
AWS IoT

开发人员指南



AWS IoT: 开发人员指南

Copyright © 2017 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

什么是 AWS IoT ?	1
AWS IoT 的组件	1
如何开始使用 AWS IoT	2
访问 AWS IoT	2
相关服务	2
AWS IoT 的工作原理	2
AWS IoT 入门	4
登录 AWS IoT 控制台	4
在 Thing Registry 中注册设备	5
创建并激活设备证书	7
创建 AWS IoT 策略	9
将 AWS IoT 策略附加到设备证书	12
将证书附加到事物	13
配置您的设备	16
配置 AWS IoT 按钮。	16
配置不同的设备	17
使用 AWS IoT MQTT 客户端查看设备 MQTT 消息	17
配置并测试规则	20
创建一个 SNS 主题	20
订阅 Amazon SNS 主题	22
创建规则	22
测试 Amazon SNS 规则	28
后续步骤	29
AWS IoT 按钮快速入门	30
AWS IoT 按钮向导快速入门	31
AWS IoT 按钮 AWS CloudFormation 快速入门	39
后续步骤	44
AWS IoT 规则教程	45
创建 DynamoDB 规则	45
创建 Lambda 规则	54
创建 Lambda 函数	54
测试您的 Lambda 函数	62
创建 Lambda 规则	64
测试您的 Lambda 规则	67
创建 Amazon SNS 规则	69
AWS IoT 软件开发工具包教程	77
连接您的 Raspberry Pi	77
先决条件	77
登录 AWS IoT 控制台	78
创建并附加事物 (设备)	79
使用 AWS IoT 嵌入式 C 软件开发工具包	86
设置适用于 AWS IoT 嵌入式 C 软件开发工具包的运行时环境	86
示例应用程序配置	86
运行示例应用程序	88
使用适用于 JavaScript 的 AWS IoT 设备软件开发工具包	89
设置适用于 JavaScript 的 AWS IoT 设备软件开发工具包的运行时环境	89
安装适用于 JavaScript 的 AWS IoT 设备软件开发工具包	91
准备运行示例应用程序	91
运行示例应用程序	91
使用 AWS IoT 管理事物	93
利用 Thing Registry 管理事物	93
创建事物	93
列出事物	94
搜索事物	94

更新事物	95
删除事物	96
将委托人附加到事物	96
将委托人与事物分离	96
事物类型	96
创建事物类型	97
列出事物类型	97
描述事物类型	97
将事物类型与事物相关联	98
弃用事物类型	98
删除事物类型	99
安全和身份	100
AWS IoT 中的身份验证	100
X.509 证书	101
IAM 用户、组和角色	107
Amazon Cognito 身份	107
授权	107
AWS IoT 策略	109
IAM IoT 策略	127
跨账户访问	130
传输安全	131
TLS 密码包支持	131
消息代理	133
协议	133
协议/端口映射	133
MQTT	133
HTTP	134
基于 WebSocket 的 MQTT 协议	135
主题	138
预留的主题	138
生命周期事件	141
接收生命周期事件时需要用到的策略	141
连接/断开连接事件	141
订阅/取消订阅事件	142
规则	144
授予 AWS IoT 所需的访问权限	145
传递角色权限	146
创建 AWS IoT 规则	147
查看您的规则	150
SQL 版本	150
2016-03-23 SQL 规则引擎版本中的新增功能	150
排查规则问题	152
删除规则	152
AWS IoT 规则操作	152
CloudWatch 警报操作	152
CloudWatch 指标操作	153
DynamoDB 操作	154
DynamoDBv2 操作	155
Amazon ES 操作	156
Firehose 操作	157
Kinesis 操作	157
Lambda 操作	158
Republish 操作	159
S3 操作	159
SNS 操作	160
SQS 操作	161
Salesforce 操作	162

AWS IoT SQL 参考	162
数据类型	163
运算符	166
函数	171
SELECT 语句	199
FROM 子句	201
WHERE 子句	202
文本	202
Case 语句	202
JSON 扩展	203
替换模板	204
事物影子	205
事物影子数据流	205
检测事物是否连接	211
事物影子文档	212
文档属性	213
事物影子的版本控制	213
客户端令牌	213
示例文档	214
空白部分	214
数组	215
使用事物影子	215
协议支持	216
更新事物影子	216
检索事物影子文档	216
删除数据	219
删除事物影子	220
增量状态	220
观察状态更改	222
消息顺序	222
修剪事物影子消息	223
RESTful API	224
GetThingShadow	224
UpdateThingShadow	225
DeleteThingShadow	225
MQTT 发布/订阅主题	226
/更新	226
/update/accepted	227
/update/documents	228
/update/rejected	228
/update/delta	229
/get	229
/get/accepted	230
/get/rejected	230
/delete	231
/delete/accepted	231
/delete/rejected	231
文档语法	232
请求状态文档	232
响应状态文档	233
错误响应文档	234
错误消息	234
AWS IoT 软件开发工具包	236
适用于 Android 的 AWS 移动软件开发工具包	236
Arduino Yún 软件开发工具包	236
适用于嵌入式 C 的 AWS IoT 设备软件开发工具包	237
适用于 iOS 的 AWS 移动软件开发工具包	237

适用于 Java 的 AWS IoT 设备软件开发工具包	237
适用于 JavaScript 的 AWS IoT 设备软件开发工具包	237
适用于 Python 的 AWS IoT 设备软件开发工具包	237
监控	239
监控工具	240
自动化工具	240
手动工具	240
使用 Amazon CloudWatch 进行监控	241
指标与维度	241
使用 AWS IoT 指标	245
创建 CloudWatch 警报	245
使用 AWS CloudTrail 记录 AWS IoT API 调用	247
CloudTrail 中的 AWS IoT 信息	247
了解 AWS IoT 日志文件条目	248
故障排除	250
诊断连接问题	250
身份验证	250
授权	250
设置 CloudWatch Logs	251
为日志记录配置 IAM 角色	251
CloudWatch 日志条目格式	252
日志记录事件和错误代码	253
诊断规则问题	255
诊断 Thing Shadows 问题	256
诊断 Salesforce 操作问题	257
执行跟踪	257
操作成功和失败	257

什么是 AWS IoT？

AWS IoT 可在连接到 Internet 的事物（如传感器、执行器、嵌入式微控制器或智能设备）与 AWS 云之间实现安全的双向通信。这样，您可以从多个事物收集遥测数据并且存储和分析数据。您也可以创建令用户能够通过手机或平板电脑控制这些设备的应用程序。

AWS IoT 的组件

AWS IoT 包括以下组件：

设备网关

使设备能够安全高效地与 AWS IoT 进行通信。

消息代理

提供安全机制以供事务和 AWS IoT 应用程序用于相互发布和接收消息。进行发布和订阅时，您可以直接使用 MQTT 协议，也可以通过 WebSocket 使用 MQTT 协议。您可以使用 HTTP REST 接口进行发布。

规则引擎

提供消息处理及与其他 AWS 服务进行集成的功能。您可以使用基于 SQL 的语言选择消息负载中的数据，处理数据并将数据发送到其他服务，如 Amazon S3、Amazon DynamoDB 和 AWS Lambda。您还可以使用消息代理面向其他订阅者重新发布消息。

安全和身份服务

在 AWS 云中共担安全责任。为了安全地将数据发送到消息代理，您的事物必须确保自身凭证的安全。

消息代理和规则引擎使用 AWS 安全功能将数据安全发送到设备或其他 AWS 服务。

Thing Registry

有时也称为 Device Registry。组织与每个事务相关的资源。您可以注册自己的事务并将每个事务与最多三个自定义属性相关联。您还可以将每个事务与相应的证书和 MQTT 客户端 ID 相关联，以提高对事务进行管理和故障排除的能力。

Thing Shadow

有时也称为 Device Shadow。一种 JSON 文档，用于存储和检索事物（设备、应用程序等等）的当前状态信息。

Thing Shadows 服务

在 AWS 云中提供事务的永久性表示形式。您可以向 Thing Shadow 发布更新后的状态信息，您的事务可在建立连接时实现状态同步。您的事务还可以将有关自身状态的信息发布到 Thing Shadow 中，以供应用程序或设备使用。

有关 AWS IoT 限制的信息，请参阅 [AWS IoT 限制](#)。

如何开始使用 AWS IoT

- 要了解有关 AWS IoT 的更多信息，请参阅 [AWS IoT 的工作原理 \(p. 2\)](#)。
- 要了解如何将事物连接到 AWS IoT，请参阅 [AWS IoT 入门 \(p. 4\)](#)。

访问 AWS IoT

AWS IoT 提供以下接口以创建事务并与之交互：

- AWS Command Line Interface (AWS CLI) – 在 Windows、OS X 和 Linux 上运行适用于 AWS IoT 的命令。您可以使用这些命令创建并管理事物、证书、规则和策略。要开始使用，请参阅 [AWS Command Line Interface 用户指南](#)。有关适用于 AWS IoT 的命令的更多信息，请参阅 AWS Command Line Interface Reference 中的 `iot`。
- AWS IoT API – 使用 HTTP 或 HTTPS 请求构建您的 IoT 应用程序。您可以使用这些 API 以编程方式创建和管理事物、证书、规则和策略。有关适用于 AWS IoT 的 API 操作的更多信息，请参阅 [AWS IoT API 参考](#)中的操作。
- AWS SDK – 使用语言特定 API 构建您的 IoT 应用程序。这些软件开发工具包中封装了 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 SDKs and Tools](#)。
- AWS IoT 设备软件开发工具包 — 构建在设备上运行的应用程序，以便与 AWS IoT 之间收发消息。有关更多信息，请参阅 [AWS IoT 软件开发工具包](#)。

相关服务

AWS IoT 可直接与以下 AWS 服务集成：

- Amazon Simple Storage Service – 在 AWS 云中提供可扩展存储。有关更多信息，请参阅 [Amazon S3](#)。
- Amazon DynamoDB – 提供托管型 NoSQL 数据库。有关更多信息，请参阅 [Amazon DynamoDB](#)。
- Amazon Kinesis – 可实时进行大规模的流数据处理。有关更多信息，请参阅 [Amazon Kinesis](#)。
- AWS Lambda – 在 Amazon EC2 的虚拟服务器上运行代码以响应事件。有关更多信息，请参阅 [AWS Lambda](#)。
- Amazon Simple Notification Service – 发送或接收通知。有关更多信息，请参阅 [Amazon SNS](#)。
- Amazon Simple Queue Service – 将数据存储在队列中以供应用程序检索。有关更多信息，请参阅 [Amazon SQS](#)。

AWS IoT 的工作原理

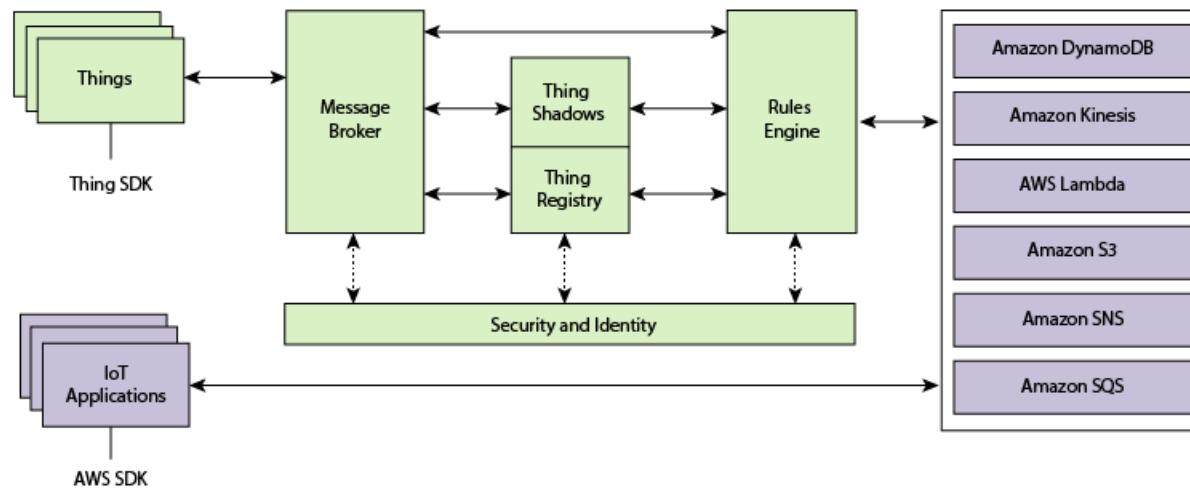
AWS IoT 使连接到 Internet 的事物能够连接到 AWS 云，并使云中的应用程序能够与连接到 Internet 的事物进行交互。常见的 IoT 应用程序可从设备收集和处理遥测数据，或者令用户能够远程控制设备。

事物通过在 MQTT 主题下以 JSON 格式发布消息来报告自身的状态。每个 MQTT 主题都有一个分层名称，可识别状态正在更新的事物。当消息在 MQTT 主题中发布时，消息将发送至 AWS IoT MQTT 消息代理，后者负责将在该 MQTT 主题下发布的所有消息发送给已订阅该主题的所有客户端。

通过使用 X.509 证书保护事物与 AWS IoT 之间的通信。AWS IoT 可为您生成一个证书，您也可以使用自己的证书。无论哪种情况，都必须通过 AWS IoT 注册并激活证书，然后再复制到您的事物中。当您的事物与 AWS IoT 进行通信时，它会将证书作为凭证提供给 AWS IoT。

我们建议在 Thing Registry 中为连接到 AWS IoT 的所有事物都创建一个条目。Thing Registry 存储事物的相关信息以及事物用于确保与 AWS IoT 安全进行通信的证书。

您可以制定相关规则，确定要基于消息内数据执行的一项或多项操作。例如，您可以插入、更新或查询 DynamoDB 表，或者调用 Lambda 功能。规则使用表达式来筛选消息。当规则与一条消息匹配时，规则引擎会使用所选属性调用该操作。规则还包含 IAM 角色，该角色可授予 AWS IoT 对用于执行操作的 AWS 资源的权限。



每个事物都有一个可存储和检索状态信息的事物影子。状态信息中的每一项都含两个条目：事物报告的最新状态，以及应用程序请求的预期状态。应用程序可以请求事物的当前状态信息。影子通过提供包含状态信息(报告状态和预期状态)、元数据和版本号的 JSON 文档来响应请求。应用程序可以通过请求更改状态来控制事物。影子将接受状态更改请求、更新其状态信息，然后发送消息来指明状态信息已更新。事物会接收消息、更改状态，然后报告其新状态。

AWS IoT 入门

本教程向您展示如何创建从使用 AWS IoT 的设备发送、接收和处理 MQTT 消息所需的资源。

完成本教程需要做以下准备：

- 一台能够访问 Wi-Fi 的计算机。
- 如果您已经有 AWS IoT 按钮 (如图所示)，则可以使用此按钮来完成本教程。
- 如果您没有按钮，则可以在[此处](#)购买一个或在 AWS IoT 控制台中使用 MQTT 客户端来模拟一个设备。



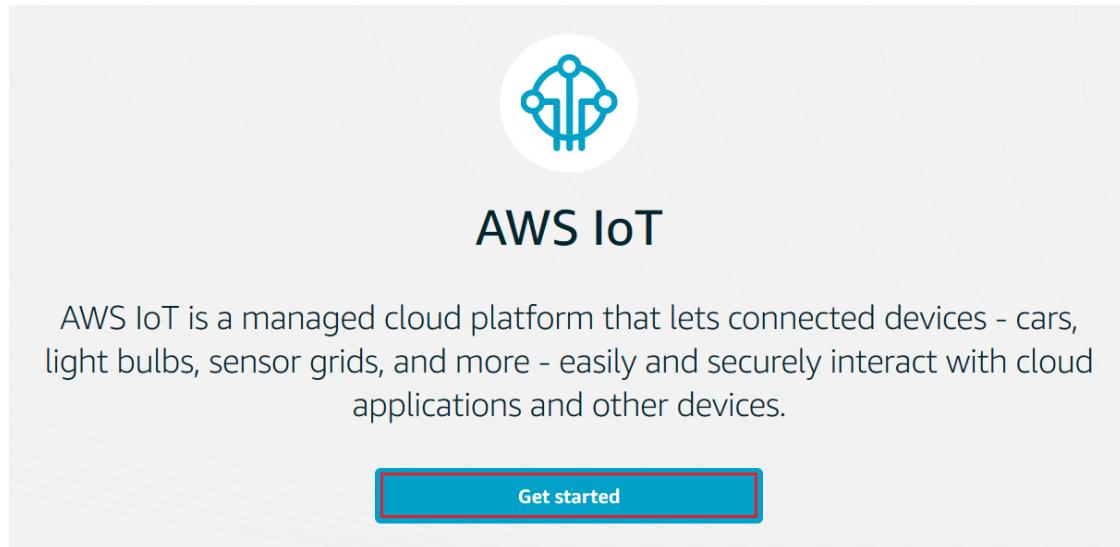
有关 AWS IoT 的更多信息，请参阅[什么是 AWS IoT \(p. 1\)](#)。

登录 AWS IoT 控制台

如果您没有 AWS 账户，请创建一个。

创建 AWS 账户：

1. 打开 [AWS 主页](#)，然后选择 Create an AWS Account。
2. 按照在线说明操作。在注册过程中，您会接到来电并且需要使用手机键盘输入 PIN 码。
3. 登录 AWS 管理控制台，然后打开 [AWS IoT 控制台](#)。
4. 在 Welcome 页面上，选择 Get started。



如果这是您第一次使用 AWS IoT 控制台，您会看到 Welcome to the AWS IoT Console 页面 (如下一个图所示)。

在 Thing Registry 中注册设备

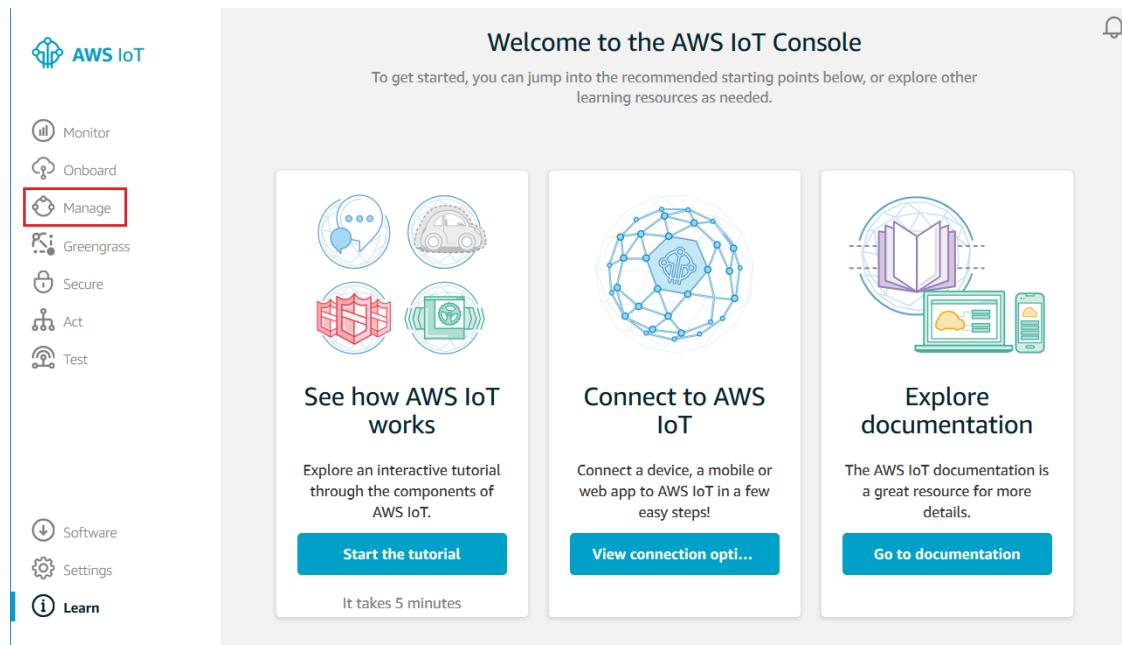
连接至 AWS IoT 的设备在 Thing Registry 中由事物代表。借助 Thing Registry，您可以将连接到您的 AWS IoT 账户的所有设备的记录保留下来。

开始使用 AWS IoT 按钮的最快方法是下载适用于 iOS 或 Android 的移动应用程序。此移动应用程序会为您创建必要的 AWS IoT 资源，然后使用 Lambda 蓝图向您的按钮添加一个调用您选择的新 AWS Lambda 函数的事件源。蓝图是预配置的 Lambda 函数，您可以通过它将按钮点击操作快速连接到最适合您的函数，如发送自动电子邮件、短信或部署其他 AWS 服务。您可以从以下位置下载移动应用程序：[Apple 应用商店](#)或 [Google Play](#)。

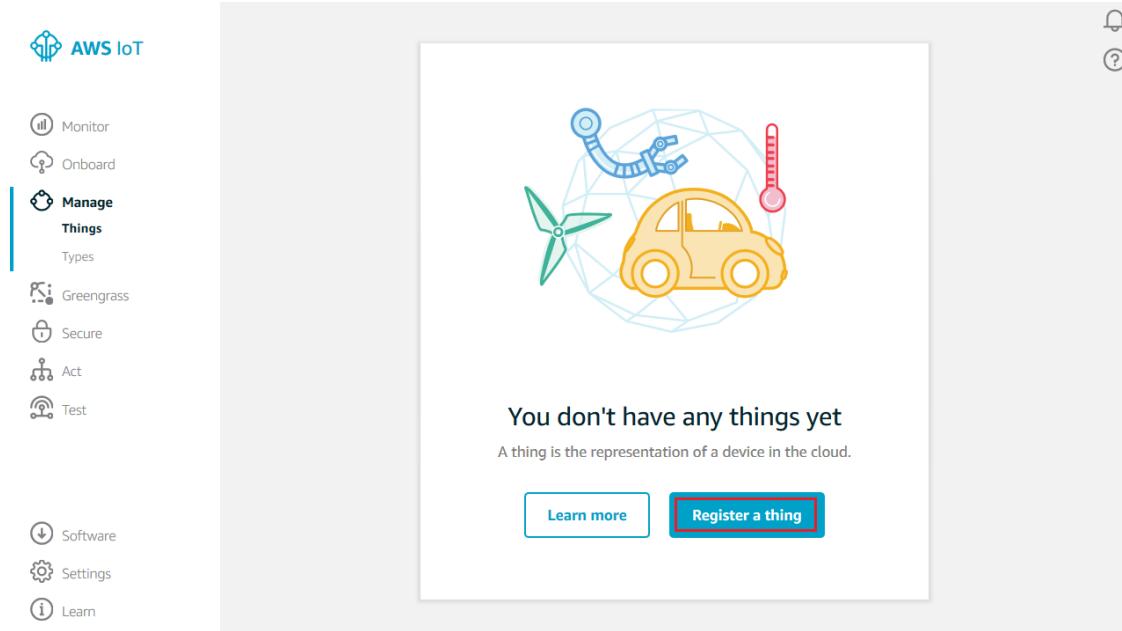
如果您无法使用移动应用程序，请查看以下说明。

在 Thing Registry 中注册您的设备：

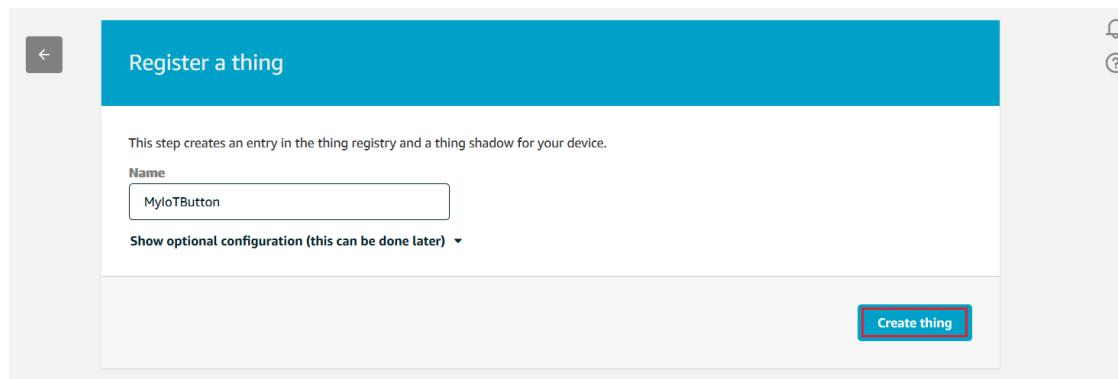
1. 在 Welcome to the AWS IoT Console 页面上的左侧导航窗格中选择 Manage 以展开选项，然后选择 Things (必要时)。



2. 在提示 You don't have any things yet 的页面上，选择 Register a thing。



3. 在 Register a thing 页面上的 Name 字段中键入设备的名称，例如 **MyIoTButton**。选择 Create thing 可将您的设备添加到 Thing Registry 中。



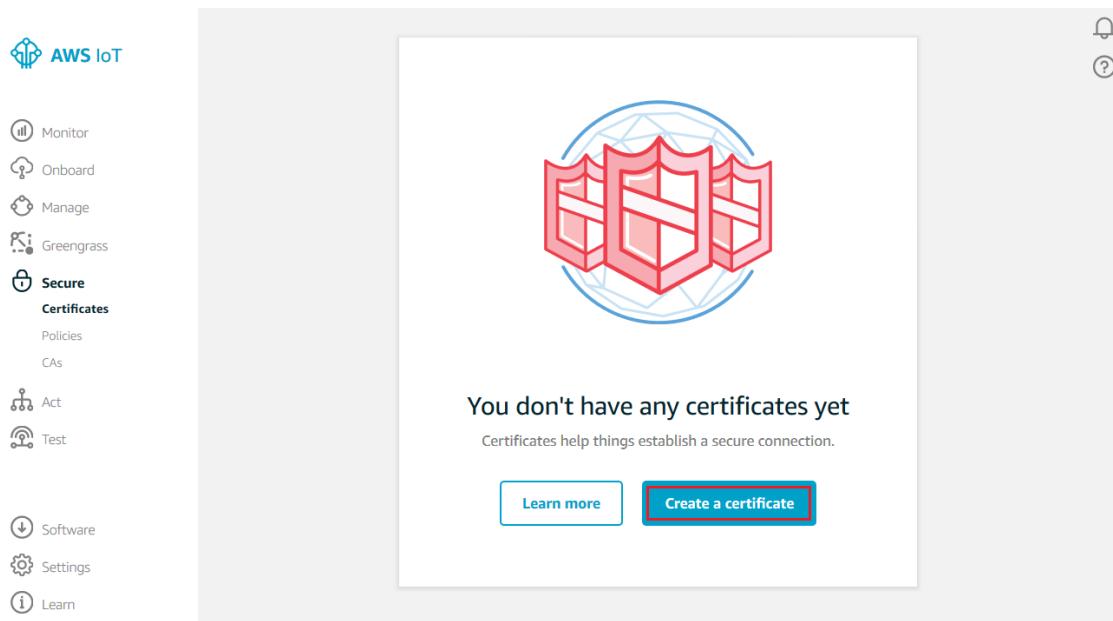
这将产生以下结果。

The screenshot shows the AWS IoT 'Things' list. On the left is a navigation sidebar with 'AWS IoT' and various management sections like 'Monitor', 'Onboard', 'Manage', 'Greengrass', 'Secure', 'Act', 'Test', 'Software', 'Settings', and 'Learn'. The main area shows a table with one row for 'MyloTButton' under the 'NO TYPE' column. A red box highlights this row. The top right of the screen has a search bar, a 'Create' button, and other navigation icons.

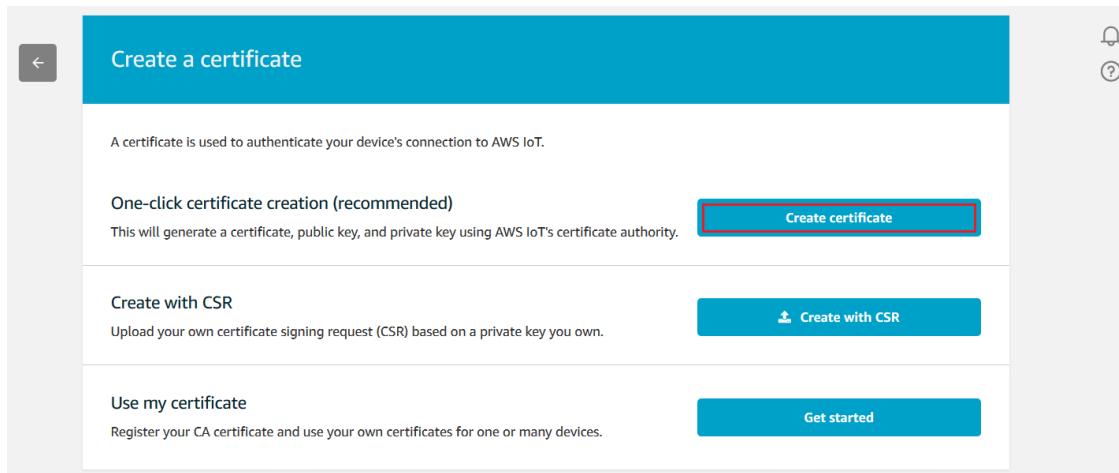
创建并激活设备证书

通过使用 X.509 证书保护您的设备与 AWS IoT 之间的通信。AWS IoT 可为您生成证书，或者您也可以使用自己的 X.509 证书。本教程假设 AWS IoT 将为您生成 X.509 证书。必须先激活证书，然后才能使用它们。

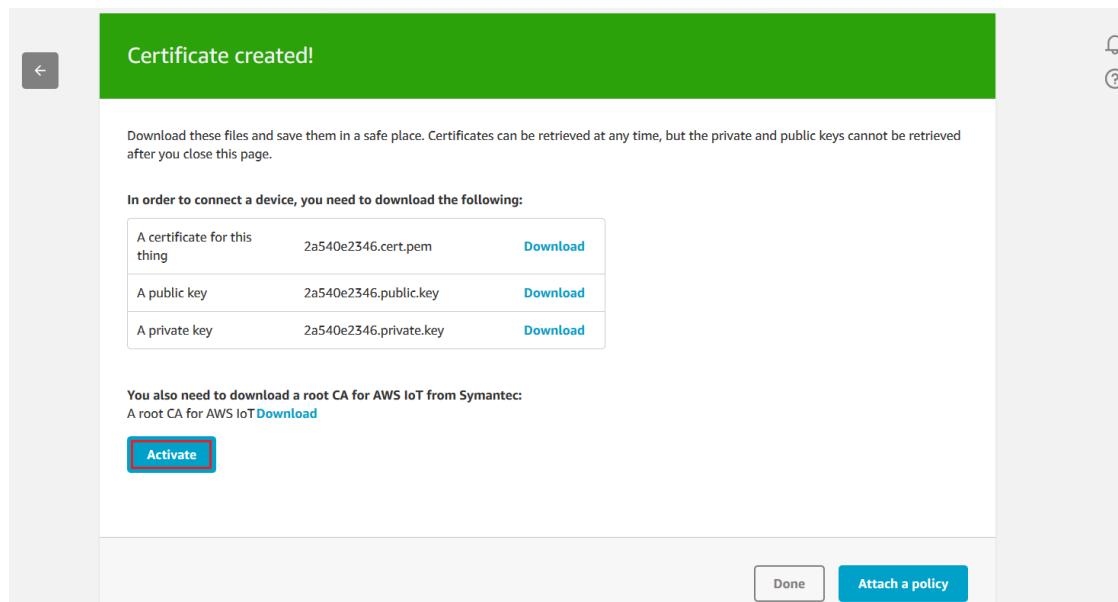
1. 在左侧导航窗格中，选择 Secure、Certificates (如有必要)，然后选择 Create a certificate。



2. 在 Create a certificate 页面上，选择 Create certificate。



3. 在 Certificate created! 页面上，为证书、私有密钥和 AWS IoT 的根 CA 选择 Download (无需下载公有密钥)。将其中每一项都保存到您的计算机上，然后选择 Activate 以继续。



请注意，已下载的文件名可能看起来不同于 Certificate created! 页上列出的文件名。例如：

- 2a540e2346-certificate.pem.crt.txt
- 2a540e2346-private.pem.key
- 2a540e2346-public.pem.key

Note

根 CA 证书可能会过期和/或被吊销，但发生这种情况的可能性不大。如果发生这种情况，您需要将新的根 CA 证书复制到您的计算机上。

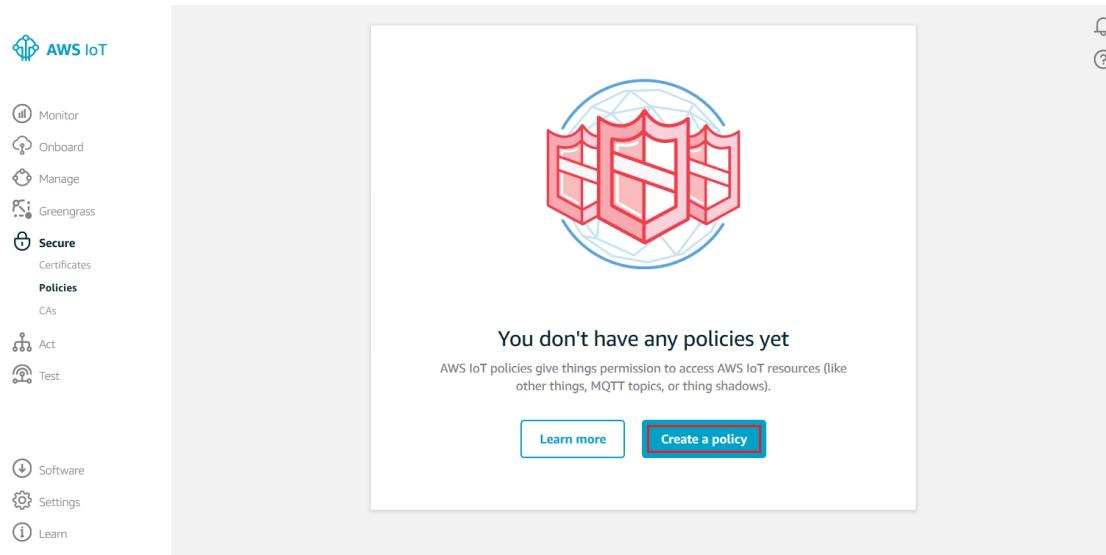
4. 选择完成。

创建 AWS IoT 策略

X.509 证书用于对您的 AWS IoT 设备进行身份验证。AWS IoT 策略用于授予您的设备执行 AWS IoT 操作的权限，比如订阅 MQTT 主题或向 MQTT 主题发布消息。您的设备将在向 AWS IoT 发送消息时展示其证书。要允许您的设备执行 AWS IoT 操作，您必须创建 AWS IoT 策略并将其附加您的设备证书中。

创建 AWS IoT 策略：

1. 在左侧导航窗格中选择 Secure，然后选择 Policies。在 You don't have a policy yet 页面上，选择 Create a policy。



2. 在 Create a policy 页面上的 Name 字段中键入策略的名称 (例如，**MyIoTButtonPolicy**)。在 Action 字段中键入 `iot:Connect`。在 Resource ARN 字段中键入 `*`。选中 Allow 复选框。这样一来，所有客户端均能连接至 AWS IoT。

Create a policy

Create a policy to define a set of authorized actions. You can authorize actions on one or more resources (things, topics, topic filters).

Name

Add statements

Policy statements define the types of actions that can be performed by a resource.

Advanced mode

Action <input type="text" value="iot:Connect"/>
Resource ARN <input type="text" value="*"/>
Effect <input checked="" type="checkbox"/> Allow <input type="checkbox"/> Deny
Remove

Add statement

Note

您可以将客户端 ARN 指定为资源，从而限定哪些客户端（设备）能够连接。客户端 ARN 应采用以下格式：

`arn:aws:iot:<your-region>:<your-aws-account>:client/<my-client-id>`

选择 Add Statement 按钮添加另一个策略语句。在 Action 字段键入 `iot:Publish`。在 Resource ARN 字段中，键入设备要在其下发布消息的主题的相应 ARN。

Note

主题 ARN 遵循以下格式：

`arn:aws:iot:<your-region>:<your-aws-account>:topic/iotbutton/<your-button-serial-number>`

例如：

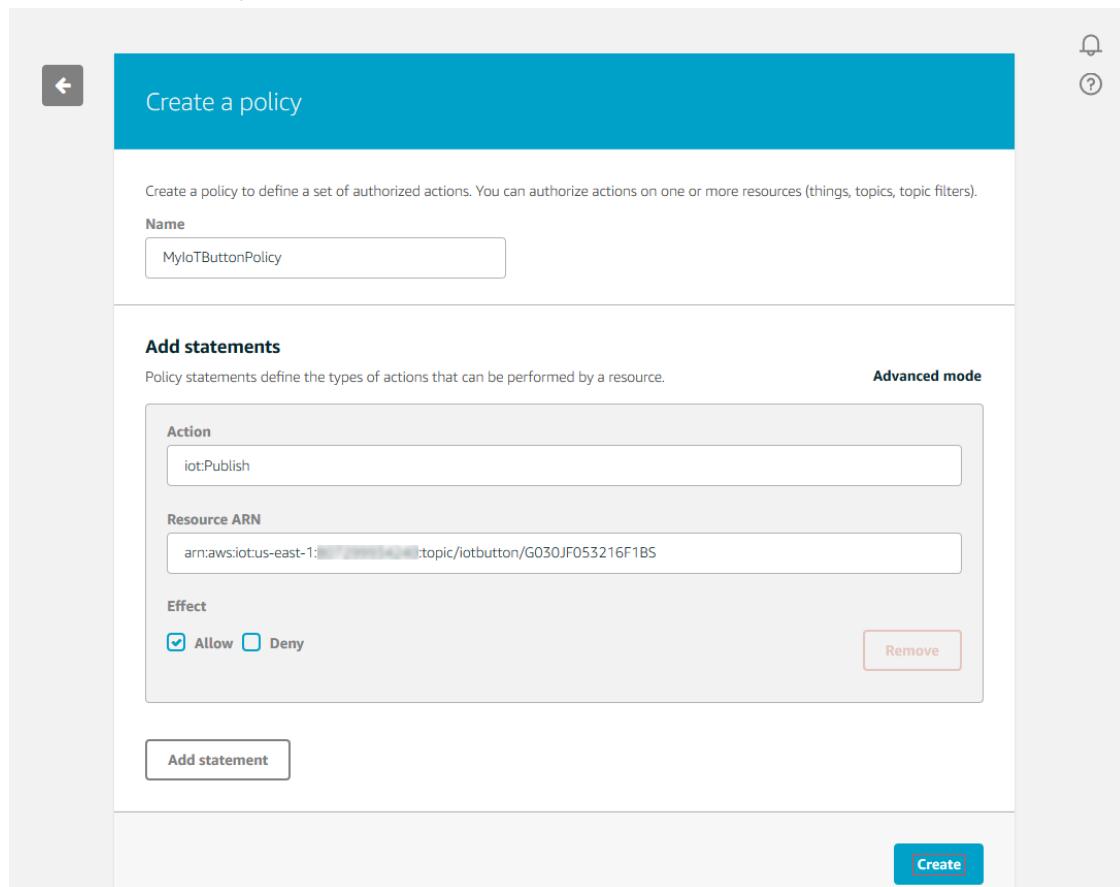
`arn:aws:iot:us-east-1:123456789012:topic/iotbutton/G030JF055364XVRB`
在按钮底部，可以找到序列号。

如果您当前未使用 AWS IoT 按钮，请在 ARN 中将设备发布的主题放在 `topic/` 之后。例如：

`arn:aws:iot:us-east-1:123456789012:topic/my/topic/here`

最后，选中 Allow 复选框。这样一来，您的设备便可以在指定主题下发布消息。

- 在输入策略的信息后，选择 Create。

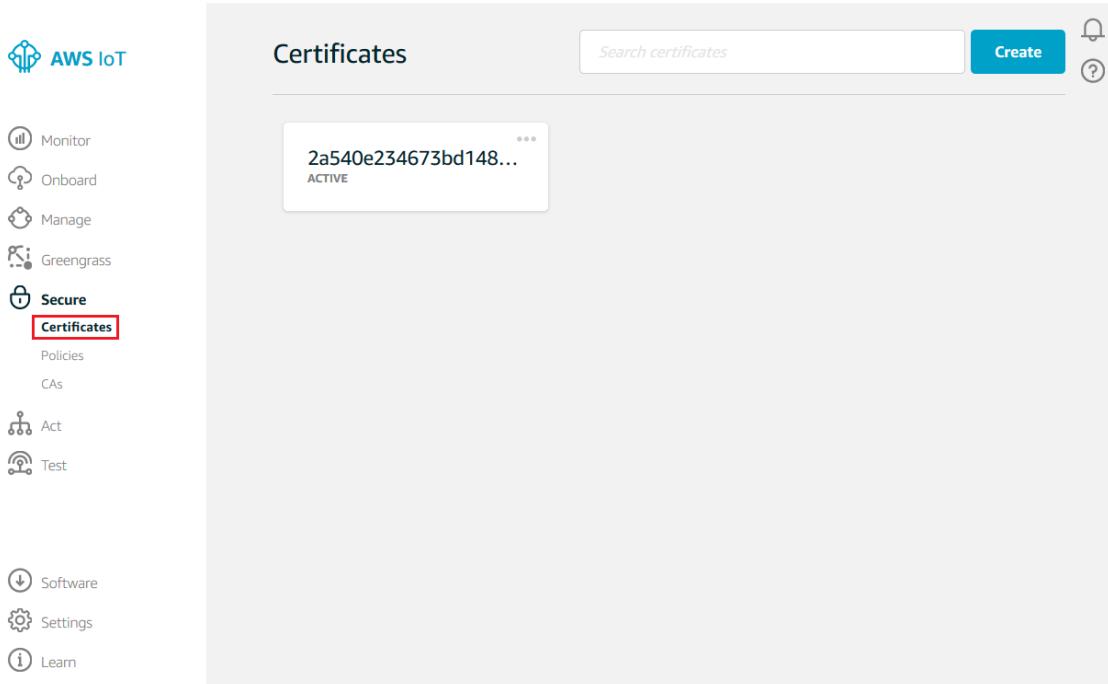


有关详细信息，请参阅管理 AWS IoT 策略。

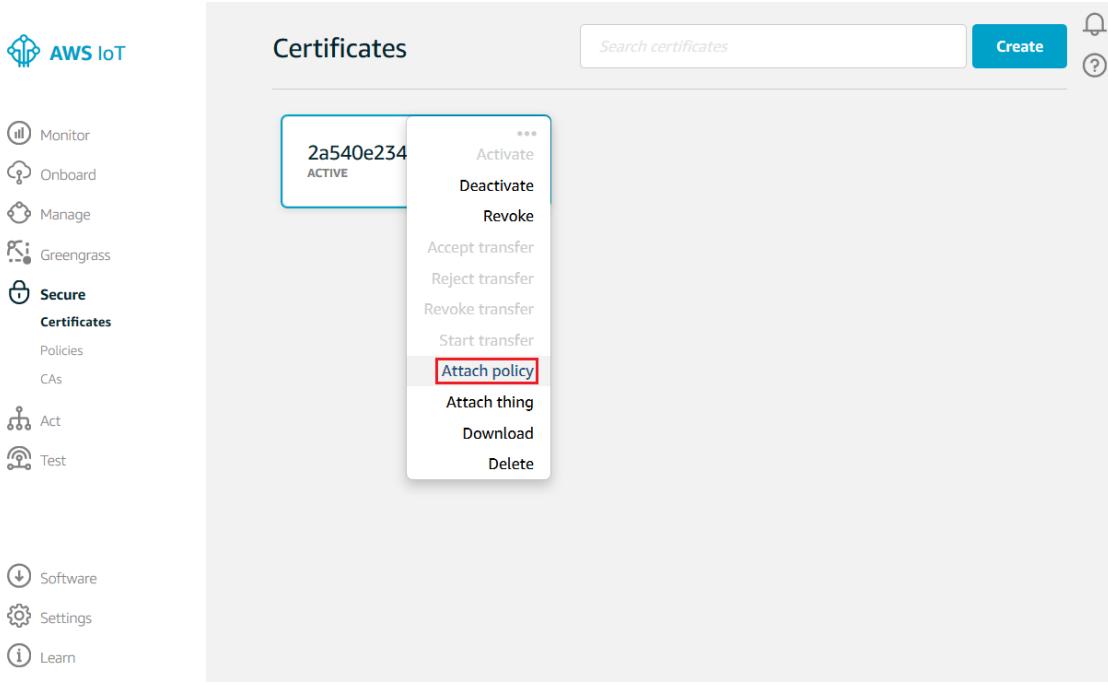
将 AWS IoT 策略附加到设备证书

现在，您已经创建了策略，您必须将它附加到您的设备证书中。通过将 AWS IoT 策略附加到证书中，可授予设备在策略中指定的权限。

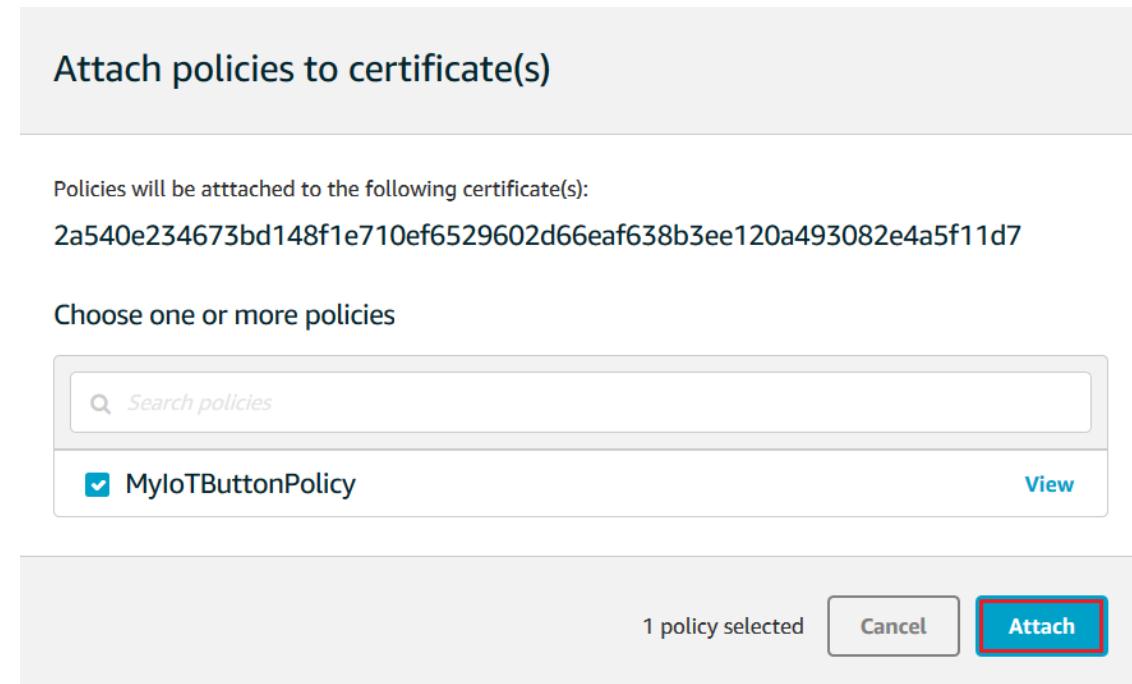
1. 在左侧导航窗格中选择 Secure，然后选择 Certificates。



2. 在您创建的证书栏，选择 ... 打开下拉菜单，然后选择 Attach policy。



3. 在 Attach policies to certificate(s) 对话框中，选中您在上一步中创建的策略旁边的复选框，然后选择 Attach。

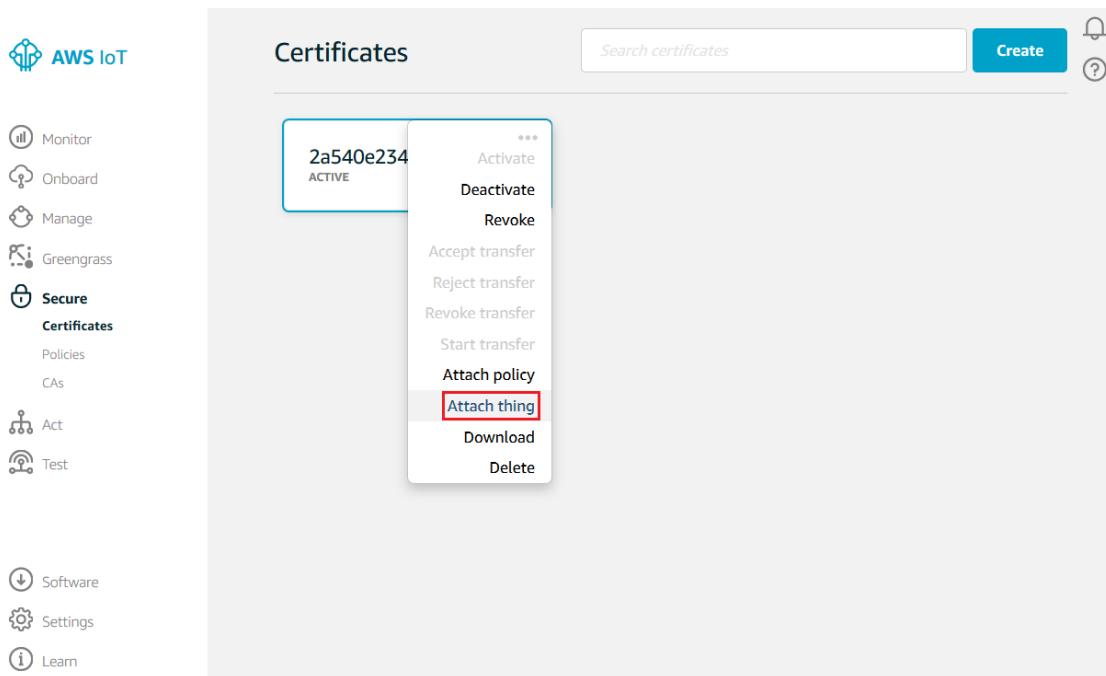


将证书附加到事物

设备必须拥有证书、私有密钥和根 CA 证书才能在 AWS IoT 中进行身份验证。我们建议您将设备证书附加到在 AWS IoT 中代表您的设备的事物。这让您能够基于附加到事物的证书来创建授予权限的 AWS IoT 策略。有关，请参阅 [事物策略变量 \(p. 114\)](#)

将证书附加到在 Thing Registry 中代表您的设备的事物：

1. 在您创建的证书栏，选择 ... 打开下拉菜单，然后选择 Attach thing。



2. 在 Attach things to certificate(s) 对话框中，选中您已经注册的事物旁边的复选框，然后选择 Attach。

Attach things to certificate(s)

Things will be attached to the following certificate(s):

2a540e234673bd148f1e710ef6529602d66eaf638b3ee120a493082e4a5f11d7

Choose one or more things

This screenshot shows the "Attach things to certificate(s)" dialog. It features a search bar labeled "Search things" and a list of registered things. One item, "MyloTButton", has a checked checkbox next to it. The entire list item is enclosed in a light gray box.

1 thing selected

Cancel

Attach

3. 要验证事物已附加，请选中代表证书的栏。

The screenshot shows the AWS IoT Certificates page. On the left, there is a navigation sidebar with various icons and links: Monitor, Onboard, Manage, Greengrass, Secure (selected), Certificates, Policies, CAs, Act, Test, Software, Settings, and Learn. The main area is titled "Certificates" and contains a search bar and a "Create" button. A single certificate entry is listed, with its ID "2a540e234673bd148..." and status "ACTIVE" highlighted by a red rectangle.

4. 在证书的 Details 页面上，在左侧导航窗格中选择 Things。

The screenshot shows the "Details" page for the certificate "2a540e234673bd148f1e710ef6529602d66eaf638b3ee120a493082e4a5f11d7". The left sidebar has "Details" (selected), "Policies", and "Things" (highlighted with a red rectangle). The main content area shows a list under the "Things" tab with one item: "MyIoTButton".

5. 要验证策略已附加，请在证书的 Details 页面上，在左侧导航窗格中选择 Policies。

The screenshot shows the "Details" page for the same certificate. The left sidebar has "Details" (selected), "Policies" (highlighted with a red rectangle), and "Things". The main content area shows a list under the "Policies" tab with one item: "MyIoTButtonPolicy".

配置您的设备

配置您的设备，使其能够连接到您的 Wi-Fi 网络。您的设备必须连接至 Wi-Fi 网络才能安装需要的文件和向 AWS IoT 发送消息。为了与 AWS IoT 通信，所有设备必须安装设备证书、私有密钥和根 CA 证书。

配置 AWS IoT 按钮。

配置您的 AWS IoT 按钮的最简单方法是使用 AWS IoT 按钮智能手机应用程序。您可以从 [Apple 应用商店](#) 或 [Google Play 商店](#) 下载该应用程序。如果您无法使用该智能手机应用程序，请按照以下说明操作来配置您的按钮：

配置您的 AWS IoT 按钮：

打开您的设备

1. 拆除 AWS IoT 按钮的包装，然后按住按钮，直到显示蓝色闪光信号灯。(该时间不应超过 15 秒。)
2. 此按钮可充当 Wi-Fi 接入点，因此，当您的计算机搜索 Wi-Fi 网络时，它将发现一个名为 Button ConfigureMe - XXX 的接入点，其中 XXX 是按钮生成的字符串(含三个字符)。使用您的计算机连接到按钮的 Wi-Fi 接入点。

Note

当蓝光停止闪烁时，按钮会停止将自身作为 Wi-Fi 接入点。因此，如果您无法足够快地完成以下过程，可能需要多次打开闪烁蓝光的指示灯才能完成以下过程。配置后，设备不需要将自身作为 Wi-Fi 接入点，并像任何其他计算机一样，使用本地 Wi-Fi 网络与 Internet 进行通信。

3. 当您第一次连接到按钮的 Wi-Fi 接入点时，系统将提示您输入 WPA2-PSK 密码。请键入设备序列号(DSN) 的后 8 个字符。在设备背面可以找到 DSN，如下所示：



将您的设备证书和私有密钥复制到 AWS IoT 按钮

要连接到 AWS IoT，您必须将您的设备证书和私有密钥复制到 AWS IoT 按钮上。

1. 在浏览器中，导航到 <http://192.168.0.1/index.html>。
2. 填写配置表单：
 - 键入您的 Wi-Fi SSID 和密码。
 - 浏览找到并选择您的证书和私有密钥。例如，分别为 2a540e2346-certificate.pem.crt.txt 和 2a540e2346-private.pem.key。
 - 在 [AWS IoT 控制台](#) 中查找您的自定义终端节点。(在控制面板的左侧导航窗格中，选择 Manage，然后选择 Things。选中代表您的按钮的栏可显示其详细信息页面。在详细信息页面上的左侧导航窗格中选择 Interact 并查找顶部附近的 HTTPS 部分。)您的终端节点看起来如下所示：

ABCDEF1234567.iot.us-east-2.amazonaws.com

其中 ABCDEF1234567 是子域，us-east-2 是区域。

- 在 Button ConfigureMe 页面上，键入子域，然后选择与您的 AWS IoT 终端节点中的区域相匹配的区域。

- 选中 Terms and Conditions 复选框。现在，您的设置看起来应如下所示：

Button ConfigureMe

Enter the value for any field that you wish to change for device: G030JF055364XVRB

Wi-Fi Configuration:

SSID	Guest
Security	<input checked="" type="checkbox"/> Open Network(No Password)
Password	None (unsecured)

AWS IoT Configuration:

Certificate	Choose File MyIoTButtonCert.pem
Private Key	Choose File MyIoTButtonKey.pem
Endpoint Subdomain	AMZN9F6MTZ77O
Endpoint Region	AMUN9F6MTZ77O.iot.us-east-1.amazonaws.com

By clicking this box, you agree to the [AWS IoT Button Terms and Conditions](#).

[Configure](#)

- 选择 Configure。您的按钮现在应连接到您的 Wi-Fi 网络。

配置不同的设备

请按照设备文档的说明连接设备，并将设备证书、私人密钥和根 CA 证书复制到设备上。

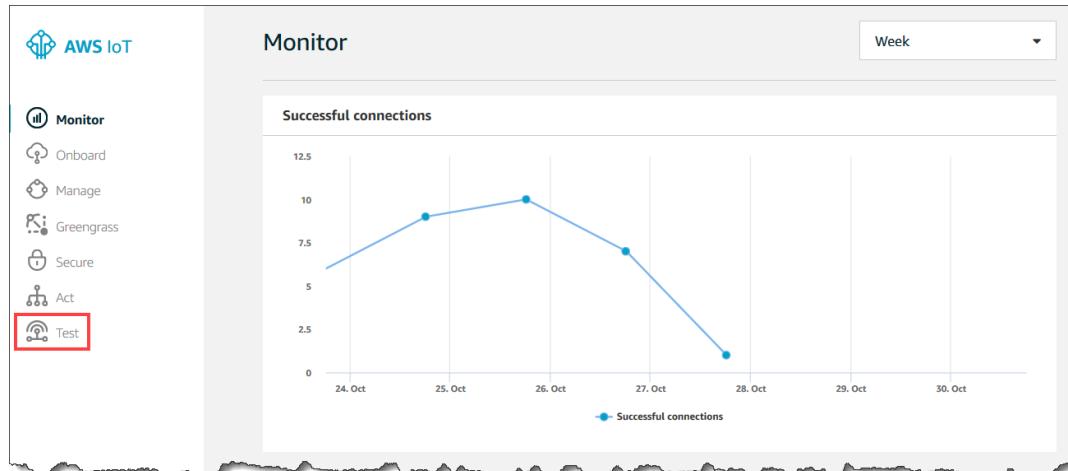
使用 AWS IoT MQTT 客户端查看设备 MQTT 消息

您可以使用 AWS IoT MQTT 客户端来更好地了解设备发送的 MQTT 消息。

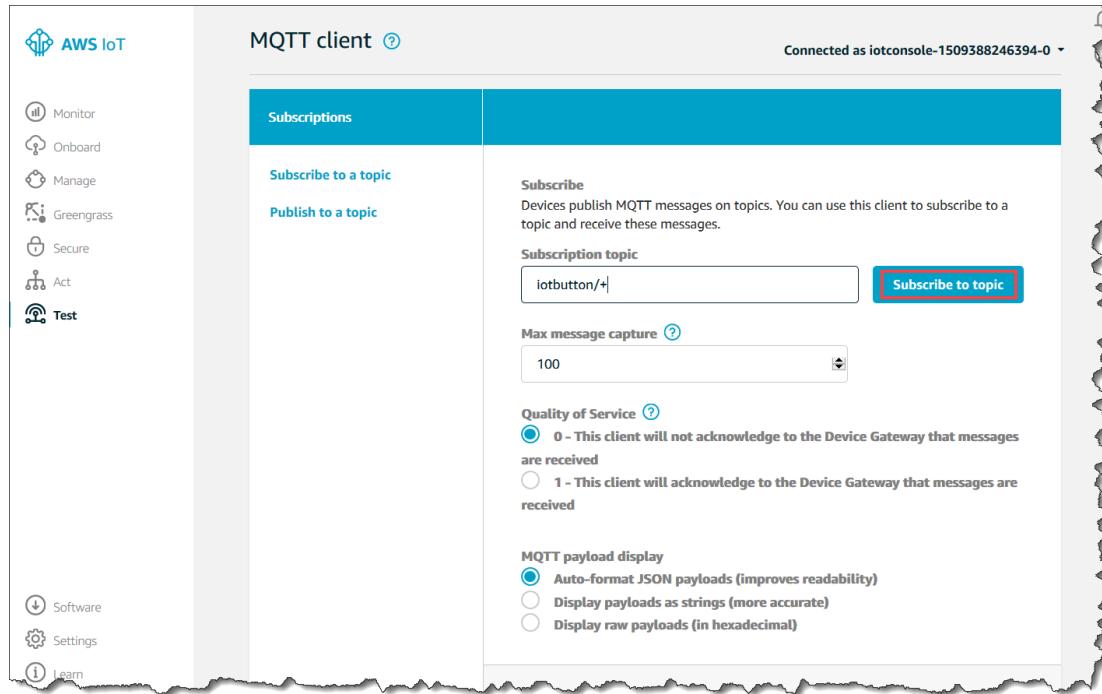
设备会在主题下发布 MQTT 消息。您可以使用 AWS IoT MQTT 客户端订阅这些主题，以查看这些消息。

查看 MQTT 消息：

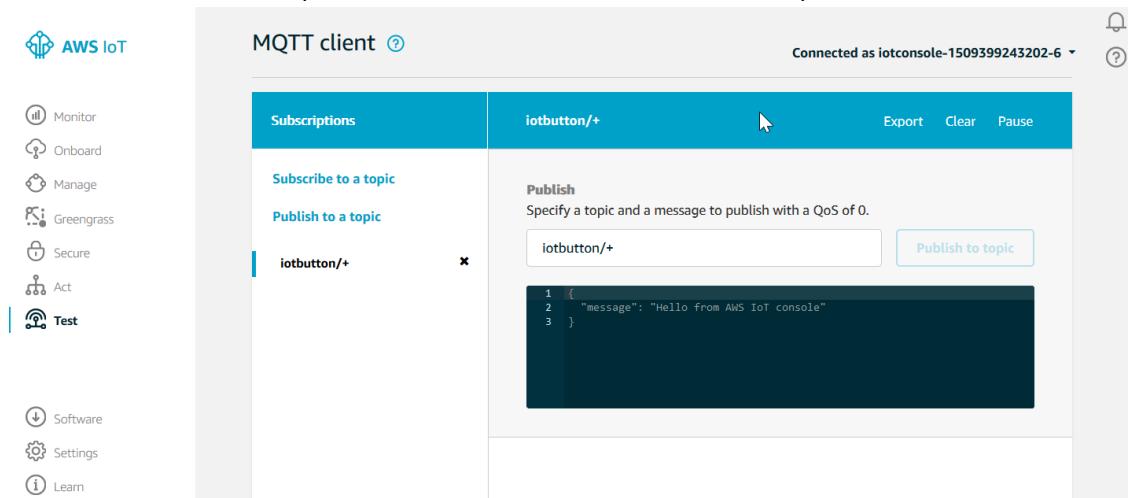
- 在 AWS IoT 控制台中，在左侧导航窗格中选择 Test。



- 订阅您的事物发布的主题。如果是 AWS IoT 按钮，您可以订阅 `iotbutton/+` (请注意，+ 是通配符)。在 Subscribe to topic 中，在 Subscription topic 字段键入 `iotbutton/+`，然后选择 Subscribe to topic。



选择上面的 Subscribe to topic 会导致主题 `iotbutton/+` 显示在 Subscriptions 列中。



- 按您的 AWS IoT 按钮，然后查看 AWS IoT MQTT 客户端中生成的消息。如果您没有按钮，则下一步需要模拟按钮按压。

The screenshot shows the AWS IoT MQTT client interface. On the left sidebar, there are several icons: Monitor, Onboard, Manage, Greengrass, Secure, Act, Test, Software, Settings, and Learn. The main area is titled "MQTT client" and shows a subscription to "iotbutton/+". In the "Publish" section, a message is being prepared with the topic "iotbutton/+". The message content is a JSON object:

```
1 {
2   "message": "Hello from AWS IoT console"
3 }
```

Below the Publish section, a log entry is shown for the topic "iotbutton/G030JF055364XVRB" at Oct 31, 2017 3:24:41 PM -0700. The message content is identical to the one being published.

