Serverless iot data processing

Steps:

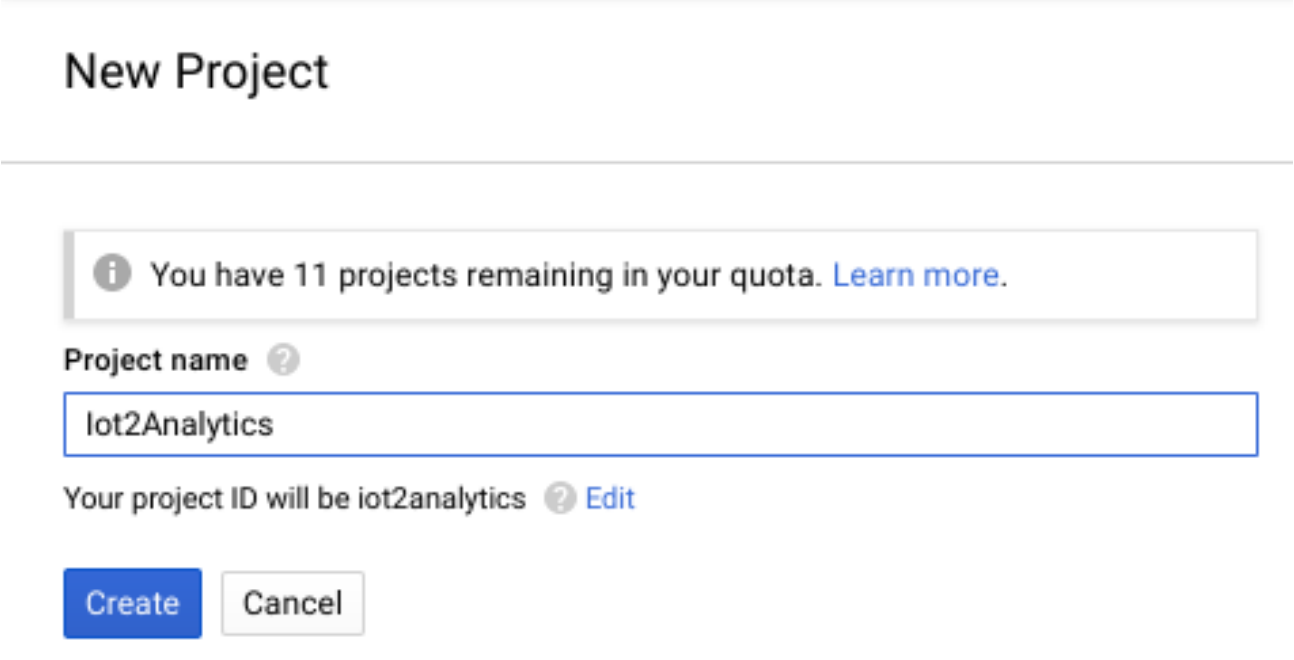
- Getting setup
- Create a bigquery table
- Create a pub/sub topic
- Create a Cloud function
- Setup the iot hardware

Getting setup

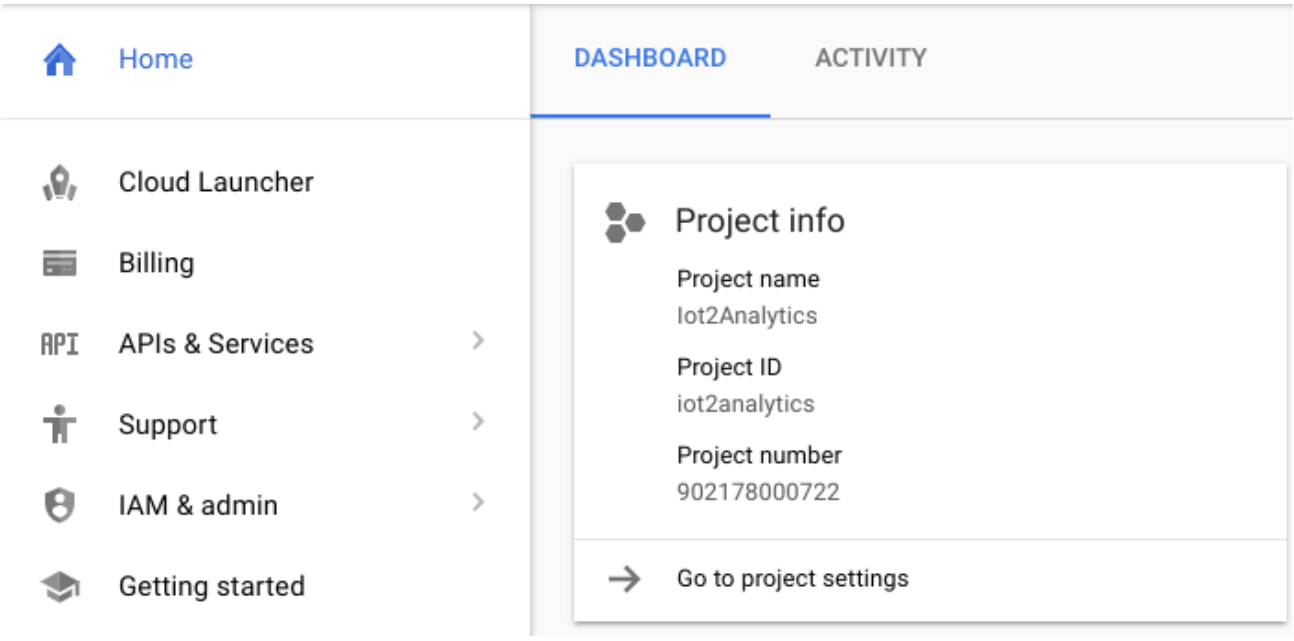
Self-paced environment setup

If we don't already have a Google account (Gmail or G Suite), we must create one. Regardless of whether we already have a Google account or not, make sure to take advantage of the $300 free trial!

Sign-in to Google Cloud Platform console (console.cloud.google.com). we can use the default project ("My First Project") for this lab or we can choose to create a new project. If we'd like to create a new project, we can use the the Manage resources page. The project ID needs to be a unique name across all Google Cloud projects (the one shown below has already been taken and won't work for we). Take note of wer project ID (i.e. wer project ID will be ____) as it will be needed later.



(If we forget wer project ID, it is located in the Cloud Console in the Home area.)

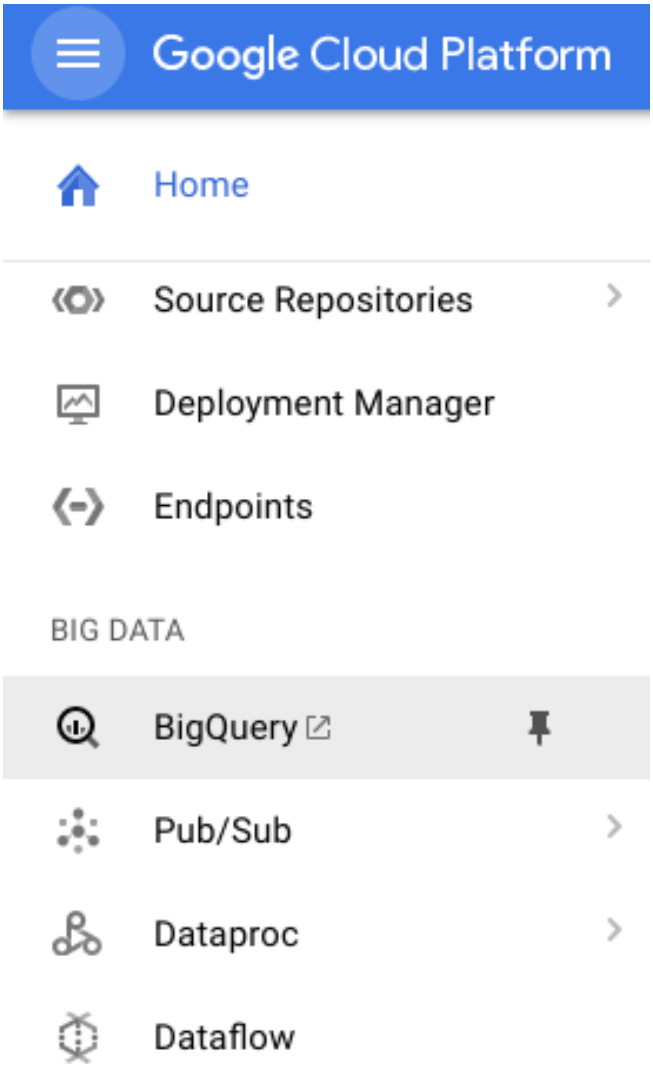


Running through this codelab shouldn't cost more than a few dollars, but it could be more if we decide to use more resources or if we leave them running. Make sure to go through the Cleanup section at the end of the codelab.

