**Cloud IoT**

**and**

**Influxdata**

**Platform**

**Integration**

## 

## 

## 

## Introduction to IoT

IoT is the network of physical devices, vehicles, cameras, alarms, home appliances and other items embedded with electronics, software, sensors, actuators, and network connectivity which enables these objects to connect and exchange data through the Internet.

## IoT Solution with Influxdata Platform

In this guide, we are giving demonstrations on how to integrate the Google Cloud IoT core with the influxdata platform. First of all, we take a [Bosch BME 280](https://www.adafruit.com/product/2652) sensor board and connect it to a Raspberry PI through a breadboard. Next, we connected the keyboard, mouse, monitor, network cable and power cable to it.

When hardware wirings are finished, we take on the tasks of installing the Operating System for the Raspberry PI. To accomplish this, we use a another computer to download the [Raspberry Pi Desktop OS image and flash it](https://www.raspberrypi.org/downloads/raspberry-pi-desktop/) an SD card. We inserted this SD card to a Raspberry PI and recycle the board’s power. After the board reboots, we can complete the installation of the Debian OS through on screen instructions.

After the installation is completed, we log into the Raspberry and use the web browser to download the GO software application. This application is required to build and run Google IoT client, which was written in GO language. This IoT client reads data from the sensor and then send them to Google Cloud IoT Core.

On a server or Google VM instance, we can install telegraf, influxdata, and chronograf applications to run the data collection, data storing, and data presentation mechanisms. We also need to install GO so that we can compile telegraf since it was written in GO language. Here is what each application does in this demonstration. Telegraf collects the data and dump it to influxdata. Influxdata store the data and allow you to query the data. Chronograf is been used to get data from influxdata and then present them on the charts in real time.

When data messages arrived at the GCP, it will be stored in the Pub/Sub queues. Therefore, we have to either create a subscription using “push” delivery type or create cloud function to push data from the Pub/Sub queues to a remote server or Google VM instance. In this guide, we demonstrate both methods. We also set up the secure SSL authentication and connection between the GCP and the influxdata platform by using the CA certificate, private key and signed certificate in the telegraf configuration file.

Finally, we use the chronograf to look at data presented on charts in real time. We can also choose how often the data is been collected.

## Required Components

* Google Cloud Platform
* Raspberry PI Main Board (Pi 3, 2, or Zero)
* [Bosch BME 280](https://www.adafruit.com/product/2652) sensor board
* Telegraf Software
* Influxdb Software
* Chronograf Software
* Monitor
* Keyboard
* Mouse

IoT Core Setup on the GCP Side

Google Cloud Platform Account

Before IoT core setup can be done, an account with Google is needed to log into the Google Cloud Platform. Please register an account with Google if you don’t have it. Once an account is created successfully, you will be able to run the gcloud commands below.

Install the ​Google Cloud Command Line Tool

Follow all the instruction from this [link](https://cloud.google.com/sdk/downloads) to install google command line tool. After the google command line tool is installed, run the commands below.

Installing the components

gcloud components install beta

Authenticate with Google Cloud

gcloud auth login

Create cloud project - choose your unique project name

gcloud projects create ​ YOUR\_PROJECT\_NAME

Set default values for gcloud

gcloud config set project ​ YOUR\_PROJECT\_NAME

Create PubSub topic for device data

gcloud beta pubsub topics create ​ <iot-topic>

Create PubSub subscription for device data

gcloud beta pubsub subscriptions create --topic ​ <iot-topic>​ ​ <iot-subscription>

Generate RSA Public and Private keys

Log into a box that can run shell scripts and use the commands below to create the private and self-signed public key and put the the keys in the folder “PemFiles”.

mkdir PemFiles

cd PemFiles

openssl req -x509 -newkey rsa:2048 -keyout private\_key.pem -nodes -out public\_key.pem -subj "/CN=unused"

Create device registry

gcloud beta iot registries create ​<iot-registry>​ --region us-central1

--event-pubsub-topic=​<iot-topic>

Create device

gcloud iot devices create <DEVICE\_ID>\

--project=<PROJECT\_ID> \

--region=<REGION> \

--registry=<REGISTRY\_ID>\

--public-key path=public\_key\_file,type=rs256

Note: replace public\_key\_file with the path to the public key file generated in the section “Generate RSA Public and Private keys” above.

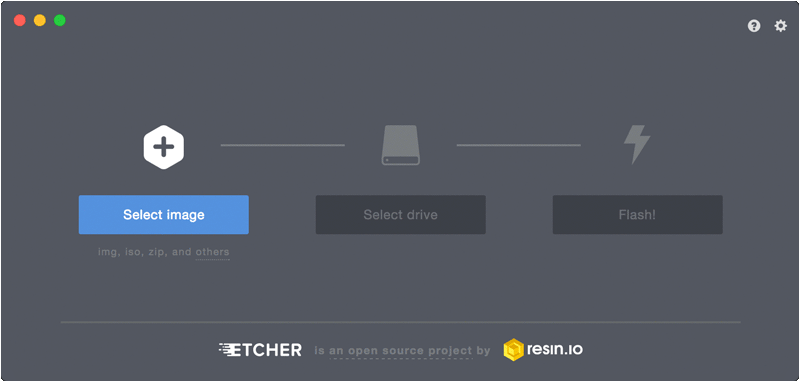
## Install the Operating System on the Device