Note

AWS IoT 按钮常见问题包含有用的按钮 LED 颜色模式信息。

4. 使用 AWS IoT 控制台发布消息：

在 MQTT 客户端页面上，在 Publish 部分的 Specify a topic and a message to publish... 字段中，键入 **iotbutton/ABCDEFG12345**。在消息负载部分，键入以下 JSON：

```
{  
  "serialNumber": "ABCDEFG12345",  
  "clickType": "SINGLE",  
  "batteryVoltage": "2000 mV"  
}
```

选择 Publish to topic。您应该在 AWS IoT MQTT 客户端中看到消息 (选择 Subscription 列中的 iotbutton/+ 来查看消息)。

The screenshot shows the AWS IoT MQTT client interface. On the left sidebar, there are several icons: Act, Test, Software, Settings, and Learn. The main area is titled "MQTT client" and shows a subscription to "iotbutton/+". In the "Publish" section, a message is being prepared with the topic "iotbutton/ABCDEFG12345". The message content is a JSON object:

```
1 {
2   "serialNumber": "ABCDEFG12345",
3   "clickType": "SINGLE",
4   "batteryVoltage": "2000 mV"
5 }
```

The "Publish to topic" button is highlighted with a red box.

配置并测试规则

AWS IoT 规则引擎侦听与规则匹配的传入 MQTT 消息。当收到匹配的消息时，规则会对 MQTT 消息中的数据执行某种操作（例如，将数据写入 Amazon S3 存储桶中、调用 Lambda 函数或向 Amazon SNS 主题发送消息）。在此步骤中，您将创建和配置规则，以将从设备接收的数据发送到 Amazon SNS 主题。具体来说，您将要：

- 创建一个 Amazon SNS 主题。
- 使用手机号码订阅 Amazon SNS 主题。
- 创建规则，以便将从您的设备接收的消息发送到 Amazon SNS 主题。
- 使用您的 AWS IoT 按钮或 MQTT 客户端测试规则。

在此页面的右上角，有一个 Filter View 下拉列表。有关使用 AWS IoT 按钮测试规则的说明，请选择 AWS IoT Button。有关使用 AWS IoT MQTT 客户端测试规则的说明，请选择 MQTT Client。

创建一个 SNS 主题

您将使用 Amazon SNS 控制台创建 Amazon SNS 主题。

Note

Amazon SNS 并未在所有 AWS 区域提供。

1. 打开 [Amazon SNS 控制台](#)。
2. 在左侧窗格选择 Topics。

The screenshot shows the AWS SNS dashboard. On the left sidebar, under the 'Topics' tab, there are three options: Applications, Subscriptions, and Text messaging (SMS). The main content area is titled 'SNS dashboard' and contains a 'Common actions' section with five items: 'Create topic', 'Create platform application', 'Create subscription', 'Publish message', and 'Publish text message (SMS)'. To the right, there is a 'Resources' section showing usage statistics for the us-west-2 region: Topic (0), Subscriptions (0), Applications (0), and Endpoints (0). Below that is a 'More info' section with links to 'Getting started', 'Documentation', 'API reference', 'Forums', and 'Service health'.

3. 选择 Create new topic。

The screenshot shows the AWS SNS Topics page. On the left, there's a navigation sidebar with links: SNS dashboard, Topics (which is selected and highlighted in orange), Applications, Subscriptions, and Text messaging (SMS). The main content area has a title 'Topics' at the top. Below it are several buttons: 'Publish to topic' (blue), 'Create new topic' (red border), 'Actions ▾', and two small icons. A 'Filter' input field is present. A table follows, with columns 'Name' and 'ARN'. The table is currently empty, showing 'Total Items: 0' and 'Selected Items: 0' at the bottom.

4. 键入主题名称和显示名称，然后选择 Create topic。

This is a 'Create new topic' dialog box. It contains two input fields: 'Topic name' with the value 'MyIoTButtonSNSTopic' and 'Display name' with the value 'IoT Button'. At the bottom right, there are 'Cancel' and 'Create topic' buttons, with 'Create topic' being highlighted in blue.

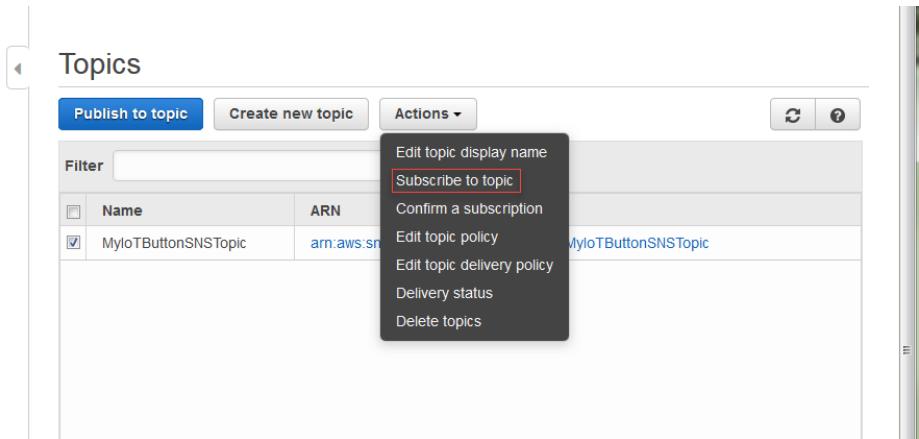
5. 请记下您刚刚创建的主题的 ARN。

The screenshot shows the AWS SNS Topics page again. The navigation sidebar is identical to the previous one. The main content area shows the 'Topics' list. A new row appears in the table, containing 'MyIoTButtonSNSTopic' in the 'Name' column and its corresponding ARN in the 'ARN' column. The ARN starts with 'arn:aws:sns:us-west-2:' followed by a redacted section and ends with ':MyIoTButtonSNSTopic'.

订阅 Amazon SNS 主题

要在您的手机上接收 SMS 消息，您需要订阅 Amazon SNS 主题。

- 在 Amazon SNS 控制台中，选中您刚创建的主题旁边的复选框。从 Actions 菜单中，选择 Subscribe to topic。

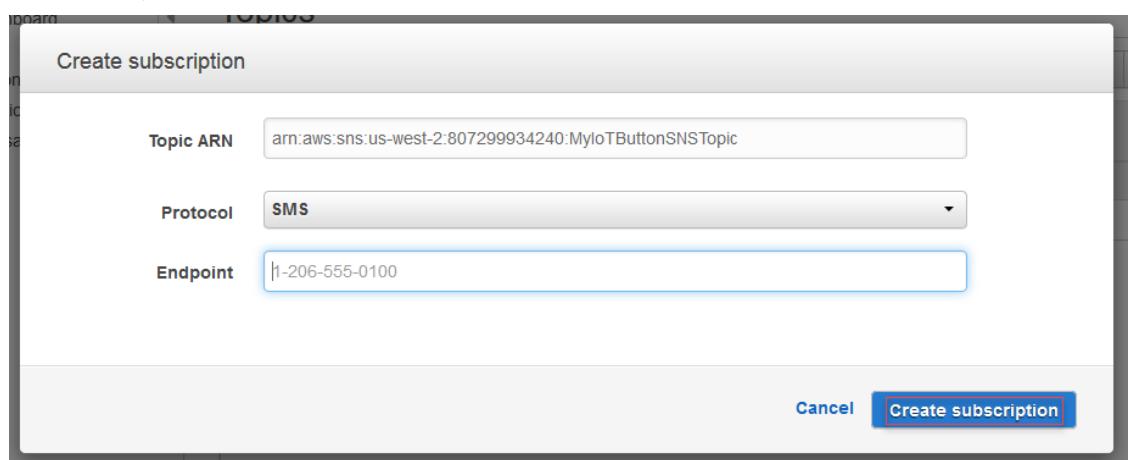


- 在 Create subscription 上，从 Protocol 下拉列表中选择 SMS。

在 Endpoint 字段中，键入启用 SMS 的手机号码，然后选择 Create subscription。

Note

仅使用数字和短划线输入电话号码。

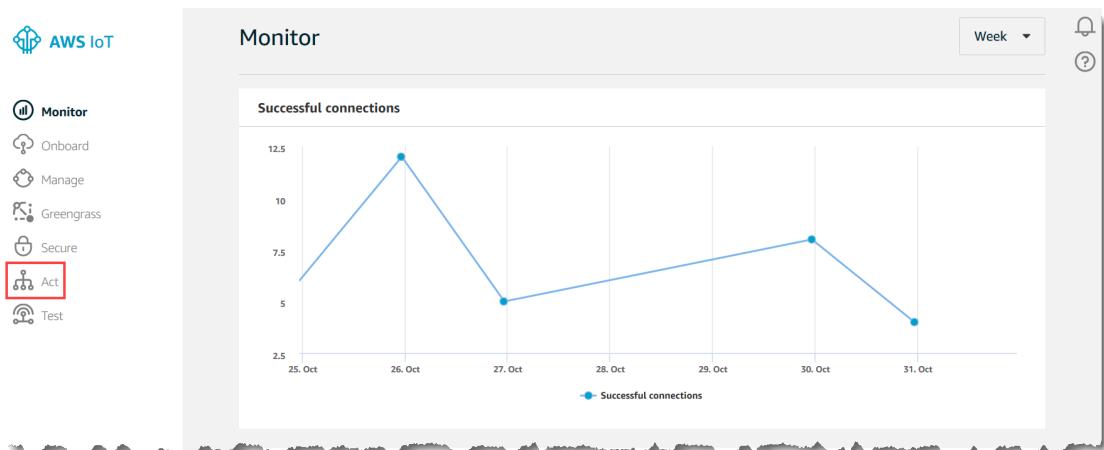


创建规则

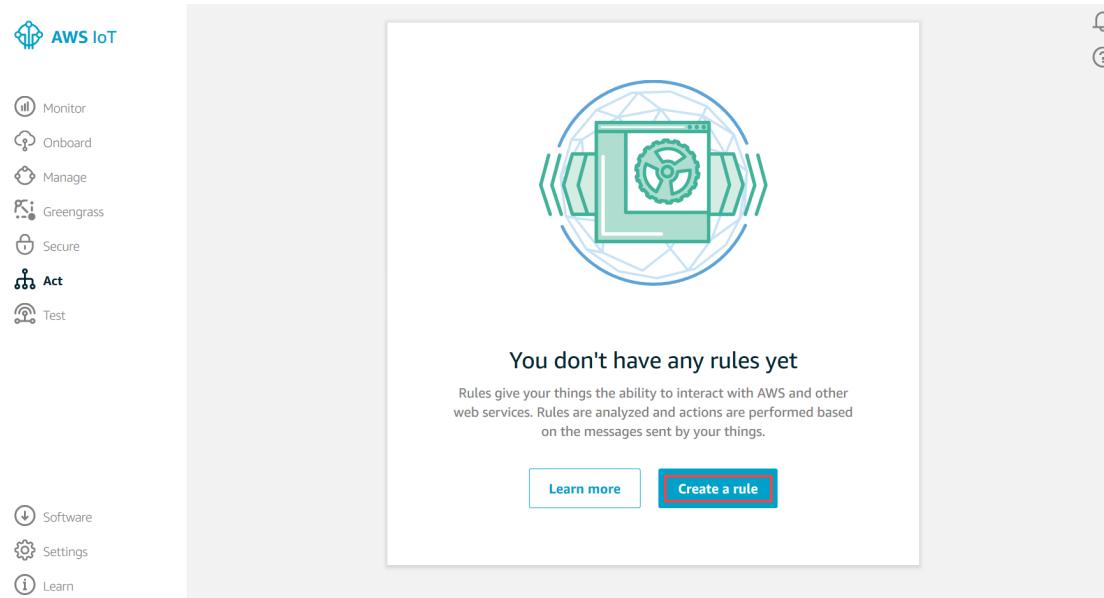
AWS IoT 规则包含主题筛选条件和规则操作，在大多数情况下还包括 IAM 角色。如果在与主题筛选条件相匹配的主题下发布消息，则会触发规则。规则操作定义了规则触发时应执行的操作。IAM 角色包含一项或多项 IAM 策略，这些策略确定规则可以访问哪些 AWS 服务。您可以创建多项规则来侦听一个主题。同样，您也可以创建一个可由多个主题触发的规则。AWS IoT 规则引擎会持续处理在与规则中定义的主题筛选条件相匹配的主题下发布的消息。

在此示例中，您将创建一条规则，以便使用 Amazon SNS 向手机号码发送 SMS 通知。

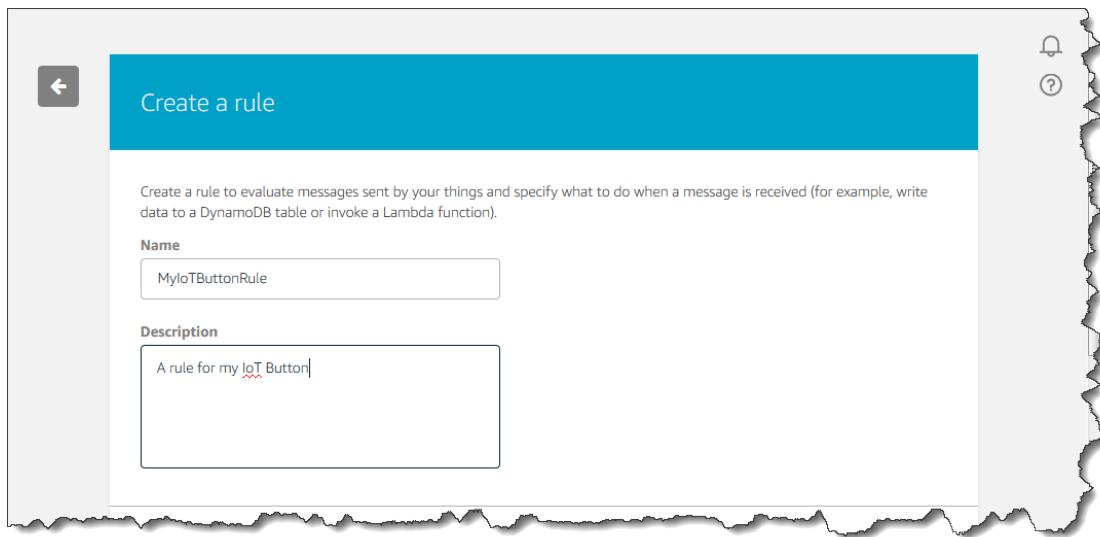
- 在 AWS IoT 控制台中，在左侧导航窗格中选择 Act。



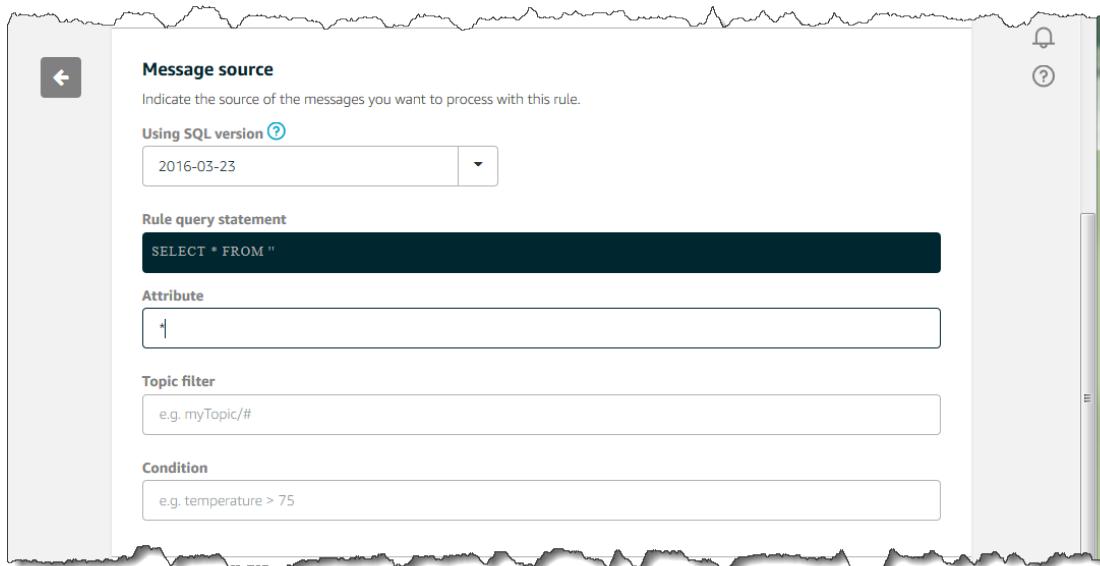
2. 在 Act 页面上，选择 Create a rule。



3. 在 Create a rule 页面上，在 Name 字段中键入规则名称。在 Description 字段键入规则的说明。



4. 向下滚动至消息源。从 Using SQL version 下拉列表中选择最新版本。在 Attribute 字段键入 *。这将指定您要发送触发了规则的完整 MQTT 消息。

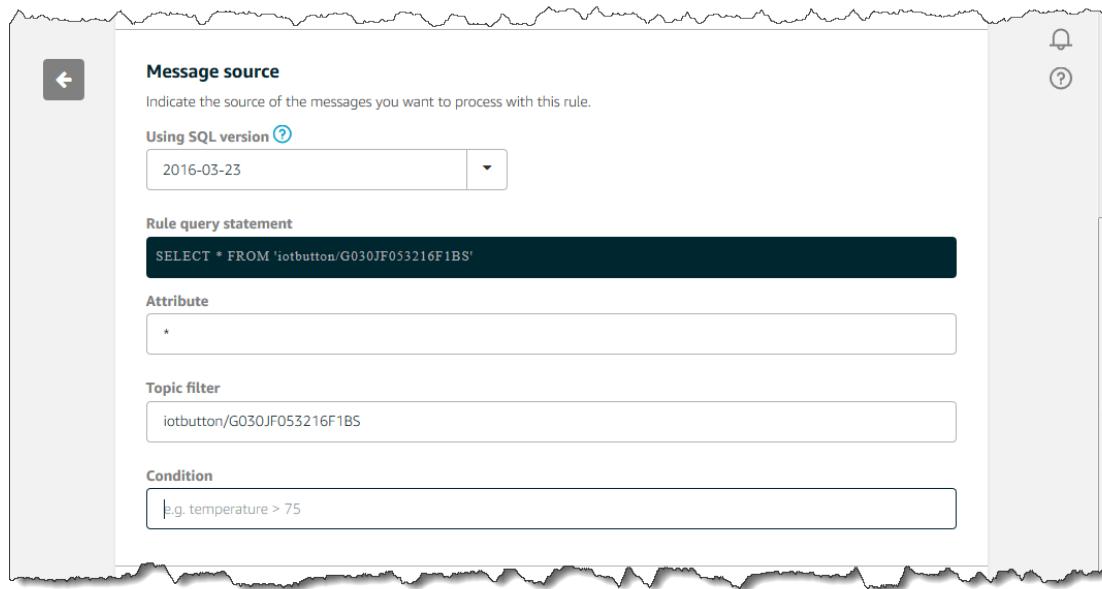


5. 规则引擎使用主题筛选条件来确定收到 MQTT 消息时将触发哪些规则。在 Topic filter 字段键入 **iotbutton/your-button-DSN**。如果您当前未使用 AWS IoT 按钮，请键入 **my/topic** 或规则中使用的主题。

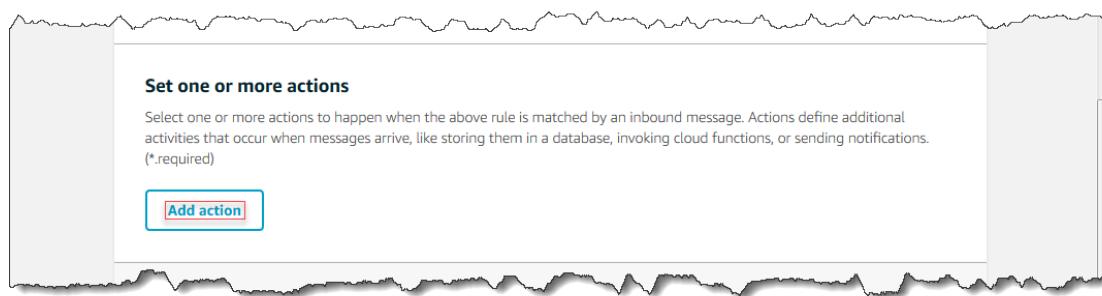
Note

您可以在按钮底部找到 DSN。

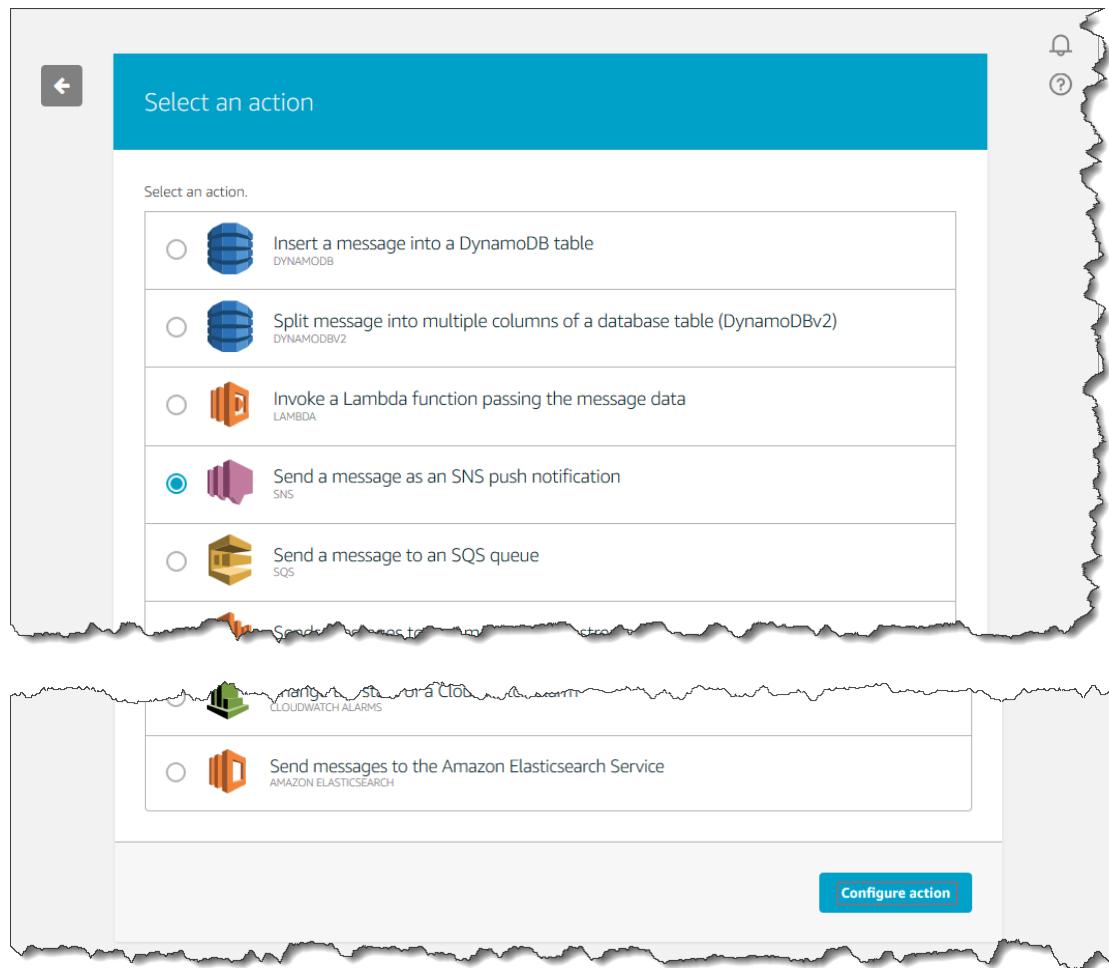
将 Condition 留空。



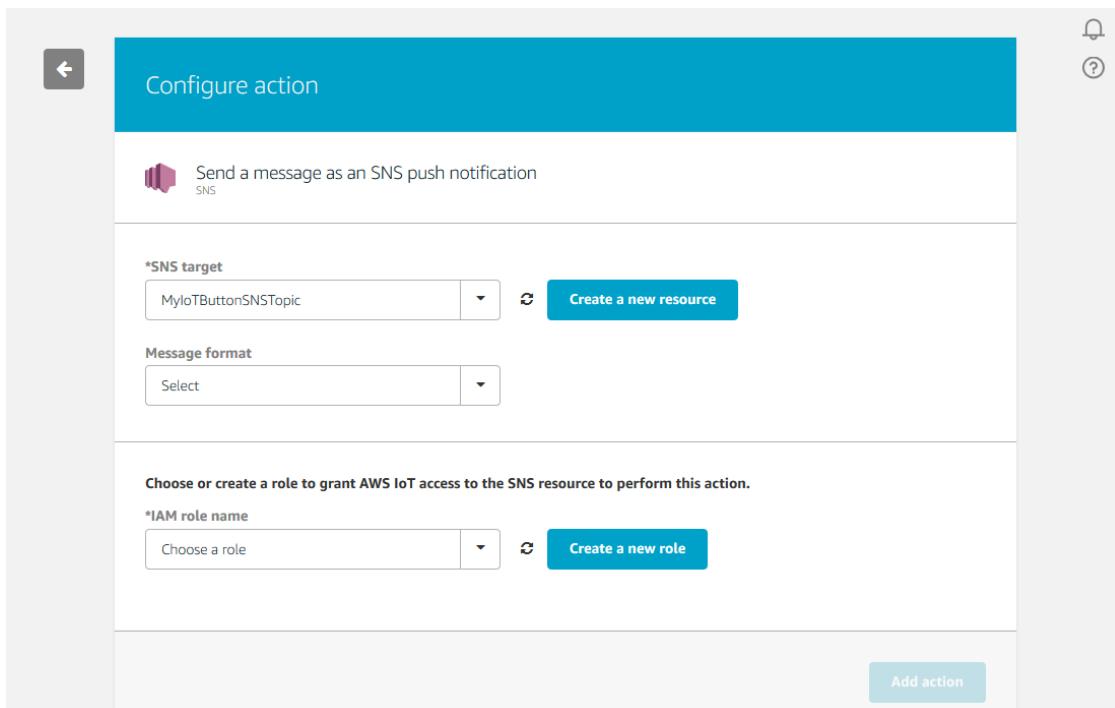
6. 在 Set one or more actions 中，选择 Add action。



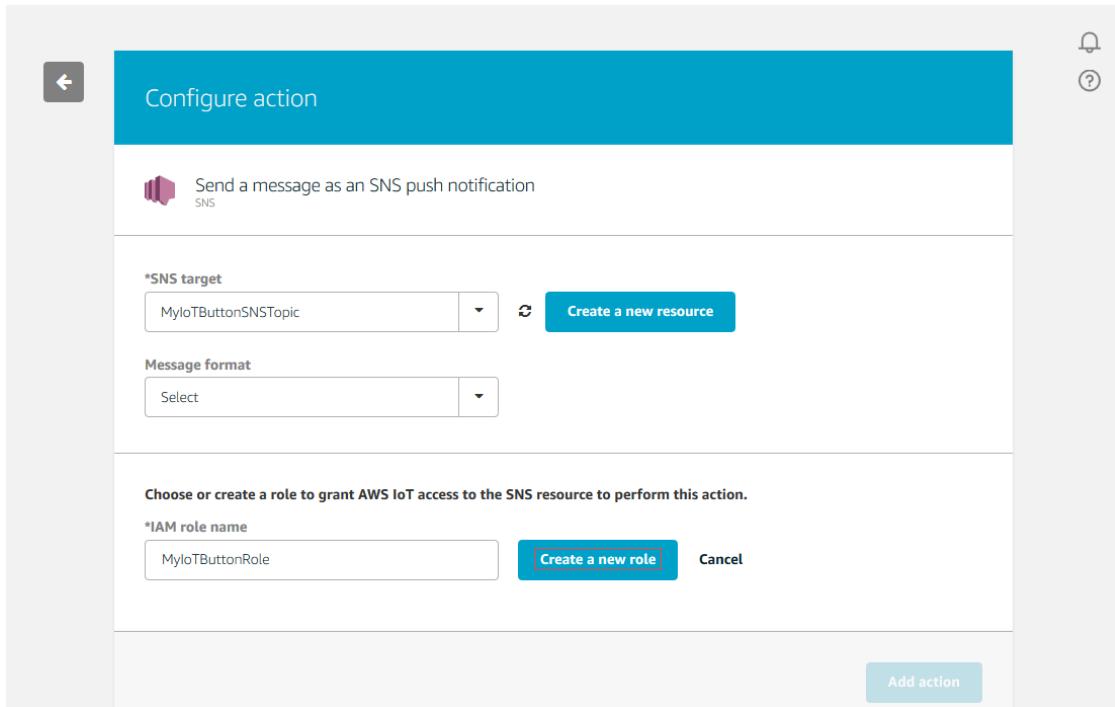
7. 在 Select an action 页面上，选择 Send a message as an SNS push notification，然后选择 Configure action。



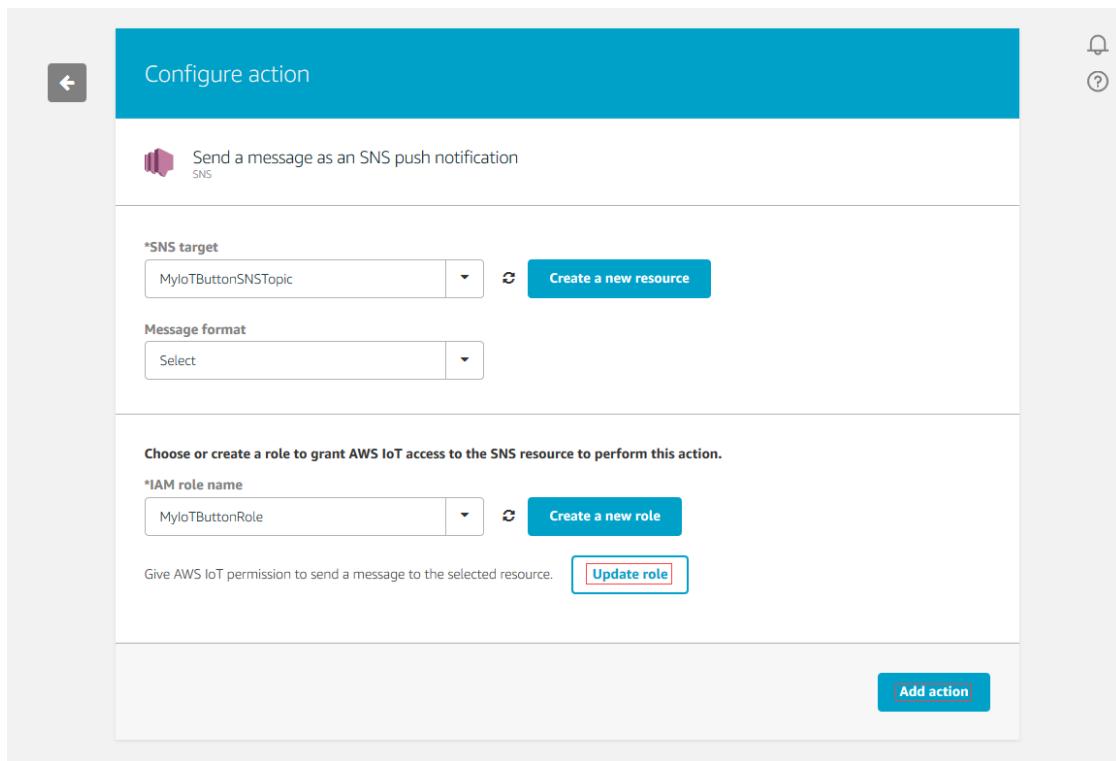
8. 在 Configure action 页面上，从 SNS target 下拉列表中选择您之前创建的 Amazon SNS 主题。



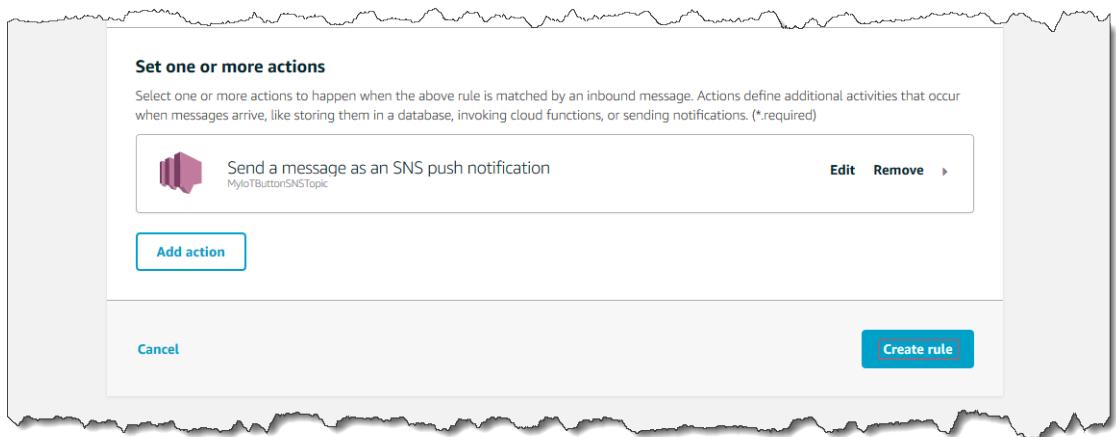
9. 现在，您需要授予 AWS IoT 在规则触发时代表您在 Amazon SNS 主题下发布消息的权限。选择 Create a new role。在 IAM role name 字段输入新角色的名称。在输入名称后，再次选择 Create a new role。从 IAM role name 下拉列表中选择新创建的角色。



10. 选择 Update role 将权限应用到新创建的角色，然后选择 Add action。



11. 在 Create a Rule 页面，选择 Create rule。



有关如何创建规则的更多信息，请参阅 [AWS IoT 规则](#)。

测试 Amazon SNS 规则

您可以使用 AWS IoT 按钮或 AWS IoT MQTT 客户端来测试规则。

AWS IoT 按钮

按您的按钮。您应该会收到短信，显示您的设备的当前电池电量水平 (以及其他信息)。尝试长按 (大约 2 秒) 和快速双按，然后注意生成的消息。

AWS IoT MQTT 客户端

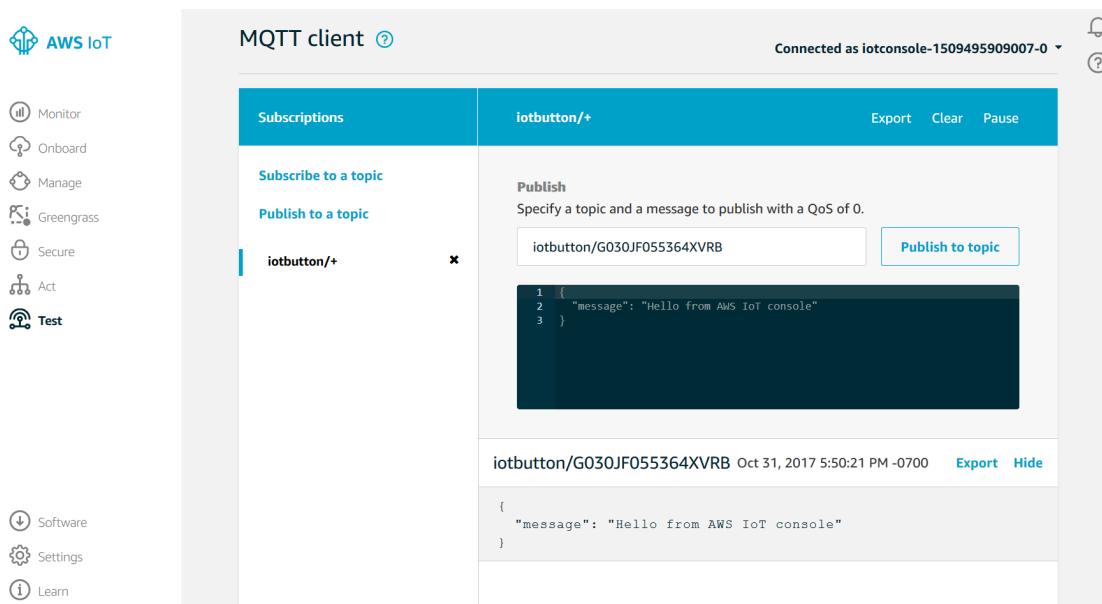
使用 AWS IoT MQTT 客户端测试您的规则：

1. 在 [AWS IoT 控制台](#) 中，在左侧导航窗格中选择 Test。
2. 在 MQTT 客户端页面上，在 Publish 部分的 Specifiy a topic and a message to publish... 字段中，键入 **my/topic** 或规则中使用的主题。在消息负载部分，键入以下 JSON：

```
{  
    "message": "Hello, from AWS IoT console"  
}
```

Note

如果您正在使用按钮，请在 Specifiy a topic and a message to publish... 字段中键入 **iotbutton/your-button-DSN**，而不是 **my/topic**。



3. 选择 Publish to topic。您的手机上应该会收到 Amazon SNS 消息。

恭喜您！您已成功创建并配置了一个规则，它会将从设备收到的数据发送到 Amazon SNS 主题。

后续步骤

有关 AWS IoT 规则的更多信息，请参阅[AWS IoT 规则教程 \(p. 45\)](#)和[AWS IoT 规则 \(p. 144\)](#)。

AWS IoT 按钮快速入门

本部分的两个快速入门为您展示了如何配置和使用 AWS IoT 按钮。您可以使用 AWS Lambda 控制台中的 AWS IoT 按钮向导快速轻松地配置 AWS IoT 按钮。AWS Lambda 控制台中包含一个蓝图，该蓝图可将 AWS IoT 按钮的设置流程自动化，具体操作如下：

- 创建并激活 X.509 证书和私有密钥，以用于在 AWS IoT 中执行身份验证。
- 指导您完成对 AWS IoT 按钮的配置，以便连接到 Wi-Fi 网络。
- 指导您将证书和私有密钥复制到 AWS IoT 按钮。
- 创建 AWS IoT 策略并将其附加到证书，以授予按钮调用 AWS IoT 的权限。
- 创建 AWS IoT 规则，以在您按 AWS IoT 按钮时调用 Lambda 函数。
- 创建 IAM 角色和策略，以允许 Lambda 函数通过 Amazon SNS 发送电子邮件。
- 创建 Lambda 函数，以向 Lambda 函数代码中指定的地址发送电子邮件。

您也可以使用 AWS CloudFormation 模板配置 AWS IoT 按钮。第二个快速入门向您展示如何配置处理 MQTT 消息所需的 AWS IoT 资源，MQTT 消息会在按下 AWS IoT 按钮时使用 AWS CloudFormation 模板发送。



如果您没有按钮，可在此处购买一个。有关 AWS IoT 的更多信息，请参阅[什么是 AWS IoT \(p. 1\)](#)。

主题

- [AWS IoT 按钮向导快速入门 \(p. 31\)](#)
- [AWS IoT 按钮 AWS CloudFormation 快速入门 \(p. 39\)](#)
- [后续步骤 \(p. 44\)](#)

AWS IoT 按钮向导快速入门

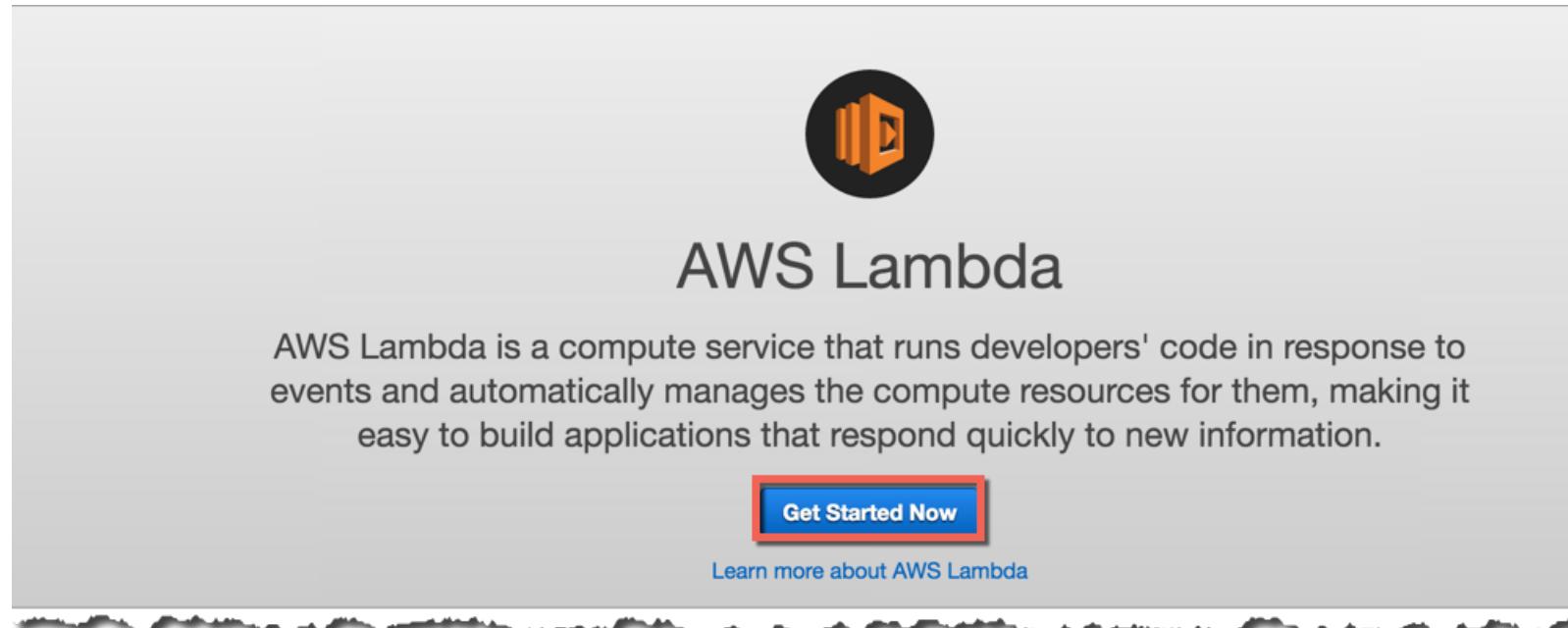
AWS IoT 按钮向导是一个 Lambda 蓝图，因此，您需要登录 AWS Lambda 控制台才能使用它。如果您没有 AWS 账户，则可以按照以下步骤创建一个。

创建 AWS 账户

1. 打开 [AWS 主页](#)，然后选择 Create an AWS Account。
2. 按照在线说明操作。在注册过程中，您会接到来电并且需要使用手机键盘输入 PIN 码。

配置 AWS IoT 按钮

1. 登录 AWS 管理控制台，然后打开 [AWS Lambda 控制台](#)。
2. 如果这是您首次登录 AWS Lambda 控制台，那么您将看到以下页面。选择 Get Started Now 按钮。



如果您使用过 AWS Lambda 控制台，那么您将看到以下页面。选择 Create a Lambda function 按钮。

Lambda

board BETA

tions

Lambda > Functions

You have 32 Lambda function(s) using 1.6 MB of code storage. Choose any Lambda function to view details on invocation requests, duration or memory usage. (Some results may take up to 60 seconds to appear).

Create a Lambda function

Actions ▾

	Function name	Description	Runtime	Code size
<input type="radio"/>	myButtonFunction	An AWS Lambda function that sends an email on the click of an IoT button.	Node.js 4.3	1.7 kB
<input type="radio"/>	michgreFunction	A starter AWS Lambda function.	Node.js 4.3	851 bytes

3. 在 Select blueprint 页面上，从 Runtime 下拉菜单中选择 Node.js 4.3。在“Filter”文本框中，键入 **button**。要选择 **iot-button-email** 蓝图，请双击它或选择 Next 按钮。

> New function

blueprint

ure triggers

ure function

Select blueprint

Blueprints are sample configurations of event sources and Lambda functions. Choose a blueprint that best aligns with your needs, and customize as needed, or skip this step if you want to author a Lambda function and configure an event source separately. Otherwise noted, blueprints are licensed under [CC0](#).

Welcome to AWS Lambda! You can get started on creating your first Lambda function by choosing one of the blueprints below.

Node.js 4.3

button

Viewing

iot-button-email

An AWS Lambda function that sends an email on the click of an IoT button.

nodejs · iot · button

edback

English

© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.

4. 在 Configure triggers 页面上，从 IoT Type 下拉菜单中选择 IoT Button。

键入设备的序列号。在按钮背面，可以找到设备序列号 (DSN)。

选择 Generate certificate and keys。

Note

您只需生成一次证书和私有密钥。然后，您可以在浏览器中导航到 <http://192.168.0.1/index.html> 以开始配置您的按钮。

ia > New function using blueprint iot-button-email

t blueprint

igure triggers

igure function

w

Configure triggers

Configure an optional trigger to automatically invoke your function.



Warning: Altering the description or SQL statement of an existing rule will overwrite it.

IoT Type ⓘ

Device Serial Number ⓘ

Generate certificate and keys ⓘ

使用页面上的这些链接下载设备证书和私有密钥。

[Generate certificate and keys](#) 

We have created the necessary AWS IoT resources (thing, policy, certificate, private key). The remaining resources (rule and action) will be created after your function is created.

Download these resources by clicking the links below. (NOTE: If you are using Internet Explorer or Safari, right click the links to save the files.)

- a. [Your certificate PEM](#)
- b. [Your private key](#)

To configure the AWS IoT Button to use your Wi-Fi and these resources to connect to AWS securely, follow these steps:

1. Place the button into configuration mode by pressing the button down for 5 seconds until it flashes blue.
2. Connect your computer to the button's Wi-Fi network SSID "Button ConfigureMe - FFD", using "5364XVRB" (last 8 digits of device serial number) as the WPA2-PSK password.
3. Click [here](#) (opens in new tab) and use the following information to fill out the form:
 - a. Enter your local network's Wi-Fi SSID and password.
 - b. Select the certificate and private key files that you just downloaded above.
 - c. Your endpoint subdomain is **a182jd32qs965e**.
 - d. Your endpoint region is **us-east-1**.
 - e. Check the box to agree to the terms and conditions.
 - f. Click "configure".
4. Re-connect to your original Wi-Fi network.

The button should stop blinking blue and you will see a white blinking light followed by a greed solid light. Your button is now configured to connect to the internet and AWS! Continue creating your function, and your button will be connected to it automatically.

页面上还提供了关于如何配置 AWS IoT 按钮的说明。在步骤 3 中，您需要选择链接以打开一个网页，以便可以在该网页中将 AWS IoT 按钮连接到网络。在 Wi-Fi Configuration 下，键入所在 Wi-Fi 网络的网络 ID (SSID) 和网络密码。在 AWS IoT Configuration 下，选择之前下载的证书和私有密钥。这会将您的证书和私有密钥复制到 AWS IoT 按钮。选择相应的复选框以表示同意遵守 AWS IoT 按钮条款和条件，然后选择 Configure 按钮。

Button ConfigureMe

Enter the value for any field that you wish to change for device: [REDACTED]

Wi-Fi Configuration:

SSID: Guest

Security: Open Network(No Password)

Password: None (unsecured)

AWS IoT Configuration:

Certificate: certificate.pem

Private Key: private.key

Endpoint Subdomain: A3T2RR9XSNT91O

Endpoint Region: us-west-2

Final Endpoint: .iot.us-west-2.amazonaws.com

By clicking this box, you agree to the [AWS IoT Button Terms and Conditions](#).

此时将显示配置确认页面。

Button ConfigureMe Setup

Thank you for configuring your device.

If you are unable to use your device, please enter configuration mode and try again.

5. 关闭 Configure 选项卡并返回到 AWS Lambda 控制台页面。选择 Enable trigger，然后选择 Next。

在 Configure function 页面上，键入函数名称。此时，系统将为您输入相应的描述、运行时间和 Lambda 函数代码。

da > New function using blueprint iot-button-email

ect blueprint

igure triggers

igure function

ew

Configure function

A Lambda function consists of the custom code you want to execute. [Learn more](#) about Lambda functions.

Name* myIoTButtonFunction

Description An AWS Lambda function that sends an em

Runtime* Node.js 4.3

Lambda function code

Provide the code for your function. Use the editor if your code does not require custom libraries (other than the aws-sdk). If libraries, you can upload your code and libraries as a .ZIP file. [Learn more](#) about deploying Lambda functions.

Code entry type [Edit code inline](#)

We have restored the code from your previous session. Would you like to revert to the last saved state? [Revert now](#).

```
1  /**
2   * This is a sample Lambda function that sends an Email on click of a
3   * button. It creates a SNS topic, subscribes an endpoint (EMAIL)
4   * to the topic and publishes to the topic.
5   *
6   * Follow these steps to complete the configuration of your function:
7   *
8   * 1. Update the EMAIL variable with your email address.
9   * 2. Enter a name for your execution role in the "Role name" field.
10  * Your function's execution role needs specific permissions for SNS operations
11  * to send an email. We have pre-selected the "AWS IoT Button permissions"
12  * policy template that will automatically add these permissions.
13  */
14
15 const EMAIL = 'my_email@example.com'; // TODO change me
```

在 Lambda 函数代码中，将示例电子邮件地址替换为您自己的电子邮件地址。

```
1  /**
2  * This is a sample Lambda function that sends an Email on click of a
3  * button. It creates a SNS topic, subscribes an endpoint (EMAIL)
4  * to the topic and publishes to the topic.
5  *
6  * Follow these steps to complete the configuration of your function:
7  *
8  * 1. Update the EMAIL variable with your email address.
9  * 2. Enter a name for your execution role in the "Role name" field.
10 * Your function's execution role needs specific permissions for SNS operations
11 * to send an email. We have pre-selected the "AWS IoT Button permissions"
12 * policy template that will automatically add these permissions.
13 */
14
15 const EMAIL = 'my_email@example.com'; // TODO change me
16
17 const AWS = require('aws-sdk');
18 const SNS = new AWS.SNS({ apiVersion: '2010-03-31' });
19
20 function findExistingSubscription(topicArn, nextToken, cb) {
21     const params = {
22         TopicArn: topicArn,
23         NextToken: nextToken || null,
24     };
25     SNS.listSubscriptionsByTopic(params, (err, data) => {
26         if (err) {
```

在 Lambda function handler and role 部分，从 Role 下拉菜单中选择 Create new role from template(s)。为角色键入唯一名称。

Lambda function handler and role

Handler* index.handler

Role* Create new role from template(s) 

Lambda will automatically create a role with permissions from the selected policy templates. Note that basic Lambda permissions (logging to CloudWatch) will automatically be added. If your function accesses a VPC, VPC permissions will also be added.

Role name myIoTButtonRole 

Policy templates  

在页面底部，选择 Next。

查看 Lambda 函数的设置，然后选择 Create function。

new function using blueprint iot-button-email

Review

Please review your Lambda function details. You can go back to edit changes for each section. When you are ready, click **Create function** to complete the setup process.

Triggers

Lambda function

Name myButtonFunction

Description An AWS Lambda function that sends an email on the click of an IoT button.

Runtime Node.js 4.3

Handler index.handler

Role name myNewRole

Policy templates AWS IoT Button permissions

Memory (MB) 128

Timeout 3

VPC No VPC

Cancel

Previous

Create

此时您应看到一个确认已创建 Lambda 函数的页面：

The screenshot shows the AWS Lambda console interface. At the top, there are tabs for 'Services' and 'Edit'. Below that, a breadcrumb navigation shows 'Functions > myButtonFunction'. There are two buttons: 'Test' (highlighted in blue) and 'Actions'. A message below the buttons states: 'Success! Your Lambda function "myButtonFunction" has been successfully created and configured with IoT: iotbutton_... as a trigger.' Below this, there are three tabs: 'Configuration' (disabled), 'Triggers' (selected and highlighted in orange), and 'Monitoring'. Under the 'Triggers' tab, it shows a single trigger named 'AWS IoT: iotbutton_G030JF055364XVRB'. It provides details: 'arn:aws:iot:us-east-1:...:rule/iotbutton_...' and 'Rule Description: Event source for your IoT Button to Lambda'. The SQL Statement is listed as 'SELECT * FROM "iotbutton/"'. At the bottom left, there is a link labeled 'trigger'.