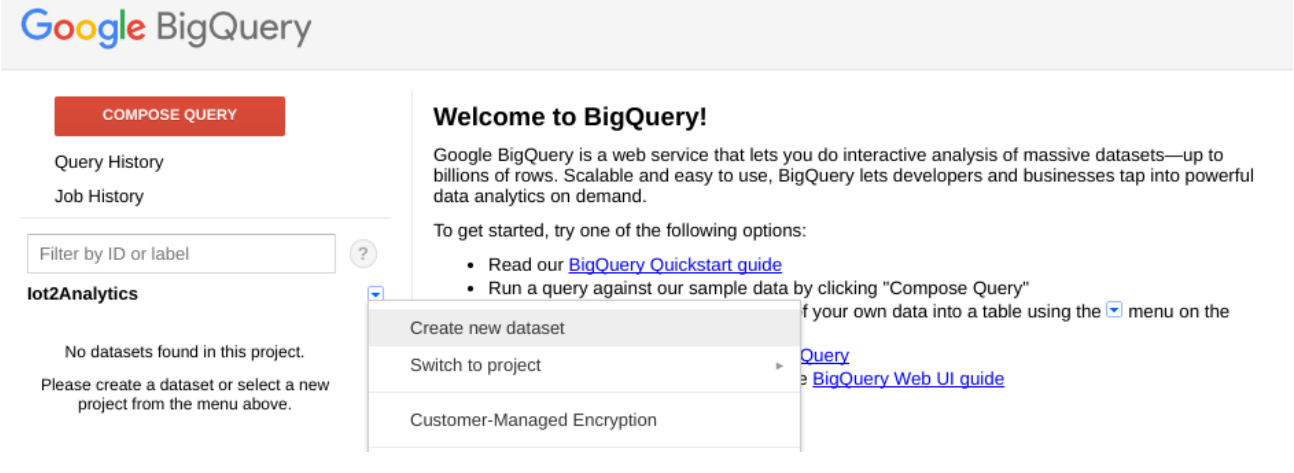
Create a bigquery table

BigQuery is a serverless, highly scalable, low cost enterprise data warehouse and will be an ideal option to store data being streamed from IoT devices while also allowing an analytics dashboard to query the information.

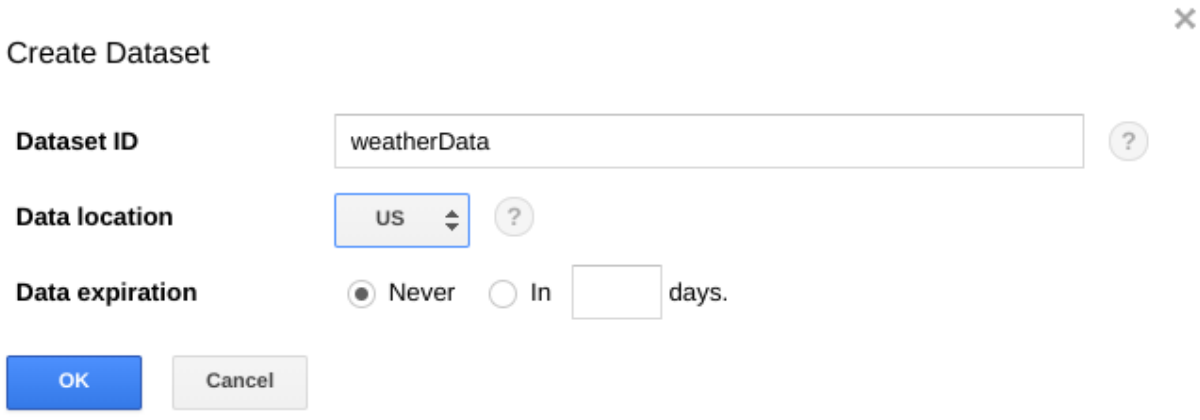
Let's create a table that will hold all of the IoT weather data. Select BigQuery from the Cloud console. **This will open BigQuery in a new window** (don't close the original window as we'll need to access it again).



Click on the down arrow icon next to wer project name and then select "Create new dataset"



Enter "weatherData" for the Dataset, select a location where it will be stored and Click "OK"

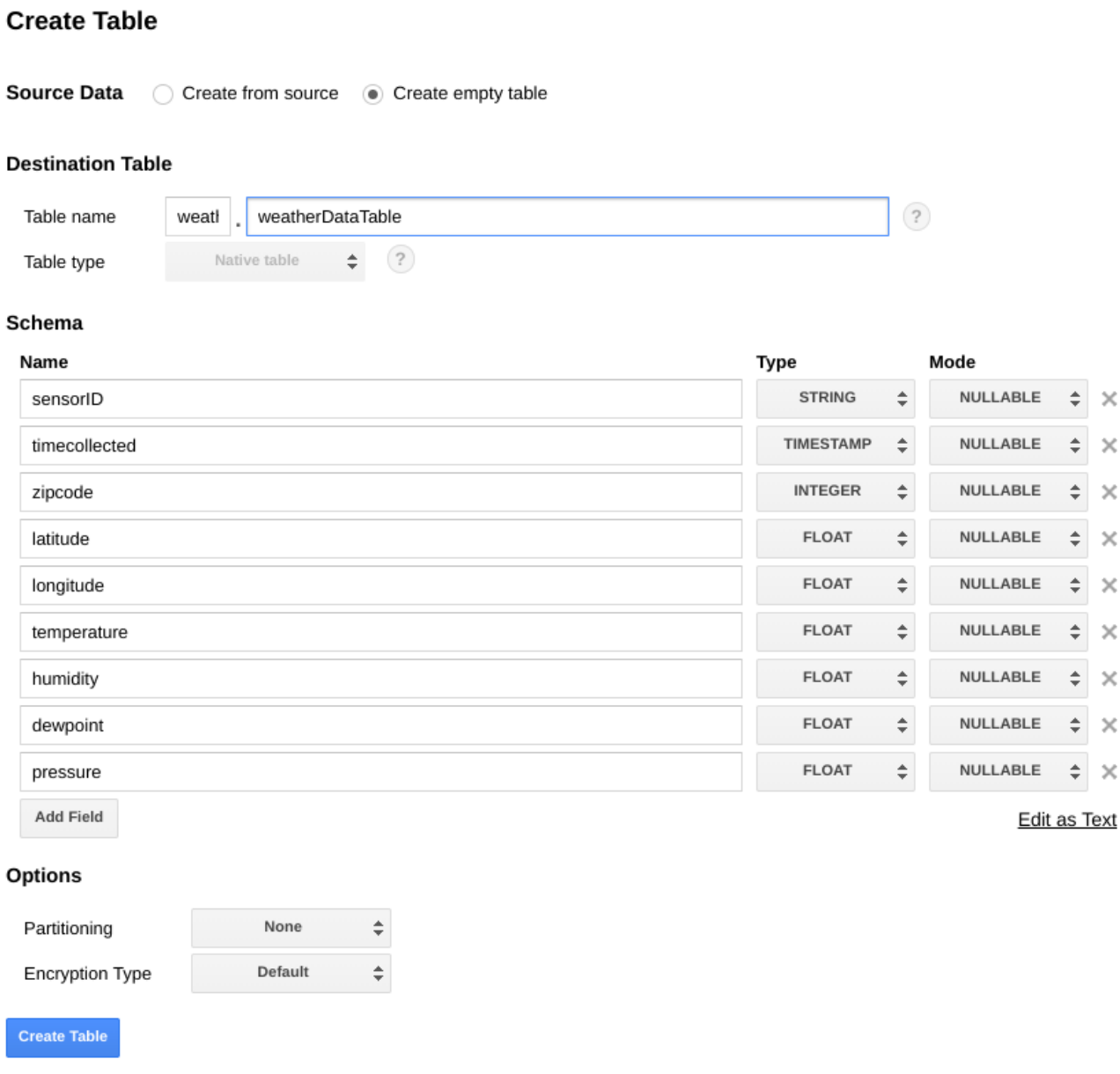


Click the "+" sign next to wer Dataset to create a new table

For **Source Data**, select **Create empty table**. For Destination table name, enter weatherDataTable. Under **Schema**, click the **Add Field** button until there are a total of 9 fields. Fill in the fields as shown below

making sure to also select the appropriate **Type** for each field. When everything is complete, click on the **Create Table** button.

