AWS SQS
Simple Queue Service

Before we learn about the SQS, we need to understand few basics and terms – which are very much necessary for building the AWS SQS Solutions in our day-to-day work.

✓ Application decoupling

Application decoupling refers to designing systems in such a way that the components or services within the application are independent of each other or different applications independent of each other even though, they are chained systems.

This means that changes in one component do not directly impact the other components, allowing for greater flexibility, scalability, and maintainability. Decoupling is an important principle in software architecture, particularly in microservices and distributed systems.

Now we are saying Application de coupling as one of the best practices ..! then how can we achieve or maintain this standard in our applications – That's where Queue comes into the picture.

✓ What are message queues? why we need them?

Message queues are used to facilitate communication between different components of a system in a decoupled manner.

The main aspects, we need keep in our mind

Decoupling: By using a queue, the sender and receiver do not need to know about each other's existence. They only need to know about the queue, which acts as an intermediary.

Reliability: Queues provide durability and can store messages until they are successfully processed, ensuring no data is lost if a component is temporarily unavailable.

Scalability: Queues make it easier to scale the components independently. You can add more instances of the receiver service to handle an increased load without affecting the sender.

Retry Mechanisms: If a component fails to process a message, the queue can implement retry mechanisms to ensure the message is eventually processed.

Load Balancing: Queues can help distribute the workload evenly across multiple instances of a service, ensuring that no single instance is overwhelmed.

Asynchronous Communication: Queues allow components to communicate without waiting for each other. This means that the sender can continue its operations without waiting for the receiver to process the message.

These are all the aspects – we need to keep in our mind, being an application developer.

Now the question would be – can't we develop an application without a queue ☺, answer would be YES – we can develop the application systems without a queue BUT you will not be getting all the above-mentioned advantages/aspects.

Now we know why we need a queue 🤞…!! We will see the definitions now

AWS SQS
Simple Queue Service

Queue

A queue is a temporary repository for messages that are waiting to be processed. Messages are sent to a queue by producers and received from the queue by consumers. AWS SQS supports two types of queues:

- **Standard Queue:** Offers maximum throughput, best-effort ordering, and at-least-once delivery.
- **FIFO Queue:** Ensures that messages are processed exactly once, in the exact order that they are sent.

Message

A message is a unit of data that is sent to and stored in a queue. Messages can contain up to **256 KB** of text in any format.

Producer

A producer (or sender) is an application or service that sends messages to a queue. Producers can send messages at any time and at any rate.

Consumer

A consumer (or receiver) is an application or service that retrieves and processes messages from a queue. Consumers can be configured to poll the queue at regular intervals or to receive messages as they arrive.

Polling

Polling is the process of a consumer continuously checking a queue for new messages. SQS supports both long polling and short polling:

- **Short Polling:** The consumer makes frequent, brief checks for messages.
- **Long Polling:** The consumer waits for a specified period before checking for messages, reducing the number of empty responses and thus, costs.

Visibility Timeout

The visibility timeout is the period of time that a message received from a queue is invisible to other consumers. During this time, the consumer can process and delete the message without other consumers receiving it. If the message is not deleted within the visibility timeout, it becomes visible again and can be received by other consumers.

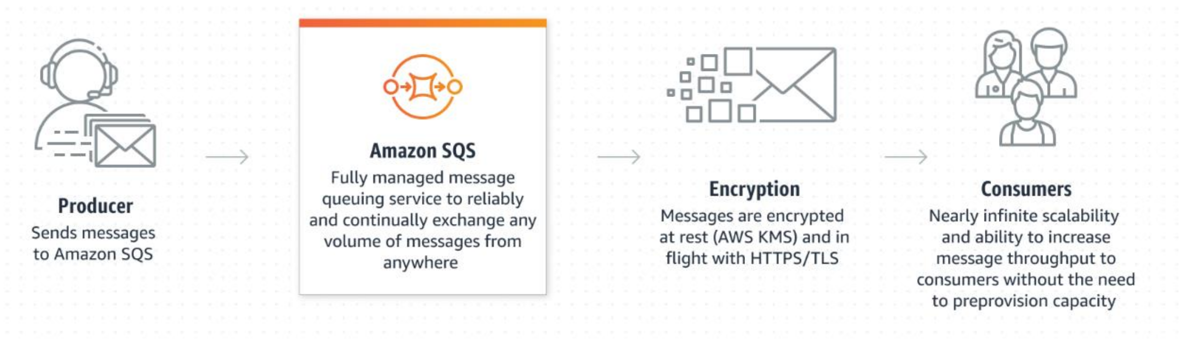
Dead-Letter Queue (DLQ)

