Simple Queue Service (SQS)

Amazon Simple Queue Service (SQS) is a fully managed message queuing service provided by Amazon Web Services (AWS). It enables decoupling of the components of a cloud application by allowing them to communicate asynchronously. SQS acts as a mediator for different parts of a distributed system, allowing messages to be sent between various components without direct connections between them.

Here are some key features and concepts related to Amazon SQS:

1. Message Queues:

- SQS uses message queues to transmit any volume of data, at any level of throughput, without the need for additional infrastructure.
- Messages can be up to 256 KB in size.

2. Queue Types:

- **Standard Queue:** Offers a high throughput, best-effort delivery, and messages are delivered at least once. However, occasional duplication of messages is possible.
- **FIFO Queue (First-In-First-Out):** Guarantees that messages are processed exactly once and in the order they are sent.

3. Message Lifecycle:

- Messages are sent to a queue by a producer and then retrieved and processed by a consumer.
- Once a message is successfully processed, it can be deleted from the queue.
- If a message is not deleted, it will eventually become visible again for retrieval after a specified visibility timeout period.

4. Visibility Timeout:

- When a message is retrieved from the queue, it becomes invisible to other consumers for a specified duration called the visibility timeout.
- This allows the consumer enough time to process the message without worrying about other consumers processing the same message.

5. Scaling:

- SQS automatically scales to handle the volume of messages without requiring intervention.
- It can be used to decouple the components of a cloud application, allowing each component to scale independently.

6. Retention Period:

- Messages are retained in the queue for a specified duration (the retention period) from the time of message enqueue.

7. Dead-Letter Queues:

- Provides a mechanism to capture and store messages that cannot be processed successfully after a certain number of attempts.
- Helps in troubleshooting and debugging.

8. Integration with Other AWS Services:

- SQS can be easily integrated with other AWS services such as AWS Lambda, Amazon EC2, Amazon S3, and more.

9. Security and Access Control:

- SQS provides fine-grained access control using AWS Identity and Access Management (IAM).
- Messages can be encrypted in transit using HTTPS and at rest using AWS Key Management Service (KMS).

Amazon SQS is a reliable, scalable, and fully managed service that facilitates the building of distributed and decoupled systems in the AWS cloud environment. It is commonly used in scenarios such as task scheduling, event notification, and handling asynchronous workflows in applications.

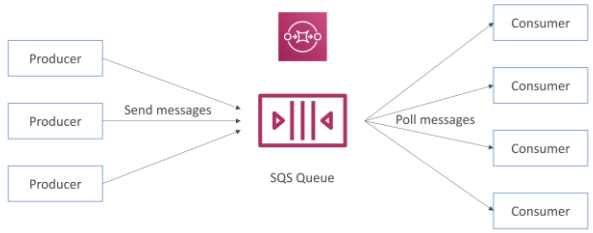
Basic information

- When we start deploying multiple applications, they will inevitably need to communicate with one another
- There are two patterns of application communication

- Synchronous between applications can be problematic if there are sudden spikes of traffic
- What if you need to suddenly encode 1000 videos but usually it's 10?

- In that case, it's better to **decouple** your applications,
  - using SQS: queue model
  - using SNS: pub/sub model
  - using Kinesis: real-time streaming model
- These services can scale independently from our application!

What is Queue:



SQS- Standard Queue

- Oldest offering (over 10 years old)
- Fully managed service, used to **decouple applications**

- Attributes:
  - Unlimited throughput, unlimited number of messages in queue
  - Default retention of messages: 4 days, maximum of 14 days
  - Low latency (<10 ms on publish and receive)
  - Limitation of 256KB per message sent

- Can have duplicate messages (at least once delivery, occasionally)
- Can have out of order messages (best effort ordering)

SQS- Producing Message

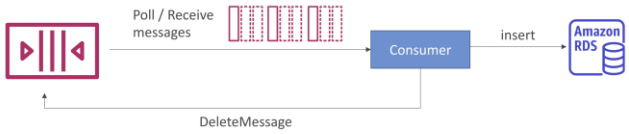
- Produced to SQS using the SDK (SendMessage API)
- The message is **persisted** in SQS until a consumer deletes it
- Message retention: default 4 days, up to 14 days

- Example: send an order to be processed
  - Order id
  - Customer id
  - Any attributes you want

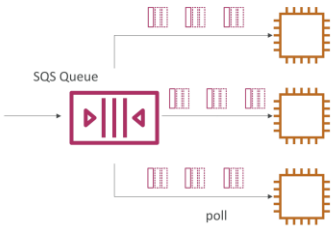
- SQS standard: unlimited throughput

SQS- Consuming Message

- Consumers (running on EC2 instances, servers, or AWS Lambda)…
- Poll SQS for messages (receive up to 10 messages at a time)
- Process the messages (example: insert the message into an RDS database)
- Delete the messages using the DeleteMessage API

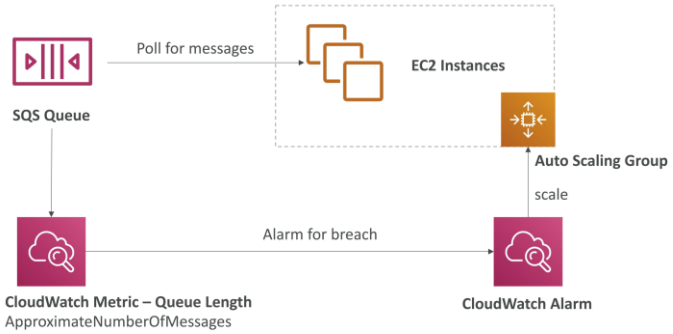


SQS - Multiple Ec2 Instances Consumers

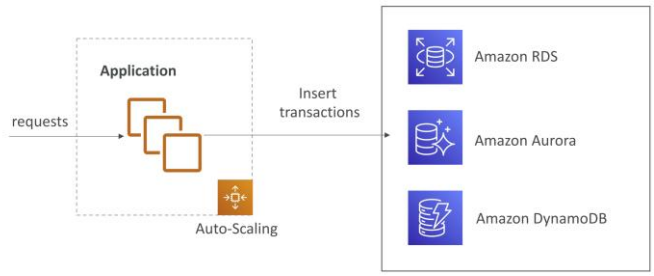


- Consumers receive and process messages in parallel
- At least once delivery
- Best-effort message ordering
- Consumers delete messages after processing them
- We can scale consumers horizontally to improve throughput of processing

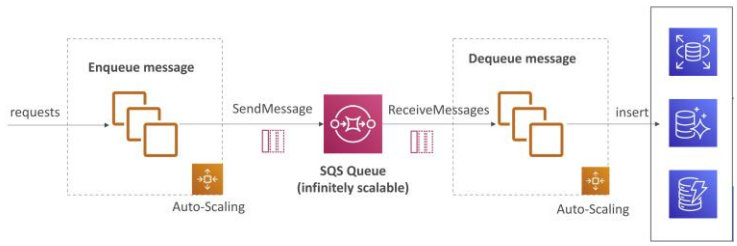
SQS – with Auto scaling Group (ASG)



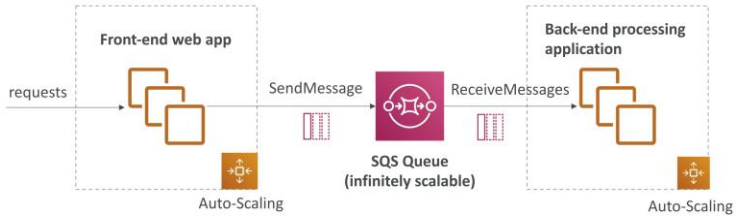
SQS – if the load is too big, some transactions may be lost



SQS – as a buffer to database writes



SQS – to decouple between applications tires:



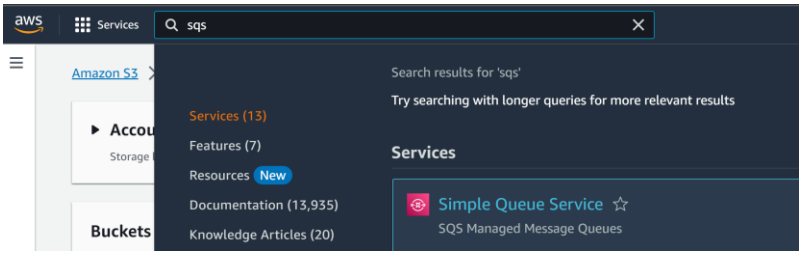
SQS – Security

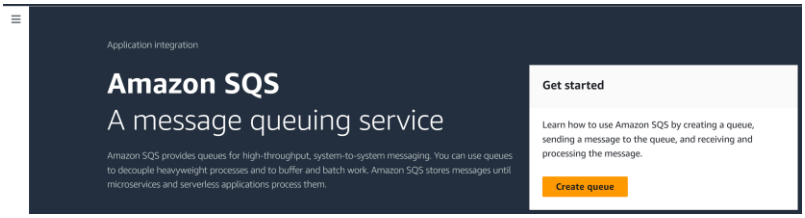
- Encryption:
  - In-flight encryption using HTTPS API
  - At-rest encryption using KMS keys
  - Client-side encryption if the client wants to perform encryption/decryption itself

