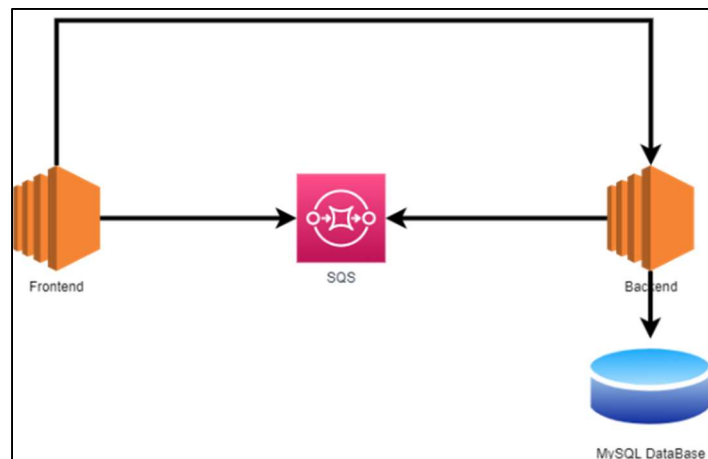


Project 1: Managing tightly coupled architecture using Amazon SQS

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

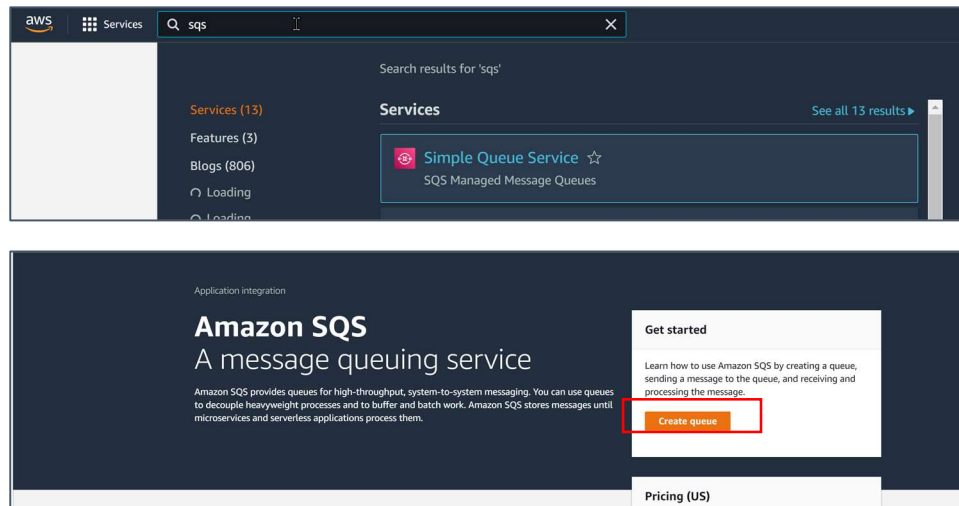
Steps executed:

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

The image shows the 'Create queue' page in the AWS Management Console. The 'Details' section is active, showing the 'Type' dropdown set to 'Standard'. The 'Name' field is filled with 'myqueue'. The 'Configuration' section is visible at the bottom, showing the default settings for visibility timeout, message retention period, delivery delay, and receive message wait time.

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

The image shows the 'Configuration' section of the 'Create queue' page. It displays the default settings for the queue configuration, including visibility timeout, message retention period, delivery delay, and receive message wait time. The settings are as follows:

Configuration	Value	Unit	Range
Visibility timeout	30	Seconds	Should be between 0 seconds and 12 hours.
Message retention period	4	Days	Should be between 1 minute and 14 days.
Delivery delay	0	Seconds	Should be between 0 seconds and 15 minutes.
Maximum message size	256	KB	Should be between 1 KB and 256 KB.
Receive message wait time	0	Seconds	Should be between 0 and 20 seconds.

Keep the access policy and the other options to defaults.

Access policy
Define who can access your queue. [Info](#)

Choose method

☒ Basic
Use simple criteria to define a basic access policy.

☐ Advanced
Use JSON to define an access policy.

Define who can send messages to the queue

☒ Only the queue owner
Only the owner of the queue can send messages to the queue.

☐ Only the specified AWS accounts, IAM users and roles
Only the specified AWS account IDs, IAM users and roles can send messages to the queue.

Define who can receive messages from the queue

☒ Only the queue owner
Only the owner of the queue can receive messages from the queue.

☐ Only the specified AWS accounts, IAM users and roles
Only the specified AWS account IDs, IAM users and roles can receive messages from the queue.

Redrive allow policy - Optional
Identify which source queues can use this queue as the dead-letter queue. [Info](#)

Encryption - Optional
Amazon SQS provides in-transit encryption by default. To add at-rest encryption to your queue, enable server-side encryption. [Info](#)

Dead-letter queue - Optional
Send undeliverable messages to a dead-letter queue. [Info](#)

Tags - Optional
A tag is a label assigned to an AWS resource. Use tags to search and filter your resources or track your AWS costs. [Learn more](#)

Cancel **Create queue**

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

Amazon SQS > Queues > myqueue

myqueue Edit Delete Purge Send and receive messages Start DLQ redrive

Details [Info](#)

Name myqueue	Type Standard	ARN arn:aws:sqs:us-east-1:410439414220:myqueue
Encryption Disabled	URL https://sqs.us-east-1.amazonaws.com/410439414220/myqueue	Dead-letter queue -

► More

SNS subscriptions Lambda triggers Dead-letter queue Monitoring Tagging Access policy Encryption Dead-letter queue redrive tasks

