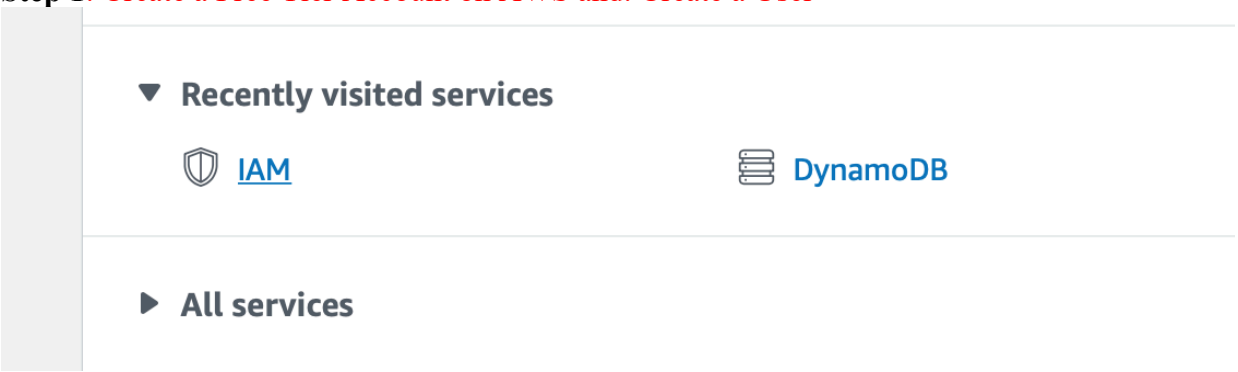
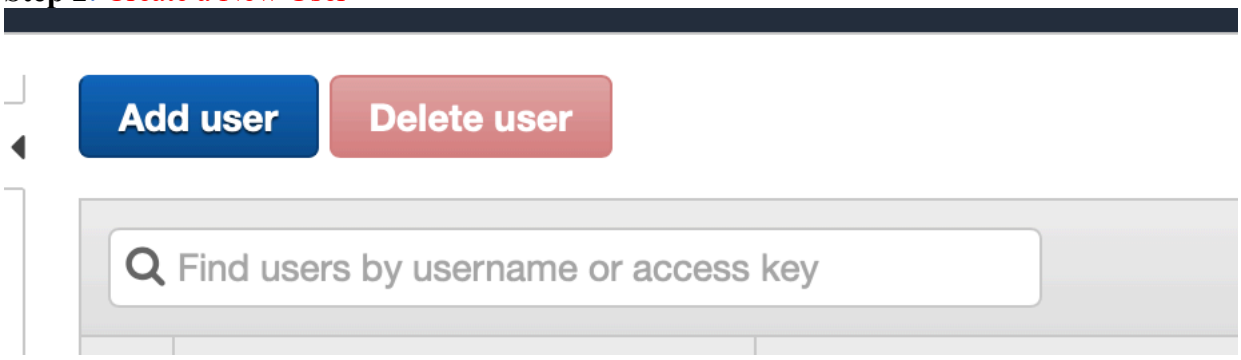
Soumil Nitn Shah
soushah@my.bridgeport.edu

Getting started with AWS IOT Core and Python SDK using Design Pattern

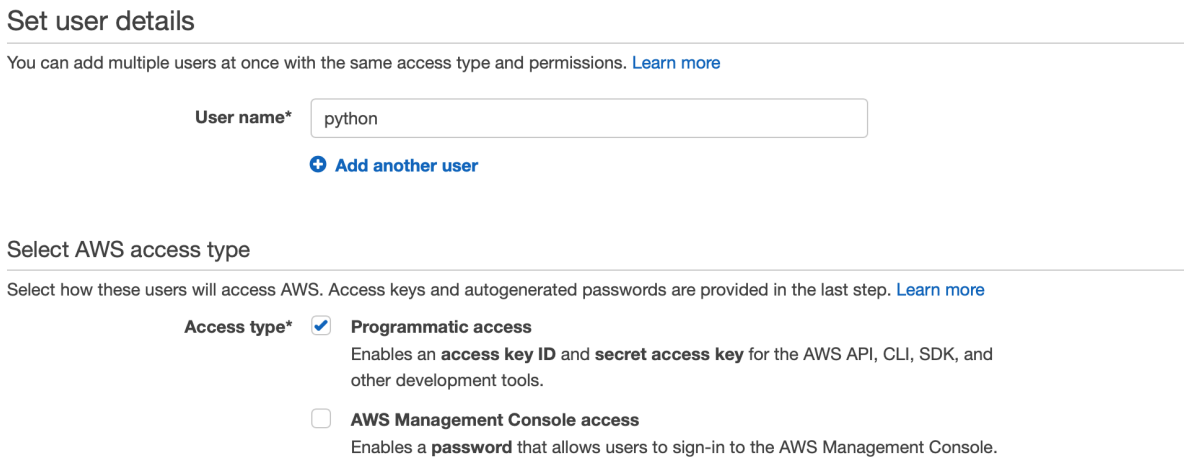
Step 1: Create a Free Tier Account on AWS and. Create a User



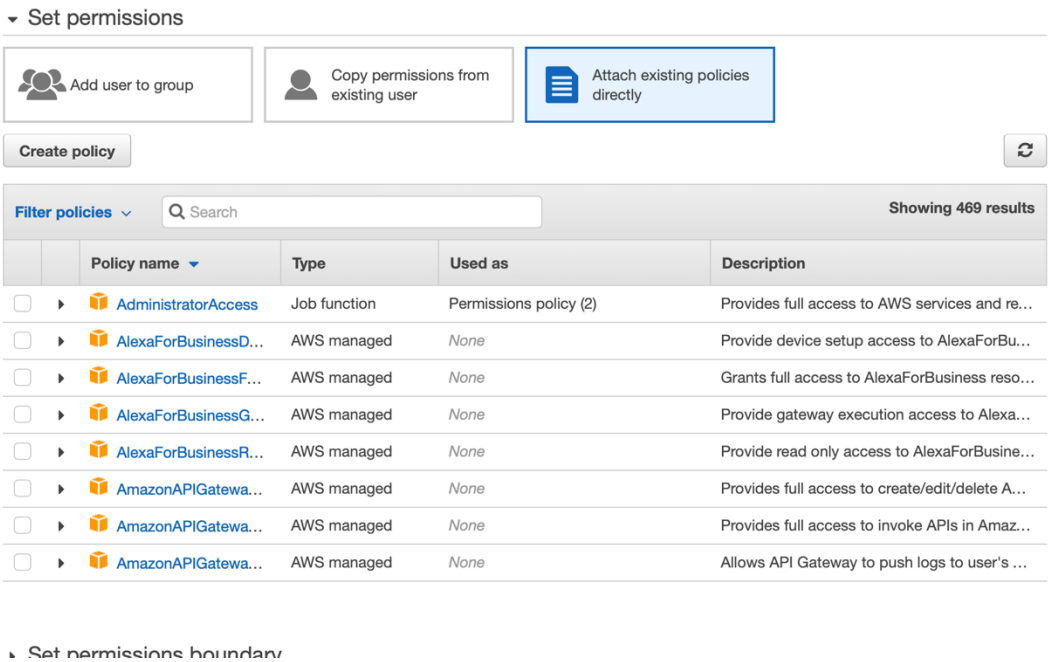
Step 2: Create a New User



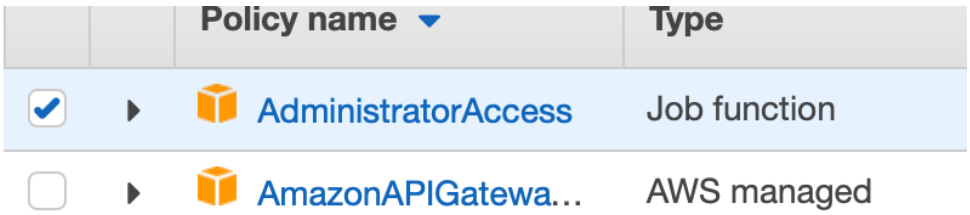
Step 3: Add username as Python and make sure access type is as Programmatic