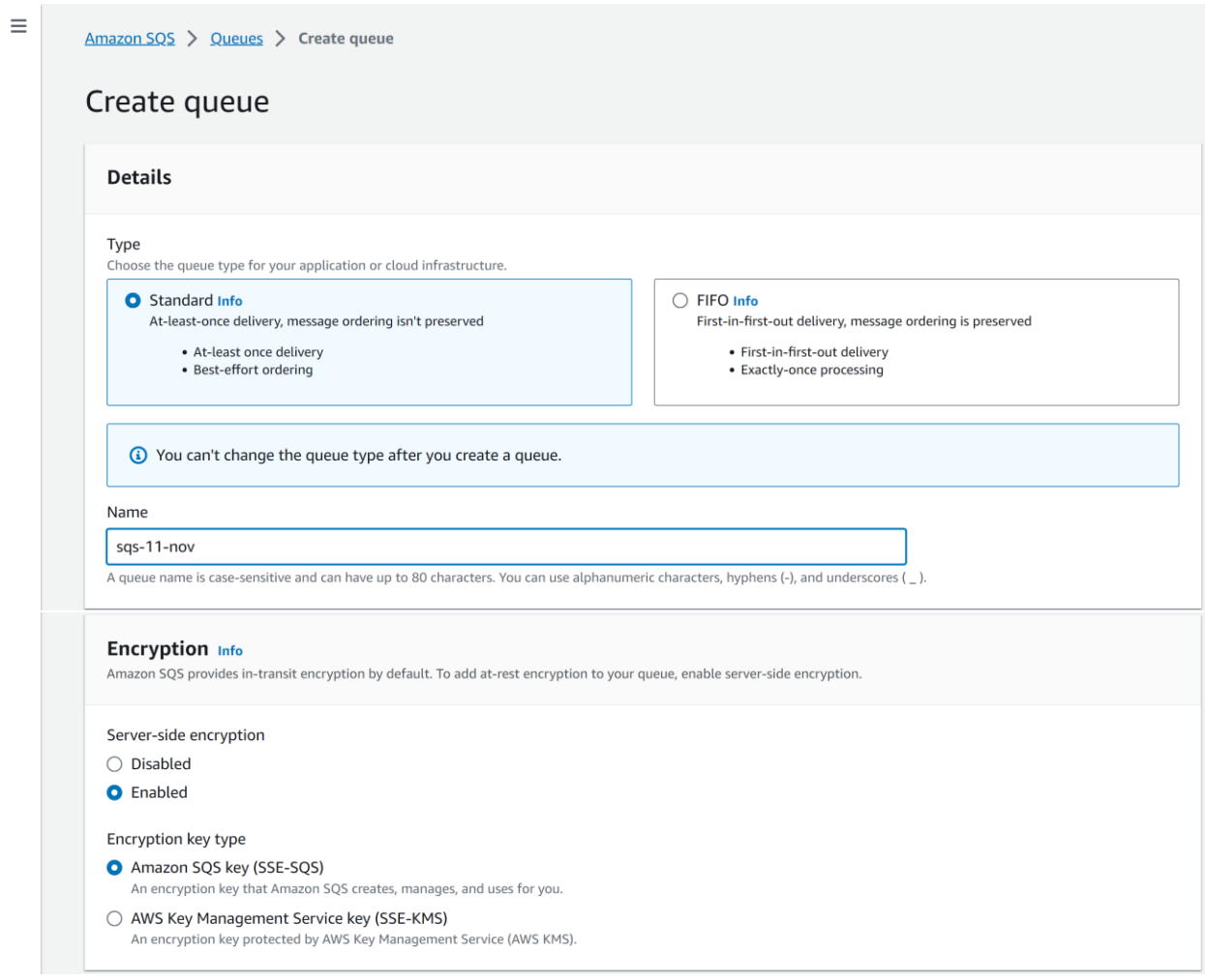
- Access Controls: IAM policies to regulate access to the SQS API

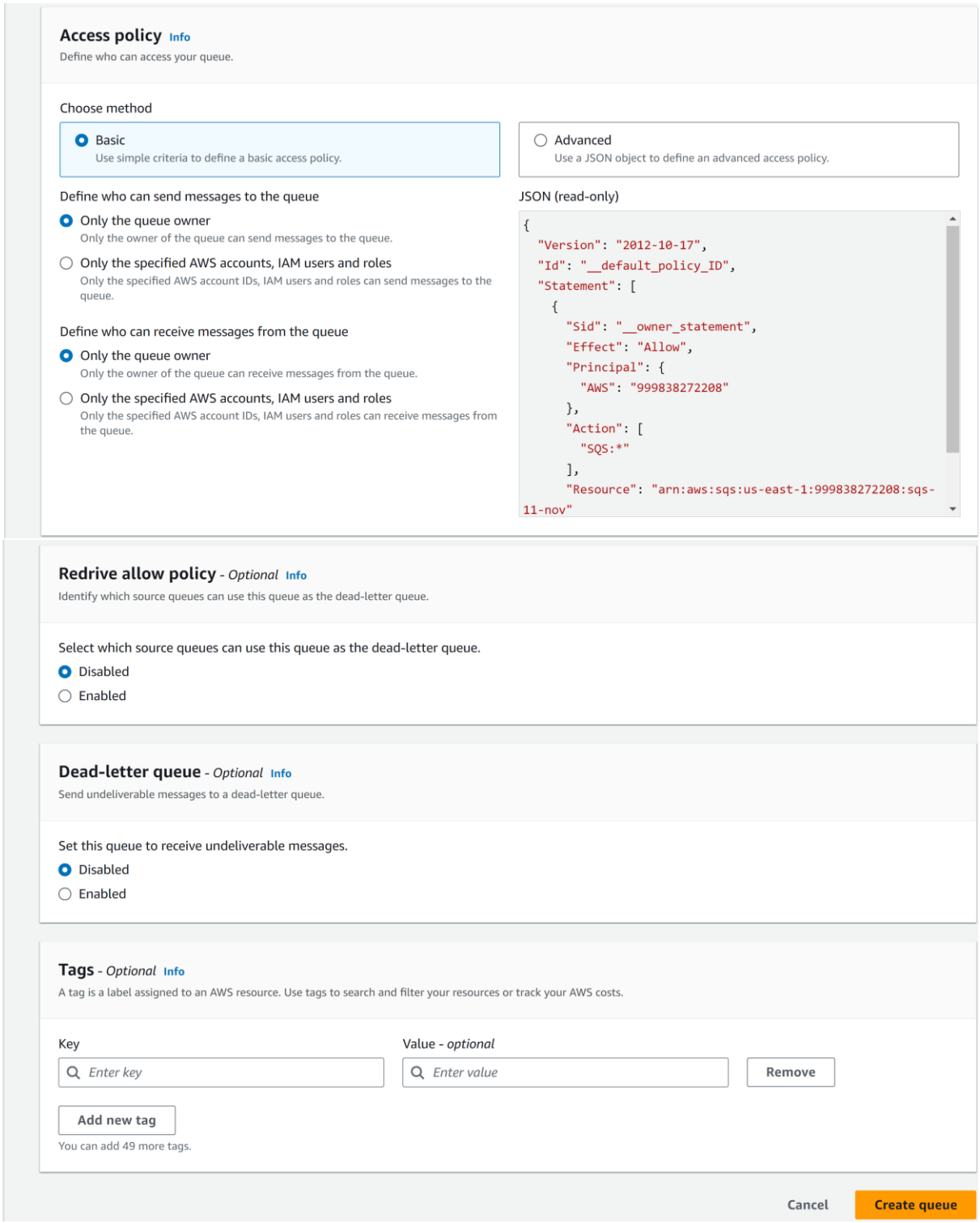
- SQS Access Policies (similar to S3 bucket policies)
  - Useful for cross-account access to SQS queues
  - Useful for allowing other services (SNS, S3…) to write to an SQS queue

Lab: create SQS then send and pull messages

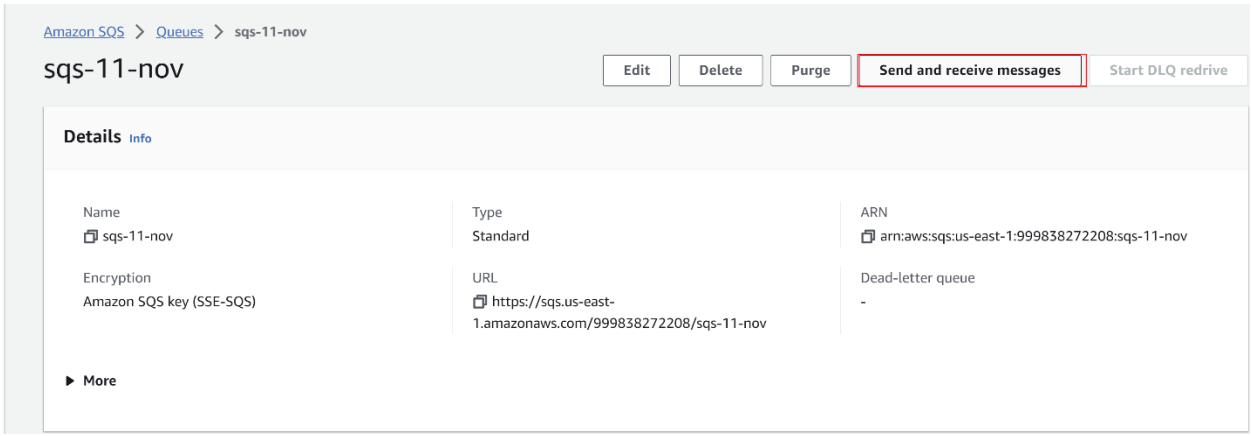




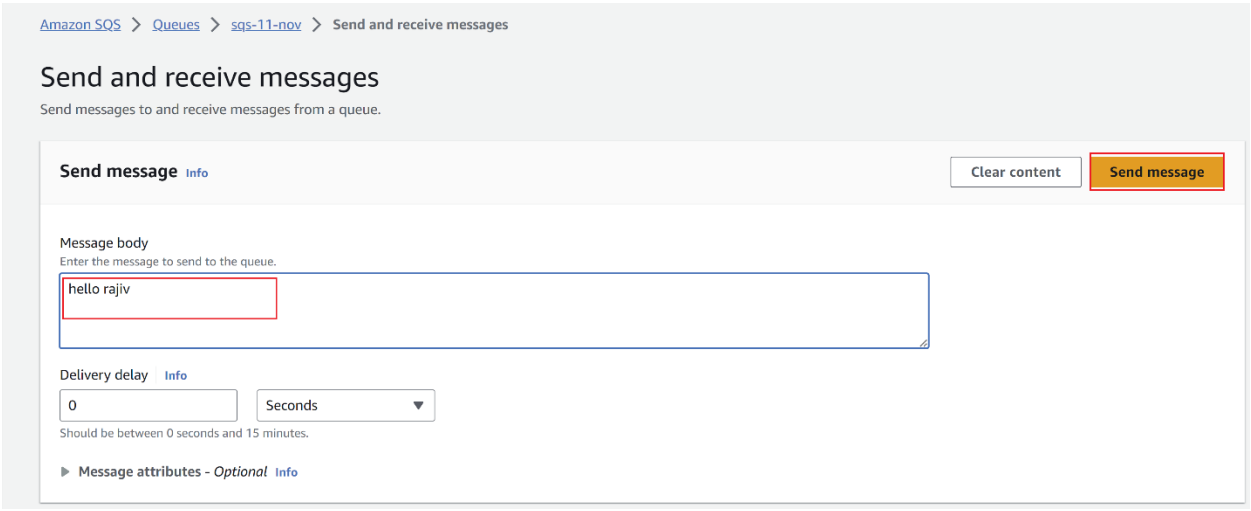

Amazon SQS > Queues > Create queue
Create queue
Details
Type
Choose the queue type for your application or cloud infrastructure.
Standard Info
At-least-once delivery, message ordering isn't preserved
• At-least once delivery
• Best-effort ordering
FIFO Info
First-in-first-out delivery, message ordering is preserved
• First-in-first-out delivery
• Exactly-once processing
You can't change the queue type after you create a queue.
Name
sqs-11-nov
A queue name is case-sensitive and can have up to 80 characters. You can use alphanumeric characters, hyphens (-), and underscores (_).
Encryption Info
Amazon SQS provides in-transit encryption by default. To add at-rest encryption to your queue, enable server-side encryption.
Server-side encryption
Disabled
Enabled
Encryption key type
Amazon SQS key (SSE-SQS)
An encryption key that Amazon SQS creates, manages, and uses for you.
AWS Key Management Service key (SSE-KMS)
An encryption key protected by AWS Key Management Service (AWS KMS).



