Streaming IoT Data to Cloud Storage

Author: Francesco Desimoni
Source: https://tcc.uniupo.it

Introduction

Overview

This tutorial is an edited version of this lab available on Qwiklabs.
In this lab you will learn how to configure Cloud IoT Core and Cloud Pub/Sub to create a Pub/Sub topic and registry on Google Cloud. You will use this topic to ingest data streaming from a simulated device. You will receive hands-on practice with the following services:

- Cloud Storage
- Cloud Pub/Sub
- Compute Engine
- IoT Core
- Dataflow

The actions you will perform are mostly a simulation of a real scenario but you will be able to use the experience acquired in this lab for real projects. For example the same code you will run on a virtual machine in this tutorial can be executed on a real device with sensors.

Support

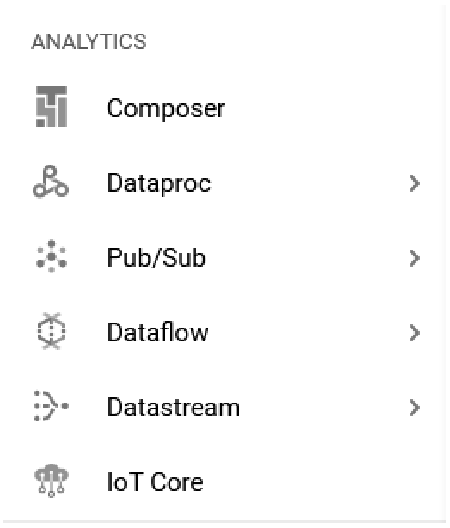
The field of cloud computing is constantly changing and improving and, as new technologies are introduced, the steps to take in tutorials such as this one may change. While the basic instructions and the goal of this tutorial will most likely stay the same, feel free to contact us for any questions or suggestions at tcc@uniupo.it.

Tutorial

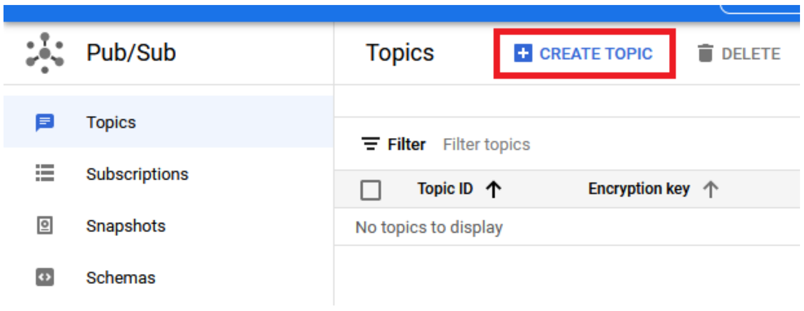
Create a Pub/Sub topic

First you need to create a Pub/Sub Topic for your streaming data.

1. On the **Navigation menu**, look for the **Analytics** section and click **Pub/Sub** > **Topics**.

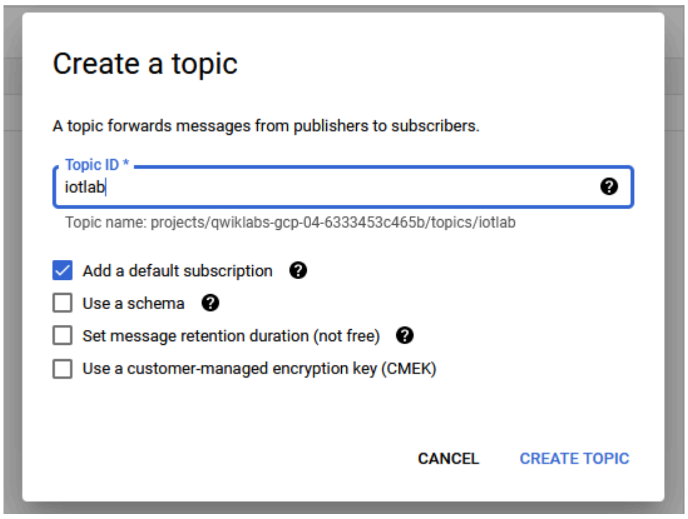


2. Click **Create topic**.



No topics to display

3. Type *iotlab* in **Topic ID** section.



Create a topic

A topic forwards messages from publishers to subscribers.

Topic ID *
iotlab

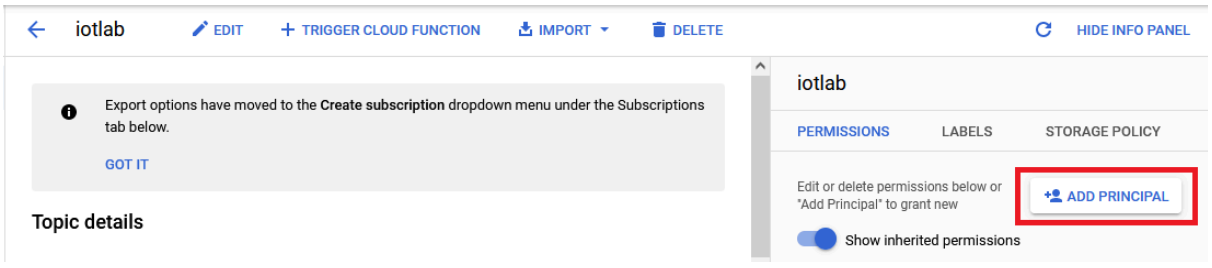
Topic name: projects/qwiklabs-gcp-04-6333453c465b/topics/iotlab

- [x] Add a default subscription
- [ ] Use a schema
- [ ] Set message retention duration (not free)
- [ ] Use a customer-managed encryption key (CMEK)

4. Leave the checkboxes as they are.
5. Click **Create topic**.

You now have a Pub/Sub Topic. To allow the project to publish this topic, you need to add the project as a member/publisher.