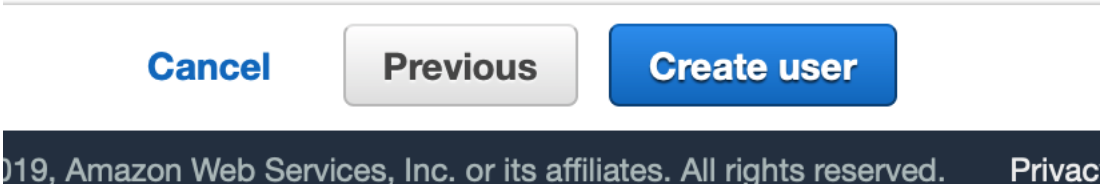
Step 4: Create a Policy Make sure to Give Admin Access



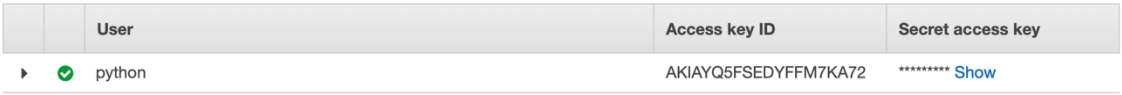
Click on Administrative Access



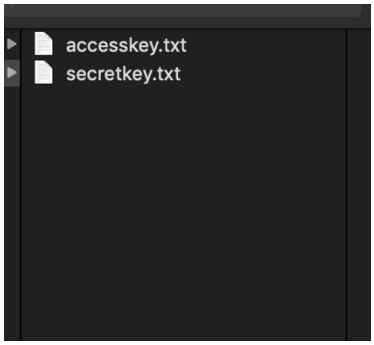
Step 5: Create the user



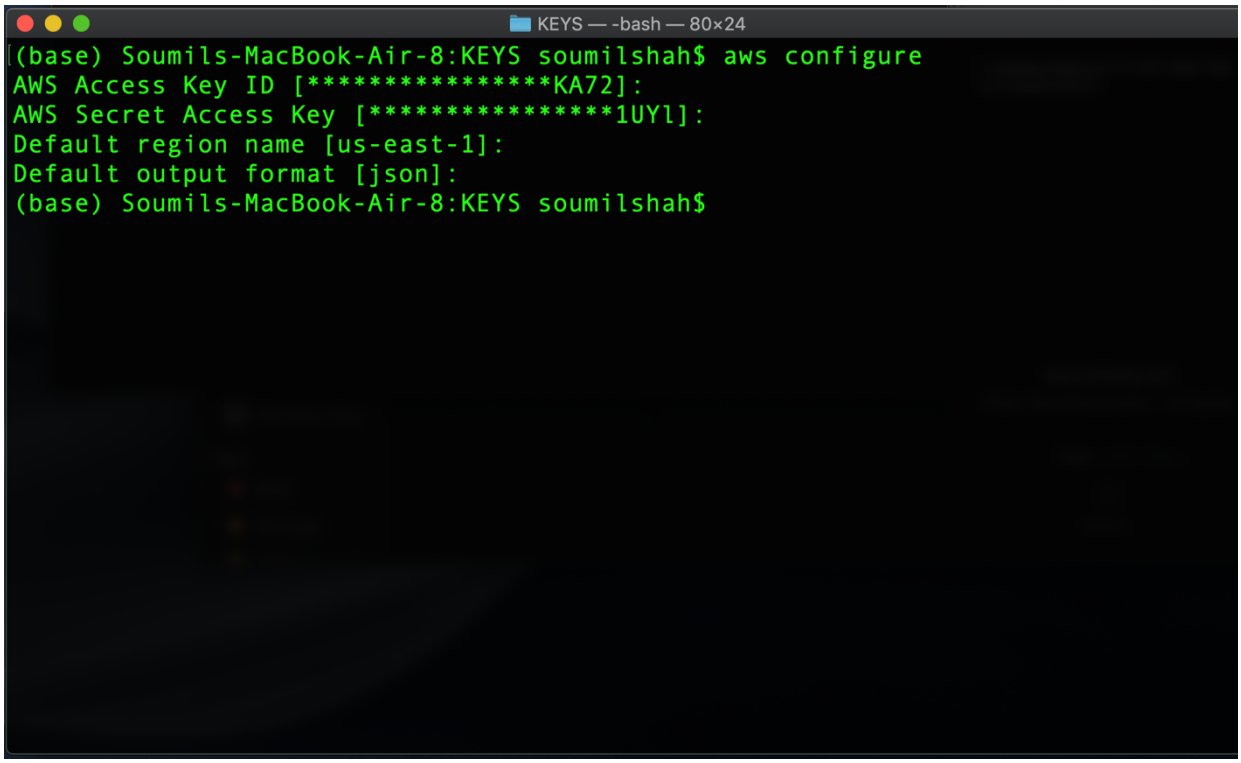
Step 7: Copy the Secrete Key and Access Key



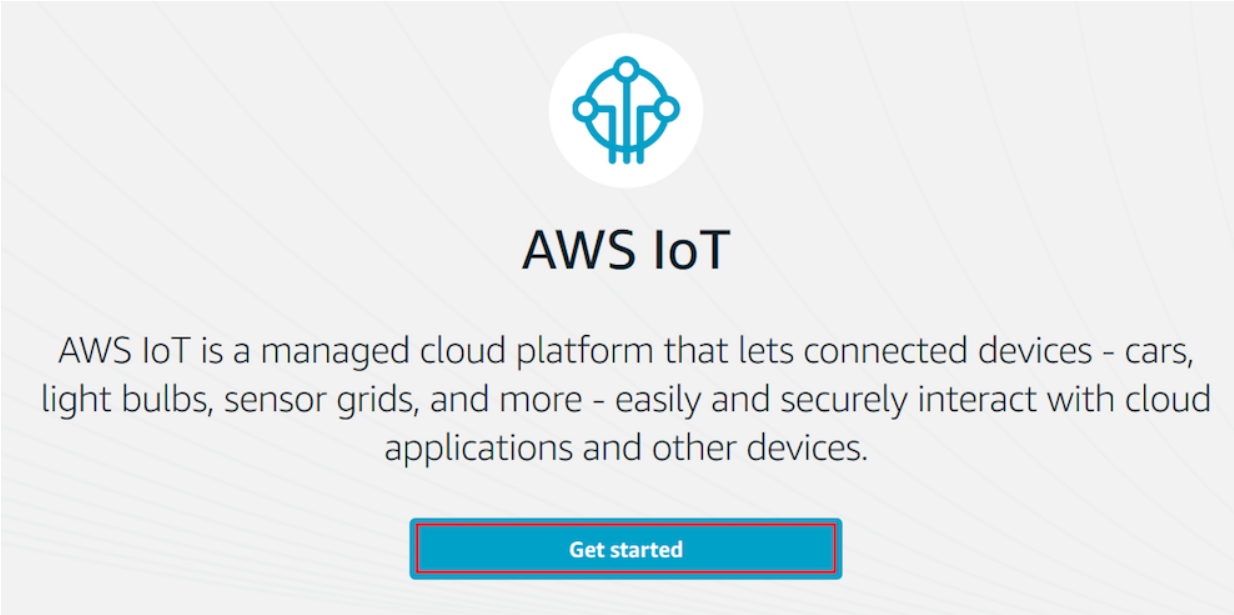
Step 7: Create a folder and store the access key and secret key in that folder



