aws

开发人员指南

AWS IoT Core

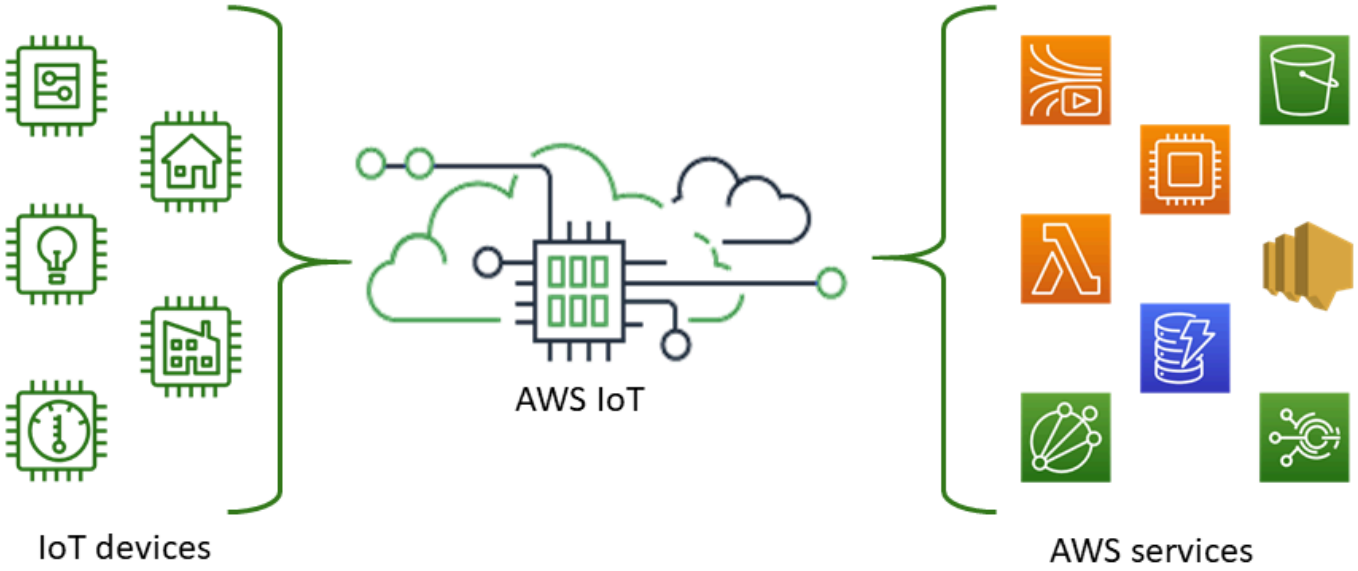
AWS IoT Core: 开发人员指南

Table of Contents

什么是 AWS IoT？

AWS IoT 提供将您的物联网设备连接到其他设备和 AWS 云服务的云服务。 AWS IoT 提供设备软件，可帮助您将物联网设备集成到 AWS IoT基于解决方案的解决方案中。如果您的设备可以连接 AWS IoT，则 AWS IoT 可以将它们连接到 AWS 提供的云服务。

有关动手操作的介绍 AWS IoT，请访问入门 AWS IoT Core。



AWS IoT 允许您为解决方案选择最合适的 up-to-date 技术。为了帮助您在现场管理和支持您的物联网设备， AWS IoT Core 支持以下协议：

- MQTT（消息队列和遥测传输）
- 采用 WSS 的 MQTT (Websockets Secure)
- HTTPS（安全超文本传输协议）
- LoRaWAN（长距离广域网）

AWS IoT Core 消息代理支持通过 WSS 协议使用 MQTT 和 MQTT 的设备和客户端来发布和订阅消息。它还支持使用 HTTPS 协议发布消息的设备和客户端。

AWS IoT Core fo LoRa r WAN 可帮助您连接和管理无线 LoRa WAN（低功耗远程广域网）设备。AWS IoT Core 对于 LoRa WAN，您无需开发和运行 LoRa WAN 网络服务器 (LNS)。

如果您不需要设备通信、规则或作业等 AWS IoT 功能，请参阅AWS 消息，了解可能更符合您要求的其他 AWS IoT 消息传递服务的信息。

您的设备和应用程序的访问方式 AWS IoT

AWS IoT 提供以下接口AWS IoT 教程：

- AWS IoT 设备软件开发工具包-在您的设备上构建用于向其发送消息和从中接收消息的应用程序。AWS IoT有关更多信息，请参阅 AWS IoT 设备 SDK、移动 SDK 和 AWS IoT 设备客户端。
- AWS IoT Core 适用于 LoRa WAN — 使用AWS IoT Core 适用于 WAN 来连接和管理您的远程 LoRa WAN (WAN) 设备和网关。 LoRa
- AWS Command Line Interface (AWS CLI) —在 Windows、macOS 和 Linux AWS IoT 上运行的命令。您可以使用这些命令创建并管理事物对象、证书、规则、任务和策略。要开始使用，请参阅 AWS Command Line Interface 用户指南。有关命令的更多信息 AWS IoT，请参阅《AWS CLI 命令参考》中的 i o t。
- AWS IoT API-使用 HTTP 或 HTTPS 请求构建您的物联网应用程序。您可以使用这些 API 操作以编程方式创建和管理事物对象、证书、规则及策略。有关的 API 操作的更多信息 AWS IoT，请参阅 AWS IoT API 参考中的操作。
- AWS 软件开发工具包-使用特定语言的 API 构建您的物联网应用程序。这些 SDK 中封装了 HTTP/HTTPS API，并且您可以用任何受支持的语言进行编程。有关更多信息，请参阅 AWS SDK 和工具。

您还可以 AWS IoT 通过AWS IoT 控制台进行访问，该控制台提供了一个图形用户界面 (GUI)，您可以通过该界面配置和管理物联网解决方案中的事物对象、证书、规则、作业、策略和其他元素。

AWS IoT 能做什么

本主题介绍了您可能需要 AWS IoT 支持的一些解决方案。

行业中的 IoT

以下是一些工业用例 AWS IoT 解决方案的示例，这些用例应用物联网技术来提高工业过程的性能和生产力。

用于工业使用案例的解决方案

- AWS IoT 用于在工业运营中建立预测性质量模型

  了解 AWS IoT 如何收集和分析来自工业运营的数据，以建立预测性质量模型。了解更多

- AWS IoT 用于支持工业运营中的预测性维护

  了解 AWS IoT 如何帮助规划预防性维护以减少计划外停机时间。了解更多

家居自动化中的 IoT

以下是一些家庭自动化用例的 AWS IoT 解决方案示例，这些用例应用物联网技术来构建可扩展的物联网应用程序，从而使用联网的家庭设备自动执行家庭活动。

家居自动化解决方案

- AWS IoT 在您的联网家庭中使用

  了解 AWS IoT 如何提供集成的家庭自动化解决方案。

- 用于 AWS IoT 提供家庭安全和监控

  了解 AWS IoT 如何将机器学习和边缘计算应用于您的家庭自动化解决方案。

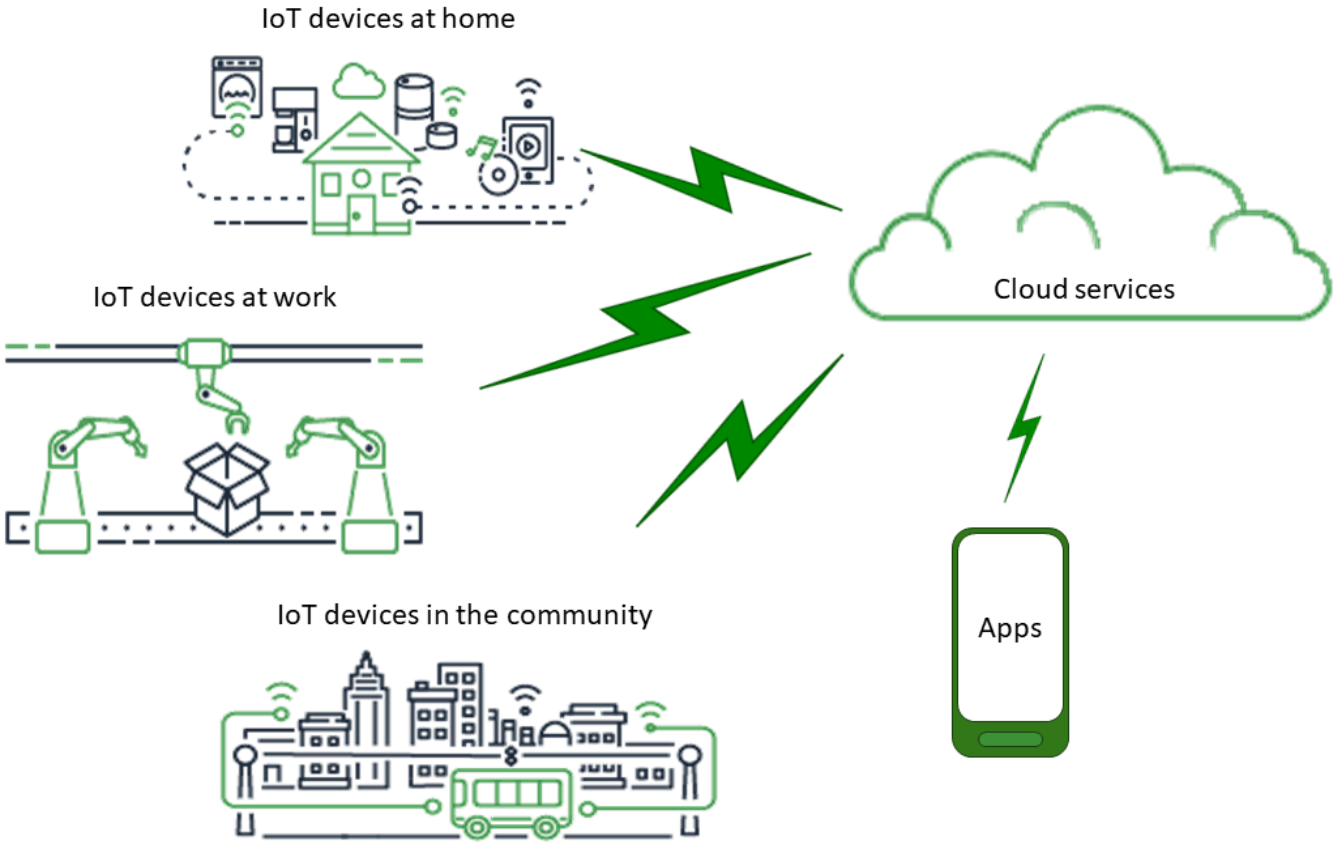
有关适用于工业、消费者和商业使用案例的解决方案列表，请参阅 AWS IoT 解决方案存储库。

如何 AWS IoT 运作

AWS IoT 提供云服务和设备支持，可用于实施物联网解决方案。 AWS 提供了许多云服务来支持基于 IoT 的应用程序。因此，为了帮助您了解从哪里开始，本部分提供了基本概念图表和定义，向您介绍 IoT 世界。

IoT 世界

一般来说，物联网 (IoT) 由下图所示的关键组成部分组成。



应用程序

应用程序让最终用户可以访问 IoT 设备以及与这些设备相连的云服务所提供的特征。

云服务

云服务是连接到互联网的分布式大规模数据存储和处理服务。示例包括：

- IoT 连接和管理服务

AWS IoT 是 IoT 连接和管理服务的一个示例。

- 计算服务，例如 Amazon 弹性计算云和 AWS Lambda
- 数据库服务，例如 Amazon DynamoDB

通信

设备通过使用各种技术和协议与云服务进行通信。示例包括：

- Wi-Fi/宽带互联网
- 宽带蜂窝数据
- 窄带蜂窝数据
- 远程广域网 (LoRaWAN)
- 专有的射频通信

设备

设备是管理接口和通信的一种硬件。设备通常位于其监控和控制的真实接口附近。设备可以包括计算和存储资源，例如微控制器、CPU、内存。示例包括：

- Raspberry Pi
- Arduino
- 语音接口助手
- LoRa广域网和设备
- Amazon Sidewalk 设备
- 自定义 IoT 设备

接口

接口是将设备连接到物理世界的组件。

- 用户接口

  允许设备和用户相互通信的组件。

  - 输入接口

使用户能够与设备通信

示例：键盘、按钮

- 输出接口

使设备能够与用户通信

示例：字母数字显示、图形显示、指示灯、闹铃

- 传感器

以设备理解的方式测量或感知外部世界中的某些内容的输入组件。示例包括：

- 温度传感器（将温度转换为模拟或数字信号）
- 湿度传感器（将相对湿度转换为模拟数字信号）
- 模拟到数字转换器（将模拟电压转换为数值）
- 超声波距离测量装置（将距离转换为数值）
- 光学传感器（将光度转换为数值）
- 相机（将图像数据转换为数字数据）

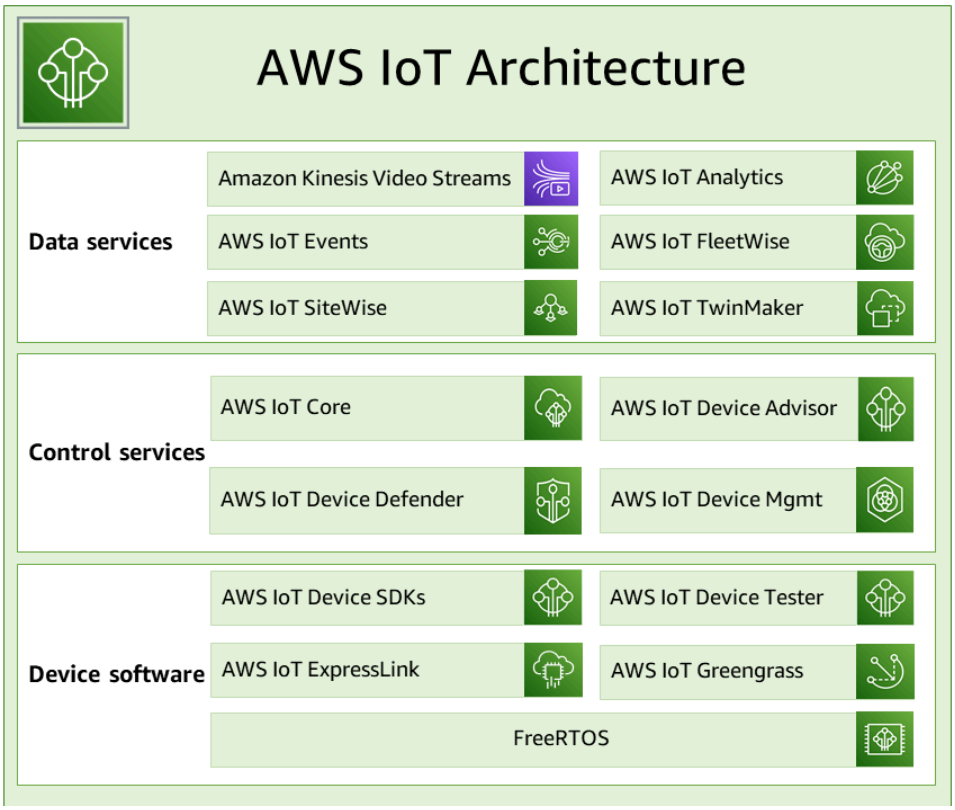
- 执行器

设备可用于控制外部世界中的某些内容的输出组件。示例包括：

- 步进电机（将电信号转换为移动）
- 继电器（控制高电压和电流）

AWS IoT 服务概述

在物联网领域，AWS IoT 提供支持与世界交互的设备以及它们之间传输的数据的服务 AWS IoT。AWS IoT 由本插图中显示的支持您的 IoT 解决方案的服务组成。



AWS IoT 设备软件

AWS IoT 提供此软件来支持您的物联网设备。

AWS IoT 设备软件开发工具包

AWS IoT 设备和移动 SDK 可帮助您高效地将设备连接到。 AWS IoT AWS IoT 设备和移动 SDK 包括开源库、带示例的开发者指南和移植指南，因此您可以在自己选择的硬件平台上构建创新的物联网产品或解决方案。

AWS IoT Device Tester

AWS IoT Device Tester适用于 FreeRTOS AWS IoT Greengrass ，是一款用于微控制器的测试自动化工具。 AWS IoT Device Tester 测试您的设备以确定它是否会运行 FreeRTOS AWS IoT Greengrass 或与服务互操作。 AWS IoT

AWS IoT ExpressLink

AWS IoT ExpressLink 为AWS 合作伙伴开发和提供的一系列硬件模块提供动力。连接模块包括AWS经过验证的软件，使您可以更快、更轻松地将设备安全地连接到云端，并与一系列 AWS 服务无缝集成。有关更多信息，请访问AWS IoT ExpressLink概述页面或参阅《AWS IoT ExpressLink程序员指南》。

AWS IoT Greengrass

AWS IoT Greengrass扩展 AWS IoT 到边缘设备，因此它们可以根据自己生成的数据在本地采取行动，根据机器学习模型进行预测，以及筛选和聚合设备数据。 AWS IoT Greengrass 使您的设备能够在离数据生成地点更近的地方收集和分析数据，对本地事件做出自主反应，并与本地网络上的其他设备进行安全通信。您可以使用预先构建的软件模块（称为组件） AWS IoT Greengrass 来构建边缘应用程序，这些模块可以将您的边缘设备连接到 AWS 服务或第三方服务。

FreeRTOS

FreeRTOS 是一款面向微控制器的开源实时操作系统，可让您在 IoT 解决方案中包含小型低功耗边缘设备。FreeRTOS 包括一个内核和越来越多的、支持许多应用程序的软件库。FreeRTOS 系统可以安全地将小型低功耗设备连接到 AWS IoT，并支持运行 AWS IoT Greengrass 的更强大的边缘设备。

AWS IoT 控制服务

连接到以下 AWS IoT 服务以管理您的 IoT 解决方案中的设备。

AWS IoT Core

AWS IoT Core是一项托管云服务，使联网设备能够安全地与云应用程序和其他设备进行交互。AWS IoT Core 可以支持许多设备和消息，它可以处理这些消息并将其路由到 AWS IoT 端点和其他设备。借 AWS IoT Core助，您的应用程序可以与您的所有设备进行交互，即使这些设备未连接也是如此。

AWS IoT Core 设备顾问

AWS IoT Core Device Advisor 是一种基于云的完全托管式测试功能，用于在设备软件开发过程中验证 IoT 设备。Device Advisor 提供预先构建的测试，在将设备部署到生产环境之前 AWS IoT Core，您可以使用这些测试来验证物联网设备的可靠性和安全连接。

AWS IoT 设备防御者

AWS IoT Device Defender 可帮助您保护您的物联网设备群。 AWS IoT Device Defender 会持续审核您的物联网配置，以确保它们不会偏离安全最佳实践。 AWS IoT Device Defender 检测到您的物联网配置中存在任何可能造成安全风险的漏洞（例如在多个设备之间共享身份证书，或者正在尝试连接身份证书已吊销的设备正在尝试连接）时，它会发送警报。AWS IoT Core

AWS IoT 设备管理

AWS IoT 设备管理服务可帮助您跟踪、监控和管理构成设备群的大量联网设备。 AWS IoT 设备管理服务可帮助您确保物联网设备在部署后能够正常安全地运行。它们提供安全隧道来访问您的设备、监控设备的运行状况、检测和远程排查问题，还提供管理设备软件和固件更新的服务。

AWS IoT 数据服务

使用以下 AWS IoT 服务分析来自物联网解决方案中设备的数据，并采取适当的措施。

Amazon Kinesis Video Streams

Amazon Kinesis V ideo Streams 允许您将直播视频从设备流式传输到 AWS 云端，在云端对视频进行持久存储、加密和索引，从而允许您通过 API 访问数据。 easy-to-use您可以使用 Amazon Kinesis Video Streams 捕获来自数百万种源（包括智能手机、安全摄像头、网络摄像头、车载摄像头、无人机及其他源）的海量实时视频数据。借助 Amazon Kinesis Video Streams，您可以播放视频以进行实时和点播观看，并通过与 Amazon Rekognition Video 和机器学习框架库集成来快速构建利用计算机视觉和视频分析的应用程序。您也可以发送非视频时间序列化数据，如音频数据、热成像、深度数据、雷达数据等。

Amazon Kinesis Video Streams with WebRTC

Amazon Kinesis Video Streams with WebRTC 提供符合标准的 WebRTC 实施作为完全托管式功能。您可以使用 Amazon Kinesis Video Streams with WebRTC 安全地进行媒体的实时流式传输，或在任何摄像头 IoT 设备与符合 WebRTC 的移动或 Web 播放器之间执行双向音频或视频交互。借助这项全面托管的功能，您不必构建、运营或扩展任何与 WebRTC 相关的云基础设施（例如信令或媒体中继服务器）便能安全地在应用程序和设备间流式传输媒体。使用带有 WebRTC 的 Amazon Kinesis Video Streams，您可以轻松构建用于 peer-to-peer 直播媒体流的应用程序，或者针对各种用例在摄像机物联网设备、网络浏览器和移动设备之间进行实时音频或视频交互。

AWS IoT 分析

AWS IoT Analytics 可让您高效地对大量非结构化物联网数据运行和实施复杂的分析。 AWS IoT Analytics 可自动执行分析来自物联网设备的数据所需的每个困难步骤。 AWS IoT Analytics 会筛选、转换和丰富物联网数据，然后将其存储在时间序列数据存储中进行分析。通过使用内置的 SQL 查询引擎或机器学习，您可以运行一次性查询或计划查询来分析数据。

AWS IoT 活动

AWS IoT 事件可检测并响应来自物联网传感器和应用程序的事件。事件是指识别比预期更复杂的情况的数据模式，例如使用运动信号激活灯光的运动探测器和安全摄像头。 AWS IoT Events

持续监控来自多个物联网传感器和应用程序的数据，并与其他服务（例如物联网 AWS IoT Core SiteWise、DynamoDB 等）集成，以实现早期检测和独特见解。

AWS IoT FleetWise

AWS IoT FleetWise是一项托管服务，可用于近乎实时地收集车辆数据并将其传输到云端。借 AWS IoT FleetWise助，您可以轻松收集和整理来自使用不同协议和数据格式的车辆的数据。 AWS IoT FleetWise 有助于将低级消息转换为人类可读的值，并标准化云端的数据格式以进行数据分析。您也可以制定数据收集方案，控制从车辆中收集哪些数据以及何时将该数据传输到云。

AWS IoT SiteWise

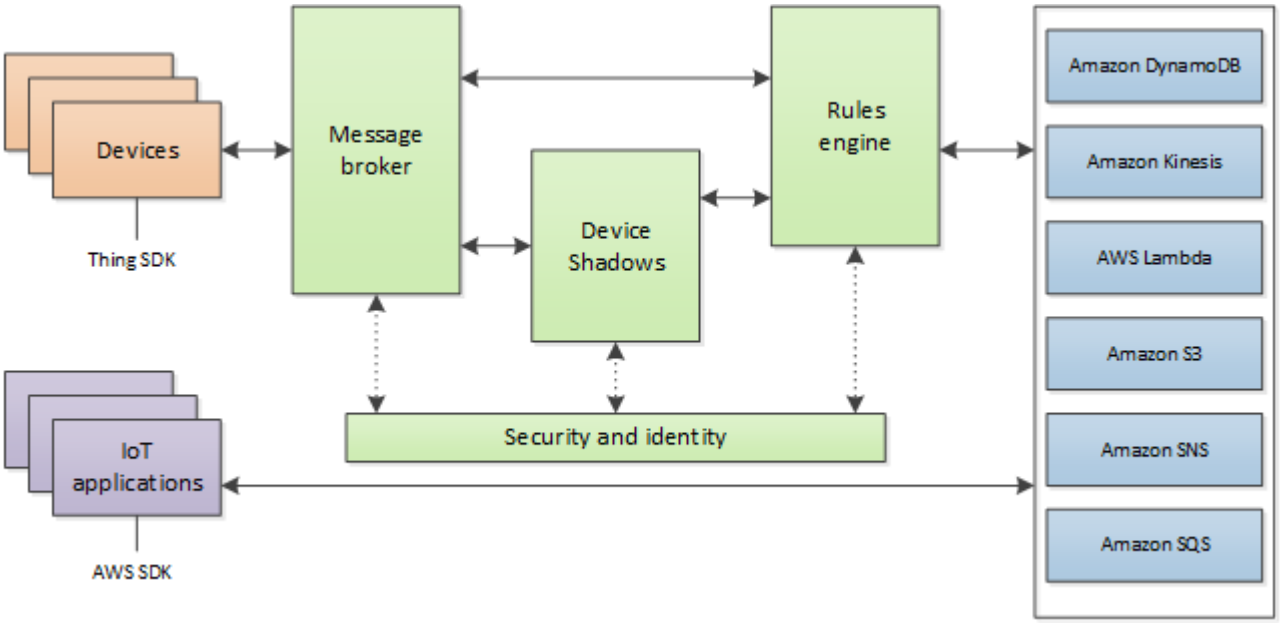
AWS IoT SiteWise通过提供在设施网关上运行的软件，大规模收集、存储、组织和监控通过 MQTT 消息或 API 从工业设备传递的数据。该网关可以安全地连接到您的本地数据服务器，并自动执行收集和组织数据并将其发送到 AWS 云端的过程。

AWS IoT TwinMaker

AWS IoT TwinMaker构建物理和数字系统的可操作数字双胞胎。 AWS IoT TwinMaker 使用来自各种现实世界传感器、摄像头和企业应用程序的测量和分析来创建数字可视化效果，以帮助您跟踪实际工厂、建筑物或工业厂房。您可以使用真实世界的数据来监控运营、进行诊断和纠正错误以及优化运营。

AWS IoT Core 服务

AWS IoT Core 提供将您的物联网设备连接到 AWS 云端的服务，以便其他云服务和应用程序可以与您的互联网连接设备进行交互。



下一节将介绍插图中显示的每项 AWS IoT Core 服务。

AWS IoT Core 消息服务

AWS IoT Core 连接服务提供与物联网设备的安全通信，并管理它们与之间传递的消息 AWS IoT。

设备网关

使设备能够安全高效地与 AWS IoT进行通信。设备通信由使用 X.509 证书的安全协议提供保护。

消息代理

为设备和 AWS IoT 应用程序提供一种安全机制，以相互发布和接收消息。您可以直接使用 MQTT 协议，也可以使用 MQTT WebSocket 进行发布和订阅。有关受 AWS IoT 支持的协议的更多信息，请参阅 the section called “设备通信协议”。设备和客户端也可以使用 HTTP REST 接口将数据发布到消息代理。

消息代理将设备数据分发给已订阅该消息的设备以及其他 AWS IoT Core 服务，例如 Device Shadow 服务和规则引擎。

AWS IoT Core 适用于 LoRa广域网

AWS IoT Core for LoRa WAN AWS 无需开发和运行 LoRa WAN 网络服务器 (LNS)，即可通过将 LoRa WAN 设备和网关连接到，从而建立专用 LoRa WAN 网络。从 LoRa WAN 设备收到的消息将发送到规则引擎，在那里可以对其进行格式化并发送到其他 AWS IoT 服务。

规则引擎

规则引擎将消息代理中的数据连接到其它 AWS IoT 服务以进行存储和额外处理。例如，您可以插入、更新或查询 DynamoDB 表，也可以根据在规则引擎中定义的表达式调用 Lambda 函数。您可以使用基于 SQL 的语言选择消息负载中的数据，然后处理数据并将数据发送到其它服务，如 Amazon Simple Storage Service (Amazon S3)、Amazon DynamoDB 和 AWS Lambda。您还可以创建规则以将消息重新发布到消息代理并面向其他订阅者。有关更多信息，请参阅 的规则 AWS IoT。

AWS IoT Core 控制服务

AWS IoT Core 控制服务提供设备安全、管理和注册功能。

自定义身份验证服务

您可以定义自定义授权方，从而通过自定义的身份验证服务和 Lambda 函数来管理自己的身份验证和授权策略。自定义授权器 AWS IoT 允许使用持有者令牌身份验证和授权策略对您的设备进行身份验证并授权操作。

自定义授权方可以实施各种身份验证策略；例如 JSON Web 令牌验证或 OAuth 提供程序标注。它们必须返回设备网关用于授权 MQTT 操作的策略文档。有关更多信息，请参阅 自定义身份验证和授权。

设备预配置服务

让您可使用描述设备所需资源的模板来预配置设备：事物对象、证书以及一个或多个策略。事物对象是注册表中的一个条目，其中包含描述设备的属性。设备使用证书进行身份验证 AWS IoT。策略确定设备在 AWS IoT中可执行的操作。

模板包含可用字典中的值替换的变量 (映射)。您可以使用同一个模板来预配置多个设备，只需在字典中为模板变量传递不同的值。有关更多信息，请参阅 设备预调配。

组注册表

通过将设备按类别分成不同的组，可以使用组来同时管理多台设备。组中还可以包含组，您可以构建组的层次结构。您对父组执行的任何操作都将应用于其子组。同样的操作也应用于父组中的所有设备以及子组中的所有设备。向某个组授予的权限将应用于该组及其子组中的所有设备。有关更多信息，请参阅 使用管理设备 AWS IoT。

Jobs 服务

允许您定义一组远程操作，这些操作将被发送到一个或多个连接到 AWS IoT的设备并在这些设备上运行。例如，您可以定义一个任务，该任务指示一组设备下载并安装应用程序或固件更新、重启、轮换证书或执行远程故障排除操作。

要创建任务，您需要指定要执行的远程操作的说明，以及应该执行这些操作的目标列表。目标可以是单个设备和/或设备组。有关更多信息，请参阅 任务。

注册表

整理与 AWS 云中的每台设备关联的资源。您可以注册自己的设备并将每台设备与最多三个自定义属性关联。您还可以将每台设备与相应的证书和 MQTT 客户端 ID 关联，以提高对设备进行管理和故障排除的能力。有关更多信息，请参阅 使用管理设备 AWS IoT。

安全和身份服务

为 AWS 云中的安全提供共担责任。为了安全地将数据发送到消息代理，您的设备必须确保自身凭证的安全。消息代理和规则引擎使用 AWS 安全特征将数据安全发送到设备或其它 AWS 服务。有关更多信息，请参阅 身份验证。

AWS IoT Core 数据服务

AWS IoT Core 数据服务可帮助您的物联网解决方案提供可靠的应用体验，即使设备并非始终处于连接状态。

Device Shadow

一种 JSON 文档，用于存储和检索设备的当前状态信息。

Device Shadow 服务

Device Shadow 服务会保持设备的状态，以便应用程序可以与设备通信，无论设备是否在线。当设备处于离线状态时，Device Shadow 服务将管理它连接的应用程序的数据。当设备重新连接时，它会将其状态与 Device Shadow 服务中的状态同步。您的设备还可以将有关其当前状态的信息发布到影子，以供不会始终保持连接的应用程序或其它设备使用。有关更多信息，请参阅 AWS IoT Device Shadow 服务。

AWS IoT Core 支持服务

适用于 Amazon 人行道集成 AWS IoT Core

Amazon Sidewalk 是一个共享网络，可改进连接选项，帮助设备更好地协同工作。Amazon Sidewalk 支持各种客户设备，例如定位宠物或贵重物品的设备、提供智能家居安全和照明控制的设备，以及为家电和工具提供远程诊断的设备。Amazon Sidewalk Integration AWS IoT Core 使设备制造商能够将他们的 Sidewalk 设备群添加到云端。 AWS IoT

有关更多信息，请参阅 适用于 Amazon Sidewalk 的 AWS IoT Core。

了解更多关于 AWS IoT

本主题可帮助您熟悉以下世界 AWS IoT. 您可以获得有关如何在各种用例中应用物联网解决方案的一般信息、培训资源、所有其他 AWS 服务的社交媒体链接， AWS IoT 以及 AWS IoT 使用的服务和通信协议列表。

的培训资源 AWS IoT

我们提供这些培训课程是为了帮助您了解 AWS IoT 以及如何将其应用到您的解决方案设计中。

- 简介 AWS IoT

  视频概述 AWS IoT 及其核心服务。

- 深入了解 AWS IoT 身份验证和授权

  一门探讨 AWS IoT 身份验证和授权概念的高级课程。您将学习如何对客户端进行身份验证和授权以访问 AWS IoT 控制平面和数据平面 API。

- 物联网基础系列

  关于不同 IoT 技术和特征的 IoT 电子学习模块的学习路径。

AWS IoT 资源和指南

这些是有关特定方面的深入技术资源 AWS IoT。

- 物联网镜头 — WellArch AWS IoT itected 框架

  该文档描述了在上架构 IoT 应用程序的最佳实践。 AWS

- 为以下对象设计 MQTT 主题 AWS IoT Core

  一份白皮书，描述了在 MQTT 中设计 MQTT 主题 AWS IoT Core 和利用 MQTT AWS IoT Core 功能的最佳实践。

- 摘要和简介

  一个 PDF 文档，描述了为配置大量设备而 AWS IoT 提供的不同方式。

- AWS IoT Core Device Advisor

  AWS IoT Core Device Advisor 提供了预先构建的测试，在将设备部署到生产环境之前，您可以使用这些测试来验证物联网设备的可靠性和安全连接最佳实践。 AWS IoT Core

- AWS IoT 资源

  IoT 特定的资源，例如技术指南、参考架构、电子书和精选的博客文章，在可搜索索引中显示。

- IoT Atlas

有关如何解决常见的 IoT 设计问题的概览。IoT Atlas 深入探讨了您在开发 IoT 解决方案时可能遇到的设计挑战。

- AWS 白皮书和指南

  我们目前收集的白皮书和指南 AWS IoT 以及其他 AWS 技术。

AWS IoT 在社交媒体上

这些社交媒体渠道提供有关 AWS IoT 和 AWS相关主题的信息。

- 物联网开启 AWS IoT — 官方博客
- AWS IoT 亚马逊 Web Services 频道中的视频 YouTube

这些社交媒体账户涵盖所有 AWS 服务，包括 AWS IoT

- 亚马逊 Web Services 频道开启 YouTube
- Twitter 上的 Amazon Web Services
- Facebook 上的 Amazon Web Services
- Instagram 上的 Amazon Web Services
- 亚马逊 Web Services 开启 LinkedIn

AWSAWS IoT Core 规则引擎使用的服务

AWS IoT Core 规则引擎可以连接到这些 AWS 服务。

- Amazon DynamoDB

  Amazon DynamoDB 是一种可扩展的 NoSQL 数据库服务，可提供快速且可预测的数据库性能。

- Amazon Kinesis

  Amazon Kinesis 可让您轻松收集、处理和分析实时串流数据，以便您及时获得见解并对新信息快速做出响应。Amazon Kinesis 可以摄取视频、音频、应用程序日志和网站点击流等实时数据，也可以获取用于机器学习、分析和其它应用程序的 IoT 遥测数据。

- AWS Lambda

AWS Lambda 允许您在不预置或管理服务器的情况下运行代码。您可以将代码设置为根据 AWS IoT 数据和事件自动触发，或者直接从 Web 或移动应用程序调用。

- Amazon Simple Storage Service

  Amazon Simple Storage Service (Amazon S3) 可以随时从网络上的任何地方存储和检索任意数量的数据。 AWS IoT 规则可以将数据发送到 Amazon S3 进行存储。

- Amazon Simple Notification Service

  Amazon Simple Notification Service (Amazon SNS) 是一项 Web 服务，通过该服务，应用程序、终端用户和设备可以发送和接收来自云的通知。

- Amazon Simple Queue Service

  Amazon Simple Queue Service (Amazon SQS) 是一种消息队列服务，可解耦和扩展微服务、分布式系统和无服务器应用程序。

- 亚马逊 OpenSearch 服务

  Amazon Service（ OpenSearch OpenSearch 服务）是一项托管服务，可轻松部署、操作和扩展 OpenSearch，是一种流行的开源搜索和分析引擎。

- Amazon SageMaker

  Amazon SageMaker 可以通过在您的物联网数据中查找模式来创建机器学习 (ML) 模型。然后，该服务会使用这些模型处理新数据并为应用程序生成预测结果。

- Amazon CloudWatch

  Amazon CloudWatch 提供可靠、可扩展且灵活的监控解决方案，帮助您设置、管理和扩展自己的监控系统和基础设施。

AWS IoT Core支持的通信协议

这些主题提供了有关 AWS IoT使用的通信协议的更多信息。有关设备和服务所使用的协议 AWS IoT 以及与之连接的协议的更多信息 AWS IoT，请参阅正在连接到 AWS IoT Core。

- MQTT（消息队列遥测传输）

  MQTT.org 网站的主页，您可以在其中找到 MQTT 协议规范。有关如何 AWS IoT 支持 MQTT 的更多信息，请参阅MQTT。

- HTTPS（安全超文本传输协议）

设备和应用程序可以使用 HTTPS 访问 AWS IoT 服务。

- LoRaWAN（长距离广域网）

  LoRaWAN 设备和网关 AWS IoT Core 可以通过用 AWS IoT Core 于 LoRa WAN 进行连接。

- TLS（传输层安全）v1.3

  TLS v1.3 (RFC 5246) 的规范。 AWS IoT 使用 TLS v1.3 在设备和 AWS IoT之间建立安全连接。

AWS IoT 控制台中的新增功能

我们正在致力更新 AWS IoT 控制台的用户界面以提供新的体验。我们正在分阶段更新用户界面，因此控制台中的某些页面将具有新的体验，有些可能同时具有原始体验和新体验，而有些可能只有原始体验。

此表显示了截至 2022 年 1 月 27 日 AWS IoT 控制台用户界面中各个区域的状态。

AWS IoT 控制台用户界面状态

控制台页面	原始体验	新体验	注释
监控	不可用	可用	
活动	不可用	可用	
注册 - 入门	不可用	可用	（并未在 CN 区域提供）
注册 - 实例集预调配模板	可用	可用	
管理 — 事物	可用	可用	
管理 — 类型	可用	可用	
管理 — 事物组	可用	可用	
管理 — 账单组	可用	可用	
管理 — 任务	可用	可用	

控制台页面	原始体验	新体验	注释
管理 — 任务模板	不可用	可用	
管理 — 隧道	不可用	可用	
Fleet Hub — 入门	不可用	可用	在所有 AWS 区域 中暂不可用
Fleet Hub — 应用程序	不可用	可用	在所有 AWS 区域 中暂不可用
Greengrass — 入门	不可用	可用	在所有 AWS 区域 中暂不可用
Greengrass — 核心设备	不可用	可用	在所有 AWS 区域 中暂不可用
Greengrass — 组件	不可用	可用	在所有 AWS 区域 中暂不可用
Greengrass — 部署	不可用	可用	在所有 AWS 区域 中暂不可用
Greengrass — 经典 (V1)	可用	可用	
无线连接 — 简介	不可用	可用	在所有 AWS 区域 中暂不可用
无线连接 — 网关	不可用	可用	在所有 AWS 区域 中暂不可用
无线连接 — 设备	不可用	可用	在所有 AWS 区域 中暂不可用
无线连接 — 配置文件	不可用	可用	在所有 AWS 区域 中暂不可用

控制台页面	原始体验	新体验	注释
无线连接 — 目标	不可用	可用	在所有 AWS 区域 中暂不可用
安全 — 证书	可用	可用	
安全 — 策略	可用	可用	
安全 — CA	可用	可用	
安全 — 角色别名	可用	可用	
安全 — 授权方	可用	可用	
防护 — 简介	不可用	可用	
防护 — 审计	不可用	可用	
防护 — 检测	不可用	可用	
防护 — 缓解操作	不可用	可用	
防护 — 设置	不可用	可用	
操作 — 规则	可用	可用	
操作 — 目标	可用	可用	
测试 — Device Advisor	可用	可用	在所有 AWS 区域 中暂不可用
测试 — MQTT 测试客户端	可用	可用	
软件	可用	可用	
设置	不可用	可用	
学习	可用	暂不可用	

图例

状态值

- 可用

  此用户界面体验可以使用。
- 不可用

  此用户界面体验无法使用。
- 暂不可用

  正在开发新的用户界面体验，但尚未准备好。
- 正在进行中

  新的用户界面体验正在更新。但是，某些页面可能仍然具有原始用户体验。

AWS IoT 与 AWS SDK 一起使用

AWS 软件开发套件 (SDK) 可用于许多流行的编程语言。每个软件开发工具包都提供 API、代码示例和文档，使开发人员能够更轻松地以其首选语言构建应用程序。

SDK 文档	代码示例
AWS SDK for C++	AWS SDK for C++ 代码示例
AWS CLI	AWS CLI 代码示例
AWS SDK for Go	AWS SDK for Go 代码示例
AWS SDK for Java	AWS SDK for Java 代码示例
AWS SDK for JavaScript	AWS SDK for JavaScript 代码示例
AWS SDK for Kotlin	AWS SDK for Kotlin 代码示例
AWS SDK for .NET	AWS SDK for .NET 代码示例
AWS SDK for PHP	AWS SDK for PHP 代码示例

SDK 文档	代码示例
AWS Tools for PowerShell	PowerShell 代码示例工具
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 代码示例
AWS SDK for Ruby	AWS SDK for Ruby 代码示例
AWS SDK for Rust	AWS SDK for Rust 代码示例
适用于 SAP ABAP 的 AWS SDK	适用于 SAP ABAP 的 AWS SDK 代码示例
AWS SDK for Swift	AWS SDK for Swift 代码示例

示例可用性

找不到所需的内容？ 通过使用此页面底部的提供反馈链接请求代码示例。

入门 AWS IoT Core

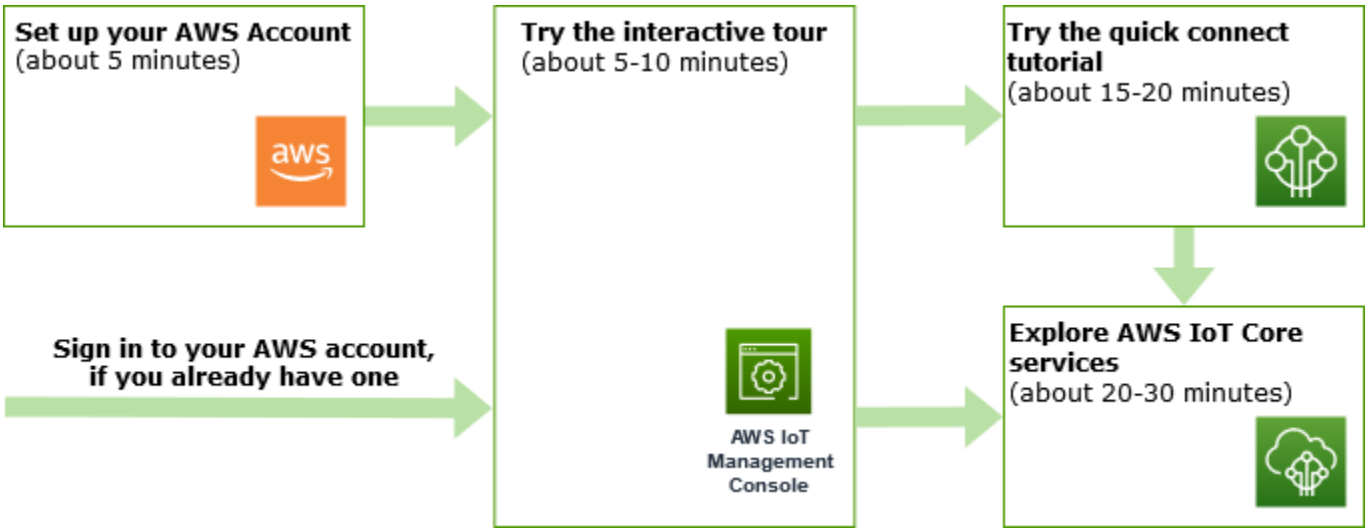
无论您是物联网新手还是拥有多年经验，这些资源都提供了可以帮助您开始使用的 AWS IoT 概念和术语 AWS IoT。

- 看看里面 AWS IoT 及其组件如何 AWS IoT 运作。
- 利用我们的培训材料和视频集详细了解AWS IoT。本主题还包括 AWS IoT 可连接的服务列表、社交媒体链接以及通信协议规范的链接。
- the section called “将您的第一台设备连接到 AWS IoT Core”.
- 使用 正在连接到 AWS IoT Core 开发您的 IoT 解决方案并探索 AWS IoT 教程。
- 使用 Device Advisor 测试和验证 您的 IoT 设备，确保安全可靠的连接。
- 使用诸如 机群索引、任务 和 AWS IoT Device Defender 等 AWS IoT Core 管理服务管理您的解决方案。
- 使用 AWS IoT 数据服务 分析设备的数据。

将您的第一台设备连接到 AWS IoT Core

AWS IoT Core 服务将物联网设备连接到 AWS IoT 服务和其他 AWS 服务。 AWS IoT Core 包括设备网关和消息代理，它们连接和处理物联网设备与云之间的消息。

以下是您可以开始使用 AWS IoT Core 和的方法 AWS IoT。



本节介绍 AWS IoT Core 以介绍其主要服务，并提供几个示例，说明如何将设备连接到设备 AWS IoT Core 并在设备之间传递消息。在设备和云端之间传递消息是每个物联网解决方案的基础，也是您的设备与其他 AWS 服务进行交互的方式。

- 设置你的 AWS 账户

  在使用 AWS IoT 服务之前，必须先设置 AWS 账户。如果您自己已经有一个 AWS 账户 和一个 IAM 用户，则可以使用它们并跳过此步骤。

- 尝试交互式教程

  如果您想了解基本 AWS IoT 解决方案在不连接设备或下载任何软件的情况下可以做什么，那么这个演示最好。交互式教程介绍了基于 AWS IoT Core 服务构建的模拟解决方案，该解决方案说明了它们是如何交互的。

- 尝试快速连接教程

  如果你想快速入门 AWS IoT 并了解它在有限的场景中是如何工作的，那么本教程是最好的。在本教程中，你需要一台设备并在上面安装一些 AWS IoT 软件。如果您没有 IoT 设备，可使用 Windows、Linux 或 macOS 个人计算机作为本教程的设备。如果你想试一试 AWS IoT，但没有设备，请尝试下一个选项。

- 通过动手教程探索 AWS IoT Core 服务

  本教程最适合想要入门的开发者， AWS IoT 这样他们就可以继续探索其他 AWS IoT Core 功能，例如规则引擎和阴影。本教程遵循的流程类似于快速连接教程，但为每个步骤提供了更多详细信息，以便学员能够更平滑地过渡到更高级的教程。

- 使用 MQTT 客户端查看 AWS IoT MQTT 消息

  了解如何使用 MQTT 测试客户端观看您的第一台设备将 MQTT 消息发布到 AWS IoT。MQTT 测试客户端可用于监控和排除设备连接故障。

> **Note**
>
> 如果您想尝试其中的多个入门教程或重复同一教程，则应在开始另一个教程之前删除在前一个教程中创建的事物对象。如果您没有从前面的教程中删除事物对象，则需要在后续教程中使用不同的事物名称。这是因为事物名称在您的账户和 AWS 区域中必须是唯一的。

有关的更多信息 AWS IoT Core，请参阅什么是 AWS IoT Core？

设置你的 AWS 账户

在 AWS IoT Core 首次使用之前，请完成以下任务：

主题

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。

- 打开控制 AWS IoT 台

如果你自己已经有一个 AWS 账户 和一个用户，你可以使用它们并直接跳到the section called “打开控制 AWS IoT 台”。

打开控制 AWS IoT 台

本节中大多数面向控制台的主题都是从控制台开始的。 AWS IoT 如果您尚未登录您的 AWS 账户，请登录，然后打开AWS IoT 主机并继续下一部分继续开始使用 AWS IoT。

试试 AWS IoT Core 互动教程

该交互式教程展示了基于 AWS IoT构建的简单 IoT 解决方案的组成部分。本教程的动画展示了物联网设备如何与 AWS IoT Core 服务交互。本主题提供了 AWS IoT Core 交互式教程的预览。控制台中的图包含未出现在本教程中的图中的动画。

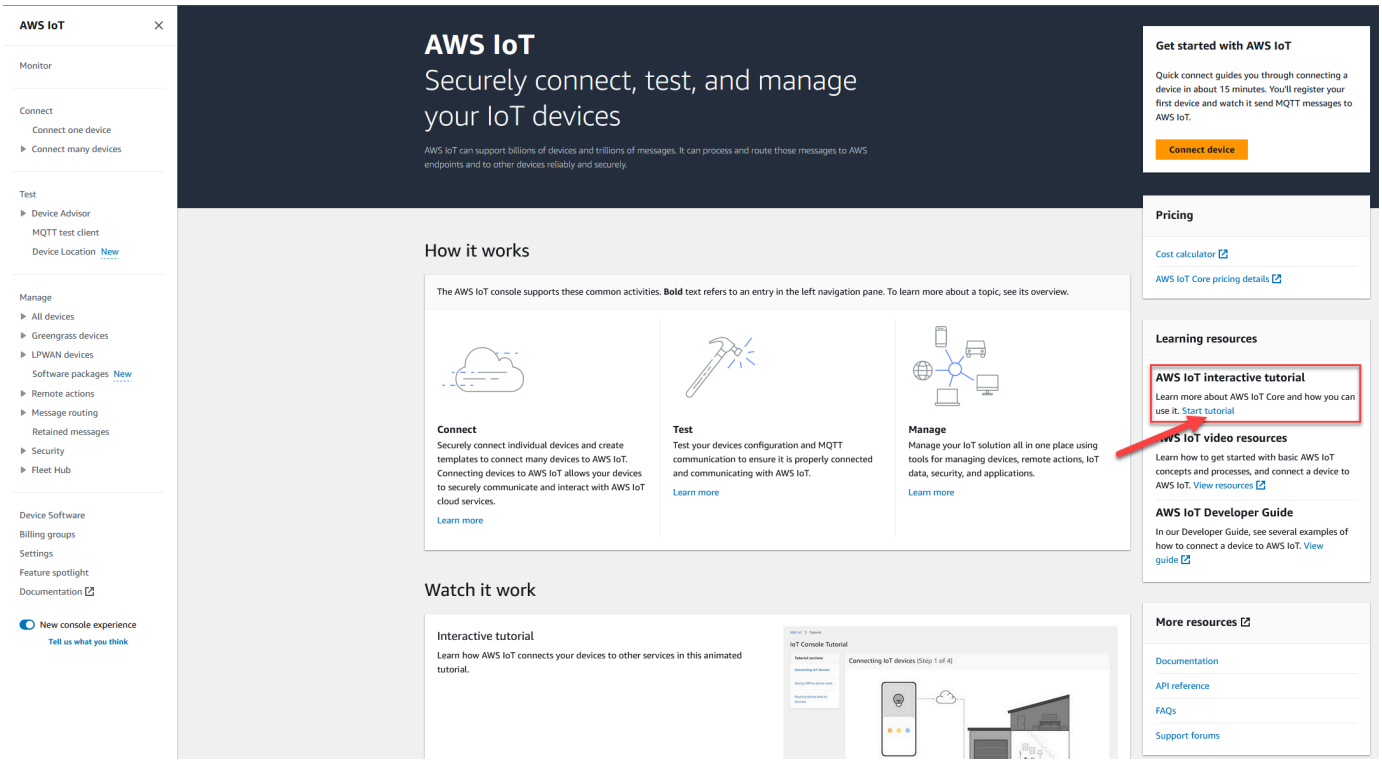
要运行演示，您必须首先the section called “设置你的 AWS 账户”。但是，本教程不需要任何 AWS IoT 资源、额外的软件或任何编码。

预计需要花大约 5-10 分钟来观看此演示。给自己留 10 分钟将有更多时间来理解每个步骤。

运行交 AWS IoT Core 互式教程

1. 在 AWS IoT 控制台中打开AWS IoT 主页。

   在 AWS IoT 主页上的学习资源窗格中，选择开始教程。



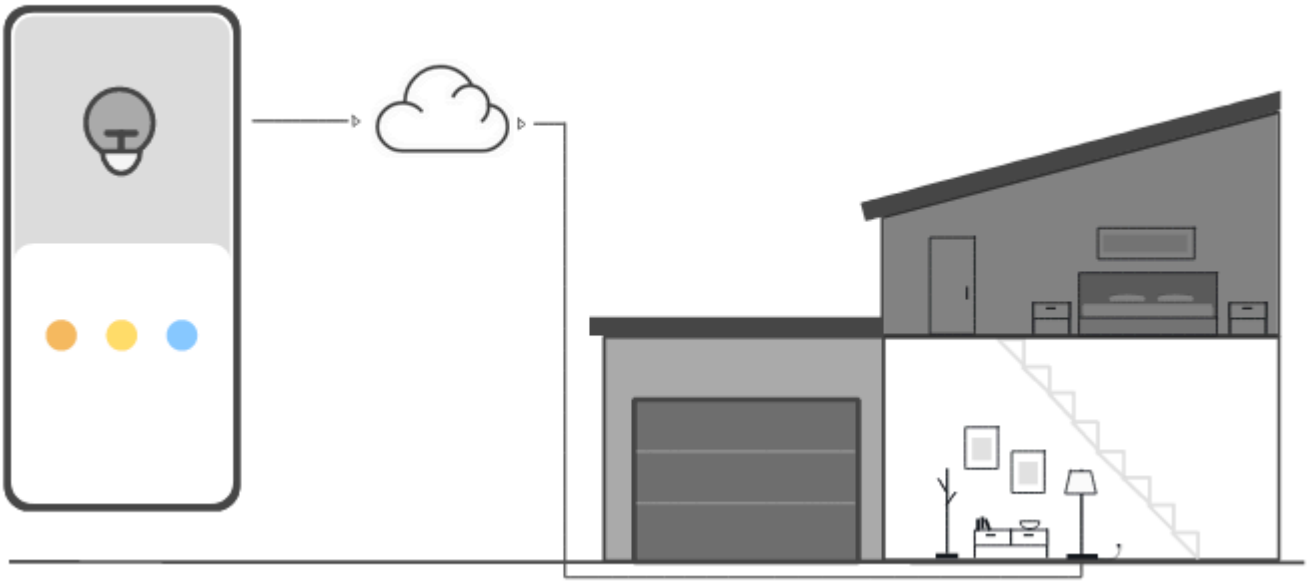
2. 在 AWS IoT 控制台教程页面中，查看教程部分，然后在准备好继续时选择开始部分。

以下各节介绍AWS IoT 控制台教程如何呈现这些 AWS IoT Core 功能：

- 连接 IoT 设备
- 保存离线设备状态
- 将设备数据路由到服务

连接 IoT 设备

了解物联网设备如何与之通信 AWS IoT Core。

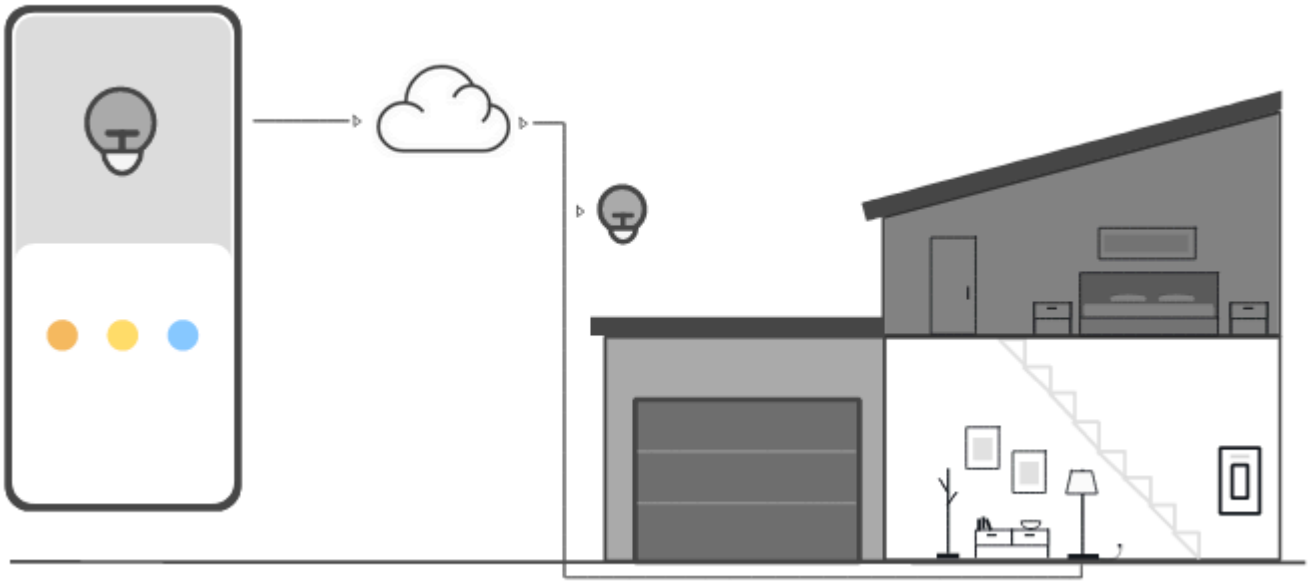


此步骤中的动画展示了两台设备（左边的控制设备和右侧房子里的智能灯）如何连接云中的 AWS IoT Core 以及与之通信。动画显示设备与收到的消息进行通信 AWS IoT Core 并做出反应。

有关将设备连接至的更多信息 AWS IoT Core，请参阅正在连接到 AWS IoT Core。

保存离线设备状态

了解如何在设备或应用程序离线时 AWS IoT Core 保存设备状态。



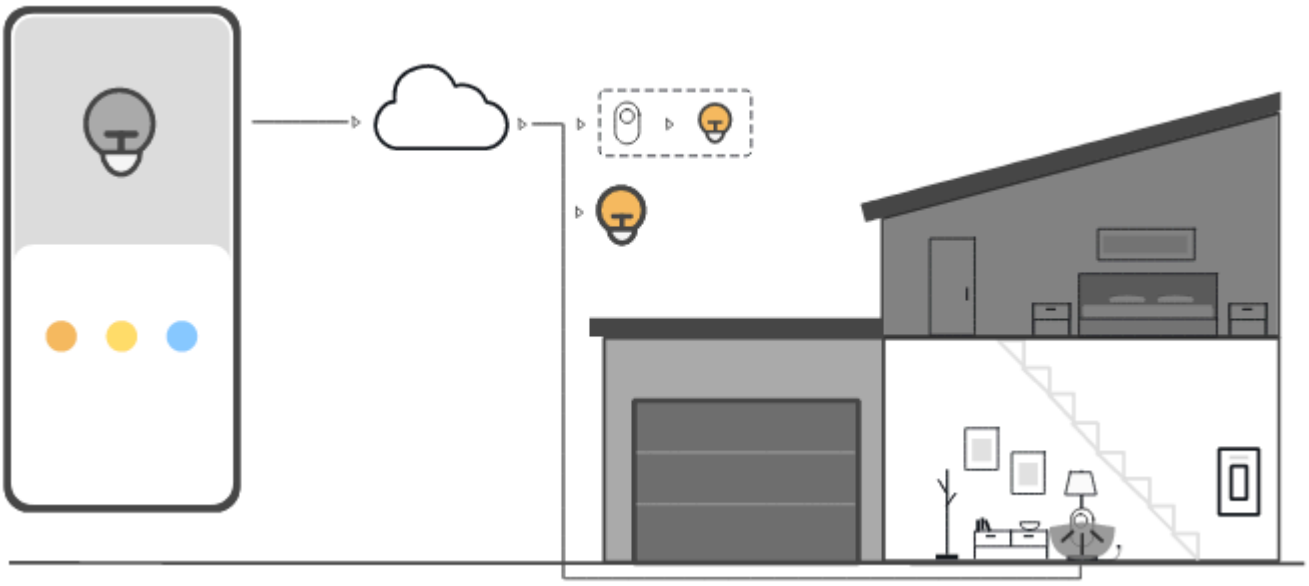
此步骤中的动画显示了 Device Shadow 服务如何 AWS IoT Core 保存控制设备和智能灯的设备状态信息。当智能灯处于离线状态时，设备影子将保存控制设备中的命令。

当智能灯重新连接到时 AWS IoT Core，它会检索这些命令。当控制设备处于离线状态时，设备影子将保存来自智能灯的状态信息。当控制设备重新连接时，它会检索智能灯的当前状态以更新其显示。

有关设备影子的更多信息，请参阅 AWS IoT Device Shadow 服务。

将设备数据路由到服务

了解如何 AWS IoT Core 将设备状态发送到其他 AWS 服务。



此步骤中的动画显示了如何使用 AWS IoT 规则 AWS IoT Core 将数据从设备发送到其他 AWS 服务。AWS IoT 规则订阅来自设备的特定消息，解释这些消息中的数据，并将解释后的数据路由到其他服务。在此示例中， AWS IoT 规则解释来自运动传感器的数据，并将命令发送到 Device Shadow，然后设备影子将命令发送到智能灯泡。与前面的示例一样，设备影子存储控制设备的设备状态信息。

有关 AWS IoT 规则的更多信息，请参阅的规则 AWS IoT。

试试 AWS IoT 快速连接

在本教程中，您将创建您的第一件事物对象，将设备连接到该对象，并观看其发送 MQTT 消息。

您预计需要花费 15-20 分钟来完成本教程。

本教程最适合想要快速入门 AWS IoT 以了解其在有限场景中的工作原理的人。如果您正在寻找一个能够帮助您入门以便您能够探索更多特征和服务的示例，请尝试 在动手教程中探索 AWS IoT Core 服务。

在本教程中，你将在连接至物联网资源的设备上下载并运行软件 AWS IoT Core ，这是非常小的物联网解决方案的一部分。该设备可以是 IoT 设备，如 Raspberry Pi，也可以是运行 Linux、OS 和 OSX 或 Windows 的电脑。如果您要将远程广域网 (W LoRa AN) 设备连接到 AWS IoT，请参阅教程 >将设备和网关连接到 LoRa WAN。 AWS IoT Core

如果您的设备支持可以运行 AWS IoT 控制台的浏览器，我们建议您在该设备上完成本教程。

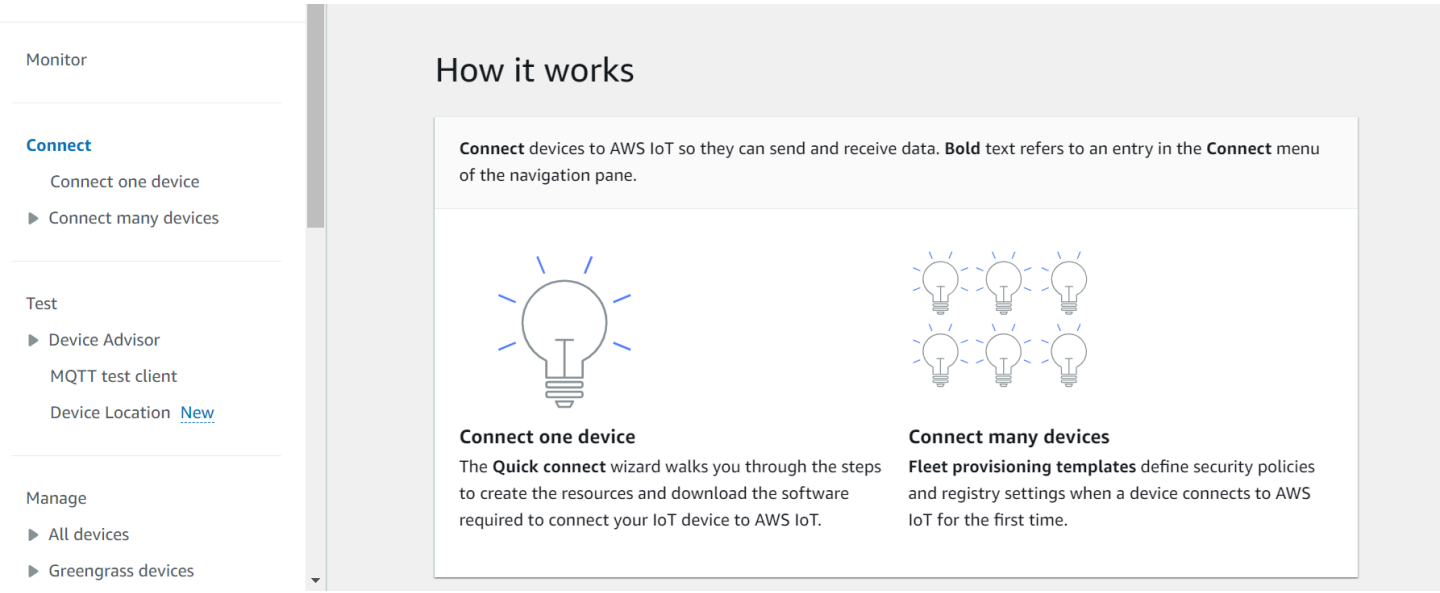
> **Note**
>
> 如果您的设备没有兼容的浏览器，请在电脑上遵循本教程。当流程要求您下载文件时，请将其下载到您的电脑，然后使用 Secure Copy (SCP) 或类似流程将下载的文件传输到您的设备。

本教程要求您的 IoT 设备与 AWS 账户的设备数据终端节点上的端口 8443 进行通信。要测试它是否可以访问该端口，请尝试测试与设备数据终端节点的连接中的步骤。

第 1 步。开始教程

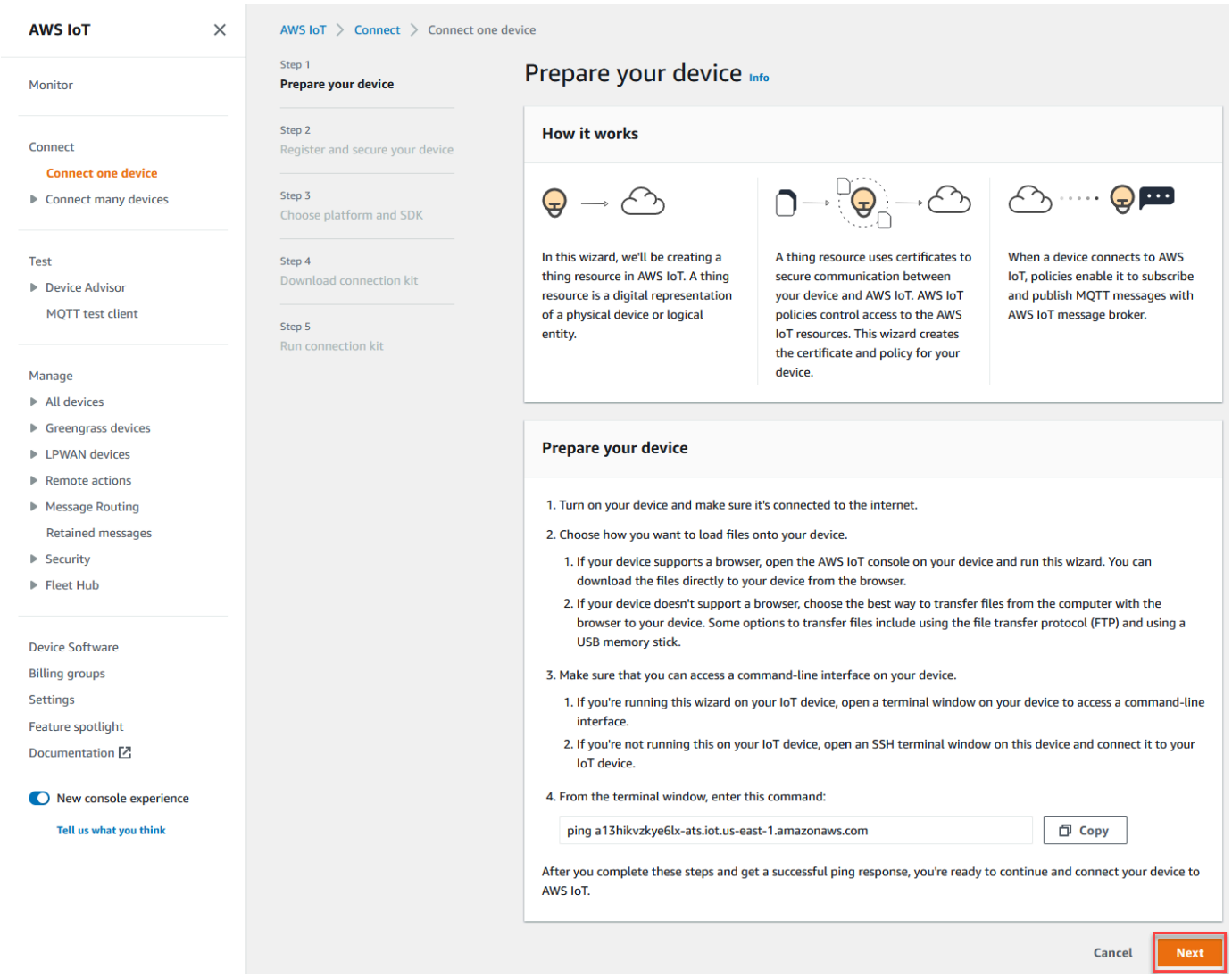
如果可能，请在您的设备上完成此流程；否则，请在本流程随后的步骤中准备好将文件传输到您的设备。

要开始本教程，请登录 AWS IoT 控制台。在 AWS IoT 主机主页的左侧，选择 Connect，然后选择 Con nect 一台设备。



第 2 步。创建一个事物对象

1. 在 Prepare your device（准备好您的设备）部分中，按照屏幕上的说明准备好要连接到 AWS IoT 的设备。

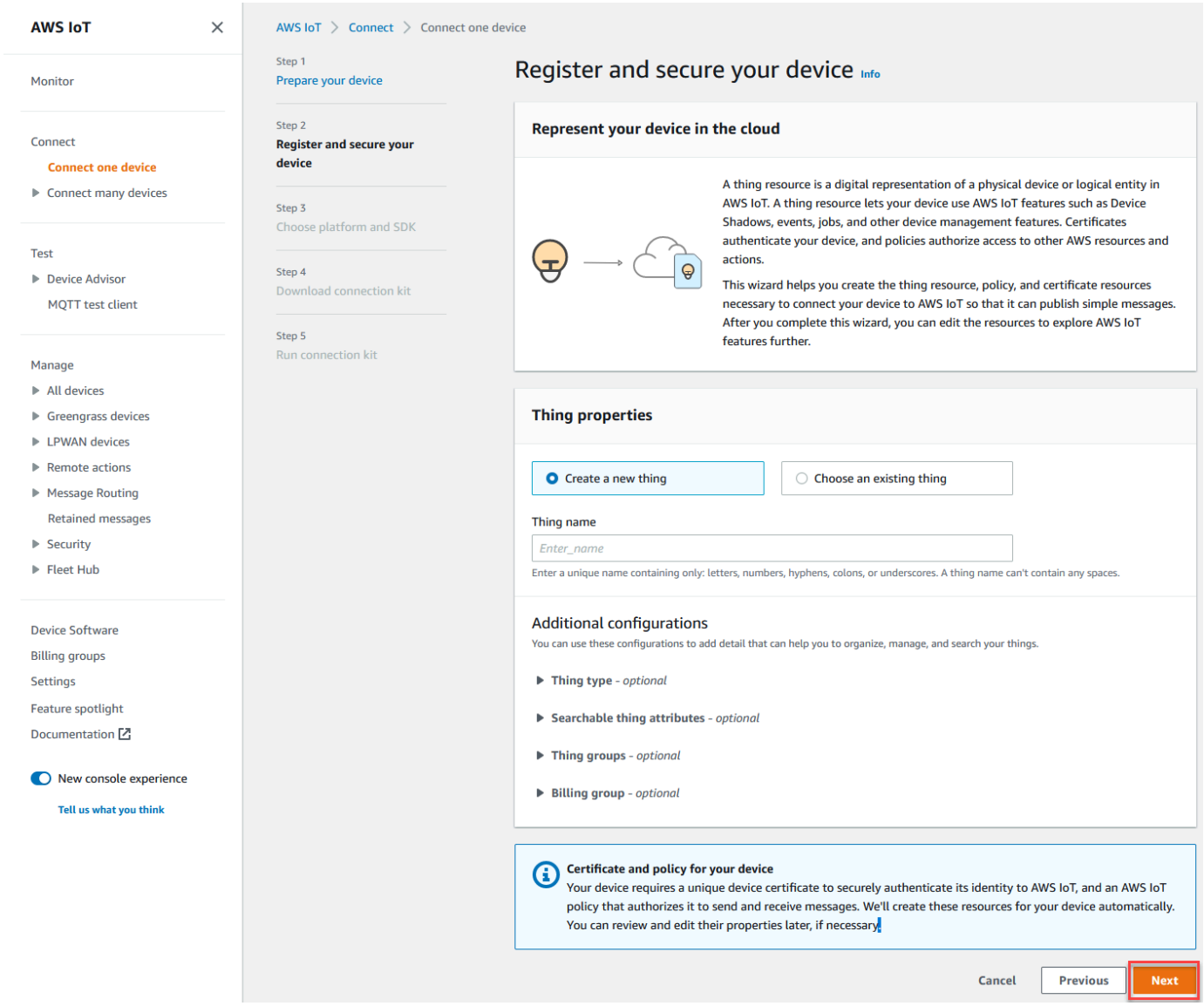


2. 在 Register and secure your device（注册并保护您的设备）部分中，选择 Create a new thing（创建新事物）或 Choose an existing thing（选择现有事物）。在 Thing name（事物名称）字段中，输入事物对象的名称。本例中使用的事物名称为 **`TutorialTestThing`**

> **Important**
>
> 在继续之前，请仔细检查您的事物名称。
> 事物对象创建后便无法更改事物名称。如果要更改事物名称，您必须使用正确的事物名称创建新事物对象，然后删除名称不正确的事物。

在 Additional configurations（其他配置）部分中，使用列出的可选配置进一步自定义您的事物资源。

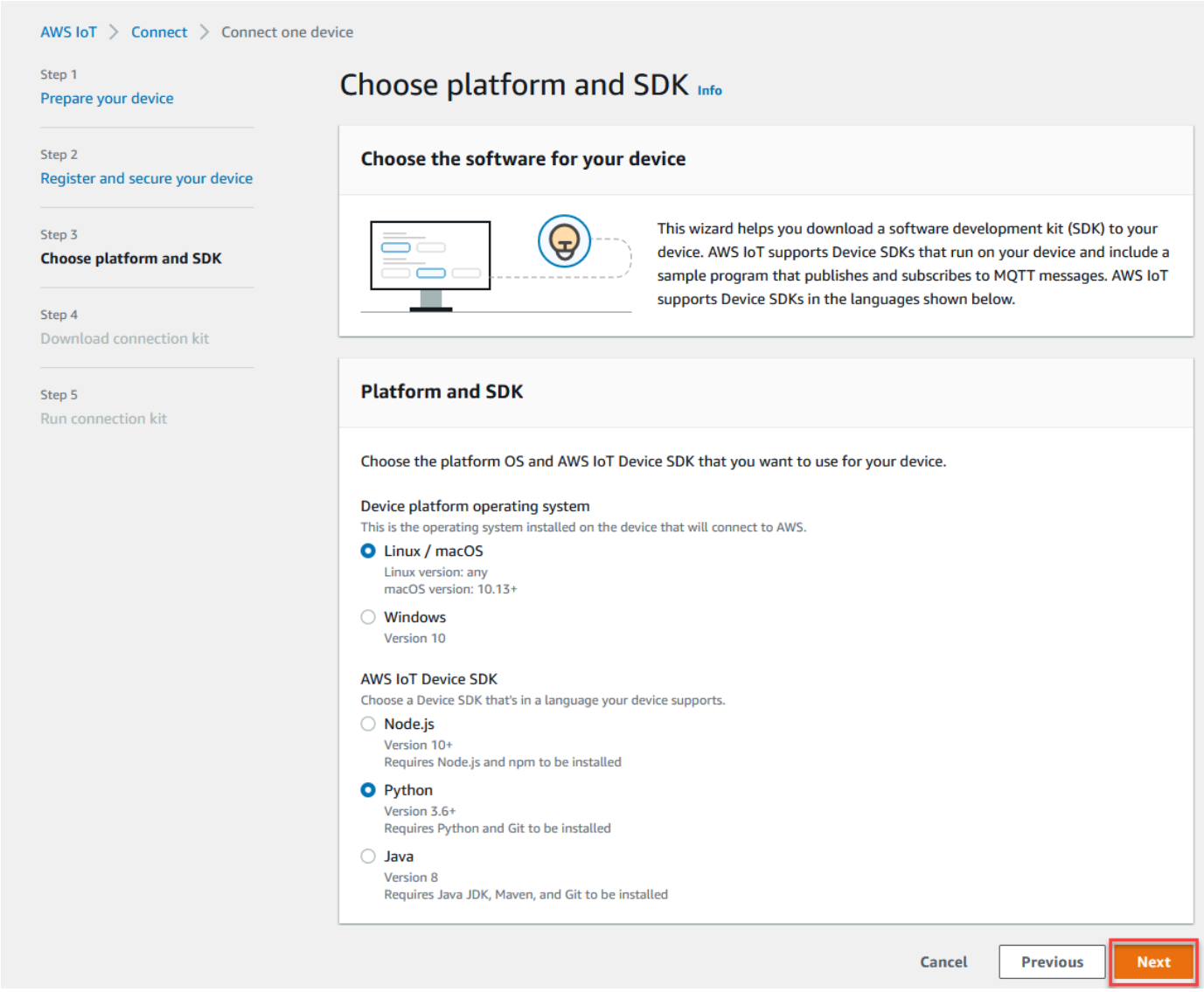
为事物对象提供名称并选择任何其他配置后，选择 Next（下一步）。



3. 在 “选择平台和 SDK” 部分，选择要使用的平台和 AWS IoT 设备 SDK 的语言。本示例使用 Linux/OSX 平台和 Python SDK。请确保在目标设备上安装了 python3 和 pip3，再继续下一步。

> **Note**
>
> 请务必在控制台页面底部检查所选 SDK 所需的必备软件列表。
> 您必须先在目标电脑上安装所需的软件，然后才能继续执行下一步。

选择好平台和 Device SDK 语言后，选择 Next（下一步）。

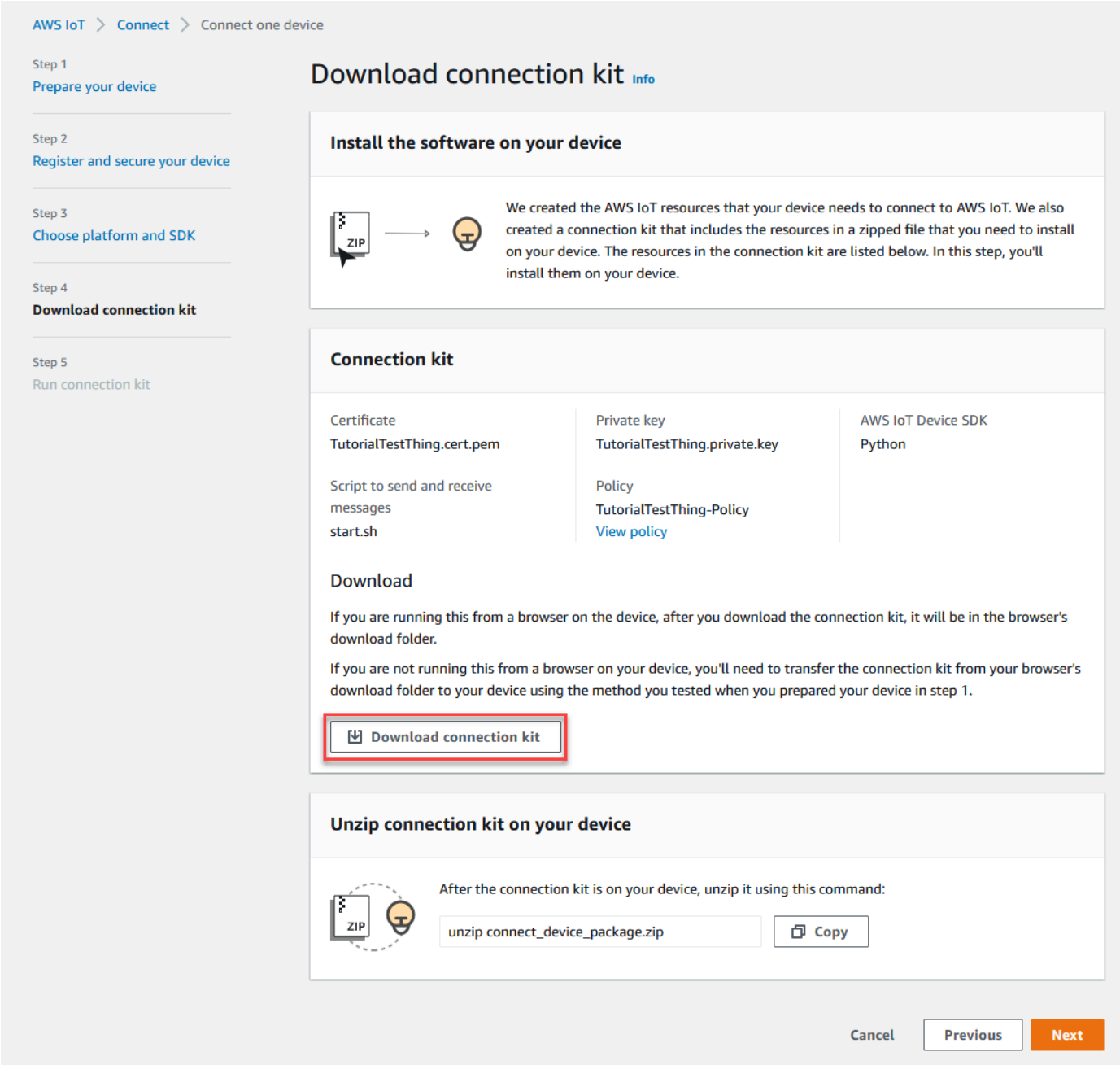


第 3 步。将文件下载到您的设备

该页面在创建连接套件后 AWS IoT 出现，其中包括您的设备所需的以下文件和资源：

- 用于对设备进行身份验证的事物证书文件
- 一个策略资源，用于授权事物对象与 AWS IoT交互
- 下载 AWS 设备 SDK 并在您的设备上运行示例程序的脚本

1. 准备好继续后，请选择下载连接套件按钮为您之前选择的平台下载连接工具包。



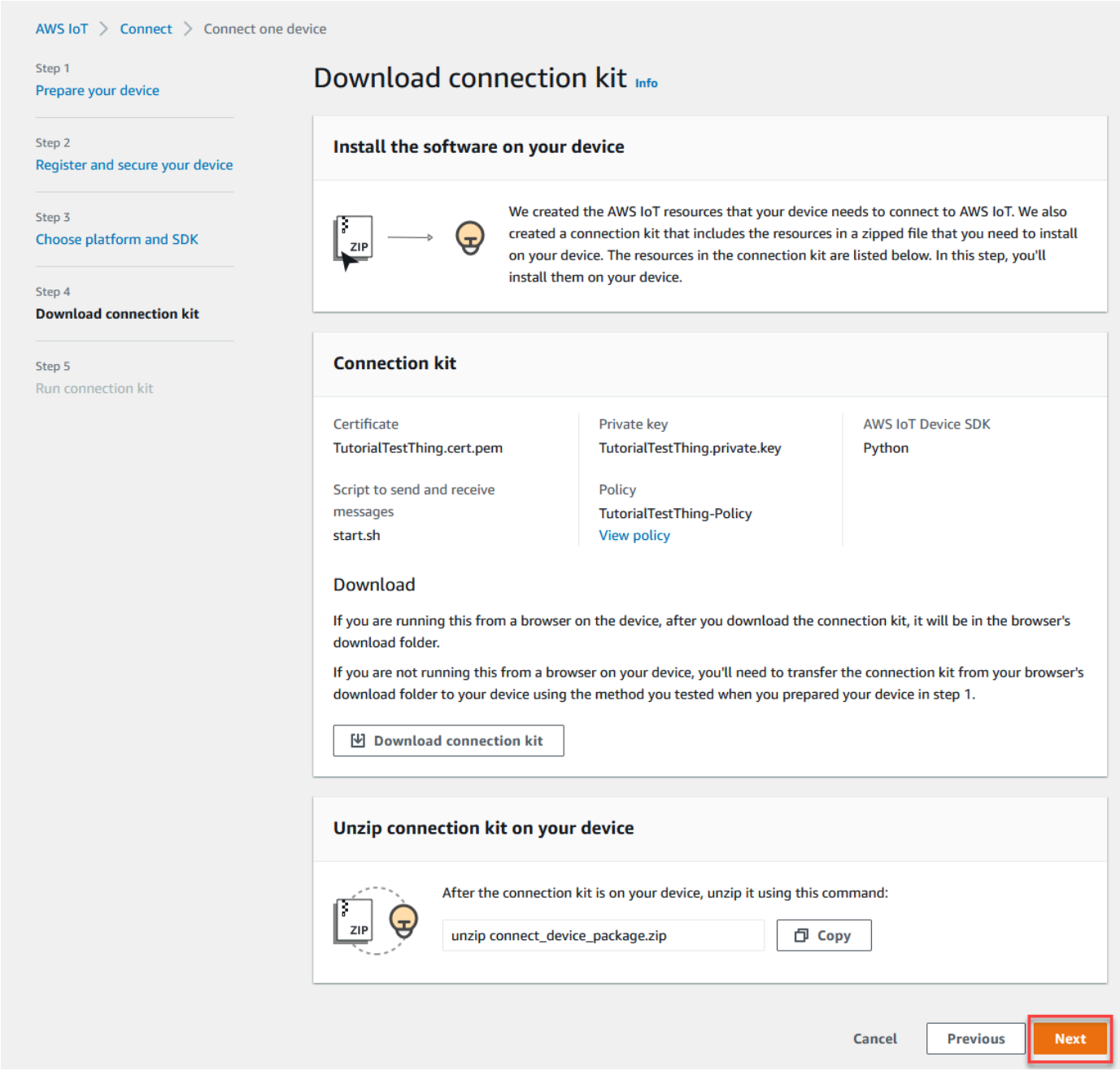
2. 如果您在设备上运行此流程，请将连接工具包文件保存到可运行命令行命令的目录中。

   如果您未在设备上运行此流程，请将连接工具包文件保存到本地目录中，然后将该文件传输到您的设备。

3. 在 Unzip connection kit on your device（在设备上解压缩连接工具包）部分中，在连接工具包文件所在的目录中输入 unzip connect_device_package.zip。

   如果您使用的是 Windows PowerShell 命令窗口，但该unzip命令不起作用，请unzip替换为expand-archive，然后重试命令行。

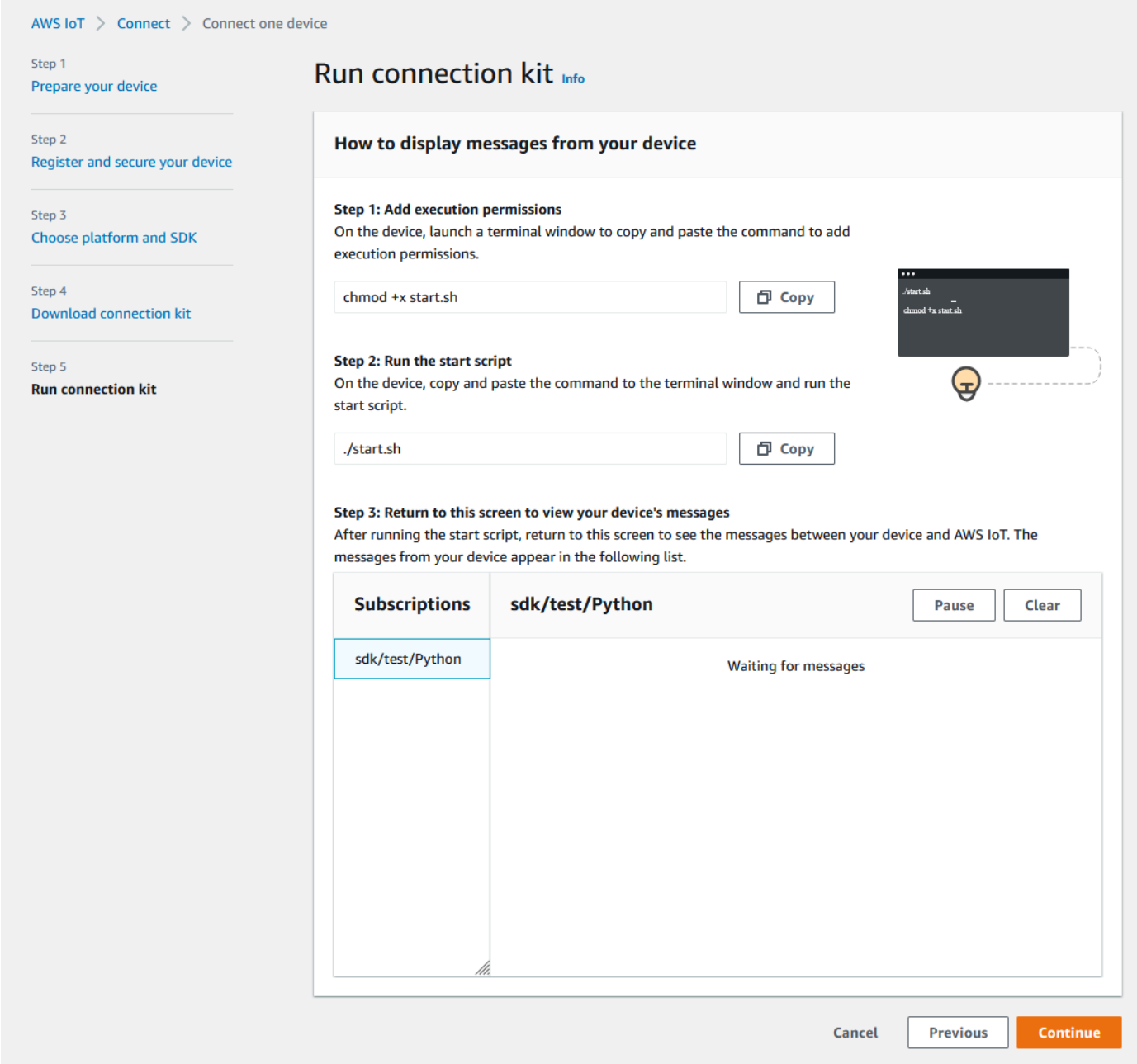
4. 当设备上具有连接工具包文件后，请选择 Next（下一步）继续本教程。



第 4 步。运行示例

您可以在设备上的终端或命令窗口中执行此程序，同时按照控制台中显示的说明进行操作。控制台中显示的命令适用于您在 the section called “第 2 步。创建一个事物对象” 中选择的操作系统。此处显示的内容适用于 Linux/OSX 操作系统。

1. 在设备上的终端或命令窗口中，在包含连接套件文件的目录中，执行 AWS IoT 控制台中显示的步骤。



2. 在控制台中输入 Step 2（步骤 2）的命令后，您应该会在设备的终端或命令窗口中看到类似以下内容的输出。此输出来自程序发出然后由 AWS IoT Core接收的消息。

```
Running pub/sub sample application...
Connecting to a13hikvzkye6lx-ats.iot.us-east-1.amazonaws.com with client ID 'basicPubSub'...
Connected!
Subscribing to topic 'sdk/test/Python'...
Subscribed with QoS.AT_LEAST_ONCE
Sending messages until program killed
Publishing message to topic 'sdk/test/Python': Hello World! [1]
Received message from topic 'sdk/test/Python': b'"Hello World! [1]"'
Publishing message to topic 'sdk/test/Python': Hello World! [2]
Received message from topic 'sdk/test/Python': b'"Hello World! [2]"'
Publishing message to topic 'sdk/test/Python': Hello World! [3]
Received message from topic 'sdk/test/Python': b'"Hello World! [3]"'
```

当示例程序运行时，也将显示测试消息 `Hello World!`。测试消息便会显示在设备的终端或命令窗口中。

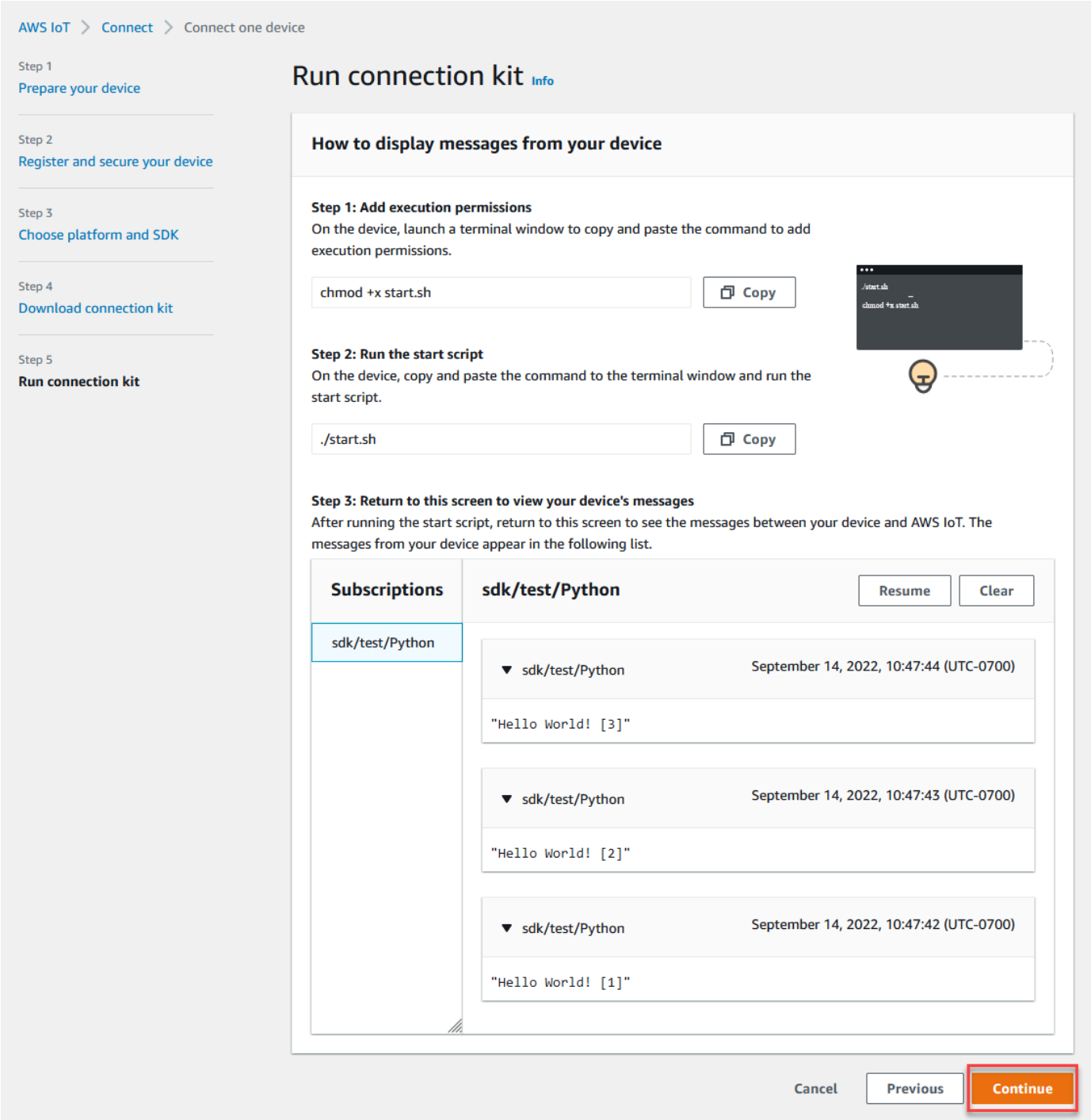
> **Note**
>
> 有关主题订阅和发布的更多信息，请参阅所选 SDK 的示例代码。

3. 要再次运行示例程序，您可以在控制台中重复本过程的 Step 2（步骤 2）中的命令。
4. （可选）如果您想在控制台中查看来自物联网客户端的消息，请在AWS IoT 控制台的测试页面上打开 MQTT 测试客户端。 AWS IoT 如果选择 Python SDK，请在 MQTT test client（MQTT 测试客户端）的 Topic filter（主题筛选条件）中输入主题（例如 **sdk/test/*python***），即可订阅来自设备的消息。主题筛选条件区分大小写，并且依赖于您在 Step 1（步骤 1）中选择的 SDK 的编程语言。有关主题订阅和发布的更多信息，请参阅所选 SDK 的代码示例。
5. 订阅测试主题后，请在设备上运行 ./start.sh。有关更多信息，请参阅 the section called “使用 MQTT 客户端查看 AWS IoT MQTT 消息”。

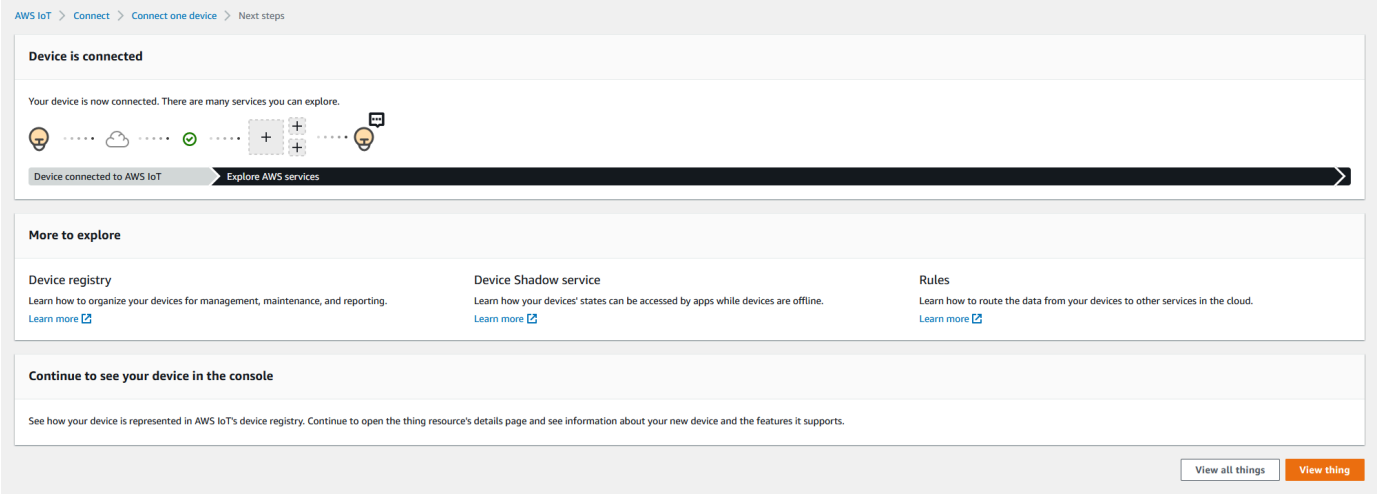
   运行 ./start.sh 后，MQTT 客户端会显示类似以下内容的消息：

   ```
   {
     "message": "Hello World!" [1]
   }
   ```

   每次接收新的 `Hello World!` 消息时，`[]` 中包含的 `sequence` 数量会递增 1，并在您结束程序时停止。
6. 要完成本教程并查看摘要，请在 AWS IoT 控制台中选择 “继续”。



7. 现在将显示您的 AWS IoT 快速连接教程摘要。



第 5 步。进一步探索

完成快速入门后，有一些想法需要 AWS IoT 进一步探索。

- 在 MQTT 测试客户端中查看 MQTT 消息

  从 AWS IoT 控制台，您可以在 AWS IoT 控制台的 Test（测试）页面上打开 MQTT 客户端。在 MQTT test client（MQTT 测试客户端）中，订阅 #，然后按前一步所述在设备上运行程序 ./start.sh。有关更多信息，请参阅 the section called “使用 MQTT 客户端查看 AWS IoT MQTT 消息”。

- 使用 Device Advisor 在您的设备上运行测试

  使用 Device Advisor 来测试您的设备能否安全可靠地连接和与之交互 AWS IoT。

- the section called “试试 AWS IoT Core 互动教程”

  要开始交互式教程，请在 AWS IoT 控制台的 “学习” 页面的 “查看 AWS IoT 工作原理” 磁贴中，选择 “开始教程”。

- 准备好了解更多教程

  此快速入门仅为您提供了以下示例 AWS IoT. 如果您想 AWS IoT 进一步探索并了解使其成为强大物联网解决方案平台的功能，请通过以下方式开始准备您的开发平台在动手教程中探索 AWS IoT Core 服务。

测试与设备数据终端节点的连接

本主题介绍如何测试设备与您账户的设备数据终端节点的连接，该终端节点是 IoT 设备用于连接 AWS IoT的终端节点。

在要测试的设备上执行这些步骤，或者使用连接到要测试的设备的 SSH 终端会话来执行这些步骤。

测试设备与设备数据终端节点的连接。

- 查找设备数据终端节点
- 快速测试连接
- 让应用程序测试与设备数据终端节点和端口的连接
- 测试与设备数据终端节点和端口的连接

查找设备数据终端节点

查找设备数据终端节点

1. 在 AWS IoT 控制台中，在导航窗格底部附近，选择 Settings（设置）。
2. 在 Settings（设置）页面的 Device data endpoint（设备数据终端节点）容器中，找到 Endpoint（终端节点）值并复制它。您的终端节点值对您来说是唯一的， AWS 账户 与以下示例类似:`a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com`.
3. 保存设备数据终端节点，以便在以下步骤中使用。

快速测试连接

此步骤测试与设备数据终端节点的常规连接，但不测试设备将使用的特定端口。此测试使用通用程序，通常足以了解您的设备是否可以连接到 AWS IoT。

如果要测试与设备将使用的特定端口的连接，请跳过此步骤并继续 让应用程序测试与设备数据终端节点和端口的连接。

快速测试设备数据终端节点

1. 在设备的终端或命令行窗口中，将示例设备数据终端节点 (*a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com*) 替换为您账户的设备数据终端节点，然后输入此命令。

Linux

```
ping -c 5 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
```

Windows

```
ping -n 5 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
```

2. 如果 ping 显示类似于以下内容的输出，则表示它已成功连接到设备数据终端节点。虽然它没有 AWS IoT 直接与之通信，但它确实找到了服务器，而且 AWS IoT 很可能可以通过此端点访问服务器。

```
PING a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com (xx.xx.xxx.xxx) 56(84) bytes of
 data.
64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
 icmp_seq=1 ttl=231 time=127 ms
64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
 icmp_seq=2 ttl=231 time=127 ms
64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
 icmp_seq=3 ttl=231 time=127 ms
64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
 icmp_seq=4 ttl=231 time=127 ms
64 bytes from ec2-EXAMPLE-218.eu-west-1.compute.amazonaws.com (xx.xx.xxx.xxx):
 icmp_seq=5 ttl=231 time=127 ms
```

如果您对此结果满意，则可以在此处停止测试。

如果想要测试与 AWS IoT使用的特定端口的连接，请继续 让应用程序测试与设备数据终端节点和端口的连接。

3. 如果 ping 没有返回成功输出，请检查终端节点值以确保您拥有正确的终端节点，然后检查设备与互联网的连接。

让应用程序测试与设备数据终端节点和端口的连接

可以通过使用 nmap 来执行更彻底的连接测试。此步骤测试 nmap 是否已安装在设备上。

检查设备上的 nmap

1. 在要测试的设备上的终端或命令行窗口中，输入此命令以查看是否已安装 nmap。

```
nmap --version
```

2. 如果您看到类似于以下内容的输出，则表示已安装 nmap，您可以继续 the section called “测试与设备数据终端节点和端口的连接”。

```
Nmap version 6.40 ( http://nmap.org )
Platform: x86_64-koji-linux-gnu
Compiled with: nmap-liblua-5.2.2 openssl-1.0.2k libpcre-8.32 libpcap-1.5.3 nmap-
libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
```

3. 如果您没有看到类似于上一步中所示的响应，则必须在设备上安装 nmap。为设备的操作系统选择相关步骤。

Linux

此步骤要求您具有在计算机上安装软件的权限。

在 Linux 计算机上安装 nmap

1. 在设备的终端或命令行窗口中，输入与所运行的 Linux 版本对应的命令。

   a. Debian 或 Ubuntu：

   ```
   sudo apt install nmap
   ```

   b. CentOS 或 RHEL：

   ```
   sudo yum install nmap
   ```

2. 使用以下命令测试安装：

   ```
   nmap --version
   ```

3. 如果您看到类似于以下内容的输出，则表示已安装 nmap，您可以继续 the section called “测试与设备数据终端节点和端口的连接”。

   ```
   Nmap version 6.40 ( http://nmap.org )
   Platform: x86_64-koji-linux-gnu
   ```

```
Compiled with: nmap-liblua-5.2.2 openssl-1.0.2k libpcre-8.32 libpcap-1.5.3 nmap-
libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
```

macOS

此步骤要求您具有在计算机上安装软件的权限。

在 macOS 计算机上安装 nmap

1. 在浏览器中打开 https://nmap.org/download#macosx，然后下载最新稳定的安装程序。

   出现提示时，选择 “打开方式” DiskImageInstaller。
2. 在安装窗口中，将程序包移到 Applications（应用程序）文件夹。
3. 在 Finder（查找工具）中，找到 Applications（应用程序）文件夹中的 nmap-xxxx-mpkg 程序包，Ctrl-click 该程序包，然后选择 Open（打开）以打开该程序包。
4. 查看安全对话框。如果您已准备好安装 nmap，请选择 Open（打开）以安装 nmap。
5. 在 Terminal 中，使用以下命令测试安装。

   ```
   nmap --version
   ```
6. 如果您看到类似于以下内容的输出，则表示已安装 nmap，您可以继续 the section called “测试与设备数据终端节点和端口的连接”。

   ```
   Nmap version 7.92 ( https://nmap.org )
   Platform: x86_64-apple-darwin17.7.0
   Compiled with: nmap-liblua-5.3.5 openssl-1.1.1k nmap-libssh2-1.9.0 libz-1.2.11
    nmap-libpcre-7.6 nmap-libpcap-1.9.1 nmap-libdnet-1.12 ipv6 Compiled without:
   Available nsock engines: kqueue poll select
   ```

Windows

此步骤要求您具有在计算机上安装软件的权限。

在 Windows 计算机上安装 nmap

1. 在浏览器中打开 https://nmap.org/download#windows，然后下载安装程序的最新稳定版本。

出现提示时，请选择 Save file（保存文件）。下载文件后，从下载文件夹中打开它。

2. 在安装程序文件完成下载后，打开已下载的 nmap-xxxx-setup.exe 以安装应用程序。
3. 安装程序时接受默认设置。

   对于此测试，您不需要 Npcap 应用程序。如果您不想安装它，则可以取消选择该选项。
4. 在 Command 中，使用以下命令测试安装。

```
nmap --version
```

5. 如果您看到类似于以下内容的输出，则表示已安装 nmap，您可以继续 the section called “测试与设备数据终端节点和端口的连接”。

```
Nmap version 7.92 ( https://nmap.org )
Platform: i686-pc-windows-windows
Compiled with: nmap-liblua-5.3.5 openssl-1.1.1k nmap-libssh2-1.9.0 nmap-
libz-1.2.11 nmap-libpcre-7.6 Npcap-1.50 nmap-libdnet-1.12 ipv6
Compiled without:
Available nsock engines: iocp poll select
```

测试与设备数据终端节点和端口的连接

测试设备数据终端节点和端口

1. 在设备的终端或命令行窗口中，将示例设备数据终端节点 (*a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com*) 替换为您账户的设备数据终端节点，然后输入此命令。

```
nmap -p 8443 a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
```

2. 如果 nmap 显示类似于以下内容的输出，nmap 能够通过所选端口成功连接到您的设备数据终端节点。

```
Starting Nmap 7.92 ( https://nmap.org ) at 2022-02-18 16:23 Pacific Standard Time
Nmap scan report for a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com
 (xx.xxx.147.160)
Host is up (0.036s latency).
Other addresses for a3qEXAMPLEsffp-ats.iot.eu-west-1.amazonaws.com (not scanned):
 xx.xxx.134.144 xx.xxx.55.139 xx.xxx.110.235 xx.xxx.174.233 xx.xxx.74.65
 xx.xxx.122.179 xx.xxx.127.126
rDNS record for xx.xxx.147.160: ec2-EXAMPLE-160.eu-west-1.compute.amazonaws.com
```

```
PORT     STATE SERVICE
8443/tcp open  https-alt
MAC Address: 00:11:22:33:44:55 (Cimsys)

Nmap done: 1 IP address (1 host up) scanned in 0.91 seconds
```

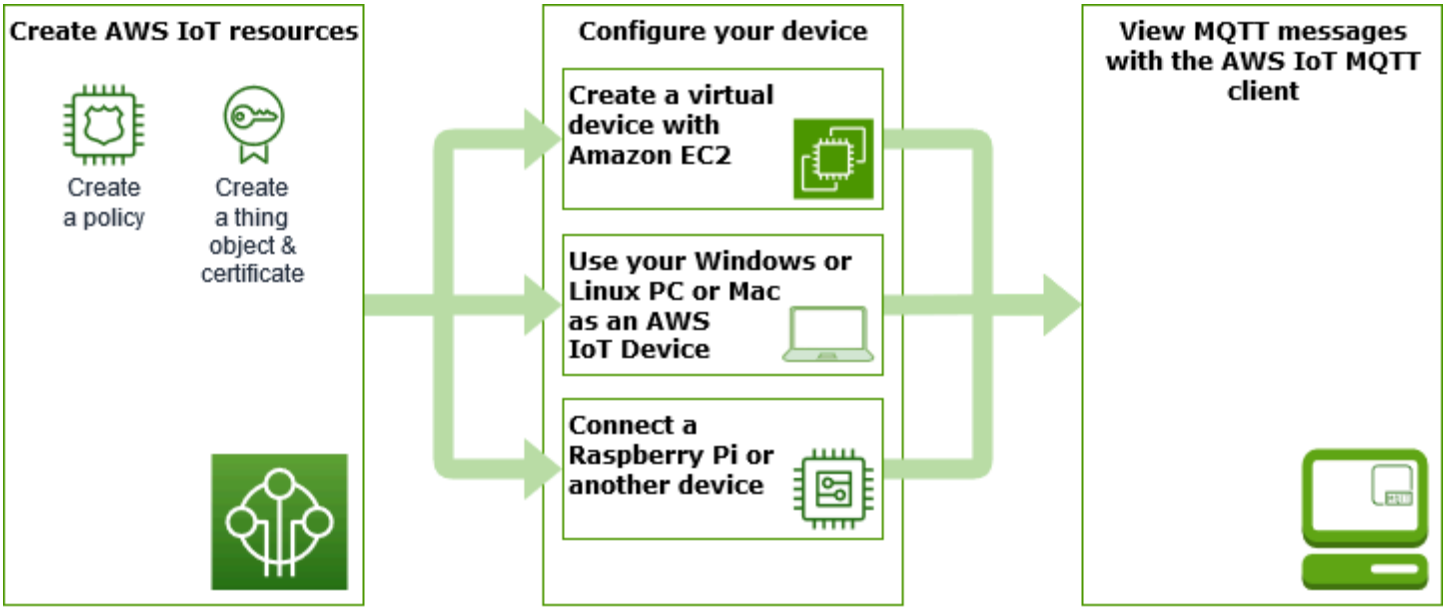
3. 如果 nmap 没有返回成功输出，请检查终端节点值以确保您拥有正确的终端节点，然后检查设备与互联网的连接。

通过将步骤 1 中使用的端口 **8443** 替换为要测试的端口，您可以测试设备数据终端节点上的其他端口，例如主 HTTPS 端口 443。

在动手教程中探索 AWS IoT Core 服务

在本教程中，您将安装软件并创建连接设备所需的 AWS IoT 资源， AWS IoT Core 以便设备可以发送和接收 MQTT 消息。 AWS IoT Core您将在 AWS IoT 控制台的 MQTT 客户端中看到这些消息。

您预计需要花费 20-30 分钟来完成本教程。如果您使用的是 IoT 设备或 Raspberry Pi，则您可能需要花费更长时间来完成本教程，因为您可能需要安装操作系统并配置设备。



本教程最适合想要入门的开发者， AWS IoT Core 这样他们就可以继续探索更高级的功能，例如规则引擎和阴影。本教程通过比快速入门教程更详细地解释这些步骤，让你为继续学习 AWS IoT Core 以及它如何与其他 AWS 服务交互做好准备。如果您只是想要快速、类似于 Hello World 这样的简单教程，请尝试 试试 AWS IoT 快速连接。

设置好您的 AWS 账户 和 AWS IoT 主机后，您将按照以下步骤查看如何连接设备并让设备向其发送消息 AWS IoT Core。

后续步骤

- 选择最适合您的设备选项
- 如果您不打算使用 Amazon EC2 创建虚拟设备，请the section called “创建 AWS IoT 资源”。
- the section called “配置您的设备”
- the section called “使用 MQTT 客户端查看 AWS IoT MQTT 消息”

有关的更多信息 AWS IoT Core，请参阅什么是 AWS IoT Core？

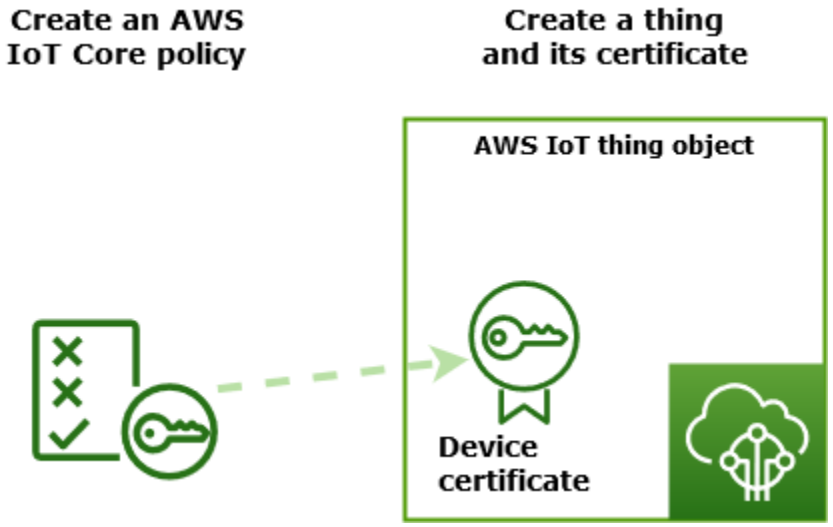
哪种设备选项最适合您？

如果您不确定要选择哪个选项，请参照下表中列出的每个选项的优缺点来帮助您确定最适合的选项。

选项	这可能是一个不错的选择，如果：	这可能不是一个好的选择，如果：
the section called “使用 Amazon EC2 创建虚拟设备”	• 您没有自己的设备可供测试。 • 您不想在自己的系统上安装任何软件。 • 您希望在 Linux OS 上进行测试。	• 您不习惯使用命令行命令。 • 您不想产生任何额外的 AWS 费用。 • 您不想在 Linux OS 上进行测试。
the section called “使用你的 Windows、Linux 电脑或 Mac 作为 AWS IoT 设备”	• 您不想产生任何额外的 AWS 费用。 • 您不想配置任何其它设备。	• 您不想在个人电脑上安装任何软件。 • 您需要一个更具代表性的测试平台。
the section called “连接 Raspberry Pi 或其他设备”	• 你想 AWS IoT 用实际设备进行测试。 • 您已经拥有要测试的设备。 • 您已有将硬件集成到系统中的经验。	• 您不想为了试用而购买或配置设备。 • 现在，你想 AWS IoT 尽可能简单地进行测试。

创建 AWS IoT 资源

在本教程中，您将创建设备连接 AWS IoT Core 和交换消息所需的 AWS IoT 资源。



1. 创建一份 AWS IoT 政策文档，该文件将授权您的设备与 AWS IoT 服务进行交互。
2. 在中创建事物对象 AWS IoT 及其 X.509 设备证书，然后附加策略文档。事物对象是您的设备在 AWS IoT 注册表中的虚拟表示形式。证书对您的设备进行身份验证 AWS IoT Core，策略文档授权您的设备与之交互。 AWS IoT

> **Note**
>
> 如果您计划 the section called “使用 Amazon EC2 创建虚拟设备”，则可以跳过此页面并继续前往 the section called “配置您的设备”。当您创建虚拟事物时，将会创建这些资源。

本教程使用 AWS IoT 控制台创建 AWS IoT 资源。如果您的设备支持 Web 浏览器，则在设备的 Web 浏览器上运行此流程可能会更容易，因为您可以将证书文件直接下载到您的设备上。如果您在另一台电脑上运行此流程，则需要先将证书文件复制到您的设备，然后才能为示例应用程序所用。

创建 AWS IoT 策略

设备使用 X.509 证书进行身份验证。 AWS IoT Core证书附有 AWS IoT 政策。这些策略决定了允许设备执行哪些 AWS IoT 操作，例如订阅或发布 MQTT 主题。您的设备在连接并向其发送消息时会出示其证书 AWS IoT Core。

按照步骤创建一个策略，以允许您的设备执行运行示例程序所需的 AWS IoT 操作。您必须先创建 AWS IoT 策略，然后才能将其附加到稍后创建的设备证书。

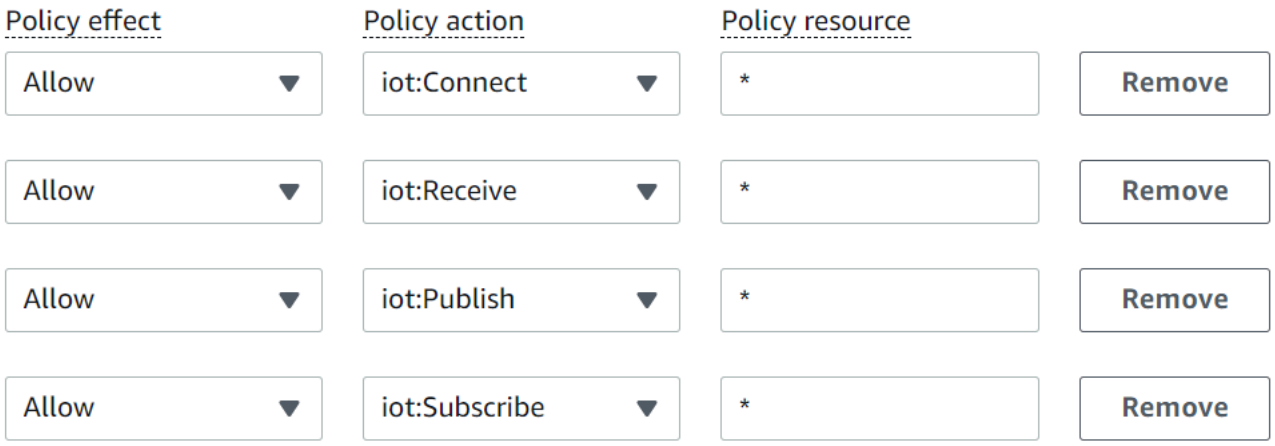
创建 AWS IoT 策略

1. 在 AWS IoT 控制台的左侧菜单中，选择安全，然后选择策略。
2. 在 You don't have a policy yet（您还没有策略）页面上，选择 Create policy（创建策略）。

   如果您的账户有现有策略，请选择创建策略。
3. 在 Create policy（创建策略）页面上：

   1. 在 Policy properties（策略属性）部分，在 Policy name（策略名称）字段中，输入策略的名称（例如，**My_Iot_Policy**）。请勿在策略名称中使用个人身份信息。
   2. 在 Policy document（策略文档）部分，创建用于授予或拒绝资源访问 AWS IoT Core 操作的策略语句。要创建允许所有客户端执行 **iot:Connect** 的策略语句，请按照以下步骤操作：
      - 在 Policy effect（策略效果）字段中，选择 Allow（允许）。这允许将此策略附加到其证书的所有客户端执行 Policy action（策略操作）字段中列出的操作。
      - 在 Policy action（策略操作）字段中，选择策略操作，例如 **iot:Connect**。策略操作是指设备在从设备 SDK 运行示例程序时需要权限才能执行的操作。
      - 在 Policy resource（策略资源）字段中，输入资源 Amazon Resource Name (ARN) 或 ***。用于选择任何客户端（设备）的 ***。

      要为 **iot:Receive**、**iot:Publish** 和 **iot:Subscribe** 创建策略语句，请选择 Add new statement（添加新语句）并重复相关步骤。



> **Note**
>
> 在此快速入门教程中，为了简单起见，将使用通配符 (*)。为了提高安全性，您应该指定客户端 ARN 而不是通配符作为资源，从而限定哪些客户端（设备）可以连接和发

布消息。客户端 ARN 应采用以下格式：`arn:aws:iot:`*`your-region`*`:`*`your-aws-account`*`:client/`*`my-client-id`*。
但是，您必须先创建资源（如客户端设备或事物影子），然后才能将其 ARN 分配给策略。有关更多信息，请参阅 AWS IoT Core 操作资源。

4. 在输入策略的信息后，选择 Create（创建）。

有关更多信息，请参阅 如何 AWS IoT 与 IAM 配合使用。

创建一个事物对象

连接的设备 AWS IoT Core 由 AWS IoT 注册表中的事物对象表示。事物对象表示特定设备或逻辑实体。它可以是物理设备或传感器（例如，灯泡或墙壁上的开关）。它也可以是一个逻辑实体，例如应用程序或物理实体的实例 AWS IoT，它不连接但与其他连接的设备相关（例如，带有发动机传感器或控制面板的汽车）。

在 AWS IoT 控制台中创建事物

1. 在 AWS IoT 控制台的左侧菜单中，选择所有设备，然后选择事物。
2. 在 Things（事物）页面上，选择 Create things（创建事物）。
3. 在 Create things（创建事物）页面上，选择 Create a single thing（创建单个事物），然后选择 Next（下一步）。
4. 在 Specify thing properties（指定事物属性）页面，对于 Thing name（事物名称），输入事物的名称，如 **`MyIotThing`**。

   仔细选择事物名称，因为以后无法更改事物名称。

   要更改事物的名称，您必须创建一个新事物，为其指定新名称，然后删除旧事物。

   > **Note**
   > 请勿在事物名称中使用个人身份信息。事物名称可以出现在未加密的通信和报告中。

5. 保持此页面上其余字段为空。选择下一步。
6. 在 Configure device certificate - optional（配置证书 - 可选）页面上，选择 Auto-generate a new certificate (recommended)（自动生成新证书（推荐））。选择下一步。
7. 在 Attach policies to certificate - optional（将策略附加到证书 - 可选）页面，选择在上一部分中创建的策略。在该部分中，该策略被命名为 **`My_Iot_Policy`**。选择 Create thing（创建事物）。

8. 在 Download certificates and keys（下载证书和密钥）页面：

   1. 下载每个证书和密钥文件，并保存它们以备稍后使用。您需要在设备上安装这些文件。

      保存证书文件时，请在下表中提供它们的名称。这些是将后续示例中使用的文件名。

      证书文件名

文件	文件路径
私有密钥	`private.pem.key`
公有密钥	（在这些示例中未使用）
设备证书	`device.pem.crt`
根 CA 证书	`Amazon-root-CA-1.pem`

   2. 要为这些文件下载根 CA 文件，请选择根 CA 证书文件的 Download（下载）链接，该链接应与您使用的数据端点和加密套件类型相对应。在本教程中，请选择 RSA 2048 bit key: Amazon Root CA 1（RSA 2048 位密钥：Amazon Root CA 1）右侧的 Download（下载），然后下载 RSA 2048 bit key: Amazon Root CA 1（RSA 2048 位密钥：Amazon Root CA 1）证书文件。

      > **Important**
      >
      > 您必须在离开此页之前保存证书文件。在控制台中离开此页面后，您将无法再访问证书文件。
      > 如果忘记下载在此步骤中创建的证书文件，则必须退出此控制台屏幕，转到控制台中的事物列表，删除您创建的事物对象，然后从头重新开始此流程。

   3. 选择完成。

完成此流程后，您应该在事物列表中看到新的事物对象。

配置您的设备

本部分介绍如何配置设备以使其连接到 AWS IoT Core。如果你想开始使用 AWS IoT Core 但还没有设备，你可以使用 Amazon EC2 创建虚拟设备，也可以使用 Windows 电脑或 Mac 作为物联网设备。

选择最适合您尝试的设备选项 AWS IoT Core。当然，您可以尝试所有的内容，但一次只能尝试一项。如果您不确定哪个设备选项最适合您，请阅读有关如何选择哪个设备选项最适合的内容，然后返回此页。

设备选项

- 使用 Amazon EC2 创建虚拟设备
- 使用你的 Windows、Linux 电脑或 Mac 作为 AWS IoT 设备
- 连接 Raspberry Pi 或其他设备

使用 Amazon EC2 创建虚拟设备

在本教程中，您将创建一个 Amazon EC2 实例，用作云中的虚拟设备。

要完成本教程，你需要一个 AWS 账户。如果您没有账户，请完成 设置你的 AWS 账户 中介绍的步骤然后继续操作。

在本教程中，您将：

- 设置 Amazon EC2 实例
- 安装 Git、Node.js 并配置 AWS CLI
- 为您的虚拟设备创建 AWS IoT 资源
- 安装适用于的 AWS IoT 设备 SDK JavaScript
- 运行示例应用程序
- 在 AWS IoT 控制台中查看来自示例应用程序的消息

设置 Amazon EC2 实例

以下步骤介绍了如何创建一个 Amazon EC2 实例，以作为您的虚拟设备代替物理设备。

如果这是您首次创建 Amazon EC2 实例，您可能会发现开始使用 Amazon EC2 Linux 实例中的说明更有用。

启动实例

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 在左侧的控制台菜单中，展开 Instances（实例）部分并选择 Instances（实例）。在 Instances（实例）控制面板中，选择右侧的 Launch instances（启动实例）以显示基本配置列表。

3. 在 Name and tags（名称和标签）部分中，输入实例的名称并可选添加标签。
4. 在 Application and OS Images (Amazon Machine Image) [应用程序和操作系统映像（Amazon 机器映像）] 下，为您的实例选择一个 AMI 模板，例如 Amazon Linux 2 AMI (HVM)。请注意，此 AMI 标记为“Free tier eligible”(符合条件的免费套餐)。
5. 在 Instance type（实例类型）部分中，您可以选择实例的硬件配置。选择 `t2.micro` 类型 (默认情况下的选择)。请注意，此实例类型适用免费套餐。
6. 在 Key pair (login) [密钥对（登录）] 部分中，从下拉列表中选择密钥对名称，或选择 Create a new key pair（创建新的密钥对）以创建新的密钥对。创建新的密钥对时，确保您下载私钥文件并将其保存在安全的地方，因为这是您下载和保存它的唯一机会。当您启动实例时，您将需要提供密钥对的名称；当您每次连接到实例时，您将需要提供相应的私有密钥。

   > **Warning**
   >
   > 请勿选择 Proceed without a key pair（在没有密钥对的情况下继续）选项。如果您启动的实例没有密钥对，就不能连接到该实例。

7. 在 Network settings（网络设置）部分和 Configure storage（配置存储）部分中，您可以保留默认设置。准备就绪后，选择 Launch instances（启动实例）。
8. 确认页面会让您知道自己的实例已启动。选择 View Instances 以关闭确认页面并返回控制台。
9. 在实例屏幕上，您可以查看启动状态。启动实例只需很短的时间。启动实例时，其初始状态为 `pending`。实例启动后，其状态变为 `running`，并且会收到一个公有 DNS 名称。(如果 Public DNS (IPv4) 列已隐藏，请选择页面右上角的 Show/Hide Columns (齿轮状图标)，然后选择 Public DNS (IPv4)。)
10. 需要几分钟准备好实例，以便您能连接到实例。检查您的实例是否通过了状态检查；您可以在 Status Checks 列中查看此信息。

    在您的新实例通过其状态检查后，继续执行下一个流程并连接到此实例。

连接到您的实例

您可以通过从 Amazon EC2 控制台中选择实例，然后选择使用 Amazon EC2 Instance Connect 进行连接，以使用基于浏览器的客户端连接到实例。Instance Connect 处理权限并提供成功的连接。

1. 通过以下网址打开 Amazon EC2 控制台：https://console.aws.amazon.com/ec2/。
2. 在左侧菜单上，选择 Instances（实例）。
3. 选择实例，然后选择连接。

4. 依次选择 Amazon EC2 Instance Connect（Amazon EC2 实例连接）和 Connect（连接）。

您现在应该有一个 Amazon EC2 Instance Connect 窗口，该窗口将登录您的新 Amazon EC2 实例。

安装 Git、Node.js 并配置 AWS CLI

在本部分中，您将在 Linux 实例上安装 Git 和 Node.js。

要安装 Git

1. 在您的 Amazon EC2 Instance Connect 窗口中，使用以下命令更新实例。

```
sudo yum update -y
```

2. 在您的 Amazon EC2 Instance Connect 窗口中，使用以下命令安装 Git。

```
sudo yum install git -y
```

3. 要检查 Git 是否已安装以及 Git 的当前版本，请运行以下命令：

```
git --version
```

要安装 Node.js

1. 在您的 Amazon EC2 Instance Connect 窗口中，使用以下命令安装节点版本管理器 (nvm)。

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
```

由于 nvm 可以安装多个版本的 Node.js 并允许您在各个版本之间切换，我们将使用 nvm 安装 Node.js。

2. 在您的 Amazon EC2 Instance Connect 窗口中，使用此命令激活 nvm。

```
. ~/.nvm/nvm.sh
```

3. 在您的 Amazon EC2 Instance Connect 窗口中，使用以下命令借助 nvm 安装最新版本的 Node.js。

```
nvm install 16
```

> **Note**
>
> 这将安装最新 LTS 版本的 Node.js。

安装 Node.js 还会安装节点程序包管理器 (npm)，以便您根据需要安装其它模块。

4. 在您的 Amazon EC2 Instance Connect 窗口中，使用此命令测试 Node.js 已安装且运行正常。

```
node -e "console.log('Running Node.js ' + process.version)"
```

本教程需要 Node v10.0 或更高版本。有关更多信息，请参阅 Tutorial: Setting Up Node.js on an Amazon EC2 Instance（教程：设置 Amazon EC2 实例上的 Node.js）。

要配置 AWS CLI

您的 Amazon EC2 实例预加载了 AWS CLI。但是，您必须完成 AWS CLI 个人资料。有关如何配置 CLI 的更多信息，请参阅 配置 AWS CLI。

1. 以下示例显示了示例值。将它们替换为您自己的值：您可以在 AWS 控制台的 My Security Credentials（我的安全凭证）项下的账户信息中找到这些值。

   在您的 Amazon EC2 Instance Connect 窗口中，输入以下命令：

   ```
   aws configure
   ```

   然后在显示的提示处输入您账户中的值。

   ```
   AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-west-2
   Default output format [None]: json
   ```

2. 你可以用这个命令测试你的 AWS CLI 配置：

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

   如果您的配置 AWS CLI 正确，则该命令应从中返回终端节点地址 AWS 账户。

为您的虚拟设备创建 AWS IoT 资源

本节介绍如何使用直接在虚拟设备上创建事物对象及其证书文件。 AWS CLI 这是直接在设备上执行的，以避免将它们从另一台电脑复制到设备时可能出现的潜在复杂问题。在本节中，您将为虚拟设备创建以下资源：

- 用于代表您的虚拟设备的事物对象 AWS IoT。
- 用于验证虚拟设备身份的证书。
- 授权虚拟设备连接到 AWS IoT以及发布、接收和订阅消息的策略文档。

在 Linux 实例中创建 AWS IoT 事物对象

连接的设备 AWS IoT 由 AWS IoT 注册表中的事物对象表示。事物对象表示特定设备或逻辑实体。在这种情况下，事物对象代表您的虚拟设备，即 Amazon EC2 实例。

1. 在您的 Amazon EC2 Instance Connect 窗口中，运行以下命令以创建事物对象。

```
aws iot create-thing --thing-name "MyIotThing"
```

2. JSON 响应应该如下所示：

```
{
    "thingArn": "arn:aws:iot:your-region:your-aws-account:thing/MyIotThing",
    "thingName": "MyIotThing",
    "thingId": "6cf922a8-d8ea-4136-f3401EXAMPLE"
}
```

在您的 Linux 实例中创建和附加 AWS IoT 密钥和证书

create-keys-and-certificate 命令将创建由 Amazon Root 证书颁发机构签名的客户端证书。此证书用于验证虚拟设备的身份。

1. 在您的 Amazon EC2 Instance Connect 窗口中，创建用于存储证书和密钥文件的目录。

```
mkdir ~/certs
```

2. 在您的 Amazon EC2 Instance Connect 窗口中，使用此命令下载 Amazon 证书颁发机构 (CA) 证书的副本。

```
curl -o ~/certs/Amazon-root-CA-1.pem \
    https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

3. 在您的 Amazon EC2 Instance Connect 窗口中，运行以下命令以创建您的私有密钥、公有密钥和 X.509 证书文件。此命令还使用注册和激活证书。 AWS IoT

```
aws iot create-keys-and-certificate \
    --set-as-active \
    --certificate-pem-outfile "~/certs/device.pem.crt" \
    --public-key-outfile "~/certs/public.pem.key" \
    --private-key-outfile "~/certs/private.pem.key"
```

响应看起来与以下内容类似。保存 certificateArn，以便您可以在后续命令中使用。您需要使用它来将证书附加到您的事物上，并在稍后的步骤中将策略附加到证书上。

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificateId":
 "9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificatePem": "
-----BEGIN CERTIFICATE-----
MIICiTCCEXAMPLE6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgEXAMPLEAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC0lBTSEXAMPLE2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
BgkqhkiG9w0BCQEWEG5vb25lQGFtYEXAMPLEb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCEXAMPLEJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAEXAMPLEsTC0lBTSBDb25z
b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEXAMPLE25lQGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+aEXAMPLE
EXAMPLEfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
rDHudUZEXAMPLELG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb3OhjZnzcvQAEXAMPLEWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9qEXAMPLEyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDEXAMPLEBHjJnyp378OD8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC
 KEY-----\nMIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h
\nMMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/
```

```
gHr99VEEXAMPLE5VF13\n59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa
\nhJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\nFQIDAQAB\n-----
END PUBLIC KEY-----\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\nkey omitted for security
 reasons\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

4. 在您的 Amazon EC2 Instance Connect 窗口中，使用以下命令和之前命令响应中的 *certificateArn* 将您的事物对象附加到您刚刚创建的证书上。

```
aws iot attach-thing-principal \
    --thing-name "MyIotThing" \
    --principal "certificateArn"
```

如果成功，此命令将不会显示任何输出。

要创建并附加策略

1. 在您的 Amazon EC2 Instance Connect 窗口中，复制此策略文档并粘贴到名为 **~/policy.json** 的文件中，以创建策略文件。

如果您没有惯用的 Linux 编辑器，则可以使用此命令打开 nano。

```
nano ~/policy.json
```

将 policy.json 的策略文档粘贴到其中。输入 ctrl-x 退出 nano 编辑器并保存文件。

policy.json 策略文档的内容。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
```

```
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

2. 在您的 Amazon EC2 Instance Connect 窗口中，使用以下命令创建策略。

```
aws iot create-policy \
    --policy-name "MyIotThingPolicy" \
    --policy-document "file://~/policy.json"
```

输出：

```
{
    "policyName": "MyIotThingPolicy",
    "policyArn": "arn:aws:iot:your-region:your-aws-account:policy/
MyIotThingPolicy",
    "policyDocument": "{
        \"Version\": \"2012-10-17\",
        \"Statement\": [
            {
                \"Effect\": \"Allow\",
                \"Action\": [
                    \"iot:Publish\",
                    \"iot:Receive\",
                    \"iot:Subscribe\",
                    \"iot:Connect\"
                ],
                \"Resource\": [
                    \"*\"
                ]
            }
        ]
    }",
    "policyVersionId": "1"
}
```

3. 在您的 Amazon EC2 Instance Connect 窗口中，使用以下命令将策略附加到虚拟设备的证书。

```
aws iot attach-policy \
    --policy-name "MyIotThingPolicy" \
    --target "certificateArn"
```

如果成功，此命令将不会显示任何输出。

安装适用于的 AWS IoT 设备 SDK JavaScript

在本节中，您将安装适用于 Device SDK JavaScript，其中包含应用程序可以用来与之通信的代码 AWS IoT 和示例程序。 AWS IoT 有关更多信息，请参阅 JavaScript GitHub 存储库的AWS IoT 设备 SDK。

在您的 Linux 实例 JavaScript 上安装适用于的 AWS IoT 设备软件开发工具包

1. 在您的 Amazon EC2 Instance Connect 窗口中，使用此命令将 JavaScript 存储库的 AWS IoT 设备软件开发工具包克隆到您的主目录目录中。`aws-iot-device-sdk-js-v2`

   ```
   cd ~
   git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
   ```

2. 导航到您在上一步中创建的 `aws-iot-device-sdk-js-v2` 目录。

   ```
   cd aws-iot-device-sdk-js-v2
   ```

3. 使用 npm 安装 SDK。

   ```
   npm install
   ```

运行示例应用程序

下一部分中的命令假定您的密钥和证书文件均已存储在设备上，如下表中所示。

证书文件名

文件	文件路径
私有密钥	`~/certs/private.pem.key`
设备证书	`~/certs/device.pem.crt`

文件	文件路径
根 CA 证书	~/certs/Amazon-root-CA-1.pem

在本节中，您将安装并运行位于 AWS IoT 设备软件开发工具包aws-iot-device-sdk-js-v2/samples/node目录中的pub-sub.js示例应用程序 JavaScript。此应用程序显示设备（即您的 Amazon EC2 实例）如何使用 MQTT 库发布和订阅 MQTT 消息。pub-sub.js 示例应用程序订阅主题 topic_1，将 10 条消息发布到该主题，并在从消息代理收到这些消息时予以显示。

要安装和运行示例应用程序

1. 在您的 Amazon EC2 Instance Connect 窗口中，导航到 SDK 创建的 aws-iot-device-sdk-js-v2/samples/node/pub_sub 目录，并使用这些命令安装示例应用程序。

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   npm install
   ```

2. 在您的 Amazon EC2 Instance Connect 窗口中，使用此命令*your-iot-endpoint*从中 AWS IoT 获取。

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

3. 在您的 Amazon EC2 Instance Connect 窗口中，*your-iot-endpoint*按照指示插入并运行此命令。

   ```
   node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

示例应用程序：

1. AWS IoT Core 为您的账户连接到。
2. 订阅消息主题 topic_1，并显示它收到的有关该主题的消息。
3. 向主题 topic_1 发布 10 条消息。
4. 输出类似于以下内容：

   ```
   Publish received. topic:"topic_1" dup:false qos:1 retain:false
   {"message":"Hello world!","sequence":1}
   ```

```
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":2}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":3}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":4}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":5}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":6}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":7}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":8}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":9}
Publish received. topic:"topic_1" dup:false qos:1 retain:false
{"message":"Hello world!","sequence":10}
```

如果您在运行示例应用程序时遇到问题，请查看 the section called “排除示例应用程序的故障”。

您还可以将 `--verbosity debug` 参数添加到命令行，以便示例应用程序显示有关其正在执行的操作的详细消息。该信息可能会为您提供帮助以便您解决问题。

在 AWS IoT 控制台中查看来自示例应用程序的消息

您可以在 AWS IoT 控制台中使用 MQTT 测试客户端，在示例应用程序的消息通过消息代理时查看它们。

要查看示例应用程序发布的 MQTT 消息

1. 审核使用 MQTT 客户端查看 AWS IoT MQTT 消息。它可以帮助您了解如何使用 AWS IoT 控制台中的 MQTT 测试客户端来查看通过消息代理时的 MQTT 消息。
2. 在 AWS IoT 控制台中打开 MQTT 测试客户端。
3. 在 Subscribe to a topic（订阅主题）中，订阅主题 topic_1。
4. 在 Amazon EC2 实例连接窗口中，再次运行示例应用程序，然后在 AWS IoT 控制台的 MQTT 测试客户端中查看消息。

```
cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
```

```
node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
```

有关 MQTT 以及如何 AWS IoT Core 支持该协议的更多信息，请参阅 MQ TT。

使用你的 Windows、Linux 电脑或 Mac 作为 AWS IoT 设备

在本教程中，您将配置一台用于的个人计算机 AWS IoT。这些说明支持 Windows 和 Linux PC 以及 Mac。要完成此操作，您需要在电脑上安装某些软件。如果您不想在电脑上安装软件，可以尝试 使用 Amazon EC2 创建虚拟设备，它将在虚拟机上安装所有软件。

在本教程中，您将：

- 设置您的个人电脑
- 安装 Git、Python 和适用于 Python 的 AWS IoT 设备 SDK
- 设置策略并运行示例应用程序
- 在 AWS IoT 控制台中查看来自示例应用程序的消息
- 在 Python 中运行共享订阅示例

设置您的个人电脑

要完成本教程，您需要一台连接至互联网的 Windows 或 Linux PC 或 Mac。

在继续下一步之前，请确保您可以在电脑上打开命令行窗口。在 Windows PC 上使用 cmd.exe。在 Linux PC 或 Mac 上，使用 Terminal。

安装 Git、Python 和适用于 Python 的 AWS IoT 设备 SDK

在本节中，你将在电脑上安装 Python 和适用于 Python 的 AWS IoT 设备 SDK。

安装并使用最新版本的 Git 和 Python

要下载 Git 和 Python 并安装在您的电脑上

1. 检查您的电脑上是否已经安装了 Git。在命令行输入此命令。

   ```
   git --version
   ```

   如果命令显示 Git 版本，则表示已安装 Git，您可以继续执行下一步。

如果命令显示错误，请打开 https://git-scm.com/download 并为您的电脑安装 Git。

2. 检查您是否已安装了 Python。在命令行中输入此命令。

```
python -V
```

> **Note**
>
> 如果此命令返回错误：`Python was not found`，这可能是因为您的操作系统将 Python v3.x 可执行文件调用为 `Python3`。在这种情况下，请将 `python` 的所有实例替换为 `python3`，并继续本教程的其余部分。

如果命令显示 Python 版本，则表示已安装 Python。此教程需要 Python v3.7 或更高版本。

3. 如果已安装 Python，您可以跳过本部分的其余步骤。如果没有，请继续。
4. 打开 https://www.python.org/downloads/ 并为您的电脑下载安装程序。
5. 如果下载的程序没有自动开始安装，请运行下载的程序以安装 Python。
6. 验证 Python 的安装。

```
python -V
```

确认该命令显示 Python 版本。如果没有显示 Python 版本，请尝试再次下载并安装 Python。

安装适用于 Python 的 AWS IoT 设备开发工具包

在电脑上安装适用于 Python 的 AWS IoT 设备 SDK

1. 安装适用于 Python 的 AWS IoT 设备开发工具包的 v2。

```
python3 -m pip install awsiotsdk
```

2. 将适用于 Python 的 AWS IoT 设备 SDK 存储库克隆到主目录的 aws-iot-device-sdk-python-v2 目录中。此流程引用您作为###安装的文件的基础目录。

#目录的实际位置取决于您的操作系统。

Linux/macOS

在 macOS 和 Linux 中，#目录为 ~。

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

Windows

在 Windows 中，您可以在 cmd 窗口中运行此命令来查找#目录路径。

```
echo %USERPROFILE%
cd %USERPROFILE%
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

> **Note**
>
> 如果你使用的是 Window PowerShell s 而不是cmd.exe，请使用以下命令。
>
> ```
> echo $home
> ```

有关更多信息，请参阅适用于 Python 的AWS IoT 设备 SDK GitHub 存储库。

准备运行示例应用程序

要准备您的系统以运行示例应用程序

- 创建 `certs` 目录。进入 `certs` 目录，复制您在 the section called “创建 AWS IoT 资源” 中创建和注册事物对象时保存的私有密钥、设备证书和根 CA 证书文件。目标目录中每个文件的文件名应与表中的文件名匹配。

  下一部分中的命令假定您的密钥和证书文件均已存储在设备上，如下表中所示。

  Linux/macOS

  运行此命令以创建您将在运行示例应用程序时使用的 `certs` 子目录。

```
mkdir ~/certs
```

进入新的子目录，将文件复制到下表所示的目标文件路径。

证书文件名

文件	文件路径
私有密钥	~/certs/private.pem.key
设备证书	~/certs/device.pem.crt
根 CA 证书	~/certs/Amazon-root-CA-1.pem

运行此命令以列出 certs 目录中的文件，并将其与表中列出的文件进行比较。

```
ls -l ~/certs
```

Windows

运行此命令以创建您将在运行示例应用程序时使用的 certs 子目录。

```
mkdir %USERPROFILE%\certs
```

进入新的子目录，将文件复制到下表所示的目标文件路径。

证书文件名

文件	文件路径
私有密钥	%USERPROFILE%\certs\private.pem.key
设备证书	%USERPROFILE%\certs\device.pem.crt
根 CA 证书	%USERPROFILE%\certs\Amazon-root-CA-1.pem

运行此命令以列出 `certs` 目录中的文件，并将其与表中列出的文件进行比较。

```
dir %USERPROFILE%\certs
```

设置策略并运行示例应用程序

在本部分中，您将设置策略并运行在 AWS IoT Device SDK for Python的 `aws-iot-device-sdk-python-v2/samples` 目录中找到的 `pubsub.py` 示例脚本。此脚本显示设备将如何使用 MQTT 库发布和订阅 MQTT 消息。

`pubsub.py` 示例应用程序订阅了一个主题，`test/topic`，将 10 条消息发布到该主题，并在从消息代理收到这些消息时予以显示。

要运行 `pubsub.py` 示例应用程序，您需要具有以下信息：

应用程序参数值

参数	在何处查找值
your-iot-endpoint	1. 在 AWS IoT 控制台的左侧菜单中，选择 Settings（设置）。 2. 在 Settings（设置）页面上，您的终端节点将显示在 Device data endpoint（设备数据终端节点）部分。

该*your-iot-endpoint*值的格式为:*endpoint_id*`-ats.iot.`*region*`.amazonaws.com`，例如`a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com`。

在运行脚本之前，请确保事物的策略为示例脚本提供了连接、订阅、发布和接收的权限。

要查找和查看事物资源的策略文档

1. 在 AWS IoT 控制台的 Things（事物）列表中，找到代表设备的事物资源。
2. 选择代表设备的事物资源名称链接来打开 Thing details（事物详细信息）页面。
3. 在Thing details（事物详细信息）页面，Certificates（证书）选项卡中，选择附加到事物资源的证书。列表中应只有一个证书。如果有多个证书，请选择文件安装在设备上并连接到 AWS IoT Core 上的证书。

在Certificate（证书）详细信息页面的Policies（策略）选项卡中，选择附加到该证书的策略。应该只有一个策略。如果有多个策略，请重复下一步，确保至少有一个策略授予所需的访问权限。

4. 在策略概述页面，找到 JSON 编辑器然后选择编辑策略文档，根据需要查看和编辑策略文档。

5. 下列示例中显示了 JSON 策略。在"Resource"元素 AWS 账户 中，将每个Resource值*region:account*替换为你的 AWS 区域 和。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account:topic/test/topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account:topicfilter/test/topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:region:account:client/test-*"
            ]
        }
    ]
}
```

Linux/macOS

在 Linux/macOS 上运行示例脚本

1. 在命令行窗口中，导航到 SDK 使用这些命令创建的 `~/aws-iot-device-sdk-python-v2/samples/node/pub_sub` 目录。

```
cd ~/aws-iot-device-sdk-python-v2/samples
```

2. 在命令行窗口中，*your-iot-endpoint*按照指示进行替换，然后运行此命令。

```
python3 pubsub.py --endpoint your-iot-endpoint --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/private.pem.key
```

Windows

要在 Windows PC 上运行示例应用程序

1. 在您的命令行窗口中，导航到 SDK 创建的 `%USERPROFILE%\aws-iot-device-sdk-python-v2\samples` 目录，并使用这些命令安装示例应用程序。

```
cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
```

2. 在命令行窗口中，*your-iot-endpoint*按照指示进行替换，然后运行此命令。

```
python3 pubsub.py --endpoint your-iot-endpoint --ca_file %USERPROFILE%\certs\Amazon-root-CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key %USERPROFILE%\certs\private.pem.key
```

示例脚本：

1. 连接到您的账户。 AWS IoT Core
2. 订阅消息主题 test/topic，并显示其收到的有关该主题的消息。
3. 向主题 test/topic 发布 10 条消息。
4. 输出类似于以下内容：

```
Connected!
```

```
Subscribing to topic 'test/topic'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 10 message(s)
Publishing message to topic 'test/topic': Hello World! [1]
Received message from topic 'test/topic': b'"Hello World! [1]"'
Publishing message to topic 'test/topic': Hello World! [2]
Received message from topic 'test/topic': b'"Hello World! [2]"'
Publishing message to topic 'test/topic': Hello World! [3]
Received message from topic 'test/topic': b'"Hello World! [3]"'
Publishing message to topic 'test/topic': Hello World! [4]
Received message from topic 'test/topic': b'"Hello World! [4]"'
Publishing message to topic 'test/topic': Hello World! [5]
Received message from topic 'test/topic': b'"Hello World! [5]"'
Publishing message to topic 'test/topic': Hello World! [6]
Received message from topic 'test/topic': b'"Hello World! [6]"'
Publishing message to topic 'test/topic': Hello World! [7]
Received message from topic 'test/topic': b'"Hello World! [7]"'
Publishing message to topic 'test/topic': Hello World! [8]
Received message from topic 'test/topic': b'"Hello World! [8]"'
Publishing message to topic 'test/topic': Hello World! [9]
Received message from topic 'test/topic': b'"Hello World! [9]"'
Publishing message to topic 'test/topic': Hello World! [10]
Received message from topic 'test/topic': b'"Hello World! [10]"'
10 message(s) received.
Disconnecting...
Disconnected!
```

如果您在运行示例应用程序时遇到问题，请查看 the section called “排除示例应用程序的故障”。

您还可以将 `--verbosity Debug` 参数添加到命令行，以便示例应用程序显示有关其正在执行的操作的详细消息。该信息可能会帮助您改正此问题。

在 AWS IoT 控制台中查看来自示例应用程序的消息

您可以在 AWS IoT 控制台中使用 MQTT 测试客户端，在示例应用程序的消息通过消息代理时查看它们。

要查看示例应用程序发布的 MQTT 消息

1. 审核使用 MQTT 客户端查看 AWS IoT MQTT 消息。它可以帮助您了解如何使用 AWS IoT 控制台中的 MQTT 测试客户端来查看通过消息代理时的 MQTT 消息。
2. 在 AWS IoT 控制台中打开 MQTT 测试客户端。

3. 在 Subscribe to a topic（订阅主题）中，订阅主题 test/topic。
4. 在命令行窗口中，再次运行示例应用程序，并在 AWS IoT 控制台的 MQTT 客户端中查看消息。

   Linux/macOS

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic test/topic --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

   Windows

   ```
   cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
   python3 pubsub.py --topic test/topic --ca_file %USERPROFILE%\certs\Amazon-root-
   CA-1.pem --cert %USERPROFILE%\certs\device.pem.crt --key %USERPROFILE%\certs
   \private.pem.key --endpoint your-iot-endpoint
   ```

有关 MQTT 以及如何 AWS IoT Core 支持该协议的更多信息，请参阅 MQ TT。

在 Python 中运行共享订阅示例

AWS IoT Core 支持 MQTT 3 和 MQTT 5 的共享订阅。共享订阅允许多个客户端共享对某个主题的订阅，并且只有一个客户端会收到使用随机分布发布到该主题的消息。要使用共享订阅，客户需要订阅共享订阅的主题筛选条件：`$share/{ShareName}/{TopicFilter}`。

设置策略并运行共享订阅示例

1. 要运行共享订阅示例，您必须按照 MQTT 5 共享订阅中的说明设置您的事物的策略。
2. 要运行共享订阅示例，请运行以下命令。

   Linux/macOS

   在 Linux/macOS 上运行示例脚本

   1. 在命令行窗口中，导航到 SDK 使用这些命令创建的 `~/aws-iot-device-sdk-python-v2/samples` 目录。

      ```
      cd ~/aws-iot-device-sdk-python-v2/samples
      ```

   2. 在命令行窗口中，*your-iot-endpoint*按照指示进行替换，然后运行此命令。

```
python3 mqtt5_shared_subscription.py --endpoint your-iot-endpoint --ca_file
 ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
private.pem.key --group_identifier consumer
```

Windows

要在 Windows PC 上运行示例应用程序

1. 在您的命令行窗口中，导航到 SDK 创建的 %USERPROFILE%\aws-iot-device-sdk-python-v2\samples 目录，并使用这些命令安装示例应用程序。

```
cd %USERPROFILE%\aws-iot-device-sdk-python-v2\samples
```

2. 在命令行窗口中，*your-iot-endpoint*按照指示进行替换，然后运行此命令。

```
python3 mqtt5_shared_subscription.py --endpoint your-iot-endpoint --ca_file
 %USERPROFILE%\certs\Amazon-root-CA-1.pem --cert %USERPROFILE%\certs
\device.pem.crt --key %USERPROFILE%\certs\private.pem.key --group_identifier
 consumer
```

> **Note**
>
> 在运行示例时，您可以根据需要选择指定组标识符（例如 --group_identifier consumer）。如果未指定组标识符，则 python-sample 为默认组标识符。

3. 命令行中的输出可能如下所示：

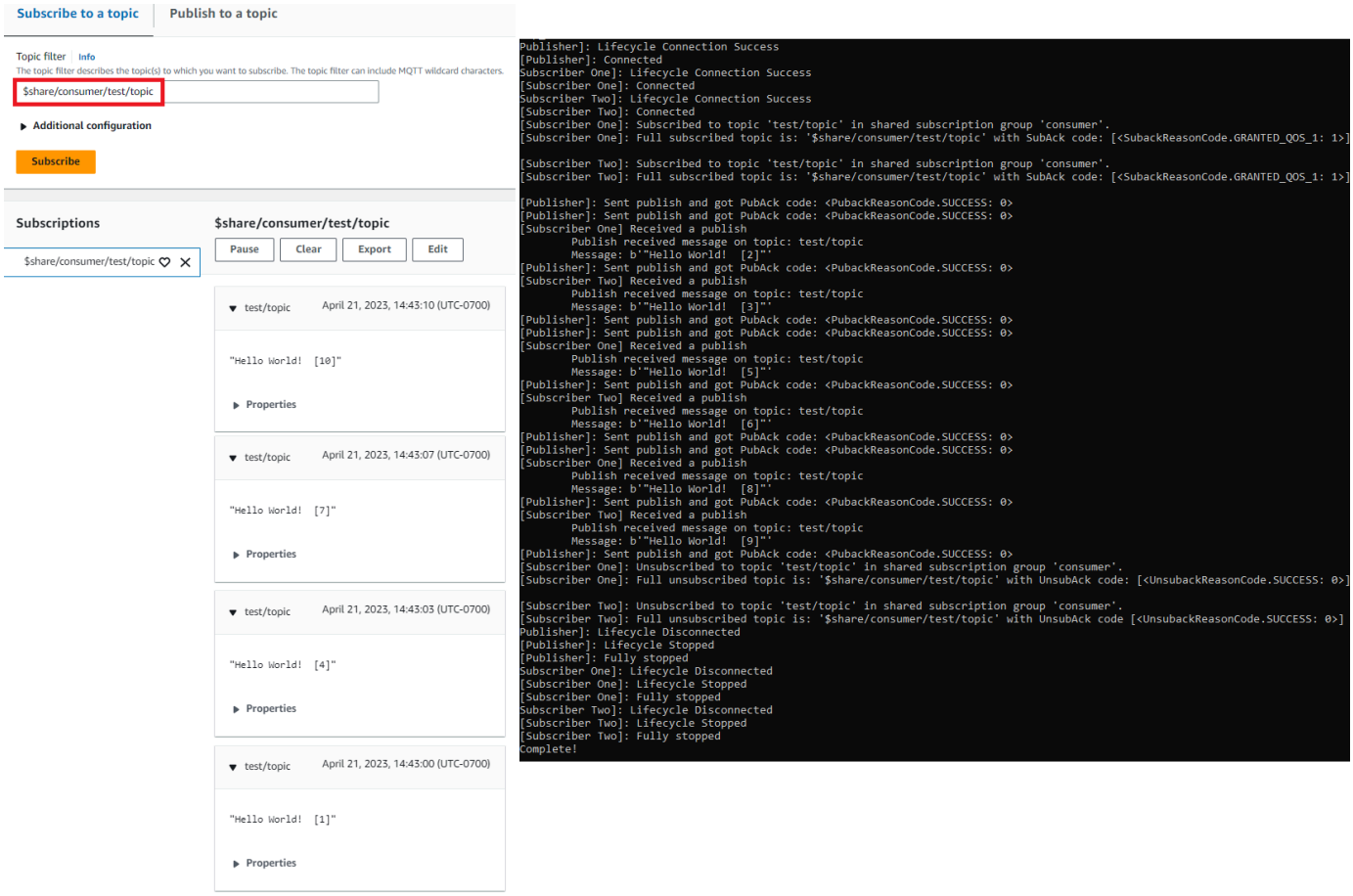
```
Publisher]: Lifecycle Connection Success
[Publisher]: Connected
Subscriber One]: Lifecycle Connection Success
[Subscriber One]: Connected
Subscriber Two]: Lifecycle Connection Success
[Subscriber Two]: Connected
[Subscriber One]: Subscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber One]: Full subscribed topic is: '$share/consumer/test/topic' with
 SubAck code: [<SubackReasonCode.GRANTED_QOS_1: 1>]
```

```
[Subscriber Two]: Subscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber Two]: Full subscribed topic is: '$share/consumer/test/topic' with
 SubAck code: [<SubackReasonCode.GRANTED_QOS_1: 1>]
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [1]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [2]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [3]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [4]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [5]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [6]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [7]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [8]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber Two] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [9]"'
[Publisher]: Sent publish and got PubAck code: <PubackReasonCode.SUCCESS: 0>
[Subscriber One] Received a publish
        Publish received message on topic: test/topic
        Message: b'"Hello World!  [10]"'
```

```
[Subscriber One]: Unsubscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber One]: Full unsubscribed topic is: '$share/consumer/test/topic' with
 UnsubAck code: [<UnsubackReasonCode.SUCCESS: 0>]
[Subscriber Two]: Unsubscribed to topic 'test/topic' in shared subscription group
 'consumer'.
[Subscriber Two]: Full unsubscribed topic is: '$share/consumer/test/topic' with
 UnsubAck code [<UnsubackReasonCode.SUCCESS: 0>]
Publisher]: Lifecycle Disconnected
[Publisher]: Lifecycle Stopped
[Publisher]: Fully stopped
Subscriber One]: Lifecycle Disconnected
[Subscriber One]: Lifecycle Stopped
[Subscriber One]: Fully stopped
Subscriber Two]: Lifecycle Disconnected
[Subscriber Two]: Lifecycle Stopped
[Subscriber Two]: Fully stopped
Complete!
```

4. 在 AWS IoT 控制台中打开 MQTT 测试客户端。在订阅主题中，订阅共享订阅的主题，例如：`$share/consumer/test/topic`。在运行示例时，您可以根据需要指定组标识符（例如 `--group_identifier consumer`）。如果您未指定组标识符，则默认值为 `python-sample`。有关更多信息，请参阅《AWS IoT Core 开发人员指南》中的 MQTT 5 共享订阅 Python 示例和共享订阅。

   在命令行窗口中，再次运行示例应用程序，并在 AWS IoT 控制台的 MQTT 测试客户端和命令行中观察消息的分布。



连接 Raspberry Pi 或其他设备

在本节中，我们将配置一个 Raspberry Pi 以与一起使用 AWS IoT。如果您要连接另一台设备，Raspberry Pi 的说明包括可以帮助您调整这些指令以用到您设备上的参考资料。

这通常需要大约 20 分钟，但如果您需要安装许多系统软件升级，则可能需要更长的时间。

在本教程中，您将：

- 设置您的设备
- 安装 AWS IoT 设备 SDK 所需的工具和库
- 安装 AWS IoT 设备 SDK
- 安装并运行示例应用程序
- 在 AWS IoT 控制台中查看来自示例应用程序的消息

> **Important**
>
> 将这些指令用于其它设备和操作系统可能会非常困难。您需要充分了解您的设备，以便能够解释这些说明并将它们应用到您的设备上。
> 如果您在配置设备时遇到困难 AWS IoT，可以尝试使用其他设备选项作为替代方案，例如使用 Amazon EC2 创建虚拟设备或使用你的 Windows、Linux 电脑或 Mac 作为 AWS IoT 设备。

设置您的设备

此步骤的目标是收集配置设备所需的信息，以便它可以启动操作系统（OS）、连接到互联网，并允许您在命令行界面与设备进行交互。

要完成本教程，您需要：

- 一个 AWS 账户。如果您没有账户，请完成 设置你的 AWS 账户 中介绍的步骤然后继续操作。
- Raspberry Pi 3 Model B 或更新型号。这可能适用于早期版本的 Raspberry Pi，但这尚未经过测试。
- Raspberry Pi OS (32 位)或更高版本。我们始终建议使用最新版本的 Raspberry Pi OS。早期版本的操作系统可能正常工作，但这尚未经过测试。

  要运行此示例，您不需要使用图形用户界面 (GUI) 安装桌面；但是，如果您不熟悉 Raspberry Pi 并且您的 Raspberry Pi 硬件支持，则使用带有 GUI 的桌面可能会使操作更容易。
- 以太网或 WiFi 连接。
- 键盘、鼠标、显示器、电缆、电源和设备所需的其它硬件。

> **Important**
>
> 您的设备必须先安装、配置和运行操作系统，然后才能继续执行下一步。设备必须连接到互联网，并且您需要能够使用其命令行界面访问设备。命令行访问可以通过直接连接的键盘、鼠标和显示器，也可以使用 SSH 终端远程接口进行访问。

如果您在 Raspberry Pi 上运行具有图形用户界面 (GUI) 的操作系统，请在设备上打开终端窗口，然后在该窗口中执行以下说明。否则，如果您使用远程终端（如 PuTTY）连接到您的设备，请打开设备的远程终端并使用该终端。

安装 AWS IoT 设备 SDK 所需的工具和库

在安装 AWS IoT 设备 SDK 和示例代码之前，请确保您的系统是最新的，并且具有安装 SDK 所需的工具和库。

1. 更新操作系统并安装所需的库

   在安装 AWS IoT 设备 SDK 之前，请在设备上的终端窗口中运行这些命令以更新操作系统并安装所需的库。

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get upgrade
   ```

   ```
   sudo apt-get install cmake
   ```

   ```
   sudo apt-get install libssl-dev
   ```

2. 安装 Git

   如果设备的操作系统未安装 Git，则必须安装该操作系统才能为其安装 AWS IoT 设备 SDK JavaScript。

   a. 通过运行此命令测试是否已安装 Git。

      ```
      git --version
      ```

   b. 如果上一个命令返回了 Git 版本，则表示 Git 已经安装，您可以跳到步骤 3。

   c. 如果在运行 git 命令时显示了错误，则请运行此命令来安装 Git。

      ```
      sudo apt-get install git
      ```

   d. 再次运行此命令测试是否已安装 Git。

      ```
      git --version
      ```

   e. 如果已安装 Git，请跳到下一个部分。如果没有，请排除故障并更正错误，然后再继续。你需要 Git 来安装的 AWS IoT 设备软件开发工具包 JavaScript。

安装 AWS IoT 设备 SDK

安装 AWS IoT 设备软件开发工具包。

Python

在本节中，你将在你的设备上安装 Python、其开发工具和适用于 Python 的 AWS IoT 设备 SDK。这些说明适用于运行最新版本 Raspberry Pi 操作系统的 Raspberry Pi。如果您有其他设备或使用其它操作系统，则可能需要为设备对以下说明的操作进行调整。

1. 安装 Python 及其开发工具

   适用于 Python 的 AWS IoT 设备 SDK 需要在 Raspberry Pi 上安装 Python v3.5 或更高版本。

   在设备的终端窗口中，运行这些命令。

   1. 运行此命令以确定设备上安装的 Python 版本。

      ```
      python3 --version
      ```

      如果安装了 Python，它将显示其版本。
   2. 如果显示的版本为 `Python 3.5` 或更高版本，则您可以跳至步骤 2。
   3. 如果显示的版本低于 `Python 3.5`，您可以运行此命令来安装正确版本。

      ```
      sudo apt install python3
      ```
   4. 运行此命令以确认现在安装了正确版本的 Python。

      ```
      python3 --version
      ```
2. 针对 pip3 的测试

   在设备的终端窗口中，运行这些命令。

   1. 运行以下命令，查看是否安装了 pip3。

      ```
      pip3 --version
      ```
   2. 如果命令返回了版本号，则表示已安装 pip3，您可以跳到步骤 3。
   3. 如果之前的命令返回了错误，请运行此命令以安装 pip3。

```
sudo apt install python3-pip
```

4. 运行以下命令，查看是否安装了 pip3。

```
pip3 --version
```

3. 安装当前适用于 Python 的 AWS IoT 设备 SDK

   安装适用于 Python 的 AWS IoT 设备 SDK 并将示例应用程序下载到您的设备上。

   在您的设备上运行这些命令。

```
cd ~
python3 -m pip install awsiotsdk
```

```
git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
```

JavaScript

在本节中，你将在你的设备 JavaScript 上安装 Node.js、npm 包管理器和 AWS IoT 设备软件开发工具包。这些说明适用于运行 Raspberry Pi 操作系统的 Raspberry Pi。如果您有其他设备或使用其它操作系统，则可能需要为设备对以下说明的操作进行调整。

1. 安装最新版本的 Node.js

   的 AWS IoT 设备 SDK JavaScript 需要在你的 Raspberry Pi 上安装 Node.js 和 npm 包管理器。

   a. 通过输入此命令以下载最新版本的 Node 存储库。

```
cd ~
curl -sL https://deb.nodesource.com/setup_12.x | sudo -E bash -
```

   b. 安装 Node 和 npm。

```
sudo apt-get install -y nodejs
```

   c. 验证 Node 的安装。

```
node -v
```

确认该命令显示了 Node 版本。本教程需要 Node v10.0 或更高版本。如果未显示 Node 版本，请尝试再次下载 Node 存储库。

d. 验证 npm 的安装。

```
npm -v
```

确认该命令显示了 npm 版本。如果未显示 npm 版本，请尝试再次安装 Node 和 npm。

e. 重启设备。

```
sudo shutdown -r 0
```

设备重新启动后继续操作。

2. 安装适用于的 AWS IoT 设备 SDK JavaScript

在你的 Raspberry Pi JavaScript 上安装适用于的 AWS IoT 设备 SDK。

a. 将 JavaScript存储库的 AWS IoT 设备 SDK 克隆到您的#目录目录中。`aws-iot-device-sdk-js-v2`在 Raspberry Pi 中，#目录为 `~/`，将在下列命令中用作#目录。如果您的设备使用不同的路径来作为#目录，则必须在以下命令中将 `~/` 替换为您设备的正确路径。

这些命令会创建 `~/aws-iot-device-sdk-js-v2` 目录并将 SDK 代码复制到其中。

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
```

b. 更改为您在之前步骤中创建的 `aws-iot-device-sdk-js-v2` 目录，然后 运行 `npm install` 以安装开发工具包。`npm install` 命令将调用可能需要几分钟才能完成的 `aws-crt` 库构建。

```
cd ~/aws-iot-device-sdk-js-v2
npm install
```

安装并运行示例应用程序

在本节中，您将安装并运行 AWS IoT 设备 SDK 中的pubsub示例应用程序。此应用程序将显示您的设备如何使用 MQTT 库发布和订阅 MQTT 消息。示例应用程序订阅了一个主题，topic_1，将 10 条消息发布到该主题，并在从消息代理收到这些消息时予以显示。

安装证书文件

示例应用程序要求在设备上安装对设备进行身份验证的证书文件。

要为示例应用程序安装设备证书文件

1. 运行这些命令，在您的#目录中创建 certs 子目录。

```
cd ~
mkdir certs
```

2. 将您之前在 the section called “创建 AWS IoT 资源” 中创建的私有密钥、设备证书和根 CA 证书复制到 ~/certs 目录中。

   如何将证书文件复制到设备取决于设备和操作系统，此处未予介绍。但是，如果您的设备支持图形用户界面 (GUI) 并且具有 Web 浏览器，则可以在设备的 Web 浏览器中执行 the section called “创建 AWS IoT 资源” 中所述的流程，以将生成的文件直接下载到您的设备。

   下一节中的命令假定您的密钥和证书文件存储在设备上，如下表中所示。

   证书文件名

文件	文件路径
根 CA 证书	~/certs/Amazon-root-CA-1.pem
设备证书	~/certs/device.pem.crt
私有密钥	~/certs/private.pem.key

要运行示例应用程序，您需要以下信息：

应用程序参数值

参数	在何处查找值
your-iot-endpoint	在 AWS IoT 控制台中，选择 All devices（所有设备），然后选择 Things（事物）。 在 AWS IoT 菜单的设置页面上。终端节点会显示在 Device data endpoint（设备数据终端节点）部分。

该*your-iot-endpoint*值的格式为:*endpoint_id*-ats.iot.*region*.amazonaws.com，例如a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com。

Python

要安装和运行示例应用程序

1. 导航到示例应用程序目录。

```
cd ~/aws-iot-device-sdk-python-v2/samples
```

2. 在命令行窗口中，*your-iot-endpoint*按照指示进行替换，然后运行此命令。

```
python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
endpoint
```

3. 观察示例应用程序：

   1. 连接到您账户的 AWS IoT 服务。
   2. 订阅消息主题 topic_1，并显示它收到的有关该主题的消息。
   3. 向主题 topic_1 发布 10 条消息。
   4. 输出类似于以下内容：

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to topic 'topic_1'...
```

```
Subscribed with QoS.AT_LEAST_ONCE
Sending 10 message(s)
Publishing message to topic 'topic_1': Hello World! [1]
Received message from topic 'topic_1': b'Hello World! [1]'
Publishing message to topic 'topic_1': Hello World! [2]
Received message from topic 'topic_1': b'Hello World! [2]'
Publishing message to topic 'topic_1': Hello World! [3]
Received message from topic 'topic_1': b'Hello World! [3]'
Publishing message to topic 'topic_1': Hello World! [4]
Received message from topic 'topic_1': b'Hello World! [4]'
Publishing message to topic 'topic_1': Hello World! [5]
Received message from topic 'topic_1': b'Hello World! [5]'
Publishing message to topic 'topic_1': Hello World! [6]
Received message from topic 'topic_1': b'Hello World! [6]'
Publishing message to topic 'topic_1': Hello World! [7]
Received message from topic 'topic_1': b'Hello World! [7]'
Publishing message to topic 'topic_1': Hello World! [8]
Received message from topic 'topic_1': b'Hello World! [8]'
Publishing message to topic 'topic_1': Hello World! [9]
Received message from topic 'topic_1': b'Hello World! [9]'
Publishing message to topic 'topic_1': Hello World! [10]
Received message from topic 'topic_1': b'Hello World! [10]'
10 message(s) received.
Disconnecting...
Disconnected!
```

如果您在运行示例应用程序时遇到问题，请查看 the section called “排除示例应用程序的故障”。

您还可以将 `--verbosity Debug` 参数添加到命令行，以便示例应用程序显示有关其正在执行的操作的详细消息。该信息可能会为您提供帮助以便您解决问题。

JavaScript

要安装和运行示例应用程序

1. 在您的命令行窗口中，导航到 SDK 创建的 `~/aws-iot-device-sdk-js-v2/samples/node/pub_sub` 目录，并使用这些命令安装示例应用程序。`npm install` 命令将调用可能需要几分钟才能完成的 `aws-crt` 库构建。

```
cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
```

```
npm install
```

2. 在命令行窗口中，*your-iot-endpoint*按照指示进行替换，然后运行此命令。

```
node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
endpoint
```

3. 观察示例应用程序：

   1. 连接到您账户的 AWS IoT 服务。
   2. 订阅消息主题 topic_1，并显示它收到的有关该主题的消息。
   3. 向主题 topic_1 发布 10 条消息。
   4. 输出类似于以下内容：

```
Publish received on topic topic_1
{"message":"Hello world!","sequence":1}
Publish received on topic topic_1
{"message":"Hello world!","sequence":2}
Publish received on topic topic_1
{"message":"Hello world!","sequence":3}
Publish received on topic topic_1
{"message":"Hello world!","sequence":4}
Publish received on topic topic_1
{"message":"Hello world!","sequence":5}
Publish received on topic topic_1
{"message":"Hello world!","sequence":6}
Publish received on topic topic_1
{"message":"Hello world!","sequence":7}
Publish received on topic topic_1
{"message":"Hello world!","sequence":8}
Publish received on topic topic_1
{"message":"Hello world!","sequence":9}
Publish received on topic topic_1
{"message":"Hello world!","sequence":10}
```

   如果您在运行示例应用程序时遇到问题，请查看 the section called “排除示例应用程序的故障”。

您还可以将 `--verbosity Debug` 参数添加到命令行，以便示例应用程序显示有关其正在执行的操作的详细消息。该信息可能会为您提供帮助以便您解决问题。

在 AWS IoT 控制台中查看来自示例应用程序的消息

您可以在 AWS IoT 控制台中使用 MQTT 测试客户端，在示例应用程序的消息通过消息代理时查看它们。

要查看示例应用程序发布的 MQTT 消息

1. 审核使用 MQTT 客户端查看 AWS IoT MQTT 消息。它可以帮助您了解如何使用 AWS IoT 控制台中的 MQTT 测试客户端来查看通过消息代理时的 MQTT 消息。
2. 在 AWS IoT 控制台中打开 MQTT 测试客户端。
3. 订阅主题 topic_1。
4. 在命令行窗口中，再次运行示例应用程序，并在 AWS IoT 控制台的 MQTT 客户端中查看消息。

   Python

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

   JavaScript

   ```
   cd ~/aws-iot-device-sdk-js-v2/samples/node/pub_sub
   node dist/index.js --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --
   cert ~/certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-
   endpoint
   ```

排除示例应用程序的故障

如果您在尝试运行示例应用程序时遇到错误，请检查以下几个事项。

检查证书

如果证书未激活， AWS IoT 则不接受任何使用该证书进行授权的连接尝试。创建证书时，很容易忽略 Activate（激活）按钮。幸运的是，您可以在 AWS IoT 控制台激活您的证书。

要检查证书的激活状态

1. 在 AWS IoT 控制台，在左侧菜单中，选择 Secure（安全），然后选择 Certificates（证书）。
2. 在证书列表中，找到您为练习创建的证书，并在 Status（状态）栏查看其状态。

   如果您不记得证书的名称，请检查 Inactive（未激活）的证书以查看其中是否包含是您正在使用的。

   在列表中选择相应证书以打开其详细信息页面。在详细信息页面中，您可以看到其 Create date（创建日期）以帮助您识别证书。
3. 要激活未激活的证书，请在证书详细信息页面上，选择 Actions（操作），然后选择 Activate（激活）。

如果找到了正确的证书且其处于激活状态，但运行示例应用程序时仍遇到了问题，请按照下一步所述检查其策略。

您还可以按照 the section called “创建一个事物对象” 中所述的步骤尝试创建新事物和新证书。如果您创建了一个新事物，则需要为其指定新事物名称并将新证书文件下载到您的设备上。

检查附加到证书上的策略。

策略授权中的操作 AWS IoT。如果用于连接 AWS IoT 的证书没有策略，或者没有允许其连接的策略，则连接将被拒绝，即使证书处于激活状态。

要检查附加到证书的策略

1. 查找上一项中所述的证书，然后打开其详细信息页面。
2. 在证书详细信息页面的左侧菜单中，选择 Policies（策略）以查看附加到证书的策略。
3. 如果证书没有附加策略，请选择 Actions（操作）菜单，然后选择 Attach policy（附加策略）。

   选择您之前在 the section called “创建 AWS IoT 资源” 中创建的策略。
4. 如果附加了策略，请选择策略磁贴以打开其详细信息页面。

   在详细信息页面上，查看 Policy document（策略文档），以确保它包含与您在 the section called “创建 AWS IoT 策略” 中创建的策略相同的信息。

检查命令行

请确保为您的系统使用了正确的命令行。Linux 和 macOS 系统上使用的命令通常与在 Windows 系统上使用的命令不同。

检查终端节点地址

查看您输入的命令，并将命令中的终端节点地址对照 AWS IoT 控制台中的信息仔细检查。

检查证书文件的文件名

将您输入的命令中的文件名与 `certs` 目录中的证书文件的文件名进行对比。

某些系统可能要求将文件名放在引号中才能正常工作。

检查 SDK 安装状况

请确保您的 SDK 已正确完成安装。

如有疑问，请在设备上重新安装 SDK。在大多数情况下，只需找到教程中标题为 “安装适用于 SDK ##的 AWS IoT 设备 SDK” 的部分，然后再次按照步骤操作即可。

如果您使用的是的 De AWS IoT vice SDK JavaScript，请记得在尝试运行示例应用程序之前先安装它们。安装 SDK 不会自动安装示例应用程序。示例应用程序必须在安装 SDK 后手动安装。

使用 MQTT 客户端查看 AWS IoT MQTT 消息

本节介绍如何在AWS IoT 控制台中使用 AWS IoT MQTT 测试客户端来查看发送和接收的 MQTT 消息。 AWS IoT本部分中使用的示例涉及 入门 AWS IoT Core 使用的示例；但您可以将示例中使用的 *topicName* 替换为您的 IoT 解决方案使用的任何主题名称或主题筛选条件。

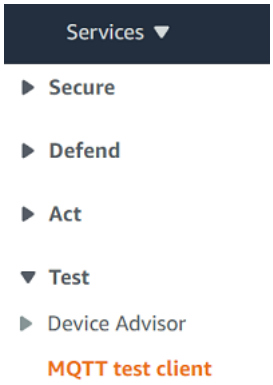
设备发布由主题标识的 MQTT 消息以将其状态传达给他们 AWS IoT，并 AWS IoT 发布 MQTT 消息以将更改和事件通知设备和应用程序。您可以使用 MQTT 客户端订阅这些主题，并在出现消息时查看这些消息。您还可以使用 MQTT 测试客户端将 MQTT 消息发布到您的中已订阅的设备和服务。 AWS 账户

内容

查看 MQTT 客户端中的 MQTT 消息

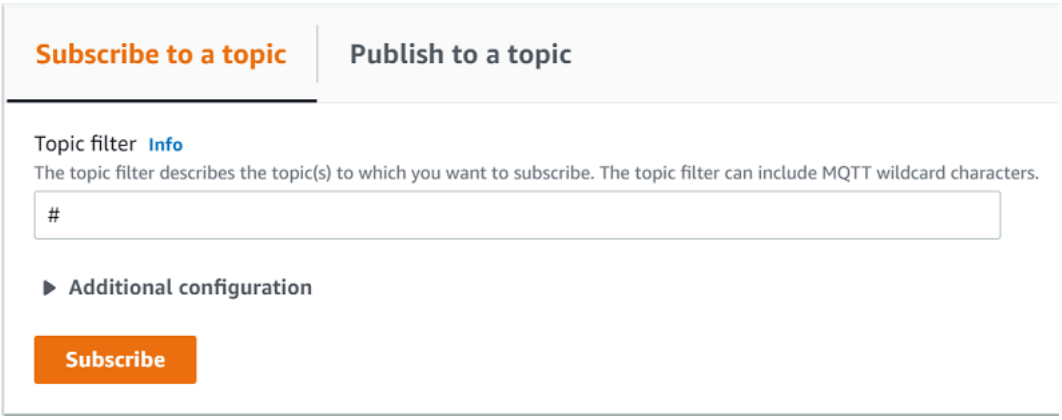
在 MQTT 测试客户端中查看 MQTT 消息

1. 打开 AWS IoT 控制台，在左侧菜单中选择 Test (测试)，然后选择 MQTT client (MQTT 客户端)。

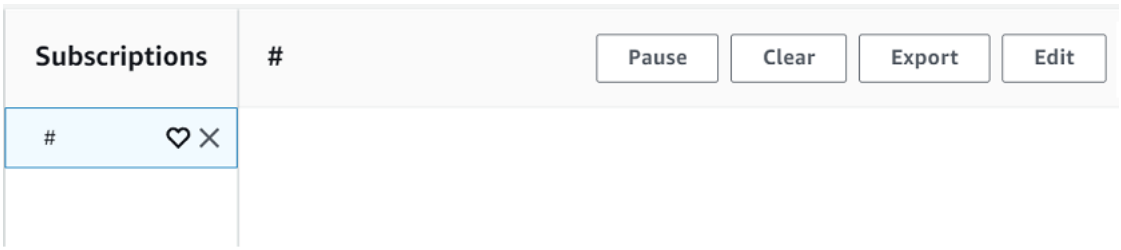


2. 在 Subscribe to a topic（订阅主题）选项卡中，输入*topicName*订阅您的设备发布的主题。对于入门示例应用程序，请订阅 **#**，它订阅了所有消息主题。

   继续使用入门示例操作，在 Subscribe to a topic（订阅主题）选项卡上的 Topic filter（主题筛选条件）字段，输入 **#**，然后选择 Subscribe（订阅）。



   主题消息日志页 # 将会打开，且 # 会显示在 Subscriptions（订阅）列表。如果您在中配置的设备the section called “配置您的设备”正在运行示例程序，则应在 # 消息日志 AWS IoT 中看到它发送到的消息。当收到带有订阅主题的消息时，消息日志条目将显示在 “发布” 部分下方。 AWS IoT



3. 在 # 消息日志页面，您还可以将消息发布到主题，但需要指定主题名称。您无法发布到 # 主题。

   发布到已订阅主题的消息将在收到时在消息日志中显示，最近的消息将首先显示。

排除 MQTT 消息的故障

使用通配符主题筛选条件

如果您的消息未按预期在消息日志中显示，请尝试按照 主题筛选条件 中所述订阅通配符主题筛选条件。MQTT 多级通配符主题筛选条件是哈希或井号 (#)，并且可以用作 Subscription topic（订阅主题）字段中的主题筛选条件。

订阅 # 主题筛选条件将订阅消息代理接收的每个主题。您可以将主题筛选条件路径替代为 # 多级通配符或“+”单级通配符，从而缩小筛选条件的范围。

在主题筛选条件中使用通配符时

- 多级通配符必须是主题筛选条件中的最后一个字符。
- 主题筛选条件路径在每个主题级别中只能有一个单级通配符。

例如：

主题筛选条件	显示带有以下内容的消息
#	任何主题名称
topic_1/#	以 topic_1/ 开头的主题名称
topic_1/level_2/#	以 topic_1/level_2/ 开头的主题名称
topic_1/+/level_3	以 topic_1/ 开头并以 /level_3 结尾的主题名称，并且在之间有一个任意值的元素。

有关主题筛选条件的更多信息，请参阅 主题筛选条件。

检查主题名称错误

MQTT 主题名称和主题筛选条件区分大小写。如果您的设备将消息发布到 Topic_1（带有一个大写的 T）而不是您订阅的主题 topic_1，则其消息将不会显示在 MQTT 测试客户端中。但是，订阅通配符主题筛选条件时，会显示设备正在发布消息，并且您会看到它使用的主题名称不是您所期望的主题名称。

从 MQTT 客户端发布 MQTT 消息

向 MQTT 主题发布消息

1. 在 MQTT 测试客户端页面上，在 Publish to a topic（发布到主题）选项卡上的 Topic name（主题名称）字段中，输入您消息的 *topicName*。在此示例中，使用 **my/topic**。

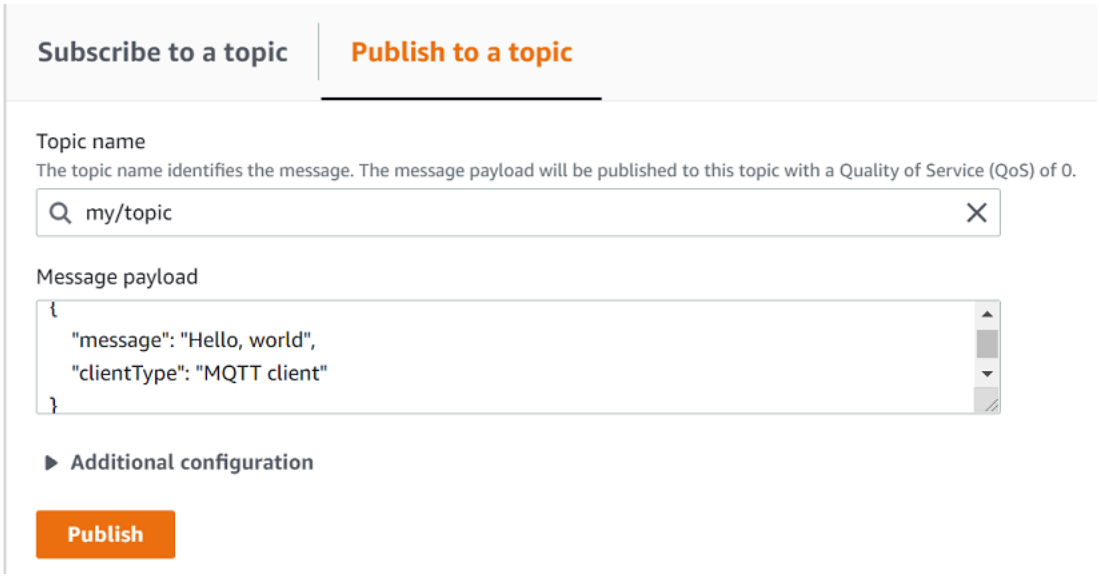
   > **Note**
   >
   > 不要在主题名称中使用个人身份信息，无论您是在测试 MQTT 客户端还是在系统实施中使用这些信息。主题名称可以出现在未加密的通信和报告中。

2. 在消息负载窗口中，输入以下 JSON：

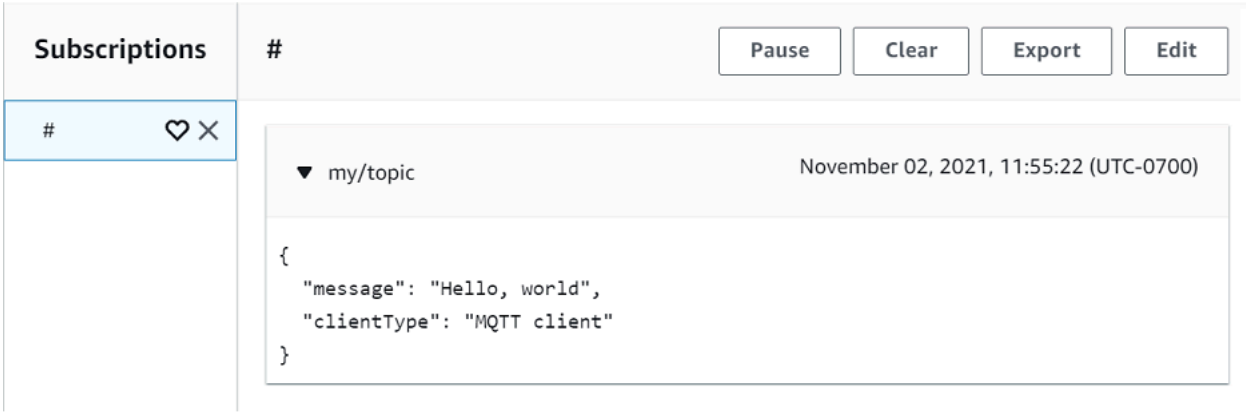
   ```
   {
       "message": "Hello, world",
       "clientType": "MQTT test client"
   }
   ```

3. 选择 Publish（发布）以将消息发布到 AWS IoT。

   > **Note**
   >
   > 确保您已订阅 my/topic 主题，然后再发布您的消息。



4. 在 Subscription（订阅）列中，选择 my/topic 以查看消息。您应该看到该消息显示在 MQTT 测试客户端中的发布消息有效负载窗口下。



您可以通过更改 Topic name（主题名称）字段中的 *topicName*，然后选择 Publish（发布）按钮，以将 MQTT 消息发布到其它主题。

> **Important**
>
> 当您创建多个具有重叠主题的订阅时（例如 probe1/temperature 和 probe1/#），则发布到与两个订阅匹配的主题的单个消息可能会被多次传送，每个重叠订阅一次发送一次。

在 MQTT 客户端中测试共享订阅

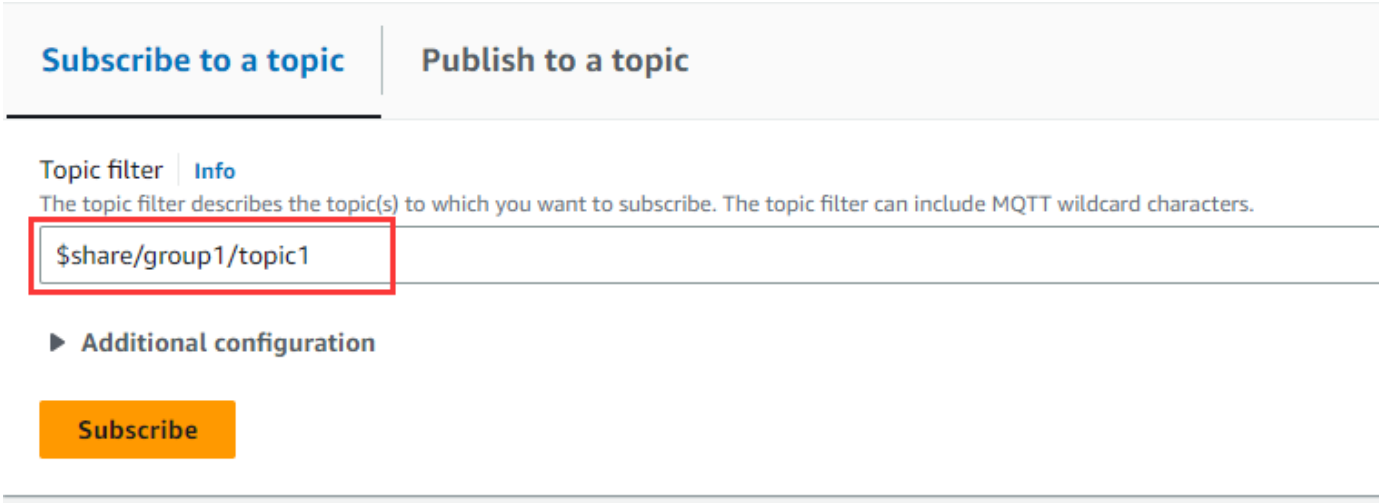
本节介绍如何在AWS IoT 控制台中使用 AWS IoT MQTT 客户端查看通过共享订阅发送和接收的 AWS IoT MQTT 消息。 ???允许多个客户端共享一个主题的订阅，只有一个客户接收使用随机分布发布到该主题的消息。要模拟多个 MQTT 客户端（在本例中为两个 MQTT 客户端）共享同一个订阅，您可以通过多个 Web 浏览器在AWS IoT 控制台中打开 AWS IoT MQTT 客户端。本节使用的示例与 入门 AWS IoT Core中使用的示例无关。有关更多信息，请参阅共享订阅。

共享对 MQTT 主题的订阅

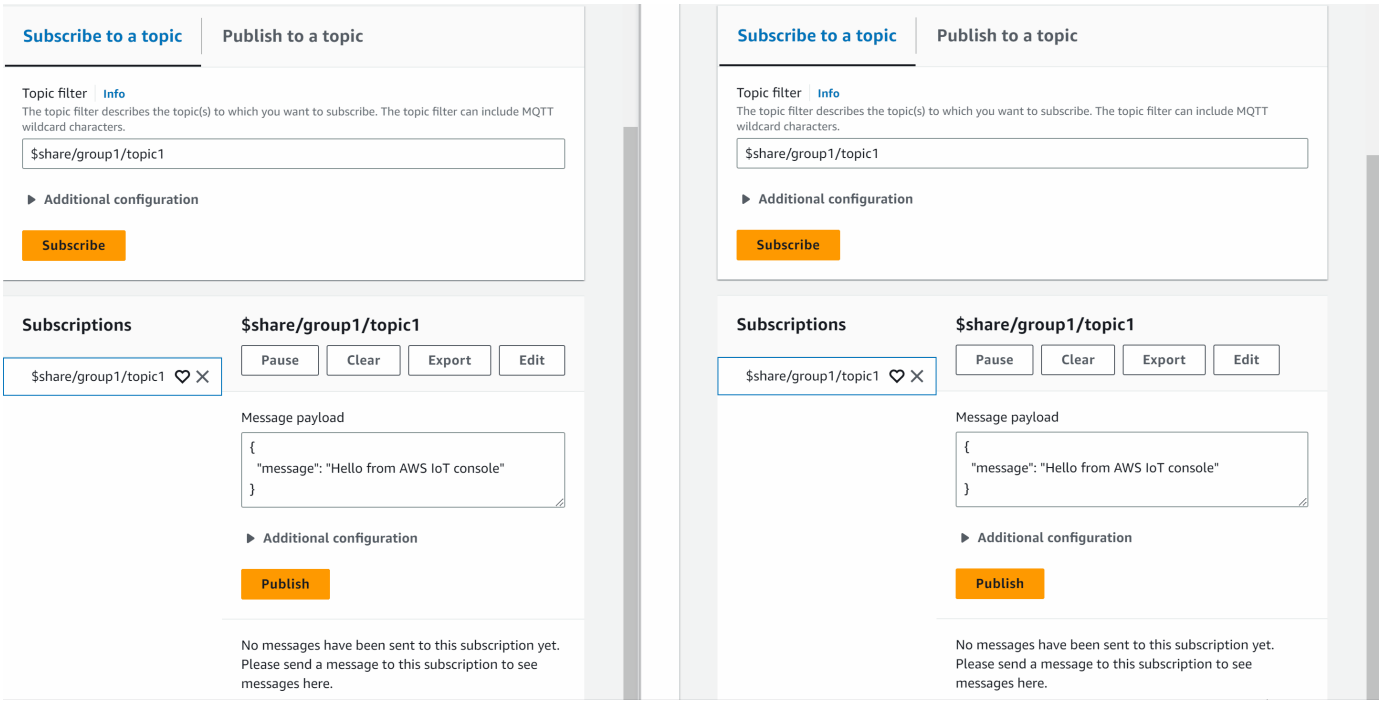
1. 在 AWS IoT 控制台中，在导航窗格中选择测试，然后选择 MQTT 测试客户端。
2. 在 Subscribe to a topic（订阅主题）选项卡中，输入*topicName*订阅您的设备发布的主题。要使用共享订阅，请按如下方式订阅共享订阅的主题筛选条件：

   ```
   $share/{ShareName}/{TopicFilter}
   ```

   示例主题筛选条件可以是 **$share/group1/topic1**，它订阅消息主题 **topic1**。

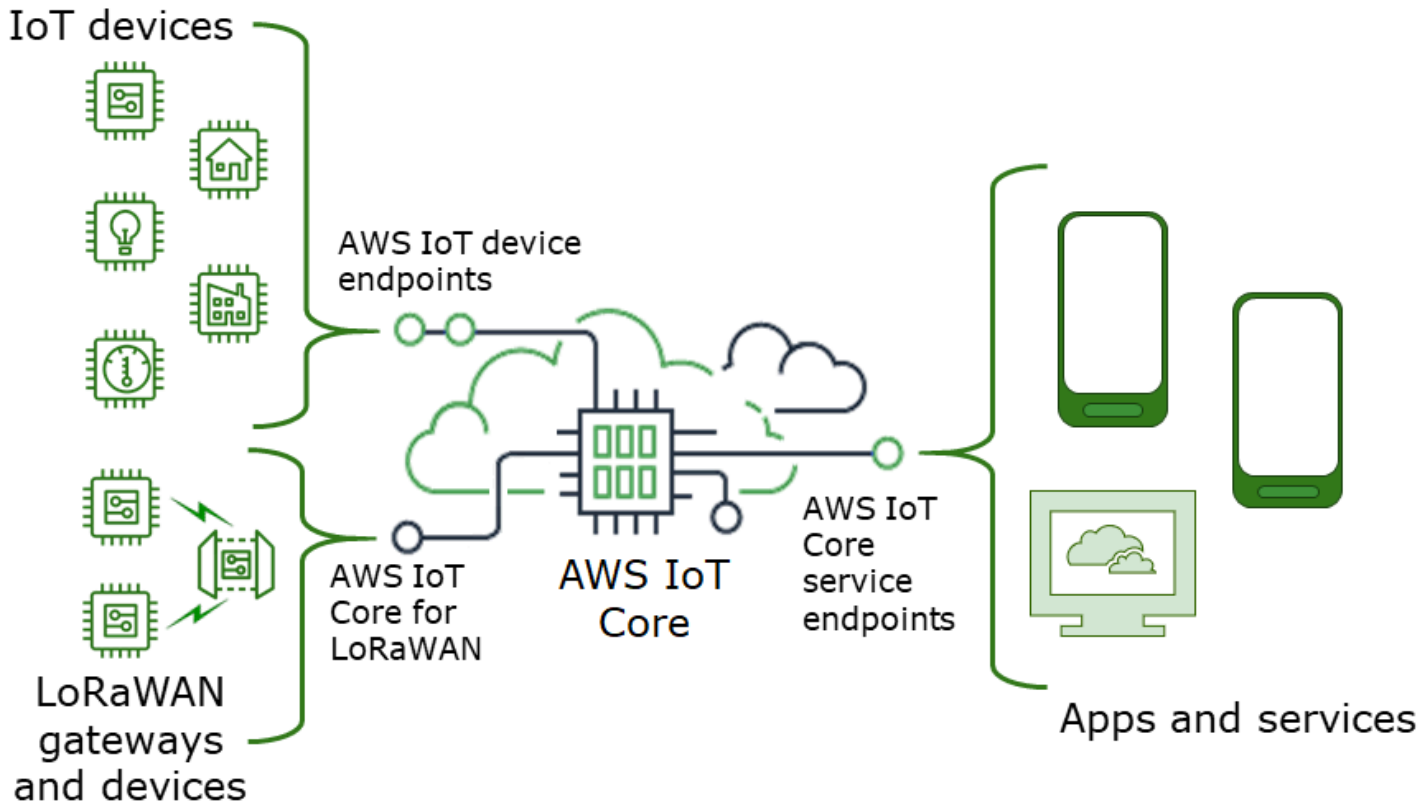
   

3. 打开另一个 Web 浏览器，然后重复步骤 1 和步骤 2。通过这种方式，您可以模拟共享相同订阅 **$share/group1/topic1** 的两个不同的 MQTT 客户端。
4. 选择一个 MQTT 客户端，在发布到主题选项卡上的主题名称字段中，输入消息的 *topicName*。在此示例中，使用 **topic1**。尝试发布消息几次。从两个 MQTT 客户端的订阅列表中，您应该能够看到客户端使用随机分布接收消息。在此示例中，我们发布了三次相同的消息“来自 AWS IoT 控制台的 Hello”。左侧的 MQTT 客户端收到消息两次，右侧的 MQTT 客户端收到消息一次。


Subscribe to a topic
Publish to a topic
Topic filter Info
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.
$share/group1/topic1
Additional configuration
Subscribe
Subscriptions
$share/group1/topic1
$share/group1/topic1
Pause
Clear
Export
Edit
Message payload
{
"message": "Hello from AWS IoT console"
}
Additional configuration
Publish
No messages have been sent to this subscription yet. Please send a message to this subscription to see messages here.

正在连接到 AWS IoT Core

AWS IoT Core 支持与 IoT 设备、无线网关、服务和应用程序的连接。设备可以连接到服务和其他设备 ， AWS IoT Core 这样它们就可以向 AWS IoT 服务和其他设备发送数据和从中接收数据。应用程序和其他服务还可以连接 AWS IoT Core 以控制和管理物联网设备，并处理来自您的物联网解决方案的数据。本节介绍如何为物联网解决方案的各个方面选择最佳的连接和通信方式。 AWS IoT Core



有几种互动方式 AWS IoT。AWS IoT Core 对于 LoRa广域网区域和终端节点，应用程序AWS IoT Core - 控制面板端点和服务可以使用，设备可以使用AWS IoT 设备端点或连接到。 AWS IoT Core

AWS IoT Core - 控制面板端点

控制平面端点提供对控制和管理 AWS IoT 解决方案的功能的访问。AWS IoT Core

- 终端节点

  AWS IoT Core - 控制面板和 AWS IoT Core Device Advisor 控制面板终端节点特定于区域，并列出在 AWS IoT Core 终端节点和配额中。终端节点的格式如下。

终端节点用途	终端节点格式	供应
AWS IoT Core - 控制面板	iot.*aws-region* .amazonaws.com	AWS IoT 控制平面 API
AWS IoT Core 设备顾问-控制平面	api.iotdeviceadvisor. *aws-region* .amazonaws.com	AWS IoT Core 设备顾问控制平面 API

- SDK 和工具

  这些AWS 软件开发工具包为 API 和其他服务的 AWS IoT Core API 提供特定语言的支持。 AWS AWS 移动 SDK 为应用程序开发者提供针对特定平台的 AWS IoT Core API 支持，以及移动设备上的其他 AWS 服务。

  AWS CLI提供对 AWS IoT 服务端点提供的功能的命令行访问权限。AWS T@@ ools for PowerShell 提供了在 PowerShell 脚本环境中管理 AWS 服务和资源的工具。

- 身份验证

  服务终端节点使用 IAM 用户和 AWS 证书对用户进行身份验证。

- 了解更多信息

  有关更多信息和 SDK 参考链接，请参阅 the section called “连接到 AWS IoT Core 服务端点”。

AWS IoT 设备端点

AWS IoT 设备端点支持您的物联网设备与之间的通信 AWS IoT。

- 端点

  设备端点的支持 AWS IoT Core 和 AWS IoT Device Management 功能。它们是你特有的 AWS 账户 ，你可以使用describe-endpoint命令来查看它们是什么。

终端节点用途	终端节点格式	供应
AWS IoT Core - 数据层面	请参阅 ???。	AWS IoT 数据平面 API

终端节点用途	终端节点格式	供应
AWS IoT Device Management - 任务数据	请参阅 ???。	AWS IoT 作业数据面板 API
AWS IoT 设备顾问-数据平面	请参阅 ???。	不适用
AWS IoT Device Management - Fleet Hub	不适用	不适用
AWS IoT Device Management - 安全隧道	`api.tunneling.iot.` *`aws-region`* `.amazonaws.com`	AWS IoT 安全隧道 API

有关这些终端节点及其支持的功能的更多信息，请参阅 the section called “AWS IoT 设备数据和服务端点”。

- SDK

  AWS IoT 设备软件开发工具包为设备用于通信的消息队列遥测传输 (MQTT) 和 WebSocket 安全 (WSS) 协议提供特定语言的支持。 AWS IoTAWS 移动 SDK还为移动设备上的 MQTT 设备通信、AWS IoT API 和其他 AWS 服务的 API 提供支持。

- 身份验证

  设备终端节点使用 X.509 证书或带有证书 AWS 的 IAM 用户对用户进行身份验证。

- 了解更多信息

  有关更多信息和 SDK 参考链接，请参阅 the section called “AWS IoT 设备软件开发工具包”。

AWS IoT Core 适用于 LoRa WAN 网关和设备

AWS IoT Core 用于 LoRa WAN 可将无线网关和设备连接到 AWS IoT Core。

- 端点

  AWS IoT Core fo LoRa r WAN 管理与账户和区域特定终端 AWS IoT Core 节点的网关连接。网关可以连接到您账户的 LoRa WAN 提供的配置和更新服务器 (CUPS) 端点。 AWS IoT Core

终端节点用途	终端节点格式	供应
Configuration and Update Server (CUPS)	*account-specific-prefix* .cups.lorawan. *aws-region* .amazonaws.com:443	与 LoRa WAN 提供的配置和更新服务器 AWS IoT Core 进行网关通信
LoRa广域网网络服务器 (LNS)	*account-specific-prefix* .gateway.lorawan. *aws-region* .amazonaws.com:443	与 LoRa WAN 提供的广域网网络服务器 AWS IoT Core 进行 LoRa网关通信

- 软件开发工具包

  AWS SDK 支持构建 LoRa广域网的 AWS IoT 无线 API。 AWS IoT Core 有关更多信息，请参阅 AWS SDK 和工具包。

- 身份验证

  AWS IoT Core 对于 LoRa WAN 设备通信，使用 X.509 证书来保护与的通信。 AWS IoT

- 了解更多信息

  有关配置和连接无线设备的更多信息，AWS IoT Core 请参阅 LoRa WAN 区域和终端。

连接到 AWS IoT Core 服务端点

您可以通过使用首选语言的AWS IoT Core AWS SDK 或直接调用 REST API 来访问控制平面的功能。AWS CLI我们建议使用 AWS CLI 或 S AWS DK 进行交互， AWS IoT Core 因为它们包含了调用 AWS 服务的最佳实践。直接调用 REST API 是一种选择，但您必须提供必要的安全凭证，以便能够访问 API。

> **Note**
>
> IoT 设备应该使用 AWS IoT 设备软件开发工具包。设备 SDK 针对设备使用进行了优化，支持与 MQTT 通信 AWS IoT，并支持设备最常使用 AWS IoT 的 API。有关 Device SDK 及其提供的特征的更多信息，请参阅 AWS IoT 设备软件开发工具包。

移动设备应该使用 AWS 移动 SDK。移动软件开发工具包支持 AWS IoT API、MQTT 设备通信以及移动设备上其他 AWS 服务的 API。有关 Mobile SDK 及其提供的特征的更多信息，请参阅 AWS 移动 SDK。

您可以使用 Web 和移动应用程序中的 AWS Amplify 工具和资源更轻松地连接 AWS IoT Core。有关使用 Amplify 连接 AWS IoT Core 的更多信息，请参阅 Amplify 文档中的 Pub Sub 入门。

以下各节介绍可用于开发和与之交互的工具和 SDK AWS IoT 以及其他 AWS 服务。有关可用于构建和管理应用程序的 AWS 工具和开发套件的完整列表 AWS，请参阅构建工具 AWS。

AWS CLI 对于 AWS IoT Core

AWS CLI 提供对 API 的命令行访问权限。 AWS

- 安装

  有关如何安装的信息 AWS CLI，请参阅安装 AWS CLI。

- 身份验证

  AWS CLI 使用您的证书 AWS 账户。

- 参考

  有关这些 AWS IoT Core 服务的 AWS CLI 命令的信息，请参阅：

  - AWS CLI 物联网命令参考
  - AWS CLI 物联网数据的命令参考
  - AWS CLI 物联网作业数据的命令参考
  - AWS CLI 物联网安全隧道的命令参考

有关在 PowerShell 脚本环境中管理 AWS 服务和资源的工具，请参阅适用于的AWS 工具 PowerShell。

AWS 软件开发工具包

借 AWS 助 SDK，您的应用程序和兼容设备可以调用 AWS IoT API 和其他 AWS 服务的 API。本节提供指向 AWS 软件开发工具包和 AWS IoT Core 服务 API 的 API 参考文档的链接。

AWS 软件开发工具包支持这些 API AWS IoT Core

- AWS IoT
- AWS IoT 数据平面
- AWS IoT 乔布斯数据平面
- AWS IoT 安全隧道
- AWS IoT 无线

C++

要安装 AWS SDK for C++ 并使用它连接到 AWS IoT：

1. 按照入门使用适用于 C++ 的 AWS SDK 中的说明进行操作

   这些说明描述了如何：

   - 从源文件安装和构建 SDK
   - 提供凭证以通过您的 AWS 账户使用 SDK
   - 在应用程序或服务中初始化和关闭 SDK
   - 创建 CMake 项目以构建应用程序或服务
2. 创建和运行示例应用程序。有关使用 AWS SDK for C++ 的示例应用程序，请参阅AWS SDK for C++ 代码示例。

所 AWS SDK for C++ 支持的 AWS IoT Core 服务的文档

- AWS::IoTClient" 参考文档
- Aws:: IoTDataPlane:: IoT 参考文档 DataPlaneClient
- Aws:: IoTJobsDataPlane:: IoT 参考文档 JobsDataPlaneClient
- Aws:: IoTSecureTunneling:: IoT 参考文档 SecureTunnelingClient

Go

要安装 AWS SDK for Go 并使用它连接到 AWS IoT：

1. 按照《入门》中的说明进行操作 AWS SDK for Go

   这些说明描述了如何：

- 安装 AWS SDK for Go
- 获取 SDK 的访问密钥以访问您的 AWS 账户
- 将程序包导入到我们的应用程序或服务的源代码中

2. 创建和运行示例应用程序。如需使用 AWS SDK for Go的示例应用程序，请参阅 AWS SDK for Go 代码示例。

所 AWS SDK for Go 支持的 AWS IoT Core 服务的文档

- IoT 参考文档
- 物联网DataPlane 参考文档
- 物联网JobsDataPlane 参考文档
- 物联网SecureTunneling 参考文档

Java

要安装 AWS SDK for Java 并使用它连接到 AWS IoT：

1. 按照《入门》中的说明进行操作 AWS SDK for Java 2.x

   这些说明描述了如何：

   - 注册 AWS 并创建 IAM 用户
   - 下载 SDK
   - 设置 AWS 凭证和区域
   - 将 SDK 与 Apache Maven 结合使用
   - 将 SDK 与 Gradle 结合使用
2. 使用 AWS SDK for Java 2.x 代码示例之一创建和运行示例应用程序。
3. 查看 SDK API 参考文档

所 AWS SDK for Java 支持的 AWS IoT Core 服务的文档

- IotClient 参考文档
- IotDataPlaneClient 参考文档
- IotJobsDataPlaneClient 参考文档
- 物联网SecureTunnelingClient 参考文档

JavaScript

要安装 AWS SDK for JavaScript 并使用它连接到 AWS IoT：

1. 按照 AWS SDK for JavaScript设置中的说明操作。这些说明适用于在浏览器 AWS SDK for JavaScript 中使用，也适用于在 Node.JS 中使用。确保按照适用于您的安装的说明进行操作。

   这些说明描述了如何：

   - 检查先决条件
   - 安装适用于 JavaScript
   - 加载适用的 SDK JavaScript
2. 如环境的入门选项所述，创建并运行示例应用程序以开始使用 SDK。
   - 开始使用浏览器 JavaScript 中的 S AWS DK，或者
   - 开始使用 Node.js JavaScript 中的 S AWS DK

所 AWS SDK for JavaScript 支持的 AWS IoT Core 服务的文档

- AWS.Iot reference documentation
- AWS.IotData reference documentation
- AWS.IotJobsDataPlane reference documentation
- AWS.IotSecureTunneling reference documentation

.NET

要安装 AWS SDK for .NET 并使用它连接到 AWS IoT：

1. 按照设置 AWS SDK for .NET 环境中的说明进行操作
2. 按照设置 AWS SDK for .NET 项目中的说明进行操作

   这些说明描述了如何：

   - 启动新项目
   - 获取和配置 AWS 凭证
   - 安装 AWS SDK 软件包
3. 在 For .NET 的 AWS SDK 中使用 AWS 服务中创建并运行其中一个示例程序
4. 查看 SDK API 参考文档

所 AWS SDK for .NET 支持的 AWS IoT Core 服务的文档

- Amazon.IoT.Model 参考文档
- 亚马逊。 IotData.Model 参考文档
- Amazon.iot .Model JobsDataPlane 参考文档
- Amazon.iot .Model SecureTunneling 参考文档

PHP

要安装 AWS SDK for PHP 并使用它连接到 AWS IoT：

1. 按照AWS SDK for PHP 版本 3 入门中的说明进行操作

   这些说明描述了如何：
   - 检查先决条件
   - 安装 SDK
   - 将 SDK 应用于 PHP 脚本
2. 使用 AWS SDK for PHP 版本 3 代码示例之一创建和运行示例应用程序

所 AWS SDK for PHP 支持的 AWS IoT Core 服务的文档

- IoTClient 参考文档
- 物联网DataPlaneClient 参考文档
- 物联网JobsDataPlaneClient 参考文档
- 物联网SecureTunnelingClient 参考文档

Python

要安装 AWS SDK for Python (Boto3) 并使用它连接到 AWS IoT：

1. 按照 AWS SDK for Python (Boto3) 快速入门中的说明操作

   这些说明描述了如何：
   - 安装 SDK
   - 配置 SDK
   - 在您的代码中使用 SDK

2. 创建并运行使用 AWS SDK for Python (Boto3)的示例程序

   此程序显示账户当前配置的日志记录选项。安装 SDK 并为您的账户配置该包后，您应该能够运行此程序。

```
import boto3
import json

# initialize client
iot = boto3.client('iot')

# get current logging levels, format them as JSON, and write them to stdout
response = iot.get_v2_logging_options()
print(json.dumps(response, indent=4))
```

   有关此示例中使用的功能的更多信息，请参阅the section called “配置 AWS IoT 日志”。

所 AWS SDK for Python (Boto3) 支持的 AWS IoT Core 服务的文档

- IoT 参考文档
- 物联网DataPlane 参考文档
- 物联网JobsDataPlane 参考文档
- 物联网SecureTunneling 参考文档

Ruby

要安装 AWS SDK for Ruby 并使用它连接到 AWS IoT：

- 按照《入门》中的说明进行操作 AWS SDK for Ruby

  这些说明描述了如何：

  - 安装 SDK
  - 配置 SDK
- 创建并运行 Hello World 教程

适用于 Ruby 的 AWS SDK 所支持的 AWS IoT Core 服务的文档

- Aws::IoT::Client 参考文档

- Aws:: IoTDataPlane:: Client 参考文档
- Aws:: IoTJobsDataPlane:: Client 参考文档
- Aws:: IoTSecureTunneling:: Client 参考文档

AWS 移动 SDK

AWS 移动软件开发工具包为移动应用开发者提供特定平台的支持，包括服务的 API AWS IoT Core 、使用 MQTT 的物联网设备通信以及其他服务的 API。 AWS

Android

AWS Mobile SDK for Android

AWS Mobile SDK for Android 包含一个库、示例和文档，供开发人员用来构建互联的移动应用程序 AWS。此 SDK 还包括对 MQTT 设备通信和调用 AWS IoT Core 服务 API 的支持。有关更多信息，请参阅下列内容：

- AWS 适用于 Android 的移动 SDK GitHub
- AWS 适用于 Android 的移动 SDK 自述文件
- AWS 适用于 Android 的移动 SDK 示例
- AWS 适用于 Android 的 SDK API 参考
- AWSIoTClient 类参考文档

iOS

AWS Mobile SDK for iOS

AWS Mobile SDK for iOS 是一个开源软件开发套件，在 Apache 开源许可证下分发。iOS 版 SDK 提供了一个库、代码示例和文档，以帮助开发者使用构建互联的移动应用程序 AWS。此 SDK 还包括对 MQTT 设备通信和调用 AWS IoT Core 服务 API 的支持。有关更多信息，请参阅下列内容：

- AWS Mobile SDK for iOS on GitHub
- AWS 适用于 iOS 的 SDK 自述文件
- AWS 适用于 iOS 的 SDK 示例
- AWS IoT iOS 版 AWS SDK 中的类参考文档

AWS IoT Core 服务的 REST API

可以使用 HTTP 请求直接调用这些 AWS IoT Core 服务的 REST API。

- 端点 URL

  公开 AWS IoT Core 服务的 REST API 的服务终端节点因区域而异，并在 AWS IoT Core 终端节点和配额中列出。您必须使用具有您要访问的 AWS IoT 资源的区域的终端节点，因为 AWS IoT 资源是特定于区域的。

- 身份验证

  这些 AWS IoT Core 服务的 REST API 使用 AWS IAM 凭证进行身份验证。有关更多信息，请参阅《 AWS 通用参考》中的 “签署 AWS API 请求”。

- API 参考

  有关 AWS IoT Core 服务的 REST API 提供的特定功能的信息，请参阅：

  - IoT 的 API 参考。
  - IoT 数据的 API 参考。
  - IoT 任务数据的 API 参考。
  - IoT 安全隧道的 API 参考。

将设备连接到 AWS IoT

设备连接到 AWS IoT ，其他服务则通过 AWS IoT Core。通过 AWS IoT Core，设备使用特定于您账户的设备端点发送和接收消息。the section called “AWS IoT 设备软件开发工具包” 支持使用 MQTT 和 WSS 协议进行的设备通信。有关设备可以使用的协议的更多信息，请参阅the section called “设备通信协议”。

消息代理

AWS IoT 通过消息代理管理设备通信。设备和客户端向消息代理发布消息，并且还订阅消息代理发布的消息。消息由应用程序定义的主题 进行标识。当消息代理收到由设备或客户端发布的消息时，它会将该消息重新发布到订阅了该消息主题的设备和客户端。消息代理还将消息转发到 AWS IoT 规则引擎，规则引擎可以根据消息的内容采取行动。

AWS IoT 消息安全

要 AWS IoT 使用的设备连接the section called “X.509 客户端证书”和用于身份验证的 V4 AWS 签名。设备通信受 TLS 版本 1.3 的保护，并且 AWS IoT 要求设备在连接时发送服务器名称指示 (SNI) 扩展名。有关更多信息，请参阅中的传输安全 AWS IoT。

AWS IoT 设备数据和服务端点

> **Important**
>
> 您可以在设备中缓存或存储端点。这意味着您无需在每次连接新设备时查询 `DescribeEndpoint` API。在为您的账户 AWS IoT Core 创建终端节点后，终端节点不会更改。

每个账户都有几个设备终端节点，这些终端节点对账户是唯一的，并支持特定的 IoT 功能。 AWS IoT 设备数据端点支持专为满足物联网设备通信需求而设计的发布/订阅协议；但是，如果其他客户端（例如应用程序和服务）的应用程序需要这些端点提供的专用功能，也可以使用此接口。 AWS IoT 设备服务端点支持以设备为中心访问安全和管理服务。

要了解您账户的设备数据端点，可以在 AWS IoT Core 主机的 “设置” 页面中找到它。

要了解您账户的特定用途的设备终端节点（包括设备数据终端节点），请使用此处显示的 describe-endpoint CLI 命令或 `DescribeEndpoint` REST API，并提供下表中的 *`endpointType`* 参数值。

```
aws iot describe-endpoint --endpoint-type endpointType
```

此命令采用以下格式返回 *`iot-endpoint`*：*`account-specific-prefix`*`.iot.`*`aws-region`*`.amazonaws.com`。

每个客户都有一个 `iot:Data-ATS` 和一个 `iot:Data` 终端节点。每个终端节点都使用 X.509 证书对客户端进行身份验证。我们强烈建议客户使用较新的 `iot:Data-ATS` 终端节点类型，以避免出现与 Symantec 证书颁发机构普遍不信任有关的问题。我们为设备提供`iot:Data`端点，用于从使用 VeriSign 证书实现向后兼容的旧端点检索数据。有关更多信息，请参阅服务器身份验证。

AWS IoT 设备端点

终端节点用途	***`endpointType`*** 值	描述
AWS IoT Core - 数据层面操作	`iot:Data-ATS`	用于与消息代理、Device Shadow 和 AWS IoT的规则引擎组件之间收发数据。

终端节点用途	*endpointType* 值	描述
		`iot:Data-ATS` 返回 ATS 签名的数据终端节点。
AWS IoT Core - 数据层面操作（旧版）	`iot:Data`	`iot:Data`返回为向后兼容而提供的 VeriSign 签名数据端点。Symantec (`iot:Data`) 端点不支持 MQTT 5。
AWS IoT Core 凭证访问权限	`iot:CredentialProvider`	用于将设备的内置 X.509 证书交换为临时凭证，以直接与其它 AWS 服务连接。有关连接到其他 AWS 服务的更多信息，请参阅授权直接调用 AWS 服务。
AWS IoT Device Management - 任务数据操作	`iot:Jobs`	用于使设备能够使用作业设备 HTTPS API 与 AWS IoT 作业服务进行交互。
AWS IoT 设备顾问操作	`iot:DeviceAdvisor`	用于使用 Device Advisor 测试设备的测试终端节点类型。有关更多信息，请参阅 ???。
AWS IoT Core 数据测试版（预览版）	`iot:Data-Beta`	为测试版预留的终端节点类型。有关当前用法的信息，请参阅 ???。

您也可以使用自己的完全限定域名 (FQDN)（*## example.com*）和关联的服务器证书来连接设备。AWS IoT the section called “可配置的终端节点”

AWS IoT 设备软件开发工具包

AWS IoT 设备软件开发工具包可帮助您连接物联网设备， AWS IoT Core 并且它们支持通过 WSS 协议的 MQTT 和 MQTT。

AWS IoT 设备软件开发工具包与 AWS 软件开发工具包的不同之处在于， AWS IoT 设备软件开发工具包支持物联网设备的特殊通信需求，但不支持软件开发工具包支持的所有服务。 AWS AWS IoT 设备

软件开发工具包与支持所有 AWS 服务的 AWS 软件开发工具包兼容；但是，它们使用不同的身份验证方法并连接到不同的端点，这可能会使在物联网设备上使用 AWS 软件开发工具包变得不切实际。

移动设备

the section called “AWS 移动 SDK”支持 MQTT 设备通信、部分 AWS IoT 服务 API 和其他 AWS 服务的 API。如果您在受支持的移动设备上进行开发，请查看其 SDK，了解它是否是开发 IoT 解决方案的最佳选择。

C++

AWS IoT C++ 设备开发工具包

C AWS IoT ++ 设备 SDK 允许开发人员使用 AWS IoT Core 服务 AWS 的 API 构建互联应用程序。特别是，此 SDK 面向没有资源限制且需要高级特征（例如，消息队列、多线程支持和最新的语言特征）的设备而设计。有关更多信息，请参阅下列内容：

- AWS IoT 设备 SDK C++ v2 已开启 GitHub
- AWS IoT 设备 SDK C++ v2 自述文件
- AWS IoT 设备 SDK C++ v2 示例
- AWS IoT 设备 SDK C++ v2 API 文档

Python

AWS IoT Python 设备软件开发工具包

适用于 Python 的 AWS IoT 设备 SDK 使开发人员能够编写 Python 脚本，以便使用他们的设备通过 WebSocket 安全 (WSS) 协议通过 MQTT 或 MQTT 访问 AWS IoT 平台。通过将设备连接到服务的 API，用户可以安全地使用消息代理、规则和 AWS IoT Core 提供其他 AWS IoT Core 服务（例如 Amazon Kinesis 和 Amazon S3 AWS Lambda等）的 Device Shadow AWS 服务。

- AWS IoT 适用于 Python v2 的设备 SDK 已开启 GitHub
- AWS IoT 适用于 Python v2 的设备 SDK 自述文件
- AWS IoT 适用于 Python v2 的设备软件开发工具包示例
- AWS IoT 适用于 Python 的设备软件开发工具包 v2 API 文档

JavaScript

AWS IoT 适用于的设备 SDK JavaScript

AWS IoT 设备软件开发工具包 JavaScript 使开发人员可以编写 JavaScript 应用程序，通过协议访问 AWS IoT Core 使用 MQTT 或 MQTT 的 API。 WebSocket 它可用于 Node.js 环境和浏览器应用程序。有关更多信息，请参阅下列内容：

- AWS IoT 适用于 JavaScript v2 的设备 SDK 已开启 GitHub
- AWS IoT 适用于 JavaScript v2 的设备 SDK 自述文件
- AWS IoT 适用于 JavaScript v2 示例的设备 SDK
- AWS IoT 适用于 JavaScript v2 的设备 SDK API 文档

Java

AWS IoT 适用于 Java 的设备 SDK

适用于 Java 的 AWS IoT 设备 SDK 使 Java 开发人员能够 AWS IoT Core 通过协议通过 MQTT 或 MQTT 访问的 API。 WebSocket该 SDK 支持 Device Shadow 服务。您可以使用 HTTP 方法 (包括 GET、UPDATE 和 DELETE) 访问影子。该 SDK 还支持简化的影子访问模型，开发人员可以使用 getter 和 setter 方法与影子交换数据，而不必对任何 JSON 文档进行序列化或反序列化。有关更多信息，请参阅下列内容：

- AWS IoT 适用于 Java v2 的设备 SDK 已开启 GitHub
- AWS IoT 适用于 Java 的设备 SDK v2 自述文件
- AWS IoT 适用于 Java 的设备 SDK v2 示例
- AWS IoT 适用于 Java 的设备 SDK v2 API 文档

Embedded C

AWS IoT 适用于嵌入式 C 的设备 SDK

> **Important**
>
> 该 SDK 供经验丰富的嵌入式软件开发人员使用。

AWS IoT Device SDK for Embedded C (C-SDK) 是 MIT 开源许可下的 C 源文件集合，可用于嵌入式应用程序，将物联网设备安全地连接到 IoT Core。 AWS 它包括 MQTT、JSON 解析 AWS IoT 器和 Device Shadow 库等。它以源代码的形式分发，用于构建到客户固件和应用程序代码、其它库，以及（可选）RTOS（实时操作系统）中。

AWS IoT Device SDK for Embedded C 通常针对需要优化 C 语言运行时的资源受限的设备。您可以在任何操作系统上使用此 SDK，并将其托管在任何类型的处理器（例如 MCU 和 MPU）上。如果您的设备有足够的内存和处理资源，我们建议您使用其他 AWS IoT 设备和移动软件开发工具包，例如适用于 C++ JavaScript、Java 或 Python 的 AWS IoT 设备 SDK。

有关更多信息，请参阅下列内容：

- AWS IoT 适用于嵌入式 C 的设备 SDK 已开启 GitHub
- AWS IoT 适用于嵌入式 C 的设备 SDK 自述文件
- AWS IoT 嵌入式 C 示例的设备 SDK

设备通信协议

AWS IoT Core 支持使用 MQTT 和 MQTT over S WebSocket ecure (WSS) 协议发布和订阅消息的设备和客户端，以及使用 HTTPS 协议发布消息的设备和客户端。所有协议都支持 IPv4 和 IPv6。本节介绍设备和客户端的不同连接选项。

TLS 1.2 和 TLS 1.3

AWS IoT Core 使用 TLS 版本 1.2 和 TLS 版本 1.3 来加密所有通信。将设备连接到 AWS IoT Core 时，客户端可以发送服务器名称指示（SNI）扩展，这不是必需的，但强烈建议您这样做。要使用多账户注册、自定义域和 VPC 端点等特征，必须使用 SNI 扩展。有关更多信息，请参阅中的传输安全 AWS IoT。

AWS IoT 设备软件开发工具包 支持 MQTT 和基于 WSS 的 MQTT，并支持客户端连接的安全要求。我们建议使用 AWS IoT 设备软件开发工具包 将客户端连接到 AWS IoT。

协议、端口映射和身份验证

设备或客户端如何使用设备终端节点连接到消息代理，取决于它使用的协议。下表列出了 AWS IoT 设备端点支持的协议及其使用的身份验证方法和端口。

协议、身份验证和端口映射

协议	支持的操作	身份验证	端口	ALPN 协议名称
MQTT 结束了 WebSocket	发布、订阅	Signature Version 4	443	不适用

协议	支持的操作	身份验证	端口	ALPN 协议名称
MQTT 结束了 WebSocket	发布、订阅	自定义身份验证	443	不适用
MQTT	发布、订阅	X.509 客户端证书	443†	`x-amzn-mqtt-ca`
MQTT	发布、订阅	X.509 客户端证书	8883	不适用
MQTT	发布、订阅	自定义身份验证	443†	`mqtt`
HTTPS	仅发布	Signature Version 4	443	不适用
HTTPS	仅发布	X.509 客户端证书	443†	`x-amzn-http-ca`
HTTPS	仅发布	X.509 客户端证书	8443	不适用
HTTPS	仅发布	自定义身份验证	443	不适用

应用程序层协议协商 (ALPN)

† 通过 X.509 客户端证书身份验证在端口 443 上连接的客户端必须实现应用层协议协商 (ALPN) TLS 扩展，并使用客户端作为消息一部分 ProtocolNameList 发送的 AP N 中列出的 ALPN 协议名称。`ClientHello`
在端口 443 上，Iot: data-ats 端点支持 ALPN x-amzn-http-ca HTTP，但是 IoT: Jobs 端点不支持。
在端口 8443 HTTPS 和带有 ALPN 的 443 MQTT 端口上 x-amzn-mqtt-ca，无法使用自定义身份验证。

客户端连接到其 AWS 账户设备端点。有关如何查找您账户的设备终端节点的信息，请参阅 the section called “AWS IoT 设备数据和服务端点”。

> **Note**
>
> AWS 软件开发工具包不需要完整的 URL。它们只需要端点主机名，例如适用于 Python 的 AWS IoT 设备 SDK 的pubsub.py示例 GitHub。如下表中所示传递整个 URL 可能会生成错误，例如主机名无效。

正在连接到 AWS IoT Core

协议	终端节点或 URL
MQTT	*iot-endpoint*
基于 WSS 的 MQTT	wss://*iot-endpoint* /mqtt
HTTPS	https://*iot-endpoint* /topics

为设备通信选择协议

对于大多数通过设备终端节点进行的 IoT 设备通信，您将希望使用 MQTT 或基于 WSS 的 MQTT 协议；但是，设备终端节点也支持 HTTPS。下表比较了如何 AWS IoT Core 使用这两种协议进行设备通信。

AWS IoT 设备协议 side-by-side

功能	MQTT	HTTPS
发布/订阅支持	发布和订阅	仅发布
SDK 支持	AWS 设备软件开发工具包支持 MQTT 和 WSS 协议	不支持 SDK，但您可以使用特定于语言的方法来发出 HTTPS 请求
服务质量支持	MQTT QoS 级别 0 和 1	通过传递查询字符串参数？qos=qos 来支持 QoS，其值可以是 0 或 1。您可以添加此查询字符串,以发布具有所需 QoS 值的消息。

功能	MQTT	HTTPS
可以接收设备离线时错过的消息	支持	不支持
clientId 现场支持	支持	不支持
设备断开检测	支持	不支持
安全通信	是。请参阅 协议、端口映射和身份验证。	是。请参阅 协议、端口映射和身份验证。
主题定义	定义的应用程序	定义的应用程序
消息数据格式	定义的应用程序	定义的应用程序
协议开销	小于	更高
功耗	小于	更高

连接持续时间限制

HTTPS 连接的持续时间不一定能超过接收和响应请求所需的时间。

MQTT 连接持续时间取决于您所使用的身份验证特征。下表列出了每个特征在理想条件下的最长连接持续时间。

按身份验证特征列出 MQTT 连接持续时间

特征	最长持续时间 *
X.509 客户端证书	1-2 周
自定义身份验证	1-2 周
Signature Version 4	长达 24 小时

*无法保证

使用 X.509 证书和自定义身份验证，则连接持续时间没有硬性限制，但最短可能只有几分钟。导致连接中断的原因多种多样。以下列表包含一些最常见的原因。

- Wi-Fi 可用性中断
- 互联网服务提供商 (ISP) 连接中断
- 服务修补
- 服务部署
- 服务自动扩展
- 服务主机不可用
- 负载均衡器问题和更新
- 客户端错误

您的设备必须执行检测断开连接的策略和重新连接的策略。获取有关断开连接事件及其处理方式的指导信息，请参阅 ??? 中的 ???。

MQTT

MQTT（消息队列遥测传输）是一种广泛采用的轻型消息传递协议，专为受限制的设备而设计。 AWS IoT Core 对 MQTT 的支持基于 MQTT v3.1.1 规范和 MQTT v5.0 规范，但如 the section called “AWS IoT 与 MQTT 规格的区别” 中所述，有一些差异。作为此标准的最新版本，MQTT 5 引入了多项使基于 MQTT 的系统更强大的关键特征，包括新的可扩展性增强、通过原因代码响应改进的错误报告、消息和会话到期计时器，以及自定义用户消息标头。有关 AWS IoT Core 支持的 MQTT 5 功能的更多信息，请参阅 MQTT 5 支持的功能。 AWS IoT Core 还支持跨版本 MQTT（MQTT 3 和 MQTT 5）通信。MQTT 3 发布者可以向将接收 MQTT 5 发布消息的 MQTT 5 订阅者发送 MQTT 3 消息，反之亦然。

AWS IoT Core 支持使用 MQTT 协议和基于 WSS 协议的 MQTT 且由客户端 ID 标识的设备连接。AWS IoT 设备软件开发工具包 支持这两种协议，并且是将设备连接到 AWS IoT Core的推荐方法。 AWS IoT 设备软件开发工具包支持设备和客户端连接和访问 AWS IoT 服务所需的功能。设备软件开发工具包支持 AWS IoT 服务所需的身份验证协议以及 MQTT 协议和基于 WSS 的 MQTT 协议所需的连接 ID 要求。有关如何 AWS IoT 使用 AWS 设备软件开发工具包进行连接的信息以及支持的语言示例 AWS IoT 的链接，请参阅the section called “使用 AWS IoT 设备软件开发工具包与 MQTT 连接”。有关 MQTT 消息的身份验证方法和端口映射的更多信息，请参阅协议、端口映射和身份验证。

虽然我们建议使用 AWS IoT 设备软件开发工具包进行连接 AWS IoT，但它们不是必需的。但是，如果您不使用 AWS IoT 设备软件开发工具包，则必须提供必要的连接和通信安全性。客户端必须在连接请求中发送服务器名称指示 (SNI) TLS 扩展。不包括 SNI 的连接尝试将被拒绝。有关更多信息，请参

阅中的传输安全 AWS IoT。使用 IAM 用户和 AWS 证书对客户端进行身份验证的客户端必须提供正确的签名版本 4 身份验证。

本主题内容：

- 使用 AWS IoT 设备软件开发工具包与 MQTT 连接
- MQTT 服务质量 (QoS) 选项
- MQTT 持久性会话
- MQTT 保留消息
- MQTT Last Will and Testament（LWT）消息
- 使用 ConnectAttributes
- MQTT 5 支持的特征
- MQTT 5 属性
- MQTT 原因代码
- AWS IoT 与 MQTT 规格的区别

使用 AWS IoT 设备软件开发工具包与 MQTT 连接

本节包含指向 AWS IoT 设备软件开发工具包的链接，以及说明如何将设备连接到 AWS IoT的示例程序的源代码的链接。此处链接的示例应用程序显示了如何 AWS IoT 使用 MQTT 协议和通过 WSS 的 MQTT 进行连接。

> **Note**
>
> AWS IoT 设备软件开发工具包已经发布了 MQTT 5 客户端。

C++

使用 C AWS IoT ++ 设备 SDK 连接设备

- 以 C++ 语言显示 MQTT 连接示例的示例应用程序源代码
- AWS IoT C++ 设备 SDK v2 已开启 GitHub

Python

使用适用于 Python 的 AWS IoT 设备 SDK 连接设备

- 以 Python 语言显示 MQTT 连接示例的示例应用程序源代码
- AWS IoT 适用于 Python v2 的设备 SDK 已开启 GitHub

JavaScript

使用 AWS IoT 设备 SD JavaScript K 连接设备

- 显示了 MQTT 连接示例的示例应用程序的源代码 JavaScript
- AWS IoT 适用于 JavaScript v2 的设备 SDK 已开启 GitHub

Java

使用适用于 Java 的 AWS IoT 设备 SDK 连接设备

> **Note**
>
> 适用于 Java v2 的 AWS IoT 设备 SDK 现在支持安卓开发。如需了解更多信息，请参阅适用于 Android 的AWS IoT 设备 SDK。

- 以 Java 语言显示 MQTT 连接示例的示例应用程序源代码
- AWS IoT 适用于 Java v2 的设备 SDK 已开启 GitHub

Embedded C

使用适用于嵌入式 C 的 AWS IoT 设备 SDK 连接设备

> **Important**
>
> 该 SDK 供经验丰富的嵌入式软件开发人员使用。

- 以嵌入式 C 语言显示 MQTT 连接示例的示例应用程序源代码
- AWS IoT 适用于嵌入式 C 的设备 SDK 已开启 GitHub

MQTT 服务质量 (QoS) 选项

AWS IoT AWS IoT 设备软件开发工具包支持 MQTT 服务质量 (QoS) 级别和。0 1MQTT 协议定义了第三级 QoS，即2级别， AWS IoT 但不支持该级别。只有 MQTT 协议支持 QoS 特征。HTTPS 通过传递查询字符串参数 ?qos=qos 来支持 QoS，其值可以是 0 或 1。

下表描述了每个 QoS 级别如何影响发布到消息代理和由消息代理发布的消息。

QoS 级别为...	消息为...	注释
QoS 级别 0	不发送或发送多次	此级别应该用于通过可靠通信链接发送的消息，或者可以毫无问题地错过的消息。
QoS 级别 1	至少发送一次，然后重复发送，直到收到 PUBACK 响应	在发送方收到指示成功传递的 PUBACK 响应之前，该消息不被认为是完整的。

MQTT 持久性会话

持久性会话存储客户端尚未确认的服务质量（QoS）为 1 的客户端的订阅和消息。当设备重新连接到持久性会话时，该会话恢复，订阅恢复，并将在重新连接之前接收和存储的未确认的订阅消息发送到客户端。

存储消息的处理记录在 CloudWatch 和 CloudWatch 日志中。有关写入 CloudWatch 和 CloudWatch 日志的条目的信息，请参见消息代理指标和排队的日志条目。

创建持久性会话

在 MQTT 3 中，您可以通过发送 CONNECT 消息并将 cleanSession 标记设置为 0 来创建 MQTT 持久性会话。如果发送 CONNECT 消息的客户端不存在会话，则会创建一个新的持久性会话。如果客户端已存在会话，则客户端会恢复现有会话。要创建干净会话，您需要发送一条 CONNECT 消息，并将 cleanSession 标志设置为 1，当客户端断开连接时，代理将不存储任何会话状态。

在 MQTT 5 中，您可以通过设置 Clean Start 标志和 Session Expiry Interval 来处理永久性会话。干净启动控制连接会话的开始和上一会话的结束。当您设置 Clean Start = 1 时，会创建一个新会话，如果之前的会话存在，将终止之前的会话。当您设置 Clean Start = 0 时，连接会话将恢复之前的会话（如果存在之前的会话）。会话过期间隔控制连接会话的结束。会话过期间隔指定断开连接

后会话将持续的时间（以秒为单位，4 字节整数）。设置 Session Expiry interval=0 可使会话在断开连接后立即终止。如果未在 CONNECT 消息中指定会话过期间隔，默认值为 0。

MQTT 5 干净启动和会话过期

属性值	描述
Clean Start= 1	创建新会话并终止之前的会话（如果存在之前的会话）。
Clean Start= 0	如果存在之前的会话，则恢复会话。
Session Expiry Interval> 0	持续会话。
Session Expiry interval= 0	不持续会话。

在 MQTT 5 中，如果设置 Clean Start = 1 和 Session Expiry Interval = 0，这相当于 MQTT 3 清理会话。如果设置 Clean Start = 0 和 Session Expiry Interval > 0，这相当于 MQTT 3 永久会话。

> **Note**
>
> 不支持跨 MQTT 版本（MQTT 3 和 MQTT 5）的持久性会话。无法将 MQTT 3 持久性会话恢复为 MQTT 5 会话，反之亦然。

持久性会话期间的操作

客户端使用连接已确认 (CONNACK) 消息中的 sessionPresent 属性确定是否存在持久性会话。如果 sessionPresent 为 1，则存在持久性会话，并且在客户端收到 CONNACK 后将为客户端存储的所有消息传送给客户端，如重新连接到持久性会话后的消息流量中所述。如果 sessionPresent 是 1，则客户端无需重新订阅。但是，如果 sessionPresent 为 0，则不存在持久性会话，并且客户端必须重新订阅主题筛选条件。

客户端加入持久性会话后，它可以发布消息并订阅主题筛选条件，而无需在每个操作上附加任何标记。

重新连接到持久性会话后的消息流量

持久性会话表示客户端与 MQTT 消息代理之间的持续连接。当客户端使用持久性会话连接到消息代理时，消息代理会保存客户端在连接期间所做的所有订阅。当客户端断开连接时，消息代理将未确认的 QoS 1 消息和发布的新 QoS 1 消息存储到客户端订阅的主题。根据账户限制存储消息。超过限制的消息将被删除。有关持久消息限制的更多信息，请参阅 AWS IoT Core 终端节点和配额。当客户端重新连接到其持久性会话时，将恢复所有订阅，并以每秒 10 条消息的最大速率将所有已存储消息发送到客户端。在 MQTT 5 中，如果具有消息过期间隔的出站 QoS1 在客户端离线时过期，则连接恢复后，客户端将不会收到过期消息。

重新连接后，以每秒限制为 10 条已存储消息的速率将已存储消息与任何当前消息流量一起发送到客户端，直到达到 `Publish requests per second per connection` 限制。由于已存储消息的传递速率受到限制，如果会话在重新连接后有超过 10 条已存储消息要传递，传递所有已存储的消息将需要几秒钟的时间。

结束持久性会话

持久性会话可通过以下方式结束：

- 持久性会话到期时间已过。当消息代理检测到客户端已断开连接（通过客户端断开连接或连接超时）时，持久性会话到期计时器将启动。
- 客户端发送将 `cleanSession` 标记设置为 1 的 `CONNECT` 消息。

在 MQTT 3 中，持久性会话过期时间的默认值为一小时，此设置适用于账户中的所有会话。

在 MQTT 5 中，您可以为 CONNECT 和 DISCONNECT 数据包上的每个会话设置会话过期间隔。

对于 DISCONNECT 数据包上的会话过期间隔：

- 如果当前会话的会话过期间隔为 0，则无法在 DISCONNECT 数据包上将会话过期间隔设置为大于 0。
- 如果当前会话的会话过期间隔大于 0，并且您在 DISCONNECT 数据包上将会话过期间隔设置为 0，会话将在 DISCONNECT 上结束。
- 否则，DISCONNECT 数据包的会话过期间隔将更新当前会话的会话过期间隔。

> **Note**
>
> 会话结束时等待发送给客户端的已存储消息将被丢弃；但是，即使无法发送，也仍按标准消息传递费率计费。有关消息定价的更多信息，请参阅 AWS IoT Core 定价。您可以配置到期时间间隔。

持久性会话到期后重新连接

如果客户端在其持久性会话到期之前未重新连接到该会话，则该会话结束并丢弃其存储的消息。当客户端在会话到期后重新连接并将 `cleanSession` 标记设置为 `0` 时，服务会创建一个新的持久性会话。上一个会话中的任何订阅或消息都不可用于此会话，因为它们在上一个会话到期时被丢弃。

持久性会话消息费用

AWS 账户 当消息代理向客户端或离线持续会话发送消息时，将向您收取消息费用。当具有持久会话的离线设备重新连接并恢复其会话时，存储的消息将传递到设备并再次向您的账户收费。有关消息定价的更多信息，请参阅 AWS IoT Core 定价 - 消息收发。

通过使用标准限制增加过程，可以将默认的持久性会话到期时间延长 1 小时。请注意，增加会话过期时间可能会增加消息费用，因为额外的时间可能允许离线设备存储更多消息，而这些额外的消息费用将按标准消息收发费率计入您的账户。会话到期时间为大约时间，会话的持续时间最多可能比账户限制长 30 分钟；但是，会话时间不会短于账户限制。有关会话限制的更多信息，请参阅 AWS Service Quotas。

MQTT 保留消息

AWS IoT Core 支持 MQTT 协议中描述的 RETAIN 标志。当客户端在其发布的 MQTT 消息上设置 RETAIN 标志时， AWS IoT Core 会保存该消息。然后可以将其发送给新的订阅者，通过调用 `GetRetainedMessage` 操作进行检索，并在 AWS IoT 控制台中查看。

使用 MQTT 保留消息的示例

- 作为初始配置消息

  客户端订阅主题后，MQTT 保留消息会被发送给客户端。如果您希望所有订阅主题的客户端在订阅后立即收到 MQTT 保留的消息，则可以发布设置了 RETAIN 标志的配置消息。只要发布新的配置消息，订阅客户端也会收到该配置更新。
- 作为最后一个已知的状态消息

设备可以在当前状态消息上设置 REATIN 标志，以便 AWS IoT Core 保存它们。当应用程序连接或重新连接时，它们可以订阅此主题，并在订阅保留的消息主题后立即获得上次报告的状态。这样，它们就不必等到设备发出下一条消息才能看到当前状态。

本节内容：

将 MQTT 保留消息存储在 AWS IoT Core中的常见任务

AWS IoT Core 保存设置了 RETAIN 标志的 MQTT 消息。这些 保留消息 被作为普通的 MQTT 消息发送给已订阅该主题的所有客户端，同时也被存储起来发送给该主题的新订阅者。

客户端需要特定的策略操作授权才能访问 MQTT 保留消息。获取有关使用保留消息策略的示例，请参阅 保留的消息策略示例。

本节介绍了涉及保留消息的常见操作。

- 创建保留消息

  客户端在发布 MQTT 消息时确定是否保留消息。客户端可以在发布消息时使用 设备 SDK 设置 RETAIN 标志。应用程序和服务可以在使用 `Publish` 操作 发布 MQTT 消息时设置 RETAIN 标志。

  每个主题名称只保留一条消息。发布到主题的带有 RETAIN 标志设置的新消息将替换先前发送到该主题的任何现有保留消息。

  注意： 如果您设置了 RETAIN 标志则不能发布到 预留主题。

- 订阅保留消息主题

  客户端订阅保留消息主题，就像订阅任何其他 MQTT 消息主题一样。通过订阅保留消息主题接收的保留消息均有 RETAIN 标志设置。

  AWS IoT Core 当客户端向保留的消息主题发布带有 0 字节消息负载的保留消息时，将删除保留的消息。已订阅保留消息主题的客户端也将收到该 0 字节消息。

订阅包含保留消息主题的通配符主题筛选条件可使客户端接收发布到保留消息主题的后续消息，但在订阅时不会传递保留消息。

注意： 要在订阅时接收保留消息，订阅请求中的主题筛选条件必须与保留消息主题完全匹配。

订阅保留消息主题时收到的保留消息有 RETAIN 标志设置。订阅客户端在订阅后收到的保留消息则没有。

- 检索保留消息

  当客户端使用保留消息订阅主题时，保留消息会被自动发送给客户端。要让客户端在订阅时收到保留消息，必须订阅保留消息的确切主题名称。通过订阅包含保留消息主题的通配符主题筛选条件，客户端可以接收发布到保留消息主题的后续消息，但不会在订阅时传送保留消息。

  服务和应用可以通过调用 `ListRetainedMessages` 和 `GetRetainedMessage` 列出和检索保留消息。

  在 未 设置 RETAIN 标志的情况下，客户端不会被阻止发布消息到保留消息主题。这可能会导致意外结果，例如保留消息与订阅主题收到的消息不匹配。

  在 MQTT 5 中，如果保留消息设置了消息过期间隔且保留消息已过期，则订阅该主题的新订阅者在成功订阅后将不会收到该保留消息。

- 列出保留消息主题

  您可以通过调用 `ListRetainedMessages` 列出保留消息，并且可以在 AWS IoT 控制台 中查看保留消息。

- 获取保留消息详细信息

  您可以通过调用 `GetRetainedMessage` 获取保留消息的详细信息，并且可以在 AWS IoT 控制台 中查看这些信息。

- 保留 Will 消息

  连接设备时创建的 MQTT Will 消息 可以通过设置 `Connect Flag bits` 字段中的 `Will Retain` 标记保留。

- 删除保留消息

  通过将设置了 RETAIN 标志的消息和空（0字节）消息负载发布到要删除的保留消息的主题名称，设备、应用程序和服务可以删除保留消息。此类消息会从中删除保留的消息 AWS IoT Core，发送给订阅了该主题的客户端，但它们不会被保留 AWS IoT Core。

还可以通过访问 AWS IoT 控制台 中的保留消息以交互的方式删除保留消息。通过 AWS IoT 控制台 删除的保留消息也会向已订阅保留消息主题的客户端发送 0 字节的消息。

保留消息在删除后无法恢复。客户端需要发布新的保留消息才能取代已删除的消息。

- 保留消息的调试和故障排查

  AWS IoT 控制台 提供多个工具来帮助您进行保留消息故障排查：

  - 保留消息 页面

    AWS IoT 控制台中的 保留消息 页面提供您的账户在当前区域中存储的保留消息的分页列表。在此页面上，您可以：

    - 查看每条保留消息的详细信息，例如消息负载、QoS、接收时间。
    - 更新保留消息的内容。
    - 删除保留消息。

  - 打开 MQTT 测试客户端

    AWS IoT 控制台中的 MQTT 测试客户端 页面可以订阅并发布到 MQTT 主题。发布选项让您可以在发布的消息上设置 REATEIN 标志，以模拟设备的行为方式。

一些意想不到的结果可能是这些方面实现保留消息的结果 AWS IoT Core。

- 保留消息限制

  当一个账户存储了最大数量的保留消息时，会对设置了 RETAIN 且有效负载大于 0 字节的消息 AWS IoT Core 返回限制响应，直到某些保留的消息被删除并且保留的消息数量降至限制以下。

- 保留消息传递顺序

  不保证保留消息和订阅消息传递的顺序。

账单和保留消息

从客户端通过使用 AWS IoT 控制台或调用 `Publish` 发布设置了 RETAIN 标志的消息会产生 AWS IoT Core 定价-消息收发 中说明的额外消息收发费用。

客户端、使用 AWS IoT 控制台或通过调用检索保留的消息除了需要支付正常的 API 使用费外，还会`GetRetainedMessage`产生消息收费。AWS IoT Core 定价-消息收发 中说明了此类额外费用。

在设备意外断开连接时发布的 MQTT Will消息 会产生 AWS IoT Core 定价-消息收发 中说明的消息收发费用。

有关消息传递定价的更多信息，请参阅 AWS IoT Core 定价 - 消息传递。

比较 MQTT 保留消息和 MQTT 持久会话

保留消息和持久性会话是 MQTT 的标准特征，使设备能够接收离线时发布的消息。持久会话中可以发布保留消息。本节介绍这些特征的主要方面及其如何协作。

	保留消息	持久会话
主要特征	在连接大量设备后,保留消息可用于配置或通知这些设备。 如果您希望设备仅接收重新连接后发布到主题的最后一条消息，也可以使用保留消息。	持久会话对于间歇性连接而可能错过多条重要消息的设备非常有用。 设备可以连接持久会话来接收离线时发送的消息。
示例	保留消息可以在设备上线时提供有关其环境的配置信息。初始配置可以包括应该订阅的其他消息主题的列表，或者如何配置本地时区的相关信息。	通过间歇连接的蜂窝网络连接的设备可以使用持久会话，避免丢失在设备超出网络覆盖范围或需要关闭其蜂窝无线电时发送的重要消息。
初始订阅主题时收到的消息	订阅带有保留消息的主题后，会收到最近的保留消息。	在订阅没有保留消息的主题后，向该主题发布消息之前不会收到任何消息。
重新连接后订阅的主题	如果没有持久会话，客户端必须在重新连接后订阅主题。	重新连接后，将恢复订阅的主题。
重新连接后收到的消息	订阅带有保留消息的主题后，会收到最近的保留消息。	在设备断开连接时，所有以 QOS = 1 发布并使用 QOS =1 订阅的消息都将在设备重新连接后发送。
数据会话过期	在 MQTT 3 中，保留消息不会过期。它们将被存储，直至被替换或删除。在 MQTT 5 中，保留消息会在您设置的消息过	如果未在超时期限内重新连接客户端，持久会话将过期。持久会话过期后，在设备断开连接时以 QOS = 1 发布并使用 QOS =1 订阅的客户端订

	保留消息	持久会话
	期间隔后过期。有关更多信息，请参阅消息过期。	阅和保存消息将被删除。将不会传送已过期的消息。获取有关持久会话的会话过期时间的更多信息，请参阅 the section called “MQTT 持久性会话”。

有关持久性存储的信息，请参阅 the section called “MQTT 持久性会话”。

使用保留消息，发布客户端可确定是否应在连接后保留消息并将其传送到设备，是否有上一个会话。是否存储消息由发布者自行选择，存储的消息将被发送给所有使用 QoS 0 或 QoS 1 订阅的当前和未来客户端。保留消息一次只保存一条给定主题的消息。

当帐户存储了最大数量的保留消息时， AWS IoT Core 将对使用 RETAIN 集和大于 0 字节的有效负载发布的消息返回限制响应，直到删除一些保留消息且保留消息数不超过限制。

MQTT 保留了消息和 AWS IoT 设备影子

保留消息和 Device Shadows 都保留来自设备的数据，但它们的行为不同，用途也不同。本节介绍它们的相似之处和不同之处。

	保留消息	设备影子
消息有效负载具有预定义的结构或架构	正如实施所定义的。MQTT 没有为其消息负载指定结构或架构。	AWS IoT 支持特定的数据结构。
更新消息负载会生成事件消息	发布保留消息会将消息发送到订阅的客户端，但不会生成其他更新消息。	更新 Device Shadow会出现更新描述更改的消息。
消息更新已编号	保留消息不会自动编号。	Device Shadow 文档具有自动版本号和时间戳。
消息有效负载附加到事物资源上	保留消息不会附加到事物资源。	Device Shadows 附加到事物资源。

	保留消息	设备影子
更新消息有效负载的单个元素	如果不更新整个消息负载，就无法更改消息的单个元素。	可以更新 Device Shadow 文档的各个元素，而无需更新整个 Device Shadow 文档。
客户端在订阅时收到消息数据	在订阅带保留消息的主题后，客户端会自动收到保留消息。	客户端可以订阅 Device Shadow 更新，但他们必须故意请求当前状态。
索引和可搜索性	保留消息不会编制索引进行搜索。	实例集索引 Device Shadow 数据进行搜索和聚合。

MQTT Last Will and Testament（LWT）消息

Last Will and Testament（LWT）是 MQTT 中的一项特征。借助 LWT，客户端可以指定一条消息，当出现未启动的断开连接时，代理将该消息发布到客户定义的主题，并发送给订阅该主题的所有客户端。客户端指定的消息称为 LWT 消息或 Will 消息，客户端定义的主题称为 Will 主题。您可以指定当设备连接到代理时的 LWT 消息。通过在连接期间在 `Connect Flag bits` 字段中设置 `Will Retain` 标志，可以保留这些消息。例如，如果 `Will Retain` 标志设置为 `1`，则 Will 消息将存储在代理中的关联 Will 主题中。有关更多信息，请参阅 Will 消息。

代理将存储 Will 消息，直到出现未启动的断开连接。发生这种情况时，代理将向所有订阅 Will 主题的客户端发布消息以通知断开连接。如果客户端使用 MQTT DISCONNECT 消息通过客户端启动的断开连接来断开与代理的连接，则代理不会发布存储的 LWT 消息。在所有其它情况下，将分派 LWT 消息。有关代理发送 LWT 消息时的断开连接场景的完整列表，请参阅连接/断开连接事件。

使用 ConnectAttributes

`ConnectAttributes` 允许您在 IAM policy 中指定要在连接消息中使用的属性，如 `PersistentConnect` 和 `LastWill`。借助 `ConnectAttributes`，您可以构建默认情况下不允许设备访问新特征的策略，这在设备受到威胁时很有帮助。

`connectAttributes` 支持以下特征：

`PersistentConnect`

当客户端和代理之间的连接中断时，使用 `PersistentConnect` 特征可以保存客户端在连接期间所做的所有订阅。

`LastWill`

当客户端意外断开连接时，使用 `LastWill` 特征可以向 `LastWillTopic` 发布消息。

默认情况下，您的策略具有非持久性连接，并且没有为此连接传递任何属性。如果您想要具有持久连接，则必须在 IAM 策略中指定持久连接。

有关 `ConnectAttributes` 示例，请参阅连接策略示例。

MQTT 5 支持的特征

AWS IoT Core 对 MQTT 5 的支持基于 MQTT v5.0 规范，但有一些区别，如中所述。the section called “AWS IoT 与 MQTT 规格的区别”

AWS IoT Core 支持以下 MQTT 5 功能：

- 共享订阅
- 干净启动和会话过期
- 所有 ACK 上的原因代码
- 主题别名
- 消息过期
- 其它 MQTT 5 特征

共享订阅

AWS IoT Core 支持 MQTT 3 和 MQTT 5 的共享订阅。共享订阅允许多个客户端共享对某个主题的订阅，并且只有一个客户端会收到使用随机分布发布到该主题的消息。共享订阅可以有效地在多个订阅用户之间对 MQTT 消息实现负载均衡。例如，假设您有 1000 台设备发布到同一个主题，有 10 个后端应用程序正在处理这些消息。在这种情况下，后端应用程序可以订阅同一个主题，并且每个应用程序都会随机接收设备向共享主题发布的消息。这实际上是“共享”这些消息的负载。共享订阅还可以提高弹性。当任何后端应用程序断开连接时，代理会将负载分配给组中剩余的订阅用户。

要使用共享订阅，客户需要按如下方式订阅共享订阅的主题筛选条件：

```
$share/{ShareName}/{TopicFilter}
```

- `$share` 是一个文本字符串，用于表示共享订阅的主题筛选条件，该筛选条件必须以 `$share` 开头。

- {ShareName} 是一个字符串，用于指定一组订阅用户使用的共享名称。共享订阅的主题筛选条件必须包含 ShareName 且后跟 / 字符。{ShareName} 不得包含以下字符：/、+ 或 #。{ShareName} 的最大大小为 128 字节。
- {TopicFilter} 遵循与非共享订阅相同的主题筛选条件语法。{TopicFilter} 的最大大小为 256 字节。
- $share/{ShareName}/{TopicFilter} 所需的两个斜杠（/）不包括在主题和主题筛选条件中的最大斜杠数限制中。

具有相同 {ShareName}/{TopicFilter} 的订阅属于同一个共享订阅组。您可以创建多个共享订阅组，但不要超过每个组的共享订阅限制。有关更多信息，请参阅 AWS 一般参考中的 AWS IoT Core 端点和配额。

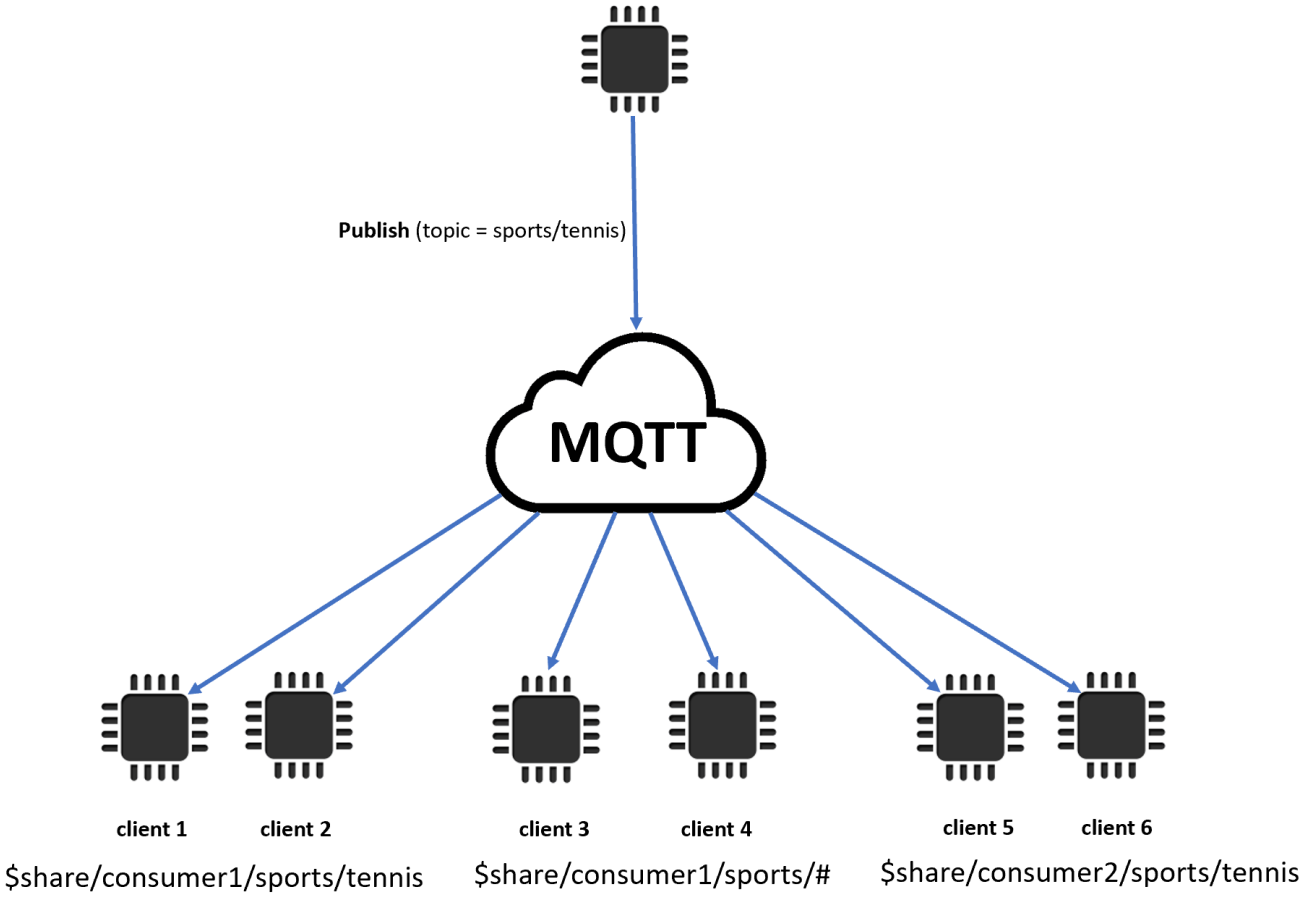
下面的各个表比较了非共享订阅和共享订阅：

订阅	描述	主题筛选条件示例
非共享订阅	每个客户端创建单独的订阅以接收已发布的消息。当消息发布到某个主题时，该主题的所有订阅用户都会收到该消息的副本。	`sports/tennis` `sports/#`
共享订阅	多个客户端可以共享对一个主题的订阅，并且只有一个客户端会收到以随机分布方式发布到该主题的消息。	`$share/consumer/sports/tennis` `$share/consumer/sports/#`

非共享订阅流程	共享订阅流程		
			

使用共享订阅的重要注意事项

- 当针对 QoS0 订阅用户的发布尝试失败时，不会尝试进行重试，并且消息将被丢弃。
- 当针对具有干净会话的 QoS1 订阅用户的发布尝试失败时，该消息将发送给该组中的另一个订阅用户以尝试进行多次重试。在尝试所有重试后仍未能传送的消息将被丢弃。
- 当针对具有持久性会话的 QoS1 订阅用户的发布尝试因订阅用户处于离线状态而失败时，消息将不会排队，而是会尝试发送给组中的其他订阅用户。在尝试所有重试后仍未能传送的消息将被丢弃。
- 共享订阅不接收保留的消息。
- 当共享订阅包含通配符（# 或 +）时，一个主题可能有多个匹配的共享订阅。如果发生这种情况，消息代理会复制发布消息，并将其发送给每个匹配的共享订阅中的随机客户端。下图可以解释共享订阅的通配符行为。



在此示例中，有三个与发布 MQTT 主题 `sports/tennis` 匹配的共享订阅。消息代理复制已发布的消息，并将消息发送给每个匹配组中的随机客户端。

客户端 1 和客户端 2 共享订阅：`$share/consumer1/sports/tennis`

客户端 3 和客户端 4 共享订阅：`$share/consumer1/sports/#`

客户端 5 和客户端 6 共享订阅：`$share/consumer2/sports/tennis`

有关共享订阅限制的更多信息，请参阅《AWS 一般参考》中的 AWS IoT Core 端点和限额。要在AWS IoT 控制台中使用 AWS IoT MQTT 客户端测试共享订阅，请参阅???。有关共享订阅的更多信息，请参阅 MQTTv5.0 规范中的共享订阅。

干净启动和会话过期

您可以使用“干净启动”和“会话过期”来更灵活地处理持久性会话。干净启动标志指示是否应在不使用现有会话的情况下启动会话。会话过期间隔指示断开连接后会话保持多长时间。可以在断开连接后修改会话过期间隔。有关更多信息，请参阅 the section called “MQTT 持久性会话”。

所有 ACK 上的原因代码

您可以使用原因代码更轻松地调试或处理错误消息。消息代理根据与代理的交互类型（订阅、发布、确认）返回原因代码。有关更多信息，请参阅 MQTT 原因代码。有关 MQTT 原因代码的完整列表，请参阅 MQTT v5 规范。

主题别名

您可以将主题名称替换为主题别名，主题别名是一个双字节整数。使用主题别名可以优化主题名称的传输，从而有可能降低计量数据服务的数据成本。 AWS IoT Core 默认限制为 8 个主题别名。有关更多信息，请参阅 AWS 一般参考中的 AWS IoT Core 端点和配额。

消息过期

您可以向已发布的消息添加消息过期值。这些值表示消息过期间隔（以秒为单位）。如果未在该间隔内将消息发送给订阅者，该消息将过期并被删除。如果不设置消息过期值，消息会过期。

在出站时，订阅者将收到一条消息，其中包含到期间隔内的剩余时间。例如，如果入站发布消息的消息过期间隔为 30 秒，并且在 20 秒后路由到订阅者，则消息过期字段将更新为 10。订阅者收到的消息的更新后的 MEI 可能为 0。这是因为只要剩余时间为 999 毫秒或更短，它就会更新为 0。

在中 AWS IoT Core，最小消息到期间隔为 1。如果从客户端将间隔设置为 0，则间隔将调整为 1。最长消息过期间隔为 604800（7 天）。高于此值的任何值都将调整为最大值。

在跨版本通信中，消息过期的行为由入站发布消息的 MQTT 版本决定。例如，通过 MQTT5 连接的会话发送的消息过期消息可能会对订阅 MQTT3 会话的设备过期。下表列出了消息过期如何支持以下类型的发布消息：

发布消息类型	消息过期间隔
定期发布	如果服务器未能在指定时间内传送消息，则过期的消息将被删除，订阅者将无法收到该消息。这包括设备未发布确诊其 QoS 1 消息等情况。
保留	如果保留消息过期，而新客户端订阅了该主题，则该客户端在订阅后将不会收到该消息。
最后遗嘱	最后遗嘱消息间隔是在客户端断开连接，并且服务器尝试向其订阅者传送最后遗嘱消息之后开始的。
已排队消息	如果具有消息过期间隔的出站 QoS1 在客户端离线时过期，则持久性会话恢复后，客户端将不会收到过期的消息。

其它 MQTT 5 特征

服务器断开连接

断开连接时，服务器可以主动向客户端发送 DISCONNECT，以通知连接关闭，同时发送断开连接的原因代码。

请求/响应

发布者可以请求接收方在收到时向发布者指定的主题发送响应。

最大数据包大小

客户端和服务器可以独立指定它们支持的最大数据包大小。

负载格式和内容类型

发布消息时，您可以指定负载格式（二进制、文本）和内容类型。这些消息将被转发给消息的接收者。

MQTT 5 属性

MQTT 5 属性是 MQTT 标准的重要补充，用于支持新的 MQTT 5 特征，例如会话过期和请求/响应模式。在中 AWS IoT Core，您可以创建可以转发出站消息中的属性的规则，或者使用 HTTP Publis h 发布带有某些新属性的 MQTT 消息。

下表列出了所有 AWS IoT Core 支持的 MQTT 5 属性。

属性	描述	输入类型	数据包
负载格式指示符	一个布尔值，指示负载是否格式化为 UTF-8。	字节	PUBLISH、CONNECT
内容类型	一个描述负载内容的 UTF-8 字符串。	UTF-8 字符串	PUBLISH、CONNECT
回复主题	一个 UTF-8 字符串，描述接收方作为请求-响应流程的一部分应发布到的主题。主题不得包含通配符。	UTF-8 字符串	PUBLISH、CONNECT
关联数据	请求消息的发送者用来指示回复消息回复哪个请求的二进制数据。	二元	PUBLISH、CONNECT
用户属性	一对 UTF-8 字符串。此属性可以在一个数据包中出现多次。接收者将按照键-值对的发送顺序接收键-值对。	UTF-8 字符串对	CONNECT、PUBLISH、Will Properties、SUBSCRIBE、DISCONNECT、UNSUBSCRIBE
消息过期间隔	一个 4 字节整数，表示消息过期间隔（以秒为单位）。如果此数值不存在，消息永远不会过期。	4 字节整数	PUBLISH、CONNECT
会话过期间隔	一个 4 字节的整数，表示会话到期时间间隔（以秒为单位）。 AWS IoT Core 最多支持 7 天，默认最长为 1 小时。如果您设置的值超过了账户的最大值，则 AWS IoT Core 会在 CONNACK 中返回调整后的值。	4 字节整数	CONNECT、CONNACK、DISCONNECT
分配的客户端标识符	设备未指定客户端 ID AWS IoT Core 时生成的随机客户端 ID。随机客户端 ID 必须是代理当前管理的任何其他会话都未使用的新客户端标识符。	UTF-8 字符串	CONNACK

属性	描述	输入类型	数据包
服务器保持活动	一个 2 字节整数，表示服务器分配的保持活动时间。如果客户端处于非活动状态的时间超过保持活状态的动时间，服务器将断开客户端的连接。	2 字节整数	CONNACK
请求问题信息	一个布尔值，表示在失败时是发送原因字符串还是用户属性。	字节	CONNECT
接收最大值	一个 2 字节整数，表示在未收到 PUBACK 的情况下可以发送的 PUBLISH QOS > 0 数据包的最大数量。	2 字节整数	CONNECT、CONNACK
主题别名最大值	此值表示将被接受作为主题别名的最大值。默认值为 0。	2 字节整数	CONNECT、CONNACK
最大 QoS 数	支持的 QoS 的最大值。 AWS IoT Core 默认值为 1。 AWS IoT Core 不支持 QoS2。	字节	CONNACK
保留可用	一个布尔值，用于指示 AWS IoT Core 消息代理是否支持保留的消息。默认 为 1。	字节	CONNACK
最大数据包大小	AWS IoT Core 接受和发送的最大数据包大小。不能超过 128 KB。	4 字节整数	CONNECT、CONNACK
通配符订阅可用	一个布尔值，用于指示 AWS IoT Core 消息代理是否支持可用的通配符订阅。默认 为 1。	字节	CONNACK
订阅标识符可用	一个布尔值，用于指示 AWS IoT Core 消息代理是否支持可用订阅标识符。默认值是 0。	字节	CONNACK

MQTT 原因代码

MQTT 5 引入了改进的错误报告和原因代码响应。 AWS IoT Core 可能会返回原因代码，包括但不限于以下按数据包分组的内容。有关 MQTT 支持的原因代码的完整列表，请参阅 MQTT 5 规范。

CONNACK 原因代码

值	十六进制	原因代码名称	描述
0	0x00	成功	连接被接受。
128	0x80	未指定的错误	服务器不希望透露失败的原因，或者其他原因代码都不适用。
133	0x85	客户端标识符无效	客户端标识符是有效的字符串，但服务器不允许使用。
134	0x86	用户名或密码错误	服务器不接受客户端指定的用户名或密码。
135	0x87	未授权	客户端无权连接。
144	0x90	主题名称无效	遗嘱主题名称格式正确，但未被服务器接受。
151	0x97	超出配额	已超出实施或管理施加的限制。
155	0x9B	不支持 QoS	服务器不支持遗嘱 QoS 中设置的 QoS。

PUBACK 原因代码

值	十六进制	原因代码名称	描述
0	0x00	成功	该消息已被接受。继续发布 QoS 1 消息。
128	0x80	未指定的错误	接收者不接受发布，但要么不想透露原因，要么与其他值不匹配。
135	0x87	未授权	PUBLISH 未经授权。
144	0x90	主题名称无效	主题名称格式正确，但不被客户端或服务器接受。
145	0x91	正在使用数据包标识符	数据包标识符已在使用中。这可能表示客户端和服务器之间的会话状态不匹配。

值	十六进制	原因代码名称	描述
151	0x97	超出配额	已超出实施或管理施加的限制。

断开连接原因代码

值	十六进制	原因代码名称	描述
129	0x81	数据包格式不正确	收到的数据包不符合此规范。
130	0x82	协议错误	收到意外或无序的数据包。
135	0x87	未授权	请求未经授权。
139	0x8B	服务器正在关闭	服务器正在关闭。
141	0x8D	保持活动状态超时	连接已关闭，因为在 1.5 倍的保持活动状态时间内没有收到任何数据包。
142	0x8E	会话已接管	使用相同 ClientID 的另一个连接已连接，导致此连接关闭。
143	0x8F	主题筛选条件无效	主题筛选条件格式正确，但未被服务器接受。
144	0x90	主题名称无效	主题名称格式正确，但未被该客户端或服务器接受。
147	0x93	超出最大接收数	客户端或服务器收到的发布数超过了未发送 PUBACK 或 PUBCOMP 的最大接收数。
148	0x94	主题别名无效	客户端或服务器收到了一个 PUBLISH 数据包，其中包含的主题别名大于其在 CONNECT 或 CONNACK 数据包中发送的最大主题别名。
151	0x97	超出配额	已超出实施或管理施加的限制。

值	十六进制	原因代码名称	描述
152	0x98	管理操作	由于管理操作，连接已关闭。
155	0x9B	不支持 QoS	客户端指定的 QoS 大于 CONNACK 的“最大 QoS”中指定的 QoS。
161	0xA1	不支持订阅标识符	服务器不支持订阅标识符；不接受订阅。

PUBACK 原因代码

值	十六进制	原因代码名称	描述
0	0x00	授予的 QoS 0	接受订阅，发送的最大 QoS 将为 QoS 0。这可能比请求的 QoS 低。
1	0x01	授予的 QoS 1	接受订阅，发送的最大 QoS 将为 QoS 1。这可能比请求的 QoS 低。
128	0x80	未指定的错误	未接受订阅，要么服务器不希望透露原因，要么其他原因代码都不适用。
135	0x87	未授权	客户端无权进行此订阅。
143	0x8F	主题筛选条件无效	主题筛选条件格式正确，但不允许此客户端使用。
145	0x91	正在使用数据包标识符	指定的数据包标识符已在使用中。
151	0x97	超出配额	已超出实施或管理施加的限制。

UNSUBACK 原因代码

值	十六进制	原因代码名称	描述
0	0x00	成功	订阅将被删除。
128	0x80	未指定的错误	无法完成取消订阅，要么服务器不希望透露原因，要么其他原因代码都不适用。
143	0x8F	主题筛选条件无效	主题筛选条件格式正确，但不允许此客户端使用。
145	0x91	正在使用数据包标识符	指定的数据包标识符已在使用中。

AWS IoT 与 MQTT 规格的区别

尽管消息代理的实施基于 MQTT v3.1.1 规范和 MQTT v5.0 规范，但与这些规范有如下区别：

- AWS IoT 不支持 MQTT 3 的以下数据包：PUBREC、PUBREL 和 PUBCOMP。
- AWS IoT 不支持 MQTT 5 的以下数据包：PUBREC、PUBREL、PUBCOMP 和 AUTH。
- AWS IoT 不支持 MQTT 5 服务器重定向。
- AWS IoT 仅支持 MQTT 服务质量 (QoS) 级别 0 和 1。 AWS IoT 不支持以 QoS 级别 2 发布或订阅。在请求 QoS 级别 2 时，消息代理不会发送 PUBACK 或 SUBACK。
- 在中 AWS IoT，订阅 QoS 级别为 0 的主题意味着消息的传送次数为零次或多次。消息可能会多次发送。多次发送的消息在发送时可能会使用不同的数据包 ID。在这些情况下，不会设置 DUP 标志。
- 在响应连接请求时，消息代理将发送 CONNACK 消息。此消息包含一个标志，用于指明该连接是否会恢复上一个会话。
- 在发送其它控制数据包或断开连接请求之前，客户端必须等待从 AWS IoT 消息代理发送到设备的 CONNACK 消息。
- 当客户端订阅主题时，在消息代理开始发送 SUBACK 和客户端开始收到新的匹配消息之间存在时间延迟。
- 当客户端在订阅主题的主题过滤条件中使用通配符 # 时，主题层次结构中处于及其级别以下的所有字符串均会匹配。但是，父主题不匹配。例如，订阅主题 `sensor/#` 接收发布到主题`sensor/`、`sensor/temperature`、`sensor/temperature/room1` 的消息，但不是发布到 `sensor` 的消息。有关通配符的更多信息，请参阅 主题筛选条件。

- 消息代理使用客户端 ID 标识每个客户。客户端 ID 作为 MQTT 负载的一部分从客户端传递到消息代理。客户端 ID 相同的两个客户端无法同时连接到消息代理。当某个客户端使用另一客户端正在使用的客户端 ID 连接到消息代理时，会接受新的客户端连接，而之前连接的客户端会断开连接。
- 在极少数情况下，消息代理可能会使用不同的数据包 ID 再次发送相同的逻辑 PUBLISH 消息。
- 订阅包含通配符的主题筛选条件无法接收保留消息。要接收保留消息，订阅请求必须包含与保留消息主题完全匹配的主题筛选条件。
- 消息代理并不保证收到消息和 ACK 的顺序。
- AWS IoT 可能有与规格不同的限制。有关更多信息，请参阅《AWS IoT 参考指南》中的AWS IoT Core 消息代理和协议限制与限额。
- 不支持 MQTT DUP 标志。

HTTPS

客户端可使用 HTTP 1.0 或 1.1 协议向 REST API 发出请求来发布消息。有关 HTTP 请求使用的身份验证和端口映射，请参阅协议、端口映射和身份验证。

> **Note**
>
> HTTPS 不像 MQTT 那样支持 `clientId` 值。`clientId` 在使用 MQTT 时可用，但在使用 HTTPS 时不可用。

HTTPS 消息 URL

设备和客户端通过向特定于客户端的终端节点和特定于主题的 URL 发出 POST 请求来发布消息：

```
https://IoT_data_endpoint/topics/url_encoded_topic_name?qos=1
```

- *IoT_data_endpoint* 是 AWS IoT 设备数据终端节点。您可以使用以下 AWS CLI 命令在 AWS IoT 控制台中的事物详细信息页面或客户端上找到终端节点：

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

  该终端节点看起来应如下所示：`a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com`
- *url_encoded_topic_name* 是正在发送的消息的完整主题名称。

HTTPS 消息代码示例

以下是一些如何向 AWS IoT发送 HTTPS 消息的示例。

Python (port 8443)

```
import requests
import argparse

# define command-line parameters
parser = argparse.ArgumentParser(description="Send messages through an HTTPS
 connection.")
parser.add_argument('--endpoint', required=True, help="Your AWS IoT data custom
 endpoint, not including a port. " +
                                                       "Ex: \"abcdEXAMPLExyz-
ats.iot.us-east-1.amazonaws.com\"")
parser.add_argument('--cert', required=True, help="File path to your client
 certificate, in PEM format.")
parser.add_argument('--key', required=True, help="File path to your private key, in
 PEM format.")
parser.add_argument('--topic', required=True, default="test/topic", help="Topic to
 publish messages to.")
parser.add_argument('--message', default="Hello World!", help="Message to publish. "
 +
                                                       "Specify empty string to
 publish nothing.")

