
- Node.js 开发工具包中的 S3ExportTaskExecutorConfig

要求

此导出目的地具有以下要求：

- Amazon S3 桶必须与 Greengrass 组处于相同的 AWS 账户 中。
- 如果 Greengrass 组的默认容器化是 Greengrass 容器，则必须将 STREAM_MANAGER_READ_ONLY_DIRS 参数设置为使用位于 `/tmp` 下或不在根文件系统下的输入文件目录。
- 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，则必须为该目录创建本地卷资源，并将该目录挂载到具有写入权限的容器中。这样可以确保将文件写入根文件系统并在容器外部可见。有关更多信息，请参阅访问本地资源。
- the section called “Greengrass 组角色” 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/*"
            ]
        }
    ]
}
```

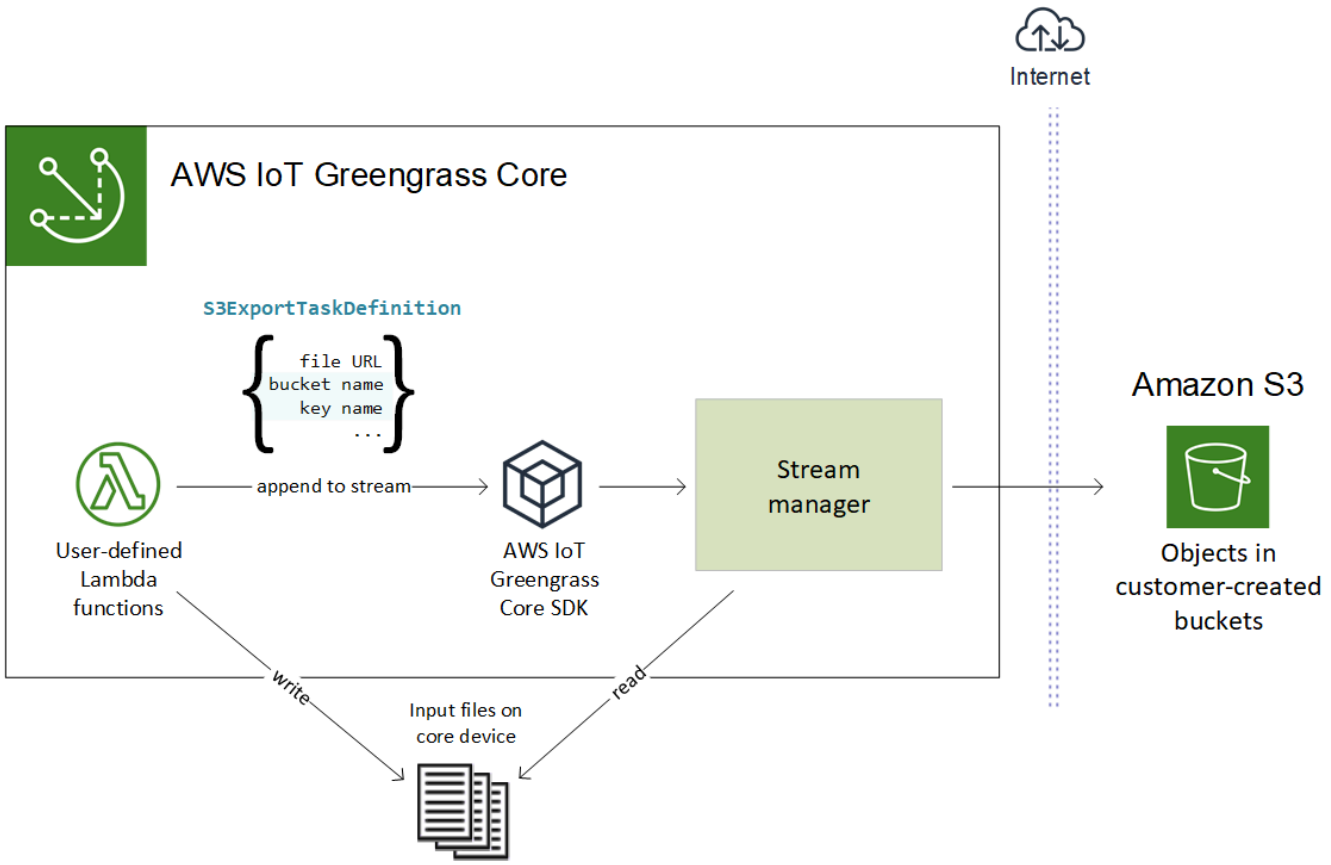
您可以授予对资源的具体或条件访问权限（例如，通过使用通配符 * 命名方案）。有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

导出到 Amazon S3

为了创建导出到 Amazon S3 的流，您的 Lambda 函数使用 `S3ExportTaskExecutorConfig` 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 Lambda 函数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。流管理器按照任务附加到流中的顺序执行任务。

> **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。

下图显示了该高级别流程。



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 Lambda 函数来管理这些数据。

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。

   > **Note**
   >
   > 要确定是否必须配置对输入文件目录的访问权限，请参阅 STREAM_MANAGER_READ_ONLY_DIRS 参数。
   > 流管理器运行的进程继承了该组默认访问身份的所有文件系统权限。流管理器必须拥有访问输入文件的权限。如有必要，您可以使用 `chmod(1)` 命令更改文件的权限。

2. Lambda 函数会扫描该目录，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. Lambda 函数从状态流读取消息以监控导出状态。导出任务完成后，Lambda 函数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 Lambda 函数必须创建状态流，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。这会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。默认为 `INFO`。

- `ERROR`
- `WARN`

- INFO
- DEBUG
- TRACE

以下示例工作流程显示了 Lambda 函数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，Lambda 函数将导出任务附加到流，后者配置为将有关导出任务的状态消息写入状态流。附加操作返回一个表示任务 ID 的序列号。
2. Lambda 函数按顺序读取状态流中的消息，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，Lambda 函数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中的 `S3ExportTaskDefinition` 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`。由于直播或导出定义已删除，或者任务的生存时间 (TTL) 期已过期，该任务已中止。

   > **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，如果无法清理已中止的部分上传，则会返回警告。

3. 导出任务完成后，Lambda 函数可以删除相应的输入文件。

以下示例显示 Lambda 函数如何读取和处理状态消息。

Python

```
import time
from greengrasssdk.stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
```

```
    StreamManagerClient,
)
from greengrasssdk.stream_manager.util import Util

client = StreamManagerClient()

try:
    # Read the statuses from the export status stream
    is_file_uploaded_to_s3 = False
    while not is_file_uploaded_to_s3:
        try:
            messages_list = client.read_messages(
                "StatusStreamName", ReadMessagesOptions(min_message_count=1,
 read_timeout_millis=1000)
            )
            for message in messages_list:
                # Deserialize the status message first.
                status_message = Util.deserialize_json_bytes_to_obj(message.payload,
 StatusMessage)

                # Check the status of the status message. If the status is
 "Success",
                # the file was successfully uploaded to S3.
                # If the status was either "Failure" or "Cancelled", the server was
 unable to upload the file to S3.
                # We will print the message for why the upload to S3 failed from the
 status message.
                # If the status was "InProgress", the status indicates that the
 server has started uploading
                # the S3 task.
                if status_message.status == Status.Success:
                    logger.info("Successfully uploaded file at path " + file_url + "
 to S3.")
                    is_file_uploaded_to_s3 = True
                elif status_message.status == Status.Failure or
 status_message.status == Status.Canceled:
                    logger.info(
                        "Unable to upload file at path " + file_url + " to S3.
 Message: " + status_message.message
                    )
                    is_file_uploaded_to_s3 = True
            time.sleep(5)
        except StreamManagerException:
            logger.exception("Exception while running")
```

```
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Python 开发工具包参考：read_messages | StatusMessage

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.utils.ValidateAndSerialize;
import com.amazonaws.greengrass.streammanager.model.ReadMessagesOptions;
import com.amazonaws.greengrass.streammanager.model.Status;
import com.amazonaws.greengrass.streammanager.model.StatusConfig;
import com.amazonaws.greengrass.streammanager.model.StatusLevel;
import com.amazonaws.greengrass.streammanager.model.StatusMessage;

try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    try {
        boolean isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                List<Message> messages = client.readMessages("StatusStreamName",
                    new
 ReadMessagesOptions().withMinMessageCount(1L).withReadTimeoutMillis(1000L));
                for (Message message : messages) {
                    // Deserialize the status message first.
                    StatusMessage statusMessage =
 ValidateAndSerialize.deserializeJsonBytesToObj(message.getPayload(),
 StatusMessage.class);
                    // Check the status of the status message. If the status is
 "Success", the file was successfully uploaded to S3.
                    // If the status was either "Failure" or "Canceled", the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (Status.Success.equals(statusMessage.getStatus())) {
```

```
                        System.out.println("Successfully uploaded file at path " +
 FILE_URL + " to S3.");
                        isS3UploadComplete = true;
                    } else if (Status.Failure.equals(statusMessage.getStatus()) ||
 Status.Canceled.equals(statusMessage.getStatus())) {
                        System.out.println(String.format("Unable to upload file at
 path %s to S3. Message %s",

 statusMessage.getStatusContext().getS3ExportTaskDefinition().getInputUrl(),
                            statusMessage.getMessage()));
                        sS3UploadComplete = true;
                    }
                }
            } catch (StreamManagerException ignored) {
            } finally {
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                Thread.sleep(5000);
            }
        } catch (e) {
        // Properly handle errors.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Java 开发工具包参考：readMessages | StatusMessage

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require('aws-greengrass-core-sdk').StreamManager;

const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        let isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                const messages = await c.readMessages("StatusStreamName",
```

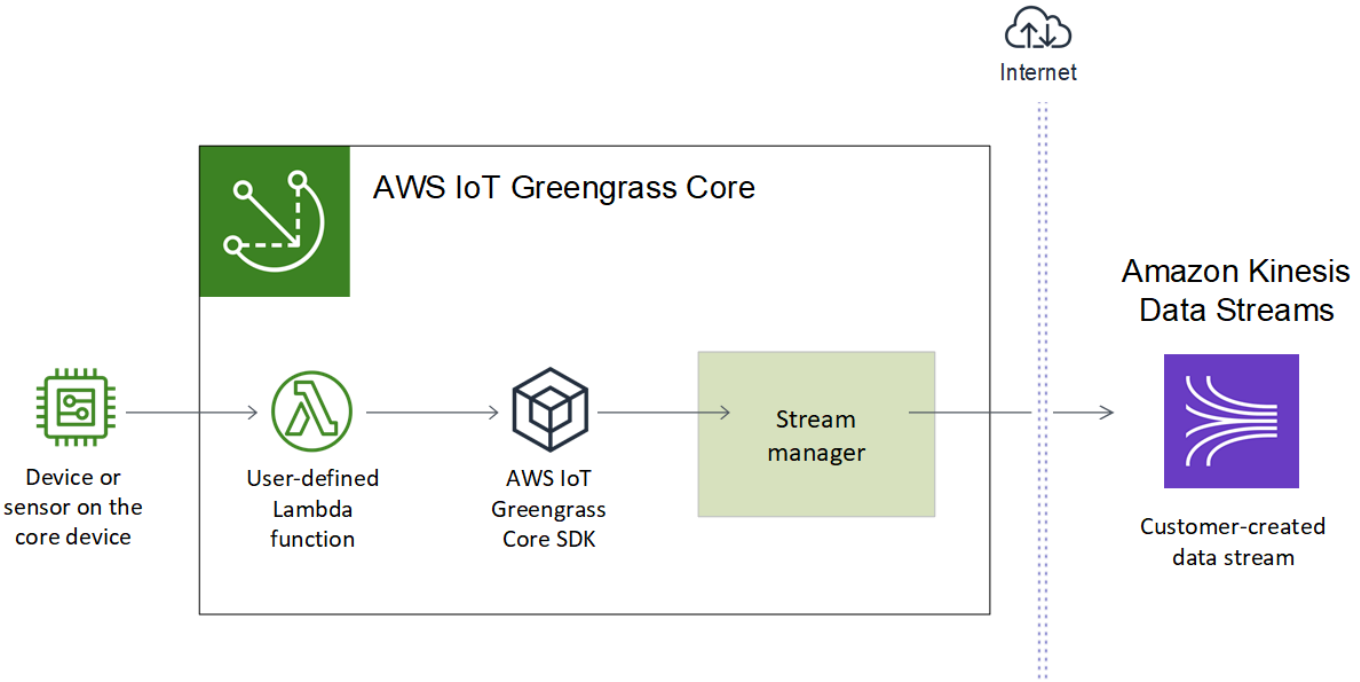
```
                    new ReadMessagesOptions()
                        .withMinMessageCount(1)
                        .withReadTimeoutMillis(1000));

                messages.forEach((message) => {
                    // Deserialize the status message first.
                    const statusMessage =
 util.deserializeJsonBytesToObj(message.payload, StatusMessage);
                    // Check the status of the status message. If the status is
 'Success', the file was successfully uploaded to S3.
                    // If the status was either 'Failure' or 'Cancelled', the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (statusMessage.status === Status.Success) {
                        console.log(`Successfully uploaded file at path ${FILE_URL}
 to S3.`);
                        isS3UploadComplete = true;
                    } else if (statusMessage.status === Status.Failure ||
 statusMessage.status === Status.Canceled) {
                        console.log(`Unable to upload file at path ${FILE_URL} to
 S3. Message: ${statusMessage.message}`);
                        isS3UploadComplete = true;
                    }
                });
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                await new Promise((r) => setTimeout(r, 5000));
            } catch (e) {
                // Ignored
            }
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Node.js 开发工具包参考：readMessages | StatusMessage

将数据流导出到 AWS Cloud（控制台）

本教程介绍如何使用 AWS IoT 控制台配置和部署启用了流管理器的 AWS IoT Greengrass 组。该组包含一个用户定义的 Lambda 函数，该函数可以在流管理器中写入流，然后将其自动导出到 AWS Cloud 中。

流管理器使得摄取、处理和导出大容量数据流更高效也更可靠。在本教程中，您将创建一个使用 IoT 数据的 `TransferStream` Lambda 函数。Lambda 函数使用 AWS IoT Greengrass 核心开发工具包在流管理器中创建流，然后对其进行读写。然后，流管理器将流导出到 Kinesis Data Streams。下图演示了此工作流程。



本教程的重点是展示用户定义的 Lambda 函数如何使用 AWS IoT Greengrass 核心开发工具包中的 `StreamManagerClient` 对象与流管理器进行交互。为简单起见，您为本教程创建的 Python Lambda 函数将生成模拟设备数据。

先决条件

要完成此教程，需要：

- Greengrass 组和 Greengrass Core（v1.10 或更高版本）。有关如何创建 Greengrass 组和核心的信息，请参阅 入门 AWS IoT Greengrass。“入门”教程还包含用于安装 AWS IoT Greengrass Core 软件的步骤。

> **Note**
>
> OpenWRT 发行版不支持流管理器。

- 核心设备上安装的 Java 8 运行时 (JDK 8)。
  - 对于基于 Debian 的发行版（包括 Raspbian）或基于 Ubuntui 的发行版，运行以下命令：

    ```
    sudo apt install openjdk-8-jdk
    ```

  - 对于基于 Red Hat 的发行版（包括 Amazon Linux），请运行以下命令：

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    有关更多信息，请参阅 OpenJDK 文档中的如何下载并安装预先构建的 OpenJDK 程序包。
- 适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包 v1.5.0 或更高版本。要在适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包中使用 `StreamManagerClient`，您必须：
  - 在核心设备上安装 Python 3.7 或更高版本。
  - 将开发工具包和其依赖项包含在 Lambda 函数部署程序包中。本教程中提供了说明。

> **Tip**
>
> 可以将 `StreamManagerClient` 与 Java 或 NodeJS 结合使用。有关示例代码，请参阅 GitHub 上的适用于 Java 的 AWS IoT Greengrass和适用于 Node.js 的 AWS IoT Greengrass 核心开发工具包。

- 在与您的 Greengrass 组相同的 AWS 区域 中，在 Amazon Kinesis Data Streams 中创建的名为 **`MyKinesisStream`** 的目标流。有关更多信息，请参阅 Amazon Kinesis 开发人员指南中的创建流。

> **Note**
>
> 在本教程中，流管理器将数据导出到 Kinesis Data Streams，这将向您的 AWS 账户 账户收取费用。有关定价的信息，请参阅 Kinesis Data Streams 定价。
> 为避免产生费用，您可以在不创建 Kinesis 数据流的情况下运行本教程。在这种情况下，您检查日志以查看流管理器试图将流导出到 Kinesis Data Streams。

- 一个添加到了 `kinesis:PutRecords` 的 IAM 策略，该策略允许对目标数据流执行 the section called “Greengrass 组角色” 操作，如以下示例所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
            ]
        }
    ]
}
```

本教程包含以下概括步骤：

1. 创建 Lambda 函数部署程序包
2. 创建 Lambda 函数
3. 将函数添加到组
4. 启用流管理器
5. 配置本地日志记录
6. 部署组
7. 测试应用程序

完成本教程大约需要 20 分钟。

步骤 1：创建 Lambda 函数部署程序包

在此步骤中，您创建包含 Python 函数代码和依赖项的 Lambda 函数部署包。您稍后在 AWS Lambda 中创建 Lambda 函数时上传此程序包。Lambda 函数使用 AWS IoT Greengrass 核心开发工具包创建本地流并与之交互。

> **Note**
>
> 用户定义的 Lambda 函数必须使用 AWS IoT Greengrass 核心开发工具包与流管理器交互。有关 Greengrass 流管理器的要求的更多信息，请参阅 Greengrass 流管理器要求。

1. 下载适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包 v1.5.0 或更高版本。
2. 解压缩下载的程序包以获取软件开发工具包。软件开发工具包是 `greengrasssdk` 文件夹。
3. 安装程序包依赖项以将其包含在 Lambda 函数部署程序包的开发工具包中。

   1. 导航到包含该 `requirements.txt` 文件的开发工具包目录。此文件列出了依赖项。
   2. 安装开发工具包依赖项。例如，运行以下 `pip` 命令将它们安装在当前目录中：

      ```
      pip install --target . -r requirements.txt
      ```
4. 将以下 Python 代码函数保存在名为 `transfer_stream.py` 的本地文件中。

   > **Tip**
   >
   > 有关使用 Java 和 NodeJS 的示例代码，请参阅 GitHub 上的适用于 Java 的 AWS IoT Greengrass 核心开发工具包和适用于 Node.js AWS IoT Greengrass 核心开发工具包。

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)


# This example creates a local stream named "SomeStream".
```

```
# It starts writing data into that stream and then stream manager automatically
 exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
 MB).
# Any data appended after the stream reaches the size limit continues to be
 appended, and
# stream manager deletes the oldest data until the total stream size is back under
 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
 script is
# uploaded to Kinesis and you will be charged for that usage.


def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
 kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
 strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
```

```
            client.append_message(stream_name, "ABCDEFGHIJKLMNO".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
 ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
 to the stream")
        # Now start putting in random data between 0 and 1000 to emulate device
 sensor input
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
 1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

    except asyncio.TimeoutError:
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")


def function_handler(event, context):
    return


logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. 将以下项目压缩到名为 `transfer_stream_python.zip` 的文件中。此即 Lambda 函数部署程序包。

   - transfer_stream.py。应用程序逻辑。
   - greengrasssdk。发布 MQTT 消息的 Python Greengrass Lambda 函数所需的库。

流管理器操作在适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包 v1.5.0 或更高版本中可用。

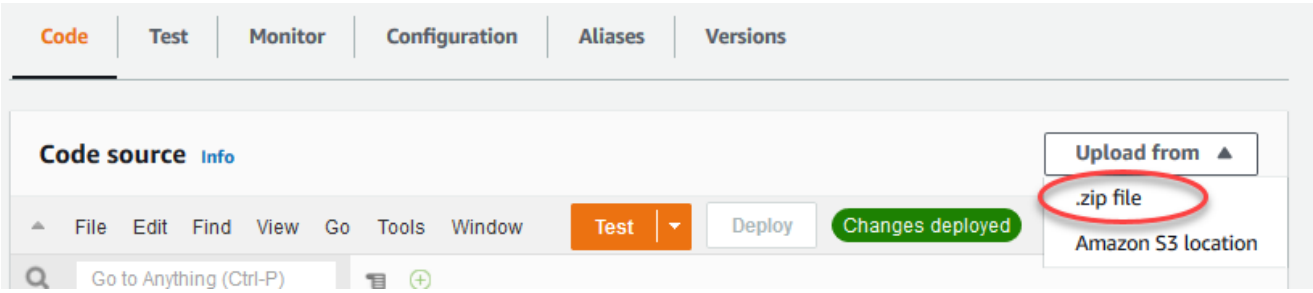
- 您为适用于 Python 的 AWS IoT Greengrass 核心开发工具包（例如，`cbor2` 目录）安装的依赖项。

创建 `zip` 文件时，仅包含这些项目，而不是包含文件夹。

步骤 2：创建 Lambda 函数

在此步骤中，您使用 AWS Lambda 控制台创建 Lambda 函数，然后将其配置为使用您的部署包。接着，发布函数版本并创建别名。

1. 首先，创建 Lambda 函数。
   a. 在AWS Management Console中，选择服务，然后打开 AWS Lambda 控制台。
   b. 选择创建函数，然后选择从头开始创作。
   c. 在基本信息部分中，使用以下值：
      - 对于函数名称，请输入 **`TransferStream`**。
      - 对于运行时系统，选择 Python 3.7。
      - 对于权限，请保留默认设置。这将创建一个授予基本 Lambda 权限的执行角色。此角色不由 AWS IoT Greengrass 使用。
   d. 在页面底部，选择创建函数。
2. 接下来，注册处理程序并上传您的 Lambda 函数部署包。
   a. 在代码选项卡上的代码源下，选择上传自。从下拉列表中选择 .zip 文件。



   b. 选择上传，然后选择您的 `transfer_stream_python.zip` 部署包。然后，选择保存。
   c. 在函数的代码选项卡中，在运行时设置下选择编辑，然后输入以下值。

- 对于运行时系统，选择 Python 3.7。
- 对于处理程序，输入 **`transfer_stream.function_handler`**。

d. 选择 Save（保存）。

> **Note**
>
> AWS Lambda 控制台上的测试按钮不适用于此功能。AWS IoT Greengrass Core 软件开发工具包不包含在 AWS Lambda 控制台中独立运行 Greengrass Lambda 函数所需的模块。这些模块（例如 `greengrass_common`）是在函数部署到您的 Greengrass 核心之后提供给它们的。

3. 现在，发布 Lambda 函数的第一个版本并创建此版本的别名。

> **Note**
>
> Greengrass 组可以按别名（推荐）或版本引用 Lambda 函数。使用别名，您可以更轻松地管理代码更新，因为您在更新函数代码时，不必更改订阅表或组定义。相反，您只需将别名指向新的函数版本。

a. 在操作菜单上，选择发布新版本。

b. 对于版本描述，输入 **`First version`**，然后选择发布。

c. 在 TransferStream: 1 (TransferStream：1) 配置页面上，从 Actions (操作) 菜单中选择 Create alias (创建别名)。

d. 在创建新别名页面上，使用以下值：

- 对于名称，输入 **`GG_TransferStream`**。
- 对于版本，选择 1。

> **Note**
>
> AWS IoT Greengrass 不支持 $LATEST 版本的 Lambda 别名。

e. 选择 Create（创建）。

现在，您已准备就绪，可以将 Lambda 函数添加到 Greengrass 组。

步骤 3：将 Lambda 函数添加到 Greengrass 组

在该步骤中，您将 Lambda 函数添加到该组，然后配置其生命周期和环境变量。有关更多信息，请参阅the section called “控制 Greengrass Lambda 函数执行”。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择目标组。
3. 在组配置页面上，选择Lambda 函数选项卡。
4. 在我的 Lambda 函数部分下，选择添加。
5. 在添加 Lambda 函数页面上，为您的 Lambda 函数选择 Lambda 函数。
6. 在 Lambda 版本中，选择 Alias:GG_TransferStream。

   现在，配置用于确定 Greengrass 组中 Lambda 函数的行为的属性。
7. 在 Lambda 函数配置部分中，进行以下更改。

   - 将 Memory limit (内存限制) 设置为 32 MB。
   - 对于已固定，选择 True。

   > **Note**
   >
   > 长时间生存（或固定）的 Lambda 函数在 AWS IoT Greengrass 启动后自动启动并在自己的容器中保持运行。这与按需 Lambda 函数相反，后者在调用时启动，并在没有要运行的任务时停止。有关更多信息，请参阅the section called “生命周期配置”。
8. 选择添加 Lambda 函数。

步骤 4：启用流管理器

在此步骤中，您确保启用流管理器。

1. 在组配置页面上，选择Lambda 函数选项卡。
2. 在系统 Lambda 函数下，选择流管理器，然后检查状态。如果禁用，请选择 Edit (编辑)。然后，选择 Enable (启用) 和 Save (保存)。您可以对本教程使用默认参数设置。有关更多信息，请参阅the section called “配置流管理器”。

> **Note**
>
> 使用控制台启用流管理器并部署组时，流管理器的内存大小默认设置为 4194304 KB (4 GB)。建议您将内存大小设置为至少 128000 KB。

步骤 5：配置本地日志记录

在此步骤中，您将配置 AWS IoT Greengrass 系统组件、用户定义的 Lambda 函数以及组中的连接器，以将日志写入核心设备的文件系统。您可以使用日志对可能遇到的任何问题进行故障排除。有关更多信息，请参阅the section called “利用 AWS IoT Greengrass 日志进行监控”。

1. 在 Local logs configuration (本地日志配置) 下，检查是否配置了本地日志记录。
2. 如果未为 Greengrass 系统组件或用户定义的 Lambda 函数配置日志，请选择编辑。
3. 选择用户 Lambda 函数日志级别和 Greengrass 系统日志级别。
4. 保留日志记录级别和磁盘空间限制的默认值，然后选择 Save (保存)。

步骤 6：部署 Greengrass 组

将组部署到核心设备。

1. 确保 AWS IoT Greengrass 核心正在运行。根据需要在您的 Raspberry Pi 终端中运行以下命令。

   a. 要检查进程守护程序是否正在运行，请执行以下操作：

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

   如果输出包含 `root` 的 `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon` 条目，则表示进程守护程序正在运行。

   > **Note**
   >
   > 路径中的版本取决于您的核心设备上安装的 AWS IoT Greengrass Core 软件版本。

   b. 启动进程守护程序：

   ```
   cd /greengrass/ggc/core/
   ```

```
sudo ./greengrassd start
```

2. 在组配置页面上，选择部署。
3. a. 在 Lambda 函数选项卡的系统 Lambda 函数部分下，选择 IP 检测器，再选择编辑。
   b. 在编辑 IP 检测器设置对话框中，选择自动检测和覆盖 MQTT 代理端点。
   c. 选择 Save（保存）。

   这使得设备可以自动获取核心的连接信息，例如 IP 地址、DNS 和端口号。建议使用自动检测，不过 AWS IoT Greengrass 也支持手动指定的端点。只有在首次部署组时，系统才会提示您选择发现方法。

   > **Note**
   >
   > 如果出现提示，请授予权限，以创建 Greengrass 服务角色并将其关联至当前 AWS 区域中的 AWS 账户。该角色将允许 AWS IoT Greengrass 访问您在 AWS 服务中的资源。

   部署页面显示了部署时间戳、版本 ID 和状态。完成后，部署的状态应显示为成功完成。

   有关问题排查帮助，请参阅排查问题。

步骤 7：测试应用程序

`TransferStream` Lambda 函数生成模拟的设备数据。它将数据写入流管理器导出到目标 Kinesis 数据流的流。

1. 在 Amazon Kinesis 控制台的 Kinesis data streams 下，选择 MyKinesisStream。

   > **Note**
   >
   > 如果您在运行教程时没有目标 Kinesis 数据流，请检查流管理器的日志文件 (`GGStreamManager`)。如果它在错误消息中包含 `export stream MyKinesisStream doesn't exist`，则测试成功。此错误意味着服务试图导出到流，但流不存在。

2. 在 MyKinesisStream 页面上，选择 Monitoring (监控)。如果测试成功，您应在 Put Records (放置记录) 图表中看到数据。根据您的连接，显示数据可能需要一分钟时间。

> **Important**
>
> 测试完成后，删除 Kinesis 数据流以避免产生更多费用。
> 运行以下命令以停止 Greengrass 守护程序。这样可以防止核心发送消息，直到您准备好继续测试。
>
> ```
> cd /greengrass/ggc/core/
> sudo ./greengrassd stop
> ```

3. 从核心中删除 TransferStream Lambda 函数。

   a. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。

   b. 在 Greengrass 组下，选择您的组。

   c. 在 Lambdas 页面上，为 TransferStream 函数选择省略号 (...)，然后选择删除函数。

   d. 从操作中，选择部署。

要查看日志记录信息或解决流的问题，请检查日志中的 `TransferStream` 和 `GGStreamManager` 函数。您必须具有 `root` 权限以读取文件系统上的 AWS IoT Greengrass 日志。

- `TransferStream` 将日志条目写入 *`greengrass-root`*`/ggc/var/log/user/`*`region`*`/`*`account-id`*`/TransferStream.log`。
- `GGStreamManager` 将日志条目写入 *`greengrass-root`*`/ggc/var/log/system/GGStreamManager.log`。

如果您需要更多故障排除信息，可以将用户 Lambda 日志的日志级别设置为调试日志，然后再次部署该组。

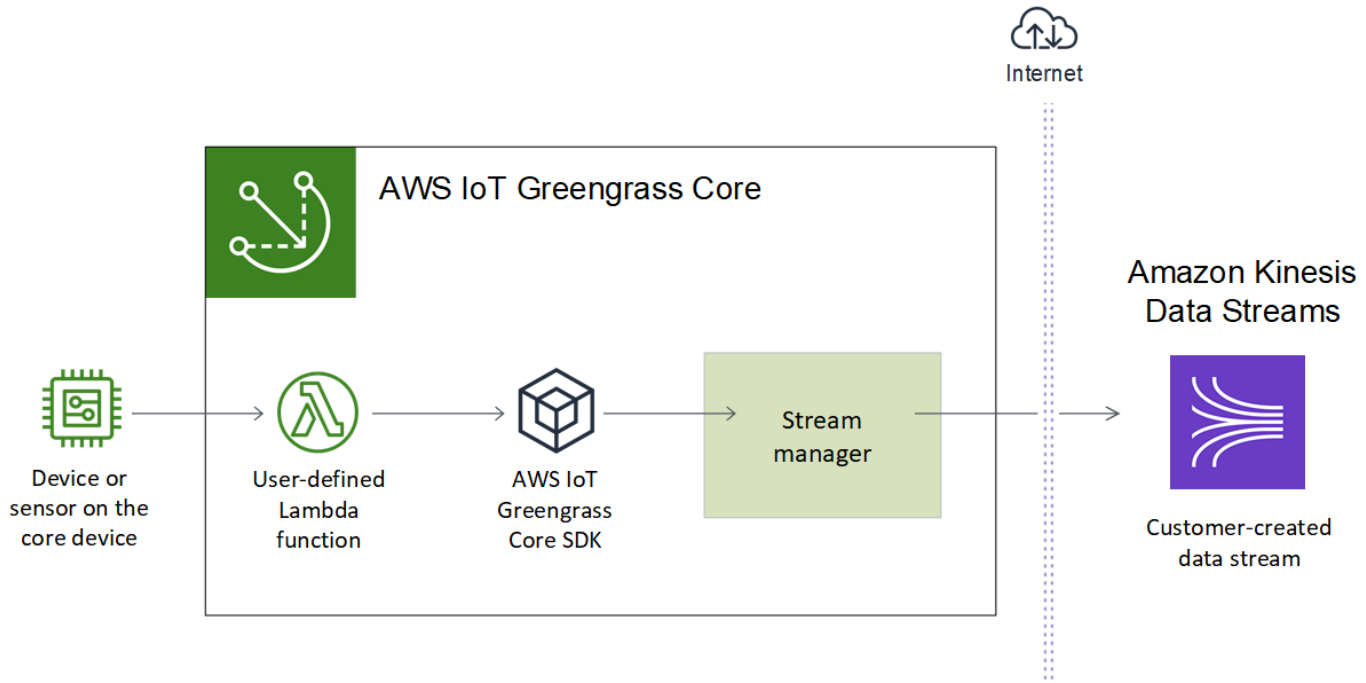
另请参阅

- 管理数据流
- the section called “配置流管理器”
- the section called “使用 StreamManagerClient 处理流”
- the section called “导出支持的 AWS Cloud 目标的配置”
- the section called “导出数据流 (CLI)”

将数据流导出到 AWS Cloud (CLI)

本教程介绍如何使用 AWS CLI 配置和部署启用了流管理器的 AWS IoT Greengrass 组。该组包含一个用户定义的 Lambda 函数，该函数可以在流管理器中写入流，然后将其自动导出到 AWS Cloud 中。

流管理器使得摄取、处理和导出大容量数据流更高效也更可靠。在本教程中，您将创建一个使用 IoT 数据的 `TransferStream` Lambda 函数。Lambda 函数使用 AWS IoT Greengrass 核心开发工具包在流管理器中创建流，然后对其进行读写。然后，流管理器将流导出到 Kinesis Data Streams。下图演示了此工作流程。



本教程的重点是展示用户定义的 Lambda 函数如何使用 AWS IoT Greengrass 核心开发工具包中的 `StreamManagerClient` 对象与流管理器进行交互。为简单起见，您为本教程创建的 Python Lambda 函数将生成模拟设备数据。

当您使用 AWS IoT Greengrass API（在 AWS CLI 中，包含 Greengrass 命令）创建组时，默认情况下会禁用流管理器。要在核心上启用流管理器，您需要创建一个函数定义版本，其中包括系统 `GGStreamManager` Lambda 函数和一个引用新的函数定义版本的组版本。然后部署组。

先决条件

要完成此教程，需要：

- Greengrass 组和 Greengrass Core（v1.10 或更高版本）。有关如何创建 Greengrass 组和核心的信息，请参阅 入门 AWS IoT Greengrass。“入门”教程还包含用于安装 AWS IoT Greengrass Core 软件的步骤。

  > **Note**
  >
  > OpenWRT 发行版不支持流管理器。

- 核心设备上安装的 Java 8 运行时 (JDK 8)。
  - 对于基于 Debian 的发行版（包括 Raspbian）或基于 Ubuntui 的发行版，运行以下命令：

    ```
    sudo apt install openjdk-8-jdk
    ```

  - 对于基于 Red Hat 的发行版（包括 Amazon Linux），请运行以下命令：

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    有关更多信息，请参阅 OpenJDK 文档中的如何下载并安装预先构建的 OpenJDK 程序包。
- 适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包 v1.5.0 或更高版本。要在适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包中使用 `StreamManagerClient`，您必须：
  - 在核心设备上安装 Python 3.7 或更高版本。
  - 将开发工具包和其依赖项包含在 Lambda 函数部署程序包中。本教程中提供了说明。

  > **Tip**
  >
  > 可以将 `StreamManagerClient` 与 Java 或 NodeJS 结合使用。有关示例代码，请参阅 GitHub 上的适用于 Java 的 AWS IoT Greengrass和适用于 Node.js 的 AWS IoT Greengrass 核心开发工具包。

- 在与您的 Greengrass 组相同的 AWS 区域 中，在 Amazon Kinesis Data Streams 中创建的名为 **`MyKinesisStream`** 的目标流。有关更多信息，请参阅 Amazon Kinesis 开发人员指南中的创建流。

  > **Note**
  >
  > 在本教程中，流管理器将数据导出到 Kinesis Data Streams，这将向您的 AWS 账户 账户收取费用。有关定价的信息，请参阅 Kinesis Data Streams 定价。

为避免产生费用，您可以在不创建 Kinesis 数据流的情况下运行本教程。在这种情况下，您检查日志以查看流管理器试图将流导出到 Kinesis Data Streams。

- 一个添加到了 `kinesis:PutRecords` 的 IAM 策略，该策略允许对目标数据流执行 the section called “Greengrass 组角色” 操作，如以下示例所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
                "arn:aws:kinesis:region:account-id:stream/MyKinesisStream"
            ]
        }
    ]
}
```

- 您的计算机上安装和配置的 AWS CLI。有关更多信息，请参阅 AWS Command Line Interface 用户指南中的安装 AWS Command Line Interface 和配置 AWS CLI 。

本教程中的示例命令是针对 Linux 及其他基于 Unix 的系统编写的。如果您使用的是 Windows，请参阅为 AWS 命令行界面指定参数值以了解语法上的差异。

如果命令包含 JSON 字符串，本教程提供了在单行包含 JSON 的示例。在某些系统上，使用此格式可能会更高效地编辑和运行命令。

本教程包含以下概括步骤：

1. 创建 Lambda 函数部署程序包
2. 创建 Lambda 函数
3. 创建函数定义和版本

4. 创建记录器定义和版本
5. 获取核心定义版本的 ARN
6. 创建组版本
7. 创建部署
8. 测试应用程序

完成本教程大约需要 30 分钟。

步骤 1：创建 Lambda 函数部署程序包

在此步骤中，您创建包含 Python 函数代码和依赖项的 Lambda 函数部署包。您稍后在 AWS Lambda 中创建 Lambda 函数时上传此程序包。Lambda 函数使用 AWS IoT Greengrass 核心开发工具包创建本地流并与之交互。

> **Note**
>
> 用户定义的 Lambda 函数必须使用 AWS IoT Greengrass 核心开发工具包与流管理器交互。有关 Greengrass 流管理器的要求的更多信息，请参阅 Greengrass 流管理器要求。

1. 下载适用于 Python 的 AWS IoT Greengrass Core 开发工具包 v1.5.0 或更高版本。
2. 解压缩下载的程序包以获取软件开发工具包。软件开发工具包是 `greengrasssdk` 文件夹。
3. 安装程序包依赖项以将其包含在 Lambda 函数部署程序包的开发工具包中。
   1. 导航到包含该 `requirements.txt` 文件的开发工具包目录。此文件列出了依赖项。
   2. 安装开发工具包依赖项。例如，运行以下 `pip` 命令将它们安装在当前目录中：

      ```
      pip install --target . -r requirements.txt
      ```
4. 将以下 Python 代码函数保存在名为 `transfer_stream.py` 的本地文件中。

> **Tip**
>
> 有关使用 Java 和 NodeJS 的示例代码，请参阅 GitHub 上的适用于 Java 的 AWS IoT Greengrass 核心开发工具包和适用于 Node.js AWS IoT Greengrass 核心开发工具包。

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)


# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
 exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
 MB).
# Any data appended after the stream reaches the size limit continues to be
 appended, and
# stream manager deletes the oldest data until the total stream size is back under
 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
 script is
# uploaded to Kinesis and you will be charged for that usage.


def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
```

```
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHIJKLMNO".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
to the stream")
        # Now start putting in random data between 0 and 1000 to emulate device
sensor input
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

    except asyncio.TimeoutError:
```

```
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")


def function_handler(event, context):
    return


logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. 将以下项目压缩到名为 `transfer_stream_python.zip` 的文件中。此即 Lambda 函数部署程序包。

   - transfer_stream.py。应用程序逻辑。
   - greengrasssdk。发布 MQTT 消息的 Python Greengrass Lambda 函数所需的库。

     流管理器操作在适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包 v1.5.0 或更高版本中可用。
   - 您为适用于 Python 的 AWS IoT Greengrass 核心开发工具包（例如，`cbor2` 目录）安装的依赖项。

   创建 `zip` 文件时，仅包含这些项目，而不是包含文件夹。

步骤 2：创建 Lambda 函数

1. 创建 IAM 角色，以便您可以在创建该函数时传入角色 ARN。

   JSON Expanded

   ```
   aws iam create-role --role-name Lambda_empty --assume-role-policy '{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "lambda.amazonaws.com"
               },
   ```

```
            "Action": "sts:AssumeRole"
        }
    ]
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
 "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
 "lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
```

> **Note**
>
> AWS IoT Greengrass 不使用此角色，因为对 Greengrass Lambda 函数的权限在 Greengrass 组角色中指定。对于本教程，您将创建一个空角色。

2. 从输出中复制 `Arn`。
3. 使用 AWS Lambda API 创建 `TransferStream` 函数。以下命令假定该 zip 文件位于当前目录中。

   - 将 *`role-arn`* 替换为复制的 `Arn`。

```
aws lambda create-function \
--function-name TransferStream \
--zip-file fileb://transfer_stream_python.zip \
--role role-arn \
--handler transfer_stream.function_handler \
--runtime python3.7
```

4. 发布该函数的版本。

```
aws lambda publish-version --function-name TransferStream --description 'First
 version'
```

5. 为发布的版本创建别名。

   Greengrass 组可以按别名（推荐）或版本引用 Lambda 函数。使用别名，您可以更轻松地管理代码更新，因为您在更新函数代码时，不必更改订阅表或组定义。相反，您只需将别名指向新的函数版本。

```
aws lambda create-alias --function-name TransferStream --name GG_TransferStream --
function-version 1
```

> **Note**
>
> AWS IoT Greengrass 不支持 $LATEST 版本的 Lambda 别名。

6. 从输出中复制 `AliasArn`。为 AWS IoT Greengrass 配置函数时，可以使用此值。

现在，您已准备就绪，可以为 AWS IoT Greengrass 配置函数。

步骤 3：创建函数定义和版本

此步骤将创建引用系统 `GGStreamManager` Lambda 函数和用户定义的 `TransferStream` Lambda 函数的函数定义版本。要在使用 AWS IoT Greengrass API 时启用流管理器，您的函数定义版本必须包含 `GGStreamManager` 函数。

1. 创建一个包含初始版本的函数定义，该初始版本包含系统和用户定义的 Lambda 函数。

   以下定义版本启用了具有默认参数设置的流管理器。要配置自定义设置，您必须为相应的流管理器参数定义环境变量。有关示例，请参阅 the section called “启用、禁用或配置流管理器”。AWS IoT Greengrass 对省略的参数使用默认设置。 `MemorySize` 应该至少是 `128000`。`Pinned` 必须设置为 `true`。

   > **Note**
   >
   > 长时间生存（或固定）的 Lambda 函数在 AWS IoT Greengrass 启动后自动启动并在自己的容器中保持运行。这与按需 Lambda 函数相反，后者在调用时启动，并在没有要运行的任务时停止。有关更多信息，请参阅the section called “生命周期配置”。

   - 将 *`arbitrary-function-id`* 替换为函数名称，如 **`stream-manager`**。
   - 将 *`alias-arn`* 替换为您为 `TransferStream` Lambda 函数创建别名时复制的 `AliasArn`。

JSON expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
    "Functions": [
        {
            "Id": "arbitrary-function-id",
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "MemorySize": 128000,
                "Pinned": true,
                "Timeout": 3
            }
        },
        {
            "Id": "TransferStreamFunction",
            "FunctionArn": "alias-arn",
            "FunctionConfiguration": {
                "Executable": "transfer_stream.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            }
        }
    ]
}'
```

JSON single

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "arbitrary-function-
id","FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
 "FunctionConfiguration": {"Environment": {"Variables":
{"STREAM_MANAGER_STORE_ROOT_DIR": "/data","STREAM_MANAGER_SERVER_PORT":
 "1234","STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH": "20000"}},"MemorySize":
 128000,"Pinned": true,"Timeout": 3}},{"Id": "TransferStreamFunction",
 "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
 "transfer_stream.function_handler", "MemorySize": 16000,"Pinned":
 true,"Timeout": 5}}]}'
```

> **Note**
>
> `Timeout` 是函数定义版本所需的，但 `GGStreamManager` 不使用它。有关 `Timeout` 和其他组级别设置的更多信息，请参阅 the section called “控制 Greengrass Lambda 函数执行”。

2. 从输出中复制 `LatestVersionArn`。您将使用此值向部署到核心的组版本添加函数定义版本。

步骤 4：创建记录器定义和版本

配置组的日志记录设置。在本教程中，您将配置 AWS IoT Greengrass 系统组件、用户定义的 Lambda 函数以及连接器，以将日志写入核心设备的文件系统。您可以使用日志对可能遇到的任何问题进行故障排除。有关更多信息，请参阅the section called “利用 AWS IoT Greengrass 日志进行监控”。

1. 创建一个包含初始版本的记录器定义。

   JSON Expanded

```
aws greengrass create-logger-definition --name "LoggingConfigs" --initial-
version '{
    "Loggers": [
        {
            "Id": "1",
            "Component": "GreengrassSystem",
            "Level": "INFO",
            "Space": 10240,
            "Type": "FileSystem"
        },
        {
            "Id": "2",
            "Component": "Lambda",
            "Level": "INFO",
            "Space": 10240,
            "Type": "FileSystem"
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-logger-definition \
    --name "LoggingConfigs" \
    --initial-version '{"Loggers":
[{"Id":"1","Component":"GreengrassSystem","Level":"INFO","Space":10240,"Type":"FileSyste
{"Id":"2","Component":"Lambda","Level":"INFO","Space":10240,"Type":"FileSystem"}]}'
```

2. 从输出中复制记录器定义的 `LatestVersionArn`。您将使用此值向部署到核心的组版本添加记录器定义版本。

步骤 5：获取核心定义版本的 ARN

获取要添加到新组版本的核心定义版本的 ARN。要部署组版本，该组版本必须引用包含确切一个核心的核心定义版本。

1. 获取目标 Greengrass 组和组版本的 ID。此过程假定这是最新的组和组版本。以下查询将返回最近创建的组。

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

或者，您也可以按名称查询。系统不要求组名称是唯一的，所以可能会返回多个组。

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

> **Note**
>
> 您还可以在 AWS IoT 控制台中找到这些值。组 ID 显示在组的设置页面上。组版本 ID 显示在组的部署选项卡上。

2. 从输出中复制目标组的 `Id`。您将使用此值获取核心定义版本及部署组的时间。
3. 从输出中复制 `LatestVersion`，这是添加到组的最后一个版本的 ID。您将使用此值获取核心定义版本。
4. 获取核心定义版本的 ARN：

   a. 获取组版本。

- 使用为组复制的 *##* group-idId。
- 使用为组复制的 LatestVersion 替换 *group-version-id*。

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

b. 从输出中复制 CoreDefinitionVersionArn。您将使用此值向部署到核心的组版本添加核心定义版本。

步骤 6：创建组版本

现在，您已准备就绪，可以创建一个包含您要部署的实体的组版本。要实现此目的，需要创建一个引用每种组件类型的目标版本的组版本。在本教程中，您将包括核心定义版本、函数定义版本和记录器定义版本。

1. 创建组版本。

   - 使用为组复制的 *##* group-idId。
   - 使用为核心函数版本复制的 CoreDefinitionVersionArn 替换 *core-definition-version-arn*。
   - 将 *function-definition-version-arn* 替换为您为新函数定义版本复制的 LatestVersionArn。
   - 将 *logger-definition-version-arn* 替换为您为新的记录器定义版本复制的 LatestVersionArn。

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

2. 从输出中复制 Version。这是新组版本的 ID。

步骤 7：创建部署

将组部署到核心设备。

1. 确保 AWS IoT Greengrass 核心正在运行。根据需要在您的 Raspberry Pi 终端中运行以下命令。

   a. 要检查守护程序是否正在运行，请执行以下操作：

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

   如果输出包含 root 的 /greengrass/ggc/packages/*ggc-version*/bin/daemon 条目，则表示守护程序正在运行。

   > **Note**
   >
   > 路径中的版本取决于您的核心设备上安装的 AWS IoT Greengrass 核心软件版本。

   b. 启动进程守护程序：

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

2. 创建 部署。

   - 使用为组复制的 *##* group-idId。
   - 使用为新组版本复制的 *##* group-version-idVersion。

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

3. 从输出中复制 DeploymentId。

4. 获取部署状态。

   - 使用为组复制的 *##* group-idId。
   - 将 *deployment-id* 替换为您针对部署复制的 DeploymentId。

```
aws greengrass get-deployment-status \
--group-id group-id \
--deployment-id deployment-id
```

如果状态为 Success，则部署成功。有关问题排查帮助，请参阅排查问题。

步骤 8：测试应用程序

TransferStream Lambda 函数生成模拟的设备数据。它将数据写入流管理器导出到目标 Kinesis 数据流的流。

1. 在 Amazon Kinesis 控制台的 Kinesis data streams 下，选择 MyKinesisStream。

> **Note**
>
> 如果您在运行教程时没有目标 Kinesis 数据流，请检查流管理器的日志文件 (GGStreamManager)。如果它在错误消息中包含 `export stream MyKinesisStream doesn't exist`，则测试成功。此错误意味着服务试图导出到流，但流不存在。

2. 在 MyKinesisStream 页面上，选择 Monitoring (监控)。如果测试成功，您应在 Put Records (放置记录) 图表中看到数据。根据您的连接，显示数据可能需要一分钟时间。

> **Important**
>
> 测试完成后，删除 Kinesis 数据流以避免产生更多费用。
> 运行以下命令以停止 Greengrass 守护程序。这样可以防止核心发送消息，直到您准备好继续测试。
>
> ```
> cd /greengrass/ggc/core/
> sudo ./greengrassd stop
> ```

3. 从核心中删除 TransferStream Lambda 函数。

   a. 按照the section called “创建组版本”创建新的组版本，但删除 `create-group-version` 命令中的 `--function-definition-version-arn` 选项。或者，创建不包括 TransferStream Lambda 函数的函数定义版本。

> **Note**
>
> 通过省略部署的组版本中的系统 GGStreamManager Lambda 函数，可以禁用核心上的流管理。

b. 按照the section called “创建部署”以部署新的组版本。

要查看日志记录信息或解决流的问题，请检查日志中的 TransferStream 和 GGStreamManager 函数。您必须具有 root 权限以读取文件系统上的 AWS IoT Greengrass 日志。

- TransferStream 将日志条目写入 *greengrass-root*/ggc/var/log/user/*region*/*account-id*/TransferStream.log。
- GGStreamManager 将日志条目写入 *greengrass-root*/ggc/var/log/system/GGStreamManager.log。

如果需要更多疑难解答信息，可以将 Lambda 日志记录级别设置为 DEBUG，然后创建并部署新的组版本。

另请参阅

- 管理数据流
- the section called “使用 StreamManagerClient 处理流”
- the section called “导出支持的 AWS Cloud 目标的配置”
- the section called “配置流管理器”
- the section called “导出数据流（控制台）”
- AWS CLI 命令参考中的AWS Identity and Access Management (IAM) 命令
- AWS CLI 命令参考中的AWS Lambda 命令
- AWS CLI 命令参考中的AWS IoT Greengrass 命令

将密钥部署到 AWS IoT Greengrass 核心

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

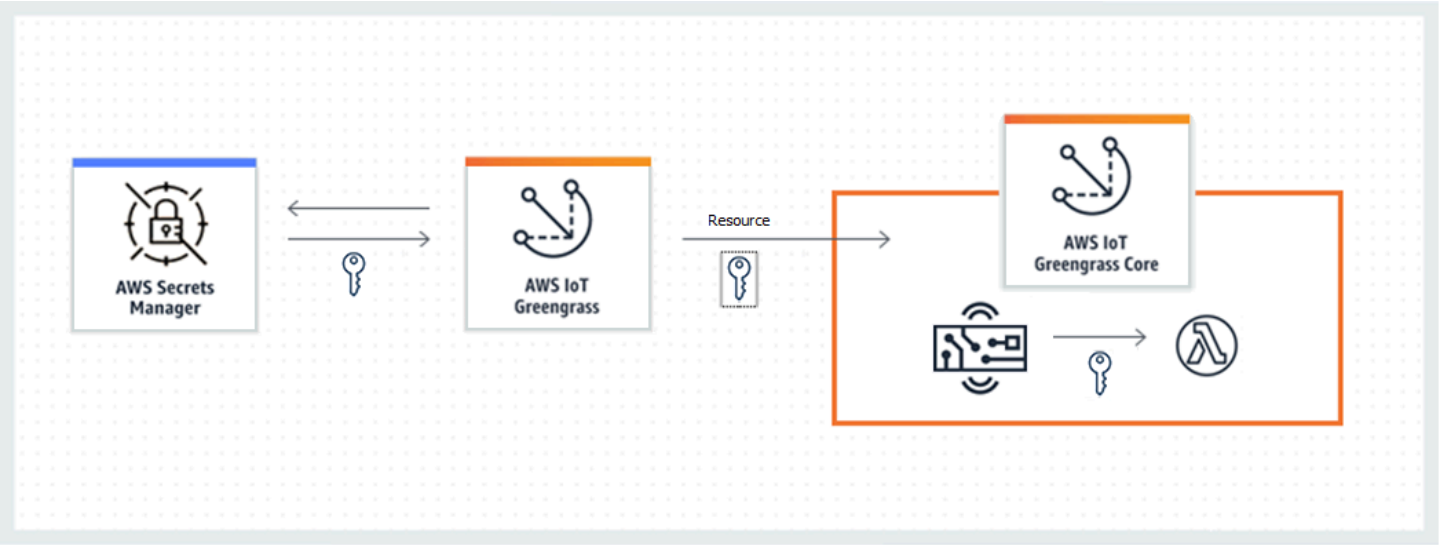
您可以从 Greengrass 设备对服务和应用程序进行身份验证，而无需对密码、令牌或其他密钥进行硬编码。

AWS Secrets Manager 是一项服务，您可以使用该服务在云中安全地存储和管理密钥。AWS IoT Greengrass 将 Secrets Manager 扩展到 Greengrass 核心设备，从而使连接器和 Lambda 函数可以使用本地密钥与服务和应用程序交互。例如，Twilio Notifications 使用本地存储的身份验证令牌。

要将密钥集成到 Greengrass 组中，您需要创建一个引用 Secrets Manager 密钥的组资源。此密钥资源引用云密钥 ARN。要了解如何创建、管理和使用私有资源，请参阅the section called “使用密钥资源”。

AWS IoT Greengrass 对传输中的密钥和静态密钥进行加密。在组部署期间，AWS IoT Greengrass 从 Secrets Manager 提取密钥并在 Greengrass 核心上创建本地加密副本。在 Secrets Manager 中轮换您的云密钥后，重新部署组，将更新后的值传播到核心。

下图显示了将密钥部署到核心的简要过程。传输和静态中的密钥均经过加密。



通过 AWS IoT Greengrass 将密钥存储在本地具有以下优势：

- 通过代码解耦（而非硬编码）。这支持集中管理的凭证，并帮助保护敏感数据免受入侵的风险。
- 适用于离线场景。连接器和函数可以在 Internet 连接断开的情况下安全访问本地服务和软件。

- 受控密钥访问。只有组中经过授权的连接器和函数才可以访问您的密钥。AWS IoT Greengrass 使用私有密钥加密来保护您的密钥。传输和静态中的密钥均经过加密。有关更多信息，请参阅the section called “密钥加密”。
- 受控轮换。在 Secrets Manager 中轮换您的密钥后，重新部署 Greengrass 组以更新密钥的本地副本。有关更多信息，请参阅the section called “创建和管理密钥”。

> **Important**
>
> 在轮换云版本后，AWS IoT Greengrass 不会自动更新本地密钥的值。要更新本地值，必须重新部署组。

密钥加密

AWS IoT Greengrass 对传输和静态中的密钥进行加密。

> **Important**
>
> 请确保用户定义的 Lambda 函数能够安全地处理密钥，并且不记录存储在密钥中的任何敏感数据。有关更多信息，请参阅 AWS Secrets Manager 用户指南中的降低记录和调试 Lambda 函数的风险。尽管本文档特别提到了轮换函数，但该建议也适用于 Greengrass Lambda 函数。

传输中加密

AWS IoT Greengrass 使用传输层安全性 (TLS) 来加密通过 Internet 和本地网络进行的所有通信。这样可以在传输中保护密钥，在从 Secrets Manager 检索密钥并将其部署到核心时会发生此类情况。有关支持的 TLS 密码套件，请参阅the section called “TLS 密码套件支持”。

静态加密

AWS IoT Greengrass 使用在 `config.json` 中指定的私有密钥对存储在核心中的密钥进行加密。因此，私有密钥的安全存储对于保护本地密钥至关重要。在 AWS 责任共担模式中，客户有责任保证私有密钥在核心设备上的安全存储。

AWS IoT Greengrass 支持两种私有密钥存储：

- 使用硬件安全模块。有关更多信息，请参阅the section called “硬件安全性集成”。

> **Note**
>
> 目前，AWS IoT Greengrass 仅支持 PKCS#1 v1.5 填充机制，用于在使用基于硬件的私钥时对本地机密进行加密和解密。如果您按照供应商提供的说明手动生成基于硬件的私钥，请务必选择 PKCS #1 v1.5。AWS IoT Greengrass 不支持最佳非对称加密填充 (OAEP)。

- 使用文件系统权限（默认）。

私有密钥用于保护数据密钥，数据密钥用于加密本地密钥。数据密钥随每次组部署进行轮换。

AWS IoT Greengrass 核心是唯一有权访问私有密钥的实体。与密钥资源关联的 Greengrass 连接器或 Lambda 函数从核心获取密钥值。

要求

以下是本地密钥支持的要求：

- 您必须使用 AWS IoT Greengrass Core v1.7 或更高版本。
- 要获取本地密钥的值，用户定义的 Lambda 函数必须使用 AWS IoT Greengrass Core SDK 1.3.0 或更高版本。
- 用于本地密钥加密的私有密钥必须在 Greengrass 配置文件中指定。默认情况下，AWS IoT Greengrass 使用存储在文件系统中的核心私有密钥。要提供您自己的私有密钥，请参阅the section called “指定用于密钥加密的私有密钥”。仅支持 RSA 密钥类型。

> **Note**
>
> 目前，AWS IoT Greengrass 仅支持 PKCS#1 v1.5 填充机制，用于在使用基于硬件的私钥时对本地机密进行加密和解密。如果您按照供应商提供的说明手动生成基于硬件的私钥，请务必选择 PKCS #1 v1.5。AWS IoT Greengrass 不支持最佳非对称加密填充 (OAEP)。

- 必须为 AWS IoT Greengrass 授予权限才能获取您的密钥值。这样，AWS IoT Greengrass 可以在组部署期间提取值。如果使用的是默认 Greengrass 服务角色，则 AWS IoT Greengrass 已经有权访问名称以 greengrass- 开头的密钥。要自定义访问权限，请参阅the section called “允许 AWS IoT Greengrass 获取密钥值”。

> **Note**
>
> 我们建议您使用此命名约定来标识允许 AWS IoT Greengrass 访问的密钥，即使您自定义权限也是如此。控制台使用不同的权限来读取您的密钥，以便您在控制台中选择 AWS IoT Greengrass 无权提取的密钥。使用命名约定有助于避免权限冲突，这会导致部署错误。

指定用于密钥加密的私有密钥

在此过程中，您将提供用于本地秘密加密的私有密钥的路径。这必须是最小长度为 2048 位 RSA 密钥。有关 AWS IoT Greengrass 核心上使用的私有密钥的更多信息，请参阅 the section called “安全委托人”。

AWS IoT Greengrass 支持两种模式的私有密钥存储：基于硬件或基于文件系统（默认）。有关更多信息，请参阅the section called “密钥加密”。

请仅在您想更改默认配置（其使用文件系统中的核心私有密钥）时，此按照此过程操作。我们在编写这些步骤时假设您创建了组和核心，如入门教程中的模块 2 所述。

1. 打开 `config.json` 文件（位于 */greengrass-root*`/config` 目录中）。

   > **Note**
   >
   greengrass-root 表示在您的设备上安装 AWS IoT Greengrass 核心软件的路径。通常，这是 `/greengrass` 目录。

2. 在 `crypto.principals.SecretsManager` 对象中，对于 `privateKeyPath` 属性，输入私有密钥的路径：

   - 如果您的私有密钥存储在文件系统中，请指定该密钥的绝对路径。例如：

     ```
     "SecretsManager" : {
       "privateKeyPath" : "file:///somepath/hash.private.key"
     }
     ```

   - 如果私有密钥存储在硬件安全模块 (HSM) 中，请使用 RFC 7512 PKCS#11 URI 方案指定路径：例如：

     ```
     "SecretsManager" : {
     ```

```
  "privateKeyPath" : "pkcs11:object=private-key-label;type=private"
}
```

有关更多信息，请参阅the section called “硬件安全性配置”。

> **Note**
>
> 目前，AWS IoT Greengrass 仅支持 PKCS#1 v1.5 填充机制，用于在使用基于硬件的私钥时对本地机密进行加密和解密。如果您按照供应商提供的说明手动生成基于硬件的私钥，请务必选择 PKCS #1 v1.5。AWS IoT Greengrass 不支持最佳非对称加密填充 (OAEP)。

允许 AWS IoT Greengrass 获取密钥值

在此过程中，您将为允许 AWS IoT Greengrass 获取密钥值的 Greengrass 服务角色添加内联策略。

请仅在需要向 AWS IoT Greengrass 授予自定义密钥权限或 Greengrass 服务角色不包括 `AWSGreengrassResourceAccessRolePolicy` 托管策略时，才按照此过程操作。`AWSGreengrassResourceAccessRolePolicy` 授予对名称以 greengrass- 开头的密钥的访问权限。

1. 运行以下 CLI 命令以获取 Greengrass 服务角色的 ARN：

   ```
   aws greengrass get-service-role-for-account --region region
   ```

   返回的 ARN 包含角色名称。

   ```
   {
     "AssociatedAt": "time-stamp",
     "RoleArn": "arn:aws:iam::account-id:role/service-role/role-name"
   }
   ```

   您将在以下步骤中使用 ARN 或名称。

2. 添加允许 `secretsmanager:GetSecretValue` 操作的内联策略。有关说明，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

您可以明确列出密钥或使用通配符 * 命名方案来授予细粒度访问权限，也可以授予对受版本控制或标记的密钥的有条件访问权限。例如，以下策略允许 AWS IoT Greengrass 仅读取指定的密钥。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
                "arn:aws:secretsmanager:region:account-id:secret:greengrass-SecretA-abc",
                "arn:aws:secretsmanager:region:account-id:secret:greengrass-SecretB-xyz"
            ]
        }
    ]
}
```

> **Note**
>
> 如果使用客户管理的 AWS KMS 密钥来加密密钥，则 Greengrass 服务角色还必须允许 `kms:Decrypt` 操作。

有关密钥管理器的 IAM 策略的详细信息，请参阅《AWS Secrets Manager 用户指南》中的 AWS Secrets Manager 的身份验证和访问控制以及可以在 IAM 策略或密钥策略中针对 AWS Secrets Manager 使用的操作、资源和上下文密钥。

另请参阅

- AWS Secrets Manager 用户指南 中的什么是 AWS Secrets Manager？
- PKCS #1：RSA 加密版本 1.5

使用密钥资源

AWS IoT Greengrass 使用密钥资源 将来自 AWS Secrets Manager 的密钥集成到 Greengrass 组中。密钥资源是对 Secrets Manager 密钥的引用。有关更多信息，请参阅将密钥部署到核心。

在AWS IoT Greengrass 核心设备上，连接器和 Lambda 函数可以使用密钥资源完成服务和应用程序的身份验证，而无需对密码、令牌或其他凭证进行硬编码。

创建和管理密钥

在 Greengrass 组中，密钥资源引用 Secrets Manager 密钥的 ARN。将密钥资源部署到核心时，密钥值经过加密，可供关联的连接器和 Lambda 函数使用。有关更多信息，请参阅the section called “密钥加密”。

您将使用 Secrets Manager 创建和管理密钥的云版本。使用 AWS IoT Greengrass 创建、管理和部署您的密钥资源。

> **Important**
>
> 我们建议您遵循在 Secrets Manager 中轮换密钥的最佳实践。然后，部署 Greengrass 组，以更新密钥的本地副本。有关更多信息，请参阅 AWS Secrets Manager用户指南中的轮换 AWS Secrets Manager 密钥。

使密钥在 Greengrass 核心上可用

1. 可以在 Secrets Manager 中创建秘密。这是密钥的云版本，它集中存储和托管在 Secrets Manager 中。管理任务包括轮换密钥值和应用资源策略。
2. 在 AWS IoT Greengrass 中创建一个密钥资源。这是一种引用云密钥 ARN 的组资源。每个组一次只能引用一个密钥。
3. 使用控制台或 配置 Lambda 函数。您必须通过指定对应的参数或属性，将资源与连接器或函数相关联。这样可使其获取本地部署的密钥资源值。有关更多信息，请参阅the section called “使用本地密钥”。
4. 部署 Greengrass 组。在部署期间，AWS IoT Greengrass 提取云密钥的值并在核心上创建（或更新）本地密钥。

每当 AWS IoT Greengrass 检索密钥值时，Secrets Manager 都会在 AWS CloudTrail 中记录事件。AWS IoT Greengrass 不会记录与本地密钥部署或使用相关的任何事件。有关 Secrets Manager 日

志记录的更多信息，请参阅《AWS Secrets Manager 用户指南》中的监控您的 AWS Secrets Manager 密钥的使用。

在密钥资源中添加暂存标签

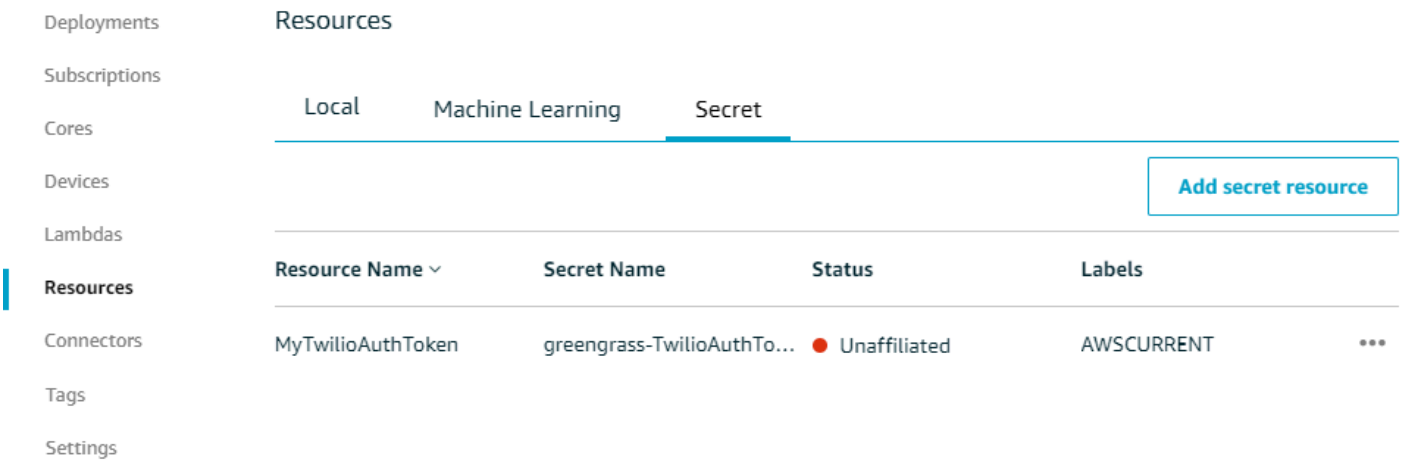
Secrets Manager 使用暂存标签来标识密钥值的特定版本。暂存标签可以是系统定义的，也可以是用户定义的。Secrets Manager 将 AWSCURRENT 标签分配给最新版本的密钥值。暂存标签通常用于管理密钥轮换。有关 Secrets Manager 版本控制的更多信息，请参阅 AWS Secrets Manager 用户指南中的 AWS Secrets Manager 的主要术语和概念。

密钥资源始终包含 AWSCURRENT 暂存标签，还可以选择包含其他暂存标签（如果 Lambda 函数或连接器需要）。在组部署期间，AWS IoT Greengrass 检索在组中引用的暂存标签值，然后在核心上创建或更新对应的值。

创建和管理密钥资源（控制台）

创建密钥资源（控制台）

在 AWS IoT Greengrass 控制台中，您可以通过组的资源页面上的密钥选项卡来创建和管理密钥资源。有关创建密钥资源和将其添加到组的教程，请参阅the section called “如何创建密钥资源（控制台）”和the section called “连接器入门（控制台）”。



> **Note**
>
> 或者，控制台允许您在配置连接器或 Lambda 函数时创建密钥和密钥资源。您可以从连接器的配置参数页面或 Lambda 函数的资源页面执行此操作。

管理密钥资源（控制台）

Greengrass 组中密钥资源的管理任务包括向组中添加密钥资源、从组中移除密钥资源以及更改密钥资源中包含的暂存标签集。

如果您指向与 Secrets Manager 不同的密钥，还必须编辑使用该密钥的任何连接器：

1. 在组配置页面上，选择 Connectors (连接器)。
2. 从连接器的上下文菜单中，选择 Edit (编辑)。
3. Edit parameters (编辑参数) 页面将显示一条消息，以通知您密钥 ARN 已发生更改。要确认更改，请选择 Save (保存)。

如果您在 Secrets Manager 中删除密钥，请从该组以及引用该密钥的连接器和 Lambda 函数中删除相应的密钥资源。否则，在组部署期间，AWS IoT Greengrass 会返回一条指示找不到密钥的错误。此外，您还可以根据需要更新 Lambda 函数代码。

创建和管理密钥资源 (CLI)

创建密钥资源 (CLI)

在 AWS IoT Greengrass API 中，密钥是一种组资源。以下示例通过包括密钥资源（名为 `MySecretResource`）的初始版本，创建资源定义。有关创建密钥资源和将其添加到组版本的教程，请参阅the section called “连接器入门 (CLI)”。

密钥资源引用对应的 Secrets Manager 密钥 ARN，并包括除 `AWSCURRENT`（其始终包括在内）之外的两个暂存标签。

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-
version '{
    "Resources": [
        {
            "Id": "my-resource-id",
            "Name": "MySecretResource",
            "ResourceDataContainer": {
                "SecretsManagerSecretResourceData": {
                    "ARN": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
                    "AdditionalStagingLabelsToDownload": [
                        "Label1",
                        "Label2"
                    ]
```

```
                }
            }
        }
    ]
}'
```

管理密钥资源 (CLI)

Greengrass 组中密钥资源的管理任务包括向组中添加密钥资源、从组中移除密钥资源以及更改密钥资源中包含的暂存标签集。

在 AWS IoT Greengrass API 中，这些更改通过使用版本来实施。

AWS IoT Greengrass API 使用版本来管理群组。版本是不可变的，因此要添加或更改组组件（例如群组的客户端设备、函数和资源），必须创建新的或更新的组件的版本。然后，创建并部署组版本，其中包含每个组件的目标版本。要详细了解组，请参阅 the section called “AWS IoT Greengrass 组”。

例如，要更改密钥资源的暂存标签集，请执行以下操作：

1. 创建资源定义版本，其中包含更新后的密钥资源。以下示例将向上一部分的密钥资源添加第三个暂存标签。

   > **Note**
   >
   > 要向版本添加更多资源，请将其纳入 `Resources` 数组中。

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-
version '{
    "Resources": [
        {
            "Id": "my-resource-id",
            "Name": "MySecretResource",
            "ResourceDataContainer": {
                "SecretsManagerSecretResourceData": {
                    "ARN": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
                    "AdditionalStagingLabelsToDownload": [
                        "Label1",
                        "Label2",
                        "Label3"
                    ]
```

```
                }
            }
        }
    ]
}'
```

2. 如果密钥资源的 ID 发生更改，请更新使用该密钥资源的连接器和函数。在新版本中，更新与资源 ID 对应的参数或属性。如果密钥的 ARN 发生更改，则您还必须更新使用该密钥的任何连接器所对应的参数。

   > **Note**
   >
   > 资源 ID 是客户提供的任意标识符。

3. 创建组版本，其中包含您要发送给核心的每个组件的目标版本。
4. 部署组版本。

有关演示如何创建和部署密钥资源、连接器与函数的教程，请参阅the section called “连接器入门 (CLI)”。

如果您在 Secrets Manager 中删除密钥，请从该组以及引用该密钥的连接器和 Lambda 函数中删除相应的密钥资源。否则，在组部署期间，AWS IoT Greengrass 会返回一条指示找不到密钥的错误。此外，您还可以根据需要更新 Lambda 函数代码。您可以部署不包含对应密钥资源的资源定义版本，以删除本地密钥。

在连接器和 Lambda 函数中使用本地密钥

Greengrass 连接器和 Lambda 函数使用本地密钥与服务和应用程序交互。系统默认使用 `AWSCURRENT` 值，但密钥资源中包含的其他暂存标签的值也可以使用。

连接器和函数必须在其可访问本地密钥之前进行配置。这样可以将密钥资源与连接器或函数关联。

连接器

如果连接器需要访问本地密钥，则连接器会提供一些参数，您通过这些参数配置连接器访问密钥所需的信息。

- 要了解如何在 AWS IoT Greengrass 控制台中执行此操作，请参阅 the section called “连接器入门（控制台）”。
- 要了解如何通过 AWS IoT Greengrass CLI 执行此操作，请参阅 the section called “连接器入门 (CLI)”。

有关各个连接器的要求的信息，请参阅the section called “AWS 提供的 Greengrass 连接器”。

用于访问和使用密钥的逻辑内置在连接器中。

Lambda 函数

要允许 Greengrass Lambda 函数访问本地密钥，您需配置该函数的属性。

- 要了解如何在 AWS IoT Greengrass 控制台中执行此操作，请参阅 the section called “如何创建密钥资源（控制台）”。
- 要通过 AWS IoT Greengrass API 执行此操作，您需在 `ResourceAccessPolicies` 属性中提供以下信息。
  - `ResourceId`：Greengrass 组中密钥资源的 ID。这是引用对应 Secrets Manager 密钥 ARN 的资源。
  - `Permission`：函数对资源的访问权限类型。密钥资源仅支持 `ro`（只读）权限。

以下示例将创建可以访问 `MyApiKey` 密钥资源的 Lambda 函数。

```
aws greengrass create-function-definition --name MyGreengrassFunctions --initial-
version '{
    "Functions": [
        {
            "Id": "MyLambdaFunction",
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:1",
            "FunctionConfiguration": {
                "Pinned": false,
                "MemorySize": 16384,
                "Timeout": 10,
                "Environment": {
                    "ResourceAccessPolicies": [
                        {
                            "ResourceId": "MyApiKey",
                            "Permission": "ro"
                        }
                    ],
                    "AccessSysfs": true
                }
            }
        }
    ]
}'
```

为在运行时访问本地密钥，Greengrass Lambda 函数从 AWS IoT Greengrass 核心开发工具包（v1.3.0 或更高版本）中的 `secretsmanager` 客户端调用 `get_secret_value` 函数。

下面的示例介绍如何使用适用于 Python 的 AWS IoT Greengrass 核心开发工具包获取密钥。它将密钥名称传递给 `get_secret_value` 函数。`SecretId` 可以是 Secrets Manager 密钥（不是密钥资源）的名称或 ARN。

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-MySecret-abc"


def function_handler(event, context):
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret = response.get("SecretString")
```

对于文本类型密钥，`get_secret_value` 函数返回一个字符串。对于二进制类型密钥，它返回一个采用 base64 编码的字符串。

> **Important**
>
> 请确保用户定义的 Lambda 函数能够安全地处理密钥，并且不记录存储在密钥中的任何敏感数据。有关更多信息，请参阅 AWS Secrets Manager 用户指南中的降低记录和调试 Lambda 函数的风险。尽管本文档特别提到了轮换函数，但该建议也适用于 Greengrass Lambda 函数。

默认情况下，返回密钥的当前值。这是 `AWSCURRENT` 暂存标签附加到的版本。要访问不同版本，请将相应暂存标签的名称传递给可选的 `VersionStage` 参数。例如：

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-TestSecret"
```

```
secret_version = "MyTargetLabel"


# Get the value of a specific secret version
def function_handler(event, context):
    response = secrets_client.get_secret_value(
        SecretId=secret_name, VersionStage=secret_version
    )
    secret = response.get("SecretString")
```

有关调用 `get_secret_value` 的另一个示例函数，请参阅创建 Lambda 函数部署程序包。

如何创建密钥资源（控制台）

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

本教程介绍如何使用AWS Management Console将密钥资源添加到 Greengrass 组。密钥资源是对 AWS Secrets Manager 中密钥的引用。有关更多信息，请参阅将密钥部署到核心。

在AWS IoT Greengrass 核心设备上，连接器和 Lambda 函数可以使用密钥资源完成服务和应用程序的身份验证，而无需对密码、令牌或其他凭证进行硬编码。

在本教程中，您需要先在 AWS Secrets Manager 控制台中创建一个密钥。然后，在 AWS IoT Greengrass 控制台中，从 Greengrass 组的资源页面向该组添加一个密钥资源。此密钥资源引用 Secrets Manager 密钥。稍后，将此密钥资源附加到一个 Lambda 函数，这将允许该函数获取本地密钥的值。

> **Note**
>
> 或者，控制台允许您在配置连接器或 Lambda 函数时创建密钥和密钥资源。您可以从连接器的配置参数页面或 Lambda 函数的资源页面执行此操作。
> 只有包含密钥参数的连接器才可以访问密钥。有关介绍 Twilio 通知连接器如何使用本地存储的身份验证令牌的教程，请参阅the section called “连接器入门（控制台）”。

本教程包含以下概括步骤：

1. 创建 Secrets Manager 密钥

2. 将密钥资源添加到组
3. 创建 Lambda 函数部署程序包
4. 创建 Lambda 函数
5. 将函数添加到组
6. 将密钥资源附加到函数
7. 将订阅添加到组
8. 部署组
9. the section called “测试 Lambda 函数”

完成本教程大约需要 20 分钟。

先决条件

要完成此教程，需要：

- Greengrass 组和 Greengrass Core（v1.7 或更高版本）。要了解如何创建 Greengrass 组和核心，请参阅 入门 AWS IoT Greengrass。“入门”教程还包含用于安装 AWS IoT Greengrass Core 软件的步骤。
- AWS IoT Greengrass 必须配置为支持本地密钥。有关更多信息，请参阅密钥要求。

> **Note**
>
> 此要求包括允许访问您的 Secrets Manager 密钥。如果使用的是默认 Greengrass 服务角色，则 Greengrass 有权获得名称以 greengrass- 开头的密钥的值。

- 要获取本地密钥的值，用户定义的 Lambda 函数必须使用 AWS IoT Greengrass Core SDK 1.3.0 或更高版本。

步骤 1：创建 Secrets Manager 密钥

在本步骤中，使用 AWS Secrets Manager 控制台创建密钥。

1. 登录到 AWS Secrets Manager 控制台。

> **Note**
>
> 有关此过程的更多信息，请参阅《AWS Secrets Manager 用户指南》中的步骤 1：在 AWS Secrets Manager 中创建和存储密钥。

2. 选择存储新密钥。
3. 在选择密钥类型下，选择其他密钥类型。
4. 在指定要为此密钥存储的键值对下：
   - 对于键，输入 **`test`**。
   - 对于值，输入 **`abcdefghi`**。
5. 为加密密钥保留选中 aws/secretsmanager，然后选择下一步。

> **Note**
>
> 如果使用 Secrets Manager 在您的账户中创建的默认 AWS 托管密钥，AWS KMS 不会对您收费。

6. 对于密钥名称，输入 **`greengrass-TestSecret`**，然后选择下一步。

> **Note**
>
> 默认情况下，Greengrass 服务角色允许 AWS IoT Greengrass 获取名称以 greengrass- 开头的密钥的值。有关更多信息，请参阅密钥要求。

7. 本教程不需要轮换，因此请选择禁用自动轮换，然后选择下一步。
8. 在审核页上，审核您的设置，然后选择存储。

接下来，在 Greengrass 组中创建一个引用该密钥的密钥资源。

步骤 2：将密钥资源添加到 Greengrass 组

在系步骤中，配置一个引用 Secrets Manager 密钥的组资源。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择要将密钥资源添加到的组。

3. 在组配置页面上，选择资源选项卡，然后向下滚动到密钥部分。密钥部分显示从属于该组的密钥资源。您可以在此部分中添加、编辑和删除密钥资源。

   > **Note**
   >
   > 或者，控制台允许您在配置连接器或 Lambda 函数时创建密钥和密钥资源。您可以从连接器的配置参数页面或 Lambda 函数的资源页面执行此操作。

4. 在密钥部分下选择添加。
5. 在添加秘密资源页面，在资源名称中输入 **`MyTestSecret`**。
6. 在密钥下，选择 GreenGrass-testSecret。
7. 在选择标签（可选）部分中，AWSCURRENT 暂存标签表示密钥的最新版本。该标签始终包含在密钥资源中。

   > **Note**
   >
   > 本教程只需要 AWSCURRENT 标签。您可以视情况包括 Lambda 函数或连接器需要的标签。

8. 选择添加资源。

步骤 3：创建 Lambda 函数部署包

要创建 Lambda 函数，您必须先创建一个包含函数代码和依赖项的 Lambda 函数部署包。Greengrass Lambda 函数需要 AWS IoT Greengrass Core 软件开发工具包来执行各项任务，例如在核心环境中与 MQTT 消息通信和访问本地机密等。本教程将创建一个 Python 函数，因此您需要在部署包中使用 Python 版本的软件开发工具包。

> **Note**
>
> 要获取本地密钥的值，用户定义的 Lambda 函数必须使用 AWS IoT Greengrass Core SDK 1.3.0 或更高版本。

1. 从 AWS IoT Greengrass Core 软件开发工具包下载页面，将适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包 下载到您的计算机上。
2. 解压缩下载的程序包以获取软件开发工具包。软件开发工具包是 `greengrasssdk` 文件夹。

3. 将以下 Python 代码函数保存在名为 `secret_test.py` 的本地文件中。

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
iot_client = greengrasssdk.client("iot-data")
secret_name = "greengrass-TestSecret"
send_topic = "secrets/output"


def function_handler(event, context):
    """
    Gets a secret and publishes a message to indicate whether the secret was
    successfully retrieved.
    """
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret_value = response.get("SecretString")
    message = (
        f"Failed to retrieve secret {secret_name}."
        if secret_value is None
        else f"Successfully retrieved secret {secret_name}."
    )
    iot_client.publish(topic=send_topic, payload=message)
    print("Published: " + message)
```

`get_secret_value` 函数支持对 `SecretId` 值使用 Secrets Manager 密钥的名称或 ARN。此示例使用密钥名称。对于此示例密钥，AWS IoT Greengrass 返回键值对：`{"test":"abcdefghi"}`。

> **Important**
>
> 请确保用户定义的 Lambda 函数能够安全地处理密钥，并且不记录存储在密钥中的任何敏感数据。有关更多信息，请参阅《AWS Secrets Manager 用户指南》中的降低记录和调试 Lambda 函数的风险。尽管本文档特别提到了轮换函数，但该建议也适用于 Greengrass Lambda 函数。

4. 将以下项目压缩到名为 `secret_test_python.zip` 的文件中。在创建 ZIP 文件时，应仅包含代码和依赖项，而不包含文件夹。

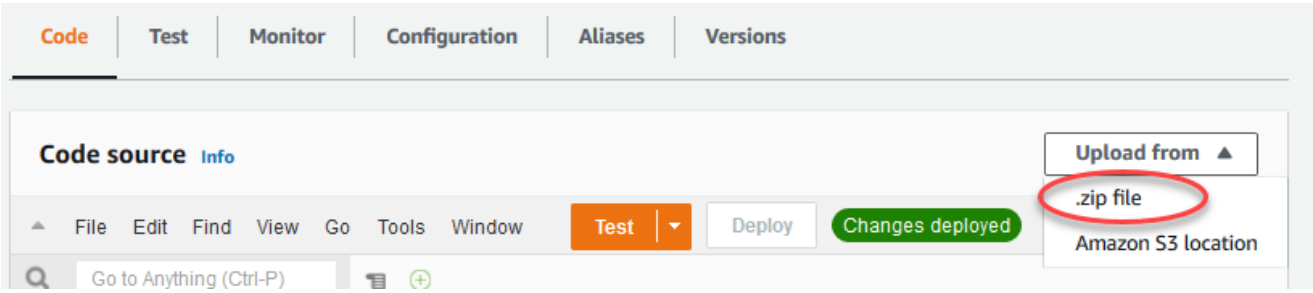
- secret_test.py。应用程序逻辑。
- greengrasssdk。所有 Python Greengrass Lambda 函数必需的库。

这是您的 Lambda 函数部署包。

步骤 4：创建 Lambda 函数

在此步骤中，您使用 AWS Lambda 控制台创建 Lambda 函数，然后将其配置为使用您的部署包。接着，发布函数版本并创建别名。

1. 首先，创建 Lambda 函数。
   a. 在AWS Management Console中，选择服务，然后打开 AWS Lambda 控制台。
   b. 选择创建函数，然后选择从头开始创作。
   c. 在基本信息部分中，使用以下值：
      - 对于函数名称，请输入 **`SecretTest`**。
      - 对于运行时系统，选择 Python 3.7。
      - 对于权限，请保留默认设置。这将创建一个授予基本 Lambda 权限的执行角色。此角色不由 AWS IoT Greengrass 使用。
   d. 在页面底部，选择创建函数。
2. 接下来，注册处理程序并上传您的 Lambda 函数部署包。
   a. 在代码选项卡上的代码源下，选择上传自。从下拉列表中选择 .zip 文件。



   b. 选择上传，然后选择您的 `secret_test_python.zip` 部署包。然后，选择保存。
   c. 在函数的代码选项卡中，在运行时设置下选择编辑，然后输入以下值。
      - 对于运行时系统，选择 Python 3.7。

- 对于处理程序，输入 **secret_test.function_handler**。

d. 选择保存。

> **Note**
>
> AWS Lambda 控制台上的测试按钮不适用于此功能。AWS IoT Greengrass Core 软件开发工具包不包含在 AWS Lambda 控制台中独立运行 Greengrass Lambda 函数所需的模块。这些模块（例如 `greengrass_common`）是在函数部署到您的 Greengrass 核心之后提供给它们的。

3. 现在，发布 Lambda 函数的第一个版本并创建此版本的别名。

> **Note**
>
> Greengrass 组可以按别名（推荐）或版本引用 Lambda 函数。使用别名，您可以更轻松地管理代码更新，因为您在更新函数代码时，不必更改订阅表或组定义。相反，您只需将别名指向新的函数版本。

a. 在操作菜单上，选择发布新版本。

b. 对于版本描述，输入 **First version**，然后选择发布。

c. 在 SecretTest: 1 配置页面上，从操作菜单中选择创建别名。

d. 在创建新别名页面上，使用以下值：

- 对于名称，输入 **GG_SecretTest**。
- 对于版本，选择 1。

> **Note**
>
> AWS IoT Greengrass 不支持 $LATEST 版本的 Lambda 别名。

e. 选择创建。

现在，您可以将 Lambda 函数添加到 Greengrass 组并附加密钥资源。

步骤 5：将 Lambda 函数添加到 Greengrass 组

在此步骤中，您要在 AWS IoT 控制台中将 Lambda 函数添加到 Greengrass 组。

1. 在组配置页面上，选择Lambda 函数选项卡。
2. 在我的 Lambda 函数部分下，选择添加。
3. 对于 Lambda 函数，请选择 SecretTest。
4. 对于 Lambda 函数版本，请选择您所发布的版本的别名。

接下来，配置 Lambda 函数的生命周期。

1. 在 Lambda 函数配置部分中，进行以下更新。

   > **Note**
   >
   > 我们建议您在不进行容器化的情况下运行 Lambda 函数，除非您的业务案例需要这样做。这有助于您在无需配置设备资源的前提下访问您的设备 GPU 和摄像头。如果您在没有容器化的情况下运行，则还必须为 AWS IoT Greengrass Lambda 函数授予根用户访问权限。

   a. 在不进行容器化的情况下运行：

      - 对于系统用户和群，请选择 **Another user ID/group ID**。在系统用户 ID 中，输入 **0**。在系统组 ID中，输入 **0**。

        这将允许您的 Lambda 函数以根用户身份运行。有关运行任务的更多信息，请参阅 the section called “为组中的 Lambda 函数设置默认访问身份”。

        > **Tip**
        >
        > 您还必须更新 `config.json` 文件，从而能够为 Lambda 函数授予根用户访问权限。有关此步骤，请参阅 the section called “以根用户身份运行 Lambda 函数”。

      - 对于 Lambda 函数容器化，请选择无容器。

        有关在无容器化情况下运行的更多信息，请参阅the section called “选择 Lambda 函数容器化时的注意事项”。

- 对于超时，输入 **10 seconds**。
- 对于已固定，选择 True。

  有关更多信息，请参阅the section called “生命周期配置”。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。

b. 改为在容器化模式下运行：

> **Note**
>
> 我们不建议以容器化模式运行，除非您的业务案例需要这样做。

- 对于系统用户和组，选择使用组默认值。
- 对于 Lambda 函数容器化，选择使用组默认值。
- 对于内存限制，输入 **1024 MB**。
- 对于超时，输入 **10 seconds**。
- 对于已固定，选择 True。

  有关更多信息，请参阅the section called “生命周期配置”。
- 在其他参数下，在对 sys 目录的只读权限中选择启用。

2. 选择添加 Lambda 函数。

接下来，将密钥资源与函数关联。

步骤 6：将密钥资源附加到 Lambda 函数

在此步骤中，将密钥资源附加到 Greengrass 组中的 Lambda 函数。这会将资源与函数关联，从而允许函数获取本地密钥的值。

1. 在组配置页面上，选择Lambda 函数选项卡。
2. 选择 SecretTest 函数。
3. 在函数的详细信息页面上，选择资源。
4. 滚动到密钥部分，然后选择关联。
5. 选择 MyTestSecret，然后选择关联。

步骤 7：将订阅添加到 Greengrass 组

在此步骤中，您将添加允许 AWS IoT 和 Lambda 函数交换消息的订阅。一个允许 AWS IoT 调用该函数的订阅，以及一个允许该函数向 AWS IoT 发送输出数据的订阅。

1. 在组配置页面中，选择订阅选项卡，然后选择添加订阅。
2. 创建一个允许 AWS IoT 向该函数发布消息的订阅：

   在组配置页面中，选择订阅选项卡，然后选择添加订阅。
3. 在创建订阅页面中，按如下所述配置源和目标：

   a. 在源类型中，选择 Lambda 函数，然后选择 IoT 云。

   b. 在目标类型中，选择服务，然后选择 SecretTest。

   c. 在主题筛选条件字段中，输入 **`secrets/input`**，然后选择订阅。
4. 添加另一个订阅。选择订阅选项卡，选择添加订阅，然后按如下所示配置源和目标：

   a. 在源类型中，选择服务，然后选择 SecretTest。

   b. 在目标类型中，选择 Lambda 函数，然后选择 IoT 云。

   c. 在主题筛选条件字段中，输入 **`secrets/output`**，然后选择订阅。

步骤 8：部署 Greengrass 组

将组部署到核心设备。在部署期间，AWS IoT Greengrass 从 Secrets Manager 中提取密钥的值，并在核心上创建一个本地加密副本。

1. 确保 AWS IoT Greengrass 核心正在运行。根据需要在您的 Raspberry Pi 终端中运行以下命令。

   a. 要检查进程守护程序是否正在运行，请执行以下操作：

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

   如果输出包含 `root` 的 `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon` 条目，则表示进程守护程序正在运行。

   > **Note**
   >
   > 路径中的版本取决于您的核心设备上安装的 AWS IoT Greengrass Core 软件版本。

b. 启动进程守护程序：

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

2. 在组配置页面上，选择部署。
3. a. 在 Lambda 函数选项卡的系统 Lambda 函数部分下，选择 IP 检测器，再选择编辑。

b. 在编辑 IP 检测器设置对话框中，选择自动检测和覆盖 MQTT 代理端点。

c. 选择保存。

这使得设备可以自动获取核心的连接信息，例如 IP 地址、DNS 和端口号。建议使用自动检测，不过 AWS IoT Greengrass 也支持手动指定的端点。只有在首次部署组时，系统才会提示您选择发现方法。

> **Note**
>
> 如果出现提示，请授予权限，以创建 Greengrass 服务角色并将其关联至当前 AWS 区域中的 AWS 账户。该角色将允许 AWS IoT Greengrass 访问您在 AWS 服务中的资源。