Download the OS Image for Raspberry PI

The OS image from Raspberry website can be downloaded from this [link](https://www.raspberrypi.org/downloads/raspberry-pi-desktop/). Once downloaded, use etcher from <https://etcher.io/> to flash the OS image to the SD card:

Create a bootable SD card with the OS image

1. Click Select image and find your application's OS image file.
2. If you haven't already done so, insert your SD card into your computer.
3. Etcher will automatically detect it. If you have more than one SD card inserted, you will need to select the appropriate one.
4. Click the Flash! button.



Etcher will now prepare a bootable SD card and validate that it was flashed correctly. You'll get a little ping when it's done, and Etcher will safely eject the SD card for you.

Once flashed, insert the SD card and power on the Raspberry Pi board by inserting the other end of the Power/Console USB cable into your computer or laptop. When the light is on, as shown below in red circle, that means the board has powered on.



Log Into the Raspberry System

The initial boot might take up to five minutes to extract the image and run setup. Next, log into the Raspberry PI using any two of the ways below.

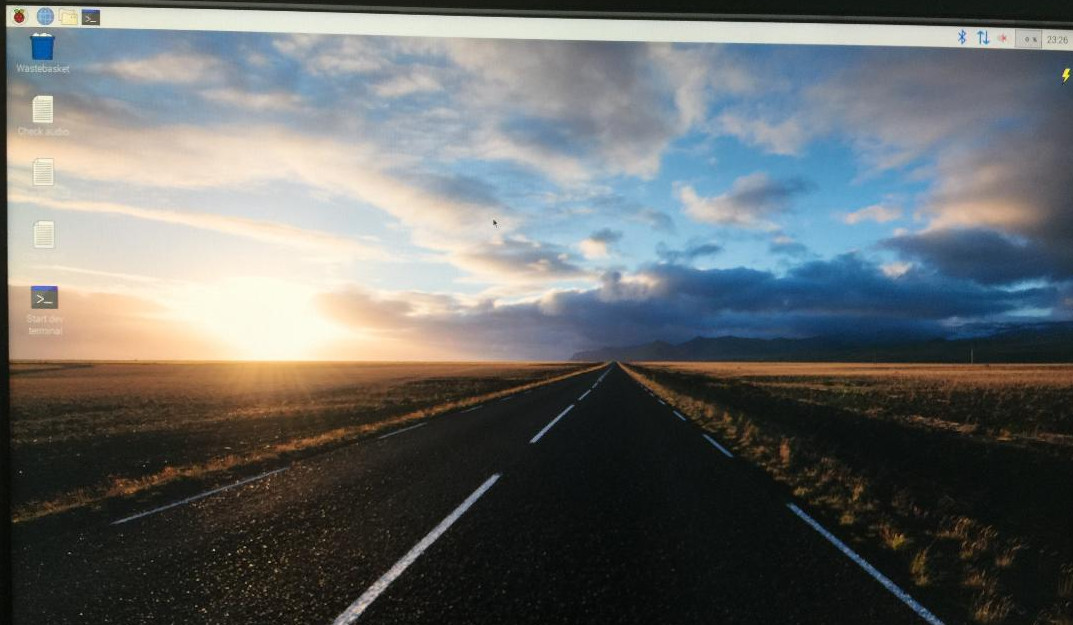
Using keyboard/screen combo

After finishing the boot up process, use keyboard/screen combo to log in with the login info below.

Username: **pi**

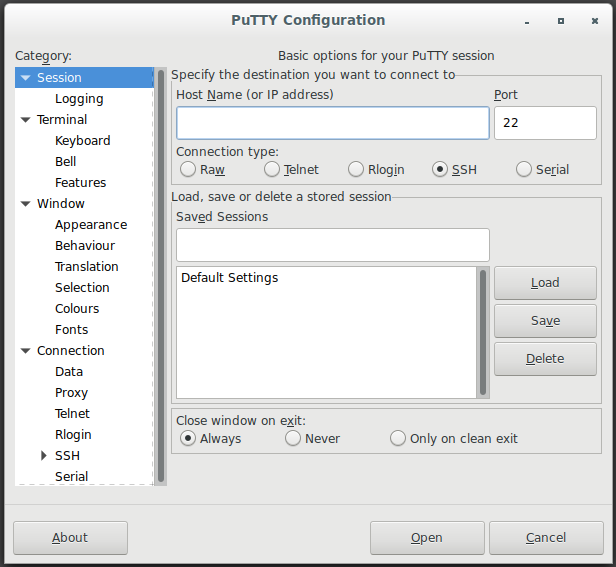
Password: **raspberry**

After successful login, you will see a desktop screen:

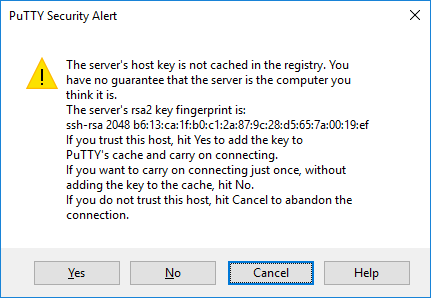


Using SSH connection

1. Connect the Raspberry Pi to your internet router with an ethernet cable, then plug in the power cord to boot it up.
2. Now we need to find out the local IP address your router has assigned to the Raspberry Pi. You can use ip scanner or directly logging into the router setup user interface.
3. Now it’s time to connect to the Pi (Download the PuTTy from <https://www.ssh.com/ssh/putty/linux/> if you don’t have one). Open up the PuTTY, and enter the local IP address into the “Host Name” field:



1. On the connection attempt, a security warning will appear. Just press “Yes” since you’re connecting to your own Raspberry Pi.



1. If the SSH connection is successful you’ll be greeted with the login prompt of your Raspberry Pi. If this is your first login and you haven’t changed the username or password yet, please enter “pi” for username and “raspberry” for password.

Username: **pi**

Password: **raspberry**

## Setting up the Google IoT Demo On the Device

1. Install Git

sudo apt-get install git

1. Download and Install go
2. Navigate to this [link](https://golang.org/dl/) and download the latest Go version. Complete the installation
3. Setup Environment Variables for Go

export GOPATH=$HOME/gorepo

export GOROOT=/usr/local/go

export PATH=$GOPATH/bin:$GOROOT/bin:$PATH

1. Install Drivers for the Sensors

Run this command:

go get -d github.com/davidgs/bme280\_go

1. Enable the I2C interface on the device

On the Raspberry PI, run this command:

sudo raspi-config

On configuration menu, select “option 5 - Interfacing Options” and the option “P5 I2C” on the next screen, and then click on “Yes” to save the settings permanently.

1. Download and Build Google IoT Client

Run these commands:

git clone <https://github.com/davidgs/GoogleIoTClient.git> [GoogleIoTClient](https://github.com/davidgs/GoogleIoTClient.git)

cd GoogleIoTClient

go build GoogleIoT.go

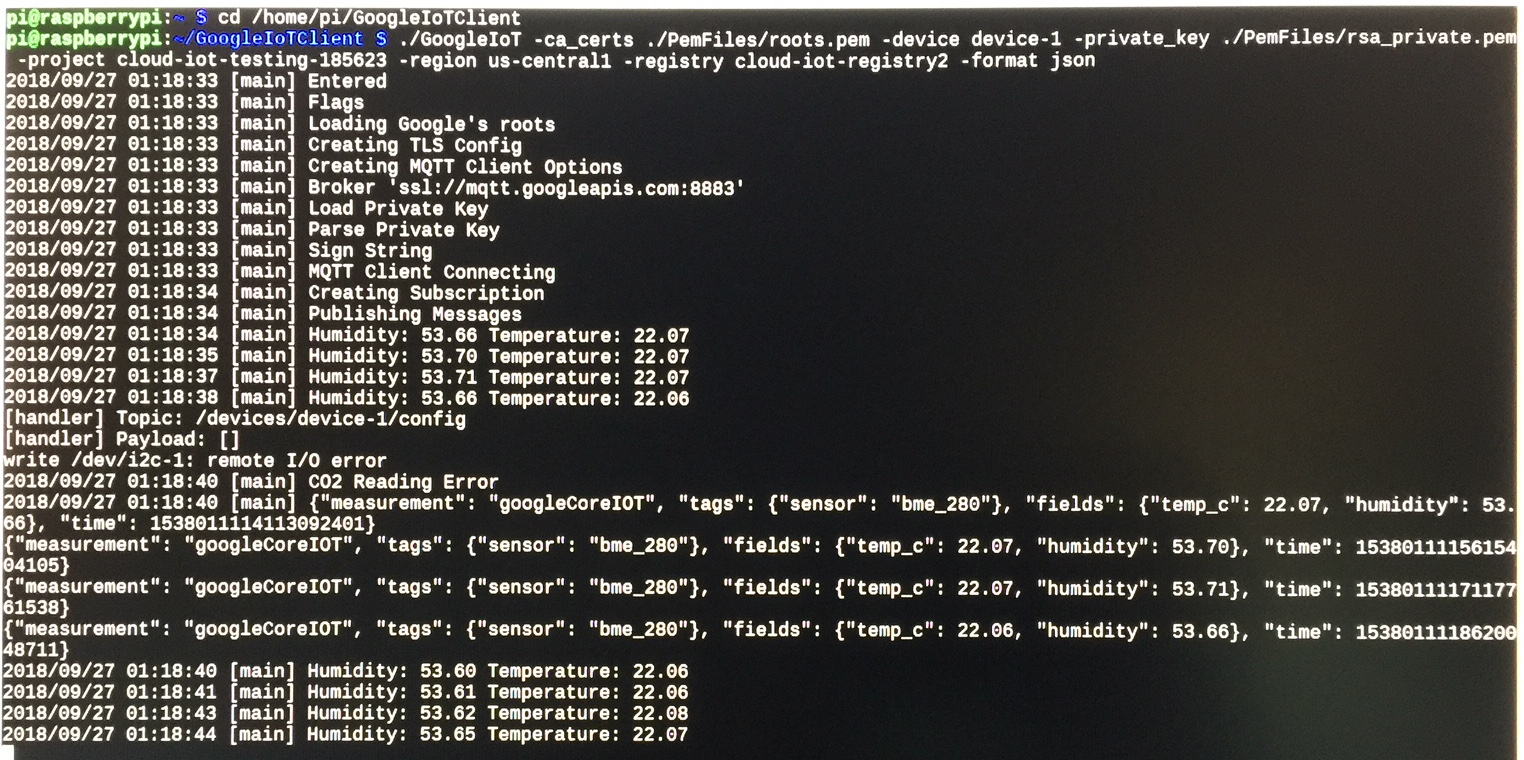
1. Publish Data to the Google Cloud IoT Core.