6. 要测试 Lambda 函数，请选择 Test 按钮。大约一分钟后，您会收到一封主题行内容为 AWS Notification - Subscription Confirmation 的电子邮件。选择电子邮件中的链接，以确认 Lambda 函数已创建 SNS 主题订阅。当 AWS IoT 收到来自您的按钮的消息后，它会向 Amazon SNS 发送一条消息。Lambda 函数创建 Amazon SNS 主题订阅时使用的是您在代码中添加的电子邮件地址。当 Amazon SNS 收到与该 Amazon SNS 主题有关的消息后，它会将消息转发到您用于进行订阅的电子邮件地址。

按您的按钮以向 AWS IoT 发送消息。该消息将导致触发 Lambda 规则，进而调用 Lambda 函数。Lambda 函数会检查您的 SNS 主题是否存在。然后，Lambda 函数会将消息内容发送至 Amazon SNS 主题。然后，Amazon SNS 会将消息转发到您在 Lambda 函数代码中指定的电子邮件地址。

AWS IoT 按钮 AWS CloudFormation 快速入门

在按下 AWS IoT 按钮时，它会将与按钮有关的基本信息发送至 Amazon SNS 主题。随后主题会将该信息以邮件消息的方式转发给您。本快速入门向您展示如何使用 AWS CloudFormation 模板配置 AWS IoT 按钮。

您需要 AWS 账户和 AWS IoT 按钮才能完成本快速入门中的步骤。

1. 使用 AWS IoT 控制台创建 AWS IoT 证书：
 - a. 打开 [AWS IoT 控制台](#)。
 - b. 出现 Welcome 页面时，请选择 Get started。
 - c. 在 AWS 区域选择器中，选择您要创建 AWS IoT 证书的 AWS 区域（例如，美国东部（弗吉尼亚北部）。您将在同一 AWS 区域创建所有支持的 AWS 资源（其他 AWS IoT 资源和 Amazon SNS 资源）。

- d. 在 Dashboard 上，在左侧导航窗格中选择 Security，然后选择 Certificates。
 - e. 在 Certificates 窗格中选择 Create。
 - f. 选择 One-click certificate creation - Create certificate。
 - g. 在 Certificate created 页面上，为证书、私有密钥和 AWS IoT 的根 CA 选择 Download，将它们保存到您的计算机中，然后选择 Activate 继续操作。
 - h. 选择完成。
 - i. 在 Certificates 页面上，选择您刚创建的证书。
 - j. 在 Details 信息中，记下证书 ARN 值（例如，arn:aws:iot:region-ID:account-ID:cert/random-ID）。您需要在本程序的后面部分中使用它。
2. 在 <https://console.aws.amazon.com/cloudformation/> 使用 AWS CloudFormation 控制台创建 AWS IoT 资源、Amazon SNS 资源和 IAM 角色：
- a. 将 AWS CloudFormation 模板文件命名为 AWSIoTButtonQuickStart.template 并保存到您的计算机。

```
{  
    "AWSTemplateFormatVersion": "2010-09-09",  
    "Description": "Creates required AWS resources to allow an AWS IoT button to send  
information through an Amazon Simple Notification Service (Amazon SNS) topic to an  
email address.",  
    "Parameters": {  
        "IoTButtonDSN": {  
            "Type": "String",  
            "AllowedPattern": "G030[A-Z][A-Z][0=9][0-9][0-9][0-5][0-9][1-7][0-9A-HJ-NP-X]  
[0-9A-HJ-NP-X][0-9A-HJ-NP-X][0-9A-HJ-NP-X]",  
            "Description": "The device serial number (DSN) of the AWS IoT Button. This can  
be found on the back of the button. The DSN must match the pattern of 'G030[A-Z]  
[A-Z][0=9][0-9][0-9][0-5][0-9][1-7][0-9A-HJ-NP-X][0-9A-HJ-NP-X][0-9A-  
HJ-NP-X]'. "  
        },  
        "CertificateARN": {  
            "Type": "String",  
            "Description": "The Amazon Resource Name (ARN) of the existing AWS IoT  
certificate."  
        },  
        "SNSTopicName": {  
            "Type": "String",  
            "Default": "aws-iot-button-sns-topic",  
            "Description": "The name of the Amazon SNS topic for AWS CloudFormation to  
create."  
        },  
        "SNSTopicRoleName": {  
            "Type": "String",  
            "Default": "aws-iot-button-sns-topic-role",  
            "Description": "The name of the IAM role for AWS CloudFormation to create. This  
IAM role allows AWS IoT to send notifications to the Amazon SNS topic."  
        },  
        "EmailAddress": {  
            "Type": "String",  
            "Description": "The email address for the Amazon SNS topic to send information  
to."  
        }  
    },  
    "Resources": {  
        "IoTThing": {  
            "Type": "AWS::IoT::Thing",  
            "Properties": {  
                "ThingName": {  
                    "Fn::Join": [ "",  
                    [ "iotbutton_"  
                ]  
            }  
        }  
    }  
}
```

```
        { "Ref": "IoTButtonDSN" }
    ]
]
}
},
"IoTPolicy": {
    "Type" : "AWS::IoT::Policy",
    "Properties": {
        "PolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": "iot:Publish",
                    "Effect": "Allow",
                    "Resource": {
                        "Fn::Join": [ "", [
                            "arn:aws:iot:",
                            { "Ref": "AWS::Region" },
                            ":",
                            { "Ref": "AWS::AccountId" },
                            ":topic/iotbutton/",
                            { "Ref": "IoTButtonDSN" }
                        ] ]
                    }
                }
            ]
        }
    }
},
"IoTPolicyPrincipalAttachment": {
    "Type": "AWS::IoT::PolicyPrincipalAttachment",
    "Properties": {
        "PolicyName": {
            "Ref": "IoTPolicy"
        },
        "Principal": {
            "Ref": "CertificateARN"
        }
    }
},
"IoTThingPrincipalAttachment": {
    "Type" : "AWS::IoT::ThingPrincipalAttachment",
    "Properties": {
        "Principal": {
            "Ref": "CertificateARN"
        },
        "ThingName": {
            "Ref": "IoTThing"
        }
    }
},
"SNSTopic": {
    "Type": "AWS::SNS::Topic",
    "Properties": {
        "DisplayName": "AWS IoT Button Press Notification",
        "Subscription": [
{
            "Endpoint": {
                "Ref": "EmailAddress"
            },
            "Protocol": "email"
        }
        ],
    }
},
```

```
        "TopicName": {
            "Ref": "SNSTopicName"
        }
    },
    "SNSTopicRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "iot.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "sns:Publish",
                                "Resource": {
                                    "Fn::Join": [ "", [
                                        "arn:aws:sns:",
                                        { "Ref": "AWS::Region" },
                                        ":",
                                        { "Ref": "AWS::AccountId" },
                                        ":",
                                        { "Ref": "SNSTopicName" }
                                    ] ]
                                }
                            }
                        ]
                    },
                    "PolicyName": {
                        "Ref": "SNSTopicRoleName"
                    }
                }
            ]
        }
    },
    "IoTTopicRule": {
        "Type": "AWS::IoT::TopicRule",
        "Properties": {
            "RuleName": {
                "Fn::Join": [ "", [
                    "iotbutton_",
                    { "Ref": "IoTButtonDSN" }
                ] ]
            },
            "TopicRulePayload": {
                "Actions": [
                    {
                        "Sns": {
                            "TopicArn": {
                                "Fn::GetAtt": [
                                    "SNSTopicName",
                                    "Arn"
                                ]
                            }
                        }
                    }
                ]
            }
        }
    }
},
```

```
        "RoleArn": {
            "Fn::GetAtt": [ "SNSTopicRole", "Arn" ]
        },
        "TargetArn": {
            "Ref": "SNSTopic"
        }
    }
],
"AwsIoTSqlVersion": "2015-10-08",
"RuleDisabled": false,
"Sql": {
    "Fn::Join": [
        [
            "SELECT * FROM 'iotbutton/'",
            { "Ref": "IoTButtonDSN" },
            "'"
        ]
    ]
}
}
```

- b. 通过以下网址打开 AWS CloudFormation 控制台：<https://console.aws.amazon.com/cloudformation/>。
 - c. 请确保 AWS 区域选择器显示您创建 AWS IoT 证书的区域（例如，美国东部（弗吉尼亚北部））。
 - d. 选择 Create Stack。
 - e. 在 Select Template 页面上，选择 Upload a template to Amazon S3，然后选择 Browse。
 - f. 选中您之前保存的 AWSIoTButtonQuickStart.template 文件，选择 Open，然后选择 Next。
 - g. 在 Specify Details 页面上，在 Stack name 中键入该 AWS CloudFormation 堆栈的名称（例如 MyAWSIoTButtonStack）。
 - h. 对于 CertificateARN，键入您之前记下的 AWS IoT 证书的 Amazon 资源名称 (ARN) (证书 ARN 值)。
 - i. 在 EmailAddress 中键入您的电子邮件地址。
 - j. 在 IoTButtonDSN 中键入设备序列号 (DSN)。您可在 AWS IoT 按钮的背面找到该序列号（例如，G030JF051234A5BC）。
 - k. SNSTopicName 和 SNSTopicRoleName 可以保留默认值，您也可指定不同的 Amazon SNS 主题名称和相关联的 IAM 角色名称。例如，如果您计划设置更多 AWS IoT 按钮，您可能需要更改这些值。选择 Next。
 - l. Options 页面中不需要进行执行操作。选择 Next。
 - m. 在 Review 页面上，选择 I acknowledge that AWS CloudFormation might create IAM resources，然后选择 Create。
 - n. 如果 MyAWSIoTButtonStack 显示 CREATE_COMPLETE，请在您的电子邮件收件箱中查找主题为“AWS IoT Button Press Notification”的消息。选择邮件消息正文中的 Confirm subscription 链接。
3. 使用您之前创建的私有密钥和证书，按照 [Configure Your Device](#) 中的步骤设置您的 AWS IoT 按钮。
 4. 在设置之后，请按一次按钮。白灯闪烁几下，然后绿灯会亮起一段时间。不久后，您应收到主题行中包含“AWS IoT Button Press Notification”的邮件消息。您将在邮件正文中看到按钮发送的信息。
 5. 在完成试用后，您可以进行清理 AWS CloudFormation 模板创建的 AWS 资源。为此，请返回 AWS CloudFormation 控制台并删除 MyAWSIoTButtonStack。在删除 MyAWSIoTButtonStack 后，按照以下步骤删除 AWS IoT 证书：
 - a. 返回 AWS IoT 控制台。
 - b. 在资源列表中，在代表 AWS IoT 证书的栏中（带有握手图标的栏）选中复选框。

- c. 对于 Actions，选择 Deactivate，然后确认。
- d. 保持选中代表 AWS IoT 证书的栏，为 Actions 选择 Delete，然后确认。
- e. 您之前下载的私有密钥和证书将失效，您现在可以将它们从计算机中删除。

后续步骤

要了解有关用于设置按钮的 Lambda 蓝图的更多信息，请参阅 [Getting Started with AWS IoT](#)。要学习如何通过 AWS IoT 按钮使用 AWS CloudFormation，请参阅 <http://docs.aws.amazon.com/iot/latest/developerguide/iot-button-cloud-formation.html>

AWS IoT 规则教程

本指南中的教程将向您介绍如何创建和测试 AWS IoT 规则。如果您尚未完成 [AWS IoT 入门教程 \(p. 4\)](#)，建议您先学习该教程。它将向您展示如何创建 AWS 账户以及如何将您的设备连接到 AWS IoT。

AWS IoT 规则由 SQL SELECT 语句、主题筛选条件和规则操作组成。设备通过将消息发布到 MQTT 主题来向 AWS IoT 发送信息。利用 SQL SELECT 语句，您可以从传入的 MQTT 消息提取数据。AWS IoT 规则的主题筛选条件用于指定一个或多个 MQTT 主题。当与主题筛选条件匹配的主题收到 MQTT 消息时，规则将被触发。借助规则操作，您可以获取从 MQTT 消息提取的信息并将其发送到其他 AWS 服务。规则操作是针对 Amazon DynamoDB、AWS Lambda、Amazon SNS 和 Amazon S3 等 AWS 服务定义的。使用 Lambda 规则，您可以调用其他 AWS 服务或第三方 Web 服务。有关规则操作的完整列表，请参阅 [AWS IoT 规则操作 \(p. 152\)](#)。

这些教程假定您正在使用 AWS IoT 按钮，并将使用 `iotbutton/+` 作为规则中的主题筛选条件。如果您没有 AWS IoT 按钮，[可在此处购买一个](#)。

或者，您可以通过使用 MQTT 客户端来模拟 AWS IoT 按钮，例如 [AWS IoT 控制台](#) 中的 AWS IoT MQTT 客户端。要模拟 AWS IoT 按钮，请在 `iotbutton/ABCDEFG12345` 主题下发布一条类似的消息。“/”后面的数字是随机的。它用作该按钮的序列号。

您还可以使用自己的设备，但需要了解您的设备会将消息发布到哪个 MQTT 主题，以便可以将其指定为规则中的主题筛选条件。有关更多信息，请参阅 [AWS IoT 规则 \(p. 144\)](#)。

AWS IoT 按钮会发送一个类似于以下示例的 JSON 有效负载：

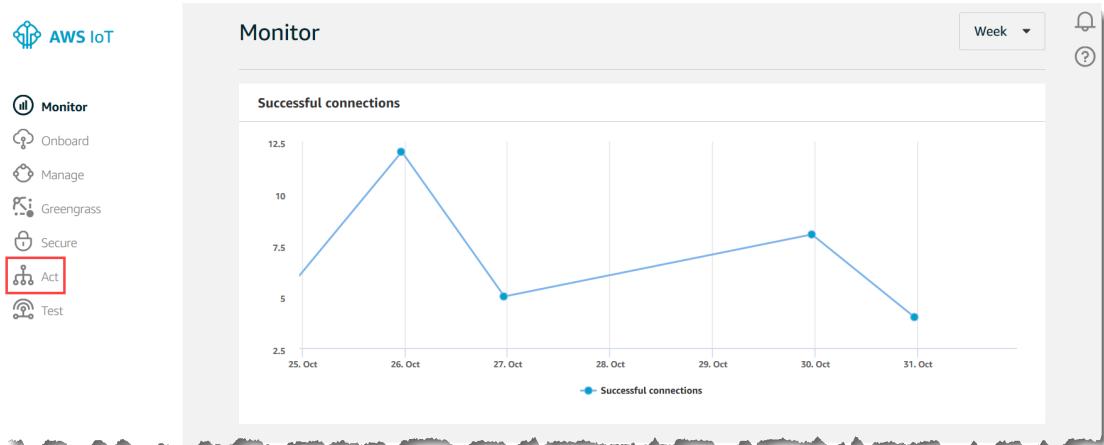
```
{  
    "serialNumber" : "ABCDEFG12345",  
    "batteryVoltage" : "2000mV",  
    "clickType" : "SINGLE"  
}
```

创建 DynamoDB 规则

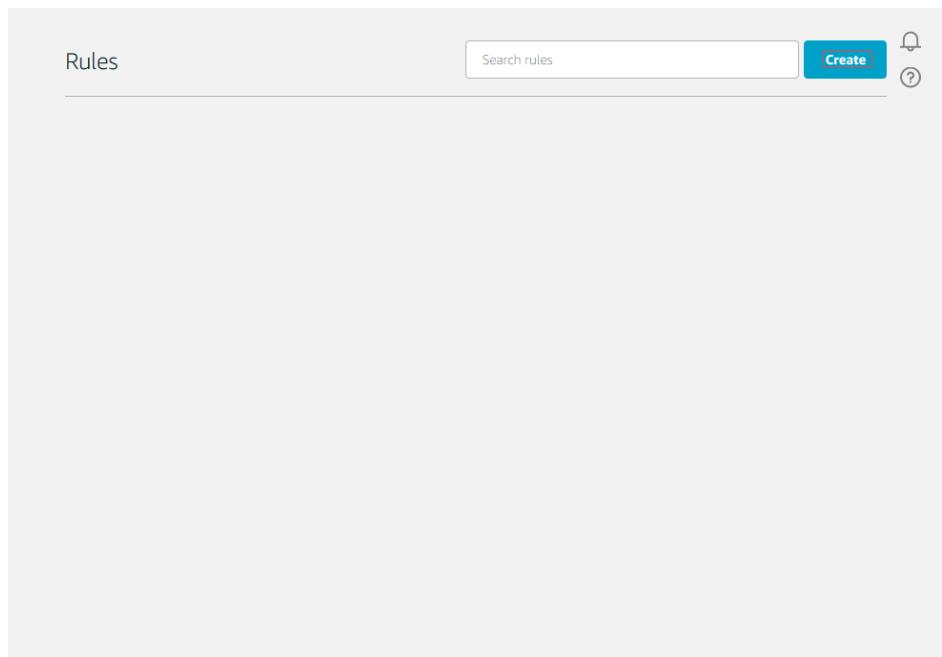
借助 DynamoDB 规则，您可以从传入的 MQTT 消息提取信息，并将其写入 DynamoDB 表。

创建 DynamoDB 规则：

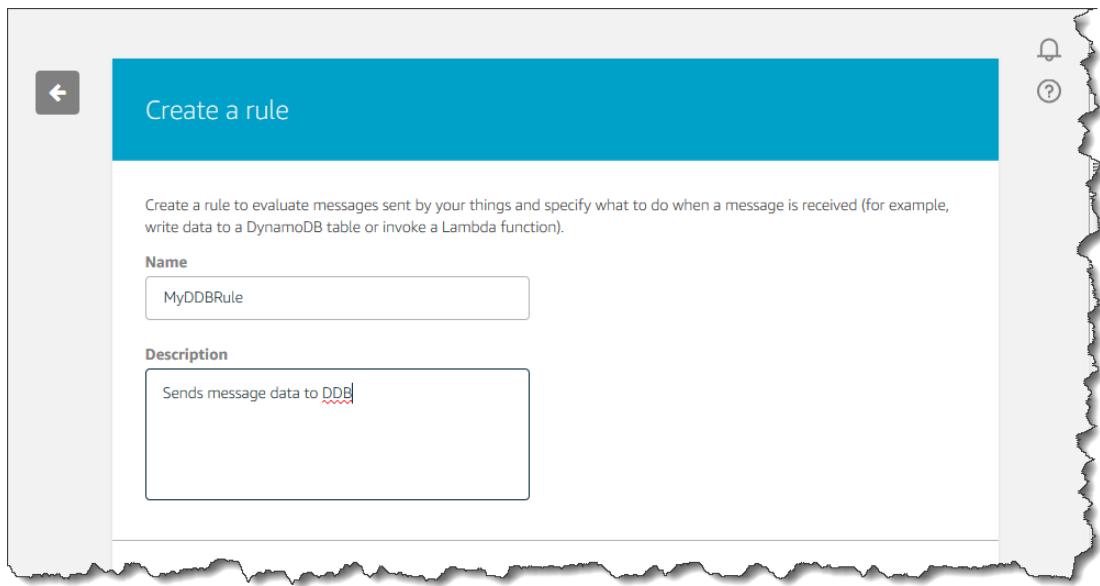
1. 在 [AWS IoT 控制台](#) 中，在左侧导航窗格中选择 Rules。



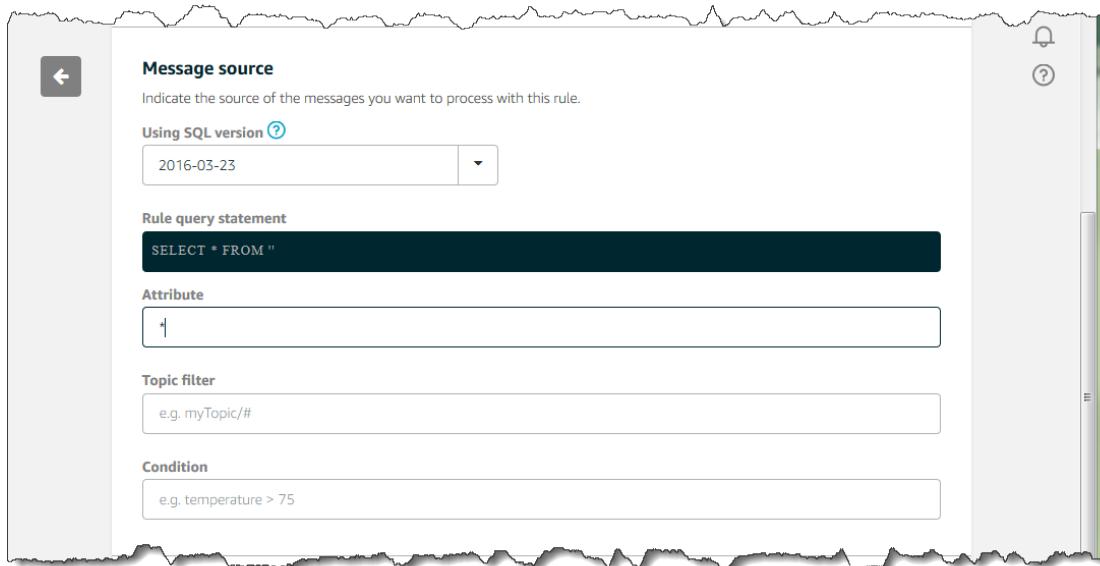
2. 在 Rules 页面，选择 Create。



3. 在 Create a rule 页面上，在 Name 字段中键入规则名称。在 Description 字段键入规则的说明。



4. 向下滚动至消息源。从 Using SQL version 下拉列表中选择最新版本。在 Attribute 字段键入 *。这将指定您要发送触发了规则的完整 MQTT 消息。

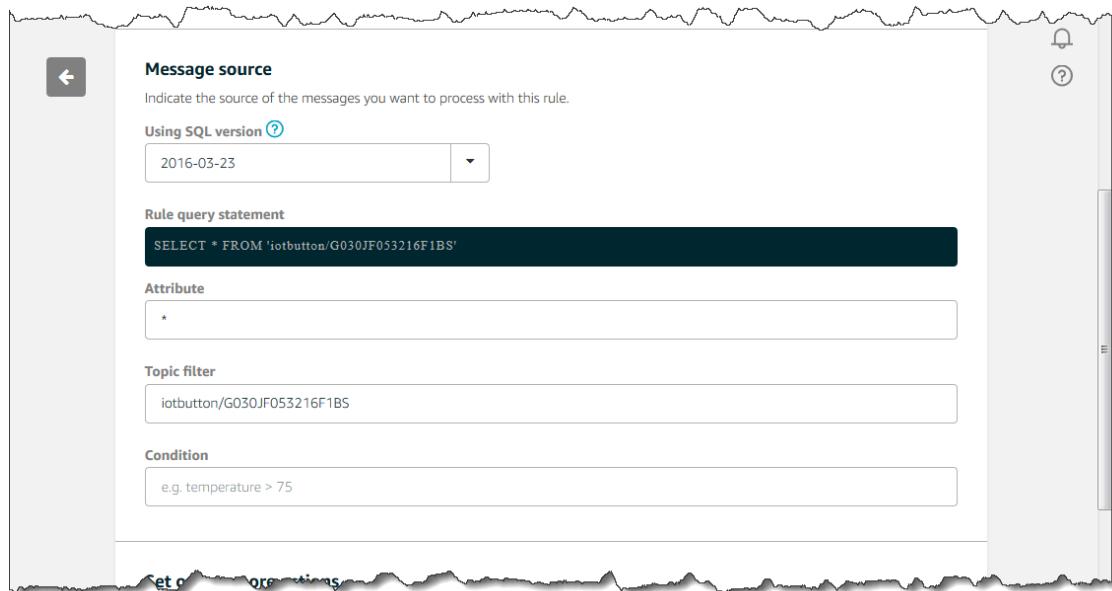


5. 规则引擎使用主题筛选条件来确定收到 MQTT 消息时将触发哪些规则。在 Topic filter 字段键入 **iotbutton/your-button-DSN**。如果您当前未使用 AWS IoT 按钮，请键入 **my/topic** 或规则中使用的主题。

Note

您可以在按钮底部找到 DSN。

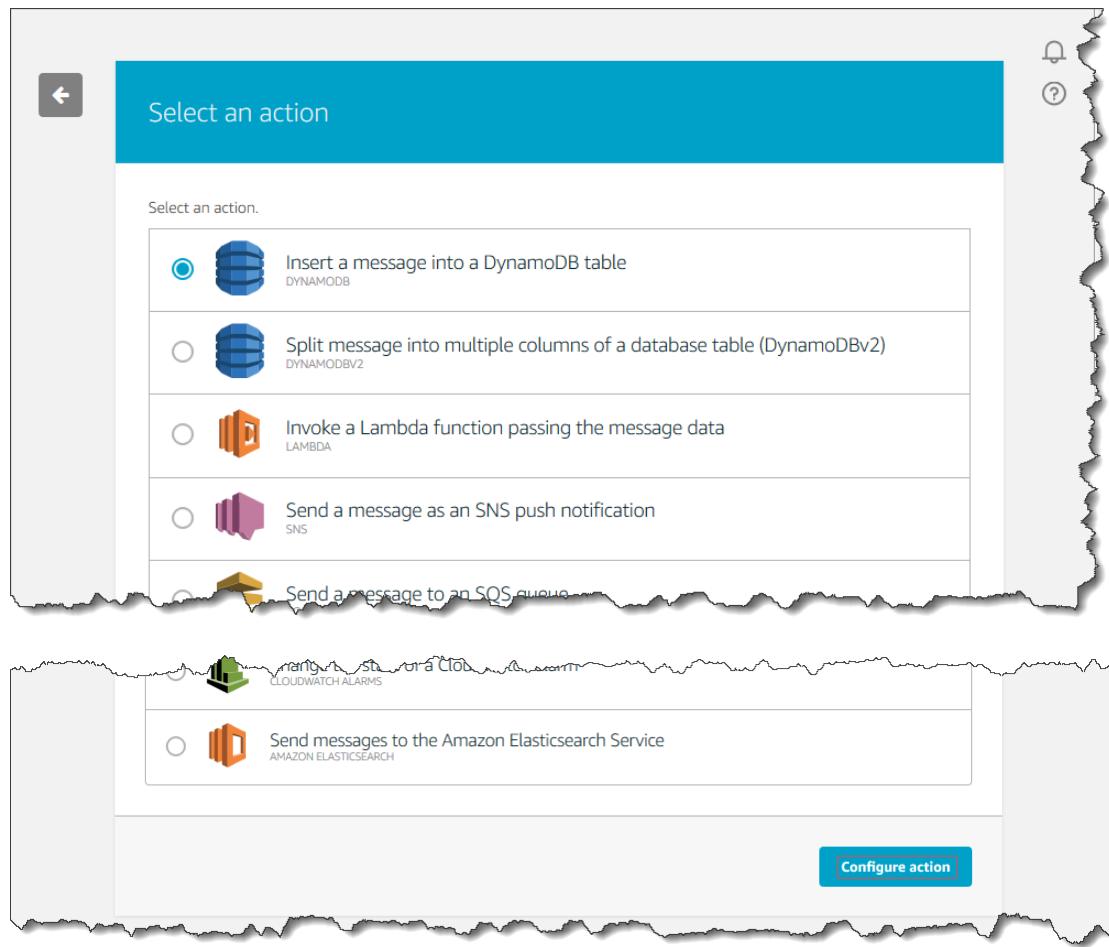
将 Condition 留空。



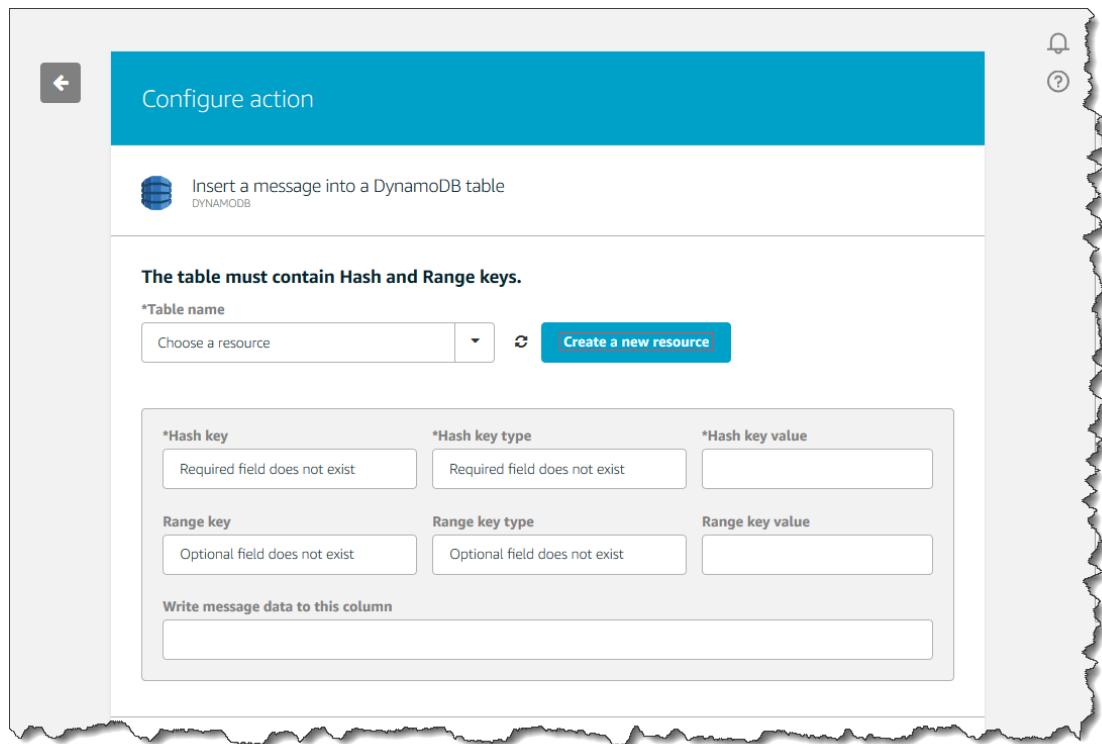
6. 在 Set one or more actions 中，选择 Add action。



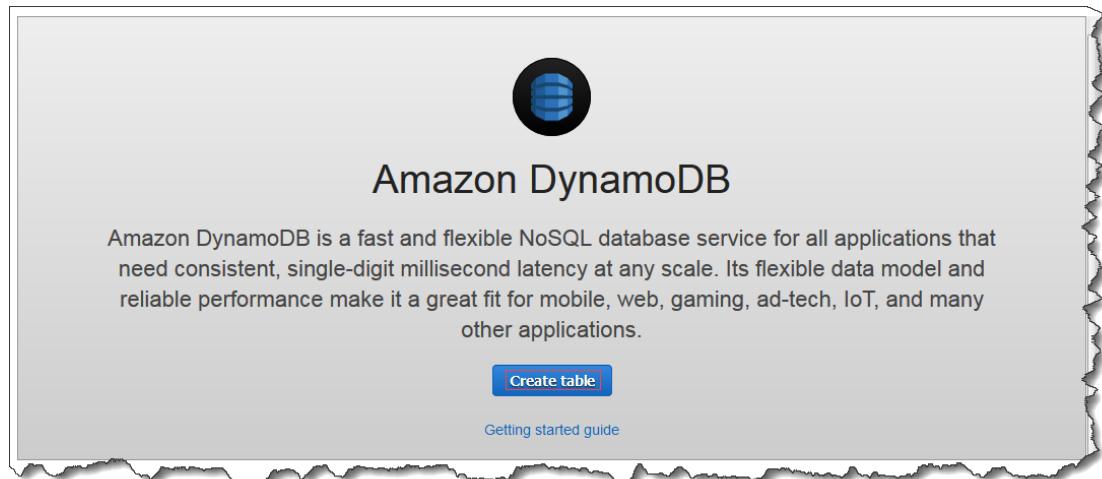
7. 在 Select an action 页面，选择 Insert a message into a DynamoDB table，然后选择 Configure action。



8. 在 Configure action 页面，选择 Create a new resource。



9. 在 Amazon DynamoDB 页面，选择 Create table。



10. 在 Create DynamoDB table 页面，在 Table name 字段键入名称。在 Partition key 中键入 **SerialNumber**。选中 Add sort key 复选框，然后在 Sort key 字段键入 **clickType**。为分区键和排序键选择 String。

Create DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* i

Primary key* Partition key

SerialNumber String i

Add sort key

ClickType String i

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

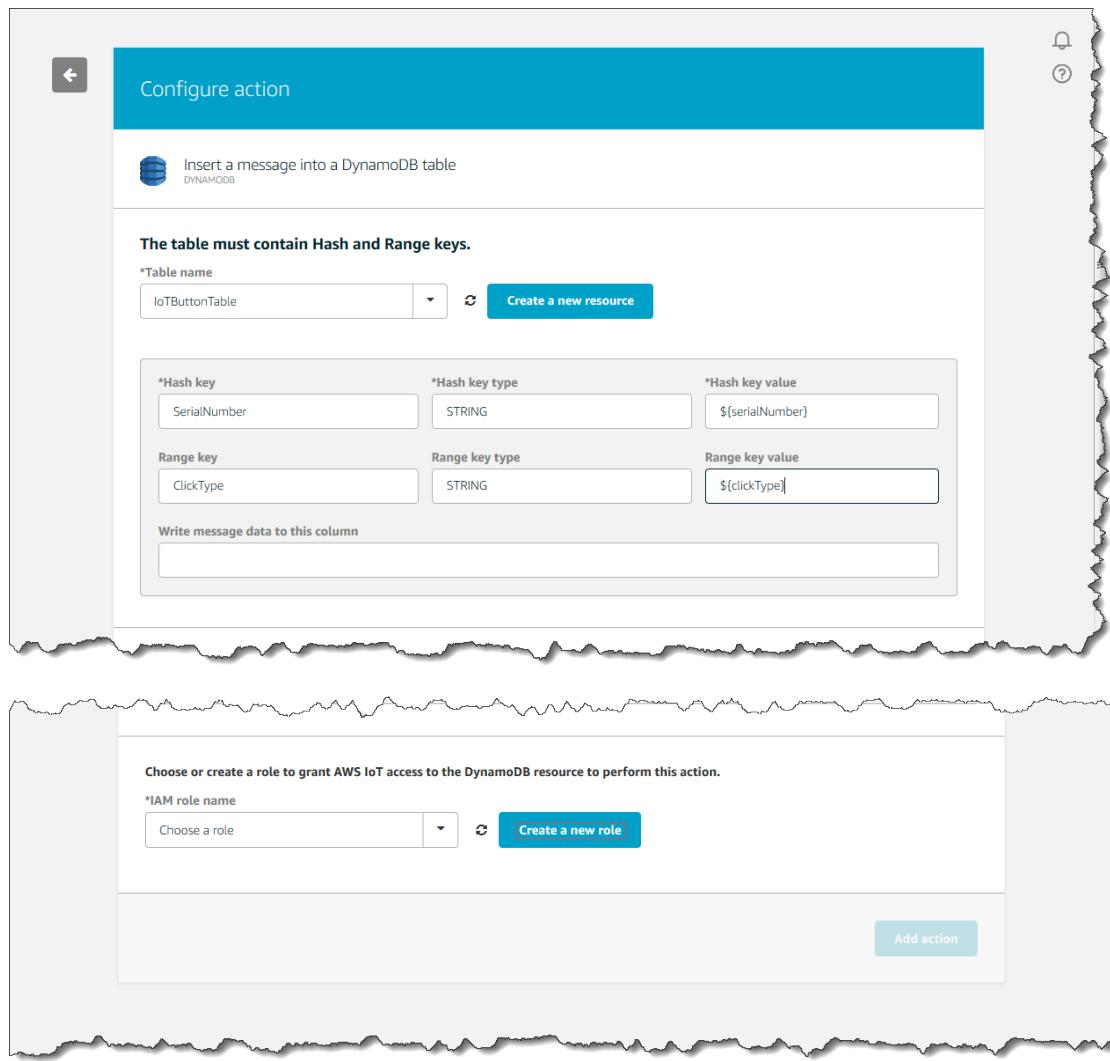
Use default settings

- No secondary indexes.
- Provisioned capacity set to 5 reads and 5 writes.
- Basic alarms with 80% upper threshold using SNS topic "dynamodb".

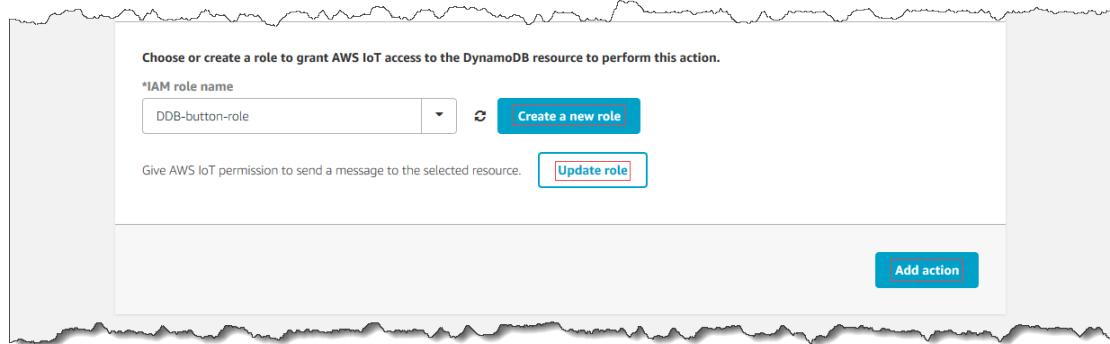
Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

Cancel Create

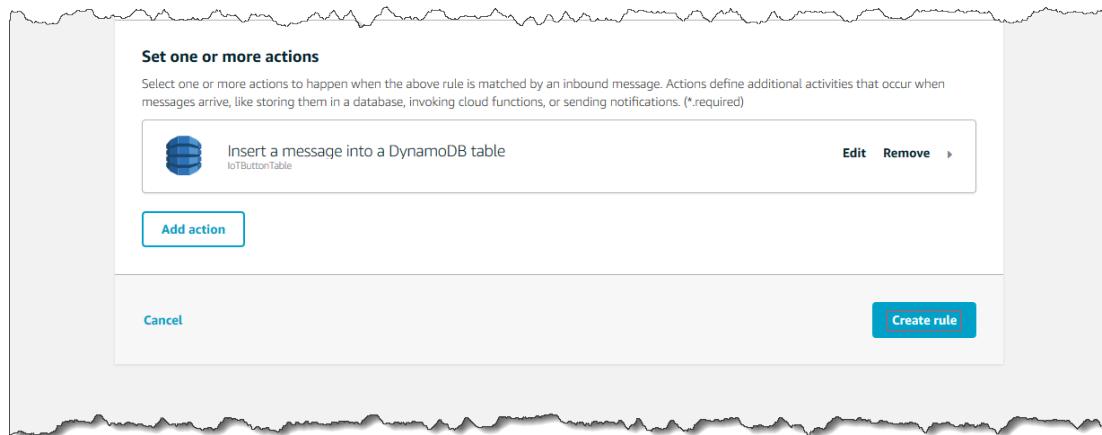
11. 选择 Create。创建 DynamoDB 表需要几秒钟时间。关闭打开 Amazon DynamoDB 控制台的浏览器标签。如果您未关闭该标签页，您的 DynamoDB 表将不会显示在 AWS IoT Configure action 页面的 Table name 下拉列表中。
12. 在 Configure action 页面，从 Table name 下拉列表中选择新表。In Hash Key Value 中，键入 `#{serialNumber}`。此操作将指示规则从 MQTT 消息中获取 serialNumber 属性的值，并将其写入 DynamoDB 表中的 SerialNumber 列。In Range Key Value 中，键入 `#{clickType}`。此操作会将 clickType 属性的值写入 ClickType 列。将 Write message data to this column 留空。默认情况下，整条消息都将写入表中名为“Payload”的列。选择 Create a new role。



13. 在 IAM role name 中键入唯一的名称，然后再次选择 Create a new role 按钮。选择您刚刚创建的角色，选择 Update role，然后选择 Add action。



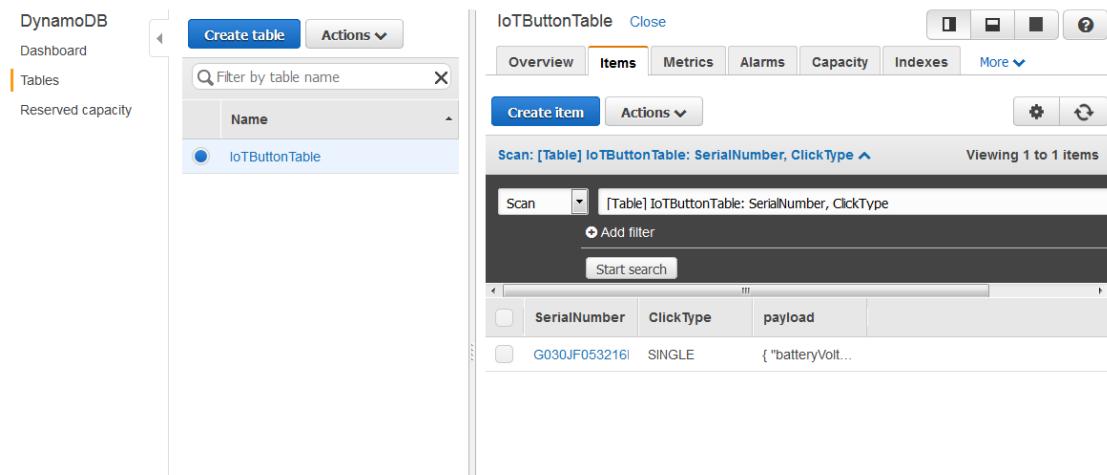
14. 选择 Create rule 来创建规则。



15. 确认消息将显示规则已创建。选择左箭头返回 Rules 页面。

The screenshot shows the AWS IoT Rules page. A new rule named 'MyDDBRule2' is listed under the 'ENABLED' section. The 'Overview' tab is selected. The 'Description' field contains the text 'Sends message data to DDB'. The 'Rule query statement' field contains the SQL query: 'SELECT * FROM `iotbutton/G030JF053216F1BS`'. The 'Actions' section shows the same 'Insert a message into a DynamoDB table' step as the previous screenshot. There are 'Edit' and 'Remove' buttons for the action, and a blue 'Add action' button below it. The 'Actions' tab is also visible.

16. 按一下完成配置的 AWS IoT 按钮，或者使用 MQTT 客户端向与您的规则主题筛选条件匹配的主题发布消息，以测试规则。最后，返回 DynamoDB 控制台并选择您创建的表来查看按钮按压或消息条目。



创建 Lambda 规则

您可以定义一个规则来调用 Lambda 函数，该函数可传入已触发规则的 MQTT 消息中的数据。如此一来，您可以处理传入的消息，然后调用其他 AWS 服务或第三方服务。

本教程假定您已经完成了 [AWS IoT 入门教程 \(p. 4\)](#)，且您已使用您的手机号码创建并订阅了一个 Amazon SNS 主题。您将创建一个 Lambda 函数，用于向您在 [AWS IoT 入门教程 \(p. 4\)](#) 中创建的 Amazon SNS 主题发布消息。此外，您还将创建一个用于调用 Lambda 函数的 Lambda 规则，该函数可以传入已触发规则的 MQTT 消息中的一些数据。

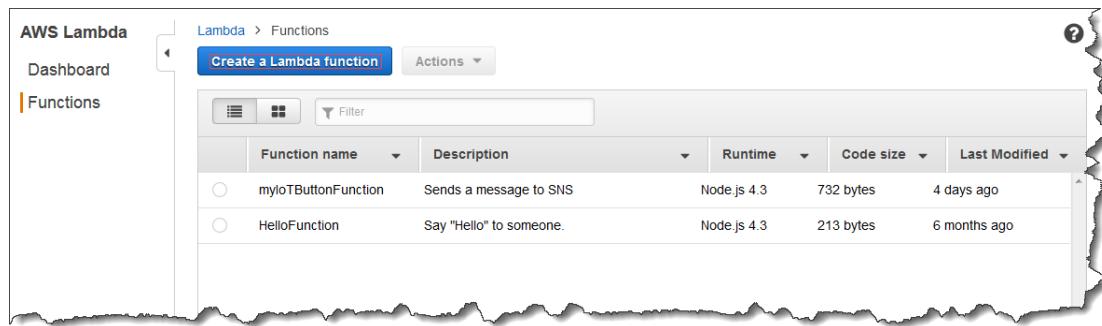
本教程还假定您正在使用 AWS IoT 按钮来触发 Lambda 规则。如果您没有 AWS IoT 按钮，[您可以在此处购买一个](#)，也可以使用 MQTT 客户端发送触发规则的 MQTT 消息。

创建 Lambda 函数

创建 Lambda 函数：

1. 在 [AWS Lambda 控制台](#) 中，选择 Get Started Now，如果您之前创建过 Lambda 函数，请选择 Create a Lambda function。





2. 在 Select blueprint 页面，在 Filter 字段键入 **hello-world**，然后选择 hello-world 蓝图。

Lambda > New function

Select blueprint

Configure triggers

Configure function

Review

Select blueprint

Welcome to AWS Lambda! You can get started on creating your first Lambda function by choosing one of the blueprints below.

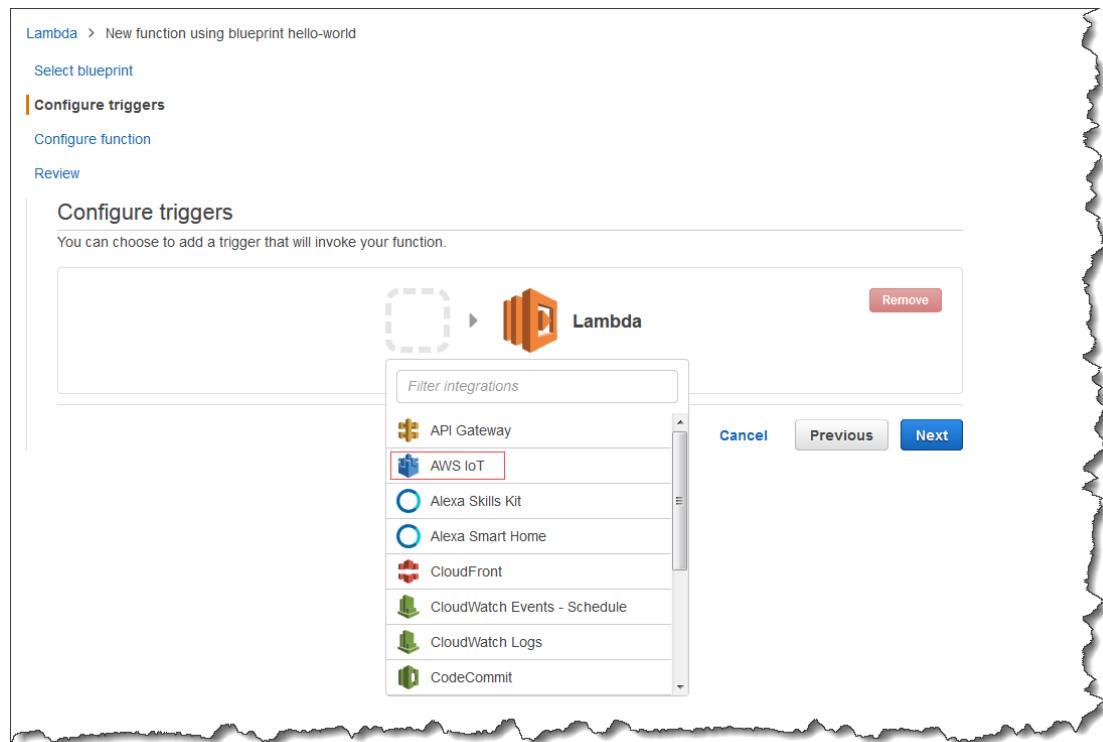
Select runtime ▾ Filter: hello-world

Viewing 1-3 of 3

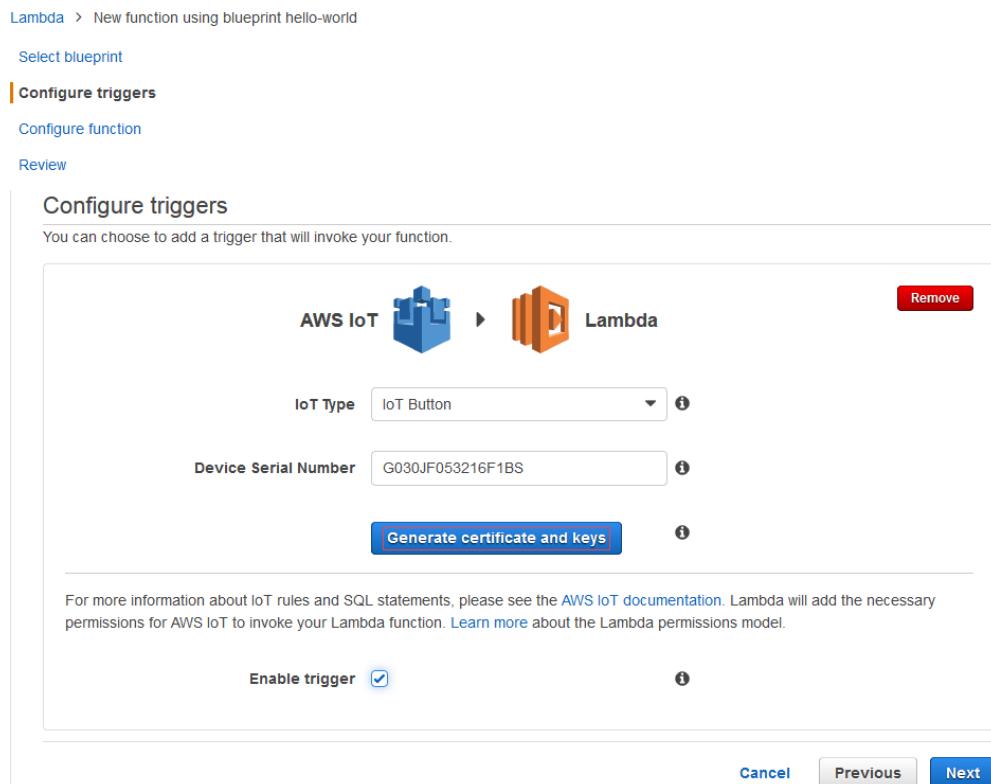
Blank Function	hello-world	hello-world-python
Configure your function from scratch. Define the trigger and deploy your code by stepping through our wizard. custom	A starter AWS Lambda function. nodejs	A starter AWS Lambda function. python2.7

Cancel

3. 在 Configure triggers 页面，选择 Lambda 图标左侧的框，然后从下拉菜单中选择 AWS IoT。



4. 在 Device Serial Number 字段，键入您的按钮的设备序列号 (DSN)。此 DSN 印在 AWS IoT 按钮的背面。如果您尚未给 AWS IoT 按钮生成证书和私有密钥，请选择 Generate certificate and keys。否则，请跳至步骤 6。



5. 选择用于下载证书 PEM 和私有密钥的链接。将这些文件保存在计算机中的安全位置。

We have created the necessary AWS IoT resources (thing, policy, certificate, private key). The remaining resources (rule and action) will be created after your function is created.

Download these resources by clicking the links below. (NOTE: If you are using Internet Explorer or Safari, right click the links to save the files.)

- a. Your certificate PEM
- b. Your private key

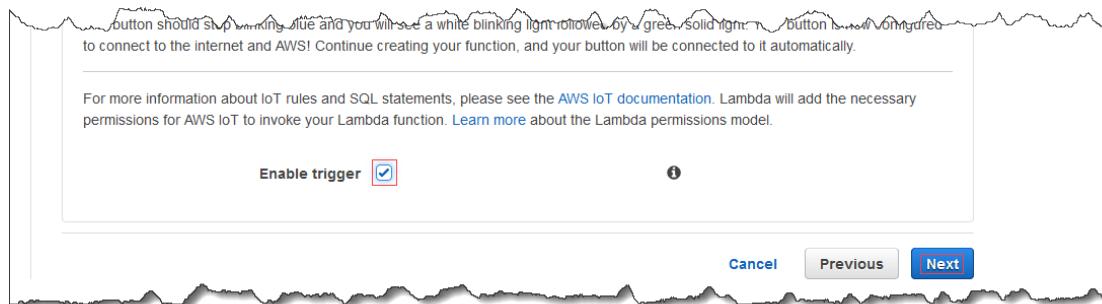
To configure the AWS IoT Button to use your Wi-Fi and these resources to connect to AWS securely, follow these steps:

1. Place the button into configuration mode by pressing the button down for 5 seconds until it flashes blue.
2. Connect your computer to the button's Wi-Fi network SSID "Button ConfigureMe - F09", using "3216F1BS" (last 8 digits of device serial number) as the WPA2-PSK password.
3. Click [here](#) (opens in new tab) and use the following information to fill out the form:
 - a. Enter your local network's Wi-Fi SSID and password.
 - b. Select the certificate and private key files that you just downloaded above.
 - c. Your endpoint subdomain is **a182jd32qs965e**.
 - d. Your endpoint region is **us-east-1**.
 - e. Check the box to agree to the terms and conditions.
 - f. Click "configure".
4. Re-connect to your original Wi-Fi network.

The button should stop blinking blue and you will see a white blinking light followed by a greed solid light. Your button is now configured to connect to the internet and AWS! Continue creating your function, and your button will be connected to it automatically.

按照在线说明操作来配置您的 AWS IoT 按钮。

6. 确保选中 Enable trigger 复选框，然后选择 Next。



7. 在 Configure function 页面上，键入 Lambda 函数的名称和描述。在 Runtime 中，选择 Node.js 6.10。

Lambda > New function using blueprint hello-world

Select blueprint Configure triggers **Configure function** Review

Configure function

A Lambda function consists of the custom code you want to execute. [Learn more about Lambda functions.](#)

Name* myiotButtonFunction

Description A starter AWS Lambda function.

Runtime* Node.js 6.10

8. 向下滚动至页面的 Lambda function code 部分。使用以下代码替换现有代码：

```
console.log('Loading function');
// Load the AWS SDK
var AWS = require("aws-sdk");

// Set up the code to call when the Lambda function is invoked
exports.handler = (event, context, callback) => {
    // Load the message passed into the Lambda function into a JSON object
    var eventText = JSON.stringify(event, null, 2);

    // Log a message to the console, you can view this text in the Monitoring tab
    // in the Lambda console or in the CloudWatch Logs console
    console.log("Received event:", eventText);

    // Create a string extracting the click type and serial number from the message
    // sent by the AWS IoT button
    var messageText = "Received " + event.clickType + " message from button ID: "
    + event.serialNumber;

    // Write the string to the console
    console.log("Message to send: " + messageText);

    // Create an SNS object
    var sns = new AWS.SNS();

    // Populate the parameters for the publish operation
    // - Message : the text of the message to send
    // - TopicArn : the ARN of the Amazon SNS topic to which you want to publish
    var params = {
        Message: messageText,
        TopicArn: "arn:aws:sns:us-east-1:123456789012:MyIoTButtonSNSTopic"
    };
    sns.publish(params, context.done);
};
```

Note

使用您先前创建的 Amazon SNS 主题的 ARN 替换 TopicArn 的值。

9. 向下滚动至页面的 Lambda function handler and role 部分。对于 Role，请选择 Create a custom role。此时 IAM 控制台将打开，您可以创建一个 Lambda 在执行 Lambda 函数时可代入的 IAM 角色。

编辑角色策略以授予该角色向您的 Amazon SNS 主题发布消息的权限：

- a. 选择 View Policy Document。

AWS Lambda requires access to your resources

AWS Lambda uses an IAM role that grants your custom code permissions to access AWS resources it needs.

▼ Hide Details

Role Summary

Role Lambda execution role permissions

Description

IAM Role lambda_basic_execution

Policy Name Create a new Role Policy

▶ [View Policy Document](#)

Don't Allow [Allow](#)

- b. 选择 Edit 以编辑角色的策略。

AWS Lambda requires access to your resources

AWS Lambda uses an IAM role that grants your custom code permissions to access AWS resources it needs.

▼ Hide Details

Role Summary

Role Lambda execution role permissions

Description

IAM Role lambda_basic_execution

Policy Name Create a new Role Policy

▼ Hide Policy Document

[Edit](#)

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "logs:CreateLogGroup",  
        "logs:CreateLogStream",  
        "logs:PutLogEvents"  
      ]  
    }  
  ]  
}
```

Don't Allow [Allow](#)

- c. 使用以下文档替换当前策略文档：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:CreateLogGroup",  
                "logs:CreateLogStream",  
                "logs:PutLogEvents"  
            ],  
            "Resource": "arn:aws:logs:*:*:  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:Publish"  
            ],  
            "Resource": "arn:aws:sns:us-east-1:123456789012:MyIoTButtonSNSTopic"  
        }  
    ]  
}
```

该策略文档添加了向您的 Amazon SNS 主题发布消息的权限。

Note

使用您先前创建 Amazon SNS 主题的 ARN 替换第二个 Resource 的值。

10. 选择 Allow。

AWS Lambda requires access to your resources

AWS Lambda uses an IAM role that grants your custom code permissions to access AWS resources it needs.

▼ Hide Details

Role Summary

Role Lambda execution role permissions

Description

IAM Role lambda_basic_execution

Policy Name Create a new Role Policy

▼ Hide Policy Document

Edit

```
"Version": "2012-10-17",  
"Statement": [  
    {  
        "Effect": "Allow",  
        "Action": [  
            "logs:CreateLogGroup",  
            "logs:CreateLogStream",  
            "logs:PutLogEvents"  
        ],  
        "Resource": "arn:aws:logs:*:*:  
    },  
    {  
        "Effect": "Allow",  
        "Action": [  
            "sns:Publish"  
        ],  
        "Resource": "arn:aws:sns:us-east-1:123456789012:MyIoTButtonSNSTopic"  
    }  
]
```

Don't Allow Allow

11. 将 Advanced settings 页面中的各项设置保留为默认设置，然后选择 Next。

Advanced settings

These settings allow you to control the code execution performance and costs for your Lambda function. Changing your resource settings (by selecting memory) or changing the timeout may impact your function cost. [Learn more](#) about how Lambda pricing works.

Memory (MB)* 128

Timeout* 0 min 3 sec

AWS Lambda will automatically retry failed executions for asynchronous invocations. You can additionally optionally configure Lambda to forward payloads that were not processed to a dead-letter queue (DLQ), such as an SQS queue or an SNS topic. Learn more about Lambda's [retry policy](#) and [DLQs](#). Please ensure your role has appropriate permissions to access the DLQ resource.

DLQ Resource Select resource

All AWS Lambda functions run securely inside a default system-managed VPC. However, you can optionally configure Lambda to access resources, such as databases, within your custom VPC. [Learn more](#) about accessing VPCs within Lambda. Please ensure your role has appropriate permissions to configure VPC.

VPC No VPC

Environment variables are encrypted at rest using a default Lambda service key. You can change the key below to one of your account's keys or paste in a full KMS key ARN.

KMS key (default) aws/lambda

* These fields are required.

[Cancel](#)

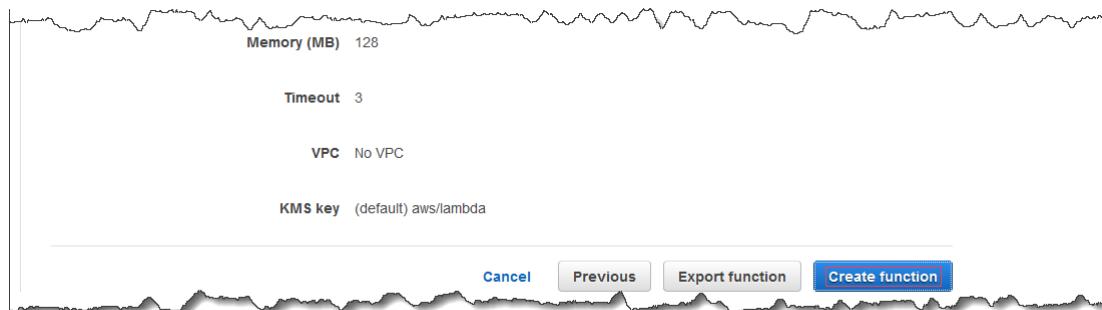
[Previous](#)

[Next](#)

12. 在 Review 页面上，选择 Create function。

The screenshot shows the 'Review' step of creating a Lambda function. At the top, it says 'Lambda > New function using blueprint hello-world'. Below that, there are tabs: 'Select blueprint', 'Configure triggers', 'Configure function', and 'Review' (which is highlighted). The 'Review' section contains the following details:

- Triggers**: An AWS IoT trigger is listed with the name 'AWS IoT' and DSN 'G030JF053216F1BS'. It is currently 'Enabled'. There is an 'Edit' button next to it.
- Lambda function**:
 - Name**: MyIoTButtonFunction
 - Description**: Sends a message to SNS
 - Runtime**: Node.js 4.3



测试您的 Lambda 函数

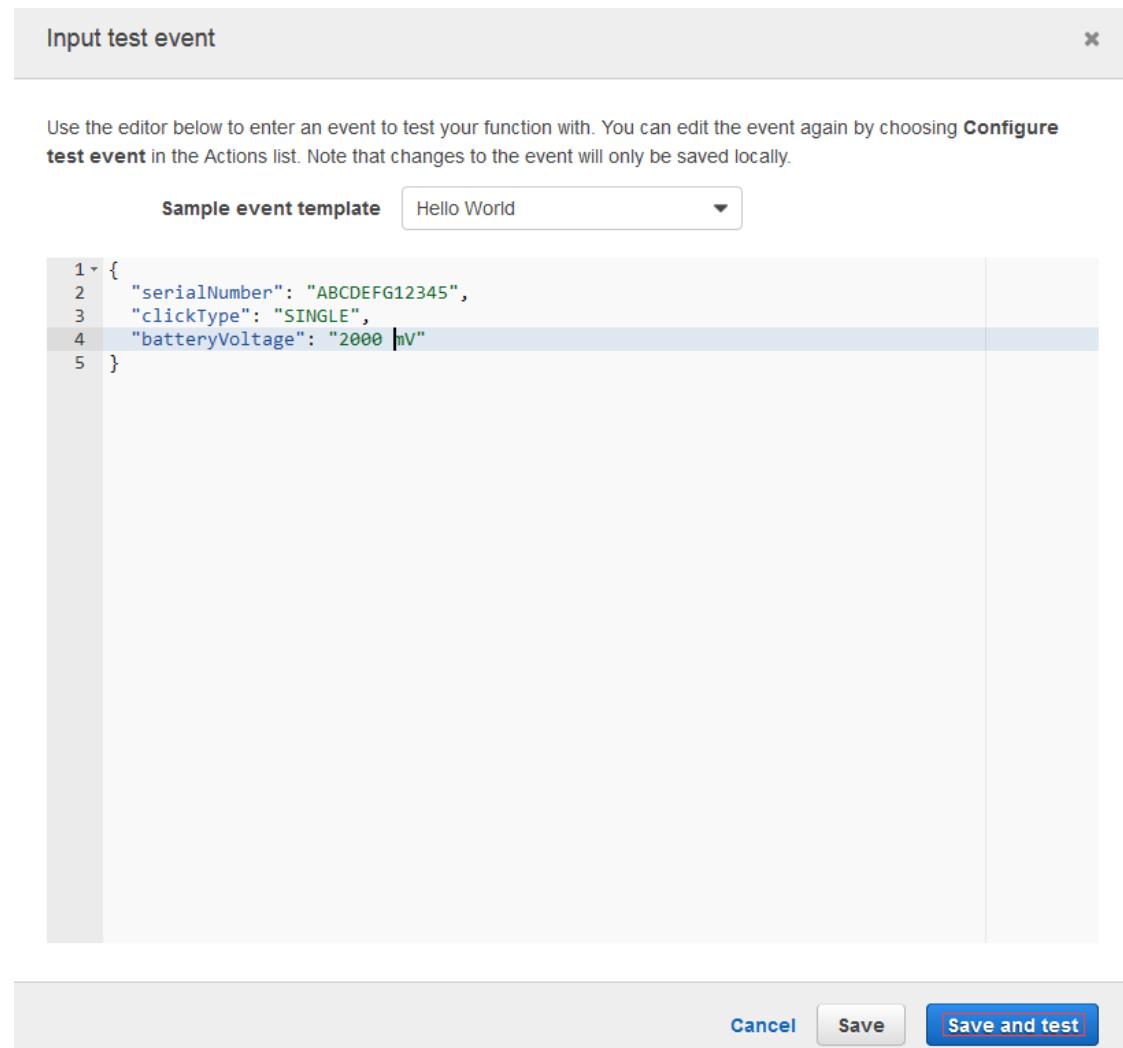
测试 Lambda 函数：

1. 在 Actions 菜单上，选择 Configure test event。

The screenshot shows the AWS Lambda Functions console. On the left, there's a sidebar with 'AWS Lambda' and 'Functions'. In the main area, there's a list of functions. One function is selected, showing its details: ARN, Lambda rule, and SQL statement. A context menu is open over the 'Actions' button, with 'Configure test event' highlighted in red. Other options in the menu include 'Publish new version', 'Create alias', 'Delete function', and 'Export function'. A message box is also visible, stating: 'Congratulations! Your Lambda function "MyIoTButtonFunction" has been successfully created and configured with IoT. Now click on the "Test" button to input a test event and test your function.'

