Assignment 3

Name – Sati, Ankit Date – 03/1/2022

Section - 001

Total in points (Maximum 100 points)–

Professors Comments –

Affirmation of Independent Effort – Ankit Sati

Question 1-

1. This questions wants us to walk through the edge computing that is being developed and take a look at it from the different perspectives listed below.
    a. NEC - Network edge compute
    b. MEC - Multi access edge compute
    c. Help of Robotics
    d. Impact on mission critical systems.
    e. Implementation and use of private networks.

Main Goal – The primary goal is to deploy everything at the end for the users so that they can reduce the time of data on the networks. We try to provide the below services at the end of the network, which makes it faster/accurate with less chances of frame drops.

- Compute Power
- Cloud space
- High Availability
- Service Space

Main Objective - Everything for the end user need to remain the same from code creation to deployment/maintenance but the end user will not know where it is being used from. It can either be on the edge of the network or the azure platforms as it has been in the past.

NEC - Network edge compute

Network Edge Compute (NEC) is the network carrier equivalent, placing the edge computing platform within their network. Instead of needing to access applications and games running in the public cloud, software providers can bring their solutions physically closer to their end-users. At AT&T's Business Summit we gave an augmented reality demonstration, working with Taqtile, and showed how to perform maintenance on an aircraft landing gear.

Prime features

- Closer to end user
- One hop communications
- Compute power at the edge.
- Deployment of services at the end.

MEC - Multi access edge compute

Through the combination of local compute resources and private mobile connectivity (private LTE), we can enable many new scenarios. For instance, in the smart factory example used earlier customers are now able to run their robotic control logic, highly available and independent of connectivity to the public cloud. MEC helps ensure that operations and any associated critical first-stage data processing remain up and production can continue uninterrupted. Advantage of near-infinite compute and storage, the cloud is ideal for large data-intensive and computational tasks, such as machine learning jobs for predictive maintenance analytics.

Prime Features

- Combination of local compute resources and private mobile connectivity
- Communication over private network.
- Complex logic like robotics can be accessed.
- **Near infinite compute storage.**

Prime advantages of EDGE COMPUTE

- Single hop computations
- Frames spend less time over the network.
- Reduce the chance of frame drops.
- Infinite compute storage at the end of network
- Complex logic can be deployed at the end of the network.
- Privatized network for mission critical projects.
- Very high speeds.
- High Availability of resources at the end.
- Cloud space with specific features as per demand of end user.

Examples of technologies used in field.

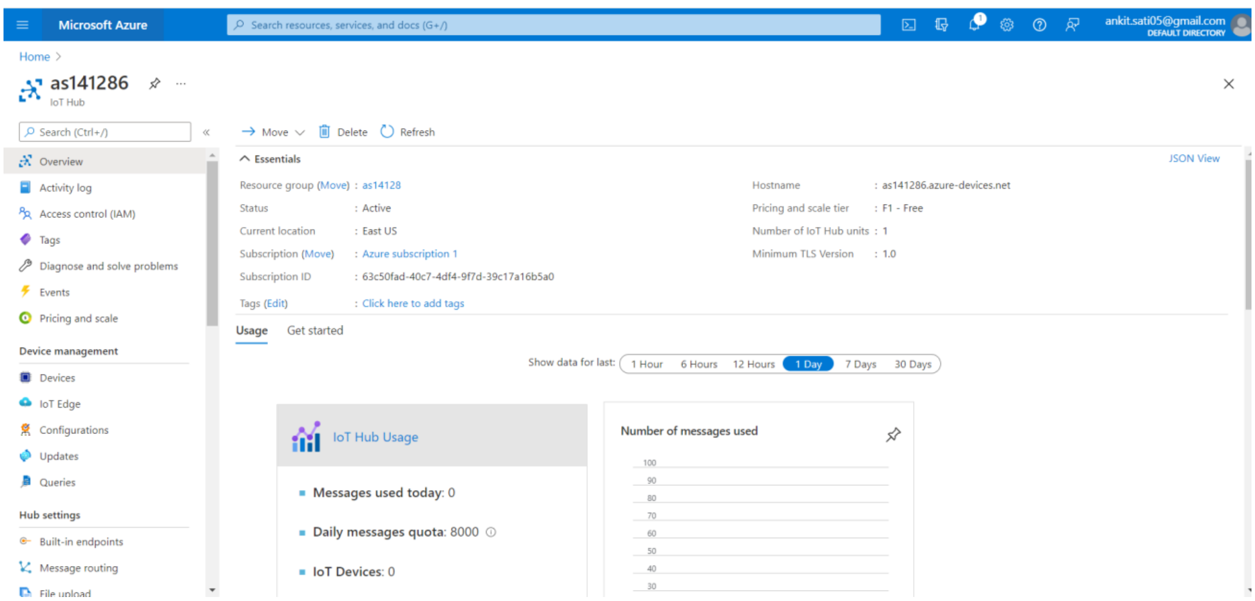
- Enterprise level – mission critical projects.
- Private LTE networks.
- Multiple cloud space
- Smart agriculture and services.
- Robotics in enterprise products.
- Product development.
- Resource deployment for projects.

Deploy your first IoT Edge module to a Windows device

Part 1 – Create your IOT HUB.

- Create an IoT Hub.
- Register an IoT Edge device to your IoT hub.
- Install and start the IoT Edge runtime on a virtual device.
- Remotely deploy a module to an IoT Edge device.

A resource group to manage all the resources you use in this quickstart. We use the example resource group name **IoTEdgeResources** throughout this quickstart.



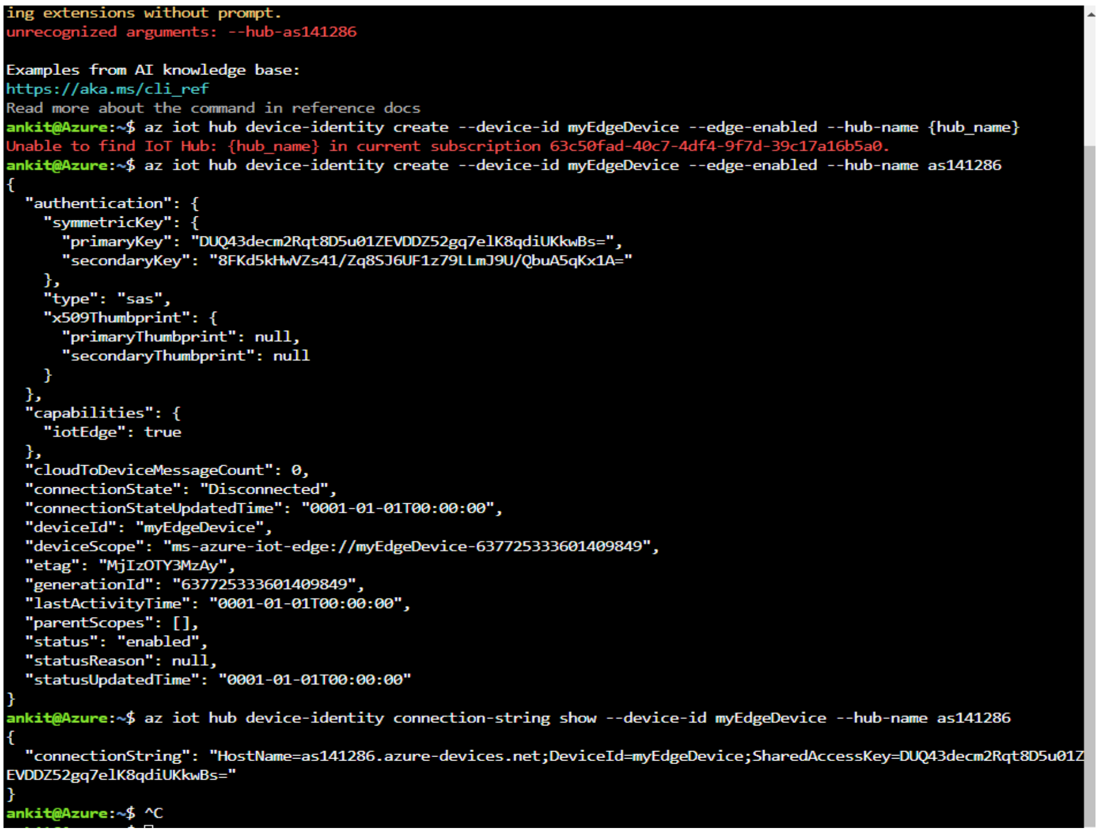
Part 2 – Register an IoT Edge device (Screenshot attached)

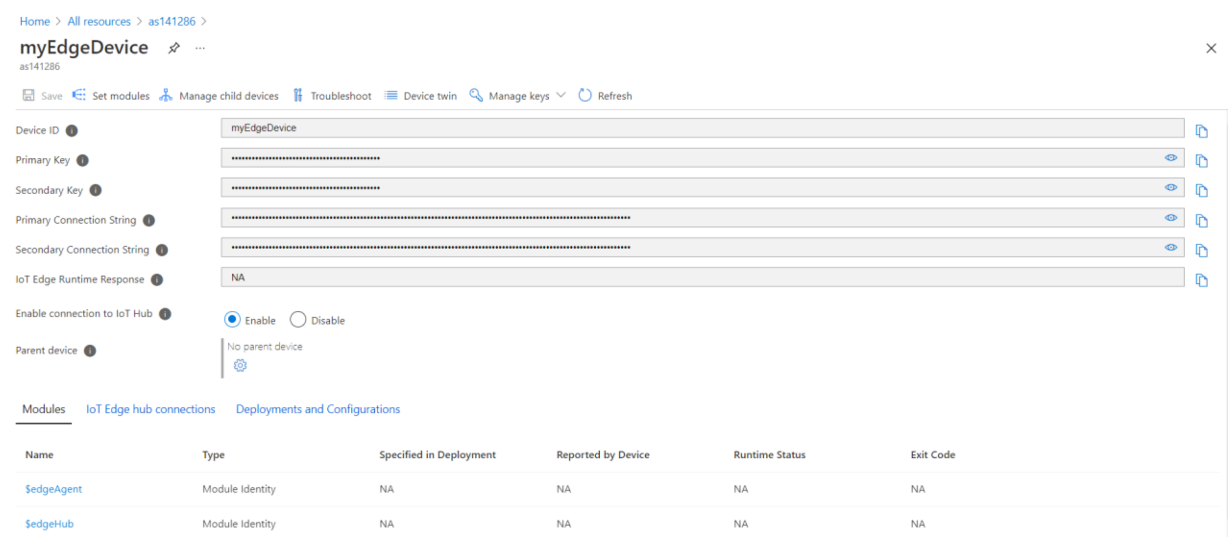
Create a device identity for your IoT Edge device so that it can communicate with your IoT hub. The device identity lives in the cloud, and you use a unique device connection string to associate a physical device to a device identity. Since IoT Edge devices behave and can be managed differently than typical IoT devices, declare this identity to be for an IoT Edge device with the --edge-enabled flag.

hub name - as141286

resource group as14128

Connection Key - HostName=as141286.azure-devices.net;DeviceId=myEdgeDevice;SharedAccessKey=DUQ43decm2Rqt8D5u01ZEVDDZ52gq7elK8qdiUKkwBs=

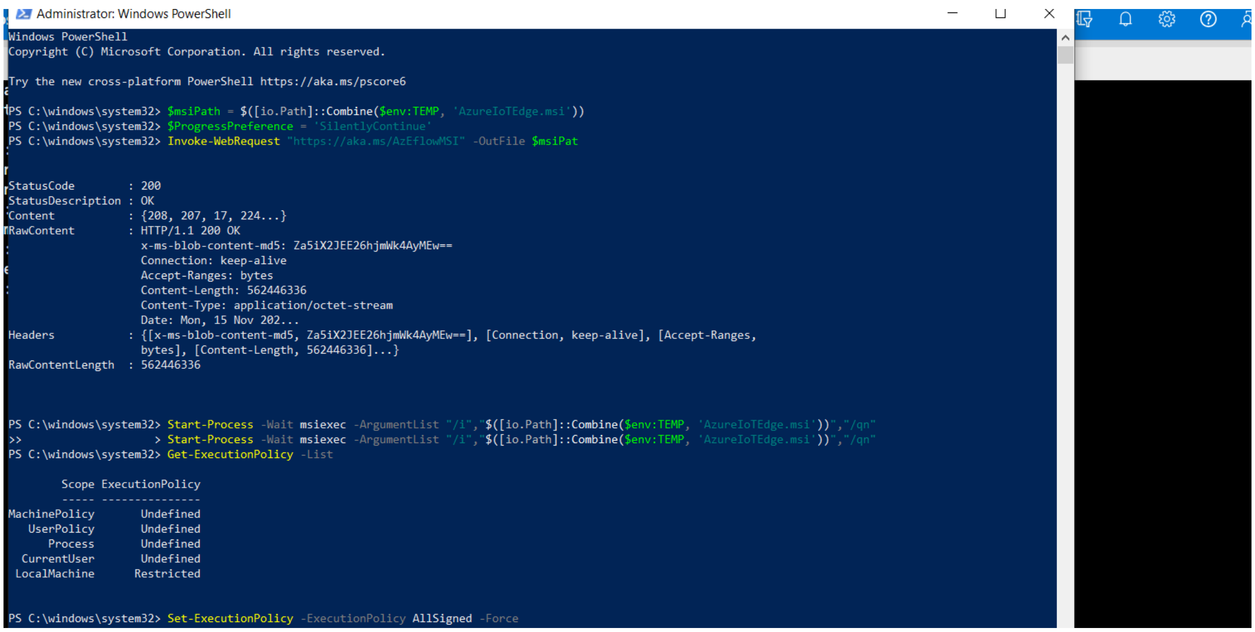




Part 3- Install and start the IoT Edge runtime

This is one of the most critical steps while deploy the edge runtime. This basically a runtime E that will help us in checking and enabling the data once it is launched in space.

