The University of British Columbia Okanagan School of Engineering

ENGR 453

Cloud Project

Student #1 Name: Graeme Paul
Student #2 Name: Daara Abbassi-Mohadjel

Submitted: April 23^{rd}, 2020

1 Introduction

For our cloud project, there will be three devices which will work together to control the Raspberry Pi as it records a temperature and controls a heating lamp. The purpose of the project is to 'cure' a composite which is in a curing oven. The internal temperature of the part is indeterminable thus it is unknown how long the part is required to cure. Figure 1 shows the typical heating curve of an object in the curing oven. A machine learning algorithm has been trained with the internal temperature data from similar composite parts thus by knowing the air temperature and current time the internal temperature of the current part can be found. A Raspberry Pi recording time and air temperature will be used with a machine learning algorithm on the cloud virtual machine which will make a prediction when to stop the Raspberry Pi.

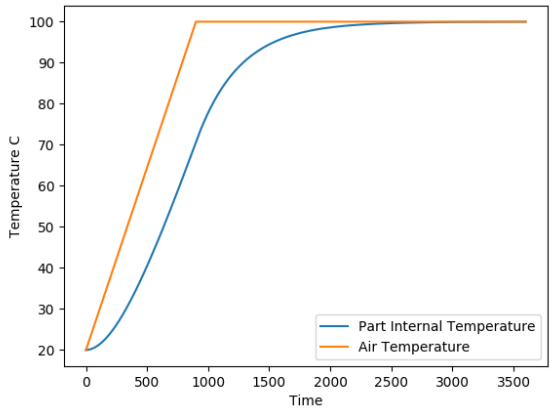

Figure 1: Heating Curve of Composite Part in Curing Oven

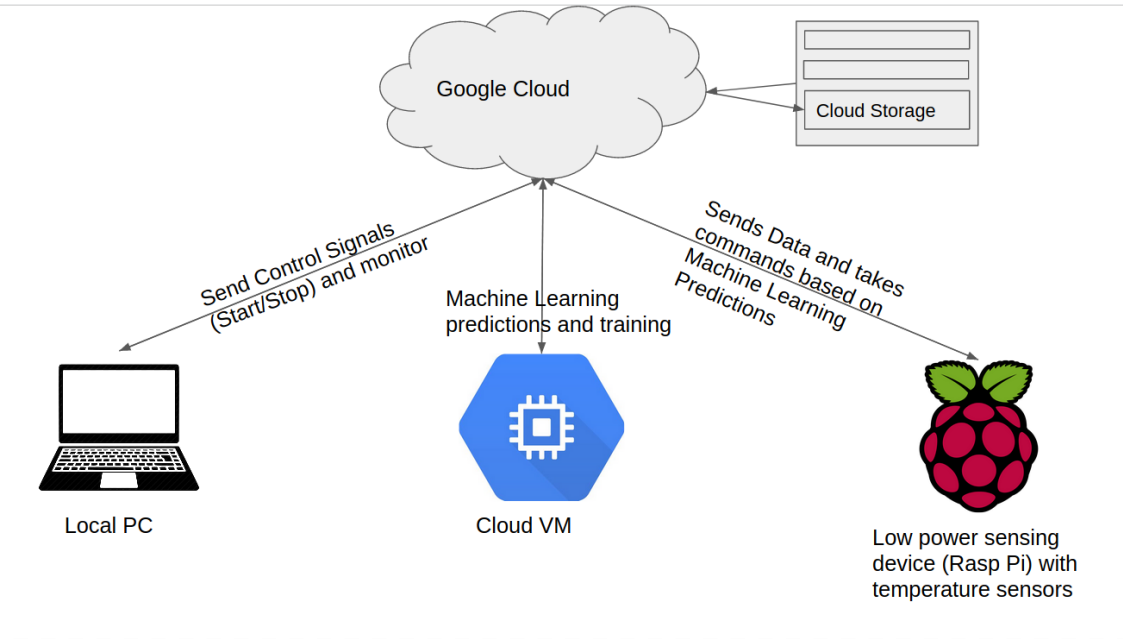

Figure 2: Data flow the IOT network

The full cycle of data is as follows. First, the local PC will be the initial stimulus of the system. When a start command is sent using MQTT, the cloud will act on this and initiate the Raspberry Pi to start heating the oven and recording air temperature and time. The collected data will be sent to the virtual machine. A prediction is done by the virtual machine using the temperature data halfway through the run and is used to determine a reasonable stop time. The Raspberry Pi will be told to stop by the virtual machine and the run data and prediction will be sent back to the control PC for review. A summary of this data flow is shown in Figure 2.

Additional notes:
Due to circumstances and no access to real data the system is using machine generated data which is essentially equivalent. No Raspberry Pi will be used but instead just another computer which will act as the Raspberry Pi and no temperature sensors will be used as neither group member has access to the equipment. Essentially, the cloud framework will be developed to show a proof of concept that control of the device is possible over the cloud.

2 Design

There were a number of considerations taken into account for the design of this project and are discussed below. For a video summary of the working deign see Section 4 Example of Performance.

2.1 Data Payload

The payload that is communicated between the devices, the virtual machine data logger and the cloud virtual machine, contain several pieces of information necessary for generating a prediction. A message sent as `data_payload` is sent as a dictionary seen from the line of code below, there are four keys in the dictionary.

```
data_payload = json.dumps(\{"Type": TYPE, "Data" :DATA, "Time" : TIME, "To" : DEVICE\})
```

First is the `"Type"` which signifies an update of the state for a device if set to 1 ,and when set to 0 it will signify temperature or air data for the virtual machine or control computer. Another item in the payload is the `"Time"`, which returns time at the instances payloads are sent throughout the process. The `"Data"` key corresponds to data that is used to run the process, such as sending the starting temperature and the stop time to the Raspberry Pi. The last key `"To"` is for setting the destination device for the payload to be sent to. The `"To"` flag is only used by the cloud to direct messages to the correct device. The below code shows an example of the VM acting on data that it is receiving, first it determines if the message is data or a configuration update by checking the `"Type"`, then it extracts the data from the `"Data"` key. It can be seen that some configurations include to start predicting, to stop/done and to reset.

```
    def on_message(self, unused_client, unused_userdata, message):
        index = 0
        """Callback when the device receives a message on a subscription."""
        payload = message.payload.decode('utf-8')
        payload_dict = json.loads(payload)
        if(payload_dict["Type"] == 0): #It's Data
            for i in payload_dict["Data"]["Index"]:
                print(payload_dict["Data"][str(i)]["Temp"])
                self.temps[0,i,0] = float(payload_dict["Data"][str(i)]["Temp"])
                self.temps[0,i,1] = float(payload_dict["Data"][str(i)]["Time"])

        if(payload_dict["Type"] == 1): #It's Configuration Update
            if(payload_dict["Data"] == "Predict"):
                self.start_predict = True
                print("Starting Prediction")
            if(payload_dict["Data"] == "Done"):
                self.save_data = True
                print("Time to save Data")
            if(payload_dict["Data"] == "Reset"):
                self.start_predict = False
                self.save_data = False
                self.temps = np.zeros((1,360,2)) #two features air and out temp
                self.reset = True
            if( "Run_ID"  in payload_dict["Data"]):
                self.Run_ID = payload_dict["Data"]["Run_ID"]
                print("ID set to {}".format(self.Run_ID))
```

```
        return
```

2.2 Data Collection and Data Processing

Once the Raspberry Pi receives the signal to begin the data collection process from the control PC, air temperature data is recorded from temperature sensors inside the heating chamber. For this project, we decided to have the Raspberry Pi send the temperature data in batches to the Cloud virtual machine so that the virtual machine may act on the data in real time. The Cloud virtual machine is storing the measurements from the Raspberry Pi into an array and checking for the temperature to reach steady state, and afterwards a set time will be sent back to the Raspberry Pi to arrange the moment the infrared lamp will turn off. The array index is also sent with each batch of data so that the data does not get disorganized, as there is no guarantee that the order that data is received is the order it was sent. The data collection process continues temporarily after the lamp is turned off because in a practical setting the temperature would then start to decrease, but as this is only a proof of concept, the temperature remains at steady state in our demonstration.