部署页面显示了部署时间戳、版本 ID 和状态。完成后，部署的状态应显示为成功完成。

有关问题排查帮助，请参阅排查问题。

测试 Lambda 函数

1. 在 AWS IoT 控制台主页上，选择测试。
2. 对于订阅主题，请使用以下值，然后选择订阅。

属性	值
订阅主题	密钥/输出
MQTT 负载显示	将负载显示为字符串

3. 对于发布，请使用以下值，然后选择发布以调用函数。

属性	值
主题	密钥/输入
消息	保留默认消息。发布消息将调用 Lambda 函数，但本教程中的这个函数不处理消息正文。

如果成功，则该函数会发布“成功”消息。

另请参阅

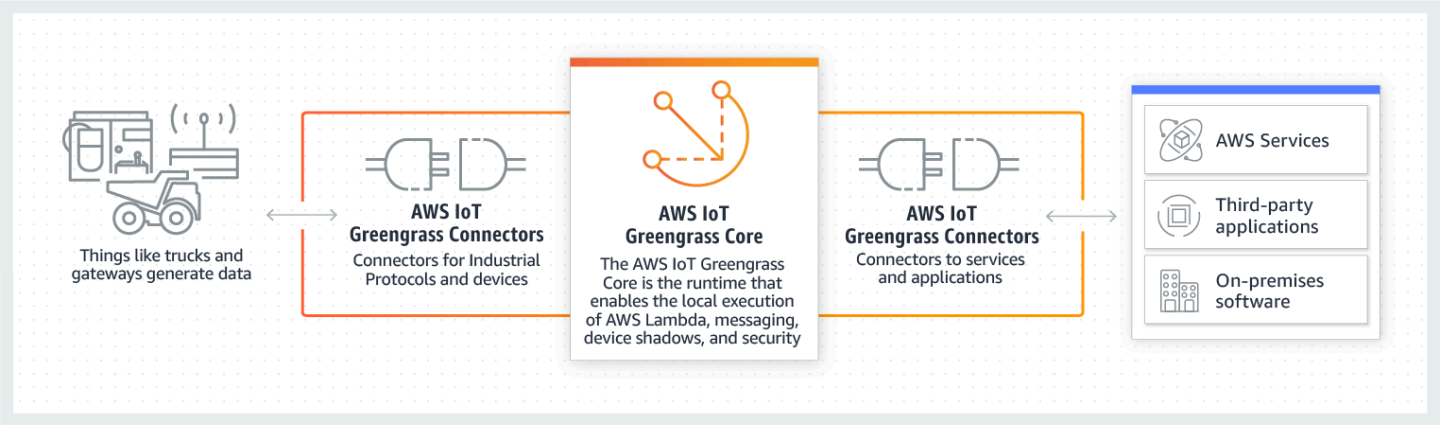
- 将密钥部署到核心

使用 Greengrass 连接器与服务和协议集成

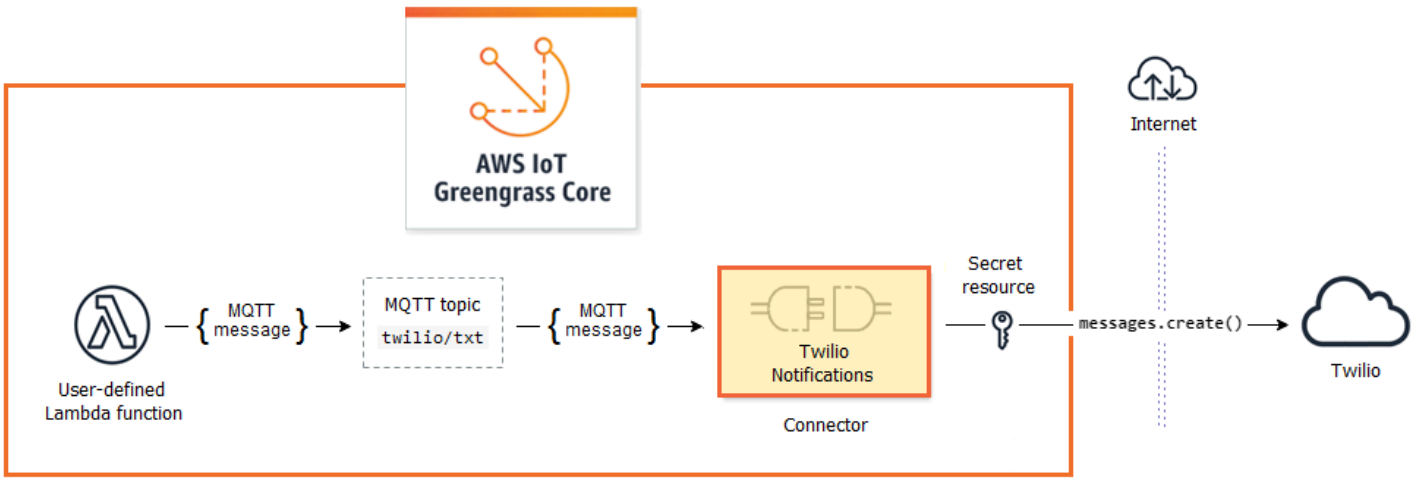
此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

AWS IoT Greengrass 中的连接器是一种预置模块，提供与本地基础设施、设备协议、AWS 和其他云服务的内置集成。借助连接器，您可以减少花在学习新协议和 API 上的时间，花更多的时间关注对您的业务十分重要的逻辑。

下图显示了连接器在AWS IoT Greengrass 格局中的作用。



许多连接器使用 MQTT 消息与组中的设备和Greengrass Lambda函数进行通信，或者与AWS IoT和本地影子服务进行通信。在以下示例中， 连接器从用户定义的 Lambda 函数接收 MQTT 消息，从 AWS Secrets Manager 使用密钥的本地引用，并调用 Twilio API。



有关创建此解决方案的教程，请参阅the section called “连接器入门（控制台）”和the section called “连接器入门 (CLI)”。

Greengrass 连接器可帮助您快速扩展设备功能或创建单一用途设备。通过使用连接器，您可以：

- 实施可重用的业务逻辑。
- 与云和本地服务（包括 AWS 和第三方服务）进行交互。
- 提取并处理设备数据。
- 使用 MQTT 主题订阅和用户定义的 Lambda 函数启用设备到设备调用。

AWS 提供了一组 Greengrass 连接器，可简化与常见服务和数据源的交互。这些预先构建的模块支持各种场景，其中包括日志记录和诊断，补充、行业数据处理以及警报和消息传递。有关更多信息，请参阅the section called “AWS 提供的 Greengrass 连接器”。

要求

要使用连接器，请记住以下几点：

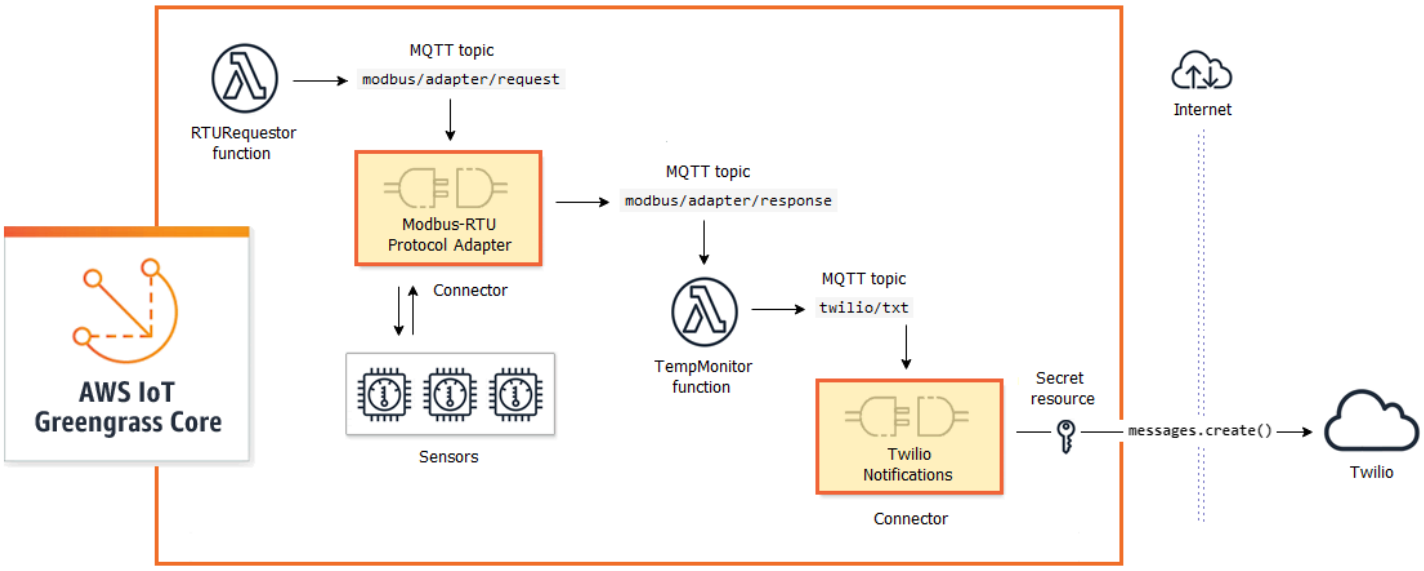
- 您必须满足要使用的每个连接器的要求。这些要求可能包括最低 AWS IoT Greengrass Core 软件版本、设备先决条件、所需权限和限制。有关更多信息，请参阅the section called “AWS 提供的 Greengrass 连接器”。
- Greengrass 组只能包含给定连接器的一个已配置实例。但是，可在多个订阅中使用该实例。有关更多信息，请参阅the section called “配置参数”。
- 当 Greengrass 组的默认容器化设置为无容器时，组中的连接器必须在不执行容器化的情况下运行。要查找支持无容器模式的连接器，请参阅the section called “AWS 提供的 Greengrass 连接器”。

使用 Greengrass 连接器

连接器是一种组组件。与其他组组件（如设备和用户定义的 Lambda 函数）一样，您可以向组添加连接器，配置其设置，并将它们部署到 AWS IoT Greengrass。连接器在核心环境中运行。

有些连接器可部署为简单的独立应用程序。例如，Device Defender 连接器从核心设备读取系统指标，并将其发送到 AWS IoT Device Defender 以供分析。

其他连接器可用作较大解决方案中的构建块。以下示例解决方案使用 Modbus-RTU Protocol Adapter 连接器处理来自传感器的消息，并使用 Twilio Notifications 连接器触发 Twilio 消息。



解决方案通常包含挨着连接器的用户定义的 Lambda 函数，并处理连接器发送或接收的数据。在此示例中，TempMonitor 函数从 Modbus-RTU 协议适配器接收数据，运行某一业务逻辑，然后将数据发送到 Twilio Notifications。

要创建和部署解决方案，您需要遵循以下一般过程：

1. 标出高级数据流。标识您需要使用的数据源、数据通道、服务、协议和资源。在示例解决方案中，这包含 Modbus RTU 协议、物理 Modbus 串行端口和 Twilio 上的数据。
2. 标识要包含在解决方案中的连接器，并将其添加到您的组。示例解决方案使用 Modbus-RTU 协议适配器和 Twilio Notifications。要帮助找到适用于您场景的连接器并了解其各自的要求，请参阅 the section called “AWS 提供的 Greengrass 连接器”。
3. 确定是需要用户定义的 Lambda 函数、设备还是资源，然后创建它们并将其添加到组。这可能包括一些函数，这些函数包含业务逻辑，或将数据处理为解决方案中其他实体所需的格式。该示例使用函数来发送解决方案 Modbus RTU 请求并启动 Twilio 通知。它还包括 Modbus RTU 串行端口的本地设备资源以及 Twilio 身份验证令牌的密钥资源。

> **Note**
>
> 密钥资源引用 AWS Secrets Manager 中的密码、令牌及其他密钥。连接器和 Lambda 函数可使用密钥对服务和应用程序进行身份验证。默认情况下，AWS IoT Greengrass 可访问其名称以“greengrass-”开头的密钥。有关更多信息，请参阅将密钥部署到核心。

4. 创建允许解决方案中的实体交换 MQTT 消息的订阅。如果在订阅中使用连接器，则该连接器以及消息源或目标必须使用该连接器支持的预定义主题语法。有关更多信息，请参阅the section called “输入和输出”。
5. 将组部署到 Greengrass 核心。

有关创建和部署连接条件的信息，请参阅以下教程：

- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”

配置参数

许多连接器提供了可让您自定义行为或输出的参数。这些参数在连接器生命周期中的初始化期间、运行时或其他时间使用。

参数类型和用法因连接器而异。例如，SNS 连接器有一个参数用于配置默认的 SNS 主题；而 Device Defender 有一个参数用于配置数据采样率。

组版本可包含多个连接器，但一次只能包含给定连接器的一个实例。这意味着组中的每个连接器只能有一个有效配置。但是，连接器实例可以在组中的多个订阅中使用。例如，您可以创建允许许多设备将数据发送到 Kinesis Firehose 连接器的多个订阅。

用于访问组资源的参数

Greengrass 连接器使用组资源访问核心设备上的文件系统、端口、外围设备及其他本地资源。如果连接器需要访问组资源，则它会提供相关的配置参数。

组资源包括：

- 本地资源。Greengrass 核心设备上存在的目录、文件、端口、引脚和外围设备。
- 机器学习资源。机器学习模型在云中训练并部署到核心进行本地推理。
- 密钥资源。来自 AWS Secrets Manager 的密码、密钥、令牌或任意文本的本地加密副本。连接器可以安全访问这些本地密钥，并使用这些密钥对服务或本地基础设施进行身份验证。

例如，Device Defender 的参数允许访问主机 `/proc` 目录中的系统指标，而 Twilio Notifications 的参数允许访问本地存储的 Twilio 身份验证令牌。

更新连接器参数

在向 Greengrass 组添加连接器时配置参数。您可以在添加连接器后更改参数值。

- 在控制台中：从组配置页面上，打开 Connectors (连接器)，然后从连接器的上下文菜单中选择 Edit (编辑)。

  > **Note**
  >
  > 如果连接器使用的某一密钥资源后来更改为引用其他密钥，您必须编辑连接器的参数并确认这一更改。

- 在 API 中：创建连接器的另一个版本，用于定义新的配置。

  AWS IoT Greengrass API 使用版本来管理群组。版本是不可变的，因此要添加或更改组组件（例如群组的客户端设备、函数和资源），必须创建新的或更新的组件的版本。然后，创建并部署组版本，其中包含每个组件的目标版本。

对连接器配置进行更改后，必须部署该组以将更改传播到核心。

输入和输出

许多 Greengrass 连接器可通过发送和接收 MQTT 消息与其他实体进行通信。MQTT 通信由订阅控制，这些订阅允许连接器与 Greengrass 组中的 Lambda 函数、设备以及其他连接器交换数据，或者与 AWS IoT 和本地影子服务交换数据。要允许此通信，您必须在该连接器所属的组中创建订阅。有关更多信息，请参阅the section called “MQTT 消息传递工作流中的托管订阅”。

连接器可以是消息发布者和/或消息订阅者。每个连接器都定义了它发布到或订阅的 MQTT 主题。这些预定义的主题必须在连接器是其消息源或消息目标的订阅中使用。有关包含为连接器配置订阅的步骤的教程，请参阅the section called “连接器入门（控制台）”和the section called “连接器入门 (CLI)”。

> **Note**
>
> 许多连接器还具有内置的通信模式以便与云或本地服务进行交互。这些通信模式因连接器而异，可能要求您为组角色配置参数或添加权限。有关连接器要求的信息，请参阅 the section called “AWS 提供的 Greengrass 连接器”。

输入主题

大多数连接器接收有关 MQTT 主题的输入数据。有些连接器订阅输入数据的多个主题。例如，Serial Stream 连接器支持两个主题：

- `serial/+/read/#`
- `serial/+/write/#`

对于此连接器，读取和写入请求将发送到相应的主题。创建订阅时，请确保使用符合您的实施的主题。

前面示例中的 + 和 # 字符为通配符。这些通配符允许订阅者接收多个主题上的消息，并允许发布者自定义他们发布到的主题。

- + 通配符可出现主题层次结构中的任何位置。它可以替换为某个层次结构项目。

  例如，对于主题 `sensor/+/input`，消息可发布到主题 `sensor/id-123/input`，但不能发布到 `sensor/group-a/id-123/input`。

- # 通配符只能出现在主题层次结构末端。它可以替换为零个或零个以上的层次结构项目。

  例如，对于主题 `sensor/#`，消息可发布到 `sensor/`, `sensor/id-123` 和 `sensor/group-a/id-123`，但不能发布到 `sensor`。

仅当订阅主题时，通配符才有效。消息不能发布到包含通配符的主题。请检查连接器的文档以了解其输入或输出主题要求。有关更多信息，请参阅the section called “AWS 提供的 Greengrass 连接器”。

对容器化的支持

默认情况下，大多数连接器在由 AWS IoT Greengrass 管理的隔离运行时环境中的 Greengrass 核心上运行。这些运行时环境称为容器，它们可以将连接器和主机系统分隔开，从而为主机和连接器提供更高的安全性。

但在某些情况下不支持这种 Greengrass 容器化，例如在 Docker 容器或没有 cgroup 的旧版 Linux 内核上运行 AWS IoT Greengrass 时。在这些情况下，连接器必须在无容器模式下运行。要查找支持无容器模式的连接器，请参阅the section called “AWS 提供的 Greengrass 连接器”。有些连接器本身就在这种模式下运行，而有些连接器则允许您设置隔离模式。

您也可以在支持 Greengrass 容器化的环境中将隔离模式设置为无容器模式，但我们建议尽可能使用 Greengrass 容器模式。

> **Note**
>
> Greengrass 组的默认容器化设置不适用于连接器。

升级连接器版本

连接器提供商可能会发布新版本的连接器，以添加功能、修复问题或提高性能。有关可用版本和相关更改的信息，请参阅每个连接器的文档。

在 AWS IoT 控制台中，您可以检查 Greengrass 组中连接器的新版本。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 在 Greengrass 组下，选择您的组。
3. 选择 Connectors (连接器) 以显示组中的连接器。

   如果连接器有新版本，则升级列中会显示可用按钮。
4. 要升级连接器版本，请执行以下操作：

   a. 在 Connectors (连接器) 页上的 Upgrade (升级) 列中，选择 Available (可用)。此时将打开 Upgrade connector (升级连接器) 页面，并显示当前参数设置（如果适用）。

      选择新的连接器版本，根据需要定义参数，然后选择 Upgrade (升级)。

   b. 在 Subscriptions (订阅) 页面上，在组中添加新订阅，以替换使用连接器作为源或目标的任何订阅。然后，删除旧订阅。

      订阅按版本引用连接器，因此，如果您更改组中的连接器版本，则这些连接器将变为无效。

   c. 从 Actions (操作) 菜单中，选择 Deploy (部署) 以将更改部署到核心。

要从 AWS IoT Greengrass API 升级连接器，请创建并部署包含更新的连接器和订阅的组版本。使用与向组添加连接器时相同的过程。有关演示如何使用 AWS CLI 配置和部署示例 Twilio Notifications 连接器的详细步骤，请参阅the section called “连接器入门 (CLI)”。

连接器的日志记录

Greengrass 包含向 Greengrass 日志写入事件和错误的 Lambda 函数。根据您的组设置，日志将写入到 CloudWatch 日志、本地文件系统或两者。来自连接器的日志包含相应函数的 ARN。下面的示例 ARN 来自 Kinesis Firehose 连接器。

```
arn:aws:lambda:aws-region:account-id:function:KinesisFirehoseClient:1
```

默认日志记录配置使用以下目录结构将信息级别日志写入到文件系统：

```
greengrass-root/ggc/var/log/user/region/aws/function-name.log
```

有关 Greengrass 日志记录的更多信息，请参阅 the section called “利用 AWS IoT Greengrass 日志进行监控”。

AWS 提供的 Greengrass 连接器

AWS 提供了以下支持常见 AWS IoT Greengrass 场景的连接器。有关连接器工作原理的更多信息，请参阅以下文档：

- 使用连接器与服务和协议集成
- 连接器入门（控制台） 或 连接器入门 (CLI)

Connector	描述	支持的 Lambda 运行时系统	支持无容器模式
CloudWatch 指标	向 Amazon 发布自定义指标 CloudWatch。	• Python 3.8 * • Python 3.7 • Python 2.7	是
Device Defender	将系统指标发送到 AWS IoT Device Defender。	• Python 3.8 * • Python 3.7 • Python 2.7	否

Connector	描述	支持的 Lambda 运行时系统	支持无容器模式
Docker 应用程序部署	运行 Docker Compose 文件以在核心设备上启动 Docker 应用程序。	• Python 3.8 • Python 3.7	是
IoT Analytics	将数据从设备和传感器发送到 AWS IoT Analytics。	• Python 3.8 * • Python 3.7 • Python 2.7	是
IoT 以太网 IP 协议适配器	从以太网/IP 设备收集数据。	• Java 8	是
物联网 SiteWise	将数据从设备和传感器发送到 AWS IoT SiteWise 中的资产属性。	• Java 8	是
Kinesis Firehose	将数据发送到 Amazon Data Firehose 传送流。	• Python 3.8 * • Python 3.7 • Python 2.7	是
机器学习反馈	将机器学习模型输入发布到云中，并将输出发布到 MQTT 主题。	• Python 3.8 • Python 3.7	否
ML 图像分类	运行本地图像分类推理服务。此连接器提供了面向多个平台的版本。	• Python 3.8 * • Python 3.7 • Python 2.7	否
ML 对象检测	运行本地对象检测推理服务。此连接器提供了面向多个平台的版本。	• Python 3.8 • Python 3.7	否

Connector	描述	支持的 Lambda 运行时系统	支持无容器模式
modbus-RTU 协议适配器	将请求发送到 Modbus RTU 设备。	• Python 3.8 * • Python 3.7 • Python 2.7	否
modbus-TCP 协议适配器	从 ModbusTCP 设备收集数据。	• Java 8	是
Raspberry Pi GPIO	控制 Raspberry Pi 核心设备上的 GPIO pin。	• Python 3.8 * • Python 3.7 • Python 2.7	否
串行流	读取和写入核心设备上的串行端口。	• Python 3.8 * • Python 3.7 • Python 2.7	否
ServiceNow MetricBase 集成	将时间序列指标发布到 ServiceNow MetricBase。	• Python 3.8 * • Python 3.7 • Python 2.7	是
SNS	向 Amazon SNS 主题发送消息。	• Python 3.8 * • Python 3.7 • Python 2.7	是
Splunk 集成	将数据发布到 Splunk HEC。	• Python 3.8 * • Python 3.7 • Python 2.7	是

Connector	描述	支持的 Lambda 运行时系统	支持无容器模式
Twilio 通知	启动 Twilio 文本或语音消息。	• Python 3.8 * • Python 3.7 • Python 2.7	是

* 要使用 Python 3.8 运行时，必须创建一个符号链接，从默认 Python 3.7 安装文件夹指向已安装的 Python 3.8 二进制文件。有关更多信息，请参阅特定于连接器的要求。

> **Note**
>
> 我们建议您将连接器版本从 Python 2.7 升级到 Python 3.7。能否继续支持 Python 2.7 连接器取决于 AWS Lambda 运行时支持。有关更多信息，请参阅 AWS Lambda 开发人员指南中的运行时支持策略。

CloudWatch 指标连接器

CloudWatch 指标连接器将来自 Greengrass 设备的自定义指标发布到亚马逊。 CloudWatch该连接器提供了一个用于发布 CloudWatch 指标的集中式基础架构，可用于监控和分析 Greengrass 核心环境，并对本地事件采取行动。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。

此连接器以 MQTT 消息形式接收指标数据。连接器会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 。

此连接器具有以下版本。

版本	ARN
5	`arn:aws:greengrass:  region::/connectors/CloudWatchMetrics/versions/5`

版本	ARN
4	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/4`
3	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/3`
2	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/2`
1	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 3 - 5

- AWS IoT Greengrass Core 软件 v1.9.3 版或更高版本。
- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

> **Note**
>
> 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- Greengrass 组角色，配置为允许执行 `cloudwatch:PutMetricData` 操作，如以下示例 AWS Identity and Access Management IAM 策略中所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

有关 CloudWatch 权限的更多信息，请参阅 IA M 用户指南中的 A mazon CloudWatch 权限参考。

Versions 1 - 2

- AWS IoT Greengrass Core 软件 v1.7 版或更高版本。
- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- Greengrass 组角色，配置为允许执行 `cloudwatch:PutMetricData` 操作，如以下示例 AWS Identity and Access Management IAM 策略中所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
```

```
        }
    ]
}
```

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

有关 CloudWatch 权限的更多信息，请参阅 IA M 用户指南中的 A mazon CloudWatch 权限参考。

连接器参数

该连接器提供以下参数：

Versions 4 - 5

`PublishInterval`

发布给定命名空间的批处理指标之前等待的最长秒数。最大值为 900。要将连接器配置为一收到指标便发布（而不进行批处理），请指定 0。

连接器在同一命名空间中收到 20 个指标后或在指定的间隔之后发布到。 CloudWatch

> **Note**
>
> 连接器不保证发布事件的顺序。

AWS IoT 控制台中的显示名称：发布间隔

必需：`true`

类型：`string`

有效值：`0 - 900`

有效模式：`[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

AWS 区域要发布 CloudWatch 指标的。该值会覆盖默认 Greengrass 指标区域。该值仅在发布跨区域指标时是必需的。

AWS IoT 控制台中的显示名称：发布区域

必需：`false`

类型：`string`

有效模式：`^$|([a-z]{2}-[a-z]+-\d{1})`

`MemorySize`

要分配给连接器的内存 (KB)。

AWS IoT 控制台中的显示名称：内存大小

必需：`true`

类型：`string`

有效模式：`^[0-9]+$`

`MaxMetricsToRetain`

替换为新指标之前所有命名空间内可在内存中保存的指标的最大数量。最小值为 2000。

此限制适用于没有 Internet 连接并且连接器稍后开始缓冲要发布的指标的情况。当缓冲区已满时，最旧的指标将被新的指标所替换。给定命名空间中的指标仅被同一命名空间中的指标所替换。

> **Note**
>
> 如果连接器的宿主进程中断，则不会保存指标。例如，在组部署或设备重新启动期间，可能会发生此中断。

AWS IoT 控制台中的显示名称：可保留的最大指标数

必需：`true`

类型：`string`

有效模式：`^([2-9]\d{3}|[1-9]\d{4,})$`

`IsolationMode`

此连接器的容器化模式。默认值为 `GreengrassContainer`，这意味着连接器在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

> **Note**
>
> 组的默认容器化设置不适用于连接器。

AWS IoT 控制台中的显示名称：容器隔离模式

必需：`false`

类型：`string`

有效值：`GreengrassContainer` 或 `NoContainer`

有效模式：`^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

`PublishInterval`

发布给定命名空间的批处理指标之前等待的最长秒数。最大值为 900。要将连接器配置为一收到指标便发布（而不进行批处理），请指定 0。

连接器在同一命名空间中收到 20 个指标后或在指定的间隔之后发布到。 CloudWatch

> **Note**
>
> 连接器不保证发布事件的顺序。

AWS IoT 控制台中的显示名称：发布间隔

必需：`true`

类型：`string`

有效值：`0 - 900`

有效模式：`[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

AWS 区域要发布 CloudWatch 指标的。该值会覆盖默认 Greengrass 指标区域。该值仅在发布跨区域指标时是必需的。

AWS IoT 控制台中的显示名称：发布区域

必需：`false`

类型：`string`

有效模式：`^$|([a-z]{2}-[a-z]+-\d{1})`

`MemorySize`

要分配给连接器的内存 (KB)。

AWS IoT 控制台中的显示名称：内存大小

必需：`true`

类型：`string`

有效模式：`^[0-9]+$`

`MaxMetricsToRetain`

替换为新指标之前所有命名空间内可在内存中保存的指标的最大数量。最小值为 2000。

此限制适用于没有 Internet 连接并且连接器稍后开始缓冲要发布的指标的情况。当缓冲区已满时，最旧的指标将被新的指标所替换。给定命名空间中的指标仅被同一命名空间中的指标所替换。

> **Note**
>
> 如果连接器的宿主进程中断，则不会保存指标。例如，在组部署或设备重新启动期间，可能会发生此中断。

AWS IoT 控制台中的显示名称：可保留的最大指标数

必需：`true`

类型：`string`

有效模式：^([2-9]\d{3}|[1-9]\d{4,})$

创建连接器示例 (AWS CLI)

以下 CLI 命令ConnectorDefinition使用包含 Metr CloudWatch ics 连接器的初始版本创建一个。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyCloudWatchMetricsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/CloudWatchMetrics/
versions/4",
            "Parameters": {
                "PublishInterval" : "600",
                "PublishRegion" : "us-west-2",
                "MemorySize" : "16",
                "MaxMetricsToRetain" : "2500",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅 the section called “连接器入门（控制台）”。

输入数据

此连接器接受 MQTT 主题的指标并将指标发布到。 CloudWatch输入消息必须采用 JSON 格式。

订阅中的主题筛选条件

`cloudwatch/metric/put`

消息属性

`request`

有关此消息中的指标的信息。

请求对象包含要发布到 CloudWatch 的指标数据。指标值必须符合 `PutMetricData` API 的规范。只有 `namespace`、`metricData.metricName` 和 `metricData.value` 属性是必需的。

必需：true

类型：包含以下属性的 object：

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。

必需：true

类型：string

有效模式：[^:].*

metricData

指标的数据。

必需：true

类型：包含以下属性的 object：

metricName

指标的名称。

必需：true

类型：string

dimensions

与指标关联的维度。维度提供有关指标及其数据的更多信息。指标最多可定义 10 个维度。

此连接器自动包含一个名为 coreName 的维度，其中的值是核心的名称。

必需：false

类型：维对象的 array 包含以下属性：

name

维度名称。

必需：false

类型：string

value

维度值。

必需：false

类型：string

timestamp

收到指标数据的时间，表示为自 Jan 1, 1970 00:00:00 UTC 以来的秒数。如果省略此值，则连接器将使用它收到消息的时间。

必需：false

类型：timestamp

> **Note**
>
> 如果您使用此连接器的版本 1 到版本 4，我们建议您在从单一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。

value

指标的值。

> **Note**
>
> CloudWatch 拒绝太小或太大的值。值必须在范围 8.515920e-109 到 1.174271e+108（以 10 为底的指数）内或范围 2e-360 到 2e360（以 2 为底的指数）内。不支持特殊值（例如 NaN、+Infinity、-Infinity）。

必需：true

类型：double

unit

指标的单位。

必需：false

类型：string

有效值：Seconds, Microseconds, Milliseconds, Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes, Bits, Kilobits, Megabits, Gigabits, Terabits, Percent, Count, Bytes/Second, Kilobytes/Second, Megabytes/Second, Gigabytes/Second, Terabytes/Second, Bits/Second, Kilobits/Second, Megabits/Second, Gigabits/Second, Terabits/Second, Count/Second, None

限制

使用此连接器时，CloudWatch PutMetricDataAPI 施加的所有限制都适用于指标。以下限制尤其重要：

- API 负载的 40 KB 限制
- 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 Amazon CloudWatch 用户指南中的CloudWatch 限制。

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

此连接器将状态信息发布为 MQTT 主题的输出数据。

订阅中的主题筛选条件

```
cloudwatch/metric/put/status
```

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"
    }
}
```

示例输出：失败

```
{
   "response" : {
        "namespace": "Greengrass",
        "error": "InvalidInputException",
        "error_message":"cw metric is invalid",
        "status":"fail"
   }
}
```

> **Note**
>
> 如果连接器检测到可重试的错误（如连接错误），它会在下一批次中重试发布。

用法示例

使用以下概括步骤设置可用于尝试连接器的示例 Python 3.7 Lambda 函数。

> **Note**
>
> - 如果要使用其他 Python 运行时，您可以创建从 Python3.x 到 Python 3.7 的符号链接。
> - 连接器入门（控制台）和 连接器入门 (CLI) 主题包含详细步骤，说明如何配置和部署示例 Twilio 通知连接器。

1. 确保满足连接器的要求。

   对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

2. 创建并发布将输入数据发送到连接器的 Lambda 函数。

   将示例代码保存为 PY 文件。下载并解压适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。然后，创建一个 zip 包，其中在根级别包含 PY 文件和 `greengrasssdk` 文件夹。此 zip 包是您上传到 AWS Lambda 的部署包。

   创建 Python 3.7 Lambda 函数后，请发布函数版本并创建别名。

3. 配置 Greengrass 组。

   a. 通过别名来添加 Lambda 函数（推荐）。将 Lambda 生命周期配置为长时间生存（或在 CLI 中设置为 `"Pinned": true`）。

   b. 添加连接器并配置其参数。

   c. 添加允许连接器接收输入数据并针对支持的主题筛选条件发送输出数据的订阅。

      - 将 Lambda 函数设置为源，将连接器设置为目标，并使用支持的输入主题筛选条件。
      - 将连接器设置为源，将 AWS IoT Core 设置为目标，并使用支持的输出主题筛选条件。您可以使用此订阅查看 AWS IoT 控制台中的状态消息。

4. 部署组。

5. 在 AWS IoT 控制台中的测试页面上，订阅输出数据主题以查看连接器中的状态消息。示例 Lambda 函数是长时间生存的，并且在部署组后立即开始发送消息。

完成测试后，您可以将 Lambda 生命周期设置为按需（或在 CLI 中设置为 `"Pinned": false`）并部署组。这会阻止函数发送消息。

示例

以下示例 Lambda 函数向连接器发送一条输入消息。

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'cloudwatch/metric/put'

def create_request_with_all_fields():
    return  {
        "request": {
            "namespace": "Greengrass_CW_Connector",
            "metricData": {
                "metricName": "Count1",
                "dimensions": [
                    {
                        "name": "test",
                        "value": "test"
                    }
                ],
                "value": 1,
                "unit": "Seconds",
                "timestamp": time.time()
            }
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
```

```
    return
```

许可证

CloudWatch 指标连接器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

此连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
5	进行了修复，以添加输入数据中对重复时间戳的支持。
4	增加了用于配置连接器容器化模式的 `IsolationMode` 参数。
3	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。
2	进行了修复，以减少过多的日志记录。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”
- 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

Device Defender 连接器

Device Defender 连接器向管理员通知 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。

此连接器从核心设备上的 `/proc` 目录中读取系统指标，然后将这些指标发布到 AWS IoT Device Defender。有关指标报告的详细信息，请参阅《AWS IoT 开发人员指南》中的设备指标文档规范。

此连接器具有以下版本。

版本	ARN
3	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/3`
2	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/2`
1	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 3

- AWS IoT Greengrass Core 软件 v1.9.3 版或更高版本。
- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

  > **Note**
  >
  > 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- AWS IoT Device Defender 配置为使用检测功能来跟踪冲突。有关更多信息，请参阅 AWS IoT 开发人员指南中的检测。
- Greengrass 组中指向 `/proc` 目录的一个本地卷资源。该资源必须使用以下属性：
  - 源路径：`/proc`
  - 目标路径：`/host_proc`（或一个与有效模式匹配的值）
  - AutoAddGroupOwner：`true`
- 安装在 Greengrass 核心上的 psutil 库。版本 5.7.0 是经验证可与连接器一起使用的最新版本。
- 安装在 Greengrass 核心上的 cbor 库。版本 1.0.0 是经验证可与连接器一起使用的最新版本。

Versions 1 - 2

- AWS IoT Greengrass Core 软件 v1.7 版或更高版本。
- Python 版本 2.7 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跟踪冲突。有关更多信息，请参阅 AWS IoT 开发人员指南中的检测。
- Greengrass 组中指向 `/proc` 目录的一个本地卷资源。该资源必须使用以下属性：
  - 源路径：`/proc`
  - 目标路径：`/host_proc`（或一个与有效模式匹配的值）
  - AutoAddGroupOwner：`true`
- 安装在 Greengrass 核心上的 psutil 库。

- 安装在 Greengrass 核心上的 cbor 库。

连接器参数

该连接器提供以下参数：

`SampleIntervalSeconds`

收集和报告指标的每个周期之间的秒数。最小值为 300 秒 (5 分钟)。

AWS IoT 控制台中的显示名称：指标报告间隔

必需：`true`

类型：`string`

有效模式：`^[0-9]*(?:3[0-9][0-9]|[4-9][0-9]{2}|[1-9][0-9]{3,})$`

`ProcDestinationPath-ResourceId`

`/proc` 卷资源的 ID。

> **Note**
> 此连接器被授予对该资源的只读访问权限。

AWS IoT 控制台中的显示名称：/proc 目录的资源

必需：`true`

类型：`string`

有效模式：`[a-zA-Z0-9_-]+`

`ProcDestinationPath`

`/proc` 卷资源的目标路径。

AWS IoT 控制台中的显示名称：/proc 资源的目标路径

必需：`true`

类型：`string`

有效模式：`\/[a-zA-Z0-9_-]+`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 Device Defender 连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyDeviceDefenderConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/DeviceDefender/
versions/3",
            "Parameters": {
                "SampleIntervalSeconds": "600",
                "ProcDestinationPath": "/host_proc",
                "ProcDestinationPath-ResourceId": "my-proc-resource"
            }
        }
    ]
}'
```

> **Note**
>
> 此连接器中的 Lambda 函数的生命周期很长。

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅the section called “连接器入门（控制台）”。

输入数据

该连接器不接受 MQTT 消息作为输入数据。

输出数据

此连接器将安全指标作为输出数据发布到 AWS IoT Device Defender。

订阅中的主题筛选条件

`$aws/things/+/defender/metrics/json`

> **Note**
>
> 这是 AWS IoT Device Defender 预期的主题语法。连接器将 + 通配符替换为设备名称（例如，`$aws/things/thing-name/defender/metrics/json`）。

输出示例

有关指标报告的详细信息，请参阅AWS IoT 开发人员指南中的设备指标文档规范。

```
{
    "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
            }
        }
    }
}
```

许可证

此连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
3	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。

版本	更改
2	进行了修复，以减少过多的日志记录。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”
- 《AWS IoT 开发人员指南》中的 Device Defender

Docker 应用程序部署连接器

Greengrass Docker 应用程序部署连接器可以更轻松地在 AWS IoT Greengrass Core 上运行 Docker 映像。连接器使用 Docker Compose 从 `docker-compose.yml` 文件启动多容器 Docker 应用程序。具体而言，连接器运行 `docker-compose` 命令来管理单个核心设备上的 Docker 容器。有关详细信息，请参阅 Docker 文档中的 Docker Compose 概述 。连接器可以访问存储在 Docker 容器镜像仓库中的 Docker 映像，如 Amazon Elastic Container Registry (Amazon ECR)、Docker Hub 和私有 Docker 信任的镜像仓库。

在部署 Greengrass 组之后，连接器将提取最新的映像并启动 Docker 容器。它运行 `docker-compose pull` 和 `docker-compose up` 命令。然后，连接器将命令的状态发布到输出 MQTT 主题。它还记录有关运行 Docker 容器的状态信息。这使您可以在 Amazon 中监控您的应用程序日志 CloudWatch。有关更多信息，请参阅 the section called “利用 AWS IoT Greengrass 日志进行监控”。连接器还会在每次 Greengrass 守护程序重新启动时启动 Docker 容器。可以在内核上运行的 Docker 容器数量取决于您的硬件。

Docker 容器在核心设备上的 Greengrass 域之外运行，因此它们无法访问核心的进程间通信 (IPC)。但是，您可以使用 Greengrass 组件配置某些通信通道，例如本地 Lambda 函数。有关更多信息，请参阅 the section called “与 Docker 容器通信”。

您可以将连接器用于在核心设备上托管 Web 服务器或 MySQL 服务器等此类方案。Docker 应用程序中的本地服务可以彼此、与本地环境中的其他进程以及云服务通信。例如，您可以在核心上运行 Web 服务器，将请求从 Lambda 函数发送到云中的 Web 服务。

此连接器在无容器隔离模式下运行，因此，您可以将其部署到在没有 Greengrass 容器化的情况下运行的 Greengrass 组。

此连接器具有以下版本。

版本	ARN
7	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/7`
6	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/6`
5	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/5`
4	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/4`
3	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/3`
2	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/2`
1	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

- AWS IoT Greengrass Core 软件 v1.10 版或更高版本。

  > **Note**
  >
  > OpenWrt 发行版不支持此连接器。

- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

  > **Note**
  >
  > 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- Greengrass 内核上至少有 36 MB RAM，供连接器监控正在运行的 Docker 容器。总内存需求取决于在内核上运行的 Docker 容器的数量。
- Greengrass 核心上安装的 Docker Engine 1.9.1 或更高版本。版本 19.0.3 是经验证可与连接器一起使用的最新版本。

  `docker` 可执行文件必须位于 `/usr/bin` 或 `/usr/local/bin` 目录中。

  > **Important**
  >
  > 我们建议您安装凭证存储以保护 Docker 凭证的本地副本。有关更多信息，请参阅 the section called “安全说明”。

  有关在 Amazon Linux 发行版上安装 Docker 的信息，请参阅《Amazon Elastic Container Service 开发人员指南》中的 Amazon ECS 的 Docker 基础知识。

- Greengrass 核心上安装的 Docker Compose。`docker-compose` 可执行文件必须位于 `/usr/bin` 或 `/usr/local/bin` 目录中。

  以下 Docker Compose 版本经验证可与此连接器一起使用。

连接器版本	经验证的 Docker Compose 版本
7	1.25.4
6	1.25.4
5	1.25.4
4	1.25.4
3	1.25.4
2	1.25.1
1	1.24.1

- 一个 Docker Compose 文件（例如，`docker-compose.yml`），存储在 Amazon Simple Storage Service (Amazon S3) 中。格式必须与安装在核心上的 Docker Compose 版本兼容。在核心上使用文件之前，应先对其进行测试。如果在部署 Greengrass 组后编辑该文件，则必须重新部署该组以更新核心上的本地副本。
- 有权调用本地 Docker 守护程序并写入存储 Compose 文件的本地副本的目录的 Linux 用户。有关更多信息，请参阅 在核心上设置 Docker 用户。
- Greengrass 组角色，配置为允许对包含 Compose 文件的 S3 存储桶执行 `s3:GetObject` 操作。此权限显示在以下示例 IAM 策略中。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToComposeFileS3Bucket",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Effect": "Allow",
```

```
            "Resource": "arn:aws:s3:::bucket-name/*"
        }
    ]
}
```

> **Note**
>
> 如果您的 S3 桶已启用版本控制，则必须将角色配置为也允许执行 `s3:GetObjectVersion` 操作。有关更多信息，请参阅 Amazon Simple Storage Service 用户指南中的使用版本控制。

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

- 如果您的 Docker Compose 文件引用 Amazon ECR 中存储的 Docker 映像，则为配置为允许执行以下操作的 Greengrass 组角色：
  - 对包含 Docker 镜像的 Amazon ECR 存储库执行的 `ecr:GetDownloadUrlForLayer` 和 `ecr:BatchGetImage` 操作。
  - 对您的资源执行的 `ecr:GetAuthorizationToken` 操作。

  存储库必须与连接器位于同一 AWS 账户 和 AWS 区域 中。

> **Important**
>
> 组角色中的权限可由 Greengrass 组中的所有 Lambda 函数和连接器代入。有关更多信息，请参阅 the section called “安全说明”。

以下示例策略中显示了这些权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGetEcrRepositories",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
```

```
            ],
            "Resource": [
                "arn:aws:ecr:region:account-id:repository/repository-name"
            ]
        },
        {
            "Sid": "AllowGetEcrAuthToken",
            "Effect": "Allow",
            "Action": "ecr:GetAuthorizationToken",
            "Resource": "*"
        }
    ]
}
```

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。

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

- 如果您的 Docker Compose 文件引用来自 AWS Marketplace 的 Docker 镜像，则连接器还具有以下要求：
  - 您必须订阅 AWS Marketplace 容器产品。有关更多信息，请参阅《AWS Marketplace 订阅者指南》中的查找和订阅容器产品。
  - AWS IoT Greengrass 必须配置为支持本地密钥，如密钥要求中所述。连接器仅使用此功能从 AWS Secrets Manager 中检索您的密钥，而不存储它们。
  - 您必须在 Secrets Manager 中为用于存储在 Compose 文件中引用的 Docker 镜像的每个 AWS Marketplace 镜像仓库创建一个密钥。有关更多信息，请参阅 the section called “从私有存储库访问 Docker 镜像”。
- 如果您的 Docker Compose 文件从 Amazon ECR（如 Docker Hub）之外的镜像仓库中的私有存储库引用 Docker 镜像，则连接器还具有以下要求：
  - AWS IoT Greengrass 必须配置为支持本地密钥，如密钥要求中所述。连接器仅使用此功能从 AWS Secrets Manager 中检索您的密钥，而不存储它们。
  - 您必须在 Secrets Manager 中为用于存储在 Compose 文件中引用的 Docker 镜像的每个私有存储库创建一个密钥。有关更多信息，请参阅 the section called “从私有存储库访问 Docker 镜像”。
- 当您部署包含此连接器的 Greengrass 组时，Docker 守护程序必须正在运行。

从私有存储库访问 Docker 镜像

如果您使用凭证问 Docker 镜像，则必须允许连接器访问它们。您执行此操作的方式取决于 Docker 镜像的位置。

对于 Amazon ECR 中存储的 Docker 镜像，您可以授予在 Greengrass 组角色中获取授权令牌的权限。有关更多信息，请参阅 the section called “要求”。

对于存储在其他私有存储库或镜像仓库中的 Docker 镜像，您必须在 AWS Secrets Manager 中创建一个密钥来存储您的登录信息。这包括您在 AWS Marketplace 中订阅的 Docker 镜像。为每个存储库创建一个密钥。如果您在 Secrets Manager 中更新您的密钥，则更改会在下次部署组时传播到核心。

> **Note**
>
> Secrets Manager 是一项可用于在 AWS Cloud 中安全存储和管理您的凭证、键和其他密钥的服务。有关更多信息，请参阅AWS Secrets Manager《用户指南》中的什么是AWS Secrets Manager？。

每个密钥必须包含以下键：

键	值
`username`	用于访问存储库或镜像仓库的用户名。
`password`	用于访问存储库或镜像仓库的密码。
`registryUrl`	镜像仓库的终端节点。这必须与 Compose 文件中相应的镜像仓库 URL 匹配。

> **Note**
>
> 要默认允许 AWS IoT Greengrass 访问密钥，密钥的名称必须以 greengrass- 开头。否则，您的 Greengrass 服务角色必须授予访问权限。有关更多信息，请参阅 the section called “允许 AWS IoT Greengrass 获取密钥值”。

从 AWS Marketplace 获取 Docker 镜像的登录信息

1. 使用 `aws ecr get-login-password` 命令从 AWS Marketplace 中获取 Docker 映像的密码。有关更多信息，请参阅《AWS CLI 命令参考》中的 get-login-password。

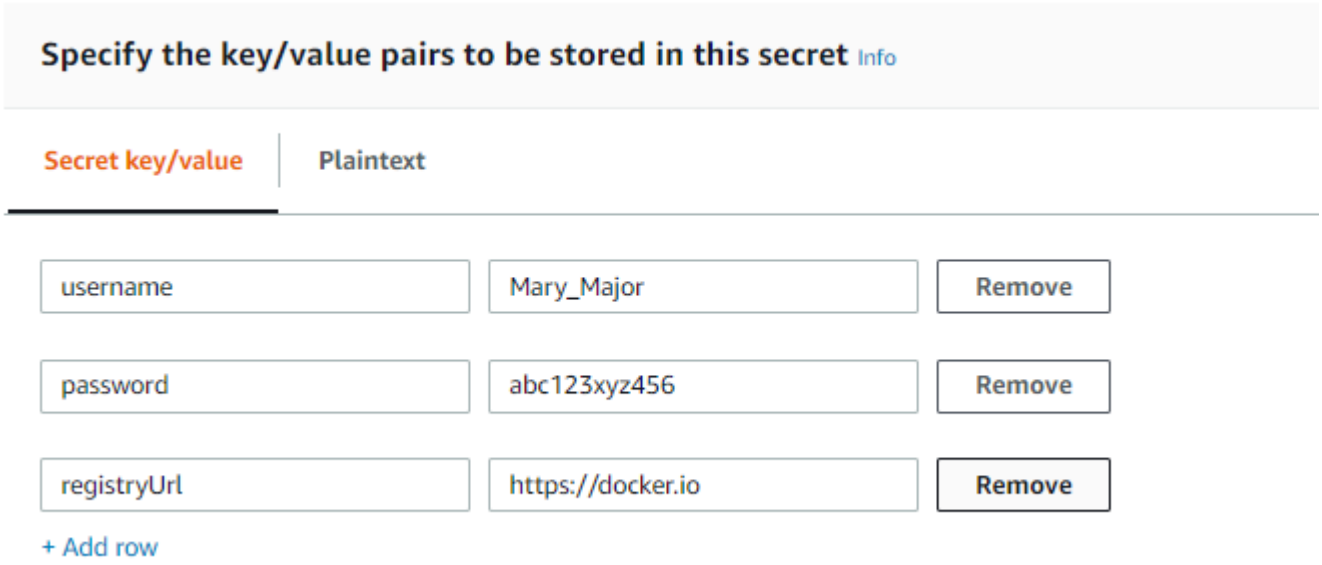
   ```
   aws ecr get-login-password
   ```

2. 检索 Docker 映像的注册表 URL。打开 AWS Marketplace 网站，然后打开容器产品启动页面。在容器映像下，选择查看容器映像详细信息以找到用户名和注册表 URL。

使用检索到的用户名、密码和注册表 URL 为每个 AWS Marketplace 注册表创建一个密钥，该注册表中存储了您的 Compose 文件中引用的 Docker 映像。

创建密钥（控制台）

在 AWS Secrets Manager 控制台中，选择其他密钥类型。在 Specify the key-value pairs to be stored for this secret (指定要为此密钥存储的键值对) 下，添加针对 `username`、`password` 和 `registryUrl` 的行。有关更多信息，请参阅 AWS Secrets Manager 用户指南中的创建基本密钥。



创建密钥 (CLI)

在 AWS CLI 中使用 Secrets Manager `create-secret` 命令，如下例所示。有关更多信息，请参阅 AWS CLI 命令参考中的 create-secret。

```
aws secretsmanager create-secret --name greengrass-MySecret --secret-string
 [{"username":"Mary_Major"},{"password":"abc123xyz456"},{"registryUrl":"https://
docker.io"}]
```

> **Important**
>
> 您有责任保护用于存储 Docker Compose 文件的`DockerComposeFileDestinationPath`目录以及来自私有存储库的 Docker 镜像的凭证。有关更多信息，请参阅 the section called “安全说明”。

参数

该连接器提供以下参数：

Version 7

`DockerComposeFileS3Bucket`

包含您的 Docker Compose 文件的 S3 存储桶的名称。创建桶时，请务必遵守《亚马逊简单存储服务用户指南》中描述的桶名称规则。

AWS IoT 控制台中显示名称：S3 中的 Docker Compose 文件

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`true`

类型：`string`

有效模式 `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Amazon S3 中的 Docker Compose 文件的对象密钥。有关更多信息（包括对象键命名指南），请参阅 Amazon Simple Storage Service 用户指南中的对象键和元数据。

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`true`

类型：`string`

有效模式 `.+`

`DockerComposeFileS3Version`

Amazon S3 中的 Docker Compose 文件的对象版本。有关更多信息（包括对象键命名指南），请参阅 Amazon Simple Storage Service 用户指南中的使用版本控制。

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`false`

类型：`string`

有效模式 `.+`

`DockerComposeFileDestinationPath`

用于存储 Docker Compose 文件副本的本地目录的绝对路径。这必须是现有目录。为 `DockerUserId` 指定的用户必须具有在此目录中创建文件的权限。有关更多信息，请参阅 the section called “在核心上设置 Docker 用户”。

> **Important**
>
> 此目录存储 Docker Compose 文件以及来自私有存储库 的 Docker 映像的凭证。您有责任确保此目录安全。有关更多信息，请参阅 the section called “安全说明”。

AWS IoT 控制台中的显示名称：本地 Compose 文件的目录路径

必需：`true`

类型：`string`

有效模式 `\/.*\/?`

例如：`/home/username/myCompose`

`DockerUserId`

连接器以其身份运行的 Linux 用户的 UID。此用户必须属于核心设备上的 `docker` Linux 组，并且具有对 `DockerComposeFileDestinationPath` 目录的写入权限。有关更多信息，请参阅在核心上设置 Docker 用户。

> **Note**
>
> 除非绝对有必要，否则建议避免以根用户身份运行。如果您确实指定了 根用户，则必须允许 Lambda 函数以 root 身份在 AWS IoT Greengrass 核心上运行。有关更多信息，请参阅 the section called “以根用户身份运行 Lambda 函数”。

AWS IoT 控制台中的显示名称：Docker 用户 ID

必需：`false`

类型：`string`

有效模式：`^[0-9]{1,5}$`

`AWSSecretsArnList`

AWS Secrets Manager 中的密钥的 Amazon 资源名称 (ARN)，其中包含用于访问私有存储库中 Docker 映像的登录信息。有关更多信息，请参阅 the section called “从私有存储库访问 Docker 镜像”。

AWS IoT 控制台中的显示名称：私有存储库的凭证

必需：false。此参数是访问存储在私有存储库中的 Docker 映像所必需的。

类型：string 的 array

有效模式：[(?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]

DockerContainerStatusLogFrequency

连接器记录有关在核心上运行的 Docker 容器的状态信息的频率（以秒为单位）。默认值为 300 秒（5 分钟）。

AWS IoT 控制台中的显示名称：日志记录频率

必需：false

类型：string

有效模式：^[1-9]{1}[0-9]{0,3}$

ForceDeploy

指示如果 Docker 部署由于上次部署的清理不当而失败，是否强制执行此 Docker 部署。默认值为 False。

AWS IoT 控制台中的显示名称：强制部署

必需：false

类型：string

有效模式：^(true|false)$

DockerPullBeforeUp

表示部署程序是否应docker-compose pull在运行 pull-down-up 行为之前运行docker-compose up。默认值为 True。

AWS IoT 控制台中的显示名称：Docker 先下拉再上拉

必需：false

类型：string

有效模式：`^(true|false)$`

`StopContainersOnNewDeployment`

指示在 GGC 停止时（部署新组或内核关闭时 GGC 会停止），连接器是否应停止 Docker Deployer 托管的 Docker 容器。默认值为 `True`。

在AWS IoT控制台中显示名称：Docker 在新部署时停止

> **Note**
>
> 我们建议将此参数设置为其默认值 `True`。如果参数设置为 `False`，则即使在终止 AWS IoT Greengrass Core 或开始新部署后，您的 Docker 容器仍能继续运行。如果将此参数设置为 `False`，则必须确保在 `docker-compose` 服务名称变化或增加时根据需要维护您的 Docker 容器。
> 有关更多信息，请参阅 `docker-compose` Compose 文件文档。

必需：`false`

类型：`string`

有效模式：`^(true|false)$`

`DockerOfflineMode`

指示当 AWS IoT Greengrass 离线启动时是否使用现有的 Docker Compose 文件。默认值为 `False`。

必需：`false`

类型：`string`

有效模式：`^(true|false)$`

Version 6

`DockerComposeFileS3Bucket`

包含您的 Docker Compose 文件的 S3 存储桶的名称。创建桶时，请务必遵守《亚马逊简单存储服务用户指南》中描述的桶名称规则。

AWS IoT 控制台中显示名称：S3 中的 Docker Compose 文件

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`true`

类型：`string`

有效模式 `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Amazon S3 中的 Docker Compose 文件的对象密钥。有关更多信息（包括对象键命名指南），请参阅 Amazon Simple Storage Service 用户指南中的对象键和元数据。

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`true`

类型：`string`

有效模式 `.+`

`DockerComposeFileS3Version`

Amazon S3 中的 Docker Compose 文件的对象版本。有关更多信息（包括对象键命名指南），请参阅 Amazon Simple Storage Service 用户指南中的使用版本控制。

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`false`

类型：`string`

有效模式 `.+`

`DockerComposeFileDestinationPath`

用于存储 Docker Compose 文件副本的本地目录的绝对路径。这必须是现有目录。为 `DockerUserId` 指定的用户必须具有在此目录中创建文件的权限。有关更多信息，请参阅 the section called “在核心上设置 Docker 用户”。

> **Important**
>
> 此目录存储 Docker Compose 文件以及来自私有存储库 的 Docker 映像的凭证。您有责任确保此目录安全。有关更多信息，请参阅 the section called “安全说明”。

AWS IoT 控制台中的显示名称：本地 Compose 文件的目录路径

必需：`true`

类型：`string`

有效模式 `\/.*\/?`

例如：`/home/username/myCompose`

`DockerUserId`

连接器以其身份运行的 Linux 用户的 UID。此用户必须属于核心设备上的 `docker` Linux 组，并且具有对 `DockerComposeFileDestinationPath` 目录的写入权限。有关更多信息，请参阅在核心上设置 Docker 用户。

> **Note**
>
> 除非绝对有必要，否则建议避免以根用户身份运行。如果您确实指定了 根用户，则必须允许 Lambda 函数以 root 身份在 AWS IoT Greengrass 核心上运行。有关更多信息，请参阅 the section called “以根用户身份运行 Lambda 函数”。

AWS IoT 控制台中的显示名称：Docker 用户 ID

必需：`false`

类型：`string`

有效模式：`^[0-9]{1,5}$`

`AWSSecretsArnList`

AWS Secrets Manager 中的密钥的 Amazon 资源名称 (ARN)，其中包含用于访问私有存储库中 Docker 映像的登录信息。有关更多信息，请参阅 the section called “从私有存储库访问 Docker 镜像”。

AWS IoT 控制台中的显示名称：私有存储库的凭证

必需：`false`。此参数是访问存储在私有存储库中的 Docker 映像所必需的。

类型：`string` 的 `array`

有效模式：`[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

连接器记录有关在核心上运行的 Docker 容器的状态信息的频率（以秒为单位）。默认值为 300 秒（5 分钟）。

AWS IoT 控制台中的显示名称：日志记录频率

必需：`false`

类型：`string`

有效模式：`^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

指示如果 Docker 部署由于上次部署的清理不当而失败，是否强制执行此 Docker 部署。默认值为 `False`。

AWS IoT 控制台中的显示名称：强制部署

必需：`false`

类型：`string`

有效模式：`^(true|false)$`

`DockerPullBeforeUp`

表示部署程序是否应`docker-compose pull`在运行 pull-down-up 行为之前运行`docker-compose up`。默认值为 `True`。

AWS IoT 控制台中的显示名称：Docker 先下拉再上拉

必需：`false`

类型：`string`

有效模式：`^(true|false)$`

`StopContainersOnNewDeployment`

指示在 GGC 停止时（部署新的组或关闭内核时 GGC 会停止），连接器是否应停止 Docker Deployer 托管的 Docker 容器。默认值为 `True`。

在AWS IoT控制台中显示名称：Docker 在新部署时停止

> **Note**
>
> 我们建议将此参数设置为其默认值 `True`。如果参数设置为 `False`，则即使在终止 AWS IoT Greengrass Core 或开始新部署后，您的 Docker 容器仍能继续运行。如果将此参数设置为 `False`，则必须确保在 `docker-compose` 服务名称变化或增加时根据需要维护您的 Docker 容器。
> 有关更多信息，请参阅 `docker-compose` Compose 文件文档。

必需：`false`

类型：`string`

有效模式：`^(true|false)$`

Version 5

`DockerComposeFileS3Bucket`

包含您的 Docker Compose 文件的 S3 存储桶的名称。创建桶时，请务必遵守《亚马逊简单存储服务用户指南》中描述的桶名称规则。

AWS IoT 控制台中显示名称：S3 中的 Docker Compose 文件

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`true`

类型：`string`

有效模式 `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Amazon S3 中的 Docker Compose 文件的对象密钥。有关更多信息（包括对象键命名指南），请参阅 Amazon Simple Storage Service 用户指南中的对象键和元数据。

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`true`

类型：`string`

有效模式 `.+`

`DockerComposeFileS3Version`

Amazon S3 中的 Docker Compose 文件的对象版本。有关更多信息（包括对象键命名指南），请参阅 Amazon Simple Storage Service 用户指南中的使用版本控制。

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`false`

类型：`string`

有效模式 `.+`

`DockerComposeFileDestinationPath`

用于存储 Docker Compose 文件副本的本地目录的绝对路径。这必须是现有目录。为 `DockerUserId` 指定的用户必须具有在此目录中创建文件的权限。有关更多信息，请参阅 the section called “在核心上设置 Docker 用户”。

> **Important**
>
> 此目录存储 Docker Compose 文件以及来自私有存储库 的 Docker 映像的凭证。您有责任确保此目录安全。有关更多信息，请参阅 the section called “安全说明”。

AWS IoT 控制台中的显示名称：本地 Compose 文件的目录路径

必需：`true`

类型：`string`

有效模式 `\/.*\/?`

例如：`/home/username/myCompose`

`DockerUserId`

连接器以其身份运行的 Linux 用户的 UID。此用户必须属于核心设备上的 `docker` Linux 组，并且具有对 `DockerComposeFileDestinationPath` 目录的写入权限。有关更多信息，请参阅在核心上设置 Docker 用户。

> **Note**
>
> 除非绝对有必要，否则建议避免以根用户身份运行。如果您确实指定了 根用户，则必须允许 Lambda 函数以 root 身份在 AWS IoT Greengrass 核心上运行。有关更多信息，请参阅 the section called “以根用户身份运行 Lambda 函数”。

AWS IoT 控制台中的显示名称：Docker 用户 ID

必需：false

类型：string

有效模式：^[0-9]{1,5}$

AWSSecretsArnList

AWS Secrets Manager 中的密钥的 Amazon 资源名称 (ARN)，其中包含用于访问私有存储库中 Docker 映像的登录信息。有关更多信息，请参阅 the section called “从私有存储库访问 Docker 镜像”。

AWS IoT 控制台中的显示名称：私有存储库的凭证

必需：false。此参数是访问存储在私有存储库中的 Docker 映像所必需的。

类型：string 的 array

有效模式：[(?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]

DockerContainerStatusLogFrequency

连接器记录有关在核心上运行的 Docker 容器的状态信息的频率（以秒为单位）。默认值为 300 秒（5 分钟）。

AWS IoT 控制台中的显示名称：日志记录频率

必需：false

类型：string

有效模式：^[1-9]{1}[0-9]{0,3}$

ForceDeploy

指示如果 Docker 部署由于上次部署的清理不当而失败，是否强制执行此 Docker 部署。默认值为 False。

AWS IoT 控制台中的显示名称：强制部署

必需：false

类型：`string`

有效模式：`^(true|false)$`

`DockerPullBeforeUp`

表示部署程序是否应`docker-compose pull`在运行 pull-down-up 行为之前运行`docker-compose up`。默认值为 `True`。

AWS IoT 控制台中的显示名称：Docker 先下拉再上拉

必需：`false`

类型：`string`

有效模式：`^(true|false)$`

Versions 2 - 4

`DockerComposeFileS3Bucket`

包含您的 Docker Compose 文件的 S3 存储桶的名称。创建桶时，请务必遵守《亚马逊简单存储服务用户指南》中描述的桶名称规则。

AWS IoT 控制台中显示名称：S3 中的 Docker Compose 文件

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`true`

类型：`string`

有效模式 `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Amazon S3 中的 Docker Compose 文件的对象密钥。有关更多信息（包括对象键命名指南），请参阅 Amazon Simple Storage Service 用户指南中的对象键和元数据。

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`true`

类型：`string`

有效模式 `.+`

`DockerComposeFileS3Version`

Amazon S3 中的 Docker Compose 文件的对象版本。有关更多信息（包括对象键命名指南），请参阅 Amazon Simple Storage Service 用户指南中的使用版本控制。

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`false`

类型：`string`

有效模式 `.+`

`DockerComposeFileDestinationPath`

用于存储 Docker Compose 文件副本的本地目录的绝对路径。这必须是现有目录。为 `DockerUserId` 指定的用户必须具有在此目录中创建文件的权限。有关更多信息，请参阅 the section called “在核心上设置 Docker 用户”。

> **Important**
>
> 此目录存储 Docker Compose 文件以及来自私有存储库 的 Docker 映像的凭证。您有责任确保此目录安全。有关更多信息，请参阅 the section called “安全说明”。

AWS IoT 控制台中的显示名称：本地 Compose 文件的目录路径

必需：`true`

类型：`string`

有效模式 `\/.*\/?`

例如：`/home/username/myCompose`

`DockerUserId`

连接器以其身份运行的 Linux 用户的 UID。此用户必须属于核心设备上的 `docker` Linux 组，并且具有对 `DockerComposeFileDestinationPath` 目录的写入权限。有关更多信息，请参阅在核心上设置 Docker 用户。

> **Note**
>
> 除非绝对有必要，否则建议避免以根用户身份运行。如果您确实指定了 根用户，则必须允许 Lambda 函数以 root 身份在 AWS IoT Greengrass 核心上运行。有关更多信息，请参阅 the section called “以根用户身份运行 Lambda 函数”。

AWS IoT 控制台中的显示名称：Docker 用户 ID

必需：`false`

类型：`string`

有效模式：`^[0-9]{1,5}$`

`AWSSecretsArnList`

AWS Secrets Manager 中的密钥的 Amazon 资源名称 (ARN)，其中包含用于访问私有存储库中 Docker 映像的登录信息。有关更多信息，请参阅 the section called “从私有存储库访问 Docker 镜像”。

AWS IoT 控制台中的显示名称：私有存储库的凭证

必需：`false`。此参数是访问存储在私有存储库中的 Docker 映像所必需的。

类型：`string` 的 `array`

有效模式：`[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

连接器记录有关在核心上运行的 Docker 容器的状态信息的频率（以秒为单位）。默认值为 300 秒（5 分钟）。

AWS IoT 控制台中的显示名称：日志记录频率

必需：`false`

类型：`string`

有效模式：`^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

指示如果 Docker 部署由于上次部署的清理不当而失败，是否强制执行此 Docker 部署。默认值为 `False`。

AWS IoT 控制台中的显示名称：强制部署

必需：`false`

类型：`string`

有效模式：`^(true|false)$`

Version 1

`DockerComposeFileS3Bucket`

包含您的 Docker Compose 文件的 S3 存储桶的名称。创建桶时，请务必遵守《亚马逊简单存储服务用户指南》中描述的桶名称规则。

AWS IoT 控制台中显示名称：S3 中的 Docker Compose 文件

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`true`

类型：`string`

有效模式 `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

Amazon S3 中的 Docker Compose 文件的对象密钥。有关更多信息（包括对象键命名指南），请参阅 Amazon Simple Storage Service 用户指南中的对象键和元数据。

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`true`

类型：`string`

有效模式 `.+`

`DockerComposeFileS3Version`

Amazon S3 中的 Docker Compose 文件的对象版本。有关更多信息（包括对象键命名指南），请参阅 Amazon Simple Storage Service 用户指南中的使用版本控制。

> **Note**
>
> 在控制台中，Docker Compose file in S3 (S3 中的 Docker Compose 文件) 属性结合了 `DockerComposeFileS3Bucket`、`DockerComposeFileS3Key` 和 `DockerComposeFileS3Version` 参数。

必需：`false`

类型：`string`

有效模式 `.+`

`DockerComposeFileDestinationPath`

用于存储 Docker Compose 文件副本的本地目录的绝对路径。这必须是现有目录。为 `DockerUserId` 指定的用户必须具有在此目录中创建文件的权限。有关更多信息，请参阅 the section called “在核心上设置 Docker 用户”。

> **Important**
>
> 此目录存储 Docker Compose 文件以及来自私有存储库 的 Docker 映像的凭证。您有责任确保此目录安全。有关更多信息，请参阅 the section called “安全说明”。

AWS IoT 控制台中的显示名称：本地 Compose 文件的目录路径

必需：`true`

类型：`string`

有效模式 `\/.*\/?`

例如：`/home/username/myCompose`

`DockerUserId`

连接器以其身份运行的 Linux 用户的 UID。此用户必须属于核心设备上的 `docker` Linux 组，并且具有对 `DockerComposeFileDestinationPath` 目录的写入权限。有关更多信息，请参阅在核心上设置 Docker 用户。

> **Note**
>
> 除非绝对有必要，否则建议避免以根用户身份运行。如果您确实指定了 根用户，则必须允许 Lambda 函数以 root 身份在 AWS IoT Greengrass 核心上运行。有关更多信息，请参阅 the section called “以根用户身份运行 Lambda 函数”。

AWS IoT 控制台中的显示名称：Docker 用户 ID

必需：`false`

类型：`string`

有效模式：`^[0-9]{1,5}$`

`AWSSecretsArnList`

AWS Secrets Manager 中的密钥的 Amazon 资源名称 (ARN)，其中包含用于访问私有存储库中 Docker 映像的登录信息。有关更多信息，请参阅 the section called “从私有存储库访问 Docker 镜像”。

AWS IoT 控制台中的显示名称：私有存储库的凭证

必需：`false`。此参数是访问存储在私有存储库中的 Docker 映像所必需的。

类型：`string` 的 `array`

有效模式：`[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

连接器记录有关在核心上运行的 Docker 容器的状态信息的频率（以秒为单位）。默认值为 300 秒（5 分钟）。

AWS IoT 控制台中的显示名称：日志记录频率

必需：`false`

类型：`string`

有效模式：`^[1-9]{1}[0-9]{0,3}$`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 Greengrass Docker 应用程序部署连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyDockerAppplicationDeploymentConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
DockerApplicationDeployment/versions/5",
            "Parameters": {
                "DockerComposeFileS3Bucket": "myS3Bucket",
```

```
                "DockerComposeFileS3Key": "production-docker-compose.yml",
                "DockerComposeFileS3Version": "123",
                "DockerComposeFileDestinationPath": "/home/username/myCompose",
                "DockerUserId": "1000",
                "AWSSecretsArnList": "[\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret1-hash\",\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret2-hash\"]",
                "DockerContainerStatusLogFrequency": "30",
                "ForceDeploy": "True",
                "DockerPullBeforeUp": "True"
            }
        }
    ]
}'
```

> **Note**
>
> 此连接器中的 Lambda 函数的生命周期很长。

输入数据

此连接器不需要或接受输入数据。

输出数据

此连接器将 `docker-compose up` 命令的状态发布为输出数据。

订阅中的主题筛选条件

```
dockerapplicationdeploymentconnector/message/status
```

示例输出：成功

```
{
  "status":"success",
  "GreengrassDockerApplicationDeploymentStatus":"Successfully triggered docker-
compose up",
  "S3Bucket":"myS3Bucket",
  "ComposeFileName":"production-docker-compose.yml",
  "ComposeFileVersion":"123"
}
```

示例输出：失败

```
{
  "status":"fail",
  "error_message":"description of error",
  "error":"InvalidParameter"
}
```

错误类型可以是 `InvalidParameter` 或 `InternalError`。

在 AWS IoT Greengrass 核心上设置 Docker 用户

Greengrass Docker 应用程序部署连接器以您为 `DockerUserId` 参数指定的用户的身份运行。如果不指定值，则连接器将以 `ggc_user` 身份运行，这是默认的 Greengrass 访问标识。

要允许连接器与 Docker 守护程序交互，Docker 用户必须属于核心上的 `docker` Linux 组。Docker 用户还必须具有对 `DockerComposeFileDestinationPath` 目录的写入权限。这是连接器存储本地 `docker-compose.yml` 文件和 Docker 凭证的位置。

> **Note**
>
> - 我们建议您创建 Linux 用户，而不是使用默认 `ggc_user`。否则，Greengrass 组中的任何 Lambda 函数都可以访问 Compose 文件和 Docker 凭证。
> - 除非绝对有必要，否则建议避免以根用户身份运行。如果您确实指定了 根用户，则必须允许 Lambda 函数以 root 身份在 AWS IoT Greengrass 核心上运行。有关更多信息，请参阅 the section called “以根用户身份运行 Lambda 函数”。

1. 创建 用户。您可以运行 `useradd` 命令并包含用于分配 UID 的可选 `-u` 选项。例如：

   ```
   sudo useradd -u 1234 user-name
   ```

2. 将用户添加到核心上的 `docker` 组。例如：

   ```
   sudo usermod -aG docker user-name
   ```

   有关详细信息，包括如何创建 `docker` 组，请参阅 Docker 文档中的以非 root 用户身份管理 Docker。

3. 授予用户写入为 DockerComposeFileDestinationPath 参数指定的目录的权限。例如：

   a. 将用户设置为目录的所有者。此示例使用步骤 1 中的 UID。

   ```
   chown 1234 docker-compose-file-destination-path
   ```

   b. 为所有者授予读写权限。

   ```
   chmod 700 docker-compose-file-destination-path
   ```

   有关详细信息，请参阅 Linux Foundation 文档中的如何在 Linux 中管理文件和文件夹权限。

   c. 如果您在创建用户时没有分配 UID，或者使用现有用户，请运行 id 命令查找 UID。

   ```
   id -u user-name
   ```

   您可以使用 UID 配置连接器的 DockerUserId 参数。

使用情况信息

当您使用 Greengrass Docker 应用程序部署连接器时，您应该注意以下特定于实现的使用信息。

- 项目名称的固定前缀。连接器将 greengrassdockerapplicationdeployment 前缀附加到其启动的 Docker 容器的名称前面。连接器在它所运行的 docker-compose 命令中使用此前缀作为项目名称。
- 日志记录行为。连接器将状态信息和疑难解答信息写入日志文件。您可以配置为将日志发送AWS IoT Greengrass到 CloudWatch 日志并在本地写入日志。有关更多信息，请参阅 the section called “日志记录”。这是连接器的本地日志的路径：

  ```
  /greengrass-root/ggc/var/log/user/region/aws/DockerApplicationDeployment.log
  ```

  您必须具有 root 权限才能访问本地日志。
- 更新 Docker 映像。Docker 在核心设备上缓存镜像。如果您更新 Docker 镜像并希望将更改传播到核心设备，请确保更改 Compose 文件中镜像的标签。更改将在部署 Greengrass 组后生效。
- 清理操作的 10 分钟超时。当 Greengrass 守护程序（在重新启动期间）停止时，将启动该 docker-compose down 命令。所有 Docker 容器在启动 docker-compose down 后最多具有 10 分钟以执行任何清理操作。如果清理在 10 分钟内未完成，则必须手动清理剩余的容器。有关更多信息，请参阅 Docker CLI 文档中的 docker rm。

- 运行 Docker 命令。要解决问题，您可以在核心设备上的终端窗口中运行 Docker 命令。例如，运行以下命令以查看由连接器启动的 Docker 容器：

  ```
  docker ps --filter name="greengrassdockerapplicationdeployment"
  ```

- 保留资源 ID。连接器使用它在 Greengrass 组中创建的 Greengrass 资源的 `DOCKER_DEPLOYER_SECRET_RESOURCE_RESERVED_ID_`*`index`* ID。资源 ID 在组中必须是唯一的，因此不要分配可能与此保留资源 ID 冲突的资源 ID。
- 离线模式。将 `DockerOfflineMode` 配置参数设置为 `True` 时，Docker 连接器就可以在离线模式下运行。如果 Greengrass 组部署在核心设备离线时重新启动，并且连接器无法建立到 Amazon S3 或 Amazon ECR 的连接以检索 Docker 合成文件，此时可能会发生这种情况。

  启用离线模式后，连接器将尝试下载合成文件，并像正常重新启动一样运行 `docker login` 命令。如果这些尝试失败，则连接器会在使用 `DockerComposeFileDestinationPath` 参数指定的文件夹中查找本地存储的 Compose 文件。如果存在本地 Compose 文件，则连接器会遵循正常的 `docker-compose` 命令顺序并从本地映像中提取。如果 Compose 文件或本地映像不存在，则连接器将失败。在离线模式下，`ForceDeploy` 和 `StopContainersOnNewDeployment` 参数的行为保持不变。

与 Docker 容器通信

AWS IoT Greengrass 支持 Greengrass 组件和 Docker 容器之间的以下通信通道：

- Greengrass Lambda 函数可以使用 REST API 与 Docker 容器中的进程进行通信。可以在打开端口的 Docker 容器中设置服务器。Lambda 函数可以使用该端口与容器通信。
- Docker 容器中的进程可以通过本地 Greengrass 消息代理交换 MQTT 消息。您可以将 Docker 容器设置为 Greengrass 组中的客户端设备，然后创建订阅以允许容器与组中的 Greengrass Lambda 函数、客户端设备和其他连接器或与 AWS IoT 本地影子服务进行通信。有关更多信息，请参阅 the section called “配置 MQTT 与 Docker 容器的通信”。
- Greengrass Lambda 函数可以更新共享文件以将信息传递给 Docker 容器。您可以使用 Compose 文件绑定装载 Docker 容器的共享文件路径。

配置 MQTT 与 Docker 容器的通信

您可以将 Docker 容器配置为客户端设备，并将其添加到 Greengrass 组中。然后，您可以创建允许在 Docker 容器和 Greengrass 组件或 AWS IoT 之间进行 MQTT 通信的订阅。在以下过程中，您将创建允许 Docker 容器设备从本地影子服务接收影子更新消息的订阅。您可以按照此模式创建其他订阅。

> **Note**
>
> 此过程假设您已经创建了 Greengrass 组和 Greengrass 核心 (v1.10 或更高版本)。有关创建 Greengrass 组和核心的信息，请参阅 入门 AWS IoT Greengrass。

将 Docker 容器配置为客户端设备并将其添加到 Greengrass 组

1. 在核心设备上创建一个文件夹，以存储用于对 Greengrass 设备进行身份验证的证书和密钥。

   文件路径必须装载在要启动的 Docker 容器上。以下代码段显示如何在您的 Compose 文件中装载文件路径。在此示例中，*path-to-device-certs*表示您在此步骤中创建的文件夹。

   ```
   version: '3.3'
   services:
     myService:
       image: user-name/repo:image-tag
       volumes:
         - /path-to-device-certs/:/path-accessible-in-container
   ```

2. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
3. 选择目标组。
4. 在组配置页面中，选择客户端设备，然后选择关联。
5. 在将客户端设备与此组关联模态中，选择新建 AWS IoT 事物。

   此时将在新选项卡中打开创建事物页面。
6. 在 创建事物页面上，选择 创建单个事物，然后选择 下一步。
7. 在指定事物属性页面上，输入设备的名称，然后选择下一步。
8. 在配置设备证书页面上，选择下一步。
9. 在 将策略附加到证书 页面上，执行下列操作之一：
   - 选择授予客户端设备所需权限的现有策略，然后选择创建事物。

     这将打开一个模态，供您下载设备用于连接到 AWS Cloud 和核心的证书和密钥。
   - 创建并附加一个新策略，以授予客户端权限。执行以下操作：

     a. 选择创建策略。

        此时将在新选项卡中打开创建策略页面。

b. 在创建策略页面上，执行以下操作：

i. 在策略名称中，输入一个名称来描述该策略，例如 **`GreengrassV1ClientDevicePolicy`**。

ii. 在策略语句选项卡的策略文档下，选择 JSON。

iii. 输入以下策略文档。此策略允许客户端设备发现 Greengrass 核心并就所有 MQTT 主题进行通信。有关如何限制此策略访问权限的信息，请参阅AWS IoT Greengrass 的设备身份验证和授权。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. 选择 Create（创建）以创建策略。

c. 返回到打开了将策略附加到证书页面的浏览器标签页中。执行以下操作：

i. 在策略列表中，选择您创建的策略，例如 GreengrassV1ClientDevicePolicy。

如果没有看到策略，请选择刷新按钮。

ii. 选择 Create thing（创建事物）。

这将打开一个模态，供您下载设备用于连接到 AWS Cloud 和核心的证书和密钥。

10. 在下载证书和密钥模态中，下载设备的证书。

> **Important**
>
> 请先下载安全资源，然后再选择 完成。

执行以下操作：

a. 在设备证书中，选择下载以下载该设备证书。
b. 在公钥文件中，选择下载以下载该证书的公钥。
c. 在私钥文件中，选择下载以下载该证书的私钥文件。
d. 查看《AWS IoT 开发人员指南》中的服务器身份验证，然后选择相应的根 CA 证书。我们建议您使用 Amazon Trust Services (ATS) 端点和 ATS 根 CA 证书。在根 CA 证书下，针对根 CA 证书选择下载。
e. 选择完成。

记下设备证书和密钥文件名中常见的证书 ID。稍后您将需要用到它。

11. 将证书和密钥复制到您在步骤 1 中创建的文件夹。

接下来，在组中创建订阅。对于此示例，您创建一个订阅，以允许 Docker 容器设备从本地影子服务接收 MQTT 消息。

> **Note**
>
> 影子文档的最大大小为 8 KB。有关更多信息，请参阅《AWS IoT 开发人员指南》中的 AWS IoT 配额。

创建允许 Docker 容器设备从本地影子服务接收 MQTT 消息的订阅。

1. 在组配置页面中，选择订阅选项卡，然后选择添加订阅。

2. 在选择您的源和目标页面，配置源和目标，如下所示：

   a. 对于选择源，选择服务，然后选择本地影子服务。

   b. 对于 Select a target (选择目标)，选择 Devices (设备)，然后选择您的设备。

   c. 请选择 Next（下一步）。

   d. 在 利用主题筛选您的数据 页面上，对于 主题筛选条件，选择 **$aws/things/*MyDockerDevice*/shadow/update/accepted**，然后选择下一步。*MyDockerDevice*替换为您之前创建的设备的名称。

   e. 选择完成。

在您的 Compose 文件中引用的 Docker 镜像中包含以下代码段。这是 Greengrass 设备代码。此外，在 Docker 容器中添加代码，以启动容器内的 Greengrass 设备。它可以作为一个单独的进程在镜像或单独的线程中运行。

```
import os
import sys
import time
import uuid

from AWSIoTPythonSDK.core.greengrass.discovery.providers import DiscoveryInfoProvider
from AWSIoTPythonSDK.exception.AWSIoTExceptions import DiscoveryInvalidRequestException
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# Replace thingName with the name you registered for the Docker device.
thingName = "MyDockerDevice"
clientId = thingName

# Replace host with the IoT endpoint for your &AWS-account;.
host = "myPrefix.iot.region.amazonaws.com"

# Replace topic with the topic where the Docker container subscribes.
topic = "$aws/things/MyDockerDevice/shadow/update/accepted"

# Replace these paths based on the download location of the certificates for the Docker
 container.
rootCAPath = "/path-accessible-in-container/AmazonRootCA1.pem"
certificatePath = "/path-accessible-in-container/certId-certificate.pem.crt"
privateKeyPath = "/path-accessible-in-container/certId-private.pem.key"
```

```
# Discover Greengrass cores.
discoveryInfoProvider = DiscoveryInfoProvider()
discoveryInfoProvider.configureEndpoint(host)
discoveryInfoProvider.configureCredentials(rootCAPath, certificatePath, privateKeyPath)
discoveryInfoProvider.configureTimeout(10)  # 10 seconds.