Step 8: Configure AWS CLI if you haven't installed pip install aws-cli or go the the documentation to download AWS CLI once downloaded we need to configure type the following command on mac or command Shell AWS configure and paste your credentials

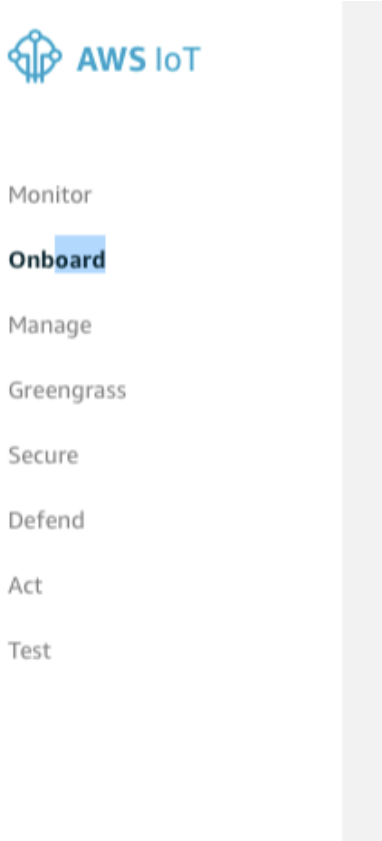


Step 9: Configure AWS IoT Core

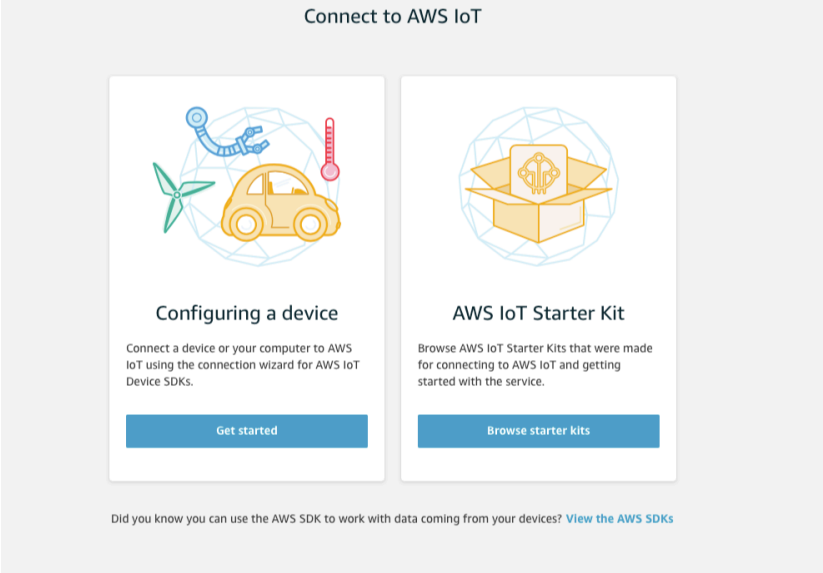


Devices connected to AWS IoT are represented by *IoT things* in the AWS IoT registry. The registry allows you to keep a record of all of the devices that are registered to your AWS IoT account

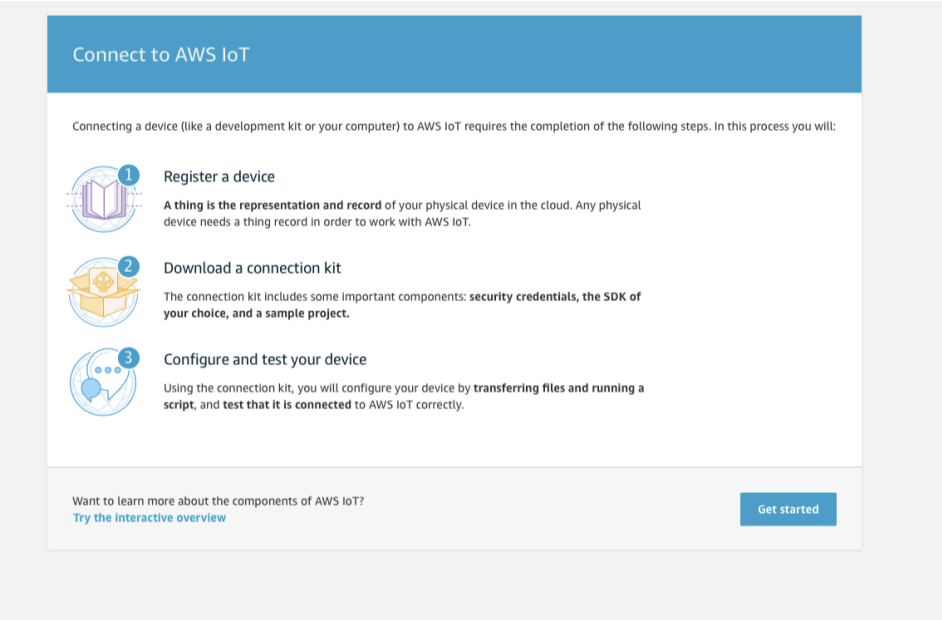
Step 10: Go to the Onboard and click on getting started