2. 复制以下 JSON 并将其粘贴到 Input test event 页面上，然后选择 Save and test。

```
{  
    "serialNumber": "ABCDEFG12345",  
    "clickType": "SINGLE",  
    "batteryVoltage": "2000 mV"  
}
```



3. 在 AWS Lambda 控制台中，滚动至页面底部。Log output 部分将显示 Lambda 函数已写入控制台的输出。

Log output

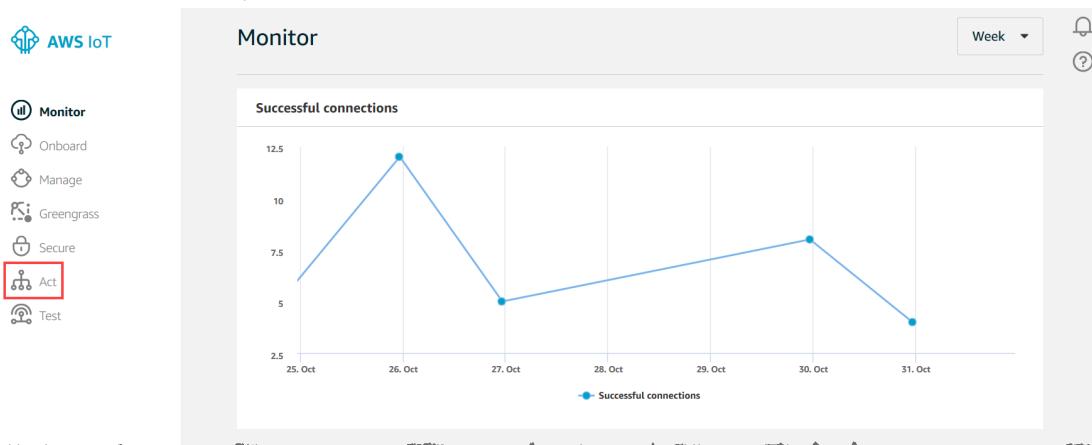
The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```
START RequestId: c4b5b4d1-1631-11e6-b78f-0d4d596724ad Version: $LATEST
2016-05-09T22:02:49.501Z      c4b5b4d1-1631-11e6-b78f-0d4d596724ad      Received event: {
    "serialNumber": "ABCDEFG12345",
    "clickType": "SINGLE",
    "batteryVoltage": "2000 mV"
}
2016-05-09T22:02:49.501Z      c4b5b4d1-1631-11e6-b78f-0d4d596724ad      Message to send: Received
END RequestId: c4b5b4d1-1631-11e6-b78f-0d4d596724ad
REPORT RequestId: c4b5b4d1-1631-11e6-b78f-0d4d596724ad Duration: 1215.14 ms      Billed Duration: 13
```

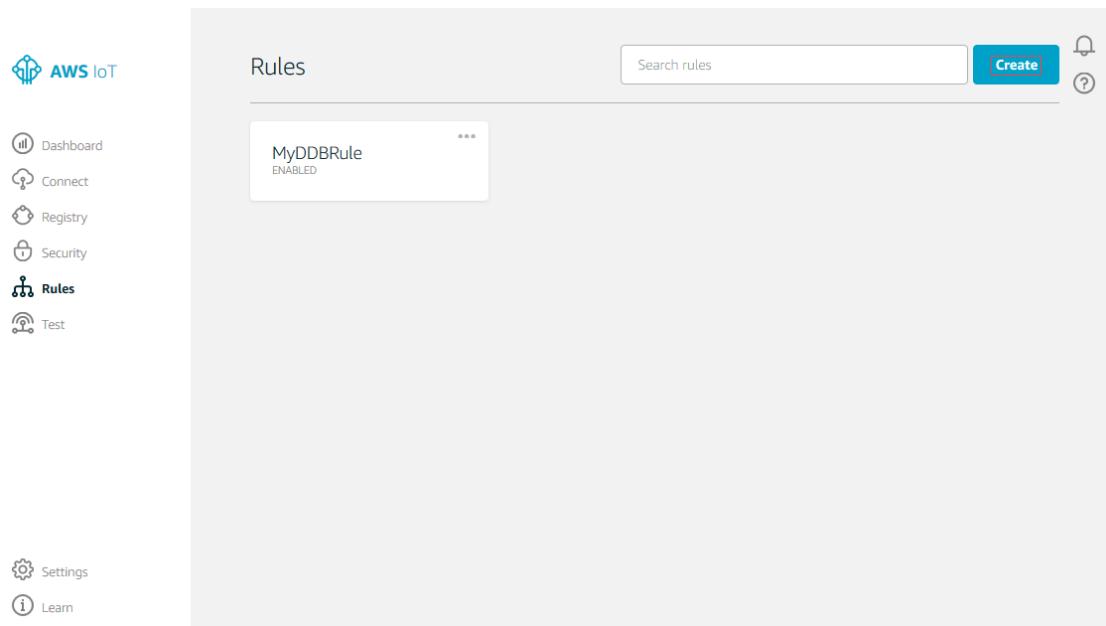
创建 Lambda 规则

您已经创建了 Lambda 函数，接下来您可以创建用于调用 Lambda 函数的规则。

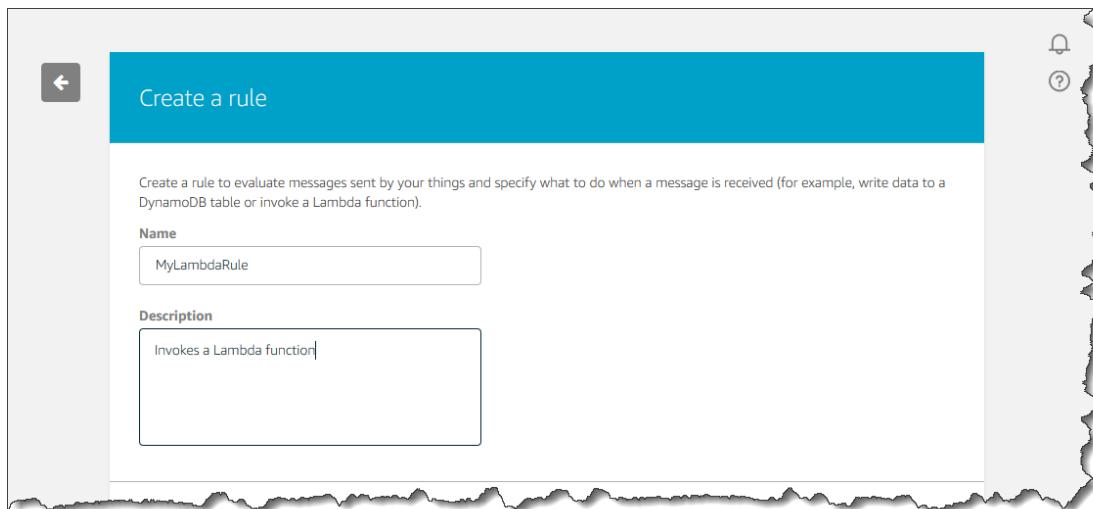
- 在 [AWS IoT 控制台](#) 中，在左侧导航窗格中选择 Rules。



- 在 Rules 页面，选择 Create。



3. 键入规则的名称和描述。



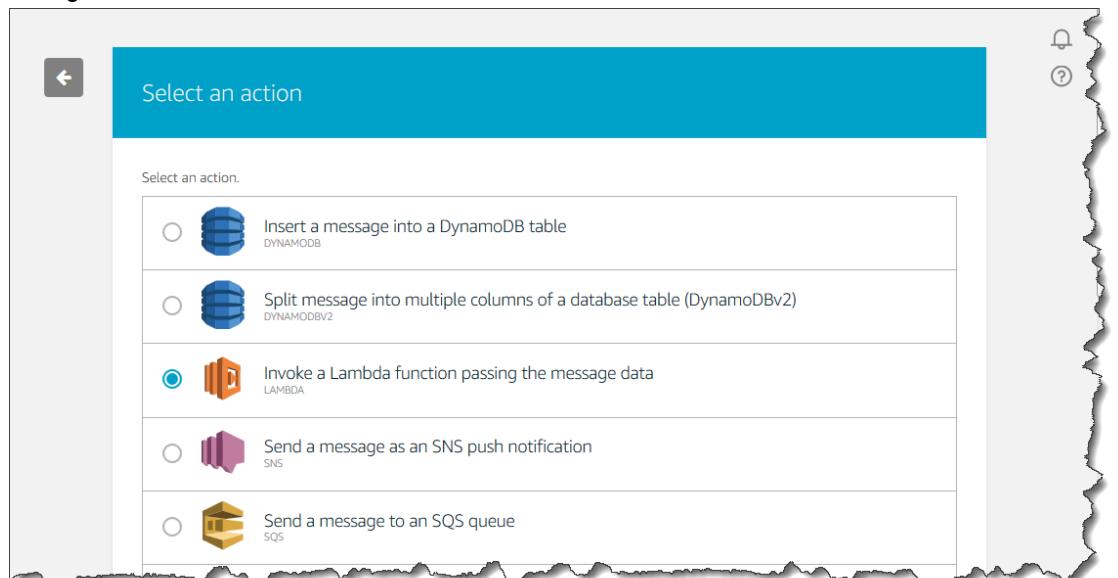
4. 输入以下规则设置：

The screenshot shows the 'Message source' configuration page. It includes fields for 'Using SQL version' (set to 2016-03-23), 'Rule query statement' (containing 'SELECT * FROM 'iotbutton/+''), 'Attribute' (containing '*'), 'Topic filter' (containing 'iotbutton/+'), and 'Condition' (containing 'e.g. temperature > 75'). Below these fields is a section titled 'Set one or more actions'.

5. 在 Set one or more actions 中，选择 Add action。

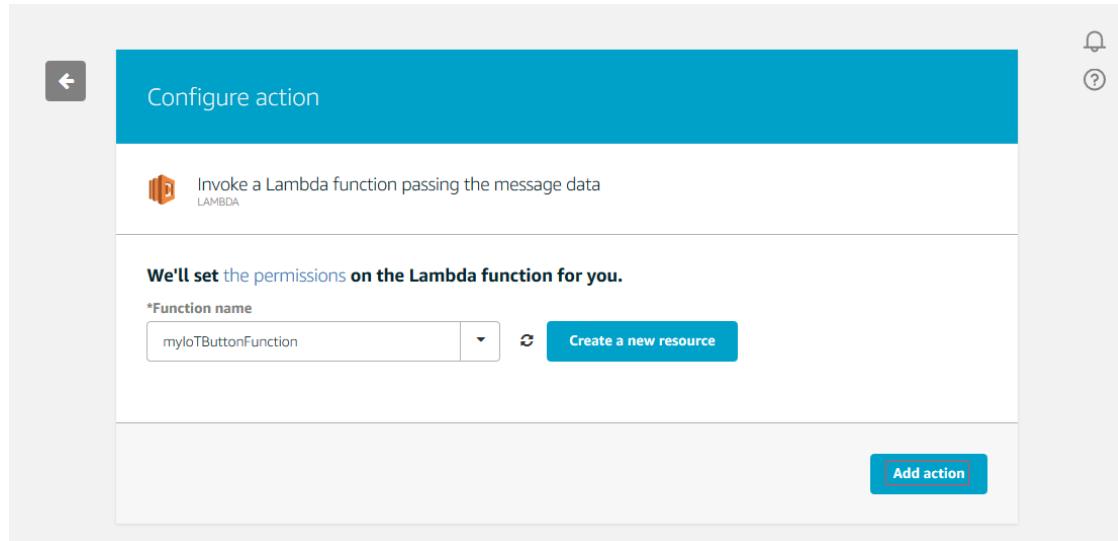


6. 在 Select an action 页面，选择 Invoke a Lambda function passing the message data，然后选择 Configure action。

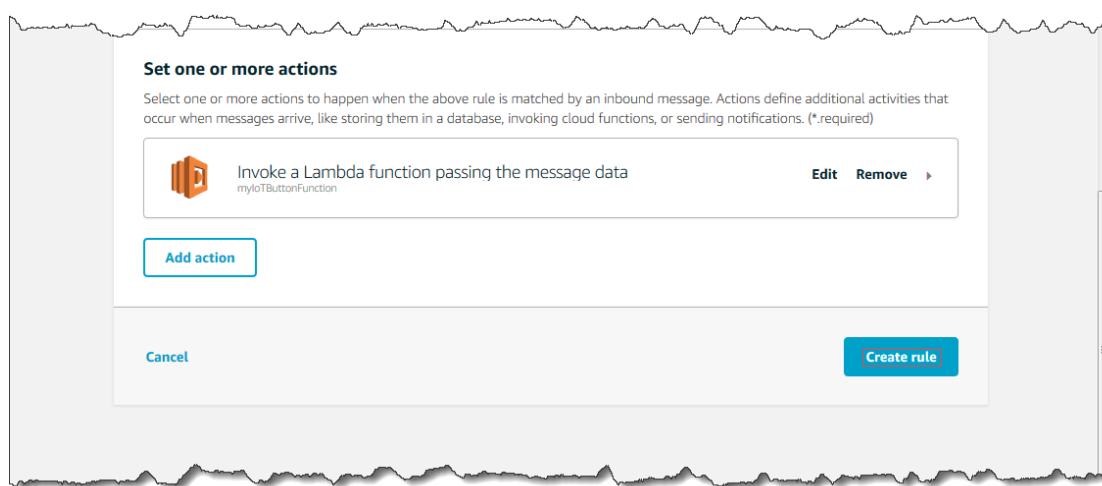




7. 从 Function name 下拉列表中选择 Lambda 函数名称，然后选择 Add action。



8. 选择 Create rule 来创建您的 Lambda 函数。



测试您的 Lambda 规则

本教程假定您已经完成了 [AWS IoT 入门教程 \(p. 4\)](#)，其中包括：

- 配置 AWS IoT 按钮。

- 使用手机号码创建和订阅 Amazon SNS 主题。

您的按钮已完成配置并连接到 Wi-Fi，且您已经配置了 Amazon SNS 主题，现在您可以按一下按钮来测试您的 Lambda 规则。您的手机应该会收到一条 SMS 文本消息，其中包含：

- 您的按钮的序列号。
- 按钮按压类型 (按一下或按两下)。
- 电池电压。

该消息的内容应与以下内容类似：

```
IOT BUTTON> {  
    "serialNumber" : "ABCDEFG12345",  
    "clickType" : "SINGLE",  
    "batteryVoltage" : "2000 mV"  
}
```

如果您没有按钮，[您可以在此处购买一个](#)，也可以使用 AWS IoT MQTT 客户端作为替代。

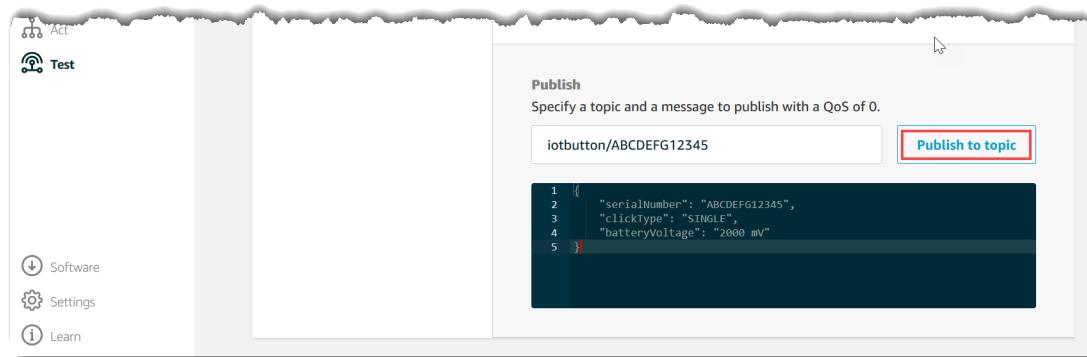
- 在 [AWS IoT 控制台](#) 中，选择 Test。



- 在 MQTT client 页面上，在 Publish 部分的 Specify a topic 中键入 `iotbutton/ABCDEFG12345`。

在 Payload 中，键入以下 JSON，然后选择 Publish to topic。

```
{  
    "serialNumber" : "ABCDEFG12345",  
    "clickType" : "SINGLE",  
    "batteryVoltage" : "2000 mV"  
}
```



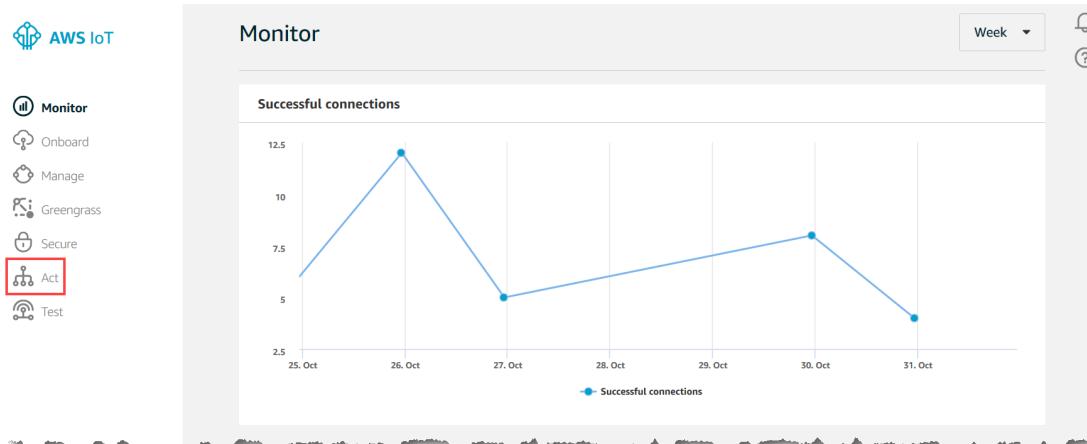
3. 您的手机会收到一条消息。

创建 Amazon SNS 规则

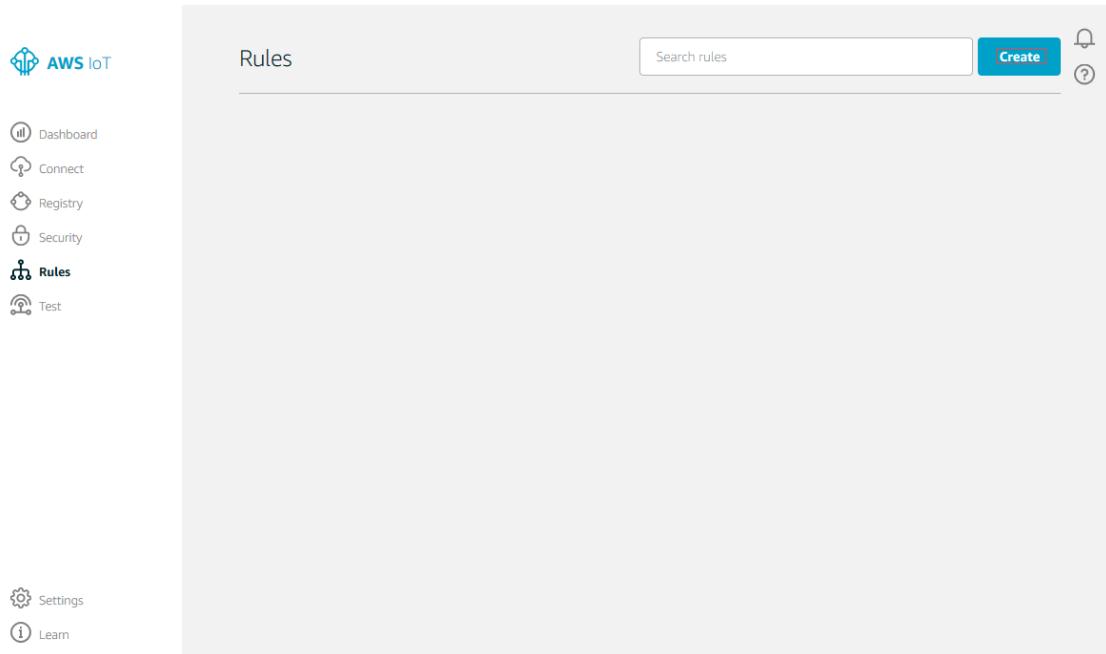
您可以定义将消息数据发送到 Amazon SNS 主题的规则。在本教程中，您将创建一条规则来将触发该规则的 AWS IoT 事物的名称发送到 Amazon SNS 主题的所有订阅者。

使用 SNS 操作创建规则：

1. 在 [AWS IoT 控制台](#) 中，在左侧导航窗格中选择 Rules。



2. 在 Rules 页面，选择 Create。



3. 为您的规则键入名称。

The screenshot shows the 'Create a rule' wizard. The title bar says 'Create a rule'. Below it is a descriptive text: 'Create a rule to evaluate messages sent by your things and specify what to do when a message is received (for example, write data to a DynamoDB table or invoke a Lambda function)'. The 'Name' field is filled with 'MySNSRule' and has a red border around it. The 'Description' field is empty.

4. 在 Message source 下，为 Attribute 键入 *，topic(3)。对于 Topic filter，请键入 \$aws/things/+ / shadow/update/accepted。主题筛选条件指定在向其发布消息时触发规则的操作的主题。主题筛选条件中使用的 + 是匹配任何事物名的通配符。此属性将事物名称附加到消息内容中。

Message source
Indicate the source of the messages you want to process with this rule.

Using SQL version ②
2016-03-23 ▾

Rule query statement

```
SELECT *, topic(3) as thing FROM '$aws/things/+shadow/update/accepted'
```

Attribute

```
*; topic(3) as thing
```

Topic filter

```
$aws/things/+shadow/update/accepted
```

Condition

```
e.g. temperature > 75
```

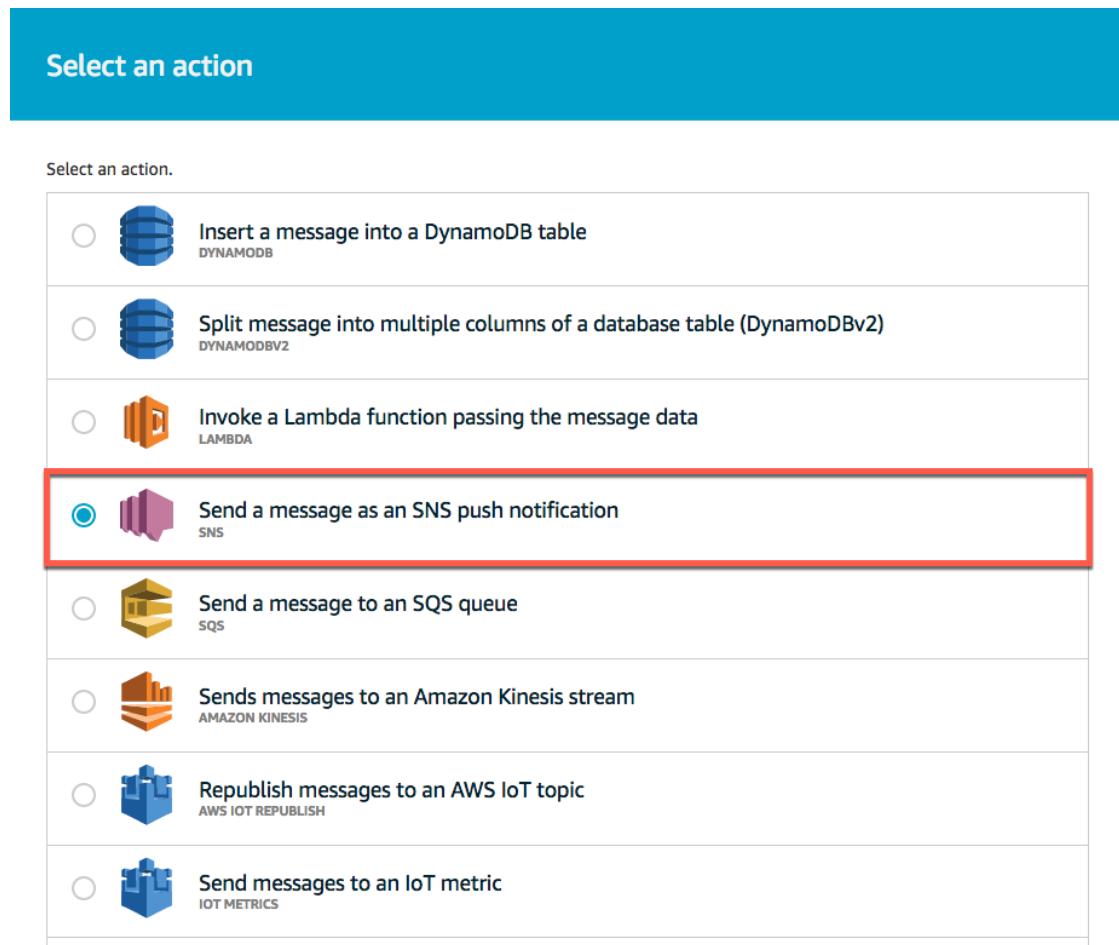
-
5. 在 Set one or more actions 选项中，选择 Add action。

Set one or more actions

Select one or more actions to happen when the above rule is matched by an inbound message. Actions define additional activities that occur when messages arrive, like storing them in a database, invoking cloud functions, or sending notifications. (*.required)

Add action

6. 在 Select an action 下，选择 Send a message as an SNS push notification，然后选择 Configure action。(此按钮未显示在屏幕截图中)。



7. 选择 Create new topic。

The screenshot shows the 'Topics' section of the AWS SNS console. The 'Create new topic' button is highlighted with a red box. The table lists existing topics:

Name	ARN
CFN-notifications	arn:aws:sns:us-west-2:803981987763:CFN-notifications
CFNEmailNotification	arn:aws:sns:us-west-2:803981987763:CFNEmailNotification
CloudFormationNotifications	arn:aws:sns:us-west-2:803981987763:CloudFormationNotifications
DDB-EXPORT-ProductCat...	arn:aws:sns:us-west-2:803981987763:DDB-EXPORT-ProductCatalog-1397236041808fBEFNVC1L
DDB-Import-Thread-1397...	arn:aws:sns:us-west-2:803981987763:DDB-Import-Thread-1397261293000fKCbKloud
ElasticBeanstalkNotificati...	arn:aws:sns:us-west-2:803981987763:ElasticBeanstalkNotifications-sampleElasticBeanstalkApplication-python123-AWSEBLoadBalancer
ElasticBeanstalkNotificati...	arn:aws:sns:us-west-2:803981987763:ElasticBeanstalkNotifications-sampleElasticBeanstalkApplication-testestest-AWSEBAutoScalingGroup
ElasticBeanstalkNotificati...	arn:aws:sns:us-west-2:803981987763:ElasticBeanstalkNotifications-sampleElasticBeanstalkApplication-testestest-AWSEBLoadBalancer
ElasticBeanstalkNotificati...	arn:aws:sns:us-west-2:803981987763:ElasticBeanstalkNotifications-sdtest1-sdtest1-AWSEBAutoScalingGroup
ElasticBeanstalkNotificati...	arn:aws:sns:us-west-2:803981987763:ElasticBeanstalkNotifications-sdtest1-sdtest1-AWSEBLoadBalancer
LambdaEmailAlert	arn:aws:sns:us-west-2:803981987763:LambdaEmailAlert
MyTopic	arn:aws:sns:us-west-2:803981987763:MyTopic

Total Items: 33
Selected Items: 0

8. 在您的浏览器中打开一个新的选项卡。键入 SNS 主题的名称和描述，然后选择 Create topic。

Create new topic

A topic name will be used to create a permanent unique identifier called an Amazon Resource Name (ARN).

Topic name	MySNSTopic	i
Display name	IoT SNS	i

Cancel **Create topic**

9. 切换到从中打开 AWS IoT 控制台的浏览器选项卡。对于 SNS target，选择您刚刚创建的 SNS 主题。对于 Message format，请选择 JSON。

Configure action

SNS target: MySNSTopic

Message format: JSON

Choose or create a role to grant AWS IoT access to the SNS resource to perform this action.

IAM role name: Choose a role

Add action

The screenshot shows the AWS IoT 'Configure action' interface. At the top, it says 'Create new topic'. Below that, a note states 'A topic name will be used to create a permanent unique identifier called an Amazon Resource Name (ARN)'. Two input fields are shown: 'Topic name' containing 'MySNSTopic' and 'Display name' containing 'IoT SNS'. Both fields have red boxes around them. At the bottom right of this section is a blue 'Create topic' button. The main part of the screen is titled 'Configure action' with a teal header. It shows a 'Send a message as an SNS push notification' icon. Underneath, there are two dropdown menus: 'SNS target' set to 'MySNSTopic' (with a red box around it) and 'Message format' set to 'JSON' (with a red box around it). To the right of these are 'Create a new resource' and 'Create a new role' buttons. Below these is a note: 'Choose or create a role to grant AWS IoT access to the SNS resource to perform this action.' A dropdown menu for 'IAM role name' is shown with 'Choose a role' selected, and 'Create a new role' and 'Update role' buttons to its right. At the bottom right of the main area is a light blue 'Add action' button.

10. 对于 IAM role name，请选择 Create a new role。

Configure action

 Send a message as an SNS push notification
SNS

*SNS target
MySNSTopic Create a new resource

Message format
JSON

Choose or create a role to grant AWS IoT access to the SNS resource to perform this action.

*IAM role name
Choose a role Update role Create a new role

Add action

11. 键入角色的名称，然后选择 Create a new role。

Configure action

 Send a message as an SNS push notification

*SNS target: MySNSTopic [Create a new resource](#)

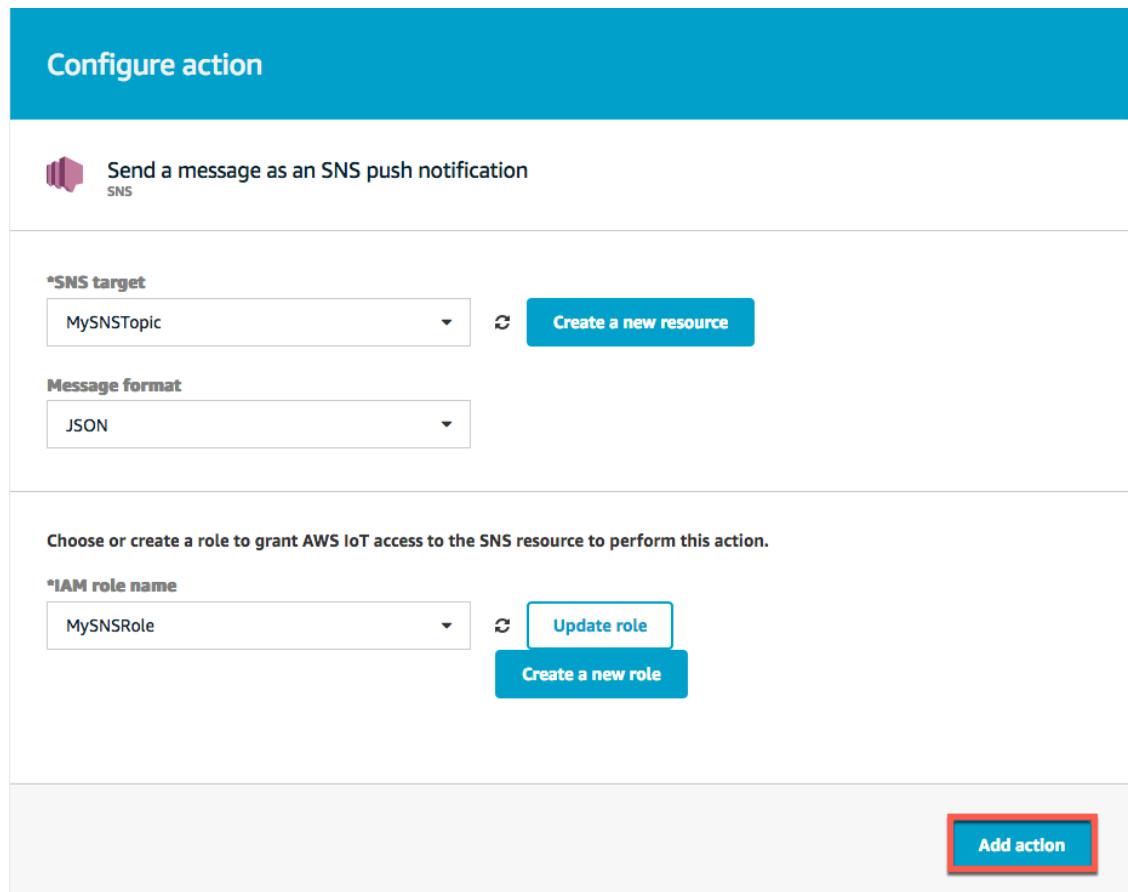
Message format: JSON

Choose or create a role to grant AWS IoT access to the SNS resource to perform this action.

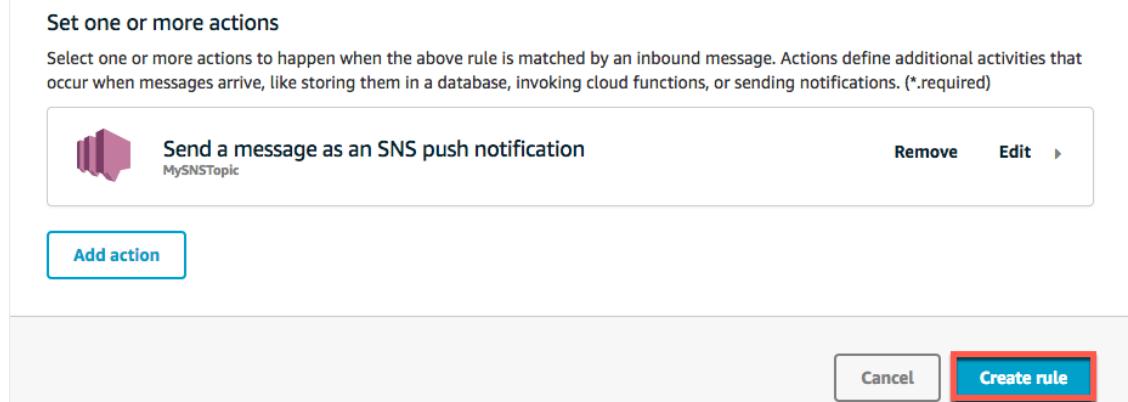
*IAM role name: MySNSRole [Create a new role](#) [Cancel](#)

Add action

12. 选择您刚刚创建的角色，然后选择 Add action。



13. 选择 Create rule。



您现在已创建规则。要测试规则，请将订阅添加到您创建的 SNS 主题，然后更新任何 AWS IoT 事物的事物影子。您可以使用 AWS IoT 控制台查找事物、打开其详细信息页面和更改事物的影子。Thing Shadow 服务在收到变更通知后将在 \$aws/things/*MySNSThing*/shadow/update/accepted 上发布消息。将触发您的规则，并且您的 SNS 主题的所有订阅者都将收到一条包含您的事物的名称的消息。

AWS IoT 软件开发工具包教程

AWS IoT 设备软件开发工具包帮助您快速轻松地将设备连接至 AWS IoT。AWS IoT 设备软件开发工具包包括开源库、开发人员指南(含示例)和移植指南，便于您在自己选择的硬件平台上构建富有创新精神的 IoT 产品或解决方案。

本指南提供分步指导，以帮助您将 Raspberry Pi 连接到 AWS IoT 平台并对其进行设置，以与 AWS IoT 嵌入式 C 软件开发工具包和适用于 Javascript 的设备软件开发工具包配合使用。按照本指南中的步骤执行操作后，您便可以连接到 AWS IoT 平台并运行这些 AWS IoT SDK 中包含的示例应用程序。

内容

- [连接您的 Raspberry Pi \(p. 77\)](#)
- [使用 AWS IoT 嵌入式 C 软件开发工具包 \(p. 86\)](#)
- [使用适用于 JavaScript 的 AWS IoT 设备软件开发工具包 \(p. 89\)](#)

连接您的 Raspberry Pi

遵循以下步骤将 Raspberry Pi 连接到 AWS IoT 平台。

先决条件

- 完全设置的 Raspberry Pi 面板及 Internet 访问

有关设置 Raspberry Pi 的更多信息，请参阅 [Raspberry Pi 快速入门指南](#)。
- Chrome 或 Firefox (Iceweasel) 浏览器

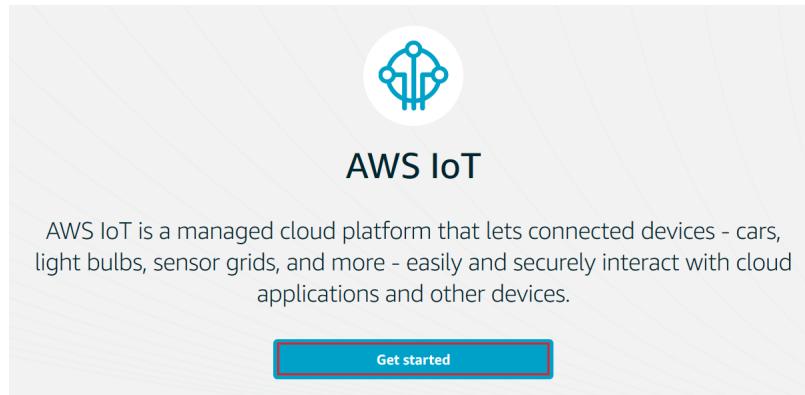
有关安装 Iceweasel 的更多信息，请参阅[有关嵌入式 Linux wiki 的说明](#)。

在本指南中，使用了下列硬件和软件：

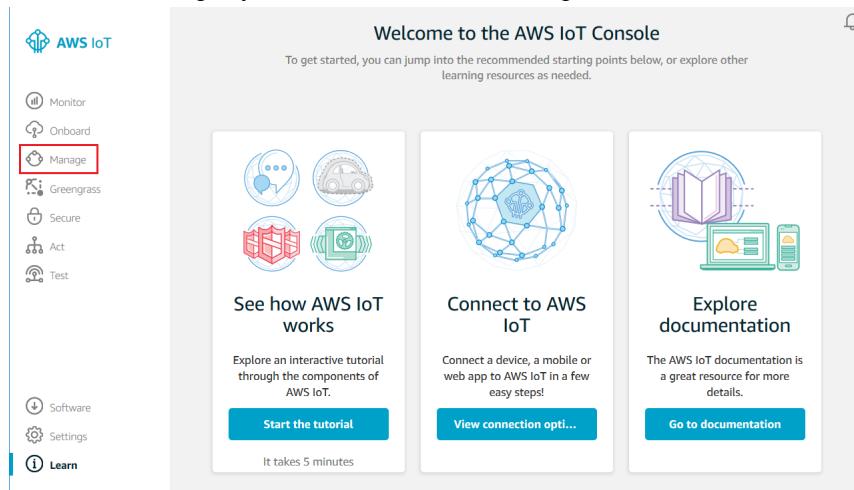
- [Raspberry Pi 2 Model B](#)
- [Raspbian Wheezy](#)
- [Raspbian Jessie](#)
- [Iceweasel 浏览器](#)

登录 AWS IoT 控制台

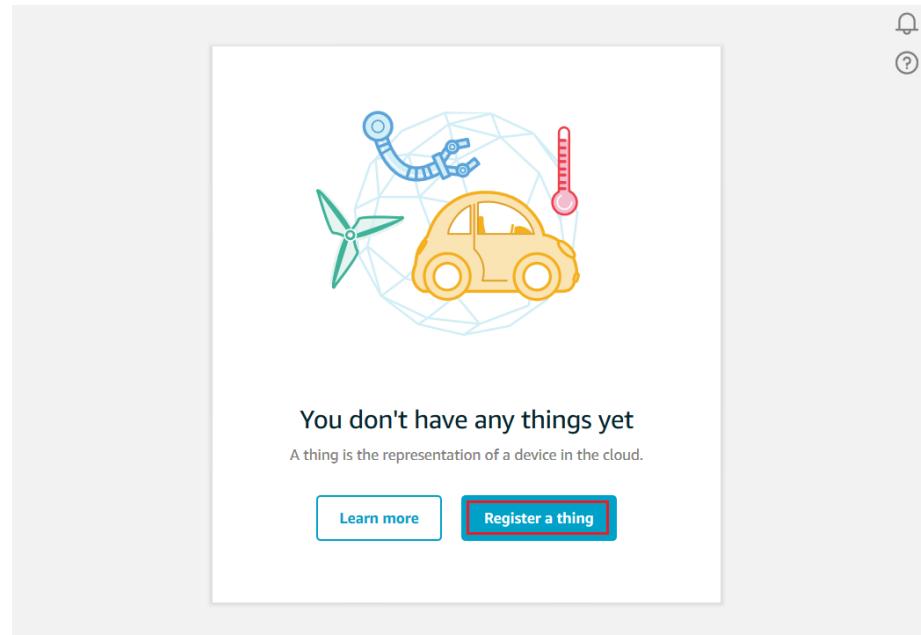
1. 打开您的 Raspberry Pi 并确认您已连接到 Internet。
2. 登录 AWS 管理控制台，并在 <https://aws.amazon.com/iot> 上打开 AWS IoT 控制台。在 Welcome 页面上，选择 Get started。



3. 如果这是您第一次使用 AWS IoT 控制台，您会看到 Welcome to the AWS IoT Console 页面。在左侧导航窗格，选择 Registry 来展开选项，然后选择 Things。



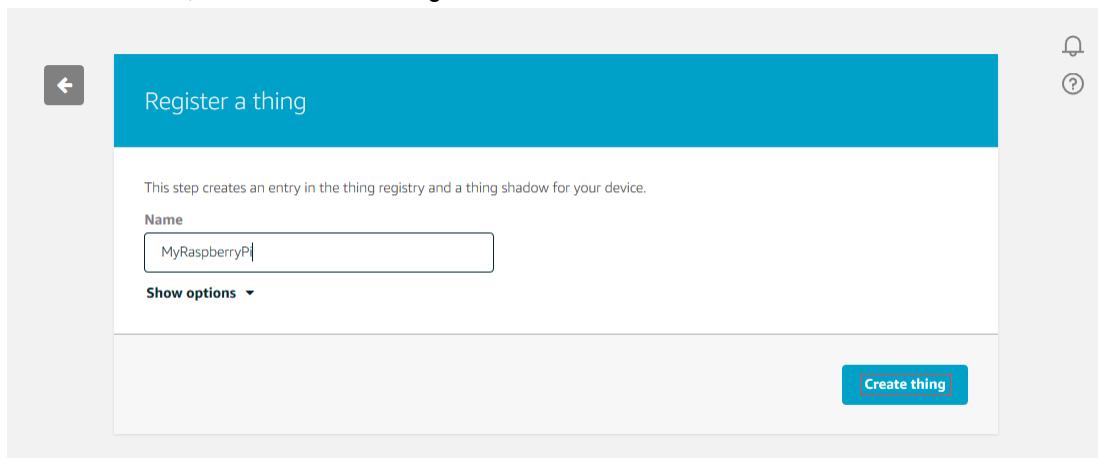
4. 在提示 You don't have any things yet 的页面上，选择 Register a thing。(如果您之前创建过事物，请选择 Create。)



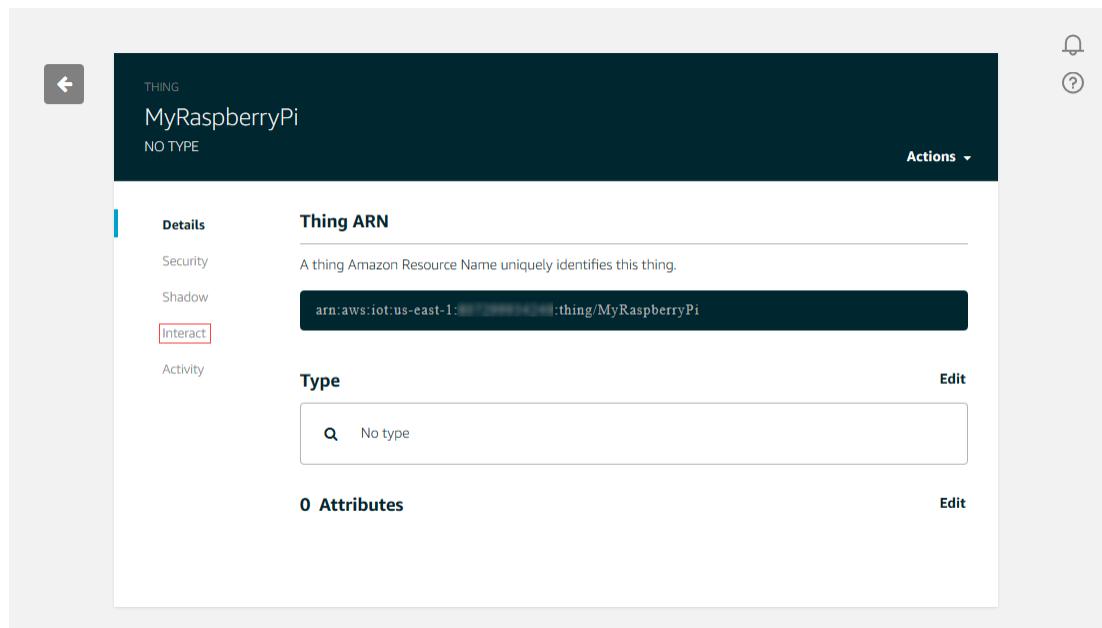
创建并附加事物(设备)

事物是指状态或数据存储在 AWS 云中的设备。Thing Shadows 服务可以让每台设备的事物影子保持连接到 AWS IoT。借助事物影子，您可以访问并修改事物状态数据。

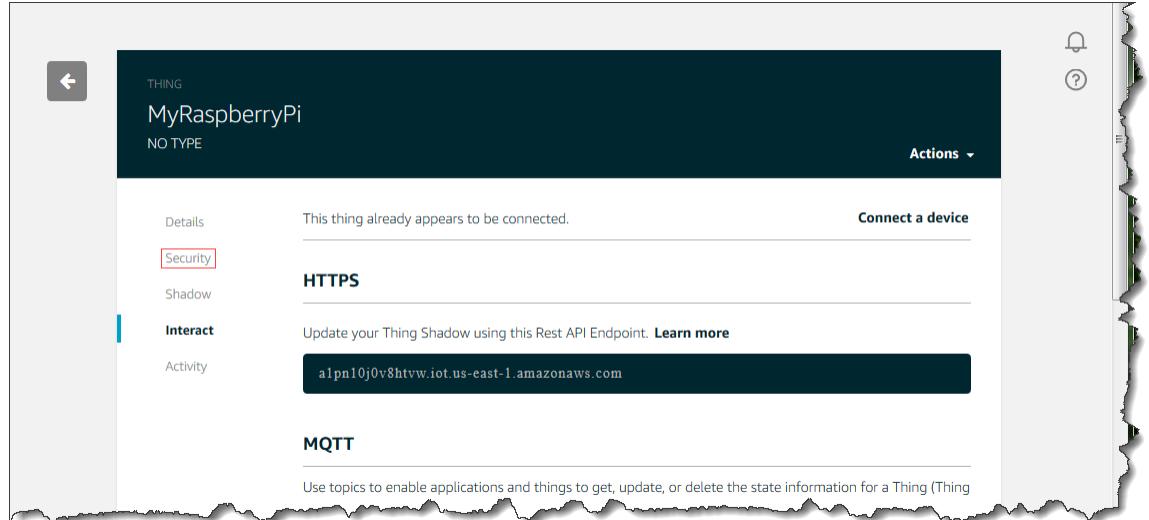
1. 输入事物的名称，然后选择 Create thing。



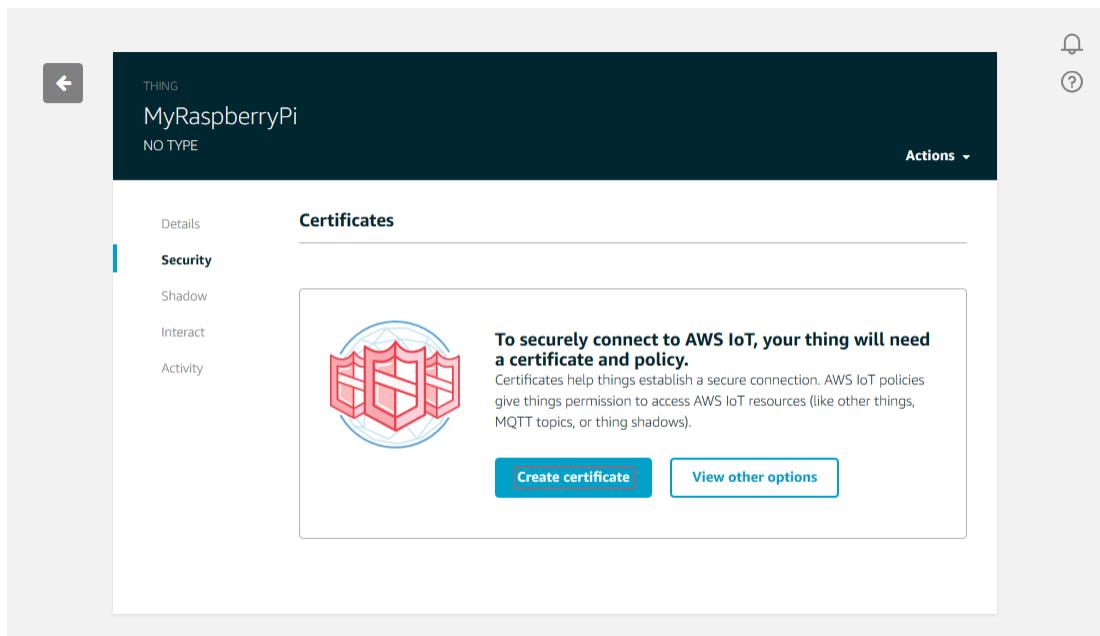
2. 在 Details 页面上，选择 Interact。



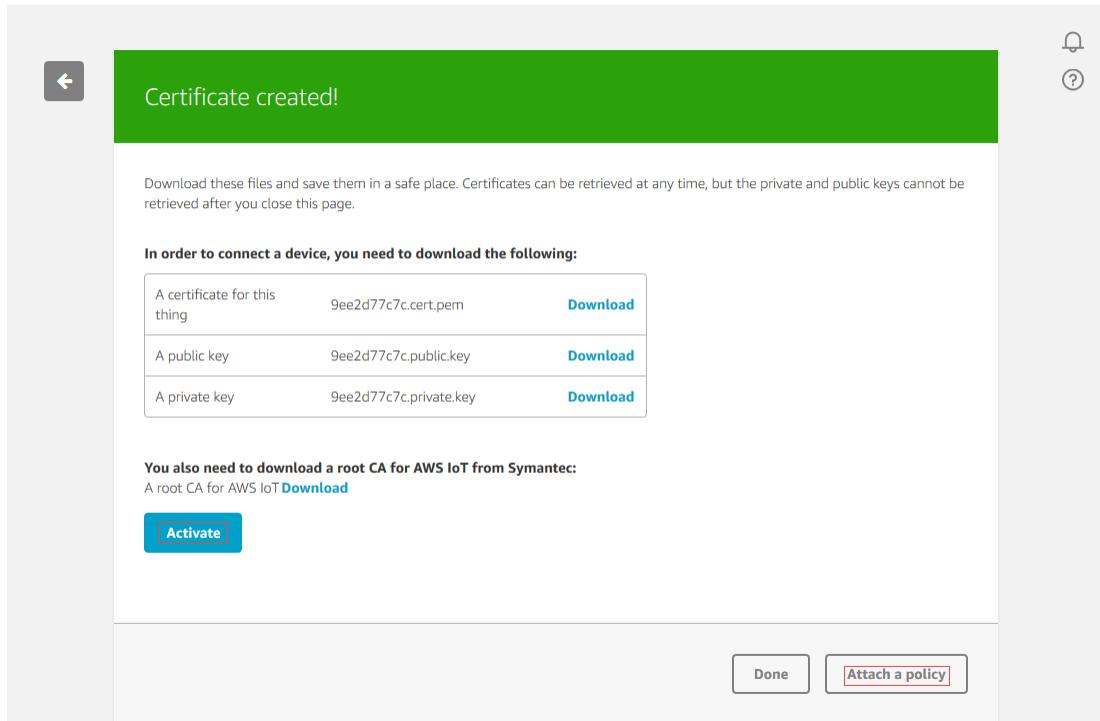
3. 记录 REST API 终端节点。您稍后会需要该值。选择 Security。



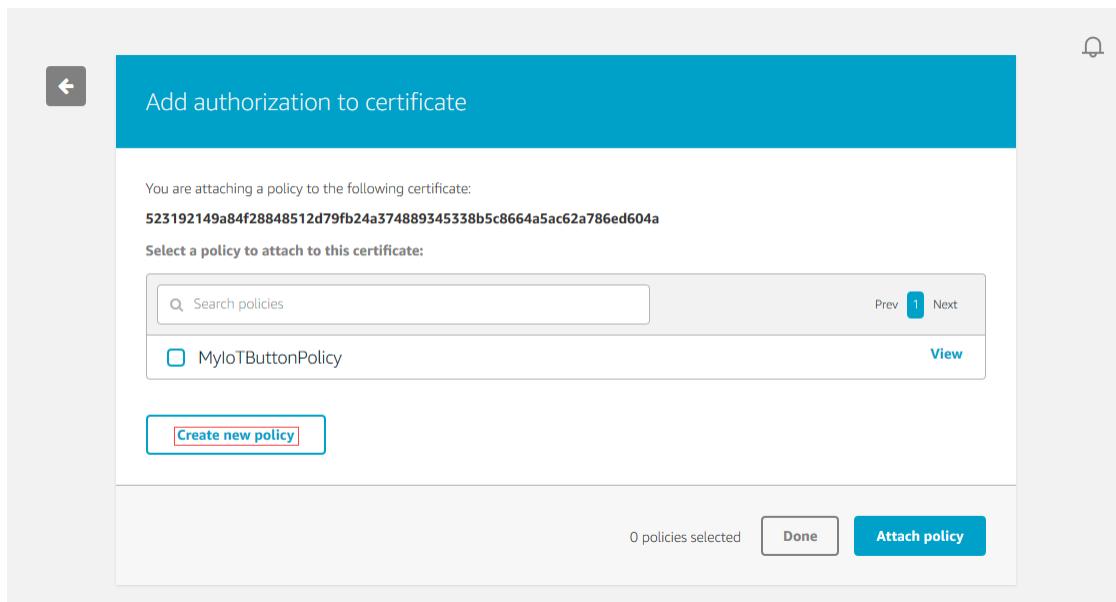
4. 选择 Create certificate。此操作将生成 X.509 证书和密钥对。



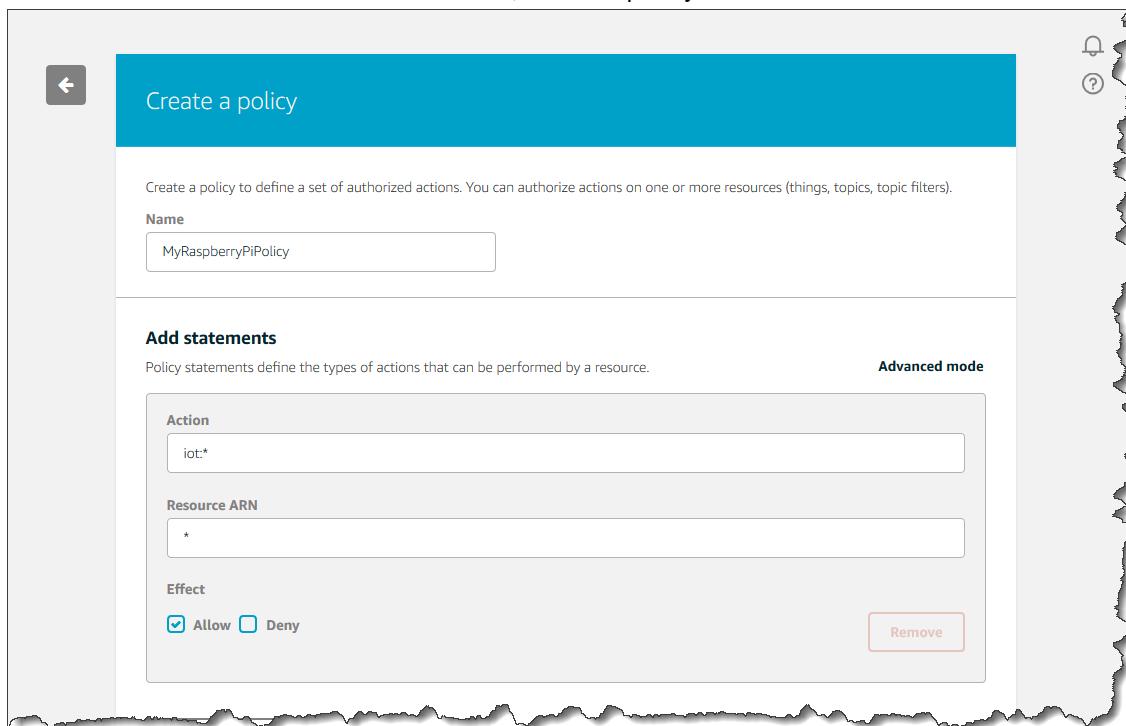
5. 创建一个名为 deviceSDK 的工作目录，以便在其中存储您的文件。选择相应链接以下载公有和私有密钥、证书以及根 CA，并将其保存在 deviceSDK 目录中。选择 Activate 来激活 X.509 证书，然后选择 Attach a policy。



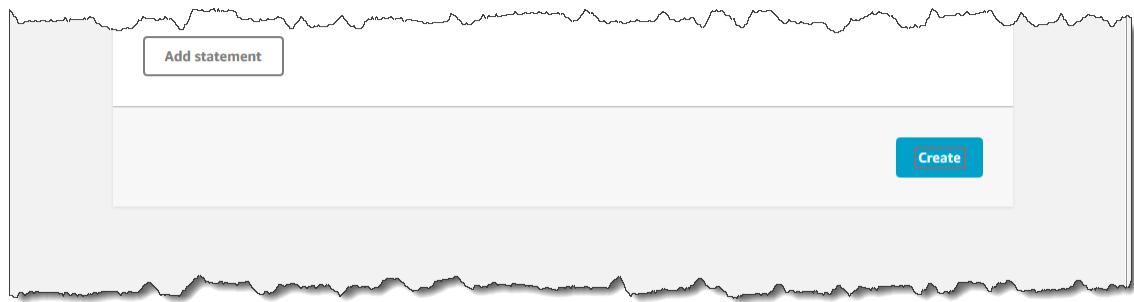
6. 选择 Create new policy。



7. 在 Create a policy 页面上，在 Name 字段键入策略的名称。在 Action 字段键入 `iot:*`。在 Resource ARN 字段中键入 *。选中 Allow 复选框。这样，您的 Raspberry Pi 便可以将向 AWS IoT 发布消息。



8. 选择 Create。

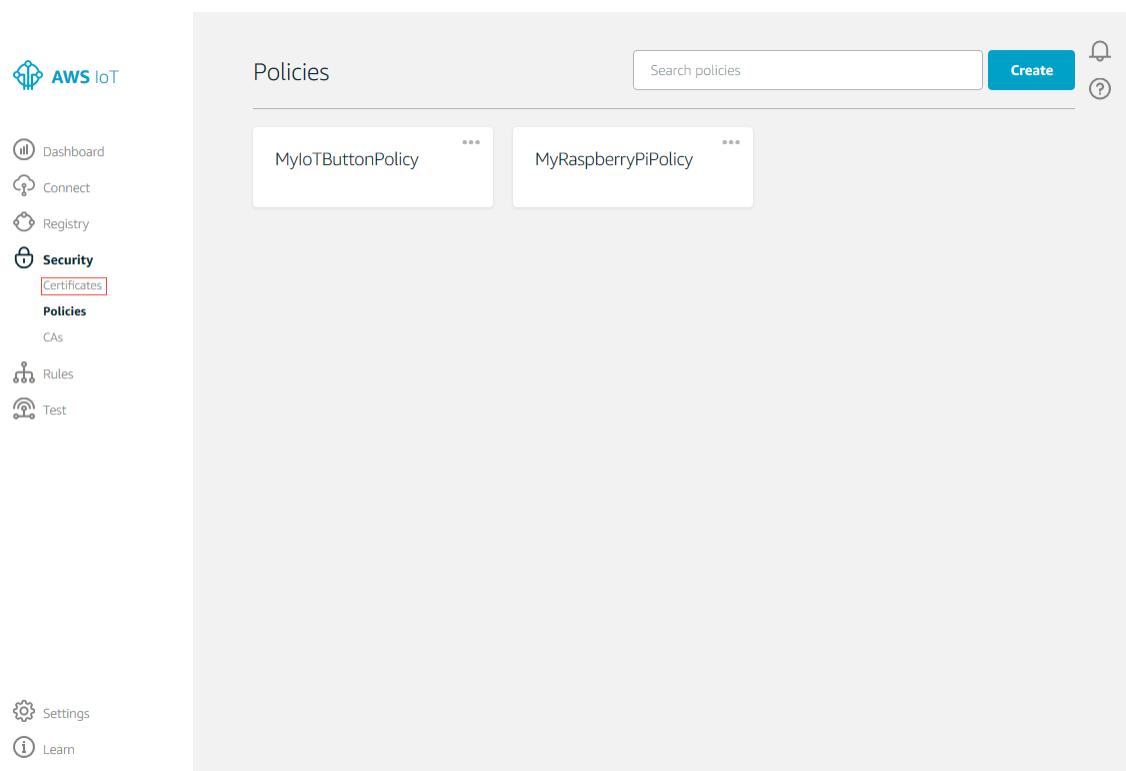


9. 选择左箭头返回 Policies 页面。

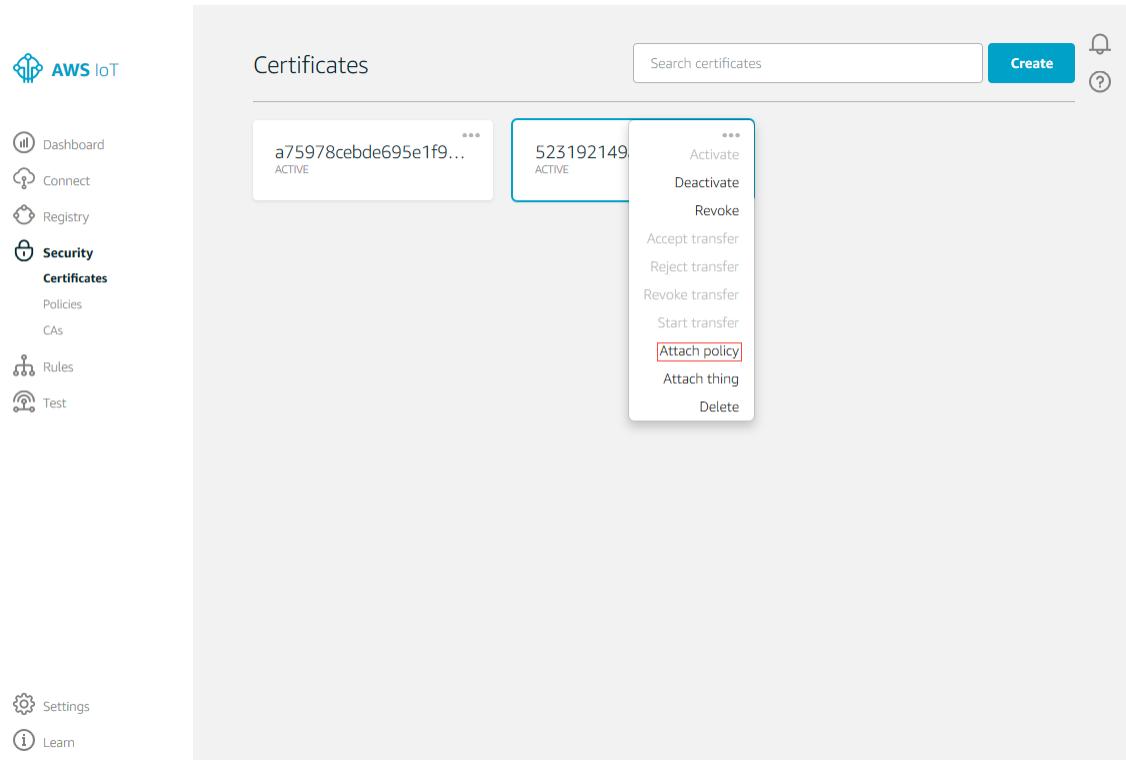
The screenshot displays the AWS IoT Policies page. At the top, it shows a policy named 'MyRaspberryPiPolicy'. Below the title, there are tabs for 'Overview' (which is selected), 'Certificates', and 'Versions'. The 'Overview' tab contains sections for 'Policy ARN' and 'Policy document'. The 'Policy ARN' section shows the ARN: arn:aws:iot:us-east-1::policy/MyRaspberryPiPolicy. The 'Policy document' section shows a JSON document with one statement:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "iot:*,  
      "Resource": "*"  
    }  
  ]  
}
```

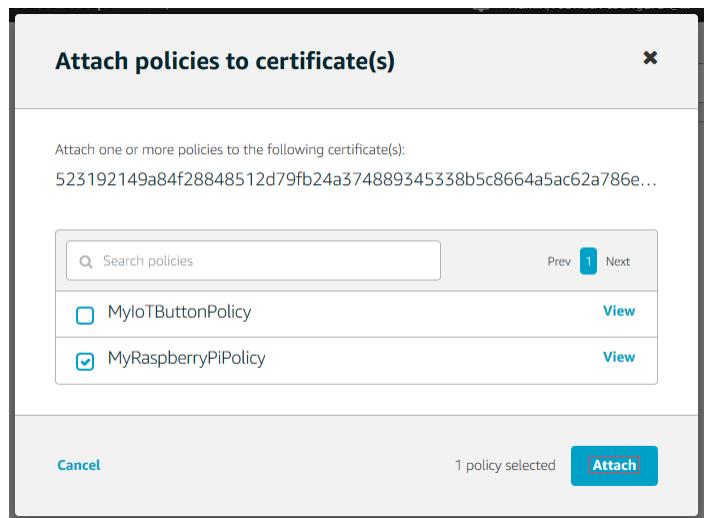
10. 在左侧导航窗格中，在 Security 下选择 Certificates。



11. 在您创建的证书栏，选择 ... 打开下拉菜单，然后选择 Attach policy。



12. 在 Attach policies to certificate(s) 对话框中，选中您已经创建的策略旁边的复选框，然后选择 Attach。



13. 在您创建的证书栏，选择 ... 打开下拉菜单，然后选择 Attach thing。

Certificates

Search certificates

Create

?