GROUP_CA_PATH = "./groupCA/"
MQTT_QOS = 1

discovered = False
groupCA = None
coreInfo = None

try:
    # Get discovery info from AWS IoT.
    discoveryInfo = discoveryInfoProvider.discover(thingName)
    caList = discoveryInfo.getAllCas()
    coreList = discoveryInfo.getAllCores()

    # Use first discovery result.
    groupId, ca = caList[0]
    coreInfo = coreList[0]

    # Save the group CA to a local file.
    groupCA = GROUP_CA_PATH + groupId + "_CA_" + str(uuid.uuid4()) + ".crt"
    if not os.path.exists(GROUP_CA_PATH):
        os.makedirs(GROUP_CA_PATH)
    groupCAFile = open(groupCA, "w")
    groupCAFile.write(ca)
    groupCAFile.close()
    discovered = True
except DiscoveryInvalidRequestException as e:
    print("Invalid discovery request detected!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")
except BaseException as e:
    print("Error in discovery!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")

myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)
myAWSIoTMQTTClient.configureCredentials(groupCA, privateKeyPath, certificatePath)
```

```
# Try to connect to the Greengrass core.
connected = False
for connectivityInfo in coreInfo.connectivityInfoList:
    currentHost = connectivityInfo.host
    currentPort = connectivityInfo.port
    myAWSIoTMQTTClient.configureEndpoint(currentHost, currentPort)
    try:
        myAWSIoTMQTTClient.connect()
        connected = True
    except BaseException as e:
        print("Error in connect!")
        print("Type: %s" % str(type(e)))
        print("Error message: %s" % str(e))
    if connected:
        break

if not connected:
    print("Cannot connect to core %s. Exiting..." % coreInfo.coreThingArn)
    sys.exit(-2)

# Handle the MQTT message received from GGShadowService.
def customCallback(client, userdata, message):
    print("Received an MQTT message")
    print(message)

# Subscribe to the MQTT topic.
myAWSIoTMQTTClient.subscribe(topic, MQTT_QOS, customCallback)

# Keep the process alive to listen for messages.
while True:
    time.sleep(1)
```

安全说明

使用 Greengrass Docker 应用程序部署连接器时，请注意以下安全注意事项。

Docker Compose 文件的本地存储

连接器将 Compose 文件的副本存储在为 `DockerComposeFileDestinationPath` 参数指定的目录中。

您负责确保此目录安全。您应该使用文件系统权限来限制对目录的访问。

Docker 凭证的本地存储

如果 Docker 镜像存储在私有存储库中，则连接器将 Docker 凭证存储在为 `DockerComposeFileDestinationPath` 参数指定的目录中。

您负责保护这些证书。例如，在安装 Docker Engine 时，应在核心设备上使用 credential-helper。

从可信来源安装 Docker Engine

您负责从可信来源安装 Docker Engine。此连接器使用核心设备上的 Docker 守护程序访问您的 Docker 资产并管理 Docker 容器。

Greengrass 组角色权限的范围

在 Greengrass 组角色中添加的权限可由 Greengrass 组中的所有 Lambda 函数和连接器代入。此连接器需要访问存储在 S3 存储桶中的 Docker Compose 文件。如果您的 Docker 镜像存储在 Amazon ECR 中的私有存储库中，它还需要访问您的 Amazon ECR 授权令牌。

许可证

Greengrass Docker 应用程序部署连接器包含以下第三方软件/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

此连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
7	增加了 `DockerOfflineMode` ，以在 AWS IoT Greengrass 离线启动时使用现有的 Docker Compose 文件。为 `docker login` 命令实现了重试。支持 32 位 UID。
6	增加了 `StopContainersOnNewDeployment` ，用于在进行新部署或 GGC 停止时覆盖容器清理。更安全的关机和启动机制。YAML 验证错误修复。
5	在运行 `docker-compose down` 之前先提取映像。
4	添加了更新 Docker 镜像的 pull-before-up 行为。
3	修复了查找环境变量的问题。
2	添加了 `ForceDeploy` 参数。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”

IoT Analytics 连接器

> **Warning**
>
> 此连接器已进入生命周期延长阶段，AWS IoT Greengrass 不会发布更新来提供功能、现有功能增强、安全补丁或错误修复。有关更多信息，请参阅AWS IoT Greengrass Version 1 维护策略。

IoT Analytics 连接器将本地设备数据发送到 AWS IoT Analytics。您可以使用此连接器作为中央枢纽来收集 Greengrass 核心设备上的传感器以及已连接的客户端设备中的数据。连接器将数据发送到当前 AWS 账户 和区域中的 AWS IoT Analytics 通道。它可以将数据发送到默认目标通道和动态指定的通道。

> **Note**
>
> AWS IoT Analytics 是一项完全托管的服务，可让您收集、存储、处理和查询 IoT 数据。在 AWS IoT Analytics 中，可以进一步分析和处理该类数据。例如，它可用于训练用于监控机器运行状况的 ML 模型或测试新的建模策略。有关更多信息，请参阅《AWS IoT Analytics 用户指南》中的什么是 AWS IoT Analytics？。

连接器接受输入 MQTT 主题上的格式化和非格式化数据。它支持两个预定义的主题，其中以内联方式指定目标通道。它还可以接收有关订阅中配置的客户定义的主题的消息。这可用于路由客户端设备中发布到固定主题的消息或处理资源有限的设备中的非结构化或堆栈相关的数据。

此连接器使用 `BatchPutMessage` API 将数据（以 JSON 或 base64 编码字符串的形式）发送到目标通道。连接器可以将原始数据处理为符合 API 要求的格式。连接器将输入消息缓存在每个通道队列中并异步处理批次。它提供了可让您控制队列和批处理行为以及限制内存消耗的参数。例如，您可以配置最大队列大小、批处理间隔、内存大小和活动通道数。

此连接器具有以下版本。

版本	ARN
4	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/4`
3	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/3`
2	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/2`

版本	ARN
1	`arn:aws:greengrass:  region::/connectors/IoTAnalytics/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 3 - 4

- AWS IoT Greengrass Core 软件 v1.9.3 版或更高版本。
- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

  > **Note**
  >
  > 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- 此连接器只能在同时支持 AWS IoT Greengrass 和 AWS IoT Analytics 的 Amazon Web Services 区域中使用。
- 创建和配置所有相关的 AWS IoT Analytics 实体和工作流程。实体包括通道、管道、数据存储和数据集。有关更多信息，请参阅《AWS IoT Analytics 用户指南》中的 AWS CLI 或控制台过程。

  > **Note**
  >
  > 目标 AWS IoT Analytics 通道必须使用与此连接器相同的账户并位于其所在的相同 AWS 区域 中。

- Greengrass 组角色，配置为允许对目标通道执行 `iotanalytics:BatchPutMessage` 操作，如以下示例 IAM 策略中所示。通道必须位于当前 AWS 账户 和区域中。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "iotanalytics:BatchPutMessage"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
                "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
            ]
        }
    ]
}
```

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

Versions 1 - 2

- AWS IoT Greengrass Core 软件 v1.7 版或更高版本。
- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- 此连接器只能在同时支持 AWS IoT Greengrass 和 AWS IoT Analytics 的 Amazon Web Services 区域中使用。
- 创建和配置所有相关的 AWS IoT Analytics 实体和工作流程。实体包括通道、管道、数据存储和数据集。有关更多信息，请参阅《AWS IoT Analytics 用户指南》中的 AWS CLI 或控制台过程。

> **Note**
>
> 目标 AWS IoT Analytics 通道必须使用与此连接器相同的账户并位于其所在的相同 AWS 区域 中。

- Greengrass 组角色，配置为允许对目标通道执行 `iotanalytics:BatchPutMessage` 操作，如以下示例 IAM 策略中所示。通道必须位于当前 AWS 账户 和区域中。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "iotanalytics:BatchPutMessage"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
                "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
            ]
        }
    ]
}
```

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

参数

`MemorySize`

要分配给此连接器的内存量（以 KB 为单位）。

AWS IoT 控制台中的显示名称：内存大小

必需：`true`

类型：`string`

有效模式：`^[0-9]+$`

`PublishRegion`

在其中创建 AWS IoT Analytics 通道的 AWS 区域。使用与连接器相同的区域。

> **Note**
>
> 这必须还与在组角色中指定的通道的区域匹配。

AWS IoT 控制台中的显示名称：发布区域

必需：false

类型：string

有效模式：^$|([a-z]{2}-[a-z]+-\\d{1})

PublishInterval

将一批收到的数据发布到 AWS IoT Analytics 的时间间隔（以秒为单位）。

AWS IoT 控制台中的显示名称：发布间隔

必需：false

类型：string

默认值：1

有效模式：$|^[0-9]+$

IotAnalyticsMaxActiveChannels

连接器主动监控的 AWS IoT Analytics 通道的最大数量。此值必须大于 0，并且至少等于您希望在某个指定时间发布到连接器的通道的数量。

您可以使用此参数通过限制连接器可在某个给定时间管理的队列总数来限制内存消耗量。在发送所有排队的消息后，队列将被删除。

AWS IoT 控制台中的显示名称：活动频道的最大数量

必需：false

类型：string

默认值：50

有效模式：^$|^[1-9][0-9]*$

IotAnalyticsQueueDropBehavior

当队列已满时从通道队列中丢弃消息的行为。

AWS IoT 控制台中的显示名称：队列丢弃行为

必需：false

类型：string

有效值：DROP_NEWEST 或 DROP_OLDEST

默认值：DROP_NEWEST

有效模式：^DROP_NEWEST$|^DROP_OLDEST$

IotAnalyticsQueueSizePerChannel

在提交或丢弃消息前要在内存（每个通道）中保留的消息的最大数量。此值必须大于 0。

AWS IoT 控制台中的显示名称：每个频道的最大队列大小

必需：false

类型：string

默认值：2048

有效模式：^$|^[1-9][0-9]*$

IotAnalyticsBatchSizePerChannel

要在一个批处理请求中发送到 AWS IoT Analytics 通道的消息的最大数量。此值必须大于 0。

AWS IoT 控制台中的显示名称：每个频道要批处理的最大消息数

必需：false

类型：string

默认值：5

有效模式：^$|^[1-9][0-9]*$

IotAnalyticsDefaultChannelName

此连接器用于已发送到客户定义的输入主题的 AWS IoT Analytics 通道的名称。

AWS IoT 控制台中的显示名称：默认频道名称

必需：`false`

类型：`string`

有效模式：`^[a-zA-Z0-9_]$`

`IsolationMode`

此连接器的容器化模式。默认值为 `GreengrassContainer`，这意味着连接器在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

> **Note**
>
> 组的默认容器化设置不适用于连接器。

AWS IoT 控制台中的显示名称：容器隔离模式

必需：`false`

类型：`string`

有效值：`GreengrassContainer` 或 `NoContainer`

有效模式：`^NoContainer$|^GreengrassContainer$`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 IoT Analytics 连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyIoTAnalyticsApplication",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTAnalytics/
versions/3",
            "Parameters": {
                "MemorySize": "65535",
                "PublishRegion": "us-west-1",
                "PublishInterval": "2",
```

```
                "IotAnalyticsMaxActiveChannels": "25",
                "IotAnalyticsQueueDropBehavior": "DROP_OLDEST",
                "IotAnalyticsQueueSizePerChannel": "1028",
                "IotAnalyticsBatchSizePerChannel": "5",
                "IotAnalyticsDefaultChannelName": "my_channel"
            }
        }
    ]
}'
```

> **Note**
>
> 此连接器中的 Lambda 函数的生命周期很长。

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅the section called “连接器入门（控制台）”。

输入数据

此连接器接受有关预定义的和客户定义的 MQTT 主题的数据。发布者可以是客户端设备、Lambda 函数或其他连接器。

预定义主题

连接器支持以下两个结构化的 MQTT 主题，这些主题允许发布者以内联方式指定通道名称。

- `iotanalytics/channels/+/messages/put` 主题上的格式化消息。这些输入消息中的 IoT 数据必须格式化为 JSON 和 base64 编码字符串。
- `iotanalytics/channels/+/messages/binary/put` 主题上的非格式化消息。此主题上接收的输入消息将被视为二进制数据，并且可包含任何数据类型。

  要发布到预定义主题，请将 + 通配符替换为通道名称。例如：

  ```
  iotanalytics/channels/my_channel/messages/put
  ```

客户定义的主题

连接器支持 # 主题语法，可让其接受您在订阅中配置的任何 MQTT 主题上的输入消息。我们建议您在订阅中指定主题路径，而不是仅使用 # 通配符。 这些消息将发送到您为连接器指定的默认通道。

客户定义的主题上的输入消息将被视为二进制数据。它们可以使用任何消息格式，并且可以包含任何数据类型。您可以使用客户定义的主题来路由设备中发布到固定主题的消息。您还可以使用这些主题来接受无法将数据处理为要发送到连接器的格式化消息的客户端设备中的输入数据。

有关订阅和 MQTT 主题的更多信息，请参阅the section called “输入和输出”。

组角色必须允许对所有目标通道执行 `iotanalytics:BatchPutMessage` 操作。有关更多信息，请参阅the section called “要求”。

主题筛选条件：`iotanalytics/channels/+/messages/put`

使用此主题将格式化消息发送到连接器并动态指定目标通道。此主题还允许您指定在响应输出中返回的 ID。连接器验证 ID 对于其发送到 AWS IoT Analytics 的出站 `BatchPutMessage` 请求中的每个消息是否是唯一的。将丢弃具有重复 ID 的消息。

发送到此主题的输入数据必须使用以下消息格式。

消息属性

`request`

要发送到指定通道的数据。

必需：`true`

类型：包含以下属性的 `object`：

`message`

采用 JSON 或 base64 编码字符串形式的设备或传感器数据。

必需：`true`

类型：`string`

`id`

请求的任意 ID。此属性用于将输入请求映射到输出响应。如果指定，响应对象中的 `id` 属性将设置为该值。如果您忽略此属性，则连接器将生成一个 ID。

必需：`false`

类型：`string`

有效模式：`.*`

示例输入

```
{
    "request": {
        "message" : "{\"temp\":23.33}"
    },
    "id" : "req123"
}
```

主题筛选条件：`iotanalytics/channels/+/messages/binary/put`

使用此主题将未格式化的消息发送到连接器并动态指定目标通道。

连接器数据不会解析在此主题上收到的输入消息。它会将这些消息视为二进制数据。在将消息发送到 AWS IoT Analytics 之前，连接器会对其进行编码和格式化以符合 `BatchPutMessage` API 要求：

- 连接器对原始数据进行 base64 编码并将已编码的负载包含在出站 `BatchPutMessage` 请求中。
- 连接器为每个输入消息生成并分配一个 ID。

> **Note**
>
> 连接器的响应输出不包含这些输入消息的 ID 关联。

消息属性

无。

主题筛选条件：`#`

使用此主题将任何消息格式发送到默认通道。在您的客户端设备发布到固定主题或您希望将数据发送到无法将数据处理为连接器支持的消息格式的客户端设备中的默认通道时，这尤其有用。

您可以在创建用于将此连接器连接到数据源的订阅中定义主题语法。我们建议您在订阅中指定主题路径，而不是仅使用 # 通配符。

连接器数据不会解析已发布到此输入主题的消息。所有输入消息将被视为二进制数据。在将消息发送到 AWS IoT Analytics 之前，连接器会对其进行编码和格式化以符合 `BatchPutMessage` API 要求：

- 连接器对原始数据进行 base64 编码并将已编码的负载包含在出站 `BatchPutMessage` 请求中。
- 连接器为每个输入消息生成并分配一个 ID。

> **Note**
>
> 连接器的响应输出不包含这些输入消息的 ID 关联。

消息属性

无。

输出数据

此连接器将状态信息发布为 MQTT 主题的输出数据。 此信息包含其接收并发送到 AWS IoT Analytics 的每个输入消息的 AWS IoT Analytics 所返回的响应。

订阅中的主题筛选条件

```
iotanalytics/messages/put/status
```

示例输出：成功

```
{
    "response" : {
        "status" : "success"
    },
    "id" : "req123"
}
```

示例输出：失败

```
{
    "response" : {
        "status" : "fail",
        "error" : "ResourceNotFoundException",
        "error_message" : "A resource with the specified name could not be found."
    },
    "id" : "req123"
}
```

> **Note**
>
> 如果连接器检测到可重试的错误（如连接错误），它会在下一批次中重试发布。指数回退由 AWS 软件开发工具包进行处理。带可重试的错误的请求将添加回通道队列，以根据 `IotAnalyticsQueueDropBehavior` 参数进行进一步发布。

用法示例

使用以下概括步骤设置可用于尝试连接器的示例 Python 3.7 Lambda 函数。

> **Note**
>
> - 如果要使用其他 Python 运行时，您可以创建从 Python3.x 到 Python 3.7 的符号链接。
> - 连接器入门（控制台）和 连接器入门 (CLI) 主题包含详细步骤，说明如何配置和部署示例 Twilio 通知连接器。

1. 确保满足连接器的要求。

   对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

2. 创建并发布将输入数据发送到连接器的 Lambda 函数。

   将示例代码保存为 PY 文件。下载并解压适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。然后，创建一个 zip 包，其中在根级别包含 PY 文件和 `greengrasssdk` 文件夹。此 zip 包是您上传到 AWS Lambda 的部署包。

   创建 Python 3.7 Lambda 函数后，请发布函数版本并创建别名。

3. 配置 Greengrass 组。

   a. 通过别名来添加 Lambda 函数（推荐）。将 Lambda 生命周期配置为长时间生存（或在 CLI 中设置为 `"Pinned": true`）。

   b. 添加连接器并配置其参数。

   c. 添加允许连接器接收输入数据并针对支持的主题筛选条件发送输出数据的订阅。

      - 将 Lambda 函数设置为源，将连接器设置为目标，并使用支持的输入主题筛选条件。

- 将连接器设置为源，将 AWS IoT Core 设置为目标，并使用支持的输出主题筛选条件。您可以使用此订阅查看 AWS IoT 控制台中的状态消息。

4. 部署组。
5. 在 AWS IoT 控制台中的测试页面上，订阅输出数据主题以查看连接器中的状态消息。示例 Lambda 函数是长时间生存的，并且在部署组后立即开始发送消息。

   完成测试后，您可以将 Lambda 生命周期设置为按需（或在 CLI 中设置为 `"Pinned": false`）并部署组。这会阻止函数发送消息。

示例

以下示例 Lambda 函数向连接器发送一条输入消息。

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'iotanalytics/channels/my_channel/messages/put'

def create_request_with_all_fields():
    return  {
        "request": {
            "message" : "{\"temp\":23.33}"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Limits

此连接器受以下限制的约束。

- AWS SDK for Python (Boto3) 对 AWS IoT Analytics `batch_put_message` 操作施加的所有限值。
- 由 AWS IoT Analytics BatchPutMessage API 施加的所有配额。有关更多信息，AWS IoT Analytics 请参阅 AWS 一般参考 中的服务限额。
  - 每个通道每秒 100000 条消息。
  - 每个批次 100 条消息。
  - 每条消息 128 KB。

  此 API 使用通道名称（而不是通道 ARN），因此，不支持将数据发送到跨区域或跨账户通道。
- 由 AWS IoT Greengrass 核心施加的所有配额。有关更多信息，请参阅 AWS 一般参考 中的适用于 AWS IoT Greengrass 核心的服务限额。

  以下配额可能尤其适用：
  - 设备发送的消息的最大大小为 128 KB。
  - Greengrass 核心路由器中的最大消息队列大小为 2.5 MB。
  - 主题字符串的最大长度为 256 字节的 UTF-8 编码的字符。

许可证

IoT Analytics 连接器包含以下第三方软件/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

此连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
4	增加用于配置连接器容器化模式的 `IsolationMode` 参数。
3	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。
2	进行了修复，以减少过多的日志记录。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”
- AWS IoT Analytics 用户指南 中的什么是 AWS IoT Analytics？

IoT 以太网 IP 协议适配器连接器

IoT 以太网 IP 协议适配器连接器使用以太网/IP 协议从本地设备收集数据。您可以使用此连接器从多个设备收集数据并将其发布到 `StreamManager` 消息流。

也可以将此连接器与 IoT SiteWise 连接器和 IoT SiteWise 网关配合使用。您的网关必须提供连接器的配置。有关更多信息，请参阅《IoT SiteWise 用户指南》中的配置以太网/IP (EIP) 源。

> **Note**
>
> 此连接器在无容器隔离模式下运行，因此您可以将其部署到在 Docker 容器中运行的 AWS IoT Greengrass 组。

此连接器具有以下版本。

版本	ARN
2（推荐）	`arn:aws:greengrass:region::/connectors/IoTEIPProtocolAdaptor/versions/2`
1	`arn:aws:greengrass:region::/connectors/IoTEIPProtocolAdaptor/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 1 and 2

- AWS IoT Greengrass Core 软件 v1.10.2 版或更高版本。
- AWS IoT Greengrass 组上启用的流管理器 。
- Java 8 安装在核心设备上并已添加到 PATH 环境变量中。
- 至少 256 MB 的额外内存。这项要求是对 AWS IoT Greengrass Core 内存要求的补充。

> **Note**
>
> 该连接器仅在以下区域推出：
>
> - cn-north-1
> - ap-southeast-1

- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

连接器参数

此连接器支持以下参数：

`LocalStoragePath`

IoT SiteWise 连接器可以向其写入持久性数据的 AWS IoT Greengrass 主机上的目录。默认目录为 `/var/sitewise`。

AWS IoT 控制台中的显示名称：本地存储路径

必需：`false`

类型：`string`

有效模式：`^\s*$|\/.`

`ProtocolAdapterConfiguration`

一组 EtherNet/IP 收集器的配置，连接器从这些收集器中收集数据或连接到这些收集器。这可以是空列表。

AWS IoT 控制台中的显示名称：协议适配器配置

必需：`true`

类型：一个格式正确的 JSON 字符串，用于定义支持的反馈配置集。

以下是 `ProtocolAdapterConfiguration` 的一个示例：

```
{
    "sources": [
        {
            "type": "EIPSource",
            "name": "TestSource",
```

```
            "endpoint": {
                "ipAddress": "52.89.2.42",
                "port": 44818
            },
            "destination": {
                "type": "StreamManager",
                "streamName": "MyOutput_Stream",
                "streamBufferSize": 10
            },
            "destinationPathPrefix": "EIPSource_Prefix",
            "propertyGroups": [
                {
                    "name": "DriveTemperatures",
                    "scanMode": {
                        "type": "POLL",
                        "rate": 10000
                    },
                    "tagPathDefinitions": [
                        {
                            "type": "EIPTagPath",
                            "path": "arrayREAL[0]",
                            "dstDataType": "double"
                        }
                    ]
                }
            ]
        }
    ]
}
```

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 IoT Ethernet IP 协议适配器连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version
'{
    "Connectors": [
        {
            "Id": "MyIoTEIPProtocolConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
IoTEIPProtocolAdaptor/versions/2",
```

```
            "Parameters": {
                "ProtocolAdaptorConfiguration": "{ \"sources\": [{ \"type
\": \"EIPSource\", \"name\": \"Source1\", \"endpoint\": { \"ipAddress\":
 \"54.245.77.218\", \"port\": 44818 }, \"destinationPathPrefix\": \"EIPConnector_Prefix
\", \"propertyGroups\": [{ \"name\": \"Values\", \"scanMode\": { \"type\": \"POLL\",
 \"rate\": 2000 }, \"tagPathDefinitions\": [{ \"type\": \"EIPTagPath\", \"path\":
 \"arrayREAL[0]\", \"dstDataType\": \"double\" }]}]}]}",
                "LocalStoragePath": "/var/MyIoTEIPProtocolConnectorState"
            }
        }
    ]
}'
```

> **Note**
>
> 此连接器中的 Lambda 函数的生命周期很长。

输入数据

该连接器不接受 MQTT 消息作为输入数据。

输出数据

此连接器将数据发布到 `StreamManager`。您必须配置目标消息流。输出消息具有以下结构：

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

许可证

IoT Ethernet IP 协议适配器连接器包含以下第三方软件/许可：

- 以太网/IP 客户端
- MapDB
- Elsa

该连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改	日期
2	此版本包含错误修复。	2021 年 12 月 23 日
1	首次发布。	2020 年 12 月 15 日

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”

物联网 SiteWise 连接器

物联网 SiteWise 连接器将本地设备和设备数据发送到中的资产属性AWS IoT SiteWise。您可以使用此连接器从多台 OPC-UA 服务器收集数据并将其发布到 IoT。 SiteWise连接器将数据发送到当前 AWS 账户 和区域中的资产属性。

> **Note**
>
> 物联网 SiteWise 是一项完全托管的服务，用于收集、处理和可视化来自工业设备和设备的数据。您可以配置资产属性，以处理从此连接器发送到资产的测量属性的原始数据。例如，您可以定义将设备的摄氏温度数据点转换为华氏度的转换属性，也可以定义计算平均每小时温

度的指标属性。有关更多信息，请参阅AWS IoT SiteWise《用户指南》中的什么是AWS IoT SiteWise？。

连接器 SiteWise 使用从 OPC-UA 服务器发送的 OPC-UA 数据流路径向物联网发送数据。例如，数据流路径 `/company/windfarm/3/turbine/7/temperature` 可能表示风电场 #3 中涡轮机 #7 的温度传感器。如果 AWS IoT Greengrass 核心断开与 Internet 的连接，连接器会缓存数据，直到它能够成功连接到 AWS Cloud。您可以配置用于缓存数据的最大磁盘缓冲区大小。如果缓存大小超过最大磁盘缓冲区大小，则连接器会丢弃队列中最旧的数据。

配置和部署 IoT SiteWise 连接器后，可以在物联网控制台中添加网关和 OPC-UA 源。 SiteWise 在控制台中配置源时，您可以筛选物联 SiteWise 网连接器发送的 OPC-UA 数据流路径或添加前缀。有关完成网关和来源设置的说明，请参阅《AWS IoT SiteWise 用户指南》中的添加网关。

IoT 仅从您映射到物联网 SiteWise 资产测量属性的数据流中 SiteWise 接收数据。要将数据流映射到资产属性，您可以将属性的别名设置为等效于 OPC-UA 数据流路径。要了解如何定义资产模型和创建资产，请参阅《AWS IoT SiteWise 用户指南》中的工业资产建模。

> **注意事项**
>
> 您可以使用流管理器将数据 SiteWise 从 OPC-UA 服务器以外的来源上传到 IoT。流管理器还为持久性和带宽管理提供可自定义的支持。有关更多信息，请参阅 管理数据流。
> 此连接器在无容器隔离模式下运行，因此，您可以将其部署到在 Docker 容器中运行的 Greengrass 组。

此连接器具有以下版本。

版本	ARN
12（推荐）	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/12`
11	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/11`

版本	ARN
10	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/10
9	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/9
8	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/8
7	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/7
6	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/6
5	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/5
4	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/4
3	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/3
2	arn:aws:greengrass:*region*::/connectors/IoTSiteWise/versions/2

版本	ARN
1	arn:aws:greengrass: *region*::/connectors/IoTSiteWise/versions/ 1

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 9, 10, 11, and 12

> **Important**
> 此版本引入了新要求：AWS IoT Greengrass Core 软件 v1.10.2 和流管理器。

- AWS IoT Greengrass Core 软件 v1.10.2。
- Greengrass 组上启用的流管理器 。
- Java 8 安装在核心设备上并已添加到 PATH 环境变量中。
- 此连接器只能在同时支持两者AWS IoT Greengrass以及物联 SiteWise网的 Amazon Web Services 区域中使用。
- 将 IAM 策略添加到 Greengrass 组角色。此角色允许 AWS IoT Greengrass 组对目标 root 资产及其子级执行 `iotsitewise:BatchPutAssetPropertyValue` 操作，如以下示例所示。您可以`Condition`从策略中移除，以允许连接器访问您的所有 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/*"
                    ]
                }
            }
        }
    ]
}
```

有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

Versions 6, 7, and 8

> **Important**
>
> 此版本引入了新要求：AWS IoT Greengrass Core 软件 v1.10.0 和流管理器。

- AWS IoT Greengrass Core 软件 v1.10.0。
- Greengrass 组上启用的流管理器 。
- Java 8 安装在核心设备上并已添加到 PATH 环境变量中。
- 此连接器只能在同时支持两者AWS IoT Greengrass以及物联 SiteWise网的 Amazon Web Services 区域中使用。
- 将 IAM 策略添加到 Greengrass 组角色。此角色允许 AWS IoT Greengrass 组对目标 root 资产及其子级执行 `iotsitewise:BatchPutAssetPropertyValue` 操作，如以下示例所示。您可以`Condition`从策略中移除，以允许连接器访问您的所有 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/*"
                    ]
                }
            }
        }
    ]
}
```

有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

Version 5

- AWS IoT Greengrass Core 软件版本 v1.9.4。
- Java 8 安装在核心设备上并已添加到 PATH 环境变量中。
- 此连接器只能在同时支持两者AWS IoT Greengrass以及物联 SiteWise网的 Amazon Web Services 区域中使用。
- 将 IAM 策略添加到 Greengrass 组角色。此角色允许 AWS IoT Greengrass 组对目标 root 资产及其子级执行 `iotsitewise:BatchPutAssetPropertyValue` 操作，如以下示例所示。您可以`Condition`从策略中移除，以允许连接器访问您的所有 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/*"
                     ]
                 }
             }
        }
    ]
}
```

有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

Version 4

- AWS IoT Greengrass Core 软件 v1.10.0。
- Java 8 安装在核心设备上并已添加到 PATH 环境变量中。
- 此连接器只能在同时支持两者AWS IoT Greengrass以及物联 SiteWise网的 Amazon Web Services 区域中使用。
- 将 IAM 策略添加到 Greengrass 组角色。此角色允许 AWS IoT Greengrass 组对目标 root 资产及其子级执行 `iotsitewise:BatchPutAssetPropertyValue` 操作，如以下示例所示。您可以`Condition`从策略中移除，以允许连接器访问您的所有 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/*"
                     ]
                 }
             }
        }
    ]
}
```

有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

Version 3

- AWS IoT Greengrass Core 软件版本 v1.9.4。
- Java 8 安装在核心设备上并已添加到 PATH 环境变量中。
- 此连接器只能在同时支持两者AWS IoT Greengrass以及物联 SiteWise网的 Amazon Web Services 区域中使用。

- 将 IAM 策略添加到 Greengrass 组角色。此角色允许 AWS IoT Greengrass 组对目标 root 资产及其子级执行 `iotsitewise:BatchPutAssetPropertyValue` 操作，如以下示例所示。您可以`Condition`从策略中移除，以允许连接器访问您的所有 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/*"
                     ]
                 }
             }
        }
    ]
}
```

有关更多信息，请参阅 IAM 用户指南中的添加和删除 IAM 策略。

Versions 1 and 2

- AWS IoT Greengrass Core 软件版本 v1.9.4。
- Java 8 安装在核心设备上并已添加到 PATH 环境变量中。
- 此连接器只能在同时支持两者AWS IoT Greengrass以及物联 SiteWise网的 Amazon Web Services 区域中使用。
- 添加到 Greengrass 组角色的 IAM 策略，允许访问 AWS IoT Core 以及对目标 root 资产及其子级执行 `iotsitewise:BatchPutAssetPropertyValue` 操作，如以下示例所示。您可以`Condition`从策略中移除，以允许连接器访问您的所有 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/*"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect",
                "iot:DescribeEndpoint",
                "iot:Publish",
                "iot:Receive",
                "iot:Subscribe"
            ],
            "Resource": "*"
        }
    ]
}
```

有关更多信息，请参阅《 IAM 用户指南》中的添加和删除 IAM 身份权限。

参数

Versions 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12

`SiteWiseLocalStoragePath`

AWS IoT Greengrass主机上的目录，物联网 SiteWise 连接器可以向其写入永久数据。默认值为 `/var/sitewise`。

AWS IoT 控制台中的显示名称：本地存储路径

必需：`false`

类型：`string`

有效模式：`^\s*$|\/.`

`AWSSecretsArnList`

AWS Secrets Manager 中的秘密列表，其中包含 OPC-UA 用户名和密码密钥值对。每个秘密都必须是密钥值对类型秘密。

在 AWS IoT 控制台显示名称：OPC-UA 用户名/密码秘密的 ARN 列表

必需：`false`

类型：`JsonArrayOfStrings`

有效模式：`\[( ?,? ?\"(arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\]+\/)*[a-zA-Z0-9\/_+=,.@\\-]+-[a-zA-Z0-9]+)*\")*\]`

`MaximumBufferSize`

物联网 SiteWise 磁盘使用量的最大大小（以 GB 为单位）。默认值为 10GB。

AWS IoT 控制台中的显示名称：最大磁盘缓冲区大小

必需：`false`

类型：`string`

有效模式：`^\s*$|[0-9]+`

Version 1

`SiteWiseLocalStoragePath`

AWS IoT Greengrass主机上的目录，物联网 SiteWise 连接器可以向其写入永久数据。默认值为 `/var/sitewise`。

AWS IoT 控制台中的显示名称：本地存储路径

必需：`false`

类型：`string`

有效模式：`^\s*$|\/.`

`SiteWiseOpcuaUserIdentityTokenSecretArn`

AWS Secrets Manager 中的密钥，其中包含 OPC-UA 用户名和密码密钥值对。此密钥必须是密钥值对类型密钥。

AWS IoT 控制台中的显示名称：OPC-UA 用户名/密码密钥的 ARN

必需：`false`

类型：`string`

有效模式：`^$|arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\]+/)*[a-zA-Z0-9/_+=,.@\\-]+-[a-zA-Z0-9]+`

`SiteWiseOpcuaUserIdentityTokenSecretArn-ResourceId`

AWS IoT Greengrass 组中引用 OPC-UA 用户名和密码密钥的私有资源。

AWS IoT 控制台中的显示名称：OPC-UA 用户名/密码密钥资源

必需：`false`

类型：`string`

有效模式：`^$|.+`

`MaximumBufferSize`

物联网 SiteWise 磁盘使用量的最大大小（以 GB 为单位）。默认值为 10GB。

AWS IoT 控制台中的显示名称：最大磁盘缓冲区大小

必需：`false`

类型：`string`

有效模式：`^\s*$|[0-9]+`

创建连接器示例 (AWS CLI)

以下AWS CLI命令`ConnectorDefinition`使用包含 IoT SiteWise 连接器的初始版本创建一个。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
```

```
            "Id": "MyIoTSiteWiseConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTSiteWise/
versions/11"
        }
    ]
}'
```

> **Note**
>
> 此连接器中的 Lambda 函数的生命周期很长。

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅 the section called “连接器入门（控制台）”。

输入数据

该连接器不接受 MQTT 消息作为输入数据。

输出数据

此连接器不发布 MQTT 消息作为输出数据。

限制

此连接器受物联网施加的以下所有限制的约束 SiteWise，包括以下限制。有关更多信息，请参阅 AWS 一般参考 中的 AWS IoT SiteWise 端点和配额。

- 每个 AWS 账户 的最大网关数量。
- 每个网关的 OPC-UA 源的最大数量。
- 存储的 timestamp-quality-value (TQV) 数据点的最大速率。AWS 账户
- 每个资产属性存储的 TQV 数据点的最大速率。

许可证

Version 9, 10, 11, and 12

物联网 SiteWise 连接器包括以下第三方软件/许可：

- MapDB

- Elsa
- Eclipse Milo

此连接器在 Greengrass Core 软件许可协议下发布。

Versions 6, 7, and 8

物联网 SiteWise 连接器包括以下第三方软件/许可：

- Milo / EDL 1.0

此连接器在 Greengrass Core 软件许可协议下发布。

Versions 1, 2, 3, 4, and 5

物联网 SiteWise 连接器包括以下第三方软件/许可：

- Milo / EDL 1.0
- Chronicle-Queue / Apache 许可证 2.0

此连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改	Date
12	• 此版本包含错误修复。	2021 年 12 月 22 日
11	• 支持含有隐藏字符或不可打印字符的字符串。在将字符串发送到 AWS Cloud 之前，自动删除隐藏和不可打印的字符。 • 修复了导致 IoT SiteWise 网关无限次重试无效请求的问题。	2021 年 3 月 24 日

版本	更改	Date
	• 修复了在物联 SiteWise 网网关连接到高频数据源时导致检查点损坏的问题。 • 改进了错误消息，以帮助对网关配置进行故障排除。	
10	配置了 `StreamManager` ，以改进源连接丢失并重新建立时的处理能力。当没有可用的 `SourceTimestamp` 时，此版本也接受 OPC-UA 值的 `ServerTimestamp` 。	2021 年 1 月 22 日
9	支持自定义 Greengrass `StreamManager` 流目标、OPC-UA 死区、自定义扫描模式和自定义扫描速率。还包括在从 IoT SiteWise 网关进行配置更新期间提高性能。	2020 年 12 月 15 日
8	提高了连接器遇到间歇性网络连接时的稳定性。	2020 年 11 月 19 日
7	修复了网关指标的问题。	2020 年 8 月 14 日
6	增加了对 CloudWatch 指标和自动发现新 OPC-UA 标签的支持。此版本需要流管理器和 AWS IoT Greengrass Core 软件 v1.10.0 或更高版本。	2020 年 4 月 29 日
5	修复了与 AWS IoT Greengrass Core 软件 v1.9.4 的兼容性问题。	2020 年 2 月 12 日

版本	更改	Date
4	修复了 OPC-UA 服务器重新连接的问题。	2020 年 2 月 7 日
3	已删除 `iot:*` 权限要求。	2019 年 12 月 17 日
2	新增对多个 OPC-UA 秘密资源的支持。	2019 年 12 月 10 日
1	首次发布。	2019 年 12 月 2 日

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”
- 请参阅 AWS IoT SiteWise 用户指南 中的以下主题：
  - 什么是 AWS IoT SiteWise？
  - 使用网关
  - 网关 CloudWatch 指标
  - 对物联网网 SiteWise 关进行故障排除

Kinesis Firehose

Kinesis Firehose 连接器通过亚马逊数据 Firehose 传输流将数据发布到亚马逊 S3、亚马逊 Redshift 或亚马逊服务等目的地。 OpenSearch

此连接器是 Kinesis 传输流的数据创建器。它接收 MQTT 主题上的输入数据，并将这些数据发送到指定的传输流。然后，该传输流将数据记录发送到已配置的目标（例如，S3 存储桶）。

此连接器具有以下版本。

版本	ARN
5	arn:aws:greengrass:*region*::/connectors/KinesisFirehose/versions/5
4	arn:aws:greengrass:*region*::/connectors/KinesisFirehose/versions/4
3	arn:aws:greengrass:*region*::/connectors/KinesisFirehose/versions/3
2	arn:aws:greengrass:*region*::/connectors/KinesisFirehose/versions/2
1	arn:aws:greengrass:*region*::/connectors/KinesisFirehose/versions/1

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 4 - 5

- AWS IoT Greengrass 核心软件 v1.9.3 或更高版本。
- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

> **Note**
>
> 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- 一个已配置的 Kinesis 传输流。有关更多信息，请参阅亚马逊 Kinesis Firehose 开发者指南中的创建亚马逊数据 Firehose 传输流。
- Greengrass 组角色，配置为允许对目标传输流执行 `firehose:PutRecord` 和 `firehose:PutRecordBatch` 操作，如以下示例 IAM policy 中所示。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
    ]
 }
```

利用此连接器，您可以动态覆盖输入消息负载中的默认传输流。如果您的实施使用此功能，则 IAM policy 应包括所有目标流作为资源。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）。

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

Versions 2 - 3

- AWS IoT Greengrass 核心软件 v1.7 或更高版本。
- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。

- 一个已配置的 Kinesis 传输流。有关更多信息，请参阅亚马逊 Kinesis Firehose 开发者指南中的创建亚马逊数据 Firehose 传输流。
- Greengrass 组角色，配置为允许对目标传输流执行 `firehose:PutRecord` 和 `firehose:PutRecordBatch` 操作，如以下示例 IAM policy 中所示。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
    ]
 }
```

利用此连接器，您可以动态覆盖输入消息负载中的默认传输流。如果您的实施使用此功能，则 IAM policy 应包括所有目标流作为资源。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）。

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

Version 1

- AWS IoT Greengrass 核心软件 v1.7 或更高版本。
- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- 一个已配置的 Kinesis 传输流。有关更多信息，请参阅亚马逊 Kinesis Firehose 开发者指南中的创建亚马逊数据 Firehose 传输流。
- Greengrass 组角色，配置为允许对目标传输流执行 `firehose:PutRecord` 操作，如以下示例 IAM policy 中所示。

```
{
```

```
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:firehose:region:account-id:deliverystream/stream-name"
            ]
        }
    ]
}
```

利用此连接器，您可以动态覆盖输入消息负载中的默认传输流。如果您的实施使用此功能，则 IAM policy 应包括所有目标流作为资源。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）。

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

连接器参数

该连接器提供以下参数：

Versions 5

`DefaultDeliveryStreamArn`

要向其发送数据的默认 Firehose 传输流的 ARN。目标流可由输入消息负载中的 `delivery_stream_arn` 属性覆盖。

> **Note**
>
> 组角色必须允许对所有目标传输流执行适当的操作。有关更多信息，请参阅the section called “要求”。

AWS IoT 控制台中的显示名称：默认传送流 ARN

必需：true

类型：string

有效模式：arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$

DeliveryStreamQueueSize

在同一传输流的新记录被拒绝之前要保留在内存中的记录的最大数目。最小值为 2000。

AWS IoT 控制台中的显示名称：要缓冲的最大记录数（每个流）

必需：true

类型：string

有效模式：^([2-9]\\d{3}|[1-9]\\d{4,})$

MemorySize

要分配给此连接器的内存量（以 KB 为单位）。

AWS IoT 控制台中的显示名称：内存大小

必需：true

类型：string

有效模式：^[0-9]+$

PublishInterval

将记录发布到 Firehose 的时间间隔（以秒为单位）。要禁用批处理，请将此值设置为 0。

AWS IoT 控制台中的显示名称：发布间隔

必需：true

类型：string

有效值：0 - 900

有效模式：[0-9]|[1-9]\\d|[1-9]\\d\\d|900

`IsolationMode`

此连接器的容器化模式。默认值为`GreengrassContainer`，这意味着连接器在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

> **Note**
>
> 组的默认容器化设置不适用于连接器。

AWS IoT 控制台中的显示名称：容器隔离模式

必需：`false`

类型：`string`

有效值：`GreengrassContainer` 或 `NoContainer`

有效模式：`^NoContainer$|^GreengrassContainer$`

Versions 2 - 4

`DefaultDeliveryStreamArn`

要向其发送数据的默认 Firehose 传输流的 ARN。目标流可由输入消息负载中的 `delivery_stream_arn` 属性覆盖。

> **Note**
>
> 组角色必须允许对所有目标传输流执行适当的操作。有关更多信息，请参阅the section called “要求”。

AWS IoT 控制台中的显示名称：默认传送流 ARN

必需：`true`

类型：`string`

有效模式：`arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`DeliveryStreamQueueSize`

在同一传输流的新记录被拒绝之前要保留在内存中的记录的最大数目。最小值为 2000。

AWS IoT 控制台中的显示名称：要缓冲的最大记录数（每个流）

必需：`true`

类型：`string`

有效模式：`^([2-9]\\d{3}|[1-9]\\d{4,})$`

`MemorySize`

要分配给此连接器的内存量（以 KB 为单位）。

AWS IoT 控制台中的显示名称：内存大小

必需：`true`

类型：`string`

有效模式：`^[0-9]+$`

`PublishInterval`

将记录发布到 Firehose 的时间间隔（以秒为单位）。要禁用批处理，请将此值设置为 0。

AWS IoT 控制台中的显示名称：发布间隔

必需：`true`

类型：`string`

有效值：`0 - 900`

有效模式：`[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

Version 1

`DefaultDeliveryStreamArn`

要向其发送数据的默认 Firehose 传输流的 ARN。目标流可由输入消息负载中的 `delivery_stream_arn` 属性覆盖。

> **Note**
>
> 组角色必须允许对所有目标传输流执行适当的操作。有关更多信息，请参阅the section called “要求”。

AWS IoT 控制台中的显示名称：默认传送流 ARN

必需：`true`

类型：`string`

有效模式：`arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

Example

创建连接器示例 (AWS CLI)

以下 CLI 命令将创建一个 `ConnectorDefinition`，其初始版本包含该连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyKinesisFirehoseConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/KinesisFirehose/
versions/5",
            "Parameters": {
                "DefaultDeliveryStreamArn": "arn:aws:firehose:region:account-
id:deliverystream/stream-name",
                "DeliveryStreamQueueSize": "5000",
                "MemorySize": "65535",
                "PublishInterval": "10",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

在 AWS IoT Greengrass 控制台中，您可以从群组的 “连接器” 页面添加连接器。有关更多信息，请参阅the section called “连接器入门（控制台）”。

输入数据

此连接器接受 MQTT 主题上的流内容，然后将这些内容发送到目标传输流。它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。

Versions 2 - 5

主题筛选条件：kinesisfirehose/message

使用此主题发送包含 JSON 数据的消息。

消息属性

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。

必需：true

类型：包含以下属性的 object：

data

要发送到传输流的数据。

必需：true

类型：string

delivery_stream_arn

目标 Kinesis 传输流的 ARN。包含此属性是为了覆盖默认传输流。

必需：false

类型：string

有效模式：arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$

id

请求的任意 ID。此属性用于将输入请求映射到输出响应。如果指定，响应对象中的 id 属性将设置为该值。如果您不使用此功能，可以忽略此属性或指定空字符串。

必需：false

类型：string

有效模式：.*

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"
}
```

主题筛选条件：kinesisfirehose/message/binary/#

使用此主题发送包含二进制数据的消息。连接器不会解析二进制数据。该数据将按原样进行流式传输。

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/。请务必包含尾斜杠。

Version 1

主题筛选条件：kinesisfirehose/message

使用此主题发送包含 JSON 数据的消息。

消息属性

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。

必需：true

类型：包含以下属性的 object：

data

要发送到传输流的数据。

必需：true

类型：string

delivery_stream_arn

目标 Kinesis 传输流的 ARN。包含此属性是为了覆盖默认传输流。

必需：false

类型：string

有效模式：arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$

id

请求的任意 ID。此属性用于将输入请求映射到输出响应。如果指定，响应对象中的 id 属性将设置为该值。如果您不使用此功能，可以忽略此属性或指定空字符串。

必需：false

类型：string

有效模式：.*

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"
}
```

主题筛选条件：kinesisfirehose/message/binary/#

使用此主题发送包含二进制数据的消息。连接器不会解析二进制数据。该数据将按原样进行流式传输。

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/。请务必包含尾斜杠。

输出数据

此连接器将状态信息发布为 MQTT 主题的输出数据。

Versions 2 - 5

订阅中的主题筛选条件

kinesisfirehose/message/status

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**
>
> 如果连接器检测到可重试的错误（如连接错误），它会在下一批次中重试发布。指数退避由 SDK 处理。 AWS 失败并带可重试的错误的请求将添加回队列的结尾以供进一步发布。

Version 1

订阅中的主题筛选条件

```
kinesisfirehose/message/status
```

示例输出：成功

```
{
   "response": {
       "firehose_record_id": "1lxfuuuFomkpJYzt/34ZU/r8JYPf8Wyf7AXqlXm",
       "status": "success"
    },
    "id": "request123"
}
```

示例输出：失败

```
{
   "response" : {
       "error": "ResourceNotFoundException",
       "error_message": "An error occurred (ResourceNotFoundException) when
 calling the PutRecord operation: Firehose test1 not found under account
 123456789012.",
```

```
        "status": "fail"
    },
    "id": "request123"
}
```

用法示例

使用以下概括步骤设置可用于尝试连接器的示例 Python 3.7 Lambda 函数。

> **Note**
>
> - 如果要使用其他 Python 运行时，您可以创建从 Python3.x 到 Python 3.7 的符号链接。
> - 连接器入门（控制台）和 连接器入门 (CLI) 主题包含详细步骤，说明如何配置和部署示例 Twilio 通知连接器。

1. 确保满足连接器的要求。

   对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。
2. 创建并发布将输入数据发送到连接器的 Lambda 函数。

   将示例代码保存为 PY 文件。下载并解压适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。然后，创建一个 zip 包，其中在根级别包含 PY 文件和 `greengrasssdk` 文件夹。此 zip 包是您上传到 AWS Lambda 的部署包。

   创建 Python 3.7 Lambda 函数后，请发布函数版本并创建别名。
3. 配置 Greengrass 组。

   a. 通过别名来添加 Lambda 函数（推荐）。将 Lambda 生命周期配置为长时间生存（或在 CLI 中设置为 `"Pinned": true`）。

   b. 添加连接器并配置其参数。

   c. 添加允许连接器接收 JSON 输入数据并针对支持的主题筛选条件发送输出数据的订阅。

      - 将 Lambda 函数设置为源，将连接器设置为目标，并使用支持的输入主题筛选条件。
      - 将连接器设置为源，将 AWS IoT Core 设置为目标，并使用支持的输出主题筛选条件。您可以使用此订阅在 AWS IoT 控制台中查看状态消息。

4. 部署组。
5. 在 AWS IoT 控制台的 “测试” 页面上，订阅输出数据主题以查看来自连接器的状态消息。示例 Lambda 函数是长时间生存的，并且在部署组后立即开始发送消息。

   完成测试后，您可以将 Lambda 生命周期设置为按需（或在 CLI 中设置为 `"Pinned": false`）并部署组。这会阻止函数发送消息。

示例

以下示例 Lambda 函数向连接器发送一条输入消息。此消息包含 JSON 数据。

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'kinesisfirehose/message'

def create_request_with_all_fields():
    return  {
        "request": {
            "data": "Message from Firehose Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

许可证

Kinesis Firehose 连接器包含以下第三方软件/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

该连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
5	增加了用于配置连接器容器化模式的 `IsolationMode` 参数。
4	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。
3	修复以减少过多的日志记录，并修复了其他较小的错误。
2	增加了按指定间隔向 Firehose 发送批量数据记录的支持。 • 还需要在组角色中执行 `firehose:PutRecordBatch` 操作。 • 新的 `MemorySize` 、`DeliveryStreamQueueSize` 和 `PublishInterval` 参数。 • 输入消息包含一组已发布的数据记录的状态响应。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”
- Amazon Kinesis 开发人员指南中的什么是 Amazon Kinesis Data Firehose？

ML 反馈连接器

> **Warning**
>
> 此连接器已进入生命周期延长阶段，AWS IoT Greengrass 不会发布更新来提供功能、现有功能增强、安全补丁或错误修复。有关更多信息，请参阅AWS IoT Greengrass Version 1 维护策略。

ML 反馈连接器使您可以更轻松地访问机器学习 (ML) 模型数据以进行模型重新训练和分析。该连接器：

- 将您的 ML 模型使用的输入数据（样本）上传到 Amazon S3。模型输入可以是任何格式，如图像、JSON 或音频。在将样本上传到云后，您可以使用它们来重新训练模型，以提高其预测的准确性和精确性。例如，您可以使用 SageMaker Ground Truth 来标记您的样本，并使用 SageMaker 来重新训练模型。
- 从模型将预测结果发布为 MQTT 消息。这可让您实时监控和分析模型的推理质量。您也可以存储预测结果，然后使用它们来分析随时间推移的趋势。
- 将有关样本上传和样本数据的指标发布到 Amazon CloudWatch。

要配置此连接器，您以 JSON 格式描述支持的反馈配置。反馈配置定义诸如目标 Amazon S3 存储桶、内容类型和采样策略之类的属性。（采样策略用于确定要上传哪些样本。）

您可以在以下情况下使用 ML 反馈连接器：

- 用户定义的 Lambda 函数。您的本地推理 Lambda 函数使用 AWS IoT Greengrass 机器学习开发工具包调用此连接器并传入目标反馈配置、模型输入和模型输出（预测结果）。有关示例，请参阅 the section called “用法示例”。
- 使用 ML 图像分类连接器 (v2)。要将此连接器与 ML 图像分类连接器一起使用，请配置 ML 图像分类连接器的 `MLFeedbackConnectorConfigId` 参数。
- 使用 ML 对象检测连接器。要将此连接器与 ML 对象检测连接器一起使用，请配置 ML 对象检测连接器的 `MLFeedbackConnectorConfigId` 参数。

ARN：`arn:aws:greengrass:region::/connectors/MLFeedback/versions/1`

要求

此连接器具有以下要求：

- AWS IoT Greengrass Core 软件 v1.9.3 版或更高版本。
- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

> **Note**
>
> 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- 一个或多个 Amazon S3 桶。您使用的存储桶数量取决于您的采样策略。
- Greengrass 组角色，配置为允许对目标 Amazon S3 存储桶中的对象执行 `s3:PutObject` 操作，如以下示例 IAM 策略所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:PutObject",
            "Resource": [
                "arn:aws:s3:::bucket-name/*"
```

```
            ]
        }
    ]
}
```

该策略应包括所有目标存储桶作为资源。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）。

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅 the section called “管理组角色（控制台）” 或 the section called “管理组角色 (CLI)”。

- 添加到 Greengrass 组且已配置的 CloudWatch Metrics 连接器。仅当您要使用指标报告功能时才需要它。
- 与此连接器进行交互时需要 AWS IoT Greengrass 机器学习开发工具包 v1.1.0。

参数

`FeedbackConfigurationMap`

此连接器可用于将样本上传到 Amazon S3 的一组一个或多个反馈配置。反馈配置定义诸如目标存储桶、内容类型和采样策略之类的参数。调用此连接器时，调用 Lambda 函数或连接器将指定目标反馈配置。

AWS IoT 控制台中的显示名称：反馈配置图

必需：`true`

类型：一个格式正确的 JSON 字符串，用于定义支持的反馈配置集。有关示例，请参阅 the section called “FeedbackConfigurationMap 示例”。

反馈配置对象的 ID 具有以下要求。

该 ID：

- 在配置对象间必须唯一。
- 必须以字母或数字开头。可以包含大小写字母、数字和连字符。
- 长度必须介于 2 - 63 个字符之间。

必需：`true`

类型：string

有效模式：^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$

示例：MyConfig0、config-a、12id。

反馈配置对象的主体包含以下属性。

s3-bucket-name

目标 Amazon S3 存储桶的名称。

> **Note**
>
> 组角色必须允许对所有目标存储桶执行 s3:PutObject 操作。有关更多信息，请参阅the section called “要求”。

必需：true

类型：string

有效模式：^[a-z0-9\.\-]{3,63}$

content-type

要上传的样本的内容类型。单个反馈配置的所有内容必须属于同一类型。

必需：true

类型：string

示例：image/jpeg、application/json、audio/ogg。

s3-prefix

用于上传的样本的键前缀。前缀类似于目录名称。它使您能够在存储桶的同一目录下存储相似的数据。有关更多信息，请参阅 Amazon Simple Storage Service 用户指南中的对象键和元数据。

必需：false

类型：string

`file-ext`

要用于上传的样本的文件扩展名。必须是内容类型的有效文件扩展名。

必需：`false`

类型：`string`

示例：`jpg`、`json`、`ogg`。

`sampling-strategy`

用于过滤要上传的样本的采样策略。如果省略，连接器尝试上传它收到的所有样本。

必需：`false`

类型：一个格式正确的 JSON 字符串，其中包含以下属性。

`strategy-name`

采样策略的名称。

必需：`true`

类型：`string`

有效值：`RANDOM_SAMPLING`、`LEAST_CONFIDENCE`、`MARGIN` 或 `ENTROPY`

`rate`

随机采样策略的速率。

如果 `strategy-name` 为 `RANDOM_SAMPLING`，则 `true` 是必需的。

类型：`number`

有效值：`0.0 - 1.0`

`threshold`

最小置信度、裕度或熵采样策略的阈值。

如果 `strategy-name` 为 `LEAST_CONFIDENCE`、`MARGIN` 或 `ENTROPY`，则 `true` 是必需的。

类型：number

有效值：

- 对于 LEAST_CONFIDENCE 或 MARGIN 策略，为 0.0 - 1.0。
- 对于 ENTROPY 策略，为 0.0 - no limit。

RequestLimit

连接器一次可以处理的最大请求数。

您可以使用此参数，通过限制连接器同时处理的请求数来限制内存消耗。超过此限制的请求将被忽略。

AWS IoT 控制台中的显示名称：请求限制

必需：false

类型：string

有效值：0 - 999

有效模式：^$|^[0-9]{1,3}$

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 ConnectorDefinition，其初始版本包含 ML 反馈连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyMLFeedbackConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/MLFeedback/
versions/1",
            "Parameters": {
                "FeedbackConfigurationMap": "{  \"RandomSamplingConfiguration\":
 {  \"s3-bucket-name\": \"my-aws-bucket-random-sampling\",  \"content-type\":
 \"image/png\",  \"file-ext\": \"png\",  \"sampling-strategy\": {  \"strategy-name
\": \"RANDOM_SAMPLING\",  \"rate\": 0.5  } },  \"LeastConfidenceConfiguration\": {
  \"s3-bucket-name\": \"my-aws-bucket-least-confidence-sampling\",  \"content-type\":
 \"image/png\",  \"file-ext\": \"png\",  \"sampling-strategy\": {  \"strategy-name\":
 \"LEAST_CONFIDENCE\",  \"threshold\": 0.4  } } }",
```

```
                "RequestLimit": "10"
            }
        }
    ]
}'
```

FeedbackConfigurationMap 示例

以下是 FeedbackConfigurationMap 参数的扩展示例值。本示例包括使用不同采样策略的几种反馈配置。

```
{
    "ConfigID1": {
        "s3-bucket-name": "my-aws-bucket-random-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "RANDOM_SAMPLING",
            "rate": 0.5
        }
    },
    "ConfigID2": {
        "s3-bucket-name": "my-aws-bucket-margin-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "MARGIN",
            "threshold": 0.4
        }
    },
    "ConfigID3": {
        "s3-bucket-name": "my-aws-bucket-least-confidence-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "LEAST_CONFIDENCE",
            "threshold": 0.4
        }
    },
    "ConfigID4": {
        "s3-bucket-name": "my-aws-bucket-entropy-sampling",
        "content-type": "image/png",
        "file-ext": "png",
```

```
        "sampling-strategy": {
            "strategy-name": "ENTROPY",
            "threshold": 2
        }
    },
    "ConfigID5": {
        "s3-bucket-name": "my-aws-bucket-no-sampling",
        "s3-prefix": "DeviceA",
        "content-type": "application/json"
    }
}
```

采样策略

连接器支持四种采样策略，这些策略确定是否上传传递给连接器的样本。样本是模型用于预测的数据的离散实例。您可以使用采样策略来筛选最有可能提高模型准确性的样本。

RANDOM_SAMPLING

根据提供的速率随机上传样本。如果随机生成的值小于该速率，则它将上传样本。速率越高，上传的样本越多。

> **Note**
>
> 此策略不考虑所提供的任何模型预测。

LEAST_CONFIDENCE

上传其最大置信概率低于提供的阈值的样本。

示例方案：

阈值：`.6`

模型预测：`[.2, .2, .4, .2]`

最大置信概率：`.4`

结果：

使用此样本，因为最大置信概率 (`.4`) <= 阈值 (`.6`)。

`MARGIN`

如果最高的两个置信概率之间的裕度落在提供的阈值内，则上传样本。裕度是最高的两个概率之间的差。

示例方案：

阈值：`.02`

模型预测：`[.3, .35, .34, .01]`

最高的两个置信概率：`[.35, .34]`

裕度：`.01` (`.35 - .34`)

结果：

使用此样本，因为裕度 (`.01`) <= 阈值 (`.02`)。

`ENTROPY`

上传其熵大于提供的阈值的样本。使用模型预测的标准化熵。

示例方案：

阈值：`0.75`

模型预测：`[.5, .25, .25]`

预测的熵：`1.03972`

结果：

使用此样本，因为熵 `1.03972`> 阈值`0.75`。

输入数据

用户定义的 Lambda 函数使用 AWS IoT Greengrass 机器学习开发工具包中的 `feedback` 客户端的 `publish` 函数来调用连接器。有关示例，请参阅 the section called “用法示例”。

> **Note**
>
> 该连接器不接受 MQTT 消息作为输入数据。

`publish` 函数采用下列参数：

ConfigId

目标反馈配置的 ID。这必须与在 ML 反馈连接器的 FeedbackConfigurationMap 参数中定义的反馈配置的 ID 相匹配。

所需：是

类型：字符串

ModelInput

传递给模型以进行推理的输入数据。除非根据采样策略将此输入数据过滤掉，否则将使用目标配置上传此数据。

所需：是

类型：字节

ModelPrediction

来自模型的预测结果。结果类型可以是词典或列表。例如，来自 ML 图像分类连接器的预测结果是概率列表（例如 `[0.25, 0.60, 0.15]`）。此数据发布到 `/feedback/message/prediction` 主题。

所需：是

类型：词典或 `float` 值的列表

元数据

客户定义的、特定于应用程序的元数据，该元数据附加到上传的样本并发布到 `/feedback/message/prediction` 主题。连接器还将具有时间戳值的 `publish-ts` 键插入元数据。

必填项：false

类型：字典

示例：`{"some-key": "some value"}`

输出数据

此连接器将数据发布到三个 MQTT 主题：

- 来自连接器的关于 feedback/message/status 主题的状态信息。
- 有关 feedback/message/prediction 主题的预测结果。
- 用于 cloudwatch/metric/put 主题上的 CloudWatch 的指标。

您必须配置订阅以允许连接器就 MQTT 主题进行通信。有关更多信息，请参阅the section called “输入和输出”。

主题筛选条件：feedback/message/status

使用此主题监控样本上传和已丢弃样本的状态。连接器每次收到请求时都会发布到该主题。

示例输出：样本上传成功


```
{
  "response": {
    "status": "success",
    "s3_response": {
      "ResponseMetadata": {
        "HostId": "IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK
+Jd1vEXAMPLEa3Km",
        "RetryAttempts": 1,
        "HTTPStatusCode": 200,
        "RequestId": "79104EXAMPLEB723",
        "HTTPHeaders": {
          "content-length": "0",
          "x-amz-id-2":
 "lbbqaDVFOhMlyU3gRvAX1ZIdg8P0WkGkCSSFsYFvSwLZk3j7QZhG5EXAMPLEdd4/pEXAMPLEUqU=",
          "server": "AmazonS3",
          "x-amz-expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\",
 rule-id=\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
          "x-amz-request-id": "79104EXAMPLEB723",
          "etag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
          "date": "Thu, 11 Jul 2019 00:12:50 GMT",
          "x-amz-server-side-encryption": "AES256"
        }
      },
      "bucket": "greengrass-feedback-connector-data-us-west-2",
      "ETag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
      "Expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\", rule-id=
\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
      "key": "s3-key-prefix/UUID.file_ext",
      "ServerSideEncryption": "AES256"
```

```
    }
  },
  "id": "5aaa913f-97a3-48ac-5907-18cd96b89eeb"
}
```

连接器将 bucket 和 key 字段添加到来自 Amazon S3 的响应中。有关 Amazon S3 响应的更多信息，请参阅 Amazon Simple Storage Service API 参考 中的 PUT 对象。

示例输出：由于采样策略而丢弃的样本示例

```
{
  "response": {
    "status": "sample_dropped_by_strategy"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

示例输出：样本上传失败

失败状态包括错误消息作为 error_message 值和异常类 error 作为值。

```
{
  "response": {
    "status": "fail",
    "error_message": "[RequestId: 4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3] Failed
 to upload model input data due to exception. Model prediction will not be
 published. Exception type: NoSuchBucket, error: An error occurred (NoSuchBucket)
 when calling the PutObject operation: The specified bucket does not exist",
    "error": "NoSuchBucket"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

示例输出：由于请求限制，请求被限制

```
{
  "response": {
    "status": "fail",
    "error_message": "Request limit has been reached (max request: 10 ). Dropping
 request.",
    "error": "Queue.Full"
  },
```

```
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

主题筛选条件：`feedback/message/prediction`

使用本主题可侦听基于上传的样本数据的预测。这使您能够实时分析您的模型性能。仅当数据成功上传到 Amazon S3 时，模型预测才会发布到该主题。在此主题上发布的消息为 JSON 格式。它们包含指向上传的数据对象的链接、模型的预测以及请求中包含的元数据。

您也可以存储预测结果，然后使用它们来报告和分析随时间推移的趋势。趋势可以提供宝贵的见解。例如，准确性随时间降低趋势可以帮助您确定是否需要重新训练模型。

输出示例

```
{
  "source-ref": "s3://greengrass-feedback-connector-data-us-west-2/s3-key-prefix/
UUID.file_ext",
  "model-prediction": [
    0.5,
    0.2,
    0.2,
    0.1
  ],
  "config-id": "ConfigID2",
  "metadata": {
    "publish-ts": "2019-07-11 00:12:48.816752"
  }
}
```

> **Tip**
>
> 您可以将 IoT Analytics 连接器配置为订阅该主题，并将信息发送给 AWS IoT Analytics 以进行进一步分析或历史分析。

主题筛选条件：`cloudwatch/metric/put`

这是用于将指标发布到 CloudWatch 的输出主题。此功能要求您安装并配置 CloudWatch 指标连接器。

指标包括：

- 上传的样本的数量。
- 上传的样本的大小。
- 上传到 Amazon S3 时出现的错误数。
- 基于采样策略的丢弃样本数。
- 受限制请求的数目。

示例输出：数据样本的大小（在实际上传之前发布）

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 47592,
      "unit": "Bytes",
      "metricName": "SampleSize"
    }
  }
}
```

示例输出：样本上传成功

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadSuccess"
    }
  }
}
```

示例输出：成功上传样本并发布预测结果

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleAndPredictionPublished"
```

```
    }
  }
}
```

示例输出：样本上传失败

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadFailure"
    }
  }
}
```

示例输出：由于采样策略而丢弃的样本示例

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleNotUsed"
    }
  }
}
```

示例输出：由于请求限制，请求被限制

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "ErrorRequestThrottled"
    }
  }
}
```

用法示例

以下示例是一个用户定义的 Lambda 函数，该函数使用 AWS IoT Greengrass 机器学习开发工具包将数据发送到 ML 反馈连接器。

> **Note**
>
> 可以从 AWS IoT Greengrass 下载页面中下载 AWS IoT Greengrass 机器学习 SDK。

```
import json
import logging
import os
import sys
import greengrass_machine_learning_sdk as ml

client = ml.client('feedback')

try:
    feedback_config_id = os.environ["FEEDBACK_CONFIG_ID"]
    model_input_data_dir = os.environ["MODEL_INPUT_DIR"]
    model_prediction_str = os.environ["MODEL_PREDICTIONS"]
    model_prediction = json.loads(model_prediction_str)
except Exception as e:
    logging.info("Failed to open environment variables. Failed with exception:
{}".format(e))
    sys.exit(1)

try:
    with open(os.path.join(model_input_data_dir, os.listdir(model_input_data_dir)[0]),
 'rb') as f:
        content = f.read()
except Exception as e:
    logging.info("Failed to open model input directory. Failed with exception:
{}".format(e))
    sys.exit(1)

def invoke_feedback_connector():
    logging.info("Invoking feedback connector.")
    try:
        client.publish(
            ConfigId=feedback_config_id,
```

```
            ModelInput=content,
            ModelPrediction=model_prediction
        )
    except Exception as e:
        logging.info("Exception raised when invoking feedback connector:{}".format(e))
        sys.exit(1)

invoke_feedback_connector()

def function_handler(event, context):
    return
```

许可证

ML 反馈连接器包含以下第三方软件/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
- six/MIT

该连接器在 Greengrass Core 软件许可协议下发布。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”

ML 图像分类连接器

> **Warning**
>
> 此连接器已进入生命周期延长阶段，AWS IoT Greengrass 不会发布更新来提供功能、现有功能增强、安全补丁或错误修复。有关更多信息，请参阅AWS IoT Greengrass Version 1 维护策略。

ML 图像分类连接器提供在 AWS IoT Greengrass 核心上运行的机器学习 (ML) 推理服务。这一本地推理服务使用由 SageMaker 图像分类算法训练的模型执行图像分类。

用户定义的 Lambda 函数使用 AWS IoT Greengrass 机器学习开发工具包将推理请求提交给本地推理服务。该服务在本地运行推理，并返回输入映像属于特定类别的概率。

AWS IoT Greengrass 提供此连接器的以下版本，可用于多个平台。

Version 2

连接器	描述和 ARN
ML 图像分类 Aarch64 JTX2	适用于 NVIDIA Jetson TX2 的映像分类推理服务。支持 GPU 加速。 ARN：`arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationAarch64JTX2/versions/2`
ML 图像分类 x86_64	适用于 x86_64 平台的映像分类推理服务。 ARN：`arn:aws:greengrass:` *`region`*`::/connectors/ImageClassificationx86-64/versions/2`
ML 图像分类 ARMv7	适用于 ARMv7 平台的映像分类推理服务。 ARN：`arn:aws:greengrass:` *`region`*`::/connectors/Imag`

连接器	描述和 ARN
	eClassificationARMv7/versio ns/2

Version 1

连接器	描述和 ARN
ML 图像分类 Aarch64 JTX2	适用于 NVIDIA Jetson TX2 的映像分类推理服务。支持 GPU 加速。 ARN：arn:aws:greengrass : *region*::/connectors/Imag eClassificationAarch64JTX2/ versions/1
ML 图像分类 x86_64	适用于 x86_64 平台的映像分类推理服务。 ARN：arn:aws:greengrass : *region*::/connectors/Imag eClassificationx86-64/versi ons/1
ML 图像分类 Armv7	适用于 ARMv7 平台的映像分类推理服务。 ARN：arn:aws:greengrass : *region*::/connectors/Imag eClassificationARMv7/versio ns/1

有关版本更改的信息，请参阅更改日志。

要求

这些连接器具有以下要求：

Version 2

- AWS IoT Greengrass Core 软件 v1.9.3 版或更高版本。
- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

  > **Note**
  >
  > 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- 安装在核心设备上的 Apache MXNet 框架的依赖项。有关更多信息，请参阅the section called “安装 MXNet 依赖项”。
- Greengrass 组中引用 SageMaker 模型源的机器学习资源。该模型必须由 SageMaker 镜像分类算法训练。有关更多信息，请参阅《Amazon SageMaker 开发人员指南》中的图像分类算法。
- 添加到 Greengrass 组且已配置的 ML 反馈连接器。仅当您要使用此连接器上传模型输入数据并将预测发布到 MQTT 主题时，这才是必需的。
- Greengrass 组角色，配置为允许对目标训练作业执行 `sagemaker:DescribeTrainingJob` 操作，如以下示例 IAM 策略中所示。

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "sagemaker:DescribeTrainingJob"
              ],
              "Resource": "arn:aws:sagemaker:region:account-id:training-
  job:training-job-name"
          }
      ]
  }
  ```

  对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

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）。如果将来要更改目标训练任务，请务必更新组角色。

- 与此连接器进行交互时需要 AWS IoT Greengrass 机器学习开发工具包 v1.1.0。

Version 1

- AWS IoT Greengrass Core 软件 v1.7 版或更高版本。
- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- 安装在核心设备上的 Apache MXNet 框架的依赖项。有关更多信息，请参阅the section called “安装 MXNet 依赖项”。
- Greengrass 组中引用 SageMaker 模型源的机器学习资源。该模型必须由 SageMaker 镜像分类算法训练。有关更多信息，请参阅《Amazon SageMaker 开发人员指南》中的图像分类算法。
- Greengrass 组角色，配置为允许对目标训练作业执行 `sagemaker:DescribeTrainingJob` 操作，如以下示例 IAM 策略中所示。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Resource": "arn:aws:sagemaker:region:account-id:training-
job:training-job-name"
        }
    ]
}
```

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

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）。如果将来要更改目标训练任务，请务必更新组角色。

- 与此连接器进行交互时需要 AWS IoT Greengrass 机器学习开发工具包 v1.0.0 或更高版本。

连接器参数

这些连接器提供以下参数。

Version 2

`MLModelDestinationPath`

Lambda 环境中 ML 资源的本地绝对路径。这是为 ML 资源指定的目标路径。

> **Note**
> 如果您在控制台中创建了 ML 资源，这便是本地路径。

AWS IoT 控制台中的显示名称：模型目标路径

必需：`true`

类型：`string`

有效模式：`.+`

`MLModelResourceId`

引用源模型的 ML 资源的 ID。

AWS IoT 控制台中的显示名称：SageMaker 作业 ARN 资源

必需：`true`

类型：`string`

有效模式：`[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

表示 SageMaker 模型源的 SageMaker 训练作业的 ARN。该模型必须由 SageMaker 镜像分类算法训练。

AWS IoT 控制台中的显示名称：SageMaker 作业 ARN

必需：`true`

类型：string

有效模式：^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$

LocalInferenceServiceName

本地推理服务的名称。用户定义的 Lambda 函数将该服务的名称传递给 AWS IoT Greengrass 机器学习开发工具包的 invoke_inference_service 函数，从而调用该服务。有关示例，请参阅 the section called “用法示例”。

AWS IoT 控制台中的显示名称：本地推理服务名称

必需：true

类型：string

有效模式：[a-zA-Z0-9][a-zA-Z0-9-]{1,62}

LocalInferenceServiceTimeoutSeconds

在推理请求终止之前经过的时间（以秒为单位）。最小值为 1。

AWS IoT 控制台中的显示名称：超时（秒）

必需：true

类型：string

有效模式：[1-9][0-9]*

LocalInferenceServiceMemoryLimitKB

该服务有权访问的内存量（以 KB 为单位）。最小值为 1。

AWS IoT 控制台中的显示名称：内存限制 (KB)

必需：true

类型：string

有效模式：[1-9][0-9]*

GPUAcceleration

CPU 或 GPU（加速）计算上下文。此属性仅适用于 ML 图像分类 Aarch64 JTX2 连接器。

AWS IoT 控制台中的显示名称：GPU 加速

必需：`true`

类型：`string`

有效值：`CPU` 或 `GPU`

`MLFeedbackConnectorConfigId`

用于上传模型输入数据的反馈配置的 ID。这必须与为 ML 反馈连接器定义的反馈配置的 ID 匹配。

仅当您要使用 ML 反馈连接器上传模型输入数据并将预测发布到 MQTT 主题时，才需要此参数。

AWS IoT 控制台中的显示名称：ML 反馈连接器配置 ID

必需：`false`

类型：`string`

有效模式：`^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Version 1

`MLModelDestinationPath`

Lambda 环境中 ML 资源的本地绝对路径。这是为 ML 资源指定的目标路径。

> **Note**
> 如果您在控制台中创建了 ML 资源，这便是本地路径。

AWS IoT 控制台中的显示名称：模型目标路径

必需：`true`

类型：`string`

有效模式：`.+`

`MLModelResourceId`

引用源模型的 ML 资源的 ID。

AWS IoT 控制台中的显示名称：SageMaker 作业 ARN 资源

必需：`true`

类型：`string`

有效模式：`[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

表示 SageMaker 模型源的 SageMaker 训练作业的 ARN。该模型必须由 SageMaker 镜像分类算法训练。

AWS IoT 控制台中的显示名称：SageMaker 作业 ARN

必需：`true`

类型：`string`

有效模式：`^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

本地推理服务的名称。用户定义的 Lambda 函数将该服务的名称传递给 AWS IoT Greengrass 机器学习开发工具包的 `invoke_inference_service` 函数，从而调用该服务。有关示例，请参阅 the section called “用法示例”。

AWS IoT 控制台中的显示名称：本地推理服务名称

必需：`true`

类型：`string`

有效模式：`[a-zA-Z0-9][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

在推理请求终止之前经过的时间（以秒为单位）。最小值为 1。

AWS IoT 控制台中的显示名称：超时（秒）

必需：`true`

类型：`string`

有效模式：`[1-9][0-9]*`

`LocalInferenceServiceMemoryLimitKB`

该服务有权访问的内存量（以 KB 为单位）。最小值为 1。

AWS IoT 控制台中的显示名称：内存限制 (KB)

必需：`true`

类型：`string`

有效模式：`[1-9][0-9]*`

`GPUAcceleration`

CPU 或 GPU（加速）计算上下文。此属性仅适用于 ML 图像分类 Aarch64 JTX2 连接器。

AWS IoT 控制台中的显示名称：GPU 加速

必需：`true`

类型：`string`

有效值：`CPU` 或 `GPU`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，它具有包含 ML 图像分类连接器的初始版本。

示例：CPU 实例

以下示例创建 ML 图像分类 ARMv7l 连接器的实例。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
```

```
            "Id": "MyImageClassificationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ImageClassificationARMv7/versions/2",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
                "LocalInferenceServiceName": "imageClassification",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "MLFeedbackConnectorConfigId": "MyConfig0"
            }
        }
    ]
}'
```

示例：GPU 实例

此示例创建 ML 图像分类 Aarch64 JTX2 连接器的实例，该实例在 NVIDIA Jetson TX2 面板上支持 GPU 加速。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyImageClassificationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ImageClassificationAarch64JTX2/versions/2",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
                "LocalInferenceServiceName": "imageClassification",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "GPUAcceleration": "GPU",
                "MLFeedbackConnectorConfigId": "MyConfig0"
            }
        }
    ]
}'
```

> **Note**
>
> 这些连接器中的 Lambda 函数的生命周期很长。

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅the section called “连接器入门（控制台）”。

输入数据

这些连接器接受一个图像文件作为输入。输入图像文件必须为 `jpeg` 或 `png` 格式。有关更多信息，请参阅the section called “用法示例”。

这些连接器不接受 MQTT 消息作为输入数据。

输出数据

这些连接器返回输入图像中识别的对象的格式化预测：

```
[0.3,0.1,0.04,...]
```

预测包含值列表，这些值与模型训练期间训练数据集中使用的类别相对应。每个值代表图像落入相应类别的概率。概率最高的类别是主导预测。

这些连接器不发布 MQTT 消息来作为输出数据。

用法示例

以下示例 Lambda 函数使用 AWS IoT Greengrass 机器学习软件开发工具包 与 ML 图像分类连接器进行交互。

> **Note**
>
> 您可以从 AWS IoT Greengrass 机器学习开发工具包下载页面下载软件开发工具包。

该示例初始化一个开发工具包客户端，并同步调用该开发工具包的 `invoke_inference_service` 函数来调用本地推理服务。它会传入算法类型、服务名称、映像类型和映像内容。然后，该示例会解析服务响应以获取概率结果（预测）。

Python 3.7

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='image-classification',
            ServiceName='imageClassification',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}("{}")'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}("{}")'.format(e.__class__.__name__,
 e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    count = len(predictions.split(','))
    predictions_arr = np.fromstring(predictions, count=count, sep=',')
```

```
    # Perform business logic that relies on the predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()
    return

infer()

def function_handler(event, context):
    return
```

Python 2.7

```
import logging
from threading import Timer

import numpy

import greengrass_machine_learning_sdk as gg_ml

# The inference input image.
with open("/test_img/test.jpg", "rb") as f:
    content = f.read()

client = gg_ml.client("inference")


def infer():
    logging.info("Invoking Greengrass ML Inference service")

    try:
        resp = client.invoke_inference_service(
            AlgoType="image-classification",
            ServiceName="imageClassification",
            ContentType="image/jpeg",
            Body=content,
        )
    except gg_ml.GreengrassInferenceException as e:
        logging.info('Inference exception %s("%s")', e.__class__.__name__, e)
        return
    except gg_ml.GreengrassDependencyException as e:
        logging.info('Dependency exception %s("%s")', e.__class__.__name__, e)
```

```
        return

    logging.info("Response: %s", resp)
    predictions = resp["Body"].read()
    logging.info("Predictions: %s", predictions)

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    predictions_arr = numpy.fromstring(predictions, sep=",")
    logging.info("Split into %s predictions.", len(predictions_arr))

    # Perform business logic that relies on predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()


infer()


# In this example, the required AWS Lambda handler is never called.
def function_handler(event, context):
    return
```

AWS IoT Greengrass 机器学习开发工具包中的 `invoke_inference_service` 函数接受以下参数：

参数	描述
`AlgoType`	要用于推理的算法类型的名称。目前仅支持 `image-classification` 。 必需：`true` 类型：`string` 有效值：`image-classification`

参数	描述
ServiceName	本地推理服务的名称。在配置了连接器时，使用为 LocalInferenceServiceName 参数指定的名称。 必需：true 类型：string
ContentType	输入映像的 mime 类型。 必需：true 类型：string 有效值：image/jpeg, image/png
Body	输入映像文件的内容。 必需：true 类型：binary

在 AWS IoT Greengrass 核心上安装 MXNet 依赖项

要使用 ML 图像分类连接器，您必须在核心设备上安装 Apache MXNet 框架的依赖项。连接器使用该框架来处理 ML 模型。

> **Note**
>
> 这些连接器捆绑到预编译的 MXNet 库，因此无需在核心设备上安装 MXNet 框架。

AWS IoT Greengrass 提供了脚本来安装以下常见平台和设备的依赖项（或用作安装参考）。如果使用的是其他平台或设备，请参阅 MXNet 文档以了解您的配置。

安装 MXNet 依赖项之前，请确保设备上存在所需的系统库（具有指定的最低版本）。

NVIDIA Jetson TX2

1. 安装 CUDA Toolkit 9.0 和 cuDNN 7.0。您可以按照入门教程中the section called “设置其他设备”中的说明进行操作。
2. 启用通用存储库，以便连接器可以安装社区维护的开放软件。有关更多信息，请参阅 Ubuntu 文档中的 Repositories/Ubuntu。

   a. 打开 `/etc/apt/sources.list` 文件。

   b. 确保以下各行已取消注释。

   ```
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   ```

3. 将以下安装脚本的副本保存到核心设备上一个名为 `nvidiajtx2.sh` 的文件。

   Python 3.7

   ```
   #!/bin/bash
   set -e

   echo "Installing dependencies on the system..."
   echo 'Assuming that universe repos are enabled and checking dependencies...'
   apt-get -y update
   apt-get -y dist-upgrade
   apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-
   dev
   apt-get install -y python3.7 python3.7-dev

   python3.7 -m pip install --upgrade pip
   python3.7 -m pip install numpy==1.15.0
   python3.7 -m pip install opencv-python || echo 'Error: Unable to install
    OpenCV with pip on this platform. Try building the latest OpenCV from source
    (https://github.com/opencv/opencv).'

   echo 'Dependency installation/upgrade complete.'
   ```

> **Note**
>
> 如果 OpenCV 使用此脚本无法成功安装，您可以尝试从源代码进行构建。有关更多信息，请参阅 OpenCV 文档中的在 Linux 中安装，或参考您平台的其他在线资源。

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
 python-dev

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install numpy==1.15.0 scipy

echo 'Dependency installation/upgrade complete.'
```

4. 从保存文件的目录中，运行以下命令：

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. 将以下安装脚本的副本保存到核心设备上一个名为 `x86_64.sh` 的文件。

   Python 3.7

   ```
   #!/bin/bash
   set -e
   ```

```
echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
  apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
  echo "OS Release not supported: $release"
  exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
 OpenCV with pip on this platform. Try building the latest OpenCV from source
 (https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 如果 OpenCV 使用此脚本无法成功安装，您可以尝试从源代码进行构建。有关更多信息，请参阅 OpenCV 文档中的在 Linux 中安装，或参考您平台的其他在线资源。

Python 2.7

```
#!/bin/bash
set -e
```

```
echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1 python-dev
 python-pip
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext python-
pip
else
  echo "OS Release not supported: $release"
  exit 1
fi

pip install numpy==1.15.0 scipy opencv-python

echo 'Dependency installation/upgrade complete.'
```

2. 从保存文件的目录中，运行以下命令：

```
sudo x86_64.sh
```

Armv7 (Raspberry Pi)

1. 将以下安装脚本的副本保存到核心设备上一个名为 `armv7l.sh` 的文件。

   Python 3.7

```
#!/bin/bash
set -e
```

```
echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
 OpenCV with pip on this platform. Try building the latest OpenCV from source
 (https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 如果 OpenCV 使用此脚本无法成功安装，您可以尝试从源代码进行构建。有关更多信息，请参阅 OpenCV 文档中的在 Linux 中安装，或参考您平台的其他在线资源。

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev python-dev

# python-opencv depends on python-numpy. The latest version in the APT
 repository is python-numpy-1.8.2
# This script installs python-numpy first so that python-opencv can be
 installed, and then install the latest
# numpy-1.15.x with pip
apt-get install -y python-numpy python-opencv
dpkg --remove --force-depends python-numpy
```

```
echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install --upgrade numpy==1.15.0 picamera scipy

echo 'Dependency installation/upgrade complete.'
```

2. 从保存文件的目录中，运行以下命令：

```
sudo bash armv7l.sh
```

> **Note**
>
> 在 Raspberry Pi 上，使用 pip 安装机器学习依赖项是一项内存密集型操作，可能会导致设备用尽内存，变得无法响应。解决办法是临时增加交换空间大小：
> 在 /etc/dphys-swapfile 中，增加 CONF_SWAPSIZE 变量的值，然后运行以下命令重启 dphys-swapfile。
>
> ```
> /etc/init.d/dphys-swapfile restart
> ```

日志记录和故障排除

根据您的组设置，事件和错误日志会写入到 CloudWatch 日志和/或本地文件系统。此连接器中的日志使用前缀 LocalInferenceServiceName。如果连接器出现异常行为，请检查连接器日志。其中经常包含有用的调试信息，例如缺失 ML 库依赖项或连接器启动故障的原因。

如果将 AWS IoT Greengrass 组配置为写入本地日志，则连接器会将日志文件写入到 *greengrass-root*/ggc/var/log/user/*region*/aws/。有关 Greengrass 日志记录的更多信息，请参阅 the section called “利用 AWS IoT Greengrass 日志进行监控”。

可以使用以下信息帮助解决 ML 图像分类连接器问题。

所需系统库

以下选项卡列出了每个 ML 图像分类连接器所需的系统库。

ML Image Classification Aarch64 JTX2

Library	最低版本
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	不适用
libcudart.so.9.0	不适用
libcudnn.so.7	不适用
libcufft.so.9.0	不适用
libcurand.so.9.0	不适用
libcusolver.so.9.0	不适用
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0，OMP_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21，CXXABI_1.3.8

ML Image Classification x86_64

Library	最低版本
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4

Library	最低版本
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8，GLIBCXX_3.4.21

ML Image Classification Armv7

Library	最低版本
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8，CXXABI_ARM_1.3.3，GLIBCXX_3.4.20

问题

症状	解决方案
在 Raspberry Pi 上，记录了以下错误消息，并且您没有使用摄像机：`Failed to initialize libdc1394`	运行以下命令以显示驱动程序： `sudo ln /dev/null /dev/raw1394` 此操作是临时的，符号链接将在重启后消失。请参阅您的 OS 分发手册以了解如何在重启时自动创建链接。

许可证

ML 图像分类连接器包含以下第三方软件/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

- Deep Neural Network Library (DNNL)/Apache 许可证 2.0
- OpenMP* 运行时库/请参阅 Intel OpenMP 运行时库许可。
- mxnet/Apache 许可证 2.0
- six/MIT

Intel OpenMP 运行时库许可。Intel® OpenMP* 运行时经过双重许可，具有商业 (COM) 许可证（作为 Intel® Parallel Studio XE Suite 产品的一部分）和 BSD 开源 (OSS) 许可证。

此连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
2	添加了 `MLFeedbackConnectorConfigId` 参数，以支持使用 ML 反馈连接器上传模型输入数据，将预测发布到 MQTT 主题以及将指标发布到 Amazon CloudWatch。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”
- 执行机器学习推理
- 《Amazon SageMaker 开发人员指南》中的图像分类算法

ML 对象检测连接器

> **Warning**
>
> 此连接器已进入生命周期延长阶段，AWS IoT Greengrass 不会发布更新来提供功能、现有功能增强、安全补丁或错误修复。有关更多信息，请参阅AWS IoT Greengrass Version 1 维护策略。

ML 对象检测连接器提供在 AWS IoT Greengrass 核心上运行的机器学习 (ML) 推理服务。该本地推理服务使用由 SageMaker Neo 深度学习编译器编译的对象检测模型执行对象检测。支持两种类型的对象检测模型：Single Shot Multibox Detector（单步多框检测器，SSD）和 You Only Look Once（只用看一遍，YOLO）v3。有关更多信息，请参阅对象检测模型要求。

用户定义的 Lambda 函数使用 AWS IoT Greengrass 机器学习开发工具包将推理请求提交给本地推理服务。该服务对输入图像执行本地推理，并针对图像中检测到的每个对象返回预测列表。每个预测都包含一个对象类别、一个预测置信度得分和像素坐标，这些像素坐标指定围绕预测对象的边界框。

AWS IoT Greengrass 为多个平台提供 ML 对象检测连接器：

连接器	描述和 ARN
ML 对象检测 Aarch64 JTX2	适用于 NVIDIA Jetson TX2 的对象检测推导服务。支持 GPU 加速。 ARN：`arn:aws:greengrass:region::/connectors/ObjectDetectionAarch64JTX2/versions/1`
ML 对象检测 x86664	适用于 x86_64 平台的对象检测推理服务。 ARN：`arn:aws:greengrass:region::/connectors/ObjectDetectionx86-64/versions/1`
ML 对象检测 ARMv7	适用于 ARMv7 平台的对象检测推理服务。 ARN：`arn:aws:greengrass:region::/connectors/ObjectDetectionARMv7/versions/1`

要求

这些连接器具有以下要求：

- AWS IoT Greengrass Core 软件 v1.9.3 版或更高版本。
- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

> **Note**
>
> 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- 核心设备上已安装 SageMaker Neo 深度学习运行时的依赖项。有关更多信息，请参阅the section called “安装 Neo 深度学习运行时依赖项”。
- Greengrass 组中的 ML 资源。ML 资源必须引用包含对象检测模型的 Amazon S3 存储桶。有关更多信息，请参阅 Amazon S3 模型来源。

> **Note**
>
> 该模型必须是 Single Shot Multibox Detector（单步多框检测器，SSD）或 You Only Look Once（只用看一遍，YOLO）v3 对象检测模型类型。它必须使用 SageMaker Neo 深度学习编译器进行编译。有关更多信息，请参阅对象检测模型要求。

- 添加到 Greengrass 组且已配置的 ML 反馈连接器。仅当您要使用此连接器上传模型输入数据并将预测发布到 MQTT 主题时，这才是必需的。
- 与此连接器进行交互时需要 AWS IoT Greengrass 机器学习开发工具包 v1.1.0。

对象检测模型要求

ML 对象检测连接器支持单步多框检测器（SSD）和只用看一遍（YOLO）v3 对象检测模型类型。您可以使用 GluonCV 提供的对象检测组件来通过您自己的数据集训练模型。或者，您可以使用 GluonCV Model Zoo 中的预训练模型：

- 预训练 SSD 模型
- 预训练 YOLO v3 模型

您的对象检测模型必须使用 512 x 512 输入图像进行训练。来自 GluonCV Model Zoo 的预训练模型已经满足了这一要求。

必须使用 SageMaker Neo 深度学习编译器编译训练的对象检测模型。编译时，请确保目标硬件与您的 Greengrass 核心设备的硬件匹配。有关更多信息，请参阅 Amazon SageMaker 开发人员指南中的 SageMaker Neo。

必须将已编译的模型作为 ML 资源（Amazon S3 模型源）添加到与连接器相同的 Greengrass 组中。

连接器参数

这些连接器提供以下参数。

`MLModelDestinationPath`

包含 Neo 兼容 ML 模型的 Amazon S3 存储桶的绝对路径。这是为 ML 模型资源指定的目标路径。

AWS IoT 控制台中的显示名称：模型目标路径

必需：`true`

类型：`string`

有效模式：`.+`

`MLModelResourceId`

引用源模型的 ML 资源的 ID。

AWS IoT 控制台中的显示名称：Greengrass 组 ML 资源

必需：`true`

类型：`S3MachineLearningModelResource`

有效模式：`^[a-zA-Z0-9:_-]+$`

`LocalInferenceServiceName`

本地推理服务的名称。用户定义的 Lambda 函数将该服务的名称传递给 AWS IoT Greengrass 机器学习开发工具包的 `invoke_inference_service` 函数，从而调用该服务。有关示例，请参阅the section called “用法示例”。

AWS IoT 控制台中的显示名称：本地推理服务名称

必需：`true`

类型：`string`

有效模式：`^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

`LocalInferenceServiceTimeoutSeconds`

在推理请求终止之前经过的时间（以秒为单位）。最小值为 1。默认值为 10。

AWS IoT 控制台中的显示名称：超时（秒）

必需：`true`

类型：`string`

有效模式：`^[1-9][0-9]*$`

`LocalInferenceServiceMemoryLimitKB`

该服务有权访问的内存量（以 KB 为单位）。最小值为 1。

AWS IoT 控制台中的显示名称：内存限制

必需：`true`

类型：`string`

有效模式：`^[1-9][0-9]*$`

`GPUAcceleration`

CPU 或 GPU（加速）计算上下文。此属性仅适用于 ML 图像分类 Aarch64 JTX2 连接器。

AWS IoT 控制台中的显示名称：GPU 加速

必需：`true`

类型：`string`

有效值：`CPU` 或 `GPU`

`MLFeedbackConnectorConfigId`

用于上传模型输入数据的反馈配置的 ID。这必须与为 ML 反馈连接器定义的反馈配置的 ID 匹配。

仅当您要使用 ML 反馈连接器上传模型输入数据并将预测发布到 MQTT 主题时，才需要此参数。

AWS IoT 控制台中的显示名称：ML 反馈连接器配置 ID

必需：`false`

类型：`string`

有效模式：`^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 ML 对象检测连接器。以下示例创建 ML 对象检测 ARMv7l 连接器的实例。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyObjectDetectionConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ObjectDetectionARMv7/versions/1",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "LocalInferenceServiceName": "objectDetection",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "MLFeedbackConnectorConfigId" : "object-detector-random-sampling"
            }
        }
    ]
}'
```

> **Note**
>
> 这些连接器中的 Lambda 函数的生命周期很长。

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅the section called “连接器入门（控制台）”。

输入数据

这些连接器接受一个图像文件作为输入。输入图像文件必须为 `jpeg` 或 `png` 格式。有关更多信息，请参阅the section called “用法示例”。

这些连接器不接受 MQTT 消息作为输入数据。

输出数据

这些连接器返回输入图像中识别的对象的预测结果的格式化列表：

```
{
    "prediction": [
        [
            14,
            0.9384938478469849,
            0.37763649225234985,
            0.5110225081443787,
            0.6697432398796082,
            0.8544386029243469
        ],
        [
            14,
            0.8859519958496094,
            0,
            0.43536216020584106,
            0.3314110040664673,
            0.9538808465003967
        ],
        [
            12,
            0.04128098487854004,
            0.5976729989051819,
            0.5747185945510864,
            0.704264223575592,
            0.857937216758728
        ],
        ...
    ]
}
```

列表中的每个预测都包含在方括号中，并包含六个值：

- 第一个值表示已识别对象的预测对象类别。在 Neo 深度学习编译器中训练您的对象检测机器学习模型时，将确定对象类别及其对应的值。
- 第二个值是对象类别预测的置信度得分。这代表预测正确的可能性。
- 最后四个值对应于像素尺寸，该像素尺寸表示图像中预测对象周围的边界框。

这些连接器不发布 MQTT 消息来作为输出数据。

用法示例

以下示例 Lambda 函数使用 AWS IoT Greengrass 机器学习软件开发工具包与 ML 对象检测连接器进行交互。

> **Note**
>
> 您可以从 AWS IoT Greengrass 机器学习开发工具包下载页面下载软件开发工具包。

该示例初始化一个开发工具包客户端，并同步调用该开发工具包的 `invoke_inference_service` 函数来调用本地推理服务。它会传入算法类型、服务名称、映像类型和映像内容。然后，该示例会解析服务响应以获取概率结果（预测）。

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='object-detection',
            ServiceName='objectDetection',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}("{}")'.format(e.__class__.__name__, e))
```

```
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}("{}")'.format(e.__class__.__name__, e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))
    predictions = eval(predictions)

    # Perform business logic that relies on the predictions.

    # Schedule the infer() function to run again in ten second.
    Timer(10, infer).start()
    return

infer()

def function_handler(event, context):
    return
```

AWS IoT Greengrass 机器学习开发工具包中的 `invoke_inference_service` 函数接受以下参数：

参数	描述
`AlgoType`	要用于推理的算法类型的名称。目前仅支持 `object-detection` 。 必需：`true` 类型：`string` 有效值：`object-detection`
`ServiceName`	本地推理服务的名称。在配置了连接器时，使用为 `LocalInferenceServiceName` 参数指定的名称。 必需：`true` 类型：`string`

参数	描述
ContentType	输入映像的 mime 类型。 必需：true 类型：string 有效值：image/jpeg, image/png
Body	输入映像文件的内容。 必需：true 类型：binary

在 AWS IoT Greengrass 核心上安装 Neo 深度学习运行时依赖项

ML 对象检测连接器与 SageMaker Neo 深度学习运行时 (DLR) 捆绑在一起。连接器使用此运行时来处理 ML 模型。要使用这些连接器，必须在核心设备上安装 DLR 的依赖项。

在您安装 DLR 依赖项之前，请确保设备上存在所需的系统库（具有指定的最低版本）。

NVIDIA Jetson TX2

1. 安装 CUDA Toolkit 9.0 和 cuDNN 7.0。您可以按照入门教程中the section called “设置其他设备”中的说明进行操作。
2. 启用通用存储库，以便连接器可以安装社区维护的开放软件。有关更多信息，请参阅 Ubuntu 文档中的 Repositories/Ubuntu。

   a. 打开 /etc/apt/sources.list 文件。

   b. 确保以下各行已取消注释。

   ```
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   ```

3. 将以下安装脚本的副本保存到核心设备上一个名为 nvidiajtx2.sh 的文件。

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 如果 OpenCV 使用此脚本无法成功安装，您可以尝试从源代码进行构建。有关更多信息，请参阅 OpenCV 文档中的在 Linux 中安装，或参考您平台的其他在线资源。

4. 从保存文件的目录中，运行以下命令：

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. 将以下安装脚本的副本保存到核心设备上一个名为 x86_64.sh 的文件。

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
```

```
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
  apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
  echo "OS Release not supported: $release"
  exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 如果 OpenCV 使用此脚本无法成功安装，您可以尝试从源代码进行构建。有关更多信息，请参阅 OpenCV 文档中的在 Linux 中安装，或参考您平台的其他在线资源。

2. 从保存文件的目录中，运行以下命令：

```
sudo x86_64.sh
```

ARMv7 (Raspberry Pi)

1. 将以下安装脚本的副本保存到核心设备上一个名为 `armv7l.sh` 的文件。

```
#!/bin/bash
```

```
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> 如果 OpenCV 使用此脚本无法成功安装，您可以尝试从源代码进行构建。有关更多信息，请参阅 OpenCV 文档中的在 Linux 中安装，或参考您平台的其他在线资源。

2. 从保存文件的目录中，运行以下命令：

```
sudo bash armv7l.sh
```

> **Note**
>
> 在 Raspberry Pi 上，使用 `pip` 安装机器学习依赖项是一项内存密集型操作，可能会导致设备用尽内存，变得无法响应。解决办法是临时增加交换空间大小。在 `/etc/dphys-swapfile` 中，增加 `CONF_SWAPSIZE` 变量的值，然后运行以下命令重启 `dphys-swapfile`。
>
> ```
> /etc/init.d/dphys-swapfile restart
> ```

日志记录和故障排除

根据您的组设置，事件和错误日志会写入到 CloudWatch 日志和/或本地文件系统。此连接器中的日志使用前缀 `LocalInferenceServiceName`。如果连接器出现异常行为，请检查连接器日志。其中经常包含有用的调试信息，例如缺失 ML 库依赖项或连接器启动故障的原因。

如果将 AWS IoT Greengrass 组配置为写入本地日志，则连接器会将日志文件写入到 *greengrass-root*`/ggc/var/log/user/`*region*`/aws/`。有关 Greengrass 日志记录的更多信息，请参阅 the section called “利用 AWS IoT Greengrass 日志进行监控”。

可以使用以下信息帮助解决 ML 对象检测连接器问题。

所需系统库

以下选项卡列出了每个 ML 对象检测连接器所需的系统库。

ML Object Detection Aarch64 JTX2

Library	最低版本
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	不适用
libcudart.so.9.0	不适用
libcudnn.so.7	不适用
libcufft.so.9.0	不适用
libcurand.so.9.0	不适用
libcusolver.so.9.0	不适用
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0，OMP_1.0
libm.so.6	GLIBC_2.23

Library	最低版本
libnvinfer.so.4	不适用
libnvrm_gpu.so	不适用
libnvrm.so	不适用
libnvidia-fatbinaryloader.so.28.2.1	不适用
libnvos.so	不适用
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21，CXXABI_1.3.8

ML Object Detection x86_64

Library	最低版本
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8，GLIBCXX_3.4.21

ML Object Detection ARMv7

Library	最低版本
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8，CXXABI_ARM_1.3.3，GLIBCXX_3.4.20

问题

症状	解决方案
在 Raspberry Pi 上，记录了以下错误消息，并且您没有使用摄像机：`Failed to initialize libdc1394`	运行以下命令以显示驱动程序： `sudo ln /dev/null /dev/raw1394` 此操作是临时的。重新启动后，符号链接消失。请参阅您的操作系统分发手册以了解如何在重启时自动创建链接。

许可证

ML 对象检测连接器包含以下第三方软件/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

- 深度学习运行时/Apache 许可证 2.0
- six/MIT

此连接器在 Greengrass Core 软件许可协议下发布。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”
- 执行机器学习推理
- 《Amazon SageMaker 开发人员指南》中的对象检测算法

Modbus-RTU 协议适配器连接器

Modbus-RTU 协议适配器连接器从 AWS IoT Greengrass 组中的 Modbus RTU 设备轮询信息。

此连接器从用户定义的 Lambda 函数接收 Modbus RTU 请求的参数。它发送相应的请求，然后从目标设备发布响应作为 MQTT 消息。

此连接器具有以下版本。

版本	ARN
3	`arn:aws:greengrass:region::/connectors/ModbusRTUProtocolAdapter/versions/3`

版本	ARN
2	`arn:aws:greengrass:  region::/connectors/ModbusRTUProtocolAdapter/versions/2`
1	`arn:aws:greengrass:  region::/connectors/ModbusRTUProtocolAdapter/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 3

- AWS IoT Greengrass Core 软件 v1.9.3 版或更高版本。
- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

  > **Note**
  >
  > 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- AWS IoT Greengrass 核心与 Modbus 设备之间的物理连接。核心必须通过串行端口（例如 USB 端口）以物理方式连接到 Modbus RTU 网络。
- Greengrass 组中一个指向物理 Modbus 串行端口的本地设备资源。
- 用户定义的 Lambda 函数，可向此连接器发送 Modbus RTU 请求参数。请求参数必须符合预期模式，并包含 Modbus RTU 网络上的目标设备的 ID 和地址。有关更多信息，请参阅the section called “输入数据”。

Versions 1 - 2

- AWS IoT Greengrass Core 软件 v1.7 版或更高版本。
- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- AWS IoT Greengrass 核心与 Modbus 设备之间的物理连接。核心必须通过串行端口（例如 USB 端口）以物理方式连接到 Modbus RTU 网络。
- Greengrass 组中一个指向物理 Modbus 串行端口的本地设备资源。
- 用户定义的 Lambda 函数，可向此连接器发送 Modbus RTU 请求参数。请求参数必须符合预期模式，并包含 Modbus RTU 网络上的目标设备的 ID 和地址。有关更多信息，请参阅the section called “输入数据”。

连接器参数

此连接器支持以下参数：

`ModbusSerialPort-ResourceId`

表示物理 Modbus 串行端口的本地设备资源的 ID。

> **Note**
> 连接器被授予对该资源的读写访问权限。

AWS IoT 控制台中的显示名称：Modbus 串行端口资源

必需：`true`

类型：`string`

有效模式：`.+`

`ModbusSerialPort`

设备上的物理 Modbus 串行端口的绝对路径。这是为 Modbus 本地设备资源指定的源路径。

AWS IoT 控制台中的显示名称：Modbus 串口资源的源路径

必需：`true`

类型：`string`

有效模式：`.+`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 Modbus-RTU 协议适配器连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyModbusRTUProtocolAdapterConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ModbusRTUProtocolAdapter/versions/3",
            "Parameters": {
                "ModbusSerialPort-ResourceId": "MyLocalModbusSerialPort",
                "ModbusSerialPort": "/path-to-port"
            }
        }
    ]
}'
```

> **Note**
>
> 此连接器中的 Lambda 函数的生命周期很长。

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅the section called “连接器入门（控制台）”。

> **Note**
>
> 部署 Modbus-RTU 协议适配器连接器后，您可以使用 AWS IoT Things Graph 协调组中设备之间的交互。有关更多信息，请参阅 AWS IoT Things Graph 用户指南中的 Modbus。

输入数据

此连接器从用户定义的 Lambda 函数接受关于 MQTT 主题的 Modbus RTU 请求参数。输入消息必须采用 JSON 格式。

订阅中的主题筛选条件

```
modbus/adapter/request
```

消息属性

请求消息因其所表示的 Modbus RTU 请求类型而异。所有请求都需要具有以下属性：

- 在 `request` 对象中：
  - `operation`. 要执行的操作的名称。例如，指定 `"operation": "ReadCoilsRequest"` 可读取线圈。该值必须是 Unicode 字符串。有关支持的操作，请参阅 the section called “Modbus RTU 请求和响应”。
  - `device`. 请求的目标设备。该值必须介于 `0 - 247` 之间。
- `id` 属性。请求的 ID。该值用于重复数据删除，并在所有响应（包括错误响应）的 `id` 属性中按原样返回。该值必须是 Unicode 字符串。