A dead-letter queue is a queue that receives messages that could not be successfully processed by the main queue. DLQs help you isolate and analyse failed messages to understand why they were not processed.

Delay Queue

A delay queue allows you to postpone the delivery of new messages to consumers for a specified period. Messages sent to a delay queue remain invisible to consumers until the delay period has elapsed.

The process would look like this


Producer
Sends messages to Amazon SQS
Amazon SQS
Fully managed message queuing service to reliably and continually exchange any volume of messages from anywhere
Encryption
Messages are encrypted at rest (AWS KMS) and in flight with HTTPS/TLS
Consumers
Nearly infinite scalability and ability to increase message throughput to consumers without the need to preprovision capacity

Now – let's get into the AWS SQS Types

SQS Standard & SQS FFIO

SQS Standard

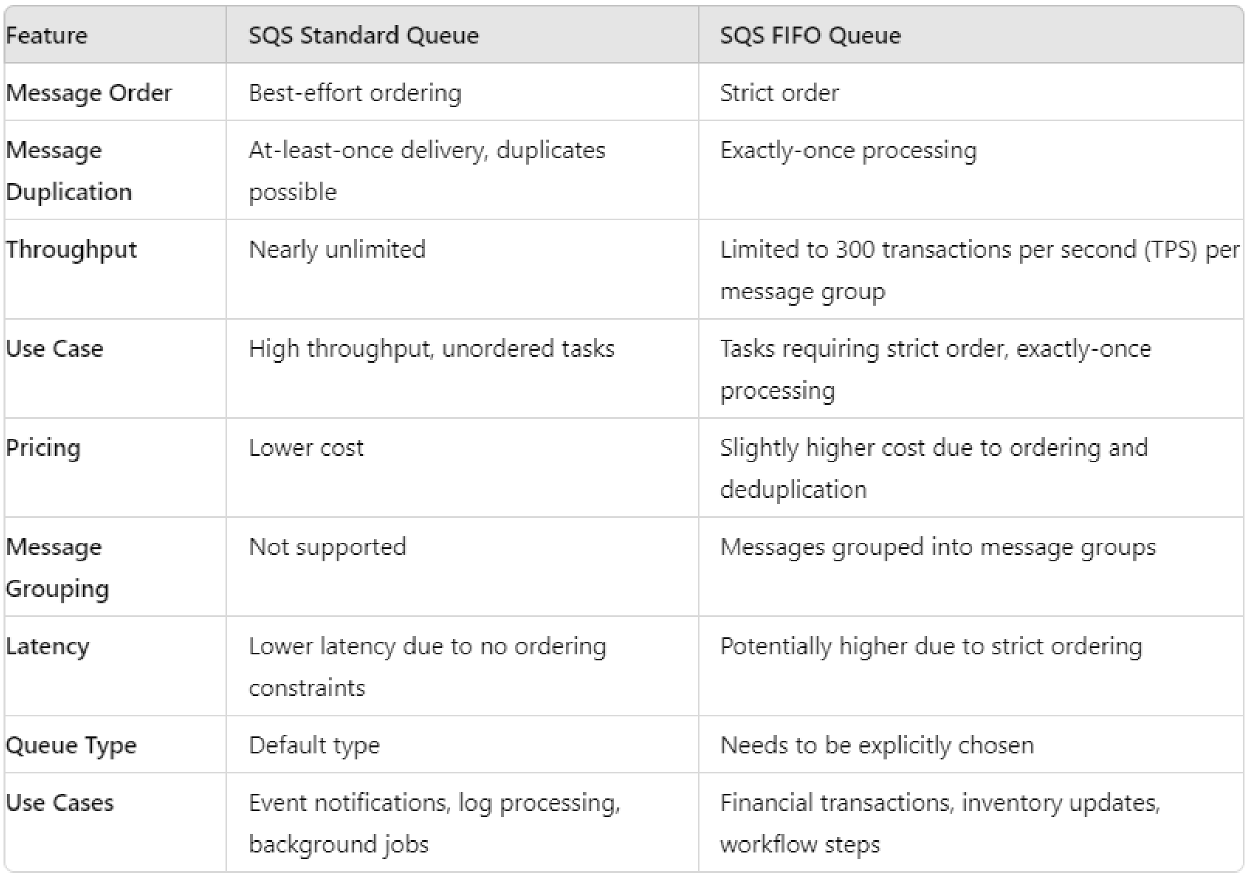
AWS Simple Queue Service (SQS) Standard Queue is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications. It provides a highly available, scalable, and cost-effective message queuing solution.

SQS FFIO

AWS Simple Queue Service (SQS) FIFO (First-In-First-Out) Queue is a fully managed message queuing service that ensures the order of messages is strictly preserved, and each message is processed exactly once.

In both types

- Max message size: **256 KB**
- Default message retention: **4 days (min: 1 min, max: 14 days)**



Feature	SQS Standard Queue	SQS FIFO Queue
Message Order	Best-effort ordering	Strict order
Message Duplication	At-least-once delivery, duplicates possible	Exactly-once processing
Throughput	Nearly unlimited	Limited to 300 transactions per second (TPS) per message group
Use Case	High throughput, unordered tasks	Tasks requiring strict order, exactly-once processing
Pricing	Lower cost	Slightly higher cost due to ordering and deduplication
Message Grouping	Not supported	Messages grouped into message groups
Latency	Lower latency due to no ordering constraints	Potentially higher due to strict ordering
Queue Type	Default type	Needs to be explicitly chosen
Use Cases	Event notifications, log processing, background jobs	Financial transactions, inventory updates, workflow steps

When to Choose What:

Choose SQS Standard Queue When:

- High throughput is required.
- Unlimited throughput (publish any number of messages per second into the queue)
- Message order is not critical.
- Low latency (<10 ms on publish and receive)
- Duplicate messages are acceptable or can be handled.
- You are looking for a cost-effective solution.
- Use cases include event notifications, batch processing, or any task where order doesn't matter.

Choose SQS FIFO Queue When:

- Message order must be strictly preserved.
- Exactly-once processing is critical.
- Throughput requirements are within the FIFO limits (300 TPS per message group). Limited throughput
  - 300 msg/s without batching (batch size = 1)
  - 3000 msg/s with batching (batch size = 10)
- Use cases include financial transactions, inventory updates, and any scenario where strict message ordering and deduplication are required.

In summary, SQS Standard Queue is suitable for scenarios where high throughput and lower costs are prioritized over strict ordering and deduplication. SQS FIFO Queue is ideal for applications were maintaining the order of messages and ensuring exactly-once processing are essential.

AWS SQS APIs

AWS SQS (Simple Queue Service) provides APIs for both Standard and FIFO (First-In-First-Out) queues. While the core APIs are the same for both types of queues, there are some additional parameters specific to FIFO queues.

Common SQS APIs (these are applicable for the Standard and FFIO Queues)

- **CreateQueue:** Creates a new queue {FIFO-Specific: FifoQueue (set to true), ContentBasedDeduplication, DeduplicationScope, FifoThroughputLimit} - The queue name must end with .fifo to be considered a FIFO queue.
- **ListQueues:** Returns a list of your queues.
- **DeleteQueue:** Deletes the specified queue.
- **SendMessage:** Delivers a message to the specified queue {FIFO-Specific: MessageGroupId, MessageDeduplicationId}
- **SendMessageBatch:** Delivers up to ten messages to the specified queue {FIFO-Specific: Each entry can have MessageGroupId, MessageDeduplicationId}
- ReceiveMessage: Retrieves one or more messages (up to 10), from the specified queue.
- **DeleteMessage:** Deletes the specified message from the specified queue.
- **DeleteMessageBatch:** Deletes up to ten messages from the specified queue.
- **ChangeMessageVisibility:** Changes the visibility timeout of a specified message in a queue.
- **ChangeMessageVisibilityBatch:** Changes the visibility timeout of multiple messages.
- **PurgeQueue:** Deletes all the messages in a queue.
- **GetQueueAttributes:** Gets attributes for the specified queue.
- **SetQueueAttributes:** Sets attributes for the specified queue.
- **GetQueueUrl:** Returns the URL of an existing queue.
- **ListDeadLetterSourceQueues:** Returns a list of your queues that are configured to use a dead-letter queue.

🤞 FIFO-Specific Parameters

- **MessageGroupId:** Required for FIFO queues. Messages that belong to the same message group are always processed one by one, in a strict order relative to the message group.
- **MessageDeduplicationId:** Required for FIFO queues if ContentBasedDeduplication is not enabled. Used to avoid sending duplicate messages.
- **ContentBasedDeduplication:** If set to true, enables content-based deduplication.

While the core API actions for SQS are common to both Standard and FIFO queues, ***FIFO queues have additional parameters and constraints that ensure messages are processed exactly once and in order***. Understanding these APIs and their parameters is crucial for effectively using SQS in your applications.

Now we might need to dig deep into few of these terms & functionalities – about what implies in daily usage ..!

Message de-duplication

Message de-duplication in AWS SQS FIFO queues ensures that duplicate messages are not delivered more than once during a specified time window (the deduplication interval). This feature is crucial for maintaining exactly-once processing semantics.

Message Group ID:

- Each message in an SQS FIFO queue must have a message group ID, which is used to ensure the ordered delivery of messages within the same group.
- This ID helps in managing message ordering and ensuring that all messages within a group are processed sequentially.

Deduplication ID:

- Each message in an SQS FIFO queue has a deduplication ID. This ID is used to identify and eliminate duplicate messages.
- You can provide your own deduplication ID for each message, or if not provided, SQS can generate one based on the content of the message.

Content-Based Deduplication:

- When content-based deduplication is enabled, SQS uses a SHA-256 hash of the message body to generate the deduplication ID.
- This means that if two messages have the same content, they will be considered duplicates within the deduplication interval.

Deduplication Interval:

- The deduplication interval is 5 minutes. Within this interval, duplicate messages (messages with the same deduplication ID) will be treated as duplicates and not delivered more than once.

Key Considerations

{Much IMP for the FFIO Queue Creation and Application development}

Setting the Deduplication ID:

- When sending a message, you can specify a MessageDeduplicationId to manually control the deduplication process.
- If MessageDeduplicationId is not provided, and content-based deduplication is enabled, SQS will generate the deduplication ID based on the message content.

Content-Based Deduplication Settings:

- To enable content-based deduplication, you need to configure it when creating the FIFO queue.
- The setting is applied at the queue level and cannot be changed on a per-message basis.

Handling Duplicate Messages:

- Ensure that your application logic is designed to handle the scenario where duplicate messages are detected and eliminated.
- Understand the implications of the deduplication interval and ensure that your application does not rely on receiving duplicate messages within that time frame.

Message Group ID Management:

- Carefully plan the use of message group IDs to ensure that message ordering requirements are met.
- All messages within the same message group are processed in order, but messages across different groups are processed independently and in parallel.

Encryption

Encryption in AWS SQS is essential for several reasons, primarily centered around data security, compliance, and data integrity.

How Encryption Works in AWS SQS

Server-Side Encryption (SSE) with KMS:

- *Encryption Process:* When a message is sent to an SQS queue, it is encrypted using an encryption key managed by AWS KMS. This key can either be a customer master key (CMK) or an AWS managed CMK.
- *Decryption Process:* When a message is retrieved from the queue, SQS decrypts it using the same encryption key.

Configuring SSE in SQS:

- *Enable SSE:* SSE can be enabled when creating a new SQS queue or can be applied to existing queues.
- *Key Selection:* You can choose the KMS key that will be used to encrypt messages. This can be an AWS managed key or a customer-managed key.

Access Management

- IAM Policies to regulate access to the SQS API
- SQS Access Policies (resource-based policy)

Message Visibility Timeout

Message Visibility Timeout is a period during which a received message is hidden from other consumers in the queue. When a consumer retrieves a message from the queue, that message becomes "invisible" to other consumers for the duration of the visibility timeout. This prevents multiple consumers from processing the same message simultaneously.

Default would be 30 Seconds (Min: 0 Seconds & Max: 12 hours)

How it Works

- ***Message Retrieval:*** When a message is retrieved from the queue by a consumer, it is not deleted immediately.
- ***Visibility Timeout:*** The message becomes invisible to other consumers for a specified period (the visibility timeout).
- ***Message Processing:*** The consumer processes the message within the visibility timeout period.
- ***Message Deletion:*** If the consumer successfully processes the message, it explicitly deletes the message from the queue.
- ***Visibility Timeout Expiry:*** If the consumer does not delete the message before the visibility timeout expires, the message becomes visible again in the queue and can be received by other consumers.

Why Use Visibility Timeout?

- ***Prevent Duplicate Processing:*** Ensures that a message is not processed by multiple consumers simultaneously.
- ***Handling Failures:*** If a consumer fails to process a message, it will become visible again for another consumer to retry processing.
- ***Adjustable:*** Can be set based on the time required to process a message, offering flexibility.

Setting Visibility Timeout

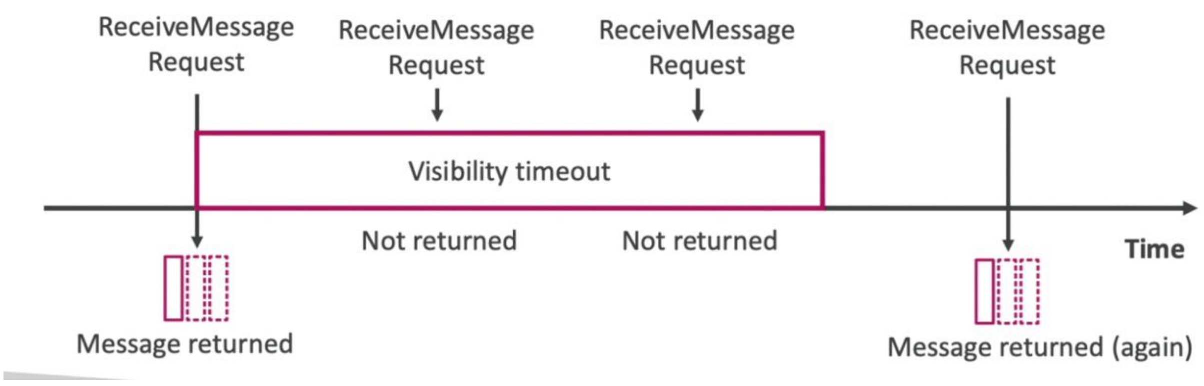
You can set the visibility timeout when creating a queue or change it for an existing queue. The default visibility timeout is 30 seconds, and it can be set from 0 seconds to 12 hours.

However, a consumer could call the **ChangeMessageVisibility API** to change the visibility timeout for that specific message. This will give the consumer more time to process the message.

Can be configured for the entire queue

- High: if the consumer crashes, re-processing will take long
- Low: may get duplicate processing of messages

Reference diagram



SQS Dead Letter Queues (DLQ)

- If a consumer fails to process the 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 MaximumReceives threshold is exceeded, the message goes into a dead letter queue (DLQ)
- 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
  PS: Good to set the retention of 14 days in the DLQ

Now we have messages in the DLQ – we go ahead and do debugging manually or with our methods & we found the issue – fixed it, now how do we put back the message into Source Queue …!!

Redrive to Source

That's where – we have feature called Redrive to Source – we can have a redrive task – which can redrive messages from the DLQ back into source queue (or any other queue) in batches without writing custom code.

AWS 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

AWS SQS – Long Polling

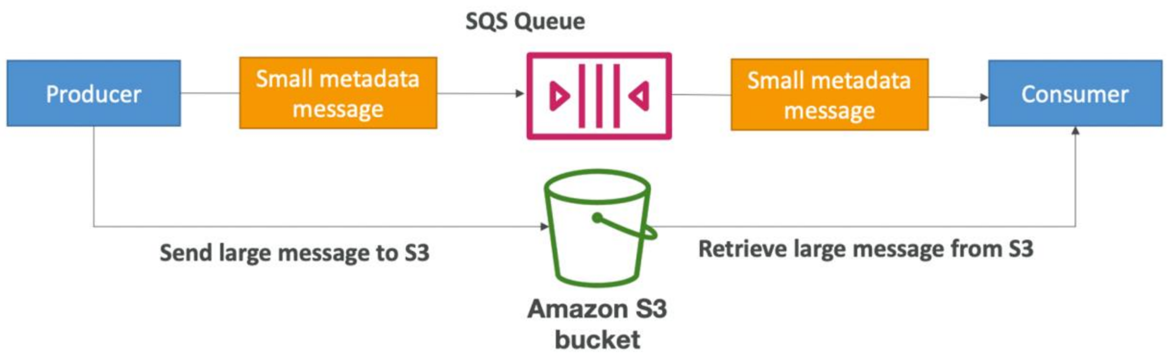
- When a consumer requests messages from the queue, it can optionally "Wait" for messages to arrive if there are none in queue, this is called Long Polling.
- Long Polling decreases the number of API Calls made to SQS while increasing the efficiency and decreasing the latency of your application.
- The wait time can be between **1 sec to 20 seconds**
- Long Polling is preferable to Short Polling
- Long Polling can be enabled at the queue level or at the API level using ReceiveMessageWaitTimeSeconds.

Short Polling: By default, SQS uses short polling, where a consumer immediately receives a response from the queue, which may be empty if no messages are available. This can lead to many empty responses and higher costs.

Long Polling: When ReceiveMessageWaitTimeSeconds is set, the consumer waits for the specified amount of time (up to 20 seconds) for a message to become available in the queue before returning a response. If a message arrives within this wait time, it is returned to the consumer immediately.

AWS SQS Extended Client

- Message size limit is 256KB, how to send large messages, e.g: 1 GB?
- Using the SQS extended Client (Java Library)
- We leverage the AWS S3 Bucket (where we keep the large messages and we take the small meta data messages point to them & we use the small messages for the Queue)



AWS SQS – a play with Numbers

- Maximum number of messages that can be retrieved from an SQS queue with a single ReceiveMessage API call is **10 messages**
- Max message size for the Queue**: 1 KB** to **256 KB**
- Message retention Period: **4 days is default (min: 1 min, max: 14 days)**
- The deduplication interval is 5 minutes **(Fixed – Can not be modified)**
- Message Visibility Timeout: **Default would be 30 Seconds (Min: 0 Seconds & Max: 12 hours)**
- Delivery Delay: **0 Seconds is default (min: 0 Seconds & max is 15 min)**
- Receive message wait time: **default would be 0 seconds (min: 0 seconds & Max is 20 seconds) – this is the parameter for the Long Polling**

Practical Questions for Practice

Numerical Questions

https://lisireddy.medium.com/aws-sqs-queue-numerical-questions-aeaed5e43426

Scenario based Questions

https://lisireddy.medium.com/aws-sqs-scenario-based-questions-559df6a1dcd5

Wish you the best …! Happy Learning ..!

Yours' Love (@lisireddy across all the platforms)

AWS SQS

Simple Queue Service