2.3 Control Computer

The control computer takes user input and is able to start, stop, or reset a test by updating the other devices configurations. The control computer also sets the run ID which makes it so that the saved data from the run is a unique filename when written to the cloud storage bucket. The control computer will receive the prediction and all of the collected air temperature at the end of the test so that it can be saved and plotted. Figure 1 shown in the introduction is the output from this plotting feature. Below is the code which implements these features.

```
    while(True):

        print("Start Temp to device 1 (test-dev) to start a test")
        print("Restart to any device to restart")
        #Use this file to Send Message based on User Input
        print("Send Message:")
        usrInputMsg=input()
        print("Data Type:")
        data_type=input()
        if(data_type == "Plot"):
            prediction = device.get_prediction_data()
            air_temp_data = device.get_air_temp_data()
            pyplot.plot(prediction)
            pyplot.plot(air_temp_data)
            pyplot.show()
        elif(data_type == "ID"):
            Run_ID_dict = {"Run_ID" : usrInputMsg}
            payload = json.dumps({'Type': 1, 'Data' : Run_ID_dict, 'To' : "test-dev2", 'Time
                ' : 0})
            print('Publishing payload', payload)
            client.publish(mqtt_telemetry_topic, payload, qos=1)
        elif(data_type == "Start"):
```

```
            payload = json.dumps({'Type': 1, 'Data' : "Start Temp", 'To' : "test-dev", 'Time
                ' : 0})
            print('Publishing payload', payload)
            client.publish(mqtt_telemetry_topic, payload, qos=1)
        else:
            print("To?:")
            usrInputAddr=input()
            # Report the device's temperature to the server by serializing it
            # as a JSON string.
            payload = json.dumps({'Type': int(data_type), 'Data' : usrInputMsg, 'To' :
                usrInputAddr, 'Time' : 0})
            print('Publishing payload', payload)
            client.publish(mqtt_telemetry_topic, payload, qos=1)
```

2.4 Prediction

Keras was chosen for this project due to its accessibility in implementing machine learning algorithms using Tensorflow in Python. The machine learning algorithm used was a Long Short-term Memory (LSTM) neural network and the algorithm was trained using data generated in MATLAB. The predictions for the part temperature act off of the time and air temperature that is sent from the data logger outlined in Section 2.2. Once the virtual machine detects the plateau of the air temperature, a prediction is then made for the part temperature and the data is stored. The code shown below is the prediction process. First the pre-trained Keras model is loaded, then the received data is scaled and used to predict.

```
        model = load_model("model_weights_data2_lstm.h5")

        #val_scaled = self.temps
        val_data_bak = self.temps
        val_scaled = np.zeros((360,1,2))
        #Scale data that change during run this way
        for j in range(val_scaled.shape[0]):
            scalers[0] = MinMaxScaler(feature_range=(0, 1))
            scalers[1] = MinMaxScaler(feature_range=(0, 1))
            val_scaled[:,0, 0:1] = scalers[0].fit_transform(val_data_bak[0,:,1:2])
            val_scaled[:,0, 1:2] = scalers[1].fit_transform(val_data_bak[0,:,0:1])
        #print(val_scaled[0:50,:,:])
        print(val_scaled[:,0:1,:].shape)
        yhat = model.predict(val_scaled[:,0:1,:] , batch_size = local_batch_size)
        val_scaler = MinMaxScaler(feature_range=(0,1)).fit(self.temps[0,:,0:1])
        inv_yhat_out = val_scaler.inverse_transform(yhat)
        print(yhat)
        print(inv_yhat_out)
        stop_time = 360
        for i in range(0, len(inv_yhat_out)):
            if(inv_yhat_out[i] > 98):
                stop_time = self.temps[0,i,1]
                break
                print("Stop Time is: "+str(stop_time))
        return stop_time, inv_yhat_out
```

2.5 Google Cloud

In this project, all the devices use MQTT to connect to the Google Cloud device registry. The MQTT message goes to a Cloud Pub/Sub topic, therefore any message published to a topic is immediately received by all of the subscribers to the topic. The Pub/Sub messaging provides a simple and reliable staging location for the event data and acts to ingest events so that it can be acted upon. The Cloud Function will then act on the event data and can send messages to the devices through MQTT. We are able to implement any number of IoT devices using this system. Buckets are also used to permanently store data in the Cloud.

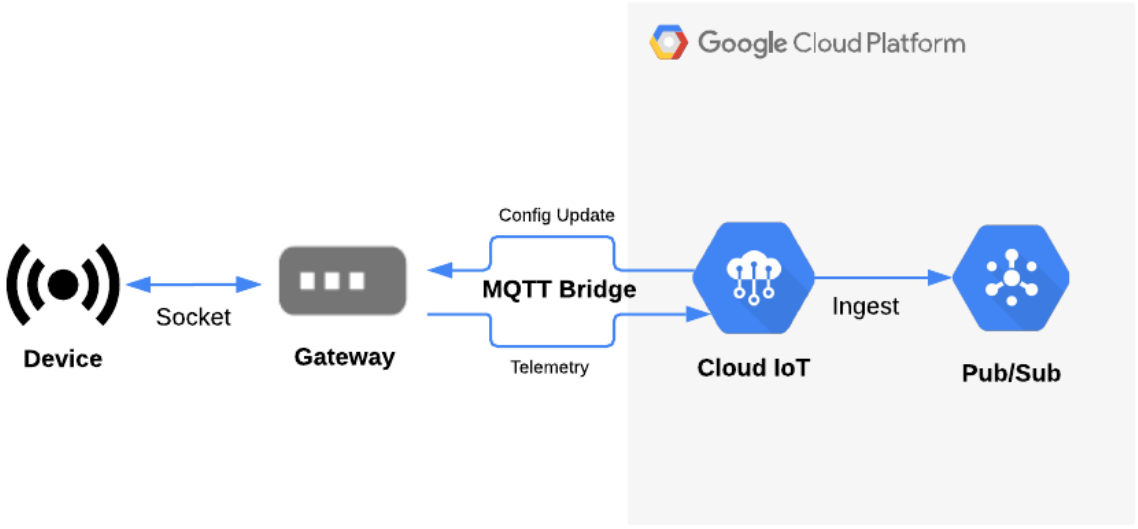

Figure 3: A Google Cloud Implementation

2.6 Code License

As the code used for this project was modified and adapted from Google Cloud examples which were using the Apache license, we also needed to use the Apache License due to copyleft.

2.7 Virtual Machine

The Cloud virtual machine used in this project is a standard Linux virtual machine. The virtual machine from Google Cloud accelerates the prediction speed and allows the Raspberry Pi to continue to measure temperatures so that there is minimal data loss while a live prediction is made.

2.8 AWS vs Google Cloud Comparison

There are a large number of differences between the Google Cloud service and AWS. AWS offers more than 200 services and is more feature-rich compared to Google Cloud. Google Cloud has the advantage in terms of flexibility with far greater opportunities for customization of compute instances compared to AWS. Furthermore, the Google Cloud platform has more advanced virtual machines than AWS. Lastly, Google Cloud had advantages in cost for the implementation of our project.

2.9 Costs

The costs for this project are primarily derived from leaving the virtual machine on in the Cloud, which resulted in roughly $7 in costs. These costs were accumulated over multiple days of testing and validating the system, sending messages, and having the virtual machine on. This was not a concern as we had a complimentary $350 dollars for use.

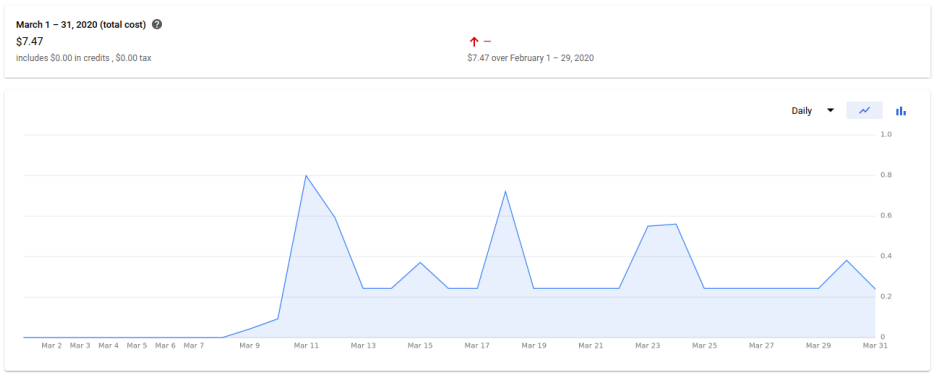

Figure 4: Costs

2.10 Implementation Assumption

Due to limited access to the university's labs, we were unable to gain access to the heat chamber setup, so a Raspberry Pi was not actually used and is represented by a Linux terminal due to these restrictions and is used for demonstrative purposes and to make the demonstration video easier to show. For additional testing purposes, two separate computers were used a laptop and desktop and it was successful, therefore it should be able to work with a Raspberry Pi.