# parse and load command-line parameter values
args = parser.parse_args()

# create and format values for HTTPS request
publish_url = 'https://' + args.endpoint + ':8443/topics/' + args.topic + '?qos=1'
publish_msg = args.message.encode('utf-8')

# make request
publish = requests.request('POST',
            publish_url,
            data=publish_msg,
            cert=[args.cert, args.key])

# print results
print("Response status: ", str(publish.status_code))
if publish.status_code == 200:
```

```
        print("Response body:", publish.text)
```

Python (port 443)

```
import requests
import http.client
import json
import ssl

ssl_context = ssl.SSLContext(protocol=ssl.PROTOCOL_TLS_CLIENT)
ssl_context.minimum_version = ssl.TLSVersion.TLSv1_2

# note the use of ALPN
ssl_context.set_alpn_protocols(["x-amzn-http-ca"])
ssl_context.load_verify_locations(cafile="./<root_certificate>")

# update the certificate and the AWS endpoint
ssl_context.load_cert_chain("./<certificate_in_PEM_Format>",
 "<private_key_in_PEM_format>")
connection = http.client.HTTPSConnection('<the ats IoT endpoint>', 443,
 context=ssl_context)
message = {'data': 'Hello, I'm using TLS Client authentication!'}
json_data = json.dumps(message)
connection.request('POST', '/topics/device%2Fmessage?qos=1', json_data)

# make request
response = connection.getresponse()

# print results
print(response.read().decode())
```

CURL

您可以使用来自客户端或设备的 curl 向 AWS IoT发送消息。

使用 curl 从 AWS IoT 客户端设备发送消息

1. 检查 curl 版本。

   a. 在客户端上，在命令提示符下运行此命令。

      curl --help

在帮助文本中，查找 TLS 选项。您应看到 `--tlsv1.2` 选项。

b. 如果您看到 `--tlsv1.2` 选项，请继续操作。

c. 如果您没有看到 `--tlsv1.2` 选项或者收到 `command not found` 错误，则您可能需要先在客户端上更新或安装 curl，或者安装 openssl，然后再继续。

2. 在客户端上安装证书。

复制您在 AWS IoT 控制台中注册客户端（事物）时创建的证书文件。请先确保客户端上有这三个证书文件，然后再继续。

- CA 证书文件（此示例中为 *Amazon-root-CA-1.pem*）。
- 客户端的证书文件（此示例中为 *device.pem.crt*）。
- 客户端的私有密钥文件（此示例中为 *private.pem.key*）。

3. 创建 curl 命令行，并替换您的账户和系统的可替换值。

```
curl --tlsv1.2 \
    --cacert Amazon-root-CA-1.pem \
    --cert device.pem.crt \
    --key private.pem.key \
    --request POST \
    --data "{ \"message\": \"Hello, world\" }" \
    "https://IoT_data_endpoint:8443/topics/topic?qos=1"
```

--tlsv1.2

使用 TLS 1.2 (SSL)。

--cacert *Amazon-root-CA-1.pem*

用于验证对等方的 CA 证书的文件名和路径（如有必要）。

--cert *device.pem.crt*

客户端的证书文件名和路径（如有必要）。

--key *private.pem.key*

客户端的私有密钥文件名和路径（如有必要）。

--request POST

HTTP 请求的类型（此示例中为 POST）。

--data "*{ \"message\": \"Hello, world\" }*"

要发布的 HTTP POST 数据。在此示例中，它是一个 JSON 字符串，其内部引号通过反斜杠字符 (\) 进行转义。

"https://*IoT_data_endpoint*:8443/topics/*topic*?qos=1"

您的客户端 AWS IoT 设备数据端点的 URL，后面是 HTTPS 端口:8443，然后是关键字/topics/和主题名称topic，在本例中是主题名称。指定服务质量作为查询参数？qos=1。

4. 在 AWS IoT 控制台中打开 MQTT 测试客户端。

   按照 使用 MQTT 客户端查看 AWS IoT MQTT 消息 中的说明执行操作，配置控制台以订阅消息，该消息应带有 curl 命令中所用的名为 *topic* 的主题，或使用 # 通配符主题筛选条件。

5. 测试命令。

   在 AWS IoT 控制台的测试客户端中监控主题时，请转到客户端并发出您在步骤 3 中创建的 curl 命令行。控制台中应显示客户端的消息。

MQTT 主题

MQTT 主题用于标识 AWS IoT 消息。 AWS IoT 客户通过提供消息主题名称来识别他们发布的消息。客户端通过将主题筛选条件注册到 AWS IoT Core来标识要订阅（接收）的消息。消息代理使用主题名称和主题筛选条件将消息从发布客户端传输到订阅客户端。

消息代理使用主题来识别使用 MQTT 发送的消息以及使用 HTTP 发送到 HTTPS 消息 URL 的消息。

虽然 AWS IoT 支持一些保留的系统主题，但大多数 MQTT 主题是由您（即系统设计者）创建和管理的。 AWS IoT 使用主题来识别从发布客户端收到的消息，并选择要发送给订阅客户端的消息，如以下各节所述。在为系统创建主题命名空间之前，请查看 MQTT 主题的特征，以创建最适合您的 IoT 系统的主题名称层次结构。

主题名称

主题名称和主题筛选条件是 UTF-8 编码的字符串。它们可以使用正斜杠 (/) 字符分隔层次结构的级别来表示信息的层次结构。例如，此主题名称可以引用房间 1 中的温度传感器：

- `sensor/temperature/room1`

在此示例中，其他房间中可能还有其它类型的传感器，其主题名称如下：

- `sensor/temperature/room2`
- `sensor/humidity/room1`
- `sensor/humidity/room2`

> **Note**
>
> 在考虑系统中消息的主题名称时，请记住：
>
> - 主题名称和主题筛选条件区分大小写。
> - 主题名称不得包含个人身份信息。
> - 以 $ 开头的主题名称均为只能由 AWS IoT Core使用的预留主题。
> - AWS IoT Core 无法在 AWS 账户 s 或地区之间发送或接收消息。

有关设计主题名称和命名空间的更多信息，请参阅我们的白皮书《为 AWS IoT Core Core 设计 MQTT 主题》。

有关应用程序如何发布和订阅消息的示例，请从 入门 AWS IoT Core 和 AWS IoT 设备 SDK、移动 SDK 和 AWS IoT 设备客户端 开始查阅。

> **Important**
>
> 主题命名空间仅限于 AWS 账户 和区域。例如，一个区域 AWS 账户 中某人使用的`sensor/temp/room1`主题与另一个区域的同一个 AWS 账户使用的`sensor/temp/room1`主题或任何其他区域中任何其他 AWS 账户 账户使用的主题不同。

主题 ARN

所有主题 ARN (Amazon Resource Name) 都采用以下形式：

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

例如，`arn:aws:iot:us-west-2:123EXAMPLE456:topic/application/topic/device/sensor` 是主题 `application/topic/device/sensor` 的 ARN。

主题筛选条件

订阅客户端会向消息代理注册主题筛选条件，以指定消息代理应将消息发送到的主题。主题筛选条件可以是用于订阅单个主题的单个主题名称，也可以包含通配符以同时订阅多个主题名称。

发布客户端无法在其发布的主题名称中使用通配符。

下表列出了可在主题筛选条件中使用的通配符。

主题通配符

通配符	匹配项	注意
#	主题层次结构中位于其级别及其以下的所有字符串。	必须是主题筛选条件中的最后一个字符。 必须是其主题层次结构级别中的唯一字符。 可以在还包含 + 通配符的主题筛选条件中使用。
+	级别中包含字符的任何字符串。	必须是其主题层次结构级别中的唯一字符。 可在主题筛选条件的多个级别中使用。

将通配符用于之前的传感器主题名称示例：

- `sensor/#` 订阅接收发布到 `sensor/`、`sensor/temperature` 和 `sensor/temperature/room1` 的消息，但不会接收发布到 `sensor` 的消息。
- `sensor/+/room1` 订阅接收发布到 `sensor/temperature/room1` 和 `sensor/humidity/room1` 的消息，但不会接收发布到 `sensor/temperature/room2` 或 `sensor/humidity/room2` 的消息。

主题筛选条件 ARN

所有主题筛选条件 ARN (Amazon Resource Name) 都采用以下形式：

```
arn:aws:iot:aws-region:AWS-account-ID:topicfilter/TopicFilter
```

例如，arn:aws:iot:us-west-2:123EXAMPLE456:topicfilter/application/topic/+/sensor 是主题筛选条件 application/topic/+/sensor 的 ARN。

MQTT 消息负载

在您的 MQTT 消息中发送的消息有效负载不是由指定的 AWS IoT，除非它是针对其中一个。the section called “保留的主题”为了满足应用程序的需求，我们建议您在协议的AWS IoT Core 服务配额限制范围内为主题定义消息有效负载。

对消息负载使用 JSON 格式可以让 AWS IoT 规则引擎解析您的消息并对其应用 SQL 查询。如果应用程序不需要规则引擎将 SQL 查询应用于消息有效负载，则您可以使用应用程序所需的任何数据格式。有关 SQL 查询中使用的 JSON 文档中的限制和预留字符的信息，请参阅 JSON 扩展。

有关设计 MQTT 主题及其相应的消息负载的更多信息，请参阅为 AWS IoT Core设计 MQTT 主题。

如果消息大小限制超过服务配额，则会导致 CLIENT_ERROR，并显示原因 PAYLOAD_LIMIT_EXCEEDED 和“消息有效负载超过消息类型的大小限制。” 有关消息大小限制的更多信息，请参阅 AWS IoT Core 消息代理限制和配额。

保留的主题

以美元符号 ($) 开头的主题保留供使用 AWS IoT。您可以在允许的情况下订阅和发布到这些保留的主题；但是，您不能创建以美元符号开头的新主题。对保留的主题执行不受支持的发布或订阅操作可能会导致连接终止。

资产模型主题

主题	允许的客户端操作	描述
$aws/sitewise/asset-models/ /assets/ assetiD /properties/ properties/ properties/ pro assetModelId	订阅	AWS IoT SiteWise 向该主题发布资产属性通知。有关更多信息，请参阅AWS IoT SiteWise 用户指南中的与其他 AWS 服务交互。

AWS IoT Device Defender 话题

这些消息支持简明二进制对象表示 (CBOR) 格式和 JavaScript 对象表示法 (JSON) 的响应缓冲区，具体取决于主题的*####*格式。 AWS IoT Device Defender 仅主题支持 MQTT 发布。

payload-format	响应格式数据类型
cbor	简洁二进制对象表示法 (CBOR)
json	JavaScript 对象表示法 (JSON)

有关更多信息，请参阅从设备发送指标。

主题	允许的操作	描述
$aws/things/*thingName* /defender/metrics/*payload-format*	Publish	AWS IoT Device Defender 代理向该主题发布指标。有关更多信息，请参阅从设备发送指标。
$aws/things/*thingName* /defender/metrics/*payload-format* /accepted	订阅	AWS IoT *# AWS IoT Device Defender ##### $aws/things/thingName /defender/metrics/payload-formation ########## ####*有关更多信息，请参阅从设备发送指标。
$aws/things/*thingName* /defender/metrics/*payload-format* /rejected	订阅	AWS IoT *# AWS IoT Device Defender ### $aws/things/thingName /defender/metrics/payload-formation ########## #########* 有关更多信息，请参阅从设备发送指标。

AWS IoT Core 设备位置主题

AWS IoT Core 设备位置可以解析来自您设备的测量数据，并提供物联网设备的估计位置。来自设备的测量数据可以包括 GNSS、Wi-Fi、蜂窝和 IP 地址。 AWS IoT Core 然后，设备位置选择可提供最佳精

度并求解设备位置信息的测量类型。有关更多信息，请参阅 AWS IoT Core 设备位置 和 使用 AWS IoT Core 设备位置 MQTT 主题解析设备位置。

主题	允许的操作	描述
$aws/device_location/*customer_device_id*/get_position_estimate	Publish	设备向本主题发布信息，以获取扫描的原始测量数据，以便通过 “ AWS IoT Core 设备位置” 进行解析。
$aws/device_location/*customer_device_id*/get_position_estimate/accepted	订阅	AWS IoT Core 成功解析设备位置后，设备位置将发布到此主题。
$aws/device_location/*customer_device_id*/get_position_estimate/rejected	订阅	AWS IoT Core 由于 4xx 错误而无法成功解析设备位置时，设备位置将发布到此主题中。

事件主题

> **Note**
>
> 有关为 LoRa WAN 事件保留的 MQTT 主题的更多信息，请参阅连接状态事件。

主题	允许的客户端操作	描述
$aws/活动/证书/已注册/*caCertificateId*	订阅	AWS IoT 当 AWS IoT 自动注册证书以及客户端出示带有PENDING_ACTIVATION 状态的证书时，会发布此消息。有关更多信息，请参阅 the section called “配置客户端的首次连接以进行自动注册”。
$aws/events/job/*jobID*/canceled	订阅	AWS IoT 取消任务时会发布此消息。有关更多信息，请参阅 任务事件。

主题	允许的客户端操作	描述
$aws/events/job/*jobID*/cancellation_in_progress	订阅	AWS IoT 在取消任务时发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/job/*jobID*/completed	订阅	AWS IoT 任务完成后发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/job/*jobID*/deleted	订阅	AWS IoT 删除作业时会发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/job/*jobID*/deletion_in_progress	订阅	AWS IoT 删除任务时会发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobID*/canceled	订阅	AWS IoT 取消任务执行时会发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobID*/deleted	订阅	AWS IoT 删除任务执行时会发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobID*/failed	订阅	AWS IoT 任务执行失败时发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobID*/rejected	订阅	AWS IoT 在任务执行被拒绝时发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobID*/removed	订阅	AWS IoT 删除任务执行时会发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobID*/succeeded	订阅	AWS IoT 任务执行成功后发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobID*/timed_out	订阅	AWS IoT 在任务执行超时时时发布此消息。有关更多信息，请参阅 任务事件。
$aws/events/presence/connected/*clientId*	订阅	AWS IoT 当具有指定客户端 ID 的 MQTT 客户端连接到时，将发布到 AWS IoT此主题。有关更多信息，请参阅 连接/断开连接事件。

主题	允许的客户端操作	描述
$aws/events/presence/disconnected/*clientId*	订阅	AWS IoT 当具有指定客户端 ID 的 MQTT 客户端断开连接时，将发布到此主题。AWS IoT有关更多信息，请参阅 连接/断开连接事件。
$aws/events/subscriptions/subscribed/*clientId*	订阅	AWS IoT 当具有指定客户端 ID 的 MQTT 客户端订阅 MQTT 主题时，将发布到此主题。有关更多信息，请参阅 订阅/取消订阅事件。
$aws/events/subscriptions/unsubscribed/*clientId*	订阅	AWS IoT 当具有指定客户端 ID 的 MQTT 客户端取消订阅 MQTT 主题时，将发布到此主题。有关更多信息，请参阅 订阅/取消订阅事件。
$aws/events/thing/*thingName* /created	订阅	AWS IoT 在创建 Thing *Name* *##*时向该主题发布。有关更多信息，请参阅 the section called “注册表事件”。
$aws/events/thing/*thingName* /updated	订阅	AWS IoT 当 Thing *Name* *##*更新时，将发布到此主题。有关更多信息，请参阅 the section called “注册表事件”。
$aws/events/thing/*thingName* /deleted	订阅	AWS IoT 删除 Thing *Name* *##*后，将发布到此主题。有关更多信息，请参阅 the section called “注册表事件”。
$aw *thingGroupName* s/events/ThingGroup//	订阅	AWS IoT 在创建事物组*thingGroupName* 时向该主题发布内容。有关更多信息，请参阅 the section called “注册表事件”。
$aw *thingGroupName* s/events/ThingGroup//	订阅	AWS IoT 事物组*thingGroupName* 更新时发布到此主题。有关更多信息，请参阅 the section called “注册表事件”。

主题	允许的客户端操作	描述
$aw *thingGroupName* s/events/ThingGroup//	订阅	AWS IoT 删除事物组*thingGroupName* 后，向该主题发布内容。有关更多信息，请参阅 the section called “注册表事件”。
$aw *thingTypeName* s/events/ThingType//	订阅	AWS IoT 在创建*thingTypeName* 事物类型时向该主题发布。有关更多信息，请参阅 the section called “注册表事件”。
$aw *thingTypeName* s/events/ThingType//	订阅	AWS IoT 在*thingTypeName* 事物类型更新时向该主题发布。有关更多信息，请参阅 the section called “注册表事件”。
$aw *thingTypeName* s/events/ThingType//	订阅	AWS IoT 删除*thingTypeName* 事物类型后，将发布到该主题。有关更多信息，请参阅 the section called “注册表事件”。
$aws/events/ / thing/ thingName/ thingTypeAssociation thingTypeName	订阅	AWS IoT 当事*# thingName #####*关联或与事物类型取消关联时，发布到此主题。*thingTypeName* 有关更多信息，请参阅 the section called “注册表事件”。
$aws/events/ / ThingGroup/ / thing/ thingName /ad thingGroupMembership ip ded thingGrou pName	订阅	AWS IoT 将事物 Thing *Name ######*时发布到此主题。*thingGroupName* 有关更多信息，请参阅 the section called “注册表事件”。
$aws/events/ / ThingGroup/ /thing/ thingName /##thingGroupMembership# thingGroupName	订阅	AWS IoT 将事物 Thing *Name ####*中移除后，将发布到该主题。*thingGroupName* 有关更多信息，请参阅 the section called “注册表事件”。

主题	允许的客户端操作	描述
thingGroupHierarchy*$aws/events/ /ThingGroup/ ##//## /addparentThingGroupedchildThingGroupchildThingGroup*	订阅	AWS IoT *########## childThingGroup## ########parentThingGroup* 有关更多信息，请参阅 the section called “注册表事件”。
thingGroupHierarchy*$aws/events/ /ThingGroup/ ##//## parentThingGroup /### childThingGroupchildThingGroup*	订阅	AWS IoT *########## childThingGroup## ##########parentThingGroup* 有关更多信息，请参阅 the section called “注册表事件”。

队列预置主题

> **Note**
>
> 此表中标为 “接收” 的客户端操作表示直接向请求它的客户端 AWS IoT 发布的主题，无论该客户是否订阅了该主题。即使客户端尚未订阅这些消息，也会收到这些消息。这些响应消息不会通过消息代理，也无法由其它客户端或规则订阅。

这些消息支持简明二进制对象表示 (CBOR) 格式和 JavaScript 对象表示法 (JSON) 的响应缓冲区，具体取决于主题的*####*格式。

payload-format	响应格式数据类型
cbor	简洁二进制对象表示法 (CBOR)
json	JavaScript 对象表示法 (JSON)

有关更多信息，请参阅 设备预调配 MQTT API。

主题	允许的客户端操作	描述
$aws/certificates/create/*payload-format*	Publish	发布到此主题以从证书签名请求 (CSR) 创建证书。
$aws/certificates/create/*payload-format* /accepted	订阅，接收	AWS IoT *#### $aws/cretificates/create/ payload-format ###### ##*
$aws/certificates/create/*payload-format* /rejected	订阅，接收	AWS IoT *### $aws/cretificates/create/ payload-format ###### ####*
$aws/certicates//payl create-from-csr oad-format	Publish	发布到此主题以从 CSR 创建证书。
$aws/##//######/create-from-csr ###	订阅，接收	AWS IoT *######## $aws/certificates//payload-create-from-csr format ######*
$aws/##//######/create-from-csr###	订阅，接收	AWS IoT *######## $aws/certificates//payload-create-from-csr format #########*
$aws/provisioning-templates/*templateName* /provision/*payload-format*	Publish	发布到此主题以注册事物。
$aws/provisioning-templates/*templateName* /provision/*payload-format* /accepted	订阅，接收	AWS IoT *#### $aws/provisioning-templates/templateName / provision/ payload-format ###### #####*
$aws/provisioning-templates/*templateN*	订阅，接收	AWS IoT *### $aws/provisioning-templates/templateName /*

主题	允许的客户端操作	描述
ame /provision/*payload-f ormat* /rejected		*provision/ payload-format ### ######*

任务主题

> **Note**
>
> 此表中标为 “接收” 的客户端操作表示直接向请求它的客户端 AWS IoT 发布的主题，无论该客户是否订阅了该主题。即使客户端尚未订阅这些消息，也会收到这些消息。
> 这些响应消息不会通过消息代理，也无法由其它客户端或规则订阅。要订阅与工作活动消息相关的任务活动，请使用 `notify` 和 `notify-next` 主题。
> 当订阅任务和您的机群监控解决方案的 `jobExecution` 事件主题时，您必须首先启用任务和任务执行事件接收云端的任何事件。
> 有关更多信息，请参阅 任务设备 MQTT API 操作。

主题	允许的客户端操作	描述
$aws/things/*thingName* /jobs/get	Publish	设备向此主题发布一条消息以发出 `GetPendingJobExecutions` 请求。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/things/*thingName* /jobs/get/accepted	订阅，接收	设备订阅此主题以接收来自 `GetPendingJobExecutions` 请求的成功响应。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/things/*thingName* /jobs/get/rejected	订阅，接收	当 `GetPendingJobExecutions` 请求被拒绝时，设备会订阅此主题以接收响应。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/things/*thingName* /jobs/start-next	Publish	设备向此主题发布一条消息以发出 `StartNextPendingJobExecutio`

主题	允许的客户端操作	描述
		n 请求。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/things/*thingName* / jobs/start-next/accepted	订阅，接收	设备订阅此主题以接收 StartNextPendingJobExecution 请求的成功响应。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/things/*thingName* / jobs/start-next/rejected	订阅，接收	当 StartNextPendingJobExecution 请求被拒绝时，设备会订阅此主题以接收响应。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/things/*thingName* / jobs/*jobId*/get	Publish	设备向此主题发布一条消息以发出 DescribeJobExecution 请求。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/things/*thingName* / jobs/*jobId*/get/accepted	订阅，接收	设备订阅此主题以接收 DescribeJobExecution 请求的成功响应。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/things/*thingName* / jobs/*jobId*/get/rejected	订阅，接收	当 DescribeJobExecution 请求被拒绝时，设备会订阅此主题以接收响应。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/things/*thingName* / jobs/*jobId*/update	Publish	设备向此主题发布一条消息以发出 UpdateJobExecution 请求。有关更多信息，请参阅 任务设备 MQTT API 操作。

主题	允许的客户端操作	描述
$aws/things/*thingName* /jobs/*jobId*/update/accepted	订阅，接收	设备订阅此主题以接收 UpdateJobExecution 请求的成功响应。有关更多信息，请参阅 任务设备 MQTT API 操作。 **备注** 仅向 $aws/things/*thingName* /jobs/*jobId*/update 发布消息的设备将收到有关此主题的消息。
$aws/things/*thingName* /jobs/*jobId*/update/rejected	订阅，接收	当 UpdateJobExecution 请求被拒绝时，设备会订阅此主题以接收响应。有关更多信息，请参阅 任务设备 MQTT API 操作。 **备注** 仅向 $aws/things/*thingName* /jobs/*jobId*/update 发布消息的设备将收到有关此主题的消息。
$aws/things/*thingName* /jobs/notify	订阅，接收	设备订阅此主题，以在某个事物的待处理执行列表中添加或删除了任务执行时接收通知。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/things/*thingName* /jobs/notify-next	订阅，接收	设备订阅此主题，以在事物的下一个待处理任务执行发生更改时接收通知。有关更多信息，请参阅 任务设备 MQTT API 操作。
$aws/events/job/*jobId*/completed	订阅	当某个任务完成时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。

主题	允许的客户端操作	描述
$aws/events/job/*jobId*/canceled	订阅	当取消某个任务时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。
$aws/events/job/*jobId*/deleted	订阅	当删除某个任务时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。
$aws/events/job/*jobId*/cancellation_in_progress	订阅	当开始取消某个任务时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。
$aws/events/job/*jobId*/deletion_in_progress	订阅	当开始删除某个任务时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobId*/succeeded	订阅	当任务执行成功时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobId*/failed	订阅	当任务执行失败时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobId*/rejected	订阅	当任务执行被拒绝时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobId*/canceled	订阅	当任务执行取消时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobId*/timed_out	订阅	当任务执行超时时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。

主题	允许的客户端操作	描述
$aws/events/jobExecution/*jobId*/removed	订阅	当任务执行被移除时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。
$aws/events/jobExecution/*jobId*/deleted	订阅	当任务执行被删除时，任务服务在此主题上发布事件。有关更多信息，请参阅 任务事件。

规则主题

主题	允许的客户端操作	描述
$aws/rules/*ruleName*	Publish	设备或应用程序向此主题发布消息以直接触发规则。有关更多信息，请参阅 借助基本摄取功能，降低消息收发成本。

安全隧道主题

主题	允许的客户端操作	描述
$aws/things/*thing-name* /tunnels/notify	订阅	AWS IoT 发布此消息以让 IoT 代理在远程设备上启动本地代理。有关更多信息，请参阅 the section called “IoT 代理代码段”。

影子主题

本节中的主题由命名和未命名的影子使用。每个影子使用的主题仅在主题前缀上有所不同。下表显示每种影子类型使用的主题前缀。

ShadowTopicPrefix 价值	影子类型
$aws/things/*thingName* /shadow	未命名的（经典）影子

ShadowTopicPrefix 价值	影子类型
$aws/things/*thingName* /shadow/name/*shadowName*	命名的影子

*##############ShadowTopicPrefix######## thing Name # sh adowName########## ######################*请记住，主题区分大小写。

主题	允许的客户端操作	描述
ShadowTopicPrefix /删除	发布/订阅	设备或应用程序向此主题发布消息以删除影子。有关更多信息，请参阅 /delete。
ShadowTopicPrefix /删除/接受	订阅	当一个影子被删除时，Device Shadow 服务将向此主题发送消息。有关更多信息，请参阅 /delete/accepted。
ShadowTopicPrefix /删除/已拒绝	订阅	当删除影子的请求遭拒时，Device Shadow 服务将向此主题发送消息。有关更多信息，请参阅 /delete/rejected。
ShadowTopicPrefix /get	发布/订阅	应用程序或事物向此主题发布空消息来获取影子。有关更多信息，请参阅 Device Shadow MQTT 主题。
ShadowTopicPrefix /get/接受	订阅	当获取影子的请求获批时，Device Shadow 服务将向此主题发送消息。有关更多信息，请参阅 /get/accepted。
ShadowTopicPrefix /get/被拒绝	订阅	当获取影子的请求遭拒时，Device Shadow 服务将向此主题发送消息。有关更多信息，请参阅 /get/rejected。
ShadowTopicPrefix /更新	发布/订阅	事物或应用程序向此主题发布消息以更新影子。有关更多信息，请参阅 /update。

主题	允许的客户端操作	描述
ShadowTopicPrefix /更新/已接受	订阅	当影子更新成功时，Device Shadow 服务将向此主题发送消息。有关更多信息，请参阅 /update/accepted。
ShadowTopicPrefix /更新/已拒绝	订阅	当影子更新遭拒时，Device Shadow 服务将向此主题发送消息。有关更多信息，请参阅 /update/rejected。
ShadowTopicPrefix /update/delta	订阅	当检测到影子的“reported”部分与“desired”部分之间存在差异时，Device Shadow 服务将向此主题发送消息。有关更多信息，请参阅 /update/delta。
ShadowTopicPrefix /更新/文档	订阅	AWS IoT 每当成功执行影子更新时，都会向该主题发布状态文档。有关更多信息，请参阅 /update/documents。

基于 MQTT 的文件传输主题

> **Note**
>
> 此表中标为 “接收” 的客户端操作表示直接向请求它的客户端 AWS IoT 发布的主题，无论该客户是否订阅了该主题。即使客户端尚未订阅这些消息，也会收到这些消息。这些响应消息不会通过消息代理，也无法由其它客户端或规则订阅。

这些消息支持简明二进制对象表示 (CBOR) 格式和 JavaScript 对象表示法 (JSON) 的响应缓冲区，具体取决于主题的*####*格式。

payload-format	响应格式数据类型
cbor	简洁二进制对象表示法 (CBOR)
json	JavaScript 对象表示法 (JSON)

主题	允许的客户端操作	描述
$aws/things/ / streams/ /data/ # ##### ThingName StreamId	订阅，接收	AWS 如果设备的 “GetStream” 请求被接受，则基于 MQTT 的文件传输会发布到此主题。负载包含流数据。有关更多信息，请参阅 在设备中使用 AWS IoT 基于 MQTT 的文件传输。
$aws/things/ / streams/ /get/ payl ThingName oad-forma t StreamId	Publish	设备向该主题发布消息以执行 “GetStream” 请求。有关更多信息，请参阅 在设备中使用 AWS IoT 基于 MQTT 的文件传输。
$aws/things/ / streams/ /descrip ThingName tion/ #### ## StreamId	订阅，接收	AWS 如果设备的 “DescribeStream” 请求被接受，则基于 MQTT 的文件传输会发布到此主题。负载包含流描述。有关更多信息，请参阅 在设备中使用 AWS IoT 基于 MQTT 的文件传输。
$aws/things/ / streams/ /describ ThingName e/ ###### StreamId	Publish	设备向该主题发布消息以执行 “DescribeStream” 请求。有关更多信息，请参阅 在设备中使用 AWS IoT 基于 MQTT 的文件传输。
$aws/things/ / streams/ /rejec ThingName ted/ payload-format StreamId	订阅，接收	AWS 如果来自设备的 “” 或 “Describe StreamGetStream” 请求被拒绝，则基于 MQTT 的文件传输会发布到此主题。有关更多信息，请参阅 在设备中使用 AWS IoT 基于 MQTT 的文件传输。

保留的主题 ARN

所有保留的主题 ARN (Amazon Resource Name) 均采用以下形式：

```
arn:aws:iot:aws-region:AWS-account-ID:topic/Topic
```

例如，`arn:aws:iot:us-west-2:123EXAMPLE456:topic/$aws/things/thingName/jobs/get/accepted` 是保留主题 `$aws/things/thingName/jobs/get/accepted` 的 ARN。

可配置的终端节点

在中 AWS IoT Core，您可以使用域配置来配置和管理数据端点的行为。通过域配置，您可以生成多个 AWS IoT Core 数据端点，使用您自己的完全限定域名 (FQDN) 和关联的服务器证书自定义这些数据端点，还可以关联自定义授权方。有关更多信息，请参阅 自定义身份验证和授权。

> **Note**
>
> 此功能在中不可用 GovCloud AWS 区域。

域配置用例

您可以使用域配置来简化以下这样的任务。

- 将设备迁移到 AWS IoT Core。
- 通过为不同的设备类型维护单独的域配置来支持异构设备实例集。
- 在将应用程序基础架构迁移到的同时，保持品牌标识（例如，通过域名） AWS IoT Core。

在中使用域配置的重要注意事项 AWS IoT Core

AWS IoT Core 使用服务器名称指示 (SNI) TLS 扩展来应用域配置。设备在连接时必须使用此扩展名 AWS IoT Core。它们还必须传递与域配置中指定的域名相同的服务器名称。要测试此服务，请使用中AWS IoT 设备软件开发工具包的 v2 版本。 GitHub

如果您在中创建多个数据端点 AWS 账户，它们将共享 MQTT 主题、设备影子和规则等 AWS IoT Core 资源。

当您为 AWS IoT Core 自定义域配置提供服务器证书时，这些证书最多有四个域名。有关更多信息，请参阅 AWS IoT Core 终端节点和限额。

在本章中：

- 创建和配置 AWS 托管域
- 创建和配置自定义域

- 管理域配置
- 在域配置中配置 TLS 设置
- OCSP 装订的服务器证书配置

创建和配置 AWS 托管域

您可以使用 CreateDomainConfigurationAPI 在 AWS 托管域上创建可配置的终端节点。 AWS 托管域的域配置包括以下内容：

- `domainConfigurationName`

  用于标识域配置的用户定义名称和值必须是唯一的 AWS 区域。您无法使用以 `IoT:` 开头的域配置名称，因为它们保留用于默认端点。

- `defaultAuthorizerName`（可选）

  要在端点上使用的自定义授权方的名称。

- `allowAuthorizerOverride`（可选）

  一个布尔值，指定设备是否可以通过在请求的 HTTP 标头中指定不同的授权方来覆盖默认授权方。如果为 `defaultAuthorizerName` 指定了值，则需要此值。

- `serviceType`（可选）

  终端节点提供的服务类型。 AWS IoT Core 仅支持DATA服务类型。如果指定 `DATA`，则 AWS IoT Core 返回端点类型为 `iot:Data-ATS` 的端点。您无法创建可配置的 `iot:Data` (VeriSign) 端点。

- `TlsConfig`（可选）

  为域指定 TLS 配置的对象。有关更多信息，请参阅 ???。

以下示例 AWS CLI 命令为Data终端节点创建域配置。

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" --service-type "DATA"
```

该命令的输出可能如下所示。

```
{
    "domainConfigurationName": "myDomainConfigurationName",
```

```
    "domainConfigurationArn": "arn:aws:iot:us-east-1:123456789012:domainconfiguration/
myDomainConfigurationName/itihw"
}
```

创建和配置自定义域

域配置允许您指定要连接到 AWS IoT Core的自定义完全限定域名 (FQDN)。使用自定义域名有很多好处：您可以出于品牌推广目的向客户公开自己的域名或公司自己的域名；您可以轻松地将自己的域名更改为指向新的经纪商；您可以支持多租户为同一域中的不同域名的客户提供服务 AWS 账户；您可以管理自己的服务器证书详细信息，例如用于签署证书的根证书颁发机构 (CA)、签名算法、证书链深度以及证书链的生命周期证书。

使用自定义域设置域配置的工作流包括以下三个阶段。

1. 在中注册服务器证书 AWS Certificate Manager
2. 创建域配置
3. 创建 DNS 记录

在证书管理器中注册服务器 AWS 证书

在使用自定义域创建域配置之前，必须在 AWS Certificate Manager (ACM) 中注册服务器证书链。您可以使用以下三种类型的服务器证书。

- ACM 生成的公有证书
- 由公有 CA 签名的外部证书
- 由私有 CA 签名的外部证书

> **Note**
>
> AWS IoT Core 如果证书包含在 Mozilla 的可信 ca-bundle 中，则认为该证书由公共 C A 签名。

证书要求

请参阅导入证书的先决条件，了解将证书导入 ACM 的要求。除了这些要求之外， AWS IoT Core 还添加了以下要求。

- 树叶证书必须包含扩展密钥用法 x509 v3 扩展，其值为 ServerAuth（TLS Web 服务器身份验证）。如果您从 ACM 请求证书，则会自动添加此扩展。
- 最大证书链深度为 5 个证书。
- 最大证书链大小为 16KB。
- 支持的加密算法和密钥大小包括 RSA 2048 位 (RSA_2048) 和 ECDSA 256 位 (ec_Prime256v1)。

对多个域使用一个证书

如果您计划使用一个证书来涵盖多个子域，请在公用名 (CN) 或使用者备用名称 (SAN) 字段中使用通配符域。例如，使用 **`*.iot.example.com`** 涵盖 cover dev.iot.example.com、qa.iot.example.com 和 prod.iot.example.com。每个 FQDN 都需要自己的域配置，但多个域配置可以使用同一个通配符值。CN 或 SAN 必须涵盖要用作自定义域的 FQDN。如果存在 SAN，会忽略 CN，SAN 必须覆盖您想要用作自定义域的 FQDN。此涵盖范围可以是完全匹配或通配符匹配。在验证通配符证书并将其注册到账户后，将阻止区域中的其他账户创建与证书重叠的自定义域。

以下各部分介绍如何获取每种类型的证书。每个证书资源都需要一个使用 ACM 注册的 Amazon Resource Name (ARN)，该 ACM 即您在创建域配置时所用的那个。

ACM 生成的公有证书

您可以使用 RequestCertificateAPI 为自定义域生成公共证书。以这种方式生成证书时，ACM 将验证您对自定义域的所有权。有关更多信息，请参阅 AWS Certificate Manager 用户指南中的请求公有证书。

由公有 CA 签名的外部证书

如果您已经拥有由公共 CA（包含在 Mozilla 可信 ca-bundle 中的 CA）签署的服务器证书，则可以使用 API 将证书链直接导入 ACM。ImportCertificate要了解有关此任务以及先决条件和证书格式要求的更多信息，请参阅导入证书。

由私有 CA 签名的外部证书

如果您已经具有由私有 CA 签名或自签名的服务器证书，则可以使用该证书创建域配置，但是还必须在 ACM 中创建一个额外的公有证书以验证域的所有权。为此，请使用 ImportCertificateAPI 在 ACM 中注册您的服务器证书链。要了解有关此任务以及先决条件和证书格式要求的更多信息，请参阅导入证书。

创建验证证书

将证书导入 ACM 后，使用 RequestCertificateAPI 为您的自定义域生成公有证书。以这种方式生成证书时，ACM 将验证您对自定义域的所有权。有关更多信息，请参阅请求公有证书。创建域配置时，请使用此公有证书作为验证证书。

创建域配置

您可以使用 CreateDomainConfigurationAPI 在自定义网域上创建可配置的终端节点。自定义域的域配置包括以下内容：

- `domainConfigurationName`

  用于标识域配置的用户定义名称。以 `IoT:` 开头的域配置名称将针对默认终端节点保留，不能使用。此外，此值必须是您的唯一值 AWS 区域。

- `domainName`

  您的设备用于连接的 FQDN。 AWS IoT Core AWS IoT Core 利用服务器名称指示 (SNI) TLS 扩展来应用域配置。设备在连接时必须使用此扩展名，并传递与域配置中指定的域名相同的服务器名称。

- `serverCertificateArns`

  您向 ACM 注册的服务器证书链的 ARN。 AWS IoT Core 目前仅支持一个服务器证书。

- `validationCertificateArn`

  您在 ACM 中生成的用于验证自定义域所有权的公有证书的 ARN。如果您使用公开签名或 ACM 生成的服务器证书，则不需要此参数。

- `defaultAuthorizerName (optional)`

  要在端点上使用的自定义授权方的名称。

- `allowAuthorizerOverride`

  一个布尔值，指定设备是否可以通过在请求的 HTTP 标头中指定不同的授权方来覆盖默认授权方。如果为 `defaultAuthorizerName` 指定了值，则需要此值。

- `serviceType`

  AWS IoT Core 目前仅支持DATA服务类型。如果指定DATA， AWS IoT 则返回终端节点类型为的端点`iot:Data-ATS`。

- `TlsConfig`（可选）

  为域指定 TLS 配置的对象。有关更多信息，请参阅 ???。

- `serverCertificateConfig`（可选）

  一个对象，用于指定域的服务器证书配置。有关更多信息，请参阅 ???。

以下 AWS CLI 命令为 iot .example.com 创建域配置。

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" --service-type "DATA"
--domain-name "iot.example.com" --server-certificate-arns serverCertARN --validation-
certificate-arn validationCertArn
```

> **Note**
>
> 创建域配置后，最多可能需要 60 分钟才能 AWS IoT Core 提供您的自定义服务器证书。

有关更多信息，请参阅 ???。

创建 DNS 记录

注册服务器证书链并创建域配置后，请创建 DNS 记录，以便您的自定义域指向 AWS IoT 域。此记录必须指向类型为的 AWS IoT 端点`iot:Data-ATS`。您可以使用 DescribeEndpointAPI 获取终端节点。

以下 AWS CLI 命令显示如何获取终端节点。

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

获取`iot:Data-ATS`终端节点后，创建从您的自定义域名到该 AWS IoT 终端节点的`CNAME`记录。如果您在同一个域中创建多个自定义域名 AWS 账户，请将它们别名为同一个`iot:Data-ATS`终端节点。

故障排除

如果您在将设备连接到自定义域时遇到问题，请确保该域 AWS IoT Core 已接受并应用您的服务器证书。您可以使用 AWS IoT Core 控制台或，验证是否 AWS IoT Core 已接受您的证书 AWS CLI。

要使用 AWS IoT Core 控制台，请导航到 “设置” 页面并选择域配置名称。在 Server certificate details（服务器证书详细信息）部分中，检查状态和状态详细信息。如果证书无效，请将其在 ACM 中替换为符合上一部分中列示的证书要求的证书。如果证书具有相同的 ARN，则 AWS IoT Core 会自动领取并应用。

要使用检查证书状态 AWS CLI，请调用 DescribeDomainConfigurationAPI 并指定您的域配置名称。

> **Note**
>
> 如果您的证书无效，则 AWS IoT Core 将继续提供最后一个有效证书。

您可以使用以下 openssl 命令检查端点上正在提供哪些证书。

```
openssl s_client -connect custom-domain-name:8883 -showcerts -servername
custom-domain-name
```

管理域配置

您可以使用以下 API 管理现有配置的生命周期。

- ListDomainConfigurations
- DescribeDomainConfiguration
- UpdateDomainConfiguration
- DeleteDomainConfiguration

查看域配置

要返回您的中所有域名配置的分页列表 AWS 账户，请使用 ListDomainConfigurationsAPI。您可以使用 DescribeDomainConfigurationAPI 查看特定域配置的详细信息。此 API 采用单个 domainConfigurationName 参数并返回指定配置的详细信息。

示例

更新域配置

要更新您的域名配置的状态或自定义授权者，请使用 UpdateDomainConfigurationAPI。您可以将状态设置为 ENABLED 或 DISABLED。如果您禁用域配置，连接到该域的设备将收到身份验证错误。目前，您无法更新域配置中的服务器证书。要更改域配置的证书，必须删除并重新创建该证书。

示例

删除域配置

在删除域配置之前，请使用 UpdateDomainConfigurationAPI 将状态设置为DISABLED。这有助于避免意外删除终端节点。禁用域名配置后，使用 DeleteDomainConfigurationAPI 将其删除。您必须将 AWS 托管域名置于 7 天DISABLED状态，然后才能将其删除。您可以将自定义域名置于DISABLED状态，然后立即将其删除。

示例

删除域配置后， AWS IoT Core 不再提供与该自定义域关联的服务器证书。

在自定义域中轮换证书

您可能需要定期使用更新的证书替换服务器证书。您执行此操作的速率取决于证书的有效期。如果您使用 AWS Certificate Manager (ACM) 生成服务器证书，您可以将证书设置为自动续订。当 ACM 续订您的证书时，AWS IoT Core 会自动领取新证书。您无需执行任何其它操作。如果从其它源导入服务器证书，则可以通过将其重新导入到 ACM 来轮换该证书。有关重新导入证书的信息，请参阅重新导入证书。

> **Note**
>
> AWS IoT Core 仅在以下条件下才会获取证书更新。
>
> - 新证书具有与旧证书相同的 ARN。
> - 新证书具有与旧证书相同的签名算法、公用名称或备用名称。

在域配置中配置 TLS 设置

AWS IoT Core 提供了预定义的安全策略，供您在域配置中自定义 TLS 1.2 和 TLS 1. 3 的传输层安全 (TLS) 设置。安全策略是 TLS 协议及其密码的组合，此协议及其密码用于确定在客户端和服务器之间的 TLS 协商期间所支持的协议和密码。借助支持的安全策略，您可以更灵活地管理设备的 TLS 设置，在连接新设备时采用最严格的 up-to-date 安全措施，并为现有设备保持一致的 TLS 配置。

下表描述了安全策略、其 TLS 版本和支持的区域：

安全策略名称	支持 AWS 区域
物联网 SecurityPolicy_TLS13_1_1_3_2022_10	全部 AWS 区域
物联网 SecurityPolicy_TLS13_1_1_2_2022_10	全部 AWS 区域
物联网 SecurityPolicy_TLS12_1_1_2_2022_10	全部 AWS 区域

安全策略名称	支持 AWS 区域
物联网 SecurityPolicy_TLS12_1_1_0_2016_01	ap-east-1、ap-northeast-2、ap-southeast-1、ap-southeast-2、ca-central-1、cn-northeast-1、eu-northeast-1、eu-northeast-1、eu-west-3、me-south-1、sa-east-1、us-east-2、us-west-1
物联网 SecurityPolicy_TLS12_1_1_0_2015_01	ap-northeast-1、ap-southeast-1、eu-central-1、eu-west-1、us-east-1、us-west-2

中的安全策略名称 AWS IoT Core 包括基于发布年份和月份的版本信息。如果您创建新的域配置，则安全策略将默认为 `IoTSecurityPolicy_TLS13_1_2_2022_10`。有关包含协议、TCP 端口和密码详细信息的安全策略的完整表，请参阅安全策略。 AWS IoT Core 不支持自定义安全策略。有关更多信息，请参阅 ???。

要在域配置中配置 TLS 设置，您可以使用 AWS IoT 控制台或 AWS CLI。

内容

- 在域配置中配置 TLS 设置（控制台）
- 在域配置中配置 TLS 设置（CLI）

在域配置中配置 TLS 设置（控制台）

使用 AWS IoT 控制台配置 TLS 设置

1. 登录 AWS Management Console 并打开AWS IoT 控制台。
2. 要在创建新的域配置时配置 TLS 设置，请按照以下步骤操作。
    1. 在左侧导航窗格中，选择设置，然后从域配置部分选择创建域配置。
    2. 在创建域配置页面的自定义域设置 - 可选部分，从选择安全策略中选择安全策略。
    3. 按照小部件操作并完成其余步骤。选择创建域配置。
3. 要更新现有域配置中的 TLS 设置，请按照以下步骤操作。
    1. 在左侧导航窗格中，选择设置，然后在域配置下选择域配置。
    2. 在域配置详细信息页面中，选择编辑。然后，在自定义域设置 - 可选部分的选择安全策略下，选择安全策略。

3. 选择更新域配置。

有关更多信息，请参阅创建域配置和管理域配置。

在域配置中配置 TLS 设置（CLI）

您可以使用 create-domain-configuration 和 update-domain-configuration CLI 命令在域配置中配置 TLS 设置。

1. 要使用 create-domain-configuration CLI 命令指定 TLS 设置，请执行以下操作：

```
aws iot create-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

此命令的输出可能如下所示：

```
{
"domainConfigurationName": "test",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

如果您在未指定安全策略的情况下创建新的域配置，则该值将默认为：IoTSecurityPolicy_TLS13_1_2_2022_10。

2. 要使用 describe-domain-configuration CLI 命令描述 TLS 设置，请执行以下操作：

```
aws iot describe-domain-configuration \
    --domain-configuration-name domainConfigurationName
```

此命令可以返回包含 TLS 设置的域配置详细信息，如下所示：

```
{
 "tlsConfig": {
 "securityPolicy": "IoTSecurityPolicy_TLS13_1_2_2022_10"
 },
 "domainConfigurationStatus": "ENABLED",
 "serviceType": "DATA",
 "domainType": "AWS_MANAGED",
 "domainName": "d1234567890abcdefghij-ats.iot.us-west-2.amazonaws.com",
```

```
 "serverCertificates": [],
 "lastStatusChangeDate": 1678750928.997,
 "domainConfigurationName": "test",
 "domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

3. 要使用 update-domain-configuration CLI 命令更新 TLS 设置，请执行以下操作：

```
aws iot update-domain-configuration \
    --domain-configuration-name domainConfigurationName \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

此命令的输出可能如下所示：

```
{
"domainConfigurationName": "test",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
test/34ga9"
}
```

4. 要更新 ATS 端点的 TLS 设置，请运行 update-domain-configuration CLI 命令。您的 ATS 端点的域配置名称为 `iot:Data-ATS`。

```
aws iot update-domain-configuration \
    --domain-configuration-name "iot:Data-ATS" \
    --tls-config securityPolicy=IoTSecurityPolicy_TLS13_1_2_2022_10
```

此命令的输出可能如下所示：

```
{
"domainConfigurationName": "iot:Data-ATS",
"domainConfigurationArn": "arn:aws:iot:us-west-2:123456789012:domainconfiguration/
iot:Data-ATS"
}
```

有关更多信息，请参阅 AWS API 参考UpdateDomainConfiguration中的CreateDomainConfiguration和。

OCSP 装订的服务器证书配置

AWS IoT Core 支持服务器证书的联机证书状态协议 (OCSP) 装订，也称为服务器证书 OCSP 装订或 OCSP 装订。它是一种安全机制，用于在传输层安全 (TLS) 握手中检查服务器证书的吊销状态。OCSP 装订 AWS IoT Core 允许您为自定义域的服务器证书有效性添加额外的验证层。

您可以通过定期查询 OCSP 响应器 AWS IoT Core 来启用服务器证书 OCSP 装订以检查证书的有效性。OCSP 装订设置是使用自定义域创建或更新域配置的过程的一部分。OCSP 装订会持续检查服务器证书上的吊销状态。这有助于验证已被 CA 吊销的所有证书是否不再受到连接到您的自定义域的客户端的信任。有关更多信息，请参阅 ???。

服务器证书 OCSP 装订提供实时吊销状态检查，减少与检查吊销状态相关的延迟，并提高安全连接的隐私性和可靠性。有关使用 OCSP 装订的好处的更多信息，请参阅。???

> **Note**
>
> 此功能在中不可用 AWS GovCloud (US) Regions。

本主题内容：

- 什么是 OCSP？
- OCSP 装订的工作原理
- 启用服务器证书 OCSP 装订 AWS IoT Core
- 在中使用服务器证书 OCSP 装订的重要注意事项 AWS IoT Core
- 对服务器证书 OCSP 装订进行故障排除 AWS IoT Core

什么是 OCSP？

重要概念

以下概念提供了有关 OCSP 和相关概念的详细信息。

OCSP

OCSP 用于在传输层安全 (TLS) 握手期间检查证书吊销状态。OCSP 允许对证书进行实时验证。这可以确认证书自颁发以来没有被吊销或过期。与传统的证书吊销列表 (CRL) 相比，OCSP 的扩展性也更强。OCSP 响应较小，可以高效生成，因此更适合大型私钥基础架构 (PKI)。

OCSP 响应器

OCSP 响应器（也称为 OCSP 服务器）接收并响应来自寻求验证证书吊销状态的客户端的 OCSP 请求。

客户端 OCSP

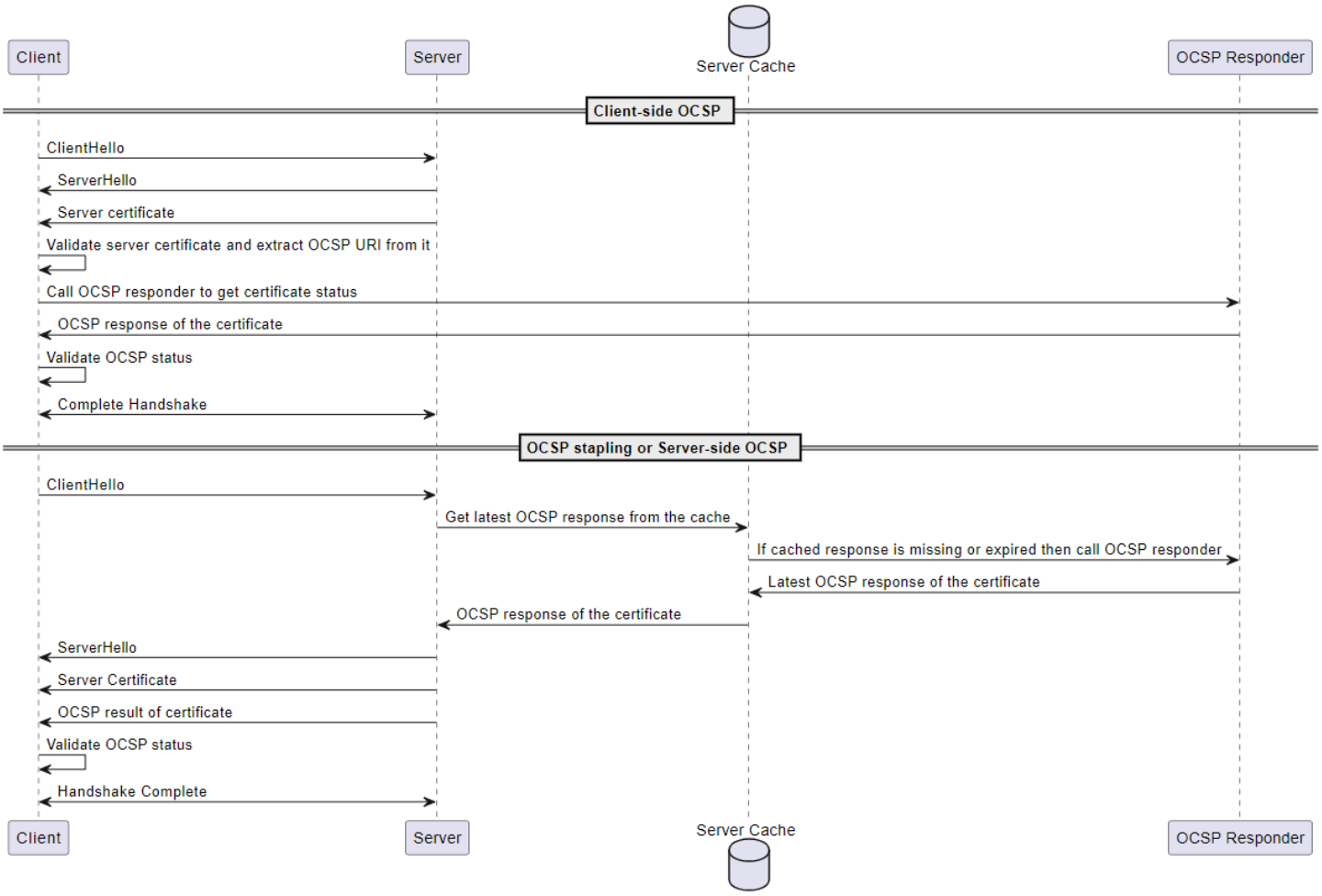
在客户端 OCSP 中，在传输层安全 (TLS) 握手期间，客户端使用 OCSP 与 OCSP 响应者联系，以检查证书的吊销状态。

服务器端 OCSP

在服务器端 OCSP（也称为 OCSP 装订）中，允许服务器（而不是客户端）向 OCSP 响应者发出请求。服务器将 OCSP 响应绑定到证书，然后在 TLS 握手期间将其返回给客户端。

OCSP 图表

下图说明了客户端 OCSP 和服务器端 OCSP 的工作原理。



客户端 OCSP

1. 客户端发送一条ClientHello消息以启动与服务器的 TLS 握手。
2. 服务器收到消息并以ServerHello消息进行响应。服务器还将服务器证书发送给客户端。
3. 客户端验证服务器证书并从中提取 OCSP URI。
4. 客户端向 OCSP 响应者发送证书吊销检查请求。
5. OCSP 响应者发送 OCSP 响应。
6. 客户端验证来自 OCSP 响应的证书状态。
7. TLS 握手已完成。

服务器端 OCSP

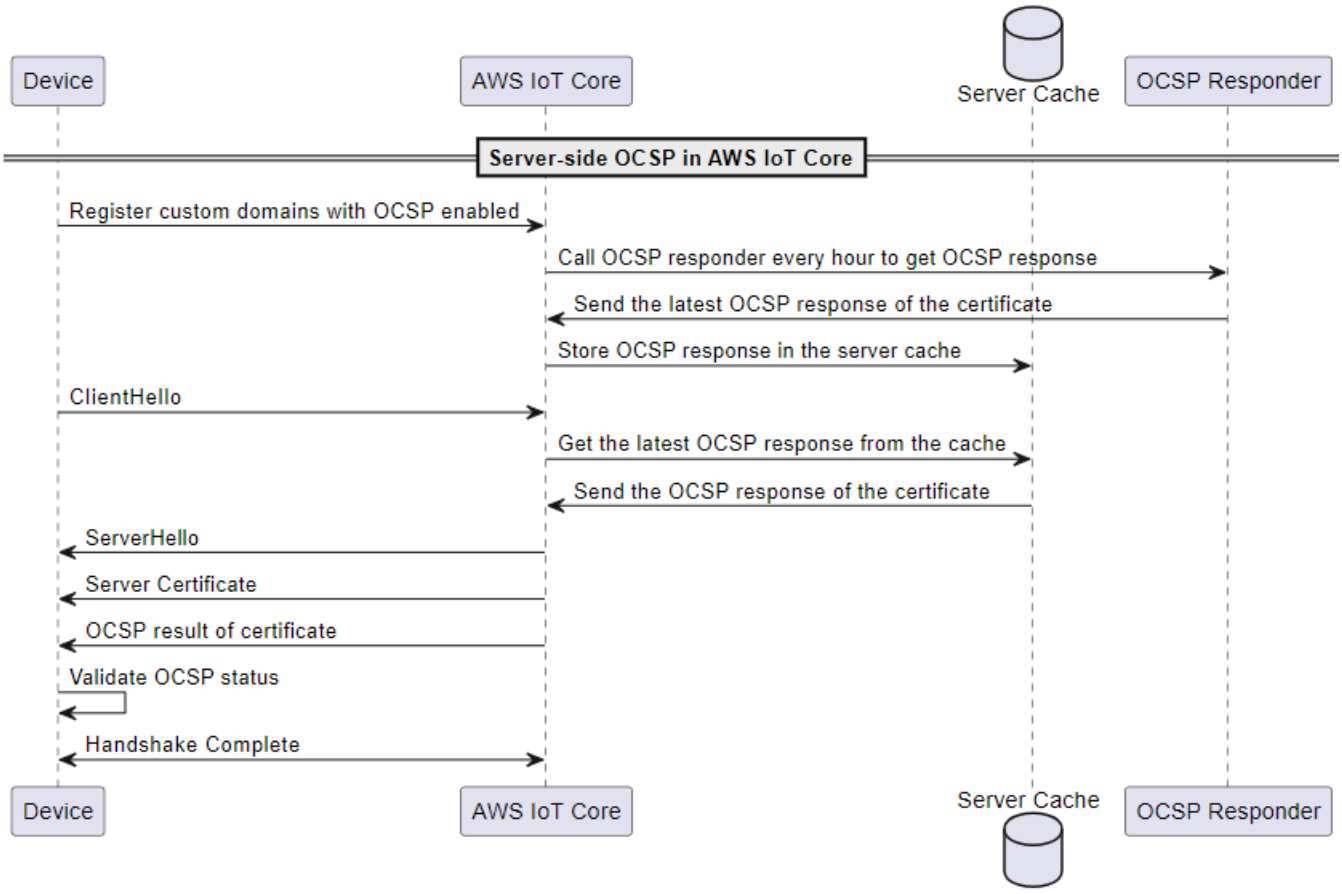
1. 客户端发送一条ClientHello消息以启动与服务器的 TLS 握手。
2. 服务器收到消息并获取最新缓存的 OCSP 响应。如果缓存的响应丢失或已过期，服务器将调用 OCSP 响应器以获取证书状态。
3. OCSP 响应器向服务器发送 OCSP 响应。
4. 服务器发送ServerHello消息。服务器还将服务器证书和证书状态发送给客户端。
5. 客户端验证 OCSP 证书状态。
6. TLS 握手已完成。

OCSP 装订的工作原理

在客户端和服务器之间的传输层安全 (TLS) 握手期间，使用 OCSP 装订来检查服务器证书吊销状态。服务器向 OCSP 响应者发出 OCSP 请求，并将 OCSP 响应绑定到返回给客户端的证书。通过让服务器向 OCSP 响应者发出请求，可以缓存响应，然后多次用于许多客户端。

OCSP 装订的工作原理 AWS IoT Core

下图显示了服务器端 OCSP 装订的工作原理。 AWS IoT Core



1. 设备需要在启用 OCSP 装订的自定义域中注册。
2. AWS IoT Core 每小时呼叫 OCSP 响应者以获取证书状态。
3. OCSP 响应者接收请求，发送最新的 OCSP 响应，并存储缓存的 OCSP 响应。
4. 设备发送一条`ClientHello`消息以启动 TLS 握手 AWS IoT Core。
5. AWS IoT Core 从服务器缓存中获取最新的 OCSP 响应，服务器缓存以证书的 OCSP 响应进行响应。
6. 服务器向设备发送`ServerHello`消息。服务器还将服务器证书和证书状态发送给客户端。
7. 设备验证 OCSP 证书状态。
8. TLS 握手已完成。

与客户端 OCSP 检查相比，使用 OCSP 装订的好处

使用服务器证书 OCSP 装订的一些优点总结如下：

改善了隐私

如果没有 OCSP 装订，客户端的设备可能会将信息暴露给第三方 OCSP 响应者，从而可能危及用户隐私。OCSP 装订通过让服务器获取 OCSP 响应并将其直接传送到客户端来缓解此问题。

提高了可靠性

OCSP 装订可以提高安全连接的可靠性，因为它可以降低 OCSP 服务器中断的风险。对 OCSP 响应进行装订时，服务器会将最新的响应与证书一起包括在内。这样，即使 OCSP 响应器暂时不可用，客户端也可以访问撤销状态。OCSP 装订有助于缓解这些问题，因为服务器会定期获取 OCSP 响应并将缓存的响应包含在 TLS 握手中，从而减少了对 OCSP 响应器实时可用性的依赖。

减少服务器负载

OCSP 装订将响应 OCSP 响应者的 OCSP 请求的负担转移到服务器。这有助于更均匀地分配负载，从而提高证书验证过程的效率和可扩展性。

减少延迟

OCSP 装订减少了 TLS 握手期间与检查证书吊销状态相关的延迟。客户端不必单独查询 OCSP 服务器，而是在握手期间发送请求并附上 OCSP 响应和服务器证书。

启用服务器证书 OCSP 装订 AWS IoT Core

要启用服务器证书 OCSP 装订 AWS IoT Core，必须为自定义域创建域配置或更新现有的自定义域配置。有关使用自定义域创建域配置的一般信息，请参阅???。

按照以下说明使用或 AWS Management Console 启用 OCSP 服务器装订。 AWS CLI

控制台

要使用 AWS IoT 控制台启用服务器证书 OCSP 装订，请执行以下操作：

1. 从菜单的左侧导航栏中选择 “设置”，然后选择为自定义域名创建域配置或现有域配置。
2. 如果您选择在上一步中创建新的域配置，则会看到创建域配置页面。在域配置属性部分中，选择自定义域。输入信息以创建域配置。

   如果您选择更新自定义域的现有域配置，您将看到域配置详细信息页面。选择编辑。
3. 要启用 OCSP 服务器装订，请在 “服务器证书配置” 小节中选择 “启用服务器证书 OCSP 装订”。
4. 选择创建域配置或更新域配置。

AWS CLI

要启用服务器证书 OCSP 装订，请使用以下命令： AWS CLI

1. 如果您为自定义域创建新的域配置，则启用 OCSP 服务器装订的命令可能如下所示：

```
aws iot create-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true|false"
```

2. 如果您更新自定义域的现有域配置，则启用 OCSP 服务器装订的命令可能如下所示：

```
aws iot update-domain-configuration --domain-configuration-name
 "myDomainConfigurationName" \
        --server-certificate-arns arn:aws:iot:us-
east-1:123456789012:cert/
f8c1e5480266caef0fdb1bf97dc1c82d7ba2d3e2642c5f25f5ba364fc6b79ba3 \
        --server-certificate-config "enableOCSPCheck=true|false"
```

有关更多信息，请参阅 AWS IoT API 参考UpdateDomainConfiguration中的CreateDomainConfiguration和。

在中使用服务器证书 OCSP 装订的重要注意事项 AWS IoT Core

在中使用服务器证书 OCSP 时 AWS IoT Core，请记住以下几点：

1. AWS IoT Core 仅支持那些可通过公有 IPv4 地址访问的 OCSP 响应器。
2. 中的 OCSP 装订功能 AWS IoT Core 不支持授权响应者。所有 OCSP 响应都必须由签署证书的 CA 签名，并且 CA 必须是自定义域证书链的一部分。
3. 中的 OCSP 装订功能 AWS IoT Core 不支持使用自签名证书创建的自定义域。
4. AWS IoT Core 每小时呼叫 OCSP 响应者并缓存响应。如果对应答者的呼叫失败， AWS IoT Core 将装订最新的有效回复。
5. 如果不再有效，`nextUpdateTime`则 AWS IoT Core 将从缓存中删除响应，并且在下次成功调用 OCSP 响应器之前，TLS 握手将不包含 OCSP 响应数据。当缓存的响应在服务器从 OCSP 响应器

获得有效响应之前已过期时，就会发生这种情况。的值`nextUpdateTime`表明 OCSP 响应在此之前将一直有效。有关 `nextUpdateTime` 的更多信息，请参阅???。

6. 有时， AWS IoT Core 无法收到 OCSP 响应或删除现有的 OCSP 响应，因为该响应已过期。如果发生此类情况， AWS IoT Core 将继续使用自定义域提供的服务器证书，而不会获得 OCSP 响应。
7. OCSP 响应的大小不能超过 4 KiB。

对服务器证书 OCSP 装订进行故障排除 AWS IoT Core

AWS IoT Core 将`RetrieveOCSPStapleData.Success`指标和`RetrieveOCSPStapleData`日志条目发送到。 CloudWatch指标和日志条目可以帮助检测与检索 OCSP 响应有关的问题。有关更多信息，请参阅 ??? 和 ???。

连接到 AWS IoT FIPS 终端节点

AWS IoT 提供支持联邦信息处理标准 (FIPS) 140 -2 的端点。符合 FIPS 标准的端点与标准 AWS 端点不同。要以符合 FIPS 的方式与 AWS IoT 进行交互，您必须将下面描述的终端节点与符合FIPS的客户端一起使用。该 AWS IoT 主机不符合 FIPS 标准。

以下各节介绍如何使用 REST API、软件开发工具包或访问符合 FIPS 标准的 AWS IoT 终端节点。 AWS CLI

主题

- AWS IoT Core - 控制面板终端节点
- AWS IoT Core - 数据层面终端节点
- AWS IoT Device Management - 任务数据终端节点
- AWS IoT Device Management - Fleet Hub 终端节点
- AWS IoT Device Management - 安全隧道终端节点

AWS IoT Core - 控制面板终端节点

符合 FIPS 标准AWS IoT Core - 控制面板支持AWS IoT操作及其相关CLI 命令列于按服务划分 FIPS 终端节点中。在按服务划分 FIPS 终端节点中，查找AWS IoT Core - 控制面板服务，然后查找您 AWS 区域的终端节点。

要在访问AWS IoT操作时使用符合 FIPS 标准的终端节点，请使用 AWS SDK 或 REST API 以及适合您的 AWS 区域终端节点。

要在运行 aws iot CLI命令时使用符合 FIPS 标准的终端节点，请将具有适合您 AWS 区域 终端节点的 --endpoint 参数添加到命令中。

AWS IoT Core - 数据层面终端节点

符合 FIPS 标准的 AWS IoT Core - 数据层面在按服务划分 FIPS 终端节点中列出终端节点。在按服务划分 FIPS 终端节点中，查找 AWS IoT Core - 数据层面服务，然后查找您 AWS 区域的终端节点。

您可以将符合 FIPS 的终端节点 AWS 区域 与兼容 FIPS 的客户端配合使用，方法是使用 AWS IoT 设备 SDK 的连接功能提供终端节点，而不是账户的默认AWS IoT Core数据平面端点。连接功能特定于 AWS IoT 设备 SDK。有关连接函数的示例，请参阅适用于 Python 的 AWS IoT 设备 SDK 中的连接函数。

> **Note**
>
> AWS IoT 不支持符合 AWS 账户 FIP AWS IoT Core S 的特定数据平面端点。无法使用需要在服务器名称指示 (SNI) 中使用 AWS 账户特定端点的服务功能。符合 FIPS 标准的 AWS IoT Core- 数据面板端点无法支持多账户注册证书、自定义域、自定义授权方和可配置端点（包括支持的 TLS 策略）。

AWS IoT Device Management - 任务数据终端节点

符合 FIPS 标准AWS IoT Device Management - 任务数据在按服务划分 FIPS 终端节点中列出了终端节点。在按服务划分 FIPS 终端节点中，查找 AWS IoT Device Management - 任务数据服务，然后查找您 AWS 区域的终端节点。

要在运行 aws iot-jobs-data CLI 命令时使用符合 FIPS 标准的 AWS IoT Device Management - 任务数据终端节点，请将具有适合您 AWS 区域 的终端节点的 --endpoint 参数添加到命令中。您还可以将 REST API 与此终端节点结合使用。

您可以将符合 FIPS 的终端节点 AWS 区域 与兼容 FIPS 的客户端配合使用，方法是使用 AWS IoT 设备 SDK 的连接功能提供终端节点，而不是账户的默认AWS IoT Device Management任务数据端点。连接功能特定于 AWS IoT 设备 SDK。有关连接函数的示例，请参阅适用于 Python 的 AWS IoT 设备 SDK 中的连接函数。

AWS IoT Device Management - Fleet Hub 终端节点

与 Flee t Hub AWS IoT 设备管理 CLI 命令一起使用的符合 FIPS 标准AWS IoT Device Management 的舰队中心端点列在按服务划分的 FIPS 端点中。在按服务划分 FIPS 终端节点中，查找 AWS IoT Device Management - Fleet Hub服务，然后查找您 AWS 区域的终端节点。

要在运行 aws iotfleethubCLI 命令时使用符合 AWS IoT Device Management FIPS 的 Fleet Hub 终端节点，请在命令中添加带有相应终端节点的--endpoint参数。 AWS 区域 您还可以将 REST API 与此终端节点结合使用。

AWS IoT Device Management - 安全隧道终端节点

AWS IoT 安全隧道 API 的符合 FIPS 标准的 AWS IoT Device Management- 安全隧道终端节点与相应 CLI 命令均在 FIPS Endpoints by Service（按服务划分 FIPS 终端节点）中列出。在按服务划分 FIPS 终端节点中，查找 AWS IoT Device Management - 安全隧道服务，然后查找您 AWS 区域的终端节点。

要在运行 aws iotsecuretunneling CLI 命令时使用符合 FIPS 标准的 AWS IoT Device Management - 安全隧道终端节点，请将具有适合您 AWS 区域 的终端节点的 --endpoint 参数添加到命令中。您还可以将 REST API 与此终端节点结合使用。

AWS IoT 教程

AWS IoT 教程分为两种学习路径来支持两个不同的目标。为您的目标选择最佳的学习路径。

- 您想要构建概念验证测试或演示 AWS IoT 解决方案想法

  要使用您设备上的 AWS IoT Device Client 展示常规 IoT任务和应用，请选择 the section called “用 AWS IoT Device Client 构建演示” 学习路径。AWS IoT Device Client 提供设备软件，您可以使用该软件应用自己的云资源来演示开发程度最低的端到端解决方案

  有关 AWS IoT Device Client 的信息，请参阅 AWS IoT Device Client 。
- 您想学习如何构建生产软件来部署解决方案

  要创建满足您特定要求的解决方案软件，请使用 AWS IoT设备开发工具包，遵循 the section called “使用 AWS IoT设备开发工具包构建解决方案” 学习路径。

  有关可用的 AWS IoT设备开发工具包信息，请参阅 ???。有关 AWS设备开发工具包的信息，请参阅在 AWS上构建的工具。

AWS IoT 教程学习路径选项

- 用 AWS IoT Device Client 构建演示
- 使用 AWS IoT设备开发工具包构建解决方案

用 AWS IoT Device Client 构建演示

此学习路径中的教程将引导您使用 AWS IoT Device Client 完成开发演示软件的步骤。AWS IoT Device Client 提供了在 IoT 设备上运行的软件，用于全方位测试和演示在 AWS IoT 上构建的 IoT 解决方案。

这些教程旨在促进探索和实验，这样在开发设备软件之前您可以有信心 AWS IoT 支持您的解决方案。

您将在本教程中学到的内容：

- 如何准备 Raspberry Pi 用作 AWS IoT 的 IoT 设备
- 如何使用设备上的 AWS IoT Device Client 演示 AWS IoT 功能

在这个学习路径中，您将在自己的 Raspberry Pi 上安装 AWS IoT Device Client 并创建云中的 AWS IoT 资源来演示 IoT 解决方案的概念。尽管此学习路径中的教程通过使用 Raspberry Pi 演示功能，也讲解了帮助您适应其他设备的目标和程序。

使用 AWS IoT Device Client 构建演示的先决条件

本节介绍在此学习路径中的教程开始之前需要了解的内容。

要完成此学习路径中的教程，您需要：

- 一个 AWS 账户

  您可以使用现有的 AWS 账户（如有），但是您可能需要添加额外的角色或权限才能使用这些教程的 AWS IoT 功能。

  如果您需要创建新 AWS 账户，请参阅 the section called “设置你的 AWS 账户”。

- Raspberry Pi 或兼容的 IoT 设备

  由于因数不同，这些教程使用 Raspberry Pi，这是一种常用的演示色斑，成本相对较低。已在 Raspberry Pi 3 Model B+、Raspberry Pi 4 Model B 以及运行 Ubuntu Server 20.04 LTS (HVM) 的 Amazon EC2 实例上测试了教程。要使用 AWS CLI 并运行命令，我们建议您使用最新版本的 Raspberry Pi OS [Raspberry Pi OS（64 位）或 OS Lite]。早期版本的操作系统可能有用，但我们还没有测试过。

  > **Note**
  >
  > 这些教程解释了每个步骤的目标，帮助您适应我们没有尝试过的 IoT 硬件；但是，没有具体描述如何适用于其他设备。

- 熟悉 IoT 设备的操作系统

  这些教程中的步骤假设您熟悉使用 Raspberry Pi 支持的命令行界面中的基本 Linux 命令和操作。如果您不熟悉这些操作，可能需要给自己更多的时间来完成教程。

  要完成这些教程，您应已经了解下列操作：

  - 安全地执行基本的设备操作，例如组装和连接组件、将设备连接到所需的电源以及安装和卸载存储卡。
  - 将系统软件和文件上传并下载到设备。如果您的设备不使用可移动存储设备，例如 microSD 卡，您需要知道如何连接到设备以及如何将系统软件和文件上传并下载到设备。

- 将您的设备连接到计划使用的网络。
- 用 SSH 终端或类似程序从另一台计算机连接到您的设备。
- 用命令行界面在设备上创建、复制、移动、重命名和设置文件和目录的权限。
- 在设备上安装新程序。
- 使用 FTP 或 SCP 等工具在设备之间传输文件。

- IoT 解决方案的开发和测试环境

  教程描述了所需的软件和硬件；然而，教程假设您能够执行可能没有明确描述的操作。此类硬件和操作的示例包括：

  - 下载并存储文件的本地主机

    对于 Raspberry Pi 而言，通常是可以读写 microSD 存储卡的个人计算机或笔记本电脑。本地主机必须：

    - 连接到互联网。
    - 已经安装并配置 AWS CLI。
    - 有支持 AWS控制台的网络浏览器。
  - 能够将本地主机连接到设备进行通信、输入命令以及传输文件

    在 Raspberry Pi 上，通常是使用本地主机上的 SSH 和 SCP 完成的。
  - 连接到 IoT 设备上的显示器和键盘

    这些可能会有所帮助，但并不是完成教程的必需内容。
  - 您本地主机和 IoT 设备能够连接到互联网

    这可能是连接到互联网的路由器或网关的有线网络连接或无线网络连接。本地主机还必须能够连接到 Raspberry Pi。这可能需要它们在同一个局域网上。这些教程无法向您展示如何针对特定设备或设备配置进行设置，但说明了如何测试连通性。
  - 访问局域网的路由器查看连接的设备

    要完成此学习路径中的教程，您需要能够找到 IoT 设备的 IP 地址。

    在局域网上，可以通过访问设备连接到的网络路由器的管理界面来完成此操作。如果您可以在路由器中为设备分配固定 IP 地址，可以在设备每次重新启动后简化重新连接。

    如果键盘和显示器连接到设备上，ifconfig 可以显示设备的 IP 地址。

    如果没有使用上述设备，需要在每次重启设备后找到一种方法来识别设备的 IP 地址。

获得所有资料之后，继续 the section called “AWS IoT Device Client 的设备准备”。

此学习路径中的教程

- 教程： AWS IoTDevice Client 的设备准备
- 教程：安装和配置 AWS IoTDevice Client
- 教程：演示 MQTT 消息与AWS IoT Device Client 通信
- 教程：使用 AWS IoTDevice Client 演示远程操作（任务）
- 教程：运行 AWS IoTDevice Client 教程后清理

教程： AWS IoTDevice Client 的设备准备

本教程将引导您完成 Raspberry Pi 的初始化，为此学习路径中的后续教程做好准备。

本教程的目标是安装当前版本的设备操作系统，确保您可以在开发环境中与设备进行通信。

要开始本教程：

- 请在 the section called “使用 AWS IoT Device Client 构建演示的先决条件”中列出可供使用的项目。

完成本教程需要大约 90 分钟。

完成本教程后：

- 您的 IoT 设备将安装最新的操作系统。
- 您的 IoT 设备将拥有后续教程所需的额外软件。
- 您了解到设备已连接到互联网。
- 您会在设备上安装所需的证书。

完成本教程后，下一个教程会帮助您的设备准备好使用 AWS IoTDevice Client。

本教程中的过程

- 步骤 1：安装并更新设备的操作系统
- 步骤 2：在设备上安装并验证所需的软件
- 步骤 3：测试您的设备并保存 Amazon CA 证书

步骤 1：安装并更新设备的操作系统

本节中的过程介绍了如何初始化 Raspberry Pi 用于系统驱动器的 microSD 卡。Raspberry Pi 的 microSD 卡包含其操作系统 (OS) 软件和应用程序文件存储空间。如果您没有使用 Raspberry Pi，请按照设备的说明安装和更新设备的操作系统软件。

完成本部分后，您应能够启动 IoT 设备并从本地主机上的终端程序进行连接。

所需的设备：

- 您的本地开发和测试环境
- 可以连接到互联网的 Raspberry Pi 或您的 IoT 设备
- 至少 8 GB 或足够空间的 microSD 存储卡，用于存储操作系统和所需的软件。

> **Note**
>
> 在选择 microSD 卡进行练习时，请选择一张存储量较小的卡。
> 小容量 SD 卡的备份和更新速度会更快。在 Raspberry Pi 上，这些教程中使用的 microSD 卡容量不需要超过 8 GB。如果您需要更大的存储量用于特定的应用程序，在这些教程中保存的较小的图片文件可以调整存储量较大卡上的文件系统大小，使用所选卡的所有支持空间。

可选设备：

- 连接到 Raspberry Pi 的 USB 键盘
- 用于将显示器连接到 Raspberry Pi 的 HDMI 显示器和电缆

本节中的步骤：

- 将设备的操作系统加载到 microSD 卡内
- 用新的操作系统启动 IoT 设备
- 将您的本地主机连接到设备上

将设备的操作系统加载到 microSD 卡内

此一步骤使用本地主机将设备的操作系统加载到 microSD 卡内。

> **Note**
>
> 如果您的设备没有用于操作系统的可移动存储设备，请使用该设备的过程安装操作系统，然后继续 the section called “用新的操作系统启动 IoT 设备”。

要在 Raspberry Pi 上安装操作系统

1. 在本地主机上，下载并解压缩要使用的 Raspberry Pi 操作系统映像。可以从 https://www.raspberrypi.com/software/operating-systems/ 下载最新版本

   选择 Raspberry Pi OS 版本

   本教程使用 Raspberry Pi OS Lite 版本，因为它是在此学习路径中支持这些教程的最小版本。这个版本的 Raspberry Pi 操作系统只有命令行界面，没有图形用户界面。带图形用户界面的最新版本 Raspberry Pi 操作系统也可以使用这些教程；但是，本学习路径中描述的步骤仅使用命令行界面对 Raspberry Pi 执行操作。
2. 将 microSD 卡插到您的计算机上。
3. 使用 SD 卡成像工具，将解压缩的操作系统映像文件写入 microSD 卡。
4. 将 Raspberry Pi OS 映像写入 microSD 卡之后：

   a. 在命令行窗口或文件资源管理器窗口中打开 microSD 卡上的 BOOT 分区。

   b. 在 microSD 卡的 BOOT 分区中的根目录下，创建一个名为 `ssh`的空文件，没有文件扩展名也没有内容。在首次开机时会告知 Raspberry Pi 启用 SSH 通信。
5. 弹出 microSD 卡并从本地主机上安全取出卡片。

您的 microSD 卡已经准备好 the section called “用新的操作系统启动 IoT 设备”。

用新的操作系统启动 IoT 设备

此过程安装 microSD 卡并使用下载的操作系统首次启动 Raspberry Pi。

用新的操作系统启动 IoT 设备

1. 从设备断开电源的情况下，插入上一步中的 microSD 卡，the section called “将设备的操作系统加载到 microSD 卡内”，进入 Raspberry Pi。
2. 将设备连接到有线网络
3. 这些教程将使用 SSH 终端从本地主机与 Raspberry Pi 进行交互。

如果还想直接与设备交互，您可以：

a. 将本地主机上的终端窗口 Connect 到 Raspberry Pi 之前，将 HDMI 显示器连接到Raspberry Pi 观看 Raspberry Pi 的控制台消息。

b. 如果想直接与 Raspberry Pi 交互，请将 USB 键盘连接。

4. 将电源连接到 Raspberry Pi 上，等待大约一分钟后才能初始化。

   如果您的显示器连接到 Raspberry Pi 上，那您可以在显示器上查看启动过程。

5. 找出设备的 IP 地址：

   - 如果您将 HDMI 显示器连接到 Raspberry Pi 上，IP 地址将出现在显示器上的消息中
   - 如果您可以访问 Raspberry Pi 连接的路由器，可以在路由器的管理界面中看到它的地址。

获得 Raspberry Pi IP 地址后，您可以进行 the section called “将您的本地主机连接到设备上”。

将您的本地主机连接到设备上

此过程使用本地主机上的终端程序连接到 Raspberry Pi 上并更改原定设置密码。

要将您的本地主机连接到设备上

1. 在本地主机上打开 SSH 终端程序：

   - Windows: PuTTY
   - Linux/macOS: `Terminal`

> **Note**
>
> 在 Windows 上没有自动安装 PuTTY。如果计算机上没有，可能需要下载并安装它。

2. 将终端程序连接到 Raspberry Pi 的 IP 地址，然后使用原定设置凭据登录。

```
username: pi
password: raspberry
```

3. 登录 Raspberry Pi 后，请更改 `pi`用户的密码。

```
passwd
```

按照提示更改密码。

```
Changing password for pi.
Current password: raspberry
New password: YourNewPassword
Retype new password: YourNewPassword
passwd: password updated successfully
```

在终端窗口中有 Raspberry Pi 的命令行提示符并更改密码后，可以继续 the section called “步骤 2：在设备上安装并验证所需的软件”。

步骤 2：在设备上安装并验证所需的软件

本节中的过程从上一节继续更新您的 Raspberry Pi 操作系统，并在下一节中使用的 Raspberry Pi 上安装该软件，用于构建和安装 AWS IoTDevice Client。

完成本节后，Raspberry Pi 将拥有最新的操作系统、本学习路径中教程所需的软件，并根据您的位置进行配置。

所需的设备：

- 上一节中您的本地开发和测试环境
- 上一节中您使用 的Raspberry Pi
- 上一节中的 microSD 存储卡

> **Note**
>
> Raspberry Pi Model 3+ 和 Raspberry Pi Model 4 可以执行此学习路径中描述的所有命令。如果您的 IoT 设备无法编译软件或运行 AWS Command Line Interface，您可能需要在本地主机上安装所需的编译器才能构建软件，然后将其传输到 IoT 设备。有关如何为设备安装和构建软件的更多信息，请参阅设备软件的文档。

本节中的步骤：

- 更新操作系统软件
- 安装所需的应用程序和库
- （可选）保存 microSD 卡映像

更新操作系统软件

这一步骤会更新操作系统软件。

要更新 Raspberry Pi 上的操作系统软件

在本地主机的终端窗口中执行这些步骤。

1. 输入以下命令在您的 Raspberry Pi 上更新系统软件。

```
sudo apt-get -y update
sudo apt-get -y upgrade
sudo apt-get -y autoremove
```

2. 更新 Raspberry Pi 的区域设置和时区设置（可选）。

   输入此命令可以更新设备的区域设置和时区设置。

```
sudo raspi-config
```

   a. 要设置设备的区域：

      i. 在 Raspberry Pi Software Configuration Tool（raspi-config）（Raspberry Pi 软件配置工具（raspi-config））屏幕上，选择选项 5。

         5 Localisation Options Configure language and regional settings

         使用 Tab键移动到 <Select>，然后按 space bar。

      ii. 在本地化选项菜单中，选择选项L1。

         L1 Locale Configure language and regional settings

         使用 Tab键移动到 <Select>，然后按 space bar。

      iii. 在区域设置选项列表中，使用箭头键滚动并选择要在 Raspberry Pi 上安装的语言环境space bar来标记您想要的选项。

         在美国，**en_US.UTF-8** 是一个很好的选择。

      iv. 为设备选择语言环境后，请使用 Tab键来选择<OK>，然后按 space bar显示配置区域设置的确认页面。

   b. 要设置设备的时区：

i. 在 aspi-config屏幕，选择选项5。

5 Localisation Options Configure language and regional settings

使用 Tab键移动到 <Select>，然后按 space bar。

ii. 在本地化选项菜单中，使用箭头键选择选项L2：

L2 time zone Configure time zone

使用 Tab键移动到 <Select>，然后按 space bar。

iii. 在配置 tzdata菜单中，从列表中选择您的地理区域。

使用 Tab键移动到 <OK>，然后按 space bar。

iv. 在城市列表中，使用箭头键选择时区内的城市。

使要设置时区，用 Tab键移动到 <OK>，然后按 space bar。

c. 更新完设置后，用 Tab键移动到<Finish>，然后按 space bar关闭 aspi-config应用程序。

3. 输入此命令可重启您的 Raspberry Pi。

```
sudo shutdown -r 0
```

4. 等您的 Raspberry Pi 重启。

5. 重新启动 Raspberry Pi 后，将本地主机上的终端窗口重新连接到 Raspberry Pi。

您的 Raspberry Pi 系统软件现已配置完毕，您已准备好继续 the section called “安装所需的应用程序和库”。

安装所需的应用程序和库

此过程将安装后续教程使用的应用程序软件和库。

如果您使用的是 Raspberry Pi，或可以在 IoT 设备上编译所需的软件，请在本地主机上的终端窗口中执行以下步骤。如果必须在本地主机上为 IoT 设备编译软件，请查看 IoT 设备的软件文档，了解有关如何在设备上执行这些步骤的信息。

在 Raspberry Pi 上安装应用程序软件和库

1. 输入此命令安装应用程序软件和库。

```
sudo apt-get -y install build-essential libssl-dev cmake unzip git python3-pip
```

2. 输入这些命令来确认安装了正确版本的软件。

```
gcc --version
cmake --version
openssl version
git --version
```

3. 确认已安装以下版本的应用程序软件：

   - gcc：9.3.0 或更高版本
   - cmake：3.10.x 或更高版本
   - OpenSSL：1.1.1 或更高版本
   - git：2.20.1 或更高版本

如果您的Raspberry Pi拥有所需应用程序软件的可接受版本，您准备好继续 the section called “（可选）保存 microSD 卡映像”。

（可选）保存 microSD 卡映像

在本学习路径中的整个教程中，您会遇到这些过程，将 Raspberry Pi 的 microSD 卡映像的副本保存到本地主机上的文件中。尽管鼓励这样做，但不是必需的任务。通过在建议的位置保存 microSD 卡映像，您可以跳过此学习路径中保存点之前的过程，如果发现需要重试某些内容，这可以节省时间。不定期保存 microSD 卡映像的后果是，如果 microSD 卡损坏或者不小心配置了应用程序或其设置错误，可能必须从头开始重新启动学习路径中的教程。

此时，Raspberry Pi 的 microSD 卡具有更新的操作系统和基本的应用程序软件加载。现在，您可以通过将 microSD 卡的内容保存到文件中来节省完成上述步骤所花的时间。采用拥有设备 microSD 卡映像的当前映像，您可以从这一点开始继续或重试教程或程序，无需从头开始安装和更新软件。

要将 microSD 卡映像保存到文件中

1. 输入此命令关闭 Raspberry Pi。

```
sudo shutdown -h 0
```

2. Raspberry Pi 完全关闭后，请移除电源。
3. 从 Raspberry Pi 中取出 microSD 卡。

4. 在本地主机上：
    a. 插入 microSD 卡。
    b. 使用 SD 卡成像工具，将 microSD 卡映像保存到文件中。
    c. 保存 microSD 卡映像后，从本地主机上弹出该卡。
5. 从 Raspberry Pi 断开电源后，将 microSD 卡插入 Raspberry Pi。
6. 给 Raspberry Pi 供电。
7. 等待大约一分钟后，在本地主机上重新连接已连接到 Raspberry Pi 的本地主机上的终端窗口，然后登录 Raspberry Pi。

步骤 3：测试您的设备并保存 Amazon CA 证书

本节中的过程从上一节继续，来安装AWS Command Line Interface以及用于验证您的连接的证书颁发机构证书 AWS IoT Core。

完成本节后，您会了解 Raspberry Pi 拥有必要的系统软件来安装 AWS IoTDevice Client 并与互联网有工作连接。

所需的设备：

- 上一节中您的本地开发和测试环境
- 上一节中您使用 的Raspberry Pi
- 上一节中的 microSD 存储卡

本节中的步骤：

- 安装 AWS Command Line Interface
- 配置 AWS 账户凭证
- 下载 Amazon Root CA 证书
- （可选）保存 microSD 卡映像

安装 AWS Command Line Interface

此过程在您的 Raspberry Pi 安装AWS CLI。

如果您使用的是 Raspberry Pi 或者可以在 IoT 设备上编译软件，请在本地主机上的终端窗口中执行以下步骤。如果必须在本地主机上为 IoT 设备编译软件，请查看 IoT 设备的软件文档，了解所需库的信息。

要在 Raspberry Pi 上安装 AWS CLI

1. 运行以下命令下载和并按照安装 AWS CLI。

```
export PATH=$PATH:~/.local/bin # configures the path to include the directory with
 the AWS CLI
git clone https://github.com/aws/aws-cli.git # download the AWS CLI code from
 GitHub
cd aws-cli && git checkout v2 # go to the directory with the repo and checkout
 version 2
pip3 install -r requirements.txt # install the prerequisite software
```

2. 运行此命令安装 AWS CLI。完成此命令可能最多需要 15 分钟。

```
pip3 install . # install the AWS CLI
```

3. 运行此命令确认已安装的 AWS CLI版本正确。

```
aws --version
```

   AWS CLI 版本应为 2.2或更高版本。

如果 AWS CLI显示其当前版本，您已准备就绪，可以继续 the section called “配置 AWS 账户凭证”。

配置 AWS 账户凭证

在此过程中，您将获得 AWS 账户凭据并添加它们在 Raspberry Pi 上使用。

要在您的设备上添加 AWS 账户凭据

1. 从获取访问密钥ID 和机密访问密钥，从您的 AWS 账户验证设备上的 AWS CLI。

   如果您是首次使用 AWSIAM， https://aws.amazon.com/premiumsupport/knowledge-center/create-access-key/ 描述在 AWS控制台中运行创建要在设备上使用的 AWSIAM 凭据的过程。

2. 在连接到 Raspberry Pi 的本地主机上的终端窗口中。并使用设备的访问密钥 ID 和密钥访问密钥凭据：

a. 使用此命令运行 AWS配置应用程序。

```
aws configure
```

b. 出现提示时输入您的凭据和配置信息：

```
AWS Access Key ID: your Access Key ID
AWS Secret Access Key: your Secret Access Key
Default region name: your AWS ## code
Default output format: json
```

3. 运行此命令测试设备对 AWS 账户和 AWS IoT Core端点的访问。

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

应返回您 AWS 账户特定的AWS IoT数据端点，例如此示例：

```
{
    "endpointAddress": "a3EXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
}
```

如果看见您 AWS 账户特定的AWS IoT数据端点，您的 Raspberry Pi 拥有连接和权限，可以继续 the section called “下载 Amazon Root CA 证书”。

> **Important**
>
> 现在，您的 AWS 账户凭据存储在 Raspberry Pi 的 microSD 卡上。尽管这样您会轻松使用 AWS在这些教程中创建的软件交互，但预设情况下，它们也将保存并复制到您在此步骤之后制作的任何 microSD 卡映像中。
> 为了保护 AWS 账户 凭证的安全，在保存任何其他 microSD 卡映像之前，请考虑通过再次运行 `aws configure` 来擦除凭证，再为 Access Key ID（访问密钥 ID）和 Secret Access Key（秘密访问密钥）输入随机字符，防止您的 AWS 账户 凭证遭盗用。
> 如果您发现无意中保存了 AWS 账户凭据，可以在 AWSIAM 控制台中停用它们。

下载 Amazon Root CA 证书

此过程下载并保存 Amazon Root 证书颁发机构（CA）的证书副本。下载此证书可保存以供后续教程中使用，还可测试设备与 AWS服务的连接。

要下载并保存 Amazon Root CA 证书

1. 运行以下命令为证书创建一个目录。

```
mkdir ~/certs
```

2. 运行此命令下载 Amazon Root CA 证书。

```
curl -o ~/certs/AmazonRootCA1.pem https://www.amazontrust.com/repository/
AmazonRootCA1.pem
```

3. 运行这些命令设置对证书目录及其文件的访问权限。

```
chmod 745 ~
chmod 700 ~/certs
chmod 644 ~/certs/AmazonRootCA1.pem
```

4. 运行此命令查看新目录中的 CA 证书文件。

```
ls -l ~/certs
```

   您会看到如下条目。日期和时间会有所不同；但是，文件大小和所有其他信息应与此处显示的相同。

```
-rw-r--r-- 1 pi pi 1188 Oct 28 13:02 AmazonRootCA1.pem
```

   如果文件大小不是 1188，检查 curl命令参数。您可能下载了错误的文件。

（可选）保存 microSD 卡映像

此时，Raspberry Pi 的 microSD 卡具有更新的操作系统和基本的应用程序软件加载。

要将 microSD 卡映像保存到文件中

1. 在本地主机上的终端窗口中，清除 AWS凭证。

a. 使用此命令运行 AWS配置应用程序。

```
aws configure
```

b. 出现提示时，替换凭证。您可以离开原定设置区域名称和原定设置输出格式，因为它们是通过按Enter（输入）。

```
AWS Access Key ID [****************YT2H]: XYXYXYXYX
AWS Secret Access Key [****************9plH]: XYXYXYXYX
Default region name [us-west-2]:
Default output format [json]:
```

2. 输入此命令关闭 Raspberry Pi。

```
sudo shutdown -h 0
```

3. Raspberry Pi 完全关闭后，卸下其电源接头。
4. 从设备中取出 microSD 卡。
5. 在本地主机上：

   a. 插入 microSD 卡。

   b. 使用 SD 卡成像工具，将 microSD 卡映像保存到文件中。

   c. 保存 microSD 卡映像后，从本地主机上弹出该卡。
6. 从 Raspberry Pi 断开电源后，将 microSD 卡插入 Raspberry Pi。
7. 对设备供电。
8. 大约一分钟后，在本地主机上重新启动终端窗口会话并登录到设备。

   请勿重新输入 AWS 账户凭据。

重新启动并登录 Raspberry Pi 之后，您准备好继续the section called “安装并配置 AWS IoTDevice Client”。

教程：安装和配置 AWS IoTDevice Client

本教程将引导您完成 AWS IoTDevice Client 的安装和配置，以及在此演示和其他演示中使用的 AWS IoT资源的创建。

要开始本教程：

- 准备好您的本地主机和上一教程中的 Raspberry Pi。

完成本教程需要大约 90 分钟。

完成本主题后：

- 您的 IoT 设备将准备好在其他 AWS IoTDevice Client 演示。
- 您将在 AWS IoT Core中配置 IoT 设备
- 您将已经下载并在设备上安装了 AWS IoTDevice Client。
- 您将保存设备 microSD 卡的映像，可在后续教程中使用。

所需的设备：

- 上一节中您的本地开发和测试环境
- 上一节中您使用 的Raspberry Pi
- 上一节中您使用的 Raspberry Pi microSD 记忆卡

本教程中的过程

- 步骤 1：下载并保存 AWS IoTDevice Client
- （可选）保存 microSD 卡映像
- 步骤 2：在 AWS IoT中预置 Raspberry Pi
- 步骤 3：配置 AWS IoTDevice Client 用于测试连接

步骤 1：下载并保存 AWS IoTDevice Client

本节中的步骤下载 AWS IoTDevice Client，编译，然后将其安装在 Raspberry Pi 上。测试安装后，您可以保存 Raspberry Pi的 microSD卡映像，以便以后需要再次尝试教程时使用。

本节中的步骤：

- 下载并构建 AWS IoTDevice Client
- 创建教程使用的目录

下载并构建 AWS IoTDevice Client

此过程安装 Raspberry Pi 中的AWS IoT Device Client。

在连接到 Raspberry Pi 的本地主机上的终端窗口中执行这些命令。

要在 Raspberry Pi 中安装 AWS IoTDevice Client

1. 输入以下命令可下载并构建 Raspberry Pi 中的 AWS IoTDevice Client。

```
cd ~
git clone https://github.com/awslabs/aws-iot-device-client aws-iot-device-client
mkdir ~/aws-iot-device-client/build && cd ~/aws-iot-device-client/build
cmake ../
```

2. 运行此命令来构建 AWS IoTDevice Client。完成此命令可能最多需要 15 分钟。

```
cmake --build . --target aws-iot-device-client
```

   警告消息显示为可以忽略 AWS IoTDevice Client 的编译文件。

   这些教程已经通过 gcc上构建的 AWS IoTDevice Client测试，gcc 上的 Raspberry Pi OS (bullseye) 2021年10月30日版本 (Raspbian 10.2.1-6+rpi1) 10.2.1 20210110，Raspberry Pi OS (buster) 2021年5月7日版本，(Raspbian 8.3.0-6+rpi1) 8.3.0 版本。

3. AWS IoT Device Client 完成构建后，通过运行此命令进行测试。

```
./aws-iot-device-client --help
```

如果您看到的命令行帮助 AWS IoTDevice Client，AWS IoT Device Client 已成功构建，可供您使用。

创建教程使用的目录

此过程将在 Raspberry Pi 上创建目录，用于存储教程在此学习路径中使用的文件。

要在此学习路径中创建教程使用的目录，请执行以下操作：

1. 运行这些命令创建所需的目录。

```
mkdir ~/dc-configs
mkdir ~/policies
```

```
mkdir ~/messages
mkdir ~/certs/testconn
mkdir ~/certs/pubsub
mkdir ~/certs/jobs
```

2. 运行这些命令设置新目录的权限。

```
chmod 745 ~
chmod 700 ~/certs/testconn
chmod 700 ~/certs/pubsub
chmod 700 ~/certs/jobs
```

创建这些目录并设置权限后，请继续 the section called “（可选）保存 microSD 卡映像”。

（可选）保存 microSD 卡映像

此时，Raspberry Pi 的 microSD 卡具有更新的操作系统和基本的应用程序以及 AWS IoTDevice Client。

如果您想再次尝试这些练习和教程，可以跳过前面的步骤，使用此步骤保存的 microSD 卡映像写入新的 microSD 卡，然后在 the section called “步骤 2：在 AWS IoT中预置 Raspberry Pi”中继续教程。

要将 microSD 卡图像保存到文件中，请执行以下操作：

在连接到 Raspberry Pi 的本地主机的终端窗口中：

1. 确认您尚未存储的 AWS 账户凭据。

   a. 使用此命令运行 AWS配置应用程序。

   ```
   aws configure
   ```

   b. 如果您的凭据已存储（如果出现在提示中），输入 **XYXYXYXYX**字符串在提示时，如下所示。保持原定设置区域名称和原定设置输出格式空白。

   ```
   AWS Access Key ID [****************YXYX]: XYXYXYXYX
   AWS Secret Access Key [****************YXYX]: XYXYXYXYX
   Default region name:
   Default output format:
   ```

2. 输入此命令可关闭 Raspberry Pi。

```
sudo shutdown -h 0
```

3. Raspberry Pi 完全关闭后，卸下其电源接头。
4. 从设备中取出 microSD 卡。
5. 在本地主机上：
   a. 插入 microSD 卡。
   b. 使用 SD 卡成像工具，将 microSD 卡映像保存到文件中。
   c. 保存 microSD 卡的映像后，从本地主机上弹出该卡。

您可以继续使用这张 microSD 卡在 the section called “步骤 2：在 AWS IoT中预置 Raspberry Pi”中操作。

步骤 2：在 AWS IoT中预置 Raspberry Pi

本节中的步骤从安装了 AWS CLI 和 AWS IoT Device Client 的已保存 microSD 映像开始，并创建 AWS IoT 资源和设备证书，以便在 AWS IoT中为您的 Raspberry Pi 提供资源和设备证书。

在您的 Raspberry Pi 中安装 microSD 卡

此过程安装 microSD 卡，加载必要的软件并配置到 Raspberry Pi 中，配置 AWS 账户这样，您就可以继续在这个学习路径中学习教程。

使用来自 the section called “（可选）保存 microSD 卡映像”的 microSD 卡，其中有这个学习路径中进行练习和教程的必要软件。

要在您的 Raspberry Pi 中安装 microSD 卡

1. 从 Raspberry Pi 断电后，将 microSD 卡插入 Raspberry Pi。
2. 给 Raspberry Pi 供电。
3. 大约一分钟后，在本地主机上重新启动终端窗口会话并登录 Raspberry Pi。
4. 在本地主机上、终端窗口中以及访问密钥 ID和您的 Raspberry Pi 上的密钥访问密钥凭证：
   a. 使用此命令运行 AWS配置应用程序。

```
aws configure
```

b. 出现提示时，输入您的 AWS 账户凭据和配置信息：

```
AWS Access Key ID [****************YXYX]: your Access Key ID
AWS Secret Access Key [****************YXYX]: your Secret Access Key
Default region name [us-west-2]: your AWS ## code
Default output format [json]: json
```

恢复 AWS 账户凭证之后，您已准备就绪继续 the section called “在 AWS IoT Core中配置您的设备”。

在 AWS IoT Core中配置您的设备

在本节的过程中创建在 AWS IoT中预置Raspberry Pi 的 AWS IoT资源。创建这些资源时，系统将要求您记录各种信息。在下一步骤中 AWS IoTDevice Client 用此信息执行 Device Client 配置。

对于使用 AWS IoT的 Raspberry Pi，必须进行预置。预置是创建和配置支持 Raspberry Pi 作为 IoT 设备所需 AWS IoT资源的过程。

启动并重新启动 Raspberry Pi 后，将本地主机上的终端窗口连接到 Raspberry Pi 上并完成这些步骤。

本节中的步骤：

- 创建并下载设备证书文件
- 创建 AWS IoT资源

创建并下载设备证书文件

此过程为此演示创建设备证书文件。

要为 Raspberry Pi 创建和下载设备证书文件

1. 在本地主机的终端窗口中，输入这些命令为您的设备创建设备证书文件。

```
mkdir ~/certs/testconn
aws iot create-keys-and-certificate \
--set-as-active \
--certificate-pem-outfile "~/certs/testconn/device.pem.crt" \
--public-key-outfile "~/certs/testconn/public.pem.key" \
--private-key-outfile "~/certs/testconn/private.pem.key"
```

此命令会返回类似以下内容的响应。记录 *certificateArn*值以供将来使用。

```
{
    "certificateArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
    "certificateId":
 "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
    "certificatePem": "-----BEGIN CERTIFICATE-----
\nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

2. 输入以下命令设置证书目录及其文件的权限。

```
chmod 745 ~
chmod 700 ~/certs/testconn
chmod 644 ~/certs/testconn/*
chmod 600 ~/certs/testconn/private.pem.key
```

3. 运行此命令可查看证书目录和文件的权限。

```
ls -l ~/certs/testconn
```

命令的输出应与您在此处看到的内容相同，但文件日期和时间会有所不同。

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

此时，您已经在 Raspberry Pi 上安装了设备证书文件，可以继续 the section called “创建 AWS IoT资源”。

创建 AWS IoT资源

此过程通过在 AWS IoT中创建您的设备访问 AWS IoT功能和服务所需的资源为您的设备提供资源。

在 AWS IoT中预置您的设备

1. 在本地主机的终端窗口中，输入以下命令获取您的 AWS 账户设备数据端点的地址。

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

上面步骤的命令会返回类似以下内容的响应。记录 *endpointAddress*值以供将来使用。

```
{
    "endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
}
```

2. 输入此命令为 Raspberry Pi 创建 AWS IoT事物资源。

```
aws iot create-thing --thing-name "DevCliTestThing"
```

如果您的AWS IoT事物资源创建后，命令会返回类似此响应。

```
{
    "thingName": "DevCliTestThing",
    "thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/DevCliTestThing",
    "thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
}
```

3. 在终端窗口中：

   a. 打开文本编辑器，例如 nano。

   b. 复制此 JSON 策略文档并将其粘贴到打开的文本编辑器中。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:Connect"
            ],
            "Resource": [
```

```
                "*"
            ]
        }
    ]
}
```

> **Note**
>
> 本策略文档慷慨授予每个资源连接、接收、发布和订阅的权限。通常，策略仅向特定资源授予权限执行特定操作。但是，对于初始设备连通性测试，这种过于笼统和宽容的策略用于尽量减少测试期间出现访问问题的可能性。在随后的教程中，将使用范围更窄的策略文档来展示策略设计中的更好做法。

c. 将文本编辑器中的文件保存为 **~/policies/dev_cli_test_thing_policy.json**。

4. 运行此命令使用之前步骤中的策略文档创建 AWS IoT策略。

```
aws iot create-policy \
--policy-name "DevCliTestThingPolicy" \
--policy-document "file://~/policies/dev_cli_test_thing_policy.json"
```

如果创建策略，该命令将返回类似此类的响应。

```
{
    "policyName": "DevCliTestThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/DevCliTestThingPolicy",
    "policyDocument": "{\n    \"Version\": \"2012-10-17\",\n    \"Statement\": [\n
        {\n            \"Effect\": \"Allow\",\n            \"Action\": [\n
           \"iot:Publish\",\n                \"iot:Subscribe\",\n
 \"iot:Receive\",\n                \"iot:Connect\"\n            ],\n
 \"Resource\": [\n                \"*\"\n            ]\n        }\n    ]\n}\n",
    "policyVersionId": "1"
}
```

5. 运行此命令将策略附加到设备证书。将 *certificateArn*替换为 您之前保存的 certificateArn值。

```
aws iot attach-policy \
--policy-name "DevCliTestThingPolicy" \
--target "certificateArn"
```

如果成功，该命令不返回任何内容。

6. 运行此命令将设备证书附加到 AWS IoT事物资源。将 *certificateArn*替换为 您之前保存的 certificateArn值。

```
aws iot attach-thing-principal \
--thing-name "DevCliTestThing" \
--principal "certificateArn"
```

如果成功，该命令不返回任何内容。

在 AWS IoT中成功配置设备后，您可以继续 the section called “步骤 3：配置 AWS IoTDevice Client 用于测试连接”。

步骤 3：配置 AWS IoTDevice Client 用于测试连接

本节中的过程配置 AWS IoTDevice Client 发布来自 Raspberry Pi 的 MQTT 消息。

本节中的步骤：

- 创建配置文件
- 打开 MQTT 测试客户端
- 运行 AWS IoTDevice Client

创建配置文件

此过程创建配置文件来测试 AWS IoTDevice Client。

要创建配置文件测试 AWS IoTDevice Client

- 在连接到 Raspberry Pi 的本地主机的终端窗口中：

  a. 输入这些命令为配置文件创建目录并设置对该目录的权限：

  ```
  mkdir ~/dc-configs
  chmod 745 ~/dc-configs
  ```

  b. 打开文本编辑器，例如 nano。

  c. 复制此 JSON 文档并将其粘贴到打开的文本编辑器中。

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/testconn/device.pem.crt",
  "key": "~/certs/testconn/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "DevCliTestThing",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": false,
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": ""
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
```

```
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

d. 将 ##替换为您在 the section called “在 AWS IoT Core中配置您的设备”中找到的 AWS 账户设备数据端点。

e. 将文本编辑器中的文件保存为 **~/dc-configs/dc-testconn-config.json**。

f. 运行这个命令在新文件上设置权限

```
chmod 644 ~/dc-configs/dc-testconn-config.json
```

保存文件后，您已准备就绪继续 the section called “打开 MQTT 测试客户端”。

打开 MQTT 测试客户端

此过程准备控制台 AWS IoT中的 MQTT测试客户端订阅 AWS IoTDevice Client 运行时发布的 MQTT 消息。

准备 MQTT 测试客户端订阅所有 MQTT 消息

1. 在本地主机上，在AWS IoT控制台，选择MQTT 测试客户端。
2. 在 Subscribe to a topic（订阅主题）选项卡的 Topic filter（主题筛选条件）中输入 #（井号），然后选择 Subscribe（订阅）来订阅每个 MQTT 主题。
3. 在订阅标签下，确认您看见 #（单独英镑符号）。

让窗口的 MQTT 测试客户端 保持打开，继续 the section called “运行 AWS IoTDevice Client”。

运行 AWS IoTDevice Client

此过程运行AWS IoT Device Client，这样发布一条 MQTT 消息接收和显示 MQTT 测试客户端。

要从 AWS IoTDevice Client 发送 MQTT 消息

1. 在执行此步骤时，确保连接到 Raspberry Pi 的终端窗口和带 MQTT 测试客户端可见。
2. 在终端窗口中，输入这些命令以使用在 the section called “创建配置文件”中创建的配置文件运行 AWS IoTDevice Client。

```
cd ~/aws-iot-device-client/build
./aws-iot-device-client --config-file ~/dc-configs/dc-testconn-config.json
```

在终端窗口中，AWS IoT Device Client 显示信息消息以及运行时发生的任何错误。

如果终端窗口中没有显示错误，请查看 MQTT 测试客户端。

3. 在 MQTT test client（MQTT 测试客户端）的 Subscriptions（订阅）窗口中，参阅发送到 `test/dc/pubtopic` 消息主题的 Hello World! 消息。
4. 如果AWS IoT Device Client 没有显示错误，您会看到 Hello World！发送到MQTT 测试客户端中的 `test/dc/pubtopic`消息，您已经证明了成功的连接。
5. 在终端窗口中，输入 **^C**(Ctrl-C) 可停止 AWS IoTDevice Client。

在您证明了AWS IoT Device Client 在Raspberry Pi 上正常运行之后，可以与 AWS IoT通信，您可以继续 the section called “演示 MQTT 消息与 AWS IoTDevice Client 通信”。

教程：演示 MQTT 消息与AWS IoT Device Client 通信

本教程演示了AWS IoT Device Client 如何订阅和发布 MQTT 消息，这些消息在 IoT 解决方案中常用。

要开始本教程：

- 将本地主机和 Raspberry Pi 配置为在上一节使用。

  如果您在安装 AWS IoTDevice Client 之后保存了 microSD 卡映像，您可以将 microSD 卡与 Raspberry Pi 一起使用该映像。
- 如果您之前运行过此演示，请查看 ???删除在之前运行中的所有 AWS IoT资源，避免重复资源错误。

完成本教程需要大约 45 分钟。

完成本主题后：

- 您将展示 IoT 设备可以通过不同的方式从 AWS IoT订阅 MQTT 消息然后将 MQTT 消息发布到 AWS IoT。

所需的设备：

- 上一节中您的本地开发和测试环境
- 上一节中您使用 的Raspberry Pi
- 上一节中您使用的 Raspberry Pi microSD 记忆卡

本教程中的过程

- 步骤 1：准备 Raspberry Pi 来演示 MQTT 消息通信
- 步骤 2：演示使用 AWS IoT Device Client 发布消息
- 步骤 3：演示使用 AWS IoTDevice Client 订阅消息

步骤 1：准备 Raspberry Pi 来演示 MQTT 消息通信

此过程在 AWS IoT中创建资源并在 Raspberry Pi中 演示 使用 AWS IoTDevice Client 的 MQTT 消息通信。

本节中的步骤：

- 创建证书文件演示 MQTT 通信
- 预置您的设备演示 MQTT 通信
- 配置 AWS IoTDevice Client 配置文件和 MQTT 测试客户端来演示 MQTT 通信

创建证书文件演示 MQTT 通信

此过程为此演示创建设备证书文件。

要为 Raspberry Pi 创建和下载设备证书文件

1. 在本地主机的终端窗口中，输入以下命令为您的设备创建设备证书文件。

```
mkdir ~/certs/pubsub
aws iot create-keys-and-certificate \
--set-as-active \
--certificate-pem-outfile "~/certs/pubsub/device.pem.crt" \
--public-key-outfile "~/certs/pubsub/public.pem.key" \
--private-key-outfile "~/certs/pubsub/private.pem.key"
```

此命令会返回类似以下内容的响应。保存 *certificateArn*值供稍后使用。

```
{
"certificateArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificateId":
 "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificatePem": "-----BEGIN CERTIFICATE-----
\nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
\n",
"keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
\n",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
}
}
```

2. 输入以下命令设置证书目录及其文件的权限。

```
chmod 700 ~/certs/pubsub
chmod 644 ~/certs/pubsub/*
chmod 600 ~/certs/pubsub/private.pem.key
```

3. 运行此命令可查看证书目录和文件的权限。

```
ls -l ~/certs/pubsub
```

命令的输出应与您在此处看到的内容相同，但文件日期和时间会有所不同。

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

4. 输入这些命令创建日志文件的目录。

```
mkdir ~/.aws-iot-device-client
mkdir ~/.aws-iot-device-client/log
chmod 745 ~/.aws-iot-device-client/log
echo " " > ~/.aws-iot-device-client/log/aws-iot-device-client.log
```

```
echo " " > ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
chmod 600 ~/.aws-iot-device-client/log/*
```

预置您的设备演示 MQTT 通信

本节将在 AWS IoT中创建预置 Raspberry Pi 的 AWS IoT资源。

在 AWS IoT中预置您的设备：

1. 在本地主机的终端窗口中，输入以下命令获取您的 AWS 账户设备数据端点的地址。

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

   自从您为上一教程运行此命令以来，端点值一直没有更改。再次在此处运行命令是为了方便地查找数据端点值并将其粘贴到本教程中使用的配置文件中。

   上面步骤的命令会返回类似以下内容的响应。记录 *endpointAddress*值以供将来使用。

   ```
   {
   "endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. 输入此命令为 Raspberry Pi 创建新的 AWS IoT事物资源。

   ```
   aws iot create-thing --thing-name "PubSubTestThing"
   ```

   因为一个 AWS IoT事物资源是您的设备在云中的虚拟表示，所以我们可以在 AWS IoT中创建多个事物资源用于不同的目的。它们都可以由同一物理 IoT 设备使用来表示设备的不同方面。

   这些教程一次只能使用一个事物资源来表示 Raspberry Pi。这样，在这些教程中，它们代表不同的演示，在为演示创建 AWS IoT资源后，可以返回并使用专门为每个演示创建的资源重复演示。

   如果您的 AWS IoT事物资源创建后，命令会返回类似此响应。

   ```
   {
   "thingName": "PubSubTestThing",
   "thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/PubSubTestThing",
   "thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
   }
   ```

3. 在终端窗口中：

   a. 打开文本编辑器，例如 nano。

   b. 复制此 JSON 文档并将其粘贴到打开的文本编辑器中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/subtopic"
      ]
```

```
    }
  ]
}
```

c. 在编辑器中，在策略文档的每个 Resource部分中，将 *us-west-2:57EXAMPLE833* 替换为您的 AWS 区域、冒号字符 (:) 和 12 位 AWS 账户数字。

d. 将文本编辑器中的文件保存为 **~/policies/pubsub_test_thing_policy.json**。

4. 运行此命令使用之前步骤中的策略文档创建 AWS IoT策略。

```
aws iot create-policy \
--policy-name "PubSubTestThingPolicy" \
--policy-document "file://~/policies/pubsub_test_thing_policy.json"
```

如果创建策略，该命令将返回类似此类的响应。

```
{
    "policyName": "PubSubTestThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/PubSubTestThingPolicy",
    "policyDocument": "{\n\"Version\": \"2012-10-17\",\n\"Statement\": [\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Connect\"\n],\n\"Resource\":
 [\n\"arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Publish\"\n],\n\"Resource\":
 [\n\"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Subscribe\"\n],\n\"Resource\": [\n
\"arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Receive\"\n],\n\"Resource\": [\n
\"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*\"\n]\n}\n]\n}\n",
    "policyVersionId": "1"
```

5. 运行此命令将策略附加到设备证书。将 *certificateArn*替换为您之前在本部分保存的 certificateArn值。

```
aws iot attach-policy \
--policy-name "PubSubTestThingPolicy" \
--target "certificateArn"
```

如果成功，该命令不返回任何内容。

6. 运行此命令将设备证书附加到 AWS IoT事物资源。将 *certificateArn*替换为您之前在本部分保存的 certificateArn值。

```
aws iot attach-thing-principal \
--thing-name "PubSubTestThing" \
--principal "certificateArn"
```

如果成功，该命令不返回任何内容。

在 AWS IoT中成功配置设备后，您准备好继续 the section called “配置 AWS IoTDevice Client 配置文件和 MQTT 测试客户端来演示 MQTT 通信”。

配置 AWS IoTDevice Client 配置文件和 MQTT 测试客户端来演示 MQTT 通信

此过程将创建一个配置文件来测试 AWS IoTDevice Client 。

要创建配置文件测试 AWS IoTDevice Client

1. 在连接到 Raspberry Pi 的本地主机的终端窗口中：

   a. 打开文本编辑器，例如 nano。

   b. 复制此 JSON 文档并将其粘贴到打开的文本编辑器中。

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/pubsub/device.pem.crt",
  "key": "~/certs/pubsub/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "PubSubTestThing",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": false,
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
```

```
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

c. 将 ##替换为您在 the section called “在 AWS IoT Core中配置您的设备”中找到的 AWS 账户设备数据端点。

d. 将文本编辑器中的文件保存为 **~/dc-configs/dc-pubsub-config.json**。

e. 运行这个命令在新文件上设置权限

```
chmod 644 ~/dc-configs/dc-pubsub-config.json
```

2. 要准备 MQTT 测试客户端订阅所有 MQTT 消息

a. 在本地主机上，在AWS IoT控制台，选择MQTT 测试客户端。

b. 在订阅主题选项卡中，在主题筛选条件中，输入**#**（一个英镑符号），然后选择订阅。

c. 在订阅标签下，确认您看见 **#**（单独英镑符号）。

继续此教程时，让窗口的 MQTT 测试客户端 保持打开。

保存文件并配置 MQTT 测试客户端后，您便可以继续 the section called “步骤 2：演示使用 AWS IoT Device Client 发布消息”。

步骤 2：演示使用 AWS IoT Device Client 发布消息

本节中的过程演示了 AWS IoTDevice Client 如何发送原定设置和自定义 MQTT 消息。

在上一步中为这些练习创建的策略中的以下策略声明授予 Raspberry Pi 执行以下操作的权限：

- **`iot:Connect`**

  为客户提供 `PubSubTestThing`命名，您的 Raspberry Pi 运行 AWS IoTDevice Client 连接。

  ```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Connect"
    ],
    "Resource": [
      "arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing"
    ]
  }
  ```

- **`iot:Publish`**

  授予 Raspberry Pi 用 `test/dc/pubtopic`MQTT 主题发布消息的全效。

  ```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Publish"
    ],
    "Resource": [
      "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic"
    ]
  }
  ```

`iot:Publish` 操作提供发布到资源数组中列出的 MQTT 主题权限。那些消息的内容不受策略声明的控制。

使用 AWS IoT Device Client发布原定设置消息

此过程运行AWS IoT Device Client，这样发布一条原定设置 MQTT 消息接收和显示 MQTT 测试客户端。

要从 AWS IoTDevice Client 发送原定设置 MQTT 消息

1. 执行此过程时，请确保连接到 Raspberry Pi 的本地主机上的终端窗口和使用 MQTT测试客户端的窗口都可见。
2. 在终端窗口中，输入这些命令以使用在 AWS IoT中创建的配置文件运行 the section called “创建配置文件”Device Client。

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-config.json
   ```

   在终端窗口中，AWS IoT Device Client 显示信息消息以及运行时发生的任何错误。

   如果终端窗口中没有显示错误，请查看 MQTT 测试客户端。
3. 在 MQTT test client（MQTT 测试客户端）的 Subscriptions（订阅）窗口中，参阅发送到 `test/dc/pubtopic` 消息主题的 Hello World! 消息。
4. 如果AWS IoT Device Client 没有显示错误，您会看到 Hello World！发送到MQTT 测试客户端中的 `test/dc/pubtopic`消息，您已经证明了成功的连接。
5. 在终端窗口中，输入 **^C**(Ctrl-C) 可停止 AWS IoTDevice Client。

在您证明了 AWS IoTDevice Client 发布了原定设置的 MQTT 消息后，您可以继续 the section called “使用 AWS IoTDevice Client 发布指标消息”。

使用 AWS IoTDevice Client 发布指标消息

本节中的过程创建一条自定义 MQTT 消息，然后运行 AWS IoTDevice Client，以便将自定义 MQTT 消息发布一次，以便 MQTT 测试客户端接收和显示。

为AWS IoT Device Client 创建自定义 MMQTT 消息

在连接到 Raspberry Pi 的本地主机上的终端窗口中执行以下步骤。

为 AWS IoTDevice Client 创建自定义消息

1. 在终端窗口中，打开文本编辑器，例如nano。
2. 在文本编辑器中，复制并粘贴以下 JSON 文档。这是AWS IoT Device Client 发布的 MQTT 消息负载。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

3. 将文本编辑器的内容另存为 **~/messages/sample-ws-message.json**。
4. 输入以下命令设置刚才创建的消息文件的权限。

```
chmod 600 ~/messages/*
```

为 AWS IoTDevice Client 创建用于发送自定义消息的配置文件

1. 在终端窗口中，在文本编辑器中，例如 nano，打开现有 AWS IoTDevice Client 配置文件：**~/dc-configs/dc-pubsub-config.json**。
2. 编辑 samples对象如下所示。此文件的其他部分不需要更改。

```
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "~/messages/sample-ws-message.json",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

3. 将文本编辑器的内容另存为 **~/dc-configs/dc-pubsub-custom-config.json**。
4. 运行这个命令在新文件上设置权限

```
chmod 644 ~/dc-configs/dc-pubsub-custom-config.json
```

通过使用AWS IoT Device Client 发布自定义 MQTT 消息

此更改仅影响 MQTT 消息负载的内容，因此当前策略将继续工作。但是，如果MQTT 主题（由 `~/dc-configs/dc-pubsub-custom-config.json`红的 `publish-topic`值定义) 已更改，还需要修改策略语句 `iot::Publish`，允许 Raspberry Pi 发布到新的 MQTT 主题。

要从 AWS IoTDevice Client 发送 MQTT 消息

1. 在执行此过程时，确保终端窗口和窗口都有 MQTT 测试客户端可见。此外，请确保 MQTT 测试客户端仍订阅为#主题筛选条件。如果没有订阅，请再次订阅#筛选条件主题。
2. 在终端窗口中，输入这些命令以使用在 AWS IoT中创建的配置文件运行 the section called “创建配置文件”Device Client。

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-custom-config.json
   ```

   在终端窗口中，AWS IoT Device Client 显示信息消息以及运行时发生的任何错误。

   如果终端窗口中没有显示错误，请查看 MQTT 测试客户端。
3. 在 MQTT 测试客户端中，在订阅窗口中，请参阅发送到 `test/dc/pubtopic`消息主题的自定义消息负载。
4. 如果 AWS IoTDevice Client 未显示任何错误，并且您在 MQTT测试客户端中看到发布到 `test/dc/pubtopic`消息的自定义消息负载，则表示您已成功发布自定义消息。
5. 在终端窗口中，输入 **^C** (Ctrl-C) 可停止 AWS IoT Device Client。

在您演示了 AWS IoT Device Client 发布了自定义消息负载之后，您可以继续 the section called “步骤 3：演示使用 AWS IoTDevice Client 订阅消息”。

步骤 3：演示使用 AWS IoTDevice Client 订阅消息

在本节中，您将演示两种类型的消息订阅：

- 单独订阅主题
- 通配符主题订阅

为这些练习创建的策略中的策略语句授予 Raspberry Pi 执行这些操作的权限：

- **`iot:Receive`**

为 AWS IoTDevice Client 提供接收与 Resource对象中命名主题匹配的 MQTT 主题的权限。

```
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/subtopic"
      ]
    }
```

- **iot:Subscribe**

  为 AWS IoTDevice Client 提供订阅与 Resource对象中命名相匹配的MQTT主题筛选条件的权限。

```
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic"
      ]
    }
```

订阅单个 MQTT 消息主题

此过程演示 AWS IoTDevice Client 如何订阅和记录 MQTT 消息。

在连接到 Raspberry Pi 的本地主机上的终端窗口中，列出 **~/dc-configs/dc-pubsub-custom-config.json**的内容或者在文本编辑器中打开该文件查看内容。定位 samples对象，应如下所示。

```
  "samples": {
    "pub-sub": {
      "enabled": true,
      "publish-topic": "test/dc/pubtopic",
      "publish-file": "~/messages/sample-ws-message.json",
      "subscribe-topic": "test/dc/subtopic",
      "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

注意 `subscribe-topic`值是AWS IoT Device Client 将在运行时订阅的 MQTT 主题。AWS IoT Device Client 将从此订阅中收到的消息有效负载写入`subscribe-file`值。

要从 AWS IoT Device Client 订阅 MQTT 消息主题

1. 在执行此过程时，确保终端窗口和窗口都有 MQTT 测试客户端可见。此外，请确保 MQTT 测试客户端仍订阅为#主题筛选条件。如果没有订阅，请再次订阅#筛选条件主题。
2. 在终端窗口中，输入这些命令以使用在 AWS IoT中创建的配置文件运行 the section called “创建配置文件”Device Client。

   ```
   cd ~/aws-iot-device-client/build
   ./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-custom-config.json
   ```

   在终端窗口中，AWS IoT Device Client 显示信息消息以及运行时发生的任何错误。

   如果终端窗口中未显示错误，请在 AWS IoT控制台继续。
3. 在 AWS IoT控制台中的 MQTT 测试客户端中，选择发布到主题选项卡。
4. 在 主题名称中，输入 **`test/dc/subtopic`**。
5. 在消息负载中，查看消息内容。
6. 选择 Publish（发布）发布 MQTT 消息。
7. 在终端窗口中，注意 AWS IoTDevice Client 的接收消息条目。

   ```
   2021-11-10T16:02:20.890Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
    subscribe topic, size: 45 bytes
   ```
8. 看见显示已收到消息接收消息的条目之后，输入**^C**(Ctrl-C) 可停止 AWS IoTDevice Client。
9. 输入此命令可查看消息日志文件的末尾并查看您从 MQTT 测试客户端发布的信息

   ```
   tail ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
   ```

通过查看日志文件中的消息，您已经演示了 AWS IoTDevice Client 收到了您从 MQTT 测试客户端发布的消息。

使用通配符订阅多个 MQTT 消息主题

这些过程演示了 AWS IoTDevice Client 如何使用通配符订阅和记录 MQTT 消息。要做到这一点，您将：

1. 更新 AWS IoTDevice Client 用于订阅 MQTT 主题的主题筛选条件。
2. 更新设备使用的策略允许新订阅。
3. 运行 AWS IoTDevice Client 并从 MQTT 测试控制台发布消息。

使用通配符 MQTT 主题筛选条件创建配置文件订阅多个 MQTT 消息主题

1. 在连接到 Raspberry Pi 的本地主机上的终端窗口中，打开 **~/dc-configs/dc-pubsub-custom-config.json**进行编辑并定位 samples对象。
2. 在文本编辑器中，定位 samples对象并更新 subscribe-topic值如下所示。

```
"samples": {
  "pub-sub": {
    "enabled": true,
    "publish-topic": "test/dc/pubtopic",
    "publish-file": "~/messages/sample-ws-message.json",
    "subscribe-topic": "test/dc/#",
    "subscribe-file": "~/.aws-iot-device-client/log/pubsub_rx_msgs.log"
```

新 subscribe-topic值是 MQTT主题筛选条件，其末尾带有 MQTT 通配符。这描述了以 test/dc/开始的所有 MQTT 主题的订阅。AWS IoT Device Client 将从该订阅接收的消息有效负载写入名为subscribe-file 的文件。

3. 将修改后的配置文件另存为**~/dc-configs/dc-pubsub-wild-config.json**，然后退出编辑器。

修改 Raspberry Pi 使用的策略允许订阅和接收多个 MQTT 消息主题

1. 在连接到 Raspberry Pi 的本地主机上的终端窗口中，在您最喜欢的文本编辑器中打开 **~/policies/pubsub_test_thing_policy.json**进行编辑，然后定位 iot::Subscribe和 iot::Receive文件中的策略语句。
2. 在iot::Subscribe策略语句中更新 Resource 对象中的字符串用 *替换 subtopic，如下所示。

```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Subscribe"
    ],
```

```
    "Resource": [
      "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/*"
    ]
  }
```

> **Note**
>
> MQTT 主题筛选条件通配符 是 +（加号）和 #（磅符号）。结尾带有 #的订阅请求订阅以字符前面的字符串开头的 #所有主题（例如，在本用例中为 test/dc/）。
> 但是，授权此订阅的策略语句中的资源值必须在主题筛选条件ARN中使用 *（星号）代替 #（磅号）。这是因为策略处理器使用的通配符不同于 MQTT 使用的通配符。
> 有关在策略中为主题和主题筛选条件使用通配符的详细信息，请参阅 在 MQTT 和 AWS IoT Core 策略中使用通配符。

3. 在 iot::Receive策略语句中更新 Resource 对象中的字符串用 subtopic替换 *，如下所示。

```
  {
    "Effect": "Allow",
    "Action": [
      "iot:Receive"
    ],
    "Resource": [
      "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*"
    ]
  }
```

4. 将更新后的策略文档另存为 **~/policies/pubsub_wild_test_thing_policy.json**，然后退出编辑器。

5. 输入此命令可更新本教程的策略使用新的资源定义。

```
aws iot create-policy-version \
--set-as-default \
--policy-name "PubSubTestThingPolicy" \
--policy-document "file://~/policies/pubsub_wild_test_thing_policy.json"
```

如果命令成功，将返回类似于此类的响应。请注意，policyVersionId 现在是 2，表示这是此策略的第二个版本。

如果成功更新策略，可以继续下一个过程。

```
{
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/PubSubTestThingPolicy",
    "policyDocument": "{\n  \"Version\": \"2012-10-17\",\n  \"Statement\": [\n
 {\n      \"Effect\": \"Allow\",\n      \"Action\": [\n        \"iot:Connect\"\n
    ],\n      \"Resource\": [\n        \"arn:aws:iot:us-west-2:57EXAMPLE833:client/
PubSubTestThing\"\n      ]\n    },\n    {\n      \"Effect\": \"Allow\",\n
   \"Action\": [\n        \"iot:Publish\"\n      ],\n      \"Resource\": [\n
        \"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic\"\n
  ]\n    },\n    {\n      \"Effect\": \"Allow\",\n      \"Action\": [\n
 \"iot:Subscribe\"\n      ],\n      \"Resource\": [\n        \"arn:aws:iot:us-
west-2:57EXAMPLE833:topicfilter/test/dc/*\"\n      ]\n    },\n    {\n      \"Effect
\": \"Allow\",\n      \"Action\": [\n        \"iot:Receive\"\n      ],\n
 \"Resource\": [\n        \"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*\"\n
     ]\n    }\n  ]\n}\n",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

如果您收到有太多策略版本无法保存新版本的错误，请输入此命令列出策略的当前版本。查看此命令返回的列表查找可以删除的策略版本。

```
aws iot list-policy-versions --policy-name "PubSubTestThingPolicy"
```

输入此命令删除不再需要的版本。请注意，您无法删除原定设置策略版本。原定设置策略版本为具有 `true`的 `isDefaultVersion`值。

```
aws iot delete-policy-version \
--policy-name "PubSubTestThingPolicy" \
--policy-version-id policyId
```

删除策略版本后，请重试此步骤。

使用更新的配置文件和策略，您可以使用AWS IoT Device Client 演示通配符订阅。

为了演示 AWS IoTDevice Client 如何订阅并接收多个 MQTT 消息主题

1. 在 MQTT 测试客户端，检查订阅。如果已在 #主题筛选条件中订阅了 MQTT 测试客户端，请继续执行下一步。如果没有订阅，进入 MQTT test client（MQTT 测试客户端）的 Subscribe

to a topic（订阅主题）选项卡，在 Topic filter（主题筛选条件）中输入 **#**（井号），然后选择 Subscribe（订阅）进行订阅。

2. 在连接到 Raspberry Pi 的本地主机上的终端窗口中，输入这些命令启动 AWS IoTDevice Client。

```
cd ~/aws-iot-device-client/build
./aws-iot-device-client --config-file ~/dc-configs/dc-pubsub-wild-config.json
```

3. 在观看在本地主机上终端窗口中的 AWS IoTDevice Client 输出，返回 MQTT 测试客户端。在发布主题选项卡中，主题名称，输入 **test/dc/subtopic**，然后选择发布。
4. 在终端窗口中，通过查找以下消息确认已收到消息，例如：

```
2021-11-10T16:34:20.101Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
 subscribe topic, size: 76 bytes
```

5. 在观看在本地主机的终端窗口中AWS IoT Device Client 输出，返回 MQTT 测试客户端。在发布主题选项卡中，主题名称，输入 **test/dc/subtopic2**，然后选择发布。
6. 在终端窗口中，通过查找以下消息确认已收到消息，例如：

```
2021-11-10T16:34:32.078Z [DEBUG] {samples/PubSubFeature.cpp}: Message received on
 subscribe topic, size: 77 bytes
```

7. 看到确认收到两条消息的消息后，请输入 **^C**(Ctrl-C) 可停止 AWS IoTDevice Client。
8. 输入此命令可查看消息日志文件的末尾并查看您从 MQTT 测试客户端发布的信息

```
tail -n 20 ~/.aws-iot-device-client/log/pubsub_rx_msgs.log
```

> **Note**
>
> 该日志文件仅包含消息有效负载。消息主题不会记录在收到的消息日志文件中。
> 您可能还会在接收到的日志中看到 AWS IoTDevice Client 发布的消息。这是因为通配符主题筛选条件包含该消息主题，有时候，在发布的消息发送给订阅者之前，消息代理可以处理订阅请求。

日志文件中的条目表明已收到消息。您可以使用其他主题名称重复此过程。主题名称以 `test/dc/`开头的所有消息应该接收并记录。主题名称以任何其他文本开头的消息将被忽略。

在演示AWS IoT Device Client 如何发布和订阅 MQTT 消息，继续 教程：使用 AWS IoTDevice Client 演示远程操作（任务）。

教程：使用 AWS IoTDevice Client 演示远程操作（任务）

在这些教程中，您将配置任务并将其部署到 Raspberry Pi，演示如何向 IoT 设备发送远程操作。

要开始本教程：

- 将本地主机和 Raspberry Pi 配置为在上一节使用。
- 如果您没有完成上一节中的教程，可以使用 Raspberry Pi 和 microSD 卡来尝试本教程，该卡具有在（可选）保存 microSD 卡映像中安装 AWS IoTDevice Client 后保存的图像。
- 如果您之前运行过此演示，请查看 ???删除在之前运行中的所有 AWS IoT资源，避免重复资源错误。

完成本教程需要大约 45 分钟。

完成本主题后：

- 您会演示 IoT 设备可以使用的不同方式使用 AWS IoT Core运行由 AWS IoT管理的远程操作。

所需的设备：

- 您在上一节中测试的本地开发和测试环境
- 您在上一节中测试的 Raspberry Pi
- 您在上一节中测试的来自Raspberry Pi 的 microSD 存储卡

本教程中的过程

- 步骤 1：准备 Raspberry Pi 运行任务
- 步骤 2：在 AWS IoT中创建并运行任务

步骤 1：准备 Raspberry Pi 运行任务

本节中的步骤介绍了如何使用AWS IoT Device Client 准备 Raspberry Pi 运行任务。

> **Note**
>
> 这些过程是特定于设备的。如果要同时对多个设备执行本节中的步骤，每个设备将需要自己的策略和唯一的、特定于设备的证书和设备名称。要为每台设备提供独特的资源，请对每台设备执行一次此过程，同时按照过程中所述更改特定于设备的元素。

本教程中的过程

- 提供您的 Raspberry Pi 来演示任务
- 配置 AWS IoTDevice Client 运行任务代理

提供您的 Raspberry Pi 来演示任务

本节中的过程通过创建 AWS IoT资源和设备证书在 AWS IoT中预调配 Raspberry Pi。

创建并下载设备证书文件演示 AWS IoT任务

此过程为此演示创建设备证书文件。

如果准备了多台设备，必须在每台设备上执行此过程。

要为 Raspberry Pi 创建和下载设备证书文件，请执行以下操作：

在连接到 Raspberry Pi 的本地主机上的终端窗口中，输入以下命令。

1. 输入以下命令为您的设备创建设备证书文件。

```
aws iot create-keys-and-certificate \
--set-as-active \
--certificate-pem-outfile "~/certs/jobs/device.pem.crt" \
--public-key-outfile "~/certs/jobs/public.pem.key" \
--private-key-outfile "~/certs/jobs/private.pem.key"
```

此命令会返回类似以下内容的响应。保存 *certificateArn*值供稍后使用。

```
{
"certificateArn": "arn:aws:iot:us-
west-2:57EXAMPLE833:cert/76e7e4edb3e52f52334be2f387a06145b2aa4c7fcd810f3aea2d92abc227d269",
"certificateId":
 "76e7e4edb3e52f5233EXAMPLE7a06145b2aa4c7fcd810f3aea2d92abc227d269",
```

```
"certificatePem": "-----BEGIN CERTIFICATE-----
\nMIIDWTCCAkGgAwIBAgI_SHORTENED_FOR_EXAMPLE_Lgn4jfgtS\n-----END CERTIFICATE-----
\n",
"keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BA_SHORTENED_FOR_EXAMPLE_ImwIDAQAB\n-----END PUBLIC KEY-----
\n",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----
\nMIIEowIBAAKCAQE_SHORTENED_FOR_EXAMPLE_T9RoDiukY\n-----END RSA PRIVATE KEY-----\n"
}
}
```

2. 输入以下命令设置证书目录及其文件的权限。

```
chmod 700 ~/certs/jobs
chmod 644 ~/certs/jobs/*
chmod 600 ~/certs/jobs/private.pem.key
```

3. 运行此命令可查看证书目录和文件的权限。

```
ls -l ~/certs/jobs
```

命令的输出应与您在此处看到的内容相同，但文件日期和时间会有所不同。

```
-rw-r--r-- 1 pi pi 1220 Oct 28 13:02 device.pem.crt
-rw------- 1 pi pi 1675 Oct 28 13:02 private.pem.key
-rw-r--r-- 1 pi pi  451 Oct 28 13:02 public.pem.key
```

将设备证书文件下载到 Raspberry Pi 之后，您准备好继续 the section called “提供您的 Raspberry Pi 来演示任务”。

创建要演示 AWS IoT任务的 AWS IoT资源

创建此设备的 AWS IoT资源。

如果准备了多台设备，必须在每台设备上执行此过程。

在 AWS IoT中预调配您的设备：

在连接到 Raspberry Pi 的本地主机的终端窗口中：

1. 输入以下命令获取您的设备数据端点的地址：AWS 账户。

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

自上次运行此命令以来，端点值未发生更改。再次在此处运行该命令可以轻松查找数据端点值并将其粘贴到本教程中使用的配置文件中。

此 describe-endpoint命令会返回类似以下内容的响应：记录 *endpointAddress*值以供将来使用。

```
{
"endpointAddress": "a3qjEXAMPLEffp-ats.iot.us-west-2.amazonaws.com"
}
```

2. 将 *uniqueThingName* 替换为您的设备提供唯一的名称。如果要使用多台设备执行本教程，请为每台设备指定自己的名称。例如，**TestDevice01**、**TestDevice02** 等等。

输入此命令为 Raspberry Pi 创建新的 AWS IoT事物资源。

```
aws iot create-thing --thing-name "uniqueThingName"
```

因为一个 AWS IoT事物资源是您的设备在云中的虚拟表示，所以我们可以在 AWS IoT中创建多个事物资源用于不同的目的。它们都可以由同一物理 IoT 设备使用来表示设备的不同方面。

> **Note**
>
> 当您想要保护多台设备的策略时，可以使用 ${iot:Thing.*ThingName*} 而不是静态事物名称 *uniqueThingName*。

这些教程每台设备一次只能使用一件事资源。这样，在这些教程中，它们代表不同的演示，以便在为演示创建 AWS IoT资源后，可以使用专门为每个演示创建的资源返回并重复演示。

如果您的 AWS IoT事物资源创建后，命令会返回类似此响应。记录 *thingArn*值以供稍后创建要在此设备上运行的任务时使用。

```
{
"thingName": "uniqueThingName",
"thingArn": "arn:aws:iot:us-west-2:57EXAMPLE833:thing/uniqueThingName",
```

```
"thingId": "8ea78707-32c3-4f8a-9232-14bEXAMPLEfd"
}
```

3. 在终端窗口中：

   a. 打开文本编辑器，例如 nano。

   b. 复制此 JSON 文档并将其粘贴到打开的文本编辑器中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:client/uniqueThingName"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/events/job/*",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/events/jobExecution/*",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/things/uniqueThingName/
jobs/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/events/jobExecution/*",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/$aws/
things/uniqueThingName/jobs/*"
```

```
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/subtopic",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/things/uniqueThingName/
jobs/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeJobExecution",
        "iot:GetPendingJobExecutions",
        "iot:StartNextPendingJobExecution",
        "iot:UpdateJobExecution"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/things/uniqueThingName"
      ]
    }
  ]
}
```

c. 在编辑器中，在策略文档的每个 Resource部分中，将 *us-west-2:57EXAMPLE833* 替换为您的 AWS 区域、冒号字符 (:) 和 12 位 AWS 账户数字。

d. 在编辑器中，在每个策略语句中，将 *uniqueThingName* 替换为您提供给此事物资源的事物名称。

e. 将文本编辑器中的文件保存为 **~/policies/jobs_test_thing_policy.json**。

如果要为多台设备运行此过程，请将文件保存为每台设备上的此文件名。

4. 将 *uniqueThingName* 替换为设备的名称，然后运行此命令创建为该设备定制的 AWS IoT策略。

```
aws iot create-policy \
--policy-name "JobTestPolicyFor uniqueThingName" \
--policy-document "file://~/policies/jobs_test_thing_policy.json"
```

如果创建策略，该命令将返回类似此类的响应。

```
{
    "policyName": "JobTestPolicyFor uniqueThingName",
    "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/
JobTestPolicyFor uniqueThingName",
    "policyDocument": "{\n\"Version\": \"2012-10-17\",\n\"Statement\": [\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Connect\"\n],\n\"Resource\":
 [\n\"arn:aws:iot:us-west-2:57EXAMPLE833:client/PubSubTestThing\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Publish\"\n],\n\"Resource\":
 [\n\"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Subscribe\"\n],\n\"Resource\": [\n
\"arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic\"\n]\n},\n{\n
\"Effect\": \"Allow\",\n\"Action\": [\n\"iot:Receive\"\n],\n\"Resource\": [\n
\"arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/*\"\n]\n}\n]\n}\n",
    "policyVersionId": "1"
```

5. 将 *uniqueThingName* 替换为设备 和 *certificateArn*的事物名称以及您在本节之前为此设备保存的 certificateArn值，然后运行此命令将策略附加到设备证书。

```
aws iot attach-policy \
--policy-name "JobTestPolicyFor uniqueThingName" \
--target "certificateArn"
```

如果成功，该命令不返回任何内容。

6. 将 *uniqueThingName* 替换为设备的事物名称，将 *certificateArn*替换为本节之前保存的 certificateArn值，然后运行此命令将设备证书附加到 AWS IoT事物资源。

```
aws iot attach-thing-principal \
--thing-name "uniqueThingName" \
--principal "certificateArn"
```

如果成功，该命令不返回任何内容。

成功预调配 Raspberry Pi 之后，您可以在测试中为另一个 Raspberry Pi 重复此部分，或者，如果所有设备均已预调配，请继续 the section called “配置 AWS IoTDevice Client 运行任务代理”。

配置 AWS IoTDevice Client 运行任务代理

此过程为 AWS IoTDevice Client 创建配置文件运行任务代理：

注意：如果准备多台设备，必须在每台设备上执行此过程。

要创建配置文件测试 AWS IoTDevice Client ：

1. 在连接到 Raspberry Pi 的本地主机的终端窗口中：

   a. 打开文本编辑器，例如 nano。

   b. 复制此 JSON 文档并将其粘贴到打开的文本编辑器中。

```
{
  "endpoint": "a3qEXAMPLEaffp-ats.iot.us-west-2.amazonaws.com",
  "cert": "~/certs/jobs/device.pem.crt",
  "key": "~/certs/jobs/private.pem.key",
  "root-ca": "~/certs/AmazonRootCA1.pem",
  "thing-name": "uniqueThingName",
  "logging": {
    "enable-sdk-logging": true,
    "level": "DEBUG",
    "type": "STDOUT",
    "file": ""
  },
  "jobs": {
    "enabled": true,
    "handler-directory": ""
  },
  "tunneling": {
    "enabled": false
  },
  "device-defender": {
    "enabled": false,
    "interval": 300
  },
  "fleet-provisioning": {
    "enabled": false,
    "template-name": "",
    "template-parameters": "",
    "csr-file": "",
    "device-key": ""
  },
  "samples": {
    "pub-sub": {
      "enabled": false,
      "publish-topic": "",
```

```
      "publish-file": "",
      "subscribe-topic": "",
      "subscribe-file": ""
    }
  },
  "config-shadow": {
    "enabled": false
  },
  "sample-shadow": {
    "enabled": false,
    "shadow-name": "",
    "shadow-input-file": "",
    "shadow-output-file": ""
  }
}
```

c. 将*##*替换为您在 AWS 账户中找到的 the section called “在 AWS IoT Core中配置您的设备”设备数据端点值。

d. 将 *uniqueThingName* 替换为此设备使用的事物名称。

e. 将文本编辑器中的文件保存为 **~/dc-configs/dc-jobs-config.json**。

2. 运行此命令设置新配置文件的文件权限。

```
chmod 644 ~/dc-configs/dc-jobs-config.json
```

此测试没有使用 MQTT 测试客户端。虽然设备将与 AWS IoT交换任务相关的 MQTT消息，但任务进度消息仅与运行任务的设备交换。由于任务进度消息仅与运行任务的设备交换，因此您无法从其他设备订阅（例如 AWS IoT控制台。

保存配置文件后，您已准备继续 the section called “步骤 2：在 AWS IoT中创建并运行任务”。

步骤 2：在 AWS IoT中创建并运行任务

本节中的过程创建一个任务文档和 AWS IoT任务资源。创建任务资源后，AWS IoT 将任务文档发送到指定的任务目标，任务代理将任务文档应用于设备或客户端。

本节中的过程

- 创建并存储任务的任务文档
- 为一台 IoT 设备在 AWS IoT中运行任务

创建并存储任务的任务文档

此过程将创建一个简单的任务文档包括在 AWS IoT任务资源内。这份任务文档显示“Hello world!” 在任务目标上。

要创建和存储任务文档：

1. 选择要将任务文档保存到的 Amazon S3 存储桶。如果您没有现有 Amazon S3 存储桶可用，需要创建一个。有关如何创建 Amazon S3 存储桶的信息，请参阅 Amazon S3 入门中的主题。
2. 为此任务创建并保存任务文档

   a. 在本地主机上，打开文本编辑器。

   b. 复制此文本并粘贴到编辑器中。

   ```
   {
       "operation": "echo",
       "args": ["Hello world!"]
   }
   ```

   c. 在本地主机上，将编辑器内容保存到名为 **hello-world-job.json**的文件中。

   d. 确认文件已正确保存。一些文本编辑器会自动追加 `.txt`用户保存文本文件时的文件名。如果您的编辑器附加 `.txt`在文件名中，在继续之前更正文件名。

3. 将 *path_to_file* 替换为到 **hello-world-job.json**的路径，如果不在当前目录中，将 *s3_bucket_name* 替换为所选存储桶的 Amazon S3 存储桶路径，然后运行此命令将您的任务文档放入 Amazon S3 存储桶中。

   ```
   aws s3api put-object \
   --key hello-world-job.json \
   --body path_to_file/hello-world-job.json --bucket s3_bucket_name
   ```

   通过替换以下 URL 中的 *s3_bucket_name* and *AWS_region*，可以确定标识存储在 Amazon S3 中的任务文档的任务文档 URL。记录生成的 URL 稍后用作*job_ocument_path*

   ```
   https://s3_bucket_name.s3.AWS_Region.amazonaws.com/hello-world-job.json
   ```

> **Note**
>
> AWS安全防止您无法在 AWS 账户外面打开此 URL，例如通过使用浏览器。预设情况下，具有文件访问权限的 AWS IoT任务引擎使用 URL。在生产环境中，您需要确保 AWS IoT 服务有权访问存储在 Amazon S3 中的任务文档。

保存任务文档的 URL 后，继续 the section called “为一台 IoT 设备在 AWS IoT中运行任务”。

为一台 IoT 设备在 AWS IoT中运行任务

本节中的过程启动 Raspberry Pi 上的AWS IoT Device Client，在设备上运行任务代理，等待任务运行。还在 AWS IoT中创建了一个任务资源，将将任务发送到 IoT 设备并在物联网设备上运行。

> **Note**
>
> 此过程仅在一台设备上运行任务。

要在 Raspberry Pi 上启动任务代理：

1. 在连接到 Raspberry Pi 的本地主机上的终端窗口中，运行此命令启动 AWS IoTDevice Client。

```
cd ~/aws-iot-device-client/build
./aws-iot-device-client --config-file ~/dc-configs/dc-jobs-config.json
```

2. 在终端窗口中，确认 AWS IoTDevice Client 并显示这些消息

```
2021-11-15T18:45:56.708Z [INFO]  {Main.cpp}: Jobs is enabled
                        .
                        .
                        .
2021-11-15T18:45:56.708Z [INFO]  {Main.cpp}: Client base has been notified that
 Jobs has started
2021-11-15T18:45:56.708Z [INFO]  {JobsFeature.cpp}: Running Jobs!
2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 startNextPendingJobExecution accepted and rejected
2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 nextJobChanged events
```

```
2021-11-15T18:45:56.708Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 updateJobExecutionStatusAccepted for jobId +
2021-11-15T18:45:56.738Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToUpdateJobExecutionAccepted with code {0}
2021-11-15T18:45:56.739Z [DEBUG] {JobsFeature.cpp}: Attempting to subscribe to
 updateJobExecutionStatusRejected for jobId +
2021-11-15T18:45:56.753Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToNextJobChanged with code {0}
2021-11-15T18:45:56.760Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToStartNextJobRejected with code {0}
2021-11-15T18:45:56.776Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToStartNextJobAccepted with code {0}
2021-11-15T18:45:56.776Z [DEBUG] {JobsFeature.cpp}: Ack received for
 SubscribeToUpdateJobExecutionRejected with code {0}
2021-11-15T18:45:56.777Z [DEBUG] {JobsFeature.cpp}: Publishing
 startNextPendingJobExecutionRequest
2021-11-15T18:45:56.785Z [DEBUG] {JobsFeature.cpp}: Ack received for
 StartNextPendingJobPub with code {0}
2021-11-15T18:45:56.785Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

3. 在终端窗口中，看到此消息后，继续下一步并创建任务资源。请注意，它可能不是列表中的最后一个条目。

```
2021-11-15T18:45:56.785Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

要创建 AWS IoT任务资源

1. 在本地主机上：

   a. 将 *job_ocument_url* 替换为 the section called “创建并存储任务的任务文档”的任务文档 URL。

   b. 将 *thing_arn* 替换为您为设备创建的事物资源的 ARN，然后运行此命令。

```
aws iot create-job \
--job-id hello-world-job-1 \
--document-source "job_document_url" \
--targets "thing_arn" \
--target-selection SNAPSHOT
```

如果成功，该命令将返回类似此结果。

```
{
  "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-job-1",
  "jobId": "hello-world-job-1"
}
```

2. 在终端窗口中，您应看到 AWS IoTDevice Client 的输出 。

```
2021-11-15T18:02:26.688Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Job ids differ
2021-11-15T18:10:24.890Z [INFO]  {JobsFeature.cpp}: Executing job: hello-world-
job-1
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Attempting to update job
 execution status!
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Not including stdout with the
 status details
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Not including stderr with the
 status details
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Assuming executable is in PATH
2021-11-15T18:10:24.890Z [INFO]  {JobsFeature.cpp}: About to execute: echo Hello
 world!
2021-11-15T18:10:24.890Z [DEBUG] {Retry.cpp}: Retryable function starting, it will
 retry until success
2021-11-15T18:10:24.890Z [DEBUG] {JobsFeature.cpp}: Created EphermalPromise for
 ClientToken 3TEWba9Xj6 in the updateJobExecution promises map
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Child process now running
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Child process about to call
 execvp
2021-11-15T18:10:24.890Z [DEBUG] {JobEngine.cpp}: Parent process now running, child
 PID is 16737
2021-11-15T18:10:24.891Z [DEBUG] {16737}: Hello world!
2021-11-15T18:10:24.891Z [DEBUG] {JobEngine.cpp}: JobEngine finished waiting for
 child process, returning 0
2021-11-15T18:10:24.891Z [INFO]  {JobsFeature.cpp}: Job exited with status: 0
2021-11-15T18:10:24.891Z [INFO]  {JobsFeature.cpp}: Job executed successfully!
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Attempting to update job
 execution status!
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Not including stdout with the
 status details
```

```
2021-11-15T18:10:24.891Z [DEBUG] {JobsFeature.cpp}: Not including stderr with the
 status details
2021-11-15T18:10:24.892Z [DEBUG] {Retry.cpp}: Retryable function starting, it will
 retry until success
2021-11-15T18:10:24.892Z [DEBUG] {JobsFeature.cpp}: Created EphermalPromise for
 ClientToken GmQ0HTzWGg in the updateJobExecution promises map
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Ack received for
 PublishUpdateJobExecutionStatus with code {0}
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Removing ClientToken 3TEWba9Xj6
 from the updateJobExecution promises map
2021-11-15T18:10:24.905Z [DEBUG] {JobsFeature.cpp}: Success response after
 UpdateJobExecution for job hello-world-job-1
2021-11-15T18:10:24.917Z [DEBUG] {JobsFeature.cpp}: Ack received for
 PublishUpdateJobExecutionStatus with code {0}
2021-11-15T18:10:24.918Z [DEBUG] {JobsFeature.cpp}: Removing ClientToken GmQ0HTzWGg
 from the updateJobExecution promises map
2021-11-15T18:10:24.918Z [DEBUG] {JobsFeature.cpp}: Success response after
 UpdateJobExecution for job hello-world-job-1
2021-11-15T18:10:25.861Z [INFO]  {JobsFeature.cpp}: No pending jobs are scheduled,
 waiting for the next incoming job
```

3. 尽管 AWS IoTDevice Client 正在运行并等待任务，您可以通过更改 `job-id`值并从步骤 1 开始重新运行 create-job。

完成运行任务后，在终端窗口中输入 ^C（control-C）来停止 AWS IoTDevice Client。

教程：运行 AWS IoTDevice Client 教程后清理

本教程中的过程将指导您在完成本学习路径中的教程时删除创建的文件和资源。

本教程中的过程

- 步骤 1：使用 Device Client AWS IoT构建演示后清理设备
- 步骤 2：使用 AWS IoT Device Client 构建演示后清理您的 AWS 账户

步骤 1：使用 Device Client AWS IoT构建演示后清理设备

本教程介绍了在此学习路径中构建演示后如何清理 microSD 卡的两个选项。选择提供所需安全级别的选项。

请注意，清理设备的 microSD 卡不会移除任何您创建的 AWS IoT资源。要在清理设备的 microSD 卡后清理 AWS IoT资源，应查看 the section called “使用 AWS IoT Device Client 构建演示后进行清理”上的教程。

选项 1：通过重写 microSD 卡进行清理

在完成本学习路径中的教程之后，清理 microSD 卡最简单、最彻底的方法是用您在首次准备设备时创建的保存图像文件覆盖 microSD 卡。

此过程使用本地主机将保存的 microSD 卡映像写入 microSD 卡。

> **Note**
>
> 如果您的设备没有将可移动存储介质用于操作系统，请参阅该设备的步骤。

要向 microSD 卡写入一个新映像

1. 在本地主机上，找到要写入 microSD 卡的已保存 microSD 卡映像。
2. 将 microSD 卡插到您的计算机上。
3. 使用 SD 卡成像工具，将选定的图像文件写入 microSD 卡。
4. 将 Raspberry Pi 操作系统映像写入 microSD 卡后，弹出 microSD 卡并将其从本地主机安全地移除。

您的 microSD 卡已准备就绪，可供使用。

选项 2：通过删除用户目录进行清理

要在完成教程后清理 microSD 卡而不重写 microSD 卡映像，可以单独删除用户目录。这并不像从保存的图像中重写 microSD 卡那么彻底，因为它不会删除可能已安装的任何系统文件。

如果删除用户目录足以满足您的需要，可以按照此步骤操作。

从设备中删除此学习路径的用户目录

1. 运行这些命令可以在连接到设备的终端窗口中删除在此学习路径中创建的用户目录、子目录及其所有文件。

> **Note**
>
> 删除这些目录和文件后，如果不再完成教程，您将无法运行演示。

```
rm -Rf ~/dc-configs
rm -Rf ~/policies
rm -Rf ~/messages
rm -Rf ~/certs
rm -Rf ~/.aws-iot-device-client
```

2. 在连接到设备的终端窗口中，运行这些命令删除应用程序源目录和文件。

> **Note**
>
> 这些命令不会卸载任何程序。他们只删除用于构建和安装它们的源文件。删除这些文件后，AWS CLI 和 AWS IoTDevice Client 可能不适用。

```
rm -Rf ~/aws-cli
rm -Rf ~/aws
rm -Rf ~/aws-iot-device-client
```

步骤 2：使用 AWS IoT Device Client 构建演示后清理您的 AWS 账户

这些过程可以帮助您识别和删除您在完成此学习路径中的教程时创建的 AWS资源。

清理 AWS IoT资源

此过程可以帮助您识别并删除您在完成此学习路径中的教程时创建的 AWS IoT资源。

在此学习路径中创建的 AWS IoT资源

教程	事物资源	策略资源
the section called “安装并配置 AWS IoTDevice Client”	DevclitestTthing	PubSubTestThingPolicy

教程	事物资源	策略资源
the section called “演示 MQTT 消息与 AWS IoTDevice Client 通信”	PubSubTestThing	PubSubTestThingPolicy
the section called “使用 AWS IoTDevice Client 演示远程操作（任务）”	定义的用户（可能有不止一个）	定义的用户（可能有不止一个）

要删除 AWS IoT资源，对于您创建的每个事物资源，请按照此过程操作

1. 将 *thing_name*替换为要删除的对象资源的名称，然后运行此命令列出从本地主机附加到对象资源的证书。

   ```
   aws iot list-thing-principals --thing-name thing_name
   ```

   此命令返回类似这样的响应，其中列出了附加到 *thing_name*的证书。在大多数情况下，列表中只有一个证书。

   ```
   {
       "principals": [
           "arn:aws:iot:us-
   west-2:57EXAMPLE833:cert/23853eea3cf0edc7f8a69c74abeafa27b2b52823cab5b3e156295e94b26ae8ac"
       ]
   }
   ```

2. 对于上一个命令列出的每个证书：

   a. 将 *certificate_ID*替换为上一个命令中的证书 ID。证书ID是上一个命令返回的 ARN 中遵循 `cert/`的字母数字字符。然后运行此命令以停用证书。

      ```
      aws iot update-certificate --new-status INACTIVE --certificate-
      id certificate_ID
      ```

      如果成功，该命令不返回任何内容。

   b. 将 *certificate_ARN*替换为之前返回的证书列表中的证书 ARN，然后运行此命令列出附加到此证书的策略。

```
aws iot list-attached-policies --target certificate_ARN
```

此命令返回类似这样的响应，其中列出了附加到证书的策略。在大多数情况下，列表中只有一个策略。

```
{
    "policies": [
        {
            "policyName": "DevCliTestThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:57EXAMPLE833:policy/
DevCliTestThingPolicy"
        }
    ]
}
```

c. 对于附加到该证书的每个策略。

i. 将 *policy_name* 替换为上一个命令中的 policyName值，将 *certificate_ARN* 替换为证书的 ARN，然后运行此命令将策略与证书分离。

```
aws iot detach-policy --policy-name policy_name --target certificate_ARN
```

如果成功，该命令不返回任何内容。

ii. 将 *policy_name* 替换为 policyName值，然后运行此命令查看策略是否附加到其他证书。

```
aws iot list-targets-for-policy --policy-name policy_name
```

如果该命令返回这样的空列表，该策略不会附加到任何证书，您将继续列出策略版本。如果还有附加到策略的证书，请继续 detach-thing-principal步骤。

```
{
    "targets": []
}
```

iii. 将 *policy_name* 替换为 policyName值，然后运行此命令检查策略版本。要删除策略，它必须只有一个版本。

```
aws iot list-policy-versions --policy-name policy_name
```

如果策略只有一个版本（如此示例），则可以跳至步骤 delete-policy立即删除策略。

```
{
    "policyVersions": [
        {
            "versionId": "1",
            "isDefaultVersion": true,
            "createDate": "2021-11-18T01:02:46.778000+00:00"
        }
    ]
}
```

如果策略有多个版本，如本例所示，则必须先删除 `false`值为 `isDefaultVersion`的策略版本，然后才能删除策略。

```
{
    "policyVersions": [
        {
            "versionId": "2",
            "isDefaultVersion": true,
            "createDate": "2021-11-18T01:52:04.423000+00:00"
        },
        {
            "versionId": "1",
            "isDefaultVersion": false,
            "createDate": "2021-11-18T01:30:18.083000+00:00"
        }
    ]
}
```

如果您需要删除策略版本，将 *policy_name*替换为 `policyName`值，将 *version_ID*替换为来自上一个命令的 `versionId`值，然后运行此命令删除策略版本。

```
aws iot delete-policy-version --policy-name policy_name --policy-version-
id version_ID
```

如果成功，该命令不返回任何内容。

删除策略版本后，请重复此步骤，直到该策略只有一个策略版本。

iv. 将 *policy_name*替换为 policyName值，然后运行此命令删除策略。

```
aws iot delete-policy --policy-name policy_name
```

d. 将 *thing_name*替换为事物名称，将 *certificate_ARN*替换为证书的 ARN，然后运行此命令将证书与事物资源分离。

```
aws iot detach-thing-principal --thing-name thing_name --
principal certificate_ARN
```

如果成功，该命令不返回任何内容。

e. 将 *certificate_ID*替换为上一个命令中的证书 ID。证书 ID 是上一个命令返回的 ARN 中遵循 cert/的字母数字字符。运行这个命令删除证书资源。

```
aws iot delete-certificate --certificate-id certificate_ID
```

如果成功，该命令不返回任何内容。

3. 将 *thing_name*替换为事物名称，然后运行此命令删除该事物。

```
aws iot delete-thing --thing-name thing_name
```

如果成功，该命令不返回任何内容。

清理 AWS资源

此过程可以帮助您识别和删除您在完成此学习路径中的教程时创建的其他 AWS资源。

在此学习路径中创建的其他 AWS资源

教程	资源类型	资源名称或 ID
the section called “使用 AWS IoTDevice Client 演示远程操作（任务）”	Amazon S3 对象	hello-world-job.json

教程	资源类型	资源名称或 ID
the section called “使用 AWS IoTDevice Client 演示远程操作（任务）”	AWS IoT 任务资源	定义的用户

要删除在此学习路径中创建的 AWS资源

1. 要删除在此学习路径中创建的职位

   a. 运行此命令列出您的任务AWS 账户。

   ```
   aws iot list-jobs
   ```

   此命令将返回您的 AWS 账户和 AWS 区域的 AWS IoT任务列表。

   ```
   {
       "jobs": [
           {
               "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-
   job-2",
               "jobId": "hello-world-job-2",
               "targetSelection": "SNAPSHOT",
               "status": "COMPLETED",
               "createdAt": "2021-11-16T23:40:36.825000+00:00",
               "lastUpdatedAt": "2021-11-16T23:40:41.375000+00:00",
               "completedAt": "2021-11-16T23:40:41.375000+00:00"
           },
           {
               "jobArn": "arn:aws:iot:us-west-2:57EXAMPLE833:job/hello-world-
   job-1",
               "jobId": "hello-world-job-1",
               "targetSelection": "SNAPSHOT",
               "status": "COMPLETED",
               "createdAt": "2021-11-16T23:35:26.381000+00:00",
               "lastUpdatedAt": "2021-11-16T23:35:29.239000+00:00",
               "completedAt": "2021-11-16T23:35:29.239000+00:00"
           }
       ]
   }
   ```

b. 对于从列表中识别为您在此学习路径中创建任务的每份任务，将 *jobId*替换为要删除的任务 jobId值，然后运行此命令删除 AWS IoT任务。

```
aws iot delete-job --job-id jobId
```

如果命令成功，该命令不返回任何内容。

2. 要删除您在此学习路径中存储在 Amazon S3 存储桶中的任务文档。

a. 将 *bucket*替换为您使用的存储桶名称，然后运行此命令列出您使用的 Amazon S3 存储桶中的对象。

```
aws s3api list-objects --bucket bucket
```

该命令返回存储桶中 Amazon S3 对象的列表，该列表如下所示。

```
{
    "Contents": [
        {
            "Key": "hello-world-job.json",
            "LastModified": "2021-11-18T03:02:12+00:00",
            "ETag": "\"868c8bc3f56b5787964764d4b18ed5ef\"",
            "Size": 54,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
 "e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        },
        {
            "Key": "iot_job_firmware_update.json",
            "LastModified": "2021-04-13T21:57:07+00:00",
            "ETag": "\"7c68c591949391791ecf625253658c61\"",
            "Size": 66,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
 "e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        },
```

```
        {
            "Key": "order66.json",
            "LastModified": "2021-04-13T21:57:07+00:00",
            "ETag": "\"bca60d5380b88e1a70cc27d321caba72\"",
            "Size": 29,
            "StorageClass": "STANDARD",
            "Owner": {
                "DisplayName": "EXAMPLE",
                "ID":
 "e9e3d6ec1EXAMPLEf5bfb5e6bd0a2b6ed03884d1ed392a82ad011c144736a4ee"
            }
        }
    ]
}
```

b. 对于从列表中识别为在此学习路径中创建的对象的每个对象，将 *bucket* 替换为要删除的对象的存储桶名称，将 *key* 替换为密钥值，然后运行此命令删除 Amazon S3 对象。

```
aws s3api delete-object --bucket bucket --key key
```

如果命令成功，该命令不返回任何内容。

删除在完成此学习路径时创建的所有 AWS资源和对象，您可以重新开始并重复教程。

使用 AWS IoT设备开发工具包构建解决方案

本节中的教程帮助您完成开发可使用 AWS IoT 部署到生产环境的 IoT 解决方案的步骤。

与上一节 the section called “用 AWS IoT Device Client 构建演示”中的教程相比，这些教程可能需要更多的时间来完成，因为它们使用 AWS IoT设备软件开发包，并更详细地解释了应用的概念，帮助您创建安全可靠的解决方案。

开始使用 AWS IoT设备开发工具包构建解决方案

这些教程将引导您完成不同的 AWS IoT 场景。在适当的情况下，这些教程将使用 AWS IoT 设备 SDK。

主题

- 教程：使用 AWS IoT设备开发工具包将设备连接到 AWS IoT Core
- 创建将设备数据路由到其他服务的 AWS IoT 规则

- 在设备处于离线状态时使用设备影子保持设备状态
- 教程：为 AWS IoT Core创建自定义授权方
- 教程：使用 AWS IoT 和 Raspberry Pi 监测土壤湿度

教程：使用 AWS IoT设备开发工具包将设备连接到 AWS IoT Core

本教程演示如何将设备连接到 AWS IoT Core，以便它可以与 AWS IoT收发数据。完成本教程后，您的设备将配置为连接到 AWS IoT Core，您将了解设备如何与 AWS IoT通信。

在本教程中，您将：

1. the section called “为 AWS IoT准备好您的设备。”
2. the section called “查看 MQTT 协议”
3. the section called “查看 pubsub.py Device SDK 示例应用程序”
4. the section called “连接设备并与 AWS IoT Core通信”
5. the section called “查看结果”

完成本教程需要大约 1 小时。

在开始本教程之前，请确保您具有：

- 已完成 入门 AWS IoT Core

  在本教程中，您必须在某部分 the section called “配置您的设备”，请为您的设备选择 the section called “连接 Raspberry Pi 或其他设备”选项，然后使用 Python 语言选项来配置设备。

  在该教程中使用的终端窗口保持开启，因为您还将在本教程中使用它。

- 可以运行 AWS IoTDevice SDK v2 for Python。

  本教程介绍了如何使用 Python 代码示例将设备连接到 AWS IoT Core，这些示例需要一个功能相对强大的设备。

  如果您使用的是资源受限的设备，则这些代码示例可能无法适用于这些设备。在这种情况下，您可能会通过 the section called “使用 AWS IoT Device SDK for Embedded C”教程成功完成更多操作。

为 AWS IoT准备好您的设备。

在 入门 AWS IoT Core，您已准备好您的设备以及 AWS账户，所以它们能够进行通信。本部分回顾了准备工作的各个方面，这些方面涉及与 AWS IoT Core进行的任何设备通信。

对于要连接到 AWS IoT Core的设备：

1. 您必须具有 AWS 账户。

   如果您没有账户，则 设置你的 AWS 账户中的流程将介绍如何创建AWS 账户。
2. 在该账户中，您必须具有以下为您在 AWS 账户和区域中的设备定义的 AWS IoT资源。

   创建 AWS IoT 资源 中的流程介绍了如何为您的 AWS 账户和区域中的设备创建这些资源。
   - 使用 AWS IoT注册并激活以验证设备的设备证书。

     该证书通常使用 AWS IoT事物对象创建并随附其上。虽然设备不需要某个事物对象才能连接到 AWS IoT，但它会使设备获得额外的 AWS IoT功能。
   - 附加到设备证书的策略，该证书授权其连接到 AWS IoT Core并执行您希望它执行的所有操作。
3. 能够访问您的 AWS 账户的设备端点的互联网连接。

   设备端点在 AWS IoT 设备数据和服务端点中描述，并且可以在 AWS IoT控制台中的设置页面看到。
4. 诸如 AWS IoTDevice SDK 提供的通信软件。本教程使用 AWS IoT Device SDK v2 for Python。

查看 MQTT 协议

在我们讨论示例应用程序之前，它能有助于了解 MQTT 协议。与其它网络通信协议（如 HTTP）相比，MQTT 协议具有一些优势，这使得它成为了 IoT 设备的常用选择。本部分回顾的是适用于本教程的 MQTT 的主要方面。有关如何将 MQTT 与 HTTP 进行比较的信息，请参阅 为设备通信选择协议。

MQTT 使用发布/订阅通信模型

MQTT 协议在其主机上使用发布/订阅通信模式。此模式与 HTTP 使用的请求/响应模式不同。借助 MQTT，设备可以与由唯一客户端 ID 标识的主机建立会话。要发送数据，设备会将主题标识的消息发布到主机中的消息代理上。要接收来自消息代理的消息，设备通过在订阅请求中向消息代理发送主题筛选条件，来订阅主题。

MQTT 支持持久性会话

消息代理接收来自设备的消息，并将消息发布到已订阅消息的设备。借助持久性会话（即使在初始设备断开连接时，仍能保持活动状态的会话），设备可以检索在断开连接时发布的消息。在设备端，MQTT 支持服务质量级别（QoS），以确保主机接收设备发送的消息。

查看 pubsub.py Device SDK 示例应用程序

本部分将回顾本教程中适用的来自 AWS IoTDevice SDK v2 for Python 中的 pubsub.py示例应用程序。在这里，我们将回顾它如何连接到 AWS IoT Core来发布和订阅 MQTT 消息。下一部分介绍了一些练习，帮助您了解设备如何与 AWS IoT Core连接及通信。

pubsub.py 示例应用程序演示了 MQTT 与 AWS IoT Core连接的各个方面：

- 通信协议
- 持久会话
- 服务质量
- 消息发布
- 消息订阅
- 设备断开和重新连接

通信协议

pubsub.py 示例演示了使用 MQTT 和 MQTT over WSS 协议进行的 MQTT 连接。AWS 通用运行时 (AWS CRT) 库提供低级通信协议支持，并包含在 AWS IoTDevice SDK v2 for Python 中。

MQTT

pubsub.py 示例在 mqtt_connection_builder 中调用 mtls_from_path（此处显示）以使用 MQTT 协议建立与 AWS IoT Core 的连接。mtls_from_path 使用 X.509 证书和 TLS v1.2 对设备进行身份验证。AWS CRT 库处理该连接的较低级别的详细信息。

```
mqtt_connection = mqtt_connection_builder.mtls_from_path(
    endpoint=args.endpoint,
    cert_filepath=args.cert,
    pri_key_filepath=args.key,
    ca_filepath=args.ca_file,
    client_bootstrap=client_bootstrap,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
    client_id=args.client_id,
```

```
    clean_session=False,
    keep_alive_secs=6
)
```

`endpoint`

您的 AWS 账户IoT 设备端点

在示例应用程序中，此值将从命令行传入。

`cert_filepath`

设备证书文件的路径。

在示例应用程序中，此值将从命令行传入。

`pri_key_filepath`

使用其证书文件创建的设备私有密钥文件的路径

在示例应用程序中，此值将从命令行传入。

`ca_filepath`

Root CA 文件的路径。仅当 MQTT 服务器使用信任存储中尚未存在的证书时才需要此项。

在示例应用程序中，此值将从命令行传入。

`client_bootstrap`

处理套接字通信活动的通用运行时对象

在示例应用程序中，此对象会在调用 `mqtt_connection_builder.mtls_from_path`前实例化。

`on_connection_interrupted`, `on_connection_resumed`

当设备连接中断和恢复时调用的回调函数

`client_id`

在 AWS 区域中唯一标识此设备的 ID

在示例应用程序中，此值将从命令行传入。

`clean_session`

启动新的持久会话，或者（如果会话已存在）重新连接到现有会话

`keep_alive_secs`

保持活动状态值（以秒为单位），在 CONNECT请求中发送。在此时间间隔内将自动发送 ping。如果服务器在此值的 1.5 倍之后没有收到 ping，假定连接丢失。

基于 WSS 的 MQTT

pubsub.py 示例在 mqtt_connection_builder 中调用 websockets_with_default_aws_signing（此处显示）以使用借助 WSS 的 MQTT 协议建立与 AWS IoT Core 的连接。websockets_with_default_aws_signing 使用 Signature V4 使用借助 WSS 的 MQTT 连接对设备进行身份验证。

```
mqtt_connection = mqtt_connection_builder.websockets_with_default_aws_signing(
    endpoint=args.endpoint,
    client_bootstrap=client_bootstrap,
    region=args.signing_region,
    credentials_provider=credentials_provider,
    websocket_proxy_options=proxy_options,
    ca_filepath=args.ca_file,
    on_connection_interrupted=on_connection_interrupted,
    on_connection_resumed=on_connection_resumed,
    client_id=args.client_id,
    clean_session=False,
    keep_alive_secs=6
)
```

`endpoint`

您的 AWS 账户IoT 设备端点

在示例应用程序中，此值将从命令行传入。

`client_bootstrap`

处理套接字通信活动的通用运行时对象

在示例应用程序中，此对象会在调用 mqtt_connection_builder.websockets_with_default_aws_signing前实例化。

`region`

Signature V4 身份验证使用的 AWS签名区域。在 `pubsub.py`中，它将在命令行中传递输入的参数。

在示例应用程序中，此值将从命令行传入。

`credentials_provider`

AWS 提供用于身份验证的凭证

在示例应用程序中，此对象会在调用 `mqtt_connection_builder.websockets_with_default_aws_signing`前实例化。

`websocket_proxy_options`

HTTP 代理选项（如果使用代理主机）

在示例应用程序中，此值在调用 `mqtt_connection_builder.websockets_with_default_aws_signing`前初始化。

`ca_filepath`

Root CA 文件的路径。仅当 MQTT 服务器使用信任存储中尚未存在的证书时才需要此项。

在示例应用程序中，此值将从命令行传入。

`on_connection_interrupted`, `on_connection_resumed`

当设备连接中断和恢复时调用的回调函数

`client_id`

在 AWS 区域中唯一标识此设备的 ID。

在示例应用程序中，此值将从命令行传入。

`clean_session`

启动新的持久会话，或者（如果会话已存在）重新连接到现有会话

`keep_alive_secs`

保持活动状态值（以秒为单位），在 `CONNECT`请求中发送。在此时间间隔内将自动发送 ping。如果服务器在此值的 1.5 倍之后没有收到 ping，则假定连接丢失。

HTTPS

HTTPS 怎么样？AWS IoT Core 支持发布 HTTPS 请求的设备。从编程的角度来看，设备会像其它应用程序一样将 HTTPS 请求发送到 AWS IoT Core。有关从设备发送 HTTP 消息的 Python 程序示例，请参阅使用 Python 的 `requests`库的 HTTPS 代码示例。本示例将使用 HTTPS 将消息发送到 AWS IoT Core，以便 AWS IoT Core将其解释为 MQTT 消息。

尽管 AWS IoT Core支持来自设备的 HTTPS 请求，请务必查看 为设备通信选择协议的消息，以便您可以对要使用哪种协议进行设备通信做出明智决定。

持久会话

在示例应用程序中，将 `clean_session`参数设置为 `False`即表示连接应该是持久连接。实际上，这意味着此调用打开的连接将重新连接到现有持久会话（如果存在）。否则，它会创建并连接到新的持久会话。

对于持久会话，如果设备未连接，则发送到设备的消息将由消息代理存储。当设备重新连接到持久会话时，消息代理会向设备发送它已订阅的所有已存储的消息。

如果没有持久会话，设备将不会接收设备未连接时发送的消息。使用哪个选项取决于您的应用程序，以及是否必须传达设备未连接时发生的消息。有关更多信息，请参阅 MQTT 持久性会话。

服务质量

当设备发布和订阅消息时，可以设置首选的服务质量 (QoS)。AWS IoT 支持用于发布和订阅操作的 QoS 级别 0 和 1。有关 AWS IoT中 QoS 级别的更多信息，请参阅 MQTT 服务质量 (QoS) 选项。

适用于 Python 的 AWSCRT 运行时为其支持的 QoS 级别定义了这些常量：

Python QoS 级别

MQTT QoS 级别	SDK 使用的 Python 符号值	描述
QoS 级别 0	`mqtt.QoS.AT_MOST_ONCE`	无论是否收到消息，均只会尝试发送一次消息。如果出现诸如设备未连接或存在网络错误的情况，则消息可能根本不会发送。
QoS 级别 1	`mqtt.QoS.AT_LEAST_ONCE`	消息将重复发送，直至收到PUBACK确认。

在示例应用程序中，发布和订阅请求的 QoS 级别为 1 (`mqtt.QoS.AT_LEAST_ONCE`)。

- 发布时的 QoS

  当设备发布 QoS 级别为 1 的消息时，它会重复发送消息，直至从消息代理收到 PUBACK响应。如果设备未连接，消息将在重新连接后排队等待发送。

- 订阅时的 QoS

  当设备订阅 QoS 级别为 1 的消息时，消息代理会保存设备订阅的消息，直到这些消息可以发送到设备。消息代理会重新发送消息，直至收到设备发出的 PUBACK响应。

消息发布

成功建立同 AWS IoT Core的连接后，设备可以发布消息。`pubsub.py` 示例通过调用 `mqtt_connection`对象的 `publish`操作来完成此操作。

```
mqtt_connection.publish(
    topic=args.topic,
    payload=message,
    qos=mqtt.QoS.AT_LEAST_ONCE
)
```

`topic`

标识消息的消息主题名称

在示例应用程序中，这是从命令行传入的。

`payload`

格式化为字符串的消息负载（例如，JSON 文档）

在示例应用程序中，这是从命令行传入的。

JSON 文档是一种常见的负载格式，并且由其它 AWS IoT服务所认可；但是，消息负载的数据格式，可以是发布者和订阅者同意的任何内容。但在某些情况下，对于大多数操作来说，其它 AWS IoT服务只能识别 JSON 和 CBOR。

`qos`

此消息的 QoS 级别

消息订阅

从 AWS IoT和其它服务及设备接收消息时，设备会按其主题名称订阅这些消息。设备可以通过指定主题名称来订阅单个消息，也可以通过指定主题筛选条件来订阅一组消息（筛选条件中可以包含通配符）。pubsub.py 示例使用此处显示的代码订阅消息并注册回调函数，以便在收到消息后处理消息。

```
subscribe_future, packet_id = mqtt_connection.subscribe(
    topic=args.topic,
    qos=mqtt.QoS.AT_LEAST_ONCE,
    callback=on_message_received
)
subscribe_result = subscribe_future.result()
```

`topic`

要订阅的主题。这可以是主题名称或主题筛选条件。

在示例应用程序中，这是从命令行传入的。

`qos`

在设备断开连接时，消息代理是否应该存储这些消息。

`mqtt.QoS.AT_LEAST_ONCE` 的值（QoS 级别 1），创建连接时需要指定持久性会话 (`clean_session=False`)。

`callback`

要调用以处理已订阅消息的函数。

`mqtt_connection.subscribe` 函数返回未来和数据包 ID。如果订阅请求成功初始化，则返回的数据包 ID 大于 0。要确保消息代理已接收并注册订阅，您必须等待异步操作的结果返回，如代码示例所示。

回调函数

pubsub.py 示例中的回调函数会在设备接收订阅的消息时处理这些消息。

```
def on_message_received(topic, payload, **kwargs):
    print("Received message from topic '{}': {}".format(topic, payload))
    global received_count
    received_count += 1
```

```
    if received_count == args.count:
        received_all_event.set()
```

`topic`

消息的主题

这是收到的消息的特定主题名称，即使您订阅了主题筛选条件也是如此。

`payload`

消息负载

此格式是特定于应用程序的格式。

`kwargs`

可能的其它实际参数，如 `mqtt.Connection.subscribe`中所述。

在 `pubsub.py`示例中，`on_message_received` 仅显示主题及其负载。它还会计算在达到限制后收到的结束程序的消息。

您的应用程序将评估主题和负载，以确定要执行的操作。

设备断开和重新连接

`pubsub.py` 示例包括在设备断开连接和重新建立连接时调用的回调函数。您的设备对这些事件采取的操作是特定于应用程序的。

当设备首次连接时，它必须订阅主题才能接收。如果设备在重新连接时存在会话，则会恢复其订阅，并在设备重新连接后，将来自这些订阅的所有存储消息发送到设备。

如果设备的会话在重新连接时不再存在，则必须重新订阅其订阅。持久会话的生命周期有限，当设备断开连接太长时间时，可能会过期。

连接设备并与 AWS IoT Core通信

本部分介绍了一些练习，可帮助您探索将设备连接到 AWS IoT Core的不同方面。在这些练习中，您将使用 AWS IoT控制台中的 MQTT 测试客户端查看您的设备发布的内容，并将消息发布到您的设备。这些练习使用来自 AWS IoTDevice SDK v2 for Python 的 `pubsub.py`示例并根据您在 入门 AWS IoT Core教程中的经验构建。

在本部分中，您将：

- 订阅通配符主题筛选条件
- 处理主题筛选条件订阅
- 从您的设备发布消息

对于这些练习，您将从 pubsub.py示例程序开始操作。

> **Note**
>
> 这些练习假定您已完成 入门 AWS IoT Core教程并使用该教程中的设备终端窗口。

订阅通配符主题筛选条件

在本练习中，您将修改用于调用 pubsub.py的命令行以订阅通配符主题筛选条件，并根据消息主题处理收到的消息。

练习流程

在本练习中，假设您的设备包含温度控制和光控。它使用这些主题名称来标识有关的消息。

1. 在开始练习之前，请尝试根据 入门 AWS IoT Core教程在您的设备上运行此命令，确保一切都准备就绪，可正常进行练习。

   ```
   cd ~/aws-iot-device-sdk-python-v2/samples
   python3 pubsub.py --topic topic_1 --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/
   certs/device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint
   ```

   您看到的输出应该与您在入门教程看到的一样。
2. 在本练习中，请更改这些命令行参数。

操作	命令行参数	效果
添加	--message ""	配置 pubsub.py 以仅侦听
添加	--count 2	收到两条消息后结束程序
更改	--topic device/+/details	设定要订阅的主题筛选条件

对初始命令行进行这些更改将得到此命令行。在设备的终端窗口中输入此命令。

```
python3 pubsub.py --message "" --count 2 --topic device/+/details --ca_file
 ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
private.pem.key --endpoint your-iot-endpoint
```

程序应该类似如下所示：

```
Connecting to a3qexamplesffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-24d7cdcc-cc01-458c-8488-2d05849691e1'...
Connected!
Subscribing to topic 'device/+/details'...
Subscribed with QoS.AT_LEAST_ONCE
Waiting for all messages to be received...
```

如果您在终端上看到类似的内容，则表示您的设备已准备就绪，并侦听主题名称以 device开始并以结束 /detail的消息。所以，让我们来测试吧。

3. 以下是您的设备可能会收到的几条消息。

主题名称	消息负载
device/temp/details	{ "desiredTemp": 20, "currentTemp": 15 }
device/light/details	{ "desiredLight": 100, "currentLight": 50 }

4. 在 AWS IoT控制台中使用 MQTT 测试客户端，将上一步中描述的消息发送到您的设备。

   a. 在 AWS IoT控制台中打开 MQTT 测试客户端。

   b. 在 Subscribe to topic（订阅主题）中，在 Subscription topic field（订阅主题字段）输入主题筛选条件：**device/+/details**，然后选择 Subscribe to topic（订阅主题）。

   c. 在 MQTT 测试客户端中的 Subscriptions（订阅）列中，选择device/+/details。

   d. 对于上表中的每个主题，请在 MQTT 测试客户端中执行以下操作：

      1. 在 Publish（发布）中，在表中输入 Topic name（主题名称）列的值。
      2. 在主题名称下方的消息负载字段中，在表中输入 Message payload（消息负载）列的值。

3. 观看 pubsub.py运行所在的终端窗口，并在 MQTT 测试客户端中选择 Publish to topic（发布到主题）。

   您应该看到该消息是由终端窗口中的 pubsub.py接收的。

练习结果

有了这个，pubsub.py，使用通配符主题筛选条件订阅消息，接收消息并在终端窗口中显示这些消息。请注意您如何订阅单个主题筛选条件，并调用回调函数来处理具有两个不同主题的消息。

处理主题筛选条件订阅

在上一练习的基础上，修改 pubsub.py示例应用程序以评估消息主题并根据主题处理订阅的消息。

练习流程

评估消息主题

1. 将 pubsub.py复制到 pubsub2.py。
2. 在您常用的文本编辑器或 IDE 中打开 pubsub2.py。
3. 在 pubsub2.py中，查找 on_message_received函数。
4. 在 on_message_received中，在以 print("Received message开头的行之后和在以 global received_count开头的行之前插入以下代码。

```
topic_parsed = False
if "/" in topic:
    parsed_topic = topic.split("/")
    if len(parsed_topic) == 3:
        # this topic has the correct format
        if (parsed_topic[0] == 'device') and (parsed_topic[2] == 'details'):
            # this is a topic we care about, so check the 2nd element
            if (parsed_topic[1] == 'temp'):
                print("Received temperature request: {}".format(payload))
                topic_parsed = True
            if (parsed_topic[1] == 'light'):
                print("Received light request: {}".format(payload))
                topic_parsed = True
if not topic_parsed:
    print("Unrecognized message topic.")
```

5. 保存更改并使用此命令行运行修改后的程序。

```
python3 pubsub2.py --message "" --count 2 --topic device/+/details --ca_file
 ~/certs/Amazon-root-CA-1.pem --cert ~/certs/device.pem.crt --key ~/certs/
private.pem.key --endpoint your-iot-endpoint
```

6. 在 AWS IoT控制台中打开 MQTT 测试客户端。
7. 在 Subscribe to topic（订阅主题）中，在 Subscription topic field（订阅主题字段）输入主题筛选条件：**device/+/details**，然后选择 Subscribe to topic（订阅主题）。
8. 在 MQTT 测试客户端中的 Subscriptions（订阅）列中，选择device/+/details。
9. 对于此表中的每个主题，请在 MQTT 测试客户端中执行以下操作：

主题名称	消息负载
device/temp/details	{ "desiredTemp": 20, "currentTemp": 15 }
device/light/details	{ "desiredLight": 100, "currentLight": 50 }

   1. 在 Publish（发布）中，在表中输入 Topic name（主题名称）列的值。
   2. 在主题名称下方的消息负载字段中，在表中输入 Message payload（消息负载）列的值。
   3. 观看 pubsub.py运行所在的终端窗口，并在 MQTT 测试客户端中选择 Publish to topic（发布到主题）。

   您应该看到该消息是由终端窗口中的 pubsub.py接收的。

您应在终端窗口中看到类似的内容。

```
Connecting to a3qexamplesffp-ats.iot.us-west-2.amazonaws.com with client ID 'test-
af794be0-7542-45a0-b0af-0b0ea7474517'...
Connected!
Subscribing to topic 'device/+/details'...
Subscribed with QoS.AT_LEAST_ONCE
Waiting for all messages to be received...
Received message from topic 'device/light/details': b'{ "desiredLight": 100,
 "currentLight": 50 }'
```

```
Received light request: b'{ "desiredLight": 100, "currentLight": 50 }'
Received message from topic 'device/temp/details': b'{ "desiredTemp": 20,
 "currentTemp": 15 }'
Received temperature request: b'{ "desiredTemp": 20, "currentTemp": 15 }'
2 message(s) received.
Disconnecting...
Disconnected!
```

练习结果

在本练习中，您添加了代码，以便示例应用程序能够识别并处理回调函数中的多条消息。有了这个，您的设备便可以接收消息并对它们采取行动。

您的设备接收和处理多条消息的另一种方式是单独订阅不同的消息，并将每个订阅分配给自己的回调函数。

从您的设备发布消息

您可以使用 pubsub.py 示例应用程序从您的设备发布消息。虽然它会按原样发布消息，但消息不能作为 JSON 文档读取。本练习修改了示例应用程序，以便能够在可以被 AWS IoT Core读取的消息负载中发布 JSON 文档。

练习流程

在本练习中，以下消息将以 `device/data`主题发布。

```
{
    "timestamp": 1601048303,
    "sensorId": 28,
    "sensorData": [
        {
        "sensorName": "Wind speed",
        "sensorValue": 34.2211224
        }
    ]
}
```

准备 MQTT 测试客户端以监控本练习中的消息

1. 在 Subscribe to topic（订阅主题）中，在 Subscription topic field（订阅主题字段）输入主题筛选条件：**`device/data`**，然后选择 Subscribe to topic（订阅主题）。

2. 在 Subscriptions（订阅）列中，选择 device/data（设备/数据）。
3. 将 MQTT 测试客户端窗口保持打开状态，等待来自设备的消息。

使用 pubsub.py 示例应用程序发送 JSON 文档

1. 在您的设备上，将 pubsub.py复制到 pubsub3.py。
2. 编辑 pubsub3.py来更改其所发布消息的格式。
   a. 在文本编辑器中打开 pubsub3.py。
   b. 查找此代码行：

      ```
      message = "{} [{}]".format(message_string, publish_count)
      ```

   c. 将其更改为：

      ```
      message = "{}".format(message_string)
      ```

   d. 查找此代码行：

      ```
      message_json = json.dumps(message)
      ```

   e. 将其更改为：

      ```
      message = "{}".json.dumps(json.loads(message))
      ```

   f. 保存您的更改。
3. 在您的设备上运行此命令以发送消息两次。

```
python3 pubsub3.py  --ca_file ~/certs/Amazon-root-CA-1.pem  --cert ~/certs/
device.pem.crt  --key ~/certs/private.pem.key  --topic device/data  --count 2 --
message '{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'  --endpoint your-iot-endpoint
```

4. 在 MQTT 测试客户端中，检查它是否已解释并格式化了消息负载中的 JSON 文档，如下所示：

预设情况下，pubsub3.py 也会订阅它发送的消息。您应该看到它在应用程序的输出中收到了消息。终端窗口应类似如下所示。

```
Connecting to a3qEXAMPLEsffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-5cff18ae-1e92-4c38-a9d4-7b9771afc52f'...
Connected!
Subscribing to topic 'device/data'...
Subscribed with QoS.AT_LEAST_ONCE
Sending 2 message(s)
Publishing message to topic 'device/data':
 {"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}
Received message from topic 'device/data':
 b'{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'
Publishing message to topic 'device/data':
 {"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}
Received message from topic 'device/data':
 b'{"timestamp":1601048303,"sensorId":28,"sensorData":[{"sensorName":"Wind
 speed","sensorValue":34.2211224}]}'
2 message(s) received.
Disconnecting...
Disconnected!
```

练习结果

有了这个，您的设备可以生成要发送到 AWS IoT Core的消息以测试基本连通性并提供 AWS IoT Core来处理的设备消息。例如，您可以使用此应用程序从您的设备发送测试数据以测试 AWS IoT规则操作。

查看结果

本教程中的示例为您提供了有关设备如何与 AWS IoT Core（这是 AWS IoT解决方案的一个基础部分）通信的基础知识实践经验。当您的设备能够与 AWS IoT Core通信时，他们可以将消息传递给 AWS服务及其可以执行操作的其它设备 同样，AWS 服务和其它设备可以处理消息发回到您的设备所产生的信息。

如果您准备好进一步探索 AWS IoT Core，请尝试以下教程：

- the section called “发送 Amazon SNS 通知”
- the section called “将设备数据存储在 DynamoDB 表中”
- the section called “使用 AWS Lambda 函数格式化通知”

教程：使用 AWS IoT Device SDK for Embedded C

本节介绍如何运行 AWS IoT Device SDK for Embedded C。

本节中的过程

- 第 1 步：安装 AWS IoT Device SDK for Embedded C
- 步骤 2：配置示例应用
- 步骤 3：构建并运行示例应用程序

第 1 步：安装 AWS IoT Device SDK for Embedded C

AWS IoT Device SDK for Embedded C 通常针对需要优化 C 语言运行时的资源受限的设备。您可以在任何操作系统上使用此 SDK，并将其托管在任何类型的处理器（例如 MCU 和 MPU）上。如果您有更多的内存和处理资源可用，我们建议您使用更高阶的 AWS IoT 设备和移动 SDK（例如 C++ JavaScript、Java 和 Python）。

通常， AWS IoT Device SDK for Embedded C 适用于使用运行嵌入式操作系统的 MCU 或低端 MPU 的系统。对于本部分中的编程示例，我们假定您的设备使用 Linux。

Example

1. 从下载 AWS IoT Device SDK for Embedded C 到您的设备GitHub。

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-embedded-c.git --recurse-submodules
   ```

   这将在当前目录中创建一个名为 aws-iot-device-sdk-embedded-c的目录。

2. 前往到该目录并签出最新版本。有关最新版本标签，请参阅 github.com/aws/-embedd aws-iot-device-sdk ed- c/Tags。

   ```
   cd aws-iot-device-sdk-embedded-c
   git checkout latest-release-tag
   ```

3. 安装 OpenSSL 1.1.0 或更高版本。当通过软件包管理器安装时，OpenSSL 开发库通常被称为“libssl-dev”或“openssl-devel”。

   ```
   sudo apt-get install libssl-dev
   ```

步骤 2：配置示例应用

AWS IoT Device SDK for Embedded C 包括供您尝试的示例应用程序。为简单起见，本教程使用了mqtt_demo_mutual_auth应用程序，该应用程序演示了如何连接到 AWS IoT Core 消息代理以及如何订阅和发布 MQTT 主题。

1. 将您在 入门 AWS IoT Core中创建的证书和私有密钥复制到 build/bin/certificates目录中。

   > **Note**
   >
   > 设备和根 CA 证书可能会过期或被吊销。如果您的证书过期或被吊销，则您必须将新的 CA 证书或私有密钥和设备证书复制到您的设备上。

2. 您必须使用您的个人 AWS IoT Core 终端节点、私钥、证书和根 CA 证书配置示例。导航到 aws-iot-device-sdk-embedded-c/demos/mqtt/mqtt_demo_mutual_auth 目录。

   如果您已 AWS CLI 安装，则可以使用此命令来查找您账户的终端节点 URL。

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

如果您尚未 AWS CLI 安装，请打开您的AWS IoT 主机。在导航窗格中，依次选择 Manage (管理) 和 Things (事物)。为您的设备选择 IoT 事物，然后选择 Interact（交互）。您的终端节点显示在事物详细信息页面的 HTTPS 部分中。

3. 打开 demo_config.h文件并更新以下各项的值：

   AWS_IOT_ENDPOINT

   您的私有终端节点。

   CLIENT_CERT_PATH

   您的证书文件路径，例如 certificates/device.pem.crt"。

   CLIENT_PRIVATE_KEY_PATH

   您的私有密钥文件名，例如 certificates/private.pem.key。

   例如：

```
// Get from demo_config.h
// =================================================
#define AWS_IOT_ENDPOINT                "my-endpoint-ats.iot.us-
east-1.amazonaws.com"
#define AWS_MQTT_PORT                   8883
#define CLIENT_IDENTIFIER               "testclient"
#define ROOT_CA_CERT_PATH               "certificates/AmazonRootCA1.crt"
#define CLIENT_CERT_PATH                "certificates/my-device-cert.pem.crt"
#define CLIENT_PRIVATE_KEY_PATH         "certificates/my-device-private-key.pem.key"
// =================================================
```

4. 使用此命令检查您的设备上是否安装了 CMake。

```
cmake --version
```

如果您看到编译器的版本信息，则可以继续下一部分。

如果出现错误或看不到任何信息，则需要使用此命令安装 cmake 软件包。

```
sudo apt-get install cmake
```

再次运行 cmake --version命令，确认 CMake 已安装并且您已准备好继续操作。

5. 使用此命令检查您的设备上是否安装了开发工具。

```
gcc --version
```

如果您看到编译器的版本信息，则可以继续下一部分。

如果出现错误或看不到任何编译器信息，则需要使用此命令安装 build-essential软件包。

```
sudo apt-get install build-essential
```

再次运行 gcc --version命令，确认构建工具已安装并且您已准备好继续操作。

步骤 3：构建并运行示例应用程序

运行 AWS IoT Device SDK for Embedded C 示例应用程序

1. 导航到 aws-iot-device-sdk-embedded-c并创建目录。

```
mkdir build && cd build
```

2. 输入以下 CMake 命令以生成 Makefiles 构建所需的文件。

```
cmake ..
```

3. 输入以下命令以构建可执行应用程序文件。

```
make
```

4. 使用此命令运行 mqtt_demo_mutual_auth应用程序。

```
cd bin
./mqtt_demo_mutual_auth
```

您应该可以看到类似于如下所示的输出内容：

```
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:584] Establishing a TLS session to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com:8883.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1264] Creating an MQTT connection to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com.
[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
[INFO] [MQTT] [core_mqtt_serializer.c:970] CONNACK session present bit not set.
[INFO] [MQTT] [core_mqtt_serializer.c:912] Connection accepted.
[INFO] [MQTT] [core_mqtt.c:1526] Received MQTT CONNACK successfully from broker.
[INFO] [MQTT] [core_mqtt.c:1792] MQTT connection established with the broker.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1033] MQTT connection successfully established with broker.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1296] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1314] Subscribing to the MQTT topic testclient/example/topic.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1097] SUBSCRIBE sent for topic testclient/example/topic to broker.


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=3.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:921] Subscribed to the topic testclient/example/topic. with maximum QoS 1.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1358] Sending Publish to the MQTT topic testclient/example/topic.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:1195] PUBLISH sent for topic testclient/example/topic to broker with packet ID 2.


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=2.
[INFO] [MQTT] [core_mqtt.c:1126] Ack packet deserialized with result: MQTTSuccess.
[INFO] [MQTT] [core_mqtt.c:1139] State record updated. New state=MQTTPublishDone.
[INFO] [DEMO] [mqtt_demo_mutual_auth.c:946] PUBACK received for packet id 2.


[INFO] [DEMO] [mqtt_demo_mutual_auth.c:672] Cleaned up outgoing publish packet with packet id 2.


[INFO] [MQTT] [core_mqtt.c:855] Packet received. ReceivedBytes=40.
[INFO] [MQTT] [core_mqtt.c:1015] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
```

现在，您的设备已 AWS IoT 使用连接到 AWS IoT Device SDK for Embedded C。

您还可以使用 AWS IoT 控制台查看示例应用程序正在发布的 MQTT 消息。有关如何在 AWS IoT 控制台中使用 MQTT 客户端的信息，请参阅the section called “使用 MQTT 客户端查看 AWS IoT MQTT 消息”。

创建将设备数据路由到其他服务的 AWS IoT 规则

这些教程向您展示了如何使用一些更常见的 AWS IoT 规则操作来创建和测试规则。

AWS IoT 规则将数据从您的设备发送到其他 AWS 服务。它们侦听特定的 MQTT 消息，格式化消息负载中的数据，并将结果发送到其它 AWS 服务。

我们建议您按照这里显示的顺序尝试这些功能，即使您的目标是使用 Lambda 或更复杂的函数创建一个规则。这些教程是按照从基本到复杂的顺序提供的。它们以递增方式呈现新概念，从而帮助您了解可用于创建没有特定教程的规则操作的概念。

> **Note**
>
> AWS IoT 规则可帮助您将物联网设备中的数据发送到其他 AWS 服务。但是，要成功完成此操作，您需要了解要向其发送数据的其它服务的工作知识。虽然这些教程提供了完成任务所需的信息，但您可能会发现在解决方案中使用数据之前，了解有关要向其发送数据的服务的详细信息会很有帮助。对其他 AWS 服务的详细说明不在这些教程的范围之内。

教程场景概览

这些教程的场景是定期发布数据的天气传感器设备。在这个虚构系统中有许多这样的传感器设备。但是，本部分中的教程侧重于单个设备，同时展示了如何容纳多个传感器。

本节中的教程向您展示如何使用 AWS IoT 规则对这个虚构的天气传感器设备系统执行以下任务。

- 教程：重新发布 MQTT 消息

  本教程介绍如何将从天气传感器收到的 MQTT 消息重新发布为仅包含传感器 ID 和温度值的消息。它只使用 AWS IoT Core 服务，并演示简单 SQL 查询以及如何使用 MQTT 客户端来测试您的规则。

- 教程：发送 Amazon SNS 通知

  本教程介绍如何在天气传感器设备的值超过特定值时发送 SNS 消息。它以上一教程中介绍的概念为基础，并添加了如何使用另一项 AWS 服务，即亚马逊简单通知服务 (Amazon SNS)。

  如果您是 Amazon SNS 的新用户，请查看其入门练习，然后再开始本教程。

- 教程：将设备数据存储在 DynamoDB 表中

  本教程介绍如何将来自气象传感器设备的数据存储在数据库表中。它使用规则查询语句和替代模板来设置目标服务的消息数据的格式，Amazon DynamoDB。

  如果您是的 DynamoDB 用户，请查看其入门练习，然后再开始本教程。

- 教程：使用 AWS Lambda 函数格式化通知

  本教程介绍如何调用 Lambda 函数来重新格式化设备数据，然后将其作为文本消息发送。它在函数中添加了 Python 脚本和 AWS SDK AWS Lambda函数，以便使用来自天气传感器设备的消息有效载荷数据进行格式化并发送短信。

  如果您是 Lambda 的新用户，请查看 Lambda 的入门练习，然后再开始本教程。

AWS IoT 规则概述

所有这些教程都创建了 AWS IoT 规则。

对于将数据从一台设备发送到另一 AWS 服务的 AWS IoT 规则，它使用：

- 规则查询语句，由以下内容组成：
  - 一个 SQL SELECT 子句，用于从消息负载中选择数据并设置其格式
  - 标识要使用的消息的主题筛选条件（规则查询语句中的 FROM 对象）
  - 可选条件语句（SQL WHERE 子句），用于指定执行操作的特定条件
- 至少一个规则操作

设备会向主题发布 MQTT 消息。SQL SELECT 语句中的主题筛选条件标识要应用规则的 MQTT 主题。SQL SELECT 语句中指定的字段将来自传入 MQTT 消息负载的数据格式化，以供规则的操作使用。有关规则操作的完整列表，请参阅 AWS IoT 规则操作。

本部分中的教程

- 教程：重新发布 MQTT 消息
- 教程：发送 Amazon SNS 通知
- 教程：将设备数据存储在 DynamoDB 表中
- 教程：使用 AWS Lambda 函数格式化通知

教程：重新发布 MQTT 消息

本教程演示如何创建在收到指定的 MQTT 消息时发布 MQTT 消息的 AWS IoT 规则。可以在发布之前通过规则修改传入的消息负载。这样就可以创建针对特定应用程序量身定制的消息，而无需更改设备或固件。您还可以使用规则的筛选功能仅在满足特定条件时发布消息。

通过规则重新发布的消息就像任何其他 AWS IoT 设备或客户端发送的消息一样。设备可以订阅重新发布的消息，就像订阅任何其它 MQTT 消息主题一样。

您将在本教程中学到的内容：

- 如何在规则查询语句中使用简单的 SQL 查询和函数
- 如何使用 MQTT 客户端测试规则 AWS IoT

完成本教程需要大约 30 分钟。

在本教程中，您将：

- 查看 MQTT 主题和规则 AWS IoT
- 步骤 1：创建 AWS IoT 规则以重新发布 MQTT 消息
- 步骤 2：测试您的新规则
- 步骤 3：查看结果和后续步骤

在开始本教程之前，请确保您具有：

- 设置你的 AWS 账户

  你需要你的 AWS 账户 和 AWS IoT 主机才能完成本教程。
- 审核 使用 MQTT 客户端查看 AWS IoT MQTT 消息

  请确保您可以使用 MQTT 客户端订阅和发布主题。您将使用 MQTT 客户端在此流程中测试新规则。

查看 MQTT 主题和规则 AWS IoT

在讨论 AWS IoT 规则之前，先了解 MQTT 协议会有所帮助。在 IoT 解决方案中，MQTT 协议相比于其它网络通信协议（如 HTTP）更具优势，这使得它成为 IoT 设备使用的常用选择。本部分回顾了 MQTT 的关键方面，因为它们适用于本教程。有关如何将 MQTT 与 HTTP 进行比较的信息，请参阅 为设备通信选择协议。

MQTT 协议

MQTT 协议在其主机上使用发布/订阅通信模式。为了发送数据，设备会向消息代理发布由主题标识的 AWS IoT 消息。要接收来自消息代理的消息，设备通过在订阅请求中向消息代理发送主题筛选条件，来订阅其将接收的主题。 AWS IoT 规则引擎从消息代理接收 MQTT 消息。

AWS IoT 规则

AWS IoT 规则由规则查询语句和一个或多个规则操作组成。当 AWS IoT 规则引擎接收 MQTT 消息，则这些元素将按如下方式操作消息。

- 规则查询语句

  规则的查询语句描述了要使用的 MQTT 主题，解释消息负载中的数据，并按类似于常用 SQL 数据库使用的语句的 SQL 语句的描述设置数据的格式。查询语句的结果是发送到规则操作的数据。

- 规则操作

  规则中的每个规则操作都对规则的查询语句生成的数据起作用。 AWS IoT 支持许多规则操作。但是，在本教程中，您将专注于 Republish规则操作，该操作将查询语句的结果发布为带有特定主题的 MQTT 消息。

步骤 1：创建 AWS IoT 规则以重新发布 MQTT 消息

你将在本教程中创建的 AWS IoT 规则订阅了 device/*device_id*/data MQTT 主题，其中 *device_id* 是发送消息的设备的 ID。这些主题由主题筛选条件描述为 device/+/data，其中，+ 是匹配两个正斜杠字符之间的任何字符串的通配符。

当规则收到来自匹配主题的消息时，它会重新发布 device_id和 temperature的值，以 device/data/temp主题发布为新的 MQTT 消息。

例如，具有 device/22/data主题的 MQTT 消息负载如下所示：

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

该规则将 temperature值从消息负载中取出，将 device_id从主题中取出，并将其作为 MQTT 消息以 device/data/temp为主题和类似下面这样的消息负载重新发布：

```
{
  "device_id": "22",
  "temperature": 28
}
```

使用此规则，只需要设备 ID 和温度数据的设备会订阅 device/data/temp主题以仅接收该信息。

创建重新发布 MQTT 消息的规则

1. 打开AWS IoT 控制台的 “规则” 中心。

2. 在 Rules（规则）中，选择 Create（创建）并开始创建新规则。
3. 在 Create a rule（创建规则）顶部：

   a. 在 Name（名称）下，输入规则的名称。在本教程中，将其命名为 **republish_temp**。

      请记住，规则名称在您的账户和区域中必须唯一，并且不能包含任何空格 我们在此名称中使用了下划线字符来分隔规则名称中的两个单词。

   b. 在 Description（说明）中，描述规则。

      有意义的描述可以帮助您记住此规则的作用以及您创建它的原因。描述可以根据需要延长，因此请尽可能详细。

4. 在 Create a rule（创建规则）的 Rule query statement（规则查询语句）中：

   a. 在 Using SQL version（使用 SQL 版本）中，选择 **2016-03-23**。
   b. 在 Rule query statement（规则查询语句）编辑框中，输入语句：

      ```
      SELECT topic(2) as device_id, temperature FROM 'device/+/data'
      ```

      本语句：

      - 侦听主题符合 `device/+/data`主题筛选条件的 MQTT 消息。
      - 从主题字符串中选择第二个元素，并将其分配给 `device_id`字段。
      - 从消息负载中选择 `temperature`字段的值，并将其分配给 `temperature`字段。

5. 在 Set one or more actions（设置一个或多个操作）中：

   a. 要打开此规则的规则操作列表，请选择 Add action（添加操作）。
   b. 在 “选择操作” 中，选择 “将消息重新发布到 AWS IoT 主题”。
   c. 在操作列表底部，选择 Configure action（配置操作）以打开选定操作的配置页面。

6. 在 Configure action（配置操作）中：

   a. 在 Topic（主题）中输入 **device/data/temp**。这是此规则将发布的消息 MQTT 主题。
   b. 在 Quality of Service（服务质量）中，选择 0 - The message is delivered zero or more times（0-消息传递零次或多次）。
   c. 在选择或创建角色以授予执行此操作的 AWS IoT 访问权限中：

      i. 选择 Create Role(创建角色)。Create a new role（创建新角色）对话框打开。
      ii. 输入描述新角色的名称。在本教程中，请使用 **republish_role**。

创建新角色时，将创建用于执行规则操作的正确策略并将其附加到新角色。如果更改此规则操作的主题或在其它规则操作中使用此角色，则必须更新该角色的策略以授权新主题或操作。要更新现有角色，请在本部分中选择 Update role（更新角色）。

iii. 选择 Create Role（创建角色）以创建角色并关闭对话框。

d. 选择 Add action（添加操作）将操作添加到规则并返回到 Create a rule（创建规则）页。

7. “将消息重新发布到 AWS IoT 主题” 操作现在列在 “设置一个或多个操作” 中。

   在新动作的磁贴中，在 Republish a message to an AWS IoT topic（将消息重新发布至 IoT 主题）下，您可以看到重新发布操作将发布到的主题。

   这是您将添加到此规则的唯一规则操作。

8. 在 Create a rule（创建规则）中，向下滚动到底部，然后选择 Create rule（创建规则）以创建规则并完成此步骤。

步骤 2：测试您的新规则

要测试新规则，您将使用 MQTT 客户端发布和订阅此规则所使用的 MQTT 消息。

在新窗口中打开 AWS IoT 控制台中的 MQTT 客户端。这将允许您在不丢失 MQTT 客户端配置的情况下编辑规则。如果您让其转到控制台中的其它页面，MQTT 客户端不会保留任何订阅或消息日志。

如需使用 MQTT 客户端来测试您的规则

1. 在 AWS IoT 控制台中的 MQTT 客户端中，订阅输入主题，在本例中为 `device/+/data`。

   a. 在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择 Subscribe to a topic（订阅主题）。

   b. 在 Subscription topic（订阅主题）中，输入输入主题筛选条件的主题，**`device/+/data`**。

   c. 将其它字段保留为默认设置。

   d. 选择 Subscribe to topic（订阅主题）。

      在 Subscriptions（订阅）栏，在 Publish to a topic（发布到主题）项下，将会显示 **`device/+/data`**。

2. 订阅您的规则将发布的主题：`device/data/temp`。

   a. 在 Subscriptions（订阅）中，再次选择 Subscribe to a topic（订阅主题），并在 Subscription topic（订阅主题）中，输入重新发布消息的主题，**`device/data/temp`**。

b. 将其它字段保留为默认设置。

c. 选择 Subscribe to topic（订阅主题）。

在 Subscriptions（订阅）栏，在 device/+/data 下，将会显示 **device/data/temp**。

3. 使用特定设备 ID **device/22/data**向输入主题发布消息。您无法发布到包含通配符的 MQTT 主题。

a. 在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择 Subscribe to a topic（订阅主题）。

b. 在 Publish（发布）字段中，输入输入主题名称，**device/22/data**。

c. 复制此处显示的示例数据，然后在主题名称下方的编辑框中粘贴示例数据。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. 要发送您的 MQTT 消息，请选择 Publish to topic（发布到主题）。

4. 查看已发送的消息。

a. 在 MQTT 客户端中，在 Subscriptions（订阅）项下，您之前订阅的两个主题旁边有一个绿点。

绿色圆点表示自您上次查看它们以来已收到一条或多条新消息。

b. 在 Subscriptions（订阅）中，选择 device/+/data 来检查消息负载是否与刚刚发布的内容匹配，如下所示：

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
```

```
}
```

c. 在 Subscriptions（订阅）中，选择 device/data/temp 来检查重新发布的消息负载，如下所示：

```
{
  "device_id": "22",
  "temperature": 28
}
```

请注意，device_id 值是一个带引号的字符串，temperature 值为数字。这是因为 topic() 函数从输入消息的主题名称中提取字符串，而 temperature 值使用输入消息负载中的数值。

如果您想使 device_id 值为数值，请在规则查询语句中将 topic(2) 替换为：

```
cast(topic(2) AS DECIMAL)
```

请注意，将 topic(2) 值设置为数字值仅在主题的该部分仅包含数字字符时才起作用。

5. 如果您看到正确的消息已发布到 device/data/temp 主题，那么您的规则已起作用。请在下一部分中了解有关“重新发布规则”操作的更多信息。

如果您没有看到正确的消息发布到 device/+/data 或者 device/data/temp 主题中，请查看故障排除提示。

重新发布消息规则疑难解答

如果您没有看到期望的结果，可检查以下事项。

- 您收到了一个错误的广告条

  如果在您发布输入消息时出现错误，请先更正该错误。以下步骤可帮助您更正此错误。

- 您未在 MQTT 客户端中看到输入消息

  每次将输入消息发布到 device/22/data 主题时，如果您如流程中所述订阅了 device/+/data 主题筛选条件，则消息应当会在 MQTT 客户端中显示。

  要检查的事项

  - 检查您订阅的主题筛选条件

如果您如流程中所述订阅了输入消息主题，则每次发布输入消息时都应看到该输入消息的副本。

如果没有看到该消息，请检查您订阅的主题名称，并将其与您发布的主题进行比较。主题名称区分大小写，您订阅的主题必须与发布消息负载的主题相同。

- 检查消息发布函数

  在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择device/+/data，检查发布消息的主题，然后选择 Publish to topic（发布到主题）。您应该会看到来自主题下方编辑框中的消息负载出现在消息列表中。

- 您未在 MQTT 客户端中看到您的重新发布的消息

  要使规则起作用，规则必须具有授权其接收和重新发布消息的正确策略，并且必须接收消息。

  要检查的事项

  - 检查您 AWS 区域 的 MQTT 客户端和您创建的规则

    您在其中运行 MQTT 客户端的控制台必须位于您所创建规则的同一 AWS 区域。

  - 检查规则查询语句中的输入消息主题

    要使规则起作用，它必须收到一条消息，消息主题名称应与规则查询语句 FROM 子句中的主题筛选条件相匹配。

    检查规则查询语句中主题筛选条件的拼写与 MQTT 客户端中主题的拼写。主题名称区分大小写，消息的主题必须与规则查询语句中的主题筛选条件匹配。

  - 检查输入消息负载的内容

    要使规则起作用，则必须在 SELECT 语句中声明的消息负载中找到数据字段。

    检查规则查询语句 `temperature`字段中的拼写与 MQTT 客户端中消息负载的拼写。字段名称区分大小写，规则查询语句中 `temperature`字段必须与消息负载中的 `temperature`字段相同。

    确保消息负载中的 JSON 文档格式正确。如果 JSON 有任何错误（例如缺少逗号），则规则将无法读取它。

  - 在规则操作中检查重新发布的消息主题

    重新发布规则操作向其发布新消息的主题必须与您在 MQTT 客户端中订阅的主题匹配。

    在控制台中打开您创建的规则，然后检查规则操作将重新发布消息的主题。

- 检查规则所使用的角色

  规则操作必须具有接收原始主题和发布新主题的权限。

  授权规则接收消息数据并重新发布的策略特定于所使用的主题。如果更改用于重新发布消息数据的主题，则必须更新规则操作的角色，以更新其策略来匹配当前主题。

  如果怀疑这里出现了问题，请编辑“重新发布”规则操作并创建新角色。规则操作创建的新角色将接收执行这些操作所需的授权。

步骤 3：查看结果和后续步骤

在本教程中：

- 您在规则查询语句中使用了一个简单的 SQL 查询和几个函数来生成新的 MQTT 消息。
- 您创建了重新发布该新消息的规则。
- 您使用 MQTT 客户端来测试您的 AWS IoT 规则。

后续步骤

使用此规则重新发布几条消息后，请尝试使用该规则进行实验，以了解更改教程的某些方面如何影响重新发布的消息。以下想法可以帮助您开始操作。

- 在输入消息的主题中更改 *device_id*，并观察重新发布的消息负载中的效果。
- 更改规则查询语句中选定的字段，并观察重新发布的消息负载中的效果。
- 尝试本系列中的下一个教程，了解如何 教程：发送 Amazon SNS 通知。

本教程中使用的“重新发布规则”操作也可以帮助您调试规则查询语句。例如，您可以将此操作添加到规则中，以查看其规则查询语句如何设置规则操作所使用的数据的格式。

教程：发送 Amazon SNS 通知

本教程演示如何创建一条 AWS IoT 规则，将 MQTT 消息数据发送到 Amazon SNS 主题，以便可以将其作为 SMS 短信发送。

在本教程中，您将创建一条规则，以在温度超过规则中设置的值时将消息数据从天气传感器发送到 Amazon SNS 主题的所有订阅者。当报告的温度超过规则设置的值时，规则会检测到，并创建一个新

的消息负载，其中仅包含设备 ID、报告的温度和超出的温度限制。规则将新消息负载作为 JSON 文档发送到 SNS 主题，该主题会通知 SNS 主题的所有订阅者。

您将在本教程中学到的内容：

- 如何创建和测试 Amazon SNS 通知
- 如何根据规则调用 Amazon SNS 通知 AWS IoT
- 如何在规则查询语句中使用简单的 SQL 查询和函数
- 如何使用 MQTT 客户端测试规则 AWS IoT

完成本教程需要大约 30 分钟。

在本教程中，您将：

- 步骤 1：创建 Amazon SNS 主题，:发送 SMS 文本消息
- 步骤 2：创建发送短信的 AWS IoT 规则
- 第 3 步：测试 AWS IoT 规则和 Amazon SNS 通知
- 步骤 4：查看结果和后续步骤

在开始本教程之前，请确保您具有：

- 设置你的 AWS 账户

  你需要你的 AWS 账户 和 AWS IoT 主机才能完成本教程。
- 审核 使用 MQTT 客户端查看 AWS IoT MQTT 消息

  请确保您可以使用 MQTT 客户端订阅和发布主题。您将使用 MQTT 客户端在此流程中测试新规则。
- 审核 Amazon Simple Notification Service

  如果您以前未使用过 Amazon SNS，请查看 设置 Amazon SNS 的访问权限。如果您已经完成了其他 AWS IoT 教程，则 AWS 账户 应该已经正确配置了您的教程。

步骤 1：创建 Amazon SNS 主题，:发送 SMS 文本消息

创建发送 SMS 文本消息的 Amazon SNS 主题

1. 创建 Amazon SNS 主题。

a. 登录 Amazon SNS 控制台。

b. 在左侧导航窗格中，选择主题。

c. 在 Topics（主题）页面上，选择 Create topic（创建主题）。

d. 在 Details（详细信息）中，选择 Standard（标准）类型。默认情况下，控制台会创建 FIFO 主题。

e. 在 Name（名称）中，输入 SNS 主题名称。在本教程中，请输入 **high_temp_notice**。

f. 滚动到页面底部，然后选择 Create topic（创建主题）。

控制台会打开新主题的详细信息页面。

2. 创建 Amazon SNS 订阅。

> **Note**
> 您将在本教程中发送的消息可能会使您在此订阅中使用的电话号码产生短信费用。

a. 在 high_temp_notice 主题详细信息页面上，选择 Create subscription（创建订阅）。

b. 在 Create subscripition（创建订阅），在 Details（详细信息）部分中，在 Protocol（协议）列表中，选择 SMS。

c. 在 Endpoint（端点）中，输入可接收文本消息的电话号码。请务必输入号码，以 +开头，包含国家和地区代码，并且不包括任何其它标点符号字符。

d. 选择 Create subscription（创建订阅）。

3. 测试 Amazon SNS 通知。

a. 在 Amazon SNS 控制台中，左侧导航窗格内，选择 Topics（主题）。

b. 要打开主题的详细信息页面，请在 Topics（主题），在主题列表中，选择 high_temp_notice。

c. 打开 Publish message to topic（将消息发布到主题）页面中的 high_temp_notice详细信息页面，选择 Publish message（发布消息）。

d. 在 Publish message to topic（将消息发布到主题），在 Message body（消息正文）部分，Message body to send to the endpoint（要发送到端点的消息正文）中，输入一条简短的消息。

e. 滚动到页面底部并选择 Publish message（发布消息）。

f. 在使用您之前创建订阅时使用的号码的电话上，确认已收到消息。

如果您没有收到测试消息，请仔细检查电话号码和手机的设置。

确保您可以从 Amazon SNS 控制台发布测试消息，然后继续本教程。

步骤 2：创建发送短信的 AWS IoT 规则

您将在本教程中创建的 AWS IoT 规则订阅了 device/*device_id*/data MQTT 主题，其中*device_id*是发送消息的设备的 ID。这些主题在主题筛选条件中描述为 device/+/data，其中，+ 是匹配两个正斜杠字符之间的任何字符串的通配符。此规则还测试消息负载中的temperature字段值。

当规则收到来自匹配主题的消息时，它会从主题名称中取 *device_id*，从消息负载中取temperature值，并为正在测试的限制添加一个常量值，并将这些值作为 JSON 文档发送到 Amazon SNS 通知主题。

例如，来自气象传感器设备编号 32 的 MQTT 消息使用 device/32/data主题，并具有如下所示的消息负载：

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

规则的规则查询语句从消息负载中取 temperature值，从主题名称中取 *device_id*，并添加常量max_temperature值以向 Amazon SNS 主题发送类似于以下内容的消息负载：

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30
}
```

创建用于检测超限温度值的 AWS IoT 规则并创建要发送至 Amazon SNS 主题的数据

1. 打开AWS IoT 控制台的 “规则” 中心。
2. 如果这是您的第一条规则，请选择 Create（创建），或者 Create a rule（创建规则）。
3. 在 Create a rule（创建规则）中：

   a. 在 Name（名称）中，输入 **temp_limit_notify**。

      请记住，规则名称在您的 AWS 账户 和区域内必须是唯一的，并且不能有任何空格。我们在此名称中使用了下划线字符来分隔规则名称中的单词。

   b. 在 Description（说明）中，描述规则。

      有意义的描述可以更容易地记住此规则的作用以及您创建它的原因。描述可以根据需要延长，因此请尽可能详细。

4. 在 Create a rule（创建规则）的 Rule query statement（规则查询语句）中：

   a. 在 Using SQL version（使用 SQL 版本）中，选择 2016-03-23。

   b. 在 Rule query statement（规则查询语句）编辑框中，输入语句：

```
SELECT topic(2) as device_id,
    temperature as reported_temperature,
    30 as max_temperature
  FROM 'device/+/data'
  WHERE temperature > 30
```

      本语句：

      - 侦听主题与 `device/+/data`主题筛选条件相符，并且 `temperature`值大于 30 的 MQTT 消息。
      - 从主题字符串中选择第二个元素，并将其分配给 `device_id`字段。
      - 从消息负载中选择 `temperature`字段的值，并将其分配给 `reported_temperature`字段。
      - 创建常数值 `30`来表示限制值，并将其赋值给 `max_temperature`字段。

5. 要打开此规则的规则操作列表，请在 Set one or more actions（设置一个或多个操作）中，选择 Add action（添加操作）。
6. 在 Select an action（选择操作）页面上，选择 Send a message as an SNS push notification（将消息发送为 SNS 推送通知）。

7. 要打开选定操作的配置页面，请在操作列表底部选择 Configure action（配置操作）。
8. 在 Configure action（配置操作）中：

   a. 在 SNS target（SNS target）中，选择 Select（选择），找到您名为 high_temp_notice 的 SNS 主题，然后选择 Select（选择）。

   b. 对于 Message format（消息格式），请选择 RAW。

   c. 在选择或创建角色以授予执行此操作的 AWS IoT 访问权限中，选择创建角色。

   d. 在 Create a new role（创建新角色）中，Name（名称）项下，输入新角色的唯一名称。在本教程中，请使用 **`sns_rule_role`**。

   e. 选择 Create role（创建角色）。

   如果要重复本教程或重复使用现有角色，请选择 Update role（更新角色）然后再继续。这将更新角色的策略文档，以便与 SNS 目标配合使用。
9. 选择 Add action（添加操作），然后返回到 Create a rule（创建规则）页。

   在新操作的磁贴中，在 Send a message as an SNS push notification（将消息作为 SNS 推送通知发送）项下，您可以看到规则将调用的 SNS 主题。

   这是您将添加到此规则的唯一规则操作。
10. 要创建规则并完成此步骤，请在 Create a rule（创建规则）中，向下滚动到底部，然后选择 Create rule（创建规则）。

第 3 步：测试 AWS IoT 规则和 Amazon SNS 通知

要测试新规则，您将使用 MQTT 客户端发布和订阅此规则所使用的 MQTT 消息。

在新窗口中打开 AWS IoT 控制台中的 MQTT 客户端。这将允许您在不丢失 MQTT 客户端配置的情况下编辑规则。如果您让 MQTT 客户端转到控制台中的其它页面，它将不会保留任何订阅或消息日志。

如需使用 MQTT 客户端来测试您的规则

1. 在 AWS IoT 控制台中的 MQTT 客户端中，订阅输入主题，在本例中为 `device/+/data`。

   a. 在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择 Subscribe to a topic（订阅主题）。

   b. 在 Subscription topic（订阅主题）中，输入输入主题筛选条件的主题，**`device/+/data`**。

   c. 将其它字段保留为默认设置。

d. 选择 Subscribe to topic（订阅主题）。

在 Subscriptions（订阅）栏，在 Publish to a topic（发布到主题）项下，将会显示 **device/+/data**。

2. 使用特定设备 ID **device/32/data**向输入主题发布消息。您无法发布到包含通配符的 MQTT 主题。

a. 在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择 Subscribe to a topic（订阅主题）。

b. 在 Publish（发布）字段中，输入输入主题名称，**device/32/data**。

c. 复制此处显示的示例数据，然后在主题名称下方的编辑框中粘贴示例数据。

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. 选择 Publish to topic（发布到主题）以发布您的 MQTT 消息。

3. 确认文本消息已发送。

a. 在 MQTT 客户端，Subscriptions（订阅）项下，您之前订阅的主题旁边会有一个绿色圆点。

绿色圆点表示自上次查看消息以来已收到一条或多条新消息。

b. 在 Subscriptions（订阅）中，选择 device/+/data 来检查消息负载是否与刚刚发布的内容匹配，如下所示：

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
```

```
}
```

c. 检查您用于订阅 SNS 主题的电话，并确认消息负载的内容，如下所示：

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

请注意，device_id 值是一个带引号的字符串，temperature 值为数字。这是因为 topic()函数从输入消息的主题名称中提取字符串，而 temperature值使用输入消息负载中的数值。

如果您想使 device_id值为数值，请在规则查询语句中将 topic(2)替换为：

```
cast(topic(2) AS DECIMAL)
```

请注意，将 topic(2)值设置为数字，DECIMAL 值仅在主题的该部分仅包含数字字符时才起作用。

4. 尝试发送温度未超过限制的 MQTT 消息。

a. 在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择 Subscribe to a topic（订阅主题）。

b. 在 Publish（发布）字段中，输入输入主题名称，**device/33/data**。

c. 复制此处显示的示例数据，然后在主题名称下方的编辑框中粘贴示例数据。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. 要发送您的 MQTT 消息，请选择 Publish to topic（发布到主题）。

您应该会看到您在 **device/+/data**订阅中发送的消息。但是，由于温度值低于规则查询语句中的最大温度，因此您应无法收到文本消息。

如果您未看到正确的行为，请检查故障排除提示。

对 SNS 消息规则进行故障排除

如果您没有看到期望的结果，可以检查以下事项。

- 您收到了一个错误的广告条

  如果在您发布输入消息时出现错误，请先更正该错误。以下步骤可帮助您更正此错误。

- 您未在 MQTT 客户端中看到输入消息

  每次将输入消息发布到 `device/22/data`主题时，如果您如流程中所述订阅了 `device/+/data`主题筛选条件，则消息应当会在 MQTT 客户端中显示。

  要检查的事项

  - 检查您订阅的主题筛选条件

    如果您如流程中所述订阅了输入消息主题，则每次发布输入消息时都应看到该输入消息的副本。

    如果没有看到该消息，请检查您订阅的主题名称，并将其与您发布的主题进行比较。主题名称区分大小写，您订阅的主题必须与发布消息负载的主题相同。

  - 检查消息发布函数

    在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择device/+/data，检查发布消息的主题，然后选择 Publish to topic（发布到主题）。您应该会看到来自主题下方编辑框中的消息负载出现在消息列表中。

- 您没有收到 SMS 消息

  要使规则起作用，规则必须具有授权其接收消息和发送 SNS 通知的正确策略，并且必须接收消息。

  要检查的事项

  - 检查您 AWS 区域 的 MQTT 客户端和您创建的规则

    您在其中运行 MQTT 客户端的控制台必须位于您所创建规则的同一 AWS 区域。

  - 检查消息负载中的温度值是否超过测试阈值

    如果温度值小于或等于 30（如规则查询语句中定义），则规则将不会执行其任何操作。

  - 检查规则查询语句中的输入消息主题

    要使规则起作用，它必须收到一条消息，消息主题名称应与规则查询语句 FROM 子句中的主题筛选条件相匹配。

检查规则查询语句中主题筛选条件的拼写与 MQTT 客户端中主题的拼写。主题名称区分大小写，消息的主题必须与规则查询语句中的主题筛选条件匹配。

- 检查输入消息负载的内容

  要使规则起作用，则必须在 SELECT 语句中声明的消息负载中找到数据字段。

  检查规则查询语句 `temperature`字段中的拼写与 MQTT 客户端中消息负载的拼写。字段名称区分大小写，规则查询语句中 `temperature`字段必须与消息负载中的 `temperature`字段相同。

  确保消息负载中的 JSON 文档格式正确。如果 JSON 有任何错误（例如缺少逗号），则规则将无法读取它。

- 在规则操作中检查重新发布的消息主题

  重新发布规则操作向其发布新消息的主题必须与您在 MQTT 客户端中订阅的主题匹配。

  在控制台中打开您创建的规则，然后检查规则操作将重新发布消息的主题。

- 检查规则所使用的角色

  规则操作必须具有接收原始主题和发布新主题的权限。

  授权规则接收消息数据并重新发布的策略特定于所使用的主题。如果更改用于重新发布消息数据的主题，则必须更新规则操作的角色，以更新其策略来匹配当前主题。

  如果怀疑这里出现了问题，请编辑“重新发布”规则操作并创建新角色。规则操作创建的新角色将接收执行这些操作所需的授权。

步骤 4：查看结果和后续步骤

在本教程中：

- 您创建并测试了 Amazon SNS 通知主题和订阅。
- 您在规则查询语句中使用了简单 SQL 查询和函数来为您的通知创建新消息。
- 您创建了使用您的自定义消息负载发送 Amazon SNS 通知的 AWS IoT 规则。
- 您使用 MQTT 客户端来测试您的 AWS IoT 规则。

后续步骤

使用此规则发送了几条文本消息后，请尝试使用它来了解更改教程的某些方面如何影响消息以及发送消息的时间。以下想法可以帮助您开始操作。

- 在输入消息主题中更改 *device_id*，并观察文本消息内容中的效果。
- 更改规则查询语句中选定的字段，并观察文本消息内容中的效果。
- 将规则查询语句中的测试更改为测试最低温度而不是最高温度。记得更改 max_temperature名称！
- 添加重新发布规则操作，以便在发送 SNS 通知时发送 MQTT 消息。
- 尝试本系列中的下一个教程，了解如何 教程：将设备数据存储在 DynamoDB 表中。

教程：将设备数据存储在 DynamoDB 表中

本教程演示如何创建将消息数据发送到 DynamoDB 表的 AWS IoT 规则。

在本教程中，您将创建一条规则来将消息数据从假想气象传感器设备发送到 DynamoDB 表。该规则将为来自许多气象传感器的数据设置格式，以便将其添加到单个数据库表中。

在本教程中您将学到

- 如何创建 DynamoDB 表
- 如何通过规则向 DynamoDB 表发送消息数据 AWS IoT
- 如何在 AWS IoT 规则中使用替换模板
- 如何在规则查询语句中使用简单的 SQL 查询和函数
- 如何使用 MQTT 客户端测试规则 AWS IoT

完成本教程需要大约 30 分钟。

在本教程中，您将：

- 步骤 1：为本教程创建 DynamoDB 表
- 步骤 2：创建向 DynamoDB 表发送数据的 AWS IoT 规则
- 步骤 3：测试 AWS IoT 规则和 DynamoDB 表
- 步骤 4：查看结果和后续步骤

在开始本教程之前，请确保您具有：

- 设置你的 AWS 账户

你需要你的 AWS 账户 和 AWS IoT 主机才能完成本教程。

- 审核 使用 MQTT 客户端查看 AWS IoT MQTT 消息

  请确保您可以使用 MQTT 客户端订阅和发布主题。您将使用 MQTT 客户端在此流程中测试新规则。

- 审核 Amazon DynamoDB 概览

  如果您之前没有使用过 DynamoDB，请查看 DynamoDB 入门来熟悉 DynamoDB 的基本概念和操作。

步骤 1：为本教程创建 DynamoDB 表

在本教程中，您将创建一个具有以下属性的 DynamoDB 表，以记录来自虚拟气象传感器设备的数据：

- `sample_time` 是主键，描述了记录样本的时间。
- `device_id` 是排序键，描述了提供示例的设备
- `device_data` 是从设备接收并由规则查询语句格式化的数据

如需为本教程创建 DynamoDB 表

1. 打开 DynamoDB 控制台，然后选择 Create table（创建表格）。
2. 在 Create table（创建表）中：
   a. 在 Table name（表名称）框中输入表名：**`wx_data`**。
   b. 在 Partition key（分区键）中，输入 **`sample_time`**，然后在字段旁边的选项列表中，选择 **`Number`**。
   c. 在 Sort key（排序键）中，输入 **`device_id`**，然后在字段旁边的选项列表中，选择 **`Number`**。
   d. 在页面底部，选择 Create（创建）。

您将在稍后定义 `device_data`，即您配置 DynamoDB 规则操作时。

步骤 2：创建向 DynamoDB 表发送数据的 AWS IoT 规则

在此步骤中，您将使用规则查询语句格式化来自虚拟气象传感器设备的数据，以便写入数据库表。

从天气传感器设备接收的示例消息负载如下所示：

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

对于数据库条目，您将使用规则查询语句将消息负载的结构展平如下：

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind_velocity": 22,
  "wind_bearing": 255
}
```

在这个规则中，您还将使用几个 替换模板。替代模板是用于从函数和消息数据中插入动态值的表达式。

创建向 DynamoDB 表发送数据的 AWS IoT 规则

1. 打开 AWS IoT 控制台的规则中心。或者，您可以在中打开 AWS IoT 主页 AWS Management Console 并导航到 “消息路由” > “规则”。
2. 要开始在 Rules（规则）中创建新规则，请选择 Create rule（创建规则）。
3. 在 Rule properties（规则属性）中：

   a. 在 Rule name（规则名称）中，输入 **wx_data_ddb**。

      请记住，规则名称在您的 AWS 账户 和区域内必须是唯一的，并且不能有任何空格。我们在此名称中使用了下划线字符来分隔规则名称中的两个单词。

   b. 在 Rule description（规则描述）中，描述规则。

      有意义的描述可以更容易地记住此规则的作用以及您创建它的原因。描述可以根据需要延长，因此请尽可能详细。

4. 选择下一步以继续。

5. 在 SQL statement（SQL 语句）中：

   a. 在 SQL version（SQL 版本）中，选择 **2016-03-23**。

   b. 在 SQL statement（SQL 语句）编辑框中，输入语句：

```
SELECT temperature, humidity, barometer,
  wind.velocity as wind_velocity,
  wind.bearing as wind_bearing,
FROM 'device/+/data'
```

   本语句：

   - 侦听主题符合 `device/+/data`主题筛选条件的 MQTT 消息。
   - 将 `wind`属性的元素格式化为单个属性。
   - 传递 `temperature`、`humidity`、`barometer` 属性，使其保持不变。

6. 选择下一步以继续。

7. 在 Rule actions（规则操作）中：

   a. 要打开此规则的规则操作列表，请在 Action 1（操作 1）中，选择 **DynamoDB**。

   > **Note**
   >
   > 确保选择 DynamoDB 而不是 DynamoDBv2 作为规则操作。

   b. 在 Table name（表名称）中，选择在上一步中创建的 DynamoDB 表的名称：**wx_data**。

   Partition key type（分区键类型）和 Sort key type（排序键类型）字段将使用 DynamoDB 表中的值填充。

   c. 在分区键中，输入 **sample_time**。

   d. 在分区键值中，输入 **${timestamp()}**。

   这是您将在此规则中使用第一个 替换模板。它将使用 timestamp 函数返回的值，而不是消息负载中的值。要了解更多信息，请参阅《AWS IoT Core 开发人员指南》中的时间戳。

   e. 在 Sort key（排序键）中，输入 **device_id**。

   f. 在排序键值中，输入 **${cast(topic(2) AS DECIMAL)}**。

   这是您将在此规则中使用的第二个 替换模板。它将第二个元素的值插入到主题名称，即设备的 ID，后面其将投射为十进制值以匹配密钥的数字格式。要了解有关主题的更多信息，请参

阅《AWS IoT Core 开发人员指南》中的主题。或者，要了解有关强制转换的更多信息，请参阅《AWS IoT Core 开发人员指南》中的强制转换。

g. 在 Write message data to this column (将消息数据写入到此列) 中，输入 **device_data**。

这将在 DynamoDB 表中创建 device_data列。

h. 将 Operation (操作) 留空。

i. 在 IAM Role（IAM 角色）中，选择 Create new role（创建新角色）。

j. 在 Create role（创建角色）对话框中，为 Role name（角色名称）输入 wx_ddb_role。这个新角色将自动包含前缀为 “aws-iot-rule” 的策略，该策略将允许该**wx_data_ddb**规则将数据发送到您创建的 D **wx_data** ynamoDB 表。

k. 在 IAM role（IAM 角色）中，选择 **wx_ddb_role**。

l. 在页面底部，选择 Next。

8. 在 Review and create（审核和创建）页面的底部，选择 Create（创建）以创建规则。

步骤 3：测试 AWS IoT 规则和 DynamoDB 表

要测试新规则，您将使用 MQTT 客户端发布和订阅此测试中使用的 MQTT 消息。

在新窗口中打开 AWS IoT 控制台中的 MQTT 客户端。这将允许您在不丢失 MQTT 客户端配置的情况下编辑规则。如果您让其转到控制台中的其它页面，MQTT 客户端不会保留任何订阅或消息日志。您还需要在控制台的 D ynamoDB 表中心打开一个单独的控制台窗口 AWS IoT ，以查看您的规则发送的新条目。

如需使用 MQTT 客户端来测试您的规则

1. 在 AWS IoT 控制台中的 MQTT 客户端中，订阅输入主题，device/+/data。

a. 在 MQTT 客户端中，选择 Subscribe to a topic（订阅主题）。

b. 对于 Topic filter（主题筛选条件）中，输入输入主题筛选条件的主题，**device/+/data**。

c. 选择订阅。

2. 现在，使用特定设备 ID **device/22/data**向输入主题发布消息。您无法发布到包含通配符的 MQTT 主题。

a. 在 MQTT 客户端中，选择 Publish to a topic（发布到主题）。

b. 对于 Topic name（主题名称），输入输入主题名称 **device/22/data**。

c. 对于 Message payload（消息负载），输入以下示例数据。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. 要发布 MQTT 消息，请选择 Publish（发布）。

e. 现在，在 MQTT 客户端中，选择 Subscribe to a topic（订阅主题）。在 Subscribe（订阅）列中，选择 **device/+/data**订阅。确认上一步骤中的示例数据是否显示在此处。

3. 检查以查看您的规则创建的 DynamoDB 表中的行。

a. 在控制台的 D ynamoDB 表中心中 AWS IoT，选择 wx_data，然后选择项目选项卡。

如果您已经在 Items（项目）选项卡上，则可能需要通过选择表格标题右上角的刷新图标来刷新显示。

b. 请注意，表中的 sample_time 值是链接，请打开一个。如果您刚刚发送了第一条消息，那么这将是列表中唯一的消息。

此链接显示表的该行中所有的数据。

c. 展开 device_data 条目以查看规则查询语句生成的数据。

d. 浏览此显示中可用数据的不同表示形式。您也可以编辑此显示中的数据。

e. 查看完此行数据后，要保存所做的任何更改，请选择 Save（保存），或者要退出而不保存任何更改，请选择 Cancel（取消）。

如果您未看到正确的行为，请检查故障排除提示。

故障排除 DynamoDB 规则

如果您没有看到期望的结果，可检查以下事项。

- 您收到了一个错误的广告条

  如果在您发布输入消息时出现错误，请先更正该错误。以下步骤可帮助您更正此错误。

- 您未在 MQTT 客户端中看到输入消息

每次将输入消息发布到 device/22/data主题时，如果您如流程中所述订阅了 device/+/data主题筛选条件，则消息应当会在 MQTT 客户端中显示。

要检查的事项

- 检查您订阅的主题筛选条件

  如果您如流程中所述订阅了输入消息主题，则每次发布输入消息时都应看到该输入消息的副本。

  如果没有看到该消息，请检查您订阅的主题名称，并将其与您发布的主题进行比较。主题名称区分大小写，您订阅的主题必须与发布消息负载的主题相同。

- 检查消息发布函数

  在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择device/+/data，检查发布消息的主题，然后选择 Publish to topic（发布到主题）。您应该会看到来自主题下方编辑框中的消息负载出现在消息列表中。

- 您在 DynamoDB 表中看不到您的数据

  要做的第一件事是通过选择表格标题右上角的刷新图标来刷新显示。如果没有显示您要查找的数据，请检查以下内容。

  要检查的事项

  - 检查您 AWS 区域 的 MQTT 客户端和您创建的规则

    您在其中运行 MQTT 客户端的控制台必须位于您所创建规则的同一 AWS 区域。

  - 检查规则查询语句中的输入消息主题

    要使规则起作用，它必须收到一条消息，消息主题名称应与规则查询语句 FROM 子句中的主题筛选条件相匹配。

    检查规则查询语句中主题筛选条件的拼写与 MQTT 客户端中主题的拼写。主题名称区分大小写，消息的主题必须与规则查询语句中的主题筛选条件匹配。

  - 检查输入消息负载的内容

    要使规则起作用，则必须在 SELECT 语句中声明的消息负载中找到数据字段。

    检查规则查询语句 temperature字段中的拼写与 MQTT 客户端中消息负载的拼写。字段名称区分大小写，规则查询语句中 temperature字段必须与消息负载中的 temperature字段相同。

确保消息负载中的 JSON 文档格式正确。如果 JSON 有任何错误（例如缺少逗号），则规则将无法读取它。

- 检查规则操作中使用的键名和字段名称

  主题规则中使用的字段名称必须与已发布消息的 JSON 消息负载中找到的字段名称匹配。

  打开在控制台中创建的规则，并检查规则操作配置中的字段名称与 MQTT 客户端中使用的字段名称。

- 检查规则所使用的角色

  规则操作必须具有接收原始主题和发布新主题的权限。

  授权规则接收消息数据和更新 DynamoDB 表的策略特定于所使用的主题。如果更改规则使用的主题或 DynamoDB 表名称，则必须更新规则操作的角色以更新其策略来匹配。

  如果您怀疑存在问题，请编辑规则操作并创建新角色。规则操作创建的新角色将接收执行这些操作所需的授权。

步骤 4：查看结果和后续步骤

使用此规则向 DynamoDB 表发送几条消息后，请尝试对其进行试验，以了解更改教程中的某些方面如何影响写入表的数据。以下想法可以帮助您开始操作。

- 在输入消息主题中更改 *device_id*，并观察对数据的影响。您可以使用它来模拟从多个气象传感器接收数据。
- 更改规则查询语句中选定的字段，并观察对数据的影响。您可以使用它筛选存储在表中的数据。
- 添加重新发布规则操作，以便为添加到表中的每一行发送 MQTT 消息。您可以使用它进行调试。

完成本教程后，检查 the section called “使用 AWS Lambda 函数格式化通知”。

教程：使用 AWS Lambda 函数格式化通知

本教程演示如何将 MQTT 消息数据发送到 AWS Lambda 操作以进行格式化并发送到其他 AWS 服务。在本教程中， AWS Lambda 操作使用 AWS 软件开发工具包将格式化的消息发送到您在教程中创建的有关如何操作的 Amazon SNS 主题。the section called “发送 Amazon SNS 通知”

在有关如何 the section called “发送 Amazon SNS 通知”的教程中，则由规则的查询语句生成的 JSON 文档将作为文本消息的正文发送。结果是一条文本消息，示例如下：

```
{"device_id":"32","reported_temperature":38,"max_temperature":30}
```

在本教程中，您将使用 AWS Lambda 规则操作来调用一个 AWS Lambda 函数，该函数将规则查询语句中的数据格式化为更友好的格式，例如以下示例：

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

您将在本教程中创建的 AWS Lambda 函数使用规则查询语句中的数据格式化消息字符串，并调用 S AWS DK 的 SNS 发布函数来创建通知。

在本教程中您将学到

- 如何创建和测试 AWS Lambda 函数
- 如何在 AWS Lambda 函数中使用 AWS 软件开发工具包发布 Amazon SNS 通知
- 如何在规则查询语句中使用简单的 SQL 查询和函数
- 如何使用 MQTT 客户端测试规则 AWS IoT

完成本教程需要大约 45 分钟。

在本教程中，您将：

- 步骤 1：创建发送短信的 AWS Lambda 函数
- 步骤 2：使用 AWS IoT 规则操作创建 AWS Lambda 规则
- 步骤 3：测试 AWS IoT 规则和 AWS Lambda 规则操作
- 步骤 4：查看结果和后续步骤

在开始本教程之前，请确保您具有：

- 设置你的 AWS 账户

  你需要你的 AWS 账户 和 AWS IoT 主机才能完成本教程。
- 审核 使用 MQTT 客户端查看 AWS IoT MQTT 消息

  请确保您可以使用 MQTT 客户端订阅和发布主题。您将使用 MQTT 客户端在此流程中测试新规则。
- 完成本部分中的其它规则教程

本教程需要您在如何 the section called “发送 Amazon SNS 通知”的教程中创建的 SNS 通知主题。本教程还假定您已完成本部分中其它与规则相关的教程。

- 审核 AWS Lambda概览

  如果您 AWS Lambda 以前从未使用过，请查看AWS Lambda并开始使用 Lambda，以了解其术语和概念。

步骤 1：创建发送短信的 AWS Lambda 函数

本教程中的 AWS Lambda 函数接收规则查询语句的结果，将元素插入文本字符串，然后将生成的字符串作为通知中的消息发送到 Amazon SNS。

与使用 AWS IoT 规则操作发送通知的操作方法教程不同，本教程使用软件开发工具包的函数从 Lambda 函数发送通知。the section called “发送 Amazon SNS 通知” AWS 但是，本教程中使用的实际 Amazon SNS 通知主题与您在有关如何the section called “发送 Amazon SNS 通知”的教程中使用的主题相同。

创建发送短信的 AWS Lambda 函数

1. 创建新 AWS Lambda 函数。

   a. 在 AWS Lambda 控制台中，选择 Create function（创建函数）。

   b. 在 Create function（创建函数）中，选择 Use a blueprint（使用蓝图）。

      搜索并选择 **hello-world-python**蓝图，然后选择 Configure（配置）。

   c. 在 Basic information（基本信息）中：

      i. 在 Function name（函数名称）中，输入函数的名称，**format-high-temp-notification**。

      ii. 在执行角色中，选择从 AWS 策略模板创建新角色。

      iii. 在 Role name（角色名称）中，为新角色输入名称，**format-high-temp-notification-role**。

      iv. 在 Policy templates - optional（策略模板 — 可选）中，搜索并选择 Amazon SNS publish popolicy（Amazon SNS 发布策略）。

      v. 选择 Create function（创建函数）。

2. 修改蓝图代码以格式化并发送 Amazon SNS 通知。

a. 创建函数后，您应该会看到format-high-temp-notification详细信息页面。如果您不这样做，请从 Lambda Functions（Lambda 函数）页打开它。

b. 在format-high-temp-notification详细信息页面中，选择配置选项卡，然后滚动到函数代码面板。

c. 在 Function code（函数代码）窗口中的 Environment（环境）窗格中，选择 Python 文件，lambda_function.py。

d. 在 Function code（函数代码）窗口中，从蓝图中删除所有原始程序代码，并将其替换为此代码。

```
import boto3
#
#   expects event parameter to contain:
#   {
#       "device_id": "32",
#       "reported_temperature": 38,
#       "max_temperature": 30,
#       "notify_topic_arn": "arn:aws:sns:us-
east-1:57EXAMPLE833:high_temp_notice"
#   }
#
#   sends a plain text string to be used in a text message
#
#      "Device {0} reports a temperature of {1}, which exceeds the limit of
 {2}."
#
#   where:
#       {0} is the device_id value
#       {1} is the reported_temperature value
#       {2} is the max_temperature value
#
def lambda_handler(event, context):

    # Create an SNS client to send notification
    sns = boto3.client('sns')

    # Format text message from data
    message_text = "Device {0} reports a temperature of {1}, which exceeds the
 limit of {2}.".format(
            str(event['device_id']),
            str(event['reported_temperature']),
            str(event['max_temperature'])
```

```
        )

    # Publish the formatted message
    response = sns.publish(
            TopicArn = event['notify_topic_arn'],
            Message = message_text
        )

    return response
```

e. 选择 Deploy（部署）。

3. 在新窗口中，从关于如何the section called “发送 Amazon SNS 通知”的教程中查找 Amazon SNS 主题的 Amazon Resource Name (ARN)。

   a. 在新窗口中，打开 Topics page of the Amazon SNS console（Amazon SNS 控制台主题页面）。

   b. 在 Topics（主题）页面上，在 Amazon SNS 主题列表中找到 high_temp_notice通知主题。

   c. 查找 high_temp_notice 通知主题的 ARN，以在下一步中使用。

4. 为您的 Lambda 函数创建一个测试用例。

   a. 在控制台的 Lambda F un ctions 页面的format-high-temp-notification详细信息页面上，选择页面右上角的选择测试事件（尽管它看起来已禁用），然后选择配置测试事件。

   b. 在 Configure test event（配置测试事件）中，选择 Create new test event（创建新测试事件）。

   c. 在 Event name（事件名称）中，输入 **`SampleRuleOutput`**。

   d. 在 Event name（事件名称）下方的 JSON 编辑器中，请粘贴此示例 JSON 文档。这是您的 AWS IoT 规则将向 Lambda 函数发送的内容的示例。

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30,
  "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
}
```

   e. 请参阅具有 high_temp_notice通知主题的 ARN 的窗口并复制 ARN 值。

   f. 在 JSON 编辑器中将 `notify_topic_arn`值替换成您通知主题中的 ARN。

   保持此窗口打开状态，以便在创建 AWS IoT 规则时再次使用该 ARN 值。

g. 选择 Choose（创建）。

5. 使用示例数据测试函数。

a. 在页面右上角的format-high-temp-notification详细信息页面中，确认SampleRuleOutput显示在 “测试” 按钮旁边。如果没有，请从可用测试事件列表中选择它。

b. 要将示例规则输出消息发送到您的函数，请选择 Test（测试）。

如果函数和通知都有效，您将在订阅通知的手机上收到一条文本消息。

如果您在手机上没有收到文本消息，请检查操作结果。在 Function code（函数代码）面板中的 Execution result（执行结果）选项卡上，查看响应以查找发生的任何错误。请勿继续执行下一步，直到您的函数可以将通知发送到您的手机。

步骤 2：使用 AWS IoT 规则操作创建 AWS Lambda 规则

在此步骤中，您将使用规则查询语句格式化来自虚拟气象传感器设备的数据，以发送到 Lambda 函数，该函数将格式化并发送文本消息。

从天气设备接收的示例消息负载如下所示：

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

在此规则中，您将使用规则查询语句为 Lambda 函数创建消息负载，如下所示：

```
{
  "device_id": "32",
  "reported_temperature": 38,
  "max_temperature": 30,
  "notify_topic_arn": "arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice"
}
```

这包含 Lambda 函数格式化和发送正确文本消息所需的所有信息。

创建调用 Lambda 函数的 AWS IoT 规则

1. 打开AWS IoT 控制台的 “规则” 中心。
2. 要在 Rule（规则）中创建新规则，请选择 Create（创建）。
3. 在 Create a rule（创建规则）顶部：

   a. 在 Name（名称）中，输入规则的名称，**wx_friendly_text**。

      请记住，规则名称在您的 AWS 账户 和区域内必须是唯一的，并且不能有任何空格。我们在此名称中使用了下划线字符来分隔规则名称中的两个单词。

   b. 在 Description（说明）中，描述规则。

      有意义的描述可以更容易地记住此规则的作用以及您创建它的原因。描述可以根据需要延长，因此请尽可能详细。

4. 在 Create a rule（创建规则）的 Rule query statement（规则查询语句）中：

   a. 在 Using SQL version（使用 SQL 版本）中，选择 **2016-03-23**。

   b. 在 Rule query statement（规则查询语句）编辑框中，输入语句：

```
SELECT
  cast(topic(2) AS DECIMAL) as device_id,
  temperature as reported_temperature,
  30 as max_temperature,
  'arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice' as notify_topic_arn
FROM 'device/+/data' WHERE temperature > 30
```

      本语句：

      - 侦听主题与 device/+/data主题筛选条件相符，并且 temperature值大于 30 的 MQTT 消息。
      - 从主题字符串中选择第二个元素，将其转换为十进制数，然后将其分配给 device_id字段。
      - 从消息负载中选择 temperature字段的值，并将其分配给 reported_temperature字段。
      - 创建常数值，30，以表示限制值，并将其分配给 max_temperature字段。
      - 为 notify_topic_arn字段创建常数值。

   c. 请参阅具有 high_temp_notice 通知主题的 ARN 的窗口并复制 ARN 值。

d. 使用通知主题的 ARN 替换 ARN 值 (*arn:aws:sns:us-east-1:57EXAMPLE833:high_temp_notice*)。

5. 在 Set one or more actions（设置一个或多个操作）中：

   a. 要打开此规则的规则操作列表，请选择 Add action（添加操作）。

   b. 在 Select an action（选择操作）中，选择 Send a message to a Lambda function（向 Lambda 函数发送消息）。

   c. 要打开选定操作的配置页面，请在操作列表底部选择 Configure action（配置操作）。

6. 在 Configure action（配置操作）中：

   a. 在 Function name（函数名称）中，选择 Select（选择）。

   b. 选择format-high-temp-notification。

   c. 在 Configure action（配置操作）底部，选择 Add action（添加操作）。

   d. 要创建规则，请在 Create a rule（创建一条规则）底部，选择 Create rule（创建规则）。

步骤 3：测试 AWS IoT 规则和 AWS Lambda 规则操作

要测试新规则，您将使用 MQTT 客户端发布和订阅此规则所使用的 MQTT 消息。

在新窗口中打开 AWS IoT 控制台中的 MQTT 客户端。现在，您可以在不丢失 MQTT 客户端配置的情况下编辑规则。如果您离开 MQTT 客户端并转到控制台中的其它页面，则将丢失您的订阅或消息日志。

如需使用 MQTT 客户端来测试您的规则

1. 在 AWS IoT 控制台中的 MQTT 客户端中，订阅输入主题，在本例中为 `device/+/data`。

   a. 在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择 Subscribe to a topic（订阅主题）。

   b. 在 Subscription topic（订阅主题）中，输入输入主题筛选条件的主题，**`device/+/data`**。

   c. 将其它字段保留为默认设置。

   d. 选择 Subscribe to topic（订阅主题）。

      在 Subscriptions（订阅）栏，在 Publish to a topic（发布到主题）项下，将会显示 **`device/+/data`**。

2. 使用特定设备 ID **`device/32/data`**向输入主题发布消息。您无法发布到包含通配符的 MQTT 主题。

a. 在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择 Subscribe to a topic（订阅主题）。

b. 在 Publish（发布）字段中，输入输入主题名称，**device/32/data**。

c. 复制此处显示的示例数据，然后在主题名称下方的编辑框中粘贴示例数据。

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

d. 如需发布 MQTT 消息，请选择 Publish to topic（发布到主题）。

3. 确认文本消息已发送。

a. 在 MQTT 客户端，Subscriptions（订阅）项下，您之前订阅的主题旁边会有一个绿色圆点。

绿色圆点表示自上次查看消息以来已收到一条或多条新消息。

b. 在 Subscriptions（订阅）中，选择 device/+/data 来检查消息负载是否与刚刚发布的内容匹配，如下所示：

```
{
  "temperature": 38,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

c. 检查您用于订阅 SNS 主题的电话，并确认消息负载的内容，如下所示：

```
Device 32 reports a temperature of 38, which exceeds the limit of 30.
```

如果您更改消息主题中的主题 ID 元素，请记住，将 `topic(2)`值投射为数字值仅在消息主题中的元素仅包含数字字符时才起作用。

4. 尝试发送温度未超过限制的 MQTT 消息。

   a. 在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择 Subscribe to a topic（订阅主题）。

   b. 在 Publish（发布）字段中，输入输入主题名称，**`device/33/data`**。

   c. 复制此处显示的示例数据，然后在主题名称下方的编辑框中粘贴示例数据。

```
{
  "temperature": 28,
  "humidity": 80,
  "barometer": 1013,
  "wind": {
    "velocity": 22,
    "bearing": 255
  }
}
```

   d. 要发送您的 MQTT 消息，请选择 Publish to topic（发布到主题）。

   您应在 **`device/+/data`**订阅中看到您发送的消息；但是，由于温度值低于规则查询语句中的最大温度，因此您不应收到文本消息。

   如果您未看到正确的行为，请检查故障排除提示。

对您的 AWS Lambda 规则和通知进行故障排除

如果您没有看到期望的结果，可以检查以下事项。

- 您收到了一个错误的广告条

  如果在您发布输入消息时出现错误，请先更正该错误。以下步骤可帮助您更正此错误。

- 您未在 MQTT 客户端中看到输入消息

  每次将输入消息发布到 `device/32/data`主题时，如果您如流程中所述订阅了 `device/+/data`主题筛选条件，则消息应当会在 MQTT 客户端中显示。

要检查的事项

- 检查您订阅的主题筛选条件

  如果您如流程中所述订阅了输入消息主题，则每次发布输入消息时都应看到该输入消息的副本。

  如果没有看到该消息，请检查您订阅的主题名称，并将其与您发布的主题进行比较。主题名称区分大小写，您订阅的主题必须与发布消息负载的主题相同。

- 检查消息发布函数

  在 MQTT 客户端中，在 Subscriptions（订阅）项下，选择device/+/data，检查发布消息的主题，然后选择 Publish to topic（发布到主题）。您应该会看到来自主题下方编辑框中的消息负载出现在消息列表中。

- 您没有收到 SMS 消息

  要使规则起作用，规则必须具有授权其接收消息和发送 SNS 通知的正确策略，并且必须接收消息。

  要检查的事项

  - 检查您 AWS 区域 的 MQTT 客户端和您创建的规则

    您在其中运行 MQTT 客户端的控制台必须位于您所创建规则的同一 AWS 区域。

  - 检查消息负载中的温度值是否超过测试阈值

    如果温度值小于或等于 30（如规则查询语句中定义），则规则将不会执行其任何操作。

  - 检查规则查询语句中的输入消息主题

    要使规则起作用，它必须收到一条消息，消息主题名称应与规则查询语句 FROM 子句中的主题筛选条件相匹配。

    检查规则查询语句中主题筛选条件的拼写与 MQTT 客户端中主题的拼写。主题名称区分大小写，消息的主题必须与规则查询语句中的主题筛选条件匹配。

  - 检查输入消息负载的内容

    要使规则起作用，则必须在 SELECT 语句中声明的消息负载中找到数据字段。

    检查规则查询语句 `temperature`字段中的拼写与 MQTT 客户端中消息负载的拼写。字段名称区分大小写，规则查询语句中 `temperature`字段必须与消息负载中的 `temperature`字段相同。

确保消息负载中的 JSON 文档格式正确。如果 JSON 有任何错误（例如缺少逗号），则规则将无法读取它。

- 查看 Amazon SNS 通知

  在 步骤 1：创建 Amazon SNS 主题，:发送 SMS 文本消息中，请参阅步骤 3，该步骤介绍如何测试 Amazon SNS 通知并测试通知以确保通知有效。

- 检查 Lambda 函数

  在 步骤 1：创建发送短信的 AWS Lambda 函数中，请参阅步骤 5，该步骤介绍如何使用测试数据测试 Lambda 函数并测试 Lambda 函数。

- 检查规则所使用的角色

  规则操作必须具有接收原始主题和发布新主题的权限。

  授权规则接收消息数据并重新发布的策略特定于所使用的主题。如果更改用于重新发布消息数据的主题，则必须更新规则操作的角色，以更新其策略来匹配当前主题。

  如果怀疑这里出现了问题，请编辑“重新发布”规则操作并创建新角色。规则操作创建的新角色将接收执行这些操作所需的授权。

步骤 4：查看结果和后续步骤

在本教程中：

- 您创建了一 AWS IoT 条规则来调用 Lambda 函数，该函数使用您的自定义消息负载发送了 Amazon SNS 通知。
- 您在规则查询语句中使用了简单的 SQL 查询和函数来为 Lambda 函数创建新的消息负载。
- 您使用 MQTT 客户端来测试您的 AWS IoT 规则。

后续步骤

使用此规则发送了几条文本消息后，请尝试使用它来了解更改教程的某些方面如何影响消息以及发送消息的时间。以下想法可以帮助您开始操作。

- 在输入消息主题中更改 *`device_id`*，并观察文本消息内容中的效果。
- 更改规则查询语句中选定的字段，更新 Lambda 函数以在新消息中使用这些字段，并观察文本消息内容中的效果。

- 将规则查询语句中的测试更改为测试最低温度而不是最高温度。更新 Lambda 函数以格式化新消息，并记得更改 `max_temperature`名称。
- 要详细了解如何查找在开发和使用 AWS IoT 规则时可能出现的错误，请参阅监控 AWS IoT。

在设备处于离线状态时使用设备影子保持设备状态

这些教程介绍如何使用 AWS IoTDevice Shadow 服务来存储和更新设备的状态信息。影子文档是 JSON 文档，根据设备、本地应用程序或服务发布的消息显示设备状态的更改。在本教程中，影子文档将显示灯泡颜色的变化。这些教程还显示影子如何存储此信息，即使设备与互联网断开连接，并在设备重新联机并请求此信息时将最新状态信息传递回设备。

我们建议您按照这里显示的顺序尝试这些教程，从需要创建的 AWS IoT资源以及必要的硬件设置开始，这也有助于您逐步学习这些概念。这些教程展示了如何配置和连接 Raspberry Pi 设备，以便与 AWS IoT搭配使用。如果您没有所需的硬件，您可以按照这些教程将它们适配于您选择的设备或使用 Amazon EC2 创建虚拟设备。

教程场景概览

这些教程的场景是一个本地应用程序或服务，该应用程序或服务可更改灯泡的颜色，并将其数据发布到预留的影子主题。这些教程类似于交互式入门教程中描述的 Device Shadow 功能，并在Raspberry Pi 设备上实现。本部分中的教程侧重于单个经典影子，同时将展示如何容纳已命名的影子或多个设备。

以下教程将帮助您了解如何使用 AWS IoTDevice Shadow 服务。

- 教程：准备 Raspberry Pi 运行影子应用程序

  本教程介绍如何设置 Raspberry Pi 设备，用于连接 AWS IoT。您还将创建一个 AWS IoT策略文档和事物资源，下载证书，然后将策略附加到该事物资源。完成本教程需要大约 30 分钟。
- 教程：安装设备软件开发包并运行 Device Shadow 示例应用程序

  本教程介绍如何安装所需的工具、软件和 AWS IoTDevice SDK for Python，然后运行示例影子应用程序。本教程基于 连接 Raspberry Pi 或其他设备中介绍的概念所打造，完成需要 20 分钟。
- 教程：示例应用程序及 MQTT 测试客户端与 Device Shadow 交互

  本教程介绍了如何使用 `shadow.py`示例应用程序和 AWS IoT控制台来观察 AWS IoTDevice Shadow 和灯泡状态变化之间的交互。本教程还介绍了如何将 MQTT 消息发送到 Device Shadow 的预留主题。完成本教程需要大约 45 分钟。

AWS IoT Device Shadow 概览

Device Shadow 是由您在 AWS IoT注册表中创建的事物资源所管理的设备的持续性虚拟化展示。设备的影子是用于存储和检索设备的当前状态信息的 JSON 或 JavaScript 注释文档。您可以使用该影子通过 MQTT 主题或 HTTP REST APIs 获取和设置设备的状态，无论该设备是否连接到互联网。

影子文档包含一个 `state`属性，以描述设备状态的以下方面。

- `desired`：应用程序更新 `desired`对象以指定设备属性的所需状态。
- `reported`：设备在 `reported`对象中报告其当前状态。
- `delta`：AWS IoT 在 `delta`对象中报告所需的状态和报告的状态之间的差异。

以下为示例影子状态文档：

```
{
  "state": {
    "desired": {
      "color": "green"
      },
    "reported": {
      "color": "blue"
      },
    "delta": {
      "color": "green"
      }
   }
}
```

要更新设备的影子文档，可以使用预留的 MQTT 主题，Device Shadow REST API支持 `GET`、`UPDATE`，和 `DELETE`操作，以及 AWS IoTCLI。

在前面的示例中，假设您希望将 `desired`颜色改为 `yellow`。要执行此操作，请将请求发送到 UpdateThingShadowAPI 或将消息发布到更新主题，`$aws/things/THING_NAME/shadow/update`。

```
{
  "state": {
    "desired": {
      "color": yellow
    }
  }
}
```

更新仅影响请求中指定的字段。成功更新 Device Shadow 后，AWS IoT 将发布新 desired状态至 delta主题。$aws/things/THING_NAME/shadow/delta在这种情况下，影子文档如下所示：

```
{
  "state": {
    "desired": {
      "color": yellow
    },
    "reported": {
      "color": green
    },
    "delta": {
      "color": yellow
      }
  }
}
```

然后，新状态将使用 Update主题 $aws/things/THING_NAME/shadow/update报告给 AWS IoTDevice Shadow，其中包含以下 JSON 消息：

```
{
  "state": {
    "reported": {
      "color": yellow
    }
  }
}
```

如果要获取当前状态信息，请将请求发送至 GetThingShadowAPI 或将 MQTT 消息发布到 Get（获取）主题，$aws/things/THING_NAME/shadow/get。

有关使用 Device Shadow 服务的更多信息，请参阅 AWS IoT Device Shadow 服务。

有关在设备、应用程序和服务中使用 Device Shadow 的更多信息，请参阅 在设备中使用影子和 在应用程序和服务中使用影子。

有关与 AWS IoT影子交互的信息，请参阅 与影子交互。

有关 MQTT 预留主题和 HTTP REST API 的信息，请参阅 Device Shadow MQTT 主题和 Device Shadow REST API。

教程：准备 Raspberry Pi 运行影子应用程序

本教程演示如何设置和配置 Raspberry Pi 设备，并创建设备连接和交换 MQTT 消息所需的 AWS IoT资源。

> **Note**
>
> 如果您计划 the section called “使用 Amazon EC2 创建虚拟设备”，您可以跳过此页面并继续执行 the section called “配置您的设备”。创建虚拟事物时，您将创建这些资源。如果您想使用不同的设备，而不是 Raspberry Pi，可以尝试按照这些教程进行操作，将它们调整到您选择的设备。

在本教程中，您将学习如何：

- 设置 Raspberry Pi 设备，并进行配置，以与 AWS IoT搭配使用。
- 创建 AWS IoT策略文档，该文档将授权您的设备与 AWS IoT服务交互。
- 在 AWS IoTX.509 设备证书中创建事物资源，然后附加策略文档。

  事物是您的设备在 AWS IoT注册表中的虚拟展示。该证书将对您的设备进行身份验证以访问 AWS IoTCore，并且策略文档授权您的设备与 AWS IoT交互。

如何运行本教程？

要为 Device Shadow 运行 `shadow.py`示例应用程序，您将需要一个连接到 AWS IoT的 Raspberry Pi 设备。我们建议您按照此处显示的顺序遵循本教程，首先设置 Raspberry Pi 及其配件，然后创建策略并将策略附加到您创建的事物资源。然后，您可以遵循此教程，使用 Raspberry Pi 支持的图形用户界面 (GUI) 在设备的 Web 浏览器上打开 AWS IoT控制台，这也使您可以更轻松地将证书直接下载到您的 Raspberry Pi，以便连接到 AWS IoT。

在开始本教程之前，请确保您具有：

- 一个 AWS 账户。如果您没有账户，请完成 设置你的 AWS 账户中介绍的步骤然后继续操作。您将需要用到您的 AWS 账户和 AWS IoT控制台以完成本教程。
- Raspberry Pi 及其必要的配件。您将需要：
  - Raspberry Pi 3 Model B 或更新型号。本教程可能适用于早期版本的 Raspberry Pi，但我们还没有测试过。

- Raspberry Pi OS (32 位)或更高版本。我们始终建议使用最新版本的 Raspberry Pi OS。早期版本的操作系统可能有用，但我们还没有测试过。
- 以太网或 Wi-Fi 连接。
- 键盘、鼠标、显示器、电缆和电源。

完成本教程需要大约 30 分钟。

步骤 1：设置和配置 Raspberry Pi 设备

在本部分中，我们将配置一个 Raspberry Pi 设备，用于 AWS IoT。

> **Important**
>
> 将这些指令用于其它设备和操作系统可能会非常困难。您需要充分了解您的设备，以便能够解释这些说明并将它们应用到您的设备上。如果遇到困难，可以尝试使用其它设备选项之一作为替代方案，例如 使用 Amazon EC2 创建虚拟设备或者 使用你的 Windows、Linux 电脑或 Mac 作为 AWS IoT 设备。

您需要配置您的 Raspberry Pi，以便它可以启动操作系统 (OS)，连接到互联网，并允许您在命令行界面与它进行交互。您还可以使用 Raspberry Pi 支持的图形用户界面 (GUI) 打开 AWS IoT控制台并运行本教程的其余部分。

要设置 Raspberry Pi

1. 将 SD 卡插入 Raspberry Pi 上的 MicroSD 卡槽。有些 SD 卡预装了一个安装管理器，将为您显示在启动主板后安装操作系统的菜单。您还可以使用 Raspberry Pi 在您的卡上安装操作系统。
2. 将 HDMI TV 或显示器 Connect 到 HDMI 电缆，然后连接到 Raspberry Pi 的 HDMI 端口。
3. 将键盘和鼠标连接到 Raspberry Pi 的 USB 端口，然后插入电源适配器以启动主板。

Raspberry Pi 启动后，如果 SD 卡预先加载了安装管理器，则会出现一个菜单来帮助您安装操作系统。如果您在安装操作系统时遇到问题，请尝试以下步骤。有关设置 Raspberry Pi 的更多信息，请参阅设置您的 Raspberry Pi。

如果您在设置 Raspberry Pi 时遇到问题：

- 在启动主板之前，请检查是否插入了 SD 卡。如果在启动主板后插入 SD 卡，则可能不会显示安装菜单。

- 确保电视或显示器已打开，并且选择了正确的输入信号源。
- 确保您使用的是兼容 Raspberry Pi 的软件。

安装操作系统后，配置 Raspberry Pi 操作系统后，打开 Raspberry Pi 的网络浏览器并导航到 AWS IoT Core控制台继续本教程中的其余步骤。

如果您能打开 AWS IoT Core控制台，您的 Raspberry Pi 已经准备就绪。可以继续 the section called “在 AWS IoT中预置设备 ”。

如果您遇到问题或需要其它帮助，请参阅为您的 Raspberry Pi 获取帮助。

教程：在 AWS IoT中预置设备

本节将创建教程将使用的 AWS IoT Core资源。

在 AWS IoT中配置设备的步骤

- 步骤 1：创建 Device Shadow 的 AWS IoT策略
- 步骤 2：创建事物资源并将策略附加到事物
- 步骤 3：查看结果和后续步骤

步骤 1：创建 Device Shadow 的 AWS IoT策略

X.509 证书使用 AWS IoT Core对您的设备进行身份验证。AWS IoT 策略附加到允许设备执行 AWS IoT操作（如订阅或发布 Device Shadow 服务使用的 MQTT 预留主题）的证书。您的设备在与 AWS IoT Core连接并向其发送消息时会提供其证书。

在此流程中，您将创建一个策略以允许您的设备执行运行示例程序所必要的 AWS IoT操作。建议您创建一个策略以仅授予执行任务所需的许可。您可以先创建 AWS IoT策略，然后将其附加到稍后将创建的设备证书。

创建 AWS IoT策略

1. 在左侧菜单中，选择 Secure（安全），然后选 Policies（策略）。如果您的账户已有策略，请选择 Create（创建），否则，在 You don’t have a policy yet（您还没有策略）页面上，选择 Create a policy（创建策略）。
2. 在 Create a policy（创建策略）页面上：

   a. 在 Name（名称）字段，输入策略的名称（例如 **`My_Device_Shadow_policy`**）。请勿在策略名称中使用个人身份信息。

b. 在策略文档中，您描述了授予设备发布和订阅 MQTT 保留主题权限的连接、订阅、接收和发布操作。

复制以下策略并将其粘贴到策略文档中。将 thingname替换为您要创建的事物名称（例如 My_light_bulb），将 region替换为您正在使用这些服务的 AWS IoT区域，将 account替换为您的 AWS 账户数量。有关 AWS IoT策略的更多信息，请参阅 AWS IoT Core 政策。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingname/shadow/get",
        "arn:aws:iot:region:account:topic/$aws/things/thingname/shadow/update"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingname/shadow/get/
accepted",
        "arn:aws:iot:region:account:topic/$aws/things/thingname/shadow/get/
rejected",
        "arn:aws:iot:region:account:topic/$aws/things/thingname/shadow/update/
accepted",
        "arn:aws:iot:region:account:topic/$aws/things/thingname/shadow/update/
rejected",
        "arn:aws:iot:region:account:topic/$aws/things/thingname/shadow/update/
delta"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
```

```
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingname/shadow/
get/accepted",
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingname/shadow/
get/rejected",
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingname/shadow/
update/accepted",
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingname/shadow/
update/rejected",
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingname/shadow/
update/delta"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:region:account:client/test-*"
    }
  ]
}
```

步骤 2：创建事物资源并将策略附加到事物

连接到 AWS IoT的设备在 AWS IoT注册表中由事物资源表示。事物资源表示特定设备或逻辑实体，如本教程中的灯泡。

要了解如何在 AWS IoT创建事物，请执行 创建一个事物对象中所述的步骤。以下是您在遵循该教程中的步骤时需要注意的几个关键事项：

1. 选择 Create a single thing（创建单个事物），并在 Name（名称）字段中，输入 与您在之前创建策略时指定的 `thingname`相同的事物名称（例如，`My_light_bulb`)。

   事物名称一旦创建，便不可更改。如果您给它一个不同于 `thingname`的名字，请以 `thingname`为名称创建新的事物，并删除旧的事物。

   > **Note**
   >
   > 请勿在事物名称中使用个人身份信息。事物名称可以出现在未加密的通信和报告中。

2. 我们建议您将 Certificate created!（已创建证书！）页面的每个证书文件下载到您可以轻松找到的位置。您需要安装这些文件才能运行示例应用程序。

建议您将文件下载到 Raspberry Pi 中您的 home目录下的 `certs`子目录中，并使用一个更简单的名称命名各个文件，如下表所示。

证书文件名

文件	文件路径
根 CA 证书	`~/certs/Amazon-root-CA-1.pem`
设备证书	`~/certs/device.pem.crt`
私有密钥	`~/certs/private.pem.key`

3. 激活证书以启用与 AWS IoT的连接后，请选择 Attach a policy（附加策略）并确保将您之前创建的策略（例如，**`My_Device_Shadow_policy`**）附加到事物。

   创建事物后，您可以看到您的事物资源在 AWS IoT控制台中的事物列表里显示。

步骤 3：查看结果和后续步骤

在本教程中，您学习了如何：

- 设置和配置 Raspberry Pi 设备。
- 创建 AWS IoT策略文档，以授权您的设备与 AWS IoT服务交互。
- 创建事物资源和关联的 X.509 设备证书，并将策略文档附加到其中。

后续步骤

您现在可以安装 AWS IoTDevice SDK for Python，运行 `shadow.py`示例应用程序，然后使用 Device Shadow 控制状态。有关如何使用该教程的更多信息，请参阅 教程：安装设备软件开发包并运行 Device Shadow 示例应用程序。

教程：安装设备软件开发包并运行 Device Shadow 示例应用程序

本部分介绍如何安装所需的软件和 AWS IoTDevice SDK for Python，并运行 `shadow.py`示例应用程序来编辑影子文档并控制影子的状态。

在本教程中，您将学习如何：

- 使用已安装的软件和 AWS IoTDevice SDK for Python 以运行示应用程序。

- 了解使用示例应用程序输入值如何在 AWS IoT控制台发布预期的值。
- 查看 shadow.py示例应用程序以及它如何使用 MQTT 协议来更新影子的状态。

在运行本教程之前：

您必须已设置好您的 AWS 账户，配置好您的 Raspberry Pi 设备，并已创建 AWS IoT事物和策略，授予设备发布和订阅 MQTT 预留的 Device Shadow 服务主题的权限。有关更多信息，请参阅 教程：准备 Raspberry Pi 运行影子应用程序。

您必须还安装了 Git、Python 和 AWS IoTDevice SDK for Python。本教程基于教程 连接 Raspberry Pi 或其他设备中介绍的概念执行。如果您尚未尝试该教程，我们建议您按照该教程中描述的步骤安装证书文件和 Device SDK，然后返回本教程运行 shadow.py示例应用程序。

在本教程中，您将：

- 步骤 1：运行 shadow.py 示例应用程序
- 步骤 2：查看 shadow.py 设备软件开发包示例应用
- 步骤 3：借助 shadow.py示例应用程序排查问题
- 步骤 4：查看结果和后续步骤

完成本教程需要大约 20 分钟。

步骤 1：运行 shadow.py 示例应用程序

在运行 shadow.py示例应用程序前，除了安装的证书文件的名称和位置之外，还需要以下信息。

应用程序参数值

参数	在何处查找值
your-iot-thing-name	您之前在 the section called “步骤 2：创建事物资源并将策略附加到事物”中创建的 AWS IoT事物的名称。 要查找此值，请在 AWS IoT控制台中，选择 Manage（管理），然后选择 Things（事物）。
your-iot-endpoint	*your-iot-endpoint* 值的格式为：*endpoint_id* -ats.iot.

参数	在何处查找值
	region.amazonaws.com ，例如，a3qj468EXAMPLE-ats.iot.us-west-2.amazonaws.com 。要查找此值： 1. 在 AWS IoT控制台中，选择 Manage（管理），然后选择 Things（事物）。 2. 选择您为您的设备创建的 IoT 事物，您在之前使用过的 My_light_bulb，然后选择 Interact（交互）。您的端点会显示在事物详细信息页面的 HTTPS 部分中。

安装并运行示例应用程序

1. 导航到示例应用程序目录。

```
cd ~/aws-iot-device-sdk-python-v2/samples
```

2. 在命令行窗口中，按照指示替换 *your-iot-endpoint* 和 *your-iot-thing-name* 的值并运行此命令。

```
python3 shadow.py --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/
device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint --
thing_name your-iot-thing-name
```

3. 观察示例应用程序：

   1. 为您的账户连接到 AWSIoT 服务。
   2. 订阅 Delta事件和 Update及 Get响应。
   3. 提示您在终端中输入所需的值。
   4. 输出类似于以下内容：

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to Delta events...
Subscribing to Update responses...
```

```
Subscribing to Get responses...
Requesting current shadow state...
Launching thread to read user input...
Finished getting initial shadow state.
Shadow contains reported value 'off'.
Enter desired value:
```

> **Note**
>
> 如果您在运行 shadow.py示例应用程序时遇到问题，请查看 the section called “步骤 3：借助 shadow.py示例应用程序排查问题”。若要获取可能帮助您更正此问题的其它信息，请将 --verbosity debug参数添加到命令行，以便示例应用程序显示有关其正在执行的操作的详细消息。

在影子文档中输入值并观察更新

您可以在终端中输入值来指定 desired值，该值还会更新 reported值。假设您在终端中输入颜色 yellow。reported 值也会更新为颜色 yellow。下面显示了终端中显示的消息：

```
Enter desired value:
yellow
Changed local shadow value to 'yellow'.
Updating reported shadow value to 'yellow'...
Update request published.
Finished updating reported shadow value to 'yellow'.
```

发布此更新请求时，AWS IoT 将为事物资源创建一个默认的经典影子。您可以在 AWS IoT控制台中观察您发布到 reported和 desired中的值，方法是查看您创建的事物资源的影子文档（例如 My_light_bulb)。要查看影子文档中的更新：

1. 在 AWS IoT控制台中，选择 Manage（管理），然后选择 Things（事物）。
2. 在显示的事物列表中，选择您创建的事物，选择 Shadows（影子），然后选择 Classic Shadow（经典影子）。

影子文档应类似于以下内容，显示为颜色 yellow设置的 reported和 desired值。您可以在文档的 Shadow state（影子状态）部分看到这些值。

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "yellow"
},
"reported": {
  "welcome": "aws-iot",
  "color": "yellow"
}
}
```

您还可以看到 Metadata（元数据）部分，其中包含请求的时间戳信息和版本号。

您可以使用状态文档版本来确保正在更新的设备影子文档为最新版本。如果您发送另一个更新请求，则版本号将增加 1。当您为更新请求提供版本时，如果状态文档的当前版本与提供的版本不符，该服务将显示 HTTP 409 冲突响应代码并拒绝请求。

```
{
"metadata": {
  "desired": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620156893
    }
  },
  "reported": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620156893
    }
  }
},
"version": 10
}
```

要了解有关影子文档的详细信息并观察状态信息的更改，请继续阅读下一教程 教程：示例应用程序及 MQTT 测试客户端与 Device Shadow 交互，如本教程的 步骤 4：查看结果和后续步骤部分所述。或者，您也可以在下一部分中了解 shadow.py示例代码以及它如何使用 MQTT 协议。

步骤 2：查看 shadow.py 设备软件开发包示例应用

本部分将回顾本教程中适用的来自 AWS IoTDevice SDK v2 for Python 中的 shadow.py示例应用程序。在这里，我们将回顾它如何使用 MQTT 和采用 WSS 协议的 MQTT 来连接到 AWS IoT Core。AWS 公共运行时 (AWS-CRT) 库提供低级通信协议支持，并包含在 AWS IoTDevice SDK v2 for Python 中。

虽然本教程使用 MQTT 和 采用 WSS 的 MQTT，AWS IoT 支持发布 HTTPS 请求的设备。有关从设备发送 HTTP 消息的 Python 程序示例，请参阅使用 Python 的 requests库的 HTTPS 代码示例。

有关如何对设备通信使用何种协议作出明智决定的信息，请查看 为设备通信选择协议。

MQTT

shadow.py 示例在 mqtt_connection_builder 中调用 mtls_from_path（此处显示）以使用 MQTT 协议建立与 AWS IoT Core 的连接。mtls_from_path 使用 X.509 证书和 TLS v1.2 对设备进行身份验证。AWS-CRT 库处理该连接的较低级别的详细信息。

```
mqtt_connection = mqtt_connection_builder.mtls_from_path(
  endpoint=args.endpoint,
  cert_filepath=args.cert,
  pri_key_filepath=args.key,
  ca_filepath=args.ca_file,
  client_bootstrap=client_bootstrap,
  on_connection_interrupted=on_connection_interrupted,
  on_connection_resumed=on_connection_resumed,
  client_id=args.client_id,
  clean_session=False,
  keep_alive_secs=6
)
```

- endpoint 是您从命令行传入的 AWS IoT端点，而 client_id是在 AWS 区域中唯一标识此设备的 ID。
- cert_filepath、pri_key_filepath 和 ca_filepath是指向设备证书和私钥文件以及根 CA 文件的路径。
- client_bootstrap 是处理套接字通信活动的通用运行时对象，并在调用 前实例化。mqtt_connection_builder.mtls_from_path
- on_connection_interrupted 和 on_connection_resumed是在设备连接中断和恢复时调用的回调函数。

- clean_session 表示是否启动新的持久会话，或者如果会话已存在，则重新连接到现有会话。keep_alive_secs 是保持活动状态的值（以秒为单位），以发送到 CONNECT请求中。在此时间间隔内将自动发送 ping。如果服务器在此值的 1.5 倍之后没有收到 ping，则假定连接丢失。

shadow.py 示例还调用 mqtt_connection_builder中的 websockets_with_default_aws_signing以使用采用 WSS 的 MQTT 协议与 AWS IoT Core建立连接。采用 WSS 的 MQTT 也使用与 MQTT 相同的参数，并采用以下附加参数：

- region 是 Signature V4 身份验证使用的AWS区域，而 credentials_provider是 AWS 凭证，用于进行身份验证。区域是从命令行传递的，credentials_provider 对象在调用 mqtt_connection_builder.websockets_with_default_aws_signing前进行实例化。
- websocket_proxy_options 是 HTTP 代理选项（如果使用代理主机）。在 shadow.py示例应用程序中，此值在调用 mqtt_connection_builder.websockets_with_default_aws_signing前进行实例化。

订阅影子主题和活动

shadow.py 示例尝试建立连接并等待完全连接。如果未连接，命令将排队。连接后，示例将订阅增量事件并更新和获取消息，并发布服务质量 (QoS) 级别为 1 的消息 (mqtt.QoS.AT_LEAST_ONCE)。

当设备订阅 QoS 级别 1 的消息时，消息代理会保存该设备订阅的消息，直到这些消息可以发送到设备。消息代理会重新发送消息，直至收到设备发出的 PUBACK响应。

有关 MQTT 协议的更多信息，请参阅 查看 MQTT 协议和 MQTT。

有关本教程中如何使用 MQTT、采用 WSS 的 MQTT、持久性会话和 QoS 级别的详细信息，请参阅 查看 pubsub.py Device SDK 示例应用程序。

步骤 3：借助 **shadow.py**示例应用程序排查问题

当您运行 shadow.py示例应用程序时，您应该看到终端中显示的一些消息以及输入 desired值的提示。如果程序抛出错误，那么请调试以解决错误，您可以从检查系统是否运行了正确的命令开始。

在某些情况下，错误消息可能会指示连接问题，并且看起来类似于：Host name was invalid for dns resolution 或者 Connection was closed unexpectedly。在这种情况下，以下是您可以检查的几个事项：

- 检查命令中的端点地址

查看您在命令中输入用来运行示例应用程序的 endpoint参数（例如 a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com），然后在 AWS IoT控制台中检查此值。

要检查是否使用了正确的值：

1. 在 AWS IoT控制台中，选择 Manage（管理），然后选择 Things（事物）。
2. 选择您为示应用程序创建的事物（例如 My_light_bulb），然后选择 Interact（交互）。

您的端点会显示在事物详细信息页面的 HTTPS 部分中。您还会看到一条消息，写着：This thing already appears to be connected.

- 检查证书激活

证书通过 AWS IoT Core对您的设备进行身份验证。

要检查您的证书是否处于活动状态：

1. 在 AWS IoT控制台中，选择 Manage（管理），然后选择 Things（事物）。
2. 选择您为示应用程序创建的事物（例如 My_light_bulb），然后选择 Security（安全）。
3. 选择证书，然后从证书的详细信息页面中选择 Actions（操作）。

如果在下拉列表中 Activate（激活）不可用，且您只能选择 Deactivate（停用），则表示您的证书处于活动状态。如果并非如此，请选择 Activate（激活）并重新运行示例程序。

如果程序仍未运行，请检查 certs文件夹中的证书文件名。

- 检查附加到事物资源的策略

当证书对您的设备进行身份验证时，AWS IoT 策略允许设备执行 AWS IoT操作，例如订阅或发布 MQTT 预留主题。

要检查是否附加了正确的策略：

1. 查找如前所述的证书，然后选择 Policies（策略）。
2. 选择显示的策略，并检查它是否描述 connect、subscribe、receive 和 publish操作，授予设备发布和订阅 MQTT 预留主题的权限。

   如需了解示例策略，请参阅 步骤 1：创建 Device Shadow 的 AWS IoT策略。

如果您看到错误消息，指示与 AWS IoT的连接出现问题，这可能是由于您正在为策略使用的权限所导致的。如果是这种情况，建议您从向 AWS IoT资源提供完全访问权限的策略开始，然后重新运行示例程序。您可以编辑当前策略，也可以选择当前策略，选择 Detach（分离），然后创建另一个提

供完全访问权限的策略并将其附加到您的事物资源。您可以稍后将策略限制为运行程序所需的操作和策略。

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

- 检查您的设备 Device SDK 安装

  如果程序仍未运行，您可以重新安装 Device SDK 包以确保 SDK 安装完整且正确。

步骤 4：查看结果和后续步骤

在本教程中，您学习了如何：

- 安装所需的软件、工具和 AWS IoTDevice SDK for Python。
- 了解示例应用程序 `shadow.py`如何使用 MQTT 协议检索和更新影子的当前状态。
- 运行 Device Shadow 的示例应用程序，并在 AWS IoT控制台中观察和更新影子文档。您还学习了如何在运行程序时解决任何问题和修复错误。

后续步骤

您现在可以运行 `shadow.py`示例应用程序，并使用 Device Shadow 控制状态。您可以在 AWS IoT控制台中观察并更新影子文档，并观察示例应用程序相应的增量事件。使用 MQTT 测试客户端，您可以订阅预留影子主题并在运行示例程序时观察主题收到的消息。有关如何使用该教程的更多信息，请参阅教程：示例应用程序及 MQTT 测试客户端与 Device Shadow 交互。

教程：示例应用程序及 MQTT 测试客户端与 Device Shadow 交互

若要与 `shadow.py`示例应用程序交互，请在终端中为 `desired`值输入一个值。例如，您可以向请求指定类似于红绿灯的颜色和 AWS IoT响应，并更新报告的值。

在本教程中，您将学习如何：

- 使用 shadow.py示例应用程序来指定所需的状态并更新影子的当前状态。
- 编辑影子文档以观察增量事件以及 shadow.py示例应用程序如何响应。
- 使用 MQTT 测试客户端订阅影子主题并在运行示例程序时观察更新。

运行本教程之前，您必须具有：

设置您的AWS 账户，配置您的 Raspberry Pi 设备，并创建 AWS IoT事物和策略。您还必须安装了所需的软件、Device SDK、证书文件，并在终端中运行示例程序。有关更多信息，请参阅教程 教程：准备 Raspberry Pi 运行影子应用程序和 步骤 1：运行 shadow.py 示例应用程序。如果您尚未完成这些教程，请先予完成。

在本教程中，您将：

- 步骤 1：使用 shadow.py示例应用程序更新所需值和报告值
- 步骤 2：查看来自 shadow.py中 MQTT 测试客户端中示例应用程序的消息
- 步骤 3：排除 Device Shadow 交互中的错误
- 步骤 4：查看结果和后续步骤

完成本教程需要大约 45 分钟。

步骤 1：使用 **shadow.py**示例应用程序更新所需值和报告值

在前面的教程 步骤 1：运行 shadow.py 示例应用程序中，您学习了在按照部分 教程：安装设备软件开发包并运行 Device Shadow 示例应用程序中所述输入必要数值后，如何在 AWS IoT控制台中观察发布到影子文档的消息。

在前面的示例中，我们将所需的颜色设置为 yellow。输入每个值后，终端将提示您输入另一个 desired值。如果您再次输入相同的值 (yellow)，应用程序会识别这一点，并提示您输入一个新的 desired值。

```
Enter desired value:
yellow
Local value is already 'yellow'.
Enter desired value:
```

现在，假设您输入了颜色 green。AWS IoT 响应请求，并将 reported值更新为 green。这就是当 desired状态不同于 reported状态时产生更新的方式，从而导致增量。

shadow.py 示例应用程序模拟 Device Shadow 交互：

1. 在终端输入 desired值（例如 yellow）来发布所需的状态。
2. 由于 desired状态不同于 reported状态（比如说颜色 green），则会产生增量，并且订阅增量的应用程序会收到此消息。
3. 应用程序响应消息并将其状态更新为 desired值，yellow。
4. 然后，应用程序会发布一条更新消息，其中包含设备状态的新报告值，yellow。

下面显示了终端中显示的消息，其展示了更新申请是如何发布的。

```
Enter desired value:
green
Changed local shadow value to 'green'.
Updating reported shadow value to 'green'...
Update request published.
Finished updating reported shadow value to 'green'.
```

在 AWS IoT控制台中，影子文档将 reported和 desired字段更新后的值反映到 green，然后版本号会增加 1。例如，如果以前的版本号显示为 10，则当前版本号将显示为 11。

> **Note**
>
> 删除影子不会将版本号重置为 0。当您发布更新请求或创建另一个具有相同名称的影子时，您将看到影子版本递增 1。

编辑影子文档以观察增量事件

shadow.py 示例应用程序也订阅 delta事件，并在 desired值出现更改时予以响应。例如，您可以将 desired值更改为颜色 red。为此，请在 AWS IoT控制台中，单击 Edit（编辑）以编辑 Shadow 文档，然后在 JSON 中将 desired设置为 red，同时将 reported值保留为 green。保存更改之前，请保持 Raspberry Pi 上的终端打开状态，因为当出现更改时，您会看到终端中显示的消息。

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "red"
},
"reported": {
```

```
  "welcome": "aws-iot",
  "color": "green"
}
}
```

保存新值后，shadow.py 示例应用程序响应此更改，并在终端中显示指示增量的消息。然后，您应该看到以下消息出现在输入 desired值的提示下方。

```
Enter desired value:
Received shadow delta event.
Delta reports that desired value is 'red'. Changing local value...
Changed local shadow value to 'red'.
Updating reported shadow value to 'red'...
Finished updating reported shadow value to 'red'.
Enter desired value:
Update request published.
Finished updating reported shadow value to 'red'.
```

步骤 2：查看来自 **shadow.py**中 MQTT 测试客户端中示例应用程序的消息

您可以使用 AWS IoT控制台中的 MQTT 测试客户端来监控传递在 AWS 账户中的 MQTT 消息。通过订阅 Device Shadow 服务使用的保留 MQTT 主题，您可以观察在运行示例应用程序时主题收到的消息。

如果尚未使用 MQTT 测试客户端，您可以查看 使用 MQTT 客户端查看 AWS IoT MQTT 消息。它可以帮助您了解如何使用 AWS IoT控制台中的 MQTT 测试客户端来查看通过消息代理时的 MQTT 消息。

1. 打开 MQTT 测试客户端

   打开 AWS IoT控制台中的 MQTT 测试客户端，以便您可以观察 MQTT 主题收到的消息，而不会丢失 MQTT 测试客户端的配置。如果您让 MQTT 测试客户端进入控制台中的其它页面，则 MQTT 测试客户端不会保留任何订阅或消息日志。对于本教程的这一部分，您可以在单独的窗口中打开 AWS IoT事物的影子文档和 MQTT 测试客户端，以便更轻松地观察与 Device Shadow 的交互。

2. 订阅 MQTT 预留的影子主题

   您可以使用 MQTT 测试客户端输入 Device Shadow 的 MQTT 预留主题的名称，并在运行 shadow.py示例应用程序时订阅它们以接收更新。要订阅主题：

   a. 在 AWS IoT控制台中的 MQTT 测试客户端中，选择 Subscribe to a topic（订阅主题）。

   b. 在 Topic filter（主题筛选条件）部分，输入：$aws/things/***thingname***/shadow/update/#。在这里，thingname 是您之前创建的事物资源的名称（例如，My_light_bulb）。

c. 保留附加配置设置的默认值，然后选择 Subscribe（订阅）。

通过使用 # 通配符，您可以同时订阅多个 MQTT 主题，并在单个窗口中观察设备与其影子之间交换的所有消息。有关通配符及其用法的更多信息，请参阅 MQTT 主题。

3. 运行 **shadow.py**示例程序和观察消息

在 Raspberry Pi 的命令行窗口中，如果您已经断开程序的连接，请再次运行示例应用程序，并在 AWS IoT控制台中的 MQTT 测试客户端观察消息。

a. 运行以下命令以重新启动示程序。将 *your-iot-thing-name* 和 *your-iot-endpoint* 替换为您早前创建的 AWS IoT事物名称（例如，My_light_bulb）以及与设备交互的端点名称。

```
cd ~/aws-iot-device-sdk-python-v2/samples
python3 shadow.py --ca_file ~/certs/Amazon-root-CA-1.pem --cert ~/certs/
device.pem.crt --key ~/certs/private.pem.key --endpoint your-iot-endpoint --
thing_name your-iot-thing-name
```

shadow.py 示例应用程序随后会开始运行并检索当前影子状态。如果已删除影子或清除当前状态，程序会将当前值设置为 off，然后提示您输入 desired 值。

```
Connecting to a3qEXAMPLEffp-ats.iot.us-west-2.amazonaws.com with client ID
 'test-0c8ae2ff-cc87-49d2-a82a-ae7ba1d0ca5a'...
Connected!
Subscribing to Delta events...
Subscribing to Update responses...
Subscribing to Get responses...
Requesting current shadow state...
Launching thread to read user input...
Finished getting initial shadow state.
Shadow document lacks 'color' property. Setting defaults...
Changed local shadow value to 'off'.
Updating reported shadow value to 'off'...
Update request published.
Finished updating reported shadow value to 'off'...
Enter desired value:
```

另一方面，如果程序正在运行，并且您将程序重新启动，您将看到终端中报告的最新颜色值。在 MQTT 测试客户端中，您将看到主题 $aws/things/***thingname***/shadow/get 和 $aws/things/***thingname***/shadow/get/accepted 的更新。

假设报告的最新颜色为 `green`。下面显示的是 $aws/things/***thingname***/shadow/get/accepted JSON 文件的内容。

```
{
"state": {
  "desired": {
    "welcome": "aws-iot",
    "color": "green"
  },
  "reported": {
    "welcome": "aws-iot",
    "color": "green"
  }
},
"metadata": {
  "desired": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620161643
    }
  },
  "reported": {
    "welcome": {
      "timestamp": 1620156892
    },
    "color": {
      "timestamp": 1620161643
    }
  }
},
"version": 10,
"timestamp": 1620173908
}
```

b. 在终端中输入 desired值，例如 yellow。shadow.py 示例应用程序响应并在终端中显示以下消息，这些消息显示了 reported值变更为 yellow的情况。

```
Enter desired value:
yellow
Changed local shadow value to 'yellow'.
Updating reported shadow value to 'yellow'...
Update request published.
Finished updating reported shadow value to 'yellow'.
```

在 AWS IoT 控制台中的 MQTT 测试客户端中，您会在 Subscriptions（订阅）下看到以下主题收到一条消息：

- $aws/things/***thingname***/shadow/update：显示两个 desired和 updated值更改为颜色 yellow。
- $aws/things/***thingname***/shadow/update/accepted：显示 desired和 reported状态当前的值及其元数据和版本信息。
- $aws/things/***thingname***/shadow/update/documents：显示 desired和 reported状态之前以及当前的值，及其元数据和版本信息。

由于文档 $aws/things/***thingname***/shadow/update/documents 还包含其它两个主题中包含的信息，我们可以查看该文档以查看状态信息。上一个状态显示报告的值设置为 green、其元数据和版本信息，而当前的状态显示报告的值更新为 yellow。

```
{
"previous": {
  "state": {
    "desired": {
      "welcome": "aws-iot",
      "color": "green"
    },
    "reported": {
      "welcome": "aws-iot",
      "color": "green"
    }
  },
  "metadata": {
    "desired": {
      "welcome": {
```

```
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297898
      }
    },
    "reported": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297898
      }
    }
  },
  "version": 10
},
"current": {
  "state": {
    "desired": {
      "welcome": "aws-iot",
      "color": "yellow"
    },
    "reported": {
      "welcome": "aws-iot",
      "color": "yellow"
    }
  },
  "metadata": {
    "desired": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297904
      }
    },
    "reported": {
      "welcome": {
        "timestamp": 1617297888
      },
      "color": {
        "timestamp": 1617297904
      }
```

```
    }
  },
  "version": 11
},
"timestamp": 1617297904
}
```

c. 现在，如果您输入另一个 desired值，您可以看到 reported值的进一步更改和这些主题收到的消息更新。版本号也会增加 1。例如，如果您输入值 green，之前的状态会报告值 yellow，而当前状态报告值 green。

4. 编辑影子文档以观察增量事件

要观察对增量主题的更改，请在 AWS IoT控制台编辑影子文档。例如，您可以将 desired值更改为颜色 red。为此，请在 AWS IoT控制台中，选择 Edit（编辑），然后在 JSON 中将 desired值设置为红色，同时保持 reported设置为 green。在保存更改之前，请保持终端处于开启状态，因为您将看到终端中报告的增量消息。

```
{
"desired": {
  "welcome": "aws-iot",
  "color": "red"
},
"reported": {
  "welcome": "aws-iot",
  "color": "green"
}
}
```

shadow.py 示例应用程序响应此更改，并在终端中显示指示增量的消息。在 MQTT 测试客户端中，update 主题将收到一条消息，显示对 desired和 reported值的更改。

您还可以看到主题 $aws/things/***thingname***/shadow/update/delta 收到一条消息。要查看消息，请选择该主题，其列于 Subscriptions（订阅）下方。

```
{
"version": 13,
"timestamp": 1617318480,
"state": {
  "color": "red"
},
"metadata": {
```

```
  "color": {
    "timestamp": 1617318480
  }
}
}
```

步骤 3：排除 Device Shadow 交互中的错误

运行影子示例应用程序时，在观察与 Device Shadow 服务的交互过程中可能会遇到问题。

如果程序运行成功并提示您输入 `desired`值，您应该能够通过使用前述影子文档和 MQTT 测试客户端来观察 Device Shadow 交互。但是，如果您无法查看交互，可以检查以下事项：

- 在 AWS IoT控制台中检查事物名称及其影子

  如果您在影子文档中没有看到这些消息，请查看该命令，并确保它与 AWS IoT控制台 中的事物名称相符。您还可以通过选择事物资源，然后选择 Shadows（影子）来查看您是否拥有经典影子。本教程主要侧重于与经典影子的交互。

  您还可以确认您使用的设备已连接到互联网。在 AWS IoT控制台，选择您之前创建的事物，然后选择 Interact（交互）。在事物详细信息页面上，您应该会在此处看到一条消息：`This thing already appears to be connected.`
- 检查您订阅的 MQTT 预留主题

  如果在 MQTT 测试客户端中看不到消息，请检查您订阅的主题是否格式正确。MQTT Device Shadow 主题具有格式 $aws/things/***thingname***/shadow/ 并且可能有 `update`、`get` 或者 `delete` 跟随其后，具体取决于要对影子执行的操作。本教程使用主题 $aws/things/***thingname***/shadow/#，因此请确保您在测试客户端的 Topic filter（主题筛选条件）部分订阅主题时正确输入。

  输入主题名称时，请确保 *thingname* 的名称与您之前创建的 AWS IoT事物的名称相同。您还可以订阅其它 MQTT 主题，以查看是否已成功执行更新。例如，您可以订阅主题 $aws/things/***thingname***/shadow/update/rejected 以在更新请求失败时收到一条消息，让您能够调试连接问题。有关预留主题的更多信息，请参阅 the section called “影子主题”和 Device Shadow MQTT 主题。

步骤 4：查看结果和后续步骤

在本教程中，您学习了如何：

- 使用 shadow.py示例应用程序来指定所需的状态并更新影子的当前状态。
- 编辑影子文档以观察增量事件以及 shadow.py示例应用程序如何响应。
- 使用 MQTT 测试客户端订阅影子主题并在运行示例程序时观察更新。

后续步骤

您可以订阅其它 MQTT 预留主题，以观察影子应用程序的更新。例如，如果您只订阅主题 $aws/things/***thingname***/shadow/update/accepted，则在成功执行更新时只能看到当前状态信息。

您还可以订阅其它影子主题以调试问题或了解有关 Device Shadow 交互的详细信息，还可以调试与 Device Shadow 交互有关的任何问题。有关更多信息，请参阅 the section called “影子主题”和 Device Shadow MQTT 主题。

您还可以选择通过使用命名影子或使用为 LED 而连接到 Raspberry Pi 的附加硬件来扩展您的应用程序，并使用从终端发送的消息观察其状态的变化。

有关 Device Shadow 服务以及在设备、应用程序和服务中使用该服务的详细信息，请参阅 AWS IoT Device Shadow 服务、在设备中使用影子 和 在应用程序和服务中使用影子。

教程：为 AWS IoT Core创建自定义授权方

本教程演示了使用 AWS CLI创建、验证和使用自定义身份验证的步骤。或者，使用本教程，您可以借助 HTTP Publish API 使用 Postman 将数据发送到 AWS IoT Core 。

本教程介绍如何创建一个示例 Lambda 函数，该函数将在启用令牌签名的情况下，借助 create-authorizer调用实现授权和身份验证逻辑。然后使用对授权方进行验证，最后test-invoke-authorizer，您可以使用 HTTP 发布 API 向测试 MQTT 主题发送数据。 AWS IoT Core 示例请求将使用标头指定要调用的授权方，并在请求x-amz-customauthorizer-name标头x-amz-customauthorizer-signature中传递 token-key-name 和。

您将在本教程中学到的内容：

- 如何创建一个 Lambda 函数作为自定义授权方处理程序
- 如何在启用令牌签名的情况下使用创建自定义授权方 AWS CLI

- 如何使用 test-invoke-authorizer命令测试您的自定义授权方
- 如何使用 Postman 发布 MQTT 主题并使用您的自定义授权方验证请求

完成本教程需要大约 60 分钟。

在本教程中，您将：

- 步骤 1：为自定义授权方创建 Lambda 函数
- 步骤 2：为自定义授权方创建公有密钥和私有密钥对
- 步骤3：创建客户授权方资源及其授权
- 步骤 4：通过调用来测试授权者 test-invoke-authorizer
- 步骤 5：测试使用 Postman 发布 MQTT 消息
- 步骤 6：在 MQTT 测试客户端中查看消息
- 步骤 7：查看结果和后续步骤
- 第 8 步：清除

在开始本教程之前，请确保您具有：

- 设置你的 AWS 账户

  你需要你的 AWS 账户 和 AWS IoT 主机才能完成本教程。

  当您用于本教程的账户中至少包含这些 AWS 托管策略时，账户的效果最佳：

  - `IAMFullAccess`
  - `AWSIoTFullAccess`
  - `AWSLambda_FullAccess`

  > **Important**
  >
  > 本教程中使用的 IAM 策略比您在生产实施中应遵循的更宽松。在生产环境中，请确保您的账户和资源策略仅授予必要的权限。
  > 在您为生产创建 IAM 策略时，请确定用户和角色需要的访问权限，然后设计允许他们仅执行这些任务的策略。
  > 有关更多信息，请参阅 IAM 安全最佳实践。

- 安装了 AWS CLI

有关如何安装的信息 AWS CLI，请参阅安装 AWS CLI。本教程需要 AWS CLI 版本`aws-cli/2.1.3 Python/3.7.4 Darwin/18.7.0 exe/x86_64`或更高版本。

- OpenSSL 工具

  本教程中的示例使用 LibreSSL 2.6.5。您还可以在本教程中使用 OpenSSL v1.1.1i 工具。

- 查看 AWS Lambda概览

  如果您 AWS Lambda 以前从未使用过，请查看AWS Lambda并开始使用 Lambda，以了解其术语和概念。

- 回顾了如何在 Postman 中构建请求

  有关更多信息，请参阅构建请求。

- 从以前的教程中删除了自定义授权方

  一次 AWS 账户 只能配置有限数量的自定义授权方。有关如何创建和删除自定义授权方的信息，请参阅 the section called “第 8 步：清除”。

步骤 1：为自定义授权方创建 Lambda 函数

中的自定义身份验证 AWS IoT Core 使用您创建的授权方资源对客户端进行身份验证和授权。您将在本节中创建的函数在客户端连接和访问 AWS IoT 资源时对其进行身份验证 AWS IoT Core 和授权。

Lambda 函数执行以下操作：

- 如果请求来自 test-invoke-authorizer，则它会返回一个带有 `Deny`操作的 IAM 策略。
- 如果请求来自使用 HTTP 的 Passport，且 `actionToken`参数的值为 `allow`，则它会返回一个带有 `Allow`操作的 IAM 策略。否则，它会返回带有 `Deny`操作的 IAM 策略。

要为自定义授权方创建 Lambda 函数：

1. 在 Lambda 控制台，打开函数。
2. 选择创建函数。
3. 确认 Author from scratch（从头开始编写）处于选中状态。
4. 在 Basic information 下：

   a. 在 Function name (函数名称) 中，输入 **`custom-auth-function`**。

b. 在运行时系统中，确认 Node.js 18.x

5. 选择 Create function（创建函数）。

   Lambda 创建 Node.js 函数和执行角色，该角色将授予函数上载日志的权限。Lambda 函数在您调用函数时担任执行角色，并使用执行角色为 AWS SDK 创建凭证和从事件源读取数据。

6. 要在AWS Cloud9编辑器中查看函数的代码和配置，请在设计器窗口custom-auth-function中选择，然后在编辑器的导航窗格中选择 index.js。

   对于脚本语言（如 Node.js），Lambda 包含返回成功响应的基本函数。您可以使用 AWS Cloud9编辑器来编辑您的函数，只要源代码不超过 3MB 即可。

7. 在编辑器中使用以下代码替换 index.js 代码：

```
// A simple Lambda function for an authorizer. It demonstrates
// How to parse a CLI and Http password to generate a response.

export const handler = async (event, context, callback) => {

    //Http parameter to initiate allow/deny request
    const HTTP_PARAM_NAME='actionToken';
    const ALLOW_ACTION = 'Allow';
    const DENY_ACTION = 'Deny';

    //Event data passed to Lambda function
    var event_str = JSON.stringify(event);
    console.log('Complete event :'+ event_str);

    //Read protocolData from the event json passed to Lambda function
    var protocolData = event.protocolData;
    console.log('protocolData value---> ' + protocolData);

    //Get the dynamic account ID from function's ARN to be used
    // as full resource for IAM policy
    var ACCOUNT_ID = context.invokedFunctionArn.split(":")[4];
    console.log("ACCOUNT_ID---"+ACCOUNT_ID);

    //Get the dynamic region from function's ARN to be used
    // as full resource for IAM policy
    var REGION = context.invokedFunctionArn.split(":")[3];
    console.log("REGION---"+REGION);

    //protocolData data will be undefined if testing is done via CLI.
```

```
    // This will help to test the set up.
    if (protocolData === undefined) {

        //If CLI testing, pass deny action as this is for testing purpose only.
        console.log('Using the test-invoke-authorizer cli for testing only');
        callback(null, generateAuthResponse(DENY_ACTION,ACCOUNT_ID,REGION));

    } else{

        //Http Testing from Postman
        //Get the query string from the request
        var queryString = event.protocolData.http.queryString;
        console.log('queryString values -- ' + queryString);
        /*         global URLSearchParams       */
        const params = new URLSearchParams(queryString);
        var action = params.get(HTTP_PARAM_NAME);

        if(action!=null && action.toLowerCase() === 'allow'){

            callback(null, generateAuthResponse(ALLOW_ACTION,ACCOUNT_ID,REGION));

        }else{

            callback(null, generateAuthResponse(DENY_ACTION,ACCOUNT_ID,REGION));

        }

    }

};

// Helper function to generate the authorization IAM response.
var generateAuthResponse = function(effect,ACCOUNT_ID,REGION) {

    var full_resource = "arn:aws:iot:"+ REGION + ":" + ACCOUNT_ID + ":*";
    console.log("full_resource---"+full_resource);

    var authResponse = {};
    authResponse.isAuthenticated = true;
    authResponse.principalId = 'principalId';

    var policyDocument = {};
    policyDocument.Version = '2012-10-17';
    policyDocument.Statement = [];
```

```
    var statement = {};
    statement.Action = 'iot:*';
    statement.Effect = effect;
    statement.Resource = full_resource;
    policyDocument.Statement[0] = statement;
    authResponse.policyDocuments = [policyDocument];
    authResponse.disconnectAfterInSeconds = 3600;
    authResponse.refreshAfterInSeconds = 600;

    console.log('custom auth policy function called from http');
    console.log('authResponse --> ' + JSON.stringify(authResponse));
    console.log(authResponse.policyDocuments[0]);

    return authResponse;
}
```

8. 选择部署。
9. 在 Changes deployed（更改已部署）出现在编辑器上方后：

   a. 滚动到编辑器上方的 Function overview（函数概览）部分。
   b. 复制 Function ARN（函数 ARN）并保存以在本教程稍后的部分中使用。

10. 测试 函数。

    a. 选择 Test（测试）选项卡。
    b. 使用默认测试设置，选择 Invoke（调用）。
    c. 如果测试成功，则在 Execution results（执行结果）下，打开 Details（详细信息）视图。您应该看到函数返回的策略文档。

       如果测试失败或没有看到策略文档，请查看代码以查找并更正错误。

步骤 2：为自定义授权方创建公有密钥和私有密钥对

您的自定义授权方需要公有密钥和私有密钥来对其进行身份验证。本部分中的命令使用 OpenSSL 工具来创建此密钥对。

要为自定义授权方创建公有和私有密钥对，请执行以下操作

1. 创建私有密钥文件。

```
openssl genrsa -out private-key.pem 4096
```

2. 验证您刚创建的私有密钥文件。

```
openssl rsa -check -in private-key.pem -noout
```

如果命令未显示任何错误，则私有密钥文件是有效的。

3. 创建公有密钥文件。

```
openssl rsa -in private-key.pem -pubout -out public-key.pem
```

4. 验证公有密钥文件。

```
openssl pkey -inform PEM -pubin -in public-key.pem -noout
```

如果命令未显示任何错误，则公有密钥文件是有效的。

步骤3：创建客户授权方资源及其授权

AWS IoT 自定义授权器是将前面步骤中创建的所有元素联系在一起的资源。在本部分中，您将创建一个自定义授权方资源，并授予其运行您之前创建的 Lambda 函数的权限。您可以使用 AWS IoT 控制台、或 AWS API 创建自定义授权方 AWS CLI资源。

在本教程中，您只需要创建一个自定义授权方。本节介绍如何使用 AWS IoT 控制台和进行创建 AWS CLI，以便您可以使用最方便的方法。以两种方法创建的自定义授权方资源之间没有区别。

创建自定义授权方资源

选择以下选项之一以创建自定义授权方资源

- 使用控制台创建自定义授权方 AWS IoT
- 使用 AWS CLI创建自定义授权方

要创建自定义授权方（控制台）

1. 打开AWS IoT 控制台的 “自定义授权者” 页面，然后选择 “创建授权者”。
2. 在创建授权方中：
   a. 在授权方名称中，输入 **my-new-authorizer**。
   b. 在授权者状态中，选中活动。

c. 在 Authorizer function（授权方函数）中，选择您之前创建的 Lambda 函数。

d. 在 Token validation - optional（令牌验证 — 可选）中：

i. 开启令牌验证。

ii. 在令牌密钥名称中，输入 **tokenKeyName**。

iii. 选择 Add key (添加密钥)。

iv. 在 密钥名称中，输入 **FirstKey**。

v. 在公有密钥中，输入 public-key.pem 文件的内容。请务必包含文件中的行以及 -----BEGIN PUBLIC KEY-----和 -----END PUBLIC KEY-----，并且不要从文件内容中添加或删除任何换行符、回车符或其它字符。您输入的字符串应类似下面这个示例。

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAvEBzOk4vhN+3LgslvEWt
sLCqNmt5Damas3bmiTRvq2gjRJ6KXGTGQChqArAJwL1a9dkS9+maaXC3vc6xzx9z
QPu/vQOe5tyzz1MsKdmtFGxMqQ3qjEXAMPLEOmqyUKPP5mff58k6ePSfXAnzBH0q
lg2HioefrpU5OSAnpuRAjYKofKjbc2Vrn6N2G7hV+IfTBvCElf0csalS/Rk4phD5
oa4Y0GHISRnevypg5C8n9Rrz91PWGqP6M/q5DNJJXjMyleG92hQgu1N696bn5Dw8
FhedszFa6b2x6xrItZFzewNQkPMLMFhNrQIIyvshtT/F1LVCS5+v8AQ8UGGDfZmv
QeqAMAF7WgagDMXcfgKSVU8yid2sIm56qsCLMvD2Sq8Lgzpey9N5ON1o1Cvldwvc
KrJJtgwW6hVqRGuShnownLpgG86M6neZ5sRMbVNZO8OzcobLngJ0Ibw9KkcUdklW
gvZ6HEJqBY2XE70iEXAMPLETPHzhqvK6Ei1HGxpHsXx6BNft582J1VpgYjXha8oa
/NN7l7Zbj/euAb41IVtmX8JrD9z613d1iM5L8HluJlUzn62Q+VeNV2tdA7MfPfMC
8btGYladFAnitThaz6+F0VSBJPu7pZQoLnqyEp5zLMtF+kFl2yOBmGAP0RBivRd9
JWBUCG0bqcLQPeQyjbXSOfUCAwEAAQ==
-----END PUBLIC KEY-----
```

3. 选择 Create Authorizer（创建授权方）。

4. 如果创建了自定义授权方资源，您将看到自定义授权方列表，并且您的新自定义授权方应显示在列表中，然后您便可以继续下一部分展开测试了。

   如果您看到错误，请检查错误并尝试再次创建自定义授权方，然后仔细检查条目。请注意，每个自定义授权方资源必须具有唯一的名称。

要创建自定义授权方（AWS CLI）

1. 将您的值替换为 authorizer-function-arn和 token-signing-public-keys，然后运行以下命令：

```
aws iot create-authorizer \
--authorizer-name "my-new-authorizer" \
--token-key-name "tokenKeyName" \
--status ACTIVE \
--no-signing-disabled \
--authorizer-function-arn "arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-function" \
--token-signing-public-keys FirstKey="-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAvEBzOk4vhN+3LgslvEWt
sLCqNmt5Damas3bmiTRvq2gjRJ6KXGTGQChqArAJwL1a9dkS9+maaXC3vc6xzx9z
QPu/vQOe5tyzz1MsKdmtFGxMqQ3qjEXAMPLEOmqyUKPP5mff58k6ePSfXAnzBH0q
lg2HioefrpU5OSAnpuRAjYKofKjbc2Vrn6N2G7hV+IfTBvCElf0csalS/Rk4phD5
oa4Y0GHISRnevypg5C8n9Rrz91PWGqP6M/q5DNJJXjMyleG92hQgu1N696bn5Dw8
FhedszFa6b2x6xrItZFzewNQkPMLMFhNrQIIyvshtT/F1LVCS5+v8AQ8UGGDfZmv
QeqAMAF7WgagDMXcfgKSVU8yid2sIm56qsCLMvD2Sq8Lgzpey9N5ON1o1Cvldwvc
KrJJtgwW6hVqRGuShnownLpgG86M6neZ5sRMbVNZO8OzcobLngJ0Ibw9KkcUdklW
gvZ6HEJqBY2XE70iEXAMPLETPHzhqvK6Ei1HGxpHsXx6BNft582J1VpgYjXha8oa
/NN7l7Zbj/euAb41IVtmX8JrD9z613d1iM5L8HluJlUzn62Q+VeNV2tdA7MfPfMC
8btGYladFAnitThaz6+F0VSBJPu7pZQoLnqyEp5zLMtF+kFl2yOBmGAP0RBivRd9
JWBUCG0bqcLQPeQyjbXSOfUCAwEAAQ==
-----END PUBLIC KEY-----"
```

其中：

- authorizer-function-arn 值是您为自定义授权方创建的 Lambda 函数的 Amazon Resource Name (ARN)。
- token-signing-public-keys 值包含密钥的名称、**FirstKey** 以及 public-key.pem文件的内容。请务必包含文件中的行以及 -----BEGIN PUBLIC KEY-----和 -----END PUBLIC KEY-----，并且不要从文件内容中添加或删除任何换行符、回车符或其它字符。

  注意：输入公有密钥时要非常小心，因为对公有密钥值的任何更改都会使其无法使用。

2. 如果创建了自定义授权方，则该命令将返回新资源的名称和 ARN，如下所示。

```
{
    "authorizerName": "my-new-authorizer",
    "authorizerArn": "arn:aws:iot:Region:57EXAMPLE833:authorizer/my-new-authorizer"
}
```

保存 authorizerArn值，以供下一步使用。

请记住，每个自定义授权方资源必须具有唯一的名称。

授权自定义授权方资源

在本节中，您将为您刚创建的自定义授权方资源授予运行 Lambda 函数的权限。要授予权限，您可以使用 add-permission CLI 命令。

使用授予您的 Lambda 函数的权限 AWS CLI

1. 插入您的值后，输入以下命令。请注意，`statement-id` 值必须唯一。如果您之前运行过本教程，或者如果您遇到了 `ResourceConflictException`错误，请将 *Id-1234*替换为另一个值。

```
aws lambda add-permission  \
--function-name "custom-auth-function" \
--principal "iot.amazonaws.com" \
--action "lambda:InvokeFunction" \
--statement-id "Id-1234" \
--source-arn authorizerArn
```

2. 如果命令成功，则返回一个权限语句，如本示例。您可以继续到下一部分来测试自定义授权方。

```
{
    "Statement": "{\"Sid\":\"Id-1234\",\"Effect\":\"Allow\",\"Principal
\":{\"Service\":\"iot.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction
\",\"Resource\":\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-
auth-function\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":
\"arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-function\"}}}"
}
```

如果命令不成功，则返回错误，如本示例。在继续操作之前，您需要查看并更正错误。

```
An error occurred (AccessDeniedException) when calling the AddPermission operation:
 User: arn:aws:iam::57EXAMPLE833:user/EXAMPLE-1 is not authorized to perform:
 lambda:AddPer
mission on resource: arn:aws:lambda:Region:57EXAMPLE833:function:custom-auth-
function
```

步骤 4：通过调用来测试授权者 test-invoke-authorizer

定义好所有资源后，在本节中，您将 test-invoke-authorizer从命令行调用以测试授权通行证。

请注意，在通过命令行调用授权方时，protocolData 并未定义，因此授权方将始终返回 DENY 文档。但是，此测试将确认您的自定义授权方和 Lambda 函数是否已正确配置，即使它并未完全测试 Lambda 函数。

要测试您的自定义授权方及其 Lambda 函数，请使用 AWS CLI

1. 在包含您在上一步中创建的 private-key.pem文件的目录中，请运行以下命令。

```
echo -n "tokenKeyValue" | openssl dgst -sha256 -sign private-key.pem | openssl
 base64 -A
```

此命令将创建要在下一步中使用的签名字符串。签名字符串如下所示：

```
dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr+rbCvmu9Jl4KHAA9DG+V
+MMWu09YSA86+64Y3Gt4tOykpZqn9mn
VB1wyxp+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg
+CPY0zrWt1jr9BikgGPDxWkjaeeh
bQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsHNig1JePgnuOBvMGCEFE09jGjj
szEHfgAUAQIWXiVGQj16BU1xKpTGSiTAwheLKUjITOEXAMPLECK3aHKYKY
+d1vTvdthKtYHBq8MjhzJ0kggbt29V
QJCb8RilN/P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca
+tsDuX
f3LzCwQQF/YSUy02u5XkWn
+sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o+K
EWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h
+flFeloZlAWQFH
xRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=
```

复制此签名字符串以在下一步中使用。请注意，不要包含任何额外的字符或遗漏任何字符。

2. 在此命令中，将 token-signature值替换为上一步中的签名字符串，并运行此命令来测试您的授权程序。

