

Amazon SQS & Amazon MQ

Overview

- Modern applications are made of **multiple services** (frontend, backend, workers, databases).
These services should not depend on each other directly, otherwise if one fails, everything fails.
- AWS provides messaging services to help applications talk to each other safely and reliably.
- The two services covered here are:
 - o Amazon Simple Queue Service (SQS) – Serverless message queues
 - o Amazon MQ – Managed traditional message brokers (RabbitMQ / ActiveMQ)
- Why Messaging Is Needed:
 - o Without Messaging:
 - App A calls App B directly
 - If App B is slow or down → App A also fails
 - Traffic spikes can crash backend services
 - o With Messaging:
 - App A sends a message
 - Backend processes it later
 - Apps are decoupled, scalable, and fault-tolerant

1. Amazon SQS

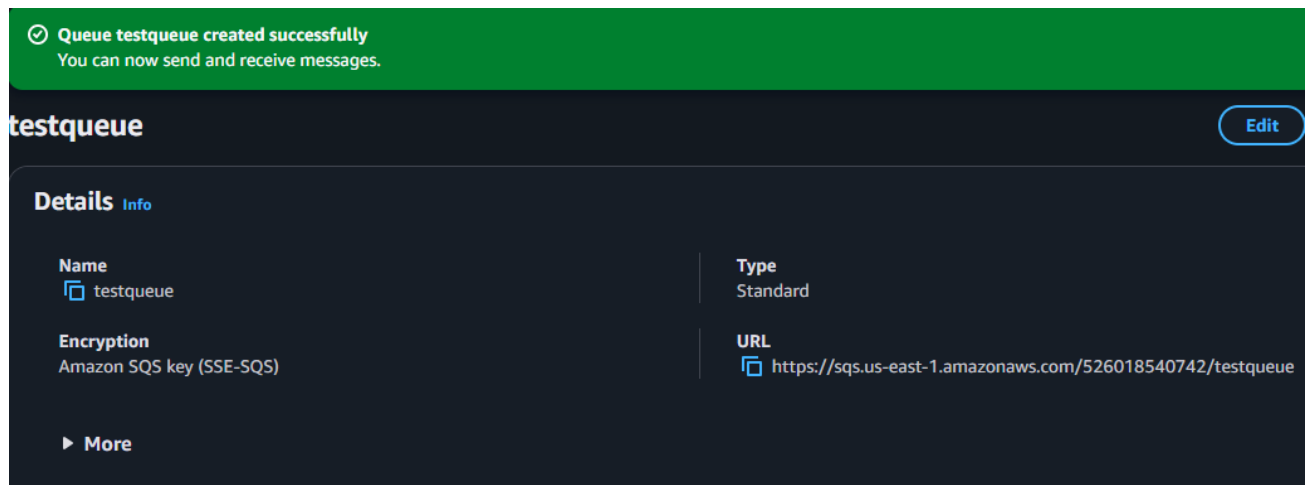
- SQS was the first service available in AWS.
- Amazon SQS is a web service that gives you access to a message queue that can be used to store messages while waiting for a computer to process them. It is a fully managed, serverless message queue service.
- With the help of SQS, you can send, store and receive messages between software components at any volume without losing messages. Messages are stored until processed.
- Producers put messages in the line, and the consumers pick messages when ready.
- Messages can contain up to 256 KB of text in any format such as Json, xml, etc. The smallest supported message size is 1 byte (1 character). The largest size is 1,048,576 bytes (1024 KiB).
- The default retention period is 4 days. The retention period has a range of 60 seconds to 1,209,600 seconds (14 days).

Amazon SQS & Amazon MQ

- **Queue:** Storage where messages wait.
- **Producer:** App that sends messages.
- **Consumer:** App that receives messages.
- **Visibility Timeout:** Time during which message is hidden while being processed.
- **Dead Letter Queue (DLQ):** Queue for failed messages.
- There are two types of SQS Queues:
 - Standard Queue
 - Unlimited throughput
 - Messages may arrive more than once
 - Best for most applications
 - FIFO Queue
 - Messages processed exactly once
 - Order is preserved
 - Slightly lower throughput
- **Use-Cases:**
 - Background job processing: SQS allows time-consuming tasks (like image processing or report generation) to run in the background without blocking the main application.
 - Order Processing Systems: SQS queues incoming orders, so they are processed reliably one at a time, even during traffic spikes.
 - Event-Driven Microservices: SQS enables microservices to communicate asynchronously, reducing tight coupling between services.
 - Buffering Traffic Spikes: SQS absorbs sudden bursts of requests and processes them gradually, preventing backend overload.
 - Decoupling Frontend and Backend: SQS separates the frontend from backend services, so failures in one do not impact the other.
- **Architecture Flow:**
 - Producer application sends message to SQS
 - SQS stores the message safely

