Benchmarking & Compare AWS SNS & Kafka

To keep everything running properly, microservice applications rely significantly on messaging and asynchronous communications.

One of the first crucial decisions you must make when designing services that must interact with one another is selecting the correct message broker. Making the "correct" decision might be a fight of features and edge situations that are difficult to distinguish.

In this article we will discuss 2 message brokers SNS & Kafka and it’s properties.

▼ SNS

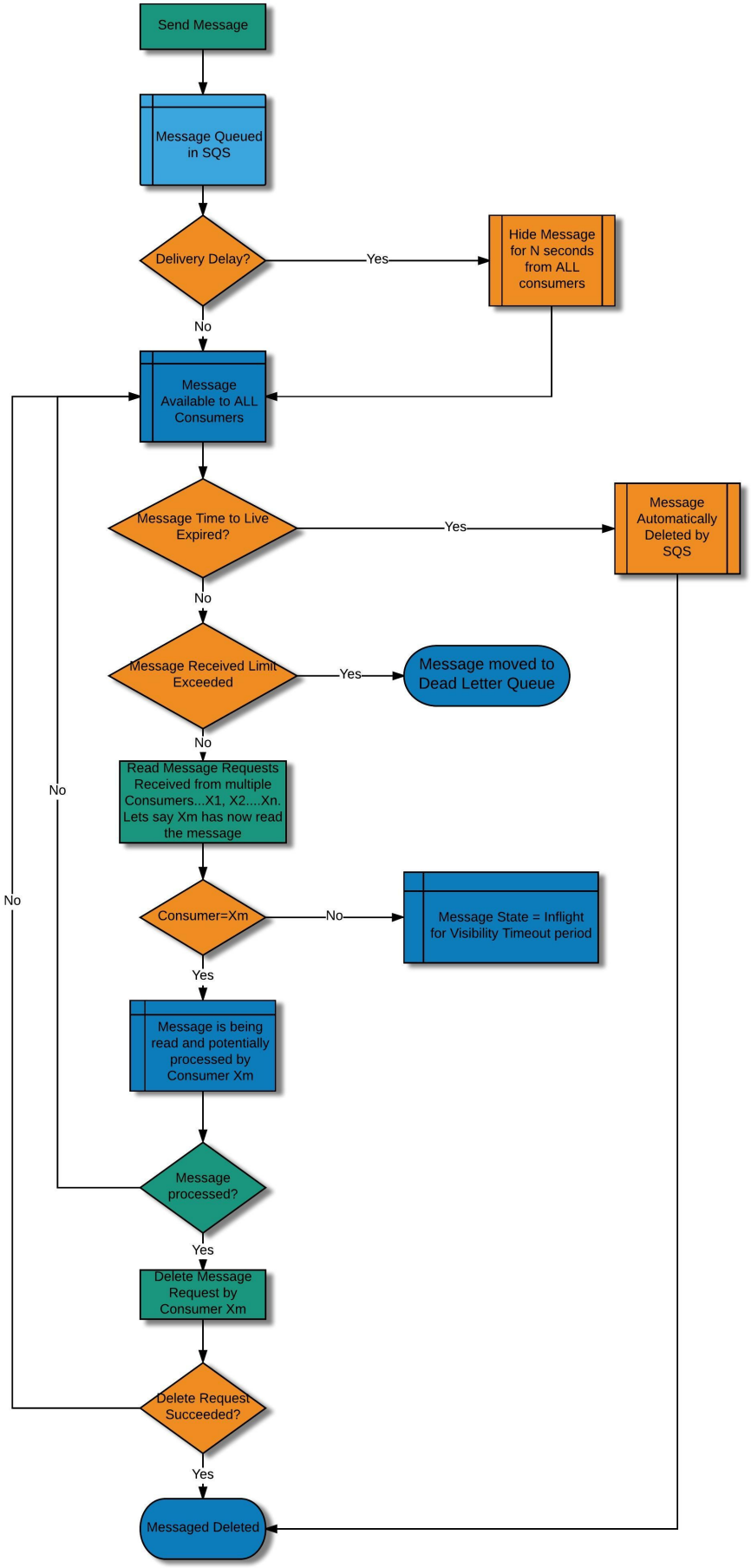
SQS and SNS are

Creating a queue service in application in AWS can be done by

- **SQS** (Simpler to use , Can expensive if you send lots of lots of events)
- **Kinesis** (Can handle large stream of incoming data such as real time data)