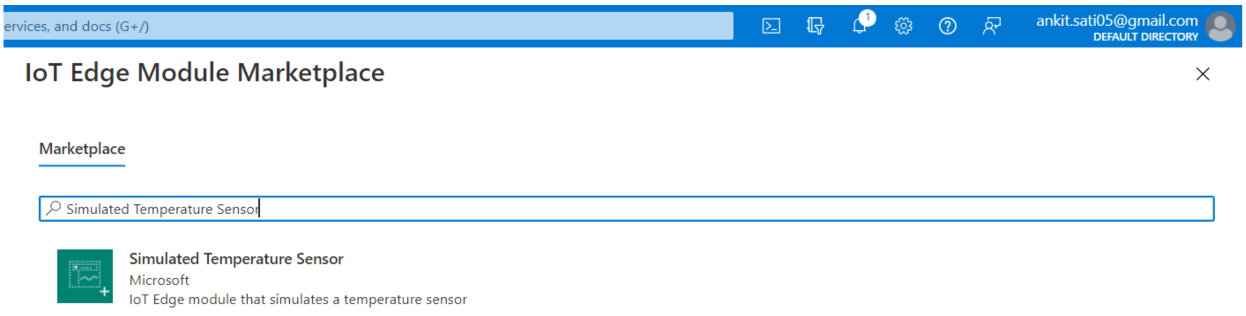
The issues that I was facing on the azure CLI were that I was not able to find and run the required hub name as it was not able to adapt the resources. Hence I soon switched to powershell to deploy my runtime.



Part 4- Deploy a module

The module that you deploy in this section simulates a sensor and sends generated data. This module is a useful piece of code when you're getting started with IoT Edge because you can use the simulated data for development and testing. If you want to see exactly what this module does, you can view the simulated temperature sensor source code.

- When you create a new IoT Edge device, it will display the status code 417 -- The device's deployment configuration is not set in the Azure portal. This status is normal, and means that the device is ready to receive a module deployment.
- In IoT Edge Module Marketplace, search for and select the Simulated Temperature Sensor module.
- The module is added to the IoT Edge Modules section with the desired running status.



Select routes

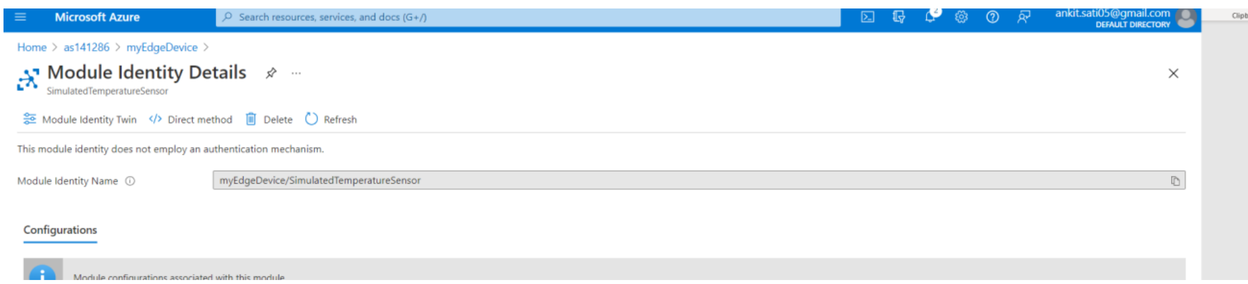
- Routes to continue to the next step of the wizard.
- On the Routes tab, remove the default route, route, and then select Next: Review + create to continue to the next step of the wizard.
- The JSON file defines all of the modules that you deploy to your IoT Edge device. You'll see the SimulatedTemperatureSensor module and the two runtime modules, edgeAgent and edgeHub.
- $edgeAgent, $edgeHub, and SimulatedTemperatureSensor. If one or more of the modules has YES under SPECIFIED IN DEPLOYMENT but not under REPORTED BY DEVICE, your IoT Edge device is still starting them. Wait a few minutes, and then refresh the page.

```
{
    "modulesContent": {
        "$edgeAgent": {
            "properties.desired": {
                "modules": {
                    "SimulatedTemperatureSensor": {
                        "settings": {
                            "image": "mcr.microsoft.com/azureiotedge-simulated-
temperature-sensor:1.0",
                            "createOptions": ""
                        },
```

```
                        "type": "docker",
                        "status": "running",
                        "restartPolicy": "always",
                        "version": "1.0"
                    }
                },
                "runtime": {
                    "settings": {
                        "minDockerVersion": "v1.25"
                    },
                    "type": "docker"
                },
                "schemaVersion": "1.1",
                "systemModules": {
                    "edgeAgent": {
                        "settings": {
                            "image": "mcr.microsoft.com/azureiotedge-agent:1.1",
                            "createOptions": ""
                        },
                        "type": "docker"
                    },
                    "edgeHub": {
                        "settings": {
                            "image": "mcr.microsoft.com/azureiotedge-hub:1.1",
                            "createOptions":
"{\"HostConfig\":{\"PortBindings\":{\"443/tcp\":[{\"HostPort\":\"443\"}],\"5671/t
cp\":[{\"HostPort\":\"5671\"}],\"8883/tcp\":[{\"HostPort\":\"8883\"}]}}}"
                        },
                        "type": "docker",
                        "status": "running",
                        "restartPolicy": "always"
                    }
                }
            }
        },
        "$edgeHub": {
            "properties.desired": {
                "routes": {
                    "route": "FROM /messages/* INTO $upstream",
                    "SimulatedTemperatureSensorToIoTHub": "FROM
/messages/modules/SimulatedTemperatureSensor/* INTO $upstream"
                },
                "schemaVersion": "1.1",
                "storeAndForwardConfiguration": {
                    "timeToLiveSecs": 7200
```

```
                }
            }
        },
        "SimulatedTemperatureSensor": {
            "properties.desired": {
                "SendData": true,
                "SendInterval": 5
            }
        }
    }
}
```

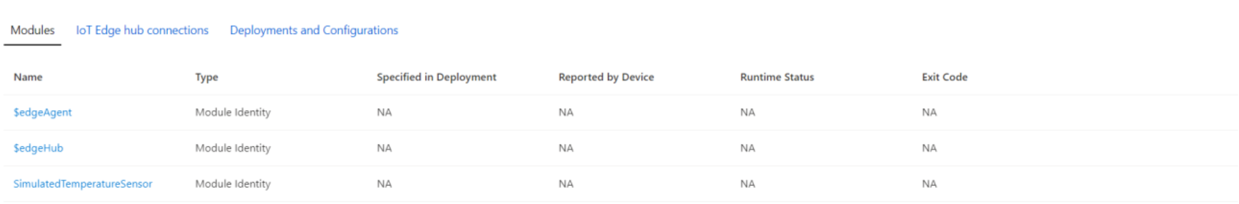
REVIEW AND CREATE



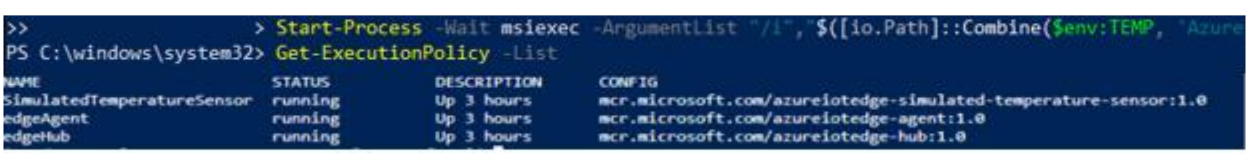
PART 5 – Viewing the device and the data

Finally we need to make sure that all of our data upto this point is up and running and we will push in the data and monitor it.

We need to push the Connect-EflowVm.

