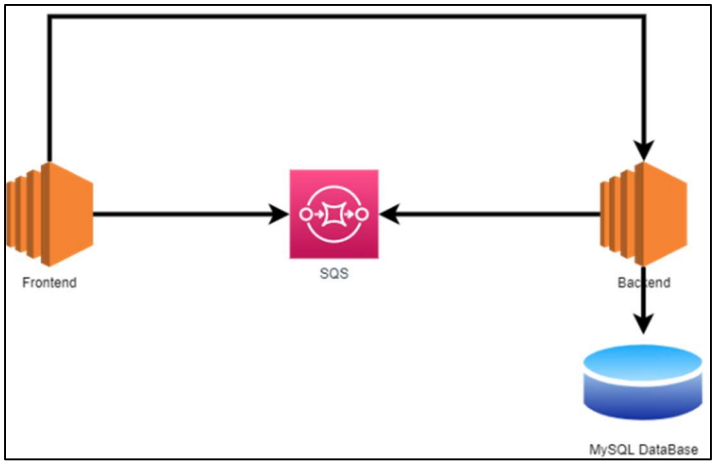
Project 1: Managing tightly coupled architecture using Amazon SQS

Description:

In the traditional way of building applications, the applications directly talk to each other. if an application is down, it impacts the other linked applications, and some data might be compromised. This is called tightly coupled architecture.

Applications cannot communicate directly with each other. This can be done through AWS SQS, which makes applications highly available. In the below diagram, the Customer Web Applications interacts with the Backend Applications via SQS Queue. For some reason, if the backend applications are down, the Customer Web Application can continue working with the messages being buffered in the SQS Queue. Once the backend application is up, it can start polling the messages from the SQS Queue and update the database. This way, none of the messages are lost, and applications are loosely coupled and not aware of the status of each other.



Tools required:

AWS Services: SQS, EC2, IAM, and RDS

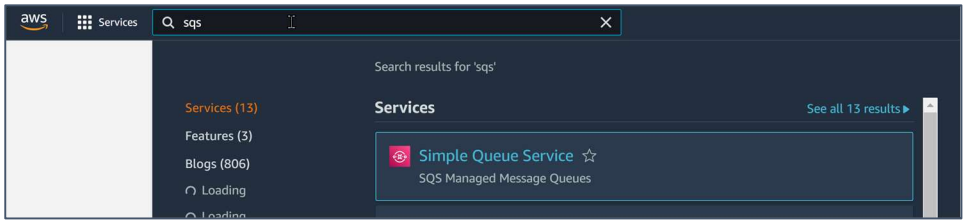
Steps executed:

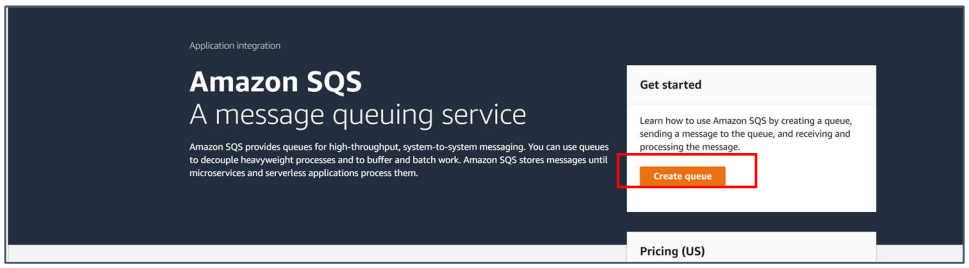
We are using a python script to mimic the frontend (webserver) and backend (database) to see the messages being sent and received. The received data will be pushed to MySQL Database (Amazon RDS).

- Configure SQS service
- Configure RDS
- IAM
    - User: setup a user to
    - Policies: will be used in Roles (Two policies, one to send messages and other to receive & delete msg)
    - Roles: Different resources in AWS to communicate to each other
- Frontend EC2 instance: to send data
- Backend EC2 instance: to receive and push data to DB and then delete.