(Note:Any spelling errors for the field names or the table name can cause issues when Cloud Functions attempts to add data later in this codelab.)



We get output like this

Table Details: weatherDataTable

Schema | Details | Preview

sensorID	STRING	NULLABLE	Describe this field...
timecollected	TIMESTAMP	NULLABLE	Describe this field...
zipcode	INTEGER	NULLABLE	Describe this field...
latitude	FLOAT	NULLABLE	Describe this field...
longitude	FLOAT	NULLABLE	Describe this field...
temperature	FLOAT	NULLABLE	Describe this field...
humidity	FLOAT	NULLABLE	Describe this field...
dewpoint	FLOAT	NULLABLE	Describe this field...
pressure	FLOAT	NULLABLE	Describe this field...

Add New Fields

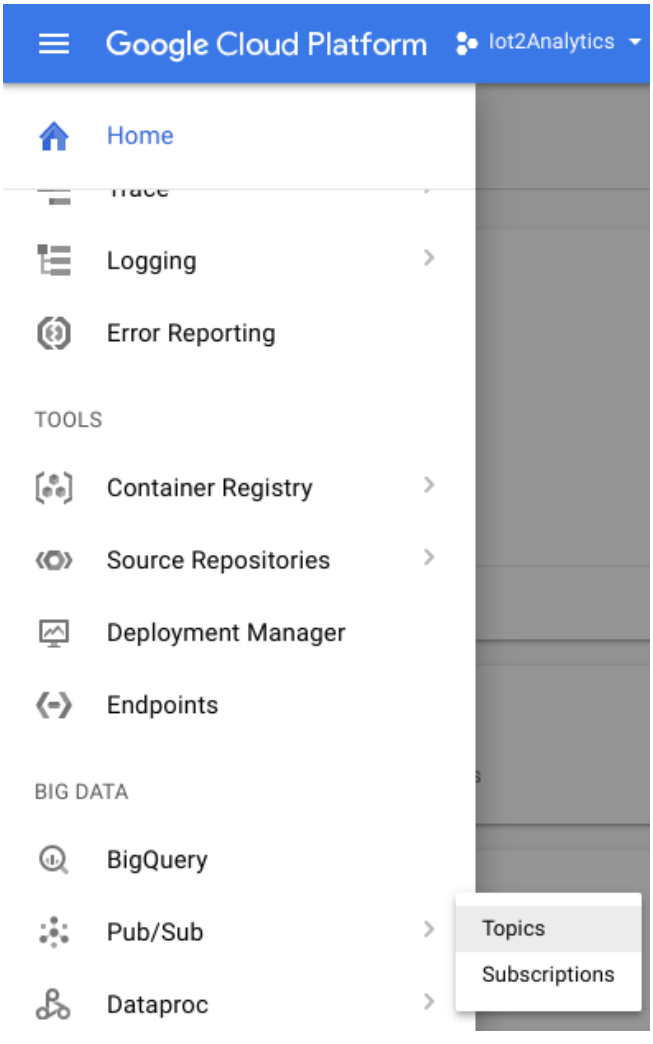
now we have a data warehouse setup to receive our weather data.

Create a Pub/Sub topic

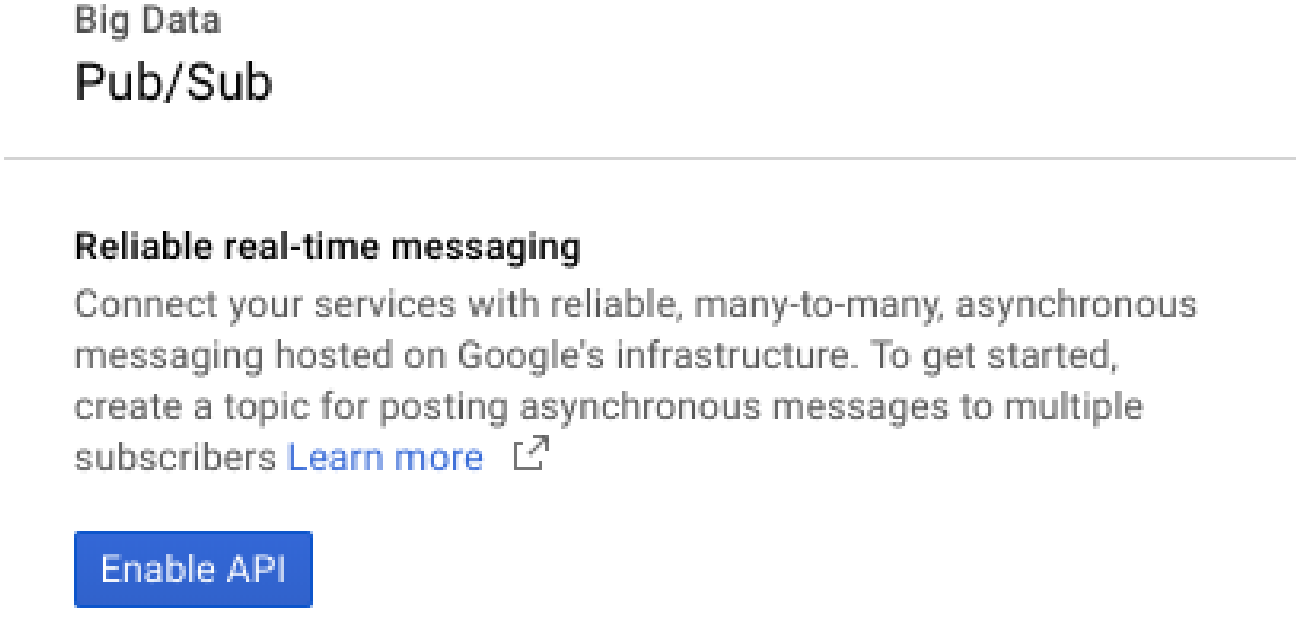
Cloud Pub/Sub is a simple, reliable, scalable foundation for stream analytics and event-driven computing systems. As a result, it is perfect for handling incoming IoT messages and then allowing downstream systems to process them.

If you are still in the window for BigQuery, switch back to the Cloud Console. If you closed the Cloud Console, go to https://console.cloud.google.com

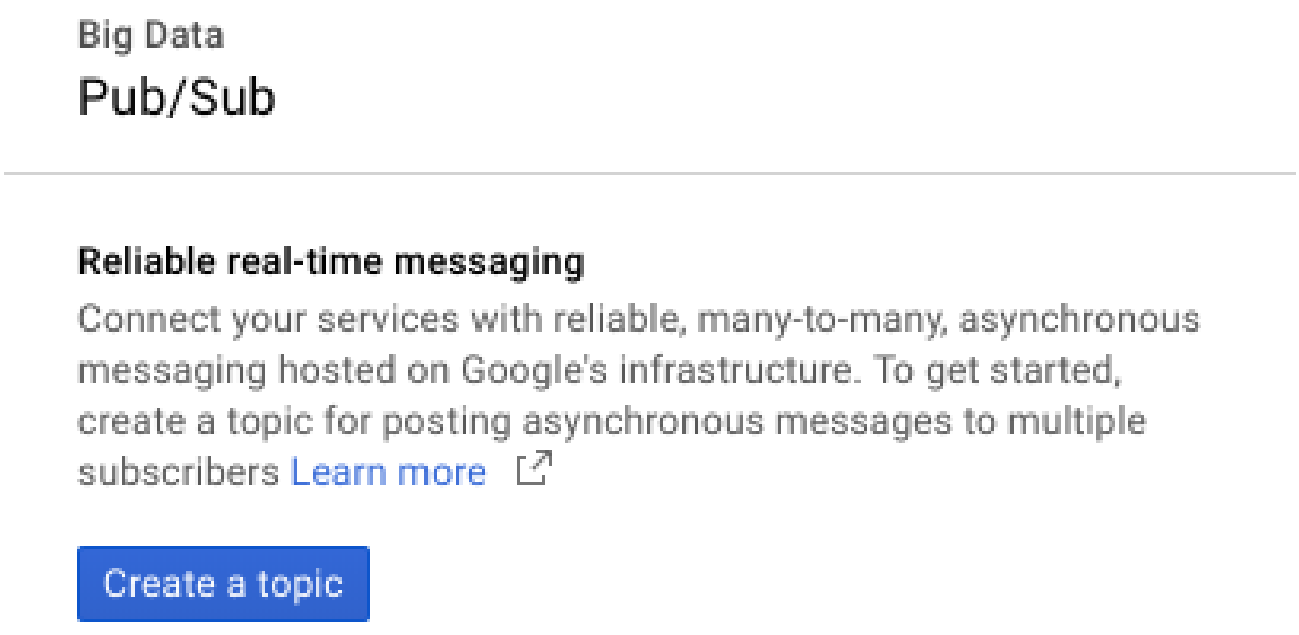
From the Cloud Console, select Pub/Sub and then Topics.