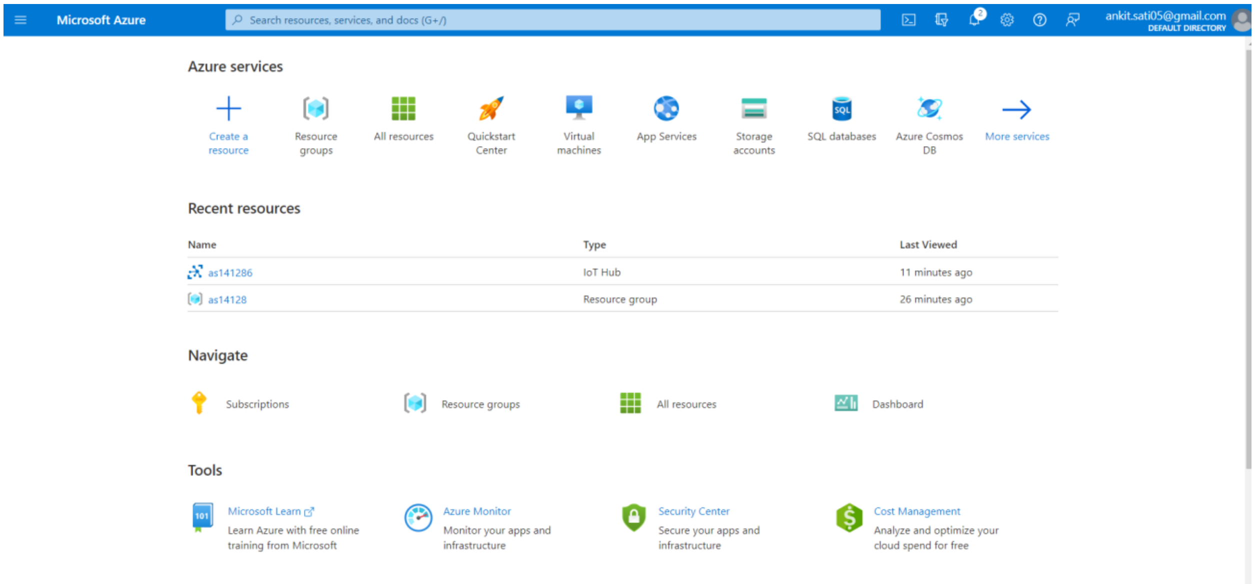


Monitor the data

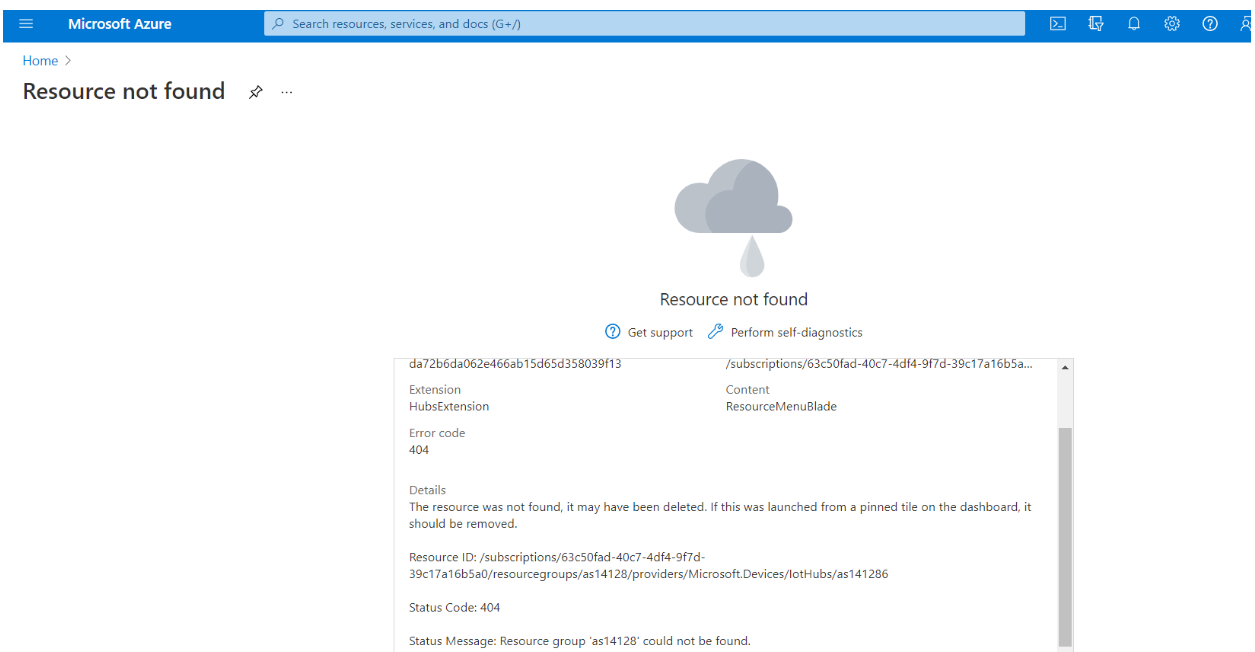


FINAL PART _ CLEANING UP THE RESOURCES

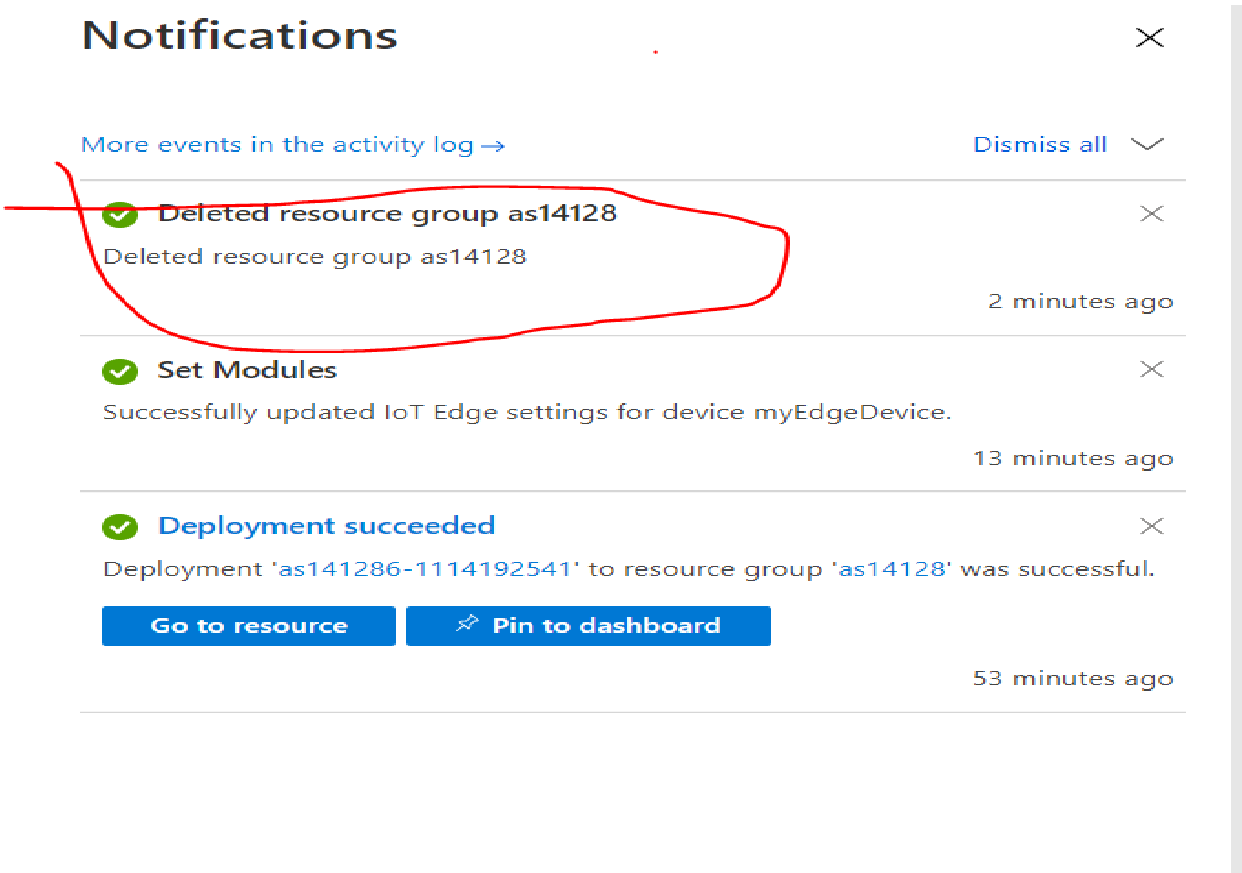
Final screenshot before deletion



Post Deletion

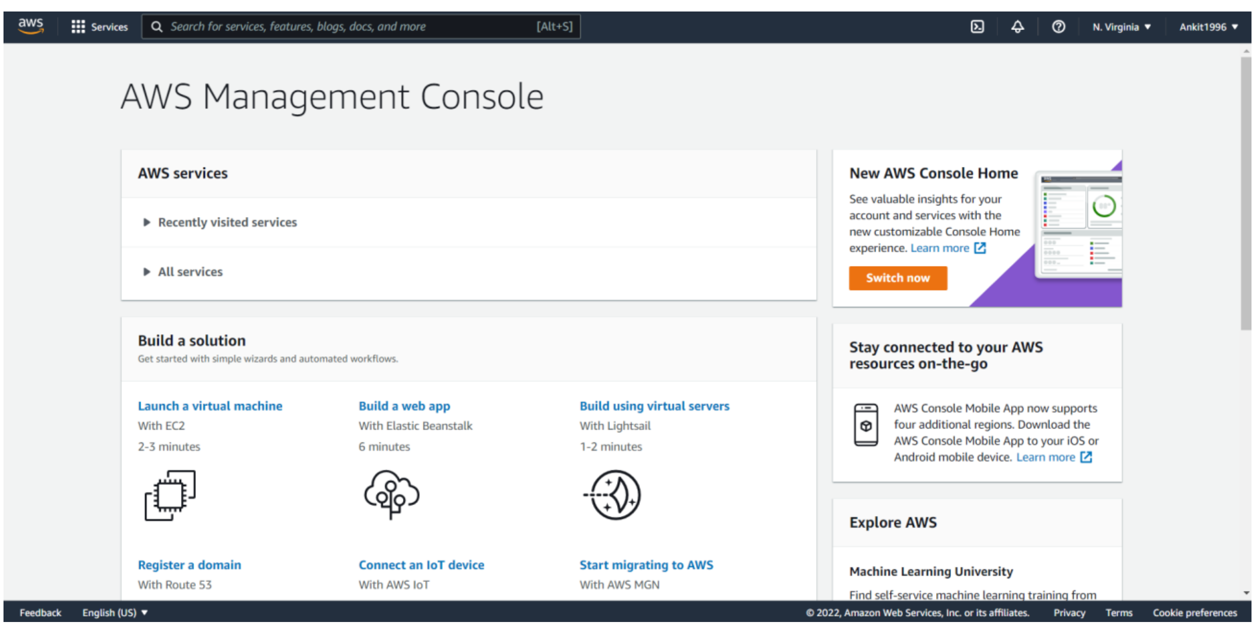


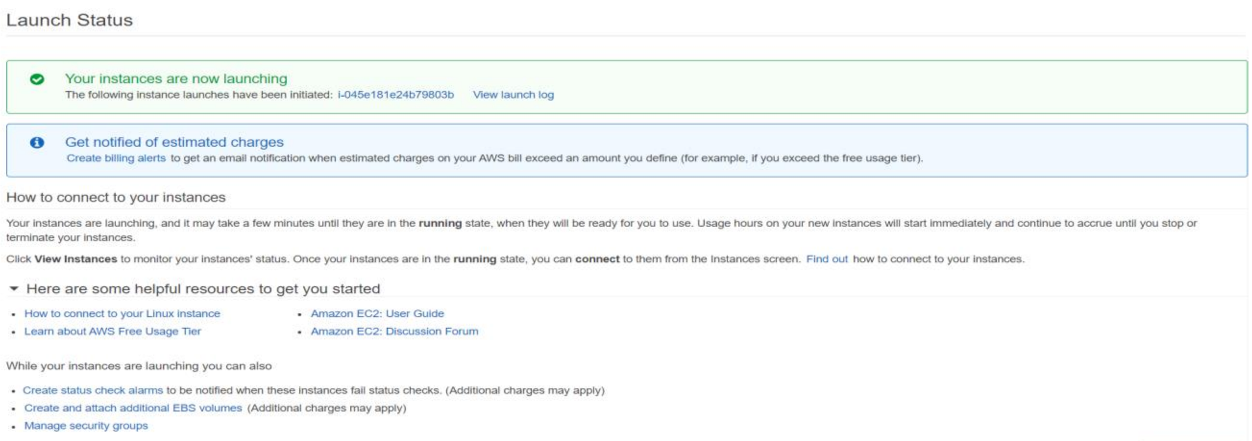
Activity log snippet

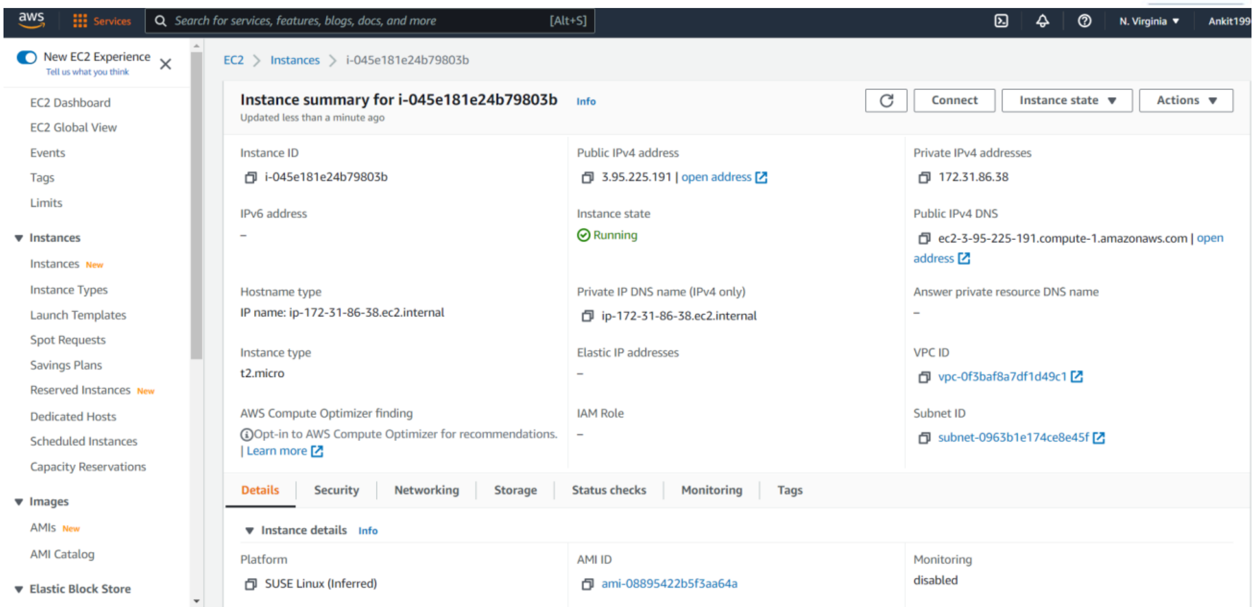


Example 2 – AWS

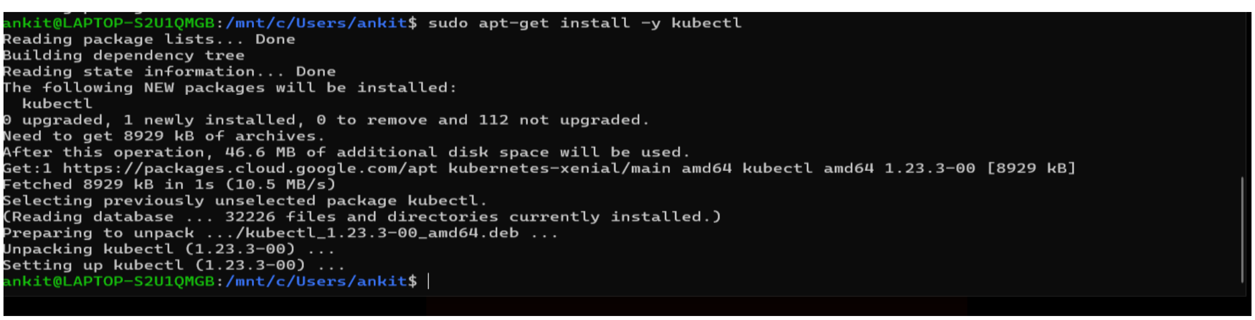
Now we need to implement the somewhat similar functionalities on AWS.