```
aws iot test-invoke-authorizer \
--authorizer-name my-new-authorizer \
--token tokenKeyValue \
--token-signature dBwykzlb+fo+JmSGdwoGr8dyC2qB/IyLefJJr
+rbCvmu9Jl4KHAA9DG+V+MMWu09YSA86+64Y3Gt4tOykpZqn9mnVB1wyxp
```

```
+0bDZh8hmqUAUH3fwi3fPjBvCa4cwNuLQNqBZzbCvsluv7i2IMjEg
+CPY0zrWt1jr9BikgGPDxWkjaeehbQHHTo357TegKs9pP30Uf4TrxypNmFswA5k7QIc01n4bIyRTm90OyZ94R4bdJsh
+d1vTvdthKtYHBq8MjhzJ0kggbt29VQJCb8RilN/
P5+vcVniSXWPplyB5jkYs9UvG08REoy64AtizfUhvSul/r/F3VV8ITtQp3aXiUtcspACi6ca
+tsDuXf3LzCwQQF/YSUy02u5XkWn
+sto6KCkpNlkD0wU8gl3+kOzxrthnQ8gEajd5Iylx230iqcXo3osjPha7JDyWM5o
+KEWckTe91I1mokDr5sJ4JXixvnJTVSx1li49IalW4en1DAkc1a0s2U2UNm236EXAMPLELotyh7h
+flFeloZlAWQFHxRlXsPqiVKS1ZIUClaZWprh/orDJplpiWfBgBIOgokJIDGP9gwhXIIk7zWrGmWpMK9o=
```

如果命令成功，它将返回自定义授权方函数生成的信息，如此示例。

```
{
    "isAuthenticated": true,
    "principalId": "principalId",
    "policyDocuments": [
        "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Action\":\"iot:*\",\"Effect
\":\"Deny\",\"Resource\":\"arn:aws:iot:Region:57EXAMPLE833:*\"}]}"
    ],
    "refreshAfterInSeconds": 600,
    "disconnectAfterInSeconds": 3600
}
```

如果命令返回错误，请查看错误并仔细检查您在本节中使用的命令。

步骤 5：测试使用 Postman 发布 MQTT 消息

1. 要从命令行获取设备数据端点，请如此处所示调用 describe-endpoint

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

保存此地址以在后续步骤中用作 *device_data_endpoint_address*。

2. 打开一个新的 Postman 窗口并创建一个新的 HTTP POST 请求。

   a. 在您的计算机中，打开 Postman 应用程序。

   b. 在 Postman 的 File（文件）菜单中，选择 New...（新建）。

   c. 在 New（新建）对话框中，选择 Request（请求）。

   d. 在“保存”请求中，

      i. 在 Request name（请求名称）中，输入 **Custom authorizer test request**。

ii. 在 Select a collection or folder to save to:（选择要保存到的收藏夹或文件夹：）中选择或创建要保存此请求的收藏夹。

iii. 选择 Save to ***collection_name***（保存到“collection_name”）。

3. 创建 POST 请求以测试您的自定义授权方。

a. 在 URL 字段旁边的请求方法选择器中，选择 POST。

b. 在 URL 字段中，通过使用以下 URL 和来自 describe-endpoint 命令的 *device_data_endpoint_address* 为您的请求创建 URL。

```
https://device_data_endpoint_address:443/topics/test/cust-auth/topic?
qos=0&actionToken=allow
```

请注意，此 URL 包含 `actionToken=allow`查询参数，该参数将告诉您的 Lambda 函数返回允许访问 AWS IoT的策略文档。输入 URL 后，查询参数也会显示在 Postman 的 Params（参数）选项卡上。

c. 在 Auth（身份验证）选项卡上的 Type（类型）字段中，选择 No Auth（无身份验证）。

d. 在标头选项卡中：

i. 如果选择了 Host（主机）密钥，请取消选中此项。

ii. 在标头列表的底部添加这些新标头并确认它们被选中。使用您在上一节中用于 test-invoke-authorize命令的签名字符串，将 **Host**值替换为您的 *device_data_endpoint_address* 和 **x-amz-customauthorizer-signature**值。

键	值
x-amz-customauthorizer-name	**my-new-authorizer**
Host	*device_data_endpoint_address*
tokenKeyName	**tokenKeyValue**
x-amz-customauthorizer-signature	dbwykzlb+fo+jms 8dyc2qb/ jjr+@@ *rbcvmu9jl4khaa9dg+v+mmwu09ysa86+64y3gt4 9mnvb1wyxp+0bdzh8hmquauh3fwi3*

键	值
	ca4cwnulqmu9jl4khaa9dg+v+mm wu09ysa86+64y3gt4 9mnvb1wyx p+0bdzh8hmquauh3fwi3 ca4cwnulqnqb 7i2I +cpy0zrwt bikggp qhhto357 9pp30uf4 a5k7qic01n4biyrtm90oyz94r4r 4bdjshnig1 obvmgcefe 09jgjszehgauaqiwxivgqj16bu1 xkptgsitawhelkujitoexamplec k3ahkyky+d1 GdwoGr yhbq8mjzj zehgauaqiwxivgqj16bu1xkptgs itawhelkujitoexampleck3ahky ky+d1 yhbq8mjzjze0kggbt2 9vqjcb8riln/p5+vcvnisxwpply b5jkys9uvg08reoy64 /r/ f3vv8i 3 aci6ca+ 3 qqf/ysuy0 2 IyLef u5 +sto6k tOykpZqn fPjBv ZzbCvsluv MjEg DxWkjaeeh b TegKs TrxypNmFsw JePgnu vTvdthKt AtizfUhvSul TtQp aXiUtcsp tsDuXf LzCw XkWnCkpNl kd0wu8gl3+kozxrthnq8geajd5i ylx230iqcxo3osjpha7jdywm5o +ke 91i1mokdr5jxixvnjt vsx1li4ialw4en1dakc1a0s2u2u nm236examplelotyh7H+ AWQF vks1ziucl4en1li4ialw4en1dak c1a0s2u2u2unm236exampleloty h7H+ AWQF vks1ziucl4en1 WckTe dakc1a0s2u2uazwprh/ord biogokjidgp9gwhxiik7 wpmk9o= flFeloZl HxRlXsPqi JplpiWfBg zWrGm

e. 在“正文”选项卡中：

i. 在数据格式选项框中，选择 Raw（原始）。

ii. 在数据类型列表中，选择JavaScript。

iii. 在文本字段中，为测试消息输入此 JSON 消息负载：

```
{
    "data_mode": "test",
    "vibration": 200,
    "temperature": 40
}
```

4. 选择 Send（发送）以提交请求。

   如果请求成功，则返回：

```
{
    "message": "OK",
    "traceId": "ff35c33f-409a-ea90-b06f-fbEXAMPLE25c"
}
```

   成功的响应表示您的自定义授权方允许连接， AWS IoT 并且测试消息已传送给中 AWS IoT Core 介服务器。

   如果返回错误，请查看错误消息、*device_data_endpoint_address*、签名字符串和其它标头值。

将此请求保留在 Postman 中以供下一节使用。

步骤 6：在 MQTT 测试客户端中查看消息

在上一步中，您使用 Postman 向发送 AWS IoT 了模拟设备消息。成功的响应表明您的自定义授权方允许与 AWS IoT 的连接并且测试消息已经传送到 AWS IoT Core中的代理处。在本节中，您将像其他设备和服务一样使用 AWS IoT 控制台中的 MQTT 测试客户端来查看该消息中的消息内容。

要查看自定义授权方授权的测试消息

1. 在 AWS IoT 控制台中，打开 MQTT 测试客户端。
2. 在 Subscribe to topic（订阅主题）选项卡中的 Topic filter（主题筛选条件）中，输入 **test/cust-auth/topic**，这是前一节中 Postman 示例使用的消息主题。

3. 选择订阅。

   在下一步中，保持此窗口可见。

4. 在 Postman 中，在您为上一节创建的请求中，选择 Send（发送）。

   查看响应以确保响应成功。如果没有，请按照上一节所述对错误进行故障排除。

5. 在 MQTT test client（MQTT 测试客户端），您应该看到一个新条目，其中显示消息主题以及来自从 Postman 发送的请求（如果已展开）的消息负载。

   如果您未在 MQTT test client（MQTT 测试客户端）中看到您的消息，可以检查以下事项：

   - 确保您的 Postman 请求成功返回。如果 AWS IoT 拒绝连接并返回错误，则请求中的消息不会传递给消息代理。
   - 确保 AWS 区域 用于打开 AWS IoT 控制台的 AWS 账户 和与你在 Postman URL 中使用的和相同。
   - 请确保您已在 MQTT test client（MQTT 测试客户端）中正确输入主题。主题筛选条件区分大小写。如有疑问，您也可以订阅该#主题，该主题将订阅通过消息代理 AWS 账户 并 AWS 区域 用于打开控制台的所有 MQTT 消息。 AWS IoT

步骤 7：查看结果和后续步骤

在本教程中：

- 您创建了一个 Lambda 函数作为自定义授权方处理程序
- 您创建了启用令牌签名的自定义授权方
- 您使用 test-invoke-authorizer命令测试了自定义授权方
- 您使用 Postman 发布了一个 MQTT 主题通过并使用您的自定义授权方验证请求
- 您使用了 MQTT test client（MQTT 测试客户端）查看 Postman 测试发送的消息

后续步骤

从 Postman 发送一些消息以验证自定义授权方是否正常工作后，请尝试试验以了解更改本教程的不同方面会对结果产生何种影响。以下是一些入门示例。

- 更改签名字符串，使其不再有权查看未经授权的连接尝试的处理方式。您应该得到一个错误响应，如此所示，并且该消息不应出现在 MQTT test client（MQTT 测试客户端）中。

```
{
    "message": "Forbidden",
    "traceId": "15969756-a4a4-917c-b47a-5433e25b1356"
}
```

- 要详细了解如何查找在开发和使用 AWS IoT 规则时可能出现的错误，请参阅监控 AWS IoT。

第 8 步：清除

如果您想重复本教程，可能需要删除一些自定义授权方。您 AWS 账户 一次只能配置有限数量的自定义授权方，当您尝试在不删除现有自定义授权方的情况下添加新的自定义授权方`LimitExceededException`时，可以获得。

要删除自定义授权方（控制台）

1. 打开AWS IoT 控制台的自定义授权者页面，在自定义授权者列表中，找到要移除的自定义授权者。
2. 打开“自定义授权方详细信息”页面，然后从 Actions（操作）菜单中，选择 Edit（编辑）。
3. 取消选中 Activate authorizer（激活授权方），然后选择 Update（更新）。

   您无法在自定义授权方处于活动状态时将其删除。
4. 在“自定义授权方详细信息”页面，打开 Actions（操作）菜单，然后选择 Delete（删除）。

要删除自定义授权方（AWS CLI）

1. 列出您已安装的自定义授权方，并查找要删除的自定义授权方的名称。

   ```
   aws iot list-authorizers
   ```
2. 在将 *Custom_Auth_Name* 替换为要删除的自定义授权方的 `authorizerName`后，运行此命令，以将自定义授权方设置为 `inactive`。

   ```
   aws iot update-authorizer --status INACTIVE --authorizer-name Custom_Auth_Name
   ```
3. 在将 *Custom_Auth_Name* 替换为要删除的自定义授权方的 `authorizerName`后，运行此命令，以删除自定义授权方。

```
aws iot delete-authorizer --authorizer-name Custom_Auth_Name
```

教程：使用 AWS IoT 和 Raspberry Pi 监测土壤湿度

本教程向您展示如何使用 Raspberry Pi、湿度传感器，以及 AWS IoT 如何监测室内植物或花园的土壤湿度水平。Raspberry Pi 运行的代码从传感器读取湿度和温度，然后将数据发送到 AWS IoT。您可以在中创建一条规则 AWS IoT ，当水分含量低于阈值时，向订阅了 Amazon SNS 主题的地址发送一封电子邮件。

> **Note**
>
> 本教程可能不是最新版本。自本主题最初发布以来，一些参考文献可能已被取代。

目录

先决条件

要完成此教程，需要：

- 一个 AWS 账户。
- 具有管理员权限的 IAM 用户。
- 运行 Windows、macOS、Linux 或 Unix 的开发计算机（用于访问 AWS IoT 控制台）。
- 运行最新 Raspbian OS 的 Raspberry Pi 3B 或 4B。有关安装说明，请参阅 Rasberry Pi 网站上的安装操作系统映像。
- 用于 Raspberry Pi 的显示器、键盘、鼠标和 Wi-Fi 网络或以太网连接。

- 与 Raspberry Pi 兼容的含水量传感器。本教程中使用的传感器是 Adafruit STEMMA I2C 电容式含水量传感器，带有 JST 4 针转母插座电缆接头。

设置 AWS IoT

要完成本教程，您需要创建以下资源。要将设备连接到 AWS IoT，您需要创建物联网事物、设备证书和 AWS IoT 策略。

- 一 AWS IoT 件事。

  事物代表物理设备（在本例中为 Rasberry Pi），包括有关该设备的静态元数据。
- 设备证书。

  所有设备都必须拥有设备证书才能连接到 AWS IoT和通过其身份验证。
- 一项 AWS IoT 政策。

  每个设备证书都有一个或多个与之关联的 AWS IoT 策略。这些策略决定了设备可以访问哪些 AWS IoT 资源。
- AWS IoT 根 CA 证书。

  设备和其他客户端使用 AWS IoT 根 CA 证书对与之通信的 AWS IoT 服务器进行身份验证。有关更多信息，请参阅服务器身份验证。
- 一条 AWS IoT 规则。

  规则包含查询和一个或多个规则操作。查询从设备消息中提取数据，以确定是否应处理该消息数据。规则操作指定当数据与查询匹配时该做些什么。
- 创建 Amazon SNS 主题和主题订阅。

  该规则侦听来自 Raspberry Pi 的含水量数据。如果该值低于阈值，它会向 Amazon SNS 主题发送消息。Amazon SNS 会将该消息发送到订阅该主题的所有电子邮件地址。

步骤 1：创建 AWS IoT 策略

创建允许你的 Raspberry Pi 连接和向其发送消息的 AWS IoT 策略 AWS IoT。

1. 在 AWS IoT 控制台中，如果显示 Get started (开始使用) 按钮，请选择该按钮。否则，请在导航窗格中展开 Secure（安全），然后选择 Policies（策略）。

2. 如果显示您还没有任何策略对话框，请选择创建策略。否则，选择 创建。
3. 输入 AWS IoT 策略的名称（例如，**MoistureSensorPolicy**）。
4. 在添加声明部分中，将现有策略替换为以下 JSON。将*##*和*##*替换为您的 AWS 区域 和 AWS 账户 号码。

```
{
   "Version": "2012-10-17",
   "Statement": [{
         "Effect": "Allow",
         "Action": "iot:Connect",
         "Resource": "arn:aws:iot:region:account:client/RaspberryPi"
      },
      {
         "Effect": "Allow",
         "Action": "iot:Publish",
         "Resource": [
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/
update",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/
delete",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/get"
         ]
      },
      {
         "Effect": "Allow",
         "Action": "iot:Receive",
         "Resource": [
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/
update/accepted",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/
delete/accepted",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/get/
accepted",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/
update/rejected",
            "arn:aws:iot:region:account:topic/$aws/things/RaspberryPi/shadow/
delete/rejected"
         ]
      },
      {
         "Effect": "Allow",
         "Action": "iot:Subscribe",
```

```
         "Resource": [
            "arn:aws:iot:region:account:topicfilter/$aws/things/RaspberryPi/shadow/
update/accepted",
            "arn:aws:iot:region:account:topicfilter/$aws/things/RaspberryPi/shadow/
delete/accepted",
            "arn:aws:iot:region:account:topicfilter/$aws/things/RaspberryPi/shadow/
get/accepted",
            "arn:aws:iot:region:account:topicfilter/$aws/things/RaspberryPi/shadow/
update/rejected",
            "arn:aws:iot:region:account:topicfilter/$aws/things/RaspberryPi/shadow/
delete/rejected"
         ]
      },
      {
         "Effect": "Allow",
         "Action": [
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "iot:DeleteThingShadow"
         ],
         "Resource": "arn:aws:iot:region:account:thing/RaspberryPi"

      }
   ]
}
```

5. 选择 创建。

第 2 步：创建 AWS IoT 事物、证书和私钥

在 AWS IoT 注册表中创建一个代表你的 Raspberry Pi 的东西。

1. 在 AWS IoT 控制台的导航窗格中，依次选择 Manage (管理) 和 Things (事物)。
2. 如果显示您还没有任何事物对话框，请选择注册事物。否则，选择 创建。
3. 在创建 AWS IoT 事物页面上，选择创建单个事物。
4. 在将您的设备添加到设备注册表 页面上，输入您的 IoT 事物的名称（例如 **RaspberryPi**），然后选择下一步。您无法在创建事物后更改其名称。要更改事物的名称，您必须创建一个新事物，为其指定新名称，然后删除旧事物。
5. 在添加事物的证书页面上，选择创建证书。
6. 选择下载链接以下载证书、私有密钥和根 CA 证书。

> **Important**
>
> 这是您下载证书和私有密钥的唯一机会。

7. 选择 Activate（激活）来激活您的证书。证书必须处于活动状态，设备才能连接到 AWS IoT。
8. 选择附加策略。
9. 在 “为您的事物添加策略” 中，选择 MoistureSensorPolicy，然后选择 “注册事物”。

步骤 3：创建 Amazon SNS 主题和订阅

创建 Amazon SNS 主题和订阅。

1. 从 AWS SNS 控制台的导航窗格中，选择 Topics（主题），然后选择 Create topic（创建主题）。
2. 选择 “类型” 作为 “标准”，然后输入主题的名称（例如**`MoistureSensorTopic`**）。
3. 输入主题的显示名称（例如，**`Moisture Sensor Topic`**）。这是在 Amazon SNS 控制台中为您的主题显示的名称。
4. 选择创建主题。
5. 在 Amazon SNS 主题详细信息页面上，选择 Create subscription（创建订阅）。
6. 对于协议，选择电子邮件。
7. 对于 Endpoint (终端节点)，输入您的电子邮件地址。
8. 选择创建订阅。
9. 打开您的电子邮件客户端，查找主题为 **`MoistureSensorTopic`**的消息。打开这封电子邮件，然后单击 Confirm subscription (确认订阅) 链接。

> **Important**
>
> 在确认订阅之前，您不会收到来自该 Amazon SNS 主题的任何电子邮件告警。

您应该会收到一封电子邮件，其中包含您输入的文本。

步骤 4：创建发送电子邮件的 AWS IoT 规则

AWS IoT 规则定义了从设备收到消息时要执行的查询和一项或多项操作。 AWS IoT 规则引擎监听设备发送的消息，并使用消息中的数据来确定是否应采取某些措施。有关更多信息，请参阅的规则 AWS IoT。

在本教程中，您的 Raspberry Pi 在 aws/things/RaspberryPi/shadow/update上发布消息。这是设备和 Thing Shadow 服务使用的内部 MQTT 主题。Raspberry Pi 发布具有以下形式的消息：

```
{
    "reported": {
        "moisture" : moisture-reading,
        "temp" : temperature-reading
    }
}
```

创建查询来从传入消息中提取含水量和温度数据。创建 Amazon SNS 操作来在含水量读数低于阈值时获取数据并将其发送给 Amazon SNS 主题订阅者。

创建 Amazon SNS 规则

1. 在 AWS IoT 控制台中，选择消息路由，然后选择规则。如果显示您还没有任何规则对话框，请选择创建规则。否则，请选择创建规则。
2. 在规则属性页面中，输入规则名称（例如 **MoistureSensorRule**），并提供简短的规则描述（例如 **Sends an alert when soil moisture level readings are too low**）。
3. 选择下一步并配置您的 SQL 语句。选择 SQL 版本为 2016-03-23，然后输入以下 AWS IoT SQL 查询语句：

   ```
   SELECT * FROM '$aws/things/RaspberryPi/shadow/update/accepted' WHERE
    state.reported.moisture < 400
   ```

   当 moisture读数小于 400时，该语句将触发规则操作。

   > **Note**
   >
   > 您可能需要使用其它值。在 Raspberry Pi 上运行代码之后，您可以通过触摸传感器、将其放入水中或放入花盆中来查看从传感器获得的值。

4. 选择下一步并附加规则操作。对于操作 1，选择 Simple Notification Service。此规则操作的描述为将消息作为 SNS 推送通知发送。
5. 对于 SNS 主题，请选择您在步骤 3：创建 Amazon SNS 主题和订阅、中创建的主题 MoistureSensorTopic，并将消息格式保留为 RAW。对于 IAM Role (IAM 角色)，选择 Create a new role (创建新角色)。输入角色的名称（例如 **`LowMoistureTopicRole`**），然后选择创建角色。
6. 选择下一步进行查看，然后选择创建来创建规则。

设置您的 Raspberry Pi 和含水量传感器

将 Micro SD 卡插入 Raspberry Pi 中，连接显示器、键盘、鼠标以及以太网电缆（如果您未使用 Wi-Fi）。先不要连接电源线。

将 JST 跳线连接到含水量传感器。跳线的另一端有四根电线：

- 绿色：I2C SCL
- 白色：I2C SDA
- 红色：电源 (3.5 V)
- 黑色：接地

握住 Raspberry Pi（以太网插孔位于右侧）。以该方向握持时，可看到顶部有两排 GPIO 引脚。按照以下顺序将含水量传感器的电线连接到下排引脚。从最左侧的引脚开始，连接红色（电源）、白色 (SDA) 和绿色 (SCL) 电线。跳过一个引脚，然后连接黑色（接地）电线。有关更多信息，请参阅 Python 计算机接线。

将电源线连接到 Raspberry Pi，然后将另一端插入墙壁插座以将其打开。

配置您的 Raspberry Pi

1. 在 Welcome to Raspberry Pi (欢迎使用 Raspberry Pi) 页面上，选择 Next (下一步)。
2. 选择您的国家/地区、语言、时区和键盘布局。选择下一步。
3. 输入 Raspberry Pi 的密码，然后选择 Next (下一步)。
4. 选择您的 Wi-Fi 网络，然后选择 Next (下一步)。如果不使用 Wi-Fi 网络，则选择 Skip (跳过)。
5. 选择 Next (下一步) 检查软件更新。完成更新后，选择 Restart (重启) 重启您的 Raspberry Pi。

在 Raspberry Pi 启动后，启用 I2C 接口。

1. 在 Raspbian 桌面的左上角，单击 Raspberry 图标，选择 Preferences (首选项)，然后选择 Raspberry Pi Configuration (Raspberry Pi 配置)。
2. 在 Interfaces (接口) 选项卡上，对于 I2C，选择 Enable (启用)。
3. 选择确定。

Adafruit STEMMA 湿度传感器的库是为之编写的。 CircuitPython要在 Raspberry Pi 上运行它们，需要安装最新版本的 Python 3。

1. 在命令提示符中运行以下命令来更新 Raspberry Pi 软件：

   ```
   sudo apt-get update
   ```

   ```
   sudo apt-get upgrade
   ```
2. 运行以下命令，更新您的 Python 3 安装：

   ```
   sudo pip3 install --upgrade setuptools
   ```
3. 运行以下命令，安装 Raspberry Pi GPIO 库：

   ```
   pip3 install RPI.GPIO
   ```
4. 运行以下命令，安装 Adafruit Blinka 库：

   ```
   pip3 install adafruit-blinka
   ```

   有关更多信息，请参阅在 Raspberry Pi 上安装 CircuitPython 库。
5. 运行以下命令，安装 Adafruit Seesaw 库：

   ```
   sudo pip3 install adafruit-circuitpython-seesaw
   ```
6. 运行以下命令安装适用于 Python 的 AWS IoT 设备 SDK：

   ```
   pip3 install AWSIoTPythonSDK
   ```

Raspberry Pi 现已具备所有必需的库。创建一个名为 **moistureSensor.py**的文件，并将以下 Python 代码复制到该文件中：

```
from adafruit_seesaw.seesaw import Seesaw
```

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTShadowClient
from board import SCL, SDA

import logging
import time
import json
import argparse
import busio

# Shadow JSON schema:
#
# {
#   "state": {
#       "desired":{
#           "moisture":<INT VALUE>,
#           "temp":<INT VALUE>
#       }
#   }
# }

# Function called when a shadow is updated
def customShadowCallback_Update(payload, responseStatus, token):

    # Display status and data from update request
    if responseStatus == "timeout":
        print("Update request " + token + " time out!")

    if responseStatus == "accepted":
        payloadDict = json.loads(payload)
        print("~~~~~~~~~~~~~~~~~~~~~~~")
        print("Update request with token: " + token + " accepted!")
        print("moisture: " + str(payloadDict["state"]["reported"]["moisture"]))
        print("temperature: " + str(payloadDict["state"]["reported"]["temp"]))
        print("~~~~~~~~~~~~~~~~~~~~~~~\n\n")

    if responseStatus == "rejected":
        print("Update request " + token + " rejected!")

# Function called when a shadow is deleted
def customShadowCallback_Delete(payload, responseStatus, token):

     # Display status and data from delete request
    if responseStatus == "timeout":
        print("Delete request " + token + " time out!")
```

```
    if responseStatus == "accepted":
        print("~~~~~~~~~~~~~~~~~~~~~~")
        print("Delete request with token: " + token + " accepted!")
        print("~~~~~~~~~~~~~~~~~~~~~~\n\n")

    if responseStatus == "rejected":
        print("Delete request " + token + " rejected!")


# Read in command-line parameters
def parseArgs():

    parser = argparse.ArgumentParser()
    parser.add_argument("-e", "--endpoint", action="store", required=True, dest="host",
 help="Your device data endpoint")
    parser.add_argument("-r", "--rootCA", action="store", required=True,
 dest="rootCAPath", help="Root CA file path")
    parser.add_argument("-c", "--cert", action="store", dest="certificatePath",
 help="Certificate file path")
    parser.add_argument("-k", "--key", action="store", dest="privateKeyPath",
 help="Private key file path")
    parser.add_argument("-p", "--port", action="store", dest="port", type=int,
 help="Port number override")
    parser.add_argument("-n", "--thingName", action="store", dest="thingName",
 default="Bot", help="Targeted thing name")
    parser.add_argument("-id", "--clientId", action="store", dest="clientId",
 default="basicShadowUpdater", help="Targeted client id")

    args = parser.parse_args()
    return args


# Configure logging
# AWSIoTMQTTShadowClient writes data to the log
def configureLogging():

    logger = logging.getLogger("AWSIoTPythonSDK.core")
    logger.setLevel(logging.DEBUG)
    streamHandler = logging.StreamHandler()
    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s -
 %(message)s')
    streamHandler.setFormatter(formatter)
    logger.addHandler(streamHandler)
```

```
# Parse command line arguments
args = parseArgs()

if not args.certificatePath or not args.privateKeyPath:
    parser.error("Missing credentials for authentication.")
    exit(2)

# If no --port argument is passed, default to 8883
if not args.port:
    args.port = 8883


# Init AWSIoTMQTTShadowClient
myAWSIoTMQTTShadowClient = None
myAWSIoTMQTTShadowClient = AWSIoTMQTTShadowClient(args.clientId)
myAWSIoTMQTTShadowClient.configureEndpoint(args.host, args.port)
myAWSIoTMQTTShadowClient.configureCredentials(args.rootCAPath, args.privateKeyPath,
 args.certificatePath)

# AWSIoTMQTTShadowClient connection configuration
myAWSIoTMQTTShadowClient.configureAutoReconnectBackoffTime(1, 32, 20)
myAWSIoTMQTTShadowClient.configureConnectDisconnectTimeout(10) # 10 sec
myAWSIoTMQTTShadowClient.configureMQTTOperationTimeout(5) # 5 sec

# Initialize Raspberry Pi's I2C interface
i2c_bus = busio.I2C(SCL, SDA)

# Intialize SeeSaw, Adafruit's Circuit Python library
ss = Seesaw(i2c_bus, addr=0x36)

# Connect to AWS IoT
myAWSIoTMQTTShadowClient.connect()

# Create a device shadow handler, use this to update and delete shadow document
deviceShadowHandler =
 myAWSIoTMQTTShadowClient.createShadowHandlerWithName(args.thingName, True)

# Delete current shadow JSON doc
deviceShadowHandler.shadowDelete(customShadowCallback_Delete, 5)

# Read data from moisture sensor and update shadow
while True:
```

```
    # read moisture level through capacitive touch pad
    moistureLevel = ss.moisture_read()

    # read temperature from the temperature sensor
    temp = ss.get_temp()

    # Display moisture and temp readings
    print("Moisture Level: {}".format(moistureLevel))
    print("Temperature: {}".format(temp))

    # Create message payload
    payload = {"state":{"reported":{"moisture":str(moistureLevel),"temp":str(temp)}}}

    # Update shadow
    deviceShadowHandler.shadowUpdate(json.dumps(payload), customShadowCallback_Update,
5)
    time.sleep(1)
```

将文件保存到您可以找到的位置。在命令行中运行 `moistureSensor.py`及以下参数：

端点

您的自定义 AWS IoT 终端节点。有关更多信息，请参阅Device Shadow REST API。

rootCA

您的 AWS IoT 根 CA 证书的完整路径。

cert

AWS IoT 设备证书的完整路径。

key

AWS IoT 设备证书私钥的完整路径。

thingName

您的事物的名称（在本例中为 `RaspberryPi`）。

clientId

MQTT 客户端 ID。使用 `RaspberryPi`。

命令行应如下所示：

```
python3 moistureSensor.py --endpoint your-endpoint --rootCA ~/certs/
AmazonRootCA1.pem --cert ~/certs/raspberrypi-certificate.pem.crt --key
~/certs/raspberrypi-private.pem.key --thingName RaspberryPi --clientId
RaspberryPi
```

尝试触摸传感器、将其放入花盆或放入一杯水中，查看传感器对各种含水量有何反应。如果需要，可以在 MoistureSensorRule中更改阈值。当湿度传感器的读数低于规则的 SQL 查询语句中指定的值时，会向 Amazon SNS 主题 AWS IoT 发布一条消息。您应会收到一封包含含水量和温度数据的电子邮件。

确认收到来自 Amazon SNS 的电子邮件后，按 CTRL + C 停止 Python 程序。该 Python 程序发送的消息应该不足以产生费用，但完成后最好将其停止。

使用管理设备 AWS IoT

AWS IoT 提供了一个可帮助您管理事物的注册表。事物是特定设备或逻辑实体的表示形式。它可以是物理设备或传感器（例如，灯泡或墙壁上的开关）。它也可以是一个逻辑实体，例如应用程序或物理实体的实例，该实体不连接 AWS IoT 但与其他连接的设备相关（例如，带有发动机传感器或控制面板的汽车）。

事物的相关信息均以 JSON 数据形式存储在 Registry 中。以下是一个事物示例：

```
{
     "version": 3,
    "thingName": "MyLightBulb",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

事物由名称进行标识。此外，事物还可以具有名称-值对形式的属性，您可以利用这些属性存储事物的相关信息，如事物的序列号或制造商。

典型的设备使用案例使用事物名称作为默认的 MQTT 客户端 ID。虽然我们没有强制在事物的注册表名称和它使用的 MQTT 客户端 ID、证书或影子状态之间进行映射，但建议您选择一个事物名称，并将其同时用作注册表和 Device Shadow 服务的 MQTT 客户端 ID。这不仅为您的 IoT 实例集带来了有序性和便利性，还保留了基础设备证书模型或影子的灵活性。

您无需在注册表中创建事物便能将设备连接到 AWS IoT。将事物添加到注册表可让您更轻松地管理和搜索设备。

如何使用注册表管理事物

您可以使用 AWS IoT 控制台、 AWS IoT API 或与注册表 AWS CLI 进行交互。以下各部分展示了如何使用 CLI 与 Registry 进行交互。

命名事物对象时：

- 请勿在事物名称中使用个人身份信息。事物名称可以出现在未加密的通信和报告中。

创建事物

以下命令显示如何使用 CLI 中的 AWS IoT CreateThing命令来创建事物。您无法在创建事物后更改其名称。要更改事物的名称，请创建一个新事物，为其指定新名称，然后删除旧事物。

```
$ aws iot create-thing --thing-name "MyLightBulb" --attribute-payload "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

CreateThing 命令显示新事物的名称和 Amazon Resource Name (ARN)：

```
{
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
    "thingName": "MyLightBulb",
    "thingId": "12345678abcdefgh12345678ijklmnop12345678"
}
```

> **Note**
>
> 我们建议不要在您的事物名称中使用个人身份信息。

有关更多信息，请参阅《 AWS CLI 命令参考》中的 create-thing。

列出事物

您可以使用 ListThings 命令列出您的账户中的所有事物：

```
$ aws iot list-things
```

```
{
    "things": [
       {
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyLightBulb"
        },
        {
```

```
            "attributes": {
                "numOfStates":"3"
             },
            "version": 11,
            "thingName": "MyWallSwitch"
        }
    ]
}
```

您可以使用 ListThings 命令搜索某个事物类型的所有事物：

```
$  aws iot list-things --thing-type-name "LightBulb"
```

```
{
    "things": [
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyRGBLight"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MySecondLightBulb"
        }
    ]
}
```

您可以使用 ListThings 命令搜索某个属性为特定值的所有事物。此命令最多可搜索三个属性。

```
$  aws iot list-things --attribute-name "wattage" --attribute-value "75"
```

```
{
    "things": [
        {
            "thingTypeName": "StopLight",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3,
            "thingName": "MyLightBulb"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MyRGBLight"
        },
        {
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1,
            "thingName": "MySecondLightBulb"
        }
    ]
}
```

有关更多信息，请参阅《 AWS CLI 命令参考》中的 list-things。

描述事物

您可以使用 DescribeThing 命令获取某事物的更多详情：

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "version": 3,
    "thingName": "MyLightBulb",
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/MyLightBulb",
```

```
    "thingId": "12345678abcdefgh12345678ijklmnop12345678",
    "defaultClientId": "MyLightBulb",
    "thingTypeName": "StopLight",
    "attributes": {
        "model": "123",
        "wattage": "75"
    }
}
```

有关更多信息，请参阅《命令参考》中的 describe- thing AWS CLI 。

更新事物

您可以使用 UpdateThing 命令更新事物。此命令仅更新事物的属性。您无法更改事物的名称。要更改事物的名称，请创建一个新事物，为其指定新名称，然后删除旧事物。

```
$ aws iot update-thing --thing-name "MyLightBulb" --attribute-payload "{\"attributes\":
 {\"wattage\":\"150\", \"model\":\"456\"}}"
```

UpdateThing 命令不会生成任何输出。您可以使用 DescribeThing 命令查看结果：

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "attributes": {
        "model": "456",
        "wattage": "150"
    },
    "version": 2,
    "thingName": "MyLightBulb"
}
```

有关更多信息，请参阅《 AWS CLI 命令参考》中的 update-thing。

删除事物

您可以使用 DeleteThing 命令删除事物：

```
$ aws iot delete-thing --thing-name "MyThing"
```

如果删除成功或者您指定的事物不存在，此命令将成功返回且没有错误。

有关更多信息，请参阅《 AWS CLI 命令参考》中的 delete-thing。

将委托人附加到事物

物理设备必须具有 X.509 证书才能与之通信。 AWS IoT您可以将设备上的证书与 Registry 中代表该设备的事物关联。要将证书附加到您的事物，请使用 AttachThingPrincipal 命令：

```
$ aws iot attach-thing-principal --thing-name "MyLightBulb" --principal
 "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

AttachThingPrincipal 命令不会生成任何输出。

有关更多信息，请参阅《命令参考》中的 attac h-thing- principal。 AWS CLI

将委托人与事物分离

您可以使用 DetachThingPrincipal 命令将证书与事物分离：

```
$ aws iot detach-thing-principal --thing-name "MyLightBulb" --principal
 "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

DetachThingPrincipal 命令不会生成任何输出。

有关更多信息，请参阅《命令参考》中的 detach-thing-principal。 AWS CLI

事物类型

通过定义事物类型，您可以存储与同一事物类型相关联的所有事物共有的描述和配置信息，这将简化 Registry 中的事物管理。例如，您可以定义 LightBulb 事物类型。与该 LightBulb 事物类型相关的所有内容都共享一组属性：序列号、制造商和瓦数。在创建类型的事物 LightBulb （或将现有事物的类型更改为 LightBulb）时，可以为该 LightBulb 事物类型中定义的每个属性指定值。

虽然事物类型是可选项，但使用它们可以更容易地搜索事物。

- 具有事物类型的事物最多可以具有 50 个属性。
- 未设置事物类型的事物最多可以具有三个属性。
- 一个事物只能与一种事物类型相关联。

- 在账户中可以创建的事物类型数量不限。

事物类型是不可变的。事物类型一旦创建，便不可更改。您随时可以弃用某事物类型，以防止新事物与之关联。您也可以删除没有与任何事物关联的事物类型。

创建事物类型

您可以使用 CreateThingType 命令创建事物类型：

```
$ aws iot create-thing-type

                --thing-type-name "LightBulb" --thing-type-properties
 "thingTypeDescription=light bulb type, searchableAttributes=wattage,model"
```

CreateThingType 命令会返回一个含有事物类型及其 ARN 的响应：

```
{
    "thingTypeName": "LightBulb",
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb"
}
```

列出事物类型

您可以使用 ListThingTypes 命令列出事物类型：

```
$ aws iot list-thing-types
```

该ListThingTypes命令会返回在您的：中定义的事物类型的列表 AWS 账户：

```
{
    "thingTypes": [
        {
            "thingTypeName": "LightBulb",
            "thingTypeProperties": {
                "searchableAttributes": [
                    "wattage",
                    "model"
                ],
```

```
                "thingTypeDescription": "light bulb type"
            },
            "thingTypeMetadata": {
                "deprecated": false,
                "creationDate": 1468423800950
            }
        }
    ]
}
```

描述事物类型

您可以使用 DescribeThingType 命令获取某事物类型的相关信息：

```
$ aws iot describe-thing-type --thing-type-name "LightBulb"
```

DescribeThingType 命令返回有关指定类型的信息：

```
{
    "thingTypeProperties": {
        "searchableAttributes": [
            "model",
            "wattage"
        ],
        "thingTypeDescription": "light bulb type"
    },
    "thingTypeId": "df9c2d8c-894d-46a9-8192-9068d01b2886",
    "thingTypeArn": "arn:aws:iot:us-west-2:123456789012:thingtype/LightBulb",
    "thingTypeName": "LightBulb",
    "thingTypeMetadata": {
        "deprecated": false,
        "creationDate": 1544466338.399
    }
}
```

将事物类型与事物相关联

创建事物时，可以使用 CreateThing 命令指定事物类型：

```
$ aws iot create-thing --thing-name "MyLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

您随时可以使用 UpdateThing 命令更改与某个事物关联的事物类型：

```
$ aws iot update-thing --thing-name "MyLightBulb"
                --thing-type-name "LightBulb" --attribute-payload  "{\"attributes\":
 {\"wattage\":\"75\", \"model\":\"123\"}}"
```

您还可以使用 UpdateThing 命令取消事物与事物类型之间的关联。

弃用事物类型

事物类型是不可变的。一旦定义了事务类型，便不可更改。然而，您可以通过弃用某种事物类型来防止用户将新事物与之关联。所有与该事物类型相关联的现有事物将保持不变。

要弃用某事物类型，请使用 DeprecateThingType 命令：

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType"
```

您可以使用 DescribeThingType 命令查看结果：

```
$ aws iot describe-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeProperties": {
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type",
    },
    "thingTypeMetadata": {
        "deprecated": true,
        "creationDate": 1468425854308,
        "deprecationDate": 1468446026349
    }
}
```

弃用事物类型是可逆操作。您可以通过在 `--undo-deprecate` CLI 命令中使用 DeprecateThingType 标志来撤消弃用：

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType" --undo-deprecate
```

您可以使用 DescribeThingType CLI 命令查看结果：

```
$ aws iot describe-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeArn": "arn:aws:iot:us-east-1:123456789012:thingtype/StopLight",
    "thingTypeId": "12345678abcdefgh12345678ijklmnop12345678"
    "thingTypeProperties": {
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type"
    },
    "thingTypeMetadata": {
        "deprecated": false,
        "creationDate": 1468425854308,
    }
}
```

删除事物类型

只有在弃用事物类型后，才能将其删除。要删除某事物类型，请使用 DeleteThingType 命令：

```
$ aws iot delete-thing-type --thing-type-name "StopLight"
```

> **Note**
>
> 在您删除某一事物类型之前，请等待五分钟后再将其弃用。

静态事物组

通过将事物分类成组，您能够使用静态事物组同时管理几个事物。静态事物组包含使用控制台、CLI 或 API 管理的一组事物。另一方面，动态事物组包含与指定查询匹配的事物。静态事物组还可以包含其它

静态组 — 您可以构建组的层次结构。您可以将策略附加到某个父组，该策略将由其所有子组和该组及其子组中的所有事物继承。这使得控制大量事物的权限非常简单。

> **Note**
>
> 事物组策略不允许访问 AWS IoT Greengrass 数据平面操作。要允许事物访问 AWS IoT Greengrass 数据平面操作，请将权限添加到附加到事物证书的 AWS IoT 策略中。有关更多信息，请参阅《AWS IoT Greengrass 开发人员指南》中的开发人员身份验证和授权。

下面是您可以对静态事物组执行的操作：

- 创建、描述或删除组。
- 将事物添加到一个或多个组。
- 从组中删除事物。
- 列出您已创建的组。
- 列出组的所有子组 (其直接和间接后代)。
- 列出组中的事物，包括其子组中的所有事物。
- 列出组的所有原级组 (其直接和间接父级)。
- 添加、删除或更新组的属性。(属性是您可用于存储与组相关的信息的名称/值对。)
- 从组中附加或分离策略。
- 列出附加到组的策略。
- 列出由某个事物继承的策略 (借助附加到其组或其父组之一的策略)。
- 为组中的事物配置日志记录选项。请参阅 配置 AWS IoT 日志。
- 创建将发送到某个组及其子组中的所有事物并在这些事物上执行的任务。请参阅 任务。

> **Note**
>
> 将事物附加到 AWS IoT Core 策略所关联的静态事物组时，事物名称必须与客户端 ID 相匹配。

下面是静态事物组的一些限制：

- 一个组最多可以有一个直接父级。

- 如果一个组是另一个组的子组，请在创建该组时指定该组。
- 以后无法更改组的父组；因此，务必先规划组的层次结构并创建父组，然后再创建父组中将包含的任何子组。
- 一个事物可以属于的组数受到限制。
- 您无法将事物添加到相同层次结构的多个组中。（换言之，您无法将事物添加到父级相同的两个组中。）
- 您无法重命名组。
- 事物组名称不能包含国际字符，如 û、é 和 ñ。
- 请勿在事物组名称中使用个人身份信息。事物组名称可以出现在未加密的通信和报告中。

在组上附加和分离策略能够以多种非常有效的方式增强 AWS IoT 操作的安全性。针对各个设备将策略附加到证书，然后附加到事物的方法非常费时，并且难于在整个设备实例集中快速更新或更改策略。在需要轮换某个事物上的证书时，将策略附加到事物的组可以节省步骤。并且，在更改组成员资格时，策略会动态应用到事物，因此您无需在每次组中设备成员资格更改时重新创建复杂的权限集。

创建静态事物组

使用 CreateThingGroup 命令创建静态事物组：

```
$ aws iot create-thing-group --thing-group-name LightBulbs
```

CreateThingGroup 命令返回包含静态事物组名称、其 ID 和 ARN 的响应：

```
{
    "thingGroupName": "LightBulbs",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
}
```

> **Note**
>
> 我们建议不要在您的事物组名称中使用个人身份信息。

以下示例在创建静态事物组时指定其父级：

```
$ aws iot create-thing-group --thing-group-name RedLights --parent-group-name
 LightBulbs
```

和之前一样，CreateThingGroup 命令返回包含静态事物组名称、其 ID 和 ARN 的响应：

```
{
    "thingGroupName": "RedLights",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
}
```

Important

创建事物组层次结构时，请注意以下限制：

- 一个事物组只可以有一个直接父级。
- 事物组可以拥有的直接子组数量受到限制。
- 组层次结构的最大深度受到限制。
- 事物组可以拥有的属性数量受到限制。(属性是您可用于存储与组相关的信息的名称/值对。) 每个属性名称和每个值的长度也受到限制。

描述事物组

您可以使用 DescribeThingGroup 命令获取某事物组的相关信息：

```
$ aws iot describe-thing-group --thing-group-name RedLights
```

DescribeThingGroup 命令返回有关指定组的信息：

```
{
    "thingGroupName": "RedLights",
    "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights",
    "thingGroupId": "12345678abcdefgh12345678ijklmnop12345678",
    "version": 1,
    "thingGroupMetadata": {
        "creationDate": 1478299948.882
        "parentGroupName": "Lights",
        "rootToParentThingGroups": [
```

```
            {
                "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ShinyObjects",
                "groupName": "ShinyObjects"
            },
            {
                "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs",
                "groupName": "LightBulbs"
            }
        ]
    },
    "thingGroupProperties": {
        "attributePayload": {
            "attributes": {
                "brightness": "3400_lumens"
            },
        },
        "thingGroupDescription": "string"
    },
}
```

将事物添加到静态事物组

您可以使用 AddThingToThingGroup 命令将事物添加到静态事物组：

```
$ aws iot add-thing-to-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

AddThingToThingGroup 命令不会生成任何输出。

> **Important**
>
> 您可以将事物添加到最多 10 个组。但是，您无法将事物添加到相同层次结构的多个组中。(换而言之，您无法将事物添加到共享相同父级的两个组中。)
> 如果一个事物属于尽可能多的事物组，并且其中一个或多个组是动态事物组，则可以使用 overrideDynamicGroups 标志使静态组优先于动态组。

从静态事物组中删除事物

您可以使用 RemoveThingFromThingGroup 命令从组中删除事物：

```
$ aws iot remove-thing-from-thing-group --thing-name MyLightBulb --thing-group-name
 RedLights
```

RemoveThingFromThingGroup 命令不会生成任何输出。

列出事物组中的事物

您可以使用 ListThingsInThingGroup 命令列出属于某个组的事物：

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs
```

ListThingsInThingGroup 命令返回指定组中的事物列表：

```
{
    "things":[
        "TestThingA"
    ]
}
```

使用 `--recursive` 参数，您可以列出属于某个组及其所有子组中的事物：

```
$ aws iot list-things-in-thing-group --thing-group-name LightBulbs --recursive
```

```
{
    "things":[
        "TestThingA",
        "MyLightBulb"
    ]
}
```

> **Note**
>
> 此操作具有最终一致性。换句话说，事物组的更改可能不会立即反映出来。

列出事物组

您可以使用 ListThingGroups 命令列出您账户的事物组：

```
$ aws iot list-thing-groups
```

该ListThingGroups命令会返回您的事物组列表 AWS 账户：

```
{
    "thingGroups": [
        {
            "groupName": "LightBulbs",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
        },
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "RedLEDLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLEDLights"
        },
        {
            "groupName": "RedIncandescentLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedIncandescentLights"
        }
        {
            "groupName": "ReplaceableObjects",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ReplaceableObjects"
        }
    ]
}
```

使用可选的筛选条件列出具有指定组作为父级 (`--parent-group`) 的组，或者名称以指定前缀 (`--name-prefix-filter`) 开头的组。使用 `--recursive` 参数可以列出所有子组，而不仅仅是事物组的直接子组：

```
$ aws iot list-thing-groups --parent-group LightBulbs
```

在这种情况下，该ListThingGroups命令将返回在您中定义的事物组的直接子组列表 AWS 账户：

```
{
    "childGroups":[
```

```
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        }
    ]
}
```

将 `--recursive` 参数与 ListThingGroups 命令结合使用可以列出事物组的所有子组，而不仅仅是直接子组：

```
$ aws iot list-thing-groups --parent-group LightBulbs --recursive
```

ListThingGroups 命令返回事物组的所有子组的列表：

```
{
    "childGroups":[
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "RedLEDLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLEDLights"
        },
        {
            "groupName": "RedIncandescentLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
RedIncandescentLights"
        }
    ]
}
```

> **Note**
>
> 此操作具有最终一致性。换句话说，事物组的更改可能不会立即反映出来。

列出事物的组

您可以使用 ListThingGroupsForThing 命令列出事物所属的直接组：

```
$ aws iot list-thing-groups-for-thing --thing-name MyLightBulb
```

ListThingGroupsForThing 命令会返回此事物所属的直接事物组列表：

```
{
    "thingGroups":[
        {
            "groupName": "LightBulbs",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs"
        },
        {
            "groupName": "RedLights",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/RedLights"
        },
        {
            "groupName": "ReplaceableObjects",
            "groupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
ReplaceableObjects"
        }
    ]
}
```

更新静态事物组

您可以使用 UpdateThingGroup 命令更新静态事物组的属性：

```
$ aws iot update-thing-group --thing-group-name "LightBulbs" --thing-group-properties
 "thingGroupDescription=\"this is a test group\", attributePayload=\"{\"attributes
\"={\"Owner\"=\"150\",\"modelNames\"=\"456\"}}"
```

UpdateThingGroup 命令返回一个响应，其中包含该组更新后的版本号：

```
{
    "version": 4
}
```

> **Note**
>
> 事物可以拥有的属性数量受到限制。

删除事物组

要删除事物组，请使用 DeleteThingGroup 命令：

```
$ aws iot delete-thing-group --thing-group-name "RedLights"
```

DeleteThingGroup 命令不会生成任何输出。

> **Important**
>
> 如果您尝试删除具有子事物组的事物组，则会收到错误：
>
> ```
> A client error (InvalidRequestException) occurred when calling the
>  DeleteThingGroup
> operation: Cannot delete thing group : RedLights when there are still child
>  groups attached to it.
> ```
>
> 在删除群组之前，请先删除所有子群组。

您可以删除具有子事物的组，但由组中的成员资格授予事物的任何权限将不再适用。删除附加了策略的组之前，请认真检查，确保移除这些权限不会导致组中的事物无法正常运行。此外，当云端记录更新时，显示某件事属于哪些群组的命令（例如ListGroupsForThing）可能会继续显示该群组。

将策略附加到静态事物组

您可以使用 AttachPolicy 命令将策略附加到静态事物组，这样可以扩展到该组中的所有事物及其所有子组中的事物：

```
$ aws iot attach-policy \
  --target "arn:aws:iot:us-west-2:123456789012:thinggroup/LightBulbs" \
  --policy-name "myLightBulbPolicy"
```

AttachPolicy 命令不会生成任何输出

> **Important**
>
> 您可以向组附加最多 2 个策略。

> **Note**
>
> 我们建议不要在您的策略名称中使用个人身份信息。

`--target` 参数可以是事物组 ARN (如上所示)、证书 ARN 或 Amazon Cognito Identity。有关策略、证书和身份验证的更多信息，请参阅身份验证。

有关更多信息，请参阅 AWS IoT Core 策略。

从静态事物组分离策略

您可以使用 DetachPolicy 命令从组中分离策略，这样会更进一步地从该组中的所有事物及其所有子组中的事物分离：

```
$ aws iot detach-policy --target "arn:aws:iot:us-west-2:123456789012:thinggroup/
LightBulbs" --policy-name "myLightBulbPolicy"
```

DetachPolicy 命令不会生成任何输出。

列出附加到静态事物组的策略

您可以使用 ListAttachedPolicies 命令列出附加到静态事物组的策略：

```
$ aws iot list-attached-policies --target "arn:aws:iot:us-
west-2:123456789012:thinggroup/RedLights"
```

`--target` 参数可以是事物组 ARN (如上所示)、证书 ARN 或 Amazon Cognito 身份。

添加可选的 `--recursive` 参数可以包括附加到组的父组的所有策略。

ListAttachedPolicies 命令返回策略列表：

```
{
    "policies": [
        "MyLightBulbPolicy"
        ...
    ]
}
```

列出策略的组

您可以使用 ListTargetsForPolicy 命令列出策略附加到的目标，包括任何组：

```
$ aws iot list-targets-for-policy --policy-name "MyLightBulbPolicy"
```

添加可选的 `--page-size` *`number`* 参数以指定为每个查询返回的最大结果数，并在后续调用上添加 `--marker` *`string`* 参数以检索下一组结果 (如果有)。

ListTargetsForPolicy 命令返回目标的列表以及用于检索更多结果的令牌：

```
{
    "nextMarker": "string",
    "targets": [ "string" ... ]
}
```

获取事物的有效策略

您可以使用 GetEffectivePolicies 命令列出对事物生效的策略，包括附加到事物所属任意组的策略 (不论该组是直接父组还是间接原级)：

```
$ aws iot get-effective-policies \
  --thing-name "MyLightBulb" \
  --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

使用 `--principal` 参数指定附加到事物的证书的 ARN。如果您使用的是 Amazon Cognito 身份验证，请使用 `--cognito-identity-pool-id` 参数，并（可选）添加 `--principal` 参数来指定特定的 Amazon Cognito 身份。如果您仅指定 `--cognito-identity-pool-id`，则返回未经身份验证用户的与该身份池角色关联的策略。如果同时使用两个参数，则返回经过身份验证用户的与该身份池角色关联的策略。

`--thing-name` 参数是可选的，可用于替代 `--principal` 参数。在使用时，将返回附加到事物所属任意组的策略以及附加到这些组的任意父组（向上直至层次结构中的根组）的策略。

GetEffectivePolicies 命令返回策略列表：

```
{
```

```
    "effectivePolicies": [
        {
            "policyArn": "string",
            "policyDocument": "string",
            "policyName": "string"
        }
        ...
    ]
}
```

测试 MQTT 操作的授权

您可以使用 TestAuthorization 命令测试某个事物是否允许 MQTT 操作 (Publish、Subscribe)：

```
aws iot test-authorization \
    --principal "arn:aws:iot:us-east-1:123456789012:cert/
a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847" \
    --auth-infos "{\"actionType\": \"PUBLISH\", \"resources\": [ \"arn:aws:iot:us-
east-1:123456789012:topic/my/topic\"]}"
```

使用 `--principal` 参数指定附加到事物的证书的 ARN。如果使用 Amazon Cognito Identity 身份验证，请指定 Cognito Identity 作为 `--principal` 并且/或者使用 `--cognito-identity-pool-id` 参数。如果您仅指定 `--cognito-identity-pool-id`，则考虑未经身份验证用户的与该身份池角色关联的策略。如果同时使用两个参数，则考虑经过身份验证用户的与该身份池角色关联的策略。

通过列出资源和操作类型的组，并且后跟 `--auth-infos` 参数，
来指定一个或多个要测试的 MQTT 操作。`actionType` 字段应包
含“PUBLISH”、“SUBSCRIBE”、“RECEIVE”或“CONNECT”。`resources` 字段应包含资源 ARN 的列表。请参阅AWS IoT Core 政策了解更多信息。

您可以通过将这些参数与 `--policy-names-to-add` 参数一起指定，测试添加策略的效果。或者，您可以通过 `--policy-names-to-skip` 参数来测试移除这些策略的效果。

您可以使用可选的 `--client-id` 参数来进一步优化结果。

TestAuthorization 命令返回您指定的每组 `--auth-infos` 查询允许或拒绝的操作的详细信息：

```
{
    "authResults": [
        {
```

```
            "allowed": {
                "policies": [
                    {
                        "policyArn": "string",
                        "policyName": "string"
                    }
                ]
            },
            "authDecision": "string",
            "authInfo": {
                "actionType": "string",
                "resources": [ "string" ]
            },
            "denied": {
                "explicitDeny": {
                    "policies": [
                        {
                            "policyArn": "string",
                            "policyName": "string"
                        }
                    ]
                },
                "implicitDeny": {
                    "policies": [
                        {
                            "policyArn": "string",
                            "policyName": "string"
                        }
                    ]
                }
            },
            "missingContextValues": [ "string" ]
        }
    ]
}
```

动态事物组

动态事物组是根据注册表中的特定搜索查询创建的。设备连接、设备影子创建和 AWS IoT Device Defender 违规数据等搜索查询参数支持此功能。动态事物组需要启用队列索引才能对设备的数据进行索引、搜索和聚合。在创建动态事物组之前，您可以使用队列索引搜索查询来预览动态事物组中的事物。有关更多信息，请参阅 机群索引 和 查询语法。

> **Note**
>
> 动态事物组操作按注册表操作进行计量。有关更多信息，请参阅 AWS IoT Core 其它计量详细信息。

动态事物组与静态事物组在以下方面不同：

- 未显式定义事物成员资格。要创建动态事物组，请定义搜索查询字符串以确定组成员资格。
- 动态事物组无法成为层次结构的一部分。
- 不能将策略应用于动态事物组。
- 您使用一组不同的命令来创建、更新和删除动态事物组。对于所有其他操作，您可以对两种类型的事物组使用相同的命令。
- 每个动态组的数量 AWS 账户 是有限的。

- 请勿在事物组名称中使用个人身份信息。事物组名称可以出现在未加密的通信和报告中。

有关静态事物组的更多信息，请参阅静态事物组。

动态事物组的用例

您可以将动态事物组用于以下用例：

将动态事物组指定为任务的目标

通过创建以动态事物组为目标的连续作业，您可以在设备满足所需标准时自动将其定位为目标。标准可以是连接状态或存储在注册表或影子中的任何标准，例如软件版本或型号。如果某件事物未出现在动态事物组中，则它不会从该任务中接收任务文档。

例如，如果您的设备群需要固件更新以最大限度地降低更新过程中中断的风险，而您只想在电池续航时间大于 80% 的设备上更新固件。您可以创建一个名为 80 的动态事物组PercentBatteryLife ，该组仅包含电池续航时间超过 80% 的设备，并将其用作工作的目标。只有符合电池寿命标准的设备才能收到固件更新。当设备达到 80% 的电池续航时间标准时，它们会自动添加到动态事物组中，并将收到固件更新。

您可能还有多个设备型号，其固件或操作系统各不相同，因此需要不同版本的新软件更新。对于具有连续作业的动态组，这是最常见的用例，您可以在其中为每种设备型号、固件和操作系统组合创建一个动

态组。然后，您可以为每个动态组设置连续作业，以便在设备根据定义的标准自动成为这些组成员时推送软件更新。

有关将事物组指定为任务目标的更多信息，请参阅CreateJob。

使用动态群组成员资格更改来执行所需的操作

每次将设备添加到动态事物组或从动态事物组中移除时，都会作为注册表事件更新的一部分向 MQTT 主题发送通知。您可以配置AWS IoT Core 规则，根据动态群组成员资格更新与 AWS 服务进行交互并采取所需的操作。示例操作包括写入 Amazon DynamoDB、调用 Lambda 函数或向亚马逊 SNS 发送通知。

将设备添加到动态事物组以进行自动违规检测

AWS IoT Device Defender 检测客户可以在动态事物组上定义安全配置文件。动态事物组中定义的安全配置文件会自动检测该组的设备是否存在违规行为。

在动态事物组上设置日志级别，以通过精细的日志记录来观察设备

您可以为动态事物组指定日志级别。如果您只想为满足特定条件的设备自定义日志级别和详细信息，则此功能非常有用。例如，如果您怀疑具有特定固件版本的设备在特定规则的已发布主题上导致错误，则可能需要设置详细的日志记录来调试这些问题。在这种情况下，您可以为所有具有此固件版本的设备创建一个动态组，我们假设该固件版本存储为注册表属性或设备影子中。然后，您可以设置调试级别，并将日志目标定义为该动态事物组。有关精细日志记录的更多信息，请参阅AWS IoT 使用日志进行 CloudWatch 监控。有关如何为特定事物组指定日志记录级别的更多信息，请参阅配置特定于资源的登录。 AWS IoT

创建动态事物组

使用 CreateDynamicThingGroup 命令创建动态事物组。要为 80 PercentBatteryLife 场景创建动态事物组，请使用 create-dynamic-thing-group CLI 命令：

```
$ aws iot create-dynamic-thing-group --thing-group-name "80PercentBatteryLife" --query-string "attributes.batterylife80"
```

> **Note**
>
> 请勿在动态事物组名称中使用个人身份信息。

该CreateDynamicThingGroup命令返回响应。响应包含您的事物组的索引名称、查询字符串、查询版本、事物组名称、事物组 ID 和亚马逊资源名称 (ARN)：

```
{
    "indexName": "AWS_Things",
    "queryVersion": "2017-09-30",
    "thingGroupName": "80PercentBatteryLife",
    "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/80PercentBatteryLife",
    "queryString": "attributes.batterylife80\n",
    "thingGroupId": "abcdefgh12345678ijklmnop12345678qrstuvwx"
}
```

动态事物组的创建不会立刻发生。动态事物组回填需要一些时间才能完成。创建动态事物组时，该组的状态设置为BUILDING。在回填完成后，状态变为 ACTIVE。要检查动态事物组的状态，请使用DescribeThing组命令。

描述动态事物组

使用 DescribeThingGroup 命令获取有关动态事物组的信息：

```
$ aws iot describe-thing-group --thing-group-name "80PercentBatteryLife"
```

DescribeThingGroup 命令返回有关指定组的信息：

```
{
    "status": "ACTIVE",
    "indexName": "AWS_Things",
    "thingGroupName": "80PercentBatteryLife",
    "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/80PercentBatteryLife",
    "queryString": "attributes.batterylife80\n",
    "version": 1,
    "thingGroupMetadata": {
        "creationDate": 1548716921.289
    },
    "thingGroupProperties": {},
    "queryVersion": "2017-09-30",
    "thingGroupId": "84dd9b5b-2b98-4c65-84e4-be0e1ecf4fd8"
}
```

在动态事物组DescribeThingGroup上运行会返回特定于动态事物组的属性。返回属性的示例是查询字符串和状态。

动态事物组的状态可以采用以下值：

`ACTIVE`

动态事物组已准备就绪，可供使用。

`BUILDING`

正在创建动态事物组，并且正在处理事物成员资格。

`REBUILDING`

正在按照组的搜索查询的调整更新动态事物组的成员资格。

> **Note**
>
> 创建动态事物组后，无论其状态如何，都要使用它。只有 `ACTIVE` 状态的动态事物组包括与该动态事物组的搜索查询匹配的所有事物。`BUILDING` 和 `REBUILDING` 状态的动态事物组可能未包括与搜索查询匹配的所有事物。

更新动态事物组

使用 UpdateDynamicThingGroup 命令更新动态事物组的属性，包括组的搜索查询。以下命令更新两个属性。一个是事物组描述，另一个是将成员资格标准更改为电池寿命 > 85 的查询字符串：

```
$ aws iot update-dynamic-thing-group --thing-group-name "80PercentBatteryLife" --thing-
group-properties "thingGroupDescription=\"This thing group contains devices with a
 battery life greater than 85 percent.\"" --query-string "attributes.batterylife85"
```

UpdateDynamicThingGroup 命令返回一个响应，其中包含该组更新后的版本号：

```
{
    "version": 2
}
```

动态事物组的更新不会立即发生。动态事物组回填需要一些时间才能完成。更新动态事物组时，群组的状态会更改为，REBUILDING而群组会更新其成员资格。在回填完成后，状态变为 ACTIVE。要检查动态事物组的状态，请使用DescribeThing组命令。

删除动态事物组

使用 DeleteDynamicThingGroup 命令删除动态事物组：

```
$ aws iot delete-dynamic-thing-group --thing-group-name "80PercentBatteryLife"
```

DeleteDynamicThingGroup 命令不会生成任何输出。

在更新云中的记录时，显示事物所属的组的命令（例如，ListGroupsForThing）可能会继续显示该组。

动态和静态事物组限制

动态事物组和静态事物组有以下限制：

- 事物组可以拥有的属性数量是有限的。
- 一个事物可以属于的组数受到限制。
- 您无法重命名事物组。
- 事物组名称不能包含国际字符，如 û、é 和 ñ。

动态事物组限制

动态事物组具有以下限制：

机群索引

启用队列索引服务后，您可以对设备群执行搜索查询。队列索引回填完成后，您可以创建和管理动态事物组。回填过程的完成时间直接受注册到的设备群规模的影响。 AWS Cloud在为动态事物组启用实例集索引服务后，您将无法禁用它，直到您删除所有动态事物组。

> **Note**
>
> 如果您有权查询实例集索引，可以访问整个实例集的事物数据。

动态事物组的数量受到限制

动态事物组的数量是有限的。

成功的命令可能记录错误

创建或更新动态事物组时，有些事物可能符合包含在动态事物组中的条件，但它们并未添加到动态事物组中。这种情况将导致在记录错误和生成指标的同时成功执行创建或更新命令。`AddThingToDynamicThingGroupsFailed`一个指标可以代表多个日志条目。

发生以下情况时，将在 CloudWatch 日志中创建错误日志条目：

- 无法将符合条件的事物添加到动态事物组中。
- 将事物从动态事物组中移除，然后将其添加到另一个组。

当某件事有资格添加到动态事物组时，请考虑以下几点：

- 事物是否已经位于尽可能多的组中了？（请参阅限制）
  - 否：该事物被添加到动态事物组中。
  - 是：该事物是任何动态事物组的成员吗？
    - 否：无法将该事物添加到动态事物组，记录错误，并生成 `AddThingToDynamicThingGroupsFailed` 指标。
    - 是：要加入的动态事物组是否早于该事物已成为其成员的任何动态事物组？
      - 否：无法将该事物添加到动态事物组，记录错误，并生成 `AddThingToDynamicThingGroupsFailed` 指标。
      - 是：将事物从最新的动态事物组中移除，记录错误，然后将该事物添加到动态事物组中。这会生成一个错误，并针对从中删除该事物的动态事物组生成一个 `AddThingToDynamicThingGroupsFailed` 指标。

当动态事物组中的某件事不再符合搜索查询时，该事物将从动态事物组中移除。同样，当更新事物以满足动态事物组的搜索查询时，该事物就会如前所述，添加到该组中。这些添加和删除操作是正常的，不会产生错误记录条目。

在启用 `overrideDynamicGroups` 的情况下，静态组优先于动态组

一个事物可以属于的组数受到限制。当您使用AddThingToThing组或UpdateThingGroupsFor事物命令更新事物成员资格时，添加`--overrideDynamicGroups`参数会使静态事物组优先于动态事物组。

向静态事物组添加事物时，请考虑以下几点：

- 事物是否已经属于最大数量的组？
  - 否：该事物被添加到静态事物组中。
  - 是：该事物是否在任何动态组中？
    - 否：该事物无法添加到该事物组中。该命令引发异常。
    - 是：是否已启用 --overrideDynamicGroups？
      - 否：该事物无法添加到该事物组中。该命令引发异常。
      - 是：从最近创建的动态事物组中删除该事物，记录错误，并针对从中删除该事物的动态事物组生成 AddThingToDynamicThingGroupsFailed 指标。然后，该事物被添加到静态事物组。

旧动态事物组优先于新动态事物组

一个事物可以属于的组数受到限制。当创建或更新操作为某事物创建了额外的组资格并且该事物已达到其组限制时，可能会从另一个动态事物组中移除以启用此添加。有关如何发生这种情况的更多信息，请参阅成功的命令可能记录错误和在启用 overrideDynamicGroups 的情况下，静态组优先于动态组了解示例。

从动态事物组中移除事物时，会记录错误并引发事件。

无法将策略应用于动态事物组

尝试将策略应用于动态事物组会生成异常。

动态事物组成员资格具有最终一致性

只为注册表评估事物的最终状态。如果状态快速更新，则可跳过中间状态。避免将规则或任务与其成员资格取决于中间状态的动态事物组相关联。

为资源添加 AWS IoT 标签

为了帮助您管理和组织事物组、事物类型、主题规则、任务、计划的审核和安全配置文件，您可以选择将自己的元数据以标签的形式分配给其中每个资源。本部分介绍标签并说明如何创建标签。

为了帮助您管理与事物相关的成本，您可以创建包含事物的账单组。然后，可以将包含您的元数据的标签分配给其中每个账单组。本部分还讨论账单组以及可用于创建和管理它们的命令。

标签基本知识

您可以使用标签以不同的方式对 AWS IoT 资源进行分类（例如，按用途、所有者或环境）。这在您具有相同类型的许多资源时会很有用 — 您可以根据分配给资源的标签快速识别资源。每个标签都包含您定义的一个键和一个可选值。例如，您可以为事物类型定义一组标签来帮助您按类型跟踪设备。我们建议您为每类资源创建一组可满足您的需求的标签键。使用一组连续的标签键，管理 资源时会更加轻松。

您可以根据添加或应用的标签搜索和筛选资源。您还可以使用账单组标签对成本进行分类和跟踪。您还可以使用标签控制对资源的访问，如在 IAM 策略中使用标签中所述。

为便于使用， AWS 管理控制台中的标签编辑器提供了一种集中、统一的方式来创建和管理标签。有关更多信息，请参阅使用AWS 管理控制台中的使用标签编辑器。

您也可以使用 AWS CLI 和 AWS IoT API 处理标签。当您在以下命令中使用 Tags 字段创建标签时，可以将标签与事物组、事物类型、主题规则、任务、安全配置文件、策略、账单组以及与事物关联的软件包和版本相关联：

- CreateBillingGroup
- CreateDestination
- CreateDeviceProfile
- CreateDynamicThingGroup
- CreateJob
- CreateOTAUpdate
- CreatePolicy
- CreateScheduledAudit
- CreateSecurityProfile

- CreateServiceProfile
- CreateStream
- CreateThingGroup
- CreateThingType
- CreateTopicRule
- CreateWirelessGateway
- CreateWirelessDevice

您可以使用以下命令为支持标记的现有资源添加、修改或删除标签：

- TagResource
- ListTagsForResource
- UntagResource

您可以修改标签的密钥和值，还可以随时删除资源的标签。您可以将标签的值设为空的字符串，但是不能将其设为空值。如果添加的标签的键与该资源上现有标签的键相同，新值就会覆盖旧值。如果删除资源，则所有与资源相关的标签都将被删除。

标签限制

下面是适用于标签的基本限制：

- 每个资源的最大标签数 - 50
- 最大密钥长度 - 127 个 Unicode 字符（采用 UTF-8 格式）
- 最大值长度 - 255 个 Unicode 字符（采用 UTF-8 格式）
- 标签键和值区分大小写。
- 请勿在标签名称或值中使用 `aws:` 前缀。它是保留供 AWS 使用的。您无法编辑或删除带此前缀的标签名称或值。具有此前缀的标签不计入每个资源的标签数限制。
- 如果在多个服务和资源中使用您的标记方案，请记住，其他服务可能对允许使用的字符有限制。允许使用的字符包括：可用 UTF-8 格式表示的字母、空格和数字以及以下特殊字符：+ - = . _ : / @。

在 IAM 策略中使用标签

可以在用于 AWS IoT API 操作的 IAM 策略中应用基于标签的资源级权限。这可让您更好地控制用户可创建、修改或使用哪些资源。在 IAM policy 中将 Condition 元素（也称作 Condition 块）与以下条件上下文键和值结合使用来基于资源标签控制用户访问（权限）：

- 使用 `aws:ResourceTag/tag-key: tag-value` 可允许或拒绝带特定标签的资源上的用户操作。
- 使用 `aws:RequestTag/tag-key: tag-value` 可要求在发出创建或修改允许标签的资源的 API 请求时使用（或不使用）特定标签。
- 使用 `aws:TagKeys: [tag-key, ...]` 可要求在发出创建或修改允许标签的资源的 API 请求时使用（或不使用）一组特定标签键。

> **Note**
>
> IAM 策略中的条件上下文密钥和值仅适用于那些以可标记的资源的标识符为必填参数的 AWS IoT 操作。例如，根据条件上下文键和值，DescribeEndpoint不允许或拒绝使用，因为此请求中未引用任何可标记的资源（事物组、事物类型、主题规则、作业或安全配置文件）。有关可标记 AWS IoT 资源及其支持的条件键的更多信息，请阅读的操作、资源和条件键。 AWS IoT

有关使用标签的更多信息，请参阅《AWS Identity and Access Management 用户指南》中的使用标签控制访问权限。该指南的 IAM JSON 策略参考一部分包含 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。

以下示例策略应用 ThingGroup 操作两个基于标签的限制。受此策略限制的 IAM 用户：

- 无法创建标签 “env=prod” 的事物组（在示例中，请参阅行 `"aws:RequestTag/env" : "prod"`）。
- 无法修改或访问具有现有标签“env=prod”的资源（在示例中，请参阅行 `"aws:ResourceTag/env" : "prod"`）。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "iot:CreateThingGroup",
```

```
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:UpdateThingGroup"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:UpdateThingGroup"
      ],
      "Resource": "*"
    }
  ]
}
```

您还可以为给定标签键指定多个标签值，方法是将它们括在列表中，如下所示：

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

> **Note**
>
> 如果您基于标签允许或拒绝用户访问资源，则必须考虑显式拒绝用户对相同资源添加或删除这些标签的能力。否则，用户可能通过修改资源标签来绕过您的限制并获得资源访问权限。

账单组

AWS IoT 不允许你直接将标签应用于单个内容，但它确实允许你将内容放在账单组中并对这些内容应用标签。对于 AWS IoT，基于标签的成本和使用量数据仅限于账单组。

AWS IoT Core 对于 LoRa WAN 资源，例如无线设备和网关，无法添加到账单组。但是，它们可以与 AWS IoT 事物关联，可以将其添加到账单组中。

提供了以下命令：

- AddThingToBillingGroup向账单组添加内容。
- CreateBillingGroup 创建账单组。
- DeleteBillingGroup 删除账单组。
- DescribeBillingGroup返回有关账单组的信息。
- ListBillingGroups列出了您创建的账单组。
- ListThingsInBillingGroup列出了您已添加到给定账单组的内容。
- RemoveThingFromBillingGroup从账单组中移除给定内容。
- UpdateBillingGroup更新账单组的相关信息。
- CreateThing允许您在创建事物时为其指定账单组。
- DescribeThing返回事物的描述，包括该事物所属的账单组（如果有）。

AWS IoT Wireless API 提供这些操作来将无线设备和网关与 AWS IoT 事物关联起来。

- AssociateWirelessDeviceWithThing
- AssociateWirelessGatewayWithThing

查看成本分配和使用率数据

您可以使用账单组标签对成本进行分类和跟踪。当您将标签应用于账单组（以及它们所包含的内容）时，AWS 会生成以逗号分隔值 (CSV) 文件形式的成本分配报告，其中包含按标签汇总的使用量和成

本。您可以设置代表业务类别（例如成本中心、应用程序名称或所有者）的标签，以便整理多种服务的成本。有关将标签用于成本分配的更多信息，请参阅《AWS 账单和成本管理用户指南》中的使用成本分配标签。

> **Note**
>
> 要准确地将使用率和成本数据与您放入账单组的那些事物相关联，每个设备或应用程序必须：
>
> - 注册为事物 AWS IoT. 有关更多信息，请参阅 使用管理设备 AWS IoT。
> - 仅使用事物的名称作为客户端 ID 通过 MQTT 连接到 AWS IoT 消息代理。有关更多信息，请参阅 the section called “设备通信协议”。
> - 使用与事物关联的客户端证书进行身份验证。

为账单组提供了以下定价维度（基于与账单组关联的事物的活动）：

- 连接（基于要连接的用作客户端 ID 的事物名称）。
- 消息收发（基于事物的入站和出站消息；仅限 MQTT）。
- 影子操作（基于其消息触发了影子更新的事物）。
- 触发的规则（基于其入站消息触发规则的事物；不适用于 MQTT 生命周期事件触发的那些规则）。
- 事物索引更新（基于添加到索引的事物）。
- 远程操作（基于已更新的事物）。
- AWS IoT Device Defender 检测报告（基于报告其活动的事物）。

（为账单组报告的）基于标签的成本和使用率数据不会反映以下活动：

- 设备注册表操作（包括更新事物、事物组和事物类型）。有关更多信息，请参阅 使用管理设备 AWS IoT）。
- 事物组索引更新（在添加事物组时）。
- 索引搜索查询。
- 设备预调配.
- AWS IoT Device Defender 审计报告。

安全性 AWS IoT

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的 安全性和云中 的安全性：

- 云安全 — AWS 负责保护在 AWS 云中运行 AWS 服务的基础架构。 AWS 还为您提供可以安全使用的服务。作为 AWS 合规性计划的一部分，第三方审核人员将定期测试和验证安全性的有效性。要了解适用的合规计划 AWS IoT，请参阅按合规计划划分的范围内的AWS 服务。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其它因素负责，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。

本文档可帮助您了解在使用时如何应用分担责任模型 AWS IoT。以下主题向您介绍如何进行配置 AWS IoT 以满足您的安全和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 AWS IoT 资源。

主题

AWS IoT 安全

每个连接的设备或客户端都必须具有与 AWS IoT进行交互所需的凭证。所有进出 AWS IoT 流量均通过传输层安全 (TLS) 安全发送。 AWS 云安全机制可在数据与其他 AWS 服务 AWS IoT 之间移动时对其进行保护。

- 您负责管理 AWS IoT中的设备证书（X.509 证书、 AWS 凭证、Amazon Cognito 身份、联合身份或自定义身份验证令牌）和策略。有关更多信息，请参阅 中的密钥管理 AWS IoT。您负责将唯一身份分配给每台设备并管理每个设备或每组设备的权限。
- 您的设备 AWS IoT 使用 X.509 证书或亚马逊 Cognito 身份通过安全的 TLS 连接进行连接。在研发期间，对于某些调用或使用 API 的应用程序 WebSockets，您还可以使用 IAM 用户和群组或自定义身份验证令牌进行身份验证。有关更多信息，请参阅 IAM 用户、组和角色。
- 使用 AWS IoT 身份验证时，消息代理负责对您的设备进行身份验证，安全地摄取设备数据，并使用策略授予或拒绝您为设备指定的访问权限。 AWS IoT
- 使用自定义身份验证时，自定义授权机构负责对您的设备进行身份验证，并使用 AWS IoT 或 IAM 策略授予或拒绝您为设备指定的访问权限。
- AWS IoT 规则引擎根据您定义的规则将设备数据转发到其他设备或其他 AWS 服务。它用于 AWS Identity and Access Management 将数据安全地传输到其最终目的地。有关更多信息，请参阅 的身份和访问管理 AWS IoT。

身份验证

身份验证是用于验证客户端或服务器的身份的机制。服务器身份验证是设备或其他客户端确保与实际 AWS IoT 端点通信的过程。客户端身份验证是设备或其他客户端对自己进行身份验证的过程 AWS IoT。

AWS 培训和认证

参加以下课程，在《深入了解身份验证和授权》中 AWS IoT学习有关 AWS IoT 身份验证的信息。

X.509 证书概览

X.509 证书是一个数字证书，它使用 X.509 公有密钥基础设施标准将公有密钥与证书中包含的身份相关联。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。X.509 证书链既用于客户端进行的服务器身份验证，又用于服务器进行的客户端身份验证。

服务器身份验证

当您的设备或其他客户端尝试连接时 AWS IoT Core， AWS IoT Core 服务器将发送一个 X.509 证书，您的设备将使用该证书对服务器进行身份验证。身份验证通过验证 X.509 证书链在 TLS 层进行。这与浏览器在您访问 HTTPS URL 时使用的方法相同。如果要使用您自己的证书颁发机构提供的证书，请参阅管理 CA 证书。

当您的设备或其他客户端与 AWS IoT Core 端点建立 TLS 连接时，会 AWS IoT Core 显示一个证书链，设备使用该证书链来验证它们是否正在 AWS IoT Core 与之通信，而不是其他服务器模仿 AWS IoT Core。呈现的链取决于设备连接的端点类型和客户端在 TLS 握手期间 AWS IoT Core 协商的密码套件的组合。

端点类型

AWS IoT Core 支持两种不同的数据端点类型，`iot:Data`和`iot:Data-ATS`。 `iot:Data`端点出示由 3 VeriSign 类公共主要 G5 根 CA 证书签名的证书。 `iot:Data-ATS`终端节点出示由亚马逊信任服务 CA 签署的服务器证书。

ATS 终端节点提供的证书由 Starfield 进行交叉签名。某些 TLS 客户端实现要求验证信任根，并要求将 Starfield CA 证书安装到客户端的信任存储中。

> **Warning**
>
> 建议不要使用对整个证书（包括颁发者名称等）进行哈希处理的证书固定方法，因为这将导致证书验证失败，因为我们提供的 ATS 证书由 Starfield 进行交叉签名并具有其它颁发者名称。

> **Important**
>
> 除非您的设备需要 Symantec 或 Verisign CA 证书，否则请使用 `iot:Data-ATS` 终端节点。Symantec 和 Verisign 证书已被弃用，并且大多数 Web 浏览器不再支持这两类证书。

您可以使用 `describe-endpoint` 命令创建 ATS 终端节点。

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

`describe-endpoint` 命令采用以下格式返回终端节点。

```
account-specific-prefix.iot.your-region.amazonaws.com
```

> **Note**
>
> 首次调用 `describe-endpoint` 时，会创建一个终端节点。对 `describe-endpoint` 的所有后续调用将返回同一终端节点。

为了向后兼容， AWS IoT Core 仍支持赛门铁克端点。有关更多信息，请参阅 AWS IoT Core 如何帮助客户应对即将发生的 Symantec 证书颁发机构不受信任情况。ATS 终端节点上运行的设备与同一账户中 Symantec 终端节点上运行的设备可完全互操作，并且不需要任何重新注册。

> **Note**
>
> 要在 AWS IoT Core 控制台中查看您的`iot:Data-ATS`终端节点，请选择设置。控制台仅显示 `iot:Data-ATS` 终端节点。默认情况下，`describe-endpoint` 命令显示 `iot:Data` 终端节点以实现向后兼容。要查看 `iot:Data-ATS` 终端节点，请指定 `--endpointType` 参数，如上例所示。

`IotDataPlaneClient`使用适用于 Java 的 AWS SDK 创建

默认情况下，AWS SDK for Java - 版本 2 使用 `IotDataPlaneClient` 终端节点创建 `iot:Data`。要创建使用 `iot:Data-ATS` 终端节点的客户端，您必须执行以下操作。

- 使用 DescribeEndpointAPI 创建`iot:Data-ATS`终端节点。
- 在创建 `IotDataPlaneClient` 时指定该终端节点。

以下示例执行这两项操作。

```
public void setup() throws Exception {
        IotClient client =
 IotClient.builder().credentialsProvider(CREDENTIALS_PROVIDER_CHAIN).region(Region.US_EAST_1).b
        String endpoint = client.describeEndpoint(r -> r.endpointType("iot:Data-
ATS")).endpointAddress();
        iot = IotDataPlaneClient.builder()
                                .credentialsProvider(CREDENTIALS_PROVIDER_CHAIN)
                                .endpointOverride(URI.create("https://" + endpoint))
                                .region(Region.US_EAST_1)
                                .build();
}
```

用于服务器身份验证的 CA 证书

根据您使用的数据端点类型和协商的密码套件， AWS IoT Core 服务器身份验证证书由以下根 CA 证书之一签名：

Amazon Trust Services 终端节点（首选）

> **Note**
>
> 您可能需要右键单击这些链接，然后选择 Save link as... (将链接另存为...) 将这些证书另存为文件。

- RSA 2048 位密钥：Amazon Root CA 1。
- RSA 4096 位密钥：Amazon Root CA 2。留待将来使用。
- ECC 256 位密钥：Amazon Root CA 3。
- ECC 384 位密钥：Amazon Root CA 4。留待将来使用。

这些证书都由 Starfield 根 CA 证书进行交叉签名。自 2018 年 5 月 9 日在亚太 AWS IoT Core 地区（孟买）地区推出以来，所有新地区仅提供 ATS 证书。 AWS IoT Core

VeriSign 终端节点（旧版）

- RSA 2048 位密钥：3 VeriSign 类公共主要 G5 根 CA 证书

服务器身份验证指南

有很多变量会影响设备验证 AWS IoT Core 服务器身份验证证书的能力。例如，设备的内存可能太有限，无法容纳所有可能的根 CA 证书，或者设备可能会实施非标准的证书验证方法。由于这些原因，我们建议遵循以下准则：

- 我们建议您使用 ATS 终端节点并安装所有受支持的 Amazon Root CA 证书。
- 如果您无法在设备上存储所有这些证书，并且您的设备未使用基于 ECC 的验证，则可以省略 Amazon Root CA 3 和 Amazon Root CA 4 ECC 证书。如果您的设备未实施基于 RSA 的证书验证，则可以省略 Amazon Root CA 1 和 Amazon Root CA 2 RSA 证书。您可能需要右键单击这些链接，然后选择 Save link as... (将链接另存为...) 将这些证书另存为文件。
- 如果在连接到 ATS 终端节点时遇到服务器证书验证问题，请尝试将相关的交叉签名的 Amazon Root CA 证书添加到您的信任存储中。您可能需要右键单击这些链接，然后选择 Save link as... (将链接另存为...) 将这些证书另存为文件。
  - 交叉签名 Amazon Root CA 1
  - 交叉签名的 Amazon Root CA 2 - 预留供将来使用。
  - 交叉签名 Amazon Root CA 3
  - 交叉签名的 Amazon Root CA 4 - 预留供将来使用。
- 如果您遇到服务器证书验证问题，则您的设备可能需要明确信任根 CA。尝试将 Starfield Root CA Certificate 添加到您的信任存储。
- 如果您在执行上述步骤后仍遇到问题，请联系AWS 开发人员支持。

> **Note**
>
> CA 证书具有一个过期日期，在该日期后，这些证书将无法用于验证服务器的证书。可能必须在 CA 证书的过期日期前替换这些证书。请确保可以更新所有设备或客户端上的根 CA 证书，以确保持续的连接并保持最新的安全最佳实践。

> **Note**
>
> 在设备代码 AWS IoT Core 中连接时，请将证书传递到您用于连接的 API。您使用的 API 因 SDK 而异。有关更多信息，请参阅 AWS IoT Core Device SDK。

客户端身份验证

AWS IoT 支持三种类型的身份主体进行设备或客户端身份验证：

- X.509 客户端证书
- IAM 用户、组和角色
- Amazon Cognito 身份

这些身份可与设备、移动、Web 或桌面应用程序结合使用。用户甚至可以键入 AWS IoT 命令行界面 (CLI) 命令来使用它们。通常， AWS IoT 设备使用 X.509 证书，而移动应用程序使用 Amazon Cognito 身份。Web 应用程序和桌面应用程序使用 IAM 或联合身份。 AWS CLI 命令使用 IAM。有关 IAM 身份的更多信息，请参阅 的身份和访问管理 AWS IoT。

X.509 客户端证书

X.509 证书提供了 AWS IoT 对客户端和设备连接进行身份验证的功能。客户端必须先注册客户证书，AWS IoT 然后才能与之通信 AWS IoT。一个客户端证书可以在同一个区域的多个 AWS 账户 s 中注册 AWS 区域 ，以便于在同一区域的设备 AWS 账户之间移动设备。请参阅在多账户注册中使用多个 AWS 账户 X.509 客户端证书了解更多信息。

我们建议为每个设备或客户端提供一个唯一的证书，以便进行精细的客户端管理操作，包括证书吊销。设备和客户端还必须支持证书轮换和更换，以帮助确保在证书过期时平稳运行。

有关使用 X.509 证书支持多个设备的信息，请参阅 设备预调配 以查看 AWS IoT 支持的不同证书管理和预调配选项。

AWS IoT 支持以下类型的 X.509 客户端证书：

- 由生成的 X.509 证书 AWS IoT
- 由注册的 CA 签署的 X.509 证书。 AWS IoT
- 由未注册到 AWS IoT的 CA 签发的 X.509 证书。

本节介绍如何在 AWS IoT中管理 X.509 证书。您可以使用 AWS IoT 控制台或 AWS CLI 执行以下证书操作：

- 创建 AWS IoT 客户证书
- 创建您自己的客户端证书
- 注册客户端证书
- 激活或停用客户端证书
- 吊销客户端证书

有关执行这些操作的 AWS CLI 命令的更多信息，请参阅 AWS IoT CLI 参考。

使用 X.509 客户端证书

X.509 证书对与的客户端和设备连接进行身份验证。 AWS IoT与其它身份和身份验证机制相比，X.509 证书具有多项优势。有了 X.509 证书，可以将非对称密钥用于设备。例如，您可以将私有密钥刻录到设备上的安全存储中，这样敏感的加密材料永远不会离开设备。X.509 证书可以通过用户名和密码或持有者令牌等其它方案提供更可靠的客户端身份验证，因为私有密钥永远不会离开设备。

AWS IoT 使用 TLS 协议的客户端身份验证模式对客户端证书进行身份验证。TLS 支持适用于多种编程语言和操作系统并且通常用于为数据加密。在 TLS 客户端身份验证中， AWS IoT 请求 X.509 客户端证书并根据证书注册表验证证书的状态。 AWS 账户 然后，它要求客户提供与证书中包含的公钥相对应的私钥的所有权证明。 AWS IoT 要求客户端向传输层安全 (TLS) 协议发送服务器名称指示 (SNI) 扩展。有关配置 SNI 扩展的更多信息，请参阅 运输安全 AWS IoT Core。

为了便于客户机与 AWS IoT 核心进行安全、一致的连接，X.509 客户端证书必须具备以下内容：

- 已在 AWS IoT Core 中注册。有关更多信息，请参阅 注册客户端证书。
- 状态为 ACTIVE。有关更多信息，请参阅 激活或停用客户端证书。
- 尚未达到证书到期日期。

您可以创建使用 Amazon Root CA 的客户端证书，并可以使用您自己的由其它证书颁发机构（CA）签发的客户端证书。有关使用 AWS IoT 控制台创建使用 Amazon 根 CA 的证书的更多信息，请参阅创建 AWS IoT 客户证书。有关使用您自己的 X.509 证书的更多信息，请参阅创建您自己的客户端证书。

对于 CA 证书签发的证书过期日期和时间，将在创建 CA 证书时设置。由此生成的 X.509 证书将于世界标准时间 2049 年 12 月 31 日午夜 AWS IoT 到期（2049-12-31T23：59:59 Z）。

AWS IoT Device Defender 可以对您的 AWS 账户 和支持常见物联网安全最佳实践的设备进行审计。这包括管理由您的 CA 或 Amazon Root CA 签署的 X.509 证书的到期日期。有关管理证书到期日期的更多信息，请参阅设备证书即将到期和 CA 证书即将到期。

在官方 AWS IoT 博客上，《如何使用管理物联网设备证书轮换》中深入探讨了设备证书轮换的管理和安全最佳实践 AWS IoT。

在多账户注册中使用多个 AWS 账户 X.509 客户端证书

多账户注册能够在同一区域或不同区域的 AWS 账户之间移动设备。您可以在预生产账户中注册、测试和配置设备，然后在生产账户中注册并使用相同的设备和设备证书。您也可以在设备上注册客户端证书，或者在没有注册的 CA 的情况下注册设备证书 AWS IoT。有关更多信息，请参阅 Register a client certificate signed by an unregistered CA (CLI)（注册由未注册的 CA 签发的客户端证书（CLI））。

> **Note**
>
> 用于多账户注册的证书在 `iot:Data-ATS`、`iot:Data`（旧式）、`iot:Jobs` 和 `iot:CredentialProvider` 终端节点类型中获得支持。有关 AWS IoT 设备端点的更多信息，请参阅AWS IoT 设备数据和服务端点。

使用多账户注册的设备必须将服务器名称指示 (SNI) 扩展发送到传输层安全 (TLS) 协议，并在连接时在`host_name`字段中提供完整的端点地址。 AWS IoT AWS IoT 使用中的终端节点地址将连接路由`host_name`到正确的 AWS IoT 帐户。未发送 `host_name` 中的有效终端节点地址的现有设备将继续工作，但它们将无法使用需要此信息的特征。有关 SNI 扩展以及如何识别 `host_name` 字段的终端节点地址的更多信息，请参阅运输安全 AWS IoT Core。

使用多账户注册

1. 您可以向 CA 注册设备证书。您可以在 `SNI_ONLY` 模式下在多个账户中注册签名 CA，并使用该 CA 向多个账户注册相同的客户端证书。有关更多信息，请参阅 在 SNI_ONLY 模式下注册 CA 证书（CLI）- 建议。
2. 您可以在没有 CA 的情况下注册设备证书。请参阅 注册由未注册的 CA (CLI) 签发的客户端证书。注册 CA 是可选的。您无需注册签署设备证书的 CA AWS IoT。

支持的证书签名算法 AWS IoT

AWS IoT 支持以下证书签名算法：

- SHA256WITHRSA
- SHA384WITHRSA
- SHA512WITHRSA
- SHA256WITHRSAANDMGF1 (RSASSA-PSS)
- SHA384WITHRSAANDMGF1 (RSASSA-PSS)
- SHA512WITHRSAANDMGF1 (RSASSA-PSS)
- DSA_WITH_SHA256
- ECDSA-WITH-SHA256
- ECDSA-WITH-SHA384
- ECDSA-WITH-SHA512

有关证书身份验证和安全的更多信息，请参阅设备证书密钥质量。

> **Note**
>
> 证书签名请求（CSR）必须包含公有密钥。该密钥可以是长度至少为 2048 位的 RSA 密钥，或者是来自 NIST P-256、NIST P-384 或 NIST P-521 曲线的 ECC 密钥。有关更多信息，请参阅 AWS IoT API 参考指南CreateCertificateFromCsr中的。

支持的密钥算法 AWS IoT

下表显示了如何支持密钥算法：

密钥算法	证书签名算法	TLS 版本	是否支持? 是或否
密钥大小至少为 2048 位的 RSA	全部	TLS 1.2 TLS 1.	是
ECC NIST P-256/P-384/ P-521	全部	TLS 1.2 TLS 1.	是
密钥大小至少为 2048 位的 RSA-PSS	全部	TLS 1.2	否

密钥算法	证书签名算法	TLS 版本	是否支持? 是或否
密钥大小至少为 2048 位的 RSA-PSS	全部	TLS 1.3	是

要使用 CreateCertificatefromCSR 创建证书，您可以使用支持的密钥算法为您的 CSR 生成公钥。要使用RegisterCertificate或RegisterCertificate不使用 CA 注册自己的证书，您可以使用支持的密钥算法为证书生成公钥。

有关更多信息，请参阅安全策略。

创建 AWS IoT 客户证书

AWS IoT 提供由 Amazon 根证书颁发机构 (CA) 签署的客户证书。

本主题介绍如何创建由 Amazon Root 证书颁发机构签发的客户端证书，以及如何下载证书文件。创建客户端证书文件后，您必须在客户端上安装这些文件。

> **Note**
>
> 提供的每个 X.509 客户端证书都 AWS IoT 包含您在创建证书时设置的颁发者和主题属性。只有在创建证书后，证书属性才是不可改变的。

您可以使用 AWS IoT 控制台或创建由 Amazon 根 AWS IoT 证书颁发机构签名的证书。 AWS CLI

创建 AWS IoT 证书（控制台）

使用 AWS IoT 控制台创建 AWS IoT 证书

1. 登录 AWS Management Console 并打开AWS IoT 控制台。
2. 在导航窗格中，依次选择安全、证书和创建。
3. 选择一键式创建证书 (建议)，然后选择创建证书。
4. 从已创建证书页面，将事物、公有密钥和私有密钥的客户端证书文件下载到安全位置。由 AWS IoT 生成的这些证书只能用于 AWS IoT 服务。

   如果您还需要 Amazon Root CA 证书文件，此页面也包含指向可用于下载此文件的页面的链接。
5. 此时客户端证书已创建并注册到 AWS IoT。在客户端中使用证书之前，您必须激活证书。

要立即激活客户端证书，请选择激活。如果您不想立即激活证书，请参阅激活客户端证书（控制台）以了解如何在以后激活证书。

6. 如果要将策略附加到证书，请选择 Attach a policy（附加策略）。

   如果您不希望立即附加策略，请选择 Done（完成）以完成操作。您可以在以后附加策略。

完成此流程后，在客户端上安装证书文件。

创建 AWS IoT 证书 (CLI)

AWS CLI 提供了创建由 Amazon 根证书颁发机构签名的客户证书的create-keys-and-certificate命令。但是，此命令不会下载 Amazon Root CA 证书文件。您可以从用于服务器身份验证的 CA 证书下载 Amazon Root CA 证书文件。

此命令创建私钥、公钥和 X.509 证书文件，并使用注册和激活证书。 AWS IoT

```
aws iot create-keys-and-certificate \
    --set-as-active \
    --certificate-pem-outfile certificate_filename.pem \
    --public-key-outfile public_filename.key \
    --private-key-outfile private_filename.key
```

如果您不想在创建和注册证书时激活证书，则此命令会创建私有密钥、公有密钥和 X.509 证书文件并注册证书，但不会激活它。激活客户端证书 (CLI) 介绍了以后如何激活证书。

```
aws iot create-keys-and-certificate \
    --no-set-as-active \
    --certificate-pem-outfile certificate_filename.pem \
    --public-key-outfile public_filename.key \
    --private-key-outfile private_filename.key
```

在客户端上安装证书文件。

创建您自己的客户端证书

AWS IoT 支持由任何根或中间证书颁发机构 (CA) 签署的客户端证书。 AWS IoT 使用 CA 证书来验证证书的所有权。要使用由非亚马逊 CA 的 CA 签名的设备证书，必须注册 CA 的证书， AWS IoT 这样我们才能验证设备证书的所有权。

AWS IoT 支持多种自带证书的方式 (BYOC)：

- 首先，注册用于签署客户端证书的 CA，然后注册各个客户端证书。如果要在设备或客户端首次连接时将其注册到其客户端证书 AWS IoT （也称为即时配置），则必须向注册签名 CA AWS IoT 并激活自动注册。
- 如果您无法注册签名 CA，则可以选择在没有 CA 的情况下注册客户端证书。对于未使用 CA 注册的设备，当您将它们连接到 AWS IoT时，您需要出示服务器名称指示（SNI）。

> **Note**
>
> 要使用 CA 注册客户端证书，必须向注册签名 CA AWS IoT，而不是向层次结构中的任何其他 CA 注册。

> **Note**
>
> 在 DEFAULT 模式下，一个 CA 证书只能由一个区域中的一个账户注册。在 SNI_ONLY 模式下，一个 CA 证书可以由一个区域中的多个账户注册。

有关使用 X.509 证书支持多个设备的更多信息，请参阅 设备预调配 以查看 AWS IoT 支持的不同证书管理和预调配选项。

主题

- 管理 CA 证书
- 使用您的 CA 证书创建客户端证书

管理 CA 证书

本节介绍管理您自己的证书颁发机构 (CA) 证书的常见任务。

AWS IoT 如果您使用的是由 AWS IoT 无法识别的 CA 签名的客户端证书，则可以向注册证书颁发机构 (CA)。

如果您希望客户端在首次连接 AWS IoT 时自动向其注册其客户端证书，则必须向注册签署客户端证书的 CA AWS IoT。否则，您不需要注册对客户端证书签发的 CA 证书。

> **Note**
>
> 在 DEFAULT 模式下，一个 CA 证书只能由一个区域中的一个账户注册。在 SNI_ONLY 模式下，一个 CA 证书可以由一个区域中的多个账户注册。

主题：

创建 CA 证书

如果没有 CA 证书，则可以使用 OpenSSL v1.1.1i 工具创建一个。

> **Note**
>
> 您无法在 AWS IoT 控制台中执行此过程。

使用 OpenSSL v1.1.1i 工具创建 CA 证书

1. 生成密钥对。

```
openssl genrsa -out root_CA_key_filename.key 2048
```

2. 使用密钥对中的私有密钥生成 CA 证书。

```
openssl req -x509 -new -nodes \
    -key root_CA_key_filename.key \
    -sha256 -days 1024 \
    -out root_CA_cert_filename.pem
```

注册 CA 证书

这些程序描述了如何注册来自非亚马逊 CA 的证书颁发机构 (CA) 的证书。 AWS IoT Core 使用 CA 证书来验证证书的所有权。要使用由非亚马逊 CA 的 CA 签名的设备证书，您必须向注册 CA 证书，AWS IoT Core 这样它才能验证设备证书的所有权。

注册 CA 证书（控制台）

> **Note**
>
> 若要在控制台中注册 CA 证书，请在控制台中的 Register CA certificate（注册 CA 证书）处开始注册。您可以在多账户模式下注册您的 CA，而无需提供验证证书或访问私有密钥的权限。在多账户模式下，多个 AWS 账户 可以在同一个 AWS 区域中注册 CA。您可以通过提供验证证书和 CA 私有密钥的所有权证明，在单账户模式下注册您的 CA。

注册 CA 证书 (CLI)

您可以在 DEFAULT 模式或 SNI_ONLY 模式下注册 CA 证书。CA 可以一对一 AWS 账户 地在DEFAULT模式下注册 AWS 区域。一个 CA 可以在同一个SNI_ONLY模式下通过多个 AWS 账户 CA 进行注册 AWS 区域。有关 CA 证书模式的更多信息，请参阅 certificateMode。

> **Note**
>
> 我们建议您在 SNI_ONLY 模式下注册 CA。您无需提供验证证书或访问私钥的权限，并且可以在同一个证书 AWS 账户 中多次注册 CA AWS 区域。

在 SNI_ONLY 模式下注册 CA 证书（CLI）- 建议

先决条件

继续操作之前，请确保电脑满足以下条件：

- 根 CA 的证书文件（在以下示例中引用为 *root_CA_cert_filename.pem*）
- OpenSSL v1.1.1i 或更高版本

要在**SNI_ONLY**模式下注册 CA 证书，请使用 AWS CLI

1. 向注册 CA 证书 AWS IoT。使用 register-ca-certificate 命令，输入 CA 证书文件名。有关更多信息，请参阅 AWS CLI 命令参考中的 register-ca-certificate。

   ```
   aws iot register-ca-certificate \
       --ca-certificate file://root_CA_cert_filename.pem \
       --certificate-mode SNI_ONLY
   ```

   如果成功，此命令将返回 *certificateId*。

2. 此时，CA 证书已注册 AWS IoT 但处于非活动状态。CA 证书必须处于活跃状态，然后您才能注册由其签发的任何客户端证书。

   此步骤将激活 CA 证书。

   要激活 CA 证书，请使用 update-certificate 命令，如下所示。有关更多信息，请参阅 AWS CLI 命令参考中的 update-certificate。

   ```
   aws iot update-ca-certificate \
       --certificate-id certificateId \
       --new-status ACTIVE
   ```

要查看 CA 证书的状态，请使用 describe-ca-certificate 命令。有关更多信息，请参阅 AWS CLI 命令参考中的 describe-ca-certificate。

在 **DEFAULT** 模式下注册 CA 证书（CLI）

先决条件

继续操作之前，请确保电脑满足以下条件：

- 根 CA 的证书文件（在以下示例中引用为 *root_CA_cert_filename.pem*）
- 根 CA 证书的私有密钥文件（在以下示例中引用为 *root_CA_key_filename.key*）
- OpenSSL v1.1.1i 或更高版本

要在**DEFAULT**模式下注册 CA 证书，请使用 AWS CLI

1. 要从中获取注册码 AWS IoT，请使用get-registration-code。保存返回的 `registrationCode`，将其用作私有密钥验证证书的 Common Name。有关更多信息，请参阅 AWS CLI 命令参考中的 get-registration-code。

   ```
   aws iot get-registration-code
   ```

2. 为私有密钥验证证书生成密钥对：

   ```
   openssl genrsa -out verification_cert_key_filename.key 2048
   ```

3. 为私有密钥验证证书创建证书签名请求 (CSR)。将证书的 Common Name 字段设置为 get-registration-code 返回的 `registrationCode`。

   ```
   openssl req -new \
       -key verification_cert_key_filename.key \
       -out verification_cert_csr_filename.csr
   ```

   将提示您输入一些信息，包括证书的Common Name。

   ```
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   -----
   Country Name (2 letter code) [AU]:
       State or Province Name (full name) []:
       Locality Name (for example, city) []:
       Organization Name (for example, company) []:
       Organizational Unit Name (for example, section) []:
       Common Name (e.g. server FQDN or YOUR name) []:your_registration_code
       Email Address []:

       Please enter the following 'extra' attributes
       to be sent with your certificate request
       A challenge password []:
       An optional company name []:
   ```

4. 使用 CSR 创建私有密钥验证证书：

```
openssl x509 -req \
    -in verification_cert_csr_filename.csr \
    -CA root_CA_cert_filename.pem \
    -CAkey root_CA_key_filename.key \
    -CAcreateserial \
    -out verification_cert_filename.pem \
    -days 500 -sha256
```

5. 向注册 CA 证书 AWS IoT。将 CA 证书文件名和私有密钥验证证书文件名传递给 register-ca-certificate 命令，如下所示：有关更多信息，请参阅 AWS CLI 命令参考中的 register-ca-certificate。

   ```
   aws iot register-ca-certificate \
       --ca-certificate file://root_CA_cert_filename.pem \
       --verification-cert file://verification_cert_filename.pem
   ```

   如果成功，此命令将返回 *certificateId*。

6. 此时，CA 证书已注册 AWS IoT 但未激活。CA 证书必须处于活跃状态，然后您才能注册由其签发的任何客户端证书。

   此步骤将激活 CA 证书。

   要激活 CA 证书，请使用 update-certificate 命令，如下所示。有关更多信息，请参阅 AWS CLI 命令参考中的 update-certificate。

   ```
   aws iot update-ca-certificate \
       --certificate-id certificateId \
       --new-status ACTIVE
   ```

要查看 CA 证书的状态，请使用 describe-ca-certificate 命令。有关更多信息，请参阅 AWS CLI 命令参考中的 describe-ca-certificate。

创建 CA 验证证书以在控制台中注册 CA 证书

> **Note**
>
> 此过程仅适用于从 AWS IoT 控制台注册 CA 证书的情况。

如果您不是从控制台进入此过程，请在 AWS IoT 控制台的 Register CA 证书上启动 CA 证书注册过程。

继续操作之前，请确保同一台电脑满足以下条件：

- 根 CA 的证书文件（在以下示例中引用为 *root_CA_cert_filename.pem*）
- 根 CA 证书的私有密钥文件（在以下示例中引用为 *root_CA_key_filename.key*）
- OpenSSL v1.1.1i 或更高版本

使用命令行界面创建 CA 验证证书，以在控制台中注册 CA 证书

1. 将 *verification_cert_key_filename.key* 替换为要创建的验证证书密钥文件的文件名（例如 **verification_cert.key**）。然后运行此命令，为私有密钥验证证书生成密钥对：

   ```
   openssl genrsa -out verification_cert_key_filename.key 2048
   ```

2. 将 *verification_cert_key_filename.key* 替换为在步骤 1 中创建的密钥文件的名称。

   将 *verification_cert_csr_filename.csr* 替换为要创建的证书签名请求 (CSR) 文件的名称。例如，**verification_cert.csr**。

   运行此命令以创建 CSR 文件。

   ```
   openssl req -new \
       -key verification_cert_key_filename.key \
       -out verification_cert_csr_filename.csr
   ```

   该命令会提示您输入其他信息（稍后说明）。

3. 在 AWS IoT 控制台的验证证书容器中，复制注册码。

4. openssl 命令提示您输入的信息显示在以下示例中。除了 Common Name 字段，您可以在其它字段中输入自己的值或将其留空。

   在 Common Name 字段中，粘贴您在上一步骤中复制的注册码。

   ```
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   ```

```
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:your_registration_code
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

完成后，命令会创建 CSR 文件。

5. 将 *verification_cert_csr_filename.csr* 替换为在上一步骤中使用的 *verification_cert_csr_filename.csr*。

   将 *root_CA_cert_filename.pem* 替换为要注册的 CA 证书的文件名。

   将 *root_CA_key_filename.key* 替换为 CA 证书的私有密钥文件的文件名。

   将 *verification_cert_filename.pem* 替换为要创建的验证证书的文件名。例如，**verification_cert.pem**。

   ```
   openssl x509 -req \
       -in verification_cert_csr_filename.csr \
       -CA root_CA_cert_filename.pem \
       -CAkey root_CA_key_filename.key \
       -CAcreateserial \
       -out verification_cert_filename.pem \
       -days 500 -sha256
   ```

6. 完成 OpenSSL 命令后，您应该准备好这些文件，以便在返回控制台时使用。

   - 您的 CA 证书文件（上一条命令中使用的 *root_CA_cert_filename.pem*）
   - 您在上一步骤中创建的验证证书（上一条命令中使用的 *verification_cert_filename.pem*）

停用 CA 证书

启用证书颁发机构 (CA) 证书进行自动客户端证书注册后， AWS IoT 会检查 CA 证书以确保 CA 已启用ACTIVE。如果 CA 证书是INACTIVE，则 AWS IoT 不允许注册客户端证书。

通过将 CA 证书设置为 INACTIVE，可防止该 CA 颁发的任何新客户端证书自动注册。

> **Note**
>
> 除非您明确吊销由受损的 CA 证书签发的每个已注册客户端证书，否则所有此类证书均将继续工作。

停用 CA 证书（控制台）

使用 AWS IoT 控制台停用 CA 证书

1. 登录 AWS Management Console 并打开AWS IoT 控制台。
2. 在左侧的导航窗格中，依次选择安全、CA。
3. 在证书颁发机构列表中，找到要停用的证书颁发机构，然后选择省略号图标以打开选项菜单。
4. 在选项菜单上，选择停用。

证书颁发机构应在列表中显示为停用。

> **Note**
>
> AWS IoT 控制台不提供列出由您停用的 CA 签署的证书的方法。有关列出这些证书的 AWS CLI 选项，请参阅停用 CA 证书 (CLI)。

停用 CA 证书 (CLI)

AWS CLI 提供了停用 CA 证书的update-ca-certificate命令。

```
aws iot update-ca-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

使用 list-certificates-by-ca 命令获取已由指定 CA 签名的所有已注册客户端证书的列表。对于由指定 CA 证书签名的每个客户端证书，请使用 update-certificate 命令吊销该客户端证书以避免使用它。

使用 describe-ca-certificate 命令查看 CA 证书的状态。

使用您的 CA 证书创建客户端证书

您可以使用自己的证书颁发机构 (CA) 创建客户端证书。必须先注册客户证书， AWS IoT 然后才能使用。有关客户端证书的注册选项的信息，请参阅注册客户端证书。

创建客户端证书 (CLI)

> **Note**
>
> 您无法在 AWS IoT 控制台中执行此过程。

要使用创建客户证书 AWS CLI

1. 生成密钥对。

```
openssl genrsa -out device_cert_key_filename.key 2048
```

2. 为客户端证书创建 CSR。

```
openssl req -new \
    -key device_cert_key_filename.key \
    -out device_cert_csr_filename.csr
```

系统将提示您输入一些信息，如下所示：

```
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
    State or Province Name (full name) []:
    Locality Name (for example, city) []:
```

```
    Organization Name (for example, company) []:
    Organizational Unit Name (for example, section) []:
    Common Name (e.g. server FQDN or YOUR name) []:
    Email Address []:

    Please enter the following 'extra' attributes
    to be sent with your certificate request
    A challenge password []:
    An optional company name []:
```

3. 从 CSR 创建客户端证书。

```
openssl x509 -req \
    -in device_cert_csr_filename.csr \
    -CA root_CA_cert_filename.pem \
    -CAkey root_CA_key_filename.key \
    -CAcreateserial \
    -out device_cert_filename.pem \
    -days 500 -sha256
```

此时，客户证书已创建，但尚未在中注册 AWS IoT。有关注册客户端证书的方式和时机的信息，请参阅注册客户端证书。

注册客户端证书

必须向注册客户端证书， AWS IoT 才能在客户端和之间进行通信 AWS IoT。您可以手动注册每个客户端证书，也可以将客户端证书配置 AWS IoT 为在客户端首次连接时自动注册。

如果您希望客户端和设备在首次连接时注册其客户端证书，则必须在要使用的区域中拥有使用 AWS IoT 签署客户端证书所需的 注册 CA 证书。Amazon Root CA 将自动向注册 AWS IoT。

客户证书可以由 AWS 账户 和地区共享。在您要使用客户端证书的每个账户和区域中，必须执行这些主题中的流程。在一个账户或区域中注册的客户端证书，不会在另一个账户或区域中自动识别。

> **Note**
>
> 使用传输层安全性 (TLS) 协议连接到 AWS IoT 的客户端必须支持 TLS 的服务器名称指示 (SNI) 扩展。有关更多信息，请参阅 运输安全 AWS IoT Core。

主题

- 手动注册客户端证书
- 在客户端连接到注册时 AWS IoT just-in-time 注册客户端证书 (JITR)

手动注册客户端证书

您可以使用 AWS IoT 控制台和手动注册客户证书 AWS CLI。

使用的注册程序取决于证书是否将由 AWS 账户和地区共享。在一个账户或区域中注册的客户端证书，不会在另一个账户或区域中自动识别。

在您要使用客户端证书的每个账户和区域中，必须执行此主题中的流程。客户证书可以由 AWS 账户 s 和 Regions 共享。

注册由已注册的 CA（控制台）签发的客户端证书

> **Note**
>
> 在执行此过程之前，请确保您拥有客户端证书的.pem 文件，并且该客户端证书由您注册的 CA 签名。 AWS IoT

AWS IoT 使用控制台注册现有证书

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在导航窗格的 Manage（管理）部分下，选择 Security（安全性），然后选择 Certificates（证书）。
3. 在 Certificates（证书）对话框的 Certificates（证书）页上，选择 Add certificate（添加证书），然后选择 Register certificates（注册证书）。
4. 在 Certificates to upload（要上载的证书）对话框的 Register certificate（注册证书）页上，执行以下操作：

   - 选择 CA is registered with AWS IoT（CA 注册到 AWS IoT）。
   - 从 Choose a CA certificate（选择 CA 证书）中，选择您的 Certification authority（证书颁发机构）。
     - 选择 Register a new CA（注册新 CA）以注册未向 AWS IoT注册的新 Certification authority（证书颁发机构）。
     - 如果 Amazon Root certificate authority（Amazon 根证书颁发机构）是您的证书颁发机构，请将 Choose a CA certificate（选择 CA 证书）留空。

- 最多选择 10 个证书进行上传和注册 AWS IoT。
  - 使用您在创建 AWS IoT 客户证书和使用您的 CA 证书创建客户端证书中创建的证书文件。
- 选择 Activate（激活）或 Deactivate（停用）。如果选择 Deactive（停用），激活或停用客户端证书介绍了如何在证书注册后激活证书。
- 选择 Register。

在 Certificates（证书）对话框的 Certificates（证书）页上，现在将显示您注册的证书。

注册由未注册的 CA（控制台）签发的客户端证书

> **Note**
>
> 在执行此流程之前，请确保您拥有客户端证书 .pem 文件。

AWS IoT 使用控制台注册现有证书

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在左侧的导航窗格中，选择安全，选择证书，然后选择创建。
3. 在 Create a certificate（创建证书）上，找到 Use my certificate（使用我的证书）条目，然后选择 Get started（入门）。
4. 在 Select CA（选择 CA）上，选择 Next（下一步）。
5. 在 Register exsiting device certificates（注册现有的设备证书）上，选择 Select certificates（选择证书），并选择最多 10 个要注册的证书文件。
6. 关闭文件对话框后，选择在注册客户端证书时是要激活还是吊销客户端证书。

   如果您在注册证书时未激活证书，激活客户端证书（控制台）中介绍了以后如何激活证书。

   如果证书在注册时被吊销，则以后无法激活。

   选择要注册的证书文件，选择注册后要执行的操作，然后选择 Register certificates（注册证书）。

成功注册的客户端证书将显示在证书列表中。

注册由已注册的 CA (CLI) 签发的客户端证书

> **Note**
>
> 在执行此流程之前，请确保您具有证书颁发机构 (CA) .pem 和客户端证书的 .pem 文件。客户端证书必须由您注册的证书颁发机构 (CA) 签名 AWS IoT。

使用 register-certificate 命令注册客户端证书但不激活。

```
aws iot register-certificate \
    --certificate-pem file://device_cert_filename.pem \
    --ca-certificate-pem file://ca_cert_filename.pem
```

客户证书已在中注册 AWS IoT，但尚未激活。有关如何在以后激活的信息，请参阅 激活客户端证书 (CLI)。

您也可以在使用此命令注册客户端证书时进行激活。

```
aws iot register-certificate \
    --set-as-active \
    --certificate-pem file://device_cert_filename.pem \
    --ca-certificate-pem file://ca_cert_filename.pem
```

有关激活证书以使其可用于连接的更多信息 AWS IoT，请参阅 激活或停用客户端证书

注册由未注册的 CA (CLI) 签发的客户端证书

> **Note**
>
> 在执行此流程之前，请确保您拥有证书 .pem 文件。

使用 register-certificate-without-ca 命令注册客户端证书但不激活。

```
aws iot register-certificate-without-ca \
    --certificate-pem file://device_cert_filename.pem
```

客户证书已在中注册 AWS IoT，但尚未激活。有关如何在以后激活的信息，请参阅 激活客户端证书 (CLI)。

您也可以在使用此命令注册客户端证书时进行激活。

```
aws iot register-certificate-without-ca \
    --status ACTIVE \
    --certificate-pem file://device_cert_filename.pem
```

有关激活证书以使其可用于连接的更多信息 AWS IoT，请参阅激活或停用客户端证书。

在客户端连接到注册时 AWS IoT just-in-time 注册客户端证书 (JITR)

您可以将 CA 证书配置为启用已签名的客户端证书，以便在客户端首次连接时 AWS IoT 自动注册该证书 AWS IoT。

要在客户端首次连接时注册客户端证书，必须启用 CA 证书以进行自动注册，并将客户端的第一个连接配置为提供所需的证书。 AWS IoT

配置 CA 证书以支持自动注册（控制台）

使用 AWS IoT 控制台配置 CA 证书以支持自动注册客户证书

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在左侧的导航窗格中，依次选择安全、CA。
3. 在证书颁发机构列表中，找到要启用自动注册的颁发机构，然后使用省略号图标打开选项菜单。
4. 在选项菜单上，选择启用自动注册。

> **Note**
>
> 证书颁发机构列表中不显示自动注册状态。要查看证书颁发机构的自动注册状态，您必须打开证书颁发机构的详细信息页面。

配置 CA 证书以支持自动注册 (CLI)

如果您已经向注册了 CA 证书 AWS IoT，请使用update-ca-certificate命令将 CA 证书设置为`autoRegistrationStatus`ENABLE。

```
aws iot update-ca-certificate \
--certificate-id caCertificateId \
--new-auto-registration-status ENABLE
```

如果您要在注册 CA 证书时启用 autoRegistrationStatus，请使用 register-ca-certificate 命令。

```
aws iot register-ca-certificate \
--allow-auto-registration  \
--ca-certificate file://root_CA_cert_filename.pem \
--verification-cert file://verification_cert_filename.pem
```

使用 describe-ca-certificate 命令查看 CA 证书的状态。

配置客户端的首次连接以进行自动注册

当客户端首次尝试 AWS IoT 连接时，在传输层安全 (TLS) 握手期间，由您的 CA 证书签名的客户端证书必须存在于客户端上。

当客户端连接到时 AWS IoT，使用您在创建客户端证书或创建自己的 AWS IoT 客户端证书中创建的客户端证书。 AWS IoT 将 CA 证书识别为已注册的 CA 证书，注册客户端证书，并将其状态设置为PENDING_ACTIVATION。这意味着，已自动注册客户端证书，该证书正在等待激活。客户端证书的状态必须为 ACTIVE，然后才能将其用于连接到 AWS IoT。有关激活客户端证书的信息，请参阅激活或停用客户端证书。

> **Note**
>
> 您可以使用 AWS IoT Core 即时注册 (JITR) 功能配置设备，而不必在设备首次连接时发送整个信任链。 AWS IoT Core可以出示 CA 证书，但连接时设备需要发送服务器名称指示 (SNI) 扩展。

当 AWS IoT 自动注册证书或客户端提供PENDING_ACTIVATION状态为证书时，会向以下 MQTT 主题 AWS IoT 发布消息：

$aws/events/certificates/registered/*caCertificateId*

其中 *caCertificateId* 是颁发客户端证书的 CA 证书的 ID。

发布到该主题的消息具有以下结构：

```
{
        "certificateId": "certificateId",
        "caCertificateId": "caCertificateId",
        "timestamp": timestamp,
```

```
        "certificateStatus": "PENDING_ACTIVATION",
        "awsAccountId": "awsAccountId",
        "certificateRegistrationTimestamp": "certificateRegistrationTimestamp"
}
```

您可以创建一项规则，以侦听此主题并执行一些操作。我们建议您创建一项 Lambda 规则，以验证客户端证书不在证书吊销列表 (CRL) 中，激活该证书，创建策略并将其附加到证书中。该策略将确定客户端能够访问的资源。有关如何创建监听`$aws/events/certificates/registered/caCertificateID`主题并执行这些操作的 Lambda 规则的更多信息，请参阅just-in-time 注册客户证书。 AWS IoT

如果在自动注册客户端证书的过程中出现任何错误或异常，则会在日志中 AWS IoT 将事件或消息发送到您的日 CloudWatch 志。有关为您的账户设置日志的更多信息，请参阅 Amazon CloudWatch 文档。

激活或停用客户端证书

AWS IoT 验证客户端证书在对连接进行身份验证时是否处于活动状态。

您可以创建和注册客户端证书但不激活，这样直到您想要使用时才让这些证书可用。您还可以停用活动的客户端证书以暂时禁用它们。最后，您可以吊销客户端证书，防止以后使用这些证书。

激活客户端证书（控制台）

使用 AWS IoT 控制台激活客户端证书

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在左侧导航窗格中，选择安全，然后选择证书。
3. 在证书列表中，找到要激活的证书，然后使用省略号图标打开选项菜单。
4. 在选项菜单中，选择激活。

证书应在证书列表中显示为活动。

停用客户端证书（控制台）

使用控制台停用客户端证书 AWS IoT

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在左侧导航窗格中，选择安全，然后选择证书。
3. 在证书列表中，找到要停用的证书，然后使用省略号图标打开选项菜单。

4. 在选项菜单中，选择停用。

证书应在证书列表中显示为停用。

激活客户端证书 (CLI)

AWS CLI 提供了激活证书的update-certificate命令。

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status ACTIVE
```

如果命令成功，证书的状态将为 ACTIVE。运行 describe-certificate 以查看证书的状态。

```
aws iot describe-certificate \
    --certificate-id certificateId
```

停用客户端证书 (CLI)

AWS CLI 提供了停用证书的update-certificate命令。

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status INACTIVE
```

如果命令成功，证书的状态将为 INACTIVE。运行 describe-certificate 以查看证书的状态。

```
aws iot describe-certificate \
    --certificate-id certificateId
```

将事物或策略附加到客户端证书

当您创建和注册与 AWS IoT 事物分开的证书时，该证书将没有任何授权任何 AWS IoT 操作的策略，也不会与任何 AWS IoT 事物对象相关联。本节介绍如何将这些关系添加到注册的证书中。

> **Important**
>
> 要完成这些流程，您必须具有要附加到证书的已创建事物或策略。

证书用于对设备进行身份验证， AWS IoT 以便设备可以连接。将证书附加到事物资源可建立设备（通过证书的方式）与事物资源之间的关系。要授权设备执行 AWS IoT 操作，例如允许设备连接和发布消息，必须将相应的策略附加到设备的证书上。

将事物附加到客户端证书（控制台）

您需要事物对象的名称才能完成此流程。

将事物对象附加到已注册的证书

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在左侧导航窗格中，选择安全，然后选择证书。
3. 在证书列表中，找到要附加策略的证书，选择省略号图标以打开证书的选项菜单，然后选择附加事物。
4. 在弹出窗口中，找到要附加到证书的事物的名称，选中其复选框，然后选择 Attach（附上）。

现在，事物对象应显示在证书详细信息页面上的事物列表中。

将策略附加到客户端证书（控制台）

您需要策略对象的名称才能完成此流程。

将策略对象附加到已注册的证书

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在左侧导航窗格中，选择安全，然后选择证书。
3. 在证书列表中，找到要附加策略的证书，选择省略号图标以打开证书的选项菜单，然后选择附加策略。
4. 在弹出窗口中，找到要附加到证书的策略的名称，选中其复选框，然后选择 Attach（附上）。

现在，策略对象应显示在证书详细信息页面上的策略列表中。

将事物附加到客户端证书 (CLI)

AWS CLI 提供了将事物对象附加到证书的attach-thing-principal命令。

```
aws iot attach-thing-principal \
    --principal certificateArn \
    --thing-name thingName
```

将策略附加到客户端证书 (CLI)

AWS CLI 提供了将策略对象附加到证书的attach-policy命令。

```
aws iot attach-policy \
    --target certificateArn \
    --policy-name policyName
```

吊销客户端证书

如果您在已注册的客户端证书上检测到可疑活动，则可以吊销该证书，使其无法再次使用。

> **Note**
>
> 证书一旦被吊销，就无法更改其状态。也就是说，证书状态不能更改为 `Active` 或任何其它状态。

吊销客户端证书（控制台）

使用控制台吊销客户端证书 AWS IoT

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在左侧导航窗格中，选择安全，然后选择证书。
3. 在证书列表中，找到要吊销的证书，然后使用省略号图标打开选项菜单。
4. 在选项菜单中，选择吊销。

如果证书已成功吊销，它将在证书列表中显示为已吊销。

吊销客户端证书 (CLI)

AWS CLI 提供了吊销证书的update-certificate命令。

```
aws iot update-certificate \
    --certificate-id certificateId \
    --new-status REVOKED
```

如果命令成功，证书的状态将为 `REVOKED`。运行 describe-certificate 以查看证书的状态。

```
aws iot describe-certificate \
```

```
    --certificate-id certificateId
```

将设备证书传输到其它账户。

属于一个证书的 X.509 证书 AWS 账户 可以转移到另一个证书。 AWS 账户

将 X.509 证书从一个证书转移到另一个证书 AWS 账户

1. the section called “开始证书传输”

   在启动传输之前，必须停用证书并从所有策略和事物中分离。
2. the section called “接受或拒绝证书传输”

   接收账户必须明确接受或拒绝传输的证书。接收账户户接受证书后，必须激活证书才能使用。
3. the section called “取消证书传输”

   如果证书未被接受，原始账户可以取消传输。

开始证书传输

您可以使用AWS IoT 控制台或开始将证书转移 AWS 账户 到另一个证书 AWS CLI。

开始证书传输（控制台）

要完成此流程，您需要提供您要传输的证书 ID。

在带有要传输证书的账户执行此流程。

要开始将证书传输到另一个 AWS 账户。

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在左侧导航窗格中，选择安全，然后选择证书。

   选择您打算传输并带有 Active（活动）或者 Inactive（非活动）状态的证书并打开其详细信息页面。
3. 在证书 Details（详细信息）页面的 Actions（操作）菜单中，如果 Deactivate（停用）选项可用，则选择 Deactivate（停用）选项来停用证书。
4. 在证书的 Details（详细信息）页面中，在左侧菜单中，选择 Policies（策略）。
5. 在证书的 Policies（策略）页面上，如果证书附加了任何策略，请打开策略的选项菜单，然后选择 Detach（分离）以分离各个证书。

在您继续操作前，该证书不能附加有任何策略。

6. 在证书的 Policies（策略）页面中，在左侧菜单中，选择 Thins（事物）。
7. 在证书的 Things（事物）页面，如果证书附加了任何事物，请打开事物的选项菜单并选择 Detach（分离）以分离各个事物。

   在您继续操作前，该证书不能附加有任何事物。
8. 在证书的 Things（事物）页面中，在左侧菜单中，选择 Details（详细信息）。
9. 在证书 Details（详细信息）页面的 Actions（操作）菜单中，选择 Start transfer（开始传输）来打开 Start transfer（开始传输）对话框。
10. 在 “开始转移” 对话框中，输入要接收证书的账户号和一条可选的短消息。 AWS 账户
11. 选择 Start transfer（开始传输）以传输证书。

控制台应显示一条消息，指示传输成功或失败。如果传输开始，证书的状态将更新为 Transferred（已传输）。

开始证书传输 (CLI)

要完成此流程，您需要使用待传输证书的 *certificateId* 和 *certificateArn*。

在带有要传输证书的账户执行此流程。

要开始将证书传输到另一个 AWS 账户

1. 使用 update-certificate 命令来停用证书。

   ```
   aws iot update-certificate --certificate-id certificateId --new-status INACTIVE
   ```
2. 分离所有策略。

   1. 使用 list-attached-policies 命令列出附加到该证书的策略。

      ```
      aws iot list-attached-policies --target certificateArn
      ```
   2. 对于每个附加的策略，请使用 detach-policy 命令予以分离。

      ```
      aws iot detach-policy --target certificateArn --policy-name policy-name
      ```
3. 分离所有事物。

1. 使用 list-principal-things 命令列示附加到证书上的事物。

```
aws iot list-principal-things --principal certificateArn
```

2. 对于每个附加的事物，请使用 detach-thing-principal 命令予以分离。

```
aws iot detach-thing-principal --principal certificateArn --thing-name thing-name
```

4. 使用 transfer-certificate 命令启动证书传输。

```
aws iot transfer-certificate --certificate-id certificateId --target-aws-
account account-id
```

接受或拒绝证书传输

您可以使用AWS IoT 控制台或，接受或拒绝 AWS 账户 从他 AWS 账户 人转移给您的证书 AWS CLI。

接受或拒绝证书传输（控制台）

要完成此流程，您需要提供已转移到您的账户的证书 ID。

在接收已转移证书的账户执行此流程。

接受或拒绝转移到您 AWS 账户的证书

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在左侧导航窗格中，选择安全，然后选择证书。

   选择您想要接受或拒绝且状态为 Pending transfer（待转移）的证书，并打开其详细信息页面。
3. 在证书的 Details（详细信息）页面上，在 Actions（操作）菜单中，

   - 如要接受证书，请选择 Accept transfer（接受传输）。
   - 如不接受证书，请选择 Reject transfer（拒绝传输）。

接受或拒绝证书传输 (CLI)

要完成此流程，您需要想要接受或拒绝的证书传输的 *certificateId*。

在接收已转移证书的账户执行此流程。

接受或拒绝转移到您 AWS 账户的证书

1. 使用 accept-certificate-transfer 命令以接受证书。

```
aws iot accept-certificate-transfer --certificate-id certificateId
```

2. 使用 reject-certificate-transfer 命令以拒绝证书。

```
aws iot reject-certificate-transfer --certificate-id certificateId
```

取消证书传输

您可以使用 AWS IoT 控制台或 AWS CLI在证书传输被接受之前将其取消。

取消证书传输（控制台）

要完成此流程，您需要提供要取消的证书传输的 ID。

在启动证书传输的账户执行此流程。

取消证书传输

1. 登录 AWS 管理控制台并打开AWS IoT 控制台。
2. 在左侧导航窗格中，选择安全，然后选择证书。

   选择状态为 Transferred（已传输）且您想要取消传输的证书，并打开其选项菜单。
3. 在证书的选项菜单上，选择 Revoke transfer（撤销传输）选项来取消证书传输。

> **Important**
>
> 请谨慎处理，不要将 Revoke transfer（撤销传输）选项与 Revoke（撤销）选项混淆。Revoke transfer（撤销传输）选项用于取消证书传输，而 Revoke（撤销）选项使证书无法为 AWS IoT所用，且操作不可逆。

取消证书传输 (CLI)

要完成此流程，您需要想要取消的证书传输的 *certificateId*。

在启动证书传输的账户执行此流程。

使用 cancel-certificate-transfer 命令取消证书传输。

```
aws iot cancel-certificate-transfer --certificate-id certificateId
```

IAM 用户、组和角色

IAM 用户、组和角色是用于在 AWS中管理身份和身份验证的标准机制。您可以使用 AWS SDK 和，使用它们连接到 AWS IoT HTTP 接口 AWS CLI。

IAM 角色还 AWS IoT 允许代表您访问您账户中的其他 AWS 资源。例如，如果您想让设备将其状态发布到 DynamoDB 表，则 IAM 角色 AWS IoT 允许与亚马逊 DynamoDB 进行交互。有关更多信息，请参阅 IAM 角色。

对于通过 HTTP 的消息代理连接，使用签名版本 4 签名过程对用户、群组和角色 AWS IoT 进行身份验证。有关信息，请参阅签署 AWS API 请求。

将 AWS 签名版本 4 与一起使用时 AWS IoT，客户端必须在 TLS 实现中支持以下内容：

- TLS 1.2
- SHA-256 RSA 证书签名验证
- 来自 TLS 密码套件支持部分的密码套件之一

有关信息，请参阅 的身份和访问管理 AWS IoT。

Amazon Cognito 身份

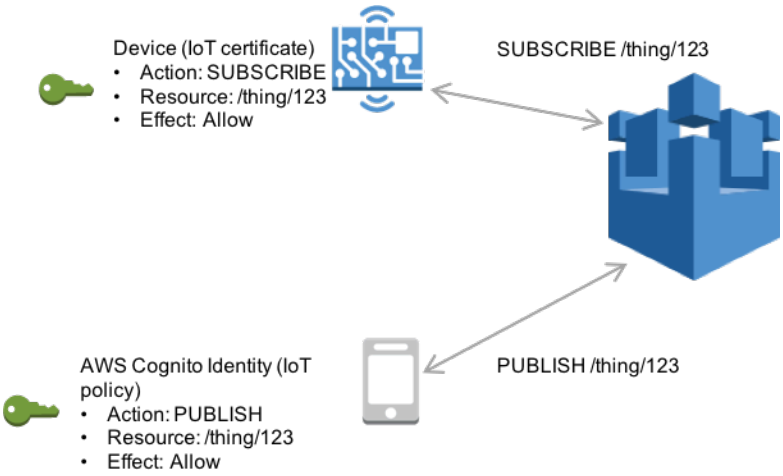
Amazon Cognito Identity 允许您创建临时的、有限的权限 AWS 凭证，以便在移动和网络应用程序中使用。当您使用 Amazon Cognito Identity 时，请创建身份池，为您的用户创建唯一身份，并使用身份提供商（例如 Login with Amazon、Facebook 和 Google）对他们进行身份验证。您还可以将 Amazon Cognito 身份与您自己的经开发人员验证的身份结合使用。有关更多信息，请参阅 Amazon Cognito Identity。

要使用 Amazon Cognito 身份，请定义与 IAM 角色关联的 Amazon Cognito 身份池。IAM 角色与一个 IAM 策略相关联，该策略向您的身份池中的身份授予访问 AWS 资源（如调用 AWS 服务）的权限。

Amazon Cognito Identity 可创建未经身份验证的身份和经过身份验证的身份。未经身份验证的身份用于希望在不登录的情况下使用该应用程序的移动或 Web 应用程序中的访客用户。未经身份验证的用户仅授予与身份池关联的 IAM policy 中指定的权限。

当您使用经过身份验证的身份时，除了附加到身份池的 IAM 策略外，还必须将 AWS IoT 策略附加到 Amazon Cognito 身份。要附加 AWS IoT 策略，请使用 AttachPolicyAPI 并向 AWS IoT 应用程序的单个用户授予权限。您可以使用该 AWS IoT 策略为特定客户及其设备分配细粒度的权限。

经过身份验证和未经身份验证的用户是不同的身份类型。如果您未将 AWS IoT 策略附加到 Amazon Cognito Identity，则经过身份验证的用户将无法在中进行授权， AWS IoT 并且无法访问 AWS IoT 资源和操作。有关为 Amazon Cognito 身份创建策略的更多信息，请参阅 发布/订阅策略示例 和 使用 Amazon Cognito 身份的授权。



自定义身份验证和授权

AWS IoT Core 允许您定义自定义授权方，以便您可以管理自己的客户端身份验证和授权。当您需要使用 AWS IoT Core 原生支持的身份验证机制以外的身份验证机制时，这很有用。（有关本地支持机制的更多信息，请参阅 the section called “客户端身份验证”）。

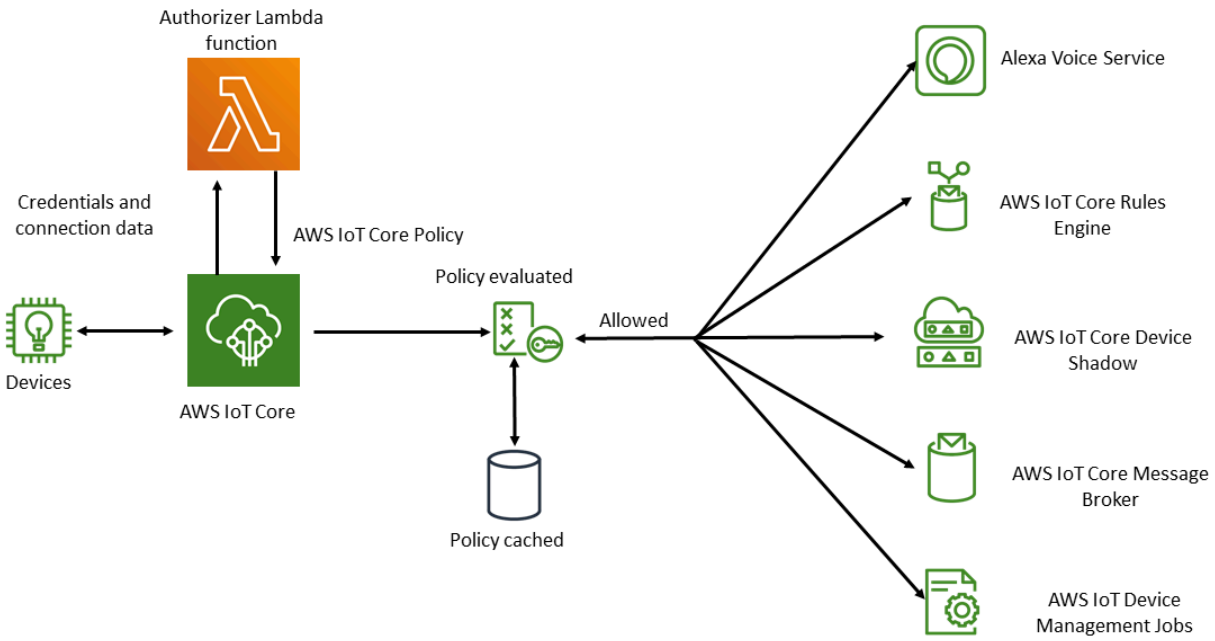
例如，如果您要将现场现有设备迁移到， AWS IoT Core 并且这些设备使用自定义不记名令牌或 MQTT 用户名和密码进行身份验证，则 AWS IoT Core 无需为其配置新身份即可将其迁移到。您可以将自定义身份验证与任何 AWS IoT Core 支持的通信协议一起使用。有关受 AWS IoT Core 支持的协议的更多信息，请参阅 the section called “设备通信协议”。

主题

- 了解自定义身份验证工作流
- 创建和管理自定义授权方
- 使用自定义身份验证 AWS IoT Core 进行连接
- 授权方故障排除

了解自定义身份验证工作流

自定义身份验证使您能够定义如何通过使用授权方资源对客户端进行身份验证和授权。 每个授权方都包含对客户管理的 Lambda 函数的引用、用于验证设备凭证的可选公钥以及其他配置信息。 下图说明了中自定义身份验证的授权工作流程 AWS IoT Core。



AWS IoT Core 自定义身份验证和授权工作流程

下面的列表说明了自定义身份验证和授权工作流中的每个步骤。

1. 设备使用支持的终端连接到客户 AWS IoT Core 的数据端点the section called “设备通信协议”。 设备在请求的标头字段或查询参数（针对基于协议的 HTTP 发布或 MQTT），或者在 MQTT CONNECT 消息的用户名和密码字段（适用于 WebSockets 协议上的 MQTT 和 MQTT）中传递凭证。 WebSockets
2. AWS IoT Core 检查以下两个条件之一：
   - 传入的请求指定授权方。
   - 接收请求 AWS IoT Core 的数据端点已为其配置了默认授权方。

   如果通过上述任一方式 AWS IoT Core 找到授权方，则 AWS IoT Core 会触发与授权方关联的 Lambda 函数。

3. （可选）如果您已启用令牌签名，则在触发 Lambda 函数之前，使用存储在授权方中的公钥来 AWS IoT Core 验证请求签名。如果验证失败， AWS IoT Core 将停止请求而不调用 Lambda 函数。
4. Lambda 函数接收请求中的凭证和连接元数据，并做出身份验证决策。
5. Lambda 函数返回身份验证决策的结果和一份 AWS IoT Core 策略文档，其中指定了连接中允许的操作。 Lambda 函数还会返回信息，指定通过调用 Lambda 函数 AWS IoT Core 重新验证请求中的证书的频率。
6. AWS IoT Core 根据连接从 Lambda 函数收到的策略评估连接上的活动。
7. 在建立连接并首次调用您的自定义授权器 Lambda 之后，在空闲连接上，下一次调用最多可延迟 5 分钟，无需任何 MQTT 操作。之后，后续调用将遵循您的自定义授权方 Lambda 中的刷新间隔。这种方法可以防止可能超过您的 Lambda 并发限制的过度调用。 AWS 账户

扩展注意事项

由于 Lambda 函数为您的授权方处理身份验证和授权，因此该函数受 Lambda 定价和服务限制的约束，例如并发执行率。有关 Lambda 定价的更多信息，请参阅 Lambda 定价。您可以通过调整 Lambda 函数响应中的 `refreshAfterInSeconds` 和 `disconnectAfterInSeconds` 参数管理您 Lambda 函数上的负载。有关 Lambda 函数响应内容的更多信息，请参阅 the section called “定义您的 Lambda 函数”。

> **Note**
>
> 如果启用签名，则可以防止无法识别的客户端过度触发 Lambda。在禁用授权方的签名之前，请考虑这一点。

> **Note**
>
> 自定义授权方的 Lambda 函数超时限制为 5 秒。

创建和管理自定义授权方

AWS IoT Core 使用授权方资源实现自定义身份验证和授权方案。各授权方均包括以下组件：

- 名称：用户定义的唯一字符串，用于标识授权方。
- Lambda 函数 ARN：Lambda 函数的 Amazon Resource Name (ARN)，用于实现授权和身份验证逻辑。

- 令牌密钥名称：用于从 HTTP 标头、查询参数或 MQTT CONNECT 用户名中提取令牌以执行签名验证的键名称。如果在授权方中启用了签名，则需要此值。
- 签名禁用标志（可选）：指定是否禁用凭证签名要求的布尔值。这对于签名凭证没有意义的情况非常有用，例如使用 MQTT 用户名和密码的身份验证方案。默认值为 `false`，因此默认情况下签名将处于启用状态。
- 令牌签名公有密钥： AWS IoT Core 用于验证令牌签名的公有密钥。其最小长度为 2048 位。如果在授权方中启用了签名，则需要此值。

Lambda 将根据 Lambda 函数的运行次数以及函数中代码执行所需的时间向您收取费用。有关 Lambda 定价的更多信息，请参阅 Lambda 定价。有关创建 Lambda 函数的更多信息，请参阅 Lambda 开发人员指南。

> **Note**
>
> 如果启用签名，则可以防止无法识别的客户端过度触发 Lambda。在禁用授权方的签名之前，请考虑这一点。

> **Note**
>
> 自定义授权方的 Lambda 函数超时限制为 5 秒。

定义您的 Lambda 函数

当 AWS IoT Core 调用您的授权方时，它会使用包含以下 JSON 对象的事件触发与授权方关联的 Lambda。示例 JSON 对象包含所有可能的字段。不包括与连接请求无关的任何字段。

```
{
    "token" :"aToken",
    "signatureVerified": Boolean, // Indicates whether the device gateway has validated
 the signature.
    "protocols": ["tls", "http", "mqtt"], // Indicates which protocols to expect for
 the request.
    "protocolData": {
        "tls" : {
            "serverName": "serverName" // The server name indication (SNI) host_name
 string.
```

```
        },
        "http": {
            "headers": {
                "#{name}": "#{value}"
            },
            "queryString": "?#{name}=#{value}"
        },
        "mqtt": {
            "username": "myUserName",
            "password": "myPassword", // A base64-encoded string.
            "clientId": "myClientId" // Included in the event only when the device
 sends the value.
        }
    },
    "connectionMetadata": {
        "id": UUID // The connection ID. You can use this for logging.
    },
}
```

Lambda 函数应使用此信息对传入连接进行身份验证，并决定允许在连接中执行哪些操作。函数应发送包含以下值的响应。

- `isAuthenticated`：一个布尔值，指示是否已对请求进行身份验证。
- `principalId`：字母数字字符串，它充当自定义授权请求发送的令牌的标识符。该值必须是包含至少一个字符且不超过 128 个字符的字母数字字符串，并与此正则表达式（正则表达式）模式匹配：`([a-zA-Z0-9]){1,128}`。不允许在 in `principalId` 中 AWS IoT Core使用非字母数字的特殊字符。如果允许使用非字母数字特殊字符，请参阅其他 AWS 服务的文档。`principalId`
- `policyDocuments`：JSON 格式的 AWS IoT Core 策略文档列表有关创建 AWS IoT Core 策略的更多信息，请参阅。the section called “AWS IoT Core 政策”策略文档的数量最多为 10 个。每个策略文档最多可以包含 2048 个字符。
- `disconnectAfterInSeconds`：指定与 AWS IoT Core 网关的连接的最大持续时间（以秒为单位）的整数。最小值为 300 秒，最大值为 86400 秒。默认值为 86,400。

  > **Note**
  >
  > 的值`disconnectAfterInSeconds`（由 Lambda 函数返回）是在连接建立时设置的。在后续的策略刷新 Lambda 调用过程中，无法修改此值。

- `refreshAfterInSeconds`：指定策略刷新之间间隔的整数。当此时间间隔过去时， AWS IoT Core 将调用 Lambda 函数以允许策略刷新。最小值为 300 秒，最大值为 86400 秒。

以下 JSON 对象包含 Lambda 函数可以发送的响应示例。

```
{
"isAuthenticated":true, //A Boolean that determines whether client can connect.
"principalId": "xxxxxxxx",  //A string that identifies the connection in logs.
"disconnectAfterInSeconds": 86400,
"refreshAfterInSeconds": 300,
 "policyDocuments": [
      {
        "Version": "2012-10-17",
        "Statement": [
           {
              "Action": "iot:Publish",
              "Effect": "Allow",
              "Resource": "arn:aws:iot:us-east-1:<your_aws_account_id>:topic/
customauthtesting"
            }
         ]
       }
    ]
}
```

该policyDocument值必须包含有效的 AWS IoT Core 策略文档。有关 AWS IoT Core 策略的更多信息，请参阅the section called “AWS IoT Core 政策”。 在 TLS 上的 MQTT 和通过 WebSockets连接的 MQ AWS IoT Core TT 中，在字段值中指定的间隔内缓存此策略。refreshAfterInSeconds在 HTTP 连接的情况下，每个授权请求都会调用 Lambda 函数，除非您的设备使用 HTTP 持久连接（也称为 HTTP 保持活动状态或 HTTP 连接重用），否则您可以在配置授权方时选择启用缓存。在此间隔内， AWS IoT Core 授权在已建立的连接中针对此缓存策略执行操作，而无需再次触发您的 Lambda 函数。如果在自定义身份验证期间出现故障，则 AWS IoT Core 终止连接。 AWS IoT Core 如果连接的打开时间超过disconnectAfterInSeconds参数中指定的值，也会终止连接。

以下内容 JavaScript 包含一个 Node.js Lambda 函数示例，该函数在 MQTT Connect 消息中查找值为test的密码，并返回一个策略，该策略授予使用myClientName名为的客户端进行连接 AWS IoT Core 并发布到包含相同客户端名称的主题的权限。如果未找到预期密码，则返回拒绝这两个操作的策略。

```
// A simple Lambda function for an authorizer. It demonstrates
// how to parse an MQTT password and generate a response.

exports.handler = function(event, context, callback) {
    var uname = event.protocolData.mqtt.username;
```

```
    var pwd = event.protocolData.mqtt.password;
    var buff = new Buffer(pwd, 'base64');
    var passwd = buff.toString('ascii');
    switch (passwd) {
        case 'test':
            callback(null, generateAuthResponse(passwd, 'Allow'));
            break;
        default:
            callback(null, generateAuthResponse(passwd, 'Deny'));
    }
};

// Helper function to generate the authorization response.
var generateAuthResponse = function(token, effect) {
    var authResponse = {};
    authResponse.isAuthenticated = true;
    authResponse.principalId = 'TEST123';

    var policyDocument = {};
    policyDocument.Version = '2012-10-17';
    policyDocument.Statement = [];
    var publishStatement = {};
    var connectStatement = {};
    connectStatement.Action = ["iot:Connect"];
    connectStatement.Effect = effect;
    connectStatement.Resource = ["arn:aws:iot:us-east-1:123456789012:client/
myClientName"];
    publishStatement.Action = ["iot:Publish"];
    publishStatement.Effect = effect;
    publishStatement.Resource = ["arn:aws:iot:us-east-1:123456789012:topic/telemetry/
myClientName"];
    policyDocument.Statement[0] = connectStatement;
    policyDocument.Statement[1] = publishStatement;
    authResponse.policyDocuments = [policyDocument];
    authResponse.disconnectAfterInSeconds = 3600;
    authResponse.refreshAfterInSeconds = 300;

    return authResponse;
}
```

上述 Lambda 函数在收到 MQTT Connect 消息中的预期密码 `test` 时返回以下 JSON。`password` 和 `principalId` 属性的值将是来自 MQTT Connect 消息的值。

```
{
  "password": "password",
  "isAuthenticated": true,
  "principalId": "principalId",
  "policyDocuments": [
    {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Action": "iot:Connect",
          "Effect": "Allow",
          "Resource": "*"
        },
        {
          "Action": "iot:Publish",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topic/telemetry/${iot:ClientId}"
        },
        {
          "Action": "iot:Subscribe",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topicfilter/telemetry/
${iot:ClientId}"
        },
        {
          "Action": "iot:Receive",
          "Effect": "Allow",
          "Resource": "arn:aws:iot:region:accountId:topic/telemetry/${iot:ClientId}"
        }
      ]
    }
  ],
  "disconnectAfterInSeconds": 3600,
  "refreshAfterInSeconds": 300
}
```

创建授权方

您可以使用 CreateAuthorizerAP I 创建授权方。 以下示例描述了该命令。

```
aws iot create-authorizer
--authorizer-name MyAuthorizer
```

```
--authorizer-function-arn arn:aws:lambda:us-
west-2:<account_id>:function:MyAuthorizerFunction  //The ARN of the Lambda function.
[--token-key-name MyAuthorizerToken //The key used to extract the token from headers.
[--token-signing-public-keys FirstKey=
 "-----BEGIN PUBLIC KEY-----
  [...insert your public key here...]
  -----END PUBLIC KEY-----"
[--status ACTIVE]
[--tags <value>]
[--signing-disabled | --no-signing-disabled]
```

您可以使用 signing-disabled 参数选择退出适用于授权方的每次调用的签名验证。除有必要，否则我们强烈建议您不要禁用签名。签名验证可防止来自未知设备的 Lambda 函数过多调用。您无法在创建授权方后更新其 signing-disabled 状态。要更改此行为，您必须创建 signing-disabled 参数具有不同值的其它自定义授权方。

如果您已禁用签名，则 tokenKeyName 和 tokenSigningPublicKeys 参数的值是可选的。如果启用了签名，则它们是必需的值。

创建 Lambda 函数和自定义授权方后，您必须明确授予 AWS IoT Core 服务权限才能代表您调用该函数。 您可以使用以下命令执行此操作。

```
aws lambda add-permission --function-name <lambda_function_name> --
principal iot.amazonaws.com --source-arn <authorizer_arn> --statement-id
Id-123 --action "lambda:InvokeFunction"
```

测试授权方

您可以使用 A TestInvokeuthorizer API 来测试授权方的调用和返回值。 此 API 使您能够在授权方中指定协议元数据并测试签名验证。

以下选项卡显示了如何使用 AWS CLI 来测试您的授权者。

Unix-like

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER \
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Windows CMD

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER ^
```

```
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

Windows PowerShell

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER `
--token TOKEN_VALUE --token-signature TOKEN_SIGNATURE
```

`token-signature` 参数的值是签名令牌。要了解如何获取此值，请参阅 the section called “签名令牌”。

如果您的授权方使用了用户名和密码，您可以使用 `--mqtt-context` 参数传递此信息。以下选项卡显示如何使用 `TestInvokeAuthorizer` API 将包含用户名、密码和客户端名称的 JSON 对象发送到您的自定义授权方。

Unix-like

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  \
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

Windows CMD

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  ^
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

Windows PowerShell

```
aws iot test-invoke-authorizer --authorizer-name NAME_OF_AUTHORIZER  `
--mqtt-context '{"username": "USER_NAME", "password": "dGVzdA==",
 "clientId":"CLIENT_NAME"}'
```

密码必须采用 base64 编码。以下示例说明如何在类 Unix 的环境中对密码进行编码。

```
echo -n PASSWORD | base64
```

管理自定义授权方

您可以使用以下 API 管理授权方。

- ListAuthorizers: 显示您账户中的所有授权人。
- DescribeAuthorizer：显示指定授权者的属性。这些值包括创建日期、上次修改日期和其它属性。
- SetDefault授权者：为您的 AWS IoT Core 数据端点指定默认授权方。 AWS IoT Core 如果设备未通过 AWS IoT Core 凭证且未指定授权方，则使用此授权方。有关使用 AWS IoT Core 证书的更多信息，请参阅the section called “客户端身份验证”。
- UpdateAuthorizer：更改指定授权方的状态、令牌密钥名称或公钥。
- DeleteAuthorizer：删除指定的授权者。

> **Note**
>
> 您无法更新授权方的签名要求。这意味着您无法禁用需要签名的现有授权方的中签名。您也不能在不需要签名的现有授权方中要求签名。

使用自定义身份验证 AWS IoT Core 进行连接

设备可以通过自定义身份验证与任何 AWS IoT Core 支持设备消息传递的协议进行连接。 AWS IoT Core 有关受支持的通信协议的更多信息，请参阅 the section called “设备通信协议”。 您传递给授权方 Lambda 函数的连接数据取决于您使用的协议。有关创建授权方 Lambda 函数的更多信息，请参阅 the section called “定义您的 Lambda 函数”。以下部分说明如何使用每个支持的协议连接到身份验证。

HTTPS

使用 HTTP 发布 API 向 AWS IoT Core 其发送数据的设备可以在其 HTTP P OST 请求中通过请求标头或查询参数传递凭证。设备可以使用 `x-amz-customauthorizer-name` 标头或查询参数指定要调用的授权方。如果您在授权方中启用了令牌签名，则必须使用请求标头或查询参数传递 *`token-key-name`* 和 `x-amz-customauthorizer-signature`。请注意，在*`token-signature`*浏览器中使用 JavaScript 该值时必须经过网址编码。

> **Note**
>
> HTTPS 协议的客户授权方仅支持发布操作。有关 HTTPS 协议的更多信息，请参阅the section called “设备通信协议”。

以下示例请求显示了如何使用请求标头和查询参数传递这些参数。

```
//Passing credentials via headers
POST /topics/topic?qos=qos HTTP/1.1
Host: your-endpoint
x-amz-customauthorizer-signature: token-signature
token-key-name: token-value
x-amz-customauthorizer-name: authorizer-name

//Passing credentials via query parameters
POST /topics/topic?qos=qos&x-amz-customauthorizer-signature=token-signature&token-key-name=token-value HTTP/1.1
```

MQTT

使用 MQTT 连接连接的设备可以通过 MQTT 消息的username和password字段传递凭证。 AWS IoT Core username 值也可以选择将其他值（包括令牌、签名和授权方名称）传递给授权方的查询字符串包含在内。如果要使用基于令牌的身份验证方案而不是 username 和 password 值，则可以使用此查询字符串。

> **Note**
>
> 密码字段中的数据由 base64 编码。 AWS IoT Core您的 Lambda 函数必须对其进行解码。

以下示例包含一个 username 字符串，其中带有指定令牌和签名的额外参数。

```
username?x-amz-customauthorizer-name=authorizer-name&x-amz-customauthorizer-signature=token-signature&token-key-name=token-value
```

要调用授权方，使用 MQTT 和自定义身份验证连接的设备必须 AWS IoT Core 通过端口 443 进行连接。它们还必须通过应用层协议协商 (ALPN) TLS 扩展（值为）mqtt和服务器名称指示 (SNI) 扩展名及其 AWS IoT Core 数据端点的主机名。为避免潜在的错误，x-amz-customauthorizer-signature 的值应采用 URL 编码。我们还强烈建议 x-amz-customauthorizer-name 和 token-key-name 的值采用 URL 编码。有关这些值的更多信息，请参阅the section called “设备通信协议”。V2 AWS IoT 设备 SDK、移动 SDK 和 AWS IoT 设备客户端 可以配置这两个扩展。

MQTT 结束了 WebSockets

使用 MQTT 连接的设备 WebSockets 可以通过以下两种方式之一传递凭证。 AWS IoT Core

- 通过 HTTP UPGRADE 请求中的请求标头或查询参数来建立 WebSockets 连接。
- 通过 `username` 和 `password` 字段中的 MQTT CONNECT 消息。

如果您通过 MQTT 连接消息传递凭证，则需要使用 ALPN 和 SNI TLS 扩展。有关这些扩展的更多信息，请参阅 the section called “MQTT”。以下示例演示如何通过 HTTP Upgrade 请求传递凭证。

```
GET /mqtt HTTP/1.1
Host: your-endpoint
Upgrade: WebSocket
Connection: Upgrade
x-amz-customauthorizer-signature: token-signature
token-key-name: token-value
sec-WebSocket-Key: any random base64 value
sec-websocket-protocol: mqtt
sec-WebSocket-Version: websocket version
```

签名令牌

必须使用 `create-authorizer` 调用中用到的公私密钥对的私有秘钥签署令牌。以下示例说明如何使用类似 Unix 的命令创建令牌签名和。 JavaScript它们使用 SHA-256 哈希算法对签名进行编码。

Command line

```
echo -n TOKEN_VALUE | openssl dgst -sha256 -sign PEM encoded RSA private key |
 openssl base64
```

JavaScript

```
const crypto = require('crypto')

const key = "PEM encoded RSA private key"

const k = crypto.createPrivateKey(key)
let sign = crypto.createSign('SHA256')
sign.write(t)
sign.end()
const s = sign.sign(k, 'base64')
```

授权方故障排除

本主题介绍可能导致自定义身份验证工作流中出现故障的常见问题以及解决这些问题的步骤。要最有效地解决问题，请启用 CloudWatch 日志 AWS IoT Core 并将日志级别设置为 DEBUG。你可以在 AWS IoT Core 控制台中启用 CloudWatch 日志 (https://console.aws.amazon.com/iot/)。有关启用和配置适用于 AWS IoT Core的日志的更多信息，请参阅 the section called “配置 AWS IoT 日志”。

> **Note**
>
> 如果您将日志级别长时间保持在 DEBUG，则 CloudWatch 可能会存储大量的日志数据。这可能会增加您的 CloudWatch 费用。考虑使用基于资源的日志记录来增加仅针对特定事物组中设备的详细程度。有关基于资源的日志记录的更多信息，请参阅 the section called “配置 AWS IoT 日志”。此外，当您完成故障排除时，将日志级别降到详细程度较低的级别。

在您开始故障排除之前，请查看 the section called “了解自定义身份验证工作流” 获取自定义身份验证过程的高度概要视图。这有助于您了解应该在何处查找问题的根源。

本主题将讨论以下两个方面供您调查。

- 与授权方 Lambda 函数相关的问题。
- 与您的设备相关的问题。

检查授权方的 Lambda 函数中是否存在问题

执行以下步骤以确保设备的连接尝试正在调用 Lambda 函数。

1. 验证哪个 Lambda 函数与您的授权方相关联。

   您可以通过调用 DescribeAuthorizerAPI 或在 AWS IoT Core 控制台的 “安全” 部分中单击所需的授权者来执行此操作。
2. 检查 Lambda 函数的调用指标。为此，请执行以下步骤。

   a. 打开 AWS Lambda 控制台 (https://console.aws.amazon.com/lambda/)，然后选择与您的授权者关联的功能。

   b. 选择监控选项卡并查看与您的问题相关的时间范围的指标。
3. 如果您没有看到任何调用，请验证是否 AWS IoT Core 有权调用您的 Lambda 函数。如果您看到调用，则跳到下一步。执行以下步骤以验证 Lambda 函数是否具有所需的权限。

a. 在 AWS Lambda 控制台中为您的函数选择 “权限” 选项卡。

b. 在页面底部找到基于资源的策略部分。如果您的 Lambda 函数具有所需的权限，则策略类似于以下示例。

```
{
  "Version": "2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "Id123",
      "Effect": "Allow",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-
east-1:111111111111:function:FunctionName",
      "Condition": {
        "ArnLike": {
          "AWS:SourceArn": "arn:aws:iot:us-east-1:111111111111:authorizer/
AuthorizerName"
        },
        "StringEquals": {
          "AWS:SourceAccount": "111111111111"
        }
      }
    }
  ]
}
```

c. 此政策将您的职能`InvokeFunction`权限授予 AWS IoT Core 委托人。如果您没有看到它，则必须使用 AddPermissionAPI 进行添加。以下示例说明如何使用 AWS CLI执行此操作。

```
aws lambda add-permission --function-name FunctionName --principal
 iot.amazonaws.com --source-arn AuthorizerARn --statement-id Id-123 --action
 "lambda:InvokeFunction"
```

4. 如果您看到调用，请验证没有错误。错误可能表明 Lambda 函数未正确处理 AWS IoT Core 发送给它的连接事件。

有关在 Lambda 函数中处理事件的信息，请参阅 the section called “定义您的 Lambda 函数”。您可以使用 AWS Lambda 控制台中的测试功能 (https://console.aws.amazon.com/lambda/) 对函数中的测试值进行硬编码，以确保该函数正确处理事件。

5. 如果您看到没有错误的调用，但您的设备无法连接（或发布、订阅和接收消息），则问题可能是 Lambda 函数返回的策略未授予设备尝试执行的操作的权限。执行以下步骤以确定函数返回的策略是否存在任何问题。

   a. 使用 Amazon CloudWatch Logs Insights 查询在短时间内扫描日志，以检查是否存在故障。以下示例查询按时间戳对事件进行排序并查找失败。

   ```
   display clientId, eventType, status, @timestamp | sort @timestamp desc | filter
    status = "Failure"
   ```

   b. 更新您的 Lambda 函数以记录它返回的数据 AWS IoT Core 以及触发该函数的事件。您可以使用这些日志检查函数创建的策略。

6. 如果您看到调用并没有错误，但您的设备无法连接（或发布、订阅和接收消息），则另一个原因可能是 Lambda 函数超出了超时限制。自定义授权方的 Lambda 函数超时限制为 5 秒。您可以在 CloudWatch 日志或指标中查看函数持续时间。

调查设备问题

如果您发现调用 Lambda 函数或函数返回的策略不存在任何问题，请查找设备连接尝试是否存在问题。格式错误的连接请求可能导致 AWS IoT Core 无法触发您的授权者。TLS 层和应用程序层均可能出现连接问题。

TLS 层可能存在的问题：

- 客户必须在所有自定义身份验证请求中传递主机名标头（HTTP、MQTT WebSockets）或服务器名称指示 TLS 扩展（HTTP、MQTT over WebSockets、MQTT）。在这两种情况下，传递的值都必须与您账户 AWS IoT Core 的数据端点之一匹配。这些是执行以下 CLI 命令时返回的终端节点。
  - `aws iot describe-endpoint --endpoint-type iot:Data-ATS`
  - `aws iot describe-endpoint --endpoint-type iot:Data`（适用于传统 VeriSign 端点）
- 在 MQTT 连接中使用自定义身份验证的设备还必须发送应用程序层协议协商 (ALPN) TLS 扩展，并具有值 `mqtt`。
- 自定义身份验证当前仅在端口 443 上可用。

应用层可能存在的问题：

- 如果启用了签名（`signingDisabled` 字段在您的授权方为 false），请查找以下签名问题。
  - 请确保使用 `x-amz-customauthorizer-signature` 标头或查询字符串参数传递令牌签名。
  - 确保服务没有签署令牌以外的值。
  - 请确保使用您在授权方的 `token-key-name` 字段中指定的标头或查询参数中传递令牌。
- 请确保您使用 `x-amz-customauthorizer-name` 标头或查询字符串参数传递的授权方名称是有效的，或者您已为您的账户定义了默认授权方。

授权

授权是向经过身份验证的身份授予权限的过程。您授予 AWS IoT Core 使用 AWS IoT Core 和 IAM 策略的权限。本主题介绍了 AWS IoT Core 策略。有关 IAM policy 的更多信息，请参阅 的身份和访问管理 AWS IoT 和 如何 AWS IoT 与 IAM 配合使用。

AWS IoT Core 策略决定了经过身份验证的身份可以做什么。经身份验证的身份由设备、移动应用程序、Web 应用程序和桌面应用程序使用。经过身份验证的身份甚至可以是用户键入 AWS IoT Core CLI 命令。只有当身份拥有向其授予 AWS IoT Core 操作权限的策略时，该身份才能执行这些操作。

AWS IoT Core 策略和 IAM 策略均 AWS IoT Core 用于控制身份（也称为委托人）可以执行的操作。您使用的策略类型取决于您用来进行身份验证的身份类型 AWS IoT Core。

AWS IoT Core 操作分为两组：

- 控制面板 API 允许您执行诸如创建或更新证书、事物、规则等管理任务。
- 数据平面 API 允许您向数据发送和接收数据 AWS IoT Core。

您使用的策略类型取决于您使用的是控制面板 API 还是数据层面 API。

下表显示了身份类型、它们使用的协议和可用于授权的策略类型。

AWS IoT Core 数据平面 API 和策略类型

协议和身份验证机制	SDK	身份类型	策略类型		
基于 TLS/TCP、TLS 双向身份验证的 MQTT（端口 8883 或 443）[‡])	AWS IoT 设备软件开发工具包	X.509 证书	AWS IoT Core 政策		
基于 HTTPS/ 的 MQTTWebSocket，AWS Sigv4 身份验证（端口 443）	AWS 移动 SDK	经过身份验证的 Amazon Cognito 身份	IAM 和 AWS IoT Core 政策		
		未经身份验证的 Amazon Cognito 身份	IAM policy		
		IAM 或联合身份	IAM policy		
HTTPS，AWS 签名版本 4 身份验证（端口 443）	AWS CLI	Amazon Cognito、IAM 或联合身份	IAM policy		
HTTPS、TLS 双向身份验证（端口 8443）	不支持 SDK	X.509 证书	AWS IoT Core 政策		
基于自定义身份验证的 HTTPS（端口 443）	AWS IoT 设备软件开发工具包	自定义授权方	自定义授权方策略		

AWS IoT Core 控制平面 API 和策略类型

协议和身份验证机制	SDK	身份类型	策略类型		
HTTPS AWS 签名版本 4 身份验证（端口 443）	AWS CLI	Amazon Cognito 身份	IAM policy		
		IAM 或联合身份	IAM policy		

AWS IoT Core 策略附加到 X.509 证书、Amazon Cognito 身份或事物组。IAM policy 附加到 IAM 用户、组或角色。如果您使用 AWS IoT 控制台或 AWS IoT Core CLI 附加策略（到证书、Amazon Cognito Identity 或事物组），则 AWS IoT Core 使用策略。否则，您将使用 IAM 策略。 AWS IoT Core 附加到事物组的策略适用于该事物组中的任何事物。要使 AWS IoT Core 策略生效，`clientId`和事物名称必须匹配。

基于策略的授权功能强大。它使您能够完全控制设备、用户或应用程序可在 AWS IoT Core中执行的操作。例如，假设使用证书连接 AWS IoT Core 的设备。您可以允许设备访问所有 MQTT 主题，也可以限制它的访问权限，只允许它访问一个主题。再举一例，假设用户在命令行中键入 CLI 命令。通过使用策略，您可以允许或拒绝用户访问任何命令或 AWS IoT Core 资源。此外，您还可以控制应用程序对 AWS IoT Core 资源的访问。

对策略所做的更改可能需要几分钟才能生效，具体取决于 AWS IoT 缓存策略文档的方式。也就是说，访问最近被授予访问权限的资源可能需要几分钟时间，并且资源可能在撤销访问权限后几分钟内仍可访问。

AWS 培训和认证

有关授权的信息 AWS IoT Core，请参加 AWS 培训和认证网站上的 “深入了解 AWS IoT Core 身份验证和授权” 课程。

AWS IoT Core 政策

AWS IoT Core 策略是 JSON 文档。它们遵循与 IAM 策略相同的惯例。 AWS IoT Core 支持命名策略，因此许多身份可以引用同一个策略文档。命名策略采用版本化，以便可以轻松回滚。

AWS IoT Core 策略允许您控制对 AWS IoT Core 数据平面的访问。 AWS IoT Core 数据层面由以下操作组成，允许您连接到 AWS IoT Core 消息代理，发送和接收 MQTT 消息以及获取或更新事物的 Device Shadow。

AWS IoT Core 策略是包含一个或多个政策声明的 JSON 文档。每个语句包含：

- `Effect`，指定是允许还是拒绝该操作。
- `Action`，用于指定策略允许或拒绝的操作。
- `Resource`，用于指定允许或拒绝对其执行操作的资源。

由于策略文档的 AWS IoT 缓存方式不同，对策略所做的更改可能需要 6 到 8 分钟才能生效。也就是说，访问最近被授予访问权限的资源可能需要几分钟时间，并且资源可能在撤销访问权限后几分钟内仍可访问。

AWS IoT Core 策略可以附加到 X.509 证书、Amazon Cognito 身份和事物组。附加到事物组的策略适用于该组中的任何事物。要使策略生效，`clientId` 和事物名称必须匹配。 AWS IoT Core 策略遵循与 IAM policy 相同的策略评估逻辑。默认情况下，所有策略都被隐式拒绝。任何基于身份或基于资源的策略中的显式允许将覆盖此默认行为。任何策略中的显式拒绝将覆盖任何允许。有关更多信息，请参阅《AWS Identity and Access Management 用户指南》中的策略评估逻辑。

主题

AWS IoT Core 政策行动

AWS IoT Core定义了以下策略操作：

MQTT 策略操作

`iot:Connect`

表示连接到 AWS IoT Core 消息代理的权限。每次向代理发送 `iot:Connect` 请求时，均检查 CONNECT 权限。消息代理禁止两个具有相同 ID 的客户端同时保持连接状态。在第二个客户端连接后，代理将关闭现有连接。`iot:Connect` 权限可用来确保仅使用特定客户端 ID 的授权客户端可以连接。

`iot:GetRetainedMessage`

代表获取保留单条消息内容的权限。保留的消息是在设置了 RETAIN 标志的情况下发布并由存储的消息 AWS IoT Core。有关获取账户所有保留消息列表的权限，请参阅 `iot:ListRetainedMessages`。

`iot:ListRetainedMessages`

表示检索有关帐户保留邮件的摘要信息（而不是邮件内容）的权限。保留的消息是在设置了 RETAIN 标志的情况下发布并由存储的消息 AWS IoT Core。为此操作指定的资源 ARN 必须为 *。有关获取保留单条消息内容的权限，请参阅 `iot:GetRetainedMessage`。

`iot:Publish`

代表向 MQTT 主题发布内容的权限。每次向代理发送 PUBLISH 请求时，均检查该权限。您可以使用此权限让客户端发布到特定主题模式。

> **Note**
>
> 要授予 `iot:Publish` 权限，还必须授予 `iot:Connect` 权限。

`iot:Receive`

表示接收来自的消息的权限 AWS IoT Core。每次向客户端交付消息时，都确认 `iot:Receive` 权限。由于每次交付时都会检查此权限，因此，可以使用它来吊销当前已订阅某个主题的客户端的权限。

`iot:RetainPublish`

表示使用 RETAIN 标志集发布 MQTT 消息的权限。

> **Note**
>
> 要授予 `iot:RetainPublish` 权限，还必须授予 `iot:Publish` 权限。

`iot:Subscribe`

代表订阅主题筛选条件的权限。每次向代理发送 SUBSCRIBE 请求时，均检查该权限。用此权限让客户端可以订阅与特定主题模式相符的主题。

> **Note**
>
> 要授予 `iot:Subscribe` 权限，还必须授予 `iot:Connect` 权限。

Device Shadow 策略操作

`iot:DeleteThingShadow`

表示删除事物的 Device Shadow 的权限。每次提出请求删除事物的 Device Shadow 内容时，都会检查 `iot:DeleteThingShadow` 权限。

`iot:GetThingShadow`

表示检索事物的 Device Shadow 的权限。每次请求检索 Device Shadow 的内容时，都会检查 `iot:GetThingShadow` 权限。

`iot:ListNamedShadowsForThing`

表示列出名为 Shadows 的事物的权限。每次请求列示名为 Shadows 的事物时，都会检查 `iot:ListNamedShadowsForThing` 权限。

`iot:UpdateThingShadow`

表示更新设备的影子的权限。每次请求更新事物的 Device Shadow 的内容时，都会检查 `iot:UpdateThingShadow` 权限。

> **Note**
>
> 任务执行策略操作仅适用于 HTTP TLS 终端节点。如果您使用了 MQTT 终端节点，则必须使用本主题中定义的 MQTT 策略操作。

有关演示此操作的任务执行策略的示例，请参阅使用 MQTT 协议的 the section called “基本任务策略示例”。

Job 执行 AWS IoT Core 策略操作

`iot:DescribeJobExecution`

表示为给定事物检索任务执行的权限。每次请求获取任务执行时，都会检查 `iot:DescribeJobExecution` 权限。

`iot:GetPendingJobExecutions`

表示一个权限，用于为事物检索未处于最终状态的任务的列表。每次请求检索该列表时，都会检查 `iot:GetPendingJobExecutions` 权限。

`iot:UpdateJobExecution`

表示更新任务执行的权限。每次请求更新任务执行的状态时，都会检查 `iot:UpdateJobExecution` 权限。

`iot:StartNextPendingJobExecution`

表示一个权限，用于为事物获取和启动下一个待处理任务执行。（即，将状态为 QUEUED 的任务执行更新为状态 IN_PROGRESS。）每次请求启动下一个待处理任务执行，都会检查 `iot:StartNextPendingJobExecution` 权限。

AWS IoT Core 凭证提供商政策操作

`iot:AssumeRoleWithCertificate`

表示允许通过基于证书的身份验证调用 AWS IoT Core 凭证提供商来担任 IAM 角色。每次向 AWS IoT Core 凭证提供者提出担任角色的请求时，都会检查`iot:AssumeRoleWithCertificate`权限。

AWS IoT Core 动作资源

要为 AWS IoT Core 策略操作指定资源，请使用该资源的 Amazon 资源名称 (ARN)。所有资源 ARN 都遵循以下格式：

```
arn:partition:iot:region:AWS-account-ID:Resource-type/Resource-name
```

下表显示了要为每种操作类型指定的资源。ARN 示例适用于账户 ID 123456789012 aws、分区以及特定于该区域。us-east-1有关 ARN 格式的更多信息，请参阅 AWS Identity and Access Management 用户指南中的 Amazon 资源名称 (ARN)。

操作	资源类型	资源名称	ARN 示例
iot:Connect	client	客户端的客户端 ID	arn:aws:iot:us-east-1:123456789012:client/myClientId
iot:DeleteThingShadow	thing	事物的名称和影子的名称（如适用）	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne
iot:DescribeJobExecution	thing	事物名称	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:GetPendingJobExecutions	thing	事物名称	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:GetRetainedMessage	topic	保留的消息主题	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:GetThingShadow	thing	事物的名称和影子的名称（如适用）	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne
iot:ListNamedShadowsForThing	全部	全部	*

操作	资源类型	资源名称	ARN 示例
iot:ListRetainedMessages	全部	全部	*
iot:Publish	topic	主题字符串	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:Receive	topic	主题字符串	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:RetainPublish	topic	用 REATIN 标志集发布的主题	arn:aws:iot:us-east-1:123456789012:topic/myTopicName
iot:StartNextPendingJobExecution	thing	事物名称	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:Subscribe	topicfilter	主题筛选条件字符串	arn:aws:iot:us-east-1:123456789012:topicfilter/myTopicFilter
iot:UpdateJobExecution	thing	事物名称	arn:aws:iot:us-east-1:123456789012:thing/thingOne
iot:UpdateThingShadow	thing	事物的名称和影子的名称（如适用）	arn:aws:iot:us-east-1:123456789012:thing/thingOne arn:aws:iot:us-east-1:123456789012:thing/thingOne/shadowOne

操作	资源类型	资源名称	ARN 示例
iot:AssumeRoleWithCertificate	rolealias	指向角色 ARN 的角色别名	arn:aws:iot:us-east-1:123456789012:rolealias/CredentialProviderRole_alias

AWS IoT Core 策略变量

AWS IoT Core 定义了可以在Resource或Condition块中的 AWS IoT Core 策略中使用的策略变量。评估策略时，将使用实际值替换策略变量。例如，如果设备连接到客户端 ID 为 100-234-3456 的 AWS IoT Core 消息代理，则策略文档中的iot:ClientId策略变量将被替换为 100-234-3456。

AWS IoT Core 策略可以使用通配符并遵循与 IAM 策略类似的惯例。在字符串中插入 *（星号）可以被视为通配符，匹配任何字符。例如，可以使用 * 在策略的 Resource 属性中描述多个 MQTT 主题名称。字符 + 和 # 在策略中被视为文字字符串。有关显示如何使用通配符的示例策略，请参阅在 MQTT 和 AWS IoT Core 策略中使用通配符。

还可以使用具有固定值的预定义策略变量来表示字符（这些字符本身有特殊的含义）。这些特殊字符包括 ${*}、${?} 和 ${$}。有关策略变量和特殊字符的更多信息，请参阅 IAM policy 元素：变量和标签和创建具有多个键或值的条件。

主题

- 基本 AWS IoT Core 策略变量
- 事物策略变量
- X.509 证书策略变量 AWS IoT Core

基本 AWS IoT Core 策略变量

AWS IoT Core 定义了以下基本策略变量：

- iot:ClientId：用于连接到 AWS IoT Core 消息代理的客户端 ID。
- aws:SourceIp：连接到 AWS IoT Core 消息代理的客户端的 IP 地址。

以下 AWS IoT Core 策略显示了使用策略变量的策略。 aws:SourceIp可以在策略的条件元素中使用，以允许委托人仅在特定地址范围内发出 API 请求。有关示例，请参阅向用户和云服务授权以使用 AWS IoT Jobs。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/clientid1"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
   ],
   "Condition": {
    "IpAddress": {
     "aws:SourceIp": "123.45.167.89"
    }
   }
  }
 ]
}
```

在这些示例中${iot:ClientId}，将替换为评估策略时连接到 AWS IoT Core 消息代理的客户端 ID。使用 ${iot:ClientId} 等策略变量时，您可能会无意中开放对意外主题的访问权限。例如，如果您使用借助 ${iot:ClientId} 来指定主题筛选条件的策略：

```
{
    "Effect": "Allow",
    "Action": ["iot:Subscribe"],
    "Resource": [
      "arn:aws:iot:us-east-1:123456789012:topicfilter/my/${iot:ClientId}/topic"
```

```
    ]
}
```

客户端可使用 + 作为客户端 ID 来进行连接。这样，用户便可以订阅与主题筛选条件 my/+/topic 匹配的任何主题。要防范此类安全漏洞，请使用 iot:Connect 策略操作来控制哪些客户端 ID 可以连接。例如，此策略仅允许那些客户端 ID 为 clientid1 的客户端建立连接：

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": ["iot:Connect"],
        "Resource": [
          "arn:aws:iot:us-east-1:123456789012:client/clientid1"
        ]
      }
    ]
}
```

> **Note**
>
> 不建议将策略变量 ${iot:ClientId} 与 Connect 一起使用。不检查 ClientId 的值，因此，具有不同客户端 ID 的附加程序可以通过验证，但会导致连接断开。由于允许任何 ClientId，因此设置随机客户端 ID 可以绕过事物组策略。

事物策略变量

事物策略变量允许您编写基于事物名称、事物类型和事物属性值等事物属性授予或拒绝权限的 AWS IoT Core 策略。您可以使用事物策略变量来应用相同的策略来控制许多 AWS IoT Core 设备。有关设备预调配的更多信息，请参阅设备预调配。事物名称是从事物连接到时发送的 MQTT Connect 消息中的客户端 ID 中获取的 AWS IoT Core。

在 AWS IoT Core 策略中使用事物策略变量时，请记住以下事项。

- 使用AttachThing委托人 API 将证书或委托人（经过身份验证的 Amazon Cognito 身份）附加到事物。
- 当您将事物名称替换为事物策略变量时，MQTT 连接消息或 TLS 连接中 clientId 的值必须与事物名称完全匹配。

可用的事物策略变量如下：

- `iot:Connection.Thing.ThingName`

  这将解析为正在评估策略的 AWS IoT Core 注册表中事物的名称。 AWS IoT Core 使用设备在进行身份验证时出示的证书来确定使用哪个东西来验证连接。只有当设备通过协议通过 MQTT 或 MQTT 进行连接时，此策略变量才可用。 WebSocket

- `iot:Connection.Thing.ThingTypeName`

  它解析为与要评估策略的事物关联的事物类型。MQTT/ WebSocket 连接的客户端 ID 必须与事物名称相同。仅当通过协议通过 MQTT 或 MQTT 进行连接时，此策略变量才可用。 WebSocket

- `iot:Connection.Thing.Attributes[`*`attributeName`*`]`

  它解析为与要评估策略的事物关联的指定属性的值。事物最多可以具有 50 个属性。每个属性都作为策略变量提供：`iot:Connection.Thing.Attributes[`*`attributeName`*`]`，其中 *`attributeName`* 是属性的名称。MQTT/ WebSocket 连接的客户端 ID 必须与事物名称相同。此策略变量仅在通过 MQTT 或 MQTT 通过协议进行连接时可用。 WebSocket

- `iot:Connection.Thing.IsAttached`

  `iot:Connection.Thing.IsAttached: ["true"]`强制规定只有同时在委托人中注册 AWS IoT 并关联到委托人的设备才能访问策略内的权限。 AWS IoT Core 如果设备提供的证书未附加到注册表AttachThing中的物联网事物，则可以使用此变量来阻止其连接 AWS IoT Core 。此变量具有值`true`，`false`表示连接对象已使用主体 API 附加到注册表中的证书或 Amazon Cognito 身份。事物名称被视为客户端 ID。

X.509 证书策略变量 AWS IoT Core

X.509 证书策略变量有助于编写策略。 AWS IoT Core 这些策略根据 X.509 证书属性授予权限。以下各节介绍如何使用这些证书策略变量。

> **Important**
>
> 如果您的 X.509 证书不包含特定的证书属性，但在策略文档中使用了相应的证书策略变量，则策略评估可能会导致意外行为。

CertificateId

在 RegisterCertificateAPI 中，certificateId显示在响应正文中。要获取有关您的证书的信息，请使用certificateId中的DescribeCertificate。

颁发者属性

根据证书颁发者设置的证书属性，以下 AWS IoT Core 策略变量支持允许或拒绝权限。

- iot:Certificate.Issuer.DistinguishedNameQualifier
- iot:Certificate.Issuer.Country
- iot:Certificate.Issuer.Organization
- iot:Certificate.Issuer.OrganizationalUnit
- iot:Certificate.Issuer.State
- iot:Certificate.Issuer.CommonName
- iot:Certificate.Issuer.SerialNumber
- iot:Certificate.Issuer.Title
- iot:Certificate.Issuer.Surname
- iot:Certificate.Issuer.GivenName
- iot:Certificate.Issuer.Initials
- iot:Certificate.Issuer.Pseudonym
- iot:Certificate.Issuer.GenerationQualifier

使用者属性

以下 AWS IoT Core 策略变量支持根据证书颁发者设置的证书主题属性授予或拒绝权限。

- iot:Certificate.Subject.DistinguishedNameQualifier
- iot:Certificate.Subject.Country
- iot:Certificate.Subject.Organization
- iot:Certificate.Subject.OrganizationalUnit
- iot:Certificate.Subject.State
- iot:Certificate.Subject.CommonName
- iot:Certificate.Subject.SerialNumber
- iot:Certificate.Subject.Title

- iot:Certificate.Subject.Surname
- iot:Certificate.Subject.GivenName
- iot:Certificate.Subject.Initials
- iot:Certificate.Subject.Pseudonym
- iot:Certificate.Subject.GenerationQualifier

X.509 证书为这些属性提供了包含一个或多个值的选项。默认情况下，每个多值属性的策略变量会返回第一个值。例如，Certificate.Subject.Country 属性可能包含国家/地区名称列表，但在策略中进行评估时，iot:Certificate.Subject.Country 会替换为第一个国家/地区名称。

您可以使用从 1 开始的索引请求第一个值以外的特定属性值。例如，iot:Certificate.Subject.Country.1 由 Certificate.Subject.Country 属性中第二个国家/地区名称替换。如果您指定不存在的索引值（例如，如果您在只有两个值分配到属性时请求第三个值），则不会执行替换功能，并且授权将失败。您可以在策略变量名称中使用 .List 后缀指定属性的所有值。

颁发者备用名称属性

以下 AWS IoT Core 策略变量支持根据证书颁发者设置的颁发者备用名称属性授予或拒绝权限。

- iot:Certificate.Issuer.AlternativeName.RFC822Name
- iot:Certificate.Issuer.AlternativeName.DNSName
- iot:Certificate.Issuer.AlternativeName.DirectoryName
- iot:Certificate.Issuer.AlternativeName.UniformResourceIdentifier
- iot:Certificate.Issuer.AlternativeName.IPAddress

使用者备用名称属性

以下 AWS IoT Core 策略变量支持根据证书颁发者设置的主题备用名称属性授予或拒绝权限。

- iot:Certificate.Subject.AlternativeName.RFC822Name
- iot:Certificate.Subject.AlternativeName.DNSName
- iot:Certificate.Subject.AlternativeName.DirectoryName
- iot:Certificate.Subject.AlternativeName.UniformResourceIdentifier
- iot:Certificate.Subject.AlternativeName.IPAddress

其它属性

根据证书`iot:Certificate.SerialNumber`的序列号，您可以使用来允许或拒绝对 AWS IoT Core 资源的访问。`iot:Certificate.AvailableKeys` 策略变量包含具有值的所有证书策略变量的名称。

使用 X.509 证书策略变量

本主题提供有关如何使用证书策略变量的详细信息。当您创建基于 X.509 证书属性授予权限的策略时，X.509 证书 AWS IoT Core 策略变量是必不可少的。如果您的 X.509 证书不包含特定的证书属性，但在策略文档中使用了相应的证书策略变量，则策略评估可能会导致意外行为。这是因为缺少的策略变量不会在策略声明中进行评估。

本主题内容：

- X.509 证书示例
- 使用证书颁发者属性作为证书策略变量
- 使用证书主题属性作为证书策略变量
- 使用证书颁发者备用名称属性作为证书策略变量
- 使用证书主题备用名称属性作为证书策略变量
- 使用其他证书属性作为证书策略变量
- X.509 证书策略变量限制
- 使用证书策略变量的策略示例

X.509 证书示例

典型的 X.509 证书可能如下所示。此示例证书包括证书属性。在评估 AWS IoT Core 策略期间，以下证书属性将填充为证书策略变量：`Serial NumberIssuer`、`Subject`、`X509v3 Issuer Alternative Name`、和`X509v3 Subject Alternative Name`。

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            92:12:85:cb:b7:a5:e0:86
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=IoT Devices, OU=SmartHome, ST=WA, CN=IoT Devices Primary CA,
    GN=Primary CA1/initials=XY/dnQualifier=Example corp,
```

```
    SN=SmartHome/ title=CA1/pseudonym=Primary_CA/generationQualifier=2/serialNumber=987

        Validity
            Not Before: Mar 26 03:25:40 2024 GMT
            Not After : Apr 28 03:25:40 2025 GMT
        Subject: C=US, O=IoT Devices, OU=LightBulb, ST=NY, CN=LightBulb Device Cert,
    GN=Bulb/initials=ZZ/dnQualifier=Bulb001,
    SN=Multi Color/title=RGB/pseudonym=RGB Device/generationQualifier=4/
serialNumber=123
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    << REDACTED >>
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            X509v3 Key Usage:
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Subject Alternative Name:
                DNS:example.com, IP Address:1.2.3.4, URI:ResourceIdentifier001,
 email:device1@example.com, DirName:/C=US/O=IoT/OU=SmartHome/CN=LightBulbCert
            X509v3 Issuer Alternative Name:
                DNS:issuer.com, IP Address:5.6.7.8, URI:PrimarySignerCA,
 email:primary@issuer.com, DirName:/C=US/O=Issuer/OU=IoT Devices/CN=Primary Issuer CA
    Signature Algorithm: sha256WithRSAEncryption
         << REDACTED >>
```

使用证书颁发者属性作为证书策略变量

下表详细说明了如何在 AWS IoT Core 策略中填充证书颁发者属性。

要在策略中填充的发行者属性

证书颁发者属性	证书策略变量
• C=US • o=物联网设备 • OU= SmartHome • ST=WA • cn=IoT 设备主要 CA	• iot:Certificate.Issuer.Country = US • iot:Certificate.Issuer.Organization = IoT Devices • iot:Certificate.Issuer.OrganizationalUnit = SmartHome

证书颁发者属性	证书策略变量
• gn=primary CA1 • 首字母缩写 =XY • dnqualifier=Example • SN= SmartHome • title=ca1 • pesudonym=primary_ • GenerationQualifier=2 • 序列号=987	• iot:Certificate.Issuer.State = WA • iot:Certificate.Issuer.CommonName = IoT Devices Primary CA • iot:Certificate.Issuer.GivenName = Primary CA1 • iot:Certificate.Issuer.initials = XY • iot:Certificate.Issuer.Distinguished NameQualifier = Example corp • iot:Certificate.Issuer.Surname = SmartHome • iot:Certificate.Issuer.Title = CA1 • iot:Certificate.Issuer.Pseudonym = Primary_CA • iot:Certificate.Issuer.GenerationQualifier = 2 • iot:Certificate.Issuer.SerialNumber = 987

使用证书主题属性作为证书策略变量

下表详细说明了如何在 AWS IoT Core 策略中填充证书主题属性。

要在策略中填充的主题属性

证书主题属性	证书策略变量
• C=US • o=物联网设备 • ST=NY • CN= LightBulb 设备证书 • gn=Bulb • 首字母缩写 =ZZ • dnqualifier=bulb001 • sn=多色	• iot:Certificate.Subject.Country = US • iot:Certificate.Subject.Organization = IoT Devices • iot:Certificate.Subject.State = NY • iot:Certificate.Subject.CommonName = LightBulb Device Cert • iot:Certificate.Subject.GivenName = Bulb • iot:Certificate.Subject.initials = ZZ

证书主题属性	证书策略变量
• title=rGB • pesudonym=RGB 设备 • GenerationQualifier=4 • 序列号=123	• `iot:Certificate.Subject.DistinguishedNameQualifier = Bulb001` • `iot:Certificate.Subject.Surname = MultiColor` • `iot:Certificate.Subject.Title = RGB` • `iot:Certificate.Subject.Pseudonym = RGB Device` • `iot:Certificate.Subject.GenerationQualifier = 4` • `iot:Certificate.Subject.SerialNumber = 123`

使用证书颁发者备用名称属性作为证书策略变量

下表详细说明了如何在 AWS IoT Core 策略中填充证书颁发者备用名称属性。

要在政策中填充的发行人备用名称属性

x509v3 发行人备用名称	策略中的属性
• DNS: issuer.com • IP 地址：5.6.7.8 • URI：PrimarySignerCA • 电子邮件：primary@issuer.com • DirName: /c=us/o=issuer/ou=IoT Devices/cn=Primary Issuer CA	• `iot:Certificate.Issuer.AlternativeName.DNSName = issuer.com` • `iot:Certificate.Issuer.AlternativeName.IPAddress = 5.6.7.8` • `iot:Certificate.Issuer.AlternativeName.UniformResourceIdentifier = PrimarySignerCA` • `iot:Certificate.Issuer.AlternativeName.RFC822Name = primary@issuer.com` • `iot:Certificate.Issuer.AlternativeName.DirectoryName = cn=Primary Issuer CA,ou=IoT Devices,o=Issuer,c=US`

使用证书主题备用名称属性作为证书策略变量

下表详细说明了如何在 AWS IoT Core 策略中填充证书使用者备用名称属性。

要在策略中填充的主题备用名称属性

x509v3 主题备用名称	策略中的属性
• DNS: example.com • IP 地址：1.2.3.4 • URI：ResourceIdentifier001 • 电子邮件：device1@example.com • DirName: /c=us/o=iot/ou= / CN = C SmartHome ert LightBulb	• `iot:Certificate.Subject.AlternativeName.DNSName = example.com` • `iot:Certificate.Subject.AlternativeName.IPAddress = 1.2.3.4` • `iot:Certificate.Subject.AlternativeName.UniformResourceIdentifier = ResourceIdentifier001` • `iot:Certificate.Subject.AlternativeName.RFC822Name = device1@example.com` • `iot:Certificate.Subject.AlternativeName.DirectoryName = cn=LightBulbCert,ou=SmartHome,o=IoT,c=US`

使用其他证书属性作为证书策略变量

下表详细介绍了如何在 AWS IoT Core 策略中填充其他证书属性。

要在策略中填充的其他属性

其他证书属性	证书策略变量
`Serial Number: 92:12:85:cb:b7:a5:e0:86`	`iot:Certificate.SerialNumber = 105256223 89124227206`

X.509 证书策略变量限制

以下限制适用于 X.509 证书策略变量：

缺少策略变量

如果您的 X.509 证书不包含特定的证书属性，但在策略文档中使用了相应的证书策略变量，则策略评估可能会导致意外行为。这是因为缺少的策略变量不会在策略声明中进行评估。

证书 SerialNumber 格式

AWS IoT Core 将证书序列号视为十进制整数的字符串表示形式。例如，如果策略只允许客户端 ID 与证书序列号匹配的连接，则客户端 ID 必须是十进制格式的序列号。

通配符

如果证书属性中存在通配符，则策略变量不会被证书属性值所取代。这会将`${policy-variable}`文本留在政策文件中。这可能会导致授权失败。可以使用以下通配符：*、$、+、？和 #。

数组字段

包含数组的证书属性限制为五项。其它的项将被忽略。

字符串长度

所有字符串值的长度限制为 1024 个字符。如果证书属性包含长度超过 1024 个字符的字符串，则该策略变量不会被证书属性值所取代。这将保留`${policy-variable}`在政策文件中。这可能会导致授权失败。

特殊字符

在策略变量中使用时，任何特殊字符（例如 `,`、`"`、`\`、+、=、<、> 和 `;`）都必须使用反斜杠 (\) 作为前缀。例如，`Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US` 改为 `Amazon Web Service O\=Amazon.com Inc. L\=Seattle ST\=Washington C\=US`。

使用证书策略变量的策略示例

以下策略文档允许使用与证书序列号匹配的客户端 ID 进行连接，并允许发布到与该模式匹配的主题:`${iot:Certificate.Subject.Organization}/device-stats/${iot:ClientId}/*`.

> **Important**
>
> 如果您的 X.509 证书不包含特定的证书属性，但在策略文档中使用了相应的证书策略变量，则策略评估可能会导致意外行为。这是因为缺少的策略变量不会在策略声明中进行评估。例如，

如果您将以下策略文档附加到不包含该iot:Certificate.Subject.Organization属性的证书，则在策略评估期间将不会填充iot:Certificate.Subject.Organization证书策略变量。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Certificate.SerialNumber}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/${iot:Certificate.Subject.Organization}/
device-stats/${iot:ClientId}/*"
   ]
  }
 ]
}
```

您也可以使用空条件运算符来确保在策略评估期间填充策略中使用的证书策略变量。以下策略文档仅在存在证书序列号和证书主体公用名属性时才允许iot:Connect使用证书。

所有证书策略变量都具有字符串值，因此支持所有字符串条件运算符。

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

```
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:client/*"
    ],
    "Condition": {
     "Null": {
      "iot:Certificate.SerialNumber": "false",
      "iot:Certificate.Subject.CommonName": "false"
     }
    }
   }
  ]
}
```

防止跨服务混淆座席

混淆代理问题是一个安全问题，即没有执行操作权限的实体可能会迫使更具权限的实体执行该操作。在中 AWS，跨服务模仿可能会导致混乱的副手问题。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。可以操纵调用服务以使用其权限对另一个客户的资源进行操作，否则该服务不应有访问权限。为了防止这种情况， AWS 提供可帮助您保护所有服务的服务委托人数据的工具，这些服务委托人有权限访问账户中的资源。

要限制为资源 AWS IoT 提供其他服务的权限，我们建议在资源策略中使用aws:SourceArn和aws:SourceAccount全局条件上下文密钥。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。

防止混淆代理问题最有效的方法是使用具有资源完整 Amazon Resource Name (ARN) 的 aws:SourceArn 全局条件上下文键。对于 AWS IoT, 您aws:SourceArn必须遵守以下格式:arn:aws:iot:*region*:*account-id*:*. 请确保该***##### AWS IoT ##***相匹配，并且***## ID ## ##### ID*** 相匹配。

以下示例说明如何通过在 AWS IoT 角色信任策略中使用aws:SourceArn和aws:SourceAccount全局条件上下文密钥来防止出现混淆的代理问题。

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

```
        "Service": "iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:iot:us-east-1:123456789012:*"
        }
      }
    }
  ]
}
```

AWS IoT Core 策略示例

本节中的策略示例说明了用于完成 AWS IoT Core中常见任务的策略文档。为解决方案创建策略时，您可以将其作为启动示例。

本节中的示例使用以下策略元素：

- the section called “AWS IoT Core 政策行动”
- the section called “AWS IoT Core 动作资源”
- the section called “基于身份的策略示例”
- the section called “基本 AWS IoT Core 策略变量”
- the section called “X.509 证书策略变量 AWS IoT Core”

此节中的策略示例：

- 连接策略示例
- 发布/订阅策略示例
- 连接和发布策略示例
- 保留的消息策略示例
- 证书策略示例
- 事物策略示例
- 基本任务策略示例

连接策略示例

以下策略拒绝访问客户端 ID client1 和连接client2的权限 AWS IoT Core，同时允许设备使用客户端 ID 进行连接。客户端 ID 与在注册表中 AWS IoT Core 注册并附加到用于连接的主体上的事物的名称相匹配：

> **Note**
>
> 对于注册的设备，我们建议您将事物策略变量用于 Connect 操作，并将事物附加到用于连接的主体。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/client1",
    "arn:aws:iot:us-east-1:123456789012:client/client2"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  }
 ]
}
```

以下策略授予 AWS IoT Core 使用客户端 ID 进行连接的权限client1。此策略示例适用于未注册的设备。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/client1"
   ]
  }
 ]
}
```

MQTT 持久性会话策略示例

connectAttributes 允许您在 IAM policy 中指定要在连接消息中使用的属性，如 PersistentConnect 和 LastWill。有关更多信息，请参阅 使用 ConnectAttributes。

以下策略允许连接 PersistentConnect 特征：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
  }
 ]
```

```
}
```

以下策略不允许 PersistentConnect，但允许使用其它特征：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringNotEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
  }
 ]
}
```

上述策略也可以使用 StringEquals 表达，但允许使用包括新特征在内的任何其它特征：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
  },
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
```

```
    "ForAnyValue:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
    }
   }
  }
 ]
}
```

以下策略允许通过 PersistentConnect 和 LastWill 连接，但不允许使用任何其它新特征：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect",
      "LastWill"
     ]
    }
   }
  }
 ]
}
```

以下策略允许客户端进行干净连接，无论是否具有 LastWill，但不允许使用其它特征：

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

```
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "LastWill"
                    ]
                }
            }
        }
    ]
}
```

以下策略仅允许使用默认特征进行连接：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": []
    }
   }
  }
 ]
}
```

以下策略仅允许使用 PersistentConnect 连接，但只要连接使用 PersistentConnect，则允许使用任何新特征：

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

```
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
                }
            }
        }
    ]
}
```

以下策略规定连接必须同时使用 PersistentConnect 和 LastWill，而不允许使用新特征：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect",
      "LastWill"
     ]
    }
   }
  },
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "PersistentConnect"
     ]
```

```
    }
   }
  },
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "LastWill"
     ]
    }
   }
  },
  {
   "Effect": "Deny",
   "Action": [
    "iot:Connect"
   ],
   "Resource": "*",
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": []
    }
   }
  }
 ]
}
```

以下策略不能拥有 PersistentConnect，但可以有 LastWill，不允许使用任何其它新特征：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "iot:ConnectAttributes": [
                        "LastWill"
                    ]
                }
            }
        }
    ]
}
```

以下策略仅允许包括带有主题 "my/lastwill/topicName" 的 LastWill 客户端连接，同时允许任何使用 LastWill 主题的特征:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ArnEquals": {
                    "iot:LastWillTopic": "arn:aws:iot:region:account-id:topic/my/
lastwill/topicName"
                }
            }
```

```
        }
    ]
}
```

以下策略仅允许使用特定 LastWillTopic 的干净连接，同时允许任何使用 LastWillTopic 的特征：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:us-east-1:123456789012:client/client1",
            "Condition": {
                "ArnEquals": {
                    "iot:LastWillTopic": "arn:aws:iot:region:account-id:topic/my/
lastwill/topicName"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:ConnectAttributes": [
                        "PersistentConnect"
                    ]
                }
            }
        }
    ]
}
```

发布/订阅策略示例

您使用的策略取决于您的连接方式 AWS IoT Core。您可以使用 MQTT 客户端、HTTP 或 WebSocket。 AWS IoT Core 通过 MQTT 客户端连接时，将使用 X.509 证书进行身份验证。当你通过 HTTP 或 WebSocket 协议连接时，你就是在使用签名版本 4 和 Amazon Cognito 进行身份验证。

> **Note**
>
> 对于注册的设备，我们建议您将事物策略变量用于 `Connect` 操作，并将事物附加到用于连接的主体。

本节内容：

- 在 MQTT 和 AWS IoT Core 策略中使用通配符
- 向特定主题发布、订阅消息/从特定主题接收消息的策略
- 向具有特定前缀的主题发布、订阅消息/从具有特定前缀的主题接收消息的策略
- 向特定于每台设备的主题发布、订阅消息/从特定于每台设备的主题接收消息的策略
- 向主题名称中包含事物属性的主题发布、订阅消息/从主题名称中包含事物属性的主题接收消息的策略
- 拒绝向主题名称的子主题发布消息的策略
- 拒绝接收来自主题名称的子主题的消息的策略
- 使用 MQTT 通配符订阅主题的策略
- 适用于 HTTP 和 WebSocket客户端的策略

在 MQTT 和 AWS IoT Core 策略中使用通配符

MQTT 和 AWS IoT Core 策略具有不同的通配符，您应在仔细考虑后选择它们。在 MQTT 中，在 MQTT 主题过滤器中使用通配符+和#来订阅多个主题名称。 AWS IoT Core 策略使用*和?作为通配符并遵守 IAM 策略的惯例。在策略文档中，* 表示字符的任意组合，问号？表示任何单个字符。在策略文档中，MQTT 通配符 + 和 # 被视为没有特殊含义的字符。要在策略的 `resource` 属性中描述多个主题名称和主题筛选条件，请使用 * 和？通配符代替 MQTT 通配符。

选择要在策略文档中使用的通配符时，请考虑该*字符不局限于单个主题级别。在 MQTT 主题筛选器中，该+角色仅限于单个主题级别。为了帮助将通配符规范限制为单个 MQTT 主题筛选条件级别，请考虑使用多个？字符。有关在策略资源中使用通配符的更多信息以及它们所匹配内容的更多示例，请参阅在资源 ARN 中使用通配符。

下表显示了 MQTT 和 MQTT 客户端的 AWS IoT Core 策略中使用的不同通配符。

通配符	是 MQTT 通配符	MQTT 中的示例	是 AWS IoT Core 策略通配符吗	MQTT AWS IoT Core 客户端策略中的示例
#	是	some/#	不支持	不适用
+	是	some/+/topic	不支持	不适用
*	否	不适用	是	topicfilter/some/*/topic topicfilter/some/sensor*/topic
?	不支持	不适用	是	topic/some/?????/topic topicfilter/some/sensor???/topic

向特定主题发布、订阅消息/从特定主题接收消息的策略

以下显示的是已注册和未注册设备向名为“some_specific_topic”的主题发布、订阅消息/从名为“some_specific_topic”的主题接收消息的示例。这些示例还强调了 Publish 和 Receive 使用“主题”作为资源，Subscribe 使用“主题筛选器”作为资源。

Registered devices

对于在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用与注册表中事物名称匹配的 ClientID 进行连接。它还为名为“some_specific_topic”的主题提供 Publish、Subscribe 和 Receive 权限。

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

```
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
    ],
    "Condition": {
     "Bool": {
      "iot:Connection.Thing.IsAttached": "true"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Publish"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Subscribe"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topicfilter/some_specific_topic"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "iot:Receive"
    ],
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
    ]
   }
  ]
}
```

Unregistered devices

对于未在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用 clientID1、clientID2 或 clientID3 进行连接。它还为名为“some_specific_topic”的主题提供 Publish、Subscribe 和 Receive 权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/some_specific_topic"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
```

```
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/some_specific_topic"
            ]
        }
    ]
}
```

向具有特定前缀的主题发布、订阅消息/从具有特定前缀的主题接收消息的策略

以下显示的是已注册和未注册的设备向具有前缀“topic_prefix”的主题发布、订阅消息/从具有前缀“topic_prefix”的主题接收消息的示例。

> **Note**
>
> 请注意本示例中通配符*的使用。尽管*在单个语句中为多个主题名称提供权限很有用，但它可能会因为向设备提供超出要求的权限而导致意想不到的后果。因此，我们建议您仅在仔细考虑*后才使用通配符。

Registered devices

对于在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用与注册表中事物名称匹配的 ClientID 进行连接。它还会为具有前缀“topic_prefix”的主题提供 `Publish`、`Subscribe` 和 `Receive` 权限。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
```

```
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish",
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix*"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix*"
   ]
  }
 ]
}
```

Unregistered devices

对于未在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用 clientID1、clientID2 或 clientID3 进行连接。它还会为具有前缀“topic_prefix”的主题提供 `Publish`、`Subscribe` 和 `Receive` 权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix*"
            ]
        }
    ]
}
```

向特定于每台设备的主题发布、订阅消息/从特定于每台设备的主题接收消息的策略

以下显示的是已注册和未注册的设备向特定于给定设备的主题发布、订阅消息/从特定于给定设备的主题接收消息的示例。

Registered devices

对于在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用与注册表中事物名称匹配的 ClientID 进行连接。它提供向特定事物主题 (`sensor/device/${iot:Connection.Thing.ThingName}`) 发布内容的权限，还提供订阅特定事物主题 (`command/device/${iot:Connection.Thing.ThingName}`) 和从特定事物主题接收内容的权限。如果注册表中的事物名称为 “thing1”，则设备将能够发布到主题 “传感器/设备/事物”。该设备还将能够订阅 “命令/设备/事物” 主题并从中接收 “命令/设备/事物”。

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

```
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/sensor/device/
${iot:Connection.Thing.ThingName}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/command/device/
${iot:Connection.Thing.ThingName}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/command/device/
${iot:Connection.Thing.ThingName}"
   ]
  }
 ]
}
```

Unregistered devices

对于未在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用 clientID1、clientID2 或 clientID3 进行连接。它提供发布到客户特定主题 (sensor/device/${iot:ClientId}) 的权限，还提供订阅客户特定主题 (command/device/${iot:ClientId}) 和从此客户特定主题接收内容的权限。如果设备以 clientID1 的身份与 clientID 连接，它将能够发布到 “传感器/设备/客户端” 主题中。该设备还将能够订阅和接收该主题的内容device/clientId1/command。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/sensor/device/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/command/device/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/command/device/
${iot:Connection.Thing.ThingName}"
            ]
        }
    ]
}
```

向主题名称中包含事物属性的主题发布、订阅消息/从主题名称中包含事物属性的主题接收消息的策略

以下显示的是已注册的设备向名称中包含事物属性的主题发布、订阅消息/从名称中包含事物属性的主题接收消息的示例。

> **Note**
>
> 只有在注册 AWS IoT Core 表中注册的设备才存在事物属性。对于未注册的设备，没有相应的示例。

Registered devices

对于在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用与注册表中事物名称匹配的 ClientID 进行连接。它提供发布到主题 (`sensor/${iot:Connection.Thing.Attributes[version]}`)，以及订阅主题名称中包含事物属性的主题 (`command/${iot:Connection.Thing.Attributes[location]}`) 和从该主题接收内容的权限。如果注册表中的事物名称包含`version=v1`和`location=Seattle`，则设备将能够发布到主题 “sensor/v1”，并订阅 “Command/Seattle” 主题并从中接收。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
```

```
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/sensor/
${iot:Connection.Thing.Attributes[version]}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Subscribe"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topicfilter/command/
${iot:Connection.Thing.Attributes[location]}"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Receive"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/command/
${iot:Connection.Thing.Attributes[location]}"
   ]
  }
 ]
}
```

Unregistered devices

由于事物属性仅存在于注册 AWS IoT Core 表中注册的设备，因此对于未注册的事物，没有相应的示例。

拒绝向主题名称的子主题发布消息的策略

以下显示的是已注册和未注册的设备向除某些子主题之外的所有主题发布消息的示例。

Registered devices

对于在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用与注册表中事物名称匹配的 ClientID 进行连接。它允许发布到所有以“department/”为前缀的主题，但不允许发布到“department/admins”子主题。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/department/*"
   ]
  },
  {
```

```
   "Effect": "Deny",
   "Action": [
    "iot:Publish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/department/admins"
   ]
  }
 ]
}
```

Unregistered devices

对于未在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用 clientID1、clientID2 或 clientID3 进行连接。它允许发布到所有以“department/”为前缀的主题，但不允许发布到“department/admins”子主题。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/department/*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
```

```
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/department/admins"
            ]
        }
    ]
}
```

拒绝接收来自主题名称的子主题的消息的策略

以下显示的是已注册和未注册的设备订阅除某些子主题之外具有特定前缀的主题，以及从此类主题接收消息的示例。

Registered devices

对于在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用与注册表中事物名称匹配的 ClientID 进行连接。此策略允许设备订阅任何具有前缀“topic_prefix”的主题。通过在 `iot:Receive` 的语句中使用 `NotResource`，我们允许设备接收来自设备订阅的所有主题（前缀为“topic_prefix/restricted”的主题除外）的消息。例如，通过此策略，设备可以订阅“topic_prefix/topic1”，甚至“topic_prefix/restricted”，但是，它们只能接收来自主题“topic_prefix/topic1”的消息，而不能接收来自主题“topic_prefix/restricted”的消息。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
  },
  {
   "Effect": "Allow",
```

```
   "Action": "iot:Subscribe",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/topic_prefix/*"
  },
  {
   "Effect": "Allow",
   "Action": "iot:Receive",
   "NotResource": "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix/restricted/
*"
  }
 ]
}
```

Unregistered devices

对于未在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用 clientID1、clientID2 或 clientID3 进行连接。此策略允许设备订阅任何具有前缀“topic_prefix”的主题。通过在 `iot:Receive` 的语句中使用 `NotResource`，我们允许设备接收来自设备订阅的所有主题（前缀为“topic_prefix/restricted”的主题除外）的消息。例如，通过此政策，设备可以订阅 “topic_prefix/topic1”，甚至 “topic_prefix/restricted”。但是，他们只会收到来自 “topic_prefix/topic1” 主题的消息，不会收到来自 “topic_prefix/restricted” 主题的消息。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/
topic_prefix/*"
        },
        {
            "Effect": "Allow",
```

```
            "Action": "iot:Receive",
            "NotResource": "arn:aws:iot:us-east-1:123456789012:topic/topic_prefix/
restricted/*"
        }
    ]
}
```

使用 MQTT 通配符订阅主题的策略

MQTT 通配符 + 和 # 被视为文字字符串，但在策略中使用时不会将其视为通配符。 AWS IoT Core 在 MQTT 中，+ 和 # 仅在订阅主题筛选器时被视为通配符，但在所有其他上下文中均被视为文字字符串。我们建议您仅在仔细考虑后才将这些 MQTT 通配符用作 AWS IoT Core 策略的一部分。

以下显示了在策略中使用 MQTT 通配符的已注册和未注册事物的示例。 AWS IoT Core 这些通配符被视为文字字符串。

Registered devices

对于在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用与注册表中事物名称匹配的 ClientID 进行连接。此策略允许设备订阅主题“部门/+/员工”和“位置/#”。由于 + 和 # 在 AWS IoT Core 策略中被视为字面字符串，因此设备可以订阅 “部门/+/员工” 主题，但不能订阅 “部门/工程/员工” 主题。同样，设备可以订阅主题“位置/#”，但不能订阅主题“位置/西雅图”。但是，设备订阅了主题“部门/+/员工”后，该策略将允许它们接收来自主题“部门/工程/员工”的消息。同样，设备订阅主题“位置/#”后，它们也将收到来自主题“位置/西雅图”的消息。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ],
   "Condition": {
    "Bool": {
     "iot:Connection.Thing.IsAttached": "true"
    }
   }
```

```
  },
  {
   "Effect": "Allow",
   "Action": "iot:Subscribe",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/department/+/
employees"
  },
  {
   "Effect": "Allow",
   "Action": "iot:Subscribe",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/location/#"
  },
  {
   "Effect": "Allow",
   "Action": "iot:Receive",
   "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
  }
 ]
}
```

Unregistered devices

对于未在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用 clientID1、clientID2 或 clientID3 进行连接。此策略允许设备订阅主题“部门/+/员工”和“位置/#”。由于 + 和 # 在 AWS IoT Core 策略中被视为字面字符串，因此设备可以订阅 “部门/+/员工” 主题，但不能订阅 “部门/工程/员工” 主题。同样，设备可以订阅主题“位置/#”，但不能订阅主题“位置/西雅图”。但是，设备订阅了主题“部门/+/员工”后，该策略将允许它们接收来自主题“部门/工程/员工”的消息。同样，设备订阅主题“位置/#”后，它们也将收到来自主题“位置/西雅图”的消息。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientId1",
                "arn:aws:iot:us-east-1:123456789012:client/clientId2",
                "arn:aws:iot:us-east-1:123456789012:client/clientId3"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/department/
+/employees"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topicfilter/location/#"
        },
        {
            "Effect": "Allow",
            "Action": "iot:Receive",
            "Resource": "arn:aws:iot:us-east-1:123456789012:topic/*"
        }
    ]
}
```

适用于 HTTP 和 WebSocket客户端的策略

当你通过 HTTP 或 WebSocket 协议连接时，你就是在使用签名版本 4 和 Amazon Cognito 进行身份验证。Amazon Cognito 身份可以是经过身份验证的，也可以是未经身份验证的。经过身份验证的身份属于已通过任何受支持的身份提供商进行身份验证的用户。未经身份验证的身份通常属于未使用身份提供商进行身份验证的来宾用户。Amazon Cognito 提供了唯一标识符和 AWS 凭证来支持未经身份验证的身份。有关更多信息，请参阅 the section called “使用 Amazon Cognito 身份的授权”。

对于以下操作， AWS IoT Core 使用通过 API 附加到 Amazon Cognito 身份的 AWS IoT Core `AttachPolicy`策略。这会缩小附加到具有经过身份验证的身份的 Amazon Cognito 身份池的权限。

- `iot:Connect`
- `iot:Publish`
- `iot:Subscribe`
- `iot:Receive`
- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

这意味着 Amazon Cognito 身份需要获得 IAM 角色策略和策略的 AWS IoT Core 许可。您可以通过 API 将 IAM 角色策略附加到池中，将 AWS IoT Core 策略附加到 Amazon Cognito 身份。 AWS IoT Core `AttachPolicy`

经过身份验证和未经身份验证的用户是不同的身份类型。如果您未将 AWS IoT 策略附加到 Amazon Cognito Identity，则经过身份验证的用户将无法在中进行授权， AWS IoT 并且无法访问 AWS IoT 资源和操作。

> **Note**
>
> 对于其他 AWS IoT Core 操作或未经身份验证的身份， AWS IoT Core 不会缩小附加到 Amazon Cognito 身份池角色的权限范围。无论是对于经过身份验证的身份还是未经过身份验证的身份，这都是我们建议附加到 Amazon Cognito 池角色的最宽松的策略。

HTTP

要允许未经过身份验证的 Amazon Cognito 身份通过 HTTP 向特定于 Amazon Cognito Identity 的主题发布消息，请将以下 IAM policy 附加到 Amazon Cognito Identity 池角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"]
        }
    ]
}
```

要允许经过身份验证的用户，请使用 API 将上述策略附加到 Amazon Cognito 身份池角色和亚马逊 Cognito 身份。 AWS IoT Core AttachPolicy

> **Note**
>
> 在授权 Amazon Cognito 身份时 AWS IoT Core ，会考虑这两个策略并授予指定的最低权限。仅当两个策略都允许请求的操作时，才允许操作。如果任一策略不允许某项操作，则该操作未经授权。

MQTT

要允许未经身份验证的 Amazon Cognito 身份发布有关您账户中特定 WebSocket 于亚马逊 Cognito 身份的主题的 MQTT 消息，请将以下 IAM 策略附加到 Amazon Cognito 身份池角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${cognito-
identity.amazonaws.com:sub}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/${cognito-
identity.amazonaws.com:sub}"]
        }
    ]
}
```

要允许经过身份验证的用户，请使用 API 将上述策略附加到 Amazon Cognito 身份池角色和亚马逊 Cognito 身份。 AWS IoT Core AttachPolicy

> **Note**
>
> 在授权 Amazon Cognito 身份时 AWS IoT Core ，会同时考虑两者并授予指定的最低权限。仅当两个策略都允许请求的操作时，才允许操作。如果任一策略不允许某项操作，则该操作未经授权。

连接和发布策略示例

对于在注册 AWS IoT Core 表中注册为事物的设备，以下策略授予 AWS IoT Core 使用与事物名称匹配的客户端 ID 进行连接的权限，并限制设备只能在客户端 ID 或事物名称特定的 MQTT 主题上发布。要成功连接，必须在注册 AWS IoT Core 表中注册事物名称，并使用附加到事物的身份或委托人进行身份验证：

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action":["iot:Publish"],
        "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}"]
      },
      {
        "Effect": "Allow",
        "Action": ["iot:Connect"],
        "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
      }
    ]
}
```

对于未在注册 AWS IoT Core 表中注册为事物的设备，以下策略授予 AWS IoT Core 使用客户端 ID 连接的权限，`client1`并限制设备只能在特定于客户的 MQTT 主题上发布：

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

```
        "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${iot:ClientId}"]
      },
      {
        "Effect": "Allow",
        "Action": ["iot:Connect"],
        "Resource": ["arn:aws:iot:us-east-1:123456789012:client/client1"]
      }
    ]
}
```

保留的消息策略示例

使用保留的消息需要具体策略。保留的消息是在设置了 RETAIN 标志并由 AWS IoT Core存储的情况下发布的 MQTT 消息。本节介绍了允许常见使用保留消息的策略示例。

本节内容：

- 连接和发布保留消息的策略
- 连接和发布保留的 Will 消息的策略
- 列出和获取保留消息的策略

连接和发布保留消息的策略

对于要发布保留消息的设备，设备必须能够连接、发布（任何 MQTT 消息）和发布 MQTT 保留的消息。以下策略为主题授予了这些权限：`device/sample/configuration` 到客户端 **`device1`**。有关授予连接权限的另一个示例，请参阅 the section called “连接和发布策略示例”。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/device1"
   ]
  },
  {
   "Effect": "Allow",
   "Action": [
```

```
    "iot:Publish",
    "iot:RetainPublish"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/device/sample/configuration"
   ]
  }
 ]
}
```

连接和发布保留的 Will 消息的策略

客户端可以配置一条消息，该消息 AWS IoT Core 将在客户端意外断开连接时发布。MQTT 称这样的消息为 Will消息。客户端必须在连接权限中添加附加条件才能包含这些条件。

以下策略文档授予所有客户端连接和发布由主题、`will` 标识的 Will 消息的权限， AWS IoT Core 也将保留。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:Connect"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/device1"
   ],
   "Condition": {
    "ForAllValues:StringEquals": {
     "iot:ConnectAttributes": [
      "LastWill"
     ]
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:Publish",
    "iot:RetainPublish"
   ],
```

```
    "Resource": [
     "arn:aws:iot:us-east-1:123456789012:topic/will"
    ]
   }
  ]
}
```

列出和获取保留消息的策略

服务和应用程序可以通过调用 ListRetainedMessages和GetRetainedMessage 来访问保留的消息，而无需支持 MQTT 客户端。调用这些操作的服务和应用程序必须使用如下示例的策略进行授权。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "iot:ListRetainedMessages"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/device1"
   ],
  },
  {
   "Effect": "Allow",
   "Action": [
    "iot:GetRetainedMessage"
   ],
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/foo"
   ]
  }
 ]
}
```

证书策略示例

对于在注册 AWS IoT Core 表中注册的设备，以下策略授予 AWS IoT Core 使用与事物名称匹配的客户端 ID 进行连接的权限，以及向名称等于该设备用于进行自我身份验证certificateId的证书的主题发布的权限：

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:CertificateId}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        }
    ]
}
```

对于未在注册 AWS IoT Core 表中注册的设备，以下策略授予 AWS IoT Core 使用客户端 ID、client1client2、client3和进行连接的权限，以及发布到名称等于设备用于进行自我身份验证certificateId的证书的主题的权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:CertificateId}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
```

```
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        }
    ]
}
```

对于在注册 AWS IoT Core 表中注册的设备，以下策略授予 AWS IoT Core 使用与事物名称匹配的客户端 ID 进行连接的权限，以及向名称等于该设备用于进行身份验证的证书主题CommonName字段的主题发布权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.CommonName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        }
    ]
}
```

> **Note**
>
> 在此示例中，证书的使用者公用名用作主题标识符，并假设使用者公用名对于每个已注册的证书都是唯一的。如果证书在多个设备之间共享，则共享此证书的所有设备的使用者公用名将相同，因而允许从多个设备向同一主题发布权限（不推荐）。

对于未在注册 AWS IoT Core 表中注册的设备，以下策略授予 AWS IoT Core 使用客户端 ID、client1client2、client3和进行连接的权限，以及向名称等于该设备用于进行身份验证的证书主题CommonName字段的主题发布的权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/
${iot:Certificate.Subject.CommonName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        }
    ]
}
```

> **Note**
>
> 在此示例中，证书的使用者公用名用作主题标识符，并假设使用者公用名对于每个已注册的证书都是唯一的。如果证书在多个设备之间共享，则共享此证书的所有设备的使用者公用名将相同，因而允许从多个设备向同一主题发布权限（不推荐）。

对于在注册 AWS IoT Core 表中注册的设备，以下策略授予使用 AWS IoT Core 与事物名称匹配的客户端 ID 进行连接的权限，以及在用于对设备进行身份验证的证书的Subject.CommonName.2字段设置为admin/时向名称前缀为的主题发布权限：Administrator

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],
            "Condition": {
                "StringEquals": {
                    "iot:Certificate.Subject.CommonName.2": "Administrator"
                }
            }
        }
    ]
}
```

对于未在注册 AWS IoT Core 表中注册的设备，当用于对设备进行身份验证的证书的Subject.CommonName.2字段设置为admin/时client1，client2以下策略授予使用客户端 ID 连接client3和发布到名称前缀为的主题的权限： AWS IoT Core Administrator

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],
            "Condition": {
                "StringEquals": {
                    "iot:Certificate.Subject.CommonName.2": "Administrator"
                }
            }
        }
    ]
}
```

对于在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用其事物名称发布特定主题，该主题包括用于对设备进行身份验证的证书ThingName何时将其任何一个Subject.CommonName字段设置为Administrator：admin/

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/
${iot:Connection.Thing.ThingName}"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
                }
            }
        }
```

```
    ]
}
```

对于未在注册 AWS IoT Core 表中注册的设备，当用于对设备进行身份验证的证书的任意一个Subject.CommonName字段设置为admin时client2，以下策略授予使用客户端 ID client1 连接client3和发布到主题的权限Administrator： AWS IoT Core

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1",
                "arn:aws:iot:us-east-1:123456789012:client/client2",
                "arn:aws:iot:us-east-1:123456789012:client/client3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin"],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "iot:Certificate.Subject.CommonName.List": "Administrator"
                }
            }
        }
    ]
}
```

事物策略示例

如果用于进行身份验证的证书附加到正在评估策略的 AWS IoT Core 对象，则以下策略允许设备进行连接：

```
{
```

```
    "Version":"2012-10-17",
    "Statement":[
        {
            "Effect":"Allow",
            "Action":["iot:Connect"],
            "Resource":[ "*" ],
            "Condition": {
                "Bool": {
                    "iot:Connection.Thing.IsAttached": ["true"]
                }
            }
        }
    ]
}
```

如果证书附加到具有特定事物类型的事物，并且该事物具有值为 attributeValue 的 attributeName 属性，则以下策略允许设备进行发布。有关事物策略变量的更多信息，请参阅事物策略变量。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/device/stats",
      "Condition": {
        "StringEquals": {
          "iot:Connection.Thing.Attributes[attributeName]": "attributeValue",
          "iot:Connection.Thing.ThingTypeName": "Thing_Type_Name"
        },
        "Bool": {
          "iot:Connection.Thing.IsAttached": "true"
        }
      }
    }
  ]
}
```

以下策略允许设备发布到以事物的属性开头的主题。如果设备证书与事物不关联，则将无法解析此变量，并将导致访问被拒绝错误。有关事物策略变量的更多信息，请参阅事物策略变量。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.Attributes[attributeName]}/*"
    }
  ]
}
```

基本任务策略示例

此示例说明了任务目标所需的策略状态，任务目标是接收任务请求并与 AWS IoT通信任务执行状态的单个设备。

将 *us-west-2:57 EXAMPLE833* 替换为你 AWS 区域的、冒号字符 (:) 和你的 12 位 AWS 账户 数字，然后用代表设备的事物资源的名称替换 uni *q ThingName* ue。 AWS IoT

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:client/uniqueThingName"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
```

```
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/pubtopic",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/events/job/*",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/events/jobExecution/*",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/things/uniqueThingName/jobs/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/test/dc/subtopic",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/$aws/events/jobExecution/*",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topicfilter/$aws/things/uniqueThingName/
jobs/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/test/dc/subtopic",
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/things/uniqueThingName/jobs/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeJobExecution",
        "iot:GetPendingJobExecutions",
        "iot:StartNextPendingJobExecution",
        "iot:UpdateJobExecution"
      ],
      "Resource": [
        "arn:aws:iot:us-west-2:57EXAMPLE833:topic/$aws/things/uniqueThingName"
      ]
    }
  ]
}
```

使用 Amazon Cognito 身份的授权

存在两种类型的 Amazon Cognito 身份：经过身份验证的身份和未经身份验证的身份。当您的应用程序支持未经身份验证的 Amazon Cognito 身份时，将不会执行身份验证，因此您不知道用户的身份。

未经身份验证的身份：对于未经身份验证的 Amazon Cognito 身份，您可以通过将 IAM 角色附加到未经身份验证的身份池来授予权限。我们建议您仅授予对希望可供未知用户使用的那些资源的访问权限。

> **Important**
>
> 对于连接 AWS IoT Core至的未经身份验证的 Amazon Cognito 用户，我们建议您在 IAM 策略中允许访问非常有限的资源。

经过身份验证的身份：对于经过身份验证的 Amazon Cognito 身份，您需要在两个位置指定权限：

- 将 IAM 策略附加到经过身份验证的 Amazon Cognito 身份池和
- 将 AWS IoT Core 策略附加到 Amazon Cognito 身份（经过身份验证的用户）。

将未经身份验证和经过身份验证的 Amazon Cognito 用户连接到 AWS IoT Core的策略示例

以下示例显示了 Amazon Cognito 身份的 IAM 策略和 IoT 策略中的权限。经过身份验证的用户希望发布到设备特定的主题（例如 device/DEVICE_ID/status）。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/Client_ID"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/device/Device_ID/status"
            ]
        }
    ]
}
```

以下示例显示了 Amazon Cognito 未经身份验证角色的 IAM 策略中的权限。未经身份验证的用户希望发布到不需要身份验证的非设备特定主题。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/non_device_specific_topic"
            ]
        }
    ]
}
```

GitHub 例子

以下示例 Web 应用程序 GitHub 展示了如何将对经过身份验证的用户的策略附件纳入用户注册和身份验证流程。

- MQTT 发布/订阅 React Web 应用程序使用和 AWS AmplifyAWS IoT Device SDK for JavaScript

- MQTT 使用 AWS Amplify、和 Lambda 函数发布/订阅 React Web 应用程序 AWS IoT Device SDK for JavaScript

Amplify 是一组工具和服务，可帮助您构建与 AWS 服务集成的网络和移动应用程序。有关 Amplify 的更多信息，请参阅 Amplify Framework Documentation。

这两个示例都执行了以下步骤。

1. 用户注册账户时，应用程序会创建 Amazon Cognito 用户池和身份。
2. 用户进行身份验证时，应用程序将创建策略并将其附加到身份。这授予用户发布和订阅的权限。
3. 用户可以使用应用程序发布和订阅 MQTT 主题。

第一个示例直接在身份验证操作中使用 `AttachPolicy` API 操作。以下示例演示如何在使用 Amplify 和 AWS IoT Device SDK for JavaScript的反应 Web 应用程序中实现此 API。

```
function attachPolicy(id, policyName) {
    var Iot = new AWS.Iot({region: AWSConfiguration.region, apiVersion:
 AWSConfiguration.apiVersion, endpoint: AWSConfiguration.endpoint});
    var params = {policyName: policyName, target: id};

    console.log("Attach IoT Policy: " + policyName + " with cognito identity id: " +
 id);
    Iot.attachPolicy(params, function(err, data) {
         if (err) {
               if (err.code !== 'ResourceAlreadyExistsException') {
                  console.log(err);
               }
          }
         else  {
            console.log("Successfully attached policy with the identity", data);
         }
     });
}
```

此代码出现在 AuthDisplay.js 文件中。

第二个示例在 Lambda 函数中实现 `AttachPolicy` API 操作。以下示例说明 Lambda 如何使用此 API 调用。

```
iot.attachPolicy(params, function(err, data) {
     if (err) {
           if (err.code !== 'ResourceAlreadyExistsException') {
              console.log(err);
              res.json({error: err, url: req.url, body: req.body});
           }
      }
     else  {
        console.log(data);
        res.json({success: 'Create and attach policy call succeed!', url: req.url,
 body: req.body});
     }
 });
```

此代码出现在app.js文件的 `iot.GetPolicy` 函数中。

> **Note**
>
> 当您使用通过 Amazon Cognito 身份池获得的 AWS 证书调用函数时，您的 Lambda 函数中的上下文对象包含的值为。`context.cognito_identity_id`有关更多信息，请参阅下列内容。
>
> - AWS Lambda Node.js 中的上下文对象
> - AWS Lambda Python 中的上下文对象
> - AWS Lambda Ruby 中的上下文对象
> - AWS Lambda Java 中的上下文对象
> - AWS Lambda Go 中的上下文对象
> - AWS Lambda C# 中的上下文对象
> - AWS Lambda 中的上下文对象 PowerShell

使用 AWS IoT Core 凭证提供者授权直接调用 AWS 服务

设备可以使用 X.509 证书通过 TLS 双向身份验证协议 AWS IoT Core 进行连接。其他 AWS 服务不支持基于证书的身份验证，但可以使用AWS 签名版本 4 格式的 AWS 凭据进行调用。签名版本 4 算法通常要求呼叫者拥有访问密钥 ID 和私有访问密钥。 AWS IoT Core 具有凭据提供程序，允许您使用内置

的 X.509 证书作为对请求进行身份验证的唯一设备身份。 AWS 这样就不再需要将访问密钥 ID 和私有访问密钥存储在您的设备上。

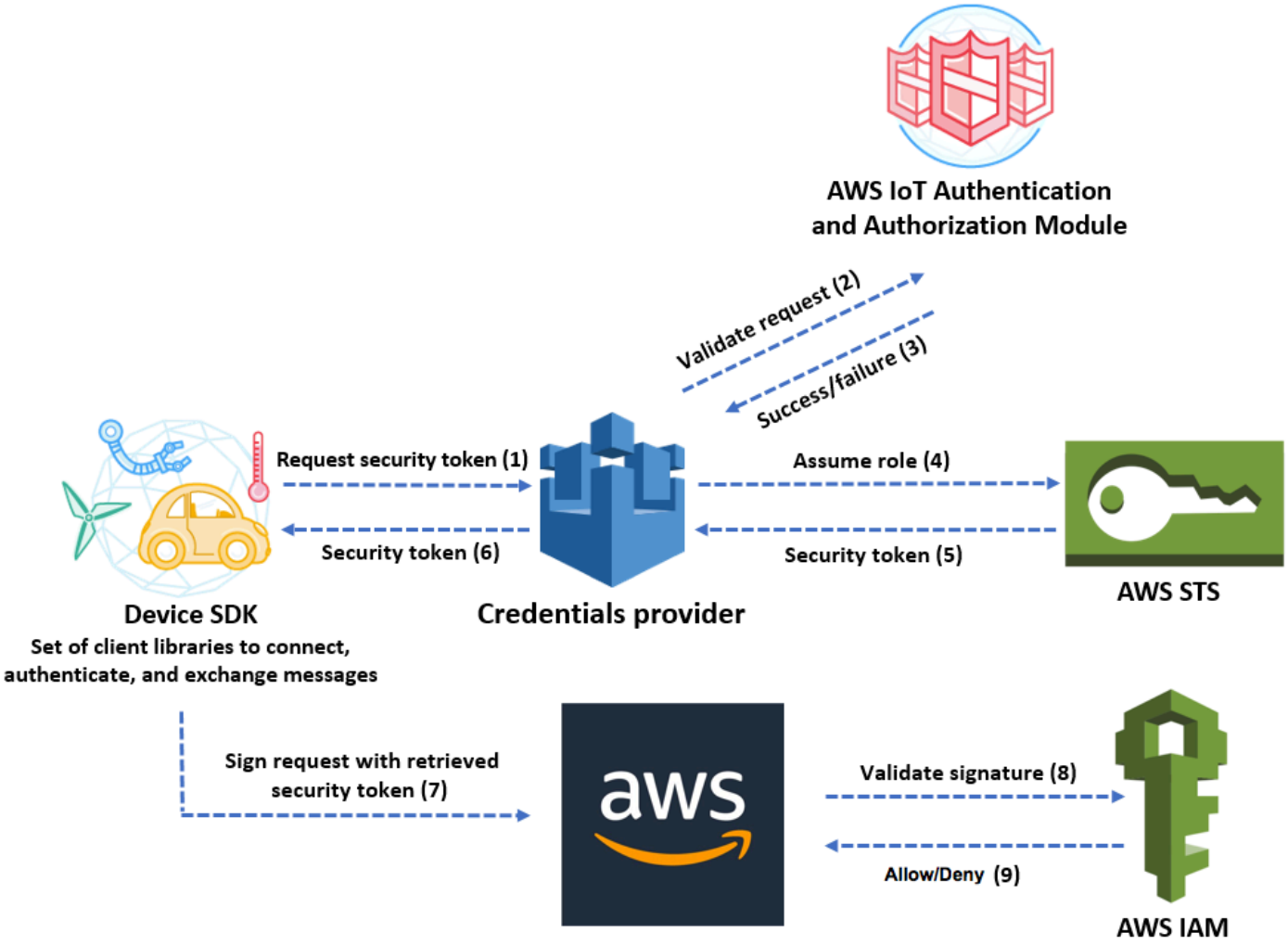
凭证提供程序使用 X.509 证书对调用方进行身份验证并颁发具有有限权限的临时安全令牌。该令牌可用于对任何 AWS 请求进行签名和身份验证。这种验证 AWS 请求的方式要求您创建和配置一个 AWS Identity and Access Management (IAM) 角色并将相应的 IAM 策略附加到该角色，以便证书提供者可以代表您担任该角色。有关 AWS IoT Core 和 IAM 的更多信息，请参阅 的身份和访问管理 AWS IoT。

AWS IoT 要求设备将服务器名称指示 (SNI) 扩展发送到传输层安全 (TLS) 协议，并在host_name字段中提供完整的端点地址。host_name 字段必须包含您调用的终端节点，并且必须是：

- `aws iot describe-endpoint --endpoint-type iot:CredentialProvider` 返回的 `endpointAddress`。

没有正确 host_name 值的设备尝试的连接将失败。

下图说明了凭证提供程序工作流程。



1. AWS IoT Core 设备向凭证提供者发出 HTTPS 请求以获取安全令牌。该请求包括用于身份验证的设备 X.509 证书。
2. 凭证提供者将请求转发到 AWS IoT Core 身份验证和授权模块，以验证证书并验证设备是否有权请求安全令牌。
3. 如果证书有效且有权请求安全令牌，则 AWS IoT Core 身份验证和授权模块将返回成功。否则，它会向设备发送异常。
4. 成功验证证书之后，凭证提供程序将调用 AWS Security Token Service (AWS STS) 来代入您为它创建的 IAM 角色。
5. AWS STS 向凭证提供者返回临时的有限权限安全令牌。
6. 凭证提供程序将该安全令牌返回给设备。
7. 设备使用安全令牌对签 AWS 名版本 4 的 AWS 请求进行签名。

8. 请求的服务调用 IAM 来验证签名并根据附加到您为凭证提供程序创建的 IAM 角色的访问策略授权请求。
9. 如果 IAM 成功验证签名并授权请求，则请求成功。否则，IAM 将发送异常。

下一节介绍如何使用证书来获取安全令牌。编写此内容时，假定您已注册了设备并为它创建并激活了您自己的证书。

如何使用证书来获取安全令牌

1. 配置凭证提供程序代表您的设备代入的 IAM 角色。将以下信任策略附加到该角色。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Principal": {"Service": "credentials.iot.amazonaws.com"},
        "Action": "sts:AssumeRole"
    }
}
```

对于您要调用的每项 AWS 服务，请为该角色附加访问策略。凭证提供程序支持以下策略变量：

- `credentials-iot:ThingName`
- `credentials-iot:ThingTypeName`
- `credentials-iot:AwsCertificateId`

当设备在向 AWS 服务发出的请求中提供了事物名称时，凭证提供程序会将 `credentials-iot:ThingName` 和 `credentials-iot:ThingTypeName` 作为上下文变量添加到安全令牌。如果设备没有在请求中提供事物名称，则凭证提供程序将提供 `credentials-iot:AwsCertificateId` 作为上下文变量。您将事物名称作为 `x-amzn-iot-thingname` HTTP 请求标头的值进行传递。

这三个变量仅适用于 IAM policy，而不适用于 AWS IoT Core 策略。

2. 确保执行下一步（创建角色别名）的用户有权将新创建的角色传递给 AWS IoT Core。以下策略向 AWS 用户同时授`iam:PassRole`予`iam:GetRole`和权限。`iam:GetRole` 权限可让用户获取有关您刚创建的角色的信息。该`iam:PassRole`权限允许用户将角色传递给其他 AWS 服务。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iam:GetRole",
            "iam:PassRole"
        ],
        "Resource": "arn:aws:iam::your AWS ## id:role/your role name"
    }
}
```

3. 创建 AWS IoT Core 角色别名。要直接呼叫 AWS 服务的设备必须知道连接时要使用哪个角色 ARN。 AWS IoT Core对角色 ARN 进行硬编码并不是一个很好的解决方案，因为它需要您在角色 ARN 发生更改时更新设备。更好的解决方案是使用 `CreateRoleAlias` API 创建一个指向角色 ARN 的角色别名。如果角色 ARN 发生更改，您只需更新角色别名。无需在设备上进行任何更改。此 API 接受以下参数：

   `roleAlias`

   必需。一个标识角色别名的任意字符串。它充当角色别名数据模型中的主键。它包含 1-128 个字符，并且必须仅包含字母数字字符以及 =、@ 和 - 符号。允许大写和小写字母字符。

   `roleArn`

   必需。角色别名所指向的角色的 ARN。

   `credentialDurationSeconds`

   可选。凭证有效的时间长度 (以秒为单位)。最小值为 900 秒（15 分钟）。最大值为 43200 秒（12 小时）。默认值为 3600 秒（1 小时）。

   > **Important**
   >
   > AWS IoT Core 凭证提供者可以颁发最长有效期为 43,200 秒（12 小时）的证书。凭证的有效期长达 12 小时，有助于通过将凭证缓存更长时间来减少对凭证提供程序的调用次数。
   >
   > `credentialDurationSeconds` 值必须小于或等于角色别名引用的 IAM 角色的最长会话持续时间。有关更多信息，请参阅《Identity and Access Managem AWS ent 用户指南》中的修改角色最长会话时长 (AWS API)。

   有关此 API 的更多信息，请参阅CreateRole别名。

4. 将策略附加到设备证书。附加到设备证书的策略必须向设备授予代入角色的权限。您可以通过授予对角色别名执行 `iot:AssumeRoleWithCertificate` 操作的权限来做到这一点，如以下示例所示。

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"iot:AssumeRoleWithCertificate",
         "Resource":"arn:aws:iot:your region:your_aws_account_id:rolealias/your
 role alias"
      }
   ]
}
```

5. 向凭证提供程序发出 HTTPS 请求来获取安全令牌。提供以下信息：

   - Certificate：由于这是使用 TLS 双向身份验证的 HTTP 请求，因此，您在发出请求时必须向客户端提供证书和私有密钥。使用与注册证书时相同的证书和私钥 AWS IoT Core。

     要确保您的设备正在与之通信 AWS IoT Core （而不是模拟它的服务），请参阅服务器身份验证，点击链接下载相应的 CA 证书，然后将其复制到您的设备上。
   - RoleAlias：您为凭证提供者创建的角色别名的名称。
   - ThingName: 您在注册事物时创建 AWS IoT Core 的事物名称。这作为 `x-amzn-iot-thingname` HTTP 标头的值进行传递。仅当您在 AWS IoT Core 或 IAM 策略中使用事物属性作为策略变量时，才需要此值。

     > **Note**
     >
     > 您在ThingName中提供的`x-amzn-iot-thingname`必须与分配给证书 AWS IoT 的事物资源的名称相匹配。如果不匹配，则返回 403 错误。

   在中运行以下命令 AWS CLI 以获取您的凭据提供程序终端节点 AWS 账户。有关此 API 的更多信息，请参阅DescribeEndpoint。

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

以下 JSON 对象是 describe-endpoint 命令的示例输出。它包含您用于请求安全令牌的 endpointAddress。

```
{
    "endpointAddress": "your_aws_account_specific_prefix.credentials.iot.your
 region.amazonaws.com"
}
```

使用端点向凭证提供程序发出 HTTPS 请求以返回安全令牌。以下示例命令使用 curl，但您可以使用任何 HTTP 客户端。

```
curl --cert your certificate --key your device certificate key pair -H "x-amzn-iot-
thingname: your thing name" --cacert AmazonRootCA1.pem https://your endpoint /role-
aliases/your role alias/credentials
```

此命令返回包含 accessKeyId、secretAccessKey、sessionToken 和过期的安全令牌对象。以下 JSON 对象是 curl 命令的示例输出。

```
   {"credentials":{"accessKeyId":"access key","secretAccessKey":"secret access
 key","sessionToken":"session token","expiration":"2018-01-18T09:18:06Z"}}
```

然后，您可以使用accessKeyIdsecretAccessKey、和sessionToken值对 AWS 服务请求进行签名。有关演 end-to-end 示，请参阅AWS 安全博客上的 “如何使用 AWS 凭据提供者博客文章来消除设备中对硬编码 AWS IoT 凭据的需求”。

通过 IAM 跨账户访问

AWS IoT Core 允许您允许委托人发布或订阅 AWS 账户 未归委托人所有的主题中定义的主题。您可以通过创建 IAM policy 和 IAM 角色并将策略附加到角色来配置跨账户访问。

首先，创建一个客户托管 IAM policy（如创建 IAM policy 中所述），就像您在 AWS 账户中为其他用户和证书创建策略一样。

对于在注册 AWS IoT Core 表中注册的设备，以下策略允许设备 AWS IoT Core 使用与设备的事物名称匹配的客户端 ID 进行连接，并允许设备发布到 *thing-name ########my/topic/thing-name*

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/my/topic/
${iot:Connection.Thing.ThingName}"],
        }
    ]
}
```

对于未在注册 AWS IoT Core 表中注册的设备，以下策略允许设备使用在您的账户 (123456789012) `client1` 注册 AWS IoT Core 表中注册的事物名称来连接 AWS IoT Core 并发布到名称前缀为的客户端 ID 专用主题：`my/topic/`

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/client1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
```

```
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/my/topic/${iot:ClientId}"
            ]
        }
    ]
}
```

接下来，按照创建向 IAM 用户委派权限的角色中的步骤操作。输入要与之共享访问权限的 AWS 账户的账户 ID。接下来是最后一步，请将您刚刚创建的策略附加到角色。如果您稍后需要修改要向其授予权限的 AWS ID，可使用以下信任策略格式执行操作：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam:us-east-1:567890123456:user/MyUser"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

中的数据保护 AWS IoT Core

分 AWS 担责任模型适用于中的数据保护 AWS IoT Core。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础架构上的内容的控制。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 Security Blog 上的 AWS Shared Responsibility Model and GDPR 博客文章。

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) 设置个人用户。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 与资源通信。 AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。

- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 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 AWS 通过命令行界面或 API 进行访问时需要经过 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）。这包括您使用控制台、API AWS IoT 或 SDK 或以其他 AWS 服务 方式使用控制台 AWS CLI、API 或 AWS SDK 的情况。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，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。

AWS IoT 设备收集数据，对这些数据进行一些操作，然后将该数据发送到其他 Web 服务。您可以选择在设备上将一些数据存储一段较短的时间。您有责任为静态数据提供任何数据保护。当您的设备向发送数据时 AWS IoT，它将通过 TLS 连接发送数据，如本节后面所述。 AWS IoT 设备可以向任何 AWS 服务发送数据。有关每项服务数据安全的更多信息，请参阅该服务的文档。 AWS IoT 可以配置为将日志写入 CloudWatch 日志并记录 AWS IoT API 调用 AWS CloudTrail。有关这些服务的数据安全的更多信息，请参阅 Amazon 的身份验证和访问控制 CloudWatch和使用 AWS KMS 托管密钥加密 CloudTrail 日志文件。

中的数据加密 AWS IoT

默认情况下，所有传输中的 AWS IoT 数据和静态数据都经过加密。传输中的数据使用 TLS 加密，静态数据使用 AWS 自有密钥进行加密。 AWS IoT 目前不支持密钥管理服务 AWS KMS keys () 中的客户管理（KMS 密钥AWS KMS）；但是，D AWS evice Advisor 和 AWS IoT Wireless 仅使用加密客户数据。 AWS 拥有的密钥

运输安全 AWS IoT Core

TLS（传输层安全性协议）是一种加密协议，旨在通过计算机网络进行安全通信。 AWS IoT Core 设备网关要求客户在设备与网关的连接中使用 TLS 对传输中的所有通信进行加密。TLS 用于实现所支持的应用程序协议（MQTT、HTTP 和 WebSocket）的 AWS IoT Core机密性。TLS 支持适用于许多编程语言和操作系统。 AWS 其中的数据由特定 AWS 服务加密。有关其他 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。

内容

TLS 协议

AWS IoT Core 支持以下版本的 TLS 协议：

- TLS 1.3
- TLS 1.2

使用 AWS IoT Core，您可以在域配置中配置 TLS 设置（适用于 TLS 1.2 和 TLS 1.3）。有关更多信息，请参阅 ???。

安全策略

安全策略是 TLS 协议及其密码的组合，此协议及其密码用于确定在客户端和服务器之间的 TLS 协商期间支持哪些协议和密码。您可以根据需要将设备配置为使用预定义的安全策略。请注意， AWS IoT Core 这不支持自定义安全策略。

在连接设备时，您可以为设备选择一种预定义的安全策略 AWS IoT Core。中最新的预定义安全策略的名称 AWS IoT Core 包括基于其发布年份和月份的版本信息。默认的预定义安全策略为 `IoTSecurityPolicy_TLS13_1_2_2022_10`。要指定安全策略，您可以使用 AWS IoT 控制台或 AWS CLI。有关更多信息，请参阅 ???。

下表描述了 AWS IoT Core 支持的最新预定义安全策略。为了使策略名称能够容纳在标题行中，已将 `IotSecurityPolicy_` 从名称中删除。

安全策略	TLS13_1_3_2022_10	TLS13_1_2_2022_10	TLS12_1_2_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*	
TCP 端口	443/8443/8883	443/8443/8883	443/8443/8883	443	8443/8883	443	8443/8883
TLS 协议							

安全策略	TLS13_1_3_2022_10	TLS13_1_2_2022_10	TLS12_1_2_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*	
TLS 1.2		✓	✓	✓	✓	✓	✓
TLS 1.3	✓	✓					
TLS 密码							
TLS_AES_128_GCM_SHA256	✓	✓					
TLS_AES_256_GCM_SHA384	✓	✓					
TLS_CHACHA20_POLY1305_SHA256	✓	✓					
ECDHE-RSA-AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES128-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES128-SHA		✓	✓	✓	✓	✓	✓

安全策略	TLS13_1_3_2022_10	TLS13_1_2_2022_10	TLS12_1_2_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*	
ECDHE-RSA-AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-SHA384		✓	✓	✓	✓	✓	✓
ECDHE-RSA-AES256-SHA		✓	✓	✓	✓	✓	✓
AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓
AES128-SHA256		✓	✓	✓		✓	✓
AES128-SHA		✓	✓	✓	✓	✓	✓
AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓
AES256-SHA256		✓	✓	✓	✓	✓	✓
AES256-SHA		✓	✓	✓	✓	✓	✓

安全策略	TLS13_1_3_2022_10	TLS13_1_2_2022_10	TLS12_1_2_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*	
DHE-RSA-AES256-SHA						✓	✓
ECDHE-ECDSA-AES128-GCM-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES128-SHA256		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES128-SHA		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-GCM-SHA384		✓	✓	✓	✓	✓	✓
ECDHE-ECDSA-AES256-SHA384		✓	✓	✓	✓	✓	✓

安全策略	TLS13_1_3_2022_10	TLS13_1_2_2022_10	TLS12_1_2_2022_10	TLS12_1_0_2016_01*		TLS12_1_0_2015_01*	
ECDHE-ECDSA-AES256-SHA		✓	✓	✓	✓	✓	✓

> **Note**
>
> TLS12_1_0_2016_01仅在以下版本中可用 AWS 区域：ap-east-1、ap-northeast-2、ap-southeast-1、ap-southeast-2、ca-central-1、cn-northeast-1、cn-northeast-1、eu-west-2、eu-west-2，eu-west-3、me-south-1、sa-east-1、us-east-2、-1、-2、us-west-1、us-west-1。 us-gov-west us-gov-west
> TLS12_1_0_2015_01仅在以下版本中可用 AWS 区域：ap-northeast-1、ap-southeast-1、eu-central-1、eu-west-1、us-east-1、us-east-1、us-west-1、us-west-2。

有关 AWS IoT Core中的传输安全性的重要注意事项

对于 AWS IoT Core 使用 MQTT 连接的设备，TLS 会加密设备与代理之间的连接，并 AWS IoT Core 使用 TLS 客户端身份验证来识别设备。有关更多信息，请参阅客户端身份验证。对于 AWS IoT Core 使用 HTTP 连接的设备，TLS 会加密设备与代理之间的连接，并将身份验证委托给 AWS 签名版本 4。有关更多信息，请参阅《AWS 一般参考》中的使用签名版本 4 签署请求。

将设备连接到时 AWS IoT Core，发送服务器名称指示 (SNI) 扩展名不是必需的，但强烈建议您这样做。要使用多账户注册、自定义域、VPC 终端节点和配置的 TLS 策略等功能，您必须使用 SNI 扩展并在字段中提供完整的终端节点地址。host_namehost_name 字段必须包含您调用的端点。该端点必须是以下端点之一：

- `aws iot describe-endpoint --endpoint-type iot:Data-ATS` 返回的 `endpointAddress`
- `aws iot describe-domain-configuration --domain-configuration-name "domain_configuration_name"` 返回的 `domainName`

使用错误或无效host_name值的设备尝试的连接将失败。 AWS IoT Core 将为 “自定义身份验证” CloudWatch 的身份验证类型记录失败。

AWS IoT Core 不支持 SessionTicket TLS 扩展。

LoRaWAN 无线设备的传输安全

LoRa广域网设备遵循金雅拓、Actility和Semtech在LoRa广域网™ 安全：为 LoRa 联盟准备的白皮书™ 中描述的安全实践。

有关 LoRa WAN 设备传输安全的更多信息，请参阅 LoRaWAN 数据和传输安全。

中的数据加密 AWS IoT

数据保护是指保护传输中（往返传输时 AWS IoT）和静态数据（存储在设备或其他 AWS 服务上时）的数据。发送到的所有数据都使用 MQTT、HTTPS 和 WebSocket 协议通过 TLS 连接发送，因此在传输过程中默认 AWS IoT 是安全的。 AWS IoT 设备收集数据，然后将其发送到其他 AWS 服务进行进一步处理。有关其他 AWS 服务上数据加密的更多信息，请参阅该服务的安全文档。

FreeRTOS 提供了一个 PKCS#11 库，用于提取密钥存储、访问加密对象和管理会话。您有责任使用此库对设备上的数据进行静态加密。有关更多信息，请参阅 FreeRTOS 公有密钥加密标准 (PKCS) #11 库。

Device Advisor

传输中加密

通过 Device Advisor 收发的所有数据都会在传输过程中加密。使用 Device Advisor API 时发送到服务和从服务发出的所有数据都使用 Signature Version 4 进行加密。有关如何签署 AWS API 请求的更多信息，请参阅签署 AWS API 请求。从测试设备发送到 Device Advisor 测试终端节点的所有数据都通过 TLS 连接发送，因此默认情况下其在传输过程中是安全的。

中的密钥管理 AWS IoT

与的所有连接均使用 TLS 完成，因此初始 TLS 连接不需要客户端加密密钥。 AWS IoT

设备必须使用 X.509 证书或 Amazon Cognito Identity 进行身份验证。您可以让 AWS IoT 为您生成一个证书，在此情况下，它将生成一个公有/私有密钥对。如果您使用的是 AWS IoT 控制台，系统将提示您下载证书和密钥。如果您使用的是 `create-keys-and-certificate` CLI 命令，则该 CLI 命令将返回证书和密钥。您有责任将证书和私有密钥复制到您的设备上并确保其安全。

AWS IoT 目前不支持 () 中的客户管理 AWS KMS keys （KMS 密钥 AWS Key Management Service AWS KMS ）；但是，Device Advisor 和 AWS IoT Wireless 仅使用加密客户数据。 AWS 拥有的密钥

Device Advisor

使用 AWS API 时发送到设备顾问的所有数据都是静态加密的。Device Advisor 使用在 AWS Key Management Service 中存储和管理的 KMS密钥对您的素有静态数据进行加密。设备顾问使用 AWS 拥有的密钥加密您的数据。有关的更多信息 AWS 拥有的密钥，请参阅AWS 拥有的密钥。

的身份和访问管理 AWS IoT

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（拥有权限）使用 AWS IoT 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

主题

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您所做的工作 AWS IoT。

服务用户-如果您使用 AWS IoT 服务完成工作，则管理员会为您提供所需的凭证和权限。当你使用更多 AWS IoT 功能来完成工作时，你可能需要额外的权限。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中的特征，请参阅 对 AWS IoT 身份和访问进行故障排除。

服务管理员-如果您负责公司的 AWS IoT 资源，则可能拥有完全访问权限 AWS IoT。您的工作是确定您的服务用户应访问哪些 AWS IoT 功能和资源。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。要详细了解您的公司如何将 IAM 与配合使用 AWS IoT，请参阅如何 AWS IoT 与 IAM 配合使用。

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的访问权限的详细信息。要查看您可以在 IAM 中使用的 AWS IoT 基于身份的策略示例，请参阅。AWS IoT 基于身份的策略示例

使用 IAM 身份进行身份验证

AWS IoT 身份中可以是设备 (X.509) 证书、Amazon Cognito 身份或 IAM 用户或群组。本主题仅讨论 IAM 身份。有关 AWS IoT 支持的其他身份的更多信息，请参阅客户端身份验证。

身份验证是您 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。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，您可能需要提供其他安全信息。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 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。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。有关需要您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 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，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代入。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

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。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。为控制您的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 用户可代入 IAM 用户或角色，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。角色是授予跨账户访问权限的主要方式。但是，对于某些资源 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代理）。要了解角色和基于资源的跨账户访问策略之间的区别，请参阅 IA M 用户指南中的跨账户资源访问。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。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 Simple Storage Service (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 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当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 请求时的策略详情，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-服务相关角色是一种与服务相关联的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

- 在 A@@ mazon 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证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该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

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。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。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 policy，用户可以代入角色。

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，假设您有一个允许 `iam:GetRole` 操作的策略。拥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 policy。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。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。委托人可以包括账户、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 IAM 中的 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 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF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 支持其他不太常见的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界：权限边界是一个高级特征，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` 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，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管

理的服务。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 限制成员账户中的实体（包括每个 AWS 账户根用户实体）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。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

如何 AWS IoT 与 IAM 配合使用

在使用 IAM 管理访问权限之前 AWS IoT，您应该了解哪些 IAM 功能可供使用 AWS IoT。要全面了解如何 AWS IoT 和其他 AWS 服务与 IAM 配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的AWS服务。

主题

- AWS IoT 基于身份的策略
- AWS IoT 基于资源的策略
- 基于 AWS IoT 标签的授权
- AWS IoT IAM 角色

AWS IoT 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，以及指定在什么条件下允许或拒绝操作。 AWS IoT 支持特定操作、资源和条件键。要了解在 JSON 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。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作需要在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行关联操作的权限。

下表列出了 IAM 物联网操作、关联 AWS IoT 的 API 以及该操作所操纵的资源。

策略操作	AWS IoT API	资源
物联网：AcceptCertificate转移	AcceptCertificate转移	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`* **Note** ARN 中 AWS 账户 指定的必须是证书要转移到的账户。
物联网：AddThingToThing群组	AddThingToThing群组	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* `arn:aws:iot:` *`region`*`:`*`account-id`* `:thing/`*`thing-name`*
物联网：AssociateTargetsWithJob	AssociateTargetsWithJob	none
物联网：AttachPolicy	AttachPolicy	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* 或者 `arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`*
物联网：AttachPrincipal政策	AttachPrincipal政策	`arn:aws:iot:` *`region`*`:`*`account-id`* `:cert/`*`cert-id`*

策略操作	AWS IoT API	资源
iot: AttachSecurity 个人资料	AttachSecurity个人资料	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot：AttachThing校长	AttachThing校长	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：CancelCertificate转移	CancelCertificate转移	arn:aws:iot: *region*:*account-id* :cert/*cert-id* **Note** ARN 中 AWS 账户 指定的必须是证书要转移到的账户。
物联网：CancelJob	CancelJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
物联网：CancelJob执行	CancelJob执行	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot: ClearDefault授权者	ClearDefault授权者	无
物联网：CreateAuthorizer	CreateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*
物联网：CreateCertificateFromCsr	CreateCertificateFromCsr	*
物联网：CreateDimension	CreateDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*

策略操作	AWS IoT API	资源
物联网：CreateJob	CreateJob	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
物联网:CreateJob模板	CreateJob模板	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
物联网：CreateKeysAndCertificate	CreateKeysAndCertificate	*
物联网：CreatePolicy	CreatePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网：CreatePolicy版本	CreatePolicy版本	arn:aws:iot: *region*:*account-id* :policy/*policy-name* **Note** 这必须是 AWS IoT 策略，而不是 IAM 策略。
iot: CreateRole别名	CreateRole别名	(参数：roleAlias) arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*

策略操作	AWS IoT API	资源
iot: CreateSecurity 个人资料	CreateSecurity个人资料	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
物联网：CreateThing	CreateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网：CreateThing群组	CreateThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 针对要创建的组和父组 (如果使用)
物联网：CreateThing类型	CreateThing类型	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
物联网:CreateTopic规则	CreateTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网：DeleteAuthorizer	DeleteAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-name*
iot:DeleteCACertificate	DeleteCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
物联网：DeleteCertificate	DeleteCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：DeleteDimension	DeleteDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
物联网：DeleteJob	DeleteJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
物联网:DeleteJob模板	DeleteJob模板	arn:aws:iot: *region*:*account-id* :job/*job-template-id*

策略操作	AWS IoT API	资源
物联网：DeleteJob执行	DeleteJob执行	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网：DeletePolicy	DeletePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网：DeletePolicy版本	DeletePolicy版本	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网:DeleteRegistration代码	DeleteRegistration代码	*
iot: DeleteRole别名	DeleteRole别名	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot: DeleteSecurity 个人资料	DeleteSecurity个人资料	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
物联网：DeleteThing	DeleteThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网：DeleteThing群组	DeleteThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网：DeleteThing类型	DeleteThing类型	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
物联网:DeleteTopic规则	DeleteTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网:deletev2LoggingLevel	deleteV2LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*

策略操作	AWS IoT API	资源
物联网：DeprecateThing类型	DeprecateThing类型	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
物联网：DescribeAuthorizer	DescribeAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name* (参数：authorizerName) none
iot:DescribeCACertificate	DescribeCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
物联网：DescribeCertificate	DescribeCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot: DescribeDefault 授权者	DescribeDefault授权者	无
物联网：DescribeEndpoint	DescribeEndpoint	*
iot: DescribeEvent 配置	DescribeEvent配置	none
物联网：DescribeIndex	DescribeIndex	arn:aws:iot: *region*:*account-id* :index/*index-name*
物联网：DescribeJob	DescribeJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
物联网：DescribeJob执行	DescribeJob执行	无
物联网:DescribeJob模板	DescribeJob模板	arn:aws:iot: *region*:*account-id* :job/*job-template-id*
iot: DescribeRole别名	DescribeRole别名	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*

策略操作	AWS IoT API	资源
物联网：DescribeThing	DescribeThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网：DescribeThing群组	DescribeThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网：DescribeThingRegistrationTask	DescribeThingRegistrationTask	无
物联网：DescribeThing类型	DescribeThing类型	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
物联网：DetachPolicy	DetachPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id* 或者 arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网：DetachPrincipal政策	DetachPrincipal政策	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot: DetachSecurity 个人资料	DetachSecurity个人资料	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot：DetachThing校长	DetachThing校长	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网:DisableTopic规则	DisableTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网:EnableTopic规则	EnableTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*

策略操作	AWS IoT API	资源
物联网:GetEffective政策	GetEffective政策	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：GetIndexing配置	GetIndexing配置	无
iot: GetJob 文档	GetJob文档	arn:aws:iot: *region*:*account-id* :job/*job-id*
物联网:GetLogging选项	GetLogging选项	*
物联网：GetPolicy	GetPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网：GetPolicy版本	GetPolicy版本	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网:GetRegistration代码	GetRegistration代码	*
物联网:GetTopic规则	GetTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网:ListAttached政策	ListAttached政策	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 或者 arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：ListAuthorizers	ListAuthorizers	无
iot:ListCACertificates	ListCACertificates	*

策略操作	AWS IoT API	资源
物联网：ListCertificates	ListCertificates	*
物联网：ListCertificatesbyCA	ListCertificatesbyCa	*
物联网：ListIndices	ListIndices	无
iot: ListJobExecutionsForJob	ListJobExecutionsForJob	无
物联网:ListJobExecutionsFor东西	ListJobExecutionsFor东西	无
物联网：ListJobs	ListJobs	`arn:aws:iot:` *`region`*`:`*`account-id`* `:thinggroup/` *`thing-group-name`* 如果使用 thingGroupName 参数
物联网:ListJob模板	ListJobs	无
物联网:ListOutgoing证书	ListOutgoing证书	*
物联网：ListPolicies	ListPolicies	*
iot: ListPolicy 校长	ListPolicy校长	*
物联网：ListPolicy版本	ListPolicy版本	`arn:aws:iot:` *`region`*`:`*`account-id`* `:policy/`*`policy-name`*

策略操作	AWS IoT API	资源
物联网:ListPrincipal政策	ListPrincipal政策	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：ListPrincipal东西	ListPrincipal事情	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot: ListRole 别名	ListRole别名	无
物联网：ListTargetsForPolicy	ListTargetsForPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
iot: ListThing 群组	ListThing群组	无
物联网:ListThingGroupsFor东西	ListThingGroupsFor东西	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot: ListThing 校长	ListThing校长	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网：ListThingRegistrationTask报告	ListThingRegistrationTask报告	无
物联网：ListThingRegistrationTasks	ListThingRegistrationTasks	无
物联网:ListThing类型	ListThing类型	*
物联网：ListThings	ListThings	*

策略操作	AWS IoT API	资源
物联网：ListThingsInThing群组	ListThingsInThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网:ListTopicRule规则	ListTopicRule规则	*
IoT: listv2LoggingLevels	Listv2 LoggingLevels	无
iot:RegisterCACertificate	RegisterCACertificate	*
物联网：RegisterCertificate	RegisterCertificate	*
物联网：RegisterThing	RegisterThing	无
物联网：RejectCertificate转移	RejectCertificate转移	arn:aws:iot: *region*:*account-id* :cert/*certificate-id*
物联网：RemoveThingFromThing群组	RemoveThingFromThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网:ReplaceTopicRule规则	ReplaceTopicRule规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网：SearchIndex	SearchIndex	arn:aws:iot: *region*:*account-id* :index/*index-id*
iot: SetDefault 授权者	SetDefaultAuthorizer授权者	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*

策略操作	AWS IoT API	资源
物联网：SetDefaultPolicyVersion	SetDefaultPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网:SetLogging选项	SetLogging选项	arn:aws:iot: *region*:*account-id* :role/*role-name*
IoT: setv2LoggingLevel	setv2 LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
IoT: setv2LoggingOptions	setv2 LoggingOptions	arn:aws:iot: *region*:*account-id* :role/*role-name*
物联网：StartThingRegistrationTask	StartThingRegistrationTask	无
物联网：StopThingRegistrationTask	StopThingRegistrationTask	无
物联网：TestAuthorization	TestAuthorization	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot: TestInvoke授权者	TestInvoke授权者	无
物联网：TransferCertificate	TransferCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：UpdateAuthorizer	UpdateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizerfunction/ *authorizer-function-name*
iot:UpdateCACertificate	UpdateCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*

策略操作	AWS IoT API	资源
物联网：UpdateCertificate	UpdateCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：UpdateDimension	UpdateDimension	arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
iot: UpdateEventConfiguration	UpdateEventConfiguration	无
物联网：UpdateIndexingConfiguration	UpdateIndexingConfiguration	无
iot: UpdateRoleAlias	UpdateRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
iot: UpdateSecurityProfile	UpdateSecurityProfile	arn:aws:iot: *region*:*account-id* :securityprofile/ *security-profile-name* arn:aws:iot: *region*:*account-id* :dimension/ *dimension-name*
物联网：UpdateThing	UpdateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网：UpdateThingGroup	UpdateThingGroup	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网:UpdateThingGroupsForThing	UpdateThingGroupsForThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name* arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*

正在执行的策略操作在操作前 AWS IoT 使用以下前缀:`iot:`. 例如，要授予某人列出他们 AWS 账户 在 `ListThings` API 中注册的所有物联网事物的权限，您需要将该`iot:ListThings`操作包含在他们的策略中。策略声明必须包含`Action`或`NotAction`元素。 AWS IoT 定义了它自己的一组操作，这些操作描述了您可以使用此服务执行的任务。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": [
      "ec2:action1",
      "ec2:action2"
```

您也可以使用通配符（*）指定多个操作。例如，要指定以单词 Describe 开头的所有操作，包括以下操作：

```
"Action": "iot:Describe*"
```

要查看 AWS IoT 操作列表，请参阅 IAM 用户指南 AWS IoT中的定义操作。

Device Advisor 操作

下表列出了 IAM IoT Device Advisor 操作、关联的 AWS IoT Device Advisor API 以及操作处理的资源。

策略操作	AWS IoT API	资源
iotdevice Advisor：定义 CreateSuite	CreateSuite定义	无
iotdevice Advisor：定义 DeleteSuite	DeleteSuite定义	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdevice Advisor：定义 GetSuite	GetSuite定义	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdevice Advisor：运行 GetSuite	GetSuite跑	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-run-id*
iotdevice Advisor：G	GetSuiteR unReport	arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/ *suite-definition-id* /*suite-run-id*

策略操作	AWS IoT API	资源
etSuiteRunReport		
iotdeviceAdvisor：定义ListSuite	ListSuite定义	无
iotdeviceAdvisor：运行ListSuite	ListSuite跑步	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceAdvisor：ListTagsForResource	ListTagsForResource	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id* arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*
iotdeviceAdvisor：运行StartSuite	StartSuite跑	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id*
iotdeviceAdvisor：TagResource	TagResource	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id* arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*
iotdeviceAdvisor：UntagResource	UntagResource	arn:aws:iotdeviceadvisor: *region*:*account-id* :suitedefinition/ *suite-definition-id* arn:aws:iotdeviceadvisor: *region*:*account-id* :suiterun/suite-definition-id/ *suite-run-id*

策略操作	AWS IoT API	资源
iotdevice Advisor：定义 UpdateSuite	UpdateSuite定义	`arn:aws:iotdeviceadvisor: region:account-id :suitedefinition/ suite-definition-id`
iotdevice Advisor：运行 StopSuite	StopSuite跑	`arn:aws:iotdeviceadvisor: region:account-id :suiterun/suite-definition-id/ suite-run-id`

AWS IoT 设备顾问中的策略操作在操作前使用以下前缀:`iotdeviceadvisor:`. 例如，要授予某人列出他们在 ListSuiteDefinitions API 中注册的所有套件定义 AWS 账户 的权限，您需要将该`iotdeviceadvisor:ListSuiteDefinitions`操作包含在他们的策略中。

资源

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。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": "*"
```

AWS IoT 资源

策略操作	AWS IoT API	资源
物联网：AcceptCertificate转移	AcceptCertificate转移	`arn:aws:iot: region:account-id :cert/cert-id` **Note** ARN 中 AWS 账户 指定的必须是证书要转移到的账户。

策略操作	AWS IoT API	资源
物联网：AddThingToThing群组	AddThingToThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网：AssociateTargetsWithJob	AssociateTargetsWithJob	无
物联网：AttachPolicy	AttachPolicy	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 或者 arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：AttachPrincipal政策	AttachPrincipal政策	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot：AttachThing校长	AttachThing校长	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：CancelCertificate转移	CancelCertificate转移	arn:aws:iot: *region*:*account-id* :cert/*cert-id* **Note** ARN 中 AWS 账户 指定的必须是证书要转移到的账户。
物联网：CancelJob	CancelJob	arn:aws:iot: *region*:*account-id* :job/*job-id*

策略操作	AWS IoT API	资源
物联网：CancelJob执行	CancelJob执行	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot: ClearDefaultAuthorizer	ClearDefaultAuthorizer	无
物联网：CreateAuthorizer	CreateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*
物联网：CreateCertificateFromCsr	CreateCertificateFromCsr	*
物联网：CreateJob	CreateJob	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
物联网:CreateJob模板	CreateJob模板	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
物联网：CreateKeysAndCertificate	CreateKeysAndCertificate	*
物联网：CreatePolicy	CreatePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*

策略操作	AWS IoT API	资源
CreatePolicy版本	物联网：CreatePolicy版本	arn:aws:iot: *region*:*account-id* :policy/*policy-name* **Note** 这必须是 AWS IoT 策略，而不是 IAM 策略。
iot: CreateRole别名	CreateRole别名	(参数：roleAlias) arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
物联网：CreateThing	CreateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网：CreateThing群组	CreateThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 针对要创建的组和父组 (如果使用)
物联网：CreateThing类型	CreateThing类型	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
物联网:CreateTopic规则	CreateTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网：DeleteAuthorizer	DeleteAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-name*
iot:DeleteCACertificate	DeleteCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
物联网：DeleteCertificate	DeleteCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：DeleteJob	DeleteJob	arn:aws:iot: *region*:*account-id* :job/*job-id*

策略操作	AWS IoT API	资源
物联网：DeleteJob执行	DeleteJob执行	arn:aws:iot: *region*:*account-id* :job/*job-id* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网:DeleteJob模板	DeleteJob模板	arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
物联网：DeletePolicy	DeletePolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网：DeletePolicy版本	DeletePolicy版本	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网:DeleteRegistration代码	DeleteRegistration代码	*
iot: DeleteRoleAlias	DeleteRoleAlias	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
物联网：DeleteThing	DeleteThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网：DeleteThing群组	DeleteThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网：DeleteThing类型	DeleteThing类型	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
物联网:DeleteTopic规则	DeleteTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网:deletev2LoggingLevel	deleteV2LoggingLevel	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网：DeprecateThing类型	DeprecateThing类型	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*

策略操作	AWS IoT API	资源
物联网：DescribeAuthorizer	DescribeAuthorizer	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name* (参数：authorizerName) none
iot:DescribeCACertificate	DescribeCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
物联网：DescribeCertificate	DescribeCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot: DescribeDefault 授权者	DescribeDefault 授权者	无
物联网：DescribeEndpoint	DescribeEndpoint	*
iot: DescribeEvent 配置	DescribeEvent配置	none
物联网：DescribeIndex	DescribeIndex	arn:aws:iot: *region*:*account-id* :index/*index-name*
物联网：DescribeJob	DescribeJob	arn:aws:iot: *region*:*account-id* :job/*job-id*
物联网：DescribeJob执行	DescribeJob执行	无
物联网:DescribeJob模板	DescribeJob模板	arn:aws:iot: *region*:*account-id* :jobtemplate/ *job-template-id*
iot: DescribeRole别名	DescribeRole别名	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
物联网：DescribeThing	DescribeThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*

策略操作	AWS IoT API	资源
物联网：DescribeThing群组	DescribeThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网：DescribeThingRegistrationTask	DescribeThingRegistrationTask	无
物联网：DescribeThing类型	DescribeThing类型	arn:aws:iot: *region*:*account-id* :thingtype/ *thing-type-name*
物联网：DetachPolicy	DetachPolicy	arn:aws:iot: *region*:*account-id* :cert/*cert-id* 或者 arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网：DetachPrincipal政策	DetachPrincipal政策	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot：DetachThing校长	DetachThing校长	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网:DisableTopic规则	DisableTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网:EnableTopic规则	EnableTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网:GetEffective政策	GetEffective政策	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：GetIndexing配置	GetIndexing配置	无
iot: GetJob 文档	GetJob文档	arn:aws:iot: *region*:*account-id* :job/*job-id*

策略操作	AWS IoT API	资源
物联网:GetLogging选项	GetLogging选项	*
物联网：GetPolicy	GetPolicy	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网：GetPolicy版本	GetPolicy版本	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网:GetRegistration代码	GetRegistration代码	*
物联网:GetTopic规则	GetTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网:ListAttached政策	ListAttached政策	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* 或者 arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：ListAuthorizers	ListAuthorizers	无
iot:ListCACertificates	ListCACertificates	*
物联网：ListCertificates	ListCertificates	*
物联网：ListCertificatesbyCA	ListCertificatesbyCa	*
物联网：ListIndices	ListIndices	无

策略操作	AWS IoT API	资源
iot: ListJob ExecutionsFor Job	ListJobEx ecutionsForJob	无
物联网:ListJ obExecutionsFor 东西	ListJobEx ecutionsFor东西	无
物联网：ListJobs	ListJobs	arn:aws:iot: *region*:*account-id* :thinggro up/ *thing-group-name* 如果使用 thingGroupName 参数
物联网:ListJob模板	ListJob模板	无
物联网:ListO utgoing证书	ListOutgoing证书	*
物联网：ListP olicies	ListPolicies	*
iot: ListPolicy 校长	ListPolicy校长	arn:aws:iot: *region*:*account-i d* :policy/*policy-name*
物联网：ListP olicy版本	ListPolicy版本	arn:aws:iot: *region*:*account-i d* :policy/*policy-name*
物联网:ListP rincipal政策	ListPrincipal政策	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：ListP rincipal东西	ListPrincipal事情	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot: ListRole 别名	ListRole别名	无
物联网：ListT argetsForPolicy	ListTarge tsForPolicy	arn:aws:iot: *region*:*account-i d* :policy/*policy-name*

策略操作	AWS IoT API	资源
iot: ListThing 群组	ListThing群组	无
物联网:ListThingGroupsFor东西	ListThingGroupsFor东西	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
iot: ListThing 校长	ListThing校长	arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网：ListThingRegistrationTask报告	ListThingRegistrationTask报告	无
物联网：ListThingRegistrationTasks	ListThingRegistrationTasks	无
物联网:ListThing类型	ListThing类型	*
物联网：ListThings	ListThings	*
物联网：ListThingsInThing群组	ListThingsInThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网:ListTopic规则	ListTopic规则	*
IoT: listv2LoggingLevels	Listv2 LoggingLevels	无
iot:RegisterCACertificate	RegisterCACertificate	*

策略操作	AWS IoT API	资源
物联网：RegisterCertificate	RegisterCertificate	*
物联网：RegisterThing	RegisterThing	无
物联网：RejectCertificate转移	RejectCertificate转移	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：RemoveThingFromThing群组	RemoveThingFromThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name* arn:aws:iot: *region*:*account-id* :thing/*thing-name*
物联网:ReplaceTopic规则	ReplaceTopic规则	arn:aws:iot: *region*:*account-id* :rule/*rule-name*
物联网：SearchIndex	SearchIndex	arn:aws:iot: *region*:*account-id* :index/*index-id*
iot: SetDefault 授权者	SetDefault授权者	arn:aws:iot: *region*:*account-id* :authorizer/ *authorizer-function-name*
物联网：SetDefaultPolicyVersion	SetDefaultPolicyVersion	arn:aws:iot: *region*:*account-id* :policy/*policy-name*
物联网:SetLogging选项	SetLogging选项	*
IoT: setv2 LoggingLevel	setv2 LoggingLevel	*
IoT: setv2 LoggingOptions	setv2 LoggingOptions	*

策略操作	AWS IoT API	资源
物联网：StartThingRegistrationTask	StartThingRegistrationTask	无
物联网：StopThingRegistrationTask	StopThingRegistrationTask	无
物联网：TestAuthorization	TestAuthorization	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot: TestInvoke授权者	TestInvoke授权者	无
物联网：TransferCertificate	TransferCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
物联网：UpdateAuthorizer	UpdateAuthorizer	arn:aws:iot: *region*:*account-id* :authorizerfunction/ *authorizer-function-name*
iot:UpdateCACertificate	UpdateCACertificate	arn:aws:iot: *region*:*account-id* :cacert/*cert-id*
物联网：UpdateCertificate	UpdateCertificate	arn:aws:iot: *region*:*account-id* :cert/*cert-id*
iot: UpdateEvent配置	UpdateEvent配置	无
物联网：UpdateIndexing配置	UpdateIndexing配置	无
iot: UpdateRole别名	UpdateRole别名	arn:aws:iot: *region*:*account-id* :rolealias/ *role-alias-name*
物联网：UpdateThing	UpdateThing	arn:aws:iot: *region*:*account-id* :thing/*thing-name*

策略操作	AWS IoT API	资源
物联网：UpdateThing群组	UpdateThing群组	arn:aws:iot: *region*:*account-id* :thinggroup/ *thing-group-name*
物联网:UpdateThingGroupsFor东西	UpdateThingGroupsFor东西	arn:aws:iot: *region*:*account-id* :thing/*thing-name*

有关 ARN 格式的更多信息，请参阅 Amazon 资源名称 (ARN) 和 AWS 服务命名空间。

某些 AWS IoT 操作（例如创建资源的操作）无法对特定资源执行。在这些情况下，您必须使用通配符（*）。

```
"Resource": "*"
```

要查看 AWS IoT 资源类型及其 ARN 的列表，请参阅 IAM 用户指南 AWS IoT中的由定义的资源。要了解您可以在哪些操作中指定每个资源的 ARN，请参阅 AWS IoT定义的操作。

Device Advisor 资源

要为 Device Advisor IAM 策略定义资源级限制，请使用以下资源 ARN 格式来定义套件和套件运行。AWS IoT

套件定义资源 ARN 格式

```
arn:aws:iotdeviceadvisor:region:account-id:suitedefinition/suite-definition-id
```

套件运行资源 ARN 格式

```
arn:aws:iotdeviceadvisor:region:account-id:suiterun/suite-definition-id/suite-run-id
```

条件键

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则使用逻辑OR运算来 AWS 评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅《IAM 用户指南》中的 IAM policy 元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件密钥。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

AWS IoT 定义自己的条件键集，还支持使用一些全局条件键。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

AWS IoT 条件键

AWS IoT 条件键	描述	类型
`aws:RequestTag/${tag-key}`	用户向 AWS IoT 发出的请求中包含的标签键。	String
`aws:ResourceTag/${tag-key}`	附加到 AWS IoT 资源的标签的标签密钥组件。	String
`aws:TagKeys`	与请求中的资源关联的所有标签键名称的列表。	String

要查看 AWS IoT 条件键列表，请参阅 IAM 用户指南 AWS IoT中的条件密钥。要了解您可以使用条件键的操作和资源，请参阅操作定义者 AWS IoT。

示例

要查看 AWS IoT 基于身份的策略的示例，请参阅。AWS IoT 基于身份的策略示例

AWS IoT 基于资源的策略

基于资源的策略是 JSON 策略文档，用于指定委托人可以在哪些条件下对 AWS IoT 资源执行哪些操作。

AWS IoT 不支持基于 IAM 资源的策略。但是，它确实支持 AWS IoT 基于资源的政策。有关更多信息，请参阅 AWS IoT Core 政策。

基于 AWS IoT 标签的授权

您可以为 AWS IoT 资源附加标签或在请求中传递标签 AWS IoT。要基于标签控制访问，您需要使用 `iot:ResourceTag/key-name`、`aws:RequestTag/key-name` 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 在 IAM 策略中使用标签。有关为 AWS IoT 资源添加标签的更多信息，请参阅为资源添加 AWS IoT 标签。

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅根据标签查看 AWS IoT 资源。

AWS IoT IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的实体。

将临时证书与 AWS IoT

可以使用临时凭证进行联合身份验证登录，分派 IAM 角色或分派跨账户角色。您可以通过调用 AWS STS API 操作（例如AssumeRole或GetFederation令牌）来获取临时安全证书。

AWS IoT 支持使用临时证书。

服务相关角色

服务相关角色允许 AWS 服务访问其他服务中的资源以代表您完成操作。服务相关角色显示在 IAM 账户中，并归该服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

AWS IoT 不支持服务相关角色。

服务角色

此功能允许服务代表您担任服务角色。此角色允许服务访问其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 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 资源的权限。他们也无法使用 AWS Management Console AWS CLI、或 AWS API 执行任务。IAM 管理员必须创建 IAM 策略，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用户或组。

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。

主题

策略最佳实践

基于身份的策略决定了某人是否可以在您的账户中创建、访问或删除 AWS IoT 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，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略 或 工作职能的AWS 托管策略。
- 应用最低权限 – 在使用 IAM policy 设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM policy 中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件 。
- 使用 IAM Access Analyzer 验证您的 IAM policy，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM policy语言 (JSON) 和 IAM 最佳实

践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。

- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。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 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

使用 AWS IoT 控制台

要访问 AWS IoT 控制台，您必须拥有一组最低权限。这些权限必须允许您列出和查看有关您的 AWS IoT 资源的详细信息 AWS 账户。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的实体（用户或角色），控制台将无法按预期正常运行。

为确保这些实体仍然可以使用 AWS IoT 控制台，还要将以下 AWS 托管策略附加到这些实体：`AWSIoTFullAccess`。有关更多信息，请参阅《IAM 用户指南》中的为用户添加权限。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。相反，只允许访问与您尝试执行的 API 操作相匹配的操作。

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策略。此策略包括在控制台上或使用 AWS CLI 或 AWS API 以编程方式完成此操作的权限。

```
{
    "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": "*"
        }
    ]
}
```

根据标签查看 AWS IoT 资源

您可以在基于身份的策略中使用条件，以便基于标签控制对 AWS IoT 资源的访问。该示例说明了如何创建策略以允许查看事物。不过，只有在事物标签 Owner 具有该用户的用户名值时，才会授予权限。此策略还授予在控制台上完成此操作的必要权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListBillingGroupsInConsole",
            "Effect": "Allow",
            "Action": "iot:ListBillingGroups",
            "Resource": "*"
        },
        {
            "Sid": "ViewBillingGroupsIfOwner",
            "Effect": "Allow",
           "Action": "iot:DescribeBillingGroup",
            "Resource": "arn:aws:iot:*:*:billinggroup/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
            }
        }
    ]
```

```
}
```

您可以将该策略附加到您账户中的 IAM 用户。如果名为的用户`richard-roe`尝试查看 AWS IoT 账单组，则该账单组必须被标记`Owner=richard-roe`或`owner=richard-roe`。否则，他将被拒绝访问。条件标签键 `Owner` 匹配 `Owner` 和 `owner`，因为条件键名称不区分大小写。有关更多信息，请参阅 IAM 用户指南 中的 IAM JSON 策略元素：条件。

根据标签查看 AWS IoT 设备顾问资源

您可以在基于身份的策略中使用条件，以便基于标签控制对 AWS IoT Device Advisor 资源的访问。以下示例说明如何创建策略以允许查看特定套件定义。但是，仅当套件定义标签具有值设置为 `MQTT` 的 `SuiteType` 时，才会授予权限。此策略还授予在控制台上完成此操作的必要权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewSuiteDefinition",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:GetSuiteDefinition",
            "Resource": "arn:aws:iotdeviceadvisor:*:*:suitedefinition/*",
            "Condition": {
                "StringEquals": {"aws:ResourceTag/SuiteType": "MQTT"}
            }
        }
    ]
}
```

AWS 的托管策略 AWS IoT

要向用户、群组和角色添加权限，使用 AWS 托管策略比自己编写策略要容易得多。创建仅为团队提供所需权限的 IAM 客户管理型策略需要时间和专业知识。要快速入门，您可以使用我们的 AWS 托管策略。这些策略涵盖常见使用案例，可在您的 AWS 账户中使用。有关 AWS 托管策略的更多信息，请参阅 IAM 用户指南中的AWS 托管策略。

AWS 服务维护和更新 AWS 托管策略。您无法更改 AWS 托管策略中的权限。服务偶尔会向 AWS 管理型策略添加额外权限以支持新特征。此类更新会影响附加策略的所有身份（用户、组和角色）。当启动新特征或新操作可用时，服务最有可能会更新 AWS 管理型策略。服务不会从 AWS 托管策略中移除权限，因此策略更新不会破坏您的现有权限。

此外，还 AWS 支持跨多个服务的工作职能的托管策略。例如，ReadOnly访问 AWS 管理策略提供对所有 AWS 服务和资源的只读访问权限。当服务启动新特征时， AWS 会为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅 IAM 用户指南中的适用于工作职能的AWS 管理型策略。

> **Note**
>
> AWS IoT 适用于两者 AWS IoT 和 IAM 策略。本主题仅讨论 IAM policy，它为控制面板和数据层面 API 操作定义策略操作。另请参阅 AWS IoT Core 政策。

AWS 托管策略： AWSIoTConfigAccess

您可以将 `AWSIoTConfigAccess` 策略附加到 IAM 身份。

此策略向相关身份授予权限，以允许访问所有 AWS IoT 配置操作。此策略可能会影响数据处理和存储。要在中查看此政策 AWS Management Console，请参阅AWSIoTConfigAccess。

权限详细信息

该策略包含以下权限。

- `iot`— 检索 AWS IoT 数据并执行物联网配置操作。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:AcceptCertificateTransfer",
                "iot:AddThingToThingGroup",
                "iot:AssociateTargetsWithJob",
                "iot:AttachPolicy",
                "iot:AttachPrincipalPolicy",
                "iot:AttachThingPrincipal",
```

```
                "iot:CancelCertificateTransfer",
                "iot:CancelJob",
                "iot:CancelJobExecution",
                "iot:ClearDefaultAuthorizer",
                "iot:CreateAuthorizer",
                "iot:CreateCertificateFromCsr",
                "iot:CreateJob",
                "iot:CreateKeysAndCertificate",
                "iot:CreateOTAUpdate",
                "iot:CreatePolicy",
                "iot:CreatePolicyVersion",
                "iot:CreateRoleAlias",
                "iot:CreateStream",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:CreateThingType",
                "iot:CreateTopicRule",
                "iot:DeleteAuthorizer",
                "iot:DeleteCACertificate",
                "iot:DeleteCertificate",
                "iot:DeleteJob",
                "iot:DeleteJobExecution",
                "iot:DeleteOTAUpdate",
                "iot:DeletePolicy",
                "iot:DeletePolicyVersion",
                "iot:DeleteRegistrationCode",
                "iot:DeleteRoleAlias",
                "iot:DeleteStream",
                "iot:DeleteThing",
                "iot:DeleteThingGroup",
                "iot:DeleteThingType",
                "iot:DeleteTopicRule",
                "iot:DeleteV2LoggingLevel",
                "iot:DeprecateThingType",
                "iot:DescribeAuthorizer",
                "iot:DescribeCACertificate",
                "iot:DescribeCertificate",
                "iot:DescribeDefaultAuthorizer",
                "iot:DescribeEndpoint",
                "iot:DescribeEventConfigurations",
                "iot:DescribeIndex",
                "iot:DescribeJob",
                "iot:DescribeJobExecution",
                "iot:DescribeRoleAlias",
```

```
                "iot:DescribeStream",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingRegistrationTask",
                "iot:DescribeThingType",
                "iot:DetachPolicy",
                "iot:DetachPrincipalPolicy",
                "iot:DetachThingPrincipal",
                "iot:DisableTopicRule",
                "iot:EnableTopicRule",
                "iot:GetEffectivePolicies",
                "iot:GetIndexingConfiguration",
                "iot:GetJobDocument",
                "iot:GetLoggingOptions",
                "iot:GetOTAUpdate",
                "iot:GetPolicy",
                "iot:GetPolicyVersion",
                "iot:GetRegistrationCode",
                "iot:GetTopicRule",
                "iot:GetV2LoggingOptions",
                "iot:ListAttachedPolicies",
                "iot:ListAuthorizers",
                "iot:ListCACertificates",
                "iot:ListCertificates",
                "iot:ListCertificatesByCA",
                "iot:ListIndices",
                "iot:ListJobExecutionsForJob",
                "iot:ListJobExecutionsForThing",
                "iot:ListJobs",
                "iot:ListOTAUpdates",
                "iot:ListOutgoingCertificates",
                "iot:ListPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPolicyVersions",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListRoleAliases",
                "iot:ListStreams",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroups",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:ListThingRegistrationTaskReports",
                "iot:ListThingRegistrationTasks",
```

```
                "iot:ListThings",
                "iot:ListThingsInThingGroup",
                "iot:ListThingTypes",
                "iot:ListTopicRules",
                "iot:ListV2LoggingLevels",
                "iot:RegisterCACertificate",
                "iot:RegisterCertificate",
                "iot:RegisterThing",
                "iot:RejectCertificateTransfer",
                "iot:RemoveThingFromThingGroup",
                "iot:ReplaceTopicRule",
                "iot:SearchIndex",
                "iot:SetDefaultAuthorizer",
                "iot:SetDefaultPolicyVersion",
                "iot:SetLoggingOptions",
                "iot:SetV2LoggingLevel",
                "iot:SetV2LoggingOptions",
                "iot:StartThingRegistrationTask",
                "iot:StopThingRegistrationTask",
                "iot:TestAuthorization",
                "iot:TestInvokeAuthorizer",
                "iot:TransferCertificate",
                "iot:UpdateAuthorizer",
                "iot:UpdateCACertificate",
                "iot:UpdateCertificate",
                "iot:UpdateEventConfigurations",
                "iot:UpdateIndexingConfiguration",
                "iot:UpdateRoleAlias",
                "iot:UpdateStream",
                "iot:UpdateThing",
                "iot:UpdateThingGroup",
                "iot:UpdateThingGroupsForThing",
                "iot:UpdateAccountAuditConfiguration",
                "iot:DescribeAccountAuditConfiguration",
                "iot:DeleteAccountAuditConfiguration",
                "iot:StartOnDemandAuditTask",
                "iot:CancelAuditTask",
                "iot:DescribeAuditTask",
                "iot:ListAuditTasks",
                "iot:CreateScheduledAudit",
                "iot:UpdateScheduledAudit",
                "iot:DeleteScheduledAudit",
                "iot:DescribeScheduledAudit",
                "iot:ListScheduledAudits",
```

```
                "iot:ListAuditFindings",
                "iot:CreateSecurityProfile",
                "iot:DescribeSecurityProfile",
                "iot:UpdateSecurityProfile",
                "iot:DeleteSecurityProfile",
                "iot:AttachSecurityProfile",
                "iot:DetachSecurityProfile",
                "iot:ListSecurityProfiles",
                "iot:ListSecurityProfilesForTarget",
                "iot:ListTargetsForSecurityProfile",
                "iot:ListActiveViolations",
                "iot:ListViolationEvents",
                "iot:ValidateSecurityProfileBehaviors"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 托管策略： AWSIoTConfigReadOnlyAccess

您可以将 `AWSIoTConfigReadOnlyAccess` 策略附加到 IAM 身份。

此策略向相关身份授予权限，以允许只读访问所有 AWS IoT 配置操作。要在中查看此政策 AWS Management Console，请参阅AWSIoTConfigReadOnlyAccess。

权限详细信息

该策略包含以下权限。

- `iot` – 对 IoT 配置操作执行只读操作。

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

```
                "iot:DescribeAuthorizer",
                "iot:DescribeCACertificate",
                "iot:DescribeCertificate",
                "iot:DescribeDefaultAuthorizer",
                "iot:DescribeEndpoint",
                "iot:DescribeEventConfigurations",
                "iot:DescribeIndex",
                "iot:DescribeJob",
                "iot:DescribeJobExecution",
                "iot:DescribeRoleAlias",
                "iot:DescribeStream",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingRegistrationTask",
                "iot:DescribeThingType",
                "iot:GetEffectivePolicies",
                "iot:GetIndexingConfiguration",
                "iot:GetJobDocument",
                "iot:GetLoggingOptions",
                "iot:GetOTAUpdate",
                "iot:GetPolicy",
                "iot:GetPolicyVersion",
                "iot:GetRegistrationCode",
                "iot:GetTopicRule",
                "iot:GetV2LoggingOptions",
                "iot:ListAttachedPolicies",
                "iot:ListAuthorizers",
                "iot:ListCACertificates",
                "iot:ListCertificates",
                "iot:ListCertificatesByCA",
                "iot:ListIndices",
                "iot:ListJobExecutionsForJob",
                "iot:ListJobExecutionsForThing",
                "iot:ListJobs",
                "iot:ListOTAUpdates",
                "iot:ListOutgoingCertificates",
                "iot:ListPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPolicyVersions",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListRoleAliases",
                "iot:ListStreams",
                "iot:ListTargetsForPolicy",
```

```
                "iot:ListThingGroups",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:ListThingRegistrationTaskReports",
                "iot:ListThingRegistrationTasks",
                "iot:ListThings",
                "iot:ListThingsInThingGroup",
                "iot:ListThingTypes",
                "iot:ListTopicRules",
                "iot:ListV2LoggingLevels",
                "iot:SearchIndex",
                "iot:TestAuthorization",
                "iot:TestInvokeAuthorizer",
                "iot:DescribeAccountAuditConfiguration",
                "iot:DescribeAuditTask",
                "iot:ListAuditTasks",
                "iot:DescribeScheduledAudit",
                "iot:ListScheduledAudits",
                "iot:ListAuditFindings",
                "iot:DescribeSecurityProfile",
                "iot:ListSecurityProfiles",
                "iot:ListSecurityProfilesForTarget",
                "iot:ListTargetsForSecurityProfile",
                "iot:ListActiveViolations",
                "iot:ListViolationEvents",
                "iot:ValidateSecurityProfileBehaviors"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 托管策略： AWSIoTDataAccess

您可以将 AWSIoTDataAccess 策略附加到 IAM 身份。

此策略授予相关的身份权限，允许访问所有 AWS IoT 数据操作。数据操作通过 MQTT 或 HTTP 协议发送数据。要在 AWS Management Console中查看该策略，请参阅 AWSIoTDataAccess。

权限详细信息

该策略包含以下权限。

- `iot`— 检索 AWS IoT 数据并允许对 AWS IoT 消息操作进行完全访问权限。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect",
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow",
                "iot:ListNamedShadowsForThing"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 托管策略： AWSIoTFullAccess

您可以将 `AWSIoTFullAccess` 策略附加到 IAM 身份。

此策略向相关身份授予权限，以允许访问所有 AWS IoT 配置和消息收发操作。要在中查看此政策 AWS Management Console，请参阅`AWSIoTFullAccess`。

权限详细信息

该策略包含以下权限。

- `iot`— 检索 AWS IoT 数据并允许完全访问 AWS IoT 配置和消息传送操作。
- `iotjobsdata`— 检索 AWS IoT 作业数据并允许对 AWS IoT 作业数据平面 API 操作进行完全访问权限。

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

AWS 托管策略： AWSIoTLogging

您可以将 `AWSIoTLogging` 策略附加到 IAM 身份。

此策略授予相关的身份权限，允许用户创建 Amazon CloudWatch Logs 群组并将日志流式传输到这些群组。此策略已附加到您的 CloudWatch 日志记录角色。要在中查看此政策 AWS Management Console，请参阅AWSIoTLogging。

权限详细信息

该策略包含以下权限。

- `logs`— 检索 CloudWatch 日志。还允许创建 CloudWatch 日志组并将日志流式传输到这些组。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:PutMetricFilter",
                "logs:PutRetentionPolicy",
                "logs:GetLogEvents",
                "logs:DeleteLogStream"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS 托管策略： AWSIoTOTAUpdate

您可以将 AWSIoTOTAUpdate 策略附加到 IAM 身份。

此策略授予相关的身份权限，允许访问创建 AWS IoT 作业、 AWS IoT 代码签名作业和描述 AWS 代码签名者作业。要在中查看此政策 AWS Management Console，请参阅AWSIoTOTAUpdate。

权限详细信息

该策略包含以下权限。

- iot— 创建 AWS IoT 工作和代码签名作业。
- signer— 创建 AWS 代码签名者作业。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iot:CreateJob",
            "signer:DescribeSigningJob"
        ],
        "Resource": "*"
    }
}
```

AWS 托管策略： AWSIoTRuleActions

您可以将 AWSIoTRuleActions 策略附加到 IAM 身份。

此策略授予相关的身份权限，允许访问 AWS IoT 规则操作中支持的所有 AWS 服务人。要在中查看此政策 AWS Management Console，请参阅AWSIoTRuleActions。

权限详细信息

该策略包含以下权限。

- iot - 执行用于发布规则操作消息的操作。
- dynamodb - 将消息插入到 DynamoDB 表或将消息拆分为 DynamoDB 表的多列。
- s3 - 将对象存储在 Amazon S3 存储桶中。
- kinesis - 将消息发送到 Amazon Kinesis 流对象。
- firehose-在 Firehose 直播对象中插入一条记录。
- cloudwatch-更改 CloudWatch 警报状态或向 CloudWatch指标发送消息数据。
- sns - 执行使用 Amazon SNS 发布通知的操作。此操作的范围仅限于 AWS IoT SNS 主题。
- sqs - 插入要添加到 SQS 队列的消息。
- es-向 OpenSearch 服务服务发送消息。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "dynamodb:PutItem",
            "kinesis:PutRecord",
            "iot:Publish",
            "s3:PutObject",
            "sns:Publish",
            "sqs:SendMessage*",
            "cloudwatch:SetAlarmState",
            "cloudwatch:PutMetricData",
            "es:ESHttpPut",
            "firehose:PutRecord"
        ],
        "Resource": "*"
    }
}
```

AWS 托管策略： AWSIoTThingsRegistration

您可以将 AWSIoTThingsRegistration 策略附加到 IAM 身份。

此策略向相关身份授予权限，以允许使用 StartThingRegistrationTask API 批量注册事物。此策略可能会影响数据处理和存储。要在中查看此政策 AWS Management Console，请参阅AWSIoTThingsRegistration。

权限详细信息

该策略包含以下权限。

- iot - 批量注册时，执行用于创建内容以及附加策略和证书的操作。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachPrincipalPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateCertificateFromCsr",
                "iot:CreatePolicy",
                "iot:CreateThing",
                "iot:DescribeCertificate",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:DescribeThingType",
                "iot:DetachPolicy",
                "iot:DetachThingPrincipal",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListPolicyPrincipals",
                "iot:ListPrincipalPolicies",
                "iot:ListPrincipalThings",
                "iot:ListTargetsForPolicy",
                "iot:ListThingGroupsForThing",
                "iot:ListThingPrincipals",
                "iot:RegisterCertificate",
                "iot:RegisterThing",
                "iot:RemoveThingFromThingGroup",
                "iot:UpdateCertificate",
                "iot:UpdateThing",
                "iot:UpdateThingGroupsForThing",
                "iot:AddThingToBillingGroup",
                "iot:DescribeBillingGroup",
                "iot:RemoveThingFromBillingGroup"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

AWS IoTAWS 托管策略的更新

查看 AWS IoT 自该服务开始跟踪这些更改以来 AWS 托管策略更新的详细信息。要获得有关此页面变更的自动提醒，请订阅 “ AWS IoT 文档历史记录” 页面上的 RSS feed。

更改	描述	日期
AWSIoTFullAccess – 对现有策略的更新	AWS IoT 添加了新的权限，允许用户使用 HTTP 协议访问 AWS IoT 任务数据平面 API 操作。 新的 IAM 策略前缀可为您提供更精细的访问控制，以访问 AWS IoT 任务数据平面终端节点。`iotjobsdata:` 对于控制面板 API 操作，您仍然使用 `iot:` 前缀。有关更多信息，请参阅 AWS IoT Core HTTPS 协议的策略。	2022 年 5 月 11 日
AWS IoT 开始跟踪更改	AWS IoT 开始跟踪其 AWS 托管策略的更改。	2022 年 5 月 11 日