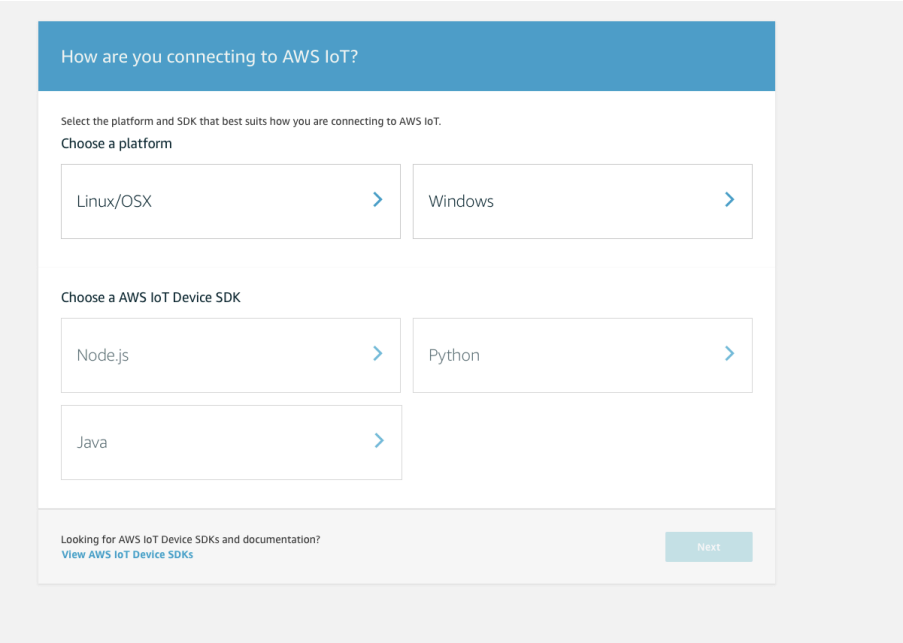
Step 11: Click on configure Device

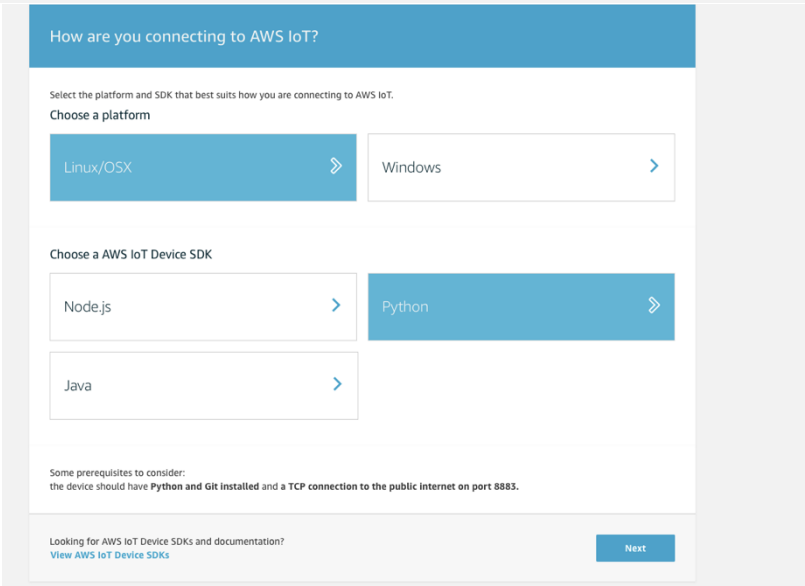


Step 12: Click on Getting started Button



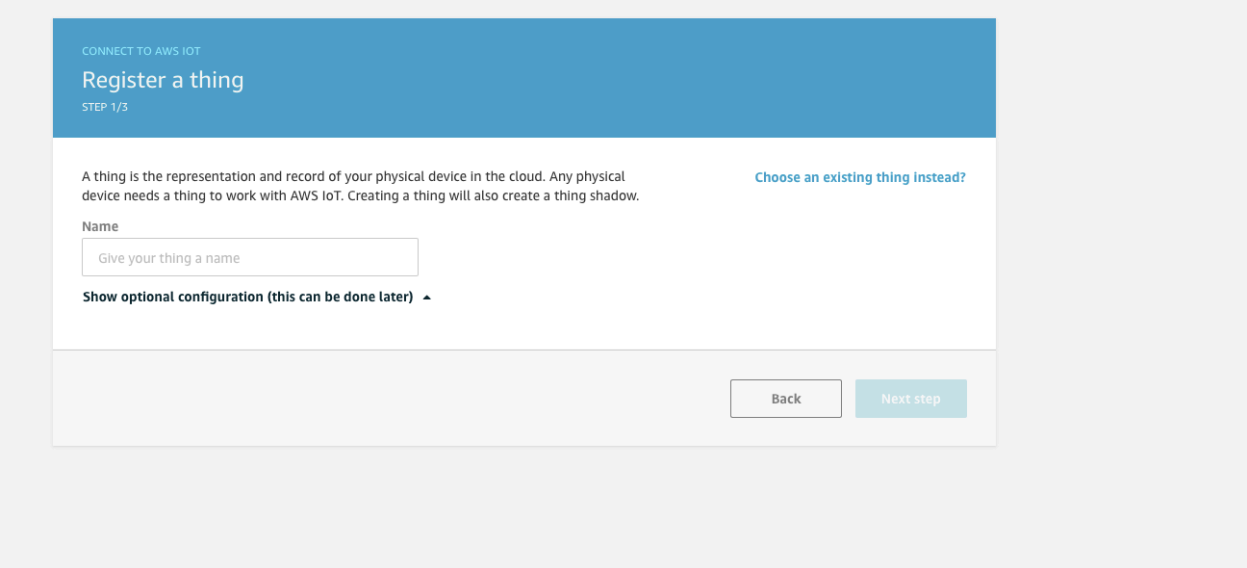
Step 13: Choose your operating system. I will choose MacOs but whatever Os you are using please choosing your operating system

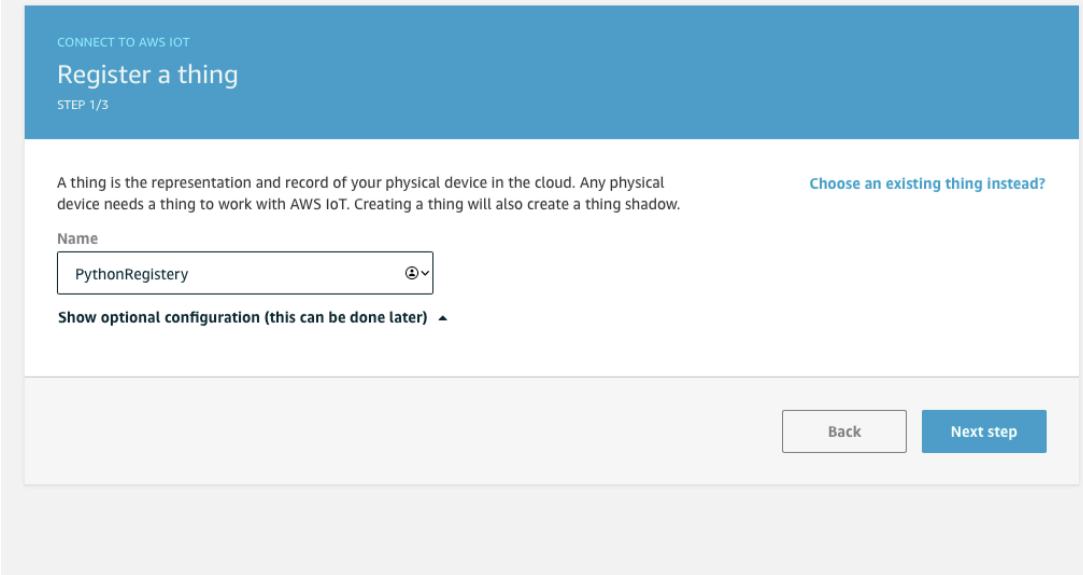




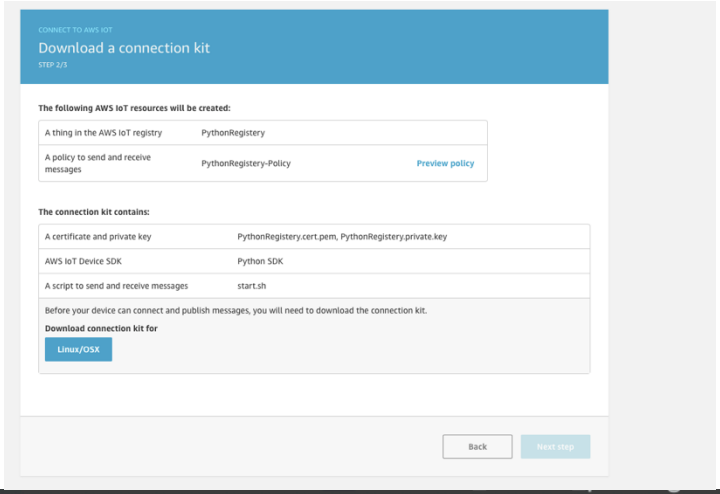
Choose the language as python and click on next

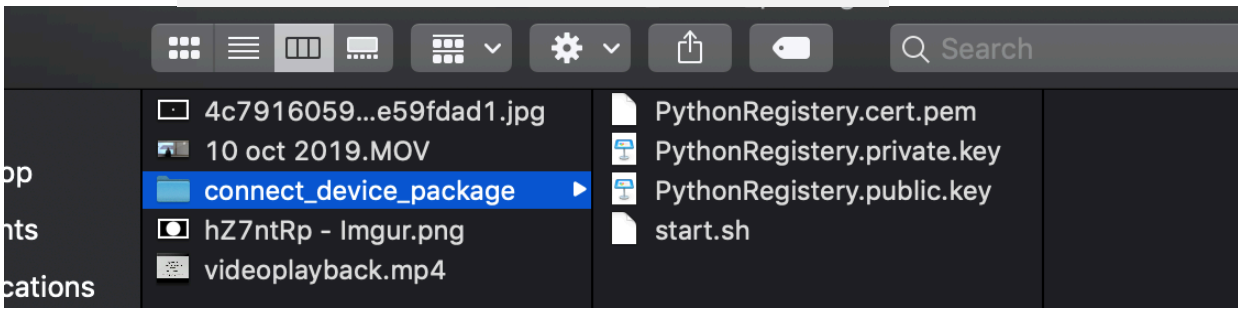
Step 14: we need to give a name to the devices I will name it as PythonRegistrer