Security

Certificates

Policies

CAs

Rules

Test

Settings

Learn

a75978cebde695e1f9... ACTIVE

523192149... ACTIVE

Activate

Deactivate

Revoke

Accept transfer

Reject transfer

Revoke transfer

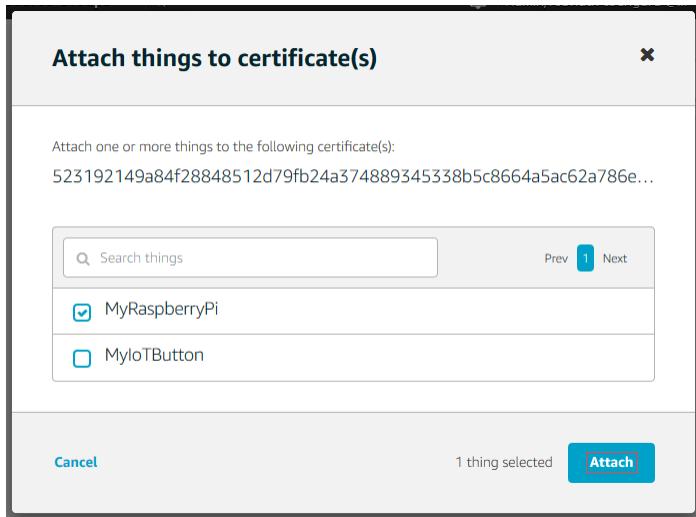
Start transfer

Attach policy

Attach thing

Delete

14. 在 Attach things to certificate(s) 对话框，选中您为代表 Raspberry Pi 而创建的事物旁边的复选框，然后选择 Attach。



使用 AWS IoT 嵌入式 C 软件开发工具包

AWS IoT 嵌入式 C 软件开发工具包有两个版本：OpenSSL 和 mbed TLS。我们使用 OpenSSL 版本。

设置适用于 AWS IoT 嵌入式 C 软件开发工具包的运行时环境

1. 在 tarball 中 (`linux_mqtt_openssl-latest.tar`) 下载适用于 C 语言的 AWS IoT 设备软件工具包。将其保存在您的 `deviceSDK` 目录中。
2. 在终端窗口中，输入以下命令将 tarball 提取到您的 `deviceSDK` 目录中：
`tar -xvf linux_mqtt_openssl-latest.tar`
3. 在使用 AWS IoT 嵌入式 C 软件开发工具包之前，您必须在 Raspberry Pi 上先安装 OpenSSL 库。在终端窗口中，运行
`sudo apt-get install libssl-dev。`

示例应用程序配置

AWS IoT 嵌入式 C 软件开发工具包包括示例应用程序以供您试用。为简便起见，我们将运行 `subscribe_publish_sample`。

1. 将您的证书、私有密钥和根 CA 证书复制到 `deviceSDK/certs` 目录。

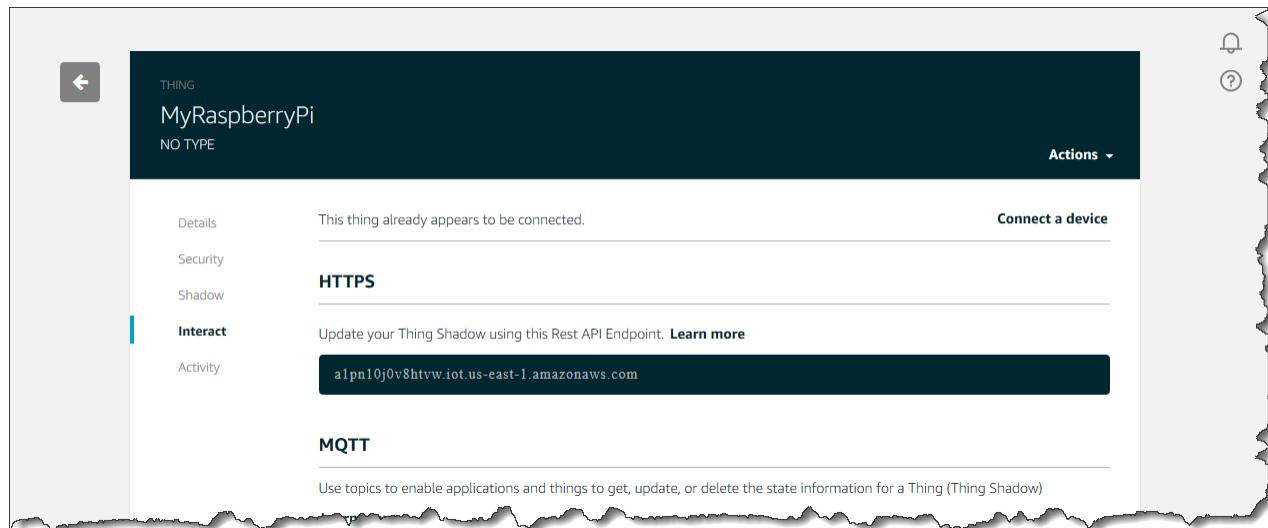
如果您未得到根 CA 证书的副本，您可以在[此处](#)下载。从浏览器复制根 CA 文本，将其粘贴到文件中，然后将其复制到 `deviceSDK/certs` 目录中。

Note

设备和根 CA 证书可能会过期和/或被吊销。如果出现这种情况，您需要将新的 CA 证书或新的私有密钥和设备证书复制到设备上。

2. 导航到 `deviceSDK/sample_apps/subscribe_publish_sample` 目录。您需要配置私有终端节点、私有密钥和证书。私有终端节点是您之前记下的 REST API 终端节点。如果忘记了终端节点，并且

您有权访问安装有 AWS CLI 的设备，您可以使用 `aws iot describe-endpoint` 命令查找您的私有终端节点 URL。转至 AWS IoT 控制台。选择 Registry，选择 Things，然后选择代表 Raspberry Pi 的事物。在事物的 Details 页面上，在左侧导航窗格中，选择 Interact。复制所有内容，包括 REST API 终端节点中的“.com”。



3. 打开 `aws_iot_config.h` 文件并在 `//Get from console` 部分更新以下各项的值：

`AWS_IOT_MQTT_HOST`

您的私有终端节点。

`AWS_IOT_MY_THING_NAME`

您的事物名称。

`AWS_IOT_ROOT_CA_FILENAME`

您的根 CA 证书。

`AWS_IOT_CERTIFICATE_FILENAME`

您的证书。

`AWS_IOT_PRIVATE_KEY_FILENAME`

您的私有密钥。

例如：

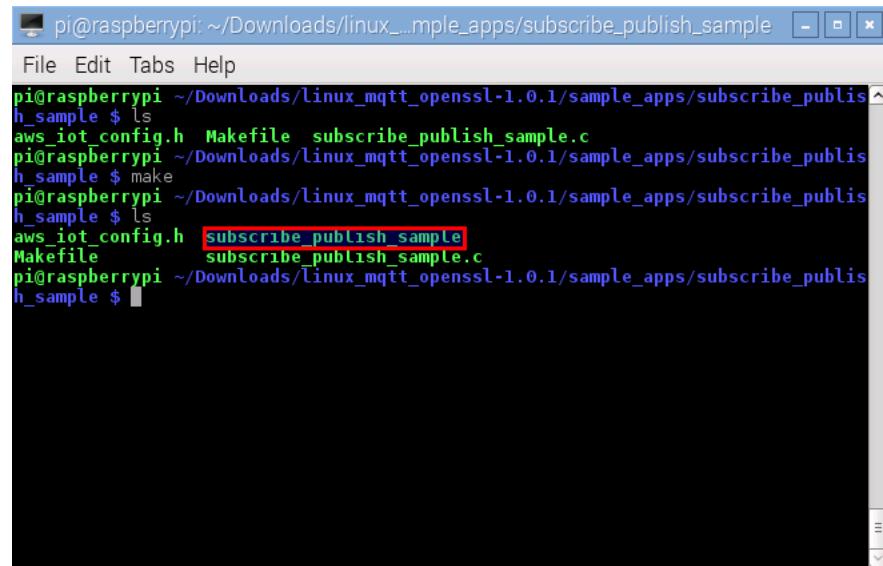
```
// Get from console
// =====
#define AWS_IOT_MQTT_HOST          "a22j5sm6o3yzc5.iot.us-east-1.amazonaws.com"
#define AWS_IOT_MQTT_PORT          8883
#define AWS_IOT_MQTT_CLIENT_ID     "MyRaspberryPi"
#define AWS_IOT_MY_THING_NAME      "MyRaspberryPi"
#define AWS_IOT_ROOT_CA_FILENAME   "root-CA.crt"
#define AWS_IOT_CERTIFICATE_FILENAME "4bbdc778b9-certificate.pem.crt"
#define AWS_IOT_PRIVATE_KEY_FILENAME "4bbdc778b9-private.pem.key"
// =====
```

运行示例应用程序

1. 使用包含的生成文件编译 subscribe_publish_sample_app。

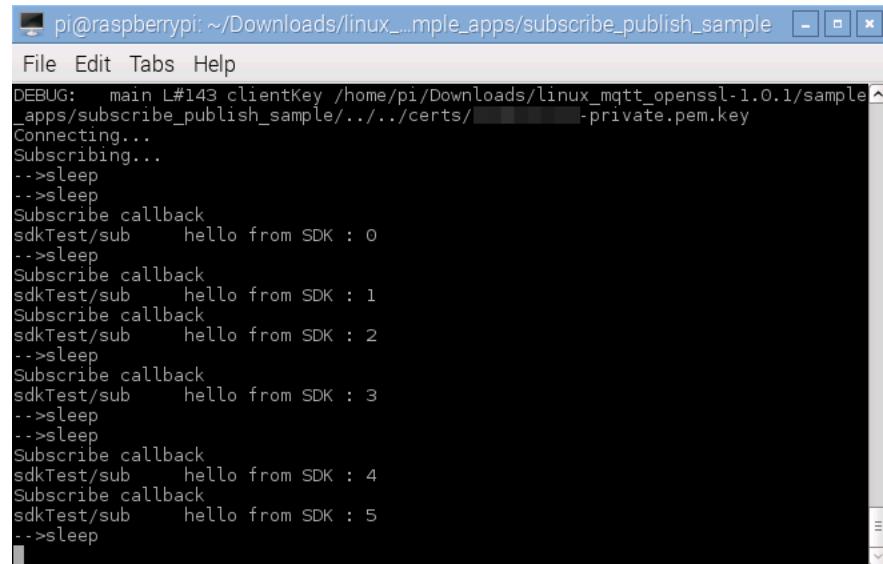
```
make -f Makefile
```

此操作将生成可执行的文件。



The screenshot shows a terminal window titled "pi@raspberrypi: ~/Downloads/linux_mqtt_openssl-1.0.1/sample_apps/subscribe_publish_sample". The terminal displays the command "make -f Makefile" being run, followed by the output of the compilation process. The output includes the AWS IoT configuration header file "aws_iot_config.h" and the source code file "subscribe_publish_sample.c". The compilation is successful, indicated by the message "make: Nothing to be done for 'clean'." at the end.

2. 现在运行 subscribe_publish_sample_app。您应该可以看到类似于如下所示的输出内容：



The screenshot shows a terminal window titled "pi@raspberrypi: ~/Downloads/linux_mqtt_openssl-1.0.1/sample_apps/subscribe_publish_sample". The terminal displays the output of the "subscribe_publish_sample_app" program. The program starts with a DEBUG message indicating it is connecting and subscribing to a topic. It then enters a loop where it receives messages from the SDK, which are printed to the console. The messages are timestamped with "sdkTest/sub" and show an increasing sequence of numbers from 0 to 5, followed by a "hello from SDK" message.

您的 Raspberry Pi 现已通过适用于 C 的 AWS IoT 设备软件开发工具包连接到 AWS IoT。

使用适用于 JavaScript 的 AWS IoT 设备软件开发工具包

安装适用于 Node.js 的 AWS IoT 设备软件开发工具包最简单的方式是使用 npm。在本部分我们将介绍如何安装 Node 和 npm。

设置适用于 JavaScript 的 AWS IoT 设备软件开发工具包的运行时环境

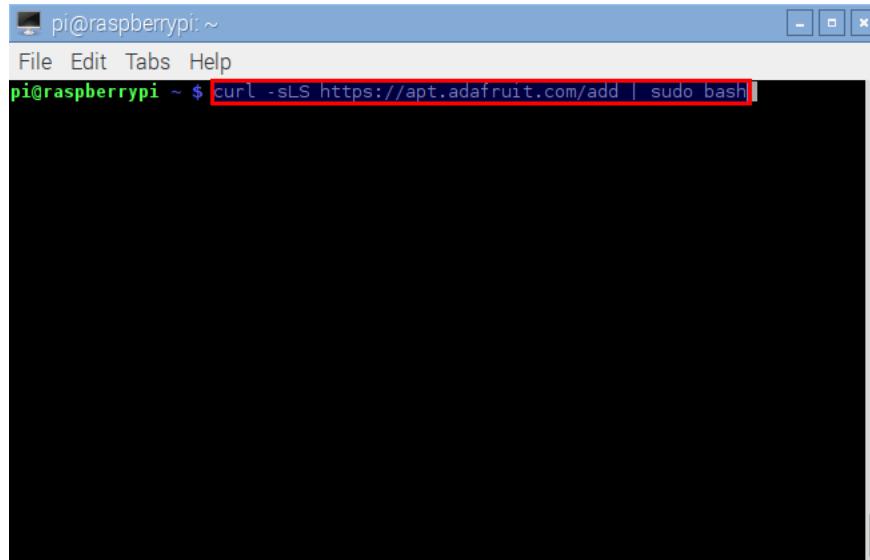
要使用适用于 JavaScript 的 AWS IoT 设备软件开发工具包，您需要在 Raspberry Pi 上安装 Node 和 npm 开发工具。默认情况下，未安装这些软件包。

Note

在继续下一步之前，您可能希望为您的 Raspberry Pi 配置键盘映射。有关更多信息，请参阅 [配置 Raspberry Pi 键盘映射](#)。

1. 要添加 Node 存储库，请打开一个终端并运行以下命令：

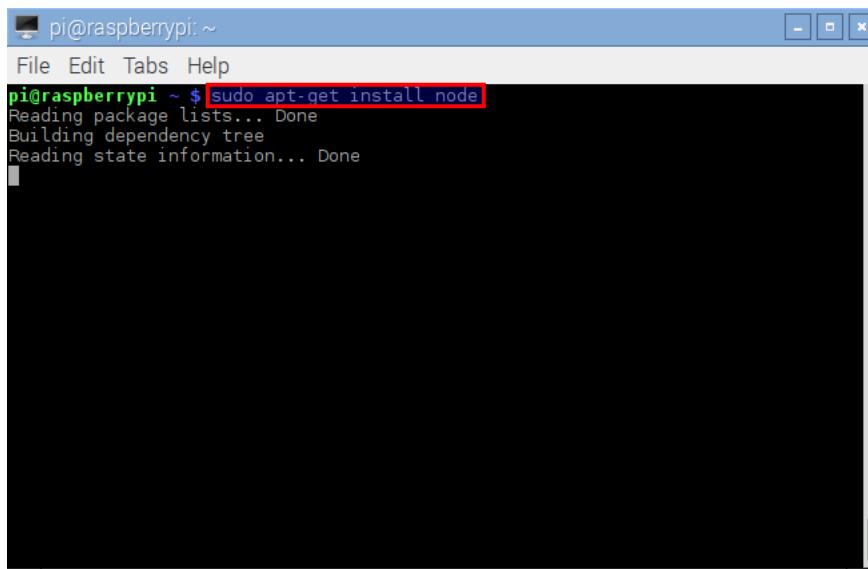
```
curl -sLS https://apt.adafruit.com/add | sudo bash
```



2. 要安装 Node，请运行

```
sudo apt-get install node
```

您应该可以看到类似于如下所示的输出内容：



pi@raspberrypi: ~

File Edit Tabs Help

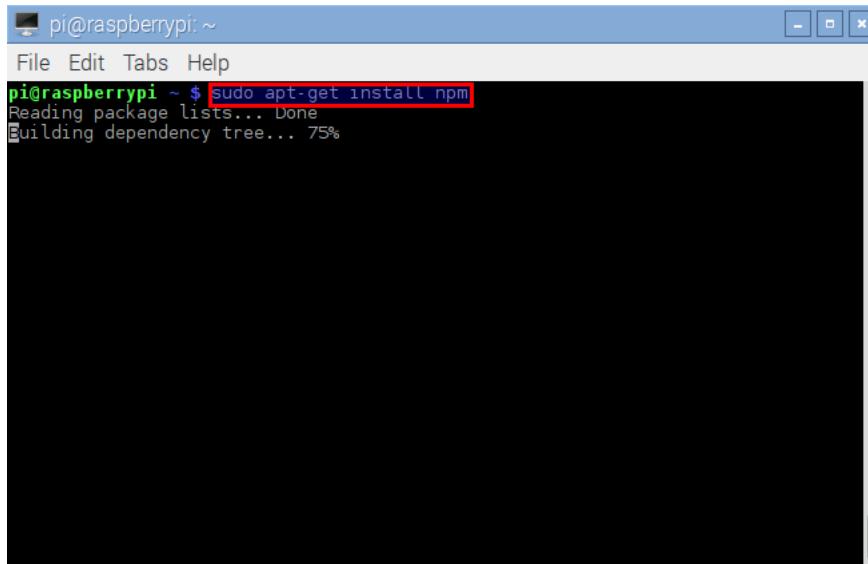
pi@raspberrypi ~ \$ sudo apt-get install node

Reading package lists... Done
Building dependency tree
Reading state information... Done

3. 要安装 npm，请运行

```
sudo apt-get install npm
```

您应该可以看到类似于如下所示的输出内容：



pi@raspberrypi: ~

File Edit Tabs Help

pi@raspberrypi ~ \$ sudo apt-get install npm

Reading package lists... Done
Building dependency tree... 75%

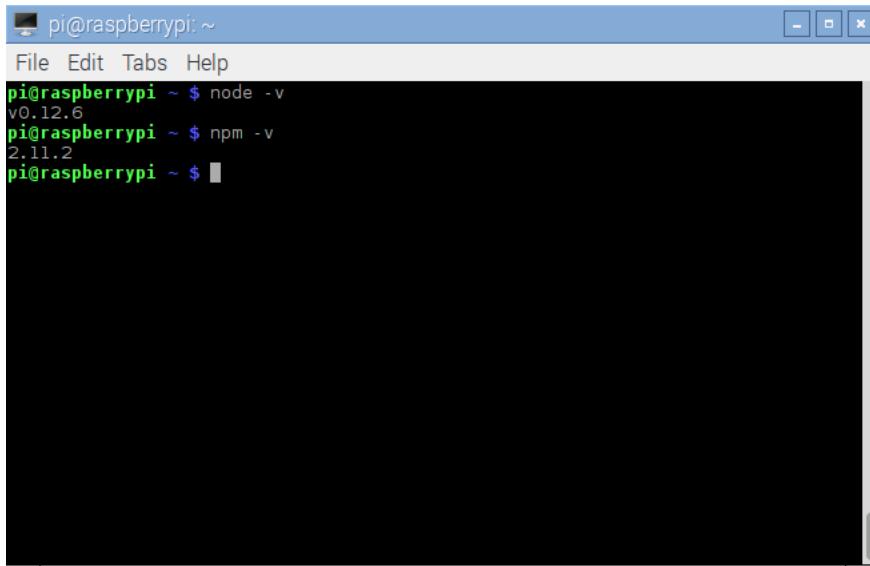
4. 要验证 Node 和 npm 的安装情况，请运行

```
node -v
```

和

```
npm -v
```

您应该可以看到类似于如下所示的输出内容：



安装适用于 JavaScript 的 AWS IoT 设备软件开发工具包

要在您的 Raspberry Pi 中安装适用于 JavaScript/Node.js 的 AWS IoT 设备软件开发工具包，请打开控制台窗口，并从 ~/deviceSDK 目录中使用 npm 来安装软件开发工具包：

```
npm install aws-iot-device-sdk
```

在安装完成后，~/deviceSDK 目录下应该有一个 node_modules 目录。

准备运行示例应用程序

适用于 JavaScript 的 AWS IoT 设备软件开发工具包包括示例应用程序以供您试用。要运行它们，您必须配置证书和私有密钥。

编辑文件 ~/deviceSDK/node_modules/aws-iot-device-sdk/examples/lib/cmdline.js 可更改各示例中私有密钥 (privateKey)、证书 (clientCert) 和 CA 根证书 (caCert) 的默认名称。例如：

```
default: {  
    region: 'us-east-1',  
    clientId: clientIdDefault,  
    privateKey: '4bbdc778b9-private.pem.key',  
    clientCert: '4bbdc778b9-certificate.pem.crt',  
    caCert: 'root-CA.crt',  
    testMode: 1,  
    reconnectPeriod: 3 * 1000, /* milliseconds */  
    delay: 4 * 1000 /* milliseconds */  
};
```

运行示例应用程序

使用以下命令运行示例

```
node examples/<YourDesiredExample>.js -f <certs location>
```

假定您在 ~/deviceSDK/node_modules/aws-iot-device-sdk/ 目录中，则证书位置为 ~/deviceSDK/certs/。有关如何使用命令行选项指定证书位置和您自己的主机地址的更多信息，请参阅[证书](#)。

如果您想创建与命令行选项 --configuration-file (-F) 配合使用的配置文件，请创建包含以下属性的文件 (JSON 格式)。例如：

```
{
    "host": "a22j5sm6o3yzc5.iot.us-east-1.amazonaws.com"
    "port": 8883
    "clientId": "MyRaspberryPi"
    "thingName": "MyRaspberryPi"
    "caCert": "root-CA.crt"
    "clientCert": "4bbdc778b9-certificate.pem.crt"
    "privateKey": "4bbdc778b9-private.pem.key"
}
```

您的 Raspberry Pi 现已通过适用于 JavaScript 的 AWS IoT 设备软件开发工具包连接到 AWS IoT。

使用 AWS IoT 管理事物

AWS IoT 提供了 Thing Registry 来帮助您管理事物。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)。此外，它也可以是逻辑实体(如应用程序实例)，或没有连接到但与其他连接到 AWS IoT 的设备相关的物理实体(例如，装有发动机传感器的汽车或控制面板)。

事物的相关信息均以 JSON 数据形式存储在 Thing 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 Thing Registry 和 Thing Shadows 服务的 MQTT 客户端 ID。这不仅为您的 IoT 队列带来了有序性和便利性，还保留了基础设备证书模型或事物影子的灵活性。

您无需在 Thing Registry 中创建事物便能将其连接到 AWS IoT。将事物添加到 Thing Registry 中会使您能够更加轻松地管理和搜索事物。

利用 Thing Registry 管理事物

您可以使用 AWS IoT 控制台或 AWS CLI 与 Thing Registry 进行交互。以下各部分展示了如何使用 CLI 与 Thing Registry 进行交互。

创建事物

以下命令展示了如何使用 AWS IoT `create-thing` CLI 命令创建事物：

```
$ aws iot create-thing --thing-name "MyLightBulb" --attribute-payload "{\"attributes\":{\"wattage\":\"75\", \"model\":\"123\"}}"
```

create-thing API 会显示新事物的名称和 ARN：

```
{  
    "thingArn": "arn:aws:iot:us-east-1:803981987763:thing/MyLightBulb",  
    "thingName": "MyLightBulb"  
}
```

列出事物

您可以使用 list-things API 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"  
        }  
    ]  
}
```

搜索事物

您可以使用 describe-thing API 列出某个事物的相关信息：

```
$ aws iot describe-thing --thing-name "MyLightBulb"  
{  
    "version": 3,  
    "thingName": "MyLightBulb",  
    "defaultClientId": "MyLightBulb",  
    "thingTypeName": "StopLight",  
    "attributes": {  
        "model": "123",  
        "wattage": "75"  
    }  
}
```

您可以使用 list-things API 搜索与某个事物类型名称相关联的所有事物：

```
$ aws iot list-things --thing-type-name "LightBulb"
```

```
{
```

```
"things": [
  {
    "thingType": "LightBulb",
    "version": 1,
    "thingName": "MyRGBLight"
  },
  {
    "thingType": "LightBulb",
    "version": 1,
    "thingName": "MySecondLightBulb"
  }
]
```

您可以使用 `list-things` API 搜索某个属性为特定值的所有事物：

```
$ aws iot list-things --attribute-name "wattage" --attribute-value "75"
```

```
{
  "things": [
    {
      "thingType": "StopLight",
      "version": 3,
      "thingName": "MyLightBulb"
    },
    {
      "thingType": "LightBulb",
      "version": 1,
      "thingName": "MyRGBLight"
    },
    {
      "thingType": "LightBulb",
      "version": 1,
      "thingName": "MySecondLightBulb"
    }
  ]
}
```

更新事物

您可以使用 `update-thing` API 更新事物：

```
$ aws iot update-thing --thing-name "MyLightBulb" --attribute-payload "{\"attributes\":{\"wattage\":\"150\", \"model\":\"456\"}}"
```

update-thing 命令不会生成任何输出。您可以使用 describe-thing API 查看结果：

```
$ aws iot describe-thing --thing-name "MyLightBulb"
{
    "attributes": {
        "model": "456",
        "wattage": "150"
    },
    "version": 2,
    "thingName": "MyLightBulb"
}
```

删除事物

您可以使用 delete-thing API 删除事物：

```
$ aws iot delete-thing --thing-name "MyThing"
```

将委托人附加到事物

物理设备必须具有 X.509 证书才能与 AWS IoT 通信。您可以将设备上的证书与 Thing Registry 中代表该设备的事物关联。要将证书附加到事物，请使用 attach-thing-principal API：

```
$ aws iot attach-thing-principal --thing-name "MyLightBulb" --principal "arn:aws:iot:us-east-1:123456789012:cert/a0c01f5835079de0a7514643d68ef8414ab739a1e94ee4162977b02b12842847"
```

attach-thing-principal 命令不会生成任何输出。

将委托人与事物分离

您可以使用 detach-thing-principal API 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"
```

detach-thing-principal 命令不会生成任何输出。

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 Thing Registry 中的事物管理。例如，您可以定义 LightBulb 事物类型。所有与 LightBulb 事物类型相关联的事物均具有一组共同的属性：序列号、制造商和功率。创建 LightBulb 类型的事物（或将现有事物的类型更改为 LightBulb）时，您可以指定在 LightBulb 事物类型中定义的每个属性的值。

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。事物类型名称一旦创建，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 `create-thing-type` API 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"
```

`create-thing-type` 命令会返回一个含有事物类型及其 ARN 的响应：

```
{
  "thingTypeName": "LightBulb",
  "thingTypeArn": "arn:aws:iot:us-west-2:803981987763:thingtype/LightBulb"
}
```

列出事物类型

您可以使用 `list-thing-types` API 列出事物类型：

```
$ aws iot list-thing-types
```

`list-thing-types` 命令会返回一个列有您的 AWS 账户中定义的事物类型的列表：

```
{
  "thingTypes": [
    {
      "thingTypeName": "LightBulb",
      "thingTypeProperties": {
        "deprecated": false,
        "creationDate": 1468423800950,
        "searchableAttributes": [
          "wattage",
          "model"
        ],
        "thingTypeDescription": "light bulb type"
      }
    }
  ]
}
```

描述事物类型

您可以使用 `describe-thing-type` API 获取某事物类型的相关信息：

```
$ aws iot describe-thing-type --thing-type-name "LightBulb"
```

`describe-thing-type` API 会使用指定事物类型的相关信息进行响应：

```
{
```

```
"thingTypeName": "LightBulb",
"thingTypeProperties": {
    "deprecated": false,
    "creationDate": 1468423800950,
    "searchableAttributes": [
        "wattage",
        "model"
    ],
    "thingTypeDescription": "light bulb type"
}
}
```

将事物类型与事物相关联

创建事物时，可以使用 `create-thing` API 指定事物类型：

```
$ aws iot create-thing --thing-name "MySecondLightBulb" --thing-type-name "LightBulb" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

您随时可以使用 `update-thing` API 更改与某个事物关联的事物类型：

```
$ aws iot update-thing --thing-name "MyLightBulb" --thing-type-name "StopLight" --
attribute-payload "{\"attributes\": {\"wattage\":\"75\", \"model\":\"123\"}}"
```

您还可以使用 `update-thing` API 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。一旦定义了事物类型，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 `deprecate-thing-type` API：

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType"
```

您可以使用 `describe-thing-type` API 查看结果：

```
$ aws iot describe-thing --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeProperties": {
        "deprecated": true,
        "creationDate": 1468425854308,
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type",
        "deprecationDate": 1468446026349
    }
}
```

弃用事物类型是可逆操作。您可以通过在 `deprecate-thing-type` CLI 命令中使用 `--undo-deprecate` 标志来撤消弃用：

```
$ aws iot deprecate-thing-type --thing-type-name "myThingType" --undo-deprecate
```

您可以使用 `deprecate-thing-type` CLI 命令查看结果：

```
$ aws iot deprecate-thing-type --thing-type-name "StopLight":
```

```
{
    "thingTypeName": "StopLight",
    "thingTypeProperties": {
        "deprecated": false,
        "creationDate": 1468425854308,
        "searchableAttributes": [
            "wattage",
            "numOfLights",
            "model"
        ],
        "thingTypeDescription": "traffic light type"
    }
}
```

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 `delete-thing-type` API：

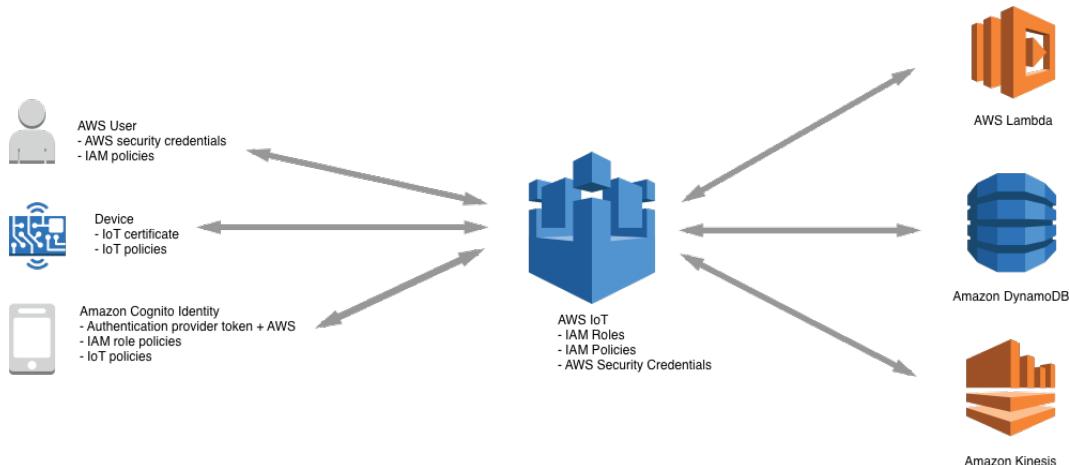
```
$ aws iot delete-thing-type --thing-type-name "StopLight"
```

Note

弃用某事物类型后，必须等待 5 分钟，然后才能将其删除。

AWS IoT 的安全和身份

所连接的每台设备必须拥有凭证才能访问消息代理或 Thing Shadows 服务。对于往返 AWS IoT 的所有流量，都必须通过传输层安全性 (TLS) 进行加密。必须保证设备凭证的安全，以便安全地将数据发送到消息代理。数据在 AWS IoT 和其他设备或 AWS 服务之间移动时，AWS 云安全机制可为数据提供保护。



- 您负责管理设备上的设备凭证 (X.509 凭证、AWS 凭证) 及 AWS IoT 中的策略。您负责将唯一身份分配给每台设备并管理设备或设备组的权限。
- 设备将按照 AWS IoT 连接模式使用您选择的身份 (X.509 证书、IAM 用户和组，或者 Amazon Cognito 身份) 通过安全连接来建立连接。
- AWS IoT 消息代理针对账户内的所有操作执行身份验证和授权。消息代理负责对设备执行身份验证、安全地接收设备数据，以及支持您通过策略授予设备的访问权限。
- 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和其他 AWS 服务。它使用 AWS 访问管理系统将数据安全地传输到最终目的地。

AWS IoT 中的身份验证

AWS IoT 支持使用四种身份委托人进行身份验证：

- X.509 证书

- IAM 用户、组和角色
- Amazon Cognito 身份
- 联合身份

这些身份可用于移动应用程序、Web 应用程序或桌面应用程序。它们甚至可由键入 AWS IoT CLI 命令的用户使用。通常，AWS IoT 设备使用 X.509 证书，移动应用程序使用 Amazon Cognito 身份。Web 和桌面应用程序使用 IAM 或联合身份。CLI 命令使用 IAM。

X.509 证书

X.509 证书属于数字证书，它按照 X.509 公有密钥基础设施标准将公有密钥与证书所含的身份相关联。X.509 证书由一家名为证书颁发机构 (CA) 的可信实体颁发。CA 持有一个或多个名为 CA 证书的特殊证书，它使用这种证书来颁发 X.509 证书。只有证书颁发机构才有权访问 CA 证书。

Note

我们建议为每个设备提供一个唯一的证书，以便进行精细的管理，包括证书吊销。

Note

设备必须支持轮换和更换证书，以确保证书过期时仍能顺畅运行。

AWS IoT 支持以下证书签名算法：

- SHA256WITHRSA
- SHA384WITHRSA
- SHA384WITHRSA
- SHA512WITHRSA
- RSASSAPSS
- DSA_WITH_SHA256
- ECDSA-WITH-SHA256
- ECDSA-WITH-SHA384
- ECDSA-WITH-SHA512

与其他身份和身份验证机制相比，证书具有多项优势。证书可将非对称密钥用于设备。这意味着您可以将私有密钥融入到设备上的安全存储中。这样，敏感的加密材料就永远不会离开设备。证书可以通过用户名和密码或持有者令牌等其他方案提供更可靠的客户端身份验证，因为密钥永远不会离开设备。

AWS IoT 身份验证证书采用 TLS 协议的客户端身份验证模式。TLS 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 请求客户端 X.509 证书并依照证书注册表验证证书的状态和 AWS 账户。随后，将要求客户端提供与证书所含公有密钥相对应的私有密钥的所有权证明。

要使用 AWS IoT 证书，客户端必须支持在 TLS 实施中使用所有以下各项：

- TLS 1.2.
- SHA-256 RSA 证书签名验证。
- 来自 TLS 密码包支持部门的密码包之一。

X.509 证书和 AWS IoT

AWS IoT 可以使用 AWS IoT 生成的证书或 CA 证书签发的证书执行设备身份验证。AWS IoT 生成的证书不会过期。对于 CA 证书签发的证书的过期日期和时间，将在创建 CA 证书时设置。

Note

我们建议为每个设备提供一个唯一的证书，以便进行精细的管理，包括证书吊销。

Note

设备必须支持轮换和更换证书，以确保证书过期时仍能顺畅运行。

要使用并非由 AWS IoT 创建的证书，您必须注册一个 CA 证书。所有设备证书都必须由您注册的 CA 证书签发。

您可以使用 AWS IoT 控制台或 CLI 执行以下操作：

- 创建并注册 AWS IoT 证书。
- 注册 CA 证书。
- 注册设备证书。
- 激活或停用设备证书。
- 撤销设备证书。
- 将设备证书转移到其他 AWS 账户。
- 列出注册到您的 AWS 账户的所有 CA 证书。
- 列出注册到您的 AWS 账户的所有设备证书。

有关用于执行这些操作的 CLI 命令的详细信息，请参阅 [AWS IoT CLI 参考](#)。

有关使用 AWS IoT 控制台创建证书的更多信息，请参阅[创建和激活设备证书](#)。

服务器身份验证

利用设备证书，AWS IoT 可以对设备执行身份验证。要确保您设备的通信对象是 AWS IoT 而非假冒 AWS IoT 的其他服务器，请将 [VeriSign Class 3 Public Primary G5 根 CA 证书](#) 复制到您的设备上。

Note

该 CA 证书在 2036 年 7 月之前一直有效，但在此之前可能需要更换 CA 证书。您应该确保可以更新所有设备上的根 CA 证书，以确保持续的连接并了解最新的安全最佳实践。

在连接到 AWS IoT 时，请参考您的设备代码中的 CA 根证书。有关更多信息，请参阅 [AWS IoT 设备软件开发工具包 \(p. 236\)](#)。

Note

您不能使用自己的 CA 证书对 AWS IoT 服务器进行身份验证，而是必须使用 VeriSign Class 3 Public Primary G5 根 CA 证书。

创建并注册 AWS IoT 设备证书

您可以使用 AWS IoT 控制台或 AWS IoT CLI 来创建 AWS IoT 证书。

创建证书 (控制台)

1. 登录 AWS 管理控制台并打开 [AWS IoT 控制台](#)，网址为：<https://console.aws.amazon.com/iot>。
2. 在左侧导航窗格中，选择 Security 展开选项，然后选择 Certificates。选择 Create。
3. 选择 One-click certificate creation - Create certificate。或者，要使用证书签名请求 (CSR) 生成证书，请选择 Create with CSR。
4. 使用指向公有密钥、私有密钥和证书的链接将每一个密钥和证书都下载到一个安全位置。

5. 选择 Activate。

创建证书 (CLI)

AWS IoT CLI 提供了两个命令来创建证书：

- [create-keys-and-certificate](#)

[CreateKeysAndCertificate](#) API 可创建私有密钥、公有密钥和 X.509 证书。

- [create-certificate-from-csr](#)

[CreateCertificateFromCSR](#) API 利用 CSR 创建证书。

使用您自己的证书

要使用您自己的 X.509 证书，必须在 AWS IoT 中注册 CA 证书。之后，CA 证书可用于签发设备证书。您可以使用相同的使用者字段为每个区域的每个 AWS 账户最多注册 10 个 CA 证书。这样，您便可以拥有多个能够签发设备证书的 CA。

Note

设备证书必须由已注册的 CA 证书签发。CA 证书通常用于创建中间 CA 证书。如果使用中间证书签发设备证书，则您必须注册中间 CA 证书。在连接到 AWS IoT 时应使用 AWS IoT 根 CA 证书，即使您注册了自己的根 CA 证书也是如此。设备使用 AWS IoT 根 CA 证书验证 AWS IoT 服务器的身份。

内容

- [注册您的 CA 证书 \(p. 103\)](#)
- [使用您的 CA 证书创建设备证书 \(p. 104\)](#)
- [注册设备证书 \(p. 105\)](#)
- [手动注册设备证书 \(p. 105\)](#)
- [使用自动/即时注册功能来注册设备证书 \(p. 105\)](#)
- [停用 CA 证书 \(p. 106\)](#)
- [吊销设备证书 \(p. 106\)](#)

如果没有 CA 证书，则可以使用 [OpenSSL](#) 工具创建一个。

创建 CA 证书

1. 生成密钥对。

```
openssl genrsa -out rootCA.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 rootCA.key -sha256 -days 1024 -out rootCA.pem
```

注册您的 CA 证书

要注册您的 CA 证书，您必须：

- 从 AWS IoT 获取注册代码。
- 使用您的 CA 证书来签署私有密钥验证证书。

- 将您的 CA 证书和私有密钥验证证书传递给 register-ca-certificate CLI 命令。

必须将私有密钥验证证书中的 Common Name 字段设置为 get-registration-code CLI 命令生成的注册代码。命令会为每个 AWS 账户生成一个注册代码。您可以使用 register-ca-certificate 命令或 AWS IoT 控制台注册 CA 证书。

注册 CA 证书

- 从 AWS IoT 获取注册代码。此代码将用作私有密钥验证证书的 Common Name。

```
aws iot get-registration-code
```

- 为私有密钥验证证书生成密钥对。

```
openssl genrsa -out verificationCert.key 2048
```

- 为私有密钥验证证书创建 CSR。将证书的 Common Name 字段设置为您的注册代码。

```
openssl req -new -key verificationCert.key -out verificationCert.csr
```

系统将提示您输入一些信息，其中包括证书的 Common Name。

```
Country Name (2 letter code) [AU]:  
State or Province Name (full name) []:  
Locality Name (eg, city) []:  
Organization Name (eg, company) []:  
Organizational Unit Name (eg, section) []:  
Common Name (e.g. server FQDN or YOUR name) []:XXXXXXXXXXXXMYREGISTRATIONCODEXXXXXX  
Email Address []:
```

- 使用 CSR 创建私有密钥验证证书。

```
openssl x509 -req -in verificationCert.csr -CA rootCA.pem -CAkey rootCA.key -  
CAcreateserial -out verificationCert.pem -days 500 -sha256
```

- 向 AWS IoT 注册 CA 证书。将 CA 证书和私有密钥验证证书传递给 register-ca-certificate CLI 命令。

```
aws iot register-ca-certificate --ca-certificate file://rootCA.pem --verification-cert  
file://verificationCert.pem
```

- 使用 update-certificate CLI 命令激活 CA 证书。

```
aws iot update-ca-certificate --certificate-id XXXXXXXXXXXX --new-status ACTIVE
```

使用您的 CA 证书创建设备证书

您可以使用在 AWS IoT 中注册的 CA 证书创建设备证书。必须先在 AWS IoT 中注册设备证书，然后才能使用它。

创建设备证书

- 生成密钥对。

```
openssl genrsa -out deviceCert.key 2048
```

2. 为设备证书创建 CSR。

```
openssl req -new -key deviceCert.key -out deviceCert.csr
```

系统将提示您输入一些信息，如下所示。

```
Country Name (2 letter code) [AU]:  
State or Province Name (full name) []:  
Locality Name (eg, city) []:  
Organization Name (eg, company) []:  
Organizational Unit Name (eg, section) []:  
Common Name (e.g. server FQDN or YOUR name) []:  
Email Address []:
```

3. 使用 CSR 创建设备证书。

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -  
out deviceCert.pem -days 500 -sha256
```

Note

您必须使用在 AWS IoT 中注册的 CA 证书创建设备证书。如果您在自己的 AWS 账户中注册了多个 CA 证书（具有相同的使用者字段和公有密钥），则必须在注册设备证书时指定用于创建设备证书的 CA 证书。

4. 注册设备证书。

```
aws iot register-certificate --certificate-pem file://deviceCert.pem --ca-certificate-pem  
file://rootCA.pem
```

5. 使用 update-certificate CLI 命令激活设备证书。

```
aws iot update-certificate --certificate-id xxxxxxxxxxxx --new-status ACTIVE
```

注册设备证书

您必须使用在 AWS IoT 中注册的 CA 证书签发设备证书。如果您在自己的 AWS 账户中注册了多个 CA 证书（具有相同的使用者字段和公有密钥），则必须在注册设备证书时指定用于签发设备证书的 CA 证书。您可以手动注册每个设备证书，也可以使用自动注册功能，利用该功能，设备可以在首次连接到 AWS IoT 时注册自己的证书。

手动注册设备证书

使用以下 CLI 命令注册设备证书：

```
aws iot register-certificate --certificate-pem file://deviceCert.crt --ca-certificate-pem  
file://caCert.crt
```

使用自动/即时注册功能来注册设备证书

要在设备首次连接到 AWS IoT 时自动注册设备证书，您必须启用 CA 证书的自动注册。这将在设备连接到 AWS IoT 时注册由您的 CA 证书签发的任何设备证书。

启用自动注册

使用 update-ca-certificate API 将 CA 证书的 auto-registration-status 设置为 ENABLE：

```
$ aws iot update-ca-certificate --cert-id caCertificateId --new-auto-registration-status  
ENABLE
```

使用 `register-ca-certificate` API 注册 CA 证书时，还可以将 `auto-registration-status` 设置为 `ENABLE`：

```
aws iot register-ca-certificate --ca-certificate file://rootCA.pem --verification-cert  
file://privateKeyVerificationCert.crt --allow-auto-registration
```

当设备首次尝试连接到 AWS IoT 时，作为 TLS 握手流程的一部分，它必须提供已注册的 CA 证书和设备证书。AWS IoT 会将 CA 证书识别为已注册的 CA 证书，自动注册设备证书并将其状态设置为 `PENDING_ACTIVATION`。这意味着，已自动注册设备证书，该证书正在等待激活。证书必须处于 `ACTIVE` 状态才能用来连接到 AWS IoT。当 AWS IoT 自动注册证书时或者当处于 `PENDING_ACTIVATION` 状态的证书建立连接时，AWS IoT 将向以下 MQTT 主题发布消息：

```
$aws/events/certificates/registered/caCertificateID
```

其中 `caCertificateID` 是颁发设备证书的 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，以验证设备证书是否位于证书吊销列表 (CRL) 中，激活证书，创建策略并将其附加到证书中。该策略可确定设备能够访问的资源。有关如何创建可侦听 `$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[即时注册](#)。

停用 CA 证书

当您注册设备证书时，AWS 将检查相关的 CA 证书是否处于 `ACTIVE` 状态。如果 CA 证书的状态是 `INACTIVE`，AWS IoT 将禁止注册设备证书。通过将 CA 证书标记为 `INACTIVE`，可以防止在账户中注册由受损的 CA 颁发的任何新设备证书。您可以使用 `update-ca-certificate` API 停用 CA 证书：

```
$ aws iot update-ca-certificate --cert-id certificateId --new-status INACTIVE
```

Note

除非您明确吊销由受损的 CA 证书签发的已注册设备证书，否则所有此类证书均将继续工作。

使用 `ListCertificatesByCA` API 获取由受损的 CA 签发的所有已注册设备证书的列表。对于由受损的 CA 证书签发的每个设备证书，请使用 `UpdateCertificate` API 吊销该设备证书以避免使用它。

吊销设备证书

如果您在已注册的设备证书上检测到可疑活动，可使用 `update-certificate` API 吊销该证书：

```
$ aws iot update-certificate --cert-id certificateId  
--new-status REVOKED
```

如果在自动注册设备证书期间发生任何错误或异常，AWS IoT 将向您的 CloudWatch Logs 日志发送相关事件或消息。有关如何为您的账户设置日志的更多信息，请参阅 [Amazon CloudWatch 文档](#)。

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。您可以使用它们通过 AWS 软件开发工具包和 CLI 连接到 AWS IoT HTTP 界面。

IAM 角色还允许 AWS IoT 代表您访问您账户中的其他 AWS 资源。例如，如果您希望设备将自身状态发布到 DynamoDB 表，则 IAM 角色允许 AWS IoT 与 Amazon DynamoDB 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，AWS IoT 可以使用签名版本 4 签名流程对 IAM 用户、组和角色执行身份验证。有关信息，请参阅[签署 AWS API 请求](#)。

将 AWS Signature 版本 4 与 AWS IoT 配合使用时，客户端必须支持在 TLS 实施中使用以下各项：

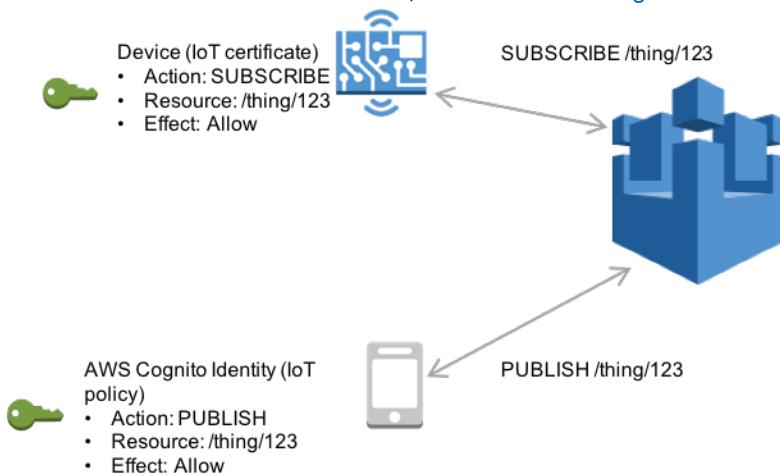
- TLS 1.2、TLS 1.1、TLS 1.0。
- SHA-256 RSA 证书签名验证。
- 来自 TLS 密码包支持部门的密码包之一。

有关更多信息，请参阅 [IAM 用户指南](#)。

Amazon Cognito 身份

借助 Amazon Cognito 身份，您可以使用自己的身份提供商或其他常见的身份提供商，例如，Login with Amazon、Facebook 或 Google。您需要用身份提供商提供的令牌来换取 AWS 安全凭证。这些凭证代表 IAM 角色，可与 AWS IoT 配合使用。

AWS IoT 可扩展 Amazon Cognito 并允许将策略附加到 Amazon Cognito 身份。您可以将策略附加到 Amazon Cognito 身份并为 AWS IoT 应用程序的单个用户提供精细权限。这样，您就可以在特定客户和他们的设备之间分配权限。有关更多信息，请参阅 [Amazon Cognito 身份](#)。



授权

策略确定经身份验证的身份可执行的操作。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。经身份验证的身份甚至可以是键入 AWS IoT CLI 命令的用户。仅当身份具有向其授予权限的策略时，才能执行 AWS IoT 操作。

AWS IoT 策略和 IAM 策略均可用于 AWS IoT，以控制身份（也称为委托人）可以执行的操作。您使用的策略类型取决于向 AWS IoT 进行身份验证时使用的身份类型。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 操作分为两组：

- 控制层面 API 允许您执行诸如创建或更新证书、事物、规则等管理任务。
- 数据层面 API 允许您向 AWS IoT 发送数据以及从其接收数据。

您使用的策略类型取决于您使用的是控制层面 API 还是数据层面 API。

AWS IoT 数据层面 API 和策略类型

协议和身份验证机制	开发工具包	身份类型	策略类型		
基于双向身份验证的 MQTT (端口 8883)	AWS IoT 设备软件开发工具包	X.509 证书	AWS IoT 策略		
基于 Websocket 的 MQTT (端口 443)	AWS 移动软件开发工具包	Amazon Cognito、IAM 或联合身份	用于 Amazon Cognito 身份的 AWS IoT 策略 用于其他身份的 IAM 策略		
基于服务器身份验证的 HTTP (端口 443)	AWS CLI	Amazon Cognito、IAM 或联合身份	用于 Amazon Cognito 身份的 AWS IoT 策略 用于其他身份的 IAM 策略		
基于双向身份验证的 HTTP (端口 8443)	不支持软件开发工具包	X.509 证书	AWS IoT 策略		

AWS IoT 控制层面 API 和策略类型

协议和身份验证机制	开发工具包	身份类型	策略类型		
基于服务器身份验证的 HTTP (端口 443)	AWS CLI	Amazon Cognito、IAM 或联合身份	用于 Amazon Cognito 身份的 AWS IoT 策略 用于其他身份的 IAM 策略		

AWS IoT 策略附加到 X.509 证书或 Amazon Cognito 身份。IAM 策略附加到 IAM 用户、组或角色。如果您使用 AWS IoT 控制台或 AWS IoT CLI 附加策略（附加到证书或 Amazon Cognito 身份），则应使用 AWS IoT 策略。否则，应使用 IAM 策略。

基于策略的授权是一种功能强大的工具。它让您能够完全控制设备、用户或应用程序在 AWS IoT 中可以执行的操作。例如，以使用证书连接到 AWS IoT 的设备为例。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。在另一个示例中，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 资源。还可以控制应用程序对 AWS IoT 资源的访问。

AWS IoT 策略

AWS IoT 策略是 JSON 文档。它们与 IAM 策略遵循相同的约定。AWS IoT 支持命名策略，因此，许多身份都可以参考相同的策略文档。命名策略采用版本化，以便可以轻松回滚。

AWS IoT 定义一组策略操作，这些操作描述可以授予或拒绝对它们的访问权限的操作和资源。例如：

- `iot:Connect` 代表连接到 AWS IoT 代理消息的权限。
- `iot:Subscribe` 代表订阅 MQTT 主题或主题筛选条件的权限。
- `iot:GetThingShadow` 代表获取事物影子的权限。

AWS IoT 策略允许您控制对 AWS IoT 数据层面的访问。AWS IoT 数据层面由允许您连接到 AWS IoT 消息代理，发送和接收 MQTT 消息以及获取或更新事物影子的操作组成。有关更多信息，请参阅 [AWS IoT 策略操作 \(p. 109\)](#)。