3 Code Analysis

3.1 Device 1: Control Device

The control device takes user input and can give a run an ID, or start/restart a run. The control device is also able to view a plot of the prediction and gathered run data. The main loop is shown in the code below.

```
    while(True):

        print("Start Temp to device 1 (test-dev) to start a test")
        print("Restart to any device to restart")
        #Use this file to Send Message based on User Input
        print("Send Message:")
        usrInputMsg=input()
        print("Data Type:")
        data_type=input()
        if(data_type == "Plot"):
            prediction = device.get_prediction_data()
            air_temp_data = device.get_air_temp_data()
            pyplot.plot(prediction)
            pyplot.plot(air_temp_data)
            pyplot.show()
        elif(data_type == "ID"):
            Run_ID_dict = {"Run_ID" : usrInputMsg}
            payload = json.dumps({'Type': 1, 'Data' : Run_ID_dict, 'To' : "test-dev2", 'Time
                ' : 0})
            print('Publishing payload', payload)
            client.publish(mqtt_telemetry_topic, payload, qos=1)
        elif(data_type == "Start"):
            payload = json.dumps({'Type': 1, 'Data' : "Start Temp", 'To' : "test-dev", 'Time
                ' : 0})
            print('Publishing payload', payload)
            client.publish(mqtt_telemetry_topic, payload, qos=1)
        else:
            print("To?:")
            usrInputAddr=input()
            # Report the device's temperature to the server by serializing it
            # as a JSON string.
            payload = json.dumps({'Type': int(data_type), 'Data' : usrInputMsg, 'To' :
                usrInputAddr, 'Time' : 0})
            print('Publishing payload', payload)
```

```
            client.publish(mqtt_telemetry_topic, payload, qos=1)
```

3.2 Device 2: Virtual Machine Device

The virtual machine saves air temp data and predictions using the Google Cloud with the following functions shown below. The code snippet below shows the main loop which reads which state its in and performs actions based on it, including returning air and part temperatures, and saving the prediction it made.

```
    while(True):
        time.sleep(.1)
        if(device.read_predict_state()): #Got config update to turn lamp on.
            start_time = time.time()
            #done_payload = payload = json.dumps({"Type": 1, "Data" : "Thanks for Data", "
                Time" : str(time.time() - start_time), "To" : "test-dev"})
            #device.write_data_to_np()
            #client.publish(mqtt_telemetry_topic, done_payload, qos=1)
            stop_time, prediction = device.make_prediction()
            with open("yhat_vm.txt", "w") as prediction_file:
                for line in prediction:
                    prediction_file.write(str(line) + "\n")
                    print(line)
                prediction_file.close()
            stop_time = "Stop Time" + str(stop_time)
            print(stop_time)
            stop_payload = json.dumps({"Type": 1, "Data" : stop_time, "Time" : str(time.time
                () - start_time)[:4], "To" : "test-dev"})
            client.publish(mqtt_telemetry_topic, stop_payload, qos=1)
            device.stop_predict()
        if(device.read_data_state()):
            device.stop_save_data_state()
            done_payload = json.dumps({"Type": 1, "Data" : "Thanks for Data", "Time" : str(
                time.time() - start_time), "To" : "test-dev"})
            client.publish(mqtt_telemetry_topic, done_payload, qos=1)
            print("Return Prediction to User")
            payload = json.dumps({"Type": 0, "Data" : prediction.tolist(), "Time" : str(time
                .time() - start_time)[:4], 'To' : "test-dev3"})
            client.publish(mqtt_telemetry_topic, payload, qos=1)
            air_temp = device.read_air_temp()
            time.sleep(1)
            payload = json.dumps({"Type": 1, "Data" : air_temp.tolist(), "Time" : str(time.
                time() - start_time)[:4], 'To' : "test-dev3"})
            client.publish(mqtt_telemetry_topic, payload, qos=1)
            print("Waiting for data to Store")
            storage_client = storage.Client(credentials=credentials, project='turnkey-banner
                -26a5721')
            bucket = storage_client.get_bucket('iot_bucket_453')
            blob = bucket.blob('yhat_vm_file_{}'.format(device.get_Run_ID()))
            blob.upload_from_filename("yhat_vm.txt")
            print("Saved File")
        if(device.read_reset()):
```

```
            print("Reset")
            payload = json.dumps({"Type": 1, "Data" : "Wait State", "Time" : str(time.time()
                 - start_time), "To" : "test-dev2"})
            client.publish(mqtt_telemetry_topic, payload, qos=1)
            payload = json.dumps({"Type": 1, "Data" : "Reset Done 2", "Time" : str(time.time
                () - start_time), "To" : "test-dev3"})
            client.publish(mqtt_telemetry_topic, payload, qos=1)
            device.reset_false()
    client.disconnect()
```

The code below shows how the virtual machine code loads a model file and the data it has collected to make a prediction in Keras.

```
        model = load_model("model_weights_data2_lstm.h5")

        #val_scaled = self.temps
        val_data_bak = self.temps
        val_scaled = np.zeros((360,1,2))
        #Scale data that change during run this way
        for j in range(val_scaled.shape[0]):
            scalers[0] = MinMaxScaler(feature_range=(0, 1))
            scalers[1] = MinMaxScaler(feature_range=(0, 1))
            val_scaled[:,0, 0:1] = scalers[0].fit_transform(val_data_bak[0,:,1:2])
            val_scaled[:,0, 1:2] = scalers[1].fit_transform(val_data_bak[0,:,0:1])
        #print(val_scaled[0:50,:,:])
        print(val_scaled[:,0:1,:].shape)
        yhat = model.predict(val_scaled[:,0:1,:] , batch_size = local_batch_size)
        val_scaler = MinMaxScaler(feature_range=(0,1)).fit(self.temps[0,:,0:1])
        inv_yhat_out = val_scaler.inverse_transform(yhat)
        print(yhat)
        print(inv_yhat_out)
        stop_time = 360
        for i in range(0, len(inv_yhat_out)):
            if(inv_yhat_out[i] > 98):
                stop_time = self.temps[0,i,1]
                break
                print("Stop Time is: "+str(stop_time))
        return stop_time, inv_yhat_out
```

3.3 Device 3: Temp logger and Lamp control device

The code below shows how the temp logger reads from a file (for lack of actual thermistors) and sends this data in batches to the virtual machine.

```
                for i in range(0,len(lines),10):
                    if(device.read_reset()):
                        break
                    line = lines[i]
                    line_data = line[:25]
                    line_data_split = line_data.split(",")
```

```
                    index_data += [index]
                    payload_data_new = {index : {"Temp" : float(line_data_split[1]), "Time"
                        : float(line_data_split[0])}, "Index" : index_data}
                    if(last_temp == float(line_data_split[1]) and start_predict_flag == 0):
                        #temp is flat
                        start_predict_flag = 1
                        lamp_off = json.dumps({"Type": 1, "Data" :"Lamp is off", "Time" :
                            str(time.time() - start_time)[:4], "To" : "test-dev2"})
                        client.publish(mqtt_telemetry_topic, lamp_off, qos=1)
                    last_temp = float(line_data_split[1])
                    print("Time: "+str(line_data_split[0])+", Temperature: "+str(
                        line_data_split[1]))
                    if(device.read_stop_time() <= float(line_data_split[0])):
                        device.stop_temp()
                        if(device.read_stop_time == float(line_data_split[0])):
                            print("Stopped at: "+str(line_data_split[0]))
                        #break
                    payload_data.update(payload_data_new)
                    index += 1
                    if(index % 12 == 0):
                        payload = json.dumps({"Type": 0, "Data" : payload_data, "Time" : str
                            (time.time() - start_time)[:4], 'To' : "test-dev2"})
                        client.publish(mqtt_telemetry_topic, payload, qos=1)
                        write_file.write(str(line)+"\n")
                        payload_data = {}
                        index_data = []
                        output_string = json.loads(payload)
                        if(start_predict_flag == 1):
                            time.sleep(1)
                            predict_payload = json.dumps({"Type": 1, "Data" :"Predict", "
                                Time" : str(time.time() - start_time)[:4], "To" : "test-dev2
                                "})
                            client.publish(mqtt_telemetry_topic, predict_payload, qos=1)
```