Amazon SQS & Amazon MQ

- Consumer polls SQS
 - Polls means the application repeatedly checks the SQS queue to retrieve new messages.
 - It cannot push messages directly to your app.
 - Our app must pull (poll) messages when it's ready.
- Consumer processes message
- Message is deleted after success
- Failed messages go to DLQ
 - A DLQ (Dead Letter Queue) is a separate backup queue where failed messages are sent, so they don't get stuck retrying forever.
- **Execution Steps:**
- Step 1: Create an SQS Queue.
 - Open AWS Console
 - Go to Amazon SQS
 - Click Create queue
 - Choose: Standard (recommended for most cases)
 - Name the queue
 - Configure: Visibility Timeout (e.g., 30 seconds)
 - Click Create queue



- Step 2: Create Dead Letter Queue (Optional but Best Practice)
 - Create another queue
 - Name it your-queue-name-DLQ
 - Attach it as DLQ in main queue settings
 - Set max receive count (e.g., 3)

Amazon SQS & Amazon MQ

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

Set this queue to receive undeliverable messages.

☐ Disabled
☒ Enabled

Choose queue

arn:aws:sqs:us-east-1:526018540742:Dead-letter ▼

Maximum receives

10

Should be between 1 and 1000

- Step 3: Send a Message
 - Producer needs these permissions:
 - sqs:SendMessage
 - Open the queue
 - Click Send and receive messages
 - Enter message body (JSON/text)
 - Click Send message

```
[ec2-user@ip-172-31-72-11 ~]$ aws sqs send-message \  
> --queue-url https://sqs.us-east-1.amazonaws.com/526018540742/testqueue \  
> --message-body "Hello from Producer EC2"  
{  
  "MD5OfMessageBody": "6f0927a1edfb5b5f18b639791a541c20",  
  "MessageId": "da5d2ace-b689-4bd4-a2b9-dbe52a305770"  
}
```

Send message [Info](#)

✓ Your message has been sent and is ready to be received.

Message body
Enter the message to send to the queue.

Hey Buddy!

Message group ID - optional, new [Info](#)

- Step 4: Receive and Process Message
 - Consumer application polls the queue
 - Consumer needs these permissions:
 - sqs:ReceiveMessage
 - sqs:DeleteMessage
 - sqs:GetQueueAttributes

Amazon SQS & Amazon MQ

- Message becomes invisible (visibility timeout)
- Consumer processes message
- On success → delete message
- On failure → message returns or moves to DLQ

```
[ec2-user@ip-172-31-16-26 ~]$ aws sqs receive-message --queue-url https://sqs.us-east-1.amazonaws.com/526018540742/testqueue
{
  "Messages": [
    {
      "MessageId": "da5d2ace-b689-4bd4-a2b9-dbe52a305770",
      "ReceiptHandle": "AQEBREGFxqX47oey3Z1vyfTDJtnVwvie0tRGqGvYaQ+tQ1N/L7wRZVim9HLt36+VbYGugGiRqvKfW9e51HoyV6foLD1FQjhvWdguLD1DDMyF9YVYPi1mozkmFSQ0StE3yernsh8Um9tXZKtZ6zykJq3LRL51pH7J80XCuYL+i5wwW1VNS+B6jsgZaBzEW2SZ1PXDjRd1Pmdic09zPXKwhFWJ0GtBRCYx6raAaF06VVJuj2aC5AZgex1c2D50kTBawMceSwSxx0b21gn+befaIjFZgznen5NhT1U6rT/EBtPq6C3weMe6q26XFzL1VFUIh0FU0ZDLtu4Znvh7a6XCjAxQFi7zK9MBPfyhryhjwdi0sxe3ayxWqC/FhXjTHrLKRb7",
      "MD5OfBody": "6f0927a1edfb5b5f18b639791a541c20",
      "Body": "Hello from Producer EC2"
    }
  ]
}
```

Received message: bdbcf26-ed1b-468b-81b8-fd2a1e1197a8

Body

Attributes

Details

Hey Buddy!

- Step 5: Monitoring
 - Use CloudWatch metrics:
 - Number of messages
 - Queue depth
 - DLQ count

- **Best Practices:**

Amazon SQS & Amazon MQ

- Always use DLQ
 - Set proper visibility timeout
 - Use IAM roles, not hardcoded keys
-
- Amazon SQS is best for modern, serverless, scalable applications.

Amazon SQS & Amazon MQ

2. Amazon MQ

- Amazon MQ is a managed message broker service for Apache ActiveMQ and RabbitMQ that makes it easy to set up, migrate and operate message brokers on AWS Cloud.
- Amazon MQ takes care of all the behind-the-scenes work like setting up the servers, configuring the messaging system, updating software, and automatically fixing issues if something goes wrong.
- Unlike SQS:
 - Amazon MQ uses traditional messaging protocols.
 - Requires broker management (connections, ports).
- Broker: Messaging server.
- Exchange (RabbitMQ): Routes messages.
- Topic / Queue: Message destination.
- Producer: Sends messages.
- Consumer: Receives messages.
- Protocols: AMQP, JMS, MQTT, STOMP.
- It provides encryption of messages at rest and in transit.
- It is integrated with Amazon CloudWatch and AWS CloudTrail.
- It supports both single-instance brokers, suitable for evaluation and testing, and active/standby brokers for high availability in production.
- It stores messages redundantly across multiple Availability Zones (AZ) within an AWS region and will continue to be available if a component or AZ fails.
- Amazon MQ supports common, industry-standard messaging methods, which makes it easy to move existing messaging applications to AWS by just changing where they connect, instead of rewriting the application.
- **Use-cases:**
 - Enterprise Integrations: Amazon MQ connects multiple enterprise systems using reliable, standards-based messaging protocols.
 - Financial Transaction Systems: Amazon MQ ensures guaranteed message delivery and ordering for critical financial operations.

