IOT Device Integration Guide

Setting up your IoT device to run an example script

Using GCP Console and the gCloud command-line tool

Contents

Introduction

This **IoT Device Integration Guide** is a companion to the IoT Partner Quickstart which provides an overview of Cloud IoT Core and other key Google Cloud Platform (GCP) products and services. That document also includes a list of technical articles and other resources that will be helpful to developers who want to put their integrated device to work.

See the Documentation Quicklist for a complete list of conceptual overviews, tutorials, APIs, and reference documentation for Google Cloud Platform products. In addition to the Quicklist, here are some IoT-specific guides and overviews:

This guide summarizes the preliminary setup requirements and shows you how to register and test a device using either the Google Cloud Platform Console (GCP Console) or the gCloud command-line tool. Each of these approaches has its own section in this guide. Other approaches, including using REST APIs, will be covered in a future release of this document.

This hands-on guide includes these topics:

- Preliminary Setup
- Using GCP Console for setup tasks
- Using gCloud for setup tasks
- Testing the setup by running sample code
- Troubleshooting

The goal of this document is to help you get a device set up and working with Cloud IoT Core as quickly as possible and yet provide enough background information to answer questions you might have.

Tip	To get a device set up as quickly as possible without any additional context, see the IoT Device Integration Cheatsheet. The Cheatsheet provides a summary of commands that you can copy/paste/edit with your own project name and other details.

Preliminary Setup

Before you get started, you should:

1. Create a Google Cloud Platform account (if necessary). If you already have an account on GCP, you'll need to know your logon account name (typically, your business email address) and your password to use either GCP Console or gCloud.
2. Set up the required libraries needed by the device for connectivity (MQTT libraries, for example) and authentication (JWT libraries). The specific libraries and versions vary by hardware platform, OS, programming language, and application development environment.

3. Create the public/private key pair and have them ready for use. You'll need the public key (or its certificate) to associate with device when you add it to the registry, and the private key must be available on the device at runtime.

4. Install and initialize the Google Cloud SDK. This step is required only if you plan to use the gCloud command-line tool. However, Google recommends having the gClouthe gCloud command-line tool because it is convenient to use, to augment your interactions with the GCP Console. For example, to test your device's connections to the Cloud IoT Core and the messages sent to Cloud Pub/Sub, you can use gCloud to pull subscriptions and make sure the events were queued.

Create an account on Google Cloud Platform (if needed)

Regardless of the approach you plan to use to develop your devices—GCP Console, gCloud command-line tool, REST APIs, or specific client language and client APIs—you must have an account on Google Cloud Platform. If you do not have an account on Google Cloud Platform, create one now before proceeding. To create the account on the Google Cloud Platform Free Tier for use while developing and testing your device with GCP and Cloud IoT core:

1. Browse to cloud.google.com/free and click **Try It Free**.

2. Enter your company email account and create a password. These comprise your login credentials to the GCP Console [console.cloud.google.com], so keep them private and secure.
    - The Free Tier requires a credit card number or other billing information. The $300 credit is applied to your Billing account as a credit. Each project you create on GCP is always associated with a Billing account.

3. Follow the other prompts on this simple setup. See GCP documentation (Platform Overview > Concepts) for more information about creating an account and getting started with GCP.

Set up required libraries on the device

To support secure communications and authentication between the device and the Google Cloud Platform, the device needs the libraries appropriate for the underlying hardware and OS for:

TLS 1.2	Transport Layer Security (TLS) is a protocol for securing communications channels between client and server processes. Version 1.2 of TLS is specifically required. With TLS on the device, specific well-known ports are used for encrypted communications: • Port 443 is the default for HTTP over TLS (HTTP/S) • Port 8883 is the MQTT over TLS (secure-mqtt)
JWT	A JSON Web Token (RFC 7519) is a JSON (JavaScript Object Notation)-formatted security artifact. In the context of Google Cloud Platform and Cloud IoT Core specifically, a JWT facilitates the authentication of the client (the device) to Google Cloud. Download the appropriate JWT library for the language you plan to use for your application. JWT libraries are available for Go, Java, JavaScript, .NET, Node.js, Perl, Python, Ruby, and more.

MQTT	Message queuing telemetry transport is a lightweight publish/subscribe messaging transport and the most frequently used communications mechanism for IoT. MQTT has lower latency and utilizes less bandwidth as wells as faster throughput than HTTP (see Cloud IoT > Concepts > Protocols for more information). MQTT also supports binary data, which is not supported by HTTP. For example, HTTP requires Base64-encoded (ASCII) text for the public/private keys, consuming more network and CPU resources. Cloud IoT Core supports both MQTT and HTTP. To use MQTT on your device, you must install the appropriate MQTT client library. See the libraries page of the mqtt/mqtt.github.io GitHub wiki for details.

As an example of the libraries needed to run the Python sample application cloudiot_mqtt_example.py on a Linux device, the following might all be installed:

Library and version example	Description
google-api-python-client 1.7.4 google-auth-httplib2 0.0.3 google-auth 1.5.1 google-cloud-pubsub 0.37.2	Google client libraries (Python client) to support authentication, authorization, and specific client API calls
cryptography 2.3.1	Python library of cryptographic primitives
pyjwt 1.6.4	A Python implementation of RFC-7519 (JWT)
paho-mqtt 1.3.1	Open source Eclipse Paho client implementation of MQTT

The GoogleCloudPlatform repository on GitHub has examples for Python, Java, and other client libraries. Be sure to set up your device with the required software and library versions and follow any other instructions in the appropriate README files before trying to run any of the sample applications.

Note: If your customers will need to set up your device with any of the prerequisites, you must **provide step-by-step documentation** so they can do so quickly and easily.

Create a public/private key pair

Devices must have several security artifacts, including a JWT library as discussed above to support authentication at runtime. In greatly oversimplified terms, a private key residing on the device is used to sign a JWT that gets sent to Google Cloud Platform, and Google Cloud Platform uses the device's public key to verify the source of the JWT as the device.

For more information about how IoT uses JWT and for an overview, see:

- Device Security
- How-to Guides > Getting Started

To create the private/public key pair, Google recommends using OpenSSL, typically available by default with any Linux distribution. If the device does not have OpenSSL available on the underlying OS, you must download and install it so you can generate the necessary keys or certificates.

Note: Google recommends generating these security artifacts on the device itself to avoid transferring the private key from another machine to the device for optimal security.

Cloud IoT Core supports keys generated using either RSA and EC (elliptic curve) algorithms:

- RSA is one of the first public/private cryptosystems. It is still widely used today, and is included in most client libraries. RSA keys can be quite large and can consume lots of CPU, however, so take this into account when integrating your specific device. Figure 1 shows an example of using OpenSSL to generate RSA key pairs and a self-signed X.509 certificate.

- EC uses elliptic curve algebra to create public/private key pairs. EC is a newer approach that results in smaller key sizes and less CPU consumption, so EC generated keys are better for devices with very limited resources. EC keys may require additional client libraries. Figure 2 shows an example of using OpenSSL to generate public/private keys using an EC algorithm.