AWS IoT 策略是包含一个或多个策略语句的 JSON 文档。每个语句都包含一个 `Effect`、一个 `Action` 和一个 `Resource`。`Effect` 指定是否允许还是拒绝该操作。`Action` 指定策略允许或拒绝的操作。`Resource` 指定是否允许或拒绝对其执行操作的资源。以下策略向所有设备授予连接到 AWS IoT 消息代理的权限，但限制设备只能在特定 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/foo/bar"]  
        },  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Connect"],  
            "Resource": ["*"]  
        }  
    ]  
}
```

AWS IoT 策略操作

AWS IoT 定义了以下策略操作：

MQTT 策略操作

`iot:Connect`

代表连接到 AWS IoT 消息代理的权限。每次向代理发送 CONNECT 请求时，均检查 `iot: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。在第二个客户端建立连接后，代理将检测此案例并断开其中一个客户端的连接。`iot:Connect` 权限可用来确保只有经授权的客户端才能使用特定的客户端 ID 进行连接。

`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。这可用于允许客户端发布到特定主题模式。

Note

还必须授予 `iot:Connect` 权限才能授予 `iot:Publish` 权限。

iot:Receive

表示从 AWS IoT 接收消息的权限。每次向客户端交付消息时，均检查 `iot:Receive` 权限。由于每次交付时都会检查此权限，因此，可以使用它来撤销当前已订阅某个主题的客户端的权限。

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。这可用于允许客户端订阅与特定主题模式相符的主题。

Note

还必须授予 `iot:Connect` 权限才能授予 `iot:Subscribe` 权限。

Thing Shadow 策略操作

iot:DeleteThingShadow

代表删除事物影子的权限。每次请求删除事物影子文档时，均检查 `iot:DeleteThingShadow` 权限。

iot:GetThingShadow

代表检索事物影子的权限。每次请求检索事物影子文档时，均检查 `iot:GetThingShadow` 权限。

iot:UpdateThingShadow

代表更新事物影子的权限。每次请求更新事物影子文档的状态时，均检查 `iot:UpdateThingShadow` 权限。

操作资源

要为 AWS IoT 策略操作指定资源，必须使用资源的 ARN。所有资源 ARN 均采用以下形式：

`arn:aws:iot:<region>:<AWS account ID>:<resource type>:<resource name>`

下表列出了要为每种类型的操作指定的操作：

操作	资源
<code>iot:DeleteThingShadow</code>	事物 ARN - <code>arn:aws:iot:us-east-1:123456789012:thing/thingOne</code>
<code>iot:Connect</code>	客户端 ID ARN - <code>arn:aws:iot:us-east1:123456789012:client/myClientId</code>
<code>iot:Publish</code>	主题 ARN - <code>arn:aws:iot:us-east-1:123456789012:topic/myTopicName</code>
<code>iot:Subscribe</code>	主题筛选条件 ARN - <code>arn:aws:iot:us-east-1:123456789012:topicfilter/myTopicFilter</code>
<code>iot:Receive</code>	主题 ARN - <code>arn:aws:iot:us-east-1:123456789012:topic/myTopicName</code>
<code>iot:UpdateThingShadow</code>	事物 ARN - <code>arn:aws:iot:us-east-1:123456789012:thing/thingOne</code>
<code>iot:GetThingShadow</code>	事物 ARN - <code>arn:aws:iot:us-east-1:123456789012:thing/thingOne</code>

AWS IoT 策略变量

AWS IoT 定义可在资源或条件块内的 AWS IoT 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，如果设备使用客户端 ID“100-234-3456”连接到 AWS IoT 消息代理，`iot:ClientId` 策略变量将在策略文档中替换为“100-234-3456”。有关策略变量的更多信息，请参阅 [IAM 策略变量和多值条件](#)。

基本策略变量

AWS IoT 定义了以下基本策略变量：

- `iot:ClientId`：用于连接到 AWS IoT 消息代理的客户端 ID。
- `aws:SourceIp`：连接到 AWS IoT 消息代理的客户端的 IP 地址。

以下 AWS IoT 策略显示了策略变量的用法：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Connect"],  
            "Resource": [  
                "arn:aws:iot:us-east-1:123451234510:client/${iot:ClientId}"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": [  
                "arn:aws:iot:us-east-1:123451234510:topic/foo/bar/${iot:ClientId}"  
            ]  
        }  
    ]  
}
```

在这些示例中，当评估策略时， `${iot:ClientId}` 将被替换为连接到 AWS IoT 消息代理的客户端的 ID。当您使用诸如 `${iot:ClientId}` 之类的策略变量时，可能会无意中开放对非预期主题的访问。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/foo/${iot:ClientId}/bar"  
    ]  
}
```

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 `foo/+ /bar` 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 能够建立连接。例如，此策略仅允许客户端 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"  
            ]  
        }  
    ]  
}
```

```
    }]  
}
```

X.509 证书策略变量

X.509 证书策略变量允许您根据 X.509 证书属性编写授予权限的 AWS IoT 策略。以下部分介绍如何使用这些证书策略变量。

发布者属性

以下 AWS IoT 策略变量允许您根据由证书发布者设置的证书属性来授予或拒绝权限。

- `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`

主题属性

以下 AWS IoT 策略变量允许您根据由证书发布者设置的证书使用者属性来授予或拒绝权限。

- `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 证书允许这些属性包含一个或多个值。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。在策略中进行评估时，`iot:Certificate.Subject.Country` 由第一个国家/地区名称替换。您可以

使用从零开始的索引请求特定的属性值。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属性值（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。以下示例策略允许任何客户端连接至 AWS IoT，但是，对于证书的 `Certificate.Subject.Organization` 属性设为 "Example Corp" 或 "AnyCompany" 的客户端，会限制发布权限。这可以通过使用为前面的操作指定条件的 "Condition" 属性来完成。本例中的条件是证书的 `Certificate.Subject.Organization` 属性必须包括一个列出的值。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot:Connect"  
            ],  
            "Resource": [  
                "*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot:Publish"  
            ],  
            "Resource": [  
                "*"  
            ],  
            "Condition": {  
                "ForAllValues:StringEquals": {  
                    "iot:Certificate.Subject.Organization.List": [  
                        "Example Corp",  
                        "AnyCompany"  
                    ]  
                }  
            }  
        }  
    ]  
}
```

发布者备用名称属性

以下 AWS IoT 策略变量允许您根据由证书发布者设置的发布者备用名称属性来授予或拒绝权限。

- `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`

主题备用名称属性

以下 AWS IoT 策略变量允许您根据由证书发布者设置的使用者备用名称属性来授予或拒绝权限。

- `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`

其他属性

您可以根据证书的序列号使用 `iot:Certificate.SerialNumber` 允许或拒绝访问 AWS IoT 资源。`iot:Certificate.AvailableKeys` 策略变量包含具有值的所有证书策略变量的名称。

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：

通配符

如果证书属性中有通配符，则策略变量不会由证书属性值替换，在策略文档中保留 `${policy-variable}` 文本。这可能会导致授权失败。

数组字段

包含数组的证书属性限制为五项。多出的项目将被忽略。

字符串长度

所有字符串值的长度限制为 1024 个字符。如果证书属性包含超过 1024 个字符的字符串，则策略变量不会由证书属性值替换，在策略文档中保留 `${policy-variable}` 。这可能会导致授权失败。

事物策略变量

事物策略变量允许您根据事物名称、事物类型和事物属性值等事物属性来编写授予或拒绝权限的 AWS IoT 策略。事物名称从事物连接至 AWS IoT 时发送的 MQTT Connect 消息中的客户端 ID 获取。当事物通过使用 TLS 双向身份验证的 MQTT 连接至 AWS IoT 时，或通过使用经过身份验证的 Amazon Cognito 身份的 WebSocket 协议 MQTT 连接至 AWS IoT 时，事物策略变量将被替换。当证书或经过身份验证的 Amazon Cognito 身份附加到事物时，事物策略变量也将被替换。可以使用 [AttachThingPrincipal API](#) 将证书和经过身份验证的 Amazon Cognito 身份附加到事物。

可用的事物策略变量如下：

- `iot:Connection.Thing.ThingName`
- `iot:Connection.Thing.ThingTypeName`
- `iot:Connection.Thing.Attributes[attributeName]`
- `iot:Connection.Thing.IsAttached`

`iot:Connection.Thing.ThingName`

该属性等于为其评估策略的事物名称。事物名称设置为 MQTT/Websocket 连接的客户端 ID。此策略变量仅在通过 MQTT 或基于 WebSocket 协议的 MQTT 连接时可用。

`iot:Connection.Thing.ThingTypeName`

该属性等于与为其评估策略的事物相关联的事物类型。事物名称设置为 MQTT/Websocket 连接的客户端 ID。事物类型名称通过调用 `DescribeThing` API 获得。此策略变量仅在通过 MQTT 或基于 WebSocket 协议的 MQTT 连接时可用。

`iot:Connection.Thing.Attributes[attributeName]`

该属性等于与为其评估策略的事物相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`

为属性的名称。事物名称设置为 MQTT/Websocket 连接的客户端 ID。此策略变量仅在通过 MQTT 或基于 WebSocket 协议的 MQTT 连接时可用。

iot:Connection.Thing.IsAttached

如果正在为其评估策略的事物附加了证书或 Amazon Cognito 身份，这将解析为 `true`。

示例策略

AWS IoT 策略是在 JSON 文档中指定的。AWS IoT 策略的组成部分如下：

版本

必须设置为 "2012-10-17"。

效果

必须设置为 "Allow" 或 "Deny"。

操作

必须设置为 "iot:*operation-name*"，其中 *operation-name* 是以下各项之一：

`iot:Connect`：连接到 AWS IoT

`iot:Receive`：从 AWS IoT 接收消息

`"iot:Publish"`：MQTT 发布。

`"iot:Subscribe"`：MQTT 订阅。

`"iot:UpdateThingShadow"`：更新事物影子。

`"iot:GetThingShadow"`：检索事物影子。

`"iot>DeleteThingShadow"`：删除事物影子。

资源

必须设置为以下各项之一：

客户端 - `arn:aws:iot:region:account-id:client/client-id`

主题 ARN - `arn:aws:iot:region:account-id:topic/topic-name`

主题筛选条件 ARN - `arn:aws:iot:region:account-id:topicfilter/topic-filter`

连接策略示例

以下策略允许一组客户端 ID 建立连接：

```
{  
    "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:Subscribe",
        "iot:Receive"
    ],
    "Resource": [
        "*"
    ]
}
]
```

以下策略禁止一组客户端 ID 建立连接：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/clientid1",
                "arn:aws:iot:us-east-1:123456789012:client/clientid2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

以下策略允许证书持有者使用任何客户端 ID 订阅主题筛选条件 `foo/*`：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/foo/*"
            ]
        }
    ]
}
```

```
        "iot:Subscribe"
    ],
    "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/foo/*"
    ]
}
}
```

订阅/发布策略示例

您使用的策略取决于您连接到 AWS IoT 的方式。您可以使用 MQTT 客户端、HTTP 或 WebSocket 连接到 AWS IoT。通过 MQTT 客户端连接时，将使用 X.509 证书进行身份验证。通过 HTTP 或 WebSocket 协议连接时，将使用签名版本 4 和 Amazon Cognito 进行身份验证。

适用于 MQTT 客户端的策略

当在 AWS IoT 策略中为 MQTT 客户端指定主题筛选条件时，MQTT 通配符“+”和“#”将被视为文本字符。使用它们可能会导致意外行为。例如，以下策略将仅允许客户端订阅主题筛选条件 `foo/+/bar`：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/foo/+/bar"
            ]
        }
    ]
}
```

Note

MQTT 通配符“+”在策略中不被视为通配符。尝试订阅符合 `foo/+/bar` 模式（如 `foo/baz/bar` 或 `foo/goo/bar`）的主题筛选条件将以失败告终，并且会导致客户端断开连接。

您可以在策略的资源属性中使用“*”作为通配符。例如，以下策略允许证书持有者使用 AWS 账户在所有主题下发布消息以及订阅所有主题筛选条件：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:publish"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

```
        "/*"
    ]
}
}
```

以下策略允许证书持有者使用 AWS 账户在所有主题下发布消息：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Connect"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

您还可以在主题筛选条件的末尾使用“*”通配符。例如，以下策略允许证书持有者订阅符合 `foo/bar/*` 模式 的主题筛选条件：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/foo/bar/*"
      ]
    }
  ]
}
```

以下策略允许证书持有者在 `foo/bar` 和 `foo/baz` 主题下发布消息：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/foo/bar",
        "arn:aws:iot:us-east-1:123456789012:topicfilter/foo/baz"
      ]
    }
  ]
}
```

```
        "*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "iot:Publish"
    ],
    "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/foo/bar",
        "arn:aws:iot:us-east-1:123456789012:topic/foo/baz"
    ]
}
]
```

以下策略禁止证书持有者在 foo/bar 主题下发布消息：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Deny",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/foo/bar"
            ]
        }
    ]
}
```

以下策略允许证书持有者在主题 foo 下发布消息，但禁止证书持有者在主题 bar 下发布消息：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/foo"
            ]
        }
    ]
}
```

```
        ],
    },
    {
        "Effect": "Deny",
        "Action": [
            "iot:Publish"
        ],
        "Resource": [
            "arn:aws:iot:us-east-1:123456789012:topic/bar"
        ]
    }
]
```

以下策略允许证书持有者订阅主题筛选条件 foo/bar：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/foo/bar"
            ]
        }
    ]
}
```

以下策略允许证书持有者在 arn:aws:iot:us-east-1:123456789012:topic/iotmonitor/provisioning/8050373158915119971 主题下发布消息以及订阅主题筛选条件 arn:aws:iot:us-east-1:123456789012:topicfilter/iotmonitor/provisioning/8050373158915119971：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [

```

```
        "arn:aws:iot:us-east-1:123456789012:topic/iotmonitor/
provisioning/8050373158915119971"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "iot:Subscribe"
    ],
    "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/iotmonitor/
provisioning/8050373158915119971"
    ]
}
]
```

适用于 HTTP 和 WebSocket 客户端的策略

对于以下操作，AWS IoT 使用附加到 Amazon Cognito 身份（通过 `AttachPrincipalPolicy` API）的 AWS IoT 策略缩小附加到 Amazon Cognito 身份池（由经过身份验证的身份组成）的权限范围。这意味着，Amazon Cognito 身份需要从附加到池的 IAM 角色策略和通过 AWS IoT `AttachPrincipalPolicy` API 附加到 Amazon Cognito 身份的 AWS IoT 策略获取权限。

- `iot:Connect`
- `iot:Publish`
- `iot:Subscribe`
- `iot:Receive`
- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Note

对于其他 AWS IoT 操作或未经身份验证的身份，AWS IoT 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。

要允许未经过身份验证的 Amazon Cognito 身份通过 HTTP 向任何主题发布消息，请将以下策略附加到 Amazon Cognito 身份池角色：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect",
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": ["*"]
        }
    ]
}
```

要允许未经过身份验证的 Amazon Cognito 身份通过 HTTP 向您账户中的任何主题发布 MQTT 消息，请将以下策略附加到 Amazon Cognito 身份池角色：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": ["*"]  
        }  
    ]  
}
```

Note

此示例仅用于说明。除非您的服务确实要求，否则，我们建议您使用较严格的策略，即，禁止未经过身份验证的 Amazon Cognito 身份在任何主题下发布消息的策略。

要允许未经过身份验证的 Amazon Cognito 身份通过 HTTP 在账户中的 topic1 下发布 MQTT 消息，请将以下策略附加到 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/topic1"]  
        }  
    ]  
}
```

要使经过身份验证的 Amazon Cognito 身份能够通过 HTTP 在 AWS 账户内的 topic1 下发布 MQTT 消息，您必须指定此处列出的两项策略。第一个策略必须附加到 Amazon Cognito 身份池角色。它允许该池中的身份进行发布调用。第二个策略必须使用 AWS IoT [AttachPrincipalPolicy](#) API 附加到 Amazon Cognito 用户。它允许指定的 Amazon Cognito 用户访问 topic1 主题。

Amazon Cognito 身份池策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/topic1"]  
        }  
    ]  
}
```

Amazon Cognito 用户策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/topic1"]  
        }  
    ]  
}
```

同样，以下示例策略允许 Amazon Cognito 用户通过 HTTP 在 topic1 和 topic2 主题下发布 MQTT 消息。需要两项策略。第一项策略允许 Amazon Cognito 身份池角色进行发布调用。第二项策略允许 Amazon Cognito 用户访问 topic1 和 topic2 主题。

Amazon Cognito 身份池策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": ["*"]  
        }  
    ]  
}
```

Amazon Cognito 用户策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": [  
                "arn:aws:iot:us-east-1:123456789012:topic/topic1",  
                "arn:aws:iot:us-east-1:123456789012:topic/topic2"  
            ]  
        }  
    ]  
}
```

以下策略允许多个 Amazon Cognito 用户向一个主题发布消息。每个 Amazon Cognito 身份需要两个策略。第一项策略允许 Amazon Cognito 身份池角色进行发布调用。第二个和第三个策略分别允许 Amazon Cognito 用户访问主题 topic1 和 topic2。

Amazon Cognito 身份池策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": ["*"]  
        }  
    ]  
}
```

Amazon Cognito user1 策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/topic1"]  
        }  
    ]  
}
```

Amazon Cognito user2 策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/topic2"]  
        }  
    ]  
}
```

```
    }]
}
```

接收策略示例

以下策略禁止证书持有者使用任何客户端 ID 接收来自某个主题的消息：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/foo/restricted"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:)"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

以下策略允许证书持有者使用任何客户端 ID 订阅并接收一个主题中的消息：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect"
      ],
      "Resource": [ * ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topicfilter/foo/bar"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/foo/bar"
      ]
    }
  ]
}
```

证书策略示例

以下策略允许设备在名称与设备用于自我身份验证的证书的 `certificateId` 相同的主题上发布：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/${iot:CertificateId}"]  
        },  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Connect"],  
            "Resource": ["*"]  
        }  
    ]  
}
```

以下策略允许设备在主题上发布，其中主题名称与设备用于自我身份验证的证书的主题常用名字段相同：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/  
${iot:Certificate.Issuer.CommonName}"]  
        },  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Connect"],  
            "Resource": ["*"]  
        }  
    ]  
}
```

当用于对设备进行身份验证的证书的 `Subject.CommonName.2` 字段设置为“Administrator”时，以下策略允许设备在前缀为“admin/”的主题上发布：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Connect"],  
            "Resource": ["*"]  
        },  
        {  
            "Effect": "Allow",  
            "Action": ["iot:Publish"],  
            "Resource": ["arn:aws:iot:us-east-1:123456789012:topic/admin/*"],  
            "Condition": {  
                "StringEquals": {  
                    "iot:Certificate.Subject.CommonName.2": "Administrator"  
                }  
            }  
        }  
    ]  
}
```

当用于对设备进行身份验证的证书的任意一个 `Subject.Common` 字段设置为“Administrator”时，以下策略允许设备在前缀为“admin/”的主题上发布：

```
{
```

```
"Version": "2012-10-17",
"Statement": [
    {
        "Effect": "Allow",
        "Action": ["iot:Connect"],
        "Resource": ["*"]
    },
    {
        "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

以下策略允许设备在包含事物类型名称和事物名称的特定主题上发布：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["iot:Publish"],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingType} / ${iot:Connection.Thing.ThingName}"
            ]
        }
    ]
}
```

如果用于向 AWS IoT 进行身份验证的证书被附加到正在为其评估策略的事物上，则以下策略允许设备连接。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["iot:Connect"],
            "Resource": ["*"],
            "Condition": {
                "Bool": {
                    "iot:Connection.Thing.IsAttached": ["true"]
                }
            }
        }
    ]
}
```

以下策略允许设备在一组主题上发布（“/foo/bar”和“/foo/baz”），前提是：

- 与设备关联的事物包含值为“foo”、“bar”或“baz”的“Manufacturer”属性。
- 与设备关联的事物位于 Thing Registry 中，并附加到用于连接 AWS IoT 的证书。

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

```

```
"Action": ["iot:Publish"],
"Resource": [
    "arn:aws:iot:us-east-1:123456789012:topic/foo/bar",
    "arn:aws:iot:us-east-1:123456789012:topic/foo/baz"
],
"Condition": {
    "ForAnyValue:StringLike": {
        "iot:Connection.Thing.Attributes[Manufacturer)": [
            "foo",
            "bar",
            "baz"
        ]
    }
}
}
```

以下策略允许设备发布主题，前提是：

- 主题由事物类型名称、“/”和事物名称组成。
- 事物位于 Thing Registry 中。
- 事物附加到用于连接 AWS IoT 的证书。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["iot:Publish"],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/
${iot:Connection.Thing.ThingTypeNames}/${iot:Connection.Thing.ThingName}"
            ]
        }
    ]
}
```

如果事物位于 Thing Registry 中，则以下策略允许设备仅在其自己的事物影子主题上发布。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["iot:Publish"],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/
${iot:Connection.Thing.ThingName}/shadow/update"
            ]
        }
    ]
}
```

IAM IoT 策略

AWS Identity and Access Management 为 AWS IoT 定义的每种操作定义一种策略操作，包括控制层面 API 和数据层面 API。

AWS IoT API 权限

下表列出了 AWS IoT API、所需的 IAM 权限以及 API 操纵的资源。

API	必需权限 (策略操作)	资源
AcceptCertificateTransfer	iot:AcceptCertificateTransfer	arn:aws:iot: region:account-id:cert/cert-id
		Note ARN 中指定的 AWS 账户必须是证书将传输到的账户。
AttachPrincipalPolicy	iot:AttachPrincipalPolicy	arn:aws:iot: region:account-id:cert/cert-id
AttachThingPrincipal	iot:AttachThingPrincipal	arn:aws:iot: region:account-id:cert/cert-id
CancelCertificateTransfer	iot:CancelCertificateTransfer	arn:aws:iot: region:account-id:cert/cert-id
		Note ARN 中指定的 AWS 账户必须是证书将传输到的账户。
CreateCertificateFromCsr	iot>CreateCertificateFromCsr	
CreateKeysAndCertificate	iot>CreateKeysAndCertificate	
CreatePolicy	iot:CreatePolicy	*
CreatePolicyVersion	iot:CreatePolicyVersion	arn:aws:iot: region:account-id:policy/policy-name
		Note 这必须是 AWS IoT 策略，而不是 IAM 策略。
CreateThing	iot:CreateThing	arn:aws:iot: region:account-id:thing/thing-name
CreateThingType	iot:CreateThingType	arn:aws:iot: region:account-id:thingtype/thing-type-name
CreateTopicRule	iot:CreateTopicRule	arn:aws:iot: region:account-id:rule/rule-name
DeleteCACertificate	iot:DeleteCACertificate	arn:aws:iot: region:account-id:cacert/cert-id
DeleteCertificate	iot:DeleteCertificate	arn:aws:iot: region:account-id:cert/cert-id
DeletePolicy	iot:DeletePolicy	arn:aws:iot: region:account-id:policy/policy-name
DeletePolicyVersion	iot:DeletePolicyVersion	arn:aws:iot: region:account-id:policy/policy-name
DeleteRegistrationCode	iot:DeleteRegistrationCode	
DeleteThing	iot:DeleteThing	arn:aws:iot: region:account-id:thing/thing-name
DeleteThingType	iot:DeleteThingType	arn:aws:iot: region:account-id:thingtype/thing-type-name
DeleteTopicRule	iot:DeleteTopicRule	arn:aws:iot: region:account-id:rule/rule-name
DeprecateThingType	iot:DeprecateThingType	arn:aws:iot: region:account-id:thingtype/thing-type-name
DescribeCaCertificate	iot:DescribeCaCertificate	arn:aws:iot: region:account-id:cacert/cert-id
DescribeCertificate	iot:DescribeCertificate	arn:aws:iot: region:account-id:cert/cert-id

API	必需权限 (策略操作)	资源
DescribeEndpoint	iot:DescribeEndpoint *	
DescribeThing	iot:DescribeThing	arn:aws:iot: <i>region:account-id:thing/thing-name</i>
DescribeThingType	iot:DescribeThingType	arn:aws:iot: <i>region:account-id:thingtype/thing-type-name</i>
DetachPrincipalPolicy	iot:DetachPrincipalPolicy	arn:aws:iot: <i>region:account-id:cert/cert-id</i>
DetachThingPrincipal	iot:DetachThingPrincipal	arn:aws:iot: <i>region:account-id:cert/cert-id</i>
DisableTopicRule	iot:DisableTopicRule	arn:aws:iot: <i>region:account-id:rule/rule-name</i>
EnableTopicRule	iot:EnableTopicRule	arn:aws:iot: <i>region:account-id:rule/rule-name</i>
GetLoggingOptions	iot:GetLoggingOptions\$	
GetPolicy	iot:GetPolicy	arn:aws:iot: <i>region:account-id:policy/policy-name</i>
GetPolicyVersion	iot:GetPolicyVersion	arn:aws:iot: <i>region:account-id:policy/policy-name</i>
GetRegistrationCode	iot:GetRegistrationCode	
GetTopicRule	iot:GetTopicRule	arn:aws:iot: <i>region:account-id:rule/rule-name</i>
ListCaCertificates	iot>ListCaCertificates	*
ListCertificates	iot>ListCertificates	*
iot>ListCertificatesByCa	iot>ListCertificatesByCa	
ListOutgoingCertificates	iot>ListOutgoingCertificates	
ListPolicies	iot>ListPolicies	*
ListPolicyPrincipals	iot>ListPolicyPrincipals	策略的 ARN : arn:aws:iot: <i>region:account-id:policy/policy-name</i>
ListPolicyVersions	iot>ListPolicyVersions	策略的 ARN : arn:aws:iot: <i>region:account-id:policy/policy-name</i>
ListPrincipalPolicies	iot>ListPrincipalPolicies	证书的 ARN : arn:aws:iot: <i>region:account-id:cert/cert-id</i>
ListPrincipalThings	iot>ListPrincipalThings	证书的 ARN : arn:aws:iot: <i>region:account-id:cert/cert-id</i>
ListThingPrincipals	iot>ListThingPrincipals	AWS IoT 事物的 ARN : arn:aws:iot: <i>region:account-id:thing/thing-name</i>
ListThings	iot>ListThings	*
ListThingTypes	iot>ListThingTypes	*
ListTopicRules	iot>ListTopicRules	*
RegisterCACertificate	iot:RegisterCACertificate	
RegisterCertificate	iot:RegisterCertificate*	

API	必需权限 (策略操作)	资源
RejectCertificateTransfer	iot:RejectCertificateTransfer	arn:aws:iot: <i>region:account-id:cert/cert-id</i>
ReplaceTopicRule	iot:ReplaceTopicRule	arn:aws:iot: <i>region:account-id:rule/rule-name</i>
SetDefaultPolicyVersion	iot:SetDefaultPolicyVersion	arn:aws:iot: <i>region:account-id:policy/policy-name</i>
SetLoggingOptions	iot:SetLoggingOptions	arn:aws:iot: <i>region:account-id:role/role-name</i>
TransferCertificate	iot:TransferCertificate	arn:aws:iot: <i>region:account-id:cert/cert-id</i>
UpdateCACertificate	iot:UpdateCACertificate	arn:aws:iot: <i>region:account-id:cacert/cert-id</i>
UpdateCertificate	iot:UpdateCertificate	arn:aws:iot: <i>region:account-id:cert/cert-id</i>
UpdateThing	iot:UpdateThing	arn:aws:iot: <i>region:account-id:thing/thing-name</i>

IAM 策略模板

AWS IoT 提供了一组 IAM 策略模板，您可以按原样使用，也可以将其作为起点来创建自定义的 IAM 策略。利用这些模板，您可以访问配置和数据操作。利用配置操作，您可以创建事物、证书、策略和规则。数据操作通过 MQTT 或 HTTP 协议发送数据。下表对这些模板进行了说明。

策略模板	描述
AWSIoTLogging	允许相关身份配置 CloudWatch 日志记录。本策略附加到您的 CloudWatch 日志记录角色。
AWSIoTConfigAccess	允许相关身份访问所有 AWS IoT 配置操作。
AWSIoTConfigReadOnlyAccess	允许相关身份调用只读配置操作。
AWSIoTDataAccess	允许相关身份全面访问所有 AWS IoT 数据操作。数据操作通过 MQTT 或 HTTP 协议发送数据。
AWSIoTFullAccess	允许相关身份全面访问所有 AWS IoT 配置和数据操作。
AWSIoTRuleActions	允许相关身份访问 AWS IoT 规则操作中所有受支持的 AWS 服务。

跨账户访问

AWS IoT 允许您支持委托人订阅并不属于委托人的 AWS 账户中定义的主题或向该主题发布消息。您可以通过创建 IAM 策略和 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 IAM 策略，就像您在 AWS 账户中为其他用户和证书创建策略一样。例如，以下策略允许连接到 /foo/bar 主题并向该主题发布消息。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Topic": "/foo/bar"
    }
  ]
}
```

```
"Action": [
    "iot:Connect"
],
"Resource": [
    "*"
]
},
{
    "Effect": "Allow",
    "Action": [
        "iot:Publish"
    ],
    "Resource": [
        "arn:aws:iot:us-east-1:123456789012:topic/foo/bar"
    ]
}
]
```

接下来，请执行为 [IAM 用户创建角色](#) 中的步骤。输入您希望其共享访问权限的 AWS 账户的 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 账户 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-1:111111111111:user/MyUser"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

传输安全

AWS IoT 消息代理和 Thing Shadows 服务可将与 TLS 进行的所有通信加密。TLS 用于确保受 AWS IoT 支持的应用程序协议 (MQTT、HTTP) 的保密性。TLS 适用于许多编程语言和操作系统。

对于 MQTT，TLS 可将设备与代理之间的连接加密。AWS IoT 使用 TLS 客户端身份验证来识别设备。对于 HTTP，TLS 可将设备与代理之间的连接加密。身份验证工作委派给 AWS Signature 版本 4 执行。

TLS 密码包支持

AWS IoT 支持以下密码包：

- ECDHE-ECDSA-AES128-GCM-SHA256 (推荐)
- ECDHE-RSA-AES128-GCM-SHA256 (推荐)
- ECDHE-ECDSA-AES128-SHA256
- ECDHE-RSA-AES128-SHA256
- ECDHE-ECDSA-AES128-SHA
- ECDHE-RSA-AES128-SHA
- ECDHE-ECDSA-AES256- GCM-SHA384
- ECDHE-RSA-AES256- GCM-SHA384
- ECDHE-ECDSA-AES256-SHA384
- ECDHE-RSA-AES256-SHA384

- ECDHE-RSA-AES256-SHA
- ECDHE-ECDSA-AES256-SHA
- AES128-GCM-SHA256
- AES128-SHA256
- AES128-SHA
- AES256-GCM-SHA384
- AES256-SHA256
- AES256-SHA

AWS IoT 的消息代理

AWS IoT 消息代理是一项发布/订阅代理服务，可与 AWS IoT 相互发送和接收消息。在与 AWS IoT 通信时，客户端将经过编址的消息发送到 Sensor/temp/room1 之类的主题。进而，消息代理将消息发送到已注册接收该主题消息的所有客户端。发送消息的操作被称为发布。已注册接收该主题筛选消息的操作被称为订阅。

每个 AWS 账户和区域对的主题命名空间均是独立的。例如，AWS 账户的 Sensor/temp/room1 主题独立于另一 AWS 账户的 Sensor/temp/room1 主题。在区域方面，情况同样如此。us-east-1 内同一 AWS 账户中的 Sensor/temp/room1 主题独立于 us-east-2 内的同一主题。AWS IoT 不支持跨 AWS 账户和区域发送和接收消息。

消息代理负责维护由所有客户端会话及每个会话的订阅组成的列表。向主题发布消息时，代理将检查订阅映射到该主题的会话。然后，代理将发布消息转发到目前连接到客户端的所有会话。

协议

消息代理支持使用 MQTT 协议进行发布和订阅，并支持使用 HTTPS 协议进行发布。两个协议均通过 IP 版本 4 和 IP 版本 6 受支持。消息代理还支持基于 WebSocket 的 MQTT 协议。

协议/端口映射

下表显示 AWS IoT 支持的每个协议、身份验证方法和每个协议使用的端口。

协议、身份验证及端口映射

协议	身份验证	Port
MQTT	客户端证书	8883
HTTP	客户端证书	8443
HTTP	SigV4	443
MQTT + WebSocket	SigV4	443

MQTT

MQTT 是专为受限设备设计的、广泛应用的轻型消息处理协议。有关更多信息，请参阅 [MQTT](#)。

尽管 AWS IoT 消息代理的实施基于 MQTT 3.1.1 版，但却与规范存在如下偏差：

- 在 AWS IoT 中，利用服务质量 (QoS) 0 订阅主题意味着消息将发送零次或多次。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。
- AWS IoT 不支持利用 QoS 2 进行发布和订阅。请求 QoS 2 时，AWS IoT 消息代理不会发送 PUBACK 或 SUBACK。
- 用于发布和订阅主题的各 QoS 级别之间毫无关系。一个客户端可使用 QoS1 订阅主题，而另一个客户端可使用 QoS0 向同一主题发布消息。
- 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。如果两个 MQTT 客户端同时连接到同一客户端 ID，该标志的值可能会不正确。
- 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。
- MQTT 规范提供了相应的配置，以供发布者用于请求代理将发送到主题的最新消息保留下来并发送给未来的所有主题订阅者。AWS IoT 不支持保留的消息。如果请求保留消息，则将断开连接。
- 消息代理使用客户端 ID 标识每个客户。客户端 ID 作为 MQTT 有效负载的一部分从客户端传递到消息代理。两个具有相同客户端 ID 的客户端不得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，系统将向两个客户端发送 CONNACK 消息，当前连接的客户端也将断开连接。
- 消息代理不支持持久性会话（将 cleanSession 标记设置为 false 进行的连接）。AWS IoT 消息代理假设所有会话均为干净会话，并且未在这些会话中存储消息。如果 MQTT 客户端尝试在 cleanSession 设置为 false 的情况下连接到 AWS IoT 消息代理，则客户端将断开连接。
- 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。
- 消息代理并不保证收到消息和 ACK 的顺序。

HTTP

消息代理支持客户端通过 REST API 使用 HTTP 协议进行连接。客户端通过将 POST 消息发送到 [`<AWS IOT Endpoint>/topics/<url_encoded_topic_name>?qos=1`](#) 进行发布。

例如，您可以使用 curl 模拟按钮按压。如果您按照 [AWS IoT 入门 \(p. 4\)](#) 中的教程操作，而不是像在 [AWS IoT MQTT 客户端 \(p. 28\)](#) 中一样使用 AWS IoT MQTT 客户端发布消息，则请使用类似如下的命令：

```
curl --tlsv1.2 --cacert root-CA.crt --cert 4b7828d2e5-certificate.pem.crt --key 4b7828d2e5-private.pem.key -X POST -d "{ \"serialNumber\": \"G030JF053216F1BS\", \"clickType\": \"SINGLE\", \"batteryVoltage\": \"2000mV\" }" "https://alpn10j0v8htvw.iot.us-east-1.amazonaws.com:8443/topics/iotbutton/virtualButton?qos=1"
```

--tlsv1.2

使用 TLSv1.2 (SSL)。必须随 OpenSSL 安装 curl，并且务必使用 TLS 1.2 版。

--cacert <filename>

用于验证对等项的 CA 证书文件名。

--cert <filename>

客户端证书文件名。

--key <filename>

私有密钥文件名。

-X POST

请求的类型，此处为 POST。

-d <data>

要发布的 HTTP POST 数据。在这种情况下，我们会模拟通过一次按钮按压发送的数据。
"https://..."

URL。此处指的是事物的 REST API 终端节点。(要查找事物的终端节点，请从 AWS IoT 控制台选择 Registry 来展开选项。选择 Things，选择事物，然后选择 Interact。)在终端节点后添加端口 (:8443)，后跟主题；最后，在查询字符串 (?qos=1) 中指定服务质量。

基于 WebSocket 的 MQTT 协议

AWS IoT 支持基于 [WebSocket](#) 的 MQTT 协议，从而使基于浏览器的远程应用程序能够通过连接到 AWS IoT 的设备使用 AWS 凭证发送和接收消息。指定 AWS 凭证时使用的是 [AWS 签名版本 4](#)。WebSocket 支持服务适用于 TCP 端口 443，因此，消息可以穿过大多数防火墙和 Web 代理。

通过发送 HTTP GET 请求在客户端上启动 WebSocket 连接。您使用的 URL 应采用以下格式：

```
wss://<endpoint>.iot.<region>.amazonaws.com/mqtt
```

wss

指定 WebSocket 协议。

终端节点

特定于您的 AWS 账户的 AWS IoT 终端节点。您可以使用 AWS IoT CLI [describe-endpoint](#) 命令找到该终端节点。

region

您的 AWS 账户所在的 AWS 区域。

mqtt

指定您将使用 WebSocket 协议发送 MQTT 消息。

当服务器进行响应时，客户端将发送升级请求，向服务器表明它将使用 WebSocket 协议进行通信。在服务器确认升级请求后，将使用 WebSocket 协议执行所有通信。您使用的 WebSocket 实施将充当传输协议。您使用 WebSocket 协议发送的数据是 MQTT 消息。

在 Web 应用程序中使用 WebSocket 协议

大多数 Web 浏览器提供的 WebSocket 实施不允许修改 HTTP 标头，因此，您必须在查询字符串中添加签名版本 4 信息。有关更多信息，请参阅[将签名信息添加到查询字符串](#)。

以下 JavaScript 将定义在生成签名版本 4 请求时使用的一些使用函数。

```
/**  
 * utilities to do sigv4  
 * @class SigV4Utils  
 */  
function SigV4Utils() {}  
  
SigV4Utils.getSignatureKey = function (key, date, region, service) {  
    var kDate = AWS.util.crypto.hmac('AWS4' + key, date, 'buffer');  
    var kRegion = AWS.util.crypto.hmac(kDate, region, 'buffer');  
    var kService = AWS.util.crypto.hmac(kRegion, service, 'buffer');  
    var kCredentials = AWS.util.crypto.hmac(kService, 'aws4_request', 'buffer');  
    return kCredentials;  
}
```

```
};

SigV4Utils.getSignedUrl = function(host, region, credentials) {
    var datetime = AWS.util.date.iso8601(new Date()).replace(/[:\ -]|\.\\d{3}/g, '');
    var date = datetime.substr(0, 8);

    var method = 'GET';
    var protocol = 'wss';
    var uri = '/mqtt';
    var service = 'iotdevicegateway';
    var algorithm = 'AWS4-HMAC-SHA256';

    var credentialScope = date + '/' + region + '/' + service + '/' + 'aws4_request';
    var canonicalQuerystring = 'X-Amz-Algorithm=' + algorithm;
    canonicalQuerystring += '&X-Amz-Credential=' +
    encodeURIComponent(credentials.accessKeyId + '/' + credentialScope);
    canonicalQuerystring += '&X-Amz-Date=' + datetime;
    canonicalQuerystring += '&X-Amz-SignedHeaders=host';

    var canonicalHeaders = 'host:' + host + '\n';
    var payloadHash = AWS.util.crypto.sha256('', 'hex')
    var canonicalRequest = method + '\n' + uri + '\n' + canonicalQuerystring + '\n' +
    canonicalHeaders + '\nhost\n' + payloadHash;

    var stringToSign = algorithm + '\n' + datetime + '\n' + credentialScope + '\n' +
    AWS.util.crypto.sha256(canonicalRequest, 'hex');
    var signingKey = SigV4Utils.getSignatureKey(credentials.secretAccessKey, date, region,
    service);
    var signature = AWS.util.crypto.hmac(signingKey, stringToSign, 'hex');

    canonicalQuerystring += '&X-Amz-Signature=' + signature;
    if (credentials.sessionToken) {
        canonicalQuerystring += '&X-Amz-Security-Token=' +
    encodeURIComponent(credentials.sessionToken);
    }

    var requestUrl = protocol + '://' + host + uri + '?' + canonicalQuerystring;
    return requestUrl;
};
```

创建签名版本 4 请求

1. 创建规范的签名版本 4 请求。

以下 JavaScript 代码将创建规范请求：

```
var datetime = AWS.util.date.iso8601(new Date()).replace(/[:\ -]|\.\\d{3}/g, '');
var date = datetime.substr(0, 8);

var method = 'GET';
var protocol = 'wss';
var uri = '/mqtt';
var service = 'iotdevicegateway';
var algorithm = 'AWS4-HMAC-SHA256';

var credentialScope = date + '/' + region + '/' + service + '/' + 'aws4_request';
var canonicalQuerystring = 'X-Amz-Algorithm=' + algorithm;
canonicalQuerystring += '&X-Amz-Credential=' +
    encodeURIComponent(credentials.accessKeyId + '/' + credentialScope);
canonicalQuerystring += '&X-Amz-Date=' + datetime;
canonicalQuerystring += '&X-Amz-SignedHeaders=host';

var canonicalHeaders = 'host:' + host + '\n';
```

```
var payloadHash = AWS.util.crypto.sha256('', 'hex')
var canonicalRequest = method + '\n' + uri + '\n' + canonicalQueryString + '\n' +
canonicalHeaders + '\nhost\n' + payloadHash;
```

2. 创建要签名的字符串，生成签名密钥，然后为该字符串签名。

采用在上一步中创建的规范 URL 并将其组合到待签名的字符串中。为此，请创建由哈希算法、日期、凭证范围以及规范请求的 SHA 组成的字符串。然后，生成签名密钥并为该字符串签名，如以下 JavaScript 代码所示。

```
var stringToSign = algorithm + '\n' + datetime + '\n' + credentialScope + '\n' +
AWS.util.crypto.sha256(canonicalRequest, 'hex');
var signingKey = SigV4Utils.getSignatureKey(credentials.secretAccessKey, date, region,
service);
var signature = AWS.util.crypto.hmac(signingKey, stringToSign, 'hex');
```

3. 将签名信息添加到请求中。

以下 JavaScript 代码表明了如何将签名信息添加到查询字符串中。

```
canonicalQueryString += '&X-Amz-Signature=' + signature;
```

4. 如果您具有会话凭证（来自 STS 服务器、AssumeRole 或 Amazon Cognito），请在签名后将会话令牌附加到 URL 的末尾：

```
canonicalQueryString += '&X-Amz-Security-Token=' +
encodeURIComponent(credentials.sessionToken);
```

5. 在规范查询字符串前面加上协议、主机和 URL：

```
var requestUrl = protocol + '://' + host + uri + '?' + canonicalQueryString;
```

6. 打开 WebSocket。

以下 JavaScript 代码将表明如何创建 Paho MQTT 客户端并将 CONNECT 调用到 AWS IoT 中。`endpoint` 参数是特定于您的 AWS 账户的终端节点。`clientId` 是您的 AWS 账户同时连接的所有客户端中的唯一文本标识符。

```
var client = new Paho.MQTT.Client(requestUrl, clientId);
var connectOptions = {
    onSuccess: function(){
        // connect succeeded
    },
    useSSL: true,
    timeout: 3,
    mqttVersion: 4,
    onFailure: function() {
        // connect failed
    }
};
client.connect(connectOptions);
```

在移动应用程序中使用 WebSocket 协议

我们建议在建立 WebSocket 连接时使用其中一个 AWS IoT Device SDK 将您的设备连接到 AWS IoT。以下 AWS IoT 设备软件开发工具包支持基于 WebSocket 的 MQTT 连接到 AWS IoT：

- Node.js
- iOS
- Android

有关使用基于 WebSocket 的 MQTT 协议将 Web 应用程序连接到 AWS IoT 的参考实施案例，请参阅 [AWS WebSocket 实验室示例](#)。

如果您使用的是当前不受支持的编程或脚本语言，则只要使用 AWS 签名版本 4 为初始 WebSocket 升级请求 (HTTP POST) 签名，便可以使用任何现有的 WebSocket 库。有些 MQTT 客户端 (如 [Eclipse Paho for JavaScript](#)) 可为 WebSocket 协议提供本机支持。

主题：

消息代理使用主题将消息从发布客户端路由到订阅客户端。正斜杠 (/) 用于分隔主题层次结构。下表列出了订阅时可在主题筛选条件中使用的通配符。

主题通配符

通配符	描述
#	必须是您要订阅的主题中的最后一个字符。通过将当前树与所有子树相匹配来发挥通配符的作用。例如，Sensor/# 订阅将接收发布到 Sensor/、Sensor/temp 和 Sensor/temp/room1 的消息，但不会接收发布到 Sensor 的消息。
+	精确匹配主题层次结构中的一个项目。例如，Sensor/+/room1 订阅将接收发布到 Sensor/temp/room1、Sensor/moisture/room1 等的消息。

预留的主题

任何以 \$ 开头的主题都被视为预留的主题，除非下面列出的主题之外，不支持用于进行发布和订阅。在任何其他以 \$ 开头的主题中尝试发布或订阅将导致连接被终止。

主题	允许的操作	描述
\$aws/events/presence/connected/ <i>clientId</i>	Subscribe	当使用特定客户端 ID 的 MQTT 客户端连接至 AWS IoT 时，AWS IoT 将向此主题发布。有关更多信息，请参阅 连接/断开连接事件 (p. 141) 。
\$aws/events/presence/disconnected/ <i>clientId</i>	Subscribe	当使用特定客户端 ID 的 MQTT 客户端与 AWS IoT 断开连接时，AWS IoT 将向此主题发

主题	允许的操作	描述
		布。有关更多信息，请参阅 连接/断开连接事件 (p. 141) 。
\$aws/events/subscriptions/subscribed/ <i>clientId</i>	Subscribe	当使用特定客户端 ID 的 MQTT 客户端订阅 MQTT 主题时，AWS IoT 将向此主题发布。有关更多信息，请参阅 订阅/取消订阅事件 (p. 142) 。
\$aws/events/subscriptions/unsubscribed/ <i>clientId</i>	Subscribe	当使用特定客户端 ID 的 MQTT 客户端取消订阅 MQTT 主题时，AWS IoT 将向此主题发布。有关更多信息，请参阅 订阅/取消订阅事件 (p. 142) 。
\$aws/things/ <i>thingName</i> /shadow/delete	发布/订阅	事物或应用程序向此主题发布来删除事物影子。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html#delete-pub-sub-topic 。
\$aws/things/ <i>thingName</i> /shadow/delete/accepted	Subscribe	当一个事物影子被删除时，Thing Shadows 服务将向该主题发送消息。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html#delete-accepted-pub-sub-topic 。
\$aws/things/ <i>thingName</i> /shadow/delete/rejected	Subscribe	当删除事物影子的请求遭拒时，Thing Shadows 服务将向该主题发送消息。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html#delete-rejected-pub-sub-topic 。
\$aws/things/ <i>thingName</i> /shadow/get	发布/订阅	应用程序或事物向此主题发布空消息来获取事物影子。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html 。
\$aws/things/ <i>thingName</i> /shadow/get/accepted	Subscribe	当获取事物影子的请求获批时，Thing Shadows 服务将向该主题发送消息。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html#get-accepted-pub-sub-topic 。

主题	允许的操作	描述
\$aws/things/ <i>thingName</i> /shadow/get/rejected	Subscribe	当获取事物影子的请求遭拒时，Thing Shadows 服务将向该主题发送消息。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html#get-rejected-pub-sub-topic 。
\$aws/things/ <i>thingName</i> /shadow/update	发布/订阅	事物或应用程序向此主题发布来更新事物影子。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html#update-pub-sub-topic 。
\$aws/things/ <i>thingName</i> /shadow/update/accepted	Subscribe	当事物影子更新成功时，Thing Shadows 服务将向该主题发送消息。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html#update-accepted-pub-sub-topic 。
\$aws/things/ <i>thingName</i> /shadow/update/rejected	Subscribe	当事物影子更新遭拒时，Thing Shadows 服务将向该主题发送消息。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html#update-rejected-pub-sub-topic 。
\$aws/things/ <i>thingName</i> /shadow/update/delta	Subscribe	当检测到事物影子的“reported”部分与“desired”部分之间存在差异时，Thing Shadows 服务将向该主题发送消息。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html#update-delta-pub-sub-topic 。
\$aws/things/ <i>thingName</i> /shadow/update/documents	Subscribe	每次影子更新成功执行时，AWS IoT 都会向该主题发布状态文档。有关更多信息，请参阅 http://docs.aws.amazon.com/iot/latest/developerguide//thing-shadow-mqtt.html#update-documents-pub-sub-topic 。

生命周期事件

AWS IoT 在以下各部分中讨论的 MQTT 主题下发布生命周期事件。借助这些消息，您可以接收消息代理发出的生命周期事件通知。

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。

接收生命周期事件时需要用到的策略

以下是接收生命周期事件时需要用到的策略的示例：

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

连接/断开连接事件

AWS IoT 在客户端建立连接或断开连接时将消息发布到以下 MQTT 主题：

```
$aws/events/presence/connected/clientId
```

或者

```
$aws/events/presence/disconnected/clientId
```

其中 **clientId** 是连接到 AWS IoT 消息代理或与之断开连接的 MQTT 客户端 ID。

发布到该主题的消息具有以下结构：

```
{  
    "clientId": "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6",  
    "timestamp": 1460065214626,  
    "eventType": "connected",  
    "sessionIdentifier": "00000000-0000-0000-0000-000000000000",  
    "principalIdentifier": "000000000000/ABCDEFGHIJKLMNPQRSTUVWXYZ:some-user/  
ABCDEFGHIJKLMNPQRSTUVWXYZ:some-user"  
}
```

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 接收不到生命周期事件。

eventType

事件类型。有效值为 connected 或 disconnected。

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。对于其他连接，这是 IAM 凭证。

sessionIdentifier

AWS IoT 中的全局唯一标识符，在会话持续时间内存在。

timestamp

大致的事件发生时间，使用自 Unix 纪元时间以来的毫秒数表示。时间戳的准确度是 +/- 2 分钟。

订阅/取消订阅事件

当客户端订阅或取消订阅 MQTT 主题时，AWS IoT 将向以下 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/ABCDEFGHIJKLMNPQRSTUVWXYZ:some-user/  
ABCDEFGHIJKLMNPQRSTUVWXYZ: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 接收不到生命周期事件。

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。对于其他连接，这是 IAM 凭证。

sessionIdentifier

AWS IoT 中的全局唯一标识符，在会话持续时间内存在。

timestamp

大致的事件发生时间，使用自 Unix 纪元时间以来的毫秒数表示。时间戳的准确度是 +/- 2 分钟。

主题

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。

AWS IoT 规则

规则使您的设备能够与 AWS 服务交互。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如下任务：

- 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。
- 使用 Amazon SNS 向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 Amazon Elasticsearch Service。
- 捕获 CloudWatch 指标。
- 更改 CloudWatch 警报。
- 将 MQTT 消息中的数据发送到 Amazon Machine Learning，以根据 Amazon ML 模型进行预测。
- 向 Salesforce IoT 输入流发送消息。

您必须先授予 AWS IoT 代表您访问 AWS 资源的权限，然后 才能够执行这些操作。执行这些操作时，即产生了您所使用的 AWS 服务的标准费用。

内容

- [授予 AWS IoT 所需的访问权限 \(p. 145\)](#)
- [传递角色权限 \(p. 146\)](#)
- [创建 AWS IoT 规则 \(p. 147\)](#)
- [查看您的规则 \(p. 150\)](#)
- [SQL 版本 \(p. 150\)](#)
- [排查规则问题 \(p. 152\)](#)

- [删除规则 \(p. 152\)](#)
- [AWS IoT 规则操作 \(p. 152\)](#)
- [AWS IoT SQL 参考 \(p. 162\)](#)

授予 AWS IoT 所需的访问权限

您可以使用 IAM 角色来控制每个规则可以访问的 AWS 资源。在创建规则之前，您必须创建一个 IAM 角色，并对其应用允许访问所需 AWS 资源的策略。执行规则时，AWS IoT 将担任此角色。

创建 IAM 角色 (AWS CLI)

1. 将以下信任策略文档 (用于授予 AWS IoT 担任此角色的权限) 保存到名为 iot-role-trust.json 的文件：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "iot.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

使用 [create-role](#) 命令创建 IAM 角色，并指定 iot-role-trust.json 文件：

```
aws iam create-role --role-name my-iot-role --assume-role-policy-document file://iot-role-trust.json
```

该命令的输出内容将如下所示：

```
{  
    "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 iot-policy.json 的文件中。

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "dynamodb:*",  
            "Resource": "*"  
        }  
    ]  
}
```

该 JSON 是授予 AWS IoT 管理员对 DynamoDB 的访问权限的示例策略文档。

使用 [create-policy](#) 命令授予 AWS IoT 在担任该角色后访问您的 AWS 资源的权限，并传入 iot-policy.json 文件：

```
aws iam create-policy --policy-name my-iot-policy --policy-document file://my-iot-policy-document.json
```

有关如何在 AWS IoT 策略中授予对 AWS 服务的访问权限的更多信息，请参阅 [创建 AWS IoT 规则 \(p. 147\)](#)。

[create-policy](#) 命令的输出内容将包含该策略的 ARN。您需要将该策略附加到角色。

```
{  
    "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，该角色授予针对规则操作中指定的资源的访问权限。一旦触发规则操作，规则引擎会代入该角色。必须在与规则相同的 AWS 账户中定义该角色。

在创建或替换规则时，您实际上将角色提交到规则引擎中。执行这项操作的用户需要具有 `iam:PassRole` 权限。为确保拥有此权限，您需要创建一个策略以授予 `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。您必须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。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 AWS IoT 规则

您可以配置规则以从连接的设备路由数据。规则包括以下部分：

规则名称

规则的名称。

可选说明

规则的文字说明。

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他位置推送数据。有关更多信息，请参阅[AWS IoT SQL 参考 \(p. 162\)](#)。

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。如果未设置该属性，将使用默认值 `2015-10-08`。

一个或多个操作

执行规则时 AWS IoT 执行的操作。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。

当您创建规则时，请注意发布到主题的数据量。如果您创建的规则包含通配符主题模式，它们可能与您的大部分消息匹配，并且您可能需要增加目标操作使用的 AWS 资源容量。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将能够访问规则处理的数据。

创建规则 (AWS CLI)

使用 `create-topic-rule` 命令创建规则：

```
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，角色 ARN 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)}",  
            "region": "us-east-1"  
        }  
    ]  
}
```

```
        "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 语句筛选消息，角色 ARN 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"
            }
        }
    ]
}
```

下面的负载文件示例包含将数据推送至 Amazon ES 的规则：

```
{
    "sql": "SELECT *, timestamp() as timestamp FROM 'iot/test'",
    "ruleDisabled": false,
    "awsIotSqlVersion": "2016-03-23",
    "actions": [
        {
            "elasticsearch": {
                "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"  
            }  
        }  
    ]  
}
```

下面的负载文件示例包含将数据推送至 Amazon Kinesis 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"  
            }  
        }  
    ]  
}
```

在下面的负载文件示例中，具有采用 Amazon Machine Learning `machinelearning_predict` 函数重新发布至某个主题（如果 MQTT 负载中的数据分类为 1）的规则。

```
{  
    "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"
    }
}
```

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 my-rule
```

SQL 版本

AWS IoT 规则引擎使用一种类似 SQL 的语法从 MQTT 消息选择数据。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 \(p. 147\)](#)。借助 `awsIotSqlVersion` 属性，您可以指定想要使用的 AWS IoT SQL 规则引擎版本。当部署新版本时，您可继续使用较旧的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"
            }
        }
    ]
}
```

当前支持的版本包括：

- 2015-10-08，2015 年 10 月 8 日构建的 SQL 原始版本。
- 2016-03-23，2016 年 3 月 23 日构建的 SQL 版本。
- beta，最新的 SQL 测试版本。使用此版本可能会给您的规则带来破坏性更改。

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。
- 对象间查询支持。

- 支持将阵列作为顶级对象输出。
- 添加 encode(value, encodingScheme) 函数，该函数可应用于 JSON 和非 JSON 格式数据。

对象间查询

此功能允许您查询 JSON 对象中的属性。例如，给定了以下 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 }  
  ]  
}
```

以及以下规则：

```
SELECT (SELECT v FROM e WHERE n = 'temperature') as temperature FROM 'my/topic'
```

该规则将生成以下输出：

```
{"temperature": [{"v":22.5}]}
```

使用相同的 MQTT 消息，并给定一个略复杂的规则，如：

```
SELECT get((SELECT v FROM e WHERE n = 'temperature'),1).v as temperature FROM 'topic'
```

该规则将生成以下输出：

```
{"temperature":22.5}
```

将 Array 作为顶级对象输出

此功能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'
```

该规则将生成以下输出：

```
[1,2,3,4]
```

对函数进行编码

根据指定的编码方案，将负载（可能是非 JSON 数据）编码为字符串表示形式。

排查规则问题

如果您遇到规则问题，应启用 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。有关使用 Amazon CloudWatch Logs 的更多信息，请参阅设置 CloudWatchLogs。

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 my-rule
```

AWS IoT 规则操作

AWS IoT 规则操作用于指定规则触发后应执行的操作。您可以定义操作以将数据写入 DynamoDB 数据库或 Kinesis 流，或者调用 Lambda 函数等。支持以下操作：

- `cloudwatchAlarm` - 更改 CloudWatch 警报。
- `cloudwatchMetric` - 捕获 CloudWatch 指标。
- `dynamoDB` - 将数据写入 DynamoDB 数据库。
- `dynamoDBv2` - 将数据写入 DynamoDB 数据库。
- `elasticsearch` - 将数据写入 Amazon Elasticsearch Service 域。
- `firehose` - 将数据写入 Amazon Kinesis Firehose 流。
- `kinesis` - 将数据写入 Kinesis 流。
- `lambda` - 调用 Lambda 函数。
- `s3` - 将数据写入 Amazon S3 存储桶。
- `sns` - 将数据编写为推送通知。
- `sqs` - 将数据写入 SQS 队列。
- `republish` - 在另一个 MQTT 主题上重新发布消息。
- `salesforce`，用于将消息写入 Salesforce IoT 输入流。

Note

AWS IoT 规则引擎当前不会重新尝试传输在发布至另一个服务时失败的消息。

以下部分将详细讨论每项操作。

CloudWatch 警报操作

CloudWatch 警报操作允许您更改 CloudWatch 警报状态。您可以在此调用中指定状态更改原因和状态值。使用 CloudWatch 警报操作创建 AWS IoT 规则时，您必须指定以下信息：

`roleArn`

允许访问 CloudWatch 警报的 IAM 角色。

alarmName

CloudWatch 警报名称。

stateReason

警报更改的原因。

stateValue

警报状态的值。可接受的值包括 OK、ALARM、INSUFFICIENT_DATA。

Note

确保与规则关联的角色拥有授予 `cloudwatch:SetAlarmState` 权限的策略。

下面的 JSON 示例介绍了如何在 AWS IoT 规则中定义 CloudWatch 警报操作：

```
{  
  "rule": {  
    "sql": "SELECT * FROM 'some/topic'",  
    "ruleDisabled": false,  
    "actions": [{  
      "cloudwatchAlarm": {  
        "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw",  
        "alarmName": "IotAlarm",  
        "stateReason": "Temperature stabilized.",  
        "stateValue": "OK"  
      }  
    }]  
  }  
}
```

有关更多信息，请参阅 [CloudWatch 警报](#)。

CloudWatch 指标操作

CloudWatch 指标操作允许您捕获 CloudWatch 指标。您可以指定指标命名空间、名称、值、单位和时间戳。使用 CloudWatch 指标操作创建 AWS IoT 规则时，您必须指定以下信息：

roleArn

允许访问 CloudWatch 指标的 IAM 角色。

metricNamespace

CloudWatch 指标命名空间名称。

metricName

CloudWatch 指标名称。

metricValue

CloudWatch 指标值。

metricUnit

CloudWatch 支持的指标单位。

metricTimestamp

可选 Unix 时间戳。

Note

确保与规则关联的角色拥有授予 `cloudwatch:PutMetricData` 权限的策略。

下面的 JSON 示例介绍了如何在 AWS IoT 规则中定义 CloudWatch 指标操作：

```
{  
    "rule": {  
        "sql": "SELECT * FROM 'some/topic'",  
        "ruleDisabled": false,  
        "actions": [  
            {  
                "cloudwatchMetric": {  
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_cw",  
                    "metricNamespace": "IoTNamespace",  
                    "metricName": "IoTMetric",  
                    "metricValue": "1",  
                    "metricUnit": "Count",  
                    "metricTimestamp": "1456821314"  
                }  
            }]  
    }  
}
```

有关更多信息，请参阅 [CloudWatch 指标](#)。

DynamoDB 操作

dynamoDB 操作允许您将所有或部分 MQTT 消息写入 DynamoDB 表。创建 DynamoDB 规则时，您必须指定以下信息：

hashKeyType

哈希键 (也称为分区键) 的数据类型。有效值为：“STRING” 或 “NUMBER”。

hashKeyField

哈希键 (也称为分区键) 的名称。

hashKeyValue

哈希键的值。

rangeKeyType

可选。范围键 (也称为排序键) 的数据类型。有效值为：“STRING” 或 “NUMBER”。

rangeKeyField

可选。范围键 (也称为排序键) 的名称。

rangeKeyValue

可选。范围键的值。

operation

可选。要执行的操作类型。该信息遵循替换模板，因此可以是 `${operation}`，但替换必须产生以下操作之一：`INSERT`、`UPDATE` 或 `DELETE`。

payloadField

可选。负载将写入的字段的名称。如果省略此值，负载将写入 `payload` 字段。

table

DynamoDB 表的名称。

roleARN

允许访问 DynamoDB 表的 IAM 角色。该角色至少须允许 `dynamoDB:PutItem` IAM 操作。

写入 DynamoDB 表的数据是规则的 SQL 语句的结果。`hashKeyValue` 和 `rangeKeyValue` 字段通常由表达式（例如，`“${topic()}”` 或 `“${timestamp()}”`）组成。

Note

非 JSON 数据以二进制数据形式写入 DynamoDB。DynamoDB 控制台以 Base64 编码文本格式显示数据。

确保与规则关联的角色拥有授予 `dynamodb:PutItem` 权限的策略。

下面的 JSON 示例介绍了如何在 AWS IoT 规则中定义 `dynamoDB` 操作：