3.4 Cloud Function

The cloud function acts as the server and routes data to different devices. echo_pubsub1 is executed whenever a message is published to the topic by any of the devices.

```
def send_message(project_id, cloud_region, registry_id, device_id, message):
    client = iot_v1.DeviceManagerClient()
    device_path = client.device_path(
    project_id, cloud_region, registry_id, device_id)
    data = message.encode('utf-8')
    version = 0 #default version
    return client.send_command_to_device(device_path, data, version)

def echo_pubsub1(event,context):
    #Function Executes when topic is updated
    message_event = event['data']
```

```
    message = base64.b64decode(message_event).decode('utf-8')
    deviceRegistryLocation= event['attributes']['deviceRegistryLocation']
    deviceRegistryId= event['attributes']['deviceRegistryId']
    projectId = event['attributes']['projectId']
    payload = json.loads(message)
    #gather info and send message to device
    if(payload['Type'] == 1): #data
        recipient = payload['To']
        my_message = message
        send_message(projectId,deviceRegistryLocation,deviceRegistryId,recipient,str(
            my_message))
```

4 Example of Performance

Video of example: Video Demonstration. The video was cut before showing the saved prediction and collected data which is saved to the cloud storage bucket as a .CSV at the end of every run, but it can be seen that it was successful in the figure below.

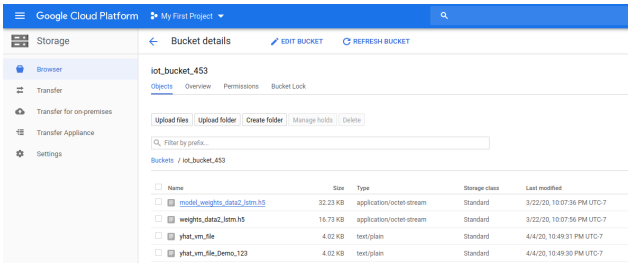

Figure 5: Image showing saved prediction

link to github: github

5 Conclusion

In this project, we were able to make use of the Google Cloud suite of cloud computing resources and services to implement a machine learning algorithm for a composite curing process. The goal of the project was accomplished by using MQTT communication between a Raspberry Pi for temperature measurements, a control PC, and a Google Cloud virtual machine for making a prediction for the part using an LSTM neural network. Ultimately, we were able to implement a data processing system with machine learning and an IoT network for composite curing.

6 Appendix

Code

Github Link

link to github: github

Control Device

```
# Copyright 2017 Google Inc. All rights reserved.
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

#Code edited from end to end example Github link below
#https://github.com/GoogleCloudPlatform/python-docs-samples.git

"""
To connect the device you must have downloaded Google's CA root certificates,
and a copy of your private key file. See cloud.google.com/iot for instructions
on how to do this. Run this script with the corresponding algorithm flag.

  $ python cloudiot_pubsub_example_mqtt_device.py \
      --project_id=my-project-id \
      --registry_id=example-my-registry-id \
      --device_id=my-device-id \
      --private_key_file=rsa_private.pem \
      --algorithm=RS256
"""

import argparse
import datetime
import json
import os
import ssl
import time
import base64
import jwt
import paho.mqtt.client as mqtt
from gcloud import storage
```

```
from oauth2client.service_account import ServiceAccountCredentials
import argparse
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pyplot

def upload_blob(bucket_name, blob_text, destination_blob_name):
    """Uploads a file to the bucket."""
    storage_client = storage.Client()
    bucket = storage_client.get_bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    blob.upload_from_string(blob_text)

    print('File {} uploaded to {}.'.format(
        source_file_name,
        destination_blob_name))

def log_data(request):
    request_json = request.get_json()
    BUCKET_NAME = 'iot_bucket_453'
    BLOB_NAME = 'test-blob'
    BLOB_STR = '{"blob": "some json"}'

    upload_blob(BUCKET_NAME, BLOB_STR, BLOB_NAME)
    return f'Success!'

def create_jwt(project_id, private_key_file, algorithm):
    """Create a JWT (https://jwt.io) to establish an MQTT connection."""
    token = {
        'iat': datetime.datetime.utcnow(),
        'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=60),
        'aud': project_id
    }
    with open(private_key_file, 'r') as f:
        private_key = f.read()
    print('Creating JWT using {} from private key file {}'.format(
        algorithm, private_key_file))
    return jwt.encode(token, private_key, algorithm=algorithm)


def error_str(rc):
    """Convert a Paho error to a human readable string."""
    return '{}: {}'.format(rc, mqtt.error_string(rc))


class Device(object):
    """Represents the state of a single device."""

    def __init__(self):
        self.connected = False
        self.prediction = np.zeros(360)
```

```
92          self.air_temp = np.zeros(360)
93      def wait_for_connection(self, timeout):
94          """Wait for the device to become connected."""
95          total_time = 0
96          while not self.connected and total_time < timeout:
97              time.sleep(1)
98              total_time += 1
99
100         if not self.connected:
101             raise RuntimeError('Could not connect to MQTT bridge.')
102     def get_prediction_data(self):
103         return self.prediction
104     def get_air_temp_data(self):
105         return self.air_temp
106     def on_connect(self, unused_client, unused_userdata, unused_flags, rc):
107         """Callback for when a device connects."""
108         print('Connection Result:', error_str(rc))
109         self.connected = True
110
111     def on_disconnect(self, unused_client, unused_userdata, rc):
112         """Callback for when a device disconnects."""
113         print('Disconnected:', error_str(rc))
114         self.connected = False
115
116     def on_publish(self, unused_client, unused_userdata, unused_mid):
117         """Callback when the device receives a PUBACK from the MQTT bridge."""
118         print('Published message acked.')
119
120     def on_subscribe(self, unused_client, unused_userdata, unused_mid,
121                      granted_qos):
122         """Callback when the device receives a SUBACK from the MQTT bridge."""
123         print('Subscribed: ', granted_qos)
124         if granted_qos[0] == 128:
125             print('Subscription failed.')
126
127     def on_message(self, unused_client, unused_userdata, message):
128         """Callback when the device receives a message on a subscription."""
129         payload = message.payload.decode('utf-8')
130         payload_dict = json.loads(payload)
131
132         print(payload)
133         if(payload_dict["Type"] == 0): #this will be prediction data
134             print(payload_dict["Data"])
135             self.prediction = payload_dict["Data"]
136             print(self.prediction)
137         if(payload_dict["Type"] == 1):
138             self.air_temp   = payload_dict["Data"]
139         if not payload:
140             return
141
142 def parse_command_line_args():
143     """Parse command line arguments."""
```

```
    parser = argparse.ArgumentParser(
        description='Example Google Cloud IoT MQTT device connection code.')
    parser.add_argument(
        '--project_id',
        default=os.environ.get("GOOGLE_CLOUD_PROJECT"),
        required=True,
        help='GCP cloud project name.')
    parser.add_argument(
        '--registry_id', required=True, help='Cloud IoT registry id')
    parser.add_argument(
        '--device_id',
        required=True,
        help='Cloud IoT device id')
    parser.add_argument(
        '--private_key_file', required=True, help='Path to private key file.')
    parser.add_argument(
        '--algorithm',
        choices=('RS256', 'ES256'),
        required=True,
        help='Which encryption algorithm to use to generate the JWT.')
    parser.add_argument(
        '--cloud_region', default='us-central1', help='GCP cloud region')
    parser.add_argument(
        '--ca_certs',
        default='roots.pem',
        help='CA root certificate. Get from https://pki.google.com/roots.pem')
    parser.add_argument(
        '--num_messages',
        type=int,
        default=100,
        help='Number of messages to publish.')
    parser.add_argument(
        '--mqtt_bridge_hostname',
        default='mqtt.googleapis.com',
        help='MQTT bridge hostname.')
    parser.add_argument(
        '--mqtt_bridge_port', type=int, default=8883, help='MQTT bridge port.')
    parser.add_argument(
        '--message_type', choices=('event', 'state'),
        default='event',
        help=('Indicates whether the message to be published is a '
              'telemetry event or a device state message.'))
    parser.add_argument('-i', '--in_file', help='Input file for Network weights',required=
        False)

    return parser.parse_args()
def dir_path(string):
    string2 = os.getcwd() + string
    if os.path.isdir(string2):
        return string2
    elif os.path.isdir(string):
        return string
```

```
195     else:
196         raise NotADirectoryError(string)
197
198 def main():
199     args = parse_command_line_args()
200     # Create the MQTT client and connect to Cloud IoT.
201     client = mqtt.Client(
202         client_id='projects/{}/locations/{}/registries/{}/devices/{}'.format(
203             args.project_id,
204             args.cloud_region,
205             args.registry_id,
206             args.device_id))
207     client.username_pw_set(
208         username='unused',
209         password=create_jwt(
210             args.project_id,
211             args.private_key_file,
212             args.algorithm))
213     client.tls_set(ca_certs=args.ca_certs, tls_version=ssl.PROTOCOL_TLSv1_2)
214
215     device = Device()
216
217     client.on_connect = device.on_connect
218     client.on_publish = device.on_publish
219     client.on_disconnect = device.on_disconnect
220     client.on_subscribe = device.on_subscribe
221     client.on_message = device.on_message
222
223     client.connect(args.mqtt_bridge_hostname, args.mqtt_bridge_port)
224
225     client.loop_start()
226
227     # This is the topic that the device will publish telemetry events
228     # (temperature data) to.
229     mqtt_telemetry_topic = '/devices/{}/events'.format(args.device_id)
230
231     # This is the topic that the device will receive configuration updates on.
232     mqtt_config_topic = '/devices/{}/commands/#'.format(args.device_id)
233
234     # Wait up to 5 seconds for the device to connect.
235     device.wait_for_connection(5)
236
237     # Subscribe to the config topic.
238     client.subscribe(mqtt_config_topic, qos=1)
239     # Update and publish temperature readings at a rate of one per second.
240     time.sleep(3)
241     while(True):
242
243         print("Start Temp to device 1 (test-dev) to start a test")
244         print("Restart to any device to restart")
245         #Use this file to Send Message based on User Input
246         print("Send Message:")
```

```
247         usrInputMsg=input()
248         print("Data Type:")
249         data_type=input()
250         if(data_type == "Plot"):
251             prediction = device.get_prediction_data()
252             air_temp_data = device.get_air_temp_data()
253             pyplot.plot(prediction)
254             pyplot.plot(air_temp_data)
255             pyplot.show()
256         elif(data_type == "ID"):
257             Run_ID_dict = {"Run_ID" : usrInputMsg}
258             payload = json.dumps({'Type': 1, 'Data' : Run_ID_dict, 'To' : "test-dev2", 'Time
                    ' : 0})
259             print('Publishing payload', payload)
260             client.publish(mqtt_telemetry_topic, payload, qos=1)
261         elif(data_type == "Start"):
262             payload = json.dumps({'Type': 1, 'Data' : "Start Temp", 'To' : "test-dev", 'Time
                    ' : 0})
263             print('Publishing payload', payload)
264             client.publish(mqtt_telemetry_topic, payload, qos=1)
265         else:
266             print("To?:")
267             usrInputAddr=input()
268             # Report the device's temperature to the server by serializing it
269             # as a JSON string.
270             payload = json.dumps({'Type': int(data_type), 'Data' : usrInputMsg, 'To' :
                    usrInputAddr, 'Time' : 0})
271             print('Publishing payload', payload)
272             client.publish(mqtt_telemetry_topic, payload, qos=1)
273
274
275     client.disconnect()
276     client.loop_stop()
277     print('Finished loop successfully. Goodbye!')
278
279
280 if __name__ == '__main__':
281     main()
```