Figure 1 shows an example of using OpenSSL directly on an example device to generate a private/public key pair using the RSA algorithm. First, a 2048-bit private key is created (labpi_private.pem in Figure 1), and then the private key is used to create a matching public key (labpi_public.pem). At this point, the private key on the device would be usable for signing token (a JWT) using RS256 and submitting to GCP (to the MQTT bridge), where its matching public key (labpi_public.pem, assuming it has been uploaded to the registry and associated with this device) would be used by Cloud IoT Core to verify the key used for the JWT.

Figure 1. Creating a private/public key pair (RSA) and self-signed public key (X.509 certificate)

```
pi@lab-pi:~ $ openssl genrsa -out labpi_private.pem 2048
Generating RSA private key, 2048 bit long modulus
......................+++
.....+++
e is 65537 (0x010001)
pi@lab-pi:~ $ openssl rsa -in labpi_private.pem -pubout -out labpi_public.pem
writing RSA key
pi@lab-pi:~ $ openssl req -x509 -nodes -newkey rsa:2048 -keyout labpi_private.pem -days 365
-out labpi_cert.pem -subj "/CN=unused"
Generating a 2048 bit RSA private key
..........................+++
.................................+++
writing new private key to 'labpi_private.pem'
-----
```

As an alternative, the third command shown in Figure 1 creates a private key also called labpi_private.pem (thus, overwriting the key created in the first line) and then generates a self-signed certificate (the -x509 option for OpenSSL's **req** command (which typically is used to create CSRs (certificate signing **req**uests) that would be sent to commercial CAs or to an internal CA controlled by your organization, which in turn creates a certificate attesting to the identity of the requesting entity. In this case, the certificate is self-signed (self-signed certificates are recommended for test and development environments only). This is then uploaded to the registry (identified as RS256_X509). This particular generated self-signed certificate will expire in 365 days, after which it can no longer verify tokens sent from the device that have been signed using the matching private key (if not specified, the default for -x509 is 30 days).

The distinguished name attributes are not used (as shown with common name set to "unused" in this example) because (as mentioned) the certificate is not used in the context of TLS to authenticate the client, but rather is used by the client to create the JSON Web Token (JWT) that gets sent from the client to Google Cloud. The -nodes option ("no DES") means don't encrypt the private key (encrypting the private key would require that a password be used each time the key is accessed, ie., each time the JWT needs to get created and signed).

Figure 2 shows the equivalent OpenSSL commands to generate private/public key pairs using elliptic curve algorithms rather than RSA. You would follow the same process to upload the public key to the registry for the device as detailed above, but would specify ES256 or ES256_X509 depending on whether you use the public key or the X.509 certificate.

Figure 2. Creating a private/public key pair (EC) and self-signed public key (X.509 certificate)

```
openssl ecparam -genkey -name prime256v1 -noout -out ec_private.pem
openssl ec -in ec_private.pem -pubout -out ec_public.pem
openssl pkcs8 -topk8 -inform PEM -outform DER -in rsa_private.pem -nocrypt > rsa_private_pkcs8
openssl pkcs8 -topk8 -inform PEM -outform DER -in ec_private.pem -nocrypt > ec_private_pkcs8
```

If the registry already exists at this point, you can use the GCP Console or gCloud command-line to add the device with its associated public key to the registry.

Note If you create the public/private key pair on a machine other than the actual device, you must **securely** transfer the private key to the device and delete the key from the system on which it was initially created. The private key must be associated exclusively with the intended device to ensure the security of the system.

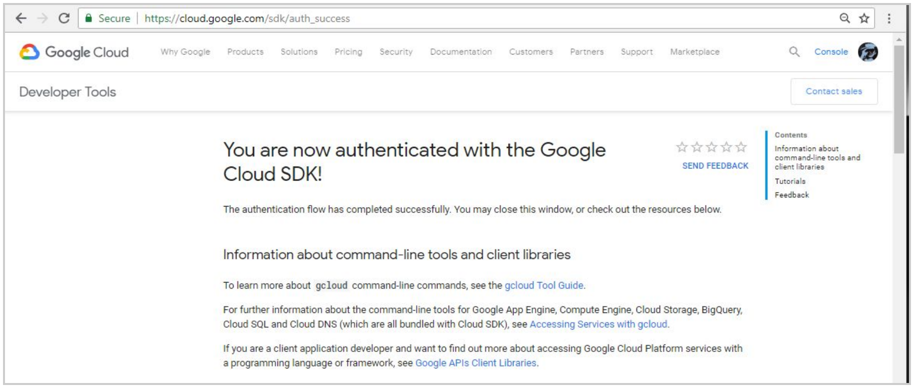
Install and initialize the Google Cloud SDK

The Google Cloud SDK includes the gCloud tool for command-line use with Google Cloud Platform. Before you will be able to use gCloud commands for useful tasks, you must:

1. Install Google Cloud SDK on your development machine, the device, or both.

2. Run the initialization script to authenticate the gCloud client to your account on the Google Cloud Platform. You'll be prompted for your GCP account and password. Assuming successful authentication, the command-line spawns a browser session and opens to an "authentication success" page, as shown in Figure 3:

Figure 3. Authenticated with Google Cloud SDK success page



You are now authenticated with the Google Cloud SDK!

The authentication flow has completed successfully. You may close this window, or check out the resources below.

Information about command-line tools and client libraries

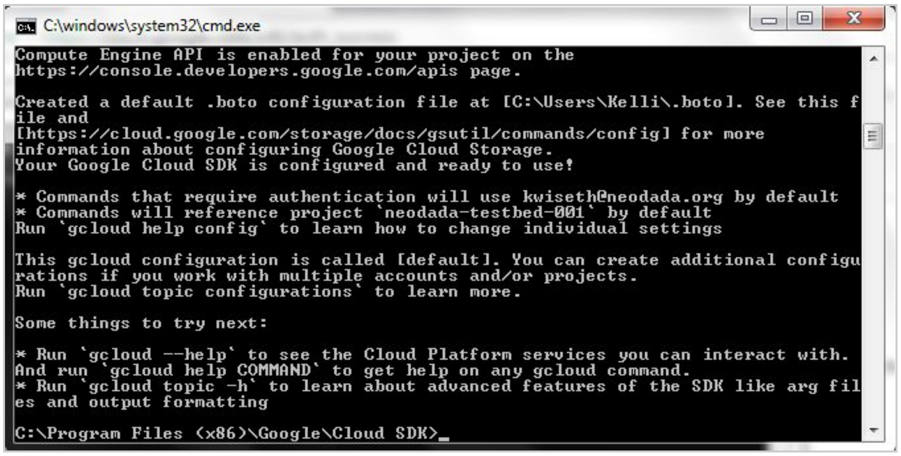
To learn more about `gcloud` command-line commands, see the gcloud Tool Guide.