Once complete click on next button





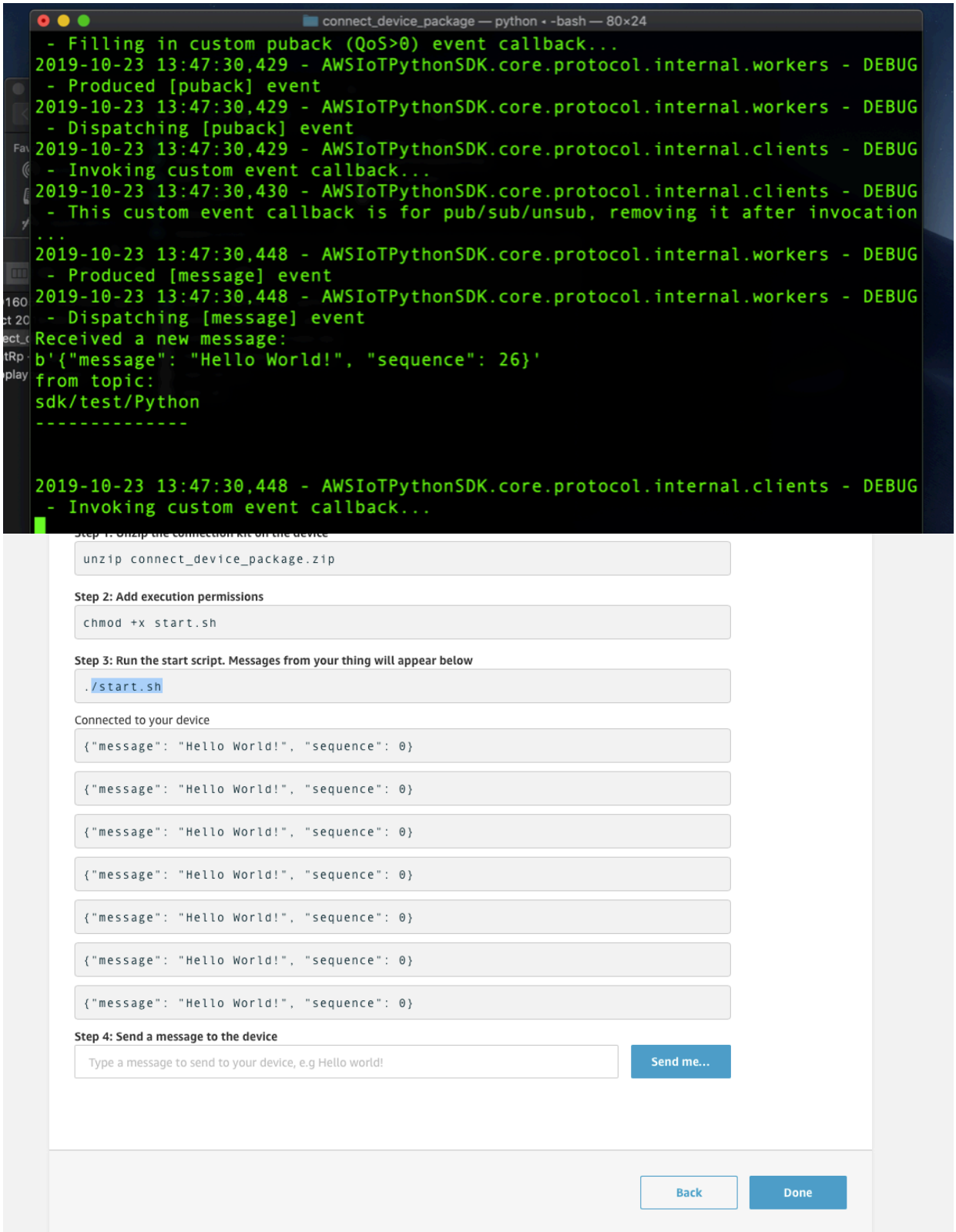
Here are all the certificates that I have just downloaded by clicking on the button. Now we need to run the Shell scripts. Click on the next button

To configure and test the device, perform the following steps.

```
chmod +x start.sh
```

```
/start.sh
```