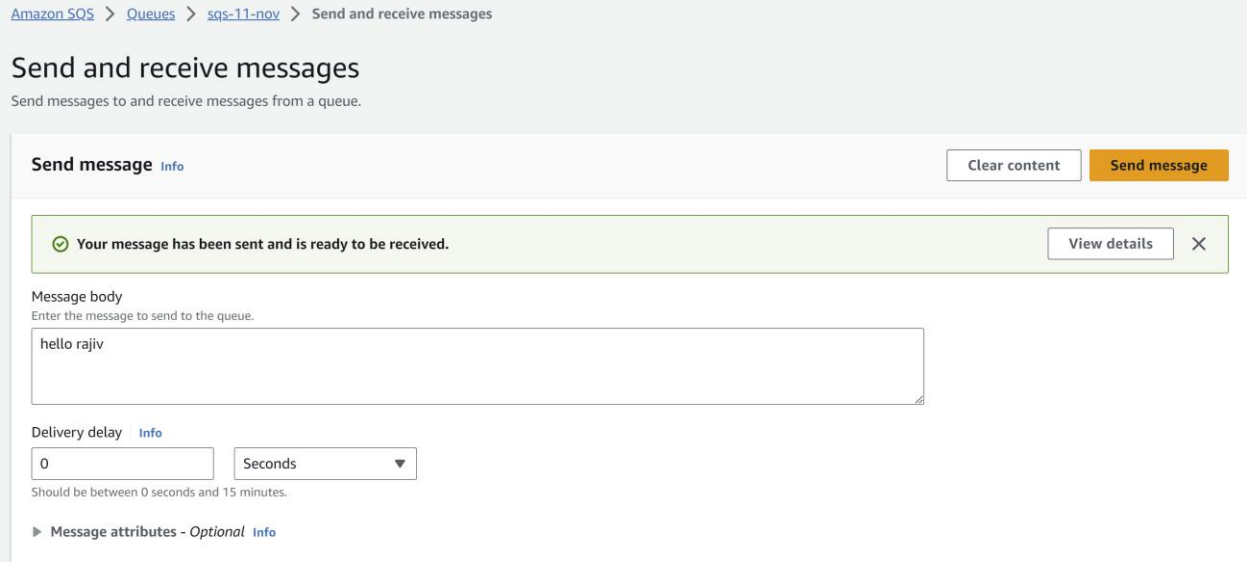
After creating SQS



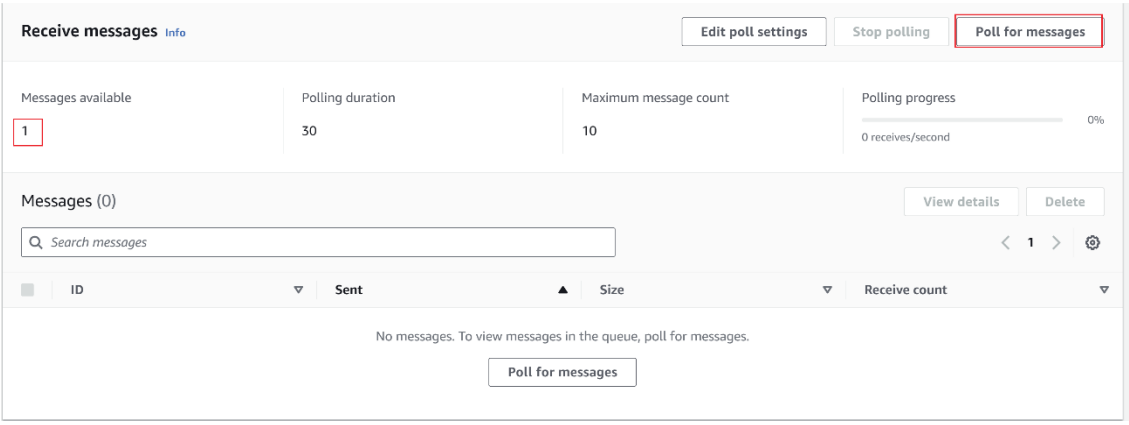
After click send and receive message we the following screen now we write a message and send it now if go down and we can see the message count 1 and if we wait some time we can see the count is increase because as long as we read and delete the message it will sending. Now pull and delete the message now again pull the message and we can see now no new message is coming.

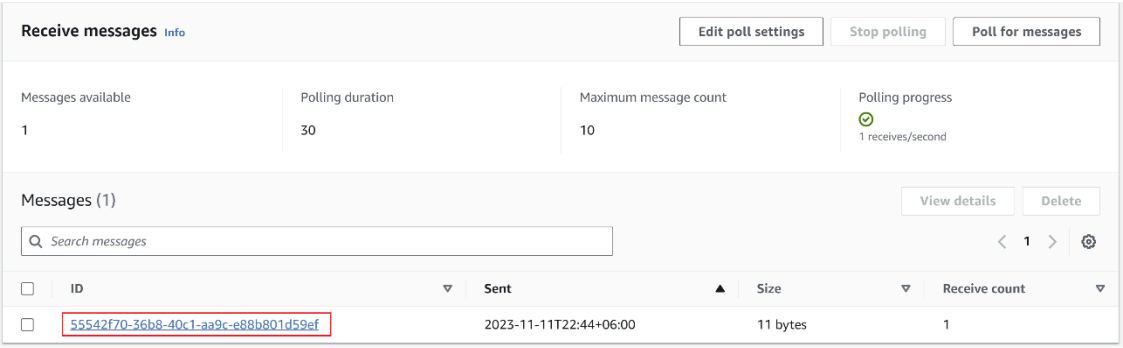


After click send message

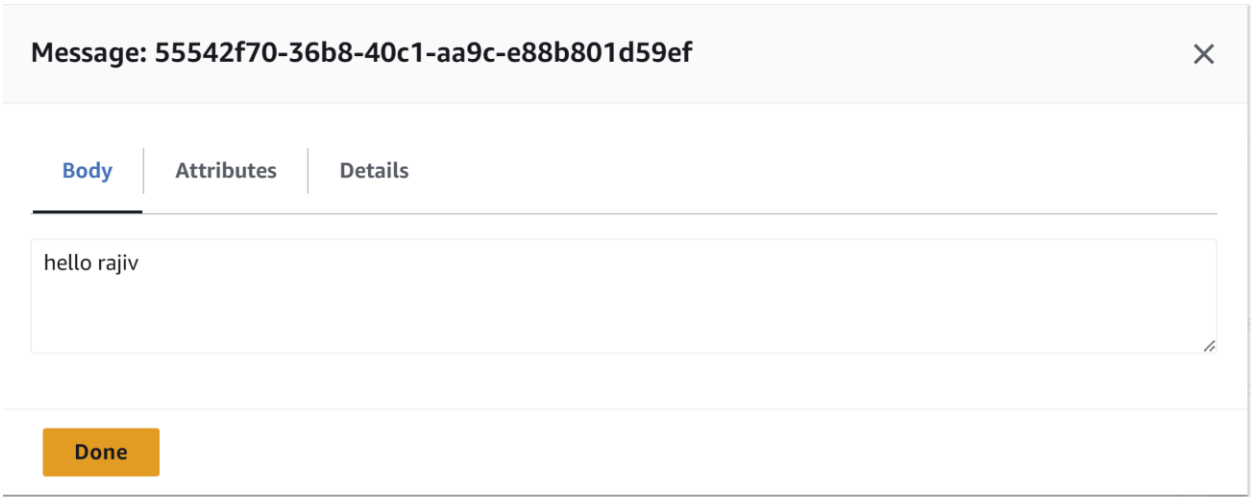


Now scroll down for getting the message and also pull the message





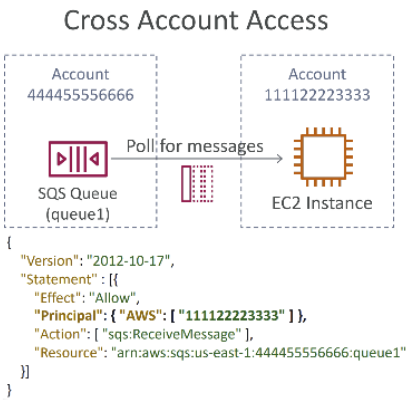
After click the message

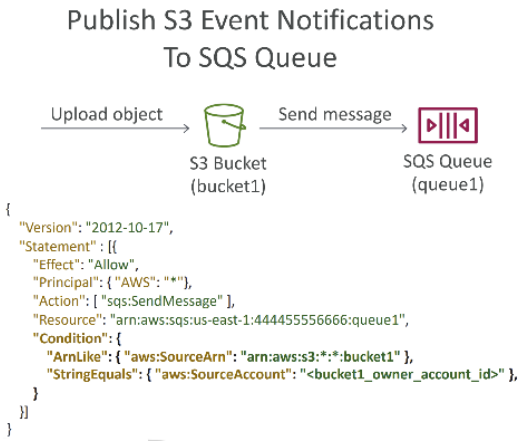


After that if you did not delete the message and refresh the page it will send message again

So delete the message then it will not send any message again.

SQS-Queue Access policy



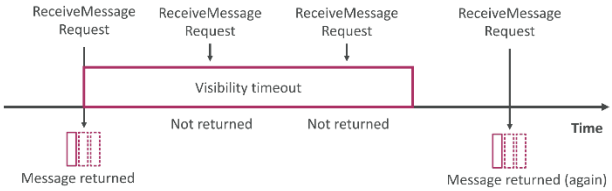


Lab: SQS access policy

Create one bucket
Create a SQS and create a access policy to allow that bucket to send messages to this SQS
now if we upload or download any image to this bucket it will send a message to this SQS
We already see the lab in S3 class

SQS – Message visibility Timeout

- After a message is polled by a consumer, it becomes **invisible** to other consumers
- By default, the "message visibility timeout" is **30 seconds**
- That means the message has 30 seconds to be processed
- After the message visibility timeout is over, the message is "visible" in SQS



- If a message is not processed within the visibility timeout, it will be processed **twice**
- A consumer could call the **ChangeMessageVisibility** API to get more time
- If visibility timeout is high (hours), and consumer crashes, re-processing will take time
- If visibility timeout is too low (seconds), we may get duplicates

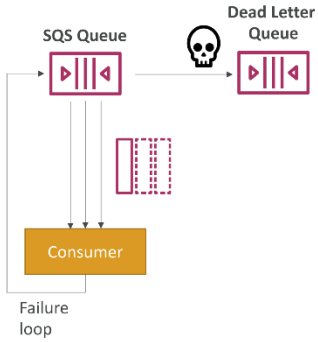
Lab: SQS visibility timeout

Create an SQS and open the SQS in two windows now send a message from the first widow and click pull message and go to the second window click pull message and we can see the message is not showing now go to the first window and click the stop pulling and go back to second window and we can see the message is show here now.

If we want to change the visibility time out edit the SQS visibility timeout and by default it set 30 seconds.