```
{  
    "rule": {  
        "ruleDisabled": false,  
        "sql": "SELECT * AS message FROM 'some/topic'",  
        "description": "A test Dynamo DB rule",  
        "actions": [{  
            "dynamodb": {  
                "hashKeyField": "key",  
                "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamodb",  
                "tableName": "my_ddb_table",  
                "hashKeyValue": "${topic()}",  
                "rangeKeyValue": "${timestamp()}",  
                "rangeKeyField": "timestamp"  
            }  
        }]  
    }  
}
```

有关更多信息，请参阅 [Amazon DynamoDB 入门指南](#)。

DynamoDBv2 操作

`dynamoDBv2` 操作允许您将所有或部分 MQTT 消息写入 DynamoDB 表。负载的每个属性将写入 DynamoDB 数据库的单独一列。创建 DynamoDB 规则时，您必须指定以下信息：

roleARN

允许访问 DynamoDB 表的 IAM 角色。该角色至少须允许 `dynamoDB:PutItem` IAM 操作。

tableName

DynamoDB 表的名称。

Note

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。

写入 DynamoDB 表的数据是规则的 SQL 语句的结果。

Note

确保与规则关联的角色拥有授予 `dynamodb:PutItem` 权限的策略。

下面的 JSON 示例介绍了如何在 AWS IoT 规则中定义 dynamoDB 操作：

```
{  
    "rule": {  
        "ruleDisabled": false,  
        "sql": "SELECT * AS message FROM 'some/topic'",  
        "description": "A test DynamoDBv2 rule",  
        "actions": [  
            {  
                "dynamodbv2": {  
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_dynamodbv2",  
                    "putItem": {  
                        "tableName": "my_ddb_table"  
                    }  
                }  
            }  
        ]  
    }  
}
```

有关更多信息，请参阅 [Amazon DynamoDB 入门指南](#)。

Amazon ES 操作

`elasticsearch` 操作允许您将 MQTT 消息中的数据写入 Amazon Elasticsearch Service 域。然后，您可以使用 Kibana 等工具来查询和可视化 Amazon ES 中的数据。当您使用 `elasticsearch` 操作创建 AWS IoT 规则时，必须指定以下信息：

`endpoint`

您的 Amazon ES 域的终端节点。

`index`

您要在其中存储数据的 Amazon ES 索引。

`type`

您存储的文档类型。

`id`

每个文档的唯一标识符。

Note

确保与规则关联的角色拥有授予 `es:ESHttpPut` 权限的策略。

下面的 JSON 示例介绍了如何在 AWS IoT 规则中定义 `elasticsearch` 操作：

```
{  
    "rule":{  
        "sql":"SELECT *, timestamp() as timestamp FROM 'iot/test'",  
        "ruleDisabled":false,  
        "actions": [  
            {  
                "elasticsearch":{  
                    "roleArn":"arn:aws:iam::123456789012:role/aws_iot_es",  
                    "endpoint":"https://my-endpoint",  
                    "index":"my-index",  
                    "type":"my-type",  
                    "id":"${newuuid()}"  
                }  
            }  
        ]  
    }  
}
```

```
        ]
    }
}
```

有关更多信息，请参阅 [Amazon ES 开发人员指南](#)。

Firehose 操作

`firehose` 操作可将触发规则的 MQTT 消息中的数据发送至 Kinesis Firehose 流。使用 `firehose` 操作创建规则时，您必须指定以下信息：

`deliveryStreamName`

消息数据写入的 Kinesis Firehose 流。

`roleArn`

允许访问 Kinesis Firehose 的 IAM 角色。

`separator`

将用于分隔写入 Firehose 流的记录的字符分隔符。有效值为：`\n` (换行符)、`\t` (制表符)、`\r\n` (Windows 换行符)、`,` (逗号)。

Note

确保与规则关联的角色拥有授予 `firehose:PutRecord` 权限的策略。

下面的 JSON 示例介绍了如何使用 `firehose` 操作创建 AWS IoT 规则：

```
{
  "rule": {
    "sql": "SELECT * FROM 'some/topic'",
    "ruleDisabled": false,
    "actions": [
      {
        "firehose": {
          "roleArn": "arn:aws:iam::123456789012:role/aws_iot_firehose",
          "deliveryStreamName": "my_firehose_stream"
        }
      }
    ]
  }
}
```

有关更多信息，请参阅 [Kinesis Firehose 开发人员指南](#)。

Kinesis 操作

`kinesis` 操作允许您将 MQTT 消息中的数据写入 Kinesis 流。使用 `kinesis` 操作创建 AWS IoT 规则时，您必须指定以下信息：

`stream`

数据写入的 Kinesis 流。

`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。

Note

确保与规则关联的策略拥有 `kinesis:PutRecord` 权限。

下面的 JSON 示例介绍了如何在 AWS IoT 规则中定义 `kinesis` 操作：

```
{  
    "rule": {  
        "sql": "SELECT * FROM 'some/topic'",  
        "ruleDisabled": false,  
        "actions": [{  
            "kinesis": {  
                "roleArn": "arn:aws:iam::123456789012:role/aws_iot_kinesis",  
                "streamName": "my_kinesis_stream",  
                "partitionKey": "${topic()}"  
            }  
        }]  
    }  
}
```

有关更多信息，请参阅 [Kinesis 开发人员指南](#)。

Lambda 操作

lambda 操作会调用 Lambda 函数，并传入触发规则的 MQTT 消息。为了让 AWS IoT 调用 Lambda 函数，您必须配置一个策略以向 AWS IoT 授予 `lambda:InvokeFunction` 权限。Lambda 函数使用基于资源的策略，因此您必须将该策略附加至 Lambda 函数本身。使用以下 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:us-east-2: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，您正在通过添加新的权限来更新其资源策略。

`--region`

您的账户所处的 AWS 区域。

`--principal`

获取权限的委托人。这应该是 `iot.amazonaws.com`，以授予 AWS IoT 调用 Lambda 函数的权限。

`--source-arn`

规则的 ARN。您可以使用 `get-topic-rule` CLI 命令来获取规则的 ARN。

`--source-account`

定义规则的 AWS 账户。

`--statement-id`

唯一的语句标识符。

`--action`

要在此语句中允许的 Lambda 操作。在本示例中，我们要允许 AWS IoT 调用 Lambda 函数，因此我们指定 `lambda:InvokeFunction`。

Note

如果您在不提供源 ARN 的情况下为 AWS IoT 委托人添加权限，则所有通过 Lambda 操作创建规则的 AWS 账户都可以触发从 AWS IoT 调用 Lambda 函数的规则。

有关更多信息，请参阅 [Lambda 权限模型](#)。

在使用 lambda 操作创建规则时，您必须指定在触发规则后调用的 Lambda 函数。

下面的 JSON 示例介绍了调用 Lambda 函数的规则：

```
{  
  "rule": {  
    "sql": "SELECT * FROM 'some/topic'",  
    "ruleDisabled": false,  
    "actions": [{  
      "lambda": {  
        "functionArn": "arn:aws:lambda:us-  
east-2:123456789012:function:myLambdaFunction"  
      }  
    }]  
  }  
}
```

有关更多信息，请参阅 [AWS Lambda 开发人员指南](#)。

Republish 操作

republish 操作允许您将触发角色的消息重新发布至另一个 MQTT 主题。使用 republish 操作创建规则时，您必须指定以下信息：

topic

消息重新发布到的 MQTT 主题。

roleArn

允许发布至 MQTT 主题的 IAM 角色。

Note

确保与规则关联的角色拥有授予 iot:Publish 权限的策略。

```
{  
  "rule": {  
    "sql": "SELECT * FROM 'some/topic'",  
    "ruleDisabled": false,  
    "actions": [{  
      "republish": {  
        "topic": "another/topic",  
        "roleArn": "arn:aws:iam::123456789012:role/aws_iot_republish"  
      }  
    }]  
  }  
}
```

S3 操作

s3 操作可将触发规则的 MQTT 消息中的数据写入 Amazon S3 存储桶。使用 s3 操作创建 AWS IoT 规则时，您必须指定以下信息：

bucket

数据写入的 Amazon S3 存储桶。

cannedacl

控制对象访问的 Amazon S3 标准 ACL 由对象键标识。有关更多信息，请参阅 [S3 标准 ACL](#)。

key

数据写入的文件路径。例如，如果此参数的值为“\${topic()}/\${timestamp()}”，消息发送至的主题为“this/is/my/topic”，当前时间戳为 1460685389，则数据将写入 Amazon S3 上“this/is/my/topic”文件夹中名为“1460685389”的文件。

Note

使用静态键将导致每次调用规则都会覆盖 Amazon S3 中的单个文件。更多的常用案例将使用消息时间戳或其他唯一的消息标识符，因此将在 Amazon S3 中针对接收的每个消息保存一个新文件。

roleArn

允许访问 Amazon S3 存储桶的 IAM 角色。

Note

确保与规则关联的角色拥有授予 s3:PutObject 权限的策略。

下面的 JSON 示例介绍了如何在 AWS IoT 规则中定义 s3 操作：

```
{  
    "rule": {  
        "sql": "SELECT * FROM 'some/topic'",  
        "ruleDisabled": false,  
        "actions": [  
            {  
                "s3": {  
                    "roleArn": "arn:aws:iam::123456789012:role/aws_iot_s3",  
                    "bucketName": "my-bucket",  
                    "key": "${topic()}/${timestamp()}"  
                }  
            }  
        ]  
    }  
}
```

有关更多信息，请参阅 [Amazon S3 开发人员指南](#)。

SNS 操作

sns 操作可将触发规则的 MQTT 消息中的数据作为 SNS 推送通知发送。使用 sns 操作创建规则时，您必须指定以下信息：

messageFormat

消息格式。接受的值为“JSON”和“RAW”。该属性的默认值为“RAW”。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。

roleArn

允许访问 SNS 的 IAM 角色。

targetArn

推送通知将发送到的 SNS 主题或单个设备。

Note

确保与规则关联的策略拥有 sns:Publish 权限。

下面的 JSON 示例介绍了如何在 AWS IoT 规则中定义 sns 操作：

```
{  
    "rule": {  
        "sql": "SELECT * FROM 'some/topic'",  
        "ruleDisabled": false,  
        "actions": [{  
            "sns": {  
                "targetArn": "arn:aws:sns:us-east-2:123456789012:my_sns_topic",  
                "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sns"  
            }  
        }]  
    }  
}
```

有关更多信息，请参阅 [Amazon SNS 开发人员指南](#)。

SQS 操作

sqs 操作可将触发规则的 MQTT 消息中的数据发送至 SQS 队列。使用 sqs 操作创建规则时，您必须指定以下信息：

queueUrl

数据写入的 SQS 队列的 URL。

useBase64

如果您希望 MQTT 消息数据在写入 SQS 队列前进行 Base64 编码，则设置为 true；否则，设置为 false。

roleArn

允许访问 SQS 队列的 IAM 角色。

Note

确保与规则关联的角色拥有授予 sqs:SendMessage 权限的策略。

下面的 JSON 示例介绍了如何使用 sqs 操作创建 AWS IoT 规则：

```
{  
    "rule": {  
        "sql": "SELECT * FROM 'some/topic'",  
        "ruleDisabled": false,  
        "actions": [{  
            "sqs": {  
                "queueUrl": "https://sqs.us-east-2.amazonaws.com/123456789012/  
my_sqeue",  
                "roleArn": "arn:aws:iam::123456789012:role/aws_iot_sqe",  
                "useBase64": false  
            }  
        }]  
    }  
}
```

有关更多信息，请参阅 [Amazon SQS 开发人员指南](#)。

Salesforce 操作

`salesforce` 操作将来自触发了规则的 MQTT 消息的数据发送到 Salesforce IoT 输入流。使用 `salesforce` 操作创建规则时，您必须指定以下信息：

`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。请参阅 [Salesforce IoT 文档](#)以了解更多信息。

`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。请参阅 [Salesforce IoT 文档](#)以了解更多信息。

Note

这些参数不支持替换。

下面的 JSON 示例介绍了如何使用 `salesforce` 操作创建 AWS IoT 规则：

```
{  
    "sql": "expression",  
    "ruleDisabled": false,  
    "awsIotSqlVersion": "2016-03-23",  
    "actions": [  
        {"  
            "salesforce": {  
                "token": "ABCDEFGHI123456789abcdefghi123456789",  
                "url": "https://ingestion-cluster-id.my-env.sfdcn.com/streams/stream-id/  
connection-id/my-event"  
            }  
        }  
    ]  
}
```

有关更多信息，请参阅 [Salesforce IoT 文档](#)。

AWS IoT SQL 参考

在 AWS IoT 中，规则通过一种类似 SQL 的语法来定义。SQL 语句由三类子句组成：

`SELECT`

必需。从传入负载提取信息并执行转换。

`FROM`

必需。规则接收消息的 MQTT 主题筛选条件。

`WHERE`

可选。添加用于确定是否评估规则以及是否执行其操作的条件逻辑。

SQL 语句的示例如下所示：

```
SELECT color AS rgb FROM 'a/b' 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 'a/b' 主题上发布，则触发规则并评估 SQL 语句。如果 "temperature" 属性大于 50，SQL 语句将提取 `rgb` 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 规则操作 \(p. 152\)](#)。

数据类型

AWS IoT 规则引擎支持所有 JSON 数据类型。

受支持数据类型

Type	意义
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 部分功能返回双精度 Decimal，而不是 34 位数字精度。
Boolean	True 或者 False。
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 'a/b'"
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}
```

Type	意义
	<p>序列化为 JSON 如下：</p> <pre>{"foo":null}</pre> <p>同样，<code>Undefined</code> 在被自身序列化时会转化为空字符串。使用无效参数调用函数（例如，错误的类型、错误的参数号等）将返回 <code>Undefined</code>。</p>

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
<code>Int</code>	没有小数点的 <code>Decimal</code> 。
<code>Decimal</code>	源值。
<code>Boolean</code>	<code>Undefined</code> 。（您可以明确使用强制转换函数使 <code>trun = 1.0</code> ， <code>false = 0.0</code> 。）
<code>String</code>	SQL 引擎会尝试将字符串解析为 <code>Decimal</code> 。我们会尝试解析与正则表达式相匹配的字符串： <code>^-?\d+(\.\d+)?((\?i)E-?\d+)?\$</code> 。可自动转换为 <code>Decimal</code> 的字符串示例包括 "0"、"-1.2"、"5E-12"。
数组	<code>Undefined</code> 。
<code>Object</code>	<code>Undefined</code> 。
<code>Null</code>	<code>Null</code> 。
未定义	<code>Undefined</code> 。

To Int

参数类型	结果
<code>Int</code>	源值。
<code>Decimal</code>	源值都舍入到最接近的 <code>Int</code> 。
<code>Boolean</code>	<code>Undefined</code> 。（您可以明确使用强制转换函数使 <code>trun = 1.0</code> ， <code>false = 0.0</code> 。）
<code>String</code>	SQL 引擎会尝试将字符串解析为 <code>Decimal</code> 。我们会尝试解析与正则表达式相匹配的字符串： <code>^-?\d+(\.\d+)?((\?i)E-?\d+)?\$</code> 。

参数类型	结果
	可自动转换为 Decimal 的字符串示例包括 "0"、"-1.2"、"5E-12"。我们会尝试将 String 转换为 Decimal，然后截断该 Decimal 的小数位，得到一个 Int。
数组	Undefined。
Object	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，任何非零值 = True。)
Decimal	Undefined。(您可以明确使用强制转换函数使 0 = False，任何非零值 = True。)
Boolean	原始值。
String	"true"=True 和 "false"=False (不区分大小写)。其他字符串值将为 Undefined。
数组	Undefined。
Object	Undefined。
Null	Undefined。
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。
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 将用引号括起来。Decimal、Int、Boolean 和 Null 不必如此。
Object	序列化为 JSON 的对象。结果字符串为键值对的逗号分隔列表，以大括号开始并结束。String 将用引号括起来。Decimal、Int、Boolean 和 Null 不必如此。

参数类型	结果
Null	Undefined。
未定义	未定义。

运算符

SELECT、FROM 和 WHERE 子句中可以使用以下运算符。

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 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。
其他值	其他值	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 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)，则它们将被转换为 Boolean 并作为 boolean OR boolean 正常处理。
其他值	其他值	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 false。需要布尔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 false。
String	如果字符串为 "true" 或 "false"（不区分大小写），它将被转换为对应的布尔值，并返回相反的值。
其他值	Undefined。

> operator

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，则返回 true；否则为 false。
String/Int/ Decimal	String/Int/ Decimal	如果所有字符串可以转换为 Decimal，则 Boolean。如果左侧操作数大于右侧操作数，则返回 true；否则为 false。
其他值	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*.

>= 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等于右侧操作数，则返回 true；否则为 false。
String/Int/ Decimal	String/Int/ Decimal	如果所有字符串可以转换为 Decimal，则 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。
其他值	Undefined。	Undefined。

< operator

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，则返回 true；否则为 false。
String/Int/ Decimal	String/Int/ Decimal	如果所有字符串可以转换为 Decimal，则 Boolean。如果左侧操作数小于右侧操作数，则返回 true；否则为 false。
其他值	Undefined	Undefined

<= operator

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，则返回 true；否则为 false。
String/Int/ Decimal	String/Int/ Decimal	如果所有字符串可以转换为 Decimal，则 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。
其他值	Undefined	Undefined

<> operator

返回 Boolean 结果。如果左右两个操作数不相等，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目不相等且顺序不同，则为 true；否则为 false
Object	Object	如果各个操作数的键和值不相等，则为 true；否则为 false。 键/值的顺序不重要。
Null	Null	False.

左侧操作数	右侧操作数	输出
任意值	Undefined	未定义。
Undefined	任意值	未定义。
不匹配的类型	不匹配的类型	True.

= operator

返回 Boolean 结果。如果左右两个操作数相等，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。
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目相等且顺序相同，则为 true；否则为 false。
Object	Object	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.

+ operator

“+”是一个重载运算符。它可用于字符串连接或相加。

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。
其他值	其他值	Undefined。

- operator

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/ Decimal	String/Int/ Decimal	如果所有字符串都正确转换为 Decimal，则返回 Decimal 值。从左侧操作数中减去右侧操作数。否则返回 Undefined。
其他值	其他值	Undefined。
其他值	其他值	Undefined。

* operator

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。
String/Int/ Decimal	String/Int/ Decimal	如果所有字符串都正确转换为 Decimal，则返回 Decimal 值。左侧操作数乘以右侧操作数。否则返回 Undefined。
其他值	其他值	Undefined。

/ operator

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/ Decimal	String/Int/ Decimal	如果所有字符串都正确转换为 Decimal，则返回 Decimal 值。左侧操作数除以右侧操作数。否则返回 Undefined。

左侧操作数	右侧操作数	输出
其他值	其他值	Undefined。

% operator

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。
String/Int/ Decimal	String/Int/ Decimal	如果所有 String 都正确转换为 Decimal，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。
其他值	其他值	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 2015-10-8 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。
Object	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 2015-10-8 及更高版本支持。

例如：

```
accountid() = "123456789012"
```

acos(Decimal)

以弧度形式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 2015-10-8 及更高版本支持。

示例： $\text{acos}(0) = 1.5707963267948966$

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。
Object	Undefined。
Null	Undefined。
未定义	Undefined。

asin(Decimal)

以弧度形式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 2015-10-8 及更高版本支持。

示例： $\text{asin}(0) = 0.0$

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。
Object	Undefined。
Null	Undefined。
未定义	Undefined。

atan(Decimal)

以弧度形式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 2015-10-8 及更高版本支持。

示例：`atan(0) = 0.0`

参数类型	结果
<code>Int</code>	<code>Decimal</code> (双精度)，参数的反正切值。虚数结果返回 <code>Undefined</code> 。
<code>Decimal</code>	<code>Decimal</code> (双精度)，参数的反正切值。虚数结果返回 <code>Undefined</code> 。
<code>Boolean</code>	<code>Undefined</code> 。
<code>String</code>	<code>Decimal</code> ，参数的反正切值。如果字符串无法转换，则结果为 <code>Undefined</code> 。虚数结果返回 <code>Undefined</code> 。
数组	<code>Undefined</code> 。
<code>Object</code>	<code>Undefined</code> 。
<code>Null</code>	<code>Undefined</code> 。
未定义	<code>Undefined</code> 。

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 2015-10-8 及更高版本支持。

示例：`atan2(1, 0) = 1.5707963267948966`

参数类型	参数类型	结果
<code>Int/Decimal</code>	<code>Int/Decimal</code>	<code>Decimal</code> (双精度)， x
<code>Int/Decimal/String</code>	<code>Int/Decimal/String</code>	<code>Decimal</code> ，所描述点的结果为 <code>Undefined</code> 。
其他值	其他值	<code>Undefined</code> 。

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 2015-10-8 及更高版本支持。

示例：`bitand(13, 5) = 5`

参数类型	参数类型	结果
<code>Int</code>	<code>Int</code>	<code>Int</code> ，对两个参数逐位

参数类型	参数类型	结果
Int/Decimal	Int/Decimal	Int , 对两个参数逐位入至最近的 Int。如果 Undefined。
Int/Decimal/String	Int/Decimal/String	Int , 对两个参数逐位 Decimal 并向下舍入 Undefined。
其他值	其他值	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 2015-10-8 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
Int/Decimal	Int/Decimal	Int , 对两个参数逐位入至最近的 Int。如果 Undefined。
Int/Decimal/String	Int/Decimal/String	Int , 对两个参数逐位 Decimal 并向下舍入 Undefined。
其他值	其他值	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 2015-10-8 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
Int/Decimal	Int/Decimal	Int , 对两个参数逐位入至最近的 Int。
Int/Decimal/String	Int/Decimal/String	Int , 对两个参数逐位 Decimal 并向下舍入结果为 Undefined。
其他值	其他值	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 2015-10-8 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。
String	Int，对参数逐位执行非运算。String 将转换为 Decimal 并向下舍入至最近的 Int。如果任何转换失败，结果为 Undefined。
其他值	其他值。

cast()

将值从一个数据类型转换为另一个数据类型。强制转换的行为在多数情况下与标准转换相似，增加了在数字与 Boolean 值之间强制转换的功能。如果无法确定一种类型如何强制转换为另一种类型，则结果为 Undefined。SQL 版本 2015-10-8 及更高版本支持。格式：cast(# as ##)。

例如：

```
cast(true as Decimal) = 1.0
```

在调用 cast 时“as”后面可以出现以下关键字：

Keyword	结果
Decimal	将值强制转换为 Decimal。
Bool	将值强制转换为 Boolean。
Boolean	将值强制转换为 Boolean。
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。
Int	将值强制转换为 Int。

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。

参数类型	结果
Decimal	源值。
Boolean	true = 1.0 , false = 0.0。
String	会尝试将字符串解析为 Decimal。我们将尝试解析字符串来匹配正则表达式： <code>^-?\d+(\.\d+)?((?)E-?)\d+)?\$</code> 。可自动转换为 Decimal 的 String 示例包括 "0"、"-1.2"、"5E-12"。
数组	Undefined。
Object	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	会尝试将字符串解析为 Decimal。我们将尝试解析字符串来匹配正则表达式： <code>^-?\d+(\.\d+)?((?)E-?)\d+)?\$</code> 。可自动转换为 Decimal} 的 String 示例包括 "0"、"-1.2"、"5E-12"。我们会尝试将字符串转换为 Decimal，然后向下舍入到最近的 Int。
数组	Undefined。
Object	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。
Decimal	0 = False，任何非零值 = True。
Boolean	源值。
String	"true" = True 和 "false" = False (不区分大小写)。其他字符串值 = Undefined。
数组	Undefined。
Object	Undefined。

参数类型	结果
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。
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)。其他字符串值 = 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 用引号括起来。Decimal、Int、Boolean 不必如此。
Object	序列化为 JSON 的对象。JSON 字符串为键值对的逗号分隔列表，以大括号开始并结束。String 用引号括起来。Decimals、Int、Boolean 和 Null 不必如此。
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 2015-10-8 及更高版本支持。

示例：

```
ceil(1.2) = 2
ceil(11.2) = -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。
其他值	Undefined。

chr(String)

返回给定 Int 参数对应的 ASCII 字符。SQL 版本 2015-10-8 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 <code>Undefined</code> 。
Decimal	与指定的 ASCII 值对应的字符。 <code>Decimal</code> 参数会向下舍入至最近的 <code>Int</code> 。如果参数不是有效的 ASCII 值，则结果为 <code>Undefined</code> 。
Boolean	<code>Undefined</code> 。
String	如果 <code>String</code> 可以转换为 <code>Decimal</code> ，则向下舍入到最近的 <code>Int</code> 。如果参数不是有效的 ASCII 值，则结果为 <code>Undefined</code> 。
数组	<code>Undefined</code> 。
Object	<code>Undefined</code> 。
Null	<code>Undefined</code> 。
其他值	<code>Undefined</code> 。

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 2015-10-8 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 2015-10-8 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 <code>Array</code> ，那么结果为包含所有参数的一个数组。如果没有参数为 <code>Array</code> ，并且至少有一个参数为 <code>String</code> ，则结果是所有参数的 <code>String</code> 表示的联接。参数将使用上文列出的标准转换被转换为 <code>String</code> 。 。

cos(Decimal)

以弧度形式返回数字的余弦值。在代入函数之前，`Decimal` 参数舍入到双精度。SQL 版本 2015-10-8 及更高版本支持。

例如：

`cos(0) = 1.`

参数类型	结果
<code>Int</code>	<code>Decimal</code> (双精度)，参数的余弦值。虚数结果返回 <code>Undefined</code> 。
<code>Decimal</code>	<code>Decimal</code> (双精度)，参数的余弦值。虚数结果返回 <code>Undefined</code> 。
<code>Boolean</code>	<code>Undefined</code> 。
<code>String</code>	<code>Decimal</code> (双精度)，参数的余弦值。如果字符串无法转换为 <code>Decimal</code> ，则结果为 <code>Undefined</code> 。虚数结果返回 <code>Undefined</code> 。
数组	<code>Undefined</code> 。
<code>Object</code>	<code>Undefined</code> 。
<code>Null</code>	<code>Undefined</code> 。
未定义	<code>Undefined</code> 。

cosh(Decimal)

以弧度形式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 2015-10-8 及更高版本支持。

示例：`cosh(2.3) = 5.037220649268761`。

参数类型	结果
<code>Int</code>	<code>Decimal</code> (双精度)，参数的双曲余弦值。虚数结果返回 <code>Undefined</code> 。

参数类型	结果
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 Undefined。虚数结果返回 Undefined。
数组	Undefined.
Object	Undefined.
Null	Undefined.
未定义	Undefined.

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。

值

AWS IoT SQL 参考 (p. 162) 中所定义的任何有效的表达式。此外，您还可以指定 * 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，评估结果将在编码之前首先转换为字符串。

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 2015-10-8 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尾，则结果为 true。如果第一个参数以第二个参数开头，则结果为 false。
其他值	其他值	两个参数都使用标准转义字符。如果第一个参数以第二个参数结尾，则结果为 true。如果第一个参数以第二个参数开头，则结果为 false。如果任一参数为 Null 或 Undefined，则结果为 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 2015-10-8 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。
其他值	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 2015-10-8 及更高版本支持。

示例：

```
get(["a", "b", "c"], 1) = "b"
get({"a": "b"}, "a") = "b"
get("abc", 1) = "b"
```

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 (已经转零开始索引得到的项目 Undefined。如果索于数组长度) , 则结果。
字符串	任何类型 (转换为 Int)	在第二个参数 (已经转零开始索引得到的字符 Undefined。如果索于字符串长度) , 则结果。
Object	String (不进行转换)	第一个参数对象中存储键相对应。
其他值	任意值	Undefined。

get_thing_shadow(thingName, roleARN)

返回指定事物的影子。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。

roleArn

String : 具有 iot:GetThingShadow 的 ARN 角色。

例如：

```
SELECT * from 'a/b'
```

```
WHERE get_thing_shadow("MyThing", "arn:aws:iam::123456789012:role/  
AllowsThingShadowAccess") .state.reported.alarm = 'ON'
```

哈希函数

我们提供以下哈希函数：

- 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 2015-10-8 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 2015-10-8 及更高版本支持。

示例：

```
indexof("abcd", "bc") = 1
```

isNull()

返回该参数是否为 Null 值。SQL 版本 2015-10-8 及更高版本支持。

示例：

```
isNull(5) = false.
```

```
isNull(null) = true.
```

参数类型	结果
Int	false
Decimal	false
Boolean	false
String	false
Array	false
Object	false

参数类型	结果
Null	true
Undefined	false

isUndefined()

返回该参数是否为 `Undefined`。SQL 版本 2015-10-8 及更高版本支持。

示例：

```
isUndefined(5) = false.  
isNull(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
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 2015-10-8 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 2015-10-8 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。

参数类型	结果
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。
Object	Undefined。
Null	Undefined。
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 2015-10-8 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。
Object	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 String。SQL 版本 2015-10-8 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。如果输入的值在这一有效范围之外 , 则参数将被设置为与其最近的值 (0 或 1000)。SQL 版本 2015-10-8 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 格。
String	Decimal	Decimal 参数将向左侧填充指定数量的空格。
String	String	第二个参数将被转换为 Int，并在 String 左侧填充指定数量的空格。参数无法转换为 Int 时，将使用标准转换规则。
其他值	Int/Decimal/String	第一个值将使用标准转换规则。String 应用 LPAD 函数。其他值将返回 Undefined。
任意值	其他值	Undefined。

Ltrim(String)

从输入的 string 中删除所有前导空白 (制表符和空格)。SQL 版本 2015-10-8 及更高版本支持。

例如：

`Ltrim(" h i ") = "hi"。`

参数类型	结果
Int	Int 在删除所有前导空白之后的 String 表示。
Decimal	Decimal 在删除所有前导空白之后的 String 表示。
Boolean	布尔值 ("true" 或 "false") 在删除所有前导空白之后的 String 表示。
String	删除所有前导空白的参数。
数组	Array (使用标准转换规则) 删除所有前导空白之后的 String 表示。
Object	对象 (使用标准转换规则) 删除所有前导空白之后的 String 表示。
Null	Undefined。
未定义	Undefined。

machinelearning_predict(modelId)

利用 `machinelearning_predict` 函数，并根据 Amazon Machine Learning (Amazon ML) 模型使用来自 MQTT 消息的数据进行预测。SQL 版本 2015-10-8 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。

record

要传递到 Amazon ML 预测 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。

Algorithm

Amazon ML 用于预测的算法。该值必须为 SGD。

predictedScores

包含与每个标签对应的原始分类分数。

predictedValue

Amazon ML 预测的值。

mod(Decimal, Decimal)

返回第一个参数除以第二个参数的余数。SQL 版本 2015-10-8 及更高版本支持。您还可以使用“%”作为相同取模功能的插入运算符。SQL 版本 2015-10-8 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
Int/Decimal	Int/Decimal	Decimal，第一个参数
String/Int/Decimal	String/Int/Decimal	如果所有字符串转换为 第二个参数取模的值；

左侧操作数	右侧操作数	输出
其他值	其他值	Undefined。

nanvl(AnyValue, AnyValue)

如果第一个参数为有效 `Decimal`，则返回第一个参数；否则返回第二个参数。SQL 版本 2015-10-8 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。
<code>Decimal</code> (NaN)	任意值	第二个参数。
<code>Decimal</code> (非 NaN)	任意值	第一个参数。
其他值	任意值	第一个参数。

newuuid()

返回随机的 16 字节 UUID。SQL 版本 2015-10-8 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 2015-10-8 及更高版本支持。

示例：

`numbytes("hi") = 2`

`numbytes("€") = 3`

principal()

根据收到请求的终端节点 (MQTT 或 HTTP) 类型，返回 X.509 证书的指纹或事物名称。SQL 版本 2015-10-8 及更高版本支持。

例如：

`principal() = "ba67293af50bf2506f5f93469686da660c7c844e7b3950fb16813e0d31e9373"`

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。`parse_time` 函数的参数如下：

`pattern`

(`String`) 符合 [ISO 8601](#) 标准格式的日期/时间模式。(具体而言，此函数支持 [Joda-Time 格式](#)。)

timestamp

(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\(\) \(p. 197\)](#)。

timezone

(String) [Optional] 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](#)

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"}` 将发送到 S3 存储桶：

```
{  
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",  
    "rule": {  
        "awsIotSqlVersion": "2016-03-23",  
        "sql": "SELECT parse_time(\"yyyy.MM.dd G 'at' HH:mm:ss z\", 100000000, \"America/  
Belize\" ) as ts FROM 'A/B'",  
  
        "ruleDisabled": false,  
        "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）的负载将发送到 S3 存储桶：

```
{  
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",  
    "rule": {  
        "awsIotSqlVersion": "2016-03-23",  
        "sql": "SELECT parse_time(\"yyyy.MM.dd G 'at' HH:mm:ss z\", timestamp() ) as ts FROM  
'A/B'",  
        "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，负载将发送到密钥为“2017”的 S3 存储桶：

```
{  
    "ruleArn": "arn:aws:iot:us-east-2:ACCOUNT_ID:rule/RULE_NAME",
```

```

"rule": {
    "awsIotSqlVersion": "2016-03-23",
    "sql": "SELECT * FROM 'A/B'",
    "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 2015-10-8 及更高版本支持。SQL 版本 2015-10-8 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)，返回值。
Int/Decimal/String	Int/Decimal/String	Decimal (双精度) 返回值。所有字符串均转换为 Decimal 失败，
其他值	其他值	Undefined。

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 2015-10-8 及更高版本支持。

例如：

`rand() = 0.8231909191640703`

regexp_matches(String, String)

返回第一个参数是否包含第二个参数的匹配值 (正则表达式)。

例如：

`Regexp_matches("aaaa", "a{2,}") = true.`

`Regexp_matches("aaaa", "b") = false.`

第一个参数：

参数类型	结果
Int	Int 的 String 表示。

参数类型	结果
Decimal	Decimal 的 String 表示。
Boolean	布尔值 ("true" 或 "false") 的 String 表示。
String	这些区域有：String.
数组	Array (使用标准转换规则) 的 String 表示。
Object	对象 (使用标准转换规则) 的 String 表示。
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，生成的字符串不一定是正则表达式。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。)非字符串类型将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_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 2015-10-8 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	Int 的 String 表示。
Decimal	Decimal 的 String 表示。
Boolean	布尔值 ("true" 或 "false") 的 String 表示。
String	源值。
数组	Array (使用标准转换规则) 的 String 表示。
Object	对象 (使用标准转换规则) 的 String 表示。
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，生成的字符串不一定是正则表达式。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。)非字符串类型将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。用“\$”引用捕获组。SQL 版本 2015-10-8 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	Int 的 String 表示。
Decimal	Decimal 的 String 表示。
Boolean	布尔值 ("true" 或 "false") 的 String 表示。
String	String 参数。
数组	Array (使用标准转换规则) 的 String 表示。
Object	对象 (使用标准转换规则) 的 String 表示。
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，生成的字符串不一定是正则表达式。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。)非字符串类型将使用标准转换规则转换为 String。如果参数不是有效的正则表达式替代字符串，则结果为 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。如果输入的值在这一有效范围之外，则参数将被设置为与其最近的值 (0 或 1000)。SQL 版本 2015-10-8 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
<code>String</code>	<code>Int</code>	在 <code>String</code> 的右侧 填充由 <code>Int</code> 指定数 量的空 格。
<code>String</code>	<code>Decimal</code>	<code>Decimal</code> 参数将 向下舍 入到最 近的 <code>Int</code> ， 并且在 字符串 的右侧 填充由 <code>Int</code> 指定数 量的空 格。
<code>String</code>	<code>String</code>	第二个 参数将 转换为 <code>Decimal</code> ， 并向下 舍入到最 近的 <code>Int</code> 。 在 <code>String</code> 的右侧 填充由 <code>Int</code> 值 指定数 量的空 格。
其他值	<code>Int/Decimal/String</code>	第一个 值将使 用标准 转换规 则转 换为 <code>String</code> ， 然后 对该 <code>String</code> 应用 <code>rpad</code>

参数类型 1	参数类型 2	结果
		函数。如果它无法转换，则结果为 <code>Undefined</code> 。
任意值	其他值	<code>Undefined</code> 。

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 2015-10-8 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
<code>Int</code>	参数。
<code>Decimal</code>	<code>Decimal</code> 会向下舍入至最近的 <code>Int</code> 。
<code>String</code>	<code>Decimal</code> 会向下舍入至最近的 <code>Int</code> 。如果字符串无法转换为 <code>Decimal</code> ，则结果为 <code>Undefined</code> 。
其他值	<code>Undefined</code> 。

rtrim(String)

从输入的 `String` 中删除所有尾随空白（制表符和空格）。SQL 版本 2015-10-8 及更高版本支持。

示例：

`rtrim(" h i ") = " h i"`

参数类型	结果
<code>Int</code>	<code>Int</code> 的 <code>String</code> 表示。
<code>Decimal</code>	<code>Decimal</code> 的 <code>String</code> 表示。
<code>Boolean</code>	布尔值（"true" 或 "false"）的 <code>String</code> 表示。
数组	<code>Array</code> （使用标准转换规则）的 <code>String</code> 表示。
<code>Object</code>	对象（使用标准转换规则）的 <code>String</code> 表示。

参数类型	结果
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 2015-10-8 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 2015-10-8 及更高版本支持。
其他值	Undefined。

sin(Decimal)

以弧度形式返回数字的正弦值。在代入函数之前，Decimal 参数舍入到双精度。SQL 版本 2015-10-8 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。
Object	Undefined。
Null	Undefined。

参数类型	结果
Undefined	Undefined.

sinh(Decimal)

返回数字的双曲正弦。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 2015-10-8 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.
Object	Undefined.
Null	Undefined.
未定义	Undefined.

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 长度的索引将设置为 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)，则将对参数应用标准转换以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 2015-10-8 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) = "".
```

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 2015-10-8 及更高版本支持。

示例：`sqrt(9) = 3.0.`

参数类型	结果
<code>Int</code>	参数的平方根。
<code>Decimal</code>	参数的平方根。
<code>Boolean</code>	<code>Undefined.</code>
<code>String</code>	参数的平方根。如果字符串无法转换为 <code>Decimal</code> ，则结果为 <code>Undefined.</code>
数组	<code>Undefined.</code>
<code>Object</code>	<code>Undefined.</code>
<code>Null</code>	<code>Undefined.</code>
未定义	<code>Undefined.</code>

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 2015-10-8 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
<code>String</code>	<code>String</code>	第一个字符串是否以第二个字符串开头。
其他值	其他值	两个参数都使用标准转义字符。如果第一个字符串是否以第二个字符串开头，则结果为 <code>true</code> ；否则或 <code>Undefined</code> ，则结果为 <code>false</code> 。

tan(Decimal)

以弧度形式返回数字的正切值。在代入函数之前，`Decimal` 值舍入到双精度。SQL 版本 2015-10-8 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。
Object	Undefined。
Null	Undefined。
未定义	Undefined。

tanh(Decimal)

以弧度形式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 2015-10-8 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。
Object	Undefined。
Null	Undefined。
未定义	Undefined。

timestamp()

返回 AWS IoT 规则引擎观察到的当前时间戳 , 用距离 1970 年 1 月 1 日 (星期四) 协调世界时 (UTC) 00:00:00 的毫秒数来表示。SQL 版本 2015-10-8 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 参数用于指定从 1 开始的特定主题段。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 2015-10-8 及更高版本支持。

示例：

```
topic() = "things/myThings/thingOne"  
topic(1) = "things"
```

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 2015-10-8 及更高版本支持。

例如：

```
traceid() = "12345678-1234-1234-1234-123456789012"
```

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，则会设置为零。如果参数第二大于 34，则会设置为 34。将从结果中删除结尾的零。SQL 版本 2015-10-8 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。  
(2.00, 5) = 2.
```

参数类型 1	参数类型 2	结果
<code>Int</code>	<code>Int</code>	源值。
<code>Int/Decimal</code>	<code>Int/Decimal</code>	第一个参数被截断到由第二个参数指定的长度。第二个参数如果不是 <code>Int</code> ，将向下舍入至最近的 <code>Int</code> 。如果字符串转换失败，则结果为 <code>Undefined</code> 。
<code>Int/Decimal/String</code>	第一个参数被截断到由第二个参数所指定的长度。第二个参数如果不是 <code>Int</code> ，将向下舍入至最近的 <code>Int</code> 。String 将转换为 <code>Decimal</code> 值。如果字符串转换失败，则结果为 <code>Undefined</code> 。	
其他值	<code>Undefined</code> 。	

trim(String)

从输入的 `String` 中删除所有前导和尾随的空白。SQL 版本 2015-10-8 及更高版本支持。

例如：

```
Trim(" hi ") = "hi"
```

参数类型	结果
<code>Int</code>	<code>Int</code> 在删除所有前导和尾随空白之后的 <code>String</code> 表示。

参数类型	结果
Decimal	Decimal 在删除所有前导和尾随空白之后的 String 表示。
Boolean	Boolean ("true" 或 "false") 在删除所有前导和尾随空白之后的 String 表示。
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。
Object	对象使用标准转换规则进行转换后的 String 表示。
Null	Undefined。
未定义	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 2015-10-8 及更高版本支持。

示例：

```
upper("hello") = "HELLO"
upper(["hello"]) = ["HELLO"]
```

SELECT 语句

AWS IoT SELECT 子句基本与 ANSI SQL SELECT 子句相同，仅存在一些细微区别。

您可以使用 SELECT 子句从传入 MQTT 消息中提取信息。SELECT * 可以用于检索整个传入消息负载。例如：

```
Incoming payload published on topic 'a/b': {"color": "red", "temperature": 50}
SQL statement: SELECT * FROM 'a/b'
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 'a/b': {"color": "red", "temperature": 50}
SQL statement: SELECT color FROM 'a/b'
Outgoing payload: {"color": "red"}
```

您可以使用 AS 关键字重命名键。例如：

```
Incoming payload published on topic 'a/b': {"color": "red", "temperature": 50}
SQL: SELECT color AS my_color FROM 'a/b'
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 'a/b': {"color": "red", "temperature": 50}
SQL: SELECT color as my_color, temperature as farenheit FROM 'a/b'
```

```
Outgoing payload: {"my_color": "red", "farenheit": 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 'a/b': {"color": "red", "temperature": 50}
SQL: SELECT *, 15 as speed FROM 'a/b'
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。您只能选择一个项目。例如：

```
Incoming payload published on topic 'a/b': {"color": "red", "temperature": 50}
SQL: SELECT VALUE color FROM 'a/b'
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 'a/b': {"color": {"red": 255, "green": 0, "blue": 0}, "temperature": 50}
SQL: SELECT color.red as red_value FROM 'a/b'
Outgoing payload: {"red_value": 255}
```

您可以使用函数 (参阅 [函数 \(p. 171\)](#)) 来转换传入负载。可使用圆括号进行分组。例如：

```
Incoming payload published on topic 'a/b': {"color": "red", "temperature": 50}
SQL: SELECT (temperature - 32) * 5 / 9 AS celsius, upper(color) as my_color FROM 'a/b'
Outgoing payload: {"celsius": 10, "my_color": "RED"}
```

使用二进制负载

当消息负载应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。

必须遵循以下规则使用 * 将消息负载作为原始二进制数据引用：

1. SQL 语句和模板不得引用除 * 之外的 JSON 名称。
2. SELECT 语句必须具有 * 作为唯一项目，或者必须只具有函数，例如：

```
SELECT * FROM 'a/b'
```

```
SELECT encode(*, 'base64') AS data, timestamp() AS ts FROM 'a/b'
```

二进制负载示例

以下 SELECT 子句可与二进制负载配合使用，因为它未引用任何 JSON 名称。

```
SELECT * FROM 'a/b'
```

以下 SELECT 无法与二进制负载配合使用，因为它在 WHERE 子句中引用了 device_type。

```
SELECT * FROM 'a/b' WHERE device_type = 'thermostat'
```

以下 SELECT 无法与二进制负载配合使用，因为它违反了第 2 条规则。

```
SELECT *, timestamp() AS timestamp FROM 'a/b'
```

以下 SELECT 可以与二进制负载配合使用，因为它并未违反任意一条规则。

```
SELECT * FROM 'a/b' WHERE timestamp() % 12 = 0
```

以下 AWS IoT 规则无法与二进制负载配合使用，因为它违反了第 1 条规则。

```
{
    "sql": "SELECT * FROM 'a/b'"
    "actions": [
        {
            "republish": {
                "topic": "device/${device_id}"
            }
        }
    ]
}
```

FROM 子句

FROM 子句在主题或主题筛选条件中订阅您的规则。主题筛选条件可用于订阅一组类似的主题。

例如：

传入负载已发布至主题 'a/b' : {temperature: 50}

传入负载已发布至主题 'a/c' : {temperature: 50}

SQL: "SELECT temperature AS t FROM 'a/b'".

规则已订阅 'a/b'，因此，传入负载已传递至规则，并且传出负载(已传递给规则操作)为 : {t: 50}。规则未订阅 'a/c'，因此在 'a/c' 上发布的消息不会触发规则。

您可以使用 # 通配符来匹配主题筛选条件中的任何子路径：

例如：

传入负载已发布至主题 'a/b' : {temperature: 50}。

传入负载已发布至主题 'a/c' : {temperature: 60}。

传入负载已发布至主题 'a/e/f' : {temperature: 70}。

传入负载已发布至主题 'b/x' : {temperature: 80}。

SQL: "SELECT temperature AS t FROM 'a/#'".

规则已订阅以 'a' 开头的所有主题，因此它将执行三次，将 {t: 50} (for a/b), {t: 60} (适用于 a/c) 和 {t: 70} (适用于 a/e/f) 的传出负载发送至其操作。它未订阅 'b/x'，因此，{temperature: 80} 消息不会触发规则。

您可以使用 "+" 字符匹配任何一个特定的路径元素：

例如：

传入负载已发布至主题 'a/b' : {temperature: 50}。

传入负载已发布至主题 'a/c' : {temperature: 60}。

传入负载已发布至主题 'a/e/f' : {temperature: 70}。

传入负载已发布至主题 'b/x' : {temperature: 80}。

SQL: "SELECT temperature AS t FROM 'a/+'".

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 'a'。规则将对发送至 'a/b' 和 'a/c' 的消息执行，但不会对发送至 'a/e/f' 或 'b/x' 的消息执行。

您可以在 WHERE 子句中使用函数和运算符。在 WHERE 子句中，您无法引用在 SELECT 子句中通过 AS 关键字创建的任何别名。(首先评估 WHERE 语句，再确定是否评估 SELECT 子句。)

WHERE 子句

WHERE 子句确定在向已订阅规则的 MQTT 主题发送消息时是否评估规则。如果 WHERE 子句评估为 true，则将评估规则；否则不评估规则。

例如：

传入负载已发布至 a/b : {"color":"red", "temperature":40}。

SQL: SELECT color AS my_color FROM 'a/b' WHERE temperature > 50 AND color <> 'red'.

在本例中，不会评估规则，不存在传出负载，也不会触发规则操作。

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，再确定是否评估 SELECT。)

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 SQL 版本 2016-03-23 或更高版本中使用。

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 a/b : "{lat_long: [47.606,-122.332]}"

SQL 语句: SELECT {'latitude': get(lat_long, 0), 'longitude':get(lat_long, 1)} as lat_long FROM 'a/b'

生成的传出负载为 : {'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 a/b : {lat: 47.696, long: -122.332}

SQL 语句: SELECT [lat,long] as lat_long FROM 'a/b'

生成的传出负载为 : {"lat_long": [47.606,-122.332]}。

Case 语句

与 Switch 语句和 If/Else 语句类似，Case 语句可用于执行分支。

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 v 表达式，并与每个 t[i] 表达式进行相等匹配。如果找到匹配，相应的 r[i] 表达式会成为 Case 语句的结果。如果可能存在多个匹配，则选择第一个匹配。如果没有匹配，则 Else 语句的 re 将用作结果。如果没有匹配也没有 Else 语句，则 Case 语句的结果为 Undefined。例如：

传入负载已发布至主题 a/b : {"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 'a/b'

生成的传出负载为 : {"instructions": "caution"}。

Case 语句需要至少一个 WHEN 子句。不需要 ELSE 子句。

Note

如果 v 等于 Undefined，则 Case 语句的结果为 Undefined。

JSON 扩展

您可以使用以下 ANSI SQL 语法扩展，更加轻松地使用嵌套 JSON 对象。

“.”运算符

此运算符可访问与 ANSI SQL 和 JavaScript 相同的嵌入式 JSON 对象和函数的成员。例如：

```
SELECT foo.bar AS bar.baz FROM 'a/b'
```

* 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 'a/b'
```

将函数应用到属性值

下面显示了一个可由设备发布的 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补充规则已触发且 AWS IoT 执行操作时返回的 JSON 数据。替换模板的语法是 \${expression}，其中 expression 可以是 SELECT 或 WHERE 子句中 AWS IoT 支持的任意表达式。有关支持的表达式的更多信息，请参阅[AWS IoT SQL 参考 \(p. 162\)](#)。

替换模板显示在规则的 SELECT 子句中：

```
{  
    "sql": "SELECT *, topic() AS topic FROM 'my/iot/topic!',  
    "ruleDisabled": false,  
    "actions": [  
        {  
            "republish": {  
                "topic": "${topic()}/republish",  
                "roleArn": "arn:aws:iam::123456789012:role/my-iot-role"  
            }  
        }  
    ]  
}
```

如果该规则由以下 JSON 触发：

```
{  
    "deviceid" : "iot123",  
    "temp" : 54.98,  
    "humidity" : 32.43,  
    "coords" : {  
        "latitude" : 47.615694,  
        "longitude" : -122.3359976  
    }  
}
```

规则的输出内容如下：

```
{  
    "coords":{  
        "longitude": -122.3359976,  
        "latitude": 47.615694  
    },  
    "humidity": 32.43,  
    "temp": 54.98,  
    "deviceid": "iot123",  
    "topic": "my/iot/topic"  
}
```

适用于 AWS IoT 的事物影子

事物影子（有时称作设备影子）是一个 JSON 文档，用于存储和检索事物（设备、应用等）的当前状态信息。Thing Shadows 服务可以保持连接到 AWS IoT 的每件事物的事物影子。您可以使用事物影子通过 MQTT 或 HTTP 获取和设置事物的状态，无论该事物是否连接到 Internet。每个事物影子都使用其名称进行唯一标识。

内容

- [事物影子数据流 \(p. 205\)](#)
- [事物影子文档 \(p. 212\)](#)
- [使用事物影子 \(p. 215\)](#)
- [事物影子 RESTful API \(p. 224\)](#)
- [事物影子 MQTT 主题 \(p. 226\)](#)
- [事物影子文档语法 \(p. 232\)](#)
- [事物影子错误消息 \(p. 234\)](#)

事物影子数据流

Thing Shadows 服务充当中介，支持设备和应用程序检索和更新事物影子。

为了说明设备和应用程序如何与 Thing Shadows 服务进行通信，本部分将介绍如何使用 AWS IoT MQTT 客户端和 AWS CLI 模拟已连接 Internet 的电灯泡、应用程序和 Thing Shadows 服务之间的通信。

Thing Shadows 服务使用 MQTT 主题来促进应用程序和设备之间的通信。要了解相应运作方式，请使用 AWS IoT MQTT 客户端订阅以下 QoS 1 级 MQTT 主题：

\$aws/things/myLightBulb/shadow/update/accepted

当事物影子更新成功时，Thing Shadows 服务将向该主题发送消息。

\$aws/things/myLightBulb/shadow/update/rejected

当事物影子更新被拒绝时，Thing Shadows 服务将向该主题发送消息。

\$aws/things/myLightBulb/shadow/update/delta

当检测到事物影子的“reported”部分与“desired”部分之间存在差异时，Thing Shadows 服务将向该主题发送消息。有关更多信息，请参阅 [/update/delta \(p. 229\)](#)。

\$aws/things/myLightBulb/shadow/get/accepted

当获取事物影子的请求获批时，Thing Shadows 服务将向该主题发送消息。

\$aws/things/myLightBulb/shadow/get/rejected

当获取事物影子的请求被拒绝时，Thing Shadows 服务将向该主题发送消息。

\$aws/things/myLightBulb/shadow/delete/accepted

当事物影子被删除时，Thing Shadows 服务将向该主题发送消息。

\$aws/things/myLightBulb/shadow/delete/rejected

当删除事物影子的请求被拒绝时，Thing Shadows 服务将向该主题发送消息。

\$aws/things/myLightBulb/shadow/update/documents

每次事物影子更新成功执行时，Thing Shadows 服务都会向该主题发布状态文档。

要了解有关 Thing Shadows 服务使用的所有 MQTT 主题的更多信息，请参阅 [事物影子 MQTT 主题 \(p. 226\)](#)。

Note

建议您订阅 `.../rejected` 主题，以查看 Thing Shadows 服务发送的所有错误消息。

当电灯泡联机时，它将向 `$aws/things/myLightBulb/shadow/update` 主题发送 MQTT 消息，告知 Thing Shadows 服务其当前状态。

要模拟这一过程，请使用 AWS IoT MQTT 客户端向 `$aws/things/myLightbulb/shadow/update` 主题发布以下消息：

```
{  
  "state": {  
    "reported": {  
      "color": "red"  
    }  
  }  
}
```

此消息将灯泡的颜色设为红色 ("red")。

Thing Shadows 服务通过向 `$aws/things/myLightBulb/shadow/update/accepted` 主题发送以下消息进行响应：

```
{  
  "messageNumber": 4,  
  "payload": {  
    "state": {  
      "reported": {  
        "color": "red"  
      }  
    },  
    "metadata": {  
      "reported": {  
        "color": {  
          "timestamp": 1469564492  
        }  
      }  
    },  
    "version": 1,  
  }  
}
```

```

        "timestamp": 1469564492
    },
    "qos": 0,
    "timestamp": 1469564492848,
    "topic": "$aws/things/myLightBulb/shadow/update/accepted"
}

```

此消息表明 Thing Shadows 服务已收到 UPDATE 请求并更新了事物影子。如果事物影子不存在，Thing Shadows 服务则将创建事物影子；如果存在，则将使用消息中的数据更新事物影子。如果您发现没有任何消息发布至 \$aws/things/myLightBulb/shadow/update/accepted，请查看 \$aws/things/myLightBulb/shadow/update/rejected 订阅，了解该主题下是否有任何错误消息。

此外，Thing Shadows 服务还向 \$aws/things/myLightBulb/shadow/update/documents 主题发布以下消息。

```

{
    "previous": null,
    "current": {
        "state": {
            "reported": {
                "color": "red"
            }
        },
        "metadata": {
            "reported": {
                "color": {
                    "timestamp": 1483467764
                }
            }
        },
        "version": 1
    },
    "timestamp": 1483467764
}

```

每次事物影子更新成功执行时，都会向 /update/documents 主题发布消息。有关发布到此主题的消息内容的更多信息，请参阅[事物影子 MQTT 主题 \(p. 226\)](#)。

与电灯泡交互的应用程序联机并请求获取电灯泡的当前状态。该应用程序将向 \$aws/things/myLightBulb/shadow/get 主题发送一条空消息。要模拟这一过程，请使用 AWS IoT MQTT 客户端向 \$aws/things/myLightBulb/shadow/get 主题发布一条空消息("")。

Thing Shadows 服务通过向 \$aws/things/myLightBulb/shadow/get/accepted 主题发布请求获取的事物影子进行响应：

```

{
    "messageNumber": 1,
    "payload": {
        "state": {
            "reported": {
                "color": "red"
            }
        },
        "metadata": {
            "reported": {
                "color": {
                    "timestamp": 1469564492
                }
            }
        }
    },
    "version": 1,
    "timestamp": 1469564571
}

```

```
    },
    "qos": 0,
    "timestamp": 1469564571533,
    "topic": "$aws/things/myLightBulb/shadow/get/accepted"
}
```

如果您在 `$aws/things/myLightBulb/shadow/get/accepted` 主题下没有看到任何消息，请查看 `$aws/things/myLightBulb/shadow/get/rejected` 主题，了解该主题下是否有任何错误消息。

应用程序向用户显示此信息，用户请求更改电灯泡的颜色（将红色更改为绿色）。为此，该应用程序将向 `$aws/things/myLightBulb/shadow/update` 主题发布消息：

```
{
  "state": {
    "desired": {
      "color": "green"
    }
  }
}
```

要模拟这一过程，请使用 AWS IoT MQTT 客户端向 `$aws/things/myLightBulb/shadow/update` 主题发布之前的消息。

Thing Shadows 服务通过向 `$aws/things/myLightBulb/shadow/update/accepted` 主题发送以下消息：

```
{
  "messageNumber": 5,
  "payload": {
    "state": {
      "desired": {
        "color": "green"
      }
    },
    "metadata": {
      "desired": {
        "color": {
          "timestamp": 1469564658
        }
      }
    },
    "version": 2,
    "timestamp": 1469564658
  },
  "qos": 0,
  "timestamp": 1469564658286,
  "topic": "$aws/things/myLightBulb/shadow/update/accepted"
}
```

并向 `$aws/things/myLightBulb/shadow/update/delta` 主题发送以下消息进行响应：

```
{
  "messageNumber": 1,
  "payload": {
    "version": 2,
    "timestamp": 1469564658,
    "state": {
      "color": "green"
    },
    "metadata": {
      "color": {

```

```

        "timestamp": 1469564658
    }
},
"qos": 0,
"timestamp": 1469564658309,
"topic": "$aws/things/myLightBulb/shadow/update/delta"
}

```

当主题接受事物影子更新并且生成的事物影子包含不同的预期状态和报告状态值时，Thing Shadow 服务向该主题发布消息。

Thing Shadow 服务还向 \$aws/things/myLightBulb/shadow/update/documents 主题发布消息：

```

{
  "previous":{
    "state":{
      "reported":{
        "color":"red"
      }
    },
    "metadata":{
      "reported":{
        "color":{
          "timestamp":1483467764
        }
      }
    },
    "version":1
  },
  "current":{
    "state":{
      "desired":{
        "color":"green"
      },
      "reported":{
        "color":"red"
      }
    },
    "metadata":{
      "desired":{
        "color":{
          "timestamp":1483468612
        }
      },
      "reported":{
        "color":{
          "timestamp":1483467764
        }
      }
    },
    "version":2
  },
  "timestamp":1483468612
}

```

电灯泡已订阅 \$aws/things/myLightBulb/shadow/update/delta 主题，因此可以接收到消息、更改其颜色并发布新状态。要模拟这一过程，请使用 AWS IoT MQTT 客户端向 \$aws/things/myLightbulb/shadow/update 主题发布以下消息，以更新影子状态：

```
{
  "state":{
    "reported":{

    }
  }
}
```

```
        "color": "green"
    },
    "desired": null
}
}
```

Thing Shadows 服务通过向 `$aws/things/myLightBulb/shadow/update/accepted` 主题发送消息进行响应：

```
{
    "messageNumber": 6,
    "payload": {
        "state": {
            "reported": {
                "color": "green"
            },
            "desired": null
        },
        "metadata": {
            "reported": {
                "color": {
                    "timestamp": 1469564801
                }
            },
            "desired": {
                "timestamp": 1469564801
            }
        },
        "version": 3,
        "timestamp": 1469564801
    },
    "qos": 0,
    "timestamp": 1469564801673,
    "topic": "$aws/things/myLightBulb/shadow/update/accepted"
}
```