Replace the parameter [privateKey] with the path to the private key generated in the “Generate RSA Public and Private keys” section and other parameters like [deviceName], [registryName], [region], [projectID] with the real values in the command below.

cd GoogleIoTClient

wget https://github.com/khanhhale/UpSquaredGroveIotDevKit/blob/master/PemFiles/roots.pem

./GoogleIoT -ca\_certs roots.pem -device [deviceName] -private\_key [privateKey] -project [projectID] -region [region] -registry [registryName] -format json



A successful run will generate sensor data like the screen shown above. Otherwise the program will terminate on failure.

## Install & configure software applications on the server

1. Install Git

sudo apt-get install git

1. Install deps Package

go get -d -u github.com/golang/dep

cd $(echo $GOPATH)/src/github.com/golang/dep

DEP\_LATEST=$(git describe --abbrev=0 --tags)

git checkout $DEP\_LATEST

go install -ldflags="-X main.version=$DEP\_LATEST" ./cmd/dep

git checkout master

3) Set Up Environment Variables

Edit the ~/.bashrc file and add the following environment entries:

export GOPATH=$HOME/gorepo

export GOROOT=/usr/local/go

export GOOS=linux

export GOARCH=amd64

export CGO\_ENABLED=0

export PATH=$GOPATH/bin:$GOROOT/bin:$PATH

4) Build the telegraf application server

1. Run cd $GOPATH/src/github.com/
2. Run mkdir influxdata
3. Run cd influxdata
4. Run git clone <https://github.com/davidgs/telegraf.git> -b GoogleIoT
5. Run cd $GOPATH/src/github.com/influxdata/telegraf
6. Run make
7. Run mkdir $HOME/conf
8. Run ./telegraf config > telegraf.conf
9. Run mv telegraf.conf $HOME/conf

5) Generate Self-signed SSL Certificates and Private Key for Testing Environment Only

cd $HOME/conf

openssl genrsa -out example.key 2048

openssl req -new -key example.key -out example.csr

openssl x509 -req -days 365 -in example.csr -signkey example.key -out example.crt

6) Set up the telegraf server to receive data from GCP Pub/Sub

Replace the Section “[[inputs.googlecoreiot]]” in the file $HOME/conf/telegraf.conf with the following configurations. In the configuration below, you must set “tls\_cert” to the path of the SSL certificate file, “tls\_key” to the path of the private key file and “tls\_allowed\_cacerts” key to the path of the CA certificate file to enable mutually authenticated TLS connections.

For testing purpose, you can just use both the self-signed SSL certificate and private key in the previous step #5 above and then comment out the line that begins with “tls\_allowed\_cacerts” in the code below. But for production environment, you really need to use the CA certificate.

# # Influx HTTP write listener

[[inputs.googlecoreiot]]

# ## Address and port to host HTTP listener on

service\_address = ":9999"

#

# ## Path to serve

# ## default is /write

path = "/write"

#

# ## maximum duration before timing out read of the request

read\_timeout = "30s"

# ## maximum duration before timing out write of the response

write\_timeout = "30s"

#

# # precision of the time stamps. can be one of the following:

# # second

# # millisecond

# # microsecond

# # nanosecond

# # Default is nanosecond

#

precision = "nanosecond"

#

# # Data Format is either line protocol or json

# protocol="line protocol"

protocol = "json"

# ## Set one or more allowed client CA certificate file names to

# ## enable mutually authenticated TLS connections

# comment out this line in testing environment

tls\_allowed\_cacerts = ["/home/user/conf/ca.pem"]

#

# ## Add service certificate and key

tls\_cert = "/home/user/conf/example.crt"

tls\_key = "/home/user/conf/example.key"

7) Install and run data collection application servers

1. Download and install influxdb and chronograf

wget https://dl.influxdata.com/influxdb/releases/influxdb\_1.4.0\_amd64.deb

sudo dpkg -i influxdb\_1.4.0\_amd64.deb

wget https://dl.influxdata.com/chronograf/releases/chronograf\_1.4.0.0\_amd64.deb

sudo dpkg -i chronograf\_1.4.0.0\_amd64.deb

b) Start up the application services

service influxdb start

sudo service chronograf start

8) Run the telegraf server to start receive data from the GCP.

sudo $HOME/gorepo/src/github.com/influxdata/telegraf -config $HOME/conf/telegraf.conf &

## Set Up Configurations to Push Data on GCP to Telegraf

There are two ways to push data to remote hosts. Below are the setup instructions for both in which you can utilize.