VM Device

```
1  # Copyright 2017 Google Inc. All rights reserved.
2  # Licensed under the Apache License, Version 2.0 (the "License");
3  # you may not use this file except in compliance with the License.
4  # You may obtain a copy of the License at
5  #
6  #    http://www.apache.org/licenses/LICENSE-2.0
7  #
8  # Unless required by applicable law or agreed to in writing, software
9  # distributed under the License is distributed on an "AS IS" BASIS,
10 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
11 # See the License for the specific language governing permissions and
```

```
# limitations under the License.

#Code edited from end to end example Github link below
#https://github.com/GoogleCloudPlatform/python-docs-samples.git

"""
To connect the device you must have downloaded Google's CA root certificates,
and a copy of your private key file. See cloud.google.com/iot for instructions
on how to do this. Run this script with the corresponding algorithm flag.

  $ python cloudiot_pubsub_example_mqtt_device.py \
      --project_id=my-project-id \
      --registry_id=example-my-registry-id \
      --device_id=my-device-id \
      --private_key_file=rsa_private.pem \
      --algorithm=RS256
"""
import argparse
import datetime
import json
import os
import ssl
import time
import base64
import jwt
import paho.mqtt.client as mqtt

from gcloud import storage
from oauth2client.service_account import ServiceAccountCredentials

import argparse
import numpy as np
from keras.models import load_model
from sklearn.preprocessing import MinMaxScaler

def upload_blob(bucket_name, blob_text, destination_blob_name):
    """Uploads a file to the bucket."""
    storage_client = storage.Client()
    bucket = storage_client.get_bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    blob.upload_from_string(blob_text)

    print('File {} uploaded to {}.'.format(
        source_file_name,
        destination_blob_name))

def log_data(request):
    request_json = request.get_json()
    BUCKET_NAME = 'iot_bucket_453'
    BLOB_NAME = 'test-blob'
    BLOB_STR = '{"blob": "some json"}'
```

```
64
65      upload_blob(BUCKET_NAME, BLOB_STR, BLOB_NAME)
66      return
67
68  def download_blob(bucket_name, source_blob_name, destination_file_name):
69      """Downloads a blob from the bucket."""
70      # bucket_name = "your-bucket-name"
71      # source_blob_name = "storage-object-name"
72      # destination_file_name = "local/path/to/file"
73
74      storage_client = storage.Client()
75
76      bucket = storage_client.bucket(bucket_name)
77      blob = bucket.blob(source_blob_name)
78      blob.download_to_filename(destination_file_name)
79
80      print(
81          "Blob {} downloaded to {}.".format(
82              source_blob_name, destination_file_name
83          )
84      )
85
86
87  def create_jwt(project_id, private_key_file, algorithm):
88      """Create a JWT (https://jwt.io) to establish an MQTT connection."""
89      token = {
90          'iat': datetime.datetime.utcnow(),
91          'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=60),
92          'aud': project_id
93      }
94      with open(private_key_file, 'r') as f:
95          private_key = f.read()
96      print('Creating JWT using {} from private key file {}'.format(
97          algorithm, private_key_file))
98      return jwt.encode(token, private_key, algorithm=algorithm)
99
100
101 def error_str(rc):
102     """Convert a Paho error to a human readable string."""
103     return '{}: {}'.format(rc, mqtt.error_string(rc))
104
105
106 class Device(object):
107     """Represents the state of a single device."""
108
109     def __init__(self):
110         self.connected = False
111         self.start_predict = False
112         self.save_data = False
113         self.reset = False
114         self.temps = np.zeros((1,360,2)) #two features air and out temp
115         self.Run_ID = ""
```

```
116     def log_air_temp(self,index,air,time):
117         self.temps[0,index,0] = air
118         self.temps[0,index,1] = time
119     def read_air_temp(self):
120         return self.temps[0,:,0]
121     def reset_false(self):
122         self.reset = False
123     def read_reset(self):
124         return self.reset
125     def start_predict(self):
126         self.start_predict = True
127         print("Start Predict On")
128     def stop_predict(self):
129         self.start_predict = False
130         print("Stop Predict")
131     def read_predict_state(self):
132         return self.start_predict
133     def read_data_state(self):
134         return self.save_data
135     def stop_save_data_state(self):
136         self.save_data = False
137     def get_Run_ID(self):
138         return self.Run_ID
139     # def write_data_to_np(self):
140     #     for i in range(0, len(self.store_data[:])):
141     #         for j in range(0, len(sself.tore_data[0][:]),2):
142     #             self.temps[0,len(self.store_data[:])*(i) + int(j/2),0] = self.store_data[i
        ][j]
143     #             self.temps[0,len(self.store_data[:])*(i) + int(j/2),1] = self.store_data[i
        ][j+1]
144     #     self.store_data = [[]]
145     #     self.time_data = []
146     def make_prediction(self):
147         #in_file_name = parsed_args.in_file
148         scalers = {}
149         local_batch_size = 18
150         for i in range(len(self.temps[0,:,0])-1,-1,-1):
151             print(self.temps[0,i,0])
152             index = i
153             if(self.temps[0,i,0] != 0):
154                 temp = self.temps[0,i,0]
155                 break
156         for i in range(index,len(self.temps[0,:,0])):
157             self.temps[0,i,0] = temp
158             self.temps[0,i,1] = i * 10
159 
160         print("temps")
161         print(self.temps[0,:,0])
162         print("times")
163         print(self.temps[0,:,1])
164 
165         # bucket_name = "iot_bucket_453"
```

```
166         # download_blob(bucket_name, "model_weights_data2_lstm.h5", "
                model_weights_data2_lstm.h5")
167
168         model = load_model("model_weights_data2_lstm.h5")
169
170         #val_scaled = self.temps
171         val_data_bak = self.temps
172         val_scaled = np.zeros((360,1,2))
173         #Scale data that change during run this way
174         for j in range(val_scaled.shape[0]):
175             scalers[0] = MinMaxScaler(feature_range=(0, 1))
176             scalers[1] = MinMaxScaler(feature_range=(0, 1))
177             val_scaled[:,0, 0:1] = scalers[0].fit_transform(val_data_bak[0,:,1:2])
178             val_scaled[:,0, 1:2] = scalers[1].fit_transform(val_data_bak[0,:,0:1])
179         #print(val_scaled[0:50,:,:])
180         print(val_scaled[:,0:1,:].shape)
181         yhat = model.predict(val_scaled[:,0:1,:] , batch_size = local_batch_size)
182         val_scaler = MinMaxScaler(feature_range=(0,1)).fit(self.temps[0,:,0:1])
183         inv_yhat_out = val_scaler.inverse_transform(yhat)
184         print(yhat)
185         print(inv_yhat_out)
186         stop_time = 360
187         for i in range(0, len(inv_yhat_out)):
188             if(inv_yhat_out[i] > 98):
189                 stop_time = self.temps[0,i,1]
190                 break
191                 print("Stop Time is: "+str(stop_time))
192         return stop_time, inv_yhat_out
193
194     def wait_for_connection(self, timeout):
195         """Wait for the device to become connected."""
196         total_time = 0
197         while not self.connected and total_time < timeout:
198             time.sleep(1)
199             total_time += 1
200
201         if not self.connected:
202             raise RuntimeError('Could not connect to MQTT bridge.')
203
204     def on_connect(self, unused_client, unused_userdata, unused_flags, rc):
205         """Callback for when a device connects."""
206         print('Connection Result:', error_str(rc))
207         self.connected = True
208
209     def on_disconnect(self, unused_client, unused_userdata, rc):
210         """Callback for when a device disconnects."""
211         print('Disconnected:', error_str(rc))
212         self.connected = False
213
214     def on_publish(self, unused_client, unused_userdata, unused_mid):
215         """Callback when the device receives a PUBACK from the MQTT bridge."""
216         print('Published message acked.')
```

```
217
218     def on_subscribe(self, unused_client, unused_userdata, unused_mid,
219                      granted_qos):
220         """Callback when the device receives a SUBACK from the MQTT bridge."""
221         print('Subscribed: ', granted_qos)
222         if granted_qos[0] == 128:
223             print('Subscription failed.')
224
225     def on_message(self, unused_client, unused_userdata, message):
226         index = 0