> **Note**
>
> 如果请求包含地址字段，您必须将值指定为整数。例如，`"address": 1`。

要包含在请求中的其他参数取决于操作。需要除 CRC 之外的所有请求参数，且分别进行处理。有关示例，请参阅 the section called “示例请求和响应”。

示例输入：读取线圈请求

```
{
    "request": {
        "operation": "ReadCoilsRequest",
     "device": 1,
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

输出数据

此连接器发布对传入 Modbus RTU 请求的响应。

订阅中的主题筛选条件

```
modbus/adapter/response
```

消息属性

响应消息的格式因相应的请求和响应状态而异。有关示例，请参阅 the section called “示例请求和响应”。

> **Note**
>
> 写入操作的响应只不过是对请求的回显。尽管对于写入响应不会返回有意义的信息，但最好还是检查一下响应的状态。

每个响应均包括以下属性：

- 在 `response` 对象中：
  - `status`. 请求的状态。状态可以是以下值之一：
    - `Success`. 请求有效，已发送至 Modbus RTU 网络并返回响应。
    - `Exception`. 请求有效，已发送至 Modbus RTU 网络并返回异常响应。有关更多信息，请参阅the section called “响应状态：异常”。
    - `No Response`. 请求无效，并且连接器在通过 Modbus RTU 网络发送请求之前捕获到错误。有关更多信息，请参阅the section called “响应状态：无响应”。
  - `device`. 请求发送到的设备。
  - `operation`. 发送的请求类型。
  - `payload`. 返回的响应内容。如果 `status` 为 `No Response`，则该对象只包含一个 `error` 属性及错误描述（例如 `"error": "[Input/Output] No Response received from the remote unit"`）。
- `id` 属性。用于重复数据删除的请求的 ID。

示例输出：成功

```
{
    "response" : {
        "status" : "success",
        "device": 1,
     "operation": "ReadCoilsRequest",
     "payload": {
```

```
          "function_code": 1,
          "bits": [1]
     }
     },
     "id" : "TestRequest"
}
```

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" : "TestRequest"
}
```

有关更多示例，请参阅 the section called “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

请求中的操作名称	响应中的函数代码
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"
}
```

读取保持寄存器

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,
     "value": 1
    },
    "id": "TestRequest"
}
```

写入单线圈

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"
```

写入单寄存器

请求示例：

```
{
    "request": {
        "operation": "WriteSingleRegisterRequest",
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

写入多线圈

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"
}
```

屏蔽写入寄存器

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"
}
```

读取写入多寄存器

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**
>
> 此响应中返回的寄存器是读取的寄存器。

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 包含错误的原因。
- 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 <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"
}
```

用法示例

使用以下概括步骤设置可用于尝试连接器的示例 Python 3.7 Lambda 函数。

> **Note**
>
> - 如果要使用其他 Python 运行时，您可以创建从 Python3.x 到 Python 3.7 的符号链接。
> - 连接器入门（控制台）和连接器入门 (CLI) 主题包含详细步骤，说明如何配置和部署示例 Twilio 通知连接器。

1. 确保满足连接器的要求。
2. 创建并发布将输入数据发送到连接器的 Lambda 函数。

   将示例代码保存为 PY 文件。下载并解压适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。然后，创建一个 zip 包，其中在根级别包含 PY 文件和 `greengrasssdk` 文件夹。此 zip 包是您上传到 AWS Lambda 的部署包。

创建 Python 3.7 Lambda 函数后，请发布函数版本并创建别名。

3. 配置 Greengrass 组。

   a. 通过别名来添加 Lambda 函数（推荐）。将 Lambda 生命周期配置为长时间生存（或在 CLI 中设置为 `"Pinned": true`）。

   b. 添加所需的本地设备资源并授予对 Lambda 函数的读/写访问权限。

   c. 添加连接器并配置其参数。

   d. 添加允许连接器接收输入数据并针对支持的主题筛选条件发送输出数据的订阅。

      - 将 Lambda 函数设置为源，将连接器设置为目标，并使用支持的输入主题筛选条件。
      - 将连接器设置为源，将 AWS IoT Core 设置为目标，并使用支持的输出主题筛选条件。您可以使用此订阅查看 AWS IoT 控制台中的状态消息。

4. 部署组。

5. 在 AWS IoT 控制台中，在测试页面上，订阅输出数据主题以查看连接器中的状态消息。示例 Lambda 函数是长时间生存的，并且在部署组后立即开始发送消息。

   完成测试后，您可以将 Lambda 生命周期设置为按需（或在 CLI 中设置为 `"Pinned": false`）并部署组。这会阻止函数发送消息。

示例

以下示例 Lambda 函数向连接器发送一条输入消息。

```
import greengrasssdk
import json

TOPIC_REQUEST = 'modbus/adapter/request'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_read_coils_request():
 request = {
  "request": {
   "operation": "ReadCoilsRequest",
   "device": 1,
   "address": 1,
   "count": 1
```

```
  },
  "id": "TestRequest"
 }
 return request

def publish_basic_request():
 iot_client.publish(payload=json.dumps(create_read_coils_request()),
 topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
 return
```

许可证

Modbus-RTU 协议适配器连接器包含以下第三方软件/许可：

- pymodbus/BSD
- pyserial/BSD

该连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
3	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。
2	更新了连接器 ARN 以获得 AWS 区域 支持。 改进了错误日志记录。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”

modbus-TCP 协议适配器连接器

Modbus-TCP 协议适配器连接器通过 modbustCP 协议从本地设备收集数据，并将其发布到选定的 `StreamManager` 数据流。

也可以将此连接器与 IoT SiteWise 连接器和 IoT SiteWise 网关配合使用。您的网关必须提供连接器的配置。有关更多信息，请参阅《IoT SiteWise 用户指南》中的配置 Modbus TCP 源。

> **Note**
>
> 此连接器在无容器隔离模式下运行，因此您可以将其部署到在 Docker 容器中运行的 AWS IoT Greengrass 组。

此连接器具有以下版本。

版本	ARN
3	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/3`
2	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/2`
1	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 1 - 3

- AWS IoT Greengrass Core 软件 v1.10.2 版或更高版本。
- AWS IoT Greengrass 组上启用的流管理器 。
- Java 8 安装在核心设备上并已添加到 PATH 环境变量中。

> **Note**
>
> 该连接器仅在以下区域推出：
>
> - ap-southeast-1
> - ap-southeast-2
> - eu-central-1
> - eu-west-1
> - us-east-1
> - us-west-2
> - cn-north-1

连接器参数

此连接器支持以下参数：

`LocalStoragePath`

IoT SiteWise 连接器可以向其写入持久性数据的 AWS IoT Greengrass 主机上的目录。默认目录为 `/var/sitewise`。

AWS IoT 控制台中的显示名称：本地存储路径

必需：`false`

类型：`string`

有效模式：`^\s*$|\/.`

`MaximumBufferSize`

IoT SiteWise 磁盘使用的最大大小（以 GB 为单位）。默认大小为 10 GB。

AWS IoT 控制台中的显示名称：最大磁盘缓冲区大小

必需：`false`

类型：`string`

有效模式：`^\s*$|[0-9]+`

`CapabilityConfiguration`

一组 Modbus TCP 收集器的配置，连接器从这些收集器中收集数据或连接到这些收集器。

AWS IoT 控制台中的显示名称：CapabilityConfiguration

必需：`false`

类型：一个格式正确的 JSON 字符串，用于定义支持的反馈配置集。

以下是 `CapabilityConfiguration` 的一个示例：

```
{
    "sources": [
        {
            "type": "ModBusTCPSource",
            "name": "SourceName1",
            "measurementDataStreamPrefix": "SourceName1_Prefix",
            "destination": {
                "type": "StreamManager",
                "streamName": "SiteWise_Stream_1",
                "streamBufferSize": 8
            },
            "endpoint": {
                "ipAddress": "127.0.0.1",
                "port": 8081,
                "unitId": 1
            },
            "propertyGroups": [
                {
                    "name": "GroupName",
```

```
                    "tagPathDefinitions": [
                        {
                            "type": "ModBusTCPAddress",
                            "tag": "TT-001",
                            "address": "30001",
                            "size": 2,
                            "srcDataType": "float",
                            "transformation": "byteWordSwap",
                            "dstDataType": "double"
                        }
                    ],
                    "scanMode": {
                        "type": "POLL",
                        "rate": 100
                    }
                }
            ]
        }
    ]
}
```

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 Modbus-TCP 协议适配器连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '
{
    "Connectors": [
        {
            "Id": "MyModbusTCPConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/ModbusTCP/
versions/3",
            "Parameters": {
                "capability_configuration": "{\"version\":1,\"namespace\":
\"iotsitewise:modbuscollector:1\",\"configuration\":\"{\"sources\":[{\"type
\":\"ModBusTCPSource\",\"name\":\"SourceName1\",\"measurementDataStreamPrefix
\":\"\",\"endpoint\":{\"ipAddress\":\"127.0.0.1\",\"port\":8081,\"unitId\":1},
\"propertyGroups\":[{\"name\":\"PropertyGroupName\",\"tagPathDefinitions\":[{\"type
\":\"ModBusTCPAddress\",\"tag\":\"TT-001\",\"address\":\"30001\",\"size\":2,
\"srcDataType\":\"hexdump\",\"transformation\":\"noSwap\",\"dstDataType\":\"string
```

```
\"}],\"scanMode\":{\"rate\":200,\"type\":\"POLL\"}}],\"destination\":{\"type\":
\"StreamManager\",\"streamName\":\"SiteWise_Stream\",\"streamBufferSize\":10},
\"minimumInterRequestDuration\":200}]}\"}"
            }
        }
    ]
}'
```

> **Note**
>
> 此连接器中的 Lambda 函数的生命周期很长。

输入数据

该连接器不接受 MQTT 消息作为输入数据。

输出数据

此连接器将数据发布到 `StreamManager`。您必须配置目标消息流。输出消息具有以下结构：

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

许可证

Modbus-TCP 协议适配器连接器包含以下第三方软件/许可：

- Digital Petri Modbus

该连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改	日期
3（推荐）	此版本包含错误修复。	2021 年 12 月 22 日
2	增加了对 ASCII、UTF8 和 ISO8859 编码的源字符串的支持。	2021 年 5 月 24 日
1	首次发布。	2020 年 12 月 15 日

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”

Raspberry Pi GPIO 连接器

> **Warning**
> 此连接器已进入生命周期延长阶段，AWS IoT Greengrass 不会发布更新来提供功能、现有功能增强、安全补丁或错误修复。有关更多信息，请参阅AWS IoT Greengrass Version 1 维护策略。

Raspberry Pi GPIO 连接器控制 Raspberry Pi 核心设备上的通用输入/输出 (GPIO) 引脚。

此连接器按指定间隔轮询输入引脚并将状态变化发布到 MQTT 主题。它还以 MQTT 消息形式接受来自用户定义的 Lambda 函数的读取和写入请求。写入请求用于将引脚设置为高压或低压。

连接器提供用于指定输入和输出引脚的参数。此行为在部署组之前配置。它无法在运行时更改。

- 输入引脚可用于从外围设备接收数据。
- 输出引脚可用于控制外围设备或将数据发送到外围设备。

您可以在许多情况下使用此连接器，例如：

- 控制交通信号灯的绿色、黄色和红色 LED 指示灯。
- 根据来自湿度传感器的数据控制风扇（连接到继电器）。
- 顾客按下按钮向零售店中的员工发出警报。
- 使用智能指示灯开关控制其他 IoT 设备。

> **Note**
>
> 此连接器不适合具有实时要求的应用。持续时间很短的事件可能会被漏掉。

此连接器具有以下版本。

版本	ARN
3	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/3`
2	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/2`
1	`arn:aws:greengrass:  region::/connectors/RaspberryPiGPIO/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 3

- AWS IoT Greengrass Core 软件 v1.9.3 版或更高版本。
- 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。
- Raspberry Pi 4 Model B 或 Raspberry Pi 3 Model B/B+ 您必须知道 Raspberry Pi 的引脚序列。有关更多信息，请参阅the section called “GPIO 引脚序列”。
- Greengrass 组中的一个本地设备资源，指向 Raspberry Pi 上的 `/dev/gpiomem`。如果您在控制台中创建资源，则必须选择自动添加拥有该资源的 Linux 组的操作系统组权限选项。在 API 中，将 `GroupOwnerSetting.AutoAddGroupOwner` 属性设置为 `true`。
- 安装在 Raspberry Pi 上的 RPi.GPIO 模块。在 Raspbian 中，默认情况下安装此模块。您可以使用以下命令重新安装它：

```
sudo pip install RPi.GPIO
```

Versions 1 - 2

- AWS IoT Greengrass Core 软件 v1.7 版或更高版本。
- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- Raspberry Pi 4 Model B 或 Raspberry Pi 3 Model B/B+ 您必须知道 Raspberry Pi 的引脚序列。有关更多信息，请参阅the section called “GPIO 引脚序列”。
- Greengrass 组中的一个本地设备资源，指向 Raspberry Pi 上的 `/dev/gpiomem`。如果您在控制台中创建资源，则必须选择自动添加拥有该资源的 Linux 组的操作系统组权限选项。在 API 中，将 `GroupOwnerSetting.AutoAddGroupOwner` 属性设置为 `true`。
- 安装在 Raspberry Pi 上的 RPi.GPIO 模块。在 Raspbian 中，默认情况下安装此模块。您可以使用以下命令重新安装它：

```
sudo pip install RPi.GPIO
```

GPIO 引脚序列

Raspberry Pi GPIO 连接器通过底层片上系统 (SoC) 的编号方案来引用 GPIO 引脚，而不是基于 GPIO 引脚的物理布局。根据不同的 Raspberry Pi 版本，引脚的物理顺序可能有所不同。有关更多信息，请参阅 Raspberry Pi 文档中的 GPIO。

连接器无法验证您所配置的输入和输出引脚是否与 Raspberry Pi 的基础硬件正常匹配。如果引脚配置无效，连接器在尝试在设备上启动时，会返回一个运行时错误。要解决此问题，请重新配置连接器，然后重新部署。

> **Note**
>
> 请确保 GPIO 引脚的外围设备布线正确，以防止组件损坏。

连接器参数

该连接器提供以下参数：

`InputGpios`

要配置为输入的 GPIO 引脚号的逗号分隔列表。可选择附加 `U` 以设置引脚的上拉电阻，或附加 `D` 以设置下拉电阻。示例：`"5,6U,7D"`。

AWS IoT 控制台中的显示名称：输入 GPIO 引脚

必需：`false`。您必须指定输入和/或输出引脚。

类型：`string`

有效模式：`^$|^[0-9]+[UD]?(,[0-9]+[UD]?)*$`

`InputPollPeriod`

每个轮询操作之间的间隔（以毫秒为单位），该操作检查输入 GPIO 引脚是否发生状态变化。最小值为 1。

此值取决于您所处的具体情况以及轮询的设备的类型。例如，值为 `50` 应该足够快速地检测到按钮按压。

AWS IoT 控制台中的显示名称：输入 GPIO 轮询周期

必需：`false`

类型：`string`

有效模式：`^$|^[1-9][0-9]*$`

`OutputGpios`

要配置为输出的 GPIO 引脚号的逗号分隔列表。可选择附加 H 以设置高状态（值为 1），或附加 L 以设置低状态（值为 0）。示例："8H,9,27L"。

AWS IoT 控制台中的显示名称：输出 GPIO 引脚

必需：`false`。您必须指定输入和/或输出引脚。

类型：`string`

有效模式：`^$|^[0-9]+[HL]?(,[0-9]+[HL]?)*$`

`GpioMem-ResourceId`

表示 `/dev/gpiomem` 的本地设备资源的 ID。

> **Note**
> 连接器被授予对该资源的读写访问权限。

AWS IoT 控制台中的显示名称：/dev/gpiomem 设备的资源

必需：`true`

类型：`string`

有效模式：`.+`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 Raspberry Pi GPIO 连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyRaspberryPiGPIOConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/
versions/3",
```

```
            "Parameters": {
                "GpioMem-ResourceId": "my-gpio-resource",
                "InputGpios": "5,6U,7D",
                "InputPollPeriod": 50,
                "OutputGpios": "8H,9,27L"
            }
        }
    ]
}'
```

> **Note**
>
> 此连接器中的 Lambda 函数的生命周期很长。

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅the section called “连接器入门（控制台）”。

输入数据

此连接器在两个 MQTT 主题上接受 GPIO 引脚的读取或写入请求。

- 读取请求位于`gpio/+/+/read` 主题上。
- 写入请求位于 `gpio/+/+/write` 主题上。

要发布到这些主题，请将 + 通配符分别替换为核心事物名称和目标引脚号。例如：

```
gpio/core-thing-name/gpio-number/read
```

> **Note**
>
> 目前，在创建使用 Raspberry Pi GPIO 连接器的订阅时，您必须为主题中的至少一个 + 通配符指定值。

主题筛选条件：`gpio/+/+/read`

使用此主题指示连接器读取该主题中指定的 GPIO 引脚的状态。

连接器会将响应发布到相应的输出主题（例如 gpio/*core-thing-name*/*gpio-number*/state）。

消息属性

无。发送到此主题的消息将被忽略。

主题筛选条件：gpio/+/+/write

使用此主题向 GPIO 引脚发送写入请求。这会指示连接器将该主题中指定的 GPIO 引脚设置为低压或电压。

- 0，将引脚设置为低压。
- 1，将引脚设置为高压。

连接器会将响应发布到相应的输出 /state 主题（例如 gpio/*core-thing-name*/*gpio-number*/state）。

消息属性

值为 0 或 1，采用整数或字符串形式。

示例输入

```
0
```

输出数据

此连接器将数据发布到两个主题：

- 高压或低压状态变化位于 gpio/+/+/state 主题上。
- 错误位于 gpio/+/error 主题上。

主题筛选条件：gpio/+/+/state

使用此主题侦听输入引脚的状态变化，并响应读取请求。如果引脚处于低状态，连接器返回字符串 "0"；如果引脚处于高状态，则返回 "1"。

当发布到此主题时，连接器会将 + 通配符分别替换为核心事物名称和目标引脚。例如：

```
gpio/core-thing-name/gpio-number/state
```

> **Note**
>
> 目前，在创建使用 Raspberry Pi GPIO 连接器的订阅时，您必须为主题中的至少一个 + 通配符指定值。

输出示例

```
0
```

主题筛选条件：`gpio/+/error`

使用此主题侦听错误。连接器因请求无效而发布到此主题（例如，当对输入引脚请求状态变化时）。

当发布到此主题，连接器会将 + 通配符替换为核心事物名称。

输出示例

```
{
   "topic": "gpio/my-core-thing/22/write",
   "error": "Invalid GPIO operation",
   "long_description": "GPIO 22 is configured as an INPUT GPIO. Write operations
 are not permitted."
 }
```

用法示例

使用以下概括步骤设置可用于尝试连接器的示例 Python 3.7 Lambda 函数。

> **Note**
>
> - 如果要使用其他 Python 运行时，您可以创建从 Python3.x 到 Python 3.7 的符号链接。
> - 连接器入门（控制台）和 连接器入门 (CLI) 主题包含详细步骤，说明如何配置和部署示例 Twilio 通知连接器。

1. 确保满足连接器的要求。
2. 创建并发布将输入数据发送到连接器的 Lambda 函数。

将示例代码保存为 PY 文件。下载并解压适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。然后，创建一个 zip 包，其中在根级别包含 PY 文件和 `greengrasssdk` 文件夹。此 zip 包是您上传到 AWS Lambda 的部署包。

创建 Python 3.7 Lambda 函数后，请发布函数版本并创建别名。

3. 配置 Greengrass 组。

   a. 通过别名来添加 Lambda 函数（推荐）。将 Lambda 生命周期配置为长时间生存（或在 CLI 中设置为 `"Pinned": true`）。
   b. 添加所需的本地设备资源并授予对 Lambda 函数的读/写访问权限。
   c. 添加连接器并配置其参数。
   d. 添加允许连接器接收输入数据并针对支持的主题筛选条件发送输出数据的订阅。

      - 将 Lambda 函数设置为源，将连接器设置为目标，并使用支持的输入主题筛选条件。
      - 将连接器设置为源，将 AWS IoT Core 设置为目标，并使用支持的输出主题筛选条件。您可以使用此订阅查看 AWS IoT 控制台中的状态消息。

4. 部署组。
5. 在 AWS IoT 控制台中的测试页面上，订阅输出数据主题以查看连接器中的状态消息。示例 Lambda 函数是长时间生存的，并且在部署组后立即开始发送消息。

   完成测试后，您可以将 Lambda 生命周期设置为按需（或在 CLI 中设置为 `"Pinned": false`）并部署组。这会阻止函数发送消息。

示例

以下示例 Lambda 函数向连接器发送一条输入消息。 此示例发送一组输入 GPIO 引脚的读取请求。它显示了如何使用核心事物名称和引脚号构建主题。

```
import greengrasssdk
import json
import os

iot_client = greengrasssdk.client('iot-data')
INPUT_GPIOS = [6, 17, 22]

thingName = os.environ['AWS_IOT_THING_NAME']

def get_read_topic(gpio_num):
```

```
    return '/'.join(['gpio', thingName, str(gpio_num), 'read'])

def get_write_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'write'])

def send_message_to_connector(topic, message=''):
    iot_client.publish(topic=topic, payload=str(message))

def set_gpio_state(gpio, state):
    send_message_to_connector(get_write_topic(gpio), str(state))

def read_gpio_state(gpio):
    send_message_to_connector(get_read_topic(gpio))

def publish_basic_message():
    for i in INPUT_GPIOS:
     read_gpio_state(i)

publish_basic_message()

def lambda_handler(event, context):
    return
```

许可证

Raspberry Pi GPIO 连接器包含以下第三方软件/许可：

- RPi.GPIO/MIT

此连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
3	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。
2	更新了连接器 ARN 以获得 AWS 区域 支持。

版本	更改
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”
- Raspberry Pi 文档中的 GPIO

串行流连接器

> **Warning**
>
> 此连接器已进入生命周期延长阶段，AWS IoT Greengrass 不会发布更新来提供功能、现有功能增强、安全补丁或错误修复。有关更多信息，请参阅AWS IoT Greengrass Version 1 维护策略。

串行流连接器读取和写入 AWS IoT Greengrass 核心设备上的串行端口。

此连接器支持两种操作模式：

- 按需读取。接收关于 MQTT 主题的读取和写入请求，并发布读取操作的响应或写入操作的状态。
- 轮询读取。定期从串行端口读取。该模式还支持“按需读取”请求。

> **Note**
>
> 读取请求的限制为最多 63994 字节的读取长度。写入请求的限制为最多 128000 字节的数据长度。

此连接器具有以下版本。

版本	ARN
3	`arn:aws:greengrass:region::/connectors/SerialStream/versions/3`
2	`arn:aws:greengrass:region::/connectors/SerialStream/versions/2`
1	`arn:aws:greengrass:region::/connectors/SerialStream/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 3

- AWS IoT Greengrass Core 软件 v1.9.3 版或更高版本。
- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

  > **Note**
  >
  > 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- Greengrass 组中一个指向目标串行端口的本地设备资源。

> **Note**
>
> 建议先设置串行端口并验证可读取和写入此串行端口，然后部署此连接器。

Versions 1 - 2

- AWS IoT Greengrass Core 软件 v1.7 版或更高版本。
- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- Greengrass 组中一个指向目标串行端口的本地设备资源。

> **Note**
>
> 建议先设置串行端口并验证可读取和写入此串行端口，然后部署此连接器。

连接器参数

该连接器提供以下参数：

`BaudRate`

串行连接的波特率。

AWS IoT 控制台中的显示名称：波特率

必需：`true`

类型：`string`

有效值：`110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 230400`

有效模式：`^110$|^300$|^600$|^1200$|^2400$|^4800$|^9600$|^14400$|^19200$|^28800$|^38400$|^56000$|^57600$|^115200$|^230400$`

`Timeout`

读取操作的超时（以秒为单位）。

AWS IoT 控制台中的显示名称：超时

必需：`true`

类型：`string`

有效值：`1 - 59`

有效模式：`^([1-9]|[1-5][0-9])$`

`SerialPort`

设备上的物理串行端口的绝对路径。这是为本地设备资源指定的源路径。

AWS IoT 控制台中的显示名称：串行端口

必需：`true`

类型：`string`

有效模式：`[/a-zA-Z0-9_-]+`

`SerialPort-ResourceId`

表示物理串行端口的本地设备资源的 ID。

> **Note**
> 连接器被授予对该资源的读写访问权限。

AWS IoT 控制台中的显示名称：串行端口资源

必需：`true`

类型：`string`

有效模式：`[a-zA-Z0-9_-]+`

`PollingRead`

设置读取模式：“轮询读取”或“按需读取”。

- 对于“轮询读取”模式，请指定 `true`。在该模式下，`PollingInterval`、`PollingReadType` 和 `PollingReadLength` 属性是必需属性。
- 对于“按需读取”模式，请指定 `false`。在该模式下，将在读取请求中指定类型和长度值。

AWS IoT 控制台中的显示名称：读取模式

必需：`true`

类型：`string`

有效值：`true, false`

有效模式：`^([Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`PollingReadLength`

要在每个轮询读取操作中读取的数据长度（字节）。仅当使用“轮询读取”模式时才适用。

AWS IoT 控制台中的显示名称：轮询读取时长

必需：`false`。当 `PollingRead` 为 `true` 时，该属性是必需属性。

类型：`string`

有效模式：`^(|[1-9][0-9]{0,3}|[1-5][0-9]{4}|6[0-2][0-9]{3}|63[0-8][0-9]{2}|639[0-8][0-9]|6399[0-4])$`

`PollingReadInterval`

轮询读取的时间间隔（以秒为单位）。仅当使用“轮询读取”模式时才适用。

AWS IoT 控制台中的显示名称：轮询读取间隔

必需：`false`。当 `PollingRead` 为 `true` 时，该属性是必需属性。

类型：`string`

有效值：1 - 999

有效模式：`^(|[1-9]|[1-9][0-9]|[1-9][0-9][0-9])$`

`PollingReadType`

轮询线程读取的数据类型。仅当使用“轮询读取”模式时才适用。

AWS IoT 控制台中的显示名称：轮询读取类型

必需：false。当 PollingRead 为 true 时，该属性是必需属性。

类型：string

有效值：ascii, hex

有效模式：^(|[Aa][Ss][Cc][Ii][Ii]|[Hh][Ee][Xx])$

RtsCts

指示是否启用 RTS/CTS 流控制。默认值为 false。有关更多信息，请参阅 RTS、CTS 和 RTR。

AWS IoT 控制台中的显示名称：RTS/CTS 流量控制

必需：false

类型：string

有效值：true, false

有效模式：^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$

XonXoff

指示是否启用软件流控制。默认值为 false。有关更多信息，请参阅软件流控制。

AWS IoT 控制台中的显示名称：软件流量控制

必需：false

类型：string

有效值：true, false

有效模式：^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$

Parity

串行端口的奇偶校验。默认值为 N。有关更多信息，请参阅奇偶校验。

AWS IoT 控制台中的显示名称：串行端口奇偶校验

必需：`false`

类型：`string`

有效值：`N, E, O, S, M`

有效模式：`^(|[NEOSMneosm])$`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含串行流连接器。它将连接器配置为“轮询读取”模式。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySerialStreamConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SerialStream/
versions/3",
            "Parameters": {
                "BaudRate" : "9600",
                "Timeout" : "25",
                "SerialPort" : "/dev/serial1",
                "SerialPort-ResourceId" : "my-serial-port-resource",
                "PollingRead" : "true",
                "PollingReadLength" : "30",
                "PollingReadInterval" : "30",
                "PollingReadType" : "hex"
            }
        }
    ]
}'
```

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅the section called “连接器入门（控制台）”。

输入数据

此连接器接受关于两个 MQTT 主题的串行端口读取或写入请求。输入消息必须采用 JSON 格式。

- 读取请求位于`serial/+/read/#` 主题上。

- 写入请求位于 serial/+/write/# 主题上。

要发布到这些主题，请将 + 通配符替换为核心事物名称，将 # 通配符替换为串行端口的路径。例如：

```
serial/core-thing-name/read/dev/serial-port
```

主题筛选条件：serial/+/read/#

使用此主题向串行引脚发送按需读取请求。读取请求的限制为最多 63994 字节的读取长度。

消息属性

readLength

要从串行端口读取的数据长度。

必需：true

类型：string

有效模式：^[1-9][0-9]*$

type

要读取的数据的类型。

必需：true

类型：string

有效值：ascii, hex

有效模式：(?i)^(ascii|hex)$

id

请求的任意 ID。此属性用于将输入请求映射到输出响应。

必需：false

类型：string

有效模式：.+

示例输入

```
{
    "readLength": "30",
    "type": "ascii",
    "id": "abc123"
}
```

主题筛选条件：serial/+/write/#

使用此主题向串行引脚发送写入请求。写入请求的限制为最多 128000 字节的数据长度。

消息属性

data

要写入到该串行端口的字符串。

必需：true

类型：string

有效模式：^[1-9][0-9]*$

type

要读取的数据的类型。

必需：true

类型：string

有效值：ascii, hex

有效模式：^(ascii|hex|ASCII|HEX)$

id

请求的任意 ID。此属性用于将输入请求映射到输出响应。

必需：false

类型：string

有效模式：`.+`

示例输入：ASCII 请求

```
{
    "data": "random serial data",
    "type": "ascii",
    "id": "abc123"
}
```

示例输入：十六进制请求

```
{
    "data": "base64 encoded data",
    "type": "hex",
    "id": "abc123"
}
```

输出数据

连接器发布关于两个主题的输出数据：

- 来自连接器的关于 `serial/+/status/#` 主题的状态信息。
- 来自读取请求的响应（关于 `serial/+/read_response/#` 主题）。

当发布到此主题时，连接器会将 `+` 通配符替换为核心事物名称，并将 `#` 通配符替换为串行端口的路径。例如：

```
serial/core-thing-name/status/dev/serial-port
```

主题筛选条件：`serial/+/status/#`

使用此主题侦听读取和写入请求的状态。如果请求包含 `id` 属性，则该属性将在响应中返回。

示例输出：成功

```
{
    "response": {
        "status": "success"
    },
```

```
    "id": "abc123"
}
```

示例输出：失败

失败响应包含一个 `error_message` 属性，用于描述在执行读取或写入操作时遇到的错误或超时。

```
{
    "response": {
        "status": "fail",
        "error_message": "Could not write to port"
    },
    "id": "abc123"
}
```

主题筛选条件：`serial/+/read_response/#`

使用此主题接收来自读取操作的响应数据。如果类型为 `hex`，则响应数据经过 Base64 编码。

输出示例

```
{
    "data": "output of serial read operation"
    "id": "abc123"
}
```

用法示例

使用以下概括步骤设置可用于尝试连接器的示例 Python 3.7 Lambda 函数。

> **Note**
>
> - 如果要使用其他 Python 运行时，您可以创建从 Python3.x 到 Python 3.7 的符号链接。
> - 连接器入门（控制台）和连接器入门 (CLI) 主题包含详细步骤，说明如何配置和部署示例 Twilio 通知连接器。

1. 确保满足连接器的要求。

2. 创建并发布将输入数据发送到连接器的 Lambda 函数。

   将示例代码保存为 PY 文件。下载并解压适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。然后，创建一个 zip 包，其中在根级别包含 PY 文件和 `greengrasssdk` 文件夹。此 zip 包是您上传到 AWS Lambda 的部署包。

   创建 Python 3.7 Lambda 函数后，请发布函数版本并创建别名。

3. 配置 Greengrass 组。

   a. 通过别名来添加 Lambda 函数（推荐）。将 Lambda 生命周期配置为长时间生存（或在 CLI 中设置为 `"Pinned": true`）。

   b. 添加所需的本地设备资源并授予对 Lambda 函数的读/写访问权限。

   c. 将连接器添加到组并配置其参数。

   d. 向组添加允许连接器接收输入数据并针对支持的主题筛选条件发送输出数据的订阅。

      - 将 Lambda 函数设置为源，将连接器设置为目标，并使用支持的输入主题筛选条件。
      - 将连接器设置为源，将 AWS IoT Core 设置为目标，并使用支持的输出主题筛选条件。您可以使用此订阅查看 AWS IoT 控制台中的状态消息。

4. 部署组。

5. 在 AWS IoT 控制台中，在测试页面上，订阅输出数据主题以查看连接器中的状态消息。示例 Lambda 函数是长时间生存的，并且在部署组后立即开始发送消息。

   完成测试后，您可以将 Lambda 生命周期设置为按需（或在 CLI 中设置为 `"Pinned": false`）并部署组。这会阻止函数发送消息。

示例

以下示例 Lambda 函数向连接器发送一条输入消息。

```
import greengrasssdk
import json

TOPIC_REQUEST = 'serial/CORE_THING_NAME/write/dev/serial1'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_serial_stream_request():
 request = {
```

```
  "data": "TEST",
  "type": "ascii",
  "id": "abc123"
 }
 return request

def publish_basic_request():
 iot_client.publish(payload=json.dumps(create_serial_stream_request()),
 topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
 return
```

许可证

串行流连接器包含以下第三方软件/许可：

- pyserial/BSD

该连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
3	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。
2	更新了连接器 ARN 以获得 AWS 区域 支持。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”

ServiceNow MetricBase 集成连接器

> **Warning**
>
> 此连接器已进入生命周期延长阶段，AWS IoT Greengrass 不会发布更新来提供功能、现有功能增强、安全补丁或错误修复。有关更多信息，请参阅AWS IoT Greengrass Version 1 维护策略。

ServiceNow MetricBase 集成连接器将时间序列指标从 Greengrass 设备发布到 ServiceNow MetricBase。这样一来，您可以存储、分析和可视化 Greengrass 核心环境中的时间序列数据，并对本地事件执行操作。

此连接器定期接收关于 MQTT 主题的时间序列数据，并将数据发布到 ServiceNow API。

您可以使用此连接器支持如下情况：

- 根据从 Greengrass 设备收集的时间序列数据，创建基于阈值的提醒和警报。
- 使用 Greengrass 设备中的时间服务数据，且自定义应用程序在 ServiceNow 平台上构建。

此连接器具有以下版本。

版本	ARN
4	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/4`
3	`arn:aws:greengrass:region::/connectors/ServiceNowMetricBaseIntegration/versions/3`

版本	ARN
2	`arn:aws:greengrass:  region::/connectors/ServiceNowMetricBaseIntegration/versions/2`
1	`arn:aws:greengrass:  region::/connectors/ServiceNowMetricBaseIntegration/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 3 - 4

- AWS IoT Greengrass Core 软件 v1.9.3 或更高版本。AWS IoT Greengrass 必须配置为支持本地密钥，如密钥要求中所述。

  > **Note**
  >
  > 此要求包括允许访问您的 Secrets Manager 密钥。如果使用的是默认 Greengrass 服务角色，则 Greengrass 有权获得名称以 greengrass- 开头的密钥的值。

- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

  > **Note**
  >
  > 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- 激活了 MetricBase 订阅的 ServiceNow 账户。此外，必须在账户中创建指标和指标表。有关更多信息，请参阅 ServiceNow 文档中的 MetricBase。

- AWS Secrets Manager 中的文本类型密钥，存储通过基本身份验证登录到 ServiceNow 实例的用户名和密码。该密钥必须包含“用户”和“密码”键及相应的值。有关更多信息，请参阅 AWS Secrets Manager 用户指南中的创建基本密钥。
- Greengrass 组中引用 Secrets Manager 密钥的密钥资源。有关更多信息，请参阅将密钥部署到核心。

Versions 1 - 2

- AWS IoT Greengrass Core 软件 v1.7 或更高版本。AWS IoT Greengrass 必须配置为支持本地密钥，如密钥要求中所述。

  > **Note**
  >
  > 此要求包括允许访问您的 Secrets Manager 密钥。如果使用的是默认 Greengrass 服务角色，则 Greengrass 有权获得名称以 greengrass- 开头的密钥的值。

- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- 激活了 MetricBase 订阅的 ServiceNow 账户。此外，必须在账户中创建指标和指标表。有关更多信息，请参阅 ServiceNow 文档中的 MetricBase。
- AWS Secrets Manager 中的文本类型密钥，存储通过基本身份验证登录到 ServiceNow 实例的用户名和密码。该密钥必须包含“用户”和“密码”键及相应的值。有关更多信息，请参阅 AWS Secrets Manager 用户指南中的创建基本密钥。
- Greengrass 组中引用 Secrets Manager 密钥的密钥资源。有关更多信息，请参阅将密钥部署到核心。

连接器参数

该连接器提供以下参数：

Version 4

`PublishInterval`

将事件发布到 ServiceNow 需要等待的最长时间间隔（秒数）。最大值为 900。

当 `PublishBatchSize` 到达或 `PublishInterval` 过期时，连接器发布到 ServiceNow。

AWS IoT 控制台中的显示名称：以秒为单位的发布间隔

必需：true

类型：string

有效值：1 - 900

有效模式：[1-9]|[1-9]\d|[1-9]\d\d|900

PublishBatchSize

可在发布到 ServiceNow 之前进行批处理的指标值的最大数量。

当 PublishBatchSize 到达或 PublishInterval 过期时，连接器发布到 ServiceNow。

AWS IoT 控制台中的显示名称：发布批处理大小

必需：true

类型：string

有效模式：^[0-9]+$

InstanceName

用于连接到 ServiceNow 的实例的名称。

AWS IoT 控制台中显示名称：ServiceNow 实例的名称

必需：true

类型：string

有效模式：.+

DefaultTableName

包含与时间序列 MetricBase 数据库关联的 GlideRecord 的表的名称。输入消息负载中的 table 属性可用于覆盖该值。

AWS IoT 控制台中的显示名称：包含指标的表的名称

必需：true

类型：string

有效模式：.+

`MaxMetricsToRetain`

替换为新指标之前可在内存中保存的指标的最大数量。

此限制适用于没有 Internet 连接并且连接器稍后开始缓冲要发布的指标的情况。当缓冲区已满时，最旧的指标将被新的指标所替换。

> **Note**
>
> 如果连接器的宿主进程中断，则不会保存指标。例如，在组部署或设备重新启动期间，可能会发生此情况。

该值应大于批处理大小，且大到足以根据传入 MQTT 消息的速率来容纳消息。

AWS IoT 控制台中的显示名称：内存中最多可保留的指标的数量

必需：`true`

类型：`string`

有效模式：`^[0-9]+$`

`AuthSecretArn`

AWS Secrets Manager 中用于存储 ServiceNow 用户名和密码的密钥。这必须是文本类型密钥。该密钥必须包含“用户”和“密码”键及相应的值。

AWS IoT 控制台中的显示名称：身份验证密钥的 ARN

必需：`true`

类型：`string`

有效模式：`arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`AuthSecretArn-ResourceId`

组中引用 ServiceNow 凭证的 Secrets Manager 密钥的密钥资源。

AWS IoT 控制台中的显示名称：身份验证令牌资源

必需：`true`

类型：`string`

有效模式：`.+`

`IsolationMode`

此连接器的容器化模式。默认值为 `GreengrassContainer`，这意味着连接器在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

> **Note**
> 组的默认容器化设置不适用于连接器。

AWS IoT 控制台中的显示名称：容器隔离模式

必需：`false`

类型：`string`

有效值：`GreengrassContainer` 或 `NoContainer`

有效模式：`^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`PublishInterval`

将事件发布到 ServiceNow 需要等待的最长时间间隔（秒数）。最大值为 900。

当 `PublishBatchSize` 到达或 `PublishInterval` 过期时，连接器发布到 ServiceNow。

AWS IoT 控制台中的显示名称：以秒为单位的发布间隔

必需：`true`

类型：`string`

有效值：`1 - 900`

有效模式：`[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

可在发布到 ServiceNow 之前进行批处理的指标值的最大数量。

当 PublishBatchSize 到达或 PublishInterval 过期时，连接器发布到 ServiceNow。

AWS IoT 控制台中的显示名称：发布批处理大小

必需：true

类型：string

有效模式：^[0-9]+$

InstanceName

用于连接到 ServiceNow 的实例的名称。

AWS IoT 控制台中显示名称：ServiceNow 实例的名称

必需：true

类型：string

有效模式：.+

DefaultTableName

包含与时间序列 MetricBase 数据库关联的 GlideRecord 的表的名称。输入消息负载中的 table 属性可用于覆盖该值。

AWS IoT 控制台中的显示名称：包含指标的表的名称

必需：true

类型：string

有效模式：.+

MaxMetricsToRetain

替换为新指标之前可在内存中保存的指标的最大数量。

此限制适用于没有 Internet 连接并且连接器稍后开始缓冲要发布的指标的情况。当缓冲区已满时，最旧的指标将被新的指标所替换。

> **Note**
>
> 如果连接器的宿主进程中断，则不会保存指标。例如，在组部署或设备重新启动期间，可能会发生此情况。

该值应大于批处理大小，且大到足以根据传入 MQTT 消息的速率来容纳消息。

AWS IoT 控制台中的显示名称：内存中最多可保留的指标的数量

必需：`true`

类型：`string`

有效模式：`^[0-9]+$`

`AuthSecretArn`

AWS Secrets Manager 中用于存储 ServiceNow 用户名和密码的密钥。这必须是文本类型密钥。该密钥必须包含“用户”和“密码”键及相应的值。

AWS IoT 控制台中的显示名称：身份验证密钥的 ARN

必需：`true`

类型：`string`

有效模式：`arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`AuthSecretArn-ResourceId`

组中引用 ServiceNow 凭证的 Secrets Manager 密钥的密钥资源。

AWS IoT 控制台中的显示名称：身份验证令牌资源

必需：`true`

类型：`string`

有效模式：`.+`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 ServiceNow MetricBase 集成连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
```

```
            "Id": "MyServiceNowMetricBaseIntegrationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ServiceNowMetricBaseIntegration/versions/4",
            "Parameters": {
                "PublishInterval" : "10",
                "PublishBatchSize" : "50",
                "InstanceName" : "myinstance",
                "DefaultTableName" : "u_greengrass_app",
                "MaxMetricsToRetain" : "20000",
                "AuthSecretArn" : "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "AuthSecretArn-ResourceId" : "MySecretResource",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

> **Note**
>
> 此连接器中的 Lambda 函数的生命周期很长。

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅the section called “连接器入门（控制台）”。

输入数据

此连接器接受关于 MQTT 主题的时间序列指标，并将指标发布到 ServiceNow。输入消息必须采用 JSON 格式。

订阅中的主题筛选条件

`servicenow/metricbase/metric`

消息属性

`request`

有关表、记录和指标的信息。该请求表示时间序列 POST 请求中的 `seriesRef` 对象。有关更多信息，请参阅 Clotho 时间序列 API - POST。

必需：`true`

类型：包含以下属性的 object：

subject

表中特定记录的 sys_id。

必需：true

类型：string

metric_name

指标字段名称。

必需：true

类型：string

table

用于存储记录的表的名称。指定该值以覆盖 DefaultTableName 参数。

必需：false

类型：string

value

单个数据点的值。

必需：true

类型：float

timestamp

单个数据点的时间戳。默认值为当前时间。

必需：false

类型：string

示例输入

```
{
    "request": {
        "subject":"ef43c6d40a0a0b5700c77f9bf387afe3",
        "metric_name":"u_count",
        "table": "u_greengrass_app"
```

```
        "value": 1.0,
        "timestamp": "2018-10-14T10:30:00"
    }
}
```

输出数据

此连接器将状态信息发布为 MQTT 主题的输出数据。

订阅中的主题筛选条件

```
servicenow/metricbase/metric/status
```

示例输出：成功

```
{
    "response": {
        "metric_name": "Errors",
        "table_name": "GliderProd",
        "processed_on": "2018-10-14T10:35:00",
        "response_id": "khjKSkj132qwr23fcba",
        "status": "success",
        "values": [
            {
                "timestamp": "2016-10-14T10:30:00",
                "value": 1.0
            },
            {
                "timestamp": "2016-10-14T10:31:00",
                "value": 1.1
            }
        ]
    }
}
```

示例输出：失败

```
{
    "response": {
        "error": "InvalidInputException",
        "error_message": "metric value is invalid",
        "status": "fail"
    }
```

```
}
```

> **Note**
>
> 如果连接器检测到可重试的错误（如连接错误），它会在下一批次中重试发布。

用法示例

使用以下概括步骤设置可用于尝试连接器的示例 Python 3.7 Lambda 函数。

> **Note**
>
> - 如果要使用其他 Python 运行时，您可以创建从 Python3.x 到 Python 3.7 的符号链接。
> - 连接器入门（控制台）和连接器入门 (CLI) 主题包含详细步骤，说明如何配置和部署示例 Twilio 通知连接器。

1. 确保满足连接器的要求。
2. 创建并发布将输入数据发送到连接器的 Lambda 函数。

   将示例代码保存为 PY 文件。下载并解压适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。然后，创建一个 zip 包，其中在根级别包含 PY 文件和 `greengrasssdk` 文件夹。此 zip 包是您上传到 AWS Lambda 的部署包。

   创建 Python 3.7 Lambda 函数后，请发布函数版本并创建别名。
3. 配置 Greengrass 组。

   a. 通过别名来添加 Lambda 函数（推荐）。将 Lambda 生命周期配置为长时间生存（或在 CLI 中设置为 `"Pinned": true`）。

   b. 添加所需的密钥资源并授予对 Lambda 函数的读取访问权限。

   c. 添加连接器并配置其参数。

   d. 添加允许连接器接收输入数据并针对支持的主题筛选条件发送输出数据的订阅。

      - 将 Lambda 函数设置为源，将连接器设置为目标，并使用支持的输入主题筛选条件。
      - 将连接器设置为源，将 AWS IoT Core 设置为目标，并使用支持的输出主题筛选条件。您可以使用此订阅查看 AWS IoT 控制台中的状态消息。

4. 部署组。
5. 在 AWS IoT 控制台中，在测试页面上，订阅输出数据主题以查看连接器中的状态消息。示例 Lambda 函数是长时间生存的，并且在部署组后立即开始发送消息。

   完成测试后，您可以将 Lambda 生命周期设置为按需（或在 CLI 中设置为 `"Pinned": false`）并部署组。这会阻止函数发送消息。

示例

以下示例 Lambda 函数向连接器发送一条输入消息。

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
SEND_TOPIC = 'servicenow/metricbase/metric'

def create_request_with_all_fields():
    return {
        "request": {
             "subject": '2efdf6badbd523803acfae441b961961',
             "metric_name": 'u_count',
             "value": 1234,
             "timestamp": '2018-10-20T20:22:20',
             "table": 'u_greengrass_metricbase_test'
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=SEND_TOPIC,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

许可证

ServiceNow MetricBase 集成连接器包含以下第三方软件/许可：

- pysnow/MIT

该连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
4	增加了用于配置连接器容器化模式的 `IsolationMode` 参数。
3	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。
2	进行了修复，以减少过多的日志记录。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”

SNS 连接器

SNS 连接器将消息发布到 Amazon SNS 主题。这使 Web 服务器、电子邮件地址和其他消息订阅者能够响应 Greengrass 组中的事件。

该连接器接收关于 MQTT 主题的 SNS 消息信息，然后将消息发送到指定的 SNS 主题。您可以视情况使用自定义 Lambda 函数对消息实施筛选或格式化逻辑，然后再将消息发布到该连接器。

此连接器具有以下版本。

版本	ARN
4	arn:aws:greengrass:*region*::/connectors/SNS/versions/4
3	arn:aws:greengrass:*region*::/connectors/SNS/versions/3
2	arn:aws:greengrass:*region*::/connectors/SNS/versions/2
1	arn:aws:greengrass:*region*::/connectors/SNS/versions/1

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 3 - 4

- AWS IoT Greengrass Core 软件 v1.9.3 版或更高版本。
- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

  > **Note**
  >
  > 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- 配置的 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 组角色，配置为允许对目标 Amazon SNS 主题执行 `sns:Publish` 操作，如以下示例 IAM 策略中所示。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "sns:Publish"
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:sns:region:account-id:topic-name"
            ]
        }
    ]
}
```

利用该连接器，您可以动态覆盖输入消息负载中的默认主题。如果您的实施使用此功能，IAM 策略必须允许对所有目标主题的 `sns:Publish` 权限。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）。

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

Versions 1 - 2

- AWS IoT Greengrass Core 软件 v1.7 版或更高版本。
- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- 配置的 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 组角色，配置为允许对目标 Amazon SNS 主题执行 `sns:Publish` 操作，如以下示例 IAM 策略中所示。

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "sns:Publish"
```

```
            ],
            "Effect":"Allow",
            "Resource":[
                "arn:aws:sns:region:account-id:topic-name"
            ]
        }
    ]
}
```

利用该连接器，您可以动态覆盖输入消息负载中的默认主题。如果您的实施使用此功能，IAM 策略必须允许对所有目标主题的 `sns:Publish` 权限。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）。

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

连接器参数

该连接器提供以下参数：

Version 4

`DefaultSNSArn`

要将消息发布到的默认 SNS 主题的 ARN。目标主题可由输入消息负载中的 `sns_topic_arn` 属性覆盖。

> **Note**
>
> 该组角色必须允许对所有目标主题的 `sns:Publish` 权限。有关更多信息，请参阅 the section called “要求”。

在 AWS IoT 控制台中显示名称：默认 SNS 主题 ARN

必需：`true`

类型：`string`

有效模式：`arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

`IsolationMode`

此连接器的容器化模式。默认值为 `GreengrassContainer`，这意味着连接器在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

> **Note**
>
> 组的默认容器化设置不适用于连接器。

AWS IoT 控制台中的显示名称：容器隔离模式

必需：`false`

类型：`string`

有效值：`GreengrassContainer` 或 `NoContainer`

有效模式：`^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

`DefaultSNSArn`

要将消息发布到的默认 SNS 主题的 ARN。目标主题可由输入消息负载中的 `sns_topic_arn` 属性覆盖。

> **Note**
>
> 该组角色必须允许对所有目标主题的 `sns:Publish` 权限。有关更多信息，请参阅 the section called “要求”。

在 AWS IoT 控制台中显示名称：默认 SNS 主题 ARN

必需：`true`

类型：`string`

有效模式：`arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 SNS 连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySNSConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SNS/versions/4",
            "Parameters": {
                "DefaultSNSArn": "arn:aws:sns:region:account-id:topic-name",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅 the section called “连接器入门（控制台）”。

输入数据

此连接器接受关于 MQTT 主题的 SNS 消息信息，然后将信息按原样发布到目标 SNS 主题。输入消息必须采用 JSON 格式。

订阅中的主题筛选条件

`sns/message`

消息属性

`request`

有关要发送到 SNS 主题的消息的信息。

必需：`true`

类型：包含以下属性的 `object`：

`message`

字符串或 JSON 格式的消息内容。有关示例，请参阅示例输入。

要发送 JSON，必须将 message_structure 属性设置为 json，并且消息必须是包含 default 密钥的字符串编码的 JSON 对象。

必需：true

类型：string

有效模式：.*

subject

消息主题。

必需：false

类型：ASCII 文本，最多 100 个字符。必须以字母、数字或标点符号开头。不得包含换行符或控制字符。

有效模式：.*

sns_topic_arn

要将消息发布到的 SNS 主题的 ARN。如果指定，则连接器将发布到该主题，而不是默认的主题。

> **Note**
>
> 该组角色必须允许对任何目标主题的 sns:Publish 权限。有关更多信息，请参阅 the section called “要求”。

必需：false

类型：string

有效模式：arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$

message_structure

消息结构。

必需：false。要发送 JSON 消息，必须指定此属性。

类型：string

有效值：json

id

请求的任意 ID。此属性用于将输入请求映射到输出响应。如果指定，响应对象中的 id 属性将设置为该值。如果您不使用此功能，可以忽略此属性或指定空字符串。

必需：false

类型：string

有效模式：.*

限制

消息大小受最大 SNS 消息大小 (256 KB) 的限制。

示例输入：字符串消息

此示例发送一条字符串消息。它指定可选的 sns_topic_arn 属性，该属性会覆盖默认目标主题。

```
{
    "request": {
        "subject": "Message subject",
        "message": "Message data",
        "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
    },
    "id": "request123"
}
```

示例输入：JSON 消息

此示例以包含 default 密钥的字符串编码的 JSON 对象形式发送消息。

```
{
    "request": {
        "subject": "Message subject",
        "message": "{ \"default\": \"Message data\" }",
        "message_structure": "json"
    },
    "id": "request123"
}
```

输出数据

此连接器将状态信息发布为 MQTT 主题的输出数据。

订阅中的主题筛选条件

```
sns/message/status
```

示例输出：成功

```
{
    "response": {
        "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
        "status": "success"
    },
    "id": "request123"
}
```

示例输出：失败

```
{
   "response" : {
        "error": "InvalidInputException",
        "error_message": "SNS Topic Arn is invalid",
        "status": "fail"
   },
   "id": "request123"
}
```

用法示例

使用以下概括步骤设置可用于尝试连接器的示例 Python 3.7 Lambda 函数。

> **Note**
>
> - 如果要使用其他 Python 运行时，您可以创建从 Python3.x 到 Python 3.7 的符号链接。
> - 连接器入门（控制台）和 连接器入门 (CLI) 主题包含详细步骤，说明如何配置和部署示例 Twilio 通知连接器。

1. 确保满足连接器的要求。

对于组角色要求，您必须将角色配置为授予所需权限，并确保角色已添加到组中。有关更多信息，请参阅the section called “管理组角色（控制台）”或the section called “管理组角色 (CLI)”。

2. 创建并发布将输入数据发送到连接器的 Lambda 函数。

   将示例代码保存为 PY 文件。下载并解压适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。然后，创建一个 zip 包，其中在根级别包含 PY 文件和 `greengrasssdk` 文件夹。此 zip 包是您上传到 AWS Lambda 的部署包。

   创建 Python 3.7 Lambda 函数后，请发布函数版本并创建别名。

3. 配置 Greengrass 组。

   a. 通过别名来添加 Lambda 函数（推荐）。将 Lambda 生命周期配置为长时间生存（或在 CLI 中设置为 `"Pinned": true`）。

   b. 添加连接器并配置其参数。

   c. 添加允许连接器接收输入数据并针对支持的主题筛选条件发送输出数据的订阅。

      - 将 Lambda 函数设置为源，将连接器设置为目标，并使用支持的输入主题筛选条件。
      - 将连接器设置为源，将 AWS IoT Core 设置为目标，并使用支持的输出主题筛选条件。您可以使用此订阅查看 AWS IoT 控制台中的状态消息。

4. 部署组。

5. 在 AWS IoT 控制台中的测试页面上，订阅输出数据主题以查看连接器中的状态消息。示例 Lambda 函数是长时间生存的，并且在部署组后立即开始发送消息。

   完成测试后，您可以将 Lambda 生命周期设置为按需（或在 CLI 中设置为 `"Pinned": false`）并部署组。这会阻止函数发送消息。

示例

以下示例 Lambda 函数向连接器发送一条输入消息。

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'sns/message'
```

```
def create_request_with_all_fields():
    return  {
        "request": {
            "message": "Message from SNS Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

许可证

SNS 连接器包含以下第三方软件/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

此连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
4	增加了用于配置连接器容器化模式的 `IsolationMode` 参数。
3	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。
2	进行了修复，以减少过多的日志记录。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”
- Boto 3 文档中的发布操作
- Amazon Simple Notification Service 开发人员指南中的什么是 Amazon Simple Notification Service？

Splunk 集成连接器

> **Warning**
>
> 此连接器已进入生命周期延长阶段，AWS IoT Greengrass 不会发布更新来提供功能、现有功能增强、安全补丁或错误修复。有关更多信息，请参阅AWS IoT Greengrass Version 1 维护策略。

Splunk 集成连接器将数据从 Greengrass 设备发布到 Splunk。这样的话，您可以使用 Splunk 来监控和分析 Greengrass 核心环境，并对本地事件执行操作。连接器与 HTTP 事件收集器 (HEC) 集成。有关更多信息，请参阅 Splunk 文档中的 Splunk HTTP 事件收集器简介。

此连接器接收关于 MQTT 主题的日志记录和事件数据，并按原样将数据发布到 Splunk API。

您可以使用该连接器支持行业方案，比如：

- 操作员可以使用致动器和传感器的周期性数据（例如，温度、压力和水分读数）在值超出特定阈值时启动警报。
- 开发人员使用从工业机械收集的数据来构建 ML 模型，以监控设备发现潜在问题。

此连接器具有以下版本。

版本	ARN
4	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/4`
3	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/3`
2	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/2`
1	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 3 - 4

- AWS IoT Greengrass Core 软件 v1.9.3 或更高版本。AWS IoT Greengrass 必须配置为支持本地密钥，如密钥要求中所述。

> **Note**
>
> 此要求包括允许访问您的 Secrets Manager 密钥。如果使用的是默认 Greengrass 服务角色，则 Greengrass 有权获得名称以 greengrass- 开头的密钥的值。

- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

  > **Note**
  >
  > 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- 必须在 Splunk 中启用 HTTP 事件收集器功能。有关更多信息，请参阅 Splunk 文档中的在 Splunk Web 中设置和使用 HTTP 事件收集器。
- AWS Secrets Manager 中的文本类型密钥，用于存储您的 Splunk HTTP 事件收集器令牌。有关更多信息，请参阅 Splunk 文档中的关于事件收集器令牌和AWS Secrets Manager 用户指南中的创建基本密钥。

  > **Note**
  >
  > 要在 Secrets Manager 控制台中创建密钥，请在明文选项卡上输入您的令牌。请勿包含引号或其他格式。在 API 中，将令牌指定为 `SecretString` 属性的值。

- Greengrass 组中引用 Secrets Manager 密钥的密钥资源。有关更多信息，请参阅将密钥部署到核心。

Versions 1 - 2

- AWS IoT Greengrass Core 软件 v1.7 或更高版本。AWS IoT Greengrass 必须配置为支持本地密钥，如密钥要求中所述。

> **Note**
>
> 此要求包括允许访问您的 Secrets Manager 密钥。如果使用的是默认 Greengrass 服务角色，则 Greengrass 有权获得名称以 greengrass- 开头的密钥的值。

- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- 必须在 Splunk 中启用 HTTP 事件收集器功能。有关更多信息，请参阅 Splunk 文档中的在 Splunk Web 中设置和使用 HTTP 事件收集器。
- AWS Secrets Manager 中的文本类型密钥，用于存储您的 Splunk HTTP 事件收集器令牌。有关更多信息，请参阅 Splunk 文档中的关于事件收集器令牌和AWS Secrets Manager 用户指南中的创建基本密钥。

> **Note**
>
> 要在 Secrets Manager 控制台中创建密钥，请在明文选项卡上输入您的令牌。请勿包含引号或其他格式。在 API 中，将令牌指定为 `SecretString` 属性的值。

- Greengrass 组中引用 Secrets Manager 密钥的密钥资源。有关更多信息，请参阅将密钥部署到核心。

连接器参数

该连接器提供以下参数：

Version 4

`SplunkEndpoint`

Splunk 实例的终端节点。该值必须包含协议、主机名和端口。

在 AWS IoT 控制台中显示名称：Splunk 端点

必需：`true`

类型：`string`

有效模式：`^(http:\/\/|https:\/\/)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$`

`MemorySize`

要分配给此连接器的内存量（以 KB 为单位）。

AWS IoT 控制台中的显示名称：内存大小

必需：`true`

类型：`string`

有效模式：`^[0-9]+$`

`SplunkQueueSize`

提交或丢弃项目前可在内存中保存的最大项目数。达到该限制后，队列中最早的项目将被替换为较新项目。该限制通常适用于无 Internet 连接的情况。

AWS IoT 控制台中的显示名称：可保留的最大项目数

必需：`true`

类型：`string`

有效模式：`^[0-9]+$`

`SplunkFlushIntervalSeconds`

将收到的数据发布到 Splunk HEC 的时间间隔（以秒为单位）。最大值为 900。要将连接器配置为一收到项目便发布（而不进行批处理），请指定 0。

AWS IoT 控制台中的显示名称：Splunk 发布间隔

必需：`true`

类型：`string`

有效模式：`[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

AWS Secrets Manager 中存放 Splunk 令牌的密钥。这必须是文本类型密钥。

AWS IoT 控制台中的显示名称：Splunk 身份验证令牌密钥的 ARN

必需：`true`

类型：`string`

有效模式：`arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

Greengrass 组中引用 Splunk 密钥的密钥资源。

AWS IoT 控制台中的显示名称：Splunk 身份验证令牌资源

必需：`true`

类型：`string`

有效模式：`.+`

`SplunkCustomCALocation`

Splunk 的自定义证书颁发机构 (CA) 的文件路径（例如，`/etc/ssl/certs/splunk.crt`）。

AWS IoT 控制台中的显示名称：Splunk 自定义证书颁发机构位置

必需：`false`

类型：`string`

有效模式：`^$|/.*`

`IsolationMode`

此连接器的容器化模式。默认值为 `GreengrassContainer`，这意味着连接器在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

> **Note**
> 组的默认容器化设置不适用于连接器。

AWS IoT 控制台中的显示名称：容器隔离模式

必需：`false`

类型：`string`

有效值：`GreengrassContainer` 或 `NoContainer`

有效模式：`^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`SplunkEndpoint`

Splunk 实例的终端节点。该值必须包含协议、主机名和端口。

在 AWS IoT 控制台中显示名称：Splunk 端点

必需：`true`

类型：`string`

有效模式：`^(http:\/\/|https:\/\/)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$`

`MemorySize`

要分配给此连接器的内存量（以 KB 为单位）。

AWS IoT 控制台中的显示名称：内存大小

必需：`true`

类型：`string`

有效模式：`^[0-9]+$`

`SplunkQueueSize`

提交或丢弃项目前可在内存中保存的最大项目数。达到该限制后，队列中最早的项目将被替换为较新项目。该限制通常适用于无 Internet 连接的情况。

AWS IoT 控制台中的显示名称：可保留的最大项目数

必需：`true`

类型：`string`

有效模式：`^[0-9]+$`

`SplunkFlushIntervalSeconds`

将收到的数据发布到 Splunk HEC 的时间间隔（以秒为单位）。最大值为 900。要将连接器配置为一收到项目便发布（而不进行批处理），请指定 0。

AWS IoT 控制台中的显示名称：Splunk 发布间隔

必需：`true`

类型：`string`

有效模式：`[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

AWS Secrets Manager 中存放 Splunk 令牌的密钥。这必须是文本类型密钥。

AWS IoT 控制台中的显示名称：Splunk 身份验证令牌密钥的 ARN

必需：`true`

类型：`string`

有效模式：`arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

Greengrass 组中引用 Splunk 密钥的密钥资源。

AWS IoT 控制台中的显示名称：Splunk 身份验证令牌资源

必需：`true`

类型：`string`

有效模式：`.+`

`SplunkCustomCALocation`

Splunk 的自定义证书颁发机构 (CA) 的文件路径（例如，`/etc/ssl/certs/splunk.crt`）。

AWS IoT 控制台中的显示名称：Splunk 自定义证书颁发机构位置

必需：`false`

类型：`string`

有效模式：`^$|/.*`

创建连接器示例 (AWS CLI)

以下 CLI 命令创建一个 `ConnectorDefinition`，其初始版本包含 Splunk 集成连接器。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySplunkIntegrationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SplunkIntegration/
versions/4",
            "Parameters": {
                "SplunkEndpoint": "https://myinstance.cloud.splunk.com:8088",
                "MemorySize": 200000,
                "SplunkQueueSize": 10000,
                "SplunkFlushIntervalSeconds": 5,
                "SplunkTokenSecretArn":"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "SplunkTokenSecretArn-ResourceId": "MySplunkResource",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

> **Note**
>
> 此连接器中的 Lambda 函数的生命周期很长。

在 AWS IoT Greengrass 控制台中，您可以从组的连接器页面添加连接器。有关更多信息，请参阅the section called “连接器入门（控制台）”。

输入数据

此连接器接受关于 MQTT 主题的日志记录和事件数据，并按原样将收到的数据发布到 Splunk API。输入消息必须采用 JSON 格式。

订阅中的主题筛选条件

`splunk/logs/put`

消息属性

`request`

要发送到 Splunk API 的事件数据。事件必须满足 services/collector API 的规范。

必需：`true`

类型：`object`。只有 `event` 属性是必需的。

`id`

请求的任意 ID。该属性用于将输入请求映射到输出状态。

必需：`false`

类型：`string`

限制

使用该连接器时，Splunk API 施加的所有限制均适用。有关更多信息，请参阅 services/collector。

示例输入

```
{
    "request": {
        "event": "some event",
        "fields": {
            "severity": "INFO",
            "category": [
                "value1",
                "value2"
            ]
        }
    },
    "id": "request123"
}
```

输出数据

此连接器发布关于两个主题的输出数据：

- 有关 `splunk/logs/put/status` 主题的状态信息。
- 错误位于 `splunk/logs/put/error` 主题上。

主题筛选条件：`splunk/logs/put/status`

使用此主题侦听请求的状态。每当连接器向 Splunk API 发送一批收到的数据时，它就会发布一个包含成功请求和失败请求的 ID 列表。

输出示例

```
{
    "response": {
        "succeeded": [
            "request123",
            ...
        ],
        "failed": [
            "request789",
            ...
        ]
    }
}
```

主题筛选条件：`splunk/logs/put/error`

使用此主题侦听连接器中的错误。`error_message` 属性描述处理请求时遇到的错误或超时。

输出示例

```
{
    "response": {
        "error": "UnauthorizedException",
        "error_message": "invalid splunk token",
        "status": "fail"
    }
}
```

> **Note**
>
> 如果连接器检测到可重试的错误（如连接错误），它会在下一批次中重试发布。

用法示例

使用以下概括步骤设置可用于尝试连接器的示例 Python 3.7 Lambda 函数。

> **Note**
>
> - 如果要使用其他 Python 运行时，您可以创建从 Python3.x 到 Python 3.7 的符号链接。

- 连接器入门（控制台）和连接器入门 (CLI) 主题包含详细步骤，说明如何配置和部署示例 Twilio 通知连接器。

1. 确保满足连接器的要求。
2. 创建并发布将输入数据发送到连接器的 Lambda 函数。

   将示例代码保存为 PY 文件。下载并解压适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。然后，创建一个 zip 包，其中在根级别包含 PY 文件和 `greengrasssdk` 文件夹。此 zip 包是您上传到 AWS Lambda 的部署包。

   创建 Python 3.7 Lambda 函数后，请发布函数版本并创建别名。
3. 配置 Greengrass 组。

   a. 通过别名来添加 Lambda 函数（推荐）。将 Lambda 生命周期配置为长时间生存（或在 CLI 中设置为 `"Pinned": true`）。

   b. 添加所需的密钥资源并授予对 Lambda 函数的读取访问权限。

   c. 添加连接器并配置其参数。

   d. 添加允许连接器接收输入数据并针对支持的主题筛选条件发送输出数据的订阅。

      - 将 Lambda 函数设置为源，将连接器设置为目标，并使用支持的输入主题筛选条件。
      - 将连接器设置为源，将 AWS IoT Core 设置为目标，并使用支持的输出主题筛选条件。您可以使用此订阅查看 AWS IoT 控制台中的状态消息。
4. 部署组。
5. 在 AWS IoT 控制台中，在测试页面上，订阅输出数据主题以查看连接器中的状态消息。示例 Lambda 函数是长时间生存的，并且在部署组后立即开始发送消息。

   完成测试后，您可以将 Lambda 生命周期设置为按需（或在 CLI 中设置为 `"Pinned": false`）并部署组。这会阻止函数发送消息。

示例

以下示例 Lambda 函数向连接器发送一条输入消息。

```
import greengrasssdk
import time
import json
```

```
iot_client = greengrasssdk.client('iot-data')
send_topic = 'splunk/logs/put'

def create_request_with_all_fields():
    return {
        "request": {
            "event": "Access log test message."
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

许可证

该连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
4	增加了用于配置连接器容器化模式的 `IsolationMode` 参数。
3	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。
2	进行了修复，以减少过多的日志记录。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”

Twilio 通知连接器

> **Warning**
>
> 此连接器已进入生命周期延长阶段，AWS IoT Greengrass 不会发布更新来提供功能、现有功能增强、安全补丁或错误修复。有关更多信息，请参阅AWS IoT Greengrass Version 1 维护策略。

Twilio 通知连接器通过 Twilio 进行自动电话呼叫或发送文本消息。您可以使用该连接器来发送通知，以响应 Greengrass 组中的事件。对于电话呼叫，连接器可以将语音消息转发给收件人。

该连接器接收关于 MQTT 主题的 Twilio 消息信息，然后触发 Twilio 通知。

> **Note**
>
> 有关介绍如何使用 Twilio 通知连接器的教程，请参阅 the section called “连接器入门（控制台）” 或 the section called “连接器入门 (CLI)”。

此连接器具有以下版本。

版本	ARN
5	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/5`

版本	ARN
4	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/4`
3	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/3`
2	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/2`
1	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/1`

有关版本更改的信息，请参阅更改日志。

要求

此连接器具有以下要求：

Version 4 - 5

- AWS IoT Greengrass Core 软件 v1.9.3 或更高版本。AWS IoT Greengrass 必须配置为支持本地密钥，如密钥要求中所述。

  > **Note**
  >
  > 此要求包括允许访问您的 Secrets Manager 密钥。如果使用的是默认 Greengrass 服务角色，则 Greengrass 有权获得名称以 greengrass- 开头的密钥的值。

- Python 版本 3.7 或 3.8 已安装在核心设备上，并已添加到 PATH 环境变量中。

> **Note**
>
> 要使用 Python 3.8，请运行以下命令来创建从默认 Python 3.7 安装文件夹到已安装的 Python 3.8 二进制文件的符号链接。
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> 这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。

- Twilio 账户 SID、身份验证令牌和支持 Twilio 的电话号码。创建 Twilio 项目后，项目控制面板上会显示这些值。

> **Note**
>
> 您可以使用 Twilio 试用账户。如果您使用的是试用账户，则必须将非 Twilio 收件人的电话号码添加到经过验证的电话号码列表中。 有关更多信息，请参阅如何使用您的免费 Twilio 试用账户。

- AWS Secrets Manager 中的文本类型密钥，用于存储 Twilio 身份验证令牌。有关更多信息，请参阅 AWS Secrets Manager 用户指南中的创建基本密钥。

> **Note**
>
> 要在 Secrets Manager 控制台中创建密钥，请在明文选项卡上输入您的令牌。请勿包含引号或其他格式。在 API 中，将令牌指定为 `SecretString` 属性的值。

- Greengrass 组中引用 Secrets Manager 密钥的密钥资源。有关更多信息，请参阅将密钥部署到核心。

Versions 1 - 3

- AWS IoT Greengrass Core 软件 v1.7 或更高版本。AWS IoT Greengrass 必须配置为支持本地密钥，如密钥要求中所述。

> **Note**
>
> 此要求包括允许访问您的 Secrets Manager 密钥。如果使用的是默认 Greengrass 服务角色，则 Greengrass 有权获得名称以 greengrass- 开头的密钥的值。

- Python 版本 2.7 已安装在核心设备上，并已添加到 PATH 环境变量中。
- Twilio 账户 SID、身份验证令牌和支持 Twilio 的电话号码。创建 Twilio 项目后，项目控制面板上会显示这些值。

> **Note**
>
> 您可以使用 Twilio 试用账户。如果您使用的是试用账户，则必须将非 Twilio 收件人的电话号码添加到经过验证的电话号码列表中。 有关更多信息，请参阅如何使用您的免费 Twilio 试用账户。

- AWS Secrets Manager 中的文本类型密钥，用于存储 Twilio 身份验证令牌。有关更多信息，请参阅 AWS Secrets Manager 用户指南中的创建基本密钥。

> **Note**
>
> 要在 Secrets Manager 控制台中创建密钥，请在明文选项卡上输入您的令牌。请勿包含引号或其他格式。在 API 中，将令牌指定为 `SecretString` 属性的值。

- Greengrass 组中引用 Secrets Manager 密钥的密钥资源。有关更多信息，请参阅将密钥部署到核心。

连接器参数

该连接器提供以下参数。

Version 5

`TWILIO_ACCOUNT_SID`

Twilio 账户 SID，用于调用 Twilio API。

AWS IoT 控制台中的显示名称：Twilio 账户 SID

必需：`true`

类型：string

有效模式：.+

TwilioAuthTokenSecretArn

存储 Twilio 身份验证令牌的 Secrets Manager 密钥的 ARN。

> **Note**
> 这用于访问核心上的本地密钥的值。

AWS IoT 控制台中的显示名称：Twilio 身份验证令牌密钥的 ARN

必需：true

类型：string

有效模式：arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+

TwilioAuthTokenSecretArn-ResourceId

Greengrass 组中密钥资源的 ID，该资源引用 Twilio 身份验证令牌的密钥。

AWS IoT 控制台中的显示名称：Twilio 身份验证令牌资源

必需：true

类型：string

有效模式：.+

DefaultFromPhoneNumber

支持 Twilio 的默认电话号码，Twilio 用该电话号码发送消息。Twilio 用该号码来启动文本或呼叫。

- 如果不配置默认电话号码，必须在输入消息正文的 from_number 属性中指定电话号码。
- 如果配置默认电话号码，则可视情况通过指定输入消息正文中的 from_number 属性来覆盖默认值。

AWS IoT 控制台中的显示姓名：默认为来自电话号码

必需：`false`

类型：`string`

有效模式：`^$|\+[0-9]+`

`IsolationMode`

此连接器的容器化模式。默认值为 `GreengrassContainer`，这意味着连接器在 AWS IoT Greengrass 容器内的隔离运行时环境中运行。

> **Note**
> 组的默认容器化设置不适用于连接器。

AWS IoT 控制台中的显示名称：容器隔离模式

必需：`false`

类型：`string`

有效值：`GreengrassContainer` 或 `NoContainer`

有效模式：`^NoContainer$|^GreengrassContainer$`

Version 1 - 4

`TWILIO_ACCOUNT_SID`

Twilio 账户 SID，用于调用 Twilio API。

AWS IoT 控制台中的显示名称：Twilio 账户 SID

必需：`true`

类型：`string`

有效模式：`.+`

`TwilioAuthTokenSecretArn`

存储 Twilio 身份验证令牌的 Secrets Manager 密钥的 ARN。

> **Note**
>
> 这用于访问核心上的本地密钥的值。

AWS IoT 控制台中的显示名称：Twilio 身份验证令牌密钥的 ARN

必需：`true`

类型：`string`

有效模式：`arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`TwilioAuthTokenSecretArn-ResourceId`

Greengrass 组中密钥资源的 ID，该资源引用 Twilio 身份验证令牌的密钥。

AWS IoT 控制台中的显示名称：Twilio 身份验证令牌资源

必需：`true`

类型：`string`

有效模式：`.+`

`DefaultFromPhoneNumber`

支持 Twilio 的默认电话号码，Twilio 用该电话号码发送消息。Twilio 用该号码来启动文本或呼叫。

- 如果不配置默认电话号码，必须在输入消息正文的 `from_number` 属性中指定电话号码。
- 如果配置默认电话号码，则可视情况通过指定输入消息正文中的 `from_number` 属性来覆盖默认值。

AWS IoT 控制台中的显示姓名：默认为来自电话号码

必需：`false`

类型：`string`

有效模式：`^$|\+[0-9]+`

创建连接器示例 (AWS CLI)

以下示例 CLI 命令创建一个 `ConnectorDefinition`，它具有包含 Twilio 通知连接器的初始版本。

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyTwilioNotificationsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/5",
            "Parameters": {
                "TWILIO_ACCOUNT_SID": "abcd12345xyz",
                "TwilioAuthTokenSecretArn": "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "TwilioAuthTokenSecretArn-ResourceId": "MyTwilioSecret",
                "DefaultFromPhoneNumber": "+19999999999",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

有关介绍如何将 Twilio 通知连接器添加到组的教程，请参阅 the section called “连接器入门 (CLI)” 和 the section called “连接器入门（控制台）”。

输入数据

此连接器接受关于两个 MQTT 主题的 Twilio 消息信息。输入消息必须采用 JSON 格式。

- 关于 `twilio/txt` 主题的文本消息信息。
- 关于 `twilio/call` 主题的电话消息信息。

> **Note**
>
> 输入消息负载可包含文本消息 (`message`) 或语音消息 (`voice_message_location`)，但不能同时包含二者。

主题筛选条件：**twilio/txt**

消息属性

request

有关 Twilio 通知的信息。

必需：true

类型：包含以下属性的 object：

recipient

消息收件人。只支持一个收件人。

必需：true

类型：包含以下属性的 object：

name

收件人的姓名。

必需：true

类型：string

有效模式：.*

phone_number

收件人的电话号码。

必需：true

类型：string

有效模式：\+[1-9]+

message

文本消息的文本内容。本主题仅支持文本消息。对于语音消息，请使用 twilio/call。

必需：true

类型：string

有效模式：.+

from_number

发件人的 电话号码。Twilio 用该电话号码来启动消息。只有在未配置 DefaultFromPhoneNumber 参数时才需要该属性。如果已配置 DefaultFromPhoneNumber，则可使用该属性来覆盖默认值。

必需：false

类型：string

有效模式：\+[1-9]+

retries

重试次数。默认值为 0。

必需：false

类型：integer

id

请求的任意 ID。此属性用于将输入请求映射到输出响应。

必需：true

类型：string

有效模式：.+

示例输入

```
{
    "request": {
        "recipient": {
            "name": "Darla",
            "phone_number": "+12345000000",
            "message": "Hello from the edge"
        },
        "from_number": "+19999999999",
        "retries": 3
    },
    "id": "request123"
}
```

主题筛选条件：**twilio/call**

消息属性

request

有关 Twilio 通知的信息。

必需：true

类型：包含以下属性的 object：

recipient

消息收件人。只支持一个收件人。

必需：true

类型：包含以下属性的 object：

name

收件人的姓名。

必需：true

类型：string

有效模式：.+

phone_number

收件人的电话号码。

必需：true

类型：string

有效模式：\+[1-9]+

voice_message_location

语音消息的音频内容的 URL。必须采用 TwiML 格式。本主题仅支持语音消息。对于文本消息，请使用 twilio/txt。

必需：true

类型：string

有效模式：.+

from_number

发件人的 电话号码。Twilio 用该电话号码来启动消息。只有在未配置 DefaultFromPhoneNumber 参数时才需要该属性。如果已配置 DefaultFromPhoneNumber，则可使用该属性来覆盖默认值。

必需：false

类型：string

有效模式：\+[1-9]+

retries

重试次数。默认值为 0。

必需：false

类型：integer

id

请求的任意 ID。此属性用于将输入请求映射到输出响应。

必需：true

类型：string

有效模式：.+

示例输入

```
{
    "request": {
        "recipient": {
            "name": "Darla",
            "phone_number": "+12345000000",
            "voice_message_location": "https://some-public-TwiML"
        },
        "from_number": "+19999999999",
        "retries": 3
    },
    "id": "request123"
}
```

输出数据

此连接器将状态信息发布为 MQTT 主题的输出数据。

订阅中的主题筛选条件

```
twilio/message/status
```

示例输出：成功

```
{
    "response": {
        "status": "success",
        "payload": {
            "from_number": "+19999999999",
            "messages": {
                "message_status": "queued",
                "to_number": "+12345000000",
                "name": "Darla"
            }
        },
    "id": "request123"
}
```

示例输出：失败

```
{
    "response": {
        "status": "fail",
        "error_message": "Recipient name cannot be None",
        "error": "InvalidParameter",
        "payload": None
        }
    },
    "id": "request123"
}
```

输出中的 payload 属性是发送消息时来自 Twilio API 的响应。如果连接器检测到输入数据无效（例如，它不指定必填的输入字段），则连接器会返回错误并将值设置为 None。以下为示例负载：

```
{
```

```
    'from_number':'+19999999999',
    'messages': {
        'name':'Darla',
        'to_number':'+12345000000',
        'message_status':'undelivered'
    }
}
```

```
{
    'from_number':'+19999999999',
    'messages': {
        'name':'Darla',
        'to_number':'+12345000000',
        'message_status':'queued'
    }
}
```

用法示例

使用以下概括步骤设置可用于尝试连接器的示例 Python 3.7 Lambda 函数。

> **Note**
>
> the section called “连接器入门（控制台）”和the section called “连接器入门 (CLI)” 主题包含端到端步骤，说明如何设置、部署和测试 Twilio 通知连接器。

1. 确保满足连接器的要求。
2. 创建并发布将输入数据发送到连接器的 Lambda 函数。

   将示例代码保存为 PY 文件。下载并解压适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包。然后，创建一个 zip 包，其中在根级别包含 PY 文件和 `greengrasssdk` 文件夹。此 zip 包是您上传到 AWS Lambda 的部署包。

   创建 Python 3.7 Lambda 函数后，请发布函数版本并创建别名。
3. 配置 Greengrass 组。

   a. 通过别名来添加 Lambda 函数（推荐）。将 Lambda 生命周期配置为长时间生存（或在 CLI 中设置为 `"Pinned": true`）。

b. 添加所需的密钥资源并授予对 Lambda 函数的读取访问权限。

c. 添加连接器并配置其参数。

d. 添加允许连接器接收输入数据并针对支持的主题筛选条件发送输出数据的订阅。

- 将 Lambda 函数设置为源，将连接器设置为目标，并使用支持的输入主题筛选条件。
- 将连接器设置为源，将 AWS IoT Core 设置为目标，并使用支持的输出主题筛选条件。您可以使用此订阅查看 AWS IoT 控制台中的状态消息。

4. 部署组。

5. 在 AWS IoT 控制台中的测试页面上，订阅输出数据主题以查看连接器中的状态消息。示例 Lambda 函数是长时间生存的，并且在部署组后立即开始发送消息。

   完成测试后，您可以将 Lambda 生命周期设置为按需（或在 CLI 中设置为 `"Pinned": false`）并部署组。这会阻止函数发送消息。

示例

以下示例 Lambda 函数向连接器发送一条输入消息。此示例触发文本消息。

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
TXT_INPUT_TOPIC = 'twilio/txt'
CALL_INPUT_TOPIC = 'twilio/call'

def publish_basic_message():

    txt = {
        "request": {
            "recipient" : {
                "name": "Darla",
                "phone_number": "+12345000000",
                "message": 'Hello from the edge'
            },
            "from_number" : "+19999999999"
        },
        "id" : "request123"
    }

    print("Message To Publish: ", txt)
```

```
    client.publish(topic=TXT_INPUT_TOPIC,
                   payload=json.dumps(txt))

publish_basic_message()

def lambda_handler(event, context):
    return
```

许可证

Twilio 通知连接器包含以下第三方软件/许可：

- twilio-python/MIT

此连接器在 Greengrass Core 软件许可协议下发布。

更改日志

下表介绍每个版本连接器的更改。

版本	更改
5	增加了用于配置连接器容器化模式的 `IsolationMode` 参数。
4	已将 Lambda 运行时升级到 Python 3.7，这会更改运行时要求。
3	进行了修复，以减少过多的日志记录。
2	少量错误修复和改进。
1	首次发布。

Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

另请参阅

- 使用连接器与服务和协议集成
- the section called “连接器入门（控制台）”
- the section called “连接器入门 (CLI)”
- Twilio API 参考

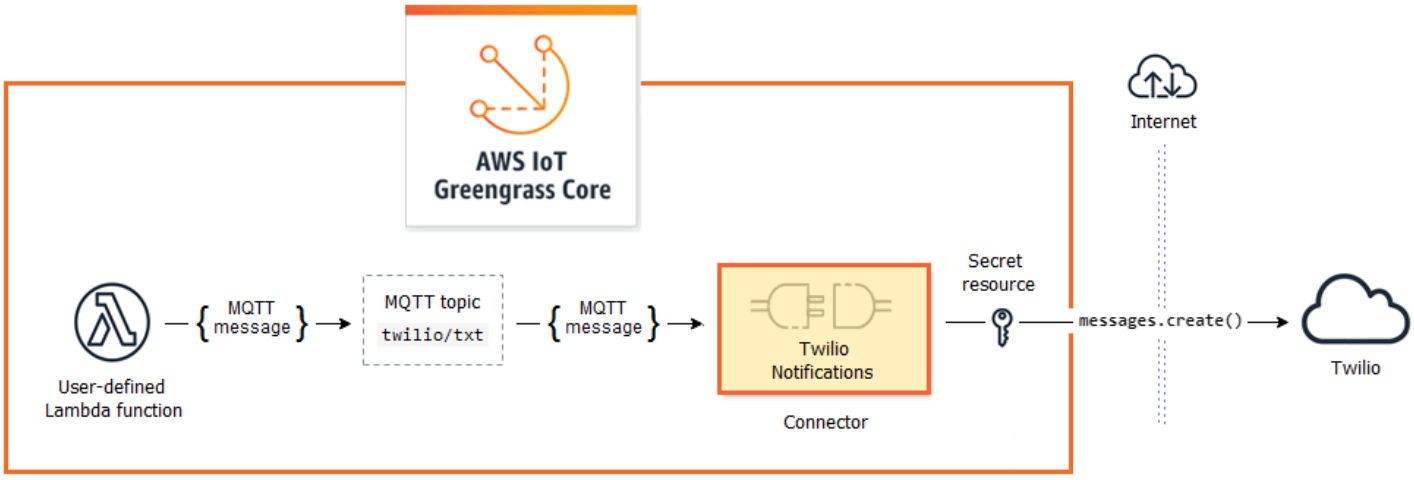
Greengrass 连接器入门（控制台）

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

本教程介绍如何通过 AWS Management Console 使用连接器。

使用连接器可加快开发生命周期。连接器是预先构建、可重复使用的模块，可以更加轻松地与服务、协议和资源进行交互。它们可以帮助您将业务逻辑更快地部署到 Greengrass 设备。有关更多信息，请参阅使用连接器与服务和协议集成。

在本教程中，您将配置和部署 Twilio 通知连接器。连接器接收 Twilio 消息信息作为输入数据，然后触发 Twilio 文本消息。下图中将显示数据流。



配置连接器后，您便可创建 Lambda 函数和订阅。

- 该函数将评估来自温度传感器的模拟数据。它有条件地将 Twilio 消息信息发布到 MQTT 主题。这是连接器订阅的主题。
- 该订阅允许函数发布到主题，并允许连接器接收来自该主题的数据。

Twilio 通知连接器需要 Twilio 身份验证令牌，才能与 Twilio API 进行交互。令牌是在 AWS Secrets Manager 中创建且从组资源中引用的文本类型密钥。这样一来，AWS IoT Greengrass 便可在 Greengrass 核心上创建密钥的本地副本，在该核心中，此类副本经过加密且可供连接器使用。有关更多信息，请参阅将密钥部署到核心。

本教程包含以下概括步骤：

1. 创建 Secrets Manager 密钥
2. 将密钥资源添加到组
3. 将连接器添加到组
4. 创建 Lambda 函数部署程序包
5. 创建 Lambda 函数
6. 将函数添加到组
7. 将订阅添加到组
8. 部署组
9. the section called “测试解决方案”

完成本教程大约需要 20 分钟。

先决条件

要完成此教程，需要：

- Greengrass 组和 Greengrass Core（v1.9.3 或更高版本）。要了解如何创建 Greengrass 组和核心，请参阅 入门 AWS IoT Greengrass。“入门”教程还包含用于安装 AWS IoT Greengrass Core 软件的步骤。
- 安装在 AWS IoT Greengrass 核心设备上的 Python 3.7。
- AWS IoT Greengrass 必须配置为支持本地密钥，如密钥要求中所述。

> **Note**
>
> 此要求包括允许访问您的 Secrets Manager 密钥。如果使用的是默认 Greengrass 服务角色，则 Greengrass 有权获得名称以 greengrass- 开头的密钥的值。

- Twilio 账户 SID、身份验证令牌和支持 Twilio 的电话号码。创建 Twilio 项目后，项目控制面板上会显示这些值。

> **Note**
>
> 您可以使用 Twilio 试用账户。如果您使用的是试用账号，则必须将非 Twilio 收件人的电话号码添加到经过验证的电话号码列表中。 有关更多信息，请参阅如何使用您的免费 Twilio 试用账户。

步骤 1：创建 Secrets Manager 密钥

在此步骤中，您将使用 AWS Secrets Manager 控制台为 Twilio 身份验证令牌创建一个文本类型密钥。

1. 登录到 AWS Secrets Manager 控制台。

   > **Note**
   >
   > 有关此过程的更多信息，请参阅AWS Secrets Manager 用户指南中的步骤 1：在 AWS Secrets Manager 中创建和存储密钥。

2. 选择 Store a new secret (存储新密钥)。
3. 在 选择密钥类型 下，选择 其他密钥类型。
4. 在指定要为此密钥存储的键/值对下面的纯文本选项卡上，输入您的 Twilio 身份验证令牌。删除所有 JSON 格式设置，然后仅输入令牌值。
5. 为加密密钥保留选中 aws/secretsmanager，然后选择下一步。

   > **Note**
   >
   > 如果使用 Secrets Manager 在您的账户中创建的默认 AWS 托管密钥，AWS KMS 不会对您收费。

6. 对于密钥名称，输入 **`greengrass-TwilioAuthToken`**，然后选择下一步。

   > **Note**
   >
   > 默认情况下，Greengrass 服务角色允许 AWS IoT Greengrass 获取名称以 greengrass- 开头的密钥的值。有关更多信息，请参阅密钥要求。

7. 本教程不需要轮换，因此，请选择“禁用自动轮换”，然后选择下一步。

8. 在 Review (审核) 页上，审核您的设置，然后选择 Store (存储)。

   接下来，在 Greengrass 组中创建一个引用该密钥的密钥资源。

步骤 2：将密钥资源添加到 Greengrass 组

在此步骤中，您将一个密钥资源 添加到 Greengrass 组。该资源是对上一步中创建的密钥的引用。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择要将密钥资源添加到的组。
3. 在组配置页面上，选择资源选项卡，然后向下滚动到密钥部分。密钥部分显示从属于该组的密钥资源。您可以从此部分添加、编辑和删除密钥资源。

   > **Note**
   >
   > 或者，控制台允许您在配置连接器或 Lambda 函数时创建密钥和密钥资源。您可以从连接器的配置参数页面或 Lambda 函数的资源页面执行此操作。

4. 在密钥部分下选择添加。
5. 在添加密钥资源页面，在资源名称中输入 **`MyTwilioAuthToken`**。
6. 对于密钥，选择 greengrass-TwilioAuthToken。
7. 在选择标签（可选）部分中，AWSCURRENT 暂存标签表示密钥的最新版本。该标签始终包含在密钥资源中。

   > **Note**
   >
   > 本教程只需要 AWSCURRENT 标签。您可以视情况包括 Lambda 函数或连接器所需的标签。

8. 选择 Add resource (添加资源)。

步骤 3：将连接器添加到 Greengrass 组

在此步骤中，您将为 Twilio 通知连接器配置参数，并将其添加到组。

1. 在组配置页面上，选择 Connectors (连接器)，然后选择 Add a connector (添加连接器)。
2. 在添加连接器页面上，选择 Twilio Notifications。

3. 选择 版本。
4. 在配置部分中：
   - 对于 Twilio 身份验证令牌资源，输入上一步中创建的资源。

     > **Note**
     > 输入该资源时，将为您填充 Twilio 身份验证令牌密钥的 ARN 属性。

   - 对于 Default from phone number (默认的来电号码)，输入您的支持 Twilio 的电话号码。
   - 对于 Twilio account SID (Twilio 账户 SID)，输入您的 Twilio 账户 SID。
5. 选择 Add resource (添加资源)。

步骤 4：创建 Lambda 函数部署程序包

要创建 Lambda 函数，您必须先创建一个包含函数代码和依赖项的 Lambda 函数部署包。Greengrass Lambda 函数需要 AWS IoT Greengrass Core 软件开发工具包来执行各项任务，例如在核心环境中与 MQTT 消息通信和访问本地机密等。本教程将创建一个 Python 函数，因此您需要在部署包中使用 Python 版本的软件开发工具包。