For further information about the command-line tools for Google App Engine, Compute Engine, Cloud Storage, BigQuery, Cloud SQL and Cloud DNS (which are all bundled with Cloud SDK), see Accessing Services with gcloud.

If you are a client application developer and want to find out more about accessing Google Cloud Platform services with a programming language or framework, see Google APIs Client Libraries.

3. Dismiss the page and return to the console session to continue setting up gCloud. If your GCP account already has one or more Projects created you can select the one you want to use with gCloud, or you can create a new project. When the initialization process completes, you'll see some useful tips displayed and the command prompt returned to its ready state.

Figure 4. The gCloud command-line interface



```
Compute Engine API is enabled for your project on the
https://console.developers.google.com/apis page.

Created a default .boto configuration file at [C:\Users\Kelli\.boto]. See this f
ile and
[https://cloud.google.com/storage/docs/gsutil/commands/config] for more
information about configuring Google Cloud Storage.
Your Google Cloud SDK is configured and ready to use!

* Commands that require authentication will use kwiseth@neodada.org by default
* Commands will reference project `neodada-testbed-001` by default
Run `gcloud help config` to learn how to change individual settings

This gcloud configuration is called [default]. You can create additional configu
rations if you work with multiple accounts and/or projects.
Run `gcloud topic configurations` to learn more.

Some things to try next:

* Run `gcloud --help` to see the Cloud Platform services you can interact with.
And run `gcloud help COMMAND` to get help on any gcloud command.
* Run `gcloud topic -h` to learn about advanced features of the SDK like arg fil
es and output formatting

C:\Program Files (x86)\Google\Cloud SDK>_
```

To get started using gCloud with your Cloud IoT Core, see Using gCloud for setup tasks.

Using GCP Console for setup tasks

If you have completed all Preliminary Setup steps for the device, you can configure Cloud IoT Core. The general process is as follows (each of these encompasses several sub-steps):

- Create a project
- Configure the project
- Create a device registry
- Register your device

As an alternative to using the GCP Console for these tasks, you can use the gCloud command-line.

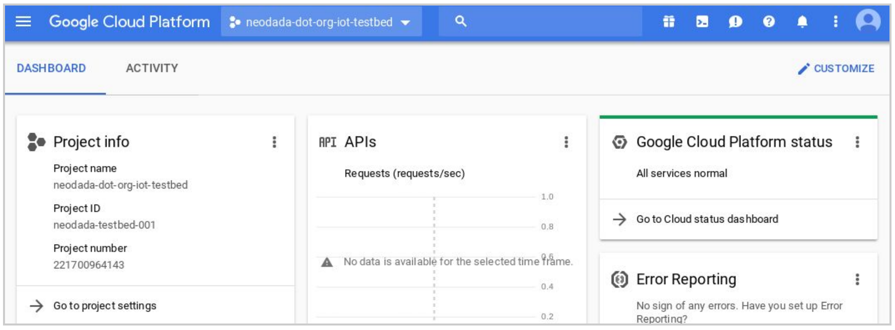
Note	The steps below assume the default GCP Console, without any selections 'pinned' to the top of the menu.

Create a project

The GCP Console uses Projects as an organizing mechanism to keep track of the GCP products and services (Cloud IoT Core, Cloud Pub/Sub), APIs, and other cloud-based objects (such as the registry for your devices) that interact as whole. For more information about Projects, see How-to Guides > Creating and Managing Projects.

1. Log in to Google Cloud Platform (GCP) by opening your browser to console.cloud.google.com and entering your GCP account credentials (Google account and password) on the login page.

2. Click the Google Cloud Platform **navigation menu** icon (the top-left-most icon that looks like three horizontal bars, sometimes referred to as the "hamburger" icon) and then click **Home** at the top of the list. The Home view displays information for the selected project:



Google Cloud Platform neodada-dot-org-iot-testbed

DASHBOARD ACTIVITY CUSTOMIZE

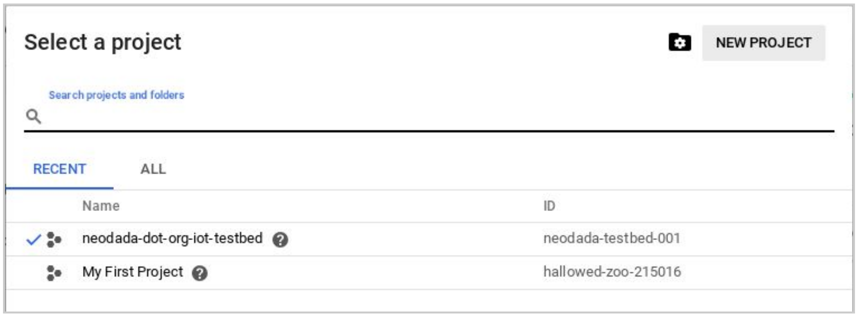
Project info
Project name
neodada-dot-org-iot-testbed
Project ID
neodada-testbed-001
Project number
221700964143
Go to project settings

APIs
Requests (requests/sec)
No data is available for the selected time frame.

Google Cloud Platform status
All services normal
Go to Cloud status dashboard

Error Reporting
No sign of any errors. Have you set up Error Reporting?

3. Create a new project by clicking the Project name displayed in the drop-down at the top of the Home page (to the right of **Google Cloud Platform**) to open the **Select a project** page:



Select a project

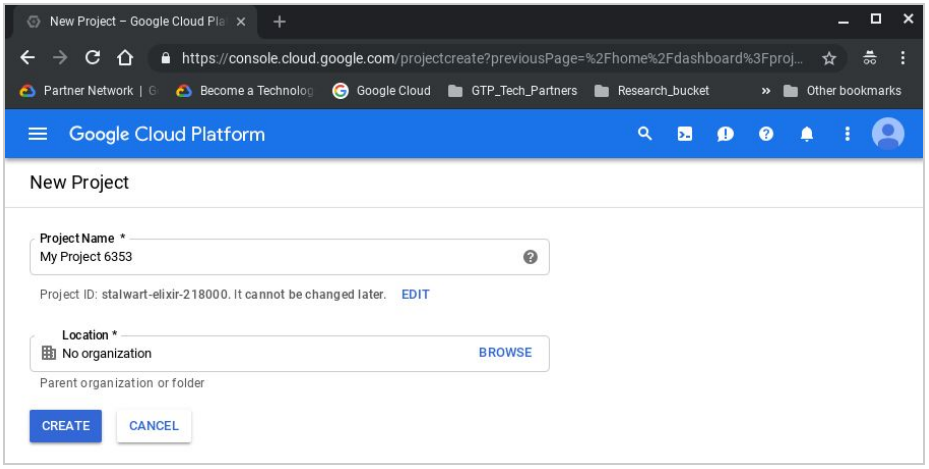
NEW PROJECT

Search projects and folders

RECENT ALL

Name	ID
✓ neodada-dot-org-iot-testbed	neodada-testbed-001
My First Project	hallowed-zoo-215016

4. Click **NEW PROJECT**. The New Project page displays a generated Project Name in the entry field and shows a generated Project ID that will be associated with the name once created.