227         """Callback when the device receives a message on a subscription."""
228         payload = message.payload.decode('utf-8')
229         payload_dict = json.loads(payload)
230         if(payload_dict["Type"] == 0): #It's Data
231             for i in payload_dict["Data"]["Index"]:
232                 print(payload_dict["Data"][str(i)]["Temp"])
233                 self.temps[0,i,0] = float(payload_dict["Data"][str(i)]["Temp"])
234                 self.temps[0,i,1] = float(payload_dict["Data"][str(i)]["Time"])
235
236         if(payload_dict["Type"] == 1): #It's Configuration Update
237             if(payload_dict["Data"] == "Predict"):
238                 self.start_predict = True
239                 print("Starting Prediction")
240             if(payload_dict["Data"] == "Done"):
241                 self.save_data = True
242                 print("Time to save Data")
243             if(payload_dict["Data"] == "Reset"):
244                 self.start_predict = False
245                 self.save_data = False
246                 self.temps = np.zeros((1,360,2)) #two features air and out temp
247                 self.reset = True
248             if( "Run_ID"  in payload_dict["Data"]):
249                 self.Run_ID = payload_dict["Data"]["Run_ID"]
250                 print("ID set to {}".format(self.Run_ID))
251         return
252
253 def parse_command_line_args():
254     """Parse command line arguments."""
255     parser = argparse.ArgumentParser(
256         description='Example Google Cloud IoT MQTT device connection code.')
257     parser.add_argument(
258         '--project_id',
259         default=os.environ.get("GOOGLE_CLOUD_PROJECT"),
260         required=True,
261         help='GCP cloud project name.')
262     parser.add_argument(
263         '--registry_id', required=True, help='Cloud IoT registry id')
264     parser.add_argument(
265         '--device_id',
266         required=True,
267         help='Cloud IoT device id')
268     parser.add_argument(
```

```
269         '--private_key_file', required=True, help='Path to private key file.')
270     parser.add_argument(
271         '--algorithm',
272         choices=('RS256', 'ES256'),
273         required=True,
274         help='Which encryption algorithm to use to generate the JWT.')
275     parser.add_argument(
276         '--cloud_region', default='us-central1', help='GCP cloud region')
277     parser.add_argument(
278         '--ca_certs',
279         default='roots.pem',
280         help='CA root certificate. Get from https://pki.google.com/roots.pem')
281     parser.add_argument(
282         '--num_messages',
283         type=int,
284         default=100,
285         help='Number of messages to publish.')
286     parser.add_argument(
287         '--mqtt_bridge_hostname',
288         default='mqtt.googleapis.com',
289         help='MQTT bridge hostname.')
290     parser.add_argument(
291         '--mqtt_bridge_port', type=int, default=8883, help='MQTT bridge port.')
292     parser.add_argument(
293         '--message_type', choices=('event', 'state'),
294         default='event',
295         help=('Indicates whether the message to be published is a '
296               'telemetry event or a device state message.'))
297     parser.add_argument('-i', '--in_file', help='Input file for Network weights',required=
        False)
298
299     return parser.parse_args()
300 def dir_path(string):
301     string2 = os.getcwd() + string
302     if os.path.isdir(string2):
303         return string2
304     elif os.path.isdir(string):
305         return string
306     else:
307         raise NotADirectoryError(string)
308
309 def main():
310     args = parse_command_line_args()
311     with open('turnkey-banner-265721-da1327341af6.json', 'r') as json_file:
312         data = json.load(json_file)
313     credentials = ServiceAccountCredentials.from_json_keyfile_dict(
314         data
315     )
316
317     data_log = 'run0.txt'
318     read_log = 'data_2025100.csv'
319     # Create the MQTT client and connect to Cloud IoT.
```

```
320     client = mqtt.Client(
321         client_id='projects/{}/locations/{}/registries/{}/devices/{}'.format(
322             args.project_id,
323             args.cloud_region,
324             args.registry_id,
325             args.device_id))
326     client.username_pw_set(
327         username='unused',
328         password=create_jwt(
329             args.project_id,
330             args.private_key_file,
331             args.algorithm))
332     client.tls_set(ca_certs=args.ca_certs, tls_version=ssl.PROTOCOL_TLSv1_2)
333
334     device = Device()
335
336     client.on_connect = device.on_connect
337     client.on_publish = device.on_publish
338     client.on_disconnect = device.on_disconnect
339     client.on_subscribe = device.on_subscribe
340     client.on_message = device.on_message
341
342     client.connect(args.mqtt_bridge_hostname, args.mqtt_bridge_port)
343
344     client.loop_start()
345
346     # This is the topic that the device will publish telemetry events
347     # (temperature data) to.
348     mqtt_telemetry_topic = '/devices/{}/events'.format(args.device_id)
349
350     # This is the topic that the device will receive configuration updates on.
351     mqtt_config_topic = '/devices/{}/commands/#'.format(args.device_id)
352
353     # Wait up to 5 seconds for the device to connect.
354     device.wait_for_connection(5)
355
356     # Subscribe to the config topic.
357     client.subscribe(mqtt_config_topic, qos=1)
358     input_string = ""
359     print("Waiting for data to Store then predict on")
360     # Update and publish temperature readings at a rate of one per second.
361     start_time = time.time()
362     while(True):
363         time.sleep(.1)
364         if(device.read_predict_state()): #Got config update to turn lamp on.
365             start_time = time.time()
366             #done_payload = payload = json.dumps({"Type": 1, "Data" : "Thanks for Data", "
                    Time" : str(time.time() - start_time), "To" : "test-dev"})
367             #device.write_data_to_np()
368             #client.publish(mqtt_telemetry_topic, done_payload, qos=1)
369             stop_time, prediction = device.make_prediction()
370             with open("yhat_vm.txt", "w") as prediction_file:
```

```
                for line in prediction:
                    prediction_file.write(str(line) + "\n")
                    print(line)
                prediction_file.close()
            stop_time = "Stop Time" + str(stop_time)
            print(stop_time)
            stop_payload = json.dumps({"Type": 1, "Data" : stop_time, "Time" : str(time.time
                () - start_time)[:4], "To" : "test-dev"})
            client.publish(mqtt_telemetry_topic, stop_payload, qos=1)
            device.stop_predict()
        if(device.read_data_state()):
            device.stop_save_data_state()
            done_payload = json.dumps({"Type": 1, "Data" : "Thanks for Data", "Time" : str(
                time.time() - start_time), "To" : "test-dev"})
            client.publish(mqtt_telemetry_topic, done_payload, qos=1)
            print("Return Prediction to User")
            payload = json.dumps({"Type": 0, "Data" : prediction.tolist(), "Time" : str(time
                .time() - start_time)[:4], 'To' : "test-dev3"})
            client.publish(mqtt_telemetry_topic, payload, qos=1)
            air_temp = device.read_air_temp()
            time.sleep(1)
            payload = json.dumps({"Type": 1, "Data" : air_temp.tolist(), "Time" : str(time.
                time() - start_time)[:4], 'To' : "test-dev3"})
            client.publish(mqtt_telemetry_topic, payload, qos=1)
            print("Waiting for data to Store")
            storage_client = storage.Client(credentials=credentials, project='turnkey-banner
                -26a5721')
            bucket = storage_client.get_bucket('iot_bucket_453')
            blob = bucket.blob('yhat_vm_file_{}'.format(device.get_Run_ID()))
            blob.upload_from_filename("yhat_vm.txt")
            print("Saved File")
        if(device.read_reset()):
            print("Reset")
            payload = json.dumps({"Type": 1, "Data" : "Wait State", "Time" : str(time.time()
                 - start_time), "To" : "test-dev2"})
            client.publish(mqtt_telemetry_topic, payload, qos=1)
            payload = json.dumps({"Type": 1, "Data" : "Reset Done 2", "Time" : str(time.time
                () - start_time), "To" : "test-dev3"})
            client.publish(mqtt_telemetry_topic, payload, qos=1)
            device.reset_false()
    client.disconnect()
    client.loop_stop()
    print('Finished loop successfully. Goodbye!')


if __name__ == '__main__':
    main()
```