1) Use the push delivery type for the subscription

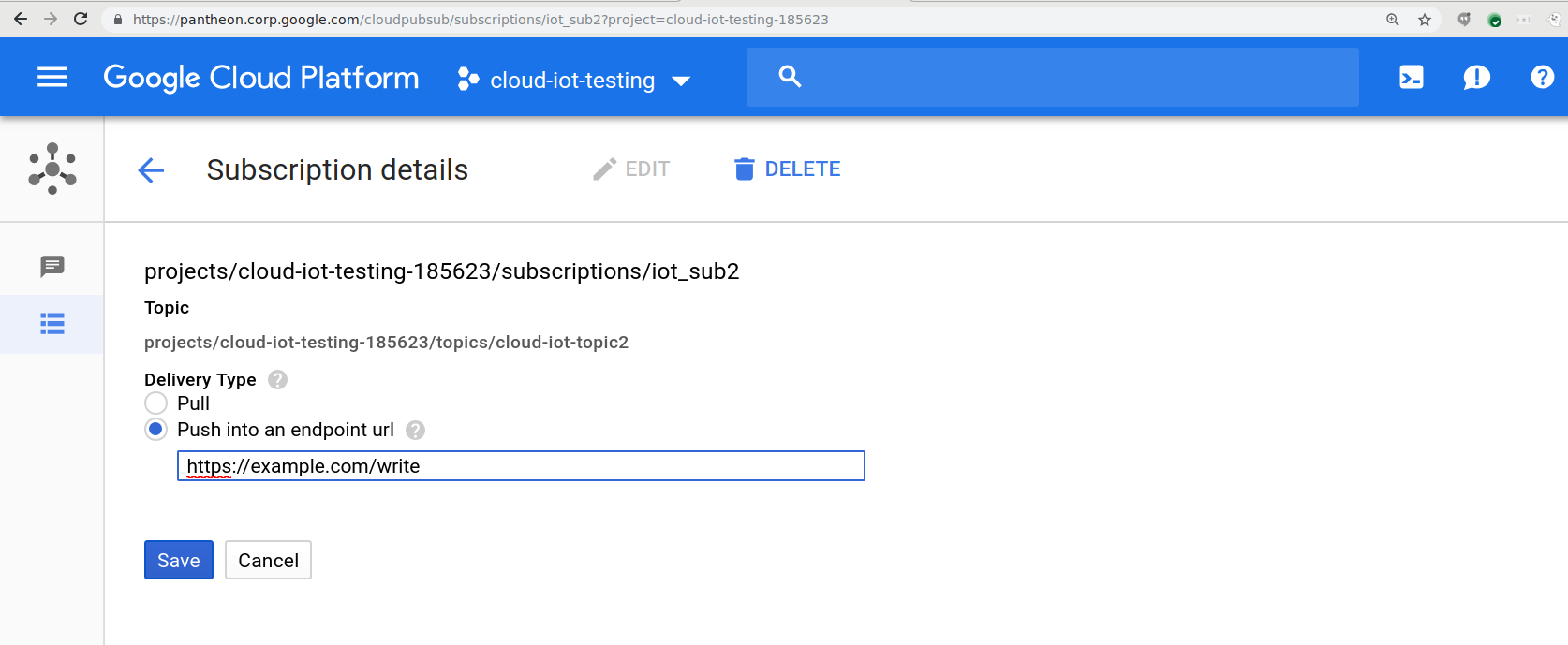
Use gcloud command to update subscription from pull to push type. Make sure to replace <subscription\_name> and <url\_to\_telegraf\_server> with actual values.

gcloud alpha pubsub subscriptions update <subscription\_name> --push-endpoint=<url\_to\_telegraf\_server>/write

Ex. gcloud alpha pubsub subscriptions update [iot\_sub2](https://pantheon.corp.google.com/cloudpubsub/subscriptions/iot_sub2?project=cloud-iot-testing-185623) --push-endpoint=<https://example.com/write>

Use GCP dashboard

Go to the Pub/Sub subscription and enter the complete url of the telegraf server with an appending string “/write” into the “Push into an endpoint url” box as shown below.

