CSCI4145/5409: Serverless

Reminder: This is an individual assignment. You are not allowed to collaborate with anyone else when completing this assignment. You can borrow code and configuration snippets from internet sources that are not from students in this class, however that code must be cited and include comments for how you have modified the original code.

Introduction

This assignment will measure your understanding of some of the serverless mechanisms of our cloud provider AWS. This assignment assures us that you have attended the tutorials and learned about AWS Lambda and Step Functions, or that you have found some other way to learn these services. In addition, you will have to do some self-learning to study how to use AWS API Gateway to turn your lambda's and step functions into REST APIs, AWS Simple Queue Service (SQS) and about the MQ Telemetry Transport (MQTT) protocol commonly used in Internet of Things (IoT) devices.

Learning Outcomes

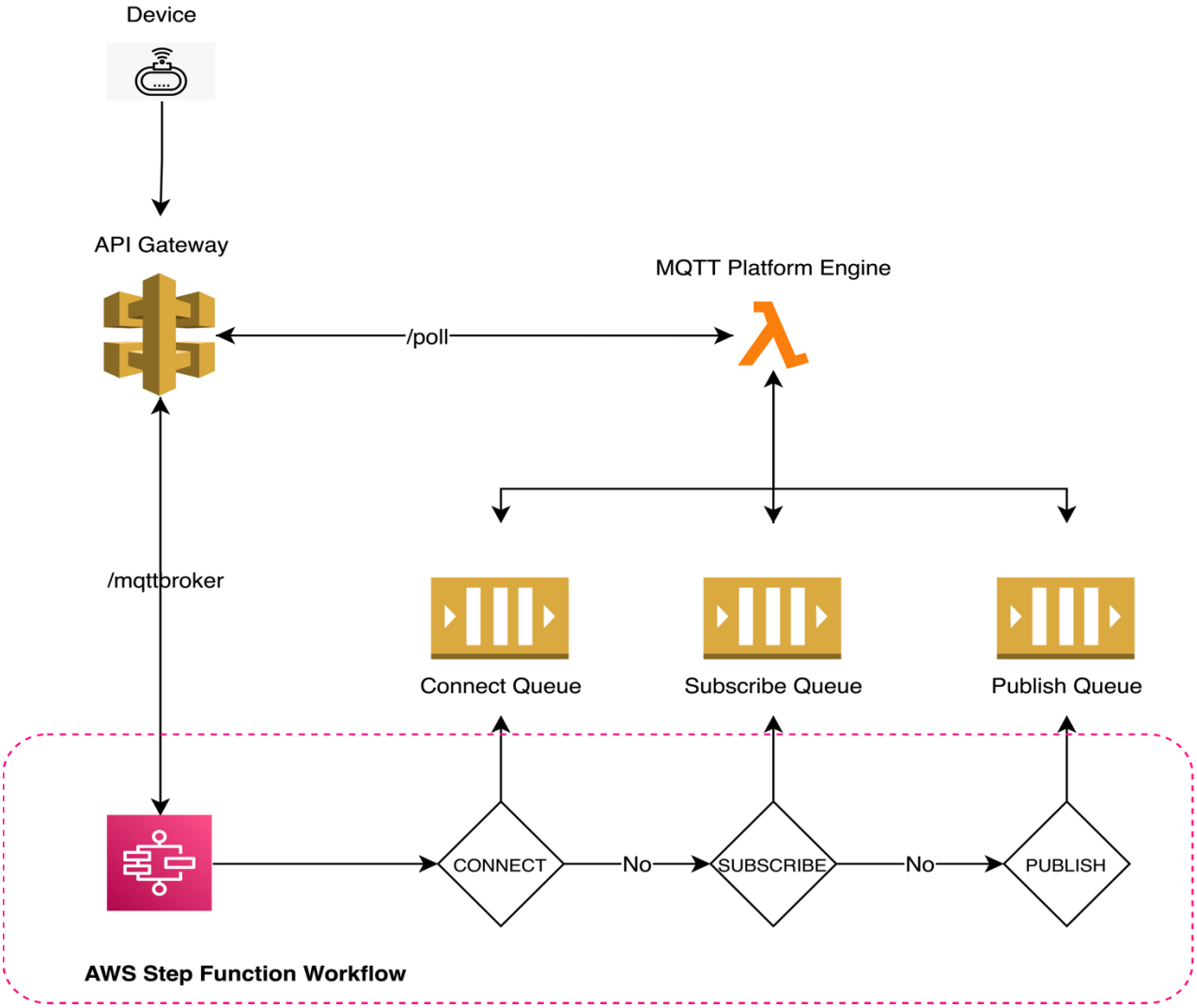
- Learn the benefits of serverless computing and apply that learning to implement a finite state machine using AWS Step Functions, serverless compute mechanisms (Lambda) and message buffering mechanisms (SQS).
- Learn AWS API Gateway to understand how to build serverless APIs in AWS.
- Learn about the MQTT protocol and how it works, as well as how to send and receive MQTT messages using AWS Lambda functions and how MQTT-based messages are integrated into a system using AWS Step Functions and SQS Queues.

