Cloud-based Weather Sensor Lab

EE 49

December 4, 2018

1 Overview

Microcontrollers such as the Huzzah ESP32 are great for building an IoT device that collects sensor readings. However, they have limitations in terms of storage and processing capabilities. A cloud-based service such as AWS provides unlimited storage capabilities and computation power for IoT devices and systems. In this lab, you will be setting up AWS in order to publish your sensor readings, store them in a database, and display them on the web.

2 Prelab: AWS Setup

For this lab, we will be using Amazon Web Services (AWS) as our cloud service. If you have never heard of AWS before, don't worry, we'll walk you through how to set up an account in this section.

2.1 Creating an account

Register for an account at https://portal.aws.amazon.com/billing/signup. This step will require you provide credit card information. You will NOT be charged on this card following the instructions of the lab, and you can shut down your account imediatly at its conclusion. If this step poses an issue for you, please reach out to the course staff.

2.2 Region Select

At the top right corner of your screen, you should see a region selector. Set your region to **US East (Ohio)**. If you're curious about what these regions

are for, feel free to read the Region and Availability Zone documentation.

2.3 EC2

Now click on the service selector in the top left of your screen and click on **EC2** under the compute section. EC2, or **E**lastic **C**ompute **C**loud, is a service that lets us create a linux machine running in the cloud. Later in the lab, this cloud will become the host of our node-red server. To set up your EC2 instance, follow the below steps:

1. From the main EC2 page, click on the large blue **Launch Instance** button. You will be redirected to the creation portal.
2. Scroll down until you see **Ubuntu Server 18.04 LTS (HVM), SSD Volume Type**. insure that the 64-bit box is selected and then click on the blue **Select** button to the right.
3. On the next screen, select the **t2.micro** instance type.
4. Switch over to the the Configure Security Group tab. Click **Add Rule**, and set the port to 1880.
5. On the final Review step, click the **Launch** button
6. The console will prompt you to configure a set of SSH keys. Select **Create a new key pair** and click **Download key pair**. Your browser will save the .pem file - keep that safe. If you lose this file you will have to create an entire new EC2 instance.
7. Finally, click **Launch**.

2.4 Dynamo DB Setup

Returning to the service selector, click on **Dynamo DB** under the data base section. Dynamo DB is the database system we will use to store the readings taken by our sensor. In order to do this, we will have to create a new table:

1. Click on the blue **Create Table** button.
2. Set the table name to **weather_station_data** and the primary key to **hybrid_key**.
3. Click the Create Button

2.5 Check Off

Be prepared to show your GSI your created EC2 instance, the saved keys for that EC2 instance, and the Dynamo DB table you have created when you come into class.

3 Communicating With the Cloud

Now we will set up AWS IoT Core to publish sensor readings from the microcontroller. AWS IoT Core is a platform that will allow us to connect our device to AWS Services so that we can store and process our data later on. Using the service selector, select **IoT Core** under the Internet of Things section. This is your IoT Core portal, where we will be setting up our device and receive sensor readings. Feel free to look around and explore the portal.

3.1 Setting Up Your Thing

To be able to receive messages on AWS IoT, we need to register our "Thing". A "Thing" is the representation of our device on the cloud. To begin, go on the AWS IoT Core portal. In the left navigation pane, chose **Manage**, and then choose **Things** and follow these steps:

1. On the **Things** page, choose **Register a thing**. On the next page, choose **Create a single thing**.

2. On the **Add your device to the thing registry** page, enter **Station1** for the device name. Leave the default values for all the other fields, and then choose **Next**.

3. On the **Add a certificate for your thing** page, choose **Create certificate**. This generates an X.509 certificate and key pair, which you will need later on to be able connect your device to AWS IoT and publish messages.

4. On the **Certificate created!** page, download your public and private keys and your certificate. Save them in a directory on your computer. Choose **Activate** to activate the X.509 certificate, and then choose **Attach a policy**.

5. On the **Add a policy for your thing** page, choose **Register Thing**. After you register your thing, you will need to create and attach a new policy to the certificate.

6. Congrats! We have successfully registered our thing! You should be able to see your device on the Things page. Now we will attach a policy to it. In the left navigation pane, choose **Secure, Policies**. Then choose **Create a policy**.

7. On the **Create a policy page**, enter a Name for the policy. For Action, enter **iot:***. For Resource ARN, enter *. Under Effect, choose **Allow**, and then choose **Create**. You should be able to see your newly created policy on the Policies page. This policy allows your device to publish messages to AWS IoT.