并向 `$aws/things/myLightBulb/shadow/update/documents` 主题发送以下消息进行响应：

```
{
    "previous": {
        "state": {
            "reported": {
                "color": "red"
            }
        },
        "metadata": {
            "reported": {
                "color": {
                    "timestamp": 1483470355
                }
            }
        },
        "version": 3
    },
    "current": {
        "state": {
            "reported": {
                "color": "green"
            }
        },
        "metadata": {
            "reported": {
                "color": {

```

```
        "timestamp":1483470364
    }
}
},
"version":4
},
"timestamp":1483470364
}
```

应用程序向 Thing Shadows 服务请求获取当前状态并显示最新状态数据。要模拟这一过程，请运行以下命令：

```
aws iot-data get-thing-shadow --thing-name "myLightBulb" "output.txt" && cat "output.txt"
```

Note

在 Windows 上，请忽略 `&& cat "output.txt"` 命令，该命令用于向控制台显示 `output.txt` 内容。您可以在记事本或任何文本编辑器中打开该文件，以查看事物影子的内容。

Thing Shadows 服务将返回事物影子文档：

```
{
  "state": {
    "reported": {
      "color": "green"
    }
  },
  "metadata": {
    "reported": {
      "color": {
        "timestamp": 1469564801
      }
    }
  },
  "version": 3,
  "timestamp": 1469564864
}
```

要删除事物影子，请将一条空消息发布到 `$aws/things/myLightBulb/shadow/delete` 主题。AWS IoT 将通过向 `$aws/things/myLightBulb/shadow/delete/accepted` 主题发布消息进行响应：

```
{
  "version" : 1,
  "timestamp" : 1488565234
}
```

检测事物是否连接

要确定设备当前是否已连接，请在事物影子中添加一个已连接设置，然后使用 MQTT Last Will and Testament (LWT) 消息，该消息将在设备由于错误而断开连接时将已连接设置设为 `false`。

Note

目前，AWS IoT Shadows 服务将忽略发送至 AWS IoT 预留主题（以 \$ 开头的主题）的 LWT 消息，但订阅的客户端和 AWS IoT 规则引擎仍会处理此类消息。如果您希望 AWS IoT Shadows 服务接收 LWT 消息，请将一条 LWT 消息注册为非预留主题，然后创建一条在预留的主题中重新发布消息的规则。以下示例将展示如何创建重新发布规则，以侦听 `my/things/myLightBulb/update` 主题发布的消息并将其重新发布到 `$aws/things/myLightBulb/shadow/update` 主题。

```
{  
    "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"  
            }  
        }]  
    }  
}
```

当设备成功连接后，它将注册一条 LWT 消息并将已连接设置为 `false`：

```
{  
    "state": {  
        "reported": {  
            {  
                "connected": "false"  
            }  
        }  
    }  
}
```

它还会在其更新主题 (`$aws/things/myLightBulb/shadow/update`) 下发布一条消息，以将其连接状态设置为 `true`：

```
{  
    "state": {  
        "reported": {  
            {  
                "connected": "true"  
            }  
        }  
    }  
}
```

当设备正常断开连接时，它将在其更新主题下发布消息并将其连接状态设置为 `false`：

```
{  
    "state": {  
        "reported": {  
            {  
                "connected": "false"  
            }  
        }  
    }  
}
```

如果设备由于错误而断开连接，其 LWT 消息将自动发布到更新主题。

事物影子文档

Thing Shadows 服务遵守 JSON 规范下的所有规则。值、对象和数组均存储在事物影子文档中。

内容

- [文档属性 \(p. 213\)](#)
- [事物影子的版本控制 \(p. 213\)](#)

- 客户端令牌 (p. 213)
- 示例文档 (p. 214)
- 空白部分 (p. 214)
- 数组 (p. 215)

文档属性

事物影子文档具有以下属性：

`state`

`desired`

事物的预期状态。应用程序可以向本文档部分写入数据来更新事物的状态，且无需直接连接到该事物。

`reported`

事物的报告状态。事物可以向本文档部分写入数据，以报告其新状态。应用程序可以读取本文档部分，以确定事物的状态。

`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。

`timestamp`

指明 AWS IoT 传输消息的时间。通过在消息中使用时间戳并对 `desired` 或 `reported` 部分中的不同属性使用时间戳，事物可以确定已更新项目的存在时间，即使它不具有内部时钟特性。

`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 [事物影子文档语法 \(p. 232\)](#)。

事物影子的版本控制

Thing Shadows 服务支持对每个更新消息（包括请求和响应）实施版本控制，这意味着，事物影子每次更新时，JSON 文档的版本都会递增。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首先进行重新同步，然后再更新事物影子。
- 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。

在有些情况下，客户端可以通过不提交版本来绕过版本匹配。

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。

示例文档

以下为事物影子文档示例：

```
{  
    "state" : {  
        "desired" : {  
            "color" : "RED",  
            "sequence" : [ "RED", "GREEN", "BLUE" ]  
        },  
        "reported" : {  
            "color" : "GREEN"  
        }  
    },  
    "metadata" : {  
        "desired" : {  
            "color" : {  
                "timestamp" : 12345  
            },  
            "sequence" : {  
                "timestamp" : 12345  
            }  
        },  
        "reported" : {  
            "color" : {  
                "timestamp" : 12345  
            }  
        }  
    },  
    "version" : 10,  
    "clientToken" : "UniqueClientToken",  
    "timestamp": 123456789  
}
```

空白部分

仅当事物影子文档具有预期状态时，它才包含 `desired` 部分。例如，以下文档为没有 `desired` 部分的有效状态文档：

```
{  
    "reported" : { "temp": 55 }  
}
```

`reported` 部分也可以为空：

```
{  
    "desired" : { "color" : "RED" }  
}
```

如果更新导致 `desired` 或 `reported` 部分为空，则该部分将从文档中删除。要从文档中删除 `desired` 部分（例如，对设备更新状态做出响应），请将“`desired`”部分设置为 `null`：

```
{  
    "state": {  
        "reported": {  
            "color": "red"  
        },  
        "desired": null  
    }  
}
```

```
}
```

事物影子文档也有可能不包含 `desired` 或 `reported` 部分。在此情况下，影子文档将为空。例如，以下文档也是一个有效文档：

```
{  
}
```

数组

事物影子支持数组，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" ]  
    }  
}
```

使用事物影子

AWS IoT 针对事物影子提供了三项操作：

UPDATE

如果事物影子不存在，则创建一个事物影子；如果存在，则使用请求中提供的数据更新事物影子的内容。存储数据时使用时间戳信息，以指明最新更新时间。向所有订阅者发送消息，告知 `desired` 状态与 `reported` 状态之间的差异（增量）。接收到消息的事物或应用程序可以根据 `desired` 状态和 `reported` 状态之间的差异执行操作。例如，设备可将其状态更新为预期状态，或者应用程序可以更新其 UI，以反映设备状态的更改。

GET

检索事物影子中存储的最新状态（例如，在设备启动期间，检索配置和最新操作状态）。此操作将返回整个 JSON 文档，其中包括元数据。

DELETE

删除事物影子，包括其所有内容。这将从数据存储中删除 JSON 文档。您无法还原已删除的事物影子，但可以创建具有相同名称的新事物影子。

协议支持

MQTT 和通过 HTTPS 的 RESTful API 都支持这些操作。由于 MQTT 是一种发布/订阅的通信模式，AWS IoT 将采用一组预留主题。为了实施请求 – 响应操作，事物或应用程序应先订阅这些主题，然后向请求主题发布消息。有关更多信息，请参阅 [事物影子 MQTT 主题 \(p. 226\)](#) 和 [事物影子 RESTful API \(p. 224\)](#)。

更新事物影子

您可以使用 [UpdateThingShadow \(p. 225\)](#) RESTful API 或发布消息到 [/更新 \(p. 226\)](#) 主题来更新事物影子。更新仅影响请求中指定的字段。

初始状态：

```
{  
  "state": {  
    "reported" : {  
      "color" : { "r" :255, "g": 255, "b": 0 }  
    }  
  }  
}
```

发送一条更新信息：

```
{  
  "state": {  
    "desired" : {  
      "color" : { "r" : 10 },  
      "engine" : "ON"  
    }  
  }  
}
```

设备收到由之前的 /update 消息触发的 desired 状态 (位于 /update/delta 主题下)，然后执行预期更改。结束后，设备应通过事物影子 JSON 文档中的 reported 部分确认其更新后的状态。

最终状态：

```
{  
  "state": {  
    "reported" : {  
      "color" : { "r" : 10, "g" : 255, "b": 0 },  
      "engine" : "ON"  
    }  
  }  
}
```

检索事物影子文档

您可以使用 [GetThingShadow \(p. 224\)](#) RESTful API 或订阅和发布消息到 [/get \(p. 229\)](#) 主题来检索事物影子。这将检索整个文档以及 desired 状态与 reported 状态之间的增量。

示例文档：

```
{  
    "state": {  
        "desired": {  
            "lights": {  
                "color": "RED"  
            },  
            "engine": "ON"  
        },  
        "reported": {  
            "lights": {  
                "color": "GREEN"  
            },  
            "engine": "ON"  
        }  
    },  
    "metadata": {  
        "desired": {  
            "lights": {  
                "color": {  
                    "timestamp": 123456  
                },  
                "engine": {  
                    "timestamp": 123456  
                }  
            }  
        },  
        "reported": {  
            "lights": {  
                "color": {  
                    "timestamp": 789012  
                }  
            },  
            "engine": {  
                "timestamp": 789012  
            }  
        },  
        "version": 10,  
        "timestamp": 123456789  
    }  
}
```

响应：

```
{  
    "state": {  
        "desired": {  
            "lights": {  
                "color": "RED"  
            },  
            "engine": "ON"  
        },  
        "reported": {  
            "lights": {  
                "color": "GREEN"  
            },  
            "engine": "ON"  
        },  
        "delta": {  
            "lights": {  
                "color": "RED"  
            }  
        }  
    }  
}
```

```
{
    },
    "metadata": {
        "desired": {
            "lights": {
                "color": {
                    "timestamp": 123456
                }
            }
        }
    },
    "reported": {
        "lights": {
            "color": {
                "timestamp": 789012
            }
        },
        "engine": {
            "timestamp": 789012
        }
    },
    "delta": {
        "lights": {
            "color": {
                "timestamp": 123456
            }
        }
    }
},
"version": 10,
"timestamp": 123456789
}
```

乐观锁

您可以使用状态文档版本来确保正在更新的事物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。

例如：

初始文档：

```
{
    "state" : {
        "desired" : { "colors" : [ "RED", "GREEN", "BLUE" ] }
    },
    "version" : 10
}
```

更新：(版本不匹配；请求将被拒绝)

```
{
    "state": {
        "desired": {
            "colors": [
                "BLUE"
            ]
        }
    },
    "version": 9
}
```

结果：

```
409 Conflict
```

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
}
```

删除数据

您可以通过将消息发布到[/更新 \(p. 226\)](#) 主题并将要删除的字段设置为空，以从事物影子中删除数据。值为 null 的所有字段均将从文档中删除。

初始状态：

```
{
  "state": {
    "desired" : {
      "lights": { "color": "RED" },
      "engine" : "ON"
    },
    "reported" : {
      "lights" : { "color": "GREEN" },
      "engine" : "OFF"
    }
  }
}
```

发送一条更新信息：

```
{
  "state": {
    "desired": null,
    "reported": {
      "engine": null
    }
  }
}
```

```
}
```

最终状态：

```
{
  "state": {
    "reported" : {
      "lights" : { "color" : "GREEN" }
    }
  }
}
```

您可以将事物影子的状态设置为 `null`，从而从中删除所有数据。例如，发送以下消息将删除所有状态数据，但仍将保留事物影子。

```
{
  "state": null
}
```

即使状态为 `null`，事物影子仍然存在。在下次更新时，事物影子的版本将递增。

删除事物影子

您可以使用 [DeleteThingShadow \(p. 225\)](#) RESTful API 或发布消息到 [/delete \(p. 231\)](#) 主题来删除事物影子文档。

Note

删除事物影子不会删除事物，而删除事物也不会删除事物影子。

初始状态：

```
{
  "state": {
    "desired" : {
      "lights": { "color": "RED" },
      "engine" : "ON"
    },
    "reported" : {
      "lights" : { "color": "GREEN" },
      "engine" : "OFF"
    }
  }
}
```

在 `/delete` 主题下发布了空消息。

最终状态：

```
HTTP 404 - resource not found
```

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
}
```

Thing Shadows 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。

观察状态更改

事物影子更新后，将在两个 MQTT 主题下发布消息：

- `$aws/things/thing-name/shadow/update/accepted`
- `$aws/things/thing-name/shadow/update/delta`

发送到 `update/delta` 主题的消息将用于状态正在更新的事物。此消息仅包含事物影子文档中 `desired` 部分与 `reported` 部分之间的差异。接收到此消息后，事物将立即决定是否要执行请求的更改。如果事物的状态发生变化，则它会向 `$aws/things/thing-name/shadow/update` 主题发布其当前最新状态。

设备和应用程序可以订阅任一主题，以便在文档状态更改后收到通知。

以下是该流程的示例：

1. 设备报告状态。
2. 系统在其持久性数据存储中更新状态文档。
3. 系统发布仅包含增量并面向订阅设备的增量消息。要接收更新，设备应订阅此主题。
4. 事物影子发布接受的消息，包括内含元数据的完整已接收文档。要接收更新，应用程序应订阅此主题。

消息顺序

不保证 AWS IoT 服务发送的消息按任何特定顺序到达设备。

初始状态文档：

```
{
    "state" : {
        "reported" : { "color" : "blue" }
    },
    "version" : 10,
    "timestamp": 123456777
}
```

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。

修剪事物影子消息

要降低发送到您的设备的事物影子消息的大小，请定义一项规则，以仅选择设备所需的字段并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(2)}/delta",  
            "qos": 1  
        }  
    }]  
}
```

```
        "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事物影子名称。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(2) 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[规则](#)。

事物影子 RESTful API

事物影子将显示以下 URI，以更新状态信息：

```
https://endpoint/things/thingName/shadow
```

您的 AWS 账户具有特定的终端节点。要检索您的终端节点，请使用 [describe-endpoint](#) 命令。终端节点的格式如下：

```
identifier.iot.region.amazonaws.com
```

API 操作

- [GetThingShadow \(p. 224\)](#)
- [UpdateThingShadow \(p. 225\)](#)
- [DeleteThingShadow \(p. 225\)](#)

GetThingShadow

获取指定事物的事物影子。

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
```

响应

请求成功后，响应将包括标准的 HTTP 标头以及以下代码和正文：

```
HTTP 200
BODY: response state document
```

有关更多信息，请参阅[响应状态文档示例 \(p. 233\)](#)。

授权

要检索事物影子，需要一项允许发起人执行 iot:GetThingShadow 操作的策略。Thing Shadows 服务接受两种形式的身份验证：使用 IAM 凭证的签名版本 4 或使用客户端证书的 TLS 双向身份验证。

以下是允许发起人检索事物影子的示例策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Action": "iot:GetThingShadow",  
        "Resource": ["arn:aws:iot:region:account:thing/thing"]  
    }]  
}
```

UpdateThingShadow

更新指定事物的事物影子。

更新仅影响请求状态文档中指定的字段。值为 `null` 的所有字段均将从事物影子中删除。

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  
BODY: request state document
```

有关更多信息，请参阅[请求状态文档示例 \(p. 232\)](#)。

响应

请求成功后，响应将包括标准的 HTTP 标头以及以下代码和正文：

```
HTTP 200  
BODY: response state document
```

有关更多信息，请参阅[响应状态文档示例 \(p. 233\)](#)。

授权

要更新事物影子，需要一项允许发起人执行 `iot:UpdateThingShadow` 操作的策略。Thing Shadows 服务接受两种形式的身份验证：使用 IAM 凭证的签名版本 4 或使用客户端证书的 TLS 双向身份验证。

以下是允许发起人更新事物影子的示例策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [{  
        "Effect": "Allow",  
        "Action": "iot:UpdateThingShadow",  
        "Resource": ["arn:aws:iot:region:account:thing/thing"]  
    }]  
}
```

DeleteThingShadow

删除指定事物的事物影子。

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
```

响应

请求成功后，响应将包括标准的 HTTP 标头以及以下代码和正文：

```
HTTP 200
BODY: Empty response state document
```

授权

要删除事物影子，需要一项允许发起人执行 `iot:DeleteThingShadow` 操作的策略。Thing Shadows 服务接受两种形式的身份验证：使用 IAM 凭证的签名版本 4 或使用客户端证书的 TLS 双向身份验证。

以下是允许发起人删除事物影子的示例策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:DeleteThingShadow",
            "Resource": ["arn:aws:iot:region:account:thing/thing"]
        }
    ]
}
```

事物影子 MQTT 主题

Thing Shadows 服务使用预留的 MQTT 主题，使应用程序和事物获取、更新或删除事物（事物影子）的状态信息。这些主题的名称以 `$aws/things/thingName/shadow` 开头。要订阅事物影子主题并向主题发布消息，需要获得基于主题的授权。AWS IoT 保留向现有主题结构添加新主题的权利。为此，建议您订阅影子主题时勿使用通配符。例如，请勿在订阅时使用诸如 `$aws/things/thingName/shadow/#` 的主题筛选条件，因为与该主题筛选条件匹配的主题数量可能会随着 AWS IoT 引入新的影子主题而增加。要了解针对这些主题发布的消息示例，请查看 [事物影子数据流 \(p. 205\)](#)。

以下是用于与事物影子交互的 MQTT 主题。

主题：

- [/更新 \(p. 226\)](#)
- [/update/accepted \(p. 227\)](#)
- [/update/documents \(p. 228\)](#)
- [/update/rejected \(p. 228\)](#)
- [/update/delta \(p. 229\)](#)
- [/get \(p. 229\)](#)
- [/get/accepted \(p. 230\)](#)
- [/get/rejected \(p. 230\)](#)
- [/delete \(p. 231\)](#)
- [/delete/accepted \(p. 231\)](#)
- [/delete/rejected \(p. 231\)](#)

/更新

向该主题发布请求状态文档来更新事物影子：

```
$aws/things/thingName/shadow/update
```

尝试更新事物状态的客户端会发送类似如下的 JSON 请求状态文档：

```
{  
    "state" : {  
        "desired" : {  
            "color" : "red",  
            "power" : "on"  
        }  
    }  
}
```

更新其事物影子的事物会发送类似如下的 JSON 请求状态文档：

```
{  
    "state" : {  
        "reported" : {  
            "color" : "red",  
            "power" : "on"  
        }  
    }  
}
```

AWS IoT 通过向 [/update/accepted \(p. 227\)](#) 或 [/update/rejected \(p. 228\)](#) 发布消息来进行响应。

有关更多信息，请参阅 [请求状态文档 \(p. 232\)](#)。

示例策略

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/accepted

AWS IoT 在接受事物影子的更改时向该主题发布响应状态文档：

```
$aws/things/thingName/shadow/update/accepted
```

有关更多信息，请参阅 [响应状态文档 \(p. 233\)](#)。

示例策略

以下是所需策略的示例：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot:Publish",  
                "iot:UpdateShadow"  
            ]  
        }  
    ]  
}
```

```
        "iot:Subscribe",
        "iot:Receive"
    ],
    "Resource": ["arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/
update/accepted"]
}
}
```

/update/documents

每次影子更新成功执行时，AWS IoT 都会向该主题发布状态文档：

```
$aws/things/thingName/shadow/update/documents
```

JSON 文档包含两个主要节点：`previous` 和 `current`。`previous` 节点包含执行更新之前完整影子文档的内容，而 `current` 节点则包含成功更新之后完整影子文档的内容。首次更新（创建）事物影子时，`previous` 节点将包含 `null`。

示例策略

以下是所需策略的示例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe",
                "iot:Receive"
            ],
            "Resource": ["arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/
update/documents"]
        }
    ]
}
```

/update/rejected

AWS IoT 在拒绝事物影子的更改时向该主题发布错误响应文档：

```
$aws/things/thingName/shadow/update/rejected
```

有关更多信息，请参阅 [错误响应文档 \(p. 234\)](#)。

示例策略

以下是所需策略的示例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe",
                "iot:Receive"
            ],
            "Resource": ["arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/
update/rejected"]
        }
    ]
}
```

```
    }]
}
```

/update/delta

AWS IoT 在接受事物影子的更改时向该主题发布响应状态文档，且请求状态文档包含不同的 `desired` 状态和 `reported` 状态值：

```
$aws/things/thingName/shadow/update/delta
```

有关更多信息，请参阅 [响应状态文档 \(p. 233\)](#)。

发布详情

- 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。它将包含所有此类属性，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。`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。

示例策略

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe",
        "iot:Receive"
      ],
      "Resource": ["arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/update/delta"]
    }
  ]
}
```

/get

要获得事物影子，请在该主题下发布一条空消息：

```
$aws/things/thingName/shadow/get
```

AWS IoT 通过向 [/get/accepted \(p. 230\)](#) 或 [/get/rejected \(p. 230\)](#) 发布消息来进行响应。

示例策略

以下是所需策略的示例：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Subscribe",
        "iot:Receive"
      ],
      "Resource": ["arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/get"]
    }
  ]
}
```

```
    "Effect": "Allow",
    "Action": [
        "iot:Publish"
    ],
    "Resource": ["arn:aws:iot:region:account:topic/$aws/things/thingName/shadow/get"]
}
}
```

/get/accepted

AWS IoT 在返回事物影子时向该主题发布响应状态文档：

```
$aws/things/thingName/shadow/get/accepted
```

有关更多信息，请参阅 [响应状态文档 \(p. 233\)](#)。

示例策略

以下是所需策略的示例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe",
                "iot:Receive"
            ],
            "Resource": ["arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/get/accepted"]
        }
    ]
}
```

/get/rejected

AWS IoT 在无法返回事物影子时向该主题发布错误响应文档：

```
$aws/things/thingName/shadow/get/rejected
```

有关更多信息，请参阅 [错误响应文档 \(p. 234\)](#)。

示例策略

以下是所需策略的示例：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "iot:Subscribe",
                "iot:Receive"
            ],
            "Resource": ["arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/get/rejected"]
        }
    ]
}
```

/delete

要删除事物影子，请在“删除”主题下发布一条空消息：

```
$aws/things/thingName/shadow/delete
```

消息内容将被忽略。

AWS IoT 通过向 [/delete/accepted \(p. 231\)](#) 或 [/delete/rejected \(p. 231\)](#) 发布消息来进行响应。

示例策略

以下是所需策略的示例：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot:Subscribe",  
                "iot:Receive"  
            ],  
            "Resource": ["arn:aws:iot:region:account:topic filter/$aws/things/thingName/shadow/  
delete"]  
        }  
    ]  
}
```

/delete/accepted

AWS IoT 在删除事物影子时在该主题下发布消息：

```
$aws/things/thingName/shadow/delete/accepted
```

示例策略

以下是所需策略的示例：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot:Subscribe",  
                "iot:Receive"  
            ],  
            "Resource": ["arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/  
delete/accepted"]  
        }  
    ]  
}
```

/delete/rejected

AWS IoT 在无法删除事物影子时向该主题发布错误响应文档：

```
$aws/things/thingName/shadow/delete/rejected
```

有关更多信息，请参阅 [错误响应文档 \(p. 234\)](#)。

示例策略

以下是所需策略的示例：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iot:Subscribe",  
                "iot:Receive"  
            ],  
            "Resource": ["arn:aws:iot:region:account:topicfilter/$aws/things/thingName/shadow/  
delete/rejected"]  
        }  
    ]  
}
```

事物影子文档语法

在使用 [RESTful API \(p. 224\)](#) 或 [MQTT 发布/订阅消息 \(p. 226\)](#) 的 UPDATE、GET 和 DELETE 操作中，Thing Shadows 服务使用以下文档。有关更多信息，请参阅 [事物影子文档 \(p. 212\)](#)。

示例

- [请求状态文档 \(p. 232\)](#)
- [响应状态文档 \(p. 233\)](#)
- [错误响应文档 \(p. 234\)](#)

请求状态文档

请求状态文档具有以下格式：

```
{  
    "state": {  
        "desired": {  
            "attribute1integer2,  
            "attribute2string2",  
            ...  
            "attributeNboolean2  
        },  
        "reported": {  
            "attribute1integer1,  
            "attribute2string1",  
            ...  
            "attributeNboolean1  
        }  
    }  
    "clientToken": "token",  
    "version": version  
}
```

- **state** – 更新仅影响指定字段。
- **clientToken** – 如果使用，您可以验证请求和响应是否包含相同的客户端令牌。

- **version** – 如果使用，仅当指定的版本与其拥有的最新版本相符时，Thing Shadows 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：

```
{  
    "state": {  
        "desired": {  
            "attribute1": "integer2",  
            "attribute2": "string2",  
            ...  
            "attributeN": "boolean2"  
        },  
        "reported": {  
            "attribute1": "integer1",  
            "attribute2": "string1",  
            ...  
            "attributeN": "boolean1"  
        },  
        "delta": {  
            "attribute3": "integerX",  
            "attribute5": "stringY"  
        }  
    },  
    "metadata": {  
        "desired": {  
            "attribute1": {  
                "timestamp": "timestamp"  
            },  
            "attribute2": {  
                "timestamp": "timestamp"  
            },  
            ...  
            "attributeN": {  
                "timestamp": "timestamp"  
            }  
        },  
        "reported": {  
            "attribute1": {  
                "timestamp": "timestamp"  
            },  
            "attribute2": {  
                "timestamp": "timestamp"  
            },  
            ...  
            "attributeN": {  
                "timestamp": "timestamp"  
            }  
        }  
    },  
    "timestamp": "timestamp",  
    "clientToken": "token",  
    "version": "version"  
}
```

- **state**
 - **reported** – 仅当事物报告了 reported 部分的任何数据时存在，且仅包含请求状态文档中的字段。
 - **desired** – 仅当事物报告了 desired 部分的任何数据时存在，且仅包含请求状态文档中的字段。
- **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` – AWS IoT 生成响应时的 Epoch 日期和时间。
- `clientToken` — 仅当客户端令牌用于向 `/update` 主题发布有效 JSON 时存在。
- `version` – AWS IoT 中共享的事物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` – AWS IoT 生成响应时的日期和时间。
- `clientToken` — 仅当客户端令牌用于向 `/update` 主题发布有效 JSON 时存在。

有关更多信息，请参阅 [事物影子错误消息 \(p. 234\)](#)。

事物影子错误消息

Thing Shadows 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预留 `$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)	<ul style="list-style-type: none">• 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• 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)	<ul style="list-style-type: none">• 未授权
403 (禁止访问)	<ul style="list-style-type: none">• 禁止
404 (未找到)	<ul style="list-style-type: none">• 事物未找到
409 (冲突)	<ul style="list-style-type: none">• 版本冲突
413 (有效负载过大)	<ul style="list-style-type: none">• 有效负载超出允许的最大值
415 (媒体类型不受支持)	<ul style="list-style-type: none">• 文档编码不受支持；受支持的编码是 UTF-8

HTTP 错误代码	错误消息
429 (请求过多)	<ul style="list-style-type: none">如果正在传输的请求超过 10 个，则 Thing Shadows 服务将生成此条错误消息。
500 (内部服务器错误)	<ul style="list-style-type: none">内部服务故障

AWS IoT 软件开发工具包

内容

- [适用于 Android 的 AWS 移动软件开发工具包 \(p. 236\)](#)
- [Arduino Yún 软件开发工具包 \(p. 236\)](#)
- [适用于嵌入式 C 的 AWS IoT 设备软件开发工具包 \(p. 237\)](#)
- [适用于 iOS 的 AWS 移动软件开发工具包 \(p. 237\)](#)
- [适用于 Java 的 AWS IoT 设备软件开发工具包 \(p. 237\)](#)
- [适用于 JavaScript 的 AWS IoT 设备软件开发工具包 \(p. 237\)](#)
- [适用于 Python 的 AWS IoT 设备软件开发工具包 \(p. 237\)](#)

AWS IoT 设备软件开发工具包帮助您快速轻松地将设备连接至 AWS IoT。AWS IoT 设备软件开发工具包包括开源库、开发人员指南 (含示例) 和移植指南，便于您在自己选择的硬件平台上构建富有创新精神的 IoT 产品或解决方案。

适用于 Android 的 AWS 移动软件开发工具包

适用于 Android 的 AWS 软件开发工具包包含为开发人员提供的库、示例和文档，以便于他们使用 AWS 构建联网的移动应用程序。此软件开发工具包也支持调用 AWS IoT API。有关更多信息，请参阅下列内容：

- [GitHub 上适用于 Android 的 AWS 移动软件开发工具包](#)
- [适用于 Android 的 AWS 移动软件开发工具包自述文件](#)
- [适用于 Android 的 AWS 移动软件开发工具包示例](#)

Arduino Yún 软件开发工具包

开发人员可以使用 AWS IoT Arduino Yún 软件开发工具包将与 Arduino Yún 兼容的设备连接到 AWS IoT。通过将设备连接到 AWS IoT，用户可以安全地使用由 AWS IoT 提供的消息代理、规则和事物影子，以及 AWS Lambda、Kinesis 和 Amazon S3 等其他 AWS 服务。有关更多信息，请参阅下列内容：

- [GitHub 上的 Arduino Yún 软件开发工具包](#)
- [Arduino Yún 软件开发工具包自述文件](#)

适用于嵌入式 C 的 AWS IoT 设备软件开发工具包

适用于嵌入式 C 的 AWS IoT 设备软件开发工具包是 C 源文件的集合，可以在嵌入式应用程序中使用，以安全地连接到 AWS IoT 平台。它包括传输客户端、TLS 实施以及它们的使用示例。它还支持特定于 AWS IoT 的功能，例如，可访问 Thing Shadows 服务的 API。它以源代码的形式分发，旨在与应用程序代码、其他库和 RTOS 一起内置到客户的固件中。有关更多信息，请参阅下列内容：

- [适用于嵌入式 C 的 AWS IoT 设备软件开发工具包 GitHub](#)
- [适用于嵌入式 C 的 AWS IoT 设备软件开发工具包自述文件](#)
- [适用于嵌入式 C 的 AWS IoT 设备软件开发工具包移植指南](#)

适用于 iOS 的 AWS 移动软件开发工具包

适用于 iOS 的 AWS 软件开发工具包是开源的软件开发套件，依据 Apache 开源许可分发。适用于 iOS 的软件开发工具包为开发人员提供库、代码示例和文档，以便于他们使用 AWS 构建联网的移动应用程序。此软件开发工具包也支持调用 AWS IoT API。

- [GitHub 上适用于 iOS 的 AWS 软件开发工具包](#)
- [适用于 iOS 的 AWS 软件开发工具包自述文件](#)
- [适用于 iOS 的 AWS 软件开发工具包示例](#)

适用于 Java 的 AWS IoT 设备软件开发工具包

适用于 Java 的 AWS IoT 设备软件开发工具包让 Java 开发人员能够通过 MQTT 或基于 WebSocket 协议的 MQTT 来访问 AWS IoT 平台。该软件开发工具包内置了 AWS IoT 事物影子支持。您可以使用 HTTP 方法（包括 GET、UPDATE 和 DELETE）访问事物影子。该软件开发工具包还支持简化的事物影子访问模型，开发人员只需要使用 getter 和 setter 方法即可与事物影子交换数据，不需要对任何 JSON 文件进行序列化或反序列化。有关更多信息，请参阅下列内容：

- [GitHub 上适用于 Java 的 AWS IoT 设备软件开发工具包](#)
- [适用于 Java 的 AWS IoT 设备软件开发工具包自述文件](#)

适用于 JavaScript 的 AWS IoT 设备软件开发工具包

借助 aws-iot-device-sdk.js 软件包，开发人员可以编写使用 MQTT 或基于 WebSocket 协议的 MQTT 来访问 AWS IoT 的 JavaScript 应用程序。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：

- [GitHub 上适用于 JavaScript 的 AWS IoT 设备软件开发工具包](#)
- [适用于 JavaScript 的 AWS IoT 设备软件开发工具包自述文件](#)

适用于 Python 的 AWS IoT 设备软件开发工具包

借助适用于 Python 的 AWS IoT 设备软件开发工具包，开发人员可以编写 Python 脚本，以使用其设备通过 MQTT 或基于 WebSocket 协议的 MQTT 来访问 AWS IoT 平台。通过将设备连接到 AWS IoT，用户可以安全地使用由 AWS IoT 提供的消息代理、规则和事物影子，以及 AWS Lambda、Kinesis、Amazon S3 和更多其他 AWS 服务。

- GitHub 上适用于 Python 的 AWS IoT 设备软件开发工具包
- 适用于 Python 的 AWS IoT 设备软件开发工具包自述文件

监控 AWS IoT

监控是保持 AWS IoT 和 AWS 解决方案的可靠性、可用性和性能的重要环节。您应从 AWS 解决方案的各个部分收集监控数据，以便更轻松地调试出现的多点故障。开始监控 AWS IoT 之前，您应制定一个监控计划并在计划中回答下列问题：

- 您的监控目标是什么？
- 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？

接下来，通过在不同时间和不同负载条件下衡量性能，在您的环境中建立为正常的 AWS IoT 性能建立基准。监控 AWS IoT 时，将历史监控数据存储下来，以便可以将其与当前性能数据进行比较，确定正常性能模式和异常性能表现，并设计出问题解决方法。

例如，如果您使用的是 Amazon EC2，则可以监控实例的 CPU 利用率、磁盘 I/O 和网络利用率。如果性能低于您所建立的基准，则您可能需要重新配置或优化实例以降低 CPU 使用率、改进磁盘 I/O 或减少网络流量。

要建立基准，您至少应监控以下各项：

- PublishIn.Success
- PublishOut.Success
- Subscribe.Success
- Ping.Success
- Connect.Success
- GetThingShadow.Accepted
- UpdateThingShadow.Accepted
- DeleteThingShadow.Accepted
- RulesExecuted

主题

- [监控工具 \(p. 240\)](#)
- [使用 Amazon CloudWatch 进行监控 \(p. 241\)](#)
- [使用 AWS CloudTrail 记录 AWS IoT API 调用 \(p. 247\)](#)

监控工具

AWS 为您提供了各种可用于监控 AWS IoT 的工具。您可以配置其中的一些工具来为您执行监控任务，但有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化监控工具来监控 AWS IoT 并在出现错误时进行报告：

- Amazon CloudWatch Alarms – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see [使用 Amazon CloudWatch 进行监控 \(p. 241\)](#).
- Amazon CloudWatch Logs – Monitor, store, and access your log files from AWS CloudTrail or other sources. For more information, see [Monitoring Log Files](#) in the Amazon CloudWatch 用户指南.
- Amazon CloudWatch Events – Match events and route them to one or more target functions or streams to make changes, capture state information, and take corrective action. For more information, see [What is Amazon CloudWatch Events](#) in the Amazon CloudWatch 用户指南.
- AWS CloudTrail Log Monitoring – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see [Working with CloudTrail Log Files](#) in the AWS CloudTrail User Guide.

手动监控工具

监控 AWS IoT 时的另一个重要环节是手动监控 CloudWatch 警报未涵盖的那些项。AWS IoT、CloudWatch 和其他 AWS 控制台仪表板均提供 AWS 环境状态的概览视图。建议您还要查看 AWS IoT 上的日志文件。

- 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

此外，您还可以使用 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
- 搜索并浏览您所有的 AWS 资源指标
- 创建和编辑警报以接收有关问题的通知

使用 Amazon CloudWatch 进行监控

您可以使用 CloudWatch 监控 AWS IoT，此工具可从 AWS IoT 收集原始数据，并将数据处理为便于读取的近乎实时的指标。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 Web 应用程序或服务的执行情况。默认情况下，系统将在 1 分钟的时间段内自动将 AWS IoT 指标数据发送至 CloudWatch。有关更多信息，请参阅[什么是 Amazon CloudWatch、Amazon CloudWatch Events 和 Amazon CloudWatch Logs？](#)（位于 Amazon CloudWatch 用户指南 中）。

主题

- [AWS IoT 指标和维度 \(p. 241\)](#)
- [如何使用 AWS IoT 指标？\(p. 245\)](#)
- [创建 CloudWatch 警报以监控 AWS IoT \(p. 245\)](#)

AWS IoT 指标和维度

当您与 AWS IoT 交互时，它每一分钟向 CloudWatch 发送一次下列指标和维度。您可以按照以下步骤查看 AWS IoT 的各项指标。

使用 CloudWatch 控制台查看指标

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. 通过以下网址打开 CloudWatch 控制台：<https://console.aws.amazon.com/cloudwatch/>。
2. 在导航窗格中，选择 Metrics。
3. 在 CloudWatch Metrics by Category 窗格中，在 AWS IoT 的指标类别下，选择一个指标类别，然后在上方窗格中，向下滚动以查看完整的指标列表。

使用 AWS CLI 查看指标

- 在命令提示符处，输入以下命令：

```
aws cloudwatch list-metrics --namespace "AWS/IoT"
```

CloudWatch 会显示 AWS IoT 的以下指标：

AWS IoT Metrics

AWS IoT sends the following metrics to CloudWatch once per received request.

IoT Metrics

Metric	Description
RulesExecuted	The number of AWS IoT rules executed.

Rule Metrics

Metric	Description
TopicMatch	The number of incoming messages published on a topic on which a rule is listening. The <code>RuleName</code> dimension contains the name of the rule.
ParseError	The number of JSON parse errors that occurred in messages published on a topic on which a rule is listening. The <code>RuleName</code> dimension contains the name of the rule.

Rule Action Metrics

Metric	Description
Success	The number of successful rule action invocations. The <code>RuleName</code> dimension contains the name of the rule that specifies the action. The <code>ActionType</code> dimension contains the type of action that was invoked.
Failure	The number of failed rule action invocations. The <code>RuleName</code> dimension contains the name of the rule that specifies the action. The <code>RuleName</code> dimension contains the name of the rule that specifies the action. The <code>ActionType</code> dimension contains the type of action that was invoked.

Message Broker Metrics

Metric	Description
Connect.AuthError	The number of connection requests that could not be authorized by the message broker. The <code>Protocol</code> dimension contains the protocol used to send the CONNECT message.
Connect.ClientError	The number of connection requests rejected because the MQTT message did not meet the requirements defined in AWS IoT Limits . The <code>Protocol</code> dimension contains the protocol used to send the CONNECT message.
Connect.ServerError	The number of connection requests that failed because an internal error occurred. The <code>Protocol</code> dimension contains the protocol used to send the CONNECT message.
Connect.Success	The number of successful connections to the message broker. The <code>Protocol</code> dimension contains the protocol used to send the CONNECT message.
Connect.Throttle	The number of connection requests that were throttled because the client exceeded the allowed connect request rate. The <code>Protocol</code> dimension contains the protocol used to send the CONNECT message.

Metric	Description
Ping.Success	The number of ping messages received by the message broker. The <code>Protocol</code> dimension contains the protocol used to send the ping message.
PublishIn.AuthError	The number of publish requests the message broker was unable to authorize. The <code>Protocol</code> dimension contains the protocol used to publish the message.
PublishIn.ClientError	The number of publish requests rejected by the message broker because the message did not meet the requirements defined in AWS IoT Limits . The <code>Protocol</code> dimension contains the protocol used to publish the message.
PublishIn.ServerError	The number of publish requests the message broker failed to process because an internal error occurred. The <code>Protocol</code> dimension contains the protocol used to send the <code>PUBLISH</code> message.
PublishIn.Success	The number of publish requests successfully processed by the message broker. The <code>Protocol</code> dimension contains the protocol used to send the <code>PUBLISH</code> message.
PublishIn.Throttle	The number of publish request that were throttled because the client exceeded the allowed inbound message rate. The <code>Protocol</code> dimension contains the protocol used to send the <code>PUBLISH</code> message.
PublishOut.AuthError	The number of publish requests made by the message broker that could not be authorized by AWS IoT. The <code>Protocol</code> dimension contains the protocol used to send the <code>PUBLISH</code> message.
PublishOut.ClientError	The number of publish requests made by the message broker that were rejected because the message did not meet the requirements defined in AWS IoT Limits . The <code>Protocol</code> dimension contains the protocol used to send the <code>PUBLISH</code> message.
PublishOut.Success	The number of publish requests successfully made by the message broker. The <code>Protocol</code> dimension contains the protocol used to send the <code>PUBLISH</code> message.
Subscribe.AuthError	The number of subscription requests made by a client that could not be authorized. The <code>Protocol</code> dimension contains the protocol used to send the <code>SUBSCRIBE</code> message.
Subscribe.ClientError	The number of subscribe requests that were rejected because the <code>SUBSCRIBE</code> message did not meet the requirements defined in AWS IoT Limits . The <code>Protocol</code> dimension contains the protocol used to send the <code>SUBSCRIBE</code> message.

Metric	Description
Subscribe.ServerError	The number of subscribe requests that were rejected because an internal error occurred. The <code>Protocol</code> dimension contains the protocol used to send the <code>SUBSCRIBE</code> message.
Subscribe.Success	The number of subscribe requests that were successfully processed by the message broker. The <code>Protocol</code> dimension contains the protocol used to send the <code>SUBSCRIBE</code> message.
Subscribe.Throttle	The number of subscribe requests that were throttled because the client exceeded the allowed subscribe request rate. The <code>Protocol</code> dimension contains the protocol used to send the <code>SUBSCRIBE</code> message.
Unsubscribe.ClientError	The number of unsubscribe requests that were rejected because the <code>UNSUBSCRIBE</code> message did not meet the requirements defined in AWS IoT Limits . The <code>Protocol</code> dimension contains the protocol used to send the <code>UNSUBSCRIBE</code> message.
Unsubscribe.ServerError	The number of unsubscribe requests that were rejected because an internal error occurred. The <code>Protocol</code> dimension contains the protocol used to send the <code>UNSUBSCRIBE</code> message.
Unsubscribe.Success	The number of unsubscribe requests that were successfully processed by the message broker. The <code>Protocol</code> dimension contains the protocol used to send the <code>UNSUBSCRIBE</code> message.
Unsubscribe.Throttle	The number of unsubscribe requests that were rejected because the client exceeded the allowed unsubscribe request rate. The <code>Protocol</code> dimension contains the protocol used to send the <code>UNSUBSCRIBE</code> message.

Note

The message broker metrics are displayed in the AWS IoT console under Protocol Metrics.

Thing Shadow Metrics

Metric	Description
DeleteThingShadow.Accepted	The number of <code>DeleteThingShadow</code> requests processed successfully. The <code>Protocol</code> dimension contains the protocol used to make the request.
GetThingShadow.Accepted	The number of <code>GetThingShadow</code> requests processed successfully. The <code>Protocol</code> dimension contains the protocol used to make the request.
UpdateThingShadow.Accepted	The number of <code>UpdateThingShadow</code> requests processed successfully. The <code>Protocol</code> dimension contains the protocol used to make the request.

Note

The thing shadow metrics are displayed in the AWS IoT console under Protocol Metrics.

Dimensions for Metrics

Metrics use the namespace and provide metrics for the following dimension(s):

Dimension	Description
ActionType	The action type specified by the rule that triggered by the request.
Protocol	The protocol used to make the request. Valid values are: MQTT or HTTP
RuleName	The name of the rule triggered by the request.

如何使用 AWS IoT 指标？

您可以通过多种方式分析 AWS IoT 报告指标提供的信息。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：

- 十个事物几乎在同一时间连接到 AWS IoT。
- 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，
- 然后断开与 AWS IoT 的连接。

下面列出的是能够带您入门的启发式问题，但并不全面。

- [我如何知道事物每天是否成功建立连接？\(p. 245\)](#)
- [我如何知道事物每天是否发布数据？\(p. 246\)](#)
- [我如何知道事物影子更新每天是否遭到拒绝？\(p. 247\)](#)

创建 CloudWatch 警报以监控 AWS IoT

您可以创建 CloudWatch 警报，以在警报改变状态时发送 Amazon SNS 消息。警报会在您指定的时间段内监控一个指标，并根据指标值在多个时间段内相对于给定阈值的情况执行一项或多项操作。操作是向 Amazon SNS 主题或 Auto Scaling 策略发送的通知。警报只会调用操作进行持续的状态变更。CloudWatch 警报将不会调用操作，因为这些操作处于特定状态；该状态必须改变并在指定数量的时间段内一直保持。

我如何知道事物每天是否成功建立连接？

1. 创建一个 Amazon SNS 主题：`arn:aws:sns:us-east-1:123456789012:things-not-connecting-successfully`。
有关更多信息，请参阅[设置 Amazon Simple Notification Service](#)。
2. 创建警报。

```
Prompt>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 \
--unit Count \
--evaluation-periods 1 \
--alarm-actions arn:aws:sns:us-east-1:1234567890:things-not-connecting-successfully
```

```
Prompt>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 \
--unit Count \
--evaluation-periods 1 \
--alarm-actions arn:aws:sns:us-east-1:1234567890:things-not-connecting-successfully
```

3. 测试警报。

```
Prompt>aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
"initializing" --state-value OK
```

```
Prompt>aws cloudwatch set-alarm-state --alarm-name ConnectSuccessAlarm --state-reason
"initializing" --state-value ALARM
```

我如何知道事物每天是否发布数据？

1. 创建一个 Amazon SNS 主题 : arn:aws:sns:us-east-1:123456789012:things-not-publishing-data。

有关更多信息，请参阅[设置 Amazon Simple Notification Service](#)。

2. 创建警报。

```
Prompt>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 \
--unit Count \
--evaluation-periods 1 \
--alarm-actions arn:aws:sns:us-east-1:1234567890:things-not-publishing-data
```

3. 测试警报。

```
Prompt>aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason "initializing" --state-value OK
```

```
Prompt>aws cloudwatch set-alarm-state --alarm-name PublishInSuccessAlarm --state-reason "initializing" --state-value ALARM
```

我如何知道事物影子更新每天是否遭到拒绝？

1. 创建一个 Amazon SNS 主题 : arn:aws:sns:us-east-1:1234567890:things-shadow-updates-rejected。

有关更多信息，请参阅[设置 Amazon Simple Notification Service](#)。

2. 创建警报。

```
Prompt>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 arn:aws:sns:us-east-1:1234567890:things-shadow-updates-rejected
```

3. 测试警报。

```
Prompt>aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-reason "initializing" --state-value OK
```

```
Prompt>aws cloudwatch set-alarm-state --alarm-name UpdateThingShadowSuccessAlarm --state-reason "initializing" --state-value ALARM
```

使用 AWS CloudTrail 记录 AWS IoT API 调用

AWS IoT 与 CloudTrail 集成，后者可捕获所有 AWS IoT API 调用并将日志文件传输到您指定的 Amazon S3 存储桶中。CloudTrail 可捕获来自 AWS IoT 控制台的 API 调用或者从您的代码对 AWS IoT API 进行的调用。通过使用 CloudTrail 收集的信息，您可以确定向 AWS IoT 发出了什么请求、发出请求的源 IP 地址、何人发出的请求以及请求的发出时间等。

要了解有关 CloudTrail 的更多信息，包括如何对其进行配置和启用，请参阅[AWS CloudTrail User Guide](#)。

CloudTrail 中的 AWS IoT 信息

在您的 AWS 账户中启用 CloudTrail 日志记录后，会在 CloudTrail 日志文件中跟踪对 AWS IoT 操作执行的 API 调用，这些操作会随其他 AWS 服务记录一起写在这些日志文件中。CloudTrail 基于时间段和文件大小来确定何时创建新文件并向其写入内容。

所有 AWS IoT 操作均由 CloudTrail 记录下来并记载到[AWS IoT API 参考](#)中。例如，对 CreateThing、ListThings 和 ListTopicRules 部分的调用均将在 CloudTrail 日志文件中生成条目。

每个日志条目都包含有关生成请求的人员的信息。日志条目中的用户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身份用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 [CloudTrail userIdentity 元素](#)。

日志文件可以在 Amazon S3 存储桶中存储任意长时间，不过您也可以定义 Amazon S3 生命周期规则以自动存档或删除日志文件。默认情况下，将使用 Amazon S3 服务器端加密 (SSE) 对日志文件进行加密。

如果您需要获得日志文件传输的通知，则可以将 CloudTrail 配置为在传输新日志文件时发布 Amazon SNS 通知。有关更多信息，请参阅[为 CloudTrail 配置 Amazon SNS 通知](#)。

您还可以将多个 AWS 区域和多个 AWS 账户中的 AWS IoT 日志文件聚合到单个 Amazon S3 存储桶中。

有关更多信息，请参阅[接收多个区域中的 CloudTrail 日志文件](#)和[从多个账户中接收 CloudTrail 日志文件](#)。

了解 AWS IoT 日志文件条目

CloudTrail 日志文件可以包含一个或多个日志条目。每个条目列出了多个 JSON 格式的事件。一个日志条目表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间、请求参数等方面的信息。日志条目不是公用 API 调用的有序堆栈跟踪，因此它们不会以任何特定顺序显示。

下面的示例显示了一个 CloudTrail 日志条目，该条目说明了 `AttachPrincipalPolicy` 操作。

```
{  
    "timestamp": "1460159496",  
    "AdditionalEventData": "",  
    "Annotation": "",  
    "ApiVersion": "",  
    "ErrorCode": "",  
    "ErrorMessage": "",  
    "EventID": "8bfff4fed-c229-4d2d-8264-4ab28a487505",  
    "EventName": "AttachPrincipalPolicy",  
    "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": "AKIAI44QH8DHBEEXAMPLE",  
        "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-1C5T1YC5MHPYT",
            "accountId": "222222222222",
            "userName": "iotmonitor-us-east-1-InstanceRole-1C5T1YC5MHPYT"
        }
    },
    "invokedBy": {
        "serviceAccountId": "111111111111"
    }
},
"VpcEndpointId": ""
}
```

AWS IoT 故障排除

以下信息可帮助您处理 AWS IoT 中的常见问题。

任务

- [诊断连接问题 \(p. 250\)](#)
- [设置 CloudWatch Logs \(p. 251\)](#)
- [诊断规则问题 \(p. 255\)](#)
- [诊断 Thing Shadows 问题 \(p. 256\)](#)
- [诊断 Salesforce IoT 输入流操作问题 \(p. 257\)](#)

诊断连接问题

身份验证

我的设备如何对 AWS IoT 终端节点进行身份验证？

将 AWS IoT CA 证书添加到您的客户端信任的存储。您可以从[此处](#)下载 CA 证书。
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.us-east-1.amazonaws.com:8443 -  
CAfile CA.pem -cert cert.pem -key privateKey.pem
```

授权

我收到了代理发送的 PUBNACK 或 SUBNACK 回复。我应该怎么办？

请确保当前有策略附加到您用于调用 AWS IoT 的证书。默认情况下，所有的发布/订阅操作均将被拒绝。

设置 CloudWatch Logs

当消息从您的设备通过消息代理和规则引擎时，AWS IoT 会发送关于每条消息的进度事件。您可以选择在 CloudWatch Logs 中查看这些事件。有关更多信息，请参阅 [CloudWatch Logs](#)。

Note

在启用 AWS IoT 日志记录之前，请务必了解您的 AWS 账户是否具有 CloudWatch Logs 访问权限。拥有 CloudWatch Logs 访问权限的用户将能够从您的设备查看调试信息。

为日志记录配置 IAM 角色

使用 IAM 控制台创建日志记录角色。

为日志记录创建 IAM 角色

以下策略文档提供了角色策略和信任策略，借助这些策略，AWS IoT 可代表您向 CloudWatch 提交日志。

角色策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs:CreateLogGroup",  
                "logs:CreateLogStream",  
                "logs:PutLogEvents",  
                "logs:PutMetricFilter",  
                "logs:PutRetentionPolicy"  
            ],  
            "Resource": [  
                "*"  
            ]  
        }  
    ]  
}
```

信任策略：

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "iot.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

将日志记录角色注册到 AWS IoT

使用 AWS IoT 控制台或以下 CLI 命令将日志记录角色注册到 AWS IoT。

```
aws iot setLoggingOptions --logging-options-payload  
roleArn="arn:aws:iam::<your-aws-account-num>:role/  
IoTLoggingRole",logLevel="INFO"
```

日志级别可以是 DEBUG、INFO、ERROR 或 DISABLED：

- DEBUG 提供最详细的 AWS IoT 活动信息。
- INFO 提供大多数操作的汇总视图。对于大多数用户而言，这已经足够了。
- ERROR 仅提供错误案例。
- DISABLED 会完全删除日志记录，但仍保留您的日志记录角色。

CloudWatch 日志条目格式

每个日志条目都包含以下信息：

事件

描述 AWS IoT 中发生的操作。

日志级别

日志记录的级别。可以是 DEBUG、INFO、ERROR、WARN、DISABLED。

时间戳

日志生成的时间。

追踪 ID

针对传入的请求随机生成的标识符，可用于筛选与某条传入消息对应的所有日志。

委托人 ID

证书指纹或事物名称，取决于接收来自设备的请求的终端节点 (MQTT 或 HTTP)。

根据消息上下文，日志消息中还可能包括以下字段：

主题名称

MQTT 主题的名称；当接收到 MQTT 发布的消息或订阅消息后，该名称将添加到一个条目。

ClientId

发送 MQTT 消息的客户端的 ID。

ThingName

事物名称，在请求被发送到 HTTP 终端节点以更新或删除事物状态时，将在条目中添加该名称。

规则 ID

规则标识符；当规则被触发时，该标识符将包含规则的 ID。

日志级别

日志级别指定了将生成哪种类型的日志。

ERROR

导致操作失败的任何错误。

日志将仅包含 ERROR 信息。

WARN

可能导致系统中出现不一致问题的任何情况，但不一定会导致操作失败。

日志将包括 ERROR 和 WARN 信息。

INFO

有关事物的高级别信息。

日志将包括 INFO、ERROR 和 WARN 信息。

DEBUG

可能有助于调试问题的信息。

日志将包括 DEBUG、INFO、ERROR 和 WARN 信息。

DISABLED

所有日志记录均处于禁用状态。

日志记录事件和错误代码

本部分列出了 AWS IoT 发送的日志记录事件和错误代码。

身份和安全

操作/事件名称	描述
Authentication Success	成功对证书进行了身份验证。
Authentication Failure	对证书进行的身份验证失败。

身份和安全错误代码

错误代码	错误描述
401	未授权

消息代理

操作/事件名称	描述
MQTT Publish	已收到 MQTT Publish。
MQTT Subscribe	已收到 MQTT Subscribe。
MQTT Connect	已收到 MQTT Connect。
MQTT Disconnect	已收到 MQTT Disconnect。
HTTP/1.1 POST	已收到 MHTTP/1.1 POST。
HTTP/1.1 GET	已收到 HTTP/1.1 GET。

操作/事件名称	描述
HTTP/1.1 Unsupported Method	当消息中包含语法错误或操作 (HTTP PUT/DELETE/) 被禁止时使用。
Malformed HTTP Message	由于 HTTP 消息格式错误，连接被终止。
Malformed MQTT Message	由于 MQTT 消息格式错误，连接被终止。
Authorization Failed	此客户端尝试针对其未获授权的主题发布消息或订阅该主题。
Package Exceeds Maximum Payload Size	此客户端尝试发布的有效负载超过了消息代理的上限。

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

规则引擎事件

操作/事件名称	描述
MessageReceived	收到关于主题的请求。
DynamoActionSuccess	成功放置 DynamoDB 记录。
DynamoActionFailure	放置 DynamoDB 记录失败。
KinesisActionSuccess	成功发布 Kinesis 消息。
KinesisActionFailure	发布 Kinesis 消息失败。
LambdaActionSuccess	成功调用 Lambda 函数。
LambdaActionFailure	调用 Lambda 函数失败。
RepublishActionSuccess	成功重新发布消息。
MessageReceived	收到关于主题的请求。
RepublishActionFailure	重新发布消息失败。
S3ActionSuccess	成功放置 Amazon S3 对象。
S3ActionFailure	放置 Amazon S3 对象失败。
SNSActionSuccess	成功发布消息到 Amazon SNS 主题。
SNSActionFailure	向 Amazon SNS 主题发布消息失败。
SQSActionSuccess	成功发送消息到 Amazon SQS。

操作/事件名称	描述
SQSActionFailure	向 Amazon SQS 发送消息失败。
SalesforceActionSuccess	已成功将消息发送到 Salesforce 输入流。
SalesforceActionFailure	未能将消息发送到 Salesforce 输入流。

事物影子事件

操作/事件名称	描述
UpdateThingState	通过 HTTP 或 MQTT 更新了事物状态。
DeleteThing	事物已删除。

事物影子错误代码

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。
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。

诊断规则问题

CloudWatch Logs 是您对规则所存在的问题进行调试的最佳位置。当您为 AWS IoT 启用 CloudWatch Logs 后，您可了解哪些规则被触发以及操作是否取得了成功。您还可以了解 WHERE 子句条件是否匹配。

最常见的问题是授权问题。在这种情况下，日志将告诉您，您的角色未获得授权，不能针对资源执行 AssumeRole 操作。

查看 CloudWatch 日志 (控制台)

1. 在 AWS 管理控制台中，导航到 CloudWatch 控制台。
2. 选择 Logs，然后从列表中选择 AWSIoTLogs 日志组。
3. 在 Streams for AWSIoTLogs 页面上，您可以在您的账户下找到调用到 AWS IoT 的每个委托人 (X.509 证书、IAM 用户或 Amazon Cognito 身份) 的日志流。

有关更多信息，请参阅 [CloudWatch Logs](#)。

外部服务由最终用户控制。在执行规则之前，请确保为外部服务设置了足够的吞吐量和容量单位。

诊断 Thing Shadows 问题

诊断 Thing Shadows

问题	故障排除指南
事物影子文档因使用“无效的 JSON 文档”而被拒绝。	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 事物影子文档语法 。
我提交了正确的 JSON，但事物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 <code>desired</code> 和 <code>reported</code> 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。
我收到一个错误消息，称事物影子超出了允许的大小。	事物影子仅支持 8KB 数据。请尝试缩短您的 JSON 文档中的字段名称。您也可以创建更多事物影子，这种方法较为简便。一个设备可拥有无限数量的事物影子。唯一的要求是，您账户中的事物名称必须是唯一的。
我收到了一个超过 8KB 的事物影子。这是怎么回事？	在您收到事物影子之后，AWS IoT 服务会向其添加元数据。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。只有发送到事物影子的状态文档中有关 <code>desired</code> 状态和 <code>reported</code> 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 <code>./update/accepted</code> 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。	单个字段和整个 JSON 文档的时间戳会在 AWS IoT 服务收到文档或状态文档发布到 <code>./update/accepted</code> 和 <code>./update/delta</code> 消息后更新。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事物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 更新事物影子文档时，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据执行相应的操作（UPDATE/GET/DELETE）。IAM 策略和证书策略是相互独立的。
其他问题。	Thing Shadows 服务会将错误记录到 CloudWatch Logs 中。要找出设备和配置问题，请启用 CloudWatch Logs 并查看日志，以获取调试信息。

诊断 Salesforce IoT 输入流操作问题

执行跟踪

如何查看 Salesforce 操作的执行跟踪？

如果未设置 CloudWatch Logs，请参阅[设置 CloudWatch Logs \(p. 251\)](#)部分。一旦激活日志，您将能够查看 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？

查看在 CloudWatch Logs 中执行 Salesforce 操作所生成的日志。如果您看到“*Action executed successfully*”，则表示 AWS IoT Rules Engine 已收到来自 Salesforce IoT 的确认信息，表明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，请参阅 Salesforce IoT 支持。

如果消息未成功发送到 Salesforce IoT 输入流，该怎么办？

查看在 CloudWatch Logs 中执行 Salesforce 操作所生成的日志。根据日志条目，您可以尝试以下操作：

`Failed to locate the host`

请检查此操作的 `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`

请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。请检查您是否正在发送 JSON 负载。

`Received Unauthorized Exception from Salesforce`

请检查此操作的 `token` 参数是否正确以及您的令牌是否仍有效。

`Received Not Found Exception from Salesforce`

请检查此操作的 `url` 参数是否正确以及您的 Salesforce IoT 输入流是否存在。

如果您遇到未列出的错误，请联系 AWS 支持。

有关更多信息，请参阅 [CloudWatch Logs](#)。