对 AWS IoT 身份和访问进行故障排除

使用以下信息来帮助您诊断和修复在使用 AWS IoT 和 IAM 时可能遇到的常见问题。

主题

- 我无权在以下位置执行操作 AWS IoT
- 我无权执行 iam：PassRole
- 我想允许我以外的人 AWS 账户 访问我的 AWS IoT 资源

我无权在以下位置执行操作 AWS IoT

如果您收到错误提示，表明您无权执行某个操作，则您必须更新策略以允许执行该操作。

如果 IAM 用户 mateojackson 尝试使用控制台查看有关事物资源的详细信息，但没有 iot:*DescribeThing* 权限，则会出现以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iot:DescribeThing on resource: MyIoTThing
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 iot:*DescribeThing* 操作访问事物资源。

如果您需要帮助，请联系您的 AWS 管理员。您的管理员是提供登录凭证的人。

使用 AWS IoT 设备顾问

如果您使用的是 De AWS IoT vice Advisor，则当用户mateojackson尝试使用控制台查看有关套件定义的详细信息但没有iotdeviceadvisor:*GetSuiteDefinition*权限时，会出现以下示例错误。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 iotdeviceadvisor:GetSuiteDefinition on resource: MySuiteDefinition
```

在此情况下，必须更新 mateojackson 用户的策略，以允许使用 iotdeviceadvisor:*GetSuiteDefinition* 操作访问 *MySuiteDefinition* 资源。

我无权执行 iam：PassRole

如果您收到一个错误，表明您无权执行 iam:PassRole 操作，则必须更新策略以允许您将角色传递给 AWS IoT。

有些 AWS 服务 允许您将现有角色传递给该服务，而不是创建新的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。您的管理员是提供登录凭证的人。

我想允许我以外的人 AWS 账户 访问我的 AWS IoT 资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以担任角色。对于支持基于资源的策略或访问控制列表（ACL）的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解是否 AWS IoT 支持这些功能，请参阅如何 AWS IoT 与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过联合身份验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（联合身份验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问的区别，请参阅 IA M 用户指南中的跨账户资源访问。

日志记录和监控

监控是维护 AWS 解决方案的可靠性、可用性和性能的重要组成部分。 AWS IoT 您应该从 AWS 解决方案的各个部分收集监控数据，以便在出现多点故障时可以更轻松地进行调试。有关日志记录和监控流程的信息，请参阅监控 AWS IoT

监控工具

AWS 提供了可用于监控的工具 AWS IoT。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。

自动监控工具

您可以使用以下自动监控工具来监视 AWS IoT 和报告何时出现问题：

- A CloudWatch mazon Alarms — 在您指定的时间段内观察单个指标，并根据该指标在多个时间段内相对于给定阈值的值执行一项或多项操作。该操作是发送到亚马逊简单通知服务 (Amazon SNS) Simple Notification Scaling 主题或亚马逊 EC2 Auto Scaling 策略的通知。 CloudWatch 警报不会仅仅因为它们处于特定状态就调用操作。该状态必须已改变并在指定的若干个时间段内保持不变。有关更多信息，请参阅 使用 Amazon 监控 AWS IoT 警报和指标 CloudWatch。

- Amazon CloudWatch Logs — 监控、存储和访问来自 AWS CloudTrail 或其他来源的日志文件。Ama CloudWatch zon Logs 还允许您查看 AWS IoT 设备顾问测试用例采取的关键步骤、生成的事件以及从您的设备或测试执行 AWS IoT Core 期间发送的 MQTT 消息。这些日志使您能够在设备上进行调试和采取纠正措施。有关更多信息，请参阅AWS IoT 使用 CloudWatch 日志进行监控有关使用 Amazon 的更多信息 CloudWatch，请参阅亚马逊 CloudWatch 用户指南中的监控日志文件。
- Amazon CloudWatch Events — 匹配事件并将其路由到一个或多个目标函数或流，以进行更改、捕获状态信息并采取纠正措施。有关更多信息，请参阅《亚马逊 CloudWatch 用户指南》中的什么是亚马逊 CloudWatch活动。
- AWS CloudTrail 日志监控-在账户之间共享日志文件，通过将 CloudTrail 日志文件发送到 “日志” 来实时监控 CloudWatch 日志文件，用 Java 编写日志处理应用程序，并验证您的日志文件在传送后是否未更改 CloudTrail。有关更多信息，请参阅使用记录 AWS IoT API 调用 AWS CloudTrail《AWS CloudTrail 用户指南》中的 “使用 CloudTrail 日志文件”。

手动监控工具

监控 AWS IoT 的另一个重要部分是手动监控 CloudWatch 警报未涵盖的项目。 AWS IoT CloudWatch、和其他 AWS 服务控制台仪表板提供了 AWS 环境状态的 at-a-glance 视图。我们建议您同时查看日志文件 AWS IoT。

- AWS IoT 仪表板显示：
  - CA 证书
  - 证书
  - 策略
  - 规则
  - 事物
- CloudWatch 主页显示：
  - 当前警报和状态。
  - 警报和资源的图表。
  - 服务运行状况。

  您可以使用 CloudWatch 执行以下操作：
  - 创建自定义控制面板以监控您关心的服务。
  - 绘制指标数据图，以排除问题并弄清楚趋势。
  - 搜索并浏览您的所有 AWS 资源指标。

- 创建和编辑告警接收有关问题的通知。

C AWS IoT ore 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 安全与合规性 — 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AWS 规资源 — 此工作簿和指南集可能适用于您的行业和所在地区。
- 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、行业准则和法规。
- 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 物联网核心的弹性

AWS 全球基础设施是围绕 AWS 区域 s和可用区构建的。 AWS 区域 s 提供多个物理分隔和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络连接。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架构相比，可用区具有更高的可用性、容错性和可扩展性。

有关 AWS 区域 s 和可用区的更多信息，请参阅AWS 全球基础设施。

AWS IoT Core 在设备注册表中存储有关您的设备的信息。此外，它还存储 CA 证书、设备证书和设备影子数据。发生硬件或网络故障时，此数据会跨可用区自动复制，但不会跨区域复制。

AWS IoT Core 更新设备注册表时发布 MQTT 事件。您可以使用这些消息来备份注册表数据并将其保存到某个位置（如 DynamoDB 表）。您负责保存为您 AWS IoT Core 创建的证书或您自己创建的证书。设备影子存储有关您设备的状态数据，并且可以在设备恢复在线时重新发送。 AWS IoT 设备顾问存储有关您的测试套件配置的信息。发生硬件或网络故障时，此数据将自动复制。

AWS IoT Core 资源是特定于区域的， AWS 区域 除非您特别这样做，否则不会跨区域复制。

有关安全性最佳实践的更多信息，请参阅 中的安全最佳实践 AWS IoT Core。

AWS IoT Core 与接口 VPC 终端节点一起使用

借助 AWS IoT Core，您可以使用接口 VPC 终端节点在您的虚拟私有云 (VPC) 中创建物联网数据终端节点。接口 VPC 终端节点由 AWS PrivateLink一种 AWS 技术提供支持，您可以使用该技术 AWS 通过私有 IP 地址访问在其上运行的服务。有关更多信息，请参阅 Amazon Virtual Private Cloud。

要将远程网络（例如公司网络）上的现场设备连接到您的 Amazon VPC，请参阅网络到 Amazon VPC 连接表中列出的选项。

内容

- 为 AWS IoT Core 数据平面创建 VPC 终端节点
- 为 AWS IoT Core 凭证提供商创建 VPC 端点
- 创建 Amazon VPC 接口端点

- 配置私有托管区域
- 控制 AWS IoT Core 通过 VPC 终端节点的访问权限
- 限制
- 使用扩展 VPC 终端节点 AWS IoT Core
- 将自定义域用于 VPC 终端节点
- VPC 终端节点的可用性 AWS IoT Core

为 AWS IoT Core 数据平面创建 VPC 终端节点

您可以为 AWS IoT Core 数据平面 API 创建 VPC 终端节点，将您的设备连接到 AWS IoT 服务和其他 AWS 服务。要开始使用 VPC 终端节点，请创建一个接口 VPC 终端节点并选择 AWS IoT Core 作为 AWS 服务。如果您使用的是 CLI，请先调用 desc ribe-vpc-endpoint-services，以确保您选择的是您的特定可用区。 AWS IoT Core AWS 区域例如，在 us-east-1 中，此命令将类似于：

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-east-1.iot.data
```

> **Note**
>
> 用于自动创建 DNS 记录的 VPC 特征已被禁用。要连接到这些端点，您必须手动创建私有 DNS 记录。有关私有 VPC DNS 记录的更多信息，请参阅 接口终端节点的私有 DNS。有关 AWS IoT Core VPC 限制的更多信息，请参阅限制。

要将 MQTT 客户端连接到 VPC 端点接口，请执行以下操作：

- 您必须在附加到 VPC 的私有托管区域中手动创建 DNS 记录。要开始使用，请参阅创建私有托管区域。

- 在您的私有托管区域中，为 VPC 端点的每个弹性网络接口 IP 创建别名记录。如果您有多个 VPC 终端节点的多个网络接口 IP，请在所有加权记录之间创建权重相等的加权 DNS 记录。按描述字段中的 VPC 终端节点 ID 筛选后，可从DescribeNetwork接口 API 调用中获得这些 IP 地址。

有关创建 Amazon VPC 接口终端节点和为 AWS IoT Core 数据平面配置私有托管区域，请参阅以下详细说明。

为 AWS IoT Core 凭证提供商创建 VPC 端点

您可以为 AWS IoT Core 凭证提供者创建 VPC 终端节点，以便使用基于客户端证书的身份验证连接设备并获取AWS 签名版本 4 AWS 格式的临时证书。要开始使用 AWS IoT Core 证书提供者的 VPC 终端节点，请运行 create-vpc-endpoint CLI 命令来创建接口 VPC 终端节点，然后 AWS IoT Core 选择凭证提供程序作为服务。 AWS 为确保您选择的可用区与您的特定区域相同，请先运行 AWS IoT Core describe-vpc- AWS 区域 endpoint-services 命令。例如，在 us-east-1 中，此命令将类似于：

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-
east-1.iot.credentials
```

> **Note**
>
> 用于自动创建 DNS 记录的 VPC 特征已被禁用。要连接到这些端点，您必须手动创建私有 DNS 记录。有关私有 VPC DNS 记录的更多信息，请参阅 接口终端节点的私有 DNS。有关 AWS IoT Core VPC 限制的更多信息，请参阅限制。

要将 HTTP 客户端连接到 VPC 端点接口，请执行以下操作：

- 您必须在附加到 VPC 的私有托管区域中手动创建 DNS 记录。要开始使用，请参阅创建私有托管区域。

- 在您的私有托管区域中，为 VPC 端点的每个弹性网络接口 IP 创建别名记录。如果您有多个 VPC 终端节点的多个网络接口 IP，请在所有加权记录之间创建权重相等的加权 DNS 记录。按描述字段中的 VPC 终端节点 ID 筛选后，可从DescribeNetwork接口 API 调用中获得这些 IP 地址。

有关创建 Amazon VPC 接口终端节点和为 AWS IoT Core 凭证提供者配置私有托管区域的信息，请参阅以下详细说明。

创建 Amazon VPC 接口端点

您可以创建接口 VPC 终端节点以连接到由提供支持的 AWS 服务 AWS PrivateLink。使用以下步骤创建连接到 AWS IoT Core 数据平面或 AWS IoT Core 凭据提供程序的接口 VPC 终端节点。有关更多信息，请参阅使用接口 VPC 终端节点访问 AWS 服务。

> **Note**
>
> 为 AWS IoT Core 数据平面和 AWS IoT Core 证书提供者创建 Amazon VPC 接口终端节点的过程类似，但您必须对终端节点进行特定更改才能使连接正常运行。

使用 VPC 端点控制台创建接口 VPC 端点

1. 导航到 VPC 端点控制台，在左侧菜单的虚拟私有云下，选择端点，然后选择创建端点。
2. 在创建端点页面上，指定以下信息。
   - 为 Service category（服务类别）选择AWS 服务。
   - 对于 Service Name（服务名称），通过输入关键字 `iot` 进行搜索。在显示的 `iot` 服务列表中，请选择端点。

     如果您为 AWS IoT Core 数据平面创建 VPC 终端节点，请为您的区域选择 AWS IoT Core 数据平面 API 终端节点。终端节点的格式为 `com.amazonaws.`*`region`*`.iot.data`。

     如果您为 AWS IoT Core 凭证提供者创建 VPC 终端节点，请选择您所在地区的 AWS IoT Core 凭证提供商终端节点。终端节点的格式为 `com.amazonaws.`*`region`*`.iot.credentials`。

     > **Note**
     >
     > 中国区域 AWS IoT Core 数据平面的服务名称将采用以下格式`cn.com.amazonaws.`*`region`*`.iot.data`。中国区域不支持为 AWS IoT Core 凭证提供者创建 VPC 终端节点。

   - 对于 VPC 和 Subnets（子网），选择要在其中创建端点的 VPC 和要在其中创建端点网络的可用区 (AZ)。
   - 对于 Enable DNS name（启用 DNS 名称），请确保未选择 Enable for this endpoint（为此端点启用）。 AWS IoT Core 数据平面和 AWS IoT Core 凭据提供商都不支持私有 DNS 名称。
   - 对于 Security group（安全组），选择要与端点网络接口关联的安全组。
   - 您可以选择添加或删除标签。标签是用于与端点关联的名称-值对。
3. 要创建 VPC 终端节点，请选择 Create endpoint（创建端点）。

创建 AWS PrivateLink 终端节点后，在终端节点的详细信息选项卡中，您将看到 DNS 名称列表。您可以使用在本部分中创建的这些 DNS 名称中的一个来配置私有托管区域。

配置私有托管区域

您可以使用在上一部分中创建的这些 DNS 名称中的一个来配置私有托管区域。

对于 AWS IoT Core 数据平面

DNS 名称必须是您的域配置名称或 `IoT:Data-ATS` 端点。DNS 名称的示例，可以是：*`xxx`*`-ats.data.iot.`*`region`*`.amazonaws.com`。

对于 AWS IoT Core 凭证提供商

DNS 名称必须是您的 `iot:CredentialProvider` 端点。DNS 名称的示例，可以是：*`xxxx`*`.credentials.iot.`*`region`*`.amazonaws.com`。

> **Note**
>
> 为 AWS IoT Core 数据平面和 AWS IoT Core 凭据提供程序配置私有托管区域的过程类似，但是您必须对端点进行特定更改才能使连接正常运行。

创建私有托管区域

使用 Route 53 控制台创建私有托管区域

1. 导航到 Route 53 托管区域控制台并选择 Create hosted zone（创建托管区域）。
2. 在 Create hosted zone（创建托管区域）页面上，指定以下信息。
   - 在域名中，输入您的 `iot:Data-ATS` 或 `iot:CredentialProvider` 端点的端点地址。以下 AWS CLI 命令显示如何通过公共网络获取端点：`aws iot describe-endpoint --endpoint-type iot:Data-ATS` 或 `aws iot describe-endpoint --endpoint-type iot:CredentialProvider`。

     > **Note**
     >
     > 如果您使用的是自定义域，请参阅将自定义域与 VPC 端点结合使用。 AWS IoT Core 凭据提供者不支持自定义域名。

   - 对于 Type（类型），选择 Private hosted zone（私有托管区域）。
   - 或者，您可以添加或删除要与托管区域关联的标签。

3. 要创建您的私有托管区域，请选择 Create hosted zone（创建托管区域）。

有关更多信息，请参阅创建私有托管区域。

创建记录

创建私有托管区域后，您可以创建一个记录，告诉 DNS 如何将流量路由到该域。

创建记录

1. 在显示的托管区域列表中，选择您之前创建的私有托管区域，然后选择 Create record（创建记录）。
2. 使用向导提供的方法创建记录。如果控制台为您提供了 Quick create（快速创建）的方法，选择 Switch to wizard（切换到向导）。
3. 在 Routing policy（路由策略）中选择 Simple Routing（简单路由），然后选择 Next（下一步）。
4. 在 Configure records（配置记录）下，选择 Define simple record（定义简单记录）。
5. 在 Define simple record（定义简单记录）页面:
   - 在记录名称中，输入 `iot:Data-ATS` 或 `iot:CredentialProvider` 端点。该名称必须与私有托管区名称相同。
   - 对于 Record type（记录类型），请将该值保持为 `A - Routes traffic to an IPv4 address and some AWS resources`。
   - 对于 Value/Route traffic to（值/流量路由至），选择 Alias to VPC endpoint（向 VPC 添加别名）。然后选择您的 Region（区域），接着从显示的端点列表中选择您之前创建的端点，如 ??? 中所述。
6. 选择 Define simple record（定义简单记录）以创建您的记录。

控制 AWS IoT Core 通过 VPC 终端节点的访问权限

您可以使用 VPC 条件上下文密钥 AWS IoT Core将设备访问限制为仅允许通过 VPC 终端节点。 AWS IoT Core 支持以下与 VPC 相关的上下文密钥：

- SourceVpc
- SourceVpce
- VPC SourceIp

> **Note**
>
> AWS IoT Core 不支持 VPC 终端节点的终端节点策略。

例如，以下策略授予 AWS IoT Core 使用与事物名称匹配的客户端 ID 进行连接的权限，以及向以事物名称为前缀的任何主题发布内容的权限，条件是设备连接到具有特定 VPC 终端节点 ID 的 VPC 终端节点。此策略将拒绝连接到您的公有 IoT 数据终端节点的尝试。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }

        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}/*"
            ]
        }
    ]
}
```

限制

目前，仅 AWS IoT Core 数据端点和 AWS IoT Core 凭证提供商端点支持 VPC 端点。

IoT 数据 VPC 端点的限制

本部分介绍 IoT 数据 VPC 端点的限制。

- MQTT 保持活动状态的时间限制为 230 秒。保持活动状态超过该时段的时间将自动减少到 230 秒。
- 每个 VPC 终端节点总共支持 100,000 台并发互联设备。如果您需要更多连接，请参阅 使用扩展 VPC 终端节点 AWS IoT Core。
- VPC 终端节点仅支持 IPv4 流量。
- VPC 终端节点将仅提供 ATS 证书，但自定义域除外。
- 不支持 VPC 端点策略。
- 对于为 AWS IoT Core 数据平面创建的 VPC 终端节点， AWS IoT Core 不支持使用地区或区域公有 DNS 记录。

凭证提供商端点的限制

本部分介绍凭证提供商 VPC 端点的限制。

- VPC 终端节点仅支持 IPv4 流量。
- VPC 端点将仅提供 ATS 证书。
- 不支持 VPC 端点策略。
- 凭证提供商端点不支持自定义域名。
- 对于为 AWS IoT Core 凭证提供者创建的 VPC 终端节点， AWS IoT Core 不支持使用地区或区域公有 DNS 记录。

使用扩展 VPC 终端节点 AWS IoT Core

AWS IoT Core 接口 VPC 终端节点限制为通过单个接口终端节点连接的 100,000 台设备。如果您的使用案例需要更多到代理的并发连接，那么我们建议您使用多个 VPC 终端节点并在接口终端节点之间手动路由您的设备。创建私有 DNS 记录将流量路由到 VPC 终端节点时，确保创建与 VPC 终端节点 数量相同的加权记录，以便在多个终端节点之间 分配流量。

将自定义域用于 VPC 终端节点

如果要将自定义域与 VPC 端点结合使用，必须在私有托管区域中创建自定义域名记录，并在 Route53 中创建路由记录。有关更多信息，请参阅创建私有托管区域。

> **Note**
>
> 仅 AWS IoT Core 数据端点支持自定义域名。

VPC 终端节点的可用性 AWS IoT Core

AWS IoT Core 所有AWS IoT Core 支持的区域均提供接口 VPC 终端节点。 AWS IoT Core 中国地区不支持 AWS IoT Core 凭证提供商的接口 VPC 终端节点，以及。 AWS GovCloud (US) Regions

中的基础设施安全 AWS IoT

作为托管服务的集合， AWS IoT 受《Amazon Web Services：安全流程概述》白皮书中描述的 AWS 全球网络安全程序的保护。

您可以使用 AWS 已发布的 API 调用 AWS IoT 通过网络进行访问。客户端必须支持传输层安全性（TLS）1.2 或更高版本。客户端还必须支持具有完全向前保密（PFS）的密码套件，例如 Ephemeral Diffie-Hellman（DHE）或 Elliptic Curve Ephemeral Diffie-Hellman（ECDHE）。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。有关更多信息，请参阅 运输安全 AWS IoT Core。

必须使用访问密钥 ID 以及与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service（AWS STS）生成临时安全凭证来对请求进行签名。

使用 Core 对生产车队或设备进行 AWS IoT 安全监控

IoT 实例集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。这些特性导致实例集设置复杂且 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 实例集成为吸引黑客的目标，而且导致难以持续保护设备实例集安全。

AWS IoT Device Defender 通过提供工具来识别安全问题和与最佳实践的偏差，从而应对这些挑战。您可以使用 De AWS IoT vice Defender 来分析、审计和监控连接的设备，以检测异常行为并降低安全风

险。 AWS IoT Device Defender 可以对设备队列进行审计，以确保它们遵守安全最佳实践并检测设备上的异常行为。这使得在您的 AWS IoT 设备群中实施一致的安全策略成为可能，并在设备遭到入侵时快速做出响应。有关更多信息，请参阅 AWS IoT Device Defender。

AWS IoT Device Advisor 会根据需要推送更新并修补您的机群。 AWS IoT 设备顾问会自动更新测试用例。 您选择 的测试用例始终为 最新版本。有关更多信息，请参阅 Device Advisor。

中的安全最佳实践 AWS IoT Core

本节包含有关安全最佳实践的信息 AWS IoT Core。有关工业 IoT 解决方案的安全规则的信息，请参阅工业 IoT 解决方案的十大安全黄金规则。

保护中的 MQTT 连接 AWS IoT

AWS IoT Core是一项托管云服务，可让联网设备轻松安全地与云应用程序和其他设备进行交互。AWS IoT Core 支持 HTTP WebSocket、和 MQTT，这是一种专为容忍间歇性连接而设计的轻量级通信协议。如果您 AWS IoT 使用 MQTT 进行连接，则每个连接都必须与称为客户端 ID 的标识符相关联。MQTT 客户端 ID 唯一地标识 MQTT 连接。如果使用已为其他连接申请的客户端 ID 建立新连接，则 AWS IoT 消息代理会丢弃旧连接以允许新连接。每个 AWS 账户 客户端 ID 必须是唯一的 AWS 区域。这意味着，您无需强制客户端 ID 在您所在区域之外 AWS 账户 或您所在区域的不同区域之间保持全球唯一性。 AWS 账户

在设备实例集上删除 MQTT 连接的影响和严重程度取决于许多因素。其中包括：

- 您的用例（例如，您的设备发送到的数据 AWS IoT、数据量以及发送数据的频率）。
- 您的 MQTT 客户端配置（例如，自动重新连接设置、关联的退避计时以及使用 MQTT 持久性会话）。
- 设备资源限制。
- 断开连接的根本原因、其主动性和持久性。

为避免客户端 ID 冲突及其潜在的负面影响，请确保每台设备或移动应用程序都有 AWS IoT 或 IAM 政策，限制哪些客户端 ID 可用于与 AWS IoT 消息代理的 MQTT 连接。例如，您可以使用 IAM policy 防止设备无意中通过使用已在使用的客户端 ID 关闭其它设备的连接。有关更多信息，请参阅 授权。

您的队列中的所有设备都必须具有仅授权预期操作的权限的证书，这些权限包括（但不限于） AWS IoT MQTT 操作，例如发布消息或订阅具有特定范围和上下文的主题。具体的权限策略可能因您的使用案例而异。确定最能满足您的业务和安全要求的权限策略。

要简化权限策略的创建和管理，您可以使用 AWS IoT Core 策略变量 和 IAM policy 变量。策略变量可以放在策略中，并且在评估策略时，变量将由来自设备请求的值替换。使用策略变量，您可以创建单个策略以授予对多个设备的权限。您可以根据您的 AWS IoT 账户配置、身份验证机制和连接到 AWS IoT 消息代理时使用的网络协议来确定与您的用例相关的策略变量。但是，要编写最佳权限策略，应考虑使用案例和威胁模型的具体情况。

例如，如果您在注册 AWS IoT 表中注册了设备，则可以在策略中 AWS IoT 使用事物策略变量，根据事物名称、事物类型和事物属性值等事物属性来授予或拒绝权限。事物名称是从事物连接到时发送的 MQTT 连接消息中的客户端 ID 中获取的 AWS IoT。当事物使用 TLS 相互身份验证通过 MQTT 连接或使用 AWS IoT 经过身份验证的 Amazon Cognito 身份通过 WebSocket 协议连接 MQTT 时，事物策略变量将被替换。您可以使用AttachThing委托人 API 将证书和经过身份验证的 Amazon Cognito 身份附加到事物。 `iot:Connection.Thing.ThingName`是强制执行客户端 ID 限制的有用策略变量。以下示例 AWS IoT 策略要求将已注册事物的名称用作与 AWS IoT 消息代理的 MQTT 连接的客户端 ID：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": "iot:Connect",
   "Resource": [
    "arn:aws:iot:us-east-1:123456789012:client/${iot:Connection.Thing.ThingName}"
   ]
  }
 ]
}
```

如果要识别持续存在的客户端 ID 冲突，可以启用和使用CloudWatch 日志 AWS IoT。对于 AWS IoT 消息代理由于客户端 ID 冲突而断开连接的每个 MQTT 连接，都会生成一条类似于以下内容的日志记录：

```
{
    "timestamp": "2019-04-28 22:05:30.105",
    "logLevel": "ERROR",
    "traceId": "02a04a93-0b3a-b608-a27c-1ae8ebdb032a",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "Disconnect",
    "protocol": "MQTT",
    "clientId": "clientId01",
    "principalId": "1670fcf6de55adc1930169142405c4a2493d9eb5487127cd0091ca0193a3d3f6",
```

```
    "sourceIp": "203.0.113.1",
    "sourcePort": 21335,
    "reason": "DUPLICATE_CLIENT_ID",
    "details": "A new connection was established with the same client ID"
}
```

您可以使用CloudWatch 日志筛选器，例如{$.reason= "DUPLICATE_CLIENT_ID" }搜索客户端 ID 冲突的实例，或者设置CloudWatch 指标筛选器和相应的 CloudWatch 警报以进行持续监控和报告。

您可以使用 Dev AWS IoT ice Defender 来识别过于宽松的策略 AWS IoT 和 IAM 策略。 AWS IoT Device Defender 还提供审计检查，如果您的队列中有多台设备使用相同的客户端 ID 连接到 AWS IoT 消息代理，则会通知您。

您可以使用 AWS IoT Device Advisor 来验证您的设备能否可靠地连接 AWS IoT Core 并遵循安全最佳实践。

另请参阅

- AWS IoT Core
- AWS IoT的安全特征
- AWS IoT Core 策略变量
- IAM policy 变量
- Amazon Cognito 身份
- AWS IoT Device Defender
- CloudWatch 的日志 AWS IoT

使设备的时钟保持同步

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。如果您要构建商用 IoT 设备，请记住，您的产品在出售前可能会存放较长时间。在此期间，实时时钟可能会出现偏移误差，并且电池可能会放电，因此在工厂设置时间是不够的。

对于大多数系统，这意味着设备的软件必须包含网络时间协议 (NTP) 客户端。设备应等待，直到它与 NTP 服务器同步，然后再尝试连接到 AWS IoT Core。如果无法做到这一点，系统将为用户提供一种设置设备时间的方法，以便后续连接成功。

设备与 NTP 服务器同步后，便能打开与 AWS IoT Core的连接。允许的时钟偏移量取决于您尝试通过此连接执行的操作。

验证服务器证书

设备与之交互的第一件事 AWS IoT 就是打开安全连接。将设备连接到时 AWS IoT，请确保您正在与服务器通话 AWS IoT ，而不是其他服务器在模仿 AWS IoT。每 AWS IoT 台服务器都配置了为该`iot.amazonaws.com`域颁发的证书。该证书 AWS IoT 由可信的证书颁发机构颁发，该机构验证了我们的身份和域名所有权。

设备连接时要 AWS IoT Core 做的第一件事就是向设备发送服务器证书。设备可以验证它们是否希望连接到 `iot.amazonaws.com`，并验证位于该连接一端的服务器是否拥有来自该域的受信任颁发机构的证书。

TLS 证书采用 X.509 格式，并包含各种信息，例如组织的名称、位置、域名和有效期。有效期被指定为一对时间值（分别名为 `notBefore` 和 `notAfter`）。 AWS IoT Core 诸如服务器证书的有效期有限（例如，一年），并在旧证书到期之前开始提供新证书。

每个设备使用一个身份

每个客户端使用一个身份。设备通常使用 X.509 客户端证书。Web 和移动应用程序使用 Amazon Cognito Identity。这使您能够对设备应用细化权限。

例如，您有一个由手机设备构成的应用程序，这些设备接收来自两个不同的智能家居对象（灯泡和恒温器）的状态更新。灯泡发送电池电量的状态，恒温器发送报告温度的消息。

AWS IoT 对设备进行单独身份验证并单独处理每个连接。您可以使用授权策略应用精细访问控制。您可以为恒温器定义一个策略，该策略允许恒温器发布到主题空间的策略。您可以为灯泡定义一个单独策略，该策略允许灯泡发布到不同的主题空间。最后，您可以为移动应用程序定义一个策略，该策略只允许它连接到和订阅恒温器和灯泡的主题以接收来自这些设备的消息。

应用最小权限原则，并尽可能缩小每个设备的权限范围。所有设备或用户都应 AWS IoT 制定政策 AWS IoT ，仅允许其使用已知的客户端 ID 进行连接，以及发布和订阅已确定和固定的主题集。

用一秒钟 AWS 区域 作为备份

考虑在一秒钟内存储一份数据副本 AWS 区域 作为备份。请注意，名为 “灾难恢复” 的 AWS解决方案 AWS IoT已不再可用。虽然相关GitHub库仍然可以访问，但已于 2023 年 7 月将其 AWS 弃用，并且不再为其提供维护或支持。要实施您自己的解决方案或探索其他支持选项，请访问联系方式 AWS。如果您的账户有 AWS 技术客户经理，请联系他们寻求帮助。

使用即时预调配

手动创建和配置每台设备可能很耗时。 AWS IoT 提供了一种定义模板的方法，以便在设备首次连接时对其进行配置 AWS IoT。有关更多信息，请参阅 J ust-in-time 资源调配。

运行 AWS IoT 设备顾问测试的权限

以下策略模板显示了运行 AWS IoT 设备顾问测试用例所需的最低权限和 IAM 实体。您需要将 *your-device-role-arn* 替换为您根据先决条件创建的设备角色 Amazon Resource Name (ARN)。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "your-device-role-arn",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "iotdeviceadvisor.amazonaws.com"
                }
            }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke*",
                "iam:ListRoles",   // Required to list device roles in the Device
 Advisor console
                "iot:Connect",
                "iot:CreateJob",
                "iot:DeleteJob",
                "iot:DescribeCertificate",
                "iot:DescribeEndpoint",
                "iot:DescribeJobExecution",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:GetPendingJobExecutions",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListCertificates",
```

```
                "iot:ListPrincipalPolicies",
                "iot:ListThingPrincipals",
                "iot:ListThings",
                "iot:Publish",
                "iot:StartNextPendingJobExecution",
                "iot:UpdateJobExecution",
                "iot:UpdateThingShadow",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": "*"
        }
    ]
}
```

针对 Device Advisor 防范跨服务混淆代理

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。在中 AWS，跨服务模仿可能会导致混乱的副手问题。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。可以操纵调用服务以使用其权限对另一个客户的资源进行操作，否则该服务不应有访问权限。为了防止这种情况，我们 AWS 提供了一些工具，帮助您保护所有服务的数据，这些服务委托人已被授予访问您账户中资源的权限。

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 Device Advisor 为其它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` 的值必须是套件定义资源的 ARN。套件定义资源是指您使用 Device Advisor 创建的测试套件。

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:iotdeviceadvisor:*:*account-id*:suitedefinition/*。

以下示例演示如何使用 Device Advisor 中的 aws:SourceArn 和 aws:SourceAccount 全局条件上下文键来防范混淆代理问题。

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "iotdeviceadvisor.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:iotdeviceadvisor:us-
east-1:123456789012:suitedefinition/ygp6rxa3tzvn"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

AWS 培训和认证

参加以下课程，了解有关安全的关键概念： AWS IoT AWS IoT 安全入门。

监控 AWS IoT

监控是维护 AWS 解决方案的可靠性、可用性和性能的重要组成部分。 AWS IoT

我们强烈建议您从 AWS 解决方案的各个部分收集监控数据，以便在出现多点故障时更轻松地进行调试。首先创建一个监控计划来回答以下问题。如果您不确定如何回答这些问题，您仍然可以继续启用日志记录并建立性能基准。

- 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？

下一步是启用日志记录，并通过测量不同时间和不同负载条件下的性能，建立环境中正常 AWS IoT 性能的基准。监控时 AWS IoT，请保留历史监控数据，以便可以将其与当前性能数据进行比较。这将帮助您确定一般的性能模式和性能异常，并设计解决问题的方法。

要确定您的基准绩效 AWS IoT，您应该从一开始就监控这些指标。您可在以后随时监控更多指标。

- PublishIn.Success
- PublishOut.Success
- Subscribe.Success
- Ping.Success
- Connect.Success
- GetThingShadow.Accepted
- UpdateThingShadow.Accepted
- DeleteThingShadow.Accepted
- RulesExecuted

本部分中的主题可帮助您启动对 AWS IoT的日志记录和监控操作。

主题

- 配置 AWS IoT 日志
- 使用 Amazon 监控 AWS IoT 警报和指标 CloudWatch
- AWS IoT 使用 CloudWatch 日志进行监控
- 将设备端日志上传到 Amazon CloudWatch
- 使用记录 AWS IoT API 调用 AWS CloudTrail

配置 AWS IoT 日志

必须先使用 AWS IoT 控制台、CLI 或 API 启用日志记录，然后才能监控和记录 AWS IoT 活动。

您可以为所有 AWS IoT 或仅为特定事物组启用日志记录。您可以使用 AWS IoT 控制台、CLI 或 API 配置 AWS IoT 日志记录；但是，必须使用 CLI 或 API 为特定事物组配置日志记录。

在考虑如何配置 AWS IoT 日志记录时，除非另有说明，否则默认的日志记录配置将决定如何记录 AWS IoT 活动。首先，您可能要获取默认日志级别为 `INFO` 或 `DEBUG` 的详细日志。查看初始日志后，您可以将默认日志级别更改为较低的详细程度级别（如 `WARN` 或 `ERROR`），并对可能需要更多关注的资源设置更详细的资源特定日志级别。日志级别可随时更改。

本主题介绍云端登录。 AWS IoT有关设备端日志记录和监控的信息，请参阅将设备端日志上传到。 CloudWatch

有关日志记录和监控的信息 AWS IoT Greengrass，请参阅中的日志和监控 AWS IoT Greengrass。截至 2023 年 6 月 30 日， AWS IoT Greengrass 核心软件已迁移到 AWS IoT Greengrass Version 2。

配置日志记录角色和策略

在启用登录功能之前 AWS IoT，您必须创建一个 IAM 角色和一个授予代表您监控 AWS IoT 活动的 AWS 权限的策略。您还可以在 AWS IoT 控制台的“日志”部分生成具有所需策略的 IAM 角色。

> **Note**
>
> 在启用 AWS IoT 日志记录之前，请确保您了解 CloudWatch 日志访问权限。有权访问 CloudWatch 日志的用户可以从您的设备中查看调试信息。有关更多信息，请参阅 Amazon CloudWatch 日志的身份验证和访问控制。
>
> 如果您预计 AWS IoT Core 由于负载测试而出现高流量模式，请考虑关闭物联网日志以防止限制。如果检测到高流量，我们的服务可能会禁用您账户中的日志记录。

下面显示了如何为 AWS IoT Core 资源创建日志记录角色和策略。

创建日志记录角色

要创建日志记录角色，请打开 IAM 控制台的角色中心并选择 Create role（创建角色）。

1. 在 Select trusted entity（选择受信任的实体）下，选择 AWS Service（AWS 服务）。然后在 Use case（用例）下，选择 IoT。如果您看不到 IoT，请在 Use cases for other AWS services:（其他 AWS 服务的用例：）下拉菜单中输入并搜索 IoT。选择下一步。
2. 在 Add permissions（添加权限）页面上，您将看到自动附加到服务角色的策略。选择下一步。
3. 在 Name, review, and create（命名、检查和创建）页面上，为此角色输入 Role name（角色名称）和 Role description（角色描述），然后选择 Create role（创建角色）。
4. 在 Role（角色）列表中，找到您刚创建的角色，打开该角色，然后复制 Role ARN（角色 ARN）(*logging-role-arn*) 以供稍后在 AWS IoT（控制台）中配置默认日志记录使用。

日志记录角色策略

以下策略文件提供了允许代表您 AWS IoT 向其提交日志条目的角色策略和信任策略。 CloudWatch 如果您还允许 AWS IoT Core LoRa WAN 提交日志条目，则会看到一份为您创建的策略文档，其中记录了这两项活动。

> **Note**
>
> 在您创建日志记录角色时已为您创建这些文档。这些文档具有变量 *${partition}*、*${region}* 和 *${accountId}*，必须用您的值替换这些变量。

角色策略：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "logs:CreateLogGroup",
    "logs:CreateLogStream",
    "logs:PutLogEvents",
```

```
    "logs:PutMetricFilter",
    "logs:PutRetentionPolicy",
    "iot:GetLoggingOptions",
    "iot:SetLoggingOptions",
    "iot:SetV2LoggingOptions",
    "iot:GetV2LoggingOptions",
    "iot:SetV2LoggingLevel",
    "iot:ListV2LoggingLevels",
    "iot:DeleteV2LoggingLevel"
   ],
   "Resource": [
    "arn:${partition}:logs:${region}:${accountId}:log-group:AWSIotLogsV2:*"
   ]
  }
 ]
}
```

仅记录 AWS IoT Core 活动的信任策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

在 AWS IoT（控制台）中配置默认日志记录

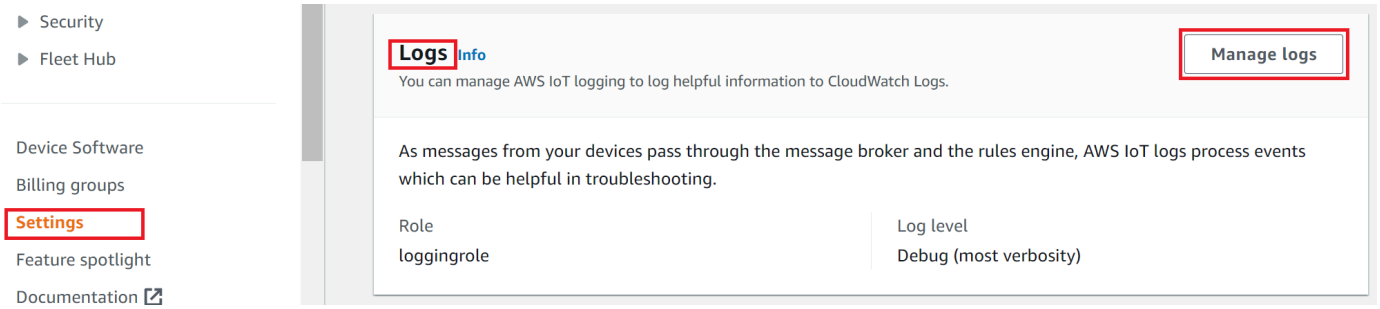
本节介绍如何使用 AWS IoT 控制台为所有配置日志记录 AWS IoT。要仅为特定事物组配置日志记录，您必须使用 CLI 或 API。有关为特定事物组配置日志记录的信息，请参阅在 AWS IoT 中配置资源特定的日志记录 (CLI)。

使用 AWS IoT 控制台为所有配置默认日志记录 AWS IoT

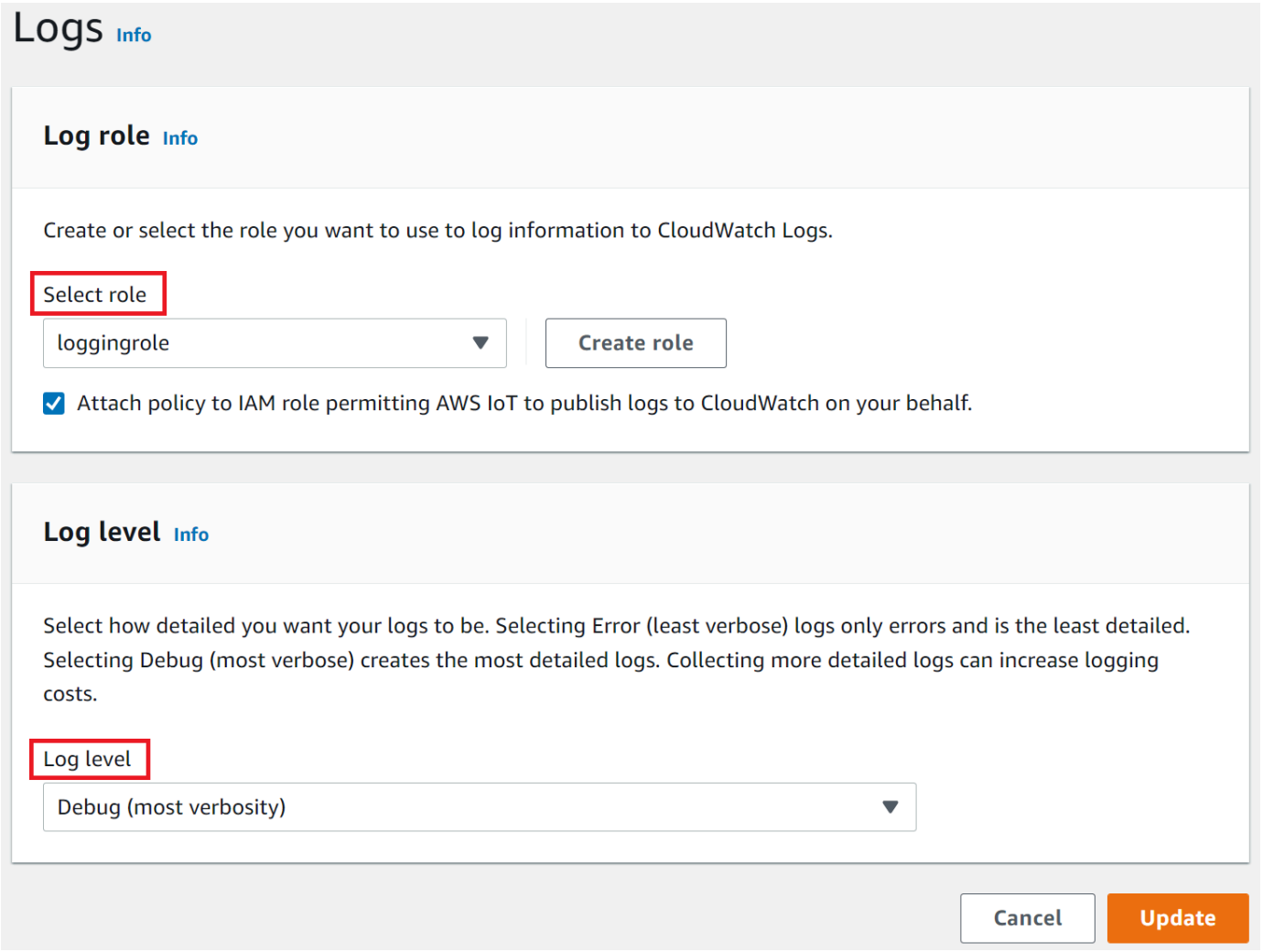
1. 登录 AWS IoT 控制台。有关更多信息，请参阅 打开控制 AWS IoT 台。

2. 在左侧导航窗格中，选择 设置。在 Settings（设置）页面的 Logs（日志）部分中，选择 Manage logs（管理日志）。

   Logs（日志）页面显示 AWS IoT使用的日志记录角色和详细程度级别。



3. 在 Logs（日志）页面上，选择 Select role（选择角色）以指定您在 创建日志记录角色 中创建的角色，或选择 Create Role（创建角色）以创建用于日志记录的新角色。



4. 选择描述要在日志中显示的日志条目的详细级别的 CloudWatch 日志级别。

5. 单击更新以保存您的更改。

启用日志记录后，请访问 在 CloudWatch 控制台中查看 AWS IoT 日志 以了解有关查看日志条目的更多信息。

配置默认登录 AWS IoT (CLI)

本节介绍如何使用 CLI 为 AWS IoT 配置全局日志。

> **Note**
>
> 您需要提供要使用角色的 Amazon Resource Name (ARN)。如果需要创建用于日志记录的角色，请参阅 创建日志记录角色，然后继续操作。
> 用于调用 API 的委托人对于您的日志记录角色必须具有传递角色权限。

您也可以使用 API 中与此处显示的 CLI 命令相对应的方法，对 AWS API 执行此过程。

使用 CLI 配置默认日志记录 AWS IoT

1. 使用 set-v2-logging-options 命令为您的账户设置日志记录选项。

```
aws iot set-v2-logging-options \
    --role-arn logging-role-arn \
    --default-log-level log-level
```

其中：

--role-arn

授予在日志中写入日志 AWS IoT 权限的角色 ARN。 CloudWatch

--default-log-level

要使用的日志级别。有效值为：ERROR、WARN、INFO、DEBUG 或 DISABLED。

--no-disable-all-logs

一个可选参数，用于启用所有 AWS IoT 日志记录。使用此参数可在当前禁用日志记录时启用日志记录。

--disable-all-logs

一个可选参数，用于禁用所有 AWS IoT 日志记录。使用此参数可在当前启用日志记录时禁用日志记录。

2. 使用 get-v2-logging-options 命令获取当前日志记录选项。

```
aws iot get-v2-logging-options
```

启用日志记录后，请访问 在 CloudWatch 控制台中查看 AWS IoT 日志 以了解有关查看日志条目的更多信息。

> **Note**
>
> AWS IoT 继续支持旧命令（set-logging-options和get-logging-options），以便在您的账户上设置和获取全局登录。请注意，使用这些命令时，生成的日志将包含纯文本而不是 JSON 负载，并且日志记录延迟通常会更高。将不会再对这些较早命令的实施进行更多改进。建议您使用“v2”版本来配置日志记录选项，如果可能，还请更改使用早期版本的传统应用程序。

在 AWS IoT 中配置资源特定的日志记录 (CLI)

本节介绍如何使用 CLI 为 AWS IoT 配置特定资源的日志记录。可通过资源特定的日志记录为特定事物组指定日志记录级别。

事物组可包含其它事物组以创建分层关系。此流程介绍如何配置单个事物组的日志记录。您可以将此流程应用于层次结构中的父事物组，以配置层次结构中所有事物组的日志记录。您也可以将此流程应用于子事情组，以覆盖其父级的日志记录配置。

除了事物组之外，您还可以录入目标，例如设备的客户端 ID、源 IP 和主体 ID。

> **Note**
>
> 您需要提供要使用角色的 Amazon Resource Name (ARN)。如果需要创建用于日志记录的角色，请参阅 创建日志记录角色，然后继续操作。
> 用于调用 API 的委托人对于您的日志记录角色必须具有传递角色权限。

您也可以使用 API 中与此处显示的 CLI 命令相对应的方法，对 AWS API 执行此过程。

使用 CLI 配置特定于资源的日志记录 AWS IoT

1. 使用 set-v2-logging-options 命令为您的账户设置日志记录选项。

```
aws iot set-v2-logging-options \
    --role-arn logging-role-arn \
    --default-log-level log-level
```

其中：

--role-arn

授予在日志中写入日志 AWS IoT 权限的角色 ARN。 CloudWatch

--default-log-level

要使用的日志级别。有效值为：ERROR、WARN、INFO、DEBUG 或 DISABLED。

--no-disable-all-logs

一个可选参数，用于启用所有 AWS IoT 日志记录。使用此参数可在当前禁用日志记录时启用日志记录。

--disable-all-logs

一个可选参数，用于禁用所有 AWS IoT 日志记录。使用此参数可在当前启用日志记录时禁用日志记录。

2. 使用 set-v2-logging-level 命令为事物组配置资源特定的日志记录。

```
aws iot set-v2-logging-level \
              --log-target targetType=THING_GROUP,targetName=thing_group_name \
              --log-level log_level
```

--log-target

您要配置日志记录的资源的类型和名称。target_type 值必须是下列项之一：THING_GROUP | CLIENT_ID | SOURCE_IP | PRINCIPAL_ID。log-target 参数值可以是文本（如前面的命令示例所示），也可以是 JSON 字符串，如以下示例所示。

```
aws iot set-v2-logging-level \
              --log-target '{"targetType": "THING_GROUP","targetName":
 "thing_group_name"}' \
```

```
                --log-level log_level
```

--log-level

为指定资源生成日志时使用的日志记录级别。有效值为：DEBUG、INFO、ERROR、WARN 和 DISABLED

```
aws iot set-v2-logging-level \
              --log-target targetType=CLIENT_ID,targetName=ClientId1 \
              --log-level DEBUG
```

3. 使用 list-v2-logging-levels 命令列出当前配置的日志记录级别。

```
aws iot list-v2-logging-levels
```

4. 使用 delete-v2-logging-level 命令删除资源特定的日志记录级别，比如下面的例子。

```
aws iot delete-v2-logging-level \
              --target-type "THING_GROUP" \
              --target-name "thing_group_name"
```

```
aws iot delete-v2-logging-level \
              --target-type=CLIENT_ID
              --target-name=ClientId1
```

--targetType

target_type 值必须是下列项之一：THING_GROUP | CLIENT_ID | SOURCE_IP | PRINCIPAL_ID。

--targetName

要删除日志记录级别的事物组的名称。

启用日志记录后，请访问 在 CloudWatch 控制台中查看 AWS IoT 日志 以了解有关查看日志条目的更多信息。

日志级别

这些日志级别确定记录的事件，并应用于默认日志级别和资源特定的日志级别。

错误

导致操作失败的任何错误。

日志仅包含 ERROR 信息。

警告

可能导致系统中出现不一致问题，但不会导致操作失败的所有情况。

日志包括 ERROR 和 WARN 信息。

信息

有关事物流的高级别信息。

日志包括 INFO、ERROR 和 WARN 信息。

调试

可能有助于调试问题的信息。

日志包括 DEBUG、INFO、ERROR 和 WARN 信息。

DISABLED

所有日志记录均处于禁用状态。

使用 Amazon 监控 AWS IoT 警报和指标 CloudWatch

您可以使用 AWS IoT 进行监控 CloudWatch，它会收集原始数据并将其处理 AWS IoT 为可读的、近乎实时的指标。这些统计数据会保存两周，从而使您能够访问历史信息，并能够更好地了解您的网络应用程序或服务的执行情况。默认情况下， AWS IoT 指标数据每隔一分钟自动发送到 CloudWatch 。有关更多信息，请参阅什么是亚马逊 CloudWatch、亚马逊 CloudWatch 事件和亚马逊 CloudWatch 日志？在《亚马逊 CloudWatch 用户指南》中。

使用 AWS IoT 指标

报告的指标 AWS IoT 提供了您可以用不同方式进行分析的信息。以下使用案例基于您每天将十个事物连接一次 Internet 的场景。每天：

- 十件事大致 AWS IoT 在同一时间相连。
- 每个事物都会订阅主题筛选条件，接着在等待一个小时后断开连接。在此期间，事物之间相互通信并探索更多与环境状态有关的信息。

- 每个事物都会基于其新发现的数据，使用 UpdateThingShadow 发布一些看法，
- 每件事都与之断开连接。 AWS IoT

为了帮助您入门，这些主题探讨了您可能会遇到的一些问题。

- 我如何在事物每天没有成功建立连接时得到通知？
- 我如何在事物每天没有发布数据时得到通知？
- 我如何在事物的影子更新每天被拒绝时得到通知？
- 如何为 Jobs 创建 CloudWatch 警报？

有关 CloudWatch 警报和指标的更多信息

- 创建要监控的 CloudWatch 警报 AWS IoT
- AWS IoT 指标和维度

创建要监控的 CloudWatch 警报 AWS IoT

您可以创建一个 CloudWatch 警报，在警报状态发生变化时发送 Amazon SNS 消息。警报会在您规定的时间范围内监控某一项指标。当指标的值在多个时间段内超过给定阈值时，将执行一项或多项操作。操作是一个发送到 Amazon SNS 主题或 Auto Scaling 策略的通知。警报仅在持续状态变化时触发操作。 CloudWatch 警报不会仅仅因为它们处于特定状态而触发操作；该状态必须已更改并维持了指定的时间段。

以下主题描述了使用 CloudWatch警报的一些示例。

- 我如何知道事物每天是否成功建立连接？
- 我如何在事物每天没有发布数据时得到通知？
- 我如何在事物的影子更新每天被拒绝时得到通知？
- 如何为作业创建 CloudWatch 警报？

您可以查看 CloudWatch 警报可以监控的所有指标AWS IoT 指标和维度。

我如何知道事物每天是否成功建立连接？

1. 创建名为 `things-not-connecting-successfully` 的 Amazon SNS 主题，并记录其 Amazon Resource Name (ARN)。此流程将您主题的 ARN 称为 *`sns-topic-arn`*。

有关如何创建 Amazon SNS 通知的更多信息，请参阅 Amazon SNS 入门。

2. 创建告警。

```
aws cloudwatch put-metric-alarm \
    --alarm-name ConnectSuccessAlarm \
    --alarm-description "Alarm when my Things don't connect successfully" \
    --namespace AWS/IoT \
    --metric-name Connect.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. 测试告警。

```
aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
 "initializing" --state-value ALARM
```

4. 确认警报显示在您的CloudWatch 控制台中。

我如何在事物每天没有发布数据时得到通知？

1. 创建名为 `things-not-publishing-data` 的 Amazon SNS 主题，并记录其 Amazon Resource Name (ARN)。此流程将您主题的 ARN 称为 *sns-topic-arn*。

   有关如何创建 Amazon SNS 通知的更多信息，请参阅 Amazon SNS 入门。

2. 创建告警。

```
aws cloudwatch put-metric-alarm \
    --alarm-name PublishInSuccessAlarm\
    --alarm-description "Alarm when my Things don't publish their data \
    --namespace AWS/IoT \
    --metric-name PublishIn.Success \
```

```
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. 测试告警。

```
aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason
 "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason
 "initializing" --state-value ALARM
```

4. 确认警报显示在您的CloudWatch 控制台中。

我如何在事物的影子更新每天被拒绝时得到通知？

1. 创建名为 `things-shadow-updates-rejected` 的 Amazon SNS 主题，并记录其 Amazon Resource Name (ARN)。此流程将您主题的 ARN 称为 *sns-topic-arn*。

   有关如何创建 Amazon SNS 通知的更多信息，请参阅 Amazon SNS 入门。

2. 创建告警。

```
aws cloudwatch put-metric-alarm \
    --alarm-name UpdateThingShadowSuccessAlarm \
    --alarm-description "Alarm when my Things Shadow updates are getting rejected"
 \
    --namespace AWS/IoT \
    --metric-name UpdateThingShadow.Success \
    --dimensions Name=Protocol,Value=MQTT \
    --statistic Sum \
    --threshold 10 \
    --comparison-operator LessThanThreshold \
    --period 86400 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions sns-topic-arn
```

3. 测试告警。

```
aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-
reason "initializing" --state-value OK
```

```
aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-
reason "initializing" --state-value ALARM
```

4. 确认警报显示在您的CloudWatch 控制台中。

如何为作业创建 CloudWatch 警报？

作业服务提供 CloudWatch 指标供您监控作业。您可以创建 CloudWatch 警报来监控任何警报任务指标。

以下命令创建 CloudWatch 警报以监控 Job s *ampleotaJo* b 的失败任务执行总数，并在超过 20 个任务执行失败时通知您。警报通过每 300 秒检查报告值来监控 Jobs 指标 `FailedJobExecutionTotalCount`。它在单个报告值大于 20 时激活，这意味着自任务启动以来失败的任务执行数超过了 20。当告警关闭时，它会向提供的 Amazon SNS 主题发送通知。

```
aws cloudwatch put-metric-alarm \
    --alarm-name TotalFailedJobExecution-SampleOTAJob \
    --alarm-description "Alarm when total number of failed job execution exceeds the
 threshold for SampleOTAJob" \
    --namespace AWS/IoT \
    --metric-name FailedJobExecutionTotalCount \
    --dimensions Name=JobId,Value=SampleOTAJob \
    --statistic Sum \
    --threshold 20 \
    --comparison-operator GreaterThanThreshold \
    --period 300 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:<AWS_REGION>:<AWS_ACCOUNT_ID>:SampleOTAJob-has-too-
many-failed-job-ececutions
```

以下命令创建 CloudWatch 警报，用于监控 Job *SampleotaJob ###########*执行次数。然后，在该时段内有超过 5 个任务执行失败时，它会通知您。警报通过每 3600 秒检查报告值来监控 Jobs 指标 `FailedJobExecutionCount`。它在单个报告值大于 5 时激活，这意味着过去 1 小时内失败的任务执行数超过了 5。当告警关闭时，它会向提供的 Amazon SNS 主题发送通知。

```
aws cloudwatch put-metric-alarm \
    --alarm-name FailedJobExecution-SampleOTAJob \
    --alarm-description "Alarm when number of failed job execution per hour exceeds the
 threshold for SampleOTAJob" \
    --namespace AWS/IoT \
    --metric-name FailedJobExecutionCount \
    --dimensions Name=JobId,Value=SampleOTAJob \
    --statistic Sum \
    --threshold 5 \
    --comparison-operator GreaterThanThreshold \
    --period 3600 \
    --unit Count \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:<AWS_REGION>:<AWS_ACCOUNT_ID>:SampleOTAJob-has-too-
many-failed-job-ececutions-per-hour
```

AWS IoT 指标和维度

当您与交互时 AWS IoT，该服务 CloudWatch 每分钟都会向其发送以下指标和维度。您可以使用以下流程查看 的指标 AWS IoT

查看指标（CloudWatch 控制台）

指标的分组首先依据服务命名空间，然后依据每个命名空间内的各种维度组合。

1. 打开CloudWatch 控制台。
2. 在导航窗格中，选择 Metrics（指标）然后选择 All metrics（所有指标）。
3. 在 “浏览” 选项卡中，搜索 AWS IoT 以查看指标列表。

查看指标 (CLI)

- 在命令提示符处输入下面的命令：

```
aws cloudwatch list-metrics --namespace "AWS/IoT"
```

CloudWatch 显示以下几组指标 AWS IoT：

- AWS IoT 指标
- AWS IoT Core 凭证提供者指标

AWS IoT 指标

指标	描述
AddThingToDynamicThingGroupsFailed	与将事物添加到动态事物组相关联的失败事件数。DynamicThingGroupName 维度包含添加事物时失败的动态组的名称。
NumLogBatchesFailedToPublishThrottled	因限制错误而无法发布的日志事件的单个批次。
NumLogEventsFailedToPublishThrottled	批处理中的因限制错误而无法发布的日志事件的数量。

AWS IoT Core 凭证提供者指标

指标	描述
CredentialExchangeSuccess	向 AWS IoT Core 凭证提供程序成功提交 AssumeRoleWithCertificate 请求的数量。

身份验证指标

> **Note**
>
> 身份验证指标显示在 CloudWatch 控制台的 “协议指标” 下方。

指标	描述
Connect.AuthNError	由于身份验证失败而 AWS IoT Core 被拒绝的连接尝试次数。此指标仅考虑发送与您的 AWS 账户端点匹配的服务器名称指示 (SNI) 字符串的连接。该指标包括来自外部来源（例如互联网扫描工具或探测活动）的连接尝试次数。该Protocol维度包含用于发送连接尝试的协议。

服务器证书 OCSP 装订指标

指标	描述
检索 StapleData OCSP .Success	OCSP 响应已成功接收并处理。此响应将包含在已配置域的 TLS 握手期间。该DomainConfigurationName 维度包含已启用服务器证书 OCSP 装订的已配置域的名称。

规则指标

指标	描述
ParseError	在规则所侦听的主题上发布的消息中出现的 JSON 分析错误的数量。RuleName 维度包含规则的名称。
RuleMessageThrottled	由于恶意行为或由于消息数超过了规则引擎的限制，规则引擎限制的消息数。RuleName 维度包含要触发的规则的名称。
RuleNotFound	找不到要触发的规则。RuleName 维度包含规则的名称。
RulesExecuted	已执行的 AWS IoT 规则数。
TopicMatch	已在规则所侦听的主题上发布的传入消息的数量。RuleName 维度包含规则的名称。

规则操作指标

指标	描述
Failure	失败的规则操作调用的数量。RuleName 维度包含指定操作的规则的名称。ActionType 维度包含已调用的操作的类型。
Success	成功的规则操作调用的数量。RuleName 维度包含指定操作的规则的名称。ActionType 维度包含已调用的操作的类型。
ErrorActionFailure	失败的错误操作数量。RuleName 维度包含指定操作的规则的名称。ActionType 维度包含已调用的操作的类型。

指标	描述
ErrorActionSuccess	成功的错误操作数量。RuleName 维度包含指定操作的规则的名称。ActionType 维度包含已调用的操作的类型。

HTTP 操作特定指标

指标	描述
HttpCode_Other	如果来自下游 Web 服务/应用程序的响应的状态代码不是 2xx、4xx 或 5xx，则生成此指标。
HttpCode_4XX	如果来自下游 Web 服务/应用程序的响应的状态代码介于 400 和 499 之间，则生成此指标。
HttpCode_5XX	如果来自下游 Web 服务/应用程序的响应的状态代码介于 500 和 599 之间，则生成此指标。
HttpInvalidUrl	如果在替换模板后的终端节点 URL 不以 https:// 开头，则生成此指标。
HttpRequestTimeout	如果下游 Web 服务/应用程序未在请求超时限制内返回响应，则生成此指标。有关更多信息，请参阅 Service Quotas。
HttpUnknownHost	如果 URL 有效，但服务不存在或无法访问，则生成此指标。

消息代理指标

> **Note**
>
> 消息代理指标显示在 CloudWatch 控制台的协议指标下。

指标	描述
Connect.AuthError	无法由消息代理授权的连接请求的数量。Protocol 维度包含用于发送 CONNECT 消息的协议。
Connect.ClientError	因 MQTT 消息未能满足AWS IoT 配额中定义的要求而被拒绝的连接请求的数量。Protocol 维度包含用于发送 CONNECT 消息的协议。
Connect.ClientIDThrottle	因该客户端超出指定客户端 ID 所允许的连接请求速率而受限的连接请求的数量。Protocol 维度包含用于发送 CONNECT 消息的协议。
Connect.ServerError	因出现内部错误而导致失败的连接请求的数量。Protocol 维度包含用于发送 CONNECT 消息的协议。
Connect.Success	与消息代理的成功连接的数量。Protocol 维度包含用于发送 CONNECT 消息的协议。
Connect.Throttle	因账户超出允许的连接请求速率而受限的连接请求的数量。Protocol 维度包含用于发送 CONNECT 消息的协议。
Ping.Success	消息代理收到的 ping 消息的数量。Protocol 维度包含用于发送 ping 消息的协议。
PublishIn.AuthError	消息代理无法授权的发布请求的数量。Protocol 维度包含用于发布消息的协议。HTTP 发布不支持此指标。
PublishIn.ClientError	因消息未能满足AWS IoT 配额中定义的要求而被消息代理拒绝的发布请求的数量。Protocol 维度包含用于发布消息的协议。HTTP 发布不支持此指标。
PublishIn.ServerError	因出现内部错误而导致消息代理无法处理的发布请求的数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。HTTP 发布不支持此指标。

指标	描述
PublishIn.Success	消息代理已成功处理的发布请求的数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。
PublishIn.Throttle	因客户端超出允许的入站消息速率而受限制的发布请求的数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。HTTP 发布不支持此指标。
PublishOut.AuthError	AWS IoT无法授权的由消息代理发出的发布请求的数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。
PublishOut.ClientError	因消息未能满足AWS IoT 配额中定义的要求而被拒绝的由消息代理发出的发布请求的数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。
PublishOut.Success	消息代理已成功发出的发布请求的数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。
PublishOut.Throttle	因客户端超出允许的入站消息速率而节流的发布请求数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。
PublishRetained.AuthError	消息代理无法授权的 RETAIN 标签集发布请求的数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。
PublishRetained.ServerError	因出现内部错误而导致消息代理无法处理的保留发布请求的数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。
PublishRetained.Success	消息代理已成功处理的 RETAIN 标签集发布请求的数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。
PublishRetained.Throttle	因客户端超出允许的入站消息速率而受限制的 RETAIN 标签集发布请求的数量。Protocol 维度包含用于发送 PUBLISH 消息的协议。

指标	描述
Queued.Success	消息代理为与持久性会话断开连接的客户端成功处理的存储消息的数量。QoS 为 1 的消息是在具有持久性会话的客户端断开连接时存储的。
Queued.Throttle	在具有持久性会话的客户端断开连接时无法存储和受限制的消息的数量。当客户端超过每个账户每秒的排队消息数限制时，就会发生这种情况。QoS 为 1 的消息是在具有持久性会话的客户端断开连接时存储的。
Queued.ServerError	由于内部错误而未为持久性会话存储的消息数量。当具有持久性会话的客户端断开连接时，将存储服务质量（QoS）为 1 的消息。
Subscribe.AuthError	无法授权的由客户端发出的订阅请求的数量。Protocol 维度包含用于发送 SUBSCRIBE 消息的协议。
Subscribe.ClientError	因 SUBSCRIBE 消息未能满足 AWS IoT 配额 中定义的要求而被拒绝的订阅请求的数量。Protocol 维度包含用于发送 SUBSCRIBE 消息的协议。
Subscribe.ServerError	因出现内部错误而被拒绝的订阅请求的数量。Protocol 维度包含用于发送 SUBSCRIBE 消息的协议。
Subscribe.Success	消息代理已成功处理的订阅请求的数量。Protocol 维度包含用于发送 SUBSCRIBE 消息的协议。
Subscribe.Throttle	由于超出您的允许的订阅请求速率限制而受到限制的订阅请求数量。 AWS 账户这些限制包括AWS IoT Core 消息代理和协议限制和配额中描述的每个账户的每秒订阅数、每个账户的订阅量以及每个连接的订阅量。Protocol 维度包含用于发送 SUBSCRIBE 消息的协议。

指标	描述
Throttle.Exceeded	此指标将显示在 MQTT 客户端根据每个连接级别限制的每秒数据包数进行限制 CloudWatch 时。此指标不适用于 HTTP 连接。
Unsubscribe.ClientError	因 UNSUBSCRIBE 消息未能满足 AWS IoT 配额 中定义的要求而被拒绝的取消订阅请求的数量。Protocol 维度包含用于发送 UNSUBSCRIBE 消息的协议。
Unsubscribe.ServerError	因出现内部错误而被拒绝的取消订阅请求的数量。Protocol 维度包含用于发送 UNSUBSCRIBE 消息的协议。
Unsubscribe.Success	消息代理已成功处理的取消订阅请求的数量。Protocol 维度包含用于发送 UNSUBSCRIBE 消息的协议。
Unsubscribe.Throttle	因客户端超出允许的取消订阅请求速率而被拒绝的取消订阅请求的数量。Protocol 维度包含用于发送 UNSUBSCRIBE 消息的协议。

设备影子指标

> **Note**
>
> 设备影子指标显示在 CloudWatch 控制台的 “协议指标” 下方。

指标	描述
DeleteThingShadow.Accepted	已成功处理的 DeleteThingShadow 请求的数量。Protocol 维度包含用于发出请求的协议。
GetThingShadow.Accepted	已成功处理的 GetThingShadow 请求的数量。Protocol 维度包含用于发出请求的协议。

指标	描述
ListThingShadow.Accepted	已成功处理的 ListThingShadow 请求的数量。Protocol 维度包含用于发出请求的协议。
UpdateThingShadow.Accepted	已成功处理的 UpdateThingShadow 请求的数量。Protocol 维度包含用于发出请求的协议。

任务指标

指标	描述
CanceledJobExecutionCount	在由确定的时间段CANCELED内状态更改为的任务执行次数 CloudWatch。（有关 CloudWatch 指标的更多信息，请参阅 Amazon CloudWatch 指标。）JobId 维度包含任务的 ID。
CanceledJobExecutionTotalCount	给定任务的状态为 CANCELED 的任务执行总数。JobId 维度包含任务的 ID。
ClientErrorCount	执行任务时生成的客户端错误数。JobId 维度包含任务的 ID。
FailedJobExecutionCount	在由确定的时间段FAILED内状态更改为的任务执行次数 CloudWatch。（有关 CloudWatch 指标的更多信息，请参阅 Amazon CloudWatch 指标。）JobId 维度包含任务的 ID。
FailedJobExecutionTotalCount	给定任务的状态为 FAILED 的任务执行总数。JobId 维度包含任务的 ID。
InProgressJobExecutionCount	在由确定的时间段IN_PROGRESS 内状态更改为的任务执行次数 CloudWatch。（有关 CloudWatch 指标的更多信息，请参阅 Amazon CloudWatch 指标。）JobId 维度包含任务的 ID。

指标	描述
InProgressJobExecutionTotalCount	给定任务的状态为 IN_PROGRESS 的任务执行总数。JobId 维度包含任务的 ID。
RejectedJobExecutionTotalCount	给定任务的状态为 REJECTED 的任务执行总数。JobId 维度包含任务的 ID。
RemovedJobExecutionTotalCount	给定任务的状态为 REMOVED 的任务执行总数。JobId 维度包含任务的 ID。
QueuedJobExecutionCount	在由确定的时间段QUEUED内状态更改为的任务执行次数 CloudWatch。（有关 CloudWatch 指标的更多信息，请参阅 Amazon CloudWatch 指标。）JobId 维度包含任务的 ID。
QueuedJobExecutionTotalCount	给定任务的状态为 QUEUED 的任务执行总数。JobId 维度包含任务的 ID。
RejectedJobExecutionCount	在由确定的时间段REJECTED内状态更改为的任务执行次数 CloudWatch。（有关 CloudWatch 指标的更多信息，请参阅 Amazon CloudWatch 指标。）JobId 维度包含任务的 ID。
RemovedJobExecutionCount	在由确定的时间段REMOVED内状态更改为的任务执行次数 CloudWatch。（有关 CloudWatch 指标的更多信息，请参阅 Amazon CloudWatch 指标。）JobId 维度包含任务的 ID。
ServerErrorCount	执行任务时生成的服务器错误数。JobId 维度包含任务的 ID。
SuccededJobExecutionCount	在由确定的时间段SUCCESS内状态更改为的任务执行次数 CloudWatch。（有关 CloudWatch 指标的更多信息，请参阅 Amazon CloudWatch 指标。）JobId 维度包含任务的 ID。
SuccededJobExecutionTotalCount	给定任务的状态为 SUCCESS 的任务执行总数。JobId 维度包含任务的 ID。

Device Defender audit 指标

指标	描述
NonCompliantResources	在检查中发现的不合规资源数。系统报告执行的每次审核中，对于每次检查发现的不合规资源数。
ResourcesEvaluated	执行合规性评估的资源数。系统报告执行的每次审核中，对于每次检查所评估的资源数。
MisconfiguredDeviceDefenderNotification	当你的 SNS 配置配置错误时会通知你。 AWS IoT Device Defender 尺寸

Device Defender detect 指标

指标	描述
NumOfMetricsExported	云端、设备端或自定义指标导出的指标数量。系统会报告该账户就某项特定指标导出的指标数量。此指标仅对使用指标导出功能的客户提供。
NumOfMetricsSkipped	云端、设备端或自定义指标跳过的指标数量。由于为发布到 mqtt 主题而向 Device Defender Detect 提供的权限不足，系统会报告该账户就某项特定指标跳过的指标数量。此指标仅对使用指标导出功能的客户提供。
NumOfMetricsExceedingSizeLimit	由于大小超过 MQTT 消息大小限制，云端、设备端或自定义指标跳过导出的指标数量。系统会报告该账户因大小超过 MQTT 消息大小限制而跳过导出的指标数量。此指标仅对使用指标导出功能的客户提供。
Violations	自上次执行评估以来，所发现的安全配置文件行为的新违规数。系统针对特定安全配置文件，以及针对某

指标	描述
	个特定安全配置文件的特定行为，报告账户的新违规数。
ViolationsCleared	自上次执行评估以来，已解决的安全配置文件行为的违规数。系统针对特定安全配置文件，以及针对某个特定安全配置文件的特定行为，报告账户的已解决违规数。
ViolationsInvalidated	自上次执行评估以来，其信息不再可用（由于报告设备已停止报告，或者由于某个原因不再监控）的安全配置文件行为的违规数。系统针对特定安全配置文件，以及针对某个特定安全配置文件的特定行为，报告整个账户的已失效违规数。
MisconfiguredDeviceDefenderNotification	当你的 SNS 配置配置错误时会通知你。 AWS IoT Device Defender 尺寸

设备预配置指标

AWS IoT 舰队配置指标

指标	描述
ApproximateNumberOfThingsRegistered	Fleet Provisioning 注册的事物的计数。 虽然计数通常是准确的，但 AWS IoT Core 的分布式架构使得很难让已注册事物保持精确计数。 此指标将使用的统计数据是： • Max（最大值），用以报告已注册事物的总数。有关在 CloudWatch聚合窗口期间注册的事物的计数，请参阅RegisterThingFailed 指标。 尺寸：同ClaimCertificate上

指标	描述
CreateKeysAndCertificateFailed	调用 CreateKeysAndCertificate MQTT API 时故障的次数。 在“成功”（值 = 0）和“故障”（值 = 1）情况下均发出该指标。此指标可用于跟踪在 CloudWatch支持的聚合时段（例如 5 分钟或 1 小时）内创建和注册的证书数量。 此指标的可用统计数据包括： • Sum（总计），用以报告失败的调用数。 • SampleCount报告成功和失败的呼叫总数。
CreateCertificateFromCsrFailed	调用 CreateCertificateFromCsr MQTT API 时故障的次数。 在“成功”（值 = 0）和“故障”（值 = 1）情况下均发出该指标。此指标可用于跟踪在 CloudWatch支持的聚合时段（例如 5 分钟或 1 小时）内注册的事物的数量。 此指标的可用统计数据包括： • Sum（总计），用以报告失败的调用数。 • SampleCount报告成功和失败的呼叫总数。

指标	描述
RegisterThingFailed	调用 RegisterThing MQTT API 时故障的次数。 在“成功”（值 = 0）和“故障”（值 = 1）情况下均发出该指标。此指标可用于跟踪在 CloudWatch支持的聚合时段（例如 5 分钟或 1 小时）内注册的事物的数量。有关注册的事物总数，请参阅 ApproximateNumberOfThingsRegistered 指标。 此指标的可用统计数据包括： • Sum（总计），用以报告失败的调用数。 • SampleCount报告成功和失败的呼叫总数。 维度：TemplateName

J ust-in-time 配置指标

指标	描述
ProvisionThing.ClientError	由于客户端错误无法预置设备的次数。例如，模板中指定的策略不存在。
ProvisionThing.ServerError	由于服务器错误而无法预置设备的次数。客户可以在等待后重新尝试预置设备，如果问题仍然存在，则可以尝试联系 AWS IoT 。
ProvisionThing.Success	成功预置设备的次数。

LoRa广域网指标

下表显示了 LoRa WAN AWS IoT Core 的指标。有关更多信息，AWS IoT Core 请参阅 LoRa WAN 指标。

AWS IoT Core 了解 LoRa广域网指标

指标	描述
活动设备/网关	您账户中活跃的 LoRa WAN 设备和网关的数量。
上行链路消息数	在指定时间段内为您的 AWS 账户所有活动网关和设备发送的上行链路消息的数量。上行链路消息是从您的设备发送到 LoRa WAN AWS IoT Core 的消息。
下行链路消息数	在指定时间段内为您的 AWS 账户所有活动网关和设备发送的下行链路消息的数量。下行链路消息是从 LoRa WAN 发送到 AWS IoT Core 您的设备的消息。
消息丢失率	添加设备并连接到 AWS IoT Core LoRa WAN 后，您的设备可以启动上行链路消息，开始与云交换消息。然后，您可以使用此指标来跟踪上行链路消息的丢失率。
加入指标	添加设备和网关后，您需要执行加入程序，以便您的设备可以发送上行链路数据并与 AWS IoT Core LoRa WAN 通信。您可以使用此指标来获取有关您中所有活跃设备的加入指标的信息 AWS 账户。
平均接收信号强度指示器 (RSSI)	您可以使用此指标来监控指定时间段内的平均 RSSI（接收信号强度指标）。RSSI 是一种衡量标准，用于指示信号是否足够强以实现良好的无线连接。此值为负值，必须接近零才能实现牢固连接。
平均信噪比 (SNR)	您可以使用此指标来监控指定时间段内的平均 SNR（S ignal-to-noise 比）。信噪比是一种衡量标准，用于指示接收的信号与噪声水平相比是否足够强，从而实现良好的无线连接。SNR 值为正且必须大于零才能表示信号功率大于噪声功率。
网关可用性	您可以使用此指标来获取有关该网关在指定时间段内的可用性的信息。此指标显示该网关在指定时间段内的 websocket 连接时间。

J ust-in-time 配置指标

指标	描述
ProvisionThing.ClientError	由于客户端错误无法预置设备的次数。例如，模板中指定的策略不存在。
ProvisionThing.ServerError	由于服务器错误而无法预置设备的次数。客户可以在等待后重新尝试预置设备，如果问题仍然存在，则可以尝试联系 AWS IoT 。
ProvisionThing.Success	成功预置设备的次数。

机群索引指标

AWS IoT 舰队索引指标

指标	描述
NamedShadowCountForDynamicGroupQueryLimitExceeded	对于不是动态事物组中特定于数据来源的查询术语，每件事物最多处理 25 个命名影子。由于事件而违反该限制时，将发出 NamedShadowCountForDynamicGroupQueryLimitExceeded 事件类型。

指标的维度

指标使用命名空间并为以下维度提供指标

维度	描述
ActionType	触发请求的规则所指定的操作类型。
BehaviorName	正受监控的 Device Defender Detect 安全配置文件行为的名称。
ClaimCertificateId	用于预置设备的申请的 certificateId 。

维度	描述
CheckName	正在监控其结果的 Device Defender Audit 检查的名称。
JobId	正在监控其进度或消息连接成功/失败的任务的 ID。
Protocol	用于提出请求的协议。有效值为：MQTT 或 HTTP
RuleName	由请求触发的规则的名称。
ScheduledAuditName	正在监控其检查结果的 Device Defender 计划审核的名称。如果报告的结果适用于按需执行的审计，此项具有值 OnDemand。
SecurityProfileName	正在监控其行为的 Device Defender Detect 安全配置文件的名称。
TemplateName	预配置模板的名称。
SourceArn	指用于检测的安全配置文件或用于审计的账户 arn。
RoleArn	指设备防御者试图扮演的角色。
TopicArn	请参阅 Device Defender 尝试发布到的 SNS 主题。

维度	描述
`Error`	简要描述在尝试发布到 SNS 主题时收到的错误。可能的值有： • “已KeyNot找到 KMS”：表示该主题的 KMS 密钥不存在。 • “InvalidTopic名称”：表示 SNS 主题无效。 • AccessDenied“KMS”：表示该角色无权访问该主题的 KMS 密钥。 • “AuthorizationError“: 表示所提供的角色未授权设备防御者向 SNS 主题发布内容。 • “已TopicNot找到 SNS”：表示所提供的 SNS 主题不存在。 • “FailureToAssumeRole“: 表示所提供的角色未授权设备防御者担任该角色。 • “CrossRegionsnStopic”：表示 SNS 主题存在于不同的区域。

AWS IoT 使用 CloudWatch 日志进行监控

启用AWS IoT 日志记录后，当每条消息从您的设备通过消息代理和规则引擎传递时， AWS IoT 发送有关每条消息的进度事件。在CloudWatch 控制台中， CloudWatch 日志显示在名为的日志组中AWSIotLogs。

有关 CloudWatch 日志的更多信息，请参阅CloudWatch 日志。有关支持的 AWS IoT CloudWatch 日志的信息，请参阅CloudWatch 记录 AWS IoT 日志条目。

在 CloudWatch 控制台中查看 AWS IoT 日志

> **Note**
>
> 该`AWSIotLogsV2`日志组在 CloudWatch 控制台中不可见，直到：
>
> • 您已启用登录功能 AWS IoT。有关如何启用登录功能的更多信息 AWS IoT，请参阅 配置 AWS IoT 日志

- 一些日志条目是由 AWS IoT 操作写入的。

在 CloudWatch 控制台中查看您的 AWS IoT 日志

1. 浏览到 https://console.aws.amazon.com/cloudwatch/。在导航窗格中，选择 日志组。
2. 在筛选条件文本框中，输入 **AWSIotLogsV2**，然后按 Enter。
3. 双击 AWSIotLogsV2 日志组。
4. 选择 Search All（搜索全部）。将显示为您的账户生成的 AWS IoT 日志的完整列表。
5. 选择展开图标可查看单个流。

您也可以在筛选事件文本框中输入查询。可以尝试以下受到较多关注的查询：

- `{ $.logLevel = "INFO" }`

  查找所有日志级别为 INFO 的日志。
- `{ $.status = "Success" }`

  查找所有状态为 Success 的日志。
- `{ $.status = "Success" && $.eventType = "GetThingShadow" }`

  查找所有状态为 Success 且事件类型为 GetThingShadow 的日志。

有关创建筛选表达式的更多信息，请参阅CloudWatch 日志查询。

CloudWatch 记录 AWS IoT 日志条目

的每个组件 AWS IoT 都会生成自己的日志条目。每个日志条目都有一个 eventType 指定导致生成日志条目的操作。本部分介绍以下 AWS IoT 组件所生成的日志条目。

主题

- 消息代理日志条目
- 服务器证书 OCSP 日志条目
- 设备影子日志条目
- 规则引擎日志条目
- 任务日志条目

- 设备预配置日志条目
- 动态事物组日志条目
- 机群索引日志条目
- 常用 CloudWatch 日志属性

消息代理日志条目

AWS IoT 消息代理生成以下事件的日志条目：

主题

- Connect 日志条目
- Disconnect 日志条目
- GetRetainedMessage 日志条目
- ListRetainedMessage 日志条目
- Publish-In 日志条目
- Publish-Out 日志条目
- 排队的日志条目
- Subscribe 日志条目

Connect 日志条目

当 MQTT 客户端连接`Connect`时，AWS IoT 消息代理会生成一个带有`eventType`的日志条目。

Connect 日志条目示例

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Connect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
```

```
}
```

除了 常用 CloudWatch 日志属性 之外，Connect 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

principalId

发出请求的委托人的 ID。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

sourceIp

请求的源 IP 地址。

sourcePort

请求的源端口。

Disconnect 日志条目

当 MQTT 客户端断开连接Disconnect时， AWS IoT 消息代理会生成一个带有eventType的日志条目。

Disconnect 日志条目示例

```
{
    "timestamp": "2017-08-10 15:37:23.476",
    "logLevel": "INFO",
    "traceId": "20b23f3f-d7f1-feae-169f-82263394fbdb",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Disconnect",
    "protocol": "MQTT",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490,
    "reason": "DUPLICATE_CLIENT_ID",
    "details": "A new connection was established with the same client ID",
```

```
    "disconnectReason": "CLIENT_INITIATED_DISCONNECT"
}
```

除了 常用 CloudWatch 日志属性 之外，Disconnect 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

principalId

发出请求的委托人的 ID。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

sourceIp

请求的源 IP 地址。

sourcePort

请求的源端口。

reason

客户端断开连接的原因。

details

错误的简要说明。

disconnectReason

客户端断开连接的原因。

GetRetainedMessage 日志条目

AWS IoT 消息代理生成一个日志条目，eventType其中包含GetRetainedMessage何时GetRetainedMessage被调用。

GetRetainedMessage 日志条目示例

```
{
    "timestamp": "2017-08-07 18:47:56.664",
```

```
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetRetainedMessage",
    "protocol": "HTTP",
    "topicName": "a/b/c",
    "qos": "1",
    "lastModifiedDate": "2017-08-07 18:47:56.664"
}
```

除了 常用 CloudWatch 日志属性 之外，GetRetainedMessage 日志条目还包含以下属性：

最后一次 ModifiedDate

存储保留消息的 Epoch 日期和时间（以毫秒为单位）。 AWS IoT

protocol

用于提出请求的协议。有效值：HTTP。

qos

发布请求中使用的服务质量 (QoS) 级别。有效值为 0 或 1。

topicName

已订阅主题的名称。

ListRetainedMessage 日志条目

AWS IoT 消息代理生成一个日志条目，eventType其中包含ListRetainedMessage何时ListRetainedMessages被调用。

ListRetainedMessage 日志条目示例

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "ListRetainedMessage",
    "protocol": "HTTP"
```

```
}
```

除了 常用 CloudWatch 日志属性 之外，ListRetainedMessage 日志条目还包含以下属性：

protocol

用于提出请求的协议。有效值：HTTP。

Publish-In 日志条目

当 AWS IoT 消息代理收到 MQTT 消息时，它会生成一个带有为eventType的Publish-In日志条目。

Publish-In 日志条目示例

```
{
        "timestamp": "2017-08-10 15:39:30.961",
        "logLevel": "INFO",
        "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
        "accountId": "123456789012",
        "status": "Success",
        "eventType": "Publish-In",
        "protocol": "MQTT",
        "topicName": "$aws/things/MyThing/shadow/get",
        "clientId": "abf27092886e49a8a5c1922749736453",
        "principalId":
 "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
        "sourceIp": "205.251.233.181",
        "sourcePort": 13490,
        "retain": "True"
    }
```

除了 常用 CloudWatch 日志属性 之外，Publish-In 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

principalId

发出请求的委托人的 ID。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

保留

当消息的 REATIN 标志的值设置为 True 时使用的属性。如果消息未设置 REATIN 标志，则该属性不会显示在日志条目中。有关更多信息，请参阅 MQTT 保留消息。

sourceIp

请求的源 IP 地址。

sourcePort

请求的源端口。

topicName

已订阅主题的名称。

Publish-Out 日志条目

当消息代理发布 MQTT 消息时，它会生成一个 eventType 为 Publish-Out 的日志条目

Publish-Out 日志条目示例

```
{
    "timestamp": "2017-08-10 15:39:30.961",
    "logLevel": "INFO",
    "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Publish-Out",
    "protocol": "MQTT",
    "topicName": "$aws/things/MyThing/shadow/get",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

除了 常用 CloudWatch 日志属性 之外，Publish-Out 日志条目还包含以下属性：

clientId

接收有关该 MQTT 主题的消息的订阅客户端的 ID。

principalId

发出请求的委托人的 ID。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

sourceIp

请求的源 IP 地址。

sourcePort

请求的源端口。

topicName

已订阅主题的名称。

排队的日志条目

当具有持续会话的设备断开连接时，MQTT 消息代理会存储该设备的消息，并 AWS IoT 生成事件类型为 EventType 的日志条目。Queued有关 MQTT 持久性会话的更多信息，请参阅MQTT 持久性会话。

排队的服务器错误日志条目示例

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "ERROR",
    "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
    "accountId": "123456789012",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
    "eventType": "Queued",
    "status": "Failure",
    "details": "Server Error"
}
```

除了常用 CloudWatch 日志属性之外，Queued 服务器错误日志条目还包含以下属性：

clientId

消息排队的客户端的 ID。

details

`Server Error`

服务器错误导致无法存储消息。

protocol

用于提出请求的协议。该值始终为 MQTT。

qos

请求的服务质量（QoS）级别。该值将始终为 1，因为不存储 QoS 为 0 的消息。

topicName

已订阅主题的名称。

排队的成功日志条目示例

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "INFO",
    "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
    "accountId": "123456789012",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
    "eventType": "Queued",
    "status": "Success"
}
```

除了常用 CloudWatch 日志属性之外，Queued 成功日志条目还包含以下属性：

clientId

消息排队的客户端的 ID。

protocol

用于提出请求的协议。该值始终为 MQTT。

qos

请求的服务质量（QoS）级别。该值将始终为 1，因为不存储 QoS 为 0 的消息。

topicName

已订阅主题的名称。

排队的限制日志条目示例

```
{
    "timestamp": "2022-08-10 15:39:30.961",
    "logLevel": "ERROR",
    "traceId": "672ec480-31ce-fd8b-b5fb-22e3ac420699",
    "accountId": "123456789012",
    "topicName": "$aws/things/MyThing/get",
    "clientId": "123123123",
    "qos": "1",
    "protocol": "MQTT",
    "eventType": "Queued",
    "status": "Failure",
    "details": "Throttled while queueing offline message"
}
```

除了常用 CloudWatch 日志属性之外，Queued 受限的日志条目还包含以下属性：

clientId

消息排队的客户端的 ID。

details

`Throttled while queueing offline message`

客户端超出了 `Queued messages per second per account` 限制，因此未存储消息。

protocol

用于提出请求的协议。该值始终为 MQTT。

qos

请求的服务质量（QoS）级别。该值将始终为 1，因为不存储 QoS 为 0 的消息。

topicName

已订阅主题的名称。

Subscribe 日志条目

当 MQTT 客户端订阅主题Subscribe时，AWS IoT 消息代理会生成一个eventType带有的日志条目。

MQTT 3 订阅日志条目示例

```
{
    "timestamp": "2017-08-10 15:39:04.413",
    "logLevel": "INFO",
    "traceId": "7aa5c38d-1b49-3753-15dc-513ce4ab9fa6",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "Subscribe",
    "protocol": "MQTT",
    "topicName": "$aws/things/MyThing/shadow/#",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "sourceIp": "205.251.233.181",
    "sourcePort": 13490
}
```

除了 常用 CloudWatch 日志属性 之外，Subscribe 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

principalId

发出请求的委托人的 ID。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

sourceIp

请求的源 IP 地址。

sourcePort

请求的源端口。

topicName

已订阅主题的名称。

MQTT 5 订阅日志条目示例

```
{
 "timestamp": "2022-11-30 16:24:15.628",
 "logLevel": "INFO",
 "traceId": "7aa5c38d-1b49-3753-15dc-513ce4ab9fa6",
 "accountId": "123456789012",
 "status": "Success",
 "eventType": "Subscribe",
 "protocol": "MQTT",
 "topicName": "test/topic1,$invalid/reserved/topic",
 "subscriptions": [
  {
   "topicName": "test/topic1",
   "reasonCode": 1
  },
  {
   "topicName": "$invalid/reserved/topic",
   "reasonCode": 143
  }
 ],
 "clientId": "abf27092886e49a8a5c1922749736453",
 "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
 "sourceIp": "205.251.233.181",
 "sourcePort": 13490
}
```

对于 MQTT 5 订阅操作，除了 常用 CloudWatch 日志属性和 MQTT 3 订阅日志条目属性外，MQTT 5 Subscribe 日志条目还包含以下属性：

订阅

订阅请求中的已请求主题与各个 MQTT 5 原因代码之间的映射列表。有关更多信息，请参阅 MQTT 原因代码。

服务器证书 OCSP 日志条目

AWS IoT Core 为以下事件生成日志条目：

主题

- 检索 OCSP 日志条目 StapleData

检索 OCSP 日志条目 StapleData

AWS IoT Core 当服务器检索 OCSP 装订数据`RetrieveOCSPStapleData`时，会生成一个`eventType`带有的日志条目。

检索 OCSP 日志条目示例 StapleData

以下是的日志条目示例`Success`。


```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "INFO",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Success",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
 "connectionDetails": {
  "httpStatusCode": "200",
  "ocspResponderUri": "http://ocsp.example.com",
  "sourceIp": "205.251.233.181",
  "targetIp": "250.15.5.3"
 },
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 },
 "ocspResponseDetails": {
  "responseCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
  "ocspResponseStatus": "successful",
  "certStatus": "good",
  "signature":
 "4C:6F:63:61:6C:20:52:65:73:70:6F:6E:64:65:72:20:53:69:67:6E:61:74:75:72:65",
```

```
  "thisUpdateTime": "Jan 31 01:21:02 2024 UTC",
  "nextUpdateTime": "Feb 02 00:21:02 2024 UTC",
  "producedAtTime": "Jan 31 01:37:03 2024 UTC",
  "stapledDataPayloadSize": "XXX"
 }
}
```

以下是的日志条目示例Failure。

```
{
 "timestamp": "2024-01-30 15:39:30.961",
 "logLevel": "ERROR",
 "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
 "accountId": "123456789012",
 "status": "Failure",
 "reason": "A non 2xx HTTP response was received from the OCSP responder.",
 "eventType": "RetrieveOCSPStapleData",
 "domainConfigName": "test-domain-config-name",
 "connectionDetails": {
  "httpStatusCode": "444",
  "ocspResponderUri": "http://ocsp.example.com",
  "sourceIp": "205.251.233.181",
  "targetIp": "250.15.5.3"
 },
 "ocspRequestDetails": {
  "requesterName": "iot.amazonaws.com",
  "requestCertId":
 "30:3A:30:09:06:05:2B:0E:03:02:1A:05:00:04:14:9C:FF:90:A1:97:B0:4D:6C:01:B9:69:96:D8:3E:E7:A2:
 }
}
```

对于该RetrieveOCSPStaple操作，除了常用 CloudWatch 日志属性，日志条目还包含以下属性：

reason

操作失败的原因。

域 ConfigName

您的域名配置的名称。

连接详情

连接详情的简要说明。

- HTTP StatusCode

  OCSP 响应者为响应客户端向服务器发出的请求而返回的 HTTP 状态码。

- ocsp ResponderUri

  从服务器证书中 AWS IoT Core 获取的 OCSP 响应器 URI。

- sourceIp

  AWS IoT Core 服务器的源 IP 地址。

- TargetIP

  OCSP 响应者的目标 IP 地址。

ocsp RequestDetails

OCSP 请求的详细信息。

- 请求者姓名

  向 OCSP 响应者发送请求的 AWS IoT Core 服务器的标识符。

- 请求 CertId

  请求的证书 ID。这是正在请求 OCSP 响应的证书的 ID。

ocsp ResponseDetails

OCSP 响应的详细信息。

- 响应 CertId

  OCSP 响应的证书 ID。

- ocsp ResponseStatus

  OCSP 响应的状态。

- CertStatus

  证书的状态。

- signature

  可信实体应用于响应的签名。

- 这个 UpdateTime

  已知所指示状态的正确时间。

- 下一步 UpdateTime

  在此时间或之前，将提供有关证书状态的新信息。

- 制作 AtTime

  OCSP 响应者签署此响应的时间。

- 装订尺寸 DataPayload

  装订数据的有效载荷大小。

设备影子日志条目

AWS IoT Device Shadow 服务会为以下事件生成日志条目：

主题

- DeleteThingShadow 日志条目
- GetThingShadow 日志条目
- UpdateThingShadow 日志条目

DeleteThingShadow 日志条目

当收到删除设备影子的请求时，Device Shadow 服务会生成一个 `eventType` 为 `DeleteThingShadow` 的日志条目。

DeleteThingShadow 日志条目示例

```
{
    "timestamp": "2017-08-07 18:47:56.664",
    "logLevel": "INFO",
    "traceId": "1a60d02e-15b9-605b-7096-a9f584a6ad3f",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DeleteThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "Jack",
    "topicName": "$aws/things/Jack/shadow/delete"
}
```

除了 常用 CloudWatch 日志属性 之外，`DeleteThingShadow` 日志条目还包含以下属性：

设备 ShadowName

要更新的影子的名称。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

topicName

发布请求时所基于的主题的名称。

GetThingShadow 日志条目

当收到影子的获取请求时，Device Shadow 服务会生成一个 eventType 为 GetThingShadow 的日志条目。

GetThingShadow 日志条目示例

```
{
    "timestamp": "2017-08-09 17:56:30.941",
    "logLevel": "INFO",
    "traceId": "b575f19a-97a2-cf72-0ed0-c64a783a2504",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "MyThing",
    "topicName": "$aws/things/MyThing/shadow/get"
}
```

除了 常用 CloudWatch 日志属性 之外，GetThingShadow 日志条目还包含以下属性：

设备 ShadowName

所请求的影子的名称。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

topicName

发布请求时所基于的主题的名称。

UpdateThingShadow 日志条目

当收到更新设备影子的请求时，Device Shadow 服务会生成一个 eventType 为 UpdateThingShadow 的日志条目。

UpdateThingShadow 日志条目示例

```
{
    "timestamp": "2017-08-07 18:43:59.436",
    "logLevel": "INFO",
    "traceId": "d0074ba8-0c4b-a400-69df-76326d414c28",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "UpdateThingShadow",
    "protocol": "MQTT",
    "deviceShadowName": "Jack",
    "topicName": "$aws/things/Jack/shadow/update"
}
```

除了 常用 CloudWatch 日志属性 之外，UpdateThingShadow 日志条目还包含以下属性：

设备 ShadowName

要更新的影子的名称。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

topicName

发布请求时所基于的主题的名称。

规则引擎日志条目

AWS IoT 规则引擎为以下事件生成日志：

主题

- FunctionExecution 日志条目
- RuleExecution 日志条目
- RuleMatch 日志条目

- RuleExecutionThrottled 日志条目
- RuleNotFound 日志条目
- StartingRuleExecution 日志条目

FunctionExecution 日志条目

当规则的 SQL 查询调用外部函数时，规则引擎会生成一个 eventType 为 FunctionExecution 的日志条目。当规则的操作向或其他 Web 服务发出 HTTP 请求（例如，调用 AWS IoT get_thing_shadow或machinelearning_predict）时，就会调用外部函数。

FunctionExecution 日志条目示例

```
{
    "timestamp": "2017-07-13 18:33:51.903",
    "logLevel": "DEBUG",
    "traceId": "180532b7-0cc7-057b-687a-5ca1824838f5",
    "status": "Success",
    "eventType": "FunctionExecution",
    "clientId": "N/A",
    "topicName":"rules/test",
    "ruleName": "ruleTestPredict",
    "ruleAction": "MachinelearningPredict",
    "resources": {
        "ModelId": "predict-model"
    },
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

除了 常用 CloudWatch 日志属性 之外，FunctionExecution 日志条目还包含以下属性：

clientId

N/A (针对 FunctionExecution 日志)。

principalId

发出请求的委托人的 ID。

资源

规则的操作所使用的资源的集合。

ruleName

匹配规则的名称。

topicName

已订阅主题的名称。

RuleExecution 日志条目

当 AWS IoT 规则引擎触发规则的操作时，它会生成一个RuleExecution日志条目。

RuleExecution 日志条目示例

```
{
    "timestamp": "2017-08-10 16:32:46.070",
    "logLevel": "INFO",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleExecution",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "ruleAction": "RepublishAction",
    "resources": {
        "RepublishTopic": "rules/republish"
    },
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

除了 常用 CloudWatch 日志属性 之外，RuleExecution 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

principalId

发出请求的委托人的 ID。

资源

规则的操作所使用的资源的集合。

ruleAction

所触发的操作的名称。

ruleName

匹配规则的名称。

topicName

已订阅主题的名称。

RuleMatch 日志条目

当消息代理收到与 AWS IoT 规则匹配`eventType`的消息`RuleMatch`时，规则引擎会生成一个带有的日志条目。

RuleMatch 日志条目示例

```
{
    "timestamp": "2017-08-10 16:32:46.002",
    "logLevel": "INFO",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "RuleMatch",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

除了 常用 CloudWatch 日志属性 之外，`RuleMatch` 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

principalId

发出请求的委托人的 ID。

ruleName

匹配规则的名称。

topicName

已订阅主题的名称。

RuleExecutionThrottled 日志条目

当执行受到限制时，AWS IoT 规则引擎会生成一个eventType带有为的日志条目。RuleExecutionThrottled

RuleExecutionThrottled 日志条目示例

```
{
    "timestamp": "2017-10-04 19:25:46.070",
    "logLevel": "ERROR",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleMessageThrottled",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "$aws/rules/example_rule",
    "ruleName": "example_rule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "RuleExecutionThrottled",
    "details": "Exection of Rule example_rule throttled"
}
```

除了 常用 CloudWatch 日志属性 之外，RuleExecutionThrottled 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

details

错误的简要说明。

principalId

发出请求的委托人的 ID。

reason

字符串 “RuleExecutionThrottled”。

ruleName

要触发的规则的名称。

topicName

已发布的主题的名称。

RuleNotFound 日志条目

当 AWS IoT 规则引擎找不到具有给定名称的规则时，它会生成一个带有 of `eventType` 的日志条目`RuleNotFound`。

RuleNotFound 日志条目示例

```
{
    "timestamp": "2017-10-04 19:25:46.070",
    "logLevel": "ERROR",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleNotFound",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "$aws/rules/example_rule",
    "ruleName": "example_rule",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167",
    "reason": "RuleNotFound",
    "details": "Rule example_rule not found"
}
```

除了 常用 CloudWatch 日志属性 之外，`RuleNotFound` 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

details

错误的简要说明。

principalId

发出请求的委托人的 ID。

reason

字符串 “已RuleNot找到”。

ruleName

找不到的规则的名称。

topicName

已发布的主题的名称。

StartingRuleExecution 日志条目

当 AWS IoT 规则引擎开始触发规则的操作时，它会生成一个带为eventType的日志条目StartingRuleExecution。

StartingRuleExecution 日志条目示例

```
{
    "timestamp": "2017-08-10 16:32:46.002",
    "logLevel": "DEBUG",
    "traceId": "30aa7ccc-1d23-0b97-aa7b-76196d83537e",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "StartingRuleExecution",
    "clientId": "abf27092886e49a8a5c1922749736453",
    "topicName": "rules/test",
    "ruleName": "JSONLogsRule",
    "ruleAction": "RepublishAction",
    "principalId": "145179c40e2219e18a909d896a5340b74cf97a39641beec2fc3eeafc5a932167"
}
```

除了 常用 CloudWatch 日志属性 之外，rule- 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

principalId

发出请求的委托人的 ID。

ruleAction

所触发的操作的名称。

ruleName

匹配规则的名称。

topicName

已订阅主题的名称。

任务日志条目

AWS IoT Job 服务为以下事件生成日志条目。当从设备收到 MQTT 或 HTTP 请求时，将生成日志条目。

主题

- DescribeJobExecution 日志条目
- GetPendingJobExecution 日志条目
- ReportFinalJobExecutionCount 日志条目
- StartNextPendingJobExecution 日志条目
- UpdateJobExecution 日志条目

DescribeJobExecution 日志条目

当服务收到描述 AWS IoT 任务执行`eventType`的请求`DescribeJobExecution`时，该服务会生成一个带有的日志条目。

DescribeJobExecution 日志条目示例

```
{
    "timestamp": "2017-08-10 19:13:22.841",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "DescribeJobExecution",
    "protocol": "MQTT",
    "clientId": "thingOne",
    "jobId": "002",
    "topicName": "$aws/things/thingOne/jobs/002/get",
    "clientToken": "myToken",
    "details": "The request status is SUCCESS."
```

```
}
```

除了 常用 CloudWatch 日志属性 之外，GetJobExecution 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

clientToken

用于确保请求幂等性的唯一、区分大小写的标识符。有关更多信息，请参阅如何确保幂等性。

details

来自 Jobs 服务的其它信息。

jobId

任务执行的任务 ID。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

topicName

发出请求所使用的主题。

GetPendingJobExecution 日志条目

当服务收到 AWS IoT 任务执行请求GetPendingJobExecution时，该服务会生成一个eventType带有的日志条目。

GetPendingJobExecution 日志条目示例

```
{
    "timestamp": "2018-06-13 17:45:17.197",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "GetPendingJobExecution",
    "protocol": "MQTT",
    "clientId": "299966ad-54de-40b4-99d3-4fc8b52da0c5",
    "topicName": "$aws/things/299966ad-54de-40b4-99d3-4fc8b52da0c5/jobs/get",
```

```
    "clientToken": "24b9a741-15a7-44fc-bd3c-1ff2e34e5e82",
    "details": "The request status is SUCCESS."
}
```

除了 常用 CloudWatch 日志属性 之外，GetPendingJobExecution 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

clientToken

用于确保请求幂等性的唯一、区分大小写的标识符。有关更多信息，请参阅如何确保幂等性。

details

来自 Jobs 服务的其它信息。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

topicName

已订阅主题的名称。

ReportFinalJobExecutionCount 日志条目

AWS IoT 作业完成ReportFinalJobExecutionCount后，作业服务会生成一个带有 “” entryType 的日志条目。

ReportFinalJobExecutionCount 日志条目示例

```
{
    "timestamp": "2017-08-10 19:44:16.776",
    "logLevel": "INFO",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "ReportFinalJobExecutionCount",
    "jobId": "002",
    "details": "Job 002 completed. QUEUED job execution count: 0 IN_PROGRESS job
 execution count: 0 FAILED job execution count: 0 SUCCEEDED job execution count: 1
 CANCELED job execution count: 0 REJECTED job execution count: 0 REMOVED job execution
 count: 0"
```

```
}
```

除了 常用 CloudWatch 日志属性 之外，ReportFinalJobExecutionCount 日志条目还包含以下属性：

details

来自 Jobs 服务的其它信息。

jobId

任务执行的任务 ID。

StartNextPendingJobExecution 日志条目

当它收到开始下一个待处理的任务执行的请求时， AWS IoT 作业服务会生成一个带有为eventType的日志条目StartNextPendingJobExecution。

StartNextPendingJobExecution 日志条目示例

```
{
    "timestamp": "2018-06-13 17:49:51.036",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "StartNextPendingJobExecution",
    "protocol": "MQTT",
    "clientId": "95c47808-b1ca-4794-bc68-a588d6d9216c",
    "topicName": "$aws/things/95c47808-b1ca-4794-bc68-a588d6d9216c/jobs/start-next",
    "clientToken": "bd7447c4-3a05-49f4-8517-dd89b2c68d94",
    "details": "The request status is SUCCESS."
}
```

除了 常用 CloudWatch 日志属性 之外，StartNextPendingJobExecution 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

clientToken

用于确保请求幂等性的唯一、区分大小写的标识符。有关更多信息，请参阅如何确保幂等性。

details

来自 Jobs 服务的其它信息。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

topicName

发出请求所使用的主题。

UpdateJobExecution 日志条目

当服务收到更新 AWS IoT 任务执行eventType的请求UpdateJobExecution时，该服务会生成一个带有的日志条目。

UpdateJobExecution 日志条目示例

```
{
    "timestamp": "2017-08-10 19:25:14.758",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "eventType": "UpdateJobExecution",
    "protocol": "MQTT",
    "clientId": "thingOne",
    "jobId": "002",
    "topicName": "$aws/things/thingOne/jobs/002/update",
    "clientToken": "myClientToken",
    "versionNumber": "1",
    "details": "The destination status is IN_PROGRESS. The request status is SUCCESS."
}
```

除了 常用 CloudWatch 日志属性 之外，UpdateJobExecution 日志条目还包含以下属性：

clientId

发出请求的客户端的 ID。

clientToken

用于确保请求幂等性的唯一、区分大小写的标识符。有关更多信息，请参阅如何确保幂等性。

details

来自 Jobs 服务的其它信息。

jobId

任务执行的任务 ID。

protocol

用于提出请求的协议。有效值为 MQTT 或 HTTP。

topicName

发出请求所使用的主题。

versionNumber

任务执行的版本。

设备预配置日志条目

AWS IoT 设备配置服务会生成以下事件的日志。

主题

- GetDeviceCredentials 日志条目
- ProvisionDevice 日志条目

GetDeviceCredentials 日志条目

当客户端呼叫GetDeviceCredential时， AWS IoT 设备配置服务会生成一个带有 “” eventType 的日志条目GetDeviceCredential。

GetDevice凭证日志条目示例

```
{
  "timestamp" : "2019-02-20 20:31:22.932",
  "logLevel" : "INFO",
  "traceId" : "8d9c016f-6cc7-441e-8909-7ee3d5563405",
  "accountId" : "123456789101",
  "status" : "Success",
  "eventType" : "GetDeviceCredentials",
```

```
  "deviceCertificateId" :
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "details" : "Additional details about this log."
}
```

除了 常用 CloudWatch 日志属性 之外，GetDeviceCredentials 日志条目还包含以下属性：

details

错误的简要说明。

设备 CertificateId

设备证书的 ID。

ProvisionDevice 日志条目

当客户端呼叫ProvisionDevice时， AWS IoT 设备配置服务会生成一个带有 “” eventType 的日志条目ProvisionDevice。

ProvisionDevice 日志条目示例

```
{
  "timestamp" : "2019-02-20 20:31:22.932",
  "logLevel" : "INFO",
  "traceId" : "8d9c016f-6cc7-441e-8909-7ee3d5563405",
  "accountId" : "123456789101",
  "status" : "Success",
  "eventType" : "ProvisionDevice",
  "provisioningTemplateName" : "myTemplate",
  "deviceCertificateId" :
 "e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "details" : "Additional details about this log."
 }
```

除了 常用 CloudWatch 日志属性 之外，ProvisionDevice 日志条目还包含以下属性：

details

错误的简要说明。

设备 CertificateId

设备证书的 ID。

资源调配 TemplateName

预配置模板的名称。

动态事物组日志条目

AWS IoT 动态事物组生成以下事件的日志。

主题

- AddThingToDynamicThingGroupsFailed 日志条目

AddThingToDynamicThingGroupsFailed 日志条目

AWS IoT 当无法将事物添加到指定的动态组时，它会生成一个带有为eventType的日志条目AddThingToDynamicThingGroupsFailed。当事物满足要处于动态事物组中的条件时，就会发生此情况；但是，无法将其添加到动态组中，或者无法将其从动态组中删除。这可能是因为：

- 事物已经属于最大数量的组数。
- --override-dynamic-groups 选项用于将事物添加到静态事物组中。它已被从一个动态事物组中删除，以便可以再添加。

有关更多信息，请参阅动态事物组限制和冲突。

AddThingToDynamicThingGroupsFailed 日志条目示例

此示例显示 AddThingToDynamicThingGroupsFailed 错误的日志条目。在此示例中，TestThing符合中列出的动态事物组的条件dynamicThingGroupNames，但无法添加到这些动态组中，如中所述reason。

```
{
 "timestamp": "2020-03-16 22:24:43.804",
 "logLevel": "ERROR",
 "traceId": "70b1f2f5-d95e-f897-9dcc-31e68c3e1a30",
 "accountId": "57EXAMPLE833",
 "status": "Failure",
 "eventType": "AddThingToDynamicThingGroupsFailed",
 "thingName": "TestThing",
 "dynamicThingGroupNames": [
  "DynamicThingGroup11",
```

```
  "DynamicThingGroup12",
  "DynamicThingGroup13",
  "DynamicThingGroup14"
 ],
 "reason": "The thing failed to be added to the given dynamic thing group(s) because
 the thing already belongs to the maximum allowed number of groups."
}
```

除了 常用 CloudWatch 日志属性 之外，AddThingToDynamicThingGroupsFailed 日志条目还包含以下属性：

动态ThingGroup名称

无法向其中添加该事物的动态事物组的数组。

reason

事物无法添加到动态事物组的原因。

thingName

无法添加到动态事物组的事物的名称。

机群索引日志条目

AWS IoT 舰队索引生成以下事件的日志条目。

主题

- NamedShadowCountForDynamicGroupQueryLimitExceeded 日志条目

NamedShadowCountForDynamicGroupQueryLimitExceeded 日志条目

对于动态组中不特定于数据来源的查询术语，每个对象最多处理 25 个命名影子。由于事件而违反该限制时，将发出 NamedShadowCountForDynamicGroupQueryLimitExceeded 事件类型。

NamedShadowCountForDynamicGroupQueryLimitExceeded 日志条目示例

此示例显示 NamedShadowCountForDynamicGroupQueryLimitExceeded 错误的日志条目。在此示例中，所有基于值的 DynamicGroup 结果都可能不准确，如 reason 字段中所述。

```
{
"timestamp": "2020-03-16 22:24:43.804",
```

```
"logLevel": "ERROR",
"traceId": "70b1f2f5-d95e-f897-9dcc-31e68c3e1a30",
"accountId": "571032923833",
"status": "Failure",
"eventType": "NamedShadowCountForDynamicGroupQueryLimitExceeded",
"thingName": "TestThing",
"reason": "A maximum of 25 named shadows per thing are processed for non-data source
 specific query terms in dynamic groups."
}
```

常用 CloudWatch 日志属性

所有 Log CloudWatch s 日志条目都包含以下属性：

accountId

你的 AWS 账户 身份证。

eventType

已为其生成日志的事件类型。事件类型的值取决于生成日志条目的事件。每个日志条目描述都包括该日志条目的 `eventType` 的值。

logLevel

所使用的日志级别。有关更多信息，请参阅 the section called “日志级别”。

status

请求的状态。

时间戳

客户端连接到 AWS IoT 消息代理的时间的便于阅读的 UNIX 时间戳。

traceId

一个随机生成的标识符，可用于将特定请求的所有日志关联起来。

将设备端日志上传到 Amazon CloudWatch

您可以将设备端的历史日志上传到 Amazon CloudWatch ，以监控和分析设备在现场的活动。设备端日志可以包括系统、应用程序和设备日志文件。此过程使用 CloudWatch Logs rules 操作参数将设备端日志发布到客户定义的日志组中。

工作方式

当 AWS IoT 设备向 AWS IoT 主题发送包含格式化日志文件的 MQTT 消息时，该过程就开始了。AWS IoT 规则监控消息主题并将日志文件发送到您定义的 CloudWatch 日志组。然后，您可以查看和分析信息。

主题

- MQTT 主题
- 规则操作

MQTT 主题

选择要用于发布日志的 MQTT 主题命名空间。我们建议将此格式用于公共主题空间 `$aws/rules/things/thing_name/logs`，并将此格式用于错误主题 `$aws/rules/things/thing_name/logs/errors`。建议使用日志和错误主题的命名结构，但不是必需要求。有关更多信息，请参阅为 AWS IoT Core设计 MQTT 主题。

通过使用推荐的公共主题空间，您可以使用 B AWS IoT asic Ingest 保留主题。 AWS IoT Basic Ingest 可以安全地将设备数据发送到 AWS IoT 规则操作支持的 AWS 服务。它从摄取路径中删除发布/订阅消息代理，因此更具成本效益。有关更多信息，请参阅借助基本摄取功能降低消息收发成本。

如果您使用 batchMode 上载日志文件，则您的消息必须遵循包括 UNIX 时间戳和消息的特定格式。有关更多信息，请参阅CloudWatch 日志规则操作中的 BatchMode 的 MQTT 消息格式要求主题。

规则操作

当 AWS IoT 收到来自客户端设备的 MQTT 消息时， AWS IoT 规则会监控客户定义的主题并将内容发布到您定义的 CloudWatch 日志组中。此过程使用 CloudWatch 日志规则操作来监控 MQTT 中是否有批量日志文件。有关更多信息，请参阅CloudWatch 日志 AWS IoT 规则操作。

批量模式

`batchMode`是 “ AWS IoT CloudWatch 日志” 规则操作中的一个布尔参数。此参数是可选的，默认情况下处于关闭状态（`false`）。要批量上传设备端日志文件，必须在创建规则时启用此参数 (`true`)。AWS IoT 有关更多信息，请参阅AWS IoT 规则操作部分中的CloudWatch 日志。

使用 AWS IoT 规则上载设备端日志

您可以使用 AWS IoT 规则引擎将现有设备端日志文件（系统、应用程序和设备客户端日志）中的日志记录上传到 Amazon。 CloudWatch将设备端日志发布到 MQTT 主题时， CloudWatch 日志规则

操作会将消息传输到日志。 CloudWatch 此过程概述了如何使用开启（设置为 `true`）的规则操作 `batchMode` 参数批量上载设备日志。

要开始将设备端日志上传到 CloudWatch，请完成以下先决条件。

先决条件

开始之前，请执行以下操作：

- 创建至少一台注册 AWS IoT Core 为 AWS IoT 事物的目标物联网设备。有关更多信息，请参阅创建事物对象。
- 确定用于摄取和错误的 MQTT 主题空间。有关 MQTT 主题和推荐命名约定的更多信息，请参阅将设备端日志上传到 Amazon 中的 MQTT 主题 MQTT 主题部分。 CloudWatch

有关这些先决条件的更多信息，请参阅将设备端日志上传到。 CloudWatch

创建 CloudWatch 日志组

要创建 CloudWatch 日志组，请完成以下步骤。根据您更喜欢执行的步骤还是 AWS Command Line Interface (AWS CLI)， AWS Management Console 选择相应的选项卡。

AWS Management Console

要创建 CloudWatch 日志组，请使用 AWS Management Console

1. 打开 AWS Management Console 并导航至CloudWatch。
2. 在导航栏上，选择 Logs（日志），然后选择 Log groups（日志组）。
3. 选择创建日志组。
4. 更新 Log group name（日志组名称），并可选择更新 Retention setting（保留设置）字段。
5. 选择创建。

AWS CLI

要创建 CloudWatch 日志组，请使用 AWS CLI

1. 要创建日志组，请运行以下命令。有关更多信息，请参阅 AWS CLI v2 命令参考`create-log-group`中的。

   将示例中的日志组名称（`uploadLogsGroup`）替换为您首选的名称。

```
aws logs create-log-group --log-group-name uploadLogsGroup
```

2. 要确认已正确创建日志组，请运行以下命令。

```
aws logs describe-log-groups --log-group-name-prefix uploadLogsGroup
```

示例输出：

```
{
    "logGroups": [
        {
            "logGroupName": "uploadLogsGroup",
            "creationTime": 1674521804657,
            "metricFilterCount": 0,
            "arn": "arn:aws:logs:us-east-1:111122223333:log-
group:uploadLogsGroup:*",
            "storedBytes": 0
        }
    ]
}
```

创建主题规则

要创建 AWS IoT 规则，请完成以下步骤。根据您更喜欢执行的步骤还是 AWS Command Line Interface (AWS CLI)， AWS Management Console 选择相应的选项卡。

AWS Management Console

要创建主题规则，请使用 AWS Management Console

1. 打开规则中心。
   a. 打开 AWS Management Console 并导航至AWS IoT。
   b. 在导航栏上，选择 Message routing（消息路由），然后选择 Rules（规则）。
   c. 选择创建规则。
2. 输入规则属性。
   a. 输入由字母数字组成的 Rule name（规则名称）。

b. （可选）输入 Rule description（规则描述）和 Tags（标签）。

c. 选择下一步。

3. 输入 SQL 语句。

a. 使用您为摄取定义的 MQTT 主题输入 SQL 语句。

例如，`SELECT * FROM '$aws/rules/things/thing_name/logs'`

b. 选择下一步。

4. 输入规则操作。

a. 在操作 1 菜单上，选择CloudWatch日志。

b. 选择 Log group name（日志组名称），然后选择您创建的日志组。

c. 选择 Use batch mode（使用批量模式）。

d. 为规则指定 IAM 角色。

如果规则具有 IAM 角色，请执行以下操作。

1. 在 IAM role（IAM 角色）菜单上，选择您的 IAM 角色。

如果规则没有 IAM 角色，请执行以下操作。

1. 选择 Create new role（创建新角色）。
2. 对于 Role name（角色名称），输入唯一名称并选择 Create（创建）。
3. 确认 IAM role（IAM 角色）字段中的 IAM 角色名称是否正确。

e. 选择下一步。

5. 查看模板配置。

a. 查看任务模板的设置以验证它们正确无误。

b. 完成后，选择 Create (创建)。

AWS CLI

要创建 IAM 角色和主题规则，请使用 AWS CLI

1. 创建授予 AWS IoT 规则权限的 IAM 角色。

a. 创建一个 IAM 策略。

要创建 IAM policy，请运行以下命令。确保更新 `policy-name` 参数值。有关更多信息，请参阅 AWS CLI v2 命令参考[create-policy]中的。

> **Note**
>
> 如果您使用的是 Microsoft Windows 操作系统，则可能需要将行尾标记 (\ ) 替换为勾号 (`) 或其他字符。

```
aws iam create-policy \
    --policy-name uploadLogsPolicy \
    --policy-document \
'{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": [
            "iot:CreateTopicRule",
            "iot:Publish",
            "logs:CreateLogGroup",
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:GetLogEvents"
        ],
        "Resource": "*"
    }
}'
```

b. 将输出中的策略 ARN 复制到文本编辑器中。

示例输出：

```
{
    "Policy": {
        "PolicyName": "uploadLogsPolicy",
        "PermissionsBoundaryUsageCount": 0,
        "CreateDate": "2023-01-23T18:30:10Z",
        "AttachmentCount": 0,
        "IsAttachable": true,
        "PolicyId": "AAABBBCCCDDDEEEFFFGGG",
        "DefaultVersionId": "v1",
```

```
        "Path": "/",
        "Arn": "arn:aws:iam::111122223333:policy/uploadLogsPolicy",
        "UpdateDate": "2023-01-23T18:30:10Z"
    }
}
```

c. 创建 IAM 角色和信任策略。

要创建 IAM policy，请运行以下命令。确保更新 `role-name` 参数值。有关更多信息，请参阅 AWS CLI v2 命令参考`create-role`中的。

```
aws iam create-role \
--role-name uploadLogsRole \
--assume-role-policy-document \
'{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}'
```

d. 向角色附加 IAM policy。

要创建 IAM policy，请运行以下命令。确保更新 `role-name` 和 `policy-arn` 参数值。有关更多信息，请参阅 AWS CLI v2 命令参考`attach-role-policy`中的。

```
aws iam attach-role-policy \
--role-name uploadLogsRole \
--policy-arn arn:aws:iam::111122223333:policy/uploadLogsPolicy
```

e. 查看角色。

要确认已正确创建 IAM 角色，请运行以下命令。确保更新 `role-name` 参数值。有关更多信息，请参阅 AWS CLI v2 命令参考`get-role`中的。

```
aws iam get-role --role-name uploadLogsRole
```

示例输出：

```
{
    "Role": {
        "Path": "/",
        "RoleName": "uploadLogsRole",
        "RoleId": "AAABBBCCCDDDEEEFFFGGG",
        "Arn": "arn:aws:iam::111122223333:role/uploadLogsRole",
        "CreateDate": "2023-01-23T19:17:15+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "Statement1",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "iot.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Description": "",
        "MaxSessionDuration": 3600,
        "RoleLastUsed": {}
    }
}
```

2. 在中创建 AWS IoT 主题规则 AWS CLI。

   a. 要创建 AWS IoT 主题规则，请运行以下命令。确保更新 `--rule-name`、`sql` 语句、`description`、`roleARN` 和 `logGroupName` 参数值。有关更多信息，请参阅 v2 命令参考中的 create-topic-rule。 AWS CLI

```
aws iot create-topic-rule \
--rule-name uploadLogsRule \
--topic-rule-payload \
 '{
  "sql":"SELECT * FROM 'rules/things/thing_name/logs'",
```

```
  "description":"Upload logs test rule",
  "ruleDisabled":false,
  "awsIotSqlVersion":"2016-03-23",
  "actions":[
   {"cloudwatchLogs":
    {"roleArn":"arn:aws:iam::111122223333:role/uploadLogsRole",
    "logGroupName":"uploadLogsGroup",
    "batchMode":true}
    }
   ]
}'
```

b. 要确认已正确创建规则，请运行以下命令。确保更新 `role-name` 参数值。有关更多信息，请参阅 v2 命令参考中的 get-topic-rule。 AWS CLI

```
aws iot get-topic-rule --rule-name uploadLogsRule
```

示例输出：

```
{
    "ruleArn": "arn:aws:iot:us-east-1:111122223333:rule/uploadLogsRule",
    "rule": {
        "ruleName": "uploadLogsRule",
        "sql": "SELECT * FROM rules/things/thing_name/logs",
        "description": "Upload logs test rule",
        "createdAt": "2023-01-24T16:28:15+00:00",
        "actions": [
            {
                "cloudwatchLogs": {
                    "roleArn": "arn:aws:iam::111122223333:role/
uploadLogsRole",
                    "logGroupName": "uploadLogsGroup",
                    "batchMode": true
                }
            }
        ],
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23"
    }
}
```

将设备端日志发送到 AWS IoT

将设备端日志发送到 AWS IoT

1. 要向发送历史日志 AWS IoT，请与您的设备通信以确保满足以下要求。

   - 日志信息将发送到在本过程的先决条件部分中指定的正确主题命名空间。

     例如，`$aws/rules/things/thing_name/logs`

   - MQTT 消息有效负载的格式正确无误。有关 MQTT 主题和建议的命名规则的更多信息，请参阅将设备端日志上传到 Amazon CloudWatch中的 MQTT 主题 部分。

2. 确认 MQTT 消息已在 M AWS IoT QTT 客户端中收到。

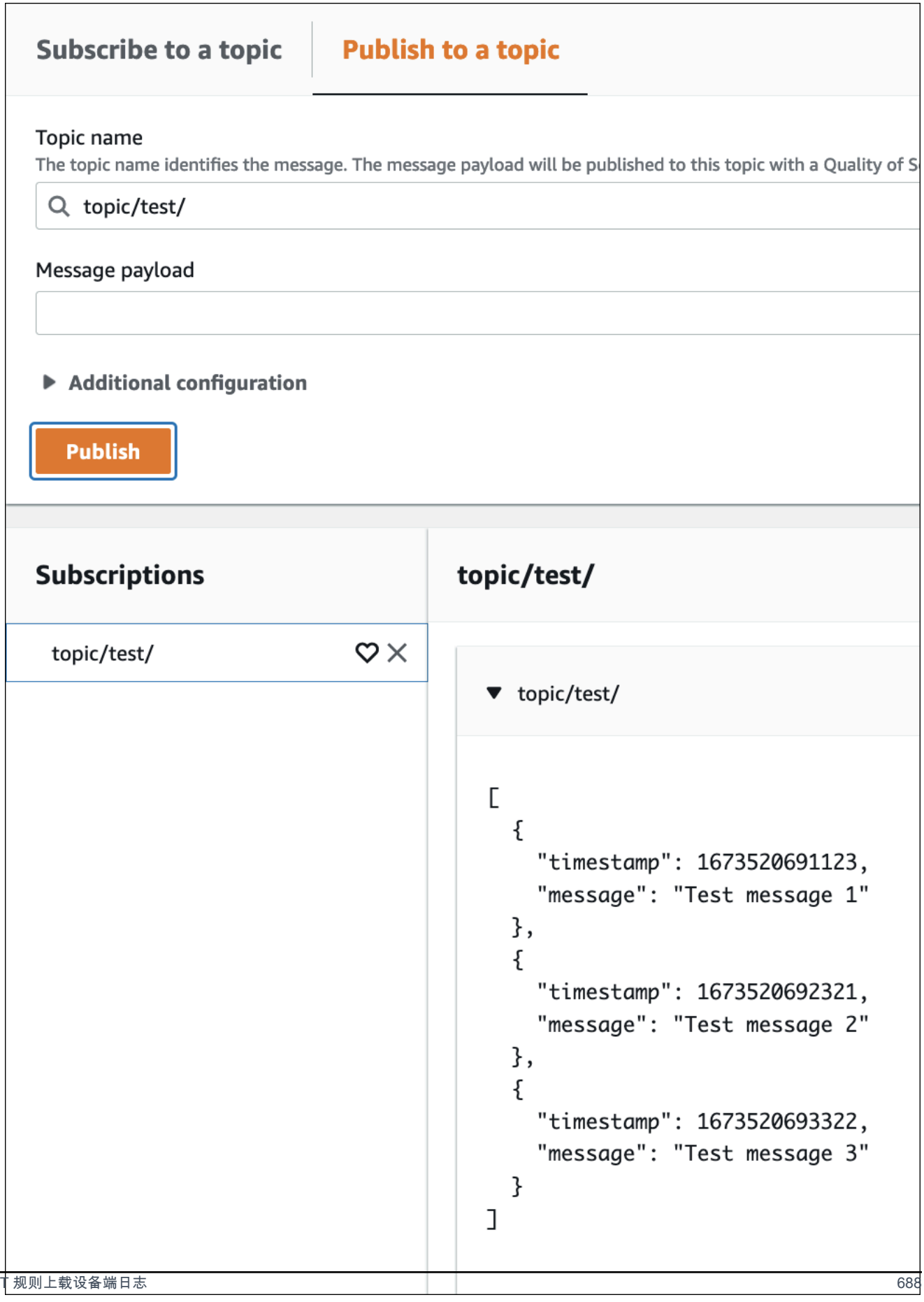
   a. 打开 AWS Management Console 并导航至AWS IoT。

   b. 要查看 MQTT test client（MQTT 测试客户端），请在导航栏上选择 Test（测试）、MQTT test client（MQTT 测试客户端）。

   c. 对于 Subscribe to a topic（订阅主题）、Topic filter（主题筛选器），输入 topic namespace（主题命名空间）。

   d. 选择订阅。

      MQTT 消息显示在 Subscriptions（订阅）和 Topic（主题）表中，如下所示。这些消息可能需要长达五分钟才会显示。


Subscribe to a topic
Publish to a topic
Topic name
The topic name identifies the message. The message payload will be published to this topic with a Quality of S
topic/test/
Message payload
Additional configuration
Publish
Subscriptions
topic/test/
topic/test/
topic/test/
[
{
"timestamp": 1673520691123,
"message": "Test message 1"
},
{
"timestamp": 1673520692321,
"message": "Test message 2"
},
{
"timestamp": 1673520693322,
"message": "Test message 3"
}
]

查看日志数据

在 “日志” 中查看您的 CloudWatch 日志记录

1. 打开 AWS Management Console，然后导航至CloudWatch。
2. 在导航栏上，依次选择 Logs（日志）和 Logs Insights（日志见解）。
3. 在选择日志组菜单上，选择您在 AWS IoT 规则中指定的日志组。
4. 在 Logs insights（日志见解）页面上，选择 Run query（运行查询）。

使用记录 AWS IoT API 调用 AWS CloudTrail

AWS IoT 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。 CloudTrail 将所有 API 调用捕获 AWS IoT 为事件，包括来自 AWS IoT 控制台的调用和对 AWS IoT API 的代码调用。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。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、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 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，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。您可以在中查看、搜索和下载最近发生的事件 AWS 账户。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

对于 AWS 账户中的事件的持续记录（包括 AWS IoT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，当您在控制台中创建跟踪时，该跟踪将应用于所有 AWS 区域跟踪。跟踪记录 AWS 分区中所有 AWS 区域事件并将日志文件传送到您指定的 Amazon S3 存储桶。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：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail日志文件

> **Note**
>
> AWS IoT 不记录数据平面操作（设备端） CloudTrail。 CloudWatch 用于监视这些操作。

一般而言，进行更改的 AWS IoT 控制平面操作由记录 CloudTrail。诸如CreateThingCreateKeysAndCertificate、和之类的呼叫UpdateCertificate留下 CloudTrail 条目，而诸如ListThings和ListTopic规则之类的呼叫则不是。

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 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。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅CloudTrail 用户身份元素。

AWS IoT 操作记录在 AWS IoT API 参考中。 AWS IoT 无线操作记录在《无AWS IoT 线 API 参考》中。

了解 AWS IoT 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顺序出现。

以下示例显示了演示该`AttachPolicy`操作的 CloudTrail 日志条目。

```
{
    "timestamp":"1460159496",
    "AdditionalEventData":"",
    "Annotation":"",
    "ApiVersion":"",
    "ErrorCode":"",
    "ErrorMessage":"",
    "EventID":"8bff4fed-c229-4d2d-8264-4ab28a487505",
    "EventName":"AttachPolicy",
    "EventTime":"2016-04-08T23:51:36Z",
```

```
    "EventType":"AwsApiCall",
    "ReadOnly":"",
    "RecipientAccountList":"",
    "RequestID":"d4875df2-fde4-11e5-b829-23bf9b56cbcd",
    "RequestParamters":{
        "principal":"arn:aws:iot:us-
east-1:123456789012:cert/528ce36e8047f6a75ee51ab7beddb4eb268ad41d2ea881a10b67e8e76924d894",
        "policyName":"ExamplePolicyForIoT"
    },
    "Resources":"",
    "ResponseElements":"",
    "SourceIpAddress":"52.90.213.26",
    "UserAgent":"aws-internal/3",
    "UserIdentity":{
        "type":"AssumedRole",
        "principalId":"AKIAI44QH8DHBEXAMPLE",
        "arn":"arn:aws:sts::12345678912:assumed-role/iotmonitor-us-east-1-beta-
InstanceRole-1C5T1YCYMHPYT/i-35d0a4b6",
        "accountId":"222222222222",
        "accessKeyId":"access-key-id",
        "sessionContext":{
            "attributes":{
                "mfaAuthenticated":"false",
                "creationDate":"Fri Apr 08 23:51:10 UTC 2016"
            },
            "sessionIssuer":{
                "type":"Role",
                "principalId":"AKIAI44QH8DHBEXAMPLE",
                "arn":"arn:aws:iam::123456789012:role/executionServiceEC2Role/
iotmonitor-us-east-1-beta-InstanceRole-1C5T1YCYMHPYT",
                "accountId":"222222222222",
                "userName":"iotmonitor-us-east-1-InstanceRole-1C5T1YCYMHPYT"
            }
        },
        "invokedBy":{
            "serviceAccountId":"111111111111"
        }
    },
    "VpcEndpointId":""
}
```

的规则 AWS IoT

规则使您的设备能够与之交互 AWS 服务。基于 MQTT 主题流分析规则并执行操作。您可以使用规则来支持以下任务：

- 补充或筛选从设备接收的数据。
- 将从设备接收的数据写入 Amazon DynamoDB 数据库。
- 将文件保存到 Amazon S3。
- 向使用 Amazon SNS 的所有用户发送推送通知。
- 将数据发布到 Amazon SQS 队列。
- 调用 Lambda 函数来提取数据。
- 使用 Amazon Kinesis 处理来自大量设备的消息。
- 将数据发送到亚马逊 OpenSearch 服务。
- 捕获 CloudWatch 指标。
- 更改 CloudWatch 警报。
- 将 MQTT 消息中的数据发送到 Amazon， SageMaker 以便根据机器学习 (ML) 模型进行预测。
- 向 Salesforce IoT 输入流发送消息。
- 向 AWS IoT Analytics 频道发送消息数据。
- 开始 Step Functions 状态机的过程。
- 将消息数据发送到 AWS IoT Events 输入。
- 将消息数据发送到 AWS IoT SiteWise中的资产属性。
- 将消息数据发送到 Web 应用程序或服务。

您的规则可以使用通过the section called “设备通信协议”支持的发布/订阅协议传递的 MQTT 消息。您还可以使用 B asic Ingest 功能将设备数据安全地发送到之前 AWS 服务 列出的设备，而不会产生消息收发费用。基本摄取特征通过从摄取路径中删除发布/订阅消息代理来优化数据流。这使其具有成本效益，同时仍保留了的安全性和数据处理功能 AWS IoT。

在执行这些操作之前 AWS IoT ，您必须向其授予代表您访问 AWS 资源的权限。执行操作时，您需要为所使用的操作支付标准费用。 AWS 服务

内容

- 传递角色权限
- 创建规则
- 查看您的规则
- 删除规则
- AWS IoT 规则动作
- 排查规则问题
- 使用规则访问跨账户资源 AWS IoT
- 错误处理（错误操作）
- 借助基本摄取功能，降低消息收发成本
- AWS IoT SQL 参考

授予 AWS IoT 规则所需的访问权限

使用 IAM 角色控制每条规则有权访问的 AWS 资源。在创建规则之前，您必须使用允许访问所需 AWS 资源的策略创建 IAM 角色。 AWS IoT 在实施规则时扮演这个角色。

完成以下步骤以创建 IAM 角色和 AWS IoT 策略，以授予 AWS IoT 规则所需的访问权限 (AWS CLI)。

1. 将以下授予角色代入 AWS IoT 权限的信任策略文档保存到名为的文件中`iot-role-trust.json`。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "123456789012"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:iot:us-east-1:123456789012:rule/
rulename"
                }
```

```
            }
        }
    ]
}
```

使用 create-role 命令，指定 iot-role-trust.json 文件，创建 IAM 角色：

```
aws iam create-role --role-name my-iot-role --assume-role-policy-document
 file://iot-role-trust.json
```

此命令的输出如下所示：

```
{
 "Role": {
  "AssumeRolePolicyDocument": "url-encoded-json",
  "RoleId": "AKIAIOSFODNN7EXAMPLE",
  "CreateDate": "2015-09-30T18:43:32.821Z",
  "RoleName": "my-iot-role",
  "Path": "/",
  "Arn": "arn:aws:iam::123456789012:role/my-iot-role"
 }
}
```

2. 将以下 JSON 保存到名为 my-iot-policy.json 的文件中。

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

此 JSON 是一个示例策略文档，用于向 AWS IoT 管理员授予对 DynamoDB 的访问权限。

在担任角色后，使用 create- p AWS IoT olicy 命令授予对 AWS 资源的访问权限，并传入my-iot-policy.json文件：

```
aws iam create-policy --policy-name my-iot-policy --policy-document file://my-iot-policy.json
```

有关如何在的策略 AWS 服务 中授予访问权限的更多信息 AWS IoT，请参阅创建规则。

create-policy 命令的输出中包含该策略的 ARN。将策略附加到角色。

```
{
 "Policy": {
  "PolicyName": "my-iot-policy",
  "CreateDate": "2015-09-30T19:31:18.620Z",
  "AttachmentCount": 0,
  "IsAttachable": true,
  "PolicyId": "ZXR6A36LTYANPAI7NJ5UV",
  "DefaultVersionId": "v1",
  "Path": "/",
  "Arn": "arn:aws:iam::123456789012:policy/my-iot-policy",
  "UpdateDate": "2015-09-30T19:31:18.620Z"
 }
}
```

3. 使用 attach-role-policy 命令将您的策略附加到角色：

```
aws iam attach-role-policy --role-name my-iot-role --policy-arn
 "arn:aws:iam::123456789012:policy/my-iot-policy"
```

传递角色权限

规则定义的一部分是 IAM 角色，该角色授予对规则操作中指定的资源的访问权限。当调用规则的操作时，规则引擎会代入该角色。角色的定义必须与规则 AWS 账户 相同。

在创建或替换规则时，您实际上将角色提交到规则引擎中。无需 iam:PassRole 权限即可执行该操作。为验证您拥有该权限，您需要创建一个策略以便授予 iam:PassRole 权限，并将其附加到您的 IAM 用户。以下策略介绍了如何向角色提供 iam:PassRole 权限。

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

```
    "Sid": "Stmt1",
    "Effect": "Allow",
    "Action": [
     "iam:PassRole"
    ],
    "Resource": [
     "arn:aws:iam::123456789012:role/myRole"
    ]
   }
 ]
}
```

在此策略示例中，将 `iam:PassRole` 权限授予了角色 `myRole`。该角色使用角色的 ARN 指定。将此策略附加到您的 IAM 用户或用户所属的角色。有关更多信息，请参阅使用管理的策略。

> **Note**
>
> Lambda 函数使用基于资源的策略，该策略直接附加至 Lambda 函数本身。在您创建调用 Lambda 函数的规则时，您未传递角色，因此创建规则的用户无需 `iam:PassRole` 权限。有关 Lambda 函数授权的更多信息，请参阅使用资源策略授予权限。

创建规则

您可以创建 AWS IoT 规则来路由来自互联事物的数据，以便与其他 AWS 服务进行交互。 AWS IoT 规则由以下部分组成：

规则的组成部分

组件	描述	必需/可选
Rule name（规则名称）	规则的名称。请注意，我们不建议在您的规则名称中使用个人身份信息。	必需。
规则描述	规则的文字描述。请注意，我们不建议在您的规则描述中使用个人身份信息。	可选。

组件	描述	必需/可选
SQL 语句	一种简化的 SQL 语法，用于筛选接收的 MQTT 主题相关消息并向其它推送数据。有关更多信息，请参阅 AWS IoT SQL 参考。	必需。
SQL 版本	评估规则时使用的 SQL 规则引擎的版本。尽管该属性是可选的，但我们强烈建议您指定 SQL 版本。默认情况下， AWS IoT Core 控制台将此属性设置`2016-03-23` 为。如果未设置此属性，例如在 AWS CLI 命令或 AWS CloudFormation 模板中，`2015-10-08` 则使用。有关更多信息，请参阅 SQL 版本。	必需。
一个或多个操作	这些操作是在制定规则时 AWS IoT 执行的。例如，您可以将数据插入 DynamoDB 表、将数据写入 Amazon S3 存储桶、发布至 Amazon SNS 主题或调用 Lambda 函数。	必需。
错误操作	该操作 AWS IoT 在无法执行规则的操作时执行。	可选。

在创建 AWS IoT 规则之前，您必须使用允许访问所需 AWS 资源的策略创建 IAM 角色。 AWS IoT 在实施规则时扮演这个角色。有关更多信息，请参阅授予 AWS IoT 规则所需的访问权限和传递角色权限。

当您创建规则时，请注意在主题上发布的数据量。如果您创建的规则包含通配符主题模式，则这些规则可能会与您的大部分消息匹配。在这种情况下，您可能需要增加目标操作使用的 AWS 资源的容量。另外，如果您创建包含通配符主题模式的重新发布规则，最终可能获得一个造成无限循环的循环规则。

> **Note**
>
> 创建和更新规则是管理员级操作。有权创建或更新规则的所有用户都能够访问规则处理的数据。

创建规则（控制台）

创建规则 (AWS Management Console)

使用AWS Management Console命令创建规则：

1. 打开AWS IoT 控制台。
2. 在左侧导航栏中，从 “管理” 部分选择 “消息路由”。然后选择 “规则”。
3. 在 规则 页面，选择 创建规则。
4. 在 “指定规则属性” 页面上，输入规则的名称。规则描述和标签是可选的。选择下一步。
5. 在 “配置 SQL 语句” 页上，选择 SQL 版本并输入 SQL 语句。SQL 语句的示例可以是`SELECT temperature FROM 'iot/topic' WHERE temperature > 50`。有关更多信息，请参阅SQL 版本和 AWS IoT SQL 参考。
6. 在附加规则操作页面上，添加规则操作以将数据路由到其他 AWS 服务。
    1. 在规则操作中，从下拉列表中选择一个规则操作。例如，你可以选择 Kinesis Stre am。有关规则操作的更多信息，请参阅AWS IoT 规则操作。
    2. 根据您选择的规则操作，输入相关的配置详细信息。例如，如果您选择 Kinesis Stream，则需要选择或创建数据流资源，并可以选择输入配置详细信息，例如分区密钥，用于在 Steam 中按分片对数据进行分组。
    3. 在 IAM 角色中，选择或创建角色以授予对终端节点的 AWS IoT 访问权限。请注意，这 AWS IoT 将在您选择的 IAM 角色`aws-iot-rule`下自动创建一个前缀为的策略。您可以选择 “查看”，从 IAM 控制台中查看您的 IAM 角色和策略。错误操作是可选的。您可以在错误处理（错误操作）中找到更多信息。有关为您的规则创建 IAM 角色的更多信息，请参阅授予规则所需的访问权限。选择下一步。
7. 在 “查看并创建” 页面上，查看所有配置并在需要时进行编辑。选择创建。

成功创建规则后，您将在规则页面上看到该规则列出。您可以选择规则以打开 “详细信息” 页面，在该页面中可以查看规则、编辑规则、停用规则和删除规则。

创建规则 (CLI)

创建规则 (AWS CLI)

使用 create-topic-rule 命令创建规则：

```
aws iot create-topic-rule --rule-name myrule --topic-rule-payload file://myrule.json
```

下面是一个负载文件示例，其中包含将发送至 iot/test 主题的所有消息插入指定 DynamoDB 表的规则。SQL 语句筛选消息，ARN 角色授予写入 DynamoDB 表的 AWS IoT 权限。

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "dynamoDB": {
    "tableName": "my-dynamodb-table",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
    "hashKeyField": "topic",
    "hashKeyValue": "${topic(2)}",
    "rangeKeyField": "timestamp",
    "rangeKeyValue": "${timestamp()}"
   }
  }
 ]
}
```

下面是一个负载文件示例，其中包含将发送至 iot/test 主题的所有消息插入指定 S3 存储桶的规则。SQL 语句会筛选消息，而角色 ARN 则授予写入 Amazon S3 存储桶的 AWS IoT 权限。

```
{
 "awsIotSqlVersion": "2016-03-23",
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "actions": [
  {
   "s3": {
    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",
    "bucketName": "my-bucket",
    "key": "myS3Key"
   }
  }
 ]
}
```

以下是载荷文件示例，其中包含将数据推送到 Amazon S OpenSearch ervice 的规则：

```
{
```

```
 "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "OpenSearch": {
    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es",
    "endpoint": "https://my-endpoint",
    "index": "my-index",
    "type": "my-type",
    "id": "${newuuid()}"
   }
  }
 ]
}
```

下面的负载文件示例包含调用 Lambda 函数的规则：

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "lambda": {
    "functionArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function"
   }
  }
 ]
}
```

下面的负载文件示例包含发布至 Amazon SNS 主题的规则：

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "sns": {
    "targetArn": "arn:aws:sns:us-west-2:123456789012:my-sns-topic",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
```

```
  }
 ]
}
```

下面的负载文件示例包含在不同 MQTT 主题上重新发布的规则：

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "republish": {
    "topic": "my-mqtt-topic",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
 ]
}
```

以下是负载文件示例，其中包含将数据推送到 Amazon Data Firehose 流的规则：

```
{
 "sql": "SELECT * FROM 'my-topic'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "firehose": {
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
    "deliveryStreamName": "my-stream-name"
   }
  }
 ]
}
```

以下是负载文件示例，其中包含一条规则，如果 MQTT 负载中的数据被归类为 1，则使用 Amazon SageMaker `machinelearning_predict` 函数重新发布到主题。

```
{
 "sql": "SELECT * FROM 'iot/test' where machinelearning_predict('my-model',
 'arn:aws:iam::123456789012:role/my-iot-aml-role', *).predictedLabel=1",
```

```
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "republish": {
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role",
    "topic": "my-mqtt-topic"
   }
  }
 ]
}
```

下面是一个示例负载文件，该文件包含将消息发布到 Salesforce IoT Cloud 输入流的规则。

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "salesforce": {
    "token": "ABCDEFGHI123456789abcdefghi123456789",
    "url": "https://ingestion-cluster-id.my-env.sfdcnow.com/streams/stream-id/
connection-id/my-event"
   }
  }
 ]
}
```

下面是一个负载文件示例，其中包含一个可以开始执行 Step Functions 状态机的规则。

```
{
 "sql": "expression",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "stepFunctions": {
    "stateMachineName": "myCoolStateMachine",
    "executionNamePrefix": "coolRunning",
    "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
   }
  }
```

```
 ]
}
```

标记规则

要为您的新规则或现有规则添加另一层特殊性，您可以应用标记。标记利用规则中的键值对，使您能够更好地控制将规则应用于资源和服务的方式和位置。 AWS IoT 例如，您可以将规则的范围限制为仅应用于发布前测试的测试版环境（`Key=environment, Value=beta`），或者仅捕获从特定端点发送到 `iot/test` 主题的所有消息并将其存储在 Amazon S3 桶中。

IAM 策略示例

有关显示如何为规则授予标记权限的示例，请考虑这样一位用户：该用户运行以下命令来创建规则，并将此规则标记为仅应用于其测试版环境。

在此示例中：

- *MyTopicRuleName*加上规则的名称。
- 将 *myrule.json* 替换为策略文档的名称。

```
aws iot create-topic-rule
    --rule-name MyTopicRuleName
    --topic-rule-payload file://myrule.json
    --tags "environment=beta"
```

对于本示例，您必须使用以下 IAM 策略：

```
{
    "Version": "2012-10-17",
    "Statement":
    {
        "Action": [ "iot:CreateTopicRule", "iot:TagResource" ],
        "Effect": "Allow",
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:rule/MyTopicRuleName"
        ]
    }
}
```

上面的示例显示了新创建的名为 MyTopicRuleName 的规则，该规则仅应用于您的测试版环境。策略声明中用 MyTopicRuleName 特别标注的 iot:TagResource 允许在创建或更新 MyTopicRuleName 时进行标记。创建规则时使用的参数 --tags "environment=beta" 将 MyTopicRuleName 的范围限制为仅您的测试版环境。如果删除参数 --tags "environment=beta"，则 MyTopicRuleName 将应用于所有环境。

有关创建特定于 AWS IoT 规则的 IAM 角色和策略的更多信息，请参阅授予 AWS IoT 规则所需的访问权限。

有关标记实例的一般信息，请参阅为资源添加 AWS IoT 标签。

查看您的规则

使用 list-topic-rules 命令列出您的规则：

```
aws iot list-topic-rules
```

使用 get-topic-rule 命令获取有关规则的信息：

```
aws iot get-topic-rule --rule-name myrule
```

删除规则

用完规则后可以将其删除。

删除规则 (AWS CLI)

使用 delete-topic-rule 命令删除规则：

```
aws iot delete-topic-rule --rule-name myrule
```

AWS IoT 规则动作

AWS IoT 规则操作指定调用规则时要执行的操作。您可以定义向亚马逊 DynamoDB 数据库发送数据、向 Amazon Kinesis Data Streams 发送数据、 AWS Lambda 调用函数等的操作。 AWS IoT AWS 区域在操作的服务可用时，支持以下操作。

规则操作	描述	API 中的名称
Apache Kafka	将消息发送到 Apache Kafka 集群。	`kafka`
CloudWatch 警报	更改 Amazon CloudWatch 警报的状态。	`cloudwatchAlarm`
CloudWatch 日志	向 Amazon CloudWatch Logs 发送消息。	`cloudwatchLogs`
CloudWatch 指标	向 CloudWatch 指标发送消息。	`cloudwatchMetric`
DynamoDB	将消息发送到 DynamoDB 表。	`dynamoDB`
DynamoDBv2	将消息数据发送到 DynamoDB 表中的多列。	`dynamoDBv2`
Elasticsearch	向 OpenSearch 终端节点发送消息。	`OpenSearch`
HTTP	将消息发布到 HTTPS 端点。	`http`
IoT Analytics	向 AWS IoT Analytics 频道发送消息。	`iotAnalytics`
AWS IoT Events	向 AWS IoT Events 输入发送消息。	`iotEvents`
AWS IoT SiteWise	向 AWS IoT SiteWise 资产属性发送消息数据。	`iotSiteWise`
Firehose	向 Firehose 传送流发送消息。	`firehose`
Kinesis Data Streams	将消息发送到 Kinesis 数据流。	`kinesis`
Lambda	使用消息数据作为输入调用 Lambda 函数。	`lambda`

规则操作	描述	API 中的名称
位置	向 Amazon Location Service 发送位置数据。	`location`
OpenSearch	向亚马逊 OpenSearch 服务终端节点发送消息。	`OpenSearch`
Republish	在另一个 MQTT 主题上重新发布消息。	`republish`
S3	将消息存储在 Amazon Simple Storage Service (Amazon S3) 存储桶中。	`s3`
Salesforce IoT	将消息发送到 Salesforce IoT 输入流。	`salesforce`
SNS	将消息发布为 Amazon Simple Notification Service (Amazon SNS) 推送通知。	`sns`
SQS	将消息发送到 Amazon Simple Queue Service (Amazon SQS) 队列	`sqs`
Step Functions	启动 AWS Step Functions 状态机。	`stepFunctions`
the section called “Timestream”	将消息发送到 Amazon Timestream 数据库表。	`timestream`

注意

- 将规则定义 AWS 区域 为与其他服务的资源相同，以便规则操作可以与该资源进行交互。
- 如果出现间歇性错误， AWS IoT 规则引擎可能会多次尝试执行某项操作。如果所有尝试都失败，则该消息将被丢弃，错误将在您的 CloudWatch 日志中显示。您可以为在发生故障后调用的每条规则指定一个错误操作。有关更多信息，请参阅 错误处理（错误操作）。

- 某些规则操作会激活与 AWS Key Management Service （AWS KMS）集成的服务中的操作，以支持静态数据加密。如果您使用客户管理 AWS KMS key （KMS 密钥）对静态数据进行加密，则服务必须有权代表呼叫者使用 KMS 密钥。要了解如何管理客户自主管理型 KMS 密钥的权限，请参阅相应服务指南中的数据加密主题。有关客户自主管理型 KMS 密钥的更多信息，请参阅《AWS Key Management Service 开发人员指南》中的 AWS Key Management Service 概念。

Apache Kafka

Apache Kafka (Kafka) 操作将消息直接发送到你的亚马逊托管流媒体for Apache Kafka（亚马逊 MSK）、由第三方提供商（例如 Confluent Cloud）管理的 Apache Kafka 集群或用于数据分析和可视化的自我管理的 Apache Kafka 集群。

> **Note**
>
> 本主题假设您熟悉 Apache Kafka 平台及相关概念。有关 Apache Kafka 的更多信息，请参阅 Apache Kafka。不@@ 支持 MSK 无服务器。MSK 无服务器集群只能通过 IAM 身份验证完成，而 Apache Kafka 规则操作目前不支持该身份验证。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`ec2:CreateNetworkInterface`、、、、、`ec2:DescribeNetworkInterfaces`和`ec2:DescribeSecurityGroups`操作的 IAM 角色。此角色创建并管理您的 Amazon Virtual Private Cloud 弹性网络接口，以便联系您的 Kafka 代理。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT Core 允许执行此规则操作的角色。

  有关网络接口的更多信息，请参阅《Amazon EC2 用户指南》中的弹性网络接口。

  附加到您指定的角色的策略应如以下示例所示：

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "ec2:CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:CreateNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:DescribeSubnets",
            "ec2:DescribeVpcs",
            "ec2:DescribeVpcAttribute",
            "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        }
    ]
}
```

- 如果您使用 AWS Secrets Manager 存储连接到 Kafka 代理所需的证书，则必须创建一个 AWS IoT Core 可以代入执行`secretsmanager:GetSecretValue`和`secretsmanager:DescribeSecret`操作的 IAM 角色。

  附加到您指定的角色的策略应如以下示例所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetSecretValue",
            "secretsmanager:DescribeSecret"
            ],
            "Resource": [

 "arn:aws:secretsmanager:region:123456789012:secret:kafka_client_truststore-*",
                "arn:aws:secretsmanager:region:123456789012:secret:kafka_keytab-*"
            ]
        }
    ]
}
```

- 您可以在 Amazon Virtual Private Cloud (Amazon VPC) 中运行您的 Apache Kafka 集群。您必须创建 Amazon VPC 目标并在子网中使用 NAT 网关将消息从 AWS IoT 转发到公有 Kafka 集群。 AWS IoT 规则引擎会在 VPC 目标中列出的每个子网中创建一个网络接口，以将流量直接路由到 VPC。当您创建 VPC 目标时， AWS IoT 规则引擎会自动创建 VPC 规则操作。有关 VPC 规则操作的更多信息，请参阅 Virtual Private Cloud (VPC) 目标。
- 如果您使用客户托管 AWS KMS key （KMS 密钥）加密静态数据，则服务必须有权代表呼叫者使用 KMS 密钥。有关更多信息，请参阅 Amazon Managed Streaming for Apache Kafka 开发人员指南中的 Amazon MSK 加密。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

destinationArn

VPC 目标的 Amazon Resource Name (ARN)。有关如何创建 VPC 目标的更多信息，请参阅 Virtual Private Cloud (VPC) 目标。

topic

要发送到 Kafka 代理的消息的 Kafka 主题。

您可以使用替代模板替换此字段。有关更多信息，请参阅 the section called “替换模板”。

键（可选）

Kafka 消息键。

您可以使用替代模板替换此字段。有关更多信息，请参阅 the section called “替换模板”。

标头（可选）

您指定的 Kafka 标头的列表。每个标头都是一个键/值对，您可以在创建 Kafka 操作时指定该键值对。您可以使用这些标头将数据从 IoT 客户端路由到下游 Kafka 集群，而无需修改消息有效负载。

您可以使用替代模板替换此字段。要了解如何在 Kafka 操作的标头中将内联规则的函数作为替换模板传递，请参阅示例。有关更多信息，请参阅 the section called “替换模板”。

> **Note**
>
> 不支持二进制格式的标头。

分区（可选）

Kafka 消息分区。

您可以使用替代模板替换此字段。有关更多信息，请参阅 the section called “替换模板”。

clientProperties

定义 Apache Kafka 生成器客户端属性的对象。

acks（可选）

生成器在考虑请求完成之前要求服务器收到的确认数。

如果指定 0 作为值，则生产者将不会等待来自服务器的任何确认。如果服务器没有收到该消息，则生成器将不会重试发送该消息。

有效值：`-1`、`0`、`1`、`all`。默认值为 `1`。

bootstrap.servers

主机和端口对列表（例如 `host1:port1`、`host2:port2`）用于建立到 Kafka 集群的初始连接。

compression.type（可选）

生成器生成的所有数据的压缩类型。

有效值：`none`、`gzip`、`snappy`、`lz4`、`zstd`。默认值为 `none`。

security.protocol

用于连接到您的 Kafka 代理的安全协议。

有效值：`SSL`、`SASL_SSL`。默认值为 `SSL`。

key.serializer

指定如何将您使用 `ProducerRecord` 提供的键对象转换为字节。

有效值：`StringSerializer`。

value.serializer

指定如何将您使用 `ProducerRecord` 提供的值对象转换为字节。

有效值：`ByteBufferSerializer`。

ssl.truststore

base64 格式的信任库文件或位于 AWS Secrets Manager 的信任库文件位置。如果您的信任存储受到 Amazon 证书授权机构（CA）的认证，则不需要此值。

此字段支持替换模板。如果使用 Secrets Manager 存储连接到 Kafka 代理所需的凭证，则可以使用 `get_secret` SQL 函数检索此字段的值。有关替换模板的更多信息，请参阅 the section called “替换模板”。有关 `get_secret` SQL 函数的更多信息，请参阅the section called “get_secret(secretId, secretType, key, roleArn)”。如果信任库采用文件的形式，请使用 `SecretBinary` 参数。如果信任库采用字符串的形式，请使用 `SecretString` 参数。

此值最大为 65 KB。

ssl.truststore.password

信任库存储的密码。仅当您为信任库创建了密码时，才需要此值。

ssl.keystore

密钥库文件。当您指定 SSL 作为 `security.protocol` 的值时，才需要此值。

此字段支持替换模板。使用 Secrets Manager 来存储连接到您的 Kafka 代理所需的凭证。要检索此字段的值，请使用 `get_secret` SQL 函数。有关替换模板的更多信息，请参阅 the section called “替换模板”。有关 `get_secret` SQL 函数的更多信息，请参阅the section called “get_secret(secretId, secretType, key, roleArn)”。使用 `SecretBinary` 参数。

ssl.keystore.password

密钥库文件存储的密码。如果为 `ssl.keystore` 指定了值，则需要此值。

此字段的值可以是纯文本。此字段还支持替代模板。使用 Secrets Manager 来存储连接到您的 Kafka 代理所需的凭证。要检索此字段的值，请使用 `get_secret` SQL 函数。有关替换模板的更多信息，请参阅 the section called “替换模板”。有关 `get_secret` SQL 函数的更多信息，请参阅the section called “get_secret(secretId, secretType, key, roleArn)”。使用 `SecretString` 参数。

ssl.key.password

密钥库文件中私有密钥的密码。

此字段支持替换模板。使用 Secrets Manager 来存储连接到您的 Kafka 代理所需的凭证。要检索此字段的值，请使用 `get_secret` SQL 函数。有关替换模板的更多信息，请参阅 the

section called “替换模板”。有关 get_secret SQL 函数的更多信息，请参阅the section called “get_secret(secretId, secretType, key, roleArn)”。使用 SecretString 参数。

sasl.mechanism

用于连接到您的 Kafka 代理的安全机制。当您为 security.protocol 指定 SASL_SSL 时，则需要此值。

有效值：PLAIN、SCRAM-SHA-512、GSSAPI。

> **Note**
>
> SCRAM-SHA-512是 cn-north-1、cn-northwest-1、-1 和 -1 区域中唯一支持的安全机制。 us-gov-east us-gov-west

sasl.plain.username

用于从 Secrets Manager 中检索密钥字符串的用户名。当您为 security.protocol 指定 SASL_SSL、为 sasl.mechanism 指定 PLAIN 时，则需要此值。

sasl.plain.password

用于从 Secrets Manager 中检索密钥字符串的密码。当您为 security.protocol 指定 SASL_SSL、为 sasl.mechanism 指定 PLAIN 时，则需要此值。

sasl.scram.username

用于从 Secrets Manager 中检索密钥字符串的用户名。当您为 security.protocol 指定 SASL_SSL、为 sasl.mechanism 指定 SCRAM-SHA-512 时，则需要此值。

sasl.scram.password

用于从 Secrets Manager 中检索密钥字符串的密码。当您为 security.protocol 指定 SASL_SSL、为 sasl.mechanism 指定 SCRAM-SHA-512 时，则需要此值。

sasl.kerberos.keytab

Secrets Manager 中用于 Kerberos 身份验证的 keytab 文件。当您为 security.protocol 指定 SASL_SSL、为 sasl.mechanism 指定 GSSAPI 时，则需要此值。

此字段支持替换模板。使用 Secrets Manager 来存储连接到您的 Kafka 代理所需的凭证。要检索此字段的值，请使用 get_secret SQL 函数。有关替换模板的更多信息，请参阅 the

section called “替换模板”。有关 get_secret SQL 函数的更多信息，请参阅the section called “get_secret(secretId, secretType, key, roleArn)”。使用 SecretBinary 参数。

sasl.kerberos.service.name

Apache Kafka 运行的 Kerberos 主要名称。当您为 security.protocol 指定 SASL_SSL、为 sasl.mechanism 指定 GSSAPI 时，则需要此值。

sasl.kerberos.krb5.kdc

您的 Apache Kafka 生成器客户端连接到的密钥分配中心 (KDC) 的主机名。当您为 security.protocol 指定 SASL_SSL、为 sasl.mechanism 指定 GSSAPI 时，则需要此值。

sasl.kerberos.krb5.realm

您的 Apache Kafka 生成器客户端连接到的领域。当您为 security.protocol 指定 SASL_SSL、为 sasl.mechanism 指定 GSSAPI 时，则需要此值。

sasl.kerberos.principal

Kerberos 可以为其分配票证以访问 Kerberos 感知服务的唯一 Kerberos 身份。当您为 security.protocol 指定 SASL_SSL、为 sasl.mechanism 指定 GSSAPI 时，则需要此值。

示例

以下 JSON 示例在规则中定义了 Apache Kafka 操作。 AWS IoT 以下示例将 sourceIp() 内联函数作为替换模板传递到 Kafka 操作标头中。

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "kafka": {
     "destinationArn": "arn:aws:iot:region:123456789012:ruledestination/vpc/
VPCDestinationARN",
     "topic": "TopicName",
     "clientProperties": {
      "bootstrap.servers": "kafka.com:9092",
```

```
      "security.protocol": "SASL_SSL",
      "ssl.truststore": "${get_secret('kafka_client_truststore',
 'SecretBinary','arn:aws:iam::123456789012:role/kafka-get-secret-role-name')}",
      "ssl.truststore.password": "kafka password",
      "sasl.mechanism": "GSSAPI",
      "sasl.kerberos.service.name": "kafka",
      "sasl.kerberos.krb5.kdc": "kerberosdns.com",
      "sasl.kerberos.keytab": "${get_secret('kafka_keytab','SecretBinary',
 'arn:aws:iam::123456789012:role/kafka-get-secret-role-name')}",
      "sasl.kerberos.krb5.realm": "KERBEROSREALM",
      "sasl.kerberos.principal": "kafka-keytab/kafka-keytab.com"
     },
     "headers": [
      {
       "key": "static_header_key",
       "value": "static_header_value"
      },
      {
       "key": "substitutable_header_key",
       "value": "${value_from_payload}"
      },
      {
       "key": "source_ip",
       "value": "${sourceIp()}"
      }
     ]
    }
   }
  ]
 }
}
```

有关 Kerberos 设置的重要注意事项

- 您的密钥分配中心 (KDC) 必须通过目标 VPC 内的私有域名系统 (DNS) 进行解析。一种可能的方法是将 KDC DNS 条目添加到私有托管区域。有关此方法的更多信息，请参阅如何使用私有托管区域。
- 每个 VPC 都必须启用 DNS 解析。有关更多信息，请参阅将 DNS 与您的 VPC 一起使用。
- VPC 目标中的网络接口安全组和实例级安全组必须允许来自 VPC 内部以下端口的流量。
  - 引导代理侦听器端口上的 TCP 流量（通常为 9092，但必须在 9000–9100 范围内）
  - KDC 端口 88 上的 TCP 和 UDP 流量

- SCRAM-SHA-512是 cn-north-1、cn-northwest-1、-1 和 -1 区域中唯一支持的安全机制。 us-gov-east us-gov-west

Virtual Private Cloud (VPC) 目标

Apache Kafka 规则操作将数据路由到 Amazon Virtual Private Cloud (Amazon VPC) 中的 Apache Kafka 集群。当您为规则操作指定 VPC 目标时，将自动启用 Apache Kafka 规则操作使用的 VPC 配置。

VPC 目标包含 VPC 内部的子网列表。规则引擎将在您在此列表中指定的每个子网中创建一个弹性网络接口。有关网络接口的更多信息，请参阅 Amazon EC2 用户指南中的弹性网络接口。

要求和注意事项

- 如果您使用的是自我管理的 Apache Kafka 集群，则将在互联网上使用公有终端节点访问此集群：
  - 为子网中的实例创建 NAT 网关。NAT 网关具有可以连接到互联网的公有 IP 地址，这允许规则引擎将您的消息转发到公有 Kafka 集群。
  - 使用由 VPC 目标创建的弹性网络接口（ENI）分配弹性 IP 地址。必须将您使用的安全组配置为阻止传入流量。

  > **Note**
  >
  > 如果 VPC 目标被禁用，然后重新启用，则必须将弹性 IP 与新的 ENI 重新关联。

- 如果 VPC 主题规则目标连续30天未收到任何流量，该目标将被禁用。
- 如果 VPC 目标使用的任何资源发生变化，目标将被禁用且无法使用。
- 可以禁用 VPC 目标的一些更改包括：删除 VPC、子网、安全组或使用的角色；将角色修改为不再拥有必要的权限；以及禁用目标。

定价

出于定价目的，除了在资源位于您的 VPC 中时向资源发送消息的操作之外，还会计量 VPC 规则操作。有关定价信息，请参阅 AWS IoT Core 定价。

创建 Virtual Private Cloud (VPC) 主题规则目标

您可以使用 CreateTopicRuleDestinationAPI 或 AWS IoT Core 控制台创建虚拟私有云 (VPC) 目标。

当您创建 VPC 目标时，必须指定以下信息。

vpcId

VPC 目标的唯一 ID。

subnetIds

规则引擎在其中创建弹性网络接口的子网列表。规则引擎为列表中的每个子网分配一个网络接口。

securityGroups（可选）

要应用到网络接口的安全组列表。

roleArn

有权代表您创建网络接口的角色的 Amazon Resource Name (ARN)。

此 ARN 应附加一个类似于以下示例的策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcs",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcAttribute",
                "ec2:DescribeSecurityGroups"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "ec2:CreateNetworkInterfacePermission",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/VPCDestinationENI": "true"
                }
            }
        },
        {
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "ec2:CreateAction": "CreateNetworkInterface",
                    "aws:RequestTag/VPCDestinationENI": "true"
                }
            }
        }
    ]
}
```

使用创建 VPC 目标 AWS CLI

以下示例显示如何使用 AWS CLI创建 VPC 目标。

```
aws --region regions iot create-topic-rule-destination --destination-configuration
 'vpcConfiguration={subnetIds=["subnet-
123456789101230456"],securityGroups=[],vpcId="vpc-
123456789101230456",roleArn="arn:aws:iam::123456789012:role/role-name"}'
```

运行此命令后，VPC 目标状态将为 IN_PROGRESS。几分钟后，其状态将更改为 ERROR（如果命令不成功）或 ENABLED。当目标状态为 ENABLED 时，目标即可使用。

您可以使用以下命令获取 VPC 目标的状态。

```
aws --region region iot get-topic-rule-destination --arn "VPCDestinationARN"
```

使用 AWS IoT Core 控制台创建 VPC 目标

以下步骤介绍如何使用 AWS IoT Core 控制台创建 VPC 目标。

1. 导航到 AWS IoT Core 控制台。在左侧窗格的操作选项卡上，选择目标。

2. 输入以下字段的值。

   - VPC ID
   - 子网 ID
   - 安全组
3. 选择具有创建网络接口所需权限的角色。前面的示例策略包含这些权限。

当 VPC 目标状态为 ENABLED（已启用）时，目标即可使用。

CloudWatch 警报

CloudWatch 警报 (`cloudWatchAlarm`) 操作会更改 Amazon CloudWatch 警报的状态。您可以在此调用中指定状态更改原因和状态值。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`cloudwatch:SetAlarmState`操作的 IAM 角色。有关更多信息，请参阅[授予 AWS IoT 规则所需的访问权限](#)。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`alarmName`

CloudWatch 警报名称。

支持[替换模板](#)：API 且 AWS CLI 仅支持

`stateReason`

警报更改的原因。

支持[替换模板](#)：是

`stateValue`

警报状态的值。有效值：`OK`、`ALARM`、`INSUFFICIENT_DATA`。

支持替换模板：是

`roleArn`

允许访问 CloudWatch 警报的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例定义了 AWS IoT 规则中的 CloudWatch 警报操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchAlarm": {
                    "alarmName": "IotAlarm",
                    "stateReason": "Temperature stabilized.",
                    "stateValue": "OK",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

另请参阅

- 什么是亚马逊 CloudWatch？ 在 Amazon CloudWatch 用户指南中
- 使用亚马逊 CloudWatch 用户指南中的亚马逊 CloudWatch警报

CloudWatch 日志

日 CloudWatch 志 (`cloudwatchLogs`) 操作将数据发送到 Amazon CloudWatch 日志。您可以使用 `batchMode` 在一条消息中上载多条设备日志记录并为其加上时间戳。您还可以指定操作要向其发送数据的日志组。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`logs:CreateLogStream`、`logs:DescribeLogStreams`和`logs:PutLogEvents`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。
- 如果您使用客户托管 AWS KMS key（KMS 密钥）加密日志中的 CloudWatch 日志数据，则该服务必须有权代表调用者使用 KMS 密钥。有关更多信息，请参阅 Amazon CloudWatch 日志用户指南 AWS KMS中的使用加密 CloudWatch 日志中的日志数据。

`batchMode` 的 MQTT 消息格式要求

如果您在`batchMode`关闭的情况下使用 CloudWatch 日志规则操作，则没有 MQTT 消息格式要求。（注意：`batchMode` 参数的默认值为 `false`。）但是，如果您在`batchMode`开启的情况下使用 CloudWatch 日志规则操作（参数值为`true`），则包含设备端日志的 MQTT 消息必须格式化为包含时间戳和消息负载。注意：`timestamp` 表示事件发生的时间，以自 1970 年 1 月 1 日 00:00:00 UTC 之后的毫秒数表示。

以下为发布格式的示例：

```
[
  {"timestamp": 1673520691093, "message": "Test message 1"},
  {"timestamp": 1673520692879, "message": "Test message 2"},
  {"timestamp": 1673520693442, "message": "Test message 3"}
]
```

根据生成设备端日志的方式，在发送这些日志之前，可能需要对其进行筛选和重新格式化，以符合此要求。有关更多信息，请参阅 MQTT 消息有效负载。

与`batchMode`参数无关，`message`内容必须符合 AWS IoT 消息大小限制。有关更多信息，请参阅 AWS IoT Core 终端节点和限额。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`logGroupName`

操作发送数据的 CloudWatch 日志组。

支持替换模板：API 且 AWS CLI 仅支持

`roleArn`

允许访问 CloudWatch 日志组的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

（可选）`batchMode`

表示是否将批量日志记录提取并上传到 CloudWatch。值包括 `true` 或 `false`（默认值）。有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例在 AWS IoT 规则中定义了 “ CloudWatch 日志” 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchLogs": {
                    "logGroupName": "IotLogs",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw",
                    "batchMode": false
                }
            }
        ]
    }
}
```

另请参阅

- 什么是 Amazon CloudWatch 日志？ 在 Amazon CloudWatch 日志用户指南中

CloudWatch 指标

CloudWatch 指标 (`cloudwatchMetric`) 操作捕获 Amazon CloudWatch 指标。您可以指定指标命名空间、名称、值、单位和时间戳。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`cloudwatch:PutMetricData`操作的 IAM 角色。有关更多信息，请参阅[授予 AWS IoT 规则所需的访问权限](#)。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`metricName`

CloudWatch 指标名称。

支持[替换模板](#)：是

`metricNamespace`

CloudWatch 指标命名空间名称。

支持[替换模板](#)：是

`metricUnit`

支持的公制单位 CloudWatch。

支持[替换模板](#)：是

`metricValue`

包含 CloudWatch 指标值的字符串。

支持[替换模板](#)：是

`metricTimestamp`

（可选）包含用 Unix 纪元时间表示的 timestamp（以秒为单位）的字符串。默认为当前 Unix 纪元时间。

支持替换模板：是

`roleArn`

允许访问 CloudWatch 指标的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例定义了 AWS IoT 规则中的 CloudWatch 指标操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchMetric": {
                    "metricName": "IotMetric",
                    "metricNamespace": "IotNamespace",
                    "metricUnit": "Count",
                    "metricValue": "1",
                    "metricTimestamp": "1456821314",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

以下 JSON 示例定义了在 AWS IoT 规则中使用替换模板的 CloudWatch 指标操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "cloudwatchMetric": {
                    "metricName": "${topic()}",
```

```
                    "metricNamespace": "${namespace}",
                    "metricUnit": "${unit}",
                    "metricValue": "${value}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw"
                }
            }
        ]
    }
}
```

另请参阅

- 什么是亚马逊 CloudWatch？ 在 Amazon CloudWatch 用户指南中
- 使用亚马逊 CloudWatch 用户指南中的亚马逊 CloudWatch指标

DynamoDB

DynamoDB (dynamoDB) 操作可将所有或部分 MQTT 消息写入 Amazon DynamoDB 表。

您可以按照教程执行操作，该教程向您说明如何使用 DynamoDB 操作创建并测试规则。有关更多信息，请参阅 教程：将设备数据存储在 DynamoDB 表中。

> **Note**
>
> 此规则将非 JSON 数据作为二进制数据写入到 DynamoDB 中。DynamoDB 控制台以 Base64 编码文本格式显示数据。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`dynamodb:PutItem`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。
- 如果您使用客户托管 AWS KMS key （KMS 密钥）对 DynamoDB 中的静态数据进行加密，则该服务必须有权代表调用方使用 KMS 密钥。有关更多信息，请参阅 Amazon DynamoDB 入门指南中的客户托管式 KMS。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`tableName`

DynamoDB 表的名称。

支持替换模板：API 且 AWS CLI 仅支持

`hashKeyField`

哈希键（也称为分区键）的名称。

支持替换模板：API 且 AWS CLI 仅支持

`hashKeyType`

（可选）哈希键（也称为分区键）的数据类型。有效值：STRING、NUMBER。

支持替换模板：API 且 AWS CLI 仅支持

`hashKeyValue`

哈希键的值。考虑使用替代模板，例如 `${topic()}` 或者 `${timestamp()}`。

支持替换模板：是

`rangeKeyField`

（可选）范围键（也称为排序键）的名称。

支持替换模板：API 且 AWS CLI 仅支持

`rangeKeyType`

（可选）范围键（也称为排序键）的数据类型。有效值：STRING、NUMBER。

支持替换模板：API 且 AWS CLI 仅支持

`rangeKeyValue`

（可选）范围键的值。考虑使用替代模板，例如 `${topic()}` 或者 `${timestamp()}`。

支持替换模板：是

`payloadField`

（可选）负载将写入的列的名称。如果省略此值，负载将写入 `payload` 列。

支持替换模板：是

`operation`

（可选）要执行的操作类型。有效值：`INSERT`、`UPDATE`、`DELETE`。

支持替换模板：是

`roleARN`

允许访问 DynamoDB 表的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

写入 DynamoDB 表的数据是规则的 SQL 语句的结果。

示例

以下 JSON 示例在规则中定义了 DynamoDB 操作。 AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * AS message FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "dynamoDB": {
                    "tableName": "my_ddb_table",
                    "hashKeyField": "key",
                    "hashKeyValue": "${topic()}",
                    "rangeKeyField": "timestamp",
                    "rangeKeyValue": "${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDB"
                }
            }
        ]
    }
}
```

另请参阅

- Amazon DynamoDB 开发人员指南中的什么是 DynamoDB？

- Amazon DynamoDB 开发人员指南中的 DynamoDB 入门
- 教程：将设备数据存储在 DynamoDB 表中

DynamoDBv2

DynamoDBv2 (dynamoDBv2) 操作可将所有或部分 MQTT 消息写入 Amazon DynamoDB 表。负载中的每个属性将写入 DynamoDB 数据库中的单独一列。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行dynamodb:PutItem操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。
- 如果要定义 MQTT 消息负载，则它必须包含一个与表的主分区键相匹配的根级键，以及一个与表的主排序键相匹配的根级键。
- 如果您使用客户托管 AWS KMS key （KMS 密钥）对 DynamoDB 中的静态数据进行加密，则该服务必须有权代表调用方使用 KMS 密钥。有关更多信息，请参阅 Amazon DynamoDB 入门指南中的客户托管式 KMS。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`putItem`

用于指定消息数据将写入的 DynamoDB 表的对象。该对象必须具备以下信息：

`tableName`

DynamoDB 表的名称。

支持替换模板：API 且 AWS CLI 仅支持

`roleARN`

允许访问 DynamoDB 表的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

写入 DynamoDB 表的数据是规则的 SQL 语句的结果。

示例

以下 JSON 示例在规则中定义了 DynamoDBv2 操作。 AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * AS message FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "dynamoDBv2": {
                    "putItem": {
                        "tableName": "my_ddb_table"
                    },
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDBv2",
                }
            }
        ]
    }
}
```

以下 JSON 示例定义了在规则中使用替换模板的 DynamoDB 操作。 AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2015-10-08",
        "actions": [
            {
                "dynamoDBv2": {
                    "putItem": {
                        "tableName": "${topic()}"
                    },
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDBv2"
                }
            }
```

```
        ]
    }
}
```

另请参阅

- Amazon DynamoDB 开发人员指南中的什么是 DynamoDB？
- Amazon DynamoDB 开发人员指南中的 DynamoDB 入门

Elasticsearch

Elasticsearch (`elasticsearch`) 操作将来自 MQTT 消息的数据写入亚马逊 OpenSearch 服务域。然后，您可以使用 OpenSearch 仪表板等工具在 Ser OpenSearch vice 中查询和可视化数据。

> **Warning**
>
> `Elasticsearch` 操作只能由现有规则操作使用。要创建新的规则操作或更新现有规则操作，请使用 `OpenSearch` 规则而不是操作。有关更多信息，请参阅 OpenSearch。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`es:ESHttpPut`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。
- 如果您使用客户托管 AWS KMS key （KMS 密钥）加密中的静态数据 OpenSearch，则服务必须有权代表呼叫者使用 KMS 密钥。有关更多信息，请参阅《亚马逊服务开发者指南》中的亚马逊 OpenSearch 服务静态数据加密。 OpenSearch

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`endpoint`

您的服务域端点。

支持替换模板：API 且 AWS CLI 仅支持

`index`

您要在其中存储数据的索引。

支持替换模板：是

`type`

您存储的文档类型。

支持替换模板：是

`id`

每个文档的唯一标识符。

支持替换模板：是

`roleARN`

允许访问 OpenSearch 服务域的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

示例

下面的 JSON 示例定义了 AWS IoT 规则中的 Elasticsearch 操作，以及如何为 `elasticsearch` 操作指定域。有关更多信息，请参阅ElasticsearchAction。

```
{
    "topicRulePayload": {
        "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "elasticsearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "my-index",
                    "type": "my-type",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es"
                }
```

```
            }
        ]
    }
}
```

以下 JSON 示例定义了一个在规则中使用替换模板的 Elasticsearch 操作。 AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "elasticsearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "${topic()}",
                    "type": "${type}",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_es"
                }
            }
        ]
    }
}
```

另请参阅

- OpenSearch
- 什么是亚马逊 OpenSearch 服务？

HTTP

HTTPS (`http`) 操作可将 MQTT 消息中的数据发送至 Web 应用程序或服务。

要求

此规则操作具有以下要求：

- 您必须先确认并启用 HTTPS 端点，然后规则引擎才能使用它们。有关更多信息，请参阅 使用 HTTP 主题规则目标。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`url`

使用 HTTP POST 方法发送消息的 HTTPS 端点。如果使用 IP 地址代替主机名，则该地址必须是 IPv4 地址。不支持 IPv6 地址。

支持替换模板：是

`confirmationUrl`

（可选）如果指定，则 AWS IoT 使用确认 URL 创建匹配的主题规则目标。您必须先启用主题规则目标，然后才能在 HTTP 操作中使用它。有关更多信息，请参阅 使用 HTTP 主题规则目标。如果您使用替换模板，则必须先手动创建主题规则目标，然后才能使用 `http` 操作。`confirmationUrl` 必须是 `url` 的前缀。

`url` 和 `confirmationUrl` 之间的关系由以下内容描述：

- 如果`url`是硬编码且`confirmationUrl`未提供，则我们会隐式地将该`url`字段视为. `confirmationUrl` AWS IoT 为创建主题规则目标`url`。
- 如果`url`和`confirmationUrl`是硬编码，则`url`必须以开头。`confirmationUrl` AWS IoT 为创建主题规则目标`confirmationUrl`。
- 如果 `url` 包含替换模板，则必须指定 `confirmationUrl` 并且 `url` 必须以 `confirmationUrl` 开头。如果 `confirmationUrl` 包含替换模板，则必须先手动创建主题规则目标，然后才能使用 `http` 操作。如果`confirmationUrl`不包含替换模板，则为 AWS IoT 创建主题规则目标`confirmationUrl`。

支持替换模板：是

`headers`

（可选）要包含在对端点的 HTTP 请求中的标头列表。每个标头都必须包含以下信息：

`key`

标头的键。

支持替换模板：否

`value`

标头的值。

支持替换模板：是

> **Note**
>
> 当负载为 JSON 格式时，默认内容类型是 application/json。否则，它是 application/octet-stream。您可以在标头中，使用键内容类型（不区分大小写）指定确切的内容类型来覆盖它。

`auth`

（可选）规则引擎在连接到 `url` 参数中指定的端点 URL 时使用的身份验证。目前，Signature Version 4 是唯一支持的身份验证类型。有关更多信息，请参阅 HTTP 授权。

支持替换模板：否

示例

以下 JSON 示例定义了一 AWS IoT 条带有 HTTP 操作的规则。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "http": {
                    "url": "https://www.example.com/subpath",
                    "confirmationUrl": "https://www.example.com",
                    "headers": [
                        {
                            "key": "static_header_key",
                            "value": "static_header_value"
                        },
                        {
                            "key": "substitutable_header_key",
                            "value": "${value_from_payload}"
                        }
                    ]
                }
            }
```

```
        ]
    }
}
```

HTTP 操作重试逻辑

AWS IoT 规则引擎将根据以下规则重试 HTTP 操作：

- 规则引擎尝试至少发送一次消息。
- 规则引擎最多重试两次。最大尝试次数为三次。
- 在下列情况中，规则引擎不会尝试重试：
  - 之前的尝试提供了大于 16384 字节的响应。
  - 下游 Web 服务或应用程序在尝试之后关闭了 TCP 连接。
  - 完成包括重试在内的请求总用时超过了请求超时限制。
  - 该请求返回除 429、500-599 之外的 HTTP 状态代码。

> **Note**
>
> 标准数据传输费用适用于重试操作。

另请参阅

- 使用 HTTP 主题规则目标
- 通过 AWS博客上的物联网将数据直接路由 AWS IoT Core 到您的 Web 服务

使用 HTTP 主题规则目标

HTTP 主题规则目标是规则引擎可以将数据从主题规则路由到的 Web 服务。 AWS IoT Core 资源描述了 Web 服务 AWS IoT。主题规则目标资源可以通过不同的规则共享。

在向其他 Web 服务发送数据之前 AWS IoT Core ，它必须确认自己可以访问该服务的终端节点。

HTTP 主题规则目标概述

HTTP 主题规则目标指的是支持确认 URL 和一个或多个数据收集 URL 的网络服务。HTTP 主题规则目标资源包含 Web 服务的确认 URL。配置 HTTP 主题规则操作时，您可以指定应该接收数据的端点的

实际 URL 以及网络服务的确认 URL。确认目标后，主题规则会将 SQL 语句的结果发送到 HTTPS 端点（而不是发送到确认 URL）。

HTTP 主体规则目标可以处于以下状态之一：

已启用

已经确认目标，可以由规则操作使用。目标必须处于 ENABLED 状态才能在规则中使用。您只能启用处于 DISABLED 状态的目标。

DISABLED

已经确认目标，但规则操作无法使用。如果您希望暂时阻止流入终端节点的流量而无需再次完成确认流程，则此功能非常有用。您只能禁用处于 ENABLED 状态的目标。

进行中

正在确认目标。

错误

目标确认超时。

确认并启用 HTTP 主题规则目标后，可以与帐户中的任何规则一起使用。

以下各节介绍了对 HTTP 主题规则目标的常见操作。

创建和确认 HTTP 主题规则目标

您可以通过调用 `CreateTopicRuleDestination` 操作或使用 AWS IoT 控制台来创建 HTTP 主题规则目标。

创建目标后， AWS IoT 向确认 URL 发送确认请求。确认请求的格式如下：

```
HTTP POST {confirmationUrl}/?confirmationToken={confirmationToken}
Headers:
x-amz-rules-engine-message-type: DestinationConfirmation
x-amz-rules-engine-destination-arn:"arn:aws:iot:us-east-1:123456789012:ruledestination/
http/7a280e37-b9c6-47a2-a751-0703693f46e4"
Content-Type: application/json
Body:
{
    "arn":"arn:aws:iot:us-east-1:123456789012:ruledestination/http/7a280e37-b9c6-47a2-
a751-0703693f46e4",
```

```
    "confirmationToken": "AYADeMXLrPrNY2wqJAKsFNn-…NBJndA",
    "enableUrl": "https://iot.us-east-1.amazonaws.com/confirmdestination/
AYADeMXLrPrNY2wqJAKsFNn-…NBJndA",
    "messageType": "DestinationConfirmation"
}
```

确认请求的内容包含以下信息：

arn

要确认的主题规则目标的 Amazon Resource Name (ARN)。

confirmationToken

发送的确认令牌 AWS IoT Core。示例中是已截断的令牌。您的令牌会更长。您需要使用此令牌来确认目的地 AWS IoT Core。

enableUrl

您需要浏览以确认主题规则目标的 URL。

messageType

消息类型。

要完成端点确认过程，必须在确认 URL 收到确认请求后执行以下操作之一。

- 在确认请求中调用 `enableUrl`，然后调用 `UpdateTopicRuleDestination` 将主题规则的状态设置为 `ENABLED`。
- 调用 `ConfirmTopicRuleDestination` 操作并从确认请求中传递 `confirmationToken`。
- 复制`confirmationToken`并粘贴到 AWS IoT 控制台中目标的确认对话框中。

发送新确认请求

要为目标激活新的确认消息，请调用 `UpdateTopicRuleDestination` 并将主题规则目标的状态设置为 `IN_PROGRESS`。

发送新的确认请求后，重复确认过程。

禁用并删除主题规则目标

要禁用目标，请调用 `UpdateTopicRuleDestination` 并将主题规则目标的状态设置为 `DISABLED`。可以再次启用处于 DISABLED 状态的主题规则，而无需发送新的确认请求。

要删除主题规则目标，请调用 DeleteTopicRuleDestination。

主题规则目标中的 HTTPS 端点支持的证书颁发机构

主题规则目标中的 HTTPS 端点支持以下证书颁发机构。您可以选择其中一个支持的证书颁发机构。签名仅供参考。请注意，您不能使用自签名证书，因为它们不起作用。

帮助我们改进此主题

让我们知道您的想法。

```
Alias name: swisssignplatinumg2ca
Certificate fingerprints:
  MD5:  C9:98:27:77:28:1E:3D:0E:15:3C:84:00:B8:85:03:E6
  SHA1: 56:E0:FA:C0:3B:8F:18:23:55:18:E5:D3:11:CA:E8:C2:43:31:AB:66
  SHA256:
 3B:22:2E:56:67:11:E9:92:30:0D:C0:B1:5A:B9:47:3D:AF:DE:F8:C8:4D:0C:EF:7D:33:17:B4:C1:82:1D:14:3

Alias name: hellenicacademicandresearchinstitutionsrootca2011
Certificate fingerprints:
  MD5:  73:9F:4C:4B:73:5B:79:E9:FA:BA:1C:EF:6E:CB:D5:C9
  SHA1: FE:45:65:9B:79:03:5B:98:A1:61:B5:51:2E:AC:DA:58:09:48:22:4D
  SHA256:
 BC:10:4F:15:A4:8B:E7:09:DC:A5:42:A7:E1:D4:B9:DF:6F:05:45:27:E8:02:EA:A9:2D:59:54:44:25:8A:FE:7

Alias name: teliasonerarootcav1
Certificate fingerprints:
  MD5:  37:41:49:1B:18:56:9A:26:F5:AD:C2:66:FB:40:A5:4C
  SHA1: 43:13:BB:96:F1:D5:86:9B:C1:4E:6A:92:F6:CF:F6:34:69:87:82:37
  SHA256:
 DD:69:36:FE:21:F8:F0:77:C1:23:A1:A5:21:C1:22:24:F7:22:55:B7:3E:03:A7:26:06:93:E8:A2:4B:0F:A3:8

Alias name: geotrustprimarycertificationauthority
Certificate fingerprints:
  MD5:  02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF
  SHA1: 32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
  SHA256:
 37:D5:10:06:C5:12:EA:AB:62:64:21:F1:EC:8C:92:01:3F:C5:F8:2A:E9:8E:E5:33:EB:46:19:B8:DE:B4:D0:6

Alias name: trustisfpsrootca
Certificate fingerprints:
  MD5:  30:C9:E7:1E:6B:E6:14:EB:65:B2:16:69:20:31:67:4D
```

```
  SHA1: 3B:C0:38:0B:33:C3:F6:A6:0C:86:15:22:93:D9:DF:F5:4B:81:C0:04
  SHA256:
 C1:B4:82:99:AB:A5:20:8F:E9:63:0A:CE:55:CA:68:A0:3E:DA:5A:51:9C:88:02:A0:D3:A6:73:BE:8F:8E:55:7

Alias name: quovadisrootca3g3
Certificate fingerprints:
  MD5:  DF:7D:B9:AD:54:6F:68:A1:DF:89:57:03:97:43:B0:D7
  SHA1: 48:12:BD:92:3C:A8:C4:39:06:E7:30:6D:27:96:E6:A4:CF:22:2E:7D
  SHA256:
 88:EF:81:DE:20:2E:B0:18:45:2E:43:F8:64:72:5C:EA:5F:BD:1F:C2:D9:D2:05:73:07:09:C5:D8:B8:69:0F:4

Alias name: buypassclass2ca
Certificate fingerprints:
  MD5:  46:A7:D2:FE:45:FB:64:5A:A8:59:90:9B:78:44:9B:29
  SHA1: 49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
  SHA256:
 9A:11:40:25:19:7C:5B:B9:5D:94:E6:3D:55:CD:43:79:08:47:B6:46:B2:3C:DF:11:AD:A4:A0:0E:FF:15:FB:4

Alias name: secureglobalca
Certificate fingerprints:
  MD5:  CF:F4:27:0D:D4:ED:DC:65:16:49:6D:3D:DA:BF:6E:DE
  SHA1: 3A:44:73:5A:E5:81:90:1F:24:86:61:46:1E:3B:9C:C4:5F:F5:3A:1B
  SHA256:
 42:00:F5:04:3A:C8:59:0E:BB:52:7D:20:9E:D1:50:30:29:FB:CB:D4:1C:A1:B5:06:EC:27:F1:5A:DE:7D:AC:6

Alias name: chunghwaepkirootca
Certificate fingerprints:
  MD5:  1B:2E:00:CA:26:06:90:3D:AD:FE:6F:15:68:D3:6B:B3
  SHA1: 67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
  SHA256:
 C0:A6:F4:DC:63:A2:4B:FD:CF:54:EF:2A:6A:08:2A:0A:72:DE:35:80:3E:2F:F5:FF:52:7A:E5:D8:72:06:DF:D

Alias name: verisignclass2g2ca
Certificate fingerprints:
  MD5:  2D:BB:E5:25:D3:D1:65:82:3A:B7:0E:FA:E6:EB:E2:E1
  SHA1: B3:EA:C4:47:76:C9:C8:1C:EA:F2:9D:95:B6:CC:A0:08:1B:67:EC:9D
  SHA256:
 3A:43:E2:20:FE:7F:3E:A9:65:3D:1E:21:74:2E:AC:2B:75:C2:0F:D8:98:03:05:BC:50:2C:AF:8C:2D:9B:41:A

Alias name: szafirrootca2
Certificate fingerprints:
  MD5:  11:64:C1:89:B0:24:B1:8C:B1:07:7E:89:9E:51:9E:99
  SHA1: E2:52:FA:95:3F:ED:DB:24:60:BD:6E:28:F3:9C:CC:CF:5E:B3:3F:DE
```

```
  SHA256:
 A1:33:9D:33:28:1A:0B:56:E5:57:D3:D3:2B:1C:E7:F9:36:7E:B0:94:BD:5F:A7:2A:7E:50:04:C8:DE:D7:CA:F

Alias name: quovadisrootca1g3
Certificate fingerprints:
  MD5:  A4:BC:5B:3F:FE:37:9A:FA:64:F0:E2:FA:05:3D:0B:AB
  SHA1: 1B:8E:EA:57:96:29:1A:C9:39:EA:B8:0A:81:1A:73:73:C0:93:79:67
  SHA256:
 8A:86:6F:D1:B2:76:B5:7E:57:8E:92:1C:65:82:8A:2B:ED:58:E9:F2:F2:88:05:41:34:B7:F1:F4:BF:C9:CC:7

Alias name: utndatacorpsgcca
Certificate fingerprints:
  MD5:  B3:A5:3E:77:21:6D:AC:4A:C0:C9:FB:D5:41:3D:CA:06
  SHA1: 58:11:9F:0E:12:82:87:EA:50:FD:D9:87:45:6F:4F:78:DC:FA:D6:D4
  SHA256:
 85:FB:2F:91:DD:12:27:5A:01:45:B6:36:53:4F:84:02:4A:D6:8B:69:B8:EE:88:68:4F:F7:11:37:58:05:B3:4

Alias name: autoridaddecertificacionfirmaprofesionalcifa62634068
Certificate fingerprints:
  MD5:  73:3A:74:7A:EC:BB:A3:96:A6:C2:E4:E2:C8:9B:C0:C3
  SHA1: AE:C5:FB:3F:C8:E1:BF:C4:E5:4F:03:07:5A:9A:E8:00:B7:F7:B6:FA
  SHA256:
 04:04:80:28:BF:1F:28:64:D4:8F:9A:D4:D8:32:94:36:6A:82:88:56:55:3F:3B:14:30:3F:90:14:7F:5D:40:E

Alias name: securesignrootca11
Certificate fingerprints:
  MD5:  B7:52:74:E2:92:B4:80:93:F2:75:E4:CC:D7:F2:EA:26
  SHA1: 3B:C4:9F:48:F8:F3:73:A0:9C:1E:BD:F8:5B:B1:C3:65:C7:D8:11:B3
  SHA256:
 BF:0F:EE:FB:9E:3A:58:1A:D5:F9:E9:DB:75:89:98:57:43:D2:61:08:5C:4D:31:4F:6F:5D:72:59:AA:42:16:1

Alias name: amazon-ca-g4-acm2
Certificate fingerprints:
  MD5:  B2:F1:03:2B:93:64:05:80:B8:A8:17:36:B9:1B:52:3C
  SHA1: A7:E6:45:32:1F:7A:B7:AD:C0:70:EA:73:5F:AB:ED:C3:DA:B4:D0:C8
  SHA256:
 D7:A8:7C:69:95:D0:E2:04:2A:32:70:A7:E2:87:FE:A7:E8:F4:C1:70:62:F7:90:C3:EB:BB:53:F2:AC:39:26:B

Alias name: isrgrootx1
Certificate fingerprints:
  MD5:  0C:D2:F9:E0:DA:17:73:E9:ED:86:4D:A5:E3:70:E7:4E
  SHA1: CA:BD:2A:79:A1:07:6A:31:F2:1D:25:36:35:CB:03:9D:43:29:A5:E8
  SHA256:
 96:BC:EC:06:26:49:76:F3:74:60:77:9A:CF:28:C5:A7:CF:E8:A3:C0:AA:E1:1A:8F:FC:EE:05:C0:BD:DF:08:C
```

```
Alias name: amazon-ca-g4-acm1
Certificate fingerprints:
  MD5:  E2:F1:18:19:61:5C:43:E0:D4:A8:5D:0B:FA:7C:89:1B
  SHA1: F2:0D:28:B6:29:C2:2C:5E:84:05:E6:02:4D:97:FE:8F:A0:84:93:A0
  SHA256:
 B0:11:A4:F7:29:6C:74:D8:2B:F5:62:DF:87:D7:28:C7:1F:B5:8C:F4:E6:73:F2:78:FC:DA:F3:FF:83:A6:8C:8

Alias name: etugracertificationauthority
Certificate fingerprints:
  MD5:  B8:A1:03:63:B0:BD:21:71:70:8A:6F:13:3A:BB:79:49
  SHA1: 51:C6:E7:08:49:06:6E:F3:92:D4:5C:A0:0D:6D:A3:62:8F:C3:52:39
  SHA256:
 B0:BF:D5:2B:B0:D7:D9:BD:92:BF:5D:4D:C1:3D:A2:55:C0:2C:54:2F:37:83:65:EA:89:39:11:F5:5E:55:F2:3

Alias name: geotrustuniversalca2
Certificate fingerprints:
  MD5:  34:FC:B8:D0:36:DB:9E:14:B3:C2:F2:DB:8F:E4:94:C7
  SHA1: 37:9A:19:7B:41:85:45:35:0C:A6:03:69:F3:3C:2E:AF:47:4F:20:79
  SHA256:
 A0:23:4F:3B:C8:52:7C:A5:62:8E:EC:81:AD:5D:69:89:5D:A5:68:0D:C9:1D:1C:B8:47:7F:33:F8:78:B9:5B:0

Alias name: digicertglobalrootca
Certificate fingerprints:
  MD5:  79:E4:A9:84:0D:7D:3A:96:D7:C0:4F:E2:43:4C:89:2E
  SHA1: A8:98:5D:3A:65:E5:E5:C4:B2:D7:D6:6D:40:C6:DD:2F:B1:9C:54:36
  SHA256:
 43:48:A0:E9:44:4C:78:CB:26:5E:05:8D:5E:89:44:B4:D8:4F:96:62:BD:26:DB:25:7F:89:34:A4:43:C7:01:6

Alias name: staatdernederlandenevrootca
Certificate fingerprints:
  MD5:  FC:06:AF:7B:E8:1A:F1:9A:B4:E8:D2:70:1F:C0:F5:BA
  SHA1: 76:E2:7E:C1:4F:DB:82:C1:C0:A6:75:B5:05:BE:3D:29:B4:ED:DB:BB
  SHA256:
 4D:24:91:41:4C:FE:95:67:46:EC:4C:EF:A6:CF:6F:72:E2:8A:13:29:43:2F:9D:8A:90:7A:C4:CB:5D:AD:C1:5

Alias name: utnuserfirstclientauthemailca
Certificate fingerprints:
  MD5:  D7:34:3D:EF:1D:27:09:28:E1:31:02:5B:13:2B:DD:F7
  SHA1: B1:72:B1:A5:6D:95:F9:1F:E5:02:87:E1:4D:37:EA:6A:44:63:76:8A
  SHA256:
 43:F2:57:41:2D:44:0D:62:74:76:97:4F:87:7D:A8:F1:FC:24:44:56:5A:36:7A:E6:0E:DD:C2:7A:41:25:31:A

Alias name: actalisauthenticationrootca
```

```
Certificate fingerprints:
  MD5:  69:C1:0D:4F:07:A3:1B:C3:FE:56:3D:04:BC:11:F6:A6
  SHA1: F3:73:B3:87:06:5A:28:84:8A:F2:F3:4A:CE:19:2B:DD:C7:8E:9C:AC
  SHA256:
 55:92:60:84:EC:96:3A:64:B9:6E:2A:BE:01:CE:0B:A8:6A:64:FB:FE:BC:C7:AA:B5:AF:C1:55:B3:7F:D7:60:6

Alias name: amazonrootca4
Certificate fingerprints:
  MD5:  89:BC:27:D5:EB:17:8D:06:6A:69:D5:FD:89:47:B4:CD
  SHA1: F6:10:84:07:D6:F8:BB:67:98:0C:C2:E2:44:C2:EB:AE:1C:EF:63:BE
  SHA256:
 E3:5D:28:41:9E:D0:20:25:CF:A6:90:38:CD:62:39:62:45:8D:A5:C6:95:FB:DE:A3:C2:2B:0B:FB:25:89:70:9

Alias name: amazonrootca3
Certificate fingerprints:
  MD5:  A0:D4:EF:0B:F7:B5:D8:49:95:2A:EC:F5:C4:FC:81:87
  SHA1: 0D:44:DD:8C:3C:8C:1A:1A:58:75:64:81:E9:0F:2E:2A:FF:B3:D2:6E
  SHA256:
 18:CE:6C:FE:7B:F1:4E:60:B2:E3:47:B8:DF:E8:68:CB:31:D0:2E:BB:3A:DA:27:15:69:F5:03:43:B4:6D:B3:A

Alias name: amazonrootca2
Certificate fingerprints:
  MD5:  C8:E5:8D:CE:A8:42:E2:7A:C0:2A:5C:7C:9E:26:BF:66
  SHA1: 5A:8C:EF:45:D7:A6:98:59:76:7A:8C:8B:44:96:B5:78:CF:47:4B:1A
  SHA256:
 1B:A5:B2:AA:8C:65:40:1A:82:96:01:18:F8:0B:EC:4F:62:30:4D:83:CE:C4:71:3A:19:C3:9C:01:1E:A4:6D:B

Alias name: amazonrootca1
Certificate fingerprints:
  MD5:  43:C6:BF:AE:EC:FE:AD:2F:18:C6:88:68:30:FC:C8:E6
  SHA1: 8D:A7:F9:65:EC:5E:FC:37:91:0F:1C:6E:59:FD:C1:CC:6A:6E:DE:16
  SHA256:
 8E:CD:E6:88:4F:3D:87:B1:12:5B:A3:1A:C3:FC:B1:3D:70:16:DE:7F:57:CC:90:4F:E1:CB:97:C6:AE:98:19:6

Alias name: affirmtrustpremium
Certificate fingerprints:
  MD5:  C4:5D:0E:48:B6:AC:28:30:4E:0A:BC:F9:38:16:87:57
  SHA1: D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
  SHA256:
 70:A7:3F:7F:37:6B:60:07:42:48:90:45:34:B1:14:82:D5:BF:0E:69:8E:CC:49:8D:F5:25:77:EB:F2:E9:3B:9

Alias name: keynectisrootca
Certificate fingerprints:
  MD5:  CC:4D:AE:FB:30:6B:D8:38:FE:50:EB:86:61:4B:D2:26
```

```
  SHA1: 9C:61:5C:4D:4D:85:10:3A:53:26:C2:4D:BA:EA:E4:A2:D2:D5:CC:97
  SHA256:
 42:10:F1:99:49:9A:9A:C3:3C:8D:E0:2B:A6:DB:AA:14:40:8B:DD:8A:6E:32:46:89:C1:92:2D:06:97:15:A3:3

Alias name: equifaxsecureglobalebusinessca1
Certificate fingerprints:
  MD5:  51:F0:2A:33:F1:F5:55:39:07:F2:16:7A:47:C7:5D:63
  SHA1: 3A:74:CB:7A:47:DB:70:DE:89:1F:24:35:98:64:B8:2D:82:BD:1A:36
  SHA256:
 86:AB:5A:65:71:D3:32:9A:BC:D2:E4:E6:37:66:8B:A8:9C:73:1E:C2:93:B6:CB:A6:0F:71:63:40:A0:91:CE:A

Alias name: affirmtrustpremiumca
Certificate fingerprints:
  MD5:  C4:5D:0E:48:B6:AC:28:30:4E:0A:BC:F9:38:16:87:57
  SHA1: D8:A6:33:2C:E0:03:6F:B1:85:F6:63:4F:7D:6A:06:65:26:32:28:27
  SHA256:
 70:A7:3F:7F:37:6B:60:07:42:48:90:45:34:B1:14:82:D5:BF:0E:69:8E:CC:49:8D:F5:25:77:EB:F2:E9:3B:9

Alias name: baltimorecodesigningca
Certificate fingerprints:
  MD5:  90:F5:28:49:56:D1:5D:2C:B0:53:D4:4B:EF:6F:90:22
  SHA1: 30:46:D8:C8:88:FF:69:30:C3:4A:FC:CD:49:27:08:7C:60:56:7B:0D
  SHA256:
 A9:15:45:DB:D2:E1:9C:4C:CD:F9:09:AA:71:90:0D:18:C7:35:1C:89:B3:15:F0:F1:3D:05:C1:3A:8F:FB:46:8

Alias name: gdcatrustauthr5root
Certificate fingerprints:
  MD5:  63:CC:D9:3D:34:35:5C:6F:53:A3:E2:08:70:48:1F:B4
  SHA1: 0F:36:38:5B:81:1A:25:C3:9B:31:4E:83:CA:E9:34:66:70:CC:74:B4
  SHA256:
 BF:FF:8F:D0:44:33:48:7D:6A:8A:A6:0C:1A:29:76:7A:9F:C2:BB:B0:5E:42:0F:71:3A:13:B9:92:89:1D:38:9

Alias name: certinomisrootca
Certificate fingerprints:
  MD5:  14:0A:FD:8D:A8:28:B5:38:69:DB:56:7E:61:22:03:3F
  SHA1: 9D:70:BB:01:A5:A4:A0:18:11:2E:F7:1C:01:B9:32:C5:34:E7:88:A8
  SHA256:
 2A:99:F5:BC:11:74:B7:3C:BB:1D:62:08:84:E0:1C:34:E5:1C:CB:39:78:DA:12:5F:0E:33:26:88:83:BF:41:5

Alias name: verisignclass3publicprimarycertificationauthorityg5
Certificate fingerprints:
  MD5:  CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
  SHA1: 4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
```

```
  SHA256:
 9A:CF:AB:7E:43:C8:D8:80:D0:6B:26:2A:94:DE:EE:E4:B4:65:99:89:C3:D0:CA:F1:9B:AF:64:05:E4:1A:B7:D

Alias name: verisignclass3publicprimarycertificationauthorityg4
Certificate fingerprints:
  MD5:  3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41
  SHA1: 22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
  SHA256:
 69:DD:D7:EA:90:BB:57:C9:3E:13:5D:C8:5E:A6:FC:D5:48:0B:60:32:39:BD:C4:54:FC:75:8B:2A:26:CF:7F:7

Alias name: verisignclass3publicprimarycertificationauthorityg3
Certificate fingerprints:
  MD5:  CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09
  SHA1: 13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
  SHA256:
 EB:04:CF:5E:B1:F3:9A:FA:76:2F:2B:B1:20:F2:96:CB:A5:20:C1:B9:7D:B1:58:95:65:B8:1C:B9:A1:7B:72:4

Alias name: swisssignsilverg2ca
Certificate fingerprints:
  MD5:  E0:06:A1:C9:7D:CF:C9:FC:0D:C0:56:75:96:D8:62:13
  SHA1: 9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
  SHA256:
 BE:6C:4D:A2:BB:B9:BA:59:B6:F3:93:97:68:37:42:46:C3:C0:05:99:3F:A9:8F:02:0D:1D:ED:BE:D4:8A:81:D

Alias name: swisssignsilvercag2
Certificate fingerprints:
  MD5:  E0:06:A1:C9:7D:CF:C9:FC:0D:C0:56:75:96:D8:62:13
  SHA1: 9B:AA:E5:9F:56:EE:21:CB:43:5A:BE:25:93:DF:A7:F0:40:D1:1D:CB
  SHA256:
 BE:6C:4D:A2:BB:B9:BA:59:B6:F3:93:97:68:37:42:46:C3:C0:05:99:3F:A9:8F:02:0D:1D:ED:BE:D4:8A:81:D

Alias name: atostrustedroot2011
Certificate fingerprints:
  MD5:  AE:B9:C4:32:4B:AC:7F:5D:66:CC:77:94:BB:2A:77:56
  SHA1: 2B:B1:F5:3E:55:0C:1D:C5:F1:D4:E6:B7:6A:46:4B:55:06:02:AC:21
  SHA256:
 F3:56:BE:A2:44:B7:A9:1E:B3:5D:53:CA:9A:D7:86:4A:CE:01:8E:2D:35:D5:F8:F9:6D:DF:68:A6:F4:1A:A4:7

Alias name: comodoecccertificationauthority
Certificate fingerprints:
  MD5:  7C:62:FF:74:9D:31:53:5E:68:4A:D5:78:AA:1E:BF:23
  SHA1: 9F:74:4E:9F:2B:4D:BA:EC:0F:31:2C:50:B6:56:3B:8E:2D:93:C3:11
  SHA256:
 17:93:92:7A:06:14:54:97:89:AD:CE:2F:8F:34:F7:F0:B6:6D:0F:3A:E3:A3:B8:4D:21:EC:15:DB:BA:4F:AD:C
```

```
Alias name: securetrustca
Certificate fingerprints:
  MD5:  DC:32:C3:A7:6D:25:57:C7:68:09:9D:EA:2D:A9:A2:D1
  SHA1: 87:82:C6:C3:04:35:3B:CF:D2:96:92:D2:59:3E:7D:44:D9:34:FF:11
  SHA256:
 F1:C1:B5:0A:E5:A2:0D:D8:03:0E:C9:F6:BC:24:82:3D:D3:67:B5:25:57:59:B4:E7:1B:61:FC:E9:F7:37:5D:7

Alias name: soneraclass1ca
Certificate fingerprints:
  MD5:  33:B7:84:F5:5F:27:D7:68:27:DE:14:DE:12:2A:ED:6F
  SHA1: 07:47:22:01:99:CE:74:B9:7C:B0:3D:79:B2:64:A2:C8:55:E9:33:FF
  SHA256:
 CD:80:82:84:CF:74:6F:F2:FD:6E:B5:8A:A1:D5:9C:4A:D4:B3:CA:56:FD:C6:27:4A:89:26:A7:83:5F:32:31:3

Alias name: cadisigrootr2
Certificate fingerprints:
  MD5:  26:01:FB:D8:27:A7:17:9A:45:54:38:1A:43:01:3B:03
  SHA1: B5:61:EB:EA:A4:DE:E4:25:4B:69:1A:98:A5:57:47:C2:34:C7:D9:71
  SHA256:
 E2:3D:4A:03:6D:7B:70:E9:F5:95:B1:42:20:79:D2:B9:1E:DF:BB:1F:B6:51:A0:63:3E:AA:8A:9D:C5:F8:07:0

Alias name: cadisigrootr1
Certificate fingerprints:
  MD5:  BE:EC:11:93:9A:F5:69:21:BC:D7:C1:C0:67:89:CC:2A
  SHA1: 8E:1C:74:F8:A6:20:B9:E5:8A:F4:61:FA:EC:2B:47:56:51:1A:52:C6
  SHA256:
 F9:6F:23:F4:C3:E7:9C:07:7A:46:98:8D:5A:F5:90:06:76:A0:F0:39:CB:64:5D:D1:75:49:B2:16:C8:24:40:C

Alias name: verisignclass3g5ca
Certificate fingerprints:
  MD5:  CB:17:E4:31:67:3E:E2:09:FE:45:57:93:F3:0A:FA:1C
  SHA1: 4E:B6:D5:78:49:9B:1C:CF:5F:58:1E:AD:56:BE:3D:9B:67:44:A5:E5
  SHA256:
 9A:CF:AB:7E:43:C8:D8:80:D0:6B:26:2A:94:DE:EE:E4:B4:65:99:89:C3:D0:CA:F1:9B:AF:64:05:E4:1A:B7:D

Alias name: utnuserfirsthardwareca
Certificate fingerprints:
  MD5:  4C:56:41:E5:0D:BB:2B:E8:CA:A3:ED:18:08:AD:43:39
  SHA1: 04:83:ED:33:99:AC:36:08:05:87:22:ED:BC:5E:46:00:E3:BE:F9:D7
  SHA256:
 6E:A5:47:41:D0:04:66:7E:ED:1B:48:16:63:4A:A3:A7:9E:6E:4B:96:95:0F:82:79:DA:FC:8D:9B:D8:81:21:3

Alias name: addtrustqualifiedca
```

```
Certificate fingerprints:
  MD5:  27:EC:39:47:CD:DA:5A:AF:E2:9A:01:65:21:A9:4C:BB
  SHA1: 4D:23:78:EC:91:95:39:B5:00:7F:75:8F:03:3B:21:1E:C5:4D:8B:CF
  SHA256:
 80:95:21:08:05:DB:4B:BC:35:5E:44:28:D8:FD:6E:C2:CD:E3:AB:5F:B9:7A:99:42:98:8E:B8:F4:DC:D0:60:1

Alias name: verisignclass3g3ca
Certificate fingerprints:
  MD5:  CD:68:B6:A7:C7:C4:CE:75:E0:1D:4F:57:44:61:92:09
  SHA1: 13:2D:0D:45:53:4B:69:97:CD:B2:D5:C3:39:E2:55:76:60:9B:5C:C6
  SHA256:
 EB:04:CF:5E:B1:F3:9A:FA:76:2F:2B:B1:20:F2:96:CB:A5:20:C1:B9:7D:B1:58:95:65:B8:1C:B9:A1:7B:72:4

Alias name: thawtepersonalfreemailca
Certificate fingerprints:
  MD5:  53:4B:1D:17:58:58:1A:30:A1:90:F8:6E:5C:F2:CF:65
  SHA1: E6:18:83:AE:84:CA:C1:C1:CD:52:AD:E8:E9:25:2B:45:A6:4F:B7:E2
  SHA256:
 5B:38:BD:12:9E:83:D5:A0:CA:D2:39:21:08:94:90:D5:0D:4A:AE:37:04:28:F8:DD:FF:FF:FA:4C:15:64:E1:8

Alias name: certplusclass3pprimaryca
Certificate fingerprints:
  MD5:  E1:4B:52:73:D7:1B:DB:93:30:E5:BD:E4:09:6E:BE:FB
  SHA1: 21:6B:2A:29:E6:2A:00:CE:82:01:46:D8:24:41:41:B9:25:11:B2:79
  SHA256:
 CC:C8:94:89:37:1B:AD:11:1C:90:61:9B:EA:24:0A:2E:6D:AD:D9:9F:9F:6E:1D:4D:41:E5:8E:D6:DE:3D:02:8

Alias name: swisssigngoldg2ca
Certificate fingerprints:
  MD5:  24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93
  SHA1: D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
  SHA256:
 62:DD:0B:E9:B9:F5:0A:16:3E:A0:F8:E7:5C:05:3B:1E:CA:57:EA:55:C8:68:8F:64:7C:68:81:F2:C8:35:7B:9

Alias name: swisssigngoldcag2
Certificate fingerprints:
  MD5:  24:77:D9:A8:91:D1:3B:FA:88:2D:C2:FF:F8:CD:33:93
  SHA1: D8:C5:38:8A:B7:30:1B:1B:6E:D4:7A:E6:45:25:3A:6F:9F:1A:27:61
  SHA256:
 62:DD:0B:E9:B9:F5:0A:16:3E:A0:F8:E7:5C:05:3B:1E:CA:57:EA:55:C8:68:8F:64:7C:68:81:F2:C8:35:7B:9

Alias name: dtrustrootclass3ca22009
Certificate fingerprints:
  MD5:  CD:E0:25:69:8D:47:AC:9C:89:35:90:F7:FD:51:3D:2F
```

```
  SHA1: 58:E8:AB:B0:36:15:33:FB:80:F7:9B:1B:6D:29:D3:FF:8D:5F:00:F0
  SHA256:
 49:E7:A4:42:AC:F0:EA:62:87:05:00:54:B5:25:64:B6:50:E4:F4:9E:42:E3:48:D6:AA:38:E0:39:E9:57:B1:C

Alias name: acraizfnmtrcm
Certificate fingerprints:
  MD5:  E2:09:04:B4:D3:BD:D1:A0:14:FD:1A:D2:47:C4:57:1D
  SHA1: EC:50:35:07:B2:15:C4:95:62:19:E2:A8:9A:5B:42:99:2C:4C:2C:20
  SHA256:
 EB:C5:57:0C:29:01:8C:4D:67:B1:AA:12:7B:AF:12:F7:03:B4:61:1E:BC:17:B7:DA:B5:57:38:94:17:9B:93:F

Alias name: securitycommunicationevrootca1
Certificate fingerprints:
  MD5:  22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3
  SHA1: FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
  SHA256:
 A2:2D:BA:68:1E:97:37:6E:2D:39:7D:72:8A:AE:3A:9B:62:96:B9:FD:BA:60:BC:2E:11:F6:47:F2:C6:75:FB:3

Alias name: starfieldclass2ca
Certificate fingerprints:
  MD5:  32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
  SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
  SHA256:
 14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB:5F:B6:5

Alias name: opentrustrootcag3
Certificate fingerprints:
  MD5:  21:37:B4:17:16:92:7B:67:46:70:A9:96:D7:A8:13:24
  SHA1: 6E:26:64:F3:56:BF:34:55:BF:D1:93:3F:7C:01:DE:D8:13:DA:8A:A6
  SHA256:
 B7:C3:62:31:70:6E:81:07:8C:36:7C:B8:96:19:8F:1E:32:08:DD:92:69:49:DD:8F:57:09:A4:10:F7:5B:62:9

Alias name: opentrustrootcag2
Certificate fingerprints:
  MD5:  57:24:B6:59:24:6B:AE:C8:FE:1C:0C:20:F2:C0:4E:EB
  SHA1: 79:5F:88:60:C5:AB:7C:3D:92:E6:CB:F4:8D:E1:45:CD:11:EF:60:0B
  SHA256:
 27:99:58:29:FE:6A:75:15:C1:BF:E8:48:F9:C4:76:1D:B1:6C:22:59:29:25:7B:F4:0D:08:94:F2:9E:A8:BA:F

Alias name: buypassclass2rootca
Certificate fingerprints:
  MD5:  46:A7:D2:FE:45:FB:64:5A:A8:59:90:9B:78:44:9B:29
  SHA1: 49:0A:75:74:DE:87:0A:47:FE:58:EE:F6:C7:6B:EB:C6:0B:12:40:99
```

```
  SHA256:
 9A:11:40:25:19:7C:5B:B9:5D:94:E6:3D:55:CD:43:79:08:47:B6:46:B2:3C:DF:11:AD:A4:A0:0E:FF:15:FB:4

Alias name: opentrustrootcag1
Certificate fingerprints:
  MD5:  76:00:CC:81:29:CD:55:5E:88:6A:7A:2E:F7:4D:39:DA
  SHA1: 79:91:E8:34:F7:E2:EE:DD:08:95:01:52:E9:55:2D:14:E9:58:D5:7E
  SHA256:
 56:C7:71:28:D9:8C:18:D9:1B:4C:FD:FF:BC:25:EE:91:03:D4:75:8E:A2:AB:AD:82:6A:90:F3:45:7D:46:0E:B

Alias name: globalsignr2ca
Certificate fingerprints:
  MD5:  94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30
  SHA1: 75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
  SHA256:
 CA:42:DD:41:74:5F:D0:B8:1E:B9:02:36:2C:F9:D8:BF:71:9D:A1:BD:1B:1E:FC:94:6F:5B:4C:99:F4:2C:1B:9

Alias name: buypassclass3rootca
Certificate fingerprints:
  MD5:  3D:3B:18:9E:2C:64:5A:E8:D5:88:CE:0E:F9:37:C2:EC
  SHA1: DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
  SHA256:
 ED:F7:EB:BC:A2:7A:2A:38:4D:38:7B:7D:40:10:C6:66:E2:ED:B4:84:3E:4C:29:B4:AE:1D:5B:93:32:E6:B2:4

Alias name: ecacc
Certificate fingerprints:
  MD5:  EB:F5:9D:29:0D:61:F9:42:1F:7C:C2:BA:6D:E3:15:09
  SHA1: 28:90:3A:63:5B:52:80:FA:E6:77:4C:0B:6D:A7:D6:BA:A6:4A:F2:E8
  SHA256:
 88:49:7F:01:60:2F:31:54:24:6A:E2:8C:4D:5A:EF:10:F1:D8:7E:BB:76:62:6F:4A:E0:B7:F9:5B:A7:96:87:9

Alias name: epkirootcertificationauthority
Certificate fingerprints:
  MD5:  1B:2E:00:CA:26:06:90:3D:AD:FE:6F:15:68:D3:6B:B3
  SHA1: 67:65:0D:F1:7E:8E:7E:5B:82:40:A4:F4:56:4B:CF:E2:3D:69:C6:F0
  SHA256:
 C0:A6:F4:DC:63:A2:4B:FD:CF:54:EF:2A:6A:08:2A:0A:72:DE:35:80:3E:2F:F5:FF:52:7A:E5:D8:72:06:DF:D

Alias name: verisignclass1g2ca
Certificate fingerprints:
  MD5:  DB:23:3D:F9:69:FA:4B:B9:95:80:44:73:5E:7D:41:83
  SHA1: 27:3E:E1:24:57:FD:C4:F9:0C:55:E8:2B:56:16:7F:62:F5:32:E5:47
  SHA256:
 34:1D:E9:8B:13:92:AB:F7:F4:AB:90:A9:60:CF:25:D4:BD:6E:C6:5B:9A:51:CE:6E:D0:67:D0:0E:C7:CE:9B:7
```

```
Alias name: certigna
Certificate fingerprints:
  MD5:  AB:57:A6:5B:7D:42:82:19:B5:D8:58:26:28:5E:FD:FF
  SHA1: B1:2E:13:63:45:86:A4:6F:1A:B2:60:68:37:58:2D:C4:AC:FD:94:97
  SHA256:
 E3:B6:A2:DB:2E:D7:CE:48:84:2F:7A:C5:32:41:C7:B7:1D:54:14:4B:FB:40:C1:1F:3F:1D:0B:42:F5:EE:A1:2

Alias name: camerfirmaglobalchambersignroot
Certificate fingerprints:
  MD5:  C5:E6:7B:BF:06:D0:4F:43:ED:C4:7A:65:8A:FB:6B:19
  SHA1: 33:9B:6B:14:50:24:9B:55:7A:01:87:72:84:D9:E0:2F:C3:D2:D8:E9
  SHA256:
 EF:3C:B4:17:FC:8E:BF:6F:97:87:6C:9E:4E:CE:39:DE:1E:A5:FE:64:91:41:D1:02:8B:7D:11:C0:B2:29:8C:E

Alias name: cfcaevroot
Certificate fingerprints:
  MD5:  74:E1:B6:ED:26:7A:7A:44:30:33:94:AB:7B:27:81:30
  SHA1: E2:B8:29:4B:55:84:AB:6B:58:C2:90:46:6C:AC:3F:B8:39:8F:84:83
  SHA256:
 5C:C3:D7:8E:4E:1D:5E:45:54:7A:04:E6:87:3E:64:F9:0C:F9:53:6D:1C:CC:2E:F8:00:F3:55:C4:C5:FD:70:F

Alias name: soneraclass2rootca
Certificate fingerprints:
  MD5:  A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB
  SHA1: 37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
  SHA256:
 79:08:B4:03:14:C1:38:10:0B:51:8D:07:35:80:7F:FB:FC:F8:51:8A:00:95:33:71:05:BA:38:6B:15:3D:D9:2

Alias name: certumtrustednetworkca
Certificate fingerprints:
  MD5:  D5:E9:81:40:C5:18:69:FC:46:2C:89:75:62:0F:AA:78
  SHA1: 07:E0:32:E0:20:B7:2C:3F:19:2F:06:28:A2:59:3A:19:A7:0F:06:9E
  SHA256:
 5C:58:46:8D:55:F5:8E:49:7E:74:39:82:D2:B5:00:10:B6:D1:65:37:4A:CF:83:A7:D4:A3:2D:B7:68:C4:40:8

Alias name: securitycommunicationrootca2
Certificate fingerprints:
  MD5:  6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43
  SHA1: 5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
  SHA256:
 51:3B:2C:EC:B8:10:D4:CD:E5:DD:85:39:1A:DF:C6:C2:DD:60:D8:7B:B7:36:D2:B5:21:48:4A:A4:7A:0E:BE:F

Alias name: globalsigneccrootcar5
```

```
Certificate fingerprints:
  MD5:  9F:AD:3B:1C:02:1E:8A:BA:17:74:38:81:0C:A2:BC:08
  SHA1: 1F:24:C6:30:CD:A4:18:EF:20:69:FF:AD:4F:DD:5F:46:3A:1B:69:AA
  SHA256:
 17:9F:BC:14:8A:3D:D0:0F:D2:4E:A1:34:58:CC:43:BF:A7:F5:9C:81:82:D7:83:A5:13:F6:EB:EC:10:0C:89:2

Alias name: globalsigneccrootcar4
Certificate fingerprints:
  MD5:  20:F0:27:68:D1:7E:A0:9D:0E:E6:2A:CA:DF:5C:89:8E
  SHA1: 69:69:56:2E:40:80:F4:24:A1:E7:19:9F:14:BA:F3:EE:58:AB:6A:BB
  SHA256:
 BE:C9:49:11:C2:95:56:76:DB:6C:0A:55:09:86:D7:6E:3B:A0:05:66:7C:44:2C:97:62:B4:FB:B7:73:DE:22:8

Alias name: chambersofcommerceroot2008
Certificate fingerprints:
  MD5:  5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7
  SHA1: 78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
  SHA256:
 06:3E:4A:FA:C4:91:DF:D3:32:F3:08:9B:85:42:E9:46:17:D8:93:D7:FE:94:4E:10:A7:93:7E:E2:9D:96:93:C

Alias name: pscprocert
Certificate fingerprints:
  MD5:  E6:24:E9:12:01:AE:0C:DE:8E:85:C4:CE:A3:12:DD:EC
  SHA1: 70:C1:8D:74:B4:28:81:0A:E4:FD:A5:75:D7:01:9F:99:B0:3D:50:74
  SHA256:
 3C:FC:3C:14:D1:F6:84:FF:17:E3:8C:43:CA:44:0C:00:B9:67:EC:93:3E:8B:FE:06:4C:A1:D7:2C:90:F2:AD:B

Alias name: thawteprimaryrootcag3
Certificate fingerprints:
  MD5:  FB:1B:5D:43:8A:94:CD:44:C6:76:F2:43:4B:47:E7:31
  SHA1: F1:8B:53:8D:1B:E9:03:B6:A6:F0:56:43:5B:17:15:89:CA:F3:6B:F2
  SHA256:
 4B:03:F4:58:07:AD:70:F2:1B:FC:2C:AE:71:C9:FD:E4:60:4C:06:4C:F5:FF:B6:86:BA:E5:DB:AA:D7:FD:D3:4

Alias name: quovadisrootca
Certificate fingerprints:
  MD5:  27:DE:36:FE:72:B7:00:03:00:9D:F4:F0:1E:6C:04:24
  SHA1: DE:3F:40:BD:50:93:D3:9B:6C:60:F6:DA:BC:07:62:01:00:89:76:C9
  SHA256:
 A4:5E:DE:3B:BB:F0:9C:8A:E1:5C:72:EF:C0:72:68:D6:93:A2:1C:99:6F:D5:1E:67:CA:07:94:60:FD:6D:88:7

Alias name: thawteprimaryrootcag2
Certificate fingerprints:
  MD5:  74:9D:EA:60:24:C4:FD:22:53:3E:CC:3A:72:D9:29:4F
```

```
  SHA1: AA:DB:BC:22:23:8F:C4:01:A1:27:BB:38:DD:F4:1D:DB:08:9E:F0:12
  SHA256:
 A4:31:0D:50:AF:18:A6:44:71:90:37:2A:86:AF:AF:8B:95:1F:FB:43:1D:83:7F:1E:56:88:B4:59:71:ED:15:5

Alias name: deprecateditsecca
Certificate fingerprints:
  MD5:  A5:96:0C:F6:B5:AB:27:E5:01:C6:00:88:9E:60:33:E5
  SHA1: 12:12:0B:03:0E:15:14:54:F4:DD:B3:F5:DE:13:6E:83:5A:29:72:9D
  SHA256:
 9A:59:DA:86:24:1A:FD:BA:A3:39:FA:9C:FD:21:6A:0B:06:69:4D:E3:7E:37:52:6B:BE:63:C8:BC:83:74:2E:C

Alias name: usertrustrsacertificationauthority
Certificate fingerprints:
  MD5:  1B:FE:69:D1:91:B7:19:33:A3:72:A8:0F:E1:55:E5:B5
  SHA1: 2B:8F:1B:57:33:0D:BB:A2:D0:7A:6C:51:F7:0E:E9:0D:DA:B9:AD:8E
  SHA256:
 E7:93:C9:B0:2F:D8:AA:13:E2:1C:31:22:8A:CC:B0:81:19:64:3B:74:9C:89:89:64:B1:74:6D:46:C3:D4:CB:D

Alias name: entrustrootcag2
Certificate fingerprints:
  MD5:  4B:E2:C9:91:96:65:0C:F4:0E:5A:93:92:A0:0A:FE:B2
  SHA1: 8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
  SHA256:
 43:DF:57:74:B0:3E:7F:EF:5F:E4:0D:93:1A:7B:ED:F1:BB:2E:6B:42:73:8C:4E:6D:38:41:10:3D:3A:A7:F3:3

Alias name: networksolutionscertificateauthority
Certificate fingerprints:
  MD5:  D3:F3:A6:16:C0:FA:6B:1D:59:B1:2D:96:4D:0E:11:2E
  SHA1: 74:F8:A3:C3:EF:E7:B3:90:06:4B:83:90:3C:21:64:60:20:E5:DF:CE
  SHA256:
 15:F0:BA:00:A3:AC:7A:F3:AC:88:4C:07:2B:10:11:A0:77:BD:77:C0:97:F4:01:64:B2:F8:59:8A:BD:83:86:0

Alias name: trustcenterclass4caii
Certificate fingerprints:
  MD5:  9D:FB:F9:AC:ED:89:33:22:F4:28:48:83:25:23:5B:E0
  SHA1: A6:9A:91:FD:05:7F:13:6A:42:63:0B:B1:76:0D:2D:51:12:0C:16:50
  SHA256:
 32:66:96:7E:59:CD:68:00:8D:9D:D3:20:81:11:85:C7:04:20:5E:8D:95:FD:D8:4F:1C:7B:31:1E:67:04:FC:3

Alias name: oistewisekeyglobalrootgaca
Certificate fingerprints:
  MD5:  BC:6C:51:33:A7:E9:D3:66:63:54:15:72:1B:21:92:93
  SHA1: 59:22:A1:E1:5A:EA:16:35:21:F8:98:39:6A:46:46:B0:44:1B:0F:A9
```

```
  SHA256:
 41:C9:23:86:6A:B4:CA:D6:B7:AD:57:80:81:58:2E:02:07:97:A6:CB:DF:4F:FF:78:CE:83:96:B3:89:37:D7:F

Alias name: verisignuniversalrootcertificationauthority
Certificate fingerprints:
  MD5:  8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
  SHA1: 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
  SHA256:
 23:99:56:11:27:A5:71:25:DE:8C:EF:EA:61:0D:DF:2F:A0:78:B5:C8:06:7F:4E:82:82:90:BF:B8:60:E8:4B:3

Alias name: ttelesecglobalrootclass3ca
Certificate fingerprints:
  MD5:  CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF
  SHA1: 55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
  SHA256:
 FD:73:DA:D3:1C:64:4F:F1:B4:3B:EF:0C:CD:DA:96:71:0B:9C:D9:87:5E:CA:7E:31:70:7A:F3:E9:6D:52:2B:B

Alias name: starfieldservicesrootg2ca
Certificate fingerprints:
  MD5:  17:35:74:AF:7B:61:1C:EB:F4:F9:3C:E2:EE:40:F9:A2
  SHA1: 92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
  SHA256:
 56:8D:69:05:A2:C8:87:08:A4:B3:02:51:90:ED:CF:ED:B1:97:4A:60:6A:13:C6:E5:29:0F:CB:2A:E6:3E:DA:B

Alias name: addtrustexternalroot
Certificate fingerprints:
  MD5:  1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F
  SHA1: 02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
  SHA256:
 68:7F:A4:51:38:22:78:FF:F0:C8:B1:1F:8D:43:D5:76:67:1C:6E:B2:BC:EA:B4:13:FB:83:D9:65:D0:6D:2F:F

Alias name: turktrustelektroniksertifikahizmetsaglayicisih5
Certificate fingerprints:
  MD5:  DA:70:8E:F0:22:DF:93:26:F6:5F:9F:D3:15:06:52:4E
  SHA1: C4:18:F6:4D:46:D1:DF:00:3D:27:30:13:72:43:A9:12:11:C6:75:FB
  SHA256:
 49:35:1B:90:34:44:C1:85:CC:DC:5C:69:3D:24:D8:55:5C:B2:08:D6:A8:14:13:07:69:9F:4A:F0:63:19:9D:7

Alias name: camerfirmachambersca
Certificate fingerprints:
  MD5:  5E:80:9E:84:5A:0E:65:0B:17:02:F3:55:18:2A:3E:D7
  SHA1: 78:6A:74:AC:76:AB:14:7F:9C:6A:30:50:BA:9E:A8:7E:FE:9A:CE:3C
  SHA256:
 06:3E:4A:FA:C4:91:DF:D3:32:F3:08:9B:85:42:E9:46:17:D8:93:D7:FE:94:4E:10:A7:93:7E:E2:9D:96:93:C
```

```
Alias name: certsignrootca
Certificate fingerprints:
  MD5:  18:98:C0:D6:E9:3A:FC:F9:B0:F5:0C:F7:4B:01:44:17
  SHA1: FA:B7:EE:36:97:26:62:FB:2D:B0:2A:F6:BF:03:FD:E8:7C:4B:2F:9B
  SHA256:
 EA:A9:62:C4:FA:4A:6B:AF:EB:E4:15:19:6D:35:1C:CD:88:8D:4F:53:F3:FA:8A:E6:D7:C4:66:A9:4E:60:42:B

Alias name: verisignuniversalrootca
Certificate fingerprints:
  MD5:  8E:AD:B5:01:AA:4D:81:E4:8C:1D:D1:E1:14:00:95:19
  SHA1: 36:79:CA:35:66:87:72:30:4D:30:A5:FB:87:3B:0F:A7:7B:B7:0D:54
  SHA256:
 23:99:56:11:27:A5:71:25:DE:8C:EF:EA:61:0D:DF:2F:A0:78:B5:C8:06:7F:4E:82:82:90:BF:B8:60:E8:4B:3

Alias name: geotrustuniversalca
Certificate fingerprints:
  MD5:  92:65:58:8B:A2:1A:31:72:73:68:5C:B4:A5:7A:07:48
  SHA1: E6:21:F3:35:43:79:05:9A:4B:68:30:9D:8A:2F:74:22:15:87:EC:79
  SHA256:
 A0:45:9B:9F:63:B2:25:59:F5:FA:5D:4C:6D:B3:F9:F7:2F:F1:93:42:03:35:78:F0:73:BF:1D:1B:46:CB:B9:1



Alias name: luxtrustglobalroot2
Certificate fingerprints:
  MD5:  B2:E1:09:00:61:AF:F7:F1:91:6F:C4:AD:8D:5E:3B:7C
  SHA1: 1E:0E:56:19:0A:D1:8B:25:98:B2:04:44:FF:66:8A:04:17:99:5F:3F
  SHA256:
 54:45:5F:71:29:C2:0B:14:47:C4:18:F9:97:16:8F:24:C5:8F:C5:02:3B:F5:DA:5B:E2:EB:6E:1D:D8:90:2E:D



Alias name: twcaglobalrootca
Certificate fingerprints:
  MD5:  F9:03:7E:CF:E6:9E:3C:73:7A:2A:90:07:69:FF:2B:96
  SHA1: 9C:BB:48:53:F6:A4:F6:D3:52:A4:E8:32:52:55:60:13:F5:AD:AF:65
  SHA256:
 59:76:90:07:F7:68:5D:0F:CD:50:87:2F:9F:95:D5:75:5A:5B:2B:45:7D:81:F3:69:2B:61:0A:98:67:2F:0E:1

Alias name: tubitakkamusmsslkoksertifikasisurum1
Certificate fingerprints:
  MD5:  DC:00:81:DC:69:2F:3E:2F:B0:3B:F6:3D:5A:91:8E:49
  SHA1: 31:43:64:9B:EC:CE:27:EC:ED:3A:3F:0B:8F:0D:E4:E8:91:DD:EE:CA
```

```
  SHA256:
 46:ED:C3:68:90:46:D5:3A:45:3F:B3:10:4A:B8:0D:CA:EC:65:8B:26:60:EA:16:29:DD:7E:86:79:90:64:87:1

Alias name: affirmtrustnetworkingca
Certificate fingerprints:
  MD5:  42:65:CA:BE:01:9A:9A:4C:A9:8C:41:49:CD:C0:D5:7F
  SHA1: 29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
  SHA256:
 0A:81:EC:5A:92:97:77:F1:45:90:4A:F3:8D:5D:50:9F:66:B5:E2:C5:8F:CD:B5:31:05:8B:0E:17:F3:F0:B4:1

Alias name: affirmtrustcommercialca
Certificate fingerprints:
  MD5:  82:92:BA:5B:EF:CD:8A:6F:A6:3D:55:F9:84:F6:D6:B7
  SHA1: F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
  SHA256:
 03:76:AB:1D:54:C5:F9:80:3C:E4:B2:E2:01:A0:EE:7E:EF:7B:57:B6:36:E8:A9:3C:9B:8D:48:60:C9:6F:5F:A

Alias name: godaddyrootcertificateauthorityg2
Certificate fingerprints:
  MD5:  80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01
  SHA1: 47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
  SHA256:
 45:14:0B:32:47:EB:9C:C8:C5:B4:F0:D7:B5:30:91:F7:32:92:08:9E:6E:5A:63:E2:74:9D:D3:AC:A9:19:8E:D

Alias name: starfieldrootg2ca
Certificate fingerprints:
  MD5:  D6:39:81:C6:52:7E:96:69:FC:FC:CA:66:ED:05:F2:96
  SHA1: B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
  SHA256:
 2C:E1:CB:0B:F9:D2:F9:E1:02:99:3F:BE:21:51:52:C3:B2:DD:0C:AB:DE:1C:68:E5:31:9B:83:91:54:DB:B7:F

Alias name: dtrustrootclass3ca2ev2009
Certificate fingerprints:
  MD5:  AA:C6:43:2C:5E:2D:CD:C4:34:C0:50:4F:11:02:4F:B6
  SHA1: 96:C9:1B:0B:95:B4:10:98:42:FA:D0:D8:22:79:FE:60:FA:B9:16:83
  SHA256:
 EE:C5:49:6B:98:8C:E9:86:25:B9:34:09:2E:EC:29:08:BE:D0:B0:F3:16:C2:D4:73:0C:84:EA:F1:F3:D3:48:8

Alias name: buypassclass3ca
Certificate fingerprints:
  MD5:  3D:3B:18:9E:2C:64:5A:E8:D5:88:CE:0E:F9:37:C2:EC
  SHA1: DA:FA:F7:FA:66:84:EC:06:8F:14:50:BD:C7:C2:81:A5:BC:A9:64:57
  SHA256:
 ED:F7:EB:BC:A2:7A:2A:38:4D:38:7B:7D:40:10:C6:66:E2:ED:B4:84:3E:4C:29:B4:AE:1D:5B:93:32:E6:B2:4
```

```
Alias name: verisignclass2g3ca
Certificate fingerprints:
  MD5:  F8:BE:C4:63:22:C9:A8:46:74:8B:B8:1D:1E:4A:2B:F6
  SHA1: 61:EF:43:D7:7F:CA:D4:61:51:BC:98:E0:C3:59:12:AF:9F:EB:63:11
  SHA256:
 92:A9:D9:83:3F:E1:94:4D:B3:66:E8:BF:AE:7A:95:B6:48:0C:2D:6C:6C:2A:1B:E6:5D:42:36:B6:08:FC:A1:B

Alias name: digicerttrustedrootg4
Certificate fingerprints:
  MD5:  78:F2:FC:AA:60:1F:2F:B4:EB:C9:37:BA:53:2E:75:49
  SHA1: DD:FB:16:CD:49:31:C9:73:A2:03:7D:3F:C8:3A:4D:7D:77:5D:05:E4
  SHA256:
 55:2F:7B:DC:F1:A7:AF:9E:6C:E6:72:01:7F:4F:12:AB:F7:72:40:C7:8E:76:1A:C2:03:D1:D9:D2:0A:C8:99:8

Alias name: quovadisrootca2g3
Certificate fingerprints:
  MD5:  AF:0C:86:6E:BF:40:2D:7F:0B:3E:12:50:BA:12:3D:06
  SHA1: 09:3C:61:F3:8B:8B:DC:7D:55:DF:75:38:02:05:00:E1:25:F5:C8:36
  SHA256:
 8F:E4:FB:0A:F9:3A:4D:0D:67:DB:0B:EB:B2:3E:37:C7:1B:F3:25:DC:BC:DD:24:0E:A0:4D:AF:58:B4:7E:18:4

Alias name: geotrustprimarycertificationauthorityg3
Certificate fingerprints:
  MD5:  B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
  SHA1: 03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
  SHA256:
 B4:78:B8:12:25:0D:F8:78:63:5C:2A:A7:EC:7D:15:5E:AA:62:5E:E8:29:16:E2:CD:29:43:61:88:6C:D1:FB:D

Alias name: geotrustprimarycertificationauthorityg2
Certificate fingerprints:
  MD5:  01:5E:D8:6B:BD:6F:3D:8E:A1:31:F8:12:E0:98:73:6A
  SHA1: 8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
  SHA256:
 5E:DB:7A:C4:3B:82:A0:6A:87:61:E8:D7:BE:49:79:EB:F2:61:1F:7D:D7:9B:F9:1C:1C:6B:56:6A:21:9E:D7:6

Alias name: godaddyclass2ca
Certificate fingerprints:
  MD5:  91:DE:06:25:AB:DA:FD:32:17:0C:BB:25:17:2A:84:67
  SHA1: 27:96:BA:E6:3F:18:01:E2:77:26:1B:A0:D7:77:70:02:8F:20:EE:E4
  SHA256:
 C3:84:6B:F2:4B:9E:93:CA:64:27:4C:0E:C6:7C:1E:CC:5E:02:4F:FC:AC:D2:D7:40:19:35:0E:81:FE:54:6A:E

Alias name: trustcoreca1
```

```
Certificate fingerprints:
  MD5:  27:92:23:1D:0A:F5:40:7C:E9:E6:6B:9D:D8:F5:E7:6C
  SHA1: 58:D1:DF:95:95:67:6B:63:C0:F0:5B:1C:17:4D:8B:84:0B:C8:78:BD
  SHA256:
 5A:88:5D:B1:9C:01:D9:12:C5:75:93:88:93:8C:AF:BB:DF:03:1A:B2:D4:8E:91:EE:15:58:9B:42:97:1D:03:9

Alias name: hellenicacademicandresearchinstitutionseccrootca2015
Certificate fingerprints:
  MD5:  81:E5:B4:17:EB:C2:F5:E1:4B:0D:41:7B:49:92:FE:EF
  SHA1: 9F:F1:71:8D:92:D5:9A:F3:7D:74:97:B4:BC:6F:84:68:0B:BA:B6:66
  SHA256:
 44:B5:45:AA:8A:25:E6:5A:73:CA:15:DC:27:FC:36:D2:4C:1C:B9:95:3A:06:65:39:B1:15:82:DC:48:7B:48:3

Alias name: utnuserfirstobjectca
Certificate fingerprints:
  MD5:  A7:F2:E4:16:06:41:11:50:30:6B:9C:E3:B4:9C:B0:C9
  SHA1: E1:2D:FB:4B:41:D7:D9:C3:2B:30:51:4B:AC:1D:81:D8:38:5E:2D:46
  SHA256:
 6F:FF:78:E4:00:A7:0C:11:01:1C:D8:59:77:C4:59:FB:5A:F9:6A:3D:F0:54:08:20:D0:F4:B8:60:78:75:E5:8

Alias name: ttelesecglobalrootclass3
Certificate fingerprints:
  MD5:  CA:FB:40:A8:4E:39:92:8A:1D:FE:8E:2F:C4:27:EA:EF
  SHA1: 55:A6:72:3E:CB:F2:EC:CD:C3:23:74:70:19:9D:2A:BE:11:E3:81:D1
  SHA256:
 FD:73:DA:D3:1C:64:4F:F1:B4:3B:EF:0C:CD:DA:96:71:0B:9C:D9:87:5E:CA:7E:31:70:7A:F3:E9:6D:52:2B:B

Alias name: ttelesecglobalrootclass2
Certificate fingerprints:
  MD5:  2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A
  SHA1: 59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
  SHA256:
 91:E2:F5:78:8D:58:10:EB:A7:BA:58:73:7D:E1:54:8A:8E:CA:CD:01:45:98:BC:0B:14:3E:04:1B:17:05:25:5

Alias name: addtrustclass1ca
Certificate fingerprints:
  MD5:  1E:42:95:02:33:92:6B:B9:5F:C0:7F:DA:D6:B2:4B:FC
  SHA1: CC:AB:0E:A0:4C:23:01:D6:69:7B:DD:37:9F:CD:12:EB:24:E3:94:9D
  SHA256:
 8C:72:09:27:9A:C0:4E:27:5E:16:D0:7F:D3:B7:75:E8:01:54:B5:96:80:46:E3:1F:52:DD:25:76:63:24:E9:A

Alias name: amzninternalrootca
Certificate fingerprints:
  MD5:  08:09:73:AC:E0:78:41:7C:0A:26:33:51:E8:CF:E6:60
```

```
  SHA1: A7:B7:F6:15:8A:FF:1E:C8:85:13:38:BC:93:EB:A2:AB:A4:09:EF:06
  SHA256:
 0E:DE:63:C1:DC:7A:8E:11:F1:AB:BC:05:4F:59:EE:49:9D:62:9A:2F:DE:9C:A7:16:32:A2:64:29:3E:8B:66:A

Alias name: starfieldrootcertificateauthorityg2
Certificate fingerprints:
  MD5:  D6:39:81:C6:52:7E:96:69:FC:FC:CA:66:ED:05:F2:96
  SHA1: B5:1C:06:7C:EE:2B:0C:3D:F8:55:AB:2D:92:F4:FE:39:D4:E7:0F:0E
  SHA256:
 2C:E1:CB:0B:F9:D2:F9:E1:02:99:3F:BE:21:51:52:C3:B2:DD:0C:AB:DE:1C:68:E5:31:9B:83:91:54:DB:B7:F

Alias name: camerfirmachambersignca
Certificate fingerprints:
  MD5:  9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
  SHA1: 4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
  SHA256:
 13:63:35:43:93:34:A7:69:80:16:A0:D3:24:DE:72:28:4E:07:9D:7B:52:20:BB:8F:BD:74:78:16:EE:BE:BA:C

Alias name: secomscrootca2
Certificate fingerprints:
  MD5:  6C:39:7D:A4:0E:55:59:B2:3F:D6:41:B1:12:50:DE:43
  SHA1: 5F:3B:8C:F2:F8:10:B3:7D:78:B4:CE:EC:19:19:C3:73:34:B9:C7:74
  SHA256:
 51:3B:2C:EC:B8:10:D4:CD:E5:DD:85:39:1A:DF:C6:C2:DD:60:D8:7B:B7:36:D2:B5:21:48:4A:A4:7A:0E:BE:F

Alias name: entrustevca
Certificate fingerprints:
  MD5:  D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4
  SHA1: B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
  SHA256:
 73:C1:76:43:4F:1B:C6:D5:AD:F4:5B:0E:76:E7:27:28:7C:8D:E5:76:16:C1:E6:E6:14:1A:2B:2C:BC:7D:8E:4

Alias name: secomscrootca1
Certificate fingerprints:
  MD5:  F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A
  SHA1: 36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
  SHA256:
 E7:5E:72:ED:9F:56:0E:EC:6E:B4:80:00:73:A4:3F:C3:AD:19:19:5A:39:22:82:01:78:95:97:4A:99:02:6B:6

Alias name: affirmtrustcommercial
Certificate fingerprints:
  MD5:  82:92:BA:5B:EF:CD:8A:6F:A6:3D:55:F9:84:F6:D6:B7
  SHA1: F9:B5:B6:32:45:5F:9C:BE:EC:57:5F:80:DC:E9:6E:2C:C7:B2:78:B7
```

```
  SHA256:
 03:76:AB:1D:54:C5:F9:80:3C:E4:B2:E2:01:A0:EE:7E:EF:7B:57:B6:36:E8:A9:3C:9B:8D:48:60:C9:6F:5F:A

Alias name: digicertassuredidrootg3
Certificate fingerprints:
  MD5:  7C:7F:65:31:0C:81:DF:8D:BA:3E:99:E2:5C:AD:6E:FB
  SHA1: F5:17:A2:4F:9A:48:C6:C9:F8:A2:00:26:9F:DC:0F:48:2C:AB:30:89
  SHA256:
 7E:37:CB:8B:4C:47:09:0C:AB:36:55:1B:A6:F4:5D:B8:40:68:0F:BA:16:6A:95:2D:B1:00:71:7F:43:05:3F:C

Alias name: affirmtrustnetworking
Certificate fingerprints:
  MD5:  42:65:CA:BE:01:9A:9A:4C:A9:8C:41:49:CD:C0:D5:7F
  SHA1: 29:36:21:02:8B:20:ED:02:F5:66:C5:32:D1:D6:ED:90:9F:45:00:2F
  SHA256:
 0A:81:EC:5A:92:97:77:F1:45:90:4A:F3:8D:5D:50:9F:66:B5:E2:C5:8F:CD:B5:31:05:8B:0E:17:F3:F0:B4:1

Alias name: izenpecom
Certificate fingerprints:
  MD5:  A6:B0:CD:85:80:DA:5C:50:34:A3:39:90:2F:55:67:73
  SHA1: 2F:78:3D:25:52:18:A7:4A:65:39:71:B5:2C:A2:9C:45:15:6F:E9:19
  SHA256:
 25:30:CC:8E:98:32:15:02:BA:D9:6F:9B:1F:BA:1B:09:9E:2D:29:9E:0F:45:48:BB:91:4F:36:3B:C0:D4:53:1

Alias name: amazon-ca-g4-legacy
Certificate fingerprints:
  MD5:  6C:E5:BD:67:A4:4F:E3:FD:C2:4C:46:E6:06:5B:6D:55
  SHA1: EA:E7:DE:F9:0A:BE:9F:0B:68:CE:B7:24:0D:80:74:03:BF:6E:B1:6E
  SHA256:
 CD:72:C4:7F:B4:AD:28:A4:67:2B:E1:86:47:D4:40:E9:3B:16:2D:95:DB:3C:2F:94:BB:81:D9:09:F7:91:24:5

Alias name: digicertassuredidrootg2
Certificate fingerprints:
  MD5:  92:38:B9:F8:63:24:82:65:2C:57:33:E6:FE:81:8F:9D
  SHA1: A1:4B:48:D9:43:EE:0A:0E:40:90:4F:3C:E0:A4:C0:91:93:51:5D:3F
  SHA256:
 7D:05:EB:B6:82:33:9F:8C:94:51:EE:09:4E:EB:FE:FA:79:53:A1:14:ED:B2:F4:49:49:45:2F:AB:7D:2F:C1:8

Alias name: comodoaaaservicesroot
Certificate fingerprints:
  MD5:  49:79:04:B0:EB:87:19:AC:47:B0:BC:11:51:9B:74:D0
  SHA1: D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
  SHA256:
 D7:A7:A0:FB:5D:7E:27:31:D7:71:E9:48:4E:BC:DE:F7:1D:5F:0C:3E:0A:29:48:78:2B:C8:3E:E0:EA:69:9E:F
```

```
Alias name: entrustnetpremium2048secureserverca
Certificate fingerprints:
  MD5:  EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90
  SHA1: 50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
  SHA256:
 6D:C4:71:72:E0:1C:BC:B0:BF:62:58:0D:89:5F:E2:B8:AC:9A:D4:F8:73:80:1E:0C:10:B9:C8:37:D2:1E:B1:7

Alias name: trustcorrootcertca2
Certificate fingerprints:
  MD5:  A2:E1:F8:18:0B:BA:45:D5:C7:41:2A:BB:37:52:45:64
  SHA1: B8:BE:6D:CB:56:F1:55:B9:63:D4:12:CA:4E:06:34:C7:94:B2:1C:C0
  SHA256:
 07:53:E9:40:37:8C:1B:D5:E3:83:6E:39:5D:AE:A5:CB:83:9E:50:46:F1:BD:0E:AE:19:51:CF:10:FE:C7:C9:6

Alias name: entrust2048ca
Certificate fingerprints:
  MD5:  EE:29:31:BC:32:7E:9A:E6:E8:B5:F7:51:B4:34:71:90
  SHA1: 50:30:06:09:1D:97:D4:F5:AE:39:F7:CB:E7:92:7D:7D:65:2D:34:31
  SHA256:
 6D:C4:71:72:E0:1C:BC:B0:BF:62:58:0D:89:5F:E2:B8:AC:9A:D4:F8:73:80:1E:0C:10:B9:C8:37:D2:1E:B1:7

Alias name: trustcorrootcertca1
Certificate fingerprints:
  MD5:  6E:85:F1:DC:1A:00:D3:22:D5:B2:B2:AC:6B:37:05:45
  SHA1: FF:BD:CD:E7:82:C8:43:5E:3C:6F:26:86:5C:CA:A8:3A:45:5B:C3:0A
  SHA256:
 D4:0E:9C:86:CD:8F:E4:68:C1:77:69:59:F4:9E:A7:74:FA:54:86:84:B6:C4:06:F3:90:92:61:F4:DC:E2:57:5

Alias name: baltimorecybertrustroot
Certificate fingerprints:
  MD5:  AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4
  SHA1: D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
  SHA256:
 16:AF:57:A9:F6:76:B0:AB:12:60:95:AA:5E:BA:DE:F2:2A:B3:11:19:D6:44:AC:95:CD:4B:93:DB:F3:F2:6A:E

Alias name: eecertificationcentrerootca
Certificate fingerprints:
  MD5:  43:5E:88:D4:7D:1A:4A:7E:FD:84:2E:52:EB:01:D4:6F
  SHA1: C9:A8:B9:E7:55:80:5E:58:E3:53:77:A7:25:EB:AF:C3:7B:27:CC:D7
  SHA256:
 3E:84:BA:43:42:90:85:16:E7:75:73:C0:99:2F:09:79:CA:08:4E:46:85:68:1F:F1:95:CC:BA:8A:22:9B:8A:7

Alias name: dstacescax6
```

```
Certificate fingerprints:
  MD5:  21:D8:4C:82:2B:99:09:33:A2:EB:14:24:8D:8E:5F:E8
  SHA1: 40:54:DA:6F:1C:3F:40:74:AC:ED:0F:EC:CD:DB:79:D1:53:FB:90:1D
  SHA256:
 76:7C:95:5A:76:41:2C:89:AF:68:8E:90:A1:C7:0F:55:6C:FD:6B:60:25:DB:EA:10:41:6D:7E:B6:83:1F:8C:4

Alias name: comodocertificationauthority
Certificate fingerprints:
  MD5:  5C:48:DC:F7:42:72:EC:56:94:6D:1C:CC:71:35:80:75
  SHA1: 66:31:BF:9E:F7:4F:9E:B6:C9:D5:A6:0C:BA:6A:BE:D1:F7:BD:EF:7B
  SHA256:
 0C:2C:D6:3D:F7:80:6F:A3:99:ED:E8:09:11:6B:57:5B:F8:79:89:F0:65:18:F9:80:8C:86:05:03:17:8B:AF:6

Alias name: thawteserverca
Certificate fingerprints:
  MD5:  EE:FE:61:69:65:6E:F8:9C:C6:2A:F4:D7:2B:63:EF:A2
  SHA1: 9F:AD:91:A6:CE:6A:C6:C5:00:47:C4:4E:C9:D4:A5:0D:92:D8:49:79
  SHA256:
 87:C6:78:BF:B8:B2:5F:38:F7:E9:7B:33:69:56:BB:CF:14:4B:BA:CA:A5:36:47:E6:1A:23:25:BC:10:55:31:6

Alias name: secomvalicertclass1ca
Certificate fingerprints:
  MD5:  65:58:AB:15:AD:57:6C:1E:A8:A7:B5:69:AC:BF:FF:EB
  SHA1: E5:DF:74:3C:B6:01:C4:9B:98:43:DC:AB:8C:E8:6A:81:10:9F:E4:8E
  SHA256:
 F4:C1:49:55:1A:30:13:A3:5B:C7:BF:FE:17:A7:F3:44:9B:C1:AB:5B:5A:0A:E7:4B:06:C2:3B:90:00:4C:01:0

Alias name: godaddyrootg2ca
Certificate fingerprints:
  MD5:  80:3A:BC:22:C1:E6:FB:8D:9B:3B:27:4A:32:1B:9A:01
  SHA1: 47:BE:AB:C9:22:EA:E8:0E:78:78:34:62:A7:9F:45:C2:54:FD:E6:8B
  SHA256:
 45:14:0B:32:47:EB:9C:C8:C5:B4:F0:D7:B5:30:91:F7:32:92:08:9E:6E:5A:63:E2:74:9D:D3:AC:A9:19:8E:D

Alias name: globalchambersignroot2008
Certificate fingerprints:
  MD5:  9E:80:FF:78:01:0C:2E:C1:36:BD:FE:96:90:6E:08:F3
  SHA1: 4A:BD:EE:EC:95:0D:35:9C:89:AE:C7:52:A1:2C:5B:29:F6:D6:AA:0C
  SHA256:
 13:63:35:43:93:34:A7:69:80:16:A0:D3:24:DE:72:28:4E:07:9D:7B:52:20:BB:8F:BD:74:78:16:EE:BE:BA:C

Alias name: equifaxsecureebusinessca1
Certificate fingerprints:
  MD5:  14:C0:08:E5:A3:85:03:A3:BE:78:E9:67:4F:27:CA:EE
```

```
  SHA1: AE:E6:3D:70:E3:76:FB:C7:3A:EB:B0:A1:C1:D4:C4:7A:A7:40:B3:F4
  SHA256:
 2E:3A:2B:B5:11:25:05:83:6C:A8:96:8B:E2:CB:37:27:CE:9B:56:84:5C:6E:E9:8E:91:85:10:4A:FB:9A:F5:9

Alias name: quovadisrootca3
Certificate fingerprints:
  MD5:  31:85:3C:62:94:97:63:B9:AA:FD:89:4E:AF:6F:E0:CF
  SHA1: 1F:49:14:F7:D8:74:95:1D:DD:AE:02:C0:BE:FD:3A:2D:82:75:51:85
  SHA256:
 18:F1:FC:7F:20:5D:F8:AD:DD:EB:7F:E0:07:DD:57:E3:AF:37:5A:9C:4D:8D:73:54:6B:F4:F1:FE:D1:E1:8D:3

Alias name: usertrustecccertificationauthority
Certificate fingerprints:
  MD5:  FA:68:BC:D9:B5:7F:AD:FD:C9:1D:06:83:28:CC:24:C1
  SHA1: D1:CB:CA:5D:B2:D5:2A:7F:69:3B:67:4D:E5:F0:5A:1D:0C:95:7D:F0
  SHA256:
 4F:F4:60:D5:4B:9C:86:DA:BF:BC:FC:57:12:E0:40:0D:2B:ED:3F:BC:4D:4F:BD:AA:86:E0:6A:DC:D2:A9:AD:7

Alias name: quovadisrootca2
Certificate fingerprints:
  MD5:  5E:39:7B:DD:F8:BA:EC:82:E9:AC:62:BA:0C:54:00:2B
  SHA1: CA:3A:FB:CF:12:40:36:4B:44:B2:16:20:88:80:48:39:19:93:7C:F7
  SHA256:
 85:A0:DD:7D:D7:20:AD:B7:FF:05:F8:3D:54:2B:20:9D:C7:FF:45:28:F7:D6:77:B1:83:89:FE:A5:E5:C4:9E:8

Alias name: soneraclass2ca
Certificate fingerprints:
  MD5:  A3:EC:75:0F:2E:88:DF:FA:48:01:4E:0B:5C:48:6F:FB
  SHA1: 37:F7:6D:E6:07:7C:90:C5:B1:3E:93:1A:B7:41:10:B4:F2:E4:9A:27
  SHA256:
 79:08:B4:03:14:C1:38:10:0B:51:8D:07:35:80:7F:FB:FC:F8:51:8A:00:95:33:71:05:BA:38:6B:15:3D:D9:2

Alias name: twcarootcertificationauthority
Certificate fingerprints:
  MD5:  AA:08:8F:F6:F9:7B:B7:F2:B1:A7:1E:9B:EA:EA:BD:79
  SHA1: CF:9E:87:6D:D3:EB:FC:42:26:97:A3:B5:A3:7A:A0:76:A9:06:23:48
  SHA256:
 BF:D8:8F:E1:10:1C:41:AE:3E:80:1B:F8:BE:56:35:0E:E9:BA:D1:A6:B9:BD:51:5E:DC:5C:6D:5B:87:11:AC:4

Alias name: baltimorecybertrustca
Certificate fingerprints:
  MD5:  AC:B6:94:A5:9C:17:E0:D7:91:52:9B:B1:97:06:A6:E4
  SHA1: D4:DE:20:D0:5E:66:FC:53:FE:1A:50:88:2C:78:DB:28:52:CA:E4:74
```

```
  SHA256:
 16:AF:57:A9:F6:76:B0:AB:12:60:95:AA:5E:BA:DE:F2:2A:B3:11:19:D6:44:AC:95:CD:4B:93:DB:F3:F2:6A:E

Alias name: cia-crt-g3-01-ca
Certificate fingerprints:
  MD5:  E3:66:DD:D6:A0:D5:40:8F:FF:29:E2:C0:CB:6E:62:1A
  SHA1: 2B:EE:2C:BA:A3:1D:B5:FE:60:40:41:95:08:ED:46:82:39:4D:ED:E2
  SHA256:
 20:48:AD:4C:EC:90:7F:FA:4A:15:D4:CE:45:E3:C8:E4:2C:EA:78:33:DC:C7:D3:40:48:FC:60:47:27:42:99:E

Alias name: entrustrootcertificationauthorityg2
Certificate fingerprints:
  MD5:  4B:E2:C9:91:96:65:0C:F4:0E:5A:93:92:A0:0A:FE:B2
  SHA1: 8C:F4:27:FD:79:0C:3A:D1:66:06:8D:E8:1E:57:EF:BB:93:22:72:D4
  SHA256:
 43:DF:57:74:B0:3E:7F:EF:5F:E4:0D:93:1A:7B:ED:F1:BB:2E:6B:42:73:8C:4E:6D:38:41:10:3D:3A:A7:F3:3

Alias name: verisignclass3g4ca
Certificate fingerprints:
  MD5:  3A:52:E1:E7:FD:6F:3A:E3:6F:F3:6F:99:1B:F9:22:41
  SHA1: 22:D5:D8:DF:8F:02:31:D1:8D:F7:9D:B7:CF:8A:2D:64:C9:3F:6C:3A
  SHA256:
 69:DD:D7:EA:90:BB:57:C9:3E:13:5D:C8:5E:A6:FC:D5:48:0B:60:32:39:BD:C4:54:FC:75:8B:2A:26:CF:7F:7

Alias name: xrampglobalcaroot
Certificate fingerprints:
  MD5:  A1:0B:44:B3:CA:10:D8:00:6E:9D:0F:D8:0F:92:0A:D1
  SHA1: B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
  SHA256:
 CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:C0:FF:0B:CF:0D:32:86:FC:1A:A

Alias name: identrustcommercialrootca1
Certificate fingerprints:
  MD5:  B3:3E:77:73:75:EE:A0:D3:E3:7E:49:63:49:59:BB:C7
  SHA1: DF:71:7E:AA:4A:D9:4E:C9:55:84:99:60:2D:48:DE:5F:BC:F0:3A:25
  SHA256:
 5D:56:49:9B:E4:D2:E0:8B:CF:CA:D0:8A:3E:38:72:3D:50:50:3B:DE:70:69:48:E4:2F:55:60:30:19:E5:28:A

Alias name: camerfirmachamberscommerceca
Certificate fingerprints:
  MD5:  B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84
  SHA1: 6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
  SHA256:
 0C:25:8A:12:A5:67:4A:EF:25:F2:8B:A7:DC:FA:EC:EE:A3:48:E5:41:E6:F5:CC:4E:E6:3B:71:B3:61:60:6A:C
```

```
Alias name: verisignclass3g2ca
Certificate fingerprints:
  MD5:  A2:33:9B:4C:74:78:73:D4:6C:E7:C1:F3:8D:CB:5C:E9
  SHA1: 85:37:1C:A6:E5:50:14:3D:CE:28:03:47:1B:DE:3A:09:E8:F8:77:0F
  SHA256:
 83:CE:3C:12:29:68:8A:59:3D:48:5F:81:97:3C:0F:91:95:43:1E:DA:37:CC:5E:36:43:0E:79:C7:A8:88:63:8

Alias name: deutschetelekomrootca2
Certificate fingerprints:
  MD5:  74:01:4A:91:B1:08:C4:58:CE:47:CD:F0:DD:11:53:08
  SHA1: 85:A4:08:C0:9C:19:3E:5D:51:58:7D:CD:D6:13:30:FD:8C:DE:37:BF
  SHA256:
 B6:19:1A:50:D0:C3:97:7F:7D:A9:9B:CD:AA:C8:6A:22:7D:AE:B9:67:9E:C7:0B:A3:B0:C9:D9:22:71:C1:70:D

Alias name: certumca
Certificate fingerprints:
  MD5:  2C:8F:9F:66:1D:18:90:B1:47:26:9D:8E:86:82:8C:A9
  SHA1: 62:52:DC:40:F7:11:43:A2:2F:DE:9E:F7:34:8E:06:42:51:B1:81:18
  SHA256:
 D8:E0:FE:BC:1D:B2:E3:8D:00:94:0F:37:D2:7D:41:34:4D:99:3E:73:4B:99:D5:65:6D:97:78:D4:D8:14:36:2

Alias name: cybertrustglobalroot
Certificate fingerprints:
  MD5:  72:E4:4A:87:E3:69:40:80:77:EA:BC:E3:F4:FF:F0:E1
  SHA1: 5F:43:E5:B1:BF:F8:78:8C:AC:1C:C7:CA:4A:9A:C6:22:2B:CC:34:C6
  SHA256:
 96:0A:DF:00:63:E9:63:56:75:0C:29:65:DD:0A:08:67:DA:0B:9C:BD:6E:77:71:4A:EA:FB:23:49:AB:39:3D:A

Alias name: globalsignrootca
Certificate fingerprints:
  MD5:  3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A
  SHA1: B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
  SHA256:
 EB:D4:10:40:E4:BB:3E:C7:42:C9:E3:81:D3:1E:F2:A4:1A:48:B6:68:5C:96:E7:CE:F3:C1:DF:6C:D4:33:1C:9

Alias name: secomevrootca1
Certificate fingerprints:
  MD5:  22:2D:A6:01:EA:7C:0A:F7:F0:6C:56:43:3F:77:76:D3
  SHA1: FE:B8:C4:32:DC:F9:76:9A:CE:AE:3D:D8:90:8F:FD:28:86:65:64:7D
  SHA256:
 A2:2D:BA:68:1E:97:37:6E:2D:39:7D:72:8A:AE:3A:9B:62:96:B9:FD:BA:60:BC:2E:11:F6:47:F2:C6:75:FB:3

Alias name: globalsignr3ca
```

```
Certificate fingerprints:
  MD5:  C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28
  SHA1: D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
  SHA256:
 CB:B5:22:D7:B7:F1:27:AD:6A:01:13:86:5B:DF:1C:D4:10:2E:7D:07:59:AF:63:5A:7C:F4:72:0D:C9:63:C5:3

Alias name: staatdernederlandenrootcag3
Certificate fingerprints:
  MD5:  0B:46:67:07:DB:10:2F:19:8C:35:50:60:D1:0B:F4:37
  SHA1: D8:EB:6B:41:51:92:59:E0:F3:E7:85:00:C0:3D:B6:88:97:C9:EE:FC
  SHA256:
 3C:4F:B0:B9:5A:B8:B3:00:32:F4:32:B8:6F:53:5F:E1:72:C1:85:D0:FD:39:86:58:37:CF:36:18:7F:A6:F4:2

Alias name: staatdernederlandenrootcag2
Certificate fingerprints:
  MD5:  7C:A5:0F:F8:5B:9A:7D:6D:30:AE:54:5A:E3:42:A2:8A
  SHA1: 59:AF:82:79:91:86:C7:B4:75:07:CB:CF:03:57:46:EB:04:DD:B7:16
  SHA256:
 66:8C:83:94:7D:A6:3B:72:4B:EC:E1:74:3C:31:A0:E6:AE:D0:DB:8E:C5:B3:1B:E3:77:BB:78:4F:91:B6:71:6

Alias name: aolrootca2
Certificate fingerprints:
  MD5:  D6:ED:3C:CA:E2:66:0F:AF:10:43:0D:77:9B:04:09:BF
  SHA1: 85:B5:FF:67:9B:0C:79:96:1F:C8:6E:44:22:00:46:13:DB:17:92:84
  SHA256:
 7D:3B:46:5A:60:14:E5:26:C0:AF:FC:EE:21:27:D2:31:17:27:AD:81:1C:26:84:2D:00:6A:F3:73:06:CC:80:B

Alias name: dstrootcax3
Certificate fingerprints:
  MD5:  41:03:52:DC:0F:F7:50:1B:16:F0:02:8E:BA:6F:45:C5
  SHA1: DA:C9:02:4F:54:D8:F6:DF:94:93:5F:B1:73:26:38:CA:6A:D7:7C:13
  SHA256:
 06:87:26:03:31:A7:24:03:D9:09:F1:05:E6:9B:CF:0D:32:E1:BD:24:93:FF:C6:D9:20:6D:11:BC:D6:77:07:3

Alias name: trustcenteruniversalcai
Certificate fingerprints:
  MD5:  45:E1:A5:72:C5:A9:36:64:40:9E:F5:E4:58:84:67:8C
  SHA1: 6B:2F:34:AD:89:58:BE:62:FD:B0:6B:5C:CE:BB:9D:D9:4F:4E:39:F3
  SHA256:
 EB:F3:C0:2A:87:89:B1:FB:7D:51:19:95:D6:63:B7:29:06:D9:13:CE:0D:5E:10:56:8A:8A:77:E2:58:61:67:E

Alias name: aolrootca1
Certificate fingerprints:
  MD5:  14:F1:08:AD:9D:FA:64:E2:89:E7:1C:CF:A8:AD:7D:5E
```

```
  SHA1: 39:21:C1:15:C1:5D:0E:CA:5C:CB:5B:C4:F0:7D:21:D8:05:0B:56:6A
  SHA256:
 77:40:73:12:C6:3A:15:3D:5B:C0:0B:4E:51:75:9C:DF:DA:C2:37:DC:2A:33:B6:79:46:E9:8E:9B:FA:68:0A:E

Alias name: affirmtrustpremiumecc
Certificate fingerprints:
  MD5:  64:B0:09:55:CF:B1:D5:99:E2:BE:13:AB:A6:5D:EA:4D
  SHA1: B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
  SHA256:
 BD:71:FD:F6:DA:97:E4:CF:62:D1:64:7A:DD:25:81:B0:7D:79:AD:F8:39:7E:B4:EC:BA:9C:5E:84:88:82:14:2

Alias name: microseceszignorootca2009
Certificate fingerprints:
  MD5:  F8:49:F4:03:BC:44:2D:83:BE:48:69:7D:29:64:FC:B1
  SHA1: 89:DF:74:FE:5C:F4:0F:4A:80:F9:E3:37:7D:54:DA:91:E1:01:31:8E
  SHA256:
 3C:5F:81:FE:A5:FA:B8:2C:64:BF:A2:EA:EC:AF:CD:E8:E0:77:FC:86:20:A7:CA:E5:37:16:3D:F3:6E:DB:F3:7

Alias name: verisignclass1g3ca
Certificate fingerprints:
  MD5:  B1:47:BC:18:57:D1:18:A0:78:2D:EC:71:E8:2A:95:73
  SHA1: 20:42:85:DC:F7:EB:76:41:95:57:8E:13:6B:D4:B7:D1:E9:8E:46:A5
  SHA256:
 CB:B5:AF:18:5E:94:2A:24:02:F9:EA:CB:C0:ED:5B:B8:76:EE:A3:C1:22:36:23:D0:04:47:E4:F3:BA:55:4B:6

Alias name: certplusrootcag2
Certificate fingerprints:
  MD5:  A7:EE:C4:78:2D:1B:EE:2D:B9:29:CE:D6:A7:96:32:31
  SHA1: 4F:65:8E:1F:E9:06:D8:28:02:E9:54:47:41:C9:54:25:5D:69:CC:1A
  SHA256:
 6C:C0:50:41:E6:44:5E:74:69:6C:4C:FB:C9:F8:0F:54:3B:7E:AB:BB:44:B4:CE:6F:78:7C:6A:99:71:C4:2F:1

Alias name: certplusrootcag1
Certificate fingerprints:
  MD5:  7F:09:9C:F7:D9:B9:5C:69:69:56:D5:37:3E:14:0D:42
  SHA1: 22:FD:D0:B7:FD:A2:4E:0D:AC:49:2C:A0:AC:A6:7B:6A:1F:E3:F7:66
  SHA256:
 15:2A:40:2B:FC:DF:2C:D5:48:05:4D:22:75:B3:9C:7F:CA:3E:C0:97:80:78:B0:F0:EA:76:E5:61:A6:C7:43:3

Alias name: addtrustexternalca
Certificate fingerprints:
  MD5:  1D:35:54:04:85:78:B0:3F:42:42:4D:BF:20:73:0A:3F
  SHA1: 02:FA:F3:E2:91:43:54:68:60:78:57:69:4D:F5:E4:5B:68:85:18:68
```

```
  SHA256:
 68:7F:A4:51:38:22:78:FF:F0:C8:B1:1F:8D:43:D5:76:67:1C:6E:B2:BC:EA:B4:13:FB:83:D9:65:D0:6D:2F:F

Alias name: entrustrootcertificationauthority
Certificate fingerprints:
  MD5:  D6:A5:C3:ED:5D:DD:3E:00:C1:3D:87:92:1F:1D:3F:E4
  SHA1: B3:1E:B1:B7:40:E3:6C:84:02:DA:DC:37:D4:4D:F5:D4:67:49:52:F9
  SHA256:
 73:C1:76:43:4F:1B:C6:D5:AD:F4:5B:0E:76:E7:27:28:7C:8D:E5:76:16:C1:E6:E6:14:1A:2B:2C:BC:7D:8E:4

Alias name: verisignclass3ca
Certificate fingerprints:
  MD5:  EF:5A:F1:33:EF:F1:CD:BB:51:02:EE:12:14:4B:96:C4
  SHA1: A1:DB:63:93:91:6F:17:E4:18:55:09:40:04:15:C7:02:40:B0:AE:6B
  SHA256:
 A4:B6:B3:99:6F:C2:F3:06:B3:FD:86:81:BD:63:41:3D:8C:50:09:CC:4F:A3:29:C2:CC:F0:E2:FA:1B:14:03:0

Alias name: digicertassuredidrootca
Certificate fingerprints:
  MD5:  87:CE:0B:7B:2A:0E:49:00:E1:58:71:9B:37:A8:93:72
  SHA1: 05:63:B8:63:0D:62:D7:5A:BB:C8:AB:1E:4B:DF:B5:A8:99:B2:4D:43
  SHA256:
 3E:90:99:B5:01:5E:8F:48:6C:00:BC:EA:9D:11:1E:E7:21:FA:BA:35:5A:89:BC:F1:DF:69:56:1E:3D:C6:32:5

Alias name: globalsignrootcar3
Certificate fingerprints:
  MD5:  C5:DF:B8:49:CA:05:13:55:EE:2D:BA:1A:C3:3E:B0:28
  SHA1: D6:9B:56:11:48:F0:1C:77:C5:45:78:C1:09:26:DF:5B:85:69:76:AD
  SHA256:
 CB:B5:22:D7:B7:F1:27:AD:6A:01:13:86:5B:DF:1C:D4:10:2E:7D:07:59:AF:63:5A:7C:F4:72:0D:C9:63:C5:3

Alias name: globalsignrootcar2
Certificate fingerprints:
  MD5:  94:14:77:7E:3E:5E:FD:8F:30:BD:41:B0:CF:E7:D0:30
  SHA1: 75:E0:AB:B6:13:85:12:27:1C:04:F8:5F:DD:DE:38:E4:B7:24:2E:FE
  SHA256:
 CA:42:DD:41:74:5F:D0:B8:1E:B9:02:36:2C:F9:D8:BF:71:9D:A1:BD:1B:1E:FC:94:6F:5B:4C:99:F4:2C:1B:9

Alias name: verisignclass1ca
Certificate fingerprints:
  MD5:  86:AC:DE:2B:C5:6D:C3:D9:8C:28:88:D3:8D:16:13:1E
  SHA1: CE:6A:64:A3:09:E4:2F:BB:D9:85:1C:45:3E:64:09:EA:E8:7D:60:F1
  SHA256:
 51:84:7C:8C:BD:2E:9A:72:C9:1E:29:2D:2A:E2:47:D7:DE:1E:3F:D2:70:54:7A:20:EF:7D:61:0F:38:B8:84:2
```

```
Alias name: thawtepremiumserverca
Certificate fingerprints:
  MD5:  A6:6B:60:90:23:9B:3F:2D:BB:98:6F:D6:A7:19:0D:46
  SHA1: E0:AB:05:94:20:72:54:93:05:60:62:02:36:70:F7:CD:2E:FC:66:66
  SHA256:
 3F:9F:27:D5:83:20:4B:9E:09:C8:A3:D2:06:6C:4B:57:D3:A2:47:9C:36:93:65:08:80:50:56:98:10:5D:BC:E

Alias name: verisigntsaca
Certificate fingerprints:
  MD5:  F2:89:95:6E:4D:05:F0:F1:A7:21:55:7D:46:11:BA:47
  SHA1: 20:CE:B1:F0:F5:1C:0E:19:A9:F3:8D:B1:AA:8E:03:8C:AA:7A:C7:01
  SHA256:
 CB:6B:05:D9:E8:E5:7C:D8:82:B1:0B:4D:B7:0D:E4:BB:1D:E4:2B:A4:8A:7B:D0:31:8B:63:5B:F6:E7:78:1A:9

Alias name: thawteprimaryrootca
Certificate fingerprints:
  MD5:  8C:CA:DC:0B:22:CE:F5:BE:72:AC:41:1A:11:A8:D8:12
  SHA1: 91:C6:D6:EE:3E:8A:C8:63:84:E5:48:C2:99:29:5C:75:6C:81:7B:81
  SHA256:
 8D:72:2F:81:A9:C1:13:C0:79:1D:F1:36:A2:96:6D:B2:6C:95:0A:97:1D:B4:6B:41:99:F4:EA:54:B7:8B:FB:9

Alias name: visaecommerceroot
Certificate fingerprints:
  MD5:  FC:11:B8:D8:08:93:30:00:6D:23:F9:7E:EB:52:1E:02
  SHA1: 70:17:9B:86:8C:00:A4:FA:60:91:52:22:3F:9F:3E:32:BD:E0:05:62
  SHA256:
 69:FA:C9:BD:55:FB:0A:C7:8D:53:BB:EE:5C:F1:D5:97:98:9F:D0:AA:AB:20:A2:51:51:BD:F1:73:3E:E7:D1:2

Alias name: digicertglobalrootg3
Certificate fingerprints:
  MD5:  F5:5D:A4:50:A5:FB:28:7E:1E:0F:0D:CC:96:57:56:CA
  SHA1: 7E:04:DE:89:6A:3E:66:6D:00:E6:87:D3:3F:FA:D9:3B:E8:3D:34:9E
  SHA256:
 31:AD:66:48:F8:10:41:38:C7:38:F3:9E:A4:32:01:33:39:3E:3A:18:CC:02:29:6E:F9:7C:2A:C9:EF:67:31:D

Alias name: xrampglobalca
Certificate fingerprints:
  MD5:  A1:0B:44:B3:CA:10:D8:00:6E:9D:0F:D8:0F:92:0A:D1
  SHA1: B8:01:86:D1:EB:9C:86:A5:41:04:CF:30:54:F3:4C:52:B7:E5:58:C6
  SHA256:
 CE:CD:DC:90:50:99:D8:DA:DF:C5:B1:D2:09:B7:37:CB:E2:C1:8C:FB:2C:10:C0:FF:0B:CF:0D:32:86:FC:1A:A

Alias name: digicertglobalrootg2
```

```
Certificate fingerprints:
  MD5:  E4:A6:8A:C8:54:AC:52:42:46:0A:FD:72:48:1B:2A:44
  SHA1: DF:3C:24:F9:BF:D6:66:76:1B:26:80:73:FE:06:D1:CC:8D:4F:82:A4
  SHA256:
 CB:3C:CB:B7:60:31:E5:E0:13:8F:8D:D3:9A:23:F9:DE:47:FF:C3:5E:43:C1:14:4C:EA:27:D4:6A:5A:B1:CB:5

Alias name: valicertclass2ca
Certificate fingerprints:
  MD5:  A9:23:75:9B:BA:49:36:6E:31:C2:DB:F2:E7:66:BA:87
  SHA1: 31:7A:2A:D0:7F:2B:33:5E:F5:A1:C3:4E:4B:57:E8:B7:D8:F1:FC:A6
  SHA256:
 58:D0:17:27:9C:D4:DC:63:AB:DD:B1:96:A6:C9:90:6C:30:C4:E0:87:83:EA:E8:C1:60:99:54:D6:93:55:59:6

Alias name: geotrustprimaryca
Certificate fingerprints:
  MD5:  02:26:C3:01:5E:08:30:37:43:A9:D0:7D:CF:37:E6:BF
  SHA1: 32:3C:11:8E:1B:F7:B8:B6:52:54:E2:E2:10:0D:D6:02:90:37:F0:96
  SHA256:
 37:D5:10:06:C5:12:EA:AB:62:64:21:F1:EC:8C:92:01:3F:C5:F8:2A:E9:8E:E5:33:EB:46:19:B8:DE:B4:D0:6

Alias name: netlockaranyclassgoldfotanusitvany
Certificate fingerprints:
  MD5:  C5:A1:B7:FF:73:DD:D6:D7:34:32:18:DF:FC:3C:AD:88
  SHA1: 06:08:3F:59:3F:15:A1:04:A0:69:A4:6B:A9:03:D0:06:B7:97:09:91
  SHA256:
 6C:61:DA:C3:A2:DE:F0:31:50:6B:E0:36:D2:A6:FE:40:19:94:FB:D1:3D:F9:C8:D4:66:59:92:74:C4:46:EC:9

Alias name: geotrustglobalca
Certificate fingerprints:
  MD5:  F7:75:AB:29:FB:51:4E:B7:77:5E:FF:05:3C:99:8E:F5
  SHA1: DE:28:F4:A4:FF:E5:B9:2F:A3:C5:03:D1:A3:49:A7:F9:96:2A:82:12
  SHA256:
 FF:85:6A:2D:25:1D:CD:88:D3:66:56:F4:50:12:67:98:CF:AB:AA:DE:40:79:9C:72:2D:E4:D2:B5:DB:36:A7:3

Alias name: oistewisekeyglobalrootgbca
Certificate fingerprints:
  MD5:  A4:EB:B9:61:28:2E:B7:2F:98:B0:35:26:90:99:51:1D
  SHA1: 0F:F9:40:76:18:D3:D7:6A:4B:98:F0:A8:35:9E:0C:FD:27:AC:CC:ED
  SHA256:
 6B:9C:08:E8:6E:B0:F7:67:CF:AD:65:CD:98:B6:21:49:E5:49:4A:67:F5:84:5E:7B:D1:ED:01:9F:27:B8:6B:D

Alias name: certumtrustednetworkca2
Certificate fingerprints:
  MD5:  6D:46:9E:D9:25:6D:08:23:5B:5E:74:7D:1E:27:DB:F2
```

```
  SHA1: D3:DD:48:3E:2B:BF:4C:05:E8:AF:10:F5:FA:76:26:CF:D3:DC:30:92
  SHA256:
 B6:76:F2:ED:DA:E8:77:5C:D3:6C:B0:F6:3C:D1:D4:60:39:61:F4:9E:62:65:BA:01:3A:2F:03:07:B6:D0:B8:0

Alias name: starfieldservicesrootcertificateauthorityg2
Certificate fingerprints:
  MD5:  17:35:74:AF:7B:61:1C:EB:F4:F9:3C:E2:EE:40:F9:A2
  SHA1: 92:5A:8F:8D:2C:6D:04:E0:66:5F:59:6A:FF:22:D8:63:E8:25:6F:3F
  SHA256:
 56:8D:69:05:A2:C8:87:08:A4:B3:02:51:90:ED:CF:ED:B1:97:4A:60:6A:13:C6:E5:29:0F:CB:2A:E6:3E:DA:B

Alias name: comodorsacertificationauthority
Certificate fingerprints:
  MD5:  1B:31:B0:71:40:36:CC:14:36:91:AD:C4:3E:FD:EC:18
  SHA1: AF:E5:D2:44:A8:D1:19:42:30:FF:47:9F:E2:F8:97:BB:CD:7A:8C:B4
  SHA256:
 52:F0:E1:C4:E5:8E:C6:29:29:1B:60:31:7F:07:46:71:B8:5D:7E:A8:0D:5B:07:27:34:63:53:4B:32:B4:02:3

Alias name: comodoaaaca
Certificate fingerprints:
  MD5:  49:79:04:B0:EB:87:19:AC:47:B0:BC:11:51:9B:74:D0
  SHA1: D1:EB:23:A4:6D:17:D6:8F:D9:25:64:C2:F1:F1:60:17:64:D8:E3:49
  SHA256:
 D7:A7:A0:FB:5D:7E:27:31:D7:71:E9:48:4E:BC:DE:F7:1D:5F:0C:3E:0A:29:48:78:2B:C8:3E:E0:EA:69:9E:F

Alias name: identrustpublicsectorrootca1
Certificate fingerprints:
  MD5:  37:06:A5:B0:FC:89:9D:BA:F4:6B:8C:1A:64:CD:D5:BA
  SHA1: BA:29:41:60:77:98:3F:F4:F3:EF:F2:31:05:3B:2E:EA:6D:4D:45:FD
  SHA256:
 30:D0:89:5A:9A:44:8A:26:20:91:63:55:22:D1:F5:20:10:B5:86:7A:CA:E1:2C:78:EF:95:8F:D4:F4:38:9F:2

Alias name: certplusclass2primaryca
Certificate fingerprints:
  MD5:  88:2C:8C:52:B8:A2:3C:F3:F7:BB:03:EA:AE:AC:42:0B
  SHA1: 74:20:74:41:72:9C:DD:92:EC:79:31:D8:23:10:8D:C2:81:92:E2:BB
  SHA256:
 0F:99:3C:8A:EF:97:BA:AF:56:87:14:0E:D5:9A:D1:82:1B:B4:AF:AC:F0:AA:9A:58:B5:D5:7A:33:8A:3A:FB:C

Alias name: ttelesecglobalrootclass2ca
Certificate fingerprints:
  MD5:  2B:9B:9E:E4:7B:6C:1F:00:72:1A:CC:C1:77:79:DF:6A
  SHA1: 59:0D:2D:7D:88:4F:40:2E:61:7E:A5:62:32:17:65:CF:17:D8:94:E9
```

```
  SHA256:
 91:E2:F5:78:8D:58:10:EB:A7:BA:58:73:7D:E1:54:8A:8E:CA:CD:01:45:98:BC:0B:14:3E:04:1B:17:05:25:5

Alias name: accvraiz1
Certificate fingerprints:
  MD5:  D0:A0:5A:EE:05:B6:09:94:21:A1:7D:F1:B2:29:82:02
  SHA1: 93:05:7A:88:15:C6:4F:CE:88:2F:FA:91:16:52:28:78:BC:53:64:17
  SHA256:
 9A:6E:C0:12:E1:A7:DA:9D:BE:34:19:4D:47:8A:D7:C0:DB:18:22:FB:07:1D:F1:29:81:49:6E:D1:04:38:41:1

Alias name: digicerthighassuranceevrootca
Certificate fingerprints:
  MD5:  D4:74:DE:57:5C:39:B2:D3:9C:85:83:C5:C0:65:49:8A
  SHA1: 5F:B7:EE:06:33:E2:59:DB:AD:0C:4C:9A:E6:D3:8F:1A:61:C7:DC:25
  SHA256:
 74:31:E5:F4:C3:C1:CE:46:90:77:4F:0B:61:E0:54:40:88:3B:A9:A0:1E:D0:0B:A6:AB:D7:80:6E:D3:B1:18:C

Alias name: amzninternalinfoseccag3
Certificate fingerprints:
  MD5:  E9:34:94:02:BA:BB:31:6B:22:E6:2B:A9:C4:F0:26:04
  SHA1: B9:B1:CA:38:F7:BF:9C:D2:D4:95:E7:B6:5E:75:32:9B:A8:78:2E:F6
  SHA256:
 81:03:0B:C7:E2:54:DA:7B:F8:B7:45:DB:DD:41:15:89:B5:A3:81:86:FB:4B:29:77:1F:84:0A:18:D9:67:6D:6

Alias name: cia-crt-g3-02-ca
Certificate fingerprints:
  MD5:  FD:B9:23:FD:D3:EB:2D:3E:57:EF:56:FF:DB:D3:E4:B9
  SHA1: 96:4A:BB:A7:BD:DA:FC:97:34:C0:0A:2D:F0:05:98:F7:E6:C6:6F:09
  SHA256:
 93:F1:72:FB:BA:43:31:5C:06:EE:0F:9F:04:89:B8:F6:88:BC:75:15:3C:BE:B4:80:AC:A7:14:3A:F6:FC:4A:C

Alias name: entrustrootcertificationauthorityec1
Certificate fingerprints:
  MD5:  B6:7E:1D:F0:58:C5:49:6C:24:3B:3D:ED:98:18:ED:BC
  SHA1: 20:D8:06:40:DF:9B:25:F5:12:25:3A:11:EA:F7:59:8A:EB:14:B5:47
  SHA256:
 02:ED:0E:B2:8C:14:DA:45:16:5C:56:67:91:70:0D:64:51:D7:FB:56:F0:B2:AB:1D:3B:8E:B0:70:E5:6E:DF:F

Alias name: securitycommunicationrootca
Certificate fingerprints:
  MD5:  F1:BC:63:6A:54:E0:B5:27:F5:CD:E7:1A:E3:4D:6E:4A
  SHA1: 36:B1:2B:49:F9:81:9E:D7:4C:9E:BC:38:0F:C6:56:8F:5D:AC:B2:F7
  SHA256:
 E7:5E:72:ED:9F:56:0E:EC:6E:B4:80:00:73:A4:3F:C3:AD:19:19:5A:39:22:82:01:78:95:97:4A:99:02:6B:6
```

```
Alias name: globalsignca
Certificate fingerprints:
  MD5:  3E:45:52:15:09:51:92:E1:B7:5D:37:9F:B1:87:29:8A
  SHA1: B1:BC:96:8B:D4:F4:9D:62:2A:A8:9A:81:F2:15:01:52:A4:1D:82:9C
  SHA256:
 EB:D4:10:40:E4:BB:3E:C7:42:C9:E3:81:D3:1E:F2:A4:1A:48:B6:68:5C:96:E7:CE:F3:C1:DF:6C:D4:33:1C:9

Alias name: trustcenterclass2caii
Certificate fingerprints:
  MD5:  CE:78:33:5C:59:78:01:6E:18:EA:B9:36:A0:B9:2E:23
  SHA1: AE:50:83:ED:7C:F4:5C:BC:8F:61:C6:21:FE:68:5D:79:42:21:15:6E
  SHA256:
 E6:B8:F8:76:64:85:F8:07:AE:7F:8D:AC:16:70:46:1F:07:C0:A1:3E:EF:3A:1F:F7:17:53:8D:7A:BA:D3:91:B

Alias name: camerfirmachambersofcommerceroot
Certificate fingerprints:
  MD5:  B0:01:EE:14:D9:AF:29:18:94:76:8E:F1:69:33:2A:84
  SHA1: 6E:3A:55:A4:19:0C:19:5C:93:84:3C:C0:DB:72:2E:31:30:61:F0:B1
  SHA256:
 0C:25:8A:12:A5:67:4A:EF:25:F2:8B:A7:DC:FA:EC:EE:A3:48:E5:41:E6:F5:CC:4E:E6:3B:71:B3:61:60:6A:C

Alias name: geotrustprimarycag3
Certificate fingerprints:
  MD5:  B5:E8:34:36:C9:10:44:58:48:70:6D:2E:83:D4:B8:05
  SHA1: 03:9E:ED:B8:0B:E7:A0:3C:69:53:89:3B:20:D2:D9:32:3A:4C:2A:FD
  SHA256:
 B4:78:B8:12:25:0D:F8:78:63:5C:2A:A7:EC:7D:15:5E:AA:62:5E:E8:29:16:E2:CD:29:43:61:88:6C:D1:FB:D

Alias name: geotrustprimarycag2
Certificate fingerprints:
  MD5:  01:5E:D8:6B:BD:6F:3D:8E:A1:31:F8:12:E0:98:73:6A
  SHA1: 8D:17:84:D5:37:F3:03:7D:EC:70:FE:57:8B:51:9A:99:E6:10:D7:B0
  SHA256:
 5E:DB:7A:C4:3B:82:A0:6A:87:61:E8:D7:BE:49:79:EB:F2:61:1F:7D:D7:9B:F9:1C:1C:6B:56:6A:21:9E:D7:6

Alias name: hongkongpostrootca1
Certificate fingerprints:
  MD5:  A8:0D:6F:39:78:B9:43:6D:77:42:6D:98:5A:CC:23:CA
  SHA1: D6:DA:A8:20:8D:09:D2:15:4D:24:B5:2F:CB:34:6E:B2:58:B2:8A:58
  SHA256:
 F9:E6:7D:33:6C:51:00:2A:C0:54:C6:32:02:2D:66:DD:A2:E7:E3:FF:F1:0A:D0:61:ED:31:D8:BB:B4:10:CF:B

Alias name: affirmtrustpremiumeccca
```

```
Certificate fingerprints:
  MD5:  64:B0:09:55:CF:B1:D5:99:E2:BE:13:AB:A6:5D:EA:4D
  SHA1: B8:23:6B:00:2F:1D:16:86:53:01:55:6C:11:A4:37:CA:EB:FF:C3:BB
  SHA256:
 BD:71:FD:F6:DA:97:E4:CF:62:D1:64:7A:DD:25:81:B0:7D:79:AD:F8:39:7E:B4:EC:BA:9C:5E:84:88:82:14:2

Alias name: hellenicacademicandresearchinstitutionsrootca2015
Certificate fingerprints:
  MD5:  CA:FF:E2:DB:03:D9:CB:4B:E9:0F:AD:84:FD:7B:18:CE
  SHA1: 01:0C:06:95:A6:98:19:14:FF:BF:5F:C6:B0:B6:95:EA:29:E9:12:A6
  SHA256:
 A0:40:92:9A:02:CE:53:B4:AC:F4:F2:FF:C6:98:1C:E4:49:6F:75:5E:6D:45:FE:0B:2A:69:2B:CD:52:52:3F:3
```

IoT Analytics

AWS IoT Analytics (`iotAnalytics`) 操作将数据从 MQTT 消息发送到 AWS IoT Analytics 频道。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`iotanalytics:BatchPutMessage`操作的 IAM 角色。有关更多信息，请参阅 [授予 AWS IoT 规则所需的访问权限]。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。

  附加到您指定角色的策略应如下所示：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotanalytics:BatchPutMessage",
            "Resource": [
                "arn:aws:iotanalytics:us-west-2:account-id:channel/mychannel"
            ]
        }
    ]
}
```

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`batchMode`

（可选）是否批处理操作。默认值为 `false`。

当`batchMode`为`true`且规则 SQL 语句的计算结果为数组时，每个数组元素在传递到 AWS IoT Analytics 通道时都[BatchPutMessage](#)将作为单独的消息传送。生成的数组，其消息不得超过 100 条。

支持[替换模板](#)：否

`channelName`

要向其写入数据的 AWS IoT Analytics 通道的名称。

支持[替换模板](#)：API 且 AWS CLI 仅支持

`roleArn`

允许访问 AWS IoT Analytics 频道的 IAM 角色。有关更多信息，请参阅 [要求](#)。

支持[替换模板](#)：否

示例

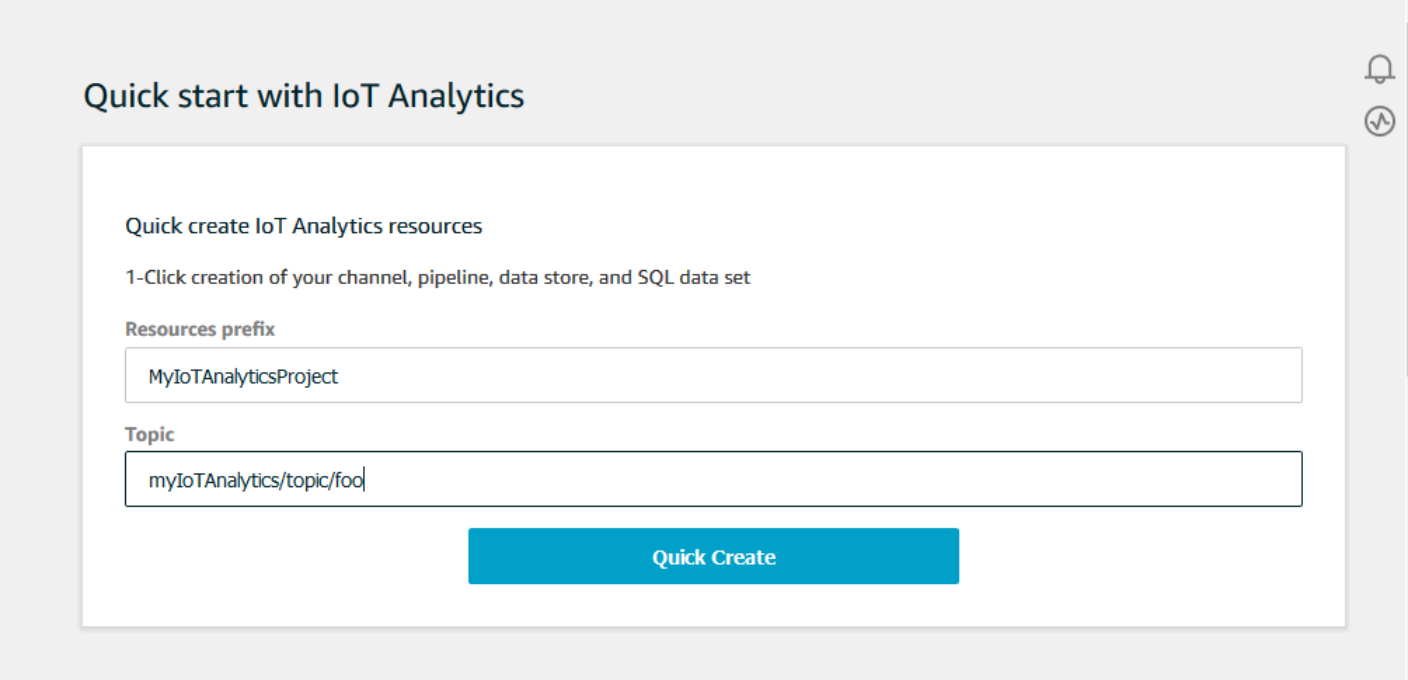
以下 JSON 示例定义了 AWS IoT 规则中的 AWS IoT Analytics 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotAnalytics": {
                    "channelName": "mychannel",
                    "roleArn": "arn:aws:iam::123456789012:role/analyticsRole",
                }
            }
        ]
    }
```

```
}
```

另请参阅

- 什么是 AWS IoT Analytics？ 在《AWS IoT Analytics 用户指南》中
- 该 AWS IoT Analytics 控制台还具有快速启动功能，可让您一键创建频道、数据存储、管道和数据存储。有关更多信息，请参阅 AWS IoT Analytics 用户指南中的 AWS IoT Analytics 控制台快速入门指南。



AWS IoT Events

AWS IoT Events (`iotEvents`) 操作将数据从 MQTT 消息发送到 AWS IoT Events 输入。

> **Important**
>
> 如果将有效负载发送到时 AWS IoT Core 不带有`Input attribute Key`，或者密钥不在密钥中指定的 JSON 路径中，则会导致物联网规则失败并出现错误`Failed to send message to Iot Events`。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`iotevents:BatchPutMessage`操作的 IAM 角色。有关更多信息，请参阅[授予 AWS IoT 规则所需的访问权限](#)。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`batchMode`

（可选）是否批处理事件操作。默认值为 `false`。

如果 `batchMode` 为 `true` 且规则 SQL 语句的求值为一个 Array 时，通过调用 [BatchPutMessage](#) 将 Array 元素发送到 AWS IoT Events 时，会将每个 Array 元素都视为单独的消息。生成的数组，其消息不得超过 10 条。

