

Group 13

37 Mikil Lalwani
56 Nilay Pophalkar
58 Sanskruti Punyarthi
61 Shree Samal

CA3 - Assignment 3 : Implement Edge to cloud Protocols (MQTT, HTTP, Websockets) using a dummy data set.

MQTT

Step 1: Search for IOT Core

Step 2: Create a policy add add policy rule with

Policy effect = allow

Policy action = *

Policy Resource = *

The screenshot shows the AWS IoT Policy creation interface. On the left, there's a sidebar with options like Monitor, Connect, Test, and Manage. The main area has a 'Policy name' input field containing 'policy'. Below it is a 'Policy statements' tab. Under the 'Policy document' section, there's a table with columns for 'Policy effect' (set to 'Allow'), 'Policy action' (set to '*'), and 'Policy resource' (set to a wildcard resource). A 'Builder' button is available for creating the policy document. At the bottom right are 'Cancel' and 'Create' buttons.

The screenshot shows the AWS IoT Things interface. The left sidebar includes sections for Test, Manage (with sub-options like All devices, Greengrass devices, LPWAN devices, and Software packages), and Things (with sub-options like Thing groups, Thing types, Fleet metrics, Greengrass devices, LPWAN devices, Software packages, Remote actions, and Message routing). The main area shows a navigation path: AWS IoT > Manage > Things > Create things > Create single thing. It's Step 3 - optional to 'Attach policies to certificate'. A table titled 'Policies (1/1)' lists a single policy named 'policy'. At the bottom right are 'Cancel', 'Previous', and 'Create thing' buttons.

Step 3: Click on create thing and add the thing name

The screenshot shows the 'Create things' step 1 interface. At the top, there's a navigation bar with the AWS logo, 'Services' dropdown, and search bar. Below it, a horizontal menu bar includes Cloud9, IAM, EC2, Elastic Beanstalk, AWS Amplify, Lambda, S3, and API Gateway. On the right, it shows 'N. Virginia' and 'Mikil Lalwani'. The main content area has a title 'Create things' with an 'Info' link. A descriptive text explains that a thing resource is a digital representation of a physical device or logical entity in AWS IoT. Below this, a section titled 'Number of things to create' contains two options: 'Create single thing' (selected) and 'Create many things'. Both options have detailed descriptions. At the bottom right are 'Cancel' and 'Next' buttons.

The screenshot shows the 'Specify thing properties' step 2 interface. At the top, there's a navigation bar with the AWS logo, 'Services' dropdown, and search bar. Below it, a horizontal menu bar includes Cloud9, IAM, EC2, Elastic Beanstalk, AWS Amplify, Lambda, S3, and API Gateway. On the right, it shows 'N. Virginia' and 'Mikil Lalwani'. The main content area has a title 'Specify thing properties' with an 'Info' link. It includes three tabs: 'Step 1 Specify thing properties' (selected), 'Step 2 - optional Configure device certificate', and 'Step 3 - optional Attach policies to certificate'. The 'Step 1' tab has a 'Thing properties' section with a 'Thing name' input field containing 'thing1'. A note says 'Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.' Below this is an 'Additional configurations' section with several optional items: Thing type, Searchable thing attributes, Thing groups, Billing group, and Packages and versions. At the bottom right are 'Cancel' and 'Next' buttons.

Step 4: Click on auto-generate a certificate

The screenshot shows the 'Configure device certificate - optional' step. It includes a note about device requirements and three options: 'Auto-generate a new certificate (recommended)', 'Use my certificate', 'Upload CSR', and 'Skip creating a certificate at this time'. The 'Auto-generate a new certificate' option is selected. Navigation buttons 'Cancel', 'Previous', and 'Next' are at the bottom.

Step 5: Attach policy and click on create thing

The screenshot shows the 'Attach policies to certificate - optional' step. It notes that AWS IoT policies grant or deny access to resources. A 'Policies (0)' section shows a search bar and a 'Create policy' button. Below it, a message says 'No policies' and 'No policies could be found in us-east-1'. Navigation buttons 'Cancel', 'Previous', and 'Create thing' are at the bottom.

This screenshot shows the same 'Attach policies to certificate - optional' step, but with a policy attached. The 'Policies (1/1)' section shows a single policy named 'policy' with a checked checkbox. Navigation buttons 'Cancel', 'Previous', and 'Create thing' are at the bottom.

Step 6: Download all certificates and keys at the same location

The screenshot shows the AWS IoT Things console. At the top, there is a file download dialog with the following contents:

File	Created	Type
private.pem.key	06-10-2023 13:15	KEY File
public.pem.key	06-10-2023 13:15	KEY File
device-certificate.pem.crt	06-10-2023 13:15	Security Certificate
AmazonRootCA1.pem	06-10-2023 13:15	PEM File
AmazonRootCA3.pem	06-10-2023 13:15	PEM File

Below the download dialog, the AWS IoT Things console interface is visible. It shows two success messages in the notification bar:

- You successfully created thing thing1.
- You successfully created certificate 944dafb192f711c4155960f0096f9b89818f11395d8d30d6487feaac758dd214.