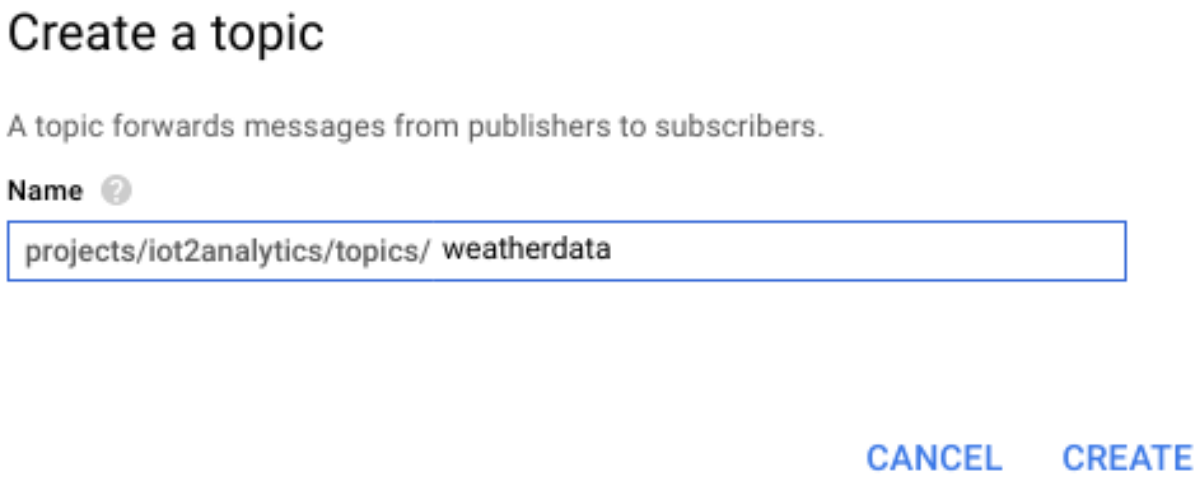
If we see an Enable API prompt, click the Enable API button.



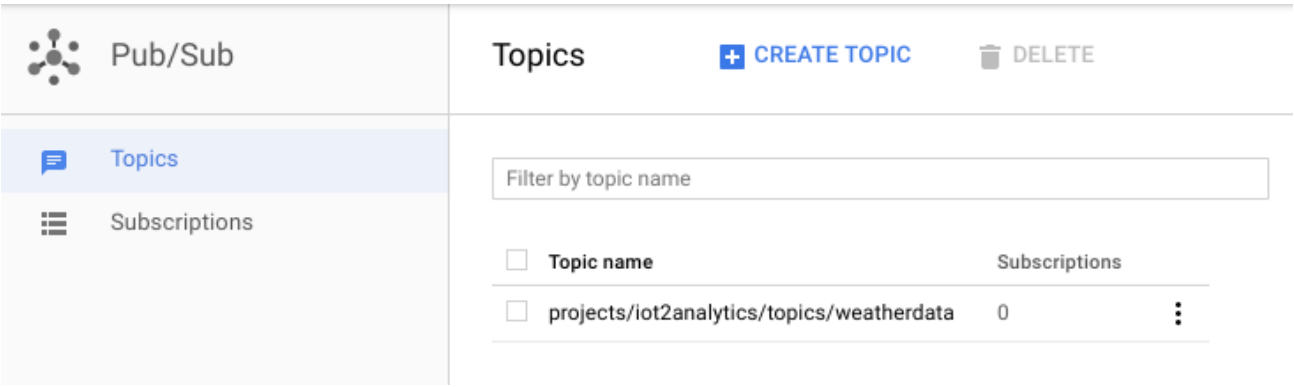
Click on the Create a topic button



Enter "weatherdata" as the topic name and the click Create



we should see the newly created topic

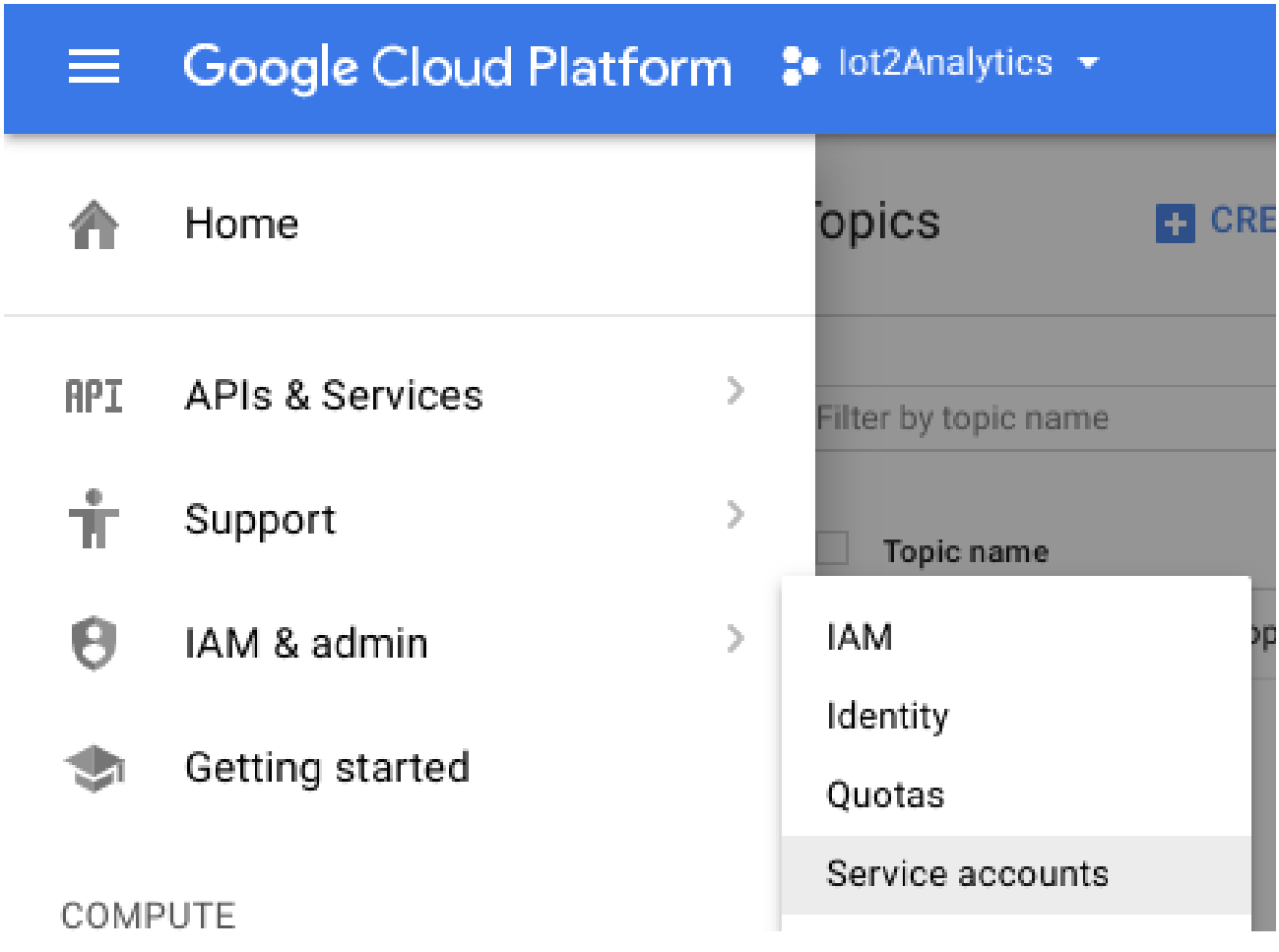


now we have a Pub/Sub topic to both publish IoT messages to and to allow other processes to access those messages.

Secure publishing to the topic

If plan to publish messages to the Pub/Sub topic from resources outside of your Google Cloud Console (e.g. an IoT sensor), it will be necessary to more tightly control access using a service account and to ensure the security of the connection by creating a trust certificate.

