Vivekanand Education Society's

Institute of Technology

(Affiliated to University of Mumbai, Approved by AICTE & Recognized by Govt. of Maharashtra)

Department of Information Technology

IOE Lab

CA Assignment - 3

Aim: **Implement Edge to cloud Protocols (Minimum 3) using a dummy data set.**

Roll No.	70
Name	MAYURI SHRIDATTA YERANDE
Class	D20B
Subject	Internet of Everything
Grade:	

AIM: Implement Edge to cloud Protocols using a dummy data set.

TO-DO:

- Explain in brief each of the protocols
- Code
- Output.

THEORY:

MQTT:

Message Queuing Telemetry Transport (MQTT) is a lightweight messaging protocol designed for use in situations where low bandwidth, high latency, or unreliable networks are common. It was developed by IBM in the late 1990s but has since become an open standard with a wide range of implementations and use cases. MQTT is particularly popular in the Internet of Things (IoT) and machine-to-machine (M2M) communication due to its efficiency and flexibility.

Key features of MQTT:

1. **Publish/Subscribe Model:** MQTT follows a publish/subscribe messaging pattern. In this model, there are two main entities: publishers and subscribers. Publishers send messages (also known as "publish" messages) to specific topics, and subscribers express interest in specific topics by subscribing to them. When a message is published to a topic, all subscribers interested in that topic receive the message.

2. **Quality of Service (QoS) Levels:** MQTT supports three different levels of message delivery assurance, which allows you to choose the level of reliability you need:

   QoS 0 (At most once): Messages are delivered with no confirmation. This is the fastest but least reliable option.
   QoS 1 (At least once): Messages are guaranteed to be delivered at least once to the receiver. Some duplicate messages may occur.
   QoS 2 (Exactly once): Messages are guaranteed to be delivered exactly once. This is the most reliable but comes with higher overhead.

3. **Broker:** MQTT uses a central message broker as an intermediary between publishers and subscribers. The broker is responsible for routing messages from publishers to the appropriate subscribers based on topic subscriptions. Popular MQTT brokers include Mosquitto, HiveMQ, and AWS IoT Core.

4. **Topics:** Topics are hierarchical strings used to categorize messages. Subscribers can express interest in specific topics using wildcard characters. For example, a subscriber could subscribe to "sensors/temperature" to receive temperature data from various sensors or "sensors/+" to receive data from all sensors.

5. **Last Will and Testament (LWT):** Clients can specify a "last will" message to be published by the broker in case the client unexpectedly disconnects. This is useful for handling client failures gracefully.

6. **Retained Messages:** MQTT allows the broker to retain the last message sent on a specific topic. When a new subscriber subscribes to a topic with a retained message, it immediately receives the last retained message.

HTTPS:

Using HTTPS (Hypertext Transfer Protocol Secure) for IoT (Internet of Things) devices and applications is a crucial security practice to protect data and communication in IoT ecosystems. HTTPS is the secure version of HTTP, and it employs encryption, authentication, and data integrity mechanisms to ensure that data transmitted between IoT devices, servers, and services remains confidential and tamper-proof.

Key features of HTTPS:

1. **Data Encryption:** HTTPS uses Transport Layer Security (TLS) or its predecessor, Secure Sockets Layer (SSL), to encrypt data during transmission. This encryption ensures that sensitive information exchanged between IoT devices and servers cannot be easily intercepted or deciphered by unauthorized parties.

2. **Authentication:** TLS in HTTPS provides a means for mutual authentication between the IoT device and the server. This helps in verifying the identity of both parties involved in the communication. Device authentication ensures that only authorized devices can access specific services or data, and server authentication ensures that devices are communicating with legitimate servers.

3. **Data Integrity:** HTTPS also ensures data integrity by employing cryptographic hashes to verify that the data received has not been tampered with during transmission. This helps prevent "man-in-the-middle" attacks where an attacker might modify data packets in transit.

4. **Privacy:** Using HTTPS enhances user privacy by preventing eavesdroppers from monitoring the data being transmitted between IoT devices and servers. This is especially important when dealing with personal or sensitive data.

5. **Secure APIs:** Many IoT ecosystems involve web-based APIs for data exchange and control. Securing these APIs with HTTPS is essential to prevent unauthorized access and data exposure.

6. **End-to-End Security:** HTTPS provides end-to-end security, which means that data is protected from the IoT device through the network to the server and back. This holistic approach to security is crucial in ensuring the overall security of the IoT system.

WEBSOCKET:

WebSocket API is a communication protocol that enables full-duplex, bidirectional data transfer between a client and server over a single, long-lived connection. It's commonly used for real-time applications such as chat, gaming, and live updates.

Key Features:

1. **Low Latency:** Provides low-latency, real-time communication by maintaining an open connection, reducing the overhead of repeatedly establishing connections.

2. **Bi-directional Communication:** Allows both the client and server to initiate data exchanges, facilitating real-time interactions.

3. **Scalability:** Supports horizontal scaling and is well-suited for handling a large number of concurrent connections efficiently.

4. **Efficient for Real-time Applications:** It's ideal for real-time applications like online gaming, instant messaging, live streaming, and collaborative tools where low latency is crucial.

5. **Reduced Overhead:** WebSocket eliminates the need for repeatedly opening and closing connections, reducing the overhead associated with traditional HTTP requests.

6. **Two-way Communication:** Enables full two-way communication between the client and server, making it easy to push updates and notifications to clients.

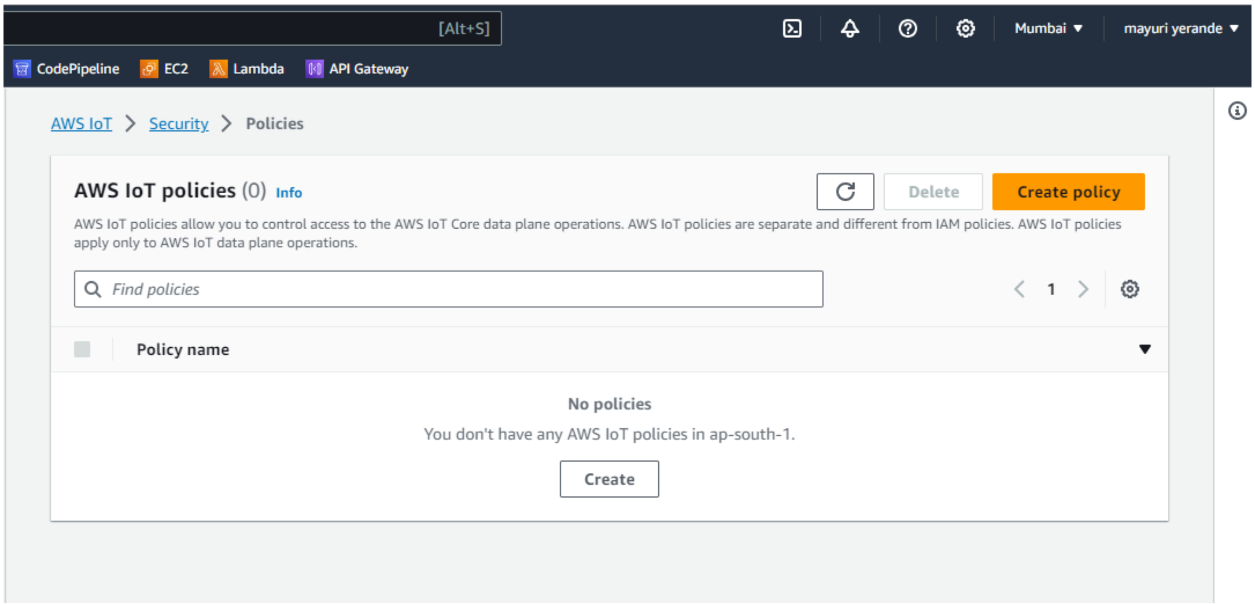
7. **Bi-directional Flow:** Supports simultaneous data transmission in both directions, allowing clients and servers to send messages independently.

IMPLEMENTATION:

❖ MQTT

Step 1: Search for IOT Core

Step 2: Create a policy add add policy rule

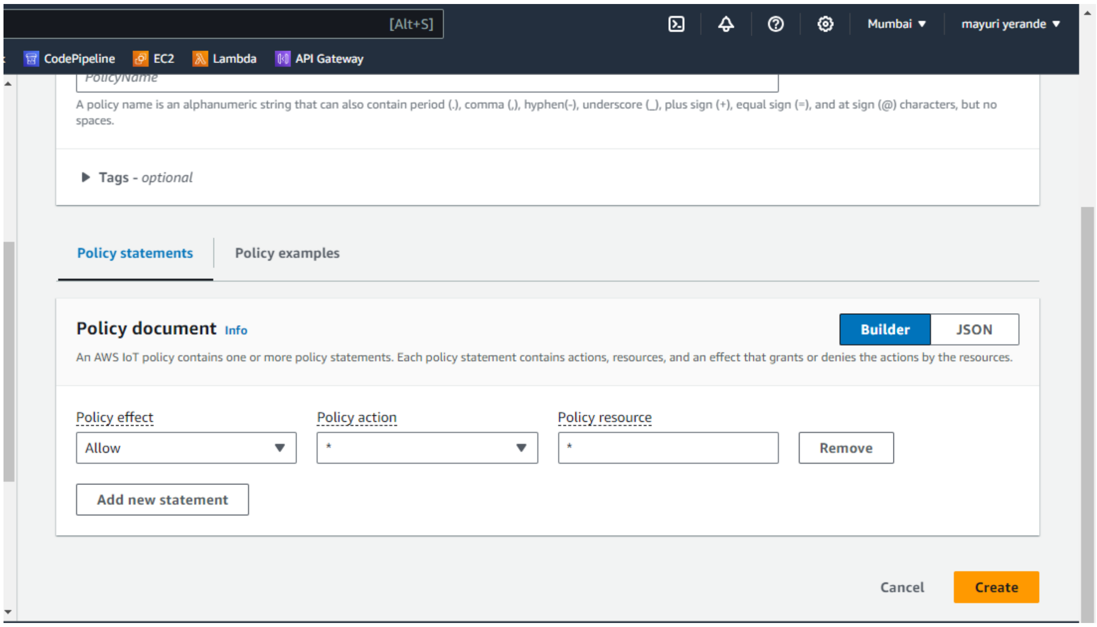


AWS IoT policies (0) Info

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations. AWS IoT policies are separate and different from IAM policies. AWS IoT policies apply only to AWS IoT data plane operations.

No policies

You don't have any AWS IoT policies in ap-south-1.



A policy name is an alphanumeric string that can also contain period (.), comma (,), hyphen(-), underscore (_), plus sign (+), equal sign (=), and at sign (@) characters, but no spaces.

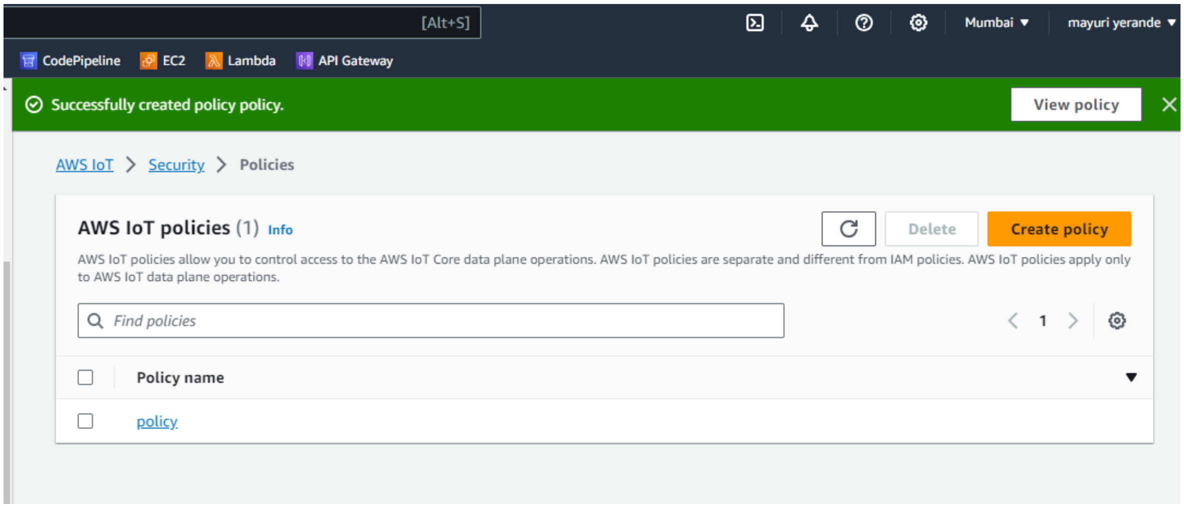
Policy document Info

An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy effect: Allow

Policy action: *

Policy resource: *



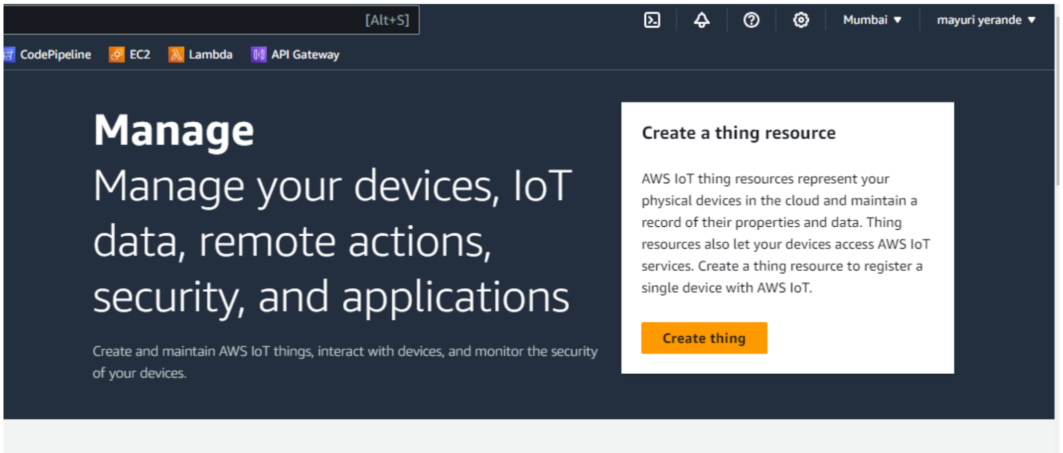
Successfully created policy policy.

AWS IoT policies (1)

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations. AWS IoT policies are separate and different from IAM policies. AWS IoT policies apply only to AWS IoT data plane operations.

□	Policy name
□	policy

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



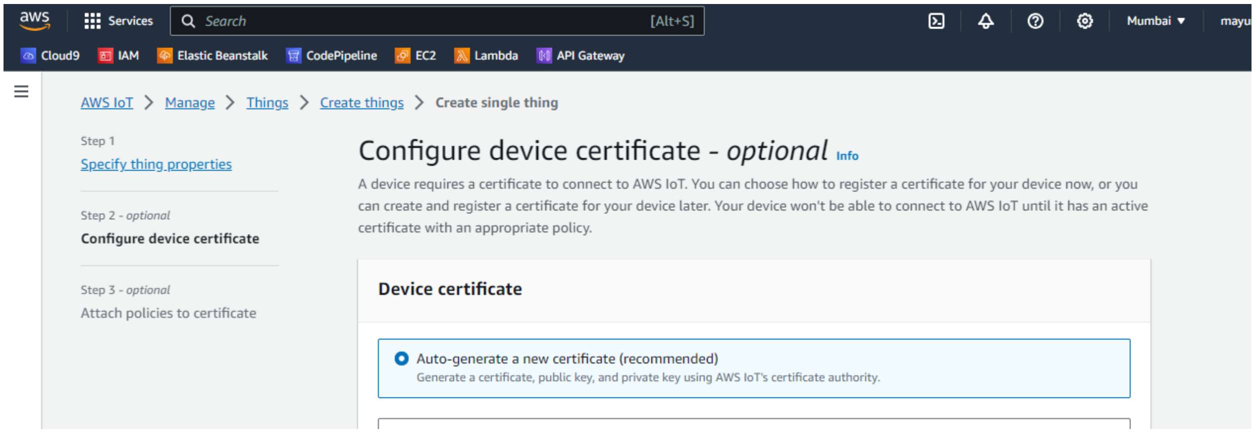
Manage

Manage your devices, IoT data, remote actions, security, and applications

Create and maintain AWS IoT things, interact with devices, and monitor the security of your devices.

Create a thing resource

AWS IoT thing resources represent your physical devices in the cloud and maintain a record of their properties and data. Thing resources also let your devices access AWS IoT services. Create a thing resource to register a single device with AWS IoT.