Amazon SQS & Amazon MQ

- ERP and CRM Messaging: Amazon MQ allows ERP and CRM systems to exchange data using traditional messaging patterns.
 - ERP (Enterprise Resource Planning) manages internal business operations, while CRM (Customer Relationship Management) manages customer interactions and relationships.
- Legacy Java Applications: Amazon MQ supports ActiveMQ and RabbitMQ, making it ideal for migrating existing Java-based systems to AWS.
- On-Premises to Cloud Migration: Amazon MQ enables lift-and-shift migration of existing messaging systems without changing application code.

- **Architecture Flow:**

- Producer connects to broker using protocol.
- Message sent to queue or topic.
- Broker routes message.
- Consumer receives message.
- Message acknowledged.
- Broker removes message.

- **Execution Steps:**

- Step 1: Create a Broker

- Open AWS Console
- Go to Amazon MQ
- Click Create broker
- Choose broker type:
 - RabbitMQ (modern, popular)
 - ActiveMQ (legacy systems)
- Choose deployment mode:
 - Single instance (dev)
 - Active/Standby (prod)

Amazon SQS & Amazon MQ

TestBroker [Info](#)

Details

Specifications	Configuration	Security and network	Maintenance
ARN Info <code>arn:aws:mq:us-east-1:526018540742:broker:TestBroker:b-31cee6a9-4a12-4714-938b-a08cc255df88</code>	Configuration name TestBroker-configuration	VPC Info <code>vpc-0535d400a67eee4e7</code> L	Automatic maintenance Yes
Broker name TestBroker	Configuration revision Revision 1 - Auto-generated default for TestBroker-configuration on RabbitMQ 4.2 L	Subnet(s) Info <code>subnet-0cd7db87245e9c9c7</code> L	Maintenance window Wednesday 21:00-23:00 UTC
Broker status Running	CloudWatch Logs General Disabled - Logs L	Public accessibility Info Yes	
Broker instance type Info mq.m7g.medium			
Deployment mode Info Single-instance broker			
Broker engine Info RabbitMQ			
Broker engine version 4.2			

- Step 2: Configure Broker
 - o Give broker name
 - o Choose instance size
 - o Set admin username and password
 - o Configure VPC and subnet
 - o Configure security group (open required ports)
- Step 3: Create Queues / Topics
 - o Access broker console URL
 - o Login using admin credentials
 - o Create:
 - Queue (point-to-point)
 - Topic (publish/subscribe)

Page **1** of 1 - Filter: ☐ Regex [?](#)

Overview					Messages			Message rates			+/-
Virtual host	Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack	
/	Testing	quorum	Args	running	0	0	0				

[Add a new queue](#)

[HTTP API](#) [Documentation](#) [Tutorials](#) [New releases](#) [Commercial edition](#) [Commercial support](#) [Discussions](#) [Discord](#) [Plugins](#)

- Step 4: Connect Producer Application
 - o Use supported protocol (AMQP/JMS)
 - o Provide:

Amazon SQS & Amazon MQ

- Broker endpoint
- Port
- Username/password
- Send messages

```
[ec2-user@ip-192-168-0-19 ~]$ python3 producer.py
Message sent
[ec2-user@ip-192-168-0-19 ~]$ |
```

- Step 5: Connect Consumer Application

- Connect to same broker
- Subscribe to queue or topic
- Process messages
- Acknowledge message

```
ec2-user@ip-192-168-0-102:~
[ec2-user@ip-192-168-0-102 ~]$ python3 consumer.py
Waiting for messages...
Received: Hello from Producer EC2
```

- Step 6: Monitoring

- CloudWatch:
 - CPU, memory
 - Queue depth
- Broker console:
 - Connections
 - Message rates

- Best Practices:

- Use Active/Standby for production
- Restrict broker access with security groups
- Monitor broker health regularly
- Amazon MQ is best for legacy and protocol-based systems.

Amazon SQS & Amazon MQ

3. Amazon SQS vs Amazon MQ

Feature	Amazon SQS	Amazon MQ
Server management	Fully serverless	Managed brokers
Protocols	AWS API only	AMQP, JMS, MQTT
Scalability	Automatic	Limited by broker
Ordering	FIFO option	Supported
Best for	Cloud-native apps	Legacy systems