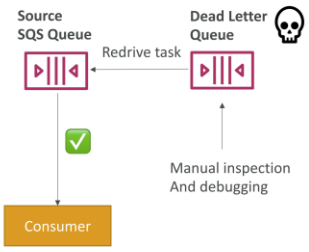
SQS – Dead Letter Queue (DLQ)

- If a consumer fails to process a message within the Visibility Timeout…
  the message goes back to the queue!
- We can set a threshold of how many times a message can go back to the queue
- After the **MaximumReceives** threshold is exceeded, the message goes into a Dead Letter Queue (DLQ)
- Useful for debugging!
- **DLQ of a FIFO queue must also be a FIFO queue**
- **DLQ of a Standard queue must also be a Standard queue**
- Make sure to process the messages in the DLQ before they expire:
  - Good to set a retention of 14 days in the DLQ



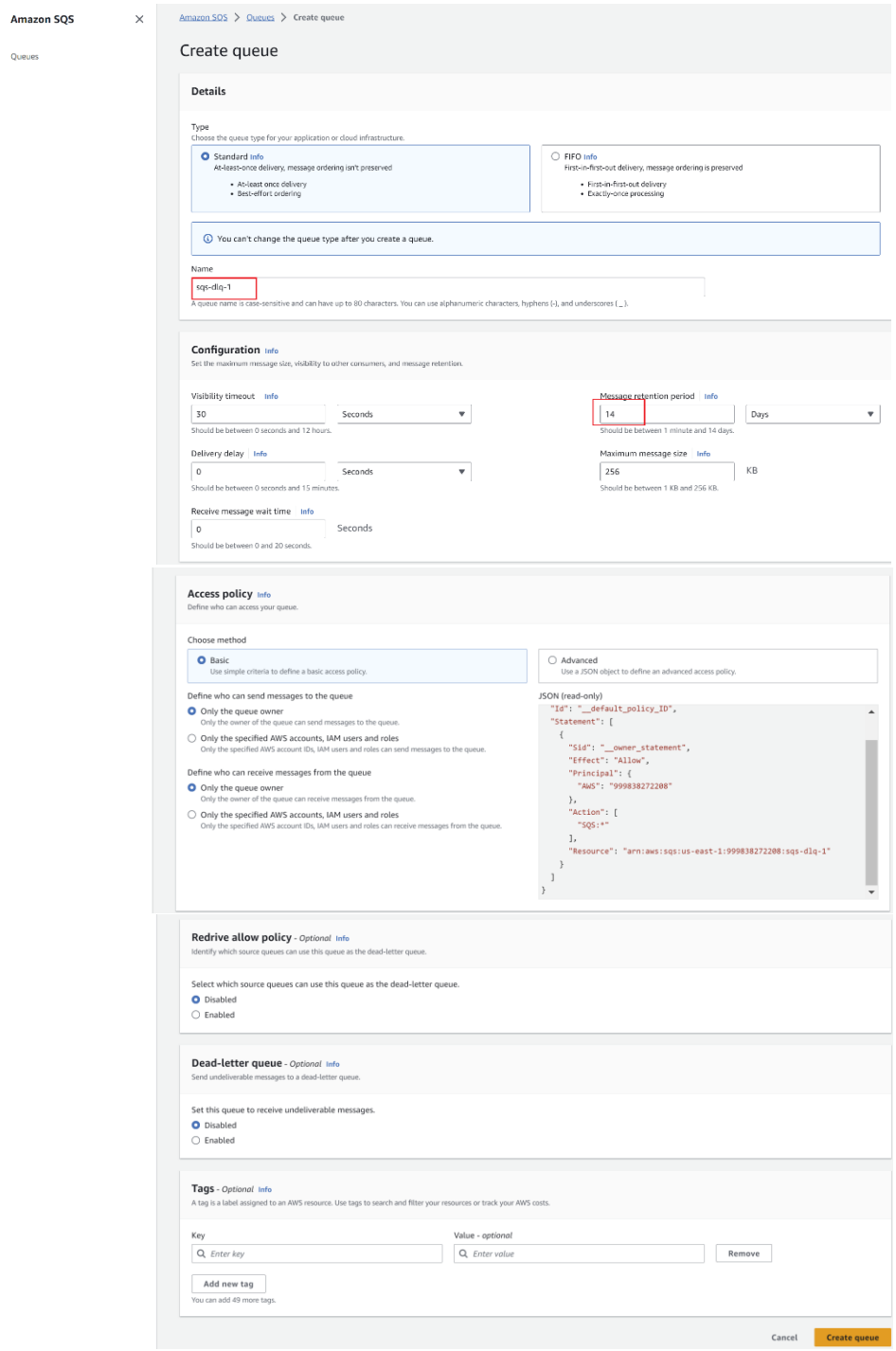
SQS DLQ – Redrive to Source

- Feature to help consume messages in the DLQ to understand what is wrong with them
- When our code is fixed, we can redrive the messages from the DLQ back into the source queue (or any other queue) in batches without writing custom code



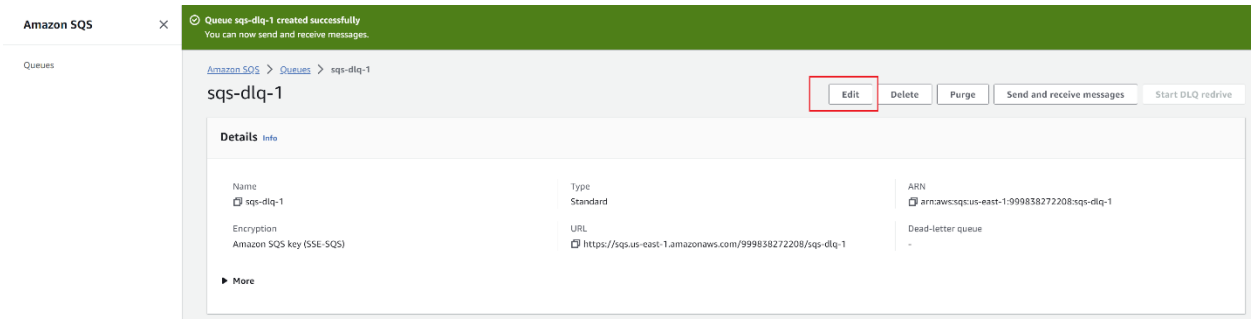
LAB: Dead Letter Queue (DLQ)

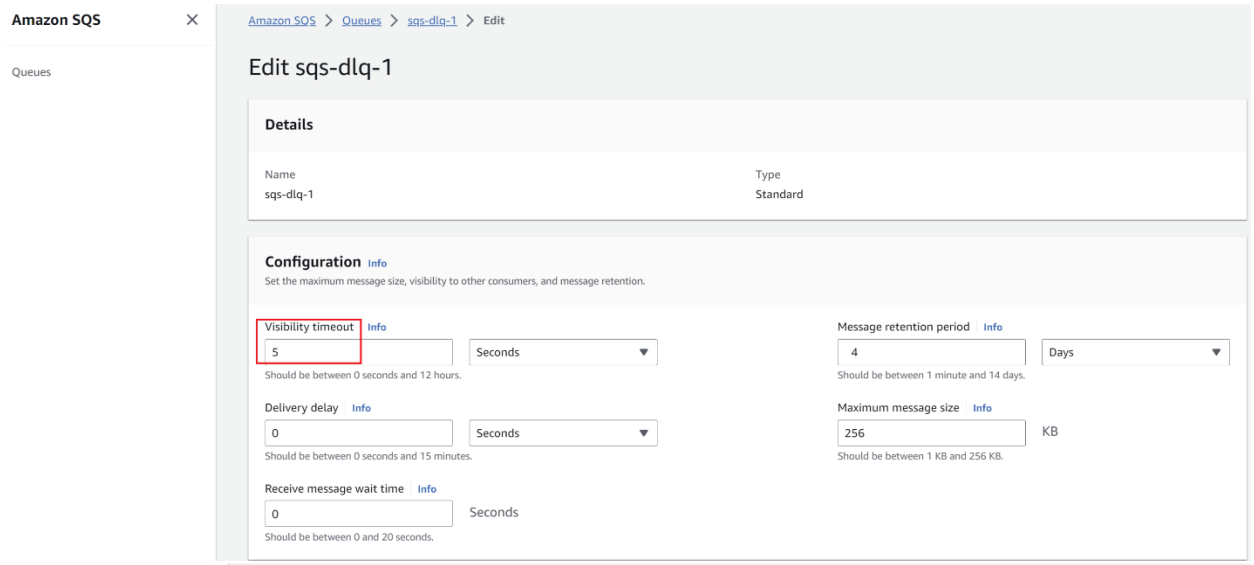
First create SQS (sqs-dlq-save) with 14 days retention period as shown below image and don't change anything just create the SQS.