Step 1
Specify thing properties

Step 2 - *optional*
Configure device certificate

Step 3 - *optional*
Attach policies to certificate

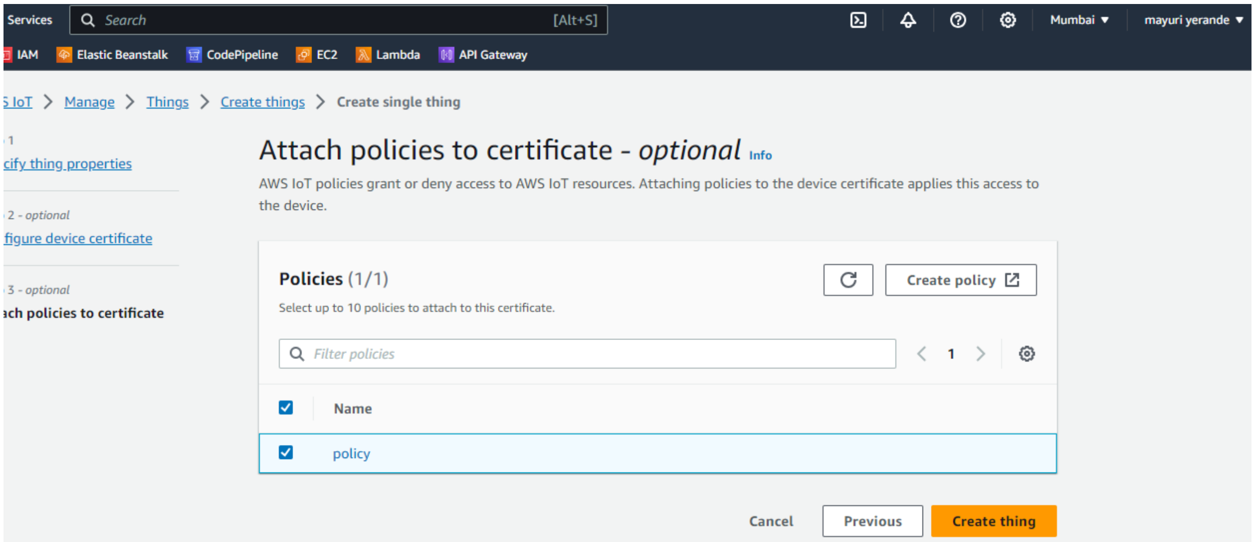
Configure device certificate - *optional*

A device requires a certificate to connect to AWS IoT. You can choose how to register a certificate for your device now, or you can create and register a certificate for your device later. Your device won't be able to connect to AWS IoT until it has an active certificate with an appropriate policy.

Device certificate

● Auto-generate a new certificate (recommended)
Generate a certificate, public key, and private key using AWS IoT's certificate authority.

Step 5: Attach policy and click on create thing



Attach policies to certificate - *optional* Info

AWS IoT policies grant or deny access to AWS IoT resources. Attaching policies to the device certificate applies this access to the device.

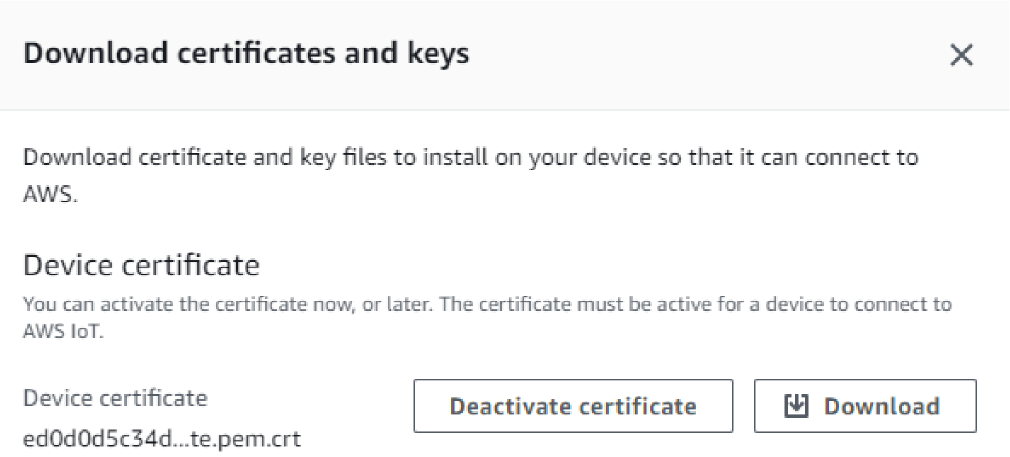
Policies (1/1)

Select up to 10 policies to attach to this certificate.

Filter policies

☑	Name
☑	policy

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



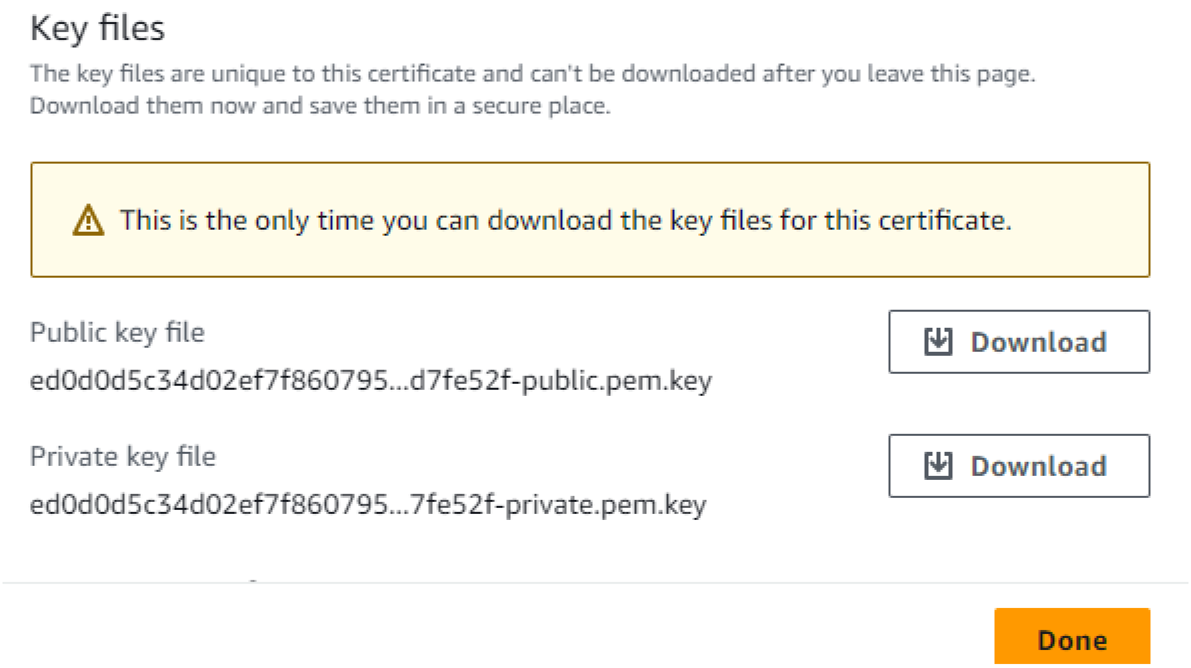
Download certificates and keys

Download certificate and key files to install on your device so that it can connect to AWS.

Device certificate

You can activate the certificate now, or later. The certificate must be active for a device to connect to AWS IoT.

Device certificate
ed0d0d5c34d...te.pem.crt



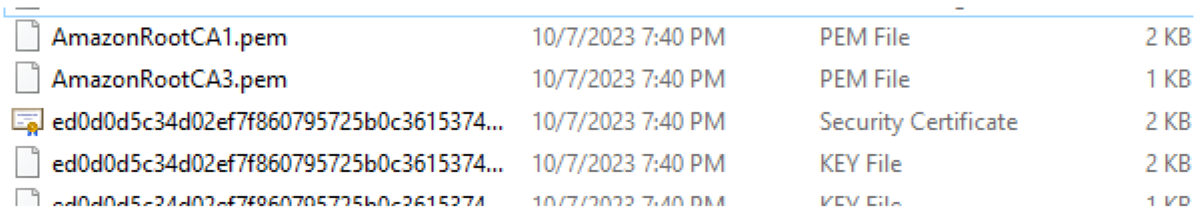
Key files

The key files are unique to this certificate and can't be downloaded after you leave this page. Download them now and save them in a secure place.

⚠ This is the only time you can download the key files for this certificate.

Public key file
ed0d0d5c34d02ef7f860795...d7fe52f-public.pem.key

Private key file
ed0d0d5c34d02ef7f860795...7fe52f-private.pem.key



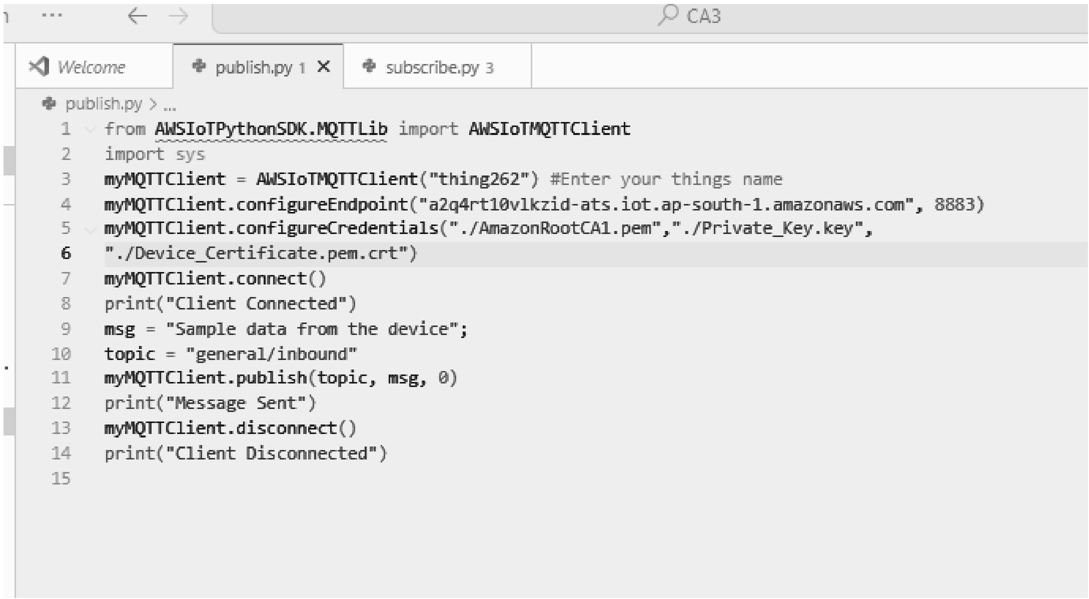
AmazonRootCA1.pem	10/7/2023 7:40 PM	PEM File	2 KB
AmazonRootCA3.pem	10/7/2023 7:40 PM	PEM File	1 KB
ed0d0d5c34d02ef7f860795725b0c3615374...	10/7/2023 7:40 PM	Security Certificate	2 KB
ed0d0d5c34d02ef7f860795725b0c3615374...	10/7/2023 7:40 PM	KEY File	2 KB
ed0d0d5c34d02ef7f860795725b0c3615374	10/7/2023 7:40 PM	KEY File	1 KB

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

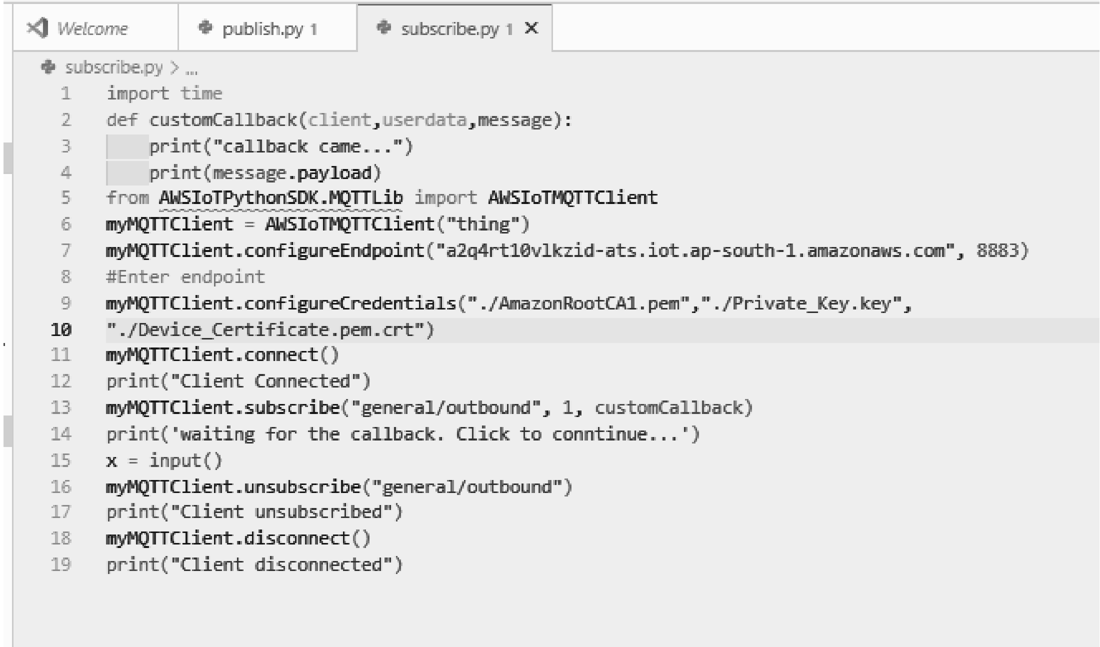
Step 8: After cloning Run below commands
-cd aws-iot-device-sdk-python
-python setup.py install (2)

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

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

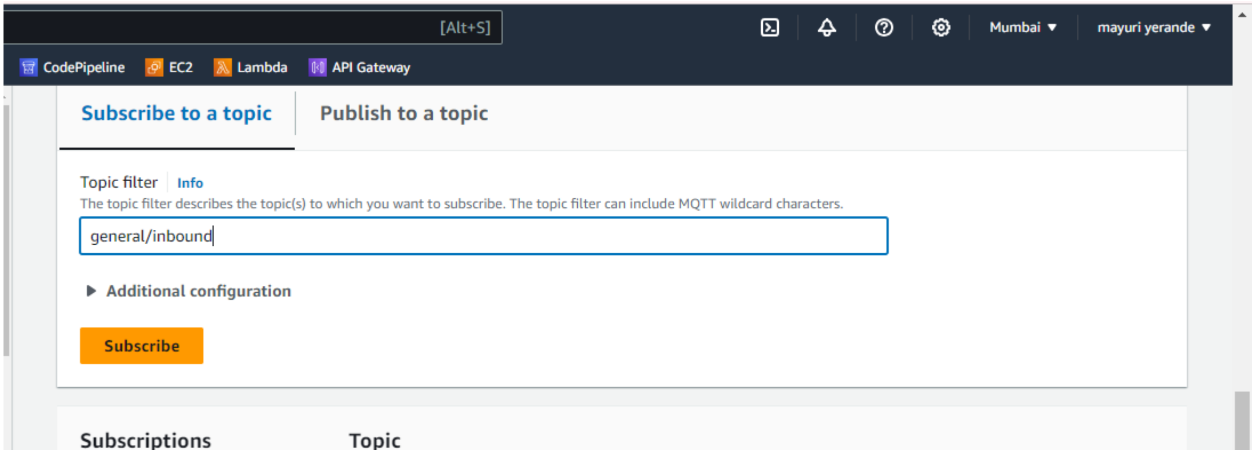


```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
import sys
myMQTTClient = AWSIoTMQTTClient("thing262") #Enter your things name
myMQTTClient.configureEndpoint("a2q4rt10vlkzid-ats.iot.ap-south-1.amazonaws.com", 8883)
myMQTTClient.configureCredentials("./AmazonRootCA1.pem","./Private_Key.key",
"./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")

```



```
import time
def customCallback(client,userdata,message):
    print("callback came...")
    print(message.payload)
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
myMQTTClient = AWSIoTMQTTClient("thing")
myMQTTClient.configureEndpoint("a2q4rt10vlkzid-ats.iot.ap-south-1.amazonaws.com", 8883)
#Enter endpoint
myMQTTClient.configureCredentials("./AmazonRootCA1.pem","./Private_Key.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 11: In MQTT go to Subscribe to a topic and type ‘general/inbound’ and click subscribe ,do not close this page.

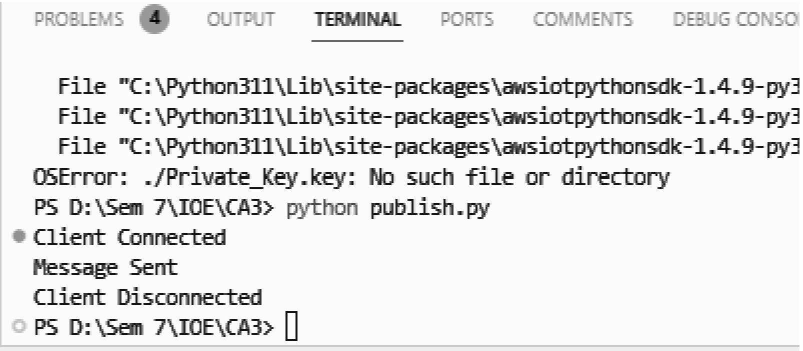


Topic filter Info

The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

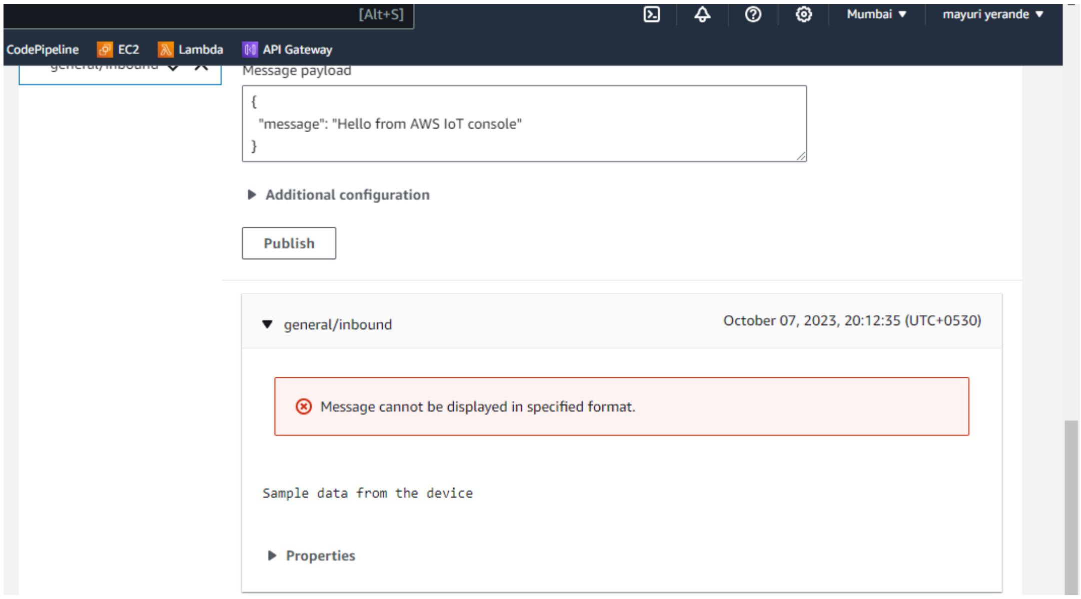
general/inbound

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