`batchMode` 为 `true` 时，不能指定 `messageId`。

支持[替换模板](#)：否

`inputName`

AWS IoT Events 输入的名称。

支持[替换模板](#)：API 且 AWS CLI 仅支持

`messageId`

（可选）使用它来验证 AWS IoT Events 检测器`messageId`是否只处理一个给定输入（消息）。您可以使用 `${newuuid()}` 替代模板，以便为每个请求生成一个唯一的 ID。

当 `batchMode` 为 `true` 时，您无法指定 `messageId` -- 将分配一个新 UUID 值。

支持[替换模板](#)：是

`roleArn`

允许 AWS IoT 向 AWS IoT Events 检测器发送输入的 IAM 角色。有关更多信息，请参阅 [要求](#)。

支持[替换模板](#)：否

示例

下面的 JSON 示例介绍了如何在 AWS IoT 规则中定义 IoT Events 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotEvents": {
                    "inputName": "MyIoTEventsInput",
                    "messageId": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_events"
                }
            }
        ]
    }
}
```

另请参阅

- 什么是 AWS IoT Events？ 在《AWS IoT Events 开发者指南》中

AWS IoT SiteWise

AWS IoT SiteWise (`iotSiteWise`) 操作将数据从 MQTT 消息发送到中的资产属性。 AWS IoT SiteWise

你可以遵循一个教程，向你展示如何从 AWS IoT 事物中提取数据。有关更多信息，请参阅《用户指南》中的 “ AWS IoT SiteWise 从 AWS IoT 事物中提取数据” 教程或 “使用 AWS IoT 核心规则提取数据” 部分。AWS IoT SiteWise

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`iotsitewise:BatchPutAssetPropertyValue`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  您可以将下列示例信任策略附加到要代入的角色。

  ```
  {
      "Version": "2012-10-17",
  ```

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*"
        }
    ]
}
```

为了提高安全性，您可以在Condition属性中指定 AWS IoT SiteWise 资产层次结构路径。以下示例是指定资产层次结构路径的信任策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotsitewise:BatchPutAssetPropertyValue",
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
                        "/root node asset ID/*"
                    ]
                }
            }
        }
    ]
}
```

- 当您通过此操作向发送数据时，您的数据必须满足BatchPutAssetPropertyValue操作的要求。AWS IoT SiteWise 有关更多信息，请参阅 AWS IoT SiteWise API 参考中的BatchPutAssetProperty值。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`putAssetPropertyValueEntries`

包含以下信息的资产属性值条目的列表：

`propertyAlias`

（可选）与您的资产属性关联的属性别名。指定 `propertyAlias` 或同时指定 `assetId` 和 `propertyId`。有关更多信息，请参阅 AWS IoT SiteWise 用户指南中的将工业数据流映射到资产属性。

支持替换模板：是

`assetId`

（可选）AWS IoT SiteWise 资产的 ID。指定 `propertyAlias` 或同时指定 `assetId` 和 `propertyId`。

支持替换模板：是

`propertyId`

（可选）资产属性的 ID。指定 `propertyAlias` 或同时指定 `assetId` 和 `propertyId`。

支持替换模板：是

`entryId`

（可选）此条目的唯一标识符。定义 `entryId`，以便在出现故障时更好地跟踪哪条消息导致了错误。默认为新 UUID。

支持替换模板：是

`propertyValues`

要插入的属性值的列表，均包含以下格式的时间戳、质量和值 (TQV)：

`timestamp`

包含以下信息的时间戳结构：

`timeInSeconds`

包含用 Unix 纪元时间表示的时间（以秒为单位）的字符串。如果您的消息负载没有时间戳，则可使用 timestamp()，它将返回当前时间（以毫秒为单位）。要将该时间转换为秒，可以使用以下替换模板：**`${floor(timestamp() / 1E3)}`**。

支持替换模板：是

`offsetInNanos`

（可选）包含与以秒为单位的时间的纳秒时间偏移量的字符串。如果您的消息负载没有时间戳，则可使用 timestamp()，它将返回当前时间（以毫秒为单位）。要计算与该时间的纳秒偏移量，可以使用以下替换模板：**`${(timestamp() % 1E3) * 1E6}`**。

支持替换模板：是

关于 Unix 纪元时间，仅 AWS IoT SiteWise 接受时间戳最长为过去 7 天、将来最多 5 分钟的条目。

`quality`

（可选）描述值的质量的字符串。有效值：`GOOD`、`BAD`、`UNCERTAIN`。

支持替换模板：是

`value`

包含以下值字段之一的值结构，具体取决于资产属性的数据类型：

`booleanValue`

（可选）包含值条目布尔值的字符串。

支持替换模板：是

`doubleValue`

（可选）包含值条目的双值的字符串。

支持替换模板：是

`integerValue`

（可选）包含值条目整数值的字符串。

支持替换模板：是

`stringValue`

（可选）值条目的字符串值。

支持替换模板：是

`roleArn`

授予向发送资产属性值的 AWS IoT 权限的 IAM 角色的 ARN。 AWS IoT SiteWise有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例在 AWS IoT 规则中定义了基本的 IoT SiteWise 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotSiteWise": {
                    "putAssetPropertyValueEntries": [
                        {
                            "propertyAlias": "/some/property/alias",
                            "propertyValues": [
                                {
                                    "timestamp": {
                                        "timeInSeconds": "${my.payload.timeInSeconds}"
                                    },
                                    "value": {
                                        "integerValue": "${my.payload.value}"
                                    }
                                }
                            ]
                        }
                    ],
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sitewise"
                }
            }
        ]
    }
}
```

以下 JSON 示例在 AWS IoT 规则中定义了物联网 SiteWise 操作。此示例将主题用作属性别名和 `timestamp()` 函数。例如，如果您将数据发布到 `/company/windfarm/3/turbine/7/rpm`，则此操作会将数据发送到具有与您指定的主题相同的属性别名的资产属性。

```
{
    "topicRulePayload": {
```

```
        "sql": "SELECT * FROM '/company/windfarm/+/turbine/+/+'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "iotSiteWise": {
                    "putAssetPropertyValueEntries": [
                        {
                            "propertyAlias": "${topic()}",
                            "propertyValues": [
                                {
                                    "timestamp": {
                                        "timeInSeconds": "${floor(timestamp() / 1E3)}",
                                        "offsetInNanos": "${(timestamp() % 1E3) * 1E6}"
                                    },
                                    "value": {
                                        "doubleValue": "${my.payload.value}"
                                    }
                                }
                            ]
                        }
                    ],
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sitewise"
                }
            }
        ]
    }
}
```

另请参阅

- AWS IoT SiteWise 用户指南 中的什么是 AWS IoT SiteWise？
- 使用用户指南中的 AWS IoT Core 规则摄取数据AWS IoT SiteWise
- AWS IoT SiteWise从《用户指南》中的 AWS IoT 事物中AWS IoT SiteWise 提取数据
- AWS IoT SiteWise 用户指南中的 AWS IoT SiteWise 规则操作疑难解答

Firehose

Firehose (`firehose`) 操作将数据从 MQTT 消息发送到亚马逊数据 Firehose 流。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`firehose:PutRecord`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。
- 如果您使用 Firehose 将数据发送到亚马逊 S3 存储桶，并且您使用托管 AWS KMS AWS KMS key 客户加密亚马逊 S3 中的静态数据，则 Firehose 必须有权访问您的存储桶，并有权 AWS KMS key 代表调用者使用这些存储桶。有关更多信息，请参阅《亚马逊数据 Firehose 开发者指南》中的授予 Fi rehose 访问亚马逊 S3 目标的权限。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`batchMode`

（可选）是否使用批量传送 Firehose 直播。`PutRecordBatch`默认值为 `false`。

当 `batchMode` 为 `true` 且规则的 SQL 语句求值为一个数组时，每个数组元素在 `PutRecordBatch` 请求中形成一条记录。生成的数组，其记录不得超过 500 条。

支持替换模板：否

`deliveryStreamName`

要向其写入消息数据的 Firehose 流。

支持替换模板：API 且 AWS CLI 仅支持

`separator`

（可选）一种字符分隔符，用于分隔写入 Firehose 流的记录。如果省略此参数，则流不使用分隔符。有效值：`,`（逗号），`\t`（选项卡），`\n`（换行符），`\r\n`（窗口换行符）。

支持替换模板：否

`roleArn`

允许访问 Firehose 直播的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例在规则中定义了 Fireh AWS IoT ose 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "firehose": {
                    "deliveryStreamName": "my_firehose_stream",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose"
                }
            }
        ]
    }
}
```

以下 JSON 示例使用规则中的替换模板定义了 Firehose AWS IoT 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "firehose": {
                    "deliveryStreamName": "${topic()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose"
                }
            }
        ]
    }
}
```

另请参阅

- 什么是 Amazon Data Firehose？ 在 Amazon Data Firehos e 开发者指南中

Kinesis Data Streams

Kinesis Data Streams (`kinesis`) 操作将 MQTT 消息中的数据写入 Amazon Kinesis Data Streams。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`kinesis:PutRecord`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。
- 如果您使用 AWS KMS 客户管理的 AWS KMS key （ KMS 密钥）对 Kinesis Data Streams 中的静态数据进行加密，则该服务必须有权代表调用方使用 AWS KMS key 。有关更多信息，请参阅 Amazon Kinesis Data Streams 开发人员指南中的使用用户生成的 AWS KMS keys的权限

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`stream`

数据写入的 Kinesis 数据流。

支持替换模板：API 且 AWS CLI 仅支持

`partitionKey`

用于确定将数据写入哪个分区的分区键。分区键通常由表达式（例如，`${topic()}` 或 `${timestamp()}`）组成。

支持替换模板：是

`roleArn`

授予访问 Kinesis 数据流 AWS IoT 权限的 IAM 角色的 ARN。有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例在规则中定义了 Kinesis Data Streams 操作 AWS IoT 。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "kinesis": {
                    "streamName": "my_kinesis_stream",
                    "partitionKey": "${topic()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis"
                }
            }
        ]
    }
}
```

以下 JSON 示例使用规则中的替换模板定义了 Kinesis 操作。 AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "kinesis": {
                    "streamName": "${topic()}",
                    "partitionKey": "${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis"
                }
            }
        ]
    }
}
```

另请参阅

- Amazon Kinesis Data Streams 开发人员指南中的什么是 Amazon Kinesis Data Streams？

Lambda

Lambda (`lambda`) 操作会调用一个 AWS Lambda 函数，传入 MQTT 消息。 AWS IoT 异步调用 Lambda 函数。

您可以按照教程执行操作，该教程向您说明如何使用 Lambda 操作创建并测试规则。有关更多信息，请参阅 教程：使用 AWS Lambda 函数格式化通知。

要求

此规则操作具有以下要求：

- AWS IoT 要调用 Lambda 函数，必须配置授予`lambda:InvokeFunction`权限的策略。 AWS IoT 您只能调用 Lambda 策略 AWS 区域 所在位置定义的 Lambda 函数。Lambda 函数使用基于资源的策略，因此您必须将该策略附加到 Lambda 函数本身。

  使用以下 AWS CLI 命令附加授予`lambda:InvokeFunction`权限的策略。

  ```
  aws lambda add-permission --function-name function_name --region region --principal
   iot.amazonaws.com --source-arn arn:aws:iot:region:account-id:rule/rule_name --
  source-account account-id --statement-id unique_id --action "lambda:InvokeFunction"
  ```

  `add-permission` 命令需要以下参数：

  `--function-name`

  Lambda 函数的名称 添加新的权限来更新函数的资源策略。

  `--region`

  函数 AWS 区域 的。

  `--principal`

  获取权限的委托人。这应该是`iot.amazonaws.com`为了 AWS IoT 允许调用 Lambda 函数。

  `--source-arn`

  规则的 ARN。您可以使用`get-topic-rule` AWS CLI 命令获取规则的 ARN。

  `--source-account`

  规则 AWS 账户 的定义位置。

`--statement-id`

唯一的语句标识符。

`--action`

要在此声明中允许的 Lambda 操作。 AWS IoT 要允许调用 Lambda 函数，请指定。`lambda:InvokeFunction`

> **Important**
>
> 如果您在不提供`source-arn`或的情况下为 AWS IoT 委托人添加权限`source-account`，则任何 AWS 账户 使用您的 Lambda 操作创建规则的人都可以激活规则，从中调用您的 Lambda 函数。 AWS IoT

有关更多信息，请参阅 AWS Lambda 权限。

- 如果您使用 AWS KMS 客户托管 AWS KMS key 对 Lambda 中的静态数据进行加密，则该服务必须有权代表调用者使用这些数据。 AWS KMS key 有关更多信息，请参阅 AWS Lambda 开发人员指南中的静态加密。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`functionArn`

要调用的 Lambda 函数的 ARN。 AWS IoT 必须具有调用该函数的权限。有关更多信息，请参阅 要求。

如果您未指定 Lambda 函数的版本或别名，则将关闭该函数的最新版本。如果要关闭 Lambda 函数的特定版本，则可指定版本或别名。要指定一个版本或别名，请将该版本或别名附加到 Lambda 函数的 ARN。

```
arn:aws:lambda:us-east-2:123456789012:function:myLambdaFunction:someAlias
```

有关版本控制和别名的更多信息，请参阅 AWS Lambda 函数版本控制和别名。

支持替换模板：API 且 AWS CLI 仅支持

示例

以下 JSON 示例在规则中定义了 Lambda 操作。 AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-
east-2:123456789012:function:myLambdaFunction"
                }
            }
        ]
    }
}
```

以下 JSON 示例定义了在规则中使用替换模板的 Lambda 操作。 AWS IoT

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "lambda": {
                    "functionArn": "arn:aws:lambda:us-east-1:123456789012:function:
${topic()}"
                }
            }
        ]
    }
}
```

另请参阅

- 什么是 AWS Lambda？ 在《AWS Lambda 开发者指南》中
- 教程：使用 AWS Lambda 函数格式化通知

位置

Location (`location`) 操作会将您的地理位置数据路由到 Amazon Location Service。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`geo:BatchUpdateDevicePosition`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`deviceId`

提供位置数据的设备的唯一 ID。有关更多信息，请参阅 Amazon Location Service API 参考 中的 `DeviceId`。

支持替换模板：是

`latitude`

计算结果为表示设备位置纬度的双精度值的字符串。

支持替换模板：是

`longitude`

计算结果为表示设备位置经度的双精度值的字符串。

支持替换模板：是

`roleArn`

允许访问 Amazon Location Service 域的 IAM 角色。有关更多信息，请参阅 要求。

`timestamp`

对位置数据进行采样的时间。默认值是处理 MQTT 消息的时间。

`timestamp` 值由以下两个值组成：

- `value`：返回长纪元时间值的表达式。您可以使用 the section called “time_to_epoch(String, String)” 函数从消息负载中传递的日期或时间值创建有效的时间戳。支持替换模板：是。
- `unit`：（可选）`value` 中描述的表达式生成的时间戳值的精度。有效值：`SECONDS` | `MILLISECONDS` |`MICROSECONDS` | `NANOSECONDS`。默认值为 `MILLISECONDS`。仅支持替换模板：API 且 AWS CLI 仅支持。

`trackerName`

Amazon Location 中更新位置的跟踪器资源的名称。有关更多信息，请参阅 Amazon Location Service 开发人员指南 中的跟踪器。

支持替换模板：API 且 AWS CLI 仅支持

示例

以下 JSON 示例在 AWS IoT 规则中定义了定位操作。

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "location": {
     "roleArn": "arn:aws:iam::123454962127:role/service-role/ExampleRole",
     "trackerName": "MyTracker",
     "deviceId": "001",
     "sampleTime": {
      "value": "${timestamp()}",
      "unit": "MILLISECONDS"
     },
     "latitude": "-12.3456",
     "longitude": "65.4321"
    }
   }
  ]
 }
}
```

以下 JSON 示例定义了在 AWS IoT 规则中使用替换模板的定位操作。

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "location": {
     "roleArn": "arn:aws:iam::123456789012:role/service-role/ExampleRole",
     "trackerName": "${TrackerName}",
     "deviceId": "${DeviceID}",
     "timestamp": {
      "value": "${timestamp()}",
      "unit": "MILLISECONDS"
     },
     "latitude": "${get(position, 0)}",
     "longitude": "${get(position, 1)}"
    }
   }
  ]
 }
}
```

以下 MQTT 有效负载示例显示了上面示例中的替换模板如何访问数据。您可以使用 get-device-position-history CLI 命令来验证 MQTT 有效负载数据是否已在位置跟踪器中传递。

```
{
 "TrackerName": "mytracker",
 "DeviceID": "001",
 "position": [
  "-12.3456",
  "65.4321"
 ]
}
```

```
aws location get-device-position-history --device-id 001 --tracker-name mytracker
```

```
{
 "DevicePositions": [
  {
   "DeviceId": "001",
```

```
    "Position": [
     -12.3456,
     65.4321
    ],
    "ReceivedTime": "2022-11-11T01:31:54.464000+00:00",
    "SampleTime": "2022-11-11T01:31:54.308000+00:00"
   }
  ]
}
```

另请参阅

- Amazon Location Service 开发人员指南 中的什么是 Amazon Location Service？。

OpenSearch

OpenSearch (`openSearch`) 操作将来自 MQTT 消息的数据写入亚马逊 OpenSearch 服务域。然后，您可以使用 OpenSearch 仪表板等工具在 Ser OpenSearch vice 中查询和可视化数据。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`es:ESHttpPut`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。
- 如果您使用客户管理 AWS KMS key 来加密 OpenSearch 服务中的静态数据，则该服务必须有权代表呼叫者使用 KMS 密钥。有关更多信息，请参阅《亚马逊服务开发者指南》中的亚马逊 OpenSearch 服务静态数据加密。 OpenSearch

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`endpoint`

您的亚马逊 OpenSearch 服务域的终端节点。

支持替换模板：API 且 AWS CLI 仅支持

`index`

您要存储数据的 OpenSearch 索引。

支持替换模板：是

`type`

您存储的文档类型。

> **Note**
>
> 对于高于 1.0 的 OpenSearch 版本，`type`参数的值必须为`_doc`。有关更多信息，请参阅OpenSearch 文档。

支持替换模板：是

`id`

每个文档的唯一标识符。

支持替换模板：是

`roleARN`

允许访问 OpenSearch 服务域的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

限制

OpenSearch (`openSearch`) 操作不能用于向 VPC Elasticsearch 集群传输数据。

示例

以下 JSON 示例定义了 AWS IoT 规则中的 OpenSearch 操作以及如何为该`OpenSearch`操作指定字段。有关更多信息，请参阅OpenSearch操作。

```
{
    "topicRulePayload": {
```

```
        "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "openSearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "my-index",
                    "type": "_doc",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_os"
                }
            }
        ]
    }
}
```

以下 JSON 示例定义了一个在 AWS IoT 规则中使用替换模板的 OpenSearch 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "openSearch": {
                    "endpoint": "https://my-endpoint",
                    "index": "${topic()}",
                    "type": "${type}",
                    "id": "${newuuid()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_os"
                }
            }
        ]
    }
}
```

> **Note**
>
> 替换的`type`字段适用于 OpenSearch 版本 1.0。对于任何高于 1.0 的版本，的值`type`必须为`_doc`。

另请参阅

什么是亚马逊 OpenSearch 服务？ 在《亚马逊 OpenSearch 服务开发者指南》中

Republish

重新发布 (`republish`) 操作可将 MQTT 消息重新发布到其它 MQTT 主题。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`iot:Publish`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`headers`

MQTT 版本 5.0 标头信息。

有关更多信息，请参阅 AWS API 参考MqttHeaders中的RepublishAction和。

`topic`

消息重新发布到的 MQTT 主题。

若要重新发布到以 $ 开头的预留主题，请使用 $$ 代替。例如，如果要重新发布到设备影子主题 `$aws/things/MyThing/shadow/update`，请将主题指定为 `$$aws/things/MyThing/shadow/update`。

> **Note**
>
> 不支持重新发布到保留的任务主题。
> AWS IoT Device Defender 保留主题不支持 HTTP 发布。

支持替换模板：是

`qos`

（可选）重新发布消息时要使用的服务质量 (QoS) 级别。有效值：`0`、`1`。默认值为 `0`。有关 MQTT QoS 的更多信息，请参阅 MQTT。

支持替换模板：否

`roleArn`

允许发布 AWS IoT 到 MQTT 主题的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例在 AWS IoT 规则中定义了重新发布操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "another/topic",
                    "qos": 1,
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
                }
            }
        ]
    }
}
```

以下 JSON 示例定义了在 AWS IoT 规则中使用替换模板的重新发布操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
```

```
        "actions": [
            {
                "republish": {
                    "topic": "${topic()}/republish",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"
                }
            }
        ]
    }
}
```

以下 JSON 示例在 AWS IoT 规则中使用 headers 定义了重新发布操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "republish": {
                    "topic": "${topic()}/republish",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish",
                    "headers": {
                        "payloadFormatIndicator": "UTF8_DATA",
                        "contentType": "rule/contentType",
                        "correlationData": "cnVsZSBjb3JyZWxhdGlvbiBkYXRh",
                        "userProperties": [
                            {
                                "key": "ruleKey1",
                                "value": "ruleValue1"
                            },
                            {
                                "key": "ruleKey2",
                                "value": "ruleValue2"
                            }
                        ]
                    }
                }
            }
        ]
    }
}
```

> **Note**
>
> 不会通过重新发布操作传递原始源 IP。

S3

S3 (s3) 操作将 MQTT 消息中的数据写入 Amazon Simple Storage Service (Amazon S3) 存储桶。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`s3:PutObject`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。
- 如果您使用 AWS KMS 客户托管 AWS KMS key 来加密Amazon S3中的静态数据，则该服务必须有权代表呼叫者使用这些数据。 AWS KMS key 有关更多信息，请参阅《Amazon 简单存储服务开发者指南》 AWS KMS keys中的托AWS 管 AWS KMS keys 和客户托管。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`bucket`

数据写入的 Amazon S3 存储桶。

支持替换模板： AWS CLI 仅支持 API

`cannedacl`

（可选）Amazon S3 封装的 ACL，用于控制对由对象键标识的对象的访问权限。有关更多信息，包括允许的值，请参阅 封装 ACL。

支持替换模板：否

`key`

数据写入的文件路径。

考虑一个例子，其中此参数是 `${topic()}/${timestamp()}`，并且规则会收到一条消息，其中主题为 `some/topic`。如果当前时间戳为 `1460685389`，那么这个操作会将数据写入一个 S3 存储桶中名为 `some/topic` 的文件夹中的文件 `1460685389`。

> **Note**
>
> 如果您使用静态密钥，则每次调用规则时都会 AWS IoT 覆盖单个文件。我们建议您使用消息时间戳或其它唯一的消息标识符，这样可以在 Amazon S3 中针对接收的每个消息保存一个新文件。

支持替换模板：是

`roleArn`

允许访问 Amazon S3 存储桶的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例在 AWS IoT 规则中定义了 S3 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "s3": {
                    "bucketName": "my-bucket",
                    "cannedacl": "public-read",
                    "key": "${topic()}/${timestamp()}",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3"
                }
            }
        ]
    }
}
```

另请参阅

- Amazon Simple Storage Service 用户指南中的什么是 Amazon S3？

Salesforce IoT

Salesforce IoT (`salesforce`) 操作将来自触发了规则的 MQTT 消息数据发送到 Salesforce IoT 输入流。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`url`

由 Salesforce IoT 输入流公开的 URL。在创建输入流时，可从 Salesforce IoT 平台获得该 URL。有关更多信息，请参阅 Salesforce IoT 文档。

支持替换模板：否

`token`

用于验证对指定的 Salesforce IoT 输入流的访问的令牌。在创建输入流时，可从 Salesforce IoT 平台获得该令牌。有关更多信息，请参阅 Salesforce IoT 文档。

支持替换模板：否

示例

以下 JSON 示例定义了 AWS IoT 规则中的 Salesforce IoT 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "salesforce": {
                    "token": "ABCDEFGHI123456789abcdefghi123456789",
                    "url": "https://ingestion-cluster-id.my-env.sfdcnow.com/streams/
stream-id/connection-id/my-event"
```

```
                }
            }
        ]
    }
}
```

SNS

SNS (sns) 操作将 MQTT 消息中的数据作为 Amazon Simple Notification Service (Amazon SNS) 推送通知发送。

您可以按照教程执行操作，该教程向您说明如何使用 SNS 操作创建和测试规则。有关更多信息，请参阅 教程：发送 Amazon SNS 通知。

> **Note**
>
> SNS 操作不支持Amazon SNS FIFO（先进先出）主题。由于规则引擎是一个完全分布式服务，因此无法保证触发 SNS 操作时的消息顺序。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`sns:Publish`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。
- 如果您使用 AWS KMS 客户托管 AWS KMS key 来加密 Amazon SNS 中的静态数据，则该服务必须有权代表呼叫 AWS KMS key 者使用这些数据。有关更多信息，请参阅 Amazon Simple Notification Service 开发人员指南中的密钥管理。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`targetArn`

推送通知将发送到的 SNS 主题或单个设备。

支持替换模板： AWS CLI 仅支持 API

`messageFormat`

（可选）消息格式。Amazon SNS 使用此设置来确定是否应解析负载，以及是否应提取负载的特定于平台的相关部分。有效值：`JSON`、`RAW`。默认值为 `RAW`。

支持替换模板：否

`roleArn`

允许访问 SNS 的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例在 AWS IoT 规则中定义了 SNS 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-east-2:123456789012:my_sns_topic",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"
                }
            }
        ]
    }
}
```

以下 JSON 示例在 AWS IoT 规则中定义了带有替换模板的 SNS 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
```

```
        "actions": [
            {
                "sns": {
                    "targetArn": "arn:aws:sns:us-east-1:123456789012:${topic()}",
                    "messageFormat": "JSON",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"
                }
            }
        ]
    }
}
```

另请参阅

- Amazon Simple Notification Service 开发人员指南中的什么是 Amazon Simple Notification Service？
- 教程：发送 Amazon SNS 通知

SQS

SQS (sqs) 操作将 MQTT 消息中的数据发送到 Amazon Simple Queue Service (Amazon SQS) 队列。

> **Note**
>
> SQS 操作不支持 Amazon SQS FIFO（先进先出）队列。由于规则引擎是一个完全分布式服务，因此无法保证触发 SQS 操作时的消息顺序。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`sqs:SendMessage`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。
- 如果您使用 AWS KMS 客户托管 AWS KMS key 对 Amazon SQS 中的静态数据进行加密，则该服务必须有权代表来电者使用这些数据。 AWS KMS key 有关更多信息，请参阅 Amazon Simple Queue Service 开发人员指南中的密钥管理。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`queueUrl`

数据写入的 Amazon SQS 队列的 URL。此 URL 中的区域不必与您的AWS IoT 规则 AWS 区域相同。

> **Note**
>
> AWS 区域 使用 SQS 规则操作进行交叉数据传输可能会产生额外费用。有关更多信息，请参阅 Amazon SQS 定价。

支持替换模板： AWS CLI 仅支持 API

`useBase64`

将该参数设置为 `true` 以配置规则操作，以便在将数据写入 Amazon SQS 队列之前对消息数据进行 base64 编码。默认值为 `false`。

支持替换模板：否

`roleArn`

允许访问 Amazon SQS 队列的 IAM 角色。有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例在 AWS IoT 规则中定义了 SQS 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
```

```
                "sqs": {
                    "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/
my_sqs_queue",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqs"
                }
            }
        ]
    }
}
```

以下 JSON 示例定义了在 AWS IoT 规则中使用替换模板的 SQS 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "sqs": {
                    "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/
${topic()}",
                    "useBase64": true,
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqs"
                }
            }
        ]
    }
}
```

另请参阅

- Amazon Simple Queue Service 开发人员指南中的什么是 Amazon Simple Queue Service？

Step Functions

Step Functions (`stepFunctions`) 操作启动 AWS Step Functions 状态机。

要求

此规则操作具有以下要求：

- AWS IoT 可以代入执行`states:StartExecution`操作的 IAM 角色。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择或创建 AWS IoT 允许执行此规则操作的角色。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`stateMachineName`

开始执行 Step Functions 状态机的名称。

支持替换模板： AWS CLI 仅支持 API

`executionNamePrefix`

（可选）将为状态机执行操作指定名称，由此前缀后加 UUID 组成。如果未提供名称，Step Functions 会自动为每次状态机执行创建一个唯一的名称。

支持替换模板：是

`roleArn`

授予启动状态机的 AWS IoT 权限的角色的 ARN。有关更多信息，请参阅 要求。

支持替换模板：否

示例

以下 JSON 示例在 AWS IoT 规则中定义了 Step Functions 操作。

```
{
    "topicRulePayload": {
        "sql": "SELECT * FROM 'some/topic'",
        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
        "actions": [
            {
                "stepFunctions": {
                    "stateMachineName": "myStateMachine",
                    "executionNamePrefix": "myExecution",
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_step_functions"
```

```
                }
            }
        ]
    }
}
```

另请参阅

- 什么是 AWS Step Functions？ 在《AWS Step Functions 开发者指南》中

Timestream

Timestream 规则操作将 MQTT 消息中的属性（度量）写入 Amazon Timestream 表。有关 Amazon Timestream 的更多信息，请参阅什么是 Amazon Timestream？。

> **Note**
>
> Amazon Timestream 并非在所有版本中都 AWS 区域可用。如果 Amazon Timestream 在您所在区域中不可用，则它将不会显示在规则操作列表中。

此规则存储在 Timestream 数据库中的属性是由规则的查询语句生成的属性。对查询语句结果中每个属性的值进行解析以推断其数据类型（如 the section called “DynamoDBv2” 操作）。每个属性的值都写入到 Timestream 表中自己的记录中。要指定或更改属性的数据类型，请在查询语句中使用 `cast()` 函数。有关每条 Timestream 记录内容的更多信息，请参阅 the section called “Timestream 记录内容”。

> **Note**
>
> 对于 SQL V2 (2016-03-23)，整数数字值，例如 `10.0`，将转换为整数表示形式 (`10`)。显式将它们转换为 `Decimal` 值，例如通过使用 cast() 函数，不会阻止此行为——结果仍然是 `Integer` 值。这可能会导致类型不匹配错误，从而阻止在 Timestream 数据库中记录数据。要将整数数值处理为 `Decimal` 值，请使用 SQL V1 (2015-10-08) 作为规则查询语句。

> **Note**
>
> Timestream 规则操作可以写入 Amazon Timestream 表的值的最大数目为 100。有关更多信息，请参阅 Amazon Timestream 配额参考。

要求

此规则操作具有以下要求：

- 一个 IAM 角色， AWS IoT 可以代入执行`timestream:DescribeEndpoints`和`timestream:WriteRecords`操作。有关更多信息，请参阅 授予 AWS IoT 规则所需的访问权限。

  在 AWS IoT 控制台中，您可以选择、更新或创建 AWS IoT 允许执行此规则操作的角色。
- 如果您使用客户- AWS KMS 来加密 Timestream 中的静态数据，则该 AWS KMS key 服务必须有权代表呼叫者使用。有关更多信息，请参阅AWS 服务如何使用 AWS KMS。

参数

使用此操作创建 AWS IoT 规则时，必须指定以下信息：

`databaseName`

Amazon Timestream 数据库的名称，该数据库中具有用于接收此操作创建的记录的表。另请参阅 **`tableName`**。

支持替换模板： AWS CLI 仅支持 API

`dimensions`

写入每个度量记录的时间序列的元数据属性。例如，EC2 实例的名称和可用区或风力涡轮机制造商的名称都是维度。

`name`

元数据维度名称。这是数据库表记录中列的名称。

维度不能命名为：`measure_name`、`measure_value` 或者 `time`。这些是预留的名称。维度名称不能以 `ts_` 或者 `measure_value` 开头，并且它们不能包含冒号 (`:`) 字符。

支持替换模板：否

`value`

在数据库记录的此列中写入的值。

支持替换模板：是

roleArn

角色的 Amazon Resource Name (ARN)，该角色授予 AWS IoT 写入 Timestream 数据库表的权限。有关更多信息，请参阅 要求。

支持替换模板：否

tableName

要将度量记录写入的数据库表的名称。另请参阅 **databaseName**。

支持替换模板： AWS CLI 仅支持 API

timestamp

用于条目的时间戳的值。如果为空，则使用处理条目的时间。

`unit`

`value` 中描述的表达式生成的时间戳值的精度。

有效值：`SECONDS` | `MILLISECONDS` |`MICROSECONDS` | `NANOSECONDS`。默认值为 `MILLISECONDS`。

`value`

返回长纪元时间值的表达式。

您可以使用 the section called “time_to_epoch(String, String)” 函数从消息负载中传递的日期或时间值创建有效的时间戳。

Timestream 记录内容

通过此操作写入 Amazon Timestream 表的数据包括时间戳、Timestream 规则操作中的元数据以及规则查询语句的结果。

对于查询语句结果中的每个属性（度量），此规则操作将记录写入具有这些列的指定 Timestream 表。

列名称	属性类型	值	注释
####	维度	在 Timestream 规则操作条目中指定的值。	每个维度都会在规则操作条目中指定的 Timestream 数据库中

列名称	属性类型	值	注释
			创建一个具有维度名称的列。
measure_name	MEASURE_NAME	属性的名称	查询语句结果中的属性名称，其值已在 measure_value:: *data-type* 列。
measure_value::*data-type*	MEASURE_VALUE	查询语句结果中属性的值。属性的名称位于 measure_name 列。	该值被解释* 并强制转换为最合适的匹配：bigint、boolean、double 或 varchar。Amazon Timestream 为每种数据类型创建一个单独的列。消息中的值可以转换为另一个数据类型，方法是在规则的查询语句中使用 cast() 函数。
time	TIMESTAMP	数据库中记录的日期和时间。	此值由规则引擎或 timestamp 属性分配（如果已定义）。

* 从消息负载读取的属性值解释如下。请参阅 the section called “示例”，了解各个案例的说明。

- 一个无引号的 true 或者 false 值将被解释为 boolean 类型。
- 十进制数字被解释为 double 类型。
- 没有小数点的数值被解释为 bigint 类型。
- 带引号的字符串被解释为 varchar 类型。
- 对象和数组值将转换为 JSON 字符串并存储为 varchar 类型。

示例

以下 JSON 示例定义了在规则中使用替换模板的 Timestream AWS IoT 规则操作。

```
{
  "topicRulePayload": {
    "sql": "SELECT * FROM 'iot/topic'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
      {
        "timestream": {
          "roleArn": "arn:aws:iam::123456789012:role/aws_iot_timestream",
          "tableName": "devices_metrics",
          "dimensions": [
            {
              "name": "device_id",
              "value": "${clientId()}"
            },
            {
              "name": "device_firmware_sku",
              "value": "My Static Metadata"
            }
          ],
          "databaseName": "record_devices"
        }
      }
    ]
  }
}
```

将上一示例中定义的 Timestream 主题规则操作与以下消息负载结合使用会使得 Amazon Timestream 记录写入下表中。

```
{
  "boolean_value": true,
  "integer_value": 123456789012,
  "double_value": 123.456789012,
  "string_value": "String value",
  "boolean_value_as_string": "true",
  "integer_value_as_string": "123456789012",
  "double_value_as_string": "123.456789012",
  "array_of_integers": [23,36,56,72],
```

```
  "array of strings": ["red", "green","blue"],
  "complex_value": {
    "simple_element": 42,
    "array_of_integers": [23,36,56,72],
    "array of strings": ["red", "green","blue"]
  }
}
```

下表显示了使用指定主题规则操作处理先前创建消息负载的数据库列和记录。`device_firmware_sku` 和 `device_id` 列是主题规则操作中定义的维度。Timestream 主题规则操作将创建 `time` 列以及 `measure_name` 和 `measure_value::*` 列，它将使用主题规则操作的查询语句结果中的值来填充这些值。

device_firmware_sku	device_id	measure_name	measure_value::bigint	measure_value::varchar	measure_value::double	measure_value::boolean	time
我的静态元数据	iotconsole-159EXAMPLE738-0	complex_value	-	{"simple_element":42,"array_of_integers":[23,36,56,72],"array of strings":["red","green","blue"]}	-	-	2020-08-26 22:42:16.423000000
我的静态元数据	iotconsole-159EXAMPLE738-0	integer_value_as_string	-	123456789012	-	-	2020-08-26 22:42:16.423000000
我的静态元数据	iotconsole-159EXAMPLE738-0	boolean_value	-	-	-	TRUE	2020-08-26 22:42:16.423000000

device_firmware_sku	device_id	measure_name	measure_value::bigint	measure_value::varchar	measure_value::double	measure_value::boolean	time
我的静态元数据	iotconsole-159EXAMPLE738-0	integer_value	123456789012	-	-	-	2020-08-26 22:42:16.423000000
我的静态元数据	iotconsole-159EXAMPLE738-0	string_value	-	字符串值	-	-	2020-08-26 22:42:16.423000000
我的静态元数据	iotconsole-159EXAMPLE738-0	array_of_integers	-	[23,36,56,72]	-	-	2020-08-26 22:42:16.423000000
我的静态元数据	iotconsole-159EXAMPLE738-0	字符串数组	-	["red","green","blue"]	-	-	2020-08-26 22:42:16.423000000
我的静态元数据	iotconsole-159EXAMPLE738-0	boolean_value_as_string	-	TRUE	-	-	2020-08-26 22:42:16.423000000
我的静态元数据	iotconsole-159EXAMPLE738-0	double_value	-	-	123.456789012	-	2020-08-26 22:42:16.423000000
我的静态元数据	iotconsole-159EXAMPLE738-0	double_value_as_string	-	123.45679	-	-	2020-08-26 22:42:16.423000000

排查规则问题

如果您对规则有疑问，我们建议您激活 CloudWatch Logs。通过分析您的日志，您可以确定问题是否与授权相关，或者是否为诸如 WHERE 子句状态不匹配的问题。有关更多信息，请参阅[设置 CloudWatch 日志]。

使用规则访问跨账户资源 AWS IoT

您可以配置跨账户访问 AWS IoT 规则，以便在一个账户的 MQTT 主题上提取的数据可以路由到另一个账户的 AWS 服务，例如 Amazon SQS 和 Lambda。以下内容说明如何设置跨账户数据提取 AWS IoT 规则，从一个账户中的 MQTT 主题到另一个账户中的目的地。

跨账户规则可以在目标资源上使用基于资源的权限。因此，只有支持基于资源的权限的目标才能启用带 AWS IoT 规则的跨账户访问权限。支持的目的地包括 Amazon SQS、Amazon SNS、Amazon S3 和 AWS Lambda。

> **Note**
>
> 对于支持的目标（Amazon SQS 除外），您必须将规则定义 AWS 区域 为与其他服务的资源相同，这样规则操作才能与该资源交互。有关 AWS IoT 规则操作的更多信息，请参阅AWS IoT 规则操作。有关规则的 SQS 操作的更多信息，请参阅???。

先决条件

- 熟悉 AWS IoT 规则
- 了解 IAM 用户、角色和基于资源的权限
- 安装 AWS CLI

Amazon SQS 的跨账户设置

场景：账户 A 将 MQTT 消息中的数据发送到账户 B 的 Amazon SQS 队列。

AWS 账户	账户名称	描述
1111-1111-1111	账户 A	规则操作：`sqs:SendMessage`
2222-2222-2222	账户 B	Amazon SQS 队列 • ARN：*arn:aws:sqs:region:2222-2222-2222:ExampleQueue*

AWS 账户	账户名称	描述
		• URL：*https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue*

> **Note**
>
> 您的目标 Amazon SQS 队列不必与您的AWS IoT 规则 AWS 区域 相同。有关规则的 SQS 操作的更多信息，请参阅???。

执行账户 A 任务

> **备注**
>
> 要运行以下命令，您的 IAM 用户应具有 `iot:CreateTopicRule` 使用规则的 Amazon Resource Name (ARN) 作为资源的权限，并且具有 `iam:PassRole` 操作使用资源作为角色的 ARN 的权限。

1. 使用账户 A 的 IAM 用户配置 AWS CLI。
2. 创建信任 AWS IoT 规则引擎的 IAM 角色，并附加允许访问账户 B 的 Amazon SQS 队列的策略。请参阅授予 AWS IoT 所需访问权限中的示例命令和策略文档。
3. 要创建附加到主题的规则，请运行该create-topic-rule 命令。

```
aws iot create-topic-rule --rule-name myRule --topic-rule-payload file://./my-rule.json
```

下面是一个负载文件示例，其中包含的规则会将发送至 `iot/test` 主题的所有消息插入指定 Amazon SQS 队列。SQL 语句筛选信息，角色 ARN 授予 AWS IoT 写入 Amazon SQS 队列的权限。

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
```

```
 "actions": [
  {
   "sqs": {
    "queueUrl": "https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue",
    "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role",
    "useBase64": false
   }
  }
 ]
}
```

有关如何在规则中定义 Amazon SQS 操作的更多信息，请参阅 AWS IoT AWS IoT 规则操作-Amazon SQS。

执行账户 B 任务

1. 使用账户 B 的 IAM 用户配置 AWS CLI。
2. 要向账户 A 授予 Amazon SQS 队列资源的权限，请运行 add-permission 命令。

```
aws sqs add-permission --queue-url https://sqs.region.amazonaws.com/2222-2222-2222/ExampleQueue --label SendMessagesToMyQueue --aws-account-ids 1111-1111-1111 --actions SendMessage
```

Amazon SNS 的跨账户设置

场景：账户 A 将 MQTT 消息中的数据发送到账户 B 的 Amazon SNS 主题。

AWS 账户	账户名称	描述
1111-1111-1111	账户 A	规则操作：`sns:Publish`
2222-2222-2222	账户 B	Amazon SNS 主题 ARN：*arn:aws:sns:region:2222-2222-2222:ExampleTopic*

执行账户 A 任务

> **注意**
>
> 要运行以下命令，您的 IAM 用户应具有 `iot:CreateTopicRule` 的权限，拥有能够作为资源的规则 ARN，并具有 `iam:PassRole` 操作的权限，拥有作为角色 ARN 的资源。

1. 使用账户 A 的 IAM 用户配置 AWS CLI。
2. 创建信任 AWS IoT 规则引擎的 IAM 角色，并附加允许访问账户 B 的 Amazon SNS 主题的策略。有关命令和策略文档的示例，请参阅授 AWS IoT 予所需访问权限。
3. 要创建附加到主题的规则，请运行该create-topic-rule 命令。

```
aws iot create-topic-rule --rule-name myRule --topic-rule-payload file://./my-rule.json
```

下面是一个负载文件示例，其中包含的规则会将发送至 `iot/test` 主题的所有消息插入指定 Amazon SNS 主题。SQL 语句筛选消息，而角色 ARN 则授予向 Amazon SNS 主题发送消息的 AWS IoT 权限。

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "sns": {
    "targetArn": "arn:aws:sns:region:2222-2222-2222:ExampleTopic",
    "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role"
   }
  }
 ]
}
```

有关如何在规则中定义 Amazon SNS 操作的更多信息，请参阅 AWS IoT AWS IoT 规则操作-Amazon SNS。

执行账户 B 任务

1. 使用账户 B 的 IAM 用户配置 AWS CLI。
2. 要向账户 A 授予 Amazon SNS 主题资源的权限，请运行 add-permission 命令。

```
aws sns add-permission --topic-arn arn:aws:sns:region:2222-2222-2222:ExampleTopic
 --label Publish-Permission --aws-account-id 1111-1111-1111 --action-name Publish
```

Amazon S3 的跨账户设置

场景：账户 A 将 MQTT 消息中的数据发送到账户 B 的 Amazon S3 存储桶。

AWS 账户	账户名称	描述
1111-1111-1111	账户 A	规则操作：`s3:PutObject`
2222-2222-2222	账户 B	Amazon S3 存储桶 ARN：*arn:aws:s3:::ExampleBucket*

执行账户 A 任务

> **备注**
>
> 要运行以下命令，您的 IAM 用户应具有 `iot:CreateTopicRule` 的权限，能够将规则 ARN 作为资源，并具有 `iam:PassRole` 操作的权限，将资源作为角色 ARN。

1. 使用账户 A 的 IAM 用户配置 AWS CLI。
2. 创建信任 AWS IoT 规则引擎的 IAM 角色并附加允许访问账户 B 的 Amazon S3 存储桶的策略。有关命令和策略文档的示例，请参阅授 AWS IoT 予所需访问权限。
3. 要创建附加到目标 S3 存储桶的规则，请运行create-topic-rule 命令。

```
aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://./my-
rule.json
```

下面是一个负载文件示例，其中包含的规则会将发送至 `iot/test` 主题的所有消息插入指定 Amazon S3 存储桶。SQL 语句筛选消息，角色 ARN 授予 AWS IoT 将消息写入 Amazon S3 存储桶的权限。

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "s3": {
    "bucketName": "ExampleBucket",
    "key": "${topic()}/${timestamp()}",
    "roleArn": "arn:aws:iam::1111-1111-1111:role/my-iot-role"
   }
  }
 ]
}
```

有关如何在规则中定义 Amazon S3 操作的更多信息，请参阅 AWS IoT AWS IoT 规则操作-Amazon S3。

执行账户 B 任务

1. 使用账户 B 的 IAM 用户配置 AWS CLI。
2. 创建信任账户 A 的委托人的存储桶策略。

   下面的示例负载文件定义信任另一账户委托人的存储桶策略。

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "AddCannedAcl",
   "Effect": "Allow",
   "Principal": {
    "AWS": [
     "arn:aws:iam::1111-1111-1111:root"
    ]
   },
```

```
    "Action": "s3:PutObject",
    "Resource": "arn:aws:s3:::ExampleBucket/*"
   }
  ]
}
```

有关更多信息，请参阅存储桶策略示例。

3. 要将存储桶策略附加到指定的存储桶，请运行put-bucket-policy 命令。

```
aws s3api put-bucket-policy --bucket ExampleBucket --policy file://./my-bucket-policy.json
```

4. 要跨账户访问工作，请确保您的阻止所有公有访问设置正确。有关更多信息，请参阅 Amazon S3 安全性最佳实践。

的跨账户设置 AWS Lambda

场景：账户 A 调用账户 B 的 AWS Lambda 函数，传入 MQTT 消息。

AWS 账户	账户名称	描述
1111-1111-1111	账户 A	规则操作：`lambda:InvokeFunction`
2222-2222-2222	账户 B	Lambda 函数 ARN： *arn:aws:lambda:region:2222-2222-2222:function:example-function*

执行账户 A 任务

> **注意**
>
> 要运行以下命令，您的 IAM 用户应具有 `iot:CreateTopicRule` 的权限，能够将规则 ARN 作为资源，以及对于 `iam:PassRole` 的权限，能够将资源作为角色 ARN。

1. 使用账户 A 的 IAM 用户配置 AWS CLI。

2. 运行create-topic-rule 命令创建规则，定义对账户 B 的 Lambda 函数的跨账户访问权限。

```
aws iot create-topic-rule --rule-name my-rule --topic-rule-payload file://./my-rule.json
```

下面是一个负载文件示例，其中包含的规则会将发送至 iot/test 主题的所有消息插入指定 Lambda 函数。SQL 语句筛选消息，ARN 角色授予将数据传递给 Lambda 函数的 AWS IoT 权限。

```
{
 "sql": "SELECT * FROM 'iot/test'",
 "ruleDisabled": false,
 "awsIotSqlVersion": "2016-03-23",
 "actions": [
  {
   "lambda": {
    "functionArn": "arn:aws:lambda:region:2222-2222-2222:function:example-function"
   }
  }
 ]
}
```

有关如何在规则中定义 AWS Lambda 操作的更多信息，请阅读 AWS IoT AWS IoT 规则操作-Lambda。

执行账户 B 任务

1. 使用账户 B 的 IAM 用户配置 AWS CLI。
2. 运行 Lambda 的添加权限命令以授予 AWS IoT 规则激活 Lambda 函数的权限。要运行以下命令，您的 IAM 用户应具有 lambda:AddPermission 操作的权限。

```
aws lambda add-permission --function-name example-function --region us-east-1 --
principal iot.amazonaws.com --source-arn arn:aws:iot:region:1111-1111-1111:rule/
example-rule --source-account 1111-1111-1111 --statement-id "unique_id" --action
 "lambda:InvokeFunction"
```

选项：

--principal

此字段授予 AWS IoT（由`iot.amazonaws.com`）调用 Lambda 函数的权限。

--source-arn

此字段确认仅 AWS IoT 中的 `arn:aws:iot:region:1111-1111-1111:rule/example-rule` 触发此 Lambda 函数，并且相同或不同账户中的任何其它规则都不能激活此 Lambda 函数。

--source-account

此字段确认仅代表 AWS IoT 账户激活此 Lambda 函数。`1111-1111-1111`

> **注意**
>
> 如果您看到一条错误消息在您的 AWS Lambda 函数的控制台中 Configuration（配置）下显示：“The rule could not be found”（找不到规则），请忽略错误消息并继续测试连接。

错误处理（错误操作）

当 AWS IoT 收到来自设备的消息时，规则引擎会检查该消息是否符合规则。如果匹配，将评估规则的查询语句，并激活规则的操作，同时传递查询语句的结果。

如果在激活操作时出现问题，则在为规则指定了错误操作的情况下，规则引擎将激活该错误操作。这可能在以下情况下发生：

- 规则没有权限访问 Amazon S3 存储桶。
- 用户错误导致超过 DynamoDB 预置吞吐量。

> **Note**
>
> 本主题中介绍的错误处理适用于规则操作。要调试 SQL 问题（包括外部函数），可以设置 AWS IoT 日志记录。有关更多信息，请参阅 ???。

错误操作消息格式

对于每个规则和每条消息，会生成一条消息。例如，如果同一个规则中的两个规则操作失败，则错误操作将收到包含这两项错误的一条消息。

错误操作消息示例如下所示。

```
{
  "ruleName": "TestAction",
  "topic": "testme/action",
  "cloudwatchTraceId": "7e146a2c-95b5-6caf-98b9-50e3969734c7",
  "clientId": "iotconsole-1511213971966-0",
  "base64OriginalPayload":
 "ewogICJtZXNzYWdlIjogIkhlbGxvIHZyb20gQVdTIElvVCBjb25zb2xlIgp9",
  "failures": [
    {
      "failedAction": "S3Action",
      "failedResource": "us-east-1-s3-verify-user",
      "errorMessage": "Failed to put S3 object. The error received was The
 specified bucket does not exist (Service: Amazon S3; Status Code: 404; Error
 Code: NoSuchBucket; Request ID: 9DF5416B9B47B9AF; S3 Extended Request ID:
 yMah1cwPhqTH267QLPhTKeVPKJB8BO5ndBHzOmWtxLTM6uAvwYYuqieAKyb6qRPTxP1tHXCoR4Y=).
 Message arrived on: error/action, Action: s3, Bucket: us-
east-1-s3-verify-user, Key: \"aaa\". Value of x-amz-id-2:
 yMah1cwPhqTH267QLPhTKeVPKJB8BO5ndBHzOmWtxLTM6uAvwYYuqieAKyb6qRPTxP1tHXCoR4Y="
    }
  ]
}
```

ruleName

触发错误操作的规则的名称。

topic

收到原始消息的主题。

云观察 TraceId

引用错误的唯一身份登录 CloudWatch。

clientId

消息发布程序的客户端 ID。

base64 OriginalPayload

Base64 编码的原始消息负载。

failures

failedAction

无法完成的操作的名称（例如，“S3Action”）。

failedResource

资源的名称（例如 S3 存储桶的名称）。

errorMessage

错误的描述和说明。

错误操作示例

下面是一个具有添加的错误操作的规则示例。以下规则具有将消息数据写入 DynamoDB 表的操作，以及将数据写入 Amazon S3 存储桶的错误操作：

```
{
    "sql" : "SELECT * FROM ..."
    "actions" : [{
        "dynamoDB" : {
            "table" : "PoorlyConfiguredTable",
            "hashKeyField" : "AConstantString",
            "hashKeyValue" : "AHashKey"}}
    ],
    "errorAction" : {
        "s3" : {
            "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",
            "bucketName" : "message-processing-errors",
            "key" : "${replace(topic(), '/', '-') + '-' + timestamp() + '-' +
 newuuid()}"
        }
    }
}
```

可以在错误操作的 SQL 语句中使用任何函数或替换模板，包括外部函数：aws_lambda()get_dynamodb()、get_thing_shadow()、get_secret()、machinelearning_和decode()。如果错误操作需要调用外部函数，则调用错误操作可能会导致外部函数的额外账单。

以下外部函数的计费方式等同于规则操作的计费：aws_lambdaget_dynamodb()、和。get_thing_shadow()仅当你将 P rotobuf 消息解码为 JSON 时，你才需要为该decode()函数付费。如需了解更多详情，请参阅定AWS IoT Core 价页面。

有关规则以及如何指定错误操作的更多信息，请参阅创建 AWS IoT 规则。

有关使用 CloudWatch 监控规则成功或失败的更多信息，请参阅AWS IoT 指标和维度。

借助基本摄取功能，降低消息收发成本

您可以使用 Basic Ingest 将设备数据安全地发送给 AWS 服务 支持者AWS IoT 规则动作，而不会产生消息收发费用。基本摄取功能通过从摄取路径中删除发布/订阅消息代理来优化数据流。

基本摄取功能可以从设备或应用程序发送消息。这些消息的主题名称以 $aws/rules/*rule_name* 开头（表示其前三个级别），其中 *rule_name* 是您要调用的 AWS IoT 规则的名称。

您可以借助基本摄取功能来使用现有规则，方法为将基本摄取前缀（$aws/rules/*rule_name*）添加到您用于调用规则的消息主题。例如，如果您有名为 BuildingManager 的规则且它由主题类似于 Buildings/Building5/Floor2/Room201/Lights（"sql": "SELECT * FROM 'Buildings/#'"）的消息调用，那么通过发送主题为 $aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights 的消息，您就可以使用基本摄取功能调用相同规则。

注意：

- 您的设备和规则无法订阅基本摄取预留主题。有关更多信息，请参阅保留的主题。
- 如果您需要发布/订阅代理来向多个订阅者分发消息（例如，将消息传送到其他设备和规则引擎），则应继续使用 AWS IoT 消息代理来处理消息分发。但是，请确保在基本摄取主题以外的主题上发布您的消息。

使用基本摄取功能

使用基本摄取功能之前，请验证您的设备或应用程序正在使用对 $aws/rules/* 具有发布权限的策略。或者，您可以在策略$aws/rules/*rule_name*/*中为各个规则指定权限。否则，您的设备和应用程序可以继续使用与 AWS IoT Core 的现有连接。

当消息到达规则引擎时，从基本摄取功能调用的规则与通过消息代理订阅调用的规则之间的实现或错误处理没有任何区别。

您可以创建与基本摄取搭配使用的规则。记住以下内容：

- 基本摄取主题的初始前缀 (`$aws/rules/rule_name`) 不可用于 topic(Decimal) 函数。
- 如果您定义的规则仅通过基本摄取功能来调用，则 FROM 子句在 `rule` 定义的 `sql` 字段中是可选的。如果规则还由必须通过消息代理发送的其他消息来调用（例如，因为这些其他消息必须分配给多个订阅者），则其仍是必要项目。有关更多信息，请参阅AWS IoT SQL 参考。
- 基本摄取主题的前三个级别 (`$aws/rules/rule_name`) 不计入 8 个分段长度限制或针对主题的 256 个总字符限制。否则，应用 AWS IoT 限制中记录的相同限制。
- 如果收到一条带有基本收录主题的消息，其中指定了非活动规则或不存在的规则，则会在 Amazon 日志中创建错误 CloudWatch 日志以帮助您进行调试。有关更多信息，请参阅规则引擎日志条目。将指示 `RuleNotFound` 指标，您可以在该指标上创建警报。有关更多信息，请参阅规则指标中的“规则指标”。
- 您仍可以使用 QoS 1 在基本摄取主题上进行发布。在消息成功传送到规则引擎后，您会收到 PUBACK。收到 PUBACK 并不意味着您的规则操作成功完成。您可以配置错误操作以在操作运行期间处理错误。有关更多信息，请参阅 错误处理（错误操作）。

AWS IoT SQL 参考

在中 AWS IoT，规则是使用类似 SQL 的语法定义的。SQL 语句由三类子句组成：

SELECT

（必需）从传入消息有效负载提取信息并执行信息转换。要使用的消息由主题筛选条件在 FROM 子句中指定。

SELECT 子句支持 数据类型、运算符、函数、文本、Case 语句、JSON 扩展、替换模板、嵌套对象查询 和 二进制负载。

FROM

MQTT 消息主题筛选条件，用于标识要从中提取数据的消息。对于发送到符合此处指定的主题筛选条件的 MQTT 主题的每条消息，将激活该规则。对于通过消息代理传递的消息所激活的规则是必需的。对于仅使用基本摄取特征激活的规则是可选的。

WHERE

（可选）添加用于确定是否执行规则指定的操作的条件逻辑。

WHERE 子句支持 数据类型、运算符、函数、文本、Case 语句、JSON 扩展、替换模板 和 嵌套对象查询。

SQL 语句的示例如下所示：

```
SELECT color AS rgb FROM 'topic/subtopic' WHERE temperature > 50
```

MQTT 消息 (也称为传入负载) 的示例如下所示：

```
{
    "color":"red",
    "temperature":100
}
```

如果此消息在 'topic/subtopic' 主题上发布，则触发规则并评估 SQL 语句。如果 color 属性大于 50，SQL 语句将提取 "temperature" 属性的值。WHERE 子句指定条件 temperature > 50。AS 关键字将 "color" 属性重命名为 "rgb"。结果（也称为传出负载）如下所示：

```
{
    "rgb":"red"
}
```

此数据随后将转发至规则的操作，在其中发送数据供后续处理。有关规则操作的更多信息，请参阅 AWS IoT 规则动作。

> **Note**
>
> AWS IoT SQL 语法目前不支持注释。
> 包含空格的属性名称不能用作 SQL 语句中的字段名称。虽然传入负载可以在属性名称中包含空格，但这些名称不能在 SQL 语句中使用。但是，如果您使用通配符 (*) 字段名称规范，它们将传递到传出负载。

SELECT 子句

SE AWS IoT LECT 子句与 ANSI SQL SELECT 子句基本相同，但有一些细微的区别。

SELECT 子句支持 数据类型、运算符、函数、文本、Case 语句、JSON 扩展、替换模板、嵌套对象查询 和 二进制负载。

您可以使用 SELECT 子句从传入 MQTT 消息中提取信息。也可以使用 SELECT * 检索整个传入消息负载。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT * FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50}
```

如果负载是 JSON 对象，您可以参考对象中的键。您的传出负载将包含键值对。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL statement: SELECT color FROM 'topic/subtopic'
Outgoing payload: {"color":"red"}
```

您可以使用 AS 关键字重命名键。例如：

```
Incoming payload published on topic 'topic/subtopic':{"color":"red", "temperature":50}
SQL:SELECT color AS my_color FROM 'topic/subtopic'
Outgoing payload: {"my_color":"red"}
```

您可以通过用逗号分隔来选择多个项目。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT color as my_color, temperature as fahrenheit FROM 'topic/subtopic'
Outgoing payload: {"my_color":"red","fahrenheit":50}
```

您可以通过在向传入负载添加项目时包括“*”来选择多个项目。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT *, 15 as speed FROM 'topic/subtopic'
Outgoing payload: {"color":"red", "temperature":50, "speed":15}
```

您可以使用 "VALUE" 关键字来生成不属于 JSON 对象的传出负载。使用 SQL 版本 2015-10-08 时，您只能选择一个项目。使用 SQL 版本 2016-03-23 或更高版本时，您还可以选择作为顶级对象输出的数组。

Example

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT VALUE color FROM 'topic/subtopic'
Outgoing payload: "red"
```

您可以使用 '.' 语法深入剖析传入负载中的嵌套 JSON 对象。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":
{"red":255,"green":0,"blue":0}, "temperature":50}
SQL: SELECT color.red as red_value FROM 'topic/subtopic'
Outgoing payload: {"red_value":255}
```

有关如何使用包含预留字符（如数字或连字符（减号）的 JSON 对象和属性名称的信息，请参阅 JSON 扩展

您可以使用函数 (参阅 函数) 来转换传入负载。您可以使用括号进行分组。例如：

```
Incoming payload published on topic 'topic/subtopic': {"color":"red", "temperature":50}
SQL: SELECT (temperature - 32) * 5 / 9 AS celsius, upper(color) as my_color FROM
 'topic/subtopic'
Outgoing payload: {"celsius":10,"my_color":"RED"}
```

FROM 子句

FROM 子句在主题或主题筛选条件中订阅您的规则。将主题或主题筛选条件用单引号 (') 引起来。对于发送到符合此处指定的主题筛选条件的 MQTT 主题的每条消息，将触发该规则。可以使用主题筛选条件订阅一组类似的主题。

示例：

传入负载已发布至主题 'topic/subtopic'：{temperature: 50}

传入负载已发布至主题 'topic/subtopic-2'：{temperature: 50}

SQL："SELECT temperature AS t FROM 'topic/subtopic'"。

该规则已订阅到 'topic/subtopic'，因此传入的负载将传递给该规则。传递给规则操作的传出负载为：{t: 50}。规则未订阅 'topic/subtopic-2'，因此在 'topic/subtopic-2' 上发布的消息不会触发规则。

通配符示例：

您可以使用“#”（多级）通配符来匹配一个或多个特定路径元素：

传入负载已发布至主题 'topic/subtopic'：{temperature: 50}。

传入负载已发布至主题 'topic/subtopic-2'：{temperature: 60}。

传入负载已发布至主题 'topic/subtopic-3/details'：{temperature: 70}。

传入负载已发布至主题 'topic-2/subtopic-x'：{temperature: 80}。

SQL："SELECT temperature AS t FROM 'topic/#'"。

规则订阅了以 'topic' 开头的所有主题，因此它会执行三次，将 {t: 50}（适用于 topic/subtopic）、{t: 60}（适用于 topic/subtopic-2）和 {t: 70}（适用于 topic/subtopic-3/details）的传出有效负载发送至其操作。它未订阅 'topic-2/subtopic-x'，因此不会针对 {temperature: 80} 消息触发规则。

+ 通配符示例：

您可以使用“+”（单级）通配符来匹配任一特定路径元素：

传入负载已发布至主题 'topic/subtopic'：{temperature: 50}。

传入负载已发布至主题 'topic/subtopic-2'：{temperature: 60}。

传入负载已发布至主题 'topic/subtopic-3/details'：{temperature: 70}。

传入负载已发布至主题 'topic-2/subtopic-x'：{temperature: 80}。

SQL："SELECT temperature AS t FROM 'topic/+'"。

规则已订阅包含两个路径元素的所有主题，其中第一个元素为 'topic'。此规则将对发送至 'topic/subtopic' 和 'topic/subtopic-2' 的消息执行，但不对 'topic/subtopic-3/details'（它的级别比主题筛选条件更多）或 'topic-2/subtopic-x'（它不以 topic 开头）执行。

WHERE 子句

WHERE 子句可确定是否执行规则指定的操作。如果 WHERE 子句的计算结果为 true，则执行规则操作。否则，不执行规则操作。

WHERE 子句支持 数据类型、运算符、函数、文本、Case 语句、JSON 扩展、替换模板 和 嵌套对象查询。

示例：

传入负载已发布至 topic/subtopic：{"color":"red", "temperature":40}。

SQL：SELECT color AS my_color FROM 'topic/subtopic' WHERE temperature > 50 AND color <> 'red'。

在这种情况下，规则将被触发，但不会执行规则所指定的操作。没有传出的负载。

您可以在 WHERE 子句中使用函数和运算符。但是，您无法引用在 SELECT 中通过 AS 关键字创建的任何别名。首先评估 WHERE 语句，以确定是否评估 SELECT。

带非 JSON 负载的示例：

传入的非 JSON 负载，发布位置：“topic/subtopic”：“80”

SQL：`SELECT decode(encode(*, 'base64'), 'base64') AS value FROM 'topic/subtopic' WHERE decode(encode(*, 'base64'), 'base64') > 50`

在这种情况下，规则将被触发，并且将执行规则所指定的操作。传出的负载将由 SELECT 子句转换为 JSON 负载 `{"value":80}`。

数据类型

AWS IoT 规则引擎支持所有 JSON 数据类型。

支持的数据类型

类型	含义
`Int`	离散的 `Int`。最大 34 位。
`Decimal`	精度为 34 位的 `Decimal`，最小非零数量级为 1E-999，最大数量级为 9.999...E999。 **Note** 部分函数返回双精度 `Decimal` 值，而不是 34 位数字精度。 对于 SQL V2 (2016-03-23)，整数数字值，例如 `10.0`，将被处理为 `Int` 值 (10)，而不是预期的 `Decimal` 值 (10.0)。若要可靠地将整数数值处理为 `Decimal` 值，请使用 SQL V1 (2015-10-08) 作为规则查询语句。
`Boolean`	`True` 或 `False`。

类型	含义
`String`	UTF-8 字符串。
`Array`	不必为相同类型的一系列值。
`Object`	包含一个键和一个值的 JSON 值。键必须是字符串。值可以是任意类型。
`Null`	`Null` 由 JSON 定义。它是表示缺少某个值的实际值。您必须通过使用 SQL 语句中 `Null` 关键字明确创建一个 `Null` 值。例如：`"SELECT NULL AS n FROM 'topic/subtopic'"`

`Undefined`	非值。在 JSON 中无法明确表示，只能忽略该值。例如，在对象 `{"foo": null}` 中，键“foo”返回 NULL，但键“bar”返回 `Undefined` 。在内部，SQL 语言将 `Undefined` 作为值处理，但它在 JSON 中无法表示，因此，在序列化为 JSON 时，结果为 `Undefined` 。
	`{"foo":null, "bar":undefined}`
	序列化为 JSON 如下：
	`{"foo":null}`
	同样，`Undefined` 在被自身序列化时会转化为空字符串。使用无效参数（例如，错误的类型、错误的参数号等）调用的函数将返回 `Undefined` 。

转换

下表列出当一个类型的值转换为另一个类型时 (为函数提供错误类型的值时) 返回的结果。例如，如果绝对值函数“abs”(期待的值类型为 `Int` 或 `Decimal`) 被赋予 `String` 时，它会尝试遵循以下规则将 `String` 转换为 `Decimal`。在这种情况下，“abs("-5.123")”将被视为“abs(-5.123)”。

> **Note**
>
> 不会尝试转换 Array、Object、Null 或 Undefined。

To decimal

参数类型	结果
Int	没有小数点的 Decimal。
Decimal	源值。
Boolean	Undefined 。（您可以明确使用 cast 函数使 true = 1.0，false = 0.0。）
String	SQL 引擎尝试将字符串解析为。Decimal AWS IoT 尝试解析与正则表达式匹配的字符串:^-?\d+(\.\d+)?((?i)E-?\d+)?$ 。可自动转换为 Decimal 的字符串示例包括 "0"、"-1.2"、"5E-12"。
数组	Undefined .
对象	Undefined .
Null	Null.
未定义	Undefined .

To int

参数类型	结果
Int	源值。
Decimal	源值都舍入到最接近的 Int。
Boolean	Undefined 。（您可以明确使用 cast 函数使 true = 1.0，false = 0.0。）

参数类型	结果
String	SQL 引擎尝试将字符串解析为。Decimal AWS IoT 尝试解析与正则表达式匹配的字符串:^-?\d+(\.\d+)?((?i)E-?\d+)?$ 。“0”、“-1.2”、“5E-12” 都是自动转换为 Decimal s 的字符串的示例。 AWS IoT 尝试将其转换为 aDecimal，然后将其中的小数位截断String为一个。Decimal Int
数组	Undefined .
对象	Undefined .
Null	Null.
未定义	Undefined .

To Boolean

参数类型	结果
Int	Undefined 。（您可以明确使用 cast 函数使 0 = False，any_nonzero_value = True。）
Decimal	Undefined 。（您可以明确使用 cast 函数使 0 = False，any_nonzero_value = True。）
Boolean	原始值。
String	"true"=True 和 "false"=False（不区分大小写）。其它字符串值为 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .

参数类型	结果
未定义	Undefined .

To string

参数类型	结果
Int	标准表示法中 Int 的字符串表示形式。
Decimal	科学表示法中 Decimal 值的字符串表示。
Boolean	"true" 或 "false"。均为小写。
String	原始值。
数组	Array 序列化为 JSON。结果字符串为逗号分隔的列表，括在方括号中。String 带引号。Decimal、Int、Boolean 和 Null 不带引号。
对象	序列化为 JSON 的对象。结果字符串为键值对的逗号分隔列表，以大括号开头和结束。String 带引号。Decimal、Int、Boolean 和 Null 不带引号。
Null	Undefined .
未定义	Undefined。

运算符

可以在 SELECT 和 WHERE 子句中使用以下运算符。

AND 运算符

返回 Boolean 结果。执行逻辑与运算。如果左右操作数为 true，则返回 true。否则，返回 false。需要提供 Boolean 操作数或不区分大小写的“true”或“false”字符串操作数。

语法： *expression* AND *expression*。

AND 运算符

左侧操作数	右侧操作数	输出
Boolean	Boolean	Boolean。如果两个操作数均为 true，则为 true。否则为 false。
String/Boolean	String/Boolean	如果所有字符串均为“true”或“false”（不区分大小写），则它们将被转换为 Boolean 并作为 *boolean* AND *boolean* 正常处理。
其它值	其它值	Undefined .

OR 运算符

返回 Boolean 结果。执行逻辑或运算。如果左右操作数至少有一个为 true，则返回 true。否则，返回 false。需要提供 Boolean 操作数或不区分大小写的“true”或“false”字符串操作数。

语法： *expression* OR *expression*。

OR 运算符

左侧操作数	右侧操作数	输出
Boolean	Boolean	Boolean。如果任意一个操作数为 true，则为 true。否则为 false。
String/Boolean	String/Boolean	如果所有字符串均为 "true" 或 "false" (不区分大小写)，则将其转换为布尔值并作为 *boolean* OR *boolean* 正常处理。
其它值	其它值	Undefined .

NOT 运算符

返回 Boolean 结果。执行逻辑非运算。如果操作数为 false，则返回 true。否则返回 true。需要 Boolean 操作数或不区分大小写的“true”或“false”字符串操作数。

语法：NOT *expression*。

NOT 运算符

操作数	输出
Boolean	Boolean。如果操作数为 false，则为 true。否则为 true。
String	如果字符串为“true”或“false”（不区分大小写），它将转换为对应的布尔值，并返回相反的值。
其它值	Undefined .

IN 运算符

返回 Boolean 结果。您可以在 WHERE 子句中使用 IN 运算符来检查某个值是否与数组中的任何值匹配。如果找到匹配项，则返回 true，否则返回 false。

语法： *expression* IN *expression*。

IN 运算符

左侧操作数	右侧操作数	输出
Int/Decimal/String/A	Array	如果在数组中找到String/Array/IntegerDecimal/Object元素，则为 true。否则为 false。

示例：

```
SQL: "select * from 'a/b' where 3 in arr"

JSON: {"arr":[1, 2, 3, "three", 5.7, null]}
```

在此示例中，条件子句的计算结果where 3 in arr将为 true，因为名为的数组中存在 3 arr。因此，在 SQL 语句中，select * from 'a/b'将执行。此示例还显示阵列可以是异构的。

存在运算符

返回 Boolean 结果。可以在条件子句中使用 EXISTS 运算符来测试子查询中是否存在元素。如果子查询返回一个或多个元素，则返回 true；如果子查询不返回任何元素，则返回 false。

语法： *expression*。

示例：

```
SQL: "select * from 'a/b' where exists (select * from arr as a where a = 3)"

JSON: {"arr":[1, 2, 3]}
```

在此示例中，条件子句的计算结果`where exists (select * from arr as a where a = 3)`将为 true，因为名为的数组中存在 3 `arr`。因此，在 SQL 语句中，`select * from 'a/b'`将执行。

示例：

```
SQL: select * from 'a/b' where exists (select * from e as e where foo = 2)

JSON: {"foo":4,"bar":5,"e":[{"foo":1},{"foo":2}]}
```

在此示例中，条件子句的计算结果`where exists (select * from e as e where foo = 2)`将为 true，因为 JSON 对象`e`中的数组包含该对象`{"foo":2}`。因此，在 SQL 语句中，`select * from 'a/b'`将执行。

> 运算符

返回 Boolean 结果。如果左侧操作数大于右侧操作数，则返回 true。两个操作数将转换为 `Decimal`，然后进行比较。

语法： *expression* > *expression*。

> 运算符

左侧操作数	右侧操作数	输出
`Int/Decimal`	`Int/Decimal`	`Boolean`。如果左侧操作数大于右侧操作数，则为 true。否则为 false。

左侧操作数	右侧操作数	输出
String/Int/Deci	String/Int/Deci	如果所有字符串可以转换为 Decimal，则为 Boolean。如果左侧操作数大于右侧操作数，则返回 true。否则为 false。
其它值	Undefined .	Undefined .

>= operator

返回 Boolean 结果。如果左侧操作数大于等于右侧操作数，则返回 true。两个操作数将转换为 Decimal，然后进行比较。

语法：*expression* >= *expression*。

>= operator

左侧操作数	右侧操作数	输出
Int/Decimal	Int/Decimal	Boolean。如果左侧操作数大于等于右侧操作数，则为 true。否则为 false。
String/Int/Deci	String/Int/Deci	如果所有字符串可以转换为 Decimal，则为 Boolean。如果左侧操作数大于等于右侧操作数，则返回 true。否则为 false。
其它值	Undefined .	Undefined .

< 运算符

返回 Boolean 结果。如果左侧操作数小于右侧操作数，则返回 true。两个操作数将转换为 Decimal，然后进行比较。

语法：*expression* < *expression*。

< 运算符

左侧操作数	右侧操作数	输出
Int/Decimal	Int/Decimal	Boolean。如果左侧操作数小于右侧操作数，则为 true。否则为 false。
String/Int/Deci	String/Int/Deci	如果所有字符串可以转换为 Decimal，则为 Boolean。如果左侧操作数小于右侧操作数，则返回 true。否则为 false。
其它值	Undefined	Undefined

<= 运算符

返回 Boolean 结果。如果左侧操作数小于等于右侧操作数，则返回 true。两个操作数将转换为 Decimal，然后进行比较。

语法：*expression* <= *expression*。

<= 运算符

左侧操作数	右侧操作数	输出
Int/Decimal	Int/Decimal	Boolean。如果左侧操作数小于等于右侧操作数，则为 true。否则为 false。
String/Int/Deci	String/Int/Deci	如果所有字符串可以转换为 Decimal，则为 Boolean。如果左侧操作数小于等于右侧操作数，则返回 true。否则为 false。
其它值	Undefined	Undefined

<> 运算符

返回 Boolean 结果。如果左右操作数不相等，则返回 true。否则返回 false。

语法： *expression* <> *expression*。

<> 运算符

左侧操作数	右侧操作数	输出
`Int`	`Int`	如果左侧操作数不等于右侧操作数，则为 true。否则为 false。
`Decimal`	`Decimal`	如果左侧操作数不等于右侧操作数，则为 true。否则为 false。在比较之前，`Int` 会被转换为 `Decimal`。
`String`	`String`	如果左侧操作数不等于右侧操作数，则为 true。否则为 false。
数组	数组	如果各个操作数中的项不相等且顺序不同，则为 true。否则为 false
对象	对象	如果各个操作数的键和值均不相等，则为 true。否则为 false。键/值的顺序不重要。
Null	Null	False。
任意值	`Undefined`	Undefined。
`Undefined`	任意值	Undefined。
不匹配的类型	不匹配的类型	True。

= 运算符

返回 `Boolean` 结果。如果左右操作数相等，则返回 true。否则返回 false。

语法： *`expression`* `=` *`expression`*。

= 运算符

左侧操作数	右侧操作数	输出
`Int`	`Int`	如果左侧操作数等于右侧操作数，则为 true。否则为 false。

左侧操作数	右侧操作数	输出
Decimal	Decimal	如果左侧操作数等于右侧操作数，则为 true。否则为 false。在比较之前，Int 会被转换为 Decimal。
String	String	如果左侧操作数等于右侧操作数，则为 true。否则为 false。
数组	数组	如果各个操作数中的项相等且顺序相同，则为 true。否则为 false。
对象	对象	如果各个操作数的键和值相等，则为 true。否则为 false。键/值的顺序不重要。
任意值	Undefined	Undefined .
Undefined	任意值	Undefined .
不匹配的类型	不匹配的类型	False。

+ 运算符

“+”是一个重载运算符。它可用于字符串联接或相加。

语法： *expression* + *expression*。

+ 运算符

左侧操作数	右侧操作数	输出
String	任意值	将右侧操作数转换为一个字符串，并联接到左侧操作数的末尾。
任意值	String	将左侧操作数转换为一个字符串，并将右侧操作数联接到转换后的左侧操作数的末尾。
Int	Int	Int 值。将操作数相加。
Int/Decimal	Int/Decimal	Decimal 值。将操作数相加。

左侧操作数	右侧操作数	输出
其它值	其它值	Undefined .

- 运算符

从左侧操作数中减去右侧操作数。

语法： *expression* - *expression*。

- 运算符

左侧操作数	右侧操作数	输出
Int	Int	Int 值。从左侧操作数中减去右侧操作数。
Int/Decimal	Int/Decimal	Decimal 值。从左侧操作数中减去右侧操作数。
String/Int/Deci	String/Int/Deci	如果所有字符串都正确地转换为小数，则返回 Decimal 值。从左侧操作数中减去右侧操作数。否则返回 Undefined 。
其它值	其它值	Undefined .
其它值	其它值	Undefined .

* 运算符

左侧操作数乘以右侧操作数。

语法： *expression* * *expression*。

* 运算符

左侧操作数	右侧操作数	输出
Int	Int	Int 值。左侧操作数乘以右侧操作数。
Int/Decimal	Int/Decimal	Decimal 值。左侧操作数乘以右侧操作数。

左侧操作数	右侧操作数	输出
String/Int/Deci	String/Int/Deci	如果所有字符串都正确地转换为小数，则返回 Decimal 值。左侧操作数乘以右侧操作数。否则返回 Undefined 。
其它值	其它值	Undefined .

/ 运算符

左侧操作数除以右侧操作数。

语法： *expression* / *expression*。

/ 运算符

左侧操作数	右侧操作数	输出
Int	Int	Int 值。左侧操作数除以右侧操作数。
Int/Decimal	Int/Decimal	Decimal 值。左侧操作数除以右侧操作数。
String/Int/Deci	String/Int/Deci	如果所有字符串都正确地转换为小数，则返回 Decimal 值。左侧操作数除以右侧操作数。否则返回 Undefined 。
其它值	其它值	Undefined .

% 运算符

返回左侧操作数除以右侧操作数得到的余数。

语法： *expression* % *expression*。

% 运算符

左侧操作数	右侧操作数	输出
Int	Int	Int 值。返回左侧操作数除以右侧操作数得到的余数。

左侧操作数	右侧操作数	输出
String/Int/Deci	String/Int/Deci	如果所有字符串都正确地转换为小数，则返回 Decimal 值。返回左侧操作数除以右侧操作数得到的余数。否则为 Undefined 。
其它值	其它值	Undefined .

函数

您可以使用 SQL 表达式的 SELECT 或 WHERE 子句中的以下内置函数。

abs(Decimal)

返回数字的绝对值。SQL 版本 2015-10-08 及更高版本支持。

示例：abs(-5) 返回 5。

参数类型	结果
Int	Int，参数的绝对值。
Decimal	Decimal，参数的绝对值。
Boolean	Undefined .
String	Decimal。结果是参数的绝对值。如果字符串无法转换，则结果为 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

accountid()

以 String 形式返回拥有该规则的账户的 ID。SQL 版本 2015-10-08 及更高版本支持。

例如：

accountid() = "123456789012"

acos(Decimal)

以弧度为单位返回数字的反余弦值。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：acos(0) = 1.5707963267948966

参数类型	结果
Int	Decimal (双精度)，参数的反余弦值。虚数结果返回 Undefined 。
Decimal	Decimal (双精度)，参数的反余弦值。虚数结果返回 Undefined 。
Boolean	Undefined .
String	Decimal，参数的反余弦值。如果字符串无法转换，则结果为 Undefined 。虚数结果返回 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

asin(Decimal)

以弧度为单位返回数字的反正弦值。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：asin(0) = 0.0

参数类型	结果
Int	Decimal (双精度)，参数的反正弦值。虚数结果返回 Undefined 。
Decimal	Decimal (双精度)，参数的反正弦值。虚数结果返回 Undefined 。
Boolean	Undefined .
String	Decimal (双精度)，参数的反正弦值。如果字符串无法转换，则结果为 Undefined 。虚数结果返回 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

atan(Decimal)

以弧度为单位返回数字的反正切值。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：atan(0) = 0.0

参数类型	结果
Int	Decimal (双精度)，参数的反正切值。虚数结果返回 Undefined 。
Decimal	Decimal (双精度)，参数的反正切值。虚数结果返回 Undefined 。

参数类型	结果
Boolean	Undefined .
String	Decimal，参数的反正切值。如果字符串无法转换，则结果为 Undefined 。虚数结果返回 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

atan2(Decimal, Decimal)

以弧度的形式返回 x 轴正方向与由两个参数定义的 (x,y) 点之间的角度。 逆时针的角，角度为正数（上半平面，y > 0），顺时针的角，角度为负数（下半平面，y < 0）。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：atan2(1, 0) = 1.5707963267948966

参数类型	参数类型	结果
Int/Decimal	Int/Decimal	Decimal (双精度)，x 轴和指定间的角度。
Int/Decimal/String	Int/Decimal/String	Decimal，所描述点的反正切值串无法转换，则结果为 Undefi
其它值	其它值	Undefined .

aws_lambda(functionArn, inputJson)

调用指定的 Lambda 函数并将 inputJson 传递给 Lambda 函数，并且返回 Lambda 函数生成的 JSON。

参数

参数	描述
`functionArn`	要调用的 Lambda 函数的 ARN。Lambda 函数必须返回 JSON 数据。
`inputJson`	传递到 Lambda 函数的 JSON 输入。要传递嵌套对象查询和文本，您必须使用 SQL 版本 2016-03-23。

您必须授予调用指定的 Lambda 函数的 AWS IoT `lambda:InvokeFunction`权限。以下示例说明如何使用 AWS CLI授予 `lambda:InvokeFunction` 权限：

```
aws lambda add-permission --function-name "function_name"
--region "region"
--principal iot.amazonaws.com
--source-arn arn:aws:iot:us-east-1:account_id:rule/rule_name
--source-account "account_id"
--statement-id "unique_id"
--action "lambda:InvokeFunction"
```

add-permission 命令的参数如下：

--function-name

Lambda 函数的名称 添加新的权限来更新函数的资源策略。

--region

您的账户的。 AWS 区域

--principal

获取权限的委托人。这应该是`iot.amazonaws.com`为了 AWS IoT 允许调用 Lambda 函数。

--source-arn

规则的 ARN。您可以使用get-topic-rule AWS CLI 命令获取规则的 ARN。

--source-account

定义规则 AWS 账户 的地方。

--statement-id

唯一的语句标识符。

--action

要在此声明中允许的 Lambda 操作。要允许 AWS IoT 调用 Lambda 函数，请指定 `lambda:InvokeFunction`。

> **Important**
>
> 如果您在不提供`source-arn`或的情况下为 AWS IoT 委托人添加权限`source-account`，则任何 AWS 账户 使用您的 Lambda 操作创建规则的人都可以触发规则，从中调用您的 Lambda 函数。 AWS IoT有关更多信息，请参阅 Lambda 权限模型。

假设 JSON 消息负载如下所示：

```
{
    "attribute1": 21,
    "attribute2": "value"
}
```

`aws_lambda` 函数可用于调用 Lambda 函数，如下所示。

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function",
 {"payload":attribute1}) as output FROM 'topic-filter'
```

如果您想要传递完整的 MQTT 消息负载，您可以使用“*”指定 JSON 负载，如下示例所示。

```
SELECT
aws_lambda("arn:aws:lambda:us-east-1:account_id:function:lambda_function", *) as output
 FROM 'topic-filter'
```

`payload.inner.element` 从在主题“topic/subtopic”上发布的消息中选择数据。

`some.value` 从 Lambda 函数生成的输出中选择数据。

> **Note**
>
> 规则引擎限制 Lambda 函数的执行持续时间。从规则执行的 Lambda 函数调用应在 2000 毫秒内完成。

bitand(Int, Int)

在两个 Int (转换成的) 参数的位表示之间逐位执行与运算。SQL 版本 2015-10-08 及更高版本支持。

示例：bitand(13, 5) = 5

参数类型	参数类型	结果
Int	Int	Int，对两个参数逐位执行与运
Int/Decimal	Int/Decimal	Int，对两个参数逐位执行与运 非 Int 数字向下舍入至最近的 I 果任意参数不能转换为 Int，则 Undefined 。
Int/Decimal/String	Int/Decimal/String	Int，对两个参数逐位执行与运 符串转换为小数并向下舍入至最 如果转换失败，结果为 Undefi
其它值	其它值	Undefined .

bitor(Int, Int)

在两个参数的位表示之间逐位执行或运算。SQL 版本 2015-10-08 及更高版本支持。

示例：bitor(8, 5) = 13

参数类型	参数类型	结果
Int	Int	Int，对两个参数逐位执行或运
Int/Decimal	Int/Decimal	Int，对两个参数逐位执行或运 Int 数字向下舍入至最近的 Int 失败，结果为 Undefined 。
Int/Decimal/String	Int/Decimal/String	Int，对两个参数逐位执行或运 符串转换为小数并向下舍入至最 如果转换失败，结果为 Undefi

参数类型	参数类型	结果
其它值	其它值	Undefined .

bitxor(Int, Int)

在两个 Int (转换成的) 参数的位表示之间逐位执行异或运算。SQL 版本 2015-10-08 及更高版本支持。

示例：bitor(13, 5) = 8

参数类型	参数类型	结果
Int	Int	Int，对两个参数逐位执行异或
Int/Decimal	Int/Decimal	Int，对两个参数逐位执行异或 Int 数字向下舍入至最近的 Int
Int/Decimal/String	Int/Decimal/String	Int，对两个参数逐位执行异或 串转换为小数并向下舍入到最近 果任何转换失败，结果为 Undef
其它值	其它值	Undefined .

bitnot(Int)

对 Int (转换成的) 参数的位表示逐位执行非运算。SQL 版本 2015-10-08 及更高版本支持。

示例：bitnot(13) = 2

参数类型	结果
Int	Int，对参数逐位执行非运算。
Decimal	Int，对参数逐位执行非运算。Decimal 值会向下舍入至最近的 Int。

参数类型	结果
String	Int，对参数逐位执行非运算。字符串转换为小数值并向下舍入至最近的 Int。如果任何转换失败，结果为 Undefined 。
其它值	其它值。

cast()

将值从一个数据类型转换为另一个数据类型。强制转换与标准转换大体相同，但是增加了在数字与布尔值之间进行强制转换的能力。如果 AWS IoT 无法确定如何将一种类型转换为另一种类型，则结果为Undefined。SQL 版本 2015-10-08 及更高版本支持。格式：cast(# as ##)。

例如：

```
cast(true as Int) = 1
```

在调用 cast 时，“as”后面可能出现以下关键字：

对于 SQL 版本 2015-10-08 和 2016-03-23

Keyword	结果
String	将值强制转换为 String。
Nvarchar	将值强制转换为 String。
文本	将值强制转换为 String。
Ntext	将值强制转换为 String。
varchar	将值强制转换为 String。
Int	将值强制转换为 Int。
整数	将值强制转换为 Int。
Double	将值强制转换为 Decimal（双精度）。

此外，对于 SQL 版本 2016-03-23

Keyword	结果
`Decimal`	将值强制转换为 `Decimal`。
布尔型	将值强制转换为 `Boolean`。
`Boolean`	将值强制转换为 `Boolean`。

强制转换规则：

强制转换为 decimal

参数类型	结果
`Int`	没有小数点的 `Decimal`。
`Decimal`	源值。 **Note** 对于 SQL V2 (2016-03-23)，整数数字值，例如 `10.0`，将返回一个 `Int` 值 (`10`)，而不是预期的 `Decimal` 值 (`10.0`)。若要可靠地将整数数值转换为 `Decimal` 值，请使用 SQL V1 (2015-10-08) 作为规则查询语句。
`Boolean`	true = 1.0，false = 0.0。
`String`	尝试将字符串解析为 `Decimal`。 AWS IoT 会尝试解析与正则表达式相匹配的字符串：^-?\d+(\.\d+)?((?i)E-?\d+)?$。可自动转换为小数的字符串示例包括 "0"、"-1.2"、"5E-12"。
数组	`Undefined` .

参数类型	结果
对象	Undefined .
Null	Undefined .
未定义	Undefined .

强制转换为 int

参数类型	结果
Int	源值。
Decimal	源值，向下舍入到最近的 Int。
Boolean	true = 1.0，false = 0.0。
String	尝试将字符串解析为 Decimal。 AWS IoT 会尝试解析与正则表达式相匹配的字符串：^-?\d+(\.\d+)?((?i)E-?\d+)?$。可自动转换为小数的字符串示例包括 "0"、"-1.2"、"5E-12"。Int 尝试将字符串转换为 AWS IoT ，然后向下舍入到最近的 Decimal。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

强制转换为 **Boolean**

参数类型	结果
Int	0 = False，任何非零值 = True。

参数类型	结果
Decimal	0 = False，任何非零值 = True。
Boolean	源值。
String	"true" = True 和 "false" = False（不区分大小写）。其它字符串值 = Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

强制转换为 string

参数类型	结果
Int	标准表示法中 Int 的字符串表示形式。
Decimal	科学表示法中 Decimal 值的字符串表示。
Boolean	"true" 或 "false"，全小写。
String	"true"=True 和 "false"=False (不区分大小写)。其它字符串值 = Undefined 。
数组	数组序列化为 JSON。结果字符串为逗号分隔的列表，括在方括号中。 String 带引号。 Decimal、Int 和 Boolean 不带引号。
对象	序列化为 JSON 的对象。JSON 字符串为键值对的逗号分隔列表，以大括号开头和结束。String 带引

参数类型	结果
	号。Decimal、Int、Boolean 和 Null 不带引号。
Null	Undefined .
未定义	Undefined .

ceil(Decimal)

将给定的 Decimal 向上舍入到最近的 Int。SQL 版本 2015-10-08 及更高版本支持。

示例：

ceil(1.2) = 2

ceil(-1.2) = -1

参数类型	结果
Int	Int，参数值。
Decimal	Int，Decimal 值向上舍入到最近的 Int。
String	Int。字符串将转换为 Decimal 并向上舍入到最近的 Int。如果字符串无法转换为 Decimal，则结果为 Undefined 。
其它值	Undefined .

chr(String)

返回给定 Int 参数对应的 ASCII 字符。SQL 版本 2015-10-08 及更高版本支持。

示例：

chr(65) = "A"。

chr(49) = "1"。

参数类型	结果
Int	与指定的 ASCII 值对应的字符。如果参数不是有效的 ASCII 值，则结果为 Undefined 。
Decimal	与指定的 ASCII 值对应的字符。Decimal 参数会向下舍入至最近的 Int。如果参数不是有效的 ASCII 值，则结果为 Undefined 。
Boolean	Undefined .
String	如果 String 可以转换为 Decimal，则向下舍入到最近的 Int。如果参数不是有效的 ASCII 值，则结果为 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
其它值	Undefined .

clientid()

返回发送消息的 MQTT 客户端的 ID，如果未通过 MQTT 发送消息，则返回 n/a。SQL 版本 2015-10-08 及更高版本支持。

例如：

```
clientid()  = "123456789012"
```

concat()

联接数组或字符串。该函数可接受任意数量的参数，并返回 String 或 Array。SQL 版本 2015-10-08 及更高版本支持。

示例：

concat() = Undefined.

concat(1) = "1"。

concat([1, 2, 3], 4) = [1, 2, 3, 4]。

concat([1, 2, 3], "hello") = [1, 2, 3, "hello"]

concat("con", "cat") = "concat"

concat(1, "hello") = "1hello"

concat("he","is","man") = "heisman"

concat([1, 2, 3], "hello", [4, 5, 6]) = [1, 2, 3, "hello", 4, 5, 6]

参数数量	结果
0	Undefined .
1	不经修改返回参数。
2+	如果任意参数为 Array，那么结果为包含所有参数的一个数组。如果没有参数为数组，并且至少有一个参数为 String，则结果是所有参数的 String 表示的联接。参数将使用上文列出的标准转换来转换为字符串。

cos(Decimal)

以弧度为单位返回数字的余弦值。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

例如：

cos(0) = 1。

参数类型	结果
Int	Decimal (双精度)，参数的余弦值。虚数结果返回 Undefined 。

参数类型	结果
Decimal	Decimal (双精度)，参数的余弦值。虚数结果返回 Undefined 。
Boolean	Undefined .
String	Decimal (双精度)，参数的余弦值。如果字符串无法转换为 Decimal，则结果为 Undefined 。虚数结果返回 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

cosh(Decimal)

以弧度为单位返回数字的双曲余弦值。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：cosh(2.3) = 5.037220649268761。

参数类型	结果
Int	Decimal (双精度)，参数的双曲余弦值。虚数结果返回 Undefined 。
Decimal	Decimal (双精度)，参数的双曲余弦值。虚数结果返回 Undefined 。
Boolean	Undefined .
String	Decimal (双精度)，参数的双曲余弦值。如果字符串无法转换为 Decimal，则结果为

参数类型	结果
	Undefined 。虚数结果返回 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

解码（值，decodingScheme）

使用 decode 函数来解码一个编码的值。如果解码字符串为 JSON 文档，则返回可寻址对象。否则，解码字符串会返回为字符串。如果字符串无法解码，则函数返回 NULL。此函数支持解码 base64 编码的字符串和协议缓冲区 (protobuf) 消息格式。

SQL 版本 2016-03-23 及更高版本支持。

value

字符串值或任何有效的表达式，如 AWS IoT SQL 参考 中所定义，将返回一个字符串。

decodingScheme

表示用于解码值的方案的文字字符串。目前，只支持 'base64' 和 'proto'。

解码 base64 编码字符串

在此示例中，消息负载包含一个编码值。

```
{
    encoded_temp: "eyAidGVtcGVyYXR1cmUiOiAzMyB9Cg=="
}
```

此 SQL 语句中的 decode 函数解码消息有效负载中的值。

```
SELECT decode(encoded_temp,"base64").temperature AS temp from 'topic/subtopic'
```

解码 encoded_temp 值会生成以下有效的 JSON 文档，该文档允许 SELECT 语句读取温度值。

```
{ "temperature": 33 }
```

此处显示了此示例中 SELECT 语句的结果。

```
{ "temp": 33 }
```

如果解码的值不是有效的 JSON 文档，则解码的值将作为字符串返回。

解码 protobuf 消息有效负载

您可以使用解码 SQL 函数来配置可以解码 protobuf 消息有效负载的规则。有关更多信息，请参阅[解码 protobuf 消息有效负载]。

此函数签名类似以下内容：

```
decode(<ENCODED DATA>, 'proto', '<S3 BUCKET NAME>', '<S3 OBJECT KEY>', '<PROTO NAME>',
 '<MESSAGE TYPE>')
```

`ENCODED DATA`

指定要解码的 protobuf 编码数据。如果发送到规则的整个消息都是 protobuf 编码的数据，则可以使用 * 引用原始二进制传入有效负载。否则，此字段必须是 base-64 编码的 JSON 字符串，并且可以直接传入对该字符串的引用。

1) 要解码原始二进制 protobuf 传入有效负载，请执行以下操作：

```
decode(*, 'proto', ...)
```

2) 要解码由 base64 编码的字符串 'a.b' 表示的 protobuf 编码的消息，请执行以下操作：

```
decode(a.b, 'proto', ...)
```

`proto`

以 protobuf 消息格式指定要解码的数据。如果您指定 `base64` 而不是 `proto`，此函数会将 base64 编码的字符串解码为 JSON。

`S3 BUCKET NAME`

您上传 `FileDescriptorSet` 文件的 Amazon S3 存储桶的名称。

S3 OBJECT KEY

指定 Amazon S3 存储桶中 FileDescriptorSet 文件的对象键。

PROTO NAME

生成 FileDescriptorSet 文件的 .proto 文件名（不包括扩展名）。

MESSAGE TYPE

FileDescriptorSet 文件中 protobuf 消息结构的名称，要解码的数据应符合该结构。

使用解码 SQL函数的示例 SQL 表达式可能如下所示：

```
SELECT VALUE decode(*, 'proto', 's3-bucket', 'messageformat.desc', 'myproto',
 'messagetype') FROM 'some/topic'
```

- *

  表示二进制传入负载，它符合名为 mymessagetype 的 protobuf 消息类型。

- messageformat.desc

  FileDescriptorSet 文件存储在名为 s3-bucket 的 Amazon S3 存储桶中。

- myproto

  用于生成名为 myproto.proto 的 FileDescriptorSet 文件的原始 .proto 文件。

- messagetype

  myproto.proto 中定义的名为 messagetype 的消息类型（以及任何导入的依赖项）。

encode(value, encodingScheme)

根据编码方案，使用 encode 函数将负载（可能是非 JSON 数据）编码为字符串表示形式。SQL 版本 2016-03-23 及更高版本支持。

value

AWS IoT SQL 参考 中所定义的任何有效的表达式。您可以指定 * 以对整个负载进行编码，无论它是否为 JSON 格式。如果您提供了表达式，评估的结果将在编码之前转换为字符串。

encodingScheme

代表您要使用的编码方案的文字字符串。目前仅支持 'base64'。

endswith(String, String)

返回 Boolean 来表示第一个 String 参数是否以第二个 String 参数结尾。如果任一参数为 Null 或 Undefined，则结果为 Undefined。SQL 版本 2015-10-08 及更高版本支持。

示例：endswith("cat","at") = true。

参数类型 1	参数类型 2	结果
String	String	如果第一个参数以第二个参数结 true。否则为 false。
其它值	其它值	两个参数都使用标准转换规则转 符串。如果第一个参数以第二个 尾，则为 true。否则为 false。如 参数为 Null 或 Undefined ， Undefined 。

exp(Decimal)

返回 e 的 Decimal 参数次方。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：exp(1) = e。

参数类型	结果
Int	Decimal (双精度)，e ^ 参数。
Decimal	Decimal (双精度)，e ^ 参数。
String	Decimal (双精度)，e ^ 参数。如果 String 无法转换为 Decimal，则结果为 Undefined 。

参数类型	结果
其它值	Undefined .

floor(Decimal)

将给定的 Decimal 向下舍入到最接近的 Int。SQL 版本 2015-10-08 及更高版本支持。

示例：

floor(1.2) = 1

floor(-1.2) = -2

参数类型	结果
Int	Int，参数值。
Decimal	Int，Decimal 值会向下舍入至最接近的 Int。
String	Int。字符串将转换为 Decimal 并向下舍入到最接近的 Int。如果字符串无法转换为 Decimal，则结果为 Undefined 。
其它值	Undefined .

get

从一个集合数据类型 (数组、字符串、对象) 中提取值。第一个参数不会进行任何转换。根据表中的记载对第二个参数进行转换。SQL 版本 2015-10-08 及更高版本支持。

示例：

get(["a", "b", "c"], 1) = "b"

get({"a":"b"}, "a") = "b"

get("abc", 0) = "a"

参数类型 1	参数类型 2	结果
数组	任何类型 (转换为 Int)	在第二个参数 (已经转换为 Arra Int 中，从零开始索引得到的项 换已不成功，结果为 Undefine 引超出 Array 的范围 (负值或大 长度)，则结果为 Undefined 。
String	任何类型 (转换为 Int)	在第二个参数 (已经转换为 Int) 串中，从零开始索引得到的字符 已不成功，结果为 Undefined 超出字符串的范围 (负值或大于等 长度)，则结果为 Undefined 。
对象	String (不进行转换)	第一个参数对象中存储的值与作 数提供的字符串键相对应。
其它值	任意值	Undefined .

get_dynamodb (tableName、、、、、roLearn partitionKeyName ） partitionKeyValue sortKeyName sortKeyValue

从 DynamoDB 表中检索数据。get_dynamodb( ) 可让您在计算规则时查询 DynamoDB 表。您可以使用从 DynamoDB 中检索到的数据筛选或增加消息负载。SQL 版本 2016-03-23 及更高版本支持。

get_dynamodb( ) 接受以下参数：

tableName

要查询的 DynamoDB 表的名称。

partitionKeyName

分区键的名称。有关更多信息，请参阅 DynamoDB Keys。

partitionKeyValue

用于标识记录的分区键的值。有关更多信息，请参阅 DynamoDB Keys。

sortKeyName

（可选）排序键的名称。仅当查询的 DynamoDB 表使用复合键时需要此参数。有关更多信息，请参阅 DynamoDB Keys。

sortKeyValue

（可选）排序键的值。仅当查询的 DynamoDB 表使用复合键时需要此参数。有关更多信息，请参阅 DynamoDB Keys。

roleArn

授予对 DynamoDB 表的访问权限的 IAM 角色的 ARN。规则引擎将承担此角色以代表您访问 DynamoDB 表。避免使用过于宽容的角色。仅向角色授予规则所需的那些权限。以下是授予对一个 DynamoDB 表的访问权限的示例策略。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "dynamodb:GetItem",
            "Resource": "arn:aws:dynamodb:aws-region:account-id:table/table-name"
        }
    ]
}}
```

举例说明您如何使用 get_dynamodb()，例如您有一个 DynamoDB 表，其中包含连接到 AWS IoT的所有设备的设备 ID 和位置信息。以下 SELECT 语句使用 get_dynamodb() 函数检索指定的设备 ID 的位置：

```
SELECT *, get_dynamodb("InServiceDevices", "deviceId", id,
"arn:aws:iam::12345678910:role/getdynamo").location AS location FROM 'some/
topic'
```

Note

- 每个 SQL 语句最多可以调用 get_dynamodb() 一次。在单个 SQL 语句中调用 get_dynamodb() 多次会导致规则在不调用任何操作的情况下终止。
- 如果 get_dynamodb() 返回 8 KB 以上的数据，则无法调用规则的操作。

get_mqtt_property(name)

引用以下任何 MQTT5 标头：`contentType`、`payLoadFormatIndicator`、`responseTopic` 和 `correlationData`。此函数将以下任何文字字符串作为参数：`content_type`、`format_indicator`、`response_topic` 和 `correlation_data`。有关更多信息，请参阅以下函数参数表。

contentType

字符串：一个 UTF-8 编码的字符串，用于描述发布消息的内容。

payLoadFormat指示器

字符串：一个枚举字符串值，指示有效负载是否格式化为 UTF-8。有效值为 `UNSPECIFIED_BYTES` 和 `UTF8_DATA`。

responseTopic

字符串：一个 UTF-8 编码的字符串，用作响应消息的主题名称。响应主题用于描述接收方应作为请求/响应流程的一部分发布到的主题。主题不得包含通配符。

correlationData

字符串：请求消息的发送方使用 base64 编码的二进制数据来识别收到响应消息时响应消息的请求。

下表显示了可接受的函数参数，及其与 `get_mqtt_property` 函数相关的返回类型：

函数参数

SQL	返回的数据类型（如果存在）	返回的数据类型（如果不存在）
`get_mqtt_property("format_indicator")`	字符串（UNSPECIFIED_BYTES 或 UTF8_DATA）	字符串 (UNSPECIFIED_BYTES)
`get_mqtt_property("content_type")`	String	未定义
`get_mqtt_property("response_topic")`	String	未定义

SQL	返回的数据类型（如果存在）	返回的数据类型（如果不存在）
`get_mqtt_property("correlation_data")`	base64 编码的字符串	未定义
`get_mqtt_property("some_invalid_name")`	未定义	未定义

以下示例规则 SQL 引用下列任何 MQTT5 标头：`contentType`、`payLoadFormatIndicator`、`responseTopic` 和 `correlationData`。

```
SELECT *, get_mqtt_property('content_type') as contentType,
          get_mqtt_property('format_indicator') as payloadFormatIndicator,
          get_mqtt_property('response_topic') as responseTopic,
          get_mqtt_property('correlation_data') as correlationData
FROM 'some/topic'
```

get_secret(secretId, secretType, key, roleArn)

在 AWS Secrets Manager 中检索密钥当前版本的加密 `SecretString` 或者 `SecretBinary`字段中的值。有关创建和维护密钥的更多信息，请参阅CreateSecretUpdateSecret、和PutSecretValue。

`get_secret()` 接受以下参数：

secretId

字符串：要检索的密钥的 Amazon Resource Name (ARN) 或友好名称。

secretType

字符串：密钥类型。有效值：`SecretString` | `SecretBinary`。

SecretString

- 对于您使用 API、或 AWS Secrets Manager 控制台创建为 JSON 对象的密钥： AWS CLI
  - 如果为 `key` 参数指定值，则函数返回指定键的值。
  - 如果您没有为 `key` 参数指定值，则函数将返回整个 JSON 对象。
- 对于通过使用 API 或 AWS CLI创建的非 JSON 对象：

- 如果为 key 参数指定值，则函数失败并出现异常。
- 如果您没有为 key 参数指定值，则函数返回密钥的内容。

SecretBinary

- 如果为 key 参数指定值，则函数失败并出现异常。
- 如果您没有为 key 参数指定值，则函数将密钥值作为 base64 编码 UTF-8 字符串返回。

key

（可选）字符串：存储在密钥的 SecretString 字段中的 JSON 对象里的键名称。如果您希望仅检索存储在密钥中而不是整个 JSON 对象中的键值，请使用此值。

如果您为此参数指定了一个值，而密钥的 SecretString 字段中不包含 JSON 对象，此函数将失败并出现异常。

roleArn

String：具有 secretsmanager:GetSecretValue 和 secretsmanager:DescribeSecret 权限的角色 ARN。

> **Note**
>
> 此函数始终返回密钥的当前版本（带有 AWSCURRENT 标签的版本）。 AWS IoT 规则引擎将每个密钥缓存最多 15 分钟。因此，规则引擎最多可能需要 15 分钟才能更新密钥。这意味着，如果您在更新后最多 15 分钟内检索到密钥 AWS Secrets Manager，则此函数可能会返回之前的版本。
>
> 此功能不是按流量计费的，但会 AWS Secrets Manager 收费。由于密钥缓存机制，规则引擎偶尔会调用 AWS Secrets Manager。由于规则引擎是完全分布的服务，因此您可能会在 15 分钟缓存窗口期间看到规则引擎中的多个 Secrets Manager API 调用。

示例：

您可以在 HTTPS 规则操作中的身份验证标头里使用 get_secret 函数，如下面的 API 密钥身份验证示例所示。

```
"API_KEY": "${get_secret('API_KEY', 'SecretString', 'API_KEY_VALUE',
 'arn:aws:iam::12345678910:role/getsecret')}"
```

有关 HTTPS 规则操作的更多信息，请参阅 the section called “HTTP”。

get_thing_shadow(thingName, shadowName, roleARN)

返回指定事物的指定影子。SQL 版本 2016-03-23 及更高版本支持。

thingName

String：您要检索其影子的事物的名称。

shadowName

（可选）字符串：影子的名称。只有在引用命名的影子时，才需要使用该参数。

roleArn

String：具有 `iot:GetThingShadow` 的 ARN 角色。

示例：

与命名的影子一起使用时，请提供 `shadowName` 参数。

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","MyThingShadow","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

与未命名的影子一起使用时，请省略 `shadowName` 参数。

```
SELECT * from 'topic/subtopic'
WHERE
    get_thing_shadow("MyThing","arn:aws:iam::123456789012:role/
AllowsThingShadowAccess")
    .state.reported.alarm = 'ON'
```

get_user_properties () userPropertyKey

引用用户属性，这是 MQTT5 中支持的一种类型的属性标头。

userProperty

字符串：用户属性是一个键-值对。此函数将键作为参数，并返回与关联的键匹配的所有值的数组。

函数参数

对于邮件标题中的以下用户属性：

键	值
某个键	某个值
其他键	其他值
某个键	带有重复键的值

下表显示了预期的 SQL 行为：

SQL	返回的数据类型	返回的数据值
get_user_properties('some key')	字符串数组	`['some value', 'value with duplicate key']`
get_user_properties('other key')	字符串数组	`['a different value']`
get_user_properties( )	键值对对象的数组	`[{'"some key": "some value"'}, {"other key": "a different value"}, {"some key": "value with duplicate key"}]`
get_user_properties('non-existent key')	未定义	

以下示例规则 SQL 将用户属性（一种类型的 MQTT5 属性标头）引用到有效负载中：

```
SELECT *, get_user_properties('user defined property key') as userProperty
FROM 'some/topic'
```

哈希函数

AWS IoT 提供以下哈希函数：

- md2
- md5
- sha1
- sha224
- sha256
- sha384
- sha512

所有哈希函数都可以输入一个字符串参数。结果为该字符串的哈希值。对非字符串参数进行标准字符串转换。所有哈希函数在 SQL 版本 2015-10-08 及更高版本中均受支持。

示例：

`md2("hello")` = "a9046c73e00331af68917d3804f70655"

`md5("hello")` = "5d41402abc4b2a76b9719d911017c592"

indexof(String, String)

返回第二个参数的第一个索引 (从零开始) 作为第一个参数的子字符串。两个参数均为字符串。如果参数数据类型不是字符串，则应用标准字符串转换规则进行转换。此函数只对字符串有效，不适用于数组。SQL 版本 2016-03-23 及更高版本支持。

示例：

`indexof("abcd", "bc")` = 1

isNull()

如果参数为 `Null` 值，则返回 true。SQL 版本 2015-10-08 及更高版本支持。

示例：

`isNull(5)` = false。

`isNull(Null)` = true。

参数类型	结果
`Int`	false

参数类型	结果
Decimal	false
Boolean	false
String	false
Array	false
Object	false
Null	true
Undefined	false

isUndefined()

如果参数为 Undefined，则返回 true。SQL 版本 2016-03-23 及更高版本支持。

示例：

`isUndefined(5)` = false。

`isUndefined(floor([1,2,3]))` = true。

参数类型	结果
Int	false
Decimal	false
Boolean	false
String	false
Array	false
Object	false
Null	false

参数类型	结果
Undefined	true

length(String)

返回输入字符数中的字符数。对非 String 参数应用标准转换规则。SQL 版本 2016-03-23 及更高版本支持。

示例：

length("hi") = 2

length(false) = 5

ln(Decimal)

返回参数的自然对数。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：ln(e) = 1。

参数类型	结果
Int	Decimal (双精度)，参数的自然对数。
Decimal	Decimal (双精度)，参数的自然对数。
Boolean	Undefined .
String	Decimal (双精度)，参数的自然对数。如果字符串无法转换为 Decimal，则结果为 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .

参数类型	结果
未定义	Undefined .

log(Decimal)

返回参数的以 10 为底的对数。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：log(100) = 2.0。

参数类型	结果
Int	Decimal (双精度)，参数以 10 为底的对数。
Decimal	Decimal (双精度)，参数以 10 为底的对数。
Boolean	Undefined .
String	Decimal (双精度)，参数以 10 为底的对数。如果 String 无法转换为 Decimal，则结果为 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

lower(String)

返回给定 String 的小写版本。非字符串参数使用标准转换规则转换为字符串。SQL 版本 2015-10-08 及更高版本支持。

示例：

`lower("HELLO")` = "hello"。

`lower(["HELLO"])` = "[\"hello\"]"。

lpad(String, Int)

返回 `String` 参数，在输入参数的左侧填充由第二个参数指定的数量的空格。`Int` 参数必须介于 0 到 1000 之间。如果提供的值在这一有效范围之外，则参数被设置为与其最近的值（0 或 1000）。SQL 版本 2015-10-08 及更高版本支持。

示例：

`lpad("hello", 2)` = " hello".

`lpad(1, 3)` = " 1"

参数类型 1	参数类型 2	结果
`String`	`Int`	`String`，在输入 `String` 左侧填 指定的数量的空格。
`String`	`Decimal`	`Decimal` 参数向下舍入到最近的 在 `String` 左侧填充指定数量的
`String`	`String`	第二个参数被转换为 `Decimal`， 到最近的 `Int`，并在 `String` 左 定数量的空格。如果第二个参数 `Int`，则结果为 `Undefined` 。
其它值	`Int/Decimal/String`	第一个值使用标准转换规则转换 `String`，然后对该 `String` 应用 函数。如果它无法转换，则结果 `Undefined` 。
任意值	其它值	`Undefined` .

ltrim(String)

从提供的 `String` 中删除所有前导空白（制表符和空格）。SQL 版本 2015-10-08 及更高版本支持。

例如：

Ltrim(" h i ") = "hi "。

参数类型	结果
Int	删除了所有前导空白的 Int 的 String 表示形式。
Decimal	删除了所有前导空白的 Decimal 的 String 表示形式。
Boolean	布尔值（“true”或“false”）在删除所有前导空格之后的 String 表示形式。
String	删除所有前导空白之后的参数。
数组	Array（使用标准转换规则）在删除所有前导空白之后的 String 表示形式。
对象	对象（使用标准转换规则）在删除所有前导空白之后的 String 表示形式。
Null	Undefined .
未定义	Undefined .

machinelearning_predict(modelId, roleArn, record)

使用该machinelearning_predict函数使用基于 Amazon SageMaker 模型的 MQTT 消息中的数据进行预测。SQL 版本 2015-10-08 及更高版本支持。machinelearning_predict 函数的参数如下：

modelId

对其运行预测的模型的 ID。必须启用模型的实时终端节点。

roleArn

IAM 角色，拥有具备 machinelearning:Predict 和 machinelearning:GetMLModel 权限的策略并允许访问运行预测所针对的模型。

记录

要传递到 Pred SageMaker ict API 的数据。该参数应表示为单层 JSON 对象。如果记录是多级 JSON 对象，该记录将通过序列化其值来进行平展。例如，以下 JSON：

```
{ "key1": {"innerKey1": "value1"}, "key2": 0}
```

会变为：

```
{ "key1": "{\"innerKey1\": \"value1\"}", "key2": 0}
```

该函数返回具有以下字段的 JSON 对象：

predictedLabel

基于模型的输入分类。

details

包含以下属性：

PredictiveModelType

模型类型。有效值为 REGRESSION、BINARY、MULTICLASS。

算法

用于进行预测 SageMaker 的算法。该值必须为 SGD。

predictedScores

包含与每个标签对应的原始分类分数。

predictedValue

预测的值 SageMaker。

mod(Decimal, Decimal)

返回第一个参数除以第二个参数的余数。等效于 remainder(Decimal, Decimal)。您还可以使用“%”作为相同取模功能的插入运算符。SQL 版本 2015-10-08 及更高版本支持。

示例：mod(8, 3) = 2。

左侧操作数	右侧操作数	输出
Int	Int	Int，第一个参数对第二个参数取
Int/Decimal	Int/Decimal	Decimal，第一个参数对第二个参数取模。
String/Int/Decimal	String/Int/Decimal	如果所有字符串转换为小数，则为第一个参数对第二个参数取模的值。否则 Undefined 。
其它值	其它值	Undefined .

nanvl (AnyValue,) AnyValue

如果第一个参数为有效的 Decimal，则返回第一个参数。否则，返回第二个参数。SQL 版本 2015-10-08 及更高版本支持。

示例：Nanvl(8, 3) = 8。

参数类型 1	参数类型 2	输出
未定义	任意值	第二个参数。
Null	任意值	第二个参数。
Decimal (NaN)	任意值	第二个参数。
Decimal (非 NaN)	任意值	第一个参数。
其它值	任意值	第一个参数。

newuuid()

返回随机的 16 字节 UUID。SQL 版本 2015-10-08 及更高版本支持。

示例：newuuid() = 123a4567-b89c-12d3-e456-789012345000

numbytes(String)

返回输入字符串 UTF-8 编码中的字节数。对非 `String` 参数应用标准转换规则。SQL 版本 2016-03-23 及更高版本支持。

示例：

`numbytes("hi")` = 2

`numbytes("€")` = 3

parse_time(String, Long[, String])

使用 `parse_time` 函数可将时间戳的格式设置为人类可读的日期/时间格式。SQL 版本 2016-03-23 及更高版本支持。要将时间戳字符串转换为毫秒，请参阅 time_to_epoch(String, String)。

`parse_time` 函数采用下列参数：

模式

（字符串）遵循 Joda 时间格式的日期/时间模式。

时间戳

(Long) 要采用自 Unix 纪元时间以来的毫秒数格式表示的时间。请参阅函数 timestamp()。

timezone

（字符串）采用日期/时间格式的时区。默认值为“UTC”。此函数支持 Joda-Time 时区。此参数是可选的。

示例：

在将此消息发布到主题“A/B”时，负载 `{"ts": "1970.01.01 AD at 21:46:40 CST"}` 发送到 S3 存储桶：

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT parse_time(\"yyyy.MM.dd G 'at' HH:mm:ss z\", 100000000,
 'America/Belize' ) as ts FROM 'A/B'",

        "ruleDisabled": false,
        "awsIotSqlVersion": "2016-03-23",
```

```
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": "BUCKET_NAME",
                    "key": "KEY_NAME"
                }
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

在将此消息发布到主题“A/B”时，与 {"ts": "2017.06.09 AD at 17:19:46 UTC"} 类似（但具有当前日期/时间）的负载发送到 S3 存储桶：

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
        "sql": "SELECT parse_time(\"yyyy.MM.dd G 'at' HH:mm:ss z\", timestamp() ) as ts
 FROM 'A/B'",
        "awsIotSqlVersion": "2016-03-23",
        "ruleDisabled": false,
        "actions": [
            {
                "s3": {
                    "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                    "bucketName": "BUCKET_NAME",
                    "key": "KEY_NAME"
                }
            }
        ],
        "ruleName": "RULE_NAME"
    }
}
```

parse_time() 也可用作替换模板。例如，在将此消息发布到主题“A/B”时，负载发送到密钥为“2017”的 S3 存储桶：

```
{
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
    "topicRulePayload": {
```

```
        "sql": "SELECT * FROM 'A/B'",
        "awsIotSqlVersion": "2016-03-23",
        "ruleDisabled": false,
        "actions": [{
            "s3": {
                "roleArn": "arn:aws:iam::ACCOUNT_ID:rule:role/ROLE_NAME",
                "bucketName": "BUCKET_NAME",
                "key": "${parse_time('yyyy', timestamp(), 'UTC')}"
            }
        }],
        "ruleName": "RULE_NAME"
    }
}
```

power(Decimal, Decimal)

返回第一个参数的第二个参数次幂的值。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。SQL 版本 2015-10-08 及更高版本支持。

示例：`power(2, 5)` = 32.0。

参数类型 1	参数类型 2	输出
Int/Decimal	Int/Decimal	Decimal (双精度)，返回第一个个参数次幂的值。
Int/Decimal/String	Int/Decimal/String	Decimal (双精度)，返回第一个第二个参数次幂的值。所有字符为小数。如果任何 String 无法 Decimal，则结果为 Undefine
其它值	其它值	Undefined .

principal()

返回设备用于身份验证的委托人，取决于触发消息的发布方式。下表介绍了为每个发布方法和协议返回的委托人。

消息的发布方式	协议	凭证类型
MQTT 客户端	MQTT	X.509 设备证书
AWS IoT 控制台 MQTT 客户端	MQTT	IAM 用户或角色
AWS CLI	HTTP	IAM 用户或角色
AWS IoT 设备软件开发工具包	MQTT	X.509 设备证书
AWS IoT 设备软件开发工具包	MQTT 结束了 WebSocket	IAM 用户或角色

下面的示例展示了 `principal()` 能够返回的不同类型的值：

- X.509 证书指纹：`ba67293af50bf2506f5f93469686da660c7c844e7b3950bfb16813e0d31e9373`
- IAM 角色 ID 和会话名称：`ABCD1EFG3HIJK2LMNOP5:my-session-name`
- 返回用户 ID：`ABCD1EFG3HIJK2LMNOP5`

rand()

返回在 0.0 到 1.0 之间均匀分布的伪随机双精度值。SQL 版本 2015-10-08 及更高版本支持。

例如：

`rand()` = 0.8231909191640703

regexp_matches(String, String)

如果字符串（第一个参数）包含正则表达式（第二个参数）的匹配项，则返回 true。如果您在正则表达式中使用 `|`，请结合使用 `()`。

示例：

`regexp_matches("aaaa", "a{2,}")` = true。

`regexp_matches("aaaa", "b")` = false。

`regexp_matches("aaa", "(aaa|bbb)")` = true。

`regexp_matches("bbb", "(aaa|bbb)")` = true。

`regexp_matches("ccc", "(aaa|bbb)")` = false。

第一个参数：

参数类型	结果
`Int`	`String` 的 `Int` 表示形式。
`Decimal`	`String` 的 `Decimal` 表示形式。
`Boolean`	布尔值 (“true”或“false”) 的 `String` 表示形式。
`String`	`String`。
数组	`Array` (使用标准转换规则) 的 `String` 表示形式。
对象	对象 (使用标准转换规则) 的 `String` 表示形式。
Null	`Undefined` .
未定义	`Undefined` .

第二个参数：

必须是有效的正则表达式。非字符串类型使用标准转换规则转换为 `String`。根据类型，生成的字符串可能不是有效的正则表达式。如果（转换后的）参数不是有效的正则表达式，则结果为 `Undefined`。

regexp_replace(String, String, String)

用第三个参数替换在第一个参数中出现的所有第二个参数 (正则表达式)。用“$”引用捕获组。SQL 版本 2015-10-08 及更高版本支持。

例如：

`regexp_replace("abcd", "bc", "x")` = "axd"。

`regexp_replace("abcd", "b(.*)d", "$1")` = "ac"。

第一个参数：

参数类型	结果
`Int`	`String` 的 `Int` 表示形式。
`Decimal`	`String` 的 `Decimal` 表示形式。
`Boolean`	布尔值（“true”或“false”）的 `String` 表示形式。
`String`	源值。
数组	`Array` (使用标准转换规则) 的 `String` 表示形式。
对象	对象 (使用标准转换规则) 的 `String` 表示形式。
Null	`Undefined` .
未定义	`Undefined` .

第二个参数：

必须是有效的正则表达式。非字符串类型使用标准转换规则转换为 `String`。根据类型，生成的字符串可能不是有效的正则表达式。如果（转换后的）参数不是有效的正则表达式，则结果为 `Undefined`。

第三个参数：

必须是有效的正则表达式替代字符串。（可以引用捕获组。）非字符串类型使用标准转换规则转换为 `String`。如果（转换后的）参数不是有效的正则表达式替代字符串，则结果为 `Undefined`。

regexp_substr(String, String)

在第一个参数中查找第二个参数（正则表达式）的第一个匹配项。用“$”引用捕获组。SQL 版本 2015-10-08 及更高版本支持。

例如：

`regexp_substr("hihihello", "hi") = "hi"`

`regexp_substr("hihihello", "(hi)*") = "hihi"`

第一个参数：

参数类型	结果
`Int`	`String` 的 `Int` 表示形式。
`Decimal`	`String` 的 `Decimal` 表示形式。
`Boolean`	布尔值 (“true”或“false”) 的 `String` 表示形式。
`String`	`String` 参数。
数组	`Array` (使用标准转换规则) 的 `String` 表示形式。
对象	对象 (使用标准转换规则) 的 `String` 表示形式。
Null	`Undefined` .
未定义	`Undefined` .

第二个参数：

必须是有效的正则表达式。非字符串类型使用标准转换规则转换为 `String`。根据类型，生成的字符串可能不是有效的正则表达式。如果（转换后的）参数不是有效的正则表达式，则结果为 `Undefined`。

remainder(Decimal, Decimal)

返回第一个参数除以第二个参数的余数。等效于 mod(Decimal, Decimal)。您还可以使用“%”作为相同取模功能的插入运算符。SQL 版本 2015-10-08 及更高版本支持。

示例：`remainder(8, 3) = 2`。

左侧操作数	右侧操作数	输出
Int	Int	Int，第一个参数对第二个参数取
Int/Decimal	Int/Decimal	Decimal，第一个参数对第二个模。
String/Int/Decimal	String/Int/Decimal	如果所有字符串转换为小数，则一个参数对第二个参数取模的值 Undefined 。
其它值	其它值	Undefined .

replace(String, String, String)

用第三个参数替换在第一个参数中出现的所有第二个参数。SQL 版本 2015-10-08 及更高版本支持。

例如：

`replace("abcd", "bc", "x") = "axd".`

`replace("abcdabcd", "b", "x") = "axcdaxcd".`

所有参数

参数类型	结果
Int	String 的 Int 表示形式。
Decimal	String 的 Decimal 表示形式。
Boolean	布尔值 (“true”或“false”) 的 String 表示形式。
String	源值。
数组	Array (使用标准转换规则) 的 String 表示形式。

参数类型	结果
对象	对象 (使用标准转换规则) 的 String 表示形式。
Null	Undefined .
未定义	Undefined .

rpad(String, Int)

返回字符串参数，在输入参数的右侧填充在第二个参数中指定的数量的空格。Int 参数必须介于 0 到 1000 之间。如果提供的值在这一有效范围之外，则参数被设置为与其最近的值（0 或 1000）。SQL 版本 2015-10-08 及更高版本支持。

示例：

`rpad("hello", 2)` = "hello ".

`rpad(1, 3)` = "1 ".

参数类型 1	参数类型 2	结果
String	Int	在 String 的右侧填充由 Int 指定数量的空格。
String	Decimal	Decimal 参数向下舍入到最近的 Int，并且在字符串的右侧填充由 Int 指定数量的空格。
String	String	第二个参数转换为 Decimal，并向下舍入到最近的 Int。在 String 的右侧填充由 Int 值指定数量的空格。

参数类型 1	参数类型 2	结果
其它值	Int/Decimal/String	第一个值将使用标准转换规则转换为 String，然后对该 String 应用 rpad 函数。如果它无法转换，则结果为 Undefined 。
任意值	其它值	Undefined .

round(Decimal)

将给定的 Decimal 舍入到最近的 Int。如果 Decimal 与上下两个 Int 值距离相同 (例如 0.5)，Decimal 将向上进位。SQL 版本 2015-10-08 及更高版本支持。

示例：Round(1.2) = 1。

Round(1.5) = 2。

Round(1.7) = 2。

Round(-1.1) = -1。

Round(-1.5) = -2。

参数类型	结果
Int	参数。
Decimal	Decimal 会向下舍入至最近的 Int。
String	Decimal 会向下舍入至最近的 Int。如果字符串无法转换为 Decimal，则结果为 Undefined 。
其它值	Undefined .

rtrim(String)

从提供的 String 中删除所有尾随空白（制表符和空格）。SQL 版本 2015-10-08 及更高版本支持。

示例：

rtrim(" h i ") = "h i"

参数类型	结果
Int	String 的 Int 表示形式。
Decimal	String 的 Decimal 表示形式。
Boolean	布尔值（“true”或“false”）的 String 表示形式。
数组	Array (使用标准转换规则) 的 String 表示形式。
对象	对象 (使用标准转换规则) 的 String 表示形式。
Null	Undefined .
未定义	Undefined

sign(Decimal)

返回给定数字的符号。当参数的符号为正时，将返回 1。当参数的符号为负时，将返回 -1。如果参数为 0，则返回 0。SQL 版本 2015-10-08 及更高版本支持。

示例：

sign(-7) = -1。

sign(0) = 0。

sign(13) = 1。

参数类型	结果
Int	Int，Int 值的符号。
Decimal	Int，Decimal 值的符号。
String	Int，Decimal 值的符号。字符串将转换为 Decimal 值，并返回 Decimal 值的符号。如果 String 无法转换为 Decimal，则结果为 Undefined 。SQL 版本 2015-10-08 及更高版本支持。
其它值	Undefined .

sin(Decimal)

以弧度为单位返回数字的正弦值。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：sin(0) = 0.0

参数类型	结果
Int	Decimal (双精度)，参数的正弦值。
Decimal	Decimal (双精度)，参数的正弦值。
Boolean	Undefined .
String	Decimal (双精度)，参数的正弦值。如果字符串无法转换为 Decimal，则结果为 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .

参数类型	结果
Undefined	Undefined .

sinh(Decimal)

以弧度为单位返回数字的双曲正弦值。在代入函数之前，Decimal 值舍入到双精度。结果是双精度的 Decimal 值。SQL 版本 2015-10-08 及更高版本支持。

示例：sinh(2.3) = 4.936961805545957

参数类型	结果
Int	Decimal (双精度)，参数的双曲正弦值。
Decimal	Decimal (双精度)，参数的双曲正弦值。
Boolean	Undefined .
String	Decimal (双精度)，参数的双曲正弦值。如果字符串无法转换为 Decimal，则结果为 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

sourceip()

检索设备或与之相连的路由器的 IP 地址。如果设备直接连接到互联网，该函数将返回设备的源 IP 地址。如果设备连接到与互联网相连的路由器，该函数将返回路由器的源 IP 地址。由 SQL 版本 2016-03-23 支持。sourceip() 不接受任何参数。

> **Important**
>
> 设备的公共源 IP 地址通常是最后一个网络地址转换（NAT）网关（例如互联网服务提供商的路由器或有线调制解调器）的 IP 地址。

示例：

```
sourceip()="192.158.1.38"
```

```
sourceip()="1.102.103.104"
```

```
sourceip()="2001:db8:ff00::12ab:34cd"
```

SQL 示例：

```
SELECT *, sourceip() as deviceIp FROM 'some/topic'
```

如何在 AWS IoT Core 规则操作中使用 sourceip () 函数的示例：

示例 1

以下示例说明如何在 DynamoDB 操作中调用 () 函数作为替换模板。

```
{
 "topicRulePayload": {
  "sql": "SELECT * AS message FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "dynamoDB": {
     "tableName": "my_ddb_table",
     "hashKeyField": "key",
     "hashKeyValue": "${sourceip()}",
     "rangeKeyField": "timestamp",
     "rangeKeyValue": "${timestamp()}",
     "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamoDB"
    }
   }
  ]
 }
```

```
}
```

示例 2

以下示例说明如何使用替换模板将 sourceip() 函数添加为 MQTT 用户属性。

```
{
 "topicRulePayload": {
  "sql": "SELECT * FROM 'some/topic'",
  "ruleDisabled": false,
  "awsIotSqlVersion": "2016-03-23",
  "actions": [
   {
    "republish": {
     "topic": "${topic()}/republish",
     "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish",
     "headers": {
      "payloadFormatIndicator": "UTF8_DATA",
      "contentType": "rule/contentType",
      "correlationData": "cnVsZSBjb3JyZWxhdGlvbiBkYXRh",
      "userProperties": [
       {
        "key": "ruleKey1",
        "value": "ruleValue1"
       },
       {
        "key": "sourceip",
        "value": "${sourceip()}"
       }
      ]
     }
    }
   }
  ]
 }
}
```

您可以从消息代理和基本收录路径中传递给 AWS IoT Core 规则的消息中检索源 IP 地址。也可以检索 IPv4 和 IPv6 消息的源 IP。源 IP 将如下所示：

IPv6：`yyyy:yyyy:yyyy::yyyy:yyyy`

IPv4：`xxx.xxx.xxx.xxx`

> **Note**
>
> 不会通过重新发布操作传递原始源 IP。

substring(String, Int[, Int])

输入值为 String 后跟一个或两个 Int 值。对于 String 和单个 Int 参数，此函数在输入的 String 中从指定的 Int 索引 (从零开始，包括零) 到 String 结束提取子字符串并返回。对于 String 和两个 Int 参数，此函数在输入的 String 中从第一个 Int 索引参数 (从零开始，包括零) 到第二个 Int 索引参数 (从零开始，包括零) 提取子字符串并返回。索引小于零时将设置为零。大于 String 长度的索引设置为 String 长度。在三个参数的版本中，如果第一个索引大于等于第二个索引，那么结果为空 String。

如果提供的参数不是 (*String*、*Int*) 或 (*String*、*Int*、*Int*)，则系统会对参数进行标准转换，尽量将其转换为正确的类型。如果无法转换类型，函数的结果为 Undefined。SQL 版本 2015-10-08 及更高版本支持。

示例：

`substring("012345", 0)` = "012345"。

`substring("012345", 2)` = "2345"。

`substring("012345", 2.745)` = "2345"。

`substring(123, 2)` = "3"。

`substring("012345", -1)` = "012345"。

`substring(true, 1.2)` = "rue"。

`substring(false, -2.411E247)` = "false"。

`substring("012345", 1, 3)` = "12"。

`substring("012345", -50, 50)` = "012345"。

`substring("012345", 3, 1)` = "".

sql_version()

返回此规则中指定的 SQL 版本。SQL 版本 2015-10-08 及更高版本支持。

例如：

`sql_version()` = "2016-03-23"

sqrt(Decimal)

返回数字的平方根。在代入函数之前，`Decimal` 参数舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：`sqrt(9)` = 3.0。

参数类型	结果
`Int`	参数的平方根。
`Decimal`	参数的平方根。
`Boolean`	`Undefined` 。
`String`	参数的平方根。如果字符串无法转换为 `Decimal`，则结果为 `Undefined` 。
数组	`Undefined` 。
对象	`Undefined` 。
Null	`Undefined` 。
未定义	`Undefined` 。

startswith(String, String)

返回显示第一个字符串参数是否以第二个字符串参数开头的 `Boolean`。如果任一参数为 `Null` 或 `Undefined`，则结果为 `Undefined`。SQL 版本 2015-10-08 及更高版本支持。

例如：

`startswith("ranger","ran")` = true

参数类型 1	参数类型 2	结果
String	String	第一个字符串是否以第二个字符
其它值	其它值	两个参数都使用标准转换规则转串。如果第一个字符串以第二个头，则返回 true。如果任一参数 Undefined ，则结果为 Undef

tan(Decimal)

以弧度为单位返回数字的正切值。在代入函数之前，Decimal 值舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：tan(3) = -0.1425465430742778

参数类型	结果
Int	Decimal (双精度)，参数的正切值。
Decimal	Decimal (双精度)，参数的正切值。
Boolean	Undefined .
String	Decimal (双精度)，参数的正切值。如果字符串无法转换为 Decimal，则结果为 Undefined 。
数组	Undefined .
对象	Undefined .
Null	Undefined .
未定义	Undefined .

tanh(Decimal)

以弧度为单位返回数字的双曲正切值。在代入函数之前，Decimal 值舍入到双精度。SQL 版本 2015-10-08 及更高版本支持。

示例：tanh(2.3) = 0.9800963962661914

参数类型	结果
Int	Decimal (双精度)，参数的双曲正切值。
Decimal	Decimal (双精度)，参数的双曲正切值。
Boolean	Undefined 。
String	Decimal (双精度)，参数的双曲正切值。如果字符串无法转换为 Decimal，则结果为 Undefined 。
数组	Undefined 。
对象	Undefined 。
Null	Undefined 。
未定义	Undefined 。

time_to_epoch(String, String)

使用 time_to_epoch 函数将时间戳字符串转换为 Unix 纪元时间的毫秒数。SQL 版本 2016-03-23 及更高版本支持。要将毫秒转换为格式化的时间戳字符串，请参阅 parse_time(String, Long[, String])。

time_to_epoch 函数采用下列参数：

时间戳

（字符串）要转换为自 Unix 纪元以来毫秒数的时间戳字符串。如果时间戳字符串未指定时区，则函数将使用 UTC 时区。

模式

（字符串）遵循 JDK11 时间格式的日期/时间模式。

示例：

`time_to_epoch("2020-04-03 09:45:18 UTC+01:00", "yyyy-MM-dd HH:mm:ss VV")` = 1585903518000

`time_to_epoch("18 December 2015", "dd MMMM yyyy")` = 1450396800000

`time_to_epoch("2007-12-03 10:15:30.592 America/Los_Angeles", "yyyy-MM-dd HH:mm:ss.SSS z")` = 1196705730592

timestamp()

返回规则引擎观察到的从 1970 年 1 月 1 日星期四协调世界时 (UTC) 00:00:00 开始的当前时间戳（以毫秒为单位）。 AWS IoT SQL 版本 2015-10-08 及更高版本支持。

示例：`timestamp()` = 1481825251155

topic(Decimal)

返回已向其发送触发规则的消息的主题。如果未指定参数，则返回整个主题。`Decimal` 参数用于指定特定主题段，使用 1 指定第一个段。对于主题 `foo/bar/baz`，主题 (1) 将返回 `foo`，主题 (2) 将返回 `bar`，以此类推。SQL 版本 2015-10-08 及更高版本支持。

示例：

`topic()` = "things/myThings/thingOne"

`topic(1)` = "things"

在使用基本提取功能时，主题的初始前缀 (`$aws/rules/`*`rule-name`*) 对 topic() 函数不可用。例如，给定以下主题：

`$aws/rules/BuildingManager/Buildings/Building5/Floor2/Room201/Lights`

`topic()` = "Buildings/Building5/Floor2/Room201/Lights"

`topic(3)` = "Floor2"

traceid()

返回 MQTT 消息的跟踪 ID (UUID)，如果未通过 MQTT 发送消息，则返回 `Undefined`。SQL 版本 2015-10-08 及更高版本支持。

例如：

`traceid()` = "12345678-1234-1234-1234-123456789012"

transform(String, Object, Array)

返回对象数组，其中包含 `Object` 参数对 `Array` 参数的指定转换结果。

SQL 版本 2016-03-23 及更高版本支持。

String

要使用的转换模式。有关支持的转换模式以及这些模式如何利用 `Object` 和 `Array` 参数创建 `Result` 的信息，请参阅下表。

对象

一个对象，其中包含要应用于 `Array` 的各个元素的属性。

数组

对象的数组，其中将应用 `Object` 的属性。

此数组中的每个对象都对应于函数响应中的一个对象。函数响应中的每个对象都包含原始对象中存在的属性以及 `Object` 提供的属性，这些属性由 `String` 中指定的转换模式决定。

String 参数	**Object** 参数	**Array** 参数	结果
`enrichArray`	对象	对象数组	一个对象的数组，其中每个对象都包含`Array` 参数中元素的属性和 `Object` 参数的属性。
其它任何值	任意值	任意值	未定义

> **Note**
>
> 此函数返回的数组限制为 128 KiB。

变换函数示例 1

此示例演示如何使用 transform() 函数从数据对象和数组中生成单个对象数组。

在此示例中，将以下消息将发布到 MQTT 主题 A/B。

```
{
    "attributes": {
        "data1": 1,
        "data2": 2
    },
    "values": [
        {
            "a": 3
        },
        {
            "b": 4
        },
        {
            "c": 5
        }
    ]
}
```

此用于主题规则操作的 SQL 语句使用 String 值为 enrichArray 的 transform() 函数。在本例中，Object 是来自消息负载的 attributes 属性，而 Array 是 values 数组，其中包含三个对象。

```
select value transform("enrichArray", attributes, values) from 'A/B'
```

在收到消息负载后，SQL 语句将计算以下响应。

```
[
  {
    "a": 3,
    "data1": 1,
    "data2": 2
  },
  {
    "b": 4,
    "data1": 1,
```

```
    "data2": 2
  },
  {
    "c": 5,
    "data1": 1,
    "data2": 2
  }
]
```

变换函数示例 2

此示例演示 transform() 函数如何使用文本值来包含和重命名消息负载中的单个属性。

在此示例中，将以下消息将发布到 MQTT 主题 A/B。此消息与 the section called “变换函数示例 1” 中使用的消息相同。

```
{
    "attributes": {
        "data1": 1,
        "data2": 2
    },
    "values": [
        {
            "a": 3
        },
        {
            "b": 4
        },
        {
            "c": 5
        }
    ]
}
```

此用于主题规则操作的 SQL 语句使用 `String` 值为 `enrichArray` 的 transform() 函数。transform() 函数中的 `Object` 在消息负载中具有一个名为 `key` 的单一属性，该属性的值为 `attributes.data1`，而 `Array` 是 `values` 数组，其中包含与上一个示例中使用的三个相同对象。

```
select value transform("enrichArray", {"key": attributes.data1}, values) from 'A/B'
```

收到消息负载后，此 SQL 语句将计算为以下响应。请注意 `data1` 属性在 `key` 响应中是如何命名的。

```
[
  {
    "a": 3,
    "key": 1
  },
  {
    "b": 4,
    "key": 1
  },
  {
    "c": 5,
    "key": 1
  }
]
```

变换函数示例 3

此示例演示如何使用 transform() 函数可在嵌套 SELECT 子句中选择多个属性并创建新对象以供后续处理。

在此示例中，将以下消息将发布到 MQTT 主题 A/B。

```
{
  "data1": "example",
  "data2": {
    "a": "first attribute",
    "b": "second attribute",
    "c": [
      {
        "x": {
          "someInt": 5,
          "someString": "hello"
        },
        "y": true
      },
      {
        "x": {
          "someInt": 10,
          "someString": "world"
        },
        "y": false
      }
    ]
```

```
  }
}
```

此转换函数的 Object 是 SELECT 语句返回的对象，其中包含消息的 data2 对象的 a 和 b 元素。Array 参数包括两个来自原始消息中 data2.c 数组的对象。

```
select value transform('enrichArray', (select a, b from data2), (select value c from
 data2)) from 'A/B'
```

借助前面的消息，SQL 语句将计算以下响应。

```
[
  {
    "x": {
      "someInt": 5,
      "someString": "hello"
    },
    "y": true,
    "a": "first attribute",
    "b": "second attribute"
  },
  {
    "x": {
      "someInt": 10,
      "someString": "world"
    },
    "y": false,
    "a": "first attribute",
    "b": "second attribute"
  }
]
```

此响应中返回的数组可以用于支持 batchMode 的主题规则操作。

trim(String)

从提供的 String 中删除所有前导空白和尾随空白。SQL 版本 2015-10-08 及更高版本支持。

例如：

Trim(" hi ") = "hi"

参数类型	结果
Int	Int 在删除所有前导和尾随空白之后的 String 表示形式。
Decimal	Decimal 在删除所有前导和尾随空白之后的 String 表示形式。
Boolean	Boolean（“true”或“false”）在删除所有前导和尾随空白之后的 String 表示形式。
String	删除所有前导和尾随空白之后的 String。
数组	Array 使用标准转换规则进行转换后的 String 表示形式。
对象	对象使用标准转换规则进行转换后的 String 表示形式。
Null	Undefined .
未定义	Undefined .

trunc(Decimal, Int)

按照第二个参数指定的 Decimal 位数截断第一个参数。如果第二个参数小于零，则它会设置为零。如果第二个参数大于 34，则它会设置为 34。将从结果中删除结尾的零。SQL 版本 2015-10-08 及更高版本支持。

示例：

trunc(2.3, 0) = 2。

trunc(2.3123, 2) = 2.31。

trunc(2.888, 2) = 2.88。

trunc(2.00, 5) = 2。

参数类型 1	参数类型 2	结果
Int	Int	源值。
Int/Decimal	Int/Decimal	第一个参数被截断到由第二个参长度。第二个参数如果不是 Int入至最近的 Int。
Int/Decimal/String	Int/Decimal	第一个参数被截断到由第二个参的长度。第二个参数如果不是 I向下舍入至最近的 Int。Strin Decimal 值。如果字符串转换失为 Undefined 。
其它值		Undefined .

upper(String)

返回给定 String 的大写版本。非 String 参数将使用标准转换规则转换为 String。SQL 版本 2015-10-08 及更高版本支持。

示例：

upper("hello") = "HELLO"

upper(["hello"]) = "[\"HELLO\"]"

文本

您可以直接在规则 SQL 的 SELECT 和 WHERE 子句中指定文本对象，该对象可用于传递信息。

> **Note**
>
> 文本只能在使用 SQL 版本 2016-03-23 或更高版本时使用。

使用 JSON 对象语法 (键值对、逗号分隔、键为字符串/值为 JSON 值、括在大括号 {} 中)。例如：

传入负载已发布至主题 topic/subtopic：{"lat_long": [47.606,-122.332]}

SQL 语句：`SELECT {'latitude': get(lat_long, 0),'longitude':get(lat_long, 1)} as lat_long FROM 'topic/subtopic'`

生成的传出负载为：`{"lat_long":{"latitude":47.606,"longitude":-122.332}}`。

您也可以在规则 SQL 的 SELECT 和 WHERE 子句中直接指定数组，用于分组信息。使用 JSON 语法 (在方括号 [] 中用逗号分隔项目来创建数组文本)。例如：

传入负载已发布至主题 `topic/subtopic`：`{"lat": 47.696, "long": -122.332}`

SQL 语句：`SELECT [lat,long] as lat_long FROM 'topic/subtopic'`

生成的传出负载为：`{"lat_long": [47.606,-122.332]}`。

Case 语句

Case 语句可以用于执行分支，就像switch 语句一样。

语法：

```
CASE v WHEN t[1] THEN r[1]
  WHEN t[2] THEN r[2] ...
  WHEN t[n] THEN r[n]
  ELSE r[e] END
```

评估表达式 *v*，并与每个 WHEN 子句的 *t[i]* 值进行相等匹配。如果找到匹配，相应的 *r[i]* 表达式会成为 CASE 语句的结果。按顺序评估 WHEN 子句，这样，如果有多个匹配子句，第一个匹配子句的结果将成为 CASE 语句的结果。如果没有匹配，ELSE 子句的 *r[e]* 就是结果。如果没有匹配项且没有 ELSE 子句，则结果为 Undefined。

CASE 语句至少需要一个 WHEN 子句。ELSE 子句是可选的。

例如：

传入负载已发布至主题 `topic/subtopic`：

```
{
    "color":"yellow"
}
```

SQL 语句

```
SELECT CASE color
        WHEN 'green' THEN 'go'
        WHEN 'yellow' THEN 'caution'
        WHEN 'red' THEN 'stop'
        ELSE 'you are not at a stop light' END as instructions
    FROM 'topic/subtopic'
```

生成的传出负载为：

```
{
    "instructions":"caution"
}
```

> **Note**
>
> 如果 *v* 是 Undefined，则 Case 语句的结果为 Undefined。

JSON 扩展

您可以使用 ANSI SQL 语法的以下扩展，以便于使用嵌套 JSON 对象。

"." 运算符

此运算符访问嵌入式 JSON 对象中的成员和函数与 ANSI SQL 和. JavaScript 例如：

```
SELECT foo.bar AS bar.baz FROM 'topic/subtopic'
```

从以下发送至 topic/subtopic 主题的消息负载中选择 foo 对象中的 bar 属性。

```
{
  "foo": {
    "bar": "RED",
    "bar1": "GREEN",
    "bar2": "BLUE"
  }
}
```

如果 JSON 属性名称包含连字符或数字字符，则“点”符号将不起作用。相反，您必须使用 get 函数来提取属性的值。

在此示例中，以下消息将发送至 iot/rules 主题。

```
{
  "mydata": {
    "item2": {
      "0": {
        "my-key": "myValue"
      }
    }
  }
}
```

通常情况下，my-key 的值将被标识为在此查询中。

```
SELECT * from iot/rules WHERE mydata.item2.0.my-key= "myValue"
```

但是，由于属性名称 my-key 包含连字符，而 item2 包含一个数字字符，因此 get 函数必须按以下查询所示的方法使用。

```
SELECT * from 'iot/rules' WHERE get(get(get(mydata,"item2"),"0"),"my-key") = "myValue"
```

* 运算符

该运算符与 ANSI SQL 中的 * 通配符的运作方式相同。该运算符仅用于 SELECT 子句，并会创建包含消息数据的全新 JSON 对象。如果消息负载不是 JSON 格式，* 将以原始字节形式返回整个消息负载。例如：

```
SELECT * FROM 'topic/subtopic'
```

将函数应用到属性值

下面显示了一个可能由设备发布的 JSON 负载示例：

```
{
    "deviceid" : "iot123",
    "temp" : 54.98,
```

```
    "humidity" : 32.43,
    "coords" : {
        "latitude" : 47.615694,
        "longitude" : -122.3359976
    }
}
```

下面的示例将函数应用到 JSON 负载中的一个属性值：

```
SELECT temp, md5(deviceid) AS hashed_id FROM topic/#
```

此查询的结果为以下 JSON 对象：

```
{
   "temp": 54.98,
   "hashed_id": "e37f81fb397e595c4aeb5645b8cbbbd1"
}
```

替换模板

您可以使用替换模板来增加触发规则并 AWS IoT 执行操作时返回的 JSON 数据。替换模板的语法是${表达式}，其中表达式可以是 SELECT 子句、WHERE 子句和 AWS IoT 中支持的任何表达式AWS IoT 规则动作。此表达式可插入到规则的操作字段中，使您可以动态配置操作。在生效时，此特征替换操作中的一段信息。这包括函数、运算符和原始消息负载中存在的信息。

> **Important**
>
> 因为替换模板中的表达式与“SELECT ...”语句分开评估，所以，您无法引用使用 AS 子句创建的别名。您只能引用原始有效负载中存在的信息、函数和运算符。

有关支持的表达式的更多信息，请参阅AWS IoT SQL 参考。

以下规则操作支持替换模板。每个操作都支持不同的可替换字段。

- Apache Kafka
- CloudWatch 警报
- CloudWatch 日志

- CloudWatch 指标
- DynamoDB
- DynamoDBv2
- Elasticsearch
- HTTP
- IoT Analytics
- AWS IoT Events
- AWS IoT SiteWise
- Kinesis Data Streams
- Firehose
- Lambda
- 位置
- OpenSearch
- Republish
- S3
- SNS
- SQS
- Step Functions
- Timestream

替换模板显示在规则内的操作参数中：

```
{
    "sql": "SELECT *, timestamp() AS timestamp FROM 'my/iot/topic'",
    "ruleDisabled": false,
    "actions": [{
        "republish": {
            "topic": "${topic()}/republish",
            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
        }
    }]
}
```

如果该规则由发布到 my/iot/topic 的以下 JSON 触发：

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    }
}
```

然后，此规则将以下 JSON 发布到my/iot/topic/republish，其 AWS IoT 替换为：${topic()}/republish

```
{
    "deviceid": "iot123",
    "temp": 54.98,
    "humidity": 32.43,
    "coords": {
        "latitude": 47.615694,
        "longitude": -122.3359976
    },
    "timestamp": 1579637878451
}
```

嵌套对象查询

您可以使用嵌套 SELECT 子句来查询数组和内部 JSON 对象中的属性。SQL 版本 2016-03-23 及更高版本支持。

请考虑以下 MQTT 消息：

```
{
    "e": [
        { "n": "temperature", "u": "Cel", "t": 1234, "v": 22.5 },
        { "n": "light", "u": "lm", "t": 1235, "v": 135 },
        { "n": "acidity", "u": "pH", "t": 1235, "v": 7 }
    ]
}
```

Example

您可以使用以下规则将值转换为新数组。

```
SELECT (SELECT VALUE n FROM e) as sensors FROM 'my/topic'
```

该规则生成以下输出。

```
{
    "sensors": [
        "temperature",
        "light",
        "acidity"
    ]
}
```

Example

通过使用相同的 MQTT 消息，您还可以使用以下规则查询嵌套对象中的特定值。

```
SELECT (SELECT v FROM e WHERE n = 'temperature') as temperature FROM 'my/topic'
```

该规则生成以下输出。

```
{
    "temperature": [
        {
            "v": 22.5
        }
    ]
}
```

Example

您也可以使用更复杂的规则来平展输出。

```
SELECT get((SELECT v FROM e WHERE n = 'temperature'), 0).v as temperature FROM 'topic'
```

该规则生成以下输出。

```
{
    "temperature": 22.5
}
```

使用二进制负载

当将消息负载作为原始二进制数据（而不是 JSON 对象）进行处理时，可以使用 * 运算符在 SELECT 子句中对其进行引用。

本主题内容：

二进制负载示例

当您使用 * 将消息负载作为原始二进制数据进行引用时，您可以向规则添加数据。如果您有空的或 JSON 有效负载，生成的有效负载可以使用规则添加数据。下面显示了支持 SELECT 子句的示例。

- 对于二进制有效负载，您可以将以下 SELECT 子句仅与 * 一起使用。
  - `SELECT * FROM 'topic/subtopic'`
  - `SELECT * FROM 'topic/subtopic' WHERE timestamp() % 12 = 0`
- 您还可以添加数据并使用以下 SELECT 子句。
  - `SELECT *, principal() as principal, timestamp() as time FROM 'topic/subtopic'`
  - `SELECT encode(*, 'base64') AS data, timestamp() AS ts FROM 'topic/subtopic'`
- 您还可以使用带二进制负载的 SELECT 子句。
  - 以下是指在 WHERE 子句中的 `device_type`。

    ```
    SELECT * FROM 'topic/subtopic' WHERE device_type = 'thermostat'
    ```

  - 还支持以下内容。

    ```
    {
     "sql": "SELECT * FROM 'topic/subtopic'",
    ```

```
 "actions": [
  {
   "republish": {
    "topic": "device/${device_id}"
   }
  }
 ]
}
```

以下规则操作不支持二进制负载，因此您必须对它们进行解码。

- 一些规则操作不支持二进制负载输入（例如，Lambda 操作），您必须解码二进制负载。如果 Lambda 规则操作是 base64 编码并在 JSON 负载中，则可以接收二进制数据。为此，您可以将规则更改如下：

```
SELECT encode(*, 'base64') AS data FROM 'my_topic'
```

- SQL 语句不支持字符串作为输入。要将字符串输入转换为 JSON，您可以运行以下命令。

```
SELECT decode(encode(*, 'base64'), 'base64') AS payload FROM 'topic'
```

解码 protobuf 消息有效负载

协议缓冲区 (protobuf) 是一种开源数据格式，用于以紧凑的二进制形式序列化结构化数据。它用于通过网络传输数据或将数据存储在文件中。Protobuf 允许您以比其他消息格式更快的速度以较小的数据包大小发送数据。 AWS IoT Core 规则通过提供解码（值，decodingScheme）SQL 函数来支持 protobuf，该函数允许您将由 protobuf 编码的消息负载解码为 JSON 格式并将其路由到下游服务。本节详细介绍了在 “规则” 中配置 protobuf 解码的 step-by-step 过程。 AWS IoT Core

本节内容：

- 先决条件
- 创建描述符文件
- 将描述符文件上传到 S3 存储桶
- 在规则中配置 protobuf 解码
- 限制
- 最佳实践

先决条件

- 对协议缓冲区 (protobuf) 有基本的了解
- 定义消息类型和相关依赖项的 `.proto` 文件
- 在您的系统上安装 Protobuf 编译器 (protoc)

创建描述符文件

如果您已有描述符文件，可以跳过此步骤。描述符文件 (`.desc`) 是 `.proto` 文件的编译版本，它是一个文本文件，用于定义 protobuf 序列化中使用的数据结构和消息类型。要生成描述符文件，必须定义一个 `.proto` 文件，并使用 protoc 编译器对其进行编译。

1. 创建定义消息类型的 `.proto` 文件。示例 `.proto` 文件可能如下所示：

```
syntax = "proto3";

message Person {
  optional string name = 1;
  optional int32 id = 2;
  optional string email = 3;
}
```

在此示例 `.proto` 文件中，使用 proto3 语法并定义消息类型 `Person`。`Person` 消息定义指定三个字段（名称、ID 和电子邮件）。有关 `.proto` 文件消息格式的更多信息，请参阅语言指南 (proto3)。

2. 使用 protoc 编译器编译 `.proto` 文件并生成一个描述符文件。创建描述符 (`.desc`) 文件的示例命令如下所示：

```
protoc --descriptor_set_out=<FILENAME>.desc \
    --proto_path=<PATH_TO_IMPORTS_DIRECTORY> \
    --include_imports \
    <PROTO_FILENAME>.proto
```

此示例命令生成一个描述符文件`<FILENAME>.desc`， AWS IoT Core 规则可以使用该文件来解码符合中定义的数据结构的 protobuf 有效负载。`<PROTO_FILENAME>.proto`

- `--descriptor_set_out`

  指定应生成的描述符文件 (`<FILENAME>.desc`) 的名称。

- `--proto_path`

  指定正在编译的文件所引用的任何导入的 `.proto` 文件的位置。如果您有多个位于不同位置的已导入 `.proto` 文件，可以多次指定此标志。

- `--include_imports`

  指定还应编译任何已导入的 `.proto` 文件，并将其包含在 `<FILENAME>.desc` 描述符文件中。

- `<PROTO_FILENAME>.proto`

  指定要编译的 `.proto` 文件的名称。

有关 protoc 参考的更多信息，请参阅 API 参考。

将描述符文件上传到 S3 存储桶

创建描述符文件后，使用 AWS API `<FILENAME>.desc`、S AWS DK 或，`<FILENAME>.desc`将描述符文件上传到 Amazon S3 存储桶。 AWS Management Console

重要注意事项

- 请务必将描述符文件上传到您的 Amazon S3 存储桶，与您打算配置规则的 AWS 区域 位置相同。AWS 账户
- 请确保您授予`FileDescriptorSet`从 S3 读取的 AWS IoT Core 权限。如果您的 S3 存储桶已禁用服务器端加密 (SSE)，或者 S3 存储桶已使用 Amazon S3 托管密钥 (SSE-S3) 加密，则不需要进行额外的策略配置。这可以通过示例存储桶策略来实现：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "Statement1",
   "Effect": "Allow",
   "Principal": {
    "Service": "iot.amazonaws.com"
   },
   "Action": "s3:Get*",
                     "Resource": "arn:aws:s3:::<BUCKET NAME>/<FILENAME>.desc"
  }
 ]
```

```
}
```

- 如果您的 S3 存储桶使用 AWS Key Management Service 密钥 (SSE-KMS) 进行加密，请确保授予在访问 S3 存储桶时使用该密钥的 AWS IoT Core 权限。这可以通过将此语句添加到密钥策略中来实现。

```
{
 "Sid": "Statement1",
 "Effect": "Allow",
 "Principal": {
  "Service": "iot.amazonaws.com"
 },
 "Action": [
  "kms:Decrypt",
  "kms:GenerateDataKey*",
  "kms:DescribeKey"
 ],
        "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"

}
```

在规则中配置 protobuf 解码

将描述符文件上传到 Amazon S3 桶后，配置一条规则，该规则可以使用 decode(value, decodingScheme) SQL 函数解码您的 protobuf 消息有效负载格式。详细的函数签名和示例可以在《AWS IoT SQL 参考》的 decode(value, decodingScheme) SQL 函数中找到。

下面是使用 decode(value, decodingScheme) 函数的 SQL 表达式示例：

```
SELECT VALUE decode(*, 'proto', '<BUCKET NAME>', '<FILENAME>.desc', '<PROTO_FILENAME>',
 '<PROTO_MESSAGE_TYPE>') FROM '<MY_TOPIC>'
```

在此示例表达式中：

- 您可以使用 decode(value, decodingScheme) SQL 函数来解码 * 引用的二进制消息有效负载。这可以是 protobuf 编码的二进制有效负载，也可以是表示 base64 编码的 protobuf 有效负载的 JSON 字符串。
- 提供的消息有效负载使用中 PROTO_FILENAME.proto 定义的 Person 消息类型进行编码。

- 名为 BUCKET_NAME 的 Amazon S3 桶中包含从 PROTO_FILENAME.proto 生成的 FILENAME.desc。

完成配置后，向发布一条 AWS IoT Core 有关订阅该规则的主题的消息。

限制

AWS IoT Core 规则支持 protobuf，但有以下限制：

- 不支持在替换模板中解码 protobuf 消息有效负载。
- 解码 protobuf 消息有效负载时，可以在单个 SQL 表达式中使用解码 SQL 函数多达 2 次。
- 最大入站有效负载大小为 128 KiB（1KiB = 1024 字节），最大出站有效负载大小为 128 KiB，存储在 Amazon S3 存储桶中的 FileDescriptorSet 对象的最大大小为 32 KiB。
- 不支持使用 SSE-C 加密进行加密的 Amazon S3 存储桶。

最佳实践

以下是一些最佳实践和故障排查提示。

- 在 Amazon S3 存储桶中备份您的原型文件。

  备份您的原型文件是一种很好的做法，以防出现问题。例如，如果您在运行 protoc 时错误地修改了没有备份的原型文件，这可能会导致您的生产堆栈出现问题。有多种方法可以在 Amazon S3 存储桶中备份您的文件。例如，您可以在 S3 存储桶中使用版本控制。有关如何备份 Amazon S3 存储桶中的文件的更多信息，请参阅 Amazon S3 开发人员指南。
- 配置 AWS IoT 日志以查看日志条目。

  最好配置 AWS IoT 日志记录，这样你就可以在中查看账户的 AWS IoT 日志 CloudWatch。当规则的 SQL 查询调用外部函数时， AWS IoT Core Rules 会生成一个带为eventType的日志条目FunctionExecution，其中包含可帮助您排除故障原因字段。可能的错误包括找不到 Amazon S3 对象，或者 protobuf 文件描述符无效。有关如何配置 AWS IoT 日志记录和查看日志条目的更多信息，请参阅配置 AWS IoT 日志记录和规则引擎日志条目。
- 使用新的对象键更新 FileDescriptorSet，并更新规则中的对象键。

  您可以通过将更新后的描述符文件上传到 Amazon S3 桶来更新 FileDescriptorSet。您对 FileDescriptorSet 的更新最多需要 15 分钟能够反映出来。为了避免这一延迟，使用新的对象键上传更新后的 FileDescriptorSet，然后更新规则中的此对象密钥是一种很好的做法。

SQL 版本

AWS IoT 规则引擎使用类似 SQL 的语法从 MQTT 消息中选择数据。SQL 语句基于 SQL 版本进行解释，该版本由描述此规则的 JSON 文档中的 `awsIotSqlVersion` 属性指定。有关 JSON 规则文档结构的更多信息，请参阅创建规则。该`awsIotSqlVersion`属性允许您指定要使用的 AWS IoT SQL 规则引擎版本。当部署新版本时，您可以继续使用早期版本或更改规则以使用新版本。您当前的规则将继续使用创建时所用的版本。

以下 JSON 示例介绍了如何使用 `awsIotSqlVersion` 属性指定 SQL 版本：

```
{
    "sql": "expression",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [{
        "republish": {
            "topic": "my-mqtt-topic",
            "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"
        }
    }]
}
```

AWS IoT 目前支持以下 SQL 版本：

- `2016-03-23` – 2016 年 3 月 23 日构建的 SQL 版本（建议）。
- `2015-10-08` – 2015 年 10 月 8 日构建的 SQL 原始版本。
- `beta` – 最新的 SQL 测试版本。此版本可能会给您的规则带来突破性变化。

2016-03-23 SQL 规则引擎版本中的新增功能

- 针对选择嵌套 JSON 对象的修复程序。
- 针对阵列查询的修复程序。
- 对象内查询支持。有关更多信息，请参阅 嵌套对象查询。
- 支持将阵列作为顶级对象输出。
- 添加可应用于 JSON 和非 JSON 格式数据的 `encode(value, encodingScheme)` 函数。有关更多信息，请参阅编码函数。

将 **Array** 作为顶级对象输出

此特征允许规则将阵列作为顶级对象返回。例如，给定了以下 MQTT 消息：

```
{
    "a": {"b":"c"},
    "arr":[1,2,3,4]
}
```

以及以下规则：

```
SELECT VALUE arr FROM 'topic'
```

该规则生成以下输出。

```
[1,2,3,4]
```

AWS IoT Device Shadow 服务

AWS IoT Device Shadow 服务为 AWS IoT 事物对象添加阴影。无论设备是否已连接，Shadows 都可以将设备的状态提供给 AWS IoT 应用程序和其他服务。 AWS IoT 事物对象可以有多个命名的阴影，这样您的物联网解决方案就有更多选项可以将您的设备连接到其他应用程序和服务。

AWS IoT 事物对象在明确创建之前没有任何阴影。可以使用 AWS IoT 控制台创建、更新和删除阴影。设备、其它 Web 客户端和服务可以使用 MQTT 和预留的 MQTT 主题、使用 Device Shadow REST API 的 HTTP 以及适用于 AWS IoT 的 AWS CLI 创建、更新和删除影子。由于影子存储 AWS 在云中，因此无论设备是否已连接，它们都可以从应用程序和其他云服务中收集和报告设备状态数据。

使用影子

影子为设备、应用程序和其它云服务提供可靠的数据存储以共享数据。它们允许设备、应用程序和其它云服务连接和断开连接，而不会丢失设备的状态。

当设备、应用程序和其他云服务连接到时 AWS IoT，它们可以通过设备的阴影访问和控制设备的当前状态。例如，应用程序可以通过更新影子来请求更改设备的状态。 AWS IoT 发布一条消息，指明对设备的更改。设备接收该消息，更新其状态以保持匹配，然后发布一条消息以指示其更新状态。Device Shadow 服务在相应的影子中反映该更新状态。应用程序可以订阅影子的更新，也可以查询影子以获取其当前状态。

当设备离线时，应用程序仍然可以 AWS IoT 与设备的影子进行通信。在设备重新连接时，它接收其影子的当前状态，以便它可以更新其状态以与其影子的状态匹配，然后发布一条消息以指示其更新状态。同样，如果应用程序脱机并且设备状态在其脱机时发生变化，则设备将影子保持更新状态，以便应用程序可以在设备重新连接时查询影子以获取其当前状态。

如果设备经常处于离线状态，但您希望将其配置为在重新连接后接收增量消息，则可以使用持久性会话特征。有关持久性会话过期期限的更多信息，请参阅持久性会话过期期限。

选择使用命名或未命名的影子

设备影子服务支持命名的影子和未命名或经典的影子。事物对象可以具有多个命名的影子，并且最多可以具有一个未命名的影子。事物对象也可以有一个预留命名影子，它的运行方式与命名影子类似，只是您无法更新其名称。有关更多信息，请参阅预留命名影子。

事物对象可以同时具有命名和未命名的影子；不过，用于访问每种影子的 API 略有不同，因此，决定哪种类型的影子最适合您的解决方案并仅使用该类型可能更高效。有关用于访问影子的 API 的更多信息，请参阅影子主题。

通过使用命名的影子，您可以为事物对象的状态创建不同的视图。例如，您可以将一个具有很多属性的事物对象划分为具有逻辑属性组的影子，每个影子由其影子名称标识。您也可以将属性分组到不同的影子，并使用策略控制访问以限制对属性的访问。有关与设备影子配合使用的策略的更多信息，请参阅AWS IoT 的操作、资源和条件键以及 AWS IoT Core 策略。

经典的未命名影子比命名的影子更简单，但在某些程度上比命名的影子更受限制。每个 AWS IoT 事物对象只能有一个未命名的阴影。如果预计您的 IoT 解决方案对影子数据的需求有限，您可能希望以这种方式开始使用影子。不过，如果您认为将来可能要添加其它影子，请考虑从一开始就使用命名的影子。

机群索引支持不同的未命名影子和命名影子。有关更多信息，请参阅管理实例集索引。

访问影子

每个影子具有预留的 MQTT 主题和 HTTP URL，该 URL 支持对影子执行 `get`、`update` 和 `delete` 操作。

影子使用 JSON 影子文档以存储和检索数据。影子的文档包含一个状态属性，以描述设备状态的以下方面：

- `desired`

  应用程序更新 `desired` 对象以指定设备属性的所需状态。

- `reported`

  设备在 `reported` 对象中报告其当前状态。

- `delta`

  AWS IoT 报告`delta`对象中所需状态和报告状态之间的差异。

存储在影子中的数据是由更新操作的消息正文的状态属性确定的。后续的更新操作可以修改现有数据对象的值，也可以在影子的状态对象中添加和删除键和其它元素。有关访问影子的更多信息，请参阅 在设备中使用影子 和 在应用程序和服务中使用影子。

> **Important**
>
> 发出更新请求的权限应限制为受信任的应用程序和设备。这可防止意外更改影子的状态属性；否则，使用影子的设备和应用程序应设计为需要更改状态属性中的键。

在设备、应用程序和其它云服务中使用影子

要在设备、应用程序和其它云服务中使用影子，需要在它们之间保持一致性和协调性。 AWS IoT Device Shadow 服务存储影子状态，在影子状态发生变化时发送消息，并对更改其状态的消息做出响应。您的 IoT 解决方案中的设备、应用程序和其它云服务必须管理其状态，并使其与设备影子的状态保持一致。

影子状态数据是动态的，有权访问影子的设备、应用程序和其它云服务可以对其进行更改。因此，请务必考虑每个设备、应用程序和其它云服务如何与影子进行交互。例如：

- 在将状态数据传送到影子时，设备 应仅写入到影子状态的 `reported` 属性中。
- 在通过影子将状态更改请求传送到设备时，应用程序和其它云服务 应仅写入到 `desired` 属性中。

> **Important**
>
> 影子数据对象中包含的数据与其它影子和其他事物对象属性的数据（例如事物的属性以及事物对象的设备可能发布的 MQTT 消息内容）无关。不过，如果需要，设备可以在不同的 MQTT 主题和影子中报告相同的数据。
> 支持多个影子的设备必须将它在不同影子中报告的数据保持一致。

消息顺序

无法保证来自该 AWS IoT 服务的消息会按任何特定的顺序到达设备。以下方案显示了此时发生的情况。

初始状态文档：

```
{
  "state": {
    "reported": {
      "color": "blue"
    }
  },
  "version": 9,
  "timestamp": 123456776
}
```

更新 1：

```
{
  "state": {
    "desired": {
      "color": "RED"
    }
  },
  "version": 10,
  "timestamp": 123456777
}
```

更新 2：

```
{
  "state": {
    "desired": {
      "color": "GREEN"
    }
  },
  "version": 11,
  "timestamp": 123456778
}
```

最终状态文档：

```
{
  "state": {
    "reported": {
      "color": "GREEN"
    }
  },
  "version": 12,
  "timestamp": 123456779
}
```

这将产生两个增量消息：

```
{
  "state": {
    "color": "RED"
  },
  "version": 11,
```

```
  "timestamp": 123456778
}
```

```
{
  "state": {
    "color": "GREEN"
  },
  "version": 12,
  "timestamp": 123456779
}
```

设备可能不会按次序收到这些消息。由于这些消息中的状态是累计的，设备可以安全地弃用版本号比正在追踪的版本号更早的所有消息。如果设备在接收版本 11 的增量之前收到版本 12 的增量，则可以安全地弃用版本 11 的消息。

修剪影子消息

要降低发送到您的设备的影子消息的大小，请定义一项规则，以仅选择设备所需的字段然后将消息重新发布到设备正在侦听的 MQTT 主题。

规则应在 JSON 中指定，且应与以下内容类似：

```
{
  "sql": "SELECT state, version FROM '$aws/things/+/shadow/update/delta'",
  "ruleDisabled": false,
  "actions": [
    {
      "republish": {
        "topic": "${topic(3)}/delta",
        "roleArn": "arn:aws:iam:123456789012:role/my-iot-role"
      }
    }
  ]
}
```

SELECT 语句用于确定将消息中的哪些字段重新发布到指定的主题。“+”通配符用于匹配所有影子名称。该规则指定，应将所有匹配的消息重新发布到指定的主题。在此情况下，可以使用 "topic()" 函数指定将消息重新发布到哪个主题。topic(3) 用于评估原始主题中的事物名称。有关创建规则的更多信息，请参阅的规则 AWS IoT。

在设备中使用影子

本节介绍使用 MQTT 消息与影子进行设备通信，MQTT 消息是设备与 Device Shadow 服务通信的 AWS IoT 首选方法。

影子通信使用 MQTT 的发布/订阅通信模型以模拟请求/响应模型。每个影子操作包含请求主题、成功的响应主题 (`accepted`) 和错误响应主题 (`rejected`)。

如果您希望应用程序和服务能够确定是否连接了设备，请参阅检测是否连接了设备。

> **Important**
>
> 由于 MQTT 使用发布/订阅通信模型，您必须先订阅响应主题，然后再发布请求主题。否则，您不会收到您发布请求的响应。
> 如果您使用 AWS IoT Device SDK 来调用 Device Shadow 服务 API，这将为您进行处理。

本节中的示例使用主题的缩写形式，其中*`ShadowTopicPrefix`*可以指已命名或未命名的影子，如下表所述。

影子可以是命名或未命名（经典）的。每个影子使用的主题仅在主题前缀上有所不同。下表显示每种影子类型使用的主题前缀。

***`ShadowTopicPrefix`* 价值**	影子类型
`$aws/things/` *`thingName`* `/shadow`	未命名的（经典）影子
`$aws/things/` *`thingName`* `/shadow/name/` *`shadowName`*	命名的影子

> **Important**
>
> 确保应用程序或服务使用的影子与设备中的相应实施保持一致，并且该实施支持使用这些影子。例如，考虑如何创建、更新和删除影子。还要考虑如何在设备以及通过影子访问设备的应用程序或服务中处理更新。您的设计应明确指定如何更新和报告设备的状态，以及应用程序和服务如何与设备及其影子进行交互。

要创建完整的主题，请为要表示的影子类型选择 *ShadowTopicPrefix*，将 *thingName* 和 *shadowName*（如果适用）替换为相应的值，然后在其后面附加主题存根，如下表中所示。请记住，主题区分大小写。

有关影子的保留主题的更多信息，请参阅 影子主题。

在首次连接设备时初始化设备 AWS IoT

设备注册后 AWS IoT，应订阅这些 MQTT 消息以获取其支持的阴影。

主题	含义	在收到该主题时设备应执行的操作
ShadowTopicPrefix / delete/accepted	delete请求被接受并 AWS IoT 删除了影子。	为满足删除的影子要求而需要执行的操作，例如，停止发布更新。
ShadowTopicPrefix / delete/rejected	delete请求被拒绝 AWS IoT ，影子未被删除。消息正文包含错误信息。	响应消息正文中的错误消息。
ShadowTopicPrefix / get/accepted	get请求已被接受 AWS IoT，消息正文包含当前的影子文档。	处理消息正文中的状态文档所需的操作。
ShadowTopicPrefix / get/rejected	get请求被拒绝 AWS IoT，消息正文包含错误信息。	响应消息正文中的错误消息。
ShadowTopicPrefix / update/accepted	update请求已被接受 AWS IoT，消息正文包含当前的影子文档。	确认消息正文中的更新数据与设备状态匹配。
ShadowTopicPrefix / update/rejected	update请求被拒绝 AWS IoT，消息正文包含错误信息。	响应消息正文中的错误消息。
ShadowTopicPrefix / update/delta	影子文档已通过向的请求进行了更新 AWS IoT，消息正文包含所请求的更改。	更新设备的状态以与消息正文中的所需状态匹配。

主题	含义	在收到该主题时设备应执行的操作
ShadowTopicPrefix /update/documents	最近完成了影子更新，并且消息正文包含当前影子文档。	确认消息正文中的更新状态与设备的状态匹配。

在为每个影子订阅上表中的消息后，设备应进行测试，以确定是否已将 /get 主题发布到它支持的每个影子以创建这些影子。如果收到 /get/accepted 消息，则消息正文包含影子文档，设备可以使用该文档初始化其状态。如果收到 /get/rejected 消息，应发布具有当前设备状态的 /update 消息以创建影子。

例如，假设您有事物 My_IoT_Thing，但其没有任何经典或命名影子。如果现在发布关于预留主题 $aws/things/My_IoT_Thing/shadow/get 的 /get 请求，其会在 $aws/things/My_IoT_Thing/shadow/get/rejected 主题返回一条错误，因为该事物没有任何影子。要想解决此错误，请先使用含当前设备状态（例如以下有效负载）的 $aws/things/My_IoT_Thing/shadow/update 主题发布一条 /update 消息。

```
{
 "state": {
  "reported": {
   "welcome": "aws-iot",
   "color": "yellow"
  }
 }
}
```

现在为该事物创建了经典影子，消息将发布到 $aws/things/My_IoT_Thing/shadow/update/accepted 主题。如果发布到 $aws/things/My_IoT_Thing/shadow/get 主题，其会将含设备状态的响应返回到 $aws/things/My_IoT_Thing/shadow/get/accepted 主题。

对于命名影子，必须先创建命名影子或发布含影子名称的更新，才能使用 get 请求。例如，要创建命名影子 namedShadow1，首先将设备状态信息发布到 $aws/things/My_IoT_Thing/shadow/name/namedShadow1/update 主题。若要检索状态信息，请将 /get 请求用于命名影子 $aws/things/My_IoT_Thing/shadow/name/namedShadow1/get。

在设备连接时处理消息 AWS IoT

当设备连接时 AWS IoT，它可以接收 /update/delta 消息，并应通过以下方式使设备状态与其阴影中的变化保持一致：

1. 读取所有收到的 /update/delta 消息并同步设备状态以保持匹配。
2. 以 `reported` 消息正文发布 /update 消息，该消息正文包含设备的当前状态，无论设备状态何时发生变化。

在连接设备后，它应在出现指示时发布这些消息。

指示	主题	有效负载
设备的状态已发生变化。	*ShadowTopicPrefix* / update	具有 `reported` 属性的影子文档。
设备可能与影子不同步。	*ShadowTopicPrefix* /get	（空）
对设备执行的操作指示设备不再支持影子，例如在移除或更换设备时。	*ShadowTopicPrefix* / delete	（空）

设备重新连接时处理消息 AWS IoT

当带有一个或多个阴影的设备连接到时 AWS IoT，它应通过以下方式将其状态与其支持的所有阴影的状态同步：

1. 读取所有收到的 /update/delta 消息并同步设备状态以保持匹配。
2. 使用 `reported` 消息正文发布 /update 消息，该消息正文包含设备的当前状态。

在应用程序和服务中使用影子

本节介绍应用程序或服务如何与 Dev AWS IoT ice Shadow 服务进行交互。该示例假定应用程序或服务仅与影子进行交互，并通过影子与设备进行交互。该示例不包括任何管理操作，例如创建或删除影子。

此示例使用 Dev AWS IoT ice Shadow 服务的 REST API 与阴影进行交互。与 在设备中使用影子 中使用的示例（它使用发布/订阅通信模型）不同，该示例使用 REST API 的请求/响应通信模型。这意味着应用程序或服务必须先发出请求，然后才能收到来自的响应 AWS IoT。但该模型的一个缺点是，它不支持通知。如果应用程序或服务需要及时通知设备状态变化，请考虑使用 MQTT 或基于 WSS 的 MQTT 协议，这些协议支持发布/订阅通信模型，如在设备中使用影子中所述。

> **Important**
>
> 确保应用程序或服务使用的影子与设备中的相应实施保持一致，并且该实施支持使用这些影子。例如，考虑如何创建、更新和删除影子，以及如何在设备以及访问影子的应用程序或服务中处理更新。您的设计应明确指定如何更新和报告设备的状态，以及应用程序和服务如何与设备及其影子进行交互。

命名的影子的 REST API URL 为：

```
https://endpoint/things/thingName/shadow?name=shadowName
```

未命名的影子的 REST API URL 为：

```
https://endpoint/things/thingName/shadow
```

其中：

端点

CLI 命令返回的终端节点：

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

thingName

影子所属的事物对象的名称

shadowName

命名的影子的名称。该参数不用于未命名的影子。

在连接到时初始化应用程序或服务 AWS IoT

当应用程序首次连接时 AWS IoT，它应该向其用于获取其正在使用的阴影的当前状态的阴影的 URL 发送 HTTP GET 请求。这样，它就可以将应用程序或服务同步到影子。

当应用程序或服务连接到时，处理状态会发生变化 AWS IoT

当应用程序或服务连接到时 AWS IoT，它可以通过在其使用的影子的 URL 上发送 HTTP GET 请求来定期查询当前状态。

在最终用户与应用程序或服务交互以更改设备状态时，应用程序或服务可以向它使用的影子的 URL 发送 HTTP POST 请求以更新影子的 `desired` 状态。该请求返回已接受的更改，但您可能需要发出 HTTP GET 请求以轮询影子，直到设备使用新状态更新了影子为止。

检测是否连接了设备

要确定当前是否连接了设备，请在影子文档中包含一个 `connected` 属性；如果设备由于错误而断开连接，则使用 MQTT Last Will and Testament (LWT) 消息将 `connected` 属性设置为 `false`。

> **Note**
>
> Dev AWS IoT ice Shadow 服务会忽略发送到 AWS IoT 保留主题（以 $ 开头的主题）的 MQTT LWT 消息。但是，它们由订阅的客户端和 AWS IoT 规则引擎处理，因此您需要创建一条发送到非保留主题的 LWT 消息，以及一条规则，将 MQTT LWT 消息作为影子更新消息重新发布到影子的保留更新主题。*ShadowTopicPrefix*/update

向 Device Shadow 服务发送 LWT 消息

1. 创建一个规则以在保留的主题上重新发布 MQTT LWT 消息。以下示例规则会侦听关于 `my/things/myLightBulb/update` 主题的消息并其重新发布到 `$aws/things/myLightBulb/shadow/update`。

```
{
    "rule": {
    "ruleDisabled": false,
    "sql": "SELECT * FROM 'my/things/myLightBulb/update'",
    "description": "Turn my/things/ into $aws/things/",
    "actions": [
        {
        "republish": {
            "topic": "$$aws/things/myLightBulb/shadow/update",
            "roleArn": "arn:aws:iam:123456789012:role/aws_iot_republish"
            }
        }
```

```
     ]
   }
}
```

2. 当设备连接到时 AWS IoT，它会向非保留主题注册一条 LWT 消息，以供重新发布规则识别。在此示例中，该主题为 my/things/myLightBulb/update，它将 connected 属性设置为 false。

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

3. 在连接后，设备在其影子更新主题 $aws/things/myLightBulb/shadow/update 上发布消息以报告其当前状态，这包括将其 connected 属性设置为 true。

```
{
     "state": {
        "reported": {
            "connected":"true"
        }
    }
}
```

4. 在设备正常断开连接之前，它在其影子更新主题 $aws/things/myLightBulb/shadow/update 上发布消息以报告其最新状态，这包括将其 connected 属性设置为 false。

```
{
    "state": {
        "reported": {
            "connected":"false"
        }
    }
}
```

5. 如果设备因错误而断开连接，则 AWS IoT 消息代理将代表设备发布设备的 LWT 消息。重新发布规则检测该消息，并发布影子更新消息以更新设备影子的 connected 属性。

模拟 Device Shadow 服务通信

本主题说明了 Device Shadow 服务如何充当中介，并允许设备和应用程序使用影子以更新、存储和检索设备的状态。

为了演示本主题中描述的交互并对其进行进一步探索，你需要一个 AWS 账户 和一个可以运行的系统 AWS CLI。如果没有该账户和系统，您仍然可以在代码示例中看到交互。

在此示例中， AWS IoT 控制台代表设备。 AWS CLI 表示通过影子访问设备的应用程序或服务。该 AWS CLI 接口与应用可能用于通信的 API 非常相似 AWS IoT。该示例中的设备是一个智能灯泡，应用程序显示灯泡的状态，并且可以更改灯泡的状态。

设置模拟

这些流程打开模拟设备的 AWS IoT 控制台和模拟应用程序的命令行窗口以初始化模拟。

设置模拟环境

1. 你需要一个 AWS 账户 才能自己运行本主题中的示例。如果您没有 AWS 账户，请按中所述创建一个设置你的 AWS 账户。
2. 打开 AWS IoT 控制台，然后在左侧菜单中选择 Test (测试) 以打开 MQTT client (MQTT 客户端)。
3. 在另一个窗口中，在已安装 AWS CLI 的系统上打开一个终端窗口。

你应该打开两个窗口：一个在 “测试” 页面上显示 AWS IoT 控制台，另一个窗口带有命令行提示符。

初始化设备

在这个模拟中，我们将使用一个名为、的事物对象 mySimulatedThing，其阴影名为 simShadow1。

创建事物对象及其 IoT 策略

要创建事物对象，请在 AWS IoT 控制台：

1. 选择 Manage（管理），然后选择 Things（事物）。
2. 如果已列出事物，请单击 “创建” 按钮，否则单击 “注册单个事物” 来创建单个 AWS IoT 事物。
3. 输入名称 `mySimulatedThing`，将其它设置保留为默认值，然后单击 Next（下一步）。
4. 使用单击证书创建可生成证书，以验证设备与 AWS IoT 的连接。单击 Activate（激活）以激活证书。

5. 您可以附上策略 My_IoT_Policy，这将授予设备发布和订阅 MQTT 预留主题的权限。有关如何创建 AWS IoT 事物以及如何创建此策略的更多详细步骤，请参阅创建一个事物对象。

创建事物对象的命名影子

按如下方式向 $aws/things/mySimulatedThing/shadow/name/simShadow1/update 主题发布更新请，您可以为物件创建命名影子。

或者，要创建命名的阴影，请执行以下操作：

1. 在 AWS IoT 控制台中，在显示的事物列表中选择您的事物对象，然后选择 Shadows（影子）。
2. 选择 Add a shadow（添加影子），输入名称 simShadow1，然后选择 Create（创建）添加命名的影子。

订阅并发布预留的 MQTT 主题

在控制台中，订阅预留的 MQTT 影子主题。这些主题是 get、update 和 delete 操作的响应，以便设备在发布操作后准备好接收响应。

在 MQTT client (MQTT 客户端) 中订阅 MQTT 主题

1. 在 MQTT 客户端中，选择 Subscribe to a topic（订阅主题）。
2. 输入 get、update 和 delete 要订阅的主题。每次从以下列表中复制一个主题，然后将其粘贴到 Topic filter（主题筛选条件）字段，然后单击 Subscribe（订阅）。您应看到主题显示在 Subscriptions（订阅）项下。

   - $aws/things/mySimulatedThing/shadow/name/simShadow1/delete/accepted
   - $aws/things/mySimulatedThing/shadow/name/simShadow1/delete/rejected
   - $aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted
   - $aws/things/mySimulatedThing/shadow/name/simShadow1/get/rejected
   - $aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted
   - $aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected
   - $aws/things/mySimulatedThing/shadow/name/simShadow1/update/delta
   - $aws/things/mySimulatedThing/shadow/name/simShadow1/update/documents

   此时，在 AWS IoT 发布主题时，模拟的设备已准备好接收这些主题。

在 MQTT client（MQTT 客户端）中发布 MQTT 主题

在设备初始化自身并订阅响应主题后，它应查询支持的影子。此模拟仅支持一个阴影，即支持名为 simShadow1 的事物对象的阴影。mySimulatedThing

从 MQTT client (MQTT 客户端) 中获取当前影子状态

1. 在 MQTT client (MQTT 客户端) 中，选择 Publish to a topic (发布到主题)。
2. 在 Publish（发布）中，输入以下主题，并从输入主题下方的消息正文窗口中删除任何内容。然后，您可以选择 Publish to topic（发布主题）以发布请求。`$aws/things/mySimulatedThing/shadow/name/simShadow1/get`。

   如果您未创建命名影子 `simShadow1`，您将在 `$aws/things/mySimulatedThing/shadow/name/simShadow1/get/rejected` 主题中收到一条消息，并且 `code` 为 `404`（例如，在该示例中，由于尚未创建影子，因此我们接下来将进行创建）。

   ```
   {
     "code": 404,
     "message": "No shadow exists with name: 'simShadow1'"
   }
   ```

使用设备的当前状态创建影子

1. 在 MQTT client（MQTT 客户端）中，选择 Publish to a topic（发布主题），然后输入该主题。

   ```
   $aws/things/mySimulatedThing/shadow/name/simShadow1/update
   ```

2. 在消息正文窗口中，在您输入主题的位置下面输入该影子文档，以显示设备正在报告其 ID 并以 RGB 值形式报告其当前颜色。选择 Publish（发布）以发布请求。

   ```
   {
     "state": {
       "reported": {
         "ID": "SmartLamp21",
         "ColorRGB": [
           128,
           128,
           128
         ]
       }
   ```

```
  },
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

如果您在主题中收到一条消息：

- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/accepted：表示创建了影子，并且消息正文包含当前影子文档。
- $aws/things/mySimulatedThing/shadow/name/simShadow1/update/rejected：查看消息正文中的错误。
- $aws/things/mySimulatedThing/shadow/name/simShadow1/get/accepted：影子已存在，并且消息正文包含当前影子状态，如示例中所示。通过使用该消息，您可以设置设备或确认它与影子状态匹配。

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        }
      ]
```

```
    }
  },
  "version": 3,
  "timestamp": 1591140517,
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

从应用程序中发送更新

本节使用 AWS CLI 来演示应用程序如何与影子交互。

要获取阴影的当前状态，请使用 AWS CLI

从命令行中，输入以下命令。

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1 /
dev/stdout
```

在 Windows 平台上，您可以使用 con 而不是 /dev/stdout。

```
aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name simShadow1
 con
```

由于影子存在并且设备已将其初始化以反映其当前状态，因此，它应返回以下影子文档。

```
{
  "state": {
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        128,
        128,
        128
      ]
    }
  },
  "metadata": {
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
```

```
      "ColorRGB": [
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        },
        {
          "timestamp": 1591140517
        }
      ]
    }
  },
  "version": 3,
  "timestamp": 1591141111
}
```

应用程序可以使用该响应以初始化设备状态的表示形式。

如果应用程序更新状态（例如，在最终用户将智能灯泡的颜色更改为黄色时），应用程序将发送 update-thing-shadow 命令。该命令对应于 UpdateThingShadow REST API。

从应用程序中更新影子

从命令行中，输入以下命令。

AWS CLI v2.x

```
aws iot-data update-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 \
    --cli-binary-format raw-in-base64-out \
    --payload '{"state":{"desired":{"ColorRGB":
[255,255,0]}},"clientToken":"21b21b21-bfd2-4279-8c65-e2f697ff4fab"}' /dev/stdout
```

AWS CLI v1.x

```
aws iot-data update-thing-shadow --thing-name mySimulatedThing --shadow-name
 simShadow1 \
    --payload '{"state":{"desired":{"ColorRGB":
[255,255,0]}},"clientToken":"21b21b21-bfd2-4279-8c65-e2f697ff4fab"}' /dev/stdout
```

如果成功，该命令应返回以下影子文档。

```
{
  "state": {
    "desired": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "desired": {
      "ColorRGB": [
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        }
      ]
    }
  },
  "version": 4,
  "timestamp": 1591141596,
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

响应设备中的更新

返回 AWS 控制台中的 MQTT 客户端，您应该会看到为反映上一节中发出的更新命令而发布的消息。AWS IoT

在 MQTT client (MQTT 客户端) 中查看更新消息

在 MQTT 客户端中，在 “订阅” 列中选择 $aws/things/ /shadow/name/simshadow1/UpdatemySimulatedThing/Delta。如果主题名称被截断，您可以在其中暂停以查看完整主题。在本主题的主题日志中，您应该看到与此类似的 `/delta` 消息。

```
{
```

```
  "version": 4,
  "timestamp": 1591141596,
  "state": {
    "ColorRGB": [
      255,
      255,
      0
    ]
  },
  "metadata": {
    "ColorRGB": [
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      },
      {
        "timestamp": 1591141596
      }
    ]
  },
  "clientToken": "21b21b21-bfd2-4279-8c65-e2f697ff4fab"
}
```

设备将处理该消息的内容，以将设备状态设置为与消息中的 `desired` 状态匹配。

在设备更新状态以匹配消息中的`desired`状态后，它必须 AWS IoT 通过发布更新消息将新的报告状态发送回去。该流程在 MQTT client (MQTT 客户端) 中模拟这种情况。

从设备中更新影子

1. 在 MQTT client (MQTT 客户端) 中，选择 Publish to a topic (发布到主题)。
2. 在消息正文窗口中，在消息正文窗口上方的主题字段中，输入影子的主题，然后输入 `/update` 操作：`$aws/things/mySimulatedThing/shadow/name/simShadow1/update`，然后在消息正文中输入此更新的影子文档，该文档描述了设备的当前状态。选择 Publish（发布）以发布更新的设备状态。

   ```
   {
     "state": {
       "reported": {
         "ColorRGB": [255,255,0]
   ```

```
      }
  },
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

如果成功接收了消息 AWS IoT，则您应该在 MQTT 客户端的 $aws/things/ /shadow/ mySimulatedThing name/simshadow1/Update/accepte/accepted 消息日志中看到新的响应，其中包含影子的当前状态，例如此示例。

```
{
  "state": {
    "reported": {
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "reported": {
      "ColorRGB": [
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        }
      ]
    }
  },
  "version": 5,
  "timestamp": 1591142747,
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

成功更新设备报告的状态还会导致 AWS IoT 在消息中向主题发送有关影子状态的全面描述，例如该消息正文，该消息正文是由设备在前面的过程中执行的影子更新所生成的消息正文。

```
{
  "previous": {
    "state": {
      "desired": {
        "ColorRGB": [
          255,
          255,
          0
        ]
      },
      "reported": {
        "ID": "SmartLamp21",
        "ColorRGB": [
          128,
          128,
          128
        ]
      }
    },
    "metadata": {
      "desired": {
        "ColorRGB": [
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          }
        ]
      },
      "reported": {
        "ID": {
          "timestamp": 1591140517
        },
        "ColorRGB": [
          {
            "timestamp": 1591140517
          },
          {
            "timestamp": 1591140517
```

```
          },
          {
            "timestamp": 1591140517
          }
        ]
      }
    },
    "version": 4
  },
  "current": {
    "state": {
      "desired": {
        "ColorRGB": [
          255,
          255,
          0
        ]
      },
      "reported": {
        "ID": "SmartLamp21",
        "ColorRGB": [
          255,
          255,
          0
        ]
      }
    },
    "metadata": {
      "desired": {
        "ColorRGB": [
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          },
          {
            "timestamp": 1591141596
          }
        ]
      },
      "reported": {
        "ID": {
          "timestamp": 1591140517
```

```
        },
        "ColorRGB": [
          {
            "timestamp": 1591142747
          },
          {
            "timestamp": 1591142747
          },
          {
            "timestamp": 1591142747
          }
        ]
      }
    },
    "version": 5
  },
  "timestamp": 1591142747,
  "clientToken": "a4dc2227-9213-4c6a-a6a5-053304f60258"
}
```

观察应用程序中的更新

应用程序现在可以查询影子以获取设备报告的当前状态。

要获取阴影的当前状态，请使用 AWS CLI

1. 从命令行中，输入以下命令。

   ```
   aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name
    simShadow1 /dev/stdout
   ```

   在 Windows 平台上，您可以使用 con 而不是 /dev/stdout。

   ```
   aws iot-data get-thing-shadow --thing-name mySimulatedThing --shadow-name
    simShadow1 con
   ```

2. 由于设备刚刚更新了影子以反映其当前状态，因此，它应返回以下影子文档。

   ```
   {
     "state": {
       "desired": {
         "ColorRGB": [
   ```

```
        255,
        255,
        0
      ]
    },
    "reported": {
      "ID": "SmartLamp21",
      "ColorRGB": [
        255,
        255,
        0
      ]
    }
  },
  "metadata": {
    "desired": {
      "ColorRGB": [
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        },
        {
          "timestamp": 1591141596
        }
      ]
    },
    "reported": {
      "ID": {
        "timestamp": 1591140517
      },
      "ColorRGB": [
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        },
        {
          "timestamp": 1591142747
        }
      ]
    }
```

```
  },
  "version": 5,
  "timestamp": 1591143269
}
```

模拟补充内容

试验 AWS CLI（表示应用程序）和控制台（表示设备）之间的交互以模拟您的 IoT 解决方案。

与影子交互

本主题介绍了 AWS IoT 为处理影子而提供的三种方法的关联消息。这些方法包括：

UPDATE

创建影子（如果不存在），或使用消息正文中提供的状态信息更新现有影子的内容。 AWS IoT 记录每次更新的时间戳，以指示上次更新状态的时间。当影子的状态发生变化时， AWS IoT 会向所有 MQTT 订阅者发送带有desired和reported状态差异的/delta消息。收到 /delta 消息的设备或应用程序可以根据该差异执行操作。例如，设备可以将其状态更新为所需的状态，或者应用程序可以更新其 UI 以反映设备的状态变化。

GET

检索包含影子的完整状态的当前影子文档，包括元数据。

DELETE

删除设备影子及其内容。

您无法还原已删除的设备影子文档，但可以创建具有已删除设备影子文档的名称的新设备影子。如果您创建的设备影子文档与过去48小时内删除的文档同名，新设备影子文档的版本号和已删除文档的版本号相同。如果设备影子文档已删除超过 48 小时，具有相同名称的新设备影子文档的版本号为 0。

协议支持

AWS IoT 支持 MQTT 和通过 HTTPS 协议的 REST API 来与阴影进行交互。 AWS IoT 为 MQTT 发布和订阅操作提供了一组预留的请求和响应主题。设备和应用程序应在发布请求主题之前订阅响应主题，以获取有关如何 AWS IoT 处理请求的信息。有关更多信息，请参阅 Device Shadow MQTT 主题 和 Device Shadow REST API。

请求和报告状态

在使用 AWS IoT and shadows 设计物联网解决方案时，应确定将请求更改的应用程序或设备以及将实施变更的应用程序或设备。通常，设备实施更改并向影子报告更改，而应用程序和服务响应并请求影子中的更改。您的解决方案可能有所不同，但本主题中的示例假定客户端应用程序或服务请求影子中的更改，而设备执行更改并向影子报告更改。

更新影子

应用程序或服务可以使用 UpdateThingShadow API 或发布到 /update 主题以更新影子的状态。更新仅影响请求中指定的字段。

在客户端请求状态更改时更新影子

在客户端使用 MQTT 协议请求影子中的状态更改时

1. 客户端应具有当前影子文档，以便它可以确定要更改的属性。有关如何获取当前影子文档的信息，请参阅 /get 操作。
2. 客户端订阅以下 MQTT 主题：
   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`
   - `$aws/things/thingName/shadow/name/shadowName/update/delta`
   - `$aws/things/thingName/shadow/name/shadowName/update/documents`
3. 客户端使用包含影子的所需状态的状态文档发布 `$aws/things/thingName/shadow/name/shadowName/update` 请求主题。只需在文档中包含要更改的属性。以下是具有所需状态的文档的示例。

```
{
  "state": {
    "desired": {
      "color": {
        "r": 10
      },
      "engine": "ON"
    }
  }
}
```

4. 如果更新请求有效，则在影子中 AWS IoT 更新所需的状态并发布有关以下主题的消息：

   - $aws/things/*thingName*/shadow/name/*shadowName*/update/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/delta

   /update/accepted 消息包含一个 /accepted 响应状态文档 影子文档，而 /update/delta 消息包含一个 /delta 响应状态文档 影子文档。
5. 如果更新请求无效，则 AWS IoT 发布带有$aws/things/*thingName*/shadow/name/*shadowName*/update/rejected主题的消息以及描述错误的错误响应文档影子文档。

在客户端使用 API 请求影子中的状态更改时

1. 客户端调用 UpdateThingShadow API 并将 请求状态文档 状态文档作为其消息正文。
2. 如果请求有效，则 AWS IoT 返回一个 HTTP 成功响应代码和一个/accepted 响应状态文档影子文档作为其响应消息正文。

   AWS IoT 还将向该$aws/things/*thingName*/shadow/name/*shadowName*/update/delta主题发布一条 MQTT 消息，其中包含订阅该消息的任何设备或客户端的/delta 响应状态文档影子文档。
3. 如果请求无效，则 AWS IoT 返回 HTTP 错误响应代码 a 错误响应文档 作为其响应消息正文。

当设备在 /update/delta 主题上收到 /desired 状态时，它将在设备中进行所需的更改。然后，它向 /update 主题发送一条消息，以向影子报告其当前状态。

在设备报告其当前状态时更新影子

在设备使用 MQTT 协议向影子报告其当前状态时

1. 在更新影子之前，设备应订阅以下 MQTT 主题：

   - $aws/things/*thingName*/shadow/name/*shadowName*/update/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/rejected
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/delta
   - $aws/things/*thingName*/shadow/name/*shadowName*/update/documents
2. 设备向 $aws/things/*thingName*/shadow/name/*shadowName*/update 主题发布一条消息以报告其当前状态，例如，在该示例中。

```
{
    "state": {
        "reported" : {
            "color" : { "r" : 10 },
            "engine" : "ON"
        }
    }
}
```

3. 如果 AWS IoT 接受更新，则会向$aws/things/*thingName*/shadow/name/*shadowName*/update/accepted主题发布带有/accepted 响应状态文档影子文档的消息。
4. 如果更新请求无效，则 AWS IoT 发布带有$aws/things/*thingName*/shadow/name/*shadowName*/update/rejected主题的消息以及描述错误的错误响应文档影子文档。

在设备使用 API 向影子报告其当前状态时

1. 设备调用 UpdateThingShadow API 并将 请求状态文档 状态文档作为其消息正文。
2. 如果请求有效，则 AWS IoT 更新影子并返回以影子文档作为其响应消息正文的 HTTP 成功响应代码。/accepted 响应状态文档

   AWS IoT 还将向该$aws/things/*thingName*/shadow/name/*shadowName*/update/delta主题发布一条 MQTT 消息，其中包含订阅该消息的任何设备或客户端的/delta 响应状态文档影子文档。
3. 如果请求无效，则 AWS IoT 返回 HTTP 错误响应代码 a 错误响应文档 作为其响应消息正文。

乐观锁

您可以使用状态文档版本来确保正在更新的设备的影子文档为最新版本。当您为更新请求提供版本时，如果状态文档的当前版本与提供的版本不符，该服务将显示 HTTP 409 冲突响应代码并拒绝请求。冲突响应代码也可能出现在任何修改 ThingShadow 的 API 上，包括 DeleteThingShadow。

例如：

初始文档：

```
{
  "state": {
    "desired": {
```

```
      "colors": [
        "RED",
        "GREEN",
        "BLUE"
      ]
    }
  },
  "version": 10
}
```

更新：（版本不匹配；该请求将被拒绝）

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 9
}
```

结果：

```
{
  "code": 409,
  "message": "Version conflict",
  "clientToken": "426bfd96-e720-46d3-95cd-014e3ef12bb6"
}
```

更新：（版本匹配；请求将被接受）

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 10
```

```
}
```

最终状态：

```
{
  "state": {
    "desired": {
      "colors": [
        "BLUE"
      ]
    }
  },
  "version": 11
}
```

检索影子文档

您可以使用 GetThingShadow API 或订阅并发布到 /get 主题以检索影子文档。这会检索完整的影子文档，包括 desired 和 reported 状态之间的任何增量。无论设备还是客户端发出请求，该任务的流程都是相同的。

使用 MQTT 协议检索影子文档

1. 在更新影子之前，设备或客户端应订阅以下 MQTT 主题：
   - $aws/things/*thingName*/shadow/name/*shadowName*/get/accepted
   - $aws/things/*thingName*/shadow/name/*shadowName*/get/rejected
2. 设备或客户端使用空消息正文向 $aws/things/*thingName*/shadow/name/*shadowName*/get 主题发布一条消息。
3. 如果请求成功，则向$aws/things/*thingName*/shadow/name/*shadowName*/get/accepted主题 AWS IoT 发布一条消息，消息正文/accepted 响应状态文档中包含一个。
4. 如果请求无效，则向$aws/things/*thingName*/shadow/name/*shadowName*/get/rejected主题 AWS IoT 发布一条消息，并在消息正文错误响应文档中加上。

使用 REST API 检索影子文档

1. 设备或客户端使用空消息正文调用 GetThingShadow API。

2. 如果请求有效，则 AWS IoT 返回一个 HTTP 成功响应代码，其响应消息正文为/accepted 响应状态文档影子文档。
3. 如果请求无效，则 AWS IoT 返回 HTTP 错误响应代码 a 错误响应文档 作为其响应消息正文。

删除影子数据

可以使用两种方法删除影子数据：您可以在影子文档中删除特定的属性，也可以完全删除影子。

- 要从影子中删除特定的属性，请更新影子，但将您要删除的属性的值设置为 `null`。将从影子文档中删除值为 `null` 的字段。
- 要删除整个影子，请使用 DeleteThingShadow API 或发布到 /delete 主题。

> **Note**
>
> 删除阴影不会立即将其版本号重置为零。将在 48 小时后重置为零。

从影子文档中删除属性

使用 MQTT 协议从影子中删除属性

1. 设备或客户端应具有当前影子文档，以便它可以确定要更改的属性。有关如何获取当前影子文档的信息，请参阅 检索影子文档。
2. 设备或客户端订阅以下 MQTT 主题：
   - `$aws/things/thingName/shadow/name/shadowName/update/accepted`
   - `$aws/things/thingName/shadow/name/shadowName/update/rejected`
3. 设备或客户端使用将 `null` 值分配给要删除的影子属性的状态文档以发布 `$aws/things/thingName/shadow/name/shadowName/update` 请求主题。只需在文档中包含要更改的属性。以下是删除 `engine` 属性的文档的示例。

```
{
  "state": {
    "desired": {
      "engine": null
    }
  }
```

```
}
```

4. 如果更新请求有效，则 AWS IoT 删除影子中的指定属性，并发布带有$aws/things/*thingName*/shadow/name/*shadowName*/update/accepted主题的消息，消息正文中包含/accepted 响应状态文档影子文档。
5. 如果更新请求无效，则 AWS IoT 发布带有$aws/things/*thingName*/shadow/name/*shadowName*/update/rejected主题的消息以及描述错误的错误响应文档影子文档。

使用 REST API 从影子中删除属性

1. 设备或客户端使用 请求状态文档 将 null 值分配给要删除的影子属性的以调用 UpdateThingShadow API。仅在文档中包含要删除的属性。以下是删除 engine 属性的文档的示例。

```
{
  "state": {
    "desired": {
      "engine": null
    }
  }
}
```

2. 如果请求有效，则 AWS IoT 返回一个 HTTP 成功响应代码和一个/accepted 响应状态文档影子文档作为其响应消息正文。
3. 如果请求无效，则 AWS IoT 返回 HTTP 错误响应代码 a 错误响应文档 作为其响应消息正文。

删除影子

以下是删除设备影子时的一些注意事项。

- 将设备的影子状态设置为 null 并不会删除影子。在下次更新时，影子版本将会增加。
- 删除设备的影子并不会删除事物对象。删除事物对象并不会删除相应设备的影子。
- 删除阴影不会立即将其版本号重置为零。将在 48 小时后重置为零。

使用 MQTT 协议删除影子

1. 设备或客户端订阅以下 MQTT 主题：

- $aws/things/*thingName*/shadow/name/*shadowName*/delete/accepted
- $aws/things/*thingName*/shadow/name/*shadowName*/delete/rejected

2. 设备或客户端使用空消息缓冲区发布 $aws/things/*thingName*/shadow/name/*shadowName*/delete。
3. 如果删除请求有效，则 AWS IoT 删除影子并在消息正文中发布带有$aws/things/*thingName*/shadow/name/*shadowName*/delete/accepted主题和缩写/accepted 响应状态文档影子文档的消息。以下是接受的删除消息的示例：

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

4. 如果更新请求无效，则 AWS IoT 发布带有$aws/things/*thingName*/shadow/name/*shadowName*/delete/rejected主题的消息以及描述错误的错误响应文档影子文档。

使用 REST API 删除影子

1. 设备或客户端使用空消息缓冲区调用 DeleteThingShadow API。
2. 如果请求有效，则在消息正文中 AWS IoT 返回一个 HTTP 成功响应代码/accepted 响应状态文档和一个缩写的/accepted 响应状态文档影子文档。以下是接受的删除消息的示例：

```
{
  "version": 4,
  "timestamp": 1591057529
}
```

3. 如果请求无效，则 AWS IoT 返回 HTTP 错误响应代码 a 错误响应文档 作为其响应消息正文。

Device Shadow REST API

影子将显示用于更新状态信息的以下 URI：

```
https://account-specific-prefix-ats.iot.region.amazonaws.com/things/thingName/shadow
```

终端节点特定于您的 AWS 账户。要查找端点，您可以：

- 从 AWS CLI 使用 describe-endpoint 命令。
- 使用 AWS IoT 控制台设置。在 Settings（设置），端点将列在 Custom endpoint（自定义端点）下。
- 使用 AWS IoT 控制台事物详细信息页面。在控制台中，执行以下操作：
  1. 打开 Manage（管理），并在 Manage（管理）中，选择 Things（事物）。
  2. 在事物列表中，选择要获取端点 URI 的事物。
  3. 选择 Device Shadows 选项卡，然后选择影子。您可以在 Device Shadow details（Device Shadow 详细信息）页面的 Device Shadow URL 部分查看端点 URI。

终端节点的格式如下：

```
identifier.iot.region.amazonaws.com
```

shadow REST API 遵循设备通信协议中所述的相同 HTTPS 协议/端口映射。

> **Note**
>
> 您必须将 `iotdevicegateway` 用作身份验证的服务名称，才能使用 API。有关更多信息，请参阅 IoT DataPlane。

API 操作

- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

您还可以通过提供 `name=shadowName` 用 API 创建命名影子，作为 API 的查询参数的一部分。

GetThingShadow

获取指定事物的影子。

响应状态文档包括 `desired` 状态与 `reported` 状态之间的增量。

请求

该请求包括标准的 HTTP 标头以及以下 URI：

```
HTTP GET https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

未命名的（经典）影子不需要使用 name 查询参数。

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
Response Body: response state document
```

有关更多信息，请参阅响应状态文档示例。

授权

要检索影子，需要一项允许调用方执行 iot:GetThingShadow 操作的策略。Device Shadow 服务接受两种形式的身份验证：使用 IAM 凭证的 Signature Version 4 或使用客户端证书的 TLS 双向身份验证。

以下是允许调用方检索设备的影子的示例策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:GetThingShadow",
      "Resource": [
        "arn:aws:iot:region:account:thing/thing"
      ]
    }
  ]
}
```

UpdateThingShadow

更新指定事物的影子。

更新仅影响请求状态文档中指定的字段。值为 null 的所有字段都会从设备的影子中删除。

请求

该请求包括标准的 HTTP 标头以及以下 URI 和正文：

```
HTTP POST https://endpoint/things/thingName/shadow?name=shadowName
Request body: request state document
```

未命名的（经典）影子不需要使用 name 查询参数。

有关更多信息，请参阅请求状态文档示例。

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
Response body: response state document
```

有关更多信息，请参阅响应状态文档示例。

授权

要更新影子，需要一项允许调用方执行 iot:UpdateThingShadow 操作的策略。Device Shadow 服务接受两种形式的身份验证：使用 IAM 凭证的 Signature Version 4 或使用客户端证书的 TLS 双向身份验证。

以下是允许调用方更新设备的影子的示例策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:UpdateThingShadow",
      "Resource": [
        "arn:aws:iot:region:account:thing/thing"
      ]
    }
  ]
}
```

DeleteThingShadow

删除指定事物的影子。

请求

该请求包括标准的 HTTP 标头以及以下 URI：

```
HTTP DELETE https://endpoint/things/thingName/shadow?name=shadowName
Request body: (none)
```

未命名的（经典）影子不需要使用 name 查询参数。

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
Response body: Empty response state document
```

请注意，删除影子不会将其版本号重置为 0。

授权

要删除设备的影子，需要一项允许调用方执行 `iot:DeleteThingShadow` 操作的策略。Device Shadow 服务接受两种形式的身份验证：使用 IAM 凭证的 Signature Version 4 或使用客户端证书的 TLS 双向身份验证。

以下是允许调用方删除设备的影子的示例策略：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:DeleteThingShadow",
      "Resource": [
        "arn:aws:iot:region:account:thing/thing"
      ]
    }
  ]
```

```
}
```

ListNamedShadowsForThing

列出指定事物的影子。

请求

该请求包括标准的 HTTP 标头以及以下 URI：

```
HTTP GET /api/things/shadow/ListNamedShadowsForThing/thingName?
nextToken=nextToken&pageSize=pageSize
Request body: (none)
```

nextToken

用于检索下一组结果的令牌。

该值在分页结果中返回，并在返回下一页的调用中使用。

pageSize

在每个调用中返回的影子名称的数量。另请参阅 nextToken。

thingName

要列出命名的影子的事物的名称。

响应

在成功后，响应将包括标准 HTTP 标头以及以下响应代码和 影子名称列表响应文档。

> **Note**
>
> 未命名的（经典）影子不会显示在该列表中。如果只有经典影子或 thingName 您指定的影子不存在，响应为空列表。

```
HTTP 200
Response body: Shadow name list document
```

授权

要列示设备的影子，需要一项允许调用方执行 `iot:ListNamedShadowsForThing` 操作的策略。Device Shadow 服务接受两种形式的身份验证：使用 IAM 凭证的 Signature Version 4 或使用客户端证书的 TLS 双向身份验证。

以下是一个示例策略，它允许调用方列出事物的命名影子：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:ListNamedShadowsForThing",
      "Resource": [
        "arn:aws:iot:region:account:thing/thing"
      ]
    }
  ]
}
```

Device Shadow MQTT 主题

Device Shadow 服务使用保留的 MQTT 主题，以使设备和应用程序能够获取、更新或删除设备的状态信息（影子）。

要订阅影子主题并向该主题发布消息，需要获得基于主题的授权。 AWS IoT 保留向现有主题结构添加新主题的权利。为此，建议您订阅影子主题时勿使用通配符。例如，避免订阅主题过滤器，`$aws/things/thingName/shadow/#`因为与该主题过滤器匹配的主题数量可能会随着新的影子主题的 AWS IoT 引入而增加。要了解针对这些主题发布的消息示例，请查看 与影子交互。

影子可以是命名或未命名（经典）的。每个影子使用的主题仅在主题前缀上有所不同。下表显示每种影子类型使用的主题前缀。

ShadowTopicPrefix 价值	影子类型
`$aws/things/` *thingName* `/shadow`	未命名的（经典）影子
`$aws/things/` *thingName* `/shadow/name/` *shadowName*	命名的影子

要创建完整的主题，请为要表示的影子类型选择 *ShadowTopicPrefix*，将 *thingName* 和 *shadowName*（如果适用）替换为相应的值，然后在其后面附加主题存根，如以下几部分中所示。

以下是用于与影子交互的 MQTT 主题。

主题

- /get
- /get/accepted
- /get/rejected
- /update
- /update/delta
- /update/accepted
- /update/documents
- /update/rejected
- /delete
- /delete/accepted
- /delete/rejected

/get

要获得设备的影子，请在此主题下发布一条空消息：

```
ShadowTopicPrefix/get
```

AWS IoT 通过发布/get/accepted或来响应/get/rejected。

策略示例

以下是所需策略的示例：

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

```
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/get"
      ]
    }
  ]
}
```

/get/accepted

AWS IoT 返回设备的影子时，会向此主题发布响应影子文档：

```
ShadowTopicPrefix/get/accepted
```

有关更多信息，请参阅响应状态文档。

策略示例

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/get/
accepted"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/get/accepted"
      ]
    }
```

```
  ]
}
```

/get/rejected

AWS IoT 当无法返回设备的影子时，会针对此主题发布错误响应文档：

```
ShadowTopicPrefix/get/rejected
```

有关更多信息，请参阅错误响应文档。

策略示例

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/get/
rejected"
      ]
    },
    {
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/get/rejected"
      ]
    }
  ]
}
```

/update

向此主题发布请求状态文档来更新设备的影子：

```
ShadowTopicPrefix/update
```

消息正文包含部分请求状态文档。

尝试更新设备状态的客户端将发送具有 desired 属性的 JSON 请求状态文档，例如：

```
{
  "state": {
    "desired": {
      "color": "red",
      "power": "on"
    }
  }
}
```

更新其影子的设备将发送具有 reported 属性的 JSON 请求状态文档，例如：

```
{
  "state": {
    "reported": {
      "color": "red",
      "power": "on"
    }
  }
}
```

AWS IoT 通过发布/update/accepted或来响应/update/rejected。

策略示例

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/update"
```

```
      ]
    }
  ]
}
```

/update/delta

AWS IoT 当该主题接受设备影子的更改时，会发布该主题的响应状态文档，并且请求状态文档包含不同的值desired和reported状态：

```
ShadowTopicPrefix/update/delta
```

消息缓冲区包含一个 /delta 响应状态文档。

消息正文详细信息

- 发布到 update/delta 的消息仅包括 desired 部分和 reported 部分之间有所不同的预期属性。无论这些属性包含在当前更新消息中还是已存储在 AWS IoT 中，它将包含所有此类属性。desired 部分和 reported 部分之间相同的属性则不包含在内。
- 如果某个属性位于 reported 部分，但在 desired 部分没有等效值，则不会包含在内。
- 如果某个属性位于 desired 部分，但在 reported 部分没有等效值，则将包含在内。
- 如果某个属性已从 reported 部分删除，但仍存在于 desired 部分，则将包含在内。

策略示例

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/update/
delta"
      ]
    },
```

```
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/update/delta"
      ]
    }
  ]
}
```

/update/accepted

AWS IoT 当它接受设备影子的更改时，会发布此主题的响应状态文档：

```
ShadowTopicPrefix/update/accepted
```

消息缓冲区包含一个 /accepted 响应状态文档。

策略示例

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/update/
accepted"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
```

```
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/update/accepted"
      ]
    }
  ]
}
```

/update/documents

AWS IoT 每当成功执行影子更新时，都会向该主题发布状态文档：

```
ShadowTopicPrefix/update/documents
```

消息正文包含一个 /documents 响应状态文档。

策略示例

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/update/
documents"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/update/
documents"
      ]
    }
  ]
```

```
}
```

/update/rejected

AWS IoT 当该主题拒绝对设备影子的更改时，会发布针对此主题的错误响应文档：

```
ShadowTopicPrefix/update/rejected
```

消息正文包含一个错误响应文档。

策略示例

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/update/
rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/update/rejected"
      ]
    }
  ]
}
```

/delete

要删除设备的影子，请将一条空消息发布到删除主题：

```
ShadowTopicPrefix/delete
```

消息内容将被忽略。

请注意，删除影子不会将其版本号重置为 0。

AWS IoT 通过发布/delete/accepted或来响应/delete/rejected。

策略示例

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/delete"
      ]
    }
  ]
}
```

/delete/accepted

AWS IoT 删除设备的影子后，会向此主题发布一条消息：

```
ShadowTopicPrefix/delete/accepted
```

策略示例

以下是所需策略的示例：

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

```
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/delete/
accepted"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/delete/accepted"
      ]
    }
  ]
}
```

/delete/rejected

AWS IoT 当无法删除设备的影子时，会针对此主题发布错误响应文档：

```
ShadowTopicPrefix/delete/rejected
```

消息正文包含一个错误响应文档。

策略示例

以下是所需策略的示例：

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

```
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/delete/
rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/delete/rejected"
      ]
    }
  ]
}
```

Device Shadow 服务文档

Device Shadow 服务遵守 JSON 规范下的所有规则。值、对象和数组均存储在设备的影子文档中。

内容

影子文档示例

在使用 REST API 或 MQTT 发布/订阅消息的 UPDATE、GET 和 DELETE 操作中，Device Shadow 服务使用以下文档。

示例

- 响应状态文档
- 错误响应文档
- 影子名称列表响应文档

请求状态文档

请求状态文档具有以下格式：

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "clientToken": "token",
    "version": version
}
```

- `state` — 更新仅影响指定字段。通常，您将使用 `desired` 或 `reported` 属性，但不能在同一请求中同时使用这两个属性。
  - `desired` — 请求在设备中更新的状态属性和值。
  - `reported` — 设备报告的状态属性和值。
- `clientToken` — 如果使用，您可以通过客户端令牌匹配请求和相应的响应。
- `version` - 如果使用，仅当指定的版本与 Device Shadow 服务拥有的最新版本相符时，该服务才会处理更新。

响应状态文档

响应状态文档具有以下格式，具体取决于响应类型。

/accepted 响应状态文档

```
{
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

/delta 响应状态文档

```
{
    "state": {
        "attribute1": integer2,
        "attribute2": "string2",
        ...
        "attributeN": boolean2
    },
    "metadata": {
        "attribute1": {
            "timestamp": timestamp
        },
```

```
        "attribute2": {
            "timestamp": timestamp
        },
        ...
        "attributeN": {
            "timestamp": timestamp
        }
    },
    "timestamp": timestamp,
    "clientToken": "token",
    "version": version
}
```

/documents 响应状态文档

```
{
  "previous" : {
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer1,
            "attribute2": "string1",
            ...
            "attributeN": boolean1
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
```

```
        },
        "reported": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "version": version-1
},
"current": {
    "state": {
        "desired": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        },
        "reported": {
            "attribute1": integer2,
            "attribute2": "string2",
            ...
            "attributeN": boolean2
        }
    },
    "metadata": {
        "desired": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        },
```

```
        "reported": {
            "attribute1": {
                "timestamp": timestamp
            },
            "attribute2": {
                "timestamp": timestamp
            },
            ...
            "attributeN": {
                "timestamp": timestamp
            }
        }
    },
    "version": version
  },
  "timestamp": timestamp,
  "clientToken": "token"
}
```

响应状态文档属性

- previous — 成功更新后，包含对象在更新之前的 state。
- current — 成功更新后，包含对象在更新之后的 state。
- state
  - reported — 只有在事物报告了 reported 部分中的任何数据时才存在，并且仅包含请求状态文档中的字段。
  - desired — 只有在设备报告了 desired 部分中的任何数据时才存在，并且仅包含请求状态文档中的字段。
  - delta — 只有在 desired 数据与影子的当前 reported 数据不同时才存在。
- metadata — 包含 desired 和 reported 部分中每个属性的时间戳，因此，您可以确定状态的更新时间。
- timestamp— 生成响应的 Epoch 日期和时间。 AWS IoT
- clientToken — 只有在向 /update 主题发布有效 JSON 时使用了客户端令牌时才存在。
- version – AWS IoT 中共享的设备影子文档的当前版本。它将在文档先前版本号的基础上增加一个数。

错误响应文档

错误响应文档具有以下格式：

```
{
    "code": error-code,
    "message": "error-message",
    "timestamp": timestamp,
    "clientToken": "token"
}
```

- code — 用于表明错误类型的 HTTP 响应代码。
- message — 用于提供额外信息的文本消息。
- timestamp— 生成响应的日期和时间 AWS IoT。此属性并非在所有错误响应文档中都存在。
- clientToken — 只有在发布的消息中使用了客户端令牌时才存在。

有关更多信息，请参阅Device Shadow 错误消息。

影子名称列表响应文档

影子名称列表响应文档具有以下格式：

```
{
    "results": [
        "shadowName-1",
        "shadowName-2",
        "shadowName-3",
        "shadowName-n"
    ],
    "nextToken": "nextToken",
    "timestamp": timestamp
}
```

- results — 影子名称数组。
- nextToken — 在获取序列中的下一页的分页请求中使用的令牌值。在没有其它要返回的影子名称时，该属性不存在。
- timestamp— 生成响应的日期和时间 AWS IoT。

文档属性

设备的影子文档具有以下属性：

`state`

`desired`

设备的所需状态。应用程序可以写入到文档的该部分以直接更新设备的状态，而无需连接到设备。

`reported`

设备的报告状态。设备写入到文档的该部分以报告其新状态。应用程序读取文档的该部分以确定设备的上次报告状态。

`metadata`

有关存储在文档 `state` 部分的数据的信息，其中包括 `state` 部分中每个属性的时间戳（以 Epoch 时间表示），让您能够确定它们的更新时间。

> **Note**
>
> 元数据不会影响服务限制或定价的文档大小。有关更多信息，请参阅 AWS IoT Service Limits。

`timestamp`

表示消息是由何时发送的 AWS IoT。通过使用消息中的时间戳以及 `desired` 或 `reported` 部分中各个属性的时间戳，设备可以确定属性的存在时间，即使设备没有内部时钟。

`clientToken`

特定于设备的字符串，让您能在 MQTT 环境中将响应与请求关联起来。

`version`

文档版本。文档每次更新时，此版本号都会递增。用于确保正在更新的文档为最新版本。

有关更多信息，请参阅影子文档示例。

增量状态

增量状态是一种虚拟类型的状态，包含 desired 状态和 reported 状态之间的差异。对于 desired 部分中的字段，如果 reported 部分没有这些字段，则它们将包含在增量中。对于 reported 部分中的字段，如果 desired 部分没有这些字段，则它们不会包含在增量中。增量包含元数据，且其值与 desired 字段的元数据相同。例如：

```
{
  "state": {
    "desired": {
      "color": "RED",
      "state": "STOP"
    },
    "reported": {
      "color": "GREEN",
      "engine": "ON"
    },
    "delta": {
      "color": "RED",
      "state": "STOP"
    }
  },
  "metadata": {
    "desired": {
      "color": {
        "timestamp": 12345
      },
      "state": {
        "timestamp": 12345
      }
      },
      "reported": {
        "color": {
          "timestamp": 12345
        },
        "engine": {
          "timestamp": 12345
        }
      },
      "delta": {
        "color": {
          "timestamp": 12345
        },
        "state": {
          "timestamp": 12345
```

```
        }
      }
    },
    "version": 17,
    "timestamp": 123456789
  }
}
```

当嵌套对象不同时，增量将包含根路径。

```
{
  "state": {
    "desired": {
      "lights": {
        "color": {
          "r": 255,
          "g": 255,
          "b": 255
        }
      }
    },
    "reported": {
      "lights": {
        "color": {
          "r": 255,
          "g": 0,
          "b": 255
        }
      }
    },
    "delta": {
      "lights": {
        "color": {
          "g": 255
        }
      }
    }
  },
  "version": 18,
  "timestamp": 123456789
}
```

Device Shadow 服务通过循环访问 desired 状态中的每个字段并将其与 reported 状态进行比较来计算增量。

数组的处理方式与值类似。如果 desired 部分中的数组与 reported 部分中的数组不匹配，则整个预期数组将被复制到增量中。

对影子文档进行版本控制

Device Shadow 服务支持在每个更新消息（请求和响应）中进行版本控制。这意味着，每次更新影子时，JSON 文档的版本将会增加。这可以确保两件事情：

- 如果客户端尝试使用旧版本号覆盖影子，它将收到错误消息。客户端将被告知必须先进行重新同步，然后才能更新设备的影子。
- 如果消息的版本比客户端存储的版本低，客户端则可决定不对该消息执行操作。

客户端可以在影子文档中不包含版本以绕过版本匹配。

影子文档中的客户端令牌

收发基于 MQTT 的消息时，您可以使用客户端令牌，以验证请求和请求响应是否包含相同的客户端令牌。这可以确保响应与请求相互关联。

> **Note**
>
> 客户端令牌不能超过 64 字节。超过 64 字节的客户端令牌将导致 400（错误请求）响应和 Invalid clientToken (无效的客户端令牌) 错误消息。

空影子文档属性

如果影子文档中的 reported 和 desired 属性不适用于当前影子状态，它们可能为空或省略。例如，只有在影子文档具有所需状态时，它才包含 desired 属性。以下是没有 desired 属性的状态文档的有效示例：

```
{
    "reported" : { "temp": 55 }
}
```

reported 属性也可能为空，例如，设备尚未更新影子时：

```
{
    "desired" : { "color" : "RED" }
}
```

如果更新导致 desired 或 reported 属性变为 null，则会将其从文档中删除。下面说明了如何将 desired 属性设置为 null 以删除该属性。例如，在设备更新其状态时，您可以执行该操作。

```
{
    "state": {
        "reported": {
            "color": "red"
        },
        "desired": null
    }
}
```

影子文档也可能没有 desired 或 reported 属性，从而使影子文档为空。这是一个空影子（但有效）文档的示例。

```
{
}
```

影子文档中的数组值

影子支持数组，但将其视为正常值进行处理，因为对数组的更新将替换整个数组。无法更新数组的某个部分。

初始状态：

```
{
    "desired" : { "colors" : ["RED", "GREEN", "BLUE" ] }
}
```

更新：

```
{
    "desired" : { "colors" : ["RED"] }
}
```

最终状态：

```
{
    "desired" : { "colors" : ["RED"] }
}
```

数组不能包含空值。例如，以下数组无效并将被拒绝。

```
{
    "desired" : {
        "colors" : [ null, "RED", "GREEN" ]
    }
}
```

Device Shadow 错误消息

Device Shadow 服务在尝试更改状态文档失败时向错误主题发布消息 (通过 MQTT)。此消息仅将作为对发布到其中一个保留的 $aws 主题的请求的响应。如果客户端使用 REST API 来更新文档，则客户端将收到作为响应一部分的 HTTP 错误代码，且不会发送任何 MQTT 错误消息。

HTTP 错误代码	错误消息
400（错误请求）	• JSON 无效 • 必需节点缺失：状态 • 状态节点必须是对象 • 预期节点必须是对象 • 报告节点必须是对象 • 版本无效 • clientToken 无效 **Note** 超过 64 字节的客户端令牌将引发此响应。 • JSON 包含的嵌套层级过多；最多嵌套 6 个层级 • 状态包含无效节点
401（未授权）	• Unauthorized

HTTP 错误代码	错误消息
403（禁止访问）	• 禁止
404（未找到）	• 事物未找到 • 不存在名为 *shadowName* 的影子
409（冲突）	• 版本冲突
413（负载过大）	• 负载超出允许的最大值
415（媒体类型不受支持）	• 文档编码不受支持；受支持的编码是 UTF-8
429（请求过多）	• 如果单一连接中正在传输的请求超过 10 个，则 Device Shadow 服务将生成此条错误消息。动态请求是已启动但尚未完成的进行中的请求。
500（内部服务器错误）	• 内部服务故障

任务

使用 AWS IoT Jobs 定义一组远程操作，这些操作可以发送到一个或多个连接到的设备并在这些设备上运行 AWS IoT。例如，您可以定义一个任务，该任务指示一组设备下载并安装应用程序或固件更新、重启、轮换证书或执行远程故障排除操作。

访问 AWS IoT 作业

您可以使用控制台或 AWS IoT Core API 开始使用 AWS IoT Jobs。

使用 控制台

登录 AWS Management Console，然后转到 AWS IoT 控制台。在导航窗格中，选择 Manage（管理），然后选择 Jobs（任务）。您可以从此部分创建和管理任务。如果要创建和管理任务模板，请在导航窗格中，选择 Job templates（任务模板）。有关更多信息，请参阅使用 AWS Management Console 创建和管理任务。。

使用 API 或 CLI

您可以使用 AWS IoT Core API 操作开始使用。有关更多信息，请参阅 AWS IoT API 参考。 AWS SDK 支持构建 AWS IoT 作业所依据的 AWS IoT Core API。有关更多信息，请参阅 AWS SDK 和工具包。

您可以使用运行命令 AWS CLI 来创建和管理作业和作业模板。有关更多信息，请参阅 AWS IoT CLI 参考。

AWS IoT 作业、区域和终端节点

AWS IoT Jobs 支持特定于您的控制平面和数据平面 API 端点 AWS 区域。数据平面 API 端点特定于您的 AWS 账户 和 AWS 区域。有关 AWS IoT 任务端点的更多信息，请参阅《AWS 一般参考》中的 AWS IoT Device Management -作业数据端点。

什么是远程操作？

远程操作是指您可以在物理设备、虚拟设备或端点上执行的任何更新或操作，无需操作员或技术人员亲自到场即可远程完成。远程操作是使用 over-the-air （OTA）更新执行的，因此您的设备不必实际存在。在中管理您的设备群 AWS Cloud 允许您在注册设备后对设备执行远程操作 AWS IoT Core。

AWS IoT Device Management Jobs 提供了一种可扩展的方法，用于在注册的设备上执行远程操作 AWS IoT Core。在中创建任务 AWS Cloud 并通过 MQTT 或 HTTP 协议通过 OTA 更新推送到所有目标设备。

AWS IoT Device Management 作业使您能够以安全、可扩展且更具成本效益的方式执行远程操作，例如恢复出厂设置、设备重启和软件 OTA 更新。

有关的更多信息 AWS IoT Core，请参阅什么是 AWS IoT？。

有关 AWS IoT Device Management 任务的更多信息，请参阅什么是 AWS IoT 乔布斯？。

使用 AWS IoT Device Management Jobs 进行远程操作的好处

使用 AWS IoT Device Management Jobs 执行远程操作可以简化设备群的管理。以下列表重点介绍了使用 AWS IoT Device Management Jobs 执行远程操作的一些主要好处：

- 与其他产品无缝集成 AWS 服务
  - AWS IoT Device Management Jobs 与以下增值 AWS 服务 和功能紧密结合：
    - Amazon S3：将您的远程操作说明存储在安全的 Amazon S3 存储桶中，您可以在其中控制该内容的访问权限。使用 Amazon S3 存储桶提供了一种可扩展且耐用的存储解决方案，该解决方案与 AWS IoT 设备管理软件包目录原生集成，允许 AWS IoT Device Management 任务在更新说明中参考和替换。有关更多信息，请参阅什么是 Amazon S3？ 。
    - Amazon CloudWatch：监控和记录每台设备任务执行的远程操作实施状态以及其他设备活动，以跟踪和分析任务中的 AWS IoT Device Management 整体作业绩效。有关更多信息，请参阅什么是亚马逊 CloudWatch？ 监控作业日志并捕获历史数据以进行故障排除。它如何处理作业。
    - AWS IoT 设备影子服务: 使用 AWS IoT Device Management Jobs 通过设备影子维护 AWS IoT 事物的数字化呈现，这样无论设备连接如何，应用程序和其他服务都可以访问设备的状态。有关更多信息，请参阅AWS IoT 设备影子服务。
    - 用于 AWS IoT 设备管理的 Fleet H ub：构建独立的 Web 应用程序，用于监控设备群的运行状况。有关更多信息，请参阅什么是用于 AWS IoT 设备管理的 Fleet Hub？ 。
- 安全最佳实践
  - 权限控制：使用 Amazon S3 控制对远程操作说明的访问权限，并确定哪些 IAM 用户可以使用 AWS IoT 策略和 IAM 用户角色将您的远程操作说明部署到您的设备群中。
    - 有关 AWS IoT 策略的更多信息，请参阅创建 AWS IoT 策略。
    - 有关 IAM 用户角色的更多信息，请参阅的身份和访问管理 AWS IoT。
- 可扩展性

  - 定向任务部署：使用创建任务时在任务文档中输入的特定设备分组标准，控制哪些设备从具有目标作业部署的作业中接收作业文档。通过为每台设备创建 AWS IoT 事物并将该信息存储在 AWS IoT 注册表中，您可以使用舰队索引进行有针对性的搜索。您可以根据队列索引搜索结果创建自定义群组，以支持您的目标任务部署。有关更多信息，请参阅使用管理设备 AWS IoT。使用作业来做快照而不是连续作业。
  - 任务状态：除了每台设备上任务文档的单独实施状态外，还可跟踪设备群中的任务文档发布状态以及设备队列级别的总体任务状态。有关更多信息，请参阅任务和任务执行状态。
  - 新的设备可扩展性：通过连续作业，将任务文档添加到使用队列索引创建的现有自定义群组中，轻松将其部署到新设备上。这将为您节省时间，而不必将作业文档单独部署到每台新设备上。或者，您可以使用更具针对性的方法来拍摄快照，方法是将作业文档部署到预先确定的设备组一次，然后作业就完成了。
- 弹性
  - 作业配置：使用可选的作业配置部署、调度、中止、超时和重试来自定义您的作业和作业文档，以满足您的特定需求。有关更多信息，请参阅任务配置。
- 具有成本效益
  - 利用 AWS IoT Device Management Jobs 部署关键更新和执行例行维护任务，引入更高效的成本结构来维护您的设备群。维护设备群的 do-it-yourself （DIY）解决方案包括经常性的、可变的成本，例如托管和管理 DIY 解决方案所需的基础架构、开发、维护和扩展 DIY 解决方案的人力成本以及数据传输成本。利用 AWS IoT Device Management 任务的透明、固定成本结构，除了便于向设备群发布任务文档和跟踪每台设备的任务执行状态所需的数据传输成本外，您还可以确切地知道设备每次执行任务的成本。有关更多信息，请参阅AWS IoT Core 定价。

什么是 AWS IoT 乔布斯？

使用 AWS IoT Jobs 定义一组远程操作，这些操作可以发送到一个或多个连接到的设备并在这些设备上运行 AWS IoT。

要创建任务，首先定义任务文档，其中包含描述设备必须远程执行的操作的说明列表。要执行这些操作，请指定目标列表，这些都是单独事物、事物组，或同时选择两者。任务文件和目标共同构成了部署。

每个部署都可以有额外的配置：

- Rollout（推出）：此配置定义了每分钟有多少设备接收任务文档。
- 中止：如果一定数量的设备没有收到任务通知，则可使用此配置取消任务。这样可以避免向整个实例集发送错误的更新。

- Timeout（超时）：如果在一定时间内没有从您的工作目标收到响应，任务可能会失败。您可以跟踪在这些设备上运行的任务。
- 重试：如果设备报告失败或作业超时，你可以使用 J AWS IoT obs 自动将任务文档重新发送到设备。
- Scheduling（计划）：此配置使您能够为未来的日期和时间安排任务。它还允许您创建定期维护时段，以便在预定义的低流量时段内更新设备。

AWS IoT 作业会发送一条消息，通知目标有任务可用。目标通过下载任务文档、执行其指定的操作并向其报告其进度来开始执行作业 AWS IoT。您可以通过运行作业提供的命令来跟踪特定目标或所有目标的 AWS IoT 任务进度。任务开始时，状态为正在进行中。然后，设备在显示此状态的同时报告增量更新，直到任务成功、失败或超时。

以下主题描述任务的一些关键概念以及任务和任务执行的生命周期。

主题

任务关键概念

以下概念提供了有关 AWS IoT 作业以及如何创建和部署作业以在设备上运行远程操作的详细信息。

基本概念

以下是您在使用 AWS IoT Jobs 时必须了解的基本概念。

任务

任务是被发送到一台或多台连接到 AWS IoT 的设备并在这些设备上运行的远程操作。例如，您可以定义一个任务，该任务指示一组设备下载并安装应用程序或运行固件更新、重启、轮换证书或执行远程故障排除操作。

任务文档

要创建任务，您必须先创建一个任务文档，该文档是要由设备执行的远程操作的描述。

任务文档是 UTF-8 编码的 JSON 文档，其中包含您的设备执行任务所需的信息。任务文档包含一个或多个 URL，设备可从这些 URL 下载更新或其它数据。任务文档可存储在 Amazon S3 存储桶中，或者随用于创建任务的命令内联包含。

> **Tip**
>
> 有关作业文档示例，请参阅软件 AWS IoT 开发工具包中的 jobs-agent.js 示例 JavaScript。

目标

在创建任务时，您需要指定一系列目标，它们是应执行操作的设备。目标可以是事物和/或事物组。AWS IoT 作业服务向每个目标发送一条消息，告知其有任务可用。

部署

通过提供任务文档并指定目标列表创建任务之后，系统会将任务文档部署到要为其执行更新的远程目标设备上。对于快照任务，任务会在部署到目标设备后完成。对于持续任务，任务会在设备添加到组时部署到该组设备。

任务执行

任务执行是目标设备上的任务的实例。目标通过下载任务文档来开始执行任务。然后，它执行文档中指定的操作，并将其进度报告给 AWS IoT。执行编号是特定目标上的任务执行的唯一标识符。AWS IoT 作业服务提供命令来跟踪目标上作业的执行进度以及所有目标上的作业进度。

任务类型概念

以下概念可以帮助您更多地了解可以使用 Jobs 创建的不同类型的 AWS IoT 作业。

快照任务

默认情况下，任务将发送到您在创建该任务时指定的所有目标。在这些目标完成任务 (或报告它们无法执行此操作) 后，任务即完成。

持续任务

持续的任务将发送到您在创建该任务时指定的所有目标。它继续运行并发送到添加到目标组的任何新设备（事物）。例如，在将设备添加到组时，可使用持续任务加入或升级设备。您可以通过在创建作业时设置可选参数，来使作业成为持续作业。

> **Note**
>
> 当使用动态事物组确定目标 IoT 机群时，我们建议您使用连续任务而不是快照任务。通过使用连续任务，加入组的设备即使在任务创建之后也会收到任务执行。

预签名 URL

为了安全、有时间限制地访问任务文档中未包含的数据，可以使用预签名的Amazon S3 URL。将数据放置在 Amazon S3 存储桶中，并添加一个指向任务文档中的数据的占位符链接。当 AWS IoT Jobs 收到任务文档请求时，它会通过查找占位符链接来解析任务文档，然后用预签名的 Amazon S3 网址替换这些链接。

占位符链接采用以下格式：

```
${aws:iot:s3-presigned-url:https://s3.amazonaws.com/bucket/key}
```

其中 *bucket* 是您的存储桶名称，*key* 是您链接到的存储桶中的对象。

在北京和宁夏区域中，仅当资源拥有者具有 ICP（互联网内容提供商）许可证时，预签名 URL 才起作用。有关更多信息，请参阅《中国服务入门》文档中的 Amazon 简单存储 AWS 服务。

任务配置概念

以下概念可以帮助您了解如何配置任务。

推出

您可以指定向目标发送待处理任务执行通知的速度。这样，您就可以创建一个分段推出以更好地管理更新、重启和其它操作。您可以使用静态推出率或指数级推出率来创建部署配置。您可以使用静态推出速率来指定每分钟可通知的最大任务目标数。

有关设置推出速率的示例，以及有关配置任务推出的更多信息，请参阅任务推出、计划和中止配置。

计划

通过任务计划，您可以计划向目标组中的所有设备推出连续任务和快照任务的任务文档的时间。此外，您可以创建一个可选的维护时段，其中包含任务向目标组中的所有设备推出任务文档的特定日期和时间。维护时段是一种周期性实例，其频率为每天、每周、每月或在初始任务或任务模板创建期间选择的自定义日期和时间。只能将连续任务安排为在维护时段内执行推出。

任务计划是特定于您的任务的。无法安排单个任务执行。有关更多信息，请参阅任务推出、计划和中止配置。

中止

您可以创建一组条件在满足您指定的特定条件时中止推出。有关更多信息，请参阅任务推出、计划和中止配置。

超时

只要任务部署卡在 IN_PROGRESS 状态的时间超出预期，您就会收到任务超时通知。有两种类型的计时器：进行中计时器和步骤计时器。当任务为 IN_PROGRESS 时，您可以监控和跟踪任务部署的进度。

推出和中止配置特定于您的任务，而超时配置特定于任务部署。有关更多信息，请参阅任务执行超时和重试配置。

重试

通过任务重试，可以在任务失败和/或超时时重试任务执行。您最多可以重试 10 次任务执行。您可以监控和跟踪重试的进度以及任务执行是否成功。

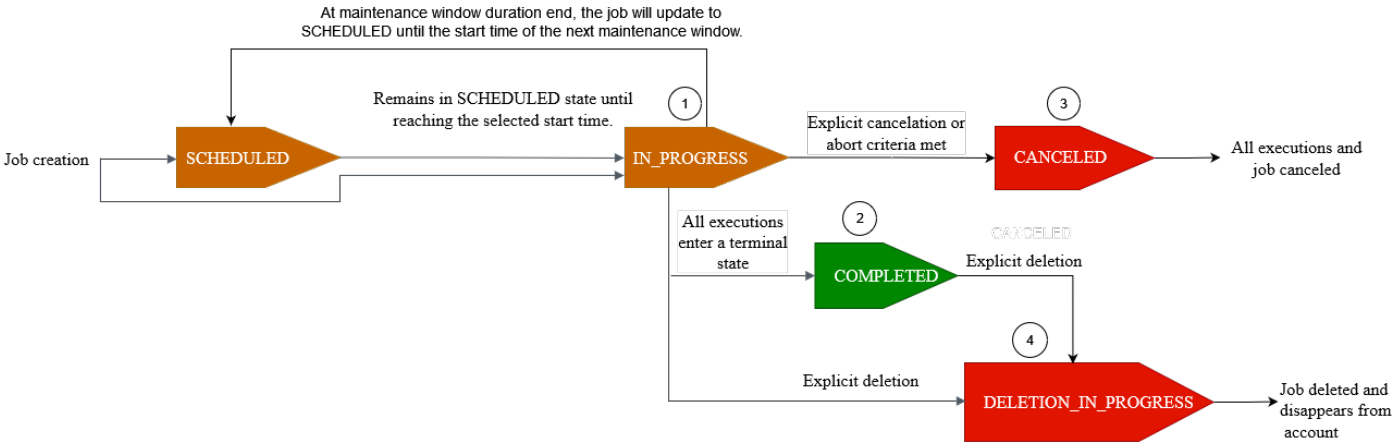
推出和中止配置特定于您的任务，而超时和重试配置特定于任务执行。有关更多信息，请参阅任务执行超时和重试配置。

任务和任务执行状态

以下各节描述了 AWS IoT 任务的生命周期和任务执行的生命周期。

作业状态

下图显示了 AWS IoT 作业的不同状态。



您使用 J AWS IoT obs 创建的任务可能处于以下状态之一：

- SCHEDULED

  在使用 AWS IoT 控制台、CreateJobAPI 或 API 创建初始任务或任务模板期间，您可以在 AWS IoT 控制台或 CreateJobTemplateAP CreateJobI 或 CreateJobTemplateAPI 中选择可

选SchedulingConfig的调度配置。当您启动包含特定 startTime、endTime 和 endBehavoir 的计划任务时，任务状态将更新为 SCHEDULED。当任务达到您选择的 startTime 或下一个维护时段的 startTime 时（如果您在维护时段中选择了任务推出），状态将从 SCHEDULED 更新为 IN_PROGRESS，并开始将任务文档推出到目标组中的所有设备。

- 进行中

  当您使用 AWS IoT 控制台或 CreateJobAPI 创建任务时，任务状态会更新为IN_PROGRESS。在任务创建期间， AWS IoT Jobs 开始向目标组中的设备推出任务执行。在所有任务执行都推出后， AWS IoT Jobs 就会等待设备完成远程操作。

  有关适用于进行中任务的并发和限制的信息，请参阅 任务限制。

  > **Note**
  >
  > 当 IN_PROGRESS 任务达到当前维护时段结束时，任务文档的推出将停止。该任务将更新为 SCHEDULED，直至下一个维护时段的 startTime。

- COMPLETED

  连续任务通过以下方式之一进行处理：

  - 对于未 选择可选计划配置的连续任务，此任务始终处于进行中，并继续针对添加到目标组的任何新设备运行。它从不会达到 COMPLETED 状态。
  - 对于选择 了可选计划配置的连续任务，以下表述是正确的：
    - 如果提供了 endTime，则连续任务将在 endTime 已过且所有任务执行都达到最终状态时达到 COMPLETED 状态。
    - 如果在可选计划配置中未提供 endTime，则连续任务将继续执行任务文档推出。

  对于快照任务，当其所有任务执行都进入最终状态（如 SUCCEEDED、FAILED、TIMED_OUT、REMOVED 或 CANCELED）时，任务状态更改为 COMPLETED。

- CANCELED

  当您使用 AWS IoT 控制台、CancelJobAPI 或取消任务时 任务中止配置，任务状态将更改为CANCELED。在任务取消期间， AWS IoT 作业开始取消先前创建的任务执行。

  有关适用于正在取消的任务的并发和限制的信息，请参阅 任务限制。

- DELETION_IN_PROGRESS

当您使用 AWS IoT 控制台或 DeleteJobAPI 删除任务时，任务状态会更改为DELETION_IN_PROGRESS。在删除任务期间， AWS IoT 作业开始删除先前创建的任务执行。删除所有任务执行后，该任务将从您的 AWS 账户中消失。

任务执行状态

下表显示了 AWS IoT 任务执行的不同状态，以及状态更改是由设备启动的，还是由 AWS IoT 任务启动的。

任务执行状态和源

任务执行状态	由设备发起？	由 AWS IoT 乔布斯发起？	终端状态？	是否可以重试？
QUEUED	否	是	否	不适用
IN_PROGRESS	是	否	否	不适用
SUCCEEDED	是	否	是	不适用
FAILED	是	否	是	是
TIMED_OUT	否	是	是	是
REJECTED	是	否	是	否
REMOVED	否	是	是	否
CANCELED	否	是	是	否

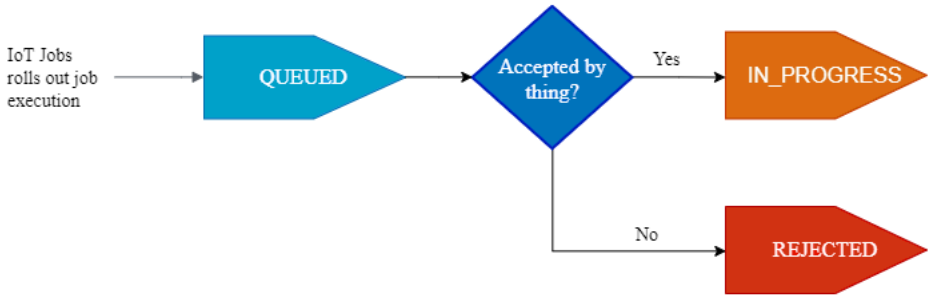
以下部分详细介绍了使用 AWS IoT 作业创建任务时推出的任务执行状态。

- QUEUED

  当 AWS IoT Jobs 为目标设备推出任务执行时，任务执行状态将设置为QUEUED。任务执行保留为 QUEUED 状态，直到：

  - 您的设备收到任务执行，调用任务 API 操作并将状态报告为 IN_PROGRESS。
  - 您取消任务或任务执行，或者当满足您指定的中止条件时，状态更改为 CANCELED。

- 您的设备已从目标组中移除且状态更改为 REMOVED。



- 进行中

  如果您的 IoT 设备订阅了预留任务主题$notify和$notify-next，并且您的设备调用了 StartNextPendingJobExecution API 或 API，状态为IN_PROGRESS，则 AWS IoT Jobs 会将任务执行状态设置为。UpdateJobExecution IN_PROGRESS

  UpdateJobExecution API 可以多次调用，状态为 IN_PROGRESS。您可以使用 statusDetails 对象指定有关执行步骤的其他详细信息。

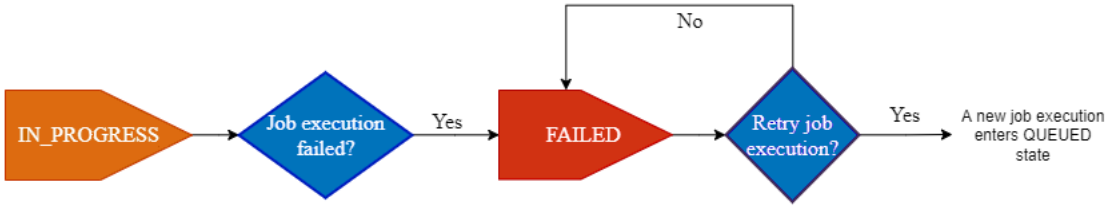
  > **Note**
  >
  > 如果您为每台设备创建多个任务，则 AWS IoT Jobs 和 MQTT 协议无法保证交付顺序。

- SUCCEEDED

  当您的设备成功完成远程操作时，设备必须调用状态为的 UpdateJobExecution API，SUCCEEDED以表示任务执行成功。 AWS IoT 然后，作业会更新任务执行状态并将其返回为SUCCEEDED。

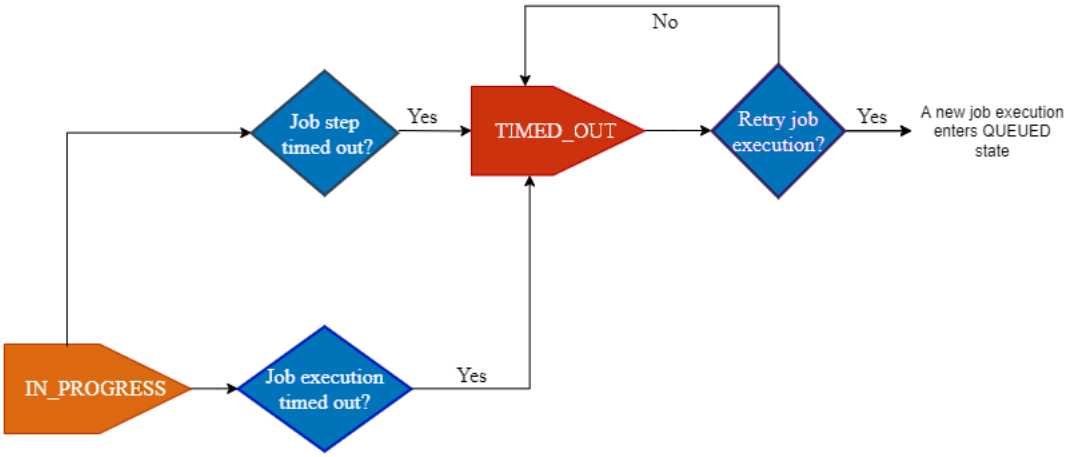
- FAILED

  当您的设备无法完成远程操作时，设备必须调用状态为UpdateJobExecution的 API，Failed以表明任务执行失败。 AWS IoT 然后，作业会更新任务执行状态并将其返回为Failed。您可以使用任务执行重试配置重试设备的这一任务执行。



- TIMED_OUT

  当您的设备在状态为时未能完成任务步骤IN_PROGRESS，或者在正在进行的计时器的超时时间内未能完成远程操作时，J AWS IoT obs 会将任务执行状态设置为TIMED_OUT。您还有一个用于正在进行的任务的每个任务步骤的步骤计时器，并且仅适用于任务执行。正在进行的计时器持续时间是使用任务执行超时配置 的 inProgressTimeoutInMinutes 属性指定的。您可以使用任务执行重试配置重试设备的这一任务执行。



- REJECTED

  当您的设备收到无效或不兼容的请求时，设备必须调用状态为UpdateJobExecution的 API REJECTED。 AWS IoT 然后，作业会更新任务执行状态并将其返回为REJECTED。

- REMOVED

  当设备不再是任务执行的有效目标时，例如当它脱离动态事物组时， AWS IoT Jobs 将任务执行状态设置为 REMOVED。您可以将事物重新附加到目标组，然后重新启动设备的任务执行。

- CANCELED

  当您使用控制台或或 CancelJobExecution API 取消任务或取消任务执行时，CancelJob或者当满足使用指定的中止条件时，J AWS IoT obs 会取消该任务并将任务执行状态设置 任务中止配置为。CANCELED

管理任务

使用任务通知设备软件或固件更新。您可以使用 AWS IoT 控制台、任务管理和控制 API 操作AWS Command Line Interface 或 AWS SDK 来创建和管理任务。

任务的代码签名

向设备发送代码时，为了让设备检测代码是否在传输过程中被修改，建议您使用 AWS CLI 对代码文件进行签名。有关说明，请参阅使用 AWS CLI 创建和管理任务。

有关更多信息，请参阅什么是 AWS IoT 的代码签名？

任务文档

在创建任务之前，您必须创建任务文档。如果您使用 AWS IoT 的代码签名，则必须将任务文档上载到受版本控制的 Amazon S3 存储桶。有关创建 Amazon S3 存储桶并将向其上载文件的信息，请参阅 Amazon S3 入门指南中的 Amazon Simple Storage Service 入门。

> **Tip**
>
> 有关任务文档示例，请参阅 AWS IoT SDK for JavaScript 中的 jobs-agent.js 示例。

预签名 URL

您的任务文档可以包含指向您的代码文件（或其它文件）的预签名 Amazon S3 URL。预签名 Amazon S3 URL 的有效期有限，因此在设备请求任务文档时才生成。因为在您创建任务文档时尚未创建预签名 URL，所以在您的任务文档中使用占位符 URL。占位符 URL 类似如下所示：

```
${aws:iot:s3-presigned-url:https://s3.region.amazonaws.com/<bucket>/<code file>}
```

其中：

- *###*是包含代码文件的 Amazon S3 存储桶。
- *####*是代码文件的 Amazon S3 密钥。

当设备请求任务文档时，AWS IoT 会生成预签名 URL 并使用预签名 URL 替换占位符 URL。然后将您的任务文档发送到设备。

IAM 角色，用于授予从 S3 下载文件的权限

当您创建使用预签名 Amazon S3 URL 的任务时，必须提供 IAM 角色。该角色必须授予从存储数据或更新的 Amazon S3 桶下载文件的权限。该角色还必须向 AWS IoT 授予代入角色的权限。

您可以为预签名 URL 指定可选的超时。有关更多信息，请参阅 CreateJob。

向 AWS IoT Jobs 授予代入您的角色的权限

1. 转到 IAM 控制台的角色中心，然后选择您的角色。
2. 在信任关系选项卡上，选择 Edit Trust Relationship（编辑信任关系），用以下 JSON 替换当前策略文档。选择 Update Trust Policy（更新信任策略）。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "iot.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. 为防止出现混淆代理人问题，请在策略中添加全局条件键 aws:SourceArn 和 aws:SourceAccount。

> **Important**
>
> 您的 aws:SourceArn 必须符合此格式：arn:aws:iot:*region*:*account-id*:*。确保##与您的 AWS IoT 区域匹配，*account-id*与您的客户账户 ID 相匹配。有关更多信息，请参阅防止跨服务混淆代理。

```
{
  "Effect": "Allow",
```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service":
          "iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole",
      "Condition": {
         "StringEquals": {
            "aws:SourceAccount": "123456789012"
         },
         "ArnLike": {
              "aws:SourceArn": "arn:aws:iot:*:123456789012:job/*"
         }
       }
     }
   ]
}
```

4. 如果您的任务使用的任务文档是一个 Amazon S3 对象，请选择权限，然后使用以下 JSON。这将添加一个策略，以授予从您的 Amazon S3 桶下载文件的权限：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::your_S3_bucket/*"
        }
    ]
}
```

主题

- 使用 AWS Management Console 创建和管理任务。
- 使用 AWS CLI 创建和管理任务。

使用 AWS Management Console 创建和管理任务。

创建 任务

1. 登录 AWS Management Console，然后登录 AWS IoT 控制台。
2. 在左侧导航窗格的“管理”部分下，选择远程操作，然后选择任务。
3. 在 Jobs（任务）对话框的 Jobs（任务）页面上，选择 Create job（创建任务）。
4. 根据您使用的设备，您可以创建自定义任务、FreeRTOS OTA 更新任务或 AWS IoT Greengrass 任务。在此示例中，选择 Create a custom job（创建自定义任务）。选择 Next（下一步）。
5. 在 Job properties（任务属性）对话框的 Custom job properties（自定义任务属性）页面上，为以下字段输入您的信息：

   - Name（名称）：输入唯一的字母数字任务名称。
   - Description - optional（描述 - 可选）：输入有关您的任务的可选描述。
   - Tags - optional（标签 - 可选）：

   > **Note**
   >
   > 我们建议不要在您的任务 ID 或描述中使用个人身份信息。

   选择 Next（下一步）。
6. 在 Job targets（任务目标）对话框的 File configuration（文件配置）页面上，选择要运行此任务的 Things（事务）或 Thing groups（事务组）。

   在 Job document（任务文档）对话框中，选择以下选项之一：

   - From file（源文件）：您之前上传到 Amazon S3 存储桶的 JSON 任务文件
     - 代码签名

       在位于 Amazon S3 URL 的任务文档中，需要 `${aws:iot:code-sign-signature:s3://region.bucket/code-file@code-file-version-id}` 作为占位符，直到使用您的代码签名配置文件将其替换为已签名的代码文件路径。新的签名代码文件最初将出现在 Amazon S3 源桶的 `SignedImages` 文件夹中。将创建一个包含 `Codesigned_` 前缀的新任务文档，其签名代码文件路径将替换代码签名占位符，并放入您的 Amazon S3 URL 中以创建新任务。

- 预签名资源 URL

  在预签名角色下拉列表中，选择您在预签名 URL 中创建的 IAM 角色。对于从 Amazon S3 下载对象的设备来说，使用 `${aws:iot:s3-presigned-url:` 对位于 Amazon S3 中的对象的 URL 进行预签名是一种最佳安全实践。

  如果要使用预签名 URL 作为代码签名占位符，请使用以下示例模板：

  ```
  ${aws:iot:s3-presigned-url:${aws:iot:code-sign-signature:<S3 URL>}
  ```

- From template（源模板）：包含任务文档和任务配置的任务模板。任务模板可以是您创建的自定义任务模板或 AWS 管理的模板。

  如果您正在创建任务执行常用的远程操作（例如重启设备），可以使用 AWS 托管模板。这些模板已经进行了预置以供使用。有关更多信息，请参阅 创建自定义任务模板 和 从托管模板创建自定义任务模板：

7. 在 Job configuration（任务配置）对话框的 Job configuration（任务配置）页面上，选择以下任务类型之一：

   - 快照任务：快照任务在目标设备和组上完成其运行后即完成。
   - 连续任务：连续任务适用于事物组，并会在以后添加到指定目标组的任何设备上运行。

8. 在 Additional configurations - optional（其他配置 - 可选）对话框中，查看以下可选任务配置并做出相应的选择：

   - 推出配置
   - 计划配置
   - 任务执行超时配置
   - 任务执行重试配置 - 新增
   - 中止配置

   有关任务配置的更多信息，请参阅以下部分：

   - 任务推出、计划和中止配置
   - 任务执行超时和重试配置

   查看您的所有任务选择，然后选择 Submit（提交）以创建任务。

在您创建任务后，控制台会生成一个 JSON 签名并将其放在您的任务文档中。您可以使用 AWS IoT 控制台查看状态、取消或删除任务。要管理任务，请转到控制台的 Job 中心。

使用 AWS CLI 创建和管理任务。

本部分介绍如何创建和管理任务。

创建任务

要创建 AWS IoT 任务，使用 CreateJob 命令。任务将进行排队以便在您指定的目标 (事物或事物组) 上执行。要创建 AWS IoT 任务，您需要一个任务文档，该文档可包含在请求的正文中或作为指向 Amazon S3 文档的链接。如果任务包含使用预签名的 Amazon S3 URL 下载文件，则您需要一个 IAM 角色 Amazon Resource Name (ARN)，它具有下载文件的权限并会向 AWS IoT Jobs 服务授予代入角色的权限。

有关使用 API 命令或 AWS CLI 输入日期和时间时的语法的更多信息，请参阅时间戳。

使用任务进行代码签名

如果您使用 AWS IoT 的代码签名，必须启动代码签名任务并将输出包括在您的任务文档中。这将替换任务文档中的代码签名占位符，在使用您的代码签名配置文件将其替换为已签名的代码文件路径之前，该占位符是必需的。代码签名占位符将如下所示：

```
    ${aws:iot:code-sign-signature:s3://region.bucket/code-file@code-file-version-id}
```

使用 start-signing-job 命令创建代码签名任务。`start-signing-job` 将返回任务 ID。要获取存储签名的 Amazon S3 位置，使用 describe-signing-job 命令。然后，您可以从 Amazon S3 中下载签名。有关代码签名任务的更多信息，请参阅 AWS IoT 的代码签名。

您的任务文档必须包含代码文件的预签名 URL 占位符和使用 start-signing-job 命令放置在 Amazon S3 桶中的 JSON 签名输出：

```
{
    "presign": "${aws:iot:s3-presigned-url:https://s3.region.amazonaws.com/bucket/
image}",
}
```

使用任务文档创建任务

以下命令说明了如何使用存储在 Amazon S3 存储桶 (*jobBucket*) 中的任务文档 (*job-document.json*) 和具有从 Amazon S3 下载文件的权限的角色 (*S3DownloadRole*)。

```
aws iot create-job  \
      --job-id 010  \
      --targets arn:aws:iot:us-east-1:123456789012:thing/thingOne  \
      --document-source https://s3.amazonaws.com/my-s3-bucket/job-document.json  \
      --timeout-config inProgressTimeoutInMinutes=100 \
      --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute
\": 50, \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings
\": 1000, \"numberOfSucceededThings\": 1000}}, \"maximumPerMinute\": 1000}" \
      --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
      --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

任务在 *thingOne* 上运行。

可选的 `timeout-config` 参数指定每个设备完成其任务执行所具有的时间。计时器在任务执行状态设置为 `IN_PROGRESS` 时启动。如果任务执行状态未在时间到期之前设置为其它最终状态，则会设置为 `TIMED_OUT`。

进行中计时器无法更新，将应用到该任务的全部任务执行。只要任务执行保持在 `IN_PROGRESS` 状态的时间长度超过了此间隔，会失败，并切换为最终 `TIMED_OUT` 状态。AWS IoT 还将发布 MQTT 通知。

有关创建任务推出和中止相关配置的更多信息，请参阅任务推出和中止配置。

> **Note**
>
> 在您创建任务时，系统会检索指定为 Amazon S3 文件的任务文档。如果在创建任务文档后，您更改用作任务文档源的 Amazon S3 文件的内容，则发送到任务目标的内容不会更改。

更新任务

要更新任务，使用 UpdateJob 命令。您可以更新任务的 `description`、`presignedUrlConfig`、`jobExecutionsRolloutConfig`、`abortConfig` 和 `timeoutConfig` 字段。

```
aws iot update-job  \
```

```
  --job-id 010  \
  --description "updated description" \
  --timeout-config inProgressTimeoutInMinutes=100 \
  --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute\": 50,
 \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings\": 1000,
 \"numberOfSucceededThings\": 1000}, \"maximumPerMinute\": 1000}}" \
  --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
  --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

有关更多信息，请参阅任务推出和中止配置。

取消任务

要取消任务，请使用 CancelJob 命令。取消任务将使 AWS IoT 停止为任务推出任何新的任务执行。这还将取消处于 QUEUED 状态的任何任务执行。AWS IoT 会将任何任务执行的最终状态保持不变，因为设备已完成任务。如果任务执行的状态为 IN_PROGRESS，它也将保持不变，除非您使用可选的 --force 参数。

以下命令说明如何取消 ID 为 010 的任务。

```
aws iot cancel-job --job-id 010
```

该命令将显示以下输出：

```
{
    "jobArn": "string",
    "jobId": "string",
    "description": "string"
}
```

当您取消任务时，将取消处于 QUEUED 状态的任务执行。如果您指定了可选的 --force 参数，将取消处于 IN_PROGRESS 状态的任务执行。不会取消处于最终状态的任务执行。

> **Warning**
>
> 取消处于 IN_PROGRESS 状态的任务（通过设置 --force 参数）将取消正在进行的任何任务执行，并且会导致运行该任务的设备无法更新任务执行状态。请谨慎使用，并且 确保执行已取消的任务的每个设备能够恢复到有效状态。

已取消任务或其某个任务执行的状态最终是一致的。AWS IoT 将尽快停止将该任务的新任务执行和 QUEUED 任务执行安排给设备。将任务执行的状态更改为 CANCELED 可能需要一些时间，具体取决于设备数和其它因素。

如果任务因满足 AbortConfig 对象定义的条件而被取消，服务会为 comment 和 reasonCode 字段添加自动填充的值。当任务取消由用户驱动时，您可以为 reasonCode 创建自己的值。

取消任务执行

要取消设备上的任务执行，请使用 CancelJobExecution 命令。它将取消处于 QUEUED 状态的任务执行。如果要取消正在进行的任务执行，您必须使用 --force 参数。

以下命令说明如何取消在 myThing 上运行的任务 010 的任务执行。

```
aws iot cancel-job-execution --job-id 010 --thing-name myThing
```

该命令不显示任何输出。

将取消处于 QUEUED 状态的任务执行。如果您指定了可选的 --force 参数，则将取消处于 IN_PROGRESS 状态的任务执行。无法取消处于最终状态的任务执行。

> **Warning**
>
> 取消处于 IN_PROGRESS 状态的任务执行后，设备无法更新任务执行状态。请谨慎使用，并且确保 设备能够恢复到有效状态。

如果任务执行处于最终状态，或者任务执行处于 IN_PROGRESS 状态而 --force 参数未设置为 true，则此命令将导致 InvalidStateTransitionException。

已取消任务执行的状态最终是一致的。将任务执行的状态更改为 CANCELED 可能需要一些时间，具体取决于各种因素。

删除任务

要删除任务及其任务执行，使用 DeleteJob 命令。默认情况下，您只能删除处于最终状态（SUCCEEDED 或 CANCELED）的任务。否则，会出现异常。但是，仅当 `force` 参数设置为 `true` 时，才能删除处于 IN_PROGRESS 状态的任务。

要删除任务，运行以下命令：

```
aws iot delete-job --job-id 010 --force|--no-force
```

该命令不显示任何输出。

> **Warning**
>
> 删除处于 IN_PROGRESS 状态的任务后，部署该任务的设备将无法访问任务信息或更新任务执行状态。请谨慎使用，并且确保部署已删除的任务的每个设备能够恢复到有效状态。

删除任务可能需要一些时间，具体取决于为该任务创建的任务执行的数量以及其它因素。在删除任务过程中，任务的状态将显示为 DELETION_IN_PROGRESS。如果您尝试删除或取消其状态已为 DELETION_IN_PROGRESS 的任务，将导致错误。

只能有 10 个任务的状态可以同时为 DELETION_IN_PROGRESS。否则，会出现 `LimitExceededException`。

获取任务文档

要检索任务的任务文档，使用 GetJobDocument 命令。任务文档是对设备将执行的远程操作的描述。

要获取任务文档，运行以下命令：

```
aws iot get-job-document --job-id 010
```

该命令返回指定任务的任务文档：

```
{
    "document": "{\n\t\"operation\":\"install\",\n\t\"url\":\"http://amazon.com/
firmWareUpate-01\",\n\t\"data\":\"${aws:iot:s3-presigned-url:https://s3.amazonaws.com/
job-test-bucket/datafile}\"\n}"
}
```

> **Note**
>
> 在使用此命令检索任务文档时，占位符 URL 不会替换为预签名 Amazon S3 URL。在设备调用GetPendingJobExecutions API 时，占位符 URL 将替换为任务文档中的预签名 Amazon S3 URL。

列出任务

要获取 AWS 账户 中的所有任务的列表，使用 ListJobs 命令。任务数据和任务执行数据将在 有限的时间内保留。运行以下命令来列出您的AWS 账户中的所有任务：

```
aws iot list-jobs
```

该命令将返回您账户中按任务状态排序的所有任务：

```
{
    "jobs": [
        {
            "status": "IN_PROGRESS",
            "lastUpdatedAt": 1486687079.743,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/013",
            "createdAt": 1486687079.743,
            "targetSelection": "SNAPSHOT",
            "jobId": "013"
        },
        {
            "status": "SUCCEEDED",
            "lastUpdatedAt": 1486685868.444,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/012",
            "createdAt": 1486685868.444,
            "completedAt": 148668789.690,
            "targetSelection": "SNAPSHOT",
            "jobId": "012"
        },
        {
            "status": "CANCELED",
            "lastUpdatedAt": 1486678850.575,
            "jobArn": "arn:aws:iot:us-east-1:123456789012:job/011",
            "createdAt": 1486678850.575,
            "targetSelection": "SNAPSHOT",
            "jobId": "011"
```

```
        }
    ]
}
```

描述任务

要获取任务的状态，运行 DescribeJob 命令。以下命令说明如何描述任务：

```
$ aws iot describe-job --job-id 010
```

该命令返回指定任务的状态。例如：

```
{
    "documentSource": "https://s3.amazonaws.com/job-test-bucket/job-document.json",
    "job": {
        "status": "IN_PROGRESS",
        "jobArn": "arn:aws:iot:us-east-1:123456789012:job/010",
        "targets": [
            "arn:aws:iot:us-east-1:123456789012:thing/myThing"
        ],
        "jobProcessDetails": {
            "numberOfCanceledThings": 0,
            "numberOfFailedThings": 0,
            "numberOfInProgressThings": 0,
            "numberOfQueuedThings": 0,
            "numberOfRejectedThings": 0,
            "numberOfRemovedThings": 0,
            "numberOfSucceededThings": 0,
            "numberOfTimedOutThings": 0,
            "processingTargets": [
                arn:aws:iot:us-east-1:123456789012:thing/thingOne,
                arn:aws:iot:us-east-1:123456789012:thinggroup/thinggroupOne,
                arn:aws:iot:us-east-1:123456789012:thing/thingTwo,
                arn:aws:iot:us-east-1:123456789012:thinggroup/thinggroupTwo
            ]
        },
        "presignedUrlConfig": {
            "expiresInSec": 60,
            "roleArn": "arn:aws:iam::123456789012:role/S3DownloadRole"
        },
        "jobId": "010",
        "lastUpdatedAt": 1486593195.006,
        "createdAt": 1486593195.006,
```

```
        "targetSelection": "SNAPSHOT",
        "jobExecutionsRolloutConfig": {
            "exponentialRate": {
                "baseRatePerMinute": integer,
                "incrementFactor": integer,
                "rateIncreaseCriteria": {
                    "numberOfNotifiedThings": integer, // Set one or the other
                    "numberOfSucceededThings": integer // of these two values.
                },
            "maximumPerMinute": integer
         }
        },
        "abortConfig": {
            "criteriaList": [
                {
                    "action": "string",
                    "failureType": "string",
                    "minNumberOfExecutedThings": integer,
                    "thresholdPercentage": integer
                }
            ]
        },
        "timeoutConfig": {
           "inProgressTimeoutInMinutes": number
          }
    }
}
```

列出任务的执行

在特定设备上运行的任务由任务执行对象表示。运行 ListJobExecutionsForJob 命令来列出任务的所有任务执行。以下命令说明如何列出任务的执行：

```
aws iot list-job-executions-for-job --job-id 010
```

该命令将返回任务执行的列表：

```
{
    "executionSummaries": [
    {
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingOne",
        "jobExecutionSummary": {
            "status": "QUEUED",
```

```
            "lastUpdatedAt": 1486593196.378,
            "queuedAt": 1486593196.378,
            "executionNumber": 1234567890
        }
    },
    {
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingTwo",
        "jobExecutionSummary": {
            "status": "IN_PROGRESS",
            "lastUpdatedAt": 1486593345.659,
            "queuedAt": 1486593196.378,
            "startedAt": 1486593345.659,
            "executionNumber": 4567890123
        }
    }
    ]
}
```

列出事物的任务执行

运行 ListJobExecutionsForThing 命令来列出在事物上运行的所有任务执行。以下命令说明如何列出事物的任务执行：

```
aws iot list-job-executions-for-thing --thing-name thingOne
```

该命令将返回正在指定事物上运行的或在指定事物上运行过的任务执行的列表：

```
{
    "executionSummaries": [
    {
        "jobExecutionSummary": {
            "status": "QUEUED",
            "lastUpdatedAt": 1486687082.071,
            "queuedAt": 1486687082.071,
            "executionNumber": 9876543210
        },
        "jobId": "013"
    },
    {
        "jobExecutionSummary": {
            "status": "IN_PROGRESS",
            "startAt": 1486685870.729,
            "lastUpdatedAt": 1486685870.729,
```

```
            "queuedAt": 1486685870.729,
            "executionNumber": 1357924680
        },
        "jobId": "012"
    },
    {
        "jobExecutionSummary": {
            "status": "SUCCEEDED",
            "startAt": 1486678853.415,
            "lastUpdatedAt": 1486678853.415,
            "queuedAt": 1486678853.415,
            "executionNumber": 4357680912
        },
        "jobId": "011"
    },
    {
        "jobExecutionSummary": {
            "status": "CANCELED",
            "startAt": 1486593196.378,
            "lastUpdatedAt": 1486593196.378,
            "queuedAt": 1486593196.378,
            "executionNumber": 2143174250
        },
        "jobId": "010"
    }
    ]
}
```

描述任务执行

运行 DescribeJobExecution 命令来获取任务执行的状态。您必须指定任务 ID 和事物名称（也可以选择指定执行编号）来标识任务执行。以下命令说明如何描述任务执行：

```
aws iot describe-job-execution --job-id 017 --thing-name thingOne
```

该命令将返回 JobExecution。例如：

```
{
    "execution": {
        "jobId": "017",
        "executionNumber": 4516820379,
        "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/thingOne",
        "versionNumber": 123,
```

```
        "createdAt": 1489084805.285,
        "lastUpdatedAt": 1489086279.937,
        "startedAt": 1489086279.937,
        "status": "IN_PROGRESS",
        "approximateSecondsBeforeTimedOut": 100,
        "statusDetails": {
            "status": "IN_PROGRESS",
            "detailsMap": {
                "percentComplete": "10"
            }
        }
    }
}
```

删除任务执行

运行 DeleteJobExecution 命令来删除任务执行。您必须指定任务 ID、事物名称和执行编号来标识任务执行。以下命令说明如何删除任务执行：

```
aws iot delete-job-execution --job-id 017 --thing-name thingOne --execution-number
 1234567890 --force|--no-force
```

该命令不显示任何输出。

默认情况下，任务执行的状态必须为 QUEUED 或处于最终状态（SUCCEEDED、FAILED、REJECTED、TIMED_OUT、REMOVED 或 CANCELED）。否则将出错。要删除状态为 IN_PROGRESS 的任务执行，您可以将 force 参数设置为 true。

> **Warning**
>
> 删除状态为 IN_PROGRESS 的任务执行后，执行该任务的设备将无法访问任务信息或更新任务执行状态。请谨慎使用，并且 确保 设备能够恢复到有效状态。

任务模板

使用任务模板预置可以部署到多组目标设备的任务。要将频繁执行的远程操作部署到设备（如重启或安装应用程序），可以使用模板来定义标准配置。要执行诸如部署安全补丁和错误修复等操作，您可以从现有任务中创建模板。

创建任务模板时，请指定以下附加配置和资源。

- 任务属性
- 任务文件和目标
- 推出、计划和取消标准
- 超时和重试标准

自定义模板和 AWS 托管模板

根据您要执行的远程操作，您可以创建自定义作业模板或使用 AWS 托管模板。使用自定义作业模板提供您自己的自定义任务文档，并创建可重复使用的作业以部署到您的设备。 AWS 托管模板是作业为常用操作提供的 AWS IoT 作业模板。这些模板具有用于某些远程操作的预定义任务文档，因此您无需创建自己的任务文档。托管式模板可帮助您创建可重复使用的任务，以便更快地启动设备。

主题

使用 AWS 托管模板部署常见的远程操作

AWS 托管模板是由提供的作业模板 AWS。这些任务模板用于频繁执行的远程操作，例如重启、下载文件或在设备上安装应用程序。这些模板具有用于每个远程操作的预定义任务文档，因此您无需创建自己的任务文档。

您可以从一组预定义的配置中进行选择，并使用这些模板创建任务，而无需编写任何其他代码 使用托管模板，您可以查看部署到机群的任务文档。您可以使用这些模板创建任务，然后创建可以重复用于远程操作的自定义任务模板。

托管模板包含哪些内容？

每个 AWS 托管模板都包含：

- 在任务文档中运行命令的环境。
- 指定操作名称及其参数的任务文档。例如，如果您使用下载文件模板，操作名称是下载文件，参数可以是：
  - 要下载到设备的文件的 URL。这可以是互联网资源，也可以是公共或预签名 Amazon Simple Storage Service (Amazon S3) URL。
  - 设备上用于存储下载文件的本地文件路径。

有任务文档及其参数的更多信息，请参阅 托管模板远程操作和任务文档。

先决条件

要使设备运行托管模板任务文档指定的远程操作，您必须：

- 在设备上安装特定软件

  使用您自己的设备软件和作业处理程序或 AWS IoT 设备客户端。根据您的业务案例，您也可以同时运行它们，以使它们执行不同的功能。

  - 使用您自己的设备软件和任务处理程序

    您可以使用 AWS IoT Device SDK 及其支持远程操作的处理程序库为设备编写自己的代码。要部署和运行任务，请验证设备代理库已正确安装并正在这些设备上运行。

    您也可以选择使用自己的支持远程操作的处理程序。有关更多信息，请参阅 AWS IoT 设备客户端 GitHub存储库中的示例作业处理程序。

  - 使用 AWS IoT 设备客户端

    或者，您可以在设备上安装和运行 AWS IoT 设备客户端，因为它默认支持直接从控制台使用所有托管模板。

    Device Client 是用 C++ 编写的开源软件，您可以在基于 Linux 的嵌入式 IoT 设备上编译和安装。Device Client 有基本客户端和离散的客户端特征。基础客户端 AWS IoT 通过 MQTT 协议建立连接，并且可以连接不同的客户端功能。

    要在设备上执行远程操作，请使用 Device Client 的客户端任务特征。此特征包含一个用于接收任务文档的解析器以及实现任务文档中指定的远程操作的任务处理程序。有关Device Client 及其特征的更多信息，请参阅 AWS IoT Device Client。

    在设备上运行时，Device Client 会收到任务文档，并具有用于在文档中运行命令的特定于平台的实现。有关设置 Device Client 和使用任务特征的详细信息，请参阅 AWS IoT 教程。

- 使用支持的环境

  对于每个托管模板，您将找到有关可用于运行远程操作的环境信息。我们建议您将模板与模板中指定的受支持的 Linux 环境结合使用。使用 AWS IoT 设备客户端运行托管模板远程操作，因为它支持常见的微处理器和 Linux 环境，例如 Debian 和 Ubuntu。

托管模板远程操作和任务文档

以下部分列出了 AWS IoT 任务的不同 AWS 托管模板，并描述了可以在设备上执行的远程操作。以下章节提供有关任务文档的信息以及每个远程操作的任务文档参数的描述。设备端软件使用模板名称和参数来执行远程操作。

AWS 托管模板接受您在使用模板创建作业时为其指定值的输入参数。所有托管式模板都有以下两个共同的可选输入参数：`runAsUser` 和 `pathToHandler`。除了 `AWS-Reboot` 模板时，模板需要额外的输入参数，在使用模板创建任务时必须为这些参数指定值。这些所需的输入参数因您所选的模板而异。例如，如果您选择 `AWS-Download-File` 模板，则必须指定要安装的软件包列表以及从中下载文件的 URL。

使用 AWS IoT 控制台或 AWS Command Line Interface (AWS CLI) 创建使用托管模板的作业时，请为输入参数指定一个值。使用 CLI 时，可以使用 `document-parameters` 对象提供这些值。有关更多信息，请参阅文档参数。

> **Note**
>
> 仅当从 AWS 托管式模板创建任务时，才使用 `document-parameters`。此参数不能与自定义任务模板一起使用，也不能从中创建作业。

下面显示了常见可选输入参数的说明。您将在下一节中看到每个托管式模板所需的其它输入参数的说明。

`runAsUser`

此参数指定是否以其他用户身份运行任务处理程序。如果在任务创建过程中未指定，任务处理程序将以与 Device Client 相同的用户身份运行。当您以其他用户身份运行任务处理程序时，请指定不超过 256 个字符的字符串值。

`pathToHandler`

设备上运行的任务处理程序的路径。如果在创建任务时未指定它，Device Client 将使用当前工作目录。

下面显示了不同的远程操作、其任务文档以及它们接受的参数。所有这些模板都支持在设备上运行远程操作的 Linux 环境。

AWS - 下载- 文件

模板名称

AWS-Download-File

模板描述

提供的 AWS 用于下载文件的托管模板。

输入参数

此模板具有以下必需参数。您还可以指定可选参数 runAsUser 和 pathToHandler

downloadUrl

从中下载文件的 URL。这可以是互联网资源、Amazon S3 中可公开访问的对象，也可以是 Amazon S3 中只能由设备使用预签名 URL 访问的对象。有关使用预签名 URL 和授予权限的详细信息，请参阅 预签名 URL。

filePath

本地文件路径，显示设备中存储下载文件的位置。

设备的行为

设备从指定位置下载文件，验证下载完成并将其存储在本地。

任务文档

下面显示了任务文档及其最新版本。模板显示了任务处理程序和 shell 脚本的路径，download-file.sh，必须运行任务处理程序才能下载文件。它还显示了必需参数 downloadUrl 和 filePath。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Download-File",
        "type": "runHandler",
        "input": {
          "handler": "download-file.sh",
          "args": [
```

```
            "${aws:iot:parameter:downloadUrl}",
            "${aws:iot:parameter:filePath}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–安装 - 应用程序。

模板名称

AWS-Install-Application

模板描述

由提供的 AWS 用于安装一个或多个应用程序的托管模板。

输入参数

此模板具有以下必需参数，packages。您还可以指定可选参数 runAsUser 和 pathToHandler

packages

要安装的一个或多个应用程序列表，以空格分隔。

设备的行为

设备将按照任务文档中的指定安装应用程序。

任务文档

下面显示了任务文档及其最新版本。模板显示了任务处理程序和 shell 脚本的路径，install-packages.sh，必须运行任务处理程序才能下载文件。它还显示必需参数 packages。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Install-Application",
        "type": "runHandler",
```

```
        "input": {
          "handler": "install-packages.sh",
          "args": [
            "${aws:iot:parameter:packages}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS - 重启

模板名称

AWS-Reboot

模板描述

提供的 AWS 用于重启设备的托管模板。

输入参数

此模板没有必需参数。但是，您可以指定可选参数 runAsUser 和 pathToHandler。

设备的行为

设备已成功重新启动。

任务文档

下面显示了任务文档及其最新版本。模板显示了任务处理程序和 shell 脚本的路径，reboot.sh，必须运行任务处理程序才能重新启动设备。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Reboot",
        "type": "runHandler",
        "input": {
```

```
          "handler": "reboot.sh",
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–删除–应用程序

模板名称

AWS-Remove-Application

模板描述

提供的托管模板， AWS 用于卸载一个或多个应用程序。

输入参数

此模板具有以下必需参数，packages。您还可以指定可选参数 runAsUser 和 pathToHandler

packages

要卸载的一个或多个应用程序列表，以空格分隔。

设备的行为

设备将按照任务文档中的指定卸载应用程序。

任务文档

下面显示了任务文档及其最新版本。模板显示了任务处理程序和 shell 脚本的路径，remove-packages.sh，必须运行任务处理程序才能下载文件。它还显示必需参数 packages。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Remove-Application",
        "type": "runHandler",
        "input": {
          "handler": "remove-packages.sh",
```

```
          "args": [
            "${aws:iot:parameter:packages}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS— 重新启动 — 应用

模板名称

AWS-Restart-Application

模板描述

提供的托管模板， AWS 用于停止和重启一项或多项服务。

输入参数

此模板具有以下必需参数，services。您还可以指定可选参数 runAsUser 和 pathToHandler

服务

将重新启动的一个或多个应用程序列表，以空格分隔。

设备的行为

指定的应用程序停止然后在设备上重新启动。

任务文档

下面显示了任务文档及其最新版本。模板显示了任务处理程序和 shell 脚本的路径，restart-services.sh，必须运行任务处理程序才能重新启动系统服务。它还显示必需参数 services。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Restart-Application",
        "type": "runHandler",
```

```
        "input": {
          "handler": "restart-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–开始–复制

模板名称

AWS-Start-Application

模板描述

为启动一项或多 AWS 项服务而提供的托管模板。

输入参数

此模板具有以下必需参数，services。您还可以指定可选参数 runAsUser 和 pathToHandler

services

将启动的一个或多个应用程序列表，以空格分隔。

设备的行为

指定的应用程序开始在设备上运行。

任务文档

下面显示了任务文档及其最新版本。模板显示了任务处理程序和 shell 脚本的路径，start-services.sh，必须运行任务处理程序才能启动系统服务。它还显示必需参数 services。

```
{
  "version": "1.0",
  "steps": [
    {
```

```
      "action": {
        "name": "Start-Application",
        "type": "runHandler",
        "input": {
          "handler": "start-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS-停止-应用程序

模板名称

AWS-Stop-Application

模板描述

提供的托管模板， AWS 用于停止一项或多项服务。

输入参数

此模板具有以下必需参数，services。您还可以指定可选参数 runAsUser 和 pathToHandler

services

将停止的一个或多个应用程序列表，以空格分隔。

设备的行为

指定的应用程序停止在设备上运行。

任务文档

下面显示了任务文档及其最新版本。模板显示了任务处理程序和 shell 脚本的路径，stop-services.sh，必须运行任务处理程序才能停止系统服务。它还显示必需参数 services。

```
{
  "version": "1.0",
```

```
  "steps": [
    {
      "action": {
        "name": "Stop-Application",
        "type": "runHandler",
        "input": {
          "handler": "stop-services.sh",
          "args": [
            "${aws:iot:parameter:services}"
          ],
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

AWS–Run–Command

模板名称

AWS-Run-Command

模板描述

由提供的 AWS 用于运行 shell 命令的托管模板。

输入参数

此模板具有以下必需参数，command。您还可以指定可选参数 runAsUser。

command

以逗号分隔的命令字符串。必须对命令本身中包含的任何逗号进行转义。

设备的行为

设备按照任务文档中的指定运行 shell 命令。

任务文档

下面显示了任务文档及其最新版本。此模板显示任务命令的路径，以及您提供的且设备将运行的命令。

```
{
```

```
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Run-Command",
        "type": "runCommand",
        "input": {
          "command": "${aws:iot:parameter:command}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

主题

- 使用 AWS 托管模板创建作业 AWS Management Console
- 使用 AWS 托管模板创建作业 AWS CLI

使用 AWS 托管模板创建作业 AWS Management Console

使用获取 AWS Management Console 有关 AWS 托管模板的信息，并使用这些模板创建作业。然后，您可以将创建的任务保存为自己的自定义模板。

获取有关托管模板的详细信息

您可以从 AWS IoT 控制台获取有关可供使用的不同托管模板的信息。

1. 要查看可用的托管模板，请前往AWS IoT 控制台的 Job Templates 中心，然后选择托管模板选项卡。
2. 要查看详细信息，请选择托管式模板。

详细信息页面包括以下信息：

- 托管模板的名称、描述和 Amazon Resource Name (ARN)。
- 可以执行远程操作的环境，例如 Linux。
- JSON 任务文档，用于指定任务处理程序的路径以及要在设备上运行的命令。例如，下面显示了一个示例任务文档 AWS-Reboot 模板。模板显示了任务处理程序和 shell 脚本的路径，`reboot.sh`，必须运行任务处理程序才能重新启动设备。

```
{
  "version": "1.0",
  "steps": [
    {
      "action": {
        "name": "Reboot",
        "type": "runHandler",
        "input": {
          "handler": "reboot.sh",
          "path": "${aws:iot:parameter:pathToHandler}"
        },
        "runAsUser": "${aws:iot:parameter:runAsUser}"
      }
    }
  ]
}
```

有关任务文档及其各种远程操作参数的详细信息，请参阅 托管模板远程操作和任务文档。

- 任务文档的最新版本。

使用托管模板创建任务

您可以使用 AWS 管理控制台选择用于创建作业的 AWS 托管模板。本节向您演示了应如何进行操作。

您也可以启动任务创建工作流程，然后选择要在创建作业时使用的 AWS 托管模板。有关工作流的更多信息，请参阅 使用 AWS Management Console 创建和管理任务。。

1. 选择您的 AWS 托管模板

   前往AWS IoT 控制台的 Job 模板中心，选择托管模板选项卡，然后选择您的模板。

2. 使用托管模板创建任务

   1. 在模板详细信息页面上，选择 Create job(创建任务)。

      控制台切换到添加模板配置的工作流的创建任务的自定义任务属性的步骤。

   2. 输入唯一的字母数字任务名称以及可选的描述和标签，然后选择下一步。
   3. 选择要在此任务中运行的事物或事物组作为任务目标。
   4. 在 Job document（任务文档）部分中，您的模板随其配置设置和输入参数一起显示。输入所选模板的输入参数的值。例如，如果您选择了 AWS-Download-File 模板：

- 对于 downloadUrl，输入要下载的文件的 URL，例如：`https://example.com/index.html`。
- 对于 filePath，输入设备上存储下载文件的路径，例如：*path/to/file*。

还可以选择输入 `runAsUser` 和 `pathToHandler` 参数的值。有关每个模板的输入参数的更多信息，请参阅托管模板远程操作和任务文档。

5. 在 Job configuration（作业配置）页面上，将作业类型选择为连续或快照作业。快照任务在目标设备和组上完成运行后即算完成任务。连续任务适用于事物组，并会添加到指定目标组的任何设备上运行。
6. 继续为您的任务添加任何其他配置，然后查看并创建任务。有关其他配置的更多信息，请参阅：
   - 任务推出、计划和中止配置
   - 任务执行超时和重试配置

从托管模板创建自定义任务模板

您可以使用 AWS 托管模板和自定义作业作为起点来创建自己的自定义作业模板。要创建自定义作业模板，请先使用您的 AWS 托管模板创建作业，如上一节所述。

然后，您可以将自定义任务另存为模板，创建自己的自定义任务模板。要另存为模板：

1. 前往AWS IoT 控制台的 Job 中心并选择包含您的托管模板的作业。
2. 选择另存为任务模板，然后创建自定义任务模板。有关创建自定义模板的更多信息，请参阅 从现有任务创建任务模板。

使用 AWS 托管模板创建作业 AWS CLI

使用获取 AWS CLI 有关 AWS 托管模板的信息，并使用这些模板创建作业。然后，您可以将任务另存为模板，并创建自己的自定义模板。

列出托管模板

该`list-managed-job-templates` AWS CLI 命令列出了您的所有作业模板 AWS 账户。

```
aws iot list-managed-job-templates
```

默认情况下，运行此命令会显示所有可用的 AWS 托管模板及其详细信息。

```
{
    "managedJobTemplates": [
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Reboot:1.0",
            "templateName": "AWS-Reboot",
            "description": "A managed job template for rebooting the device.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Remove-
Application:1.0",
            "templateName": "AWS-Remove-Application",
            "description": "A managed job template for uninstalling one or more
 applications.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },
        {
            "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Stop-Application:1.0",
            "templateName": "AWS-Stop-Application",
            "description": "A managed job template for stopping one or more system
 services.",
            "environments": [
                "LINUX"
            ],
            "templateVersion": "1.0"
        },

        ...