The main pane displays the 'Things' list with one item named 'thing1'. The sidebar on the left shows navigation options like 'All devices', 'Things', 'Thing groups', etc.

Step 7: Clone AWS-MQTT-SDK git repo using the below link in the same folder where you have downloaded the keys and certificate

```
-git clone https://github.com/aws/aws-iot-device-sdk-python.git
```

Step 8: After cloning Run below commands

```
-cd aws-iot-device-sdk-python  
-python setup.py install (2)
```

Step 9: In main directory save two codes one to publish message to MQTT server and one to subscribe

Publish.py

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient  
import sys
```

```
myMQTTClient = AWSIoTMQTTClient("thing1") #Enter your things name
```

```
myMQTTClient.configureEndpoint("a1ek9vqv3jnbud-ats.iot.us-east-1.amazonaws.com", 8883)  
myMQTTClient.configureCredentials("./AmazonRootCA1.pem","./private.pem.key",  
"./device-certificate.pem.crt")
```

```
myMQTTClient.connect()
```

```
print("Client Connected")

msg = "Sample data from the device"
topic = "general/inbound"
myMQTTClient.publish(topic, msg, 0)
print("Message Sent")

myMQTTClient.disconnect()
print("Client Disconnected")
```

Subscribe.py

```
import time

def customCallback(client,userdata,message):
    print("callback came... ")
    print(message.payload)

from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

myMQTTClient = AWSIoTMQTTClient("thing1")
myMQTTClient.configureEndpoint("a1ek9vqv3jnbud-ats.iot.us-east-1.amazonaws.com", 8883)
myMQTTClient.configureCredentials("./AmazonRootCA1.pem","./private.pem.key",
                                 "./device-certificate.pem.crt")

myMQTTClient.connect()
print("Client Connected")

myMQTTClient.subscribe("general/outbound", 1, customCallback)
print('waiting for the callback. Click to conntinue... ')
x = input()

myMQTTClient.unsubscribe("general/outbound")
print("Client unsubscribed")