Logger Device

```
# Copyright 2017 Google Inc. All rights reserved.
# Licensed under the Apache License, Version 2.0 (the "License");
```

```
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

#Code edited from end to end example Github link below
#https://github.com/GoogleCloudPlatform/python-docs-samples.git

"""
To connect the device you must have downloaded Google's CA root certificates,
and a copy of your private key file. See cloud.google.com/iot for instructions
on how to do this. Run this script with the corresponding algorithm flag.

  $ python cloudiot_pubsub_example_mqtt_device.py \
      --project_id=my-project-id \
      --registry_id=example-my-registry-id \
      --device_id=my-device-id \
      --private_key_file=rsa_private.pem \
      --algorithm=RS256
"""

import argparse
import datetime
import json
import os
import ssl
import time
import base64
import jwt
import paho.mqtt.client as mqtt
from gcloud import storage
from oauth2client.service_account import ServiceAccountCredentials
import argparse
import numpy as np

def upload_blob(bucket_name, blob_text, destination_blob_name):
    """Uploads a file to the bucket."""
    storage_client = storage.Client()
    bucket = storage_client.get_bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    blob.upload_from_string(blob_text)

    print('File {} uploaded to {}.'.format(
        source_file_name,
        destination_blob_name))
```

```
55
56 def log_data(request):
57     request_json = request.get_json()
58     BUCKET_NAME = 'iot_bucket_453'
59     BLOB_NAME = 'test-blob'
60     BLOB_STR = '{"blob": "some json"}'
61
62     upload_blob(BUCKET_NAME, BLOB_STR, BLOB_NAME)
63     return f'Success!'
64
65 def create_jwt(project_id, private_key_file, algorithm):
66     """Create a JWT (https://jwt.io) to establish an MQTT connection."""
67     token = {
68         'iat': datetime.datetime.utcnow(),
69         'exp': datetime.datetime.utcnow() + datetime.timedelta(minutes=60),
70         'aud': project_id
71     }
72     with open(private_key_file, 'r') as f:
73         private_key = f.read()
74     print('Creating JWT using {} from private key file {}'.format(
75         algorithm, private_key_file))
76     return jwt.encode(token, private_key, algorithm=algorithm)
77
78
79 def error_str(rc):
80     """Convert a Paho error to a human readable string."""
81     return '{}: {}'.format(rc, mqtt.error_string(rc))
82
83
84 class Device(object):
85     """Represents the state of a single device."""
86
87     def __init__(self):
88         self.connected = False
89         self.lamp_on = False
90         self.stop_time = 3600
91         self.reset=False
92     def start_temp(self):
93         self.lamp_on = True
94         print("Lamp On")
95     def stop_temp(self):
96         self.lamp_on = False
97         print("Lamp Off")
98     def reset_false(self):
99         self.reset = False
100    def read_reset(self):
101        return self.reset
102    def read_lamp(self):
103        return self.lamp_on
104    def read_stop_time(self):
105        return self.stop_time
106    def wait_for_connection(self, timeout):
```

```
107         """Wait for the device to become connected."""
108         total_time = 0
109         while not self.connected and total_time < timeout:
110             time.sleep(1)
111             total_time += 1
112 
113         if not self.connected:
114             raise RuntimeError('Could not connect to MQTT bridge.')
115 
116     def on_connect(self, unused_client, unused_userdata, unused_flags, rc):
117         """Callback for when a device connects."""
118         print('Connection Result:', error_str(rc))
119         self.connected = True
120 
121     def on_disconnect(self, unused_client, unused_userdata, rc):
122         """Callback for when a device disconnects."""
123         print('Disconnected:', error_str(rc))
124         self.connected = False
125 
126     def on_publish(self, unused_client, unused_userdata, unused_mid):
127         """Callback when the device receives a PUBACK from the MQTT bridge."""
128         print('Published message acked.')
129 
130     def on_subscribe(self, unused_client, unused_userdata, unused_mid,
131                      granted_qos):
132         """Callback when the device receives a SUBACK from the MQTT bridge."""
133         print('Subscribed: ', granted_qos)
134         if granted_qos[0] == 128:
135             print('Subscription failed.')
136 
137     def on_message(self, unused_client, unused_userdata, message):
138         """Callback when the device receives a message on a subscription."""
139         payload = message.payload.decode('utf-8')
140         print(payload)
141         #print('Received message \'{}\' on topic \'{}\' with Qos {}'.format(
142             #base64.b64decode(message.payload), message.topic, str(message.qos)))
143 
144         # The device will receive its latest config when it subscribes to the
145         # config topic. If there is no configuration for the device, the device
146         # will receive a config with an empty payload.
147         if not payload:
148             return
149         payload = json.loads(payload)
150         if(payload["Type"] == 1): #its config update
151             if(payload["Data"] == "Start Temp"):
152                 self.lamp_on = True
153         if(payload["Type"] == 1):
154             if(payload["Data"].find("Stop Time") != -1):
155                 stop_time_str = payload["Data"].replace("Stop Time","")
156                 self.stop_time = float(stop_time_str)
157                 print("Set Stop Time: "+str(self.stop_time))
158         if(payload["Type"] == 1):
```

```
159             if(payload["Data"] == "Reset"):
160                 self.lamp_on = False
161                 self.stop_time = 3600
162                 self.reset = True
163                 print("Reset")
164 
165         # The config is passed in the payload of the message. In this example,
166         # the server sends a serialized JSON string.
167 
168 def parse_command_line_args():
169     """Parse command line arguments."""
170     parser = argparse.ArgumentParser(
171         description='Example Google Cloud IoT MQTT device connection code.')
172     parser.add_argument(
173         '--project_id',
174         default=os.environ.get("GOOGLE_CLOUD_PROJECT"),
175         required=True,
176         help='GCP cloud project name.')
177     parser.add_argument(
178         '--registry_id', required=True, help='Cloud IoT registry id')
179     parser.add_argument(
180         '--device_id',
181         required=True,
182         help='Cloud IoT device id')
183     parser.add_argument(
184         '--private_key_file', required=True, help='Path to private key file.')
185     parser.add_argument(
186         '--algorithm',
187         choices=('RS256', 'ES256'),
188         required=True,
189         help='Which encryption algorithm to use to generate the JWT.')
190     parser.add_argument(
191         '--cloud_region', default='us-central1', help='GCP cloud region')
192     parser.add_argument(
193         '--ca_certs',
194         default='roots.pem',
195         help='CA root certificate. Get from https://pki.google.com/roots.pem')
196     parser.add_argument(
197         '--num_messages',
198         type=int,
199         default=100,
200         help='Number of messages to publish.')
201     parser.add_argument(
202         '--mqtt_bridge_hostname',
203         default='mqtt.googleapis.com',
204         help='MQTT bridge hostname.')
205     parser.add_argument(
206         '--mqtt_bridge_port', type=int, default=8883, help='MQTT bridge port.')
207     parser.add_argument(
208         '--message_type', choices=('event', 'state'),
209         default='event',
210         help=('Indicates whether the message to be published is a '
```

```
211                 'telemetry event or a device state message.'))
212     parser.add_argument('-i', '--in_file', help='Input file for Network weights',required=
            False)
213
214     return parser.parse_args()
215 def dir_path(string):
216     string2 = os.getcwd() + string
217     if os.path.isdir(string2):
218         return string2
219     elif os.path.isdir(string):
220         return string
221     else:
222         raise NotADirectoryError(string)
223
224 def main():
225     args = parse_command_line_args()
226     data_log = 'run0.txt'
227     read_log = 'data_2025100.csv'
228     # Create the MQTT client and connect to Cloud IoT.
229     client = mqtt.Client(
230         client_id='projects/{}/locations/{}/registries/{}/devices/{}'.format(
231             args.project_id,
232             args.cloud_region,
233             args.registry_id,
234             args.device_id))
235     client.username_pw_set(
236         username='unused',
237         password=create_jwt(
238             args.project_id,
239             args.private_key_file,
240             args.algorithm))
241     client.tls_set(ca_certs=args.ca_certs, tls_version=ssl.PROTOCOL_TLSv1_2)
242
243     device = Device()
244
245     client.on_connect = device.on_connect
246     client.on_publish = device.on_publish
247     client.on_disconnect = device.on_disconnect
248     client.on_subscribe = device.on_subscribe
249     client.on_message = device.on_message
250
251     client.connect(args.mqtt_bridge_hostname, args.mqtt_bridge_port)
252
253     client.loop_start()
254
255     # This is the topic that the device will publish telemetry events
256     # (temperature data) to.
257     mqtt_telemetry_topic = '/devices/{}/events'.format(args.device_id)
258
259     # This is the topic that the device will receive configuration updates on.
260     mqtt_config_topic = '/devices/{}/commands/#'.format(args.device_id)
261
```

```
    # Wait up to 5 seconds for the device to connect.
    device.wait_for_connection(5)

    # Subscribe to the config topic.
    client.subscribe(mqtt_config_topic, qos=1)
    input_string = ""
    with open(read_log,"r") as read_file:
        lines = [line.rstrip() for line in read_file]
    print("Waiting To Start")
    # Update and publish temperature readings at a rate of one per second.
    start_time = time.time()

    while(True):
        last_temp = -300

        time.sleep(.1)
        if(device.read_lamp()): #Got config update to turn lamp on.
            start_predict_flag = 0
            start_time = time.time()
            done_payload = payload = json.dumps({"Type": 1, "Data" : "On", "Time" : str(time
                .time() - start_time), "To" : "test-dev2"})
            client.publish(mqtt_telemetry_topic, done_payload, qos=1)
            with open(data_log, "w") as write_file:
                start_time = time.time()
                index = 0
                time.sleep(1)
                payload_data = {}
                index_data = []
                for i in range(0,len(lines),10):
                    if(device.read_reset()):
                        break
                    line = lines[i]
                    line_data = line[:25]
                    line_data_split = line_data.split(",")
                    index_data += [index]
                    payload_data_new = {index : {"Temp" : float(line_data_split[1]), "Time"
                        : float(line_data_split[0])}, "Index" : index_data}
                    if(last_temp == float(line_data_split[1]) and start_predict_flag == 0):
                        #temp is flat
                        start_predict_flag = 1
                        lamp_off = json.dumps({"Type": 1, "Data" :"Lamp is off", "Time" :
                            str(time.time() - start_time)[:4], "To" : "test-dev2"})
                        client.publish(mqtt_telemetry_topic, lamp_off, qos=1)
                    last_temp = float(line_data_split[1])
                    print("Time: "+str(line_data_split[0])+", Temperature: "+str(
                        line_data_split[1]))
                    if(device.read_stop_time() <= float(line_data_split[0])):
                        device.stop_temp()
                        if(device.read_stop_time == float(line_data_split[0])):
                            print("Stopped at: "+str(line_data_split[0]))
                        #break
                    payload_data.update(payload_data_new)
```

```
                    index += 1
                    if(index % 12 == 0):
                        payload = json.dumps({"Type": 0, "Data" : payload_data, "Time" : str
                            (time.time() - start_time)[:4], 'To' : "test-dev2"})
                        client.publish(mqtt_telemetry_topic, payload, qos=1)
                        write_file.write(str(line)+"\n")
                        payload_data = {}
                        index_data = []
                        output_string = json.loads(payload)
                        if(start_predict_flag == 1):
                            time.sleep(1)
                            predict_payload = json.dumps({"Type": 1, "Data" :"Predict", "
                                Time" : str(time.time() - start_time)[:4], "To" : "test-dev2
                                "})
                            client.publish(mqtt_telemetry_topic, predict_payload, qos=1)
                            start_predict_flag = 2
                    time.sleep(.25)
            write_file.close()
            done_payload = json.dumps({"Type": 1, "Data" :"Done State", "Time" : str(time.
                time() - start_time)[:4], "To" : "test-dev"})
            client.publish(mqtt_telemetry_topic, done_payload, qos=1)
            done_payload = json.dumps({"Type": 1, "Data" :"Done", "Time" : str(time.time() -
                 start_time)[:4], "To" : "test-dev2"})
            client.publish(mqtt_telemetry_topic, done_payload, qos=1)
            print("Done State")
        if(device.read_reset()):
            print("Send Reset")
            payload = json.dumps({"Type": 1, "Data" : "Wait State", "Time" : str(time.time()
                 - start_time), "To" : "test-dev"})
            client.publish(mqtt_telemetry_topic, payload, qos=1)
            payload = json.dumps({"Type": 1, "Data" : "Reset Done 1", "Time" : str(time.time
                () - start_time), "To" : "test-dev3"})
            client.publish(mqtt_telemetry_topic, payload, qos=1)
            device.reset_false()

    client.disconnect()
    client.loop_stop()
    print('Finished loop successfully. Goodbye!')


if __name__ == '__main__':
    main()
```