From the Cloud Console, select IAM & Admin and then Service accounts



Click on the Create service account button

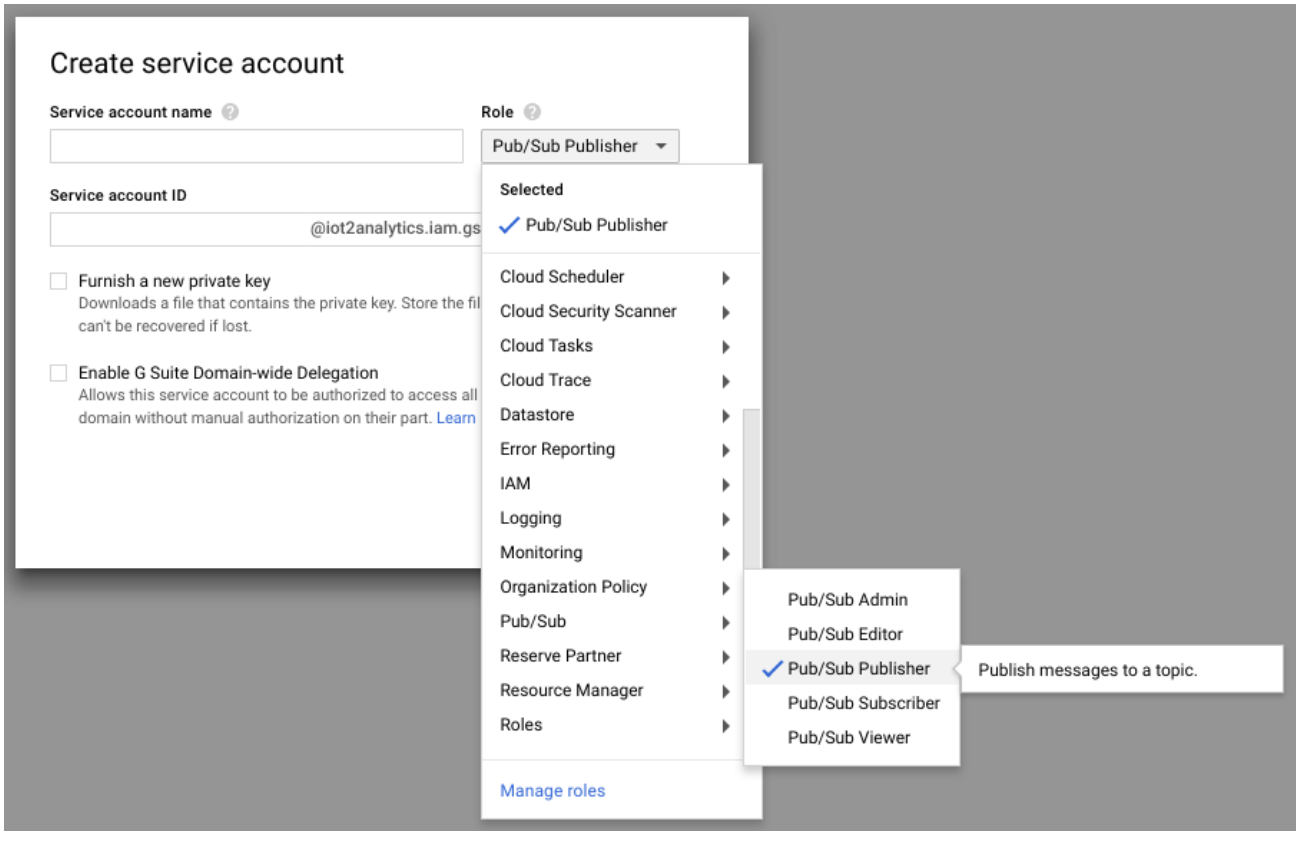
Permissions

Service account management

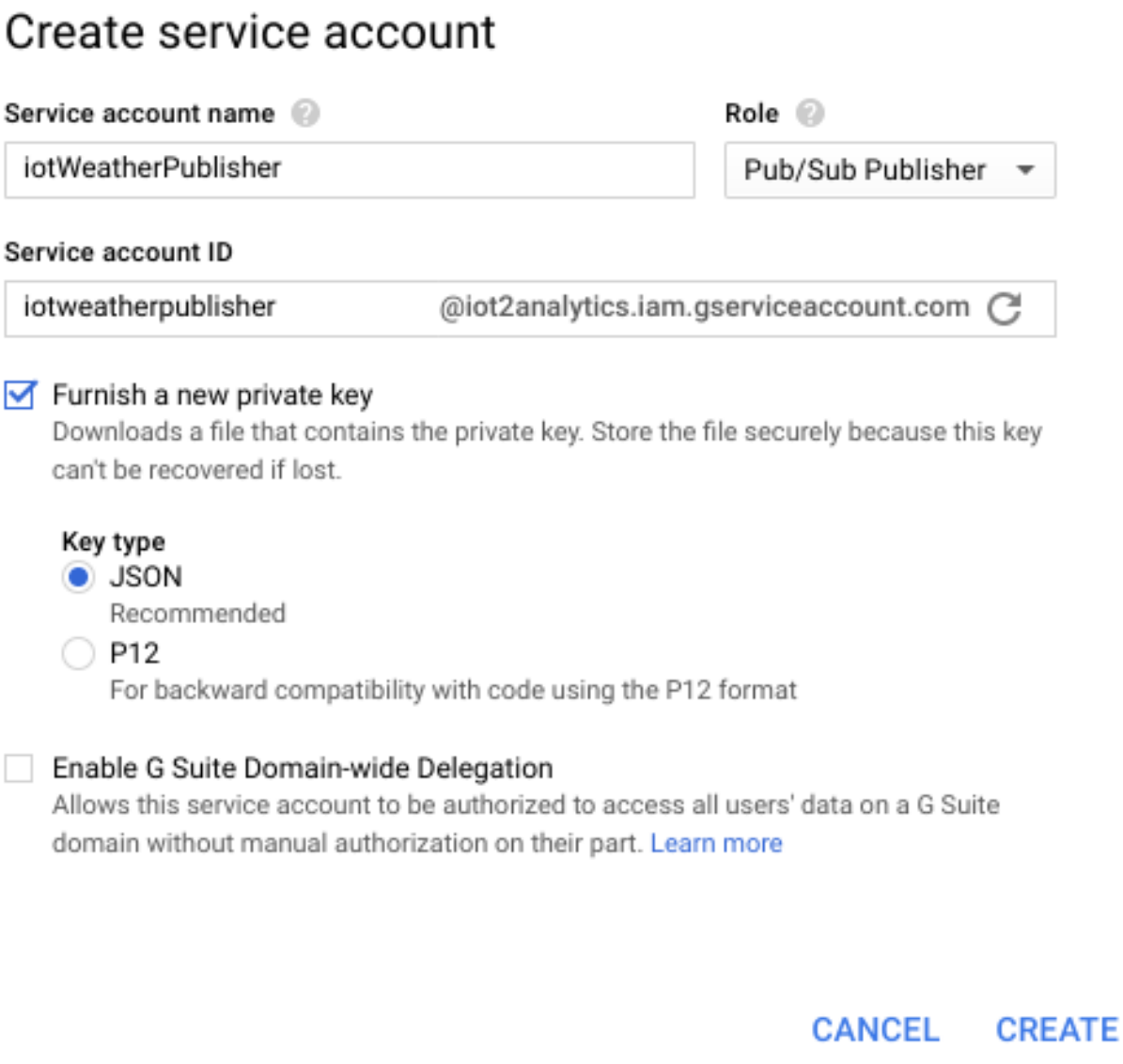
A service account represents a Google Cloud service identity, such as code running on Compute Engine VMs, App Engine apps, or systems running outside Google. Learn more