To add members, click **Add principal**.



Export options have moved to the **Create subscription** dropdown menu under the Subscriptions tab below.

GOT IT

Topic details

iotlab

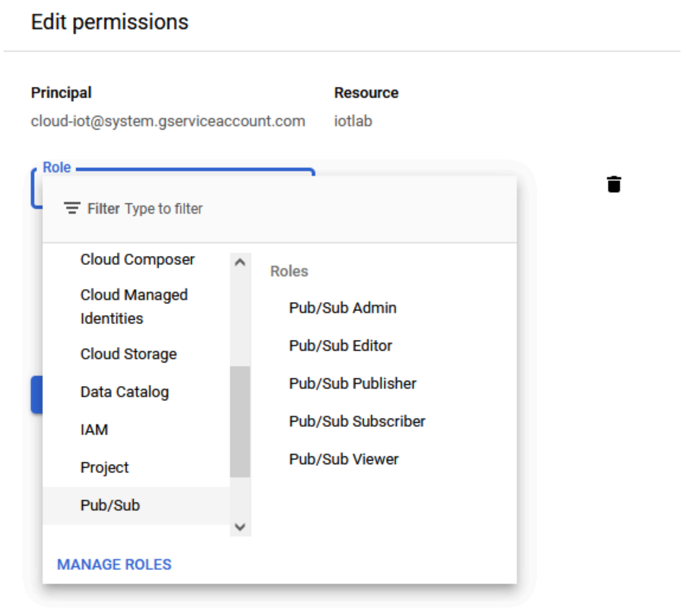
PERMISSIONS LABELS STORAGE POLICY

Edit or delete permissions below or "Add Principal" to grant new

ADD PRINCIPAL

Show inherited permissions

Add the project as a member to the topic: *cloud-iot@system.gserviceaccount.com*. Select the role of **Pub/Sub Publisher**, and then click **Save** to add the member.


Edit permissions

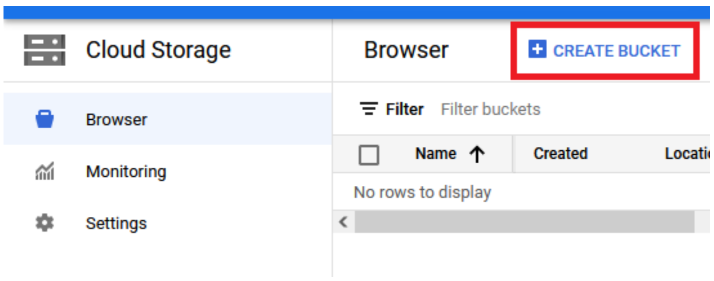
Principal: cloud-iot@system.gserviceaccount.com

Resource: iotlab

Create a storage

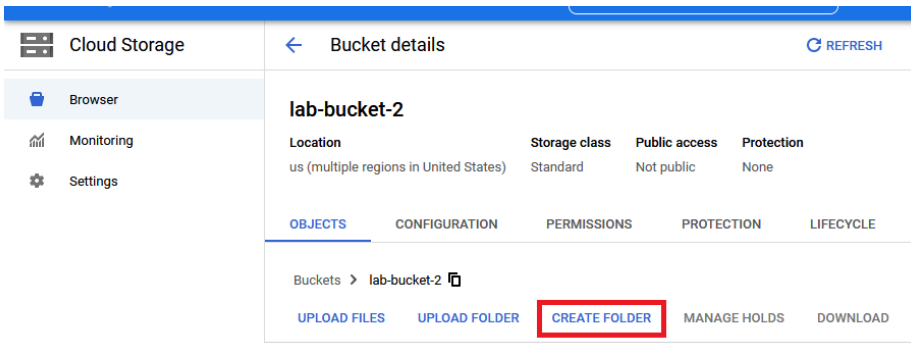
You need to create a storage bucket and folder to store the data streaming from the IoT device.

1. From the **Navigation menu**, in the **Storage** section, click **Cloud Storage** > **Browser**.
2. Click **Create bucket**.



No rows to display

3. Bucket names must be unique. Enter a unique bucket name, and then click **Create**.
4. In the bucket you just created, click **Create folder**.

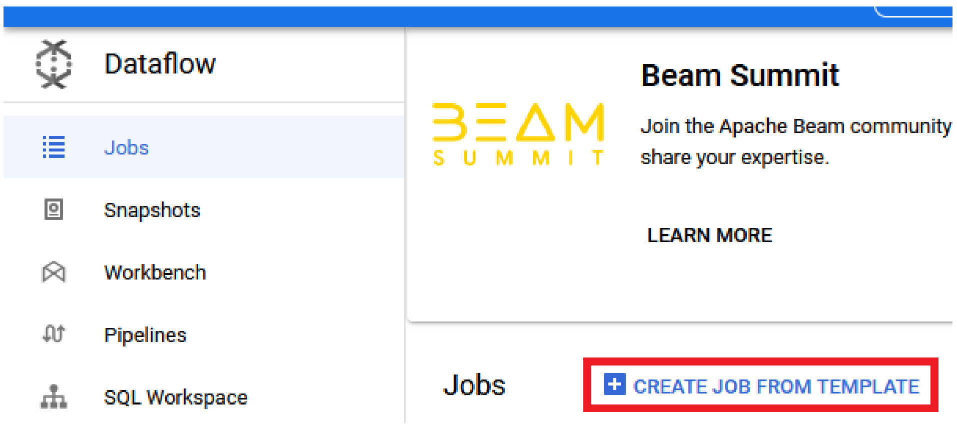


5. For **Name**, type **Sensor-Data**.
6. Click **Create**.

Start a Dataflow job

You now have a device publishing data, and your Google Cloud Project is authorized to receive this data. Now you can start a Dataflow job to save the data to your bucket.