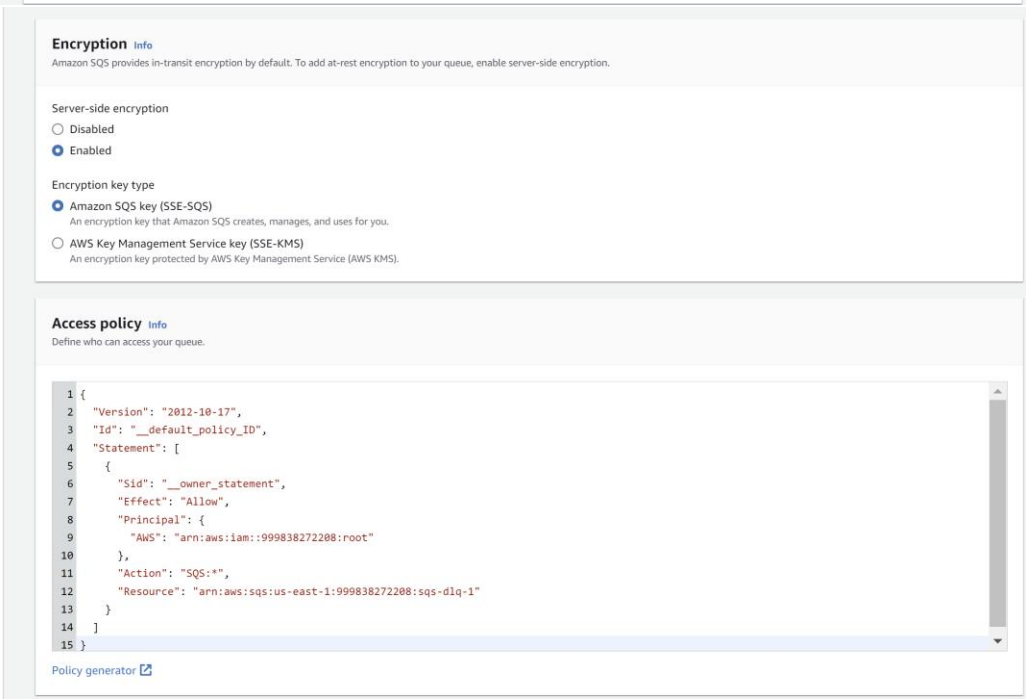


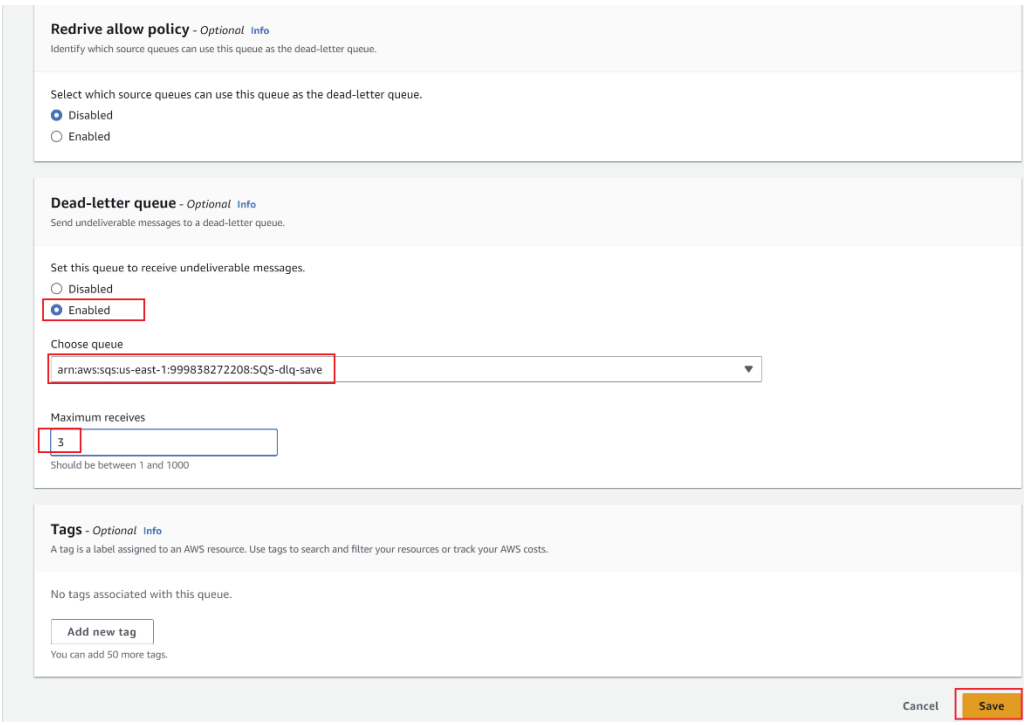
Now create another SQS (sqs-main) and edit it set visibility timeout 5 and scroll down and enable dead letter que and chose where we save the dead letter queue and we will the SQS just we create before

which is SQS (sqs-dlq-save) and maximum receive 3 as shown below image

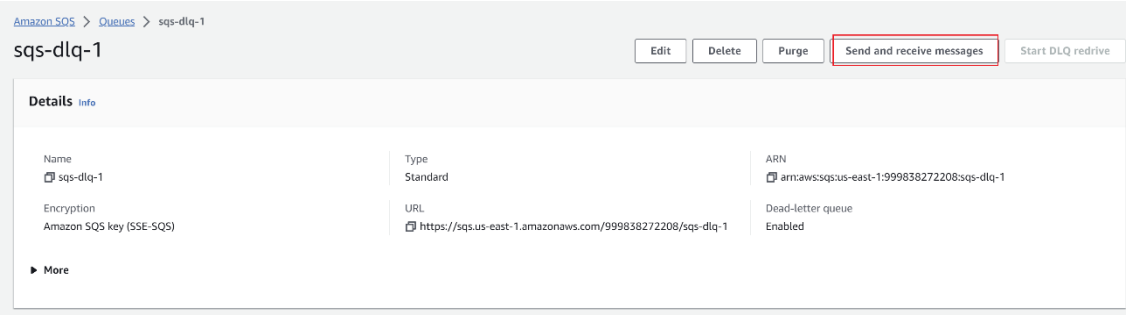


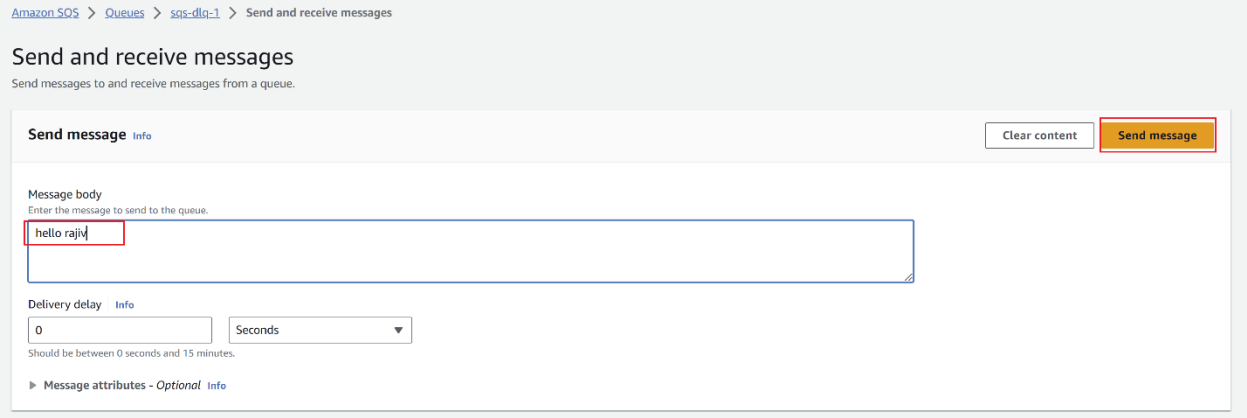




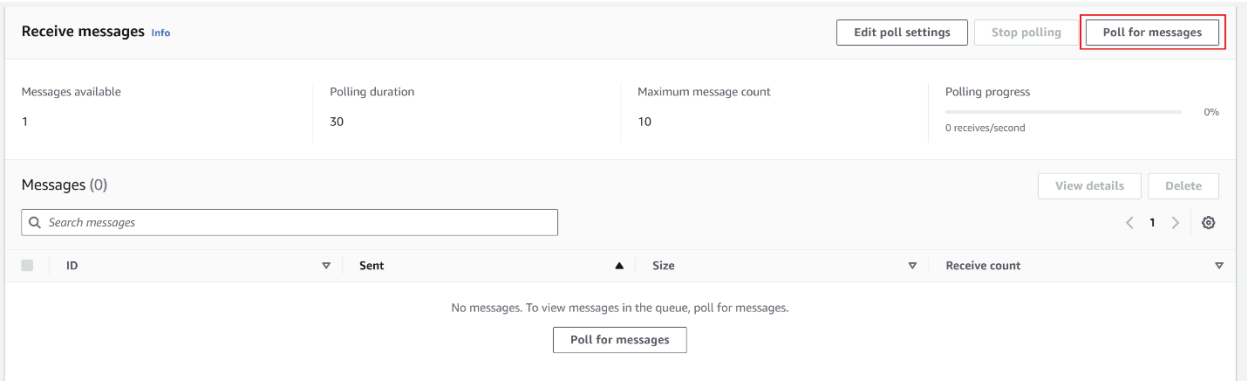


Now go to the SQS (sqs-dlq-save) and click send and pull message and then scroll down and click pull message and then go back to main SQS (sqs-main) and send a message and pull the message and wait for 3 messages is come and now stop pull message and click the pull message again we see now no message is coming.

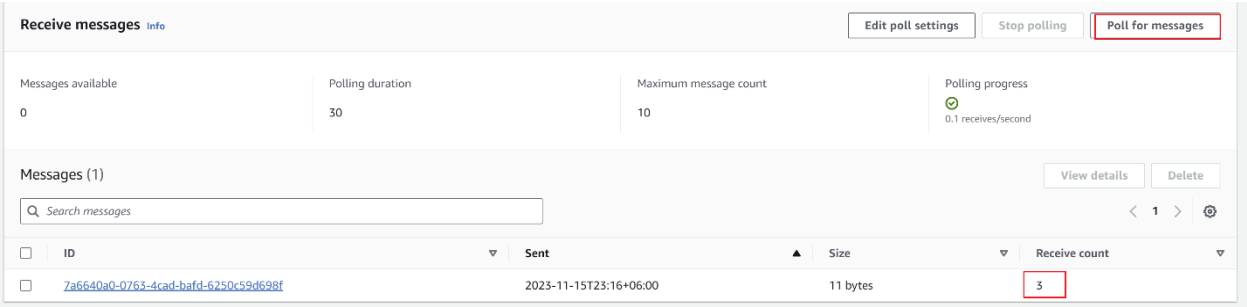




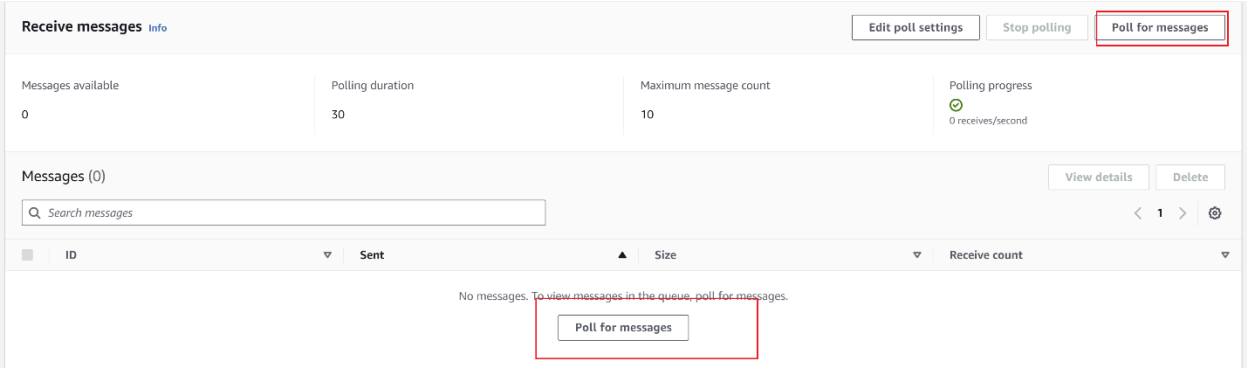
Scroll down



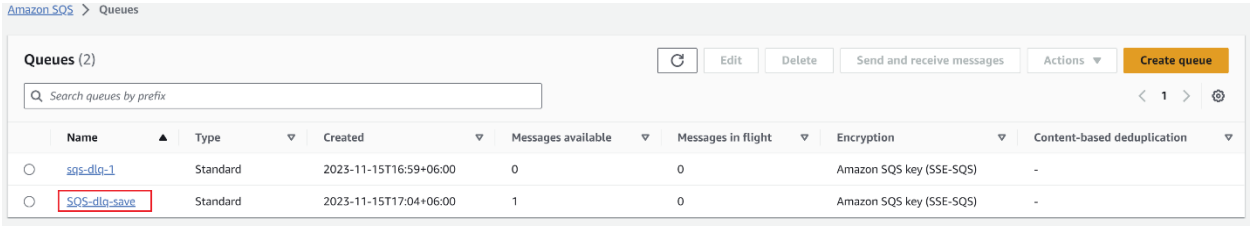
Now click stop polling and we see the receive count