1. 从 AWS IoT Greengrass Core 软件开发工具包下载页面，将适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包下载到您的计算机上。
2. 解压缩下载的程序包以获取软件开发工具包。软件开发工具包是 `greengrasssdk` 文件夹。
3. 将以下 Python 代码函数保存在名为 `temp_monitor.py` 的本地文件中。

```
import greengrasssdk
import json
import random

client = greengrasssdk.client('iot-data')

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
```

```
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. 将以下项目压缩到名为 `temp_monitor_python.zip` 的文件中。在创建 ZIP 文件时，仅包含代码和依赖项，而不包含文件夹。

   - temp_monitor.py。应用程序逻辑。
   - greengrasssdk。发布 MQTT 消息的 Python Greengrass Lambda 函数所需的库。

   此即 Lambda 函数部署程序包。

现在，创建一个使用部署程序包的 Lambda 函数。

步骤 5：在 AWS Lambda 控制台创建一个 Lambda 函数

在该步骤中，您使用 AWS Lambda 控制台创建 Lambda 函数，然后将其配置为使用您的部署包。接着，发布函数版本并创建别名。

1. 首先，创建 Lambda 函数。

   a. 在 AWS Management Console中，选择 Services (服务)，然后打开 AWS Lambda 控制台。

b. 选择 创建函数，然后选择 从头开始创作。

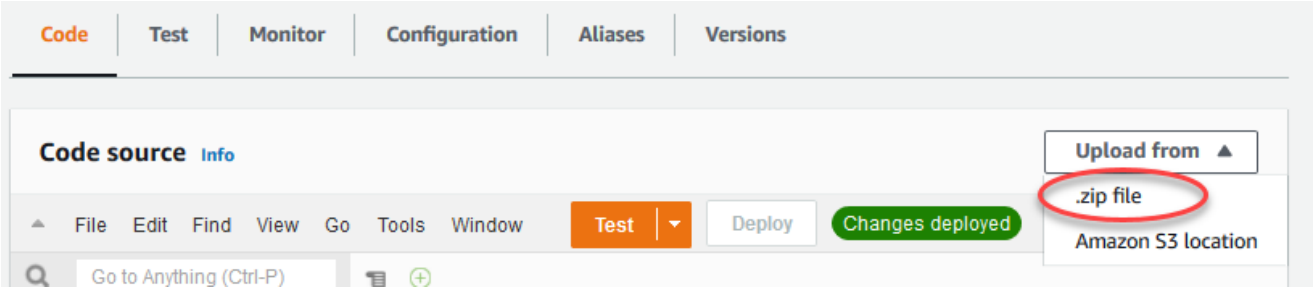
c. 在 Basic information (基本信息) 部分中，使用以下值：

- 对于 Function name（函数名称），请输入 **`TempMonitor`**。
- 对于 Runtime (运行时)，选择 Python 3.7。
- 对于权限，请保留默认设置。这将创建一个授予基本 Lambda 权限的执行角色。此角色未由 AWS IoT Greengrass 使用。

d. 在页面底部，选择创建函数。

2. 接下来，注册处理程序并上传您的 Lambda 函数部署程序包。

a. 在代码选项卡上的代码源下，选择上传自。从下拉列表中选择 .zip 文件。



b. 选择上传，然后选择您的 `temp_monitor_python.zip` 部署包。然后，选择 Save (保存)。

c. 在函数的代码选项卡中，在运行时设置下选择编辑，然后输入以下值。

- 对于 Runtime (运行时)，选择 Python 3.7。
- 对于 Handler (处理程序)，输入 **`temp_monitor.function_handler`**。

d. 选择 Save（保存）。

> **Note**
>
> AWS Lambda 控制台上的测试按钮不适用于此功能。AWS IoT Greengrass Core 软件开发工具包不包含在 AWS Lambda 控制台中独立运行 Greengrass Lambda 函数所需的模块。这些模块（例如 `greengrass_common`）是在函数部署到您的 Greengrass 核心之后提供给它们的。

3. 现在，发布 Lambda 函数的第一个版本并创建此版本的别名。

> **Note**
>
> Greengrass 组可以按别名（推荐）或版本引用 Lambda 函数。使用别名，您可以更轻松地管理代码更新，因为您在更新函数代码时，不必更改订阅表或组定义。相反，您只需将别名指向新的函数版本。

a. 在 Actions 菜单上，选择 Publish new version。

b. 对于 Version description (版本描述)，输入 **`First version`**，然后选择 Publish (发布)。

c. 在 TempMonitor: 1 配置页面上，从 Actions (操作) 菜单中选择 Create alias (创建别名)。

d. 在创建新别名页面上，使用以下值：

- 对于 Name（名称），请输入 **`GG_TempMonitor`**。
- 对于 Version (版本)，选择 1。

> **Note**
>
> AWS IoT Greengrass 不支持 $LATEST 版本的 Lambda 别名。

e. 选择 Create（创建）。

现在，您已准备就绪，可以将 Lambda 函数添加到 Greengrass 组。

步骤 6：将 Lambda 函数添加到 Greengrass 组

在该步骤中，您将 Lambda 函数添加到该组，然后配置其生命周期和环境变量。有关更多信息，请参阅the section called “控制 Greengrass Lambda 函数执行”。

1. 在组配置页面上，选择Lambda 函数选项卡。
2. 在我的 Lambda 函数下，选择添加。
3. 在添加 Lambda 函数页面上，为您的 Lambda 函数选择 TempMonitor。
4. 对于 Lambda 函数版本，请选择 Alias: GG_TempMonitor。
5. 选择 添加 Lambda 函数。

步骤 7：将订阅添加到 Greengrass 组

在该步骤中，您将添加一个订阅，使 Lambda 函数将输入数据发送到连接器。此连接器定义它订阅的 MQTT 主题，因此该订阅使用其中一个主题。这与示例函数发布到的主题相同。

对于本教程，您还可以创建订阅，以允许函数从 AWS IoT 接收模拟温度读数，并允许 AWS IoT 从连接器接收状态信息。

1. 在组配置页面中，选择订阅选项卡，然后选择添加订阅。
2. 在创建订阅页面中，按如下所述配置源和目标：
   a. 在源类型中，选择 Lambda 函数，然后选择 TempMonitor。
   b. 对于目标类型，选择连接器，然后选择 Twilio Notifications。
3. 对于主题筛选条件，输入 **`twilio/txt`**。
4. 选择 Create subscription（创建订阅）。
5. 重复步骤 1 至 4，以创建允许 AWS IoT 将消息发布到该函数的订阅。
   a. 在源类型中，选择服务，然后选择 IoT 云。
   b. 对于选择目标，选择 Lambda 函数，然后选择 TempMonitor。
   c. 对于 Topic filter (主题筛选条件)，输入 **`temperature/input`**。
6. 重复步骤 1 至 4，以创建允许连接器将消息发布到 AWS IoT 的订阅。
   a. 对于源类型，选择连接器，然后选择 Twilio Notifications。
   b. 对于目标类型，选择服务，然后选择 IoT 云。
   c. 对于主题筛选条件，已为您输入 **`twilio/message/status`**。这是连接器发布到的预定义主题。

步骤 8：部署 Greengrass 组

将组部署到核心设备。

1. 确保 AWS IoT Greengrass 核心正在运行。根据需要在您的 Raspberry Pi 终端中运行以下命令。
   a. 要检查守护程序是否正在运行，请执行以下操作：

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

如果输出包含 root 的 /greengrass/ggc/packages/*ggc-version*/bin/daemon 条目，则表示守护程序正在运行。

> **Note**
>
> 路径中的版本取决于您的核心设备上安装的 AWS IoT Greengrass 核心软件版本。

b. 启动进程守护程序：

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

2. 在组配置页面上，选择部署。
3. a. 在 Lambda 函数选项卡的系统 Lambda 函数部分下，选择 IP 检测器，再选择编辑。

   b. 在编辑 IP 检测器设置对话框中，选择自动检测和覆盖 MQTT 代理端点。

   c. 选择 Save（保存）。

   这使得设备可以自动获取核心的连接信息，例如 IP 地址、DNS 和端口号。建议使用自动检测，不过 AWS IoT Greengrass 也支持手动指定的终端节点。只有在首次部署组时，系统才会提示您选择发现方法。

   > **Note**
   >
   > 如果出现提示，请授予权限，以创建 Greengrass 服务角色并将其关联至当前 AWS 区域中的 AWS 账户。该角色将允许 AWS IoT Greengrass 访问您的 AWS 服务资源。

   Deployments (部署) 页面显示了部署时间戳、版本 ID 和状态。完成后，部署的状态应显示为已完成。

   有关问题排查帮助，请参阅排查问题。

> **Note**
>
> Greengrass 组在一个时间上只能包含一个版本的连接器。有关升级连接器版本的信息，请参阅the section called “升级连接器版本”。

测试解决方案

1. 在 AWS IoT 控制台主页上，选择测试。
2. 对于订阅主题，请使用以下值，然后选择订阅。Twilio 通知连接器向此主题发布状态信息。

属性	Value
订阅主题	twilio/message/status
MQTT 负载显示	将负载显示为字符串

3. 对于发布到主题，请使用以下值，然后选择发布 来调用函数。

属性	Value
主题	temperature/input
消息	将 *recipient-name* 替换为文本消息收件人的姓名，将 *recipient-phone-number* 替换为文本消息收件人的电话号码。示例：+12345000000 `{ "to_name": " recipient-name ", "to_number": " recipient-phone-number ", "temperature": 31 }` 如果您使用的是试用账号，则必须将非 Twilio 收件人的电话号码添加到经过验证的电话号码

属性	Value
	列表中。有关更多信息，请参阅验证您的个人电话号码。

如果成功，收件人将收到短信，控制台将在输出数据中显示 `success` 状态。

现在，将输入消息中的 `temperature` 更改为 **29** 并发布。由于温度小于 30，因此 TempMonitor 函数不会触发 Twilio 消息。

另请参阅

- 使用连接器与服务和协议集成
- the section called “AWS 提供的 Greengrass 连接器”

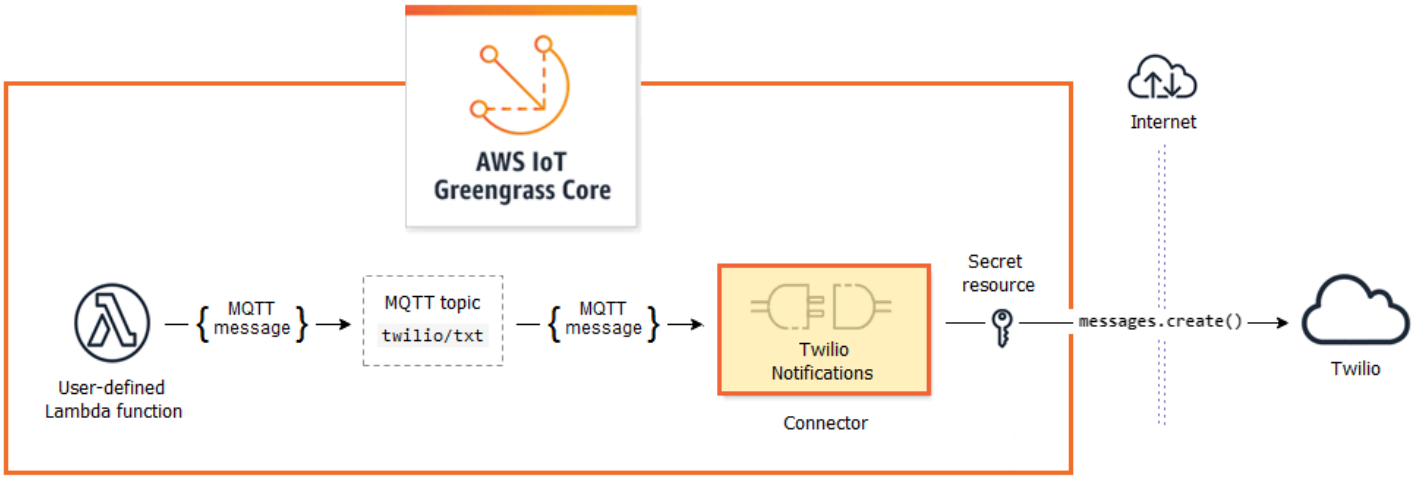
Greengrass 连接器入门 (CLI)

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

本教程介绍如何通过 AWS CLI 使用连接器。

使用连接器可加快开发生命周期。连接器是预先构建、可重复使用的模块，可以更加轻松地与服务、协议和资源进行交互。它们可以帮助您将业务逻辑更快地部署到 Greengrass 设备。有关更多信息，请参阅使用连接器与服务和协议集成。

在本教程中，您将配置和部署 Twilio 通知连接器。连接器接收 Twilio 消息信息作为输入数据，然后触发 Twilio 文本消息。下图中将显示数据流。



配置连接器后，您便可创建 Lambda 函数和订阅。

- 该函数将评估来自温度传感器的模拟数据。它有条件地将 Twilio 消息信息发布到 MQTT 主题。这是连接器订阅的主题。
- 该订阅允许函数发布到主题，并允许连接器接收来自该主题的数据。

Twilio 通知连接器需要 Twilio 身份验证令牌，才能与 Twilio API 进行交互。令牌是在 AWS Secrets Manager 中创建且从组资源中引用的文本类型密钥。这样一来，AWS IoT Greengrass 便可在 Greengrass 核心上创建密钥的本地副本，在该核心中，此类副本经过加密且可供连接器使用。有关更多信息，请参阅将密钥部署到核心。

本教程包含以下概括步骤：

1. 创建 Secrets Manager 密钥
2. 创建资源定义和版本
3. 创建连接器定义和版本
4. 创建 Lambda 函数部署程序包
5. 创建 Lambda 函数
6. 创建函数定义和版本
7. 创建订阅定义和版本
8. 创建组版本
9. 创建部署

10the section called “测试解决方案”

完成本教程大约需要 30 分钟。

使用 AWS IoT Greengrass API

使用 Greengrass 组及组组件（例如，组中的连接器、函数和资源）时，了解以下模式会很有帮助。

- 在层次结构顶部，组件有一个 定义 对象，该对象是版本 对象的容器。反过来，版本又是实际连接器、函数或其他组件类型的容器。
- 当您部署到 Greengrass 核心时，将部署一个特定的组版本。组版本可包含每种类型的组件的一个版本。核心是必需的，而其他组件可根据需要提供。
- 版本是不可变的，因此当您要进行更改时必须创建新版本。

> **Tip**
>
> 如果您在运行 AWS CLI 命令时收到错误，请添加 `--debug` 参数，然后重新运行该命令以获取有关错误的更多信息。

利用 AWS IoT Greengrass API，您可以为某一组件类型创建多个定义。例如，您可以在每次创建 `FunctionDefinition` 时都创建一个 `FunctionDefinitionVersion` 对象，也可以向现有定义添加新版本。这种灵活性使您能够自定义版本管理系统。

先决条件

要完成此教程，需要：

- Greengrass 组和 Greengrass Core（v1.9.3 或更高版本）。要了解如何创建 Greengrass 组和核心，请参阅 入门 AWS IoT Greengrass。“入门”教程还包含用于安装 AWS IoT Greengrass Core 软件的步骤。
- 安装在 AWS IoT Greengrass 核心设备上的 Python 3.7。
- AWS IoT Greengrass 必须配置为支持本地密钥，如密钥要求中所述。

> **Note**
>
> 此要求包括允许访问您的 Secrets Manager 密钥。如果使用的是默认 Greengrass 服务角色，则 Greengrass 有权获得名称以 greengrass- 开头的密钥的值。

- Twilio 账户 SID、身份验证令牌和支持 Twilio 的电话号码。创建 Twilio 项目后，项目控制面板上会显示这些值。

> **Note**
>
> 您可以使用 Twilio 试用账户。如果您使用的是试用账号，则必须将非 Twilio 收件人的电话号码添加到经过验证的电话号码列表中。 有关更多信息，请参阅如何使用您的免费 Twilio 试用账户。

- 在您的计算机上安装并配置 AWS CLI。有关更多信息，请参阅 AWS Command Line Interface 用户指南中的安装 AWS Command Line Interface 和配置 AWS CLI。

本教程中的示例是针对 Linux 及其他基于 Unix 的系统编写的。如果您使用的是 Windows，请参阅为 AWS Command Line Interface 指定参数值以了解语法上的差异。

如果命令包含 JSON 字符串，本教程提供了在单行包含 JSON 的示例。在某些系统上，使用此格式可能会更容易地编辑和运行命令。

步骤 1：创建 Secrets Manager 密钥

在此步骤中，您将使用 AWS Secrets Manager API 为 Twilio 身份验证令牌创建一个密钥。

1. 首先，创建该密钥。

   - 使用您的 Twilio 身份验证令牌替换 *twilio-auth-token*。

```
aws secretsmanager create-secret --name greengrass-TwilioAuthToken --secret-
string twilio-auth-token
```

> **Note**
>
> 默认情况下，Greengrass 服务角色允许 AWS IoT Greengrass 获取名称以 greengrass- 开头的密钥的值。有关更多信息，请参阅密钥要求。

2. 从输出中复制该密钥的 ARN。您将使用此密钥创建并配置 Twilio 通知连接器。

步骤 2：创建资源定义和版本

在此步骤中，您将使用 AWS IoT Greengrass API 为 Secrets Manager 密钥创建密钥资源。

1. 创建一个包含初始版本的资源定义。

   - 将 *secret-arn* 替换为您在上一步中复制的密钥的 ARN。

   JSON Expanded

   ```
   aws greengrass create-resource-definition --name MyGreengrassResources --
   initial-version '{
       "Resources": [
           {
               "Id": "TwilioAuthToken",
               "Name": "MyTwilioAuthToken",
               "ResourceDataContainer": {
                   "SecretsManagerSecretResourceData": {
                       "ARN": "secret-arn"
                   }
               }
           }
       ]
   }'
   ```

   JSON Single-line

   ```
   aws greengrass create-resource-definition \
   --name MyGreengrassResources \
   ```

```
--initial-version '{"Resources": [{"Id": "TwilioAuthToken",
 "Name": "MyTwilioAuthToken", "ResourceDataContainer":
 {"SecretsManagerSecretResourceData": {"ARN": "secret-arn"}}}]}'
```

2. 从输出中复制该资源定义的 `LatestVersionArn`。您将使用此值向部署到核心的组版本添加资源定义版本。

步骤 3：创建连接器定义和版本

在此步骤中，您将为 Twilio 通知连接器配置参数。

1. 创建包含初始版本的连接器定义。

   - 用您的 Twilio 账户 SID 替换 *account-sid*。
   - 将 *secret-arn* 替换为您的 Secrets Manager 密钥的 ARN。连接器使用此参数获取本地密钥的值。
   - 使用您的支持 Twilio 的电话号码替换 *phone-number*。Twilio 使用此参数来启动文本消息。可在输入消息负载中覆盖此参数。采用以下格式：+19999999999

   JSON Expanded

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --
initial-version '{
    "Connectors": [
        {
            "Id": "MyTwilioNotificationsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
            "Parameters": {
                "TWILIO_ACCOUNT_SID": "account-sid",
                "TwilioAuthTokenSecretArn": "secret-arn",
                "TwilioAuthTokenSecretArn-ResourceId": "TwilioAuthToken",
                "DefaultFromPhoneNumber": "phone-number"
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-connector-definition \
--name MyGreengrassConnectors \
--initial-version '{"Connectors": [{"Id": "MyTwilioNotificationsConnector",
 "ConnectorArn": "arn:aws:greengrass:region::/connectors/TwilioNotifications/
versions/4", "Parameters": {"TWILIO_ACCOUNT_SID": "account-sid",
 "TwilioAuthTokenSecretArn": "secret-arn", "TwilioAuthTokenSecretArn-
ResourceId": "TwilioAuthToken", "DefaultFromPhoneNumber": "phone-number"}}]}'
```

> **Note**
>
> TwilioAuthToken 是上一步中创建密钥资源时所使用的 ID。

2. 从输出中复制该连接器定义的 LatestVersionArn。您将使用此值向部署到核心的组版本添加连接器定义版本。

步骤 4：创建 Lambda 函数部署程序包

要创建 Lambda 函数，您必须先创建一个包含函数代码和依赖项的 Lambda 函数部署包。Greengrass Lambda 函数需要 AWS IoT Greengrass Core 软件开发工具包来执行各项任务，例如在核心环境中与 MQTT 消息通信和访问本地机密等。本教程将创建一个 Python 函数，因此您需要在部署包中使用 Python 版本的软件开发工具包。

1. 从 AWS IoT Greengrass Core 软件开发工具包下载页面，将适用于 Python 的 AWS IoT Greengrass Core 软件开发工具包下载到您的计算机上。
2. 解压缩下载的程序包以获取软件开发工具包。软件开发工具包是 greengrasssdk 文件夹。
3. 将以下 Python 代码函数保存在名为 temp_monitor.py 的本地文件中。

```
import greengrasssdk
import json
import random

client = greengrasssdk.client('iot-data')

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']
```

```
    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. 将以下项目压缩到名为 `temp_monitor_python.zip` 的文件中。在创建 ZIP 文件时，仅包含代码和依赖项，而不包含文件夹。

   - temp_monitor.py。应用程序逻辑。
   - greengrasssdk。发布 MQTT 消息的 Python Greengrass Lambda 函数所需的库。

   此即 Lambda 函数部署程序包。

步骤 5：创建 Lambda 函数

现在，创建一个使用部署程序包的 Lambda 函数。

1. 创建 IAM 角色，以便您可以在创建该函数时传入角色 ARN。

JSON Expanded

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "lambda.amazonaws.com"
            },
           "Action": "sts:AssumeRole"
        }
    ]
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
 "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": {"Service":
 "lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
```

> **Note**
>
> AWS IoT Greengrass 不使用此角色，因为对 Greengrass Lambda 函数的权限在 Greengrass 组角色中指定。对于本教程，您将创建一个空角色。

2. 从输出中复制 `Arn`。
3. 使用 AWS Lambda API 创建 TempMonitor 函数。以下命令假定该 zip 文件位于当前目录中。

   - 将 *`role-arn`* 替换为复制的 `Arn`。

```
aws lambda create-function \
--function-name TempMonitor \
--zip-file fileb://temp_monitor_python.zip \
--role role-arn \
--handler temp_monitor.function_handler \
--runtime python3.7
```

4. 发布该函数的版本。

```
aws lambda publish-version --function-name TempMonitor --description 'First
 version'
```

5. 为发布的版本创建别名。

   Greengrass 组可以按别名（推荐）或版本引用 Lambda 函数。使用别名，您可以更轻松地管理代码更新，因为您在更新函数代码时，不必更改订阅表或组定义。相反，您只需将别名指向新的函数版本。

   > **Note**
   >
   > AWS IoT Greengrass 不支持 $LATEST 版本的 Lambda 别名。

```
aws lambda create-alias --function-name TempMonitor --name GG_TempMonitor --
function-version 1
```

6. 从输出中复制 `AliasArn`。在为 AWS IoT Greengrass 配置函数时以及在创建订阅时，使用此值。

现在，您已准备就绪，可以为 AWS IoT Greengrass 配置函数。

步骤 6：创建函数定义和版本

要在 AWS IoT Greengrass 核心上使用 Lambda 函数，请创建一个按别名引用 Lambda 函数的函数定义版本，并定义组级别的配置。有关更多信息，请参阅the section called “控制 Greengrass Lambda 函数执行”。

1. 创建一个包含初始版本的函数定义。

   - 将 *alias-arn* 替换为您创建别名时复制的 `AliasArn`。

JSON Expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
    "Functions": [
        {
            "Id": "TempMonitorFunction",
            "FunctionArn": "alias-arn",
            "FunctionConfiguration": {
                "Executable": "temp_monitor.function_handler",
                "MemorySize": 16000,
                "Timeout": 5
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "TempMonitorFunction",
 "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
 "temp_monitor.function_handler", "MemorySize": 16000,"Timeout": 5}}]}'
```

2. 从输出中复制 `LatestVersionArn`。您将使用此值向部署到核心的组版本添加函数定义版本。
3. 从输出中复制 `Id`。您在稍后更新该函数时将使用此值。

步骤 7：创建订阅定义和版本

在该步骤中，您将添加一个订阅，使 Lambda 函数将输入数据发送到连接器。此连接器定义它订阅的 MQTT 主题，因此该订阅使用其中一个主题。这与示例函数发布到的主题相同。

对于本教程，您还可以创建订阅，以允许函数从 AWS IoT 接收模拟温度读数，并允许 AWS IoT 从连接器接收状态信息。

1. 创建一个订阅定义，其中包含带有订阅的初始版本。

   - 将 *alias-arn* 替换为您为函数创建别名时复制的 `AliasArn`。将此 ARN 用于使用它的两个订阅。

JSON Expanded

```
aws greengrass create-subscription-definition --initial-version '{
    "Subscriptions": [
        {
            "Id": "TriggerNotification",
            "Source": "alias-arn",
            "Subject": "twilio/txt",
            "Target": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4"
        },
        {
            "Id": "TemperatureInput",
            "Source": "cloud",
            "Subject": "temperature/input",
            "Target": "alias-arn"
        },
        {
            "Id": "OutputStatus",
            "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
            "Subject": "twilio/message/status",
            "Target": "cloud"
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-subscription-definition \
--initial-version '{"Subscriptions": [{"Id": "TriggerNotification", "Source":
 "alias-arn", "Subject": "twilio/txt", "Target": "arn:aws:greengrass:region::/
connectors/TwilioNotifications/versions/4"},{"Id": "TemperatureInput",
 "Source": "cloud", "Subject": "temperature/input", "Target": "alias-arn"},
{"Id": "OutputStatus", "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4", "Subject": "twilio/message/status", "Target":
 "cloud"}]}'
```

2. 从输出中复制 `LatestVersionArn`。您将使用此值向部署到核心的组版本添加订阅定义版本。

步骤 8：创建组版本

现在已准备就绪，可以创建一个包含要部署的所有项目的组版本。要实现此目的，需要创建一个引用每种组件类型的目标版本的组版本。

首先，获取核心定义版本的组 ID 和 ARN。这些值是创建组版本所必需的。

1. 获取组的 ID 和最新组版本：

   a. 获取目标 Greengrass 组和组版本的 ID。此过程假定这是最新的组和组版本。以下查询将返回最近创建的组。

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

   或者，您也可以按名称查询。系统不要求组名称是唯一的，所以可能会返回多个组。

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

   > **Note**
   >
   > 您还可以在 AWS IoT 控制台中找到这些值。组 ID 显示在组的设置页面上。组版本 ID 显示在组的部署选项卡上。

   b. 从输出中复制目标组的 `Id`。您将使用此值获取核心定义版本及部署组的时间。

   c. 从输出中复制 `LatestVersion`，这是添加到组的最后一个版本的 ID。您将使用此值获取核心定义版本。

2. 获取核心定义版本的 ARN：

   a. 获取组版本。在此步骤中，我们假定最新组版本包含一个核心定义版本。

   - 使用为组复制的 *## group-id*Id。
   - 使用为组复制的 `LatestVersion` 替换 *group-version-id*。

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

b. 从输出中复制 CoreDefinitionVersionArn。

3. 创建组版本。

   - 使用为组复制的 *##* group-idId。
   - 使用为核心函数版本复制的 CoreDefinitionVersionArn 替换 *core-definition-version-arn*。
   - 将 *resource-definition-version-arn* 替换为您针对资源定义复制的 LatestVersionArn。
   - 将 *connector-definition-version-arn* 替换为您针对连接器定义复制的 LatestVersionArn。
   - 使用为函数定义复制的 LatestVersionArn 替换 *function-definition-version-arn*。
   - 将 *subscription-definition-version-arn* 替换为您针对订阅定义复制的 LatestVersionArn。

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

4. 从输出中复制 Version 的值。这是组版本的 ID。您将使用此值部署组版本。

步骤 9：创建部署

将组部署到核心设备。

1. 在核心设备终端中，确保 AWS IoT Greengrass 守护程序正在运行。

   a. 要检查守护程序是否正在运行，请执行以下操作：

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

如果输出包含 root 的 /greengrass/ggc/packages/1.11.6/bin/daemon 条目，则表示守护程序正在运行。

b. 启动进程守护程序：

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

2. 创建 部署。

- 使用为组复制的 *##* group-idId。
- 使用为新组版本复制的 *##* group-version-idVersion。

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

3. 从输出中复制 DeploymentId。
4. 获取部署状态。

- 使用为组复制的 *##* group-idId。
- 将 *deployment-id* 替换为您针对部署复制的 DeploymentId。

```
aws greengrass get-deployment-status \
--group-id group-id \
--deployment-id deployment-id
```

如果状态为 Success，则部署成功。有关问题排查帮助，请参阅排查问题。

测试解决方案

1. 在 AWS IoT 控制台主页上，选择测试。
2. 对于订阅主题，请使用以下值，然后选择订阅。Twilio 通知连接器向此主题发布状态信息。

属性	Value
订阅主题	twilio/message/status
MQTT 负载显示	将负载显示为字符串

3. 对于发布到主题，请使用以下值，然后选择发布 来调用函数。

属性	Value
主题	temperature/input

消息	将 *recipient-name* 替换为文本消息收件人的姓名，将 *recipient-phone-number* 替换为文本消息收件人的电话号码。示例：+12345000000
	
	{
	"to_name": " *recipient-name* ",
	"to_number": " *recipient-phone-number* ",
	"temperature": 31
	}
	
	如果您使用的是试用账号，则必须将非 Twilio 收件人的电话号码添加到经过验证的电话号码列表中。有关更多信息，请参阅验证您的个人电话号码。

如果成功，收件人将收到短信，控制台将在输出数据中显示 `success` 状态。

现在，将输入消息中的 `temperature` 更改为 **29** 并发布。由于温度小于 30，因此 TempMonitor 函数不会触发 Twilio 消息。

另请参阅

- 使用连接器与服务和协议集成
- the section called “AWS 提供的 Greengrass 连接器”
- the section called “连接器入门（控制台）”
- AWS CLI 命令参考中的AWS Secrets Manager 命令
- AWS CLI 命令参考中的AWS Identity and Access Management (IAM) 命令
- AWS CLI 命令参考中的AWS Lambda 命令
- AWS CLI 命令参考中的AWS IoT Greengrass 命令

Greengrass Discovery RESTful API

与 AWS IoT Greengrass 核心通信的所有客户端设备都必须是 Greengrass 组的成员。每个组必须有一个 Greengrass 核心。发现 API 使设备能够检索连接到 Greengrass 核心（与客户端设备位于同一 Greengrass 组中）所需的信息。当客户端设备首次联机时，它可以连接到 AWS IoT Greengrass 服务并使用发现 API 来查找以下内容：

- 设备所属的组。一个客户端设备最多可以是 10 个组的成员。
- 组中 Greengrass 核心的 IP 地址和端口。
- 组 CA 证书，该证书可用于对 Greengrass 核心设备进行身份验证。

> **Note**
>
> 客户端设备还可以使用 AWS IoT 设备软件开发工具包发现 Greengrass 核心的连接信息。有关更多信息，请参阅AWS IoT Device 软件开发工具包。

要使用此 API，请将 HTTP 请求发送到发现 API 终端节点。例如：

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

有关 AWS IoT Greengrass Discovery API 支持的 Amazon Web Services 区域和端点的列表，请参阅AWS 一般参考中 AWS IoT Greengrass 端点和配额。这是一个仅限数据层面的 API。组管理和 AWS IoT Core 操作的终端节点与发现 API 终端节点不同。

请求

请求包含标准 HTTP 标头，并发送到 Greengrass 发现终端节点，如下面的示例中所示。

端口号取决于核心配置为是通过端口 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”。

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
```

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。

> **Note**
>
> 这些示例使用 Amazon Trust Services (ATS) 终端节点，该终端节点用于 ATS 根 CA 证书（推荐）。终端节点必须与 CA 证书类型匹配。有关更多信息，请参阅the section called “服务端点必须与证书类型匹配”。

响应

请求成功后，响应将包括标准的 HTTP 标头以及以下代码和正文：

```
HTTP 200
BODY: response document
```

有关更多信息，请参阅示例发现响应文档。

发现授权

检索连接信息需要一个允许发起人执行 `greengrass:Discover` 操作的策略。使用客户端证书的 TLS 双向身份验证是唯一接受的身份验证形式。以下是允许发起人执行该操作的示例策略：

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "greengrass:Discover",
        "Resource": ["arn:aws:iot:us-west-2:123456789012:thing/MyThingName"]
     }]
}
```

示例发现响应文档

以下文档显示了对属于组（包含一个 Greengrass 核心、一个终端节点和一个组 CA 证书）的客户端设备的响应：

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

以下文档显示了对属于两个组（包含一个 Greengrass 核心、多个终端节点和多个组 CA 证书）的客户端设备的响应：

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
```

```
            "hostAddress": "core-01-address",
            "portNumber": core-01-port,
            "metadata": "core-01-connection-1-description"
          },
          {
            "id": "core-01-connection-id-2",
            "hostAddress": "core-01-address-2",
            "portNumber": core-01-port-2,
            "metadata": "core-01-connection-2-description"
          }
        ]
      }
    ],
    "CAs": [
      "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
      "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
      "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
    ]
  },
  {
    "GGGroupId": "gg-group-02-id",
    "Cores": [
       {
          "thingArn":"core-02-thing-arn",
          "Connectivity" : [
          {
            "id": "core-02-connection-id",
            "hostAddress": "core-02-address",
            "portNumber": core-02-port,
            "metadata": "core-02-connection-1-description"
          }
          ],
          "CAs": [
              "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
              "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
              "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
          ]
      }
  ]
  }
}
```

> **Note**
>
> 一个 Greengrass 组必须恰好定义一个 Greengrass 核心。来自包含一组 Greengrass 核心的 AWS IoT Greengrass 服务的任何响应仅包含一个 Greengrass 核心。

如果已安装 cURL，则可测试发现请求。例如：

```
$ curl --cert 1a23bc4d56.cert.pem --key 1a23bc4d56.private.key https://greengrass-
ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/thing/MyDevice
{"GGGroups":[{"GGGroupId":"1234a5b6-78cd-901e-2fgh-3i45j6k1789","Cores":
[{"thingArn":"arn:aws:iot:us-west-2:1234567
89012:thing/MyFirstGroup_Core","Connectivity":
[{"Id":"AUTOIP_192.168.1.4_1","HostAddress":"192.168.1.5","PortNumber
":8883,"Metadata":""}]}],"CAs":["-----BEGIN CERTIFICATE-----\ncert-contents\n-----END
 CERTIFICATE-----\n"]}]}
```

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。

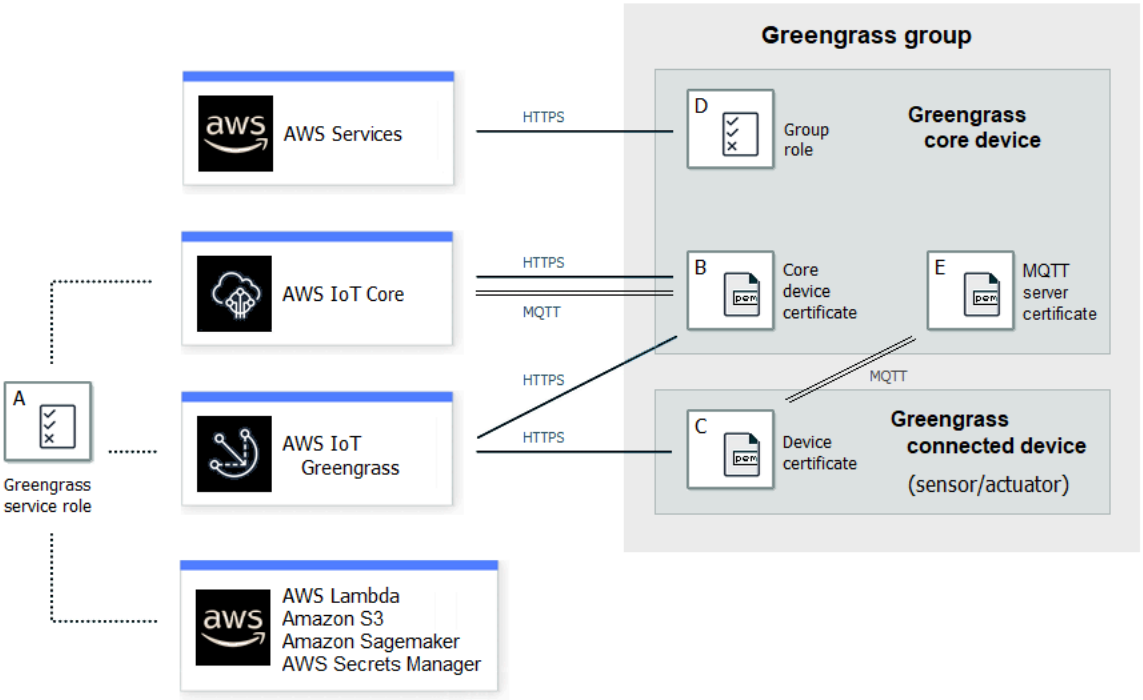
此文档将帮助您了解如何在使用 AWS IoT Greengrass 时应用责任共担模式。以下主题说明如何配置 AWS IoT Greengrass 以实现您的安全性和合规性目标。您还会了解如何使用其它 AWS 服务以帮助您监控和保护 AWS IoT Greengrass 资源。

主题

AWS IoT Greengrass 安全概述

AWS IoT Greengrass 使用 X.509 证书、 AWS IoT 策略以及 IAM 策略和角色来保护在本地 Greengrass 环境中的设备上运行的应用程序。

下图显示了 AWS IoT Greengrass 安全模型的组件：



A - Greengrass 服务角色

从 AWS IoT Core、 AWS Lambda和其他 AWS 服务访问您的 AWS 资源 AWS IoT Greengrass 时由客户创建的 IAM 角色。有关更多信息，请参阅the section called “Greengrass 服务角色”。

B - 核心设备证书

一个 X.509 证书，用于使用和对 Greengrass 内核进行身份验证。 AWS IoT Core AWS IoT Greengrass有关更多信息，请参阅the section called “设备身份验证和授权”。

C - 设备证书

一个 X.509 证书，用于使用和对客户端设备（也称为连接的设备）进行身份验证。 AWS IoT Core AWS IoT Greengrass有关更多信息，请参阅the section called “设备身份验证和授权”。

D - 组角色

从 Greengrass 核心调用 AWS 服务 AWS IoT Greengrass 时由客户创建的 IAM 角色担任。

您可以使用此角色来指定用户定义的 Lambda 函数和连接器访问 AWS 服务（例如 DynamoDB）所需的访问权限。您还可以使用它 AWS IoT Greengrass 来允许将流管理器流导出到 AWS 服务并写入 CloudWatch 日志。有关更多信息，请参阅the section called “Greengrass 组角色”。

> **Note**
>
> AWS IoT Greengrass 不使用中为 Lambda 函数的云版本指定的 Lambda 执行角色。 AWS Lambda

E - MQTT 服务器证书

用于 Greengrass 核心设备与 Greengrass 组中的客户端设备之间传输层安全性协议（TLS）相互身份验证的证书。该证书由存储在 AWS Cloud 中的组 CA 证书签名。

设备连接工作流程

本节介绍客户端设备如何连接到 AWS IoT Greengrass 服务和 Greengrass 核心设备。客户端设备是与核心 AWS IoT Core 设备属于同一 Greengrass 组的注册设备。

- Greengrass 核心设备使用其设备证书、私钥和根 CA 证书 AWS IoT Core 来连接服务。 AWS IoT Greengrass 在核心设备上，配置文件中的 `crypto` 对象指定这些项目的文件路径。
- Greengrass 核心设备从 AWS IoT Greengrass 服务下载组成员信息。
- 部署 Greengrass 核心设备时，Device Certificate Manager (DCM) 会处理 Greengrass 核心设备的本地服务器证书管理。
- 客户端设备使用其设备证书、私钥和 AWS IoT Core 根 CA 证书连接到 AWS IoT Greengrass 服务。连接后，客户端设备会使用 Greengrass 发现服务查找其 Greengrass 核心设备的 IP 地址。客户端设备还会下载组 CA 证书，此证书用于与 Greengrass 核心设备进行 TLS 相互身份验证。
- 客户端设备会尝试连接到 Greengrass 核心设备，并传递其设备证书和客户端 ID。如果客户端 ID 与客户端设备的事物名称匹配并且证书有效（属于 Greengrass 组），将进行连接。否则，将终止连接。

客户端设备的 AWS IoT 策略必须授予`greengrass:Discover`允许客户端设备发现核心连接信息的权限。有关策略语句的更多信息，请参阅the section called “发现授权”。

配置 AWS IoT Greengrass 安全性

要配置您的 Greengrass 应用程序的安全性，请执行以下操作：

1. 为你的 Greengrass 核心设备创建一 AWS IoT Core 件东西。
2. 为您的 Greengrass 核心设备生成密钥对和设备证书。
3. 创建 AWS IoT 策略并将其附加到设备证书。该证书和政策允许 Greengrass 核心设备访问和服务。 AWS IoT Core AWS IoT Greengrass 有关更多信息，请参阅核心设备的最低 AWS IoT 策略。

   > **Note**
   >
   > 不支持在核心设备的策略中使用事物 AWS IoT 策略变量 (`iot:Connection.Thing.*`)。核心使用相同的设备证书与之建立多个连接， AWS IoT Core 但连接中的客户端 ID 可能与核心事物名称不完全匹配。

4. 创建 Greengrass 服务角色。此 IAM 角色 AWS IoT Greengrass 授权代表您访问其他 AWS 服务的资源。这 AWS IoT Greengrass 允许执行基本任务，例如检索 AWS Lambda 功能和管理设备影子。

   您可以跨 AWS 区域使用相同的服务角色，但在您使用的每个 AWS 区域 位置都必须与您的 AWS 账户 服务角色相关联 AWS IoT Greengrass。

5. （可选）创建 Greengrass 组角色。此 IAM 角色向在 Greengrass 核心上运行的 Lambda 函数和连接器授予调用服务的权限。 AWS 例如，Kinesis Firehose 连接器需要权限才能将记录写入亚马逊数据 Firehos e 传输流。

   您只能将一个角色附加到 Greengrass 组。

6. 为每台连接到 Greengrass 核心的设备创建一个 AWS IoT Core 东西。

   > **Note**
   >
   > 你也可以使用现有的 AWS IoT Core 东西和证书。

7. 为连接到 Greengrass 核心的每台设备创建设备证书、密钥对和 AWS IoT 策略。

AWS IoT Greengrass 核心安全主体

Greengrass 核心使用以下安全原则 AWS IoT ：客户端、本地 MQTT 服务器和本地密钥管理器。这些委托人的配置存储在 `config.json` 配置文件的 `crypto` 对象中。有关更多信息，请参阅the section called “AWS IoT Greengrass 核心配置文件”。

此配置包含用于身份验证和加密的主要组件使用的私有密钥的路径。 AWS IoT Greengrass 支持两种模式的私有密钥存储：基于硬件或基于文件系统（默认）。有关在硬件安全模块上存储密钥的更多信息，请参阅the section called “硬件安全性集成”。

AWS IoT 客户端

AWS IoT 客户端（物联网客户端）通过互联网管理Greengrass核心和之间的通信。 AWS IoT Core AWS IoT Greengrass 在为此通信建立 TLS 连接时，使用带有公钥和私钥的 X.509 证书进行相互身份验证。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的 X.509 证书和 AWS IoT Core。

该 IoT 客户端支持 RSA 和 EC 证书和密钥。该证书和私有密钥是为 `config.json` 中的 `IoTCertificate` 委托人指定的。

MQTT 服务器

本地 MQTT 服务器通过本地网络管理 Greengrass 核心与组中客户端设备之间的通信。 AWS IoT Greengrass 在为此通信建立 TLS 连接时，使用带有公钥和私钥的 X.509 证书进行相互身份验证。

默认情况下， AWS IoT Greengrass 会为您生成 RSA 私钥。要将核心配置为使用其他私有密钥，您必须为 `config.json` 中的 `MQTTServerCertificate` 委托人提供关键路径。您负责轮换客户提供的密钥。

私有密钥支持

	RSA 密钥	EC 密钥
密钥类型	Supported	Supported
关键参数	Minimum 2048-bit length	NIST P-256 or NIST P-384 curve
磁盘格式	PKCS#1, PKCS#8	SECG1, PKCS#8
最低 GGC 版本	• 使用默认 RSA 密钥：1.0 • 指定 RSA 密钥：1.7	• 指定 EC 密钥：1.9

私有密钥的配置决定了相关过程。有关 Greengrass 核心作为服务器支持的密码套件的列表，请参阅 the section called “TLS 密码套件支持”。

如果未指定私有密钥（默认）

- AWS IoT Greengrass 根据您的轮换设置轮换密钥。
- 核心生成一个 RSA 密钥，用于生成证书。
- MQTT 服务器证书包含一个 RSA 公有密钥和一个 SHA-256 RSA 签名。

如果指定了 RSA 私有密钥（需要 GGC v1.7 或更高版本）

- 您负责轮换密钥。
- 核心使用指定密钥来生成证书。
- RSA 密钥的最小长度必须为 2048 位。
- MQTT 服务器证书包含一个 RSA 公有密钥和一个 SHA-256 RSA 签名。

如果指定了 EC 私有密钥（需要 GGC v1.9 或更高版本）

- 您负责轮换密钥。
- 核心使用指定密钥来生成证书。
- EC 私有密钥必须使用 NIST P-256 或 NIST P-384 曲线。
- MQTT 服务器证书包含一个 EC 公有密钥和一个 SHA-256 RSA 签名。

  核心提供的 MQTT 服务器证书包含一个 SHA-256 RSA 签名，无理论密钥类型如何。因此，客户端必须支持 SHA-256 RSA 证书验证才能与核心建立安全连接。

Secrets Manager

本地密钥管理器可以安全地管理您在中创建的密钥的本地副本 AWS Secrets Manager。它使用私有密钥来保护用于对密钥进行加密的数据密钥。有关更多信息，请参阅将密钥部署到核心。

默认情况下将使用 IoT 客户端私有密钥，但您可以为 `config.json` 中的 `SecretsManager` 委托人指定其他私有密钥。仅支持 RSA 密钥类型。有关更多信息，请参阅the section called “指定用于密钥加密的私有密钥”。

> **Note**
>
> 当前，在使用基于硬件的私钥时，仅 AWS IoT Greengrass 支持 PKCS #1 v1.5 填充机制，用于加密和解密本地机密。如果您按照供应商提供的说明手动生成基于硬件的私钥，请务必选择 PKCS #1 v1.5。 AWS IoT Greengrass 不支持最佳非对称加密填充 (OAEP)。

私有密钥支持

	RSA 密钥	EC 密钥
密钥类型	Supported	Not supported
关键参数	Minimum 2048-bit length	Not applicable
磁盘格式	PKCS#1, PKCS#8	Not applicable
最低 GGC 版本	1.7	Not applicable

MQTT 消息传递工作流中的托管订阅

AWS IoT Greengrass 使用订阅表来定义如何在 Greengrass 组中的客户端设备、函数和连接器之间以及 AWS IoT Core 如何与本地影子服务交换 MQTT 消息。每次订阅都指定了发送或接收消息的来源、目标和 MQTT 主题（或主题）。 AWS IoT Greengrass 仅当定义了相应的订阅时，才允许将消息从源发送到目标。

订阅仅定义从源到目标的单向消息流。要支持双向消息交换，您必须创建两个订阅，每个订阅针对一个方向。

TLS 密码套件支持

AWS IoT Greengrass 使用 AWS IoT Core 传输安全模型通过 TLS 密码套件对与云的通信进行加密。此外， AWS IoT Greengrass 数据在静态时会被加密（在云中）。有关 AWS IoT Core 传输安全和支持的密码套件的更多信息，请参阅《AWS IoT Core 开发人员指南》中的传输安全。

用于本地网络通信的受支持密码套件

相反 AWS IoT Core，该 AWS IoT Greengrass 内核支持以下用于证书签名算法的本地网络 TLS 密码套件。当私有密钥存储在文件系统上时，所有这些密码套件都受支持。当核心被配置为使用硬件安全模块 (HSM) 时，子集将受支持。有关更多信息，请参阅 the section called “安全委托人” 和 the section called “硬件安全性集成”。该表还包括支持所需的 AWS IoT Greengrass Core 软件的最低版本。

	密码	HSM 支持	最低 GGC 版本
TLSv1.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Supported	1.0
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Supported	1.0
	TLS_RSA_WITH_AES_128_CBC_SHA	Not supported	1.0
	TLS_RSA_WITH_AES_128_GCM_SHA256	Not supported	1.0
	TLS_RSA_WITH_AES_256_CBC_SHA	Not supported	1.0
	TLS_RSA_WITH_AES_256_GCM_SHA384	Not supported	1.0
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Supported	1.9
	TLS_ECDHE_ECDSA_WI	Supported	1.9

	密码	HSM 支持	最低 GGC 版本
	TH_AES_25 6_GCM_SHA384		
TLSv1.1	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Supported	1.0
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	Not supported	1.0
	TLS_RSA_W ITH_AES_2 56_CBC_SHA	Not supported	1.0
TLSv1.0	TLS_ECDHE _RSA_WITH _AES_128_CBC_SHA	Supported	1.0
	TLS_ECDHE _RSA_WITH _AES_256_CBC_SHA	Supported	1.0
	TLS_RSA_W ITH_AES_1 28_CBC_SHA	Not supported	1.0
	TLS_RSA_W ITH_AES_2 56_CBC_SHA	Not supported	1.0

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

AWS IoT Greengrass 有三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 通过互联网进行的通信是加密的。
- the section called “通过本地网络传输的数据”. Greengrass 核心设备和客户端设备通过本地网络进行的通信是加密的。
- 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 AWS Cloud 的所有数据都使用 MQTT 或 HTTPS 协议通过 TLS 连接发送，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 AWS IoT Core 开发人员指南中的传输安全。

通过本地网络传输的数据

AWS IoT Greengrass 使用 TLS 对 Greengrass 核心设备和客户端设备之间通过本地网络传输的所有通信进行加密。有关详细信息，请参阅用于本地网络通信的支持密码套件。

您需要负责保护本地网络和私钥的安全。

对于 Greengrass 核心设备，您需要负责：

- 使用最新的安全补丁保持内核为最新状态。
- 使用最新的安全补丁保持系统库为最新状态。
- 保护私钥安全。有关更多信息，请参阅the section called “密钥管理”。

对于客户端设备，您需要负责：

- 使 TLS 堆栈保持最新状态。
- 保护私钥安全。

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 Lambda 函数、连接器、AWS IoT Greengrass 核心开发工具包和系统组件（如流管理器）之间的通信。

静态加密

AWS IoT Greengrass 存储您的数据：

- the section called “AWS Cloud 中的静态数据”. 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）。

AWS Cloud 中的静态数据

AWS IoT Greengrass 加密存储在 AWS Cloud 中的客户数据。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the section called “密钥加密”。

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。在以下情况下，AWS IoT Greengrass 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。

- MQTT 服务器密钥用于 MQTT 服务器证书对核心设备和客户端设备之间的 TLS 连接进行身份验证。有关更多信息，请参阅the section called “设备身份验证和授权”。
- 本地 Secrets Manager 还使用 IoT 客户端密钥来保护用于加密本地机密的数据密钥，但您可以提供自己的私钥。有关更多信息，请参阅the section called “密钥加密”。

Greengrass 核心支持使用文件系统权限、硬件安全模块或同时使用两者进行的私钥存储。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。

在 Greengrass 核心上，您的私钥位置在 `config.json` 文件的 `crypto` 部分中指定。如果您将核心配置为使用客户提供的 MQTT 服务器证书的密钥，您需要负责轮换该密钥。有关更多信息，请参阅the section called “安全委托人”。

对于客户端设备，您需要负责使 TLS 堆栈保持最新状态并保护私钥安全。私钥与设备证书一起使用，用于对与 AWS IoT Greengrass 服务的 TLS 连接进行身份验证。

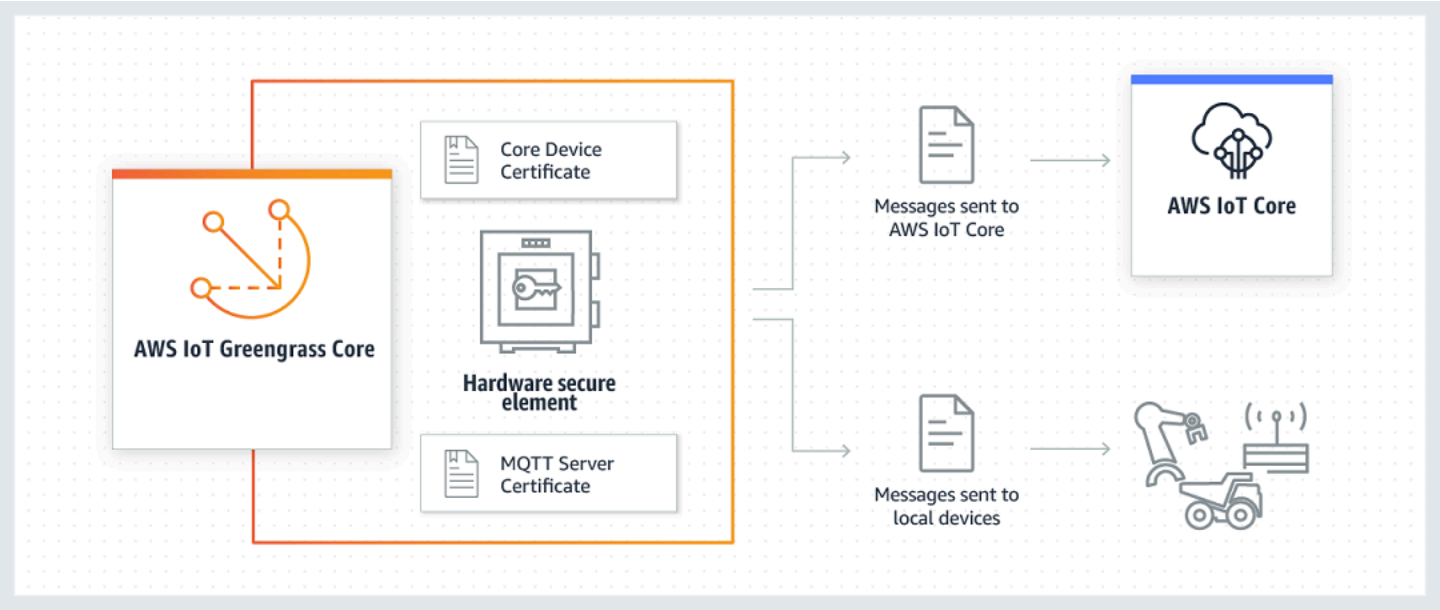
硬件安全性集成

此功能适用于 AWS IoT Greengrass Core v1.7 及更高版本。

AWS IoT Greengrass 支持通过 PKCS#11 接口使用硬件安全模块 (HSM)，以实现安全存储和私有密钥分载。这样可防止密钥在软件中暴露或重复。私有密钥可以安全地存储在硬件模块中，如 HSM、受信任平台模块 (TPM) 或其他加密元素。

在AWS Partner设备目录中搜索符合此功能条件的设备。

下图显示了 AWS IoT Greengrass 核心的硬件安全架构。



在标准安装中，AWS IoT Greengrass 使用两个私有密钥。一个密钥在 Greengrass 核心连接到 AWS IoT Core 时由 AWS IoT 客户端（IoT 客户端）组件在传输层安全性 (TLS) 握手过程中使用。（此密钥也称为核心私有密钥。）另一个密钥由本地 MQTT 服务器使用，它使 Greengrass 设备能够与 Greengrass 核心进行通信。如果您要为这两个组件使用硬件安全性，可以使用共享私有密钥或单独的私有密钥。有关更多信息，请参阅the section called “ 预配置实践”。

> **Note**
>
> 对于标准安装，本地 Secrets Manager 还对其加密过程使用 IoT 客户端密钥，但您可以使用您自己的私有密钥。它必须是最小长度为 2048 位的 RSA 密钥。有关更多信息，请参阅the section called “指定用于密钥加密的私有密钥”。

要求

在为 Greengrass 核心配置硬件安全性之前，您必须具备以下对象：

- 一个硬件安全模块 (HSM)，可支持您的用于 IoT 客户端、本地 MQTT 服务器和本地 Secrets Manager 组件的目标私有密钥配置。该配置可以包含一个、两个或三个基于硬件的私有密钥，具体取决于您是否配置组件共享密钥。有关私有密钥支持的更多信息，请参阅 the section called “安全委托人”。
  - 对于 RSA 密钥：RSA-2048 密钥大小（或更大）和 PKCS # 1 1.5 版本签名方案。
  - 对于 EC 密钥：NIST P-256 或 NIST P-384 曲线。

> **Note**
>
> 在AWS Partner设备目录中搜索符合此功能条件的设备。

- 一个 PKCS#11 提供程序库，可在运行时加载（使用 libdl）并提供 PKCS#11 函数。
- 硬件模块必须可按槽标签解析，如 PKCS#11 规范所定义。
- 私有密钥必须使用供应商提供的预配置工具在 HSM 上生成和加载。
- 私有密钥必须由对象标签来解析。
- 核心设备证书。这是与私有密钥对应的 IoT 客户端证书。
- 如果使用的是 Greengrass OTA 更新代理，则必须安装 OpenSSL libp11 PKCS#11 包装程序库。有关更多信息，请参阅the section called “配置 OTA 更新”。

此外，确保满足以下条件：

- 与私有密钥关联的 IoT 客户端证书在 AWS IoT 中注册并激活。您可以在AWS IoT控制台的管理下进行验证，展开所有设备，选择事物，然后为核心内容选择证书选项卡。
- 在核心设备上安装 AWS IoT Greengrass 核心软件 v1.7 或更高版本，如“入门”教程中的模块 2 所述。对 MQTT 服务器使用 EC 密钥需要 1.9 或更高版本。
- 将证书附加到 Greengrass 核心。您可以从 AWS IoT 控制台的核心内容的管理页面进行验证。

> **Note**
>
> 目前，AWS IoT Greengrass 不支持直接从 HSM 加载 CA 证书或 IoT 客户端证书。证书必须以纯文本文件形式加载到文件系统中 Greengrass 可读取的位置。

AWS IoT Greengrass 核心的硬件安全性配置

硬件安全性在 Greengrass 配置文件中进行配置。这是 `config.json` 文件（位于 */greengrass-root*`/config` 目录中）。

> **Note**
>
> 要演练使用纯软件实施来设置 HSM 配置的过程，请参阅 the section called “模块 7：模拟硬件安全集成”。

> **Important**
>
> 示例中的模拟配置不提供任何安全优势。它只是让您了解 PKCS#11 规范，并在将来计划使用基于硬件的 HSM 时对软件进行初始测试。

要在 AWS IoT Greengrass 中配置硬件安全性，您需在 `config.json` 中编辑 `crypto` 对象。

使用硬件安全性时，`crypto` 对象用于在核心上为 PKCS#11 提供程序库指定证书、私有密钥和资产的路径，如以下示例所示。

```
"crypto": {
  "PKCS11" : {
    "OpenSSLEngine" : "/path-to-p11-openssl-engine",
    "P11Provider" : "/path-to-pkcs11-provider-so",
    "slotLabel" : "crypto-token-name",
    "slotUserPin" : "crypto-token-user-pin"
  },
  "principals" : {
    "IoTCertificate" : {
      "privateKeyPath" : "pkcs11:object=core-private-key-label;type=private",
      "certificatePath" : "file:///path-to-core-device-certificate"
    },
    "MQTTServerCertificate" : {
      "privateKeyPath" : "pkcs11:object=server-private-key-label;type=private"
    },
    "SecretsManager" : {
      "privateKeyPath": "pkcs11:object=core-private-key-label;type=private"
    }
  },
  "caPath" : "file:///path-to-root-ca"
```

`crypto` 对象包含以下属性：

Field	描述	注意
`caPath`	AWS IoT 根 CA 的绝对路径。	必须为以下格式的文件 URI：`file:///absolute/path/to/file` 。

Field	描述	注意
		Note 确保您的终端节点与证书类型对应。
PKCS11		
OpenSSLEngine	可选。OpenSSL 引擎 .so 文件（用于在 OpenSSL 上启用 PKCS # 11 支持）的绝对路径。	必须是文件系统上的文件的路径。 如果您使用 Greengrass OTA 更新代理来实现硬件安全性，则此属性是必需的。有关更多信息，请参阅the section called “配置 OTA 更新”。
P11Provider	PKCS#11 实施的 libdl-loadable 库的绝对路径。	必须是文件系统上的文件的路径。
slotLabel	用于标识硬件模块的槽标签。	必须符合 PKCS # 11 标签规范。
slotUserPin	用于对模块的 Greengrass 核心进行身份验证的用户 PIN。	必须具有足够的权限才能使用配置的私有密钥执行 C_Sign。
principals		
IoTCertificate	The certificate and private key that the core uses to make requests to AWS IoT.	

Field	描述	注意
`IoTCertificate .privateKeyPath`	核心私有密钥的路径。	对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。
`IoTCertificate .certificatePath`	核心设备证书的绝对路径。	必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。
`MQTTServerCertificate`	可选。核心为充当 MQTT 服务器或网关而将其证书结合使用的私有密钥。	
`MQTTServerCertificate .privateKeyPath`	本地 MQTT 服务器私有密钥的路径。	使用此值为本地 MQTT 服务器指定您自己的私有密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。 如果忽略此属性，AWS IoT Greengrass 会根据轮换设置对密钥进行轮换。如果指定，客户将负责对密钥进行轮换。
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 将密钥部署到核心.	

Field	描述	注意
`SecretsManager .privateKeyPath`	本地 Secrets Manager 私有密钥的路径。	仅支持 RSA 密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。必须使用 PKCS#1 v1.5 填充机制生成私有密钥。

Field	描述	注意
`caPath`	AWS IoT 根 CA 的绝对路径。	必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 **Note** 确保您的终端节点与证书类型对应。
`PKCS11`		
`OpenSSLEngine`	可选。OpenSSL 引擎 `.so` 文件（用于在 OpenSSL 上启用 PKCS # 11 支持）的绝对路径。	必须是文件系统上的文件的路径。 如果您使用 Greengrass OTA 更新代理来实现硬件安全性，则此属性是必需的。有关更多信息，请参阅the section called “配置 OTA 更新”。

Field	描述	注意
`P11Provider`	PKCS#11 实施的 libdl-loadable 库的绝对路径。	必须是文件系统上的文件的路径。
`slotLabel`	用于标识硬件模块的槽标签。	必须符合 PKCS # 11 标签规范。
`slotUserPin`	用于对模块的 Greengrass 核心进行身份验证的用户 PIN。	必须具有足够的权限才能使用配置的私有密钥执行 C_Sign。
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoTCertificate.privateKeyPath`	核心私有密钥的路径。	对于文件系统存储，必须为以下格式的文件 URI：`file:///absolute/path/to/file` 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。
`IoTCertificate.certificatePath`	核心设备证书的绝对路径。	必须为以下格式的文件 URI：`file:///absolute/path/to/file` 。
`MQTTServerCertificate`	可选。核心为充当 MQTT 服务器或网关而将其证书结合使用的私有密钥。	

Field	描述	注意
`MQTTServerCertificate .privateKeyPath`	本地 MQTT 服务器私有密钥的路径。	使用此值为本地 MQTT 服务器指定您自己的私有密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`*。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。 如果忽略此属性，AWS IoT Greengrass 会根据轮换设置对密钥进行轮换。如果指定，客户将负责对密钥进行轮换。
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 将密钥部署到核心.	
`SecretsManager .privateKeyPath`	本地 Secrets Manager 私有密钥的路径。	仅支持 RSA 密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`*。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。必须使用 PKCS#1 v1.5 填充机制生成私有密钥。

Field	描述	注意
`caPath`	AWS IoT 根 CA 的绝对路径。	必须为以下格式的文件 URI：`file:///absolute/path/to/file` 。 **Note** 确保您的终端节点与证书类型对应。
`PKCS11`		
`OpenSSLEngine`	可选。OpenSSL 引擎 `.so` 文件（用于在 OpenSSL 上启用 PKCS # 11 支持）的绝对路径。	必须是文件系统上的文件的路径。 如果您使用 Greengrass OTA 更新代理来实现硬件安全性，则此属性是必需的。有关更多信息，请参阅the section called “配置 OTA 更新”。
`P11Provider`	PKCS#11 实施的 libdl-loadable 库的绝对路径。	必须是文件系统上的文件的路径。
`slotLabel`	用于标识硬件模块的槽标签。	必须符合 PKCS # 11 标签规范。
`slotUserPin`	用于对模块的 Greengrass 核心进行身份验证的用户 PIN。	必须具有足够的权限才能使用配置的私有密钥执行 C_Sign。
`principals`		
`IoTCertificate`	The certificate and private key that the core uses to make requests to AWS IoT.	
`IoTCertificate.privateKeyPath`	核心私有密钥的路径。	对于文件系统存储，必须为以下格式的文件

Field	描述	注意
		URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。
`IoTCertificate.certificatePath`	核心设备证书的绝对路径。	必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。
`MQTTServerCertificate`	可选。核心为充当 MQTT 服务器或网关而将其证书结合使用的私有密钥。	
`MQTTServerCertificate.privateKeyPath`	本地 MQTT 服务器私有密钥的路径。	使用此值为本地 MQTT 服务器指定您自己的私有密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。 如果忽略此属性，AWS IoT Greengrass 会根据轮换设置对密钥进行轮换。如果指定，客户将负责对密钥进行轮换。
`SecretsManager`	The private key that secures the data key used for encryption. For more information, see 将密钥部署到核心.	

Field	描述	注意
`SecretsManager .privateKeyPath`	本地 Secrets Manager 私有密钥的路径。	仅支持 RSA 密钥。 对于文件系统存储，必须为以下格式的文件 URI：`file:///`*`absolute/path/to/file`* 。 对于 HSM 存储，必须是指定对象标签的 RFC 7512 PKCS#11 路径。必须使用 PKCS#1 v1.5 填充机制生成私有密钥。

AWS IoT Greengrass 硬件安全性的预配置实践

以下是安全性和性能相关的预配置实践。

安全性

- 使用内部硬件随机数字生成器直接在 HSM 上生成私有密钥。

  > **Note**
  >
  > 在配置要与此功能一起使用的私有密钥（按照硬件供应商提供的说明操作）时，请注意 AWS IoT Greengrass 目前只支持使用 PKCS1 v1.5 填充机制进行本地密钥加密和解密。AWS IoT Greengrass 不支持最优非对称加密填充 (OAEP)。

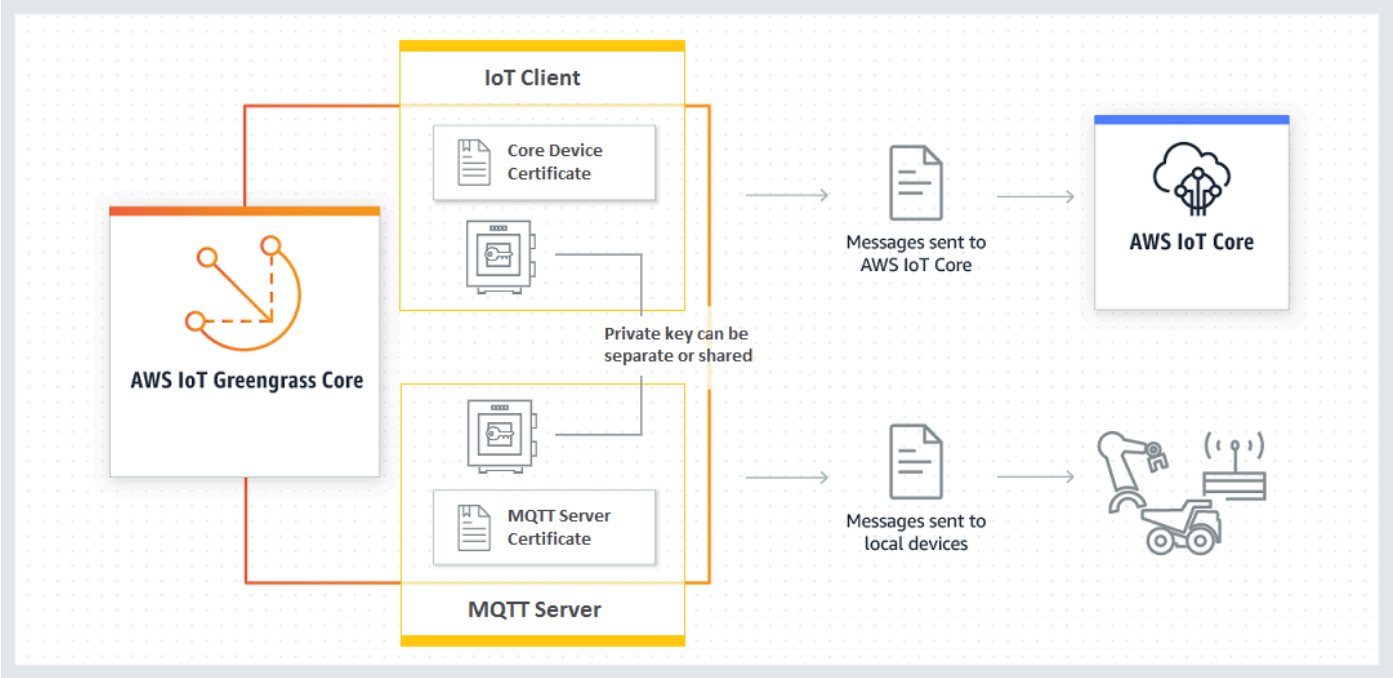
- 配置私有密钥以禁止导出。
- 使用硬件供应商提供的预配置工具，通过硬件保护的私有密钥生成证书签名请求 (CSR)，然后使用 AWS IoT 控制台生成客户端证书。

> **Note**
>
> 在 HSM 上生成私有密钥时，轮换密钥的做法并不适用。

性能

下图显示了 AWS IoT Greengrass 核心上的 IoT 客户端组件和本地 MQTT 服务器。如果您要为这两个组件使用 HSM 配置，可以使用相同的私有密钥或单独的私有密钥。如果您使用单独的密钥，则它们必须存储在相同的插槽中。

> **Note**
>
> AWS IoT Greengrass 不对您存储在 HSM 上的密钥数量施加任何限制，因此，您可以存储用于 IoT 客户端、MQTT 服务器和 Secrets Manager 组件的私有密钥。但是，某些 HSM 供应商可能会对您可以存储在插槽中的密钥数量施加限制。



一般来说，IoT 客户端密钥使用的频率并不高，因为 AWS IoT Greengrass 核心软件维护与云的长期连接。但是，每当 Greengrass 设备连接到核心时，都会使用 MQTT 服务器密钥。这些交互直接影响着性能。

当 MQTT 服务器密钥存储在 HSM 中时，设备连接的速度取决于 HSM 每秒能够执行的 RSA 签名操作数量。例如，如果 HSM 针对 RSA-2048 私有密钥执行 RSASSA-PKCS1-v1.5 签名需要 300 毫秒，则每秒只有三台设备可以连接到 Greengrass 核心。连接完成后，将不再使用 HSM，并且会应用标准 AWS IoT Greengrass 配额。

为了消除性能瓶颈，您可以将 MQTT 服务器的私有密钥存储在文件系统上，而不是 HSM 上。通过这种配置，MQTT 服务器的行为就好像未启用硬件安全性一样。

AWS IoT Greengrass 支持对 IoT 客户端和 MQTT 服务器组件使用多个密钥存储配置，因此您可以优化您的安全性和性能要求。下表包含一些示例配置。

配置	IoT 密钥	MQTT 密钥	性能
HSM 共享密钥	HSM：密钥 A	HSM：密钥 A	受 HSM 或 CPU 限制
HSM 单独密钥	HSM：密钥 A	HSM：密钥 B	受 HSM 或 CPU 限制
仅限适用于 IoT 的 HSM	HSM：密钥 A	文件系统：密钥 B	受 CPU 限制
传统	文件系统：密钥 A	文件系统：密钥 B	受 CPU 限制

要配置 Greengrass 核心以便为 MQTT 服务器使用基于文件系统的密钥，请省略 `config.json` 中的 `principals.MQTTServerCertificate` 部分（或在您不使用 AWS IoT Greengrass 生成的默认密钥时为密钥指定基于文件的路径）。生成的 `crypto` 对象如下所示：

```
"crypto": {
  "PKCS11": {
    "OpenSSLEngine": "...",
    "P11Provider": "...",
    "slotLabel": "...",
    "slotUserPin": "..."
  },
  "principals": {
    "IoTCertificate": {
      "privateKeyPath": "...",
      "certificatePath": "..."
    },
    "SecretsManager": {
      "privateKeyPath": "..."
    }
  },
  "caPath" : "..."
}
```

硬件安全性集成支持的密码套件

在对核心进行硬件安全配置时，AWS IoT Greengrass 支持一组密码套件。这是在配置核心以使用基于文件的安全性时所支持的密码套件的一部分。有关更多信息，请参阅the section called “TLS 密码套件支持”。

> **Note**
>
> 当通过本地网络从 Greengrass 设备连接到 Greengrass 核心时，请确保使用一个受支持的加密套件来建立 TLS 连接。

配置对无线更新的支持

要在使用硬件安全性时启用 AWS IoT Greengrass 核心软件的无线 (OTA) 更新，您必须安装 OpenSC libp11 PKCS#11 包装程序库并编辑 Greengrass 配置文件。有关 OTA 更新的更多信息，请参阅 AWS IoT Greengrass Core 软件的 OTA 更新。

1. 停止 Greengrass 守护程序。

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

   > **Note**
   >
   > *greengrass-root* 表示在您的设备上安装 AWS IoT Greengrass 核心软件的路径。通常，这是 /greengrass 目录。

2. 安装 OpenSSL 引擎。支持 OpenSSL 1.0 或 1.1。

   ```
   sudo apt-get install libengine-pkcs11-openssl
   ```

3. 在您的系统上找到 OpenSSL 引擎的路径 (libpkcs11.so)：

   a. 获取库的已安装程序包的列表。

      ```
      sudo dpkg -L libengine-pkcs11-openssl
      ```

      libpkcs11.so 文件位于 engines 目录中。

b. 复制文件的完整路径（例如 /usr/lib/ssl/engines/libpkcs11.so）。

4. 打开 Greengrass 配置文件。这是位于 */greengrass-root*/config 目录中的 config.json 文件。

5. 对于 OpenSSLEngine 属性，输入 libpkcs11.so 文件的路径。

```
{
 "crypto": {
   "caPath" : "file:///path-to-root-ca",
   "PKCS11" : {
     "OpenSSLEngine" : "/path-to-p11-openssl-engine",
     "P11Provider" : "/path-to-pkcs11-provider-so",
     "slotLabel" : "crypto-token-name",
     "slotUserPin" : "crypto-token-user-pin"
    },
    ...
  }
  ...
}
```

> **Note**
>
> 如果 PKCS11 对象中不存在 OpenSSLEngine 属性，请进行添加。

6. 启动 Greengrass 守护程序。

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

与早期版本的 AWS IoT Greengrass 核心软件的向后兼容性

具有硬件安全性支持的 AWS IoT Greengrass 核心软件完全与为 1.6 和较旧版本生成的 config.json 文件向后兼容。如果 config.json 配置文件中不存在 crypto 对象，则 AWS IoT Greengrass 将使用基于文件的 coreThing.certPath、coreThing.keyPath 和 coreThing.caPath 属性。此向后兼容性应用于 Greengrass OTA 更新，这不会覆盖 config.json 中指定的基于文件的配置。

不具有 PKCS#11 支持的硬件

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 库实施，或如果您想要编写自定义 PKCS#11 提供程序，则应联系您的 AWS 企业支持代表，咨询与集成相关的问题。

另请参阅

- PKCS #11 加密令牌接口使用指南 2.40 版本。编辑者：John Leiseboer 和 Robert Griffin。2014 年 11 月 16 日。OASIS 委员会注意 02。http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html。最新版本：http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html。
- 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。Greengrass 核心设备使用证书和 AWS IoT 策略连接到 AWS IoT Core。该证书和策略还使 AWS IoT Greengrass 能够将配置信息、Lambda 函数、连接器和托管订阅部署到核心设备。
- 客户端设备。客户端设备（也称为联网设备、Greengrass 设备或设备）是通过 MQTT 连接到 Greengrass 核心的设备。它们使用证书和策略来连接到 AWS IoT Core 和 AWS IoT Greengrass 服务。这使客户端设备能够使用 AWS IoT Greengrass 发现服务查找并连接到核心设备。客户端设备使用的是连接到 AWS IoT Core 设备网关和核心设备所用的同一证书。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 the section called “设备连接工作流程”和 the section called “使用 Greengrass 核心管理设备身份验证”。

X.509 证书

核心设备和客户端设备之间以及设备和 AWS IoT Core 或 AWS IoT Greengrass 之间的通信必须经过身份验证。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：

- 通过互联网连接到 AWS IoT Core 和 AWS IoT Greengrass 的 Greengrass 核心上的 AWS IoT 客户端组件。
- 通过互联网连接到 AWS IoT Greengrass 以获取核心发现信息的客户端设备。
- 通过本地网络连接到组中的客户端设备的 Greengrass 核心上的 MQTT 服务器组件。

AWS IoT Greengrass 核心设备在两个位置存储证书：

- */greengrass-root*/certs 中的核心设备证书。通常，核心设备证书命名为 *hash*.cert.pem（例如，86c84488a5.cert.pem）。当核心连接到 AWS IoT Core 和 AWS IoT Greengrass 服务时，AWS IoT 客户端将使用此证书进行相互身份验证。
- */greengrass-root*/ggc/var/state/server 中的 MQTT 服务器证书。MQTT 服务器证书名为 server.crt。此证书用于本地 MQTT 服务器（在 Greengrass 核心上）和 Greengrass 设备之间的相互身份验证。

> **Note**
>
> *greengrass-root* 表示在您的设备上安装 AWS IoT Greengrass Core 软件的路径。通常，这是 /greengrass 目录。

有关更多信息，请参阅 the section called “安全委托人”。

证书颁发机构 (CA) 证书

核心设备和客户端设备将下载用于使用 AWS IoT Core 和 AWS IoT Greengrass 服务进行身份验证的根 CA 证书。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。

> **Note**
>
> 您的根 CA 证书类型必须与终端节点匹配。将 ATS 根 CA 证书与 ATS 端点（首选）一起使用，或者将 VeriSign 根 CA 证书与传统端点一起使用。只有一些 Amazon Web Services 区域支持传统端点。有关更多信息，请参阅 the section called “服务端点必须与证书类型匹配”。

客户端设备还将下载 Greengrass 组 CA 证书。这用于在双向身份验证期间验证 Greengrass 核心上的 MQTT 服务器证书。有关更多信息，请参阅 the section called “设备连接工作流程”。MQTT 服务器证书的默认期限为 7 天。

本地 MQTT 服务器上的证书轮换

客户端设备使用本地 MQTT 服务器证书与 Greengrass 核心设备进行双向身份验证。默认情况下，此证书将在 7 天后过期。此时间期限基于安全最佳实践。MQTT 服务器证书由存储在云中的组 CA 证书签署。

要进行证书轮换，Greengrass 核心设备必须在线并能够定期直接访问 AWS IoT Greengrass 服务。当证书过期后，核心设备将尝试连接到 AWS IoT Greengrass 服务以获取新证书。如果连接成功，核心设备将下载新的 MQTT 服务器证书并重新启动本地 MQTT 服务。此时，连接到此核心的所有客户端设备都将断开连接。如果核心设备在过期时处于离线状态，它不会接收替换证书。任何新的连接到核心服务的尝试都会被拒绝。现有连接不会受影响。客户端设备将无法连接到核心设备，直至恢复了到 AWS IoT Greengrass 服务的连接，并且可以下载新的 MQTT 服务器证书。

您可以将过期时间设置为 7 天到 30 天之间的任何值，具体取决于您的需要。轮换越频繁，就需要越频繁地进行云连接。轮换频率较低可能会带来安全问题。如果您要将证书过期时间值设置为高于 30 天，请联系 AWS Support。

在 AWS IoT 控制台中，您可以在组的 设置 页面管理证书。在 AWS IoT Greengrass API 中，您可以使用UpdateGroupCertificateConfiguration操作。

当 MQTT 服务器证书过期后，所有验证证书的尝试都将失败。客户端设备必须能够检测故障并终止连接。

数据层面操作的 AWS IoT 策略

使用 AWS IoT 策略授权对 AWS IoT Core 和 AWS IoT Greengrass 数据层面的访问。AWS IoT Core 数据层面由设备、用户和应用程序的操作组成，如连接到 AWS IoT Core 和订阅主题。AWS IoT Greengrass 数据层面由 Greengrass 设备的操作组成，如检索部署和更新连接信息。

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。

> **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 策略操作

Greengrass 核心操作

AWS IoT Greengrass 定义 Greengrass 核心设备可以在 AWS IoT 策略中使用的以下策略操作：

`greengrass:AssumeRoleForGroup`

Greengrass 核心设备使用 Token Exchange Service (TES) 系统 Lambda 函数检索凭证的权限。绑定到检索到的凭证的权限基于附加到已配置组角色的策略。

当 Greengrass 核心设备尝试检索凭证时（假设凭证未在本地缓存），将检查此权限。

`greengrass:CreateCertificate`

Greengrass 核心设备创建自己的服务器证书的权限。

当 Greengrass 核心设备创建证书时，将检查此权限。Greengrass 核心设备尝试在第一次运行时、当核心的连接信息更改时，以及在指定的轮换周期内创建服务器证书。

`greengrass:GetConnectivityInfo`

Greengrass 核心设备检索自己的连接信息的权限。

当 Greengrass 核心设备尝试从 AWS IoT Core 检索其连接信息时，将检查此权限。

`greengrass:GetDeployment`

Greengrass 核心设备检索部署的权限。

当 Greengrass 核心设备尝试从云中检索部署和部署状态时，将检查此权限。

`greengrass:GetDeploymentArtifacts`

Greengrass 核心设备检索部署构件（如组信息或 Lambda 函数）的权限。

当 Greengrass 核心设备接收部署，然后尝试检索部署构件时，将检查此权限。

`greengrass:UpdateConnectivityInfo`

Greengrass 核心设备使用 IP 或主机名信息更新自己的连接信息的权限。

当 Greengrass 核心设备尝试更新云中的连接信息时，将检查此权限。

`greengrass:UpdateCoreDeploymentStatus`

Greengrass 核心设备更新部署状态的权限。

当 Greengrass 核心设备接收部署，然后尝试更新部署状态时，将检查此权限。

Greengrass 设备操作

AWS IoT Greengrass 定义客户端设备可以在 AWS IoT 策略中使用的以下策略操作：

`greengrass:Discover`

客户端设备使用发现 API 检索其组的核心连接信息和组证书颁发机构的权限。

当客户端设备通过 TLS 双向身份验证调用发现 API 时，将检查此权限。

AWS IoT Greengrass 核心设备的最低 AWS IoT 策略

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）。要支持在 AWS IoT Core、Lambda 函数、连接器和 Greengrass 组中的客户端设备之间进行信息交换，请指定要允许的主题和主题筛选条件。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。
- 该策略包含一个部分，该部分允许 AWS IoT Core 获取、更新和删除核心设备的阴影。要允许对 Greengrass 组中的客户端设备进行影子同步，请在 `Resource` 列表中指定目标 Amazon 资源名称 (ARN)（例如 `arn:aws:iot:region:account-id:thing/device-name`）。
- 不支持在核心设备的 AWS IoT 策略中使用事物策略变量 (`iot:Connection.Thing.*`)。核心使用同一个设备证书与 AWS IoT Core 建立多个连接，但是连接中的客户端 ID 可能与核心事物名称不完全匹配。
- 对于 `greengrass:UpdateCoreDeploymentStatus` 权限，`Resource` ARN 中的最终段是核心设备的 URL 编码的 ARN。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:client/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
```

```
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:AssumeRoleForGroup",
                "greengrass:CreateCertificate"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetDeployment"
            ],
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetDeploymentArtifacts"
            ],
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*"
            ]
```

```
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:UpdateCoreDeploymentStatus"
            ],
            "Resource": [
                "arn:aws:greengrass:region:account-id:/greengrass/groups/group-id/
deployments/*/cores/arn%3Aaws%3Aiot%3Aregion%3Aaccount-id%3Athing%2Fcore-name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        }
    ]
}
```

Note

客户端设备的 AWS IoT 策略通常需要 `iot:Connect`、`iot:Publish`、`iot:Receive` 和 `iot:Subscribe` 操作的类似权限。

要允许客户端设备自动检测设备所属 Greengrass 组中核心的连接信息，客户端设备的 AWS IoT 策略必须包含 `greengrass:Discover` 操作。在 `Resource` 部分中，指定客户端设备的 ARN，而不是 Greengrass 核心设备的 ARN。例如：

```
{
    "Effect": "Allow",
    "Action": [
        "greengrass:Discover"
    ],
    "Resource": [
        "arn:aws:iot:region:account-id:thing/device-name"
    ]
}
```

客户端设备的 AWS IoT 策略通常不需要 `iot:GetThingShadow`、`iot:UpdateThingShadow` 或 `iot:DeleteThingShadow` 操作的权限，因为 Greengrass 核心会处理客户端设备的影子同步操作。在这种情况下，请确保核心的 AWS IoT 策略中针对影子操作的 `Resource` 部分包含客户端设备的 ARN。

在 AWS IoT 控制台中，您可以查看和编辑附加到核心证书的策略。

1. 在导航窗格中中的管理下，展开所有设备，然后选择事物。
2. 选择您的核心。
3. 在核心的配置页面上，选择证书选项卡。
4. 在 证书 选项卡上，选择您的证书。
5. 在证书的配置页面上，选择 Policies (策略)，然后选择该策略。

   如果您要编辑策略，请选择编辑活动版本。
6. 查看策略，并根据需要添加、删除或编辑权限。
7. 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。
8. 选择另存为新版本。

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 AWS IoT Greengrass 的访问权限之前，您应该了解可与 AWS IoT Greengrass 一起使用的 IAM 功能。

IAM 特征	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

IAM 特征	Greengrass 支持吗？
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。策略操作通常与相关联的 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关联操作的权限。

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 `ListGroups` API 操作列出其 AWS 账户 中的组，您可以将 `greengrass:ListGroups` 操作包含在其策略中。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，请在方括号 (`[ ]`) 中列出这些操作，并用逗号将它们分隔开，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”。

要查看 AWS IoT Greengrass 操作完整列表，请参阅 IAM 用户指南中的 AWS IoT Greengrass 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 AWS IoT Greengrass 操作支持的资源级权限的映射，请参阅《IAM 用户指南》中的 AWS IoT Greengrass 定义的操作。

资源	ARN
Group	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}`
GroupVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/versions/${VersionId}`

资源	ARN
CertificateAuthority	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/certificateauthorities/${CertificateAuthorityId}
Deployment	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/deployments/${DeploymentId}
BulkDeployment	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/bulk/deployments/${BulkDeploymentId}
ConnectorDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}
ConnectorDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}/versions/${VersionId}
CoreDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}
CoreDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}/versions/${VersionId}
DeviceDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}
DeviceDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}/versions/${VersionId}
FunctionDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}

资源	ARN
FunctionDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}/versions/${VersionId}
LoggerDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}
LoggerDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}/versions/${VersionId}
ResourceDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}
ResourceDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}/versions/${VersionId}
SubscriptionDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}
SubscriptionDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}/versions/${VersionId}
ConnectivityInfo	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/things/${ThingName}/connectivityInfo

以下示例 Resource 元素指定 AWS 账户 123456789012 中美国西部（俄勒冈州）区域中组的 ARN：

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

或者，要指定属于特定 AWS 区域 中某个 AWS 账户 的所有组，请使用通配符代替组 ID：

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/*"
```

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，请在方括号 ([ ]) 中列出这些资源 ARN，然后使用逗号将它们分隔开，如下所示：

```
"Resource": [
    "resource-arn1",
    "resource-arn2",
    "resource-arn3"
]
```

有关 ARN 格式的更多信息，请参阅 Amazon Web Services 一般参考 中的 Amazon 资源名称（ARN）和 AWS 服务命名空间。

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，您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 元素中指定多个密钥，则 AWS 使用逻辑 AND 运算评估它们。如果您要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。

您也可以在指定条件时使用占位符变量。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 策略元素：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。

AWS IoT Greengrass 支持以下全局条件键。

键	描述
aws:CurrentTime	检查当前日期和时间的日期/时间条件以筛选访问。
aws:EpochTime	检查当前日期和时间（纪元或 Unix 时间）的日期/时间条件以筛选访问。
aws:MultiFactorAuthAge	检查使用 Multi-Factor Authentication (MFA) 颁发请求中的 MFA 验证的安全凭证之前经过的时间（以秒为单位）以筛选访问。
aws:MultiFactorAuthPresent	检查是否使用 Multi-Factor Authentication (MFA) 验证发出当前请求的临时安全凭证以筛选访问。
aws:RequestTag/${TagKey}	根据每个必需标签的允许值集筛选创建请求。
aws:ResourceTag/${TagKey}	根据与资源关联的标签值筛选操作。
aws:SecureTransport	检查是否使用 SSL 发送请求以筛选访问。
aws:TagKeys	根据在请求中是否具有必需标签以筛选创建请求。
aws:UserAgent	按请求者的客户端应用程序筛选访问。

有关更多信息，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，您需要使用 `aws:ResourceTag/${TagKey}`、`aws:RequestTag/${TagKey}` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。有关更多信息，请参阅 标记您的 Greengrass 资源。

适用于 AWS IoT Greengrass 的 IAM 角色

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

临时凭证可用于进行联合身份登录，代入 IAM 角色或代入跨账户角色。您可以调用 AWS STS API 操作（如AssumeRole 或 GetFederationToken ）以获取临时安全凭证。

在 Greengrass 核心上，组角色的临时凭证可供用户定义的 Lambda 函数和连接器使用。如果您的 Lambda 函数使用 AWS 开发工具包，您无需添加逻辑即可获取凭证，因为 AWS 开发工具包会为您执行此操作。

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分派服务角色。此角色允许服务访问其他服务中的资源以代表您完成操作。服务角色显示在 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 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 the section called “Greengrass 服务角色”。

在 AWS IoT Greengrass 控制台中选择 IAM 角色

在 AWS IoT Greengrass 控制台中，您可能需要从账户中的 IAM 角色列表中选择 Greengrass 服务角色或 Greengrass 组角色。

- Greengrass 服务角色允许 AWS IoT Greengrass 代表您访问您在其他服务中的 AWS 资源。通常，您不需要选择服务角色，因为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 “Greengrass 服务角色”。

- Greengrass 组角色用于允许组中的 Greengrass Lambda 函数和连接器访问您的 AWS 资源。它还可以授予将流导出到AWS服务和写入 CloudWatch 日志的AWS IoT Greengrass权限。有关更多信息，请参阅 the section called “Greengrass 组角色”。

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。这使得 AWS IoT Greengrass 可以执行关键任务，如检索您的 AWS Lambda 函数和管理 AWS IoT 影子。

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，该策略定义您使用的 AWS IoT Greengrass 功能的等效权限。此策略由 AWS 维护，定义一组 AWS IoT Greengrass 用于访问您的 AWS 资源的权限。

您可以在 AWS 区域 中重复使用同一 Greengrass 服务角色，但必须将其与您在其中使用 AWS IoT Greengrass 的每个 AWS 区域 中的账户相关联。如果当前 AWS 账户 和区域中不存在该服务角色，则组部署失败。

以下部分介绍了如何在 AWS Management Console 或 AWS CLI 中创建和管理 Greengrass 服务角色。

- 管理服务角色（控制台）
- 管理服务角色 (CLI)

> **Note**
>
> 除了授权服务级别访问权限的服务角色外，您还可以为 AWS IoT Greengrass 组分配一个组角色。组角色是一个单独的 IAM 角色，可控制组中的 Greengrass Lambda 函数和连接器访问 AWS 服务的方式。

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，当您创建或部署 Greengrass 组时，控制台会检查您的 AWS 账户 是否附加到您当前在控制台中选择的 AWS 区域 中的 Greengrass 服务角色。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 AWS IoT 控制台执行以下角色管理任务：

- 查找您的 Greengrass 服务角色
- 创建 Greengrass 服务角色
- 更改 Greengrass 服务角色
- 移除 Greengrass 服务角色

> **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 AWS IoT Greengrass 在当前 AWS 区域 中使用的服务角色。

1. 从AWS IoT控制台导航窗格中，选择设置。
2. 滚动到 Greengrass 服务角色部分以查看您的服务角色及其策略。

   如果没有看到服务角色，可以让控制台为您创建或配置一个。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	Value
名称	`Greengrass_ServiceRole`
可信任的实体	`AWS service: greengrass`
策略	AWSGreengrassResourceAccessRolePolicy

> **Note**
>
> 如果 Greengrass 设备安装程序创建服务角色，则角色名称为 `GreengrassServiceRole_random-string`。

当您从 AWS IoT 控制台创建或部署 Greengrass 组时，控制台会检查 Greengrass 服务角色是否与您当前在控制台中选择的 AWS 区域 中的 AWS 账户 相关联。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。
>
> 如果创建了服务角色，请返回到 AWS IoT 控制台并将该角色附加到组。可以在组的设置页面上的 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. 从AWS IoT控制台导航窗格中，选择设置。

2. 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。

3. 选择要附加的 Greengrass 服务角色。

4. 选择附加角色。

> **Note**
>
> 要允许控制台为您创建默认 Greengrass 服务角色，请选择 Create role for me (为我创建角色)，而不是从列表中选择一个角色。如果您的 AWS 账户 角色中包含名为 `Greengrass_ServiceRole` 的角色，则不会显示为我创建角色链接。

移除 Greengrass 服务角色（控制台）

使用以下过程从控制台当前所选 AWS 区域 中您的 AWS 账户 移除 Greengrass 服务角色。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. 从AWS IoT控制台导航窗格中，选择设置。

2. 在 Greengrass 服务角色 下，选择 移除角色。

3. 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)

在以下过程中，我们假定 AWS CLI 已安装并配置为使用您的 AWS 账户 ID。有关更多信息，请参阅AWS Command Line Interface 用户指南中的安装 AWS 命令行界面和配置 AWS CLI。

您可以使用 AWS CLI 执行以下角色管理任务：

- 获取您的 Greengrass 服务角色
- 创建 Greengrass 服务角色
- 删除 Greengrass 服务角色

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 Greengrass get-service-role-for-account --region region
  ```

  如果 Greengrass 服务角色已与您的账户关联，则将返回以下角色元数据。