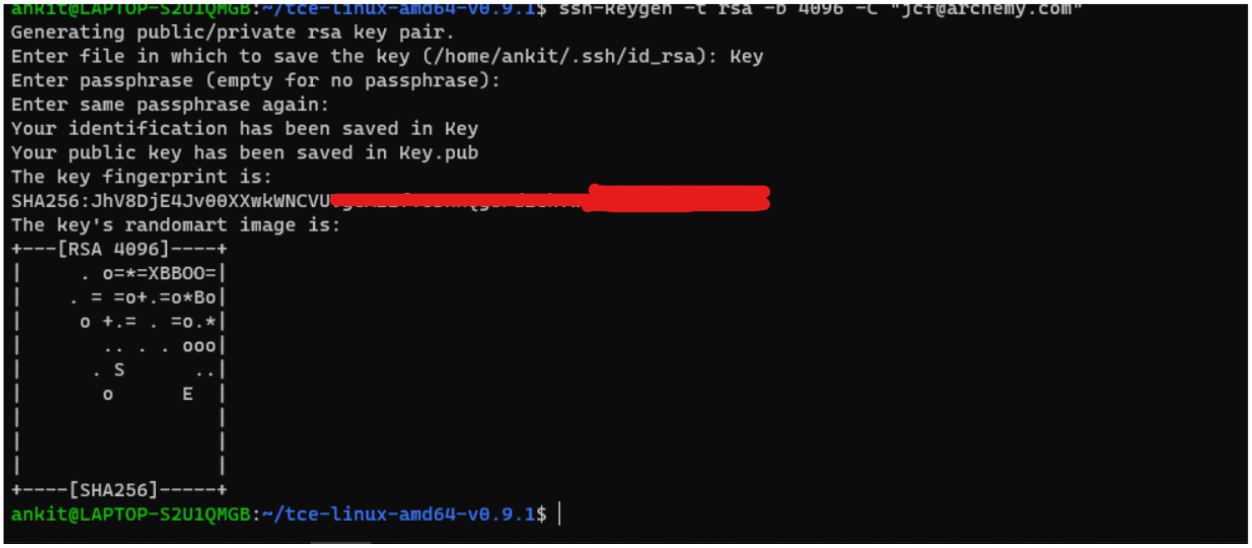


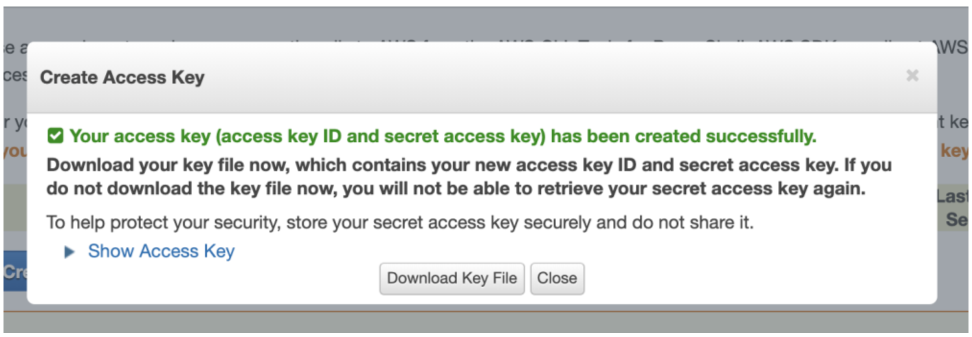


1. Step 2 and 3 – SSH into the instance that we have created.



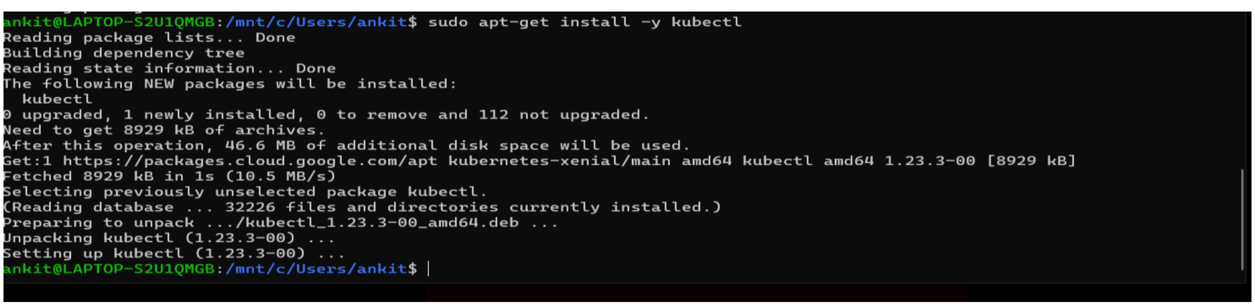
➔ Saving the required keys as mentioned in the assignment.
Key.pub identification
Type – RSA 4096
Key – Screenshot attached (Blurred the actual key for privacy)





- ➔ The final build needs to be created on the GUI first.
- ➔ Post that we need to setup the EC2 instance and the S3 storage bucket
- ➔ Finally, we can **ssh** into the created instance.

- **Screenshot of the final build**

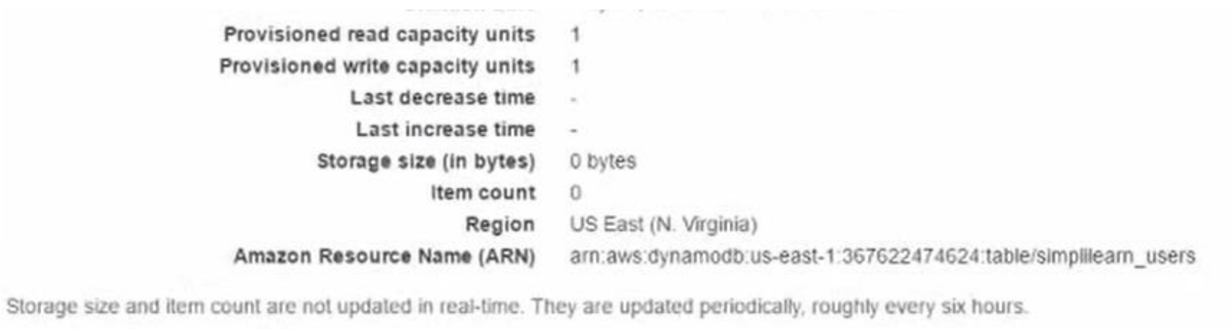


The CSV file- https://github.com/domoritz/random-csv

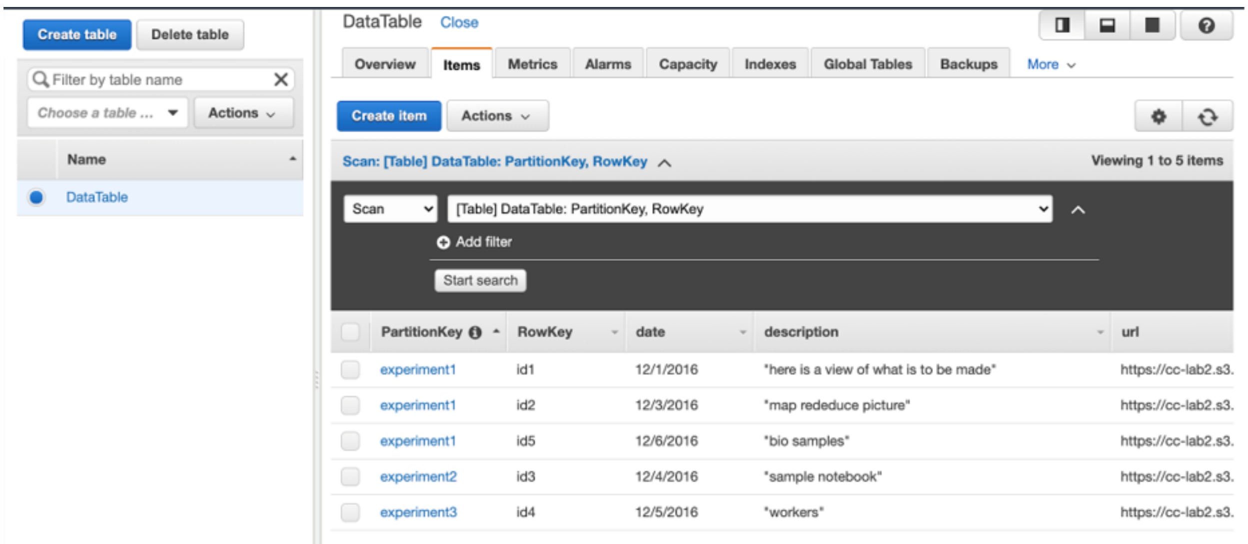
Step 4 – Single GPU push

- ➔ Here I have used a regular CSV file instead of the IOT repository as it is of no cost to us.

- ➔ The CSV file has the data of the mobile and temperature models which are the same ones used in azure cloud.
- ➔ The reason for it not implementing is that the storage buckets are built different here so we need more than 1 GPU and multiple **nodes to emulate the IOT sensors.**

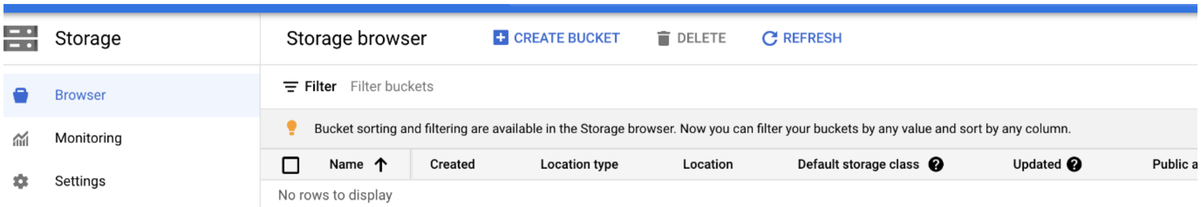


After running the notebook, go to aws console and go to dynamodb from there. Click on tables and under that click on DataTable and click on items tab to view the contents of the table.

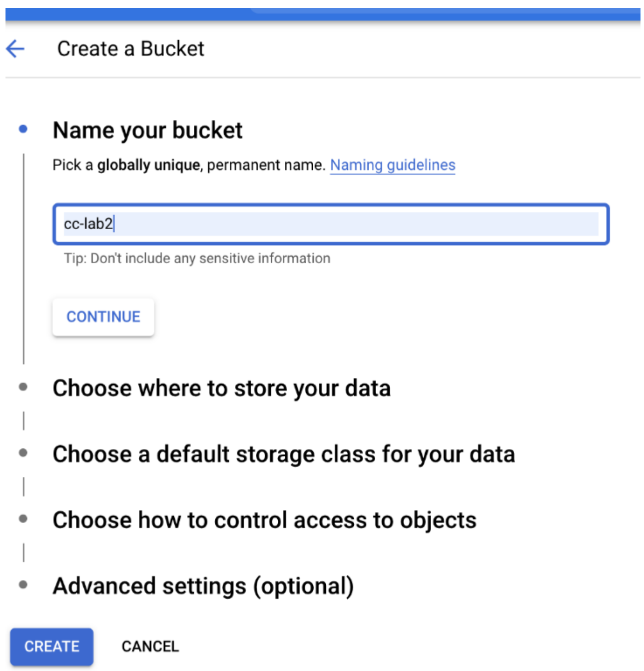


GCP

Login to GCP account and go to Storage from the navigation panel.