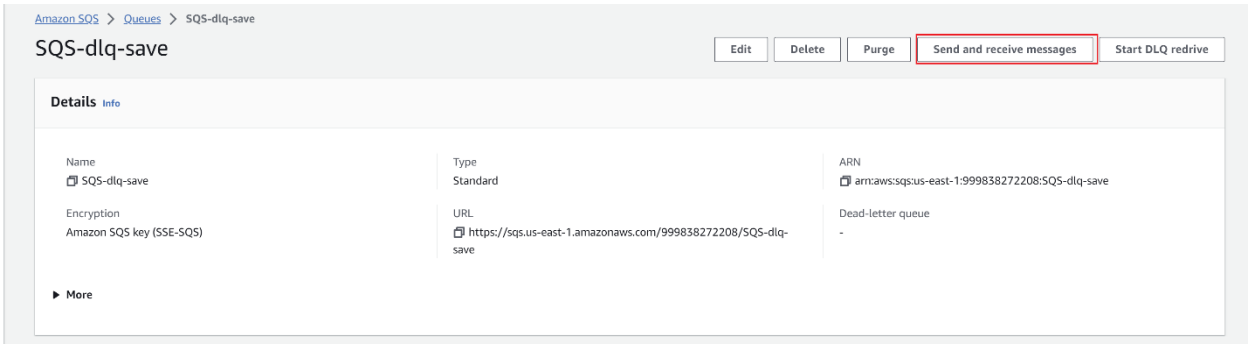


Now pull for message again we can the message is gone

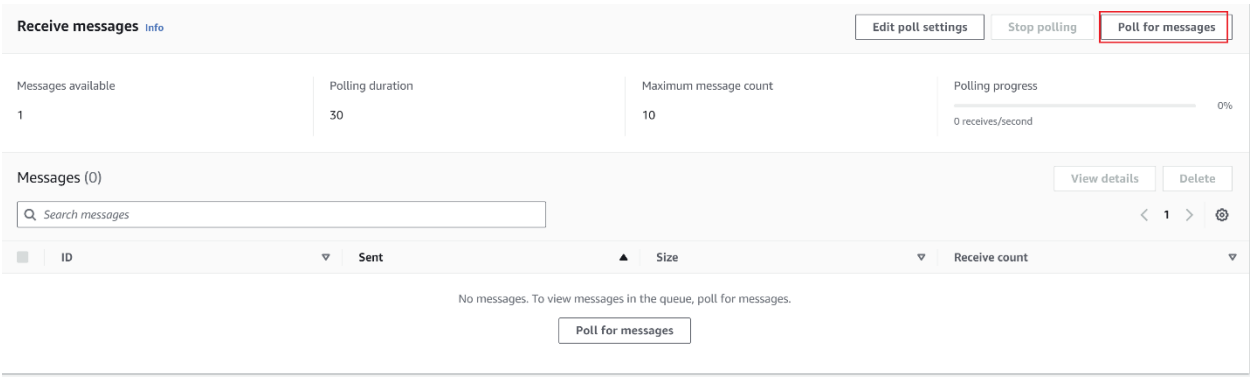


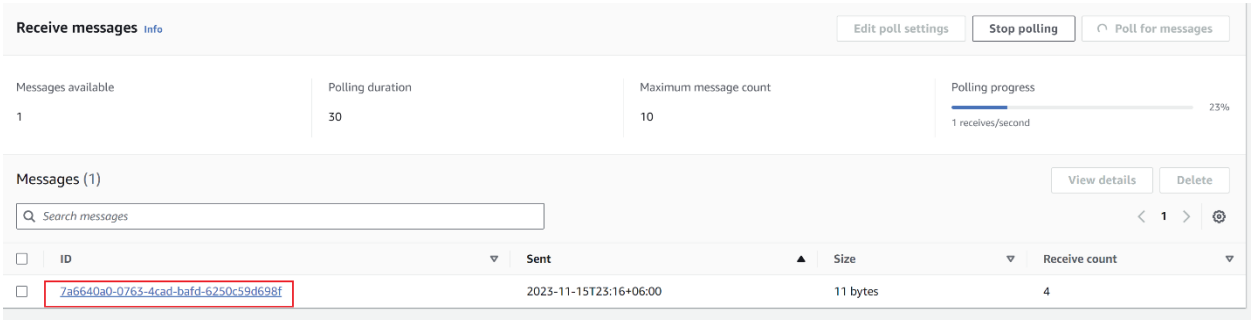
Now go to DLQ SQS where our message is saved pull for message and click the message and we see the message



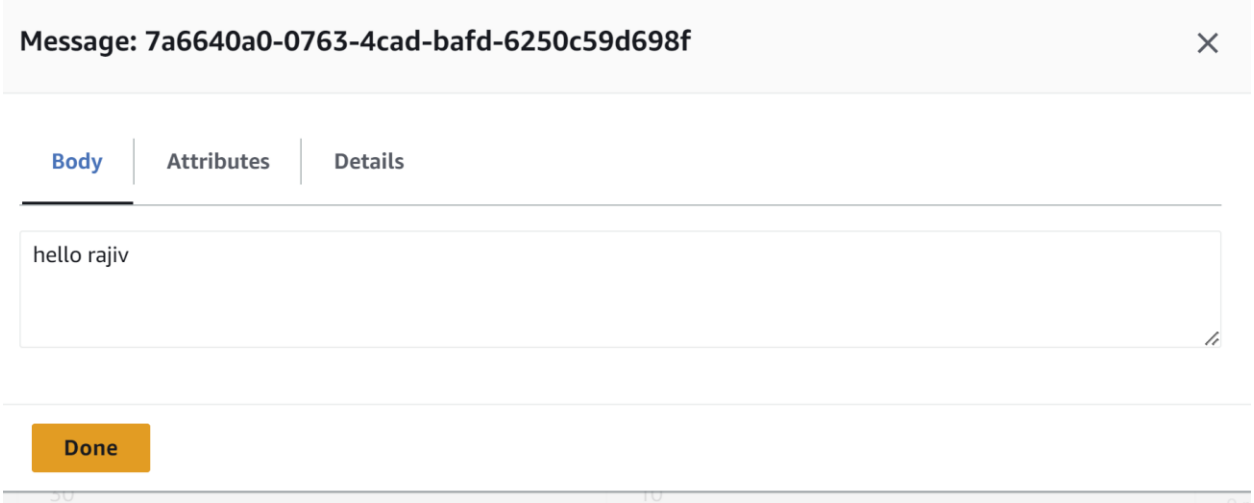


Scroll down



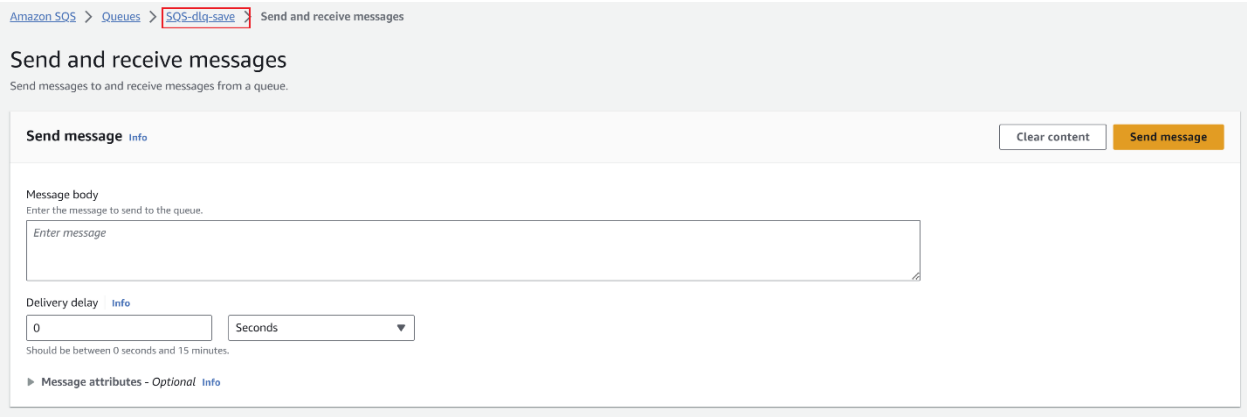


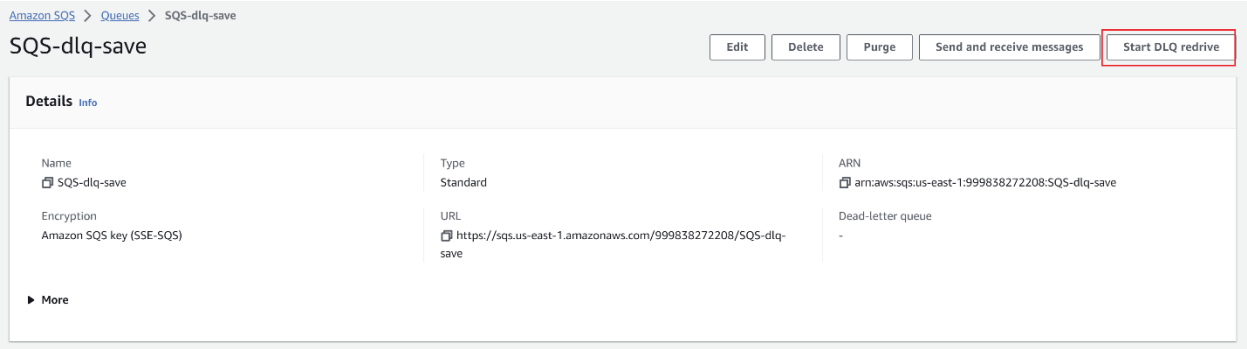
Now click the message and we can see that message