  ```
  {
    "AssociatedAt": "timestamp",
    "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
  }
  ```

  如果未返回任何角色元数据，则您必须创建服务角色（如果该角色不存在）并将其与您在 AWS 区域 中的账户关联。

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该角色。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, macOS, or Unix

   ```
   aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
   document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "greengrass.amazonaws.com"
         },
         "Action": "sts:AssumeRole",
         "Condition": {
           "StringEquals": {
             "aws:SourceAccount": "account-id"
           },
           "ArnLike": {
             "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
           }
         }
       }
     ]
   }'
   ```

   Windows command prompt

   ```
   aws iam create-role --role-name Greengrass_ServiceRole --assume-role-
   policy-document "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect
   \":\"Allow\",\"Principal\":{\"Service\":\"greengrass.amazonaws.com\"},
   \"Action\":\"sts:AssumeRole\",\"Condition\":{\"ArnLike\":{\"aws:SourceArn
   ```

```
\":\"arn:aws:greengrass:region:account-id:*\"},\"StringEquals\":
{\"aws:SourceAccount\":\"account-id\"}}}]}"
```

2. 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 greengrass associate-service-role-to-account --role-arn role-arn --
region region
```

如果成功，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 greengrass 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-account
  - disassociate-service-role-from-account
  - get-service-role-for-account
- AWS CLI 命令参考中的 IAM 命令
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

Greengrass 组角色

Greengrass 组角色是授权在 Greengrass 核心上运行的代码访问您的 AWS 资源的 IAM 角色。您可以在 AWS Identity and Access Management (IAM) 中创建此角色和管理权限，并将该角色附加到您的 Greengrass 组。一个 Greengrass 组有一个组角色。要添加或更改权限，您可以附加其他角色或更改附加到该角色的 IAM policies。

角色必须将 AWS IoT Greengrass 定义为受信任的实体。根据您的业务用例，组角色可能包含定义以下内容的 IAM 策略：

- 用户定义的 Lambda 函数 访问 AWS 服务的权限。

- 连接器 访问 AWS 服务的权限。
- 流管理器将流导出到 AWS IoT Analytics 和 Kinesis Data Streams 的权限。
- 允许 CloudWatch 日志记录的权限。

以下各部分将介绍如何在 AWS Management Console 或 AWS CLI 中附加或移除 Greengrass 组角色。

- 管理组角色（控制台）
- 管理组角色 (CLI)

> **Note**
>
> 除了授权从 Greengrass 核心进行访问的群组角色外，您还可以分配一个 Greengrass 服务角色以允许 AWS IoT Greengrass 代表您访问 AWS 资源。

管理 Greengrass 组角色（控制台）

您可以使用 AWS IoT 控制台执行以下角色管理任务：

- 查找您的 Greengrass 组角色
- 添加或更改 Greengrass 组角色
- 移除 Greengrass 组角色

> **Note**
>
> 登录到控制台的用户必须具有管理该角色的权限。

查找您的 Greengrass 组角色（控制台）

请按照以下步骤查找附加到 Greengrass 组的角色。

1. 在 AWS IoT 控制台导航窗格的管理下，展开Greengrass 设备，然后选择组 (V1)。

2. 选择目标组。
3. 在组配置页面上，选择查看设置。

如果某个角色已附加到该组，它会显示在组角色)下。

添加或更改 Greengrass 组角色（控制台）

请按照以下步骤从您的 AWS 账户 中选择要添加到 Greengrass 组的 IAM 角色。

组角色具有以下要求：

- 将 AWS IoT Greengrass 定义为可信任的实体。
- 附加到角色的权限策略必须授予组中的 Lambda 函数和连接器，以及 Greengrass 系统组件所需的对 AWS 资源的权限。

> **Note**
>
> 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。条件上下文密钥限制访问权限，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。

使用 IAM 控制台创建和配置此角色及其权限。有关创建允许访问 Amazon DynamoDB 表的示例角色的步骤，请参阅the section called “配置组角色”。有关常见步骤，请参阅 IAM 用户指南中的为 AWS 服务（控制台）创建一个角色。

配置角色后，请使用 AWS IoT 控制台将角色添加到组。

> **Note**
>
> 仅在为组选择角色时才需要执行此过程。更改当前所选组角色的权限后，不需要执行此过程。

1. 在 AWS IoT 控制台导航窗格的管理下，展开Greengrass 设备，然后选择组 (V1)。

2. 选择目标组。
3. 在组配置页面上，选择查看设置。
4. 在组角色下，选择添加或更改角色：
   - 要添加角色，请选择关联角色，然后从角色列表中选择您的角色。这些是您的 AWS 账户 中将 AWS IoT Greengrass 定义为受信任实体的角色。
   - 要选择其他角色，请选择编辑角色，然后从角色列表中选择您的角色。
5. 选择 Save（保存）。

移除 Greengrass 组角色（控制台）

请按照以下步骤从 Greengrass 组中移除角色。

1. 在 AWS IoT 控制台导航窗格的管理下，展开Greengrass 设备，然后选择组 (V1)。
2. 选择目标组。
3. 在组配置页面上，选择查看设置。
4. 在组角色下，选择取消关联角色。
5. 在确认对话框中，选择取消关联角色。此步骤将从组中移除相应角色，但不真正删除该角色。如果要删除角色，请使用 IAM 控制台。

管理 Greengrass 组角色 (CLI)

您可以使用 AWS CLI 执行以下角色管理任务：

- 获取 Greengrass 组角色
- 创建 Greengrass 组角色
- 移除 Greengrass 组角色

获取 Greengrass 组角色 (CLI)

请按照以下步骤查看 Greengrass 组是否具有关联的角色。

1. 从组列表中获取目标组的 ID。

```
aws greengrass list-groups
```

以下为 `list-groups` 响应示例。响应中的每个组都包括一个包含该组 ID 的 `Id` 属性。

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

有关更多信息，包括使用 `query` 选项筛选结果的示例，请参阅the section called “获取组 ID”。

2. 从输出中复制目标组的 `Id`。
3. 获取组角色。将 *`group-id`* 替换为目标组的 ID。

```
aws greengrass get-associated-role --group-id group-id
```

如果某个角色与您的 Greengrass 组关联，将返回以下角色元数据。

```
{
  "AssociatedAt": "timestamp",
  "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

如果您的组没有关联的角色，将返回以下错误。

```
An error occurred (404) when calling the GetAssociatedRole operation: You need to
 attach an IAM role to this deployment group.
```

创建 Greengrass 组角色 (CLI)

请按照以下步骤创建角色并将其与 Greengrass 组关联。

要使用 IAM 创建组角色：

1. 使用允许 AWS IoT Greengrass 代入该角色的信任策略创建该角色。此示例将创建一个名为 `MyGreengrassGroupRole` 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。条件上下文密钥限制访问权限，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, macOS, or Unix

   ```
   aws iam create-role --role-name MyGreengrassGroupRole --assume-role-policy-
   document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "greengrass.amazonaws.com"
         },
         "Action": "sts:AssumeRole",
         "Condition": {
           "StringEquals": {
   ```

```
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:/greengrass/
groups/group-id"
        }
      }
    }
  ]
}'
```

Windows command prompt

```
aws iam create-role --role-name MyGreengrassGroupRole --assume-role-
policy-document "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect
\":\"Allow\",\"Principal\":{\"Service\":\"greengrass.amazonaws.com\"},
\"Action\":\"sts:AssumeRole\",\"Condition\":{\"ArnLike\":{\"aws:SourceArn
\":\"arn:aws:greengrass:region:account-id:/greengrass/groups/group-id\"},
\"StringEquals\":{\"aws:SourceAccount\":\"account-id\"}}}]}"
```

2. 从输出中的角色元数据复制角色 ARN。使用该 ARN 将角色与您的组关联。
3. 将托管策略或内联策略附加到角色以支持您的业务案例。例如，如果用户定义的 Lambda 函数从 Amazon S3 中读取数据，可以将 `AmazonS3ReadOnlyAccess` 托管策略附加到该角色。

```
aws iam attach-role-policy --role-name MyGreengrassGroupRole --policy-arn
 arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

如果成功，将不返回任何响应。

要将角色与您的 Greengrass 组关联，请执行以下操作：

1. 从组列表中获取目标组的 ID。

```
aws greengrass list-groups
```

以下为 `list-groups` 响应示例。响应中的每个组都包括一个包含该组 ID 的 `Id` 属性。

```
{
```

```
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

有关更多信息，包括使用 query 选项筛选结果的示例，请参阅the section called “获取组 ID”。

2. 从输出中复制目标组的 Id。
3. 将角色与您的组关联。将 *group-id* 替换为目标组的 ID，将 *role-arn* 替换为组角色的 ARN。

```
aws greengrass associate-role-to-group --group-id group-id --role-arn role-arn
```

如果成功，将返回以下响应。

```
{
  "AssociatedAt": "timestamp"
}
```

移除 Greengrass 组角色 (CLI)

请按照以下步骤解除组角色与您的 Greengrass 组的关联。

1. 从组列表中获取目标组的 ID。

```
aws greengrass list-groups
```

以下为 `list-groups` 响应示例。响应中的每个组都包括一个包含该组 ID 的 `Id` 属性。

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

有关更多信息，包括使用 query 选项筛选结果的示例，请参阅the section called “获取组 ID”。

2. 从输出中复制目标组的 Id。
3. 解除该角色与您的组的关联。将 *group-id* 替换为目标组的 ID。

```
aws greengrass disassociate-role-from-group --group-id group-id
```

如果成功，将返回以下响应。

```
{
  "DisassociatedAt": "timestamp"
}
```

> **Note**
>
> 如果未使用组角色，可以将其删除。先使用 delete-role-policy 从角色中移除每个托管的策略，然后使用 delete-role 删除该角色。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 用户指南中的相关主题
  - 创建向 AWS 服务委派权限的角色
  - 修改角色
  - 添加和删除 IAM 身份权限
  - 删除角色或实例配置文件
- AWS CLI 命令参考中的 AWS IoT Greengrass 命令
  - list-groups
  - associate-role-to-group
  - disassociate-role-from-group
  - get-associated-role
- AWS CLI 命令参考中的 IAM 命令
  - attach-role-policy
  - create-role

- delete-role
- delete-role-policy

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。

`aws:SourceArn` 的值必须是与 `sts:AssumeRole` 请求关联的 Greengrass 客户资源。

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:`*`region`*`:`*`account-id`*`:*`。

有关使用 `aws:SourceArn` 和 `aws:SourceAccount` 全局条件上下文键的策略示例，请参阅以下主题：

- 创建 Greengrass 服务角色
- 创建 Greengrass 组角色
- 创建并配置 IAM 执行角色用于批量部署

适用于 AWS IoT Greengrass 的基于身份的策略示例

默认情况下，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 托管式策略 或 工作职能的 AWS 托管式策略。
- 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。
- Require multi-factor authentication（MFA）[需要多重身份验证（MFA）] – 如果您所处的场景要求您的 AWS 账户 中有 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。

AWS适用于 AWS IoT Greengrass 的托管策略

AWS IoT Greengrass 会维护以下可用于向 IAM 用户和角色授予权限的 AWS 托管策略。

策略	描述
AWSGreengrassFullAccess	允许对所有 AWS 资源执行所有 AWS IoT Greengrass 操作。建议 AWS IoT Greengrass 服务管理员或出于测试目的使用此策略。

策略	描述
AWSGreengrassReadOnlyAccess	允许对所有 AWS 资源执行 `List` 和 `Get` AWS IoT Greengrass 操作。
AWSGreengrassResourceAccessRolePolicy	允许从包括 AWS Lambda 和 AWS IoT 设备影子在内的 AWS 服务访问资源。这是用于 Greengrass 服务角色的默认策略。*本策略旨在提供一般的访问便捷性。您可以定义限制性更强的自定义策略。
GreengrassOTAUpdateArtifactAccess	允许对所有 AWS 区域 中适用于 AWS IoT Greengrass Core 软件的空中下载（OTA）更新构件进行只读访问。

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": "*"
        }
    ]
}
```

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
- 错误：未授权 Greengrass 担任与该账户相关的服务角色，或错误：失败：TES 服务角色未与此账户关联。
- 错误：在试图使用角色 arn:aws:iam::<account-id>:role/<role-name> 访问 s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz 时权限被拒绝。
- 设备影子未与云同步。
- 未授权我执行 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，请参阅排查问题。

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。

当 `mateojackson` IAM用户尝试查看有关核心定义版本的详细信息，但不具备 `greengrass:GetCoreDefinitionVersion` 权限时，会发生以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 greengrass:GetCoreDefinitionVersion on resource: resource: arn:aws:greengrass:us-
west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/
versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE
```

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:GetCoreDefinitionVersion` 操作访问 `arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE` 资源。

错误：未授权 Greengrass 担任与该账户相关的服务角色，或错误：失败：TES 服务角色未与此账户关联。

解决方案：当部署失败时，您可能会看到此错误。检查 Greengrass 服务角色是否与您在当前 AWS 区域 中的 AWS 账户 相关联。有关更多信息，请参阅 the section called “管理服务角色 (CLI)” 或 the section called “管理服务角色（控制台）”。

错误：在试图使用角色 arn:aws:iam::<account-id>:role/<role-name> 访问 s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz 时权限被拒绝。

解决方案：当无线 (OTA) 更新失败时，您可能会看到此错误。在签署人角色策略中，将目标 AWS 区域 添加为 `Resource`。此签署人角色用于为 AWS IoT Greengrass 软件更新的 S3 URL 进行预签名。有关更多信息，请参阅 S3 URL 签署人角色。

设备影子未与云同步。

解决方案：确保 AWS IoT Greengrass 在 Greengrass 服务角色中具有 `iot:UpdateThingShadow` 和 `iot:GetThingShadow` 操作的权限。如果服务角色使用 `AWSGreengrassResourceAccessRolePolicy` 托管策略，则在默认情况下包含这些权限。

请参阅影子同步超时问题排查。

以下是您在使用 AWS IoT Greengrass 时可能会遇到的一般 IAM 问题。

未授权我执行 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以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《IAM 用户指南》中的在您拥有的其他 AWS 账户 中向 IAM 用户提供访问权限和向第三方拥有的 Amazon Web Services 账户提供访问权限

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。

AWS IoT Greengrass的合规性验证

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全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，并提供了在这些基础上 AWS 部署以安全性和合规性为重点的基准环境的步骤。
- 在 A@@ mazon Web Services 上构建 HIPAA 安全与合规架构 — 本白皮书描述了各公司如何使用 AWS 来创建符合 HIPAA 资格的应用程序。

> **Note**
>
> 并非所有 AWS 服务 人都符合 HIPAA 资格。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS 合规资源AWS — 此工作簿和指南集可能适用于您所在的行业和所在地区。
- 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的最佳实践， AWS 服务并将指南映射到跨多个框架（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)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业指导方针和法规。
- AWS Security Hub— 这 AWS 服务 提供了您内部安全状态的全面视图 AWS。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践。有关受支持服务及控件的列表，请参阅 Security Hub 控件参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

AWS IoT Greengrass 中的故障恢复能力

AWS 全球基础设施是围绕 Amazon Web Services 区域和可用区而构建的。每个AWS 区域 提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、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。

有关 Amazon Web Services 区域和可用区的更多信息，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。

- 如果核心失去互联网连接，客户端设备可以继续通过本地网络进行通信。
- 您可以将核心配置为将要发到 AWS Cloud 目标的未处理消息存储在本地存储缓存中，而不是存储在内存存储中。本地存储缓存可以在核心重新启动期间（例如，在组部署或设备重新启动后）保持存储，以便 AWS IoT Greengrass 可以继续处理要发到 AWS IoT Core 的消息。有关更多信息，请参阅the section called “MQTT 消息队列”。
- 您可以将核心配置为与 AWS IoT Core 消息代理建立持久会话。这将允许核心接收在其脱机时发送的消息。有关更多信息，请参阅the section called “与 AWS IoT Core 的 MQTT 持久性会话”。
- 您可以将 Greengrass 组配置为将日志写入本地文件系统和 CloudWatch Logs。如果核心失去连接，可以继续本地日志记录，但发送 CloudWatch 日志的重试次数有限。在重试次数用尽后，该事件将被丢弃。您还应该注意日志记录限制。
- 您可以编写 Lambda 函数来读取流管理器流，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 受 AWS 全球网络安全保护。有关 AWS 安全服务以及 AWS 如何保护基础架构的信息，请参阅 AWS 云安全。要按照基础设施安全最佳实践设计您的 AWS 环境，请参阅《安全性支柱 AWS Well-Architected Framework》中的 基础设施保护。

您可以使用 AWS 发布的 API 调用通过网络访问 AWS IoT Greengrass。客户端必须支持以下内容：

- 传输层安全性协议（TLS） 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 具有完全向前保密 (PFS) 的密码套件，例如 DHE（Ephemeral Diffie-Hellman）或 ECDHE（Elliptic Curve Ephemeral Diffie-Hellman）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 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 证书和加密密钥连接到 AWS Cloud 并进行身份验证。有关更多信息，请参阅the section called “设备身份验证和授权”。

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。与 AWS IoT Core 连接时，AWS IoT Greengrass 将生成可预测的客户端 ID，您可以将其与 AWS IoT Device Defender 结合使用。有关更多信息，请参阅 AWS IoT Core 开发人员指南中的 AWS IoT Device Defender。

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 Lambda 函数强制实施网络隔离，无论它们是否在 Greengrass 容器中运行。因此，Lambda 函数可以通过网络与系统内外运行的任何其他进程进行通信。

如果您失去了对 Greengrass 核心设备的控制，并且希望阻止客户端设备将数据传输到核心，请执行以下操作：

1. 从 Greengrass 组中移除 Greengrass 核心。
2. 轮换组 CA 证书。在 AWS IoT 控制台中，您可以在组的 设置 页面上轮换 CA 证书。在 AWS IoT Greengrass API 中，您可以使用CreateGroupCertificateAuthority操作。

   如果您的核心设备的硬盘驱动器容易被盗，我们还建议使用全磁盘加密。

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。您可以使用此端点来管理 AWS IoT Greengrass 服务中的组、Lambda 函数、部署和其他资源。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。

> **Note**
>
> 目前，您无法将 Greengrass 核心设备配置为完全在您的 VPC 内部运行。

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

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 CreateDeployment 和 StartBulkDeployment 等操作。控制平面不包括 GetDeployment 和 Discover 之类的操作，它们属于数据平面操作。
- 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。

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:StartBulkDeployment"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

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

通过在 IAM 角色中使用最低权限集，遵循最低特权原则。在 IAM 策略中限制对 `Action` 和 `Resource` 属性使用 * 通配符。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”。

最低权限最佳实践也适用于您附加到 Greengrass 核心设备和客户端设备的 AWS IoT 策略。

不要在 Lambda 函数中对凭证进行硬编码

不要在用户定义的 Lambda 函数中对凭证进行硬编码。为了更好地保护您的凭证：

- 要与 AWS 服务交互，请为 Greengrass 组角色中的特定操作和资源定义权限。
- 使用本地密钥存储您的凭证。或者，如果函数使用 AWS 开发工具包，请使用默认凭证提供程序链中的凭证。

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)。尽管访问核心设备上的本地日志需要根权限，而访问 CloudWatch 日志需要 IAM 权限，但我们仍建议您实施以下保护措施。

- 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 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：
  - 连接器
  - 内核

- 设备
- 函数
- 组
- 日志记录程序
- 资源（本地、机器学习或密钥）
- 订阅

创建目标订阅

订阅通过定义服务、设备和 Lambda 函数之间的消息交换方式来控制 Greengrass 组中的信息流。为了确保应用程序只能够按预期执行操作，您的订阅应该允许发布者仅向特定主题发送消息，并限制订阅者仅从其功能所需的主题接收消息。

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《AWS IoT Core 开发者指南》中的保持设备时钟同步最佳实践。

使用 Greengrass 核心管理设备身份验证

客户端设备可以运行 FreeRTOS 或使用 AWS IoT 设备开发工具包或 AWS IoT Greengrass 发现 API，以获取用于连接和验证同一 Greengrass 组中核心的发现信息。发现信息包括：

- 与客户端设备位于同一个 Greengrass 组的 Greengrass 核心的连接信息。此信息包括核心设备每个终端节点的主机地址和端口号。
- 用于签署本地 MQTT 服务器证书的组 CA 证书。客户端设备使用组 CA 证书验证核心提供的 MQTT 服务器证书。

以下是客户端设备使用 Greengrass 核心管理双向身份验证的的最佳实践。如果您的核心设备遭到破坏，这些实践可以帮助您降低风险。

验证每个连接的本地 MQTT 服务器证书。

> 客户端设备应该在每次与核心建立连接时验证核心提供的 MQTT 服务器证书。此验证是核心设备与客户端设备之间进行双向身份验证的客户端设备端。客户端设备必须能够检测故障并终止连接。

不要对发现信息进行硬编码。

即使核心使用静态 IP 地址，客户端设备也应该依靠发现操作来获取核心连接信息和组 CA 证书。客户端设备不应该对此发现信息进行硬编码。

定期更新发现信息。

客户端设备应该定期运行发现，以更新核心连接信息和组 CA 证书。我们建议客户端设备在与核心建立连接之前更新此信息。由于发现操作之间较短的持续时间可以最大限度地减少潜在的暴露时间，因此我们建议客户端设备定期断开连接，然后重新连接来触发更新。

如果您失去了对 Greengrass 核心设备的控制，并且希望阻止客户端设备将数据传输到核心，请执行以下操作：

1. 从 Greengrass 组中移除 Greengrass 核心。
2. 轮换组 CA 证书。在 AWS IoT 控制台中，您可以在组的 设置 页面上轮换 CA 证书。在 AWS IoT Greengrass API 中，您可以使用CreateGroupCertificateAuthority操作。

   如果您的核心设备的硬盘驱动器容易被盗，我们还建议使用全磁盘加密。

有关更多信息，请参阅 the section called “设备身份验证和授权”。

另请参阅

- 《AWS IoT开发人员指南》中的 AWS IoT Core 中的安全最佳实践
- AWS 物联网官方博客上的工业物联网解决方案的十大安全黄金法则。

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？

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 – 在账户间共享日志文件，通过将 CloudTrail 日志文件发送到 CloudWatch Logs 来进行实时监控，用 Java 编写日志处理应用程序，验证 CloudTrail 提供的日志文件未发生更改。有关更多信息，请参阅《AWS CloudTrail 用户指南》中的使用 CloudTrail 日志文件。
- Amazon EventBridge– 利用 EventBridge 事件规则获取关于 Greengrass 组部署或使用 CloudTrail 记录的 API 调用的状态更改的通知。有关更多信息，请参阅 the section called “获取部署通知” 或 Amazon EventBridge 用户指南中的什么是 Amazon EventBridge？。
- 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”。
- 本地运行状况检查 - 使用运行状况 API 获取核心设备上本地 AWS IoT Greengrass 进程状态的快照。有关更多信息，请参阅the section called “调用本地运行状况检查 API”。

另请参阅

- the section called “利用 AWS IoT Greengrass 日志进行监控”
- the section called “使用 AWS CloudTrail 记录 AWS IoT Greengrass API 调用”
- the section called “获取部署通知”

利用 AWS IoT Greengrass 日志进行监控

AWS IoT Greengrass 由云服务和 AWS IoT Greengrass 核心软件组成。C AWS IoT Greengrass ore 软件可以将日志写入 Amazon CloudWatch 和核心设备的本地文件系统。在核心上运行的 Lambda 函数和连接器也可以将日志写入 CloudWatch 日志和本地文件系统。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。对日志记录设置所做的更改在部署组后生效。

在组级别配置日志记录。有关说明如何为 Greengrass 组配置日志记录的步骤，请参阅the section called “为 AWS IoT Greengrass 配置日志记录”。

访问 CloudWatch 日志

如果您配置了 CloudWatch 日志记录，则可以在 Amazon CloudWatch 控制台的 “日志” 页面上查看日志。AWS IoT Greengrass 日志的日志组使用以下命名约定：

```
/aws/greengrass/GreengrassSystem/greengrass-system-component-name
/aws/greengrass/Lambda/aws-region/account-id/lambda-function-name
```

每个日志组包含的日志流使用以下命名约定：

```
date/account-id/greengrass-group-id/name-of-core-that-generated-log
```

使用 CloudWatch 日志时，需要考虑以下注意事项：

- 如果没有互联网连接， CloudWatch 日志将发送到日志，重试次数有限。在重试次数用尽后，该事件将被丢弃。
- 事务、内存以及其他限制将适用。有关更多信息，请参阅 the section called “日志记录限制”。
- 你的 Greengrass 群组角色必须AWS IoT Greengrass允许写入日志。 CloudWatch 要授予权限，请在您的组角色中嵌入以下内联策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

> **Note**
>
> 您可以针对您的日志资源授予更细粒度的访问。有关更多信息，请参阅 A mazon CloudWatch 用户指南中的使用基于身份的 CloudWatch 日志策略（IAM 策略）。

组角色是您创建并附加到 Greengrass 组的 IAM 角色。您可以使用控制台或 AWS IoT Greengrass API 管理组角色。

使用控制台

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择目标组。
3. 选择 查看设置。在组角色下，您可以查看、关联或取消关联群组角色。

   有关演示如何附加组角色的步骤，请参阅组角色。

使用 CLI

- 要查找组角色，请使用get-associated-role命令。
- 要附加群组角色，请使用associate-role-to-group命令。
- 要移除组角色，请使用disassociate-role-from-group命令。

要了解如何获取组 ID 以使用这些命令，请参阅the section called “获取组 ID”。

访问文件系统日志

如果您配置文件系统日志记录，则日志文件存储在核心设备上的 *greengrass-root*/ggc/var/log 下。下面是高级别目录结构：

```
greengrass-root/ggc/var/log
    - crash.log
    - system
        - log files for each Greengrass system component
    - user
        - region
            - account-id
                - log files generated by each user-defined Lambda function
            - aws
                - log files generated by each connector
```

> **Note**
>
> 默认情况下，*greengrass-root* 是 /greengrass 目录。如果配置了写入目录，则日志位于该目录中。

在使用文件系统日志时，请注意以下几点：

- 读取文件系统上的 AWS IoT Greengrass 日志需要根权限。
- AWS IoT Greengrass 支持在日志数据的数量接近配置的限制时进行基于大小的轮换和自动清除。
- crash.log 文件仅在文件系统日志中提供。此日志未写入 CloudWatch 日志。
- 磁盘使用率限制将适用。有关更多信息，请参阅 the section called “日志记录限制”。

> **Note**
>
> AWS IoT Greengrass 核心软件 1.0 版本的日志存储在 *greengrass-root*/var/log 目录下。

默认日志记录配置

如果未显式配置日志记录设置，AWS IoT Greengrass 将在第一个组部署后使用以下默认日志记录配置。

AWS IoT Greengrass 系统组件

- Type - `FileSystem`
- 组件 - `GreengrassSystem`
- Level - `INFO`
- 空格 - `128 KB`

用户定义的 Lambda 函数

- Type - `FileSystem`
- 组件 - `Lambda`
- Level - `INFO`
- 空格 - `128 KB`

> **Note**
>
> 在第一次部署前，仅系统组件将日志写入文件系统，因为不会部署任何用户定义的 Lambda 函数。

为 AWS IoT Greengrass 配置日志记录

您可以使用 AWS IoT 控制台或 AWS IoT Greengrass API 配置 AWS IoT Greengrass 日志记录。

> **Note**
>
> AWS IoT Greengrass要允许将日志写入 CloudWatch 日志，您的组角色必须允许所需的 CloudWatch 日志操作。

配置日志记录 (控制台)

您可以在组的 Settings (设置) 页面上配置日志记录。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择要在其中配置日志记录的组。
3. 在组配置页面上，选择日志选项卡。
4. 选择日志记录位置，如下所示：

   - 要配置 CloudWatch 日志记录，要配置CloudWatch 日志，请选择编辑。
   - 要配置文件系统日志记录，请对 Local logs configuration (本地日志配置) 选择 Edit (编辑)。

   您可以为一个位置或两个位置配置日志记录。
5. 在编辑日志配置模式中，选择 Greengrass 系统日志级别或用户 Lambda 函数日志级别。您可以选择一个组件或两个组件。
6. 选择要记录的事件的最低级别。低于此阈值的事件会被过滤掉，也不会进行存储。
7. 选择保存。更改在部署组后生效。

配置日志记录 (API)

您可以使用 AWS IoT Greengrass 记录器 API 以编程方式配置日志记录。例如，使用 `CreateLoggerDefinition` 操作根据 `LoggerDefinitionVersion` 负载创建记录器定义，该定义使用以下语法：

```
{
  "Loggers": [
    {
      "Id": "string",
      "Type": "FileSystem|AWSCloudWatch",
      "Component": "GreengrassSystem|Lambda",
      "Level": "DEBUG|INFO|WARN|ERROR|FATAL",
```

```
      "Space": "integer"
    },
    {
      "Id": "string",
      ...
    }
  ]
}
```

LoggerDefinitionVersion 是一个或多个具有以下属性的 Logger 对象的数组：

Id

记录器的标识符。

Type

日志事件的存储机制。使用AWSCloudWatch时，日志事件将发送到 CloudWatch 日志。使用 FileSystem 时，日志事件将存储在本地文件系统中。

有效值：AWSCloudWatch、FileSystem

Component

日志事件的源。使用 GreengrassSystem 时，将记录来自 Greengrass 系统组件的事件。使用 Lambda 时，将记录来自用户定义的 Lambda 函数的事件。

有效值：GreengrassSystem、Lambda

Level

日志级别的阈值。低于此阈值的日志事件不会筛选出，也不会进行存储。

有效值：DEBUG、INFO (推荐)、WARN、ERROR、FATAL

Space

用于存储日志的最大本地存储量 (以 KB 为单位)。此字段仅在 Type 设置为 FileSystem 时适用。

配置示例

以下 LoggerDefinitionVersion 示例指定的日志记录配置可以：

- 为 AWS IoT Greengrass 系统组件启用文件系统 ERROR 及更高版本日志记录。

- 为用户定义的 Lambda 函数启用文件系统 INFO (及更高版本) 日志记录。
- 为用户定义的 Lambda 函数开启 CloudWatch INFO（及以上）日志记录。

```
{
  "Name": "LoggingExample",
  "InitialVersion": {
    "Loggers": [
      {
        "Id": "1",
        "Component": "GreengrassSystem",
        "Level": "ERROR",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "2",
        "Component": "Lambda",
        "Level": "INFO",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "3",
        "Component": "Lambda",
        "Level": "INFO",
        "Type": "AWSCloudWatch"
      }
    ]
  }
}
```

在创建日志记录器定义版本后，可以使用其版本 ARN 创建组版本，然后再部署组。

日志记录限制

AWS IoT Greengrass 具有以下日志记录限制。

每秒事务数

启用登录到后 CloudWatch ，日志组件会在本地批量记录事件，然后再将事件发送到 CloudWatch，因此您可以以高于每个日志流每秒五个请求的速率进行记录。

内存

如果配置AWS IoT Greengrass为向发送日志， CloudWatch 且 Lambda 函数的日志长时间超过 5 MB/秒，则内部处理管道最终会被填满。理论上，最差的情况为每个 Lambda 函数 6 MB。

时钟偏差

启用登录后 CloudWatch ，日志组件将 CloudWatch使用正常的签名版本 4 签名流程对请求进行签名。如果 AWS IoT Greengrass 核心设备上的系统时间不同步的时长超过 15 分钟，则请求会被拒绝。

磁盘使用量

使用以下公式可计算用于日志记录的最大总磁盘使用量。

```
greengrass-system-component-space * 8    // 7 if automatic IP detection is disabled
  + 128KB                                // the internal log for the local logging
 component
  + lambda-space * lambda-count          // different versions of a Lambda function are
 treated as one
```

其中：

`greengrass-system-component-space`

AWS IoT Greengrass 系统组件日志的最大本地存储量。

`lambda-space`

Lambda 函数日志的最大本地存储量。

`lambda-count`

已部署的 Lambda 函数的数量。

记录丢失

如果您的AWS IoT Greengrass核心设备配置为仅登录 CloudWatch 且没有互联网连接，则无法检索内存中当前的日志。

当 Lambda 函数终止时（例如，在部署期间），将不会写入几秒钟的日志。 CloudWatch

CloudTrail 日志

AWS IoT Greengrass 使用 AWS CloudTrail 运行，后者是在 AWS IoT Greengrass 中记录用户、角色或 AWS 服务所执行操作的服务。有关更多信息，请参阅 the section called “使用 AWS CloudTrail 记录 AWS IoT Greengrass API 调用”。

使用 AWS CloudTrail 记录 AWS IoT Greengrass API 调用

AWS IoT Greengrass与AWS CloudTrail一项服务集成，该服务提供用户、角色或AWS服务在中执行的操作的记录AWS IoT Greengrass。 CloudTrail 将所有 API 调用捕获AWS IoT Greengrass为事件。捕获的调用包含来自 AWS IoT Greengrass 控制台和代码的 AWS IoT Greengrass API 操作调用。如果您创建了跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括的事件AWS IoT Greengrass。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。

要了解更多信息 CloudTrail，请参阅《AWS CloudTrail用户指南》。

AWS IoT Greengrass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查看 CloudTrail 事件。

对于 AWS 账户中的事件的持续记录（包括 AWS IoT Greengrass 的事件），请创建跟踪记录。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有AWS 区域。此跟踪记录在 AWS 分区中记录所有区域中的事件，并将日志文件传送至您指定的 Amazon S3 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：

- Overview for creating a trail
- 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操作均由 API 参考记录 CloudTrail 并记录在 AWS IoT GreengrassAPI 参考中。例如，

对、AssociateServiceRoleToAccountGetGroupVersionGetConnectivityInfo、和CreateFunctionDefinition操作的调用会在 CloudTrail 日志文件中生成条目。

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 AWS Identity and Access Management（IAM）用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其它 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

了解 AWS IoT Greengrass 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 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，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AssociateServiceRoleToAccount操作的 CloudTrail 日志条目。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T17:04:02Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "AssociateServiceRoleToAccount",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "errorCode": "BadRequestException",
    "requestParameters": null,
    "responseElements": {
        "Message": "That role ARN is invalid."
    },
```

```
    "requestID": "a5990ec6-d22e-11e8-8ae5-c7d2eEXAMPLE",
    "eventID": "b9070ce2-0238-451a-a9db-2dbf1EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

以下示例显示了演示该GetGroupVersion操作的 CloudTrail 日志条目。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-10-17T18:14:57Z"
            }
        },
        "invokedBy": "apimanager.amazonaws.com"
    },
    "eventTime": "2018-10-17T18:15:11Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "GetGroupVersion",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "GroupVersionId": "6c477753-dbf2-4cb8-acc3-5ba4eEXAMPLE",
        "GroupId": "90fcf6df-413c-4515-93a8-00056EXAMPLE"
    },
    "responseElements": null,
    "requestID": "95dcffce-d238-11e8-9240-a3993EXAMPLE",
    "eventID": "8a608034-82ed-431b-b5e0-87fbdEXAMPLE",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

以下示例显示了演示该GetConnectivityInfo操作的 CloudTrail 日志条目。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T17:02:12Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "GetConnectivityInfo",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "ThingName": "us-east-1_CIS_1539795000000_"
    },
    "responseElements": null,
    "requestID": "63e3ebe3-d22e-11e8-9ddd-5baf3EXAMPLE",
    "eventID": "db2260d1-a8cc-4a65-b92a-13f65EXAMPLE",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

以下示例显示了演示该CreateFunctionDefinition操作的 CloudTrail 日志条目。

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T18:01:11Z",
    "eventSource": "greengrass.amazonaws.com",
```

```
    "eventName": "CreateFunctionDefinition",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "InitialVersion": "***"
    },
    "responseElements": {
        "CreationTimestamp": "2018-10-17T18:01:11.449Z",
        "LatestVersion": "dae06a61-c32c-41e9-b983-ee5cfEXAMPLE",
        "LatestVersionArn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
definition/functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE/versions/dae06a61-c32c-41e9-
b983-ee5cfEXAMPLE",
        "LastUpdatedTimestamp": "2018-10-17T18:01:11.449Z",
        "Id": "7a94847d-d4d2-406c-9796-a3529EXAMPLE",
        "Arn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/definition/
functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE"
    },
    "requestID": "a17d4b96-d236-11e8-a74e-3db27EXAMPLE",
    "eventID": "bdbf6677-a47a-4c78-b227-c5f64EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


另请参阅

- AWS CloudTrail 用户指南 中的什么是 AWS CloudTrail？
- 使用 Amazon EventBridge 用户指南创建在 AWS API 调用 CloudTrail时触发的 EventBridge 规则
- AWS IoT Greengrass API 参考

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。Greengrass 核心上的遥测代理收集本地遥测数据，并将其发布到 Amazon EventBridge，无需任何客户交互。核心设备会以最大努力将遥测数据发布到 EntBridge。例如，核心设备在离线时可能无法传输遥测数据。

> **Note**
>
> Amazon EventBridge 是一种事件总线服务，您可以用其轻松地将应用程序与来自各种来源的数据相连接，例如 Greengrass 核心设备和部署通知。有关更多信息，请参阅 Amazon EventBridge 用户指南中的什么是 Amazon EventBridge？。

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。

为确保 Greengrass 边缘组件正常运行，AWS IoT Greengrass 会使用这些数据来满足开发和质量改进用途。此功能还有助于为新增和增强的边缘功能提供信息。AWS IoT Greengrass 仅将遥测数据保留最多 7 天。

此功能在 AWS IoT Greengrass Core 软件 v1.11.0 中可用，并且所有 Greengrass 核心（包括现有的核心）都会默认启用此功能。升级到 AWS IoT Greengrass Core 软件 v1.11.0 或更高版本后，您将自动开始接收数据。

有关如何访问或管理已发布遥测数据的信息，请参阅the section called “订阅以接收遥测数据”。

遥测代理收集并发布以下系统指标。

遥测指标

名称	描述	源
`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。	System
`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。	System
`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。	System

名称	描述	源
LambdaOutOfMemory	导致 Lambda 函数用完内存的运行次数。	System
DroppedMessageCount	发往 AWS IoT Core 的已丢弃消息的数量。	GGCloudSpooler 系统组件
LambdaTimeout	运行用户定义的 Lambda 函数的超时次数。	用户定义的 Lambda 函数、AWS Cloud 和系统
LambdaUngracefullyKilled	用户定义的 Lambda 函数未能完成的运行次数。	用户定义的 Lambda 函数、AWS Cloud 和系统
LambdaError	产生用户定义的 Lambda 函数写入错误日志的运行次数。	用户定义的 Lambda 函数、AWS Cloud 和系统
BytesAppended	附加到流管理器的数据的字节数。	GGStreamManager 系统组件
BytesUploadedToIoTAnalytics	流管理器导出到 AWS IoT Analytics 中频道的数据的字节数。	GGStreamManager 系统组件
BytesUploadedToKinesis	流管理器导出到 Amazon Kinesis Data Streams 中流的数据的字节数。	GGStreamManager 系统组件
BytesUploadedToIoTSiteWise	流管理器导出到 AWS IoT SiteWise 中资源属性的数据的字节数。	GGStreamManager 系统组件
BytesUploadedToS3ExportTaskExecutor	流管理器导出到 Amazon S3 中对象的数据的字节数。	GGStreamManager 系统组件
BytesUploadedToHTTP	流管理器导出到 HTTP 的数据的字节数。	GGStreamManager 系统组件

配置遥测设置

Greengrass 遥测使用以下设置：

- 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。

> **Note**
>
> 这些设置不可更改。

您可以为 Greengrass 核心设备启用或禁用遥测功能。AWS IoT Greengrass 将使用影子来管理遥测配置。当核心与 AWS IoT Core 与连接时，您的更改会立即生效。

遥测代理使用服务质量 (QoS) 级别为 0 的 MQTT 协议来发布数据。这意味着它不会确认传输或重试发布操作。遥测消息与其他发往 AWS IoT Core 的订阅消息共享 MQTT 连接。

除数据链路成本外，从核心到 AWS IoT Core 的数据传输是免费的。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。

要求

配置遥测设置时适用以下要求：

- 您必须使用 AWS IoT Greengrass Core v1.11.0 或更高版本。

  > **Note**
  >
  > 如果您运行的是早期版本并且不想使用遥测，您不需要执行任何操作。

- 在更新遥测设置之前，您必须提供 IAM 权限以更新核心（事物）影子和调用配置 API。

  以下示例 IAM 策略允许您管理特定核心的影子和运行时配置：

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

```
            "Sid": "AllowManageShadow",
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow",
                "iot:DescribeThing"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-name-*"
            ]
        },
        {
            "Sid": "AllowManageRuntimeConfig",
            "Effect": "Allow",
            "Action": [
                "greengrass:GetCoreRuntimeConfiguration",
                "greengrass:UpdateCoreRuntimeConfiguration"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-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。

配置遥测设置（控制台）

以下内容显示了如何在 AWS IoT Greengrass 控制台中更新 Greengrass 核心的遥测设置。

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 在 Greengrass 组下，选择您的目标组。
3. 在组配置页面的概述部分中，选择您的 Greengrass 核心。
4. 在核心的配置页面上，选择遥测选项卡。
5. 在系统运行状况遥测部分中，选择配置。
6. 在配置遥测中，选择遥测以启用或禁用遥测状态。

> **Important**
>
> 默认情况下，AWS IoT Greengrass Core 软件 v1.11.0 或更高版本的遥测功能处于启用状态。

这些更改在运行时生效。您不需要部署这个组。

配置遥测设置 (CLI)

在 AWS IoT Greengrass API 中，`TelemetryConfiguration` 对象表示 Greengrass 核心的遥测设置。此对象是与核心关联的 `RuntimeConfiguration` 对象的一部分。您可以使用 AWS IoT Greengrass API、AWS CLI 或 AWS 开发工具包来管理 Greengrass 遥测数据。本节中的示例使用 AWS CLI。

检查遥测设置

以下命令会获取 Greengrass 核心的遥测设置。

- 用目标核心的名称替换 *core-thing-name*。

  要获取事物名称，请使用 get-core-definition-version 命令。该命令返回包含事物名称的事物 ARN。

```
aws greengrass get-thing-runtime-configuration --thing-name core-thing-name
```

该命令返回 JSON 响应中的 `GetCoreRuntimeConfigurationResponse` 对象。例如：

```
{
    "RuntimeConfiguration": {
        "TelemetryConfiguration": {
            "ConfigurationSyncStatus": "OutOfSync",
            "Telemetry": "On"
        }
    }
}
```

配置遥测设置

以下命令会更新 Greengrass 核心的遥测设置。

- 用目标核心的名称替换 *core-thing-name*。

要获取事物名称，请使用 get-core-definition-version 命令。该命令返回包含事物名称的事物 ARN。

JSON expanded

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration  '{
    "RuntimeConfiguration": {
        "TelemetryConfiguration": {
            "ConfigurationSyncStatus": "InSync",
            "Telemetry": "Off"
        }
    }
}
```

JSON single-line

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration "{\"TelemetryConfiguration\":{\"ConfigurationSyncStatus
\":\"InSync\",\"Telemetry\":\"Off\"}}"
```

如果 ConfigurationSyncStatus 是 InSync，则已应用对遥测设置的更改。这些更改在运行时生效。您不需要部署这个组。

TelemetryConfiguration 对象

TelemetryConfiguration 对象具有以下属性：

ConfigurationSyncStatus

检查遥测设置是否同步。您可能无法更改此属性。

类型：字符串

有效值：InSync 或 OutOfSync

Telemetry

开启或关闭遥测。默认为 On。

类型：字符串

有效值：On 或 Off

订阅以接收遥测数据

您可以在 Amazon EventBridge 中创建规则，以定义如何处理从 Greengrass 核心设备发布的遥测数据。EntBridge 接收到数据时，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

部署状态更改的事件（包括遥测数据）采用以下格式：

```
{
    "version": "0",
    "id": "f70f943b-9ae2-e7a5-fec4-4c22178a3e6a",
    "detail-type": "Greengrass Telemetry Data",
    "source": "aws.greengrass",
    "account": "123456789012",
    "time": "2020-07-28T20:45:53Z",
    "region": "us-west-1",
    "resources": [],
    "detail": {
        "ThingName": "CoolThing",
        "Schema": "2020-06-30",
        "ADP": [
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToKinesis",
                        "Sum": 11,
                        "U": "Bytes"
                    }
                ]
            },
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToS3ExportTaskExecutor",
```

```
                    "Sum": 11,
                    "U": "Bytes"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "StreamManager",
            "M": [
                {
                    "N": "BytesAppended|BytesUploadedToHTTP",
                    "Sum": 11,
                    "U": "Bytes"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "StreamManager",
            "M": [
                {
                    "N": "BytesAppended|BytesUploadedToIoTAnalytics",
                    "Sum": 11,
                    "U": "Bytes"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "StreamManager",
            "M": [
                {
                    "N": "BytesAppended|BytesUploadedToIoTSiteWise",
                    "Sum": 11,
                    "U": "Bytes"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
            "M": [
                {
                    "N": "LambdaTimeout",
```

```
                    "Sum": 15,
                    "U": "Count"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "CloudSpooler",
            "M": [
                {
                    "N": "DroppedMessageCount",
                    "Sum": 15,
                    "U": "Count"
                }
            ]
        },
        {
            "TS": 1593727692,
            "NS": "SystemMetrics",
            "M": [
                {
                    "N": "SystemMemUsage",
                    "Sum": 11.23,
                    "U": "Megabytes"
                },
                {
                    "N": "CpuUsage",
                    "Sum": 35.63,
                    "U": "Percent"
                },
                {
                    "N": "TotalNumberOfFDs",
                    "Sum": 416,
                    "U": "Count"
                }
            ]
        },
        {
            "TS": 1593727692,
            "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
            "M": [
                {
                    "N": "LambdaOutOfMemory",
                    "Sum": 12,
```

```
                        "U": "Count"
                    },
                    {
                        "N": "LambdaUngracefullyKilled",
                        "Sum": 100,
                        "U": "Count"
                    },
                    {
                        "N": "LambdaError",
                        "Sum": 7,
                        "U": "Count"
                    }
                ]
            }
        ]
    }
}
```

ADP 数组包含具有以下属性的聚合数据点的列表：

TS

必需。表示数据在何时聚合的时间戳。

NS

必需。系统的命名空间。

M

必需。指标列表。一个指标包含以下属性：

N

指标的名称。

Sum

聚合的指标值。遥测代理会将新值加到之前的总数上，因此总和是一个不断增大的值。您可以使用时间戳来查找特定聚合的值。例如，要查找最新的聚合值，请从最新时间戳的值中减去上一时间戳的值。

U

指标值的单位。

`ThingName`

必需。您的目标事物的名称。

创建 EventBridge 规则的先决条件

在为 AWS IoT Greengrass 创建 EventBridge 规则之前，您应该先执行以下操作：

- 熟悉 EventBridge 中的事件、规则和目标。
- 创建和配置由 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 规则和关联的目标必须位于您在其中创建 Greengrass 资源的 AWS 区域中。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 Amazon EventBridge 用户指南中的什么是 Amazon EventBridge？和 Amazon EventBridge 入门。

创建事件规则以获取遥测数据（控制台）

按照以下步骤，使用 AWS Management Console 来创建 EventBridge 规则，以便接收 Greengrass 核心发布的遥测数据。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 Amazon EventBridge 控制台，并选择 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. 选择时间总线，并对选定事件总线启用规则。
4. 选择规则类型，并选择具有事件模式的规则。
5. 选择 Next（下一步）。
6. 对于 Event source（事件源），选择 AWS 事件或 EventBridge 合作伙伴事件。
7. 在示例事件中，选择 AWS 事件，然后选择 Greengrass 遥测数据。
8. 在事件模式中，请进行下列选择：

   a. 对于 Event source（事件源），选择 AWS services（服务）。

   b. 对于 AWS 服务，选择 Greengrass。

   c. 在事件类型中，选择 Greengrass 遥测数据。

9. 选择 Next（下一步）。
10. 对于 目标，选择 AWS 服务。
11. 在选择目标中，选择 SQS 队列。
12. 在队列中，选择您的函数。

创建事件规则以获取遥测数据 (CLI)

按照以下步骤，使用 AWS CLI 来创建 EventBridge 规则，以便接收 Greengrass 核心发布的遥测数据。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 *thing-name* 替换为核心的事物名称。

     要获取事物名称，请使用 get-core-definition-version 命令。该命令返回包含事物名称的事物 ARN。

   ```
   aws events put-rule \
     --name TestRule \
     --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。

   ```
   aws events put-targets \
     --rule TestRule \
     --targets "Id"="1","Arn"="queue-arn"
   ```

   > **Note**
   >
   > 要允许 Amazon EventBridge 调用您的目标队列，您必须将基于资源的策略添加到您的主题中。有关更多信息，请参阅 Amazon EventBridge 用户指南中的 Amazon SQS 权限。

有关更多信息，请参阅 Amazon EventBridge 用户指南中的 EventBridge 中的事件和事件模式。

AWS IoT Greengrass 遥测故障排除

可以使用以下信息帮助解决配置 AWS IoT Greengrass 遥测的问题。

错误：运行 get-thing-runtime-configuration 命令后，响应中包含 "ConfigurationStatus": "OutOfSync"

解决方案

- AWS IoT 设备影子服务需要时间来处理运行时配置更新，并将更新传输到 Greengrass 核心设备。您可以稍等片刻，然后再检查遥测设置是否同步。
- 确保您的核心设备在线。
- 启用 AWS IoT Core 中的 Amazon CloudWatch Logs 以监控影子。
- 使用 AWS IoT 指标监控事物。

调用本地运行状况检查 API

AWS IoT Greengrass 包含本地 HTTP API，该 API 提供由 AWS IoT Greengrass 启动的本地工作进程的当前状态快照。此快照包括用户定义的 Lambda 函数和系统 Lambda 函数。系统 Lambda 函数是 AWS IoT Greengrass 核心软件的一部分。它们在核心设备上作为本地工作进程运行，并管理消息路由、本地影子同步和自动 IP 地址检测等操作。

运行状况检查 API 支持以下请求：

- 发送 GET 请求以获取所有工作人员的健康信息。
- 发送 POST 请求以获取指定工作人员的健康信息。

请求在设备上本地发送，不需要互联网连接。

获取所有工作人员的健康信息

发送 GET 请求以获取有关所有在职工作人员的健康信息。

- 用 IPC 的端口号替换##。

```
GET http://localhost:port/2016-11-01/health/workers
```

port

IPC 的端口号。

该值可以在 1024 和 65535 之间变化。默认值为 8000。

要更改此端口号，可以更新 config.json 文件中的 ggDaemonPort 属性。有关更多信息，请参阅AWS IoT Greengrass 核心配置文件。

示例请求

以下示例 curl 请求获取所有工作人员的健康信息。

```
curl http://localhost:8000/2016-11-01/health/workers
```

JSON 响应

此请求返回一组工作人员健康信息对象。

响应示例

以下示例响应列出了由 AWS IoT Greengrass 启动的所有工作进程的运行状况信息对象。

```
[
    {
      "FuncArn": "arn:aws:lambda:::function:GGShadowService:1",
      "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
      "ProcessId": "1234",
      "WorkerState": "Waiting"
    },
    {
      "FuncArn": "arn:aws:lambda:::function:GGSecretManager:1",
      "WorkerId": "a9916cc2-1b4d-4f0e-4b12-b1872EXAMPLE",
      "ProcessId": "9798",
      "WorkerState": "Waiting"
    },
    {
      "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3",
      "WorkerId": "2e6f785e-66a5-42c9-67df-42073EXAMPLE",
```

```
      "ProcessId": "11837",
      "WorkerState": "Waiting"
    },
    ...
]
```

获取有关指定工作人员的健康信息

发送 POST 请求以获取有关指定工作人员的健康信息。用 IPC 的端口号替换*##*。默认为 8000。

```
POST http://localhost:port/2016-11-01/health/workers
```

示例请求

以下示例 curl 请求获取指定工作人员的健康信息。

```
curl --data "@body.json" http://localhost:8000/2016-11-01/health/workers
```

以下为 body.json 请求正文示例：

```
{
    "FuncArns": [
        "arn:aws:lambda:::function:GGShadowService:1",
        "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3"
    ]
}
```

请求正文包含一个 FuncArns 阵列。

FuncArns

代表目标工作线程环境的 Lambda 函数的 Amazon 资源名称 (ARN) 列表。

- 对于用户定义的 Lambda 函数，指定当前部署版本的 ARN。如果您使用别名 ARN 将 Lambda 函数添加到群组，则可以使用 GET 请求获取所有工作人员，然后选择要查询的 ARN。
- 对于系统 Lambda 函数，指定相应的 Lambda 函数的 ARN。有关更多信息，请参阅the section called “系统 Lambda 函数”。

类型：字符串数组

最小长度：1

最大长度：由核心设备上 AWS IoT Greengrass 启动的工作人员总数。

JSON 响应

此请求返回一个 `Workers` 阵列和一个 `InvalidArns` 阵列。

`Workers`

指定工作人员的健康信息对象列表。

类型：健康信息对象阵列

`InvalidArns`

无效函数 ARN 的列表，包括没有关联工作人员的函数 ARN。

类型：字符串数组

响应示例

以下示例响应列出了指定工作人员的健康信息对象。

```
{
    "Workers": [
        {
            "FuncArn": "arn:aws:lambda:::function:GGShadowService:1",
            "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
            "ProcessId": "1234",
            "WorkerState": "Waiting"
        },
        {
            "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-
function:3",
            "WorkerId": "2e6f785e-66a5-42c9-67df-42073ESAMPLE",
            "ProcessId": "11837",
            "WorkerState": "Waiting"
        }
    ],
    "InvalidArns" : [
        "some-malformed-arn",
        "arn:aws:lambda:us-west-2:123456789012:function:some-unknown-function:1"
    ]
```

```
}
```

此请求返回以下错误：

400 请求无效

请求正文格式不正确。要解决此问题，请使用以下格式重新发送请求：

```
{"FuncArns":["function-1-arn","function-2-arn"]}
```

400 请求超过了工作人员的最大数量

数组 FuncArns 中指定的 ARN 数量超过了工作人员的数量。

工作人员健康信息

健康信息对象包含以下属性：

FuncArn

代表工作人员的系统 Lambda 函数的 ARN。

类型：string

WorkerId

工作人员的 ID。该属性对于调试很有用。runtime.log 文件和 Lambda 函数日志包含工作程序 ID，因此此属性对于调试启动多个实例的按需 Lambda 函数特别有用。

类型：string

ProcessId

工作人员进程的进程 ID (PID)。

类型：int

WorkerState

工作人员的状态。

类型：string

以下是可能的工件状态。

`Working`

处理消息

`Waiting`

正在等待消息。适用于作为守护程序或独立进程运行的长期存在的 Lambda 函数。

`Starting`

快点，开始吧。

`FailedInitialization`

初始化失败。

`Terminated`

被 Greengrass 守护程序阻止

`NotStarted`

启动失败，正在再次尝试启动。

`Initialized`

已成功初始化。

系统 Lambda 函数

您可以请求以下系统 Lambda 函数的运行状况信息：

`GGCloudSpooler`

管理 AWS IoT Core 作为源或目标的 MQTT 消息队列。

ARN：`arn:aws:lambda:::function:GGCloudSpooler:1`

`GGConnManager`

在 Greengrass 核心设备和客户端设备之间路由 MQTT 消息。

ARN：`arn:aws:lambda:::function:GGConnManager`

`GGDeviceCertificateManager`

监听 AWS IoT 阴影内核 IP 端点的更改，并生成 GGConnManager 用于相互身份验证的服务器端证书。

ARN：`arn:aws:lambda:::function:GGDeviceCertificateManager`

`GGIPDetector`

管理 IP 地址自动检测功能，该功能使 Greengrass 组内设备能够发现 Greengrass 核心设备。当您手动提供 IP 地址时，此服务不适用。

ARN：`arn:aws:lambda:::function:GGIPDetector:1`

`GGSecretManager`

管理本地密钥的安全存储，并通过用户定义的 Lambda 和连接器进行访问。

ARN：`arn:aws:lambda:::function:GGSecretManager:1`

`GGShadowService`

管理客户端设备的本地影子。

ARN：`arn:aws:lambda:::function:GGShadowService`

`GGShadowSyncManager`

如果设备 `syncShadow` 的属性设置为 `true`，则将本地阴影与核心设备和客户端设备的 AWS Cloud 同步。

ARN：`arn:aws:lambda:::function:GGShadowSyncManager`

`GGStreamManager`

在本地处理数据流并自动导出到 AWS Cloud。

ARN：`arn:aws:lambda:::function:GGStreamManager:1`

`GGTES`

本地令牌交换服务，用于检索 Greengrass 组角色中定义的 IAM 凭证，可供本地代码用于访问 AWS 服务。

ARN：`arn:aws:lambda:::function:GGTES`

标记您的 AWS IoT Greengrass 资源

标签可以帮助您组织和管理您的 AWS IoT Greengrass 组。您可以使用标签将元数据分配给组、批量部署以及添加到组的核心、设备和其他资源。也可以在 IAM 策略中使用标签定义对 Greengrass 资源的条件访问。

> **Note**
>
> 目前，AWS IoT 账单组或成本分配报告不支持 Greengrass 资源标签。

标签基本知识

标签可让您按各种标准（例如用途、所有者和环境）对 AWS IoT Greengrass 资源进行分类。当您具有相同类型的许多资源时，可以根据附加到资源的标签来快速识别资源。标签包含您定义的一个键和一个可选值。我们建议您为每个资源类型设计一组标签键。使用一组连续的标签键，管理 资源时会更加轻松。例如，您可以为组定义一组标签来以帮助您跟踪核心设备的出厂位置。有关更多信息，请参阅 AWS 标记策略。

AWS IoT 控制台中的标记支持

您可以在 AWS IoT 控制台中为 Greengrass `Group` 资源创建、查看和管理标签。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《Amazon Web Services 一般参考》中的标签命名和使用惯例。

创建组时分配标签

创建组时，可以将标签分配给组。在标签部分下选择添加标签以显示标记输入字段。

从组配置页面查看和管理标签

您可以选择查看设置，从组配置页面查看和管理标签。在组的标签页面上，选择管理标签，以添加、编辑或删除组标签。

AWS IoT Greengrass API 中的标记支持

您可以使用 AWS IoT Greengrass API 为支持标记的 AWS IoT Greengrass 资源创建、列出和管理标签。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《Amazon Web Services 一般参考》中的标签命名和使用惯例。

- 要在资源创建期间添加标签，请在资源的 tags 属性中定义这些标签。
- 要在创建资源后添加标签或更新标签值，请使用 TagResource 操作。
- 要从资源中删除标签，请使用 UntagResource 操作。
- 要检索与资源关联的标签，请使用 ListTagsForResource 操作或获取资源并检查其 tags 属性。

下表列出了您可以在 AWS IoT Greengrass API 中标记的资源及其相应的 Create 和 Get 操作。

资源	创建	获取
Group	CreateGroup	GetGroup
ConnectorDefinition	CreateConnectorDefinition	GetConnectorDefinition
CoreDefinition	CreateCoreDefinition	GetCoreDefinition
DeviceDefinition	CreateDeviceDefinition	GetDeviceDefinition
FunctionDefinition	CreateFunctionDefinition	GetFunctionDefinition
LoggerDefinition	CreateLoggerDefinition	GetLoggerDefinition
ResourceDefinition	CreateResourceDefinition	GetResourceDefinition
SubscriptionDefinition	CreateSubscriptionDefinition	GetSubscriptionDefinition
BulkDeployment	StartBulkDeployment	GetBulkDeploymentStatus

使用以下操作可为支持标记的资源列出和管理标签：

- TagResource。为资源添加标签。还用于更改标签的键值对中的值。

- ListTagsForResource。列出资源的标签。
- UntagResource。删除资源的标签。

您可以随时在资源中添加或删除标签 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。新值将覆盖旧值。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。

> **Note**
>
> 请勿将资源标签与可分配给 AWS IoT 事物的属性混淆。虽然 Greengrass 核心是 AWS IoT 事物，但本主题中描述的资源标签将附加到 `CoreDefinition` 而不是核心事物。

在 IAM policy 中使用标签

在您的 IAM 策略中，您可以使用资源标签来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。您可以在创建期间标记资源（称作在创建时标记），因此，您稍后无需运行自定义标记脚本。当启动带标签的新环境时，系统会自动应用相应的 IAM 权限。

可以在策略的 `Condition` 元素（也称作 `Condition` 块）中使用以下条件上下文密钥和值。

`greengrass:ResourceTag/`*`tag-key`*`: `*`tag-value`*

允许或拒绝带特定标签的资源上的用户操作。

`aws:RequestTag/`*`tag-key`*`: `*`tag-value`*

要求在发出创建或修改可标记资源的标签的 API 请求时使用（或不使用）特定标签。

`aws:TagKeys: [`*`tag-key`*`, ...]`

要求在发出创建或修改可标记资源的 API 请求时使用（或不使用）一组特定标签键。

只能在对可标记资源执行的 AWS IoT Greengrass 操作中使用的条件上下文密钥和值。这些操作将资源作为必需参数。例如，可以在 `GetGroupVersion` 上设置条件访问。无法在 `AssociateServiceRoleToAccount` 上设置条件访问，因为请求中未引用可标记资源（例如，组、核心定义或设备定义）。

有关更多信息，请参阅《IAM 用户指南》中的使用标签控制访问 和 IAM JSON 策略引用。JSON 策略参考包含 IAM 中的 JSON 策略的元素、变量和评估逻辑的详细语法、描述和示例。

示例 IAM policies

以下示例策略应用基于标签的权限，这些权限仅允许测试用户操作测试资源。

- 第一个语句允许 IAM 用户操作仅具有 env=beta 标签的资源。
- 第二个语句防止 IAM 用户从资源中删除 env=beta 标签。这可以防止用户删除自己的访问权限。

> **Note**
>
> 如果使用标签控制对资源的访问，则还应管理允许用户添加标签或从这些资源中删除标签的权限。否则，在某些情况下，用户将能够通过修改资源标签来绕过您的限制并获得资源访问权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "greengrass:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "greengrass:ResourceTag/env": "beta"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": "greengrass:UntagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/env": "beta"
                }
            }
        }
    ]
```

```
}
```

要允许用户在创建时添加标签，您必须向用户授予适当的权限。以下示例策略包括 `greengrass:TagResource` 和 `greengrass:CreateGroup` 操作的 `"aws:RequestTag/env": "beta"` 条件，这使得用户仅在为具有 env=beta 的组添加标签时能够创建组。这有效地迫使用户为新组添加标签。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "greengrass:TagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "beta"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "greengrass:CreateGroup",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "beta"
                }
            }
        }
    ]
}
```

以下代码段说明了如何通过将标签键括在列表中来为其指定多个标签值：

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

另请参阅

- 《Amazon Web Services 一般参考》中的标记 AWS 资源

对于 AWS IoT Greengrass 的 AWS CloudFormation 支持

AWS CloudFormation 是一项服务，可帮助您创建、管理和复制您的 AWS 资源。您可以使用 AWS CloudFormation 模板来定义要部署的 AWS IoT Greengrass 组和客户端设备、订阅及其他组件。有关示例，请参阅the section called “示例 模板”。

您从模板中生成的资源和基础设施称为堆栈。您可以在一个模板中定义所有资源，也可以引用其他堆栈中的资源。有关 AWS CloudFormation 模板和功能的更多信息，请参阅 AWS CloudFormation 用户指南中的什么是 AWS CloudFormation？。

创建 资源

AWS CloudFormation 模板是描述 AWS 资源的属性和关系的 JSON 或 YAML 文档。支持以下 AWS IoT Greengrass 资源：

- 组
- 内核
- 客户端设备（设备）
- Lambda 函数
- 连接器
- 资源（本地、机器学习和密钥）
- 订阅
- 日志记录程序（日志记录配置）

在 AWS CloudFormation 模板中，Greengrass 资源的结构和语法基于 AWS IoT Greengrass API。例如，示例模板将一个顶级 `DeviceDefinition` 与包含单个客户端设备的 `DeviceDefinitionVersion` 关联。有关更多信息，请参阅 the section called “组对象模型概述”。

《AWS CloudFormation 用户指南》中的 AWS IoT Greengrass资源类型参考介绍了您可以通过 AWS CloudFormation 管理的 Greengrass 资源。当您使用 AWS CloudFormation 模板创建 Greengrass 资源时，我们建议您仅从 AWS CloudFormation 管理这些资源。例如，如果您要添加、更改或删除设备，则应更新您的模板（而不是使用 AWS IoT Greengrass API 或 AWS IoT控制台）。这允许您使用回滚及其他 AWS CloudFormation 更改管理功能。有关使用AWS CloudFormation 创建和管理资源和堆栈的更多信息，请参阅《AWS CloudFormation 用户指南》中的使用堆栈。

有关演示如何在 AWS CloudFormation 模板中创建和部署 AWS IoT Greengrass 资源的演练，请参阅 AWS 物联网官方博客上的使用 AWS CloudFormation 实现 AWS IoT Greengrass 设置自动化。

部署资源

在创建包含组版本的 AWS CloudFormation 堆栈后，您可以使用 AWS CLI 或 AWS IoT控制台部署它。

> **Note**
>
> 要部署组，您必须具有与您的 AWS 账户 关联的 Greengrass 服务角色。该服务角色允许 AWS IoT Greengrass 访问您在 AWS Lambda 和其他 AWS 服务中的资源。如果您已在当前 AWS 区域 中部署了 Greengrass 组，则该角色应存在。有关更多信息，请参阅 the section called “Greengrass 服务角色”。

部署组 (AWS CLI)

- 运行 create-deployment 命令。

```
aws greengrass create-deployment --group-id GroupId --group-version-
id GroupVersionId --deployment-type NewDeployment
```

> **Note**
>
> 示例模板中的 CommandToDeployGroup 语句显示了在创建堆栈时应如何使用您的组和组版本 ID 来输出命令。

部署组（控制台）

1. 在 AWS IoT 控制台导航窗格的管理下，展开 Greengrass 设备，然后选择组 (V1)。
2. 选择您的组。
3. 在组配置页面上，选择部署。

示例 模板

以下示例模板创建一个包含核心、客户端设备、功能、日志记录程序、订阅和两种资源的 Greengrass 组。为此，该模板采用 AWS IoT Greengrass API 的对象模型。例如，要添加到组的客户端设备包含在 `DeviceDefinitionVersion` 资源中，而后者与 `DeviceDefinition` 资源关联。要将设备添加到组，组版本将引用 `DeviceDefinitionVersion` 的 ARN。

该模板包含的参数可让您指定核心和设备的证书 ARN 以及源 Lambda 函数（这是一种 AWS Lambda 资源）的版本 ARN。该模板使用 `Ref` 和 `GetAtt` 内部函数引用创建 Greengrass 资源所需的 ID、ARN 和其他属性。

该模板还定义了两个 AWS IoT 设备（事物），这表示要添加到 Greengrass 组的核心和客户端设备。

在使用您的 Greengrass 资源创建堆栈后，您可以使用 AWS CLI 或 AWS IoT 控制台来部署组。

> **Note**
>
> 该示例中的 `CommandToDeployGroup` 语句说明如何输出可用于部署组的完整 create-deployment CLI 命令。

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "AWS IoT Greengrass example template that creates a group version
 with a core, device, function, logger, subscription, and resources.",
    "Parameters": {
        "CoreCertificateArn": {
            "Type": "String"
        },
        "DeviceCertificateArn": {
            "Type": "String"
        },
        "LambdaVersionArn": {
            "Type": "String"
        }
    },
    "Resources": {
        "TestCore1": {
            "Type": "AWS::IoT::Thing",
            "Properties": {
```

```
                "ThingName": "TestCore1"
            }
        },
        "TestCoreDefinition": {
            "Type": "AWS::Greengrass::CoreDefinition",
            "Properties": {
                "Name": "DemoTestCoreDefinition"
            }
        },
        "TestCoreDefinitionVersion": {
            "Type": "AWS::Greengrass::CoreDefinitionVersion",
            "Properties": {
                "CoreDefinitionId": {
                    "Ref": "TestCoreDefinition"
                },
                "Cores": [
                    {
                        "Id": "TestCore1",
                        "CertificateArn": {
                            "Ref": "CoreCertificateArn"
                        },
                        "SyncShadow": "false",
                        "ThingArn": {
                            "Fn::Join": [
                                ":",
                                [
                                    "arn:aws:iot",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    "thing/TestCore1"
                                ]
                            ]
                        }
                    }
                ]
            }
        },
        "TestClientDevice1": {
            "Type": "AWS::IoT::Thing",
            "Properties": {
```

```
                "ThingName": "TestClientDevice1"
            }
        },
        "TestDeviceDefinition": {
            "Type": "AWS::Greengrass::DeviceDefinition",
            "Properties": {
                "Name": "DemoTestDeviceDefinition"
            }
        },
        "TestDeviceDefinitionVersion": {
            "Type": "AWS::Greengrass::DeviceDefinitionVersion",
            "Properties": {
                "DeviceDefinitionId": {
                    "Fn::GetAtt": [
                        "TestDeviceDefinition",
                        "Id"
                    ]
                },
                "Devices": [
                    {
                        "Id": "TestClientDevice1",
                        "CertificateArn": {
                            "Ref": "DeviceCertificateArn"
                        },
                        "SyncShadow": "true",
                        "ThingArn": {
                            "Fn::Join": [
                                ":",
                                [
                                    "arn:aws:iot",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    "thing/TestClientDevice1"
                                ]
                            ]
                        }
                    }
                ]
            }
        },
```

```
        "TestFunctionDefinition": {
            "Type": "AWS::Greengrass::FunctionDefinition",
            "Properties": {
                "Name": "DemoTestFunctionDefinition"
            }
        },
        "TestFunctionDefinitionVersion": {
            "Type": "AWS::Greengrass::FunctionDefinitionVersion",
            "Properties": {
                "FunctionDefinitionId": {
                    "Fn::GetAtt": [
                        "TestFunctionDefinition",
                        "Id"
                    ]
                },
                "DefaultConfig": {
                    "Execution": {
                        "IsolationMode": "GreengrassContainer"
                    }
                },
                "Functions": [
                    {
                        "Id": "TestLambda1",
                        "FunctionArn": {
                            "Ref": "LambdaVersionArn"
                        },
                        "FunctionConfiguration": {
                            "Pinned": "true",
                            "Executable": "run.exe",
                            "ExecArgs": "argument1",
                            "MemorySize": "512",
                            "Timeout": "2000",
                            "EncodingType": "binary",
                            "Environment": {
                                "Variables": {
                                    "variable1": "value1"
                                },
                                "ResourceAccessPolicies": [
                                    {
                                        "ResourceId": "ResourceId1",
                                        "Permission": "ro"
                                    },
                                    {
                                        "ResourceId": "ResourceId2",
```

```
                                    "Permission": "rw"
                                }
                            ],
                            "AccessSysfs": "false",
                            "Execution": {
                                "IsolationMode": "GreengrassContainer",
                                "RunAs": {
                                    "Uid": "1",
                                    "Gid": "10"
                                }
                            }
                        }
                    }
                }
            ]
        }
    },
    "TestLoggerDefinition": {
        "Type": "AWS::Greengrass::LoggerDefinition",
        "Properties": {
            "Name": "DemoTestLoggerDefinition"
        }
    },
    "TestLoggerDefinitionVersion": {
        "Type": "AWS::Greengrass::LoggerDefinitionVersion",
        "Properties": {
            "LoggerDefinitionId": {
                "Ref": "TestLoggerDefinition"
            },
            "Loggers": [
                {
                    "Id": "TestLogger1",
                    "Type": "AWSCloudWatch",
                    "Component": "GreengrassSystem",
                    "Level": "INFO"
                }
            ]
        }
    },
    "TestResourceDefinition": {
        "Type": "AWS::Greengrass::ResourceDefinition",
        "Properties": {
            "Name": "DemoTestResourceDefinition"
        }
```

```
        },
        "TestResourceDefinitionVersion": {
            "Type": "AWS::Greengrass::ResourceDefinitionVersion",
            "Properties": {
                "ResourceDefinitionId": {
                    "Ref": "TestResourceDefinition"
                },
                "Resources": [
                    {
                        "Id": "ResourceId1",
                        "Name": "LocalDeviceResource",
                        "ResourceDataContainer": {
                            "LocalDeviceResourceData": {
                                "SourcePath": "/dev/TestSourcePath1",
                                "GroupOwnerSetting": {
                                    "AutoAddGroupOwner": "false",
                                    "GroupOwner": "TestOwner"
                                }
                            }
                        }
                    },
                    {
                        "Id": "ResourceId2",
                        "Name": "LocalVolumeResourceData",
                        "ResourceDataContainer": {
                            "LocalVolumeResourceData": {
                                "SourcePath": "/dev/TestSourcePath2",
                                "DestinationPath": "/volumes/TestDestinationPath2",
                                "GroupOwnerSetting": {
                                    "AutoAddGroupOwner": "false",
                                    "GroupOwner": "TestOwner"
                                }
                            }
                        }
                    }
                ]
            }
        },
        "TestSubscriptionDefinition": {
            "Type": "AWS::Greengrass::SubscriptionDefinition",
            "Properties": {
                "Name": "DemoTestSubscriptionDefinition"
            }
        },
```

```
        "TestSubscriptionDefinitionVersion": {
            "Type": "AWS::Greengrass::SubscriptionDefinitionVersion",
            "Properties": {
                "SubscriptionDefinitionId": {
                    "Ref": "TestSubscriptionDefinition"
                },
                "Subscriptions": [
                    {
                        "Id": "TestSubscription1",
                        "Source": {
                            "Fn::Join": [
                                ":",
                                [
                                    "arn:aws:iot",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    "thing/TestClientDevice1"
                                ]
                            ]
                        },
                        "Subject": "TestSubjectUpdated",
                        "Target": {
                            "Ref": "LambdaVersionArn"
                        }
                    }
                ]
            }
        },
        "TestGroup": {
            "Type": "AWS::Greengrass::Group",
            "Properties": {
                "Name": "DemoTestGroupNewName",
                "RoleArn": {
                    "Fn::Join": [
                        ":",
                        [
                            "arn:aws:iam:",
                            {
                                "Ref": "AWS::AccountId"
                            },
```

```
                        "role/TestUser"
                    ]
                ]
            },
            "InitialVersion": {
                "CoreDefinitionVersionArn": {
                    "Ref": "TestCoreDefinitionVersion"
                },
                "DeviceDefinitionVersionArn": {
                    "Ref": "TestDeviceDefinitionVersion"
                },
                "FunctionDefinitionVersionArn": {
                    "Ref": "TestFunctionDefinitionVersion"
                },
                "SubscriptionDefinitionVersionArn": {
                    "Ref": "TestSubscriptionDefinitionVersion"
                },
                "LoggerDefinitionVersionArn": {
                    "Ref": "TestLoggerDefinitionVersion"
                },
                "ResourceDefinitionVersionArn": {
                    "Ref": "TestResourceDefinitionVersion"
                }
            },
            "Tags": {
                "KeyName0": "value",
                "KeyName1": "value",
                "KeyName2": "value"
            }
        }
    },
    "Outputs": {
        "CommandToDeployGroup": {
            "Value": {
                "Fn::Join": [
                    " ",
                    [
                        "groupVersion=$(cut -d'/' -f6 <<<",
                        {
                            "Fn::GetAtt": [
                                "TestGroup",
                                "LatestVersionArn"
                            ]
```

```
                        },
                        ");",
                        "aws --region",
                        {
                            "Ref": "AWS::Region"
                        },
                        "greengrass create-deployment --group-id",
                        {
                            "Ref": "TestGroup"
                        },
                        "--deployment-type NewDeployment --group-version-id",
                        "$groupVersion"
                    ]
                ]
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: >-
  AWS IoT Greengrass example template that creates a group version with a core,
  device, function, logger, subscription, and resources.
Parameters:
  CoreCertificateArn:
    Type: String
  DeviceCertificateArn:
    Type: String
  LambdaVersionArn:
    Type: String
Resources:
  TestCore1:
    Type: 'AWS::IoT::Thing'
    Properties:
      ThingName: TestCore1
  TestCoreDefinition:
    Type: 'AWS::Greengrass::CoreDefinition'
    Properties:
      Name: DemoTestCoreDefinition
  TestCoreDefinitionVersion:
    Type: 'AWS::Greengrass::CoreDefinitionVersion'
```

```
  Properties:
    CoreDefinitionId: !Ref TestCoreDefinition
    Cores:
      - Id: TestCore1
        CertificateArn: !Ref CoreCertificateArn
        SyncShadow: 'false'
        ThingArn: !Join
          - ':'
          - - 'arn:aws:iot'
            - !Ref 'AWS::Region'
            - !Ref 'AWS::AccountId'
            - thing/TestCore1
TestClientDevice1:
  Type: 'AWS::IoT::Thing'
  Properties:
    ThingName: TestClientDevice1
TestDeviceDefinition:
  Type: 'AWS::Greengrass::DeviceDefinition'
  Properties:
    Name: DemoTestDeviceDefinition
TestDeviceDefinitionVersion:
  Type: 'AWS::Greengrass::DeviceDefinitionVersion'
  Properties:
    DeviceDefinitionId: !GetAtt
      - TestDeviceDefinition
      - Id
    Devices:
      - Id: TestClientDevice1
        CertificateArn: !Ref DeviceCertificateArn
        SyncShadow: 'true'
        ThingArn: !Join
          - ':'
          - - 'arn:aws:iot'
            - !Ref 'AWS::Region'
            - !Ref 'AWS::AccountId'
            - thing/TestClientDevice1
TestFunctionDefinition:
  Type: 'AWS::Greengrass::FunctionDefinition'
  Properties:
    Name: DemoTestFunctionDefinition
TestFunctionDefinitionVersion:
  Type: 'AWS::Greengrass::FunctionDefinitionVersion'
  Properties:
    FunctionDefinitionId: !GetAtt
```

```
        - TestFunctionDefinition
        - Id
      DefaultConfig:
        Execution:
          IsolationMode: GreengrassContainer
      Functions:
        - Id: TestLambda1
          FunctionArn: !Ref LambdaVersionArn
          FunctionConfiguration:
            Pinned: 'true'
            Executable: run.exe
            ExecArgs: argument1
            MemorySize: '512'
            Timeout: '2000'
            EncodingType: binary
            Environment:
              Variables:
                variable1: value1
              ResourceAccessPolicies:
                - ResourceId: ResourceId1
                  Permission: ro
                - ResourceId: ResourceId2
                  Permission: rw
              AccessSysfs: 'false'
              Execution:
                IsolationMode: GreengrassContainer
                RunAs:
                  Uid: '1'
                  Gid: '10'
  TestLoggerDefinition:
    Type: 'AWS::Greengrass::LoggerDefinition'
    Properties:
      Name: DemoTestLoggerDefinition
  TestLoggerDefinitionVersion:
    Type: 'AWS::Greengrass::LoggerDefinitionVersion'
    Properties:
      LoggerDefinitionId: !Ref TestLoggerDefinition
      Loggers:
        - Id: TestLogger1
          Type: AWSCloudWatch
          Component: GreengrassSystem
          Level: INFO
  TestResourceDefinition:
    Type: 'AWS::Greengrass::ResourceDefinition'
```

```
  Properties:
    Name: DemoTestResourceDefinition
TestResourceDefinitionVersion:
  Type: 'AWS::Greengrass::ResourceDefinitionVersion'
  Properties:
    ResourceDefinitionId: !Ref TestResourceDefinition
    Resources:
      - Id: ResourceId1
        Name: LocalDeviceResource
        ResourceDataContainer:
          LocalDeviceResourceData:
            SourcePath: /dev/TestSourcePath1
            GroupOwnerSetting:
              AutoAddGroupOwner: 'false'
              GroupOwner: TestOwner
      - Id: ResourceId2
        Name: LocalVolumeResourceData
        ResourceDataContainer:
          LocalVolumeResourceData:
            SourcePath: /dev/TestSourcePath2
            DestinationPath: /volumes/TestDestinationPath2
            GroupOwnerSetting:
              AutoAddGroupOwner: 'false'
              GroupOwner: TestOwner
TestSubscriptionDefinition:
  Type: 'AWS::Greengrass::SubscriptionDefinition'
  Properties:
    Name: DemoTestSubscriptionDefinition
TestSubscriptionDefinitionVersion:
  Type: 'AWS::Greengrass::SubscriptionDefinitionVersion'
  Properties:
    SubscriptionDefinitionId: !Ref TestSubscriptionDefinition
    Subscriptions:
      - Id: TestSubscription1
        Source: !Join
          - ':'
          - - 'arn:aws:iot'
            - !Ref 'AWS::Region'
            - !Ref 'AWS::AccountId'
            - thing/TestClientDevice1
        Subject: TestSubjectUpdated
        Target: !Ref LambdaVersionArn
TestGroup:
  Type: 'AWS::Greengrass::Group'
```

```
    Properties:
      Name: DemoTestGroupNewName
      RoleArn: !Join
        - ':'
        - - 'arn:aws:iam:'
          - !Ref 'AWS::AccountId'
          - role/TestUser
      InitialVersion:
        CoreDefinitionVersionArn: !Ref TestCoreDefinitionVersion
        DeviceDefinitionVersionArn: !Ref TestDeviceDefinitionVersion
        FunctionDefinitionVersionArn: !Ref TestFunctionDefinitionVersion
        SubscriptionDefinitionVersionArn: !Ref TestSubscriptionDefinitionVersion
        LoggerDefinitionVersionArn: !Ref TestLoggerDefinitionVersion
        ResourceDefinitionVersionArn: !Ref TestResourceDefinitionVersion
      Tags:
        KeyName0: value
        KeyName1: value
        KeyName2: value
Outputs:
  CommandToDeployGroup:
    Value: !Join
      - ' '
      - - groupVersion=$(cut -d'/' -f6 <<<
        - !GetAtt
          - TestGroup
          - LatestVersionArn
        - );
        - aws --region
        - !Ref 'AWS::Region'
        - greengrass create-deployment --group-id
        - !Ref TestGroup
        - '--deployment-type NewDeployment --group-version-id'
        - $groupVersion
```

支持 AWS 区域

目前，您仅可以在以下 AWS 区域 中创建并管理 AWS IoT Greengrass 资源。

- 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）

- 亚太地区（孟买）
- 亚太地区（首尔）
- 亚太地区（新加坡）
- 亚太地区（悉尼）
- 亚太地区（东京）
- 中国（北京）
- 欧洲地区（法兰克福）
- 欧洲地区（爱尔兰）
- 欧洲地区（伦敦）
- AWS GovCloud （美国西部）

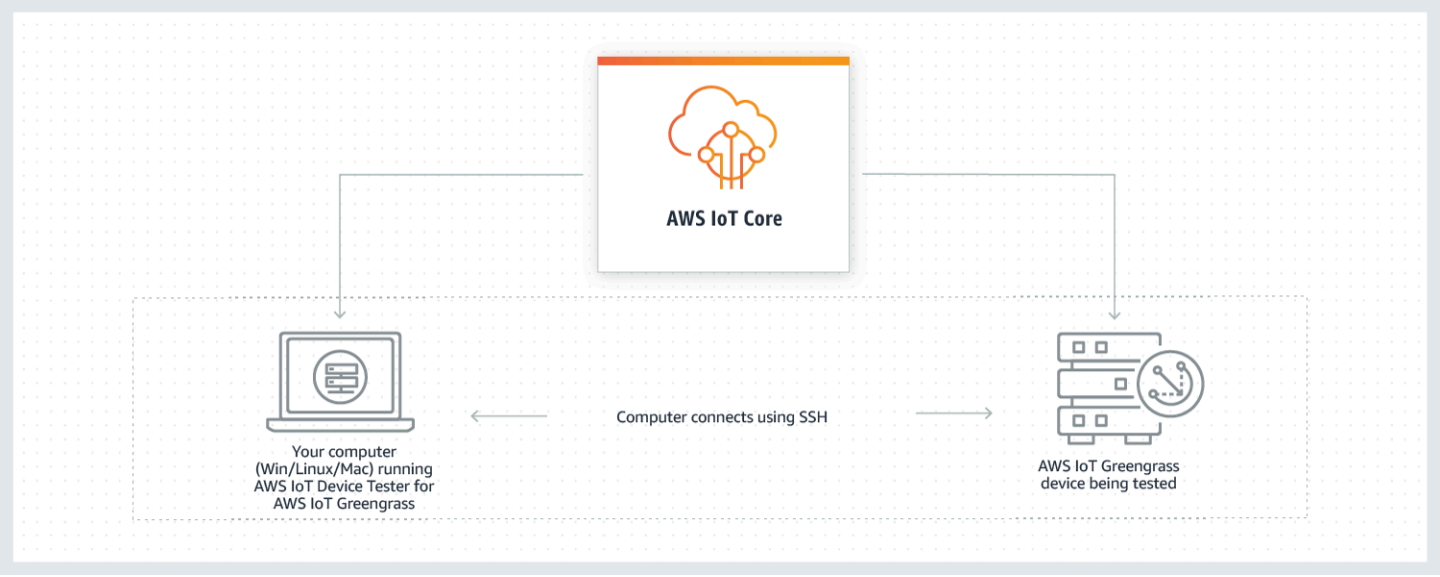
使用适用于 AWS IoT Greengrass V1 的 AWS IoT 设备测试器

AWS IoT 设备测试器 (IDT) 是一个可下载的测试框架，可让您验证物联网设备。由于 AWS IoT Greengrass Version 1 已进入维护模式，IDT for AWS IoT Greengrass V1 不再生成签名的资格报告。您将无法再通过设备资格认证计划使新 AWS IoT Greengrass V1 设备符合在AWS Partner 设备目录中发布的AWS 资格。但是，你可以继续使用 IDT AWS IoT Greengrass V1 来测试你的 Greengrass V1 设备。我们建议您使用适用于 AWS IoT Greengrass V2的 IDT 来进行资格认证并在 AWS Partner 设备目录中列示 Greengrass 设备。

请注意在连接到待测试设备的主机（Windows、macOS 或 Linux）上 AWS IoT Greengrass 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 资格套件

使用 IDT 验证 AWS IoT Greengrass 核心软件是否在您的硬件上运行并且可以与通信。 AWS IoT Greengrass AWS Cloud它还使用执行 end-to-end 测试 AWS IoT Core。例如，它验证您的设备是否能够发送和接收 MQTT 消息并正确处理它们。



AWS IoT Device Tester for 使用测试套件和测试组的概念 AWS IoT Greengrass 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相关的一组单独测试，例如 Greengrass 组部署和 MQTT 消息传递。

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 v4.0.0 开始，IDT for 将标准化配置设置和结果格式与测试套件环境 AWS IoT Greengrass 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。

适用于 AWS IoT Greengrass V1 的 AWS IoT Device Tester 的受支持版本

由于 AWS IoT Greengrass Version 1 已进入维护模式，AWS IoT Greengrass V1 IDT 不再生成签名的资格报告。我们建议使用适用于 AWS IoT Greengrass V2 的 IDT。

有关 IDT for AWS IoT Greengrass V2 的信息，请参阅《AWS IoT Greengrass V2 开发人员指南》中的使用 AWS IoT Device Tester for AWS IoT Greengrass V2。

> **Note**
>
> 如果适用于 AWS IoT Greengrass 的 IDT 与您使用的 AWS IoT Greengrass 版本不兼容，则您在开始测试运行时会收到通知。

下载此软件即表示您同意 AWS IoT Device Tester 许可协议。

不受支持的适用于 AWS IoT Greengrass 的 IDT 版本

本主题列出了不支持的适用于 AWS IoT Greengrass 的 IDT 版本。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 “适用于 AWS IoT Greengrass V1 的 AWS IoT Device Tester 的支持策略”。

适用于 AWS IoT Greengrass 版本 v1.11.6 和 v1.10.5 的 IDT v4.4.1

发行说明：

- 允许您验证和鉴定运行 AWS IoT Greengrass Core 软件 v1.11.6 和 v1.10.5 的设备。

- 包含次要错误修复。

测试套件版本：

GGQ_1.3.1

- 发布时间：2021 年 12 月 20 日

适用于 AWS IoT Greengrass 版本 v1.11.4 和 v1.10.4 的 IDT v4.4.0

发行说明：

- 允许您验证和鉴定运行 AWS IoT Greengrass Core 软件 v1.11.4 和 v1.10.4 的设备。
- 修复了一个问题，该问题导致测试运行期间显示的日志使用冗余的标签。

测试套件版本：

GGQ_1.3.0

- 已发布时间：2021 年 6 月 23 日
- 增加了对 Lambda、IAM 和 AWS STS 的 API 调用的重试次数，并改进了对节流或服务器问题的处理。
- 在 ML 和 Docker 测试用例中添加了对 Python 3.8 的支持。

适用于 AWS IoT Greengrass 版本 v1.11.1、v1.11.0 和 v1.10.3 的 IDT v4.0.2

发行说明：

- 修复了导致 IDT 掩盖硬件安全集成 (HSI) 错误的问题。
- 允许您使用 AWS IoT Device Tester for AWS IoT Greengrass 开发和运行自定义测试套件。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 中，如果显示安全警告消息，您可能需要为 IDT 授予安全例外。有关更多信息，请参阅 macOS 上的安全例外。

> **Note**
>
> AWS IoT Greengrass 不提供适用于 AWS IoT Greengrass Core 软件 v1.11.1 版本的 Dockerfile 或 Docker 映像。要测试您的设备是否符合 Docker 资格，请使用早期版本的 AWS IoT Greengrass Core 软件。

适用于 AWS IoT Greengrass 版本 v1.11.0、v1.10.1 和 v1.10.03 的 IDT v3.2.0

发行说明：

- 默认情况下，IDT 仅运行必要的资格测试。要获得其他功能的资格，您可以修改 `device.json` 文件。
- 在 `device.json` 中添加了一个可配置用于 SSH 连接配置的端口号。
- Docker 仅在无容器化情况下支持流管理器和机器学习 (ML)。容器、Docker 和硬件安全集成 (HSI) 不适用于 Docker 设备。
- 我们将 `device-ml.json` 和 `device-hsm.json` 合并成了 `device.json`。

适用于 AWS IoT Greengrass 版本的 IDT v3.1.3：v1.10.x, v1.9.x, v1.8.x

发行说明：

- 增加了对 AWS IoT Greengrass 版本 1.10.x 和 1.9.x 的 ML 功能资格认证的支持。现在，您可以使用 IDT 验证您的设备是否可以使用在云中存储和训练的模型在本地执行 ML 推理。
- 为 `run-suite` 命令添加了 `--stop-on-first-failure`。您可以使用此选项将 IDT 配置为在第一次失败时停止运行。建议在调试阶段在测试组级别使用此选项。
- 为 MQTT 测试添加了时钟偏移检查，以确保被测设备使用正确的系统时间。使用的时间必须在可接受的时间范围之内。
- 为 `run-suite` 命令添加了 `--update-idt`。您可以使用此选项设置对更新 IDT 的提示的响应。
- 为 `run-suite` 命令添加了 `--update-managed-policy`。您可以使用此选项设置对更新托管策略的提示的响应。
- 为自动更新 IDT 测试套件版本添加了错误修复。该修复可确保 IDT 可以下载适用于您的 AWS IoT Greengrass 版本的最新测试套件。

适用于 AWS IoT Greengrass 的 IDT 版本 3.0.1

发行说明：

- 增加了对 AWS IoT Greengrass 版本 1.10.1 的支持。
- IDT 测试套件版本的自动更新。IDT 可以下载适用于您的 AWS IoT Greengrass 版本的最新测试套件。通过此功能：
  - 测试套件使用 *`major.minor.patch`* 格式进行版本化。初始测试套件版本为 `GGQ_1.0.0`。
  - 您可以在命令行界面中以交互方式下载新的测试套件，或在启动 IDT 时设置 `upgrade-test-suite` 标记。

  有关更多信息，请参阅 the section called “测试套件版本”。

- 增加了 `list-supported-products`。您可以使用此命令列出已安装的 IDT 版本支持的 AWS IoT Greengrass 和测试套件版本。
- 增加了 `list-test-cases`。您可以使用此命令列出测试组中可用的测试用例。
- 为 `run-suite` 命令添加了 `test-id`。您可以使用此选项运行测试组中的单个测试用例。

适用于 AWS IoT Greengrass v1.10、v1.9.x 和 v1.8.x 的 IDT v2.3.0

在物理设备上进行测试时，支持 AWS IoT Greengrass v1.10、v1.9.x 和 v1.8.x。

在 Docker 容器中进行测试时，支持 AWS IoT Greengrass v1.10 和 v1.9.x。

发行说明：

- 增加了对 the section called “在 Docker 容器中运行 AWS IoT Greengrass” 的支持。您现在可以使用 IDT 来确定和验证设备是否可以在 Docker 容器中运行 AWS IoT Greengrass。
- 添加 AWS 托管策略 (`AWSIoTDeviceTesterForGreengrassFullAccess`)，该策略定义运行 AWS IoT Device Tester 所需的权限。如果新版本需要额外的权限，AWS 会将这些权限添加到此托管策略中，因此您无需更新 IAM 权限。
- 引入了检查，用于先验证您的环境（例如，设备连接和 Internet 连接）是否已正确设置，然后再运行测试用例。
- 改进了 IDT 中的 Greengrass 依赖项检查程序，使其可以更灵活地检查设备上的 libc。

适用于 AWS IoT Greengrass v1.10、v1.9.x 和 v1.8.x 的 IDT v2.2.0

发行说明：

- 增加了对 AWS IoT Greengrass v1.10 的支持。
- 增加了对 Greengrass Docker 应用程序部署连接器的支持。
- 增加了对 AWS IoT Greengrass 流管理器的支持。
- 在中国（北京）区域中添加对 AWS IoT Greengrass 的支持。