▼ *SQS*

SQS Message Lifecycle


Send Message
Message Queued in SQS
Delivery Delay?
Yes
Hide Message for N seconds from ALL consumers
No
Message Available to ALL Consumers
Message Time to Live Expired?
Yes
Message Automatically Deleted by SQS
No
Message Received Limit Exceeded
Yes
Message moved to Dead Letter Queue
No
Read Message Requests Received from multiple Consumers...X1, X2....Xn. Lets say Xm has now read the message
Consumer=Xm
No
Message State = Inflight for Visibility Timeout period
Yes
Message is being read and potentially processed by Consumer Xm
Message processed?
Yes
Delete Message Request by Consumer Xm
Delete Request Succeeded?
Yes
Messaged Deleted
No
No
No

Configuration

- **Visibility timeout** ➖ It's a time period in which only one consumer can access the message in the queue , and during this visibility timeout period no other consumer can access the same message . It can varies from 0 sec to 12 hr.
- **Message retention period** ➖ It's the maximum amount of time that a msg will be stored in sqs queue before sqs deletes it . It varies from 1 minute to 14 days .
- **Delivery delay** ➖ It's the amount of time to delay the first delivery of each messages . Ex- If we give this as 15 sec so only after 15 sec the messages will reach the queue .
- **Maximum message size** ➖ You can set the maximum message size for your queue. The smallest supported message size is 1 byte. The largest size is 256 KB.
- **Receive message wait time** ➖ This is the maximum time that the consumer has to wait in order to pull message from the queue (0 - 20 sec) .

Persistent of Message

- Messages are persisted for some duration if no consumer available. The retention period value is from 1 minute to 14 days. The default is 4 days.

▼ SNS Message Lifecycle

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/1ea9410a-4a8d-49c5-8b5c-00229ad90a02/Lifecycle_of_Message_in_SNS.pdf

▼ Persistent of Message

- No persistence. Whichever consumer is present at the time of message arrival, get the message and the message is deleted. If no consumers available then the message is lost

- In SQS the message delivery is guaranteed but in SNS it is not.

https://medium.com/awesome-cloud/aws-difference-between-sqs-and-sns-61a397bf76c5#:~:text=SNS %3A No persistence.,in SNS it is not.

▼ SNS Dead-Letter Queue

- Create 2 Queue's (1) Simple Queue (2) Dead-letter Queue
- Set Dead-Letter queue in Simple Queue
- Trigger Lambda in Simple Queue
- Subscribe Simple Queue to SNS Topic
- Make Sure SQS is triggered in Subscriber lambda
- Go to Configuration>**Asynchronous invocation > edit and put** Dead-letter Queue in SQS
- After that you can redrive DLQ message to Simple queue by going redrive of Dead-letter queue

▼ Q&A

Q: Will messages be delivered to me in the exact order they were published?

The Amazon SNS service will attempt to deliver messages from the publisher **in the order** they were published into the topic. However, network issues could potentially result in out-of-order messages at the subscriber end.

Q: How durable is my data once published to Amazon SNS?

SNS provides durable storage of all messages that it receives. Upon receiving a publish request, SNS stores multiple copies (to disk) of the message across multiple Availability Zones before acknowledging receipt of the request to the sender. Each AWS Region has multiple, isolated locations known as Availability Zones. Although rare, should a failure occur in one zone, the operation of SNS and the durability of your messages continue without disruption.

Q: Will a notification contain more than one message?

No, all notification messages will contain a single published message.

Q: How many times will a subscriber receive each message?

Although most of the time each message will be delivered to your application exactly once, the distributed nature of Amazon SNS and transient network conditions could result in occasional, duplicate messages at the subscriber end. Developers should design their applications such that processing a message more than once does not create any errors or inconsistencies.