Create service account

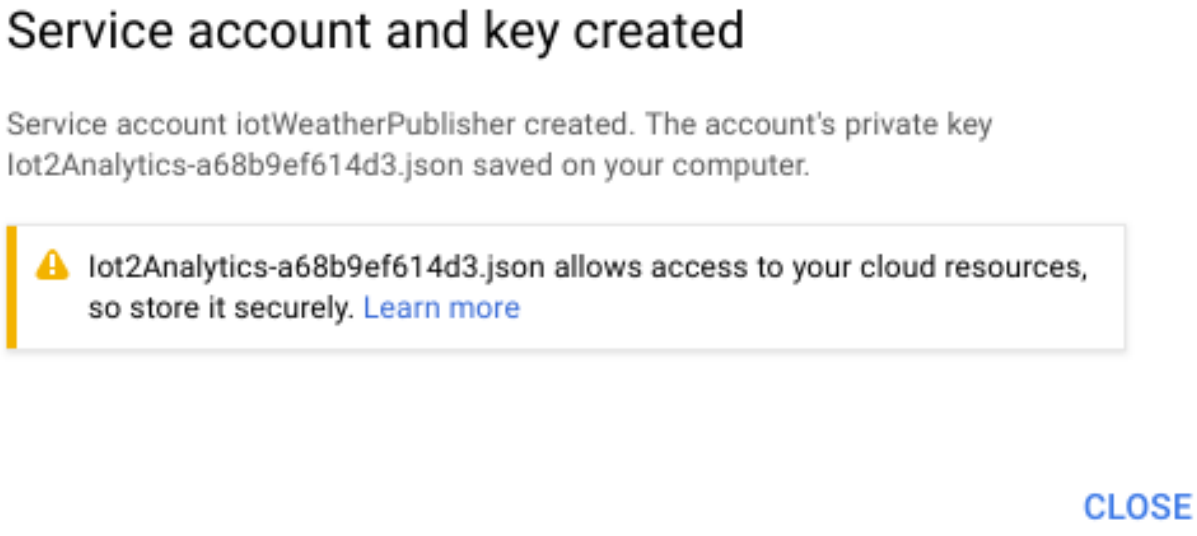
In the Role dropdown, select the Pub/Sub Publisher role



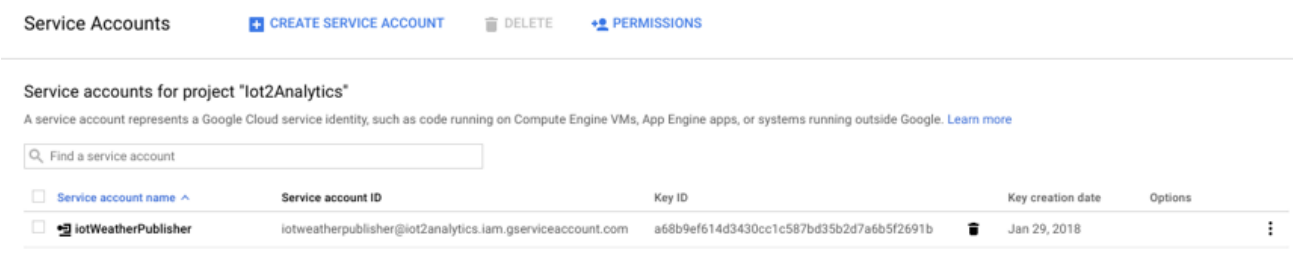
Type in a service account name (iotWeatherPublisher), check the Furnish a new private key checkbox, make sure that Key type is set to JSON and click on "Create"



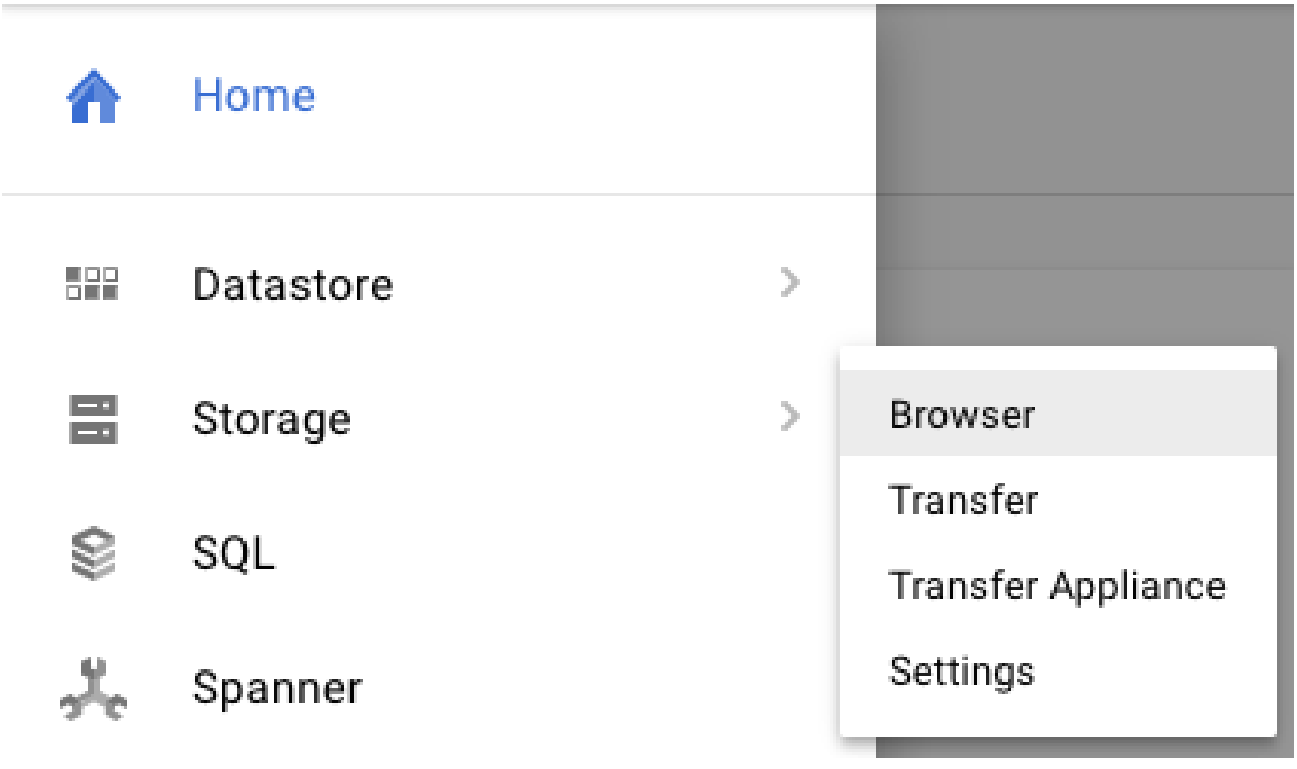
The security key will download automatically. There is only one key, so it is important to not lose it. Click Close.



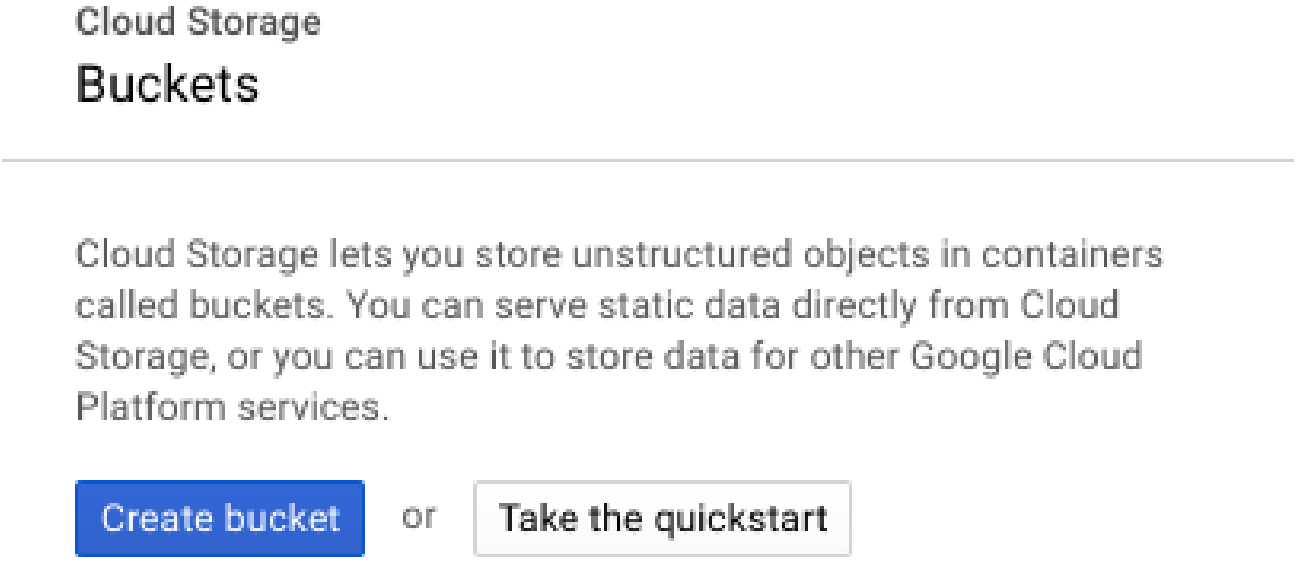
We should see that a service account has been created and that there is a Key ID associated with it.