```
  File "C:\Python311\Lib\site-packages\awsiotpythonsdk-1.4.9-py3
  File "C:\Python311\Lib\site-packages\awsiotpythonsdk-1.4.9-py3
  File "C:\Python311\Lib\site-packages\awsiotpythonsdk-1.4.9-py3
OSError: ./Private_Key.key: No such file or directory
PS D:\Sem 7\IOE\CA3> python publish.py
Client Connected
Message Sent
Client Disconnected
PS D:\Sem 7\IOE\CA3>
```

Step 13: You can see message on subscribe page



Message payload

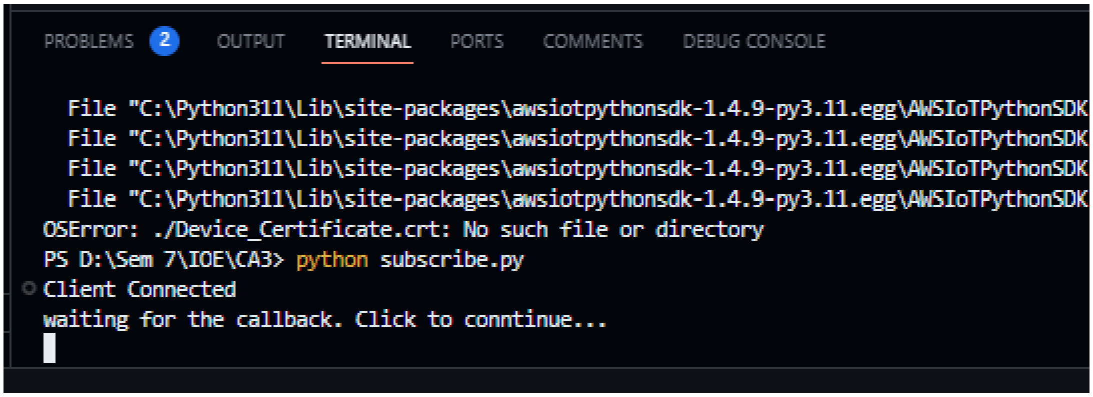
```
{
 "message": "Hello from AWS IoT console"
}
```

general/inbound October 07, 2023, 20:12:35 (UTC+0530)

Message cannot be displayed in specified format.

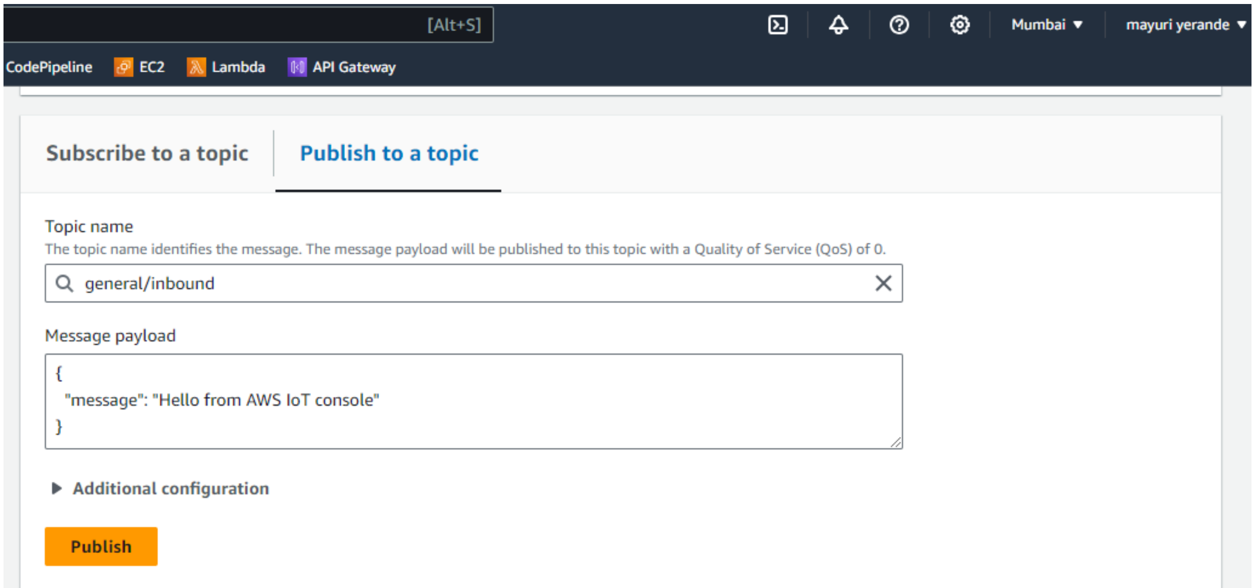
```
Sample data from the device
```

Step 14: Then run subscribe.py



```
  File "C:\Python311\Lib\site-packages\awsiotpythonsdk-1.4.9-py3.11.egg\AWSIoTPythonSDK
  File "C:\Python311\Lib\site-packages\awsiotpythonsdk-1.4.9-py3.11.egg\AWSIoTPythonSDK
  File "C:\Python311\Lib\site-packages\awsiotpythonsdk-1.4.9-py3.11.egg\AWSIoTPythonSDK
  File "C:\Python311\Lib\site-packages\awsiotpythonsdk-1.4.9-py3.11.egg\AWSIoTPythonSDK
OSError: ./Device_Certificate.crt: No such file or directory
PS D:\Sem 7\IOE\CA3> python subscribe.py
Client Connected
waiting for the callback. Click to conntinue...
```

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



Topic name

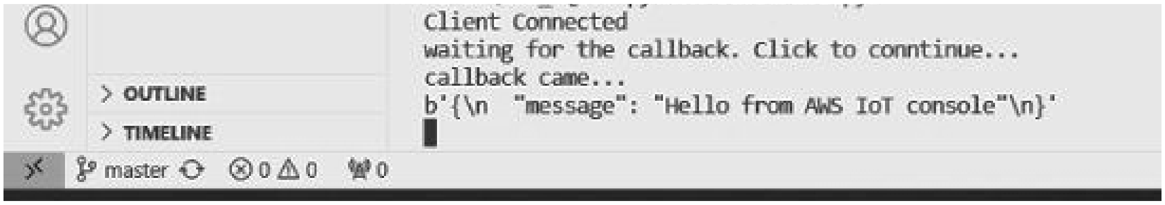
The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.

general/inbound

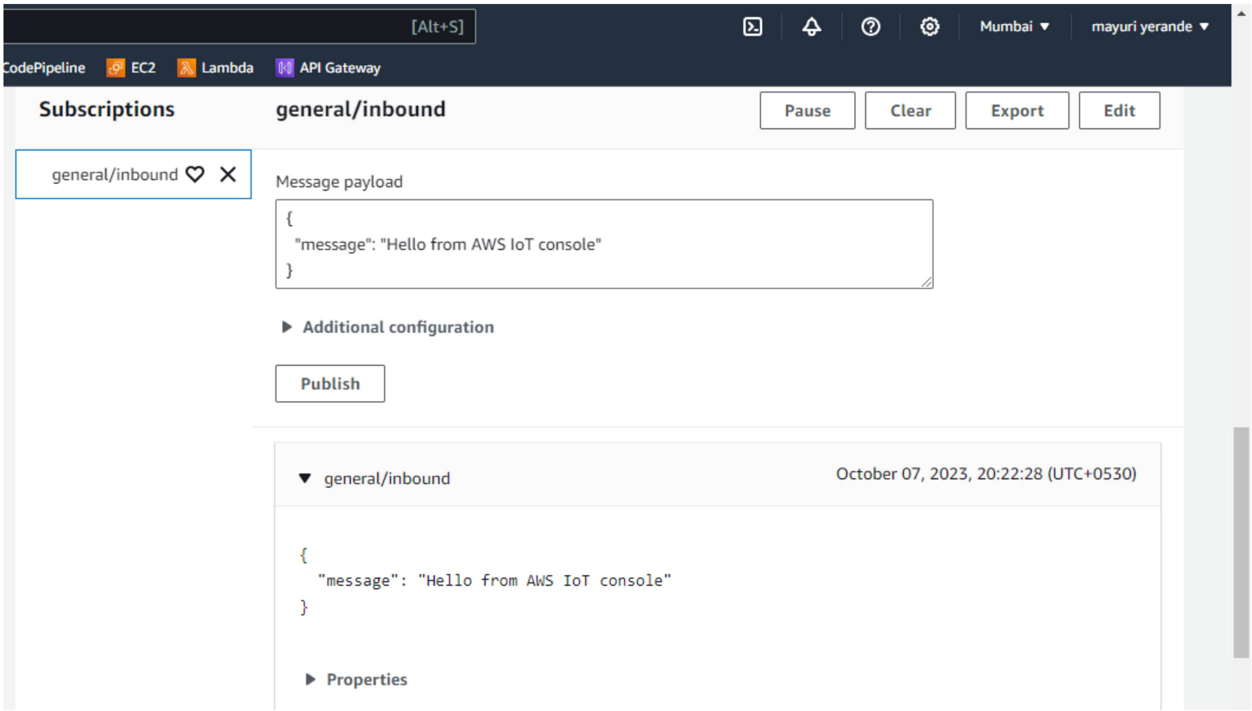
Message payload

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

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



```
Client Connected
waiting for the callback. Click to conntinue...
callback came...
b'{\n  "message": "Hello from AWS IoT console"\n}'
```



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

October 07, 2023, 20:22:28 (UTC+0530)

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

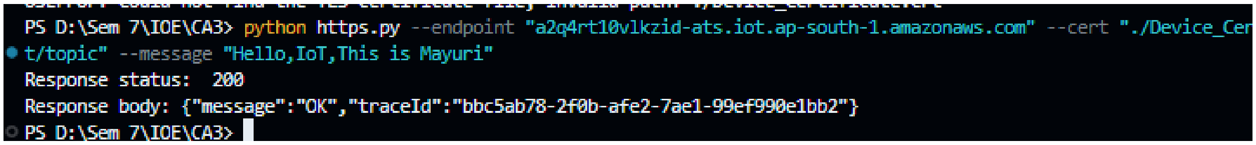
❖ HTTPS

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()
```

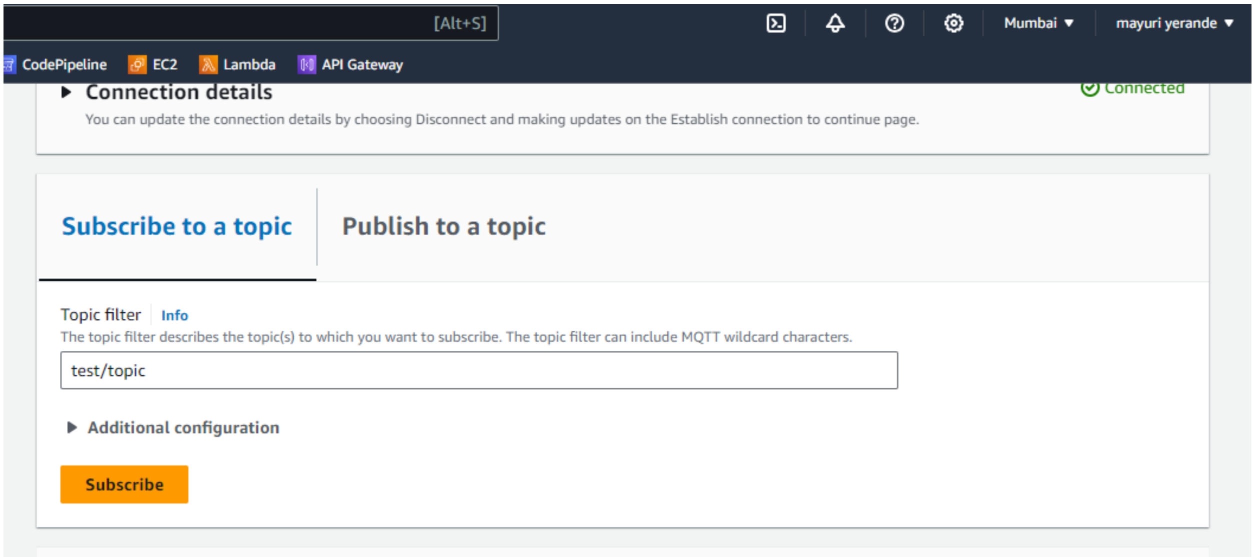
```
publish_url = 'https://' + args.endpoint + ':8443/topics/' +
args.topic + '?qos=1'
publish_msg = args.message.encode('utf-8')
# make request
publish = requests.request('POST',publish_url, data=publish_msg,
cert=[args.cert, args.key])
# print results
print("Response status: ", str(publish.status_code))
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_ioe.py --endpoint "a1x4rget4tmruj-ats.iot.eu-north-1.amazonaws.com" --cert "./Device_Certificate.crt" --key "./Private_Key.key" --topic "test/topic" --message "Hello, IoT, This is Mayuri”