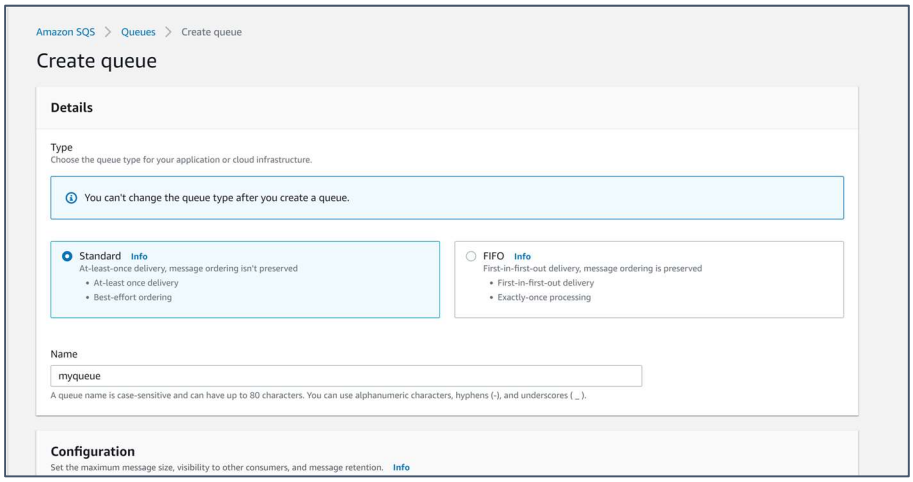
Configure SQS

Navigate to SQS and create queue

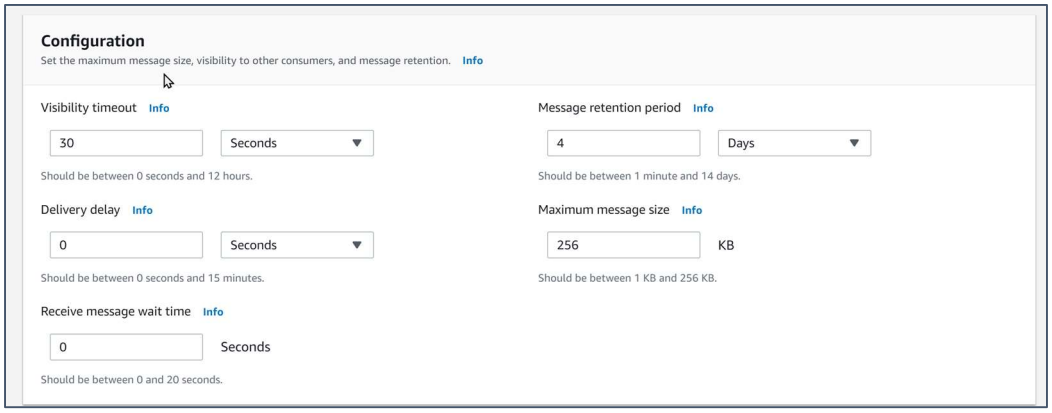




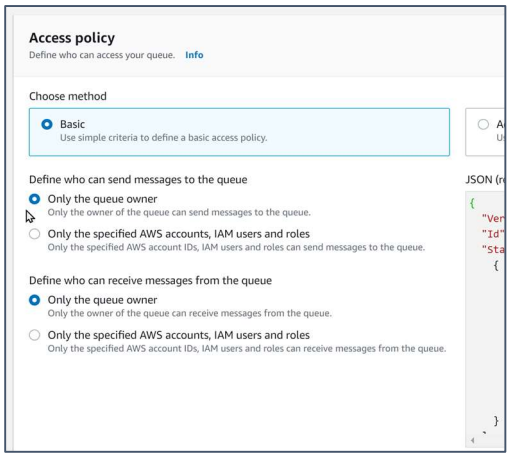
Create a Standard queue

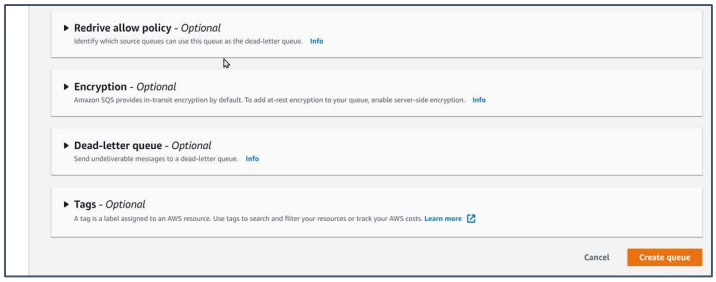


Keep the defaults for the configuration section (since we use it for a demo)