        {
            "templateArn": "arn:aws:iot:us-east-1::jobtemplate/AWS-Restart-
Application:1.0",
            "templateName": "AWS-Restart-Application",
            "description": "A managed job template for restarting one or more system
 services.",
            "environments": [
```

```
                "LINUX"
            ],
            "templateVersion": "1.0"
        }
    ]
}
```

有关更多信息，请参阅ListManagedJobTemplates。

获取有关托管式模板的详细信息

该describe-managed-job-template AWS CLI 命令获取有关指定作业模板的详细信息。指定任务模板名称和可选的模板版本。如果未指定模板版本，则返回预定义的默认版本。以下示例显示运行命令以获取有关 AWS-Download-File 模板的详细信息。

```
aws iot describe-managed-job-template \
    --template-name AWS-Download-File
```

该命令显示模板详细信息和 ARN、其任务文档以及 documentParameters 参数，此参数是模板输入参数的键/值对的列表。有关不同模板和输入参数的信息，请参阅托管模板远程操作和任务文档。

> **Note**
>
> 使用此 API 时返回的documentParameters对象只能在通过 AWS 托管模板创建任务时使用。此对象不得用于自定义任务模板。有关演示如何使用此参数的示例，请参阅使用托管模板创建任务。

```
{
    "templateName": "AWS-Download-File",
    "templateArn": "arn:aws:iot:region::jobtemplate/AWS-Download-File:1.0",
    "description": "A managed job template for downloading a file.",
    "templateVersion": "1.0",
    "environments": [
        "LINUX"
    ],
    "documentParameters": [
        {
            "key": "downloadUrl",
            "description": "URL of file to download.",
```

```
            "regex": "(.*?)",
            "example": "http://www.example.com/index.html",
            "optional": false
        },
        {
            "key": "filePath",
            "description": "Path on the device where downloaded file is written.",
            "regex": "(.*?)",
            "example": "/path/to/file",
            "optional": false
        },
        {
            "key": "runAsUser",
            "description": "Execute handler as another user. If not specified, then
 handler is executed as the same user as device client.",
            "regex": "(.){0,256}",
            "example": "user1",
            "optional": true
        },
        {
            "key": "pathToHandler",
            "description": "Path to handler on the device. If not specified, then
 device client will use the current working directory.",
            "regex": "(.){0,4096}",
            "example": "/path/to/handler/script",
            "optional": true
        }
    ],
    "document": "{\"version\":\"1.0\",\"steps\":[{\"action\":{\"name
\":\"Download-File\",\"type\":\"runHandler\",\"input\":{\"handler\":
\"download-file.sh\",\"args\":[\"${aws:iot:parameter:downloadUrl}\",
\"${aws:iot:parameter:filePath}\"],\"path\":\"${aws:iot:parameter:pathToHandler}\"},
\"runAsUser\":\"${aws:iot:parameter:runAsUser}\"}}]}"
}
```

有关更多信息，请参阅DescribeManagedJobTemplate。

使用托管模板创建任务

该create-job AWS CLI 命令可用于根据作业模板创建作业。它的目标是名为 thingOne 的设备，并将指定托管模板的 Amazon Resource Name (ARN)，用作该任务的基础。您可以覆盖高级配置（如超时和取消配置），方法是传递 create-job 命令的关联参数。

该示例显示如何创建使用 AWS-Download-File 模板的任务。它还展示了如何使用 document-parameters 参数指定模板的输入参数。

> **Note**
>
> 仅在 AWS 托管模板中使用该document-parameters对象。此对象不得与自定义任务模板一起使用。

```
aws iot create-job \
   --targets arn:aws:iot:region:account-id:thing/thingOne \
   --job-id "new-managed-template-job" \
   --job-template-arn arn:aws:iot:region::jobtemplate/AWS-Download-File:1.0 \
   --document-parameters downloadUrl=https://example.com/index.html,filePath=path/to/
file
```

其中：

- *##*是 AWS 区域.
- *account-id* 唯一的 AWS 账户 数字。
- *thingOne* 是任务指向的 IoT 事物的名称。
- *AWS-Download-File:1.0* 是托管式模板的名称。
- https://*example.com/index*.html 是从中下载文件的 URL。
- https://*pathto/file/index* 是设备上用于存储下载文件的路径。

运行以下命令创建模板的任务：*AWS-Download-File*。

```
{
    "jobArn": "arn:aws:iot:region:account-id:job/new-managed-template-job",
    "jobId": "new-managed-template-job",
    "description": "A managed job template for downloading a file."
}
```

从托管模板创建自定义任务模板

1. 使用上一节中介绍的托管模板创建任务。
2. 使用您创建的任务的 ARN 创建自定义任务模板。有关更多信息，请参阅从现有任务创建任务模板。

创建自定义任务模板

您可以使用 AWS CLI 和 AWS IoT 控制台创建作业模板。您还可以使用用于 AWS IoT 设备管理 Web 应用程序的 Fleet Hub AWS CLI、 AWS IoT 控制台和舰队中心，根据任务模板创建作业。有关在 Fleet Hub 应用程序中使用作业模板的更多信息，请参阅在 F leet Hub 中使用任务模板进行 AWS IoT 设备管理。

> **Note**
>
> 作业文档中的替换模式总数应小于或等于 10。

主题

通过使用 AWS Management Console 创建自定义任务模板

本主题介绍如何使用 AWS IoT 控制台创建、删除和查看有关作业模板的详细信息。

创建自定义任务模板

您可以创建一个原始 自定义任务模板或从现有任务创建任务模板。您也可以根据使用 AWS 托管模板创建的现有作业创建自定义作业模板。有关更多信息，请参阅从托管模板创建自定义任务模板。

创建原始任务模板

1. 开始创建您的任务模板

    1. 前往AWS IoT 控制台的 Job 模板中心，然后选择自定义模板选项卡。
    2. 选择 Create job template（创建任务模板）。

    > **Note**
    >
    > 您还可以从 Fleet Hub 下的 Related services（相关服务）页面导航到 Job templates（任务模板）页面。

2. 指定任务模板属性

   在创建任务模板页面中，输入任务名称的字母数字标识符和字母数字描述，提供有关该模板的其他详细信息。

   > **Note**
   >
   > 我们建议不要在您的任务 ID 或描述中使用个人身份信息。

3. 提供任务文档

   提供 JSON 任务文件，该文件存储在 S3 存储桶中或作为任务中指定的内联任务文档。当您使用此模板创建任务时，此任务文件将成为任务文档。

   如果任务文件存储在 S3 存储桶中，请输入 S3 URL 或选择浏览 S3，然后导航到您的工作文档并选择它。

   > **Note**
   >
   > 您可以选择当前区域中的 S3 存储桶。

4. 继续为您的任务添加任何其他配置，然后查看并创建任务。有关其他可选配置的信息，请参阅以下链接：

   - 任务推出、计划和中止配置
   - 任务执行超时和重试配置

从现有任务创建任务模板

1. 选择您的任务

   1. 转到AWS IoT 控制台的 Job Hub，然后选择要用作作业模板基础的作业。
   2. 选择另存为任务模板。

   > **Note**
   >
   > 您可以选择不同的任务文档或编辑原始任务中的高级配置，然后选择 Create job template（创建任务模板）。您的新任务模板将显示在 Job template（任务模板）页面。

2. 指定任务模板属性

   在创建任务模板页面中，输入任务名称的字母数字标识符和字母数字描述，提供有关该模板的其他详细信息。

   > **Note**
   >
   > 任务文档是您在创建模板时指定的任务文件。如果是在任务中指定任务文档的，而不是 S3 位置，可以在此任务的详细信息页面中看到任务文档。

3. 继续为您的任务添加任何其他配置，然后查看并创建任务。有关其他配置的更多信息，请参阅：

   - 任务推出、计划和中止配置
   - 任务执行超时和重试配置

从自定义任务模板创建任务。

您可以通过转到任务模板的详细信息页面，从自定义任务模板创建任务，如本主题中所述。您还可以创建任务，或者通过选择运行任务创建工作流程时要使用的任务模板来创建任务。有关更多信息，请参阅使用 AWS Management Console 创建和管理任务。。

本主题介绍如何从自定义任务模板的详细信息页面创建任务。您也可以使用 AWS 托管模板创建作业。有关更多信息，请参阅使用托管模板创建任务。

1. 选择自定义任务模板

   前往AWS IoT 控制台的 Job Tem plates 中心，选择自定义模板选项卡，然后选择您的模板。

2. 使用自定义模板创建任务

   要创建任务：

   1. 在模板详细信息页面上，选择 Create job(创建任务)。

      控制台切换到添加模板配置的工作流的创建任务的自定义任务属性的步骤。

   2. 输入唯一的字母数字任务名称以及可选的描述和标签，然后选择下一步。
   3. 选择要在此任务中运行的事物或事物组作为任务目标。

      在任务文档部分中，您的模板随其配置设置一起显示。如果要使用不同的任务文档，请选择 Browse（浏览），然后选择不同的存储桶和文档。选择下一步。

4. 在 Job configuration（作业配置）页面上，将作业类型选择为连续或快照作业。快照任务在目标设备和组上完成运行后即算完成任务。连续任务适用于事物组，并会添加到指定目标组的任何设备上运行。
5. 继续为您的任务添加任何其他配置，然后查看并创建任务。有关其他配置的更多信息，请参阅：
   - 任务推出、计划和中止配置
   - 任务执行超时和重试配置

> **Note**
>
> 当根据任务模板创建的任务更新该任务模板提供的现有参数时，这些更新的参数将覆盖该任务模板为该任务提供的现有参数。

您还可以使用 Fleet Hub Web 应用程序从任务模板创建任务。有关在 Fleet Hub 中创建任务的信息，请参阅在 Fleet Hub 中使用任务模板进行 AWS IoT 设备管理。

删除任务模板

要删除作业模板，请先进入AWS IoT 控制台的 Job Templates 中心，然后选择自定义模板选项卡。然后，选择要删除的任务模板，然后选择下一步。

> **Note**
>
> 删除是永久性的，任务模板不再显示在自定义模板选项卡。

通过使用 AWS CLI 创建自定义任务模板

本主题介绍如何使用 AWS CLI 创建、删除任务模板以及检索有关详细信息。

从头开始创建任务模板

```
## AWS CLI ########### S3 ######### (job-document.json) #### Amazon S3 (S3) ############DownloadRole
```

```
aws iot create-job-template  \
      --job-template-id 010  \
      --document-source https://s3.amazonaws.com/my-s3-bucket/job-document.json  \
      --timeout-config inProgressTimeoutInMinutes=100 \
      --job-executions-rollout-config "{ \"exponentialRate\": { \"baseRatePerMinute\":
 50, \"incrementFactor\": 2, \"rateIncreaseCriteria\": { \"numberOfNotifiedThings\":
 1000, \"numberOfSucceededThings\": 1000}}, \"maximumPerMinute\": 1000}" \
      --abort-config "{ \"criteriaList\": [ { \"action\": \"CANCEL\", \"failureType
\": \"FAILED\", \"minNumberOfExecutedThings\": 100, \"thresholdPercentage\": 20},
 { \"action\": \"CANCEL\", \"failureType\": \"TIMED_OUT\", \"minNumberOfExecutedThings
\": 200, \"thresholdPercentage\": 50}]}" \
      --presigned-url-config "{\"roleArn\":\"arn:aws:iam::123456789012:role/
S3DownloadRole\", \"expiresInSec\":3600}"
```

可选的 `timeout-config` 参数指定每个设备完成任务运行所具有的时间。计时器在任务执行状态设置为 `IN_PROGRESS` 时启动。如果任务执行状态未在时间到期之前设置为其它最终状态，则会设置为 `TIMED_OUT`。

进行中的计时器无法更新，将应用到该任务的所有任务启动。每当任务启动保持IN_PROGRESS状态的时间超过此时间间隔时，任务启动就会失败并切换到终端TIMED_OUT状态。 AWS IoT 还会发布 MQTT 通知。

有关创建任务推出和中止相关配置的更多信息，请参阅任务推出和中止配置。

> **Note**
>
> 在您创建任务时，系统会检索指定为 Amazon S3 文件的任务文档。如果在创建任务后更改任务文档源 Amazon S3文件的内容，发送到任务目标的内容不会更改。

从现有任务创建任务模板

以下 AWS CLI 命令通过指定现有任务的 Amazon 资源名称 (ARN) 来创建任务模板。新任务模板将使用任务中指定的所有配置。或者，您可以使用任何可选参数更改现有任务中的任何配置。

```
aws iot create-job-template  \
      --job-arn arn:aws:iot:region:123456789012:job/job-name  \
      --timeout-config inProgressTimeoutInMinutes=100
```

获取有关任务模板的详细信息

以下 AWS CLI 命令获取有关指定作业模板的详细信息。

```
aws iot describe-job-template \
      --job-template-id template-id
```

该命令将显示以下输出。

```
{
   "abortConfig": {
      "criteriaList": [
         {
            "action": "string",
            "failureType": "string",
            "minNumberOfExecutedThings": number,
            "thresholdPercentage": number
         }
      ]
   },
   "createdAt": number,
   "description": "string",
   "document": "string",
   "documentSource": "string",
   "jobExecutionsRolloutConfig": {
      "exponentialRate": {
         "baseRatePerMinute": number,
         "incrementFactor": number,
         "rateIncreaseCriteria": {
            "numberOfNotifiedThings": number,
            "numberOfSucceededThings": number
         }
      },
      "maximumPerMinute": number
   },
   "jobTemplateArn": "string",
   "jobTemplateId": "string",
   "presignedUrlConfig": {
      "expiresInSec": number,
      "roleArn": "string"
```

```
   },
   "timeoutConfig": {
      "inProgressTimeoutInMinutes": number
   }
}
```

列出任务模板

以下 AWS CLI 命令列出了您的所有作业模板 AWS 账户。

```
aws iot list-job-templates
```

该命令将显示以下输出。

```
{
   "jobTemplates": [
      {
         "createdAt": number,
         "description": "string",
         "jobTemplateArn": "string",
         "jobTemplateId": "string"
      }
   ],
   "nextToken": "string"
}
```

要检索其它结果页面，使用 nextToken 字段的值。

删除任务模板

以下 AWS CLI 命令删除指定的作业模板。

```
aws iot delete-job-template \
      --job-template-id template-id
```

该命令不显示任何输出。

从自定义任务模板创建任务。

以下 AWS CLI 命令根据自定义作业模板创建作业。它的目标是名为 thingOne 的设备，并将指定任务模板的 Amazon Resource Name (ARN)，用作该任务的基础。您可以覆盖高级配置（如超时和取消配置），方法是传递 create-job 命令的关联参数。

> **Warning**
>
> 仅当从 AWS 托管式模板创建任务时，document-parameters 对象才能与 create-job 命令一起使用。此对象不得与自定义任务模板一起使用。有关演示如何使用此参数创建任务的示例，请参阅使用托管模板创建任务。

```
aws iot create-job \
      --targets arn:aws:iot:region:123456789012:thing/thingOne  \
      --job-template-arn arn:aws:iot:region:123456789012:jobtemplate/template-id
```

任务配置

对于部署到指定目标的每个任务，您可以拥有以下附加配置。

- 推出：定义了每分钟有多少台设备接收任务文档。
- 计划：除了使用定期维护时段外，还可针对将来的日期和时间计划任务。
- 中止：如果部分设备未收到任务通知或设备报告其任务执行失败，则取消任务。
- Timeout（超时）：如果任务目标在其任务执行开始后的一段时间内没有响应，则任务可能会失败。
- 重试：如果设备在尝试完成任务执行时报告失败或任务执行超时，则重试任务执行。

通过使用这些配置，您可以监控任务执行状态，避免向整个机群发送错误的更新。

主题

任务配置的工作原理

您可以在部署任务时使用推出和中止配置，在执行任务时使用超时和重试配置。以下各部分显示了有关这些配置如何工作的更多信息。

主题

任务推出、计划和中止配置

您可以使用任务推出、计划和中止配置来定义接收任务文档的设备数量、计划任务推出，并确定取消任务的条件。

任务推出配置

您可以指定向目标发送待处理任务执行通知的速度。您可以创建分段推出来管理更新、重启和其它操作。若要指定目标的通知方式，请使用任务推出速率。

任务推出速率

您可以使用恒定推出速率或指数推出速率来创建推出配置。您可以使用恒定推出速率来指定每分钟可通知的最大任务目标数。

AWS IoT 当满足各种标准和阈值时，可以使用指数级部署率来部署工作。如果失败任务数与指定的一组条件相匹配，则您可以取消任务推出。在创建任务时，使用 `JobExecutionsRolloutConfig` 对象设置任务推出速率条件。在创建任务时，使用 `AbortConfig` 对象设置任务中止条件。

下面的示例显示推出速率的工作原理。例如，基本速率为每分钟 50 次、增量因子为 2、通知和成功设备数量各为 1000 的任务推出，其工作方式如下：该任务将以每分钟 50 次任务执行的速率开始，并以该速率继续工作，直到 1000 个事物收到任务执行通知，或发生了 1000 次成功的任务执行。

下表说明了推出将如何通过前四个增量继续。

每分钟的推出速率	50	100	200	400
满足速率提高的通知设备数或成功的任务执行	1000	2000	3000	4,000

> **Note**
>
> 如果您的最大并发任务限制为 500 个 (`isConcurrent = True`)，则所有活动任务的状态将保持为 `IN-PROGRESS`，并且在并发作业数达到 499 个 (`isConcurrent = False`)) 之前，不会推出任何新任务执行。此规则适用于快照任务和连续任务。
> 如果 `isConcurrent = True`，表示该作业当前正在向目标组中的所有设备推出作业执行。
> 如果 `isConcurrent = False`，则表示该任务已完成向目标组中的所有设备推出全部任务执行。如果您选择了任务中止配置，则当目标组中的所有设备都达到最终状态或目标组的阈值百分比时，任务将更新其状态。`isConcurrent = True` 和 `isConcurrent = False` 的任务级别状态都为 `IN_PROGRESS`。
> 有关活动和并发任务限制的更多信息，请参阅活动和并发任务限制。

使用动态事物组的连续任务的任务推出速率

当你使用连续任务在队列上部署远程操作时，J AWS IoT obs 会为目标事物组中的设备推出任务执行。对于添加到动态事务组中的新设备，这些任务执行会继续推出到这些设备，甚至在创建任务之后也是如此。

推出配置只能控制在创建任务之前添加到组中的设备的推出速率。创建任务后，对于任何新设备，一旦设备加入目标组，就会立即近乎实时地创建任务执行。

任务计划配置

您可以使用预先确定的开始时间、结束时间和结束行为，最多提前一年安排连续任务或快照任务，以确定在到达结束时间后每个任务执行将发生的情况。此外，您可以创建一个可选的定期维护时段，该时段具有灵活的频率、开始时间和持续时间，以供连续任务将任务文档推出到目标组中的所有设备。

任务计划配置

开始时间

计划任务的开始时间是该任务开始向目标组中的所有设备推出任务文档的未来日期和时间。计划任务的开始时间适用于连续任务和快照任务。在最初创建计划任务时，该任务保持 `SCHEDULED` 状态。到达您选择的 `startTime` 后，任务将更新为 `IN_PROGRESS` 并开始推出任务文档。`startTime` 必须小于或等于自您创建计划任务的初始日期和时间起一年。

有关使用 API 命令或`startTime`时的语法的更多信息 AWS CLI，请参阅时间戳。

对于具有可选计划配置的任务，且该任务在定期维护时段内在采用夏令时（DST）的位置发生，从 DST 切换到标准时间和从标准时间切换到 DST 时，时间将变动一小时。

> **Note**
>
> 中显示的时区 AWS Management Console 是您当前的系统时区。但是，这些时区将在系统中转换为 UTC。

结束时间

计划任务的结束时间是该任务停止向目标组中的任何剩余设备推出任务文档的未来日期和时间。计划任务的结束时间适用于连续任务和快照任务。在计划任务达到所选的 endTime 并且所有任务执行都达到最终状态后，它会将其状态从 IN_PROGRESS 更新为 COMPLETED。endTime 必须小于或等于自您创建计划任务的初始日期和时间起两年。介于 startTime 和 endTime 之间的最短持续时间为 30 分钟。将重试任务执行，直到任务达到 endTime，然后 endBehavior 将决定如何继续。

有关使用 API 命令或endTime时的语法的更多信息 AWS CLI，请参阅时间戳。

对于具有可选计划配置的任务，且该任务在定期维护时段内在采用夏令时（DST）的位置发生，从 DST 切换到标准时间和从标准时间切换到 DST 时，时间将变动一小时。

> **Note**
>
> 中显示的时区 AWS Management Console 是您当前的系统时区。但是，这些时区将在系统中转换为 UTC。

结束行为

计划任务的结束行为决定了当任务达到所选的 endTime 时，任务执行和所有未完成的任务执行会发生什么情况。

以下列出了创建任务或任务模板时可以选择的结束行为：

- STOP_ROLLOUT
  - STOP_ROLLOUT 会停止向任务的目标组中的所有剩余设备推出任务文档。此外，所有 QUEUED 和 IN_PROGRESS 任务执行都将继续，直到达到最终状态。除非选择 CANCEL 或 FORCE_CANCEL，否则这是默认结束行为。
- CANCEL
  - CANCEL 会停止向任务的目标组中的所有剩余设备推出任务文档。此外，所有 QUEUED 任务执行都将被取消，而所有 IN_PROGRESS 任务执行都将继续，直到达到终止状态。

- FORCE_CANCEL
  - FORCE_CANCEL 会停止向任务的目标组中的所有剩余设备推出任务文档。此外，所有 QUEUED 和 IN_PROGRESS 任务执行都将被取消。

> **Note**
>
> 要选择一个endbehavior，必须选择一个 endtime

最长持续时间

无论 startTime 和 endTime 如何，计划任务的最长持续时间都必须小于或等于两年。

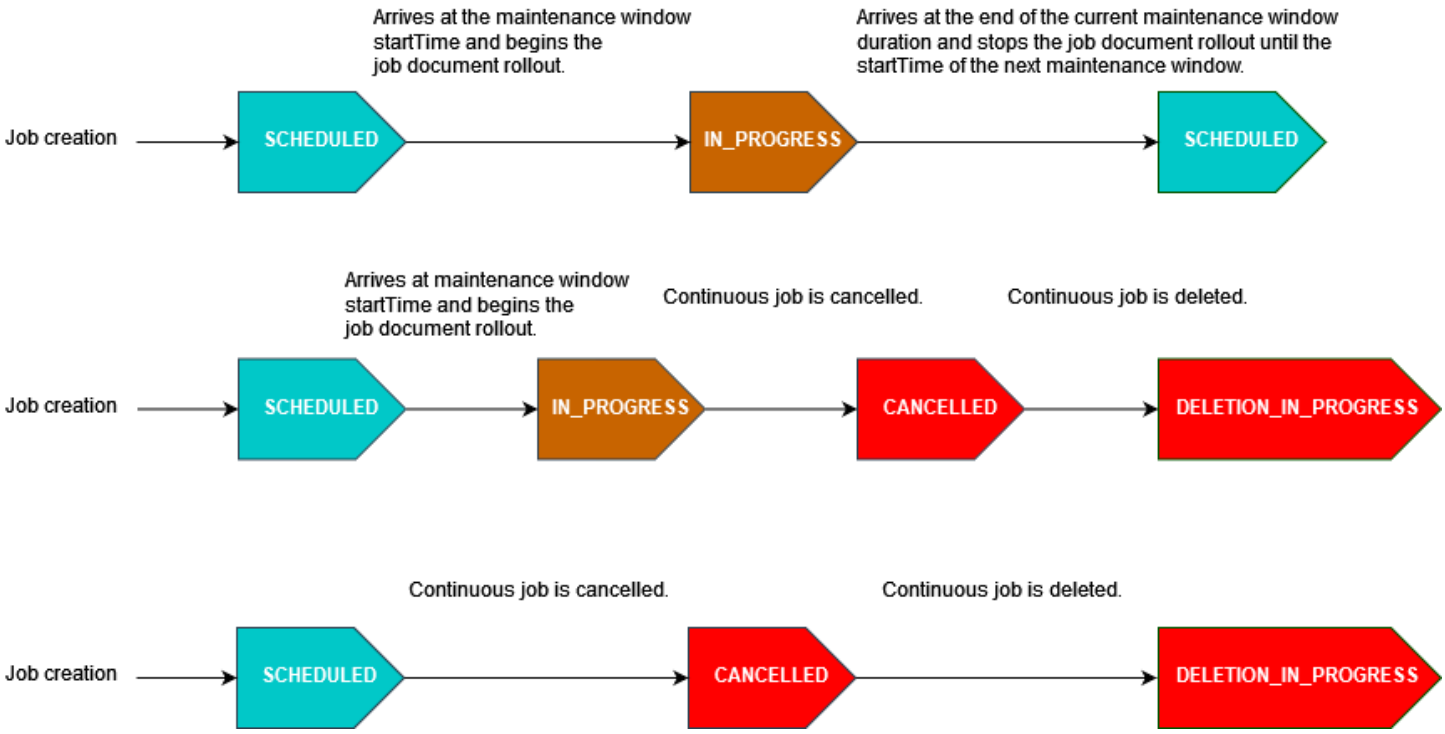
下表列出了计划任务的常见持续时间场景：

计划任务示例编号	startTime	endTime	最长持续时间
1	在初始任务创建后立即。	在初始任务创建后一年。	一年
2	在初始任务创建后一个月。	在初始任务创建后 13 个月。	一年
3	在初始任务创建后一年。	在初始任务创建后两年。	一年
4	在初始任务创建后立即。	在初始任务创建后两年。	两年

定期维护时段

维护时段是 AWS Management Console CreateJob和 CreateJobTemplate API 计划配置SchedulingConfig中的可选配置。您可以设置定期维护时段，其中包含预先确定的开始时间、持续时间和维护时段发生的频率（每天、每周或每月）。维护时段仅适用于连续任务。定期维护时段的最长持续时间为 23 小时 50 分钟。

下图显示了具有可选维护时段的各种计划任务场景的任务状态：



有关任务状态的更多信息，请参阅任务和任务执行状态。

> **Note**
>
> 如果任务在维护时段内达到 `endTime`，它将从 `IN_PROGRESS` 更新为 `COMPLETED`。此外，任何剩余的任务执行都将遵循任务的 `endBehavior`。

Cron 表达式

对于在维护时段内以自定义频率推出任务文档的计划任务，使用 cron 表达式输入自定义频率。Cron 表达式有六个必填字段，字段之间以空格分隔。

语法

```
cron(fields)
```

字段	值	通配符
分钟	0-59	, - * /
小时	0-23	, - * /

字段	值	通配符
D ay-of-month	1-31	, - * ? / L W
月	1-12 或 JAN-DEC	, - * /
D ay-of-week	1-7 或 SUN-SAT	, - * ? L #
年	1970-2199	, - * /

通配符

- , (逗号) 通配符包含其他值。在“月份”字段中，JAN、FEB 和 MAR 将包含 January、February 和 March。
- - (破折号) 通配符用于指定范围。在“日”字段中，1-15 将包含指定月份的 1 - 15 日。
- * (星号) 通配符包含该字段中的所有值。在“小时”字段中，* 将包含每个小时。不能同时在 D ay-of-month 和 D ay-of-week 字段中使用 *。如果您在一个中使用它，则必须在另一个中使用 ？。
- / (正斜杠) 通配符用于指定增量。在“分钟”字段中，您可以输入 1/10 以指定从一个小时的第一分钟开始的每个第十分钟 (例如，第 11 分钟、第 21 分钟和第 31 分钟，依此类推)。
- ? (问号) 通配符用于指定一个或另一个。在 D ay-of-month 字段中，你可以输入 7，如果你不在乎 7 号是一周中的哪一天，你可以输入？ 在 D ay-of-week 字段中。
- D ay-of-month 或 D ay-of-week 字段中的 L 通配符指定一个月或一周的最后一天。
- D ay-of-month 字段中的**W**通配符指定工作日。在 D ay-of-month 字段中，**3W**指定最接近该月第三天的工作日。
- D ay-of-week 字段中的 # 通配符指定一个月内一周中指定某一天的特定实例。例如，3#2 指该月的第二个星期二：3 指的是星期二，因为它是每周的第三天，2 是指该月内该类型的第二天。

> **Note**
>
> 如果使用 '#' 字符，则只能在 day-of-week 字段中定义一个表达式。例如，"3#1,6#3" 是无效的，因为它被解释为两个表达式。

限制

- 您不能在同一 cron 表达式中指定 D ay-of-month 和 D ay-of-week 字段。如果您在其中一个字段中指定了值（或一个 *），则必须在另一个字段中使用 ?。

示例

在为定期维护时段的 `startTime` 使用 cron 表达式时，请参阅以下示例 cron 字符串。

分钟	小时	日期	月份	星期几	年	含义
0	10	*	*	?	*	每天上午的 10:00 (UTC) 运行
15	12	*	*	?	*	每天在下午 12:15 (UTC) 运行
0	18	?	*	MON-FRI	*	每星期一到星期五的下午 6:00 (UTC) 运行
0	8	1	*	?	*	每月第 1 天上午 8:00 (UTC) 运行

定期维护时段持续时间结束逻辑

当维护时段期间的任务推出达到当前维护时段事件持续时间结束时，将发生以下操作：

- 任务将停止所有向目标组中的任何剩余设备推出任务文档的过程。它将在下一个维护时段的 `startTime` 继续运行。
- 所有状态为 QUEUED 的任务执行都将保持 QUEUED 状态，直到下一个维护时段事件的 `startTime` 为止。在下一个时段中，当设备准备好开始执行在任务文档中指定的操作时，任务可以切换到 `IN_PROGRESS`。

- 所有状态为 IN_PROGRESS 的任务执行都将继续执行在任务文档中指定的操作，直到它们达到最终状态。JobExecutionsRetryConfig 中指定的任何重试都将在下一个维护时段的 startTime 进行。

任务中止配置

使用此配置可创建条件，以便设备的阈值百分比符合该条件时取消任务。例如，在以下情况，您可以使用此配置取消任务：

- 当设备的某个阈值百分比没有收到任务执行通知时，例如设备与空中下载（OTA）更新不兼容。在这种情况下，设备可以报告 REJECTED 状态。
- 设备的阈值百分比报告其任务执行失败时，例如设备尝试从 Amazon S3 URL 下载任务文档时发生连接中断。在这种情况下，设备必须进行编程，才能向 AWS IoT报告 FAILURE 状态。
- 任务执行开始后，由于任务执行超时达到设备的某个阈值百分比而报告 TIMED_OUT 状态时。
- 出现多次重试失败时。添加重试配置时，每次重试都可能让您的 AWS 账户产生额外费用。在这种情况下，取消任务可以取消排队的任务执行，避免重试这些执行。有关重试配置以及其与中止配置搭配使用的更多信息，请参阅 任务执行超时和重试配置。

您可以使用 AWS IoT 控制台或 Jobs API 设置 AWS IoT 任务中止条件。

任务执行超时和重试配置

当任务执行的时间超过设置的持续时间时，系统使用任务执行超时配置向您发送 任务通知。如果任务失败或超时，系统会使用任务执行重试配置来重试执行。

任务执行超时配置

只要任务执行卡在 IN_PROGRESS 状态的时间超出预期，系统便会使用任务执行超时配置通知您。任务处于 IN_PROGRESS 状态时，您可以监控任务执行的进度。

任务超时计时器

有两种类型的计时器：进行中计时器和步骤计时器。

进行中计时器

创建任务或任务模板时，您可以为进行中计时器指定介于 1 分钟到 7 天之间的值。在任务执行开始之前，您可以更新此计时器的值。计时器启动后便无法更新，且计时器值应用于任务的所有任务执

行。每当任务执行保持IN_PROGRESS状态的时间超过此时间间隔时，任务执行就会失败并切换到终端TIMED_OUT状态。 AWS IoT 还会发布 MQTT 通知。

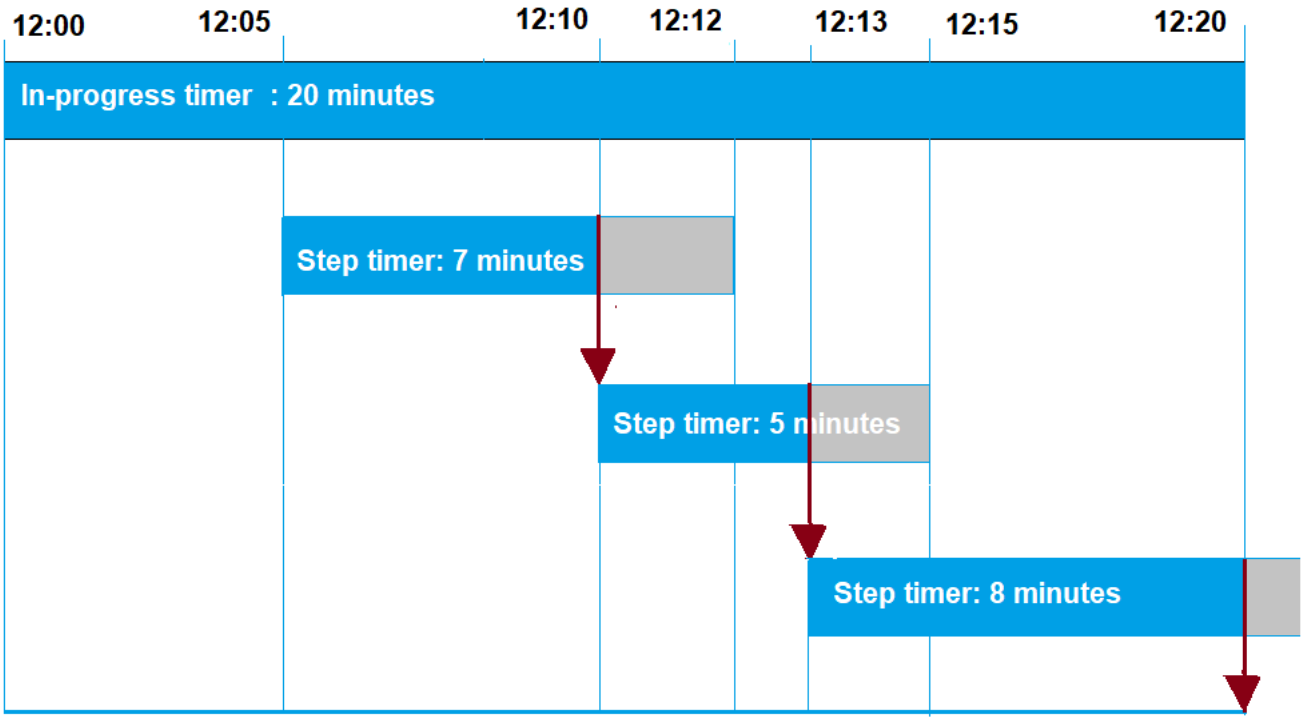
步骤计时器

您还可以设置仅适用于要更新的任务执行的步骤计时器。这种计时器不会对进行中计时器产生影响。您每次更新任务执行时，都可以为此步骤计时器设置新值。为事物启动下一个待处理任务执行时，您还可以创建新的步骤计时器。如果任务执行保持在 IN_PROGRESS 状态的时间长度超过了此步骤计时器间隔，它将失败，并切换为最终 TIMED_OUT 状态。

> **Note**
>
> 您可以使用 AWS IoT 控制台或 AWS IoT Jobs API 设置进行中的计时器。若要指定步骤计时器，请使用 API。

任务超时计时器工作原理

下图说明了进行中超时计时器与步骤超时计时器在 20 分钟超时期限内相互作用的方式。



下面介绍了不同步骤：

1. 12:00

此时会创建一个新任务，还会在创建任务时启动 20 分钟的进行中计时器。进行中计时器开始运行，任务执行切换到 IN_PROGRESS 状态。

2. 12:05 PM

   此时会创建一个值为 7 分钟的新步骤计时器。任务执行现在会在 12:12 PM 超时。

3. 12:10 PM

   此时会创建一个值为 5 分钟的新步骤计时器。创建新步骤计时器后，系统会丢弃旧步骤计时器，任务执行现在会在 12:15 PM 超时。

4. 12:13 PM

   此时会创建一个值为 9 分钟的新步骤计时器。系统会丢弃旧步骤计时器，任务执行现在会在 12:20 PM 超时，因为进行中计时器会在 12:20 PM 超时。步骤计时器不能超过进行中计时器的绝对界限。

任务执行重试配置

您可以在满足某组条件时使用重试配置来重试任务执行。如果任务超时或设备报告失败，就可以尝试重试。如果要因超时错误重试执行，则必须启用超时配置。

如何使用重试配置

使用以下步骤重试配置：

1. 确定是否对 FAILED、TIMED_OUT 或这两种失败条件使用重试配置。对于TIMED_OUT状态，在报告状态后， AWS IoT Jobs 会自动重试设备执行任务。
2. 对于 FAILED 状态，请检查是否可以重试任务执行失败。如果可以重试，请对设备进行编程，以便向 AWS IoT报告 FAILURE 状态。以下部分介绍了有关可重试和不可重试失败的更多信息。
3. 使用前述的信息指定每种失败类型的重试次数。最多可以为单台设备的两种失败类型组合指定 10 次重试。当执行成功或达到指定的尝试次数时，重试尝试会自动停止。
4. 添加中止配置，以便在重复出现重试失败时取消任务，以避免大量重试产生额外费用。

> **Note**
>
> 当任务达到定期维护时段事件结束时，所有 IN_PROGRESS 任务执行都将继续执行在任务文档中确定的操作，直到它们达到最终状态。如果任务执行在维护时段之外达到最终状态 FAILED 或 TIMED_OUT，则当尝试次数未耗尽时，将在下一个时段中尝试进行重试。在下一个维护时

段事件的 `startTime`，将创建一个新的任务执行并进入 QUEUED 状态，直到设备准备好开始为止。

重试和中止配置

每次重试都会向您收取额外费用。 AWS 账户为避免重复出现的重试失败产生额外费用，我们建议添加中止配置。有关定价的更多信息，请参阅 AWS IoT Device Management 定价。

当设备的高阈值百分比超时或报告失败时，您就可能遇到多次重试失败。在这种情况下，您可以使用中止配置取消任务，避免任何排队的任务执行或更多重试尝试。

> **Note**
>
> 符合取消任务执行的中止条件仅会取消 QUEUED 任务执行。系统不会尝试设备的任何排队重试。不过，当前具有 IN_PROGRESS 状态的任务执行不会遭到取消。

在重试失败的任务执行之前，我们还建议您检查任务执行失败是否可以重试，如下面的部分所述。

FAILED 失败类型重试

若要重试 FAILED 失败类型，就必须对设备进行编程，以便向 FAILURE 报告失败任务执行 AWS IoT 状态。使用重试 FAILED 任务执行的条件来设置重试配置，并指定要执行的重试次数。当 AWS IoT Jobs 检测到FAILURE状态时，它将自动尝试重试设备执行作业。任务执行成功或达到最大重试次数，重试才会停止。

您可以跟踪每次重试以及在这些设备上运行的任务。通过跟踪执行状态，您可以在达到指定的重试次数后，使用设备报告失败以及发起另一次重试。

可重试和不可重试失败

任务执行失败可以是可重试或不可重试的失败。每次重试都会让您的 AWS 账户产生费用。为避免因多次重试而产生额外费用，请先检查任务执行失败是否可以重试。可重试失败示例包括设备尝试从 Amazon S3 URL 下载任务文档时发生的连接错误。如果任务执行失败是可重试的，可对设备进行编程，以便在出现任务执行失败时报告 FAILURE 状态。然后，将重试配置设置为重试 FAILED 执行。

如果执行不可重试，为避免重试以及可能让您的账户产生额外费用，我们建议您对设备进行编程以向 AWS IoT报告 REJECTED 状态。不可重试失败的示例包括：设备与接收任务更新不兼容，或

执行任务时遇到内存错误。在这些情况下， AWS IoT Jobs 不会重试任务执行，因为它只有在检测到FAILED或TIMED_OUT状态时才会重试任务执行。

在确定任务执行失败可重试后，如果重试仍然失败，请考虑检查设备日志。

> **Note**
>
> 当具有可选计划配置的任务达到其 endTime 时，所选的 endBehavior 将停止向目标组中的所有剩余设备推出任务文档，并指示如何继续剩余的任务执行。如果通过重试配置选择了这些尝试，则会重试这些尝试。

TIMEOUT 失败类型重试

如果您在创建任务时启用超时，则 AWS IoT 当状态从IN_PROGRESS变为时，任务将尝试重试设备执行任务。TIMED_OUT如果进行中计时器超时，或者您指定的步骤计时器处于 IN_PROGRESS 中然后超时，此状态可能会发生更改。任务执行成功或达到此失败类型的最大重试次数后，重试才会停止。

持续任务和事物组成员资格更新

对于任务状态为 IN_PROGRESS 的连续任务，当事物的组成员资格更新时，重试次数将重置为零。例如，假设您指定了五次重试，并且已经执行了三次重试。如果现已从事物组中删除事物，然后该事物重新加入该组（例如动态事物组），则重试次数将重置为零。您现在可以对事物组执行五次重试，而非剩余的两次重试。此外，从事物组中删除事物将取消其他重试次数。

指定其他配置

您可以在创建任务或任务模板时指定其他配置。下文介绍了何时可以指定这些配置。

- 创建自定义任务模板时。从模板创建任务时，系统会保存您指定的其他配置设置。
- 使用任务文件创建自定义任务时。任务文件可以是在 S3 存储桶中上传的 JSON 文件。
- 使用自定义任务模板创建自定义任务时。如果模板已经指定了这些设置，则可以重复使用，也可以通过指定新的配置设置将其覆盖。
- 使用 AWS 托管模板创建自定义作业时。

主题

- 使用 AWS Management Console指定任务配置
- 使用 AWS IoT Jobs API 指定任务配置

使用 AWS Management Console指定任务配置

您可以使用 AWS IoT 控制台为任务添加不同的配置。创建任务后，您可以在任务详细信息页面上查看任务配置的状态详细信息。有关不同配置及其工作方式的更多信息，请参阅 任务配置的工作原理。

创建任务或任务模板时添加任务配置。

创建自定义任务模板

创建自定义任务模板时指定推出配置

1. 转到AWS IoT 控制台的作业模板中心，然后选择创建作业模板。
2. 指定任务模板属性、提供任务文档、展开要添加的配置，然后指定配置参数。

创建自定义任务

创建自定义任务时指定推出配置

1. 前往AWS IoT 控制台的 Job 中心并选择 C reate job。
2. 请选择 Create a custom job（创建自定义任务），指定任务属性、目标，然后指定任务文档使用任务文件还是模板。您可以使用自定义模板或 AWS 托管模板。
3. 请选择任务配置，展开 Rollout configuration（推出部署），再在其中指定使用 Constant rate（恒定速率）或 Exponential rate（指数速率）。然后，指定配置参数。

下一部分介绍可以为每个配置指定的参数。

推出配置

您可以指定使用恒定推出速率还是指数速率。

- 设置恒定推出速率

  要为任务执行设置固定速率，请选择固定速率，然后为速率上限指定每分钟最大值。此值为可选值，范围从 1 到 1000。如果不设置该值，则会使用 1000 作为默认值。

- 设置指数推出速率

  若要设置指数速率，请选择 Exponential rate（指数速率），然后指定这些参数：

  - 每分钟的基本速率

指通知的设备数或成功的设备数量阈值满足速率提高标准之前，执行任务的速率。

- 增量因子

  指 Number of notified devices（通知设备数）或 Number of succeeded devices（成功设备数）阈值满足 Rate increase criteria（速率增加条件）之后，推出速率增加的指数因子。

- 速率增加条件

  Number of notified devices（通知设备数）或 Number of succeeded devices（成功设备数）的阈值。

中止配置

请选择 Add new configuration（添加新配置），然后为每个配置指定以下参数：

- 失败类型

  指定启动任务中止的失败类型。其中包括 FAILED（失败）、REJECTED（已被拒绝）、TIMED_OUT（超时）或 ALL（全部）。

- 增量因子

  指定在满足任务中止条件之前必须完成的任务执行数量。

- 阈值百分比

  指定启动任务中止的已执行事物的总数。

计划配置

每项任务可以在初始创建时立即启动，计划在以后的日期和时间启动，也可以在定期维护时段进行。

请选择 Add new configuration（添加新配置），然后为每个配置指定以下参数：

- 任务开始

  指定任务开始的日期和时间。

- 定期维护时段

  定期维护时段定义了任务可以将任务文档部署到任务中的目标设备的具体日期和时间。维护时段可以按每天、每周、每月重复，也可以按自定义日期和时间重复。

- 任务结束

  指定任务结束的日期和时间。

- 任务结束行为

  为所有未完成的任务执行选择任务结束时的结束行为。

> **Note**
>
> 当具有可选计划配置和所选结束时间的任务达到结束时间时，该任务将停止向目标组中的所有剩余设备进行推出。它还利用所选的结束行为，来决定如何继续剩余的任务执行以及根据重试配置进行重试。

超时配置

默认无超时，任务运行会遭到取消或删除。要使用超时，请选择启用超时，然后指定 1 分钟到 7 天之间的超时值。

重试配置

> **Note**
>
> 创建任务后，便无法更新重试次数。您只能删除所有失败类型的重试配置。创建任务时，请考虑为配置采用适当的重试次数。为避免因潜在的重试失败产生额外成本，请添加中止配置。

请选择 Add new configuration（添加新配置），然后为每个配置指定以下参数：

- 失败类型

  指定应触发任务执行重试的失败类型。其中包括 Failed（失败）、Timeout（超时）和 All（全部）。

- 重试次数

  请为所选择的 Failure type（失败类型）指定重试次数。最多可以为两种失败类型组合指定 10 次重试。

使用 AWS IoT Jobs API 指定任务配置

您可以使用CreateJob或 CreateJobTemplateAPI 来指定不同的任务配置。以下各节介绍如何添加这些配置。添加配置后，您可以使用JobExecutionSummary和JobExecutionSummaryForJob来查看其状态。

有关不同配置及其工作方式的更多信息，请参阅 任务配置的工作原理。

推出配置

您可以为推出配置指定恒定推出速率或指数推出速率。

- 设置恒定推出速率

  若要设置恒定推出速率，请使用 JobExecutionsRolloutConfig 对象将 maximumPerMinute 参数添加到 CreateJob 请求。该参数会指定任务执行的速率上限。此值为可选值，范围从 1 到 1000。如果不设置该值，则会使用 1000 作为默认值。

  ```
      "jobExecutionsRolloutConfig": {
          "maximumPerMinute": 1000
      }
  ```

- 设置指数推出速率

  若要设置任务可变推出速率，请使用 JobExecutionsRolloutConfig 对象。您可以在运行 ExponentialRolloutRate API 操作时配置 CreateJob 属性。以下示例使用 exponentialRate 参数设置指数推出速率。有关这些参数的更多信息，请参阅ExponentialRolloutRate。

  ```
  {
  ...
    "jobExecutionsRolloutConfig": {
      "exponentialRate": {
        "baseRatePerMinute": 50,
        "incrementFactor": 2,
        "rateIncreaseCriteria": {
          "numberOfNotifiedThings": 1000,
          "numberOfSucceededThings": 1000
        },
        "maximumPerMinute": 1000
      }
    }
  ```

```
...
}
```

其中，参数：

baseRatePer分钟

指定达到 numberOfNotifiedThings 或 numberOfSucceededThings 阈值之前任务的执行速率。

incrementFactor

指定达到 numberOfNotifiedThings 或 numberOfSucceededThings 阈值之后推出速率增加的指数因子。

rateIncreaseCriteria

指定 numberOfNotifiedThings 或 numberOfSucceededThings 阈值。

中止配置

要使用 API 添加此配置，请在运行 CreateJob 或 CreateJobTemplate API 操作时指定 AbortConfig 参数。以下示例介绍了任务推出的中止配置，该任务经历了如 CreateJob API 操作指定的多次失败执行。

> **Note**
>
> 删除任务执行会影响已完成执行总数的计算值。当任务中止时，服务会创建自动的 comment 和 reasonCode 以将用户驱动的取消与任务中止取消区分开来。

```
  "abortConfig": {
     "criteriaList": [
        {
           "action": "CANCEL",
           "failureType": "FAILED",
           "minNumberOfExecutedThings": 100,
           "thresholdPercentage": 20
        },
        {
           "action": "CANCEL",
```

```
            "failureType": "TIMED_OUT",
            "minNumberOfExecutedThings": 200,
            "thresholdPercentage": 50
        }
      ]
    }
```

其中，参数：

action

指定满足中止条件时要采取的操作。此参数是必需的，并且 CANCEL 是唯一的有效值。

failureType

指定哪种失败类型应启动任务中止。有效值为 FAILED、REJECTED、TIMED_OUT 和 ALL。

minNumberOfExecutedThings

指定在满足任务中止条件之前必须完成的任务执行数量。在此示例中， AWS IoT 直到至少 100 个设备已完成任务执行时才检查是否应执行任务中止。

thresholdPercentage

指定可启动任务中止的已执行任务的事物总数。在此示例中，如果达到阈值百分比，则按顺序 AWS IoT 检查并启动任务中止。如果完成 100 次执行后至少有 20% 的完整执行失败，则会取消任务推出。如果不符合此标准， AWS IoT 则检查是否有至少 50% 的已完成执行在 200 次执行完成后超时。如果是这样，则会取消任务推出。

计划配置

要使用此 API 添加此配置，请在运行 CreateJob 或 CreateJobTemplate API 操作时指定可选的 SchedulingConfig。

```
  "SchedulingConfig": {
    "endBehavior": string
    "endTime": string
    "maintenanceWindows": string
    "startTime": string
  }
```

其中，参数：

startTime

指定任务开始的日期和时间。

endTime

指定任务结束的日期和时间。

maintenanceWindows

指定是否为计划任务选择了可选维护时段，以便将任务文档推出到目标组中的所有设备。maintenanceWindow 的字符串格式为 YYYY/MM/DD（表示日期）和 hh: mm（表示时间）。

endBehavior

指定计划任务到达 endTime 时的任务行为。

> **Note**
>
> 任务的可选 SchedulingConfig 可在 DescribeJob 和 DescribeJobTemplate API 中查看。

超时配置

要使用 API 添加此配置，请在运行 CreateJob 或 CreateJobTemplate API 操作时指定 TimeoutConfig 参数。

使用超时配置

1. 要在创建作业或作业模板时设置正在进行的计时器，请为可选TimeoutConfig对象的inProgressTimeoutInMinutes属性设置一个值。

```
 "timeoutConfig": {
   "inProgressTimeoutInMinutes": number
}
```

2. 要为任务执行指定步数计时器，请为调用stepTimeoutInMinutes时设置一个值UpdateJobExecution。步骤计时器仅应用于您更新的任务执行。您每次更新任务执行时，可以为此计时器设置新值。

> **Note**
>
> UpdateJobExecution 可以通过创建新的值为 -1 的步骤计时器来丢弃已经创建的步骤计时器。

```
{
   ...
    "statusDetails": {
      "string" : "string"
   },
   "stepTimeoutInMinutes": number
}
```

3. 要创建新的步数计时器，您也可以调用 StartNextPendingJobExecutionAPI 操作。

重试配置

> **Note**
>
> 创建任务时，请考虑为配置采用适当的重试次数。为避免因潜在的重试失败产生额外成本，请添加中止配置。创建任务后，便无法更新重试次数。您只能使用 UpdateJobAPI 操作将重试次数设置为 0。

要使用 API 添加此配置，请在运行 CreateJob 或 CreateJobTemplate API 操作时指定 jobExecutionsRetryConfig 参数。

```
{
...
  "jobExecutionsRetryConfig": {
      "criteriaList": [
         {
            "failureType": "string",
            "numberOfRetries": number
         }
      ]
  }
...
```

```
}
```

其中 criteriaList 是指定条件列表的数组，用于确定任务的每种失败类型允许的重试次数。

设备和任务

设备可以使用 MQTT、HTTP 签名版本 4 或 HTTP TLS 与 AWS IoT 任务通信。要确定您的设备与 AWS IoT Jobs 通信时要使用的端点，请运行DescribeEndpoint命令。例如，如果您运行此命令：

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

您会获得与以下内容相似的结果：

```
{
    "endpointAddress": "a1b2c3d4e5f6g7-ats.iot.us-west-2.amazonaws.com"
}
```

使用 MQTT 协议

设备可以使用 MQTT 协议与 AWS IoT 任务通信。设备订阅 MQTT 主题是为了收到新任务的通知并接收来自 AWS IoT 任务服务的响应。设备在 MQTT 主题上发布以查询或更新任务启动的状态。每台设备都有自己的一般 MQTT 主题。有关发布和订阅 MQTT 主题的更多信息，请参阅 设备通信协议。

通过这种通信方式，您的设备使用其设备专用证书和私钥向 Jobs 进行 AWS IoT 身份验证。

您的设备可以订阅以下主题。`thing-name` 是与设备关联的事物名称。

- **`$aws/things/thing-name/jobs/notify`**

  订阅此主题，以便在待处理任务启动列表中添加或删除任务启动时接收通知。
- **`$aws/things/thing-name/jobs/notify-next`**

  订阅此主题，以在下一个待处理任务执行发生更改时通知您。
- **`$aws/things/thing-name/jobs/request-name/accepted`**

  AWS IoT 作业服务发布有关 MQTT 主题的成功和失败消息。通过将 `accepted` 或 `rejected` 追加到用于发出请求的主题来构成该主题。这里，`request-name`是请求的名称，例如`Get`，主题可以是：`$aws/things/myThing/jobs/get`。 AWS IoT 然后，Jobs 会发布有关该`$aws/things/myThing/jobs/get/accepted`主题的成功消息。

- **$aws/things/*thing-name*/jobs/*request-name*/rejected**

  此处，request-name 是请求的名称，例如 Get。如果请求失败， AWS IoT Jobs 将发布有关该$aws/things/myThing/jobs/get/rejected主题的失败消息。

您还可以使用以下 HTTPS API 操作：

- 通过调用 UpdateJobExecution API 来更新任务执行的状态。
- 通过调用 DescribeJobExecution API 来查询任务执行的状态。
- 通过调用 GetPendingJobExecutions API 来检索待处理任务执行的列表。
- 通过使用 jobId 作为 $next 调用 DescribeJobExecution API 来检索下一个待定任务执行。
- 通过调用 StartNextPendingJobExecution API 来获取和开启下一个待处理任务执行。

使用 HTTP Signature 版本 4

设备可以在端口 443 上使用 HTTP 签名版本 4 与 AWS IoT Job 通信。这是由 AWS SDK 和 CLI 使用的方法。有关这些工具的更多信息，请参阅AWS CLI 命令参考： iot-jobs-data或AWS 软件开发工具包和工具，并参阅您的首选语言 IotJobsDataPlane 部分。

通过这种通信方式，您的设备使用 IAM 凭证通过 AWS IoT Jobs 进行身份验证。

可以使用此方法执行以下命令：

- DescribeJobExecution

  ```
  aws iot-jobs-data describe-job-execution ...
  ```

- GetPendingJobExecutions

  ```
  aws iot-jobs-data get-pending-job-executions ...
  ```

- StartNextPendingJobExecution

  ```
  aws iot-jobs-data start-next-pending-job-execution ...
  ```

- UpdateJobExecution

  ```
  aws iot-jobs-data update-job-execution ...
  ```

使用 HTTP TLS

设备可以使用支持该协议的第三方软件客户端，在端口 8443 上使用 HTTP TLS 与 AWS IoT Jobs 通信。

利用此方法，您的设备可使用基于 X.509 证书的身份验证（例如，其特定于设备的证书和私有密钥）。

可以使用此方法执行以下命令：

- DescribeJobExecution
- GetPendingJobExecutions
- StartNextPendingJobExecution
- UpdateJobExecution

对设备进行编程以使用任务

本部分中的示例使用 MQTT 来演示设备如何使用 AWS IoT Jobs 服务。或者，您可以使用相应的 API 或 CLI 命令。对于这些示例，我们假定名为 MyThing 的设备订阅以下 MQTT 主题：

- `$aws/things/MyThing/jobs/notify` –、 或 `$aws/things/MyThing/jobs/notify-next`
- `$aws/things/MyThing/jobs/get/accepted`
- `$aws/things/MyThing/jobs/get/rejected`
- `$aws/things/MyThing/jobs/jobId/get/accepted`
- `$aws/things/MyThing/jobs/jobId/get/rejected`

如果您使用代码签名 AWS IoT，则您的设备代码必须验证代码文件的签名。该签名在任务文档中的 codesign 属性内。有关验证代码文件签名的更多信息，请参阅设备代理示例。

主题

- 设备工作流程
- 任务流
- 任务通知

设备工作流程

设备可以使用以下任一方式处理它运行的任务。

- 获取下一个任务

  1. 当设备首次联机时，它应订阅设备的 `notify-next` 主题。
  2. 使用 jobId `$next` 调用 DescribeJobExecution MQTT API 以获取下一个任务、其任务文档和其他详细信息，包括保存在 `statusDetails` 中的任何状态。如果任务文档具有代码文件签名，则您必须验证签名，然后才能继续处理任务请求。
  3. 调用 UpdateJobExecution MQTT API 以更新任务状态。或者，要在一个调用中将此步骤与上一步骤合并，设备可调用 StartNextPendingJobExecution。
  4. （可选）您可在调用 UpdateJobExecution 或 StartNextPendingJobExecution 时为 `stepTimeoutInMinutes` 设置值来添加步骤计时器。
  5. 使用 UpdateJobExecution MQTT API 执行任务文档所指定的操作以报告任务的进度。
  6. 通过使用此 jobId 调用 DescribeJobExecution MQTT API 来继续监控任务执行。如果删除了任务执行，DescribeJobExecution 将返回 `ResourceNotFoundException`。

     如果在设备运行任务时取消或删除了任务执行，则设备应能够恢复到有效状态。
  7. 在任务完成后调用 UpdateJobExecution MQTT API 以更新任务状态并报告成功或失败。
  8. 由于此任务的执行状态已更改为最终状态，因此，可用于执行的下一个任务（如果有）将发生更改。设备将收到下一个待处理任务执行已更改的通知。此时，设备将按照步骤 2 中所述继续操作。

  如果设备保持联机状态，则将继续收到下一个待处理任务执行的通知。这包括它在完成任务或添加新的待处理任务执行时的任务执行数据。在发生此情况时，设备将按照步骤 2 中所述继续操作。

- 从可用任务中选择

  1. 当设备首次联机时，它应订阅事物的 `notify` 主题。
  2. 调用 GetPendingJobExecutions MQTT API 以获取待处理任务执行的列表。
  3. 如果列表中包含一个或多个任务执行，请选择一个任务执行。
  4. 调用 DescribeJobExecution MQTT API 以获取任务文档和其它详细信息，包括保存在 `statusDetails` 中的任何状态。
  5. 调用 UpdateJobExecution MQTT API 以更新任务状态。如果在此命令中将 `includeJobDocument` 字段设置为 `true`，则设备可跳过上一个步骤并在此时检索任务文档。

6. （可选）您可在调用 UpdateJobExecution 时为 `stepTimeoutInMinutes` 设置值来添加步骤计时器。
7. 使用 UpdateJobExecution MQTT API 执行任务文档所指定的操作以报告任务的进度。
8. 通过使用此 jobId 调用 DescribeJobExecution MQTT API 来继续监控任务执行。如果在设备运行任务时取消或删除了任务执行，则设备应能够恢复到有效状态。
9. 在任务完成后调用 UpdateJobExecution MQTT API 以更新任务状态并报告成功或失败。

如果设备保持联机状态，则在一个新的待处理任务执行变为可用时，它将收到所有待处理任务执行的通知。在发生此情况时，设备可按照步骤 2 中所述继续操作。

如果设备无法执行任务，它应调用 UpdateJobExecution MQTT API 来将任务状态更新为 `REJECTED`。

任务流

下面显示了任务工作流程中的不同步骤，从启动新任务到报告任务执行的完成状态。

开始新任务

创建新任务后，J AWS IoT obs 会为每台目标设备发布一条有关该`$aws/things/thing-name/jobs/notify`主题的消息。

消息包含以下信息：

```
{
    "timestamp":1476214217017,
    "jobs":{
        "QUEUED":[{
            "jobId":"0001",
            "queuedAt":1476214216981,
            "lastUpdatedAt":1476214216981,
            "versionNumber" : 1
        }]
    }
}
```

在对任务执行进行排队时，设备将在 '`$aws/things/thingName/jobs/notify`' 主题上收到此消息。

> **Note**
>
> 对于具有可选 SchedulingConfig 的任务，该任务将保持初始状态 SCHEDULED。当任务达到所选的 startTime 时，将发生以下情况：
>
> - 任务状态将更新为 IN_PROGRESS。
> - 任务将开始向目标组中的所有设备推出任务文档。

获取任务信息

要获取有关任务执行的更多信息，请调用 DescribeJobExecution MQTT API 并将 includeJobDocument 字段设置为 true（默认值）。

如果请求成功，AWS IoT 作业服务将发布一条有关该$aws/things/MyThing/jobs/0023/get/accepted主题的消息：

```
{
    "clientToken" : "client-001",
    "timestamp" : 1489097434407,
    "execution" : {
        "approximateSecondsBeforeTimedOut": number,
        "jobId" : "023",
        "status" : "QUEUED",
        "queuedAt" : 1489097374841,
        "lastUpdatedAt" : 1489097374841,
        "versionNumber" : 1,
        "jobDocument" : {
            < contents of job document >
        }
    }
}
```

如果请求失败，AWS IoT 作业服务将发布一条有关该$aws/things/MyThing/jobs/0023/get/rejected主题的消息。

设备现在已有任务文档，它可使用该文档执行任务的远程操作。如果任务文档包含一个 Amazon S3 预签名 URL，则设备可使用该 URL 下载任务的任何所需文件。

报告任务执行状态

当设备执行任务时，它可调用 UpdateJobExecution MQTT API 来更新任务执行的状态。

例如，设备可通过在 IN_PROGRESS 主题上发布以下消息来将任务执行状态更新为 $aws/things/MyThing/jobs/0023/update：

```
{
    "status":"IN_PROGRESS",
    "statusDetails": {
        "progress":"50%"
    },
    "expectedVersion":"1",
    "clientToken":"client001"
}
```

Jobs 通过将消息发布到 $aws/things/MyThing/jobs/0023/update/accepted 或 $aws/things/MyThing/jobs/0023/update/rejected 主题来进行响应：

```
{
    "clientToken":"client001",
    "timestamp":1476289222841
}
```

设备可通过调用 StartNextPendingJobExecution 来合并两个以前的请求。这将获取并开始下一个待处理任务执行，并允许设备更新任务执行状态。此请求还在有待处理任务执行时返回任务文档。

如果作业包含 TimeoutConfig，则正在进行的计时器开始运行。您还可以通过设置调用时的值来为任务执行设置步进计stepTimeoutInMinutes时器UpdateJobExecution。步骤计时器仅应用于您更新的任务执行。您每次更新任务执行时，可以为此计时器设置新值。您也可以在呼叫时创建计步器StartNextPendingJobExecution。如果任务执行保持在 IN_PROGRESS 状态的时间长度超过了此步骤计时器间隔，它将失败，并切换为最终 TIMED_OUT 状态。步骤计时器对您在创建任务时设置的进行中计时器没有任何影响。

status 字段可设置为 IN_PROGRESS、SUCCEEDED 或 FAILED。您无法更新已处于最终状态的任务执行的状态。

报告执行已完成

在设备执行完任务后，它将调用 UpdateJobExecution MQTT API。如果任务已成功，则将 status 设置为 SUCCEEDED，并在消息负载的 statusDetails 中，将有关任务的其它信息作为名称-值对添加。任务执行完成时，进行中计时器和步骤计时器结束。

例如：

```
{
    "status":"SUCCEEDED",
    "statusDetails": {
        "progress":"100%"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

如果任务未成功，则将 status 设置为 FAILED，并在 statusDetails 中，添加有关出现的错误的信息：

```
{
    "status":"FAILED",
    "statusDetails": {
        "errorCode":"101",
        "errorMsg":"Unable to install update"
    },
    "expectedVersion":"2",
    "clientToken":"client-001"
}
```

> **Note**
>
> statusDetails 属性可包含任意数量的名称-值对。

当 AWS IoT Jobs 服务收到此更新时，它会发布一条有关该$aws/things/MyThing/jobs/notify主题的消息，表示任务执行已完成：

```
{
    "timestamp":1476290692776,
    "jobs":{}
```

```
}
```

其它任务

如果设备有其它待处理的任务执行，这些任务执行将包含在发布到 $aws/things/MyThing/jobs/notify 的消息中。

例如：

```
{
    "timestamp":1476290692776,
    "jobs":{
        "QUEUED":[{
            "jobId":"0002",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }],
        "IN_PROGRESS":[{
            "jobId":"0003",
            "queuedAt":1476290646230,
            "lastUpdatedAt":1476290646230
        }]
    }
}
```

任务通知

当任务处于待处理状态或列表中的第一个任务执行发生变化时， AWS IoT 作业服务会向保留主题发布 MQTT 消息。设备可通过订阅这些主题来跟踪待处理任务。

任务通知类型。

任务通知将作为 JSON 负载发布到 MQTT 主题。通知有两种：

ListNotification

ListNotification 包含一个列表，其中待处理的任务执行不超过 15 个。它们依次按照状态（IN_PROGRESS 任务执行在 QUEUED 任务执行之前）和排队的时间进行排序。

如果满足以下条件之一，则发布 ListNotification。

- 新的任务执行已排队或更改为非最终状态（IN_PROGRESS 或 QUEUED）。

- 旧的任务执行更改为最终状态（FAILED、SUCCEEDED、CANCELED、TIMED_OUT、REJECTED 或 REMOVED）。

列出通知（**QUEUED** 或 **IN_PROGRESS** 中待处理的任务执行不超过 15 个）	
没有可选的计划配置和定期维护时段 （最多 10 个任务执行）	具有可选的计划配置和定期维护时段 （最多 5 个任务执行）
总是出现在 ListNotification.	仅在维护 ListNotification 时段内出现在中。

NextNotification

- NextNotification 包含有关队列中下一个任务执行的摘要信息。

  每当列表中第一个任务执行更改时，则会发布 NextNotification。

  - 新的任务执行作为 QUEUED 添加到列表，并且在列表中排在第一位。
  - 不是列表中第一位的现有任务执行的状态从 QUEUED 更改为 IN_PROGRESS，然后成为列表中的第一个。（当列表中没有其它 IN_PROGRESS 任务执行时，或者当状态从 QUEUED 更改为 IN_PROGRESS 的任务执行排队的时间早于列表中其他所有 IN_PROGRESS 任务执行时，会出现这种情况。）
  - 列表第一的任务执行状态更改为最终状态，并且从列表中删除。

有关发布和订阅 MQTT 主题的更多信息，请参阅 the section called “设备通信协议”。

> **Note**
>
> 当您使用 HTTP Signature Version 4 或 HTTP TLS 与 Jobs 通信时，通知不可用。

任务待处理

在某事物的待处理任务执行列表中添加或从中删除任务或列表中的第一个任务执行发生变化时， AWS IoT 作业服务会在 MQTT 主题上发布一条消息：

- $aws/things/*thingName*/jobs/notify

- $aws/things/*thingName*/jobs/notify-next

消息包含以下示例负载：

$aws/things/*thingName*/jobs/notify:

```
{
  "timestamp" : 10011,
  "jobs" : {
    "IN_PROGRESS" : [ {
      "jobId" : "other-job",
      "queuedAt" : 10003,
      "lastUpdatedAt" : 10009,
      "executionNumber" : 1,
      "versionNumber" : 1
    } ],
    "QUEUED" : [ {
      "jobId" : "this-job",
      "queuedAt" : 10011,
      "lastUpdatedAt" : 10011,
      "executionNumber" : 1,
      "versionNumber" : 0
    } ]
  }
}
```

如果名为 this-job 的任务执行源自某个选定了可选计划配置且任务文档推出计划在维护时段内进行的任务，则该任务只会在定期维护时段内出现。在维护时段之外，名为 this-job 的任务将从待处理的任务执行列表中排除，如以下示例所示。

```
{
  "timestamp" : 10011,
  "jobs" : {
    "IN_PROGRESS" : [ {
      "jobId" : "other-job",
      "queuedAt" : 10003,
      "lastUpdatedAt" : 10009,
      "executionNumber" : 1,
      "versionNumber" : 1
    } ],
    "QUEUED" : []
  }
```

```
}
```

$aws/things/*thingName*/jobs/notify-next:

```
{
  "timestamp" : 10011,
  "execution" : {
    "jobId" : "other-job",
    "status" : "IN_PROGRESS",
    "queuedAt" : 10009,
    "lastUpdatedAt" : 10009,
    "versionNumber" : 1,
    "executionNumber" : 1,
    "jobDocument" : {"c":"d"}
  }
}
```

如果名为 other-job 的任务执行源自某个选定了可选计划配置且任务文档推出计划在维护时段内进行的任务，则该任务只会在定期维护时段内出现。在维护时段之外，名为 other-job 的任务不会列为下一个任务执行，如以下示例所示。

```
{} //No other pending jobs
```

```
{
  "timestamp" : 10011,
  "execution" : {
      "jobId" : "this-job",
      "queuedAt" : 10011,
      "lastUpdatedAt" : 10011,
      "executionNumber" : 1,
      "versionNumber" : 0,
      "jobDocument" : {"a":"b"}
  }
} // "this-job" is pending next to "other-job"
```

可能的任务执行状态值为
QUEUED、IN_PROGRESS、FAILED、SUCCEEDED、CANCELED、TIMED_OUT、REJECTED 和 REMOVED。

以下一系列示例显示了在创建任务执行以及任务执行从一种状态更改为另一种状态时向每个主题发布的通知。

首先，创建了一个名为 job1 的任务。此通知发布到 jobs/notify 主题：

```
{
  "timestamp": 1517016948,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

此通知发布到 jobs/notify-next 主题：

```
{
  "timestamp": 1517016948,
  "execution": {
    "jobId": "job1",
    "status": "QUEUED",
    "queuedAt": 1517016947,
    "lastUpdatedAt": 1517016947,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

当创建另一个任务 (job2) 时，此通知将发布到 jobs/notify 主题：

```
{
  "timestamp": 1517017192,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
```

```
        "lastUpdatedAt": 1517016947,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

通知未发布到 jobs/notify-next 主题，因为队列中的下一个任务 (job1) 尚未更改。当 job1 开始执行时，其状态更改为 IN_PROGRESS。没有发布任何通知，因为任务列表和队列中的下一个任务尚未更改。

当添加第三个任务 (job3) 时，此通知将发布到 jobs/notify 主题：

```
{
  "timestamp": 1517017906,
  "jobs": {
    "IN_PROGRESS": [
      {
        "jobId": "job1",
        "queuedAt": 1517016947,
        "lastUpdatedAt": 1517017472,
        "startedAt": 1517017472,
        "executionNumber": 1,
        "versionNumber": 2
      }
    ],
    "QUEUED": [
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
```

```
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

通知未发布到 jobs/notify-next 主题，因为队列中的下一个任务仍为 job1。

当 job1 完成后，其状态更改为 SUCCEEDED，此通知将发布到 jobs/notify 主题：

```
{
  "timestamp": 1517186269,
  "jobs": {
    "QUEUED": [
      {
        "jobId": "job2",
        "queuedAt": 1517017191,
        "lastUpdatedAt": 1517017191,
        "executionNumber": 1,
        "versionNumber": 1
      },
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517017905,
        "executionNumber": 1,
        "versionNumber": 1
      }
    ]
  }
}
```

此时，已从队列中删除 job1，要执行的下一个任务是 job2。此通知发布到 jobs/notify-next 主题：

```
{
  "timestamp": 1517186269,
  "execution": {
    "jobId": "job2",
```

```
    "status": "QUEUED",
    "queuedAt": 1517017191,
    "lastUpdatedAt": 1517017191,
    "versionNumber": 1,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

如果 job3 必须在 job2 之前开始执行（不推荐），job3 的状态可以更改为 IN_PROGRESS。更改后，job2 则不再是队列中的下一个任务，此通知将发布到 jobs/notify-next 主题：

```
{
  "timestamp": 1517186779,
  "execution": {
    "jobId": "job3",
    "status": "IN_PROGRESS",
    "queuedAt": 1517017905,
    "startedAt": 1517186779,
    "lastUpdatedAt": 1517186779,
    "versionNumber": 2,
    "executionNumber": 1,
    "jobDocument": {
      "operation": "test"
    }
  }
}
```

没有向 jobs/notify 主题发布任何通知，因为没有添加或删除任何任务。

如果设备拒绝 job2，并且将其状态更新为 REJECTED，则此通知将发布到 jobs/notify 主题：

```
{
  "timestamp": 1517189392,
  "jobs": {
    "IN_PROGRESS": [
      {
        "jobId": "job3",
        "queuedAt": 1517017905,
        "lastUpdatedAt": 1517186779,
        "startedAt": 1517186779,
```

```
        "executionNumber": 1,
        "versionNumber": 2
      }
    ]
  }
}
```

如果 job3（仍在进行）被强制删除，则此通知将发布到 jobs/notify 主题：

```
{
  "timestamp": 1517189551,
  "jobs": {}
}
```

此时队列为空。此通知发布到 jobs/notify-next 主题：

```
{
  "timestamp": 1517189551
}
```

AWS IoT 任务 API 操作

AWS IoT 任务 API 可用于以下任一类别：

- 诸如管理和控制任务等管理任务。这是控制面板。
- 执行这些任务的设备。这是数据面板，允许您发送和接收数据。

任务的管理和控制操作将使用 HTTPS 协议 API。设备可以使用 MQTT 或 HTTPS 协议 API。控制面板 API 旨在用于在创建和跟踪任务时通常进行的少量调用。它通常会为一个请求打开一个连接，然后在收到响应后关闭此连接。数据面板 HTTPS 和 MQTT API 允许进行长时间轮询。这些 API 操作旨在用于可扩展至数百万台设备的大量流量。

每个 AWS IoT Jobs HTTPS API 均有一个对应的命令，您可使用该命令从 AWS Command Line Interface（AWS CLI）调用该 API。命令采用小写形式，并且在构成 API 名称的单词之间使用连字符。例如，您可以通过键入以下内容，在 CLI 上调用 CreateJob API：

```
aws iot create-job ...
```

如果在操作过程中发生错误，您会收到包含有关错误的信息的错误响应。

ErrorResponse

包含有关 AWS IoT Jobs 服务操作期间发生的错误的信息。

以下示例显示此操作的语法：

```
{
    "code": "ErrorCode",
    "message": "string",
    "clientToken": "string",
    "timestamp": timestamp,
    "executionState": JobExecutionState
}
```

以下是此 `ErrorResponse` 的描述：

`code`

可将 ErrorCode 设置为：

InvalidTopic

请求发送至 AWS IoT Jobs 命名空间中未映射到任何 API 操作的主题。

InvalidJson

请求的内容无法解释为有效的 UTF-8 编码的 JSON。

InvalidRequest

请求的内容无效。例如，当 `UpdateJobExecution` 请求包含无效的状态详细信息时，将返回此代码。消息包含有关错误的详细信息。

InvalidStateTransition

由于任务执行的当前状态，更新已尝试将任务执行更改为无效的状态。例如，尝试将状态为 SUCCEEDED 的请求更改为状态 IN_PROGRESS。在这种情况下，错误消息的正文还包含 `executionState` 字段。

ResourceNotFound

请求主题所指定的 `JobExecution` 不存在。

VersionMismatch

请求中指定的预期版本与 AWS IoT Jobs 服务中的任务执行版本不匹配。在这种情况下，错误消息的正文还包含 `executionState` 字段。

InternalError

处理请求期间出现内部错误。

RequestThrottled

请求已被阻止。

TerminalStateReached

在处于最终状态的任务上执行描述任务的命令时发生。

`message`

错误消息字符串。

`clientToken`

用于将请求与其答复关联起来的任意字符串。

`timestamp`

用自纪元以来的秒数表示的时间。

`executionState`

一个 `JobExecutionState` 对象。仅在 `code` 字段具有 `InvalidStateTransition` 或 `VersionMismatch` 值时包含此字段。在这些情况下，这使得不必执行单独的 `DescribeJobExecution` 请求以获取当前任务执行状态数据。

下面列出了任务 API 操作和数据类型。

- 任务管理和控制 API 以及数据类型
- 任务设备 MQTT 和 HTTPS API 操作以及数据类型

任务管理和控制 API 以及数据类型

以下命令可供 CLI 中的 Job 管理和控制和通过 HTTPS 协议使用。

- 任务管理和控制数据类型
- 任务管理和控制 API 操作

要确定 CLI 命令的 *`endpoint-url`* 参数，请运行此命令。

```
aws iot describe-endpoint --endpoint-type=iot:Jobs
```

此命令将返回以下输出。

```
{
"endpointAddress": "account-specific-prefix.jobs.iot.aws-region.amazonaws.com"
}
```

> **Note**
>
> Jobs 终端节点不支持 ALPN `z-amzn-http-ca`。

任务管理和控制数据类型

管理和控制应用程序使用以下数据类型与 AWS IoT Jobs 进行通信。

任务

Job 对象包含有关任务的详细信息。以下示例显示该语法：

```
{
    "jobArn": "string",
    "jobId": "string",
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED",
    "forceCanceled": boolean,
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "comment": "string",
    "targets": ["string"],
    "description": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp,
    "jobProcessDetails": {
        "processingTargets": ["string"],
        "numberOfCanceledThings": long,
        "numberOfSucceededThings": long,
        "numberOfFailedThings": long,
        "numberOfRejectedThings": long,
        "numberOfQueuedThings": long,
```

```
        "numberOfInProgressThings": long,
        "numberOfRemovedThings": long,
        "numberOfTimedOutThings": long
    },
    "presignedUrlConfig": {
        "expiresInSec": number,
        "roleArn": "string"
    },
    "jobExecutionsRolloutConfig": {
        "exponentialRate": {
           "baseRatePerMinute": integer,
           "incrementFactor": integer,
           "rateIncreaseCriteria": {
              "numberOfNotifiedThings": integer, // Set one or the other
              "numberOfSucceededThings": integer // of these two values.
           },
           "maximumPerMinute": integer
      }
    },
    "abortConfig": {
       "criteriaList": [
          {
             "action": "string",
             "failureType": "string",
             "minNumberOfExecutedThings": integer,
             "thresholdPercentage": integer
          }
       ]
    },
    "SchedulingConfig": {
      "startTime": string
      "endTime": string
      "timeZone": string


      "endTimeBehavior": string

   },
    "timeoutConfig": {
        "inProgressTimeoutInMinutes": long
    }
}
```

有关更多信息，请参阅 Job 或 job。

JobSummary

JobSummary 对象包含任务摘要。以下示例显示该语法：

```
{
    "jobArn": "string",
    "jobId": "string",
    "status": "IN_PROGRESS|CANCELED|SUCCEEDED|SCHEDULED",
    "targetSelection": "CONTINUOUS|SNAPSHOT",
    "thingGroupId": "string",
    "createdAt": timestamp,
    "lastUpdatedAt": timestamp,
    "completedAt": timestamp
}
```

有关更多信息，请参阅 JobSummary 或 job-summary。

JobExecution

JobExecution 对象表示设备上的任务执行。以下示例显示该语法：

> **Note**
>
> 当您使用控制面板 API 操作时，JobExecution 数据类型不包含 JobDocument 字段。要获取此信息，您可以使用 GetJobDocument API 操作或 get-job-document CLI 命令。

```
{
    "approximateSecondsBeforeTimedOut": 50,
    "executionNumber": 1234567890,
    "forceCanceled": true|false,
    "jobId": "string",
    "lastUpdatedAt": timestamp,
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "forceCanceled": boolean,
    "statusDetails": {
```

```
        "detailsMap": {
            "string": "string" ...
        },
        "status": "string"
    },
    "thingArn": "string",
    "versionNumber": 123
}
```

有关更多信息，请参阅 JobExecution 或 job-execution。

JobExecutionSummary

JobExecutionSummary 对象包含任务执行摘要信息。以下示例显示该语法：

```
{
    "executionNumber": 1234567890,
    "queuedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "startedAt": timestamp,
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|REMOVED"
}
```

有关更多信息，请参阅 JobExecutionSummary 或 job-execution-summary。

JobExecutionSummaryForJob

JobExecutionSummaryForJob 对象包含有关特定任务的任务执行的信息摘要。以下示例显示该语法：

```
{
    "executionSummaries": [
        {
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyThing",
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            }
        },
        ...
```

```
    ]
}
```

有关更多信息，请参阅 JobExecutionSummaryForJob 或 job-execution-summary-for-job。

JobExecutionSummaryForThing

JobExecutionSummaryForThing 对象包含有关特定事物上的任务执行的信息摘要。以下示例显示该语法：

```
{
    "executionSummaries": [
        {
            "jobExecutionSummary": {
                "status": "IN_PROGRESS",
                "lastUpdatedAt": 1549395301.389,
                "queuedAt": 1541526002.609,
                "executionNumber": 1
            },
            "jobId": "MyThingJob"
        },
        ...
    ]
}
```

有关更多信息，请参阅 JobExecutionSummaryForThing 或 job-execution-summary-for-thing。

任务管理和控制 API 操作

使用以下 API 操作或 CLI 命令：

AssociateTargetsWithJob

将组与持续任务关联。必须满足以下标准：

- 必须已在 targetSelection 字段设置为 CONTINUOUS 的情况下创建任务。
- 任务状态当前必须为 IN_PROGRESS。
- 与任务关联的目标总数不能超过 100。

HTTPS request

```
POST /jobs/jobId/targets

{
"targets": [ "string" ],
"comment": "string"
}
```

有关更多信息，请参阅AssociateTargetsWithJob。

CLI syntax

```
aws iot  associate-targets-with-job \
--targets <value> \
--job-id <value> \
[--comment <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"targets": [
"string"
],
"jobId": "string",
"comment": "string"
}
```

有关更多信息，请参阅associate-targets-with-job。

CancelJob

取消任务。

HTTPS request

```
PUT /jobs/jobId/cancel

{
```

```
"force": boolean,
"comment": "string",
"reasonCode": "string"
}
```

有关更多信息，请参阅CancelJob。

CLI syntax

```
aws iot cancel-job \
    --job-id <value> \
    [--force <value>]  \
    [--comment <value>]  \
    [--reasonCode <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
    "jobId": "string",
    "force": boolean,
    "comment": "string"
}
```

有关更多信息，请参阅cancel-job。

CancelJobExecution

取消设备上的任务执行。

HTTPS request

```
PUT /things/thingName/jobs/jobId/cancel

{
"force": boolean,
"expectedVersion": "string",
"statusDetails": {
    "string": "string"
    ...
```

```
}
}
```

有关更多信息，请参阅CancelJobExecution。

CLI syntax

```
aws iot cancel-job-execution \
--job-id <value> \
--thing-name <value> \
[--force | --no-force] \
[--expected-version <value>] \
[--status-details <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"thingName": "string",
"force": boolean,
"expectedVersion": long,
"statusDetails": {
"string": "string"
}
}
```

有关更多信息，请参阅cancel-job-execution。

CreateJob

创建任务。您可以提供任务文档，作为指向 Amazon S3 桶（documentSource 参数）中或请求正文（document 参数）中的文件的链接。

通过将可选的 targetSelection 参数设置为 CONTINUOUS（默认为 SNAPSHOT），可使任务成为连续任务。连续任务可用于在将设备添加到组时登记或升级设备，因为它将继续运行并在新添加的事物上启动。即使在创建任务时组中的事物已完成任务，也可能发生这种情况。

任务可以具有可选的 TimeoutConfig，这会设置进行中计时器的值。进行中计时器无法更新，应用到该任务的全部执行。

对 CreateJob API 的参数执行以下验证：

- targets 参数必须是有效事物或事物组 ARN 的列表。所有事物和事物组必须在您的AWS 账户中。
- documentSource 参数必须是指向任务文档的有效 Amazon S3 URL。Amazon S3 URL 的格式为：https://s3.amazonaws.com/*bucketName*/*objectName*。
- 存储在由 documentSource 参数指定的 URL 中的文档必须是 UTF-8 编码的 JSON 文档。
- 由于 MQTT 消息大小限制 (128 KB) 和加密的原因，任务文档的大小限制为 32 KB。
- jobId 必须在您的AWS 账户中是唯一的。

HTTPS request

```
PUT /jobs/jobId

{
"targets": [ "string" ],
"document": "string",
"documentSource": "string",
"description": "string",
"jobTemplateArn": "string",
"presignedUrlConfigData": {
    "roleArn": "string",
    "expiresInSec": "integer"
},
"targetSelection": "CONTINUOUS|SNAPSHOT",
"jobExecutionsRolloutConfig": {
    "exponentialRate": {
       "baseRatePerMinute": integer,
       "incrementFactor": integer,
       "rateIncreaseCriteria": {
          "numberOfNotifiedThings": integer, // Set one or the other
          "numberOfSucceededThings": integer // of these two values.
       },
       "maximumPerMinute": integer
  }
},
"abortConfig": {
   "criteriaList": [
      {
         "action": "string",
         "failureType": "string",
         "minNumberOfExecutedThings": integer,
```

```
          "thresholdPercentage": integer
       }
    ]
},
"SchedulingConfig": {
    "startTime": string
    "endTime": string
    "timeZone": string


    "endTimeBehavior": string

   }
"timeoutConfig": {
  "inProgressTimeoutInMinutes": long
}
}
```

有关更多信息，请参阅CreateJob。

CLI syntax

```
aws iot create-job \
    --job-id <value> \
    --targets <value> \
    [--document-source <value>] \
    [--document <value>] \
    [--description <value>] \
    [--job-template-arn <value>] \
    [--presigned-url-config <value>] \
    [--target-selection <value>] \
    [--job-executions-rollout-config <value>] \
    [--abort-config <value>] \
    [--timeout-config <value>] \
    [--document-parameters <value>]  \
    [--cli-input-json <value>] \
    [--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
    "jobId": "string",
    "targets": [ "string" ],
```

```
    "documentSource": "string",
    "document": "string",
    "description": "string",
    "jobTemplateArn": "string",
    "presignedUrlConfig": {
        "roleArn": "string",
        "expiresInSec": long
     },
    "targetSelection": "string",
    "jobExecutionsRolloutConfig": {
          "exponentialRate": {
              "baseRatePerMinute": integer,
              "incrementFactor": integer,
              "rateIncreaseCriteria": {
                 "numberOfNotifiedThings": integer, // Set one or the other
                 "numberOfSucceededThings": integer // of these two values.
              },
      "maximumPerMinute": integer
      }
    },
    "abortConfig": {
    "criteriaList": [
        {
           "action": "string",
           "failureType": "string",
           "minNumberOfExecutedThings": integer,
           "thresholdPercentage": integer
         }
      ]
    },
    "timeoutConfig": {
          "inProgressTimeoutInMinutes": long
    },
    "documentParameters": {
    "string": "string"
    }
}
```

有关更多信息，请参阅[create-job](#)。

DeleteJob

删除任务及其相关的任务执行。

删除任务可能需要一些时间，具体取决于为该任务创建的任务执行的数量以及其它各种因素。正在删除任务时，任务的状态显示为“DELETION_IN_PROGRESS”。尝试删除或取消已处于“DELETION_IN_PROGRESS”状态的任务将导致错误。

HTTPS request

```
DELETE /jobs/jobId?force=force
```

有关更多信息，请参阅DeleteJob。

CLI syntax

```
aws iot  delete-job \
--job-id <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"force": boolean
}
```

有关更多信息，请参阅delete-job。

DeleteJobExecution

删除任务执行。

HTTPS request

```
DELETE /things/thingName/jobs/jobId/executionNumber/executionNumber?force=force
```

有关更多信息，请参阅DeleteJobExecution。

CLI syntax

```
aws iot  delete-job-execution \
```

```
--job-id <value> \
--thing-name <value> \
--execution-number <value> \
[--force | --no-force]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"thingName": "string",
"executionNumber": long,
"force": boolean
}
```

有关更多信息，请参阅delete-job-execution。

DescribeJob

获取任务执行的详细信息。

HTTPS request

```
GET /jobs/jobId
```

有关更多信息，请参阅DescribeJob。

CLI syntax

```
aws iot describe-job \
--job-id <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string"
}
```

有关更多信息，请参阅describe-job。

DescribeJobExecution

获取任务执行的详细信息。任务的执行状态必须为 SUCCEEDED 或 FAILED。

HTTPS request

```
GET /things/thingName/jobs/jobId?executionNumber=executionNumber
```

有关更多信息，请参阅DescribeJobExecution。

CLI syntax

```
aws iot  describe-job-execution \
--job-id <value> \
--thing-name <value> \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"thingName": "string",
"executionNumber": long
}
```

有关更多信息，请参阅describe-job-execution。

GetJobDocument

获取任务的任务文档。

> **Note**
>
> 占位符 URL 不会替换为返回文档中的预签名 Amazon S3 URL。仅在 AWS IoT Jobs 服务通过 MQTT 接收请求时生成预签名 URL。

HTTPS request

```
GET /jobs/jobId/job-document
```

有关更多信息，请参阅GetJobDocument。

CLI syntax

```
aws iot get-job-document \
--job-id <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string"
}
```

有关更多信息，请参阅get-job-document。

ListJobExecutionsForJob

获取任务的任务执行的列表。

HTTPS request

```
GET /jobs/jobId/things?status=status&maxResults=maxResults&nextToken=nextToken
```

有关更多信息，请参阅ListJobExecutionsForJob。

CLI syntax

```
aws iot  list-job-executions-for-job \
--job-id <value> \
[--status <value>] \
[--max-results <value>] \
[--next-token <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

有关更多信息，请参阅list-job-executions-for-job。

ListJobExecutionsForThing

获取事物的任务执行的列表。

HTTPS request

```
GET /things/thingName/jobs?status=status&maxResults=maxResults&nextToken=nextToken
```

有关更多信息，请参阅ListJobExecutionsForThing。

CLI syntax

```
aws iot list-job-executions-for-thing \
--thing-name <value> \
[--status <value>] \
[--max-results <value>] \
[--next-token <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"thingName": "string",
"status": "string",
"maxResults": "integer",
"nextToken": "string"
}
```

有关更多信息，请参阅list-job-executions-for-thing。

ListJobs

获取 AWS 账户中的任务列表。

HTTPS request

```
GET /jobs?
status=status&targetSelection=targetSelection&thingGroupName=thingGroupName&thingGroupId=thi
```

有关更多信息，请参阅ListJobs。

CLI syntax

```
aws iot list-jobs \
[--status <value>] \
[--target-selection <value>] \
[--max-results <value>] \
[--next-token <value>] \
[--thing-group-name <value>] \
[--thing-group-id <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"status": "string",
"targetSelection": "string",
"maxResults": "integer",
"nextToken": "string",
"thingGroupName": "string",
"thingGroupId": "string"
}
```

有关更多信息，请参阅list-jobs。

UpdateJob

更新指定任务的受支持字段。timeoutConfig 的更新值仅对新的正在进行的启动生效。目前，正在进行的启动将继续使用之前的超时配置启动。

HTTPS request

```
PATCH /jobs/jobId
{
"description": "string",
"presignedUrlConfig": {
  "expiresInSec": number,
  "roleArn": "string"
},
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
     "baseRatePerMinute": number,
     "incrementFactor": number,
     "rateIncreaseCriteria": {
        "numberOfNotifiedThings": number,
        "numberOfSucceededThings": number
     },
  "maximumPerMinute": number
  },
"abortConfig": {
  "criteriaList": [
     {
        "action": "string",
        "failureType": "string",
        "minNumberOfExecutedThings": number,
        "thresholdPercentage": number
     }
  ]
},
"timeoutConfig": {
  "inProgressTimeoutInMinutes": number
}
}
```

有关更多信息，请参阅UpdateJob。

CLI syntax

```
aws iot  update-job \
--job-id <value> \
[--description <value>] \
[--presigned-url-config <value>] \
[--job-executions-rollout-config <value>] \
[--abort-config <value>] \
```

```
[--timeout-config <value>] \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"description": "string",
"presignedUrlConfig": {
  "expiresInSec": number,
  "roleArn": "string"
},
"jobExecutionsRolloutConfig": {
  "exponentialRate": {
     "baseRatePerMinute": number,
     "incrementFactor": number,
     "rateIncreaseCriteria": {
        "numberOfNotifiedThings": number,
        "numberOfSucceededThings": number
     }
  },
  "maximumPerMinute": number
},
"abortConfig": {
  "criteriaList": [
     {
        "action": "string",
        "failureType": "string",
        "minNumberOfExecutedThings": number,
        "thresholdPercentage": number
     }
  ]
},
"timeoutConfig": {
  "inProgressTimeoutInMinutes": number
}
}
```

有关更多信息，请参阅update-job。

任务设备 MQTT 和 HTTPS API 操作以及数据类型

可通过 MQTT 和 HTTPS 协议使用以下命令。在数据层面上对执行任务的设备使用这些 API 操作。

任务设备 MQTT 和 HTTPS 数据类型

可使用以下数据类型通过 MQTT 和 HTTPS 协议与 AWS IoT Jobs 服务进行通信。

JobExecution

`JobExecution` 对象表示设备上的任务执行。以下示例显示该语法：

> **Note**
>
> 当您使用 MQTT 和 HTTP 数据面板 API 操作时，`JobExecution` 数据类型包含 `JobDocument` 字段。您的设备可以使用此信息从任务执行中检索任务文档。

```
{
    "jobId" : "string",
    "thingName" : "string",
    "jobDocument" : "string",
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
        "string": "string"
    },
    "queuedAt" : "timestamp",
    "startedAt" : "timestamp",
    "lastUpdatedAt" : "timestamp",
    "versionNumber" : "number",
    "executionNumber": long
}
```

有关更多信息，请参阅 `JobExecution` 或 `job-execution`。

JobExecutionState

`JobExecutionState` 包含有关任务执行的状态的信息。以下示例显示该语法：

```
{
```

```
    "status": "QUEUED|IN_PROGRESS|FAILED|SUCCEEDED|CANCELED|TIMED_OUT|REJECTED|
REMOVED",
    "statusDetails": {
        "string": "string"
        ...
    }
    "versionNumber": "number"
}
```

有关更多信息，请参阅 JobExecutionState 或 job-execution-state。

JobExecutionSummary

包含有关任务执行的信息的子集。以下示例显示该语法：

```
{
    "jobId": "string",
    "queuedAt": timestamp,
    "startedAt": timestamp,
    "lastUpdatedAt": timestamp,
    "versionNumber": "number",
    "executionNumber": long
}
```

有关更多信息，请参阅 JobExecutionSummary 或 job-execution-summary。

请参阅以下部分了解有关 MQTT 和 HTTPS API 操作的更多信息：

- 任务设备 MQTT API 操作
- 任务设备 HTTP API

任务设备 MQTT API 操作

可以通过将 MQTT 消息发布到用于 Jobs 命令的保留主题来发出任务设备命令。

您的设备端客户端必须订阅这些命令的响应消息主题。如果您使用 AWS IoT 设备客户端，您的设备将自动订阅响应主题。这意味着消息代理将向发布命令消息的客户端发布响应消息主题，无论您的客户端是否订阅了该响应消息主题。这些响应消息不会通过消息代理，也无法由其它客户端或规则订阅。

订阅机群监控解决方案的任务和 jobExecution 事件主题时，首先启用任务和任务执行事件以接收云端的任何事件。Job 进度消息，这些消息通过消息代理处理并且可以由 AWS IoT 规则使用，将发布为

任务事件。由于消息代理会发布响应消息，即使没有明确订阅响应消息也是如此，因此必须配置您的客户端以接收和标识其接收的消息。您的客户端还必须确认传入消息中的 *thingName* 在客户端对消息执行操作前应用于客户端的事物名称。

> **Note**
>
> AWS IoT 发送的用于响应 MQTT Jobs API 命令消息的消息会计入您的账户费用，无论您是否显式订阅了这些消息。

下面显示了 MQTT API 操作及其请求和响应语法。所有 MQTT API 操作都具有以下参数：

clientToken

用于将请求和响应关联起来的可选客户端令牌。在此处输入任意值，它将反映在响应中。

timestamp

发送消息的时间（用从纪元开始的秒数表示）。

GetPendingJobExecutions

针对指定的事物，获取未处于最终状态的所有任务的列表。

要调用此 API，请在 $aws/things/*thingName*/jobs/get 上发布消息。

请求负载：

```
{ "clientToken": "string" }
```

消息代理将发布 $aws/things/*thingName*/jobs/get/accepted 和 $aws/things/*thingName*/jobs/get/rejected，即使您没有指定订阅它们。但是，为了让您的客户端接收消息，客户端必须侦听这些消息。有关更多信息，请参阅关于 Job API 消息的说明。

响应负载：

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ],
"timestamp" : 1489096425069,
"clientToken" : "client-001"
```

```
}
```

其中，inProgressJobs 和 queuedJobs 返回状态为 IN_PROGRESS 或 QUEUED 的 JobExecutionSummary 对象的列表。

StartNextPendingJobExecution

获取并启动事物的下一个待处理任务执行（状态为 IN_PROGRESS 或 QUEUED）。

- 首先返回状态为 IN_PROGRESS 的所有任务执行。
- 按任务执行的排队顺序返回这些任务执行。在任务的目标组中添加或移除某个事物时，请确认任何新的任务执行与现有任务执行相比的推出顺序。
- 如果下一个待处理任务执行的状态为 QUEUED，则其状态将更改为 IN_PROGRESS，并且任务执行的状态详细信息将设定为指定内容。
- 如果下一个待处理任务执行已处于 IN_PROGRESS 状态，则不会更改其状态详细信息。
- 如果没有待处理的任务执行，则响应不包括 execution 字段。
- （可选）您可以通过为 stepTimeoutInMinutes 属性设置值来创建步骤计时器。如果您没有通过运行 UpdateJobExecution 更新此属性的值，任务执行将在步骤计时器到期时超时。

要调用此 API，请在 $aws/things/*thingName*/jobs/start-next 上发布消息。

请求负载：

```
{
"statusDetails": {
    "string": "job-execution-state"
    ...
},
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

statusDetails

描述任务执行状态的名称-值对的集合。如果未指定，则 statusDetails 保持不变。

stepTimeOutInMinutes

指定此设备完成该任务执行所具有的时间。如果任务执行状态未在此计时器过期或者重置计时器（通过调用 UpdateJobExecution，将状态设置为 IN_PROGRESS，并在字段

stepTimeoutInMinutes 中指定新的超时值）之前设置为最终状态，则任务执行状态将设置为 TIMED_OUT。设置此超时不会影响在创建任务时（CreateJob，使用 timeoutConfig 字段）可能指定的任务执行超时。

消息代理将发布 $aws/things/*thingName*/jobs/start-next/accepted 和 $aws/things/*thingName*/jobs/start-next/rejected，即使您没有指定订阅它们。但是，为了让您的客户端接收消息，客户端必须侦听这些消息。有关更多信息，请参阅关于 Job API 消息的说明。

响应负载：

```
{
"execution" : JobExecutionData,
"timestamp" : timestamp,
"clientToken" : "string"
}
```

其中，execution 是 JobExecution 对象。例如：

```
{
"execution" : {
    "jobId" : "022",
    "thingName" : "MyThing",
    "jobDocument" : "< contents of job document >",
    "status" : "IN_PROGRESS",
    "queuedAt" : 1489096123309,
    "lastUpdatedAt" : 1489096123309,
    "versionNumber" : 1,
    "executionNumber" : 1234567890
},
"clientToken" : "client-1",
"timestamp" : 1489088524284,
}
```

DescribeJobExecution

获取有关任务执行的详细信息。

您可以将 jobId 设置为 $next 以返回事物的下一个待处理任务执行（状态为 IN_PROGRESS 或 QUEUED）。

要调用此 API，请在 $aws/things/*thingName*/jobs/*jobId*/get 上发布消息。

请求负载：

```
{
"jobId" : "022",
"thingName" : "MyThing",
"executionNumber": long,
"includeJobDocument": boolean,
"clientToken": "string"
}
```

`thingName`

与设备关联的事物的名称。

`jobId`

创建此任务时向其分配的唯一标识符。

或使用 `$next` 返回事物的下一个待处理任务执行（状态为 `IN_PROGRESS` 或 `QUEUED`）。在此情况下，首先返回状态为 `IN_PROGRESS` 的所有任务执行。按任务执行的创建顺序返回这些执行。

`executionNumber`

（可选）标识设备上的任务执行的数字。如果未指定，则返回最新的任务执行。

`includeJobDocument`

（可选）除非设置为 `false`，否则响应将包含任务文档。默认为 `true`。

消息代理将发布 `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/get/accepted` 和 `$aws/things/`*`thingName`*`/jobs/`*`jobId`*`/get/rejected`，即使您没有指定订阅它们。但是，为了让您的客户端接收消息，客户端必须侦听这些消息。有关更多信息，请参阅关于 Job API 消息的说明。

响应负载：

```
{
"execution" : JobExecutionData,
"timestamp": "timestamp",
"clientToken": "string"
}
```

其中，`execution` 是 JobExecution 对象。

UpdateJobExecution

更新任务执行的状态。（可选）您可以通过为 stepTimeoutInMinutes 属性设置值创建步骤计时器。如果您没有通过再次运行 UpdateJobExecution 更新此属性的值，任务执行将在步骤计时器到期时超时。

要调用此 API，请在 $aws/things/*thingName*/jobs/*jobId*/update 上发布消息。

请求负载：

```
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"executionNumber": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"clientToken": "string"
}
```

`status`

任务执行的新状态（IN_PROGRESS、FAILED、SUCCEEDED，或 REJECTED）。这必须在每次更新时指定。

`statusDetails`

描述任务执行状态的名称-值对的集合。如果未指定，则 statusDetails 保持不变。

`expectedVersion`

任务执行的预期当前版本。每次更新任务执行时，其版本将递增。如果存储在 AWS IoT Jobs 服务中的任务执行的版本不匹配，则更新将因 VersionMismatch 错误而被拒绝。还会返回包含当前任务执行状态数据的ErrorResponse。（这样就不必执行单独的 DescribeJobExecution 请求以获取任务执行状态数据。）

`executionNumber`

（可选）标识设备上的任务执行的数字。如果未指定，则使用最新的任务执行。

includeJobExecutionState

（可选）在包含此参数且设置为 true 时，响应将包含 JobExecutionState 字段。默认为 false。

includeJobDocument

（可选）在包含此参数且设置为 true 时，响应将包含 JobDocument。默认为 false。

stepTimeoutInMinutes

指定此设备完成该任务执行所具有的时间。如果在该计时器到期之前或在重置计时器之前，任务执行状态未设置为最终状态，则任务执行状态设置为 TIMED_OUT。设置或重置此超时不会影响在创建任务时可能已指定的任务执行超时。

消息代理将发布 $aws/things/*thingName*/jobs/*jobId*/update/accepted 和 $aws/things/*thingName*/jobs/*jobId*/update/rejected，即使您没有指定订阅它们。但是，为了让您的客户端接收消息，客户端必须侦听这些消息。有关更多信息，请参阅关于 Job API 消息的说明。

响应负载：

```
{
"executionState": JobExecutionState,
"jobDocument": "string",
"timestamp": timestamp,
"clientToken": "string"
}
```

executionState

一个 JobExecutionState 对象。

jobDocument

一个任务文档对象。

timestamp

发送消息的时间（用从纪元 开始的秒数表示）。

clientToken

用于将请求和响应关联起来的客户端令牌。

使用 MQTT 协议时，您还可以执行以下更新：

JobExecutionsChanged

在事物的待处理任务执行列表中添加或删除任务执行时发送。

使用 主题：

$aws/things/*thingName*/jobs/notify

消息负载：

```
{
"jobs" : {
    "JobExecutionState": [ JobExecutionSummary ... ]
         },
    "timestamp": timestamp
}
```

NextJobExecutionChanged

当存在对事物的待处理任务执行列表中的下一个任务执行的更改（使用 jobId $next 为 DescribeJobExecution 定义）时发送。当下一个任务的执行详细信息发生更改时不会发送此消息，而仅在将由 DescribeJobExecution（jobId 为 $next）返回的下一个任务发生更改时发送。考虑状态为 QUEUED 的任务执行 J1 和 J2。J1 是待处理任务执行列表中的下一个待处理任务执行。如果 J2 的状态更改为 IN_PROGRESS，而 J1 的状态保持不变，则将发送此通知，并且此通知包含 J2 的详细信息。

使用 主题：

$aws/things/*thingName*/jobs/notify-next

消息负载：

```
{
"execution" : JobExecution,
"timestamp": timestamp,
}
```

任务设备 HTTP API

设备可以使用端口443 上的 HTTP 签名版本 4 与 AWS IoT Jobs 通信。这是由AWS SDK 和 CLI 使用的方法。有关这些工具的更多信息，请参阅 AWS CLI 命令引用：iot-jobs-data 或 AWS SDK 和工具。

以下命令可用于执行任务的设备。有关将 API 操作与 MQTT 协议结合使用的信息，请参阅任务设备 MQTT API 操作。

GetPendingJobExecutions

针对指定的事物，获取未处于最终状态的所有任务的列表。

HTTPS request

```
GET /things/thingName/jobs
```

响应:

```
{
"inProgressJobs" : [ JobExecutionSummary ... ],
"queuedJobs" : [ JobExecutionSummary ... ]
}
```

有关更多信息，请参阅GetPendingJobExecutions。

CLI syntax

```
aws iot-jobs-data get-pending-job-executions \
--thing-name <value>  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"thingName": "string"
}
```

有关更多信息，请参阅get-pending-job-executions。

StartNextPendingJobExecution

获取并启动事物的下一个待处理任务执行（状态为 IN_PROGRESS 或 QUEUED）。

- 首先返回状态为 IN_PROGRESS 的所有任务执行。
- 按任务执行的创建顺序返回这些执行。
- 如果下一个待处理任务执行的状态为 QUEUED，则其状态将更改为 IN_PROGRESS，并且任务执行的状态详细信息将设定为指定内容。
- 如果下一个待处理任务执行已处于 IN_PROGRESS 状态，其状态详细信息不会发生变化。
- 如果没有待处理的任务执行，则响应不包括 execution 字段。
- （可选）您可以通过为 stepTimeoutInMinutes 属性设置值来创建步骤计时器。如果您没有通过运行 UpdateJobExecution 更新此属性的值，任务执行将在步骤计时器到期时超时。

HTTPS request

以下示例显示请求语法：

```
PUT /things/thingName/jobs/$next
{
"statusDetails": {
    "string": "string"
    ...
},
"stepTimeoutInMinutes": long
}
```

有关更多信息，请参阅StartNextPendingJobExecution。

CLI syntax

摘要：

```
aws iot-jobs-data  start-next-pending-job-execution \
--thing-name <value> \
{--step-timeout-in-minutes <value>] \
[--status-details <value>]  \
[--cli-input-json <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"thingName": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": long
}
```

有关更多信息，请参阅start-next-pending-job-execution。

DescribeJobExecution

获取有关任务执行的详细信息。

您可以将 jobId 设置为 $next 以返回事物的下一个待处理任务执行。任务的执行状态必须为 QUEUED 或 IN_PROGRESS。

HTTPS request

请求:

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument
```

响应:

```
{
"execution" : JobExecution,
}
```

有关更多信息，请参阅DescribeJobExecution。

CLI syntax

摘要：

```
aws iot-jobs-data  describe-job-execution \
--job-id <value> \
--thing-name <value> \
[--include-job-document | --no-include-job-document] \
[--execution-number <value>]  \
[--cli-input-json <value>] \
```

```
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"thingName": "string",
"includeJobDocument": boolean,
"executionNumber": long
}
```

有关更多信息，请参阅describe-job-execution。

UpdateJobExecution

更新任务执行的状态。（可选）您可以通过为 stepTimeoutInMinutes 属性设置值来创建步骤计时器。如果您没有通过再次运行 UpdateJobExecution 更新此属性的值，任务执行将在步骤计时器到期时超时。

HTTPS request

请求:

```
POST /things/thingName/jobs/jobId
{
"status": "job-execution-state",
"statusDetails": {
    "string": "string"
    ...
},
"expectedVersion": "number",
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"stepTimeoutInMinutes": long,
"executionNumber": long
}
```

有关更多信息，请参阅UpdateJobExecution。

CLI syntax

摘要：

```
aws iot-jobs-data  update-job-execution \
--job-id <value> \
--thing-name <value> \
--status <value> \
[--status-details <value>] \
[--expected-version <value>] \
[--include-job-execution-state | --no-include-job-execution-state] \
[--include-job-document | --no-include-job-document] \
[--execution-number <value>]  \
[--cli-input-json <value>] \
[--step-timeout-in-minutes <value>] \
[--generate-cli-skeleton]
```

cli-input-json 格式：

```
{
"jobId": "string",
"thingName": "string",
"status": "string",
"statusDetails": {
"string": "string"
},
"stepTimeoutInMinutes": number,
"expectedVersion": long,
"includeJobExecutionState": boolean,
"includeJobDocument": boolean,
"executionNumber": long
}
```

有关更多信息，请参阅update-job-execution。

使用 AWS IoT Jobs 保护用户和设备

要授权用户在其设备上使用 AWS IoT Jobs，您必须使用 IAM 策略向他们授予权限。然后，必须使用 AWS IoT Core 策略对设备进行授权 AWS IoT，才能安全连接、接收任务执行并更新执行状态。

AWS IoT 任务必需的策略类型

下表显示了进行授权所必须使用的不同类型的策略。有关要使用的所需策略的更多信息，请参阅 授权。

所需策略类型

应用场景	协议	身份验证	控制面板/数据层面	身份类型	所需策略类型
授权管理员、运营商或云服务安全地使用 Jobs	HTTPS	AWS 签名版本 4 身份验证（端口 443）	控制面板和数据层面	Amazon Cognito 身份、IAM 或联合身份用户	IAM policy
授权您的 IoT 设备安全地使用 Jobs	MQTT/HTTPS	TCP 或 TLS 双向身份验证（端口 8883 或 443 ）	数据层面	X.509 证书	AWS IoT Core 政策

要授权可在控制平面和数据平面上执行的 AWS IoT 任务操作，您必须使用 IAM 策略。身份必须已向 AWS IoT 进行身份验证才能执行这些操作，必须为 Amazon Cognito 身份或 IAM 用户、组和角色。有关身份验证的更多信息，请参阅 身份验证。

现在，必须使用 AWS IoT Core 策略在数据平面上对设备进行授权，才能安全地连接到设备网关。设备网关使设备能够安全地与之通信 AWS IoT、接收任务执行并更新任务执行状态。通过使用安全的 MQTT 或 HTTPS 通信协议保护设备通信。这些协议使用X.509 客户端证书提供的协议 AWS IoT 对设备连接进行身份验证。

以下内容显示了您如何授权您的用户、云服务和设备使用 AWS IoT Jobs。有关控制面板和数据层面 API 操作的更多信息，请参阅AWS IoT 任务 API 操作。

主题

- 向用户和云服务授权以使用 AWS IoT Jobs
- 授权您的设备在数据平面上安全地使用 AWS IoT 作业

向用户和云服务授权以使用 AWS IoT Jobs

要向用户和云服务授权，您必须同时在控制面板和数据层面上使用 IAM 策略。这些策略必须与 HTTPS 协议一起使用，并且必须使用 AWS 签名版本 4 身份验证（端口 443）对用户进行身份验证。

> **Note**
>
> AWS IoT Core 不得在控制平面上使用策略。只有 IAM 策略用于向用户或云服务授权。有关使用所需策略类型的更多信息，请参阅 AWS IoT 任务必需的策略类型。

IAM 策略是包含策略声明的 JSON 文档。策略语句使用效果、操作 和资源 元素以指定资源、允许或拒绝的操作以及允许或拒绝操作的条件。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素参考。

> **Warning**
>
> 我们建议您不要在您的 IAM 策略或 AWS IoT Core 策略`"Action": ["iot:*"]`中使用通配符权限。使用通配符权限不是推荐的安全最佳实践。有关更多信息，请参阅AWS IoT 政策过于宽松。

控制面板上的 IAM 策略

在控制面板上，IAM 策略对操作使用 `iot:` 前缀，以便授权执行相应的任务 API 操作。例如，`iot:CreateJob` 策略操作授予用户使用 `CreateJob` API 的权限。

策略操作

下表显示了 IAM 策略操作和使用 API 操作的权限的列表。有关资源类型的信息，请参阅由定义的资源类型 AWS IoT。有关 AWS IoT 操作的更多信息，请参阅由定义的操作 AWS IoT。

控制面板上的 IAM 策略操作

策略操作	API 操作	资源类型	描述
`iot:AssociateTargetsWithJob`	`AssociateTargetsWithJob`	• 作业 • thing • thinggroup	表示将组与连续任务关联的权限。每次请求关联目标时，都会检查 `iot:AssociateTargetsWithJob` 权限。
`iot:CancelJob`	`CancelJob`	作业	表示取消任务的权限。每次请求取消任务时，都会检查 `iot:CancelJob` 权限。

策略操作	API 操作	资源类型	描述
iot:CancelJobExecution	CancelJobExecution	• 作业 • thing	表示取消任务执行的权限。每次请求取消任务执行时，都会检查 iot: CancelJobExecution 权限。
iot:CreateJob	CreateJob	• 作业 • thing • thinggroup • jobtemplate • 程序包	表示创建任务的权限。每次请求创建任务时，都会检查 iot: CreateJob 权限。
iot:CreateJobTemplate	CreateJobTemplate	• 作业 • jobtemplate • 程序包	表示创建任务模板的权限。每次请求创建任务模板时，都会检查 iot: CreateJobTemplate 权限。
iot:DeleteJob	DeleteJob	作业	表示删除任务的权限。每次请求删除任务时，都会检查 iot: DeleteJob 权限。
iot:DeleteJobTemplate	DeleteJobTemplate	jobtemplate	表示删除任务模板的权限。每次请求删除任务模板时，都会检查 iot: CreateJobTemplate 权限。
iot:DeleteJobExecution	DeleteJobTemplate	• 作业 • thing	表示删除任务执行的权限。每次请求删除任务执行时，都会检查 iot: DeleteJobExecution 权限。
iot:DescribeJob	DescribeJob	作业	表示描述任务的权限。每次请求描述任务时，都会检查 iot: DescribeJob 权限。

策略操作	API 操作	资源类型	描述
iot:DescribeJobExecution	DescribeJobExecution	• 作业 • thing	表示描述任务执行的权限。每次请求描述任务执行时，都会检查 iot: DescribeJobExecution 权限。
iot:DescribeJobTemplate	DescribeJobTemplate	jobtemplate	表示描述任务模板的权限。每次请求描述任务模板时，都会检查 iot: DescribeJobTemplate 权限。
iot:DescribeManagedJobTemplate	DescribeManagedJobTemplate	jobtemplate	表示描述托管式任务模板的权限。每次请求描述托管式任务模板时，都会检查 iot: DescribeManagedJobTemplate 权限。
iot:GetJobDocument	GetJobDocument	作业	表示获取任务的任务文档的权限。每次请求获取任务文档时，都会检查 iot:GetJobDocument 权限。
iot:ListJobExecutionsForJob	ListJobExecutionsForJob	作业	表示列出任务的任务执行的权限。每次请求列出任务的任务执行时，都会检查 iot:ListJobExecutionsForJob 权限。
iot:ListJobExecutionsForThing	ListJobExecutionsForThing	thing	表示列出任务的任务执行的权限。每次请求列出事物的任务执行时，都会检查 iot:ListJobExecutionsForThing 权限。
iot:ListJobs	ListJobs	none	表示列出任务的权限。每次请求列出任务时，都会检查 iot:ListJobs 权限。
iot:ListJobTemplates	ListJobTemplates	none	表示列出任务模板的权限。每次请求列出任务模板时，都会检查 iot:ListJobTemplates 权限。

策略操作	API 操作	资源类型	描述
iot:ListManagedJobTemplates	ListManagedJobTemplates	none	表示列出托管式任务模板的权限。每次请求列出托管式任务模板时，都会检查 iot:ListManagedJobTemplates 权限。
iot:UpdateJob	UpdateJob	作业	表示更新任务的权限。每次请求更新任务时，都会检查 iot:UpdateJob 权限。
iot:TagResource	TagResource	• 作业 • jobtemplate • thing	授予标记特定资源的权限。
iot:UntagResource	UntagResource	• 作业 • jobtemplate • thing	授予取消标记特定资源的权限。

基本 IAM 策略示例

下面的示例显示了 IAM policy，该策略授予用户对 IoT 事物和事物组执行以下操作的权限。

在此示例中：

- 与您所在@@ *###* AWS 区域，例如us-east-1。
- 带有您的 AWS 账户 号码的@@ *## ID*，例如。57EXAMPLE833
- *thing-group-name*上面写上你要定位工作的物联网事物组的名称，例如FirmwareUpdateGroup。
- 将 *thing-name* 替换为目标任务所属 IoT 事物的名称，例如 MyIoTThing。

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

```
        {
            "Action": [
                "iot:CreateJobTemplate",
                "iot:CreateJob",
                ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:region:account-id:thinggroup/thing-group-name"
        },
        {
            "Action": [
                "iot:DescribeJob",
                "iot:CancelJob",
                "iot:DeleteJob",
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:region:account-id:job/*"
        },
        {
            "Action": [
                "iot:DescribeJobExecution",
                "iot:CancelJobExecution",
                "iot:DeleteJobExecution",
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:region:account-id:thing/thing-name"
                "arn:aws:iot:region:account-id:job/*"
            ]
        }
    ]
}
```

适用于基于 IP 的授权的 IAM policy 示例

您可以限制主体 从特定 IP 地址对控制面板端点进行 API 调用。要指定可以允许的 IP 地址，请在 IAM policy 的条件元素中使用 aws:SourceIp 全局条件键。

使用此条件密钥还可以拒绝其他 AWS 服务人代表您进行这些 API 调用，例如 AWS CloudFormation。要允许访问这些服务，请使用带有 aws: 密钥的aws:ViaAWSService全局条件SourceIp 密钥。这可以确保源 IP 地址访问限制仅适用于由主体直接发出的请求。有关更多信息，请参阅 AWS： AWS 根据源 IP 拒绝访问。

以下示例说明如何仅允许能够对控制面板端点进行 API 调用的特定 IP 地址。aws:ViaAWSService 键设置为 true，这允许其它服务代表您进行 API 调用。

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJobTemplate",
                "iot:CreateJob"
             ],
            "Resource": ["*"],
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": "123.45.167.89"
                }
            },
            "Bool": {"aws:ViaAWSService": "true"}
        }
    ],
}
```

数据层面上的 IAM 策略

数据层面上的 IAM 策略使用 iotjobsdata: 前缀以授权用户可以执行的任务 API 操作。在数据层面上，可以通过使用 iotjobsdata:DescribeJobExecution 策略操作授予用户使用 DescribeJobExecution API 的权限。

> **Warning**
>
> 当设备的目标指向 AWS IoT Jobs 时，不建议在数据层面上使用 IAM 策略。我们建议您在控制面板上使用 IAM 策略，供用户创建和管理任务。在数据层面上，要授权设备检索任务执行和更新执行状态，请使用 AWS IoT Core HTTPS 协议的策略。

基本 IAM 策略示例

必须授权的 API 操作通常由您键入 CLI 命令来执行。下面显示了用户执行 DescribeJobExecution 操作的示例。

在此示例中：

- 与您所在@@ *###* AWS 区域，例如us-east-1。
- 带有您的 AWS 账户 号码的@@ *## ID*，例如。57EXAMPLE833
- 将 *thing-name* 替换为目标任务所属 IoT 事物的名称，例如 myRegisteredThing。
- *job-id* 是使用 API 确定的目标任务的唯一标识符。

```
aws iot-jobs-data describe-job-execution \
    --endpoint-url "https://account-id.jobs.iot.region.amazonaws.com" \
    --job-id jobID --thing-name thing-name
```

下面显示了授权执行此操作的示例 IAM 策略：

```
{
    "Version": "2012-10-17",
    "Statement":
    {
        "Action": ["iotjobsdata:DescribeJobExecution"],
        "Effect": "Allow",
        "Resource": "arn:aws:iot:region:account-id:thing/thing-name",
    }
   }
```

适用于基于 IP 的授权的 IAM policy 示例

您可以限制主体 从特定 IP 地址对数据面板端点进行 API 调用。要指定可以允许的 IP 地址，请在 IAM policy 的条件元素中使用 aws:SourceIp 全局条件键。

使用此条件密钥还可以拒绝其他 AWS 服务人代表您进行这些 API 调用，例如 AWS CloudFormation。要允许访问这些服务，请将 aws:ViaAWSService 全局条件键与 aws:SourceIp 条件键一起使用。这可确保 IP 地址访问限制仅适用于主体直接发出的请求。有关更多信息，请参阅 AWS： AWS 根据源 IP 拒绝访问。

以下示例说明如何仅允许能够对数据面板端点进行 API 调用的特定 IP 地址。

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

```
            "Action": ["iotjobsdata:*"],
            "Resource": ["*"],
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": "123.45.167.89"
                }
            },
            "Bool": {"aws:ViaAWSService": "false"}
        }
    ],
}
```

以下示例说明如何限制特定 IP 地址或地址范围，使其无法对数据面板端点进行 API 调用。

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
            "Effect": "Deny",
            "Action": ["iotjobsdata:*"],
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "123.45.167.89",
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            },
            "Resource": ["*"],
        }
    ],
}
```

控制面板和数据层面的 IAM 策略示例

如果您同时在控制面板和数据面板上执行 API 操作，则控制面板策略操作必须使用 `iot:` 前缀，而数据面板策略操作必须使用 `iotjobsdata:` 前缀。

例如，`DescribeJobExecution` API 可以同时用于控制面板和数据层面中。在控制平面上，DescribeJobExecutionAPI 用于描述任务执行。在数据层面上，DescribeJobExecutionAPI 用于获取任务执行的详细信息。

以下 IAM 策略授予用户同时在控制面板和数据层面上使用 DescribeJobExecution API 的权限。

在此示例中：

- 与您所在@@ *###* AWS 区域，例如us-east-1。
- 带有您的 AWS 账户 号码的@@ *## ID*，例如。57EXAMPLE833
- 将 *thing-name* 替换为目标任务所属 IoT 事物的名称，例如 MyIoTThing。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": ["iotjobsdata:DescribeJobExecution"],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:region:account-id:thing/thing-name"
        },
        {
            "Action": [
                "iot:DescribeJobExecution",
                "iot:CancelJobExecution",
                "iot:DeleteJobExecution",
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:region:account-id:thing/thing-name"
                "arn:aws:iot:region:account-id:job/*"
            ]
        }
    ]
}
```

授权对 IoT 资源进行标记

为了更好地控制您可以创建、修改或使用的任务和任务模板，可以将标签附加到任务或任务模板。还可以通过将标签放在账单组中并向其附加标签，帮助您辨别所有权并分配和分摊成本。

当用户想要标记他们使用或创建的任务或任务模板时 AWS CLI，您的 IAM 策略必须向该用户授予对其进行标记的权限。 AWS Management Console 要授予权限，IAM 策略必须使用 iot:TagResource 操作。

> **Note**
>
> 如果您的 IAM 策略不包含 `iot:TagResource` 操作，则任何带有标签的 `CreateJob` 或 `CreateJobTemplate` 都将返回 `AccessDeniedException` 错误。

当您想要标记使用或创建的任务或任务模板时，您的 IAM 策略必须授予对它们进行标记的权限。 AWS Management Console AWS CLI要授予权限，IAM 策略必须使用 `iot:TagResource` 操作。

有关标记实例的一般信息，请参阅为资源添加 AWS IoT 标签。

IAM 策略示例

请参阅以下授予标记权限的 IAM 策略示例：

示例 1

某位用户运行以下命令来创建任务并将其标记到特定环境。

在此示例中：

- 与您所在@@ *###* AWS 区域，例如`us-east-1`。
- 带有您的 AWS 账户 号码的@@ *## ID*，例如。`57EXAMPLE833`
- 将 *thing-name* 替换为目标任务所属 IoT 事物的名称，例如 `MyIoTThing`。

```
aws iot create-job
    --job-id test_job
    --targets "arn:aws:iot:region:account-id:thing/thingOne"
    --document-source "https://s3.amazonaws.com/my-s3-bucket/job-document.json"
    --description "test job description"
    --tags Key=environment,Value=beta
```

对于本示例，您必须使用以下 IAM 策略：

```
{
    "Version": "2012-10-17",
    "Statement":
    {
        "Action": [ "iot:CreateJob", "iot:CreateJobTemplate", "iot:TagResource" ],
```

```
        "Effect": "Allow",
        "Resource": [
            "arn:aws:iot:aws-region:account-id:job/*",
            "arn:aws:iot:aws-region:account-id:jobtemplate/*"
        ]
    }
}
```

授权您的设备在数据平面上安全地使用 AWS IoT 作业

要授权您的设备在数据层面上安全地与 AWS IoT 任务交互，您必须使用 AWS IoT Core 策略。 AWS IoT Core 作业策略是包含政策声明的 JSON 文档。这些策略还使用效果、操作 和资源 元素，并遵循与 IAM 策略类似的约定。有关这些元素的更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素参考。

这些策略可以与 MQTT 和 HTTPS 协议一起使用，并且必须使用 TCP 或 TLS 双向身份验证来对设备进行身份验证。下面展示了如何在不同的通信协议中使用这些策略。

> **Warning**
>
> 我们建议您不要在您的 IAM 策略或 AWS IoT Core 策略`"Action": ["iot:*"]`中使用通配符权限。使用通配符权限不是推荐的安全最佳实践。有关更多信息，请参阅AWS IoT 政策过于宽松。

AWS IoT Core MQTT 协议的策略

AWS IoT Core MQTT 协议策略授予您使用作业设备 MQTT API 操作的权限。MQTT API 操作用于处理为任务命令保留的 MQTT 主题。有关这些 API 操作的更多信息，请参阅 任务设备 MQTT API 操作。

MQTT 策略使用策略操作（如 `iot:Connect`、`iot:Publish`、`iot:Subscribe` 和 `iot:Receieve`）来处理任务主题。这些策略允许您连接到消息代理，订阅任务 MQTT 主题，以及在设备和云之间发送和接收 MQTT 消息。有关这些操作的更多信息，请参阅AWS IoT Core 政策行动。

有关 AWS IoT 作业主题的信息，请参阅任务主题。

基本 MQTT 策略示例

下面的示例显示如何使用 `iot:Publish` 和 `iot:Subscribe` 发布和订阅任务和任务执行。

在此示例中：

- 与您所在@@ *###* AWS 区域，例如us-east-1。
- 带有您的 AWS 账户 号码的@@ *## ID*，例如。57EXAMPLE833
- 将 *thing-name* 替换为目标任务所属 IoT 事物的名称，例如 MyIoTThing。

```
{
    "Statement": [
        {
           "Effect": "Allow",
           "Action": [
                "iot:Publish",
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/events/job/*",
                "arn:aws:iot:region:account-id:topic/$aws/events/jobExecution/*",
                "arn:aws:iot:region:account-id:topic/$aws/things/thing-name/jobs/*"
            ]
        }
    ],
    "Version": "2012-10-17"
}
```

AWS IoT Core HTTPS 协议的策略

AWS IoT Core 数据层面的策略还可以使用带有 TLS 身份验证机制的 HTTPS 协议来授权您的设备。在数据层面上，策略使用 iotjobsdata: 前缀以授权您的设备可以执行的任务 API 操作。例如，iotjobsdata:DescribeJobExecution 策略操作授予用户使用 DescribeJobExecution API 的权限。

> **Note**
>
> 数据层面策略操作必须使用 iotjobsdata: 前缀。在控制面板上，操作必须使用 iot: 前缀。有关使用控制面板和数据层面策略操作时的 IAM 策略示例，请参阅 控制面板和数据层面的 IAM 策略示例。

策略操作

下表显示了授权设备使用 API 操作的 AWS IoT Core 策略操作和权限列表。有关可以在数据层面执行的 API 操作的列表，请参阅任务设备 HTTP API。

> **Note**
>
> 这些任务执行策略操作仅适用于 HTTP TLS 终端节点。如果您使用 MQTT 终端节点，则必须之前定义的 MQTT 策略操作。

AWS IoT Core 数据层面的策略操作

策略操作	API 操作	资源类型	描述
iotjobsdata:DescribeJobExecution	DescribeJobExecution	• 作业 • thing	表示检索任务执行的权限。每次请求检索任务执行时，都会检查 iotjobsdata:DescribeJobExecution 权限。
iotjobsdata:GetPendingJobExecutions	GetPendingJobExecutions	thing	表示一个权限，用于为事物检索未处于最终状态的任务的列表。每次请求检索该列表时，都会检查 iotjobsdata:GetPendingJobExecutions 权限。
iotjobsdata:StartNextPendingJobExecution	StartNextPendingJobExecution	thing	表示一个权限，用于为事物获取和启动下一个待处理任务执行。也即，将状态为 QUEUED 的任务执行更新为状态 IN_PROGRESS 。每次请求启动下一个待处理任务执行，都会检查 iot:StartNextPendingJobExecution 权限。
iotjobsdata:UpdateJobExecution	UpdateJobExecution	thing	表示更新任务执行的权限。每次请求更新任务执行的状态时，都会检查 iot:UpdateJobExecution 权限。

基本策略示例

以下是一个策略示例，该 AWS IoT Core 策略授予对任何资源在数据平面 API 操作上执行操作的权限。您可以将策略范围限定为特定资源，例如 IoT 事物。在您的示例中：

- 你所在@@ *###*， AWS 区域 比如us-east-1。
- 带有您的 AWS 账户 号码的@@ *## ID*，例如。57EXAMPLE833
- 将 *thing-name* 替换为 IoT 事物的名称，例如 MyIoTthing。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "iotjobsdata:GetPendingJobExecutions",
                "iotjobsdata:StartNextPendingJobExecution",
                "iotjobsdata:DescribeJobExecution",
                "iotjobsdata:UpdateJobExecution"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:region:account-id:thing/thing-name"
        }
    ]
}
```

必须使用这些策略的一个示例情形是：IoT 设备使用 AWS IoT Core 策略来访问其中一个 API 操作，例如下面的 DescribeJobExecution API 示例：

```
GET /things/thingName/jobs/jobId?
executionNumber=executionNumber&includeJobDocument=includeJobDocument&namespaceId=namespaceId
 HTTP/1.1
```

任务限制

AWS IoT 作业的服务配额或限制与您的服务资源或操作的最大数量相对应 AWS 账户。

活动和并发任务限制

本节将帮助您了解有关活动和并发任务以及适用于这些任务的限制的更多信息。

活动任务和活动任务限制

当您使用 AWS IoT 控制台或 CreateJob API 创建任务时，任务状态会更改为IN_PROGRESS。所有正在进行的任务都是活动任务 并计入活动任务限制。这包括正在推出新任务执行的任务，或者正在等待设备完成任务执行的任务。此限制同时适用于连续任务和快照任务。

并发任务和任务并发限制

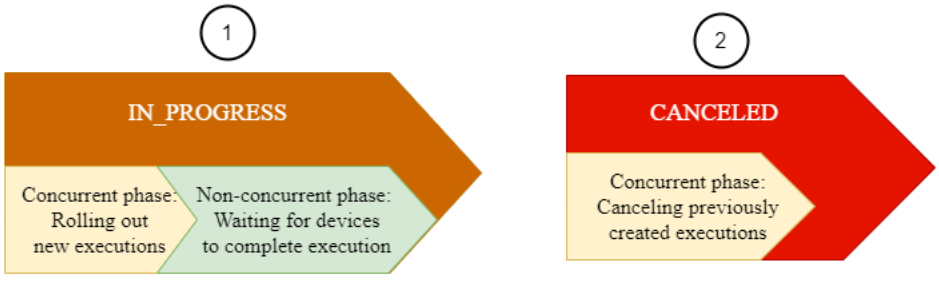
正在推出新任务执行或取消先前创建的任务执行的正在进行的作业均为并发作业，计入作业并发限制。 AWS IoT 任务可以以每分钟 1000 台设备的速度快速推出和取消任务执行。每个任务都是 concurrent，并且仅在短时间内计入任务并发限制。在推出或取消任务执行之后，该任务将不再为并发的，而不计入任务并发限制。在等待设备完成任务执行的同时，您可以使用任务并发创建大量任务。

> **Note**
>
> 如果计划在维护时段内执行的具有可选计划配置和任务文档推出的任务达到选定的 startTime，并且您处于最大任务并发限制，则该计划任务将移至 CANCELED 状态。

要确定作业是否为并发任务，可以在 AWS IoT 控制台中使用作业的IsConcurrent属性，也可以使用DescribeJob或 ListJob API。此限制同时适用于连续任务和快照任务。

要查看您的活动任务和任务并发限制以及其他 AWS IoT 任务配额 AWS 账户 并请求提高限制，请参阅中的AWS IoT 设备管理终端节点和配额。 AWS 一般参考

下图显示了任务并发如何适用于正在进行的任务和正在取消的任务。



> **Note**
>
> 带有可选 SchedulingConfig 的新任务将保持初始状态 SCHEDULED，并在达到选定 startTime 时更新为 IN_PROGRESS。在带有可选 SchedulingConfig 的新任务达到选定

的 startTime，并更新为 IN_PROGRESS 后，它将计入活动任务限制和任务并发限制。状态为 SCHEDULED 的任务将计入活动任务限制，但不会计入任务并发限制。

下表显示了适用于活跃和并发任务以及任务状态的并发和非并发阶段的限制。

活动和并发任务限制

作业状态	阶段	活动任务限制	任务并发限制
SCHEDULED	非并发阶段： AWS IoT 作业等待任务startTime 的调度，开始向您的设备发送任务执行通知。此阶段的任务只计入活动任务限制，并且会将 IsConcurrent 属性设置为 false。	适用	不适用
IN_PROGRESS	并发阶段： AWS IoT 作业接受创建任务的请求并开始向您的设备推出任务执行通知。这个阶段的任务是并发的（由 IsConcurrent 属性设置为 true 来指示），并同时计入活动任务和任务并发限制。	适用	适用
	非并发阶段： AWS IoT 作业等待设备报告其任务执行结果。此阶段的任务只计入活动任务限制，并且会将 IsConcurrent 属性设置为 false。	适用	不适用
Canceled	并发阶段： AWS IoT 作业接受取消任务的请求并开始取消之前为您的设备创建的任务执行。此阶段的任务是并发的，并且会将 IsConcurrent 属性设置为 true。取消任务和任务执行后，该任务将不再为并发的，而不计入任务并发限制。	不适用	适用

> **Note**
>
> 定期维护时段的最长持续时间为 23 小时 50 分钟。

AWS IoT 安全隧道

当设备部署在远程站点的受限防火墙后方时，您需要一种方法来获取对这些设备的访问权限，以便进行故障排除、配置更新和其它操作任务。使用安全隧道通过由管理的安全连接与远程设备建立双向通信。AWS IoT安全隧道不要求更新现有的入站防火墙规则，因此，您可以在远程站点上保留防火墙规则提供的相同安全级别。

例如，位于几百英里之外的工厂的传感器设备在测量工厂温度时遇到问题。您可以使用安全隧道打开并快速启动到该传感器设备的会话。确定问题（例如，配置文件错误）后，可以重置文件并通过同一会话重新启动传感器设备。与更传统的故障排除（例如，向工厂派遣技术人员调查传感器设备）相比，安全隧道可减少事件响应和恢复时间以及运营成本。

什么是安全隧道？

使用安全隧道访问部署在远程站点端口限制防火墙后的设备。您可以使用 AWS Cloud从作为源设备的笔记本电脑或台式计算机连接到目标设备。源和目标通过使用在每台设备上运行的开源本地代理进行通信。本地代理使用防火墙允许 AWS Cloud 的开放端口（通常为 443）与通信。通过隧道传输的数据已使用传输层安全性 (TLS) 进行加密。

主题

- 安全隧道概念
- 安全隧道的工作原理
- 安全隧道生命周期

安全隧道概念

在与远程设备建立通信时，安全隧道会使用以下术语。有关安全隧道工作原理的信息，请参阅 安全隧道的工作原理。

客户端访问令牌 (CAT)

> 创建新隧道时由安全隧道生成的一对令牌。源设备和目标设备使用 CAT 连接到安全隧道服务。在连接到隧道时只能使用 CAT 一次。要重新连接到隧道，请使用 RotateTunnelAccessTokenAPI 操作或 rotate-tunnel-access-tokenCLI 命令轮换客户端访问令牌。

客户端令牌

由客户端生成的唯一值，AWS IoT 安全隧道可用于随后与同一隧道的所有重试连接。该字段是可选的。如果未提供客户端令牌，则客户端访问令牌 (CAT) 对于同一隧道只能使用一次。使用同一 CAT 的后续连接尝试将遭到拒绝。有关使用客户端令牌的更多信息，请参阅中的本地代理参考实现 GitHub。

目标应用程序

在目标设备上运行的应用程序。例如，目标应用程序可以是使用安全隧道建立 SSH 会话的 SSH 守护程序。

目标设备

要访问的远程设备。

设备代理

连接 AWS IoT 设备网关并通过 MQTT 监听新隧道通知的物联网应用程序。有关更多信息，请参阅 IoT 代理代码段。

本地代理

在源设备和目标设备上运行并在安全隧道和设备应用程序之间中继数据流的软件代理。本地代理可以在源模式或目标模式下运行。有关更多信息，请参阅 本地代理。

源设备

操作员用于启动与目标设备（通常是笔记本电脑或台式计算机）的会话的设备。

隧道

一种逻辑通路 AWS IoT ，用于在源设备和目的设备之间实现双向通信。

安全隧道的工作原理

下面显示了安全隧道如何在源设备和目标设备之间建立连接。有关客户端访问令牌 (CAT) 等不同术语的信息，请参阅 安全隧道概念。

1. 打开隧道

   要打开隧道以启动与远程目标设备的会话，您可以使用、open-t AWS CLI unnel 命令或 OpenTunnelAPI。 AWS Management Console

2. 下载客户端访问令牌对

打开隧道后，您可以下载源和目标的客户端访问令牌 (CAT) 并将其保存在源设备上。在启动本地代理之前，必须先检索 CAT 并立即保存。

3. 在目标模式下启动本地代理

   已安装且正在目标设备上运行的 IoT 代理，会订阅 MQTT 预留主题 `$aws/things/thing-name/tunnels/notify` 并接收 CAT。在这里，*thing-n* ame 是你为目的地创建的 AWS IoT 东西的名称。有关更多信息，请参阅 安全隧道主题。

   然后，IoT 代理使用 CAT 在目标模式下启动本地代理，再在隧道的目标端设置连接。有关更多信息，请参阅 IoT 代理代码段。

4. 在源模式下启动本地代理

   打开隧道后， AWS IoT Device Management 提供源文件的 CAT，供您在源设备上下载。您可以使用 CAT 在源模式下启动本地代理，然后连接隧道的源端。有关本地代理的更多信息，请参阅 本地代理。

5. 打开 SSH 会话

   由于隧道的两端都已连线，您可以使用源端的本地代理来开启 SSH 会话。

有关如何使用打开隧道和启动 SSH 会话的更多信息，请参阅打开隧道并启动与远程设备的 SSH 会话。 AWS Management Console

以下视频不仅会介绍安全隧道的工作原理，还会全程指导您设置与 Raspberry Pi 设备的 SSH 会话。

安全隧道生命周期

隧道的状态可能是 `OPEN` 或 `CLOSED`。与隧道的连接状态可能是 `CONNECTED` 或 `DISCONNECTED`。下面展示了不同隧道和连接状态的工作原理。

1. 打开隧道时，其状态为 `OPEN`。隧道的源和目标连接状态设置为 `DISCONNECTED`。
2. 当设备（源或目标）连接到隧道时，相应的连接状态会更改为 `CONNECTED`。
3. 当设备与隧道断开连接且隧道状态保持为 `OPEN` 时，相应的连接状态会更改回 `DISCONNECTED`。只要隧道状态保持为 `OPEN`，设备就可以重复连接隧道并断开与隧道的连接。

> **Note**
>
> 在连接到隧道时只能使用客户端访问令牌 (CAT) 一次。要重新连接到隧道，请使用 RotateTunnelAccessTokenAPI 操作或 rotate-tunnel-access-tokenCLI 命令轮换客户端访问令牌。有关示例，请参阅通过轮换客户端访问令牌来解决 AWS IoT 安全隧道连接问题。

4. 在调用 `CloseTunnel` 时或者隧道状态保持 `OPEN` 的时间长于 `MaxLifetimeTimeout` 值时，隧道的状态会变为 `CLOSED`。您可以在调用 `OpenTunnel` 时配置 `MaxLifetimeTimeout`。如果您未指定值，则 `MaxLifetimeTimeout` 默认为 12 小时。

> **Note**
>
> 隧道状态为 `CLOSED` 时，无法重新打开该隧道。

5. 当隧道可见时，您可以调用 `DescribeTunnel` 和 `ListTunnels` 来查看隧道元数据。隧道在 AWS IoT 控制台中至少可以显示三个小时，然后才会被删除。

AWS IoT 安全隧道教程

AWS IoT 安全隧道可帮助客户通过由管理的安全连接与防火墙后面的远程设备建立双向通信。 AWS IoT

要演示 AWS IoT 安全隧道，请使用我们的AWS IoT 安全隧道演示。 GitHub

以下教程会帮助您了解如何开始使用安全隧道。您将了解如何：

1. 使用快速设置和手动设置方法创建安全隧道以访问远程设备。
2. 使用手动设置方法时配置本地代理，并连接到隧道以访问目标设备。
3. 无需配置本地代理，即可从浏览器 SSH 到远程设备。
4. 将使用 AWS CLI 或使用手动设置方法创建的隧道转换为使用快速设置方法。

本部分中的教程

本节中的教程重点介绍如何使用 AWS Management Console 和 AWS IoT API 参考创建隧道。在 AWS IoT 控制台中，您可以从隧道中心页面或您创建的事物的详细信息页面创建隧道。有关更多信息，请参阅 AWS IoT 控制台中的隧道创建方法。

以下是本节中的教程：

- 打开隧道并使用基于浏览器的 SSH 访问远程设备

  本教程介绍如何使用快速设置方法从 Tunnels hub（隧道中心）页面打开隧道。您还将学习如何使用基于浏览器的 SSH 通过控制台中的上下文命令行界面访问远程设备。 AWS IoT

- 使用手动设置打开隧道并连接到远程设备

  本教程介绍如何使用手动设置方法从 Tunnels hub（隧道中心）页面打开隧道。您还将学习如何从源设备中的终端配置和启动本地代理并连接到隧道。

- 为远程设备打开隧道并使用基于浏览器的 SSH

  本教程介绍如何从您创建的事物的详细信息页面打开隧道。您将了解如何创建新隧道和如何使用现有隧道。现有隧道对应于为设备创建的最新开放隧道。您还可以使用基于浏览器的 SSH 来访问远程设备。

AWS IoT 安全隧道教程

- 打开隧道并启动与远程设备的 SSH 会话
- 为远程设备打开隧道并使用基于浏览器的 SSH

打开隧道并启动与远程设备的 SSH 会话

在这些教程中，您将学习如何远程访问防火墙后面的设备。您无法启动到设备的直接 SSH 会话，因为防火墙阻止所有入站流量。这些教程介绍如何打开隧道，然后使用该隧道启动到远程设备的 SSH 会话。

教程的先决条件

运行本教程的先决条件可能会有所不同，具体取决于您使用手动设置方法还是快速设置方法来打开隧道和访问远程设备。

> **Note**
>
> 对于这两种设置方法，您必须都允许端口 443 上的出站流量。

- 有关快速设置方法教程的先决条件的信息，请参阅快速设置方法的先决条件。

- 有关手动设置方法教程的先决条件的信息，请参阅手动设置方法的先决条件。如果使用此设置方法，必须在源设备上配置本地代理。要下载本地代理源代码，请参阅上的本地代理参考实现 GitHub。

隧道设置方法

在这些教程中，您将学习打开隧道和连接到远程设备的手动设置方法和快速设置方法。下表显示了这两种安装方法之间的差异。创建隧道后，您可以使用浏览器内命令行界面通过 SSH 连接到远程设备。如果放错了令牌或隧道断开连接，可以发送新的访问令牌以重新连接到隧道。

快速设置方法和手动设置方法

标准	快速设置	手动设置
创建隧道	使用默认的可编辑配置创建新隧道。要访问远程设备，只能使用 SSH 作为目标服务。	通过手动指定隧道配置来创建隧道。可以使用此方法通过 SSH 以外的服务连接到远程设备。
访问令牌	如果在创建隧道时指定了事物名称，则目标访问令牌将通过预留的 MQTT 主题自动传送到您的设备。您无需在源设备上下载或管理令牌。	您必须在源设备上手动下载或管理令牌。如果在创建隧道时指定了事物名称，则目标访问令牌将通过预留的 MQTT 主题自动传送到远程设备。
本地代理	系统会自动为您配置基于 Web 的本地代理，用于与设备进行交互。您不必手动配置本地代理。	您必须手动配置和启动本地代理。要配置本地代理，您可以使用 AWS IoT 设备客户端，也可以下载本地代理参考实现 GitHub。

AWS IoT 控制台中的隧道创建方法

本节中的教程介绍如何使用 AWS Management Console 和 OpenTunnelAPI 创建隧道。如果您在创建隧道时配置了目标，则 AWS IoT 安全隧道将通过 MQTT 和保留的 MQTT 主题将目标客户端访问令牌传送到远程设备，)。`$aws/things/RemoteDeviceA/tunnels/notify`收到 MQTT 消息后，远程设备上的 IoT 代理会以目标模式启动本地代理。有关更多信息，请参阅 保留的主题。

> **Note**
>
> 如果要通过另一种方法将目标客户端访问令牌传送到远程设备，则可以省略目标配置。有关更多信息，请参阅 配置远程设备和使用 IoT 代理。

在 AWS IoT 控制台中，您可以使用以下任一方法创建隧道。有关可帮助您了解如何使用这些方法创建隧道的教程的信息，请参阅本部分中的教程。

- 隧道中心

  创建隧道时，您可以指定是使用快速设置方法还是手动设置方法来创建隧道，并提供可选的隧道配置详细信息。配置详细信息还包括目标设备的名称和要用于连接设备的服务。创建隧道后，您可以在浏览器中使用 SSH，也可以在 AWS IoT 控制台之外打开终端来访问您的远程设备。

- 事物详细信息页面

  创建隧道时，除了选择设置方法和提供任何可选的隧道配置详细信息外，还可以指定是使用最新的开放隧道，还是为设备创建新隧道。您无法编辑现有隧道的配置详细信息。您可以使用快速设置方法在浏览器中将访问令牌和 SSH 轮换到远程设备中。要使用此方法打开隧道，您必须在 AWS IoT 注册表中创建了物联网事物（例如`RemoteDeviceA`）。有关更多信息，请参阅在注册表中 AWS IoT 注册设备。

本部分中的教程

- 打开隧道并使用基于浏览器的 SSH 访问远程设备
- 使用手动设置打开隧道并连接到远程设备

打开隧道并使用基于浏览器的 SSH 访问远程设备

您可以使用快速设置方法或手动设置方法来创建隧道。本教程介绍如何使用快速设置方法打开隧道，以及如何使用基于浏览器的 SSH 连接到远程设备。有关演示如何使用手动设置方法打开隧道的示例，请参阅使用手动设置打开隧道并连接到远程设备。

使用快速设置方法时，您可以通过可编辑的默认配置创建新隧道。系统会为您配置基于 Web 的本地代理，并使用 MQTT 将访问令牌自动传递到远程目标设备。创建隧道后，您可以使用控制台中的命令行界面开始与远程设备进行交互。

使用快速设置方法，必须使用 SSH 作为目标服务才能访问远程设备。有关不同设置方法的信息，请参阅隧道设置方法。

快速设置方法的先决条件

- 位于远程设备前面的防火墙必须允许端口 443 上的出站流量。您创建的隧道将使用此端口连接到远程设备。

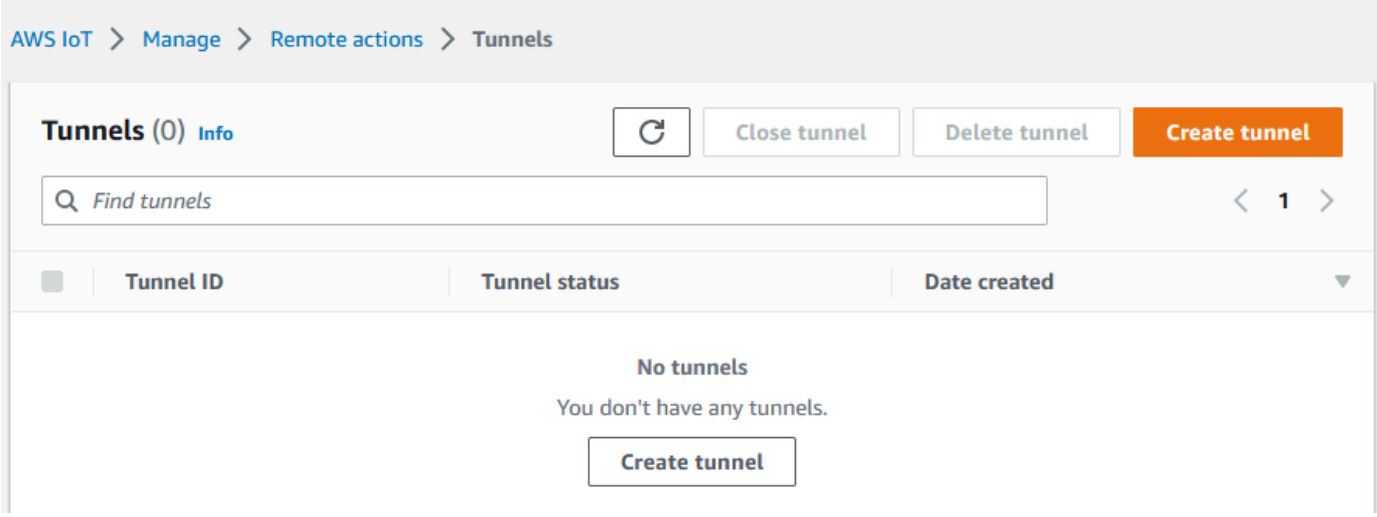
- 您在远程设备上运行 IoT 设备代理（参见IoT 代理代码段），该设备连接到 AWS IoT 设备网关，并配置了 MQTT 主题订阅。有关更多信息，请参阅将设备连接到 AWS IoT 设备网关。
- 您必须在远程设备上运行 SSH 守护进程。

打开隧道

您可以使用、 AWS IoT API 参考或 AWS Management Console，打开安全隧道 AWS CLI。您可以选择配置目标名称，但本教程不要求这样做。如果您配置了目标，安全隧道将使用 MQTT 自动将访问令牌传送到远程设备。有关更多信息，请参阅 AWS IoT 控制台中的隧道创建方法。

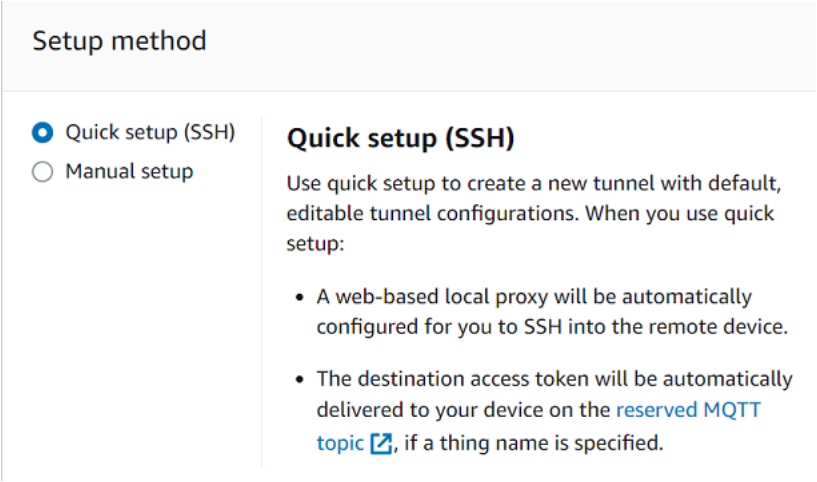
使用控制台打开隧道

1. 转至 AWS IoT 控制台的隧道中心，然后选择 Create tunnel（创建隧道）。



2. 对于本教程，请选择 Quick setup（快速设置）作为隧道创建方法，然后选择 Next（下一步）。

> **Note**
>
> 如果您从所创建事物的详细信息页面创建安全隧道，则可以选择是创建新隧道还是使用现有隧道。有关更多信息，请参阅 为远程设备打开隧道并使用基于浏览器的 SSH。

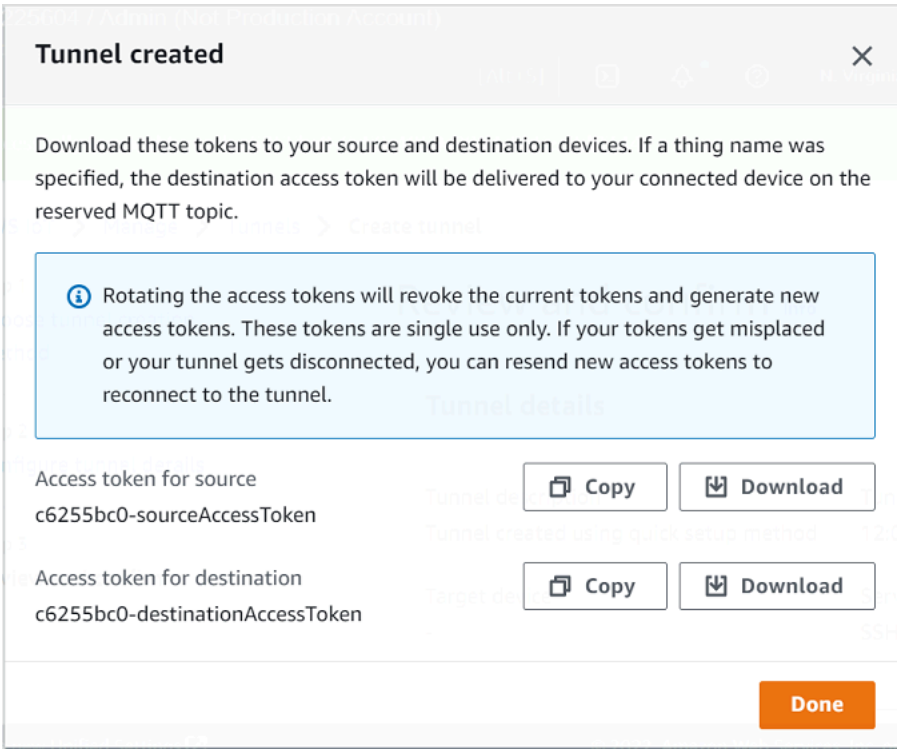


3. 查看并确认隧道配置详细信息。要创建隧道，请选择 Confirm and create（确认并创建）。如果要编辑这些详细信息，请选择 Previous（上一页）返回上一页，然后确认并创建隧道。

> **Note**
>
> 使用快速设置时，无法编辑服务名称。您必须使用 SSH 作为 Service（服务）。

4. 要创建隧道，请选择 Done（完成）。

   对于本教程，您不必下载源或目标访问令牌。这些令牌在连接到隧道时只能使用一次。如果您的隧道断开连接，您可以生成新令牌并将其发送到远程设备以重新连接到隧道。有关更多信息，请参阅重新发送隧道访问令牌。



使用 API 打开隧道

要打开新隧道，可以使用 OpenTunnelAPI 操作。

> **Note**
>
> 您只能从 AWS IoT 控制台使用快速设置方法创建隧道。当您使用 AWS IoT API 参考 API 或时 AWS CLI，它将使用手动设置方法。您可以打开所创建的现有隧道，然后将隧道设置方法更改为使用快速设置。有关更多信息，请参阅 打开现有隧道并使用基于浏览器的 SSH。

下面介绍一个有关如何运行此 API 操作的示例。或者，如果要指定事物名称和目标服务，请使用 `DestinationConfig` 参数。有关演示如何使用此参数的示例，请参阅为远程设备打开新隧道。

```
aws iotsecuretunneling open-tunnel
```

运行此命令将创建新隧道，并为您提供源和目标访问令牌。

```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
```

```
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

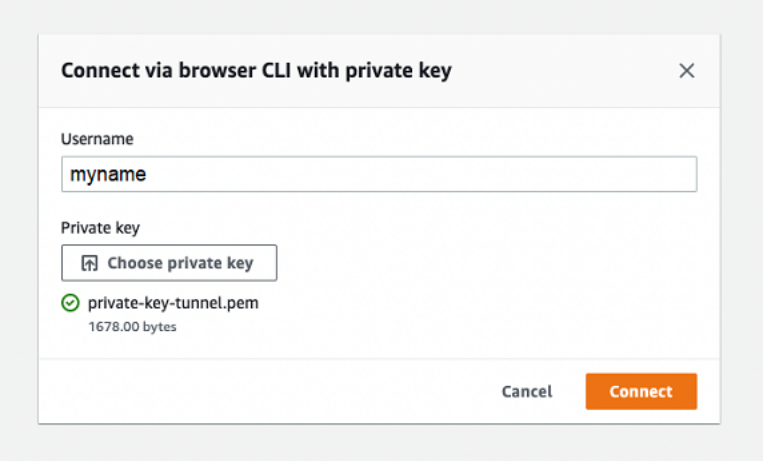
使用基于浏览器的 SSH

使用快速设置方法创建隧道，并且目标设备已连接到隧道后，您可以使用基于浏览器的 SSH 访问远程设备。使用基于浏览器的 SSH，您可以通过在控制台中的上下文命令行界面中输入命令来直接与远程设备通信。此特征使您可以更轻松地与远程设备进行交互，因为您不必在控制台外部打开终端，也不必配置本地代理。

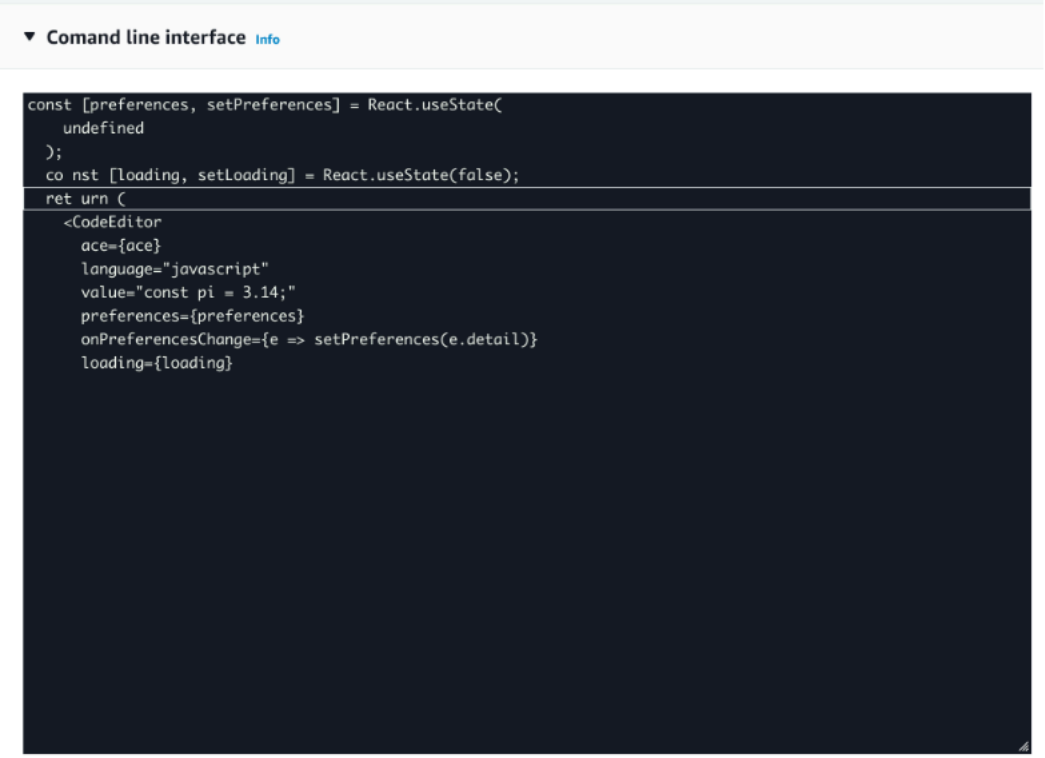
使用基于浏览器的 SSH

1. 转至 AWS IoT 控制台的隧道中心，然后选择所创建的隧道以查看详细信息。
2. 展开 Secure Shell (SSH) [安全外壳 (SSH)] 部分，然后选择 Connect（连接）。
3. 选择是否要通过提供用户名和密码在 SSH 连接中进行身份验证，或者，为了进行更安全的身份验证，您可以使用设备的私钥。如果您使用私有密钥进行身份验证，则可以使用 PEM（PKCS#1、PKCS#8）和 OpenSSH 格式的 RSA、DSA、ECDSA（nistp-*）和 ED25519 密钥类型。

   - 要使用您的用户名和密码进行连接，请选择 Use password（使用密码）。然后，您可以输入您的用户名和密码，并开始使用浏览器内 CLI。
   - 要使用目标设备的私钥进行连接，请选择 Use private key（使用私钥）。指定您的用户名并上传设备的私钥文件，然后选择 Connect（连接）以开始使用浏览器内 CLI。



通过 SSH 连接的身份验证后，您可以快速开始输入命令，并使用浏览器 CLI 与设备进行交互，因为已经为您配置了本地代理。



如果浏览器 CLI 在隧道持续时间后保持打开状态，则可能会超时，从而导致命令行界面断开连接。您可以复制隧道，并启动另一个会话以在控制台本身内与远程设备进行交互。

对使用基于浏览器的 SSH 时出现的问题进行故障排查

以下显示了如何对在使用基于浏览器的 SSH 时可能出现的一些问题进行故障排查。

- 您看到的是错误，而不是命令行界面

  您可能会看到错误，因为您的目标设备已断开连接。您可以选择 Generate new access tokens（生成新访问令牌）来生成新的访问令牌，并使用 MQTT 将令牌发送到您的远程设备。新令牌可用于重新连接到隧道。重新连接到隧道将清除历史记录，并刷新命令行会话。

- 使用私钥进行身份验证时，您会看到隧道已断开连接错误

  您可能会看到此错误是因为您的私钥可能未被目标设备接受。要解决此错误，请检查您上传的用于身份验证的私钥文件。如果您仍然看到错误，请检查您的设备日志。您也可以通过向远程设备发送新访问令牌来尝试重新连接到隧道。

- 使用会话时隧道已关闭

如果您的隧道因保持打开超过指定的持续时间而关闭，您的命令行会话可能会断开连接。隧道一旦关闭就无法重新打开。要重新连接，必须打开另一条通往设备的隧道。

您可以通过复制隧道来创建与已关闭隧道配置相同的新隧道。您可以从 AWS IoT 控制台复制已关闭的隧道。要复制隧道，请选择已关闭的隧道以查看其详细信息，然后选择 Duplicate tunnel（复制隧道）。指定要使用的隧道持续时间，然后创建新隧道。

清理

- 关闭隧道

  我们建议您在使用完隧道后将其关闭。如果隧道的打开时间超过指定的隧道持续时间，隧道也可能会关闭。隧道一旦关闭就无法重新打开。您仍然可以通过选择关闭的隧道，然后选择 Duplicate tunnel（复制隧道）来复制隧道。指定要使用的隧道持续时间，然后创建新隧道。

  - 要从 AWS IoT 控制台关闭单条隧道或多条隧道，请转到 Tunnels hub（隧道中心），选择要关闭的隧道，然后选择 Close tunnel（关闭隧道）。
  - 要使用 AWS IoT API 参考 API 关闭单个或多个隧道，请使用 CloseTunnelAPI。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
```

- 删除隧道

  您可以从中永久删除隧道 AWS 账户。

> **Warning**
>
> 删除是永久性操作，无法撤消。

  - 要从 AWS IoT 控制台删除单条隧道或多条隧道，请转到 Tunnels hub（隧道中心），选择要删除的隧道，然后选择 Delete tunnel（删除隧道）。
  - 要使用 AWS IoT API 参考 API 删除单个或多个隧道，请使用 CloseTunnelAPI。使用 API 时，将 `delete` 标志设置为 `true`。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
```

```
    --delete true
```

使用手动设置打开隧道并连接到远程设备

当您打开隧道时，可以选择快速设置方法或手动设置方法来打开通往远程设备的隧道。本教程介绍如何使用手动设置方法打开隧道，以及如何配置和启动本地代理以连接到远程设备。

使用手动设置方法时，必须在创建隧道时手动指定隧道配置。创建隧道后，您可以在浏览器中使用 SSH 或在 AWS IoT 控制台之外打开终端。本教程说明如何使用控制台外部的终端访问远程设备。您还将学习如何配置本地代理，然后连接到本地代理以与远程设备进行交互。要连接到本地代理，必须在创建隧道时下载源访问令牌。

通过此设置方法，可以使用 SSH 以外的服务（例如 FTP）连接到远程设备。有关不同设置方法的信息，请参阅隧道设置方法。

手动设置方法的先决条件

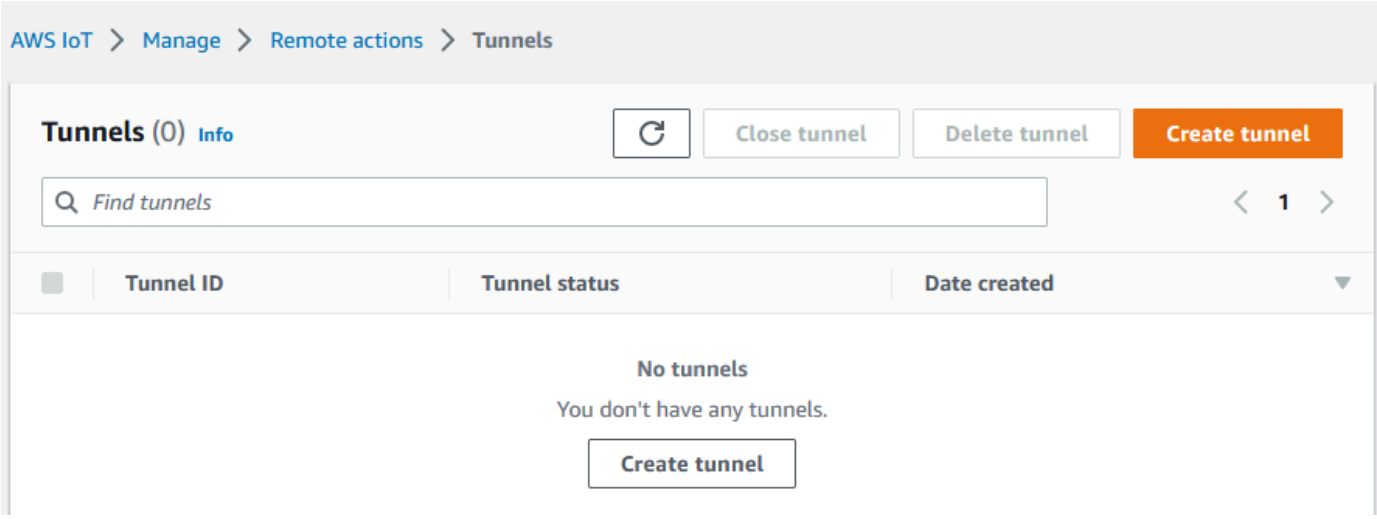
- 位于远程设备前面的防火墙必须允许端口 443 上的出站流量。您创建的隧道将使用此端口连接到远程设备。
- 您在远程设备上运行 IoT 设备代理（参见IoT 代理代码段），该设备连接到 AWS IoT 设备网关，并配置了 MQTT 主题订阅。有关更多信息，请参阅将设备连接到 AWS IoT 设备网关。
- 您必须在远程设备上运行 SSH 守护进程。
- 您已经从中下载了本地代理源代码，GitHub并针对您选择的平台进行了构建。在本教程中，我们将构建的本地代理可执行文件称为 `localproxy`。

打开隧道

您可以使用、 AWS IoT API 参考或 AWS Management Console，打开安全隧道 AWS CLI。您可以选择配置目标名称，但本教程不要求这样做。如果您配置了目标，安全隧道将使用 MQTT 自动将访问令牌传送到远程设备。有关更多信息，请参阅 AWS IoT 控制台中的隧道创建方法。

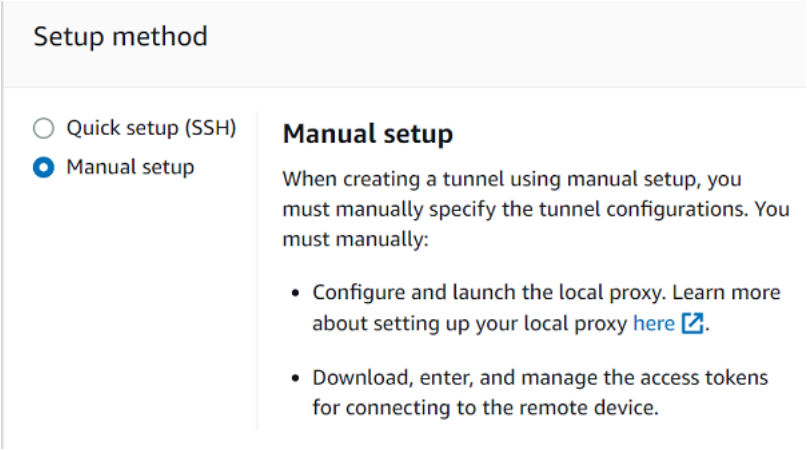
在控制台中打开隧道

1. 转至 AWS IoT 控制台的隧道中心，然后选择 Create tunnel（创建隧道）。



2. 在本教程中，选择 Manual setup（手动设置）作为隧道创建方法，然后选择 Next（下一步）。有关使用快速设置方法创建隧道的信息，请参阅打开隧道并使用基于浏览器的 SSH 访问远程设备。

> **Note**
>
> 如果您从事物详细信息页面创建安全隧道，则可以选择是创建新隧道还是使用现有隧道。有关更多信息，请参阅 为远程设备打开隧道并使用基于浏览器的 SSH。



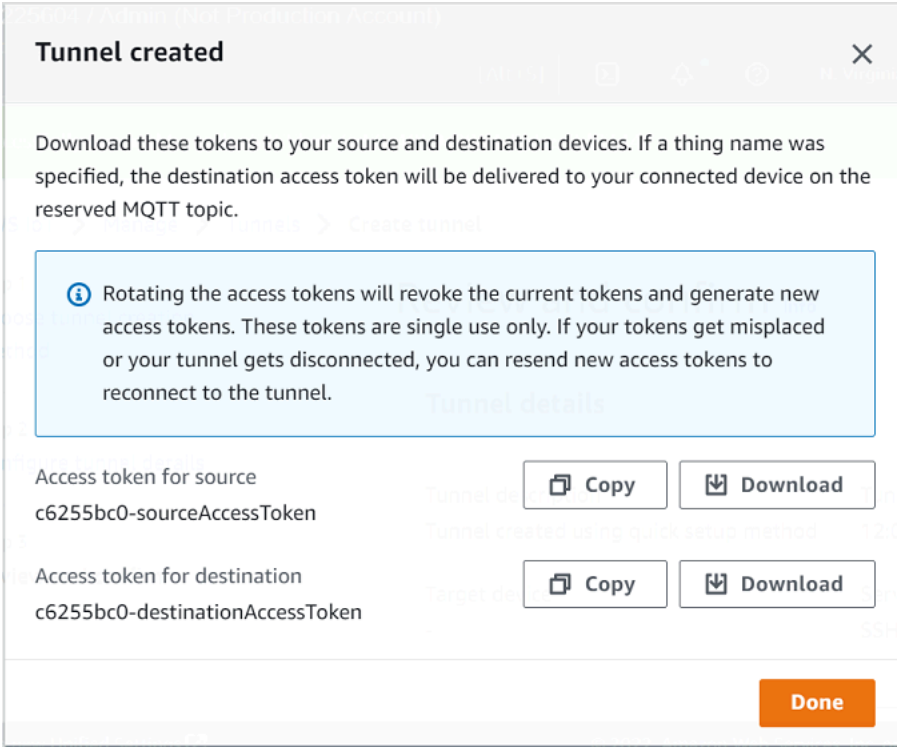
3. （可选）输入隧道的配置设置。您也可以跳过此步骤，继续下一步以创建隧道。

   请以键-值对的形式输入隧道描述、隧道超时持续时间和资源标记，以帮助您识别资源。对于本教程，您可以跳过目标配置。

> **Note**
>
> 不会根据您保持隧道打开的持续时间向您收取费用。只有在创建新隧道时才会产生费用。有关定价的信息，请参阅 AWS IoT Device Management 定价中的安全隧道。

4. 下载客户端访问令牌，然后选择 Done（完成）。选择 Done（完成）之后，令牌将无法下载。

   这些令牌在连接到隧道时只能使用一次。如果您放错了令牌或隧道断开连接，则可以生成新令牌并将其发送到远程设备以重新连接到隧道。



使用 API 打开隧道

要打开新隧道，可以使用 OpenTunnelAPI 操作。您还可以使用 API 指定其他配置，例如隧道持续时间和目标配置。

```
aws iotsecuretunneling open-tunnel \
    --region us-east-1 \
    --endpoint https://api.us-east-1.tunneling.iot.amazonaws.com
```

运行此命令将创建新隧道，并为您提供源和目标访问令牌。

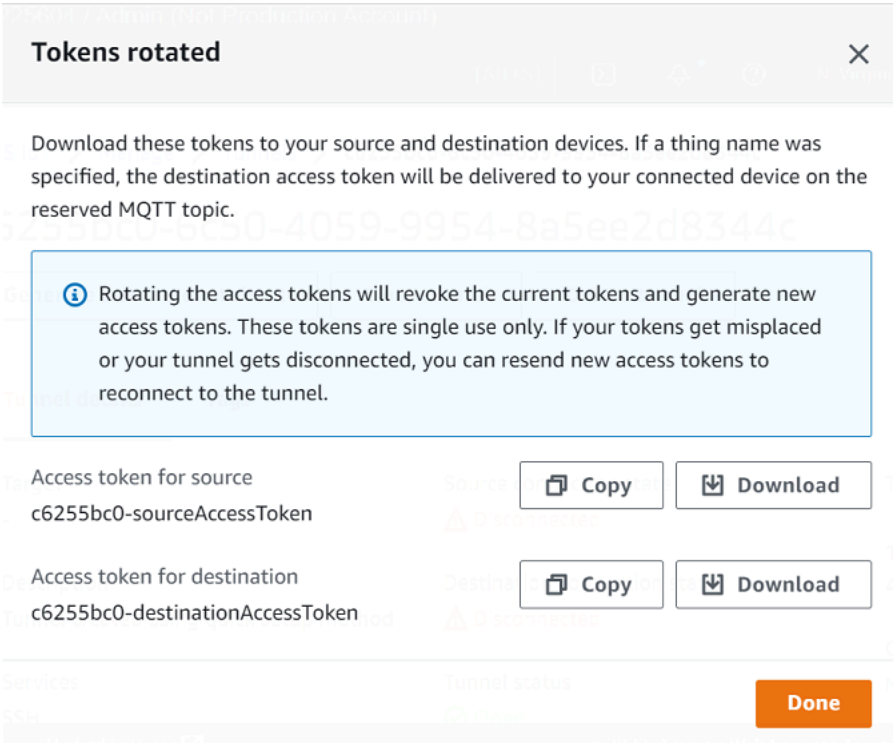
```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

重新发送隧道访问令牌

创建隧道时获得的令牌只能用于连接到隧道一次。如果您放错了访问令牌或隧道断开连接，则可以使用 MQTT 向远程设备重新发送新的访问令牌，无需支付额外费用。 AWS IoT 安全隧道将撤消当前令牌并返回新的访问令牌以重新连接到隧道。

从控制台轮换令牌

1. 前往AWS IoT 控制台的 Tunnels 中心并选择您创建的隧道。
2. 在隧道详细信息页面中，选择 Generate new access tokens（生成新访问令牌），然后选择 Next（下一步）。
3. 为您的隧道下载新访问令牌，然后选择 Done（完成）。这些令牌只能使用一次。如果放错了令牌或隧道断开连接，可以发送新访问令牌。



使用 API 轮换访问令牌

要轮换隧道访问令牌，您可以使用 RotateTunnelAccessTokenAPI 操作撤消当前令牌并返回新的访问令牌以重新连接到隧道。例如，以下命令可轮换目标设备 *RemoteThing1* 的访问令牌。

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --tunnel-id <tunnel-id> \
    --client-mode DESTINATION \
    --destination-config thingName=<RemoteThing1>,services=SSH \
    --region <region>
```

运行此命令将生成以下示例中所示的新访问令牌。然后，如果设备代理设置正确，系统会使用 MQTT 将令牌传送到设备以连接到隧道。

```
{
    "destinationAccessToken": "destination-access-token",
    "tunnelArn": "arn:aws:iot:region:account-id:tunnel/tunnel-id"
}
```

有关说明如何以及何时轮换访问令牌的示例，请参阅通过轮换客户端访问令牌来解决 AWS IoT 安全隧道连接问题。

配置和启动本地代理

要连接到远程设备，请在笔记本电脑上打开终端，然后配置并启动本地代理。本地代理通过安全连接使用安全隧道传输源设备上运行的应用程序发送的数据。 WebSocket 您可以从中下载本地代理源GitHub。

配置本地代理后，复制源客户端访问令牌，然后在源模式下使用它来启动本地代理。以下显示了启动本地代理的示例命令。在以下命令中，本地代理配置用以侦听端口 5555 上的新连接。在此命令中：

- `-r`指定 AWS 区域，该区域必须与创建隧道的区域相同。
- `-s` 指定代理应连接到的端口。
- `-t` 指定客户端令牌文本。

```
./localproxy -r us-east-1 -s 5555 -t source-client-access-token
```

运行此命令将在源模式下启动本地代理。如果在运行此命令后收到以下错误，请设置 CA 路径。有关信息，请参阅上的安全隧道本地代理。 GitHub

```
Could not perform SSL handshake with proxy server: certificate verify failed
```

以下显示了在 source 模式下运行本地代理的示例输出。

```
...
...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-east-1.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-east-1.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

启动 SSH 会话

打开另一个终端并使用以下命令通过连接到端口 5555 上的本地代理来启动新的 SSH 会话。

```
ssh username@localhost -p 5555
```

系统可能会提示您输入 SSH 会话的密码。完成 SSH 会话后，键入 **exit** 以关闭会话。

清理

- 关闭隧道

  我们建议您在使用完隧道后将其关闭。如果隧道的打开时间超过指定的隧道持续时间，隧道也可能会关闭。隧道一旦关闭就无法重新打开。您仍然可以通过打开关闭的隧道，然后选择 Duplicate tunnel（复制隧道）来复制隧道。指定要使用的隧道持续时间，然后创建新隧道。

  - 要从 AWS IoT 控制台关闭单条隧道或多条隧道，请转到 Tunnels hub（隧道中心），选择要关闭的隧道，然后选择 Close tunnel（关闭隧道）。
  - 要使用 AWS IoT API 参考 API 关闭单个或多个隧道，请使用 CloseTunnelAPI 操作。

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    ```

- 删除隧道

  您可以从中永久删除隧道 AWS 账户。

  > **Warning**
  >
  > 删除是永久性操作，无法撤消。

  - 要从 AWS IoT 控制台删除单条隧道或多条隧道，请转到 Tunnels hub（隧道中心），选择要删除的隧道，然后选择 Delete tunnel（删除隧道）。
  - 要使用 AWS IoT API 参考 API 删除单个或多个隧道，请使用 CloseTunnelAPI 操作。使用 API 时，将 `delete` 标志设置为 `true`。

    ```
    aws iotsecuretunneling close-tunnel \
        --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
        --delete true
    ```

为远程设备打开隧道并使用基于浏览器的 SSH

在 AWS IoT 控制台中，您可以从隧道中心创建隧道，也可以从您创建的 IoT 事物的详细信息页面创建隧道。从隧道中心创建隧道时，可以指定是使用快速设置还是手动设置来创建隧道。有关示例教程，请参阅打开隧道并启动与远程设备的 SSH 会话。

从 AWS IoT 控制台的事物详细信息页面创建隧道时，您还可以指定是为该事物创建新隧道还是打开现有隧道，如本教程所示。如果选择现有隧道，可以访问为该设备创建的最新的已打开隧道。然后，可以使用终端中的命令行界面通过 SSH 连接到设备。

先决条件

- 位于远程设备前面的防火墙必须允许端口 443 上的出站流量。您创建的隧道将使用此端口连接到远程设备。
- 您已在 AWS IoT 注册表中创建了一个物联网事物（例如`RemoteDevice1`）。此事物对应于云中您的远程设备表示。有关更多信息，请参阅在 AWS IoT 注册表中注册设备。
- 您在远程设备上运行 IoT 设备代理（参见IoT 代理代码段），该设备连接到 AWS IoT 设备网关，并配置了 MQTT 主题订阅。有关更多信息，请参阅将设备连接到 AWS IoT 设备网关。
- 您必须在远程设备上运行 SSH 守护进程。

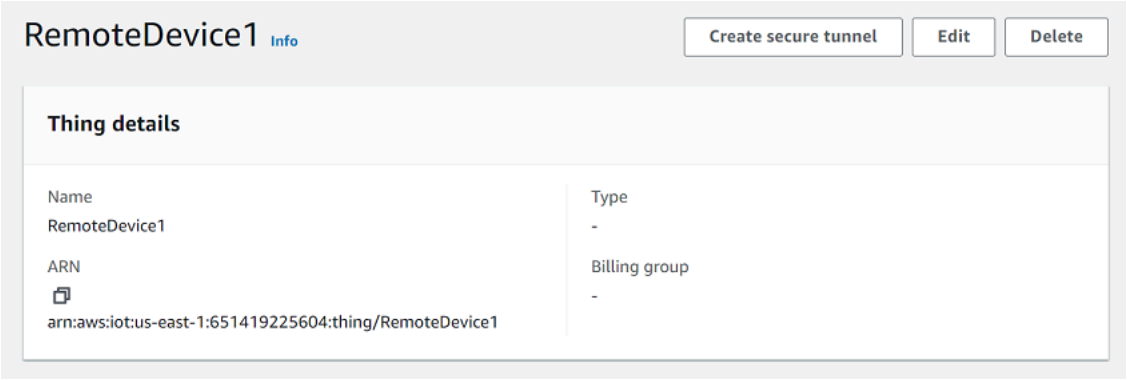
为远程设备打开新隧道

假设您要打开一条通往远程设备 `RemoteDevice1` 的隧道。首先，在 AWS IoT 注册表中创建一个名为 `RemoteDevice1` 的 IoT 事物。然后，您可以使用 AWS Management Console、 AWS IoT API 参考 API 或创建隧道 AWS CLI。

通过在创建隧道时配置目标，安全隧道服务可通过 MQTT 和保留的 MQTT 主题 (`$aws/things/RemoteDeviceA/tunnels/notify`) 将目标客户端访问令牌传递给远程设备。有关更多信息，请参阅 AWS IoT 控制台中的隧道创建方法。

从控制台为远程设备创建隧道

1. 选择事物 `RemoteDevice1` 以查看其详细信息，然后选择 Create secure tunnel（创建安全隧道）。



2. 选择创建新隧道还是打开现有隧道。要创建新隧道，请选择 Create new tunnel（创建新隧道）。然后，您可以选择是使用手动设置方法还是快速设置方法来创建隧道。有关更多信息，请参阅 使用手动设置打开隧道并连接到远程设备 和 打开隧道并使用基于浏览器的 SSH 访问远程设备。

使用 API 为远程设备创建隧道

要打开新隧道，可以使用 OpenTunnelAPI 操作。以下代码显示了运行此命令的示例。

```
aws iotsecuretunneling open-tunnel \
    --region us-east-1 \
    --endpoint https://api.us-east-1.tunneling.iot.amazonaws.com
    --cli-input-json file://input.json
```

以下显示了 input.json 文件的内容。您可以使用 destinationConfig 参数指定目标设备的名称（例如 *RemoteDevice1*）和要用来访问目标设备的服务（例如 *SSH*）。您还可以选择指定其他参数，例如隧道描述和标签。

input.json 的内容

```
{
   "description": "Tunnel to remote device1",
   "destinationConfig": {
      "services": [ "SSH" ],
      "thingName": "RemoteDevice1"
   }
}
```

运行此命令将创建新隧道，并为您提供源和目标访问令牌。

```
{
    "tunnelId": "01234567-89ab-0123-4c56-789a01234bcd",
```

```
    "tunnelArn": "arn:aws:iot:us-
east-1:123456789012:tunnel/01234567-89ab-0123-4c56-789a01234bcd",
    "sourceAccessToken": "<SOURCE_ACCESS_TOKEN>",
    "destinationAccessToken": "<DESTINATION_ACCESS_TOKEN>"
}
```

打开现有隧道并使用基于浏览器的 SSH

假设您使用手动设置方法或使用 AWS IoT API 参考 API 为远程设备创建了隧道。RemoteDevice1然后，您可以打开设备的现有隧道，并选择 Quick setup（快速设置）以使用基于浏览器的 SSH 特征。由于无法编辑现有隧道的配置，因此您无法使用手动设置方法。

要使用基于浏览器的 SSH 特征，您无需下载源访问令牌，也无需配置本地代理。系统将自动为您配置基于 Web 的本地代理，以便您可以开始与远程设备交互。

使用快速设置方法和基于浏览器的 SSH

1. 转到您创建的事物 RemoteDevice1 的详细信息页面，然后 Create secure tunnel（创建安全隧道）。
2. 选择 Use existing tunnel（使用现有隧道）打开您为远程设备创建的最新开放隧道。由于无法编辑隧道配置，因此您无法为隧道使用手动设置方法。要使用快速设置方法，请选择 Quick setup（快速设置）。
3. 继续查看并确认隧道配置详细信息并创建隧道。无法编辑隧道配置。

   创建隧道时，安全隧道将使用 RotateTunnelAccessTokenAPI 操作撤消原始访问令牌并生成新的访问令牌。如果您的远程设备使用 MQTT，这些令牌将通过其订阅的 MQTT 主题自动传送到远程设备。也可以选择将这些令牌手动下载到源设备。

创建隧道后，您可以使用基于浏览器的 SSH 和上下文命令行界面直接从控制台与远程设备进行交互。要使用此命令行界面，请为创建的事物选择隧道，然后在详细信息页面中展开 Command-line interface（命令行界面）部分。由于已经为您配置了本地代理，因此，您可以开始输入命令以快速开始访问远程设备 RemoteDevice1 并与之交互。

有关快速设置方法和使用基于浏览器的 SSH 的更多信息，请参阅打开隧道并使用基于浏览器的 SSH 访问远程设备。

清理

- 关闭隧道

我们建议您在使用完隧道后将其关闭。如果隧道的打开时间超过指定的隧道持续时间，隧道也可能会关闭。隧道一旦关闭就无法重新打开。您仍然可以通过打开关闭的隧道，然后选择 Duplicate tunnel（复制隧道）来复制隧道。指定要使用的隧道持续时间，然后创建新隧道。

- 要从 AWS IoT 控制台关闭单条隧道或多条隧道，请转到 Tunnels hub（隧道中心），选择要关闭的隧道，然后选择 Close tunnel（关闭隧道）。
- 要使用 AWS IoT API 参考 API 关闭单个或多个隧道，请使用 CloseTunnelAPI 操作。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
```

- 删除隧道

  您可以从中永久删除隧道 AWS 账户。

  > **Warning**
  >
  > 删除是永久性操作，无法撤消。

  - 要从 AWS IoT 控制台删除单条隧道或多条隧道，请转到 Tunnels hub（隧道中心），选择要删除的隧道，然后选择 Delete tunnel（删除隧道）。
  - 要使用 AWS IoT API 参考 API 删除单个或多个隧道，请使用 CloseTunnelAPI 操作。使用 API 时，将 `delete` 标志设置为 `true`。

```
aws iotsecuretunneling close-tunnel \
    --tunnel-id "01234567-89ab-0123-4c56-789a01234bcd"
    --delete true
```

本地代理

本地代理通过安全连接使用安全隧道传输源设备上运行的应用程序发送的数据。 WebSocket 您可以从中下载本地代理源GitHub。

本地代理可以在两种模式下运行：`source` 或 `destination`。在源模式下，本地代理与启动 TCP 连接的客户端应用程序在同一设备或网络上运行。在目标模式下，本地代理与目标应用程序一起在远程设备上运行。通过使用隧道多路复用功能，一条隧道一次最多可以支持三个数据流。对于每个数据流，安

全隧道使用多个 TCP 连接，这将减小发生超时的几率。有关更多信息，请参阅 多路复用数据流并在安全隧道中使用同步 TCP 连接。

如何使用本地代理

您可以在源设备和目标设备上运行本地代理，将数据传输到安全隧道终端节点。如果您的设备位于使用 Web 代理的网络中，Web 代理可以在将连接转发到互联网之前拦截它们。在这种情况下，您需要配置本地代理，才能使用 Web 代理。有关更多信息，请参阅 为使用 Web 代理的设备配置本地代理。

本地代理工作流

以下步骤展示如何在源设备和目标设备上运行本地代理。

1. 将本地代理连接到安全隧道

   首先，本地代理必须与安全隧道建立连接。启动本地代理时，请使用以下参数：

   - 用于指定 AWS 区域 在其中打开隧道的`-r`参数。
   - `-t` 参数，用于传递从 `OpenTunnel` 返回的源或目标客户端访问令牌。

   > **Note**
   >
   > 不能同时连接两个使用相同客户端访问令牌值的本地代理。

2. 执行源操作或目标操作

   建立 WebSocket 连接后，本地代理将根据其配置执行源模式或目标模式操作。

   默认情况下，如果出现任何输入/输出 (I/O) 错误或连接意外关闭，本地代理会尝试重新连接到安全隧道。 WebSocket 这会导致 TCP 连接关闭。如果出现任何 TCP 套接字错误，本地代理将通过隧道发送一条消息，通知对方关闭其 TCP 连接。预设情况下，本地代理始终使用 SSL 通信。

3. 终止本地代理

   当您使用隧道后，可以安全地终止本地代理进程。我们建议您通过调用 `CloseTunnel` 显式关闭隧道。活动隧道客户端可能无法在呼叫后立即关闭`CloseTunnel`。

有关如何使用打开隧道和启动 SSH 会话的更多信息，请参阅打开隧道并启动与远程设备的 SSH 会话。 AWS Management Console

本地代理最佳实践

运行本地代理时，请遵循以下最佳实践：

- 避免使用本地代理参数 `-t` 传入访问令牌。我们建议您使用 `AWSIOT_TUNNEL_ACCESS_TOKEN` 环境变量设置本地代理的访问令牌。
- 在操作系统或环境中以最少权限运行本地代理可执行文件。
  - 避免以管理员身份在 Windows 上运行本地代理。
  - 避免以 root 身份在 Linux 和 macOS 上运行本地代理。
- 考虑在单独的主机、容器、沙盒、chroot jail 或虚拟化环境上运行本地代理。
- 使用相关安全标志构建本地代理，具体取决于您的工具链。
- 在具有多个网络接口的设备上，使用 `-b` 参数将 TCP 套接字绑定到用于与目标应用程序通信的网络接口。

命令和输出示例

下面显示了您运行的命令示例以及相应的输出。该示例说明如何在 `source` 和 `destination` 模式下配置本地代理。本地代理将 HTTPS 协议升级 WebSockets 为以建立长寿命连接，然后开始通过连接将数据传输到安全隧道设备端点。

在运行这些命令之前：

您必须已打开隧道并获得源和目标的客户端访问令牌。您还必须已按照前面所述构建本地代理。要构建本地代理，请在 GitHub 存储库中打开本地代理源代码，然后按照说明构建和安装本地代理。

> **Note**
>
> 示例中使用的以下命令使用 `verbosity` 标志来说明运行本地代理后先前描述的不同步骤概览。建议您仅将此标记用于测试。

在源模式下运行本地代理

以下命令显示如何在源模式下运行本地代理。

Linux/macOS

在 Linux 或 macOS 的终端中运行以下命令来配置和启动源上的本地代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
./localproxy -s 5555 -v 5 -r us-west-2
```

其中：

- `-s` 是源侦听端口，用于在源模式下启动本地代理。
- `-v` 表示输出的详细程度，可以是 0 到 6 之间的值。
- `-r` 是打开隧道的终端节点区域。

有关这些参数的更多信息，请参阅使用命令行参数设置的选项。

Windows

在 Windows 中，您可以配置本地代理，操作类似于 Linux 或 macOS 的配置，但是如何定义环境变量则与其它平台不同。在 cmd 窗口中运行以下命令来配置和启动源上的本地代理。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
.\localproxy -s 5555 -v 5 -r us-west-2
```

其中：

- `-s` 是源侦听端口，用于在源模式下启动本地代理。
- `-v` 表示输出的详细程度，可以是 0 到 6 之间的值。
- `-r` 是打开隧道的终端节点区域。

有关这些参数的更多信息，请参阅使用命令行参数设置的选项。

以下显示了在 `source` 模式下运行本地代理的示例输出。

```
...
...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
```

```
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

在目标模式下运行本地代理

以下命令显示如何在目标模式下运行本地代理。

Linux/macOS

在 Linux 或 macOS 的终端中运行以下命令来配置和启动目标上的本地代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
./localproxy -d 22 -v 5 -r us-west-2
```

其中：

- `-d` 是在目标模式下启动本地代理的目标应用程序。
- `-v` 表示输出的详细程度，可以是 0 到 6 之间的值。
- `-r` 是打开隧道的终端节点区域。

有关这些参数的更多信息，请参阅使用命令行参数设置的选项。

Windows

在 Windows 中，您可以配置本地代理，操作类似于 Linux 或 macOS 的配置，但是如何定义环境变量则与其它平台不同。在 cmd 窗口中运行以下命令来配置和启动目标上的本地代理。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
.\localproxy -d 22 -v 5 -r us-west-2
```

其中：

- `-d` 是在目标模式下启动本地代理的目标应用程序。
- `-v` 表示输出的详细程度，可以是 0 到 6 之间的值。
- `-r` 是打开隧道的终端节点区域。

有关这些参数的更多信息，请参阅使用命令行参数设置的选项。

以下显示了在 `destination` 模式下运行本地代理的示例输出。

```
...
...

Starting proxy in destination mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved proxy  server IP: 10.10.0.11
Connected successfully with proxy server
Performing SSL handshake with proxy server
Successfully completed SSL handshake with proxy server
HTTP/1.1 101 Switching Protocols

...

Connection: upgrade
channel-id: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
upgrade: websocket

...

Web socket session ID: 01234567890abc23-00001234-0005678a-b1234c5de677a001-2bc3d456
Web socket subprotocol selected: aws.iot.securetunneling-2.0
```

```
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Setting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
```

为使用 Web 代理的设备配置本地代理

您可以在 AWS IoT 设备上使用本地代理与 AWS IoT 安全隧道 API 进行通信。本地代理通过安全连接使用安全隧道传输设备应用程序发送的数据。 WebSocket 本地代理可以以 `source` 或者 `destination` 模式工作。在 `source` 模式中，它在启动 TCP 连接的同一设备或网络上运行。在 `destination` 模式下，本地代理与目标应用程序一起在远程设备上运行。有关更多信息，请参阅 本地代理。

本地代理需要直接连接到互联网才能使用 AWS IoT 安全隧道。对于使用安全隧道的长期 TCP 连接，本地代理会升级 HTTPS 请求以建立与安全隧道设备 WebSockets 连接端点之一的连接。

如果您的设备位于使用 Web 代理的网络中，Web 代理可以在将连接转发到互联网之前将其拦截。要建立与安全隧道设备连接终端节点的长期连接，请将本地代理配置为使用 Web 代理，如 Websocket 规范中所述。

> **Note**
>
> AWS IoT 设备客户端 不支持使用 Web 代理的设备。要使用 Web 代理，您需要使用本地代理并将其配置为使用 Web 代理，如下所述。

以下步骤显示了本地代理如何与 Web 代理协同工作。

1. 本地代理发送一个 HTTP `CONNECT` 请求发送到包含安全隧道服务远程地址的 Web 代理服务，同时发布的还有 Web 代理身份验证信息。
2. 然后，Web 代理将创建到远程安全隧道终端节点的长期连接。
3. TCP 连接已建立，并且本地代理现在将在源模式和目标模式下工作以进行数据传输。

要完成本流程，请执行以下步骤。

- 构建本地代理

- 配置 Web 代理
- 配置和启动本地代理

构建本地代理

打开 GitHub 存储库中的本地代理源代码，然后按照说明构建和安装本地代理。

配置 Web 代理

本地代理依赖于 HTTP 隧道机制，如 HTTP/1.1 规范中所述。为了符合规范，您的 Web 代理必须允许设备使用 CONNECT 方法。

如何配置 Web 代理取决于您使用的 Web 代理和 Web 代理版本。要确保正确配置 Web 代理，请检查 Web 代理的文档。

要配置 Web 代理，请首先确定 Web 代理 URL 并确认 Web 代理是否支持 HTTP 隧道。稍后在配置和启动本地代理时，将使用 Web 代理 URL。

1. 识别您的 Web 代理 URL

   您的 Web 代理 URL 采用以下格式。

   ```
   protocol://web_proxy_host_domain:web_proxy_port
   ```

   AWS IoT 安全隧道仅支持 Web 代理的基本身份验证。要使用基本身份验证，您必须指定 **username** 和 **password** 作为 Web 代理 URL 的一部分。Web 代理 URL 采用以下格式。

   ```
   protocol://username:password@web_proxy_host_domain:web_proxy_port
   ```

   - *##*可以是 http 或者 https。建议使用 https。
   - *web_proxy_host_domain* 是 Web 代理的 IP 地址或解析为 Web 代理 IP 地址的 DNS 名称。
   - *web_proxy_port* 是 Web 代理正在侦听的端口。
   - Web 代理使用此 **username** 和 **password** 来验证请求。
2. 测试您的 Web 代理 URL

   要确认您的 Web 代理是否支持 TCP 隧道，请使用 curl 命令，并确保您获得 2xx 或 3xx 响应。

   例如，如果您的 Web 代理 URL 是 https://server.com:1235，请使用 proxy-insecure 标记与 curl 命令，因为 Web 代理可能依赖于自签名证书。

```
export HTTPS_PROXY=https://server.com:1235
curl -I https://aws.amazon.com --proxy-insecure
```

如果您的 Web 代理 URL 具有 http 端口（例如，http://server.com:1234），您不一定要使用 proxy-insecure 标记。

```
export HTTPS_PROXY=http://server.com:1234
curl -I https://aws.amazon.com
```

配置和启动本地代理

要配置本地代理以使用 Web 代理，您必须以 DNS 域名或 Web 代理所用的 IP 地址和端口号码来配置 HTTPS_PROXY 环境变量。

配置了本地代理后，您可以使用本地代理，如本自述文件文档所述。

> **Note**
>
> 环境变量声明区分大小写。我们建议您使用全部大写或全部小写字母定义每个变量一次。以下示例显示了全部使用大写字母的环境变量名称。如果同时使用大写字母和小写字母指定相同的变量，则使用小写字母指定的变量优先。

以下命令说明如何将目标上运行的本地代理配置为使用 Web 代理并启动本地代理。

- AWSIOT_TUNNEL_ACCESS_TOKEN：此变量保存目标的客户端访问令牌 (CAT)。
- HTTPS_PROXY：此变量保存用于配置本地代理的 Web 代理 URL 或 IP 地址。

以下示例中显示的命令取决于您使用的操作系统以及 Web 代理侦听的是 HTTP 端口还是 HTTPS 端口。

Web 代理侦听 HTTP 端口

如果您的 Web 代理正在侦听 HTTP 端口，您可以为 HTTPS_PROXY 变量提供 Web 代理 URL 或 IP 地址。

Linux/macOS

在 Linux 或 macOS 中，在终端中运行以下命令，以配置和启动目标上的本地代理，使用侦听 HTTP 端口的 Web 代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:proxy.example.com:1234
./localproxy -r us-east-1 -d 22
```

如果您必须使用代理进行身份验证，则必须指定 **username** 和 **password** 作为 HTTPS_PROXY 变量的一部分。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http://username:password@proxy.example.com:1234
./localproxy -r us-east-1 -d 22
```

Windows

在 Windows 中，您可以配置本地代理，操作类似于 Linux 或 macOS 的配置，但是如何定义环境变量则与其它平台不同。在 cmd 窗口中运行以下命令以配置和启动目标上的本地代理，以使用侦听 HTTP 端口的 Web 代理。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://proxy.example.com:1234
.\localproxy -r us-east-1 -d 22
```

如果您必须使用代理进行身份验证，则必须指定 **username** 和 **password** 作为 HTTPS_PROXY 变量的一部分。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://username:password@10.15.20.25:1234
.\localproxy -r us-east-1 -d 22
```

侦听 HTTPS 端口的 Web 代理

如果您的 Web 代理正在侦听 HTTPS 端口，请运行以下命令。

> **Note**
>
> 如果您为 Web 代理使用自签名证书，或者在不支持本机 OpenSSL 和默认配置的操作系统上运行本地代理，则必须按照存储库中 “证书设置” 部分所述设置 Web 代理证书。 GitHub

以下命令看起来与您为 HTTP 代理配置 Web 代理的方式相似，但您还需要指定您安装的证书文件的路径，如前所述。

Linux/macOS

在 Linux 或 macOS 中，在终端中运行以下命令，以配置目标上运行的本地代理，使其使用侦听 HTTPS 端口的 Web 代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:proxy.example.com:1234
./localproxy -r us-east-1 -d 22 -c /path/to/certs
```

如果您必须使用代理进行身份验证，则必须指定 **username** 和 **password** 作为 HTTPS_PROXY 变量的一部分。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http://username:password@proxy.example.com:1234
./localproxy -r us-east-1 -d 22 -c /path/to/certs
```

Windows

在 Windows 中，在 cmd 窗口中运行以下命令以配置和启动目标上运行的本地代理，以使用侦听 HTTP 端口的 Web 代理。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://proxy.example.com:1234
.\localproxy -r us-east-1 -d 22 -c \path\to\certs
```

如果您必须使用代理进行身份验证，则必须指定 **username** 和 **password** 作为 HTTPS_PROXY 变量的一部分。

```
set AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
set HTTPS_PROXY=http://username:password@10.15.20.25:1234
```

```
.\localproxy -r us-east-1 -d 22 -c \path\to\certs
```

命令和输出示例

下面显示了在 Linux 操作系统上运行的命令示例以及相应的输出。该示例显示了正在侦听 HTTP 端口的 Web 代理，以及如何在 source 和 destination 模式下使用 Web 代理配置本地代理。在运行这些命令之前，您必须已打开隧道并获得源和目标的客户端访问令牌。您还必须已构建本地代理并按照前面所述配置 Web 代理。

以下是启动本地代理后的步骤概览。本地代理：

- 标识 Web 代理 URL，以便它可以使用 URL 连接到代理服务器。
- 建立与 Web 代理的 TCP 连接。
- 发送 HTTP CONNECT 请求到 Web 代理，并等待 HTTP/1.1 200 响应，这表示连接已建立。
- 将 HTTPS 协议升级 WebSockets 为以建立长寿命连接。
- 开始通过与安全隧道设备终端节点的连接进行数据传输。

> **Note**
>
> 示例中使用的以下命令使用 verbosity 标志来说明运行本地代理后先前描述的不同步骤概览。建议您仅将此标记用于测试。

在源模式下运行本地代理

以下命令显示如何在源模式下运行本地代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:username:password@10.15.10.25:1234
./localproxy -s 5555 -v 5 -r us-west-2
```

以下显示了在 source 模式下运行本地代理的示例输出。

```
...

Parsed basic auth credentials for the URL
Found Web proxy information in the environment variables, will use it to connect via
 the proxy.
```

```
...

Starting proxy in source mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved Web proxy IP: 10.10.0.11
Connected successfully with Web Proxy
Successfully sent HTTP CONNECT to the Web proxy
Full response from the Web proxy:
HTTP/1.1 200 Connection established
TCP tunnel established successfully
Connected successfully with proxy server
Successfully completed SSL handshake with proxy server
Web socket session ID: 0a109afffee745f5-00001341-000b8138-cc6c878d80e8adb0-f186064b
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Seting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
Resolved bind IP: 127.0.0.1
Listening for new connection on port 5555
```

在目标模式下运行本地代理

以下命令显示如何在目标模式下运行本地代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=${access_token}
export HTTPS_PROXY=http:username:password@10.15.10.25:1234
./localproxy -d 22 -v 5 -r us-west-2
```

以下显示了在 destination 模式下运行本地代理的示例输出。

```
...

Parsed basic auth credentials for the URL
Found Web proxy information in the environment variables, will use it to connect via
 the proxy.
```

```
...

Starting proxy in destination mode
Attempting to establish web socket connection with endpoint wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Resolved Web proxy IP: 10.10.0.1
Connected successfully with Web Proxy
Successfully sent HTTP CONNECT to the Web proxy
Full response from the Web proxy:
HTTP/1.1 200 Connection established
TCP tunnel established successfully
Connected successfully with proxy server
Successfully completed SSL handshake with proxy server
Web socket session ID: 06717bfffed3fd05-00001355-000b8315-da3109a85da804dd-24c3d10d
Web socket subprotocol selected: aws.iot.securetunneling-2.0
Successfully established websocket connection with proxy server: wss://
data.tunneling.iot.us-west-2.amazonaws.com:443
Seting up web socket pings for every 5000 milliseconds
Scheduled next read:

...

Starting web socket read loop continue reading...
```

多路复用数据流并在安全隧道中使用同步 TCP 连接

您可以借助安全隧道多路复用功能，为每条隧道使用多个数据流。利用多路复用功能，您可以使用多个数据流对设备进行故障排除。您还可以通过取消构建、部署和启动多个本地代理或向同一设备打开多个隧道来减少操作负载。例如，对于需要发送多个 HTTP 和 SSH 数据流的 Web 浏览器，可以使用多路复用。

对于每个数据流， AWS IoT 安全隧道支持同步的 TCP 连接。使用同步连接可减小客户端发出多个请求时发生超时的几率。例如，它可以减少远程访问目标设备的本地 Web 服务器时的加载时间。

以下各节详细介绍了多路复用和使用同步 TCP 连接及其不同的使用案例。

主题

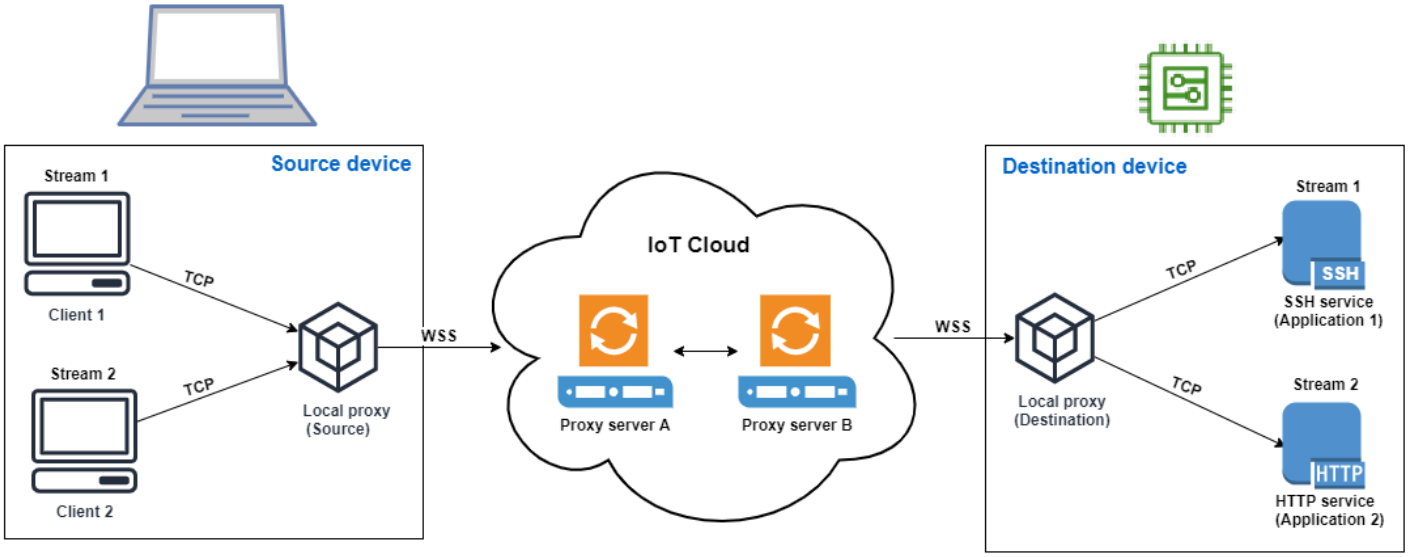
- 在安全隧道中多路复用多个数据流
- 在安全隧道中使用同步 TCP 连接

在安全隧道中多路复用多个数据流

您可以对使用多个连接或端口的设备使用多路复用功能。当您需要与远程设备建立多个连接以解决任何问题时，也可以使用多路复用功能。例如，对于需要发送多个 HTTP 和 SSH 数据流的 Web 浏览器，可以使用此功能。通过多路复用隧道将来自两个数据流的应用程序数据同时发送到设备。

使用案例示例

假设您需要连接到设备上的 Web 应用程序来更改某些联网参数，同时通过终端发出 shell 命令，以验证设备是否使用新的联网参数正常运行。在这种情况下，您可能需要通过 HTTP 和 SSH 连接到设备并传输两个并行数据流，才能同时访问 Web 应用程序和终端。通过多路复用功能，这两个独立的流可以同时通过同一隧道传输。



如何设置多路复用隧道

以下流程将指导您如何设置多路复用隧道，以便使用需要连接到多个端口的应用程序对设备进行故障排除。您将设置一个隧道，其中包含两个多路复用流：一个 HTTP 流和一个 SSH 流。

1. （可选）创建配置文件

   您可以选择使用配置文件来配置源和目标设备。如果您的端口映射可能会经常发生更改，请使用配置文件。如果您希望使用 CLI 显式指定端口映射，或者如果您不需要在指定侦听端口上启动本地代理，则可跳过此步骤。有关如何使用配置文件的更多信息，请参阅中的通过--config 设置的 GitHub选项。

   1. 在源设备上，在将运行本地代理的文件夹中，创建一个名为 `Config` 的配置文件夹。在此文件夹中，使用以下内容创建一个名为 `SSHSource.ini` 的文件：

```
HTTP1 = 5555
SSH1 = 3333
```

2. 在目标设备上，在将运行本地代理的文件夹中，创建一个名为 Config 的配置文件夹。在此文件夹中，使用以下内容创建一个名为 SSHDestination.ini 的文件：

```
HTTP1 = 80
SSH1 = 22
```

2. 打开隧道

使用 OpenTunnel API 操作或 open-tunnel CLI 命令打开隧道。通过将SSH1和指定HTTP1为服务以及与您的远程设备对应 AWS IoT 的事物的名称来配置目的地。您的 SSH 和 HTTP 应用程序正在此远程设备上运行。您必须已经在 AWS IoT 注册表中创建了物联网事物。有关更多信息，请参阅 如何使用注册表管理事物。

```
aws iotsecuretunneling open-tunnel \
 --destination-config thingName=RemoteDevice1,services=HTTP1,SSH1
```

运行此命令将生成源和目标访问令牌，您将使用这些令牌运行本地代理。

```
{
 "tunnelId": "b2de92a3-b8ff-46c0-b0f2-afa28b00cecd",
 "tunnelArn": "arn:aws:iot:us-west-2:431600097591:tunnel/b2de92a3-b8ff-46c0-b0f2-
afa28b00cecd",
 "sourceAccessToken": source_client_access_token,
 "destinationAccessToken": destination_client_access_token
}
```

3. 配置和启动本地代理

在运行本地代理之前，请先设置 AWS IoT 设备客户端，或者从中下载本地代理源代码GitHub并针对您选择的平台进行构建。之后，您可以启动目标和源本地代理以连接到安全隧道。有关配置和使用本地代理的更多信息，请参阅如何使用本地代理。

> **Note**
>
> 在源设备上，如果您不使用任何配置文件或使用 CLI 指定端口映射，则仍可以使用相同的命令来运行本地代理。在源模式下，本地代理将自动为您选取要使用的可用端口和映射。

Start local proxy using configuration files

运行以下命令可使用配置文件在源和目标模式下运行本地代理。

```
// ----------------- Start the destination local proxy -----------------------
./localproxy -r us-east-1 -m dst -t destination_client_access_token

// ----------------- Start the source local proxy ---------------------------
// You also run the same command below if you want the local proxy to
// choose the mappings for you instead of using configuration files.
./localproxy -r us-east-1 -m src -t source_client_access_token
```

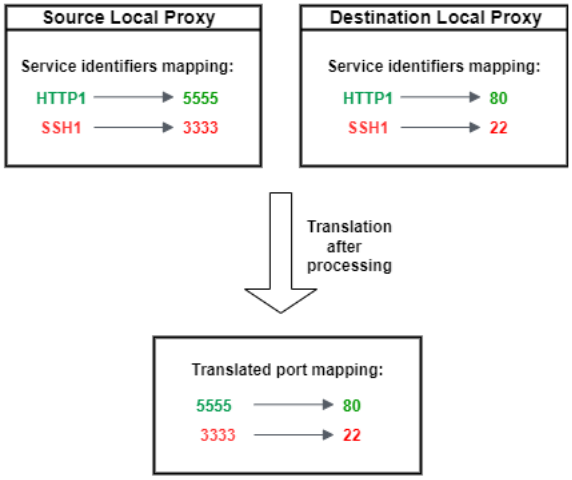
Start local proxy using CLI port mapping

运行以下命令可通过使用 CLI 显式指定端口映射，以便在源和目标模式下运行本地代理。

```
// ----------------- Start the destination local proxy
 ----------------------------------
./localproxy -r us-east-1 -d HTTP1=80,SSH1=22 -t destination_client_access_token

// ----------------- Start the source local proxy
 ---------------------------------------
./localproxy -r us-east-1 -s HTTP1=5555,SSH1=33 -t source_client_access_token
```

现在可以通过多路复用隧道同时传输来自 SSH 和 HTTP 连接的应用程序数据。如下面的地图中所示，服务标识符充当可读格式，用于转换源设备和目标设备之间的端口映射。利用此配置，安全隧道将来自源设备上的端口 *5555* 的所有传入 HTTP 流量转发到目标设备上的端口 *80*，并将来自端口 *3333* 的所有传入 SSH 流量转发到目标设备上的端口 *22*。



在安全隧道中使用同步 TCP 连接

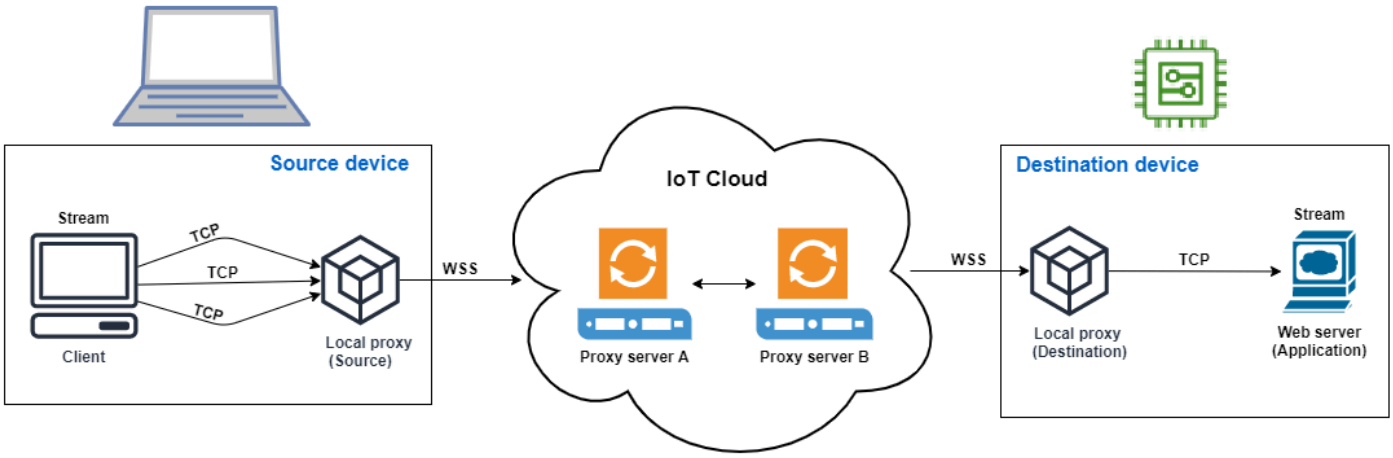
AWS IoT 安全隧道支持每个数据流同时使用多个 TCP 连接。当您需要建立与远程设备的同步连接时，可以使用此功能。使用同步 TCP 连接可减小客户端发出多个请求时发生超时的几率。例如，当访问正在运行多个组件的 Web 服务器时，同步 TCP 连接可减少加载站点所需的时间。

> **Note**
>
> 同时 TCP 连接的带宽限制为每秒 800 千字节。 AWS 账户 AWS IoT 安全隧道可以根据传入请求的数量为您配置此限制。

使用案例示例

假设您需要远程访问目标设备的本地 Web 服务器，并且该 Web 服务器上运行了多个组件。利用单一 TCP 连接，在尝试访问此 Web 服务器时，顺序加载会增加在站点上加载资源所花费的时间。同步 TCP 连接可以满足站点的资源要求来减少加载时间，从而减少访问时间。下图说明如何支持同步 TCP 连接，以便将数据流传输到远程设备上运行的 Web 服务器应用程序。

> **Note**
>
> 如果要使用隧道访问远程设备上运行的多个应用程序，则可以使用隧道多路复用功能。有关更多信息，请参阅 在安全隧道中多路复用多个数据流。



如何使用同步 TCP 连接

以下过程将为您演练如何使用同步 TCP 连接来访问远程设备上的 Web 浏览器。当客户端有多个请求时，AWS IoT 安全隧道会自动设置同时的 TCP 连接来处理这些请求，从而缩短加载时间。

1. 打开隧道

   使用 OpenTunnel API 操作或 open-tunnel CLI 命令打开隧道。通过将 HTTP 指定为服务并指定与远程设备对应的 AWS IoT 事物的名称来配置目标。您的 Web 服务器应用程序正在此远程设备上运行。您必须已经在 AWS IoT 注册表中创建了物联网事物。有关更多信息，请参阅 如何使用注册表管理事物。

   ```
   aws iotsecuretunneling open-tunnel \
    --destination-config thingName=RemoteDevice1,services=HTTP
   ```

   运行此命令将生成源和目标访问令牌，您将使用这些令牌运行本地代理。

   ```
   {
    "tunnelId": "b2de92a3-b8ff-46c0-b0f2-afa28b00cecd",
    "tunnelArn": "arn:aws:iot:us-west-2:431600097591:tunnel/b2de92a3-b8ff-46c0-b0f2-
   afa28b00cecd",
    "sourceAccessToken": source_client_access_token,
    "destinationAccessToken": destination_client_access_token
   }
   ```

2. 配置和启动本地代理

   在运行本地代理之前，请从中下载本地代理源代码，GitHub然后针对您选择的平台进行构建。之后，您可以启动目标和源本地代理以连接到安全隧道，并开始使用远程 Web 服务器应用程序。

   > **Note**
   >
   > 要使用同步的 TCP 连接进行 AWS IoT 安全隧道，必须升级到最新版本的本地代理。如果您使用 AWS IoT 设备客户端配置本地代理，则此功能不可用。

   ```
   // Start the destination local proxy
   ./localproxy -r us-east-1 -d HTTP=80 -t destination_client_access_token

   // Start the source local proxy
   ./localproxy -r us-east-1 -s HTTP=5555 -t source_client_access_token
   ```

   有关配置和使用本地代理的更多信息，请参阅如何使用本地代理。

现在，您可以使用隧道访问 Web 服务器应用程序。 AWS IoT 当客户端有多个请求时，安全隧道将自动设置和处理同时的 TCP 连接。

配置远程设备和使用 IoT 代理

IoT 代理用于接收包含客户端访问令牌的 MQTT 消息，并在远程设备上启动本地代理。如果希望安全隧道使用 MQTT 传送客户端访问令牌，则必须在远程设备上安装并运行 IoT 代理。IoT 代理必须订阅以下保留的 IoT MQTT 主题：

> **Note**
>
> 如果要通过订阅保留 MQTT 主题之外的方法将目标客户端访问令牌传送到远程设备，您可能需要目标客户端访问令牌 (CAT) 侦听器和本地代理。CAT 侦听器必须使用您选择的客户端访问令牌传送机制，并且能够在目标模式下启动本地代理。

IoT 代理代码段

IoT 代理必须订阅以下保留 IoT MQTT 主题，这样才能接收 MQTT 消息并启动本地代理：

$aws/things/*thing-name*/tunnels/notify

thing-name与远程设备关联 AWS IoT 的事物的名称在哪里。

以下是 MQTT 消息负载示例：

```
{
    "clientAccessToken": "destination-client-access-token",
    "clientMode": "destination",
    "region": "aws-region",
    "services": ["destination-service"]
}
```

当您收到 MQTT 消息后，IoT 代理必须在远程设备上使用适当参数启动本地代理。

以下 Java 代码演示了如何使用AWS IoT 设备 SDK 和 ProcessBuilderJava 库来构建用于安全隧道的简单物联网代理。

```
// Find the IoT device endpoint for your AWS ##
final String endpoint = iotClient.describeEndpoint(new
 DescribeEndpointRequest().withEndpointType("iot:Data-ATS")).getEndpointAddress();

// Instantiate the IoT Agent with your AWS credentials
final String thingName = "RemoteDeviceA";
final String tunnelNotificationTopic = String.format("$aws/things/%s/tunnels/notify",
 thingName);
final AWSIotMqttClient mqttClient = new AWSIotMqttClient(endpoint, thingName,
                 "your_aws_access_key", "your_aws_secret_key");

try {
    mqttClient.connect();
    final TunnelNotificationListener listener = new
 TunnelNotificationListener(tunnelNotificationTopic);
    mqttClient.subscribe(listener, true);
}
finally {
    mqttClient.disconnect();
}

private static class TunnelNotificationListener extends AWSIotTopic {
    public TunnelNotificationListener(String topic) {
        super(topic);
    }
```

```
    @Override
    public void onMessage(AWSIotMessage message) {
        try {
            // Deserialize the MQTT message
            final JSONObject json = new JSONObject(message.getStringPayload());

            final String accessToken = json.getString("clientAccessToken");
            final String region = json.getString("region");

            final String clientMode = json.getString("clientMode");
            if (!clientMode.equals("destination")) {
                throw new RuntimeException("Client mode " + clientMode + " in the MQTT
 message is not expected");
            }

            final JSONArray servicesArray = json.getJSONArray("services");
            if (servicesArray.length() > 1) {
                throw new RuntimeException("Services in the MQTT message has more than
 1 service");
            }
            final String service = servicesArray.get(0).toString();
            if (!service.equals("SSH")) {
                throw new RuntimeException("Service " + service + " is not supported");
            }

            // Start the destination local proxy in a separate process to connect to
 the SSH Daemon listening port 22
            final ProcessBuilder pb = new ProcessBuilder("localproxy",
                        "-t", accessToken,
                        "-r", region,
                        "-d", "localhost:22");
            pb.start();
        }
        catch (Exception e) {
            log.error("Failed to start the local proxy", e);
        }
    }
}
```

控制对隧道的访问

安全隧道提供特定于服务的操作、资源和条件上下文键，以供在 IAM 权限策略中使用。

隧道访问先决条件

- 了解如何使用 IAM 策略保护 AWS 资源。
- 了解如何创建和评估 IAM 条件。
- 了解如何使用 AWS 资源标签保护资源。

隧道访问策略

您必须使用以下策略来授予使用安全隧道 API 的权限。有关 AWS IoT 安全性的更多信息，请参阅的身份和访问管理 AWS IoT。

物联网：OpenTunnel

该`iot:OpenTunnel`政策操作授予委托人拨打电话的权限OpenTunnel。

在 IAM policy 声明的 `Resource` 元素中：

- 指定通配符隧道 ARN：

  `arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:tunnel/*`
- 指定事物 ARN 来管理对特定 IoT 事物的 `OpenTunnel` 权限：

  `arn:aws:iot:`*`aws-region`*`:`*`aws-account-id`*`:thing/`*`thing-name`*

例如，以下策略语句允许您打开到名为 `TestDevice` 的 IoT 事物的隧道。

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

`iot:OpenTunnel` 策略操作支持以下条件键：

- `iot:ThingGroupArn`
- `iot:TunnelDestinationService`

- aws:RequestTag/####
- aws:SecureTransport
- aws:TagKeys

如果事物属于名称以 TestGroup 开头的事物组，并且隧道上配置的目标服务是 SSH，则以下策略语句允许您打开到该事物的隧道。

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "ForAnyValue:StringLike": {
            "iot:ThingGroupArn": [
                "arn:aws:iot:aws-region:aws-account-id:thinggroup/TestGroup*"
            ]
        },
        "ForAllValues:StringEquals": {
            "iot:TunnelDestinationService": [
                "SSH"
            ]
        }
    }
}
```

您还可以使用资源标签来控制打开隧道的权限。例如，如果存在值为 Admin 的标签键 Owner 并且未指定其他标签，则以下策略语句允许打开隧道。有关使用标签的一般信息，请参阅为资源添加 AWS IoT 标签。

```
{
    "Effect": "Allow",
    "Action": "iot:OpenTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/Owner": "Admin"
        },
```

```
        "ForAllValues:StringEquals": {
            "aws:TagKeys": "Owner"
        }
    }
}
```

物联网：RotateTunnelAccessToken

该iot:RotateTunnelAccessToken政策操作授予委托人拨打电话的权限RotateTunnelAccessToken。

在 IAM policy 声明的 Resource 元素中：

- 指定完全限定的隧道 ARN：

  arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

  您也可以使用通配符隧道 ARN：

  arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

- 指定事物 ARN 来管理对特定 IoT 事物的 RotateTunnelAccessToken 权限：

  arn:aws:iot:*aws-region*:*aws-account-id*:thing/*thing-name*

例如，以下策略语句允许您为名为 TestDevice 的 IoT 事物轮换隧道的源访问令牌或客户端的目标访问令牌。

```
{
    "Effect": "Allow",
    "Action": "iot:RotateTunnelAccessToken",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
}
```

iot:RotateTunnelAccessToken 策略操作支持以下条件键：

- iot:ThingGroupArn
- iot:TunnelDestinationService

- iot:ClientMode
- aws:SecureTransport

如果事物属于名称以 TestGroup 开头的事物组，隧道上配置的目标服务是 SSH，并且客户端处于 DESTINATION 模式，则以下策略语句允许您对此事物轮换目标访问令牌。

```
{
    "Effect": "Allow",
    "Action": "iot:RotateTunnelAccessToken",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "ForAnyValue:StringLike": {
            "iot:ThingGroupArn": [
                "arn:aws:iot:aws-region:aws-account-id:thinggroup/TestGroup*"
            ]
        },
        "ForAllValues:StringEquals": {
            "iot:TunnelDestinationService": [
                "SSH"
            ],
            "iot:ClientMode": "DESTINATION"
        }
    }
}
```

物联网：DescribeTunnel

该iot:DescribeTunnel政策操作授予委托人拨打电话的权限DescribeTunnel。

在 IAM policy 语句的 Resource 元素中，指定完全限定的隧道 ARN：

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

您也可以使用通配符 ARN：

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:DescribeTunnel 策略操作支持以下条件键：

- aws:ResourceTag/*tag-key*

- aws:SecureTransport

以下策略语句允许您在请求的隧道使用值为 Admin 的键 Owner 进行标记时调用 DescribeTunnel。

```
{
    "Effect": "Allow",
    "Action": "iot:DescribeTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/Owner": "Admin"
        }
    }
}
```

物联网：ListTunnels

该iot:ListTunnels政策操作授予委托人拨打电话的权限ListTunnels。

在 IAM policy 声明的 Resource 元素中：

- 指定通配符隧道 ARN：

  arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*
- 指定事物 ARN 来管理对所选 IoT 事物的 ListTunnels 权限：

  arn:aws:iot:*aws-region*:*aws-account-id*:thing/*thing-name*

iot:ListTunnels 策略操作支持条件键 aws:SecureTransport。

以下策略语句允许您列出名为 TestDevice 的事物的隧道。

```
{
    "Effect": "Allow",
    "Action": "iot:ListTunnels",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*",
        "arn:aws:iot:aws-region:aws-account-id:thing/TestDevice"
    ]
```

```
}
```

物联网：ListTagsForResource

iot:ListTagsForResource 策略操作授予调用 ListTagsForResource 的委托人权限。

在 IAM policy 语句的 Resource 元素中，指定完全限定的隧道 ARN：

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

您也可以使用通配符隧道 ARN：

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:ListTagsForResource 策略操作支持条件键 aws:SecureTransport。

物联网：CloseTunnel

该iot:CloseTunnel政策操作授予委托人拨打电话的权限CloseTunnel。

在 IAM policy 语句的 Resource 元素中，指定完全限定的隧道 ARN：

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

您也可以使用通配符隧道 ARN：

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:CloseTunnel 策略操作支持以下条件键：

- iot:Delete
- aws:ResourceTag/*tag-key*
- aws:SecureTransport

以下策略语句允许您在以下情况下调用 CloseTunnel：请求的 Delete 参数是 false，并且请求的隧道使用值为 QATeam 的键 Owner 进行标记。

```
{
    "Effect": "Allow",
    "Action": "iot:CloseTunnel",
    "Resource": [
        "arn:aws:iot:aws-region:aws-account-id:tunnel/*"
```

```
    ],
    "Condition": {
        "Bool": {
            "iot:Delete": "false"
        },
        "StringEquals": {
            "aws:ResourceTag/Owner": "QATeam"
        }
    }
}
```

物联网：TagResource

iot:TagResource 策略操作授予调用 TagResource 的委托人权限。

在 IAM policy 语句的 Resource 元素中，指定完全限定的隧道 ARN：

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

您也可以使用通配符隧道 ARN：

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:TagResource 策略操作支持条件键 aws:SecureTransport。

物联网：UntagResource

iot:UntagResource 策略操作授予调用 UntagResource 的委托人权限。

在 IAM policy 语句的 Resource 元素中，指定完全限定的隧道 ARN：

arn:aws:iot:*aws-region*: *aws-account-id*:tunnel/*tunnel-id*

您也可以使用通配符隧道 ARN：

arn:aws:iot:*aws-region*:*aws-account-id*:tunnel/*

iot:UntagResource 策略操作支持条件键 aws:SecureTransport。

通过轮换客户端访问令牌来解决 AWS IoT 安全隧道连接问题

使用 AWS IoT 安全隧道时，即使隧道已打开，也可能会遇到连接问题。以下各节显示了一些可能的问题，以及如何通过轮换客户端访问令牌来解决这些问题。要轮换客户端访问令牌 (CAT)，请使用

RotateTunnelAccessTokenAPI 或rotate-tunnel-access-token AWS CLI。根据是在源模式还是目标模式下使用客户端时遇到错误，您可以在源模式和/或目标模式下轮换 CAT。

> **Note**
>
> - 如果您不确定是否需要在源或目标上轮换 CAT，则可以在使用 `RotateTunnelAccessToken` API 时通过将 `ClientMode` 设置为 `ALL`，这样就能同时在源和目标上轮换 CAT。
> - 轮换 CAT 不会延长隧道持续时间。例如，假设隧道持续时间为 12 小时，隧道已经打开了 4 个小时。轮换访问令牌时，生成的新令牌只能在剩余的 8 小时内使用。

主题

客户端访问令牌错误无效

使用 AWS IoT 安全隧道时，使用相同的客户端访问令牌 (CAT) 重新连接到同一隧道时，可能会遇到连接错误。在这种情况下，本地代理无法连接到安全隧道代理服务器。如果您在源模式下使用客户端，您可能会看到以下错误消息：

```
Invalid access token: The access token was previously used and cannot be used again
```

出现此错误的原因是，本地代理只能使用一次客户端访问令牌 (CAT)，然后它变为无效。要纠正此错误，请在 `SOURCE` 模式下轮换客户端访问令牌以便为源生成新的 CAT。有关演示如何轮换源 CAT 的示例，请参阅轮换源 CAT 示例。

客户端令牌不匹配错误

> **Note**
>
> 不建议通过客户端令牌来重复使用 CAT。我们建议您改用 `RotateTunnelAccessToken` API 来轮换客户端访问令牌，以重新连接到隧道。

如果您使用的是客户端令牌，则可以重复使用 CAT 以重新连接到隧道。要重复使用 CAT，您必须在首次连接到安全隧道时向 CAT 提供客户端令牌。安全隧道存储客户端令牌，因此，对于使用相同令牌的后续连接尝试，还必须提供客户端令牌。有关使用客户端令牌的更多信息，请参阅中的本地代理参考实现 GitHub。

使用客户端令牌时，如果在源模式下使用客户端，则可能会看到以下错误：

```
Invalid client token: The provided client token does not match the client token
    that was previously set.
```

出现此错误的原因是提供的客户端令牌与访问隧道时随 CAT 提供的客户端令牌不匹配。要纠正此错误，请在 SOURCE 模式下轮换 CAT 以便为源生成新的 CAT。下面是一个示例：

轮换源 CAT 示例

以下示例显示如何在 SOURCE 模式下运行 RotateTunnelAccessToken API，以便为源代码生成新的 CAT：

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --region <region> \
    --tunnel-id <tunnel-id> \
    --client-mode SOURCE
```

运行此命令将生成一个新的源访问令牌并返回隧道的 ARN。

```
{
    "sourceAccessToken": "<source-access-token>",
    "tunnelArn": "arn:aws:iot:<region>:<account-id>:tunnel/<tunnel-id>"
}
```

您现在可以使用新的源令牌在源模式下连接本地代理。

```
export AWSIOT_TUNNEL_ACCESS_TOKEN=<source-access-token>
./localproxy -r <region> -s <port>
```

下面的内容显示运行本地代理的示例输出。

```
...

[info]    Starting proxy in source mode
```

```
...
[info]    Successfully established websocket connection with proxy server ...
[info]    Listening for new connection on port <port>
...
```

远程设备连接问题

使用 AWS IoT 安全隧道时，即使隧道已打开，设备也可能会意外断开连接。要确定设备是否仍连接到隧道，您可以使用 DescribeTunnelAPI 或 desc ribe AWS CLI-tunnel。

设备可能由于多种原因而断开连接。要解决连接问题，如果设备由于以下可能的原因断开连接，则可以在目标上轮换 CAT：

- 目标上的 CAT 变为无效。
- 令牌未通过安全隧道保留的 MQTT 主题传送到设备：

  $aws/things/*<thing-name>*/tunnels/notify

下面的示例演示了如何解决此问题：

轮换目标 CAT 示例

考虑远程设备 *<RemoteThing1>*。要为该事物打开隧道，您可以使用以下命令：

```
aws iotsecuretunneling open-tunnel \
    --region <region> \
    --destination-config thingName=<RemoteThing1>,services=SSH
```

运行此命令将为源和目标生成隧道详细信息和 CAT。

```
{
    "sourceAccessToken": "<source-access-token>",
    "destinationAccessToken": "<destination-access-token>",
    "tunnelId": "<tunnel-id>",
    "tunnelArn": "arn:aws:iot:<region>:<account-id>:tunnel/tunnel-id"
}
```

但是，当您使用 DescribeTunnelAPI 时，输出表明设备已断开连接，如下图所示：

```
aws iotsecuretunneling describe-tunnel \
```

```
    --tunnel-id <tunnel-id> \
    --region <region>
```

运行此命令将显示设备仍未连接。

```
{
    "tunnel": {
        ...
        "destinationConnectionState": {
            "status": "DISCONNECTED"
        },
        ...
    }
}
```

要解决此错误，请在客户端处于 DESTINATION 模式下运行 RotateTunnelAccessToken API，并为目标运行配置。运行此命令将撤销旧的访问令牌，生成新令牌，并将此令牌重新发送到 MQTT 主题：

$aws/things/*<thing-name>*/tunnels/notify

```
aws iotsecuretunneling rotate-tunnel-access-token \
    --tunnel-id <tunnel-id> \
    --client-mode DESTINATION \
    --destination-config thingName=<RemoteThing1>,services=SSH \
    --region <region>
```

运行此命令将生成新的访问令牌，如下所示。然后，如果设备代理设置正确，令牌将传送到设备以连接到隧道。

```
{
    "destinationAccessToken": "destination-access-token",
    "tunnelArn": "arn:aws:iot:region:account-id:tunnel/tunnel-id"
}
```

设备预调配

AWS 提供了几种不同的方法来配置设备并在其上安装唯一的客户端证书。本部分介绍各种方式，以及如何选择适合您的 IoT 解决方案的最佳方法。这些选项在标题为在 AWS IoT Core中使用 X.509 证书进行设备制造和预调配的白皮书中进行了详细介绍。

选择最适合您情况的选项

- 您可以在 IoT 设备上安装证书，然后再予交付

  如果您可以在 IoT 设备上安全地安装唯一的客户端证书以供最终用户使用，则需要使用即时预调配 (JITP) 或者即时注册 (JITR)。

  使用 JITP 和 JITR，用于签署设备证书的证书颁发机构 (CA) 在设备首次连接 AWS IoT 时注册 AWS IoT 并被识别。设备将在首次连接时使用其配置模板的详细信息进行配置。 AWS IoT

  有关单件事物、JITP、JITR 和具有唯一证书的设备批量预调配的详细信息，请参阅 the section called “预调配具有设备证书的设备”。
- 终端用户或安装人员可以使用应用程序在其 IoT 设备上安装证书

  如果您无法在 IoT 设备上安全地安装唯一的客户端证书以供终端用户使用，但终端用户或安装程序可以使用应用程序注册设备并安装唯一设备证书，则需要执行通过信任用户预调配的过程。

  使用信任用户（如终端用户或具有已知账户的安装程序）可以简化设备制造过程。设备没有唯一的客户端证书，而是临时证书，仅允许设备连接 5 分钟。 AWS IoT 在这 5 分钟的窗口中，信任用户将获得具有更长寿命的唯一客户端证书，并将其安装在设备上。申请证书的寿命有限，因此能够最大限度地减少证书受损的风险。

  有关更多信息，请参阅 the section called “由可信用户预调配”。
- 终端用户无法使用应用程序在其 IoT 设备上安装证书

  如果上述两个选项都不适用于您的 IoT 解决方案，则可选择通过申请预调配流程。通过此流程，您的 IoT 设备将拥有由实例集中其它设备共享的申请证书。设备首次使用声明证书连接时，使用其配置模板 AWS IoT 注册该设备，并向设备颁发其唯一的客户端证书以供后续访问 AWS IoT。

  此选项允许在设备连接时自动配置设备 AWS IoT，但如果索赔证书泄露，则可能会带来更大的风险。如果申请证书受损，您可以停用该证书。停用申请证书可防止具有该申请证书的所有设备在将来不会被注册。但是，停用申请证书不会阻止已预调配的设备。

有关更多信息，请参阅 the section called “通过申请进行预调配”。

在 AWS IoT中预调配设备

使用配置设备时 AWS IoT，必须创建资源，这样您的设备 AWS IoT 才能安全地通信。您可以创建其它资源来帮助管理设备实例集。在预调配过程中可以创建以下资源：

- IoT 事物。

  物联网事物是 AWS IoT 设备注册表中的条目。每个事物都有一个唯一的名称和一组属性，并与物理设备相关联。事物可以使用事物类型来定义，或者分组为事物组。有关更多信息，请参阅 使用管理设备 AWS IoT。

  虽然创建事物并非必需，但这使您能够按事物类型、事物组和事物属性来搜索设备，以更有效地管理设备实例集。有关更多信息，请参阅 机群索引。

  > **Note**
  >
  > 要索引事物的连接状态数据，请配置您的事物并对其进行配置，使事物名称与 Connect 请求中使用的客户端 ID 相匹配。

- X.509 证书。

  设备使用 X.509 证书与执行相互身份验证。 AWS IoT您可以注册现有证书，也可以为您 AWS IoT 生成和注册新证书。通过将证书附加到表示设备的事物，您可以将证书与设备关联。您还必须将证书和关联的私有密钥复制到设备上。设备在连接时会出示证书 AWS IoT。有关更多信息，请参阅 身份验证。

- IoT 策略。

  IoT 策略确定设备可在 AWS IoT中执行的操作。IoT 策略附加到设备证书。当设备向其提供证书时 AWS IoT，它将被授予策略中指定的权限。有关更多信息，请参阅 授权。每个设备都需要一个证书来与 AWS IoT通信。

AWS IoT 支持使用配置模板自动配置队列。配置模板描述了配置设备 AWS IoT 所需的资源。模板包含变量，使您能够使用一个模板来预调配多个设备。在您预调配设备时，您可以使用字典或映射 为设备特定的变量指定值。要预调配其它设备，请在字典中指定新值。

无论设备是否具有唯一的证书（及其关联的私有密钥），您都可以使用自动预调配。

实例集预调配 API

实例集预调配中使用的 API 分为几类：

- 这些控制面板函数可创建和管理实例集预调配模板，并配置可信用户策略。
  - CreateProvisioning模板
  - CreateProvisioningTemplateVersion
  - DeleteProvisioning模板
  - DeleteProvisioningTemplateVersion
  - DescribeProvisioning模板
  - DescribeProvisioningTemplateVersion
  - ListProvisioning模板
  - ListProvisioningTemplateVersions
  - UpdateProvisioning模板
- 可信用户可以使用此控制面板函数生成临时信息载入申请。此临时申请在 Wi-Fi 配置或类似方法期间传递给设备。
  - CreateProvisioning索赔
- 预调配过程中设备使用的 MQTT API，其中包含嵌入设备中的预调配申请证书，或由可信用户传递到其中。
  - the section called “CreateCertificateFromCsr”
  - the section called “CreateKeysAndCertificate”
  - the section called “RegisterThing”

使用实例集预调配来预调配没有设备证书的设备

通过使用 AWS IoT 队列配置， AWS IoT 可以在设备首次连接时生成设备证书和私钥并将其安全地交付 AWS IoT 给您的设备。 AWS IoT 提供由 Amazon 根证书颁发机构 (CA) 签署的客户证书。

使用实例集预调配的方法有两种：

- 通过申请进行预调配
- 由可信用户预调配

通过申请进行预调配

设备在制造时可以嵌入预调配申请证书和私有密钥（这是特殊用途凭证）。如果注册了这些证书 AWS IoT，则该服务可以将它们交换为设备可用于常规操作的唯一设备证书。此过程包括以下步骤：

在交付设备之前

1. 调用 CreateProvisioningTemplate 以创建预调配模板。此 API 返回模板 ARN。有关更多信息，请参阅 设备预调配 MQTT API。

   您也可以在 AWS IoT 控制台中创建队列配置模板。

   a. 从导航窗格中选择 Connect（连接），然后选择 Fleet provisioning templates（实例集预调配模板）。

   b. 选择 Create template（创建模板），再按照提示操作。

2. 创建用作预调配申请证书的证书以及关联的私有密钥。

3. 将这些证书注册到限制证书使用的 IoT 策略 AWS IoT 并关联该策略。以下示例 IoT 策略限制使用与此策略关联的证书来预调配设备。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["iot:Connect"],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": ["iot:Publish","iot:Receive"],
            "Resource": [
                "arn:aws:iot:aws-region:aws-account-id:topic/$aws/certificates/
create/*",
                "arn:aws:iot:aws-region:aws-account-id:topic/$aws/provisioning-
templates/templateName/provision/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
```

```
                "arn:aws:iot:aws-region:aws-account-id:topicfilter/$aws/
certificates/create/*",
                "arn:aws:iot:aws-region:aws-account-id:topicfilter/$aws/
provisioning-templates/templateName/provision/*"
            ]
        }
    ]
}
```

4. 在配置设备时，授予 AWS IoT 服务权限以创建或更新您账户中的物联网资源和证书。为此，请将AWSIoTThingsRegistration托管策略附加到信任 AWS IoT 服务委托人的 IAM 角色（称为配置角色）。
5. 制造在其中安全地嵌入了预调配申请证书的设备。

设备现在已准备就绪，可以运输到要安装以使用它的位置。

> **Important**
>
> 预调配申请私有密钥应始终得到保护，包括在设备上时。我们建议您使用 AWS IoT CloudWatch 指标和日志来监控是否存在滥用迹象。如果您检测到滥用，请关闭预调配申请证书，以使其不能用于设备预调配。

初始化设备以供使用

1. 设备AWS IoT 设备 SDK、移动 SDK 和 AWS IoT 设备客户端使用连接设备上安装的 AWS IoT 配置声明证书并进行身份验证。

> **Note**
>
> 为了安全起见，由 CreateCertificateFromCsr 和 CreateKeysAndCertificate 返回的 certificateOwnershipToken 将在一小时后过期。必须在 certificateOwnershipToken 过期前调用 RegisterThing。如果由 CreateCertificateFromCsr 或 CreateKeysAndCertificate 创建的证书尚未激活，并且在令牌过期时尚未附加到策略或事物，则该证书将被删除。如果令牌过期，设备可以再次调用 CreateCertificateFromCsr 或 CreateKeysAndCertificate 以生成新证书。

2. 设备通过使用以下选项之一获取永久证书和私有密钥。设备将使用证书和密钥进行 future 的所有身份验证 AWS IoT。

   a. 调用CreateKeysAndCertificate以使用证书颁发机构创建新的 AWS 证书和私钥。

      Or

   b. 调用 CreateCertificateFromCsr 以通过证书签名请求生成证书，这可确保其私有密钥安全。

3. 从设备中，调用 RegisterThing 以向 AWS IoT 注册设备并创建云资源。

   实例集预调配服务使用预调配模板来定义和创建云资源，如 IoT 事物。该模板可以指定事物的属性和所属的组。在将新事物添加到事物组之前，事物组必须先存在。

4. 在设备上保存永久证书后，设备必须断开与它使用预调配申请证书发起的会话的连接，并使用此永久证书重新连接。

设备现已准备就绪，可以与之正常通信 AWS IoT。

由可信用户预调配

在许多情况下，当可信用户（例如最终用户或安装技术人员）使用移动应用程序在其部署位置配置设备时，设备才会首次连接到该设备。 AWS IoT

> **Important**
>
> 您必须管理可信用户的访问和权限才能执行此流程。做到这一点的一种方法是为可信用户提供和维护一个账户，该账户对他们进行身份验证，并授予他们对执行此过程所需的 AWS IoT 特征和 API 操作的访问权限。

在交付设备之前

1. 调用 CreateProvisioningTemplate 以创建预调配模板并返回其 *templateArn* 和 *templateName*。
2. 创建由信任用户用于启动预调配过程的 IAM 角色。预调配模板仅允许该用户预调配设备。例如：

```
{
    "Effect": "Allow",
    "Action": [
```

```
            "iot:CreateProvisioningClaim"
        ],
        "Resource": [
            "arn:aws:iot:aws-region:aws-account-id:provisioningtemplate/templateName"
        ]
}
```

3. 在配置设备时，向 AWS IoT 服务授予创建或更新物联网资源的权限，例如您账户中的东西和证书。为此，您可以将AWSIoTThingsRegistration托管策略附加到信任 AWS IoT 服务委托人的 IAM 角色（称为配置角色）。
4. 提供识别可信用户的方式，例如为他们提供一个账户，该账户可以对他们进行身份验证，并授权他们与注册其设备所需 AWS 的 API 操作进行交互。

初始化设备以供使用

1. 可信用户登录您的预调配移动应用程序或 Web 服务。
2. 移动应用程序或 Web 应用程序使用 IAM 角色并调用 CreateProvisioningClaim，以从 AWS IoT获取临时预调配申请证书。

> **Note**
>
> 出于安全考虑，CreateProvisioningClaim 返回的临时预调配申请证书只有五分钟的有效期。在临时预调配申请证书过期之前，以下步骤必须成功地返回有效证书。临时预调配申请证书不会显示在您账户的证书列表中。

3. 移动应用程序或 Web 应用程序将临时预调配申请证书以及任何必需的配置信息（如 Wi-Fi 凭证）提供给设备。
4. 设备使用临时预调配申请证书，通过 AWS IoT 设备 SDK、移动 SDK 和 AWS IoT 设备客户端 连接到 AWS IoT 。
5. 设备在使用临时配置声明证书连接后的五分钟内使用其中一个选项获得永久证书和私钥。 AWS IoT 设备将使用这些选项返回的证书和密钥进行未来的所有身份验证 AWS IoT。

   a. 调用CreateKeysAndCertificate以使用证书颁发机构创建新的 AWS 证书和私钥。

      Or

   b. 调用 CreateCertificateFromCsr 以通过证书签名请求生成证书，这可确保其私有密钥安全。

> **Note**
>
> 请记住CreateKeysAndCertificate或CreateCertificateFromCsr必须在连接到 AWS IoT 临时配置声明证书后的五分钟内返回有效的证书。

6. 设备调RegisterThing用注册设备 AWS IoT 并创建云资源。

   实例集预调配服务使用预调配模板来定义和创建云资源，如 IoT 事物。该模板可以指定事物的属性和所属的组。在将新事物添加到事物组之前，事物组必须先存在。

7. 在设备上保存永久证书后，设备必须断开与它使用临时预调配申请证书发起的会话的连接，并使用此永久证书重新连接。

设备现已准备就绪，可以与之正常通信 AWS IoT。

在 AWS CLI 中使用预先预调配挂钩

以下流程使用预先预调配挂钩创建预调配模板。此处使用的 Lambda 函数是一个可以修改的示例。

创建预先预调配挂钩并将其应用到预调配模板

1. 创建一个具有定义输入和输出的 Lambda 函数。Lambda 函数高度可自定义，需要 `allowProvisioning` 和 `parameterOverrides` 来创建预调配挂钩。有关创建 Lambda 函数的更多信息，请参阅在 AWS 命令行界 AWS Lambda 面中使用。

   以下是 Lambda 函数输出的示例：

   ```
   {
     "allowProvisioning": True,
     "parameterOverrides": {
       "incomingKey0": "incomingValue0",
       "incomingKey1": "incomingValue1"
     }
   }
   ```

2. AWS IoT 使用基于资源的策略来调用 Lambda，因此您必须授 AWS IoT 予调用 Lambda 函数的权限。

> **Important**
>
> 请务必在附加到 Lambda 操作策略的全局条件上下文键中包含 `source-arn` 或 `source-account`，以便防范权限操纵。有关此问题的更多信息，请参阅 防止跨服务混淆座席。

以下示例使用 add-permission 将 IoT 权限授予您的 Lambda。

```
aws lambda add-permission \
    --function-name myLambdaFunction \
    --statement-id iot-permission \
    --action lambda:InvokeFunction \
    --principal iot.amazonaws.com
```

3. 使用 create-provisioning-template 或 update-provisioning-template 命令添加预先预调配挂钩到模板。

   以下 CLI 示例使用 create-provisioning-template 创建具有预先预调配挂钩的预调配模板：

   ```
   aws iot create-provisioning-template \
       --template-name myTemplate \
       --provisioning-role-arn arn:aws:iam:us-east-1:1234564789012:role/myRole \
       --template-body file://template.json \
       --pre-provisioning-hook file://hooks.json
   ```

   此命令的输出如下所示：

   ```
   {
       "templateArn": "arn:aws:iot:us-east-1:1234564789012:provisioningtemplate/
   myTemplate",
       "defaultVersionId": 1,
       "templateName": myTemplate
   }
   ```

   您也可以从文件加载参数，而不是将其作为命令行参数值完全键入，以节省时间。有关更多信息，请参阅从文件加载 AWS CLI 参数。下面显示了扩展 JSON 格式的 `template` 参数：

   ```
   {
       "Parameters" : {
   ```

```
        "DeviceLocation": {
            "Type": "String"
        }
    },
    "Mappings": {
        "LocationTable": {
            "Seattle": {
                "LocationUrl": "https://example.aws"
            }
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "AttributePayload" : {
                    "version" : "v1",
                    "serialNumber" : "serialNumber"
                },
                "ThingName" : {"Fn::Join":["",["ThingPrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingTypeName" : {"Fn::Join":["",["ThingTypePrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingGroups" : ["widgets", "WA"],
                "BillingGroup": "BillingGroup"
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateId": {"Ref": "AWS::IoT::Certificate::Id"},
                "Status" : "Active"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : {
                    "Version": "2012-10-17",
```

```
                    "Statement": [{
                        "Effect": "Allow",
                        "Action":["iot:Publish"],
                        "Resource": ["arn:aws:iot:us-east-1:504350838278:topic/foo/
bar"]
                    }]
                }
            }
        }
    },
    "DeviceConfiguration": {
        "FallbackUrl": "https://www.example.com/test-site",
        "LocationUrl": {
            "Fn::FindInMap": ["LocationTable",{"Ref": "DeviceLocation"},
 "LocationUrl"]}
    }
}
```

下面显示了扩展 JSON 格式的 pre-provisioning-hook 参数：

```
{
     "targetArn" : "arn:aws:lambda:us-
east-1:765219403047:function:pre_provisioning_test",
     "payloadVersion" : "2020-04-01"
}
```

预调配具有设备证书的设备

AWS IoT 当设备上已有设备证书（和关联的私钥）时，提供了三种配置设备的方式：

- 使用预调配模板进行单一事物预调配。如果您只需一次预调配一台设备，那么这是一个很好的选择。
- J ust-in-time 配置 (JITP)，其模板可在设备首次连接时对其进行 AWS IoT配置。如果您需要注册大量设备，但没有可以整合到批量预调配列表中的有关这些设备的信息，那么这是一个很好的选择。
- 批量注册。此选项可让您指定存储在 S3 存储桶内的文件中的单一事物预调配模板值。如果您有大量已知设备，并且可以将它们所需的特性整合到列表中，则此方法很有用。

主题

- 单个事物预调配

- J ust-in-time 资源调配
- 批量注册

单个事物预调配

要配置事物，请使用 RegisterThingAPI 或 `register-thing` CLI 命令。`register-thing` CLI 命令接受以下参数：

--template-body

预调配模板。

--parameters

在预调配模板中使用的参数的名称-值对列表，采用 JSON 格式（例如，`{"ThingName" : "MyProvisionedThing", "CSR" : "csr-text"}`）。

请参阅 预调配模板。

RegisterThing或者`register-thing`返回资源的 ARN 及其创建的证书的文本：

```
{
    "certificatePem": "certificate-text",
    "resourceArns": {
    "PolicyLogicalName": "arn:aws:iot:us-
west-2:123456789012:policy/2A6577675B7CD1823E271C7AAD8184F44630FFD7",
    "certificate": "arn:aws:iot:us-west-2:123456789012:cert/
cd82bb924d4c6ccbb14986dcb4f40f30d892cc6b3ce7ad5008ed6542eea2b049",
    "thing": "arn:aws:iot:us-west-2:123456789012:thing/MyProvisionedThing"
    }
}
```

如果在字典中省略了参数，则使用默认值。如果未指定默认值，则不使用值来替换参数。

J ust-in-time 资源调配

当您的设备首次尝试连接时，您可以使用 just-in-time 配置 (JITP) 对其进行配置。 AWS IoT要预调配设备，您必须启用自动注册，并将预调配模板与用于对设备证书进行签名的 CA 证书关联。配置成功和错误的记录与设备预配置指标在 Amazon 中相同 CloudWatch。

主题

JITP 概览

当设备尝试使用 AWS IoT 由已注册的 CA 证书签名的证书进行连接时，会从 CA 证书 AWS IoT 加载模板并使用它进行调用RegisterThing。JITP 工作流首先将注册一个状态值为 PENDING_ACTIVATION 的证书。当设备预调配流程完成时，该证书的状态将更改为 ACTIVE。

AWS IoT 定义了您可以在置备模板中声明和引用的以下参数：

- `AWS::IoT::Certificate::Country`
- `AWS::IoT::Certificate::Organization`
- `AWS::IoT::Certificate::OrganizationalUnit`
- `AWS::IoT::Certificate::DistinguishedNameQualifier`
- `AWS::IoT::Certificate::StateName`
- `AWS::IoT::Certificate::CommonName`
- `AWS::IoT::Certificate::SerialNumber`
- `AWS::IoT::Certificate::Id`

这些预调配模板参数的值被限制为 JITP 可从正在预调配的设备的证书的使用者字段中提取的内容。证书必须包含模板正文中所有参数的值。`AWS::IoT::Certificate::Id` 参数指内部生成的 ID，而不是证书包含的 ID。您可以使用 AWS IoT 规则中的`principal()`函数获取此 ID 的值。

> **Note**
>
> 您可以使用 AWS IoT Core just-in-time 配置 (JITP) 功能配置设备，而不必在设备首次连接时将整个信任链发送到 AWS IoT Core。可以选择出示 CA 证书，但当设备连接到 AWS IoT Core 时，需要发送服务器名称指示（SNI）扩展。

示例模板正文

以下 JSON 文件是完整 JITP 模板的示例模板正文。

```
{
   "Parameters":{
      "AWS::IoT::Certificate::CommonName":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::SerialNumber":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Country":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Id":{
         "Type":"String"
      }
   },
   "Resources":{
      "thing":{
         "Type":"AWS::IoT::Thing",
         "Properties":{
            "ThingName":{
               "Ref":"AWS::IoT::Certificate::CommonName"
            },
            "AttributePayload":{
               "version":"v1",
               "serialNumber":{
                  "Ref":"AWS::IoT::Certificate::SerialNumber"
               }
            },
            "ThingTypeName":"lightBulb-versionA",
            "ThingGroups":[
               "v1-lightbulbs",
               {
                  "Ref":"AWS::IoT::Certificate::Country"
               }
            ]
         },
         "OverrideSettings":{
            "AttributePayload":"MERGE",
            "ThingTypeName":"REPLACE",
            "ThingGroups":"DO_NOTHING"
         }
      },
      "certificate":{
```

```
         "Type":"AWS::IoT::Certificate",
         "Properties":{
            "CertificateId":{
               "Ref":"AWS::IoT::Certificate::Id"
            },
            "Status":"ACTIVE"
         }
      },
      "policy":{
         "Type":"AWS::IoT::Policy",
         "Properties":{
            "PolicyDocument":"{ \"Version\": \"2012-10-17\", \"Statement\": [{ \"Effect
\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\": [\"arn:aws:iot:us-
east-1:123456789012:topic/foo/bar\"] }] }"
         }
      }
   }
}
```

此示例模板声明了从证书中提取并在 Resources 部分中使用的 AWS::IoT::Certificate::CommonName、AWS::IoT::Certificate::SerialNumber、AWS::IoT::Certificate::Country 和 AWS::IoT::Certificate::Id 预调配参数的值。JITP 工作流随后将使用此模板执行以下操作：

- 注册一个证书并将其状态设置为 PENDING_ACTIVE。
- 创建一个事物资源。
- 创建一个策略资源。
- 将策略附加到证书。
- 将证书附加到事物。
- 将证书状态更新为 ACTIVE。

如果证书不具备Parameters部分中提及的所有属性，则设备配置将失败templateBody。例如，如果 AWS::IoT::Certificate::Country 包含在模板中，但证书没有 Country 属性，设备预调配操作将会失败。

您还可以使用 CloudTrail 对 JITP 模板的问题进行故障排除。有关 Amazon 中记录的指标的信息 CloudWatch，请参阅设备预配置指标。有关预调配模板的更多信息，请参阅预调配模板。

> **Note**
>
> 在配置过程中，just-in-time 配置 (JITP) 会调用其他 AWS IoT 控制平面 API 操作。这些调用可能会超过为您账户设置的 AWS IoT 节流配额，导致调用受到限制。如有必要，请联系 AWS 客户支持来增大节流配额。

使用预调配模板注册 CA

要使用完整的预调配模板注册 CA，请执行以下步骤：

1. 将预调配模板和角色 ARN 信息（如下例所示）保存为 JSON 文件：

```
{
     "templateBody" : "{\r\n    \"Parameters\" : {\r\n
 \"AWS::IoT::Certificate::CommonName\": {\r\n           \"Type\": \"String\"\r
\n        },\r\n        \"AWS::IoT::Certificate::SerialNumber\": {\r\n
  \"Type\": \"String\"\r\n        },\r\n        \"AWS::IoT::Certificate::Country
\": {\r\n            \"Type\": \"String\"\r\n        },\r\n
 \"AWS::IoT::Certificate::Id\": {\r\n            \"Type\": \"String\"\r
\n        }\r\n    },\r\n    \"Resources\": {\r\n        \"thing\": {\r
\n            \"Type\": \"AWS::IoT::Thing\",\r\n            \"Properties
\": {\r\n                \"ThingName\": {\r\n                    \"Ref\":
 \"AWS::IoT::Certificate::CommonName\"\r\n                },\r\n
     \"AttributePayload\": {\r\n                    \"version\": \"v1\",\r\n
                  \"serialNumber\": {\r\n                        \"Ref\":
 \"AWS::IoT::Certificate::SerialNumber\"\r\n                    }\r\n
     },\r\n                \"ThingTypeName\": \"lightBulb-versionA\",\r\n
        \"ThingGroups\": [\r\n                    \"v1-lightbulbs\",\r\n
           {\r\n                        \"Ref\": \"AWS::IoT::Certificate::Country
\"\r\n                    }\r\n                ]\r\n            },\r\n
  \"OverrideSettings\": {\r\n                \"AttributePayload\": \"MERGE\",\r\n
               \"ThingTypeName\": \"REPLACE\",\r\n                \"ThingGroups
\": \"DO_NOTHING\"\r\n            }\r\n        },\r\n        \"certificate\": {\r
\n            \"Type\": \"AWS::IoT::Certificate\",\r\n            \"Properties
\": {\r\n                \"CertificateId\": {\r\n                    \"Ref\":
 \"AWS::IoT::Certificate::Id\"\r\n                },\r\n                \"Status\":
 \"ACTIVE\"\r\n            },\r\n            \"OverrideSettings\": {\r\n
     \"Status\": \"DO_NOTHING\"\r\n            }\r\n        },\r\n        \"policy
\": {\r\n            \"Type\": \"AWS::IoT::Policy\",\r\n            \"Properties
\": {\r\n                \"PolicyDocument\": \"{ \\\"Version\\\": \\\"2012-10-17\
\\", \\\"Statement\\\": [{ \\\"Effect\\\": \\\"Allow\\\", \\\"Action\\\":[\\
```

```
\"iot:Publish\\\"], \\\"Resource\\\": [\\\"arn:aws:iot:us-east-1:123456789012:topic
\/foo\/bar\\\"] }] }\"\r\n            }\r\n        }\r\n    }\r\n}",
     "roleArn" : "arn:aws:iam::123456789012:role/JITPRole"
}
```

在本例中，templateBody 字段的值必须是指定为转义字符串的 JSON 对象，并且只能使用前述表中的值。您可以使用各种工具来创建所需的 JSON 输出，例如 json.dumps (Python) 或 JSON.stringify（节点）。roleARN 字段的值必须是附加有 AWSIoTThingsRegistration 的角色的 ARN。此外，您的模板还可以在示例中使用现有PolicyName，而不是内联 PolicyDocument。

2. 使用 RegisterCACertificate API 操作或 register-ca-certificate CLI 命令注册 CA 证书。您将指定预调配模板的目录以及您在上一步中保存的角色 ARN 信息：

   下面显示了如何使用 AWS CLI在 DEFAULT 模式中注册 CA 证书的示例：

   ```
   aws iot register-ca-certificate --ca-certificate file://your-ca-cert --
   verification-cert file://your-verification-cert
                  --set-as-active --allow-auto-registration --registration-config
    file://your-template
   ```

   下面显示了如何使用 SNI_ONLY 在 AWS CLI模式中注册 CA 证书的示例：

   ```
   aws iot register-ca-certificate --ca-certificate file://your-ca-cert --certificate-
   mode SNI_ONLY
                   --set-as-active --allow-auto-registration --registration-config
    file://your-template
   ```

   有关更多信息，请参阅注册 CA 证书。

3. （可选）使用 UpdateCACertificate API 操作或 update-ca-certificate CLI 命令更新 CA 证书的设置。

   下面显示了如何使用 AWS CLI更新 CA 证书的示例：

   ```
   aws iot update-ca-certificate --certificate-id caCertificateId
                  --new-auto-registration-status ENABLE --registration-config
    file://your-template
   ```

使用预调配模板名称注册 CA

要使用预调配模板名称注册 CA，请执行以下步骤：

1. 将预调配模板正文保存为 JSON 文件。您可以在示例模板正文中找到一个示例模板正文。
2. 要创建配置模板，请使用CreateProvisioning模板 API 或 `create-provisioning-template`CLI 命令：

```
aws iot create-provisioning-template --template-name your-template-name \
        --template-body file://your-template-body.json --type JITP \
        --provisioning-role-arn arn:aws:iam::123456789012:role/test
```

> **Note**
>
> 对于 just-in-time 配置 (JITP)，您必须在创建置备模板JITP时指定模板类型。有关模板类型的更多信息，请参阅 AWS API 参考中的CreateProvisioning模板。

3. 要使用模板名称注册 CA，请使用 RegisterCACertificate API 或`register-ca-certificate` CLI 命令：

```
aws iot register-ca-certificate --ca-certificate file://your-ca-cert --
verification-cert file://your-verification-cert \
        --set-as-active --allow-auto-registration --registration-config
 templateName=your-template-name
```

批量注册

您可以使用 `start-thing-registration-task` 命令批量注册事物。此命令接受预调配模板、S3 桶名称、键名称以及允许访问 S3 桶中文件的角色 ARN。S3 存储桶中的文件包含用于替换模板中参数的值。该文件必须为以换行符分隔的 JSON 文件。每一行包含用于注册单个设备的所有参数值。例如：

```
{"ThingName": "foo", "SerialNumber": "123", "CSR": "csr1"}
{"ThingName": "bar", "SerialNumber": "456", "CSR": "csr2"}
```

以下批量注册相关的 API 操作可能很有用：

- ListThingRegistrationTasks：列出当前的批量事物配置任务。

- DescribeThingRegistrationTask：提供有关特定批量事物注册任务的信息。
- StopThingRegistrationTask: 停止批量事物注册任务。
- ListThingRegistrationTask报告：用于检查批量事物注册任务的结果和失败。

> **Note**
>
> - 一次只能运行一个批量注册操作任务（每个账户）。
> - 批量注册操作会调用其他 AWS IoT 控制平面 API 操作。这些调用可能会超出您账户的 AWS IoT 节流配额，导致节流错误。如有必AWS 要，请联系 Customer Supp ort 以提高您的限 AWS IoT 流配额。

预调配模板

配置模板是一个 JSON 文档，它使用参数来描述您的设备必须使用哪些资源才能与之交互 AWS IoT。预调配模板包含两个部分：Parameters 和 Resources。中有两种类型的配置模板 AWS IoT。一个用于 just-in-time 配置 (JITP) 和批量注册，第二个用于队列配置。

主题

- 参数部分
- 资源部分
- 批量注册的模板示例
- just-in-time配置模板示例 (JITP)
- 实例集预调配

参数部分

Parameters 部分声明在 Resources 部分中使用的参数。每个参数声明一个名称、一个类型以及一个可选的默认值。在随模板传入的字典不包含参数的值时，会使用默认值。模板文档的 Parameters 部分类似于以下所示：

```
{
    "Parameters" : {
        "ThingName" : {
```

```
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
        },
        "CSR" : {
            "Type" : "String"
        }
    }
}
```

此模板正文片段声明四个参数：ThingName、SerialNumber、Location 和 CSR。所有这些参数均为 String 类型。Location 参数声明了默认值 "WA"。

资源部分

模板正Resources文的部分声明了您的设备与之通信所需的资源 AWS IoT：事物、证书以及一个或多个 IoT 策略。每个资源指定一个逻辑名称、一个类型和一组属性。

您可以使用逻辑名称在模板的其它位置引用资源。

类型指定您所声明的资源的种类。有效类型为：

- AWS::IoT::Thing
- AWS::IoT::Certificate
- AWS::IoT::Policy

您指定的属性取决于所声明的资源的类型。

事物资源

事物资源使用以下属性进行声明：

- ThingName: 字符串。
- AttributePayload：可选。名称-值对的列表。
- ThingTypeName：可选。用于事物的关联事物类型的字符串。

- `ThingGroups`：可选。事物所属的组的列表。
- `BillingGroup`：可选。关联账单组名称的字符串。
- `PackageVersions`：可选。关联软件包名称和版本名称的字符串。

证书资源

您可以通过以下方式之一指定证书：

- 证书签名请求 (CSR)。
- 现有设备证书的证书 ID。（仅证书 ID 可与实例集预调配模板一起使用。）
- 使用注册到 AWS IoT的 CA 证书创建的设备证书。如果您有多个 CA 证书注册到同一使用者字段，则还必须传入用于对设备证书进行签名的 CA 证书。

> **Note**
>
> 当您在模板中声明证书时，请只使用这些方法之一。例如，如果您使用了 CSR，就不能同时指定证书 ID 或设备证书。有关更多信息，请参阅 X.509 客户端证书。

有关更多信息，请参阅 X.509 证书概览。

证书资源使用以下属性进行声明：

- `CertificateSigningRequest`: 字符串。
- `CertificateId`: 字符串。
- `CertificatePem`: 字符串。
- `CACertificatePem`: 字符串。
- `Status`：可选。字符串可以是 `ACTIVE` 或 `INACTIVE`。默认值为 ACTIVE。

示例：

- 使用 CSR 指定的证书：

```
{
    "certificate" : {
```

```
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateSigningRequest": {"Ref" : "CSR"},
            "Status" : "ACTIVE"
        }
    }
}
```

- 使用现有证书 ID 指定的证书：

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CertificateId": {"Ref" : "CertificateId"}
        }
    }
}
```

- 使用现有证书 .pem 和 CA 证书 .pem 指定的证书：

```
{
    "certificate" : {
        "Type" : "AWS::IoT::Certificate",
        "Properties" : {
            "CACertificatePem": {"Ref" : "CACertificatePem"},
            "CertificatePem": {"Ref" : "CertificatePem"}
        }
    }
}
```

策略资源

策略资源使用以下属性之一进行声明：

- PolicyName：可选。字符串。默认值为策略文档的哈希值。PolicyName 只能参考 AWS IoT 策略但不是 IAM policy。如果您使用的是现有 AWS IoT 策略，请为该PolicyName属性输入策略的名称。请勿包含 PolicyDocument 属性。
- PolicyDocument：可选。指定为转义字符串的 JSON 对象。如果未提供 PolicyDocument，则必须已经创建了策略。

> **Note**
>
> 如果存在 Policy 部分，则必须指定 PolicyName 或 PolicyDocument，但不能同时指定。

覆盖设置

如果模板指定了已经存在的资源，则使用 OverrideSettings 部分可以指定要采取的操作：

DO_NOTHING

将资源保留为原样。

REPLACE

使用在模板中指定的资源替换该资源。

FAIL

导致请求失败，出现 ResourceConflictsException。

MERGE

仅对 ThingGroups 的 AttributePayload 和 thing 属性有效。将事物的现有属性或组成员资格与模板中指定的同等内容合并。

当您声明事物资源时，您可以为以下属性指定 OverrideSettings：

- ATTRIBUTE_PAYLOAD
- THING_TYPE_NAME
- THING_GROUPS

当您声明证书资源时，您可以为 OverrideSettings 属性指定 Status。

OverrideSettings 不可用于策略资源。

资源示例

以下模板片段声明了一个事物、一个证书和一个策略：

```
{
```

```
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateSigningRequest": {"Ref" : "CSR"},
                "Status" : "ACTIVE"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\", \"Statement
\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

使用以下项声明事物：

- 逻辑名称 "thing"。
- 类型 AWS::IoT::Thing。
- 一组事物属性。

  事物属性包括事物名称、一组属性、一个事物类型名称 (可选) 以及事物所属的事物组列表 (可选)。

使用 {"Ref":"*parameter-name*"} 引用参数。评估模板时，使用随模板传入的字典中提供的参数值来替换参数。

使用以下项声明证书：

- 逻辑名称 "certificate"。
- 类型 AWS::IoT::Certificate。
- 一组属性。

  属性包括证书的 CSR 并将状态设置为 ACTIVE。CSR 文本作为随模板传入的字典中的参数传递。

使用以下项声明策略：

- 逻辑名称 "policy"。
- 类型 AWS::IoT::Policy。
- 现有策略的名称或策略文档的名称。

批量注册的模板示例

以下 JSON 文件是使用 CSR 指定证书的完整预调配模板的一个示例：

(PolicyDocument 字段值必须是指定为转义字符串的 JSON 对象。)

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber" : {
            "Type" : "String"
        },
        "Location" : {
            "Type" : "String",
            "Default" : "WA"
        },
        "CSR" : {
            "Type" : "String"
        }
    },
    "Resources" : {
```

```
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "ThingName" : {"Ref" : "ThingName"},
                "AttributePayload" : { "version" : "v1", "serialNumber" :  {"Ref" :
 "SerialNumber"}},
                "ThingTypeName" :  "lightBulb-versionA",
                "ThingGroups" : ["v1-lightbulbs", {"Ref" : "Location"}]
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateSigningRequest": {"Ref" : "CSR"},
                "Status" : "ACTIVE"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : "{ \"Version\": \"2012-10-17\", \"Statement
\": [{ \"Effect\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\":
 [\"arn:aws:iot:us-east-1:123456789012:topic/foo/bar\"] }] }"
            }
        }
    }
}
```

just-in-time配置模板示例 (JITP)

以下 JSON 文件是使用证书 ID 指定现有证书的完整预调配模板的一个示例：

```
{
   "Parameters":{
      "AWS::IoT::Certificate::CommonName":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::SerialNumber":{
         "Type":"String"
      },
      "AWS::IoT::Certificate::Country":{
         "Type":"String"
      },
```

```
      "AWS::IoT::Certificate::Id":{
         "Type":"String"
      }
   },
   "Resources":{
      "thing":{
         "Type":"AWS::IoT::Thing",
         "Properties":{
            "ThingName":{
               "Ref":"AWS::IoT::Certificate::CommonName"
            },
            "AttributePayload":{
               "version":"v1",
               "serialNumber":{
                  "Ref":"AWS::IoT::Certificate::SerialNumber"
               }
            },
            "ThingTypeName":"lightBulb-versionA",
            "ThingGroups":[
               "v1-lightbulbs",
               {
                  "Ref":"AWS::IoT::Certificate::Country"
               }
            ]
         },
         "OverrideSettings":{
            "AttributePayload":"MERGE",
            "ThingTypeName":"REPLACE",
            "ThingGroups":"DO_NOTHING"
         }
      },
      "certificate":{
         "Type":"AWS::IoT::Certificate",
         "Properties":{
            "CertificateId":{
               "Ref":"AWS::IoT::Certificate::Id"
            },
            "Status":"ACTIVE"
         }
      },
      "policy":{
         "Type":"AWS::IoT::Policy",
         "Properties":{
```

```
            "PolicyDocument":"{ \"Version\": \"2012-10-17\", \"Statement\": [{ \"Effect
\": \"Allow\", \"Action\":[\"iot:Publish\"], \"Resource\": [\"arn:aws:iot:us-
east-1:123456789012:topic/foo/bar\"] }] }"
         }
      }
   }
}
```

> **Important**
>
> 您必须在用于 JIT 预调配的模板中使用 `CertificateId`。

有关配置模板类型的更多信息，请参阅 AWS API 参考`CreateProvisioningTemplate`中的。

有关如何使用此模板进行置备的更多信息，请参阅：即时 just-in-time 配置。

实例集预调配

队列配置模板 AWS IoT 用于设置云和设备配置。这些模板使用与 JITP 和批量注册模板相同的参数和资源。有关更多信息，请参阅 预调配模板。实例集预调配模板可以包含一个 `Mapping` 部分和一个 `DeviceConfiguration` 部分。您可以在实例集预调配模板中使用内置函数来生成设备特定的配置。实例集预调配模板是命名的资源，由 ARN 标识（例如，`arn:aws:iot:us-west-2:1234568788:provisioningtemplate/`*`templateName`*）。

映像

可选的 `Mappings` 部分将密钥与对应的一组命名值相匹配。例如，如果要根据某个 AWS 区域设置值，则可以创建一个使用该 AWS 区域 名称作为键并包含要为每个特定区域指定的值的映射。您使用 `Fn::FindInMap` 内部函数来检索映射中的值。

您不得在 `Mappings` 部分包含参数、虚拟参数或调用内部函数。

设备配置

设备配置部分包含要在预调配时发送到设备的任意数据。例如：

```
{
    "DeviceConfiguration": {
        "Foo":"Bar"
    }
```

```
}
```

如果您使用 JavaScript 对象表示法 (JSON) 有效负载格式向设备发送消息， AWS IoT Core 请将此数据格式化为 JSON。如果您使用的是简明二进制对象表示 (CBOR) 有效载荷格式， AWS IoT Core 请将此数据格式化为 CBOR。`DeviceConfiguration` 部分不支持嵌套 JSON 对象。

内置函数

内部函数在预调配模板中除了 `Mappings` 部分以外的任何部分使用。

`Fn::Join`

将一组值附加到单值中，中间用特定分隔符隔开。如果分隔符为空字符串，则值连接在一起而不使用分隔符。

> **Important**
>
> `Fn::Join` 不受 the section called “策略资源” 支持。

`Fn::Select`

通过索引返回对象列表中的单个对象。

> **Important**
>
> `Fn::Select` 不会检查 `null` 值，或检查索引是否超出数组边界。这两个条件都会导致配置错误，因此请确保您选择了有效的索引值并且列表包含非空值。

`Fn::FindInMap`

返回与 `Mappings` 部分声明的双层映射中的键对应的值。

`Fn::Split`

将字符串拆分为字符串值列表，以便您可以从字符串列表中选择一个元素。您可以指定一个分隔符（如逗号），用于确定拆分字符串的位置。拆分字符串后，使用 `Fn::Select` 选择一个元素。

例如，如果将以逗号分隔的子网 ID 字符串导入您的堆栈模板，您可以在每个逗号的位置拆分字符串。从子网 ID 列表中，使用 `Fn::Select` 指定资源的子网 ID。

Fn::Sub

将输入字符串中的变量替换为您指定的值。您可以使用此函数构建命令或输出，其中包含在创建或更新堆栈之前不可用的值。

实例集预调配的模板示例

```
{
    "Parameters" : {
        "ThingName" : {
            "Type" : "String"
        },
        "SerialNumber": {
            "Type": "String"
        },
        "DeviceLocation": {
            "Type": "String"
        }
    },
    "Mappings": {
        "LocationTable": {
            "Seattle": {
                "LocationUrl": "https://example.aws"
            }
        }
    },
    "Resources" : {
        "thing" : {
            "Type" : "AWS::IoT::Thing",
            "Properties" : {
                "AttributePayload" : {
                    "version" : "v1",
                    "serialNumber" : "serialNumber"
                },
                "ThingName" : {"Ref" : "ThingName"},
                "ThingTypeName" : {"Fn::Join":["",["ThingPrefix_",
{"Ref":"SerialNumber"}]]},
                "ThingGroups" : ["v1-lightbulbs", "WA"],
                "BillingGroup": "LightBulbBillingGroup"
            },
            "OverrideSettings" : {
                "AttributePayload" : "MERGE",
                "ThingTypeName" : "REPLACE",
```

```
                "ThingGroups" : "DO_NOTHING"
            }
        },
        "certificate" : {
            "Type" : "AWS::IoT::Certificate",
            "Properties" : {
                "CertificateId": {"Ref": "AWS::IoT::Certificate::Id"},
                "Status" : "Active"
            }
        },
        "policy" : {
            "Type" : "AWS::IoT::Policy",
            "Properties" : {
                "PolicyDocument" : {
                    "Version": "2012-10-17",
                    "Statement": [{
                        "Effect": "Allow",
                        "Action":["iot:Publish"],
                        "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/foo/
bar"]
                    }]
                }
            }
        }
    },
    "DeviceConfiguration": {
        "FallbackUrl": "https://www.example.com/test-site",
        "LocationUrl": {
            "Fn::FindInMap": ["LocationTable",{"Ref": "DeviceLocation"},
 "LocationUrl"]}
        }
}
```

> **Note**
>
> 可以对现有预调配模板进行更新以添加预先预调配挂钩。

预先预调配挂钩

AWS 建议在创建配置模板时使用预配置挂钩功能，以便更好地控制您的账户已登录的设备和设备数量。预调配挂钩是 Lambda 函数，可在验证设备传递的参数后允许预调配设备。在您预调配设备之

前，此 Lambda 函数必须存在于您的账户中，因为每次设备通过 the section called “RegisterThing” 发送请求时都会调用该函数。

> **Important**
>
> 请务必在附加到 Lambda 操作策略的全局条件上下文键中包含 `source-arn` 或 `source-account`，以便防范权限操纵。有关此问题的更多信息，请参阅 防止跨服务混淆座席。

对于要预调配的设备，Lambda 函数必须接受输入对象并返回本节所述的输出对象。仅当 Lambda 函数返回带有 `"allowProvisioning": True` 的对象时，预调配才会继续。

预先预调配挂钩输入

AWS IoT 当设备向注册时，会将此对象发送到 Lambda 函数。 AWS IoT

```
{
    "claimCertificateId" : "string",
    "certificateId" : "string",
    "certificatePem" : "string",
    "templateArn" : "arn:aws:iot:us-east-1:1234567890:provisioningtemplate/MyTemplate",
    "clientId" : "221a6d10-9c7f-42f1-9153-e52e6fc869c1",
    "parameters" : {
        "string" : "string",
        ...
    }
}
```

传递给 Lambda 函数的 `parameters` 对象包含 `parameters` 参数中的属性，该参数在 the section called “RegisterThing” 请求负载中传入。

预先预调配挂钩返回值

Lambda 函数必须返回一个响应，指示它是否已授权预调配请求以及要覆盖的任意属性的值。

以下是来自预先预调配函数的成功响应示例。

```
{
    "allowProvisioning": true,
    "parameterOverrides" : {
```

```
        "Key": "newCustomValue",
        ...
    }
}
```

"parameterOverrides" 值将添加到 the section called “RegisterThing” 申请负载的 "parameters" 参数中。

> **Note**
>
> - 如果 Lambda 函数失败，则配置请求将失败，ACCESS_DENIED并在 Logs 中记录错误。 CloudWatch
> - 如果 Lambda 函数未在响应中返回 "allowProvisioning": "true"，则预调配请求将失败并显示 ACCESS_DENIED。
> - Lambda 函数必须在 5 秒内完成执行并返回，否则预调配请求将失败。

预先预调配挂钩 Lambda 示例

Python

在 Python 中预先预调配挂钩 Lambda 示例。

```
import json

def pre_provisioning_hook(event, context):
    print(event)

    return {
        'allowProvisioning': True,
        'parameterOverrides': {
            'DeviceLocation': 'Seattle'
        }
    }
```

Java

在 Java 中预先预调配挂钩 Lambda 示例。

处理程序类：

```
package example;

import java.util.Map;
import java.util.HashMap;
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

public class PreProvisioningHook implements
 RequestHandler<PreProvisioningHookRequest, PreProvisioningHookResponse> {

    public PreProvisioningHookResponse handleRequest(PreProvisioningHookRequest
 object, Context context) {
        Map<String, String> parameterOverrides = new HashMap<String, String>();
        parameterOverrides.put("DeviceLocation", "Seattle");

        PreProvisioningHookResponse response = PreProvisioningHookResponse.builder()
                .allowProvisioning(true)
                .parameterOverrides(parameterOverrides)
                .build();

        return response;
    }

}
```

请求类：

```
package example;

import java.util.Map;
import lombok.Builder;
import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;

@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PreProvisioningHookRequest {
    private String claimCertificateId;
    private String certificateId;
    private String certificatePem;
```

```
    private String templateArn;
    private String clientId;
    private Map<String, String> parameters;
}
```

响应类：

```
package example;

import java.util.Map;
import lombok.Builder;
import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;


@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class PreProvisioningHookResponse {
    private boolean allowProvisioning;
    private Map<String, String> parameterOverrides;
}
```

JavaScript

中预置挂钩 Lambda JavaScript 的示例。

```
exports.handler = function(event, context, callback) {
    console.log(JSON.stringify(event, null, 2));
    var reply = {
        allowProvisioning: true,
        parameterOverrides: {
            DeviceLocation: 'Seattle'
        }
     };
     callback(null, reply);
}
```

使用证书提供程序进行自我管理的 AWS IoT Core 证书签名

您可以创建 AWS IoT Core 证书提供商来签署 AWS IoT 队列配置中的证书签名请求 (CSR)。证书提供商引用 Lambda 函数和 `CreateCertificateFromCsr`MQTT API 进行队列配置。Lambda 函数接受 CSR 并返回签名的客户端证书。

当您的证书提供商没有证书提供商时 AWS 账户，系统会在队列配置中调用 CreateCertificateFromCsr MQTT API 来从 CSR 生成证书。创建证书提供商后，CreateCertificateFromCsr MQTT API 的行为将发生变化，对此 MQTT API 的所有调用都将调用证书提供商来颁发证书。

借助 AWS IoT Core 证书提供商，您可以实施利用私有证书颁发机构 (CA)（例如AWS Private CA其他公开信任的 CA）或您自己的公钥基础架构 (PKI) 来签署 CSR 的解决方案。此外，您还可以使用证书提供商自定义客户端证书的字段，例如有效期、签名算法、颁发者和扩展名。

> **Important**
>
> 每个证书提供商只能创建一个证书提供商 AWS 账户。签名行为更改适用于调用 CreateCertificateFromCsr MQTT API 的整个队列，直到您 AWS 账户从中删除证书提供商。

本主题内容：

自行管理的证书签名在队列配置中的工作原理

重要概念

以下概念提供的详细信息可以帮助您了解自管理证书签名在 AWS IoT 队列配置中的工作原理。有关更多信息，请参阅使用队列配置配置没有设备证书的设备。

AWS IoT 舰队配置

使用 AWS IoT 队列配置（队列配置的缩写），可在设备首次连接时 AWS IoT Core 生成设备证书并将其安全地交付 AWS IoT Core 给您的设备。您可以使用队列配置将没有设备证书的设备连接到 AWS IoT Core。

证书签名请求 (CSR)

在队列配置过程中，设备 AWS IoT Core 通过队列配置 MQTT API 向发出请求。此请求包括证书签名请求 (CSR)，该请求将被签名以创建客户端证书。

AWS 队列配置中的托管证书签名

AWS managed 是队列配置中证书签名的默认设置。使用 AWS 托管证书签名， AWS IoT Core 将使用自己的 CA 签署 CSR。

队列配置中的自管理证书签名

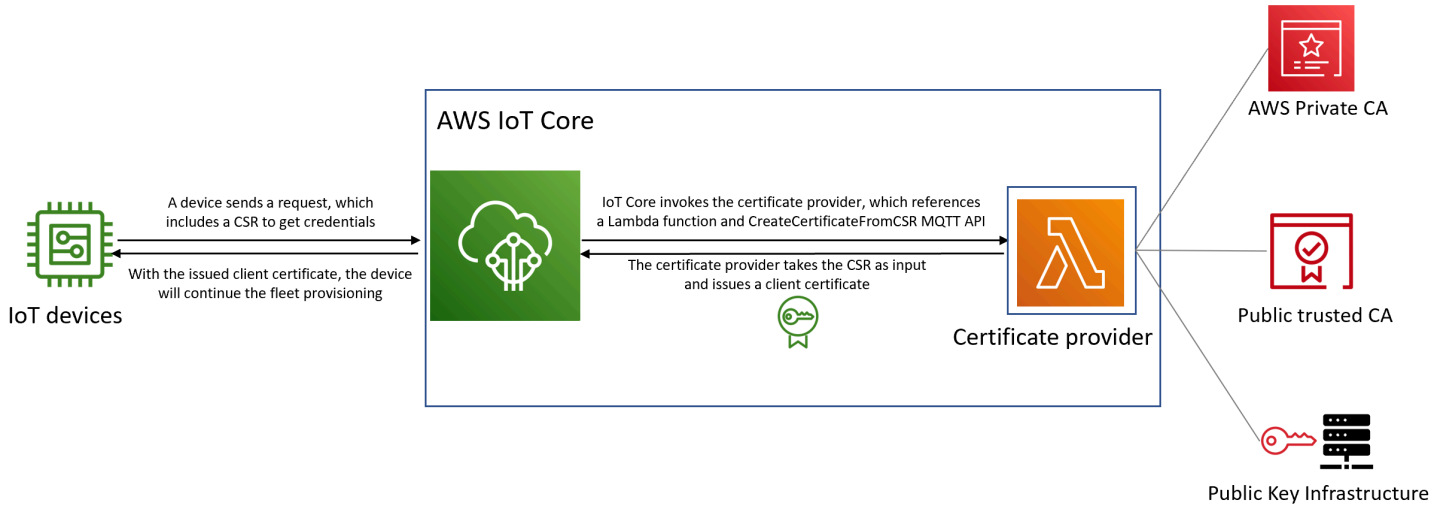
自行管理是队列配置中证书签名的另一种选择。使用自我管理的证书签名，您可以创建 AWS IoT Core 证书提供商来签署 CSR。您可以使用自我管理的证书签名与 AWS 私有 CA、其他公开信任的 CA 或您自己的公钥基础架构 (PKI) 生成的 CA 签署 CSR。

AWS IoT Core 证书提供商

AWS IoT Core 证书提供商（证书提供者的缩写）是一种客户管理的资源，用于队列配置中的自我管理证书签名。

图表

下图简要说明了自证书签名在 AWS IoT 队列配置中的工作原理。



- 在制造新的物联网设备或将其引入设备队列时，它需要使用客户端证书进行身份验证 AWS IoT Core。
- 作为队列配置过程的一部分，设备通过队列配置 MQTT API 向 AWS IoT Core 请求客户端证书。此请求包括证书签名请求 (CSR)。
- AWS IoT Core 调用证书提供者并将 CSR 作为输入传递给提供者。
- 证书提供者将 CSR 作为输入并颁发客户端证书。

  对于 AWS 托管证书签名， AWS IoT Core 使用自己的 CA 对 CSR 进行签名并颁发客户端证书。
- 有了已颁发的客户端证书，设备将继续进行队列配置并与建立安全连接 AWS IoT Core。

证书提供商 Lambda 函数输入

AWS IoT Core 当设备向 Lambda 函数注册时，会将以下对象发送到 Lambda 函数。的值`certificateSigningRequest`是请求中提供的隐私增强邮件 (PEM) 格式的 CSR。`CreateCertificateFromCsr``principalId`是发出`CreateCertificateFromCsr`请求 AWS IoT Core 时用于连接的主体的 ID。 `clientId`是为 MQTT 连接设置的客户端 ID。

```
{
 "certificateSigningRequest": "string",
 "principalId": "string",
 "clientId": "string"
}
```

证书提供商 Lambda 函数返回值

Lambda 函数必须返回包含该`certificatePem`值的响应。以下是成功响应的示例。 AWS IoT Core 将使用返回值 (`certificatePem`) 来创建证书。

```
{
 "certificatePem": "string"
}
```

如果注册成功，`CreateCertificateFromCsr`将在`CreateCertificateFromCsr`响应`certificatePem`中返回相同的结果。有关更多信息，请参阅的响应负载示例CreateCertificateFromCsr。

示例 Lambda 函数

在创建证书提供商之前，您必须创建一个 Lambda 函数来签署 CSR。以下是 Python 中的 Lambda 函数示例。此函数使用私有 CA 和签名算法调 AWS Private CA 用对输入 CSR 进行SHA256WITHRSA签名。返回的客户证书有效期为一年。有关私有 CA AWS Private CA 以及如何创建私有 CA 的更多信息，请参阅什么是 AWS 私有 CA？ 以及创建私有 CA。

```
import os
import time
import uuid
import boto3

def lambda_handler(event, context):
    ca_arn = os.environ['CA_ARN']
    csr = (event['certificateSigningRequest']).encode('utf-8')

    acmpca = boto3.client('acm-pca')
    cert_arn = acmpca.issue_certificate(
        CertificateAuthorityArn=ca_arn,
        Csr=csr,
        Validity={"Type": "DAYS", "Value": 365},
        SigningAlgorithm='SHA256WITHRSA',
        IdempotencyToken=str(uuid.uuid4())
    )['CertificateArn']

    # Wait for certificate to be issued
    time.sleep(1)
    cert_pem = acmpca.get_certificate(
        CertificateAuthorityArn=ca_arn,
        CertificateArn=cert_arn
    )['Certificate']

    return {
        'certificatePem': cert_pem
    }
```

Important

- Lambda 函数返回的证书必须具有与证书签名请求 (CSR) 相同的主题名称和公钥。
- Lambda 函数必须在 5 秒钟内完成运行。
- Lambda 函数必须与证书提供商 AWS 账户 资源位于同一区域中。

- 必须向 AWS IoT 服务委托人授予 Lambda 函数的调用权限。为避免混淆代理问题，我们建议您sourceAccount为调用权限设置sourceArn和。有关更多信息，请参阅防止跨服务混淆代理。

以下基于资源的 Lambd a 策略示例 AWS IoT 授予了调用 Lambda 函数的权限：

```
{
 "Version": "2012-10-17",
 "Id": "InvokePermission",
 "Statement": [
  {
   "Sid": "LambdaAllowIotProvider",
   "Effect": "Allow",
   "Principal": {
    "Service": "iot.amazonaws.com"
   },
   "Action": "lambda:InvokeFunction",
   "Resource": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
   "Condition": {
    "StringEquals": {
     "AWS:SourceAccount": "123456789012"
    },
    "ArnLike": {
     "AWS:SourceArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-
certificate-provider"
    }
   }
  }
 ]
}
```

用于队列配置的自管理证书签名

您可以使用 AWS CLI 或 AWS Management Console选择自行管理的证书签名进行队列置备。

AWS CLI

要选择自我管理的证书签名，您必须创建一个 AWS IoT Core 证书提供商，以便在队列配置中对 CSR 进行签名。 AWS IoT Core 调用证书提供程序，该提供程序将 CSR 作为输入并返回客户端证书。要

创建证书提供者，请使用 CreateCertificateProvider API 操作或 create-certificate-provider CLI 命令。

> **Note**
>
> 创建证书提供商后，用于队列预配 CreateCertificateFromCsr API 的行为将发生变化，因此对的所有调用都CreateCertificateFromCsr将调用证书提供商来创建证书。创建证书提供商后，这种行为可能需要几分钟才能改变。

```
aws iot create-certificate-provider \
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-function-1 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

以下显示了此命令的输出示例：

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-certificate-provider"
}
```

有关更多信息，请参阅 CreateCertificateProvider AWS IoTAPI 参考中的。

AWS Management Console

要选择使用自行管理的证书签名 AWS Management Console，请执行以下步骤：

1. 转到 AWS IoT 控制台。
2. 在左侧导航栏的 “安全” 下，选择 “证书签名”。
3. 在证书签名页面的证书签名详细信息下，选择编辑证书签名方法。
4. 在编辑证书签名方法页面的证书签名方法下，选择自我管理。
5. 在自管理设置部分，输入证书提供商的名称，然后创建或选择 Lambda 函数。
6. 选择更新证书签名。

AWS CLI 证书提供商的命令

创建证书提供商

要创建证书提供者，请使用 CreateCertificateProvider API 操作或 create-certificate-provider CLI 命令。

> **Note**
>
> 创建证书提供商后，用于队列预配 CreateCertificateFromCsr API 的行为将发生变化，因此对的所有调用都CreateCertificateFromCsr将调用证书提供商来创建证书。创建证书提供商后，这种行为可能需要几分钟才能改变。

```
aws iot create-certificate-provider \
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-function-1 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

以下显示了此命令的输出示例：

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-certificate-provider"
}
```

有关更多信息，请参阅 CreateCertificateProvider AWS IoTAPI 参考中的。

更新证书提供商

要更新证书提供者，请使用 UpdateCertificateProvider API 操作或 update-certificate-provider CLI 命令。

```
aws iot update-certificate-provider \
                --certificateProviderName my-certificate-provider \
                --lambdaFunctionArn arn:aws:lambda:us-east-1:123456789012:function:my-function-2 \
                --accountDefaultForOperations CreateCertificateFromCsr
```

以下显示了此命令的输出示例：

```
{
 "certificateProviderName": "my-certificate-provider",
 "certificateProviderArn": "arn:aws:iot:us-east-1:123456789012:certificateprovider:my-
certificate-provider"
}
```

有关更多信息，请参阅 UpdateCertificateProvider AWS IoTAPI 参考中的。

描述证书提供商

要描述证书提供者，请使用 DescribeCertificateProvider API 操作或 describe-certificate-provider CLI 命令。

```
aws iot describe-certificate-provider --certificateProviderName my-certificate-provider
```

以下显示了此命令的输出示例：

```
{
 "certificateProviderName": "my-certificate-provider",
 "lambdaFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function",
 "accountDefaultForOperations": [
  "CreateCertificateFromCsr"
 ],
 "creationDate": "2022-11-03T00:15",
 "lastModifiedDate": "2022-11-18T00:15"
}
```

有关更多信息，请参阅 DescribeCertificateProvider AWS IoTAPI 参考中的。

删除证书提供商

要删除证书提供商，请使用 DeleteCertificateProvider API 操作或 delete-certificate-provider CLI 命令。如果您删除证书提供商资源，则的行为CreateCertificateFromCsr将恢复， AWS IoT 并将创建 AWS IoT 由 CSR 签名的证书。

```
aws iot delete-certificate-provider --certificateProviderName my-certificate-provider
```

此命令不会生成任何输出。

有关更多信息，请参阅 DeleteCertificateProvider AWS IoTAPI 参考中的。

列出证书提供商

要列出您的证书提供商 AWS 账户，请使用 ListCertificateProviders API 操作或 list-certificate-providers CLI 命令。

```
aws iot list-certificate-providers
```

以下显示了此命令的输出示例：

```
{
 "certificateProviders": [
  {
   "certificateProviderName": "my-certificate-provider",
   "certificateProviderArn": "arn:aws:iot:us-
east-1:123456789012:certificateprovider:my-certificate-provider"
  }
 ]
}
```

有关更多信息，请参阅 ListCertificateProviderAWS IoTAPI 参考中的。

为安装设备的用户创建 IAM policy 和角色

> **Note**
>
> 这些过程仅在 AWS IoT 控制台的指导下使用。
> 要从控制台转到此页面，请打开创建新的预调配模板。

为什么不能在 AWS IoT 控制台中完成此操作？

为了获得最安全的体验，应在 IAM 控制台中执行 IAM 操作。本节中的过程将引导您完成创建相关 IAM 角色和策略的步骤，使用预调配模板需要这些 IAM 角色和策略。

为将安装设备的用户创建 IAM policy

此过程介绍如何创建授权用户使用预调配模板安装设备的 IAM policy。

执行此过程时，您将在 IAM 控制台和控制台之间切换。 AWS IoT 我们建议您在完成此过程时同时打开这两个控制台。

为将安装设备的用户创建 IAM policy

1. 打开 IAM 控制台中的 Policies（策略）中心。
2. 选择创建策略。
3. 在创建策略页面上，选择 JSON 选项卡。
4. 切换到 AWS IoT 控制台中您选择配置用户策略和角色的页面。
5. 在 Sample provisioning policy（示例预调配策略）中，选择 Copy（复制）。
6. 切换回 IAM 控制台。
7. 在 JSON 编辑器中，粘贴您从 AWS IoT 控制台复制的策略。此政策特定于您在 AWS IoT 控制台中创建的模板。
8. 要继续，请选择下一步：标签。
9. 在 Add tags (Optional) [添加标签（可选）] 页面上，为要添加到此策略的每个标签选择 Add tag（添加标签）。如果没有任何标签要添加，您可以跳过该步骤。
10. 要继续，请选择下一步：审核。
11. 在查看策略页面上，执行以下操作：

    a. 对于 Name*（名称*），为策略输入可帮助您记住其作用的名称。

       记下您为该策略提供的名称，因为您将在下一步中使用该名称。

    b. 您可以选择输入您创建的策略的可选描述。

    c. 查看本策略的其余部分及其标签。
12. 要完成创建新策略，请选择 Create policy（创建策略）。

创建新策略后，继续执行the section called “为将安装设备的用户创建 IAM 角色”，以创建要附加此策略的用户角色条目。

为将安装设备的用户创建 IAM 角色

以下步骤介绍如何创建一个 IAM 角色，以对将使用预调配模板安装设备的用户进行身份验证。

为将安装设备的用户创建 IAM policy

1. 打开 IAM 控制台中的 Role（角色）中心。

2. 选择 创建角色。
3. 在 Select trusted entity（选择可信实体）中，选择可信实体的类型，您要向该可信实体授予对您正在创建的模板的访问权限。
4. 选择或输入您要向其授予访问权限的可信实体的标识，然后选择 Next（下一步）。
5. 在 Add permissions（添加权限）页面的 Permission policies（权限策略）中的搜索框中，输入您在上一个过程中创建的策略的名称。
6. 对于策略列表，选择在上一个过程中创建的策略，然后选择 Next（下一步）。
7. 在 Name, review, and create（命名、查看和创建）部分中，执行以下操作：

   a. 对于 Role name（角色名称），键入有助于您记住此角色的作用的角色名称。

   b. 对于 Description（描述），您可以选择输入角色的可选描述。不需要执行此操作即可继续。

   c. 查看 Step 1（步骤 1）和 Step 2（步骤 2）中的值。

   d. 对于 Add tags (Optional) [添加标签（可选）]，可以选择向该角色添加标签。不需要执行此操作即可继续。

   e. 确保该页面上的信息完整且正确无误，然后选择 Create role（创建角色）。

创建新角色后，返回 AWS IoT 控制台继续创建模板。

更新现有策略以向新模板授权

以下步骤介绍如何向 IAM policy 添加新模板，此策略授权用户使用预调配模板安装设备。

向现有 IAM policy 添加新的模板

1. 打开 IAM 控制台中的 Policies（策略）中心。
2. 在搜索框中，输入要更新的策略的名称。
3. 在搜索框下方的列表中，找到要更新的策略并选择策略名称。
4. 对于 Policy summary（策略摘要），选择 JSON 选项卡（如果该面板尚不可见）。
5. 要编辑策略文档，请选择 Edit policy（编辑策略）。
6. 在编辑器中，选择 JSON 选项卡（如果该面板尚不可见）。
7. 在策略文档中，查找包含 `iot:CreateProvisioningClaim` 操作的策略语句。

   如果策略文档不包含带有 `iot:CreateProvisioningClaim` 操作的策略语句，请复制以下语句片段，并将其作为附加项粘贴到策略文档的 `Statement` 数组中。

> **Note**
>
> 此片段必须放在 Statement 数组中的结束] 字符之前。您可能需要在此片段之前或之后添加逗号，以更正任何语法错误。

```
{
    "Effect": "Allow",
    "Action": [
        "iot:CreateProvisioningClaim"
    ],
    "Resource": [
        "--PUT YOUR NEW TEMPLATE ARN HERE--"
    ]
}
```

8. 切换到 AWS IoT 控制台中您选择修改用户角色权限的页面。
9. 找到模板的 Resource ARN（资源 ARN）并选择 Copy（复制）。
10. 切换回 IAM 控制台。
11. 将复制的 Amazon 资源名称（ARN）粘贴到 Statement 数组中模板 ARN 列表的顶部，使其成为第一个条目。

    如果这是数组中唯一的 ARN，请删除刚粘贴的值末尾的逗号。
12. 查看更新的策略语句，并更正编辑器指示的任何错误。
13. 要保存更新的策略文档，请选择 Review policy（查看策略）。
14. 查看策略，然后选择 Save changes（保存更改）。
15. 返回 AWS IoT 控制台。

设备预调配 MQTT API

队列配置服务支持以下 MQTT API 操作：

- the section called “CreateCertificateFromCsr”
- the section called “CreateKeysAndCertificate”
- the section called “RegisterThing”

此 API 支持简明二进制对象表示 (CBOR) 格式和 JavaScript 对象表示法 (JSON) 的响应缓冲区，具体取决于主题的*####*格式。为清楚起见，本节中的响应和请求示例以 JSON 格式显示。

payload-format	响应格式数据类型
cbor	简洁二进制对象表示法 (CBOR)
json	JavaScript 对象表示法 (JSON)

> **Important**
>
> 在发布请求消息主题之前，请订阅响应主题以接收响应。此 API 使用的消息使用 MQTT 的发布/订阅协议来提供请求和响应交互。
> 如果您在发布请求之前没有订阅响应主题，则可能不会收到该请求的结果。

CreateCertificateFromCsr

根据证书签名请求 (CSR) 创建证书。 AWS IoT 提供由 Amazon 根证书颁发机构 (CA) 签署的客户证书。新证书的状态为 PENDING_ACTIVATION。当您调用 RegisterThing 使用此证书预调配事物时，证书状态将如模板中所述，更改为 ACTIVE 或 INACTIVE。

有关使用证书颁发机构证书和证书签名请求来创建客户端证书的更多信息，请参阅使用您的 CA 证书创建客户端证书。

> **Note**
>
> 为了安全起见，CreateCertificateFromCsr 返回的 certificateOwnershipToken 会在一小时后过期。必须在 certificateOwnershipToken 过期前调用 RegisterThing。如果在令牌到期时，由创建的证书CreateCertificateFromCsr尚未激活并附加到策略或事物，则该证书将被删除。如果令牌过期，设备可以调用 CreateCertificateFromCsr 以生成新证书。

CreateCertificateFromCsr请求

发布包含 $aws/certificates/create-from-csr/*payload-format* 主题的消息。

`payload-format`

消息负载格式为 `cbor` 或 `json`。

CreateCertificateFromCsr请求有效载荷

```
{
    "certificateSigningRequest": "string"
}
```

`certificateSigningRequest`

CSR，采用 PEM 格式。

CreateCertificateFromCsr响应

订阅 `$aws/certificates/create-from-csr/payload-format/accepted`。

`payload-format`

消息负载格式为 `cbor` 或 `json`。

CreateCertificateFromCsr 响应有效载荷

```
{
    "certificateOwnershipToken": "string",
    "certificateId": "string",
    "certificatePem": "string"
}
```

`certificateOwnershipToken`

在预调配期间证明证书所有权的令牌。

`certificateId`

证书的 ID。证书管理操作仅接受 certificateId。

`certificatePem`

PEM 格式的证书数据。

CreateCertificateFromCsr 错误

要接收错误响应，请订阅 `$aws/certificates/create-from-csr/payload-format/rejected`。

`payload-format`

消息负载格式为 `cbor` 或 `json`。

CreateCertificateFromCsr 错误有效载荷

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

状态代码。

`errorCode`

错误代码。

`errorMessage`

错误消息。

CreateKeysAndCertificate

创建新密钥和证书。 AWS IoT 提供由 Amazon 根证书颁发机构 (CA) 签署的客户证书。新证书的状态为 `PENDING_ACTIVATION`。当您调用 `RegisterThing` 使用此证书预调配事物时，证书状态将如模板中所述，更改为 `ACTIVE` 或 `INACTIVE`。

> **Note**
>
> 为了安全起见，CreateKeysAndCertificate 返回的 `certificateOwnershipToken` 会在一小时后过期。必须在 `certificateOwnershipToken` 过期前调用 RegisterThing。如果在令牌到期时，由创建的证书CreateKeysAndCertificate尚未激活并附加到策略或事

物，则该证书将被删除。如果令牌过期，设备可以调用 CreateKeysAndCertificate 以生成新证书。

CreateKeysAndCertificate请求

在 $aws/certificates/create/*payload-format* 上发布带有空消息负载的消息。

payload-format

消息负载格式为 cbor 或 json。

CreateKeysAndCertificate响应

订阅 $aws/certificates/create/*payload-format*/accepted。

payload-format

消息负载格式为 cbor 或 json。

CreateKeysAndCertificate响应

```
{
    "certificateId": "string",
    "certificatePem": "string",
    "privateKey": "string",
    "certificateOwnershipToken": "string"
}
```

certificateId

证书 ID。

certificatePem

PEM 格式的证书数据。

privateKey

私有密钥。

`certificateOwnershipToken`

在预调配期间证明证书所有权的令牌。

CreateKeysAndCertificate 错误

要接收错误响应，请订阅 `$aws/certificates/create/payload-format/rejected`。

`payload-format`

消息负载格式为 `cbor` 或 `json`。

CreateKeysAndCertificate错误有效载荷

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

状态代码。

`errorCode`

错误代码。

`errorMessage`

错误消息。

RegisterThing

使用预定义的模板预调配事物。

RegisterThing 请求

在 `$aws/provisioning-templates/templateName/provision/payload-format` 上发布消息。

`payload-format`

消息负载格式为 `cbor` 或 `json`。

`templateName`

预调配模板名称。

RegisterThing 请求有效载荷

```
{
    "certificateOwnershipToken": "string",
    "parameters": {
        "string": "string",
        ...
    }
}
```

`certificateOwnershipToken`

用于证明证书所有权的令牌。 AWS IoT 当您通过 MQTT 创建证书时生成令牌。

`parameters`

可选。来自设备的键/值对由预先预调配挂钩用于评估注册请求。

RegisterThing 响应

订阅 `$aws/provisioning-templates/`*`templateName`*`/provision/`*`payload-format`*`/accepted`。

`payload-format`

消息负载格式为 `cbor` 或 `json`。

`templateName`

预调配模板名称。

RegisterThing 响应有效载荷

```
{
```

```
    "deviceConfiguration": {
        "string": "string",
        ...
    },
    "thingName": "string"
}
```

`deviceConfiguration`

在模板中定义的设备配置。

`thingName`

预调配期间创建的 IoT 事物的名称。

RegisterThing 错误响应

要接收错误响应，请订阅 `$aws/provisioning-templates/templateName/provision/payload-format/rejected`。

`payload-format`

消息负载格式为 `cbor` 或 `json`。

`templateName`

预调配模板名称。

RegisterThing 错误响应有效负载

```
{
    "statusCode": int,
    "errorCode": "string",
    "errorMessage": "string"
}
```

`statusCode`

状态代码。

`errorCode`

错误代码。

`errorMessage`

错误消息。

机群索引

您可以使用实例集索引从以下来源对设备的数据编制索引、进行搜索和聚合：AWS IoT 注册表、AWS IoT 设备影子、AWS IoT 连接、AWS IoT 设备管理软件包目录和 AWS IoT Device Defender 违规。您可以查询一组设备，并根据设备属性的不同组合（包括状态、连接性和设备违规）聚合设备记录的统计信息。借助机群索引，您可以组织、调查设备机群并进行故障排除。

机群索引提供以下功能。

管理索引更新

您可以设置实例集索引，以便对有关事物组、事物注册表、设备影子、设备连接和设备违规的更新编制索引。激活实例集索引时， AWS IoT 会为事物或事物组创建索引。AWS_Things 是为所有事物创建的索引。AWS_ThingGroups 是包含所有事物组的索引。在实例集索引处于活动状态之后，您可以对索引运行查询。例如，您可以找到所有电池续航时间超过 70% 的手持设备。 AWS IoT 使用您的最新数据不断更新索引。更多信息，请参阅管理机群索引。