1. On the **Navigation menu**, in the **Analytics** section, click **Dataflow**.
2. Click **Create job from template**.



3. Enter the following values in the template.

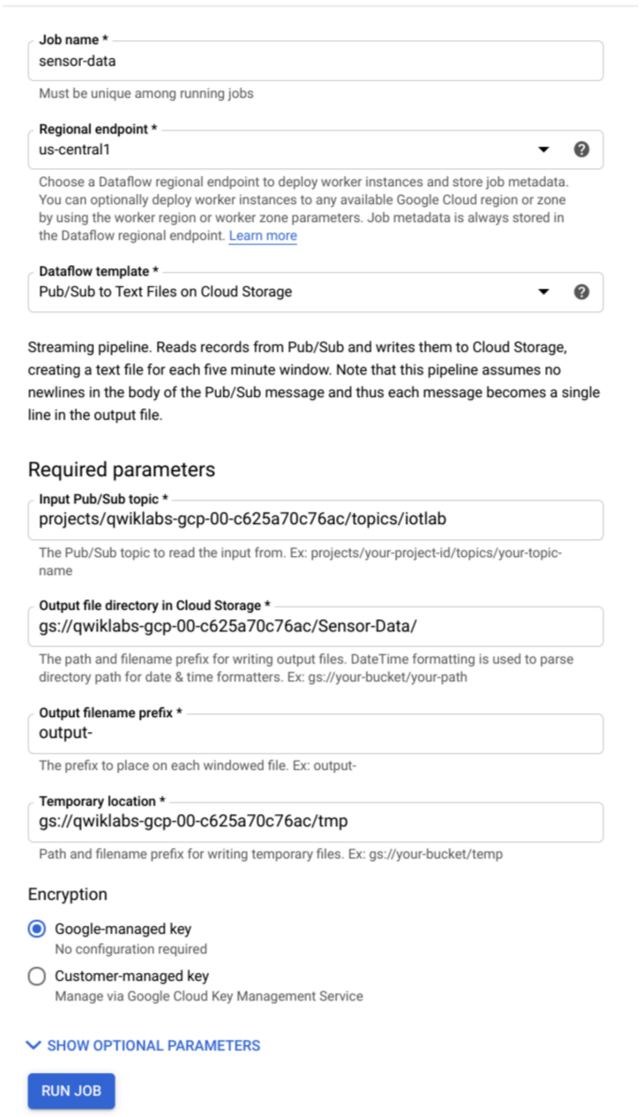
Property	**Value**
Job name	*sensor-data*
Regional endpoint	*us-central1*

Cloud Dataflow template	*Pub/Sub to Text Files on Cloud Storage*

4. The template page will expand to display a series of textboxes. Some of the textboxes are optional and some are required. You will only modify the required textboxes.

Property	**Value**
Input Pub/Sub topic	*projects/**<project-id>**/topics/iotlab*
Output file directory in Cloud Storage	*gs://**<bucket-name>**/Sensor-Data/*
Output filename prefix	*output-*
Temporary Location	*gs://**<bucket-name>**/tmp*

Your template should resemble the following:


Job name *
sensor-data
Must be unique among running jobs

Regional endpoint *
us-central1
Choose a Dataflow regional endpoint to deploy worker instances and store job metadata. You can optionally deploy worker instances to any available Google Cloud region or zone by using the worker region or worker zone parameters. Job metadata is always stored in the Dataflow regional endpoint. Learn more

Dataflow template *
Pub/Sub to Text Files on Cloud Storage

Streaming pipeline. Reads records from Pub/Sub and writes them to Cloud Storage, creating a text file for each five minute window. Note that this pipeline assumes no newlines in the body of the Pub/Sub message and thus each message becomes a single line in the output file.

Required parameters

Input Pub/Sub topic *
projects/qwiklabs-gcp-00-c625a70c76ac/topics/iotlab
The Pub/Sub topic to read the input from. Ex: projects/your-project-id/topics/your-topic-name

Output file directory in Cloud Storage *
gs://qwiklabs-gcp-00-c625a70c76ac/Sensor-Data/
The path and filename prefix for writing output files. DateTime formatting is used to parse directory path for date & time formatters. Ex: gs://your-bucket/your-path

Output filename prefix *
output-
The prefix to place on each windowed file. Ex: output-

Temporary location *
gs://qwiklabs-gcp-00-c625a70c76ac/tmp
Path and filename prefix for writing temporary files. Ex: gs://your-bucket/temp

Encryption
● Google-managed key
No configuration required
○ Customer-managed key
Manage via Google Cloud Key Management Service