Requirements

You will build API entry points for an IoT platform using serverless compute mechanisms. In a typical serverless IoT platform, a device (e.g., a mobile phone, smart watch, sensor, etc.) sends MQTT messages to an MQTT Broker instance and these messages are stored on different queues. Each queue has a process that consumes messages in the queue, processes the message, and sends an acknowledgement back to the device. We will again program a server which will automatically test your code by sending POST requests to each path on your API Gateway. Once you initiate the testing process it will test and then return a grade to you.

The entry point of your system will be your AWS Step Function (behind an API gateway) which will act as an MQTT Broker instance. The purpose of this AWS Step Function is to receive MQTT messages from devices and place the messages on a queue based on the message type (CONNECT, SUBSCRIBE or PUBLISH). You will perform polling from the queue(s) with the help of a lambda function, also accessed through your API gateway, that deletes the message from the queue and returns the acknowledgement as a response.

Here is a rough state diagram for your system:



MQTT Broker

Configure an API endpoint '**/mqttbroker**' using the API Gateway service for a step function you create. Your step function must accept MQTT based messages, and based on the message type choice place this message on the appropriate SQS Queue (Standard).

Note: A standard queue and a standard step function can be used in the system. You can refer to this link on how to configure API Gateway and Step Function. Also, make sure the response status code is 200.

- Connect Queue: When the message type is 'CONNECT', the state machine should send the message to this queue.
- Subscribe Queue: When the message type is 'SUBSCRIBE', the state machine should send the message to this queue.
- Publish Queue: When the message type is 'PUBLISH', state machine should send the message to this queue.

Expected request body (JSON) for MQTT Broker (/mqttbroker)

The following are **samples** that demonstrate the input formats for the JSON body of our **POST**s to your API. We will support whatever HTTP format you put in the `apiURL` field so feel free to use HTTP or the default HTTPS for API gateway.

CONNECT Message request:

```
{
    "type": "CONNECT",
    "username": "user1",
    "password": "pass123"
}
```

SUBSCRIBE Message request:

```
{
    "type": "SUBSCRIBE",
    "qos": 0
}
```

PUBLISH Message request:

```
{
    "type": "PUBLISH",
    "qos": 0,
    "payload": {
        "key": "location",
        "value": "44.637437,-63.587206"
    }
}
```

IoT Platform Engine (Lambda Function)

Create a lambda function configured with another API Gateway endpoint '**/poll**', this function will perform SQS polling by using the queue URLs of any one of the three Queues mentioned above and should delete the message from the queue. The JSON body of the request POSTed to this lambda function will contain a 'type' property which will be used to select which queue to poll from. After successfully polling and deleting from the appropriate queue, this lambda function should return an acknowledgement as a response with additional information.

Expected request and response (JSON) from IoT Platform Engine

Note that the JSON content in the request and response is case sensitive. And only 200 response status code is accepted.

To poll from 'CONNECT' queue (JSON request body we will send to you):

```
{
  "type": "CONNECT"
}
```

Response from lambda:

```
{
```

```
  "type": "CONNACK",
  "returnCode": 0,
  "username": "user1",
  "password": "pass123"
}
```

To poll from 'SUBSCRIBE' queue (JSON request body we will send to you):

```
{
  "type": "SUBSCRIBE"
}
```

Response from lambda:

```
{
  "type": "SUBACK",
  "returnCode":0
}
```

To poll from 'PUBLISH' queue (JSON request body we will send to you):

```
{
 "type": "PUBLISH"
}
```

Response from lambda:

```
{
 "type": "PUBACK",
 "returnCode": 0,
 "payload": {
  "key": "location",
  "value": "44.637437,-63.587206"
 }
}
```

Observe *that the message sent to MQTT broker is similar to the messages coming as response from IoT Platform Engine.*

How To Submit

There will be an API end point provided to you a few days before the deadline that will accept a POST request with the following JSON body:

```
{
  "arnBroker": "ARN_STEP_FUNCTION",
  "arnEngine": "ARN_FOR_MQTT_PLATFORM_ENGINE_LAMBDA_FUNC"
  "apiURL": "URL_FOR_API_GATEWAY_ENDPOINT",
  "bannerId": "BANNER_ID",
  "email": "rhawkey@dal.ca"
}
```

Please observe that we require the ARN (Amazon Resource Name) for your step function (arnBroker) and lambda (arnEngine). You can find these in the dashboard where you've provisioned them. We'll verify these ARNs to ensure that you have completed the assignment with serverless mechanisms that *you* have provisioned, not say a Flask API running on EC2.

Initiate a call to this API and our server will test your system and return a grade and feedback to you in plain text. You must build, provision, and test your app and finally make your POST to our provided API **before the assignment deadline.**

Create a folder in your repository labeled **A4**. Put your lambda source code (no matter what language you wrote it in) in this folder. Also include in the folder either a screenshot of your Step Function implementation or a text export. Push these files to your individual repository on gitlab **before the assignment deadline.**

Marking Rubric

For successfully executing the state machine for all the three messages (CONNECT, SUBSCRIBE and PUBLISH) you will earn 30%. For successfully polling all 3 message types and returning correct acknowledgement message responses you can earn up to 70%. Detailed marking based on the type of MQTT message is as follows,

CONNECT_TEST = 10%
SUBSCRIBE_TEST = 10%
PUBLISH_TEST = 10%

CONNACK_TEST = 20%
SUBACK_TEST = 20%
PUBACK_TEST = 30%