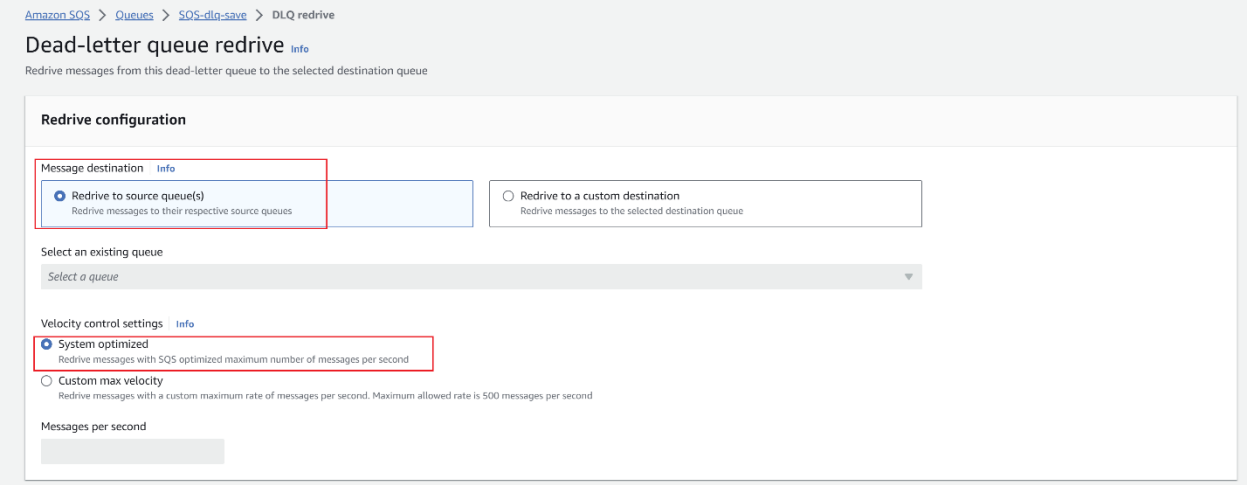


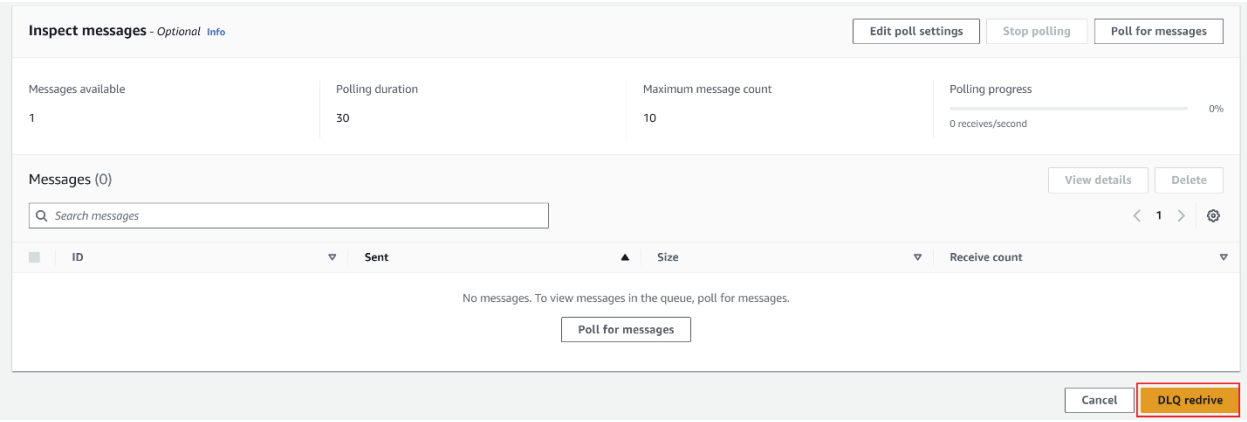
Now retrieve my DQL message

Go to DLQ SQS

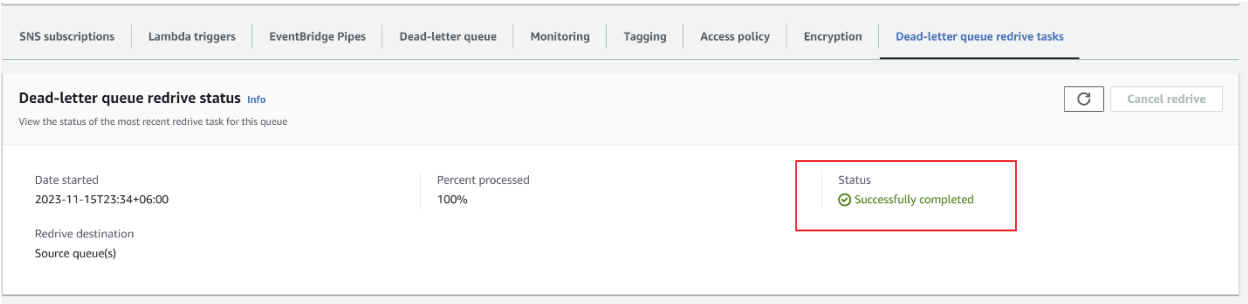




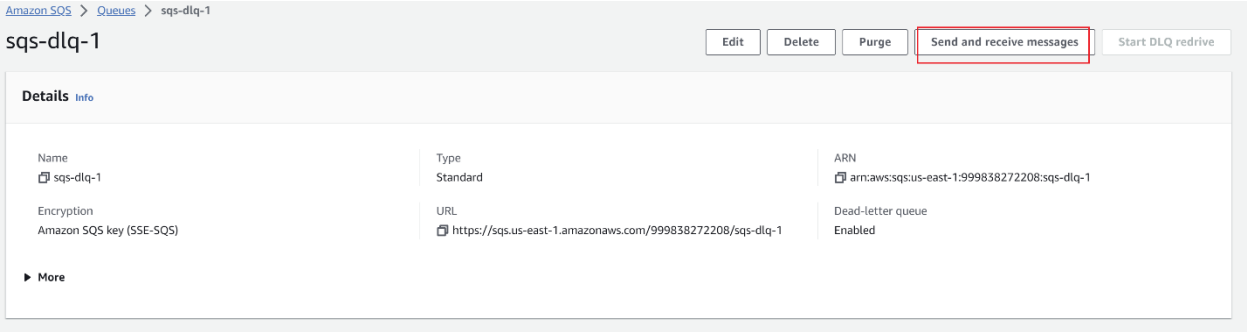




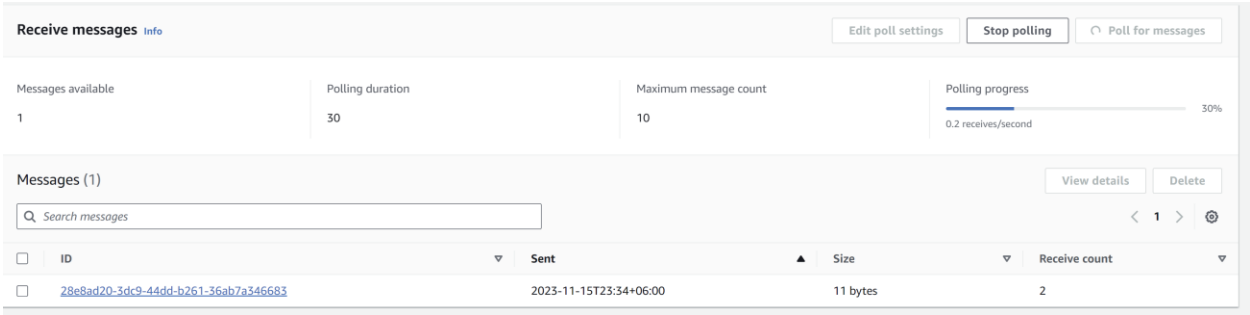
After clicking the DLQ go back to min SQS and pull the message and we see the message is showing now.



Now go back to main SQS



Click pull messages and we can see the message is retrieve

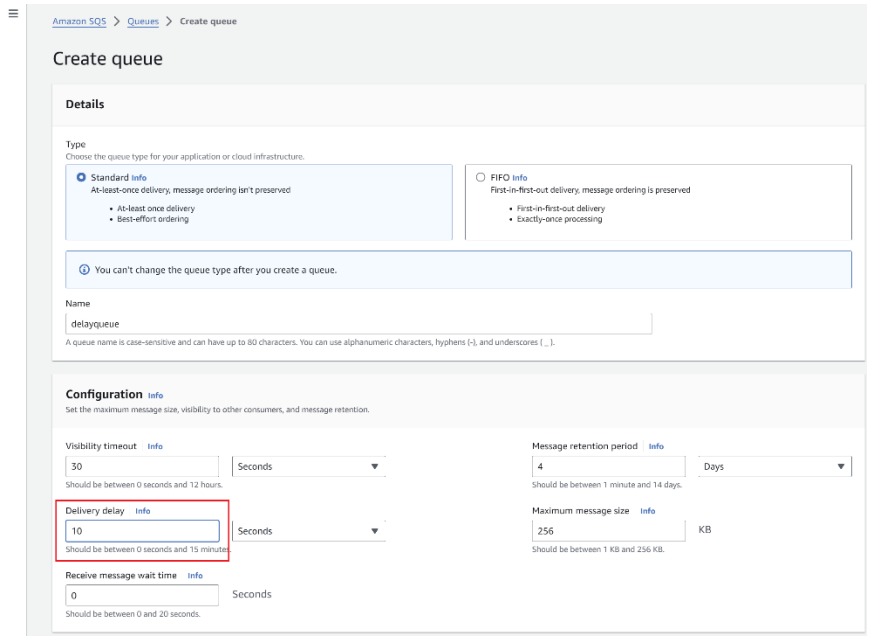


SQS – Delay Queue

- Delay a message (consumers don't see it immediately) up to 15 minutes
- Default is 0 seconds (message is available right away)
- Can set a default at queue level
- Can override the default on send using the **DelaySeconds** parameter

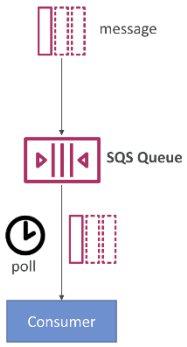
LAB: Delay Queue

Create SQS with 10 sec delay and after creating the delay and send a message and pull the message then we can see we need to wait 10 sec to get the message.



SQS - Long polling

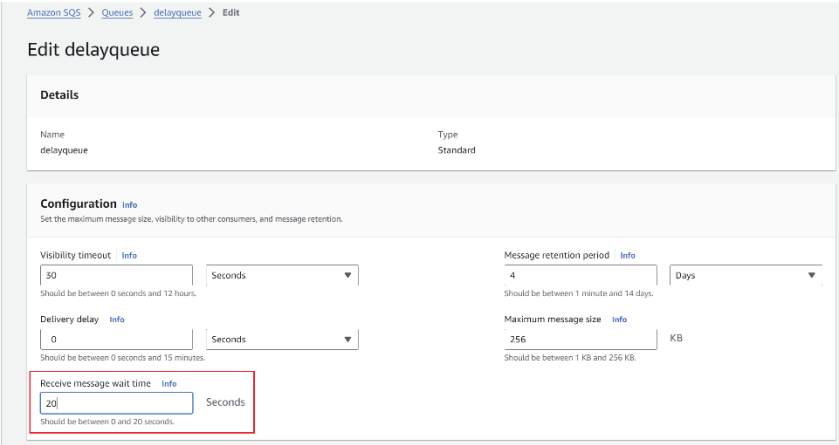
- When a consumer requests messages from the queue, it can optionally "wait" for messages to arrive if there are none in the queue
- This is called Long Polling
- LongPolling decreases the number of API calls made to SQS while increasing the efficiency and reducing latency of your application
- The wait time can be between 1 sec to 20 sec (20 sec preferable)
- Long Polling is preferable to Short Polling
- Long polling can be enabled at the queue level or at the API level using WaitTimeSeconds



Lab: Long polling

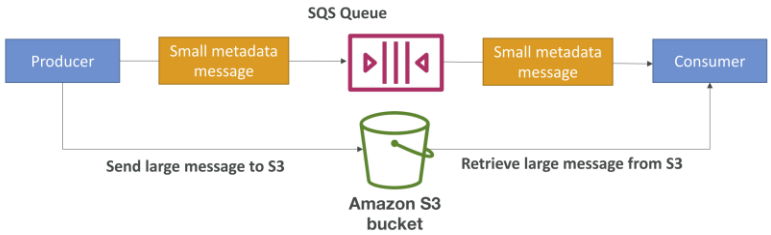
when create a SQS we set receive message wait time 20 sec and click pull the message queue and now within this time if the producer sends any message then user don't need to wait the message will come right way.