```
PS D:\Sem 7\IOE\CA3> python https.py --endpoint "a2q4rt10vlkzid-ats.iot.ap-south-1.amazonaws.com" --cert "./Device_Cer
t/topic" --message "Hello,IoT,This is Mayuri"
Response status:  200
Response body: {"message":"OK","traceId":"bbc5ab78-2f0b-afe2-7ae1-99ef990e1bb2"}
PS D:\Sem 7\IOE\CA3>
```

Step 2: Check message received by subscribing to the topic



▶ Connection details

✓ Connected

You can update the connection details by choosing Disconnect and making updates on the Establish connection to continue page.

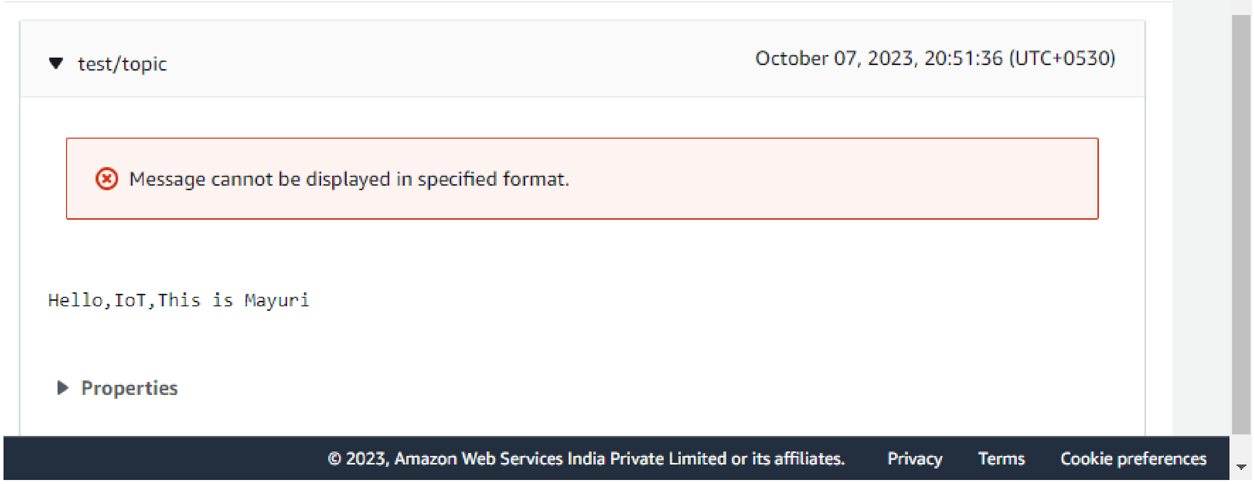
Subscribe to a topic | Publish to a topic

Topic filter Info

The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

test/topic

▶ Additional configuration



▼ test/topic

October 07, 2023, 20:51:36 (UTC+0530)

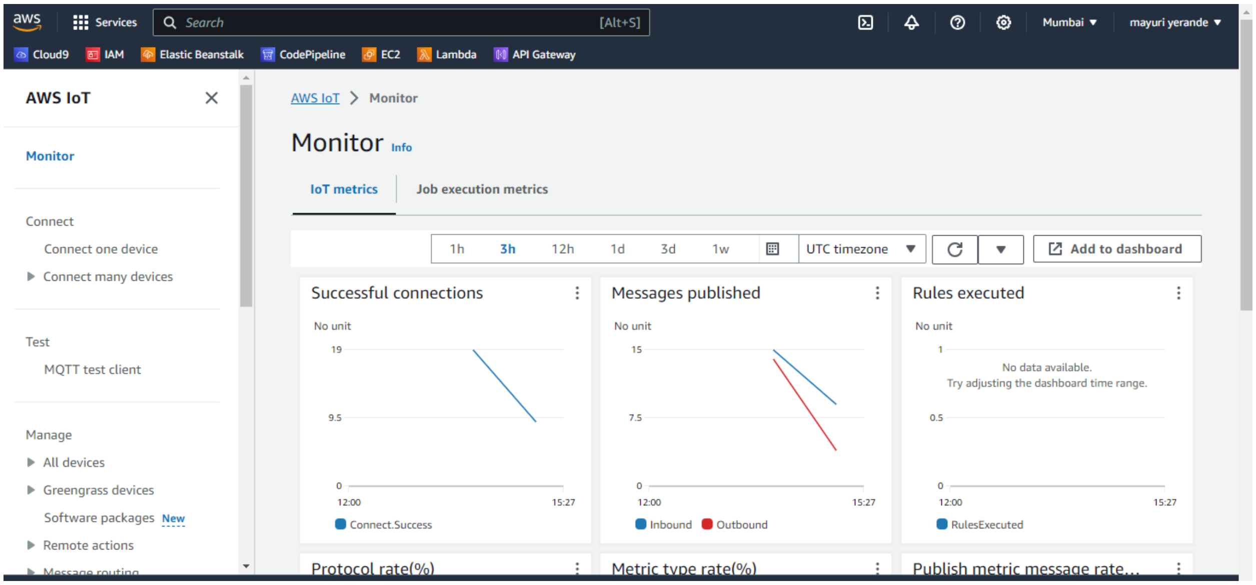
Message cannot be displayed in specified format.

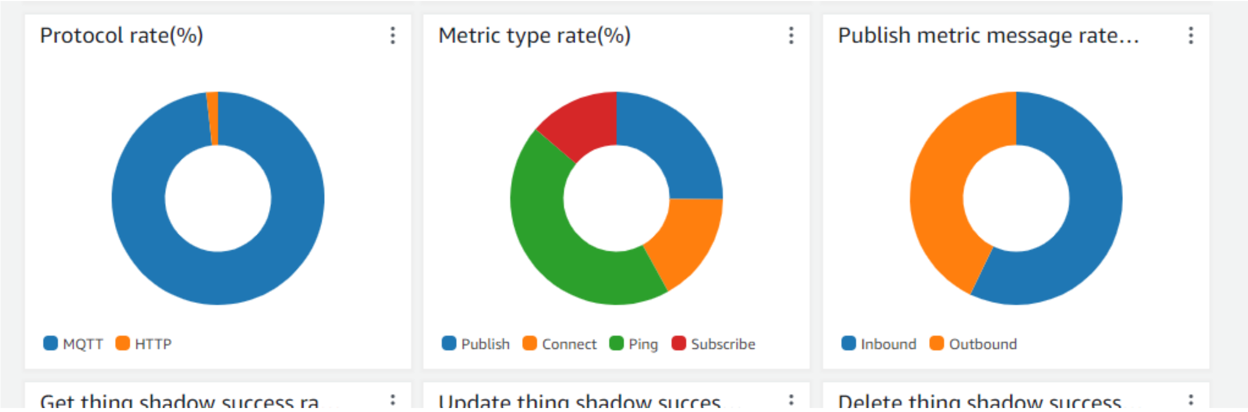
```
Hello,IoT,This is Mayuri
```

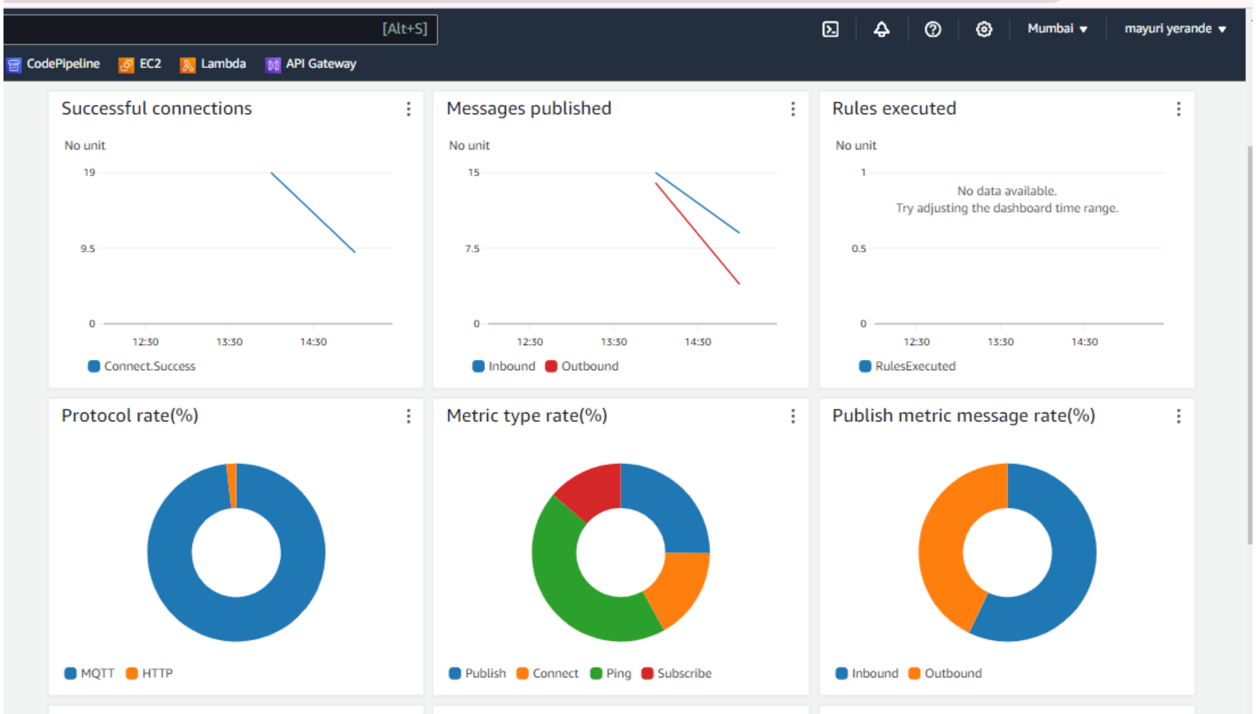
▶ Properties

© 2023, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

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

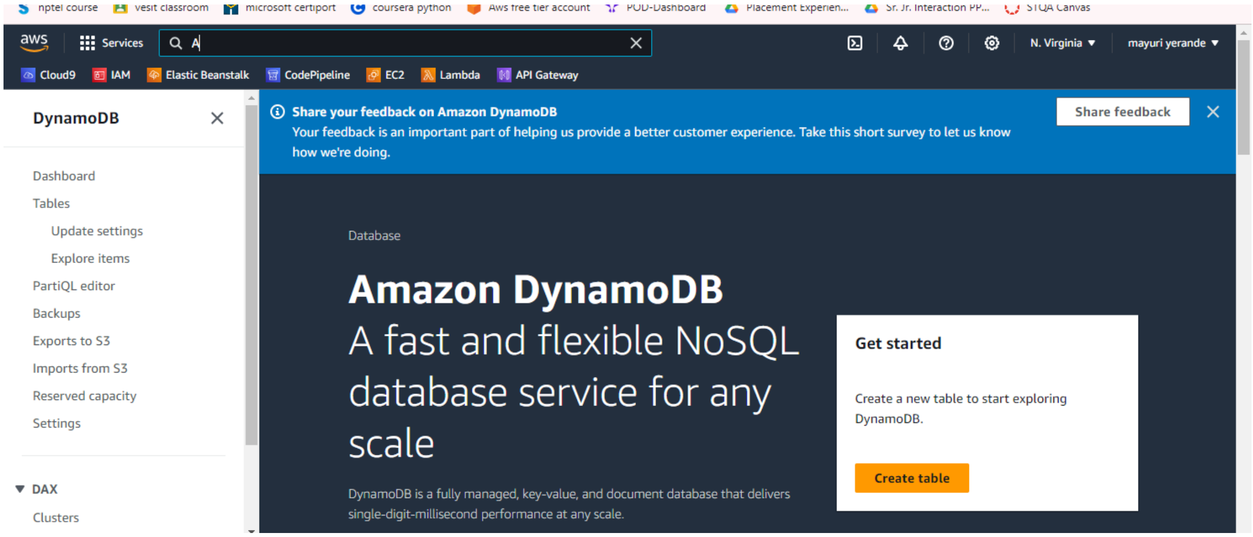






❖ **WEBSOCKET**

- Open DynamoDB



Share your feedback on Amazon DynamoDB
Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.

Database

Amazon DynamoDB
A fast and flexible NoSQL database service for any scale

DynamoDB is a fully managed, key-value, and document database that delivers single-digit-millisecond performance at any scale.

Get started

Create a new table to start exploring DynamoDB.

- Create DynamoDB Table

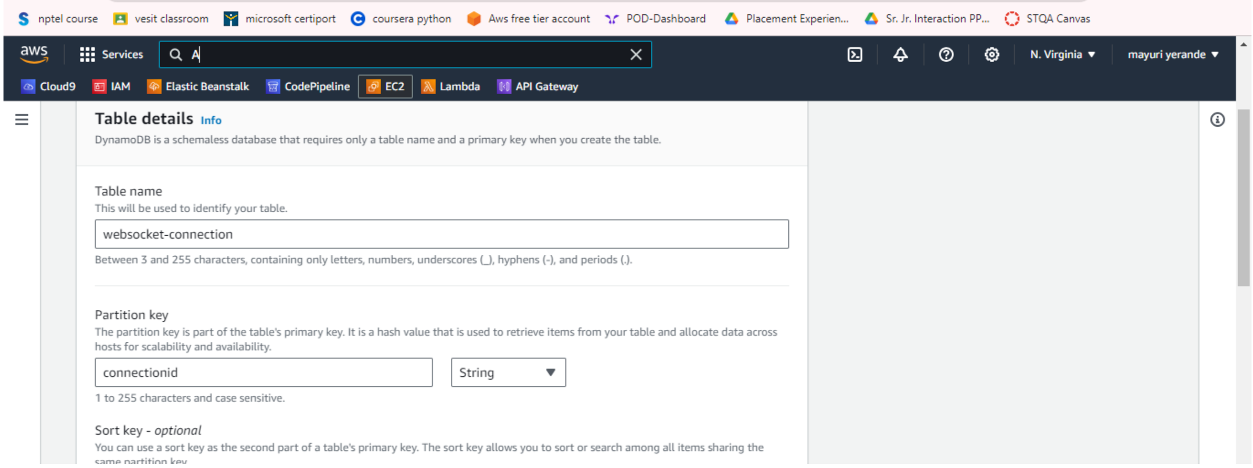


Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

websocket-connection

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

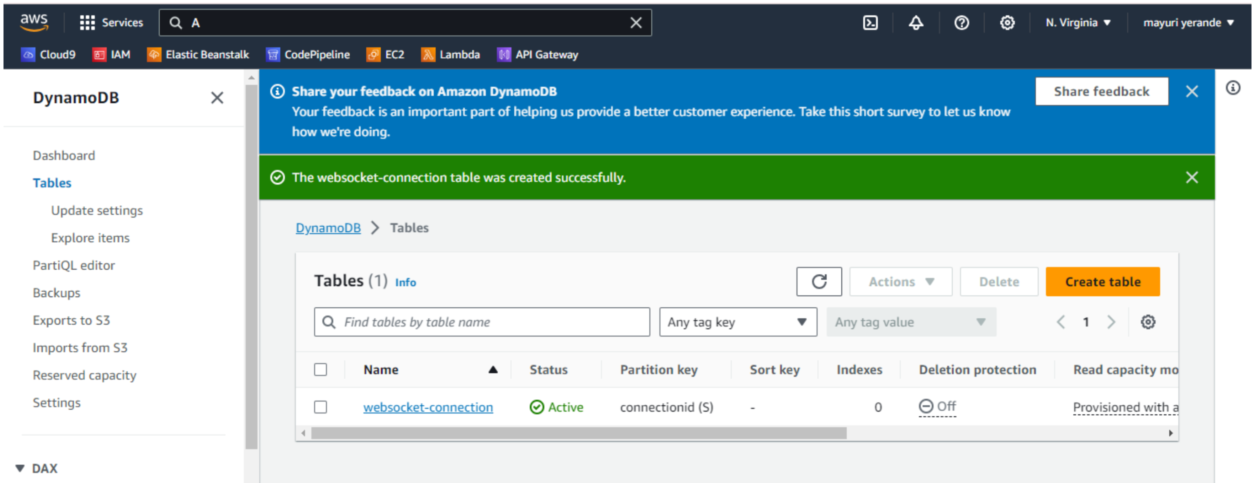
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

connectionid String

1 to 255 characters and case sensitive.

Sort key - *optional*

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.



Share your feedback on Amazon DynamoDB

Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.

The websocket-connection table was created successfully.

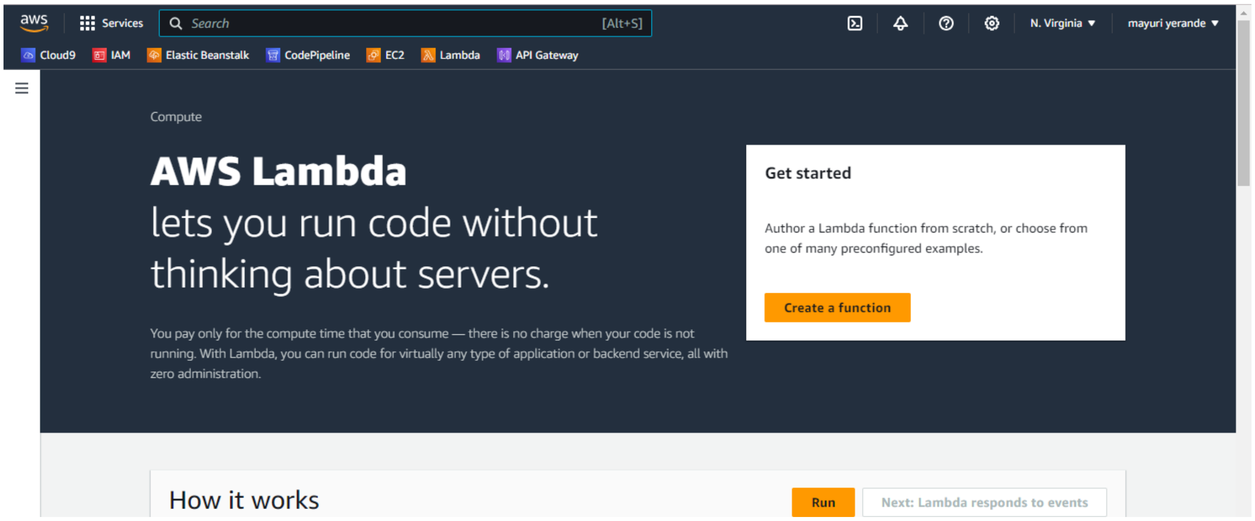
DynamoDB > Tables

Tables (1) Info

Find tables by table name | Any tag key | Any tag value | 1

□	Name ▲	Status	Partition key	Sort key	Indexes	Deletion protection	Read capacity mo
□	websocket-connection	✓ Active	connectionid (S)	-	0	Off	Provisioned with a

- Now we will be creating our lambda functions for connecting ,disconnecting and sending data for websocket



Compute

AWS Lambda

lets you run code without thinking about servers.

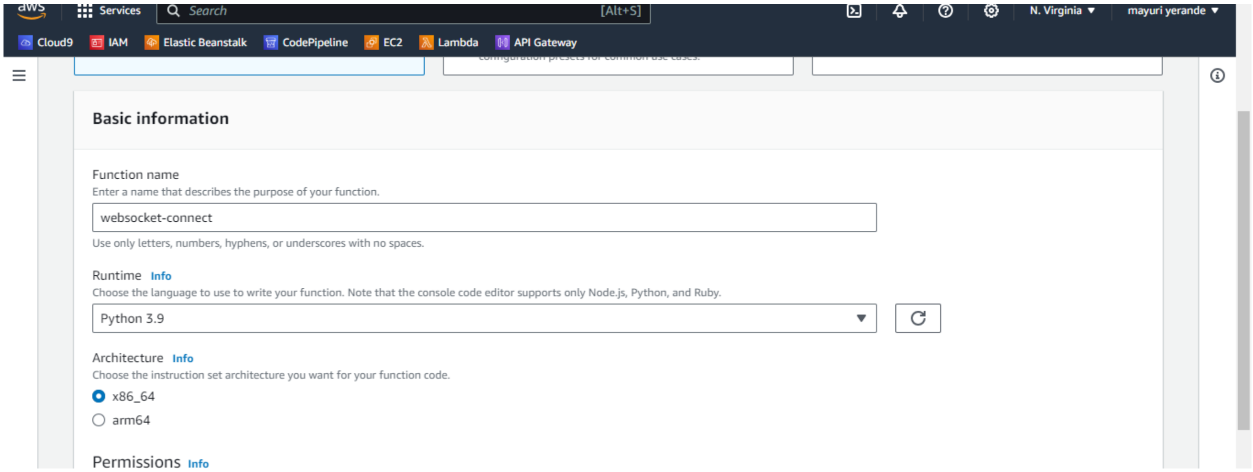
You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

How it works

- Creation of Lambda function for connecting websocket



Basic information

Function name
Enter a name that describes the purpose of your function.
websocket-connect
Use only letters, numbers, hyphens, or underscores with no spaces.

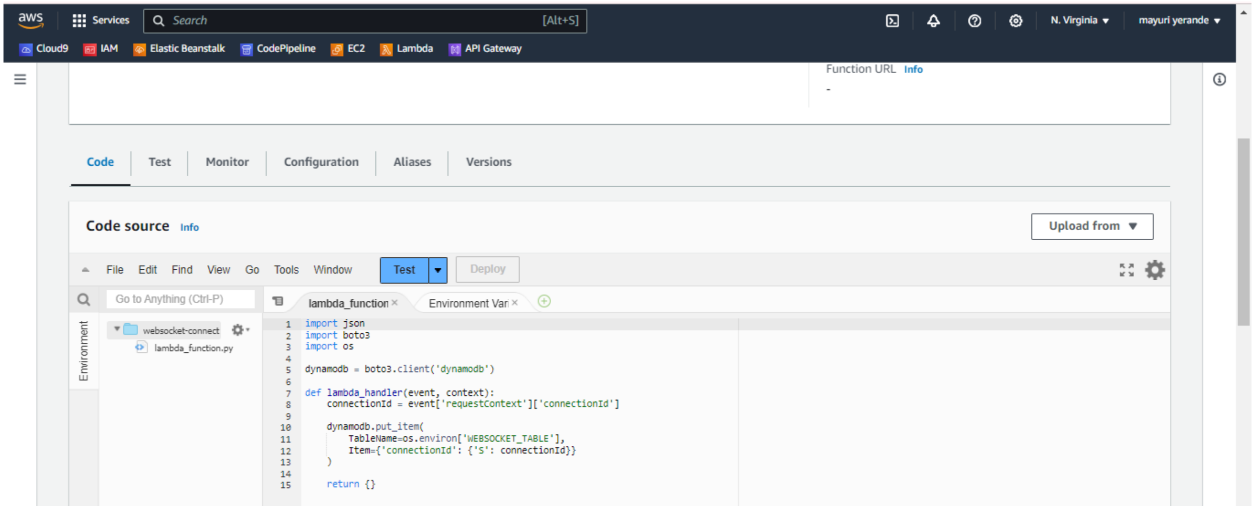
Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.9

Architecture Info
Choose the instruction set architecture you want for your function code.
● x86_64
○ arm64

Permissions Info

Add this code in our function