New Project – Google Cloud Pla

https://console.cloud.google.com/projectcreate?previousPage=%2Fhome%2Fdashboard%3Fproj...

Partner Network | G · Become a Technolog · Google Cloud · GTP_Tech_Partners · Research_bucket · Other bookmarks

Google Cloud Platform

New Project

Project Name *
My Project 6353

Project ID: stalwart-elixir-218000. It cannot be changed later. EDIT

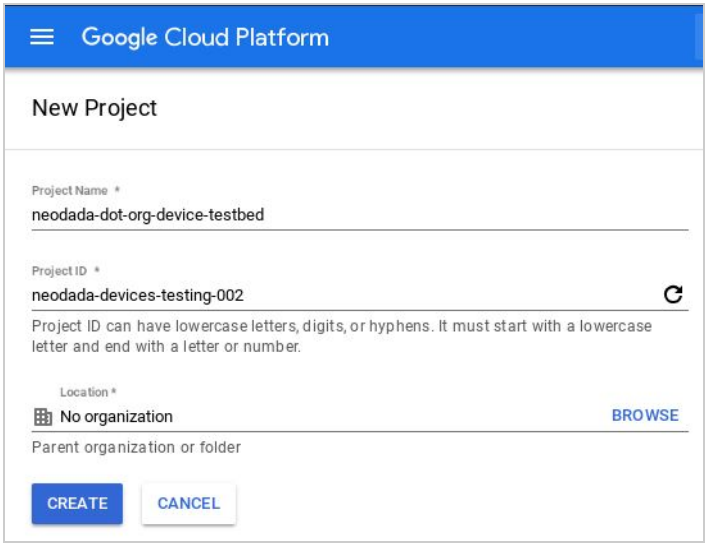
Location *
No organization BROWSE

Parent organization or folder

CREATE CANCEL

You can accept these if you like but they are not especially user friendly and cannot be changed later (see screenshot above) so it's best to replace both values before proceeding.

5. Click the **EDIT** link (to open the New Project information page:



Google Cloud Platform

New Project

Project Name *
neodada-dot-org-device-testbed

Project ID *
neodada-devices-testing-002

Project ID can have lowercase letters, digits, or hyphens. It must start with a lowercase letter and end with a letter or number.

Location *
No organization BROWSE

Parent organization or folder

CREATE CANCEL

6. Enter meaningful values for the **Project Name** and a **Project ID** as shown in the screenshot above. The Location (parent organization or folder) field applies only to those accounts that have been set up in the context of G Suite or Cloud Identity & Access Management so disregard if not relevant.

7. Click **CREATE**. A Project number is generated and assigned to your project by Google. (The Home view screenshot above shows a complete example.) If you accepted the pre-populated Project Name and Project ID, be sure to make a note of them. See Google Cloud Platform Overview > Concepts for more information about GCP and Projects.

Each project is linked to a specific Billing account.

- If you have only one Billing account, the projects you create are automatically linked to it by default.
- If you have multiple Billing accounts you must modify the project's Billing settings to link the project to a specific Billing account. Do so before configuring the other details of the project so that the appropriate Billing account is linked to the project.

Configure the project for Cloud IoT Core and Cloud Pub/Sub

Cloud IoT Core integrates with Cloud Pub/Sub to handle message queuing needed by IoT use cases. The APIs for both of these services must be enabled for your project. In addition, the notification mechanisms of Cloud Pub/Sub must be configured. The steps in this section guide you through these configurations. These steps assume that the appropriate project is selected and displayed in the Home page of the GCP Console.

8. Click the Navigation menu and select **APIs & Services > Library** to open the API Library. There are many ways to filter the pages of cards available but for simplicity's sake, filter by **Category > Big data** which contains both APIs needed.

9. Click the **Google Cloud IoT API** card to open its details and then click **ENABLE**. The API is enabled on your project, and the GCP Console redisplays your project open to APIs & Services > Dashboard.

10. From the Dashboard, click the **ENABLE APIS AND SERVICES** link to return to the API Library. Click Category > Big data, click the **Cloud Pub/Sub API** card to open its details, and then click **ENABLE**.

When the Dashboard redisplays, visually confirm that both APIs have been enabled for the project.

The next step is to configure the queuing (topic and subscription in Cloud Pub/Sub) and set permissions so that Cloud IoT Core can use the queue.

10. Click the Navigation menu and scroll down to the BIG DATA section of the menu.

11. Select **Pub/Sub > Topics** to open the Topics panel, and then click the **CREATE TOPIC** link to open the **Create a topic** page. Give the topic a name and click **Create** when finished. For example:

    neodada-test-topic

12. The topic is added to the list in the Topics panel.

11. Click the topic to select it (checkmark in the box) and then click the pop-up menu to the right side to display a menu. From this menu, select **Permissions** to open an info panel for the topic's permissions (the PERMISSIONS tab of the info panel).
    - In the **Add members** entry field (on the PERMISSIONS tab of the info panel), add the following service account:
      cloud-iot@system.gserviceaccount.com

      This step enables Cloud IoT Core to access and use Cloud Pub/Sub. The protocol bridge's internal MQTT queuing mechanism will be able to send messages to Cloud Pub/Sub topics.
    - From the drop-down selector adjacent to the Add members field, select the following role:
      Pub/Sub Publisher
    - Click **Add** to save the topic and its permission settings.