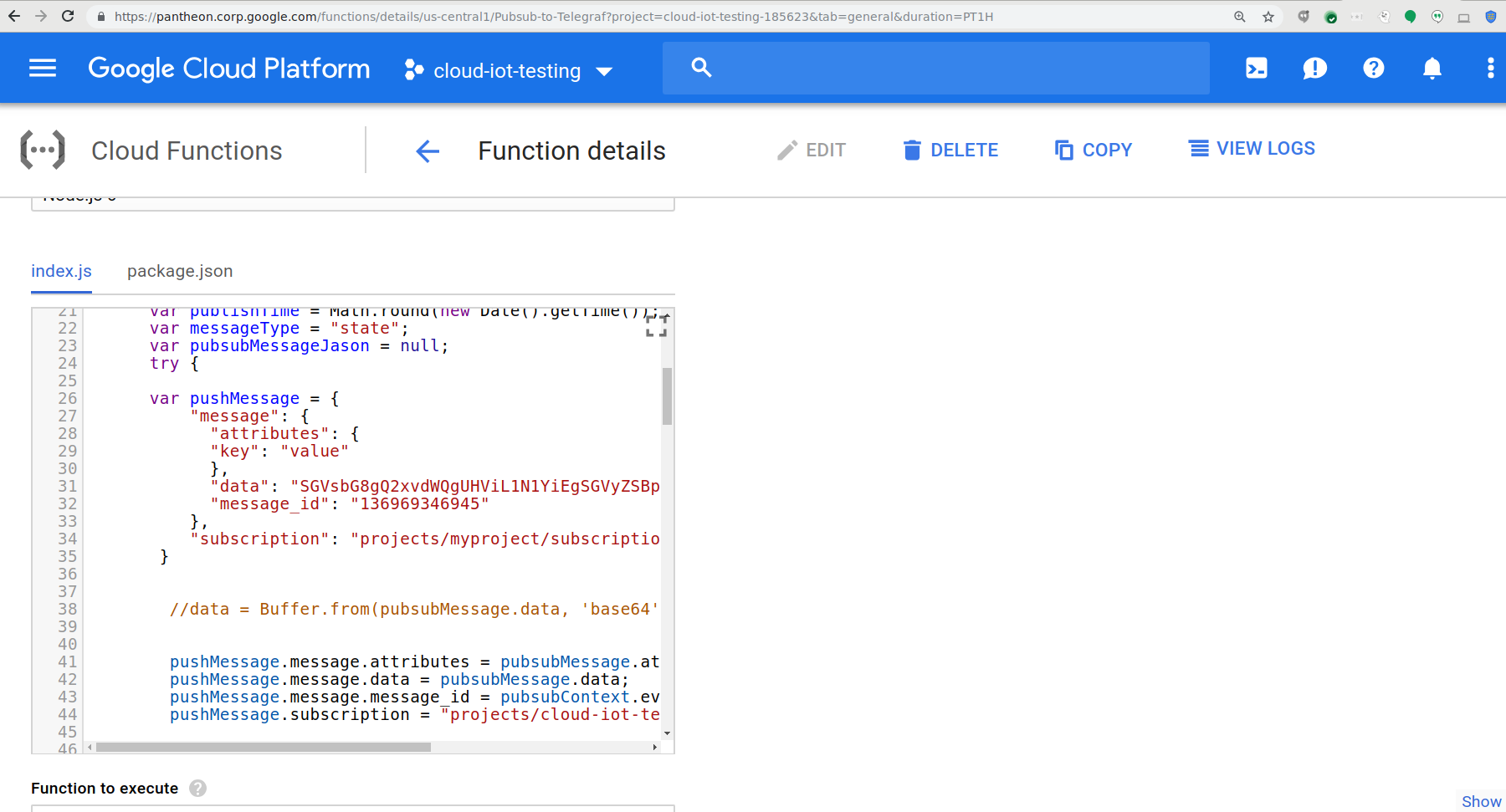


2) Use the Cloud Function Code

Instead of using the endpoint url to push data to telegraf application server, you can also use cloud function with the same effect. Let’s navigate to the “Cloud Function” link on GCP and create a cloud function and then fill in the code for both files index.js and package.json as illustrated on the screen below.

In the index.js file, you must set the pkey variable with private key, cert variable with signed certificate and ca variable with CA certificate, and subscription variable with the path to the subscription. These certificate and private key files are the same ones as we have under the “[[inputs.googlecoreiot]]” section in the telegraf.conf configuration file.

In addition, you must replace the values for both host and port in the options variable with your own.



index.js

/\*\*

\* Triggered from a message on a Cloud Pub/Sub topic.

\*

\* @param {!Object} event The Cloud Functions event.

\* @param {!Function} The callback function.

\*/

const request = require('retry-request');

const fs = require('fs');

const pubSub = require('@google-cloud/pubsub');

const BigQuery = require('@google-cloud/bigquery');

var https = require('https');

var querystring = require('querystring');

var subscription = "projects/cloud-iot-testing/subscriptions/iot\_sub2";

exports.pubsubtoserver = (event, callback) => {

const pubsubMessage = event.data;

const pubsubContext = event.context;

var data = null;

var payload = [];

var publishTime = Math.round(new Date().getTime());

var messageType = "state";

var pubsubMessageJason = null;

try {

var pushMessage = {

"message": {

"attributes": {

"key": "value"

},

"data": "SGVsbG8gQ2xvdWQgUHViL1N1YiEgSGVyZSBpcyBteSBtZXNzYWdlIQ==",

"message\_id": "136969346945"

},

"subscription": "projects/myproject/subscriptions/mysubscription"

}

//data = Buffer.from(pubsubMessage.data, 'base64').toString();

pushMessage.message.attributes = pubsubMessage.attributes;

pushMessage.message.data = pubsubMessage.data;

pushMessage.message.message\_id = pubsubContext.eventId;

pushMessage.subscription = subscription;

var jsonData = JSON.stringify(pushMessage);