myMQTTClient.disconnect()
```

```
print("Client Disconnected")
```

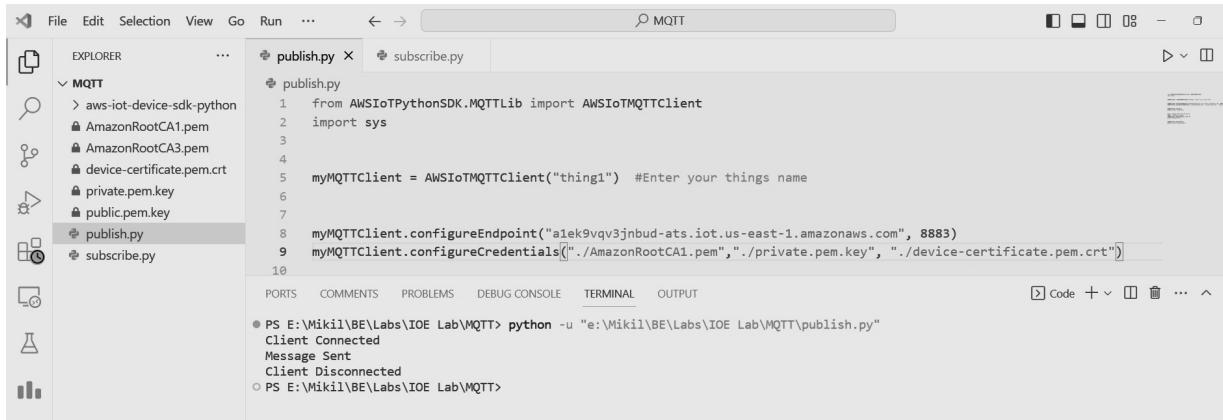
Step 10: In both of above codes replace endpoint with your MQTT test client endpoint

The screenshot shows the 'MQTT test client' interface. At the top, there's a header with the title 'MQTT test client' and a 'Info' link. Below the header, a message states: 'You can use the MQTT test client to monitor their state to AWS IoT. AWS IoT also publishes messages to topics by using the MQTT test cli'. Under the message, there's a section titled 'Connection details' with a subtitle 'You can update the connection details by choosing the connection profile'. The 'Client ID' is listed as 'iotconsole-5d484cb2-d2b5-424f-b52f-ec5dee87d535'. The 'Endpoint' is listed as 'a1ek9vqv3jnbud-ats.iot.us-east-1.amazonaws.com'.

Step 11: In MQTT go to Subscribe to a topic and type ‘general/inbound’ and click subscribe don’t close this page

The screenshot shows the 'AWS IoT' service in the AWS console. On the left, there's a sidebar with options like 'Monitor', 'Connect', 'Test', and 'MQTT test client' (which is currently selected). The main area is titled 'Subscribe to a topic' and has a 'Topic filter' input field containing 'general/inbound'. Below the input field is a 'Subscribe' button. At the bottom, there's a table with two columns: 'Subscriptions' and 'Topic'. A note says 'You have no topic subscriptions.' and 'Subscribe or select a topic to view incoming messages.'

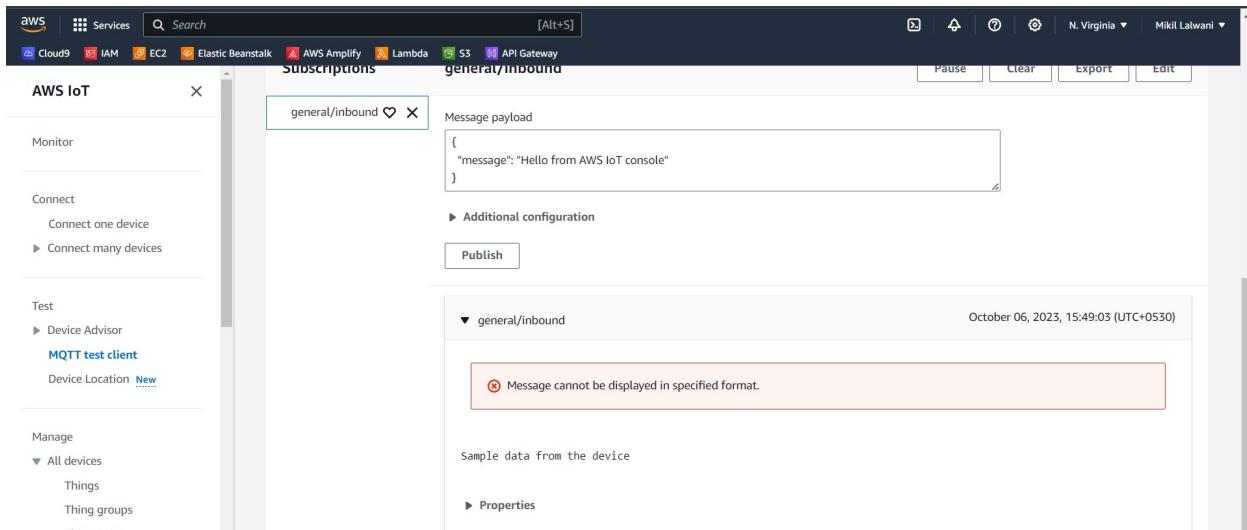
Step 12: After that run publish.py and go to subscribe page



The screenshot shows a code editor with an 'EXPLORER' sidebar containing files: aws-iot-device-sdk-python, publish.py, and subscribe.py. The 'publish.py' tab is active, displaying Python code for AWS IoT. The terminal below shows the output of running the script:

```
PS E:\Mikil\BE\Lab\IOE Lab\MQTT> python -u "e:\Mikil\BE\Lab\IOE Lab\MQTT\publish.py"
Client Connected
Message Sent
Client Disconnected
PS E:\Mikil\BE\Lab\IOE Lab\MQTT>
```

Step 13: You can see message on subscribe page

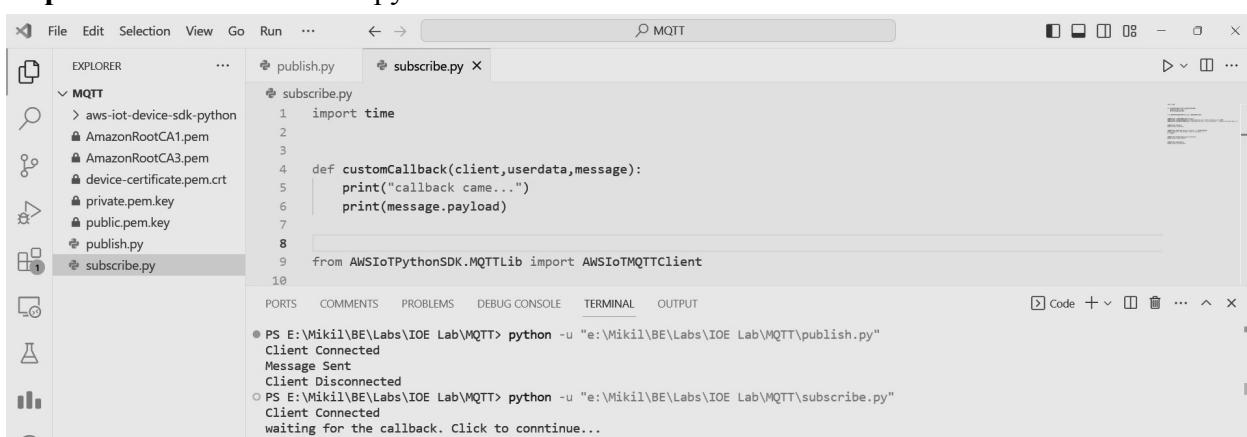


The screenshot shows the AWS IoT Subscriptions page. A message titled 'general/inbound' is displayed with the payload:

```
{
  "message": "Hello from AWS IoT console"
}
```

The message status is shown as 'Message cannot be displayed in specified format.'

Step 14: Then run subscribe.py



The screenshot shows a code editor with an 'EXPLORER' sidebar containing files: aws-iot-device-sdk-python, publish.py, and subscribe.py. The 'subscribe.py' tab is active, displaying Python code for AWS IoT. The terminal below shows the output of running the script:

```
PS E:\Mikil\BE\Lab\IOE Lab\MQTT> python -u "e:\Mikil\BE\Lab\IOE Lab\MQTT\publish.py"
Client Connected
Message Sent
Client Disconnected
PS E:\Mikil\BE\Lab\IOE Lab\MQTT> python -u "e:\Mikil\BE\Lab\IOE Lab\MQTT\subscribe.py"
Client Connected
waiting for the callback. Click to continue...
```

Step 15: Go to publish to topic and type general/outbound and click on publish

The screenshot shows the AWS IoT Publish interface. On the left sidebar, under 'Test', 'MQTT test client' is selected. In the main area, the 'Publish to a topic' tab is active. The 'Topic name' field contains 'general/outbound'. The 'Message payload' field contains the JSON message: { "message": "Hello from AWS IoT console" }. Below the message payload is an 'Additional configuration' section with a 'Publish' button. At the bottom, there is a 'Subscriptions' section for 'general/inbound' with buttons for 'Pause', 'Clear', 'Export', and 'Edit'.

Step 16: After clicking on publish go to terminal and you can see the message

The screenshot shows a terminal window with the title bar 'MQTT'. The terminal displays the following Python code and its execution:

```
publish.py
import time
def customCallback(client, userdata, message):
    print("callback came...")
    print(message.payload)
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
PORTS COMMENTS PROBLEMS DEBUG CONSOLE TERMINAL OUTPUT
PS E:\Mikil\BE\Labs\IOE Lab\MQTT> python -u "e:\Mikil\BE\Labs\IOE Lab\MQTT\publish.py"
Client Connected
Message Sent
Client Disconnected
PS E:\Mikil\BE\Labs\IOE Lab\MQTT> python -u "e:\Mikil\BE\Labs\IOE Lab\MQTT\subscribe.py"
Client Connected
waiting for the callback. Click to continue...
callback came...
b'\n "message": "Hello from AWS IoT console"\n'
```

The screenshot shows the AWS IoT Subscribe interface. Under 'Test', 'MQTT test client' is selected. In the main area, the 'Subscribe' tab is active. The 'Topic filter' field contains 'general/inbound'. Below it is an 'Additional configuration' section with a 'Subscribe' button. At the bottom, there is a 'Subscriptions' section for 'general/inbound' with a 'Message payload' field containing the received message: { "message": "Hello from AWS IoT console" }, a 'Publish' button, and a timestamp 'October 06, 2023, 16:31:10 (UTC+0530)'.

HTTP:

Code

```
import requests  
import argparse
```

```
# define command-line parameters  
parser = argparse.ArgumentParser(description="Send messages through an HTTPS connection.")  
parser.add_argument('--endpoint', required=True, help="Your AWS IoT data custom endpoint,  
not including a port. " + "Ex: \"abcdEXAMPLExyz-ats.iot.us-east-1.amazonaws.com\"")  
parser.add_argument('--cert', required=True, help="File path to your client certificate, in PEM  
format.")  
parser.add_argument('--key', required=True, help="File path to your private key, in PEM  
format.")  
parser.add_argument('--topic', required=True, default="test/topic", help="Topic to publish  
messages to.")  
parser.add_argument('--message', default="Hello World!", help="Message to publish. " +  
"Specify empty string to publish nothing.")  
  
