BUILDING SERVERLESS DATA PIPELINE
IOT TO ANALYTICS

Advancements in Internet and Telecommunications (ICT) accelerated the large –scale deployments of IoT applications including smart city, smart healthcare, and smart factory aiming for faster data processing. The latency and other challenges of cloud-centric IoT, driven edge computing-based data processing architectures. In precise, IoT data analytic requires control for dealing with the complete life cycle of data flow between data sources to sinks by building a set of data pipelines seamlessly deployed on the IoT computing continuum (Edge and clouds).

The execution of data analytic tasks is challenging in the IoT computing continuum due to heterogeneous hardware architecture at edge and cloud environments. This can be succeeded using Serverless or Function as a service (Faas) cloud computing model, where in tasks are defined as virtual functions and seamlessly migrated and executed within the computing continuum. The serverless data pipelines constitutes a data source, a set of serverless functions that performs specific operations on the data and then ultimately a data sink. Similarly, AWS, Microsoft Azure and Google cloud solutions for IoT data processing that includes the serverless entities as a part of the system. However, cost and latency vary according to cloud provider subscriptions and architecture components.

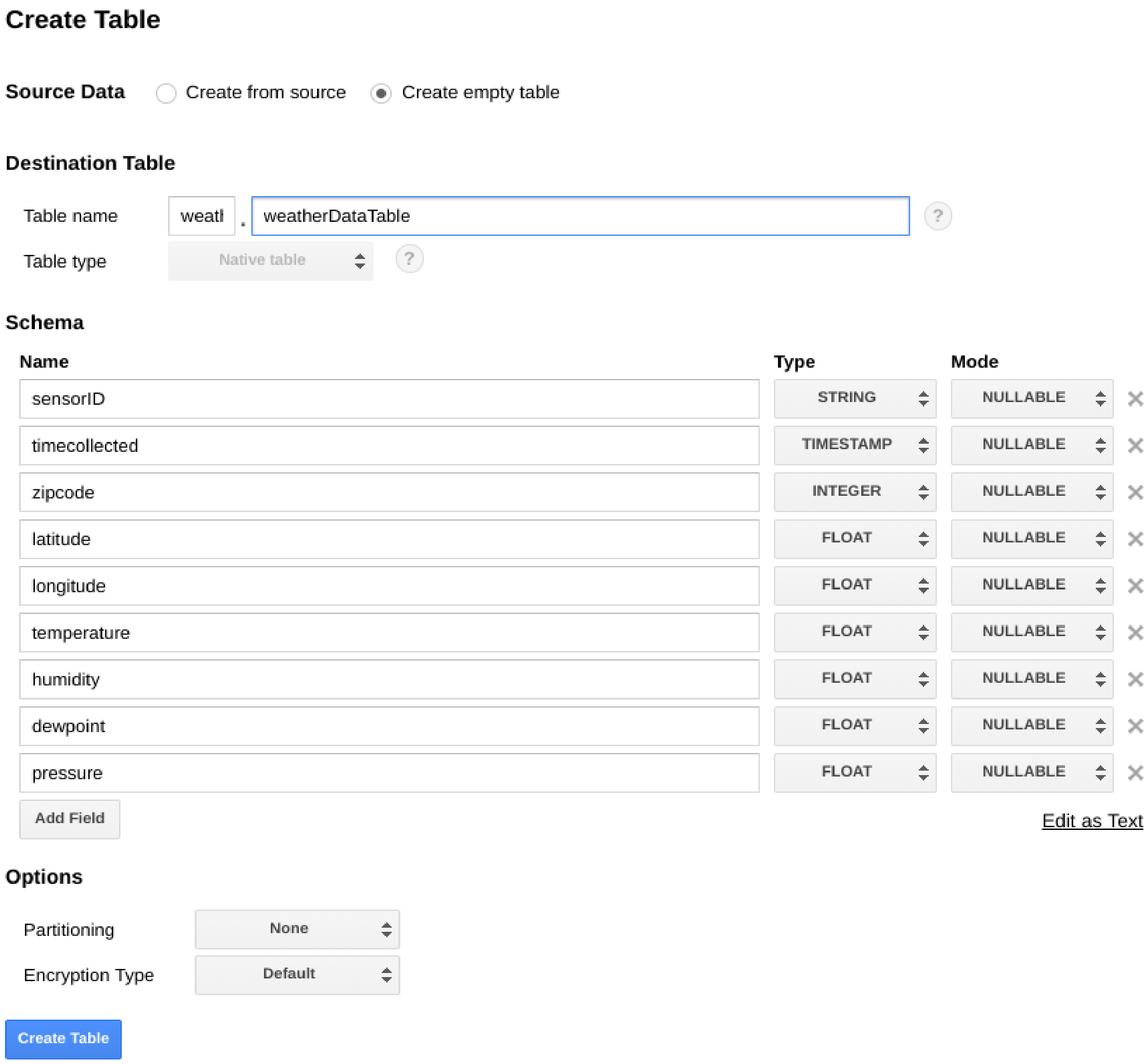
So, our proposed work outlines the cloud service provider specific IoT services that are composed in designing the serverless data pipelines (SDP) for IoT data processing. Further, we investigate their performance in terms of latency, cost and cold/warm start time of serverless functions for IoT data-oriented pipelines. Aligning to this, we propose AWS and Microsoft Azure based SDP architectures for predictive analytics in Industrial IoT environments and implement the real time predictive Maintenance of Industrial Motor application. Accordingly, we measure the efficiency of AWS and Azure SDP architecture for various performance metrics such as cost, processing time and cold/warm start time for serverless invocations.