适用于 AWS IoT Greengrass v1.9.x、v1.8.x 和 v1.7.x 的 IDT v2.1.0

发行说明：

- 增加了对 AWS IoT Greengrass v1.9.4 的支持
- 增加了对 Linux-Armv6L 设备的支持。

适用于 AWS IoT Greengrass v1.9.3、v1.9.2、v.1.9.1、v1.9.0、v1.8.4、v1.8.3 和 v1.8.2 的 IDT v2.0.0

发行说明：

- 删除了所测试设备对 Python 的依赖。
- 测试套件的执行时间减少了 50% 以上，从而使鉴定过程更快。
- 可执行文件大小减少了 50% 以上，使下载和安装过程更快。
- 改进了对所有测试用例的超时乘数支持。
- 增强的诊断后消息，可更快地排除错误。
- 更新了运行 IDT 所需的权限策略模板。
- 增加了对 AWS IoT Greengrass v1.9.3 的支持

适用于 AWS IoT Greengrass v1.9.2、v1.9.1、v1.9.0、v1.8.3 和 v1.8.2 的 IDT v1.3.3

发行说明：

- 添加了对 Greengrass 1.9.1 和 1.8.3 版本的支持。
- 增加了对 Greengrass OpenWrt 的支持。
- 添加了 SSH 用户名和密码设备登录。
- 为 OpenWrt-armv7L 平台添加了原生测试错误修复。

适用于 AWS IoT Greengrass v1.8.1 的 IDT v1.2

发行说明：

- 添加了一个可配置的超时乘数，用于查找和排除超时问题（例如，低带宽连接）。

适用于 AWS IoT Greengrass 1.8.0 版本的 IDT 1.1 版本

发行说明：

- 添加了对 AWS IoT Greengrass 硬件安全集成 (HSI) 的支持。
- 添加了对 AWS IoT Greengrass 容器和无容器的支持。
- 添加了自动化 AWS IoT Greengrass 服务角色创建。

- 改进了测试资源清理。
- 添加了测试执行摘要报告。

适用于 AWS IoT Greengrass v1.7.1 的 IDT v1.1

发行说明：

- 添加了对 AWS IoT Greengrass 硬件安全集成 (HSI) 的支持。
- 添加了对 AWS IoT Greengrass 容器和无容器的支持。
- 添加了自动化 AWS IoT Greengrass 服务角色创建。
- 改进了测试资源清理。
- 添加了测试执行摘要报告。

适用于 AWS IoT Greengrass v1.6.1 的 IDT v1.0

发行说明：

- 增加了 OTA 测试错误修复以实现与将来版本的 AWS IoT Greengrass 的兼容性。

> **Note**
>
> 如果您使用适用于 AWS IoT Greengrass v1.6.1 的 IDT v1.0，则必须创建 Greengrass 服务角色。在更高版本中，IDT 会为您创建服务角色。

使用 IDT 运行 AWS IoT Greengrass 资格套件

您可以使用适用于 AWS IoT Greengrass 的 AWS IoT Device Tester (IDT) 来验证 AWS IoT Greengrass 核心软件是否能在您的硬件上运行并与 AWS Cloud 进行通信。它还使用 AWS IoT Core 执行端到端测试。例如，它验证您的设备是否能够发送和接收 MQTT 消息并正确处理它们。

由于 AWS IoT Greengrass Version 1 已进入维护模式，AWS IoT Greengrass V1 IDT 不再生成签名的资格报告。如果要将硬件添加到 AWS Partner 设备目录中，请运行 AWS IoT Greengrass V2 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设备资格认证计划和AWS IoT Greengrass V2 IDT 支持的版本。

除了测试设备之外，适用于 AWS IoT Greengrass 的 IDT 还在您的 AWS 账户 创建资源（例如，AWS IoT 事物、AWS IoT Greengrass 组、Lambda 函数等）来以促进资格认证过程。

为了创建这些资源，适用于 AWS IoT Greengrass 的 IDT 使用在 `config.json` 文件中配置的 AWS 凭证来代表您发出 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 AWS IoT Greengrass IDT 来运行 AWS IoT Greengrass 资格套件时，IDT 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，指明您的设备是否已通过资格认证所需的测试。

测试套件版本

适用于 AWS IoT Greengrass 的 IDT 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相关的一组单独测试，例如 Greengrass 组部署和 MQTT 消息传递。

从 IDT v3.0.0 开始，测试套件使用 *`major`*`.`*`minor`*`.`*`patch`* 格式进行版本化，例如，`GGQ_1.0.0`。当您下载 IDT 时，数据包中包含最新的测试套件版本。

> **Important**
>
> IDT 支持三个最新的测试套件版本以获得设备资格认证。有关更多信息，请参阅the section called “适用于 AWS IoT Greengrass V1 的 AWS IoT Device Tester 的支持策略”。
> 您可以运行 `list-supported-products` 来列出当前版本的 IDT 支持的 AWS IoT Greengrass 和测试套件版本。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。

IDT 配置设置的更新

新测试可能会引入新的 IDT 配置设置。

- 如果设置是可选的，IDT 将继续运行测试。
- 如果需要这些设置，IDT 会通知您并停止运行。配置设置后，请重新启动测试运行。

  配置设置位于 *`<device-tester-extract-location>`*`/configs` 文件夹中。有关更多信息，请参阅the section called “配置 IDT 设置”。

如果更新的测试套件版本添加了配置设置，IDT 会在 *<device-tester-extract-location>*/configs 中创建原始配置文件的副本。

测试组描述

IDT v2.0.0 and later

核心资格必备测试组

要确定您的 AWS IoT Greengrass 设备是否符合 AWS Partner Device Catalog 的要求，需要完成这些测试组。

AWS IoT Greengrass 核心依赖项

验证您的设备是否满足 AWS IoT Greengrass Core 软件的所有软硬件要求。

在 Docker 容器中进行测试时，此测试组中的 `Software Packages Dependencies` 测试用例不适用。

部署

验证 Lambda 函数是否可以部署到您的设备上。

MQTT

通过检查 Greengrass 核心与客户端设备（即本地 IoT 设备）之间的本地通信来验证 AWS IoT Greengrass 消息路由器的功能。

Over-the-Air (OTA)

验证您的设备是否可以成功执行 AWS IoT Greengrass Core 软件的 OTA 更新。

在 Docker 容器中进行测试时，此测试组不适用。

版本

检查提供的 AWS IoT Greengrass 版本是否与您使用的 AWS IoT Device Tester 版本兼容。

可选测试组

这些测试组可选。如果您选择进行这些可选测试，您的设备将在 AWS Partner Device Catalog 中与附加功能一起列出。

容器依赖项

验证设备是否满足在 Greengrass 核心上以容器模式运行 Lambda 函数的所有软硬件要求。

在 Docker 容器中进行测试时，此测试组不适用。

部署容器

验证 Lambda 函数是否可以部署在设备上并在 Greengrass 核心上以容器模式运行。

在 Docker 容器中进行测试时，此测试组不适用。

Docker 依赖项（IDT v2.2.0 和更高版本支持）

验证设备是否满足使用 Greengrass Docker 应用程序部署连接器以运行容器的所有必要技术依赖项

在 Docker 容器中进行测试时，此测试组不适用。

硬件安全性集成 (HSI)

验证提供的 HSI 共享库是否可以与硬件安全模块 (HSM) 交互，并正确实施所需的 PKCS#11 API。HSM 和共享库必须能够签署 CSR，执行 TLS 操作，并提供正确的密钥长度和公有密钥算法。

流管理器依赖项（IDT v2.2.0 及更高版本支持）

验证设备是否满足运行 AWS IoT Greengrass 流管理器所需的所有技术依赖项。

机器学习依赖项（IDT 版本 3.1.0 及更高版本支持）

验证设备是否满足在本地执行 ML 推理所需的所有技术依赖项。

机器学习推理测试（IDT 版本 3.1.0 及更高版本支持）

验证是否可以在给定的被测设备上执行 ML 推理。有关更多信息，请参阅the section called “可选：配置设备进行 ML 资格认证”。

机器学习推理容器测试（IDT 版本 3.1.0 及更高版本支持）

验证是否可以在给定的被测设备上执行 ML 推理，以及是否可以在 Greengrass 核心上以容器模式运行 ML 推理。有关更多信息，请参阅the section called “可选：配置设备进行 ML 资格认证”。

IDT v1.3.3 and earlier

核心资格必备测试组

要确定您的 AWS IoT Greengrass 设备是否符合 AWS Partner Device Catalog 的要求，需要完成这些测试。

AWS IoT Greengrass 核心依赖项

验证您的设备是否满足 AWS IoT Greengrass Core 软件的所有软硬件要求。

Combination（设备安全交互）

通过更改云中 Greengrass 组的连接信息来验证 Greengrass 核心设备上的设备证书管理器和 IP 检测的功能。测试组将轮换 AWS IoT Greengrass 服务器证书并验证 AWS IoT Greengrass 是否允许连接。

部署（是 IDT v1.2 及更早版本所必需的）

验证 Lambda 函数是否可以部署到您的设备上。

Device Certificate Manager (DCM)

验证 AWS IoT Greengrass 设备证书管理器是否可以在启动时生成服务器证书，并在证书即将到期时轮换证书。

IP 检测 (IPD)

验证当 Greengrass 核心设备的 IP 地址更改时，核心的连接信息是否会更新。有关更多信息，请参阅激活自动 IP 检测。

日志记录

验证 AWS IoT Greengrass 日志记录服务是否能够通过用 Python 编写的用户 Lambda 函数写入日志文件。

MQTT

通过发送路由到两个 Lambda 函数的主题消息来验证 AWS IoT Greengrass 消息路由器功能。

Native

验证 AWS IoT Greengrass 是否能够运行本机（已编译）Lambda 函数。

Over-the-Air (OTA)

验证您的设备是否可以成功执行 AWS IoT Greengrass Core 软件的 OTA 更新。

Penetration

如果未启用硬链接/软链接保护和 seccomp，则验证 AWS IoT Greengrass 核心软件是否无法启动。它还可用于验证其他与安全相关的功能。

影子

验证本地影子和影子云同步功能。

Spooler

验证 MQTT 消息是否依据默认后台处理程序配置进行排队。

Token Exchange Service (TES)

验证 AWS IoT Greengrass 是否能够使用其核心证书来交换有效的 AWS 凭证。

版本

检查提供的 AWS IoT Greengrass 版本是否与您使用的 AWS IoT Device Tester 版本兼容。

可选测试组

这些测试是可选的。如果您选择进行这些可选测试，您的设备将在 AWS Partner Device Catalog 中与附加功能一起列出。

容器依赖项

检查设备是否满足在容器模式下运行 Lambda 函数所需的所有依赖项。

硬件安全性集成 (HSI)

验证提供的 HSI 共享库是否可以与硬件安全模块 (HSM) 交互，并正确实施所需的 PKCS#11 API。HSM 和共享库必须能够签署 CSR，执行 TLS 操作，并提供正确的密钥长度和公有密钥算法。

本地资源访问

通过 AWS IoT Greengrass LRA API 向容器化 Lambda 函数提供对各种 Linux 用户和组拥有的本地文件和目录的访问权限来验证 AWS IoT Greengrass 的本地资源访问 (LRA) 功能。应基于本地资源访问配置允许或拒绝 Lambda 函数访问本地资源。

Network

验证是否可以从 Lambda 函数建立套接字连接。应根据 Greengrass 核心配置来允许或拒绝这些套接字连接。

运行 AWS IoT Greengrass 资格套件的先决条件

本节介绍使用 AWS IoT 设备测试器 (IDT) 运行 AWS IoT Greengrass 资格套件的 AWS IoT Greengrass 先决条件。

下载最新版本的 Dev AWS IoT ice Tester AWS IoT Greengrass

下载 IDT 的最新版本并将软件提取到文件系统中您具有读取和写入权限的位置。

> **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。

创建和配置 AWS 账户

在使用 IDT 之前 AWS IoT Greengrass，必须执行以下步骤：

1. 创建一个 AWS 账户. 如果您已经有 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、Greengrass 群组和 Lambda 函数。

要创建这些资源，IDT fo AWS IoT Greengrass r 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 Amazon Web Services 免费套餐的要求，但您在注册 AWS 账户时必须提供信用卡。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？

步骤 1：创建一个 AWS 账户

在此步骤中，将创建并配置 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”。

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 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。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。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，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后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。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。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. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

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《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 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。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

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 AWS IoT Greengrass 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，请跳转至 the section called “配置设备以运行 IDT 测试” 或 the section called “可选：配置 Docker 容器”。

- 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 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。

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "ManageRolePoliciesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:DetachRolePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ],
            "Condition": {
                "ArnEquals": {
                    "iam:PolicyARN": [
                        "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
                        "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
                        "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                    ]
                }
            }
        },
        {
            "Sid": "ManageRolesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:PassRole",
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ]
        }
    ]
}
```

> **Important**
>
> 以下策略授权来创建和管理适用于 AWS IoT Greengrass的 IDT 所需的角色。这包括附加以下 AWS 托管策略的权限：
>
> - AWSGreengrassResourceAccessRolePolicy
> - GreenGrassota UpdateArtifactAccess
> - AWSLambdaBasicExecutionRole

c. 选择下一步：标签。

d. 选择下一步：审核。

e. 对于名称，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 设置权限 页面上，选择 直接附加现有策略。

ii. 搜索您在上一步中创建的 IDTGreengrassIAMPermissions 策略。选中复选框。

iii. 搜索AWSIoTDeviceTesterForGreengrassFullAccess政策。选中复选框。

> **Note**
>
> AWSIoTDeviceTesterForGreengrassFullAccess是一种 AWS 托管策略，它定义了 IDT 创建和访问用于测试的 AWS 资源所需的权限。有关更多信息，请参阅 the section called “AWS IDT 的托管策略”。

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)

按照以下步骤 AWS CLI 使用配置 IDT 的 AWS IoT Greengrass权限。如果您已在控制台中配置权限，请跳转至 the section called “配置设备以运行 IDT 测试” 或 the section called “可选：配置 Docker 容器”。

1. AWS CLI 如果尚未安装，请在您的计算机上进行安装和配置。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，可用于通过命令行 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。

   Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ManageRolePoliciesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:DetachRolePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ],
            "Condition": {
                "ArnEquals": {
                    "iam:PolicyARN": [
                        "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
```

```
                        "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
                        "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                    ]
                }
            }
        },
        {
            "Sid": "ManageRolesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:PassRole",
                "iam:GetRole"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ]
        }
    ]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --
policy-document '{\"Version\": \"2012-10-17\", \"Statement\": [{\"Sid
\": \"ManageRolePoliciesForIDTGreengrass\",\"Effect\": \"Allow\",
\"Action\": [\"iam:DetachRolePolicy\", \"iam:AttachRolePolicy\"],
 \"Resource\": [\"arn:aws:iam::*:role/idt-*\",\"arn:aws:iam::*:role/
GreengrassServiceRole\"],\"Condition\": {\"ArnEquals\": {\"iam:PolicyARN\":
 [\"arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
\",\"arn:aws:iam::aws:policy/service-role/GreengrassOTAUpdateArtifactAccess
\",\"arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole\"]}}},
{\"Sid\": \"ManageRolesForIDTGreengrass\",\"Effect\": \"Allow\",\"Action\":
 [\"iam:CreateRole\",\"iam:DeleteRole\", \"iam:PassRole\", \"iam:GetRole
\"],\"Resource\": [\"arn:aws:iam::*:role/idt-*\",\"arn:aws:iam::*:role/
GreengrassServiceRole\"]}]}'
```

> **Note**
>
> 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。

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。<account-id>在命令中替换为您的 ID AWS 账户。

   ```
   aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
    arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
   ```

   c. 将 `AWSIoTDeviceTesterForGreengrassFullAccess` 策略附加到您的 IAM 用户。

   ```
   aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
    arn:aws:iam::aws:policy/AWSIoTDeviceTesterForGreengrassFullAccess
   ```

   > **Note**
   >
   > AWSIoTDeviceTesterForGreengrassFullAccess是一种 AWS 托管策略，它定义了 IDT 创建和访问用于测试的 AWS 资源所需的权限。有关更多信息，请参阅 the section called “AWS IDT 的托管策略”。

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。稍后您将使用此信息来配置您的 AWS 凭据文件。

5. 下一步：配置物理设备。

AWSAWS IoT 设备测试器的托管策略

AWSIoTDeviceTesterForGreengrassFullAccess托管策略允许 IDT 运行操作和收集使用量指标。此策略授予以下 IDT 权限：

- `iot-device-tester:CheckVersion`。 检查一组 AWS IoT Greengrass测试套件和 IDT 版本是否兼容。
- `iot-device-tester:DownloadTestSuite`. 下载测试套件。
- `iot-device-tester:LatestIdt`. 获取有关可供下载的最新 IDT 版本的信息。
- `iot-device-tester:SendMetrics`. 发布 IDT 收集的有关测试的使用情况数据。
- `iot-device-tester:SupportedVersion`。 获取 IDT 支持的测试套件版本列表 AWS IoT Greengrass 和测试套件版本。此信息显示在命令行窗口中。

配置设备以运行 IDT 测试

要配置设备，您必须执行以下操作：安装 AWS IoT Greengrass 依赖项，配置 AWS IoT Greengrass 核心软件，配置主机以访问设备以及在设备上配置用户权限。

在所测试设备上验证 AWS IoT Greengrass 依赖项

在 AWS IoT Greengrass 的 IDT 测试您的设备之前，确保已按照AWS IoT Greengrass 入门中所述设置好设备。有关支持的平台的信息，请参阅支持的平台。

配置 AWS IoT Greengrass 软件

适用于 AWS IoT Greengrass 的 IDT 将测试设备是否与特定版本的 AWS IoT Greengrass 兼容。IDT 提供了两种在设备上测试 AWS IoT Greengrass 的选项：

- 下载并使用某一版本的 AWS IoT Greengrass 核心软件。IDT 为您安装此软件。
- 使用设备上已安装的 AWS IoT Greengrass 核心软件版本。

> **Note**
>
> 每个版本的 AWS IoT Greengrass 都有相应的 IDT 版本。您必须下载与所用的 AWS IoT Greengrass 版本对应的 IDT 版本。

以下各节介绍了这些选项。您只需要使用其中一个选项。

选项 1：下载 AWS IoT Greengrass Core 软件并配置 AWS IoT Device Tester 以使用该软件

您可以从 AWS IoT Greengrass核心软件下载页面下载 AWS IoT Greengrass 核心软件。

1. 找到正确的架构和 Linux 发行版，然后选择 Download (下载)。
2. 将该 tar.gz 文件复制到 *<device-tester-extract-location>*/products/greengrass/ggc。

> **Note**
>
> 请勿更改 AWS IoT Greengrass tar.gz 文件的名称。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。例如，在该目录中同时放置 greengrass-linux-armv7l-1.7.1.tar.gz 和 greengrass-linux-armv7l-1.8.1.tar.gz 文件将导致测试失败。

选项 2：使用 AWS IoT Greengrass 及 AWS IoT Device Tester 的现有安装

通过将 greengrassLocation 属性添加到 *<device-tester-extract-location>*/configs 文件夹中的 device.json 文件，将 IDT 配置为测试设备上安装的 AWS IoT Greengrass 核心软件。例如：

```
"greengrassLocation" : "<path-to-greengrass-on-device>"
```

有关 device.json 文件的更多信息，请参阅配置 device.json。

在 Linux 设备上，AWS IoT Greengrass 核心软件的默认位置为 /greengrass。

> **Note**
>
> 您的设备应具有尚未开始的 AWS IoT Greengrass 核心软件安装。确保您已在设备上添加 ggc_user 用户和 ggc_group。有关更多信息，请参阅 AWS IoT Greengrass 的环境设置。

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>***

   其中 *`remote-ssh-user`* 是用于登录所测试设备的用户名，*`remote-device-ip`* 是用于运行测试的所测试设备的 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`。

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，适用于 AWS IoT Greengrass 的 IDT 必须能够在不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
   ```

配置设备以测试可选功能

以下主题介绍如何配置设备以针对可选功能运行 IDT 测试。请仅在要测试这些功能时才执行以下配置步骤。否则，请继续查看the section called “配置 IDT 设置”。

主题

可选：为适用于 AWS IoT Greengrass 的 IDT 配置 Docker 容器

AWS IoT Greengrass 提供了 Docker 映像和 Dockerfile，使您能够更轻松地在 Docker 容器中运行 AWS IoT Greengrass Core 软件。设置 AWS IoT Greengrass 容器后，您可以运行 IDT 测试。目前，只有 x86_64 Docker 架构支持运行适用于 AWS IoT Greengrass 的 IDT。

此功能需要 IDT v2.3.0 或更高版本。

设置 Docker 容器以运行 IDT 测试的过程，取决于您是使用由 AWS IoT Greengrass 提供的 Docker 映像还是 Dockerfile。

- 使用 Docker 映像。Docker 映像安装了 AWS IoT Greengrass Core 软件和依赖项。
- 使用 Dockerfile。Dockerfile 包含可用于构建自定义 AWS IoT Greengrass 容器映像的源代码。可对镜像进行修改以在不同的平台架构上运行或减少镜像的大小。

> **Note**
>
> AWS IoT Greengrass 不提供适用于 AWS IoT Greengrass Core 软件版本 v1.11.1 的 Dockerfile 或 Docker 映像。要在您的自定义容器映像上运行 IDT 测试，映像必须包含由 AWS IoT Greengrass 提供的 Dockerfile 中定义的依赖项。

在 Docker 容器中运行 AWS IoT Greengrass 时，不支持以下功能：

- 在 Greengrass 容器模式下运行的连接器。要在 Docker 容器中运行连接器，该连接器必须在无容器模式下运行。要查找支持无容器模式的连接器，请参阅the section called “AWS 提供的 Greengrass 连接器”。其中一些连接器具有隔离模式参数，您必须将此参数设为无容器。
- 本地设备和卷资源。在 Docker 容器中运行的用户定义 Lambda 函数必须直接访问核心上的设备和卷。

配置由 AWS IoT Greengrass 提供的 Docker 映像

请按照以下步骤配置 AWS IoT Greengrass Docker 映像以运行 IDT 测试。

先决条件

在开始本教程之前，必须执行以下操作。

- 您必须根据选择的 AWS Command Line Interface (AWS CLI) 版本，在主机上安装以下软件和版本。

  AWS CLI version 2

  - Docker，版本 18.09 或更高版本。早期版本也可能有效，但我们建议使用 18.09 或更高版本。
  - AWS CLI 2.0.0 版或更高版本。
    - 要安装 AWS CLI 版本 2，请参阅安装 AWS CLI 版本 2。

- 要配置 AWS CLI，请参阅配置 AWS CLI。

> **Note**
>
> 要在 Windows 计算机上升级到更高的 AWS CLI 版本 2，您必须重复 MSI 安装过程。

AWS CLI version 1

- Docker，版本 18.09 或更高版本。早期版本也可能有效，但我们建议使用 18.09 或更高版本。
- Python，版本 3.6 或更高版本。
- pip 版本 18.1 或更高版本。
- AWS CLI 1.17.10 版或更高版本
  - 要安装 AWS CLI 版本 1，请参阅安装 AWS CLI 版本 1。
  - 要配置 AWS CLI，请参阅配置 AWS CLI。
  - 要升级到 AWS CLI 版本 1 的最新版本，请运行以下命令。

```
pip install awscli --upgrade --user
```

> **Note**
>
> 如果你在 Windows 上使用 MSI 安装的 AWS CLI 版本 1，请注意以下几点：
>
> - 如果 AWS CLI 版本 1 安装过程中未能安装 botocore，请尝试使用 Python 和 pip 安装。
> - 要升级到更新的 AWS CLI 版本 1，您必须重复 MSI 安装过程。

- 要访问 Amazon Elastic Container Registry (Amazon ECR) 资源，您必须授予以下权限。
  - Amazon ECR 要求用户通过 AWS Identity and Access Management IAM 策略授予 `ecr:GetAuthorizationToken` 权限，然后才能对注册表进行身份验证并对 Amazon ECR 存储库推送或提取镜像。有关更多信息，请参阅 Amazon Elastic Container Registry 用户指南中的 Amazon ECR 存储库策略示例和访问一个 Amazon ECR 存储库。

1. 下载 Docker 映像并配置容器。您可以从 Docker Hub 或 Amazon Elastic Container Registry (Amazon ECR) 下载预构建的映像并在 Windows、macOS 和 Linux (x86_64) 平台上运行它。

要从 Amazon ECR 下载 Docker 映像，请完成 the section called “从 Amazon ECR 获取 AWS IoT Greengrass 容器镜像” 中的所有步骤。然后，返回到本主题以继续配置。

2. 仅限 Linux 用户：请确保运行 IDT 的用户有权运行 Docker 命令。有关更多信息，请参阅 Docker 文档中的以非根用户身份管理 Docker。

3. 要运行 AWS IoT Greengrass 容器，请使用适用于操作系统的命令：

   Linux

   ```
   docker run --rm --init -it -d --name aws-iot-greengrass \
   -p 8883:8883 \
   -v <host-path-to-kernel-config-file>:<container-path> \
   <image-repository>:<tag>
   ```

   - 将 *<host-path-to-kernel-config-file>* 替换为主机上的内核配置文件，将 *<container-path>* 替换为在容器中装载卷的路径。

     主机上的内核配置文件通常位于 /proc/config.gz 或 /boot/config-*<kernel-release-date>* 中。您可以运行 uname -r 以查找 *<kernel-release-date>* 值。

     示例：从 /boot/config-*<kernel-release-date>* 装载 Config 文件

     ```
     -v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
     ```

     示例：从 proc/config.gz 装载 Config 文件

     ```
     -v /proc/config.gz:/proc/config.gz \
     ```

   - 将命令中的 *<image-repository>*:*<tag>* 替换为目标镜像的存储库的名称以及标签。

     示例：指向 AWS IoT Greengrass Core 软件的最新版本

     ```
     216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
     ```

     要获取 AWS IoT Greengrass Docker 镜像的列表，请运行以下命令。

     ```
     aws ecr list-images --region us-west-2 --registry-id 216483018798 --
     repository-name aws-iot-greengrass
     ```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- 将命令中的 *<image-repository>*:*<tag>* 替换为目标镜像的存储库的名称以及标签。

  示例：指向 AWS IoT Greengrass Core 软件的最新版本

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  要获取 AWS IoT Greengrass Docker 镜像的列表，请运行以下命令：

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- 将命令中的 *<image-repository>*:*<tag>* 替换为目标镜像的存储库的名称以及标签。

  示例：指向 AWS IoT Greengrass Core 软件的最新版本

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  要获取 AWS IoT Greengrass Docker 镜像的列表，请运行以下命令：

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

> **Important**
>
> 使用 IDT 进行测试时，请勿包含 `--entrypoint /greengrass-entrypoint.sh \` 参数，该参数针对常规 AWS IoT Greengrass 使用运行镜像。

4. 下一步：配置 AWS 凭证和 `device.json` 文件。

配置 AWS IoT Greengrass 提供的 dockerfile

请按照以下步骤配置从 AWS IoT Greengrass Dockerfile 构建的 Docker 映像以运行 IDT 测试。

1. 从 the section called “AWS IoT Greengrass Docker 软件”，将 Dockerfile 包下载到您的主机并将其解压缩。
2. 打开 `README.md`。接下来的三个步骤将参考此文件中的部分。
3. 请确保您满足先决条件部分中的要求。
4. 仅限 Linux 用户：完成启用符号链接和硬链接保护以及启用 IPv4 网络转发步骤。
5. 要构建 Docker 映像，请完成步骤 1。构建 AWS IoT Greengrass Docker 映像。然后，返回到本主题以继续配置。
6. 要运行 AWS IoT Greengrass 容器，请使用适用于操作系统的命令：

   Linux

   ```
   docker run --rm --init -it -d --name aws-iot-greengrass \
   -p 8883:8883 \
   -v <host-path-to-kernel-config-file>:<container-path> \
   <image-repository>:<tag>
   ```

   - 将 *<host-path-to-kernel-config-file>* 替换为主机上的内核配置文件，将 *<container-path>* 替换为在容器中装载卷的路径。

     主机上的内核配置文件通常位于 `/proc/config.gz` 或 `/boot/config-`*<kernel-release-date>* 中。您可以运行 `uname -r` 以查找 *<kernel-release-date>* 值。

     示例：从 `/boot/config-`*<kernel-release-date>* 装载 Config 文件

     ```
     -v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
     ```

示例：从 `proc/config.gz` 装载 Config 文件

```
-v /proc/config.gz:/proc/config.gz \
```

- 将命令中的 *<image-repository>*:*<tag>* 替换为目标镜像的存储库的名称以及标签。

  示例：指向 AWS IoT Greengrass Core 软件的最新版本

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  要获取 AWS IoT Greengrass Docker 镜像的列表，请运行以下命令。

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- 将命令中的 *<image-repository>*:*<tag>* 替换为目标镜像的存储库的名称以及标签。

  示例：指向 AWS IoT Greengrass Core 软件的最新版本

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  要获取 AWS IoT Greengrass Docker 镜像的列表，请运行以下命令：

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
```

```
<image-repository>:<tag>
```

- 将命令中的 *<image-repository>*:*<tag>* 替换为目标镜像的存储库的名称以及标签。

  示例：指向 AWS IoT Greengrass Core 软件的最新版本

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  要获取 AWS IoT Greengrass Docker 镜像的列表，请运行以下命令：

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

> **Important**
>
> 使用 IDT 进行测试时，请勿包含 `--entrypoint /greengrass-entrypoint.sh \` 参数，该参数针对常规 AWS IoT Greengrass 使用运行镜像。

7. 下一步：配置 AWS 凭证和 device.json 文件。

对适用于 AWS IoT Greengrass 的 IDT 的 Docker 容器设置进行问题排查

使用以下信息可帮助解决与为 IDT 运行 Docker 容器以进行 AWS IoT Greengrass 测试相关的问题。

警告：加载 config 文件：/home/user/.docker/config.json - stat /home/<user>/.docker/config.json 时出错：权限被拒绝

如果在 Linux 上运行 `docker` 命令时出现此错误，请运行以下命令。在以下命令中，将 *<user>* 替换为运行 IDT 的用户。

```
sudo chown <user>:<user> /home/<user>/.docker -R
sudo chmod g+rwx /home/<user>/.docker -R
```

可选：配置设备进行 ML 资格认证

适用于 AWS IoT Greengrass 的 IDT 提供机器学习 (ML) 资格认证测试，以验证设备是否可以使用经过云训练的模型在本地执行 ML 推理。

要运行 ML 资格认证测试，必须先按照the section called “配置设备以运行 IDT 测试”中的说明配置设备。然后，按照本主题中的步骤安装要运行的 ML 框架的依赖项。

需要有 IDT v3.1.0 或更高版本才能运行 ML 资格认证测试。

安装 ML 框架依赖项

所有 ML 框架依赖项都必须安装在 `/usr/local/lib/python3.x/site-packages` 目录下。为确保将这些依赖项安装在正确的目录下，建议您在安装时使用 `sudo` root 权限。资格认证测试不支持虚拟环境。

> **Note**
>
> 如果您正在测试使用容器化运行的 Lambda 函数（在 Greengrass 容器模式下），则不支持为 `/usr/local/lib/python3.x` 下的 Python 库创建符号链接。为避免错误，必须在正确的目录下安装依赖项。

按照以下步骤安装目标框架的依赖项：

- 安装 MXNet 依赖项
- the section called “安装 TensorFlow 依赖项”
- 安装 DLR 依赖项

安装 Apache MXNet 依赖项

此框架的 IDT 资格认证测试具有以下依赖项：

- Python 3.6 或 Python 3.7。

> **Note**
>
> 如果您使用的是 Python 3.6，则必须创建从 Python 3.7 二进制文件到 Python 3.6 二进制文件的符号链接。这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。例如：
>
> ```
> sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
> ```

- Apache MXNet v1.2.1 或更高版本。
- NumPy。该版本必须与您的 MXNet 版本兼容。

正在安装 MXNet

按照 MXNet 文档中的说明安装 MXNet。

> **Note**
>
> 如果设备上同时安装了 Python 2.x 和 Python 3.x，请在运行来安装依赖项的命令中使用 Python 3.x。

验证 MXNet 安装

选择以下一种方式来验证 MXNet 安装。

方式 1：通过 SSH 接入设备并运行脚本

1. 通过 SSH 接入设备。
2. 运行以下脚本，验证是否正确安装了依赖项。

```
sudo python3.7 -c "import mxnet; print(mxnet.__version__)"
```

```
sudo python3.7 -c "import numpy; print(numpy.__version__)"
```

结果将输出版本号，而脚本应该顺利退出。

方式 2：运行 IDT 依赖项测试

1. 确保已配置 `device.json` 用于执行 ML 资格认证。有关更多信息，请参阅the section called “为 ML 资格认证配置 device.json”。
2. 运行框架的依赖项测试。

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

测试摘要会为 `mldependencies` 显示 PASSED 结果。

安装 TensorFlow 依赖项

此框架的 IDT 资格认证测试具有以下依赖项：

- Python 3.6 或 Python 3.7。

  > **Note**
  >
  > 如果您使用的是 Python 3.6，则必须创建从 Python 3.7 二进制文件到 Python 3.6 二进制文件的符号链接。这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。例如：
  >
  > ```
  > sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
  > ```

- TensorFlow 1.x。

安装 TensorFlow

按照 TensorFlow 文档中的说明，使用 pip 或从源代码安装 TensorFlow 1.x。

> **Note**
>
> 如果设备上同时安装了 Python 2.x 和 Python 3.x，请在运行来安装依赖项的命令中使用 Python 3.x。

验证 TensorFlow 安装

选择以下一种方式来验证 TensorFlow 安装。

方式 1：通过 SSH 接入设备并运行脚本

1. 通过 SSH 接入设备。
2. 运行以下脚本，验证是否正确安装了依赖项。

   ```
   sudo python3.7 -c "import tensorflow; print(tensorflow.__version__)"
   ```

   结果将输出版本号，而脚本应该顺利退出。

方式 2：运行 IDT 依赖项测试

1. 确保已配置 `device.json` 用于执行 ML 资格认证。有关更多信息，请参阅the section called “为 ML 资格认证配置 device.json”。
2. 运行框架的依赖项测试。

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

   测试摘要会为 `mldependencies` 显示 `PASSED` 结果。

安装 Amazon SageMaker Neo 深度学习运行时 (DLR) 依赖项

此框架的 IDT 资格认证测试具有以下依赖项：

- Python 3.6 或 Python 3.7。

  > **Note**
  >
  > 如果您使用的是 Python 3.6，则必须创建从 Python 3.7 二进制文件到 Python 3.6 二进制文件的符号链接。这会将设备配置为满足 AWS IoT Greengrass 的 Python 要求。例如：
  >
  > ```
  > sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
  > ```

- SageMaker Neo DLR。
- numpy。

安装 DLR 测试依赖项后，必须编译模型。

安装 DLR

按照 DLR 文档中的说明安装 Neo DLR。

> **Note**
>
> 如果设备上同时安装了 Python 2.x 和 Python 3.x，请在运行来安装依赖项的命令中使用 Python 3.x。

验证 DLR 安装

选择以下一种选项来验证 DLR 安装。

方式 1：通过 SSH 接入设备并运行脚本

1. 通过 SSH 接入设备。
2. 运行以下脚本，验证是否正确安装了依赖项。

```
sudo python3.7 -c "import dlr; print(dlr.__version__)"
```

```
sudo python3.7 -c "import numpy; print(numpy.__version__)"
```

结果将输出版本号，而脚本应该顺利退出。

方式 2：运行 IDT 依赖项测试

1. 确保已配置 `device.json` 用于执行 ML 资格认证。有关更多信息，请参阅the section called “为 ML 资格认证配置 device.json”。
2. 运行框架的依赖项测试。

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

测试摘要会为 `mldependencies` 显示 PASSED 结果。

编译 DLR 模型

必须先编译 DLR 模型，然后才能将其用于 ML 资格认证测试。有关具体步骤，请选择以下一种选项：

方式 1：使用 Amazon SageMaker 编译模型

按照以下步骤使用 SageMaker 编译 IDT 提供的 ML 模型。该模型已预先使用 Apache MXNet 进行训练。

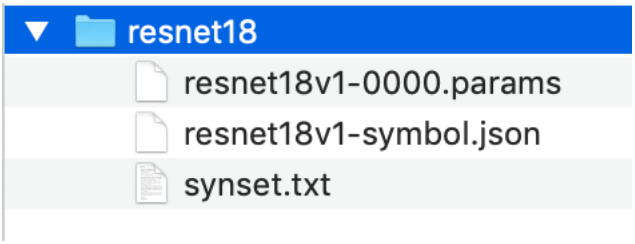
1. 确认 SageMaker 是否支持您的设备类型。有关更多信息，请参阅 Amazon SageMaker API 参考中的目标设备选项。如果当前 SageMaker 不支持您的设备类型，请按 the section called “方式 2：使用 TVM 编译 DLR 模型” 中的步骤操作。

   > **Note**
   >
   > 使用经 SageMaker 编译的模型运行 DLR 测试可能需要 4 到 5 分钟。在此过程中请勿停止 IDT。

2. 下载包含 DLR 的未编译、预训练 MXNet 模型的 tarball 文件：

   - dlr-noncompiled-model-1.0.tar.gz

3. 解压缩该 tarball 文件。此命令会生成以下目录结构。

4. 将 `synset.txt` 移出 `resnet18` 目录。记下新位置。稍后，您需要将此文件复制到已编译的模型目录中。

5. 压缩 `resnet18` 目录的内容。

   ```
   tar cvfz model.tar.gz resnet18v1-symbol.json resnet18v1-0000.params
   ```

6. 将压缩文件上传到 AWS 账户 中的 Amazon S3 存储桶中，然后按照编译模型（控制台）中的步骤创建编译作业。

   a. 对于输入配置，请使用以下值：

      - 对于数据输入配置，请输入 `{"data": [1, 3, 224, 224]}`。
      - 对于机器学习框架，请选择 `MXNet`。

   b. 对于输出配置，请使用以下值：

- 对于 S3 输出位置，请输入要存储已编译模型的 Amazon S3 存储桶或文件夹的路径。
- 对于目标设备，请选择您的设备类型。

7. 从指定的输出位置下载已编译的模型，然后解压缩该文件。
8. 将 `synset.txt` 复制到已编译的模型目录中。
9. 将已编译的模型目录的名称更改为 `resnet18`。

   已编译的模型目录必须具有以下目录结构。

方式 2：使用 TVM 编译 DLR 模型

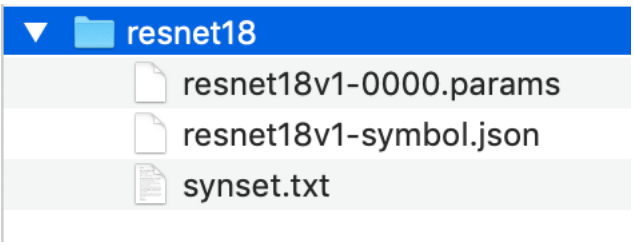
按照以下步骤使用 TVM 编译 IDT 提供的 ML 模型。此模型已预先使用 Apache MXNet 进行训练，因此您必须在编译模型的计算机或设备上安装 MXNet。要安装 MXNet，请按照 MXNet 文档中的说明进行操作。

> **Note**
>
> 我们建议您在目标设备上编译模型。此做法是可选的，但它可以帮助确保兼容性并减少潜在问题。

1. 下载包含 DLR 的未编译、预训练 MXNet 模型的 tarball 文件：

   - dlr-noncompiled-model-1.0.tar.gz

2. 解压缩该 tarball 文件。此命令会生成以下目录结构。



3. 按照 TVM 文档中的说明，从源代码为平台构建和安装 TVM。
4. 构建 TVM 后，为 resnet18 模型运行 TVM 编译。以下步骤根据 TVM 文档中的编译深度学习模型快速入门教程的内容设计而成。

   a. 从克隆的 TVM 存储库中打开 `relay_quick_start.py` 文件。

   b. 更新在 Relay 中定义神经网络的代码。可以使用以下选项之一：

   - 选项 1：使用 `mxnet.gluon.model_zoo.vision.get_model` 获取 Relay 模块和参数：

     ```
     from mxnet.gluon.model_zoo.vision import get_model
     block = get_model('resnet18_v1', pretrained=True)
     mod, params = relay.frontend.from_mxnet(block, {"data": data_shape})
     ```

   - 选项 2：将您在步骤 1 中下载的未编译模型中的以下文件复制到 `relay_quick_start.py` 文件所在的目录。这些文件中包含了 Relay 模块和参数。
     - `resnet18v1-symbol.json`
     - `resnet18v1-0000.params`

   c. 将保存和加载已编译模块的代码更新为以下代码。

   ```
   from tvm.contrib import util
   path_lib = "deploy_lib.so"
   #  Export the model library based on your device architecture
   lib.export_library("deploy_lib.so", cc="aarch64-linux-gnu-g++")
   with open("deploy_graph.json", "w") as fo:
       fo.write(graph)
   with open("deploy_param.params", "wb") as fo:
       fo.write(relay.save_param_dict(params))
   ```

   d. 构建模型：

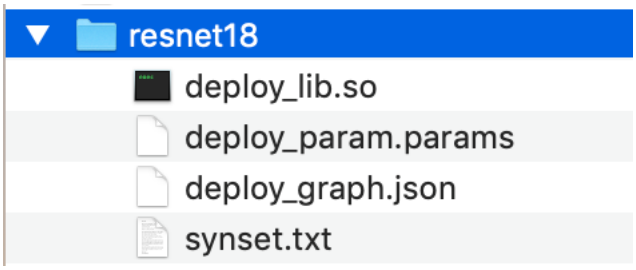
   ```
   python3 tutorials/relay_quick_start.py --build-dir ./model
   ```

此命令会生成以下文件。

- deploy_graph.json
- deploy_lib.so
- deploy_param.params

5. 将生成的模型文件复制到名为 resnet18 的目录中。这是已编译的模型目录。
6. 将已编译的模型目录复制到主机计算机。然后，将您在步骤 1 中下载的未编译模型中的 synset.txt 复制到已编译的模型目录中。

   已编译的模型目录必须具有以下目录结构。



接下来，配置 AWS 凭证和 device.json 文件。

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。

配置 AWS凭证

您必须在 *<device-tester-extract-location>* /configs/config.json 文件中配置 IAM 用户凭证。使用在the section called “创建和配置 AWS 账户”中创建的适用于 AWS IoT Greengrass 的 IDT 用户的凭证。您可以采用以下两种方法之一来指定凭证：

- 凭证文件
- 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`

用以下格式将 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>
```

要将适用于 AWS IoT Greengrass 的 IDT 配置为使用 `credentials` 文件中的 AWS 凭证，请编辑 `config.json` 文件，如下所示：

```
{
 "awsRegion": "us-west-2",
 "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 的 IDT 可使用 `AWS_ACCESS_KEY_ID` 和 `AWS_SECRET_ACCESS_KEY` 环境变量来存储您的 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 以使用环境变量，请编辑 `config.json` 文件中的 `auth` 部分。示例如下：

```
{
 "awsRegion": "us-west-2",
 "auth": {
  "method": "environment"
 }
}
```

配置 device.json

除了 AWS 凭证以外，适用于 AWS IoT Greengrass 的 IDT 还需要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）。

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：

Physical device

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "container",
        "value": "yes | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
```

```
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "yes | no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ********** Remove the section below if the device is not qualifying for ML
**************,
      {
        "name": "mlLambdaContainerizationMode",
        "value": "container | process | both"
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
      },

*******************************************************************************************
   ],
   ********** Remove the section below if the device is not qualifying for HSI
***************
   "hsm": {
     "p11Provider": "/path/to/pkcs11ProviderLibrary",
     "slotLabel": "<slot_label>",
     "slotUserPin": "<slot_pin>",
     "privateKeyLabel": "<key_label>",
     "openSSLEngine": "/path/to/openssl/engine"
   },

*******************************************************************************************
   ********** Remove the section below if the device is not qualifying for ML
***************
   "machineLearning": {
     "dlrModelPath": "/path/to/compiled/dlr/model",
     "environmentVariables": [
       {
         "key": "<environment-variable-name>",
```

```
          "value": "<Path:$PATH>"
        }
      ],
      "deviceResources": [
        {
          "name": "<resource-name>",
          "path": "<resource-path>",
          "type": "device | volume"
        }
      ]
    },

**********************************************************************************************
    "kernelConfigLocation": "",
    "greengrassLocation": "",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

> **Note**
>
> 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。

Docker container

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64"
      },
      {
        "name": "container",
        "value": "no"
      },
      {
        "name": "docker",
        "value": "no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ********** Remove the section below if the device is not qualifying for ML
 **************,
      {
        "name": "mlLambdaContainerizationMode",
        "value": "process"
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
```

```
      },

 ****************************************************************************************
    ],
    ********** Remove the section below if the device is not qualifying for ML
 ***************
    "machineLearning": {
      "dlrModelPath": "/path/to/compiled/dlr/model",
      "environmentVariables": [
        {
          "key": "<environment-variable-name>",
          "value": "<Path:$PATH>"
        }
      ],
      "deviceResources": [
        {
          "name": "<resource-name>",
          "path": "<resource-path>",
          "type": "device | volume"
        }
      ]
    },

 ****************************************************************************************
    "kernelConfigLocation": "",
    "greengrassLocation": "",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "docker",
          "containerId": "<container-name | container-id>",
          "containerUser": "<user>"
        }
      }
    ]
  }
]
```

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主板。

> **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`

包含设备支持的功能的数组。所有功能都是必需的。

`os` 和 `arch`

支持的操作系统 (OS) 和架构组合：

- `linux`, `x86_64`
- `linux`, `armv6l`
- `linux`, `armv7l`
- `linux`, `aarch64`
- `ubuntu`, `x86_64`
- `openwrt`, `armv7l`
- `openwrt`, `aarch64`

> **Note**
>
> 如果您使用 IDT 测试在 Docker 容器中运行 AWS IoT Greengrass，则仅支持 x86_64 Docker 架构。

`container`

验证设备是否满足在 Greengrass 核心上以容器模式运行 Lambda 函数的所有软硬件要求。

有效值为 yes 或 no。

`docker`

验证设备是否满足使用 Greengrass Docker 应用程序部署连接器以运行容器的所有必要技术依赖项

有效值为 yes 或 no。

`streamManagement`

验证设备是否满足运行 AWS IoT Greengrass 流管理器所需的所有技术依赖项。

有效值为 yes 或 no。

`hsi`

验证提供的 HSI 共享库是否可以与硬件安全模块 (HSM) 交互，并正确实施所需的 PKCS#11 API。HSM 和共享库必须能够签署 CSR，执行 TLS 操作，并提供正确的密钥长度和公有密钥算法。

有效值为 yes 或 no。

`ml`

验证设备是否满足在本地执行 ML 推理所需的所有技术依赖项。

有效值可以是 mxnet、tensorflow、dlr 和 no 的任意组合（例如 mxnet、mxnet,tensorflow、mxnet,tensorflow,dlr 或 no）。

`mlLambdaContainerizationMode`

验证设备是否满足在 Greengrass 设备的容器模式下执行 ML 推理所需的所有技术依赖关系。

有效值为 container、process 或 both。

`processor`

验证设备是否满足指定处理器类型的所有硬件要求。

有效值为 cpu 或 gpu。

> **Note**
>
> 如果您不想使用 container、docker、streamManager、hsi 或 ml 功能，则可以将对应的 value 设置为 no。

Docker 仅支持 `streamManagement` 和`ml` 的功能资格认证。

`machineLearning`

可选。ML 资格认证测试的配置信息。有关更多信息，请参阅the section called “为 ML 资格认证配置 device.json”。

`hsm`

可选。用于通过 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。

`hsm.p11Provider`

PKCS#11 实施的 libdl-loadable 库的绝对路径。

`hsm.slotLabel`

用于标识硬件模块的槽标签。

`hsm.slotUserPin`

用于针对模块对 AWS IoT Greengrass 核心进行身份验证的用户 PIN。

`hsm.privateKeyLabel`

用于标识硬件模块中的键的标签。

`hsm.openSSLEngine`

指向 OpenSSL 引擎的 `.so` 文件（用于在 OpenSSL 上启用 PKCS # 11 支持）的绝对路径。由 AWS IoT Greengrass OTA 更新代理使用。

`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。目前，唯一支持的值，对于物理设备为 `ssh`，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.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 docker 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.auth.credentials.privKeyPath

用于登录所测试设备的私有密钥的完整路径。

`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。

`greengrassLocation`

设备上 AWS IoT Greengrass 核心软件的位置。

对于物理设备，此值仅在您使用 AWS IoT Greengrass 的现有安装时使用。使用该属性可告知 IDT 使用设备上安装的 AWS IoT Greengrass 核心软件版本。

在源自 AWS IoT Greengrass 提供的 Docker 镜像或 Dockerfile 的 Docker 容器中运行测试时，请将此值设置为 `/greengrass`。

`kernelConfigLocation`

可选。内核配置文件的路径。AWS IoTDevice Tester 使用此文件检查设备是否启用了所需的内核功能。如果未指定，IDT 使用以下路径搜索内核配置文件：`/proc/config.gz` 和 `/boot/config-`*`<kernel-version>`*。AWS IoTDevice Tester 使用它所找到的第一个路径。

为 ML 资格认证配置 device.json

本节介绍设备配置文件中适用于 ML 资格认证的可选属性。如果计划运行 ML 资格认证测试，则必须定义适用于您的使用案例的属性。

您可以使用 `device-ml.json` 模板来定义设备的配置设置。此模板包含可选的 ML 属性。您也可以使用 `device.json` 并添加 ML 资格认证属性。这些文件位于 *`<device-tester-extract-location>`*`/configs` 中，包括 ML 资格认证属性。如果使用 `device-ml.json`，则必须将该文件重命名为 `device.json`，然后再运行 IDT 测试。

有关不适用于 ML 资格认证的设备配置属性的信息，请参阅the section called “配置 device.json”。

`features` 数组中的 `ml`

您的主板支持的 ML 框架。此属性需要 IDT v3.1.0 或更高版本。

- 如果您的主板仅支持一个框架，请指定该框架。例如：

```
{
    "name": "ml",
    "value": "mxnet"
}
```

- 如果您的主板支持多个框架，请以逗号分隔列表的形式指定这些框架。例如：

```
{
    "name": "ml",
    "value": "mxnet,tensorflow"
}
```

`features` 数组中的 `mlLambdaContainerizationMode`

测试将采用的容器化模式。此属性需要 IDT v3.1.0 或更高版本。

- 选择 `process`，将使用非容器化的 Lambda 函数运行 ML 推理代码。此选项需要 AWS IoT Greengrass 1.10.x 或更高版本。
- 选择 `container`，将使用容器化的 Lambda 函数运行 ML 推理代码。
- 选择 `both`，可在两种模式下运行 ML 推理代码。此选项需要 AWS IoT Greengrass 1.10.x 或更高版本。

`features` 数组中的 `processor`

指示您的主板支持的硬件加速器。此属性需要 IDT v3.1.0 或更高版本。

- 如果主板使用 CPU 作为处理器，则选择 `cpu`。
- 如果主板使用 GPU 作为处理器，则选择 `gpu`。

`machineLearning`

可选。ML 资格认证测试的配置信息。此属性需要 IDT v3.1.0 或更高版本。

`dlrModelPath`

使用 `dlr` 框架时需要。DLR 已编译模型目录的绝对路径，必须命名为 `resnet18`。有关更多信息，请参阅the section called “编译 DLR 模型”。

> **Note**
>
> 以下是 macOS 上的路径示例：`/Users/<user>/Downloads/resnet18`。

`environmentVariables`

可将设置动态传递给 ML 推理测试的键值对数组。对于 CPU 设备为可选。您可以使用此部分添加设备类型所需的特定于框架的环境变量。有关这些要求的信息，请参阅框架或设备的官方网站。例如，要在某些设备上运行 MxNet 推理测试，可能需要以下环境变量。

```
"environmentVariables": [
    ...
    {
        "key": "PYTHONPATH",
        "value": "$MXNET_HOME/python:$PYTHONPATH"
    },
    {
        "key": "MXNET_HOME",
        "value": "$HOME/mxnet/"
    },
    ...
]
```

> **Note**
>
> `value` 字段可能因您的 MXNet 安装而异。

如果您要测试在 GPU 设备上使用容器化运行的 Lambda 函数，请为 GPU 库添加环境变量。这将使得 GPU 执行计算成为可能。要使用不同的 GPU 库，请参阅库或设备的官方文档。

> **Note**
>
> 如果将 `mlLambdaContainerizationMode` 功能设置为 `container` 或 `both`，请配置以下键。

```
"environmentVariables": [
    {
        "key": "PATH",
        "value": "<path/to/software/bin>:$PATH"
    },
    {
        "key": "LD_LIBRARY_PATH",
```

```
        "value": "<path/to/ld/lib>"
    },
    ...
]
```

`deviceResources`

对于 GPU 设备为必需。包含可通过 Lambda 函数访问的本地资源。可使用此部分来添加本地设备和卷资源。

- 对于设备资源，请指定 `"type": "device"`。对于 GPU 设备，设备资源应为 `/dev` 下的与 GPU 相关的设备文件。

  > **Note**
  >
  > 但 `/dev/shm` 目录除外。只能将其配置为卷资源。

- 对于卷资源，请指定 `"type": "volume"`。

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。

以下 `run-suite` 命令示例介绍如何针对某个设备池运行资格测试。设备池是一组相同的设备。

IDT v3.0.0 and later

在指定的测试套件中运行所有测试组。

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --pool-
id <pool-id>
```

使用 `list-suites` 命令列出 `tests` 文件夹中的测试套件。

在测试套件中运行特定的测试组。

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --group-
id <group-id> --pool-id <pool-id>
```

使用 `list-groups` 命令列出测试套件中的测试组。

在测试组中运行特定测试用例。

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

在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>
```

列出测试组中的测试用例。

```
devicetester_[linux | mac | win_x86-64] list-test-cases --group-id <group-id>
```

`run-suite` 命令的选项是可选的。例如，如果您的 `device.json` 文件中只定义了一个设备池，则可以忽略 `pool-id`。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 “设置默认更新行为”。

有关 `run-suite` 和其他 IDT 命令的更多信息，请参阅 the section called “IDT 命令”。

IDT v2.3.0 and earlier

在指定套件中运行所有测试组。

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --pool-
id <pool-id>
```

运行特定测试组。

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --group-
id <group-id> --pool-id <pool-id>
```

如果您在单个设备池上运行单个测试套件，则 `suite-id` 和 `pool-id` 是可选的。这意味着您的 `device.json` 文件中只定义了一个设备池。

检查 Greengrass 的依赖关系

我们建议您运行依赖项检查程序测试组，以确保在运行相关测试组之前已安装所有 Greengrass 依赖项。例如：

- 在运行核心资格测试组之前运行 ggcdependencies。
- 在运行特定于容器的测试组之前运行 containerdependencies。
- 在运行特定于 Docker 的测试组之前运行 dockerdependencies。
- 在运行特定于流管理器的测试组之前运行 ggcstreammanagementdependencies。

设置默认更新行为

当您开始测试运行时，IDT 会在线检查是否有更新的测试套件版本。如果有，IDT 会提示您更新到最新的可用版本。您可以设置 upgrade-test-suite（或 u）标记来控制默认更新行为。有效值为：

- y. IDT 会下载并使用最新的可用版本。
- n (默认值)。IDT 会使用 suite-id 选项中指定的版本。如果未指定 suite-id，IDT 将使用 tests 文件夹中的最新版本。

如果您未使用 upgrade-test-suite 标记，IDT 会在有更新可用时提示您，并会等待 30 秒让您输入（y 或 n）。如果您未输入信息，则默认为 n 并继续运行测试。

以下示例显示了此功能的常见用例：

自动使用可用于测试组的最新测试。

```
devicetester_linux run-suite -u y --group-id mqtt --pool-id DevicePool1
```

在特定测试套件版本中运行测试。

```
devicetester_linux run-suite -u n --suite-id GGQ_1.0.0 --group-id mqtt --pool-id
 DevicePool1
```

运行时提示更新。

```
devicetester_linux run-suite --pool-id DevicePool1
```

适用于 AWS IoT Greengrass 的 IDT 命令

IDT 命令位于 *<device-tester-extract-location>*/bin 目录中。它们可以用于以下操作：

IDT v3.0.0 and later

`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，则必须指定一个池。
- `upgrade-test-suite`. 控制如何处理测试套件版本更新。从 IDT v3.0.0 开始，IDT 在线检查更新的测试套件版本。有关更多信息，请参阅the section called “测试套件版本”。
- `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。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. 设置对更新 IDT 的提示的响应。如果 IDT 检测到有更新的版本，输入 Y 将会停止执行测试。输入 N 将会继续执行测试。
- update-managed-policy。如果 IDT 检测到用户的托管策略未更新，输入 Y 将会停止执行测试。输入 N 将会继续执行测试。

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
```

IDT v2.3.0 and earlier

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。

run-suite

对某个设备池运行一组测试。

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。

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。可在 *<device-tester-extract-location>*/results/*<execution-id>*/ 中找到这些报告。两个报告都捕获资格测试套件执行的结果。

`awsiotdevicetester_report.xml` 是您提交给 AWS 的资格测试报告，用于在 AWS Partner 设备目录中列出您的设备。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。
- 按照基于设备功能（例如，本地资源访问、影子、MQTT 等）测试的库细分的测试结果。

`GGQ_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。

解读 IDT 报告

`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>` 包含测试执行情况的摘要。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。

`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。

```
<feature name="aws-iot-greengrass-no-container" value="supported" type="required"></
feature>
```

标记为 `optional` 的功能不是资格认证所必需的。以下代码段显示了可选功能。

```
<feature name="aws-iot-greengrass-container" value="supported" type="optional"></
feature>

<feature name="aws-iot-greengrass-hsi" value="not-supported" type="optional"></
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。如果您想要在 AWS Partner 设备目录中列出您的设备，则可以使用此报告作为资格证明。

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 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 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="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 从测试执行生成的日志位于 *<devicetester-extract-location>*/results/*<execution-id>*/logs 中。它会生成两组日志：

`test_manager.log`

从 AWS IoT Device Tester 的 Test Manager 组件生成的日志（例如，与配置、测试序列和报告生成相关的日志）。

<test_case_id>`.log (for example, ota.log)`

测试组的日志，包括来自所测试设备的日志。当测试失败时，将创建一个 tar.gz 文件，其中包含关于该测试的所测试设备日志（例如，`ota_prod_test_1_ggc_logs.tar.gz`）。

有关更多信息，请参阅适用于 AWS IoT Greengrass 的 IDT 问题排查。

使用 IDT 开发和运行自己的测试套件

自 IDT v4.0.0 起，AWS IoT Greengrass 的 IDT 将标准化配置设置和结果格式与测试套件环境相结合，使您可以针对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。

下载适用于 AWS IoT Greengrass 的最新版本的 AWS IoT Device Tester

下载 IDT 的最新版本并将软件提取到文件系统中您具有读取和写入权限的位置。

> **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：

- JSON 配置文件，为 IDT 提供了有关如何执行测试套件的信息。
- 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. 为您的测试套件创建 JSON 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 下载项中包含示例测试套件的源代码。您可以按照本教程所示构建和运行示例测试套件，以了解如何使用 AWS IoT Device Tester 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
  - AWS IoT Device Tester 的最新版本
  - 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。

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。检查 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。

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
```

故障排除

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 JSON 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 JSON 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
  - AWS IoT Device Tester 的最新版本
  - 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 JSON 配置文件

您的测试套件必须包含以下必需的 JSON 配置文件：

所需的 JSON 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
```

故障排除

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 JSON 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。

所需的 JSON 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 JSON 文件

`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 state_machine.json。

`userdata_schema.json`

定义测试运行者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。

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
    ### state_machine.json
    ### 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 中供测试运行者使用。

`details`

测试套件用途的简短描述。

`userDataRequired`

定义测试运行者是否需要在 `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。id 的值必须与 group.json 文件所在的测试组文件夹的名称相匹配，并且不得包含下划线 (_)。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 中供测试运行者使用。

`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。`id` 的值必须与 `test.json` 文件所在的测试用例文件夹的名称相匹配，并且不得包含下划线 (_)。ID 用于 IDT 生成的测试报告中。

`title`

测试用例的描述性名称。该名称显示在 IDT CLI 中供测试运行者使用。

`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`。测试运行者将在其 `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` 文件中测试运行者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` 文件中测试运行者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 state_machine.json

状态机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状态机，IDT 将为您生成状态机。默认状态机执行以下功能：

- 使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 状态机工作原理的更多信息，请参阅 配置 IDT 状态机。

配置 userdata_schema.json

userdata_schema.json 文件确定了测试运行者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。

要指示测试运行者必须提供 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者提供的 userdata.json 文件。如果有效，则 userdata.json 文件内容在 IDT 上下文和状态机上下文中均可用。

配置 IDT 状态机

状态机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状态机，IDT 将为您生成状态机。默认状态机执行以下功能：

- 使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。

IDT 测试套件的状态机必须满足以下条件：

- 每个状态都对应于 IDT 要采取的操作，例如运行测试组或产品报告文件。
- 过渡到状态会执行与该状态关联的操作。
- 每个状态都定义了下一个状态的过渡规则。
- 结束状态必须为 Succeed 或 Fail。

状态机格式

您可以使用以下模板来配置自己的*<custom-test-suite-folder>*/suite/state_machine.json文件：

```
{
  "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。`StartAt` 的值必须设置为 `States` 对象中列出的其中一个状态。

`States`

将用户定义的状态名称映射到有效的 IDT 状态的对象。每个状态。*`state-name`* 对象包含映射到 *`state-name`* 有效状态的定义。

`States` 对象必须包含 `Succeed` 和 `Fail` 状态。有关有效状态的信息，请参阅有效状态和状态定义。

有效状态和状态定义

本节介绍可在 IDT 状态机中使用的所有有效状态的状态定义。以下某些状态支持测试用例级别的配置。但是，除非绝对必要，否则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。

状态定义

- RunTask
- Choice
- Parallel
- AddProductFeatures
- 报告
- LogMessage
- SelectGroup
- Fail
- Succeed

RunTask

RunTask 状态运行测试套件中定义的测试组中的测试用例。

```
{
    "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过渡到的状态的名称。

`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。来自 `TestGroup` 中指定的组的测试用例 ID 数组。IDT 根据 `TestGroup` 和 `TestCases` 的值确定测试执行行为，如下所示：

- 同时指定 `TestGroup` 和 `TestCases` 时，IDT 会运行测试组中的指定测试用例。
- 如果 `TestCases` 已指定但 `TestGroup` 未指定，IDT 将运行指定的测试用例。
- 如果 `TestGroup` 已指定但 `TestCases` 未指定，则 IDT 将运行指定测试组中的所有测试用例。
- 如果未指定 `TestGroup` 或 `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。要为测试运行者启用组选择，必须在 `statemachine.json` 文件中同时包含 `RunTask` 和 `Choice` 状态。有关其工作原理的示例，请参阅状态机示例：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，请不要指定此值 `TestGroup`。基于以下内容，IDT 将您在 `ResultVar` 中定义的变量的值设置为 `true` 或 `false`：

- 如果变量名称的形式为 *`text`*`_`*`text`*`_passed`，则该值设置为第一个测试组中的所有测试均已通过或跳过。
- 在所有其他情况下，该值设置为所有测试组中的所有测试已通过或跳过。

通常，您将使用 `RunTask` 状态来指定测试组 ID，而不指定单个测试用例 ID，这样 IDT 就可以运行指定测试组中的所有测试用例。在此状态下运行的所有测试用例均按随机顺序并行运行。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，则此状态会发出 `RunTaskError` 执行错误。如果状态遇到执行错误，则它还会将状态机上下文中的 `hasExecutionError` 变量设置为 `true`。

Choice

`Choice` 状态允许您根据用户定义的条件动态设置要过渡到的下一个状态。

```
{
```

```
    "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`

如果 `Choices` 中定义的表达式都无法评估到 `true`，则设置要过渡到的默认状态。

`FallthroughOnError`

可选。指定状态在评估表达式时遇到错误时的行为。如果要在评估结果出错时跳过表达式，则设置为 `true`。如果没有匹配的表达式，则状态机将过渡到 `Default` 状态。如果 `FallthroughOnError` 的值未指定，将默认为 `false`。

`Choices`

一组表达式和状态，用于确定在当前状态下执行操作后要过渡到哪个状态。

`Choices.Expression`

计算结果为布尔值的表达式字符串。如果表达式的评估结果为 `true`，则状态机将过渡到 `Choices.Next` 中定义的状态。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，请参见 状态机上下文。

`Choices.Next`

如果 `Choices.Expression` 中定义的表达式的评估结果为 `true`，则设置要过渡到的状态的名称。

错误处理

在以下情况下，`Choice` 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 查询的结果不是字符串、数字或布尔值。

在这种状态下，不能使用 Catch 数据块来处理错误。如果要在状态机遇到错误时停止执行它，则必须将 FallthroughOnError 设置为 false。但是，建议您将 FallthroughOnError 设置为 true，并根据您的用例，执行以下操作之一：

- 如果您正在访问的变量预计在某些情况下不存在，则使用的值 Default 和其他 Choices 数据块来指定下一个状态。
- 如果您正在访问的变量应始终存在，则将 Default 状态设置为 Fail。

Parallel

Parallel 状态允许您并行地定义和运行新的状态机。

```
{
    "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过渡到的状态的名称。

Branches

要运行的状态机定义阵列。每个状态机定义都必须包含自己的StartAt、Succeed 和 Fail 状态。此阵列中的状态机定义不能引用其自身定义之外的状态。

> **Note**
>
> 由于每个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。

只有在 Parallel 运行所有分支状态机之后，该状态才会移动到下一个状态。每种需要设备的状态都将等到设备可用后才运行。如果有多个设备可用，则此状态会并行运行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按随机顺序运行，因此可能会使用不同的设备来运行来自同一个测试组的测试。

错误处理

确保分支状态机和父状态机都过渡到 Fail 状态以处理执行错误。

由于分支状态机不会将执行错误传输到父状态机，因此您不能使用 Catch 数据块来处理分支状态机中的执行错误。相反，在共享状态机上下文中使用 hasExecutionErrors 值。有关如何执行此操作的示例，请参阅 状态机示例：并行运行两个测试组。

AddProductFeatures

AddProductFeatures 状态允许您将产品功能添加到 IDT 生成的 awsiotdevicetester_report.xml 文件中。

产品功能是关于设备可能符合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，根据指定的测试是否通过，将产品功能设置为 supported、not-supported 或自定义值。

> **Note**
>
> AddProductFeatures 状态本身不生成报告。此状态必须过渡到Report状态才能生成报告。

```
{
    "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过渡到的状态的名称。

`Features`

要在 `awsiotdevicetester_report.xml` 文件中显示的一系列产品功能。

`Feature`

功能的名称

`FeatureValue`

可选。要在报告中使用的自定义值，而不是 `supported`。如果未指定此值，则根据测试结果，将特征值设置为 `supported` 或 `not-supported`。

如果您使用 `FeatureValue` 自定义值，则可以在不同的条件下测试相同的功能，IDT 会将支持条件的特征值串联起来。例如，以下摘录显示具有两个独立 `MyFeature` 特征值的要素：

```
...
{
    "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，则特征值将设置为 `first-feature-supported, second-feature-supported`。

`Groups`

可选。测试组 ID 的阵列。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 的阵列。至少一个指定测试组中的所有测试都必须通过才能支持该功能。

`TestCases`

可选。测试用例 ID 的数组。如果您指定此值，则适用以下条件：

- 必须通过所有指定的测试用例才能支持该功能。
- `Groups` 必须仅包含一个测试组 ID。
- `OneOfGroups` 不得指定。

`IsRequired`

可选。设置为 `false` 可在报告中将此功能标记为可选功能。默认值为 `true`。

`ExecutionMethods`

可选。与 `device.json` 文件中指定的 `protocol` 值相匹配的执行方法阵列。如果指定了此值，则测试运行者必须指定与该阵列中的一个 `protocol` 值相匹配的值，才能将该功能包含在报告中。如果未指定此值，则该要素将始终包含在报告中。

要使用 `AddProductFeatures` 状态，必须将 `RunTask` 状态 `ResultVar` 中的值设置为以下值之一：

- 如果您指定了单个测试用例 ID，则将 `ResultVar` 设置为 *`group-id_test-id`*`_passed`。
- 如果您未指定单个测试用例 ID，则将 `ResultVar` 设置为 *`group-id`*`_passed`。

`AddProductFeatures` 状态通过以下方式检查测试结果：

- 如果您未指定任何测试用例 ID，则每个测试组的结果将根据状态机上下文中的 *`group-id`*`_passed` 变量值确定。
- 如果您确实指定了测试用例 ID，则每个测试的结果都是根据状态机上下文中 *`group-id_test-id`*`_passed` 变量的值确定的。

错误处理

如果在此状态下提供的群组 ID 不是有效的组 ID，则此状态会导致 AddProductFeaturesError 执行错误。如果状态遇到执行错误，则它还会将状态机上下文中的 hasExecutionErrors 变量设置为 true。

报告

Report 状态会生成 *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过渡到的状态的名称。

您应始终过渡到测试执行流程即将结束时的 Report 状态，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。

LogMessage

LogMessage 状态生成 test_manager.log 文件并将日志消息流式传输到控制台。

```
{
    "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过渡到的状态的名称。

`Level`

创建日志消息的错误级别。如果您指定的级别无效，则此状态会生成一条错误消息并将其丢弃。

`Message`

要记入日志的消息。

SelectGroup

`SelectGroup` 状态会更新状态机上下文以指示选择了哪些组。任何后续 `Choice` 状态都使用此状态设置的值。

```
{
    "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过渡到的状态的名称。

`TestGroups`

一组将被标记为选中的测试组。对于此阵列中的每个测试组 ID，*group-id*_selected 变量在上下文中都设置为 `true`。请务必提供有效的测试组 ID，因为 IDT 不会验证指定的组是否存在。

Fail

`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

Succeed 状态表示状态机已正确执行。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，其中包含在执行期间可供状态机使用的数据。状态机上下文只能从状态机访问，并且包含决定测试流程的信息。例如，您可以使用 userdata.json 文件中测试运行者配置的信息来确定是否需要运行特定的测试。

状态机上下文使用以下格式：

```
{
    "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 device.json 文件中定义的相应顶级设备池阵列元素中检索。

`userData`

userdata.json 文件中的信息

`config`

信息会锁定 config.json 文件。

`suiteFailed`

该值在状态机启动时设置为 `false`。如果测试组在某种 RunTask 状态下失败，则在状态机执行的剩余时间内，该值将设置为 `true`。

`specificTestGroups`

如果测试运行者选择特定的测试组而不是整个测试套件来运行，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，则会在状态机执行的剩余时间内创建此变量并将其设置为 `true`。

您可以使用 JSONPath 表示法查询上下文。状态定义中 jsonPath 查询的语法是 {{$.*query*}}。在某些状态下，您可以将 JSONPath 查询用作占位符字符串。IDT 将占位符字符串替换为上下文中评估的 jsonPath 查询的值。您可以对占位符使用以下值：

- RunTask状态的 `TestCases` 值。
- `Expression` 值 `Choice` 状态。

当您访问状态机上下文中的数据时，请确保满足以下条件：

- JSON 路径必须以 `$.` 开头
- 每个值的计算结果必须为字符串、数字或布尔值。

有关使用 JSONPath 表示法从上下文中访问数据的更多信息，请参阅 使用 IDT 上下文。

执行错误

执行错误是状态机在执行状态时遇到的状态机定义中的错误。IDT 在 test_manager.log 文件中记录有关每个错误的信息，并将日志消息流式传输到控制台。

您可以使用以下方法来处理执行错误：

- 在状态定义中添加一个Catch 数据块。
- 在状态机上下文中检查 hasExecutionErrors值。

捕获

要使用 Catch，请将以下内容添加到您的状态定义中：

```
"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

要捕获的错误类型的数组。如果执行错误与其中一个指定值匹配，则状态机将转换为 Catch.Next 中指定的状态。有关其产生的错误类型的信息，请参阅每个状态定义。

Catch.Next

当前状态遇到与 Catch.ErrorEquals 中指定值之一相匹配的执行错误时，设置要过渡到的下一个状态。

捕获数据块按顺序处理，直到匹配。如果没有错误与捕获数据块中列出的错误相匹配，则状态机将继续执行。由于执行错误是由状态定义不正确造成的，因此我们建议您在状态遇到执行错误时过渡到失败状态。

hasExecutionError

当某些状态遇到执行错误时，除了发出错误外，它们还会在状态机上下文中将 hasExecutionError 值设置为 true。您可以使用此值来检测何时发生错误，然后使用 Choice 状态将状态机过渡到 Fail 状态。

该方式具有以下特征。

- 状态机不以分配给 hasExecutionError 的任何值启动，并且在特定状态设置该值之前，该值不可用。这意味着必须将访问此值的 Choice 状态的 FallthroughOnError 明确设置为 false，以防止在没有发生执行错误时状态机停止。
- 一旦将其设置为 true，hasExecutionError 就永远不会设置为 false 或将其从上下文中删除。这意味着该值仅在首次设置为 true 时才有用，并且对于所有后续状态，它不会提供有意义的值。
- hasExecutionError 值与处于 Parallel 状态的所有分支状态机共享，这可能会导致意想不到的结果，具体取决于访问该值的顺序。

由于这些特性，如果您可以改用捕获数据块，我们不建议您使用此方法。

示例状态机

本部分提供了一些状态机配置示例。

示例

- 状态机示例：运行单个测试组
- 状态机示例：运行用户选择的测试组
- 状态机示例：使用产品功能运行单个测试组
- 状态机示例：并行运行两个测试组

状态机示例：运行单个测试组

该状态机：

- 运行带有 id GroupA 的测试组，该组必须以 group.json 文件形式存在于套件中。
- 检查是否存在执行错误，如果发现任何错误，则过渡到 Fail。
- 生成报告，如果没有错误，则过渡到 Succeed，否则过渡到 Fail。

```
{
```

```
    "Comment": "Runs a single group and then generates a report.",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

状态机示例：运行用户选择的测试组

该状态机：

- 检查测试运行者是否选择了特定的测试组。状态机不检查特定的测试用例，因为如果不选择测试组，测试运行者就无法选择测试用例。

- 如果选择了测试组：
  - 在选定的测试组中运行测试用例。为此，状态机不会明确指定处于 RunTask 状态中的任何测试组或测试用例。
  - 运行所有测试后生成报告并退出。
- 如果未选择测试组：
  - 在测试组 GroupA 中运行测试。
  - 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"
        }
    }
}
```

状态机示例：使用产品功能运行单个测试组

该状态机：

- 运行测试组 GroupA。
- 检查是否存在执行错误，如果发现任何错误，则过渡到 Fail。
- 将 FeatureThatDependsOnGroupA 功能添加到 awsiotdevicetester_report.xml 文件中：
  - 如果 GroupA 通过，则该功能将设置为 supported。
  - 报告中未将该功能标记为可选。
- 生成报告，如果没有错误，则过渡到 Succeed，否则过渡到 Fail

```
{
    "Comment": "Runs GroupA and adds product features based on GroupA",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "AddProductFeatures",
            "TestGroup": "GroupA",
            "ResultVar": "GroupA_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "AddProductFeatures": {
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
```

```
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

状态机示例：并行运行两个测试组

该状态机：

- 并行运行 GroupA 和 GroupB 测试组。通过分支状态机中的 RunTask 状态存储在上下文中的 ResultVar 变量可供 AddProductFeatures 状态使用。
- 检查是否存在执行错误，如果发现任何错误，则过渡到 Fail。此状态机不使用 Catch 数据块，因为该方法不会检测分支状态机中的执行错误。
- 根据通过的群组向 awsiotdevicetester_report.xml 文件添加功能
  - 如果 GroupA 通过，则该功能将设置为 supported。
  - 报告中未将该功能标记为可选。
- 生成报告，如果没有错误，则过渡到 Succeed，否则过渡到 Fail

如果在设备池中配置了两个设备，则 GroupA 和 GroupB 可以同时运行。但是，如果其中一个 GroupA 或 GroupB 有多个测试，则两个设备都可能分配给这些测试。如果只配置了一台设备，则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在测试套件文件夹中：

- 如果测试套件使用 `test.json` 文件中的参数或环境变量来确定要运行哪些测试，您可以为整个测试套件创建一个测试用例可执行文件，也可为测试套件中的每个测试组分别创建一个测试可执行文件。
- 对于要基于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测试用例可执行文件。

作为测试编写者，您可以决定哪一种方法适合您的用例，并相应地构建测试用例可执行文件。请确保在每个 `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能够正确地运行。

当所有设备都准备好运行测试用例时，IDT 将会读取以下文件：

- 所选测试用例的 `test.json` 决定了要启动的进程以及要设置的环境变量。
- 测试套件的 `suite.json` 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文件进程，并将必要的环境变量传递给这个进程。

使用 IDT 客户端软件开发工具包

通过 IDT 客户端软件开发工具包，您可以使用 API 命令来简化在测试可执行文件中编写测试逻辑的方式，这些命令可用于跟 IDT 和被测设备交互。IDT 当前提供以下开发工具包：

- 适用于 Python 的 IDT 客户端软件开发工具包

- 适用于 Go 的 IDT 客户端软件开发工具包

这些开发工具包位于 *<device-tester-extract-location>*/sdks 文件夹中。创建新的测试用例可执行文件时，您必须将要使用的软件开发工具包复制到包含测试用例可执行文件的文件夹中，并在代码中引用这个软件开发工具包。本节简要描述了可以在测试用例可执行文件中使用的 API 命令。

本节内容

设备交互

通过运行以下命令，您无需实施任何其他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进行读取。

> **Note**
>
> 由于 IDT 不管理使用上下文中设备访问信息与设备建立的直接连接，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，如果这些命令不满足测试用例要求，您可以从 IDT 环境中检索设备访问信息，并使用该信息从测试套件建立与设备的直接连接。
> 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 交互

您可以通过运行以下命令，使测试套件与 IDT 进行通信。

`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 上下文中检索值。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。此命令必须在测试套件中每个测试用例结束时进行调用。

主机交互

您可以通过运行以下命令，使测试套件与主机进行通信。

`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 上下文中检索值。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，并使 IDT 能够管理测试用例可执行文件的生命周期。

启用 IDT CLI 命令

IDT CLI `run-suite` 命令提供了多个选项，允许测试运行者自定义测试执行。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，您需要实施对 IDT CLI 的支持。如果您不实施支持，测试运行者仍然可以运行测试，但是某些 CLI 选项将无法正常运作。为了提供理想的客户体验，我们建议您在 IDT CLI 中实施对以下 `run-suite` 命令参数的支持：

`timeout-multiplier`

指定一个大于 1.0 的值，该值将应用于测试运行期间的所有超时。

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。测试运行者在其 run-suite 命令中指定此参数后，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 config.timeoutMultiplier 字段。要支持这个参数，您必须执行以下操作：

- 读取 IDT_TEST_TIMEOUT 环境变量来获取正确计算的超时值，而不是直接使用 test.json 文件中的超时值。
- 从 IDT 上下文中检索 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 在遇到故障时应当停止运行所有测试。

测试运行者在其 run-suite 命令中指定此参数后，IDT 会在遇到故障时立即停止运行测试。但是，如果测试用例是并行运行的，可能会导致意外的结果。要实施支持，请确保当 IDT 遇到此事件时，您的测试逻辑会指示所有运行中的测试用例停止运行、清理临时资源并向 IDT 报告测试结果。有关失败时提早退出的更多信息，请参阅 指定退出行为。

group-id 和 test-id

指定 IDT 应当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：

- 在指定的测试组中运行所有测试。
- 运行一组来自指定测试组的测试。

为了支持这些参数，测试套件的状态机必须在状态机中包含一组特定的 RunTask 和 Choice 状态。如果您不使用自定义状态机，默认 IDT 状态机包含了所需的状态，您无需采取其他操作。但是，如果您要使用自定义状态机，则可以将 状态机示例：运行用户选择的测试组 作为示例，在状态机中添加所需的状态。

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

在测试运行期间，您可以通过向 stdout 和 stderr 发送数据，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。

IDT 运行完测试套件后，也可以在 *<devicetester-extract-location>*/results/*<execution-id>*/logs 文件夹下的 test_manager.log 文件中找到此信息。

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 *<device-tester-extract-location>*/results/*execution-id*/logs 文件夹下的 *<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这个文件中。IDT 会根据正在运行的测试用例来自动更新路径。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。

此外，也可将文本可执行文件设置为按需在 *<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的前缀，以避免您的文件被覆盖。

向 IDT 报告结果

IDT 将测试结果写入 awsiotdevicetester_report.xml 和 *suite-name*_report.xml 文件中。这些报告文件位于 *<device-tester-extract-location>*/results/*<execution-id>*/ 下。两个报告都捕获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，请参阅 查看 IDT 测试结果和日志

要在 *suite-name*_report.xml 文件中填充内容，您必须在测试执行结束前使用 SendResult 命令向 IDT 报告测试结果。如果 IDT 找不到测试结果，则会针对该测试用例发出错误。以下 Python 摘录显示了用于向 IDT 发送测试结果的命令：

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

如果您不通过 API 报告结果，IDT 会在测试构件文件夹中查找测试结果。此文件夹的路径存储在 IDT 上下文中的 testData.testArtifactsPath 文件中。在此文件夹中，IDT 将找到的第一个按字母顺序排序的 XML 文件用作测试结果。

如果您的测试逻辑生成 JUnit XML 结果，您可以将测试结果写入构件文件夹下的 XML 文件中，以直接向 IDT 提供结果，而不必解析结果后再使用 API 将其提交给 IDT。

如果使用此方法，请确保您的测试逻辑准确汇总了测试结果，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，它会从其他报告的测试组结果计算标签属性。
- testsuites 中必须至少存在一个 testsuite 标签。

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，此方法也可能会导致错误的报告。我们建议您在所有测试用例中对生成的

XML 结果文件使用相同的名称，以覆盖每个测试用例的结果，并将正确的结果提供给 IDT 使用。您可以在测试套件中使用混合方法来进行报告，即：对某些测试用例使用 XML 结果文件，同时通过 API 提交其他测试用例的结果。但是，我们不建议采用这种方法。

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 0 退出，即使测试用例报告失败或错误结果时也不例外。仅使用非零退出代码来表示测试用例未运行，或者表示测试用例可执行文件无法向 IDT 传达任何结果。当 IDT 收到非零退出代码时，这表示测试用例遇到了阻碍其运行的错误。

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。您可以使用此信息来配置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。如果测试运行者使用 `timeout-multiplier` 参数来指定超时乘数，则 IDT 会使用该乘数来计算超时值。

要检测此事件，请使用 IDT_TEST_TIMEOUT 环境变量。当测试运行者启动测试时，IDT 会将 IDT_TEST_TIMEOUT 环境变量的值设置为计算得出的超时值（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相应的计时器。

中断

在测试运行者中断 IDT 时发生。例如，按下 Ctrl+C。

终端会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来检测中断信号即可。

或者，也可以定期轮询 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`。

首次失败时停止

发生的条件为：与当前测试用例并行运行的测试用例失败，并且测试运行者使用了 `stop-on-first-failure` 参数来指定 IDT 应当在遇到任何失败时停止运行。

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。如果 IDT 遇到故障并且已配置为在首次失败时停止，它会将 `CancellationRequested` 布尔值设置为 `true`。

发生其中任何一个事件时，IDT 会等待 5 分钟，让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未能在 5 分钟内退出，IDT 会强制停止它们的每个进程。如果 IDT 没有在进程结束之前收到测试结果，它会将测试用例标记为已超时。作为一种最佳实践，您应确保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. 清理所有的临时资源，例如被测设备上的测试构件。
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

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。这些数据被称为 IDT 上下文。例如，测试运行器在 `userdata.json` 文件中提供的用户数据配置可用于 IDT 环境中的测试套件。

IDT 上下文可以被视为只读 JSON 文档。测试套件可以使用对象、阵列、数字等标准 JSON 数据类型从上下文中检索数据并将数据写入上下文。

上下文架构

IDT 上下文采用以下格式：

```
{
    "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

config.json文件中的信息。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。此值由 IDT CLI 中的测试运行器指定。默认值为 1。

device

有关为测试运行选择的设备的信息。此信息等同于所选设备device.json文件中的 devices 阵列元素。

devicePool

有关为测试运行选择的设备池的信息。此信息等同于 device.json 文件中为所选设备池定义的顶级设备池阵列元素。

resource

resource.json 文件中有关资源设备的信息。

resource.devices

此信息等同于 resource.json 文件中定义的 devices 阵列。每个 devices 元素都包括以下附加字段：

resource.device.name

资源的名称。此值设置为 test.json 文件中的 requiredResource.name 值。

`testData.awsCredentials`

测试用于连接到 AWS 云的 AWS 凭证。此信息是从 `config.json` 文件中获得的。

`testData.logFilePath`

测试用例写入日志消息的日志文件的路径。如果日志文件不存在，则会创建。

`userData`

`userdata.json`文件中由测试运行器提供的信息。

在上下文中访问数据

您可以使用 JSON 文件中的 JSONPath 表示法以及带 `GetContextValue` 和 `GetContextString` API 的文本可执行文件查询上下文。访问 IDT 上下文的 JSONPath 字符串的语法各不相同，如下所示：

- 在 `suite.json` 和`test.json` 中，使用 {{*query*}}。也就是说，不要使用根元素 `$.` 开始表达式。
- 在 `statemachine.json` 中，使用 {{`$.`*query*}}。
- 在 API 命令中，根据命令的不同，您可以使用 *query* 或 {{`$.`*query*}}。有关更多信息，请参阅开发工具包中的内联文档。

下表描述典型 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 `childName` from an object. If applied to an array, yields a new array with this operator applied to each element. The element name is case sensitive. For example, the query to access the `awsRegion` value in the `config` object is `$.config.awsRegion`.

Operator	Description
`[start:end]`	Filters elements from an array, retrieving items beginning from the `start` index and going up to the `end` index, both inclusive.
`[index1, index2, ... , indexN]`	Filters elements from an array, retrieving items from only the specified indices.
`[?(expr)]`	Filters elements from an array using the `expr` expression. This expression must evaluate to a boolean value.