Q: Will messages be delivered to me in the exact order they were published?

The Amazon SNS service will attempt to deliver messages from the publisher in the order they were published into the topic. However, network issues could potentially result in out-of-order messages at the subscriber end.

Q: Can a message be deleted after being published?

No, once a message has been successfully published to a topic, it cannot be recalled.

Q: Does Amazon SNS guarantee that messages are delivered to the subscribed endpoint?

Yes, as long as the subscribed endpoint is accessible. A message delivery fails when Amazon SNS can't access a subscribed endpoint, due to either a client-side or a server-side error. A client-side error happens when the subscribed endpoint has been deleted by the endpoint owner, or when its access permissions have changed in a way that prevents Amazon SNS from delivering messages to this endpoint. A server-side error happens when the service that powers the subscribed endpoint is unavailable, such as Amazon SQS or AWS Lambda. When Amazon SNS receives a client-side error, or continues to receive a server-side error for a message beyond the number of retries specified by the corresponding retry policy, Amazon SNS discards the message — unless a dead-letter queue is attached to the subscription. For more information, see Message Delivery Retries. and Amazon SNS Dead-Letter Queues.

Q: What happens to Amazon SNS messages if the subscribing endpoint is not available?

If a message cannot be successfully delivered on the first attempt, Amazon SNS executes a 4-phase retry policy: 1) retries with no delay in between attempts, 2) retries with minimum delay between attempts, 3) retries according to a back-off model, and 4) retries with maximum delay between attempts. When the message delivery retry policy is exhausted, Amazon SNS

can move the message to a dead-letter queue (DLQ). For more information, see Message Delivery Retries and Amazon SNS Dead-Letter Queues.

Benchmarking of AWS SNS

- [x] ~~Cost~~
- [x] ~~Through put~~
- [x] ~~Ensure that message is at least delivered~~
- [x] ~~message broker is down~~
- [x] ~~Publisher is down~~
- [x] ~~Subscriber is down~~
- [x] ~~Persistence of messages~~
- [x] ~~Security/Authentication~~
- [ ] Modes of access- API, TCP etc.
- [x] ~~Limitation~~
- [ ] Which one is suitable for a Task queue pattern
- [ ] Task queues let applications perform work, called tasks, asynchronously outside of a user request. If an app needs to execute work in the background, it adds tasks to task queues. The tasks are executed later, by worker services.

▼ Useful Links

https://www.serverless.com/guides/amazon-sns#:~:text=Yes%2C SNS supports email%2C SMS,across your entire AWS account.

https://aws.amazon.com/blogs/compute/building-resilient-no-code-serverless-patterns-by-combining-messaging-services/#:~:text=SNS has a robust retry,its subscriber has two benefits.

▼ Cost

Amazon SNS Standard topic pricing is based on the number of monthly API requests made, and the number of

deliveries to various endpoints (the cost of the delivery depends on the *endpoint type*).

Standard topics

Divided into 3 parts for Asia pacific(Mumbai)

- **API Requests**

▼ **0 ———Free———1M Request——$0.5/1M Request———Future**

▼ 1 Request → 64KB of published data

- **Notification deliveries**

▼ No charge for deliveries to Lambda but some charges apply.

▼ Price performance of Lambda running on Graviton2(Arm-processor) is 34% better then x86 processor.

▼ In Lambda 1M free request per month & 4 Lakh GB-second of compute per month.

▼ For x86 → $0.0000166667 for every GB-second , $0.20 per 1M requests

- **Data transfer**

▼ Incoming data transfer : $0.00 per GB

▼ Outgoing data transfer : ~$0.08 per GB

▼ Throughput

@https://www.serverless.com/guides/amazon-sns#:~:text=Yes%2C SNS supports email%2C SMS,across your entire AWS account.

https://docs.aws.amazon.com/general/latest/gr/sns.html

- Throughput of publish API to SNS is 1500 calls/second in ap-south-1 region **.**
- Throughput varies from region to region it's highest(30,000 calls/second) in the us-east-1 region .