8. Now we will attach this policy to our device's certificate. In the **AWS IoT console**, choose **Manage, Things**. On the Things page, choose **Station1**. In the navigation pane, choose **Security** and click on the certificate we created earlier.

9. On the certificate's **Details** page, in **Actions**, choose **Attach policy**. Choose the policy we created earlier, and then choose **Attach**.

That's it! We have successfully registered our Thing and created certificates to be able to connect and publish messages.

3.2 Setting Up Config.py

Now we will set up our Config.py file to store device configurations and certificates. To begin, open the Config.py file in your starter code. You will see some variables initialized to empty strings.

1. Our device will need to be able to connect to a network to publish sensor data to AWS IoT. In **Config.py**, set the variables **netname** and **netwpd** to your strings containing your Wifi Network Name and Password. (You can create a hotspot from your phone. Note: CalVisitor and AirBears will not work.)

2. Next, we will set **client_id** to the Thing ARN that AWS IoT provides, which is a unique string that identifies your device. To find your Thing

ARN, go to the your AWS IoT console and choose Manage, Things on the navigation pane. Choose your Thing that we created earlier, and you will see the Thing ARN, which will be in the format:

$$arn : aws : iot : us - east - 2 : XXXXXXXX : thing/Station1$$

Set **client_id** to this string.

3. Now we will get the Rest API Endpoint for our device to connect to. Go to your Thing page on the AWS IoT console once again. On the left navigation pane, choose **Interact** and look for the HTTPS section, near the top. Your end point will look like:

$$ABCDEFG1234567.iot.us - east - 2.amazonaws.com$$

Set the **server** variable in Config.py to this string.

4. Lastly, we will copy our Thing certificates that we created earlier to **Config.py**. Head over to the directory where you downloaded your certificates. Copy the contents of the private.pem.key file to the **private_key** variable, and the certificate.pem.crt file to the **cert** variable. Make sure to store them in multi-line string (triple quotes) and copy everything including "—BEGIN RSA PRIVATE KEY—" and "—END RSA PRIVATE KEY—".

3.3 Publishing MQTT Messages

Now that we have everything set up, we can publish messages to AWS IoT from our device! Open WeatherUtils.py in your starter code. In the class constructer, you need to create an instance of MQTTClient and store it in self.client. You will need to provide it with the arguments client_id, server, keepalive, port, ssl, and ssl_params.
Hint: some of the arguments have been provided for you, and the rest are in Config.py. To fetch them, you might find python's getattr() function helpful. After you create an instance of MQTT Client, call the connect() function on self.client.

To test your code, connect your microcontroller to your computer using a USB cable and load the starter files on to the microcontroller using Ampy.

(Instructions for Ampy installation and usage: https://github.com/adafruit/ampy)
Run Shell49 on your terminal and get a Repl Prompt from MicroPython.

Once you have a Repl prompt from MicroPython, reset the board. In the repl prompt, run:

from `weather_utils` import WeatherUtils
weatherUtils.publish("myTopic","Hello World!")

To see the message on AWS IoT, go to your AWS IoT Console. On the left navigation pane, choose **Test**. Under Subscribe, type myTopic in the Subscription topic box and click **Subscribe to Topic**. If you re run the code above, you should see your message "Hello World" under the topic "myTopic". Feel free to send your own messages by changing Hello World to any string and running the weatherUtils.publish method.

3.4 Publishing Sensor Data

It's time to sample sensor readings and publish them on to AWS IoT! We have done most of the work for this part. Open up **Main.py** and try to understand what it does. For this part, you have to implement the functions **get_json** and **publish_data** in WeatherUtils.
When publishing our sensor data, we want to convert our data to a JSON Object. In the get_json function, create a dictionary with the keys "station_id", "value", and "data_type" and assign them to their appropriate values. Then return the JSON representation of the Python dictionary. You might find the method ujson.dumps() useful for this part.
Next, implement publish_data to publish the input Data to the topic "Data".
Lastly, set the **READ_ENABLED** variable in WeatherUtils.py to **True** in order to enable sensor reading and publishing. That's it! If you load the files on to your board and reset the device, you will be able to see your sensor data published on the AWS IoT platform when you subscribe to the topic "Data".

3.5 Check off

To get checked off for this part, run your code on the microcontroller by pressing the reset button and show your GSI the published sensor data on

your AWS IoT portal.

4 Displaying our Sensor Reading

While reading out sensor data from the IoT Cloud portal is cool, we want a clean way to look at the trends in the data we have recieved. To acomplish this, we will host a node-red server on our EC2 instance that will interact with our MQTT client to store and display readings!