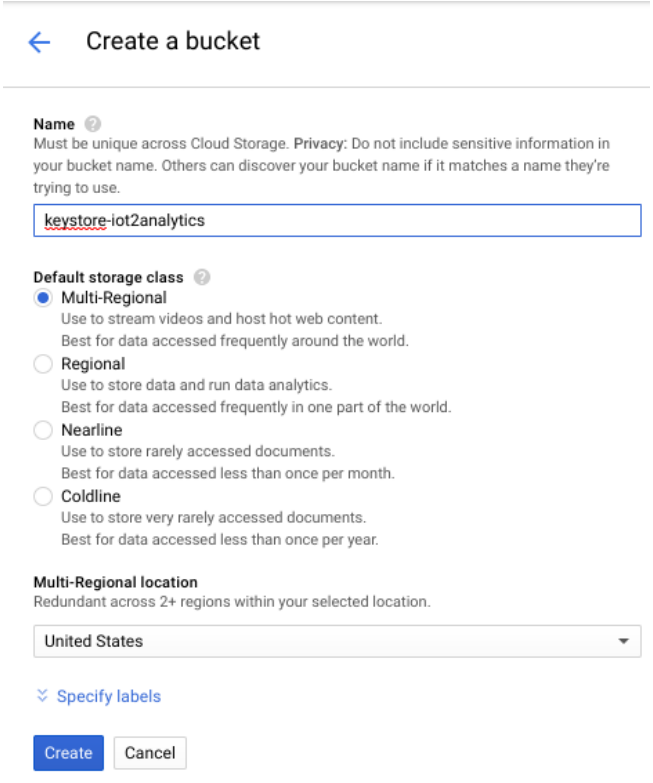
In order to easily access the key later, we will store it in Google Cloud Storage. From the Cloud Console, select Storage and then Browser.



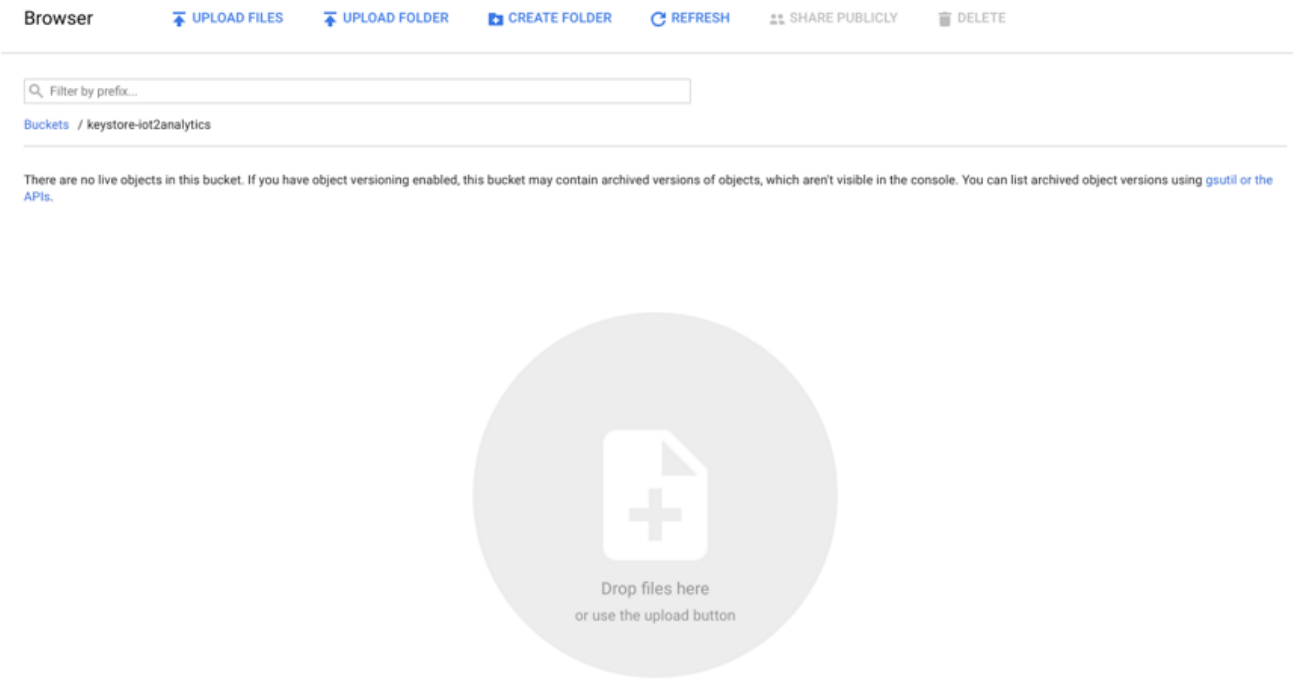
Click the Create Bucket button



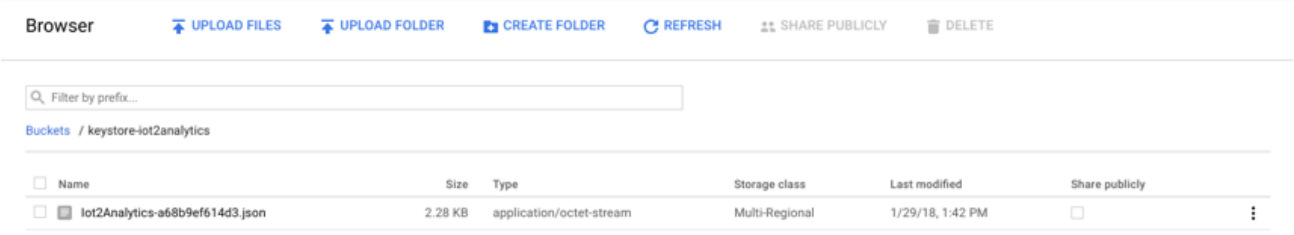
Choose a name for the storage bucket (it must be a name that is globally unique across all of Google Cloud) and click on the Create button

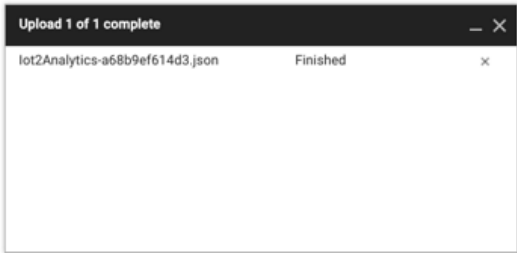


Locate the security key that was automatically downloaded and either drag/drop or upload it into the storage bucket



After the key upload is complete, it should appear in the Cloud Storage browser



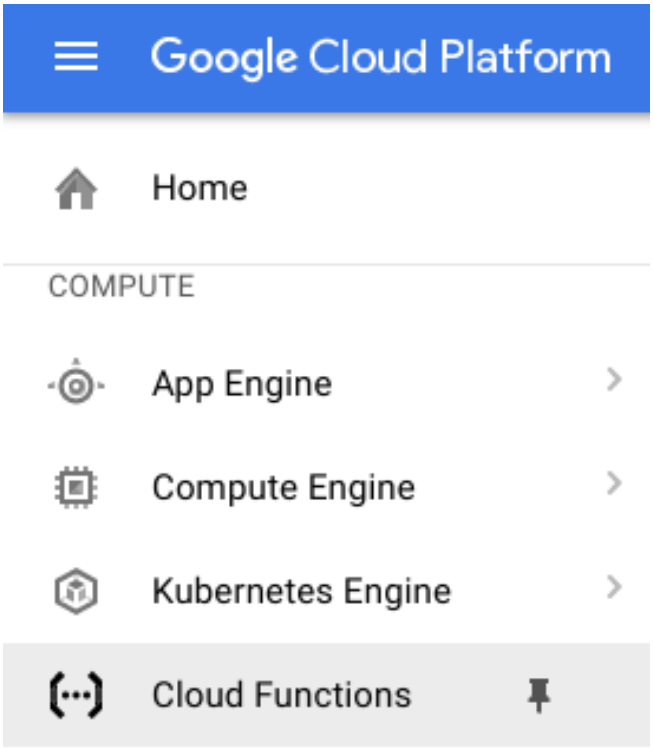


Make note of the storage bucket name and the security key file name for later.