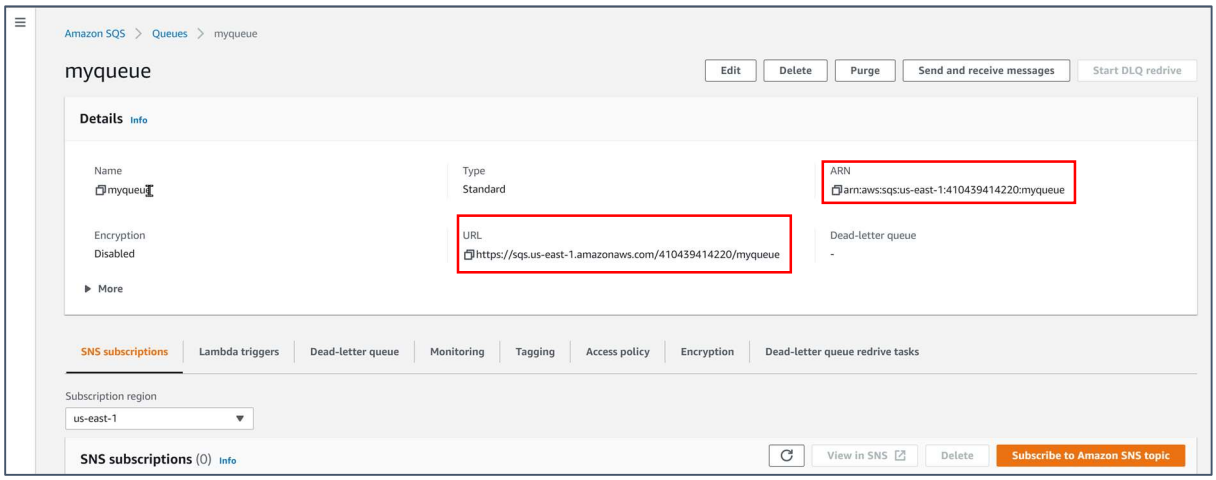


Keep the access policy and the other options to defaults.





Make a note of the ARN and the URL for further use in the project.



NOTE: The following resources' configuration are provided in a separate document and the working of SQS queue will presented after this block.

RDS Configuration - MySQL database will be configured to store the received messages

EC2 Instances Configuration – 3 separate instances will be created to mimic front-end, backend and the other to use as mysql client connecting to RDS

IAM – Users, Policies and Roles created to allow communication between different AWS resources.

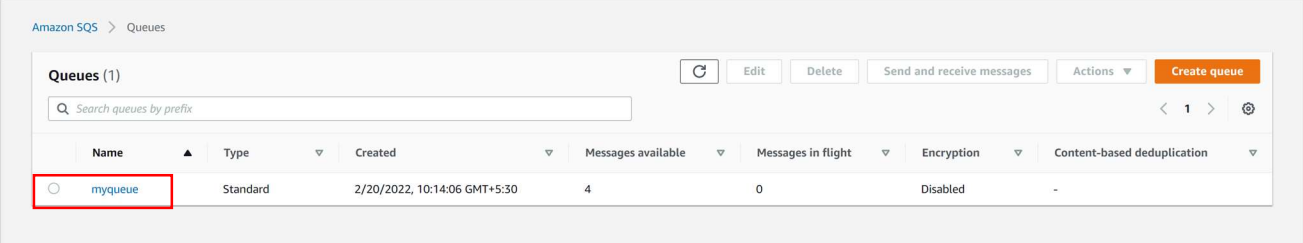
Source code and related documentation available on github.
https://github.com/devops4eng/AWS-Dev-Associate

SQS Testing:

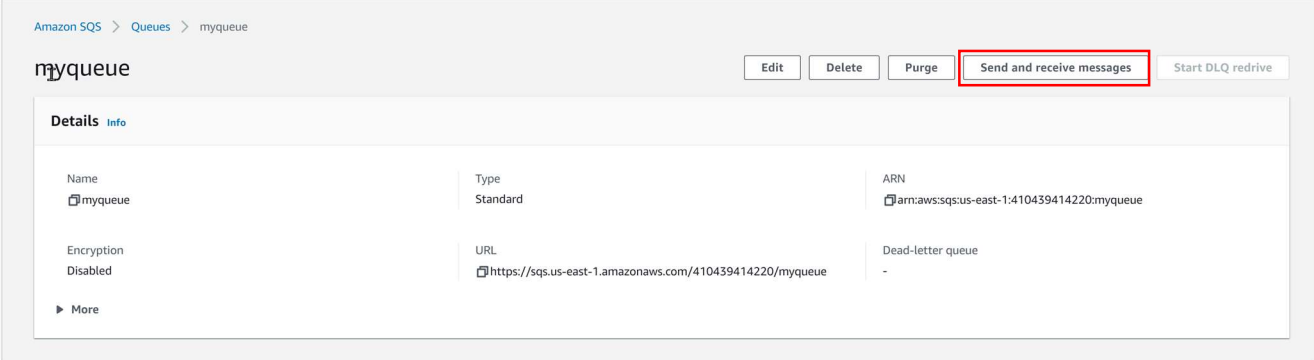
On frontend EC2 instance, send some messages as a comma separated value.

```
ubuntu@ip-172-31-23-255:~$ python3 send.py Jane,UK
8fb1b177-a341-4c02-a85d-5e5d74df6e07
ubuntu@ip-172-31-23-255:~$ python3 send.py Mark,USA
27f1ed7f-ab28-4993-ac21-aa2063cb8fee
ubuntu@ip-172-31-23-255:~$ python3 send.py Tim,Australia
b9dd1b18-9f61-4795-9f05-75e90cee2dcd
ubuntu@ip-172-31-23-255:~$ python3 send.py Lee,China
af92a718-6cf1-47a8-ab86-465895ccc5c5
ubuntu@ip-172-31-23-255:~$
```

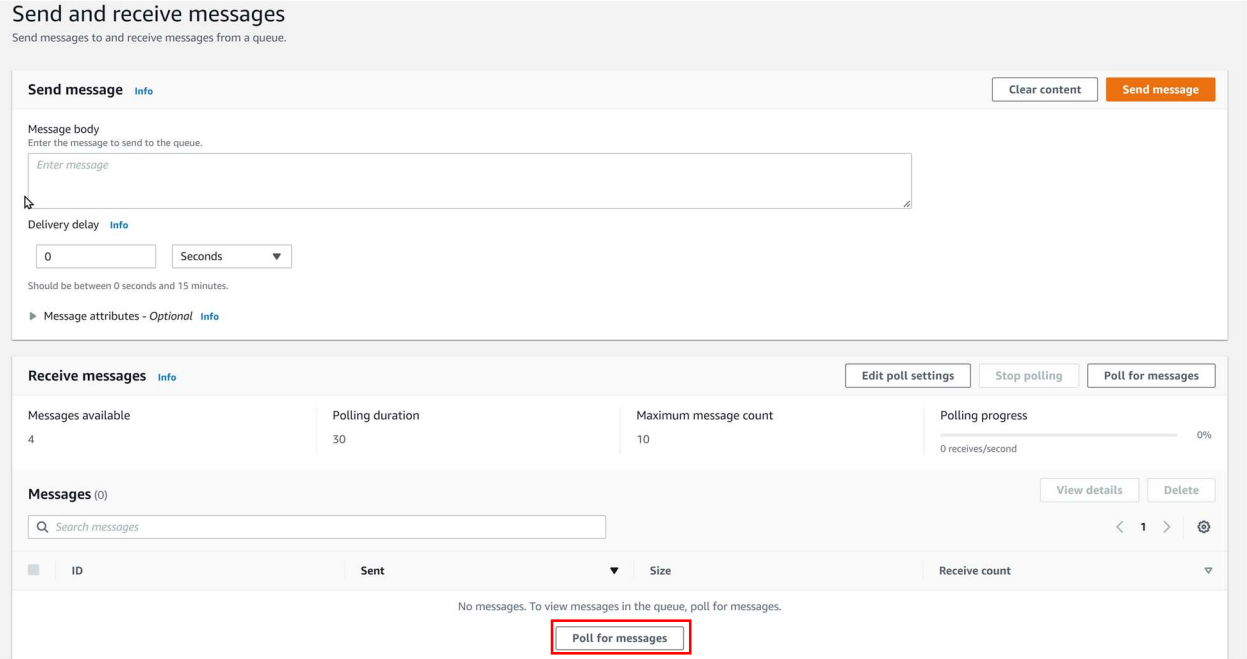
On SQS dashboard, click on the queue created “myqueue”