//delete pubsubMessage.attributes;

//delete pubsubMessage["@type"];

console.log("data: " + jsonData);

var pkey = '-----BEGIN RSA PRIVATE KEY-----\n\

MIIEowIBAAKCAQEAqIBbtcgHBUfGim840CxBNJYNdj/G56AAKmnSMQXTmtmQfCNl\n\

9Oq7ofnaYI0caOvKNmptPKtnNKRwVRpt9erb8q4sQdxAUacwDH9O\n\

-----END RSA PRIVATE KEY-----';

var cert = '-----BEGIN CERTIFICATE-----\n\

MIIDTjCCAjYCCQCEhtewoJYi4DANBgkqhkiG9w0BAQsFADBpMQswCQYDVQQGEwJV\n\

9eND1jOXWEMIk8HPc86URRx1OxAkQ2ws3S/Kwkv0+bQ6jnhnd0IiB5v0+tPkT2F8\n\

xwndLO9QdeYxQgs6z3pvTTnaZRLlxncONDigPTWe+tCiHC22SD+2wZEfS4cVzykN\n\

2O5+HBSPbyFlVNGWtwOMdLyjarQzWeqcpQOEzS+TdeU+ifNaSIjQnBUCAwEAATAN\n\

AAxkl+c1uM2rqjfuHHtIXipbp3pwCUyw/6nnKyI5j5A42g==\n\

-----END CERTIFICATE-----';

var ca = '-----BEGIN CERTIFICATE-----\n\

MIIDqDCCApCgAwIBAgIJAK1Z8uhd6t2OMA0GCSqGSIb3DQEBCwUAMGkxCzAJBgNV\n\

dFvbRmwigTnh9q1HcEcp9GEalYQcKWWnz2O6zuSvGxczhfGlr18H4EAPdJSLqxol\n\

G7jr7uoPhXpaONeZIWyCLdmAB1i3wOVNRdm6wyzmuZRTCrOBzLCdHb62tVDS5GMy\n\

a0nvBMKtJ1XSmI9IC1AJxjrAzDfRsWhzx7ZObQ==\n\

-----END CERTIFICATE-----';

var options = {

host: '35.192.227.193',

port: 443,

method: 'POST',

path: '/write',

key: pkey,

cert: cert,

ca:ca,

rejectUnauthorized: false,

//requestCert: true,

agent: false,

headers: {

'Content-Type': 'application/json',

'Content-Length': jsonData.length

}

};

// request object

var req = https.request(options, function (res) {

var result = '';

res.on('data', function (chunk) {

result += chunk;

});

res.on('end', function () {

console.log(result);

});

res.on('error', function (err) {

console.log(err);

})

});

// req error

req.on('error', function (err) {

console.log(err);

});

//send request witht the postData form

req.write(jsonData);

req.end();

} catch (err) {

console.error(err);

throw new Error(

`"Invalid Data.`

);

}

};

package.json

{

"name": "sample-pubsub",

"version": "0.0.1",

"dependencies": {

"@google-cloud/pubsub": "0.18.0",

"@google-cloud/storage":"1.6.0",

"jsonwebtoken": "7.4.1",

"retry-request": "3.3.1",

"@google-cloud/bigquery" : "1.2.0",

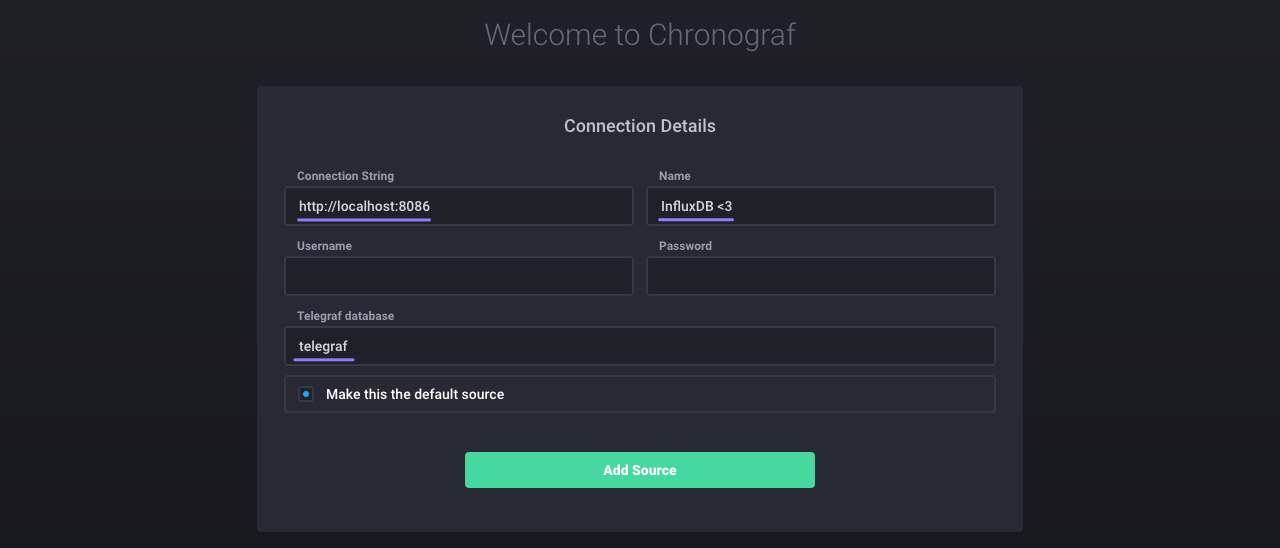
"https": "\*"