```
import json
import boto3
import os

dynamodb = boto3.client('dynamodb')

def lambda_handler(event, context):
    connectionId = event['requestContext']['connectionId']

    dynamodb.put_item(
        TableName=os.environ['WEBSOCKET_TABLE'],
        Item={'connectionId': {'S': connectionId}}
    )
    return {}
```

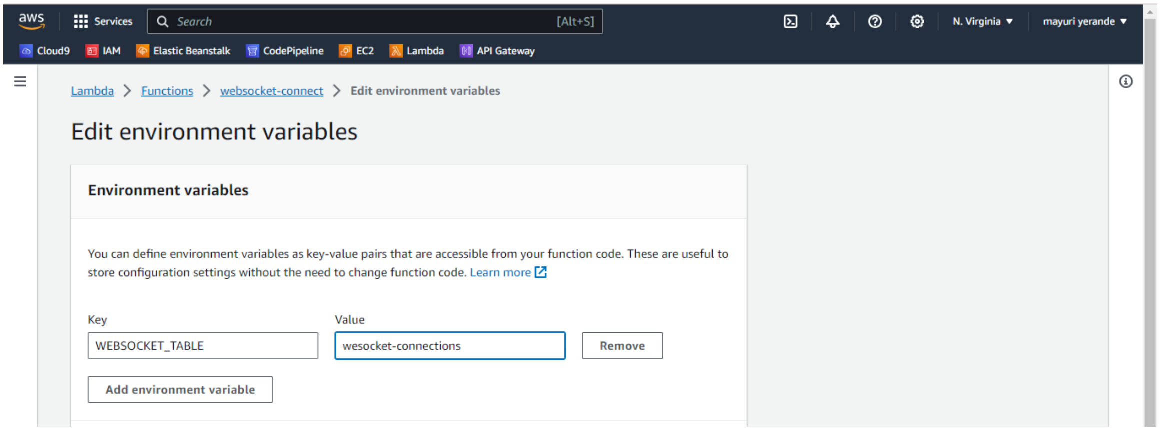


Function URL Info
-

Code source Info

```
import json
import boto3
import os

dynamodb = boto3.client('dynamodb')

def lambda_handler(event, context):
    connectionId = event['requestContext']['connectionId']

    dynamodb.put_item(
        TableName=os.environ['WEBSOCKET_TABLE'],
        Item={'connectionId': {'S': connectionId}}
    )

    return {}
```

- Go to lambda function configuration and choose environment variables and add this variable to our table.



Lambda > Functions > websocket-connect > Edit environment variables

Edit environment variables

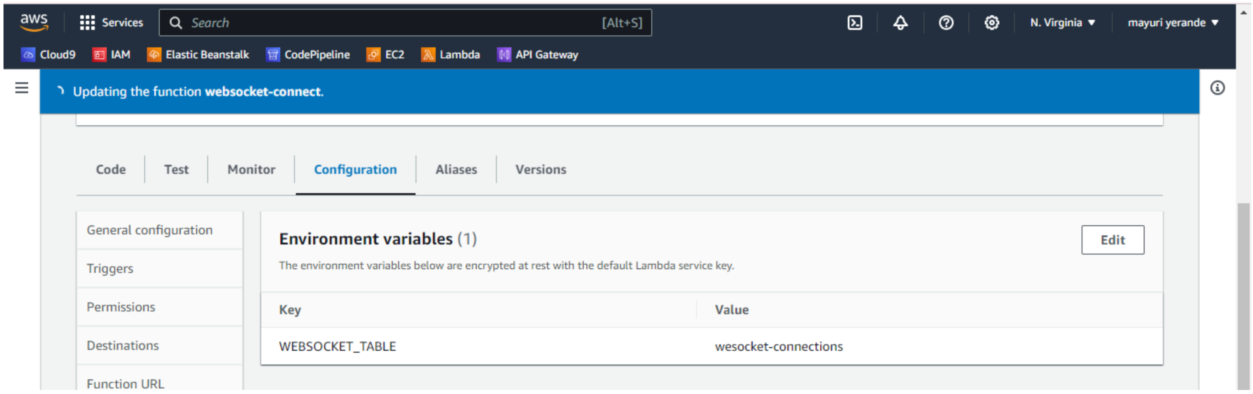
Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more

Key: WEBSOCKET_TABLE

Value: wesocket-connections

- Our environment variable is created



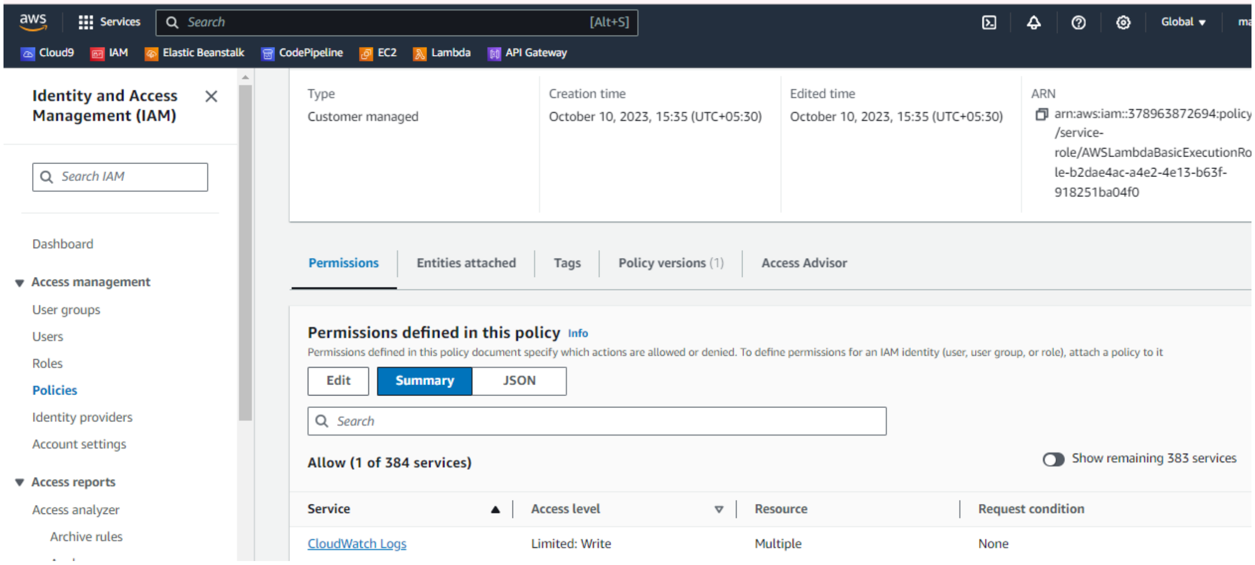
Updating the function **websocket-connect**.

Environment variables (1)

The environment variables below are encrypted at rest with the default Lambda service key.

Key	Value
WEBSOCKET_TABLE	wesocket-connections

- Now we have to add certain permissions for connecting the websocket.
- For "websocket-connect" function, we will add the permission of "putitem" action



Type	Creation time	Edited time	ARN
Customer managed	October 10, 2023, 15:35 (UTC+05:30)	October 10, 2023, 15:35 (UTC+05:30)	arn:aws:iam::378963872694:policy/service-role/AWSLambdaBasicExecutionRole-b2dae4ac-a4e2-4e13-b63f-918251ba04f0

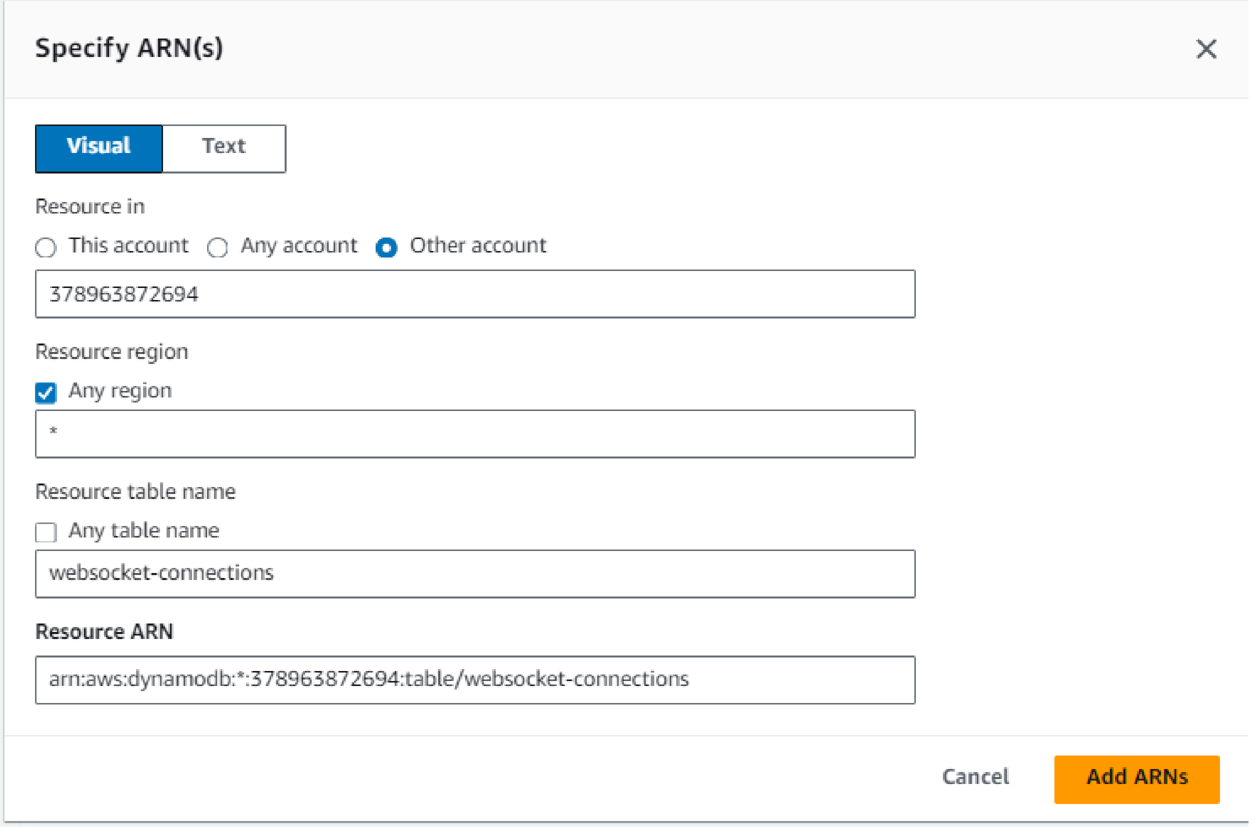
Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Allow (1 of 384 services)

Service	Access level	Resource	Request condition
CloudWatch Logs	Limited: Write	Multiple	None

- Add the "arn" for our specific table. Add the table name in it



Specify ARN(s)

Resource in

○ This account ○ Any account ● Other account

378963872694

Resource region

☑ Any region

*

Resource table name

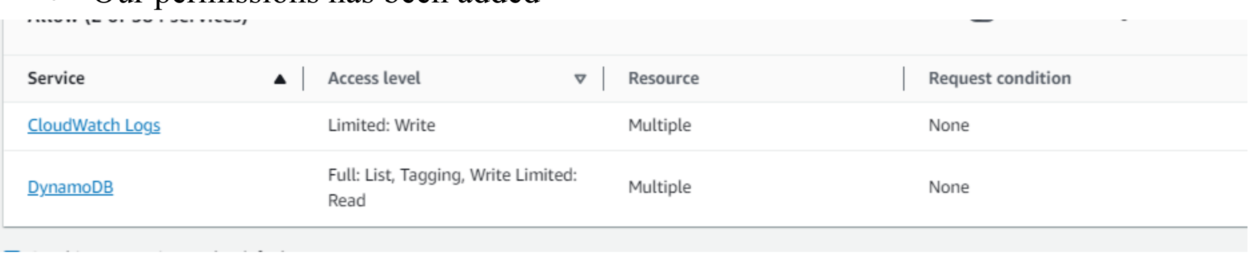
□ Any table name

websocket-connections

Resource ARN

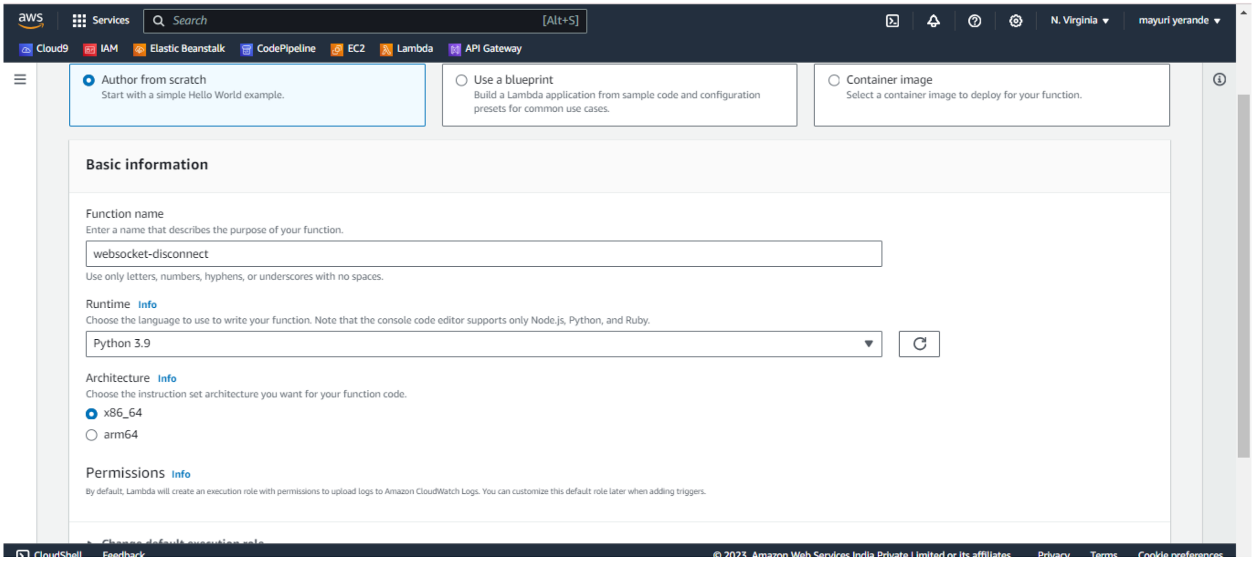
arn:aws:dynamodb:*:378963872694:table/websocket-connections

- Our permissions has been added



Service	Access level	Resource	Request condition
CloudWatch Logs	Limited: Write	Multiple	None
DynamoDB	Full: List, Tagging, Write Limited: Read	Multiple	None

- Create new lambda function for disconnecting from websocket



● Author from scratch
Start with a simple Hello World example.

○ Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

○ Container image
Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.
websocket-disconnect
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.9

Architecture Info
Choose the instruction set architecture you want for your function code.
● x86_64
○ arm64

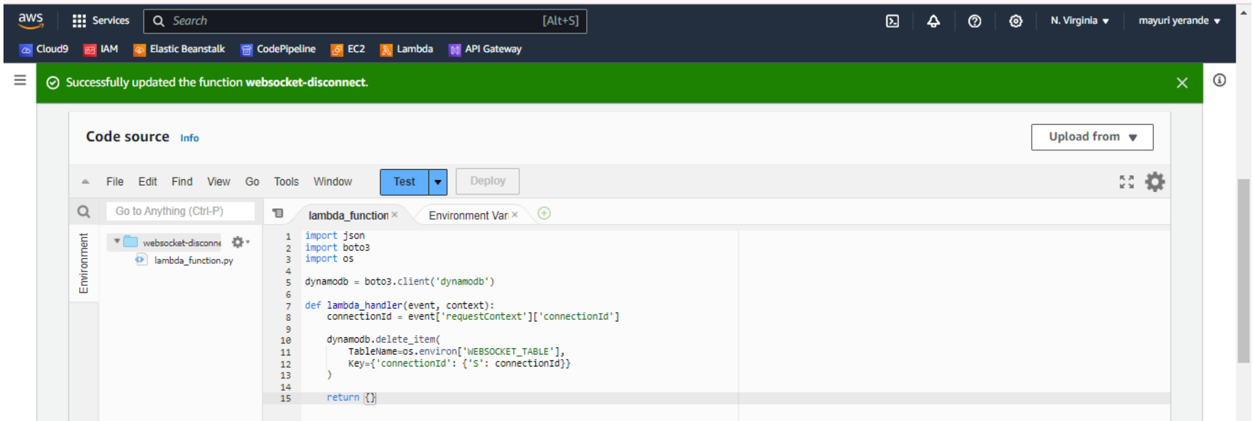
Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▸ Change default execution role

CloudShell
Feedback
© 2023, Amazon Web Services India Private Limited or its affiliates.
Privacy
Terms
Cookie preferences

- Add the code for disconnect in this function