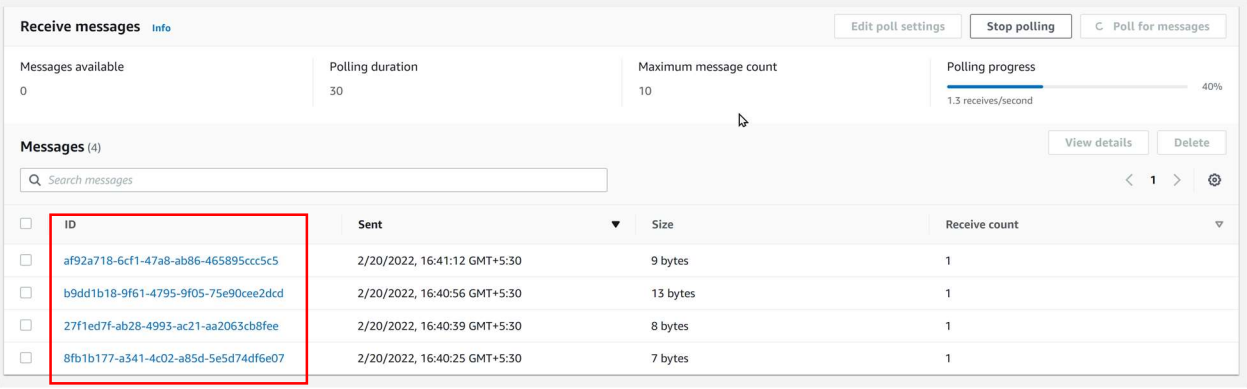
On the details page of “myqueue” click on Send and receive messages



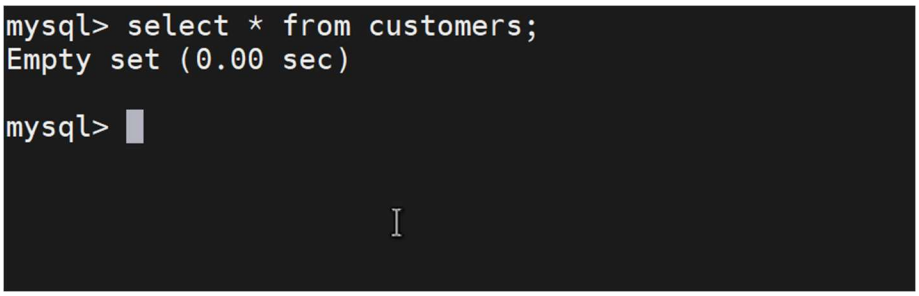
Poll for messages under send and receive messages.



Notice the messages received



At this time, the messages are on the backend, and not yet pushed to RDS.