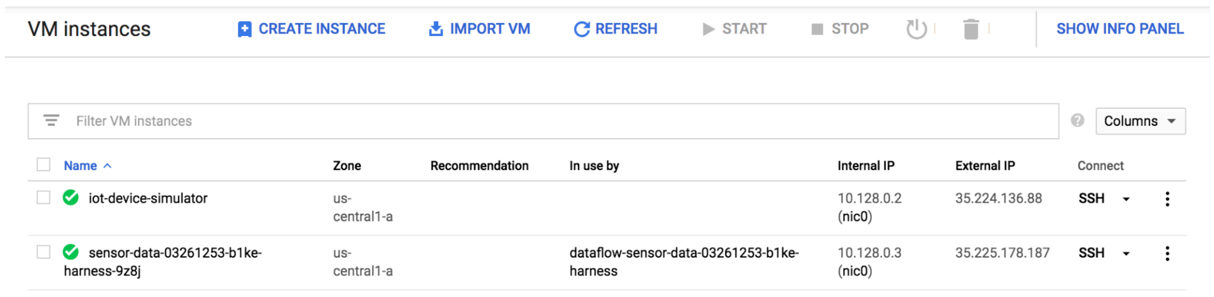
5. Once you have verified that all the fields are properly filled out, click **Run job**.

NOTE: If the Dataflow job fails the first time, please re-create and run the job.

Device simulator VM

In your project, a pre-provisioned VM instance named **iot-device-simulator** will let you run instances of a Python script that emulate an MQTT-connected IoT device. Before you emulate the devices, you will also use this VM instance to populate your Cloud IoT Core device registry. If you are not following along in a lab environment you can launch a virtual machine, use a physical one or use a real device instead.

1. In the Cloud Console, go to the Navigation **menu**, look for the **Compute** section and click on **Compute Engine**> **VM Instances**. You'll see your VM instance listed as *iot-device-simulator*.


VM instances

Name	Zone	Recommendation	In use by	Internal IP	External IP	Connect
iot-device-simulator	us-central1-a			10.128.0.2 (nic0)	35.224.136.88	SSH
sensor-data-03261253-b1ke-harness-9z8j	us-central1-a		dataflow-sensor-data-03261253-b1ke-harness	10.128.0.3 (nic0)	35.225.178.187	SSH

2. Click the **SSH** drop-down arrow and select **Open in browser window**.
3. In your SSH session on **iot-device-simulator**, enter this command to remove the default Google Cloud SDK installation. (In subsequent steps, you will install the latest version, including the beta component.)

```
sudo apt-get remove google-cloud-sdk -y
```

4. Now install the latest version of the Google Cloud SDK and accept all defaults by running:

```
curl https://sdk.cloud.google.com | bash
```

   Enter through all the prompts that follow.

5. End your SSH session on the **iot-device-simulator** VM instance:

```
exit
```

6. Start another SSH session on the **iot-device-simulator** VM instance.
7. Enter this command to make sure that the components of the SDK are up to date:

```
gcloud components update
```

If you get the error message "Command not found", you might have forgotten to exit your previous SSH session and start a new one. If you get a different error try closing the window and opening a new SSH session.

8. Enter the following command to install the beta components. Enter **Y** when prompted to continue:

```
gcloud components install beta
```

Enter through all the prompts that follow.

9. Enter this command to update the system's information about Debian Linux package repositories:

```
sudo apt-get update
```

10. Enter this command to make sure that various required software packages are installed:

```
sudo apt-get install python3-pip openssl git -y
```

11. Use **pip** to add needed Python components:

```
sudo pip3 install pyjwt paho-mqtt cryptography
```

12. Enter this command to add data to analyze during this lab:

```
git clone http://github.com/GoogleCloudPlatform/training-data-analyst
```

13. In your SSH session on the **iot-device-simulator** VM instance, run the following, adding your project ID as the value for **PROJECT_ID**:

```
export PROJECT_ID=
```

Your completed command will look like this:

```
export PROJECT_ID=qwiklabs-gcp-d2e509fed105b3ed
```

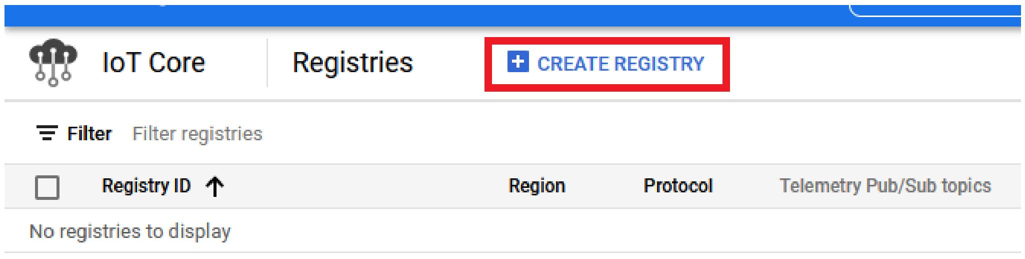
14. You must choose a region for your IoT registry. Set an environment variable containing the us-central1 region with the following command:

```
export MY_REGION=us-central1
```

Leave this SSH window open. You will come back to it after creating a device registry in the following step.

IoT Core

1. Return to the Cloud Console. From the Navigation menu, in the **Analytics** section, click **IoT Core**.
2. Click **Create registry**.



No registries to display

3. On the **Create a registry** page, specify the following, and leave the remaining settings as their defaults:

Property	**Value**
Registry ID	*iotlab-registry*
Region	*us-central1*
Select a Cloud Pub/Sub topic	*projects/**<project-id>**/topics/iotlab*

4. Click **Create**.

A device registry is a container of devices with shared properties. A registry can have one or more Pub/Sub Topics to which devices in the registry can publish data.