Subscription region
us-east-1

SNS subscriptions (0) [Info](#) View in SNS Delete **Subscribe to Amazon SNS topic**

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

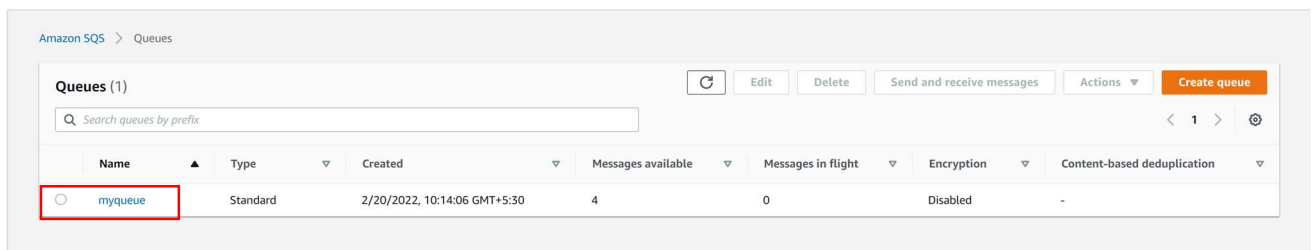
<https://github.com/devops4eng/AWS-Dev-Associate>

SQS Testing:

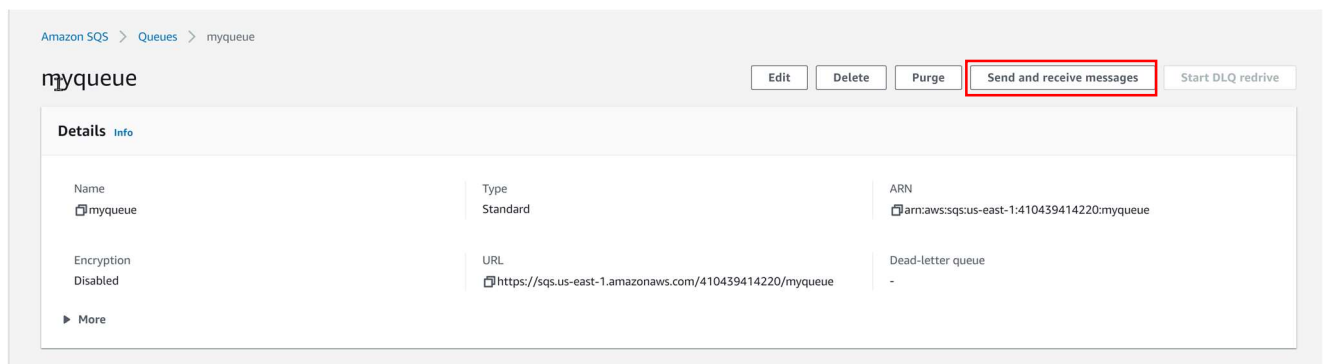
On frontend EC2 instance, send some messages as a comma separated value.

```
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.

Send and receive messages

Send messages to and receive messages from a queue.

Send message [Info](#)

Clear content

Send message

Message body

Enter the message to send to the queue.

Enter message

Delivery delay [Info](#)

0

Seconds

Should be between 0 seconds and 15 minutes.

► Message attributes - Optional [Info](#)

Receive messages [Info](#)

Edit poll settings

Stop polling

Poll for messages

Messages available

4

Polling duration

30

Maximum message count

10

Polling progress

0 receives/second

0%

Messages (0)

View details

Delete

Search messages

< 1 >

ID	Sent	Size	Receive count
No messages. To view messages in the queue, poll for messages.			

Poll for messages

Notice the messages received

Receive messages [Info](#)

Edit poll settings

Stop polling

C Poll for messages

Messages available

0

Polling duration

30

Maximum message count

10

Polling progress

1.3 receives/second

40%

Messages (4)

View details

Delete

Search messages

< 1 >

ID	Sent	Size	Receive count
af92a718-6cf1-47a8-ab86-465895ccc5c5	2/20/2022, 16:41:12 GMT+5:30	9 bytes	1
b9dd1b18-9f61-4795-9f05-75e90cee2dcd	2/20/2022, 16:40:56 GMT+5:30	13 bytes	1
27f1ed7f-ab28-4993-ac21-aa2063cb8fee	2/20/2022, 16:40:39 GMT+5:30	8 bytes	1
8fb1b177-a341-4c02-a85d-5e5d74df6e07	2/20/2022, 16:40:25 GMT+5:30	7 bytes	1

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

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

mysql> 
```

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.

Send and receive messages

Send messages to and receive messages from a queue.

Send message Info

Clear content Send message

Message body
Enter the message to send to the queue.

Delivery delay Info
 Seconds
Should be between 0 seconds and 15 minutes.

► Message attributes - Optional Info

Receive messages Info

Edit poll settings Stop polling Poll for messages

Messages available
0

Polling duration
30

Maximum message count
10

Polling progress
0 receives/second

Messages (0)

ID Sent Size Receive count

No messages. To view messages in the queue, poll for messages.

Poll for messages

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.

The screenshot displays the DBeaver 21.3.4 interface. The left sidebar shows the 'Database Navigator' with a tree view of the database structure. The 'customers' table is selected under the 'customer' database. The main window shows the SQL Editor with the query 'SELECT * FROM customers;'. The results are displayed in a table view at the bottom right, which is highlighted with a red box. The table has two columns: 'name' and 'country'. The data rows are: Mark (USA), Jane (UK), Lee (China), and Tim (Australia).

	name	country
1	Mark	USA
2	Jane	UK
3	Lee	China
4	Tim	Australia