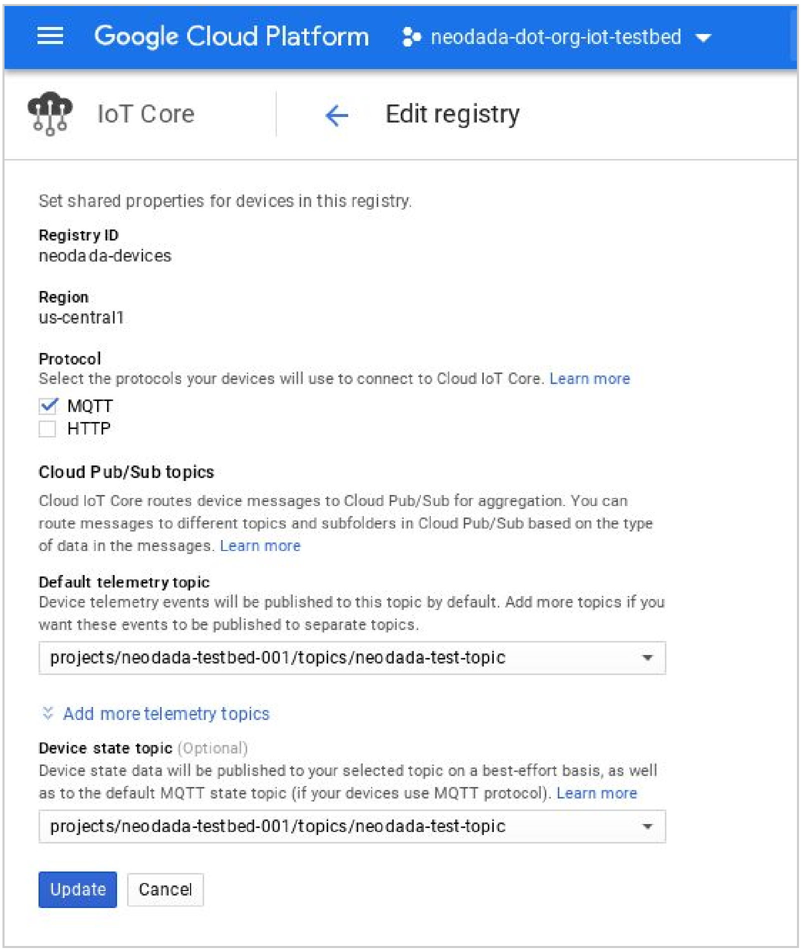
Create and configure a device registry

The device registry is Cloud IoT Core mechanism for managing and obtaining data from devices. Each registry is associated with a specific region and contains details about all the devices that can connect to it. Currently, US-Central1 is the only region available for North America.

1. Click the Navigation menu and scroll down to the BIG DATA section of the menu.

2. Select **IoT Core**.
    - If the IoT Core has no registries defined yet, you are prompted to create one. Click **Create a device registry** to open the **Create a registry** page.
    - If any registries already exist, the Device registries page displays. Click the **CREATE DEVICE REGISTRY** link to open the **Create a registry** page.
3. In the Registry ID field, enter a meaningful name for the registry.
    - For the **Region**, select from those available in the drop-down selector.
    - For **Protocol**, select MQTT, HTTP, or both.
        - For **Default telemetry topic**, select the topic created in the steps above from the drop-down selector.
        - For **Device state topic**, select the same topic.

3. Click **Create** to save the settings. Here's a completed example (shown in Edit mode, after creation):



Set shared properties for devices in this registry.

Registry ID
neodada-devices

Region
us-central1

Protocol
Select the protocols your devices will use to connect to Cloud IoT Core. Learn more

- [✓] MQTT
- [ ] HTTP

Cloud Pub/Sub topics
Cloud IoT Core routes device messages to Cloud Pub/Sub for aggregation. You can route messages to different topics and subfolders in Cloud Pub/Sub based on the type of data in the messages. Learn more

Default telemetry topic
Device telemetry events will be published to this topic by default. Add more topics if you want these events to be published to separate topics.

projects/neodada-testbed-001/topics/neodada-test-topic

Add more telemetry topics

Device state topic (Optional)
Device state data will be published to your selected topic on a best-effort basis, as well as to the default MQTT state topic (if your devices use MQTT protocol). Learn more

projects/neodada-testbed-001/topics/neodada-test-topic

This screenshot shows a test setup in which device **telemetry data** and device **state data** both target the same topic:

```
projects/neodada-testbed-001
/topics/neodada-test-topic
```

For a non-test use case, you may want to set up separate topics for each of these types of data.

Also note that the Topic name as shown in this interface is not the name that should be specified in your code. For telemetry data, you must prepend the topic with the following:

```
/devices/{device-id}/events
```

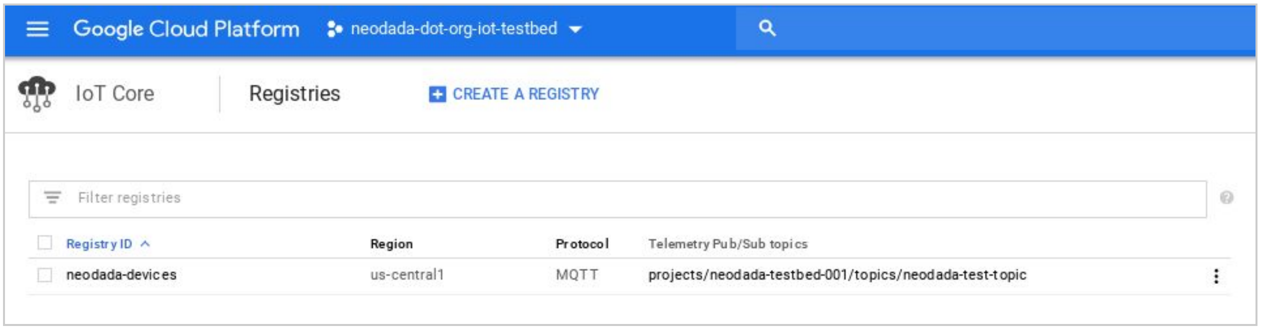
For example:

```
/devices/lab-rasp-pi-device/
events/projects/neodada-test
bed-001/topics/neodada-test-
topic
```

Data from each device in a given registry is thus distinguishable from other devices in the same registry.

Add devices to the registry

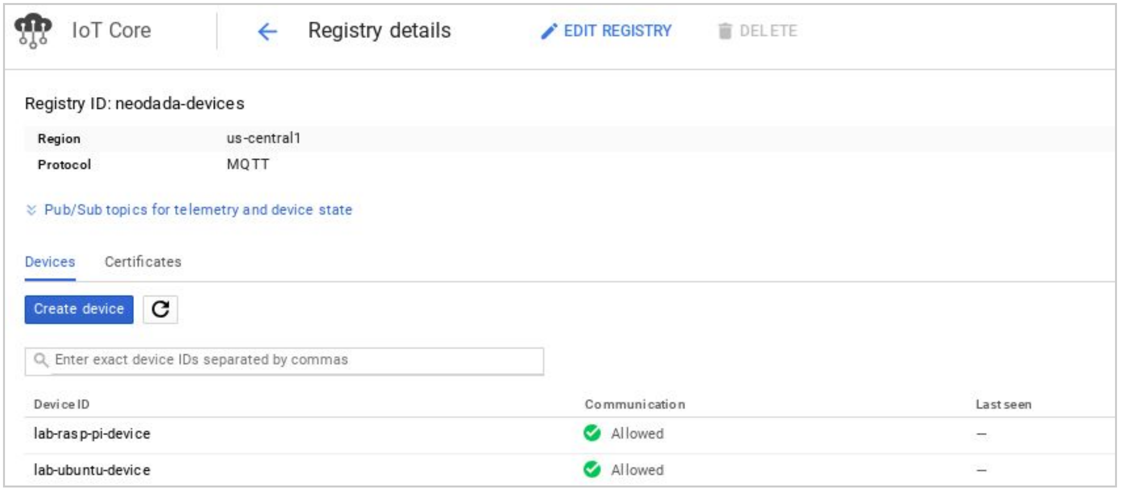
You can now add one or more devices to this registry. Assuming that the GCP Console is open to the IoT Core > Registries page:



Registry ID	Region	Protocol	Telemetry Pub/Sub topics
neodada-devices	us-central1	MQTT	projects/neodada-testbed-001/topics/neodada-test-topic

To add a device to the Registry:

1. Double-click on the name (Registry ID) in the list to open the Registry details page:


IoT Core
Registry details
EDIT REGISTRY
DELETE

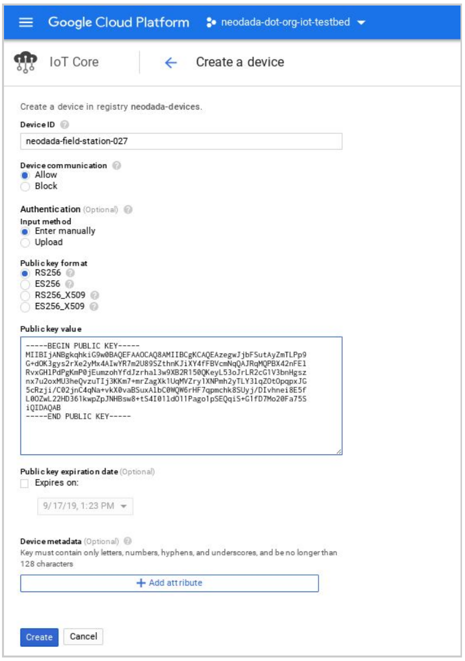
Registry ID: neodada-devices

Region	us-central1
Protocol	MQTT

Pub/Sub topics for telemetry and device state
Devices
Certificates
Create device
Enter exact device IDs separated by commas

Device ID	Communication	Last seen
lab-rasp-pi-device	✓ Allowed	–
lab-ubuntu-device	✓ Allowed	–

2. Click **Create device** to open the **Create a device** entry page.
   - In the **Device ID** field, enter a name for the device.
   - For **Authentication**, choose how you will input the device's public key or its certificate if you are using that mechanism. Select the **Public key format** that was used to create the public/private key pair or the certificate. You must have the public key or certificate available to be uploaded or to copy/paste:
     - Select **Upload** to select the file on your local file system in the Public key value field (click Browse to navigate to the file and select it).
     - Select **Enter manually** to copy/paste the content from the PEM file into the Public key value text entry field:



Create a device in registry neodada-devices.

Device ID: neodada-field-station-027

Device communication: [✓ Allow] Block

Authentication (Optional)
Input method: [✓ Enter manually] Upload

Public key format: [✓ RS256] ES256 RS256_X509 ES256_X509

Public key value

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzegwJjbFSutAyZmTLPp9
G+dOK3gys2rXe2yMx4AIwYR7m2U89SZthnKJiXY4fFBVcmNqQAJRqMQPBX42nFE1
RvxGH1PdPgKmP0jEumzohYfdJzrhal3w9XB2R150QKeyL53oJrLR2cG1V3bnHgsz
nx7u2oxMU3heQvzuTIj3KKm7+mrZagXk1UqMVZry1XNPmh2yTLY31qZOtOpqpxJG
5cRzji/C02jnC4qNa+vkX0vaBSuxA1bC0WQW6rHF7qpmchk8SUyj/DIvhnei8E5f
L0OZwL22HD361kwpZpJNHBsw8+tS4I011dO11Pago1pSEQqiS+G1fD7Mo20Fa75S
iQIDAQAB
-----END PUBLIC KEY-----
```

Public key expiration date (Optional)
☐ Expires on: 9/17/19, 1:23 PM

Device metadata (Optional)
Key must contain only letters, numbers, hyphens, and underscores, and be no longer than 128 characters

    - Optionally, set a date and time in the future at which the key will no longer be valid. (This feature and the fact that up to three different keys can be stored for each device at any time is intended to support key rotation.)

3. Click **Create** to add the device to the registry.

Note	GCP Console will alert you if the Public key format selected in the page is incorrect for the actual PEM file name/path or its content uploaded or pasted into the page. You cannot add a device if its public key is not formatted as specified.

At this point, your device has been set up in the registry and you can test your configuration by running an example. See Testing the setup by running sample code for details.

Using gCloud for setup tasks

If you have completed all Preliminary Setup steps for the device including the steps to Install and initialize the Google Cloud SDK, you can configure Cloud IoT Core using the gCloud command-line tool. The general process is as follows:

- Create a project
- Configure the project
- Create a device registry
- Register your device

The steps below assume you have created the private/public key pair and that these are available on the system that is being used to execute the gCloud commands. See Using OpenSSL to create public/private key pair for details about how to create and do that now, before continuing.

See the gCloud Overview and the gCloud Reference for more information about using gCloud.

For the example run-through in this section, the Google Cloud SDK is running on the device (a Raspberry Pi 3+ with 32-GB microSD and Raspbian OS) and all keys and other artifacts were also created on the device, along with the Python example (cloudiot_mqtt_example.py). See the IoT Device Integration Cheatsheet for a summary of all commands needed to integrate a device.

Create a project

See gCloud reference > projects for details.

gcloud projects create <project-id> --name=<project-name>

For example:

```
./google-cloud-sdk/bin/gcloud create neodada-testbed-001 --name=neodada-dot-org-deploy
```

Progress displays in the terminal, for example:

```
Create in progress for
[https://cloudresourcemanager.googleapis.com/v1/projects/neodada-project-001].

Waiting for [operations/cp.5349365235427647091] to finish...done.
```

Many of the examples shown in this section assume that the gCloud client has been configured for the specific project. If necessary, set your gCloud installation to point to the new project:

gcloud config set project <project-id>

For example:

```
./google-cloud-sdk/bin/gcloud config set project neodada-testbed-001
```

Configure the project for Cloud IoT Core and Cloud Pub/Sub

The project has now been created but it still needs to be granted appropriate permissions so that Cloud Pub/Sub can send events to Cloud IoT Core. To do that:

gcloud projects add-iam-policy-binding <project-id> --member=<user>|<group>|serviceAccount:<email>|<domain>:<domain> --role=<role>

For example:

```
gcloud projects add-iam-policy-binding neodada-testbed-001
--member=serviceAccount:cloud-iot@system.gserviceaccount.com--role='roles/pubsub.publisher'
```

Note that you can use the **gcloud services list** command to see the full list of APIs that can be enabled from gCloud. Having done that, you can now use the **gcloud services enable** command as shown here:

gcloud services enable cloudiot.googleapis.com

gcloud services enable pubsub.googleapis.com

For example:

```
pi@lab-pi:~ $ ./google-cloud-sdk/bin/gcloud services enable pubsub.googleapis.com
Operation "operations/acf.e6050272-e2f4-4753-b689-50a39062e4e8" finished successfully.

pi@lab-pi:~ $ ./google-cloud-sdk/bin/gcloud services enable cloudiot.googleapis.com
Operation "operations/acf.ac6448a0-737b-49b4-9bad-29168658655d" finished successfully.
```

See gCloud reference > services > enable for details.

The next step is to configure the queuing (topic and subscription in Cloud Pub/Sub).

gcloud pubsub topics create *<topic-name>*

gcloud pubsub create subscriptions *<subscription-name>* --topic=*<topic-name>*

For example:

```
pi@tech-lab-pi:~/dev $ gcloud pubsub topics create neodada-test-topic
Created topic [projects/neodada-testbed-001/topics/neodada-test-topic].
pi@tech-lab-pi:~/dev $ gcloud pubsub subscriptions create neodada-test-sub
--topic=neodada-test-topic
Created subscription [projects/neodada-testbed-001/subscriptions/integration-test-sub].
pi@tech-lab-pi:~/dev $
```

Create and configure a device registry

Cloud IoT Core uses a registry as the mechanism distinguish one set of devices from another, and within each collection, to distinguish one device from another. Telemetry messages from each device are queued up to a specific topic as configured for the registry. With gCloud, you can create the device registry and specify the topic to associate with the registry as follows:

gcloud iot registries create *<registry-name>* --region=us-central1 --project=<project-id> --event-notification-config=topic=*<topic-name>*

For example:

```
gcloud iot registries create neodada-devices --region=us-central1 --project=neodada-testbed-001
--event-notification-config=topic=neodada-test-topic
```

See gCloud reference > iot > registries for details.

Add devices to the registry

This step assumes you have the public key for the device available. The gCloud command accepts the path to the public key and adds it to the device configuration on Google Cloud. The device must have the matching private key at runtime.

In the gCloud command, specify whether RSA (type=RS256) or EC (elliptical curve, type=ES256) algorithm was used to create the key. The syntax varies by only the type field:

gcloud iot devices create *<device-name>* --project=*<project-name>* --region=us-central1 --registry=*<registry-name>* --public-key path=*<path-to-filename_pub.pem>*, type=[rs256 | es256]

For example, to add a device to the registry with its public key:

```
gcloud iot devices create neodada-field-station-027 --project=my-iot-project
--region=us-central1 --registry=neodada-devices --public-key path=neodada_pub.pem,type=rs256
```

Alternatively, if the public key uploaded to Google Cloud Platform IoT registry was created using an elliptical curve (EC) algorithm, the statement might be as shown here (assuming the companion private key was created using the same algorithm as detailed in Create a public/private key pair for the device).

```
gcloud iot devices create neodada-field-station-028 --project=my-iot-project
--region=us-central1 --registry=neodada-devices --public-key path=neodada_ecpub.pem,type=es256
```

At this point, your device has been set up in the registry and you can test your configuration by running an example. See Testing the setup by running sample code for details.

Testing the setup by running sample code

To verify your completed setup, you can run one of the samples available on Google Cloud Platform GitHub. Samples are available for both MQTT and HTTP and using various clients.

The Google Cloud Platform repository on GitHub has many other examples, including the End-to-End example (cloudiot_pubsub_example_mqtt_device.py) demonstrated in An Overview Cloud IoT Core (Google I/O '18). Here are two direct links to the **iot/api-client** samples for Java and Python, respectively.

- GoogleCloudPlatform/java-docs-samples
- GoogleCloudPlatform/python-docs-samples

The example run-through in this section uses the **cloudiot_mqtt_example.py** script from the GoogleCloudPlatform/python-docs-samples directory.

To run any of the MQTT example scripts, you need Google's certificates available on the device to verify the certificate received in response to device's connection request can be trusted. To obtain these root certificates, downloading Google's **roots.pem** using one of the following:

wget https://pki.goog/roots.pem

curl https://pki.goog/roots.pem > roots.pem

For many of the MQTT examples, the **ca_cert** parameter is set to **roots.pem** by default, and the scripts typically look for the file in the local path. How you handle this for your own code is up to you.

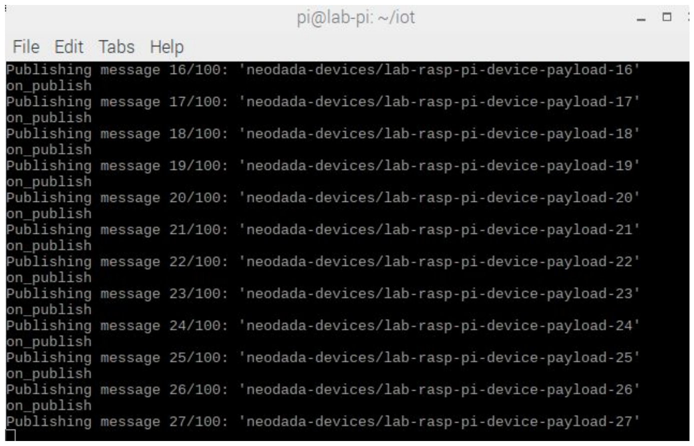
If you've set up your device properly, you can run the example script as follows:

python cloudiot_mqtt_example_device.py --project_id=<your-project-id> --registry_id=<your-registry> --cloud_region=us-central1 --device_id=<your-device-id> --private_key_file=<your_private-key>.pem --algorithm=<RS256 | ES256> --message_type=<event | state>

For example:

```
python cloudiot_mqtt_example_device.py --project_id=neodada-testbed-001
--registry_id=neodada-devices --cloud_region=us-central1 --device_id=lab-rasp-pi-device
--private_key_file=labdevice_priv.pem --algorithm=RS256 --message_type=event
```

The screenshot below shows a successful sample run:



```
Publishing message 16/100: 'neodada-devices/lab-rasp-pi-device-payload-16'
on_publish
Publishing message 17/100: 'neodada-devices/lab-rasp-pi-device-payload-17'
on_publish
Publishing message 18/100: 'neodada-devices/lab-rasp-pi-device-payload-18'
on_publish
Publishing message 19/100: 'neodada-devices/lab-rasp-pi-device-payload-19'
on_publish
Publishing message 20/100: 'neodada-devices/lab-rasp-pi-device-payload-20'
on_publish
Publishing message 21/100: 'neodada-devices/lab-rasp-pi-device-payload-21'
on_publish
Publishing message 22/100: 'neodada-devices/lab-rasp-pi-device-payload-22'
on_publish
Publishing message 23/100: 'neodada-devices/lab-rasp-pi-device-payload-23'
on_publish
Publishing message 24/100: 'neodada-devices/lab-rasp-pi-device-payload-24'
on_publish
Publishing message 25/100: 'neodada-devices/lab-rasp-pi-device-payload-25'
on_publish
Publishing message 26/100: 'neodada-devices/lab-rasp-pi-device-payload-26'
on_publish
Publishing message 27/100: 'neodada-devices/lab-rasp-pi-device-payload-27'
```

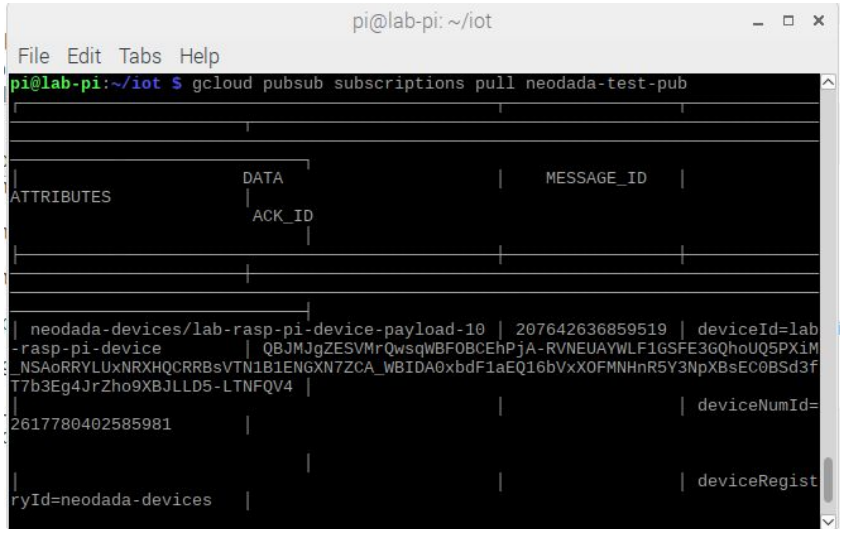
To confirm that the messages sent by the script are actually being queued to Cloud Pub/Sub, you can use the gCloud command to pull from the subscriptions:

gcloud pubsub subscriptions pull <subscription-name>

For example:

```
gcloud pubsub subscriptions pull neodada-test-pub
```

The console should display some of the payload content that was posted to the topic associated with the subscription, as shown here:

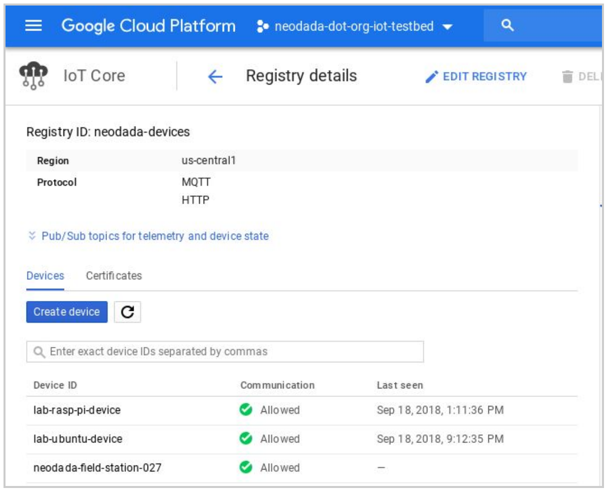


```
pi@lab-pi:~/iot $ gcloud pubsub subscriptions pull neodada-test-pub
|                      DATA                      |    MESSAGE_ID    |
ATTRIBUTES              |
                         ACK_ID
                              |
| neodada-devices/lab-rasp-pi-device-payload-10 | 207642636859519 | deviceId=lab
-rasp-pi-device         | QBJMJgZESVMrQwsqWBFOBCEhPjA-RVNEUAYWLF1GSFE3GQhoUQ5PXiM
_NSAoRRYLUxNRXHQCRRBsVTN1B1ENGXN7ZCA_WBIDA0xbdF1aEQ16bVxXOFMNHnR5Y3NpXBsEC0BSd3f
T7b3Eg4JrZho9XBJLLD5-LTNFQV4 |
|                                               |                 | deviceNumId=
2617780402585981        |

                              |
|                                               |                 | deviceRegist
ryId=neodada-devices    |
```

Troubleshooting

Various issues can arise, and you may not always see useful error messages. The example script may seem to run just fine, but if the device doesn't have a **Last seen** timestamp listed in the GCP Console (**Registry details** page), then something is not working.


Google Cloud Platform
neodada-dot-org-iot-testbed
IoT Core
Registry details
EDIT REGISTRY
DEL
Pub/Sub topics for telemetry and device state
Devices
Certificates
Create device
Enter exact device IDs separated by commas

Registry ID: neodada-devices

Region	us-central1
Protocol	MQTT HTTP

Device ID	Communication	Last seen
lab-rasp-pi-device	✓ Allowed	Sep 18, 2018, 1:11:36 PM
lab-ubuntu-device	✓ Allowed	Sep 18, 2018, 9:12:35 PM
neodada-field-station-027	✓ Allowed	–

For example, the Project Name might erroneously be passed as the Project_ID without raising any error message. The only indication you may have that something is not working is an empty date-time field (-) for the **Last seen** column.

A good place to start is to check TLS connectivity from the device to GCP. See Troubleshooting in the Google Cloud documentation for many other common issues and solutions.

Check TLS Connectivity

Use OpenSSL from the client device to query the MQTT endpoint.

openssl s_client -connect mqtt.googleapis.com:8883

If TLS is in place correctly, you should see an entire stream of detail including the server certificate (the Base64-encoded ASCII string), the certificate chain, and many other details, including the TLS version returned from the endpoint. For example:

```
pi@lab-pi:~/iot $ openssl s_client -connect mqtt.googleapis.com:8883
CONNECTED(00000003)
depth=2 OU = GlobalSign Root CA - R2, O = GlobalSign, CN = GlobalSign
verify return:1
depth=1 C = US, O = Google Trust Services, CN = Google Internet Authority G3
verify return:1
depth=0 C = US, ST = California, L = Mountain View, O = Google LLC, CN
= mqtt.googleapis.com
verify return:1
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google LLC/CN=mqtt.googleapis.com
   i:/C=US/O=Google Trust Services/CN=Google Internet Authority G3
 1 s:/C=US/O=Google Trust Services/CN=Google Internet Authority G3
```

```
   i:/OU=GlobalSign Root CA - R2/O=GlobalSign/CN=GlobalSign
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEjDCCA3SgAwIBAgIIb4AVAEu46mMwDQYJKoZIhvcNAQELBQAwVDELMAkGA1UE
BhMCVVMxHjAcBgNVBAoTFUdvb2dsZSBUcnVzdCBTZXJ2aWNlczElMCMGA1UEAxMc
R29vZ2xlIEludGVybmV0IEF1dGhvcml0eSBHMzAeFw0xODA4MjEwODA1MDBaFw0x
ODExMTMwODA1MDBaMG0xCzAJBgNVBAYTAlVTMRMwEQYDVQQIDApDYWxpZm9ybmlh
MRYwFAYDVQQHDA1Nb3VudGFpbiBWaWV3MRMwEQYDVQQKDApHb29nbGUgTExDMRww
. . .
-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Mountain View/O=Google LLC/CN=mqtt.googleapis.com
issuer=/C=US/O=Google Trust Services/CN=Google Internet Authority G3
---
No client certificate CA names sent
Peer signing digest: SHA256
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 2964 bytes and written 261 bytes
Verification: OK
---
New, TLSv1.2, Cipher is ECDHE-RSA-CHACHA20-POLY1305
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
. . .
```

Last updated: 16-October-2018