```
import json
import boto3
import os

dynamodb = boto3.client('dynamodb')

def lambda_handler(event, context):
    connectionId = event['requestContext']['connectionId']

    dynamodb.delete_item(
        TableName=os.environ['WEBSOCKET_TABLE'],
        Key={'connectionId': {'S': connectionId}}
    )

    return {}
```



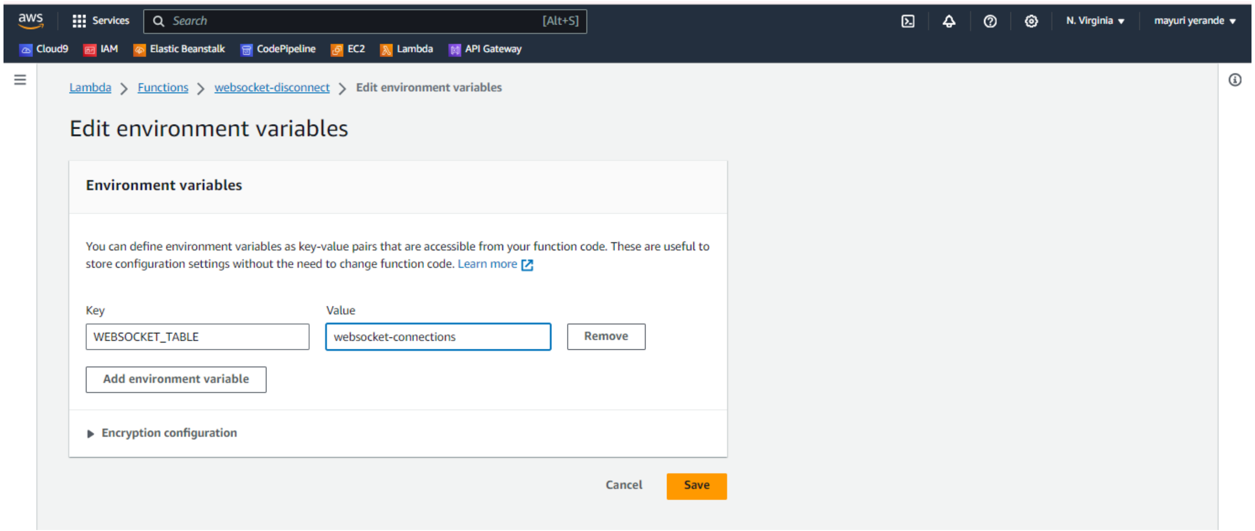
Successfully updated the function **websocket-disconnect**.

Code source Info

```
import json
import boto3
import os

dynamodb = boto3.client('dynamodb')

def lambda_handler(event, context):
    connectionId = event['requestContext']['connectionId']

    dynamodb.delete_item(
        TableName=os.environ['WEBSOCKET_TABLE'],
        Key={'connectionId': {'S': connectionId}}
    )

    return {}
```

- Repeat same steps of variable creation and adding permissions

- Create new environment variable



Lambda > Functions > websocket-disconnect > Edit environment variables

Edit environment variables

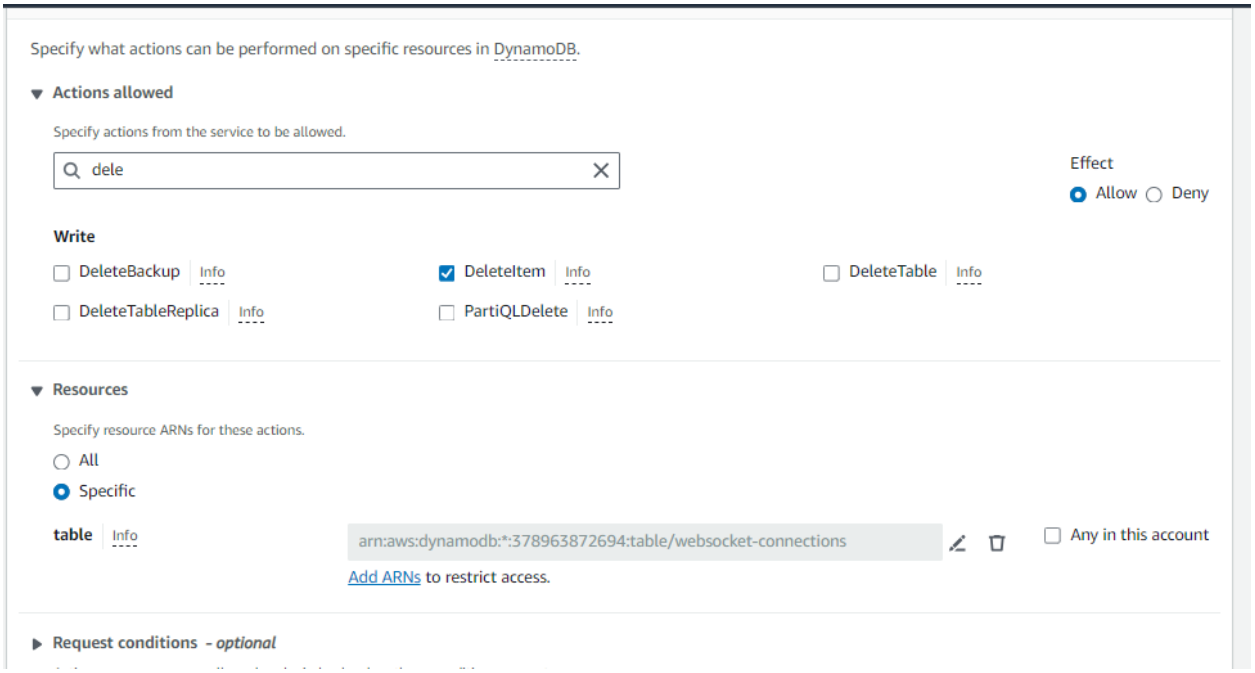
Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more

Key: WEBSOCKET_TABLE

Value: websocket-connections

- Add permission of “delete item” for dynamodb



Specify what actions can be performed on specific resources in DynamoDB.

Actions allowed

Specify actions from the service to be allowed.

dele

Effect
● Allow ○ Deny

Write

□ DeleteBackup Info
☑ DeleteItem Info
□ DeleteTable Info
□ DeleteTableReplica Info
□ PartiQLDelete Info

Resources

Specify resource ARNs for these actions.

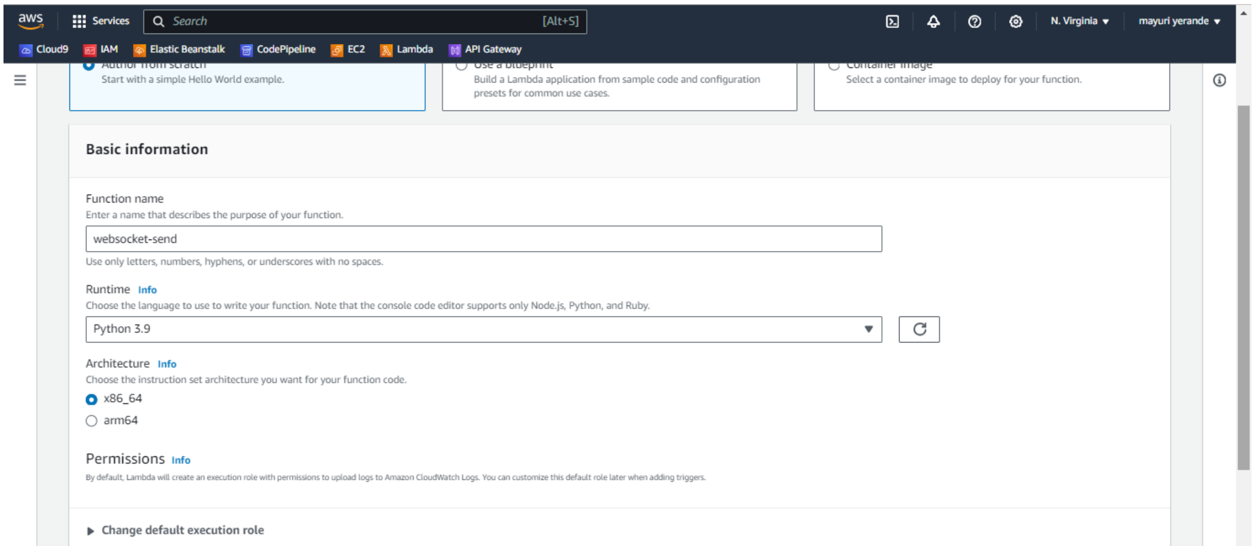
○ All
● Specific

table Info arn:aws:dynamodb:*:378963872694:table/websocket-connections □ Any in this account

Add ARNs to restrict access.

Request conditions - *optional*

- Now creating our final lambda function named “websocket-send”



○ Author from scratch
Start with a simple Hello World example.

○ Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

○ Container image
Select a container image to deploy for your function.

Basic information

Function name
Enter a name that describes the purpose of your function.
websocket-send
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.9

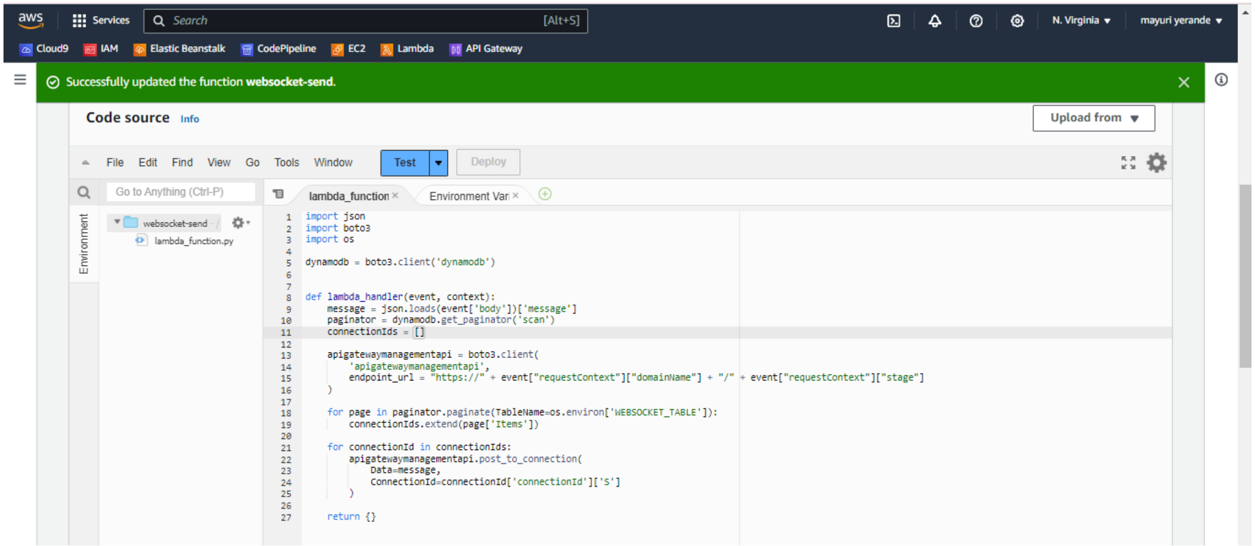
Architecture Info
Choose the instruction set architecture you want for your function code.
● x86_64
○ arm64

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▶ Change default execution role

- Add the code in it and deploy

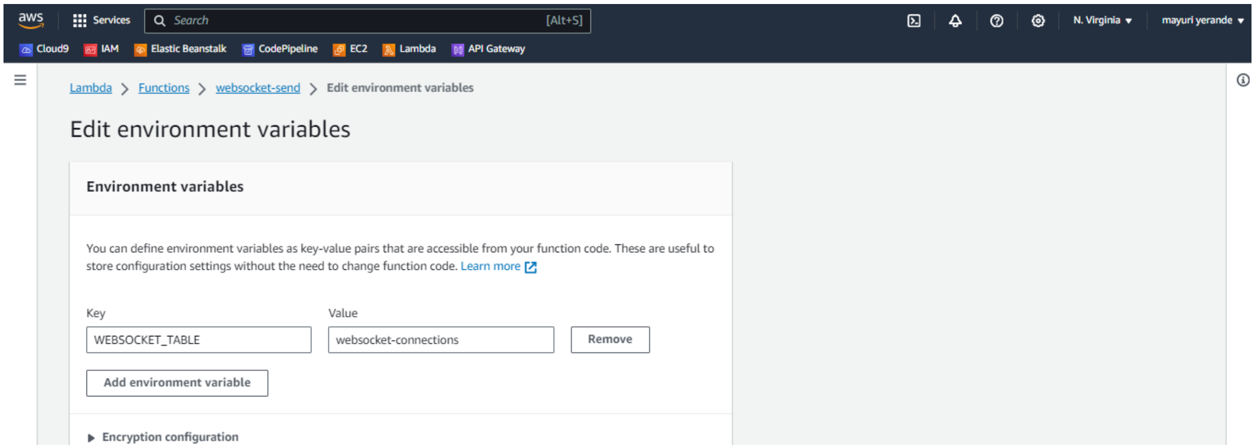
```
import json
import boto3
import os

dynamodb = boto3.client('dynamodb')
def lambda_handler(event, context):
    message = json.loads(event['body'])['message']
    paginator = dynamodb.get_paginator('scan')
    connectionIds = []

    apigatewaymanagementapi = boto3.client(
        'apigatewaymanagementapi',
        endpoint_url = "https://" +
event["requestContext"]["domainName"] + "/" +
event["requestContext"]["stage"]
    )

    for page in
paginator.paginate(TableName=os.environ['WEBSOCKET_TABLE']):
        connectionIds.extend(page['Items'])

    for connectionId in connectionIds:
        apigatewaymanagementapi.post_to_connection(
            Data=message,
            ConnectionId=connectionId['connectionId']['S']
        )

    return {}
```



Successfully updated the function **websocket-send**.