On the backend server, execute the python script to receive the messages.

```
ubuntu@ip-172-31-30-112:~$ python3 receive.py
```

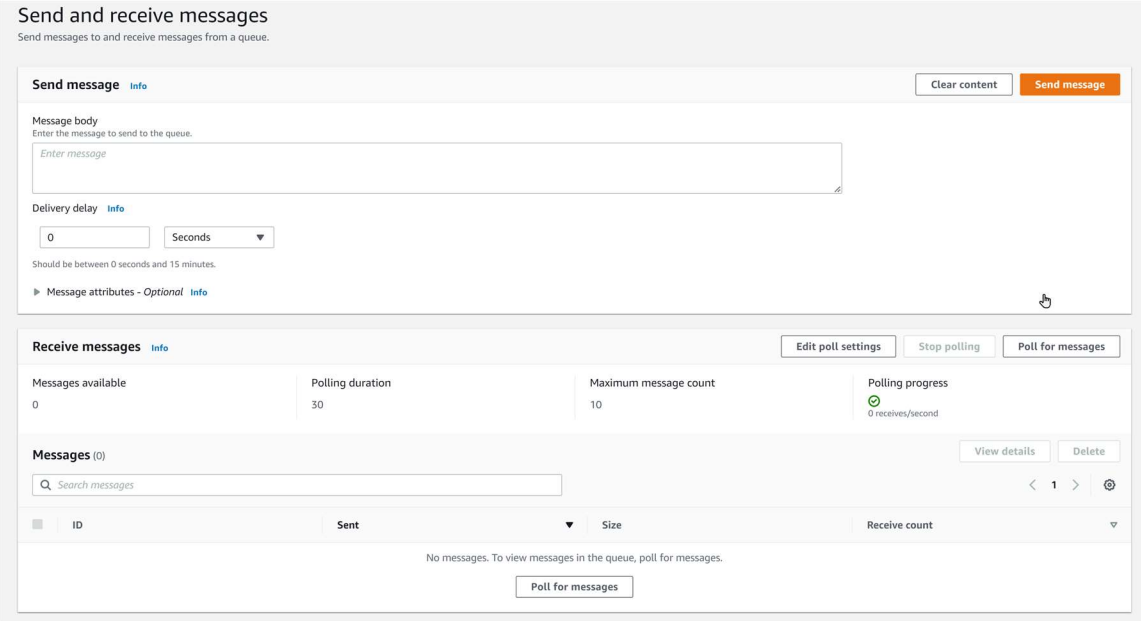
Execute the script until all the messages are received.

```
ubuntu@ip-172-31-30-112:~$ python3 receive.py
Received and deleted message: Mark,USA
Record inserted in the DB
ubuntu@ip-172-31-30-112:~$ python3 receive.py
Received and deleted message: Jane,UK
Record inserted in the DB
ubuntu@ip-172-31-30-112:~$ python3 receive.py
Received and deleted message: Lee,China
Record inserted in the DB
ubuntu@ip-172-31-30-112:~$ python3 receive.py
Received and deleted message: Tim,Australia
Record inserted in the DB
ubuntu@ip-172-31-30-112:~$
ubuntu@ip-172-31-30-112:~$ python3 receive.py
Traceback (most recent call last):
  File "receive.py", line 20, in <module>
    message = response['Messages'][0]
KeyError: 'Messages'
ubuntu@ip-172-31-30-112:~$
```

Once all the messages are received, we notice this message.

```
Traceback (most recent call last):
  File "receive.py", line 20, in <module>
    message = response['Messages'][0]
KeyError: 'Messages'
```

Also, the messages that are received are now pushed to RDS.



Now we see the data inserted in the database.

```
mysql> select * from customers;
Empty set (0.00 sec)

mysql> select * from customers;
+------+-----------+
| name | country   |
+------+-----------+
| Mark | USA       |
| Jane | UK        |
| Lee  | China     |
| Tim  | Australia |
+------+-----------+
4 rows in set (0.00 sec)

mysql>
```

Using **DBeaver** tool to establish a connection externally and verify the Database.