Create a Cryptographic Keypair

To allow IoT devices to connect securely to Cloud IoT Core, you must create a cryptographic keypair.

In your SSH session on the **iot-device-simulator** VM instance, enter these commands to create the keypair in the appropriate directory:

```
cd $HOME/training-data-analyst/quests/iotlab/
openssl req -x509 -newkey rsa:2048 -keyout rsa_private.pem \
    -nodes -out rsa_cert.pem -subj "/CN=unused"
```

This openssl command creates an RSA cryptographic keypair and writes it to a file called *rsa_private.pem*. When connecting to Cloud IoT Core, each device creates a JWT token signed with its private key, which Cloud IoT Core authenticates using the device's public key.

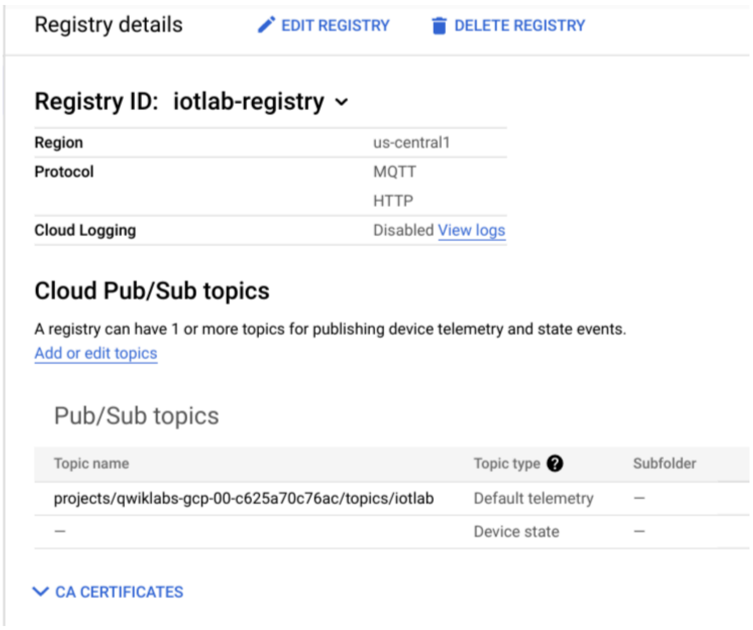
1. In your SSH session on the *iot-device-simulator*, type

```
cat rsa_cert.pem
```

2. The output will be your certificate. Select and copy the entire certificate (including the *begin/end certificate*).

```
----BEGIN CERTIFICATE-----
MIIC+DCCAeCgAwIBAgIJAOJikTScq9oPMA0GCSqGSIb3DQEBCwUAMBExDzANBgNV
BAMMBnVudXNlZDAeFw0xODA4MTMxNjQ2MTNaFw0xODA5MTIxNjQ2MTNaMBExD
zANBgNVBAMMBnVudXNlZDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggE
BAL+wLyITE5TjlH50I63ew3HdvoGty2aOpP04nMyOYZoooAw5o2rj5mkNb/hbkoMTkzo
6/5Jo0zgDYPVRpz2nGAhTfeQzPuvOfPZe7KPpZxYvmSN3pYT9kkiVo9pXwynG7q8kW
72Q9f0pffXS/VElPrC63Y9kcAgOyveZVX61qSokz4DVIj0Z6+1b1utxe2TnxR1q3Hce289
1re6qnxYp6Yuw0gVYtn8HdgEKKMqeSozqJP7dq8EvNkwY8BAUFU2NmuvwK2Z6hB1E
u0DImyhtKRxZ4pUbWuefC+P6GU2fB3rp4pR9Lc7xd5BuWXHgR6f0lV57elL9f1Q/iXippP
8RjhMCAwEAAaNTMFEwHQYDVR0OBBYEFF7808W+vP7vbgg6cS5Fky9xCstNMB8GA
1UdIwQYMBaAFF7808W+vP7vbgg6cS5Fky9xCstNMA8GA1UdEwEB/wQFMAMBAf8w
DQYJKoZIhvcNAQELBQADggEBAD9mSbWQRz8QHI947gGSMrsA+aO4dgWIujkypFw/p
7gSefleCCwGV4Wpfq6zoIjru9bnciWRLHZMKVbhptBDseyBnoPXxnJMgVYBAVzRRMhT
qPeo146Pv99dn3c310M2tkpQeQzP/wE9XFVqEud2sZCKXgXtydIsyTEX3wmG9s9m7f
6TJDknvJltOj1R7m+xO6GHPebK29x/r+LzPuYjIDYoG+mxLQUltDOM3v8QwZ4bneo+HI
BZX6FOBRb+x/fEE3EANCY3J5sKwRCxxXJ6l/Mts7aLUE6MrT8BM0n1fxnY7BX+6dvsJ
H/OeONG2tk3Y0ci/ly245NQyurqa3x35Ws=
-----END CERTIFICATE-----
```