You should see some message on Terminal

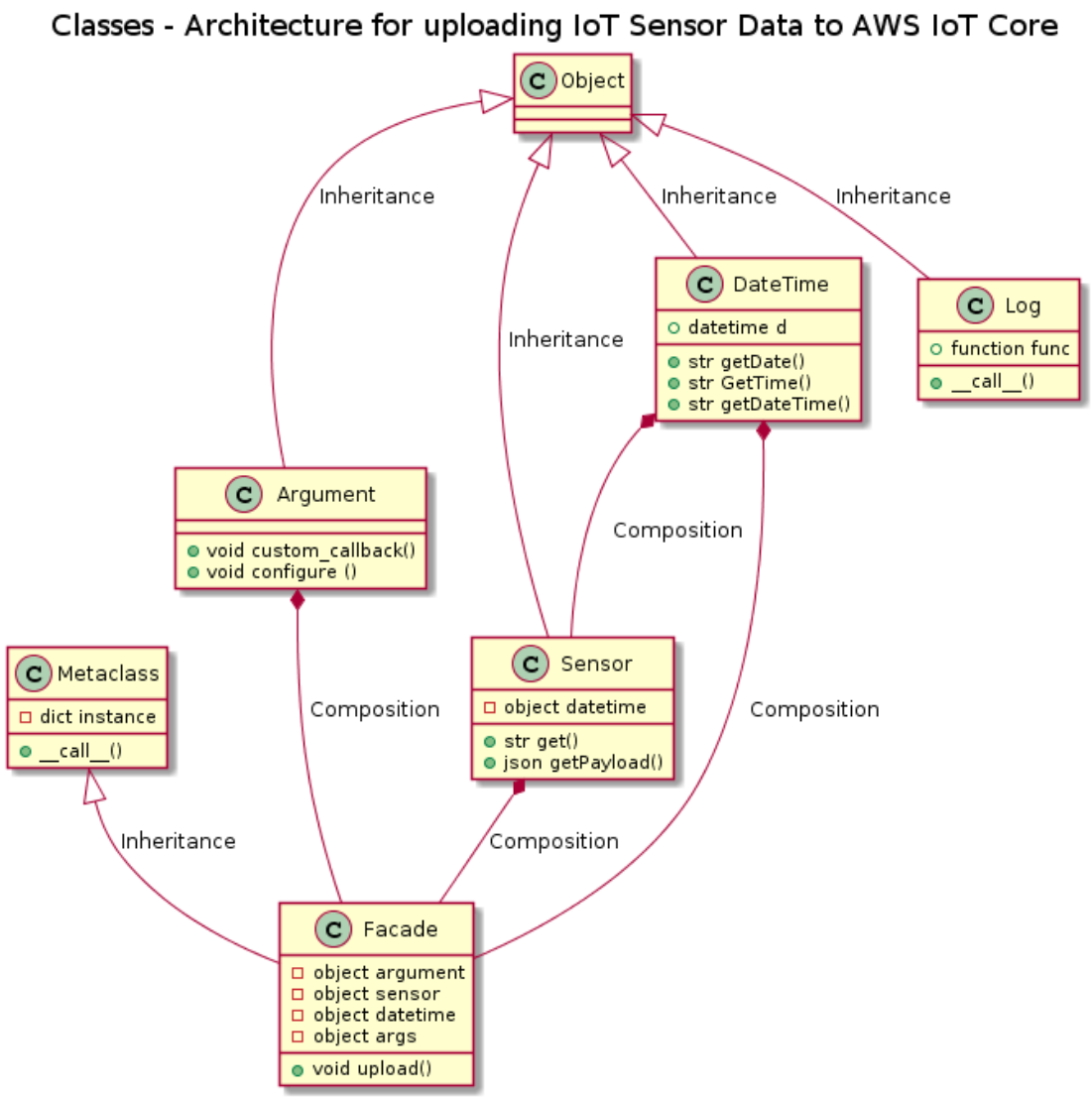

connect_device_package — python • -bash — 80×24
- Filling in custom puback (QoS>0) event callback...
2019-10-23 13:47:30,429 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG
- Produced [puback] event
2019-10-23 13:47:30,429 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG
- Dispatching [puback] event
2019-10-23 13:47:30,429 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG
- Invoking custom event callback...
2019-10-23 13:47:30,430 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG
- This custom event callback is for pub/sub/unsub, removing it after invocation
...
2019-10-23 13:47:30,448 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG
- Produced [message] event
2019-10-23 13:47:30,448 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG
- Dispatching [message] event
Received a new message:
b'{"message": "Hello World!", "sequence": 26}'
from topic:
sdk/test/Python

2019-10-23 13:47:30,448 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG
- Invoking custom event callback...
unzip connect_device_package.zip
Step 2: Add execution permissions
chmod +x start.sh
Step 3: Run the start script. Messages from your thing will appear below
./start.sh
Connected to your device
{"message": "Hello World!", "sequence": 0}
{"message": "Hello World!", "sequence": 0}
{"message": "Hello World!", "sequence": 0}
{"message": "Hello World!", "sequence": 0}
{"message": "Hello World!", "sequence": 0}
{"message": "Hello World!", "sequence": 0}
{"message": "Hello World!", "sequence": 0}
Step 4: Send a message to the device
Type a message to send to your device, e.g Hello world!
Send me...
Back
Done

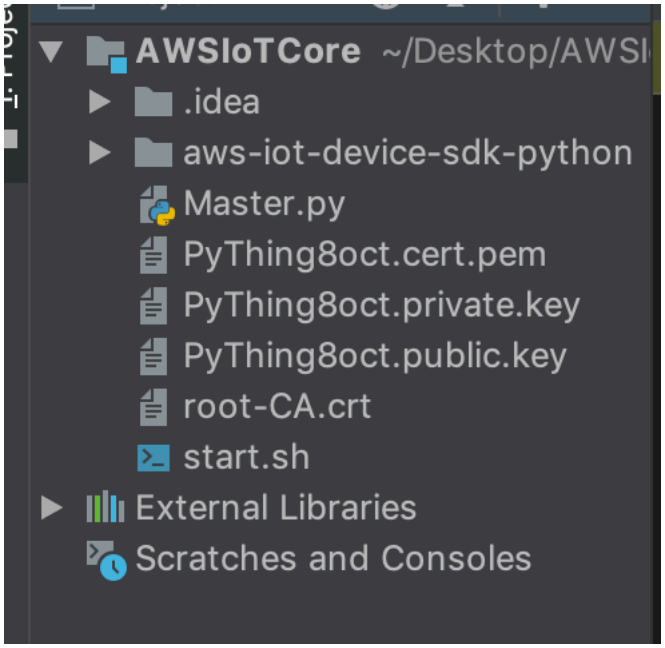
Great

Section II

This is Design Pattern we will Follow



After completing part I we will try Mock the Sensor Behavior and upload the Dummy Values to IoT Core so we can learn how AWS Iot core works.
Following Architecture shown in above Picture.



Create a new python file called as master.py directory should be as follow
Once done to go to the GitHub and download the code [Link] and paste the code into the master.py
You Need to change the start.sh file edit it and add your path to python file and give arguments and the value would be your certificate name whatever may be your certificate name just add that

```
  GNU nano 2.0.6                    File: start.sh

# stop script on error
set -e

# Check to see if root CA file exists, download if not
if [ ! -f ./root-CA.crt ]; then
  printf "\nDownloading AWS IoT Root CA certificate from AWS...\n"
  curl https://www.amazontrust.com/repository/AmazonRootCA1.pem > root-CA.crt
fi

# install AWS Device SDK for Python if not already installed
if [ ! -d ./aws-iot-device-sdk-python ]; then
  printf "\nInstalling AWS SDK...\n"
  git clone https://github.com/aws/aws-iot-device-sdk-python.git
  pushd aws-iot-device-sdk-python
  python setup.py install
  popd
fi

# run pub/sub sample app using certificates downloaded in package
printf "\nRunning pub/sub sample application...\n"
python aws-iot-device-sdk-python/samples/basicPubSub/basicPubSub.py -e a15uvuidocrdo1-ats.iot.us-east-1.am

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

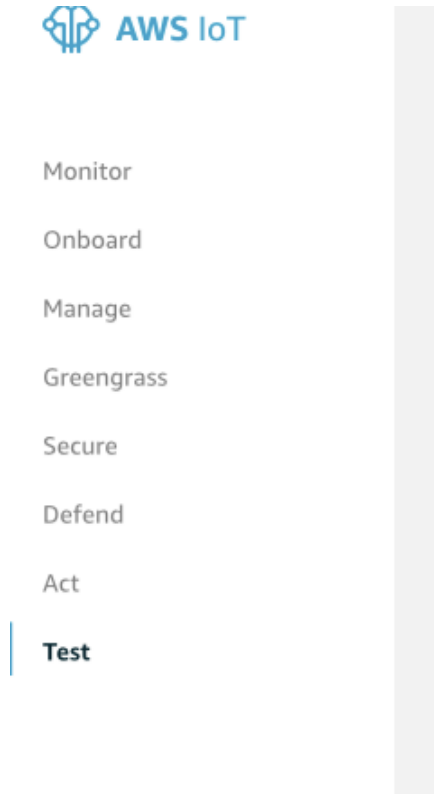
These are the content of my files and now I will just add one line which is path of my python file

Add the following line