# parse and load command-line parameter values  
args = parser.parse_args()  
  
# create and format values for HTTPS request  
publish_url = 'https://' + args.endpoint + ':8443/topics/' + args.topic + '?qos=1'  
publish_msg = args.message.encode('utf-8')  
  
# make request  
publish = requests.request('POST',  
    publish_url,  
    data=publish_msg,  
    cert=[args.cert, args.key])  
  
# print results  
print("Response status: ", str(publish.status_code))  
if publish.status_code == 200:
```

```
print("Response body:", publish.text)
```

Step 1: Change your endpoint and device certificate and private key file name in the below code
`python http_code.py --endpoint "a1ek9vqv3jnbud-ats.iot.us-east-1.amazonaws.com" --cert
"./device-certificate.pem.crt" --key "./private.pem.key" --topic "test/topic" --message
"Test37"`

The screenshot shows a code editor interface with several tabs open. The current tab is `http_code.py`, which contains the following code:

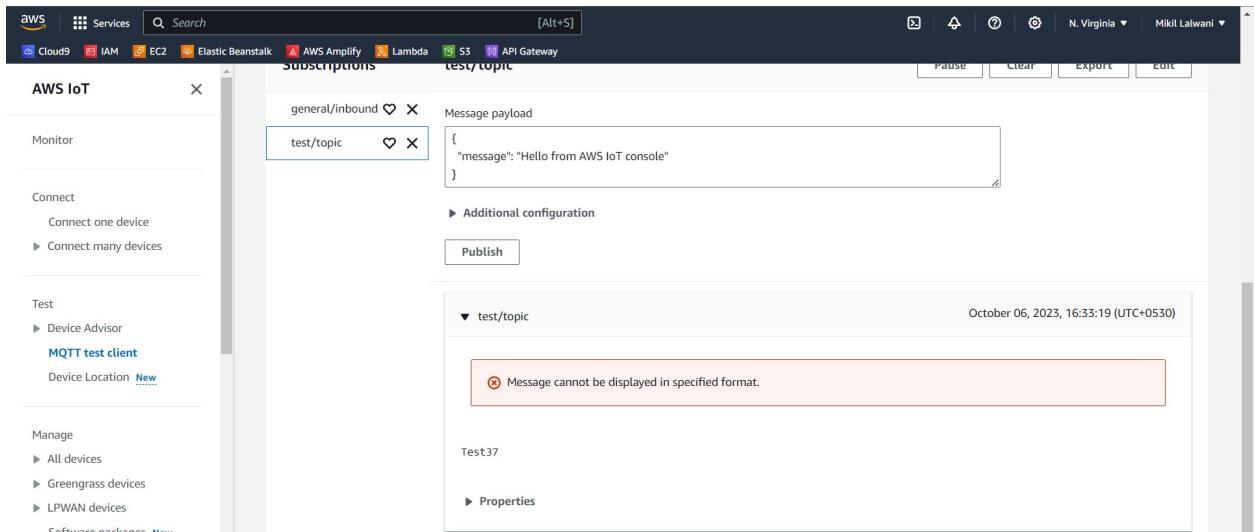
```
publish.py subscribe.py http_code.py X

14
15
16 # parse and load command-line parameter values
17 args = parser.parse_args()
18
19
20 # create and format values for HTTPS request
21 publish_url = 'https://' + args.endpoint + ':8443/topics/' + args.topic + '?qos=1'
22 publish_msg = args.message.encode('utf-8')
23
24
```

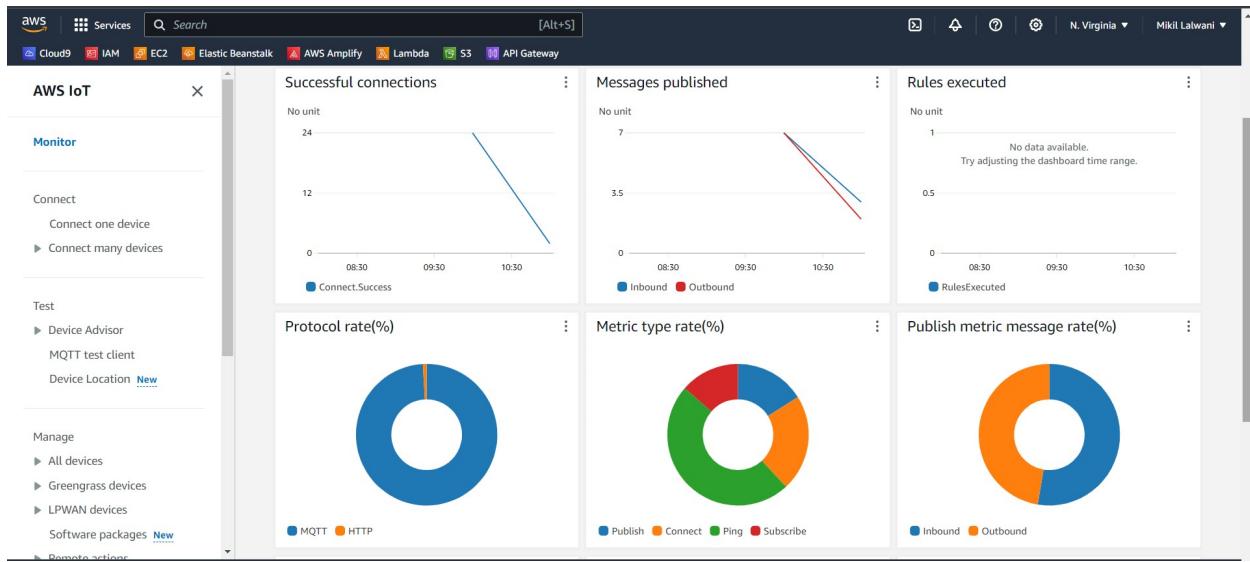
Below the code editor, there are tabs for `POROS`, `COMMENTS`, `PROBLEMS`, `DEBUG CONSOLE`, `TERMINAL`, and `OUTPUT`. The `TERMINAL` tab is active, displaying the following command and its output:

```
PS E:\Mikil\BE\Lab\IOE Lab\MQTT> python http_code.py --endpoint "a1ek9vqv3jnbd-ats.iot.us-east-1.amazonaws.com" --cert "./device-certificate.pem.crt" --key "./private.pem.key" --topic "test/topic" --message "Test37"
Response status: 200
Response body: {"message": "OK", "traceId": "ecd885d9-3329-6123-7f06-3e06f870c9e9"}
```

Step 2: Check message received by subscribing to the topic

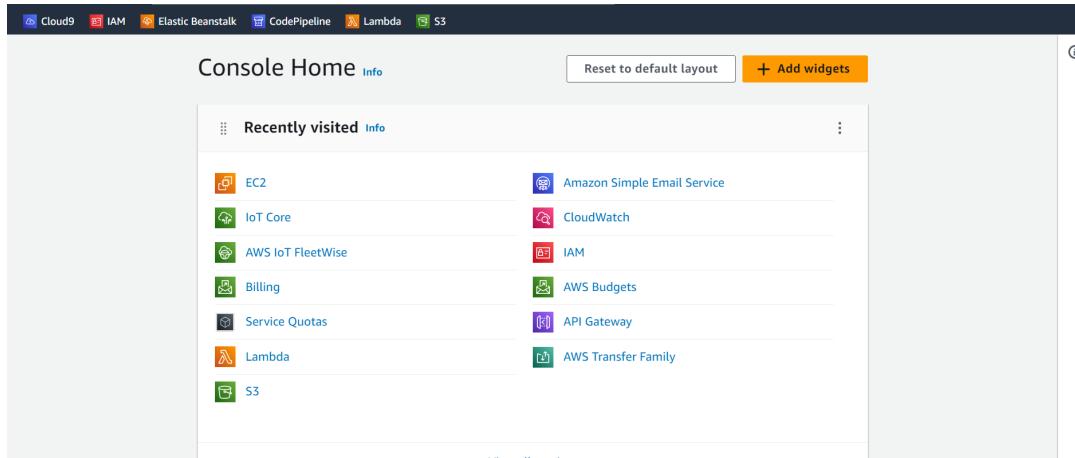


Step 3: On dashboard you can see both protocols has been implemented successfully



WebSockets:

Step 1: Go to the EC2 Console and create an EC2 instance.

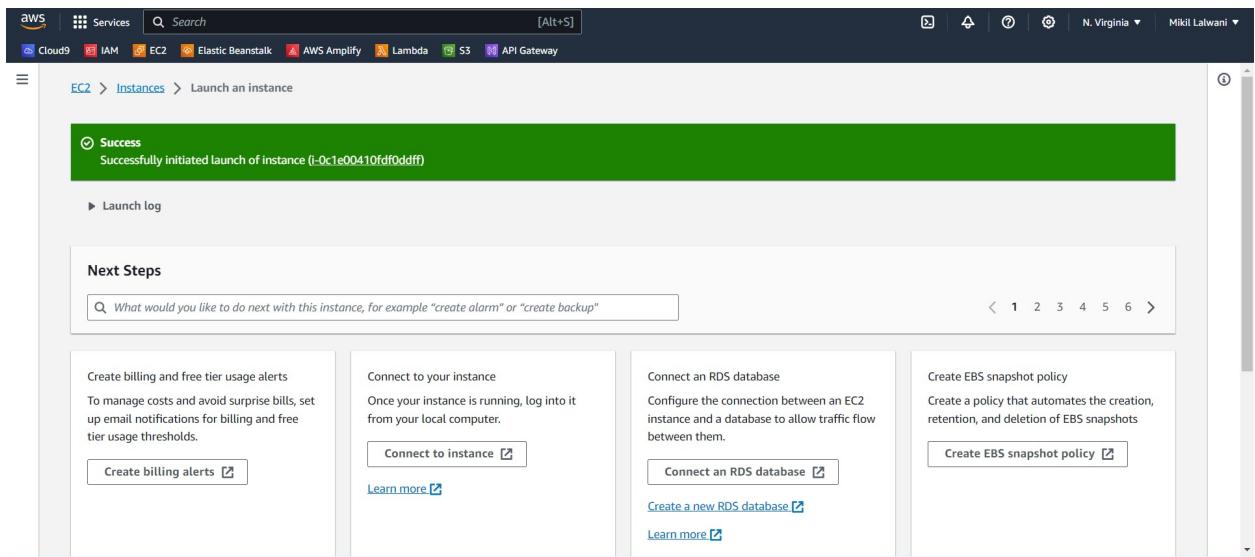
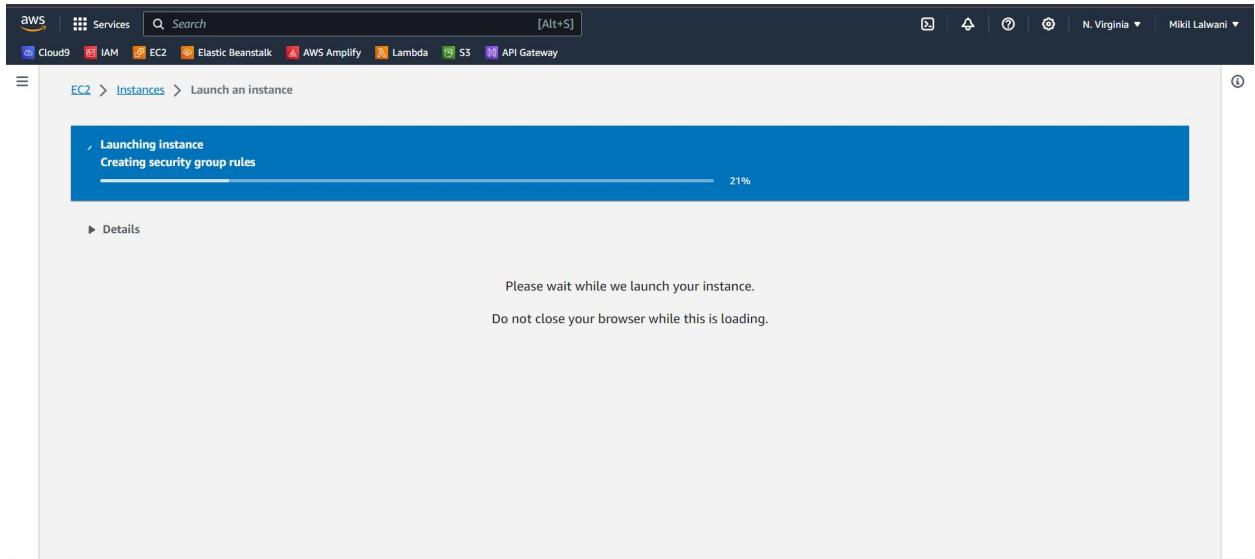


The screenshot shows the AWS EC2 Dashboard. On the left, there's a sidebar with navigation links like EC2 Dashboard, Instances, Images, and Elastic Block Store. The main area has sections for Resources (listing Instances (running), Auto Scaling Groups, Dedicated Hosts, etc.), Launch instance (with a prominent orange 'Launch instance' button), Service health (AWS Health Dashboard), and Account attributes (Default VPC set to vpc-06f870b50074eb753). An 'Explore AWS' sidebar on the right offers performance improvements and cost savings tips.

Step 2: An instance named websocket is created.