Click on create bucket and give a valid name.



Once the bucket is created, it should be visible

Once done, go to terminal and run the following command:

docker run -i -t -p 8888:8888 dbgannon/tutorial

After this go to local host link and run the python notebook called gcloud.ipynb after making necessary edits related to bucket name and file paths.

```
In [22]: client = storage.Client()

In [23]: from gcloud import datastore
         clientds = datastore.Client()

In [29]: import csv

In [53]: bucket = client.bucket('cc-lab2')
         key = clientds.key('book-table')

In [55]: with open('/datadir/experiments.csv','rt') as csvfile:
             csvf = csv.reader(csvfile, delimiter=',', quotechar='|')
             for item in csvf:
                 print(item)
                 blob = bucket.blob(item[3])
                 data = open("/datadir/"+item[3], 'rb')
                 blob.upload_from_file(data)
                 blob.make_public()
                 url = "https://console.cloud.google.com/storage/browser/cc-lab2/"+item[3]
                 entity = datastore.Entity(key=key)
                 entity['experiment-name'] = item[0]
                 entity['experiment-id'] = item[1]
                 entity['date'] = item[2]
                 entity['description'] = item[4]
                 entity['url'] = url
```

Once the above steps are executed, go to GCP again and go to storage to see the uploaded objects.

Buckets > cc-lab2

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER MANAGE HOLDS DOWNLOAD DELETE

Filter by name prefix only ▾ Filter Filter objects and folders

Name	Size	Type	Created time	Storage class	Last modified	Public access		Encryption	Retent
a	54.5 KB	application/octet-stream	18 Feb 2021, ...	Standard	18 Feb 202...	Public to Internet	Copy URL	Google-managed key	–
b	68 KB	application/octet-stream	18 Feb 2021, ...	Standard	18 Feb 202...	Public to Internet	Copy URL	Google-managed key	–
c	86.2 KB	application/octet-stream	18 Feb 2021, ...	Standard	18 Feb 202...	Public to Internet	Copy URL	Google-managed key	–
d	41.8 KB	application/octet-stream	18 Feb 2021, ...	Standard	18 Feb 202...	Public to Internet	Copy URL	Google-managed key	–
e	114.9 KB	application/octet-stream	18 Feb 2021, ...	Standard	18 Feb 202...	Public to Internet	Copy URL	Google-managed key	–

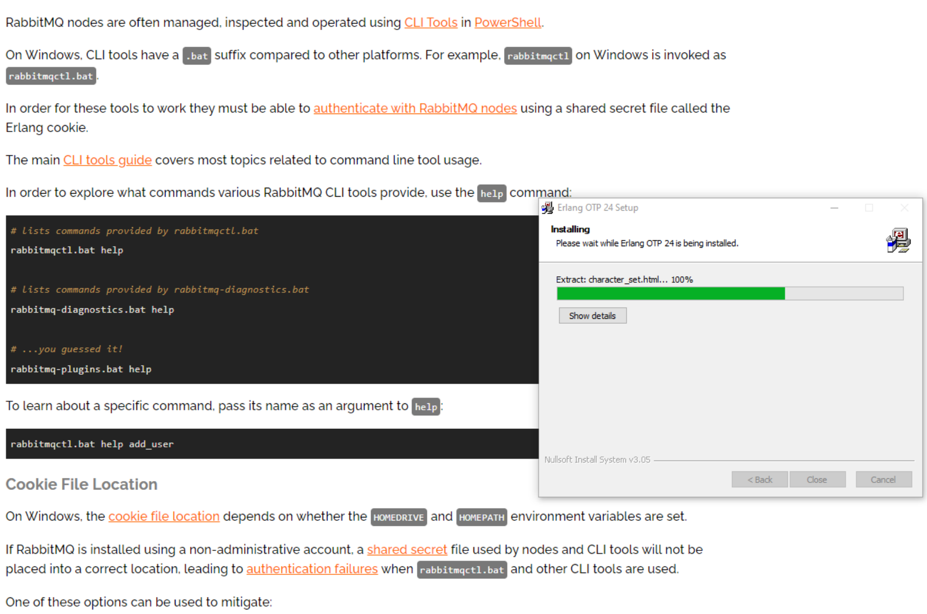
Check the datastore to have a look at created table

c. RabbitMQ

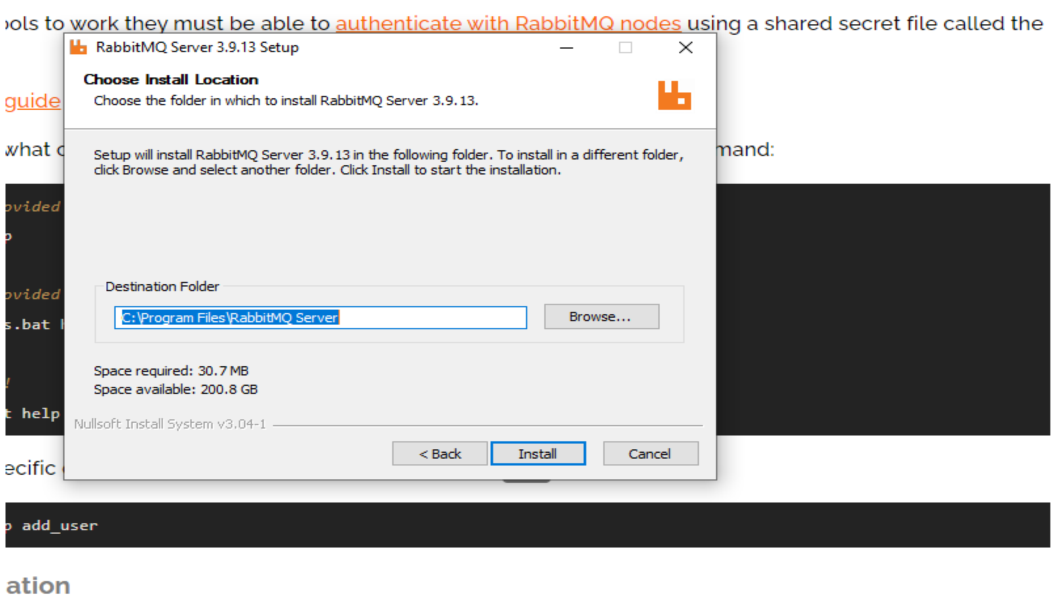
Now we will setup and push the same data in the RabbitMQ environment. RabbitMQ is an open-source message-broker software that originally implemented the Advanced Message Queuing Protocol and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol, MQ Telemetry Transport, and other protocols.

Installation and Setup part

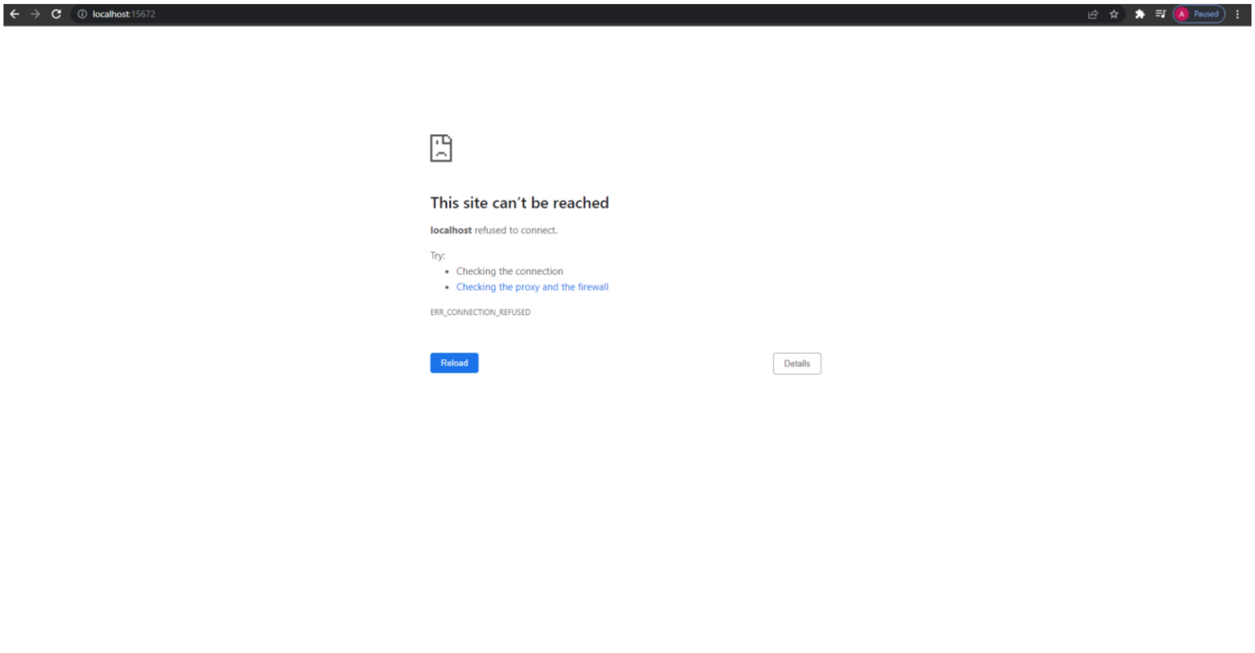
➔ Before setting up RabbitMQ we need to to download the ERLangPackges.



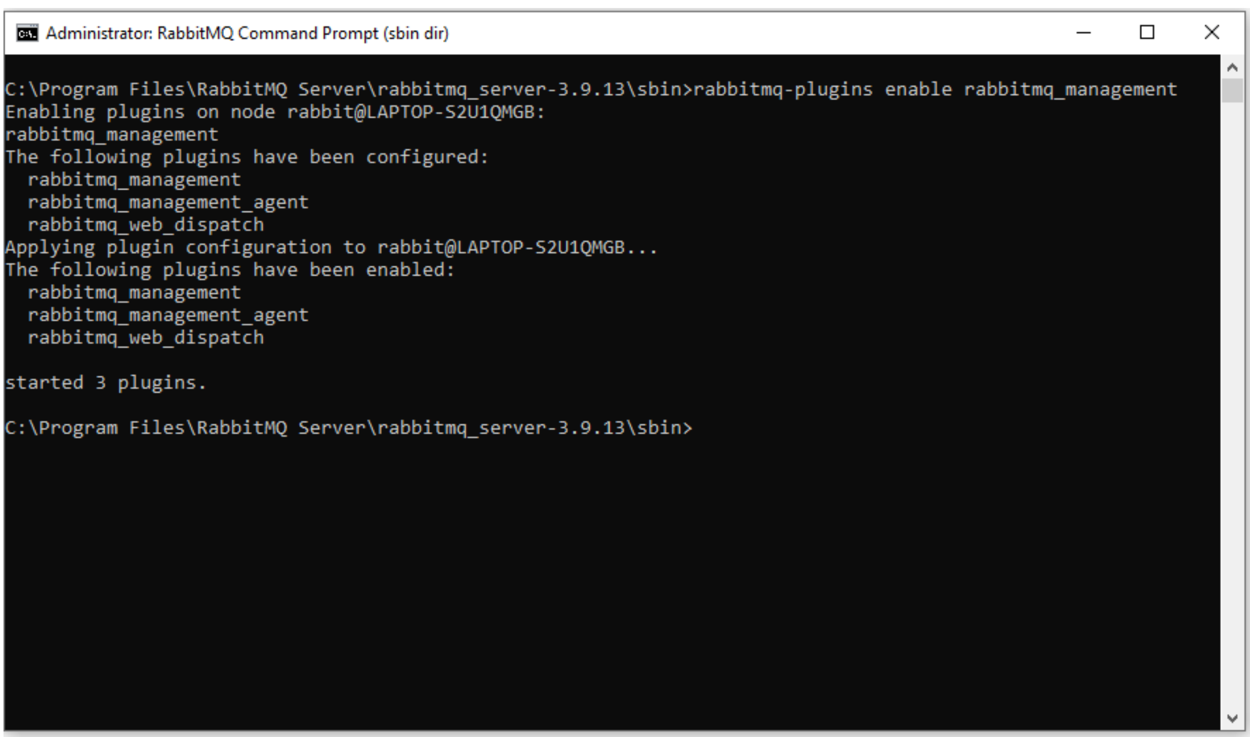
➔ After this we need to download and setup the RabbitMQ over our local machine.