```
python /Users/soumilshah/Desktop/AWSIoTCore/Master.py -e a15uvuidocrdo1-ats.iot.us-east-1.amazonaws.com -r root-CA.crt -c PyThing8oct.cert.pem -k PyThing8oct.private.key
```

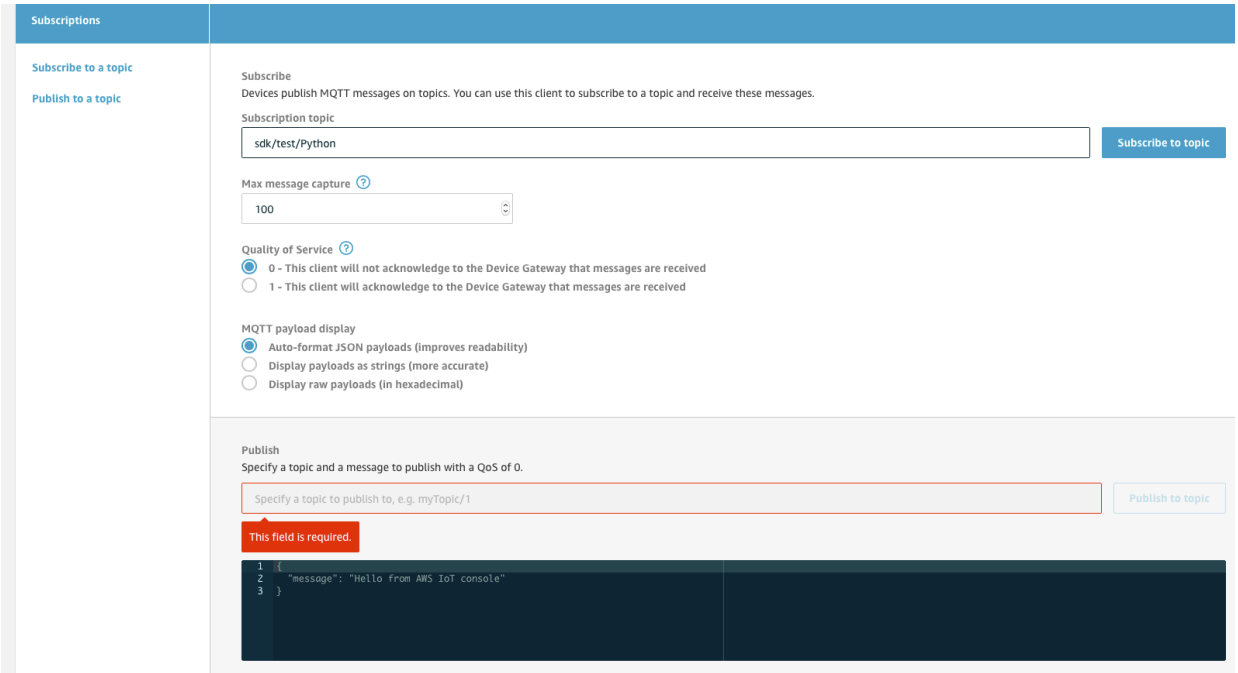
make sure to change the certificates and rootCa name to whatever name of file you have and run the code

go back to AWS IoT core and go to the test section

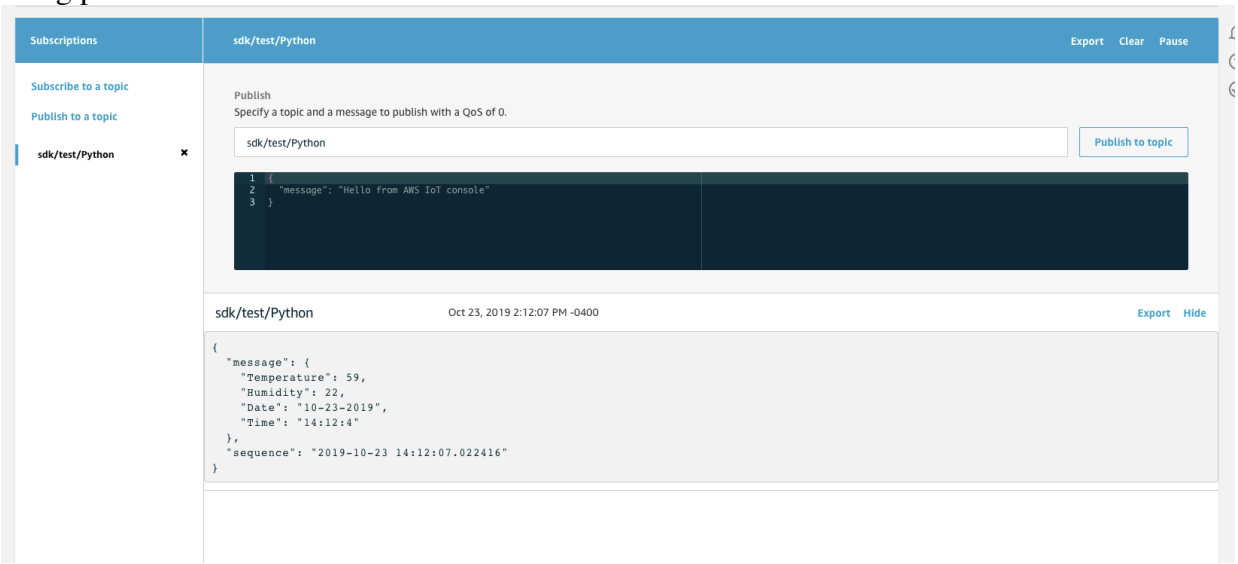


Enter the topic name

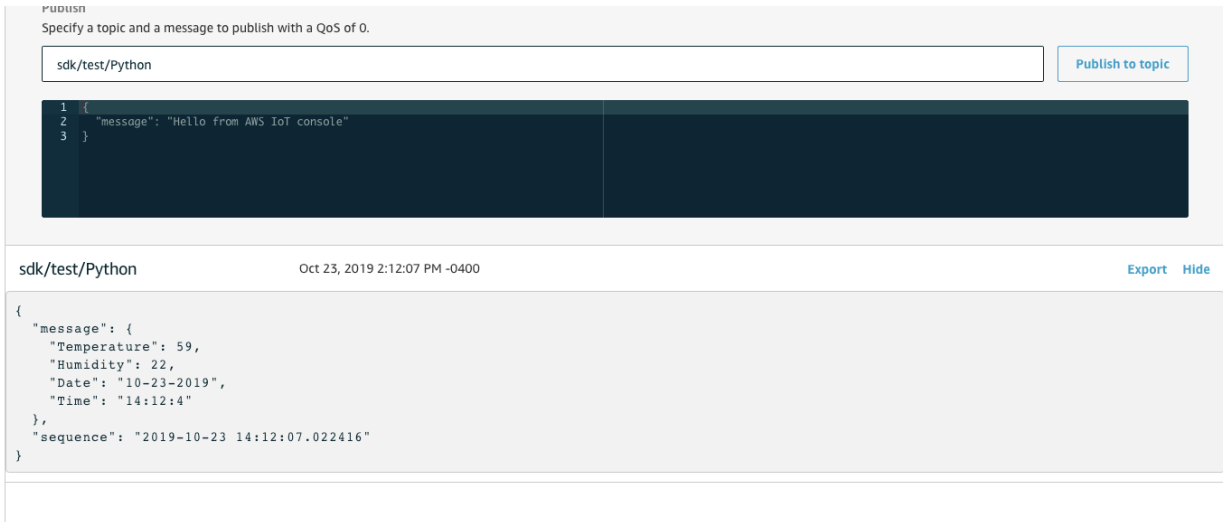
```
sdk/test/Python
```



Once done click on subscribe topic and run the code again you should see MQTT Messages being published



Once you will run the code you will see the Messages being published



Code:

```
__Author__ = "Soumil Nitn Shah"
__Version__ = "0.0.1"
__Email__ = ["soushah@my.bridgeport.edu", "shahsoumil519@gmail.com"]

"""

Hello my Name is Soumil Nitin Shah this is what i want to say about coding
"coding is art "
well i have use Facade and Singleton Design Pattern
Entire  Architecture can be broken down into Severals Objects

    >-----Sensor
    >-----Datetime
    >-----Metaclass
    >-----Argument
    >-----log
    >-----Facade

    Facade is Central Controller

"""

try:

    from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
```

```
    import logging
    import time
    import argparse
    import json
    import datetime
    import random
    import os
    import sys

except Exception as e:
    print("Some modules are misisng {}".format(e))


class MetaClass(type):

    """ Singleton Design Pattern  """

    _instance = {}

    def __call__(cls, *args, **kwargs):

        """ if instance already exist dont create one  """

        if cls not in cls._instance:
            cls._instance[cls] = super(MetaClass, cls).__call__(*args,
**kwargs)
            return cls._instance[cls]


class log(object):

    """ Create a Log File regarding Execution Time memory Address size etc
"""

    def __init__(self, func):
        """ Constructor  """
        self.func = func

    def __call__(self, *args, **kwargs):
        """ Wrapper Function """

        start = datetime.datetime.now()     # start time
        Tem = self.func(self, *args, **kwargs)    # call Function
        FunName = self.func.__name__        # get Function Name
        end = datetime.datetime.now()       # End time

        message = """                       # Form Message

            Function : {}
            Execution Time : {}
            Address : {}
```

```
            Memory: {} Bytes
            Date: {}
            Args: {}
            Kwargs {}

            """.format(FunName,
                       end-start,
                       self.func.__name__,
                       sys.getsizeof(self.func),
                       start, args, kwargs)

        cwd = os.getcwd()                       # get CWD
        folder = 'Logs'                         # Create Folder Logs
        newPath = os.path.join(cwd, folder)     # change Path

        try:
            """ try to create directory """
            os.mkdir(newPath)                   # create Folder
            logging.basicConfig(filename='{}/log.log'.format(newPath),
level=logging.DEBUG)
            logging.debug(message)

        except Exception as e:

            """ Directory already exists """

            logging.basicConfig(filename='{}/log.log'.format(newPath),
level=logging.DEBUG)
            logging.debug(message)

        return Tem


class Argument(object):

    __slots__ = []

    def __init__(self):
        pass

    @staticmethod
    def customCallback(client, userdata, message):
        print("Received a new message: ")
        print(message.payload)
        print("from topic: ")
        print(message.topic)
        print("--------------\n\n")

    @staticmethod
    def configure(topic = "sdk/test/Python" ):
```

```
        """AWS configuration  """

        AllowedActions = ['both', 'publish', 'subscribe']
        # Read in command-line parameters
        parser = argparse.ArgumentParser()
        parser.add_argument("-e", "--endpoint", action="store",
required=True, dest="host", help="Your AWS IoT custom endpoint")
        parser.add_argument("-r", "--rootCA", action="store",
required=True, dest="rootCAPath", help="Root CA file path")
        parser.add_argument("-c", "--cert", action="store",
dest="certificatePath", help="Certificate file path")
        parser.add_argument("-k", "--key", action="store",
dest="privateKeyPath", help="Private key file path")
        parser.add_argument("-p", "--port", action="store", dest="port",
type=int, help="Port number override")
        parser.add_argument("-w", "--websocket", action="store_true",
dest="useWebsocket", default=False,
                            help="Use MQTT over WebSocket")
        parser.add_argument("-id", "--clientId", action="store",
dest="clientId", default="basicPubSub",
                            help="Targeted client id")
        parser.add_argument("-t", "--topic", action="store", dest="topic",
default="sdk/test/Python", help="Targeted topic")
        parser.add_argument("-m", "--mode", action="store", dest="mode",
default="both",
                            help="Operation modes:
%s"%str(AllowedActions))
        parser.add_argument("-M", "--message", action="store",
dest="message", default="Hello World!",
                            help="Message to publish")

        args = parser.parse_args()
        host = args.host
        rootCAPath = args.rootCAPath
        certificatePath = args.certificatePath
        privateKeyPath = args.privateKeyPath
        port = args.port
        useWebsocket = args.useWebsocket
        clientId = args.clientId
        # topic = args.topic
        topic = topic

        if args.mode not in AllowedActions:
            parser.error("Unknown --mode option %s. Must be one of %s" %
(args.mode, str(AllowedActions)))
            exit(2)

        if args.useWebsocket and args.certificatePath and
args.privateKeyPath:
            parser.error("X.509 cert authentication and WebSocket are
mutual exclusive. Please pick one.")
```

```
            exit(2)

        if not args.useWebsocket and (not args.certificatePath or not
args.privateKeyPath):
            parser.error("Missing credentials for authentication.")
            exit(2)

        # Port defaults
        if args.useWebsocket and not args.port:  # When no port override
for WebSocket, default to 443
            port = 443
        if not args.useWebsocket and not args.port:  # When no port
override for non-WebSocket, default to 8883
            port = 8883

        # Configure logging
        logger = logging.getLogger("AWSIoTPythonSDK.core")
        logger.setLevel(logging.DEBUG)
        streamHandler = logging.StreamHandler()
        formatter = logging.Formatter('%(asctime)s - %(name)s -
%(levelname)s - %(message)s')
        streamHandler.setFormatter(formatter)
        logger.addHandler(streamHandler)

        # Init AWSIoTMQTTClient
        myAWSIoTMQTTClient = None
        if useWebsocket:
            myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId,
useWebsocket=True)
            myAWSIoTMQTTClient.configureEndpoint(host, port)
            myAWSIoTMQTTClient.configureCredentials(rootCAPath)
        else:
            myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)
            myAWSIoTMQTTClient.configureEndpoint(host, port)
            myAWSIoTMQTTClient.configureCredentials(rootCAPath,
privateKeyPath, certificatePath)

        # AWSIoTMQTTClient connection configuration
        myAWSIoTMQTTClient.configureAutoReconnectBackoffTime(1, 32, 20)
        myAWSIoTMQTTClient.configureOfflinePublishQueueing(-1)  # Infinite
offline Publish queueing
        myAWSIoTMQTTClient.configureDrainingFrequency(2)  # Draining: 2 Hz
        myAWSIoTMQTTClient.configureConnectDisconnectTimeout(10)  # 10 sec
        myAWSIoTMQTTClient.configureMQTTOperationTimeout(5)  # 5 sec

        # Connect and subscribe to AWS IoT
        myAWSIoTMQTTClient.connect()
        if args.mode == 'both' or args.mode == 'subscribe':
            myAWSIoTMQTTClient.subscribe(topic, 1,
Argument.customCallback)
        time.sleep(2)
```

```
        return args, myAWSIoTMQTTClient, topic


class Sensor(object):

    """Sensor Class for IoT """

    def __init__(self):
        self._datetime = DateTime()

    def get(self):
        Temperature = random.randint(1,89)
        Humidity = random.randint(1,77)
        return Temperature, Humidity

    def getPayload(self):

        Temperature, Humidity = self.get()
        message = {}
        Payload = {}

        Payload["Temperature"] = Temperature
        Payload["Humidity"] = Humidity
        Payload["Date"] = str(self._datetime.getDate())
        Payload["Time"] = str(self._datetime.getTime())
        message['message'] = Payload

        message['sequence'] = str(datetime.datetime.now())
        messageJson = json.dumps(message)

        return messageJson

class DateTime(object):

    """ Datetime Class """

    __slots__ = ["d"]

    def __init__(self):
        self.d = datetime.datetime.now()

    def getDate(self):
        return "{}-{}-{}".format(self.d.month, self.d.day, self.d.year)

    def getTime(self):
        return "{}:{}:{}".format(self.d.hour, self.d.minute,
self.d.second)

    def getDateTime(self):
        return self.d
```

```
class Facade(metaclass=MetaClass):

    __slots__ = ["_argument", "_sensor", "_datetime", "_args",
"_myAWSIoTMQTTClient", "_topic"]

    def __init__(self):

        """ Constructor """

        self._argument = Argument()
        self._sensor = Sensor()
        self._datetime = DateTime()
        self._args, self._myAWSIoTMQTTClient, self._topic =
self._argument.configure()

    def upload(self):

        """ Upload Sensor Data """

        if self._args.mode == 'both' or self._args.mode == 'publish':
            payload = self._sensor.getPayload()
            self._myAWSIoTMQTTClient.publish(self._topic, payload, 1)
            time.sleep(4)


if __name__ == "__main__":
    obj = Facade()
    obj.upload()
```