```python
import json
import boto3
import os

dynamodb = boto3.client('dynamodb')


def lambda_handler(event, context):
    message = json.loads(event['body'])['message']
    paginator = dynamodb.get_paginator('scan')
    connectionIds = []

    apigatewaymanagementapi = boto3.client(
        'apigatewaymanagementapi',
        endpoint_url = "https://" + event["requestContext"]["domainName"] + "/" + event["requestContext"]["stage"]
    )

    for page in paginator.paginate(TableName=os.environ['WEBSOCKET_TABLE']):
        connectionIds.extend(page['Items'])

    for connectionId in connectionIds:
        apigatewaymanagementapi.post_to_connection(
            Data=message,
            ConnectionId=connectionId['connectionId']['S']
        )

    return {}
```

- Repeat the same thing again
- Set up the environment variable



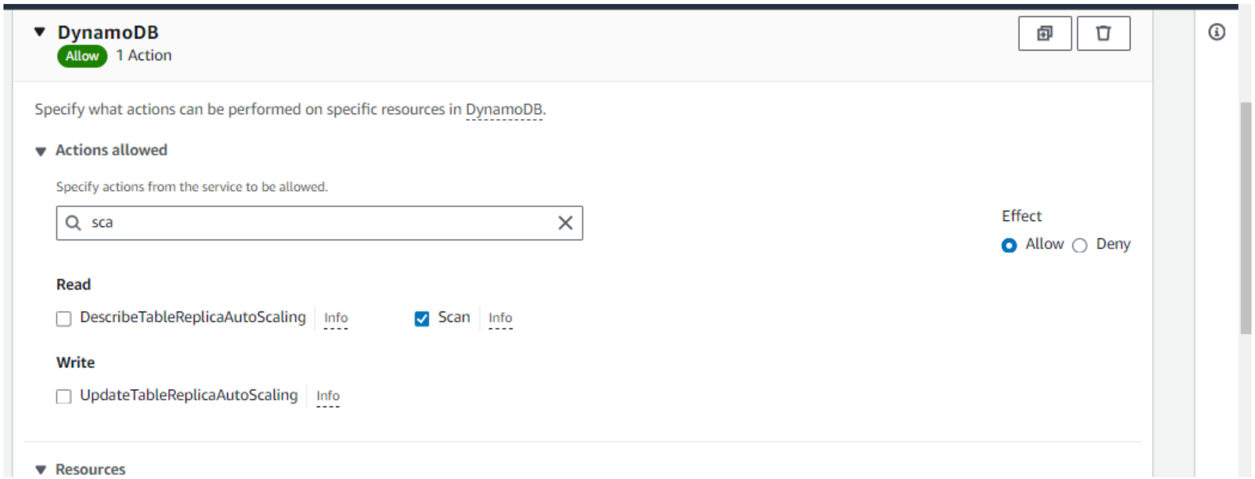
Edit environment variables

Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more

Key	Value
WEBSOCKET_TABLE	websocket-connections

- Add permission here with action "scan"



DynamoDB

Specify what actions can be performed on specific resources in DynamoDB.

Actions allowed

Specify actions from the service to be allowed.

sca

Effect

● Allow ○ Deny

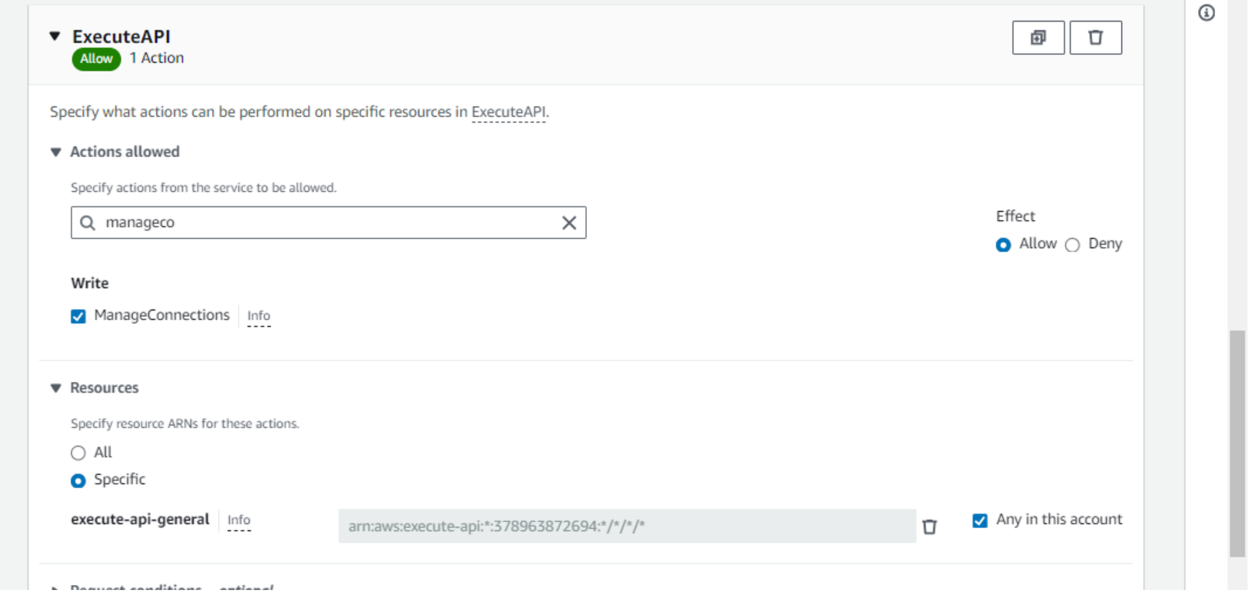
Read

□ DescribeTableReplicaAutoScaling ☑ Scan

Write

□ UpdateTableReplicaAutoScaling

- And add another permission with "executeApi" action



ExecuteAPI

Allow 1 Action

Specify what actions can be performed on specific resources in ExecuteAPI.

Actions allowed

Specify actions from the service to be allowed.

manageco

Effect ● Allow ○ Deny

Write

☑ ManageConnections Info

Resources

Specify resource ARNs for these actions.

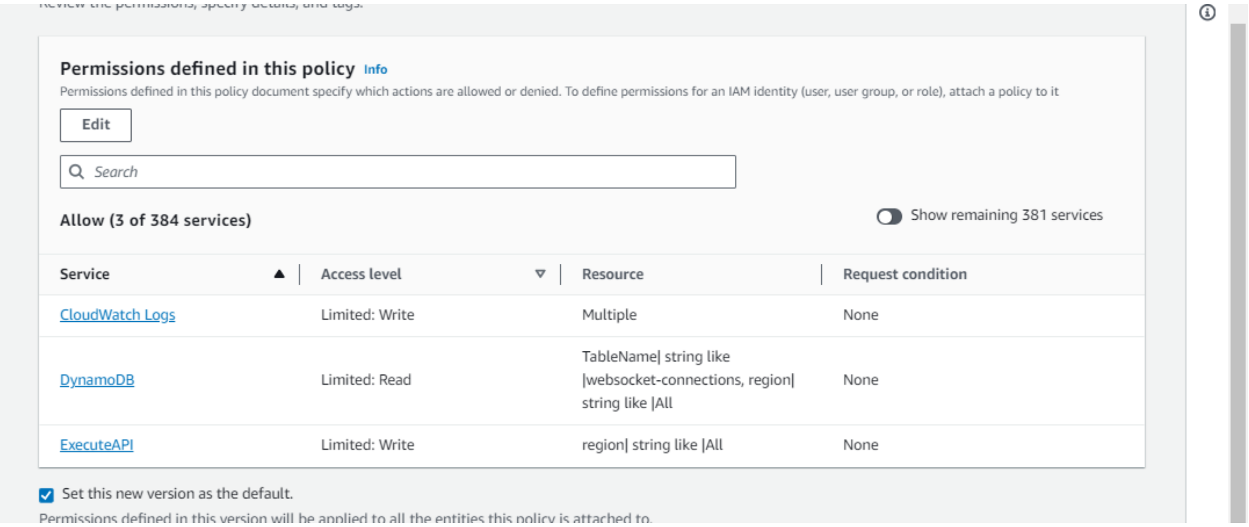
○ All

● Specific

execute-api-general Info arn:aws:execute-api:*:378963872694:*/*/*/* ☑ Any in this account

Request conditions - optional

- Save the permissions



Review the permissions, specify details, and tags.

Permissions defined in this policy Info

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Edit

Search

Allow (3 of 384 services)

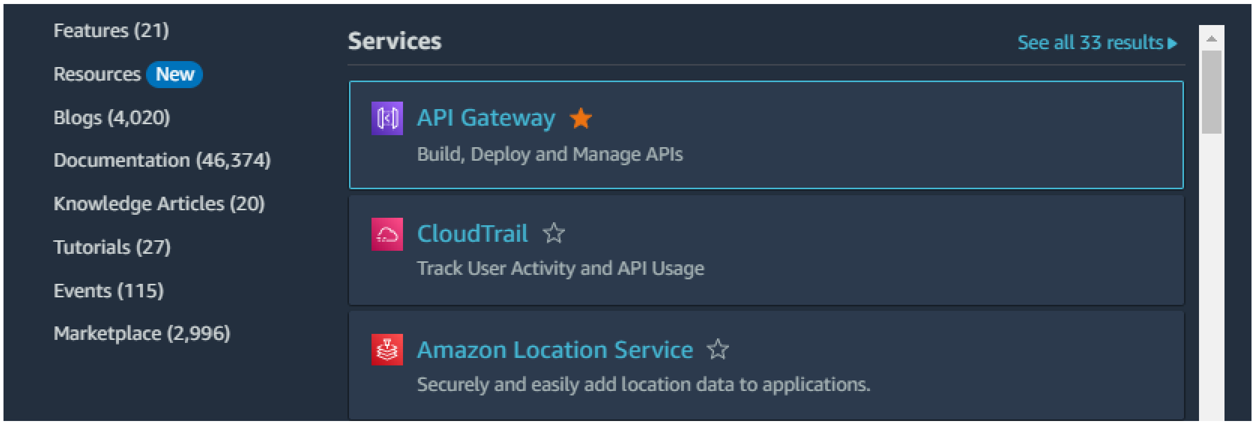
Show remaining 381 services

Service	Access level	Resource	Request condition
CloudWatch Logs	Limited: Write	Multiple	None
DynamoDB	Limited: Read	TableName| string like |websocket-connections, region| string like |All	None
ExecuteAPI	Limited: Write	region| string like |All	None

☑ Set this new version as the default.

Permissions defined in this version will be applied to all the entities this policy is attached to.

- Now we will create the API gateway



Features (21)

Resources New

Blogs (4,020)

Documentation (46,374)

Knowledge Articles (20)

Tutorials (27)

Events (115)

Marketplace (2,996)

Services See all 33 results ▸

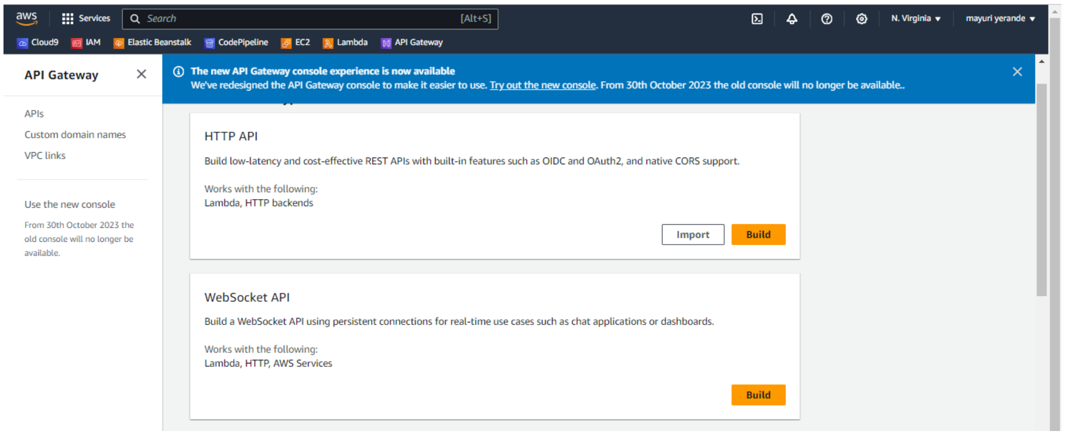
API Gateway ★
Build, Deploy and Manage APIs

CloudTrail ☆
Track User Activity and API Usage

Amazon Location Service ☆
Securely and easily add location data to applications.

- Choose the “websocket api”

- Click on build



From 30th October 2023 the old console will no longer be available.

The new API Gateway console experience is now available
We've redesigned the API Gateway console to make it easier to use. Try out the new console. From 30th October 2023 the old console will no longer be available..

HTTP API

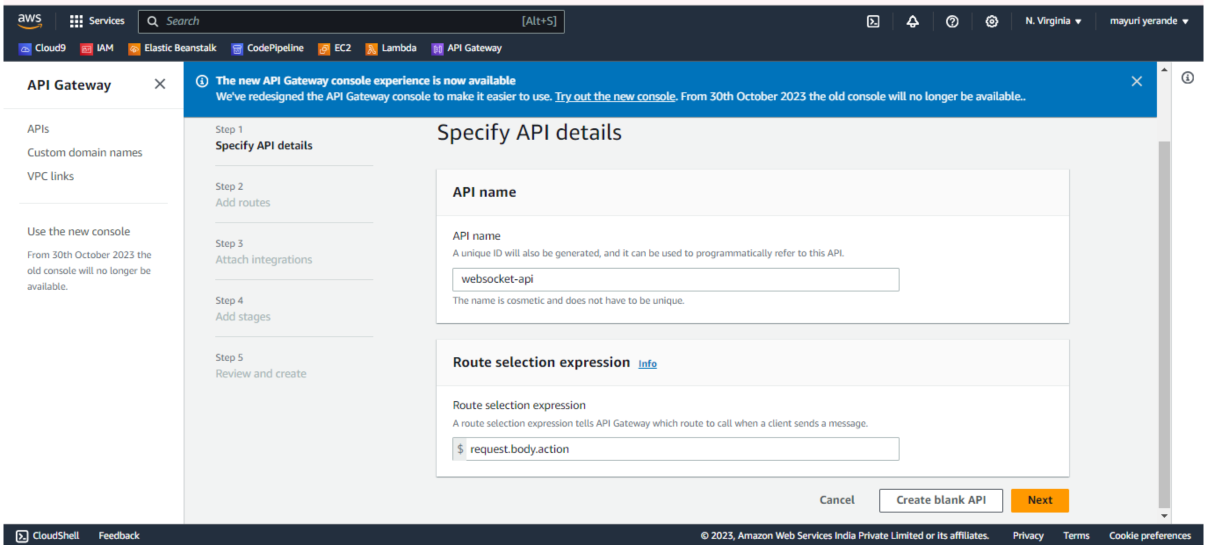
Build low-latency and cost-effective REST APIs with built-in features such as OIDC and OAuth2, and native CORS support.

Works with the following:
Lambda, HTTP backends

WebSocket API

Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.

Works with the following:
Lambda, HTTP, AWS Services



From 30th October 2023 the old console will no longer be available.

The new API Gateway console experience is now available
We've redesigned the API Gateway console to make it easier to use. Try out the new console. From 30th October 2023 the old console will no longer be available..

Step 1
Specify API details

Step 2
Add routes

Step 3
Attach integrations

Step 4
Add stages

Step 5
Review and create

Specify API details

API name

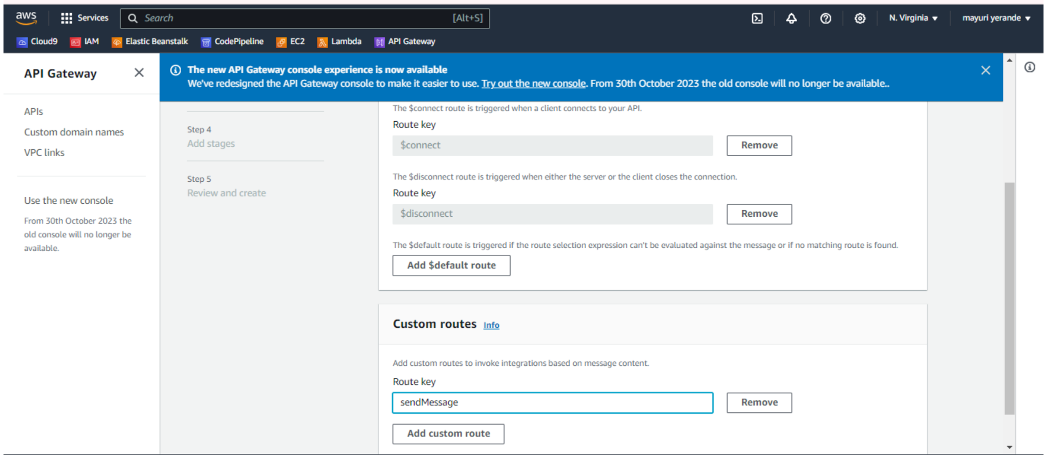
API name
A unique ID will also be generated, and it can be used to programmatically refer to this API.
websocket-api
The name is cosmetic and does not have to be unique.

Route selection expression

Route selection expression