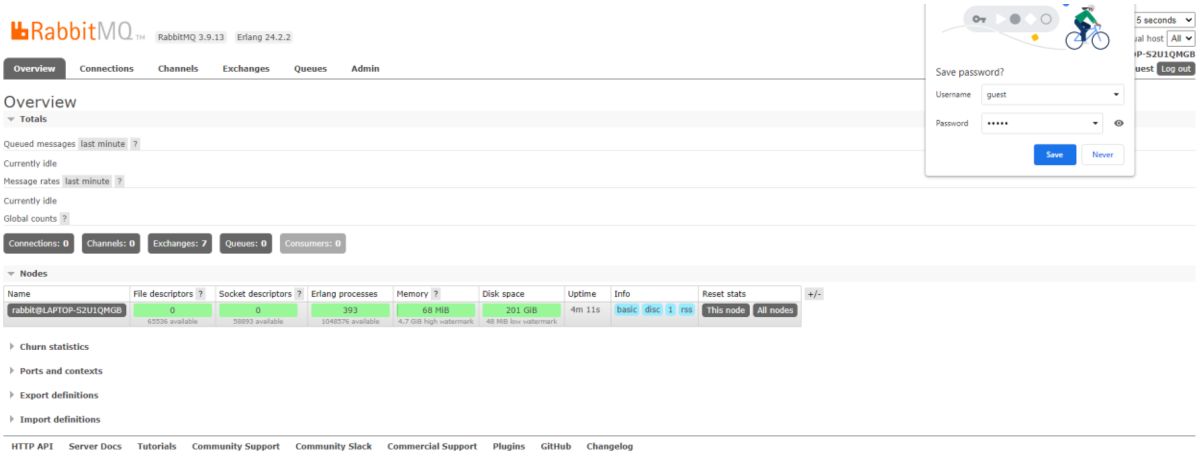
➔ Once this is setup we need to try and open the local host - http://localhost:15672/#/



➔ As we can see that the local host is not setup.
➔ We need to open the management plugnin to make it work,

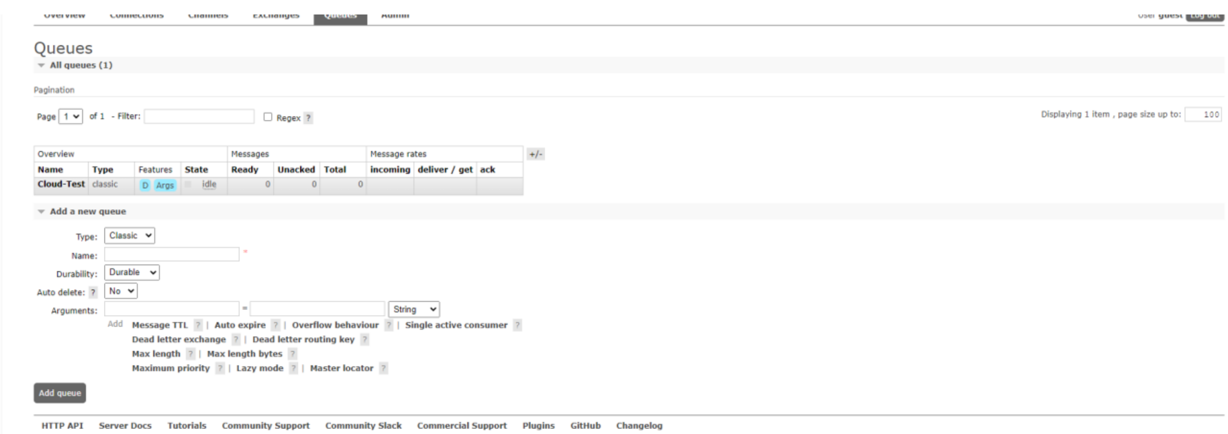


➔ After this we need to open the localhost to check the status of **RabbitMQ.**

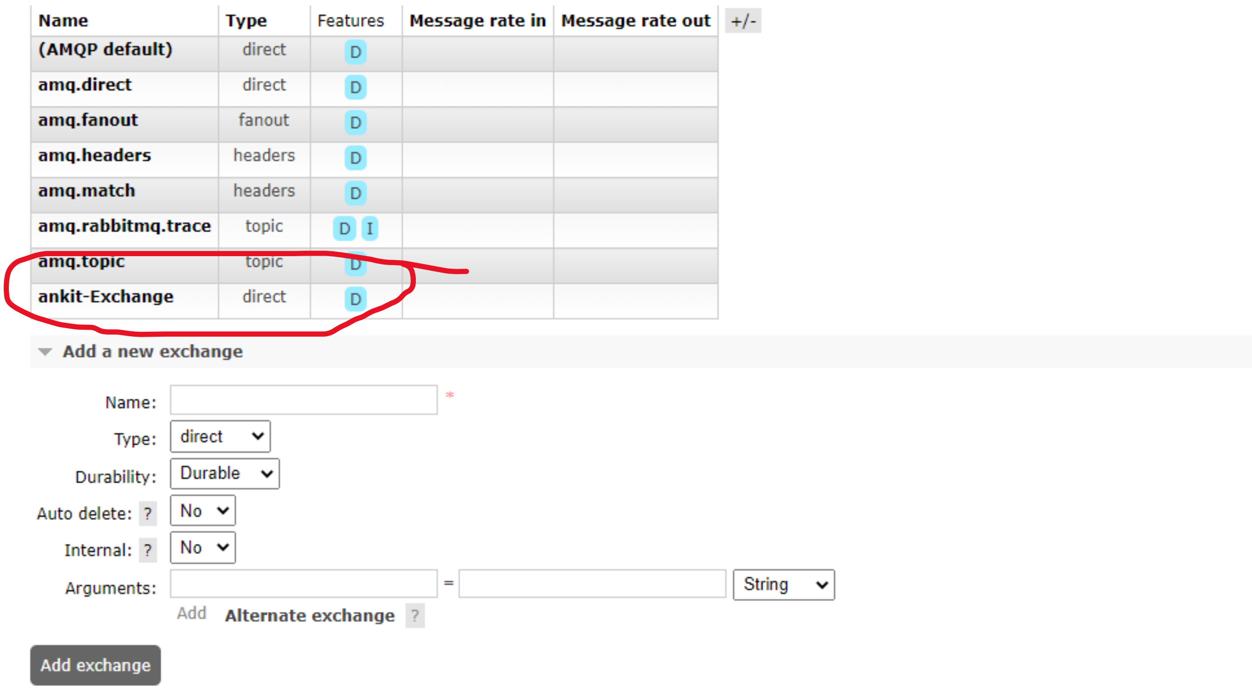


➔ Once this is done, we need to run a simple num.py file.

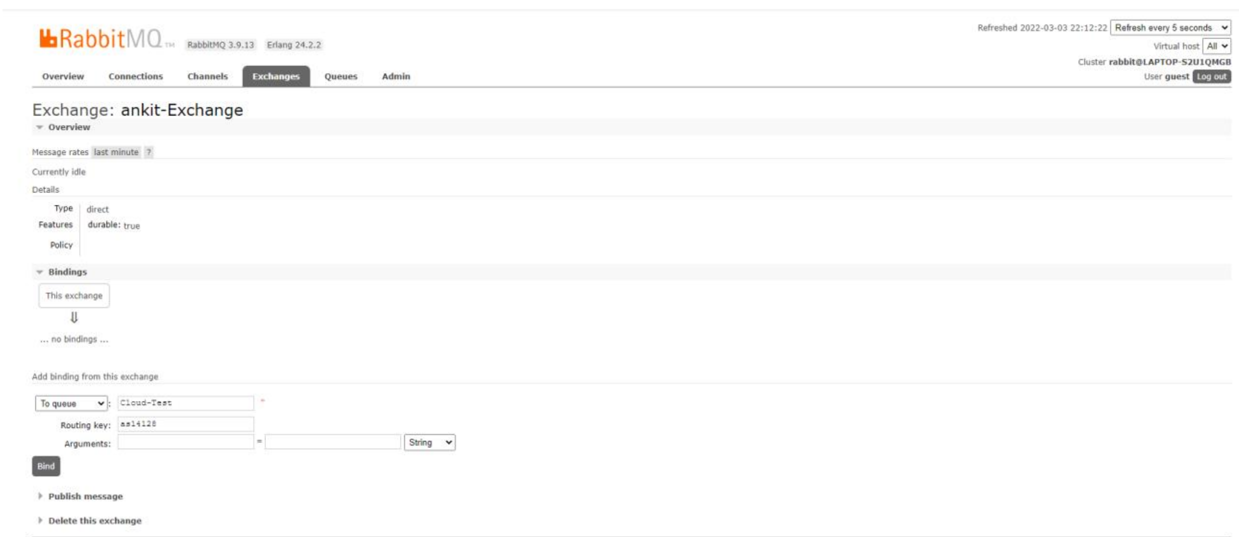
➔ Before that we will create a queue and open an channel.



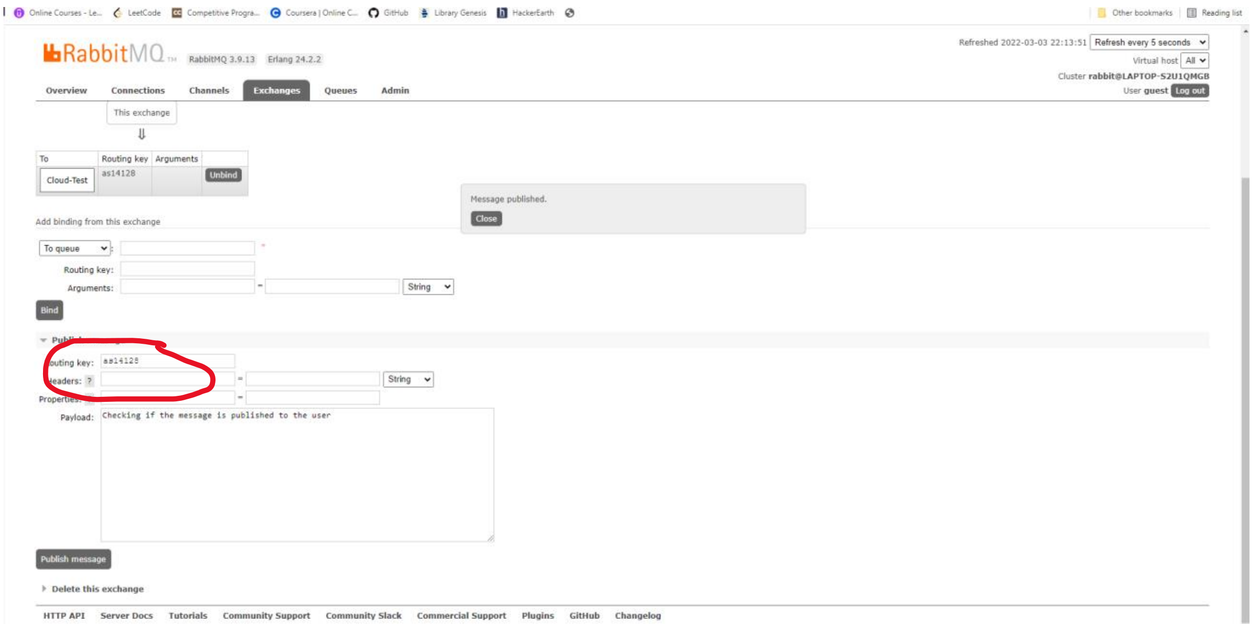
➔ After this we need to create an exchange so that the **run.py file** uploaded**.**