跨数据源搜索

您可以基于查询语言创建查询字符串，并使用它在数据来源之间进行搜索。您还需要在队列索引设置中配置数据源，以便索引配置包含要搜索的数据源。查询字符串描述了您要查找的内容。您可以使用 AWS 托管字段、自定义字段以及索引数据源中的任何属性来创建查询。有关支持机群索引的数据源的更多信息，请参阅管理事物索引。

查询聚合数据

您可以在设备中搜索聚合数据，然后返回统计数据、百分位数、基数或与特定字段相关的搜索查询的事物列表。您可以对 AWS 托管字段或在队列索引设置中配置为自定义字段的任何属性运行聚合。有关聚合查询的更多信息，请参阅查询聚合数据。

使用实例集指标监控聚合数据并创建警报

您可以使用舰队指标 CloudWatch 自动向其发送汇总数据，分析趋势，并创建警报以根据预定义的阈值监控车队的汇总状态。有关机群指标的更多信息，请参阅机群指标。

管理机群索引

实例集索引为您管理两种类型的索引：事物索引和事物组索引。

事物索引

为您的所有事物创建的索引称为 AWS_Things。事物索引支持以下数据源：AWS IoT 注册表数据，AWS IoT Device ShadowDATA，AWS IoT 连接数据以及AWS IoT Device Defender违规数据。通过将这些数据来源添加到实例集索引配置，您可以搜索事物，查询聚合数据，并根据搜索查询创建动态事物组和实例集指标。

注册表-AWS IoT 提供可帮助您管理事物的注册表。您可以将注册表数据添加到实例集索引配置，以根据事物名称、描述和其他注册表属性搜索设备。有关注册表的更多信息，请参阅如何使用注册表管理事物。

影子 - AWS IoT Device Shadow 服务提供了有助于存储设备状态数据的影子。事物索引支持经典未命名影子和命名影子。要为命名影子编制索引，请激活您的命名影子设置并在事物索引配置中指定影子名称。默认情况下，您最多可以为每个影子添加 10 个阴影名称 AWS 账户。要了解如何增加影子名称的数量限制，请参阅《AWS 一般参考》中的 AWS IoT Device Management 配额。

要为索引添加命名影子，请执行以下操作：

- 如果您使用 AWS IoT 控制台，请打开 Thing indexing（事物索引），选择 Add named shadows（添加命名的影子），然后通过 Named shadow selection（命名的影子选择）添加您的影子名称。
- 如果使用 AWS Command Line Interface (AWS CLI)，则设置namedShadowIndexingMode为ON，然后在中指定阴影名称IndexingFilter。要查看 CLI 命令示例，请参阅管理事物索引。

> **Important**
>
> 2022 年 7 月 20 日是 AWS IoT 设备管理队列索引集成的正式上市 (GA) 版本，其中包含 AWS IoT Core 命名阴影和 AWS IoT Device Defender 检测违规行为。在此 GA 版中，您可以通过指定影子名称为特定的命名影子编制索引。如果您在 2021 年 11 月 30 日至 2022 年 7 月 19 日此特征的公开预览期添加了命名影子以编制索引，我们鼓励您重新配置实例集索引设置并选择特定的影子名称，以降低索引成本并优化性能。

有关影子的更多信息，请参阅 AWS IoT Device Shadow 服务。

连接 - 设备连接数据可帮助您识别设备的连接状态。此连接数据由生命期事件驱动。当客户端连接或断开连接时，会向 MQTT 主题 AWS IoT 发布带有消息的生命周期事件。连接或断开连接消息可以是提供连接状态详细信息的 JSON 元素的列表。有关设备连接的更多信息，请参阅生命周期事件。

Device Defender AWS IoT Device Defender 违规-违规数据有助于根据您在安全配置文件中定义的正常行为来识别异常设备行为。安全配置文件包含一组目标设备行为。每种行为都使用一个指标来指定设备的正常行为。有关 Device Defender 违规的更多信息，请参阅AWS IoT Device Defender 检测。

有关更多信息，请参阅管理事物索引。

事物组索引

AWS_ThingGroups是包含您的所有事物组和账单组的索引。您可以使用此索引基于组名称、描述、属性和所有父组名称搜索事物组。

有关更多信息，请参阅管理事物组索引。

托管字段

托管字段包含与事物、事物组、设备影子、设备连接和 Device Defender 违规相关的数据。 AWS IoT 定义托管字段中的数据类型。创建 AWS IoT 事物时，您可以指定每个托管字段的值。例如，事物名称、事物组和事物描述都是托管字段。实例集索引根据您指定的索引模式为托管式字段编制索引。托管字段无法更改或显示在 customFields 中。有关更多信息，请参阅自定义字段。

以下列出了用于事物索引的托管字段：

- 注册表的托管字段

```
"managedFields" : [
  {name:thingId, type:String},
  {name:thingName, type:String},
  {name:registry.version, type:Number},
  {name:registry.thingTypeName, type:String},
  {name:registry.thingGroupNames, type:String},
]
```

- 经典未命名影子的托管字段

```
"managedFields" : [
  {name:shadow.version, type:Number},
```

```
  {name:shadow.hasDelta, type:Boolean}
]
```

- 命名影子的托管字段

```
"managedFields" : [
  {name:shadow.name.shadowName.version, type:Number},
  {name:shadow.name.shadowName.hasDelta, type:Boolean}
]
```

- 事物连接的托管字段

```
"managedFields" : [
  {name:connectivity.timestamp, type:Number},
  {name:connectivity.version, type:Number},
  {name:connectivity.connected, type:Boolean},
  {name:connectivity.disconnectReason, type:String}
]
```

- Device Defender 的托管字段

```
"managedFields" : [
  {name:deviceDefender.violationCount, type:Number},
  {name:deviceDefender.securityprofile.behaviorname.metricName, type:String},
  {name:deviceDefender.securityprofile.behaviorname.lastViolationTime, type:Number},
  {name:deviceDefender.securityprofile.behaviorname.lastViolationValue, type:String},
  {name:deviceDefender.securityprofile.behaviorname.inViolation, type:Boolean}
]
```

- 事物组的托管字段

```
"managedFields" : [
  {name:description, type:String},
  {name:parentGroupNames, type:String},
  {name:thingGroupId, type:String},
  {name:thingGroupName, type:String},
  {name:version, type:Number},
]
```

下表列出了不可搜索的托管字段。

数据来源	无法搜索的托管字段
注册表	`registry.version`
未命名的影子	`shadow.version`
命名的影子	`shadow.name.*.version`
Device Defender	`deviceDefender.version`
事物组	`version`

自定义字段

您可以通过创建自定义字段来对事物属性、Device Shadow 数据和 Device Defender 违规数据进行汇总来进行索引。`customFields` 属性是字段名称和数据类型对的列表。您可以根据数据类型执行聚合查询。您选择的索引模式会影响可以在 `customFields` 中指定的字段。例如，如果您指定 `REGISTRY` 索引模式，则无法从事物影子中指定字段。您可以使用更新索引配置 CLI 命令创建或更新自定义字段的（请参阅更新索引配置示例中的示例命令)。

- 自定义字段名称

事物和事物组属性的自定义字段名称以 `attributes.` 开头，后面是属性名称。如果启用了未命名的影子索引，事物可以有以 `shadow.desired` 或 `shadow.reported` 开头的自定义字段名称，然后是未命名的影子数据值名称。如果启用命名影子索引，事物可以有以 `shadow.name.*.desired.` 或 `shadow.name.*.reported.` 开头的自定义字段名称，然后是命名的影子数据值。如果 Device Defender 违规索引处于启用状态，则可以有以开头的自定义字段名称`deviceDefender.`，然后是 Device Defender 违规数据值。

前缀后面的属性或数据值名称只能包含字母数字、-（连字符）和 _（下划线）字符。不能有任何空格。

如果配置中的自定义字段与要为其编制索引的值存在类型不一致，实例集索引将忽略聚合查询的不一致值。 CloudWatch 在解决聚合查询问题时，日志很有用。有关更多信息，请参阅 队列索引服务的聚合查询疑难解答。

- 自定义字段类型

自定义字段类型具有以下支持的值：Number、String 和 Boolean。

管理事物索引

AWS_Things 是为您的所有事物创建的索引。您可以从以下数据源控制要索引的内容：AWS IoT 注册表数据，AWS IoT Device Shadow数据，AWS IoT 连接数据以及AWS IoT Device Defender违规数据。

本主题内容：

- 启用事物索引
- 描述事物索引
- 查询事物索引
- 限制和局限性
- 授权

启用事物索引

您可以使用 update-indexing-con figuration CLI 命令或UpdateIndexing配置 API 操作来创建索引并控制AWS_Things其配置。--thing-indexing-configuration (thingIndexingConfiguration) 参数允许您控制建立索引的数据类型（例如，注册表、影子和设备连接数据和Device Defender 违规数据）。

--thing-indexing-configuration 参数采用具有以下结构的字符串：

```
{
  "thingIndexingMode": "OFF"|"REGISTRY"|"REGISTRY_AND_SHADOW",
  "thingConnectivityIndexingMode": "OFF"|"STATUS",
  "deviceDefenderIndexingMode": "OFF"|"VIOLATIONS",
  "namedShadowIndexingMode": "OFF"|"ON",
  "managedFields": [
    {
      "name": "string",
      "type": "Number"|"String"|"Boolean"
    },
    ...
  ],
  "customFields": [
    {
```

```
      "name": "string",
      "type": "Number"|"String"|"Boolean"
    },
    ...
  ],
  "filter": {
     "namedShadowNames": [ "string" ],
     "geoLocations": [
        {
            "name": "String",
            "order": "LonLat|LatLon"
        }
    ]
  }
}
```

事物索引模式

您可以在索引配置中指定不同的索引模式，具体取决于要索引的数据源和搜索设备：

- thingIndexingMode：控制是否为注册表或影子编制索引。如果设置thingIndexingMode为OFF，则禁用事物索引。
- thingConnectivityIndexingMode：指定是否对事物连接数据进行索引。
- deviceDefenderIndexingMode：指定是否为设备防御者违规数据编制索引。
- namedShadowIndexingMode：指定是否对命名的影子数据进行索引。要选择要添加到实例集索引配置的命名影子，请将 namedShadowIndexingMode 设置为 ON并在 filter 中指定命名的影子名称。

下表显示了每种索引模式的有效值以及为每个值编制索引的数据源。

属性	有效值	注册表	影子	连接	DD 违规行为	命名的影子
thingIndexingMode	关闭					
	REGISTRY	✓				

属性	有效值	注册表	影子	连接	DD 违规行为	命名的影子
	REGISTRY_AND_SHADOW	✓	✓			
thingConnectivityIndexingMode	未指定。					
	关闭					
	STATUS			✓		
deviceDefenderIndexingMode	未指定。					
	关闭					
	违规				✓	
namedShadowIndexingMode	未指定。					
	关闭					
	开					✓

托管字段和自定义字段

托管字段

托管字段包含与事物、事物组、设备影子、设备连接和 Device Defender 违规相关的数据。 AWS IoT 定义托管字段中的数据类型。您可以在创建 AWS IoT 事物时指定每个托管式字段的值。例如，事物名称、事物组和事物描述都是托管字段。实例集索引根据您指定的索引模式为托管式字段编制索引。托管字段无法更改或显示在 customFields 中。

自定义字段

您可以通过创建自定义字段来对属性、Device Shadow 数据和 Device Defender 违规数据进行汇总来进行索引。customFields 属性是字段名称和数据类型对的列表。您可以根据数据类型执行聚合查询。您选择的索引模式会影响可以在 customFields 中指定的字段。例如，如果您指定 REGISTRY

索引模式，则无法从事物影子中指定字段。您可以使用更新索引配置 CLI 命令创建或更新自定义字段的（请参阅更新索引配置示例中的示例命令)。有关更多信息，请参阅自定义字段。

索引过滤器

索引过滤器为命名阴影和地理定位数据提供了其他选择。

namedShadowNames

要将命名阴影添加到队列索引配置中，请namedShadowIndexingMode将其设置为，ON然后在namedShadowNames过滤器中指定您的命名阴影名称。

示例

```
"filter": {
     "namedShadowNames": [ "namedShadow1", "namedShadow2" ]
}
```

geoLocations

要将地理位置数据添加到舰队索引配置中，请执行以下操作：

- 如果您的地理位置数据存储在经典（未命名）阴影中，请thingIndexingMode将其设置为 REGISTRY_AND_SHADOW，并在过滤器中指定您的地理位置数据。geoLocations

  下面的示例过滤器在经典（未命名）阴影中指定了 GeoLocation 对象：

  ```
  "filter": {
       "geoLocations": [
          {
              "name": "shadow.reported.location",
              "order": "LonLat"
          }
       ]
    }
  ```

- 如果您的地理定位数据存储在命名的阴影中，请将其设置namedShadowIndexingMode为开启，在过滤器中添加阴影名称，然后在namedShadowNames过滤器中指定您的地理位置数据。geoLocations

  下面的示例过滤器在命名的 shadow (nameShadow1) 中指定了 GeoLocation 对象：

```
"filter": {
     "namedShadowNames": [ "namedShadow1" ],
     "geoLocations": [
        {
            "name": "shadow.name.namedShadow1.reported.location",
            "order": "LonLat"
        }
     ]
  }
```

有关更多信息，请参阅《AWS IoTAPI IndexingFilter参考》。

更新索引配置示例

要更新您的索引配置，请使用 AWS IoT update-indexing-configuration CLI 命令。以下示例显示了如何使用 update-indexing-configuration 标签。

简短语法：

```
aws iot update-indexing-configuration --thing-indexing-configuration \
'thingIndexingMode=REGISTRY_AND_SHADOW, deviceDefenderIndexingMode=VIOLATIONS,
namedShadowIndexingMode=ON,filter={namedShadowNames=[thing1shadow]},
 thingConnectivityIndexingMode=STATUS,
customFields=[{name=attributes.version,type=Number},
{name=shadow.name.thing1shadow.desired.DefaultDesired, type=String},
 {name=shadow.desired.power, type=Boolean},
{name=deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number,
 type=Number}]'
```

JSON 语法：

```
aws iot update-indexing-configuration --cli-input-json \ '{
          "thingIndexingConfiguration": { "thingIndexingMode": "REGISTRY_AND_SHADOW",
          "thingConnectivityIndexingMode": "STATUS",
          "deviceDefenderIndexingMode": "VIOLATIONS",
          "namedShadowIndexingMode": "ON",
          "filter": { "namedShadowNames": ["thing1shadow"]},
          "customFields": [ { "name": "shadow.desired.power", "type": "Boolean" },
          {"name": "attributes.version", "type": "Number"},
          {"name": "shadow.name.thing1shadow.desired.DefaultDesired", "type":
 "String"},
```

```
          {"name":
 "deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number",
 "type": Number} ] } }'
```

此命令不会生成任何输出。

要检查事物索引状态，请运行 `describe-index` CLI 命令：

```
aws iot describe-index --index-name "AWS_Things"
```

`describe-index` 命令的输出如下所示：

```
{
    "indexName": "AWS_Things",
    "indexStatus": "ACTIVE",
    "schema": "MULTI_INDEXING_MODE"
}
```

> **Note**
>
> 机群索引可能需要一点时间才能更新机群指数。我们建议 `indexStatus` 在使用之前显示 ACTIVE。根据配置的数据来源，您可以在模式字段中具有不同的值。有关更多信息，请参阅描述事物索引。

要获取事物索引配置详细信息，请运行 `get-indexing-configuration` CLI 命令：

```
aws iot get-indexing-configuration
```

`get-indexing-configuration` 命令的输出如下所示：

```
{
    "thingIndexingConfiguration": {
        "thingIndexingMode": "REGISTRY_AND_SHADOW",
        "thingConnectivityIndexingMode": "STATUS",
        "deviceDefenderIndexingMode": "VIOLATIONS",
        "namedShadowIndexingMode": "ON",
        "managedFields": [
            {
                "name": "connectivity.disconnectReason",
                "type": "String"
```

```
        },
        {
            "name": "registry.version",
            "type": "Number"
        },
        {
            "name": "thingName",
            "type": "String"
        },
        {
            "name": "deviceDefender.violationCount",
            "type": "Number"
        },
        {
            "name": "shadow.hasDelta",
            "type": "Boolean"
        },
        {
            "name": "shadow.name.*.version",
            "type": "Number"
        },
        {
            "name": "shadow.version",
            "type": "Number"
        },
        {
            "name": "connectivity.version",
            "type": "Number"
        },
        {
            "name": "connectivity.timestamp",
            "type": "Number"
        },
        {
            "name": "shadow.name.*.hasDelta",
            "type": "Boolean"
        },
        {
            "name": "registry.thingTypeName",
            "type": "String"
        },
        {
            "name": "thingId",
            "type": "String"
```

```
            },
            {
                "name": "connectivity.connected",
                "type": "Boolean"
            },
            {
                "name": "registry.thingGroupNames",
                "type": "String"
            }
        ],
        "customFields": [
            {
                "name": "shadow.name.thing1shadow.desired.DefaultDesired",
                "type": "String"
            },

            {
                "name":
 "deviceDefender.securityProfile1.NUMBER_VALUE_BEHAVIOR.lastViolationValue.number",
                "type": "Number"
            },
            {
                "name": "shadow.desired.power",
                "type": "Boolean"
            },
            {
                "name": "attributes.version",
                "type": "Number"
            }
        ],
        "filter": {
              "namedShadowNames": [
                  "thing1shadow"
              ]
          }
      },
    "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "OFF"
    }
}
```

要更新自定义字段，可以运行 update-indexing-configuration 命令。示例如下：

```
aws iot update-indexing-configuration --thing-indexing-configuration

 'thingIndexingMode=REGISTRY_AND_SHADOW,customFields=[{name=attributes.version,type=Number},
{name=attributes.color,type=String},{name=shadow.desired.power,type=Boolean},
{name=shadow.desired.intensity,type=Number}]'
```

此命令已将 shadow.desired.intensity 添加到索引配置中。

> **Note**
>
> 更新自定义字段索引配置将覆盖所有现有的自定义字段。请确保在调用 update-indexing-configuration 时指定所有自定义字段。

重建索引后，您可以对新添加的字段、搜索注册表数据、影子数据和事物连接状态数据使用聚合查询。

更改索引模式时，请使用新的索引模式以确保所有自定义字段均有效。例如，如果从 REGISTRY_AND_SHADOW 模式开始使用名为 shadow.desired.temperature 的自定义字段，则必须先删除 shadow.desired.temperature 自定义字段，然后再将索引模式更改为 REGISTRY。如果您的索引配置包含未通过索引模式建立索引的自定义字段，则更新失败。

描述事物索引

以下命令说明了如何使用 describe-index CLI 命令来检索事物索引的当前状态。

```
aws iot describe-index --index-name "AWS_Things"
```

命令的响应如下所示：

```
{
    "indexName": "AWS_Things",
    "indexStatus": "BUILDING",
    "schema": "REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS"
}
```

首次为舰队编制索引时， AWS IoT 会生成索引。如果 indexStatus 处于 BUILDING 状态，您无法查询索引。事物索引的 schema 指示将对什么类型的数据 (REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS) 建立索引。

更改索引的配置会导致重新生成索引。此过程中的 indexStatus 为 REBUILDING。在重建期间，您可以对事物索引中的数据运行查询。例如，如果您将索引配置从 REGISTRY 更改为 REGISTRY_AND_SHADOW，同时正在重新生成索引，则您可以查询注册表数据，包括最新的更新。但是，在重新生成操作完成之前，无法查询影子数据。生成或重新生成索引所需的时间量取决于数据量。

根据配置的数据来源，您可能在模式字段中看到不同的值。下表显示了不同的模式值和相应的说明：

架构	描述
关闭	没有配置或索引数据源。
REGISTRY	对注册表数据建立索引。
REGISTRY_AND_SHADOW	对注册表数据和未命名（经典）影子数据建立索引。
REGISTRY_AND_CONNECTIVITY	为注册数据和连接数据建立索引。
REGISTRY_AND_SHADOW_AND_CONNECTIVITY_STATUS	对注册表数据、未命名（典型）影子数据和连接数据都建立了索引。
MULTI_INDEXING_MODE	除了注册表、未命名（经典）影子或连接数据之外，也对命名影子或 Device Defender 违规数据建立了索引。

查询事物索引

使用 search-index CLI 命令可查询索引中的数据。

```
aws iot search-index --index-name "AWS_Things" --query-string
          "thingName:mything*"
```

```
{
    "things":[{
         "thingName":"mything1",
         "thingGroupNames":[
            "mygroup1"
         ],
         "thingId":"a4b9f759-b0f2-4857-8a4b-967745ed9f4e",
         "attributes":{
```

```
            "attribute1":"abc"
         },
         "connectivity": {
            "connected":false,
            "timestamp":1556649874716,
            "disconnectReason": "CONNECTION_LOST"
         }
    },
    {
        "thingName":"mything2",
        "thingTypeName":"MyThingType",
        "thingGroupNames":[
            "mygroup1",
            "mygroup2"
        ],
        "thingId":"01014ef9-e97e-44c6-985a-d0b06924f2af",
        "attributes":{
            "model":"1.2",
            "country":"usa"
        },
        "shadow":{
            "desired":{
                "location":"new york",
                "myvalues":[3, 4, 5]
            },
            "reported":{
                "location":"new york",
                "myvalues":[1, 2, 3],
                "stats":{
                    "battery":78
                }
            },
            "metadata":{
                 "desired":{
                       "location":{
                            "timestamp":123456789
                        },
                       "myvalues":{
                             "timestamp":123456789
                       }
                  },
                  "reported":{
                        "location":{
                             "timestamp":34535454
```

```
                        },
                        "myvalues":{
                            "timestamp":34535454
                        },
                        "stats":{
                            "battery":{
                                "timestamp":34535454
                            }
                        }
                    }
                },
                "version":10,
                "timestamp":34535454
            },
            "connectivity": {
                "connected":true,
                "timestamp":1556649855046
            }
        }],
        "nextToken":"AQFCuvk7zZ3D9pOYMbFCeHbdZ+h=G"
}
```

在 JSON 响应中，"connectivity"（由 thingConnectivityIndexingMode=STATUS 设置启用）提供了一个布尔值、时间戳和 disconnectReason（用于指示设备是否连接到 AWS IoT Core）。出于 CONNECTION_LOST 原因，设备 "mything1" 在 POSIX 时间 1556649874716 断开连接 (false)。有关断开原因的更多信息，请参阅生命周期事件。

```
"connectivity": {
    "connected":false,
    "timestamp":1556649874716,
    "disconnectReason": "CONNECTION_LOST"
}
```

设备 "mything2" 在 POSIX 时间 1556649855046 连接 (true)：

```
"connectivity": {
    "connected":true,
    "timestamp":1556649855046
}
```

时间戳以自纪元以来的毫秒给出，因此 1556649855046 表示 2019 年 4 月 30 日星期二上午 6:44:15.046 (UTC)。

> **Important**
>
> 如果设备已断开连接大约一小时，则连接状态的 "timestamp" 值和 "disconnectReason" 值可能会缺失。

限制和局限性

AWS_Things 具有这些限制。

具有复杂类型的影子字段

只有字段的值是简单类型、不包含数组的 JSON 对象或完全由简单类型组成的数组时，才会对影子字段编制索引。“简单类型”是指字符串、数字或文本 true 或 false 之一。例如，如果是以下影子状态，则系统将不会编制索引字段 "palette" 的值，因为它是一个包含复杂类型项目的数组。字段 "colors" 的值将会编制索引，因为该数组中的每个值都是一个字符串。

```
{
    "state": {
        "reported": {
            "switched": "ON",
            "colors": [ "RED", "GREEN", "BLUE" ],
            "palette": [
                {
                    "name": "RED",
                    "intensity": 124
                },
                {
                    "name": "GREEN",
                    "intensity": 68
                },
                {
                    "name": "BLUE",
                    "intensity": 201
                }
            ]
        }
    }
}
```

嵌套影子字段名称

嵌套影子字段的名称存储为以句点 (.) 分隔的字符串。例如，假设存在以下影子文档：

```
{
  "state": {
    "desired": {
      "one": {
        "two": {
          "three": "v2"
        }
      }
    }
  }
}
```

字段 three 的名称存储为 desired.one.two.three。如果您还有影子文档，存储方式如下：

```
{
  "state": {
    "desired": {
      "one.two.three": "v2"
    }
  }
}
```

都匹配 shadow.desired.one.two.three:v2 的查询。最佳实践是不要在影子字段名称中使用句点。

影子元数据

影子的元数据部分中的字段编制了索引，但仅当影子的 "state" 部分中的相应字段编制了索引时才会出现这种情况。（在之前的示例中，影子的元数据部分中的 "palette" 字段也不会编制索引。）

未注册的设备

实例集索引对设备的连接状态编制索引，该设备的连接 clientId 与注册表中已注册事物的 thingName 相同。

未注册的影子

如果您使用 UpdateThingShadow 使用尚未在 AWS IoT 账户中注册的事物名称创建影子，则不会为该影子中的字段编制索引。这适用于经典的未命名影子和命名影子。

数字值

如果任何注册表或影子数据被服务识别为数值，则会对其照此编制索引。您可以针对数字值创建涉及范围和比较运算符的查询（例如 "attribute.foo<5" 或 "shadow.reported.foo:[75 TO 80]"）。要识别为数字，数据的值必须是有效的文字类型 JSON 数字。该值可以是2^53...2^53-1 范围内的整数，具有可选指数表示法的双精度浮点，或仅包含这些值的数组的一部分。

Null 值

未对 Null 值建立索引。

最大值

聚合查询的最大自定义字段数是 5。

聚合查询请求的最大百分位数是 100。

授权

您可以在 AWS IoT 策略操作中将事物索引指定为 Amazon 资源名称 (ARN)，如下所示。

操作	资源
iot:SearchIndex	索引 ARN（例如，arn:aws:iot: *your-aws-region your-aws-account* :index/AWS_Things ）。
iot:DescribeIndex	索引 ARN（例如，arn:aws:iot: *your-aws-region* :index/AWS_Things ）。

> **Note**
>
> 如果您有权查询机群索引，可以访问整个机群的事物数据。

管理事物组索引

AWS_ThingGroups是包含您的所有事物组和账单组的索引。您可以使用此索引基于组名称、描述、属性和所有父组名称搜索事物组。

启用事物组索引

您可以使用UpdateIndexing配置 API 中的thing-group-indexing-configuration设置来创建AWS_ThingGroups索引并控制其配置。您可以使用GetIndexing配置 API 来检索当前的索引配置。

要更新对象组索引配置，请运行 update-indexing-configuration CLI 命令：

```
aws iot update-indexing-configuration --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

您还可以在单个命令中更新事物和事物组索引的配置，如下所示：

```
aws iot update-indexing-configuration --thing-indexing-configuration
 thingIndexingMode=REGISTRY --thing-group-indexing-configuration
 thingGroupIndexingMode=ON
```

thingGroupIndexingMode 的有效值如下所示。

关闭

无索引/删除索引。

开

创建或配置 AWS_ThingGroups 索引。

要检索当前的事物和事物组索引配置，运行 get-indexing-configuration CLI 命令。

```
aws iot get-indexing-configuration
```

命令的响应如下所示：

```
{
   "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "ON"
    }
}
```

描述组索引

要检索 AWS_ThingGroups 索引的当前状态，使用describe-indexCLI 命令。

```
aws iot describe-index --index-name "AWS_ThingGroups"
```

命令的响应如下所示：

```
{
   "indexStatus": "ACTIVE",
   "indexName": "AWS_ThingGroups",
   "schema": "THING_GROUPS"
}
```

AWS IoT 首次建立索引时会生成索引。如果 indexStatus 为 BUILDING，则您无法查询索引。

查询事物组索引

要查询索引中的数据，使用 search-index CLI 命令：

```
aws iot search-index --index-name "AWS_ThingGroups" --query-string
 "thingGroupName:mythinggroup*"
```

授权

您可以在 AWS IoT 策略操作中将事物组索引指定为资源 ARN，如下所示。

操作	资源
iot:SearchIndex	索引 ARN（例如，arn:aws:iot: *your-aws-region* :index/AWS_ThingGroups ）。
iot:DescribeIndex	索引 ARN（例如，arn:aws:iot: *your-aws-region* :index/AWS_ThingGroups ）。

查询聚合数据

AWS IoT 提供四个 API（GetStatisticsGetCardinalityGetPercentiles、、和GetBucketsAggregation），允许您在设备群中搜索聚合数据。

> **Note**
>
> 有关聚合API缺少或意外值的问题，请参阅机群索引故障排查指南。

GetStatistics

GetStatisticsAPI 和 get-statistics CLI 命令返回指定聚合字段的计数、平均值、总和、最小值、最大值、平方和、方差和标准差。

get-statistics CLI 命令使用以下参数：

`index-name`

要搜索的索引的名称。默认值为 `AWS_Things`。

`query-string`

用于搜索索引的查询。您可以指定"*"获取您 AWS 账户中所有已编入索引的内容的数量。

`aggregationField`

（可选）要聚合的字段。此字段必须是在调用 update-indexing-configuration 时定义的托管字段或自定义字段。如果不指定聚合字段，则 `registry.version` 用作聚合字段。

`query-version`

要使用的查询的版本。默认值为 `2017-09-30`。

聚合字段的类型可能会影响返回的统计信息。

GetStatistics 使用字符串值

如果在字符串字段上进行聚合，则调用 `GetStatistics` 会返回具有与查询匹配的属性的设备计数。例如：

```
aws iot get-statistics --aggregation-field 'attributes.stringAttribute'
            --query-string '*'
```

此命令会返回包含 `stringAttribute` 属性的设备数量：

```
{
```

```
  "statistics": {
    "count": 3
  }
}
```

GetStatistics 使用布尔值

当您使用布尔聚合字段调GetStatistics用时：

- AVERAGE 是匹配查询的设备的百分比。
- 根据以下规则，MINIMUM 为 0 或 1：
  - 如果聚合字段的所有值均为 `false`，则 MINIMUM 为 0。
  - 如果聚合字段的所有值均为 `true`，则 MINIMUM 为 1。
  - 如果聚合字段的值既有 `false` 又有 `true`，则 MINIMUM 为 0。
- 根据以下规则，MAXIMUM 为 0 或 1：
  - 如果聚合字段的所有值均为 `false`，则 MAXIMUM 为 0。
  - 如果聚合字段的所有值均为 `true`，则 MAXIMUM 为 1。
  - 如果聚合字段的值既有 `false` 又有 `true`，则 MAXIMUM 为 1。
- SUM 是等效于布尔值的整数之和。
- COUNT 是符合查询字符串条件且包含有效聚合字段值的内容计数。

GetStatistics 用数值

调用 `GetStatistics` 并指定 `Number` 类型的聚合字段时，`GetStatistics` 返回以下值：

count

与查询字符串条件匹配并包含有效聚合字段值的事物的计数。

average

与查询匹配的数值的平均值。

sum

与查询匹配的数值的总和。

minimum

与查询匹配的数值中的最小值。

maximum

与查询匹配的数值中的最大值。

总和 OfSquares

与查询匹配的数值的平方和。

variance

与查询匹配的数值的方差。一组值的方差是每个值与该组值平均值之间差值的平方的平均值。

stdDeviation

与查询匹配的数值的标准差。一组值的标准差用于衡量值的分布程度。

以下示例演示了如何使用数值自定义字段调用 get-statistics。

```
aws iot get-statistics --aggregation-field 'attributes.numericAttribute2'
            --query-string '*'
```

```
{
  "statistics": {
    "count": 3,
    "average": 33.333333333333336,
    "sum": 100.0,
    "minimum": -125.0,
    "maximum": 150.0,
    "sumOfSquares": 43750.0,
    "variance": 13472.22222222222,
    "stdDeviation": 116.06990230986766
  }
}
```

对于数值聚合字段，如果字段值超过最大双精度值，则统计值为空。

GetCardinality

GetCardinalityAPI 和 get-cardinality CLI 命令返回与查询相匹配的唯一值的大致数量。例如，您可能需要查找电池电量低于 50% 的设备数量：

```
aws iot get-cardinality --index-name AWS_Things --query-string "batterylevel
```

```
          > 50" --aggregation-field "shadow.reported.batterylevel"
```

此命令返回电池电量超过 50% 的事物数量：

```
{
    "cardinality": 100
}
```

即使没有匹配字段，get-cardinality 也始终返回 `cardinality`。例如：

```
aws iot get-cardinality --query-string "thingName:Non-existent*"
          --aggregation-field "attributes.customField_STR"
```

```
{
    "cardinality": 0
}
```

get-cardinality CLI 命令使用以下参数：

`index-name`

要搜索的索引的名称。默认值为 `AWS_Things`。

`query-string`

用于搜索索引的查询。您可以指定"*"获取您 AWS 账户中所有已编入索引的内容的数量。

`aggregationField`

要聚合的字段。

`query-version`

要使用的查询的版本。默认值为 `2017-09-30`。

GetPercentiles

GetPercentilesAPI 和 get-percentiles CLI 命令将与查询匹配的聚合值分组为百分位分组。默认百分位数分组为：1,5,25,50,75,95,99，不过您可以在调用 `GetPercentiles` 时指定自己的百分位数分组。此函数为指定的每个百分位数组（或默认百分位数分组）返回一个值。百分位数组“1”包含在与查询匹配的值的大约 1% 中出现的聚合字段值。百分位数组“5”包含在与查询匹配的值的大约 5% 中出现的聚合字段值，以此类推。结果是近似值，与查询匹配的值越多，百分位数值就越准确。

以下示例演示了如何调用 get-percentiles CLI 命令。

```
aws iot get-percentiles --query-string "thingName:*" --aggregation-field
          "attributes.customField_NUM" --percents 10 20 30 40 50 60 70 80 90 99
```


```
{
    "percentiles": [
        {
            "value": 3.0,
            "percent": 80.0
        },
        {
            "value": 2.5999999999999996,
            "percent": 70.0
        },
        {
            "value": 3.0,
            "percent": 90.0
        },
        {
            "value": 2.0,
            "percent": 50.0
        },
        {
            "value": 2.0,
            "percent": 60.0
        },
        {
            "value": 1.0,
            "percent": 10.0
        },
        {
            "value": 2.0,
            "percent": 40.0
        },
        {
            "value": 1.0,
            "percent": 20.0
        },
        {
            "value": 1.4,
            "percent": 30.0
        },
```

```
        {
            "value": 3.0,
            "percent": 99.0
        }
    ]
}
```

以下命令显示没有匹配文档时从 get-percentiles 返回的输出。

```
aws iot get-percentiles --query-string "thingName:Non-existent*"
          --aggregation-field "attributes.customField_NUM"
```

```
{
    "percentiles": []
}
```

get-percentile CLI 命令使用以下参数：

`index-name`

要搜索的索引的名称。默认值为 `AWS_Things`。

`query-string`

用于搜索索引的查询。您可以指定"*"获取您 AWS 账户中所有已编入索引的内容的数量。

`aggregationField`

要聚合的字段，必须为 `Number` 类型。

`query-version`

要使用的查询的版本。默认值为 `2017-09-30`。

`percents`

（可选）您可以使用此参数指定自定义百分位数分组。

GetBuckets聚合

A GetBucketsg gregation AP get-buckets-aggregation I 和 CLI 命令会返回存储桶列表以及符合查询字符串条件的事物总数。

以下示例显示如何调用 get-buckets-aggregation CLI 命令。

```
aws iot get-buckets-aggregation --query-string '*' --index-name AWS_Things --
aggregation-field 'shadow.reported.batterylevelpercent' --buckets-aggregation-type
 'termsAggregation={maxBuckets=5}'
```

此命令将返回以下：

```
{
    "totalCount": 20,
    "buckets": [
        {
            "keyValue": "100",
            "count": 12
        },
        {
            "keyValue": "90",
            "count": 5
        },
        {
            "keyValue": "75",
            "count": 3
        }
    ]
}
```

C get-buckets-aggregation LI 命令采用以下参数：

`index-name`

要搜索的索引的名称。默认值为 AWS_Things。

`query-string`

用于搜索索引的查询。您可以指定"*"获取您 AWS 账户中所有已编入索引的内容的数量。

`aggregation-field`

要聚合的字段。

`buckets-aggregation-type`

对响应形状和要执行的存储桶聚合类型的基本控制。

授权

您可以在 AWS IoT 策略操作中将事物组索引指定为资源 ARN，如下所示。

操作	资源
iot:GetStatistics	索引 ARN（例如，arn:aws:iot: *your-aws-region* :index/AWS_Things 或 arn:aws:iot: *your-aws-region* :index/AWS_ThingGroups ）。

查询语法

在机群索引中，您可以使用查询语法来指定查询。

支持的特征

查询语法支持以下特征：

- 术语和短语
- 搜索字段
- 前缀搜索
- 范围搜索
- 布尔运算符 AND、OR、NOT 和 –。使用连字符从搜索结果中排除某事物（例如 thingName:(tv* AND -plasma)）。
- 分组
- 字段分组
- 对特殊字符转义（如，使用 \）

不支持的 特征

查询语法不支持以下特征：

- 前导通配符搜索（例如“*xyz”），搜索“*”将匹配所有事物
- 正则表达式

- 提升
- 排名
- 模糊搜索
- 近似搜索
- 排序
- 聚合
- 特殊字符：`、@、#、%、\、/、'、;和,。请注意，,只有地理查询才支持这一点。

注意

关于查询语言需要注意的几点：

- 默认运算符为 AND。"thingName:abc thingType:xyz" 的查询等同于 "thingName:abc AND thingType:xyz"。
- 如果未指定字段，则会在所有注册表、Device Shadow 和 Device Defender 字段中 AWS IoT 搜索该术语。
- 所有字段名称均区分大小写。
- 搜索不区分大小写。用空格字符分隔单词，如 Java 的 Character.isWhitespace(int) 中所定义的。
- Device Shadow 数据（未命名影子和命名影子）的索引包括“reported”、“desired”、“delta”和“metadata”部分。
- 设备影子和注册表版本不可搜索，但会在响应中提供。
- 查询中字词的数量上限为 12 个。
- ,仅在地理查询中支持特殊字符。

事物查询示例

使用查询语法在查询字符串中指定查询。查询将传递给SearchIndexAPI。下表列出了一些查询字符串示例。

查询字符串	结果
`abc`	在任何注册表、影子（经典未命名影子和命名影子）或 Device Defender 违规字段中查询“abc”。
`thingName:myThingName`	查询名为 “我的ThingName” 的事物。
`thingName:my*`	查询名称以“my”开头的事物。
`thingName:ab?`	查询名称为“ab”加另外一个字符的事物（例如：“aba”、“abb”、“abc”等）。
`thingTypeName:aa`	查询与类型 aa 关联的事物。
`thingGroupNames:a`	查询父事物组名称为“a”的事物。
`thingGroupNames:a*`	查询父事物组名称与模式“a*”匹配的事物。
`attributes.myAttribute:75`	查询属性名为“myAttribute”且属性值为 75 的事物。
`attributes.myAttribute:[75 TO 80]`	查询属性名为“myAttribute”，且属性值在数字范围（75-80，含 75 和 80）之内的事物。
`attributes.myAttribute:{75 TO 80]`	查询属性名为“myAttribute”，且属性值在数字范围（大于 75 且小于等于 80）之内的事物。
`attributes.serialNumber:["abcd" TO "abcf"]`	查询属性名为“serialNumber”，且属性值在字母数字字符串范围之内的事物。此查询将返回属性名为“serialNumber”，且其值为“abcd”、“abce”或“abcf”的事物。
`attributes.myAttribute:i*t`	查询属性名为“myAttribute”，且属性值为以“i”开头、以“t”结尾、中间有任意数量字符的事物。
`attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10`	使用布尔表达式组合术语来查询事物。此查询将返回具有下列特征的事物：属性名为“attr1”且值为“abc”；属性名为“attr2”且值小于 5；以及属性名为“attr3”且值不大于 10。
`shadow.hasDelta:true`	查询具有增量元素的未命名影子的事物。

查询字符串	结果
`NOT attributes.model:legacy`	查询属性名为“model”且不是“legacy”的事物。
`shadow.reported.stats.battery:{70 TO 100} (v2 OR v3) NOT attributes.model:legacy`	查询具有以下特征的事物： • 事物的影子 `stats.battery` 属性具有介于 70 到 100 之间的值。 • 文本“v2”或“v3”出现在事物的名称、类型名称或属性值中。 • 事物的 `model` 属性未设置为“legacy”。
`shadow.reported.myvalues:2`	查询具有以下特征的事物：影子的“reported”部分中的 `myvalues` 数组包含值 2。
`shadow.reported.location:* NOT shadow.desired.stats.battery:*`	查询具有以下特征的事物： • 影子的 `location` 部分中存在 `reported` 属性。 • 影子的 `desired` 部分中不存在 `stats.battery` 属性。
`shadow.name.<shadowName>.hasDelta:true`	查询具有给定名称和增量元素影子的东西。
`shadow.name.<shadowName>.desired.filament:*`	查询具有给定名称和所需灯丝特性的阴影的对象。
`shadow.name.<shadowName>.reported.location:*`	查询具有具有给定命名影子且 `location` 属性存在于命名影子的报告部分中的事物。
`connectivity.connected:true`	查询所有连接的设备。
`connectivity.connected:false`	查询所有断开连接的设备。

查询字符串	结果
`connectivity.connected:true AND connectivity.timestamp : [1557651600000 TO 1557867600000]`	查询所有已连接且连接时间戳 >= 1557651600000 且 <= 1557867600000 的设备。时间戳以自纪元以来的毫秒数给出。
`connectivity.connected:false AND connectivity.timestamp : [1557651600000 TO 1557867600000]`	查询所有已断开连接且断开连接时间戳 >= 1557651600000 且 <= 1557867600000 的设备。时间戳以自纪元以来的毫秒数给出。
`connectivity.connected:true AND connectivity.timestamp > 1557651600000`	查询所有已连接且连接时间戳 > 1557651600000 的设备。时间戳以自纪元以来的毫秒数给出。
`connectivity.connected:*`	查询具有连接信息的所有设备。
`connectivity.disconnectReason:*`	查询具有连接断开原因的所有设备。
`connectivity.disconnectReason:CLIENT_INITIATED_DISCONNECT`	查询由于 CLIENT_INITATE_DISKINATE_DISKINTION 而断开的所有设备
`deviceDefender.violationCount:[0 TO 100]`	查询设备计数值在数字范围（0-100）内的 Device Defender 违规事物。
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.inViolation:true`	查询违反安全配置文件 device-SecurityProfile 中定义的行为 disconnectBehavior 的事物。请注意，inViolation:false 不是有效的查询。
`deviceDefender.<device-SecurityProfile>.disconnectBehavior.lastViolationValue.number>2`	查询与安全配置文件设备中disconnectBehavior 定义的行为相违规的事情—— SecurityProfile 上次违规事件值大于 2。

查询字符串	结果
deviceDefender.<device-SecurityProfile>.disconnectBehavior.lastViolationTime>1634227200000	查询SecurityProfile 与安全配置文件设备中定义的行为disconnectBehavior 相违规的事情——最后一次违规事件是在指定的 epoch 时间之后发生的。
shadow.name.gps-tracker.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km	查询距离坐标 47.6204、-122.3491 的径向距离在 15.5 千米以内的事物。此查询字符串适用于您的位置数据存储在命名影子中的情况。
shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km	查询距离坐标 47.6204、-122.3491 的径向距离在 15.5 千米以内的事物。此查询字符串适用于您的位置数据存储在经典阴影中的情况。

事物组查询示例

系统在查询字符串中使用查询语法指定查询，并将其传送至 SearchIndex API。下表列出了一些查询字符串示例。

查询字符串	结果
abc	在所有字段中查询“abc”。
thingGroupName:myGroupThingName	查询名为 “我的名GroupThing字” 的事物组。
thingGroupName:my*	查询名称以“my”开头的事物组。
thingGroupName:ab?	查询名称为“ab”加另外一个字符的事物组（例如：“aba”、“abb”、“abc”等。）
attributes.myAttribute:75	查询属性名为“myAttribute”且属性值为 75 的事物组。

查询字符串	结果
`attributes.myAttribute:[75 TO 80]`	查询属性名为“myAttribute”，且属性值在数字范围（75-80，含 75 和 80）之内的事物组。
`attributes.myAttribute:[75 TO 80]`	查询属性名为“myAttribute”，且属性值在数字范围（大于 75 且小于等于 80）之内的事物组。
`attributes.myAttribute: ["abcd" TO "abcf"]`	查询属性名为“myAttribute”，且属性值在字母数字字符串范围之内的事物组。此查询将返回属性名为“serialNumber”，且其值为“abcd”、“abce”或“abcf”的事物组。
`attributes.myAttribute:i*t`	查询属性名为“myAttribute”，且其值为以“i”开头、以“t”结尾、中间有任意数量字符的事物组。
`attributes.attr1:abc AND attributes.attr2<5 NOT attributes.attr3>10`	使用布尔表达式组合术语来查询事物组。此查询将返回具有下列特征的事物组：属性名为“attr1”且值为“abc”；属性名为“attr2”且值小于 5；以及属性名为“attr3”且值不大于 10。
`NOT attributes.myAttribute:cde`	查询属性名为“myAttribute”且不是“cde”的事物组。
`parentGroupNames:(myParentThingGroupName)`	查询父组名称与 “我的 ParentThingGroupName” 匹配的事物组。
`parentGroupNames:(myParentThingGroupName OR myRootThingGroupName)`	查询父组名称与 “我的” 或 “我ParentThingGroupName的 RootThingGroupName” 匹配的事物组。
`parentGroupNames:(myParentThingGroupNa*)`	查询父组名称以 “我的 ParentThingGroupNa” 开头的事物组。

索引位置数据

您可以使用AWS IoT 队列索引将设备上次发送的位置数据编入索引，并使用地理查询搜索设备。此功能解决了设备监控和管理用例，例如位置跟踪和邻近搜索。位置索引的工作原理与其他队列索引功能类似，并且需要在事物索引中指定其他配置。

常见用例包括：搜索和聚合位于所需地理边界内的设备，使用索引数据源中与设备元数据和状态相关的查询词获取特定位置的见解，提供精细视图，例如筛选特定地理区域的结果，以减少车队监控地图中的渲染延迟，跟踪上次报告的设备位置，识别超出所需边界限制的设备并使用队列指标生成警报。要开始使用位置索引和地理查询，请参阅???。

支持的数据格式

AWS IoT 舰队索引支持以下位置数据格式：

1. 坐标参考系的众所周知的文本表示

   遵循地理信息-众所周知的坐标参考系文本表示格式的字符串。一个例子可以是"POINT(long lat)"。

2. 一个表示坐标的字符串

   格式为"latitude, longitude"或的字符串"longitude, latitude"。如果使用"longitude, latitude"，则还必须在order中指定geoLocations。一个例子可以是"41.12,-71.34"。

3. 由纬度（纬度）、经度（经度）键组成的对象

   此格式适用于经典阴影和命名阴影。支持的密钥：lat、latitude、lon、long、longitude。一个例子可以是{"lat": 41.12, "lon": -71.34}。

4. 表示坐标的数组

   格式为[lat,lon]或的数组[lon,lat]。如果您使用的格式[lon,lat]与 GeoJSON 中的坐标相同（适用于经典阴影和命名阴影），则还必须在中指定order。geoLocations

   一个例子可以是：

```
{
  "location": {
    "coordinates": [
      **Longitude**,
      **Latitude**
    ],
    "type": "Point",
    "properties": {
      "country": "United States",
      "city": "New York",
```

```
      "postalCode": "*****",
      "horizontalAccuracy": 20,
      "horizontalConfidenceLevel": 0.67,
      "state": "New York",
      "timestamp": "2023-01-04T20:59:13.024Z"
    }
  }
}
```

如何为位置数据编制索引

以下步骤说明如何更新位置数据的索引配置以及如何使用地理查询来搜索设备。

1. 知道您的位置数据存储在哪里

   舰队索引目前支持索引存储在经典阴影或命名阴影中的位置数据。

2. 使用支持的位置数据格式

   确保您的位置数据格式遵循支持的数据格式之一。

3. 更新索引配置

   只需最低限度，即可启用事物（注册表）索引配置。您还必须对包含您的位置数据的经典阴影或命名阴影启用索引。更新事物索引时，应在索引配置中包含位置数据。

4. 创建和运行地理查询

   根据您的用例，创建地理查询并运行它们来搜索设备。您编写的地理查询必须遵循查询语法。您可以在 ??? 中找到一些示例。

更新事物索引配置

要将位置数据编入索引，您必须更新索引配置并包含您的位置数据。根据您的位置数据的存储位置，请按照以下步骤更新您的索引配置：

存储在经典阴影中的位置数据

如果您的位置数据存储在经典阴影中，则必须`thingIndexingMode`将其设置为，`REGISTRY_AND_SHADOW`并在中的`geoLocations`字段（`name`和`order`）中指定您的位置数据`filter`。

在以下事物索引配置示例中，您可以将位置数据路径shadow.reported.coordinates指定LonLat为order。name

```
{
 "thingIndexingMode": "REGISTRY_AND_SHADOW",
 "filter": {
  "geoLocations": [
   {
    "name": "shadow.reported.coordinates",
    "order": "LonLat"
   }
  ]
 }
}
```

- thingIndexingMode

  索引模式控制是否对注册表或影子进行索引。如果设置thingIndexingMode为OFF，则禁用事物索引。

  要索引存储在经典阴影中的位置数据，必须thingIndexingMode将其设置为REGISTRY_AND_SHADOW。有关更多信息，请参阅 ???。

- filter

  索引过滤器为命名阴影和地理定位数据提供了其他选择。有关更多信息，请参阅 ???。

- geoLocations

  您选择要编制索引的地理定位目标列表。用于索引的地理定位目标的默认最大数量为1。要提高限制，请参阅AWS IoT Device Management 配额。

- name

  地理定位目标字段的名称。的示例值name可以是阴影的位置数据路径:shadow.reported.coordinates.

- order

  地理定位目标字段的顺序。有效值：LatLon和LonLat。 LatLon表示纬度和经度。 LonLat表示经度和纬度。该字段是可选的。默认值为 LatLon。

存储在命名阴影中的位置数据

如果您的位置数据存储在命名的阴影中，请namedShadowIndexingMode将其设置为ON，将您命名的影子名称添加到中的namedShadowNames字段中 filter，并在中的geoLocations字段中指定您的位置数据路径filter。

在以下事物索引配置示例中，您可以将位置数据路径shadow.name.namedShadow1.reported.coordinates指定LonLat为order。name

```
{
 "thingIndexingMode": "REGISTRY",
 "namedShadowIndexingMode": "ON",
 "filter": {
  "namedShadowNames": [
   "namedShadow1"
  ],
  "geoLocations": [
   {
    "name": "shadow.name.namedShadow1.reported.coordinates",
    "order": "LonLat"
   }
  ]
 }
}
```

- thingIndexingMode

  索引模式控制是否对注册表或影子进行索引。如果设置thingIndexingMode为OFF，则禁用事物索引。

  要索引存储在命名影子中的位置数据，thingIndexingMode必须设置为REGISTRY（或REGISTRY_AND_SHADOW）。有关更多信息，请参阅 ???。

- filter

  索引过滤器为命名阴影和地理定位数据提供了其他选择。有关更多信息，请参阅 ???。

- geoLocations

  您选择要编制索引的地理定位目标列表。用于索引的地理定位目标的默认最大数量为1。要提高限制，请参阅AWS IoT Device Management 配额。

- name

地理定位目标字段的名称。的示例值name可以是阴影的位置数据路径:shadow.name.namedShadow1.reported.coordinates.

- order

  地理定位目标字段的顺序。有效值：LatLon和LonLat。 LatLon表示纬度和经度。 LonLat表示经度和纬度。该字段是可选的。默认值为 LatLon。

地理查询示例

完成位置数据的索引配置后，运行地理查询来搜索设备。您也可以将地理查询与其他查询字符串合并。有关更多信息，请参阅 ??? 和 ???。

查询示例 1

此示例假设位置数据存储在命名的阴影中gps-tracker。此命令的输出是距离中心点（47.6204，-122.3491）在径向距离内 15.5 千米的设备列表。

```
aws iot search-index --query-string \
"shadow.name.gps-tracker.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

查询示例 2

此示例假设位置数据存储在经典阴影中。此命令的输出是距离中心点（47.6204，-122.3491）在径向距离内 15.5 千米的设备列表。

```
aws iot search-index --query-string \
"shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

查询示例 3

此示例假设位置数据存储在经典阴影中。此命令的输出是未连接且距离中心点（47.6204，-122.3491）的径向距离之外的设备列表。

```
aws iot search-index --query-string \
"connectivity.connected:false AND (NOT
 shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km)"
```

入门教程

本教程演示了如何使用舰队索引来索引您的位置数据。为简单起见，您可以创建一个代表您的设备的事物并将位置数据存储在命名的阴影中，更新位置索引的事物索引配置，并运行示例地理查询来搜索径向边界内的设备。

完成本教程需要大约 15 分钟。

本主题内容：

- 先决条件
- 创建事物和阴影
- 更新事物索引配置
- 运行地理查询

先决条件

- 安装最新版本的AWS CLI。
- 熟悉位置索引和地理查询、管理事物索引和查询语法。

创建事物和阴影

您可以创建一个代表您的设备的事物，以及一个命名的影子来存储其位置数据（坐标 47.61564，-122.33584）。

1. 运行以下命令来创建代表你的自行车的东西，名为 Bike-1。有关如何使用创建事物的更多信息 AWS CLI，请参阅通过参考创建事物AWS CLI。

```
aws iot create-thing --thing-name "Bike-1" \
--attribute-payload '{"attributes": {"model":"OEM-2302-12", "battery":"35",
 "acqDate":"06/09/23"}}'
```

此命令的输出可能如下所示：

```
{
    "thingName": "Bike-1",
    "thingArn": "arn:aws:iot:us-east-1:123456789012:thing/Bike-1",
    "thingId": "df9cf01d-b0c8-48fe-a2e2-e16cff6b23df"
```

```
}
```

2. 运行以下命令创建一个命名的阴影来存储 Bike-1 的位置数据（坐标 47.61564，-122.33584）。有关如何使用 AWS CLI创建命名阴影的更多信息，请参阅 Reference 中的 update-thing-shadow。AWS CLI

```
aws iot-data update-thing-shadow \
--thing-name Bike-1 \
--shadow-name Bike1-shadow \
--cli-binary-format raw-in-base64-out \
--payload '{"state":{"reported":{"coordinates":{"lat": 47.6153, "lon": -122.3333}}}}'
 \
"output.txt" \
```

此命令不会生成任何输出。要查看你创建的命名阴影，你可以运行 list-named-shadows-for-thing CLI 命令。

```
aws iot-data list-named-shadows-for-thing --thing-name Bike-1
```

此命令的输出可能如下所示：

```
{
    "results": [
        "Bike1-shadow"
    ],
    "timestamp": 1699574309
}
```

更新事物索引配置

要索引您的位置数据，您必须更新您的事物索引配置以包含位置数据。由于您的位置数据存储在本教程中的命名阴影中，因此`thingIndexingMode`请将其设置为`REGISTRY`（最低要求），设置为`namedShadowIndexingMode`ON，然后将您的位置数据添加到配置中。在此示例中，您必须将命名阴影的名称和阴影的位置数据路径添加到`filter`。

1. 运行命令以更新位置索引的索引配置。

```
aws iot update-indexing-configuration --cli-input-json '{
"thingIndexingConfiguration": { "thingIndexingMode": "REGISTRY",
```

```
"thingConnectivityIndexingMode": "OFF",
"deviceDefenderIndexingMode": "OFF",
"namedShadowIndexingMode": "ON",
"filter": {
    "namedShadowNames": ["Bike1-shadow"],
    "geoLocations":[{
        "name":"shadow.name.Bike1-shadow.reported.coordinates"
    }]
},
"customFields": [
{ "name":"attributes.battery",
"type":"Number"}] } }'
```

命令不会生成任何输出。您可能需要稍等片刻，直到更新完成。要检查状态，请运行 desc ribe-index CLI 命令。如果你看到 sh indexStatus ows:ACTIVE，则你的事物索引更新已完成。

2. 运行命令以验证您的索引配置。此为可选步骤。

```
aws iot get-indexing-configuration
```

输出可能如下所示：

```
{
    "thingIndexingConfiguration": {
        "thingIndexingMode": "REGISTRY",
        "thingConnectivityIndexingMode": "OFF",
        "deviceDefenderIndexingMode": "OFF",
        "namedShadowIndexingMode": "ON",
        "managedFields": [
            {
                "name": "shadow.name.*.hasDelta",
                "type": "Boolean"
            },
            {
                "name": "registry.version",
                "type": "Number"
            },
            {
                "name": "registry.thingTypeName",
                "type": "String"
            },
            {
                "name": "registry.thingGroupNames",
```

```
                "type": "String"
            },
            {
                "name": "shadow.name.*.version",
                "type": "Number"
            },
            {
                "name": "thingName",
                "type": "String"
            },
            {
                "name": "thingId",
                "type": "String"
            }
        ],
        "customFields": [
            {
                "name": "attributes.battery",
                "type": "Number"
            }
        ],
        "filter": {
            "namedShadowNames": [
                "Bike1-shadow"
            ],
            "geoLocations": [
                {
                    "name": "shadow.name.Bike1-shadow.reported.coordinates",
                    "order": "LatLon"
                }
            ]
        }
    },
    "thingGroupIndexingConfiguration": {
        "thingGroupIndexingMode": "OFF"
    }
}
```

运行地理查询

现在，您已经更新了事物索引配置以包含位置数据。尝试创建一些地理查询并运行它们，看看能否获得所需的搜索结果。地理查询必须遵循查询语法。您可以在中找到一些有用的示例地理查询。???

在以下示例命令中，使用地理查询shadow.name.Bike1-shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km搜索距离中心点 15.5 千米径向距离范围内的设备，坐标为 47.6204，-122.3491）。

```
aws iot search-index --query-string "shadow.name.Bike1-
shadow.reported.coordinates:geo_distance,47.6204,-122.3491,15.5km"
```

由于您的设备位于坐标 “lat”：47.6153，“lon”：-122.3333，位于距离中心点 15.5 km 的距离之内，因此您应该能够在输出中看到此设备（Bike-1）。输出可能如下所示：

```
{
    "things": [
        {
            "thingName": "Bike-1",
            "thingId": "df9cf01d-b0c8-48fe-a2e2-e16cff6b23df",
            "attributes": {
                "acqDate": "06/09/23",
                "battery": "35",
                "model": "OEM-2302-12"
            },
            "shadow": "{\"reported\":{\"coordinates\":{\"lat\":47.6153,\"lon
\":-122.3333}},\"metadata\":{\"reported\":{\"coordinates\":{\"lat\":{\"timestamp
\":1699572906},\"lon\":{\"timestamp\":1699572906}}}},\"hasDelta\":false,\"version\":1}"
        }
    ]
}
```

有关更多信息，请参阅 ???。

机群指标

舰队指标是舰队索引的一项功能，舰队索引是一项托管服务，允许您在其中索引、搜索和聚合设备数据。 AWS IoT您可以使用队列指标来监控您的车队设备CloudWatch在一段时间内的汇总状态，包括查看队列设备的断开连接率或指定时间段内的平均电池电量变化。

使用队列指标，您可以构建聚合查询，其结果会CloudWatch作为指标持续发送到用于分析趋势和创建警报的指标。对于监控任务，您可以指定不同聚合类型的聚合查询（统计数据、基数和百分位)。您可以保存所有聚合查询创建机群指标供将来重复使用。

入门教程

在本教程中，您将创建一个机群指标监控传感器的温度来检测潜在的异常。创建机群指标时，您需要定义检测温度超过 80 华氏度的传感器数量的聚合查询 您可以将查询指定为每 60 秒运行一次，查询结果将发送到其中 CloudWatch，您可以在其中查看存在潜在高温风险的传感器的数量并设置警报。要完成此教程，需要使用 AWS CLI：

在本教程中，您将学习如何：

- 设置
- 创建机群指标
- 在中查看指标 CloudWatch
- 清理资源

完成本教程需要大约 15 分钟。

先决条件

- 安装 AWS CLI 的最新版本
- 让自己熟悉查询聚合数据
- 熟悉 “使用 A mazon” 指标 CloudWatch

设置

要使用实例集指标，请启用实例集索引。要为具有指定数据源和相关配置的事物或事物组启用机群索引，请按照管理事物索引和管理事物组索引中的说明操作。

设置

1. 运行以下命令以启用机群索引并指定要搜索的数据源。

```
aws iot update-indexing-configuration \
--thing-indexing-configuration
 "thingIndexingMode=REGISTRY_AND_SHADOW,customFields=[{name=attributes.temperature,type=Num
{name=attributes.rackId,type=String},
{name=attributes.stateNormal,type=Boolean}],thingConnectivityIndexingMode=STATUS" \
```

上述的 CLI 命令示例启用了实例集索引，以支持使用 AWS_Things 索引搜索注册表数据、影子数据和事物连接状态。

可能需要几分钟才能完成此配置更改。验证在创建实例集指标之前已启用实例集索引。

要检查您的机群索引是否已启用，请运行以下 CLI 命令：

```
aws --region us-east-1 iot describe-index --index-name "AWS_Things"
```

有关更多信息，请参阅启用事物索引。

2. 运行以下 bash 脚本创建十个事物并对其进行描述。

```
# Bash script. Type `bash` before running in other shells.

Temperatures=(70 71 72 73 74 75 47 97 98 99)
Racks=(Rack1 Rack1 Rack2 Rack2 Rack3 Rack4 Rack5 Rack6 Rack6 Rack6)
IsNormal=(true true true true true true false false false false)

for ((i=0; i < 10; i++))
do
  thing=$(aws iot create-thing --thing-name "TempSensor$i" --attribute-
payload attributes="{temperature=${Temperatures[@]:$i:1},rackId=${Racks[@]:
$i:1},stateNormal=${IsNormal[@]:$i:1}}")
  aws iot describe-thing --thing-name "TempSensor$i"
done
```

这个脚本创建了十个事物来表示十个传感器。每个事物都有 temperature、rackId 和 stateNormal 属性如下表所述：

属性	数据类型	描述
temperature	数字	温度（单位“华氏”）
rackId	字符串	包含传感器的服务器机架 ID
stateNormal	布尔值	传感器的温度值是否正常

此脚本的输出包含十个 JSON 文件。其中一个 JSON 文件如下所示：

```
{
    "version": 1,
    "thingName": "TempSensor0",
    "defaultClientId": "TempSensor0",
    "attributes": {
        "rackId": "Rack1",
        "stateNormal": "true",
        "temperature": "70"
    },
    "thingArn": "arn:aws:iot:region:account:thing/TempSensor0",
    "thingId": "example-thing-id"
}
```

有关更多信息，请参阅创建事物。

创建机群指标

要创建机群指标

1. 运行以下命令创建名为*high_temp_FM*的机群指标：您可以创建队列指标来监控温度超过 80 华氏度的传感器的数量。 CloudWatch

```
aws iot create-fleet-metric --metric-name "high_temp_FM" --query-string
 "thingName:TempSensor* AND attributes.temperature >80" --period 60 --aggregation-
field "attributes.temperature" --aggregation-type name=Statistics,values=count
```

指标名称

数据类型：字符串 `--metric-name` 参数指定机群指标名称。在此示例中，您正在创建名为high_temp_FM的机群指标。

--查询-字符串

数据类型：字符串 `--query-string`参数指定查询字符串。在此示例中，查询字符串表示查询名称以华氏度开头TempSensor且温度高于 80 华氏度的所有内容。有关更多信息，请参阅查询语法。

--时期

数据类型：整数 --period参数指定检索聚合数据的时间（以秒为单位）。在此示例中，您指定要创建的机群指标每 60 秒检索一次聚合数据。

--字段-聚合

数据类型：字符串 --aggregation-field 参数指定要评估的属性。在此示例中，要评估温度属性。

--聚合-类型

--aggregation-type 参数指定要在机群指标中显示的统计摘要。对于监控任务，您可以为不同聚合类型自定义聚合查询属性（统计数据、基数和百分位)。在此示例中，您可以为聚合类型指定计数，并指定统计信息以返回属性与查询相匹配的设备计数，换句话说，返回名称以华氏度开头且温度高于 80 华氏度的设备的计数。TempSensor有关更多信息，请参阅查询聚合数据。

此命令的输出如下所示：

```
{
    "metricArn": "arn:aws:iot:region:111122223333:fleetmetric/high_temp_FM",
    "metricName": "high_temp_FM"
}
```

> **Note**
>
> 数据点可能需要一点时间才能显示出来 CloudWatch。

要了解有关如何创建机群指标的更多信息，请参阅管理机群指标。

如果您无法创建机群指标，请阅读机群指标故障排查。

2. （可选）运行以下命令以描述名为 high_temp_FM 的机群指标：

```
aws iot describe-fleet-metric --metric-name "high_temp_FM"
```

此命令的输出如下所示：

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625249775.834,
```

```
    "queryString": "*",
    "period": 60,
    "metricArn": "arn:aws:iot:region:111122223333:fleetmetric/high_temp_FM",
    "aggregationField": "registry.version",
    "version": 1,
    "aggregationType": {
        "values": [
            "count"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625249775.834,
    "metricName": "high_temp_FM"
}
```

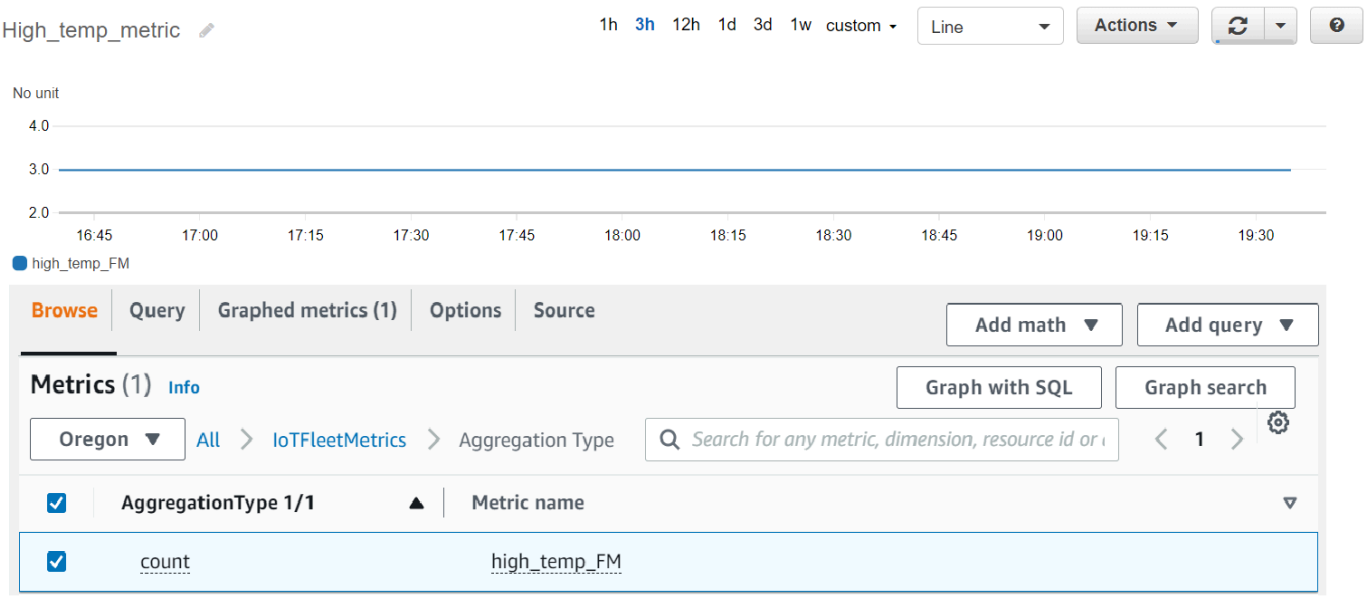
在中查看舰队指标 CloudWatch

创建队列指标后，您可以在中查看指标数据 CloudWatch。在本教程中，您将看到一个指标，该指标显示名称以华氏度开头TempSensor且温度高于 80 华氏度的传感器数量。

要查看中的数据点 CloudWatch

1. 打开 CloudWatch 控制台，网址为 https://console.aws.amazon.com/cloudwatch/。
2. 在左侧面板的 CloudWatch 菜单上，选择指标以展开子菜单，然后选择所有指标。这将打开上半部分的页面显示图表，下半部分包含四个选项卡式部分。
3. 第一个选项卡部分所有指标列出了您可以分组查看的所有指标，请选择 IoT FleetMetrics。这包含您的所有实例集指标。
4. 在所有指标选项卡上的聚合类型部分，选择 Aggregation type（聚合类型）查看您创建的所有机群指标。
5. 在 聚合类型左侧选择机群指标显示图片。您将在指标名称左侧看到值##，这是您在本教程的创建实例集指标部分中指定的聚合类型的值。
6. 选择所有指标选项卡右侧叫做图表化指标的第二个选项卡，查看从上一步中选择的机群指标。

   您能够看到一张图表，显示温度高于 80 华氏度的传感器数量，如下所示：



> **Note**
>
> “周期” 属性 CloudWatch 默认为 5 分钟。这是中显示的数据点之间的时间间隔 CloudWatch。您可以根据您的需求更改时期设置。

7. （可选）您可以设置指标告警。

   1. 在左侧面板的 CloudWatch 菜单上，选择警报以展开子菜单，然后选择所有警报。
   2. 在 Alarms（告警）页面上，在右上角选择 Create alarm（创建告警）。按照控制台中 Create alarm（创建告警）的说明根据需要创建警报。有关更多信息，请参阅使用 Amazon CloudWatch 警报。

要了解更多信息，请阅读使用 Amazon CloudWatch 指标。

如果您在中看不到数据点 CloudWatch，请阅读故障排除队列指标。

清理

要删除机群指标

您使用 delete-fleet-metric CLI 命令删除机群指标。

要删除名为 high_temp_FM 的实例集指标，请运行以下命令。

```
aws iot delete-fleet-metric --metric-name "high_temp_FM"
```

要清除事物

您可以使用 delete-thing CLI 命令删除事物。

要删除您创建的十个事物，请运行以下脚本：

```
# Bash script. Type `bash` before running in other shells.

for ((i=0; i < 10; i++))
do
  thing=$(aws iot delete-thing --thing-name "TempSensor$i")
done
```

要清理中的指标 CloudWatch

CloudWatch 不支持删除指标。指标根据保留时间表过期。要了解更多信息，请参阅使用 Amazon CloudWatch 指标。

管理机群指标

本主题介绍如何使用 AWS IoT 控制台和管理 AWS CLI 您的队列指标。

主题

- 管理机群指标 (控制台)
- 管理机群指标 (CLI)
- 授权对 IoT 资源进行标记

管理机群指标 (控制台)

以下各节介绍如何使用 AWS IoT 控制台管理您的队列指标。在创建机群指标之前，请确保您已使用关联的数据源和配置启用了机群索引。

启用机群索引

如果您已经启用了机群索引，请跳过此部分。

如果还没有启用机群索引，请按照以下说明操作。

1. 通过 https://console.aws.amazon.com/iot/ 打开你的 AWS IoT 主机。
2. 在 AWS IoT 菜单上，选择设置。

3. 要查看详细设置，请在设置页面，向下滚动到机群索引部分。
4. 要更新您的机群索引设置，请在机群索引部分右侧选择 Manage indexing （管理索引）。
5. 在管理机群索引页面上，根据您的需求更新机群索引设置。
   - 配置

     要打开事物索引，请开启事物索引，然后选择要从中索引的数据源。

     要启用事物组索引，请打开事物组索引。
   - Custom fields for aggregation - optional（聚合的自定义字段 - 可选）

     自定义字段是字段名称和字段类型对的列表。

     要添加自定义字段对，请选择 Add new field（添加新字段）。输入自定义字段名称，如 `attributes.temperature`，然后从字段类型菜单选择一个字段类型。请注意，自定义以 `attributes.` 开头的字段名称并将保存为属性来运行事物聚合查询。

     要更新和保存设置，请选择 Update（更新）。

创建机群指标

1. 通过 https://console.aws.amazon.com/iot/ 打开你的 AWS IoT 主机。
2. 在 AWS IoT 菜单上，选择管理，然后选择队列指标。
3. 在机群指标页面上，选择 Create fleet metric（创建机群指标）然后完成创建步骤。
4. 在步骤 1 中配置机群指标
   - 在查询部分中，输入查询字符串以指定要执行聚合搜索的事物或事物组。查询由属性和值组成的字符串。对于属性，选择所需的属性，或，如果该属性没有出现在列表中，则在字段中输入属性。在 : 之后输入值。一个查询字符串示例可以是 `thingName:TempSensor*`。对于输入的每个查询字符串，请按键盘上的输入。如果输入多个查询字符串，请通过选择and、or、and not 或, or or not 指定他们之间的关系。
   - 在报告属性中，从各自的列表中选择 Index name（索引名称）、Aggregation type（聚合类型）和 Aggregation field（聚合字段）。接下来，在选择数据中选择您想要聚合的数据，您可以在其中选择多个数据值。
   - 选择下一步。
5. 在步骤 2 中指定机群指标属性
   - 在机群指标名称字段中，输入要创建机群指标的名称。
   - 在说明-可选在段中，输入要创建机群指标的描述。该字段是可选的。

- 在小时和分钟字段中，输入您希望舰队指标向其发送数据的时间（频率）。 CloudWatch
- 选择下一步。

6. 步骤 3：审核并创建
   - 查看步骤 1 和步骤 2 的设置。要编辑设置，请选择 Edit（编辑）。
   - 选择创建机群指标。

成功创建后，机群指标页面会列出机群指标。

更新机群指标

1. 在实例集指标页面上，选择要更新的实例集指标。
2. 在机群指标 Details(详细信息) 选项卡上，选择 Edit (编辑)。这将打开创建步骤，您可以在这三个步骤中的任何一个步骤中更新机群指标。
3. 完成更新机群指标后，选择 Update fleet metric（更新机群指标）。

删除机群指标

1. 在实例集指标页面上，选择要删除的实例集指标。
2. 在显示机群指标详细信息的下一页上，选择 Delete（删除）。
3. 在对话框中，输入机群指标的名称确认删除。
4. 选择 Delete （删除)。此步骤将永久删除您的机群指标。

管理机群指标 (CLI)

以下各节介绍如何使用 AWS CLI 来管理您的车队指标。在创建机群指标之前，请确保您已使用关联的数据源和配置启用了机群索引。要为您的事物或事物组启用机群索引，请按照管理事物索引或管理事物组索引中的说明操作。

创建机群指标

您可以使用 create-fleet-metric CLI 命令创建队列指标。

```
aws iot create-fleet-metric --metric-name "YourFleetMetricName" --query-string
 "*" --period 60 --aggregation-field "registry.version" --aggregation-type
 name=Statistics,values=sum
```

此命令的输出包含机群指标的名称和 Amazon Resource Name (ARN)。输出内容如下所示：

```
{
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "metricName": "YourFleetMetricName"
}
```

列出机群指标

您可以使用 list-fleet-metric CLI 命令列出您账户中的所有队列指标。

```
aws iot list-fleet-metrics
```

此命令的输出包含所有机群指标。输出内容如下所示：

```
{
    "fleetMetrics": [
        {
            "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/
YourFleetMetric1",
            "metricName": "YourFleetMetric1"
        },
        {
            "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/
YourFleetMetric2",
            "metricName": "YourFleetMetric2"
        }
    ]
}
```

描述机群指标

您可以使用 describe-fleet-metric CLI 命令显示有关队列指标的更多详细信息。

```
aws iot describe-fleet-metric --metric-name "YourFleetMetricName"
```

此命令输出包含有关指定机群指标的详细信息。输出内容如下所示：

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625790642.355,
```

```
    "queryString": "*",
    "period": 60,
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "aggregationField": "registry.version",
    "version": 1,
    "aggregationType": {
        "values": [
            "sum"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625790642.355,
    "metricName": "YourFleetMetricName"
}
```

更新机群指标

您可以使用 update-fleet-metric CLI 命令更新队列指标。

```
aws iot update-fleet-metric --metric-name "YourFleetMetricName" --query-string
 "*" --period 120 --aggregation-field "registry.version" --aggregation-type
 name=Statistics,values=sum,count --index-name AWS_Things
```

该 update-fleet-metric 命令不产生任何输出。您可以使用 C describe-fleet-metric LI 命令查看结果。

```
{
    "queryVersion": "2017-09-30",
    "lastModifiedDate": 1625792300.881,
    "queryString": "*",
    "period": 120,
    "metricArn": "arn:aws:iot:us-east-1:111122223333:fleetmetric/YourFleetMetricName",
    "aggregationField": "registry.version",
    "version": 2,
    "aggregationType": {
        "values": [
            "sum",
            "count"
        ],
        "name": "Statistics"
    },
    "indexName": "AWS_Things",
    "creationDate": 1625792300.881,
```

```
    "metricName": "YourFleetMetricName"
}
```

删除机群指标

使用 C delete-fleet-metric LI 命令删除队列指标。

```
aws iot delete-fleet-metric --metric-name "YourFleetMetricName"
```

如果删除成功或者您指定的机群指标不存在，此命令将不产生任何输出。

有关更多信息，请参阅实例集指标故障排除。

授权对 IoT 资源进行标记

为了更好地控制可以创建、修改或使用的队列指标，您可以为队列指标附加标签。

要标记您使用 AWS Management Console 或创建的队列指标 AWS CLI，您必须在 IAM 策略中包含该`iot:TagResource`操作以授予用户权限。如果您的 IAM 策略不包括`iot:TagResource`，则任何创建带标签的队列指标的操作都将返回`AccessDeniedException`错误。

有关为资源添加标签的一般信息，请参阅为资源添加 AWS IoT 标签。

IAM 策略示例

请参阅以下 IAM 策略示例，在创建队列指标时授予标记权限：

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Action": [
    "iot:TagResource"
   ],
   "Effect": "Allow",
   "Resource": [
    "arn:aws:iot:*:*:fleetmetric/*"
   ]
  },
  {
   "Action": [
    "iot:CreateFleetMetric"
```

```
    ],
    "Effect": "Allow",
    "Resource": [
     "arn:aws:iot:*:*:index/*",
     "arn:aws:iot:*:*:fleetmetric/*"
    ]
   }
  ]
}
```

有关更多信息，请参阅用于 AWS IoT 的操作、资源和条件键。

基于 MQTT 的文件传输

您可以用来管理文件并将其传输到队列中的 AWS IoT 设备的一个选项是基于 MQTT 的文件传输。借助 AWS 云端的此功能，您可以创建包含多个文件的流、更新流数据（文件列表和描述）、获取流数据等。 AWS IoT 基于 MQTT 的文件交付可以使用支持 JSON 或 CBOR 请求和响应消息的 MQTT 协议，将数据以小块形式传输到您的物联网设备。

有关使用向物联网设备传输数据和从物联网设备传输数据的方法的更多信息 AWS IoT，请参阅将设备连接到 AWS IoT。

主题

什么是流？

在中 AWS IoT，流是一种可公开寻址的资源，它是可以传输到物联网设备的文件列表的抽象。通常而言，流包含以下信息：

- Amazon Resource Name (ARN)，在给定时间唯一标识流。此 ARN 具有模式 `arn:partition:iot:region:account-ID:stream/stream ID`。
- 用于识别您的@@ 直播的直播 ID，在 () 或 SDK 命令中使用 AWS Command Line Interface （通常是必需的AWS CLI）。
- 流描述提供对流资源的说明。
- 流版本用来标识流的特定版本。由于流数据可以在设备开启数据传输之前立即修改，因此设备可以使用流版本来强制执行一致性检查。
- 一系列文件，可以传输到设备。对于列表中的每个文件，流都会记录文件 ID、文件大小以及文件的地址信息（例如，Amazon S3 存储桶名称、对象密钥和对象版本）。
- 一个 AWS Identity and Access Management (IAM) 角色，它授予 AWS IoT 基于 MQTT 的文件传输读取存储在数据存储中的流文件的权限。

AWS IoT 基于 MQTT 的文件传输提供以下功能，以便设备可以从云端传输数据： AWS

- 使用 MQTT 协议进行数据传输。
- 支持 JSON 或 CBOR 格式。
- 通过描述流 (`DescribeStream` API) 来获取流文件列表、流版本和相关信息的能力。
- 能够在小数据块中发送数据 (`GetStream` API)，以便具有硬件约束的设备可以接收数据块的能力。
- 支持每个请求的动态数据块大小，以支持具有不同内存容量的设备。
- 当多个设备从同一流文件请求数据块时，对并发流请求的优化。
- Amazon S3 作为流文件的数据存储。
- Support 支持将数据传输日志从 AWS IoT 基于 MQTT 的文件传输到。 CloudWatch

有关基于 MQTT 的文件传输限额，请参阅《AWS 一般参考》中的 AWS IoT Core 服务限额。

在 AWS 云端管理直播

AWS IoT 提供可用于管理 AWS 云端直播的 AWS SDK 和 AWS CLI 命令。您可以使用这些命令来进行以下操作：

- 创建流 CLI / SDK
- 描述流以获取其信息。CLI / SDK
- 列出你中的直播 AWS 账户. CLI / SDK
- 更新流中的文件列表或流描述。CLI / SDK
- 删除流。CLI / SDK

> **Note**
>
> 此时，流在 AWS Management Console中不可见。您必须使用 AWS CLI 或 S AWS DK 来管理中的直播 AWS IoT。此外，嵌入式 C SDK 是唯一支持基于 MQTT 的文件传输功能的 SDK。

在您的设备上使用 AWS IoT 基于 MQTT 的文件传输之前，必须确保您的设备满足以下条件，如下一节所示：

- 反映通过 MQTT 传输数据所需的正确权限的策略。
- 您的设备可以连接到 AWS IoT 设备网关。

- 声明您可以为资源添加标签的策略声明。如果使用标签调用 `CreateStream`，则需要 `iot:TagResource`。

在设备上使用 AWS IoT 基于 MQTT 的文件传输之前，必须按照下一节中的步骤进行操作，以确保您的设备已获得适当的授权并且可以连接到 AWS IoT 设备网关。

向您的设备授予权限

您可以按照创建 AWS IoT 策略中的步骤创建设备策略，或使用现有设备策略。将策略附加到与您的设备关联的证书，并将以下权限添加到设备策略中。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {   "Effect": "Allow",
            "Action": [ "iot:Connect" ],
            "Resource": [
                "arn:partition:iot:region:accountID:client/
${iot:Connection.Thing.ThingName}"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [ "iot:Receive", "iot:Publish" ],
            "Resource": [
                "arn:partition:iot:region:accountID:topic/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:Subscribe",
            "Resource": [
                "arn:partition:iot:region:accountID:topicfilter/$aws/things/
${iot:Connection.Thing.ThingName}/streams/*"
            ]
        }
    ]
}
```

将您的设备连接到 AWS IoT

使用 AWS IoT 基于 MQTT 的文件传输的设备需要连接。 AWS IoT AWS IoT 基于 MQTT 的文件传输与 AWS IoT AWS 云端集成，因此您的设备应直接连接到AWS IoT 数据平面的端点。

> **Note**
>
> AWS IoT 数据平面的端点特定于 AWS 账户 和区域。您必须使用设备注册所在地区的终端节点 AWS IoT。 AWS 账户

请参阅正在连接到 AWS IoT Core了解更多信息。

TagResource 用法

`CreateStream` API 操作将创建一个流，以便通过 MQTT 以分块形式传送一个或多个大文件。

成功的 `CreateStream` API 调用需要以下权限：

- `iot:CreateStream`
- `iot:TagResource`（如果 `CreateStream` 带有标签）

支持这两个权限的策略如下所示：

```
{
 "Version": "2012-10-17",
 "Statement": {
     "Action": [ "iot:CreateStream", "iot:TagResource" ],
     "Effect": "Allow",
     "Resource": "arn:partition:iot:region:accountID:stream/streamId",
 }
}
```

为确保用户在没有适当权限的情况下无法在资源上创建或更新标签，需要 `iot:TagResource` 策略声明操作。没有具体的 `iot:TagResource` 策略声明操作，`CreateStream` API 调用将返回 `AccessDeniedException`（如果请求附带标签）。

有关更多信息，请参阅以下链接：

- CreateStream

- TagResource
- 标签

在设备中使用 AWS IoT 基于 MQTT 的文件传输

要启动数据传输过程，设备必须接收初始数据集，其中至少包含一个流 ID。您可以通过在任务文档中包含初始数据集来使用 任务 计划设备的数据传输任务。当设备收到初始数据集时，它应开始与 AWS IoT 基于 MQTT 的文件传输进行交互。要使用 AWS IoT 基于 MQTT 的文件传输交换数据，设备应该：

- 使用 MQTT 协议订阅 基于 MQTT 的文件传输主题。
- 发送请求，然后使用 MQTT 消息等待接收响应。

您可以选择在初始数据集中包含流文件 ID 和流版本。将流文件 ID 发送到设备可以简化设备固件/软件的编程，因为设备无需发送 DescribeStream 请求来获取此 ID。设备可以在 GetStream 请求中指定流版本以强制执行一致性检查，以防流意外更新。

DescribeStream 用于获取直播数据

AWS IoT 基于 MQTT 的文件传输提供了向设备发送流数据的 DescribeStream API。此 API 返回的流数据包括流 ID、流版本、流描述和流文件列表，每个流文件都有一个文件 ID 和文件大小信息（以字节为单位）。通过这些信息，设备可以选择任意文件来启动数据传输过程。

> **Note**
>
> 如果您的设备在初始数据集中收到了所有必需的流文件 ID，则无需使用 DescribeStream API。

请按照以下步骤发送 DescribeStream 请求。

1. 订阅“已接受”主题筛选条件 $aws/things/*ThingName*/streams/*StreamId*/description/json。
2. 订阅“已被拒绝”主题筛选条件 $aws/things/*ThingName*/streams/*StreamId*/rejected/json。
3. 发送消息到 $aws/things/*ThingName*/streams/*StreamId*/describe/json 以发布 DescribeStream 请求。

4. 如果请求被接受，则您的设备会在“已接受”主题筛选条件中收到 DescribeStream 响应。
5. 如果请求被拒绝，则您的设备会在“已被拒绝”主题筛选条件条件中收到的错误响应。

> **Note**
>
> 如果您在显示的主题和主题筛选条件中将 json 替换为 cbor，则您的设备将收到 CBOR 格式的消息，该格式比 JSON 更紧凑。

DescribeStream 请求

使用 JSON 格式的典型 DescribeStream 请求类似于以下示例。

```
{
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039"
}
```

- （可选）“c”是客户端令牌字段。

  客户端令牌不能超过 64 字节。超过 64 字节的客户端令牌将导致错误响应和 InvalidRequest 错误消息。

DescribeStream 响应

使用 JSON 格式的 DescribeStream 响应类似于以下示例。

```
{
    "c": "ec944cfb-1e3c-49ac-97de-9dc4aaad0039",
    "s": 1,
    "d": "This is the description of stream ABC.",
    "r": [
        {
            "f": 0,
            "z": 131072
        },
        {
            "f": 1,
            "z": 51200
        }
```

```
    ]
}
```

- “c”是客户端令牌字段。如果它在 DescribeStream 请求中给出，则将会返回。使用客户端令牌将响应与其请求相关联。
- “s”是整数形式的流版本。您可以使用此版本对 GetStream 请求执行一致性检查。
- “r”包含流中的文件列表。
  - “f”是整数形式的流文件 ID。
  - “z”是以字节为单位的流文件大小。
- “d”包含流的描述。

从流文件中获取数据块

您可以使用 GetStream API，以便设备可以以小数据块的形式接收流文件，因此可以用于那些对处理大型数据块大小有限制的设备。要接收整个数据文件，设备可能需要发送或接收多个请求和响应，直到接收和处理了所有数据块。

GetStream 请求

请按照以下步骤发送 GetStream 请求。

1. 订阅“已接受”主题筛选条件 $aws/things/*ThingName*/streams/*StreamId*/data/json。
2. 订阅“已被拒绝”主题筛选条件 $aws/things/*ThingName*/streams/*StreamId*/rejected/json。
3. 发布 GetStream 请求到主题 $aws/things/*ThingName*/streams/*StreamId*/get/json。
4. 如果请求被接受，您的设备将在“已接受”主题筛选条件下收到一个或多个 GetStream 响应。每个响应消息都包含单个数据块的基本信息和数据负载。
5. 重复步骤 3 和 4 以接收所有数据块。如果请求的数据量大于 128 KB，则必须重复这些步骤。您必须对您的设备进行编程，以使用多个 GetStream 请求接收所有请求的数据。
6. 如果请求被拒绝，您的设备将在“已被拒绝”主题筛选条件下收到错误响应。

> **Note**
>
> - 如果您在显示的主题和主题筛选条件中将“json”替换为“cbor”，则您的设备将收到 CBOR 格式的消息，这种格式比 JSON 更紧凑。

- AWS IoT 基于 MQTT 的文件传输将数据块的大小限制为 128 KB。如果您对超过 128 KB 的数据块发出请求，请求将失败。
- 您可以对总大小大于 128 KB 的多个数据块发出请求（例如，如果您请求 5 个数据块，各 32 KB，则共有 160 KB 的数据）。在这种情况下，请求不会失败，但您的设备必须发出多个请求才能接收所请求的所有数据。当您的设备提出额外请求时，该服务将发送额外的数据块。我们建议您仅在正确接收和处理之前的响应之后，才继续执行新的请求。
- 无论请求的数据总大小如何，您都应该对设备进行编程，以便在未收到数据块或未正确接收数据块时启动重试。

使用 JSON 格式的典型 GetStream 请求类似于以下示例。

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "s": 1,
    "f": 0,
    "l": 4096,
    "o": 2,
    "n": 100,
    "b": "..."
}
```

- [可选]“c”是客户端令牌字段。

  客户端令牌不能超过 64 字节。超过 64 字节的客户端令牌将导致错误响应和 InvalidRequest 错误消息。

- [可选]“s”是流版本字段（整数）。

  基于 MQTT 的文件传输应用基于此请求的版本和云中的最新流版本的一致性检查。如果从设备发送的 GetStream 请求中的流版本与云中的最新流版本不匹配，则服务会发送错误响应和 VersionMismatch 错误消息。通常，设备会在初始数据集或对 DescribeStream 的响应中接收预期（最新）的流版本。

- “f”是流文件 ID（范围为 0 到 255 的整数）。

  使用或 SDK 创建或更新直播时，需要提供流文件 ID。 AWS CLI 如果设备请求了 ID 不存在的流文件，则该服务将发送错误响应和 ResourceNotFound 错误消息。

- “l”是以字节为单位的数据块大小（范围为 256 到 131072 的整数）。

请参阅 为请求生成位 GetStream 图 以了解如何使用位图字段指定流文件的哪些部分将在 GetStream 响应中返回。如果设备指定的数据块大小超出范围，则服务会发送错误响应和 BlockSizeOutOfBounds 错误消息。

- [可选]“o”是流文件中数据块的偏移量（范围为 0 到 98304 的整数）。

  请参阅 为请求生成位 GetStream 图 以了解如何使用位图字段指定流文件的哪些部分将在 GetStream 响应中返回。最大值 98304 是基于 24 MB 流文件大小限制和 256 字节的最小数据块大小计算得出的。如果未指定，默认值为 0。

- [可选]“n”是请求的数据块数量（范围为 0 到 98304 的整数）。

  “n”字段指定 (1) 请求的数据块数量，或 (2) 使用位图字段 (“b”) 时，位图请求将返回的数据块数量的限制。第二种用法是可选的。如果未定义，则默认为 131072/ *DataBlockSize*。

- [可选]“b”是一个位图，表示正在请求的数据块。

  使用位图，您的设备可以请求非连续的数据块，这使得处理错误后重试的操作更加方便。请参阅 为请求生成位 GetStream 图 了解如何使用位图字段指定流文件的哪一部分将在 GetStream 响应中返回。对于此字段，将位图转换为以十六进制表示法表示位图值的字符串。位图必须小于 12288 个字节。

> **Important**
>
> 应指定 "n" 或 "b"。如果两者都未指定，当文件小于 131072 字节 (128 KB) 时，GetStream 请求可能无效。

GetStream 响应

JSON 格式的 GetStream 响应对于每个请求的数据块而言均如此示例所示。

```
{
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380",
    "f": 0,
    "l": 4096,
    "i": 2,
    "p": "..."
}
```

- “c”是客户端令牌字段。如果它在 GetStream 请求中给出，则将会返回。使用客户端令牌将响应与其请求相关联。
- “f”是当前数据块负载所属的流文件的 ID。
- “l”是数据块负载的大小（以字节为单位）。
- “i”是负载中包含的数据块的 ID。数据块从 0 开始编号。
- “p”包含数据块负载。此字段是一个字符串，表示 Base64 编码中数据块的值。

为请求生成位 GetStream 图

您可以使用 GetStream 请求中的位图字段 (b) 从流文件中获取非连续数据块。这有助于 RAM 容量有限的设备处理网络交付问题。设备只能请求那些未接收或未正确接收的数据块。位图确定将返回流文件的哪些数据块。对于位图中设置为 1 的每个位，将返回相应的流文件数据块。

下面的示例将演示如何在 GetStream 请中指定位图机器支持的字段。例如，您希望以 256 字节的数据块（块大小）接收流文件。将每个 256 字节的数据块视为有一个指定其在文件中位置的数字，从 0 开始。因此，数据块 0 是文件中第一个 256 字节的数据块，数据块 1 是第二个，依此类推。您想要从文件中请求数据块 20、21、24 和 43。

数据块偏移

由于第一个数据块是第 20 号，因此指定偏移量（字段 o）为 20，以节省位图中的空间。

数据块数量

为了确保您的设备收到的数据块数量不会超过其在有限内存资源中所能处理的数量，您可以指定基于 MQTT 的文件传输发送的每条消息中应返回的最大数据块数量。请注意，如果位图本身指定的数据块数量小于此数值，或者如果它会使基于 MQTT 的文件传输发送的响应消息总大小大于每个 GetStream 请求 128 KB 的服务限制，则将忽略此值。

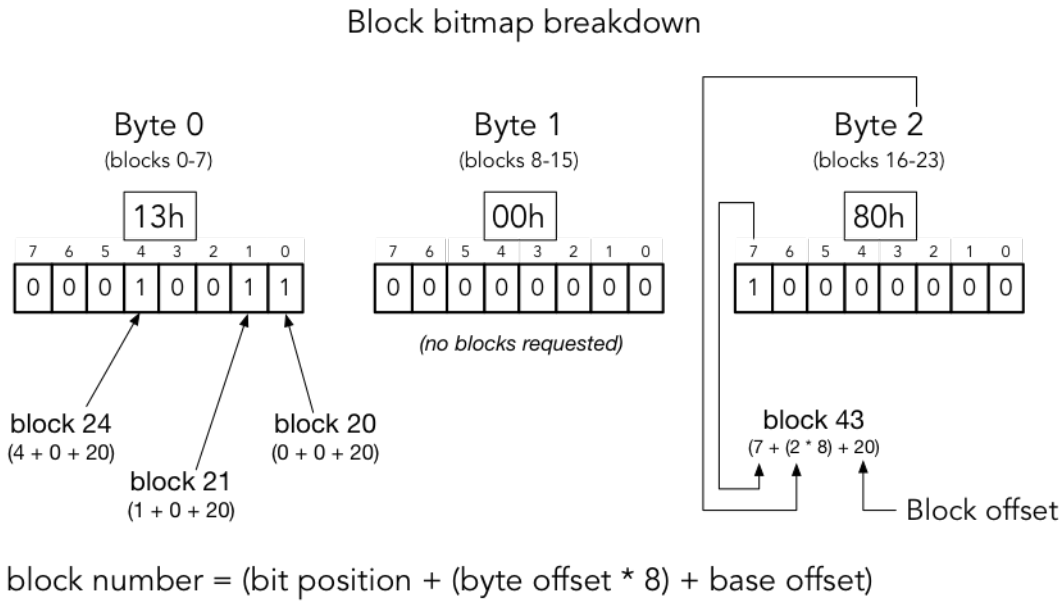
数据块位图

位图本身是一个以十六进制表示法表示的无签名字节数组，并以数字字符串的形式包含在 GetStream 请求中。但要构造这个字符串，我们首先要将位图视为一个长序列的位（二进制数）。如果此序列中的位设置为 1，则流文件中的相应数据块将被发送回设备。在我们的示例中，我们希望接收数据块 20、21、24 和 43，因此我们必须在位图中设置位 20、21、24 和 43。我们可以使用数据块偏移来节省空间，因此在从每个数据块编号中减去偏移后，我们希望设置位 0、1、4 和 23，如下例所示。

```
1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

一次取一个字节（8 位），这个字节通常被写作：“0b00010011”、“0b00000000”和“0b10000000”。位 0 显示在第一个字节末尾的二进制表示中，位 23 显示在最后一个字节的开头。除非您早已了解惯例，否则这种表示方式可能会令人困惑不已。第一个字节包含位 7-0（按该顺序），第二个字节包含位 15-8，第三个字节包含位 23-16，依此类推。在十六进制表示法中，这将转换为“0x130080”。

> **Tip**
>
> 您可以将标准二进制转换为十六进制表示法。一次取四个二进制数字，并将它们转换为十六进制等效数字。例如，“0001”变为“1”，“0011”变成 “3”，依此类推。



将这一切结合起来，我们的 `GetStream` 请求在 JSON 格式中看起来与下方类似。

```
{
    "c" : "1",
    "s" : 1,
    "l" : 256,
    "f" : 1,
    "o" : 20,
    "n" : 32,
    "b" : "130080"
}
```

- “c”是客户端令牌字段。

- “s”是预期的流版本。
- “l”是数据块负载的大小（以字节为单位）。
- “f”是源文件索引的 ID。
- “o”是数据块偏移量。
- “n”是数据块的数量。
- “b”是从偏移量开始缺失的 blockId 位图。此值必须采用 base64 编码。

处理 AWS IoT 基于 MQTT 的文件交付产生的错误

为 DescribeStream 和 GetStream API 发送到设备的错误响应包含客户端令牌、错误代码和错误消息。典型的错误响应类似于以下示例。

```
{
    "o": "BlockSizeOutOfBounds",
    "m": "The block size is out of bounds",
    "c": "1bb8aaa1-5c18-4d21-80c2-0b44fee10380"
}
```

- 如果出现错误，则“o”为指示错误原因的代码。有关更多详细信息，请参阅本部分后面的错误代码。
- “m”是包含错误详细信息的错误消息。
- “c”是客户端令牌字段。如果它在 DescribeStream 请求中提供，则可能会返回。您可以使用客户端令牌将响应与其请求相关联。

  客户端令牌字段并不总是包含在错误响应中。当请求中给出的客户端令牌无效或格式不正确时，它不会在错误响应中返回。

> **Note**
>
> 为了向后兼容，错误响应中的字段可能采用非缩写形式。例如，错误代码可能由“code”或“o”字段表示，客户端令牌字段可以由“clientToken”或“c”字段表示。建议您使用上面显示的缩写形式。

InvalidTopic

流消息的 MQTT 主题无效。

InvalidJson

流请求不是有效的 JSON 文档。

InvalidCbor

流请求不是有效的 CBOR 文档。

InvalidRequest

请求整体被标识为格式错误。有关更多信息，请参阅错误消息。

Unauthorized

请求未获授权访问存储介质（如 Amazon S3）中的流数据文件。有关更多信息，请参阅错误消息。

BlockSizeOutOfBounds

数据块大小超出了界限。请参阅 AWS IoT Core Service Quotas 中的“基于 MQTT 的文件传输”部分。

OffsetOutOfBounds

偏移超出界限。请参阅 AWS IoT Core Service Quotas 中的“基于 MQTT 的文件传输”部分。

BlockCountLimitExceeded

请求数据块的数量超出了界限。请参阅 AWS IoT Core Service Quotas 中的“基于 MQTT 的文件传输”部分。

BlockBitmapLimitExceeded

请求位图的大小超出了界限。请参阅 AWS IoT Core Service Quotas 中的“基于 MQTT 的文件传输”部分。

ResourceNotFound

找不到请求的流、文件、文件版本或数据块。有关更多详细信息，请参阅错误消息。

VersionMismatch

请求中的流版本与基于 MQTT 的文件传输特征中的流版本不匹配。这表示自设备最初接收流版本以来，流数据已被修改。

E TagMismatch

流中的 S3 ETag 与最新 S3 对象版本的 ETag 不匹配。

InternalError

基于 MQTT 的文件传输中发生内部错误。

FreeRTOS OTA 中的一个示例使用案例

FreeRTOS OTA over-the-air () 代理 AWS IoT 使用基于 MQTT 的文件传输将 FreeRTOS 固件映像传输到 FreeRTOS 设备。为了将初始数据集发送到设备，它使用 AWS IoT Job 服务为 FreeRTOS 设备安排一个 OTA 更新任务。

有关基于 MQTT 的文件传输客户端的实施案例参考，请参阅 FreeRTOS 文档中的 FreeRTOS OTA 代理代码。

Device Advisor

Device Advisor 是一种基于云的完全托管式测试功能，用于在设备软件开发过程中验证 IoT 设备。Device Advisor 提供预先构建的测试，在将设备部署到生产环境之前 AWS IoT Core，您可以使用这些测试来验证物联网设备的可靠性和安全连接。Device Advisor 的预构建测试可帮助您根据最佳实践验证您的设备软件，以便使用 TLS、MQTT、Device Shadow 和 IoT Jobs。您还可以下载已签名的资格报告，以提交 AWS 合作伙伴网络，让您的设备符合 AWS 合作伙伴设备目录的要求，而无需将您的设备发送到其中，更无需等待它进行测试。

> **Note**
>
> Device Advisor 在 us-east-1、us-west-2、ap-northeast-1、eu-west-1 区域中获得支持。
> 设备顾问支持使用 MQTT 和 MQTT over S WebSocket ecure (WSS) 协议发布和订阅消息的设备和客户端。所有协议都支持 IPv4 和 IPv6。
> Device Advisor 支持 RSA 服务器证书。

任何专为连接而设计的设备 AWS IoT Core 都可以利用设备顾问。您可以从AWS IoT 控制台或使用 AWS CLI 或 SDK 访问设备顾问。准备好测试您的设备时，请在设备顾问端点注册设备 AWS IoT Core 并配置设备软件。然后选择预构建的测试，加以配置，在您的设备上运行测试，随之获取测试结果以及详细的日志或资格报告。

设备顾问是 AWS 云端的测试端点。若要测试设备，您可以配置设备使其连接到 Device Advisor 提供的测试终端节点。将设备配置为连接到测试端点后，您可以访问设备顾问的控制台或使用 AWS 软件开发工具包选择要在设备上运行的测试。然后，Device Advisor 会管理测试的整个生命周期，包括资源调配、测试过程调度、管理状态机、记录设备行为、记录结果并以测试报告的形式提供最终结果。

TLS 协议

传输层安全性协议（TLS）用于加密互联网等不安全的网络上的机密数据。TLS 协议是安全套接字层（SSL）协议的后继协议。

Device Advisor 支持以下 TLS 协议：

- TLS 1.3（建议）
- TLS 1.2

协议、端口映射和身份验证

设备或客户端使用设备通信协议通过设备端点连接到消息代理。下表列出了 Device Advisor 端点支持的协议以及使用的身份验证方法和端口。

协议、身份验证和端口映射

协议	支持的操作	身份验证	端口	ALPN 协议名称
MQTT 结束了 WebSocket	发布、订阅	Signature Version 4	443	不适用
MQTT	发布、订阅	X.509 客户端证书	8883	`x-amzn-mqtt-ca`
MQTT	发布、订阅	X.509 客户端证书	443	不适用

本章包含以下部分：

- 设置
- 在控制台中开始使用 Device Advisor
- Device Advisor 工作流
- Device Advisor 详细控制台工作流
- 长时间测试控制台的工作流程
- Device Advisor VPC 端点（AWS PrivateLink）
- Device Advisor 测试用例

设置

首次使用 Device Advisor 之前，请完成以下任务：

创建 IoT 事物

首先，创建一个 IoT 事物并为该事物附加证书。有关如何创建事物的教程，请参阅创建事物对象。

创建要用作设备角色的 IAM 角色

> **Note**
>
> 您可以使用 Device Advisor 控制台快速创建设备角色。要了解如何使用 Device Advisor 控制台设置设备角色，请参阅控制台中的 Device Advisor 入门。

1. 转到AWS Identity and Access Management 控制台并登录 AWS 账户 您用于设备顾问测试的。
2. 在左侧导航窗格中，选择 Policies（策略）。
3. 选择 创建策略。
4. 在 Create Policy（创建策略）下，执行以下操作。

   a. 对于 Service (服务)，请选择 IoT。

   b. 在操作下，执行以下操作之一：

      - （建议）根据您在上一节中创建的 IoT 事物或证书所附的策略选择操作。
      - 在筛选操作框中搜索以下操作并选择它们：

        - `Connect`
        - `Publish`
        - `Subscribe`
        - `Receive`
        - `RetainPublish`

   c. 在资源下，限制客户端、主题和主题资源。限制这些资源是一种最佳安全实践。要限制资源，请执行以下操作：

      i. 选择 Specify client resource ARN for the Connect action（为 Connect 操作指定客户端资源 ARN）。

      ii. 选择添加 ARN，然后执行以下任一操作：

      > **Note**
      >
      > clientId 是设备用于与 Device Advisor 交互的 MQTT 客户端 ID。

      - 在可视化 ARN 编辑器中指定区域、账户 ID 和客户端 ID。

- 手动输入要用于运行测试用例的 IoT 主题的 Amazon 资源名称（ARN）。

iii. 选择 添加。

iv. 选择为接收及另一个操作指定主题资源 ARN。

v. 选择添加 ARN，然后执行以下任一操作：

> **Note**
>
> 主题名称是设备向其发布消息的 MQTT 主题。

- 在可视化 ARN 编辑器中指定区域、账户 ID 和主题名称。
- 手动输入要用于运行测试用例的 IoT 主题的 ARN。

vi. 选择 添加。

vii. 选择为订阅操作指定主题筛选条件资源 ARN。

viii. 选择添加 ARN，然后执行以下任一操作：

> **Note**
>
> 主题名称是您的设备订阅的 MQTT 主题。

- 在可视化 ARN 编辑器中指定区域、账户 ID 和主题名称。
- 手动输入要用于运行测试用例的 IoT 主题的 ARN。

ix. 选择 添加。

5. 选择下一步：标签。
6. 选择下一步：审核。
7. 在查看策略下，输入策略的名称。
8. 选择 创建策略。
9. 在左侧导航窗格中，选择 Roles（角色）。
10. 请选择 创建角色。
11. 在选择可信实体下，选择自定义信任策略。
12. 在自定义信任策略框中输入以下信任策略。为防止出现混淆代理人问题，请在策略中添加全局条件键 `aws:SourceArn` 和 `aws:SourceAccount`。

> **Important**
>
> 您的 `aws:SourceArn` 必须与 `format: arn:aws:iotdeviceadvisor:region:account-id:*`. 相符。确保 *region* 与您的 AWS IoT 区域匹配，*account-id* 与您的客户账户 ID 匹配。有关更多信息，请参阅防止跨服务混淆代理。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAwsIoTCoreDeviceAdvisor",
            "Effect": "Allow",
            "Principal": {
                "Service": "iotdeviceadvisor.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn":
 "arn:aws:iotdeviceadvisor:*:111122223333:suitedefinition/*"
                }
            }
        }
    ]
}
```

13. 选择下一步。
14. 选择您在步骤 4 中创建的策略。
15. （可选）在设置权限边界下，选择使用权限边界控制最大角色权限，然后选择您创建的策略。
16. 选择下一步。
17. 输入 Role name（角色名称）和 Role description（角色描述）。
18. 选择 创建角色。

为 IAM 用户创建自定义托管策略来使用设备顾问

1. 通过以下网址导航到 IAM 控制台：https://console.aws.amazon.com/iam/。如果提示，请输入您的 AWS 凭证。
2. 在左侧导航窗格中，选择 Policies（策略）。
3. 选择 Create policy（创建策略），然后选择 JSON 选项卡。
4. 添加必要的权限以使用 Device Advisor。可在安全最佳实践主题中找到策略文档。
5. 选择 Review Policy（查看策略）。
6. 输入名称和描述。
7. 选择创建策略。

创建 IAM 用户来使用 Device Advisor。

> **Note**
>
> 我们建议您创建一个要在运行 Device Advisor 测试时使用的 IAM 用户。由管理员用户运行 Device Advisor 测试可能会带来安全风险，因此建议不要这样做。

1. 导航到 IAM 控制台：https://console.aws.amazon.com/iam/如果出现提示，请输入要登录的 AWS 凭证。
2. 在左侧导航窗格中，选择 Users（用户）。
3. 选择添加用户。
4. 输入用户名称。
5. 如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS Management Console。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

   要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
人力身份	使用临时证书签署向 AWS CLI、 AWS 软件开发工具包	按照您希望使用的界面的说明进行操作。

哪个用户需要编程式访问权限？	目的	方式
（在 IAM Identity Center 中管理的用户）	或 AWS API 发出的编程请求。	• 有关的 AWS CLI，请参阅《AWS Command Line Interface 用户指南》 AWS IAM Identity Center中的“配置 AWS CLI 要使用”。 • 有关 AWS 软件开发工具包、工具和 AWS API，请参阅《软件开发工具包和 AWS 工具参考指南》中的 IAM 身份中心身份验证。
IAM	使用临时证书签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照 IAM 用户指南中的将临时证书与 AWS 资源配合使用中的说明进行操作。
IAM	（不推荐使用） 使用长期凭证签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关信息 AWS CLI，请参阅用户指南中的使用 IAM 用户证书进行身份验证。AWS Command Line Interface • 有关 AWS SDK 和工具，请参阅 S AWS DK 和工具参考指南中的使用长期凭证进行身份验证。 • 有关 AWS API，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

6. 选择下一步：权限。
7. 要提供访问权限，请为您的用户、组或角色添加权限：

- 中的用户和群组 AWS IAM Identity Center：

  创建权限集合。按照《AWS IAM Identity Center 用户指南》中 创建权限集 的说明进行操作。

- 通过身份提供商在 IAM 中托管的用户：

  创建适用于身份联合验证的角色。按照《IAM 用户指南》中为第三方身份提供商创建角色（联合身份验证）的说明进行操作。

- IAM 用户：
  - 创建您的用户可以担任的角色。按照《IAM 用户指南》中为 IAM 用户创建角色的说明进行操作。
  - （不推荐使用）将策略直接附加到用户或将用户添加到用户组。按照《IAM 用户指南》中向用户添加权限（控制台）中的说明进行操作。

8. 在搜索框中输入您创建的客户管理型策略的名称。然后，选中与策略名称对应的复选框。
9. 选择下一步：标签。
10. 选择 Next: Review (下一步: 审核)。
11. 选择 Create user。
12. 选择关闭。

Device Advisor 需要代表您访问您的 AWS 资源（内容、证书和终端节点）。您的 IAM 用户必须具有必要的权限。 CloudWatch 如果您向 IAM 用户附加必要的权限策略，设备顾问还将向 Amazon 发布日志。

配置您的设备

Device Advisor 使用服务器名称指示 (SNI) TLS 扩展来应用 TLS 配置。设备在连接时必须使用此扩展，并传递与 Device Advisor 测试端点相同的服务器名称。

当测试处于 `Running` 状态时，Device Advisor 允许 TLS 连接。它在每次测试运行之前和之后都会拒绝 TLS 连接。因此，我们建议使用设备连接重试机制，以便在使用 Device Advisor 时获得完全自动的测试体验。您可以运行包含多个测试用例的测试套件，例如 TLS 连接、MQTT 连接和 MQTT 发布。如果您运行多个测试用例，我们建议您的设备尝试每五秒钟连接到我们的测试端点。这样，您就可以自动按顺序运行多个测试用例。

> **Note**
>
> 为了让您的设备软件做好测试准备，我们建议您使用可以连接到 AWS IoT Core的 SDK。然后，您应该使用为您的 AWS 账户提供的 Device Advisor 测试端点更新 SDK 。

Device Advisor 支持两种类型的端点：账户级端点和设备级端点。请选择最符合您使用案例的终端节点。要针对不同的设备同时运行多个测试套件，请使用设备级端点。

运行以下命令以获取设备级别终端节点：

对于使用 X.509 客户端证书的 MQTT 客户：

```
aws iotdeviceadvisor get-endpoint --thing-arn your-thing-arn
```

或者

```
aws iotdeviceadvisor get-endpoint --certificate-arn your-certificate-arn
```

对于使用签名版本 4 的 MQTT 以上的 WebSocket 客户：

```
aws iotdeviceadvisor get-endpoint --device-role-arn your-device-role-arn --
authentication-method SignatureVersion4
```

要一次运行一个测试套件，请选择账户级终端节点。运行以下命令获取账户级别终端节点：

```
aws iotdeviceadvisor get-endpoint
```

在控制台中开始使用 Device Advisor

本教程可帮助您开始使用 AWS IoT Core Device Advisor 控制台。Device Advisor 提供了一些特征，如必需的测试和签名的资格报告等。您可以使用这些测试和报告来确定设备的资格并在 AWS 合作伙伴设备目录中列出设备，详见 AWS IoT Core 资格计划。

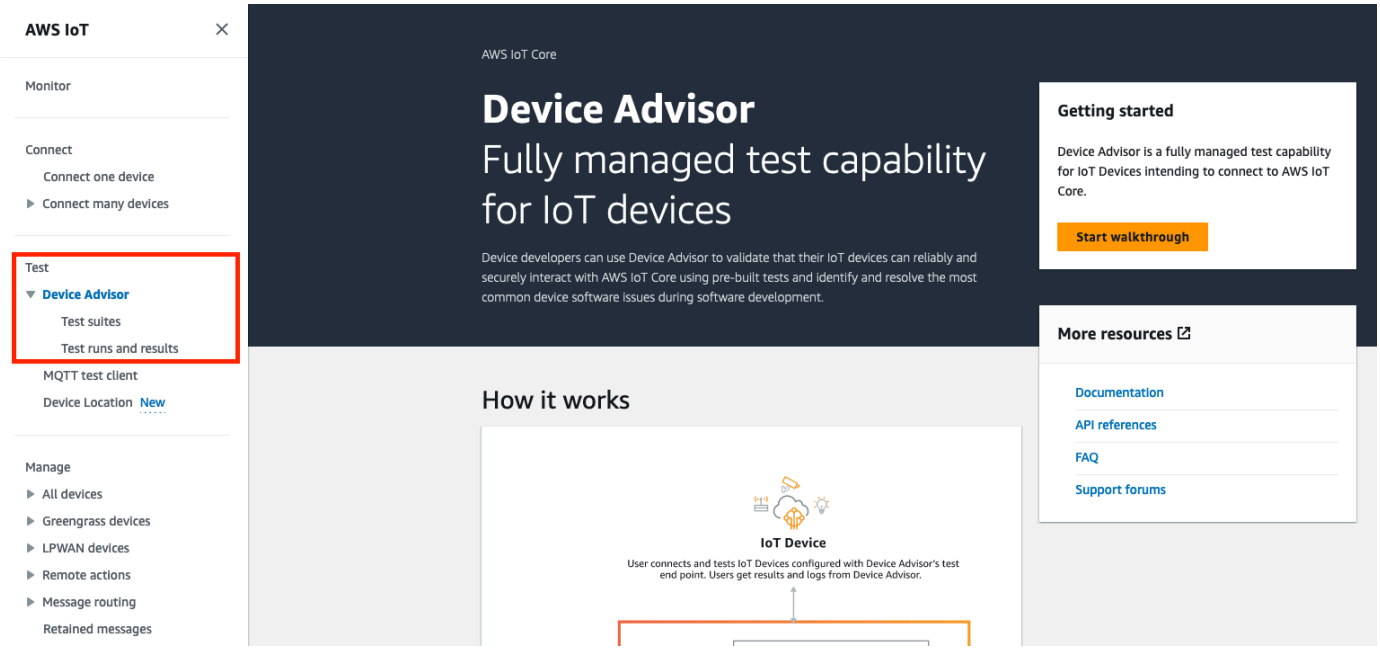
有关使用 Device Advisor 的更多信息，请参阅 Device Advisor 工作流 和 Device Advisor 详细控制台工作流。

要完成本教程，请执行 设置 中概述的步骤。

> **Note**
>
> 以下各项支持设备顾问 AWS 区域：
>
> - 美国东部（弗吉尼亚州北部）
> - 美国西部（俄勒冈）
> - 亚太地区（东京）
> - 欧洲地区（爱尔兰）

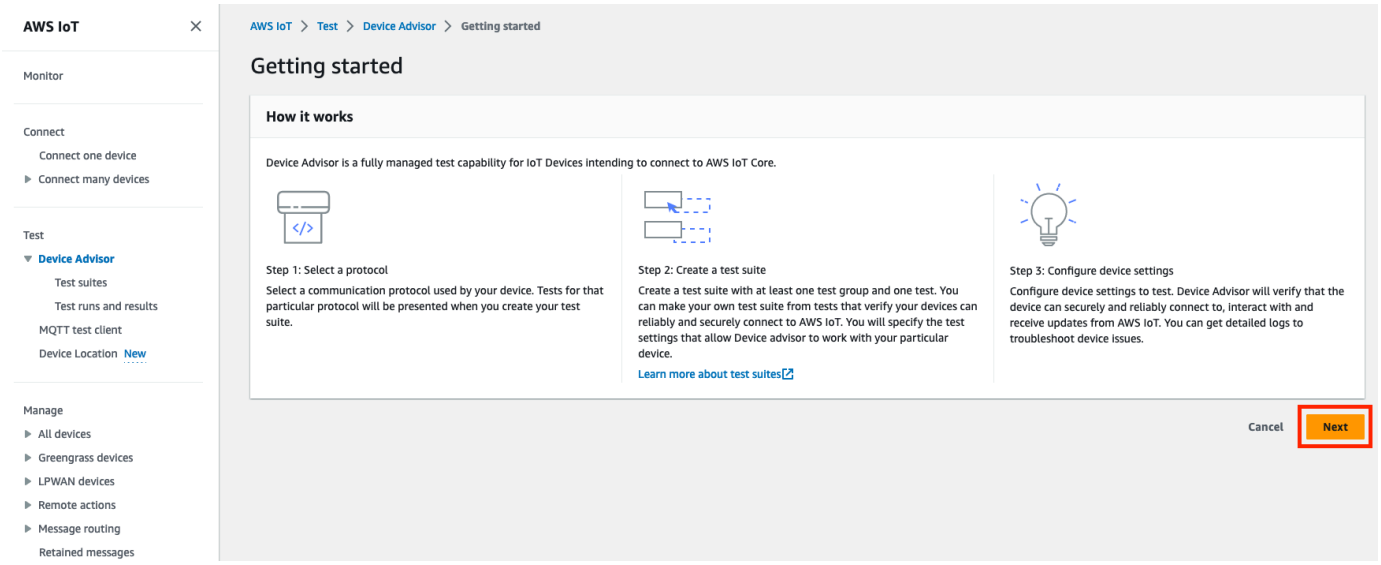
开始使用

1. 在 AWS IoT 控制台的导航窗格中的测试下，选择 Device Advisor。然后，选择控制台上的开始演练按钮。

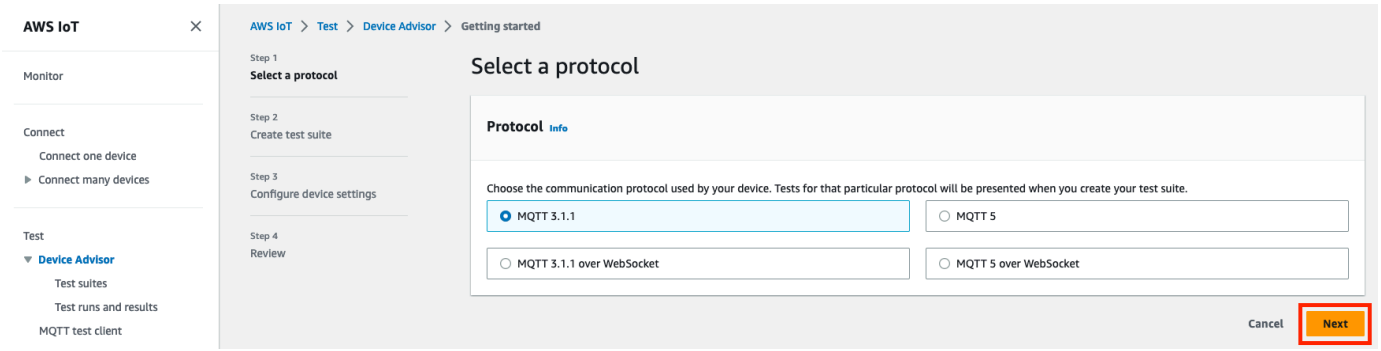


2. Device Advisor 入门页面概述了创建测试套件和对您的设备运行测试所需的各个步骤。您还可以在此处查找适用于您账户的 Device Advisor 测试端点。您必须在用于测试的设备上配置固件或软件，以连接到此测试端点。

   要完成本教程，请先创建事物和证书。查看工作原理下的信息后，选择下一步。

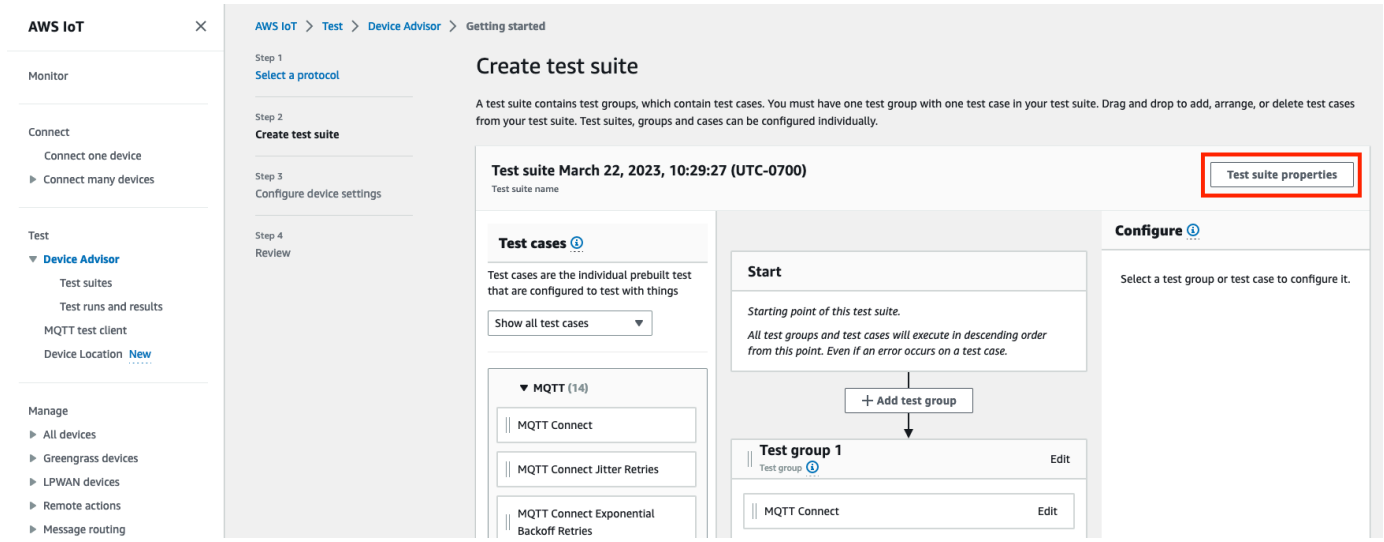


3. 在步骤 1：选择协议中，从列出的选项中选择协议。然后选择下一步。



4. 在 Step 2（步骤 2）中，您可以创建和配置自定义测试套件。自定义测试套件必须拥有至少一个测试组，并且每个测试组必须拥有至少一个测试用例。我们为您添加了 MQTT Connect 测试用例，以便您能开始操作。

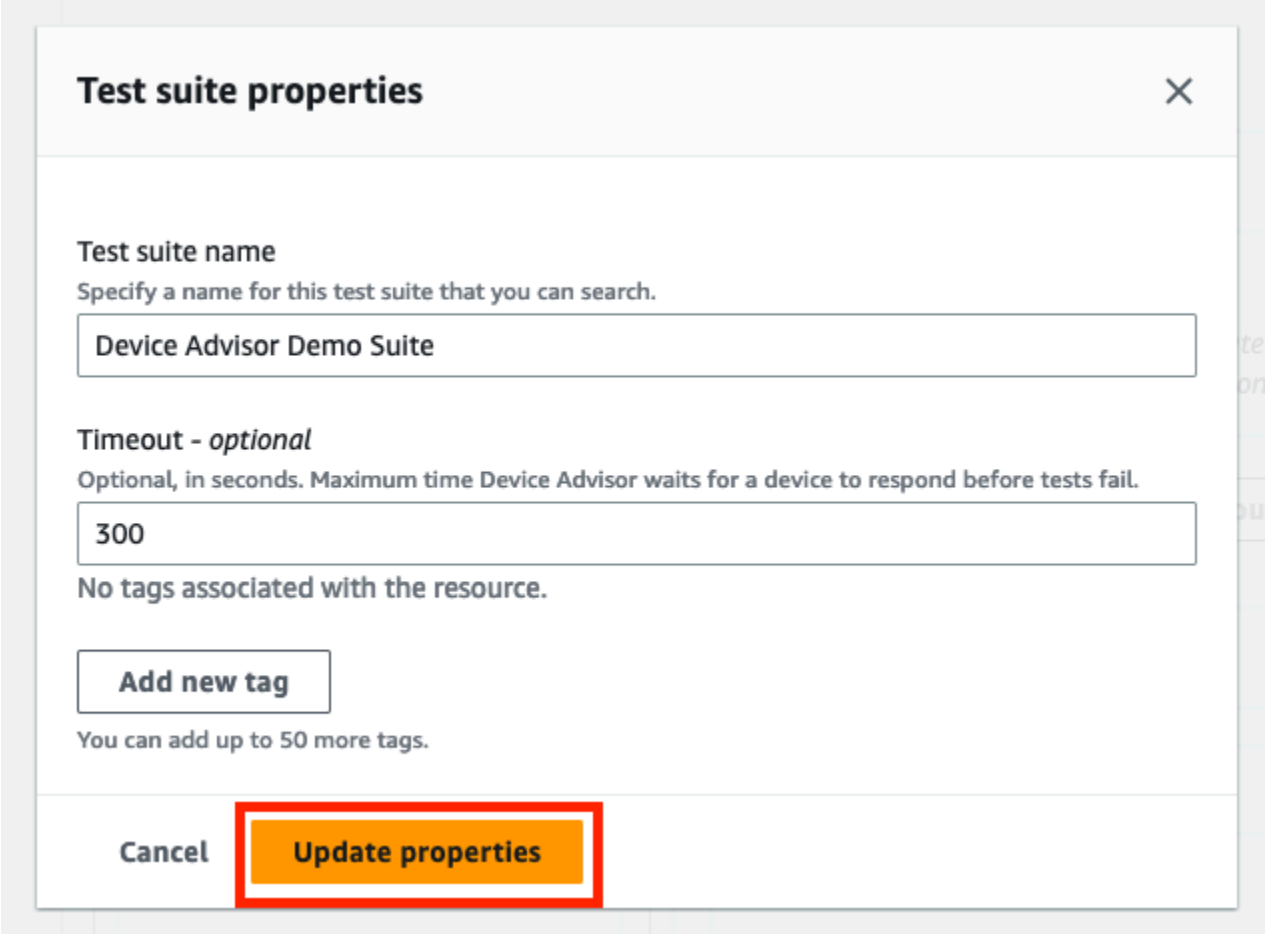
   选择 Test suite properties（测试套件属性）。



在创建测试套件时提供测试套件属性。您可以配置如下套件级属性：

- 测试套件名称：输入测试套件的名称。
- 超时（可选）：当前测试套件中每个测试用例的超时（以秒为单位）。如果您未指定超时值，则将使用默认值。
- 标签（可选）：向测试套件添加标签。

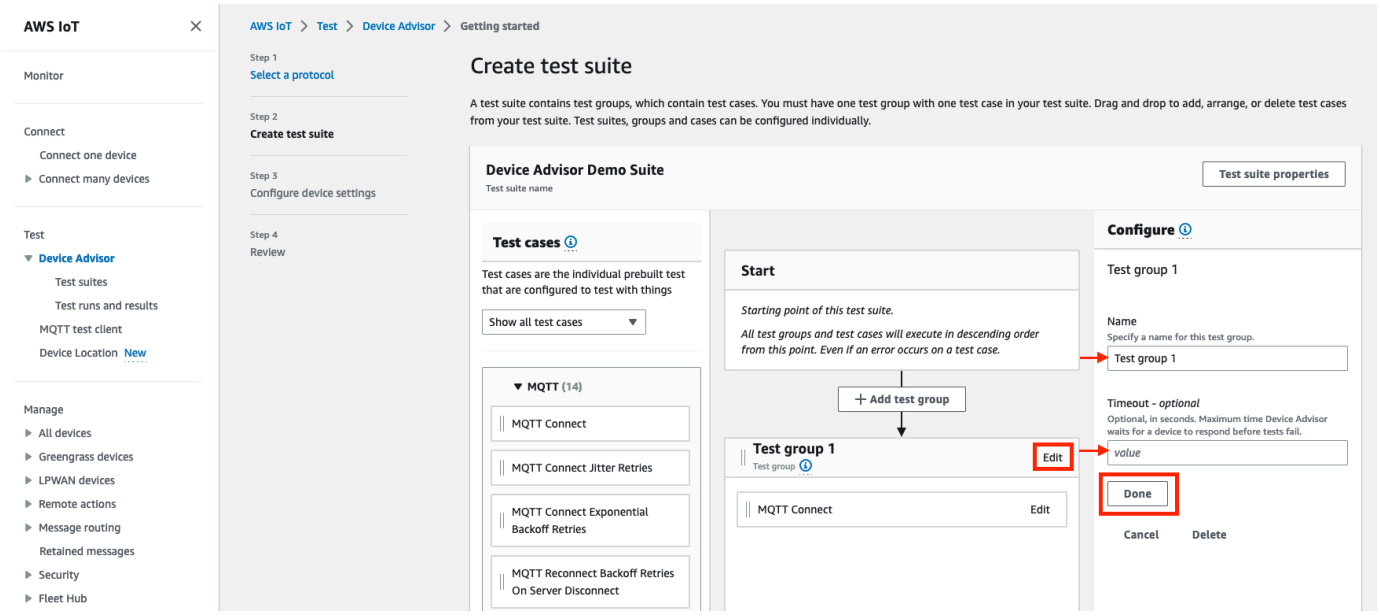
完成后，选择 Update properties（更新属性）。



5. （可选）要更新测试套件组配置，请选择测试组名称旁的编辑按钮。

   - 名称：输入测试套件组的自定义名称。
   - 超时（可选）：当前测试套件中每个测试用例的超时（以秒为单位）。如果您未指定超时值，则将使用默认值。

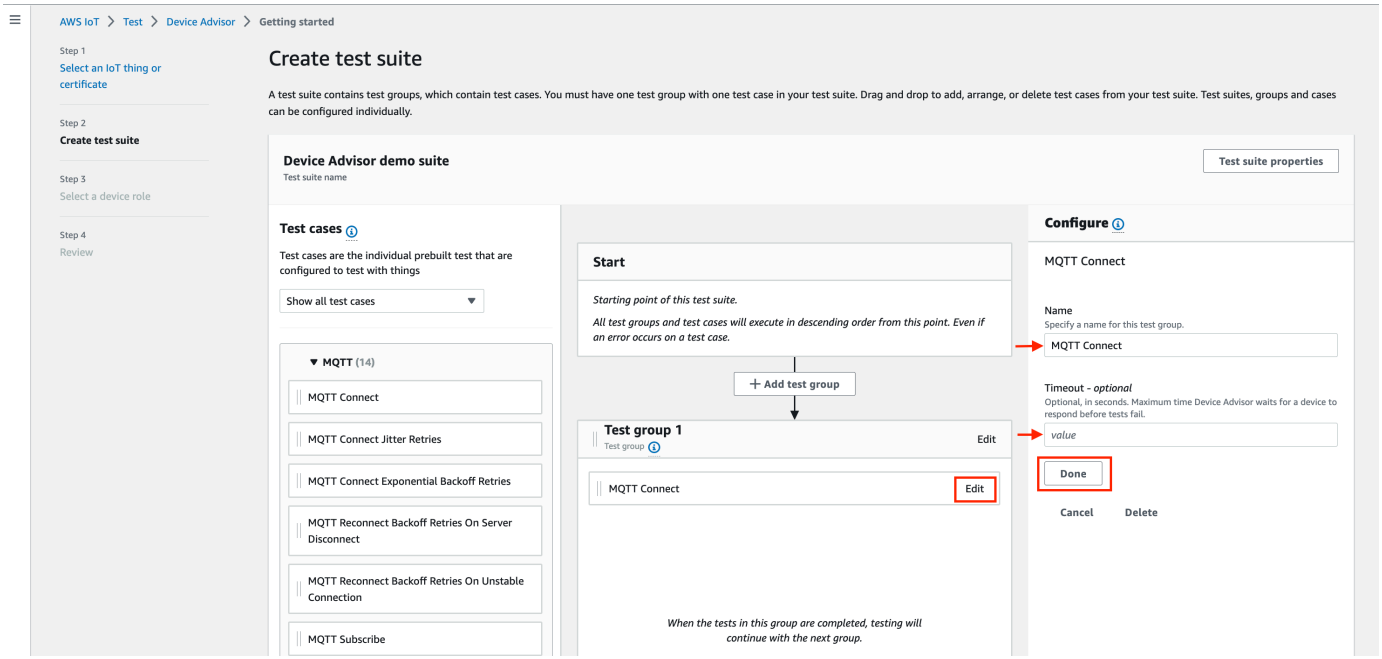
   完成后，选择完成以继续。



6. （可选）要更新测试用例的测试用例配置，请选择测试用例名称旁边的编辑按钮。

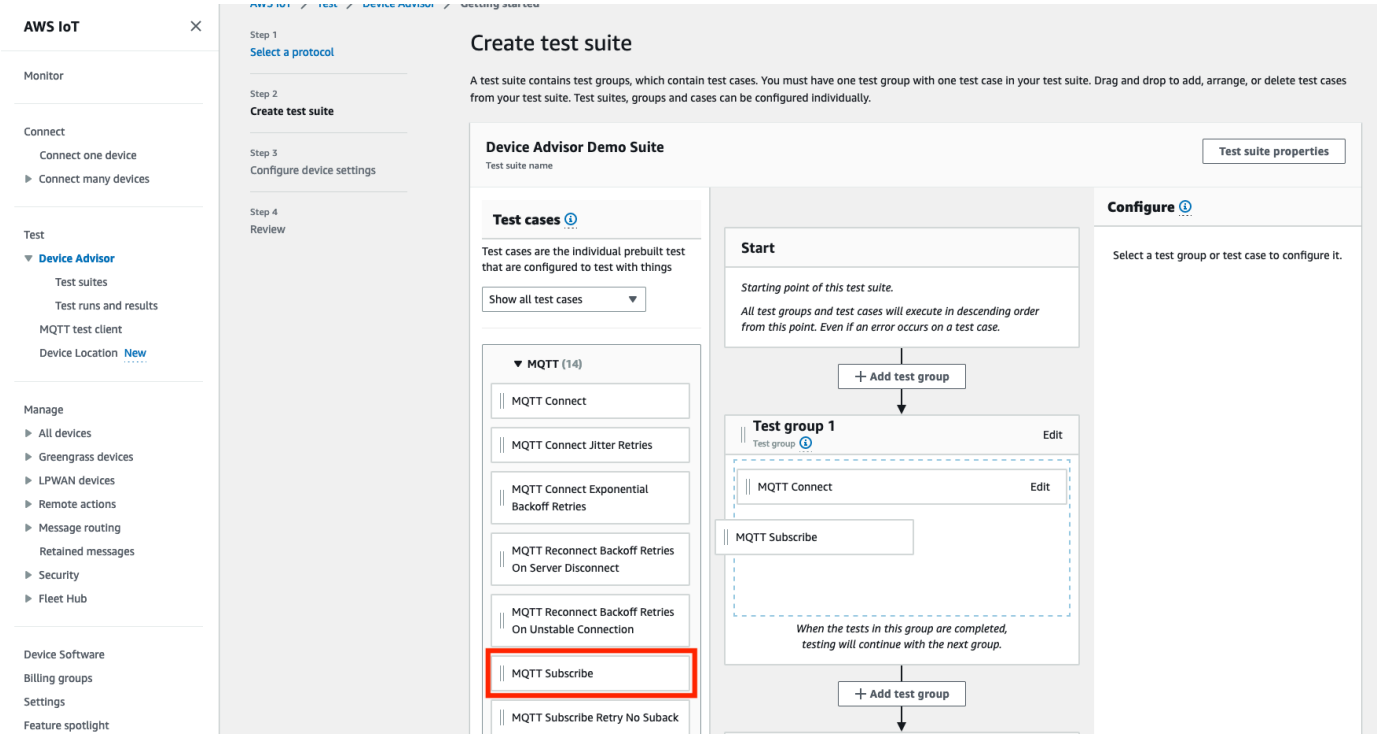
   - 名称：输入测试套件组的自定义名称。
   - 超时（可选）：所选测试用例的超时（以秒为单位）。如果您未指定超时值，则将使用默认值。

   完成后，选择完成以继续。



7. （可选）要向测试套件添加更多测试组，请选择添加测试组，然后按照步骤 5 中的说明进行操作。

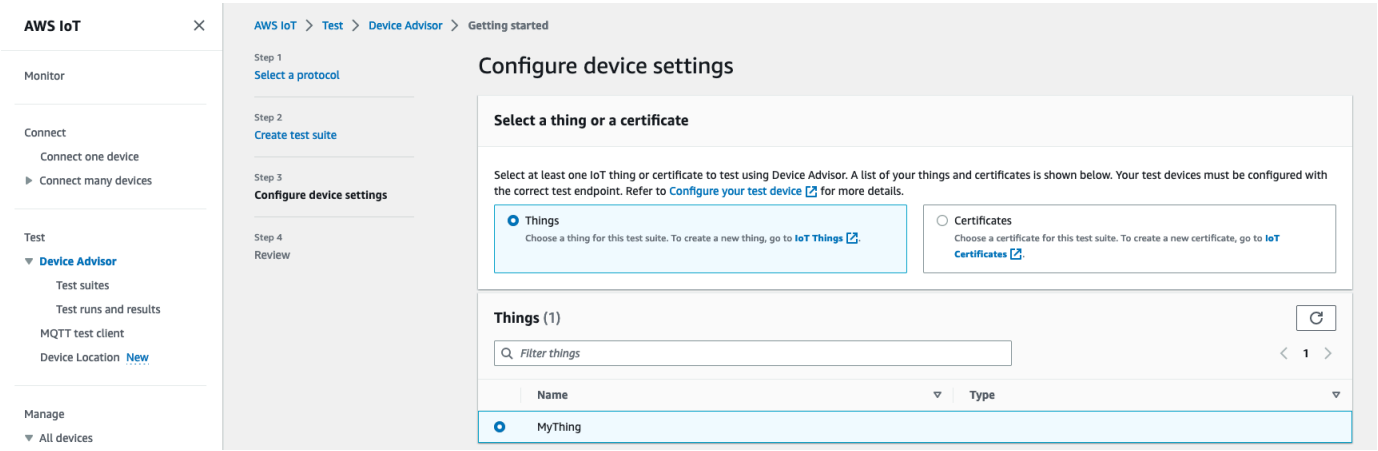
8. （可选）要添加更多测试用例，请将测试用例部分中的测试用例拖到您的任何测试组中。



9. 您可以更改测试组和测试用例的顺序。要进行更改，请在列表中向上或向下拖动列出的测试用例。Device Advisor 按照您列出的测试顺序运行测试。

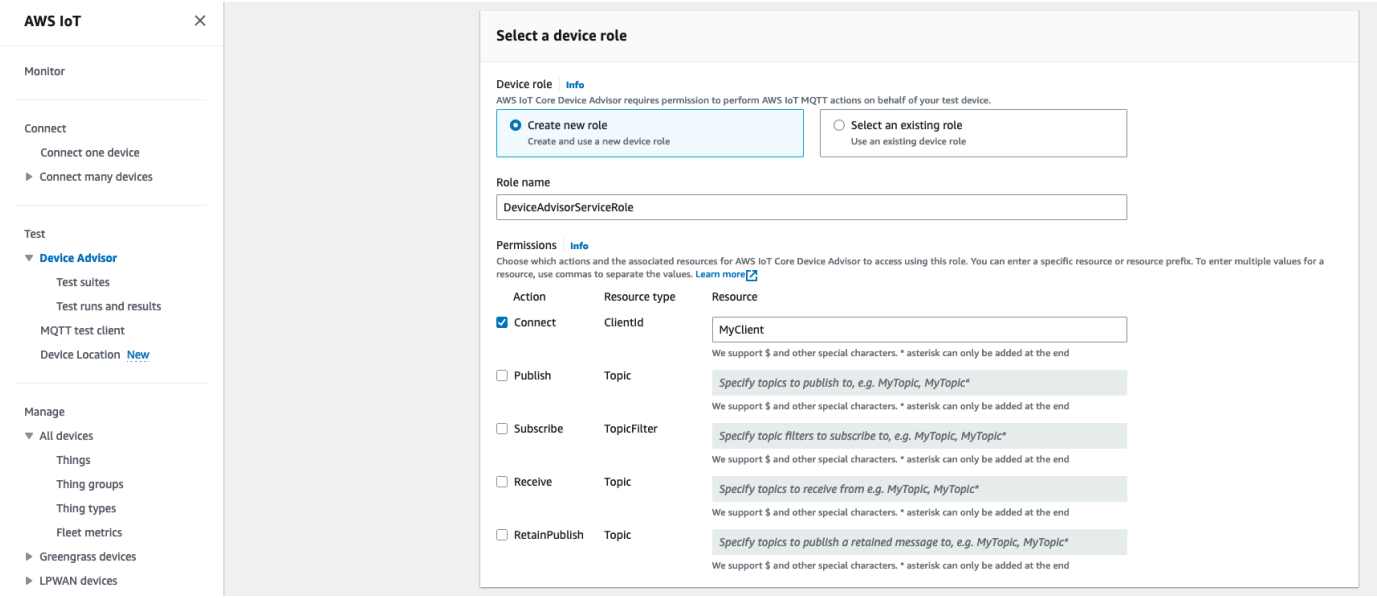
   配置测试套件后，选择 Next（下一步）。

10. 在步骤 3 中，选择要使用设备顾问进行测试 AWS IoT 的事物或证书。如果您没有任何现有的事物或证书，请参阅 设置。

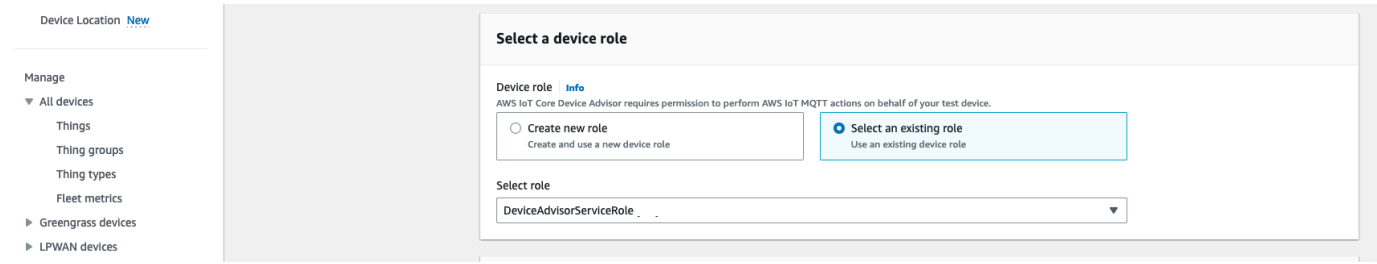


11. 您可以配置设备角色，设备顾问使用该角色代表您的测试设备执行 AWS IoT MQTT 操作。仅对于 MQTT 连接测试用例，自动选择连接操作。这是因为设备角色需要此权限才能运行测试套件。对于其它测试用例，将选择相应的操作。

为每个选定的操作提供资源值。例如，对于连接操作，提供客户端 ID，您的设备使用该 ID 连接到 Device Advisor 端点。您可以使用逗号分隔的值提供多个值，并使用通配符（*）作为值的前缀。例如，要为任何以 `MyTopic` 开头的主题提供发布权限，请输入 **`MyTopic*`** 作为资源值。

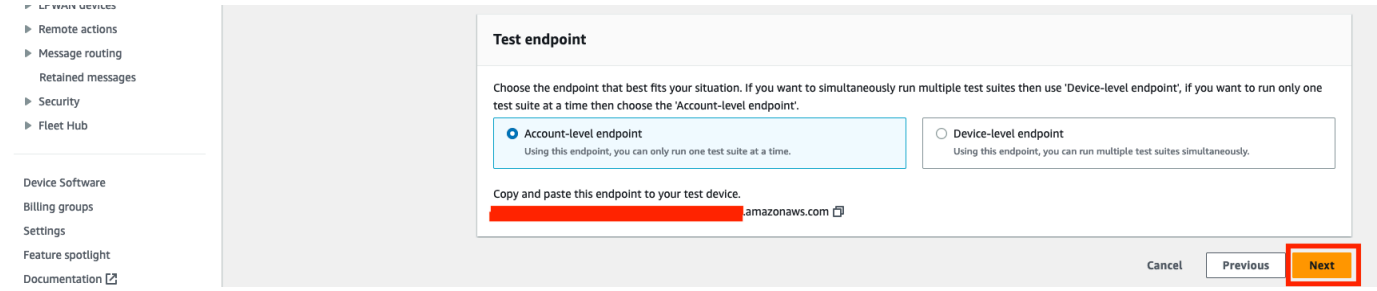


要使用之前从设置中创建的设备角色，请选择选择现有角色。然后，在选择角色下选择您的设备角色。



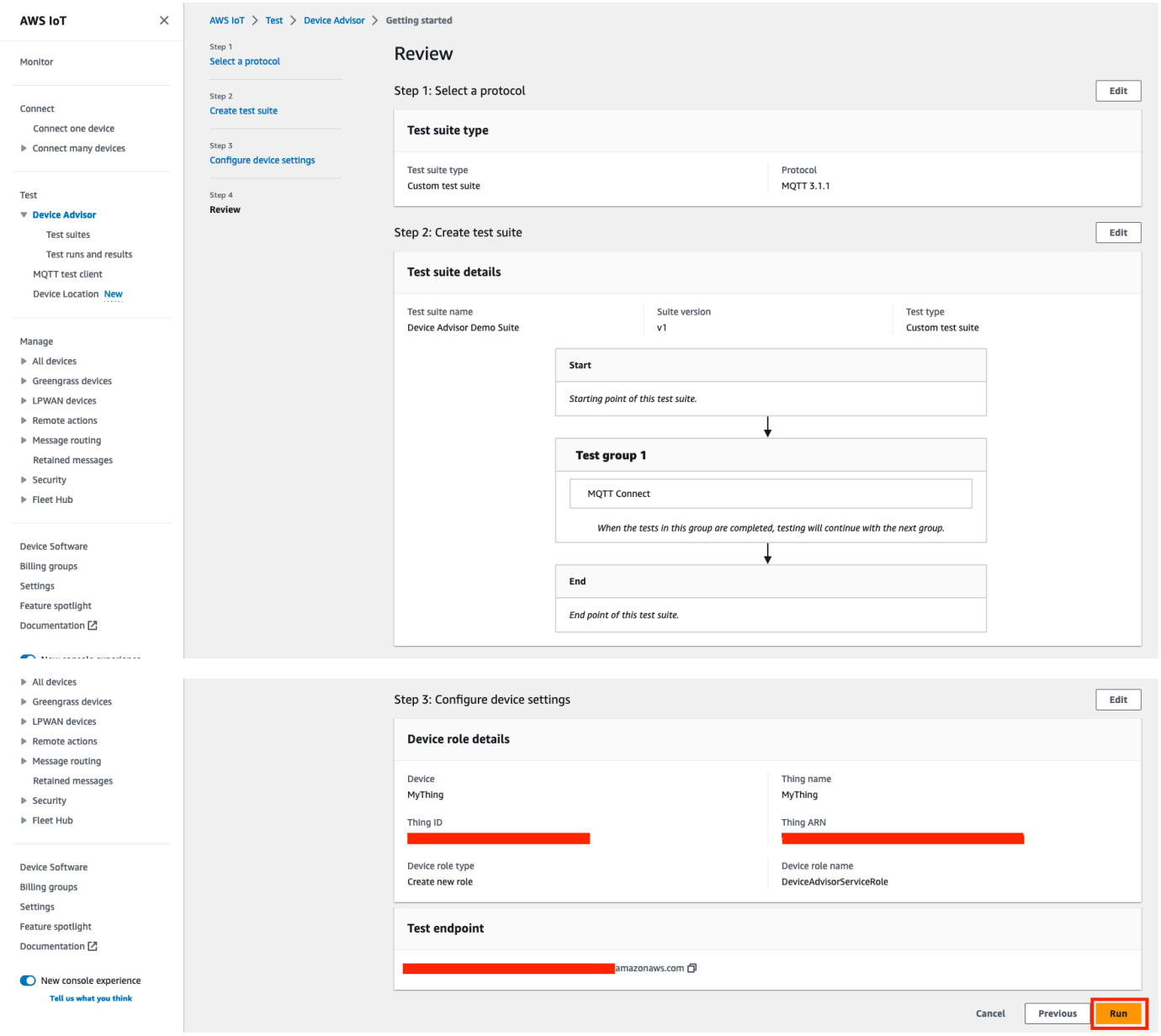
使用提供的两个选项之一配置设备角色，然后选择下一步。

12. 在测试端点部分中，选择最适合您的用例的端点。要使用相同的测试套件同时运行多个测试套件 AWS 账户，请选择设备级端点。要一次运行一个测试套件，请选择账户级端点。

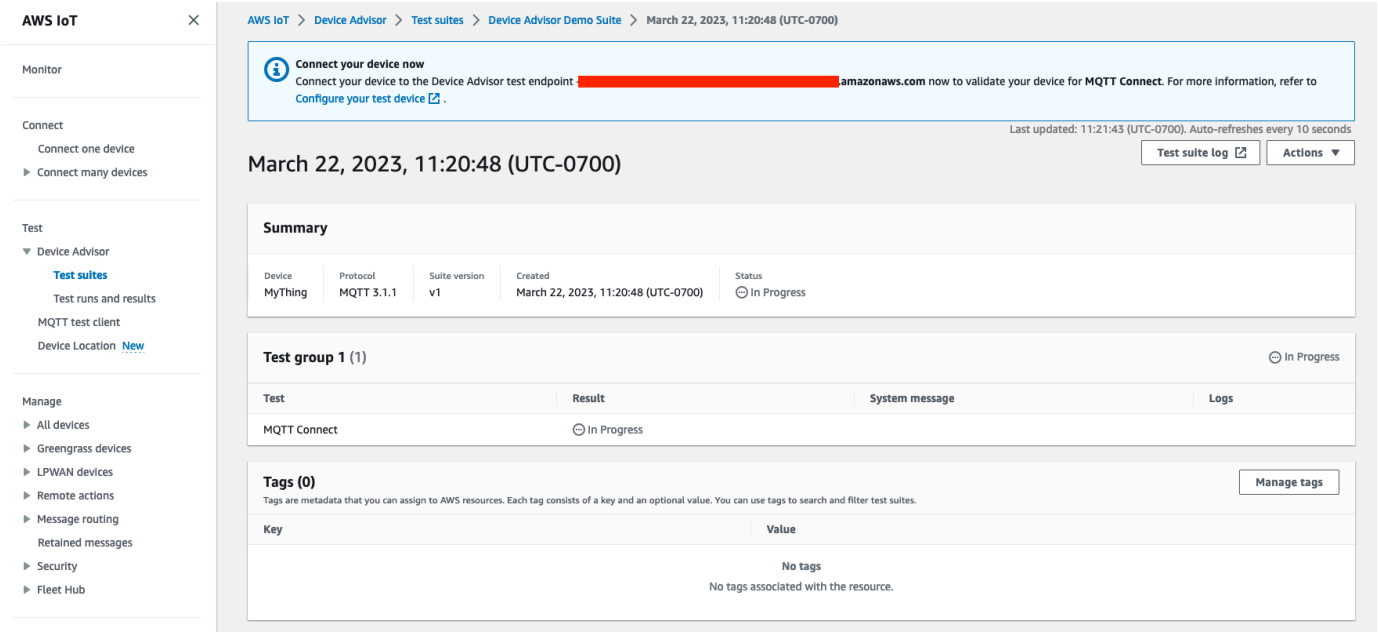


13. 步骤 4 显示了所配置的所选测试设备、测试端点、测试套件和测试设备角色的概览。要对某个部分进行更改，请对于要编辑的部分选择编辑按钮。确认测试配置后，选择运行以创建测试套件并运行测试。

> **Note**
>
> 为了获得最佳结果，您可以在启动套件运行之前，将选定的测试设备连接到 Device Advisor 测试端点。我们建议您为设备建立一个机制，以便每五秒尝试连接到测试端点一次，最多持续一到两分钟。



14. 在导航窗格的测试下的，选择 Device Advisor，然后选择测试运行和结果。选择测试套件运行以查看其运行详细信息和日志。



15. 要访问该套件的 Amazon CloudWatch 日志，请运行：

    - 选择 “测试套件 CloudWatch 日志” 以查看测试套件运行的日志。
    - 为任何测试用例选择测试用例日志，以查看特定于测试用例的 CloudWatch 日志。

16. 根据您的测试结果，对您的设备进行故障排除操作，直到通过所有测试。

Device Advisor 工作流

本教程说明如何创建自定义测试套件，以及如何在控制台中对要测试的设备运行测试。测试完成后，您可以查看测试结果和详细日志。

先决条件

在开始此教程之前，请完成设置中概述的步骤。

创建测试套件定义

首先，安装一个 AWS SDK。

rootGroup 语法

根组是一个 JSON 字符串，用于指定要在测试套件中包含哪些测试用例。它还为这些测试用例指定了任何必要的配置。使用根组根据您的需要构建和排列测试套件。测试套件的层次结构如下：

```
test suite # test group(s) # test case(s)
```

测试套件必须至少有一个测试组，并且每个测试组必须至少有一个测试用例。Device Advisor 将按照定义的测试组和测试用例顺序运行测试。

每个根组都遵循此基本结构：

```
{
    "configuration": {  // for all tests in the test suite
        "": ""
    }
    "tests": [{
        "name": ""
        "configuration": {  // for all sub-groups in this test group
            "": ""
        },
        "tests": [{
            "name": ""
            "configuration": {  // for all test cases in this test group
                "": ""
            },
            "test": {
                "id": ""
                "version": ""
            }
        }]
    }]
}
```

在根组中，您使用组包含的 name、configuration 和 tests 定义测试套件。tests 组包含各个测试的定义。您使用 name、configuration 和一个 test 数据块来定义每个测试，该数据块定义了该测试的测试用例。最后，每个测试用例都使用 id 和 version 来定义。

有关如何使用每个测试用例（test 数据块）中 "id" 和 "version" 字段的信息，请参阅Device Advisor 测试用例。该部分还包含有关可用 configuration 设置的信息。

以下数据块是根组配置的示例。此配置指定 MQTT Connect Happy Case 和 MQTT Connect Exponential Backoff Retries 测试用例，以及配置字段的描述。

```
{
    "configuration": {},  // Suite-level configuration
    "tests": [            // Group definitions should be provided here
      {
        "name": "My_MQTT_Connect_Group",  // Group definition name
        "configuration": {}               // Group definition-level configuration,
        "tests": [                        // Test case definitions should be provided
 here
        {
            "name": "My_MQTT_Connect_Happy_Case",  // Test case definition name
            "configuration": {
                "EXECUTION_TIMEOUT": 300        // Test case definition-level
 configuration, in seconds
            },
            "test": {
                "id": "MQTT_Connect",              // test case id
                "version": "0.0.0"                 // test case version
            }
        },
        {
            "name": "My_MQTT_Connect_Jitter_Backoff_Retries",  // Test case definition
 name
            "configuration": {
                "EXECUTION_TIMEOUT": 600                // Test case definition-level
 configuration,  in seconds
            },
            "test": {
                "id": "MQTT_Connect_Jitter_Backoff_Retries",  // test case id
                "version": "0.0.0"                                  // test case version
            }
        }]
    }]
}
```

创建测试套件定义时，您必须提供根组配置。保存在响应对象中返回的 `suiteDefinitionId`。您可以使用此 ID 检索测试套件定义信息和运行测试套件。

以下是一个 Java SDK 示例：

```
response = iotDeviceAdvisorClient.createSuiteDefinition(
```

```
        CreateSuiteDefinitionRequest.builder()
            .suiteDefinitionConfiguration(SuiteDefinitionConfiguration.builder()
                .suiteDefinitionName("your-suite-definition-name")
                .devices(
                    DeviceUnderTest.builder()
                        .thingArn("your-test-device-thing-arn")
                        .certificateArn("your-test-device-certificate-arn")
                        .deviceRoleArn("your-device-role-arn") //if using SigV4 for
 MQTT over WebSocket
                        .build()
                )
                .rootGroup("your-root-group-configuration")
                .devicePermissionRoleArn("your-device-permission-role-arn")
                .protocol("MqttV3_1_1 || MqttV5 || MqttV3_1_1_OverWebSocket ||
 MqttV5_OverWebSocket")
                .build()
            )
            .build()
)
```

获取测试套件定义

创建测试套件定义后，您会在 `CreateSuiteDefinition` API 操作的响应对象中收到 `suiteDefinitionId`。

当操作返回 `suiteDefinitionId` 时，您可能会在每个组中看到新的 `id` 字段，并在根组中看到测试用例定义。您可以使用这些 ID 来运行测试套件定义的子集。

Java SDK 示例：

```
response = iotDeviceAdvisorClient.GetSuiteDefinition(
    GetSuiteDefinitionRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .build()
)
```

获取测试终端节点

使用 `GetEndpoint` API 操作获取设备使用的测试端点。选择最适合您的测试的端点。要同时运行多个测试套件，请通过提供 `thing ARN`、`certificate ARN` 或 `device role ARN` 使用设备级端点。要运行单个测试套件，请不要为 GetEndpoint 操作提供任何参数来选择账户级端点。

SDK 示例：

```
response = iotDeviceAdvisorClient.getEndpoint(GetEndpointRequest.builder()
.certificateArn("your-test-device-certificate-arn")
.thingArn("your-test-device-thing-arn")
.deviceRoleArn("your-device-role-arn") //if using SigV4 for MQTT over WebSocket

.build())
```

启动测试套件运行

在创建测试套件定义并配置测试设备以连接到 Device Advisor 测试端点之后，请使用 `StartSuiteRun` API 运行您的测试套件。

对于 MQTT 客户，使用 `certificateArn` 或 `thingArn` 运行测试套件。如果两者都完成了配置，当证书属于该事物时，则使用该证书。

对于 MQTT 而不是 WebSocket 客户，请使用`deviceRoleArn`来运行测试套件。如果指定的角色与在测试套件定义中指定的角色不同，则指定的角色将覆盖定义的角色。

对于 `.parallelRun()`，如果通过一个 AWS 账户使用设备级端点并行运行多个测试套件，则使用 `true`。

SDK 示例：

```
response = iotDeviceAdvisorClient.startSuiteRun(StartSuiteRunRequest.builder()
.suiteDefinitionId("your-suite-definition-id")
.suiteRunConfiguration(SuiteRunConfiguration.builder()
    .primaryDevice(DeviceUnderTest.builder()
        .certificateArn("your-test-device-certificate-arn")
        .thingArn("your-test-device-thing-arn")
        .deviceRoleArn("your-device-role-arn") //if using SigV4 for MQTT over WebSocket

        .build())
    .parallelRun(true | false)
    .build())
.build())
```

保存响应中的 `suiteRunId`。您会用此来检索此测试套件运行的结果。

运行一个测试套件

启动测试套件运行后，您可以使用 GetSuiteRun API 检查其进度及结果。

SDK 示例：

```
// Using the SDK, call the GetSuiteRun API.

response = iotDeviceAdvisorClient.GetSuiteRun(
GetSuiteRunRequest.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

停止测试套件运行

要停止仍在运行的测试套件，可以调用 StopSuiteRun API 操作。在您调用 StopSuiteRun API 操作后，此服务启动清理过程。当服务运行清理过程时，测试套件运行状态更新为 Stopping。清理过程可能需要几分钟时间。此过程完成后，测试套件运行状态更新为 Stopped。测试运行完全停止后，您可以启动另一个测试套件运行。您可以使用 GetSuiteRun API 操作定期检查套件运行状态，如上一节所示。

SDK 示例：

```
// Using the SDK, call the StopSuiteRun API.

response = iotDeviceAdvisorClient.StopSuiteRun(
StopSuiteRun.builder()
    .suiteDefinitionId("your-suite-definition-id")
    .suiteRunId("your-suite-run-id")
.build())
```

获取成功的资格测试套件运行的资格报告

如果您运行的资格测试套件成功完成，则可以使用 GetSuiteRunReport API 操作检索资格报告。您使用此资格报告来通过 AWS IoT Core 资格计划为您的设备获取资格。要确定测试套件是否为合资格测试套件，请检查 intendedForQualification 参数是否设置为 true。调用 GetSuiteRunReport API 操作后，您最多可以从返回的 URL 下载报告，持续时间最长 90 秒。如果从上次调用 GetSuiteRunReport API 操作之后超过 90 秒，请再次调用该 API 操作以检索新的有效 URL。

SDK 示例：

```
// Using the SDK, call the getSuiteRunReport API.

response = iotDeviceAdvisorClient.getSuiteRunReport(
    GetSuiteRunReportRequest.builder()
        .suiteDefinitionId("your-suite-definition-id")
        .suiteRunId("your-suite-run-id")
        .build()
)
```

Device Advisor 详细控制台工作流

在本教程中，您将创建自定义测试套件，并对要在控制台中测试的设备运行测试。测试完成后，您可以查看测试结果和详细日志。

教程

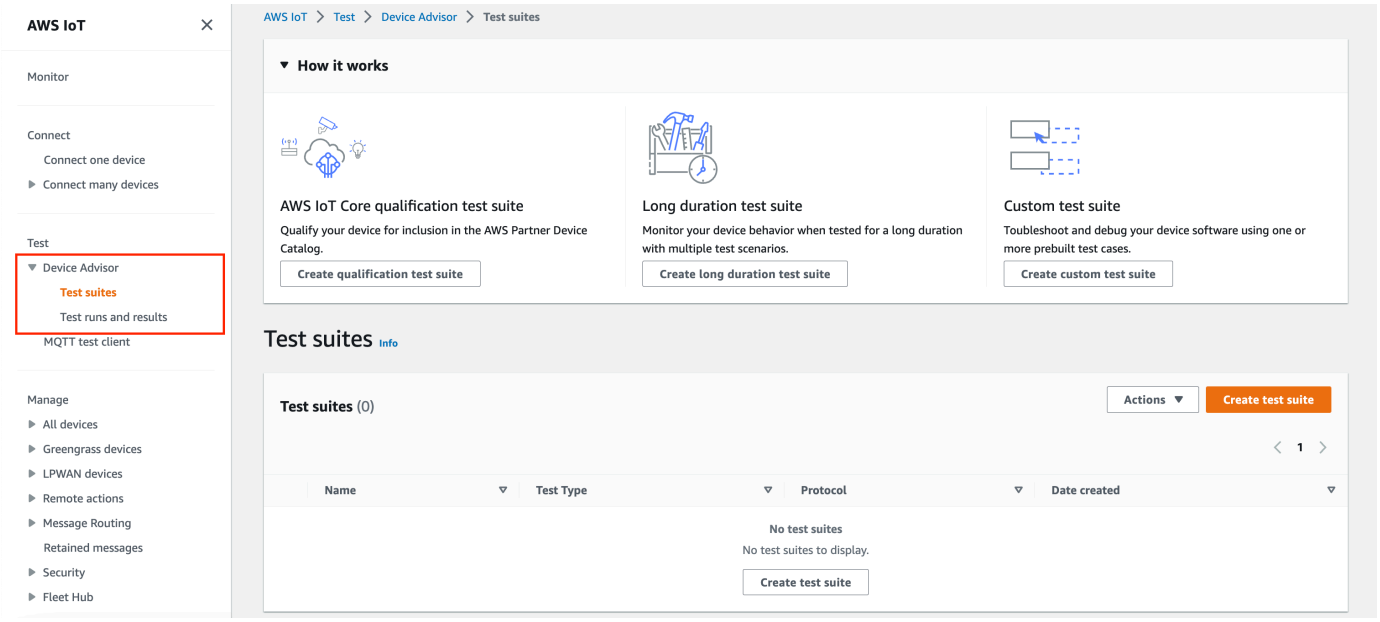
- 先决条件
- 创建测试套件定义
- 启动测试套件运行
- 停止测试套件运行（可选）
- 查看测试套件运行详细信息和日志
- 下载 AWS IoT 资格报告

先决条件

要完成此教程，您需要创建事物和证书。

创建测试套件定义

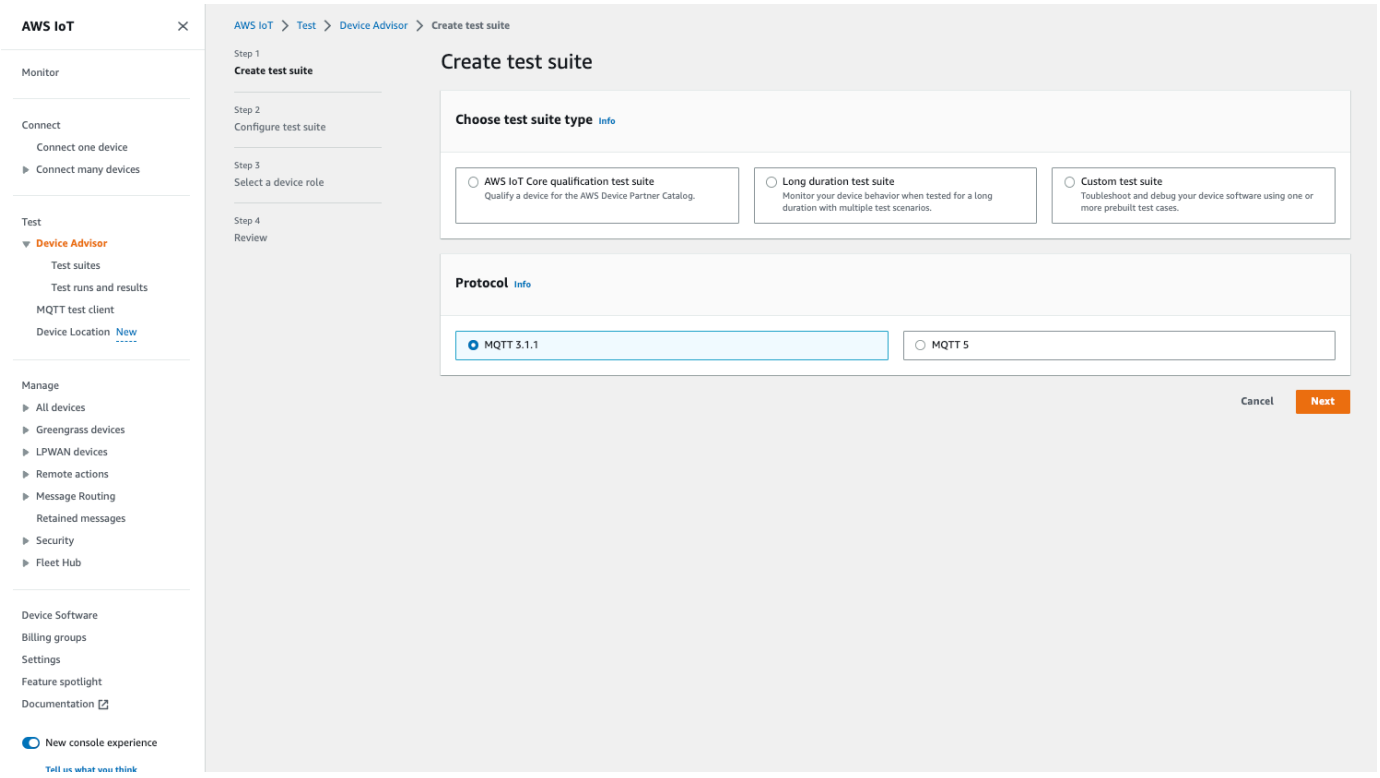
1. 在 AWS IoT 控制台的导航窗格中，展开 Test（测试）、Device Advisor，然后选择 Test suites（测试套件）。



选择 Create Test Suite（创建测试套件）。

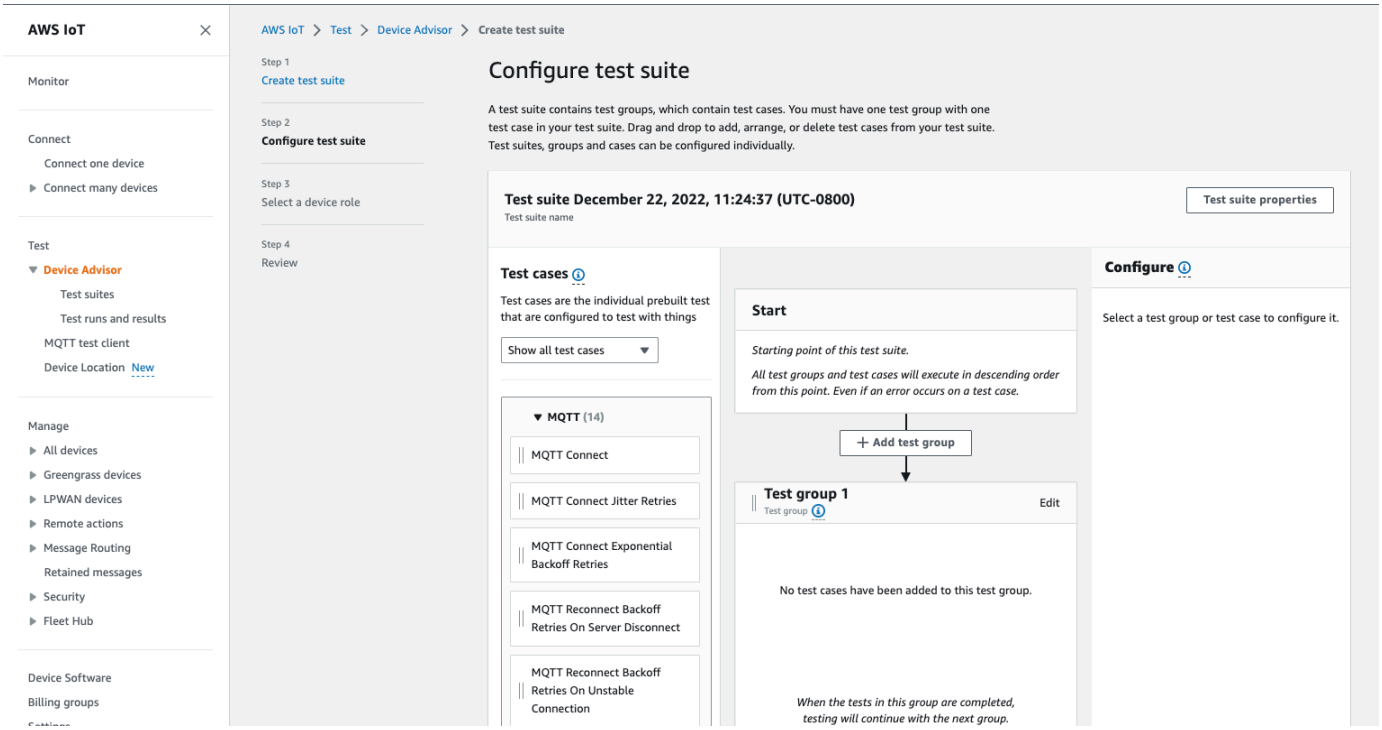
2. 选择 `Use the AWS Qualification test suite` 或 `Create a new test suite`。

对于协议，选择 MQTT 3.1.1 或 MQTT 5。

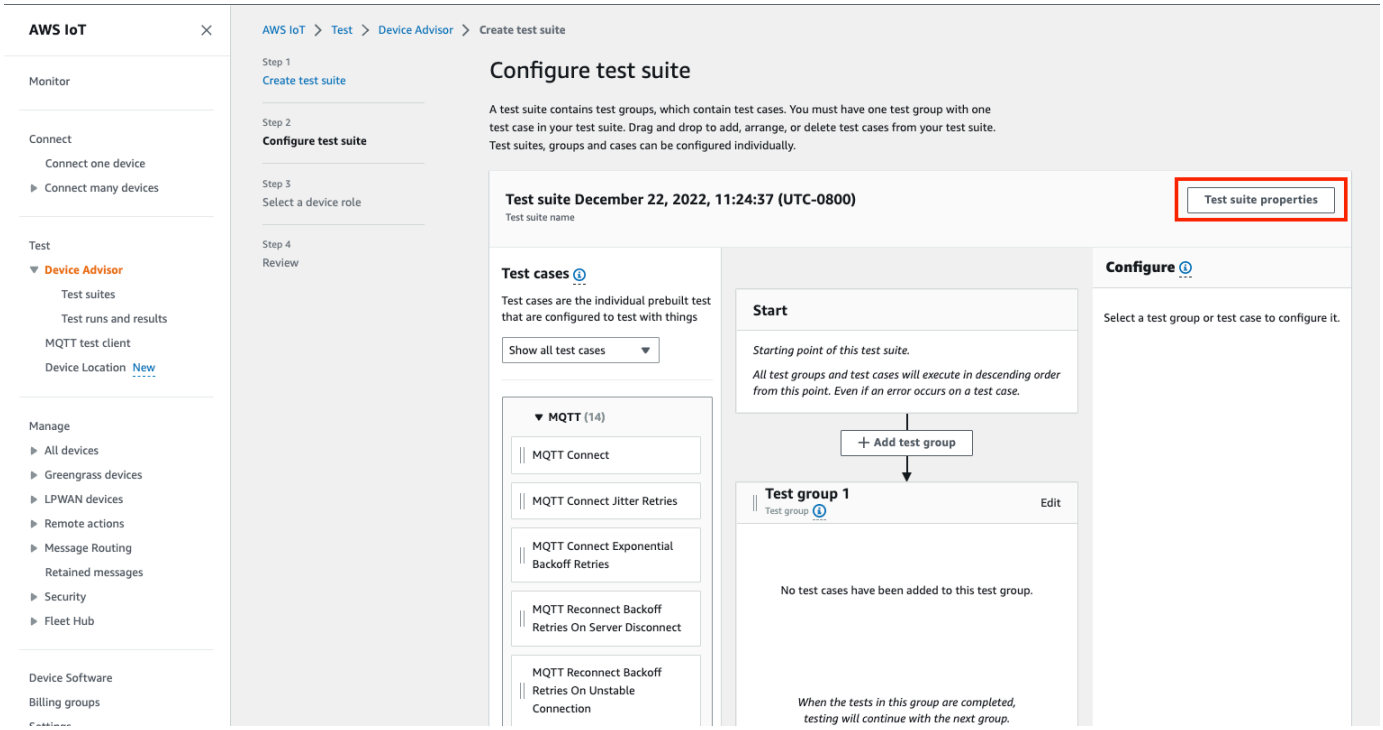


选择Use the AWS Qualification test suite获得资格并将您的设备发布到 AWS 合作伙伴设备目录中。通过选择此选项，可以预先选择使您的设备符合 AWS IoT Core 资格计划所需资格的测试用例。无法添加或删除测试组和测试用例。您仍然需要配置测试套件属性。

选择 Create a new test suite 来创建并配置自定义测试套件。我们建议从此选项开始进行初始测试和故障排除。自定义测试套件必须拥有至少一个测试组，并且每个测试组必须拥有至少一个测试用例。在本教程中，我们将选择此选项，然后选择 Next（下一步）。

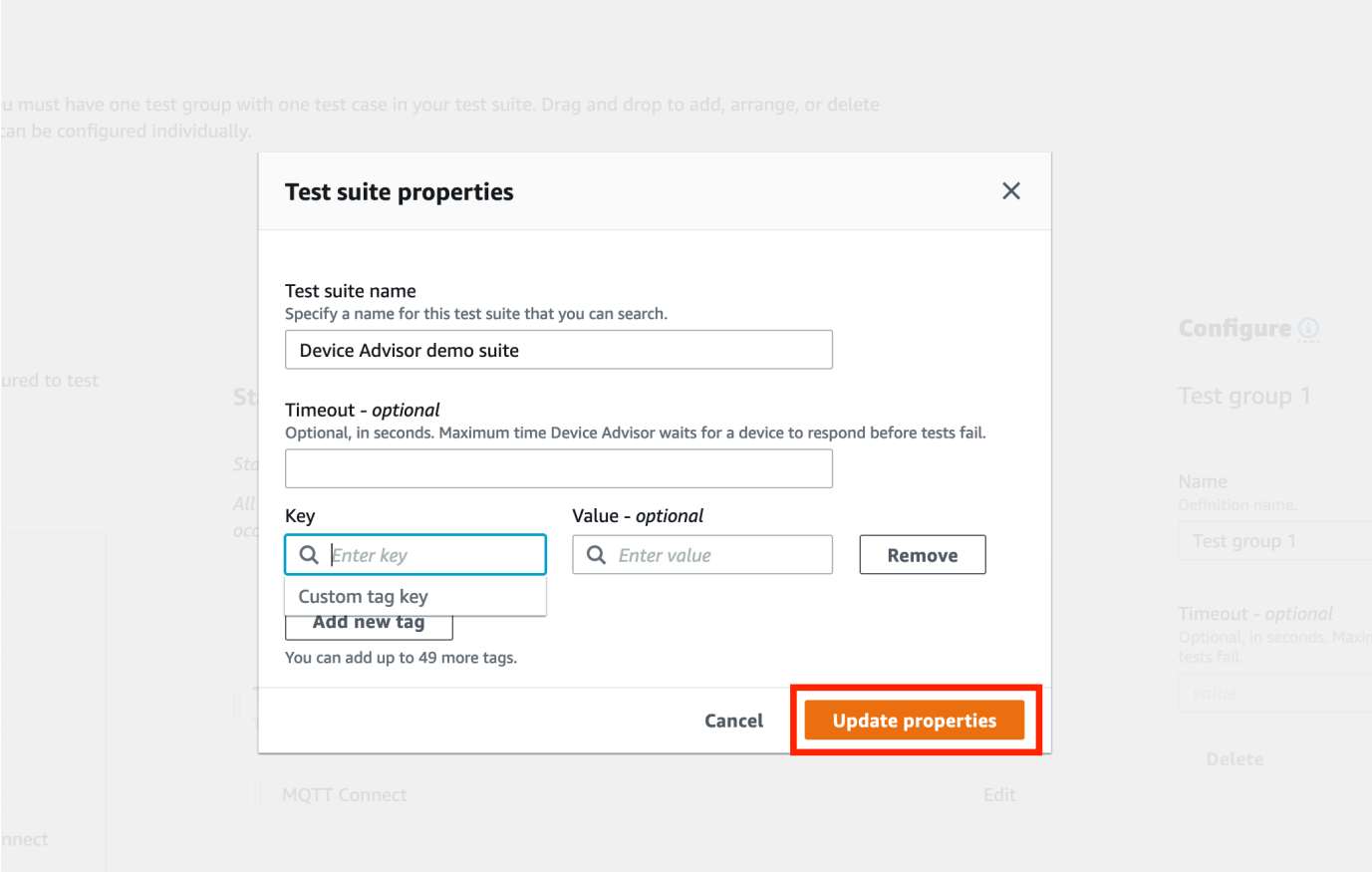


3. 选择 Test suite properties（测试套件属性）。创建测试套件时，您必须创建测试套件属性。



在 Test suite properties（测试套件属性）项下，请填写以下内容：

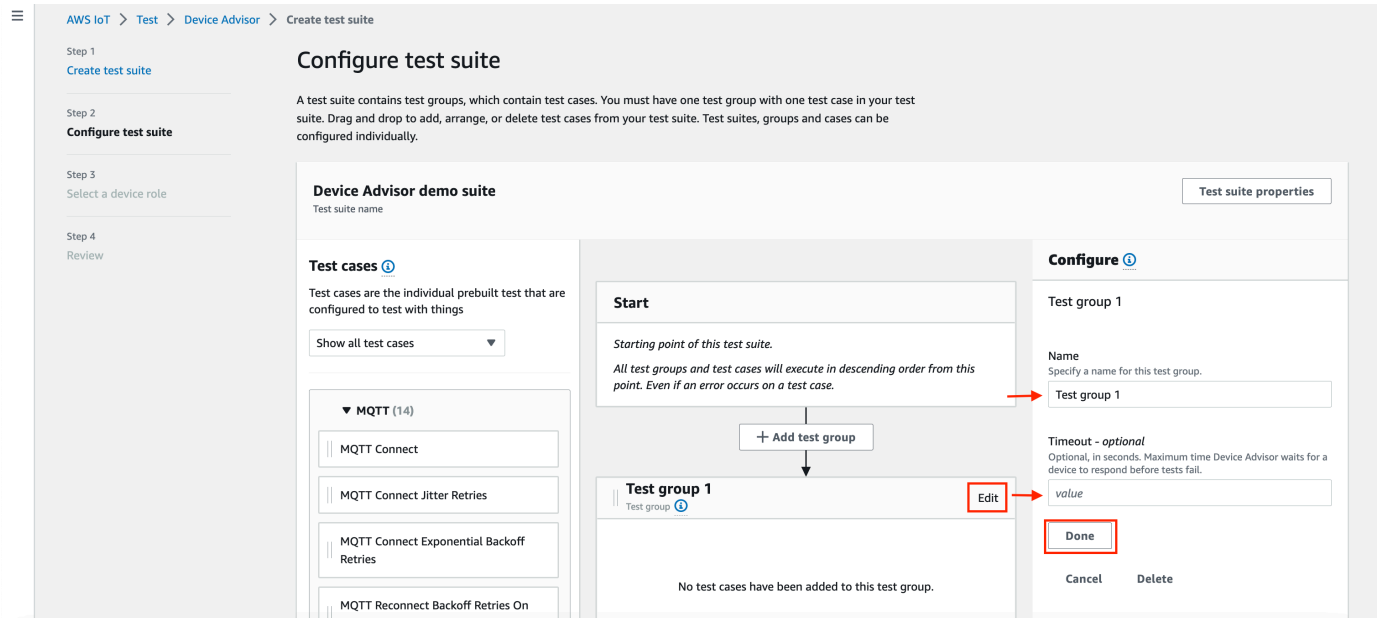
- Test suite name（测试套件名称）：您可以使用自定义名称创建套件。
- Timeout（超时）（可选）：当前测试套件中每个测试用例的超时时间（以秒为单位）。如果您未指定超时值，则将使用默认值。
- 标签（可选）：向测试套件添加标签。



完成后，选择 Update properties（更新属性）。

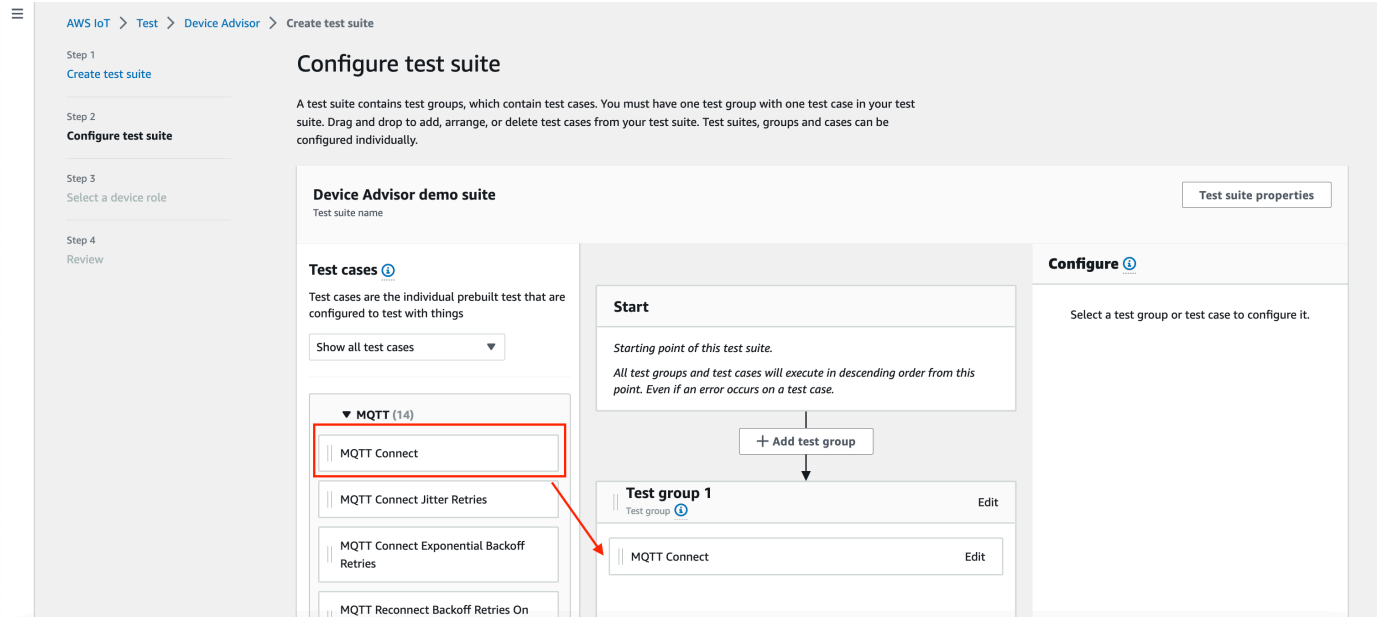
4. 要修改组级别配置，请在 `Test group 1` 下，选择 Edit（编辑）。然后，输入 Name（名称）为组指定一个自定义名称。

   或者，您也可以在所选的测试组下输入 Timeout（超时）值（以秒为单位）。如果您未指定超时值，则将使用默认值。



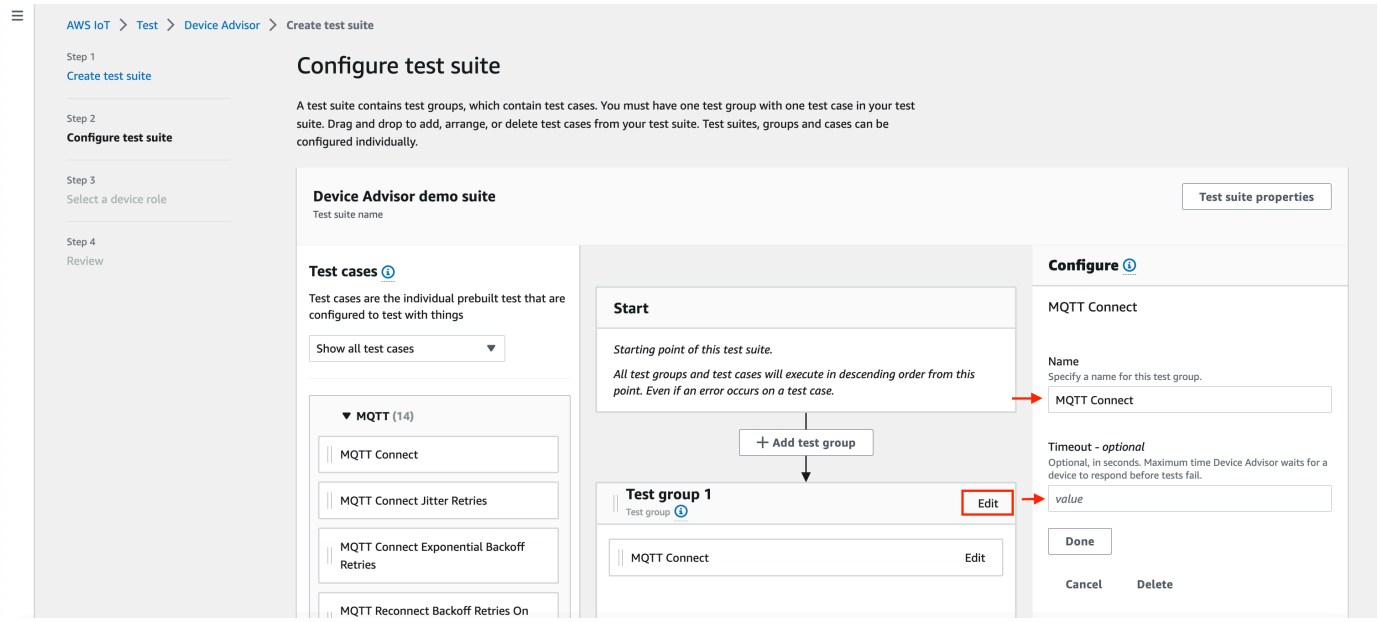
选择完成。

5. 从 Test cases（测试用例）中拖动一个可用的测试用例加入测试组。



6. 要修改您添加到测试组的测试用例级别配置，请选择 Edit（编辑）。然后，输入 Name（名称）为组指定一个自定义名称。

   或者，您也可以在所选的测试组下输入 Timeout（超时）值（以秒为单位）。如果您未指定超时值，则将使用默认值。

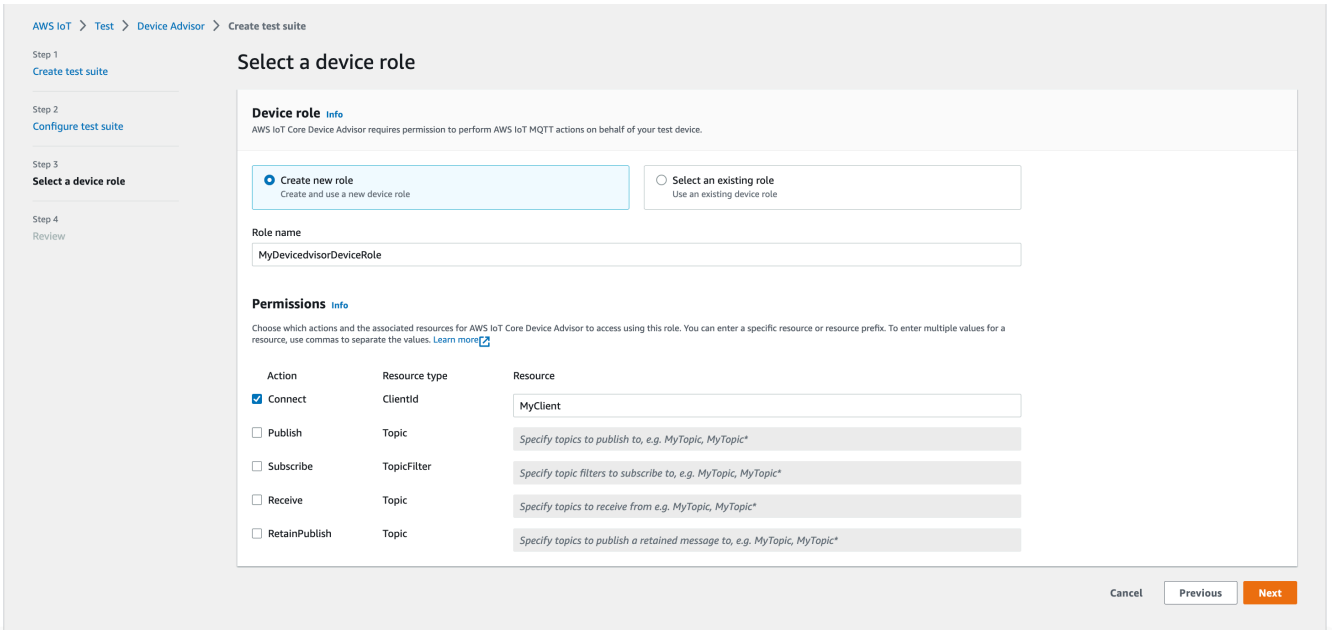


选择完成。

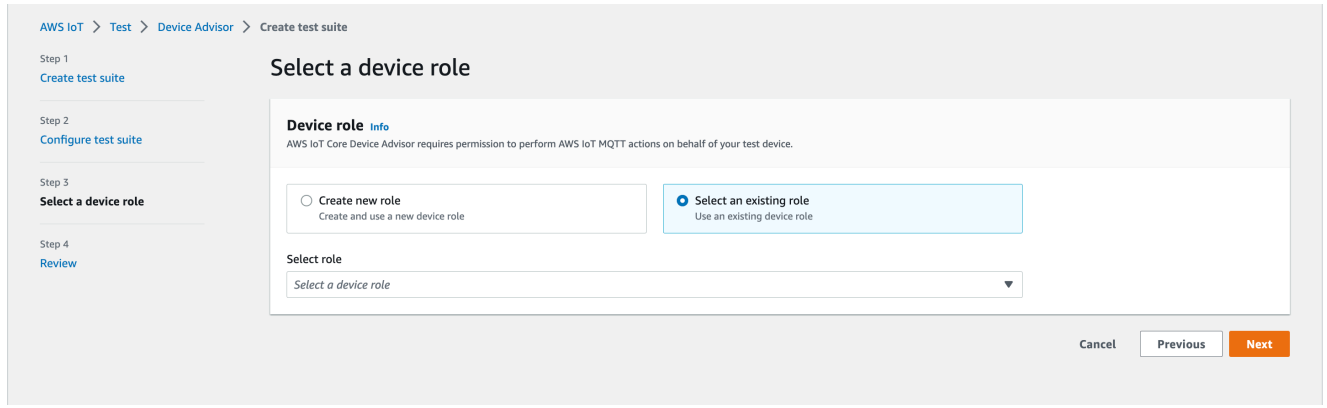
> **Note**
>
> 要向测试套件添加更多测试组，请选择 Add test group（添加测试组）。按照上述步骤创建和配置更多的测试组，或将更多测试用例添加到一个或多个测试组。可以通过选择并拖动测试用例到目标位置来重新排序测试组和测试用例。Device Advisor 将按照定义的测试组和测试用例顺序运行测试。

7. 选择下一步。
8. 在步骤 3 中，配置设备顾问将使用该角色代表您的测试设备执行 AWS IoT MQTT 操作。

如果您仅在步骤 2中选择 MQTT Connect 测试用例，将自动检查 Connect(连接)操作，因为运行此测试套件需要对设备角色具有该权限。如果您选择了其他测试用例，将检查相应的所需操作。确保提供了每个操作的资源值。例如，对于 Connect(连接)操作，提供设备将连接到 Device Advisor 终端节点的客户端 id。您可以通过使用逗号分隔值来提供多个值，也可以使用通配符 (*) 字符提供前缀值。例如，为任何 `MyTopic` 开头的主题提供发布权限，您可以将“`MyTopic*`”作为资源值。



如果您之前已经创建了设备角色，您希望使用该角色，请选择 Select an existing role（选择现有角色）然后在 Select role（选择角色）下面选择您的设备角色。



使用提供的两个选项之一配置设备角色，然后选择 Next（下一步）。

9. 在步骤 4中，请确保每个步骤中提供的配置准确无误。要编辑为特定步骤提供的配置，请选择 Edit（编辑）用于相应的步骤。

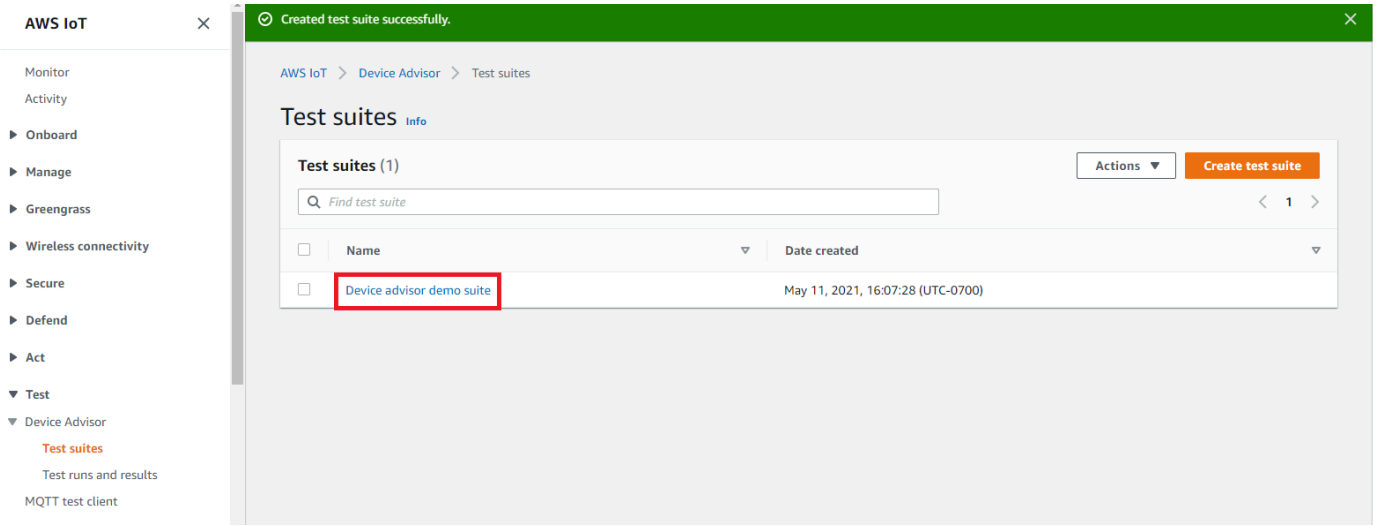
   验证配置后，选择 Create test suite（创建测试套件）。

   测试套件应能成功创建，并且您将被重定向到 Test suite（测试套件）页面，您可以在其中查看已创建的所有测试套件。

   如果测试套件创建失败，请确保已按照之前的说明配置测试套件、测试组、测试用例和设备角色。

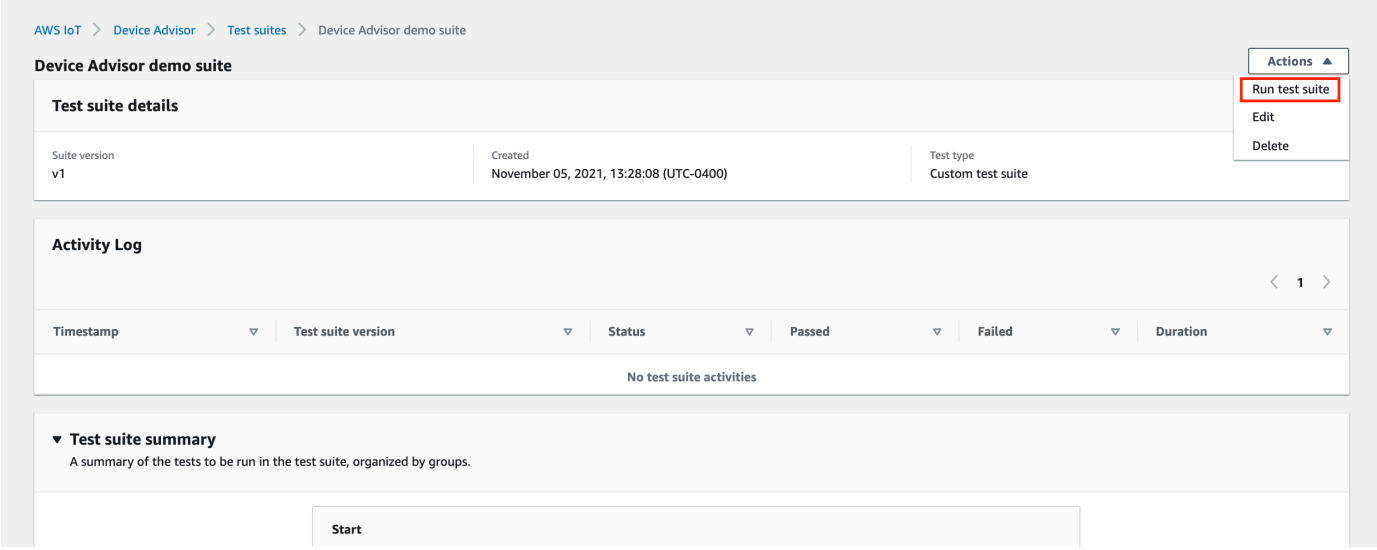
启动测试套件运行

1. 在AWS IoT 控制台的导航窗格中，展开测试、Device Advisor，然后选择 Test suite（测试套件）。
2. 选择要查看其测试套件详细信息的测试套件。



测试套件详细信息页面将显示与测试套件相关的所有信息。

3. 选择 Actions（操作），然后选择 Run test suite（运行测试套件）。

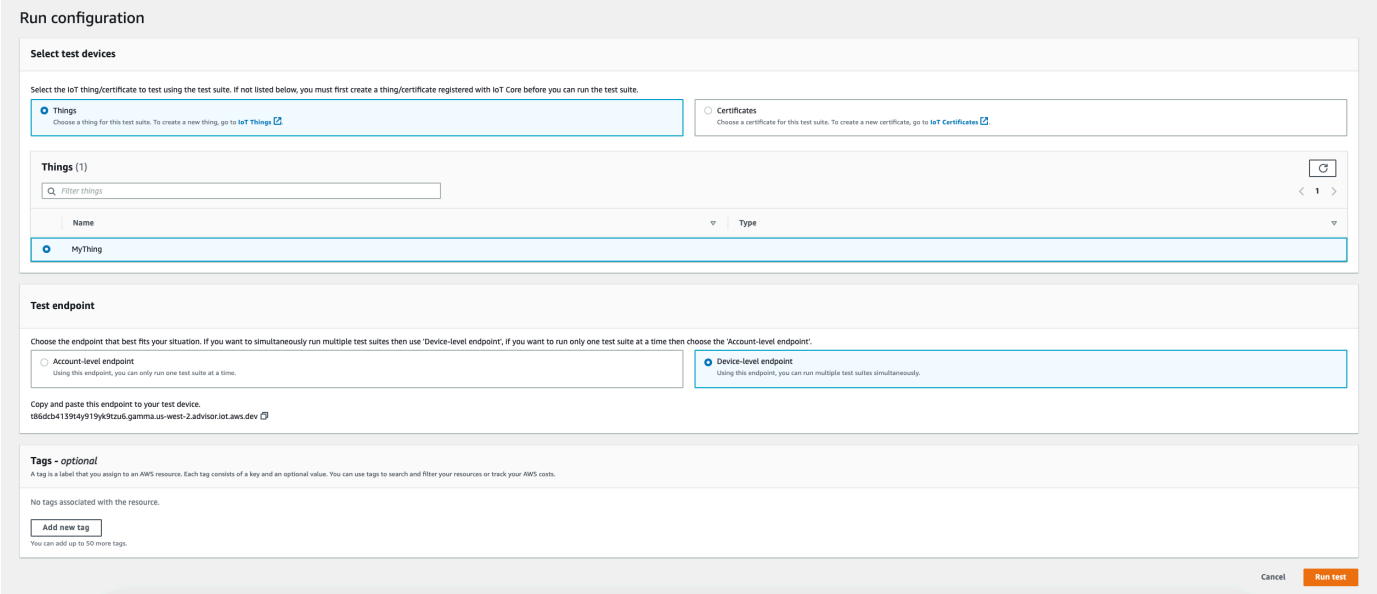


4. 在 “运行配置” 下，您需要选择要使用设备顾问进行测试 AWS IoT 的事物或证书。如果您没有任何现有内容或证书，请先创建 AWS IoT Core 资源。

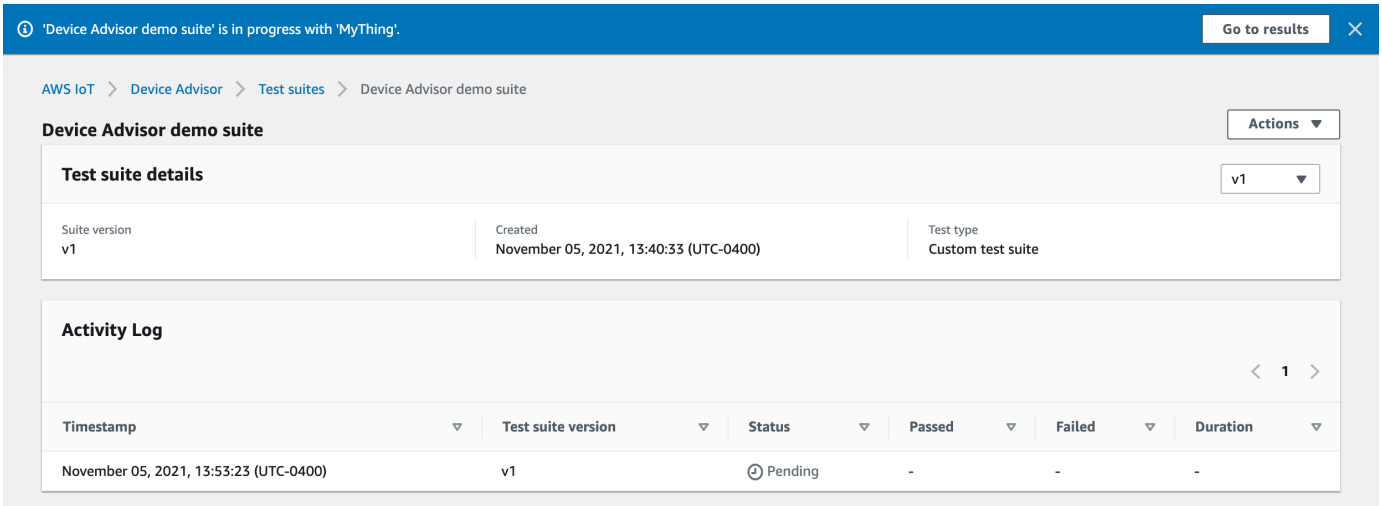
在测试终端节点中，选择最适合您用例的终端节点。如果您计划将来使用同一个 AWS 账户同时运行多个测试套件，请选择设备级终端节点。否则，如果您计划一次只运行一个测试套件，请选择 Account-level endpoint（账户级别终端节点）。

使用选定的 Device Advisor 的测试终端节点配置测试设备。

选择事物或证书以及 Device Advisor 终端节点后，选择 Run test（运行测试）。

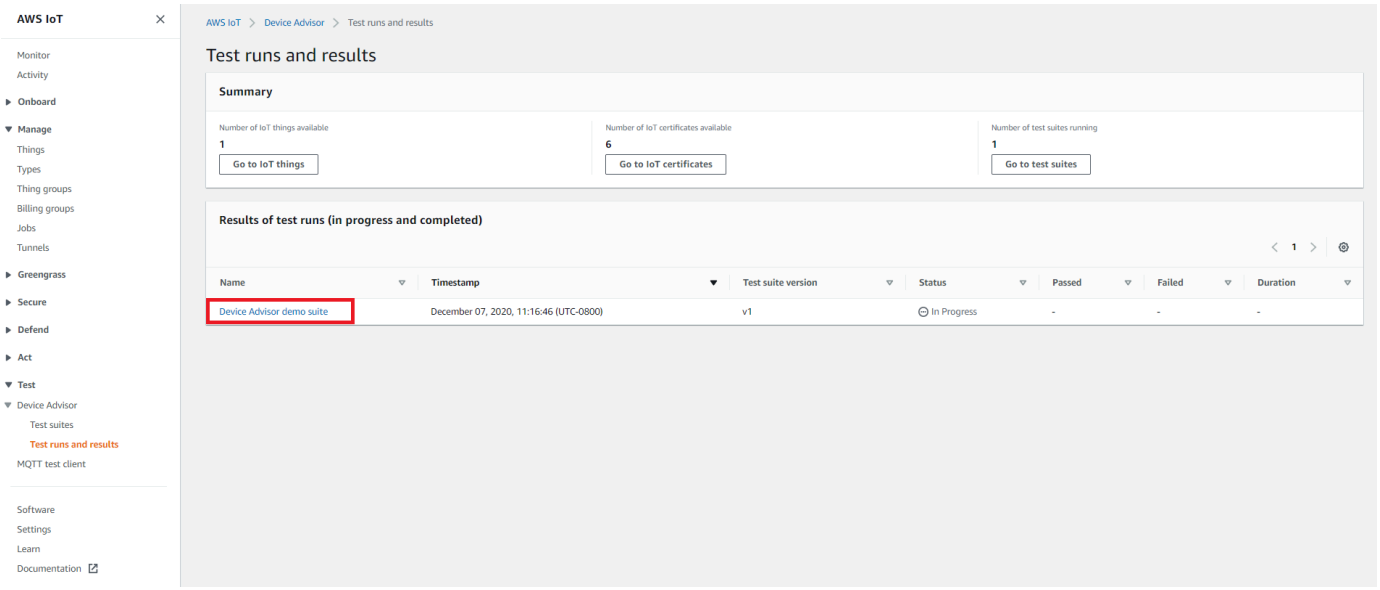


5. 选择顶部广告条的 Go to result（转到结果），以查看测试运行详细信息。

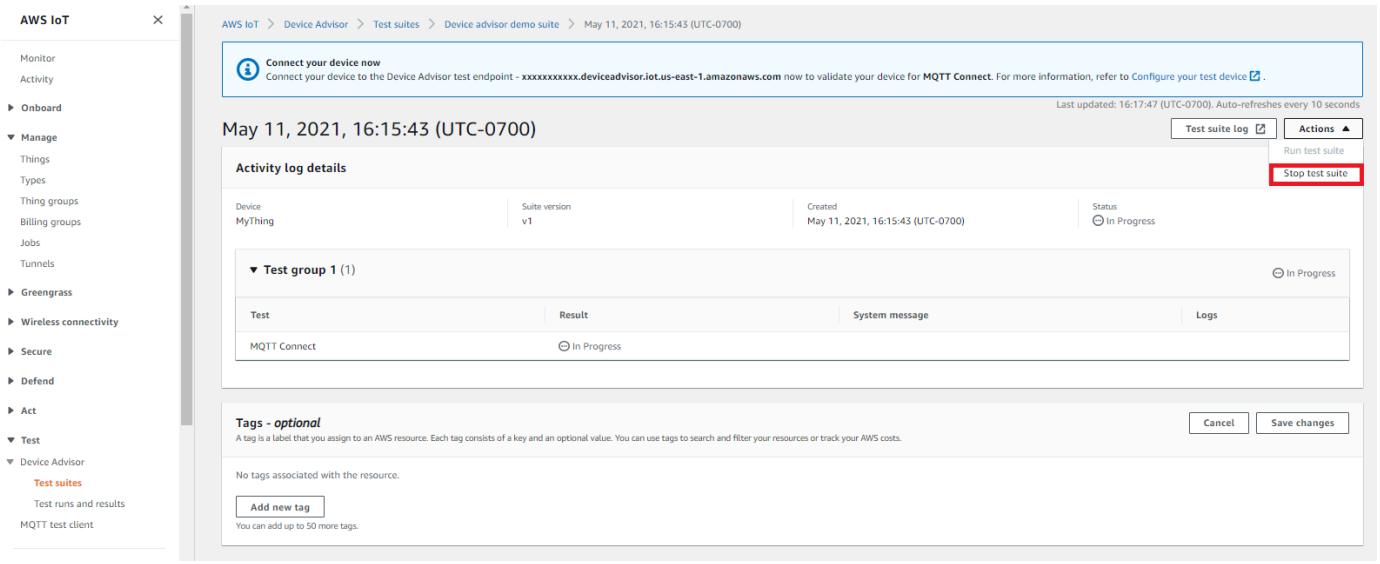


停止测试套件运行（可选）

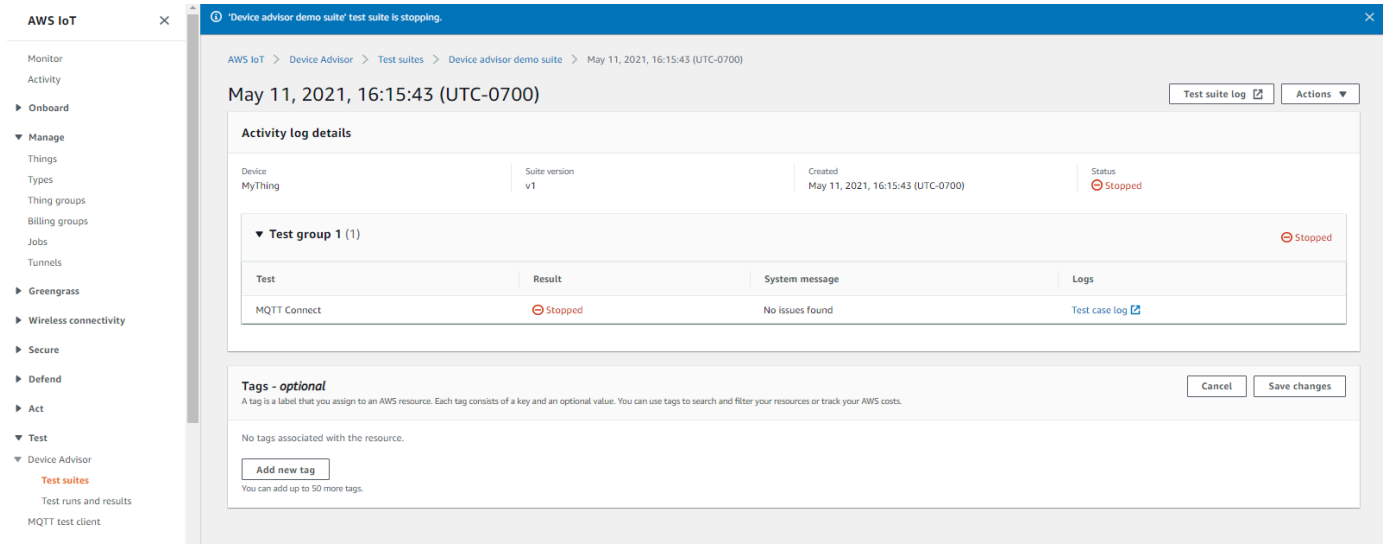
1. 在 AWS IoT 控制台的导航窗格中，展开 Test（测试）、Device Advisor，然后选择 Test run and results（测试运行和结果）。
2. 选择要停止的正在进行的测试套件。



3. 选择 Actions（操作），然后选择 Stop test suite（停止测试套件）。



4. 此清理过程可能需要几分钟才能完成。清理程序运行时，测试运行状态将为 STOPPING。等待清理程序完成，并等待测试套件状态更改为 STOPPED 状态，然后再开始运行新套件。



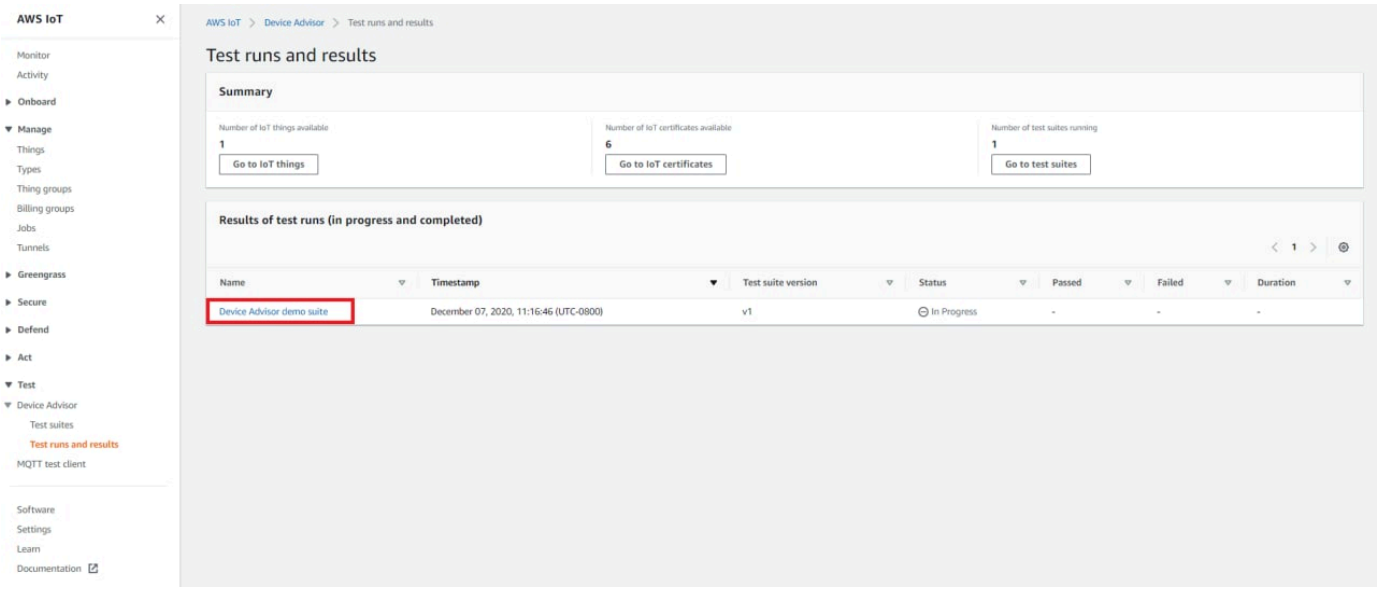
查看测试套件运行详细信息和日志

1. 在 AWS IoT 控制台的导航窗格中，展开 Test（测试）、Device Advisor，然后选择 Test run and results（测试运行和结果）。

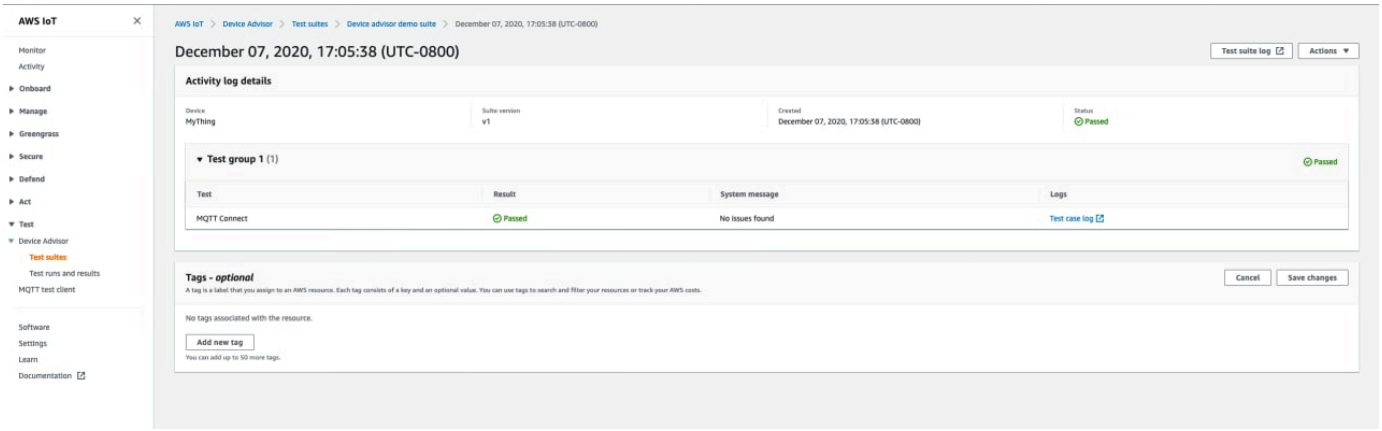
   本页面将显示：

   - IoT 事物的数量
   - IoT 证书的数量
   - 当前运行的测试套件数
   - 所有已创建的测试套件运行

2. 选择要查看其运行详细信息和日志的测试套件。



运行摘要页面显示当前测试套件运行的状态。此页面每 10 秒自动刷新一次。我们建议您为设备建立一个机制，以便每五秒便尝试一次连接到测试终端节点，每次持续一到两分钟。然后，您能够以自动的方式按顺序运行多个测试用例。



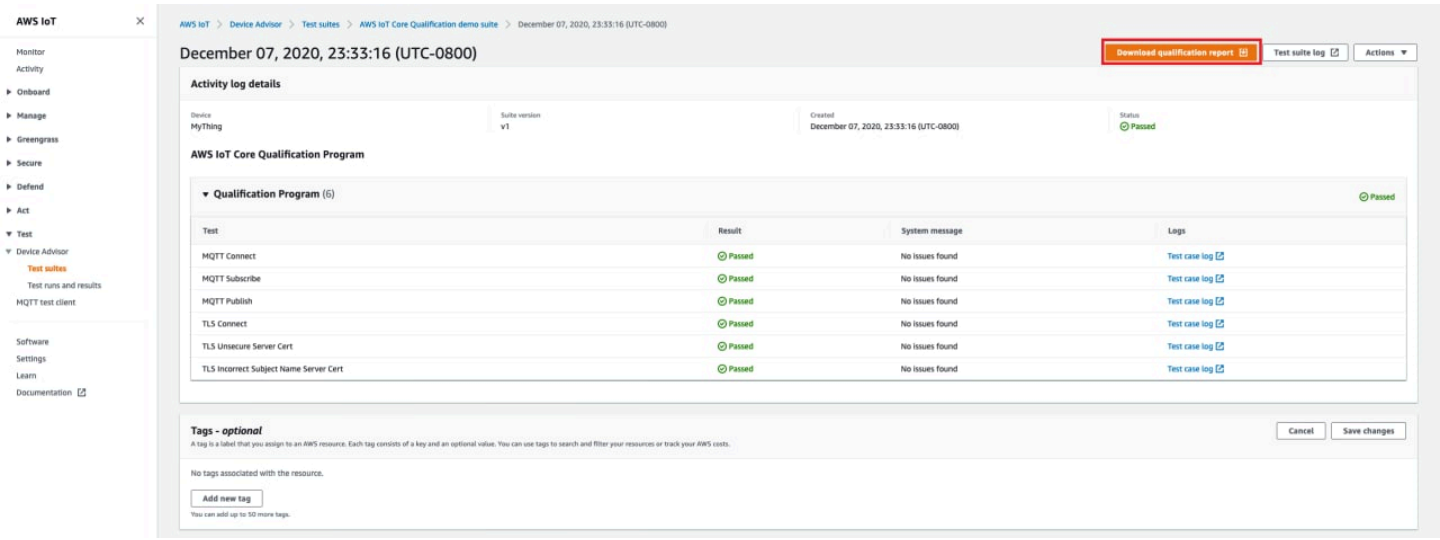
3. 要访问测试套件运行的 CloudWatch 日志，请选择测试套件日志。

   要访问任何测试用例的 CloudWatch 日志，请选择测试用例日志。

4. 根据您的测试结果，对您的设备进行故障排除操作，直到通过所有测试。

下载 AWS IoT 资格报告

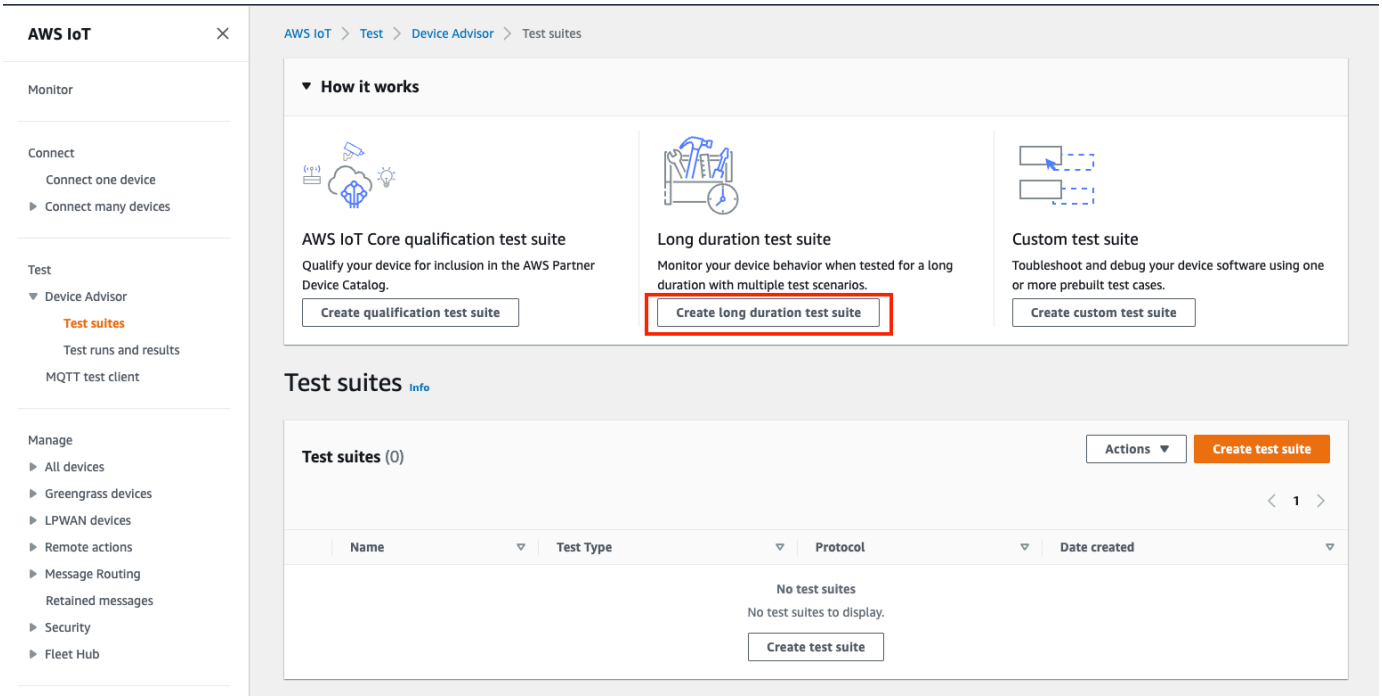
如果您在创建测试套件时选择了使用 AWS IoT 资格测试套件选项，并且能够运行资格测试套件，则可以通过在测试运行摘要页面中选择下载资格报告来下载资格报告。



长时间测试控制台的工作流程

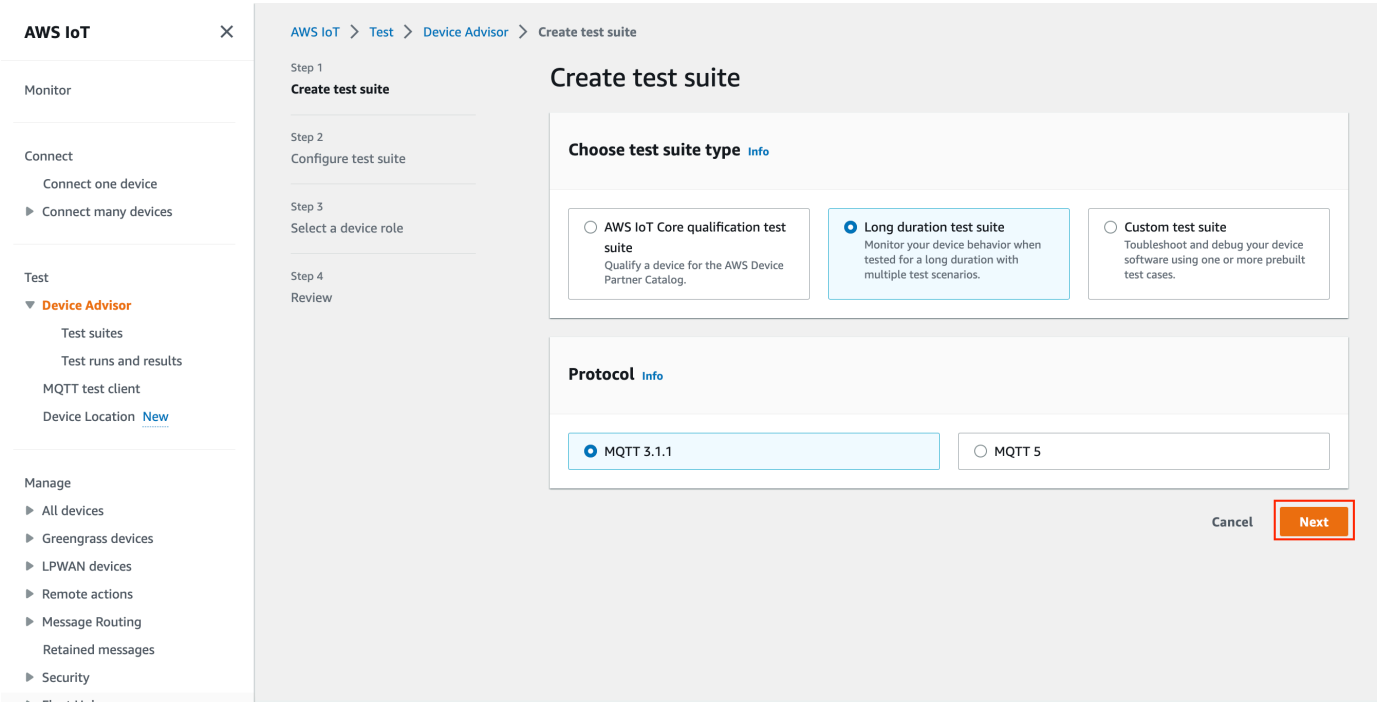
本教程可帮助您使用控制台开始在 Device Advisor 上进行长时间测试。要完成本教程，请执行设置中的步骤。

1. 在 AWS IoT 控制台的导航窗格中，依次展开 Test（测试）、Device Advisor 和 Test suites（测试套件）。在此页面上，选择 Create long duration test suite（创建长时间测试套件）。



2. 在 Create test suite（创建测试套件）页面上，选择 Long duration test suite（长时间测试套件），然后选择 Next（下一步）。

对于协议，选择 MQTT 3.1.1 或 MQTT 5。



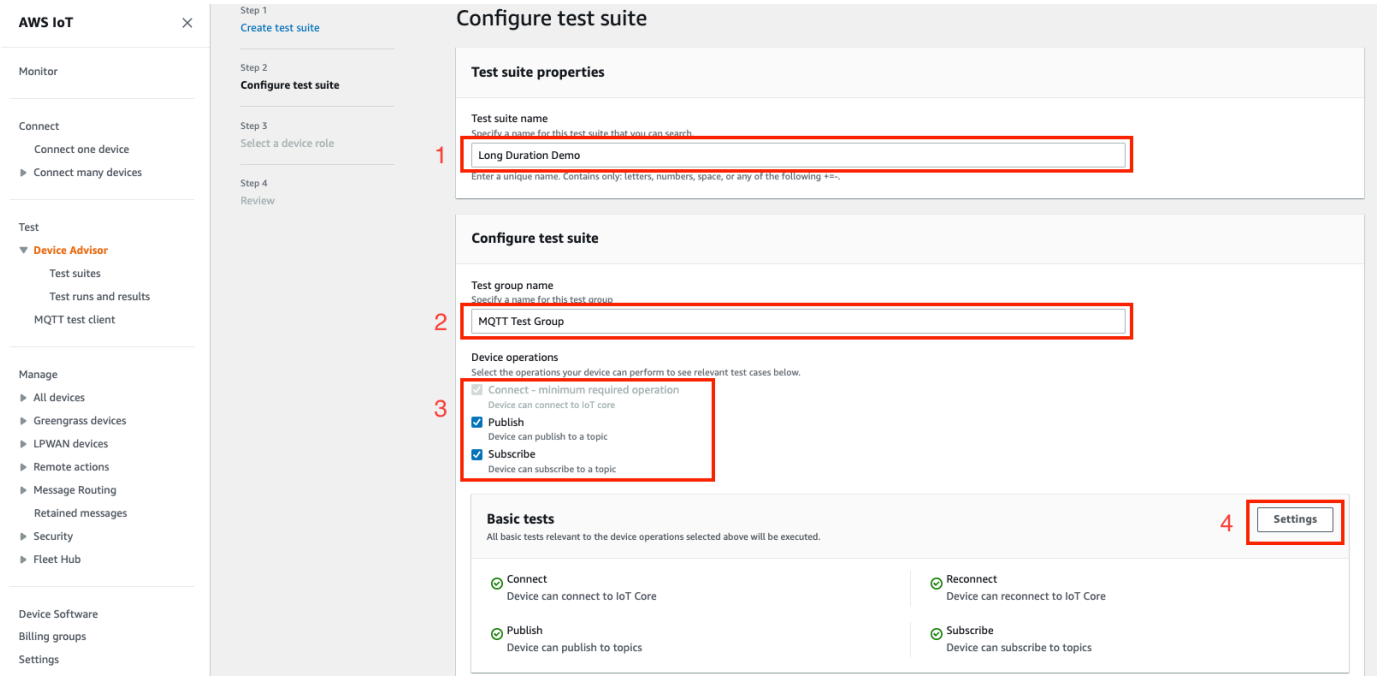
3. 在 Configure test suite（配置测试套件）页面上，执行以下操作：

   a. 更新 Test suite name（测试套件名称）字段。

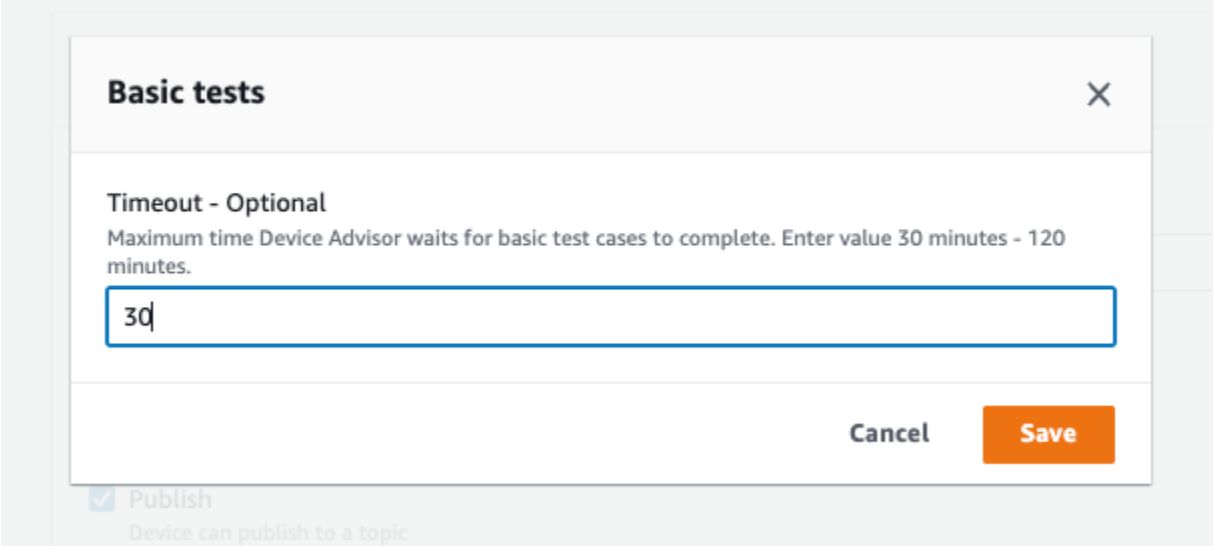
   b. 更新 Test group name（测试组名称）字段。

   c. 选择设备可以执行的 Device operations（设备操作）。这将选择要运行的测试。

   d. 选择 Settings（设置）选项。



4. （可选）输入 Device Advisor 必须等待基本测试完成的最长时间。选择 Save（保存）。



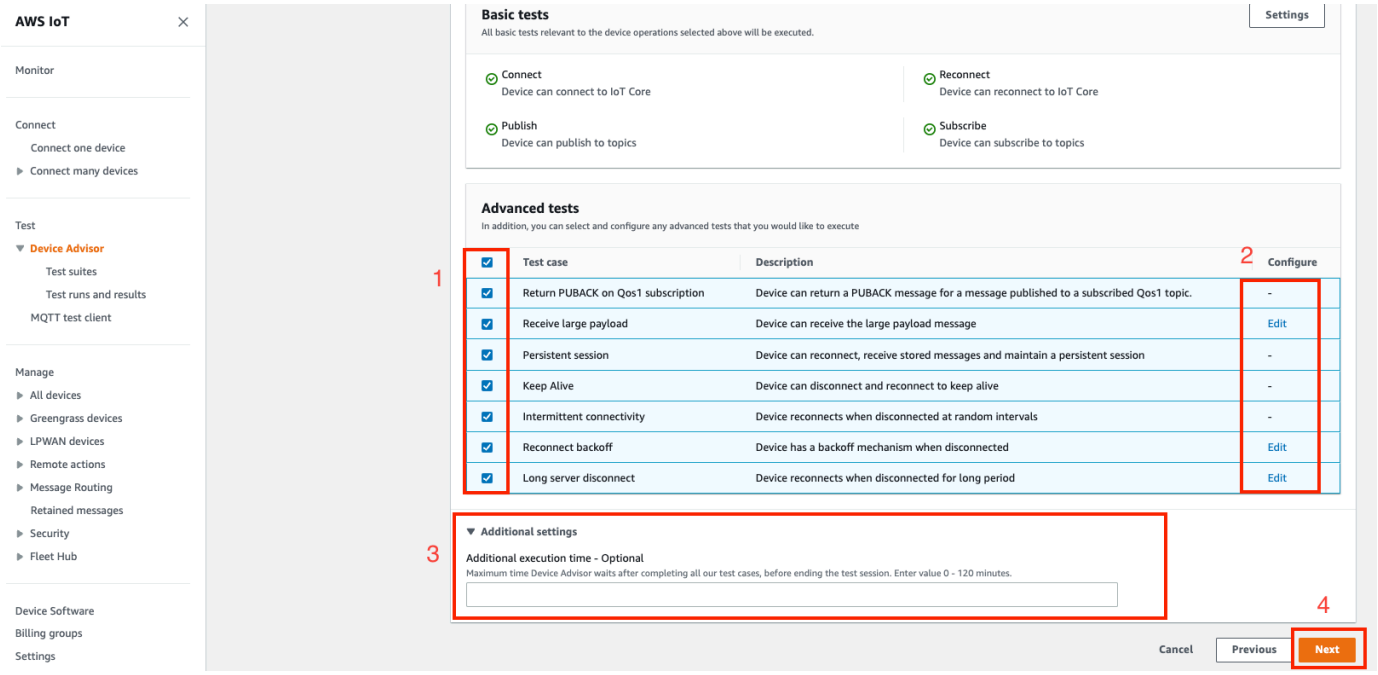
5. 在 Advanced tests（高级测试）和 Additional settings（其他设置）部分中执行以下操作。

   a. 选择或取消选择要在此测试中运行的 Advanced tests（高级测试）。

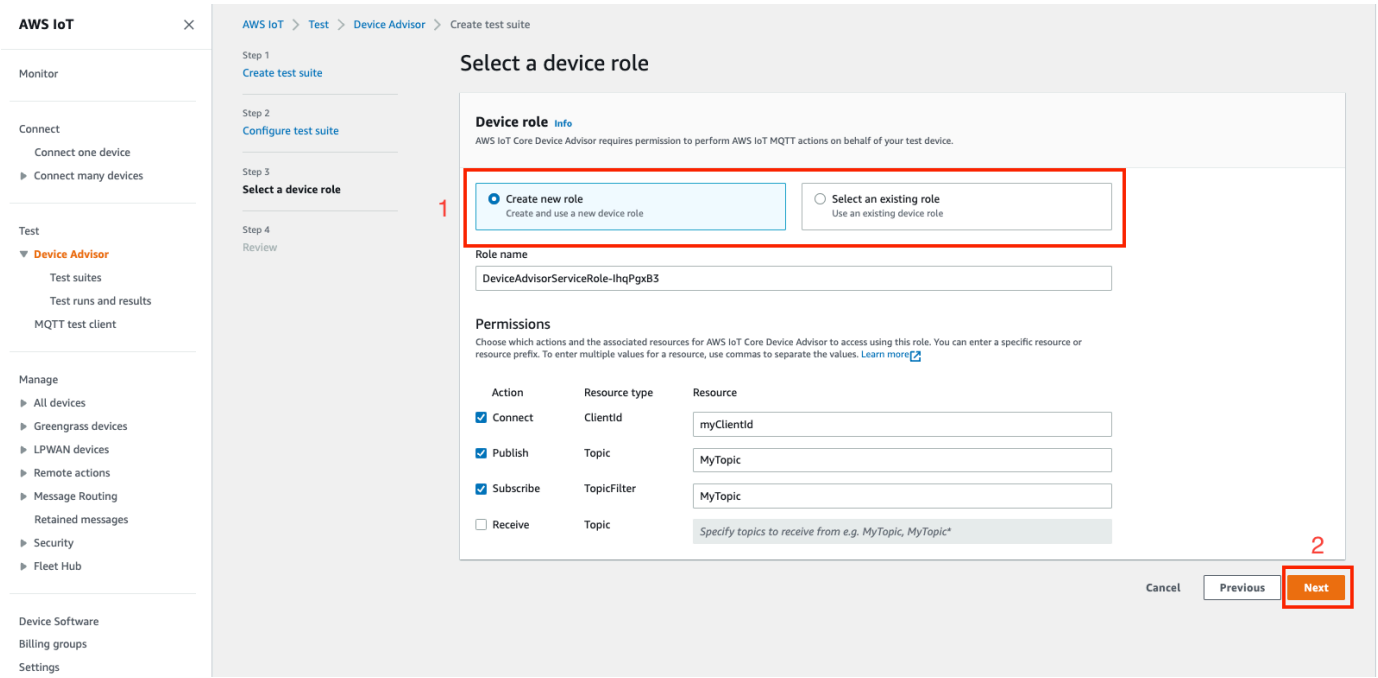
   b. 如果需要，Edit（编辑）测试的配置。

   c. 在 Additional settings（其他设置）部分下，配置 Additional execution time（其他执行时间）。

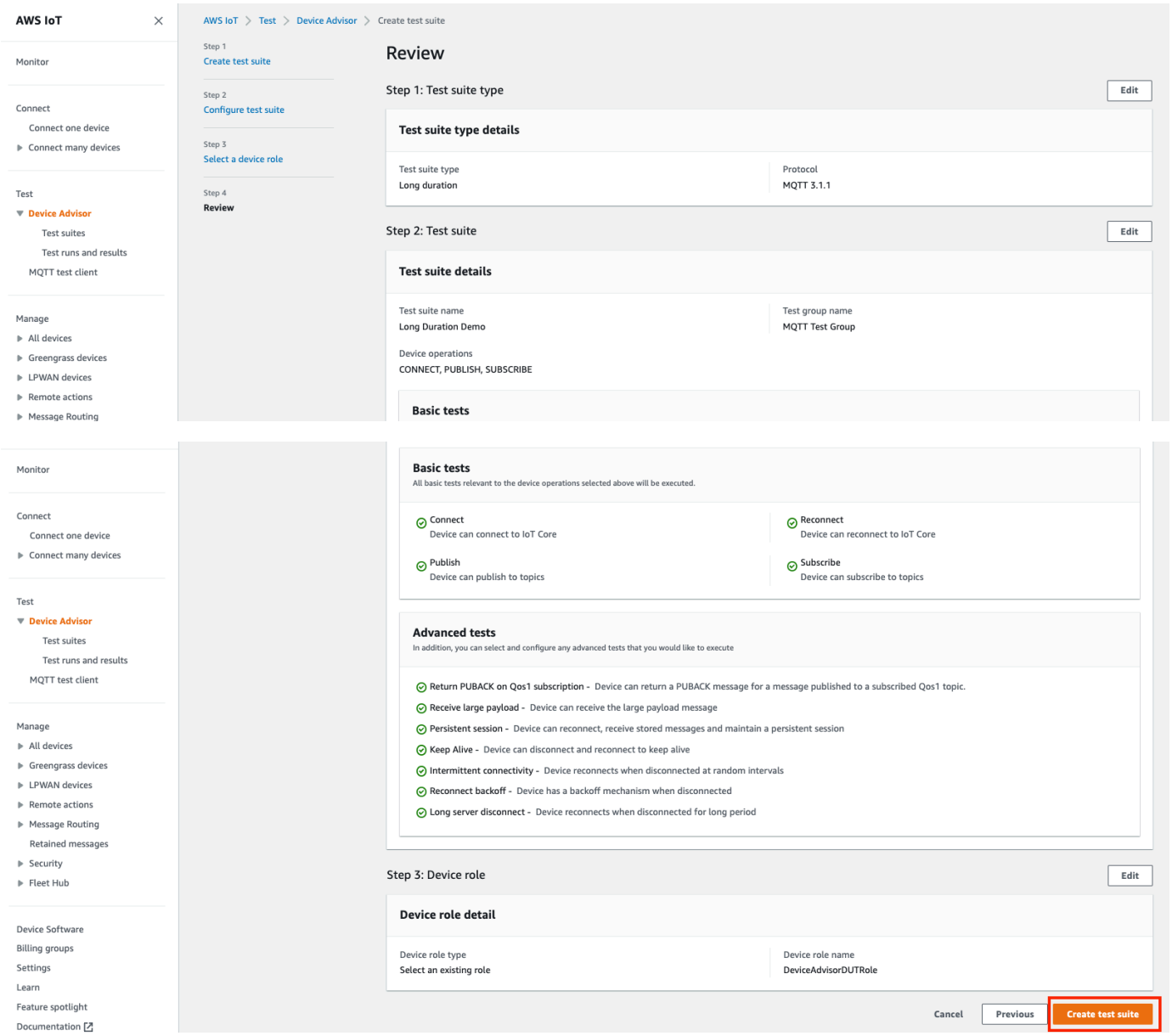
   d. 选择 Next（下一步）以继续执行下一步。



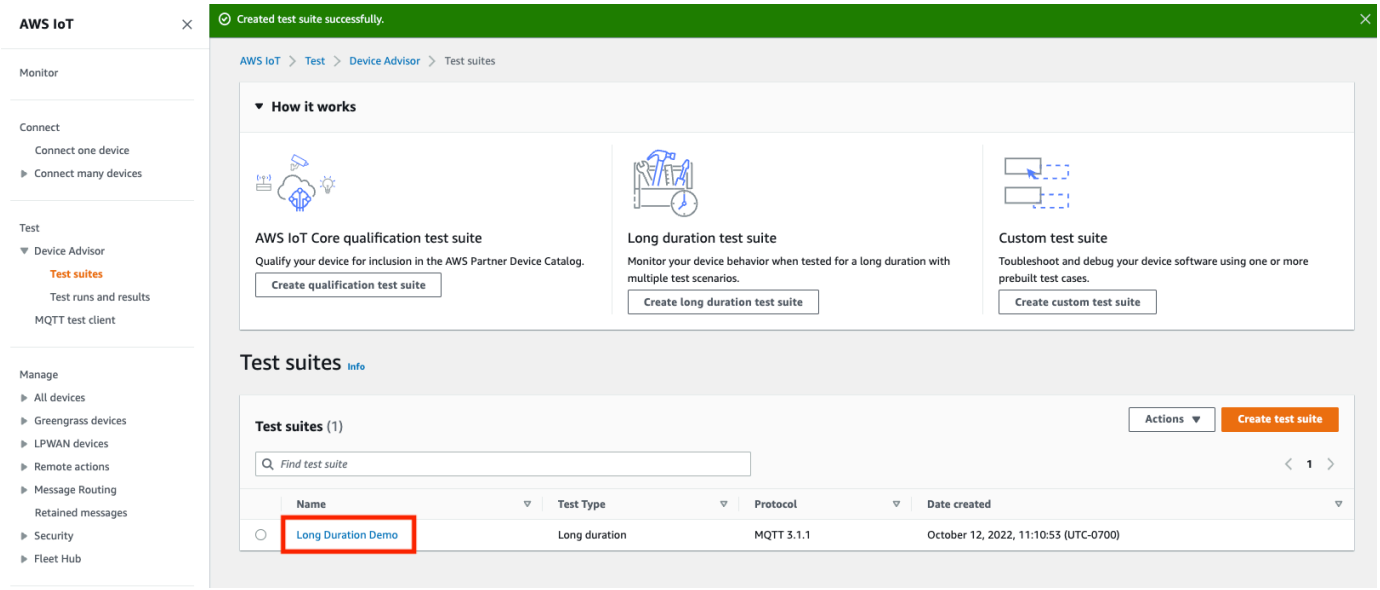
6. 在此步骤中，Create a new role（创建新角色）或 Select an existing role（选择现有角色）。有关详细信息，请参阅创建要用作设备角色的 IAM 角色。



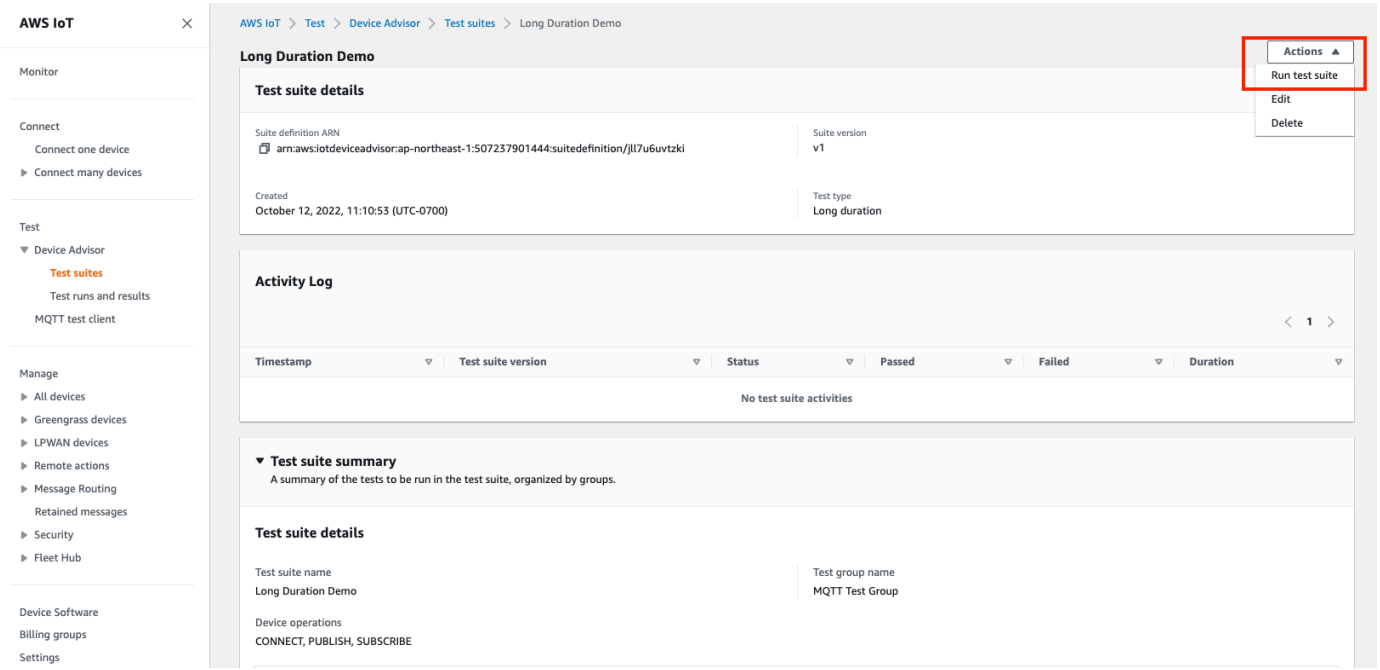
7. 查看在此步骤之前创建的所有配置，然后选择 Create test suite（创建测试套件）。



8. 创建的测试套件位于 Test suites（测试套件）部分下。选择此套件以查看详细信息。



9. 要运行创建的测试套件，请选择 Actions（操作），然后选择 Run test suite（运行测试套件）。

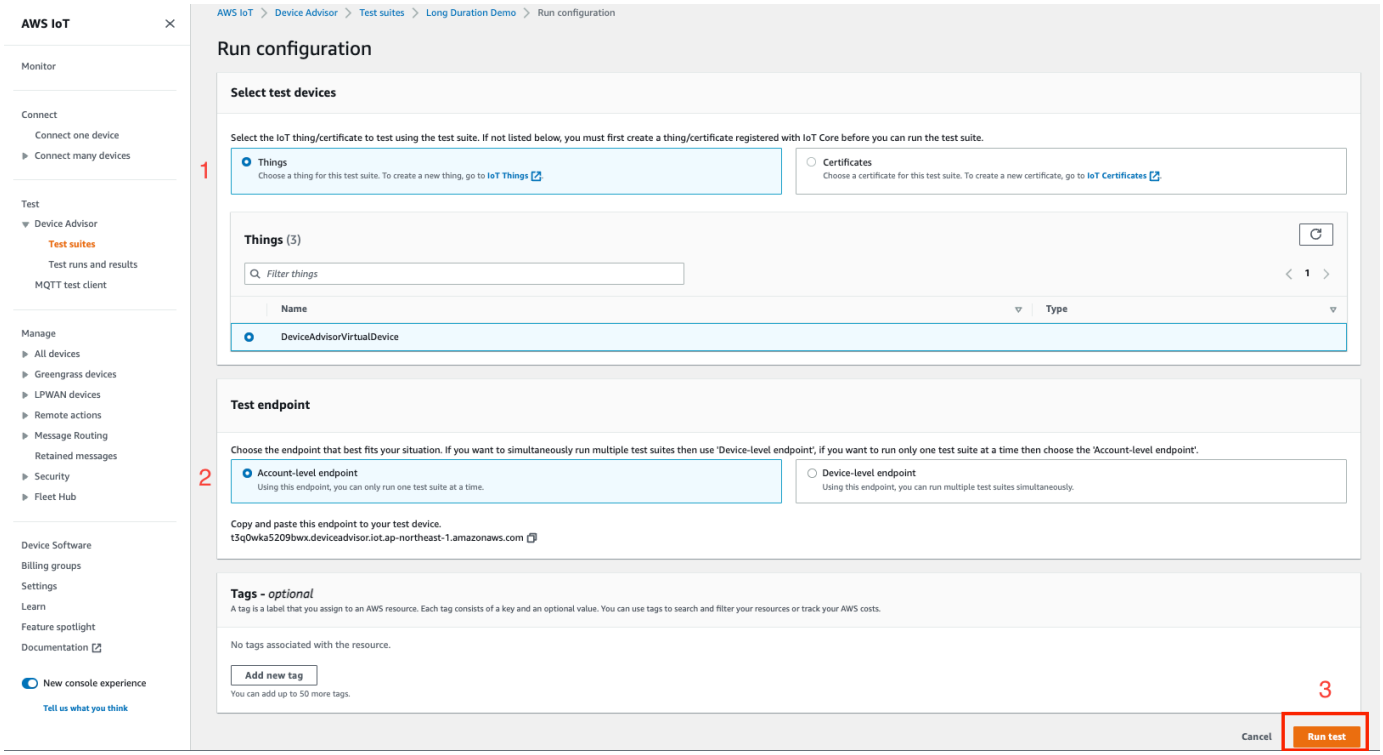


10. 在 Run configuration（运行配置）页面中选择配置选项。

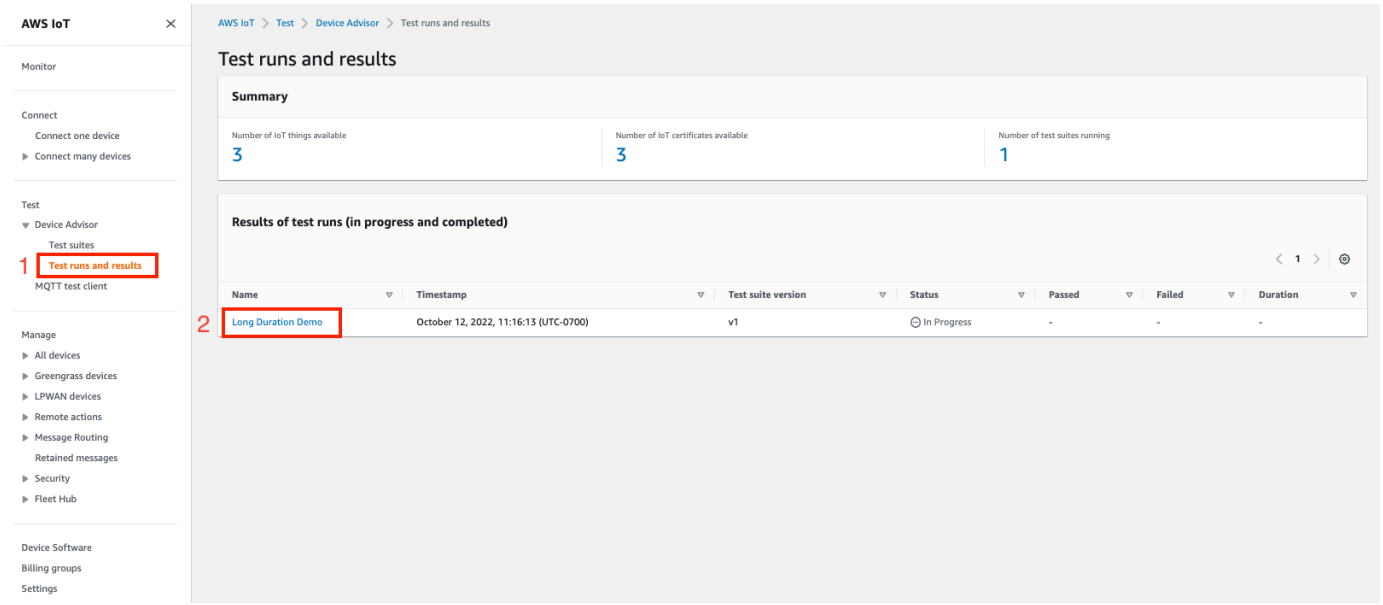
    a. 选择要运行测试的 Things（事物）或 Certificate（证书）。

    b. 选择 Account-level endpoint（账户级端点）或 Device-level endpoint（设备级端点）。

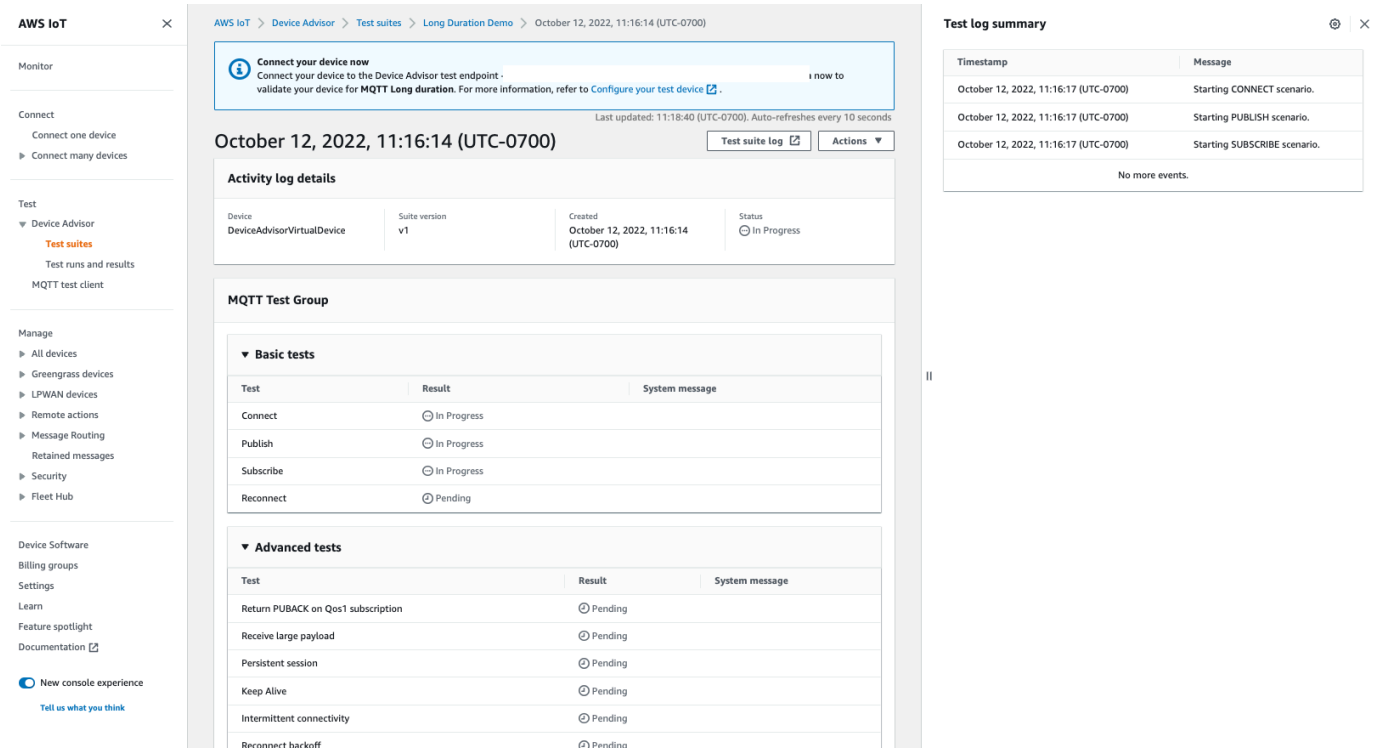
    c. 选择 Run test（运行测试）以运行此测试。



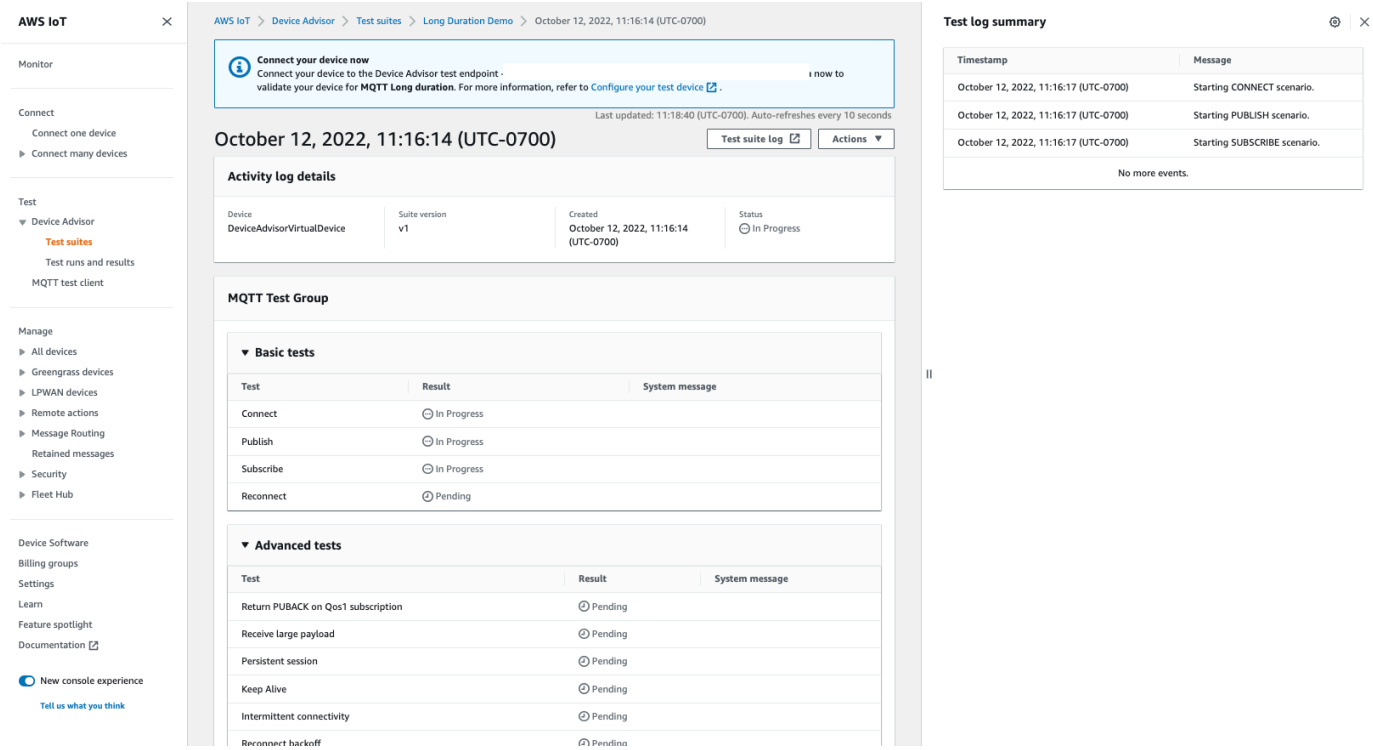
11. 要查看测试套件的运行结果，请在左侧导航窗格中选择 Test runs and results（测试运行和结果）。选择运行的测试套件以查看结果的详细信息。



12. 上一步将打开“test summary”（测试摘要）页面。测试运行的所有详细信息都显示在此页面中。当控制台提示启动设备连接时，将您的设备连接到提供的端点。在此页面上可以看到测试的进度。



13. 长时间测试在侧面板上提供了额外的 Test log summary（测试日志摘要），该摘要几乎实时显示设备和代理之间发生的所有重要事件。要查看更深入的详细日志，请单击 Test case log（测试用例日志）。



Device Advisor VPC 端点（AWS PrivateLink）

您可以通过创建接口 VPC 终端节点在您的 VPC 和 AWS IoT Core Device Advisor 测试终端节点（数据平面）之间建立私有连接。在将 AWS IoT 设备部署到生产环境 AWS IoT Core 之前，您可以使用此端点验证设备的可靠性和安全连接。Device Advisor 的预构建测试可帮助您根据最佳实践验证您的设备软件，以便使用 TLS、MQTT、设备影子和 AWS IoT Jobs。

AWS PrivateLink为物联网设备使用的接口端点提供动力。此服务可帮助您以私有方式访问 AWS IoT Core Device Advisor 测试端点，而无需互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。您的 VPC 中发送 TCP 和 MQTT 数据包的实例不需要公有 IP 地址即可与 AWS IoT Core Device Advisor 测试终端节点通信。您的 VPC 和 VPC 之间的流量 AWS IoT Core Device Advisor 不会离开 AWS Cloud。IoT 设备和 Device Advisor 测试案例之间的任何 TLS 和 MQTT 通信均在您 AWS 账户中的资源范围内。

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。

要了解有关使用接口 VPC 端点的更多信息，请参阅《Amazon VPC 用户指南》中的接口 VPC 端点（AWS PrivateLink）。

AWS IoT Core Device Advisor VPC 终端节点的注意事项

在设置接口 VPC 端点之前，请先查看《Amazon VPC 用户指南》中的接口端点属性和限制。继续之前，请注意以下各项：

- AWS IoT Core Device Advisor 目前支持从您的 VPC 调用设备顾问测试端点（数据平面）。消息代理使用数据面板通信发送和接收数据。这是在 TLS 和 MQTT 数据包的帮助下实现的。用于 AWS IoT Core Device Advisor 将您的 AWS IoT 设备连接到设备顾问测试端点的 VPC 终端节点。此 VPC 端点不使用控制面板 API 操作。要创建或运行测试套件或其他控制平面 API，请通过公共互联网使用控制台、 AWS SDK 或 AWS 命令行界面。
- 以下 AWS 区域 支持 VPC 终端节点 AWS IoT Core Device Advisor：
  - 美国东部（弗吉尼亚州北部）
  - 美国西部（俄勒冈）
  - 亚太地区（东京）
  - 欧洲地区（爱尔兰）
- Device Advisor 支持使用 X.509 客户端证书和 RSA 服务器证书的 MQTT。
- 目前不支持 VPC 端点策略。

- 查看 VPC 端点先决条件，了解有关如何创建连接 VPC 端点的资源的说明。您必须创建 VPC 和私有子网才能使用 AWS IoT Core Device Advisor VPC 终端节点。
- 您的 AWS PrivateLink 资源有配额。有关更多信息，请参阅 AWS PrivateLink 配额。
- VPC 端点仅支持 IPv4 流量。

为 AWS IoT Core Device Advisor创建接口 VPC 终端节点

要开始使用 VPC 端点，请创建接口 VPC 端点。接下来，选择 AWS IoT Core Device Advisor 作为 AWS 服务。如果您使用 AWS IoT Core Device Advisor 的是 AWS CLI，请describe-vpc-endpoint-services致电确认您的可用区中是否存在 AWS 区域。确认连接到端点的安全组允许对 MQTT 和 TLS 流量使用 TCP 协议通信。例如，在美国东部（弗吉尼亚州北部）区域中，使用以下命令：

```
aws ec2 describe-vpc-endpoint-services --service-name com.amazonaws.us-
east-1.deviceadvisor.iot
```

您可以使用以下服务名称创建 VPC 终端节点： AWS IoT Core

- com.amazonaws.region.deviceadvisor.iot

默认情况下，对于此端点开启私有 DNS。这确保默认测试端点的使用保持在您的私有子网内。要获取您的账户或设备级终端节点，请使用控制台 AWS CLI 或 S AWS DK。例如，如果您在公有子网或公共互联网上运行 get-endpoint，则可以获取您的端点并使用它来连接到 Device Advisor。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。

为了将 MQTT 客户端连接到 VPC 终端节点接口，该 AWS PrivateLink 服务会在连接到您的 VPC 的私有托管区域中创建 DNS 记录。这些 DNS 记录将 AWS IoT 设备的请求定向到 VPC 端点。

控制 AWS IoT Core Device Advisor 通过 VPC 终端节点的访问权限

您可以使用 VPC 条件上下文密钥限制设备访问 AWS IoT Core Device Advisor 和仅允许通过 VPC 终端节点进行访问。 AWS IoT Core 支持以下与 VPC 相关的上下文密钥：

- SourceVpc
- SourceVpce
- VPCSourceIp

> **Note**
>
> AWS IoT Core Device Advisor 目前不支持 VPC 终端节点策略。

以下策略授予 AWS IoT Core Device Advisor 使用与事物名称匹配的客户端 ID 进行连接的权限。它还会发布到任何以事物名称为前缀的主题。该策略以设备连接到具有特定 VPC 端点 ID 的 VPC 端点为条件。此策略拒绝连接到您的公有 AWS IoT Core Device Advisor 测试端点的尝试。

```
{
"Version": "2012-10-17",
    "Statement": [
        {
"Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/
${iot:Connection.Thing.ThingName}"
            ],
            "Condition": {
"StringEquals": {
"aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }

        },
        {
"Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingName}/*"
            ]
        }
    ]
}
```

Device Advisor 测试用例

Device Advisor 提供六个类别的预构建测试。

- TLS
- MQTT
- 影子
- 任务执行
- 权限与策略
- 长时间测试

符合设备资格认证计划的 AWS 设备顾问测试用例。

根据 AWS 设备资格认证计划，您的设备必须通过以下测试才符合资格。

> **Note**
>
> 这是一份经修订的资格认证测试列表。

- TLS Connect（“TLS Connect”）
- TLS Incorrect Subject Name Server Cert（TLS 不正确的主题名称服务器证书）("Incorrect Subject Common Name (CN) / Subject Alternative Name (SAN)")（“不正确的主题通用名称 (CN)/主题备用名称 (SAN)”）
- TLS Unsecure Server Cert（TLS 不安全的服务器证书）("Not Signed By Recognized CA")（“未被认可的 CA 签名”）
- AWS IoT 密码套件的 TLS 设备支持（“ AWS IoT 推荐密码套件的 TLS 设备支持”）
- TLS 接收最大大小片段（“TLS 接收最大大小片段”）
- TLS 已过期的服务器证书（“已过期的服务器证书”）
- TLS 较大服务器证书（“TLS 较大服务器证书”）
- MQTT Connect（“设备发送连接到 AWS IoT Core （ Happy case ）”）
- MQTT 订阅 ("Can Subscribe (Happy Case)")（“可以订阅（ 快乐用例 ）”）
- MQTT 发布 ("QoS0 (Happy Case)")（“QoS0（ 快乐用例 ）”）
- MQTT 连接抖动重试次数（“使用抖动退避功能重试设备连接 - 无 CONNACK 响应”）

TLS

使用这些测试来确定您的设备和之间的传输层安全协议 (TLS) AWS IoT 是否安全。

> **Note**
>
> Device Advisor 现在支持 TLS 1.3。

满意路径

TLS 连接

验证被测设备能否完成 TLS 握手。 AWS IoT此测试不会验证客户端设备的 MQTT 执行。

Example API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。为了获得最佳效果，我们建议超时值为 2 分钟。

```
"tests":[
   {
      "name":"my_tls_connect_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Connect",
         "version":"0.0.0"
      }
   }
]
```

Example 测试用例输出：

- 通过 — 被测设备已完成 TLS 握手 AWS IoT。

- 通过但有警告 ——被测设备完成了 TLS 握手 AWS IoT，但设备或 AWS IoT发来了 TLS 警告消息。
- 失败 — AWS IoT 由于握手错误，被测设备未能完成 TLS 握手。

TLS 接收最大大小片段

此测试用例验证您的设备是否能够接收和处理 TLS 最大大小片段。您的测试设备必须使用 QoS 1 订阅预配置的主题，才能接收大的有效负载。您可以使用配置 `${payload}` 自定义有效负载。

Example API 测试用例定义：

> **Note**
>
> `EXECUTION_TIMEOUT` 的默认值为 5 分钟。为了获得最佳效果，我们建议超时值为 2 分钟。

```
"tests":[
   {
      "name":"TLS Receive Maximum Size Fragments",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
         "PAYLOAD_FORMAT":"{"message":"${payload}"}", // A string with a placeholder
 ${payload}, or leave it empty to receive a plain string.
         "TRIGGER_TOPIC": "test_1" // A topic to which a device will subscribe, and
 to which a test case will publish a large payload.
      },
      "test":{
         "id":"TLS_Receive_Maximum_Size_Fragments",
         "version":"0.0.0"
      }
   }
]
```

密码套件

AWS IoT 推荐的密码套件的 TLS Device Support

验证来自受测设备的 TLS Client Hello 消息中的密码套件是否包含建议的 AWS IoT 密码套件。它提供了对设备支持的密码套件的更多见解。

Example API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。

```
"tests":[
   {
      "name":"my_tls_support_aws_iot_cipher_suites_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"TLS_Support_AWS_IoT_Cipher_Suites",
         "version":"0.0.0"
      }
   }
]
```

Example 测试用例输出：

- 通过 — 被测试的设备密码套件至少包含一个推荐的 AWS IoT 密码套件，并且不包含任何不支持的密码套件。
- 通过但带有警告 — 设备密码套件至少包含一个 AWS IoT 密码套件，但是：

  1. 它不包含任何建议的密码套件。
  2. 它包含不支持的密码套件。 AWS IoT

  我们建议您验证任何不受支持的密码套件是否安全。
- 失败 — 被测设备密码套件不包含任何 AWS IoT 支持的密码套件。

较大的服务器证书

TLS 较大服务器证书

验证您的设备在接收和处理较大的服务器证书时，是否可以完成与 AWS IoT 的 TLS 握手。此测试使用的服务器证书的大小（以字节为单位）比 TLS Connect 测试用例和 IoT Core 中当前使用的证

书大小 20 在此测试用例中，请 AWS IoT 测试设备的 TLS 缓冲空间如果缓冲空间足够大，则 TLS 握手可以顺利完成。此测试不验证设备的 MQTT 实施。测试用例在 TLS 握手过程完成后结束。

Example API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。为了获得最佳效果，我们建议超时值为 2 分钟。如果此测试用例失败但 TLS 连接测试用例通过，我们建议您提高设备的 TLS 缓冲区空间限制。提高缓冲空间限制可确保设备在服务器证书大小增加时可以处理大小更大的服务器证书。

```
"tests":[
   {
      "name":"my_tls_large_size_server_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"TLS_Large_Size_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example 测试用例输出：

- 通过 — 被测设备完成了与 AWS IoT的 TLS 握手。
- 通过但有警告 ——被测设备完成了 TLS 握手 AWS IoT，但设备或 AWS IoT TLS 警告消息来自该设备。
- 失败 — AWS IoT 由于握手过程中出现错误，被测设备未能完成 TLS 握手。

TLS 不安全服务器证书

未被认可的 CA 签名

验证在向受测设备提供没有 ATS CA 有效签名的服务器证书时，该设备是否关闭连接。设备只能连接到提供有效证书的终端节点。

Example API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。

```
"tests":[
   {
      "name":"my_tls_unsecure_server_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Unsecure_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example 测试用例输出：

- 通过 — 被测设备关闭了连接。
- 失败 — 被测设备已完成 TLS 握手 AWS IoT。

TLS 不正确的主题名称服务器证书/不正确的主题通用名称（CN）/主题备用名称（SAN）

验证如果提供与请求的域名不同的域名服务器证书，被测设备是否会关闭连接。

Example API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。

```
"tests":[
   {
      "name":"my_tls_incorrect_subject_name_cert_test",
      "configuration": {
```

```
         // optional:
         "EXECUTION_TIMEOUT":"300",   // in seconds
      },
      "test":{
         "id":"TLS_Incorrect_Subject_Name_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example 测试用例输出：

- 通过 — 被测设备关闭了连接。
- 失败 — 被测设备完成了 TLS 握手 AWS IoT。

TLS 已过期的服务器证书

已过期的服务器证书

验证如果被测试的设备的服务器证书已过期，该设备是否关闭连接。

Example API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。

```
"tests":[
   {
      "name":"my_tls_expired_cert_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",  //in seconds
      },
      "test":{
         "id":"TLS_Expired_Server_Cert",
         "version":"0.0.0"
      }
   }
]
```

Example 测试用例输出：

- 通过 — 被测设备拒绝完成 TLS 握手。 AWS IoT设备在关闭连接之前发送 TLS 警报消息。
- 通过但带有警告 — 受测设备拒绝完成与 AWS IoT的 TLS 握手。但是，它在关闭连接之前不会发送 TLS 警报消息。
- 失败 — 被测设备使用完成 TLS 握手 AWS IoT。

MQTT

CONNECT、DISCONNECT 和 RECONNECT

“设备发送 CONNECT 到 AWS IoT Core （ Happy case ）”

验证被测设备是否发送 CONNECT 请求。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。

```
"tests":[
   {
      "name":"my_mqtt_connect_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",   // in seconds
      },
      "test":{
         "id":"MQTT_Connect",
         "version":"0.0.0"
      }
   }
]
```

“设备可以将 PUBACK 返回到 QoS1 的任意主题”

此测试用例将检查如果设备（客户端）在通过 QoS1 订阅主题后收到来自代理的发布消息，它能否返回 PUBACK 消息。

可针对此测试用例配置负载内容和负载大小。如果配置了负载大小，Device Advisor 将覆盖负载内容的值，并将预定义的具有所需大小的负载发送到设备。负载大小的值介于 0 到 128 之间，不能超过 128KB。 AWS IoT Core 拒绝大于 128KB 的发布和连接请求，如 AWS IoT Core 消息代理及协议限制和限额页面所示。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议超时值为 2 分钟。PAYLOAD_SIZE 可以配置为 0 到 128KB 之间的值。定义负载大小会覆盖负载内容，因为 Device Advisor 会将具有给定大小的预定义负载发送回设备。

```
"tests":[
{
        "name":"my_mqtt_client_puback_qos1",
        "configuration": {
            // optional:"TRIGGER_TOPIC": "myTopic",
            "EXECUTION_TIMEOUT":"300", // in seconds
            "PAYLOAD_FOR_PUBLISH_VALIDATION":"custom payload",
            "PAYLOAD_SIZE":"100" // in kilobytes
        },
        "test": {
            "id": "MQTT_Client_Puback_QoS1",
            "version": "0.0.0"
        }
    }
]
```

"Device re-connect with jitter backoff - No CONNACK response"（“设备以抖动退避功能重新连接 - 没有 CONNACK 响应”）

验证被测设备与代理重新连接至少五次时是否使用了正确的抖动退避。代理记录被测设备的 CONNECT 请求时间戳，执行数据包验证，暂停而不向被测设备发送 CONNACK，并等待被测设备重新发送请求。允许第六次连接尝试通过，并允许 CONNACK 流回到被测设备。

将再次执行上述过程。总体而言，此测试用例要求设备总共至少连接 12 次。收集的时间戳用于验证被测设备是否使用了抖动退避。如果被测设备具有严格的指数退避延迟，则此测试用例将通过但带有警告。

我们建议实施指数退避和抖动在所测试设备上的机制通过此测试案例。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 4 分钟。

```
"tests":[
   {
      "name":"my_mqtt_jitter_backoff_retries_test",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300",    // in seconds
      },
      "test":{
         "id":"MQTT_Connect_Jitter_Backoff_Retries",
         "version":"0.0.0"
      }
   }
]
```

“设备使用指数退避重新连接 - 没有 CONNACK 响应”

验证被测设备与代理重新连接至少五次时是否使用了正确的指数退避。代理记录被测设备的 CONNECT 请求时间戳，执行数据包验证，暂停而不向客户端设备发送 CONNACK，并等待被测设备重新发送请求。收集的时间戳用于验证被测设备是否使用了指数退避。

我们建议实施指数退避和抖动在所测试设备上的机制通过此测试案例。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 4 分钟。

```
"tests":[
   {
      "name":"my_mqtt_exponential_backoff_retries_test",
```

```
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"600",  // in seconds
      },
      "test":{
         "id":"MQTT_Connect_Exponential_Backoff_Retries",
         "version":"0.0.0"
      }
   }
]
```

“使用抖动退避重新连接设备-服务器断开连接后”

验证正在测试的设备在与服务器断开连接后重新连接时是否使用必要的抖动和回退。Device Advisor 至少将设备与服务器断开五次，并观察设备的 MQTT 重新连接的行为。Device Advisor 记录被测设备的连接请求的时间戳，执行数据包验证，暂停而不向客户端设备发送连接，并等待被测设备重新发送请求。收集的时间戳用于验证被测设备在重新连接时是否使用抖动和退避。如果被测设备具有严格的指数退避或未实现适当的抖动退避机制，此测试用例将通过但是带有警告。如果被测设备实施了线性退避或恒定退避机制，测试将失败。

要通过此测试用例，我们建议在所测试设备上实施指数退避和抖动机制。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 4 分钟。
> 通过指定 RECONNECTION_ATTEMPTS，可以更改验证退避的重新连接尝试次数。该数字必须介于 5 到 10 之间。默认值是 5。

```
"tests":[
   {
      "name":"my_mqtt_reconnect_backoff_retries_on_server_disconnect",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "RECONNECTION_ATTEMPTS": 5
      },
      "test":{
         "id":"MQTT_Reconnect_Backoff_Retries_On_Server_Disconnect",
```

```
         "version":"0.0.0"
      }
   }
]
```

"Device re-connect with jitter backoff - On unstable connection" (“设备以抖动退避功能重新连接 - 连接不稳定”)

验证被测设备在不稳定连接上重新连接时是否使用了必要的抖动和退避。Device Advisor 在五次成功连接后将设备与服务器断开连接，并观察设备的 MQTT 重新连接的行为。Device Advisor 记录被测设备的 CONNECT (连接) 请求的时间戳，执行数据包验证，发送回 CONNACK，断开连接，记录断开连接的时间戳，并等待被测设备重新发送请求。收集的时间戳用于验证被测设备在成功但不稳定的连接后重新连接时，是否使用抖动和退避。如果被测设备具有严格的指数退避或未实现适当的抖动退避机制，此测试用例将通过但是带有警告。如果被测设备实施了线性退避或恒定退避机制，测试将失败。

要通过此测试用例，我们建议在所测试设备上实施指数退避和抖动机制。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 4 分钟。
> 通过指定 RECONNECTION_ATTEMPTS，可以更改验证退避的重新连接尝试次数。该数字必须介于 5 到 10 之间。默认值是 5。

```
"tests":[
   {
      "name":"my_mqtt_reconnect_backoff_retries_on_unstable_connection",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "RECONNECTION_ATTEMPTS": 5
      },
      "test":{
         "id":"MQTT_Reconnect_Backoff_Retries_On_Unstable_Connection",
         "version":"0.0.0"
      }
   }
]
```

Publish

"QoS0 (Happy Case)" (“QoS0（快乐用例）”)

验证测试设备是使用 QoS0 还是 QoS1 发布消息。您还可以通过在测试设置中指定此主题值和有效负载来验证消息的主题和有效负载。

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。

```
"tests":[
   {
      "name":"my_mqtt_publish_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_FOR_PUBLISH_VALIDATION": "my_TOPIC_FOR_PUBLISH_VALIDATION",
         "PAYLOAD_FOR_PUBLISH_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish",
         "version":"0.0.0"
      }
   }
]
```

"QoS1 publish retry - No PUBACK" (“QoS1 发布重试 - 没有 PUBACK”)

如果代理没有发送 PUBACK，则验证被测设备是否重新发布了随 QoS1 发送的消息。您还可以通过在测试设置中指定此主题来验证消息的主题。在重新发布消息之前，客户端设备不得断开连接。此测试还验证重新发布的消息与原始消息是否具有相同的数据包标识符。在测试执行期间，如果设备丢失连接并重新连接，则测试用例将顺利重置，并且设备必须重新执行测试用例步骤。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。建议至少 4 分钟。

```
"tests":[
   {
      "name":"my_mqtt_publish_retry_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_FOR_PUBLISH_VALIDATION": "my_TOPIC_FOR_PUBLISH_VALIDATION",
         "PAYLOAD_FOR_PUBLISH_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_VALIDATION",
      },
      "test":{
         "id":"MQTT_Publish_Retry_No_Puback",
         "version":"0.0.0"
      }
   }
]
```

"Publish Retained messages"（“发布保留消息”）

验证被测设备是否发布 `retainFlag` 设置为 true 的消息。您可以通过在测试设置中设置主题值和有效负载来验证消息的主题和有效负载。如果未将 PUBLISH 数据包中发送的 `retainFlag` 设置为 true，则测试用例将失败。

API 测试用例定义：

> **Note**
>
> `EXECUTION_TIMEOUT` 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。要运行此测试用例，请在您的 device role（设备角色）中添加 `iot:RetainPublish` 操作。

```
"tests":[
   {
      "name":"my_mqtt_publish_retained_messages_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds

 "TOPIC_FOR_PUBLISH_RETAINED_VALIDATION": "my_TOPIC_FOR_PUBLISH_RETAINED_VALIDATION",

 "PAYLOAD_FOR_PUBLISH_RETAINED_VALIDATION": "my_PAYLOAD_FOR_PUBLISH_RETAINED_VALIDATION",
      },
```

```
      "test":{
         "id":"MQTT_Publish_Retained_Messages",
         "version":"0.0.0"
      }
   }
]
```

"Publish with User Property" (“使用用户属性发布”)

验证被测设备是否使用正确的用户属性发布消息。您可以通过在测试设置中设置名称-值对来验证用户属性。如果未提供用户属性或用户属性不匹配，则测试用例将失败。

API 测试用例定义：

> **Note**
>
> 这是仅限 MQTT5 的测试用例。
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。

```
"tests":[
   {
      "name":"my_mqtt_user_property_test",
      "test":{
        "USER_PROPERTIES": [
            {"name": "name1", "value":"value1"},
            {"name": "name2", "value":"value2"}
        ],
        "EXECUTION_TIMEOUT":"300",  // in seconds
      },
      "test":{
         "id":"MQTT_Publish_User_Property",
         "version":"0.0.0"
      }
   }
]
```

订阅

"Can Subscribe (Happy Case)" (“可以订阅（快乐用例）”)

验证被测设备是否订阅了 MQTT 主题。您还可以通过在测试设置中指定此主题来验证被测设备订阅的主题。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。

```
"tests":[
   {
      "name":"my_mqtt_subscribe_test",
      "configuration":{
         // optional:
         "EXECUTION_TIMEOUT":"300",  // in seconds
         "TOPIC_LIST_FOR_SUBSCRIPTION_VALIDATION":
["my_TOPIC_FOR_PUBLISH_VALIDATION_a","my_TOPIC_FOR_PUBLISH_VALIDATION_b"]
      },
      "test":{
         "id":"MQTT_Subscribe",
         "version":"0.0.0"
      }
   }
]
```

"Subscribe Retry - No SUBACK" (“订阅重试 - 无 SUBACK”)

验证被测设备是否重试 MQTT 主题的失败订阅。然后，服务器会等待，不发送 SUBACK。如果客户端设备未重试订阅，则测试失败。客户端设备必须使用相同的数据包 ID 重试失败的订阅。您还可以通过在测试设置中指定此主题来验证被测设备订阅的主题。在测试执行期间，如果设备丢失连接并重新连接，则测试用例将顺利重置，并且设备必须重新执行测试用例步骤。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 4 分钟。

```
"tests":[
   {
      "name":"my_mqtt_subscribe_retry_test",
      "configuration":{
         "EXECUTION_TIMEOUT":"300",  // in seconds
         // optional:
         "TOPIC_LIST_FOR_SUBSCRIPTION_VALIDATION":
["myTOPIC_FOR_PUBLISH_VALIDATION_a","my_TOPIC_FOR_PUBLISH_VALIDATION_b"]
      },
      "test":{
         "id":"MQTT_Subscribe_Retry_No_Suback",
         "version":"0.0.0"
      }
   }
]
```

Keep-Alive

“Mattt No Ack PingResp”

此测试用例验证被测设备在未收到 ping 响应时是否断开连接。作为此测试用例的一部分，设备顾问会屏蔽来自 AWS IoT Core 的发布、订阅和 ping 请求的响应。它还验证正在测试的设备是否断开了 MQTT 连接。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议超时时间大于 keepAliveTime 值的 1.5 倍。
> 此测试的最大 keepAliveTime 不得超过 230 秒。

```
"tests":[
```

```
    {
       "name":"Mqtt No Ack PingResp",
       "configuration":
          //optional:
          "EXECUTION_TIMEOUT":"306",   // in seconds
       },
       "test":{
          "id":"MQTT_No_Ack_PingResp",
          "version":"0.0.0"
       }
    }
]
```

持久会话

“持久会话（快乐用例）”

此测试用例验证设备在与持久会话断开连接时的行为。该测试用例检查设备是否可以重新连接，恢复对其触发器主题的订阅而无需显式重新订阅，接收主题中存储的消息以及在持久会话期间按预期工作。当此测试用例通过时，它表明客户端设备能够以预期的方式与 AWS IoT Core 代理保持持续会话。有关 AWS IoT 持久会话的更多信息，请参阅使用 MQTT 持久会话。

在此测试用例中，客户端设备应该与 AWS IoT Core 进行连接 [clean session（清理会话）标志设置为 false]，然后订阅一个触发器主题。成功订阅后，设备顾问将断开 AWS IoT Core 设备连接。当设备处于断开连接状态时，该主题中将存储 QoS 1 消息负载。然后，Device Advisor 将允许客户端设备与测试终端节点重新连接。此时，由于存在持久会话，客户端设备应该恢复其主题订阅而不发送任何其他 SUBSCRIBE 数据包，然后接收来自代理的 QoS 1 消息。重新连接后，如果客户端设备通过发送额外的 SUBSCRIBE 数据包再次订阅其主题触发器和/或客户端未能从触发器主题接收存储的消息，则测试用例将失败。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为至少 4 分钟。在第一个连接中，客户端设备需要显式订阅之前没有订阅的 TRIGGER_TOPIC。要通过测试用例，客户端设备必须使用 QoS 1 成功订阅 TRIGGER_TOPIC。重新连接后，客户端设备应该了解存在活动的持久会话；因此它应该接受由触发器主题发送的已存储消息，然后对该特定消息返回 PUBACK。

```
"tests":[
   {
      "name":"my_mqtt_persistent_session_happy_case",
      "configuration":{
         //required:
         "TRIGGER_TOPIC": "myTrigger/topic",
         // optional:
         // if Payload not provided, a string will be stored in the trigger topic to
 be sent back to the client device
         "PAYLOAD": "The message which should be received from AWS IoT Broker after
 re-connecting to a persistent session from the specified trigger topic.",

         "EXECUTION_TIMEOUT":"300" // in seconds
      },
      "test":{
         "id":"MQTT_Persistent_Session_Happy_Case",
         "version":"0.0.0"
      }
   }
]
```

“持久会话 - 会话到期”

此测试用例有助于验证断开连接的设备重新连接到过期的持久会话时的设备行为。会话过期后，我们希望设备通过显式发送新的 SUBSUBE 数据包来重新订阅之前订阅的主题。

在第一次连接期间，我们希望测试设备与 AWS 物联网代理连接，因为其`CleanSession`标志设置为 false 以启动持续会话。然后，设备应该订阅触发器主题。然后，在成功订阅并启动持续会话后， AWS IoT Core 设备顾问将断开设备的连接。断开连接后， AWS IoT Core 设备顾问允许测试设备与测试端点重新连接。此时，当测试设备发送另一个 CONNECT 数据包时， AWS IoT Core 设备顾问会发回一个 CONNACK 数据包，表示持续会话已过期。测试设备需要正确地解释此数据包，并且在持久会话终止时，它应该会再次重新订阅同一触发器主题。如果测试设备不再重新订阅其主题触发器，测试用例将失败。要通过测试，设备需要了解持久会话已经结束，然后为第二个连接中的相同触发器主题发回一个新的 SUBSCRIBE 数据包。

如果测试设备的此测试用例获得通过，则表示该设备能够按照预期方式在持久会话到期时进行重新连接。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为至少 4 分钟。测试设备需要显式订阅之前没有订阅的 TRIGGER_TOPIC。要通过测试用例，测试设备必须发送 CONNECT 数据包（CleanSession 标志设置为 false），并使用 QoS 1 成功订阅触发器主题。成功连接后， AWS IoT Core 设备顾问会断开设备的连接。断开连接后， AWS IoT Core Device Advisor 允许设备重新连接，并且设备应该重新订阅相同的连接，TRIGGER_TOPIC因为 AWS IoT Core 设备顾问本来会终止持续会话。

```
"tests":[
   {
      "name":"my_expired_persistent_session_test",
      "configuration":{
         //required:
         "TRIGGER_TOPIC": "myTrigger/topic",
         // optional:
         "EXECUTION_TIMEOUT":"300" // in seconds
      },
      "test":{
         "id":"MQTT_Expired_Persistent_Session",
         "version":"0.0.0"
      }
   }
]
```

影子

使用这些测试来验证您的被测设备是否正确使用 AWS IoT 了 Device Shadow 服务。请参阅AWS IoT Device Shadow 服务了解更多信息。如果在测试套件中配置了这些测试用例，则在启动套件运行时需要提供一个事物。

WebSocket目前不支持 MQTT 版本。

Publish

“设备连接后发布状态（快乐用例）”

验证设备在连接后能否发布其状态 AWS IoT Core

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。

```
"tests":[
   {
      "name":"my_shadow_publish_reported_state",
      "configuration": {
         // optional:
         "EXECUTION_TIMEOUT":"300", // in seconds
         "SHADOW_NAME": "SHADOW_NAME",
         "REPORTED_STATE": {
            "STATE_ATTRIBUTE": "STATE_VALUE"
         }
      },
      "test":{
         "id":"Shadow_Publish_Reported_State",
         "version":"0.0.0"
      }
   }
]
```

REPORTED_STATE 可以在设备连接后对其确切影子状态进行额外验证。默认情况下，此测试用例会验证您的设备发布状态。

如果未提供 *SHADOW_NAME*，则测试用例将默认查找发布到 Unnamed（经典）影子类型的主题前缀的消息。如果您的设备使用命名的影子类型，请提供影子名称。请参阅在设备中使用影子，了解更多信息。

更新

“设备更新报告状态为理想状态（快乐用例）”

验证设备是否读取所有收到的更新消息，并同步设备的状态以与所需的状态属性匹配。您的设备应在同步后发布其最新报告状态。如果您的设备在运行测试之前已存在影子，请确保为测试用例配置的所需状态与现有报告状态不匹配。您可以通过查看 Shadow 文档中的ClientToken字段来识别 Device Advisor 发送的 Shadow 更新消息DeviceAdvisorShadowTestCaseSetup。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 2 分钟。

```
"tests":[
   {
      "name":"my_shadow_update_reported_state",
      "configuration": {
         "DESIRED_STATE": {
            "STATE_ATTRIBUTE": "STATE_VALUE"
         },
         // optional:
         "EXECUTION_TIMEOUT":"300", // in seconds
         "SHADOW_NAME": "SHADOW_NAME"
      },
      "test":{
         "id":"Shadow_Update_Reported_State",
         "version":"0.0.0"
      }
   }
]
```

DESIRED_STATE 应具有至少一个属性和关联的值。

如果未提供 SHADOW_NAME，则测试用例将默认查找发布到 Unnamed（经典）影子类型的主题前缀的消息。如果您的设备使用命名的影子类型，请提供影子名称。请参阅在设备中使用影子，了解更多信息。

任务执行

"Device can complete a job execution"（“设备可以完成任务执行”）

此测试用例可帮助您验证您的设备是否能够使用 AWS IoT 任务接收更新，并发布成功更新的状态。有关 AWS IoT 作业的更多信息，请参阅作业。

要成功运行此测试用例，您需要为设备角色授予两个保留 AWS 主题。要订阅与消息相关的作业活动，请使用 notify 和 notify-next 主题。您的设备角色必须为以下主题授予 PUBLISH 操作：

- $aws/things/thingName/jobs/jobId/get
- $aws/things/thingName/jobs/jobId/update

建议为以下主题授予 SUBSCRIBE 和 RECEIVE 操作：

- $aws/things/thingName/jobs/get/accepted
- $aws/things/thingName/jobs/jobId/get/rejected
- $aws/things/thingName/jobs/jobId/update/accepted
- $aws/things/thingName/jobs/jobId/update/rejected

建议为以下主题授予 SUBSCRIBE 操作：

- $aws/things/thingName/jobs/notify-next

有关这些保留主题的更多信息，请参阅 AWS IoT 作业的保留主题。

WebSocket目前不支持 MQTT 版本。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 5 分钟。我们建议将超时值设置为 3 分钟。根据所提供的 Jo AWS IoT b 文档或来源，调整超时值（例如，如果作业需要很长时间才能运行，请为测试用例定义更长的超时值）。要运行测试，需要有效的 AWS IoT Job 文档或已经存在的作业 ID。 AWS IoT Job 文档可以以 JSON 文档或 S3 链接的形式提供。如果已提供作业文档，则可以选择提供作业 ID。如果提供了任务 ID，Device Advisor 将在代表您创建 AWS IoT 任务时使用该ID。如果未提供作业文档，您可以提供用于运行测试用例的同一区域中的现有 ID。在这种情况下，设备顾问将在运行测试用例时使用该 AWS IoT Job。

```
"tests": [
   {
      "name":"my_job_execution",
      "configuration": {
         // optional:
         // Test case will create a job task by using either JOB_DOCUMENT or
 JOB_DOCUMENT_SOURCE.
         // If you manage the job task on your own, leave it empty and provide the
 JOB_JOBID (self-managed job task).
         // JOB_DOCUMENT is a JSON formatted string
         "JOB_DOCUMENT": "{
```

```
            \"operation\":\"reboot\",
            \"files\" : {
               \"fileName\" : \"install.py\",
               \"url\" : \"${aws:iot:s3-presigned-url:https://s3.amazonaws.com/
bucket-name/key}\"
            }
         }",
         // JOB_DOCUMENT_SOURCE is an S3 link to the job document. It will be used
 only if JOB_DOCUMENT is not provided.
         "JOB_DOCUMENT_SOURCE": "https://s3.amazonaws.com/bucket-name/key",
         // JOB_JOBID is mandatory, only if neither document nor document source is
 provided. (Test case needs to know the self-managed job task id).
         "JOB_JOBID": "String",
         // JOB_PRESIGN_ROLE_ARN is used for the presign Url, which will replace the
 placeholder in the JOB_DOCUMENT field
         "JOB_PRESIGN_ROLE_ARN": "String",
         // Presigned Url expiration time. It must be between 60 and 3600 seconds,
 with the default value being 3600.
         "JOB_PRESIGN_EXPIRES_IN_SEC": "Long"
         "EXECUTION_TIMEOUT": "300", // in seconds
      },
      "test": {
         "id": "Job_Execution",
         "version": "0.0.0"
      }
   }
]
```

有关创建和使用任务文档的更多信息，请参阅任务文档。

权限与策略

您可以使用下列测试来确定附加到设备证书的策略是否符合下列标准最佳实践。

WebSocket目前不支持 MQTT 版本。

“设备证书附加策略不包含通配符”

验证与设备关联的权限策略是否遵循最佳实践，并且未向设备授予除所需权限以外的其它权限。

API 测试用例定义：

> **Note**
>
> EXECUTION_TIMEOUT 的默认值为 1 分钟。我们建议将超时设置为至少 30 秒。

```
"tests":[
   {
        "name":"my_security_device_policies",
        "configuration": {
            // optional:
            "EXECUTION_TIMEOUT":"60"    // in seconds
        },
        "test": {
            "id": "Security_Device_Policies",
            "version": "0.0.0"
        }
    }
]
```

长时间测试

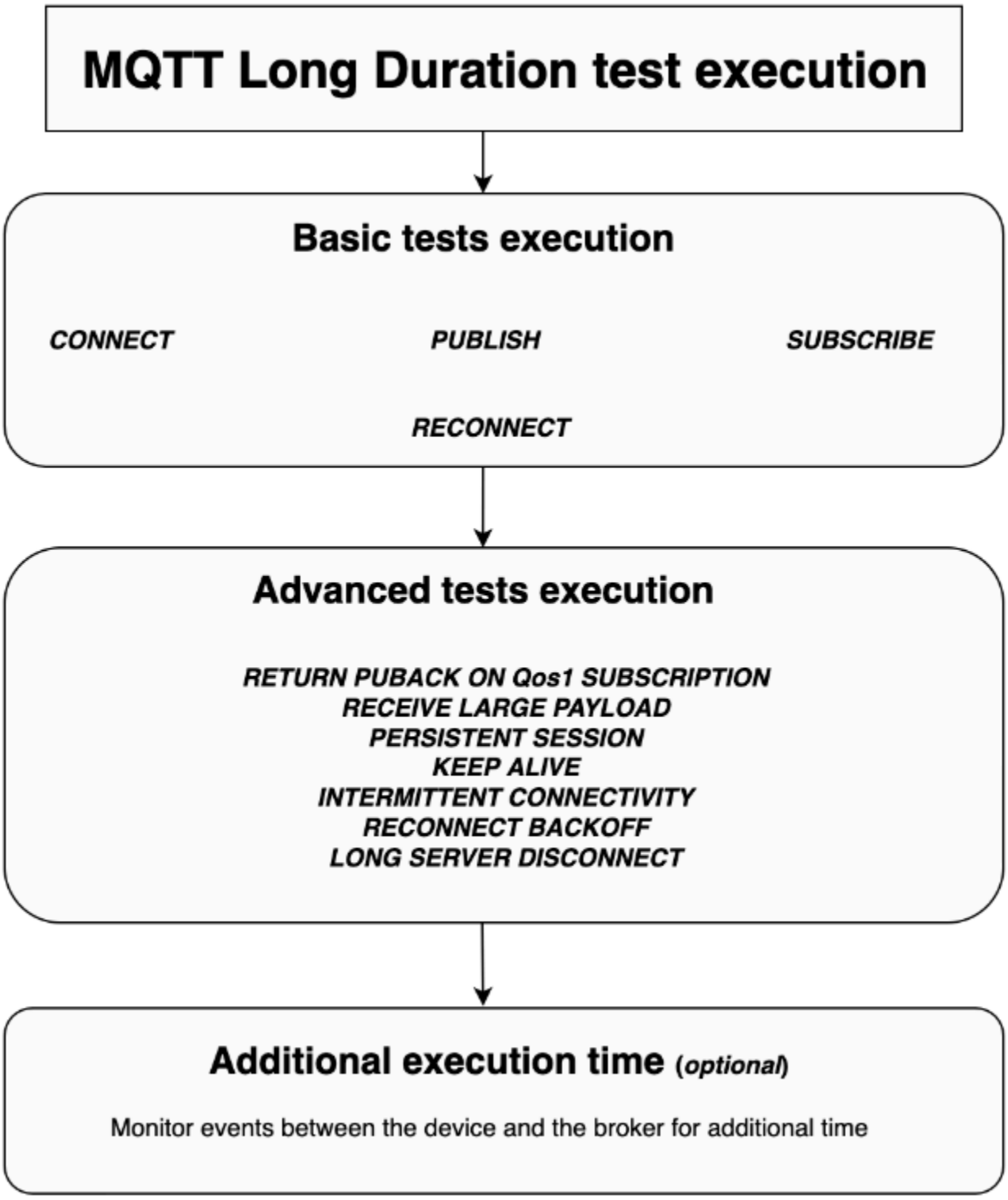
长时间测试是一种新的测试套件，可以在设备长时间运行时监控设备的行为。与运行专注于设备特定行为的单独测试相比，长时间测试可在设备的使用寿命内检查设备在各种真实场景中的行为。Device Advisor 会以尽可能最有效的顺序编排测试。测试生成结果和日志，包括摘要日志，其中包含有关设备在测试期间的性能的有用指标。