3. Your IoT Core Console page should resemble the following:



Registry ID: iotlab-registry

Region	us-central1
Protocol	MQTT HTTP
Cloud Logging	Disabled View logs

Cloud Pub/Sub topics

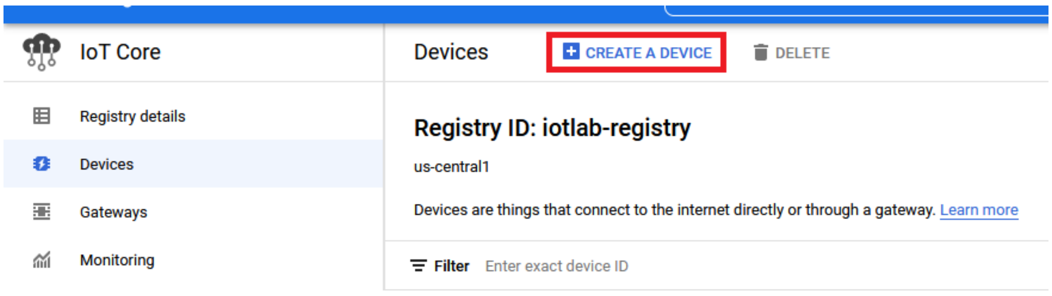
A registry can have 1 or more topics for publishing device telemetry and state events.
Add or edit topics

Pub/Sub topics

Topic name	Topic type	Subfolder
projects/qwiklabs-gcp-00-c625a70c76ac/topics/iotlab	Default telemetry	–
–	Device state	–

From the left-hand menu, select **Devices** and ensure that the Registry ID is set to **iotlab-registry**.

4. Click **Create a device** from the top menu and enter in the following:



Registry ID: iotlab-registry

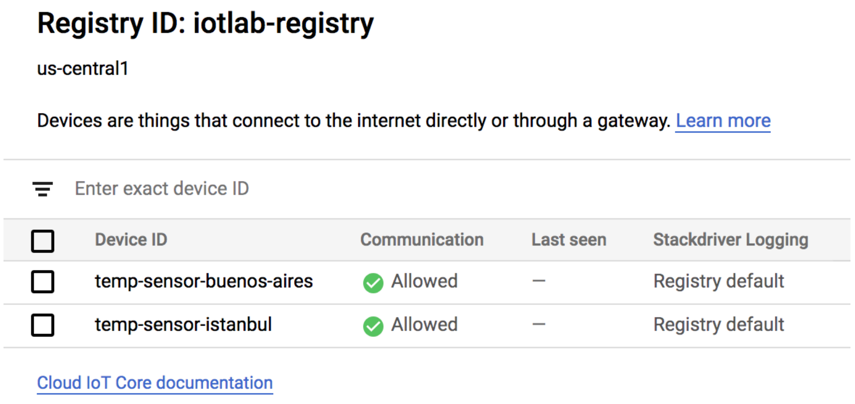
us-central1

Devices are things that connect to the internet directly or through a gateway. Learn more

Property	**Value**
Device ID	*temp-sensor-istanbul*
Authentication	*Enter manually*
Public key format	*RS256_X509*
Public key value	**Paste the certificate that you copied**

Click on the **Communication, Cloud logging, Authentication** option to see advanced fields.

5. Click **Create**.
6. You will now add a second device. Hit the back button to return to the device menu and once again click **Create a device**. Then enter the settings.
7. Hit the back button to return to the device overview. Your page should resemble the following:



Registry ID: iotlab-registry

us-central1

Devices are things that connect to the internet directly or through a gateway. Learn more

	Device ID	Communication	Last seen	Stackdriver Logging
☐	temp-sensor-buenos-aires	✓ Allowed	—	Registry default
☐	temp-sensor-istanbul	✓ Allowed	—	Registry default

Run simulated devices

1. In your SSH session on the **iot-device-simulator** VM instance, enter these commands to download the CA root certificates from pki.google.com to the appropriate directory:

```
cd $HOME/training-data-analyst/quests/iotlab/
curl -LO https://pki.google.com/roots.pem
```

2. Enter this command to run the first simulated device:

```
python3 cloudiot_mqtt_example_json.py \
   --project_id=$PROJECT_ID \
   --cloud_region=$MY_REGION \
   --registry_id=iotlab-registry \
   --device_id=temp-sensor-istanbul \
   --private_key_file=rsa_private.pem \
   --message_type=event \
   --algorithm=RS256 --num_messages=200 > istanbul-log.txt 2>&1 &
```

It will continue to run in the background.

3. Enter the command to run the second simulated device.

```
python3 cloudiot_mqtt_example_json.py \
   --project_id=$PROJECT_ID \
   --cloud_region=$MY_REGION \
   --registry_id=iotlab-registry \
   --device_id=temp-sensor-buenos-aires \
   --private_key_file=rsa_private.pem \
   --message_type=event \
   --algorithm=RS256 \
   --num_messages=200
```

Cloud IoT Core supports two protocols for device connection and communication: MQTT and HTTP. Devices communicate with Cloud IoT Core across a "bridge", either the MQTT bridge or the HTTP bridge.
Telemetry data will flow from the simulated devices through Cloud IoT Core to your Cloud Pub/Sub topic. In turn, your Dataflow job will read messages from your Pub/Sub topic and write their contents to your Cloud Storage bucket.

NOTE: Wait for a few seconds for the logs to get generated in the file.