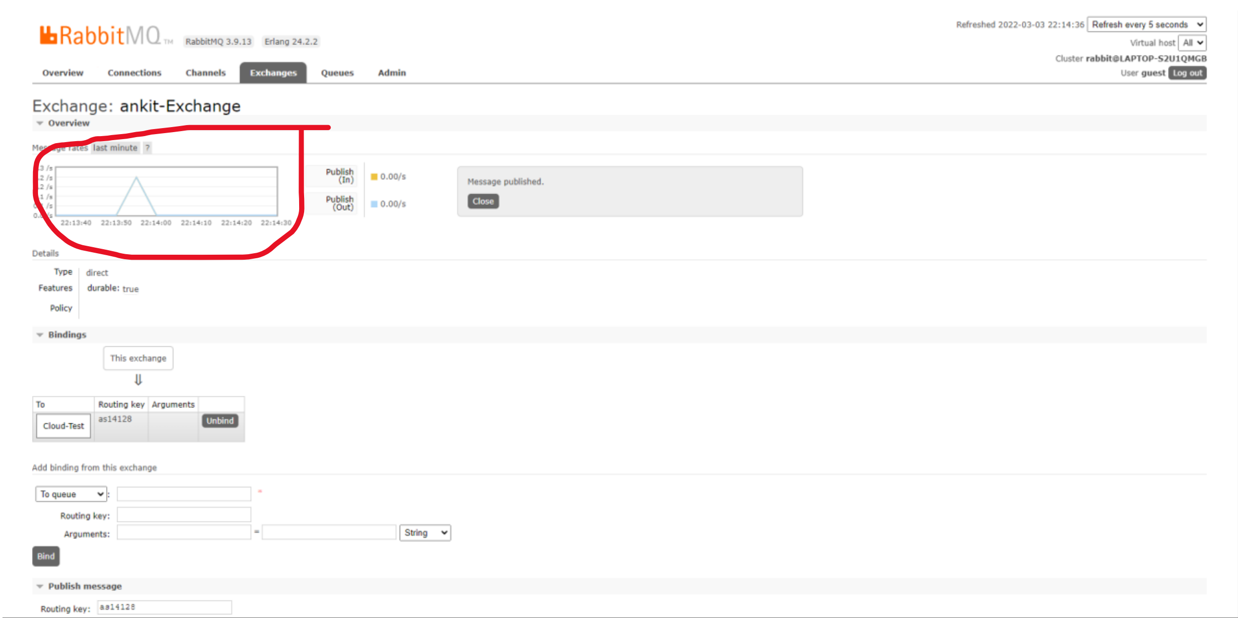
➔ Now we need to bind the file to the queue so that we can monitor the activity on the same. – **Binding the queue**



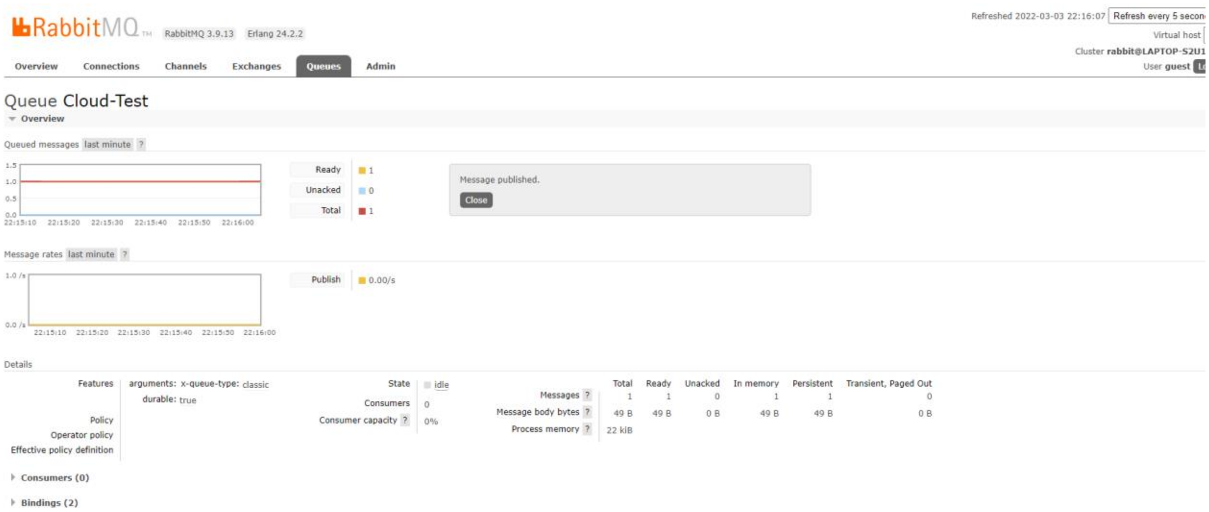
➔ Publishing the file and printing the message.



➔ Checking the activity on the client side.



- ➔ Checking the activity after pushing the data in a local cloud.
- ➔ CLOUD QUEUE TEST.

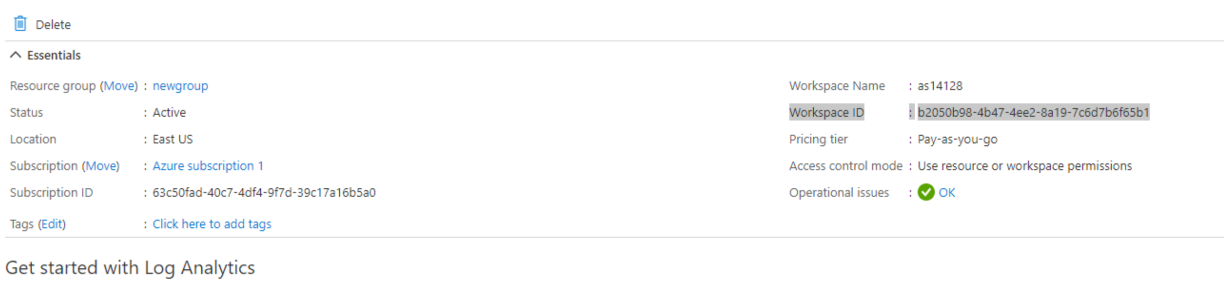


PART 3 and Extra credits.

Tutorial 1 - Monitor IoT Edge devices

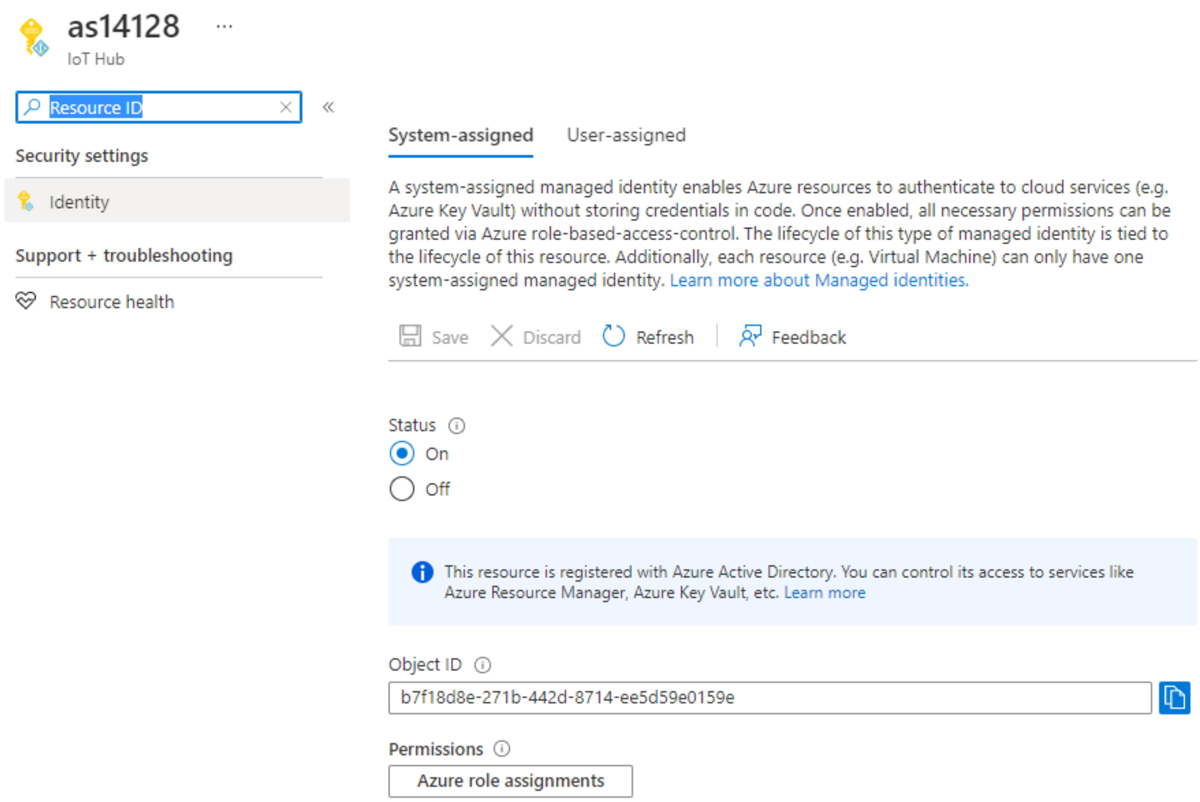
Step 1 - Create a Log Analytics workspace

Workspace ID : b2050b98-4b47-4ee2-8a19-7c6d7b6f65b1

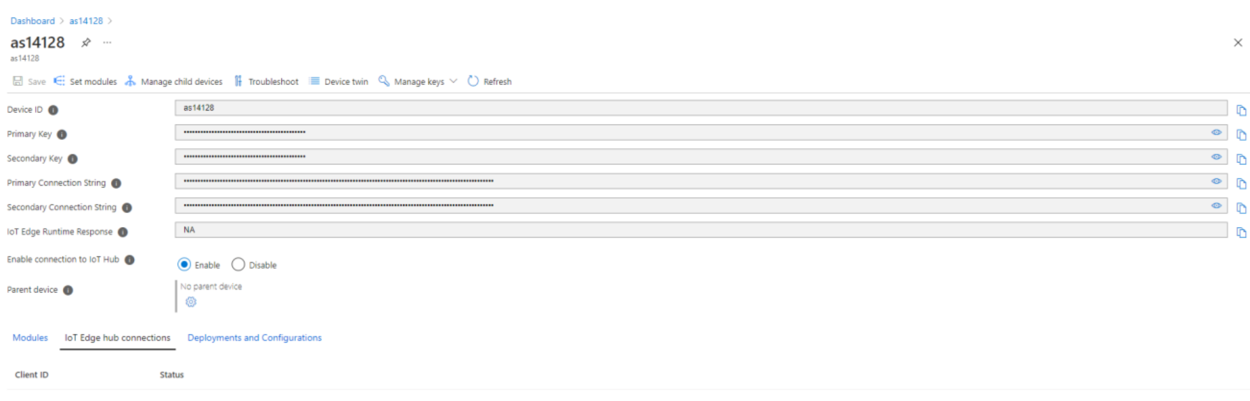


Step 2 - Create a Log Analytics workspace

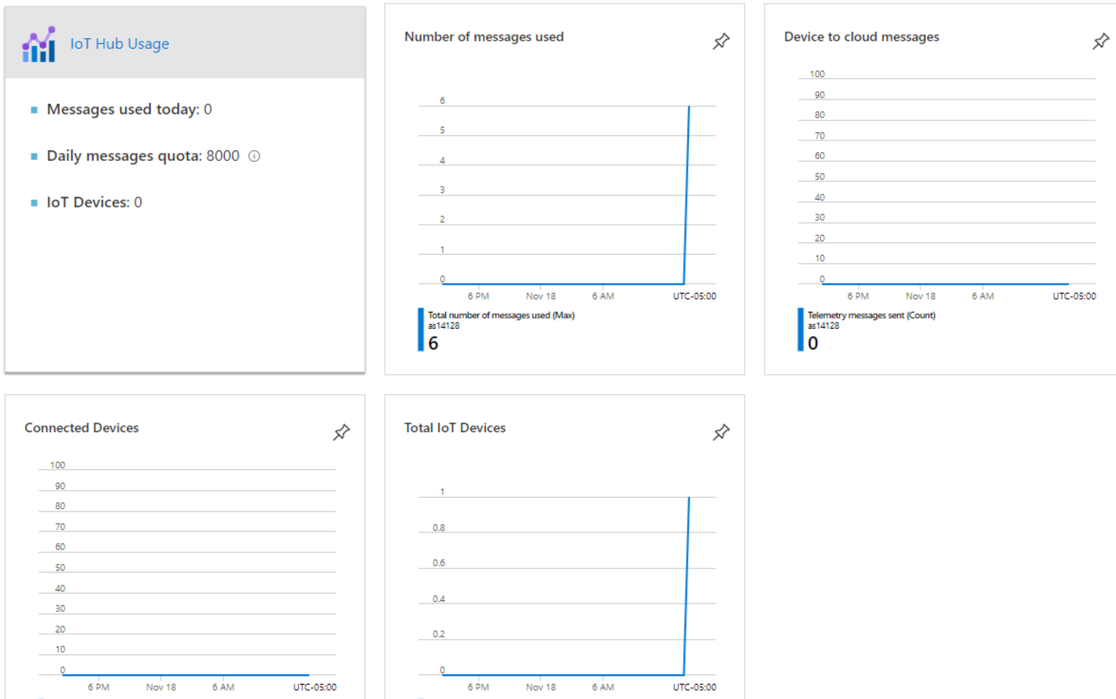
ID - b7f18d8e-271b-442d-8714-ee5d59e0159e