The Raspberry Pi Zero W is recommended for this code lab, but a Raspberry Pi 3 Model B will also work, won't require hammer header pins or a USB hub, has a better CPU, but will cost more. If you choose to move forward with the alternate device, make certain that you have a power supply and an appropriate SD card.

Create a Big Query table

Big Query 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 the IoT weather data. Select Big Query from the cloud console. This will open Big Query in a new window.



Create Table

Source Data ○ Create from source ● Create empty table

Destination Table

Table name: weatl . weatherDataTable

Table type: Native table

Schema

Name	Type	Mode
sensorID	STRING	NULLABLE
timecollected	TIMESTAMP	NULLABLE
zipcode	INTEGER	NULLABLE
latitude	FLOAT	NULLABLE
longitude	FLOAT	NULLABLE
temperature	FLOAT	NULLABLE
humidity	FLOAT	NULLABLE
dewpoint	FLOAT	NULLABLE
pressure	FLOAT	NULLABLE

Options

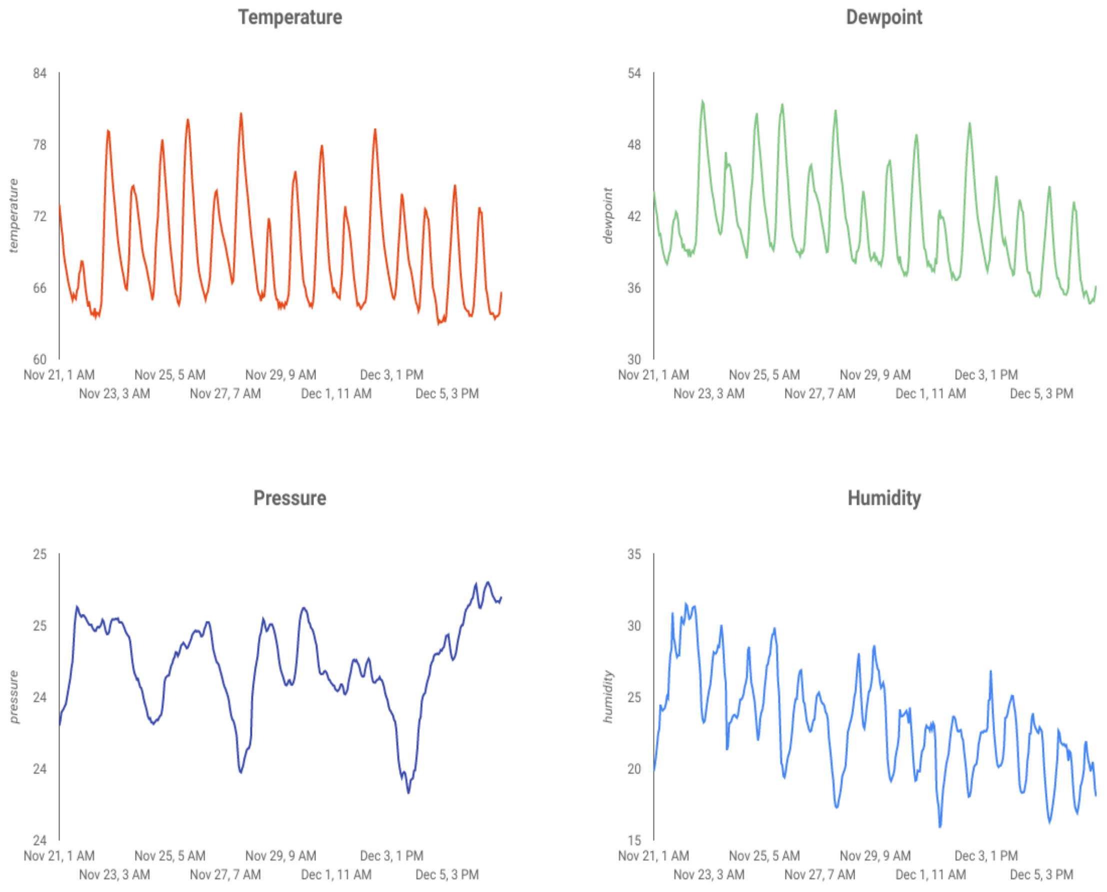
Partitioning: None

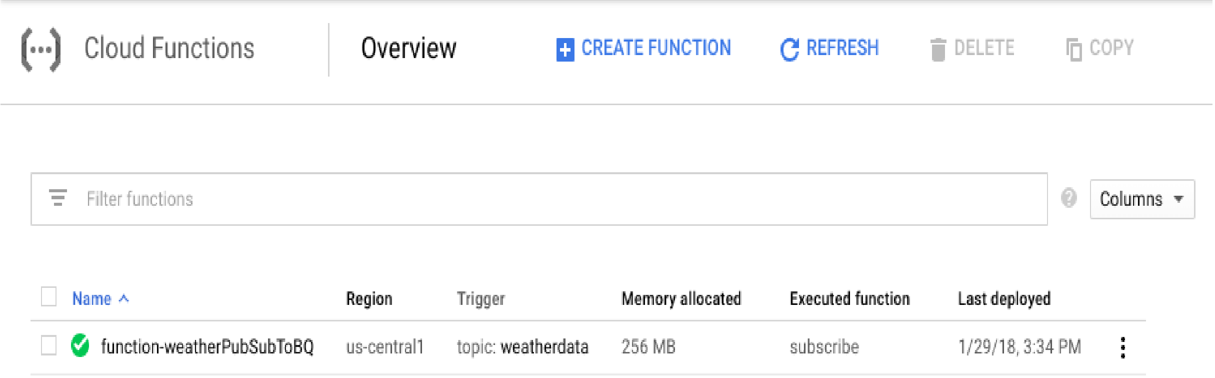
Encryption Type: Default

```
/**
 * 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','dewpoint':'-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"
  }
}
```





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

Start the data pipeline

Data streaming from a Raspberry Pi

If you constructed a Raspberry Pi IoT weather sensor, start the script that will read the weather data and push it to Google Cloud Pub/Sub. If you aren't in the /home/pi/iot-data-pipeline directory, move there first

```
cd /home/pi/iot-data-pipeline
```