This screenshot shows the 'Create New Instance' wizard. In the 'Name and tags' step, the instance is named 'websocket1'. The 'Application and OS Images (Amazon Machine Image)' step lists various AMI categories. A tooltip indicates a free tier of 750 hours for the t2.micro instance type. The final step shows summary details: 1 instance, Canonical Ubuntu 22.04 LTS AMI, t2.micro instance type, and a new security group. The 'Launch instance' button is highlighted in orange.

Step 3: Launch the instance.



Step 4: Check the inbound rules of the instance and edit the inbound rules to add the rule for All traffic.

The screenshot shows the AWS EC2 Instances page. A single instance named "websocket1" is listed, running in the t2.micro instance type. The instance has a Public IPv4 address of 3.90.223.147 and a Private IPv4 address of 172.31.38.80.

The screenshot shows the AWS Security Groups page under the "Edit inbound rules" section. It lists two rules:

- Security group rule ID: sgr-0e471b2b5fa7bb4a7, Type: SSH, Protocol: TCP, Port range: 22, Source: 0.0.0.0/0, Description: optional.
- Security group rule ID: -, Type: All traffic, Protocol: All, Port range: All, Source: Anywhere, Description: optional.

A warning message at the bottom states: "⚠ Rules with source of 0.0.0.0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only."

Step 5: Open Ec2 instance and run the following commands:-