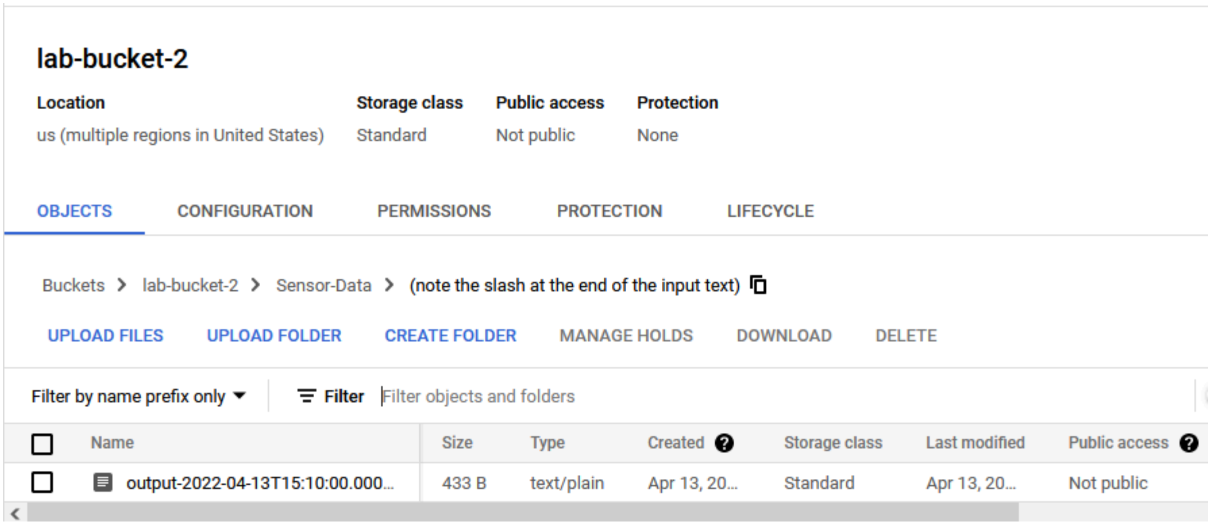
Dataflow is collecting the data published by Pub/Sub and saving it in output files in the bucket and folder specified in the job template. The files are written every 5 minutes, and each begins with the prefix specified in the job template.

1. Return to the Cloud Console. Open the Navigation menu and select **Cloud Storage**.
2. Select the bucket you created for this project.

3. Select the folder **Sensor-Data**. Dataflow is writing the data from the device to this folder.

Files are written every five minutes. If the folder is empty, wait about 5-10 minutes and periodically click the Refresh Bucket button.

4. After refreshing, you will soon see an output file in your directory:



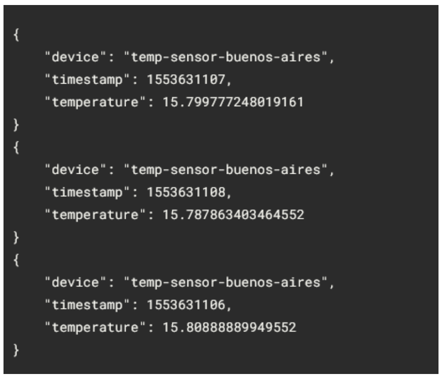
lab-bucket-2

Location	Storage class	Public access	Protection
us (multiple regions in United States)	Standard	Not public	None

Buckets > lab-bucket-2 > Sensor-Data > (note the slash at the end of the input text)

Name	Size	Type	Created	Storage class	Last modified	Public access
output-2022-04-13T15:10:00.000...	433 B	text/plain	Apr 13, 20...	Standard	Apr 13, 20...	Not public

Open the file by clicking on its name. Your file contents should be similar to what is shown below. You can click on **Download** to see the contents.

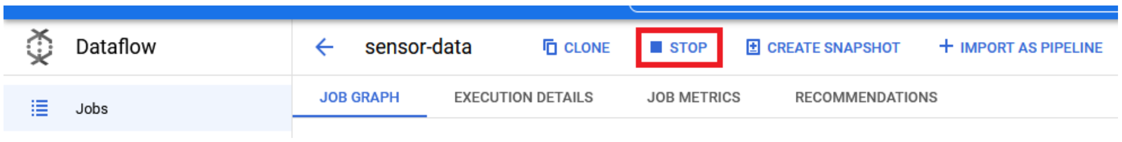


```
{
    "device": "temp-sensor-buenos-aires",
    "timestamp": 1553631107,
    "temperature": 15.799777248019161
}
{
    "device": "temp-sensor-buenos-aires",
    "timestamp": 1553631108,
    "temperature": 15.787863403464552
}
{
    "device": "temp-sensor-buenos-aires",
    "timestamp": 1553631106,
    "temperature": 15.80888889949552
}
```

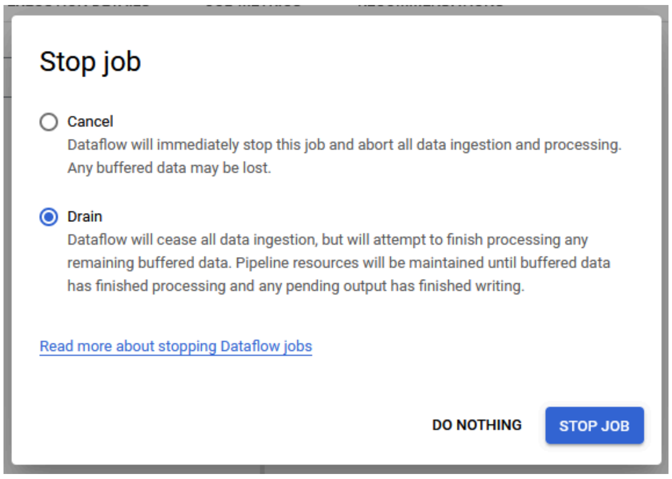
Stop the dataflow job

In this section you will learn how to stop collecting data through Dataflow by stopping the provisioned job.