Step 3 - Deploy the metrics collector module



Step 4 - Explore the fleet view and health snapshot workbooks

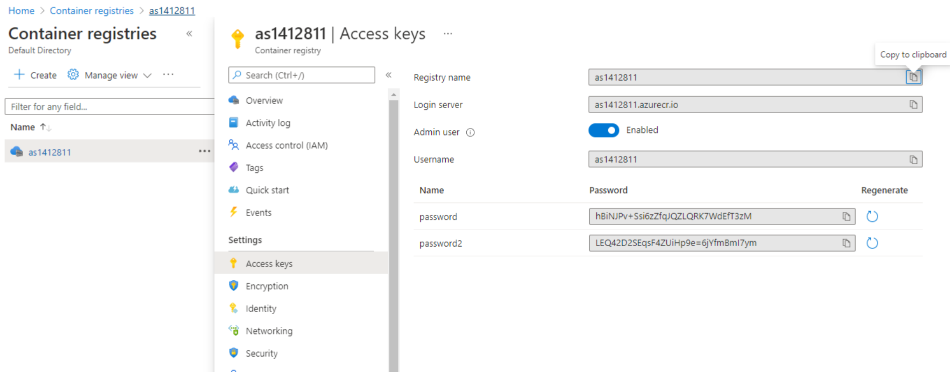


Tutorial 2 - Develop IoT Edge modules using Windows containers

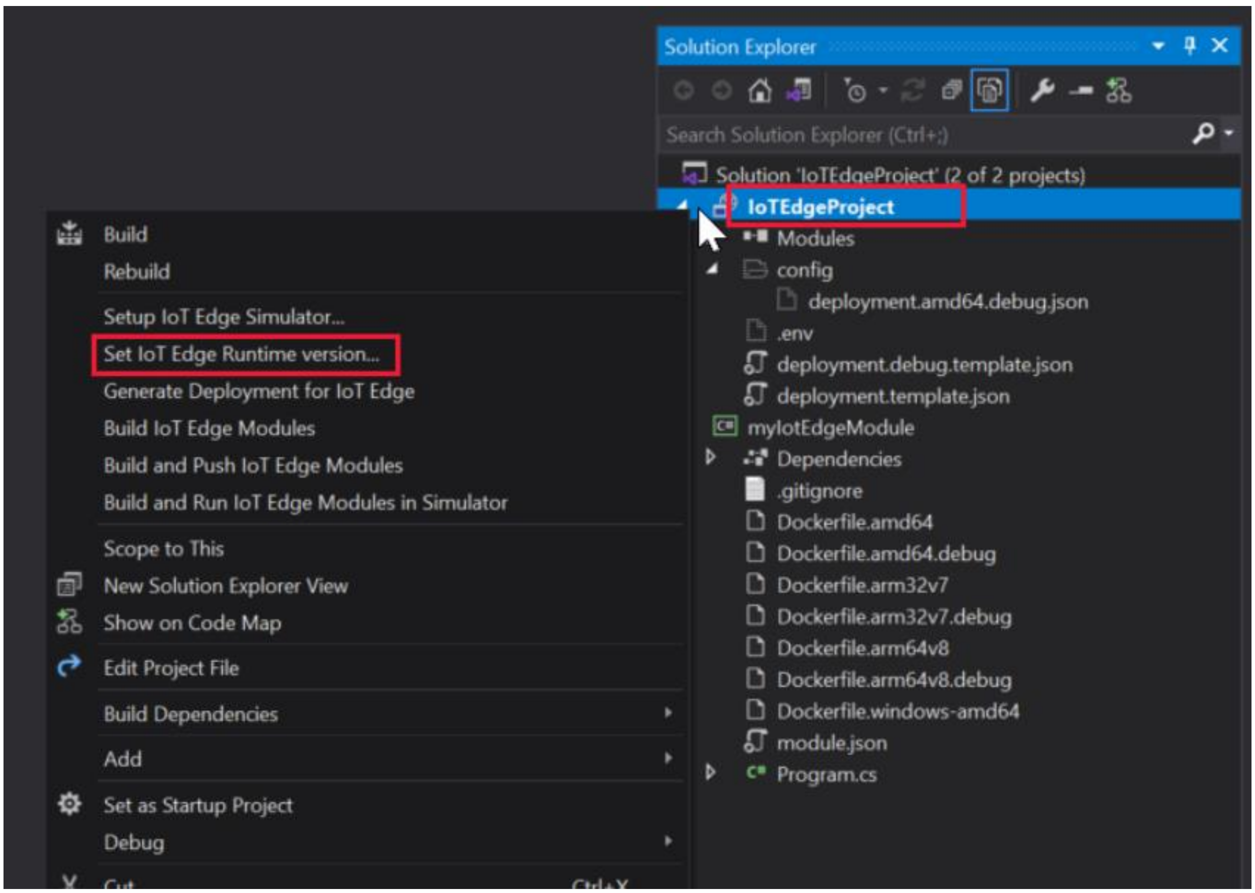
Step 1 - Set up VS

Installing environment and reps.

Step -2 – Create a container registry



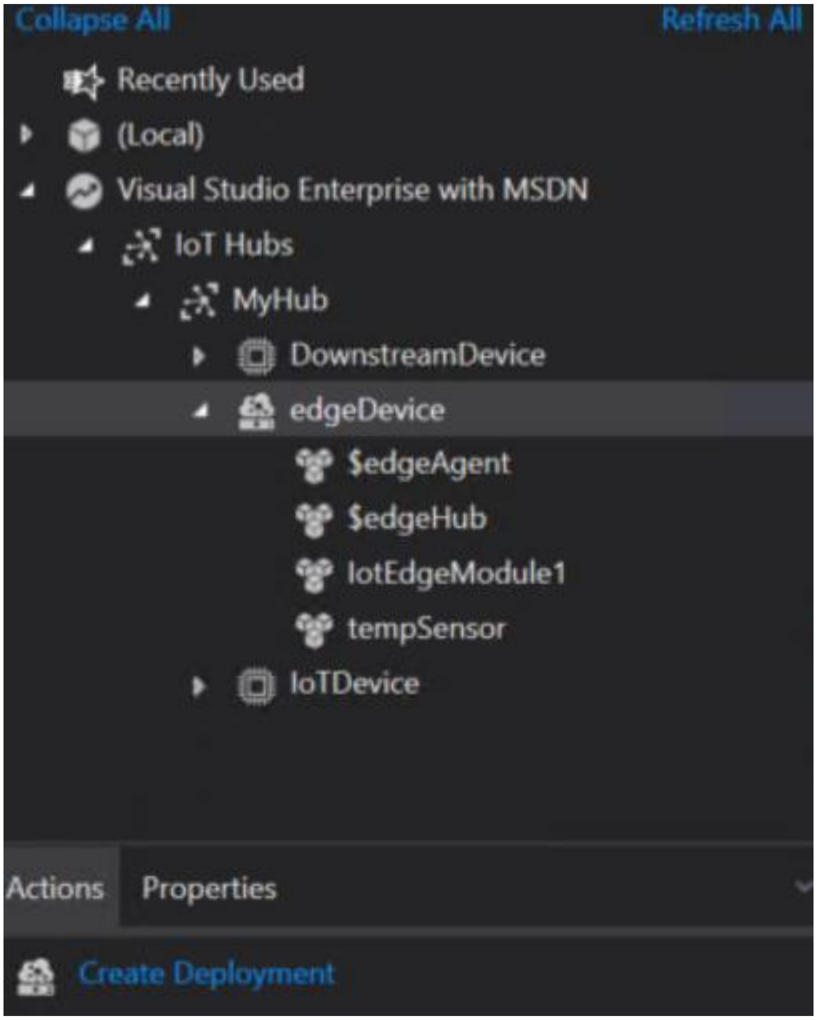
Step 3 – Edge Runtime



Step 4 - Provide your registry credentials to the IoT Edge agent

```
"registryCredentials": {
  "<registry name>": {
    "username": "$CONTAINER_REGISTRY_USERNAME_<registry name>",
    "password": "$CONTAINER_REGISTRY_PASSWORD_<registry name>",
    "address": "<registry name>.azurecr.io"
  }
}
```

Step 5 - Changes on Device



STEP 6 –

Clean up resource

- **Resources – Done**
- **Modules – Done**
- **Devices done**

Part 4 – Custom code

Step 1 - Set up Java

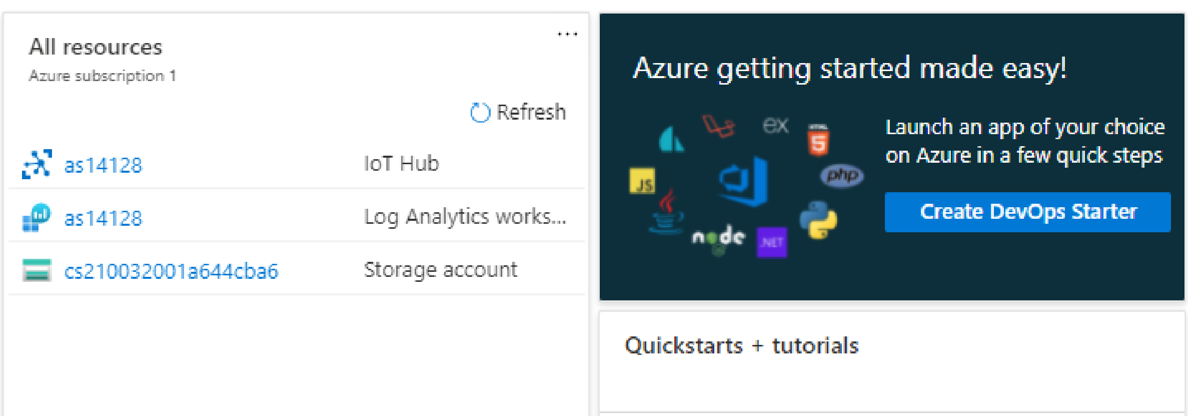
Installing environment and reps.

Step 2 – Code applet

```
import java.util.Map;

import javax.json.Json;
import javax.json.JsonObject;
import javax.json.JsonReader;

import com.microsoft.azure.sdk.iot.device.DeviceTwin.Pair;
import com.microsoft.azure.sdk.iot.device.DeviceTwin.Property;
import com.microsoft.azure.sdk.iot.device.DeviceTwin.TwinPropertyCallBack;

private static final String TEMP_THRESHOLD = "TemperatureThreshold";
private static AtomicLong tempThreshold = new AtomicLong(25);

protected static class MessageCallbackMqtt implements MessageCallback {
    private int counter = 0;
    @Override
    public IotHubMessageResult execute(Message msg, Object context) {
        this.counter += 1;

        String msgString = new String(msg.getBytes(), Message.DEFAULT_IOTHUB_MESSAGE_CHARSET);
        System.out.println(
               String.format("Received message %d: %s",
                        this.counter, msgString));
        if (context instanceof ModuleClient) {
            try (JsonReader jsonReader = Json.createReader(new StringReader(msgString))) {
                final JsonObject msgObject = jsonReader.readObject();
                double temperature = msgObject.getJsonObject("machine").getJsonNumber("temperature").doubleValue();
                long threshold = App.tempThreshold.get();
                if (temperature >= threshold) {
                    ModuleClient client = (ModuleClient) context;
                    System.out.println(
                        String.format("Temperature above threshold %d. Sending message: %s",
                        threshold, msgString));
                    client.sendEventAsync(msg, eventCallback, msg, App.OUTPUT_NAME);
                }
```

Step 3 – Push the custom modules

Step 4 - Deploy modules to device



Step 5 - Edit the module twin

```java
import com.microsoft.azure.sdk.iot.device.DeviceTwin.TwinPropertyCallBack;

private static final String TEMP_THRESHOLD = "TemperatureThreshold";
private static AtomicLong tempThreshold = new AtomicLong(5);

protected static class MessageCallbackMqtt implements MessageCallback {
    private int counter = 0;
```

Step 6 – Delete all IOT edge modules.

- **Resources – Done**
- **Modules – Done**
- **Devices done**