So, after set Received message wait time scroll down and click the pull message button now scroll up and send a message and we can see the message comes right way.



SQS - Extended client

- Message size limit is 256KB, how to send large messages, e.g. 1GB?
- Using the SQS Extended Client (Java Library)



SQS- Important things need to be:

- CreateQueue (MessageRetentionPeriod), DeleteQueue
- PurgeQueue: delete all the messages in queue
- SendMessage (DelaySeconds), ReceiveMessage, DeleteMessage
- MaxNumberOfMessages: default 1, max 10 (for ReceiveMessage API)
- ReceiveMessageWaitTimeSeconds: Long Polling
- ChangeMessageVisibility: change the message timeout

- Batch APIs for SendMessage, DeleteMessage, ChangeMessageVisibility helps decrease your costs

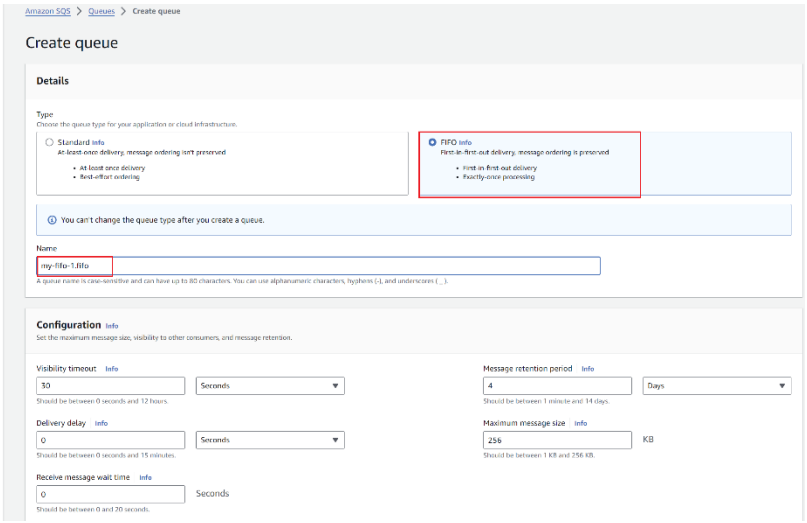
SQS – FIFO Queue

- FIFO = First In First Out (ordering of messages in the queue)

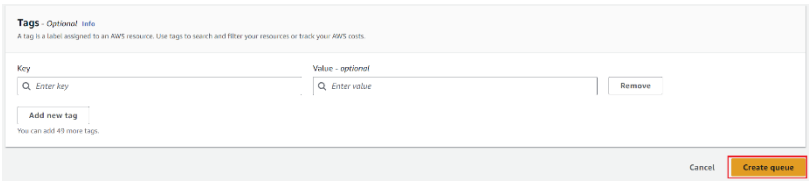
- Limited throughput: 300 msg/s without batching, 3000 msg/s with
- Exactly-once send capability (by removing duplicates)
- Messages are processed in order by the consumer

LAB: SQS - FIFO Queue

Create a FIFO Queue and keep default settings now send 4 messages rajiv1, rajiv2, rajiv3, rajiv4 now pull the messages and check the messages and start form the top one and we can see which message we send first that one come first. Here we send rajiv1 first so we can see rajiv1 message come first.



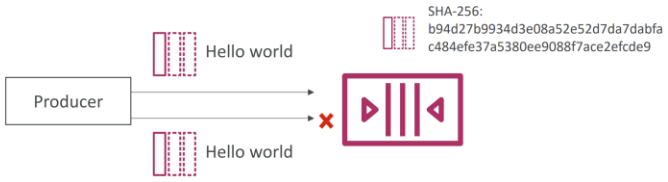
.
.
.



After creating the queue send 4 messages raijv1, rajiv2, rajiv3, rajiv4 and pull the message and check

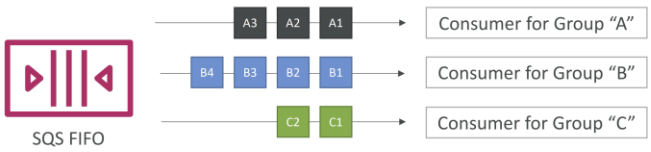
SQS- FIFO – Deduplication

- De-duplication interval is 5 minutes
- Two de-duplication methods:
  - Content-based deduplication: will do a SHA-256 hash of the message body
  - Explicitly provide a Message Deduplication ID



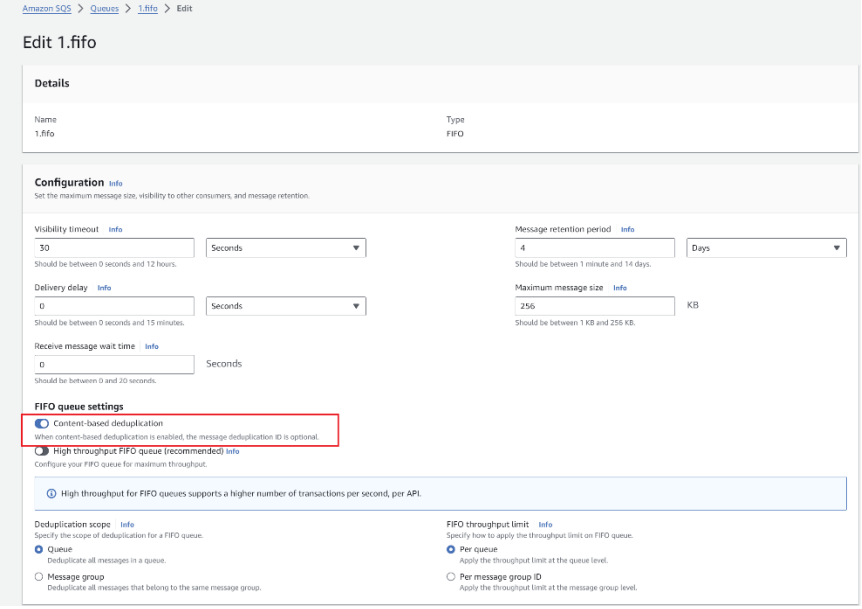
SQS- FIFO – Message Grouping

- If you specify the same value of **MessageGroupID** in an SQS FIFO queue, you can only have one consumer, and all the messages are in order
- To get ordering at the level of a subset of messages, specify different values for **MessageGroupID**
  - Messages that share a common Message Group ID will be in order within the group
  - Each Group ID can have a different consumer (parallel processing!)
  - Ordering across groups is not guaranteed



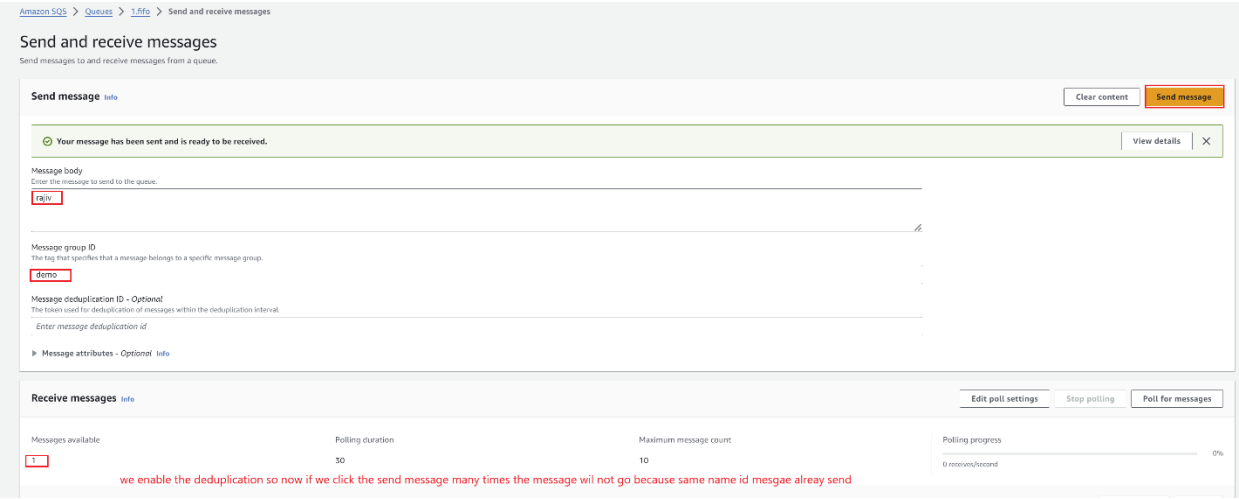
Lab: FIFO – Deduplication

Edit the fifo SQS and enable the content-based deduplication as shown blow

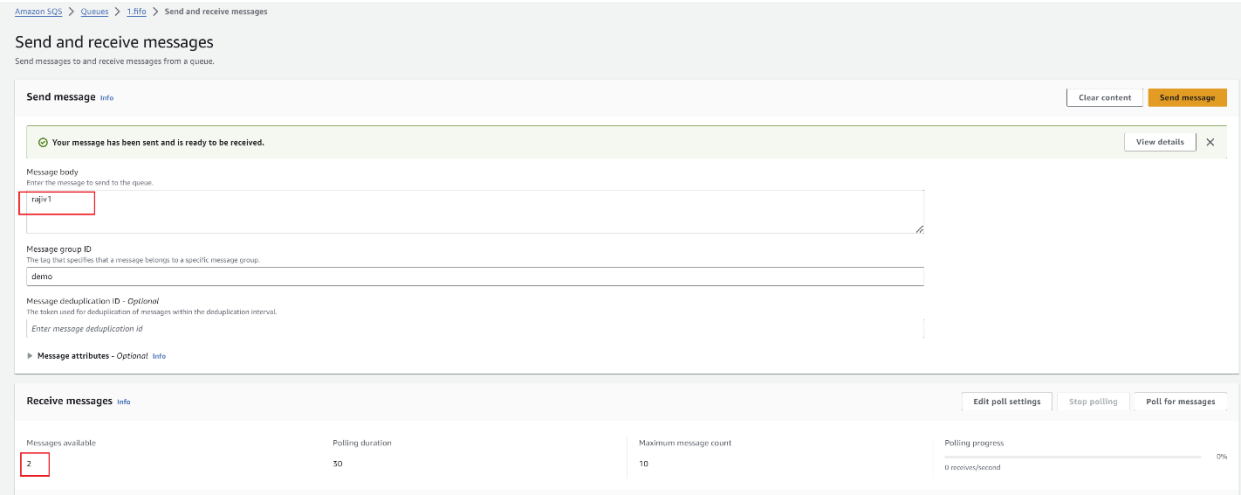


After clicking the send and receive message

Now create and send a message we can see the message is going only one times even though we click send message button several times.



Now if we change the message body and send the message we can see message is going



Now if we add our own duplication id and send message then the message is going because now the id is different but now it I try to send the message again it will not go because the id 123 already have the same message