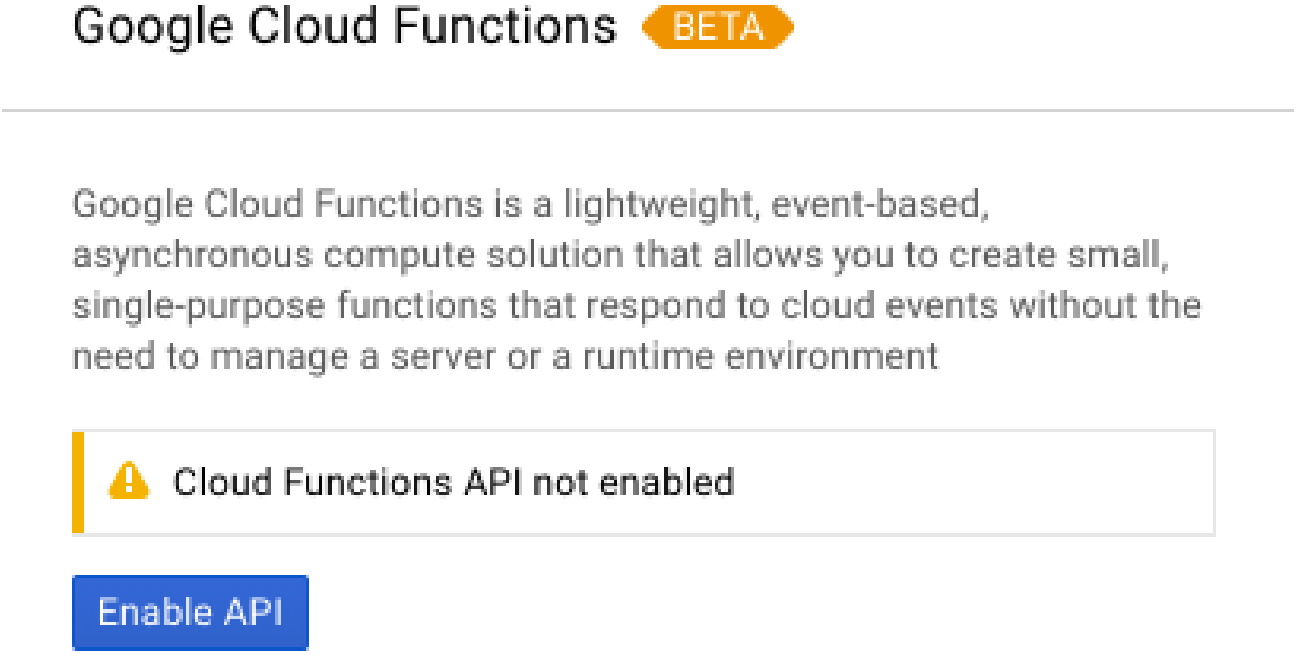
Create a Cloud function

Cloud computing has made possible fully serverless models of computing where logic can be spun up on-demand in response to events originating from anywhere. For this lab, a Cloud Function will start each time a message is published to the weather topic, will read the message and then store it in BigQuery.

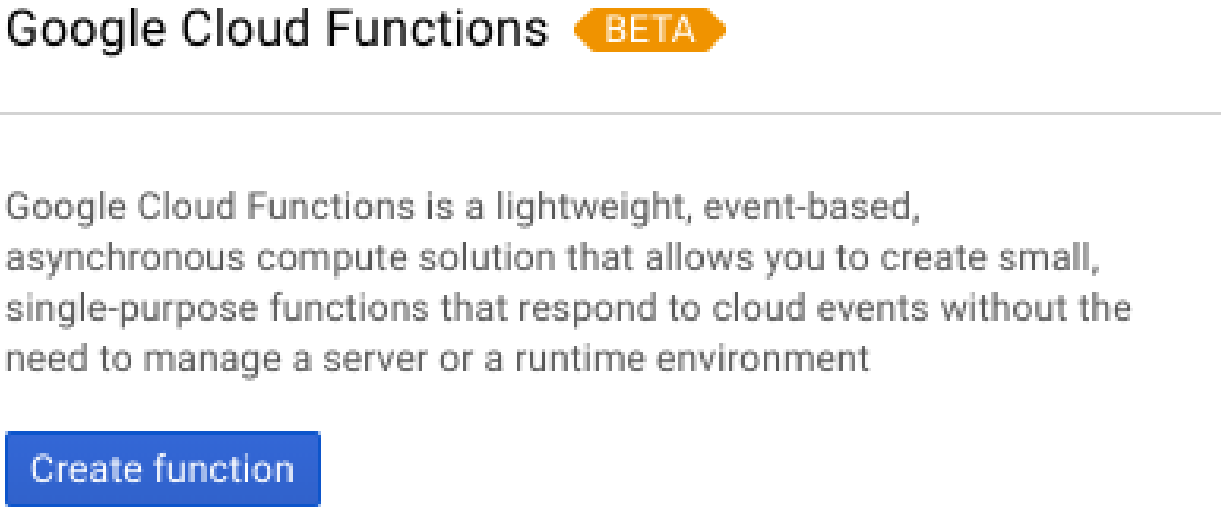
From the Cloud Console, select Cloud Functions



If you see an API message, click on the Enable API button



Click on the Create function button



In the Name field, type function-weatherPubSubToBQ. For Trigger select Cloud Pub/Sub topic and in the Topic dropdown select weatherdata. For source code, select inline editor. In the index.js tab, paste the following code over what is there to start with. **Make sure to change the constants for projectId, datasetId and tableId to fit your environment.**

Code

```
/**
 * Background Cloud Function to be triggered by PubSub.
 *
 * @param {object} event The Cloud Functions event.
 * @param {function} callback The callback function.
 */
exports.subscribe = function (event, callback) {
  const BigQuery = require('@google-cloud/bigquery');
  const projectId = "myProject"; //Enter your project ID here
  const datasetId = "myDataset"; //Enter your BigQuery dataset name here
  const tableId = "myTable"; //Enter your BigQuery table name here -- make sure it is setup
correctly
  const PubSubMessage = event.data;
  // Incoming data is in JSON format
  const incomingData = PubSubMessage.data ? Buffer.from(PubSubMessage.data,
'base64').toString() : "{'sensorID':'na','timecollected':'1/1/1970
00:00:00','zipcode':'00000','latitude':'0.0','longitude':'0.0','temperature':'-273','humidity':'-1','de
wpoint':'-273','pressure':'0'}";
  const jsonData = JSON.parse(incomingData);
  var rows = [jsonData];

  console.log(`Uploading data: ${JSON.stringify(rows)}`);
```

```
  // Instantiates a client
  const bigquery = BigQuery({
    projectId: projectId
  });

  // Inserts data into a table
  bigquery
    .dataset(datasetId)
    .table(tableId)
    .insert(rows)
    .then((foundErrors) => {
      rows.forEach((row) => console.log('Inserted: ', row));

      if (foundErrors && foundErrors.insertErrors != undefined) {
        foundErrors.forEach((err) => {
            console.log('Error: ', err);
        })
      }
    })
    .catch((err) => {
      console.error('ERROR:', err);
    });
  // [END bigquery_insert_stream]


  callback();
};
```

In the package.json tab, paste the following code over the placeholder code that is there

```
{
  "name": "function-weatherPubSubToBQ",
  "version": "0.0.1",
  "private": true,
  "license": "Apache-2.0",
  "author": "Google Inc.",
  "dependencies": {
    "@google-cloud/bigquery": "^0.9.6"
  }
}
```

If the Function to execute is set to "HelloWorld", change it to "subscribe". Click the Create button

Cloud Functions | Create function

Name

function-weatherPubSubToBQ

Memory allocated

256 MB

Trigger

- HTTP trigger
- (•) Cloud Pub/Sub topic
- Cloud Storage bucket

Topic

weatherdata

Source code

- (•) Inline editor
- ZIP upload
- ZIP from Cloud Storage
- Cloud Source repository

index.js package.json

```
{
  "name": "function-weatherPubSubToBQ",
  "version": "0.0.1",
  "private": true,
  "license": "Apache-2.0",
  "author": "Google Inc.",
  "dependencies": {
    "@google-cloud/bigquery": "^0.9.6"
  }
}
```

Function to execute

subscribe

More

Function deployment might take a few minutes. To build and test the function locally, use the local emulator

Create Cancel

It will take about 2 minutes until your function will show that it has deployed

Cloud Functions | Overview CREATE FUNCTION REFRESH DELETE COPY

Filter functions Columns

Name	Region	Trigger	Memory allocated	Executed function	Last deployed
function-weatherPubSubToBQ	us-central1	topic: weatherdata	256 MB	subscribe	1/29/18, 3:34 PM

Congratulations! now we just connected Pub/Sub to BigQuery via Functions.