1. From the Navigation menu, click **Dataflow** and select the dataflow you created earlier.
2. Then click the **Stop** button:



3. In the dialog box, select **Drain**, then **Stop Job**.



Stop job

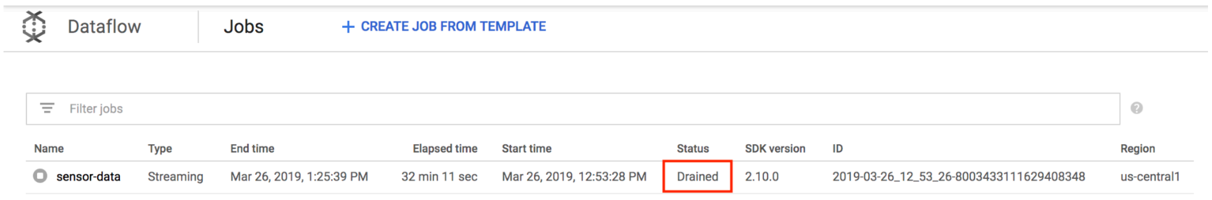
Cancel
Dataflow will immediately stop this job and abort all data ingestion and processing. Any buffered data may be lost.

Drain
Dataflow will cease all data ingestion, but will attempt to finish processing any remaining buffered data. Pipeline resources will be maintained until buffered data has finished processing and any pending output has finished writing.

Read more about stopping Dataflow jobs

It will take a few minutes for the job to stop.

4. Using the back arrow, return to the Dataflow page. Make sure the job status of the sensor-data job is *Drained*:

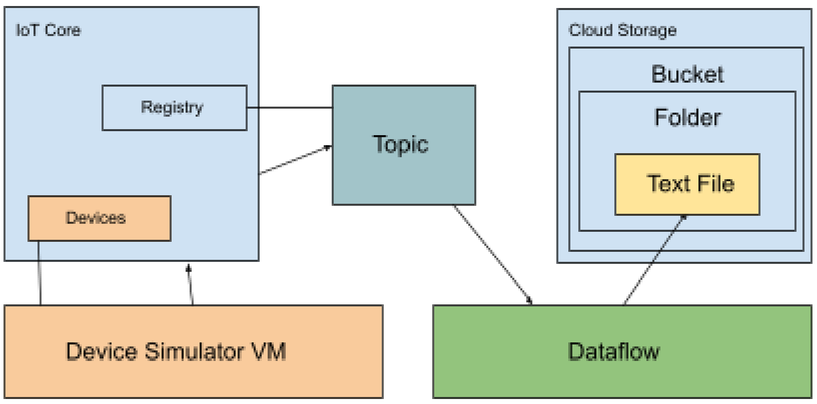


Name	Type	End time	Elapsed time	Start time	Status	SDK version	ID	Region
sensor-data	Streaming	Mar 26, 2019, 1:25:39 PM	32 min 11 sec	Mar 26, 2019, 12:53:28 PM	Drained	2.10.0	2019-03-26_12_53_26-8003433111629408348	us-central1

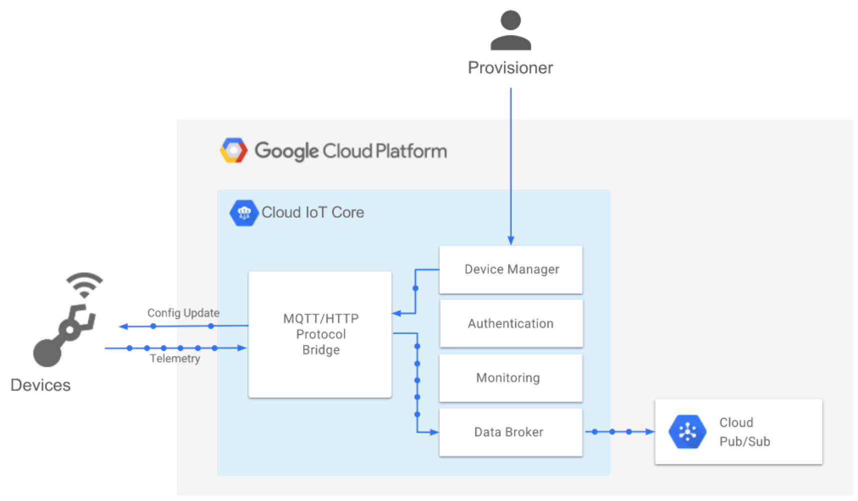
Conclusion

Congratulations, you made it to the end! Since this may be your first time using these services let's go over what you've done once more.

The system has 5 main components.



- The **Cloud Storage** stores the data, there’s not much to say about it for this tutorial.
- The **Dataflow** reads the data from the topic and puts it in the Cloud Storage, it basically acts as a subscriber.
- The **Pub/Sub Topic** is a MQTT topic which the Dataflow subscribes to and the IoT Core uses for publishing.
- The **IoT Core** is a service for connecting and managing iot devices, in this tutorial it’s used to allow the simulated devices to publish to the topic. It’s slightly more complicated to understand than the rest of the services used here but you don’t have to learn everything about it all at once. Here’s a brief explanation of what it does in the simple system you created: any device that wants to publish to a Pub/Sub Topic doesn’t directly do so, the IoT Core provides a bridge on which the devices can publish using a device-specific topic. This allows the IoT Core to manage things like authentication and it does this by using **Devices** and **Registries**. Devices define real life (or simulated) devices as IoT Cloud resources and let you set a few properties like the public key used for authentication. Registries are containers of devices with shared properties and allow, for instance, to specify one or more Pub/Sub Topics to which devices can publish.



- A **Virtual Machine** simulates two devices with sensors.