TODO: more background on node-red, how it is a drag and drop system

4.1 Setting up node-red

We will be running node-red on the EC2 instance we created in the prelab. First, we must access our instance by sshing into it.

1. Find the certificate (.pem) file that you downloaded when you created your instance and copy it into the .ssh directory with the following command:

   ```
   $ cp path/to/your/cert.pem ~/.ssh/weather\_station\_server.pem
   ```

2. Next, open your AWS console and use the service selector to open the EC2 portal.

3. In the left sidebar, click on **Instances**. You should see a single instance listed. Note that if no instances are listed, you should check your region to make sure it is set to Ohio.

4. Expand the **Public DNS (IPv4)** column and copy the value listed for your instance. It should appear as XXXX.us-east-2.compute.amazonaws.com where XXXX is a character combination, such as ec2-18-224-18-187

5. Now return to your terminal and replace address in the following command with the information you copied in the previous step. Then execute the result to ssh into your server:

   ```
   $ ssh -i "~/.ssh/weather\_station\_server.pem" ubuntu@address
   ```

Now that we're in to our instance, it's time to install node-red and configure it.

1. First, we will install node.js and node-red by executing the following commands on our instance.

```
$ curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -
$ sudo apt-get install -y nodejs build-essential
$ sudo npm install -g node-red
```

2. Then we will install node-red-dashboard, a tool that will help us create charts for that data that we collect.

```
$ cd ~/.node-red
$ npm i node-red-dashboard
```

3. To insure that your instance runs as expected, open the staff copy of settings.js and copy its contents into the settings.js file located inside of the ./node-red directory

The final step in setting up our node-red server is to make sure it can communicate with our other AWS services.

1. We will create a folder to store AWS certificates in the root directory. Similar to the certificates on our weather station, these certificates allow us to send and receive messages from the AWS IoT cloud

```
$ mkdir -p ~/.aws/certs
$ cd ~/.aws/certs
```

2. Copy the 4 certificates you used for the IoT core onto your device: The Amazon root certificate, the IoT certificate, the public key, and the private key. The names of these files are not critical, but name them such that you know which is which later on.

3. In your browser, open up the AWS control panel and use the service selector to open up **IAM**.

4. In the navigation pane of the console, choose Users.

5. Choose your IAM user name (not the check box). This will be your email

6. Choose the Security credentials tab and then choose Create access key.

7. To see the new access key, choose Show. Your credentials will look something like this:
   Access key ID: AKIAIOSFODNN7EXAMPLE
   Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Make sure to write these keys down.

8. Copy the **credentials** file from the staff files into your ~/.aws directory and replace the missing information with the keys you just generated.

With all this done, you have successfully set up your node-red server to communicate with the cloud! Now we can move on to making a simple data display.

4.2 Creating A Data Page

We'll now create flow on our node-red server that aggregates our data into the table we created and then displays that data out to the world.

1. Open the staff **flow.json** file and look at its contents, this file describes all the blocks and wires in our node-red flow. You'll see that some sections are left incomplete, fill in those sections with the appropriate identifiers and paths.

2. Now on your instance, execute the node-red command. This will launch your server.

   ```
   $ node-red
   ```

3. Remembering the IP you sshed into your instance with, use a browser to navigate to XXXX.us-east-2.compute.amazonaws.com:1880. You should be greeted with a login screen for your node-red configuration portal. Use the below credentials to log in.
   username: admin
   password: weatherstation

4. You should now see the flows page of your node-red server. In this page, you can drag and drop blocks to add elements to your server, linking the data flow between them by drawing wires block to block. In order to save you time (and a lot of tedious data parsing code) we have supplied you with the entire flow to create the display portal.

Simply click on the menu icon in the top left corner of your portal, then click on import and from there select clipboard.

5. In the box that pops up, copy in the flow.json file you modified and set the import destination to **new flow**. Click on import and you should see a flow pop up! Look around this flow and try to see if you can trace the flow of data though the system.

6. With the flow imported, click on the **Deploy** button in the top left.

With the flow deployed, our portal is now up and running! Navigate to XXXX.us-east-2.compute.amazonaws.com:1880/data to see it in action.

4.3 Check off

To checkoff this step, please show your GSI your final weather station portal, and demonstrate that the readings displayed on your portal match those taken in by your weather station. Then, explain how data flows through the node-red server by describing the path of a message sent from your weather station until it is eventually displayed. Once you have done this, you are done with the lab!

5 Post Lab: Clean Up

To make sure that you are not charged for your AWS account, you will have to delete all your running instance and databases, this will prevent you from incurring any compute time in the future. For information on how to do this, click on the following links.
Deleting EC2
Deleting DynamoDB