```
sudo su
sudo apt-get update
sudo apt-get install python3-pip
sudo apt-get install python3-venv
```

Create an environment:

```
mkdir ioe
cd ioe
python3 -m venv env
```

```
source env/bin/activate  
pip install websockets
```

```
sudo nano websocket_cloud.py
```

Step 6: Add this following code in the websocket_cloud.py.



```
GNU nano 6.2  
import asyncio  
import websockets  
  
async def receive_data(websocket, path):  
    async for message in websocket:  
        print(f"Received: {message}")  
  
if __name__ == '__main__':  
    start_server = websockets.serve(receive_data, '0.0.0.0', 8080)  
  
    asyncio.get_event_loop().run_until_complete(start_server)  
    asyncio.get_event_loop().run_forever()
```

Step 7: Add this following code in the edge_websocket.py.

```

edge_websocket.py X
edge_websocket.py > send_data
1 import asyncio
2 import websockets
3 import random
4 import json
5 import pandas as pd
6 data = pd.read_csv('data.csv')
7
8 async def send_data():
9     async with websockets.connect('ws://3.90.223.147:8080') as websocket:
10         while True:
11             random_index = random.randint(0, len(data)-1)
12             random_data = data.iloc[random_index]
13             data_dict = {
14                 "id": int(random_data["id"]),
15                 "name": random_data["name"]
16             }
17
18             await websocket.send(json.dumps(data_dict))
19             print(f"Sent: {data_dict}")
20             await asyncio.sleep(1)
21
22 if __name__ == '__main__':
23     asyncio.run(send_data())

```

Step 8: Run the file websockety_cloud.py on the EC2 instance. The output is:



The screenshot shows the AWS CloudWatch Log Stream interface. The log stream is titled 'websockety_cloud' and shows the execution of the script. The log output includes:

```

AWS Services Search [Alt+S] N. Virginia ▾ Mikel Lalwani ▾
Cloud9 IAM EC2 Elastic Beanstalk AWS Amplify Lambda S3 API Gateway
^X^X^XTraceback (most recent call last):
File "/home/ubuntu/ice/websockety_cloud.py", line 12, in <module>
    asyncio.get_event_loop().run_forever()
File "/usr/lib/python3.10/asyncio/base_events.py", line 603, in run_forever
    self._run_once()
File "/usr/lib/python3.10/asyncio/base_events.py", line 1871, in _run_once
    event_list = self._selector.select(timeout)
File "/usr/lib/python3.10/selectors.py", line 469, in select
    fd_event_list = self._selector.poll(timeout, max_ev)
KeyboardInterrupt

(env) root@ip-172-31-38-80:/home/ubuntu/ice# sudo nano websockety_cloud.py
(env) root@ip-172-31-38-80:/home/ubuntu/ice# python websockety_cloud.py
Received: {"id": 2, "name": "Nilay"}
Received: {"id": 4, "name": "Sanskriti"}
Received: {"id": 4, "name": "Sanskriti"}
Received: {"id": 6, "name": "Rameesh"}
Received: {"id": 9, "name": "Tiger"}
Received: {"id": 2, "name": "Nilay"}
Received: {"id": 5, "name": "Raj"}
Received: {"id": 5, "name": "Raj"}
Received: {"id": 2, "name": "Nilay"}
Received: {"id": 7, "name": "Suresh"}
Received: {"id": 2, "name": "Nilay"}
```

Step 9: Run the file of VSCode in the terminal. The output is:

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** Contains icons for file operations like Open, Save, New, and Help.
- Title Bar:** Shows the file name "edge_websocket.py" and a close button.
- Code Cell:** Displays the Python code for "send_data".
- Output Cell:** Shows the command run in the terminal and its output.
- Bottom Navigation:** Includes tabs for PORTS, COMMENTS, PROBLEMS, DEBUG CONSOLE, TERMINAL, and OUTPUT, along with a toolbar with icons for Code, Run, Stop, and More.

```
edge_websocket.py
edge_websocket.py > send_data
1 import asyncio
2 import websockets
3 import random
4 import json
5 import pandas as pd
6 data = pd.read_csv('data.csv')
7
8 async def send_data():
9     async with websockets.connect('ws://3.90.223.147:8080') as websocket:
10         while True:
11             random_index = random.randint(0,len(data)-1)

PORTS COMMENTS PROBLEMS DEBUG CONSOLE TERMINAL OUTPUT
```

PS E:\Mikil\BE\Labs\IOE Lab\CA3> python -u "e:\Mikil\BE\Labs\IOE Lab\CA3\edge_websocket.py"
Sent: {"id": 2, "name": "Nilay"}
Sent: {"id": 4, "name": "Sanskriti"}
Sent: {"id": 4, "name": "Sanskriti"}
Sent: {"id": 6, "name": "Ramesh"}
Sent: {"id": 9, "name": "Tiger"}
Sent: {"id": 2, "name": "Nilay"}
Sent: {"id": 5, "name": "Raj"}
Sent: {"id": 5, "name": "Raj"}