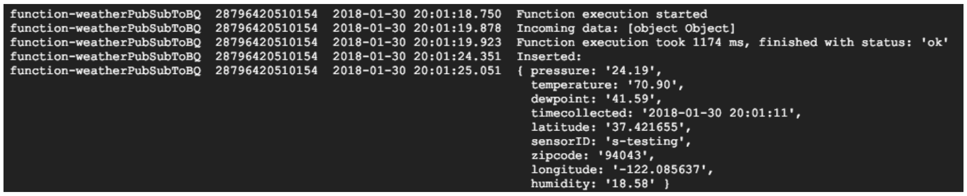
Start the weather script

```
python checkWeather.py
```

Check that data is flowing

```
gcloud beta functions logs read function-weatherPubSubToBQ
```

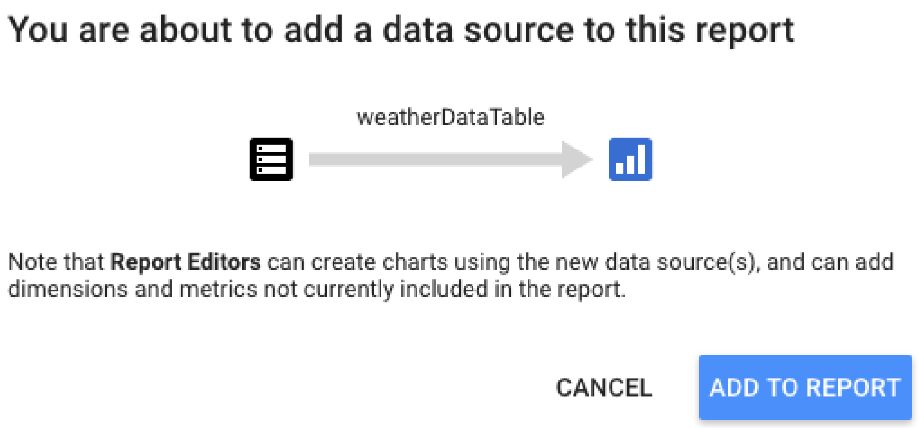
The logs should show that the function is executing, data is being received and that it is being inserted into Big Query



```
function-weatherPubSubToBQ  28796420510154  2018-01-30 20:01:18.750  Function execution started
function-weatherPubSubToBQ  28796420510154  2018-01-30 20:01:19.878  Incoming data: [object Object]
function-weatherPubSubToBQ  28796420510154  2018-01-30 20:01:19.923  Function execution took 1174 ms, finished with status: 'ok'
function-weatherPubSubToBQ  28796420510154  2018-01-30 20:01:24.351  Inserted:
function-weatherPubSubToBQ  28796420510154  2018-01-30 20:01:25.051  { pressure: '24.19',
                                                                       temperature: '70.90',
                                                                       dewpoint: '41.59',
                                                                       timecollected: '2018-01-30 20:01:11',
                                                                       latitude: '37.421655',
                                                                       sensorID: 's-testing',
                                                                       zipcode: '94043',
                                                                       longitude: '-122.085637',
                                                                       humidity: '18.58' }
```