MQTT 长时间测试用例

在 MQTT 长时间测试用例中，设备的行为最初是在诸如 MQTT 连接、订阅、发布和重新连接之类的快乐用例场景中观察的。然后，在多种复杂的故障场景中观察该设备，例如 MQTT 重新连接回退、长时间服务器断开连接和间歇性连接。

MQTT 长时间测试用例执行流程

MQTT 长时间测试用例的执行分为三个阶段：



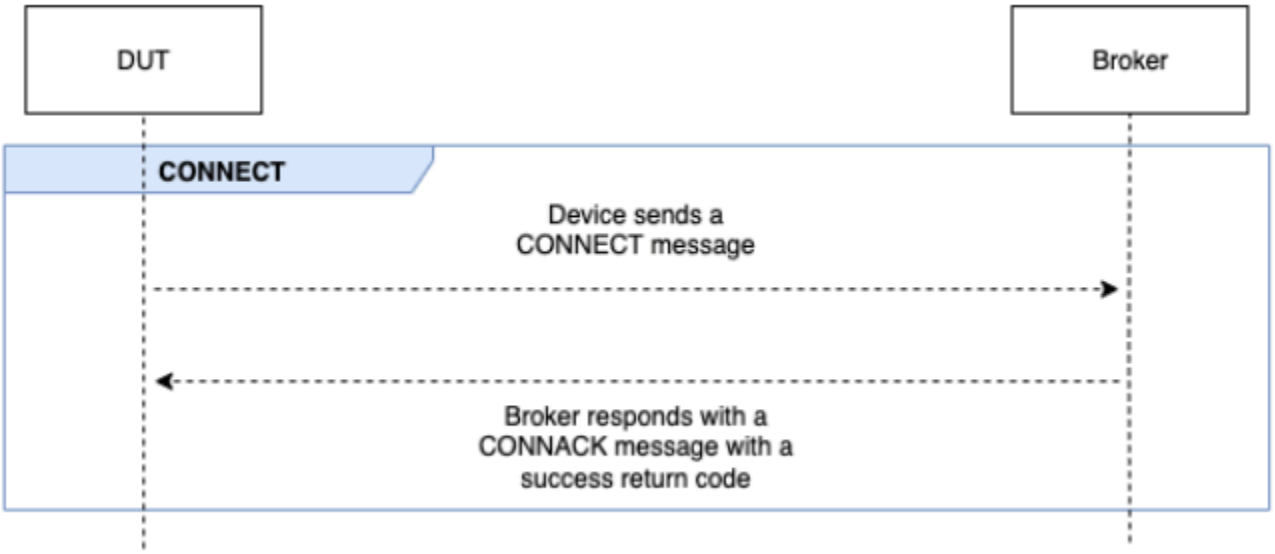
执行基本测试

在此阶段，测试用例并行运行多项简单测试。该测试验证设备是否在配置中选择了这些操作。

根据所选操作，基本测试集可以包括以下内容：

连接

此场景验证设备是否能够成功与代理建立连接。

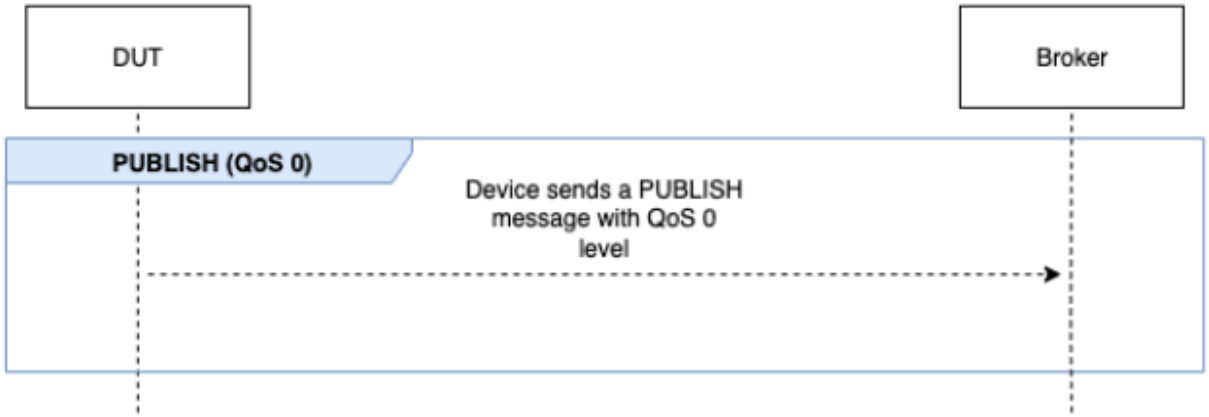


发布

此场景验证设备是否能够成功向代理发布。

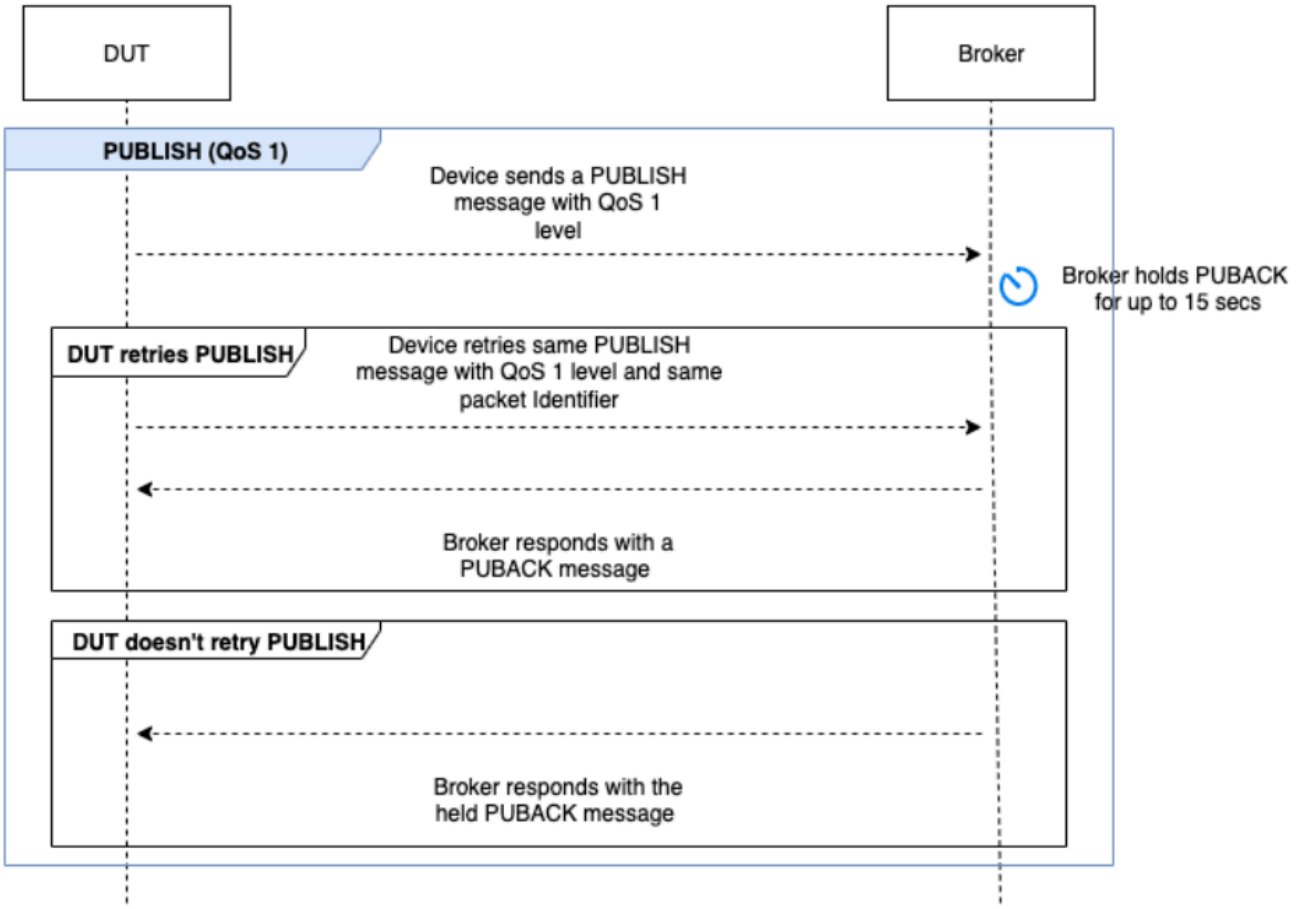
QoS 0

此测试用例验证设备在使用 QoS 0 发布期间是否能够成功向代理发送 PUBLISH 消息。测试不会等待设备收到 PUBACK 消息。



QoS 1

在此测试用例中，设备应使用 QoS 1 向代理发送两条 PUBLISH 消息。在收到第一条 PUBLISH 消息后，代理最多等待 15 秒钟就会做出响应。设备必须在 15 秒的期限内使用相同的数据包标识符重试原始 PUBLISH 消息。如果是这样，代理会回复一条 PUBACK 消息，测试将进行验证。如果设备不重试 PUBLISH，会将原始 PUBACK 发送到设备，测试将被标记为 Pass with warnings（通过但带有警告），并会显示系统消息。在测试执行期间，如果设备丢失连接并重新连接，则测试场景将顺利重置，并且设备必须重新执行测试场景步骤。

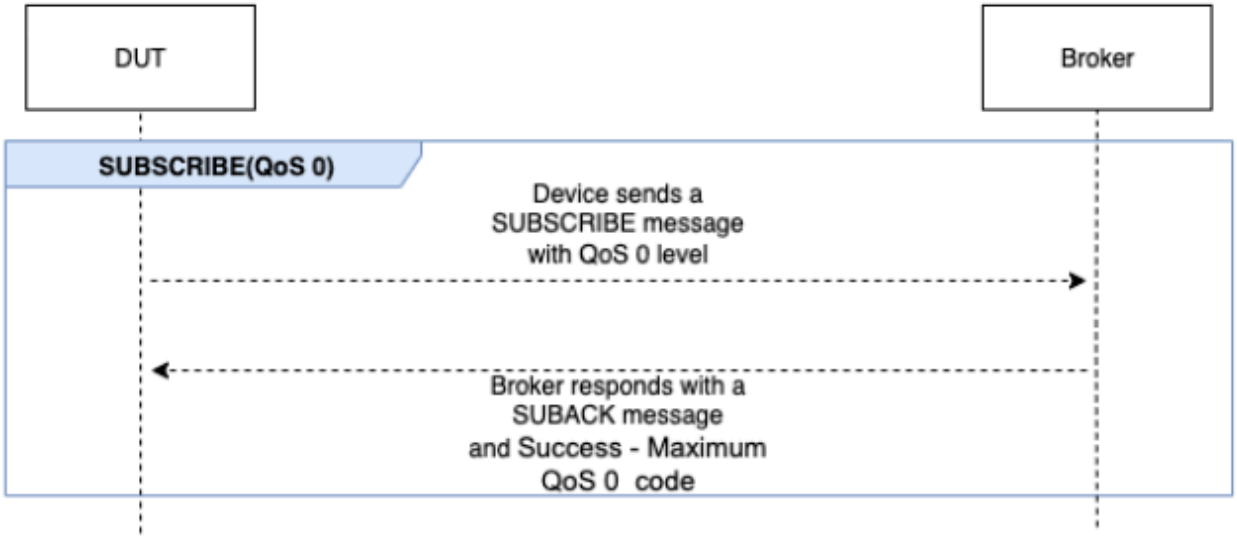


订阅

此场景验证设备是否能够成功向代理订阅。

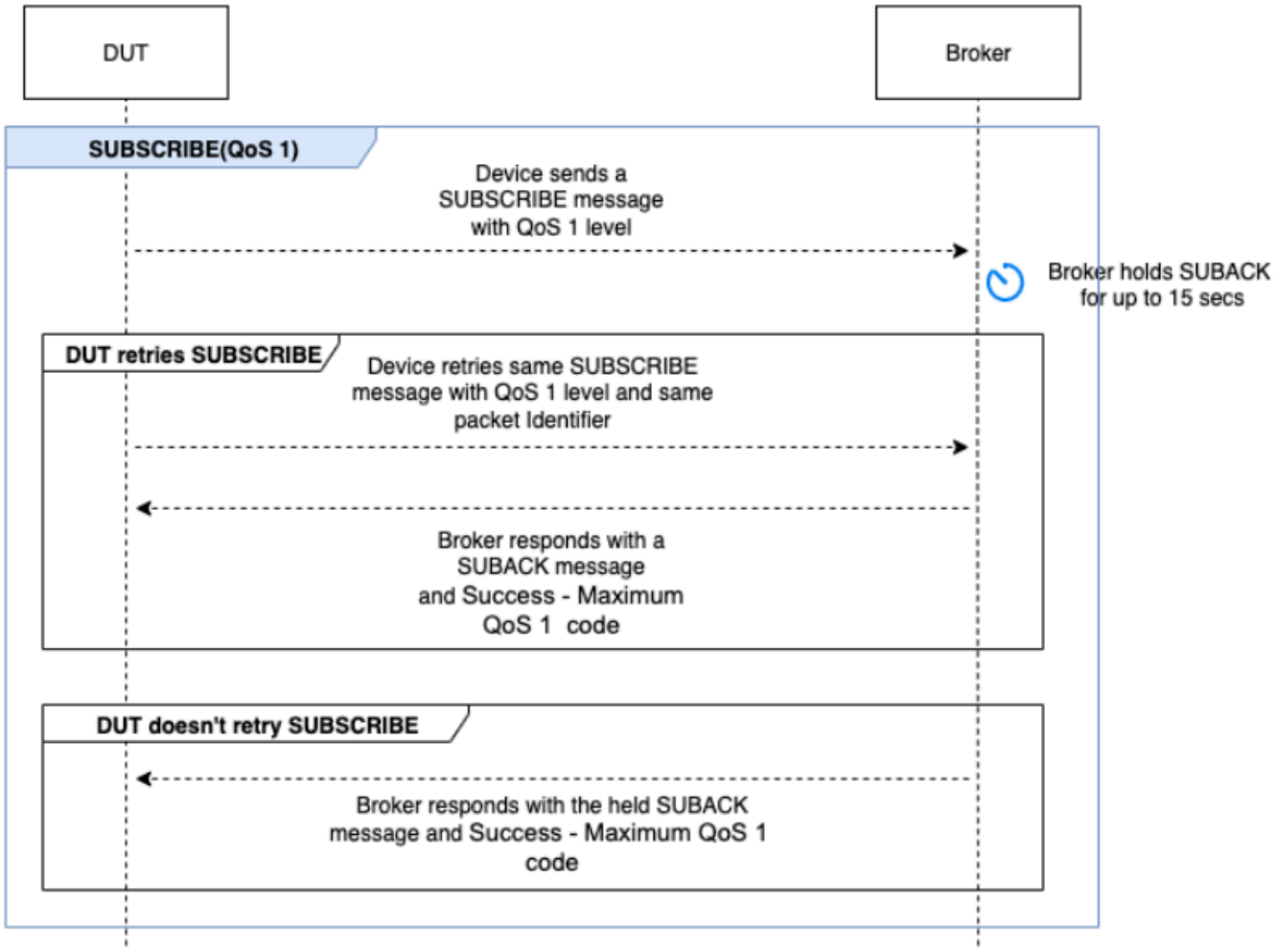
QoS 0

此测试用例验证设备在使用 QoS 0 订阅期间是否能够成功向代理发送 SUBSCRIBE 消息。测试不会等待设备收到 SUBACK 消息。



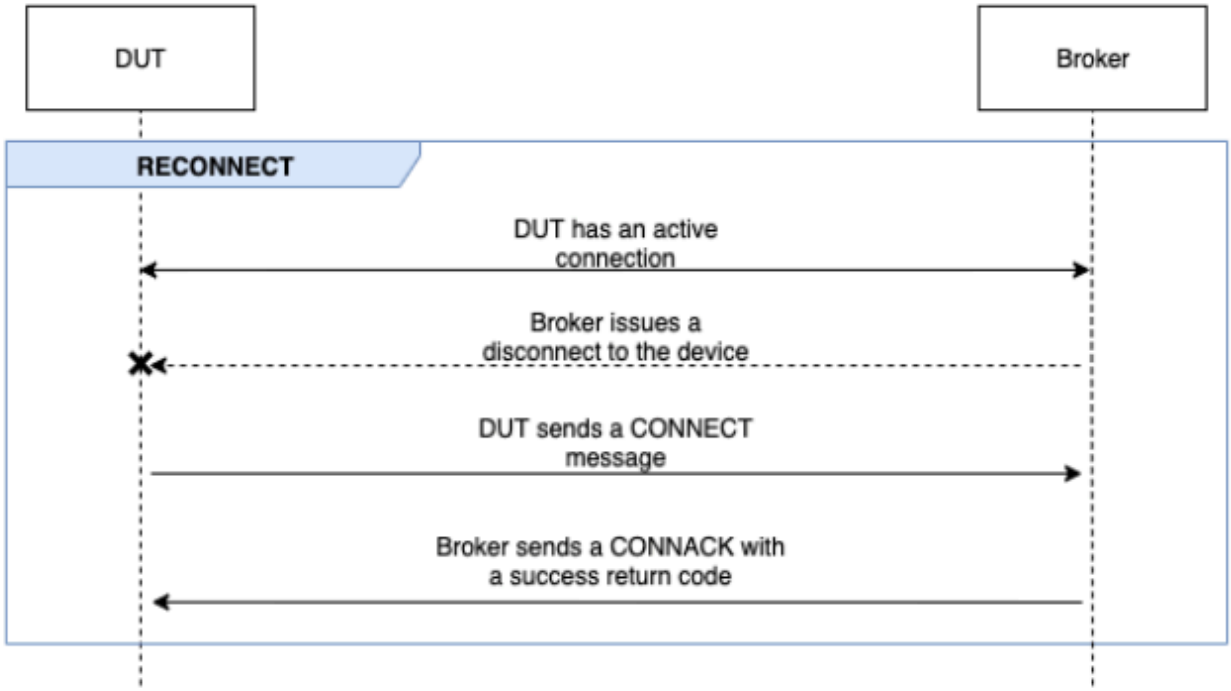
QoS 1

在此测试用例中，设备应使用 QoS 1 向代理发送两条 SUBSCRIBE 消息。在收到第一条 SUBSCRIBE 消息后，代理最多等待 15 秒钟就会做出响应。设备必须在 15 秒的期限内使用相同的数据包标识符重试原始 SUBSCRIBE 消息。如果是这样，代理会回复一条 SUBACK 消息，测试将进行验证。如果设备不重试 SUBSCRIBE，会将原始 SUBACK 发送到设备，测试将被标记为 Pass with warnings（通过但带有警告），并会显示系统消息。在测试执行期间，如果设备丢失连接并重新连接，则测试场景将顺利重置，并且设备必须重新执行测试场景步骤。



重新连接

此场景验证在设备从成功连接断开连接后，是否能够成功与代理重新连接。如果在之前的测试套件期间多次连接设备，Device Advisor 不会断开设备此连接。相反，它会将测试标记为 Pass（通过）。



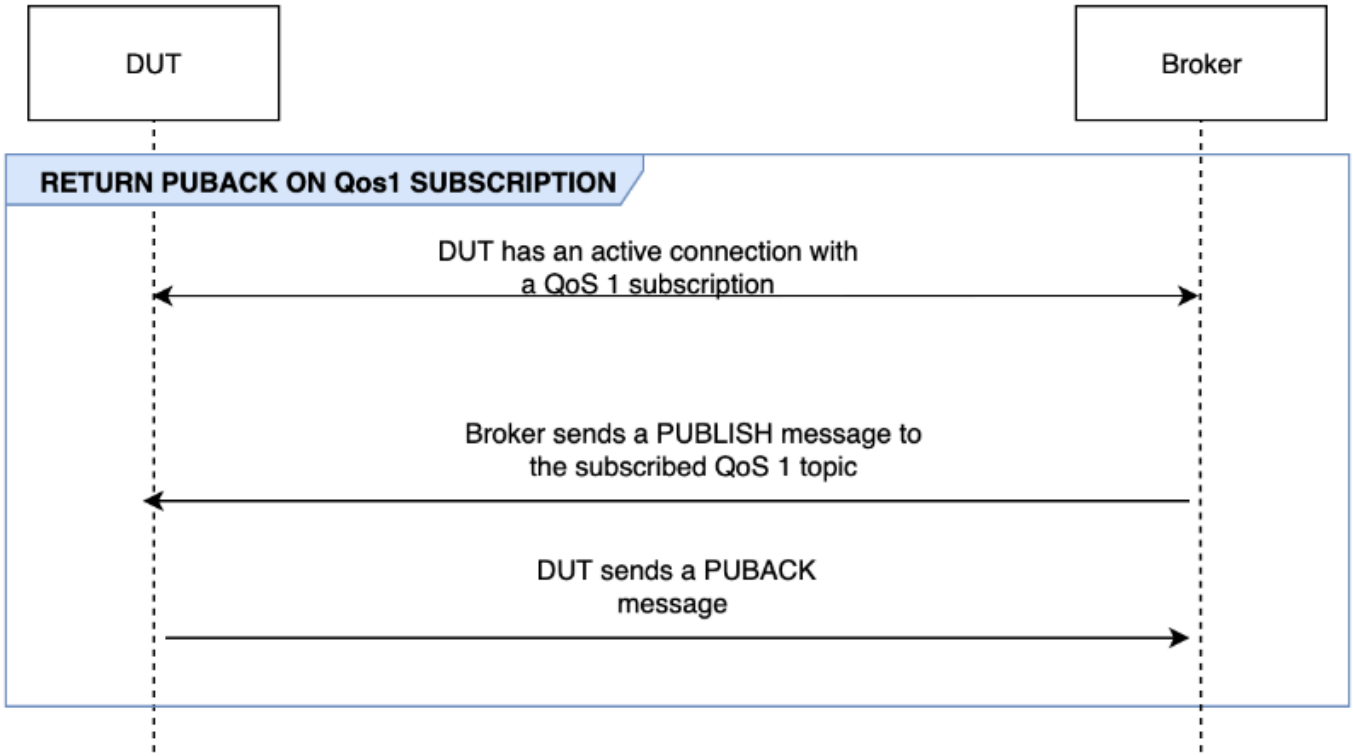
执行高级测试

在此阶段，测试用例连续运行更复杂的测试，以验证设备是否遵循最佳实践。这些高级测试可供选择，如果不需要，可以选择退出。根据场景的要求，每个高级测试都有自己的超时值。

订阅 QoS 1 时返回 PUBACK

> **Note**
>
> 仅当您的设备能够执行 QoS 1 订阅时才选择此场景。

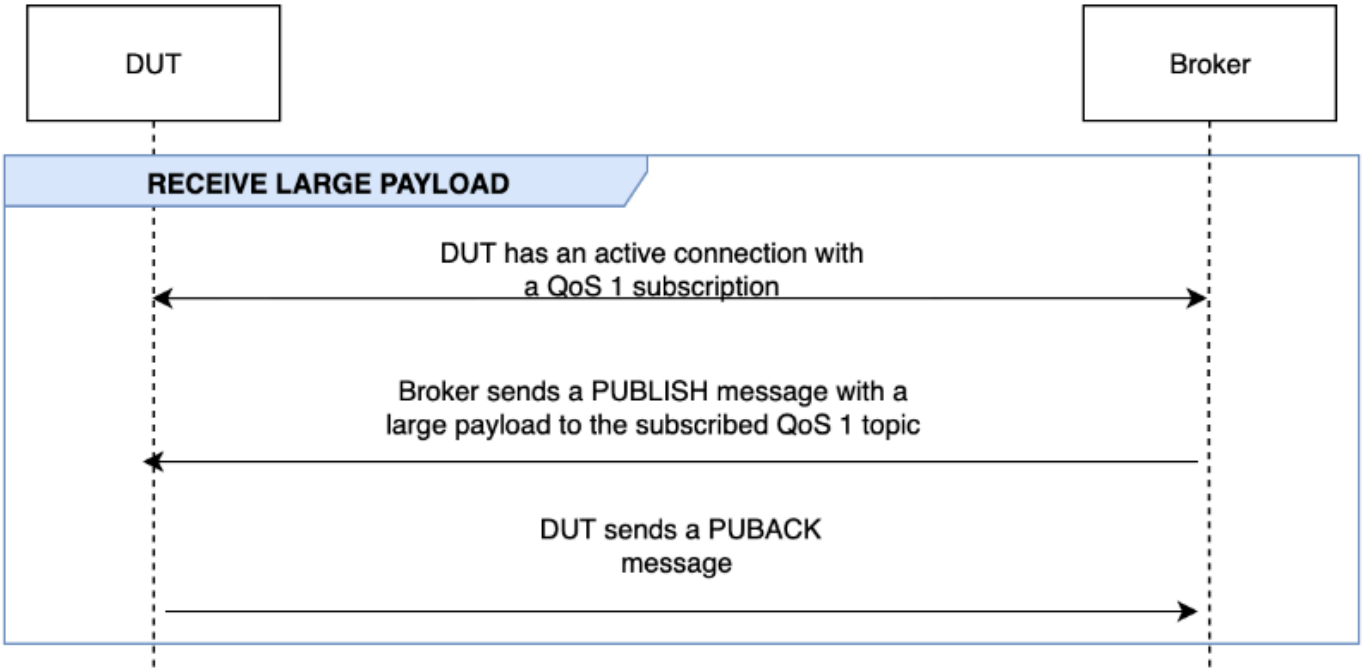
此场景验证设备订阅主题并收到来自代理的 PUBLISH 消息后是否返回 PUBACK 消息。



接收较大有效负载

> **Note**
>
> 仅当您的设备能够执行 QoS 1 订阅时才选择此场景。

此场景验证设备在从代理接收到 QoS 1 主题的具有较大有效负载的 PUBLISH 消息后，是否以 PUBACK 消息进行响应。可以使用 LONG_PAYLOAD_FORMAT 选项配置预期有效负载的格式。



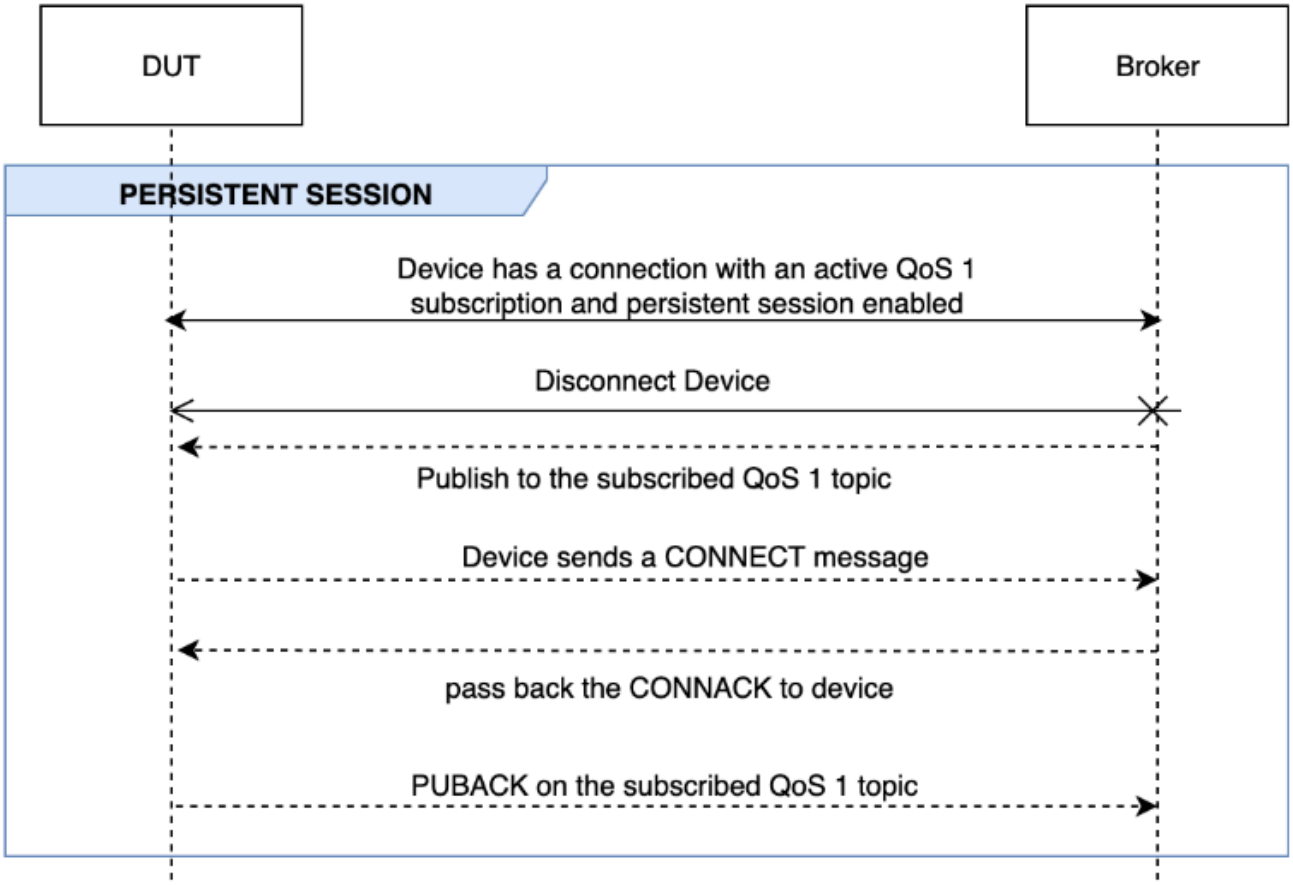
持久性会话

> **Note**
>
> 仅当您的设备能够执行 QoS 1 订阅，并且能够保留持久性会话时才选择此场景。

此场景验证设备在保留持续性会话方面的行为。当满足以下条件时，此测试将进行验证：

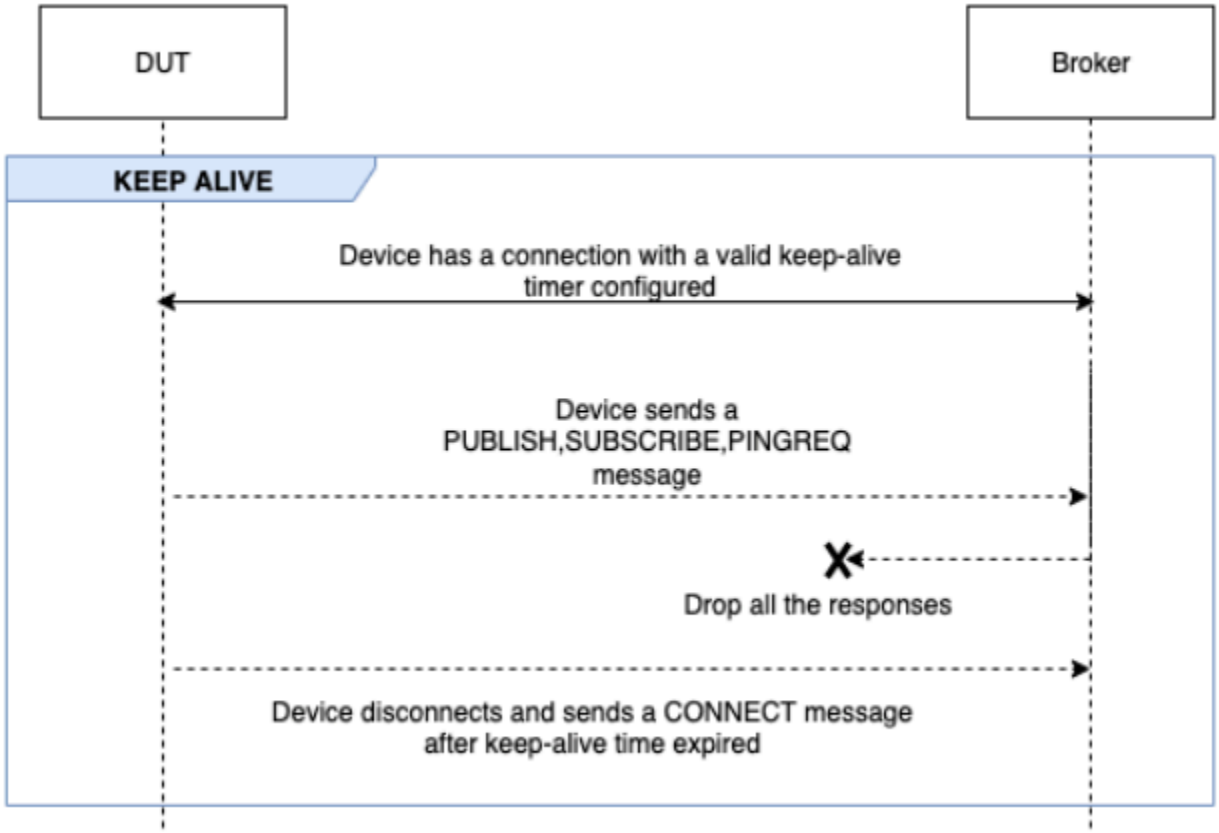
- 设备在具有有效 QoS 1 订阅并启用持久性会话的情况下连接到代理。
- 设备在会话期间成功断开与代理的连接。
- 设备重新连接到代理并恢复对其触发器主题的订阅，而没有明确重新订阅这些主题。
- 设备成功接收代理存储的有关其订阅主题的消息，并按预期运行。

有关 AWS IoT 持久会话的更多信息，请参阅使用 MQTT 持久会话。



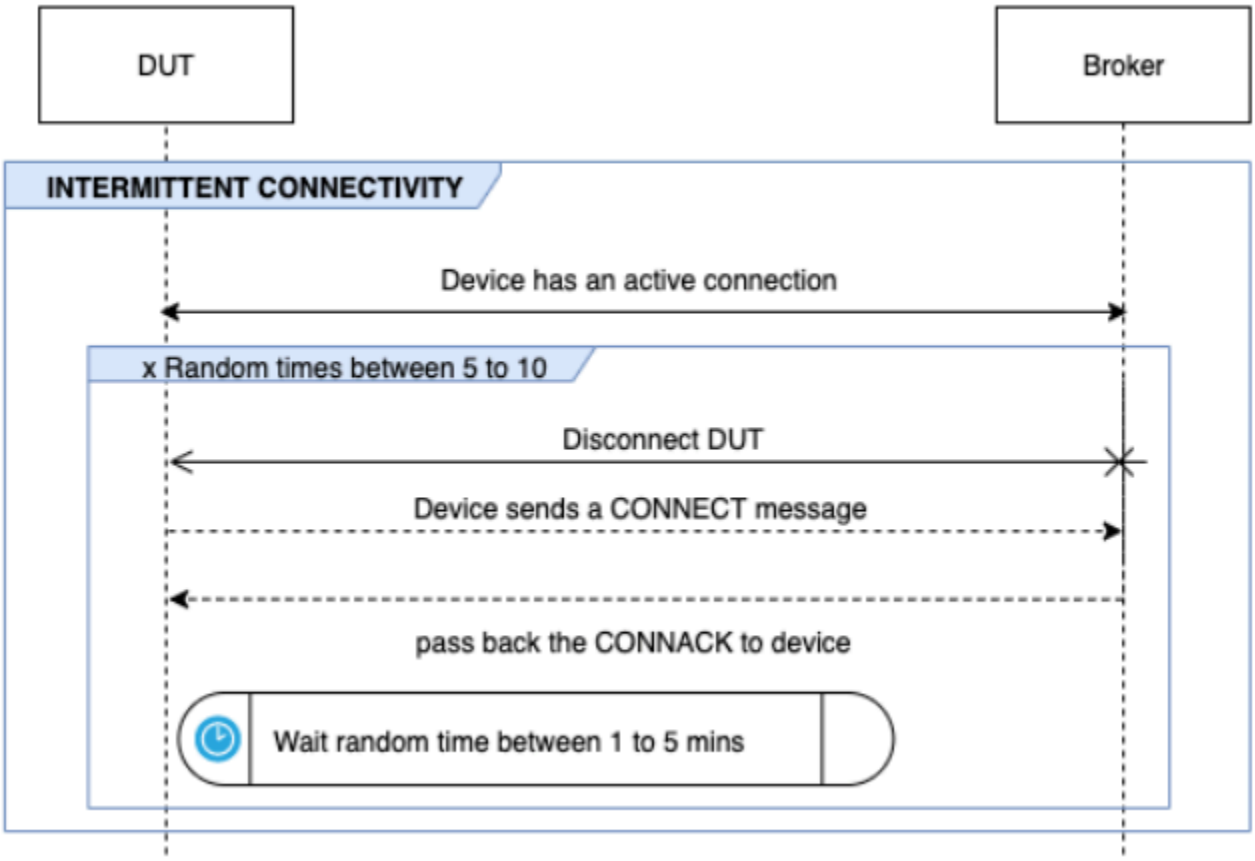
保持活动

此场景验证设备在未收到来自代理的 ping 响应后是否成功断开连接。连接必须配置有效的保持活动计时器。作为此测试的一部分，代理会阻止针对 PUBLISH、SUBSCRIBE 和 PINGREQ 消息发送的所有响应。它还验证正在测试的设备是否断开了 MQTT 连接。



间歇性连接

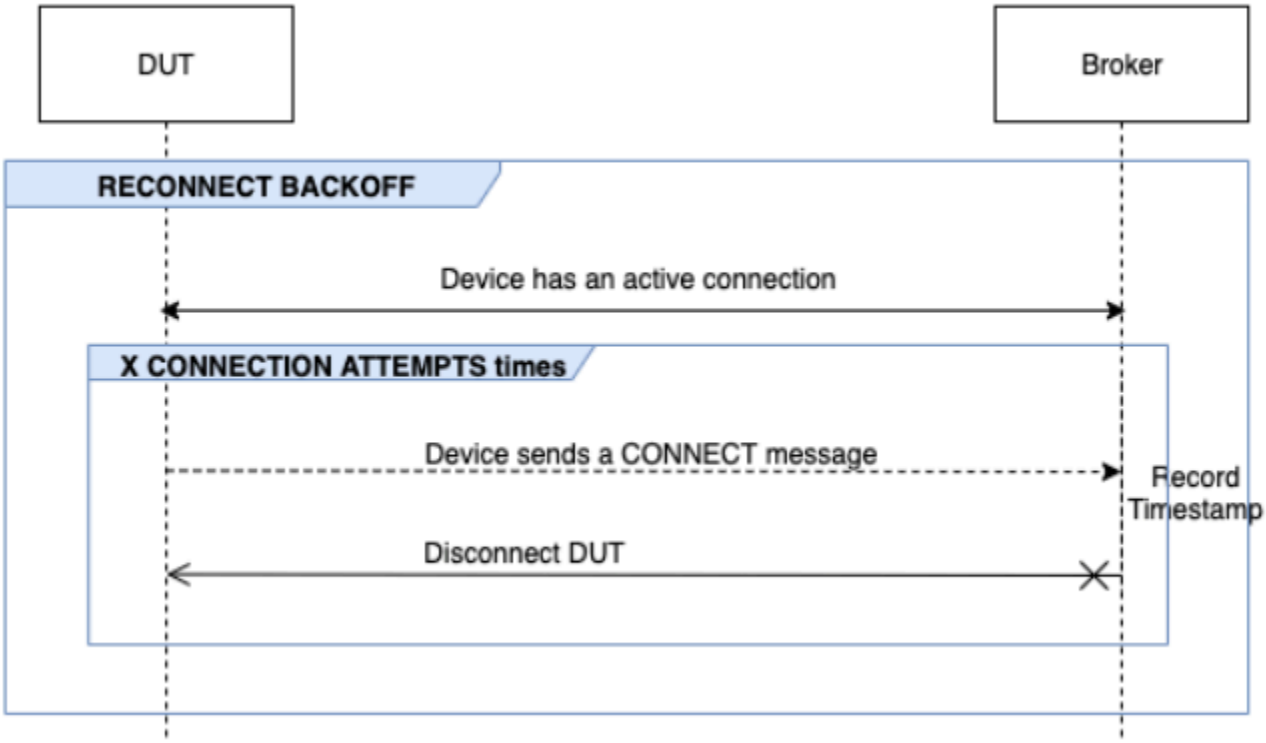
此场景验证在代理以随机间隔断开设备连接一段随机时间后，设备是否可以连接回代理。



重新连接回退

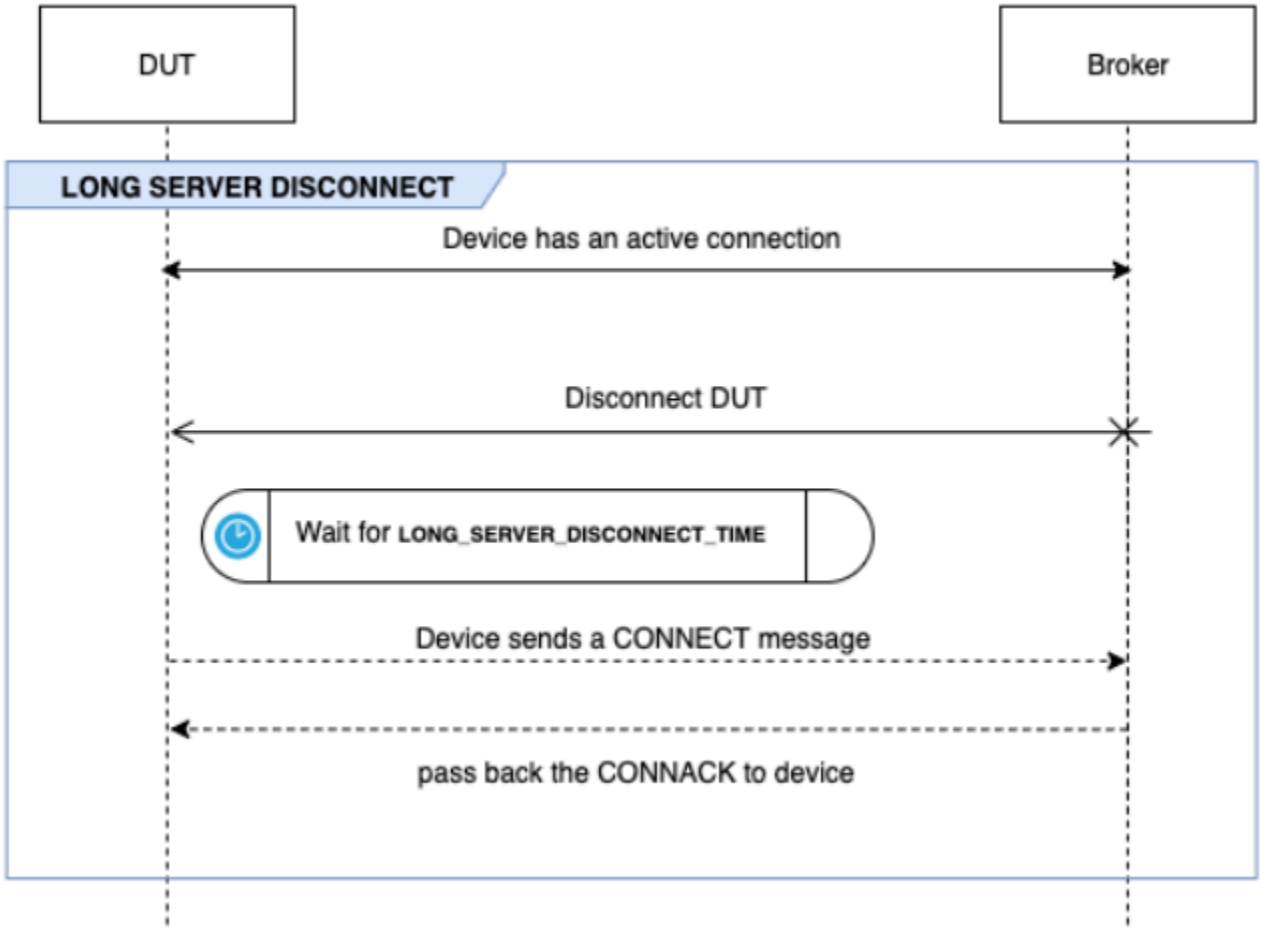
此场景验证代理多次与设备断开连接时，设备是否实现了回退机制。Device Advisor 将回退类型报告为指数回退、抖动回退、线性回退或恒定回退。回退尝试次数可使用 BACKOFF_CONNECTION_ATTEMPTS 选项进行配置。默认值是 5。此值可配置为 5 到 10。

要通过此测试，我们建议在所测试设备上实施指数回退和抖动机制。



服务器长时间断开连接

此场景验证代理在较长一段时间（最多 120 分钟）断开设备连接后，设备是否可以成功重新连接。可以使用 LONG_SERVER_DISCONNECT_TIME 选项配置服务器断开连接的时间。默认值为 120 分钟。此值可配置为 30 到 120 分钟。



额外执行时间

额外执行时间是测试在完成所有上述测试之后，以及结束测试用例之前等待的时间。客户利用这段额外时间来监控和记录设备与代理之间的所有通信。可以使用 ADDITIONAL_EXECUTION_TIME 选项配置额外执行时间。默认情况下，此选项设置为 0 分钟，也可以设置为 0 到 120 分钟。

MQTT 长时间测试配置选项

为 MQTT 长时间测试提供的所有配置选项都是可选的。以下选项可用：

OPERATIONS（操作）

设备执行的操作列表，例如 CONNECT、PUBLISH 和 SUBSCRIBE。测试用例根据指定的操作运行场景。未指定的操作假定为有效。

```
{
"OPERATIONS": ["PUBLISH", "SUBSCRIBE"]
```

```
//by default the test assumes device can CONNECT
}
```

SCENARIOS（场景）

测试用例根据所选操作运行场景来验证设备的行为。有两种类型的场景：

- 基本场景是一些简单的测试，用于验证设备是否可以执行上面作为配置的一部分选择的操作。这些场景是根据配置中指定的操作预先选择的。配置中不需要更多输入。
- 高级场景是对设备执行的更为复杂的场景，用于验证设备在满足现实条件时是否遵循了最佳实践。这些场景是可选的，可以作为场景数组传递给测试套件的配置输入。

```
{
    "SCENARIOS": [      // list of advanced scenarios
                "PUBACK_QOS_1",
                "RECEIVE_LARGE_PAYLOAD",
                "PERSISTENT_SESSION",
                "KEEP_ALIVE",
                "INTERMITTENT_CONNECTIVITY",
                "RECONNECT_BACK_OFF",
                "LONG_SERVER_DISCONNECT"
    ]
}
```

BASIC_TESTS_EXECUTION_TIME_OUT：

测试用例等待所有基本测试完成的最长时间。默认值为 60 分钟。此值可配置为 30 到 120 分钟。

LONG_SERVER_DISCONNECT_TIME：

在“服务器长时间断开连接”测试期间，测试用例断开连接和重新连接设备所用的时间。默认值为 60 分钟。此值可配置为 30 到 120 分钟。

ADDITIONAL_EXECUTION_TIME：

配置此选项可在所有测试完成后提供一个时间范围，用于监控设备和代理之间的事件。默认值为 5 分钟。此值可配置为 0 到 120 分钟。

BACKOFF_CONNECTION_ATTEMPTS：

此选项可配置测试用例断开连接设备的次数。此选项由重新连接回退测试使用。默认值为 5 次。此值可配置为 5 到 10。

LONG_PAYLOAD_FORMAT：

当测试用例发布到设备订阅的 QoS 1 主题时，设备期望的消息有效负载格式。

API 测试用例定义：

```
{
"tests":[
   {
      "name":"my_mqtt_long_duration_test",
      "configuration": {
         // optional
         "OPERATIONS": ["PUBLISH", "SUBSCRIBE"],
         "SCENARIOS": [
            "LONG_SERVER_DISCONNECT",
            "RECONNECT_BACK_OFF",
            "KEEP_ALIVE",
            "RECEIVE_LARGE_PAYLOAD",
            "INTERMITTENT_CONNECTIVITY",
            "PERSISTENT_SESSION",
         ],
         "BASIC_TESTS_EXECUTION_TIMEOUT": 60, // in minutes (60 minutes by default)
         "LONG_SERVER_DISCONNECT_TIME": 60,   // in minutes (120 minutes by default)
         "ADDITIONAL_EXECUTION_TIME": 60,     // in minutes (0 minutes by default)
         "BACKOFF_CONNECTION_ATTEMPTS": "5",
         "LONG_PAYLOAD_FORMAT":"{"message":"${payload}"}"
      },
      "test":{
         "id":"MQTT_Long_Duration",
         "version":"0.0.0"
      }
   }
 ]
}
```

MQTT 长时间测试用例摘要日志

MQTT 长时间测试用例的运行时间比常规测试用例长。提供了单独的摘要日志，其中列出了运行期间的重要事件，例如设备连接、发布和订阅。详细信息包括测试的内容、未测试的内容和失败的内容。测试会在日志的末尾包含测试用例运行期间发生的所有事件的摘要。摘要包括：

- 设备上配置的“保持活动”计时器。

- 设备上配置的持久性会话标志。
- 测试运行期间的设备连接数。
- 设备重新连接回退类型（如果已通过重新连接回退测试验证）。
- 在测试用例运行期间，设备发布到的主题。
- 在测试用例运行期间，设备订阅的主题。

AWS IoT Device Management 软件包 Package 目录

使用 AWS IoT Device Management Software Package Catalog，您可以维护软件包及其版本的清单。您可以将包版本与单个事物和 AWS IoT 动态事物组相关联，并通过内部流程或AWS IoT 作业进行部署。

软件包中包含一个或多个软件包版本，这些版本是可以作为单个单元部署的文件集合。软件包版本可以包含固件、操作系统更新、设备应用程序、配置和安全补丁。随着软件发展，您可以创建新的软件包版本并将其部署到您的实例集中。

AWS IoT 软件包中心位于其中 AWS IoT Core。您可以使用该中心来集中注册和维护您的软件包清单和元数据，从而创建软件包及其版本的目录。您可以选择根据设备上部署的软件包和软件包版本对设备进行分组。此特征提供了此类机会：将设备端软件包清单保留为命名影子，根据版本关联设备和对设备分组，以及使用实例集指标可视化软件包版本在实例集中的分布。

如果您已建立内部软件部署系统，则可以继续使用该过程来部署软件包版本。如果您尚未建立部署过程，或者如果您愿意，我们建议您通过 AWS IoT 任务来使用软件包目录中的特征。有关更多信息，请参阅准备 AWS IoT 作业。

本章包含以下部分：

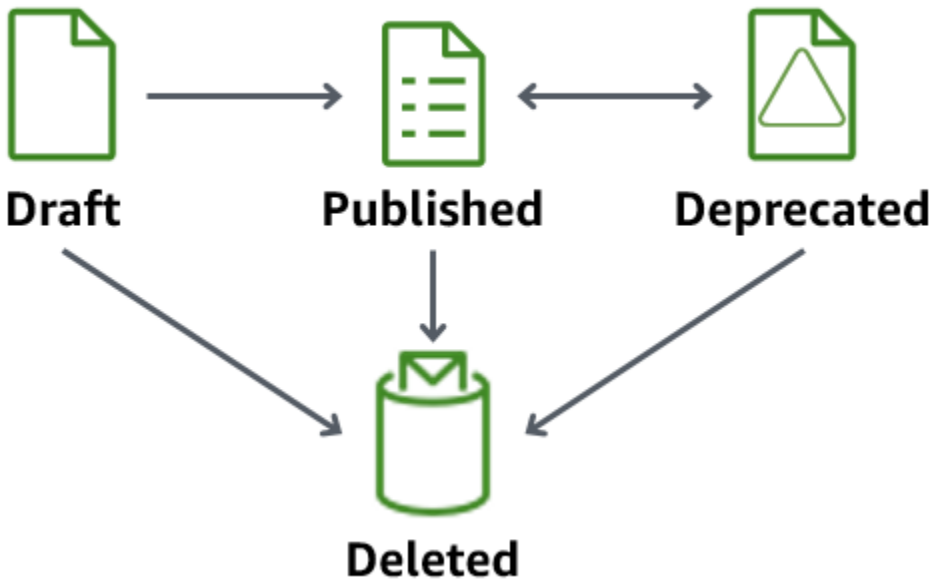
- 准备使用软件包目录
- 准备安全性
- 准备实例集索引
- 准备 AWS IoT 工作
- 软件包目录入门

准备使用软件包目录

以下部分概述了 AWS IoT Device Management 软件包的版本生命周期以及使用 Software Package Catalog 的信息。

软件包版本生命周期

软件包版本可以通过以下生命周期状态演变：`draft`、`published` 和 `deprecated`。也可以是 `deleted`。



- 草稿

  当你创建包版本时，它处于draft状态。此状态表示软件包正在准备中或不完整。

  当软件包版本处于这种状态时，您无法对其进行部署。您可以编辑软件包版本的描述、属性和标签。

  您可以使用控制台，或者deleted通过发出版本published或版本 API 操作将处于或draft处于状态的UpdatePackage软件包DeletePackage版本过渡。

- 已发布

  当您的软件包版本准备好部署时，请将软件包版本过渡到published状态。在此状态下，您可以选择通过在控制台中编辑软件包或通过 UpdatePackageAPI 操作将软件包版本标识为默认版本。在此状态下，您只能编辑描述和标签。

  您可以使用控制台或发出版本deprecated或版本 API 操作将deleted处于或published处于状态的UpdatePackage软件包DeletePackage版本过渡。

- 已弃用

  如果新的软件包版本可用，则可以将较早的软件包版本转换为 deprecated。您仍然可以使用已弃用的软件包版本部署作业。您也可以将已弃用的软件包版本命名为默认版本，并仅编辑描述和标签。

  可以考虑将软件包版本过渡到版本过deprecated时的时候，但由于运行时依赖性，现场仍有设备使用旧版本，或者需要对其进行维护。

您可以使用控制台，或者发布版本或版本 API 操作，将处于published或deprecated 处于状态deleted的软件包UpdatePackage版本过渡到或处于该状态的软件包DeletePackage版本。

- Deleted

  当您不再打算使用某个软件包版本时，您可以使用控制台或发出DeletePackage版本 API 操作将其删除。

  > **Note**
  >
  > 如果您在有待处理的任务引用软件包版本的情况下删除此版本，则当该任务成功完成并尝试更新预留命名影子时，您将收到错误消息。
  > 如果您要删除的软件包版本已指定为默认软件包版本，则必须先更新软件包以将另一个版本指定为默认版本，或者将该字段保留为未指定。您可以使用控制台或UpdatePackage版本 API 操作来执行此操作。（要将任何已命名的软件包版本作为默认值删除，请在发出UpdatePackageAPI 操作时将 un DefaultVersion set 参数设置为 true）。

  如果您通过控制台删除软件包，则会删除与该软件包关联的所有软件包版本，除非其中一个版本指定为默认版本。

软件包版本命名约定

在命名软件包版本时，重要的是要规划和应用合乎逻辑的命名策略，这样您和他人就可以轻松地识别最新的软件包版本和版本进展。创建软件包版本时必须提供版本名称，但策略和格式在很大程度上取决于您的业务案例。

作为最佳实践，我们建议使用语义版本控制格式。SemVer例如 1.2.3，其中 1 是功能发生不兼容更改的主要版本，2 是功能发生兼容更改的主要版本，而 3 是补丁版本（适用于错误修复）。有关更多信息，请参阅语义版本控制 2.0.0。有关软件包版本名称要求的更多信息，请参阅 AWS IoT API 参考指南中的 versionN ame。

默认版本

将版本设置为默认版本是可选的。您可以添加或删除默认软件包版本。您也可以部署未指定为默认版本的软件包版本。

创建软件包版本时，它处于 `draft` 状态，在将软件包版本转换为已发布状态之前，无法将其指定为默认版本。软件包目录不会自动选择一个版本作为默认版本，也不会将更高的软件包版本更新为默认版本。您必须通过控制台或发出 Version API 操作来故意命名您选择的软件包UpdatePackage版本。

版本属性

版本属性及其值拥有有关软件包版本的重要信息。我们建议您为软件包或软件包版本定义通用属性。例如，您可以为平台、架构、操作系统、发布日期、作者或 Amazon S3 URL 创建名称/值对。

使用作业文档创建 AWS IoT 作业时，也可以选择使用引用属性值的替代变量 (`$parameter`)。有关更多信息，请参阅准备 AWS IoT 作业。

软件包版本中使用的版本属性不会自动添加到预留命名影子中，也无法直接通过队列索引进行索引或查询。要通过队列索引对软件包版本属性进行索引或查询，可以在预留命名影子中填充版本属性。

我们建议预留命名影子中的版本属性参数捕获报告设备的属性，例如操作系统和安装时间。也可以通过队列索引对它们进行索引和查询。

版本属性无需遵循特定的命名惯例。您可以创建名称/值对以满足您的业务需求。软件包版本上所有属性的总大小限制为 3KB。有关更多信息，请参阅软件包目录软件包和软件包版本限制。

启用 AWS IoT 舰队索引

必须为软件包目录激活实例集索引，才能创建或更新软件包和软件包版本。队列索引提供的支持使 AWS IoT 事物能够通过按版本筛选的动态事物组进行分组。例如，实例集索引可以识别已安装或尚未安装特定软件包版本、未安装任何软件包版本或匹配特定名称/值对的事物。最后，实例集索引提供了标准指标和自定义指标，您可以使用这些指标来深入了解实例集的状态。有关更多信息，请参阅 准备实例集索引。

> **Note**
>
> 为软件包目录启用实例集索引会产生标准服务成本。有关更多信息，请参阅 AWS IoT Device Management定价。

预留命名影子

预留命名影子 `$package` 反映了设备已安装的软件包和软件包版本的状态。实例集索引使用预留命名影子作为数据来源来构建标准指标和自定义指标，以便您可以查询实例集的状态。有关更多信息，请参阅准备实例集索引。

预留命名影子与命名影子类似，唯一的不同是前者的名称是预定义的，您无法对其进行更改。此外，预留命名影子不会使用元数据进行更新，而只使用 `version` 和 `attributes` 关键字。

包含其它关键字（例如 `description`）的更新请求将在 `rejected` 主题下收到错误响应。有关更多信息，请参阅设备影子错误消息。

它可以在您通过控制台创建 AWS IoT 事物、 AWS IoT 任务成功完成并更新影子以及发出 UpdateThingShadowAPI 操作时创建。有关更多信息，请参阅 AWS IoT Core 开发者指南中的 UpdateThingShadow。

> **Note**
>
> 对预留命名影子编制索引不计入实例集索引可以编制索引的命名影子数量。有关更多信息，请参阅 AWS IoT Device Management 实例集索引限制和限额。此外，如果您选择在任务成功完成时让 AWS IoT 任务更新预留的名为 shadow，则 API 调用将计入您的 Device Shadow 和注册表操作，并且可能会产生费用。有关更多信息，请参阅AWS IoT Device Management 任务限制和配额以及 IndexingFilterAPI 数据类型。

$package 影子的结构

预留命名影子包含以下内容：

```
{
    "state": {
        "reported": {
            "<packageName>": {
                "version": "",
                "attributes": {
                }
            }
        }
    },
    "version" : 1
    "timestamp" : 1672531201
}
```

影子属性将使用以下信息进行更新：

- `<packageName>`：已安装软件包的名称，该名称使用 packageName 参数进行更新。

- `version`：已安装软件包版本的名称，该版本使用 versionName 参数进行更新。
- `attributes`：由设备存储并由实例集索引编制索引的可选元数据。这允许客户根据存储的数据查询其索引。
- `version`：影子的版本号。每次更新影子时，版本号都会自动递增，从 1 开始。
- `timestamp`：表示上次更新影子的时间，并以 Unix 时间记录。

有关命名影子的格式和行为的更多信息，请参阅 AWS IoT 设备影子服务 消息顺序。

删除软件包及其软件包版本

在删除软件包之前，请执行以下操作：

- 确认软件包及其版本未处于活动部署状态。
- 请先删除所有关联的版本。如果其中一个版本指定为默认版本，则必须从软件包中删除指定的默认版本。由于指定默认版本是可选的，因此删除默认版本不会发生冲突。要从软件包中删除默认版本，请通过控制台编辑软件包或使用 UpdatePackageVersionAPI 操作。

只要没有已指定的默认软件包版本，您就可以使用控制台删除软件包，其所有软件包版本也将被删除。如果您使用 API 调用删除软件包，则必须先删除软件包版本，然后再删除软件包。

准备安全性

本节讨论 Software Package Catal AWS IoT Device Management og 的主要安全要求。

基于资源的身份验证

在更新实例集中的软件时，软件包目录使用基于资源的授权来提高安全性。这意味着您必须创建一个 AWS Identity and Access Management (IAM) 策略，该策略授予对软件包和软件包版本执行`create`、`read`、`update`、`delete`和`list`操作的权限，并在该`Resources`部分中引用要部署的特定软件包和软件包版本。您也需要这些权限，以便更新预留命名影子。您可以通过为每个实体包括 Amazon 资源名称（ARN）来引用软件包和软件包版本。

> **Note**
>
> 如果您打算通过策略授予软件包版本 API 调用（例如CreatePackage版本、UpdatePackage版本、DeletePackage版本）的权限，则需要在策略中同时包含软件包和软件包版本 ARN。

如果您打算通过策略授予软件包 API 调用（例如CreatePackageUpdatePackage、和DeletePackage）的权限，则必须在策略中仅包含软件包 ARN。

按如下方式构造软件包 ARN 和软件包版本 ARN：

- 软件包：`arn:aws:iot:<region>:<accountID>:package/<packageName>/package`
- 程序包版本：`arn:aws:iot:<region>:<accountID>:package/<packageName>/version/<versionName>`

> **Note**
>
> 您还可以此策略中包含其它相关权限。例如，您可以为 job、thinggroup 和 jobtemplate 加入一个 ARN。有关更多信息以及策略选项的完整列表，请参阅使用AWS IoT 任务保护用户和设备。

例如，如果您的软件包和软件包版本命名如下：

- AWS IoT 东西：myThing
- 软件包名称：samplePackage
- 版本 1.0.0

该策略可能类似于以下示例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:createPackage",
                "iot:createPackageVersion",
                "iot:updatePackage",
                "iot:updatePackageVersion"
            ],
            "Resource": [
               "arn:aws:iot:us-east-1:111122223333:package/samplePackage",
```

```
                "arn:aws:iot:us-east-1:111122223333:package/samplePackage/version/1.0.0"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Resource": "arn:aws:iot:us-east-1:111122223333:thing/myThing/$package"
        }
    ]
}
```

AWS IoT 部署包版本的 Job 权限

出于安全考虑，您务必授予部署软件包和软件包版本的权限，并指定允许部署的特定软件包和软件包版本。为此，您需要创建 IAM 角色和策略，以授予使用软件包版本部署任务的权限。该策略必须将目标软件包版本指定为资源。

IAM policy

IAM policy 授予创建任务的权限，此任务包括在 `Resource` 部分中指定的软件包和版本。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJob",
                "iot:CreateJobTemplate"
            ],
            "Resource":[
                "arn:aws:iot:*:111122223333:job/<jobId>",
                "arn:aws:iot:*:111122223333:thing/<thingName>/$package",
                "arn:aws:iot:*:111122223333:thinggroup/<thingGroupName>",
                "arn:aws:iot:*:111122223333:jobtemplate/<jobTemplateName>",
                "arn:aws:iot:*:111122223333:package/<packageName>/
version/<versionName>"
            ]
        }
    ]
```

```
}
```

> **Note**
>
> 如果要部署卸载软件包和软件包版本的任务，则必须授权其中软件包版本为 $null 的 ARN，如下所示：

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$null
```

AWS IoT 更新保留的名为 shadow 的 Job 权限

要允许任务在任务成功完成时更新事物的预留名称影子，您必须创建 IAM 角色和策略。有两种方法可以在 AWS IoT 控制台中执行此操作。第一种是在控制台中创建软件包时。如果您看到为软件包管理启用依赖项对话框，则可以选择使用现有角色或创建新角色。或者，在 AWS IoT 控制台中，选择设置，选择管理索引，然后选择管理设备软件包和版本的索引。

> **Note**
>
> 如果您选择在 AWS IoT 任务成功完成时让 Job 服务更新预留的名为 shadow，则 API 调用将计入您的 Dev ice Shadow 和注册表操作，并且可能会产生费用。有关更多信息，请参阅AWS IoT Core 定价。

使用创建角色选项时，所生成角色的名称以 aws-iot-role-update-shadows 开头并包含以下策略：

设置角色

权限

权限策略授予查询和更新事物影子的权限。资源 ARN 中的 $package 参数以预留命名影子为目标。

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

```
            "Action": "iot:DescribeEndpoint",
            "Resource": ""
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Resource": [
                "arn:aws:iot:<regionCode>:111122223333:thing/<thingName>/$package"
            ]
        }
    ]
}
```

信任关系

除了权限策略外，该角色还需要与 AWS IoT Core 建立信任关系，以便实体可以代入该角色并更新预留命名影子。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

设置用户策略

iam：PassRole 权限

最后，在调用 UpdatePackageConfigurationAPI 操作 AWS IoT Core 时，您必须拥有将角色传递给的权限。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "iot:UpdatePackageConfiguration"
            ],
            "Resource": "arn:aws:iam::111122223333:role/<roleName>"
        }
    ]
}
```

AWS IoT 从 Amazon S3 下载的任务权限

任务文件保存在 Amazon S3 中。当您通过 AWS IoT Jobs 进行分派时，应参考该文件。您必须向 AWS IoT Jobs 提供下载文件的权限 (`s3:GetObject`)。您还必须在 Amazon S3 和 AWS IoT Jobs 之间设置信任关系。有关创建这些策略的说明，请参阅管理任务中的预签名 URL。

准备实例集索引

使用 AWS IoT 队列索引，您可以使用名为 shadow (`$package`) 的预留名称来搜索和聚合数据。您还可以通过查询预留命名影子和动态 AWS IoT 事物组对事物进行分组。例如，您可以找到有关哪些 AWS IoT 内容使用特定的软件包版本、未安装特定的软件包版本或未安装任何软件包版本的信息。您可以通过组合属性来获得进一步的见解。例如，识别具有特定版本且属于特定事物类型的事物（例如版本 1.0.0 和事物类型 pump_sensor）。有关更多信息，请参阅实例集索引。

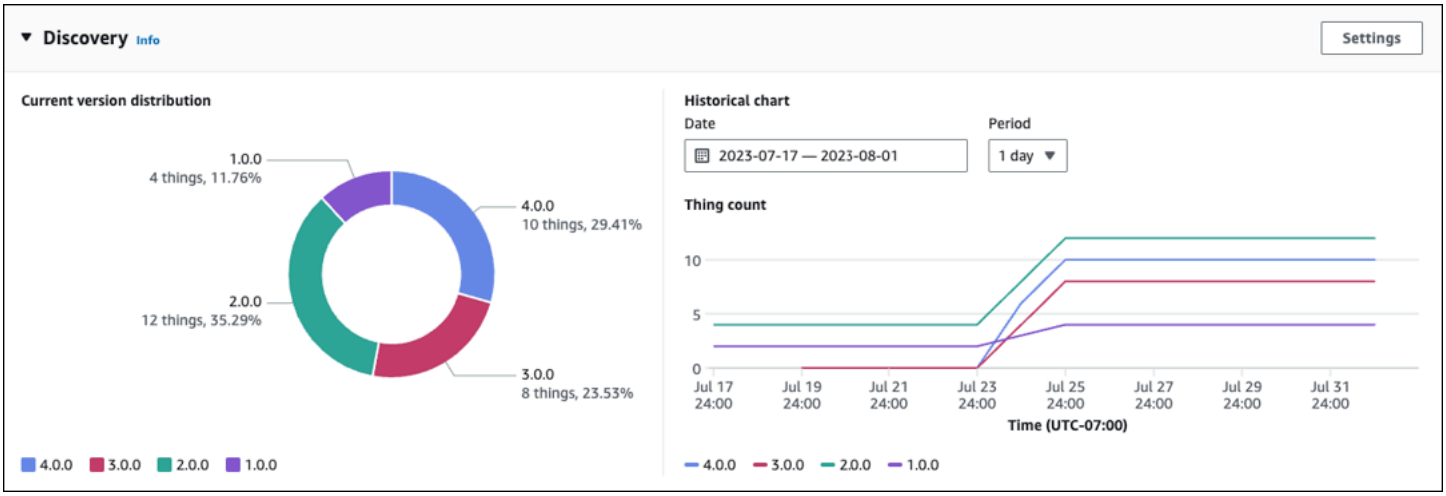
将 $package 影子设置为数据来源

要在软件包目录中使用实例集索引，必须启用实例集索引，将命名影子设置为数据来源，并将 `$package` 定义为命名影子筛选条件。如果您尚未启用实例集索引，则可以在此过程中将其启用。从控制台中的 AWS IoT Core，打开设置，选择管理索引，然后依次选择添加命名影子、添加设备软件包和版本和更新。有关更多信息，请参阅管理事物索引。

或者，您可以在创建第一个软件包时启用实例集索引。出现为软件包管理启用依赖项对话框时，选择将设备软件包和版本作为数据来源添加到实例集索引的选项。通过选择此选项，还可以启用实例集索引。

> **Note**
>
> 为软件包目录启用实例集索引会产生标准服务成本。有关更多信息，请参阅 AWS IoT Device Management定价。

控制台中显示的指标



在 AWS IoT 控制台软件包详细信息页面上，Discovery 面板显示通过$package阴影获取的标准指标。

- 当前版本分布图显示了与该软件包关联的所有设备中与某 AWS IoT 件事物关联的 10 个最新软件包版本的设备数量和百分比。注意：如果软件包的软件包版本多于图表中标注的版本，则可以发现它们分组在其它中。
- 历史图表显示指定时间段内与所选软件包版本关联的设备数量。该图表最初为空，直至您选择多达 5 个软件包版本并定义日期范围和时间间隔。要选择图表的参数，请选择设置。历史图表中显示的数据可能与当前版本分发图不同，这是因为它们显示的软件包版本数量不同，也因为您可以在历史图表中选择要分析的软件包版本。注意：当您选择要可视化的软件包版本时，它将计入实例集指标的最大数量限制。有关更多信息，请参阅实例集限制和限额。

有关深入了解收集软件包版本分发的另一种方法，请参阅通过 getBucketsAggregation 收集软件包版本分发。

查询模式

对软件包目录实施实例集索引将使用大多数支持的特征（例如，术语和短语以及搜索字段），这些特征是实例集索引的标准特征。例外情况是，comparison 和 range 查询不可用于预留命名影子（$package）version 键。但是，这些查询可用于 attributes 键。有关更多信息，请参阅查询语法。

示例数据

注意：有关预留命名影子及其结构的信息，请参阅预留命名影子。

在此示例中，第一台设备命名为 AnyThing 并安装了以下软件包：

- 软件包：SamplePackage

  程序包版本：1.0.0

  软件包 ID：1111

影子如下所示：

```
{
    "state": {
        "reported": {
            "SamplePackage": {
                "version": "1.0.0",
                "attributes": {
                    "s3UrlForSamplePackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile1",
                    "packageID": "1111"
                    }
            }
        }
    }
}
```

第二台设备命名为 AnotherThing 并安装了以下软件包：

- 软件包：SamplePackage

  程序包版本：1.0.0

软件包 ID：1111

- 软件包：OtherPackage

  程序包版本：1.2.5

  软件包 ID：2222

影子如下所示：

```
{
    "state": {
        "reported": {
            "SamplePackage": {
                "version": "1.0.0",
                "attributes": {
                    "s3UrlForSamplePackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile1",
                    "packageID": "1111"
                }
            },
            "OtherPackage": {
                "version": "1.2.5",
                "attributes": {
                    "s3UrlForOtherPackage": "https://EXAMPIEBUCKET.s3.us-
west-2.amazonaws.com/exampleCodeFile2",
                    "packageID": "2222"
                    }
            },
        }
    }
}
```

示例查询

下表根据 AnyThing 和 AnotherThing 的示例设备影子列出了查询示例。有关更多信息，请参阅[示例事物查询]。

AWS IoT Device Tester 适用于 FreeRTOS 的最新版本

所请求的信息	查询	结果
安装了特定软件包版本的事物	`shadow.name.$package.reported.SamplePackage.version:1.0.0`	AnyThing, OtherThing
未安装特定软件包版本的事物	`NOT shadow.name.$package.reported.OtherPackage.version:1.2.5`	AnyThing
使用软件包 ID 大于 1500 的软件包版本的任何设备	`shadow.name.$package.reported.*.attributes.packageID>1500"`	OtherThing
安装了特定软件包和安装了多个软件包的事物	`shadow.name.$package.reported.SamplePackage.version:1.0.0 AND shadow.name.$package.reported.totalCount:2`	OtherThing

通过 getBucketsAggregation 收集软件包版本分发

除了 AWS IoT 控制台中的发现面板外，您还可以使用 GetBucketsAggregationAPI 操作获取软件包版本分发信息。要获取软件包版本分发信息，必须执行以下操作：

- 在实例集索引中为每个软件包定义一个自定义字段。注意：创建自定义字段会计入 AWS IoT 实例集索引服务限额。
- 按如下方式格式化自定义字段：

  `shadow.name.$package.reported.<packageName>.version`

有关更多信息，请参阅 AWS IoT 舰队索引中的 “自定义字段” 部分。

准备 AWS IoT 工作

AWS IoT Device Management Software Package Catalog 通过替换参数以及与 AWS IoT 舰队索引、动态事物组和 AWS IoT 事物的保留名为 shadow 的集成来扩展 AWS IoT 任务。

> **Note**
>
> 要使用 Software Package Catalog 提供的所有功能，您必须创建以下 AWS Identity and Access Management (IAM) 角色和策略：AWS IoT 部署软件包版本的AWS IoT 任务权限和更新保留名为 shadow 的任务权限。有关更多信息，请参阅准备安全性。

AWS IoT 任务的替代参数

您可以在 AWS IoT 工作文档中使用替代参数作为占位符。当任务服务遇到替代参数时，它会将任务指向指定软件版本的属性以获取参数值。您可以使用此过程创建单个任务文档，并通过通用属性将元数据传递到任务中。例如，您可以通过软件包版本属性，将 Amazon Simple Storage Service (Amazon S3) URL、软件包 Amazon 资源名称（ARN）或签名传递到任务文档中。

在任务文档中，替代参数的格式应如下所示：

`${aws:iot:package:<packageName>:version:<versionName>:attributes:<anyAttributeNa`

在此示例中，有一个名为 `samplePackage` 的软件包，它有一个名为 `2.1.5` 的软件包版本，该版本具有以下属性：

- 名称：`s3URL`，值：`https://EXAMPIEBUCKET.s3.us-west-2.amazonaws.com/exampleCodeFile`
  - 此属性标识存储在 Amazon S3 中的代码文件的位置。
- 名称：`signature`，值：`aaaaabbbbbccccddddeeeeeffffgggggghhhhhiiiiijjjj`
  - 此属性提供了设备所需的作为安全措施的代码签名值。有关更多信息，请参阅任务的代码签名。注意：此属性是一个示例，而不是软件包目录或任务的必需属性。

对于 `downloads`，任务文档参数编写如下：

```
{
```

```
"samplePackage": "${aws:iot:package:samplePackage1:version:2.1.5:attributes:s3URL}"
}
```

对于 signature，任务文档参数编写如下：

```
{
"samplePackage": "${aws:iot:package:samplePackage1:version:2.1.5:attributes:signature}"
}
```

完整的任务文档编写如下：

```
{
  ...
  "Steps": {
    "uninstall": ["samplePackage"],
    "download": [
      {
        "samplePackage":
 "${aws:iot:package:samplePackage1:version:2.1.5:attributes:s3URL}"
      },
    ],
    "signature": [
      "samplePackage" :
 "${aws:iot:package:samplePackage1:version:2.1.5:attributes:signature}"
    ]
  }
}
```

完成替换后，将以下任务文档部署到设备上：

```
{
  ...
  "Steps": {
    "uninstall": ["samplePackage"],
    "download": [
      {
        "samplePackage": "https://EXAMPIEBUCKET.s3.us-west-2.amazonaws.com/
exampleCodeFile"
      },
    ],
    "signature": [
      "samplePackage" : "aaaaabbbbbccccddddeeeeeffffgggggghhhhhiiiiijjjj"
```

```
    ]
  }
}
```

有关 AWS IoT 作业、创建作业文档和部署作业的更多信息，请参阅作业。

准备任务文档和软件包版本以进行部署

创建软件包版本时，其`draft`状态表示正在准备部署。要为部署准备包版本，您必须创建任务文档，将文档保存到任务可以访问的位置（例如 Amazon S3），并确认包版本具有您希望任务文档使用的属性值。（注意：您只能在软件包版本`draft`处于状态时更新其属性。）

如果您对软件包版本感到满意，请通过 AWS IoT 控制台中的软件包详细信息页面或通过发布UpdatePackage版本 API 操作进行发布。然后，您可以在通过 AWS IoT 控制台或发出CreateJobAPI 操作创建任务时引用软件包版本。

部署时指定软件包和版本

部署 AWS IoT 作业时，必须命名作业部署中作业文档中命名的相同软件包和软件包版本（`destinationPackageVersions`）。否则，您将收到一条错误消息，说明缺少软件包版本。

您可以加入未包含在任务文档内的其它软件包和软件包版本。如果您这样做，则该任务不会向设备提供有关如何处理这些文件的说明，并且设备应该知道怎么做。例如，如果其它文件包含设备可能引用的数据，则可以向设备发送这些文件。

通过 AWS IoT 动态事物组定位工作

软件包目录与实例集索引、AWS IoT 任务和 AWS IoT 动态事物组配合使用来筛选和定位实例集中的设备，以选择要部署到设备上的软件包版本。您可以根据设备当前的包裹信息运行队列索引查询，并将这些内容定位到 AWS IoT 任务中。您也可以发布软件更新，但只能发布到符合条件的目标设备。例如，您可以指定只想将配置部署到当前运行 `iot-device-client 1.5.09` 的设备。有关更多信息，请参阅创建动态事物组。

预留命名影子和软件包版本

如果已配置，则 AWS IoT 任务可以在任务成功完成时更新名为 shadow (`$package`) 的事物的保留内容。如果这样做，则无需手动将软件包版本与事物的预留命名影子相关联。

在以下情况下，您可以选择手动将软件包版本关联或更新到事物的预留命名影子：

- 您 AWS IoT Core 无需关联已安装的软件包版本即可向其注册事物。
- AWS IoT 作业未配置为更新事物的预留名为 shadow。
- 您使用内部流程将软件包版本发送到您的车队，该流程 AWS IoT Core 在完成后不会更新。

> **Note**
>
> 我们建议您使用 AWS IoT Jobs 更新名为 shadow ($package) 的预留文件中的软件包版本。当 AWS IoT Jobs 也配置为更新$package影子时，通过其他流程（例如手动或编程 API 调用）更新影子中的版本参数可能会导致设备上的实际版本与报告的预留命名影子的版本不一致。

您可以通过控制台或 UpdateThingShadow API 操作，将软件包版本添加到或更新为预留命名影子（$package）。有关更多信息，请参阅将包版本与 AWS IoT 事物关联。

> **Note**
>
> 将软件包版本与 AWS IoT 事物关联不会直接更新设备软件。必须将软件包版本部署到设备才能更新设备软件。

卸载软件包及其软件包版本

$null是一个保留的占位符，它会提示 AWS IoT 作业服务从设备的预留名为 shadow $package 的软件包中删除现有的软件包和软件包版本。有关更多信息，请参阅预留命名影子。

要使用此功能，请将目标 PackageVersion Amazon 资源名称 (ARN) 末尾的版本名称替换为。$null之后，您必须指示您的服务从设备中删除该软件。

授权的 ARN 使用以下格式：

```
arn:aws:iot:<regionCode>:111122223333:package/<packageName>/version/$null
```

例如，

```
$ aws iot create-job \
    ... \
```

```
    --destinationPackageVersions ["arn:aws:iot:us-east-1:111122223333:package/
samplePackage/version/$null"]
```

软件包目录入门

您可以通过、 AWS IoT Core API 操作和 AWS Command Line Interface (AWS CLI) 来构建和维护 Soft AWS IoT Device Management ware Package 目录。 AWS Management Console

使用控制台

要使用 AWS Management Console，请登录您的 AWS 帐户并导航至AWS IoT Core。在导航窗格中，选择软件包。然后，您可以通过此部分创建和管理软件包及其版本。

使用 API 或 CLI 操作

您可以使用 AWS IoT Core API 操作来创建和管理 Software Package Catalog 功能。有关更多信息，请参阅 AWS IoT API 参考和 AWS SDK 和工具包.。这些 AWS CLI 命令还可以管理您的目录。有关更多信息，请参阅 AWS IoT CLI 命令参考。

本章包含以下部分：

- 创建软件包和软件包版本
- 通过 AWS IoT 作业部署软件包版本
- 将软件包版本与事物关联 AWS IoT

创建软件包和软件包版本

您可以使用以下步骤通过 AWS Management Console创建软件包和初始版本事物。

创建软件包

1. 登录您的 AWS 帐户并导航到AWS IoT 控制台。
2. 在导航窗格上，选择软件包。
3. 在 AWS IoT 软件包页面上，选择创建软件包。将出现为软件包管理启用依赖项对话框。
4. 在实例集索引下，选择添加设备软件包和版本。这是软件包目录所必需的，它提供了有关实例集的实例集索引和指标。
5. [可选] 如果您希望 AWS IoT 任务在任务成功完成时更新预留的命名影子，请选择 “自动更新作业中的阴影”。如果您不希望 AWS IoT 作业进行此更新，请取消选中此复选框。

6. [可选] 要授予 AWS IoT 作业更新预留名为 shadow 的权限，请在选择角色下选择创建角色。如果您不希望 AWS IoT 作业进行此更新，则不需要此角色。
7. 创建或选择一个角色。
   a. 如果您没有用于此用途的角色：当创建角色对话框出现时，输入角色名称，然后选择创建。
   b. 如果您有用于此用途的角色：对于选择角色，请选择您的角色，然后确保选中将策略附加到 IAM 角色复选框。
8. 选择确认。此时将出现创建新软件包页面。
9. 在软件包详细信息下，输入软件包名称。
10. 在软件包描述下，输入可帮助您识别和管理此软件包的信息。
11. [可选] 您可以使用标签来帮助您对该软件包进行分类和管理。要添加标签，请展开标签，选择添加标签，然后输入键/值对。您最多可以输入 50 个标签。有关更多信息，请参阅为AWS IoT 资源添加标签。

在创建新的软件包时添加软件包版本

1. 在第一个版本下，输入版本名称。

   我们建议使用SemVer 格式（例如`1.0.0.0`）来唯一标识您的软件包版本。还可以使用更适合您的用例的不同格式策略。有关更多信息，请参阅 软件包版本生命周期。
2. 在版本描述下，输入有助于您识别和管理此软件包版本的信息。

   > **Note**
   >
   > 默认版本复选框已停用，因为软件包版本是在 `draft` 状态下创建的。在创建软件包版本之后以及将状态更改为时，可以命名默认版本`published`。有关更多信息，请参阅 软件包版本生命周期。
3. [可选] 为了帮助您管理此版本或将信息传达给您的设备，请为版本属性输入一个或多个名称/值对。为您输入的每个名称/值对选择添加属性。有关更多信息，请参阅 版本属性。
4. [可选] 您可以使用标签来帮助您对该软件包进行分类和管理。要添加标签，请展开标签，选择添加标签，然后输入键/值对。您最多可以输入 50 个标签。有关更多信息，请参阅为AWS IoT 资源添加标签。
5. 选择 Create package (创建软件包)。随即显示 AWS IoT 软件包页面，您的软件包将列在软件包表中。

6. [可选] 要查看有关您创建的软件包和软件包版本的信息，请选择您的软件包名称。此时将显示软件包详细信息页面。

通过 AWS IoT 作业部署软件包版本

您可以使用以下步骤通过 AWS Management Console部署软件包版本。

先决条件：

开始之前，请执行以下操作：

- 向注册 AWS IoT 事物 AWS IoT Core. 有关向其中添加设备的说明 AWS IoT Core，请参阅创建事物对象。
- [可选] 创建 AWS IoT 事物组或动态事物组，将要部署软件包版本的设备作为目标。有关创建事物组的说明，请参阅创建静态事物组。有关创建动态事物组的说明，请参阅创建动态事物组。
- 创建软件包和软件包版本。有关更多信息，请参阅 创建软件包和软件包版本。
- 创建任务文档。有关更多信息，请参阅准备任务文档和软件包版本以进行部署。

部署作 AWS IoT 业

1. 在 AWS IoT 控制台上，选择软件包。
2. 选择要部署的软件包。此时将显示软件包详细信息页面。
3. 在版本下选择要部署的软件包版本，然后选择部署任务版本。
4. 如果这是您第一次通过此门户部署任务，则会出现一个描述要求的对话框。检查信息并选择确认。
5. 输入部署的名称或在名称字段中保留自动生成的名称。
6. [可选] 在描述字段中，输入描述以标识部署的目的或内容，或者保留自动生成的信息。

   注意：我们建议您不要在任务名称和描述字段中使用个人身份信息。
7. [可选] 添加要与此任务关联的所有标签。
8. 选择下一步。
9. 在任务目标下，选择应接收任务的事物或事物组。
10. 在任务文件字段中，指定任务文档 JSON 文件。
11. 打开与软件包目录服务的任务集成。
12. 选择在任务文档中指定的软件包和版本。

> **Note**
>
> 您需要选择在任务文档中指定的相同软件包和软件包版本。您可以包含更多内容，但该任务将仅针对任务文档中包含的软件包和版本发出说明。有关更多信息，请参阅部署时命名软件包和版本。

13. 选择下一步。
14. 在“任务配置”页面上，在“任务配置”对话框中选择以下任务类型之一：
    - 快照任务：快照任务在目标设备和组上完成运行后即完成。
    - 连续任务：连续任务适用于事物组，并会在随后添加到指定目标组的任何设备上运行。
15. 在其它配置 - 可选对话框中，查看以下可选任务配置并相应进行选择。有关更多信息，请参阅任务推出、计划和中止配置以及任务执行超时和重试配置。
    - 推出配置
    - 计划配置
    - 任务执行超时配置
    - 任务执行重试配置
    - 中止配置
16. 查看任务选择，然后选择提交。

在您创建任务后，控制台会生成一个 JSON 签名并将其放在您的任务文档中。您可以使用 AWS IoT 控制台查看任务的状态，也可以取消或删除作业。要管理任务，请转到控制台的 Job 中心。

将软件包版本与事物关联 AWS IoT

在设备上安装软件后，您可以将软件包版本与名为 shadow AWS IoT 的预留内容关联起来。如果已将 AWS IoT 作业配置为在任务部署并成功完成后更新事物的预留名为 shadow，则无需完成此过程。有关更多信息，请参阅 预留命名影子。

先决条件：

开始之前，请执行以下操作：

- 创造一个或 AWS IoT 多个事物，并通过 AWS IoT Core它建立遥测。有关更多信息，请参阅入门 AWS IoT Core。

- 创建软件包和软件包版本。有关更多信息，请参阅 创建软件包和软件包版本。
- 在设备上安装软件包版本软件。

> **Note**
>
> 将软件包版本与 AWS IoT 事物关联不会在物理设备上更新或安装软件。软件包版本必须部署到设备上。

将包版本与 AWS IoT 事物关联

1. 在 AWS IoT 控制台导航窗格上，展开所有设备菜单并选择事物。
2. 从列表中确定要更新的 AWS IoT 事物，然后选择事物名称以显示其详细信息页面。
3. 在详细信息部分中，选择软件包和版本。
4. 选择添加到软件包和版本。
5. 对于选择设备软件包，选择所需的软件包。
6. 对于选择版本，选择所需的软件版本。
7. 选择添加设备软件包。

   软件包和版本显示在选定的软件包和版本列表中。
8. 对要与此事物关联的每个软件包和版本重复这些步骤。
9. 完成后，选择添加软件包和版本详细信息。事物详细信息页面打开，您可以在列表中看到新的软件包和版本。

AWS IoT Core 设备位置

在使用 AWS IoT Core 设备定位功能之前，请查看此功能的条款和条件。请注意， AWS 可能会将您的地理位置搜索请求参数（例如用于进行搜索的位置数据和其他信息）传输给您选择的第三方数据提供商，而第三方数据提供商可能不在 AWS 区域 您当前使用的数据提供商范围内。有关更多信息，请参阅 AWS 服务条款。

使用 AWS IoT Core 设备位置使用第三方求解器测试物联网设备的位置。求解器是第三方供应商提供的算法，用于解析测量数据并估计设备的位置。通过确定设备的位置，您可以在现场跟踪和调试设备以解决任何问题。

将解析从各种来源收集的测量数据，并将地理位置信息报告为 GeoJSON 有效负载。GeoJSON 格式是一种用于对地理数据结构进行编码的格式。此有效负载中包含设备位置的纬度和经度坐标，这些坐标基于世界大地坐标系统 (WGS84)。

主题

测量类型和求解器

AWS IoT Core 设备定位与第三方供应商合作，解析测量数据并提供估计的设备位置。下表显示了测量类型和第三方位置求解器，以及有关支持设备的信息。有关 LoRa WAN 设备以及为其配置设备位置的信息，请参阅配置 LoRa WAN 资源的位置。

测量类型和求解器

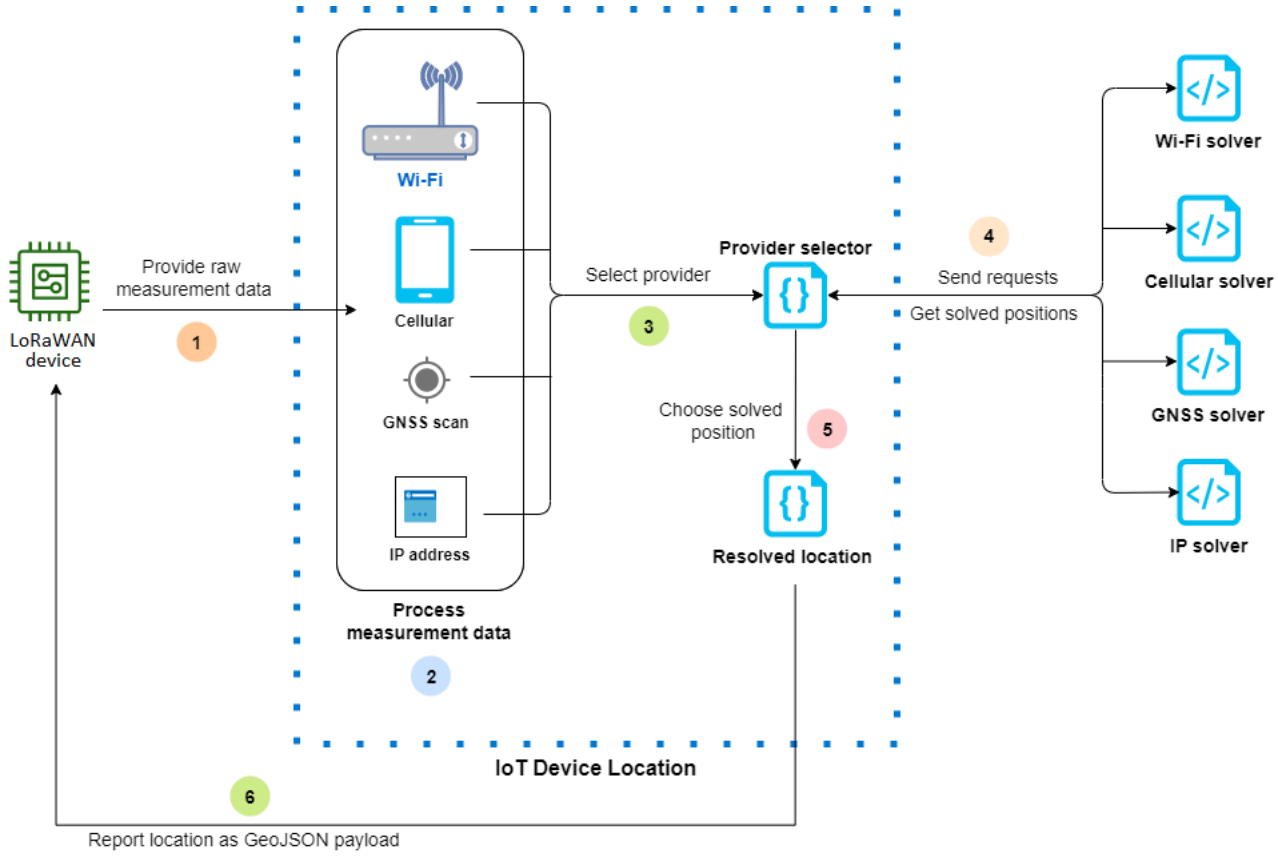
测量类型	第三方求解器	支持的设备
Wi-Fi 接入点	基于 Wi-Fi 的求解器	一般物联网设备和 LoRa广域网设备

测量类型	第三方求解器	支持的设备
蜂窝无线电发射塔：GSM、LTE、CDMA、SCDMA、WCMDA 和 TD-SCDMA 数据	基于蜂窝的求解器	一般物联网设备和 LoRa广域网设备
IP 地址	IP 反向查找求解器	一般 IoT 设备
GNSS 扫描数据（导航消息）	GNSS 求解器	一般物联网设备和 LoRa广域网设备

有关位置求解器的更多信息，以及显示各种测量类型的设备有效负载的示例，请参阅位置求解器和设备有效负载。

AWS IoT Core 设备定位的工作原理

下图显示了 AWS IoT Core 设备定位如何收集测量数据并解析设备的位置信息。



以下步骤显示了 AWS IoT Core 设备定位的工作原理。

1. 接收测量数据

   首先从您的设备发送与设备位置相关的原始测量数据。测量数据被指定为 JSON 有效负载。

2. 处理测量数据

   对测量数据进行处理，AWS IoT Core 设备定位选择要使用的测量数据，这些数据可以是 Wi-Fi、蜂窝网络、GNSS 扫描或 IP 地址信息。

3. 选择求解器

   根据测量数据选择第三方求解器。例如，如果测量数据中包含 Wi-Fi 和 IP 地址信息，则会选择 Wi-Fi 求解器和 IP 反向查找求解器。

4. 获取已解析的位置

   API 请求会发送给求解器提供商，请求解析位置。 AWS IoT Core 然后，设备位置会从求解器那里获取估计的地理位置信息。

5. 选择已解析的位置

   将比较已解析的位置信息及其准确性， AWS IoT Core 设备位置选择精度最高的地理定位结果。

6. 输出位置信息

   将地理位置信息作为 GeoJSON 有效负载发送给您。此有效负载中包含 WGS84 地理坐标、准确性信息、置信度，以及获得解析位置的时间戳。

如何使用 AWS IoT Core 设备定位

以下步骤说明如何使用 AWS IoT Core 设备定位。

1. 提供测量数据

   将与设备位置相关的原始测量数据指定为 JSON 有效负载。要检索有效载荷测量数据，请转到您的设备日志，或者使用 CloudWatch 日志，然后复制有效载荷数据信息。JSON 有效负载中必须包含一种或多种类型的数据测量。有关显示各种求解器有效负载格式的示例，请参阅位置求解器和设备有效负载。

2. 解析位置信息

   使用 AWS IoT 控制台中的设备位置页面或 GetPositionEstimateAPI 操作，传递有效载荷测量数据并解析设备位置。 AWS IoT Core 然后，设备位置选择精度最高的求解器并报告设备位置。有关更多信息，请参阅 解析 IoT 设备的位置。

3. 复制位置信息

   验证由 AWS IoT Core 设备位置解析并报告为 GeoJSON 有效负载的地理位置信息。您可以复制有效负载以用于您的应用程序和其他 AWS 服务应用程序。例如，您可以使用位置 AWS IoT 规则操作将您的地理位置数据发送到 Amazon Location Service。

以下主题介绍如何使用 AWS IoT Core 设备定位和设备位置负载示例。

- 解析 IoT 设备的位置
- 位置求解器和设备有效负载

解析 IoT 设备的位置

使用 AWS IoT Core 设备定位来解码来自设备的测量数据，并使用第三方求解器解析设备位置。解析后的位置以 GeoJSON 有效负载的形式生成，其中包含地理坐标和准确性信息。您可以通过 AWS IoT 控制台、 AWS IoT Wireless API 或解析设备的位置 AWS CLI。

主题

- 解析设备位置（控制台）
- 解析设备位置 (API)
- 对解析位置时出现的错误进行故障排查

解析设备位置（控制台）

解析设备位置（控制台）

1. 转到 AWS IoT 控制台中的设备位置页面。
2. 从设备日志或日志中获取有效载荷测量数据，然后将其输入到 CloudWatch 通过负载解析位置部分。

以下代码显示了 JSON 有效负载示例。有效负载包含蜂窝和 Wi-Fi 测量数据。如果您的有效负载包含其他类型的测量数据，将使用准确性最高的求解器。有关更多信息以及有效负载示例，请参阅the section called “位置求解器和设备有效负载”。

> **Note**
>
> JSON 有效负载中必须包含一种类型的数据测量。

```
{
    "Timestamp": 1664313161,
    "Ip":{
        "IpAddress": "54.240.198.35"
    },
    "WiFiAccessPoints": [{
        "MacAddress": "A0:EC:F9:1E:32:C1",
        "Rss": -77
    }],
    "CellTowers": {
        "Gsm": [{
            "Mcc": 262,
            "Mnc": 1,
            "Lac": 5126,
            "GeranCid": 16504,
            "GsmLocalId": {
                "Bsic": 6,
                "Bcch": 82
            },
            "GsmTimingAdvance": 1,
            "RxLevel": -110,
            "GsmNmr": [{
                "Bsic": 7,
                "Bcch": 85,
                "RxLevel": -100,
                "GlobalIdentity": {
                    "Lac": 1,
                    "GeranCid": 1
                }
            }]
        }],
        "Wcdma": [{
```

```
            "Mcc": 262,
            "Mnc": 7,
            "Lac": 65535,
            "UtranCid": 14674663,
            "WcdmaNmr": [{
                    "Uarfcndl": 10786,
                    "UtranCid": 14674663,
                    "Psc": 149
                },
                {
                    "Uarfcndl": 10762,
                    "UtranCid": 14674663,
                    "Psc": 211
                }
            ]
        }],
        "Lte": [{
            "Mcc": 262,
            "Mnc": 2,
            "EutranCid": 2898945,
            "Rsrp": -50,
            "Rsrq": -5,
            "LteNmr": [{
                    "Earfcn": 6300,
                    "Pci": 237,
                    "Rsrp": -60,
                    "Rsrq": -6,
                    "EutranCid": 2898945
                },
                {
                    "Earfcn": 6300,
                    "Pci": 442,
                    "Rsrp": -70,
                    "Rsrq": -7,
                    "EutranCid": 2898945
                }
            ]
        }]
    }
}
```

3. 要解析位置信息，请选择 Resolve（解析）。

位置信息属于类型 blob 并作为使用 GeoJSON 格式的有效负载返回，此格式用于对地理数据结构进行编码。有效负载中包含：

- WGS84 地理坐标，包括纬度和经度信息。还可能包含海拔信息。
- 报告的位置信息类型，例如 Point（点）。点位置类型将位置表示为 WGS84 纬度和经度，编码为 GeoJSON 点。
- 水平和垂直准确性信息，表示解析器估计的位置信息与实际设备位置之间的差异（单位为米）。
- 置信水平，表示位置估计响应中的不确定性。默认值为 0.68，表示实际设备位置在估计位置的不确定性半径内的概率为 68%。
- 设备所在的城市、省/直辖市/自治区、国家/地区和邮政编码。只有在使用 IP 反向查找解析器时，才会报告此信息。
- 时间戳信息与解析该位置的日期和时间相对应。它使用 Unix 时间戳格式。

以下代码显示了解析位置后返回的 GeoJSON 有效负载示例。

> **Note**
>
> 如果 “ AWS IoT Core 设备位置” 在尝试解析位置时报告错误，则可以对错误进行故障排除并解决位置问题。有关更多信息，请参阅 对解析位置时出现的错误进行故障排查。

```
{
    "coordinates": [
        13.376076698303223,
        52.51823043823242
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 45,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 303,
        "horizontalConfidenceLevel": 0.68,
        "country": "USA",
        "state": "CA",
        "city": "Sunnyvalue",
        "postalCode": "91234",
        "timestamp": "2022-11-18T12:23:58.189Z"
    }
```

```
}
```

4. 前往资源位置部分并验证 AWS IoT Core 设备位置报告的地理位置信息。您可以复制有效负载以用于其他应用程序 AWS 服务和。例如，您可以使用 位置 将地理位置数据发送到 Amazon Location Service。

解析设备位置 (API)

要使用 AWS IoT Wireless API 解析设备位置，请使用 GetPositionEstimateAPI 操作或 get-position-estimateCLI 命令。将有效负载测量数据指定为输入，然后运行 API 操作以解析设备位置。

> **Note**
>
> GetPositionEstimate API 操作不存储任何设备或状态信息，也不能用于检索历史位置数据。它执行一次性操作，解析测量数据并生成估计位置。要检索位置信息，每次执行此 API 操作时都必须指定有效负载信息。

以下命令显示了如何使用此 API 操作解析位置的示例。

> **Note**
>
> 运行 get-position-estimate CLI 命令时，必须将输出 JSON 文件指定为第一个输入。此 JSON 文件将以 GeoJSON 格式存储作为响应从 CLI 获得的估计位置信息。例如，以下命令将位置信息存储在 *locationout.json* 文件中。

```
aws iotwireless get-position-estimate locationout.json \
    --ip IpAddress=""54.240.198.35"" \
    --wi-fi-access-points \
        MacAddress="A0:EC:F9:1E:32:C1",Rss=-75 \
        MacAddress="A0:EC:F9:15:72:5E",Rss=-67
```

此示例包括 Wi-Fi 接入点和 IP 地址作为测量类型。 AWS IoT Core 设备位置在 Wi-Fi 求解器和 IP 反向查找求解器之间进行选择，然后选择精度更高的求解器。

解析的位置作为使用 GeoJSON 格式的有效负载返回，此格式用于对地理数据结构进行编码。然后将其存储在 *locationout.json* 文件中。此有效负载中包含 WGS84 经度和纬度坐标、准确性和置信度信息、位置数据类型，以及解析位置的时间戳。

```
{
    "coordinates": [
        13.37704086303711,
        52.51865005493164
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 707,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 389,
        "horizontalConfidenceLevel": 0.68,
        "country": "USA",
        "state": "CA",
        "city": "Sunnyvalue",
        "postalCode": "91234",
        "timestamp": "2022-11-18T14:03:57.391Z"
    }
}
```

对解析位置时出现的错误进行故障排查

尝试解析位置时，可能会看到以下任何错误代码。 AWS IoT Core 使用 `GetPositionEstimate` API 操作时，设备位置可能会生成错误，或者引用 AWS IoT 控制台中与错误对应的行号。

- 400 错误

  此错误表示设备有效负载 JSON 的格式无法通过 AWS IoT Core 设备位置进行验证。出现此错误可能是因为：

  - JSON 测量数据的格式不正确。
  - 有效负载中仅包含时间戳信息。
  - 诸如 IP 地址之类的测量数据参数无效。

  要解决此错误，请检查您的 JSON 格式是否正确，是否包含来自一种或多种测量类型的数据作为输入。如果 IP 地址无效，有关如何提供有效 IP 地址来解决此错误的信息，请参阅 IP 反向查找求解器。

- 403 错误

  此错误表示您无权执行 API 操作或使用 AWS IoT 控制台检索设备位置。要解决此错误，请验证您是否拥有执行此操作所需的权限。如果您的 AWS Management Console 会话或会 AWS CLI 话令牌已

过期，则可能会发生此错误。要解决此错误，请刷新会话令牌以使用 AWS CLI，或者注销， AWS Management Console 然后使用您的凭据登录。

- 404 错误

  此错误表明 “ AWS IoT Core 设备位置” 未找到或解决任何位置信息。此错误可能是由于测量数据输入中数据不足等原因造成的。例如：

  - MAC 地址或蜂窝发射塔信息不足。
  - IP 地址不可用于查找和检索位置。
  - GNSS 有效负载不足。

  要解决此类情况下出现的错误，请检查您的测量数据是否包含解析设备位置所需的足够信息。

- 404 错误

  此错误表明，当 AWS IoT Core 设备位置功能尝试解析位置时，出现了内部服务器异常。要尝试修复此错误，请刷新会话，并重试发送需要解析的测量数据。

使用 AWS IoT Core 设备位置 MQTT 主题解析设备位置

您可以使用保留的 MQTT 主题通过设备定位功能获取设备的最新位置信息。 AWS IoT Core

设备位置 MQTT 主题的格式

AWS IoT Core 设备定位的保留主题使用以下前缀：

`$aws/device_location/{customer_device_id}/`

要创建完整的主题，请先将 *customer_device_id* 替换为用于识别设备的唯一 ID。我们建议您指定`WirelessDeviceId`，例如 LoRa WAN 和 Sidewalk 设备*thingName*，以及您的设备是否已注册为 AWS IoT 事物。然后在主题后面附加主题存根，如下一节所示的 `get_position_estimate` 或 `get_position_estimate/accepted`。

> **Note**
>
> *{customer_device_id}* 仅可包含字母、数字和短划线。订阅设备位置主题时，只能使用加号（ + ）作为通配符。例如，您可以将 + 通配符用于 *{customer_device_id}* 来获取设备的位置信息。当您订阅主题 `$aws/device_location/+/get_position_estimate/`

accepted 时，系统将发布一条消息，其中包含与任何设备 ID 匹配的设备的位置信息（如果成功解析）。

以下是用于与 AWS IoT Core 设备位置进行交互的保留主题。

设备位置 MQTT 主题

主题	允许的操作	描述
$aws/device_location/*customer_device_id*/get_position_estimate	Publish	设备向本主题发布信息，以获取扫描的原始测量数据，以便通过 “ AWS IoT Core 设备位置” 进行解析。
$aws/device_location/*customer_device_id*/get_position_estimate/accepted	订阅	AWS IoT Core 设备定位成功解析设备位置后，会向该主题发布位置信息。
$aws/device_location/*customer_device_id*/get_position_estimate/rejected	订阅	AWS IoT Core 当设备位置无法解析设备位置时，它会将错误信息发布到此主题。

设备位置 MQTT 主题的策略

要接收来自设备位置主题的消息，您的设备必须使用允许其连接到 AWS IoT 设备网关并订阅 MQTT 主题的策略。

以下是接收各种主题的消息时需要的策略示例。

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

```
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/device_location/customer_device_id/
get_position_estimate"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/device_location/customer_device_id/
get_position_estimate/accepted",
        "arn:aws:iot:region:account:topic/$aws/device_location/customer_device_id/
get_position_estimate/rejected"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/
device_location/customer_device_id/get_position_estimate/accepted",
        "arn:aws:iot:region:account:topicfilter/$aws/
device_location/customer_device_id/get_position_estimate/rejected"
      ]
    }
  ]
}
```

设备位置主题和有效负载

下面显示了 “ AWS IoT Core 设备位置” 主题、其消息负载的格式以及每个主题的示例策略。

主题

- /get_position_estimate
- /get_position_estimate/accepted
- /get_position_estimate/rejected

/get_position_estimate

向此主题发布一条消息，以获取设备上的原始测量数据，以供 AWS IoT Core 设备位置解析。

```
$aws/device_location/customer_device_id/get_position_estimate
```

AWS IoT Core 设备位置通过发布到/get_position_estimate/accepted或来响应/get_position_estimate/rejected。

> **Note**
>
> 发布到该主题的消息必须是有效的 JSON 有效负载。如果输入消息不是有效的 JSON 格式，则不会得到任何响应。有关更多信息，请参阅消息有效负载。

消息负载

消息有效负载格式遵循与 AWS IoT Wireless API 操作请求正文 GetPositionEstimate 类似的结构。其中包含：

- 一个可选 Timestamp 字符串，对应于解析该位置的日期和时间。Timestamp 字符串的最小长度可以为 1，最大长度可以为 10。
- 一个可选 MessageId 字符串，可用于将请求映射到响应。如果您指定此字符串，则发布到 get_position_estimate/accepted 或 get_position_estimate/rejected 主题的消息将包含此 MessageId。MessageID 字符串的最小长度可以为 1，最大长度可以为 256。
- 来自包含以下一种或多种测量类型的设备的测量数据：
  - WiFiAccessPoint
  - CellTowers
  - IpAddress
  - Gnss

以下内容显示了示例消息有效负载示例。

```
{
    "Timestamp": "1664313161",
    "MessageId": "ABCD1",
    "WiFiAccessPoints":  [
```

```
        {
            "MacAddress": "A0:EC:F9:1E:32:C1",
            "Rss": -66
        }
    ],
    "Ip":{
        "IpAddress": "54.192.168.0"
    },
    "Gnss":{
      "Payload":"8295A614A2029517F4F77C0A7823B161A6FC57E25183D96535E3689783F6CA48",
      "CaptureTime":1354393948
    }
}
```

策略示例

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/device_location/customer_device_id/
get_position_estimate"
      ]
    }
  ]
}
```

/get_position_estimate/accepted

AWS IoT Core 设备定位在返回设备已解析的位置信息时，会发布对此主题的回复。位置信息以 GeoJSON 格式返回。

```
$aws/device_location/customer_device_id/get_position_estimate/accepted
```

下面显示了消息有效负载和示例策略。

消息负载

以下是 GeoJSON 格式的消息有效负载示例。如果您在原始测量数据`MessageId`中指定了，并且 AWS IoT Core 设备位置成功解析了位置信息，则消息负载将返回相同`MessageId`的信息。

```
{
    "coordinates": [
        13.37704086303711,
        52.51865005493164
    ],
    "type": "Point",
    "properties": {
        "verticalAccuracy": 707,
        "verticalConfidenceLevel": 0.68,
        "horizontalAccuracy": 389,
        "horizontalConfidenceLevel": 0.68,
        "country": "USA",
        "state": "CA",
        "city": "Sunnyvalue",
        "postalCode": "91234",
        "timestamp": "2022-11-18T14:03:57.391Z",
        "messageId": "ABCD1"
    }
}
```

策略示例

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/
device_location/customer_device_id/get_position_estimate/accepted"
      ]
    },
    {
```

```
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/device_location/customer_device_id/
get_position_estimate/accepted"
      ]
    }
  ]
}
```

/get_position_estimate/rejected

AWS IoT Core 当设备位置无法解析设备位置时，它会发布对此主题的错误响应。

```
$aws/device_location/customer_device_id/get_position_estimate/rejected
```

下面显示了消息有效负载和示例策略。有关这些错误的信息，请参阅[对解析位置时出现的错误进行故障排查](#)。

消息负载

以下是提供错误代码和消息的消息负载示例，这表明了 AWS IoT Core 设备定位未能解析位置信息的原因。如果您`MessageId`在提供原始测量数据时指定了，而 AWS IoT Core 设备位置未能解析位置信息，则消息负载中将返回相同`MessageId`的信息。

```
{
    "errorCode": 500,
    "errorMessage":"Internal server error",
    "messageId": "ABCD1"
}
```

策略示例

以下是所需策略的示例：

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

```
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topicfilter/$aws/
device_location/customer_device_id/get_position_estimate/rejected"
      ]
    },
    {
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account:topic/$aws/device_location/customer_device_id/
get_position_estimate/rejected"
      ]
    }
  ]
}
```

位置求解器和设备有效负载

位置求解器是可用于解析物联网设备位置的算法。 AWS IoT Core 设备定位支持以下位置求解器。您将看到这些测量类型的 JSON 有效负载格式的示例，求解器支持的设备，以及如何解析位置。

要解析设备位置，请指定一种或多种这些测量数据类型。将为所有测量数据组合返回单个解析位置。

主题

- 基于 Wi-Fi 的求解器
- 基于蜂窝的求解器
- IP 反向查找求解器
- GNSS 求解器

基于 Wi-Fi 的求解器

使用基于 Wi-Fi 的求解器和来自 Wi-Fi 接入点的扫描信息来解析位置。求解器支持 WLAN 技术，可用于计算一般物联网设备和 W LoRa AN 无线设备的设备位置。

LoRa广域网设备必须配备 LoRa Edge 芯片组，该芯片组可以解码传入的 Wi-Fi 扫描信息。 LoRa Edge 是一个超低功耗平台，它集成了远程 LoRa 收发器、多星座 GNSS 扫描器和针对地理定位应用的

被动 Wi-Fi MAC 扫描器。收到来自设备的上行链路消息时，Wi-Fi 扫描数据将发送到 AWS IoT Core 设备位置，并根据 Wi-Fi 扫描结果估算位置。然后将解码后的信息传递给基于 Wi-Fi 的求解器，以检索位置信息。

基于 Wi-Fi 的求解器有效负载示例

以下代码显示了来自包含测量数据的设备的 JSON 有效负载示例。当 AWS IoT Core Device Location 收到这些数据作为输入时，它会向求解器提供者发送 HTTP 请求以解析位置信息。要检索此信息，请指定 MAC 地址和 RSS（接收的信号强度）的值。为此，要么使用此格式提供 JSON 负载，要么使用 GetPositionEstimateAPI 操作的WiFiAccessPoints对象参数。

```
{
    "Timestamp": 1664313161,    // optional
    "WiFiAccessPoints": [
        {
            "MacAddress": "A0:EC:F9:1E:32:C1",  // required
            "Rss": -75                          // required
        }
    ]
}
```

基于蜂窝的求解器

您可以使用基于蜂窝的求解器和从蜂窝无线发射塔获得的测量数据来解析位置。此求解器支持以下技术。即使包含来自任何或所有这些技术的测量数据，也只能获得单个已解析的位置信息。

- GSM
- CDMA
- WCDMA
- TD-SCDMA
- LTE

基于蜂窝的求解器有效负载示例

以下代码显示了来自包含蜂窝测量数据的设备的 JSON 有效负载示例。当 AWS IoT Core Device Location 收到这些数据作为输入时，它会向求解器提供者发送 HTTP 请求以解析位置信息。要检索信息，您可以在控制台中使用此格式提供 JSON 负载，或者为 GetPositionEstimateAPI 操作的CellTowers参数指定值。您可以通过使用任何或所有这些蜂窝技术指定参数值来提供测量数据。

LTE（长期演变）

使用此测量数据时，必须指定诸如移动网络的网络和国家/地区代码之类的信息，以及包括有关本地 ID 的信息在内的其他可选参数。以下代码显示了有效负载格式的示例。有关这些参数的更多信息，请参阅 LTE 对象。

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Lte": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "EutranCid": int,             // required. Make sure that you use int for
 EutranCid.
          "Tac": int,                   // optional
          "LteLocalId": {               // optional
              "Pci": int,               // required
              "Earfcn": int,            // required
          },
          "LteTimingAdvance": int,      // optional
          "Rsrp": int,                  // optional
          "Rsrq": float,                // optional
          "NrCapable": boolean,         // optional
          "LteNmr": [                   // optional
                {
                    "Pci": int,         // required
                    "Earfcn": int,      // required
                    "EutranCid": int,   // required
                    "Rsrp": int,        // optional
                    "Rsrq": float       // optional
                }
            ]
         }
      ]
   }
}
```

GSM（全球移动通信系统）

使用此测量数据时，必须指定诸如移动网络的网络和国家/地区代码之类的信息、基站信息，以及其他可选参数等信息。以下代码显示了有效负载格式的示例。有关这些参数的更多信息，请参阅 GSM 对象。

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Gsm": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "Lac": int,                   // required
          "GeranCid": int,              // required
          "GsmLocalId": {               // optional
              "Bsic": int,              // required
              "Bcch": int,              // required
          },
          "GsmTimingAdvance": int,      // optional
          "RxLevel": int,               // optional
          "GsmNmr": [                   // optional
            {
                "Bsic": int,            // required
                "Bcch": int,            // required
                "RxLevel": int,         // optional
                "GlobalIdentity": {
                    "Lac": int,         // required
                    "GeranCid": int     // required
                }
             }
          ]
       }
    ]
}
```

CDMA（码分多址）

使用此测量数据时，必须指定诸如信号功率和标识信息、基站信息，以及其他可选参数等信息。以下代码显示了有效负载格式的示例。有关这些参数的更多信息，请参阅 CDMA 对象。

```
{
    "Timestamp": 1664313161,              // optional
    "CellTowers": {
        "Cdma": [
        {
            "SystemId": int,               // required
            "NetworkId": int,              // required
            "BaseStationId": int,          // required
```

```
            "RegistrationZone": int,        // optional
            "CdmaLocalId": {                // optional
              "PnOffset": int,              // required
              "CdmaChannel": int,           // required
            },
            "PilotPower": int,              // optional
            "BaseLat": float,               // optional
            "BaseLng": float,               // optional
            "CdmaNmr": [                    // optional
                {
                    "PnOffset": int,        // required
                    "CdmaChannel": int,     // required
                    "PilotPower": int,      // optional
                    "BaseStationId": int    // optional
                }
             ]
          }
       ]
    }
}
```

WCDMA（宽带码分多址）

使用此测量数据时，必须指定诸如网络和国家/地区代码、信号功率和标识信息、基站信息，以及其他可选参数等信息。以下代码显示了有效负载格式的示例。有关这些参数的更多信息，请参阅 CDMA 对象。

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Wcdma": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "UtranCid": int,              // required
          "Lac": int,                   // optional
          "WcdmaLocalId": {             // optional
              "Uarfcndl": int,          // required
              "Psc": int,               // required
          },
          "Rscp": int,                  // optional
          "Pathloss": int,              // optional
          "WcdmaNmr": [                 // optional
```

```
                {
                  "Uarfcndl": int,      // required
                  "Psc": int,           // required
                  "UtranCid": int,      // required
                  "Rscp": int,          // optional
                  "Pathloss": int,      // optional
                }
             ]
          }
       ]
    }
}
```

TD-SCDMA（时分同步码分多址）

使用此测量数据时，必须指定诸如网络和国家/地区代码、信号功率和标识信息、基站信息，以及其他可选参数等信息。以下代码显示了有效负载格式的示例。有关这些参数的更多信息，请参阅 CDMA 对象。

```
{
    "Timestamp": 1664313161,           // optional
    "CellTowers": {
        "Tdscdma": [
        {
          "Mcc": int,                   // required
          "Mnc": int,                   // required
          "UtranCid": int,              // required
          "Lac": int,                   // optional
          "TdscdmaLocalId": {           // optional
              "Uarfcn": int,            // required
              "CellParams": int,        // required
          },
          "TdscdmaTimingAdvance": int,  // optional
          "Rscp": int,                  // optional
          "Pathloss": int,              // optional
          "TdscdmaNmr": [               // optional
                {
                  "Uarfcn": int,        // required
                  "CellParams": int,    // required
                  "UtranCid": int,      // optional
                  "Rscp": int,          // optional
                  "Pathloss": int,      // optional
                }
```

```
            ]
         }
      ]
   }
}
```

IP 反向查找求解器

您可以使用 IP 反向查找求解器，将 IP 地址作为输入来解析位置。求解器可以从已配置的设备中获取位置信息。 AWS IoT使用 IPv4 或 IPv6 标准模式或 IPv6 十六进制压缩模式的格式指定 IP 地址信息。然后，将获得已解析的估计位置，包括设备所在的城市和国家/地区等其他信息。

> **Note**
>
> 使用 IP 反向查找，即表示您同意不将其用于识别或定位特定的家庭或街道地址。

IP 反向查找求解器有效负载示例

以下代码显示了来自包含测量数据的设备的 JSON 有效负载示例。当 AWS IoT Core Device Location 收到测量数据中的 IP 地址信息时，它会在求解器提供者的数据库中查找这些信息，然后使用该数据库来解析位置信息。要检索信息，请使用此格式提供 JSON 负载，或者为 GetPositionEstimateAPI 操作的 I p 参数指定值。

> **Note**
>
> 使用此解析器时，除了坐标外，还会报告设备所在的城市、省/直辖市/自治区、国家/地区和邮政编码。有关示例，请参阅解析设备位置（控制台）。

```
{
    "Timestamp": 1664313161,
    "Ip":{
        "IpAddress":"54.240.198.35"
      }
}
```

GNSS 求解器

使用 GNSS（全球导航卫星系统）求解器和 GNSS 扫描结果消息或 NAV 消息中包含的信息检索设备位置。可以选择提供额外的 GNSS 辅助信息，从而减少求解器搜索信号时必须使用的变量数量。通过提供这种辅助信息（包括位置、高度，以及捕获时间和准确度信息），求解器可以轻松识别视野中的卫星并计算设备位置。

此求解器可用于 LoRa WAN 设备和其他已配置的设备。 AWS IoT对于一般 IoT 设备，如果设备支持使用 GNSS 进行位置估计，当从设备接收 GNSS 扫描信息时，收发器会解析位置信息。对于 LoRa WAN 设备，这些设备必须配备 LoRa Edge 芯片组。当收到来自设备的上行链路消息时，GNSS 扫描数据将发送到 适用于 LoRaWAN 的 AWS IoT Core，并根据收发器的扫描结果估算位置。

GNSS 求解器有效负载示例

以下代码显示了来自包含测量数据的设备的 JSON 有效负载示例。当 AWS IoT Core Device Location 收到包含测量数据中有效载荷的 GNSS 扫描信息时，它会使用收发器和随附的任何其他辅助信息来搜索信号并解析位置信息。要检索信息，请使用此格式提供 JSON 有效负载，或者为 GetPositionEstimateAPI 操作的 Gnss 参数指定值。

> **Note**
>
> 在 AWS IoT Core 设备定位可以解析设备位置之前，必须从有效负载中移除目标字节。

```
{
    "Timestamp": 1664313161,                    // optional
    "Gnss": {
        "AssistAltitude": number,               // optional
        "AssistPosition": [ number ],           // optional
        "CaptureTime": number,                  // optional
        "CaptureTimeAccuracy": number,          // optional
        "Payload": "string",                    // required
        "Use2DSolver": boolean                  // optional
   }
}
```

事件消息

本节包含有关内容或作业更新或更改 AWS IoT 时发布的消息的信息。有关允许您创建检测器来监控设备是否存在故障或操作变化并在出现故障或操作时触发操作的 AWS IoT Events 服务的信息，请参阅AWS IoT Events。

事件消息的生成方式

AWS IoT 在某些事件发生时发布事件消息。例如，在添加、更新或删除事物时，由注册表生成事件。每个事件将导致发送单个事件消息。事件消息通过 MQTT 随 JSON 负载一起发布。负载的内容取决于事件的类型。

> **Note**
>
> 保证事件消息发布一次。事件消息可能会多次发布。事件消息的顺序无法保证。

接收事件消息的策略

要接收事件消息，您的设备必须使用适当的策略，允许其连接到 AWS IoT 设备网关并订阅 MQTT 事件主题。您还必须订阅合适的主题筛选条件。

以下是接收生命周期事件时需要的策略示例：

```
{
    "Version":"2012-10-17",
    "Statement":[{
        "Effect":"Allow",
        "Action":[
            "iot:Subscribe",
            "iot:Receive"
        ],
        "Resource":[
            "arn:aws:iot:region:account:/$aws/events/*"
        ]
    }]
}
```

启用以下项的事件 AWS IoT

在保留主题的订阅者可以接收消息之前，必须使用 API AWS Management Console 或 CLI 启用来自或的事件消息。有关不同选项管理的事件消息的信息，请参阅AWS IoT 事件配置设置表。

- 要启用事件消息，请转到 AWS IoT 控制台的设置选项卡，然后在基于事件的消息部分，选择管理事件。您可以指定要管理的事件。
- 要控制使用 API 或 CLI 发布哪些事件类型，请调用UpdateEvent配置 API 或使用 update-event-configurations CLI 命令。例如：

```
aws iot update-event-configurations --event-configurations "{\"THING\":{\"Enabled\":
 true}}"
```

> **Note**
>
> 使用反斜杠 (\) 转义所有双引号 (")。

您可以通过调用DescribeEvent配置 API 或使用 describe-event-configurations CLI 命令来获取当前的事件配置。例如：。

```
aws iot describe-event-configurations
```

AWS IoT 事件配置设置表

事件类别 （AWS IoT 控制台：设置：基于事件的消息）	**eventConfigurations** 键值 (AWS CLI/API)	事件消息主题
（只能使用 AWS CLI/API 进行配置）	`CA_CERTIFICATE`	`$aws/events/certificates/registered/`*`caCertificateId`*

事件类别 （AWS IoT 控制台：设置：基于事件的消息）	**eventConfigurations** 键值 (AWS CLI/API)	事件消息主题
（只能使用 AWS CLI/API 进行配置）	CERTIFICATE	$aws/events/presence/connected/ *clientId*
（只能使用 AWS CLI/API 进行配置）	CERTIFICATE	$aws/events/presence/disconnected/ *clientId*
（只能使用 AWS CLI/API 进行配置）	CERTIFICATE	$aws/events/subscriptions/subscribed/ *clientId*
（只能使用 AWS CLI/API 进行配置）	CERTIFICATE	$aws/events/subscriptions/unsubscribed/ *clientId*
任务已完成、已取消	JOB	$aws/events/job/*jobID*/canceled
任务已完成、已取消	JOB	$aws/events/job/*jobID*/cancellation_in_progress
任务已完成、已取消	JOB	$aws/events/job/*jobID*/completed
任务已完成、已取消	JOB	$aws/events/job/*jobID*/deleted
任务已完成、已取消	JOB	$aws/events/job/*jobID*/deletion_in_progress

事件类别 （AWS IoT 控制台：设置：基于事件的消息）	**eventConfigurations** 键值 (AWS CLI/API)	事件消息主题
任务执行：成功、失败、已被拒绝、已取消、已删除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/canceled
任务执行：成功、失败、已被拒绝、已取消、已删除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/deleted
任务执行：成功、失败、已被拒绝、已取消、已删除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/failed
任务执行：成功、失败、已被拒绝、已取消、已删除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/rejected
任务执行：成功、失败、已被拒绝、已取消、已删除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/removed
任务执行：成功、失败、已被拒绝、已取消、已删除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/succeeded
任务执行：成功、失败、已被拒绝、已取消、已删除	JOB_EXECUTION	$aws/events/jobExecution/ *jobID*/timed_out
事物：已创建、已更新、已删除	THING	$aws/events/thing/ *thingName* /created
事物：已创建、已更新、已删除	THING	$aws/events/thing/ *thingName* /updated
事物：已创建、已更新、已删除	THING	$aws/events/thing/ *thingName* /deleted

事件类别 （AWS IoT 控制台：设置：基于事件的消息）	**eventConfigurations** 键值 (AWS CLI/API)	事件消息主题
事物组：已添加、已删除	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /created
事物组：已添加、已删除	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /updated
事物组：已添加、已删除	THING_GROUP	$aws/events/thingGroup/ *thingGroupName* /deleted
事物组层次结构：已添加、已删除	THING_GROUP_HIERARCHY	$aws/events/thingGroupHierarchy/thingGroup/ *parentThingGroupName* /childThingGroup/ *childThingGroupName* /added
事物组层次结构：已添加、已删除	THING_GROUP_HIERARCHY	$aws/events/thingGroupHierarchy/thingGroup/ *parentThingGroupName* /childThingGroup/ *childThingGroupName* /removed
事物组成员资格：已添加、已删除	THING_GROUP_MEMBERSHIP	$aws/events/thingGroupMembership/thingGroup/ *thingGroupName* /thing/*thingName* /added

事件类别 （AWS IoT 控制台：设置：基于事件的消息）	**eventConfigurations** 键值 (AWS CLI/API)	事件消息主题
事物组成员资格：已添加、已删除	THING_GROUP_MEMBERSHIP	$aws/events/thingGroupMembership/thingGroup/ *thingGroupName* /thing/*thingName* /removed
事物类型：已创建、已更新、已删除	THING_TYPE	$aws/events/thingType/ *thingTypeName* /created
事物类型：已创建、已更新、已删除	THING_TYPE	$aws/events/thingType/ *thingTypeName* /updated
事物类型：已创建、已更新、已删除	THING_TYPE	$aws/events/thingType/ *thingTypeName* /deleted
事物类型关联：已添加、已删除	THING_TYPE_ASSOCIATION	$aws/events/thingTypeAssociation/thing/ *thingName* /thingType/ *thingTypeName* /added $aws/events/thingTypeAssociation/thing/ *thingName* /thingType/ *thingTypeName* /removed

注册表事件

在创建、更新或删除了事物、事物类型和事物组时，注册表将发布事件消息。但是，默认情况下，这些事件不可用。有关如何启用这些事件的更多信息，请参阅 启用以下项的事件 AWS IoT。

注册表可以提供以下事件类型：

- 事物事件
- 事物类型事件
- 事物组事件

事物事件

事物已创建/已更新/已删除

在创建、更新或删除了事物时，注册表将发布以下事件消息：

- $aws/events/thing/*thingName*/created
- $aws/events/thing/*thingName*/updated
- $aws/events/thing/*thingName*/deleted

消息包含以下示例负载：

```
{
    "eventType" : "THING_EVENT",
    "eventId" : "f5ae9b94-8b8e-4d8e-8c8f-b3266dd89853",
    "timestamp" : 1234567890123,
    "operation" : "CREATED|UPDATED|DELETED",
    "accountId" : "123456789012",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "thingName" : "MyThing",
    "versionNumber" : 1,
    "thingTypeName" : null,
    "attributes": {
                "attribute3": "value3",
                "attribute1": "value1",
                "attribute2": "value2"
    }
}
```

负载包含以下属性：

eventType

设置为“THING_EVENT”。

eventId

唯一事件 ID (字符串)。

时间戳

事件发生的 UNIX 时间戳。

operation

触发事件的操作。有效值为：

- CREATED
- 已更新
- DELETED

accountId

你的 AWS 账户 身份证。

thingId

要创建、更新或删除的事物的 ID。

thingName

要创建、更新或删除的事物的名称。

versionNumber

要创建、更新或删除的事物的版本。在创建事物时，此值设置为 1。每次更新事物时，此值增加 1。

东西 TypeName

与事物关联的事物类型 (如果存在)。否则为 `null`。

attributes

与事物关联的名称/值对的集合。

事物类型事件

事物类型相关事件：

- 事物类型已创建/已弃用/已取消弃用/已删除
- 事物类型已与某个事物关联或取消关联

事物类型已创建/已弃用/已取消弃用/已删除

在创建、弃用、取消弃用或删除了事物类型时，注册表将发布以下事件消息：

- $aws/events/thingType/*thingTypeName*/created
- $aws/events/thingType/*thingTypeName*/updated
- $aws/events/thingType/*thingTypeName*/deleted

消息包含以下示例负载：

```
{
    "eventType" : "THING_TYPE_EVENT",
    "eventId" : "8827376c-4b05-49a3-9b3b-733729df7ed5",
    "timestamp" : 1234567890123,
    "operation" : "CREATED|UPDATED|DELETED",
    "accountId" : "123456789012",
    "thingTypeId" : "c530ae83-32aa-4592-94d3-da29879d1aac",
    "thingTypeName" : "MyThingType",
    "isDeprecated" : false|true,
    "deprecationDate" : null,
    "searchableAttributes" : [ "attribute1", "attribute2", "attribute3" ],
    "description" : "My thing type"
}
```

负载包含以下属性：

eventType

设置为“THING_TYPE_EVENT”。

eventId

唯一事件 ID (字符串)。

时间戳

事件发生的 UNIX 时间戳。

operation

触发事件的操作。有效值为：

- CREATED
- 已更新
- DELETED

accountId

你的 AWS 账户 身份证。

东西 TypeId

要创建、弃用或删除的事物类型的 ID。

东西 TypeName

要创建、弃用或删除的事物类型的名称。

isDeprecated

如果事物类型已弃用，则为 `true`。否则为 `false`。

deprecationDate

弃用事物类型的 UNIX 时间戳。

searchableAttributes

与可用于搜索的事物类型关联的名称/值对集合。

description

事物类型的描述。

事物类型已与某个事物关联或取消关联

当事物类型与某个事物关联或取消关联时，注册表将发布以下事件消息。

- `$aws/events/thingTypeAssociation/thing/`*`thingName`*`/thingType/`*`typeName`*`/added`

- $aws/events/thingTypeAssociation/thing/*thingName*/thingType/*typeName*/removed

以下是 added 负载的示例。removed 消息的有效负载类似。

```
{
    "eventId" : "87f8e095-531c-47b3-aab5-5171364d138d",
    "eventType" : "THING_TYPE_ASSOCIATION_EVENT",
    "operation" : "ADDED",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "thingName": "myThing",
    "thingTypeName" : "MyThingType",
    "timestamp" : 1234567890123,
}
```

负载包含以下属性：

eventId

唯一事件 ID (字符串)。

eventType

设置为“THING_TYPE_ASSOCIATION_EVENT”。

operation

触发事件的操作。有效值为：

- 已添加
- REMOVED

thingId

已更改其类型关联的事物的 ID。

thingName

已更改其类型关联的事物的名称。

东西 TypeName

已与事物关联或不再与事物关联的事物类型。

时间戳

事件发生的 UNIX 时间戳。

事物组事件

事物组相关事件：

- 事物组已创建/已更新/已删除
- 事物已添加到事物组或从事物组中删除
- 事物组已添加到事物组或从事物组中删除

事物组已创建/已更新/已删除

在创建、更新或删除了事物组时，注册表将发布以下事件消息。

- $aws/events/thingGroup/*groupName*/created
- $aws/events/thingGroup/*groupName*/updated
- $aws/events/thingGroup/*groupName*/deleted

以下是 updated 负载的示例。created 和 deleted 消息的负载类似。

```
{
  "eventType": "THING_GROUP_EVENT",
  "eventId": "8b9ea8626aeaa1e42100f3f32b975899",
  "timestamp": 1603995417409,
  "operation": "UPDATED",
  "accountId": "571EXAMPLE833",
  "thingGroupId": "8757eec8-bb37-4cca-a6fa-403b003d139f",
  "thingGroupName": "Tg_level5",
  "versionNumber": 3,
  "parentGroupName": "Tg_level4",
  "parentGroupId": "5fce366a-7875-4c0e-870b-79d8d1dce119",
  "description": "New description for Tg_level5",
  "rootToParentThingGroups": [
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/TgTopLevel",
      "groupId": "36aa0482-f80d-4e13-9bff-1c0a75c055f6"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level1",
      "groupId": "bc1643e1-5a85-4eac-b45a-92509cbe2a77"
    },
    {
```

```
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level2",
      "groupId": "0476f3d2-9beb-48bb-ae2c-ea8bd6458158"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level3",
      "groupId": "1d9d4ffe-a6b0-48d6-9de6-2e54d1eae78f"
    },
    {
      "groupArn": "arn:aws:iot:us-west-2:571EXAMPLE833:thinggroup/Tg_level4",
      "groupId": "5fce366a-7875-4c0e-870b-79d8d1dce119"
    }
  ],
  "attributes": {
    "attribute1": "value1",
    "attribute3": "value3",
    "attribute2": "value2"
  },
  "dynamicGroupMappingId": null
}
```

负载包含以下属性：

eventType

设置为“THING_GROUP_EVENT”。

eventId

唯一事件 ID (字符串)。

时间戳

事件发生的 UNIX 时间戳。

operation

触发事件的操作。有效值为：

- CREATED
- 已更新
- DELETED

accountId

你的 AWS 账户 身份证。

东西 GroupId

要创建、更新或删除的事物组的 ID。

东西 GroupName

要创建、更新或删除的事物组的名称。

versionNumber

事物组的版本。在创建事物组时，此值设置为 1。每次更新事物组时，此值增加 1。

父母 GroupName

父事物组的名称（如果存在）。

父母 GroupId

父事物组的 ID（如果存在）。

description

事物组的描述。

根 ToParent ThingGroups

有关父事物组的信息数组。每个父事物组都有一个条目，从根事物组开始，继续直至达到父事物组。每个条目均包含事物组的 `groupArn` 和 `groupId`。

attributes

与事物组关联的名称/值对的集合。

事物已添加到事物组或从事物组中删除

当事物已添加到事物组或从事物组中删除时，注册表将发布以下事件消息。

- `$aws/events/thingGroupMembership/thingGroup/`*`thingGroupName`*`/thing/`*`thingName`*`/added`
- `$aws/events/thingGroupMembership/thingGroup/`*`thingGroupName`*`/thing/`*`thingName`*`/removed`

消息包含以下示例负载：

```
{
    "eventType" : "THING_GROUP_MEMBERSHIP_EVENT",
```

```
    "eventId" : "d684bd5f-6f6e-48e1-950c-766ac7f02fd1",
    "timestamp" : 1234567890123,
    "operation" : "ADDED|REMOVED",
    "accountId" : "123456789012",
    "groupArn" : "arn:aws:iot:ap-northeast-2:123456789012:thinggroup/
MyChildThingGroup",
    "groupId" : "06838589-373f-4312-b1f2-53f2192291c4",
    "thingArn" : "arn:aws:iot:ap-northeast-2:123456789012:thing/MyThing",
    "thingId" : "b604f69c-aa9a-4d4a-829e-c480e958a0b5",
    "membershipId" : "8505ebf8-4d32-4286-80e9-c23a4a16bbd8"
}
```

负载包含以下属性：

eventType

设置为“THING_GROUP_MEMBERSHIP_EVENT”。

eventId

事件 ID。

时间戳

事件发生的 UNIX 时间戳。

operation

当事物添加到事物组时为 ADDED。当事物从事物组中删除时为 REMOVED。

accountId

你的 AWS 账户 身份证。

groupArn

事物组的 ARN。

groupId

组的 ID。

thingArn

在事物组中添加或删除的事物的 ARN。

thingId

在事物组中添加或删除的事物的 ID。

membershipId

表示事物与事物组之间关系的 ID。将事物添加到事物组时生成此值。

事物组已添加到事物组或从事物组中删除

当某个事物组已添加到另一个事物组或从另一个事物组中删除时，注册表将发布以下事件消息。

- $aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName*/childThingGroup/*childThingGroupName*/added
- $aws/events/thingGroupHierarchy/thingGroup/*parentThingGroupName*/childThingGroup/*childThingGroupName*/removed

消息包含以下示例负载：

```
{
    "eventType" : "THING_GROUP_HIERARCHY_EVENT",
    "eventId" : "264192c7-b573-46ef-ab7b-489fcd47da41",
    "timestamp" : 1234567890123,
    "operation" : "ADDED|REMOVED",
    "accountId" : "123456789012",
    "thingGroupId" : "8f82a106-6b1d-4331-8984-a84db5f6f8cb",
    "thingGroupName" : "MyRootThingGroup",
    "childGroupId" : "06838589-373f-4312-b1f2-53f2192291c4",
    "childGroupName" : "MyChildThingGroup"
}
```

负载包含以下属性：

eventType

设置为“THING_GROUP_HIERARCHY_EVENT”。

eventId

事件 ID。

时间戳

事件发生的 UNIX 时间戳。

operation

当事物添加到事物组时为 ADDED。当事物从事物组中删除时为 REMOVED。

accountId

你的 AWS 账户 身份证。

东西 GroupId

父事物组的 ID。

东西 GroupName

父事物组的名称。

孩子 GroupId

子事物组的 ID。

孩子 GroupName

子事物组的名称。

任务事件

当 AWS IoT 任务处于待处理状态、已完成或取消状态时，以及设备在运行作业时报告成功或失败时，作业服务将发布到 MQTT 协议上的保留主题。设备或管理和监控应用程序可以通过订阅这些主题来跟踪任务的状态。

如何启用任务事件

来自 AWS IoT 任务服务的响应消息不会通过消息代理，也无法被其他客户端或规则订阅。要订阅与任务活动相关的消息，请使用 `notify` 和 `notify-next` 主题。有关任务主题的更多信息，请参阅 任务主题。

要收到任务更新的通知，请使用或使用 API 或 CLI 启用这些作业事件。 AWS Management Console有关更多信息，请参阅 启用以下项的事件 AWS IoT。

任务事件的工作原理

由于取消或删除任务可能需要一些时间，所以发送两条消息来指示请求的开始和结束。例如，当取消请求开始时，系统会向 `$aws/events/job/jobID/cancellation_in_progress` 主题发送一条消息。当取消请求完成时，系统会向 `$aws/events/job/jobID/canceled` 主题发送一条消息。

任务删除请求的过程与之相似。管理和监控应用程序可以订阅这些主题来跟踪任务的状态。有关发布和订阅 MQTT 主题的更多信息，请参阅 the section called “设备通信协议”。

任务事件类型

不同的任务事件类型如下：

任务已完成/已取消/已删除

当任务完成、取消、删除或正在取消或删除时， AWS IoT 作业服务会在 MQTT 主题上发布一条消息：

- $aws/events/job/*jobID*/completed
- $aws/events/job/*jobID*/canceled
- $aws/events/job/*jobID*/deleted
- $aws/events/job/*jobID*/cancellation_in_progress
- $aws/events/job/*jobID*/deletion_in_progress

completed 消息包含以下示例负载：

```
{
  "eventType": "JOB",
  "eventId": "7364ffd1-8b65-4824-85d5-6c14686c97c6",
  "timestamp": 1234567890,
  "operation": "completed",
  "jobId": "27450507-bf6f-4012-92af-bb8a1c8c4484",
  "status": "COMPLETED",
  "targetSelection": "SNAPSHOT|CONTINUOUS",
  "targets": [
    "arn:aws:iot:us-east-1:123456789012:thing/a39f6f91-70cf-4bd2-a381-9c66df1a80d0",
    "arn:aws:iot:us-east-1:123456789012:thinggroup/2fc4c0a4-6e45-4525-
a238-0fe8d3dd21bb"
  ],
  "description": "My Job Description",
  "completedAt": 1234567890123,
  "createdAt": 1234567890123,
  "lastUpdatedAt": 1234567890123,
  "jobProcessDetails": {
    "numberOfCanceledThings": 0,
    "numberOfRejectedThings": 0,
    "numberOfFailedThings": 0,
    "numberOfRemovedThings": 0,
    "numberOfSucceededThings": 3
```

```
  }
}
```

canceled 消息包含以下示例负载。

```
{
  "eventType": "JOB",
  "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
  "timestamp": 1234567890,
  "operation": "canceled",
  "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
  "status": "CANCELED",
  "targetSelection": "SNAPSHOT|CONTINUOUS",
  "targets": [
    "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
    "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
  ],
  "description": "My job description",
  "createdAt": 1234567890123,
  "lastUpdatedAt": 1234567890123
}
```

deleted 消息包含以下示例负载。

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "deleted",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "DELETED",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
```

```
      "comment": "Comment for this operation"
    }
```

cancellation_in_progress 消息包含以下示例负载：

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "cancellation_in_progress",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "CANCELLATION_IN_PROGRESS",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
      "lastUpdatedAt": 1234567890123,
      "comment": "Comment for this operation"
    }
```

deletion_in_progress 消息包含以下示例负载：

```
{
      "eventType": "JOB",
      "eventId": "568d2ade-2e9c-46e6-a115-18afa1286b06",
      "timestamp": 1234567890,
      "operation": "deletion_in_progress",
      "jobId": "4d2a531a-da2e-47bb-8b9e-ff5adcd53ef0",
      "status": "DELETION_IN_PROGRESS",
      "targetSelection": "SNAPSHOT|CONTINUOUS",
      "targets": [
        "arn:aws:iot:us-east-1:123456789012:thing/Thing0-947b9c0c-ff10-4a80-b4b3-
cd33d0145a0f",
        "arn:aws:iot:us-east-1:123456789012:thinggroup/
ThingGroup1-95c644d5-1621-41a6-9aa5-ad2de581d18f"
      ],
      "description": "My job description",
      "createdAt": 1234567890123,
```

```
    "lastUpdatedAt": 1234567890123,
    "comment": "Comment for this operation"
  }
```

任务执行最终状态

当设备将任务执行更新为终端状态时，AWS IoT 作业服务会发布一条消息：

- $aws/events/jobExecution/*jobID*/succeeded
- $aws/events/jobExecution/*jobID*/failed
- $aws/events/jobExecution/*jobID*/rejected
- $aws/events/jobExecution/*jobID*/canceled
- $aws/events/jobExecution/*jobID*/timed_out
- $aws/events/jobExecution/*jobID*/removed
- $aws/events/jobExecution/*jobID*/deleted

消息包含以下示例负载：

```
{
  "eventType": "JOB_EXECUTION",
  "eventId": "cca89fa5-8a7f-4ced-8c20-5e653afb3572",
  "timestamp": 1234567890,
  "operation": "succeeded|failed|rejected|canceled|removed|timed_out",
  "jobId": "154b39e5-60b0-48a4-9b73-f6f8dd032d27",
  "thingArn": "arn:aws:iot:us-east-1:123456789012:myThing/6d639fbc-8f85-4a90-924d-
a2867f8366a7",
  "status": "SUCCEEDED|FAILED|REJECTED|CANCELED|REMOVED|TIMED_OUT",
  "statusDetails": {
    "key": "value"
  }
}
```

生命周期事件

AWS IoT 可以发布有关 MQTT 主题的生命周期事件。这些事件在默认情况下可用，且不能被禁用。

> **Note**
>
> 生命周期消息可能不会按顺序发送。您可能会收到重复的消息。

本主题内容：

连接/断开连接事件

> **Note**
>
> 借助 AWS IoT 设备管理队列索引，您可以搜索事物、运行聚合查询，并根据事物 “连接/断开连接” 事件创建动态组。有关更多信息，请参阅实例集索引。

AWS IoT 当客户端连接或断开连接时，向以下 MQTT 主题发布消息：

- `$aws/events/presence/connected/clientId` — 连接到消息代理的客户端。
- `$aws/events/presence/disconnected/clientId` — 与消息代理断开连接的客户端。

下面是一系列 JSON 元素，发布到 `$aws/events/presence/connected/clientId` 主题的连接/断开连接消息中包含这些元素。

clientId

建立连接或断开连接的客户端的 ID。

> **Note**
>
> 包含 # 或 + 的客户端 ID 不接收生命周期事件。

客户 InitiatedDisconnect

如果客户端启动断开连接，则为“True”。否则为 false。只能在断开连接消息中找到。

disconnectReason

客户端断开连接的原因。只能在断开连接消息中找到。下表包含有效值，以及在断开连接时代理是否会发送 Last Will and Testament（LWT）消息。

断开连接原因	描述	代理将发送 LWT 消息
AUTH_ERROR	客户端无法进行身份验证或授权失败。	是。如果设备在收到此错误之前连接处于活动状态。
CLIENT_INITIATED_DISCONNECT	客户端指示它将断开连接。客户端可以通过发送 MQTT DISCONNECT 控制数据包或在客户端使用 WebSocket 连接Close frame时发送 MQTT 控制数据包来实现此目的。	不是。
CLIENT_ERROR	客户端出错，导致它断开连接。例如，如果客户端在同一连接上发送多个 MQTT CONNECT 数据包，或者客户端尝试使用超过负载限制的负载进行发布，则将断开连接。	是。
CONNECTION_LOST	客户端-服务器连接被切断。在高网络延迟期间或互联网连接断开时，可能会发生此情况。	是。
DUPLICATE_CLIENTID	客户端使用的是已在使用的客户端 ID。在这种情况下，已连接的客户端将因这一断开连接原因而断开连接。	是。
FORBIDDEN_ACCESS	不允许连接客户端。例如，IP 地址被拒绝的客户端将无法连接。	是。如果设备在收到此错误之前连接处于活动状态。
MQTT_KEEP_ALIVE_TIMEOUT	如果在 1.5 倍的客户端保持连接时间内没有客户端-服务器通信，则客户端将断开连接。	是。
SERVER_ERROR	由于意外的服务器问题而断开连接。	是。
SERVER_INITIATED_DISCONNECT	服务器出于操作原因而故意断开与客户端的连接。	是。

断开连接原因	描述	代理将发送 LWT 消息
THROTTLED	客户端由于超出限制而断开连接。	是。
WEBSOCKET_TTL_EXPIRATION	客户端断开连接是因 WebSocket 为 a 的连接时间超过其 time-to-live值。	是。
CUSTOMAUTH_TTL_EXPIRATION	客户端之所以断开连接，是因为它的连接时间超过了其自定义授权者的 time-to-live 值。	是。

eventType

事件类型。有效值为 `connected` 或 `disconnected`。

ipAddress

连接客户端的 IP 地址。该地址可以采用 IPv4 或 IPv6 格式。只能在连接消息中找到。

principalIdentifier

用于执行身份验证的凭证。对于 TLS 双向身份验证证书，这是证书 ID。对于其它连接，这是 IAM 凭证。

sessionIdentifier

在会话生命周期中 AWS IoT 存在的全局唯一标识符。

时间戳

事件发生时间的近似值。

versionNumber

生命周期事件的版本号。对于每个客户端 ID 连接，这是一个单调递增的长整数值。订阅者可以使用版本号来推断生命周期事件的顺序。

> **Note**
>
> 客户端连接的连接和断开消息具有相同的版本号。
> 版本号可能会跳过值，而不保证对于每个事件始终增加 1。

如果客户端约一小时未连接，则版本号将重置为 0。对于持续会话，当客户端断开连接的时间超过为持续会话配置的 time-to-live (TTL) 时间后，版本号将重置为 0。

连接消息具有以下结构。

```
{
    "clientId": "186b5",
    "timestamp": 1573002230757,
    "eventType": "connected",
    "sessionIdentifier": "a4666d2a7d844ae4ac5d7b38c9cb7967",
    "principalIdentifier": "12345678901234567890123456789012",
    "ipAddress": "192.0.2.0",
    "versionNumber": 0
}
```

断开连接消息具有以下结构。

```
{
    "clientId": "186b5",
    "timestamp": 1573002340451,
    "eventType": "disconnected",
    "sessionIdentifier": "a4666d2a7d844ae4ac5d7b38c9cb7967",
    "principalIdentifier": "12345678901234567890123456789012",
    "clientInitiatedDisconnect": true,
    "disconnectReason": "CLIENT_INITIATED_DISCONNECT",
    "versionNumber": 0
}
```

处理客户端断开连接

最佳实践是始终为生命周期事件实现等待状态，包括 Last Will and Testament（LWT）消息。收到断开连接消息后，您的代码应等待一段时间，并在采取措施之前验证设备是否仍处于脱机状态。实现此目标的一种方法是使用 SQS 延迟队列。当客户端收到 LWT 或生命周期事件时，您可以将消息列入队列进行排队（例如 5 秒）。当消息可用并被处理（通过 Lambda 或其它服务）时，您可以先检查设备是否仍处于离线状态，然后再采取进一步操作。

订阅/取消订阅事件

AWS IoT 当客户端订阅或取消订阅 MQTT 主题时，向以下 MQTT 主题发布消息：

```
$aws/events/subscriptions/subscribed/clientId
```

或者

```
$aws/events/subscriptions/unsubscribed/clientId
```

其中 clientId 是连接到 AWS IoT 消息代理的 MQTT 客户端 ID。

发布到该主题的消息具有以下结构：

```
{
    "clientId": "186b5",
    "timestamp": 1460065214626,
    "eventType": "subscribed" | "unsubscribed",
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",
    "principalIdentifier": "000000000000/ABCDEFGHIJKLMNOPQRSTU:some-user/
ABCDEFGHIJKLMNOPQRSTU:some-user",
    "topics" : ["foo/bar","device/data","dog/cat"]
}
```

下面是一系列 JSON 元素，发布到 $aws/events/subscriptions/subscribed/*clientId* 和 $aws/events/subscriptions/unsubscribed/*clientId* 主题的已订阅/未订阅消息中包含这些元素。

clientId

订阅或取消订阅的客户端的 ID。

> **Note**
>
> 包含 # 或 + 的客户端 ID 不接收生命周期事件。

eventType

事件类型。有效值为 subscribed 或 unsubscribed。

principalIdentifier

用于执行身份验证的凭证。对于 TLS 双向身份验证证书，这是证书 ID。对于其它连接，这是 IAM 凭证。

sessionIdentifier

在会话生命周期中 AWS IoT 存在的全局唯一标识符。

时间戳

事件发生时间的近似值。

topics

客户端已订阅的一系列 MQTT 主题。

> **Note**
>
> 生命周期消息可能不会按顺序发送。您可能会收到重复的消息。

故障排除 AWS IoT

帮助我们改进此主题

告诉我们如何优化内容

以下信息可帮助您处理 AWS IoT中的常见问题。

任务

- AWS IoT Core 疑难解答指南
- AWS IoT 设备顾问疑难解答指南
- AWS IoT Device Management 疑难解答指南
- AWS IoT 错误

AWS IoT Core 疑难解答指南

帮助我们改进此主题

告诉我们如何优化内容

这是的疑难解答部分 AWS IoT Core。

主题

- 诊断连接问题
- 诊断规则问题
- 诊断影子问题
- 诊断 Salesforce IoT 输入流操作问题
- 诊断流限制
- 排除设备机群断开连接的故障

诊断连接问题

帮助我们改进此主题

告诉我们如何优化内容

要成功连接到，AWS IoT 需要：

- 有效的连接
- 有效且已激活的证书
- 允许所需连接和操作的策略

Connection

如何找到正确的终端节点？

- `aws iot describe-endpoint --endpoint-type iot:Data-ATS` 返回的 `endpointAddress`

  或者

- `aws iot describe-domain-configuration --domain-configuration-name "domain_configuration_name"` 返回的 `domainName`

如何找到正确的 Server Name Indication (SNI) 值？

正确的 SNI 值是由 describe-endpoint 返回的 `endpointAddress` 或由 describe-domain-configuration 命令返回的 `domainName`。它是与上一步中的终端节点相同的地址。将设备连接到时 AWS IoT Core，客户端可以发送服务器名称指示 (SNI) 扩展名，这不是必需的，但强烈建议这样做。要使用多账户注册、自定义域和 VPC 端点等特征，必须使用 SNI 扩展。有关更多信息，请参阅中的传输安全 AWS IoT。

我该如何解决持续存在的连接问题？

您可以使用 AWS 设备顾问来帮助排除故障。Device Advisor 的预构建测试可帮助您根据最佳实践验证您的设备软件，以便使用 TLS、MQTT、AWS IoT 设备影子和 AWS IoT Jobs。

以下是现有 Device Advisor 内容的链接。

身份验证

设备必须经过身份验证才能连接到 AWS IoT 端点。对于用于身份验证X.509 客户端证书的设备，证书必须注册 AWS IoT 并处于活动状态。

我的设备如何对 AWS IoT 端点进行身份验证？

将 AWS IoT CA 证书添加到客户的信任存储中。请参阅 AWS IoT Core中的服务器身份验证上的文档，然后通过链接下载适当的 CA 证书。

设备连接时会检查什么 AWS IoT？

当设备尝试连接到 AWS IoT时：

1. AWS IoT 检查证书和服务器名称指示 (SNI) 值是否有效。
2. AWS IoT 检查使用的证书是否已在 AWS IoT 账户中注册并已激活。
3. 当设备尝试在中 AWS IoT执行任何操作（例如订阅或发布消息）时，系统会检查其用于连接的证书所附的策略，以确认该设备已获得执行该操作的授权。

如何验证证书的配置是否正确？

请使用 OpenSSL `s_client` 命令测试与 AWS IoT 终端节点的连接：

```
openssl s_client -connect custom_endpoint.iot.aws-region.amazonaws.com:8443 -
CAfile CA.pem -cert cert.pem -key privateKey.pem
```

有关如何使用 `openssl s_client` 的更多信息，请参阅 OpenSSL s_client 文档。

如何检查证书的状态？

- 列出证书

  如果您不知道该证书 ID，则可以使用 `aws iot list-certificates` 命令查看所有证书的状态。

- 显示证书的详细信息

  如果您知道证书的 ID，此命令将显示有关证书的更详细信息。

  ```
  aws iot describe-certificate --certificate-id "certificateId"
  ```

- 在 AWS IoT 控制台中查看证书

  在 AWS IoT 控制台，在左侧菜单中，选择 Secure（安全），然后选择 Certificates（证书）。

从列表中选择用于连接的证书以打开其详细信息页面。

在证书的详细信息页面中，您可以查看其当前状态。

证书的状态可以通过使用详细信息页面右上角的 Actions（操作）菜单来更改。

授权

AWS IoT 资源AWS IoT Core 政策用于授权这些资源执行操作。要对某项操作进行授权，指定的 AWS IoT 资源必须附有一份策略文档，以授予执行该操作的权限。

我收到了代理发送的 PUBNACK 或 SUBNACK 回复。我应该怎么办？

请确保在您用来调用的证书上附有策略 AWS IoT。默认情况下，所有的发布/订阅操作均将被拒绝。

确保附加的策略授权您尝试执行的操作。

确保附加的策略授权正在尝试执行授权操作的资源。

我的日志中有一个 AUTHORIZATION_FAILURE 的条目。

请确保在您用来调用的证书上附有策略 AWS IoT。默认情况下，所有的发布/订阅操作均将被拒绝。

确保附加的策略授权您尝试执行的操作。

确保附加的策略授权正在尝试执行授权操作的资源。

如何检查策略授权的内容？

在 AWS IoT 控制台，在左侧菜单中，选择 Secure（安全），然后选择 Certificates（证书）。

从列表中选择用于连接的证书以打开其详细信息页面。

在证书的详细信息页面中，您可以查看其当前状态。

在证书详细信息页面的左侧菜单中，选择 Policies（策略）查看附加到该证书的策略。

选择所需策略以查看其详细信息页面。

在策略的详细信息页面中，查看策略的策略文档以查看它授权的内容。

选择 Edit policy document（编辑策略文档）对策略文档进行更改。

安全和身份

当您为 AWS IoT 自定义域配置提供服务器证书时，这些证书最多有四个域名。

有关更多信息，请参阅 AWS IoT Core 终端节点和限额。

诊断规则问题

帮助我们改进此主题

告诉我们如何优化内容

本部分介绍了在遇到规则问题时要检查的一些事项。

配置用于故障排除的 CloudWatch 日志

调试规则问题的最佳方法是使用 CloudWatch 日志。启用 CloudWatch 日志后 AWS IoT，您可以看到哪些规则被触发以及它们的成功或失败。您还可以了解 WHERE 子句条件是否匹配。有关更多信息，请参阅 AWS IoT 使用 CloudWatch 日志进行监控。

最常见的规则问题是授权问题。日志会显示您的角色是否无权在资源 AssumeRole 上执行。下面是一个精细日志记录功能生成的示例日志：

```
{
    "timestamp": "2017-12-09 22:49:17.954",
    "logLevel": "ERROR",
    "traceId": "ff563525-6469-506a-e141-78d40375fc4e",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "RuleExecution",
    "clientId": "iotconsole-123456789012-3",
    "topicName": "test-topic",
    "ruleName": "rule1",
    "ruleAction": "DynamoAction",
    "resources": {
        "ItemHashKeyField": "id",
        "Table": "trashbin",
        "Operation": "Insert",
        "ItemHashKeyValue": "id",
        "IsPayloadJSON": "true"
    },
```

```
    "principalId": "ABCDEFG1234567ABCD890:outis",
    "details": "User: arn:aws:sts::123456789012:assumed-role/dynamo-
testbin/5aUMInJH is not authorized to perform: dynamodb:PutItem on
 resource: arn:aws:dynamodb:us-east-1:123456789012:table/testbin (Service:
 AmazonDynamoDBv2; Status Code: 400; Error Code: AccessDeniedException; Request ID:
 AKQJ987654321AKQJ123456789AKQJ987654321AKQJ987654321)"
}
```

下面是一个全局日志记录功能生成的类似示例日志：

```
2017-12-09 22:49:17.954 TRACEID:ff562535-6964-506a-e141-78d40375fc4e
PRINCIPALID:ABCDEFG1234567ABCD890:outis [ERROR] EVENT:DynamoActionFailure
TOPICNAME:test-topic CLIENTID:iotconsole-123456789012-3
MESSAGE:Dynamo Insert record failed. The error received was User:
 arn:aws:sts::123456789012:assumed-role/dynamo-testbin/5aUMInJI is not authorized to
 perform: dynamodb:PutItem on resource: arn:aws:dynamodb:us-east-1:123456789012:table/
testbin
(Service: AmazonDynamoDBv2; Status Code: 400; Error Code: AccessDeniedException;
 Request ID: AKQJ987654321AKQJ987654321AKQJ987654321AKQJ987654321).
Message arrived on: test-topic, Action: dynamo, Table: trashbin, HashKeyField: id,
 HashKeyValue: id, RangeKeyField: None, RangeKeyValue: 123456789012
No newer events found at the moment. Retry.
```

有关更多信息，请参阅 the section called “在 CloudWatch 控制台中查看 AWS IoT 日志”。

诊断外部服务

外部服务由最终用户控制。在执行规则之前，请确保已设置链接到规则的外部服务，并且具有足够的应用程序吞吐量和容量单位。

诊断 SQL 问题

如果您的 SQL 查询没有返回您期望的数据：

- 查看日志中的错误消息。
- 确认您的 SQL 语法与消息中的 JSON 文档匹配。

  查看查询中使用的对象和属性名称，以及主题消息负载的 JSON 文档中使用的对象名和属性名称。有关 SQL 查询中 JSON 格式的更多信息，请参阅 JSON 扩展。
- 检查 JSON 对象或属性名称是否包含预留字符或数字字符。

有关 SQL 查询中 JSON 对象引用中预留字符的更多信息，请参阅 JSON 扩展。

诊断影子问题

帮助我们改进此主题

告诉我们如何优化内容

诊断影子

问题	故障排除指南
设备的影子文档被拒绝，并显示 `Invalid JSON document`。	如果您不熟悉 JSON，请修改本指南中提供的示例，以供您自己使用。有关更多信息，请参阅影子文档示例。
我提交了正确的 JSON，但设备的影子文档中没有存储其中的任何内容或仅存储了一部分。	请确保您遵循了以下 JSON 格式指南。仅存储 `desired` 和 `reported` 部分中的 JSON 字段。这些部分以外的 JSON 内容 (即使格式正确) 将被忽略。
我收到一条错误消息，称设备的影子超出了允许的大小。	设备的影子仅支持 8KB 数据。请尝试缩短您的 JSON 文档中的字段名称，也可以通过创建更多事物来创建更多影子。一个设备可拥有无限数量的事物/与之关联的影子。唯一的要求是，您账户中的每个事物名称都必须是唯一的。
我收到了一个超过 8KB 的设备的影子。这是怎么回事？	收到后，该 AWS IoT 服务会将元数据添加到设备的影子中。该服务将此类数据添加到其响应中，但此类数据不会计入 8KB 的限制中。只有发送到设备的影子的状态文档中有关 `desired` 状态和 `reported` 状态的数据才会计入到此限制中。
我的请求由于版本错误而被拒绝了。我应该怎么办？	执行 GET 操作，以同步至最新的状态文档版本。使用 MQTT 时，请订阅 ./update/accepted

问题	故障排除指南
	主题，这样您便会收到有关状态更改的通知以及最新版本的 JSON 文档。
时间戳关闭了几秒钟。	当 AWS IoT 服务收到文档或状态文档发布到上时，会更新各个字段和整个 JSON 文档的时间戳。 /update/已接受和。 /update/delta 消息。消息可能会由于网络发生延迟，从而导致时间戳关闭几秒钟。
我的设备可以在相应的影子主题下发布消息并订阅这些主题，但当我尝试通过 HTTP REST API 更新影子文档时，我收到了 HTTP 403 错误消息。	请确保您已在 IAM 中创建了相应的策略，能够访问这些主题并针对您使用的凭证执行相应的操作 (UPDATE/GET/DELETE)。IAM policy 和证书策略是相互独立的。
其它问题。	Device Shadow 服务将错误 CloudWatch 记录到日志中。要识别设备和配置问题，请启用 CloudWatch 日志并查看日志以获取调试信息。

诊断 Salesforce IoT 输入流操作问题

帮助我们改进此主题

告诉我们如何优化内容

执行跟踪

如何查看 Salesforce 操作的执行跟踪？

请参阅 AWS IoT 使用 CloudWatch 日志进行监控 部分。激活日志后，您可以查看 Salesforce 操作的执行跟踪。

操作成功和失败

如何检查是否已成功将消息发送到 Salesforce IoT 输入流？

在 “日志” 中查看执行 Salesforce 操作所生成的 CloudWatch 日志。如果你看到`Action executed successfully`，则表示 AWS IoT 规则引擎收到了来自 Salesforce IoT 的确认，即消息已成功推送到目标输入流。

如果您在使用 Salesforce IoT 平台时遇到问题，请联系 Salesforce IoT 支持。

如果消息未成功发送到 Salesforce IoT 输入流，该怎么办？

在 “日志” 中查看执行 Salesforce 操作所生成的 CloudWatch 日志。您可以尝试以下操作，具体取决于日志条目：

`Failed to locate the host`

检查此操作的 `url` 参数是否正确，以及您的 Salesforce IoT 输入流是否存在。

`Received Internal Server Error from Salesforce`

重试。如果问题仍然存在，请联系 Salesforce IoT 支持。

`Received Bad Request Exception from Salesforce`

检查您正在发送的负载是否存在错误。

`Received Unsupported Media Type Exception from Salesforce`

Salesforce IoT 目前不支持二进制负载。检查您是否正在发送 JSON 负载。

`Received Unauthorized Exception from Salesforce`

检查此操作的 `token` 参数是否正确，以及您的令牌是否仍有效。

`Received Not Found Exception from Salesforce`

检查此操作的 `url` 参数是否正确，以及您的 Salesforce IoT 输入流是否存在。

如果您收到的错误未在此处列出，请联系 Su AWS IoT pport。

诊断流限制

“您的 AWS 账户已超过流限制”问题排查

如果您看到 "Error: You have exceeded the limit for the number of streams in your AWS account."，您可以清除账户中未使用的流，而不是请求提高限制。

要清理您使用 AWS CLI 或 SDK 创建的未使用的直播，请执行以下操作：

```
aws iot delete-stream –stream-id value
```

有关更多详细信息，请参阅 delete-stream。

> **Note**
>
> 您可以使用 list-streams 命令查找流 ID。

排除设备机群断开连接的故障

> **帮助我们改进此主题**
>
> 告诉我们如何优化内容

AWS IoT 设备群断开连接可能有多种原因。本文介绍如何诊断断开连接的原因，以及如何处理因定期维护 AWS IoT 服务或限制而导致的断开连接。

诊断断开连接的原因

您可以查看 AWSIotLogsV2 日志组 CloudWatch，在日志条目的disconnectReason字段中确定断开连接的原因。

您还可以使用 AWS IoT生命周期事件功能来确定断开连接的原因。如果您订阅了生命周期的断开连接事件 ($aws/events/presence/disconnected/*clientId*)，则会在断开连接发生 AWS IoT 时收到通知。您可以在通知的 disconnectReason 字段找到断开连接的原因。

有关更多信息，请参阅CloudWatch AWS IoT 日志条目和生命周期事件。

排除因 AWS IoT 服务维护而导致的断开连接故障

由 AWS IoT的服务维护导致的断开连接将记录为SERVER_INITIATED_DISCONNECT生命周期事件 AWS IoT，并且 CloudWatch. 要处理这些断开连接，请调整您的客户端设置，以确保您的设备可以自动重新连接到平台。 AWS IoT

排除由于节流限制而导致的断开连接故障

限制限制导致的断开连接将作为生命周期事件进行记录THROTTLED，并 AWS IoT且. CloudWatch 要处理这些断开连接的问题，您可以在设备计数增长时请求消息代理限制增加。

有关更多信息，请参阅 AWS IoT 核心消息代理。

AWS IoT 设备顾问疑难解答指南

帮助我们改进此主题

告诉我们如何优化内容

常规

问：我是否可以并行运行多个测试套件？

答：能。Device Advisor 现在支持使用设备级终端节点在不同的设备上运行多个测试套件。如果使用账户级终端节点，则可以一次运行一个套件，因为每个账户有一个 Device Advisor 终端节点。有关更多信息，请参阅配置您的设备。

问：我从设备看到 TLS 连接被 Device Advisor 拒绝。这样是对的吗？

答：能。Device Advisor 在每次测试运行之前和之后都会拒绝 TLS 连接。我们建议用户实施设备重试机制，以便使用 Device Advisor 获得全自动的测试体验。如果您执行具有多个测试用例的测试套件（例如 TLS 连接、MQTT 连接和 MQTT 发布），我们建议您为设备构建一个机制。该机制可以每 5 秒便尝试一次连接到测试终端节点，每次持续一到两分钟。通过这种方式，您能够以自动的方式按顺序运行多个测试用例。

问：我能否获得 Device Advisor 在我的账户上发起的所有 API 调用的历史记录，以便用于安全分析和运营方面的故障排除？

答：可以。要接收使用您的账户进行的 Device Advisor API 调用的历史记录，您只需 CloudTrail 在 AWS IoT 管理控制台中打开并筛选事件源即可`iotdeviceadvisor.amazonaws.com`。

问：如何查看 Device Advisor 登录信息 CloudWatch？

答： CloudWatch 如果您向服务角色添加所需的策略（例如 CloudWatchFullAccess），则在测试套件运行期间生成的日志将上传到（请参阅设置）。如果测试套件中至少有一个测试用例，则会创建一个包含两个日志流的日志组 “aws/iot/deviceadvisor/ testSuiteId $”。一个流是 “$testRunId”，它包含在测试套件中执行测试用例之前和之后所采取的操作的日志，例如设置和清理步骤。另一个日

志流是 “$ suiteRunId _$”testRunId，它特定于测试套件的运行。从设备发送的事件 AWS IoT Core 将记录到此日志流中。

问：设备权限角色的目的是什么？

答：Device Advisor 介 AWS IoT Core 于您的测试设备和模拟测试场景之间。它接受来自测试设备的连接和消息，并通过承担您的设备权限角色并代表您启动连接来将它们转发到 AWS IoT Core 。请务必确保设备角色权限与用于运行测试的证书上的权限相同。 AWS IoT 当 Device Advisor 使用设备权限角色 AWS IoT Core 代表您启动与的连接时，不会强制执行证书策略。但是，您设置的设备权限角色中的权限将强制执行。

问：Device Advisor 在哪些区域受支持？

答：us-east-1、us-west-2、ap-northeast-1 和 eu-west-1 区域支持 Device Advisor。

问：为什么我看到不一致的结果？

答：结果不一致的主要原因之一是为测试的 `EXECUTION_TIMEOUT` 设置了过低的值。有关推荐和默认 `EXECUTION_TIMEOUT` 值的更多信息，请参阅 Device Advisor 测试案例。

问：Device Advisor 支持什么 MQTT 协议？

答：Device Advisor 支持使用 X509 客户端证书的 MQTT 3.1.1 版。

问：即使我尝试将设备连接到测试终端节点，如果我的测试用例失败并显示执行超时消息，该怎么办？

答：验证创建要用作设备角色的 IAM 角色下的所有步骤。如果测试仍然失败，则可能是设备没有发送正确的服务器名称指示 (SNI) 扩展，这是 Device Advisor 工作所必需的。正确的 SNI 值是按照 “配置您的设备” 部分操作时返回的端点地址。 AWS IoT 还要求设备向传输层安全 (TLS) 协议发送服务器名称指示 (SNI) 扩展。有关更多信息，请参阅中的传输安全 AWS IoT。

问：我的 MQTT 连接失败并出现 “libaws-c-mqtt: AWS_ERROR_MQTT_EXPERTD_HANGUP” 错误（或者）我的设备的 MQTT 连接已自动与设备顾问终端节点断开连接。如何解决此错误？

答：此特定错误代码和意外断开连接可能是由许多不同的原因造成的，但很可能与附加到设备的设备角色相关。下面的检查点（按优先级顺序）将解决此问题。

- 附加到设备的设备角色必须具有运行测试所需的最低 IAM 权限。设备顾问将使用附加的设备角色代表测试设备执行 AWS IoT MQTT 操作。如果缺少所需权限，则当设备尝试连接到 Device Advisor 端点时，会看到 `AWS_ERROR_MQTT_UNEXPECTED_HANGUP` 错误或发生意外断开连接。例如，如果您选择运行 MQTT Publish 测试用例，则角色中必须包含相应的 Connect ClientId 和 Publish 操作（您可以使用逗号分隔值来提供多个值，也可以使用通配符 (*) 字符提供前缀值。例如，要为任何以 `TestTopic` 开头的主题提供发布权限，您可以将“`TestTopic*`”作为资源值。下面是部分策略示例。

- 在设备角色中为资源类型定义的值与代码中使用的实际值不匹配。例如：角色 ClientId 定义与设备代码中实际 ClientId 使用的角色不匹配。像 Topic 和这样的 ClientId值在设备角色和代码中 TopicFilter 必须相同。
- 附加到设备的设备证书必须处于活跃状态，并且附加了具有资源所需操作权限的策略。请注意，设备证书策略允许或拒绝访问 AWS IoT 资源和 AWS IoT Core 数据平面操作。Device Advisor 要求您将活跃设备证书附加到您的设备，该证书可授予在测试用例期间使用的操作权限。

AWS IoT Device Management 疑难解答指南

> 帮助我们改进此主题
>
> 告诉我们如何优化内容

这是的疑难解答部分 AWS IoT Device Management。

主题

AWS IoT 作业疑难解答

这是 AWS IoT 作业的疑难解答部分。

如何找到 AWS IoT 作业终端节点？

如何找到 AWS IoT Jobs 控制平面端点？

AWS IoT 作业支持使用 HTTPS 协议控制平面 API 操作。确认您已使用 HTTPS 协议连接到正确的控制平面端点。

有关 AWS 特定区域终端节点的列表，请参阅AWS IoT 核心-控制平面端点。

有关符合 FIPS 的 AWS IoT Jobs 控制平面端点的列表，请参阅按服务划分的 FIPS 端点

> Note
>
> AWS IoT 任务和 AWS IoT Core 共享相同的 AWS 特定于区域的终端节点。

如何找到 AWS IoT 作业数据平面端点？

AWS IoT 作业支持使用 HTTPS 和 MQTT 协议进行数据平面 API 操作。确认您已使用 HTTPS 或 MQTT 协议连接到正确的数据端点。

- HTTPS 协议
  - 使用下面显示的 describe-endpoint CLI 命令或 DescribeEndpoint REST API。对于端点类型，请使用 `iot:Jobs`。

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

- MQTT 协议
  - 使用下面显示的 describe-endpoint CLI 命令或 DescribeEndpoint REST API。对于端点类型，请使用 `iot:Data-ATS`（推荐）或 `iot:Data`。

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS (recommended)
```

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

有关符合 FIPS 的 AWS IoT Jobs 数据平面端点的列表，请参阅按服务划分的 FIPS 端点

如何监控 AWS IoT 作业活动并提供指标？

使用 Amaz CloudWatch on 监控 AWS IoT 任务活动可以实时了解正在进行的 AWS IoT 任务操作，并通过 AWS IoT 规则 CloudWatch 发出警报，帮助控制成本。必须先配置日志记录，然后才能监控 AWS IoT 作业活动和设置 CloudWatch警报。有关如何设置日志记录的更多信息，请参阅配置 AWS IoT 日志。

有关亚马逊 CloudWatch 以及如何通过 IAM 用户角色设置 CloudWatch 资源使用权限的更多信息，请参阅亚马逊的身份和访问管理 CloudWatch。

如何使用 Amazon 设置 AWS IoT 任务指标和监控 CloudWatch？

要设置 AWS IoT 日志记录，请按照配置 AWS IoT 日志中概述的步骤进行操作。 AWS IoT 日志设置可以在 AWS Management Console AWS CLI、或 API 中完成。 AWS IoT 为特定事物组设置的日志只能在 AWS CLI 或 API 中完成。

AWS IoT 作业指标部分包含用于监控 AWS IoT 作业活动的 AWS IoT 作业指标。它说明了如何查看 AWS Management Console 和中的指标 AWS CLI。

此外，您可以设置 CloudWatch 警报，提醒您要密切监控的特定指标。有关警报设置的指导，请参阅使用 Amazon CloudWatch 警报。

设备队列和单台设备故障排除

任务执行无限期保持 **QUEUED** 状态

当状态为 QUEUED 的任务执行未进入下一个逻辑状态（例如 IN_PROGRESS、FAILED、或 TIMED_OUT）时，原因可能是以下情况之一：

- 在CloudWatch 控制台中的 CloudWatch 日志中查看您的设备活动。有关更多信息，请参阅AWS IoT 使用 CloudWatch 日志进行监控。
- 与该任务和后续任务执行关联的 IAM 角色可能不具备附加到该 IAM 角色的 IAM 策略其中一项策略声明中所列的正确权限。使用 `describe-job` API 确定与该任务和后续任务执行关联的 IAM 角色，并查看 IAM 策略了解正确的权限。更新策略权限声明后，您应该能够对资源执行 `AssumeRole` API 命令。

没有为我的事物或事物组创建任务执行

当任务将其状态更新为 IN_PROGRESS 时，它就会开始向目标组中的所有设备推出任务文档。此状态更新将为每台目标设备创建任务执行。如果未为其中一台目标设备创建任务执行，请参阅以下指导：

- thing 是否由任务直接定向，任务的状态是否为 IN_PROGRESS，任务是否是并发任务？ 如果所有三个条件都满足，则该任务仍向目标组中的所有设备发送任务执行，并且该特定 thing 尚未收到其任务执行。
  - 在 AWS 管理控制台中查看目标组中的设备是否有任务和任务状态或使用 `describe-job`API 命令。
  - 使用 `describe-job` API 命令查看任务的 IsConcurrent 属性设置为 true 还是 false。有关更多信息，请参阅任务限制。
- thing 并非直接由任务定向。
  - 如果 Thing 已添加到 ThingGroup，且任务已定向 ThingGroup，则请验证 Thing 是否属于 ThingGroup。
  - 如果该任务状态为 IN_PROGRESS 且属于并发任务的快照任务，则该任务仍向目标组中的所有设备发送任务执行，并且该特定 Thing 尚未收到其任务执行。

- 如果该任务状态为 IN_PROGRESS 且属于并发任务的持续任务，则该任务仍向目标组中的所有设备发送任务执行，并且该特定 Thing 尚未收到其任务执行。仅对于持续任务，您也可以从 ThingGroup 中删除 Thing，然后将 Thing 重新添加回 ThingGroup。
- 如果作业是状态为IN_PROGRESS且非并发状态的快照作 AWS IoT 业，则任务很可能未确认Thing或ThingGroup成员关系的情况。建议在AddThingToThingGroup呼叫后增加几秒钟的等待时间，然后再创建Job。或者，您可以将目标选择切换到Continuous，从而使服务回填延迟Thing和ThingGroup成员资格附件事件。

由于 **LimitedExceededException** 错误，新任务失败

如果您的任务创建失败并显示错误响应 LimitedExceededException，请调用 list-jobs API 并查看 isConcurrent=true 的所有任务，以确定您是否已达到任务并发限制。有关并发任务的更多信息，请参阅任务限制。要查看任务并发限制以及请求提高限制，请参阅 AWS IoT Device Management 任务限制和配额。

任务文档大小限制

任务文档的大小受 MQTT 负载大小的限制。如果您需要一个大于 32 kB（千字节）、32000 B（字节）的任务文档，请在 Amazon S3 中创建并存储该任务文档，然后在 CreateJob API 的 documentSource 字段中（或使用 AWS CLI）添加一个 Amazon S3 对象 URL。对于 AWS Management Console，请在创建任务时在 Amazon S3 网址文本框中添加 Amazon S3 对象 URL。

- AWS Management Console 创建任务文档：使用创建和管理作业 AWS Management Console
- AWS CLI 创建任务文档：使用创建和管理作业 AWS CLI
- CreateJobAPI 文档：CreateJob

设备端 MQTT 消息请求阈值限制

如果您收到错误代码 400 ThrottlingException，则设备端 MQTT 消息会因已达到设备端同时请求的限制而失败。有关阈值限制及其是否可调整的更多信息，请参阅 AWS IoT Device Management 任务限制和配额。

连接超时错误

错误代码 400 RequestExpired 表示由于延迟时间较长或客户端超时值较低而导致连接失败。

- 有关测试客户端和服务器端之间连接的信息，请参阅测试与设备数据端点的连接。

API 命令无效

确认输入了正确的 API 命令，以免出现错误消息，指出 API 命令无效。有关所有 AWS IoT API 命令的完整列表，请参阅 AWS IoT API 参考。

服务端连接错误

错误代码 503 `ServiceUnavailable` 表示服务器端出现错误。

- 有关所有 AWS 服务的当前状态，请参阅AWS Health Dashboard （所有 AWS 服务）。
- 有关您个人的当前状态，请参阅AWS Health Dashboard （个人 AWS 账户） AWS 账户。

机群索引排除指南

队列索引服务的聚合查询疑难解答

如果您遇到类型不匹配错误，则可以使用 CloudWatch Logs 来解决问题。 CloudWatch 在舰队索引服务写入日志之前，必须启用日志。有关更多信息，请参阅 AWS IoT 使用 CloudWatch 日志进行监控。

要在非托管式字段上进行聚合查询，必须指定您在传递给 `UpdateIndexingConfiguration` 或 update-indexing-configuration 的 `customFields` 参数中定义的字段。如果字段值与配置的字段数据类型不一致，则在执行聚合查询时忽略此值。

如果由于类型不匹配而无法对字段进行索引，则队列索引服务会向 Logs 发送错误日志。 CloudWatch 错误日志包含字段名称、无法转换的值以及设备的事物名称。下面是一个错误日志示例：

```
{
  "timestamp": "2017-02-20 20:31:22.932",
  "logLevel": "ERROR",
  "traceId": "79738924-1025-3a00-a669-7bec69f7f07a",
  "accountId": "000000000000",
  "status": "SucceededWithIssues",
  "eventType": "IndexingCustomFieldFailed",
  "thingName": "thing0",
  "failedCustomFields": [
    {
      "Name": "attributeName1",
      "Value": "apple",
      "ExpectedType": "String"
    },
```

```
    {
      "Name": "attributeName2",
      "Value": "2",
      "ExpectedType": "Boolean"
    }
  ]
}
```

如果设备已断开连接大约一小时，则连接状态 `timestamp` 值可能会缺失。对于持续会话，在客户端断开连接的时间超过为持续会话配置的时间 time-to-live (TTL) 之后，该值可能会丢失。仅为客户端 ID 具有匹配事物名称的连接，对连接状态数据建立索引。（客户端 ID 是用于将设备连接到的值 AWS IoT Core。）

实例集索引配置故障排除

无法降级实例集索引配置

当您想要移除与实例集指标或动态组关联的数据来源时，不支持降级实例集索引配置。

例如，如果您的索引配置包含注册表数据、影子数据和连接数据，并且查询 `thingName:TempSensor* AND shadow.desired.temperature>80` 中存在实例集指标，则更新索引配置以仅包含注册表数据将导致错误。

不支持修改现有机群指标使用的自定义字段。

由于实例集指标或动态组不兼容，无法更新您的索引配置

如果由于实例集指标或动态组不兼容而无法更新索引配置，请在更新索引配置之前删除不兼容的实例集指标或动态组。

位置索引和地理查询疑难解答

要解决位置索引和地理查询中的类型不匹配错误，您可以启用 CloudWatch 日志。有关如何监控 AWS IoT 使用的更多信息 CloudWatch，请按照 step-by-step 指南进行操作。

使用地理查询为位置数据编制索引时，您在中指定的位置字段`geoLocations`必须与传递到`UpdateIndexingConfiguration`的位置字段相匹配。如果存在不匹配的情况，队列索引会向发送不匹配的类型错误。 CloudWatch错误日志包含字段名称、无法转换的值以及设备的事物名称。

下面是一个错误日志示例：

```
{
"timestamp": "2023-11-09 01:39:43.466",
    "logLevel": "ERROR",
    "traceId": "79738924-1025-3a00-a669-7bec69f7f07a",
    "accountId": "123456789012",
    "status": "Failure",
    "eventType": "IndexingGeoLocationFieldFailed",
    "thingName": "thing0",
    "failedGeolocationFields": [
        {
"Name": "attributeName1",
            "Value": "apple",
            "ExpectedType": "Geopoint"
        }
    ],
    "reason": "failed to index the field because it could not be converted to one of
 the expected geoLocation formats."
}
```

有关更多信息，请参阅 ???。

机群故障排除指标

看不到中的数据点 CloudWatch

如果您能够创建队列指标，但无法在中看到数据点 CloudWatch，则很可能没有符合查询字符串条件的东西。

请参阅以下示例命令，了解如何创建机群指标：

```
aws iot create-fleet-metric --metric-name "example_FM" --query-string
 "thingName:TempSensor* AND attributes.temperature>80" --period 60 --aggregation-field
 "attributes.temperature" --aggregation-type name=Statistics,values=count
```

如果没有符合查询字符串条件 `--query-string "thingName:TempSensor* AND attributes.temperature>80"` 的事物：

- 使用`values=count`，您将能够创建舰队指标，并且会有数据点可供显示 CloudWatch。该值 `count` 的数据点始终为0。
- 使用`values`其他`count`，您将能够创建舰队指标，但不会在中看到舰队指标 CloudWatch ，也不会显示任何数据点 CloudWatch。

AWS IoT 错误

帮助我们改进此主题

告诉我们如何优化内容

本节列出了发送的错误代码 AWS IoT。

消息代理错误代码

错误代码	错误描述
400	错误请求。
401	未授权。
403	禁止。
503	服务不可用。

身份和安全错误代码

错误代码	错误描述
401	未授权。

设备影子错误代码

错误代码	错误描述
400	错误请求。
401	未授权。
403	禁止。
404	未找到。
409	冲突。

错误代码	错误描述
413	请求太大。
422	无法处理请求。
429	过多请求。
500	内部错误。
503	服务不可用。

AWS IoT 设备 SDK、移动 SDK 和 AWS IoT 设备客户端

本页汇总了 AWS IoT 设备软件开发工具包、开源库、开发者指南、示例应用程序和移植指南，以帮助您使用 AWS IoT 自己选择的硬件平台构建创新的物联网解决方案。

这些 SDK 供您在 IoT 设备上使用。如果您正在开发用于移动设备的 IoT 应用，请参阅 AWS 移动 SDK。如果您正在开发 IoT 应用程序或服务器端程序，请参阅 AWS 软件开发工具包。

AWS IoT 设备软件开发工具包

AWS IoT 设备软件开发工具包包括开源库、带有示例的开发者指南和移植指南，因此您可以在自己选择的硬件平台上构建创新的物联网产品或解决方案。

> **Note**
>
> AWS IoT 设备软件开发工具包已经发布了 MQTT 5 客户端。 AWS IoT 设备软件开发工具包不支持在 macOS 上使用 TLS 1.3。

这些 SDK 可帮助您使用 MQTT 和 WSS 协议将 IoT 设备连接到 AWS IoT 。

C++

AWS IoT C++ 设备开发工具包

C AWS IoT ++ 设备 SDK 允许开发人员使用 AWS 和 AWS IoT API 构建互联应用程序。特别是，此 SDK 面向没有资源限制且需要高级特征（例如，消息队列、多线程支持和最新的语言特征）的设备而设计。有关更多信息，请参阅下列内容：

- AWS IoT 设备 SDK C++ v2 已开启 GitHub
- AWS IoT 设备 SDK C++ v2 自述文件
- AWS IoT 设备 SDK C++ v2 示例
- AWS IoT 设备 SDK C++ v2 API 文档

Python

AWS IoT Python 设备软件开发工具包

适用于 Python 的 AWS IoT 设备 SDK 使开发人员可以编写 Python 脚本，以便使用他们的设备通过协议通过 MQTT 或 MQTT 访问 AWS IoT 平台。 WebSocket 通过将设备连接到 AWS IoT，用户可以安全地使用消息代理、规则和影子，以及由其他 AWS 服务（如 Kinesis AWS IoT 和 Amazon S3 AWS Lambda 等）提供的消息代理、规则和影子。

- AWS IoT 适用于 Python v2 的设备 SDK 已开启 GitHub
- AWS IoT 适用于 Python v2 的设备 SDK 自述文件
- AWS IoT 适用于 Python v2 的设备软件开发工具包示例
- AWS IoT 适用于 Python 的设备开发工具包 v2 API 文档

JavaScript

AWS IoT 适用于的设备 SDK JavaScript

aws-iot-device-sdk.js 包使开发人员可以编写 AWS IoT 通过协议使用 MQTT 或 MQTT 进行访问的 JavaScript 应用程序。 WebSocket 它可用于 Node.js 环境和浏览器应用程序。有关更多信息，请参阅下列内容：

- AWS IoT 适用于 JavaScript v2 的设备 SDK 已开启 GitHub
- AWS IoT 适用于 JavaScript v2 的设备 SDK 自述文件
- AWS IoT 适用于 JavaScript v2 示例的设备 SDK
- AWS IoT 适用于 JavaScript v2 的设备 SDK API 文档

Java

AWS IoT 适用于 Java 的设备 SDK

适用于 Java 的 AWS IoT 设备 SDK 使 Java 开发人员能够通过协议通过 MQTT 或 MQTT 访问该 AWS IoT WebSocket 平台。该 SDK 内置有影子支持。您可以使用 HTTP 方法 (包括 GET、UPDATE 和 DELETE) 访问影子。该 SDK 还支持简化的影子访问模型，开发人员只需要使用 getter 和 setter 方法即可与影子交换数据，而不必对任何 JSON 文档进行序列化或反序列化。

> **Note**
>
> 适用于 Java v2 的 AWS IoT 设备 SDK 现在支持安卓开发。如需了解更多信息，请参阅适用于 Android 的AWS IoT 设备 SDK。

有关更多信息，请参阅下列内容：

- AWS IoT 适用于 Java v2 的设备 SDK 已开启 GitHub
- AWS IoT 适用于 Java 的设备 SDK v2 自述文件
- AWS IoT 适用于 Java 的设备 SDK v2 示例
- AWS IoT 适用于 Java 的设备 SDK v2 API 文档

AWS IoT 适用于嵌入式 C 的设备 SDK

> **Note**
>
> 该 SDK 供经验丰富的嵌入式软件开发人员使用。

AWS IoT Device SDK for Embedded C (C-SDK) 是 MIT 开源许可下的 C 源文件集合，可用于嵌入式应用程序，将物联网设备安全地连接到。 AWS IoT Core它包括 MQTT 客户端、JSON 解析 AWS IoT 器和 Device Shadow、 AWS IoT 作业、 AWS IoT 队列配置和 AWS IoT Device Defender 库。该开发工具包以源代码形式分发，可构建到客户固件和应用程序代码、其他库以及您选择的操作系统 (OS) 中。

AWS IoT Device SDK for Embedded C 通常针对需要优化 C 语言运行时的资源受限的设备。您可以在任何操作系统上使用此 SDK，并将其托管在任何类型的处理器（例如 MCU 和 MPU）上。

有关更多信息，请参阅下列内容：

- AWS IoT 适用于嵌入式 C 的设备 SDK 已开启 GitHub
- AWS IoT 适用于嵌入式 C 的设备 SDK 自述文件
- AWS IoT 嵌入式 C 示例的设备 SDK

较早的 AWS IoT 设备 SDK 版本

这些是 AWS IoT 设备软件开发工具包的早期版本，已被上面列出的较新版本所取代。这些 SDK 仅接收维护和安全更新。它们不会更新以获取新特征，也不应用于新项目。

- AWS IoT C++ 设备 SDK 已打开 GitHub
- AWS IoT C++ 设备 SDK 自述文件

- AWS IoT 适用于 Python v1 的设备 SDK 已开启 GitHub
- AWS IoT 适用于 Python 的设备 SDK v1 自述文件
- AWS IoT 适用于 Java 的设备 SDK 已开启 GitHub
- AWS IoT 适用于 Java 的设备 SDK 自述文件
- AWS IoT 适用于开启的设备 S JavaScript DK GitHub
- AWS IoT 适用于 JavaScript 自述文件的设备 SDK
- Arduino Yún SDK 已开启 GitHub
- Arduino Yún 软件开发工具包自述文件

AWS 移动 SDK

AWS 移动软件开发工具包为移动应用开发者提供特定平台的支持，包括服务的 API AWS IoT Core 、使用 MQTT 的物联网设备通信以及其他服务的 API。 AWS

Android

AWS Mobile SDK for Android

AWS Mobile SDK for Android 包含一个库、示例和文档，供开发人员用来构建互联的移动应用程序 AWS。此 SDK 还包括对 MQTT 设备通信和调用 AWS IoT Core 服务 API 的支持。有关更多信息，请参阅下列内容：

- AWS Mobile SDK for Android on GitHub
- AWS Mobile SDK for Android Readme (自述文件)
- AWS Mobile SDK for Android 示例
- AWS Mobile SDK for Android API 参考
- AWSIoTClient 类参考文档

iOS

AWS Mobile SDK for iOS

AWS Mobile SDK for iOS 是一个开源软件开发套件，在 Apache 开源许可证下分发。 AWS Mobile SDK for iOS 提供了一个库、代码示例和文档，以帮助开发人员使用构建互联的移动应用程序 AWS。此 SDK 还包括对 MQTT 设备通信和调用 AWS IoT Core 服务 API 的支持。有关更多信息，请参阅下列内容：

- AWS Mobile SDK for iOS on GitHub
- AWS Mobile SDK for iOS Readme (自述文件)
- AWS Mobile SDK for iOS 示例
- AWSIoT 中的类参考文档 AWS Mobile SDK for iOS

AWS IoT 设备客户端

AWS IoT 设备客户端提供的代码可帮助您的设备连接 AWS IoT、执行队列配置任务、支持设备安全策略、使用安全隧道连接以及处理设备上的作业。您可以在设备上安装此软件来处理这些常规设备任务，以便专注于特定的解决方案。

> **Note**
>
> AWS IoT 设备客户端适用于搭载 x86_64 或 ARM 处理器的基于微处理器的物联网设备以及常见 Linux 操作系统。

C++

AWS IoT 设备客户端

有关 C++ 中 AWS IoT 设备客户端的更多信息，请参阅以下内容：

- AWS IoT C++ 中的设备客户端源代码在 GitHub
- AWS IoT C++ 中的设备客户端自述文件

AWS IoT 使用 AWS SDK 的代码示例

以下代码示例说明如何 AWS IoT 使用 AWS 软件开发套件 (SDK)。

操作是大型程序的代码摘录，必须在上下文中运行。您可以通过操作了解如何调用单个服务函数，还可以通过函数相关场景和跨服务示例的上下文查看操作。

场景是展示如何通过在同一服务中调用多个函数来完成特定任务任务的代码示例。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

开始使用

你好 AWS IoT

以下代码示例展示了如何开始使用 AWS IoT。

C++

SDK for C++

C MakeLists .txt cMake 文件的代码。

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS iot)

# Set this project's name.
project("hello_iot")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
```

```
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
    # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
    # and set the proper subdirectory to the executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_iot.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

hello_iot.cpp 源文件的代码。

```
#include <aws/core/Aws.h>
#include <aws/iot/IoTClient.h>
#include <aws/iot/model/ListThingsRequest.h>
#include <iostream>

/*
 *  A "Hello IoT" starter application which initializes an AWS IoT client and
 *  lists the AWS IoT topics in the current account.
 *
 *  main function
 *
 *  Usage: 'hello_iot'
 *
 */
```

```
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    //  Optional: change the log level for debugging.
    //  options.loggingOptions.logLevel = Aws::Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::IoT::IoTClient iotClient(clientConfig);
        // List the things in the current account.
        Aws::IoT::Model::ListThingsRequest listThingsRequest;

        Aws::String nextToken; // Used for pagination.
        Aws::Vector<Aws::IoT::Model::ThingAttribute> allThings;

        do {
            if (!nextToken.empty()) {
                listThingsRequest.SetNextToken(nextToken);
            }

            Aws::IoT::Model::ListThingsOutcome listThingsOutcome =
 iotClient.ListThings(
                    listThingsRequest);
            if (listThingsOutcome.IsSuccess()) {
                const Aws::Vector<Aws::IoT::Model::ThingAttribute> &things =
 listThingsOutcome.GetResult().GetThings();
                allThings.insert(allThings.end(), things.begin(), things.end());
                nextToken = listThingsOutcome.GetResult().GetNextToken();
            }
            else {
                std::cerr << "List things failed"
                          << listThingsOutcome.GetError().GetMessage() <<
 std::endl;
                break;
            }
        } while (!nextToken.empty());

        std::cout << allThings.size() << " thing(s) found." << std::endl;
        for (auto const &thing: allThings) {
            std::cout << thing.GetThingName() << std::endl;
        }
```

```
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- 有关 API 详细信息，请参阅《AWS SDK for C++ API Reference》中的 listThings。

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iot.IotClient;
import software.amazon.awssdk.services.iot.model.ListThingsRequest;
import software.amazon.awssdk.services.iot.model.ListThingsResponse;
import software.amazon.awssdk.services.iot.model.ThingAttribute;
import java.util.List;

public class HelloIoT {
    public static void main(String[] args) {
        System.out.println("Hello AWS IoT. Here is a listing of your AWS IoT
 Things:");
        IotClient iotClient = IotClient.builder()
            .region(Region.US_EAST_1)
            .build();

        listAllThings(iotClient);
```

```
    }

    public static void listAllThings( IotClient iotClient) {
        ListThingsRequest thingsRequest = ListThingsRequest.builder()
            .maxResults(10)
            .build();

        ListThingsResponse response = iotClient.listThings(thingsRequest) ;
        List<ThingAttribute> thingList = response.things();
        for (ThingAttribute attribute : thingList) {
            System.out.println("Thing name: "+attribute.thingName());
            System.out.println("Thing ARN: "+attribute.thingArn());
        }
    }
}
```

- 有关 API 详细信息，请参阅《AWS SDK for Java 2.x API Reference》中的 listThings。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import aws.sdk.kotlin.services.iot.IotClient
import aws.sdk.kotlin.services.iot.model.ListThingsRequest

suspend fun main() {
    println("A listing of your AWS IoT Things:")
    listAllThings()
}

suspend fun listAllThings() {
    val thingsRequest =
        ListThingsRequest {
            maxResults = 10
        }
```

```
    IotClient { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.listThings(thingsRequest)
        val thingList = response.things
        if (thingList != null) {
            for (attribute in thingList) {
                println("Thing name ${attribute.thingName}")
                println("Thing ARN: ${attribute.thingArn}")
            }
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用于 K otlin 的AWS SD K 中的 List Things API 参考。

代码示例

- AWS IoT 使用 AWS SDK 的操作
  - AttachThingPrincipal与 AWS SDK 或 CLI 配合使用
  - CreateKeysAndCertificate与 AWS SDK 或 CLI 配合使用
  - CreateThing与 AWS SDK 或 CLI 配合使用
  - CreateTopicRule与 AWS SDK 或 CLI 配合使用
  - DeleteCertificate与 AWS SDK 或 CLI 配合使用
  - DeleteThing与 AWS SDK 或 CLI 配合使用
  - DeleteTopicRule与 AWS SDK 或 CLI 配合使用
  - DescribeEndpoint与 AWS SDK 或 CLI 配合使用
  - DescribeThing与 AWS SDK 或 CLI 配合使用
  - DetachThingPrincipal与 AWS SDK 或 CLI 配合使用
  - ListCertificates与 AWS SDK 或 CLI 配合使用
  - ListThings与 AWS SDK 或 CLI 配合使用
  - SearchIndex与 AWS SDK 或 CLI 配合使用
  - UpdateIndexingConfiguration与 AWS SDK 或 CLI 配合使用
  - UpdateThing与 AWS SDK 或 CLI 配合使用
- AWS IoT 使用 AWS SDK 的场景
  - 使用 AWS IoT SDK 处理 AWS IoT 设备、事物和影子

AWS IoT 使用 AWS SDK 的操作

以下代码示例演示如何使用 AWS 软件开发工具包执行单个 AWS IoT 操作。这些摘录调用 AWS IoT API，是大型程序的代码摘录，这些程序必须在上下文中运行。每个示例都包含一个指向的链接 GitHub，您可以在其中找到有关设置和运行代码的说明。

以下示例仅包括最常用的操作。有关完整列表，请参阅 AWS IoT API 参考。

示例

- AttachThingPrincipal与 AWS SDK 或 CLI 配合使用
- CreateKeysAndCertificate与 AWS SDK 或 CLI 配合使用
- CreateThing与 AWS SDK 或 CLI 配合使用
- CreateTopicRule与 AWS SDK 或 CLI 配合使用
- DeleteCertificate与 AWS SDK 或 CLI 配合使用
- DeleteThing与 AWS SDK 或 CLI 配合使用
- DeleteTopicRule与 AWS SDK 或 CLI 配合使用
- DescribeEndpoint与 AWS SDK 或 CLI 配合使用
- DescribeThing与 AWS SDK 或 CLI 配合使用
- DetachThingPrincipal与 AWS SDK 或 CLI 配合使用
- ListCertificates与 AWS SDK 或 CLI 配合使用
- ListThings与 AWS SDK 或 CLI 配合使用
- SearchIndex与 AWS SDK 或 CLI 配合使用
- UpdateIndexingConfiguration与 AWS SDK 或 CLI 配合使用
- UpdateThing与 AWS SDK 或 CLI 配合使用

`AttachThingPrincipal`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `AttachThingPrincipal`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Attach a principal to an AWS IoT thing.
/*!
  \param principal: A principal to attach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::attachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::AttachThingPrincipalRequest request;
    request.SetPrincipal(principal);
    request.SetThingName(thingName);
    Aws::IoT::Model::AttachThingPrincipalOutcome outcome =
 client.AttachThingPrincipal(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully attached principal to thing." << std::endl;
    }
    else {
        std::cerr << "Failed to attach principal to thing." <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考中的AttachThing委托人。

CLI

AWS CLI

在你的东西上附加证书

以下attach-thing-principal示例将证书附加到 MyTemperatureSensor 事物。该证书由 ARN 标识。您可以在物 AWS 联网控制台中找到证书的 ARN。

```
aws iot attach-thing-principal \
    --thing-name MyTemperatureSensor \
    --principal arn:aws:iot:us-
west-2:123456789012:cert/2e1eb273792174ec2b9bf4e9b37e6c6c692345499506002a35159767055278e8
```

此命令不生成任何输出。

有关更多信息，请参阅《AWS IoT 开发人员指南》中的如何使用注册表管理事物。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》中的 “AttachThing主体”。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   public static void attachCertificateToThing(IotClient iotClient, String
thingName, String certificateArn) {
       // Attach the certificate to the thing.
       AttachThingPrincipalRequest principalRequest =
AttachThingPrincipalRequest.builder()
           .thingName(thingName)
           .principal(certificateArn)
           .build();

       AttachThingPrincipalResponse attachResponse =
iotClient.attachThingPrincipal(principalRequest);
```

```
        // Verify the attachment was successful.
        if (attachResponse.sdkHttpResponse().isSuccessful()) {
            System.out.println("Certificate attached to Thing successfully.");

            // Print additional information about the Thing.
            describeThing(iotClient, thingName);
        } else {
            System.err.println("Failed to attach certificate to Thing. HTTP
 Status Code: " +
                    attachResponse.sdkHttpResponse().statusCode());
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考中的AttachThing委托人。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun attachCertificateToThing(
    thingNameVal: String?,
    certificateArn: String?,
) {
    val principalRequest =
        AttachThingPrincipalRequest {
            thingName = thingNameVal
            principal = certificateArn
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.attachThingPrincipal(principalRequest)
        println("Certificate attached to $thingNameVal successfully.")
    }
}
```

- 有关 API 的详细信息，请参阅适用于 K otlin AWS 的 SDK 中的AttachThing主体 API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`CreateKeysAndCertificate`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateKeysAndCertificate`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Create keys and certificate for an Aws IoT device.
//! This routine will save certificates and keys to an output folder, if
 provided.
/*!
  \param outputFolder: Location for storing output in files, ignored when string
 is empty.
  \param certificateARNResult: A string to receive the ARN of the created
 certificate.
  \param certificateID: A string to receive the ID of the created certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createKeysAndCertificate(const Aws::String &outputFolder,
                                           Aws::String &certificateARNResult,
                                           Aws::String &certificateID,
                                           const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::CreateKeysAndCertificateRequest
 createKeysAndCertificateRequest;
```

```
    Aws::IoT::Model::CreateKeysAndCertificateOutcome outcome =
            client.CreateKeysAndCertificate(createKeysAndCertificateRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created a certificate and keys" << std::endl;
        certificateARNResult = outcome.GetResult().GetCertificateArn();
        certificateID = outcome.GetResult().GetCertificateId();
        std::cout << "Certificate ARN: " << certificateARNResult << ",
 certificate ID: "
                  << certificateID << std::endl;

        if (!outputFolder.empty()) {
            std::cout << "Writing certificate and keys to the folder '" <<
 outputFolder
                      << "'." << std::endl;
            std::cout << "Be sure these files are stored securely." << std::endl;

            Aws::String certificateFilePath = outputFolder + "/
certificate.pem.crt";
            std::ofstream certificateFile(certificateFilePath);
            if (!certificateFile.is_open()) {
                std::cerr << "Error opening certificate file, '" <<
 certificateFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            certificateFile << outcome.GetResult().GetCertificatePem();
            certificateFile.close();

            const Aws::IoT::Model::KeyPair &keyPair =
 outcome.GetResult().GetKeyPair();

            Aws::String privateKeyFilePath = outputFolder + "/private.pem.key";
            std::ofstream privateKeyFile(privateKeyFilePath);
            if (!privateKeyFile.is_open()) {
                std::cerr << "Error opening private key file, '" <<
 privateKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            privateKeyFile << keyPair.GetPrivateKey();
            privateKeyFile.close();
```

```
            Aws::String publicKeyFilePath = outputFolder + "/public.pem.key";
            std::ofstream publicKeyFile(publicKeyFilePath);
            if (!publicKeyFile.is_open()) {
                std::cerr << "Error opening public key file, '" <<
 publicKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            publicKeyFile << keyPair.GetPublicKey();
        }
    }
    else {
        std::cerr << "Error creating keys and certificate: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考CreateKeysAndCertificate中的。

CLI

AWS CLI

创建 RSA 密钥对并颁发 X.509 证书

以下内容create-keys-and-certificate创建一个 2048 位的 RSA 密钥对，并使用已颁发的公钥颁发 X.509 证书。由于这是 AWS 物联网唯一一次为此证书提供私钥，因此请务必将其保存在安全的地方。

```
aws iot create-keys-and-certificate \
    --certificate-pem-outfile "myTest.cert.pem" \
    --public-key-outfile "myTest.public.key" \
    --private-key-outfile "myTest.private.key"
```

输出：

```
{
```

```
    "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2
    "certificateId":
 "9894ba17925e663f1d29c23af4582b8e3b7619c31f3fbd93adcb51ae54b83dc2",
    "certificatePem": "
-----BEGIN CERTIFICATE-----
MIICiTCCEXAMPLE6m7oRw0uXOjANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMC
VVMxCzAJBgNVBAgEXAMPLEAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6
b24xFDASBgNVBAsTC0lBTSEXAMPLE2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAd
BgkqhkiG9w0BCQEWEG5vb25lQGFtYEXAMPLEb20wHhcNMTEwNDI1MjA0NTIxWhcN
MTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBhMCEXAMPLEJBgNVBAgTAldBMRAwDgYD
VQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDAEXAMPLEsTC0lBTSBDb25z
b2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEXAMPLE25lQGFt
YXpvbi5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMaK0dn+aEXAMPLE
EXAMPLEfEvySWtC2XADZ4nB+BLYgVIk60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9T
rDHudUZEXAMPLELG5M43q7Wgc/MbQITxOUSQv7c7ugFFDzQGBzZswY6786m86gpE
Ibb3OhjZnzcvQAEXAMPLEWIMm2nrAgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4
nUhVVxYUntneD9+h8Mg9qEXAMPLEyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0Fkb
FFBjvSfpJIlJ00zbhNYS5f6GuoEDEXAMPLEBHjJnyp378OD8uTs7fLvjx79LjSTb
NYiytVbZPQUQ5Yaxu2jXnimvw3rrszlaEXAMPLE=
-----END CERTIFICATE-----\n",
    "keyPair": {
        "PublicKey": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\nMMEXAMPLEuuN/
dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\n59VK7cEXAMPLE67GK+y
+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQ
\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa
\nhJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\nFQIDAQAB
\n-----END PUBLIC KEY-----\n",
        "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\nkey omittted for security
 reasons\n-----END RSA PRIVATE KEY-----\n"
    }
}
```

有关更多信息，请参阅《物 AWS 联网开发者指南》中的创建和注册物AWS 联网设备证书。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateKeysAndCertificate中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public static String createCertificate(IotClient iotClient) {
        try {
            CreateKeysAndCertificateResponse response =
iotClient.createKeysAndCertificate();
            String certificatePem = response.certificatePem();
            String certificateArn = response.certificateArn();

            // Print the details.
            System.out.println("\nCertificate:");
            System.out.println(certificatePem);
            System.out.println("\nCertificate ARN:");
            System.out.println(certificateArn);
            return certificateArn;

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return "";
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CreateKeysAndCertificate中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public static String createCertificate(IotClient iotClient) {
        try {
            CreateKeysAndCertificateResponse response =
iotClient.createKeysAndCertificate();
            String certificatePem = response.certificatePem();
            String certificateArn = response.certificateArn();

            // Print the details.
            System.out.println("\nCertificate:");
            System.out.println(certificatePem);
            System.out.println("\nCertificate ARN:");
            System.out.println(certificateArn);
            return certificateArn;

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return "";
    }
```

- 有关 API 的详细信息，请参阅适用CreateKeysAndCertificate于 K otlin 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`CreateThing`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateThing`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Create an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::CreateThingRequest createThingRequest;
    createThingRequest.SetThingName(thingName);

    Aws::IoT::Model::CreateThingOutcome outcome = iotClient.CreateThing(
            createThingRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to create thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考CreateThing中的。

CLI

AWS CLI

示例 1：在注册表中创建事物记录

以下create-thing示例在 AWS IoT 事物注册表中为设备创建条目。

```
aws iot create-thing \
    --thing-name SampleIoTThing
```

输出：

```
{
    "thingName": "SampleIoTThing",
    "thingArn": "arn:aws:iot:us-west-2: 123456789012:thing/SampleIoTThing",
    "thingId": " EXAMPLE1-90ab-cdef-fedc-ba987EXAMPLE "
}
```

示例 2：定义与事物类型关联的事物

以下create-thing示例创建一个具有指定事物类型及其属性的事物。

```
aws iot create-thing \
    --thing-name "MyLightBulb" \
    --thing-type-name "LightBulb" \
    --attribute-payload "{"attributes": {"wattage":"75", "model":"123"}}"
```

输出：

```
{
    "thingName": "MyLightBulb",
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
    "thingId": "40da2e73-c6af-406e-b415-15acae538797"
}
```

有关更多信息，请参阅《物AWS 联网开发人员指南》中的 “如何使用注册表和事物类型管理事物”。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考CreateThing中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public static void createIoTThing(IotClient iotClient, String thingName) {
        try {
            CreateThingRequest createThingRequest = CreateThingRequest.builder()
                .thingName(thingName)
                .build();

            CreateThingResponse createThingResponse =
iotClient.createThing(createThingRequest);
            System.out.println(thingName +" was successfully created. The ARN
value is " + createThingResponse.thingArn());

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考CreateThing中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun createIoTThing(thingNameVal: String) {
    val createThingRequest =
        CreateThingRequest {
            thingName = thingNameVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.createThing(createThingRequest)
        println("Created $thingNameVal}")
    }
}
```

- 有关 API 的详细信息，请参阅适用CreateThing于 K otlin 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`CreateTopicRule`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `CreateTopicRule`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Create an AWS IoT rule with an SNS topic as the target.
/*!
  \param ruleName: The name for the rule.
  \param snsTopic: The SNS topic ARN for the action.
  \param sql: The SQL statement used to query the topic.
  \param roleARN: The IAM role ARN for the action.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
```

```
 */
bool
AwsDoc::IoT::createTopicRule(const Aws::String &ruleName,
                             const Aws::String &snsTopicARN, const Aws::String
 &sql,
                             const Aws::String &roleARN,
                             const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::CreateTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::SnsAction snsAction;
    snsAction.SetTargetArn(snsTopicARN);
    snsAction.SetRoleArn(roleARN);

    Aws::IoT::Model::Action action;
    action.SetSns(snsAction);

    Aws::IoT::Model::TopicRulePayload topicRulePayload;
    topicRulePayload.SetSql(sql);
    topicRulePayload.SetActions({action});

    request.SetTopicRulePayload(topicRulePayload);
    auto outcome = iotClient.CreateTopicRule(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created topic rule " << ruleName << "." <<
 std::endl;
    }
    else {
        std::cerr << "Error creating topic rule " << ruleName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考中的CreateTopic规则。

CLI

AWS CLI

创建发送 Amazon SNS 警报的规则

以下create-topic-rule示例创建了一条规则，该规则用于在设备影子中发现的土壤湿度读数较低时发送 Amazon SNS 消息。

```
aws iot create-topic-rule \
    --rule-name "LowMoistureRule" \
    --topic-rule-payload file://plant-rule.json
```

该示例要求将以下 JSON 代码保存到名为的文件中plant-rule.json：

```
{
    "sql": "SELECT * FROM '$aws/things/MyRPi/shadow/update/accepted' WHERE
 state.reported.moisture = 'low'\n",
    "description": "Sends an alert whenever soil moisture level readings are too
 low.",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [{
            "sns": {
                "targetArn": "arn:aws:sns:us-
west-2:123456789012:MyRPiLowMoistureTopic",
                "roleArn": "arn:aws:iam::123456789012:role/service-role/
MyRPiLowMoistureTopicRole",
                "messageFormat": "RAW"
            }
    }]
}
```

此命令不生成任何输出。

有关更多信息，请参阅《AWS 物联网开发人员指南》中的创建AWS 物联网规则。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》中的CreateTopic规则。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   public static void createIoTRule(IotClient iotClient, String roleARN, String
ruleName, String action) {
       try {
           String sql = "SELECT * FROM '" + TOPIC + "'";
           SnsAction action1 = SnsAction.builder()
               .targetArn(action)
               .roleArn(roleARN)
               .build();

           // Create the action.
           Action myAction = Action.builder()
               .sns(action1)
               .build();

           // Create the topic rule payload.
           TopicRulePayload topicRulePayload = TopicRulePayload.builder()
               .sql(sql)
               .actions(myAction)
               .build();

           // Create the topic rule request.
           CreateTopicRuleRequest topicRuleRequest =
CreateTopicRuleRequest.builder()
               .ruleName(ruleName)
               .topicRulePayload(topicRulePayload)
               .build();

           // Create the rule.
           iotClient.createTopicRule(topicRuleRequest);
           System.out.println("IoT Rule created successfully.");

       } catch (IotException e) {
```

```
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考中的CreateTopic规则。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun createIoTRule(
    roleARNVal: String?,
    ruleNameVal: String?,
    action: String?,
) {
    val sqlVal = "SELECT * FROM '$TOPIC '"
    val action1 =
        SnsAction {
            targetArn = action
            roleArn = roleARNVal
        }

    val myAction =
        Action {
            sns = action1
        }

    val topicRulePayloadVal =
        TopicRulePayload {
            sql = sqlVal
            actions = listOf(myAction)
        }

    val topicRuleRequest =
```

```
        CreateTopicRuleRequest {
            ruleName = ruleNameVal
            topicRulePayload = topicRulePayloadVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.createTopicRule(topicRuleRequest)
        println("IoT rule created successfully.")
    }
}
```

- 有关 API 的详细信息，请参阅适用于 K otlin 的AWS SDK 中的CreateTopic规则 API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteCertificate`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteCertificate`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Delete a certificate.
/*!
  \param certificateID: The ID of a certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteCertificate(const Aws::String &certificateID,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
```

```
    Aws::IoT::Model::DeleteCertificateRequest request;
    request.SetCertificateId(certificateID);

    Aws::IoT::Model::DeleteCertificateOutcome outcome =
 iotClient.DeleteCertificate(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted certificate " << certificateID <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting certificate " << certificateID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考DeleteCertificate中的。

CLI

AWS CLI

删除设备证书

以下delete-certificate示例删除具有指定 ID 的设备证书。

```
aws iot delete-certificate \
    --certificate-id
 c0c57bbc8baaf4631a9a0345c957657f5e710473e3ddbee1428d216d54d53ac9
```

此命令不生成任何输出。

有关更多信息，请参阅 AWS IoT API 参考DeleteCertificate中的。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteCertificate中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   public static void deleteCertificate(IotClient iotClient, String
certificateArn ) {
       DeleteCertificateRequest certificateProviderRequest =
DeleteCertificateRequest.builder()
           .certificateId(extractCertificateId(certificateArn))
           .build();

       iotClient.deleteCertificate(certificateProviderRequest);
       System.out.println(certificateArn +" was successfully deleted.");
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteCertificate中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun deleteCertificate(certificateArn: String) {
    val certificateProviderRequest =
        DeleteCertificateRequest {
            certificateId = extractCertificateId(certificateArn)
        }
    IotClient { region = "us-east-1" }.use { iotClient ->
```

```
        iotClient.deleteCertificate(certificateProviderRequest)
        println("$certificateArn was successfully deleted.")
    }
}
```

- 有关 API 的详细信息，请参阅适用DeleteCertificate于 K otlin 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DeleteThing`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DeleteThing`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Delete an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteThingRequest request;
    request.SetThingName(thingName);
    const auto outcome = iotClient.DeleteThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted thing " << thingName << std::endl;
    }
```

```
    else {
        std::cerr << "Error deleting thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考DeleteThing中的。

CLI

AWS CLI

显示有关事物的详细信息

以下delete-thing示例从您 AWS 账户的 AWS IoT 注册表中删除一个事物。

aws iot 删除-thing-thing-name “” FourthBulb

此命令不生成任何输出。

有关更多信息，请参阅《AWS IoT 开发人员指南》中的如何使用注册表管理事物。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DeleteThing中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public static void deleteIoTThing(IotClient iotClient, String thingName) {
        try {
            DeleteThingRequest deleteThingRequest = DeleteThingRequest.builder()
                .thingName(thingName)
                .build();
```

```
            iotClient.deleteThing(deleteThingRequest);
            System.out.println("Deleted Thing " + thingName);

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DeleteThing中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun deleteIoTThing(thingNameVal: String) {
    val deleteThingRequest =
        DeleteThingRequest {
            thingName = thingNameVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteThing(deleteThingRequest)
        println("Deleted $thingNameVal")
    }
}
```

- 有关 API 的详细信息，请参阅适用DeleteThing于 K otlin 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

DeleteTopicRule与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DeleteTopicRule。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Delete an AWS IoT rule.
/*!
  \param ruleName: The name for the rule.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteTopicRule(const Aws::String &ruleName,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::DeleteTopicRuleOutcome outcome = iotClient.DeleteTopicRule(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted rule " << ruleName << std::endl;
    }
    else {
        std::cerr << "Failed to delete rule " << ruleName <<
                  ": " << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考中的DeleteTopic规则。

CLI

AWS CLI

删除规则

以下delete-topic-rule示例删除了指定的规则。

```
aws iot delete-topic-rule \
    --rule-name "LowMoistureRule"
```

此命令不生成任何输出。

有关更多信息，请参阅《AWS 物联网开发人员指南》中的删除规则。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》中的DeleteTopic规则。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

DescribeEndpoint与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 DescribeEndpoint。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Describe the endpoint specific to the AWS account making the call.
/*!
  \param endpointResult: String to receive the endpoint result.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
```

```
bool AwsDoc::IoT::describeEndpoint(Aws::String &endpointResult,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String endpoint;
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DescribeEndpointRequest describeEndpointRequest;
    describeEndpointRequest.SetEndpointType(
            "iot:Data-ATS"); // Recommended endpoint type.

    Aws::IoT::Model::DescribeEndpointOutcome outcome =
 iotClient.DescribeEndpoint(
            describeEndpointRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully described endpoint." << std::endl;
        endpointResult = outcome.GetResult().GetEndpointAddress();
    }
    else {
        std::cerr << "Error describing endpoint" <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考DescribeEndpoint中的。

CLI

AWS CLI

示例 1：获取您当前的 AWS 终端节点

以下`describe-endpoint`示例检索应用所有命令的默认 AWS 端点。

```
aws iot describe-endpoint
```

输出：

```
{
```

```
    "endpointAddress": "abc123defghijk.iot.us-west-2.amazonaws.com"
}
```

有关更多信息，请参阅DescribeEndpoint《AWS 物联网开发人员指南》。

示例 2：获取您的 ATS 端点

以下 `describe-endpoint` 示例检索 Amazon Trust Services（ATS）端点。

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

输出：

```
{
    "endpointAddress": "abc123defghijk-ats.iot.us-west-2.amazonaws.com"
}
```

有关更多信息，请参阅《物联网开发者指南》中的 X.509 证书和物 AWSAWS 联网。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DescribeEndpoint中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public static String describeEndpoint(IotClient iotClient) {
        try {
            DescribeEndpointResponse endpointResponse =
iotClient.describeEndpoint(DescribeEndpointRequest.builder().build());

            // Get the endpoint URL.
            String endpointUrl = endpointResponse.endpointAddress();
```

```
            String exString = getValue(endpointUrl);
            String fullEndpoint = "https://"+exString+"-ats.iot.us-
east-1.amazonaws.com";

            System.out.println("Full Endpoint URL: "+fullEndpoint);
            return fullEndpoint;

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "" ;
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DescribeEndpoint中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun describeEndpoint(): String? {
    val request = DescribeEndpointRequest {}
    IotClient { region = "us-east-1" }.use { iotClient ->
        val endpointResponse = iotClient.describeEndpoint(request)
        val endpointUrl: String? = endpointResponse.endpointAddress
        val exString: String = getValue(endpointUrl)
        val fullEndpoint = "https://$exString-ats.iot.us-east-1.amazonaws.com"
        println("Full endpoint URL: $fullEndpoint")
        return fullEndpoint
    }
}
```

- 有关 API 的详细信息，请参阅适用DescribeEndpoint于 K otlin 的AWS SDK API 参考。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
async fn show_address(client: &Client, endpoint_type: &str) -> Result<(), Error>
 {
    let resp = client
        .describe_endpoint()
        .endpoint_type(endpoint_type)
        .send()
        .await?;

    println!("Endpoint address: {}", resp.endpoint_address.unwrap());

    println!();

    Ok(())
}
```

- 有关 API 的详细信息，请参阅适用DescribeEndpoint于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DescribeThing`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DescribeThing`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Describe an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeThing(const Aws::String &thingName,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DescribeThingRequest request;
    request.SetThingName(thingName);

    Aws::IoT::Model::DescribeThingOutcome outcome =
 iotClient.DescribeThing(request);

    if (outcome.IsSuccess()) {
        const Aws::IoT::Model::DescribeThingResult &result = outcome.GetResult();
        std::cout << "Retrieved thing '" << result.GetThingName() << "'" <<
 std::endl;
        std::cout << "thingArn: " << result.GetThingArn() << std::endl;
        std::cout << result.GetAttributes().size() << " attribute(s) retrieved"
                  << std::endl;
        for (const auto &attribute: result.GetAttributes()) {
            std::cout << "  attribute: " << attribute.first << "=" <<
 attribute.second
                      << std::endl;
        }
    }
    else {
        std::cerr << "Error describing thing " << thingName << ": " <<
```

```
                    outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考DescribeThing中的。

CLI

AWS CLI

显示有关事物的详细信息

以下describe-thing示例显示有关在您的 AWS 账户的 AWS IoT 注册表中定义的事物（设备）的信息。

aws iot 描述-thing-thing-name “Bulb” MyLight

输出：

```
{
    "defaultClientId": "MyLightBulb",
    "thingName": "MyLightBulb",
    "thingId": "40da2e73-c6af-406e-b415-15acae538797",
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
    "thingTypeName": "LightBulb",
    "attributes": {
        "model": "123",
        "wattage": "75"
    },
    "version": 1
}
```

有关更多信息，请参阅《AWS IoT 开发人员指南》中的如何使用注册表管理事物。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考DescribeThing中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    private static void describeThing(IotClient iotClient, String thingName) {
        try {
            DescribeThingRequest thingRequest = DescribeThingRequest.builder()
                .thingName(thingName)
                .build() ;

            // Print Thing details.
            DescribeThingResponse describeResponse =
iotClient.describeThing(thingRequest);
            System.out.println("Thing Details:");
            System.out.println("Thing Name: " + describeResponse.thingName());
            System.out.println("Thing ARN: " + describeResponse.thingArn());

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考DescribeThing中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun describeThing(thingNameVal: String) {
    val thingRequest =
        DescribeThingRequest {
            thingName = thingNameVal
        }

    // Print Thing details.
    IotClient { region = "us-east-1" }.use { iotClient ->
        val describeResponse = iotClient.describeThing(thingRequest)
        println("Thing details:")
        println("Thing name: ${describeResponse.thingName}")
        println("Thing ARN:  ${describeResponse.thingArn}")
    }
}
```

- 有关 API 的详细信息，请参阅适用DescribeThing于 K otlin 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`DetachThingPrincipal`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `DetachThingPrincipal`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Detach a principal from an AWS IoT thing.
/*!
  \param principal: A principal to detach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
```

```
 */
bool AwsDoc::IoT::detachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DetachThingPrincipalRequest detachThingPrincipalRequest;
    detachThingPrincipalRequest.SetThingName(thingName);
    detachThingPrincipalRequest.SetPrincipal(principal);

    Aws::IoT::Model::DetachThingPrincipalOutcome outcome =
 iotClient.DetachThingPrincipal(
            detachThingPrincipalRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully detached principal " << principal << " from
 thing "
                  << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to detach principal " << principal << " from thing "
                  << thingName << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考中的DetachThing委托人。

CLI

AWS CLI

将证书/委托人与事物分离

以下detach-thing-principal示例从指定事物中删除代表委托人的证书。

```
aws iot detach-thing-principal \
    --thing-name "MyLightBulb" \
```

```
    --principal "arn:aws:iot:us-
west-2:123456789012:cert/604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36
```

此命令不生成任何输出。

有关更多信息，请参阅《AWS IoT 开发人员指南》中的如何使用注册表管理事物。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》中的 “DetachThing主体”。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
   public static void detachThingPrincipal(IotClient iotClient, String
thingName, String certificateArn){
       try {
           DetachThingPrincipalRequest thingPrincipalRequest =
DetachThingPrincipalRequest.builder()
                .principal(certificateArn)
                .thingName(thingName)
                .build();

           iotClient.detachThingPrincipal(thingPrincipalRequest);
           System.out.println(certificateArn +" was successfully removed from "
+thingName);

       } catch (IotException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考中的DetachThing委托人。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun detachThingPrincipal(
    thingNameVal: String,
    certificateArn: String,
) {
    val thingPrincipalRequest =
        DetachThingPrincipalRequest {
            principal = certificateArn
            thingName = thingNameVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.detachThingPrincipal(thingPrincipalRequest)
        println("$certificateArn was successfully removed from $thingNameVal")
    }
}
```

- 有关 API 的详细信息，请参阅适用于 K otlin AWS 的 SDK 中的DetachThing主体 API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListCertificates`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListCertificates`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! List certificates registered in the AWS account making the call.
/*!
   \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listCertificates(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListCertificatesRequest request;

    Aws::Vector<Aws::IoT::Model::Certificate> allCertificates;
    Aws::String marker; // Used to paginate results.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::IoT::Model::ListCertificatesOutcome outcome =
 iotClient.ListCertificates(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListCertificatesResult &result =
 outcome.GetResult();
            marker = result.GetNextMarker();
            allCertificates.insert(allCertificates.end(),
                                   result.GetCertificates().begin(),
                                   result.GetCertificates().end());
        }
        else {
            std::cerr << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
```

```
            return false;
        }
    } while (!marker.empty());

    std::cout << allCertificates.size() << " certificate(s) found." << std::endl;

    for (auto &certificate: allCertificates) {
        std::cout << "Certificate ID: " << certificate.GetCertificateId() <<
 std::endl;
        std::cout << "Certificate ARN: " << certificate.GetCertificateArn()
                  << std::endl;
        std::cout << std::endl;
    }

    return true;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考ListCertificates中的。

CLI

AWS CLI

示例 1：列出在您的 AWS 账户中注册的证书

以下list-certificates示例列出了在您的账户中注册的所有证书。如果您的分页限制超过默认的 25，则可以使用此命令中的nextMarker响应值并将其提供给下一个命令以获取下一批结果。重复此操作，直到nextMarker返回时没有值。

```
aws iot list-certificates
```

输出：

```
{
    "certificates": [
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36
            "certificateId":
 "604c48437a57b7d5fc5d137c5be75011c6ee67c9a6943683a1acb4b1626bac36",
            "status": "ACTIVE",
```

```
            "creationDate": 1556810537.617
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/262a1ac8a7d8aa72f6e96e365480f7313aa9db74b8339ec65d34dc3074e1c31e
            "certificateId":
 "262a1ac8a7d8aa72f6e96e365480f7313aa9db74b8339ec65d34dc3074e1c31e",
            "status": "ACTIVE",
            "creationDate": 1546447050.885
        },
        {
            "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
b193ab7162c0fadca83246d24fa090300a1236fe58137e121b011804d8ac1d6b",
            "certificateId":
 "b193ab7162c0fadca83246d24fa090300a1236fe58137e121b011804d8ac1d6b",
            "status": "ACTIVE",
            "creationDate": 1546292258.322
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/7aebeea3845d14a44ec80b06b8b78a89f3f8a706974b8b34d18f5adf0741db42
            "certificateId":
 "7aebeea3845d14a44ec80b06b8b78a89f3f8a706974b8b34d18f5adf0741db42",
            "status": "ACTIVE",
            "creationDate": 1541457693.453
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/54458aa39ebb3eb39c91ffbbdcc3a6ca1c7c094d1644b889f735a6fc2cd9a7e3
            "certificateId":
 "54458aa39ebb3eb39c91ffbbdcc3a6ca1c7c094d1644b889f735a6fc2cd9a7e3",
            "status": "ACTIVE",
            "creationDate": 1541113568.611
        },
        {
            "certificateArn": "arn:aws:iot:us-
west-2:123456789012:cert/4f0ba725787aa94d67d2fca420eca022242532e8b3c58e7465c7778b443fd65e
            "certificateId":
 "4f0ba725787aa94d67d2fca420eca022242532e8b3c58e7465c7778b443fd65e",
            "status": "ACTIVE",
            "creationDate": 1541022751.983
        }
    ]
```

```
}
```

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListCertificates中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public static void listCertificates(IotClient iotClient) {
        ListCertificatesResponse response = iotClient.listCertificates();
        List<Certificate> certList = response.certificates();
        for (Certificate cert : certList) {
            System.out.println("Cert id: " + cert.certificateId());
            System.out.println("Cert Arn: " + cert.certificateArn());
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考ListCertificates中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun listCertificates() {
    IotClient { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.listCertificates()
```

```
        val certList = response.certificates
        certList?.forEach { cert ->
            println("Cert id: ${cert.certificateId}")
            println("Cert Arn: ${cert.certificateArn}")
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用ListCertificates于 K otlin 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`ListThings`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `ListThings`。

CLI

AWS CLI

示例 1：列出注册表中的所有事物

以下`list-things`示例列出了在您的 AWS 账户的 AWS IoT 注册表中定义的事物（设备）。

```
aws iot list-things
```

输出：

```
{
    "things": [
        {
            "thingName": "ThirdBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/ThirdBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 2
        },
```

```
        {
            "thingName": "MyOtherLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/
MyOtherLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3
        },
        {
            "thingName": "MyLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1
        },
        {
        "thingName": "SampleIoTThing",
        "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/SampleIoTThing",
        "attributes": {},
        "version": 1
        }
    ]
}
```

示例 2：列出具有特定属性的已定义事物

以下 `list-things` 示例显示了具有名为 `wattage` 的属性的事物列表。

```
aws iot list-things \
    --attribute-name wattage
```

输出：

```
{
    "things": [
        {
            "thingName": "MyLightBulb",
```

```
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 1
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingTypeName": "LightBulb",
            "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/
MyOtherLightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "version": 3
        }
    ]
}
```

有关更多信息，请参阅《AWS IoT 开发人员指南》中的如何使用注册表管理事物。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考ListThings中的。

Rust

适用于 Rust 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
async fn show_things(client: &Client) -> Result<(), Error> {
    let resp = client.list_things().send().await?;

    println!("Things:");

    for thing in resp.things.unwrap() {
```

```
        println!(
            "  Name:  {}",
            thing.thing_name.as_deref().unwrap_or_default()
        );
        println!(
            "  Type:  {}",
            thing.thing_type_name.as_deref().unwrap_or_default()
        );
        println!(
            "  ARN:   {}",
            thing.thing_arn.as_deref().unwrap_or_default()
        );
        println!();
    }

    println!();

    Ok(())
}
```

- 有关 API 的详细信息，请参阅适用ListThings于 Rust 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`SearchIndex`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `SearchIndex`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Query the AWS IoT fleet index.
```

```
//! For query information, see https://docs.aws.amazon.com/iot/latest/
developerguide/query-syntax.html
/*!
  \param: query: The query string.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::searchIndex(const Aws::String &query,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::SearchIndexRequest request;
    request.SetQueryString(query);

    Aws::Vector<Aws::IoT::Model::ThingDocument> allThingDocuments;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::SearchIndexOutcome outcome =
 iotClient.SearchIndex(request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::SearchIndexResult &result =
 outcome.GetResult();
            allThingDocuments.insert(allThingDocuments.end(),
                                     result.GetThings().cbegin(),
                                     result.GetThings().cend());
            nextToken = result.GetNextToken();

        }
        else {
            std::cerr << "Error in SearchIndex: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!nextToken.empty());

    std::cout << allThingDocuments.size() << " thing document(s) found." <<
 std::endl;
```

```
    for (const auto thingDocument: allThingDocuments) {
        std::cout << "  Thing name: " << thingDocument.GetThingName() << "."
                  << std::endl;
    }
    return true;
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考SearchIndex中的。

CLI

AWS CLI

查询事物索引

以下search-index示例在AWS_Things索引中查询类型为的事物LightBulb。

```
aws iot search-index \
    --index-name "AWS_Things" \
    --query-string "thingTypeName:LightBulb"
```

输出：

```
{
    "things": [
        {
            "thingName": "MyLightBulb",
            "thingId": "40da2e73-c6af-406e-b415-15acae538797",
            "thingTypeName": "LightBulb",
            "thingGroupNames": [
                "LightBulbs",
                "DeadBulbs"
            ],
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        },
        {
```

```
            "thingName": "ThirdBulb",
            "thingId": "615c8455-33d5-40e8-95fd-3ee8b24490af",
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        },
        {
            "thingName": "MyOtherLightBulb",
            "thingId": "6dae0d3f-40c1-476a-80c4-1ed24ba6aa11",
            "thingTypeName": "LightBulb",
            "attributes": {
                "model": "123",
                "wattage": "75"
            },
            "connectivity": {
                "connected": false
            }
        }
    ]
}
```

有关更多信息，请参阅《AWS 物联网开发者指南》中的管理事物索引。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考SearchIndex中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
    public static void searchThings(IotClient iotClient, String queryString){
        SearchIndexRequest searchIndexRequest = SearchIndexRequest.builder()
```

```
            .queryString(queryString)
            .build();

        try {
            // Perform the search and get the result.
            SearchIndexResponse searchIndexResponse =
 iotClient.searchIndex(searchIndexRequest);

            // Process the result.
            if (searchIndexResponse.things().isEmpty()) {
                System.out.println("No things found.");
            } else {
                searchIndexResponse.things().forEach(thing ->
 System.out.println("Thing id found using search is " + thing.thingId()));
            }
        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考SearchIndex中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun searchThings(queryStringVal: String?) {
    val searchIndexRequest =
        SearchIndexRequest {
            queryString = queryStringVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        val searchIndexResponse = iotClient.searchIndex(searchIndexRequest)
```

```
        if (searchIndexResponse.things?.isEmpty() == true) {
            println("No things found.")
        } else {
            searchIndexResponse.things
                ?.forEach { thing -> println("Thing id found using search is
 ${thing.thingId}") }
        }
    }
}
```

- 有关 API 的详细信息，请参阅适用SearchIndex于 K otlin 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

`UpdateIndexingConfiguration`与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 `UpdateIndexingConfiguration`。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Update the indexing configuration.
/*!
  \param thingIndexingConfiguration: A ThingIndexingConfiguration object which is
 ignored if not set.
  \param thingGroupIndexingConfiguration: A ThingGroupIndexingConfiguration
 object which is ignored if not set.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateIndexingConfiguration(
```

```
        const Aws::IoT::Model::ThingIndexingConfiguration
 &thingIndexingConfiguration,
        const Aws::IoT::Model::ThingGroupIndexingConfiguration
 &thingGroupIndexingConfiguration,
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::UpdateIndexingConfigurationRequest request;

    if (thingIndexingConfiguration.ThingIndexingModeHasBeenSet()) {
        request.SetThingIndexingConfiguration(thingIndexingConfiguration);
    }

    if (thingGroupIndexingConfiguration.ThingGroupIndexingModeHasBeenSet()) {

 request.SetThingGroupIndexingConfiguration(thingGroupIndexingConfiguration);
    }

    Aws::IoT::Model::UpdateIndexingConfigurationOutcome outcome =
 iotClient.UpdateIndexingConfiguration(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "UpdateIndexingConfiguration succeeded." << std::endl;
    }
    else {
        std::cerr << "UpdateIndexingConfiguration failed."
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考中的UpdateIndexing配置。

CLI

AWS CLI

启用事物索引

以下update-indexing-configuration示例启用事物索引，以支持使用 AWS_Things 索引搜索注册表数据、影子数据和事物连接状态。

```
aws iot update-indexing-configuration
    --thing-indexing-configuration
 thingIndexingMode=REGISTRY_AND_SHADOW,thingConnectivityIndexingMode=STATUS
```

此命令不生成任何输出。

有关更多信息，请参阅《AWS 物联网开发人员指南》中的管理事物索引。

- 有关 API 的详细信息，请参阅《AWS CLI 命令参考》UpdateIndexing中的 “配置”。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

UpdateThing与 AWS SDK 或 CLI 配合使用

以下代码示例演示如何使用 UpdateThing。

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
//! Update an AWS IoT thing with attributes.
/*!
  \param thingName: The name for the thing.
  \param attributeMap: A map of key/value attributes/
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThing(const Aws::String &thingName,
                              const std::map<Aws::String, Aws::String>
 &attributeMap,
```

```
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::UpdateThingRequest request;
    request.SetThingName(thingName);
    Aws::IoT::Model::AttributePayload attributePayload;
    for (const auto &attribute: attributeMap) {
        attributePayload.AddAttributes(attribute.first, attribute.second);
    }
    request.SetAttributePayload(attributePayload);

    Aws::IoT::Model::UpdateThingOutcome outcome = iotClient.UpdateThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to update thing " << thingName << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for C++ API 参考UpdateThing中的。

CLI

AWS CLI

将事物与事物类型关联

以下update-thing示例将 AWS IoT 注册表中的事物与事物类型相关联。建立关联时，您需要为事物类型定义的属性提供值。

```
aws iot update-thing \
    --thing-name "MyOtherLightBulb" \
    --thing-type-name "LightBulb" \
    --attribute-payload "{"attributes": {"wattage":"75", "model":"123"}}"
```

此命令不产生输出。使用describe-thing命令查看结果。

有关更多信息，请参阅《AWS 物联网开发人员指南》中的事物类型。

- 有关 API 的详细信息，请参阅AWS CLI 命令参考UpdateThing中的。

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
public static void updateThing(IotClient iotClient, String thingName) {
    // Specify the new attribute values.
    String newLocation = "Office";
    String newFirmwareVersion = "v2.0";

    Map<String, String> attMap = new HashMap<>();
    attMap.put("location", newLocation);
    attMap.put("firmwareVersion", newFirmwareVersion);

    AttributePayload attributePayload = AttributePayload.builder()
        .attributes(attMap)
        .build();

    UpdateThingRequest updateThingRequest = UpdateThingRequest.builder()
        .thingName(thingName)
        .attributePayload(attributePayload)
        .build();

    try {
        // Update the IoT Thing attributes.
        iotClient.updateThing(updateThingRequest);
        System.out.println("Thing attributes updated successfully.");

    } catch (IotException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
}
```

- 有关 API 的详细信息，请参阅 AWS SDK for Java 2.x API 参考UpdateThing中的。

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
suspend fun updateThing(thingNameVal: String?) {
    val newLocation = "Office"
    val newFirmwareVersion = "v2.0"
    val attMap: MutableMap<String, String> = HashMap()
    attMap["location"] = newLocation
    attMap["firmwareVersion"] = newFirmwareVersion

    val attributePayloadVal =
        AttributePayload {
            attributes = attMap
        }

    val updateThingRequest =
        UpdateThingRequest {
            thingName = thingNameVal
            attributePayload = attributePayloadVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        // Update the IoT thing attributes.
        iotClient.updateThing(updateThingRequest)
        println("$thingNameVal attributes updated successfully.")
    }
}
```

- 有关 API 的详细信息，请参阅适用UpdateThing于 K otlin 的AWS SDK API 参考。

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

AWS IoT 使用 AWS SDK 的场景

以下代码示例向您展示了如何 AWS IoT 使用 AWS 软件开发工具包实现常见场景。这些场景向您展示了如何通过在其中调用多个函数来完成特定任务 AWS IoT。每个场景都包含一个指向的链接 GitHub，您可以在其中找到有关如何设置和运行代码的说明。

示例

- 使用 AWS IoT SDK 处理 AWS IoT 设备、事物和影子

使用 AWS IoT SDK 处理 AWS IoT 设备、事物和影子

以下代码示例展示了如何使用 AWS IoT SDK 处理 AWS IoT 设备管理用例

C++

SDK for C++

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

创造一个 AWS IoT 东西。

```
    Aws::String thingName = askQuestion("Enter a thing name: ");

    if (!createThing(thingName, clientConfiguration)) {
        std::cerr << "Exiting because createThing failed." << std::endl;
        cleanup("", "", "", "", "", false, clientConfiguration);
        return false;
    }
```

```
//! Create an AWS IoT thing.
/*!
```

```
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::CreateThingRequest createThingRequest;
    createThingRequest.SetThingName(thingName);

    Aws::IoT::Model::CreateThingOutcome outcome = iotClient.CreateThing(
            createThingRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to create thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

生成并附加设备证书。

```
    Aws::String certificateARN;
    Aws::String certificateID;
    if (askYesNoQuestion("Would you like to create a certificate for your thing?
 (y/n) ")) {
        Aws::String outputFolder;
        if (askYesNoQuestion(
                "Would you like to save the certificate and keys to file? (y/n)
 ")) {
            outputFolder = std::filesystem::current_path();
            outputFolder += "/device_keys_and_certificates";

            std::filesystem::create_directories(outputFolder);

            std::cout << "The certificate and keys will be saved to the folder: "
                      << outputFolder << std::endl;
        }
```

```
       if (!createKeysAndCertificate(outputFolder, certificateARN,
 certificateID,
                                     clientConfiguration)) {
           std::cerr << "Exiting because createKeysAndCertificate failed."
                     << std::endl;
           cleanup(thingName, "", "", "", "", false, clientConfiguration);
           return false;
       }

       std::cout << "\nNext, the certificate will be attached to the thing.\n"
                 << std::endl;
       if (!attachThingPrincipal(certificateARN, thingName,
 clientConfiguration)) {
           std::cerr << "Exiting because attachThingPrincipal failed." <<
 std::endl;
           cleanup(thingName, certificateARN, certificateID, "", "",
                   false,
                   clientConfiguration);
           return false;
       }
   }
```

```
//! Create keys and certificate for an Aws IoT device.
//! This routine will save certificates and keys to an output folder, if
 provided.
/*!
  \param outputFolder: Location for storing output in files, ignored when string
 is empty.
  \param certificateARNResult: A string to receive the ARN of the created
 certificate.
  \param certificateID: A string to receive the ID of the created certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::createKeysAndCertificate(const Aws::String &outputFolder,
                                           Aws::String &certificateARNResult,
                                           Aws::String &certificateID,
                                           const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
```

```
    Aws::IoT::Model::CreateKeysAndCertificateRequest
 createKeysAndCertificateRequest;

    Aws::IoT::Model::CreateKeysAndCertificateOutcome outcome =
            client.CreateKeysAndCertificate(createKeysAndCertificateRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created a certificate and keys" << std::endl;
        certificateARNResult = outcome.GetResult().GetCertificateArn();
        certificateID = outcome.GetResult().GetCertificateId();
        std::cout << "Certificate ARN: " << certificateARNResult << ",
 certificate ID: "
                  << certificateID << std::endl;

        if (!outputFolder.empty()) {
            std::cout << "Writing certificate and keys to the folder '" <<
 outputFolder
                      << "'." << std::endl;
            std::cout << "Be sure these files are stored securely." << std::endl;

            Aws::String certificateFilePath = outputFolder + "/
certificate.pem.crt";
            std::ofstream certificateFile(certificateFilePath);
            if (!certificateFile.is_open()) {
                std::cerr << "Error opening certificate file, '" <<
 certificateFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            certificateFile << outcome.GetResult().GetCertificatePem();
            certificateFile.close();

            const Aws::IoT::Model::KeyPair &keyPair =
 outcome.GetResult().GetKeyPair();

            Aws::String privateKeyFilePath = outputFolder + "/private.pem.key";
            std::ofstream privateKeyFile(privateKeyFilePath);
            if (!privateKeyFile.is_open()) {
                std::cerr << "Error opening private key file, '" <<
 privateKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
```

```
            privateKeyFile << keyPair.GetPrivateKey();
            privateKeyFile.close();

            Aws::String publicKeyFilePath = outputFolder + "/public.pem.key";
            std::ofstream publicKeyFile(publicKeyFilePath);
            if (!publicKeyFile.is_open()) {
                std::cerr << "Error opening public key file, '" <<
 publicKeyFilePath
                          << "'."
                          << std::endl;
                return false;
            }
            publicKeyFile << keyPair.GetPublicKey();
        }
    }
    else {
        std::cerr << "Error creating keys and certificate: "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Attach a principal to an AWS IoT thing.
/*!
  \param principal: A principal to attach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::attachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient client(clientConfiguration);
    Aws::IoT::Model::AttachThingPrincipalRequest request;
    request.SetPrincipal(principal);
    request.SetThingName(thingName);
    Aws::IoT::Model::AttachThingPrincipalOutcome outcome =
 client.AttachThingPrincipal(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully attached principal to thing." << std::endl;
    }
```

```
    else {
        std::cerr << "Failed to attach principal to thing." <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

对 AWS IoT 事物执行各种操作。

```
   if (!updateThing(thingName, { {"location", "Office"}, {"firmwareVersion",
 "v2.0"} }, clientConfiguration)) {
        std::cerr << "Exiting because updateThing failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
   }

   printAsterisksLine();

   std::cout << "Now an endpoint will be retrieved for your account.\n" <<
 std::endl;
   std::cout << "An IoT Endpoint refers to a specific URL or Uniform Resource
 Locator that serves as the entry point\n"
   << "for communication between IoT devices and the AWS IoT service." <<
 std::endl;

   askQuestion("Press Enter to continue:", alwaysTrueTest);

   Aws::String endpoint;
   if (!describeEndpoint(endpoint, clientConfiguration)) {
       std::cerr << "Exiting because getEndpoint failed." << std::endl;
       cleanup(thingName, certificateARN, certificateID, "", "", false,
               clientConfiguration);
       return false;
   }
   std::cout <<"Your endpoint is " << endpoint << "." << std::endl;
   printAsterisksLine();

   std::cout << "Now the certificates in your account will be listed." <<
 std::endl;
   askQuestion("Press Enter to continue:", alwaysTrueTest);
```

```
    if (!listCertificates(clientConfiguration)) {
        std::cerr << "Exiting because listCertificates failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();

    std::cout << "Now the shadow for the thing will be updated.\n" << std::endl;
    std::cout << "A thing shadow refers to a feature that enables you to create a
 virtual representation, or \"shadow,\"\n"
    << "of a physical device or thing. The thing shadow allows you to synchronize
 and control the state of a device between\n"
    << "the cloud and the device itself. and the AWS IoT service. For example,
 you can write and retrieve JSON data from a thing shadow." << std::endl;
    askQuestion("Press Enter to continue:", alwaysTrueTest);

    if (!updateThingShadow(thingName, R"({"state":{"reported":
{"temperature":25,"humidity":50}}})", clientConfiguration)) {
        std::cerr << "Exiting because updateThingShadow failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();

    std::cout << "Now, the state information for the shadow will be retrieved.\n"
 << std::endl;
    askQuestion("Press Enter to continue:", alwaysTrueTest);

    Aws::String shadowState;
    if (!getThingShadow(thingName, shadowState, clientConfiguration)) {
        std::cerr << "Exiting because getThingShadow failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, "", "", false,
                clientConfiguration);
        return false;
    }
    std::cout << "The retrieved shadow state is: " << shadowState << std::endl;

    printAsterisksLine();
```

```
   std::cout << "A rule with now be added to to the thing.\n" << std::endl;
   std::cout << "Any user who has permission to create rules will be able to
access data processed by the rule." << std::endl;
   std::cout << "In this case, the rule will use an Simple Notification Service
(SNS) topic and an IAM rule." << std::endl;
   std::cout << "These resources will be created using a CloudFormation
template." << std::endl;
   std::cout << "Stack creation may take a few minutes." << std::endl;

   askQuestion("Press Enter to continue: ", alwaysTrueTest);
   Aws::Map<Aws::String, Aws::String> outputs
=createCloudFormationStack(STACK_NAME,clientConfiguration);
   if (outputs.empty()) {
       std::cerr << "Exiting because createCloudFormationStack failed." <<
std::endl;
       cleanup(thingName, certificateARN, certificateID, "", "", false,
               clientConfiguration);
       return false;
   }

   // Retrieve the topic ARN and role ARN from the CloudFormation stack outputs.
   auto topicArnIter = outputs.find(SNS_TOPIC_ARN_OUTPUT);
   auto roleArnIter = outputs.find(ROLE_ARN_OUTPUT);
   if ((topicArnIter == outputs.end()) || (roleArnIter == outputs.end())) {
       std::cerr << "Exiting because output '" << SNS_TOPIC_ARN_OUTPUT <<
       "' or '" << ROLE_ARN_OUTPUT << "'not found in the CloudFormation stack."
<< std::endl;
       cleanup(thingName, certificateARN, certificateID, STACK_NAME, "",
               false,
               clientConfiguration);
       return false;
   }

   Aws::String topicArn = topicArnIter->second;
   Aws::String roleArn = roleArnIter->second;
   Aws::String sqlStatement = "SELECT * FROM '";
   sqlStatement += MQTT_MESSAGE_TOPIC_FILTER;
   sqlStatement += "'";

   printAsterisksLine();

   std::cout << "Now a rule will be created.\n" << std::endl;
   std::cout << "Rules are an administrator-level action. Any user who has
permission\n"
```

```
                 << "to create rules will be able to access data processed by the
 rule." << std::endl;
    std::cout << "In this case, the rule will use an SNS topic" << std::endl;
    std::cout << "and the following SQL statement '" << sqlStatement << "'." <<
 std::endl;
    std::cout << "For more information on IoT SQL, see https://
docs.aws.amazon.com/iot/latest/developerguide/iot-sql-reference.html" <<
 std::endl;
    Aws::String ruleName = askQuestion("Enter a rule name: ");
    if (!createTopicRule(ruleName, topicArn, sqlStatement, roleArn,
 clientConfiguration)) {
        std::cerr << "Exiting because createRule failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, "",
                false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();

    std::cout << "Now your rules will be listed.\n" << std::endl;
    askQuestion("Press Enter to continue: ", alwaysTrueTest);
    if (!listTopicRules(clientConfiguration)) {
        std::cerr << "Exiting because listRules failed." << std::endl;
        cleanup(thingName, certificateARN, certificateID, STACK_NAME, ruleName,
                false,
                clientConfiguration);
        return false;
    }

    printAsterisksLine();
    Aws::String queryString = "thingName:" + thingName;
    std::cout << "Now the AWS IoT fleet index will be queried with the query\n'"
    << queryString << "'.\n" << std::endl;
    std::cout << "For query information, see https://docs.aws.amazon.com/iot/
latest/developerguide/query-syntax.html" << std::endl;

    std::cout << "For this query to work, thing indexing must be enabled in your
 account.\n"
    << "This can be done with the awscli command line by calling 'aws iot update-
indexing-configuration'\n"
       << "or it can be done programmatically." << std::endl;
    std::cout << "For more information, see https://docs.aws.amazon.com/iot/
latest/developerguide/managing-index.html" << std::endl;
```

```
   if (askYesNoQuestion("Do you want to enable thing indexing in your account?
 (y/n) "))
   {
       Aws::IoT::Model::ThingIndexingConfiguration thingIndexingConfiguration;

 thingIndexingConfiguration.SetThingIndexingMode(Aws::IoT::Model::ThingIndexingMode::REGI

 thingIndexingConfiguration.SetThingConnectivityIndexingMode(Aws::IoT::Model::ThingConnec
       // The ThingGroupIndexingConfiguration object is ignored if not set.
       Aws::IoT::Model::ThingGroupIndexingConfiguration
 thingGroupIndexingConfiguration;
       if (!updateIndexingConfiguration(thingIndexingConfiguration,
 thingGroupIndexingConfiguration, clientConfiguration)) {
           std::cerr << "Exiting because updateIndexingConfiguration failed." <<
 std::endl;
           cleanup(thingName, certificateARN, certificateID, STACK_NAME,
                   ruleName, false,
                   clientConfiguration);
           return false;
       }
   }

   if (!searchIndex(queryString, clientConfiguration)) {

       std::cerr << "Exiting because searchIndex failed." << std::endl;
       cleanup(thingName, certificateARN, certificateID, STACK_NAME, ruleName,
               false,
               clientConfiguration);
       return false;
   }
```

```
//! Update an AWS IoT thing with attributes.
/*!
  \param thingName: The name for the thing.
  \param attributeMap: A map of key/value attributes/
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThing(const Aws::String &thingName,
                              const std::map<Aws::String, Aws::String>
 &attributeMap,
```

```
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::UpdateThingRequest request;
    request.SetThingName(thingName);
    Aws::IoT::Model::AttributePayload attributePayload;
    for (const auto &attribute: attributeMap) {
        attributePayload.AddAttributes(attribute.first, attribute.second);
    }
    request.SetAttributePayload(attributePayload);

    Aws::IoT::Model::UpdateThingOutcome outcome = iotClient.UpdateThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to update thing " << thingName << ":" <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Describe the endpoint specific to the AWS account making the call.
/*!
  \param endpointResult: String to receive the endpoint result.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::describeEndpoint(Aws::String &endpointResult,
                                   const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::String endpoint;
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DescribeEndpointRequest describeEndpointRequest;
    describeEndpointRequest.SetEndpointType(
            "iot:Data-ATS"); // Recommended endpoint type.

    Aws::IoT::Model::DescribeEndpointOutcome outcome =
 iotClient.DescribeEndpoint(
            describeEndpointRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully described endpoint." << std::endl;
```

```
        endpointResult = outcome.GetResult().GetEndpointAddress();
    }
    else {
        std::cerr << "Error describing endpoint" <<
 outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}

//! List certificates registered in the AWS account making the call.
/*!
   \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listCertificates(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListCertificatesRequest request;

    Aws::Vector<Aws::IoT::Model::Certificate> allCertificates;
    Aws::String marker; // Used to paginate results.
    do {
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::IoT::Model::ListCertificatesOutcome outcome =
 iotClient.ListCertificates(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListCertificatesResult &result =
 outcome.GetResult();
            marker = result.GetNextMarker();
            allCertificates.insert(allCertificates.end(),
                                   result.GetCertificates().begin(),
                                   result.GetCertificates().end());
        }
        else {
            std::cerr << "Error: " << outcome.GetError().GetMessage() <<
 std::endl;
            return false;
```

```
        }
    } while (!marker.empty());

    std::cout << allCertificates.size() << " certificate(s) found." << std::endl;

    for (auto &certificate: allCertificates) {
        std::cout << "Certificate ID: " << certificate.GetCertificateId() <<
 std::endl;
        std::cout << "Certificate ARN: " << certificate.GetCertificateArn()
                  << std::endl;
        std::cout << std::endl;
    }

    return true;
}

//! Update the shadow of an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param document: The state information, in JSON format.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::updateThingShadow(const Aws::String &thingName,
                                    const Aws::String &document,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoTDataPlane::IoTDataPlaneClient
 iotDataPlaneClient(clientConfiguration);
    Aws::IoTDataPlane::Model::UpdateThingShadowRequest updateThingShadowRequest;
    updateThingShadowRequest.SetThingName(thingName);
    std::shared_ptr<std::stringstream> streamBuf =
 std::make_shared<std::stringstream>(
            document);
    updateThingShadowRequest.SetBody(streamBuf);
    Aws::IoTDataPlane::Model::UpdateThingShadowOutcome outcome =
 iotDataPlaneClient.UpdateThingShadow(
            updateThingShadowRequest);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully updated thing shadow." << std::endl;
    }
    else {
        std::cerr << "Error while updating thing shadow."
                  << outcome.GetError().GetMessage() << std::endl;
```

```
    }

    return outcome.IsSuccess();
}

//! Get the shadow of an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param documentResult: String to receive the state information, in JSON format.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::getThingShadow(const Aws::String &thingName,
                                 Aws::String &documentResult,
                                 const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoTDataPlane::IoTDataPlaneClient iotClient(clientConfiguration);
    Aws::IoTDataPlane::Model::GetThingShadowRequest request;
    request.SetThingName(thingName);
    auto outcome = iotClient.GetThingShadow(request);
    if (outcome.IsSuccess()) {
        std::stringstream ss;
        ss << outcome.GetResult().GetPayload().rdbuf();
        documentResult = ss.str();
    }
    else {
        std::cerr << "Error getting thing shadow: " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Create an AWS IoT rule with an SNS topic as the target.
/*!
  \param ruleName: The name for the rule.
  \param snsTopic: The SNS topic ARN for the action.
  \param sql: The SQL statement used to query the topic.
  \param roleARN: The IAM role ARN for the action.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool
AwsDoc::IoT::createTopicRule(const Aws::String &ruleName,
```

```
                                const Aws::String &snsTopicARN, const Aws::String
 &sql,
                                const Aws::String &roleARN,
                                const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::CreateTopicRuleRequest request;
    request.SetRuleName(ruleName);

    Aws::IoT::Model::SnsAction snsAction;
    snsAction.SetTargetArn(snsTopicARN);
    snsAction.SetRoleArn(roleARN);

    Aws::IoT::Model::Action action;
    action.SetSns(snsAction);

    Aws::IoT::Model::TopicRulePayload topicRulePayload;
    topicRulePayload.SetSql(sql);
    topicRulePayload.SetActions({action});

    request.SetTopicRulePayload(topicRulePayload);
    auto outcome = iotClient.CreateTopicRule(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully created topic rule " << ruleName << "." <<
 std::endl;
    }
    else {
        std::cerr << "Error creating topic rule " << ruleName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }
    return outcome.IsSuccess();
}

//! Lists the AWS IoT topic rules.
/*!
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::listTopicRules(
        const Aws::Client::ClientConfiguration &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::ListTopicRulesRequest request;
```

```
    Aws::Vector<Aws::IoT::Model::TopicRuleListItem> allRules;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::ListTopicRulesOutcome outcome =
 iotClient.ListTopicRules(
                request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::ListTopicRulesResult &result =
 outcome.GetResult();
            allRules.insert(allRules.end(),
                            result.GetRules().cbegin(),
                            result.GetRules().cend());

            nextToken = result.GetNextToken();
        }
        else {
            std::cerr << "ListTopicRules error: " <<
                      outcome.GetError().GetMessage() << std::endl;
            return false;
        }

    } while (!nextToken.empty());

    std::cout << "ListTopicRules: " << allRules.size() << " rule(s) found."
              << std::endl;
    for (auto &rule: allRules) {
        std::cout << "  Rule name: " << rule.GetRuleName() << ", rule ARN: "
                  << rule.GetRuleArn() << "." << std::endl;
    }

    return true;
}

//! Query the AWS IoT fleet index.
//! For query information, see https://docs.aws.amazon.com/iot/latest/
developerguide/query-syntax.html
/*!
  \param: query: The query string.
  \param clientConfiguration: AWS client configuration.
```

```
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::searchIndex(const Aws::String &query,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::SearchIndexRequest request;
    request.SetQueryString(query);

    Aws::Vector<Aws::IoT::Model::ThingDocument> allThingDocuments;
    Aws::String nextToken; // Used for pagination.
    do {
        if (!nextToken.empty()) {
            request.SetNextToken(nextToken);
        }

        Aws::IoT::Model::SearchIndexOutcome outcome =
 iotClient.SearchIndex(request);

        if (outcome.IsSuccess()) {
            const Aws::IoT::Model::SearchIndexResult &result =
 outcome.GetResult();
            allThingDocuments.insert(allThingDocuments.end(),
                                     result.GetThings().cbegin(),
                                     result.GetThings().cend());
            nextToken = result.GetNextToken();

        }
        else {
            std::cerr << "Error in SearchIndex: " <<
 outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!nextToken.empty());

    std::cout << allThingDocuments.size() << " thing document(s) found." <<
 std::endl;
    for (const auto thingDocument: allThingDocuments) {
        std::cout << "  Thing name: " << thingDocument.GetThingName() << "."
                  << std::endl;
    }
    return true;
```

```
}
```

清理资源。

```
bool
AwsDoc::IoT::cleanup(const Aws::String &thingName, const Aws::String
 &certificateARN,
                     const Aws::String &certificateID, const Aws::String
 &stackName,
                     const Aws::String &ruleName, bool askForConfirmation,
                     const Aws::Client::ClientConfiguration &clientConfiguration)
 {
    bool result = true;

    if (!ruleName.empty() && (!askForConfirmation ||
                              askYesNoQuestion("Delete the rule '" + ruleName +
                                               "'? (y/n) "))) {
        result &= deleteTopicRule(ruleName, clientConfiguration);
    }

    Aws::CloudFormation::CloudFormationClient
 cloudFormationClient(clientConfiguration);

    if (!stackName.empty() && (!askForConfirmation ||
                               askYesNoQuestion(
                                       "Delete the CloudFormation stack '" +
 stackName +
                                       "'? (y/n) "))) {
        result &= deleteStack(stackName, clientConfiguration);
    }

    if (!certificateARN.empty() && (!askForConfirmation ||
                                    askYesNoQuestion("Delete the certificate '" +
                                                     certificateARN + "'? (y/n)
 "))) {
        result &= detachThingPrincipal(certificateARN, thingName,
 clientConfiguration);
        result &= deleteCertificate(certificateID, clientConfiguration);
    }

    if (!thingName.empty() && (!askForConfirmation ||
```

```
                                askYesNoQuestion("Delete the thing '" + thingName
 +
                                                 "'? (y/n) "))) {
        result &= deleteThing(thingName, clientConfiguration);
    }

    return result;
}
```

```
//! Detach a principal from an AWS IoT thing.
/*!
  \param principal: A principal to detach.
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::detachThingPrincipal(const Aws::String &principal,
                                       const Aws::String &thingName,
                                       const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DetachThingPrincipalRequest detachThingPrincipalRequest;
    detachThingPrincipalRequest.SetThingName(thingName);
    detachThingPrincipalRequest.SetPrincipal(principal);

    Aws::IoT::Model::DetachThingPrincipalOutcome outcome =
 iotClient.DetachThingPrincipal(
            detachThingPrincipalRequest);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully detached principal " << principal << " from
 thing "
                  << thingName << std::endl;
    }
    else {
        std::cerr << "Failed to detach principal " << principal << " from thing "
                  << thingName << ": "
                  << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
```

```
}

//! Delete a certificate.
/*!
  \param certificateID: The ID of a certificate.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteCertificate(const Aws::String &certificateID,
                                    const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);

    Aws::IoT::Model::DeleteCertificateRequest request;
    request.SetCertificateId(certificateID);

    Aws::IoT::Model::DeleteCertificateOutcome outcome =
 iotClient.DeleteCertificate(
            request);

    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted certificate " << certificateID <<
 std::endl;
    }
    else {
        std::cerr << "Error deleting certificate " << certificateID << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Delete an AWS IoT rule.
/*!
  \param ruleName: The name for the rule.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteTopicRule(const Aws::String &ruleName,
                                  const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteTopicRuleRequest request;
    request.SetRuleName(ruleName);
```

```
    Aws::IoT::Model::DeleteTopicRuleOutcome outcome = iotClient.DeleteTopicRule(
            request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted rule " << ruleName << std::endl;
    }
    else {
        std::cerr << "Failed to delete rule " << ruleName <<
                  ": " << outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}

//! Delete an AWS IoT thing.
/*!
  \param thingName: The name for the thing.
  \param clientConfiguration: AWS client configuration.
  \return bool: Function succeeded.
 */
bool AwsDoc::IoT::deleteThing(const Aws::String &thingName,
                              const Aws::Client::ClientConfiguration
 &clientConfiguration) {
    Aws::IoT::IoTClient iotClient(clientConfiguration);
    Aws::IoT::Model::DeleteThingRequest request;
    request.SetThingName(thingName);
    const auto outcome = iotClient.DeleteThing(request);
    if (outcome.IsSuccess()) {
        std::cout << "Successfully deleted thing " << thingName << std::endl;
    }
    else {
        std::cerr << "Error deleting thing " << thingName << ": " <<
                  outcome.GetError().GetMessage() << std::endl;
    }

    return outcome.IsSuccess();
}
```

Java

适用于 Java 2.x 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.iot.IotClient;
import software.amazon.awssdk.services.iot.model.Action;
import software.amazon.awssdk.services.iot.model.AttachThingPrincipalRequest;
import software.amazon.awssdk.services.iot.model.AttachThingPrincipalResponse;
import software.amazon.awssdk.services.iot.model.AttributePayload;
import software.amazon.awssdk.services.iot.model.Certificate;
import
 software.amazon.awssdk.services.iot.model.CreateKeysAndCertificateResponse;
import software.amazon.awssdk.services.iot.model.CreateThingRequest;
import software.amazon.awssdk.services.iot.model.CreateTopicRuleRequest;
import software.amazon.awssdk.services.iot.model.DeleteCertificateRequest;
import software.amazon.awssdk.services.iot.model.CreateThingResponse;
import software.amazon.awssdk.services.iot.model.DeleteThingRequest;
import software.amazon.awssdk.services.iot.model.DescribeEndpointRequest;
import software.amazon.awssdk.services.iot.model.DescribeEndpointResponse;
import software.amazon.awssdk.services.iot.model.DescribeThingRequest;
import software.amazon.awssdk.services.iot.model.DescribeThingResponse;
import software.amazon.awssdk.services.iot.model.DetachThingPrincipalRequest;
import software.amazon.awssdk.services.iot.model.IotException;
import software.amazon.awssdk.services.iot.model.ListCertificatesResponse;
import software.amazon.awssdk.services.iot.model.ListTopicRulesRequest;
import software.amazon.awssdk.services.iot.model.ListTopicRulesResponse;
import software.amazon.awssdk.services.iot.model.SearchIndexRequest;
import software.amazon.awssdk.services.iot.model.SearchIndexResponse;
import software.amazon.awssdk.services.iot.model.SnsAction;
import software.amazon.awssdk.services.iot.model.TopicRuleListItem;
import software.amazon.awssdk.services.iot.model.TopicRulePayload;
import software.amazon.awssdk.services.iot.model.UpdateThingRequest;
import software.amazon.awssdk.services.iotdataplane.IotDataPlaneClient;
import software.amazon.awssdk.services.iotdataplane.model.GetThingShadowRequest;
```

```
import software.amazon.awssdk.services.iotdataplane.model.GetThingShadowResponse;
import
 software.amazon.awssdk.services.iotdataplane.model.UpdateThingShadowRequest;
import java.net.URI;
import java.nio.charset.StandardCharsets;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This Java example performs these tasks:
 *
 * 1. Creates an AWS IoT Thing.
 * 2. Generate and attach a device certificate.
 * 3. Update an AWS IoT Thing with Attributes.
 * 4. Get an AWS IoT Endpoint.
 * 5. List your certificates.
 * 6. Updates the shadow for the specified thing..
 * 7. Write out the state information, in JSON format
 * 8. Creates a rule
 * 9. List rules
 * 10. Search things
 * 11. Detach amd delete the certificate.
 * 12. Delete Thing.
 */
public class IotScenario {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");
    private static final String TOPIC = "your-iot-topic";
    public static void main(String[] args) {
        final String usage =
            """
                Usage:
```

```
                    <roleARN> <snsAction>

                Where:
                    roleARN - The ARN of an IAM role that has permission to work
with AWS IOT.
                    snsAction  - An ARN of an SNS topic.
                """;

      if (args.length != 2) {
          System.out.println(usage);
          System.exit(1);
      }

       String thingName;
       String ruleName;
       String roleARN = args[0];
       String snsAction = args[1];
       Scanner scanner = new Scanner(System.in);
       IotClient iotClient = IotClient.builder()
           .region(Region.US_EAST_1)
           .build();

       System.out.println(DASHES);
       System.out.println("Welcome to the AWS IoT example workflow.");
       System.out.println("""
           This example program demonstrates various interactions with the AWS
Internet of Things (IoT) Core service. The program guides you through a series
of steps,
           including creating an IoT Thing, generating a device certificate,
updating the Thing with attributes, and so on.
           It utilizes the AWS SDK for Java V2 and incorporates functionality
for creating and managing IoT Things, certificates, rules,
           shadows, and performing searches. The program aims to showcase AWS
IoT capabilities and provides a comprehensive example for
           developers working with AWS IoT in a Java environment.

           """);
       System.out.print("Press Enter to continue...");
       scanner.nextLine();
       System.out.println(DASHES);

       System.out.println(DASHES);
       System.out.println("1. Create an AWS IoT Thing.");
       System.out.println("""
```

```
            An AWS IoT Thing represents a virtual entity in the AWS IoT service
that can be associated with a physical device.
            """);
        // Prompt the user for input.
        System.out.print("Enter Thing name: ");
        thingName = scanner.nextLine();
        createIoTThing(iotClient, thingName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Generate a device certificate.");
        System.out.println("""
            A device certificate performs a role in securing the communication
between devices (Things) and the AWS IoT platform.
            """);

        System.out.print("Do you want to create a certificate for " +thingName
+"? (y/n)");
        String certAns = scanner.nextLine();
        String certificateArn="" ;
        if (certAns != null && certAns.trim().equalsIgnoreCase("y")) {
            certificateArn = createCertificate(iotClient);
            System.out.println("Attach the certificate to the AWS IoT Thing.");
            attachCertificateToThing(iotClient, thingName, certificateArn);
        } else {
            System.out.println("A device certificate was not created.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Update an AWS IoT Thing with Attributes.");
        System.out.println("""
             IoT Thing attributes, represented as key-value pairs, offer a
pivotal advantage in facilitating efficient data
             management and retrieval within the AWS IoT ecosystem.
            """);
        System.out.print("Press Enter to continue...");
        scanner.nextLine();
        updateThing(iotClient, thingName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Return a unique endpoint specific to the Amazon
Web Services account.");
```

```
        System.out.println("""
            An IoT Endpoint refers to a specific URL or Uniform Resource Locator
that serves as the entry point for communication between IoT devices and the AWS
IoT service.
           """);
        System.out.print("Press Enter to continue...");
        scanner.nextLine();
        String endpointUrl = describeEndpoint(iotClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. List your AWS IoT certificates");
        System.out.print("Press Enter to continue...");
        scanner.nextLine();
        if (certificateArn.length() > 0) {
            listCertificates(iotClient);
        } else {
            System.out.println("You did not create a certificates. Skipping this
step.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Create an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
        System.out.println("""
            A Thing Shadow refers to a feature that enables you to create a
virtual representation, or "shadow,"
            of a physical device or thing. The Thing Shadow allows you to
synchronize and control the state of a device between
            the cloud and the device itself. and the AWS IoT service. For
example, you can write and retrieve JSON data from a Thing Shadow.
           """);
        System.out.print("Press Enter to continue...");
        scanner.nextLine();
        IotDataPlaneClient iotPlaneClient = IotDataPlaneClient.builder()
            .region(Region.US_EAST_1)
            .endpointOverride(URI.create(endpointUrl))
            .build();

        updateShadowThing(iotPlaneClient, thingName);
        System.out.println(DASHES);

        System.out.println(DASHES);
```

```
        System.out.println("7. Write out the state information, in JSON
format.");
        System.out.print("Press Enter to continue...");
        scanner.nextLine();
        getPayload(iotPlaneClient, thingName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Creates a rule");
        System.out.println("""
        Creates a rule that is an administrator-level action.
        Any user who has permission to create rules will be able to access data
processed by the rule.
        """);
        System.out.print("Enter Rule name: ");
        ruleName = scanner.nextLine();
        createIoTRule(iotClient, roleARN, ruleName, snsAction);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. List your rules.");
        System.out.print("Press Enter to continue...");
        scanner.nextLine();
        listIoTRules(iotClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Search things using the Thing name.");
        System.out.print("Press Enter to continue...");
        scanner.nextLine();
        String queryString = "thingName:"+thingName ;
        searchThings(iotClient, queryString);
        System.out.println(DASHES);

        System.out.println(DASHES);
        if (certificateArn.length() > 0) {
            System.out.print("Do you want to detach and delete the certificate
for " +thingName +"? (y/n)");
            String delAns = scanner.nextLine();
            if (delAns != null && delAns.trim().equalsIgnoreCase("y")) {
                System.out.println("11. You selected to detach amd delete the
certificate.");
                System.out.print("Press Enter to continue...");
                scanner.nextLine();
```

```
                detachThingPrincipal(iotClient, thingName, certificateArn);
                deleteCertificate(iotClient, certificateArn);
            } else {
                System.out.println("11. You selected not to delete the
certificate.");
            }
        } else {
            System.out.println("11. You did not create a certificate so there is
nothing to delete.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Delete the AWS IoT Thing.");
        System.out.print("Do you want to delete the IoT Thing? (y/n)");
        String delAns = scanner.nextLine();
        if (delAns != null && delAns.trim().equalsIgnoreCase("y")) {
            deleteIoTThing(iotClient, thingName);
        } else {
            System.out.println("The IoT Thing was not deleted.");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The AWS IoT workflow has successfully completed.");
        System.out.println(DASHES);
    }

    public static void listCertificates(IotClient iotClient) {
        ListCertificatesResponse response = iotClient.listCertificates();
        List<Certificate> certList = response.certificates();
        for (Certificate cert : certList) {
            System.out.println("Cert id: " + cert.certificateId());
            System.out.println("Cert Arn: " + cert.certificateArn());
        }
    }

    public static void listIoTRules(IotClient iotClient) {
        try {
            ListTopicRulesRequest listTopicRulesRequest =
ListTopicRulesRequest.builder().build();
            ListTopicRulesResponse listTopicRulesResponse =
iotClient.listTopicRules(listTopicRulesRequest);
            System.out.println("List of IoT Rules:");
```

```
            List<TopicRuleListItem> ruleList = listTopicRulesResponse.rules();
            for (TopicRuleListItem rule : ruleList) {
                System.out.println("Rule Name: " + rule.ruleName());
                System.out.println("Rule ARN: " + rule.ruleArn());
                System.out.println("--------------");
            }

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void createIoTRule(IotClient iotClient, String roleARN, String
ruleName, String action) {
        try {
            String sql = "SELECT * FROM '" + TOPIC + "'";
            SnsAction action1 = SnsAction.builder()
                .targetArn(action)
                .roleArn(roleARN)
                .build();

            // Create the action.
            Action myAction = Action.builder()
                .sns(action1)
                .build();

            // Create the topic rule payload.
            TopicRulePayload topicRulePayload = TopicRulePayload.builder()
                .sql(sql)
                .actions(myAction)
                .build();

            // Create the topic rule request.
            CreateTopicRuleRequest topicRuleRequest =
CreateTopicRuleRequest.builder()
                .ruleName(ruleName)
                .topicRulePayload(topicRulePayload)
                .build();

            // Create the rule.
            iotClient.createTopicRule(topicRuleRequest);
            System.out.println("IoT Rule created successfully.");
```

```
        } catch (IotException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void getPayload(IotDataPlaneClient iotPlaneClient, String
thingName) {
        try {
            GetThingShadowRequest getThingShadowRequest =
GetThingShadowRequest.builder()
                .thingName(thingName)
                .build();

            GetThingShadowResponse getThingShadowResponse =
iotPlaneClient.getThingShadow(getThingShadowRequest);

            // Extracting payload from response.
            SdkBytes payload = getThingShadowResponse.payload();
            String payloadString = payload.asUtf8String();
            System.out.println("Received Shadow Data: " + payloadString);

        } catch (IotException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static void updateShadowThing(IotDataPlaneClient iotPlaneClient,
String thingName) {
        try {
            // Create Thing Shadow State Document.
            String stateDocument = "{\"state\":{\"reported\":{\"temperature\":25,
\"humidity\":50}}}";
            SdkBytes data= SdkBytes.fromString(stateDocument,
StandardCharsets.UTF_8 );
            UpdateThingShadowRequest updateThingShadowRequest =
UpdateThingShadowRequest.builder()
                .thingName(thingName)
                .payload(data)
                .build();

            // Update Thing Shadow.
            iotPlaneClient.updateThingShadow(updateThingShadowRequest);
```

```
            System.out.println("Thing Shadow updated successfully.");

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void updateThing(IotClient iotClient, String thingName) {
        // Specify the new attribute values.
        String newLocation = "Office";
        String newFirmwareVersion = "v2.0";

        Map<String, String> attMap = new HashMap<>();
        attMap.put("location", newLocation);
        attMap.put("firmwareVersion", newFirmwareVersion);

        AttributePayload attributePayload = AttributePayload.builder()
            .attributes(attMap)
            .build();

        UpdateThingRequest updateThingRequest = UpdateThingRequest.builder()
            .thingName(thingName)
            .attributePayload(attributePayload)
            .build();

        try {
            // Update the IoT Thing attributes.
            iotClient.updateThing(updateThingRequest);
            System.out.println("Thing attributes updated successfully.");

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static String describeEndpoint(IotClient iotClient) {
        try {
            DescribeEndpointResponse endpointResponse =
iotClient.describeEndpoint(DescribeEndpointRequest.builder().build());

            // Get the endpoint URL.
            String endpointUrl = endpointResponse.endpointAddress();
```

```
            String exString = getValue(endpointUrl);
            String fullEndpoint = "https://"+exString+"-ats.iot.us-
east-1.amazonaws.com";

            System.out.println("Full Endpoint URL: "+fullEndpoint);
            return fullEndpoint;

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "" ;
    }

    public static void detachThingPrincipal(IotClient iotClient, String
 thingName, String certificateArn){
        try {
            DetachThingPrincipalRequest thingPrincipalRequest =
 DetachThingPrincipalRequest.builder()
                .principal(certificateArn)
                .thingName(thingName)
                .build();

            iotClient.detachThingPrincipal(thingPrincipalRequest);
            System.out.println(certificateArn +" was successfully removed from "
 +thingName);

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteCertificate(IotClient iotClient, String
 certificateArn ) {
        DeleteCertificateRequest certificateProviderRequest =
 DeleteCertificateRequest.builder()
            .certificateId(extractCertificateId(certificateArn))
            .build();

        iotClient.deleteCertificate(certificateProviderRequest);
        System.out.println(certificateArn +" was successfully deleted.");
    }
```

```
   // Get the cert Id  from the Cert ARN value.
   private static String extractCertificateId(String certificateArn) {
       // Example ARN: arn:aws:iot:region:account-id:cert/certificate-id.
       String[] arnParts = certificateArn.split(":");
       String certificateIdPart = arnParts[arnParts.length - 1];
       return certificateIdPart.substring(certificateIdPart.lastIndexOf("/") +
1);
   }

   public static String createCertificate(IotClient iotClient) {
       try {
           CreateKeysAndCertificateResponse response =
iotClient.createKeysAndCertificate();
           String certificatePem = response.certificatePem();
           String certificateArn = response.certificateArn();

           // Print the details.
           System.out.println("\nCertificate:");
           System.out.println(certificatePem);
           System.out.println("\nCertificate ARN:");
           System.out.println(certificateArn);
           return certificateArn;

       } catch (IotException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }

       return "";
   }

   public static void attachCertificateToThing(IotClient iotClient, String
thingName, String certificateArn) {
       // Attach the certificate to the thing.
       AttachThingPrincipalRequest principalRequest =
AttachThingPrincipalRequest.builder()
           .thingName(thingName)
           .principal(certificateArn)
           .build();

       AttachThingPrincipalResponse attachResponse =
iotClient.attachThingPrincipal(principalRequest);

       // Verify the attachment was successful.
```

```
        if (attachResponse.sdkHttpResponse().isSuccessful()) {
            System.out.println("Certificate attached to Thing successfully.");

            // Print additional information about the Thing.
            describeThing(iotClient, thingName);
        } else {
            System.err.println("Failed to attach certificate to Thing. HTTP
Status Code: " +
                attachResponse.sdkHttpResponse().statusCode());
        }
    }

    private static void describeThing(IotClient iotClient, String thingName) {
        try {
            DescribeThingRequest thingRequest = DescribeThingRequest.builder()
                .thingName(thingName)
                .build() ;

            // Print Thing details.
            DescribeThingResponse describeResponse =
iotClient.describeThing(thingRequest);
            System.out.println("Thing Details:");
            System.out.println("Thing Name: " + describeResponse.thingName());
            System.out.println("Thing ARN: " + describeResponse.thingArn());

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void deleteIoTThing(IotClient iotClient, String thingName) {
        try {
            DeleteThingRequest deleteThingRequest = DeleteThingRequest.builder()
                .thingName(thingName)
                .build();

            iotClient.deleteThing(deleteThingRequest);
            System.out.println("Deleted Thing " + thingName);

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
```

```
    }

    public static void createIoTThing(IotClient iotClient, String thingName) {
        try {
            CreateThingRequest createThingRequest = CreateThingRequest.builder()
                .thingName(thingName)
                .build();

            CreateThingResponse createThingResponse =
 iotClient.createThing(createThingRequest);
            System.out.println(thingName +" was successfully created. The ARN
 value is " + createThingResponse.thingArn());

        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    private static String getValue(String input) {
        // Define a regular expression pattern for extracting the subdomain.
        Pattern pattern = Pattern.compile("^(.*?)\\.iot\\.us-east-1\\.amazonaws\
\.com");

        // Match the pattern against the input string.
        Matcher matcher = pattern.matcher(input);

        // Check if a match is found.
        if (matcher.find()) {
            // Extract the subdomain from the first capturing group.
            String subdomain = matcher.group(1);
            System.out.println("Extracted subdomain: " + subdomain);
            return subdomain ;
        } else {
            System.out.println("No match found");
        }
        return "" ;
    }

    public static void searchThings(IotClient iotClient, String queryString){
        SearchIndexRequest searchIndexRequest = SearchIndexRequest.builder()
            .queryString(queryString)
            .build();
```

```
        try {
            // Perform the search and get the result.
            SearchIndexResponse searchIndexResponse =
 iotClient.searchIndex(searchIndexRequest);

            // Process the result.
            if (searchIndexResponse.things().isEmpty()) {
                System.out.println("No things found.");
            } else {
                searchIndexResponse.things().forEach(thing ->
 System.out.println("Thing id found using search is " + thing.thingId()));
            }
        } catch (IotException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

Kotlin

适用于 Kotlin 的 SDK

> **Note**
>
> 还有更多相关信息 GitHub。在 AWS 代码示例存储库中查找完整示例，了解如何进行设置和运行。

```
import aws.sdk.kotlin.services.iot.IotClient
import aws.sdk.kotlin.services.iot.model.Action
import aws.sdk.kotlin.services.iot.model.AttachThingPrincipalRequest
import aws.sdk.kotlin.services.iot.model.AttributePayload
import aws.sdk.kotlin.services.iot.model.CreateThingRequest
import aws.sdk.kotlin.services.iot.model.CreateTopicRuleRequest
import aws.sdk.kotlin.services.iot.model.DeleteCertificateRequest
import aws.sdk.kotlin.services.iot.model.DeleteThingRequest
import aws.sdk.kotlin.services.iot.model.DescribeEndpointRequest
import aws.sdk.kotlin.services.iot.model.DescribeThingRequest
import aws.sdk.kotlin.services.iot.model.DetachThingPrincipalRequest
```

```
import aws.sdk.kotlin.services.iot.model.ListTopicRulesRequest
import aws.sdk.kotlin.services.iot.model.SearchIndexRequest
import aws.sdk.kotlin.services.iot.model.SnsAction
import aws.sdk.kotlin.services.iot.model.TopicRulePayload
import aws.sdk.kotlin.services.iot.model.UpdateThingRequest
import aws.sdk.kotlin.services.iotdataplane.IotDataPlaneClient
import aws.sdk.kotlin.services.iotdataplane.model.GetThingShadowRequest
import aws.sdk.kotlin.services.iotdataplane.model.UpdateThingShadowRequest
import aws.smithy.kotlin.runtime.content.ByteStream
import aws.smithy.kotlin.runtime.content.toByteArray
import java.util.Scanner
import java.util.regex.Pattern
import kotlin.system.exitProcess

/**
 * Before running this Kotlin code example, ensure that your development
 environment
 * is set up, including configuring your credentials.
 *
 * For detailed instructions, refer to the following documentation topic:
 * [Setting Up Your Development Environment](https://docs.aws.amazon.com/sdk-for-
kotlin/latest/developer-guide/setup.html)
 *
 * This code example requires an SNS topic and an IAM Role.
 * Follow the steps in the documentation to set up these resources:
 *
 * - [Creating an SNS Topic](https://docs.aws.amazon.com/sns/latest/dg/sns-
getting-started.html#step-create-topic)
 * - [Creating an IAM Role](https://docs.aws.amazon.com/IAM/latest/UserGuide/
id_roles_create.html)
 */

val DASHES = String(CharArray(80)).replace("\u0000", "-")
val TOPIC = "your-iot-topic"

suspend fun main(args: Array<String>) {
    val usage =
        """
        Usage:
            <roleARN> <snsAction>

        Where:
            roleARN - The ARN of an IAM role that has permission to work with AWS
 IOT.
```

```
            snsAction  - An ARN of an SNS topic.

        """.trimIndent()

    if (args.size != 2) {
        println(usage)
        exitProcess(1)
    }

    var thingName: String
    val roleARN = args[0]
    val snsAction = args[1]
    val scanner = Scanner(System.`in`)

    println(DASHES)
    println("Welcome to the AWS IoT example scenario.")
    println(
        """
        This example program demonstrates various interactions with the AWS
Internet of Things (IoT) Core service.
        The program guides you through a series of steps, including creating an
IoT thing, generating a device certificate,
        updating the thing with attributes, and so on.

        It utilizes the AWS SDK for Kotlin and incorporates functionality for
creating and managing IoT things, certificates, rules,
        shadows, and performing searches. The program aims to showcase AWS IoT
capabilities and provides a comprehensive example for
        developers working with AWS IoT in a Kotlin environment.
        """.trimIndent(),
    )

    print("Press Enter to continue...")
    scanner.nextLine()
    println(DASHES)

    println(DASHES)
    println("1. Create an AWS IoT thing.")
    println(
        """
        An AWS IoT thing represents a virtual entity in the AWS IoT service that
can be associated with a physical device.
        """.trimIndent(),
    )
```

```
    // Prompt the user for input.
    print("Enter thing name: ")
    thingName = scanner.nextLine()
    createIoTThing(thingName)
    describeThing(thingName)
    println(DASHES)

    println(DASHES)
    println("2. Generate a device certificate.")
    println(
        """
        A device certificate performs a role in securing the communication
between devices (things) and the AWS IoT platform.
        """.trimIndent(),
    )

    print("Do you want to create a certificate for $thingName? (y/n)")
    val certAns = scanner.nextLine()
    var certificateArn: String? = ""
    if (certAns != null && certAns.trim { it <= ' ' }.equals("y", ignoreCase =
true)) {
        certificateArn = createCertificate()
        println("Attach the certificate to the AWS IoT thing.")
        attachCertificateToThing(thingName, certificateArn)
    } else {
        println("A device certificate was not created.")
    }
    println(DASHES)

    println(DASHES)
    println("3. Update an AWS IoT thing with Attributes.")
    println(
        """
        IoT thing attributes, represented as key-value pairs, offer a pivotal
advantage in facilitating efficient data
        management and retrieval within the AWS IoT ecosystem.
        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    updateThing(thingName)
    println(DASHES)

    println(DASHES)
```

```
    println("4. Return a unique endpoint specific to the Amazon Web Services
account.")
    println(
        """
        An IoT Endpoint refers to a specific URL or Uniform Resource Locator that
serves as the entry point for communication between IoT devices and the AWS IoT
service.
        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    val endpointUrl = describeEndpoint()
    println(DASHES)

    println(DASHES)
    println("5. List your AWS IoT certificates")
    print("Press Enter to continue...")
    scanner.nextLine()
    if (certificateArn!!.isNotEmpty()) {
        listCertificates()
    } else {
        println("You did not create a certificates. Skipping this step.")
    }
    println(DASHES)

    println(DASHES)
    println("6. Create an IoT shadow that refers to a digital representation or
virtual twin of a physical IoT device")
    println(
        """
        A thing shadow refers to a feature that enables you to create a virtual
representation, or "shadow,"
        of a physical device or thing. The thing shadow allows you to synchronize
and control the state of a device between
        the cloud and the device itself. and the AWS IoT service. For example,
you can write and retrieve JSON data from a thing shadow.

        """.trimIndent(),
    )
    print("Press Enter to continue...")
    scanner.nextLine()
    updateShawdowThing(thingName)
    println(DASHES)
```

```
    println(DASHES)
    println("7. Write out the state information, in JSON format.")
    print("Press Enter to continue...")
    scanner.nextLine()
    getPayload(thingName)
    println(DASHES)

    println(DASHES)
    println("8. Creates a rule")
    println(
        """
        Creates a rule that is an administrator-level action.
        Any user who has permission to create rules will be able to access data
processed by the rule.
        """.trimIndent(),
    )
    print("Enter Rule name: ")
    val ruleName = scanner.nextLine()
    createIoTRule(roleARN, ruleName, snsAction)
    println(DASHES)

    println(DASHES)
    println("9. List your rules.")
    print("Press Enter to continue...")
    scanner.nextLine()
    listIoTRules()
    println(DASHES)

    println(DASHES)
    println("10. Search things using the name.")
    print("Press Enter to continue...")
    scanner.nextLine()
    val queryString = "thingName:$thingName"
    searchThings(queryString)
    println(DASHES)

    println(DASHES)
    if (certificateArn.length > 0) {
        print("Do you want to detach and delete the certificate for $thingName?
(y/n)")
        val delAns = scanner.nextLine()
        if (delAns != null && delAns.trim { it <= ' ' }.equals("y", ignoreCase =
true)) {
            println("11. You selected to detach amd delete the certificate.")
```

```
            print("Press Enter to continue...")
            scanner.nextLine()
            detachThingPrincipal(thingName, certificateArn)
            deleteCertificate(certificateArn)
        } else {
            println("11. You selected not to delete the certificate.")
        }
    } else {
        println("11. You did not create a certificate so there is nothing to
 delete.")
    }
    println(DASHES)

    println(DASHES)
    println("12. Delete the AWS IoT thing.")
    print("Do you want to delete the IoT thing? (y/n)")
    val delAns = scanner.nextLine()
    if (delAns != null && delAns.trim { it <= ' ' }.equals("y", ignoreCase =
 true)) {
        deleteIoTThing(thingName)
    } else {
        println("The IoT thing was not deleted.")
    }
    println(DASHES)

    println(DASHES)
    println("The AWS IoT workflow has successfully completed.")
    println(DASHES)
}

suspend fun deleteIoTThing(thingNameVal: String) {
    val deleteThingRequest =
        DeleteThingRequest {
            thingName = thingNameVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteThing(deleteThingRequest)
        println("Deleted $thingNameVal")
    }
}

suspend fun deleteCertificate(certificateArn: String) {
    val certificateProviderRequest =
```

```
        DeleteCertificateRequest {
            certificateId = extractCertificateId(certificateArn)
        }
    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.deleteCertificate(certificateProviderRequest)
        println("$certificateArn was successfully deleted.")
    }
}

private fun extractCertificateId(certificateArn: String): String? {
    // Example ARN: arn:aws:iot:region:account-id:cert/certificate-id.
    val arnParts = certificateArn.split(":".toRegex()).dropLastWhile
 { it.isEmpty() }.toTypedArray()
    val certificateIdPart = arnParts[arnParts.size - 1]
    return certificateIdPart.substring(certificateIdPart.lastIndexOf("/") + 1)
}

suspend fun detachThingPrincipal(
    thingNameVal: String,
    certificateArn: String,
) {
    val thingPrincipalRequest =
        DetachThingPrincipalRequest {
            principal = certificateArn
            thingName = thingNameVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.detachThingPrincipal(thingPrincipalRequest)
        println("$certificateArn was successfully removed from $thingNameVal")
    }
}

suspend fun searchThings(queryStringVal: String?) {
    val searchIndexRequest =
        SearchIndexRequest {
            queryString = queryStringVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        val searchIndexResponse = iotClient.searchIndex(searchIndexRequest)
        if (searchIndexResponse.things?.isEmpty() == true) {
            println("No things found.")
        } else {
```

```
            searchIndexResponse.things
                ?.forEach { thing -> println("Thing id found using search is
 ${thing.thingId}") }
        }
    }
}

suspend fun listIoTRules() {
    val listTopicRulesRequest = ListTopicRulesRequest {}

    IotClient { region = "us-east-1" }.use { iotClient ->
        val listTopicRulesResponse =
 iotClient.listTopicRules(listTopicRulesRequest)
        println("List of IoT rules:")
        val ruleList = listTopicRulesResponse.rules
        ruleList?.forEach { rule ->
            println("Rule name: ${rule.ruleName}")
            println("Rule ARN: ${rule.ruleArn}")
            println("--------------")
        }
    }
}

suspend fun createIoTRule(
    roleARNVal: String?,
    ruleNameVal: String?,
    action: String?,
) {
    val sqlVal = "SELECT * FROM '$TOPIC '"
    val action1 =
        SnsAction {
            targetArn = action
            roleArn = roleARNVal
        }

    val myAction =
        Action {
            sns = action1
        }

    val topicRulePayloadVal =
        TopicRulePayload {
            sql = sqlVal
            actions = listOf(myAction)
```

```
        }

    val topicRuleRequest =
        CreateTopicRuleRequest {
            ruleName = ruleNameVal
            topicRulePayload = topicRulePayloadVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.createTopicRule(topicRuleRequest)
        println("IoT rule created successfully.")
    }
}

suspend fun getPayload(thingNameVal: String?) {
    val getThingShadowRequest =
        GetThingShadowRequest {
            thingName = thingNameVal
        }

    IotDataPlaneClient { region = "us-east-1" }.use { iotPlaneClient ->
        val getThingShadowResponse =
 iotPlaneClient.getThingShadow(getThingShadowRequest)
        val payload = getThingShadowResponse.payload
        val payloadString = payload?.let { java.lang.String(it, Charsets.UTF_8) }
        println("Received shadow data: $payloadString")
    }
}

suspend fun listCertificates() {
    IotClient { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.listCertificates()
        val certList = response.certificates
        certList?.forEach { cert ->
            println("Cert id: ${cert.certificateId}")
            println("Cert Arn: ${cert.certificateArn}")
        }
    }
}

suspend fun describeEndpoint(): String? {
    val request = DescribeEndpointRequest {}
    IotClient { region = "us-east-1" }.use { iotClient ->
        val endpointResponse = iotClient.describeEndpoint(request)
```

```
        val endpointUrl: String? = endpointResponse.endpointAddress
        val exString: String = getValue(endpointUrl)
        val fullEndpoint = "https://$exString-ats.iot.us-east-1.amazonaws.com"
        println("Full endpoint URL: $fullEndpoint")
        return fullEndpoint
    }
}

private fun getValue(input: String?): String {
    // Define a regular expression pattern for extracting the subdomain.
    val pattern = Pattern.compile("^(.*?)\\.iot\\.us-east-1\\.amazonaws\\.com")

    // Match the pattern against the input string.
    val matcher = pattern.matcher(input)

    // Check if a match is found.
    if (matcher.find()) {
        val subdomain = matcher.group(1)
        println("Extracted subdomain: $subdomain")
        return subdomain
    } else {
        println("No match found")
    }
    return ""
}

suspend fun updateThing(thingNameVal: String?) {
    val newLocation = "Office"
    val newFirmwareVersion = "v2.0"
    val attMap: MutableMap<String, String> = HashMap()
    attMap["location"] = newLocation
    attMap["firmwareVersion"] = newFirmwareVersion

    val attributePayloadVal =
        AttributePayload {
            attributes = attMap
        }

    val updateThingRequest =
        UpdateThingRequest {
            thingName = thingNameVal
            attributePayload = attributePayloadVal
        }
```

```
    IotClient { region = "us-east-1" }.use { iotClient ->
        // Update the IoT thing attributes.
        iotClient.updateThing(updateThingRequest)
        println("$thingNameVal attributes updated successfully.")
    }
}

suspend fun updateShawdowThing(thingNameVal: String?) {
    // Create the thing shadow state document.
    val stateDocument = "{\"state\":{\"reported\":{\"temperature\":25, \"humidity
\":50}}}"
    val byteStream: ByteStream = ByteStream.fromString(stateDocument)
    val byteArray: ByteArray = byteStream.toByteArray()

    val updateThingShadowRequest =
        UpdateThingShadowRequest {
            thingName = thingNameVal
            payload = byteArray
        }

    IotDataPlaneClient { region = "us-east-1" }.use { iotPlaneClient ->
        iotPlaneClient.updateThingShadow(updateThingShadowRequest)
        println("The thing shadow was updated successfully.")
    }
}

suspend fun attachCertificateToThing(
    thingNameVal: String?,
    certificateArn: String?,
) {
    val principalRequest =
        AttachThingPrincipalRequest {
            thingName = thingNameVal
            principal = certificateArn
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.attachThingPrincipal(principalRequest)
        println("Certificate attached to $thingNameVal successfully.")
    }
}

suspend fun describeThing(thingNameVal: String) {
    val thingRequest =
```

```
        DescribeThingRequest {
            thingName = thingNameVal
        }

    // Print Thing details.
    IotClient { region = "us-east-1" }.use { iotClient ->
        val describeResponse = iotClient.describeThing(thingRequest)
        println("Thing details:")
        println("Thing name: ${describeResponse.thingName}")
        println("Thing ARN:  ${describeResponse.thingArn}")
    }
}

suspend fun createCertificate(): String? {
    IotClient { region = "us-east-1" }.use { iotClient ->
        val response = iotClient.createKeysAndCertificate()
        val certificatePem = response.certificatePem
        val certificateArn = response.certificateArn

        // Print the details.
        println("\nCertificate:")
        println(certificatePem)
        println("\nCertificate ARN:")
        println(certificateArn)
        return certificateArn
    }
}

suspend fun createIoTThing(thingNameVal: String) {
    val createThingRequest =
        CreateThingRequest {
            thingName = thingNameVal
        }

    IotClient { region = "us-east-1" }.use { iotClient ->
        iotClient.createThing(createThingRequest)
        println("Created $thingNameVal}")
    }
}
```

有关 S AWS DK 开发者指南和代码示例的完整列表，请参阅AWS IoT 与 AWS SDK 一起使用。本主题还包括有关入门的信息以及有关先前的 SDK 版本的详细信息。

AWS IoT 配额

您可以在《AWS 一般参考》中查找有关 AWS IoT 配额的信息。

- 有关 AWS IoT Core 配额信息，请参阅 AWS IoT Core 端点和配额。
- 有关 AWS IoT Device Management 配额信息，请参阅 AWS IoT Device Management 端点和配额。
- 有关 AWS IoT Device Defender 配额信息，请参阅 AWS IoT Device Defender 端点和配额。

AWS IoT Core 定价

您可以在 AWS 营销页面和 AWS 定价计算器中找到有关 AWS IoT Core 定价的信息。

- 要查看 AWS IoT Core 定价信息，请参阅 AWS IoT Core 定价。
- 要估算架构解决方案的成本，请参阅 AWS 定价计算器。

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。