}

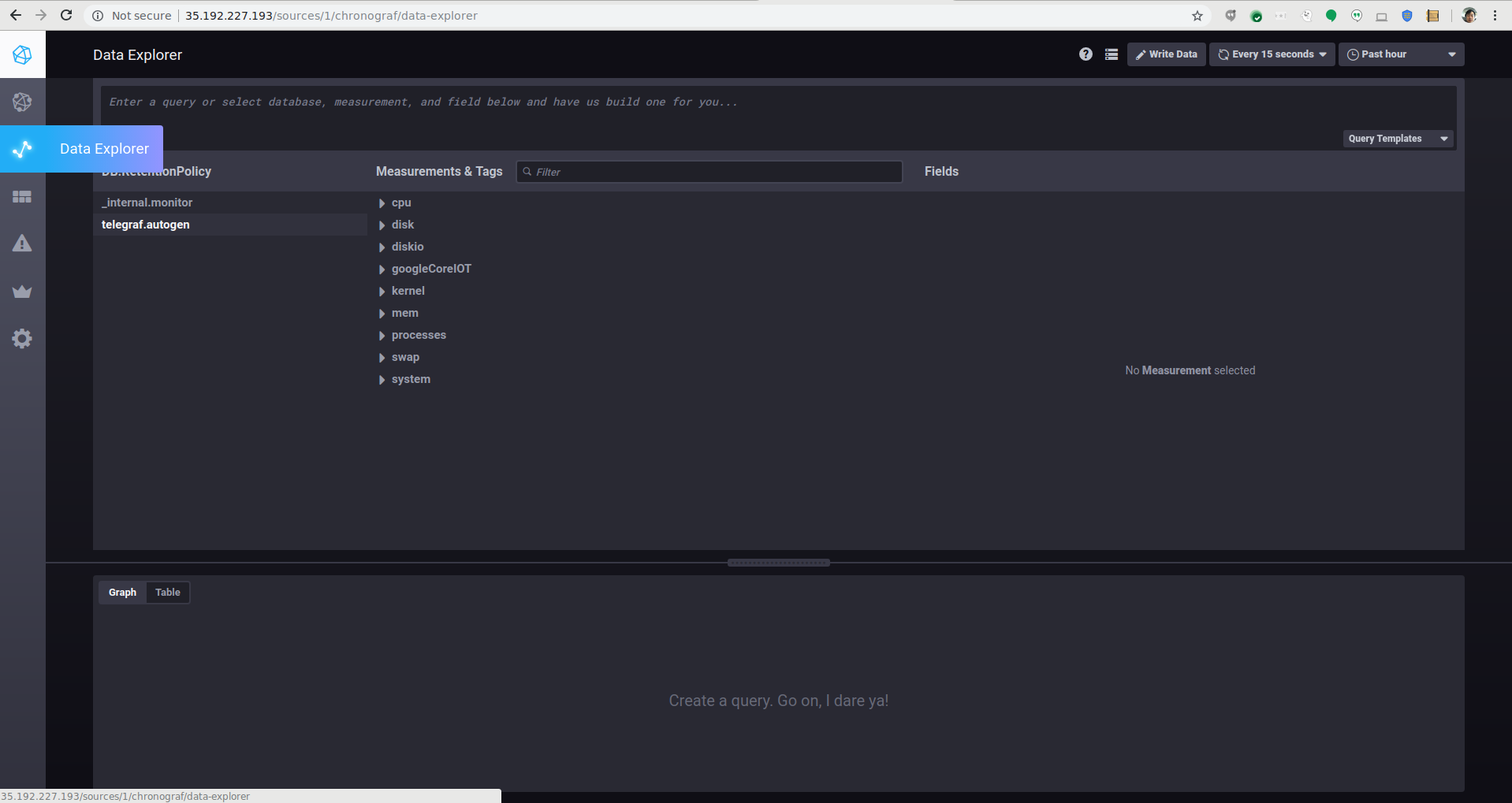
}

## Verify published data collected from the GCP

Bring up any browser and navigate to the url of the chronograf application server on port #8888. This server is running on the default port #8888 - url ex. [http://<chronograf\_server\_url>:8888](http://chronograf_server_hostname:8888). Next, click on “Add Source” to add the influxdb application server to the chronograf application server for it to live stream data on its dashboard.



On the dashboard, click on “Data Explorer” and then select telegraf.autogen. Under Measurements and Tags, select “googleCoreIOT” link to expand it.

￼

To see graph reports in real time, select “humidity” and “temp\_c” options under Fields as shown below:

