SQS, SNS, Kinesis, Active MQ

Decoupling mechanism

When we deploy multiple applications they will inevitably need to communicate. There is two ways of doing the application.

1. Synchronous communication: Direct connection for direct communication between application
2. Asynchronous / event based communication: Application is connected to a queue, then the message will come asynchronous and the application will react to that message to do whatever it needs to do

Synchronous can be a problem if there is sudden spike of traffic. It is better to decouple your application so that your application can scale independently, without worrying about direct communication

Amazon SQS (Simple queue service)

It is a queue, it will have messages. There will be a producer that will be sending messages to the SQS queue. There can be multiple producer sending the messages.

Message can be anything.

Consumer will be polling (checking for messages) the messages from the queue (hey do you have any messages for me), then will process the message then delete the queue.

There are two types of SQS, standard queues and FIFO queues.

Standard queues

The first services in AWS. Used for decoupling applications (If two applications are directly connected, we break it up by introducing an intermediate service to deliver the messages asynchronously to scale better).

For standard queue you get unlimited throughput (send as many as you want) and unlimited messages in queue.

Max retention of message: 4 days, up to 14 days. Low latency for sending and receiving it < 10ms for publish and receive.

Size < 256KB

Duplicate message can happen! Standard queue is delivery at least once occasionally service. You would need to take that into account in your application.

It is also best effort ordering, so messages can be out of order not in the order that they are sent.

Sending/reading messages

Application will produce message by using the SDK (SendMessage API).

Consumers can be running on EC2 instances, your own server, or AWS lambda (You can trigger AWS lambda via queues). Consumer poll for messages and can get up to 10 messages at a time and process them i.e. insert them into RDS database or just print it out.

After consuming message you will delete the message using DeleteMessage API.

Multiple consumers

SQS can have multiple consumers and process messages in parallel. So in the case that the consumer doesn't process the message fast enough it will be received by other consumer, since the message hasn't been deleted yet.

This is why it is at least once delivery and also best effort ordering.

ASG with metric of SQS

You can scale consumer using auto scaling group polling from SQS. The metric to scale from is the Queue Length.ApproximateNumberOfMessages. Once the length go over a certain level it will trigger the auto scaling group to spin up more EC2 instances to handle those messages. So you can handle more messages per second.

SQS as database buffer!

With the same setup as the previous statement. If we have request going into an auto scaling group and then the EC2 instances write say transactions into database in order to store records of the transactions. Sudden spike of requests will OVERWHELM the databases and some writes to the database will fail! How do we solve this? We use SQS as a buffer to store the messages that we need to write, because it is infinitely scalable, can store unlimited amount of messages. Then we can poll the messages with another auto scaling group to then process the transactions, and then the transaction are only deleted if the insertion succeed. If it failed then we don't delete the message and just try again.

Without the SQS buffer there are chances that the request writing to the database will fail and the transaction is lost! If it is done asynchronously, which mostly is.

Decoupling frontend and backend application

If you have an application that receives request to process a video and store it into a S3 bucket. If you have many request then it might not be able to handle it quickly. So you can decouple your application separating the request from the handling of the request, by storing the request into a SQS then process it with another application.

This is so that your frontend application will not be lagged or stuck at receiving request and HANDLING it at the same time. You can scale the frontend and backend independently.

SQS Security

It has in-flight encryption using HTTPS.

At-rest encryption using KMS keys.

client-side encryption needs to be done by client itself.

Access control is done by IAM policies, but you also have SQS access policies just like S3 bucket policies for allowing other services to access the SQS queue.

Message visibility timeout

After a message is polled by a consumer, it becomes invisible to other consumers for some time. That timeout is 30 seconds, which means the message has 30 seconds to be processed. But if the message hasn't been deleted it will be "put back to the queue", and other consumer can receive the same message again.

This is how the message can be deliver multiple time time if it hasn't processed fast enough.

If your application know it is processing it and needs a little bit more time before it finishes consumer can call ChangeMessageVisibility to change the default invisible time. The visibility timeout if you set it high and consumer crash seeing the message again will take long time. If too long, then duplicate messages can occur.

The visibility timeout is per message.

Long polling

Consumer can wait for messages to arrive if there are none in queue. **This is done to reduce the number of API calls to SQS queues and decrease latency for your application for the time for message to arrive to your application.** This is referred as long polling.

Long polling can last from 1 to 20 seconds. 20 seconds preferred.

You can configure it at the queue level (every poll you do will be long poll) or you do it at API level by specifying how long you will be waiting.

FIFO queues

First in first out. The message order will be guaranteed as oppose to standard queue which ordering isn't guaranteed but best effort. When you poll for messages it will receive the message in the order it was sent.

It has limited throughput, can only sent 300 messages per second or 3000 messages per second with batching.

You will only receive the messages once! And messages are processed in the order sent by the consumer.

SNS (Simple notification service)

One producer that can sent same message to many receiver! Publish-subscribe pattern.

You will have your producer sent a message to the SNS Topic, and the topic can be subscribed by consumer who would like to receive the message, there can be many subscriber, and process the message.

Producer send message to one SNS topic. Consumer listen to the topic that they want. You can have 12m+ subscription per topic which is a lot!

SNS can sent data to subscriber which includes email subscriber, mobile notification subscriber, HTTP endpoint, SQS, lambda, Kinesis data firehose.

Lots of services will publish messages too, and they will sent it to a specified SNS topic. ASG scaling event for example.

Publish

To publish message you use SDK, you create topic, create subscription, then you publish to the topic

Security

In-flight encryption, at-rest encryption using KMS keys. If client wants to handle encryption they have to do it.

IAM policies to regulate access to SNS resources. Also can define SNS policies to define what other services can access SNS.

SNS + SQS Fan out pattern

You want message to sent to multiple SQS. If you sent it individually then it might not be guaranteed. Use fan out pattern!

Push once in SNS topic, SQS will be the subscriber to the topic. This will guarantee the message be sent to the SQS topic.

This is fully decoupled, no data loss. SQS allow for data persistence, delaying the process work since is stored in queues.

You need to allow SNS to write to the SQS. SQS can also be in other region as well and it will work!

S3 event to multiple queues

You can only sent S3 event like an object being created to one location, but what if you want to sent it to multiple places? Then you can use the SNS + SQS fan out pattern. Let the event notification be publish to SNS. Then you have SQS as subscribers, then that event notification can then go to multiple places where ever you like! Services, services, other lambda!

SNS to S3 via Kinesis data firehose

Kinesis data firehose let you deliver streaming data (data that are continuously being produced, logs of customers, ...) to a destination. You can apply transformation before storing the data to say S3 bucket.

SNS can also be used to sent data to Kinesis. So you sent the data to SNS topic, then Kinesis data firehose can subscribe to it. because it let you set a destination for the streaming data you can set destination to say a S3 bucket to store it there. Or do some transformation before storing it into S3.

FIFO topic + SQS FIFO

Same like FIFO queue, the messages will be ordered for a topic that is published to.

You get ordering and no duplication. However you can only have SQS FIFO as the subscriber and limited throughput just like FIFO queue.

You use this to do fan out for SQS FIFO

Messaging filtering

You can also add a JSON policy used as a filtering policy for each subscriber. Based on the JSON policy the subscriber will receive message that it filters for.

If a subscriber doesn't have a filter policy it will receive every message.

Amazon Kinesis

It let you collect, process, and analyze streaming data in real-time. The streaming data can be from logs, metrics, IoT telemetry data. As long as the data is generated fast then is considered streaming data.

Kinesis data stream, Kinesis data firehose, Kinesis data analytics, Kinesis video streams. Four Kinesis services.

Kinesis data stream

A way for you to stream and output your streaming data to consumers.

It is made up of multiple shards that you can provision ahead of time. Each shard provide a channel for your data to float through.

Producer will be sending data to your kinesis data stream, they will be using SDK or Kinesis producer library which uses SDK at the lower level for streaming data like logs.

SDK will sent records to the kinesis data stream, record is made up of **partition key** and **data blob.** Partition key tells which shard the data will go through (after hashing the partition key, data blob is the actual data can be up to 1 MB.

The producer can sent data 1 MB/sec or 1000 record/sec per shard.

After the data is stored in Kinesis data stream it can be consumed by application using SDK or Kinesis client library which again also uses SDK. Lambda, Kinesis data firehose, or Kinesis data analytics.

The consumer receives the record which made up of **partition key**, **sequence number,** and **data blob**. Sequence number represent where it was in the specified shard.

Consumer consume at 2 MB/sec per shard for all consumer. OR
at 2 MB/sec per shard per consumer. (Enhanced fan-out pattern)

Properties of data stream

Data in data stream can be stored up to a year. The data cannot be deleted. Gives you the ability to reply the data.

Data are ordered by the same partition.

Capacity mode

Provisioned mode: You pick the number of shard the data stream has in advance.

You also pay per shard provisioned per hour.

On-demand mode: No need to provision or manage the capacity. Default capacity provision is 4 shards. You will be paying per stream per hour and data in/out per GB.

Pick provisioned mode if you know what you need in advance.

Security

IAM policies to control where the data goes.

HTTPS encryption in-flight, and at rest using KMS. You can also encrypt data yourself.

VPC endpoint can be used to Kinesis access in a VPC.

Kinesis firehose

A managed service that also takes streaming data, and then it can store it into a destination, including AWS services (S3, OpenSearch, and Redshift, need to know these by heart). You can also store it into third party destination (data dog, Splunk) or sent it to your own HTTP endpoints.

The producer for firehose are the same as data stream, can be from application, clients, that ultimately sent streaming data via the SDK or Kinesis producer library. However, in addition, it can also take Kinesis data stream as producer, CloudWatch, and SNS topics.

Streaming data can be optionally transformed via a lambda function.

The records are sent up to 1 MB per second. The data are sent to destination via **batches**. Which means the data are sent near real time since the data are sent in batches. The data will only be sent at 1 MB minimum or after 60 latency if not full batch.

You can also sent backup data or failed data to another S3 bucket for backup.

Firehose scales automatically and is serverless, and you pay for the data that is going through the firehose.

Data stream vs Firehose

- Data stream you have to write your own custom code to producer and consume. It is real time! And you would have to provision the capacity, and there are data retention with replying data stream
- Firehose is fully managed service so you don't need to configure anything. You use it to sent data to S3 / Redshift / OpenSearch / 3rd party / HTTP. However, it is near real time due to data being in batches
- No data storage option for firehose and no reply capability

Kinesis Data Analytics

Not really needed to know but here it is.

Kinesis data analytics for SQL application

The source of the streaming data can be from Kinesis Data Stream or Kinesis Data Firehose. Then you can perform SQL statement for real-time analytics.

Along the analytics you can enrich the data from a S3 source, add more data to it.

Destination that you can sent it to is the same, Data Stream and Firehose, then you can sent it to its final destination that these two services can sent to.

Kinesis data analytics for Apache Flink

Use Apache Flink on the service. Use this if you want more powerful query compared to SQL.

Source can be from Kiensis Data Stream and Amazon MSK.

This is so that you can run any Flink application on managed cluster on AWS.

It does not read from firehose!

Data ordering Kinesis vs SQS FIFO

Kinesis

For Kinesis, imagine you have 20 trucks sending their GPS coordinates to 5 shards. The partition key will be truck_id where id is the number for the truck.

Now before inserting into one of the shard for data stream, it has to figure out which shard it goes into right? Well it will take the truck_id and hash it to figure out the appropriate place it is in and then sends the record there. Just like how HashMap does. It hashes the key and insert the data.

Same key will always go to the same place, so if your truck_id doesn't change your GPS will be sent to the same shard always. Therefore data for Kinesis data stream **are ordered on shard level**. Multiple truck mapped to same shard will sent the record to the same shard.

SQS FIFO

Standard queue: No ordering is guaranteed because of best effort ordering and at least once delivery.

FIFO queue: If you do not use group ID for the messages, then the message are consumed in the order they are sent but with only one consumer.

But you can group messages together using group ID. Then the messages will be ordered FIFO in group ID level. Say group ID 1 will have a FIFO ordering, and group ID 2 will have FIFO ordering.

Then you can have multiple consumers if you have multiple group ID.

In the scenario of sending truck GPS. You would have each truck's GPS assigned a group ID. 20 trucks = 20 consumers. Still sent 300 msg per second, and 3000 if batching.

Amazon MQ

SQS, SNS are AWS protocol from AWS. You can use your own on-premise open protocol like MQTT, AMQB, TOMP, Openwire, WSS.

Amazon MQ can be used to allow migrating to the cloud. Is also a message broker service.

However, it doesn't scale as much as SQS / SNS. Runs on servers so you need Multi-AZ with failover.

To do failover, you need a standby MQ instance, they will mount to EFS, and when one AZ fails, it will failover to the standby instances.

Revision #8
Created 2023-02-21 23:29:52 UTC by Tamarine
Updated 2023-02-28 22:57:29 UTC by Tamarine