Cloud function

```
import json
import base64

from google.cloud import iot_v1
from google.cloud import pubsub

def set_config(project_id, cloud_region, registry_id, device_id, config):
```

```
    client = iot_v1.PublisherClient()
    device_path = client.device_path(
    project_id, cloud_region, registry_id,device_id)
    data = config.encode('utf-8')
    version = 0 #default version
    return client.modify_cloud_to_device_config(device_path, data, version)

def send_message(project_id, cloud_region, registry_id, device_id, message):
    client = iot_v1.DeviceManagerClient()
    device_path = client.device_path(
    project_id, cloud_region, registry_id, device_id)
    data = message.encode('utf-8')
    version = 0 #default version
    return client.send_command_to_device(device_path, data, version)

def echo_pubsub1(event,context):
    #Function Executes when topic is updated
    message_event = event['data']
    message = base64.b64decode(message_event).decode('utf-8')
    deviceRegistryLocation= event['attributes']['deviceRegistryLocation']
    deviceRegistryId= event['attributes']['deviceRegistryId']
    projectId = event['attributes']['projectId']
    payload = json.loads(message)
    #gather info and send message to device
    if(payload['Type'] == 1): #data
        recipient = payload['To']
        my_message = message
        send_message(projectId,deviceRegistryLocation,deviceRegistryId,recipient,str(
            my_message))
```