Create a data studio dashboard

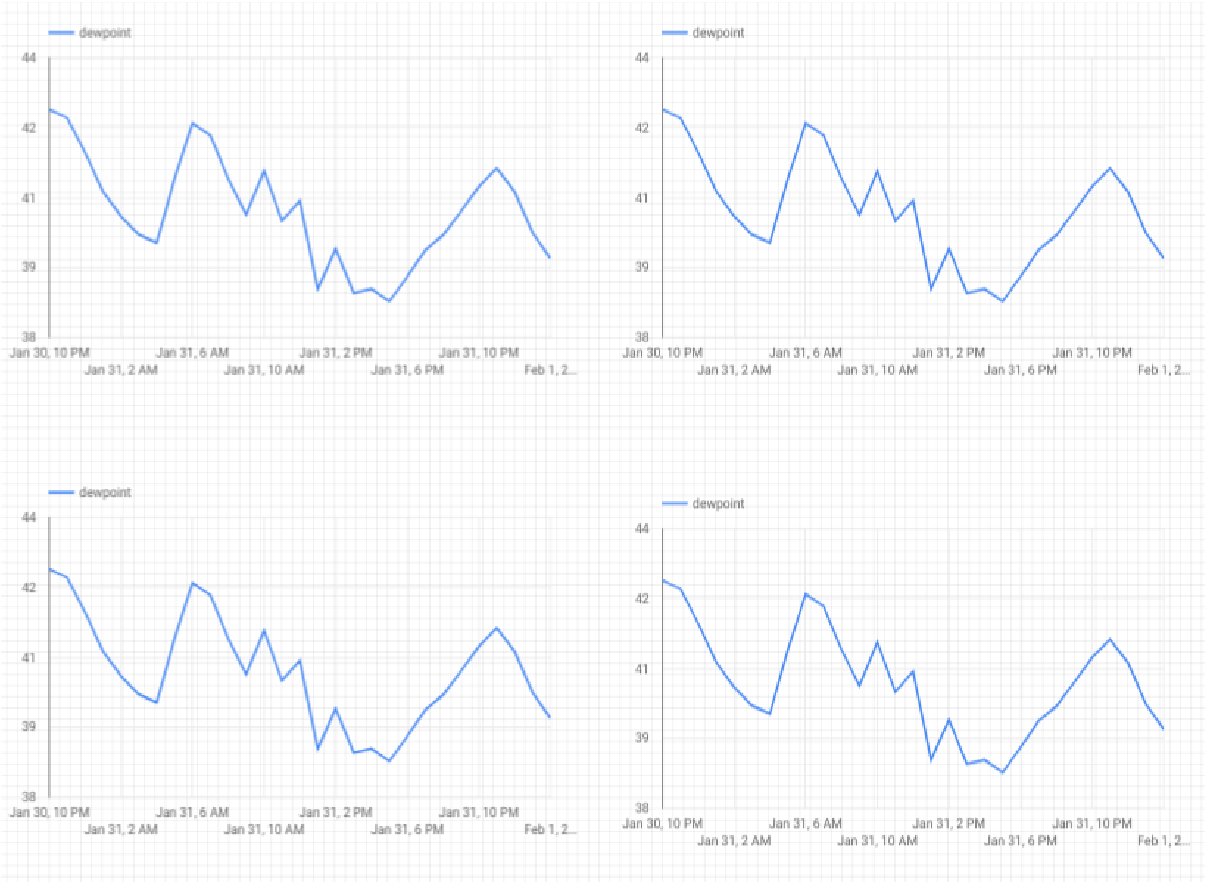
Confirm by clicking the add a data report button



You are about to add a data source to this report

Note that **Report Editors** can create charts using the new data source(s), and can add dimensions and metrics not currently included in the report.

Click the graph on the sheet and copy/paste (Ctrl-C/Ctrl-V) it 3 times. Align the graphs so that each has ¼ of the layout.



Conclusion

We've created an entire data pipeline! In doing so, we've learned how to use Google Pub/Sub, how to deploy a serverless Function, how to leverage Big Query and how to create an analytics dashboard using Data Studio. In addition, we've seen how the Google Cloud SDK can be used securely to bring data into the Google Cloud Platform. Finally, we now have some hands-on experience with an important architectural pattern that can handle high volumes while maintaining availability.