A route selection expression tells API Gateway which route to call when a client sends a message.
$ request.body.action

© 2023, Amazon Web Services India Private Limited or its affiliates.

- Now we will add "sendMessage" route which will be custom route
- Click on next



From 30th October 2023 the old console will no longer be available.

The new API Gateway console experience is now available
We've redesigned the API Gateway console to make it easier to use. Try out the new console. From 30th October 2023 the old console will no longer be available..

Step 4
Add stages

Step 5
Review and create

The $connect route is triggered when a client connects to your API.
Route key
$connect

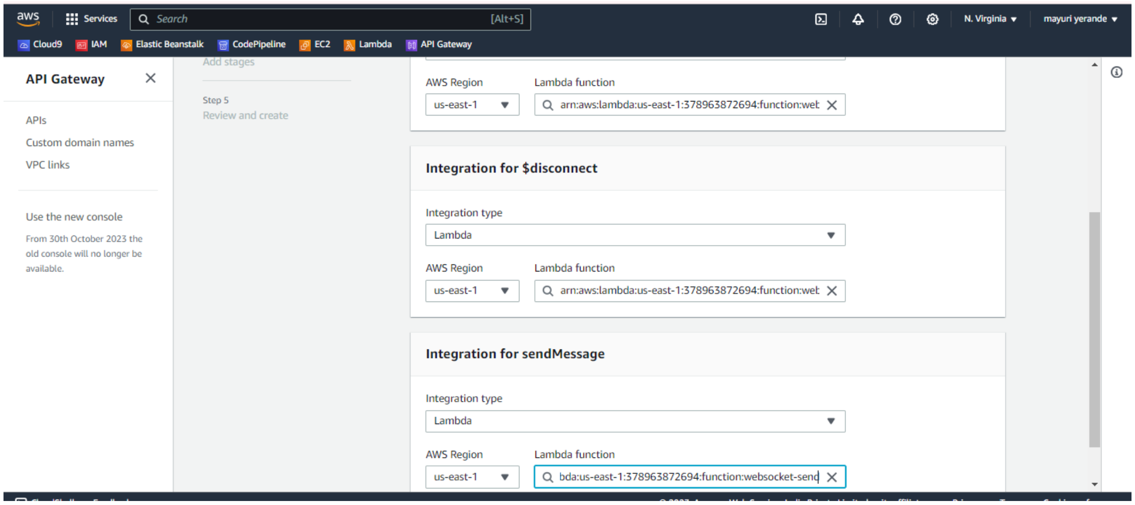
The $disconnect route is triggered when either the server or the client closes the connection.
Route key
$disconnect

The $default route is triggered if the route selection expression can't be evaluated against the message or if no matching route is found.

Custom routes

Add custom routes to invoke integrations based on message content.
Route key
sendMessage

- Now add all the lambda functions here to integrate



AWS Region: us-east-1
Lambda function: arn:aws:lambda:us-east-1:378963872694:function:wet

Integration for $disconnect

Integration type: Lambda

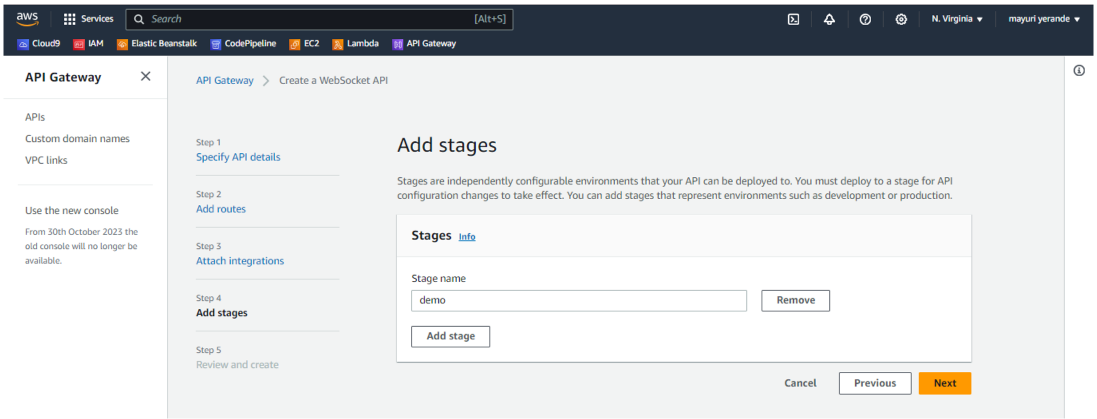
AWS Region: us-east-1
Lambda function: arn:aws:lambda:us-east-1:378963872694:function:wet

Integration for sendMessage

Integration type: Lambda

AWS Region: us-east-1
Lambda function: bda:us-east-1:378963872694:function:websocket-send

- Add the stage name(We named it “demo”)



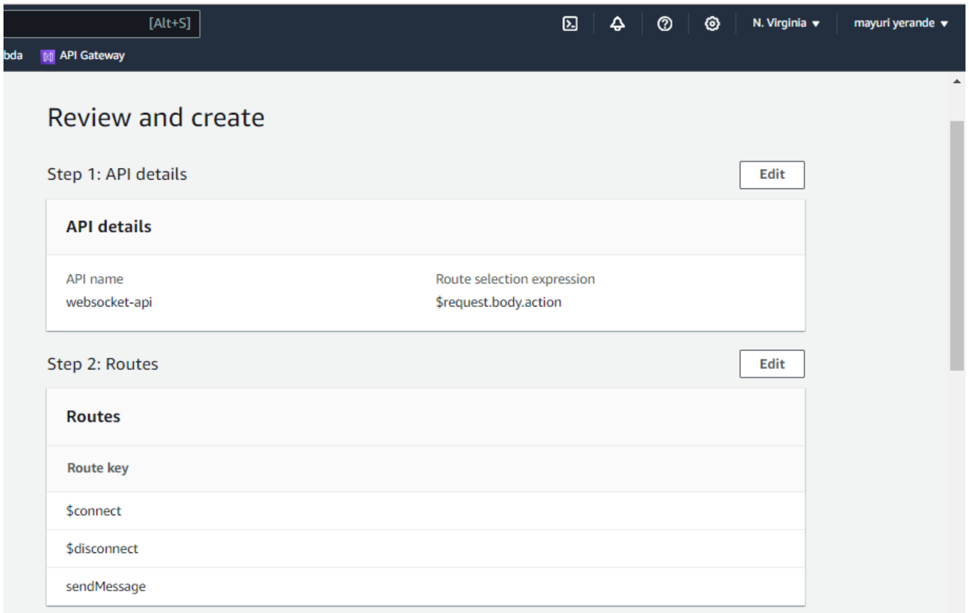
Add stages

Stages are independently configurable environments that your API can be deployed to. You must deploy to a stage for API configuration changes to take effect. You can add stages that represent environments such as development or production.

Stages Info

Stage name: demo

- Finally review all the details



Review and create

Step 1: API details

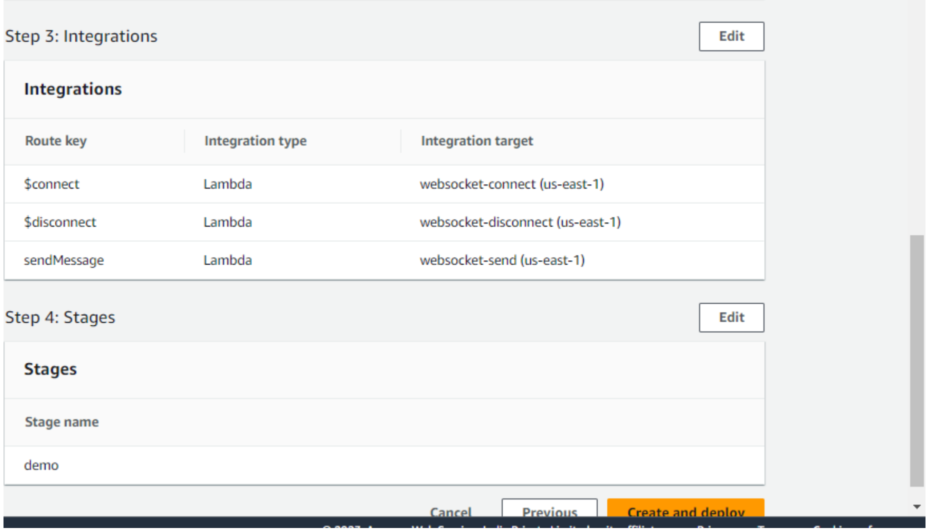
API details

API name	Route selection expression
websocket-api	$request.body.action

Step 2: Routes

Routes

Route key
$connect
$disconnect
sendMessage



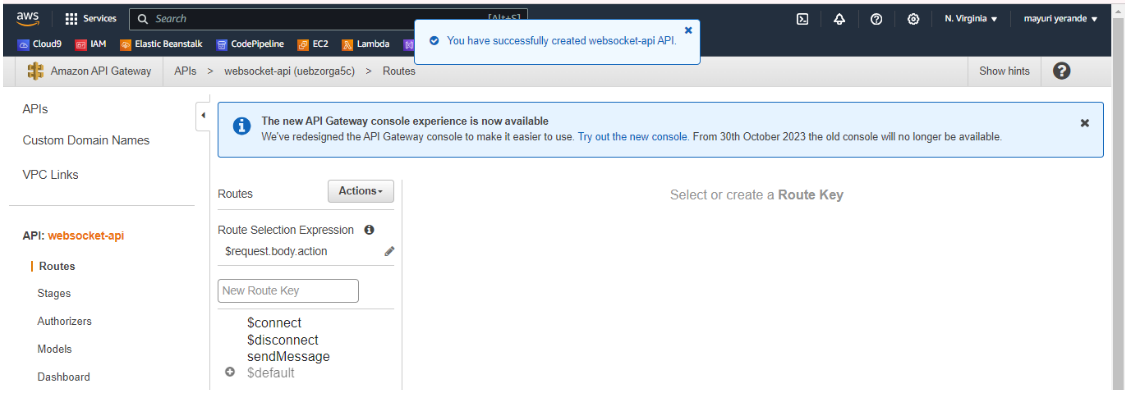
Integrations

Route key	Integration type	Integration target
$connect	Lambda	websocket-connect (us-east-1)
$disconnect	Lambda	websocket-disconnect (us-east-1)
sendMessage	Lambda	websocket-send (us-east-1)

Stages

Stage name
demo

- Our API Gateway is ready



You have successfully created websocket-api API.

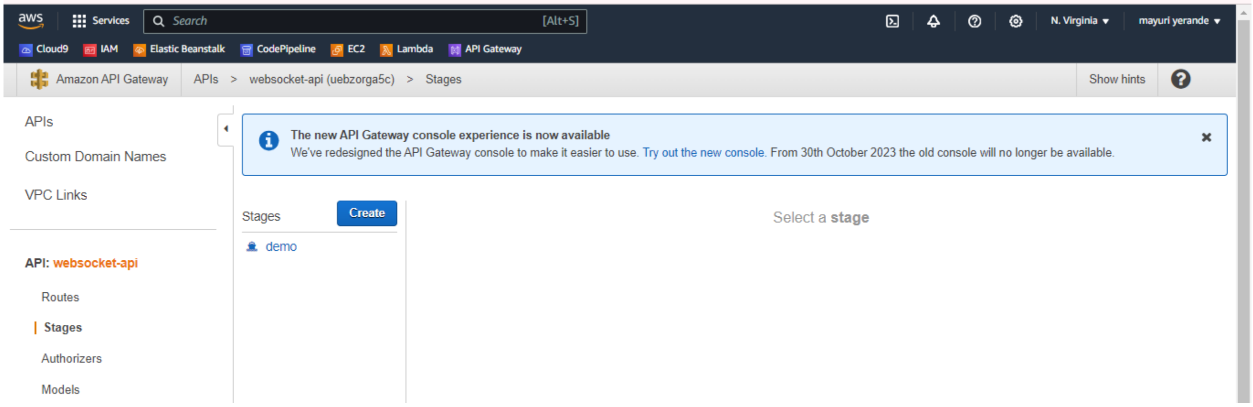
The new API Gateway console experience is now available
We've redesigned the API Gateway console to make it easier to use. Try out the new console. From 30th October 2023 the old console will no longer be available.

Route Selection Expression
$request.body.action

$connect
$disconnect
sendMessage
$default

Select or create a **Route Key**

- Now go to stages section, Your stage should be visible here



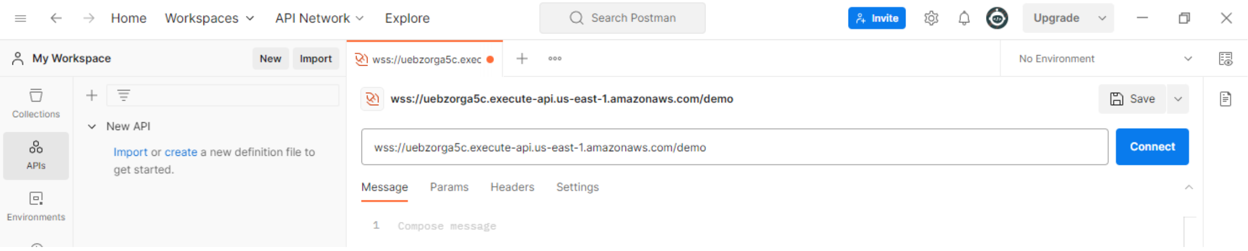
The new API Gateway console experience is now available
We've redesigned the API Gateway console to make it easier to use. Try out the new console. From 30th October 2023 the old console will no longer be available.

demo

Select a **stage**

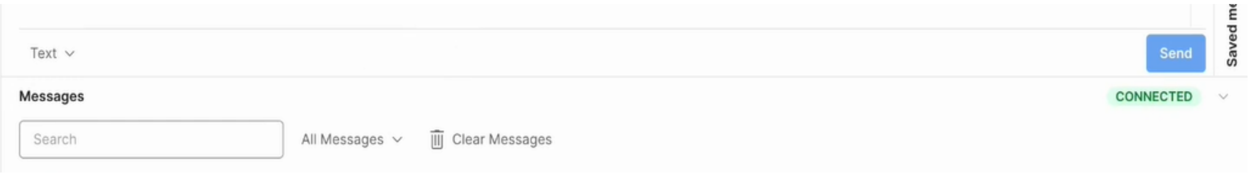
- Now we will be using postman api here
- Download postman: https://www.postman.com/downloads/
- Sign in with your account
- Go to “API” section
- Then choose “websocket” in postman

- Enter the websocket url in postman
- Click on “connect” and you will see that it gets connected.



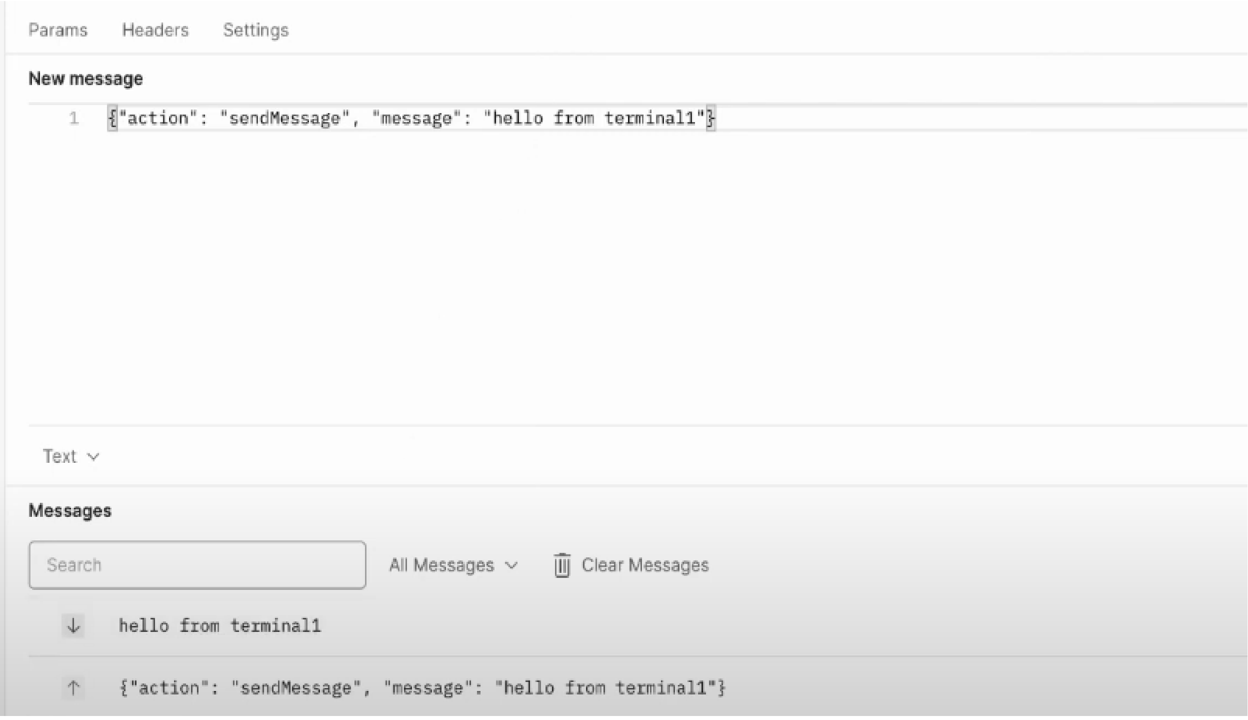
wss://uebzorga5c.execute-api.us-east-1.amazonaws.com/demo

Connect



CONNECTED

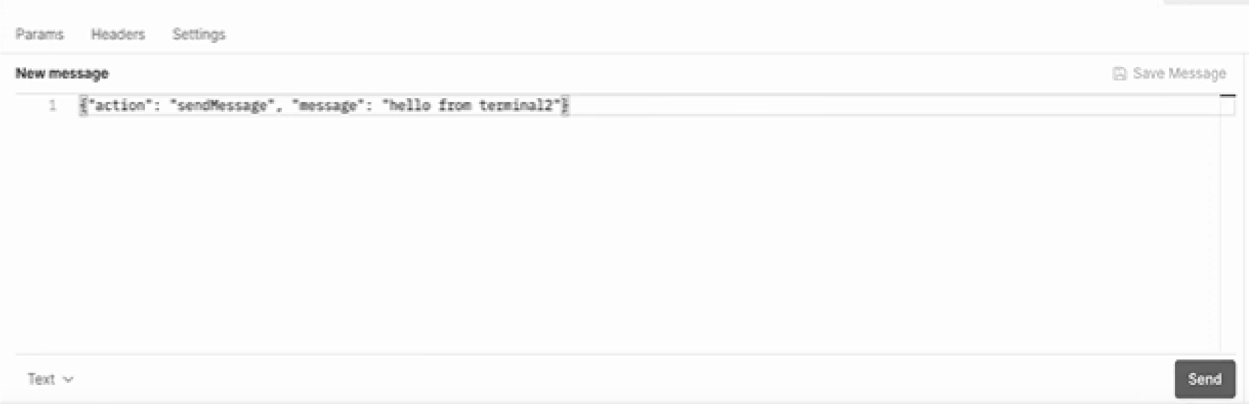
- Now we will send a message. Write the following command and you will see your message getting displayed on the terminal



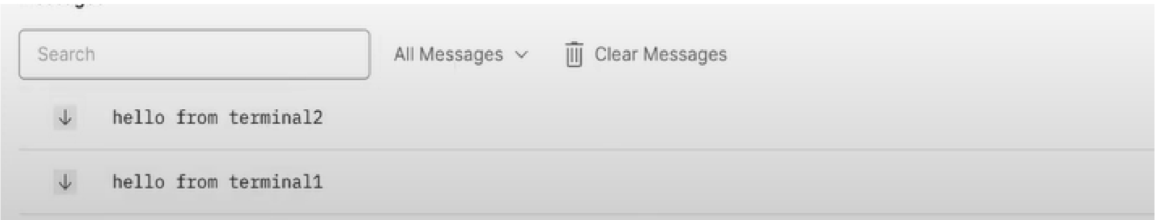
```
{"action": "sendMessage", "message": "hello from terminal1"}
```

```
↓ hello from terminal1
↑ {"action": "sendMessage", "message": "hello from terminal1"}
```

- We can add a new message



```
{"action": "sendMessage", "message": "hello from terminal2"}
```



```
hello from terminal2
hello from terminal1
```

CONCLUSION:

Thus we successfully implemented MQTT, HTTPS and WEBSOCKET protocols in AWS IOT. Also we implemented a publisher subscriber model in AWS IOT.