要创建筛选表达式，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` 下列运算符之一：
  - < (小于)
  - <= (小于或等于)
  - == (等于)

    如果表达式中的 jsonPath 或值是阵列、布尔值或对象值，则这是您可以使用的唯一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，而右侧必须是遵循 RE2 语法的模式值。

您可以使用 {{*query*}} 形式的 JSONPath 查询作为 `test.json` 文件 `args` 和`environmentVariables` 字段以及 `suite.json` 文件 `environmentVariables` 字段中的占位

符字符串。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变化的环境变量值，IDT 将使用每个测试用例的正确值填充环境变量。但是，当您在 test.json 和 suite.json 文件中使用占位符字符串时，以下注意事项适用于您的查询：

- 查询中每次出现的 devicePool 密钥都必须全部使用小写字母。也就是说，改用 devicepool。
- 对于阵列，只能使用字符串阵列。此外，阵列使用非标准 item1, item2,...,itemN 格式。如果阵列仅包含一个元素，则将其序列化为 item，使其与字符串字段没有区别。
- 不能使用占位符从上下文中检索对象。

出于这些考虑，我们建议您尽可能使用 API 来访问测试逻辑中的上下文，而不是 test.json 和 suite.json 文件中的占位符字符串。但是，在某些情况下，使用 JsonPath 占位符检索要设置为环境变量的单个字符串可能会更方便。

为测试运行者配置设置

要运行自定义测试套件，测试运行者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 JSON 配置文件模板指定的。如果需要，测试运行者还必须设置 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。您必须向测试运行者提供说明，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）。

测试运行者可以使用位于 *<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。您必须向测试运行者提供有关要包含在 `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。测试运行者提供此功能名称，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`查询从状态机上下文中检索所`pool.features`提供的信息。

`features.name`

功能的名称。

`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 docker 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 docker 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，可以从状态机上下文中检索 pool.Devices[0].Connectivity.Protocol，并将其与 Choice 状态下的预期值进行比较。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，测试运行者使用 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。

测试运行者可以使用位于 *<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-value>",
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",
```

```
                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

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 `docker` 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 `docker` 时适用。

（可选）配置 config.json

`config.json` 文件包含 IDT 的配置信息。通常，测试运行者无需修改此文件，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 设备测试人员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IDT 使用量指标。

测试运行者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，测试运行者需要编辑 *<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，如下所述：

> **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。运行以下命令查看以下 `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状态机；相反，它会与您的 IDE 交互以响应从 IDE 中运行的测试套件所发出的请求，并将日志打印到控制台。IDT 不会超时，而会等待退出，直到手动中断。在调试模式下，IDT 也不运行状态机，并且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，则测试运行者必须指定一个池。
- `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。以下摘录显示了一个由 IDT 生成的控制台消息示例。

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

大多数控制台消息都包含以下字段：

time

所记录事件的完整 ISO 8601 时间戳。

level

所记录事件的消息级别。通常，记录的消息级别为 info、warn 或 error。如果遇到导致其提前退出的预期事件，IDT 会发出 fatal 或 panic 消息。

msg

记录的消息。

executionId

当前 IDT 流程的唯一 ID 字符串。此 ID 用于区分各个 IDT 运行。

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更多信息。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
```

控制台消息中特定于测试套件的部分包含以下字段：

`suiteId`

当前正在运行的测试套件的名称。

`groupId`

当前正在运行的测试组的 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被测设备的 ID。

要在 IDT 完成测试运行后将测试摘要打印到控制台，您必须在状态机中包含一个Report 状态。测试摘要包含有关测试套件、每个已运行组的测试结果以及生成的日志和报告文件位置的信息。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 设备测试器报告架构

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。
- 用于对报告进行签名的报告签名和密钥。
- device.json 文件中指定的设备 SKU 和设备池名称。
- 经过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 *suite-name*_report.xml 文件中包含的信息相同。

```
<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 的功能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 的功能不是验证所必需的。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。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>
```

awsiotdevicetester_report.xml 或 *suite-name*_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>
```

IDT 使用量指标

如果您提供具有所需权限的 AWS 凭证， AWS IoT 设备测试人员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。IDT 将收集以下信息：

- 用于运行 AWS 账户 IDT 的 ID
- 用于运行测试的 IDT 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，例如测试是通过、失败、遇到错误，还是已被跳过
- 测试的产品功能
- 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 *<device-tester-extract-location>*/results/*<execution-id>*/ 文件夹下的 metrics.log 文件中。您可以查看日志文件以检查在测试运行期间收集的信息。只有选择了收集使用量指标后，才会生成此文件。

要禁用指标收集，您无需采取其他操作。请不要存储您的 AWS 凭据，如果您确实存储了 AWS 凭据，也不要将 config.jso n 文件配置为访问它们。

配置您的 AWS 凭证

如果您还没有 AWS 账户，则必须创建一个。如果您已经拥有 AWS 账户，则只需为您的账户配置所需的权限，以允许 IDT 代表您向其发送使用量指标。 AWS

步骤 1：创建一个 AWS 账户

在此步骤中，将创建并配置 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”。

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 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。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。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，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后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。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。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. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

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《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 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。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

步骤 2：为 IDT 配置权限

在此步骤中，将配置 IDT 运行测试和收集 IDT 使用情况数据所需的权限。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 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。

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

```
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}
```

c. 选择下一步：标签。

d. 选择下一步：审核。

e. 对于名称，请输入 **IDTUsageMetricsIAMPermissions**。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 用户并将权限附加到该用户。

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。如果您已创建 IAM 用户，请跳到下一步。

b. 将权限附加到您的 IAM 用户：

i. 在 设置权限 页面上，选择 直接附加现有策略。

ii. 搜索您在上一步中创建的 IDT UsageMetrics iamP ermissions 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。稍后您将使用此信息来配置您的 AWS 凭据文件。

为 IDT 配置权限 (AWS CLI)

按照以下步骤 AWS CLI 使用配置 IDT 的 AWS IoT Greengrass权限。如果您已在控制台中配置权限，请跳转至 the section called “配置设备以运行 IDT 测试” 或 the section called “可选：配置 Docker 容器”。

1. AWS CLI 如果尚未安装，请在您的计算机上进行安装和配置。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，可用于通过命令行 shell 与 AWS 服务进行交互。

2. 创建以下客户托管策略，授予管理 IDT 和 AWS IoT Greengrass 角色的权限。

   Linux, macOS, or Unix

   ```
   aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
   document '{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "iot-device-tester:SendMetrics"
               ],
               "Resource": "*"
           }
       ]
   }'
   ```

   Windows command prompt

   ```
   aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
   document
                                           '{\"Version\": \"2012-10-17\",
    \"Statement\": [{\"Effect\": \"Allow\", \"Action\": [\"iot-device-
   tester:SendMetrics\"], \"Resource\": \"*\"}]}'
   ```

   > **Note**
   >
   > 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。

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。

```
aws iam create-user --user-name user-name
```

b. 将您创建的 IDTUsageMetricsIAMPermissions 策略附加到 IAM 用户。将命令中的 *user-name* 替换为您的 IAM 用户名，并将 *<account-id>* 替换为您的 AWS 账户的 ID。

```
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。稍后您将使用此信息来配置您的 AWS 凭据文件。

向 ID AWS T 提供凭证

要允许 IDT 访问您的 AWS 凭证并向其提交指标 AWS，请执行以下操作：

1. 将您的 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：

```
AWS_ACCESS_KEY_ID=access-key
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 file:：

```
[profile-name]
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

2. 配置 config.json 文件的 auth 部分。有关更多信息，请参阅 （可选）配置 config.json。

适用于 AWS IoT Greengrass 的 IDT 问题排查

适用于 AWS IoT Greengrass 的 IDT 根据错误类型将这些错误写入各种位置。错误将被写入到控制台、日志文件和测试报告。

错误代码

下表列出了由适用于 AWS IoT Greengrass 的 IDT 生成的错误代码。

错误代码	错误代码名称	可能的根本原因	故障排查
101	InternalError	出现内部错误。	检查 *<device-tester-extract-location>* / results 目录下的日志。如果您无法调试该问题，请联系 AWS 开发人员支持。
102	TimeoutError	测试无法在限定的时间范围内完成。在以下条件下会出现这种情况： • 测试计算机与设备之间的网络连接速度很慢（例如，如果您使用的是 VPN 网络）。 • 慢速网络延迟了设备和云之间的通信。 • 测试配置文件 (test.json) 中的 timeout 字段被错误地修改。	• 检查网络连接和速度。 • 确保您未修改 / test 目录下的任何文件。 • 尝试使用 "--group-id" 标志手动运行失败的测试组。 • 尝试通过增加测试超时来运行测试套件。有关更多信息，请参阅超时错误。
103	PlatformNotSupportError	device.json 中指定的操作系统/架构组合不正确。	将您的配置更改为支持的组合之一：

错误代码	错误代码名称	可能的根本原因	故障排查
			• Linux，x86_64 • Linux，ARMv6l • Linux，ARMv7l • Linux，AArch64 • Ubuntu，x86_64 • OpenWRT，ARMv7l • OpenWrt，AArch64 有关更多信息，请参阅配置 device.json。

错误代码	错误代码名称	可能的根本原因	故障排查
104	VersionNotSupportError	您使用的 IDT 版本不支持 AWS IoT Greengrass 核心软件版本。	使用 device_tester_bin version 命令查找支持的 AWS IoT Greengrass 核心软件版本。例如，如果您使用的是 macOS，请使用 ./devicetester_mac_x86_64 version。 查找您使用的 AWS IoT Greengrass Core 软件的版本： • 如果您使用预安装的 AWS IoT Greengrass Core 软件运行测试，请使用 SSH 连接到您的 AWS IoT Greengrass 核心设备并运行 ***<path-to-preinstalled-greengrass-location>*** /greengrass/ggc/core/greengrassd --version • 如果您使用其他版本的 AWS IoT Greengrass Core 软件运行测试，请转到 devicetester_green

错误代码	错误代码名称	可能的根本原因	故障排查
			grass_<os>/products/greengrass/gcc 目录。AWS IoT Greengrass 核心软件版本是 .zip 文件名的一部分。 您可以测试不同版本的 AWS IoT Greengrass 核心软件。有关更多信息，请参阅入门 AWS IoT Greengrass。

错误代码	错误代码名称	可能的根本原因	故障排查
105	LanguageNotSupportError	IDT 仅对于 AWS IoT Greengrass 库和开发工具包支持 Python。	确保： • devicetester_greengrass_ *<os>*/products/greengrass/ggsdk 下的开发工具包是 Python 开发工具包。 • devicetester_greengrass_ *<os>* /tests/GGQ_1.0.0/suite/resources/runtimearm/bin 下的 bin 文件夹中的内容尚未更改。

错误代码	错误代码名称	可能的根本原因	故障排查
106	ValidationError	`device.json` 或 `config.json` 中的有些字段无效。	检查报告中错误代码右侧的错误消息。 • 设备的身份验证类型无效：指定连接到设备的正确方法。有关更多信息，请参阅the section called “配置 device.json”。 • 私有密钥路径无效：指定私有密钥的正确路径。有关更多信息，请参阅配置 device.json。 • 无效的 AWS 区域：在您的 `config.json` 文件中指定有效的 AWS 区域。有关更多信息，请参阅AWS服务端点。 • AWS 凭证：在测试计算机上设置有效的 AWS 凭证（使用环境变量或 `credentials` 文件进行设置）。验证 `auth` 字段是否配置正确。有关更多信息，请参阅the

错误代码	错误代码名称	可能的根本原因	故障排查
			section called “创建和配置 AWS 账户”。 • 无效的 HSM 输入：检查 device.json 中的 p11Provider 、privateKeyLabel 、slotLabel 、slotUserPin 和 openSSLEngine 字段。
107	SSHConnectionFailed	测试计算机无法连接到已配置的设备。	验证 device.json 文件中的以下字段是否正确： • ip • user • privKeyPath • password 有关更多信息，请参阅配置 device.json。

错误代码	错误代码名称	可能的根本原因	故障排查
108	RunCommandError	测试在测试中的设备上执行命令失败。	验证是否允许在 `device.json` 中配置的用户拥有根访问权限。 当通过根访问权限执行命令时，某些设备需要密码。确保在使用不密码的情况下允许获得根访问权限。有关更多信息，请参阅设备的文档。 尝试在您的设备上手动运行失败的命令以查看是否发生错误。
109	PermissionDeniedError	无根访问权限。	在您的设备上为配置的用户设置根访问权限。
110	CreateFileError	无法创建文件。	检查您设备的磁盘空间和目录权限。
111	CreateDirError	无法创建目录。	检查您设备的磁盘空间和目录权限。
112	InvalidPathError	指向 AWS IoT Greengrass 核心软件的路径不正确。	验证错误消息中的路径是有效的。请勿编辑 `devicetester_greengrass_` *`<os>`* 目录下的任何文件。
113	InvalidFileError	文件无效。	验证错误消息中的文件是否有效。

错误代码	错误代码名称	可能的根本原因	故障排查
114	ReadFileError	无法读取指定的文件。	请验证以下内容： • 文件权限正确无误。 • `limits.config` 允许打开足够的文件。 • 错误消息中指定的文件存在，并且是有效的。 如果您正在 macOS 上进行测试，请增加打开文件数量限制。默认限制为 256，这足够用于测试。

错误代码	错误代码名称	可能的根本原因	故障排查
115	FileNotFoundError	找不到所需的文件。	请验证以下内容： • 压缩的 Greengrass 文件位于 devicetester_greengrass_ *<os>*/products/greengrass/ggc 下。您可以从 AWS IoT Greengrass核心软件下载页面下载 AWS IoT Greengrass 核心 tar 文件。 • 开发工具包存在于 devicetester_greengrass_ *<os>*/products/greengrass/ggsdk 下。 • devicetester_greengrass_ *<os>*/tests 下的文件未经修改。

错误代码	错误代码名称	可能的根本原因	故障排查
116	OpenFileFailed	无法打开指定的文件。	请验证以下内容： • 错误消息中指定的文件存在，并且是有效的。 • `limits.config` 允许打开足够的文件。 如果您正在 macOS 上进行测试，请增加打开文件数量限制。默认限制为 256，这足够用于测试。
117	WriteFileFailed	无法写入文件（可以是 DUT 或测试机）。	验证错误消息中指定的目录是否存在，并且您是否具有写入权限。
118	FileCleanUpError	测试未能删除指定的文件或目录，或未能在远程设备上卸载指定的文件。	如果二进制文件仍在运行，则文件可能被锁定。结束进程并删除指定的文件。
119	InvalidInputError	配置无效。	验证 `suite.json` 文件是否有效。

错误代码	错误代码名称	可能的根本原因	故障排查
120	InvalidCredentialError	AWS 凭证无效。	• 验证您的 AWS 凭证。有关更多信息，请参阅the section called “配置 AWS凭证”。 • 检查您的网络连接并重新运行测试组。网络问题也可能导致此错误。
121	AWSSessionError	未能创建 AWS 会话。	如果 AWS 凭证无效或 Internet 连接不稳定，也可能会发生此错误。尝试使用 AWS CLI 来调用 AWS API 操作。
122	AWSApiCallError	发生了 AWS API 错误。	此错误可能是由于网络问题导致的。检查您的网络，然后重试测试组。

错误代码	错误代码名称	可能的根本原因	故障排查
123	IpNotExistError	IP 地址未包含在连接信息中。	检查您的 Internet 连接。您可以使用 AWS IoT Greengrass 控制台检查测试正在使用的 AWS IoT Greengrass 核心事物的连接信息。如果连接信息中包含 10 个端点，则您可以删除部分或所有端点并重新运行测试。有关更多信息，请参阅连接信息。
124	OTAJobNotCompleteError	一个 OTA 作业未完成。	检查您的 Internet 连接并重试 OTA 测试组。
125	CreateGreengrassServiceRoleError	发生了以下错误之一： • 创建角色时出错。 • 将策略附加到 AWS IoT Greengrass 服务角色时出错。 • 与服务角色关联的策略无效。 • 将角色与 AWS 账户关联时出错。	配置 AWS IoT Greengrass 服务角色。有关更多信息，请参阅the section called “Greengrass 服务角色”。

错误代码	错误代码名称	可能的根本原因	故障排查
126	DependenciesNotPresentError	设备上不存在特定测试所需的一个或多个依赖项。	检查测试日志以查看该设备上缺少哪些依赖项：*<device-tester-extract-location>* /results/ *<execution-id>*/logs/*<test-case-name.log>*
127	InvalidHSMConfiguration	提供的 HSM/PKCS 配置不正确。	在 device.json 文件中，提供使用 PKCS＃11 与 HSM 交互所需的配置。

错误代码	错误代码名称	可能的根本原因	故障排查

128	OTAJobNotSuccededError	该 OTA 作业没有获得成功。	• 如果单独运行 ota 测试组，请运行 ggcdependencies 测试组以验证是否存在所有依赖项（例如 wget）。然后重试 ota 测试组。
			• 查看 *<device-tester-extract-location>* /results/ *<execution-id>*/logs/ 下面的详细日志，了解故障排除和错误信息。具体来说，请查看以下日志：
			  • 控制台日志 (test_manager.log)
			  • OTA 测试用例日志 (ota_test.log)
			  • GGC 守护程序日志 (ota_test_ggc_logs.tar.gz)
			  •

错误代码	错误代码名称	可能的根本原因	故障排查
			OTA 代理日志 (`ota_test_ota_logs.tar.gz`) • 检查您的 Internet 连接并重试 `ota` 测试组。 • 如果问题仍然存在，请联系 AWS 开发人员支持。
129	NoConnectivityError	主机代理无法连接到 Internet。	请检查您的网络连接和防火墙设置。连接问题得到解决后，重试此测试组。
130	NoPermissionError	您用于运行适用于 AWS IoT Greengrass 的 IDT 的 IAM 用户不具备创建运行 IDT 所需的 AWS 资源的权限。	请参阅权限策略模板，以了解授予运行适用于 AWS IoT Greengrass 的 IDT 所需的权限的策略模板。

错误代码	错误代码名称	可能的根本原因	故障排查
131	LeftoverAgentExistError	当您尝试启动适用于 AWS IoT Greengrass 的 IDT 时，设备正在运行 AWS IoT Greengrass 进程。	确保您的设备上没有正在运行的 Greengrass 守护程序。 • 您可以使用此命令来停止守护程序：sudo ./*<absolute-path-to-greengrass-daemon>* / greengrassd stop。 • 您也可以通过 PID 终止 Greengrass 守护程序。 **Note** 如果您使用的是已配置为在重新引导后自动启动的现有 AWS IoT Greengrass 安装，则必须在重新引导之后、运行测试套件之前停止守护程序。
132	DeviceTimeOffsetError	设备的时间不正确。	将设备设置为正确的时间。

错误代码	错误代码名称	可能的根本原因	故障排查
133	InvalidMLConfiguration	提供的 ML 配置不正确。	在 `device.json` 文件中，提供运行 ML 推理测试所需的正确配置。有关更多信息，请参阅the section called “可选：配置设备进行 ML 资格认证”。

纠正适用于 AWS IoT Greengrass 的 IDT 错误

在使用 IDT 时，您必须先获取正确的配置文件，然后再运行适用于 AWS IoT Greengrass 的 IDT。如果出现解析和配置错误，第一步应找到并使用适合您的环境的配置模板。

如果仍有问题，请参阅以下调试过程。

主题

应该在哪里寻找错误？

在执行期间，控制台上会显示高级别错误，并且在所有测试完成后会显示失败测试及其错误的摘要。`awsiotdevicetester_report.xml` 包含导致测试失败的所有错误的摘要。每次测试运行的日志文件都存储在一个以 UUID 命名的目录中，用于在测试运行期间在控制台上显示的测试执行。

测试日志目录位于 *<device-tester-extract-location>*/results/*<execution-id>*/logs/。此目录包含以下对调试有用的文件。

文件	描述
test_manager.log	在测试执行期间写入控制台的所有日志。结果的摘要位于该文件的末尾，其中包含失败的测试的列表。 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-name>* .log	特定测试的详细日志。
<test-name> _ggc_logs.tar.gz	AWS IoT Greengrass 核心守护程序在测试期间生成的所有日志的压缩集合。有关更多信息，请参阅故障排除AWS IoT Greengrass。
<test-name> _ota_logs.tar.gz	AWS IoT Greengrass OTA 代理在测试期间生成的日志的压缩集合。仅适用于 OTA 测试。
<test-name> _basic_assertion_publisher_ggad_logs.tar.gz	测试期间由 AWS IoT 发布者设备生成的日志的压缩集合。
<test-name> _basic_assertion_subscriber_ggad_logs.tar.gz	测试期间由 AWS IoT 订阅者设备生成的日志的压缩集合。

解析错误

有时候，JSON 配置中的输入错误会导致解析错误。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。这些信息应足以帮助您修复错误，但是如果仍不能定位错误，则可以通过您的 IDE、文本编辑器（例如 Atom 或 Sublime）或 JSONLint 等在线工具手动执行验证。

缺少必需参数错误

由于将新功能添加到 IDT 中，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 还将验证 JSON 配置文件架构，以确保使用支持的最新版本。

无法启动测试错误

在测试启动期间，您可能遇到指示失败的错误。有几种可能的原因，因此，请执行以下操作：

- 确保包含在执行命令中的池名称实际存在。池名称将从您的 device.json 文件中直接引用。
- 确保池中的设备具有正确的配置参数。

无权访问资源错误

您可能会在终端输出或 /results/*<execution-id>*/logs 下的 test_manager.log 文件中看到 <user or role> is not authorized to access this resource 错误消息。要解决此问题，请将 AWSIoTDeviceTesterForGreengrassFullAccess 托管策略附加到您的测试用户。有关更多信息，请参阅the section called “创建和配置 AWS 账户”。

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。

SSH 连接错误

当 IDT 无法连接到所测试设备时，会在 `/results/<execution-id>/logs/<test-case-id>.log` 中记录连接失败。SSH 失败消息将显示在此日志文件的顶部，因为连接到所测试设备是 IDT 最先执行的操作之一。

大多数 Windows 设置使用 PuTTy 终端应用程序连接到 Linux 主机。该应用程序要求将标准 PEM 私有密钥文件转换为称为 PPK 的专有 Windows 格式。在 `device.json` 文件中配置 IDT 时，仅使用 PEM 文件。如果使用 PPK 文件，IDT 将无法与 AWS IoT Greengrass 设备建立 SSH 连接，并且无法运行测试。

超时错误

您可以通过指定超时乘数来增加每个测试的超时，该超时乘数应用于每个测试超时的默认值。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 GGQ_1.0.0 --pool-id DevicePool1 --timeout-multiplier 2.5
```

有关更多信息，请运行 `run-suite --help`。

测试时出现找不到命令错误

您需要较旧版本的 OpenSSL 库 (libssl1.0.0) 才能在 AWS IoT Greengrass 设备上运行测试。当前的大多数 Linux 发行版都使用 libssl 1.0.2 或更高版本 (v1.1.0)。

例如，在 Raspberry Pi 上，可以运行以下命令来安装所需的 libssl 版本：

1. ```
   wget http://ftp.us.debian.org/debian/pool/main/o/openssl/libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
   ```
2. ```
   sudo dpkg -i libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
   ```

macOS 上的安全例外

当在使用 macOS 10.15 的主机上运行 IDT 时，系统无法正确检测到 IDT 的公证票证，而且 IDT 也会被阻止运行。要运行 IDT，您需要为 `devicetester_mac_x86-64` 可执行文件授予安全例外。

为 IDT 可执行文件授予安全例外

1. 从 Apple 菜单中启动系统偏好设置。
2. 选择 安全与隐私，然后在通用选项卡上，单击小锁图标以更改安全设置。
3. 查找以下消息并选择仍然允许：`"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.`。
4. 接受安全警告。

如果您对 IDT 支持策略有疑问，请联系 AWS 客户支持。

适用于 AWS IoT Greengrass V1 的 AWS IoT Device Tester 的支持策略

AWS IoT适用于 AWS IoT Greengrass 的Device Tester (IDT) 是一个可下载的测试框架，使您能够验证和确认您的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 IDT 来测试您的设备或核定其资格。有关更多信息，请参阅 AWS IoT Greengrass Version 2 开发人员指南中的支持的适用于 AWS IoT Greengrass V2 的 IDT 版本。

您还可以使用任何受支持的 AWS IoT Greengrass 和 IDT 版本来测试您的设备或核定其资格。尽管您可以继续使用不受支持的 IDT 版本，但这些版本不会收到错误修复或更新。

> **Important**
>
> 自 2022 年 4 月 4 日起，AWS IoT Device Tester (IDT) for AWS IoT Greengrass V1 不再生成带签名的资格报告。您将无法再通过 AWS 设备资格认证计划将新的 AWS IoT Greengrass V1 设备列入 AWS Partner 设备目录。虽然您无法获得 Greengrass V1 设备的资格认证，但可以继续使用适用于 AWS IoT Greengrass V1 的 IDT 来测试您的 Greengrass V1 设备。我们建议您使用适用于 AWS IoT Greengrass V2 的 IDT 来进行资格认证并在 AWS Partner 设备目录中列示 Greengrass 设备。

如果您对支持策略有疑问，请联系 AWS 客户支持。

故障排除 AWS IoT Greengrass

本节提供故障排除信息和可能的解决方案，以帮助解决 AWS IoT Greengrass 的问题。

有关 AWS IoT Greengrass 配额（限值）的信息，请参阅 Amazon Web Services 一般参考 中的服务限额。

AWS IoT Greengrass Core 问题

如果 AWS IoT Greengrass 核心软件无法启动，请尝试以下常规故障排除步骤：

- 确保安装适合您的架构的二进制文件。有关更多信息，请参阅 AWS IoT Greengrass 核心软件。
- 确保您的核心设备有可用的本地存储。有关更多信息，请参阅 the section called “排查存储问题”。
- 检查 `runtime.log` 和 `crash.log` 是否有错误消息。有关更多信息，请参阅 the section called “使用日志排查问题”。

搜索以下症状和错误，以查找信息来帮助排查 AWS IoT Greengrass 核心的问题。

问题

- 错误：配置文件缺少 CaPath、 CertPath 或 KeyPath。具有 [pid = <pid>] 的 Greengrass 守护程序进程终止。
- 错误：无法解析 /<greengrass-root>/config/config.json。
- 错误：生成 TLS 配置时出错： ErrUnknownuriScheme
- 错误：运行时未能启动：无法启动工作线程：容器测试超时。
- <address>错误：无法在本地 Cloudwatch PutLogEvents 上调用，logGroup: GreengrassSystem // connection_manager，错误:: 发送请求失败原因 RequestError是：Post http:// <path>cloudwatch/logs/：拨打 tcp：getsockopt：连接被拒绝，响应：{}。
- 错误：Unable to create server due to: failed to load group: chmod /<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda:<region>:<account-id>:function:<function-name>:<version>/<file-name>: no such file or directory。
- 在您从无容器化的情况下运行更改为在 Greengrass 容器中运行后，AWS IoT Greengrass 核心软件无法启动。
- 错误：后台打印大小应至少为 262144 字节。

- 错误：“[ERROR]-Cloud messaging error: Error occurred while trying to publish a message. {"errorString": "operation timed out"} ([ERROR]-云消息传递错误: 尝试发布消息时出错。{"errorString": "操作超时"})”
- 错误：container_linux.go:344: starting container process caused "process_linux.go:424: container init caused \"rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\\\"\"。
- 错误：Greengrass 守护程序在 PID 为 <process-id> 的情况下运行。某些系统组件无法启动。检查“runtime.log”中有无错误。
- 设备影子未与云同步。
- 错误：无法接受 TCP 连接。接受 tcp [::]:8000: accept4：打开的文件太多。
- 错误：运行时执行错误：无法启动 lambda 容器。container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"。
- 警告：[警告]-[5] GK Remote：检索公钥数据时出错 ErrPrincipalNotConfigured:: 未设置的 MqttCertificate 私钥。
- 错误：在试图使用角色 arn:aws:iam::<account-id>:role/<role-name> 访问 s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz 时权限被拒绝。
- AWS IoT Greengrass 核心配置为使用网络代理，您的 Lambda 函数无法进行传出连接。
- 核心处于无限的连接-断开循环中。runtime.log 文件包含一系列连续的连接和断开条目。
- 错误：无法启动 lambda 容器。container_linux.go:259：启动容器进程导致出现 "process_linux.go:345：容器初始化导致出现 \"rootfs_linux.go:62：正在将 \\\"proc\\\" 装载到 \\\"
- [ERROR] 运行时执行错误：无法启动 lambda 容器。{"errorString": "无法初始化容器装载：无法对叠加层上层目录中的 greengrass 根进行掩码处理：无法在目录 <ggc-path> 中创建掩码设备：文件已存在"}
- [ERROR]-部署失败。 {"deploymentId": "<deployment-id>", "errorString": "PID 为 <pid> 的容器测试进程失败: 容器进程状态: 出口状态 1"}
- 错误：[ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to create overlay fs for container: mounting overlay at /greengrass/ggc/packages/<ggc-version>/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/

ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}

- 错误：[DEBUG] – 未能获取路由。丢弃消息。
- 错误：[Errno 24] Too many open <lambda-function>,[Errno 24] Too many open files
- 错误: ds 服务器无法开始侦听套接字: listen unix <ggc-path>/ggc/socket/greengrass_ipc.sock: bind: 参数无效
- [信息]（复印机）aws.greengrass。 StreamManager: stdout。由：com.fasterxml.jackson.databind 引起。 JsonMappingException: 即时超过最小或最大瞬间
- GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

错误：配置文件缺少 CaPath、 CertPath 或 KeyPath。具有 [pid = <pid>] 的 Greengrass 守护程序进程终止。

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会在 `crash.log` 中看到此错误。如果您正在运行 v1.6 或更早版本，可能会发生此错误。请执行以下操作之一：

- 升级至 v1.7 或更高版本。我们建议您始终运行最新版本的 AWS IoT Greengrass 核心软件。有关下载信息，请参阅 AWS IoT Greengrass 核心软件。
- 为您的 AWS IoT Greengrass 核心软件版本使用正确的 `config.json` 格式。有关更多信息，请参阅 the section called “AWS IoT Greengrass 核心配置文件”。

> **Note**
>
> 要查找核心设备上安装了哪一版本的 AWS IoT Greengrass Core 软件，请在您的设备终端上运行以下命令。
>
> ```
> cd /greengrass-root/ggc/core/
> sudo ./greengrassd --version
> ```

错误：无法解析 /<greengrass-root>/config/config.json。

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会看到此错误。确保 Greengrass 配置文件使用了有效的 JSON 格式。

打开 `config.json`（位于 */greengrass-root*`/config` 中），并验证 JSON 格式。例如，请确保逗号使用正确。

错误：生成 TLS 配置时出错： ErrUnknownuriScheme

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会看到此错误。确保 Greengrass 配置文件的 crypto 部分中的属性是有效的。错误消息应提供更多信息。

打开 `config.json`（位于 */greengrass-root*`/config` 中），并检查 `crypto` 部分。例如，证书和密钥路径必须使用正确的 URI 格式并指向正确的位置。

错误：运行时未能启动：无法启动工作线程：容器测试超时。

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会看到此错误。在 Greengrass 配置文件中设置 `postStartHealthCheckTimeout` 属性。此可选属性将配置 Greengrass 守护程序等待启动后运行状况检查完成的时间量（以毫秒为单位）。默认值为 30 秒（30000 毫秒）。

打开 `config.json`（位于 */greengrass-root*`/config` 中）。在 `runtime` 对象中，添加 `postStartHealthCheckTimeout` 属性并将值设置为一个大于 30000 的数。在需要的位置添加逗号以创建有效的 JSON 文档。例如：

```
...
"runtime" : {
  "cgroup" : {
    "useSystemd" : "yes"
  },
  "postStartHealthCheckTimeout" : 40000
},
...
```

<address>错误：无法在本地 Cloudwatch PutLogEvents 上调用，logGroup: GreengrassSystem //connection_manager，错误:: 发送请求失败原因 RequestError是：Post http:// <path>cloudwatch/logs/：拨打 tcp：getsockopt：连接被拒绝，响应：{}。

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会看到此错误。如果您在 Raspberry Pi 上运行 AWS IoT Greengrass，并且尚未完成所需的内存设置，则可能会发生此错误。有关更多信息，请参阅此步骤。

错误：Unable to create server due to: failed to load group: chmod /<greengrass-root>/ggc/deployment/lambda/arn:aws:lambda:<region>:<account-id>:function:<function-name>:<version>/<file-name>: no such file or directory。

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会看到此错误。如果将 Lambda 可执行文件部署到核心中，请在 `group.json` 文件中检查函数的 `Handler` 属性（位于 */greengrass-root*/ggc/deployment/group 中）。如果处理程序不是编译的可执行文件的确切名称，请将 `group.json` 文件内容替换为空 JSON 对象 (`{}`)，并运行以下命令以启动 AWS IoT Greengrass：

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

然后，使用 AWS Lambda API 更新函数配置的 `handler` 参数，发布新的函数版本并更新别名。有关更多信息，请参阅 AWS Lambda 函数版本控制和别名。

假设按别名将函数添加到 Greengrass 组（建议），您现在可以重新部署组。（否则，您必须在组定义和订阅中指向新的函数版本或别名，然后再部署组。）

在您从无容器化的情况下运行更改为在 Greengrass 容器中运行后，AWS IoT Greengrass 核心软件无法启动。

解决方案：检查是否缺失任何容器依赖项。

错误：后台打印大小应至少为 262144 字节。

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会看到此错误。打开 `group.json` 文件（位于 */greengrass-root*`/ggc/deployment/group` 中），将文件内容替换为空 JSON 对象 (`{}`)，然后运行以下命令以启动 AWS IoT Greengrass：

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

然后，遵循过程 the section called “在本地存储中缓存消息” 中的步骤。对于 `GGCloudSpooler` 函数，请确保指定一个大于或等于 262144 的 `GG_CONFIG_MAX_SIZE_BYTES` 值。

错误：“[ERROR]-Cloud messaging error: Error occurred while trying to publish a message. {"errorString": "operation timed out"} ([ERROR]-云消息传递错误: 尝试发布消息时出错。{"errorString": "操作超时"})”

解决方案：当 Greengrass 核心无法向 AWS IoT Core 发送 MQTT 消息时，您可能会在 `GGCloudSpooler.log` 中看到此错误。如果核心环境存在有限的带宽和高延迟，则可能会发生此情况。如果你运行的是 AWS IoT Greengrass v1.10.2 或更高版本，请尝试将 config.json 文件中的 `mqttOperationTimeout` 值增大。如果此属性不存在，请将其添加到 `coreThing` 对象中。例如：

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

默认值为 5，最小值为 5。

错误：container_linux.go:344: starting container process caused "process_linux.go:424: container init caused \"rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: permission denied\\\"\""。

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会在 `runtime.log` 中看到此错误。如果您的 `umask` 高于 `0022`，则会出现此情况。要解决此问题，您必须将 `umask` 设置为 `0022` 或更低。默认情况下，值 `0022` 向每个人授予对新文件的读取权限。

错误：Greengrass 守护程序在 PID 为 <process-id> 的情况下运行。某些系统组件无法启动。检查“runtime.log”中有无错误。

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会看到此错误。检查 `runtime.log` 和 `crash.log` 中有无特定的错误信息。有关更多信息，请参阅 the section called “使用日志排查问题”。

设备影子未与云同步。

解决方案：确保 AWS IoT Greengrass 在 Greengrass 服务角色中具有 `iot:UpdateThingShadow` 和 `iot:GetThingShadow` 操作的权限。如果服务角色使用 `AWSGreengrassResourceAccessRolePolicy` 托管策略，则在默认情况下包含这些权限。

请参阅影子同步超时问题排查。

错误：无法接受 TCP 连接。接受 tcp [::]:8000: accept4：打开的文件太多。

解决方案： 您可能会在 `greengrassd` 脚本输出中看到此错误。如果 AWS IoT Greengrass 核心软件的文件描述符限制已达到阈值，必须调高，则可能会发生此错误。

使用以下命令，然后重新启动 AWS IoT Greengrass 核心软件。

```
ulimit -n 2048
```

> **Note**
>
> 在本示例中，该限制被调高到 2048。请选择符合您的使用案例的值。

错误：运行时执行错误：无法启动 lambda 容器。container_linux.go:259: starting container process caused "process_linux.go:345: container init caused \"rootfs_linux.go:50: preparing rootfs caused \\\"permission denied\\\"\"\"。

解决方案：直接在根目录下安装 AWS IoT Greengrass，或确保每个人对于安装了 AWS IoT Greengrass 核心软件的目录及其父目录都有 `execute` 权限。

警告：[警告]-[5] GK Remote：检索公钥数据时出错 ErrPrincipalNotConfigured:: 未设置的 MqttCertificate 私钥。

解决方案：AWS IoT Greengrass 使用常见的处理程序来验证所有安全主体的属性。除非您为本地 MQTT 服务器指定了自定义私有密钥，否则将会在 `runtime.log` 中收到此警告。有关更多信息，请参阅 the section called “安全委托人”。

错误：在试图使用角色 arn:aws:iam::<account-id>:role/<role-name> 访问 s3 url https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz 时权限被拒绝。

解决方案：当 over-the-air (OTA) 更新失败时，您可能会看到此错误。在签署人角色策略中，将目标 AWS 区域 添加为 `Resource`。此签署人角色用于为 AWS IoT Greengrass 软件更新的 S3 URL 进行预签名。有关更多信息，请参阅 S3 URL 签署人角色。

AWS IoT Greengrass 核心配置为使用网络代理，您的 Lambda 函数无法进行传出连接。

解决方案：根据您的运行时和 Lambda 函数用于创建连接的可执行文件，您也可能会收到连接超时错误。确保您的 Lambda 函数使用适当的代理配置通过网络代理进行连接。AWS IoT Greengrass 通过 `http_proxy`、`https_proxy` 和 `no_proxy` 环境变量将代理配置传递给用户定义的 Lambda 函数。可以按照以下 Python 代码段所示访问它们。

```
import os
print(os.environ['http_proxy'])
```

使用与环境中定义的变量相同的大小写，例如全部小写 `http_proxy` 或全部大写 `HTTP_PROXY`。对于这些变量，AWS IoT Greengrass 同时支持这两种写法。

> **Note**
>
> 用于建立连接的大多数公用库（例如 boto3 或 cURL 和 python `requests` 程序包）默认使用这些环境变量。

核心处于无限的连接-断开循环中。runtime.log 文件包含一系列连续的连接和断开条目。

解决方案：当另一个设备硬编码为使用核心事物名称作为与 AWS IoT 的 MQTT 连接的客户端 ID 时，可能会发生这种情况。同一 AWS 区域 和 AWS 账户 中的同时连接必须使用唯一的客户端 ID。默认情况下，核心使用核心事物名称作为这些连接的客户端 ID。

为解决此问题，您可以更改另一个设备用于连接的客户端 ID（推荐），或覆盖核心的默认值。

覆盖核心设备的默认客户端 ID

1. 运行以下命令以停止 Greengrass 进程守护程序：

   ```
   cd /greengrass-root/ggc/core/
   ```

```
sudo ./greengrassd stop
```

2. 打开 *greengrass-root*/config/config.json 以作为 su 用户进行编辑。
3. 在 coreThing 对象中，添加 coreClientId 属性，并将值设置为您的自定义客户端 ID。值的长度必须介于 1 到 128 个字符之间。它对于 AWS 账户 的当前 AWS 区域 必须是唯一的。

```
"coreClientId": "MyCustomClientId"
```

4. 启动进程守护程序。

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

错误：无法启动 lambda 容器。container_linux.go:259：启动容器进程导致出现 "process_linux.go:345：容器初始化导致出现 \"rootfs_linux.go:62：正在将 \\\"proc\\\" 装载到 \\\"

解决方案：在部分平台上，当 AWS IoT Greengrass 尝试装载 /proc 文件系统以创建 Lambda 容器时，您可能会在 runtime.log 中看到此错误。或者，您可能会看到类似的错误，例如 operation not permitted 或 EPERM。即使通过了由依赖关系检查器脚本在平台上运行的测试，仍可能出现这些错误。

请尝试下列可能的解决方案之一：

- 在 Linux 内核中启用 CONFIG_DEVPTS_MULTIPLE_INSTANCES 选项。
- 仅在主机上将 /proc 装载选项设置为 rw,relatim。
- 将 Linux 内核升级到 4.9 或更高版本。

> **Note**
>
> 此问题与装载 /proc 供本地资源访问无关。

[ERROR] 运行时执行错误：无法启动 lambda 容器。{"errorString": "无法初始化容器装载：无法对叠加层上层目录中的 greengrass 根进行掩码处理：无法在目录 <ggc-path> 中创建掩码设备：文件已存在"}

解决方案：在部署失败时，您可能会在 runtime.log 中看到此错误。如果 AWS IoT Greengrass 组中的 Lambda 函数无法访问核心文件系统中的 `/usr` 目录，则会出现此错误。

要解决此问题，请将本地卷资源添加到组，然后部署该组。该资源必须：

- 指定 `/usr` 作为 Source path (源路径) 和 Destination path (目标路径)。
- 自动添加拥有资源的 Linux 组的操作系统组权限。
- 与 Lambda 函数关联并允许只读访问。

[ERROR]-部署失败。 {"deploymentId": "<deployment-id>", "errorString": "PID 为 <pid> 的容器测试进程失败: 容器进程状态: 出口状态 1"}

解决方案：在部署失败时，您可能会在 runtime.log 中看到此错误。如果 AWS IoT Greengrass 组中的 Lambda 函数无法访问核心文件系统中的 `/usr` 目录，则会出现此错误。

可以通过检查 `GGCanary.log` 是否存在其他错误来确认情况确实如此。如果 Lambda 函数无法访问 `/usr` 目录，则 `GGCanary.log` 将包含以下错误：

```
[ERROR]-standard_init_linux.go:207: exec user process caused "no such file or
 directory"
```

要解决此问题，请将本地卷资源添加到组，然后部署该组。该资源必须：

- 指定 `/usr` 作为 Source path (源路径) 和 Destination path (目标路径)。
- 自动添加拥有资源的 Linux 组的操作系统组权限。
- 与 Lambda 函数关联并允许只读访问。

错误：[ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to create overlay fs for container: mounting overlay at /greengrass/ggc/packages/<ggc-version>/rootfs/merged failed: failed to mount with args source=\"no_source\" dest=\"/greengrass/ggc/packages/<ggc-version>/rootfs/merged\" fstype=\"overlay\" flags=\"0\" data=\"lowerdir=/greengrass/ggc/packages/<ggc-version>/dns:/,upperdir=/greengrass/ggc/packages/<ggc-version>/rootfs/upper,workdir=/greengrass/ggc/packages/<ggc-version>/rootfs/work\": too many levels of symbolic links"}

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会在 `runtime.log` 文件中看到此错误。此问题在 Debian 操作系统中可能更常见。

要解决此问题，请执行以下操作：

1. 将 AWS IoT Greengrass Core 软件升级到 v1.9.3 或更高版本。这应该可以自动解决此问题。
2. 如果在升级 AWS IoT Greengrass Core 软件后仍然出现此错误，请在 config.json 文件中将 `system.useOverlayWithTmpfs` 属性设置为 `true`。

   Example 示例

```
{
  "system": {
    "useOverlayWithTmpfs": true
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

> **Note**
>
> 错误消息中会显示您的 AWS IoT Greengrass Core 软件版本。要查找 Linux 的内核版本，请运行 `uname -r`。

错误：[DEBUG] – 未能获取路由。丢弃消息。

解决方案：检查您的组中的订阅并确保订阅列出在 `[DEBUG]` 消息中。

错误：[Errno 24] Too many open <lambda-function>,[Errno 24] Too many open files

解决方案：如果函数在函数处理程序中实例化 `StreamManagerClient`，您可能会在 Lambda 函数日志文件中看到此错误。我们建议您在处理程序之外创建客户端。有关更多信息，请参阅 the section called “使用 StreamManagerClient 处理流”。

错误: ds 服务器无法开始侦听套接字: listen unix <ggc-path>/ggc/socket/greengrass_ipc.sock: bind: 参数无效

解决方案：当 AWS IoT Greengrass 核心软件无法启动时，您可能会看到此错误。如果 AWS IoT Greengrass Core 软件安装到的文件夹有很长的文件路径，就会发生此错误。如果不使用写目录，则将 AWS IoT Greengrass Core 软件重新安装到文件路径小于 79 字节的文件夹中；如果使用写目录，则重新安装到文件路径小于 83 字节的文件夹中。

[信息]（复印机）aws.greengrass。 StreamManager: stdout。由：com.fasterxml.jackson.databind 引起。 JsonMappingException: 即时超过最小或最大瞬间

将 AWS IoT Greengrass Core 软件升级到 v1.11.3 时，如果流管理器无法启动，则您可能会在流管理器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}
```

如果您使用的 AWS IoT Greengrass Core 软件版本低于 v1.11.3，并且想要升级到更高版本，请使用 OTA 更新升级到 v1.11.4。

GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key

如果您一台设备上运行 `apt update`，并且在这台设备上，您是从 APT 存储库中安装了 AWS IoT Greengrass Core 软件，则可能会看到以下错误。

```
Err:4 https://dnw9lb6lzp2d8.cloudfront.net stable InRelease
  The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass
 Master Key
Reading package lists... Done
W: GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following
 signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key
```

之所以出现此错误，是因为 AWS IoT Greengrass 不再提供从 APT 存储库安装或更新 AWS IoT Greengrass Core 软件的选项。要成功运行 `apt update`，请从设备的来源列表中移除 AWS IoT Greengrass 存储库。

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

部署问题

使用以下信息可帮助您排查部署问题。

问题

- 您当前的部署不起作用，并且您希望恢复到以前有效的部署。
- 您在日志中看到有关部署的“403 禁止访问”错误。
- 首次运行 create-deployment 命令时会发生 ConcurrentDeployment 错误。
- 错误：未授权 Greengrass 担任与该账户相关的服务角色，或错误：失败：TES 服务角色未与此账户关联。
- 错误：无法在部署中执行下载步骤。下载时出现错误：下载组定义文件时出错：... x509: certificate has expired or is not yet valid
- 部署未完成。
- 错误：找不到 java 或 java8 可执行文件，或者出现错误：<deployment-id> NewDeployment 为组部署类型<group-id>失败错误：worker 无法<worker-id>初始化原因安装的 Java 版本必须大于或等于 8
- 部署未完成，并且 runtime.log 包含多个“等待 1 秒钟让容器停止”条目。
- 部署未完成，runtime.log 中包含“[ERROR]-Greengrass deployment error: failed to report deployment status back to cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}”
- <path>错误：<deployment-id> NewDeployment 为组部署类型<group-id>失败错误：处理时出错。组配置无效：112 或 [119 0] 对文件没有 rw 权限：。
- 错误：< list-of-function-arns > 配置为以根用户身份运行，但是 Greengrass 未配置为使用根权限运行 Lambda 函数。
- 错误：<deployment-id>组类型 NewDeployment 部署<group-id>失败错误：Greengrass 部署错误：无法在部署中执行下载步骤。处理时出错：无法加载下载的组文件：无法根据用户名找到 UID，用户名：ggc_user：用户：未知用户 ggc_user：未知用户 ggc_user。
- 错误：[ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to mask greengrass root in overlay upper dir: failed to create mask device at directory <ggc-path>: file exists"} ([ERROR] 运行时执行错误：无法启动 lambda 容器。{"errorString": "无法初始化容器装载：无法对叠加层上层目录中的 greengrass 根进行掩码处理：无法在目录 <ggc-path> 中创建掩码设备：文件已存在"})
- 错误：<deployment-id> NewDeployment 为组部署类型失败<group-id>错误：进程启动失败：container_linux.go: 259：启动容器进程导致 “process_linux.go: 250：为初始化运行 exec setns 进程导致\” 等待：没有子进程\ “”。

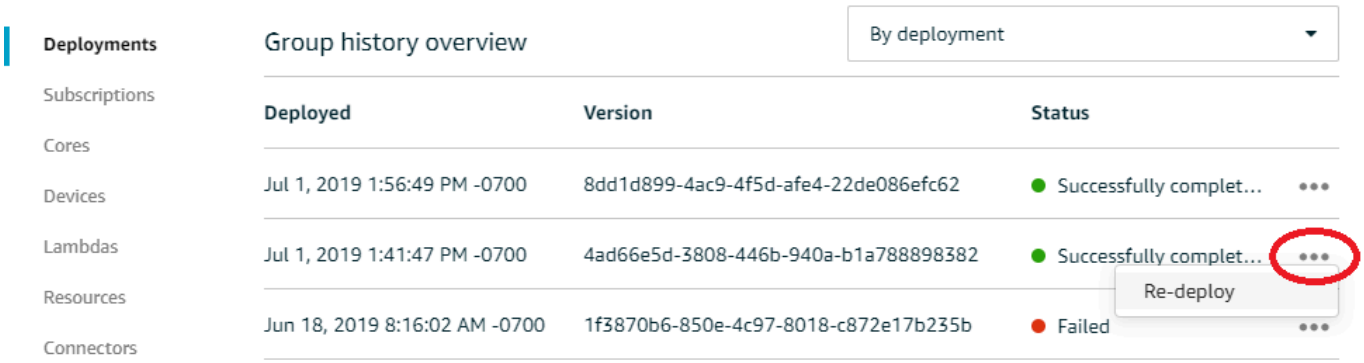
- 错误: [WARN]-MQTT[client] dial tcp: lookup <host-prefix>-ats.iot.<region>.amazonaws.com: 没有此类主机 ... [ERROR]-Greengrass 部署错误: 无法将部署状态报告回云... net/http: 等待连接时请求被取消 (等待标头时 Client.Timeout 超时)

您当前的部署不起作用，并且您希望恢复到以前有效的部署。

解决方案：使用 AWS IoT 控制台或 AWS IoT Greengrass API 重新部署先前有效的部署。这会将相应的组版本部署到您的核心设备。

重新部署一个部署（控制台）

1. 在组配置页面上，选择 部署选项卡。该页面显示组的部署历史记录，包括日期和时间、组版本以及每次部署尝试的状态。
2. 查找包含您要重新部署的部署的行。选择要重新部署的部署，然后选择重新部署。



重新部署一个部署 (CLI)

1. 用于查找ListDeployments要重新部署的部署的 ID。例如：

```
aws greengrass list-deployments --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7
```

该命令将返回组的部署列表。

```
{
    "Deployments": [
        {
            "DeploymentId": "8d179428-f617-4a77-8a0c-3d61fb8446a6",
            "DeploymentType": "NewDeployment",
```

```
            "GroupArn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/8dd1d899-4ac9-4f5d-
afe4-22de086efc62",
            "CreatedAt": "2019-07-01T20:56:49.641Z"
        },
        {
            "DeploymentId": "f8e4c455-8ac4-453a-8252-512dc3e9c596",
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/4ad66e5d-3808-446b-940a-
b1a788898382",
            "CreatedAt": "2019-07-01T20:41:47.048Z"
        },
        {
            "DeploymentId": "e4aca044-bbd8-41b4-b697-930ca7c40f3e",
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/1f3870b6-850e-4c97-8018-
c872e17b235b",
            "CreatedAt": "2019-06-18T15:16:02.965Z"
        }
    ]
}
```

> **Note**
>
> 这些 AWS CLI 命令将示例值用于组和部署 ID。当您运行这些命令时，确保替换示例值。

2. 用于CreateDeployment重新部署目标部署。将部署类型设置为 `Redeployment`。例如：

```
aws greengrass create-deployment --deployment-type Redeployment \
  --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7 \
  --deployment-id f8e4c455-8ac4-453a-8252-512dc3e9c596
```

该命令会返回新部署的 ARN 和 ID。

```
{
    "DeploymentId": "f9ed02b7-c28e-4df6-83b1-e9553ddd0fc2",
    "DeploymentArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/deployments/f9ed02b7-c28e-4df6-83b1-
e9553ddd0fc2"
```

```
}
```

3. GetDeploymentStatus用于获取部署状态。

您在日志中看到有关部署的“403 禁止访问”错误。

解决方案：确保 AWS IoT Greengrass 核心在云中的策略包含 `"greengrass:*"` 作为允许的操作。

首次运行 create-deployment 命令时会发生 ConcurrentDeployment 错误。

解决方案：可能正在进行部署。您可以运行 get-deployment-status 以查看部署是否已创建。如果没有，请再次尝试创建部署。

错误：未授权 Greengrass 担任与该账户相关的服务角色，或错误：失败：TES 服务角色未与此账户关联。

解决方案：当部署失败时，您可能会看到此错误。检查 Greengrass 服务角色是否与您在当前 AWS 区域 中的 AWS 账户 相关联。有关更多信息，请参阅the section called “管理服务角色 (CLI)”或the section called “管理服务角色（控制台）”。

错误：无法在部署中执行下载步骤。下载时出现错误：下载组定义文件时出错：... x509: certificate has expired or is not yet valid

解决方案：在部署失败时，您可能会在 `runtime.log` 中看到此错误。如果您收到包含 `x509: certificate has expired or is not yet valid` 消息的 `Deployment failed` 错误，请检查设备时钟。TLS 和 X.509 证书为构建 IoT 系统提供了安全的基础，但它们要求服务器和客户端上的时间准确无误。IoT 设备在尝试连接 AWS IoT Greengrass 或使用服务器证书的其他 TLS 服务之前应获得正确的时间（15 分钟内）。有关更多信息，请参阅 AWS 官方物联网博客上的使用设备时间验证 AWS IoT 服务器证书。

部署未完成。

解决方案：执行以下操作：

- 确保 AWS IoT Greengrass 守护程序正在您的核心设备上运行。在您的核心设备终端中运行以下命令来检查进程守护程序是否正在运行并启动它（如果需要）。

  1. 要检查进程守护程序是否正在运行，请执行以下操作：

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

     如果输出包含 `root` 的 `/greengrass/ggc/packages/1.11.6/bin/daemon` 条目，则表示进程守护程序正在运行。

     路径中的版本取决于您的核心设备上安装的 AWS IoT Greengrass Core 软件版本。

  2. 启动进程守护程序：

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

- 确保已连接核心设备并正确配置核心连接终端节点。

错误：找不到 java 或 java8 可执行文件，或者出现错误：<deployment-id> NewDeployment 为组部署类型<group-id>失败错误：worker 无法<worker-id>初始化原因安装的 Java 版本必须大于或等于 8

解决方案：如果为AWS IoT Greengrass 核心启用流管理器，必须在核心设备上安装 Java 8 运行时，然后再部署组。有关更多信息，请参阅流管理器的要求。使用 AWS IoT 控制台上的默认组创建工作流来创建组时，默认情况下会启用流管理器。

或者，禁用流管理器，然后部署该组。有关更多信息，请参阅 the section called “配置设置（控制台）”。

部署未完成，并且 runtime.log 包含多个“等待 1 秒钟让容器停止”条目。

解决方案：在您的核心设备终端中运行以下命令，以重新启动 AWS IoT Greengrass 守护程序。

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

部署未完成，**runtime.log** 中包含“[ERROR]-Greengrass deployment error: failed to report deployment status back to cloud {"deploymentId": "<deployment-id>", "errorString": "Failed to initiate PUT, endpoint: https://<deployment-status>, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificate signed by unknown authority"}”

解决方案：当 Greengrass 核心被配置为使用 HTTPS 代理连接且代理服务器证书链在系统上不受信任时，您可能会在 `runtime.log` 中看到此错误。若要尝试解决此问题，请将该证书链添加到根 CA 证书中。Greengrass 核心会将此文件中的证书添加到在与 AWS IoT Greengrass 建立 HTTPS 和 MQTT 连接时用于进行 TLS 身份验证的证书池。

以下示例显示了添加到根 CA 证书文件中的代理服务器 CA 证书：

```
# My proxy CA
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

# Amazon Root CA 1
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
```

```
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

默认情况下，根 CA 证书文件位于 */greengrass-root*/certs/root.ca.pem 中。要查找核心设备上的位置，请在 config.json 中查看 crypto.caPath 属性。

> **Note**
>
> *greengrass-root* 表示在您的设备上安装 AWS IoT Greengrass Core 软件的路径。通常，这是 /greengrass 目录。

<path>错误：<deployment-id> NewDeployment 为组部署类型<group-id>失败错误：处理时出错。组配置无效：112 或 [119 0] 对文件没有 rw 权限：。

解决方案：确保 *<path>* 目录的所有者组具有该目录的读取和写入权限。

错误：< list-of-function-arns > 配置为以根用户身份运行，但是 Greengrass 未配置为使用根权限运行 Lambda 函数。

解决方案：在部署失败时，您可能会在 runtime.log 中看到此错误。确保已将 AWS IoT Greengrass 配置为允许 Lambda 函数以根权限运行。将 greengrass_root/config/config.json 中 allowFunctionsToRunAsRoot 的值更改为 yes，或者将 Lambda 函数更改为以另一个用户/组的身份运行。有关更多信息，请参阅 the section called “以根用户身份运行 Lambda 函数”。

错误：<deployment-id>组类型 NewDeployment 部署<group-id>失败错误：Greengrass 部署错误：无法在部署中执行下载步骤。处理时出错：无法加载下载的组文件：无法根据用户名找到 UID，用户名：ggc_user：用户：未知用户 ggc_user：未知用户 ggc_user。

解决方案：如果 AWS IoT Greengrass 组的默认访问身份使用标准系统帐户，则 `ggc_user` 用户和 `ggc_group` 组必须出现在设备上。有关介绍如何添加用户和组的说明，请参阅此步骤。请务必完全按所述的方式输入名称。

错误：[ERROR]-runtime execution error: unable to start lambda container. {"errorString": "failed to initialize container mounts: failed to mask greengrass root in overlay upper dir: failed to create mask device at directory <ggc-path>: file exists"} ([ERROR] 运行时执行错误：无法启动 lambda 容器。{"errorString": "无法初始化容器装载：无法对叠加层上层目录中的 greengrass 根进行掩码处理：无法在目录 <ggc-path> 中创建掩码设备：文件已存在"})

解决方案：在部署失败时，您可能会在 `runtime.log` 中看到此错误。如果 Greengrass 组中的 Lambda 函数无法访问核心文件系统中的 `/usr` 目录，则会出现此错误。要解决此问题，请将本地卷资源添加到组，然后部署该组。资源必须：

- 指定 `/usr` 作为 Source path (源路径) 和 Destination path (目标路径)。
- 自动添加拥有资源的 Linux 组的操作系统组权限。
- 与 Lambda 函数关联并允许只读访问。

错误：<deployment-id> NewDeployment 为组部署类型失败<group-id>错误：进程启动失败：container_linux.go: 259：启动容器进程导致 “process_linux.go: 250：为初始化运行 exec setns 进程导致\” 等待：没有子进程\ “”。

解决方案：当部署失败时，您可能会看到此错误。重试部署。

错误: [WARN]-MQTT[client] dial tcp: lookup <host-prefix>-ats.iot.<region>.amazonaws.com: 没有此类主机 ... [ERROR]-Greengrass 部署错误: 无法将部署状态报告回云... net/http: 等待连接时请求被取消 (等待标头时 Client.Timeout 超时)

解决方案：如果您使用 `systemd-resolved`（默认情况下会启用 DNSSEC 设置），则可能会看到此错误。因此，无法识别许多公共域。尝试到达 AWS IoT Greengrass 终端节点时找不到主机，因此您的部署保持在 `In Progress` 状态。

您可以使用以下命令和输出以测试此问题。将终端节点中的 *##* 占位符替换为您的 AWS 区域。

```
$ ping greengrass-ats.iot.region.amazonaws.com
ping: greengrass-ats.iot.region.amazonaws.com: Name or service not known
```

```
$ systemd-resolve greengrass-ats.iot.region.amazonaws.com
greengrass-ats.iot.region.amazonaws.com: resolve call failed: DNSSEC validation failed:
 failed-auxiliary
```

一个可能的解决方案是禁用 DNSSEC。当 DNSSEC 为 `false` 时，DNS 查找未经过 DNSSEC 验证。有关更多信息，请参阅 `systemd` 的已知问题。

1. 将 `DNSSEC=false` 添加到 `/etc/systemd/resolved.conf`。
2. 重新启动 `systemd-resolved`。

有关 `resolved.conf` 和 DNSSEC 的信息，请在终端中运行 `man resolved.conf`。

有关创建组和创建函数的问题

使用以下信息来帮助排查与创建 AWS IoT Greengrass 组或 Greengrass Lambda 函数有关的问题。

问题

错误：您为该组设置IsolationMode的 “” 配置无效。

解决方案：当 `function-definition-version` 的 `DefaultConfig` 中的 `IsolationMode` 值不受支持时，会出现此错误。支持的值为 `GreengrassContainer` 和 `NoContainer`。

错误：你对带有 arn 的函数IsolationMode的 “” 配置<function-arn>无效。

解决方案：当 `function-definition-version` 的 <function-arn> 中的 `IsolationMode` 值不受支持时，会出现此错误。支持的值为 `GreengrassContainer` 和 `NoContainer`。

错误：<function-arn>不允许在 IsolationMode = NoContainer 中 MemorySize配置带有 arn 的函数。

解决方案：如果您指定一个 `MemorySize` 值并选择在不使用容器化的情况下运行，就会发生此错误。在没有容器化的情况下运行的 Lambda 函数不能有内存限制。您可以删除限制，也可以更改 Lambda 函数以使其在 AWS IoT Greengrass 容器中运行。

错误：<function-arn>不允许在 = 中访问带有 arn 的函数的 Sysfs 配置。IsolationMode NoContainer

解决方案：如果您为 `AccessSysfs` 指定 `true` 并选择在不使用容器化的情况下运行，就会发生此错误。在没有容器化的情况下运行的 Lambda 函数必须更新其代码才能直接访问文件系统，并且不能使用 `AccessSysfs`。您可以为 `AccessSysfs` 指定 `false` 值，也可以将 Lambda 函数更改为在 AWS IoT Greengrass 容器中运行。

错误：<function-arn>需要在 IsolationMode = GreengrassContainer 中 MemorySize配置带有 arn 的函数。

解决方案：发生该错误的原因是您没有为在 AWS IoT Greengrass 容器中运行的 Lambda 函数指定 `MemorySize` 限制。指定 `MemorySize` 值可解决该错误。

错误：函数<function-arn>指<resource-type>的是 IsolationMode = 中不允许使用的类型的资源NoContainer。

解决方案：当您在不进行容器化的情况下运行 Lambda 函数时，您不能访问 `Local.Device`、`Local.Volume`、`ML_Model.SageMaker.Job`、`ML_Model.S3_Object` 或 `S3_Object.Generic_Archive` 资源类型。如果需要这些资源类型，则必须在 AWS IoT Greengrass 容器中运行。您也可以更改 Lambda 函数中的代码，从而在不进行容器化的情况下直接访问本地设备。

错误：不允许使用具有 arn <function-arn> 的函数的 Execution 配置。

解决方案：当您创建一个具有 `GGIPDetector` 或 `GGCloudSpooler` 的系统 Lambda 函数并且您指定了 `IsolationMode` 或 `RunAs` 配置时，会出现此错误。您必须省略该系统 Lambda 函数的 `Execution` 参数。

发现问题

可以使用以下信息帮助解决 AWS IoT Greengrass 发现服务的问题。

问题

- 错误：Device is a member of too many groups, devices may not be in more than 10 groups

错误：Device is a member of too many groups, devices may not be in more than 10 groups

解决方案： 这是一个已知的限制。一个客户端设备最多可以是 10 个组的成员。

机器学习资源问题

使用以下信息帮助解决机器学习资源的问题。

问题

- InvalidML ModelOwner - GroupOwnerSetting 在 ML 模型资源中提供，但不存在 GroupOwner 或 GroupPermission 不存在
- NoContainer 在附加 Machine Learning 资源时，函数无法配置权限。 <function-arn>指在资源 <resource-id><ro/rw> 访问策略中具有权限的机器学习资源。
- 函数<function-arn>指的是 Machine <resource-id>Learning 资源，但两者都缺少权限 ResourceAccessPolicy 和资源 OwnerSetting。
- 函数<function-arn>指的是具有<resource-id>\ “rw\” 权限的 Machine Learning 资源，而资源所有者设置 GroupPermission仅允许\ “ro\”。
- NoContainer 函数<function-arn>是指嵌套目标路径的资源。
- Lambda <function-arn> 通过共享同一组所有者 ID 获得对资源 <resource-id> 的访问权限

InvalidML ModelOwner - GroupOwnerSetting 在 ML 模型资源中提供，但不存在 GroupOwner 或 GroupPermission 不存在

解决方案：如果机器学习资源包含ResourceDownloadOwnerSetting对象，但未定义必需的`GroupOwner`或`GroupPermission`属性，则会收到此错误。要解决此问题，请定义缺失的属性。

NoContainer 在附加 Machine Learning 资源时，函数无法配置权限。<function-arn>指在资源 <resource-id><ro/rw> 访问策略中具有权限的机器学习资源。

解决方案：如果非容器化 Lambda 函数指定了对机器学习资源的函数级权限，则会收到此错误。非容器化函数必须从在机器学习资源上定义的资源所有者权限继承权限。要解决此问题，请选择继承资源所有者权限（控制台）或从 Lambda 函数的资源访问策略 (API) 中删除权限。

函数<function-arn>指的是 Machine <resource-id>Learning 资源，但两者都缺少权限 ResourceAccessPolicy 和资源 OwnerSetting。

解决方案：如果未为附加的 Lambda 函数或资源配置对机器学习资源的权限，则会收到此错误。要解决此问题，请在 Lambda 函数的ResourceAccessPolicy属性或资源的OwnerSetting属性中配置权限。

函数<function-arn>指的是具有<resource-id>\ “rw\” 权限的 Machine Learning 资源，而资源所有者设置 GroupPermission仅允许\ “ro\”。

解决方案：如果为附加的 Lambda 函数定义的访问权限超过为机器学习资源定义的资源所有者权限，则会收到此错误。要解决此问题，请为 Lambda 函数设置限制更多的权限或为资源所有者设置限制较少的权限。

NoContainer 函数<function-arn>是指嵌套目标路径的资源。

解决方案：如果附加到非容器化 Lambda 函数的多个机器学习资源使用相同的目标路径或嵌套的目标路径，则会收到此错误。要解决此问题，请为资源指定单独的目标路径。

Lambda <function-arn> 通过共享同一组所有者 ID 获得对资源 <resource-id> 的访问权限

解决方案：如果将相同的操作系统组指定为 Lambda 函数的运行方式身份和机器学习资源的资源所有者，但该资源未附加到 Lambda 函数，您会 `runtime.log` 在中收到此错误。此配置为 Lambda 函数提供隐式权限，它可以使用这些权限来访问资源而无需 AWS IoT Greengrass 授权。

要解决此问题，请为其中一个属性使用不同的操作系统组，或将机器学习资源附加到 Lambda 函数。

Docker 中的 AWS IoT Greengrass 核心问题

使用以下信息可帮助解决与在 Docker 容器中运行 AWS IoT Greengrass 核心相关的问题。

问题

- 错误：未知选项：-no-include-email。
- 警告：IPv4 处于禁用状态。网络将不起作用。
- 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::: user/ <account-id><user-name>无权在资源上执行:ecr:: GetAuthorizationToken
- 错误：无法为服务 greengrass 创建容器：冲突。容器名称 “/aws-iot-greengrass” 已在使用中。
- 错误：[FATAL] - 由于意外错误，无法重置线程的 mount 命名空间：“操作不被允许”。为了保持一致性，GGC 将崩溃，需要手动重新启动。

错误：未知选项：-no-include-email。

解决方案：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`）。如果您使用的是 Windows，并且您已使用 MSI 安装程序安装 CLI，则必须重复安装过程。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。

警告：IPv4 处于禁用状态。网络将不起作用。

解决方案：当在 Linux 计算机上运行 AWS IoT Greengrass 时，您可能会收到此警告或类似的消息。按照此步骤中所述进行操作来启用 IPv4 网络转发。AWS IoT Greengrass 云部署和 MQTT 通信在未启用 IPv4 转发时将不运行。有关更多信息，请参阅 Docker 文档中的在运行时配置具有命名空间的内核参数 (sysctls)。

错误：防火墙阻止 Windows 和容器之间的文件共享。

解决方案：在 Windows 计算机上运行 Docker 时，您可能会收到此错误或 `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::: user/ <account-id><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：无法为服务 greengrass 创建容器：冲突。容器名称 “/aws-iot-greengrass” 已在使用中。

解决方案：当较旧的容器使用该容器名称时，可能会发生此错误。要解决此问题，请运行以下命令以删除旧的 Docker 容器：

```
docker rm -f $(docker ps -a -q -f "name=aws-iot-greengrass")
```

错误：[FATAL] - 由于意外错误，无法重置线程的 mount 命名空间：“操作不被允许”。为了保持一致性，GGC 将崩溃，需要手动重新启动。

解决方案：如果您尝试将一个 `GreengrassContainer` Lambda 函数部署到 Docker 容器中运行的 AWS IoT Greengrass 核心，则可能会在 `runtime.log` 中出现此错误。目前，只有 `NoContainer` Lambda 函数可以部署到 Greengrass Docker 容器。

要解决此问题，请确保所有 Lambda 函数处于 `NoContainer` 模式并启动新的部署。然后，当启动容器时，不要将现有 `deployment` 目录绑定挂载到 AWS IoT Greengrass 核心 Docker 容器。相反，在它的位置创建一个空 `deployment` 目录，并将该目录绑定挂载到 Docker 容器中。这样，新的 Docker 容器就可以接收具有在 `NoContainer` 模式下运行的 Lambda 函数的最新部署。

有关更多信息，请参阅 the section called “在 Docker 容器中运行 AWS IoT Greengrass”。

使用日志排查问题

您可以为 Greengrass 组配置日志设置，例如是将日志发送到日志，还是将日志 CloudWatch 存储在本地文件系统上，或者两者兼而有之。要在排查问题时获取详细信息，您可以暂时将日志记录级别更改为 `DEBUG`。对日志记录设置所做的更改在部署组时生效。有关更多信息，请参阅 the section called “为 AWS IoT Greengrass 配置日志记录”。

在本地文件系统上，AWS IoT Greengrass 将日志存储在以下位置。阅读文件系统上的日志需要根权限。

greengrass-root`/ggc/var/log/crash.log`

显示在 AWS IoT Greengrass 核心崩溃时生成的消息。

greengrass-root`/ggc/var/log/system/runtime.log`

显示有关哪些组件失败的消息。

greengrass-root`/ggc/var/log/system/`

包含来自 AWS IoT Greengrass 系统组件的所有日志，例如，证书管理器和连接管理器。通过利用 `ggc/var/log/system/` 和 `ggc/var/log/system/runtime.log` 中的消息，您应该能够找出 AWS IoT Greengrass 系统组件中出现了哪些错误。

greengrass-root`/ggc/var/log/system/localwatch/`

包含处理将 Greengrass 日志上传到日志的AWS IoT Greengrass组件的日志。 CloudWatch 如果您无法查看 Greengrass 登录信息，则可以使用这些日志 CloudWatch进行故障排除。

greengrass-root/ggc/var/log/user/

包含来自用户定义的 Lambda 函数的所有日志。检查此文件夹以查找来自您的本地 Lambda 函数的错误消息。

> **Note**
>
> 默认情况下，*greengrass-root* 是 /greengrass 目录。如果配置了写入目录，则日志位于该目录中。

如果将日志配置为存储在云上，请使用 CloudWatch 日志查看日志消息。 crash.log只能在AWS IoT Greengrass核心设备上的文件系统日志中找到。

如果配置AWS IoT为写入日志 CloudWatch，则如果系统组件尝试连接时出现连接错误，请检查这些日志AWS IoT。

有关 AWS IoT Greengrass 日志记录的更多信息，请参阅 the section called “利用 AWS IoT Greengrass 日志进行监控”。

> **Note**
>
> AWS IoT Greengrass 核心软件 1.0 版本的日志存储在 *greengrass-root*/var/log 目录下。

排查存储问题

当本地文件存储空间已满时，某些组件可能会开始失败：

- 本地影子更新未进行。
- 新的 AWS IoT Greengrass 核心 MQTT 服务器证书无法在本地下载。
- 部署失败。

您应该始终留意本地的可用空间大小。您可以基于已部署的 Lambda 函数的大小、日志记录配置（请参阅 the section called “使用日志排查问题”）和在本地存储的影子的数量计算可用空间。

对消息进行问题排查

在 AWS IoT Greengrass 中本地发送的所有消息都使用 QoS 0 发送。默认情况下，AWS IoT Greengrass 在内存中队列中存储消息。因此，在 Greengrass 核心重新启动时，未处理的消息将会丢失；例如，在组部署或设备重启后。不过，您可以配置 AWS IoT Greengrass (v1.6 或更高版本) 以将消息缓存到文件系统中，以便在核心重新启动时永久保存这些消息。您也可以配置队列大小。如果配置队列大小，请确保它大于或等于 262144 字节 (256 KB)。否则，AWS IoT Greengrass 可能无法正确启动。有关更多信息，请参阅 the section called “MQTT 消息队列”。

> **Note**
>
> 在使用默认内存中队列时，我们建议您部署组或在服务中断最少的情况下重新启动设备。

您还可以配置核心以建立与 AWS IoT 的持久会话。这将允许核心接收在其脱机时从 AWS Cloud 发送的消息。有关更多信息，请参阅 the section called “与 AWS IoT Core 的 MQTT 持久性会话”。

影子同步超时问题排查

如果 Greengrass 核心设备和云之间的通信明显延迟，则影子同步可能因超时而失败。在这种情况下，您应会看到如下所示的日志条目：

```
[2017-07-20T10:01:58.006Z][ERROR]-cloud_shadow_client.go:57,Cloud shadow
 client error: unable to get cloud shadow what_the_thing_is_named for
 synchronization. Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
 while awaiting headers)
[2017-07-20T10:01:58.006Z][WARN]-sync_manager.go:263,Failed to get cloud
 copy: Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
 while awaiting headers)
[2017-07-20T10:01:58.006Z][ERROR]-sync_manager.go:375,Failed to execute sync operation
 {what_the_thing_is_named VersionDiscontinued []}"
```

一种可能的解决方法是配置核心设备等待主机响应的时长。在 *greengrass-root*/config 中打开 config.json 文件并为 system.shadowSyncTimeout 字段添加一个超时值（以秒为单位）。例如：

```
{
  "system": {
```

```
    "shadowSyncTimeout": 10
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

如果未在 `config.json` 中指定 `shadowSyncTimeout` 值，则默认值为 5 秒。

> **Note**
>
> 对于 AWS IoT Greengrass 核心软件 1.6 及更早版本，默认 `shadowSyncTimeout` 为 1 秒。

查看 AWS re:Post

如果您无法使用本主题中的问题排查信息解决问题，可以在 AWS re:Post 上搜索 排查问题 或查看 AWS IoT Greengrass 标签搜索相关问题或发布新的问题。AWS IoT Greengrass 团队成员主动监控 AWS re:Post。

的文档历史记录 AWS IoT Greengrass

下表介绍了 2018 年 6 月之后《 AWS IoT Greengrass 开发者指南》的重要更改。要获得本文档的更新通知，您可以订阅 RSS 源。

变更	说明	日期
v1.11.x Snap 的支持终止更新	在 snapcraft.io 上更新了 AWS IoT Greengrass core v 1.11.x Snap 的终止支持信息。	2023 年 9 月 22 日
v1.11.x Snap 的支持终止	在 snapcraft.io 上添加了 AWS IoT Greengrass core v 1.11.x Snap 的终止支持信息。	2023 年 9 月 19 日
适用 AWS IoT Greengrass 于 v1.11.6 的 Docker 镜像	AWS IoT Greengrass 核心软件 v1.11.6 的 Docker 镜像可在亚马逊弹性容器注册表 (Amazon ECR) Container Registry 和 Docker Hub 上线。建议您始终运行最新版本。	2022 年 4 月 12 日
AWS IoT 设备测试器 (IDT) 是否已弃用 AWS IoT Greengrass V1	适用于 AWS IoT Greengrass V1 的 IDT 将不再生成已签署的资格报告。	2022 年 4 月 22 日
AWS IoT 设备测试器的 Support 更新适用于 AWS IoT Greengrass	AWS IoT Greengrass 版本 4.4.1 的 IDT 现在支持使用 AWS IoT Greengrass 核心软件版本 v1.11.6 进行设备认证。	2022 年 3 月 24 日
AWS IoT Greengrass 1.11.6 版本已发布	AWS IoT Greengrass 核心软件版本 1.11.6 现已推出。此版本包含性能改进和错误修复。建议您始终运行最新版本。	2022 年 3 月 24 日

物联网 SiteWise 连接器版本 12 已发布	物联网 SiteWise 连接器的版本 12 现已推出。此版本包含错误修复。	2021 年 12 月 23 日
适用于 v1.11.5 和 AWS IoT Greengrass v1.10.5 的 Docker 镜像	AWS IoT Greengrass 核心软件 v1.11.5 和 v1.10.5 的 Docker 镜像可在亚马逊 Elastic Container Registry（亚马逊 ECR）和 Docker Hub 上线。建议您始终运行最新版本。	2021 年 12 月 22 日
AWS IoT Greengrass V1 维护政策	AWS IoT Greengrass V1 维护策略定义了 AWS IoT Greengrass V1 服务和 AWS IoT Greengrass 核心软件 v1.x 的不同维护和更新级别。	2021 年 12 月 20 日
AWS IoT 设备测试器版本 4.4.1 已发布	4.4.1 AWS IoT Greengrass 版本的 IDT 现已推出。此版本包括 AWS IoT Greengrass 资格认证套件 (GGQ) v1.3.1，并支持使用 AWS IoT Greengrass 核心软件版本 v1.11.5 和 v1.10.5 进行设备认证。	2021 年 12 月 20 日
AWS IoT Greengrass 1.11.5 和 1.10.5 版本已发布	AWS IoT Greengrass 核心软件的版本为 1.11.5 和 1.10.5。这些版本包含性能改进和错误修复。建议您始终运行最新版本。	2021 年 12 月 12 日

重新发布了 AWS IoT Greengrass v1.11.4 和 v1.10.4 的 Docker 镜像	AWS IoT Greengrass 核心软件版本 1.11.4 和 1.10.4 的 Docker 镜像已在亚马逊 Elastic Container Registry (Amazon ECR) 和 Docker Hub 上重新发布，以解决错误修复。BusyBox要使用最新的 Docker 映像，请使用 `1.11.4-1` 或 `1.10.4-1` 标签。有关可用标签的更多信息，请参阅 Docker Hub aws-iot-greengrass 中的 ama zon/。	2021 年 12 月 8 日
CloudWatch 指标连接器支持在输入数据中使用重复的时间戳	现在，您可以将带有重复时间戳的输入数据发送到此连接器。	2021 年 11 月 19 日
防止跨服务混淆代理更新	AWS IoT Greengras s 支持在 IAM 资源策略中使用`aws:Sourc eArn` 和`aws:Sourc eAccount` 全局条件上下文密钥来防止混淆副手问题。	2021 年 11 月 1 日
适用 AWS IoT Greengrass 于 v1.11.4 的 Docker 镜像	AWS IoT Greengrass 核心软件 v1.11.4 的 Docker 镜像可在亚马逊弹性容器注册表 (Amazon ECR) Container Registry 和 Docker Hub 上线。建议您始终运行最新版本。	2021 年 8 月 24 日
已发布 AWS IoT Greengrass v1.11.4 快照	AWS IoT Greengrass 快照的 1.11.4 版本可用。建议您始终运行最新版本。	2021 年 8 月 20 日

AWS IoT 设备测试器的 Support 更新适用于 AWS IoT Greengrass	AWS IoT Greengrass 版本 4.1.0 的 IDT 现在支持使用 AWS IoT Greengrass 核心软件版本 v1.11.4 进行设备认证。	2021 年 8 月 18 日
AWS IoT Greengrass 1.11.4 版本已发布	AWS IoT Greengrass 核心软件版本 1.11.4 已推出。此版本修复了直播管理器的一个问题，该问题导致无法从早期版本的核心软件升级到 v1.11.3。AWS IoT Greengrass 建议您始终运行最新版本。	2021 年 8 月 17 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 控制面板之间建立私有连接。	2021 年 8 月 16 日
AWS IoT 设备测试器 4.1.0 版本已发布	AWS IoT 设备测试器的 4.1.0 版 AWS IoT Greengrass 现已推出。此版本支持使用 AWS IoT Greengrass 核心软件版本 1.11.3 和 1.10.4 进行设备认证。	2021 年 6 月 23 日
已发布 AWS IoT Greengrass v1.11.3 快照	AWS IoT Greengrass 快照的 1.11.3 版本包含性能改进和错误修复。作为最佳实践，我们建议您始终运行最新版本。	2021 年 6 月 15 日

AWS IoT Greengrass 1.11.3 和 v1.10.4 的 Docker 镜像已发布	AWS IoT Greengrass 核心软件 v1.11.3 和 v1.10.4 的 Docker 镜像可在亚马逊 Elastic Container Registry (Amazon ECR) 和 Docker Hub 上线。这些版本的 AWS IoT Greengrass Core 包含性能改进和错误修复。建议您始终运行最新版本。	2021 年 6 月 15 日
AWS IoT Greengrass 版本 1.11.3 已发布	AWS IoT Greengrass 核心软件版本 1.11.3 现已推出。此版本包含性能改进和错误修复。建议您始终运行最新版本。	2021 年 6 月 14 日
AWS IoT Greengrass 1.10.4 版本已发布	AWS IoT Greengrass 核心软件版本 1.10.4 现已推出。此版本包含性能改进和错误修复。建议您始终运行最新版本。	2021 年 6 月 14 日
Modbus-TCP 协议适配器版本 2 发布	Modbus-TCP 协议适配器连接器版本 2 现已推出。此版本增加了对 ASCII、UTF8 和 ISO8859 编码源字符串的支持。	2021 年 5 月 24 日
Docker 应用程序部署连接器版本 7 已发布	Greengrass Docker 应用程序部署连接器的版本 7 已发布。	2021 年 4 月 5 日
AWS IoT Greengrass 1.11.1 版本已发布	AWS IoT Greengrass 核心软件版本 1.11.1 现已推出。此版本包含性能改进和错误修复。建议您始终运行最新版本。	2021 年 3 月 29 日

AWS IoT 设备测试器版本 4.0.2 已发布	4.0.2 版的 AWS IoT 设备测试器 AWS IoT Greengrass 现已推出。此版本取代了 IDT v4.0.0，并增加了对核心软件版本 1.11.1 的支持。 AWS IoT Greengrass 此版本还修复了导致 IDT 掩盖硬件安全集成 (HSI) 错误的问题。	2021 年 3 月 29 日
物联网 SiteWise 连接器版本 11 已发布	物联网 SiteWise 连接器版本 11 现已推出。这将开始支持含有隐藏字符或不可打印字符的字符串。此版本还包括一般性能改进和错误修复。	2021 年 3 月 24 日
重新发布了 AWS IoT Greengrass v1.11.0 快照	AWS IoT Greengrass snap 版本 1.11.0 已在 Snapcraft 上重新发布，以解决错误修复以及使用 Python 解释器时可能出现的应用程序崩溃问题。 AWS IoT Greengrass 不提供软件版本 1.10 和 1.9 的快照。	2021 年 3 月 19 日
AWS IoT 设备测试器的 Support 更新适用于 AWS IoT Greengrass	AWS IoT Greengrass 版本 4.0.0 的 IDT 现在支持使用 AWS IoT Greengrass 核心软件版本 v1.10.3 进行设备认证。	2021 年 3 月 18 日
已重新发布 AWS IoT Greengrass v1.8.4 快照	AWS IoT Greengrass snap 版本 1.8.4 已在 Snapcraft 上重新发布，以解决错误修复以及使用 Python 解释器时可能出现的应用程序崩溃问题。	2021 年 3 月 15 日

已重新发布适用于 arm AWS IoT Greengrass v7L 的 v1.11.0 Docker 镜像	适用于Armv7L平台的 AWS IoT Greengrass 核心软件版本 1.11.0的Docker镜像已在亚马逊弹性容器注册表（Amazon ECR）和Docker Hub上重新发布，以解决使用Python解释器时错误修复和可能的应用程序崩溃问题。	2021 年 3 月 8 日
AWS IoT Greengrass v1.10.3 Docker 镜像已发布	AWS IoT Greengrass 核心软件版本 1.10.3 的 Docker 镜像现已在亚马逊 Elastic Container Registry (Amazon ECR) 和 Docker Hub 上线。	2021 年 3 月 8 日
重新发布了 AWS IoT Greengrass v1.11.0 和 v1.9.4 的 Docker 镜像	AWS IoT Greengrass 核心软件版本 1.11.0 和 1.9.4 的 Docker 镜像已在亚马逊 Elastic Container Registry (Amazon ECR) 和 Docker Hub 上重新发布，以解决错误修复和使用 Python 解释器时可能出现的应用程序崩溃问题。适用于 Armv7L 的 Docker 映像目前尚未重新发布。	2021 年 2 月 26 日
AWS IoT Greengrass 1.10.3 版本已发布	AWS IoT Greengrass 核心软件版本 1.10.3 已推出。此版本添加了 `systemComponentAuthTimeout` 核心配置属性，并包含性能改进和错误修复。建议您始终运行最新版本。	2021 年 2 月 24 日

物联网 SiteWise 连接器版本 10 已发布	物联网 SiteWise 连接器的版本 10 现已推出。此版本解决了连接中断时 StreamManager 客户端的稳定性问题，并改进了缺少连接时的 OPC-UA 值处理。`SourceTimestamp` 使用物联网 SiteWise 连接器将本地设备和设备数据发送到物联网中的资产属性 SiteWise。	2021 年 1 月 22 日
物联网 SiteWise 连接器版本 9 已发布	物联网 SiteWise 连接器版本 9 现已推出。这将开始支持自定义 Greengrass `StreamManager` 流目标、OPC-UA 死区、自定义扫描模式和自定义扫描速率。这还包括在通过物联 SiteWise 网网关进行配置更新期间的性能得到改善。使用物联网 SiteWise 连接器将本地设备和设备数据发送到物联网中的资产属性 SiteWise。	2020 年 12 月 15 日
AWS IoT 设备测试器 4.0.0 版已发布	适用于 4.0.0 的 AWS IoT 设备测试器版本现 AWS IoT Greengrass 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 15 日
AWS IoT Greengrass snap v1.11	AWS IoT Greengrass 快照版本 1.11.0 支持非容器化的 Lambda 函数。作为最佳实践，我们建议您始终运行最新版本。	2020 年 12 月 6 日

物联网 SiteWise 连接器版本 8 已发布	物联网 SiteWise 连接器的版本 8 现已推出。此版本提高了连接器遇到间歇性网络连接时的稳定性。使用物联网 SiteWise 连接器将本地设备和设备数据发送到物联网中的资产属性 SiteWise。	2020 年 11 月 19 日
Kinesis Firehose 连接器支持无容器模式	您可以使用 `IsolationMode` 参数配置连接器容器化模式。	2020 年 10 月 19 日
Docker 应用程序部署连接器版本 6 已发布	Greengrass Docker 应用程序部署连接器的版本 6 已发布。	2020 年 9 月 18 日
AWS IoT Greengrass 1.11.0 版本已发布	AWS IoT Greengrass 核心软件版本 1.11.0 已推出。此版本添加了系统运行状况遥测功能和本地运行状况检查 API。流管理器现在可以将数据导出到亚马逊简单存储服务 (Amazon S3) Service 和物联网。 SiteWise此版本也包含性能改进和错误修复。建议您始终运行最新版本。	2020 年 9 月 16 日
物联网 SiteWise 连接器版本 7 已发布	物联网 SiteWise 连接器版本 7现已推出。此版本修复了网关指标的问题。使用物联网 SiteWise 连接器将本地设备和设备数据发送到物联网中的资产属性 SiteWise。	2020 年 8 月 14 日
ServiceNow MetricBase 集成、Splunk 集成和 Twilio 通知连接器支持无容器模式	您可以使用 `IsolationMode` 参数配置连接器容器化模式。	2020 年 7 月 30 日
SNS 连接器支持无容器模式	您可以使用 `IsolationMode` 参数配置连接器容器化模式。	2020 年 7 月 6 日

CloudWatch 指标连接器支持“无容器” 模式	您可以使用 `IsolationMode` 参数配置连接器容器化模式。	2020 年 6 月 17 日
AWS IoT Greengrass 1.10.2 版本已发布	AWS IoT Greengrass 核心软件版本 1.10.2 现已推出。此版本添加了 `mqttOperationTimeout` 核心配置属性，并包含性能改进和错误修复。建议您始终运行最新版本。	2020 年 6 月 8 日
Tensorflow 机器学习安装程序已弃用	AWS IoT Greengrass Tensorflow 预打包的机器学习安装程序已被弃用。机器学习示例已升级到 Python 3.7。	2020 年 5 月 29 日
Chainer 框架支持和 Greengrass 机器学习安装程序已被弃用	AWS IoT Greengrass MXNet 和 DLR 的预打包机器学习安装程序和下载已被弃用。Chainer 框架支持和相关下载项目已被弃用。	2020 年 5 月 4 日
物联网 SiteWise 连接器版本 6 已发布	物联网 SiteWise 连接器的版本 6 已推出。此版本增加了对 CloudWatch 指标和自动发现新 OPC-UA 标签的支持。这意味着，当 OPC-UA 源的标签发生更改时，您无需重新启动网关。此版本的连接器需要流管理器和 AWS IoT Greengrass 核心软件 v1.10.0 或更高版本。使用物联网 SiteWise 连接器将本地设备和设备数据发送到物联网中的资产属性 SiteWise。	2020 年 4 月 29 日

连接器已升级到 Python 3.7	支持 Python 运行时的连接器已升级到 Python 3.7。我们建议您将连接器版本从 Python 2.7 升级到 Python 3.7。	2020 年 4 月 29 日
Greengrass 设备安装程序可在无提示模式下运行	您可以在无提示模式下运行 Greengrass 设备安装程序，以便脚本不会提示您输入任何值。	2020 年 4 月 27 日
新的 Docker 基本映像	你可以下载基于 Alpine Linux（x86_64、armv7L 或 aarch64）基础镜像构建的 AWS IoT Greengrass Docker 镜像。	2020 年 4 月 23 日
AWS IoT Greengrass 1.10.1 版本已发布	AWS IoT Greengrass 核心软件版本 1.10.1 现已推出。此版本包含性能改进和错误修复。建议您始终运行最新版本。	2020 年 4 月 16 日
新增安全性章节	AWS IoT Greengrass 安全内容已重新组织，添加了新信息。	2020 年 3 月 30 日
使用 APT 软件包管理器进行安装 AWS IoT Greengrass	在支持的基于 Debian 的 Linux 发行版上，您可以使用`apt`在设备上安装 AWS IoT Greengrass 核心软件。	2020 年 2 月 26 日
物联网 SiteWise 连接器版本 5 已发布	物联网 SiteWise 连接器的版本 5 现已推出。此版本修复了与 AWS IoT Greengrass 核心软件 v1.9.4 的兼容性问题。使用物联网 SiteWise 连接器将本地设备和设备数据发送到物联网中的资产属性 SiteWise。	2020 年 2 月 12 日

用于快速设置核心设备的新脚本	您可以使用 Greengrass 设备设置在数分钟内配置您的核心设备。此外，AWS IoT Greengrass 现在还支持 Node.js 12.x Lambda 函数。	2019 年 12 月 20 日
AWS IoT Greengrass 1.10.0 版本已发布	AWS IoT Greengrass 核心软件版本 1.10.0 已推出。此版本的新增功能包括流管理器、容器支持 Docker 应用程序部署连接器、非容器化 Lambda 函数可以访问机器学习资源、支持与 AWS IoT 之间的 MQTT 持久会话，以及支持本地 MQTT 流量通过指定端口传输。	2019 年 11 月 25 日
对部署通知的控制台支持	使用亚马逊 EventBridge 控制台创建在您的 Greengrass 群组部署状态发生变化时触发的事件规则。	2019 年 11 月 14 日
AWS IoT Greengrass 1.9.4 版本已发布	AWS IoT Greengrass 核心软件版本 1.9.4 现已推出。此版本包含性能改进和错误修复。作为最佳实践，我们建议您始终运行最新版本。	2019 年 10 月 17 日
对于管理 Greengrass 服务角色的控制台支持	使用 AWS IoT 控制台中的新增和改进功能来管理您的 Greengrass 服务角色。	2019 年 10 月 4 日
对于管理组级标签的控制台支持	您可以在 AWS IoT 控制台中为 Greengrass 组创建、查看和管理标签。	2019 年 9 月 23 日

全新机器学习连接器	使用 ML 反馈连接器发布模型输入和预测，并使用 ML 对象检测连接器运行本地对象检测推理服务。	2019 年 9 月 19 日
AWS IoT Greengrass 1.9.3 版本已发布	AWS IoT Greengrass 核心软件版本 1.9.3 已推出。此版本允许你在 armv6L 架构上的 Raspbian 发行版上安装 AWS IoT Greengrass 核心软件，支持使用 ALPN 在 443 端口 OTA 更新，并包含对从 Python 2.7 Lambda 函数发送到其他 Lambda 函数的二进制有效负载的错误修复。	2019 年 9 月 12 日
AWS IoT Greengrass 版本 1.8.4 已发布	AWS IoT Greengrass 核心软件版本 1.8.4 现已推出。此版本包含性能改进和错误修复。如果您正在运行 1.8.x 版本，我们建议您升级到 1.8.4 版本或 1.9.4 版本。对于更早的版本，我们建议您升级到 1.9.3 版本。	2019 年 8 月 30 日
AWS IoT Greengrass 1.9.2 版本已发布，支持以下内容 OpenWrt	AWS IoT Greengrass 核心软件版本 1.9.2 已推出。此版本允许您在采用 Armv8 (aarch64) 和 armv7L 架构的 OpenWrt 发行版上安装 AWS IoT Greengrass 核心软件。	2019 年 6 月 20 日

AWS IoT Greengrass 版本 1.8.3 已发布	AWS IoT Greengrass 核心软件版本 1.8.3 已推出。此版本包含常规性能改进和错误修复。如果您正在运行 1.8.x 版本，我们建议您升级到 1.8.3 版本或 1.9.2 版本。对于更早的版本，我们建议您升级到 1.9.2 版本。	2019 年 6 月 20 日
AWS IoT Greengrass 版本 1.9.1 已发布	AWS IoT Greengrass 核心软件版本 1.9.1 现已推出。此版本修复了来自主题中 AWS IoT 包含通配符的消息的错误。	2019 年 5 月 10 日
AWS IoT Greengrass 版本 1.8.2 已发布	AWS IoT Greengrass 核心软件版本 1.8.2 现已推出。此版本包含常规性能改进和错误修复。如果您正在运行 1.8.x 版本，我们建议您升级到 1.8.2 版本或 1.9.0 版本。对于更早期的版本，我们建议您升级到 1.9.0 版本。	2019 年 5 月 2 日
AWS IoT Greengrass 1.9.0 版本已发布	新功能：支持 Python 3.7 和 Node.js 8.10 Lambda 运行时，优化了 MQTT 连接，并且 Elliptic Curve (EC) 密钥支持本地 MQTT 服务器。	2019 年 5 月 1 日
AWS IoT Greengrass 版本 1.8.1 已发布	AWS IoT Greengrass 核心软件版本 1.8.1 现已推出。此版本包含常规性能改进和错误修复。作为最佳实践，我们建议您始终运行最新版本。	2019 年 4 月 18 日

AWS IoT Greengrass 快照可在 snapcraft	使用 AWS IoT Greengrass Snap Store 应用程序在 Linux 设备上快速设计、测试和部署软件 AWS IoT Greengrass。	2019 年 4 月 1 日
支持使用基于标签的权限进行更多访问控制	您可以在 AWS Identity and Access Management (IAM) 策略中使用标签来控制对 AWS IoT Greengrass 资源的访问权限。	2019 年 3 月 29 日
IoT Analytics 连接器已发布	使用 IoT Analytics 连接器将本地设备数据发送到 AWS IoT Analytics 通道。	2019 年 3 月 15 日
Kinesis Firehose 连接器中支持批处理	Kinesis Firehose 连接器支持按指定的间隔向亚马逊数据 Firehose 发送批量数据记录。	2019 年 3 月 15 日
AWS CloudFormation 对 AWS IoT Greengrass 资源的支持	使用 AWS CloudFormation 模板来创建和管理 AWS IoT Greengrass 资源。	2019 年 3 月 15 日
AWS IoT Greengrass 1.8.0 版本已发布	新功能：Lambda 函数可配置的默认访问身份，支持端口 443 上的 HTTPS 流量，以及与的 MQTT 连接的可预测命名客户端 ID。 AWS IoT	2019 年 3 月 7 日
AWS IoT Greengrass 1.7.1 和 1.6.1 版本已发布	AWS IoT Greengrass 核心软件版本为 1.7.1 和 1.6.1。这些版本需要 Linux 内核版本 3.17 或更高版本。我们建议运行 Greengrass Core 软件的任何版本的客户立即升级到版本 1.7.1。	2019 年 2 月 11 日

SageMaker Neo 深度学习运行时	SageMaker Neo 深度学习运行时支持由 N SageMaker eo 深度学习编译器优化的机器学习模型。	2018 年 11 月 28 日
AWS IoT Greengrass 在 Docker 容器中运行	你可以将 Greengrass 组配置为 AWS IoT Greengrass 在不进行容器化的情况下运行，从而在 Docker 容器中运行。	2018 年 11 月 26 日
AWS IoT Greengrass 1.7.0 版本已发布	新功能：Greengrass 连接器、本地 Secrets Manager、Lambda 函数的隔离和权限设置、可信硬件根安全、使用 ALPN 或网络代理进行连接以及 Raspbian Stretch 支持。	2018 年 11 月 26 日
AWS IoT Greengrass 软件下载	AWS IoT Greengrass 核心软件、Cor AWS IoT Greengrass e SDK 和 Mach AWS IoT Greengrass ine Learning SDK 包可通过亚马逊 CloudFront下载。	2018 年 11 月 26 日
AWS IoT 的设备测试器 AWS IoT Greengrass	使用 AWS IoT 设备测试器 AWS IoT Greengrass 来验证您的 CPU 架构、内核配置和驱动程序是否可以正常工作 AWS IoT Greengrass。	2018 年 11 月 26 日
AWS CloudTrail 记录 AWS IoT Greengrass API 调用	AWS IoT Greengrass 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在中执行的操作的记录 AWS IoT Greengrass。	2018 年 10 月 29 日

TensorFlow 在 NVIDIA Jetson TX2 上支持 v1.10.1	AWS IoT Greengrass 提供的 NVIDIA Jetson TX2 TensorFlow 预编译库现在使用 v1.10.1。TensorFlow 这支持 Jetpack 3.3 和 CUDA Toolkit 9.0。	2018 年 10 月 18 日
支持 MXNet v1.2.1 机器学习资源	AWS IoT Greengrass 支持使用 MxNet v1.2.1 训练的机器学习模型。	2018 年 8 月 29 日
AWS IoT Greengrass 1.6.0 版本已发布	新功能：Lambda 可执行文件、可配置的消息队列、可配置的重新连接重试间隔、/proc 中的卷资源以及可配置的写入目录。	2018 年 7 月 26 日

早期更新

下表介绍了 2018 年 7 月之前对《 AWS IoT Greengrass 开发者指南》所做的重要更改。

更改	描述	日期
AWS IoT Greengrass 版本 1.5.0 已发布	新功能： • 使用云训练模型的本地机器学习推理。有关更多信息，请参阅执行机器学习推理。 • 除了 JSON 以外，Greengrass Lambda 函数还支持二进制输入数据。 有关更多信息，请参阅 AWS IoT Greengrass 核心版本。	2018 年 3 月 29 日
AWS IoT Greengrass 版本 1.3.0 已发布	新功能： • O ver-the-air (OTA) 更新代理能够处理云端部署的 Greengrass 更新任务。有关更多信息，请参阅AWS IoT Greengrass Core 软件的 OTA 更新。	2017 年 11 月 27 日

更改	描述	日期
	• 从 Greengrass Lambda 函数访问本地外围设备和资源。有关更多信息，请参阅使用 Lambda 函数和连接器访问本地资源。	
AWS IoT Greengrass 版本 1.1.0 已发布	新功能： • 重置已部署的 AWS IoT Greengrass 群组。有关更多信息，请参阅重置部署。 • 除 Python 2.7 以外，还支持 Node.js 6.10 和 Java 8 Lambda 运行时。	2017 年 9 月 20 日
AWS IoT Greengrass 1.0.0 版本已发布	AWS IoT Greengrass 已正式上市。	2017 年 6 月 7 日