▼ Ensure that Message is at least Delivered

- SNS provide guarantee msg delivery to SQS but **not mentioned about Lambda**.
- But we can get same for Lambda too by using some functionality:

1. We have to use **dead-letter queue** in the subscriber lambda. And use DLQ redrive to send again unprocessed msg.
2. We have to increase the **Retention period** of dead-letter queue so that it can hold the unprocessed messages in the DLQ.
3. We have to to use **more retry** in SNS topic and **smartly use Retry-back-off function**.
4. Amazon SNS stores each message published across **geographically-separated data centers**. If a subscribed system isn't available.
5. Amazon SNS can also archive messages in **S3 via AWS Kinesis data Firehose** subscriptions.

▼ Message Broker is Down

- If message broker(SNS) is down then you will get “NotFoundException:Topic does not exist” error.
-

▼ Publisher is Down

- In this case you **can’t send message to SNS**.
- No message present at SNS Topic.
- You should again send message by **invoking publisher lambda**.

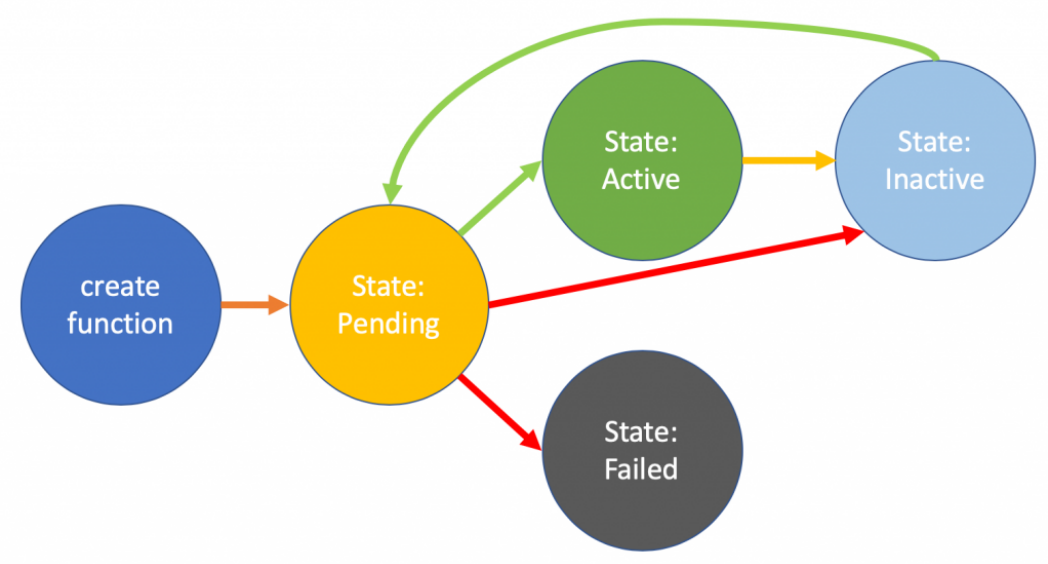
▼ Subscriber is Down

- If an initial delivery attempt does not result in a successful response from the subscriber, Amazon **SNS tries again(Retry limit is 0-100)**.

- If retry limit exceeds then **Dead-letter queue(Retention period or "**maximum amount of time that a msg will be stored in SQS queue before SQS deletes it" **is 14 day max)** is another option to prevent message from lost.
- Subscriber can down from the following reasons

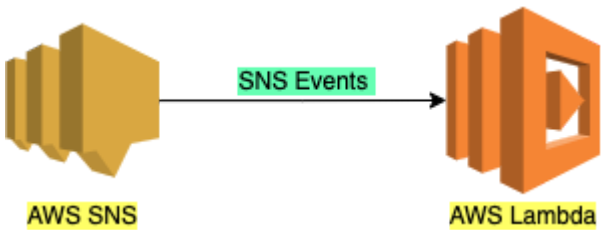
1. Web server that hosts the subscribed endpoint is **down for maintenance**.
2. Web server that hosts the subscribed endpoint is experiencing **heavy traffic**.

- If Lambda remains **idle** for several weeks it will goes down or inactive.
  - invoke that lambda again
    - From here lambda can be go to active state or failed state



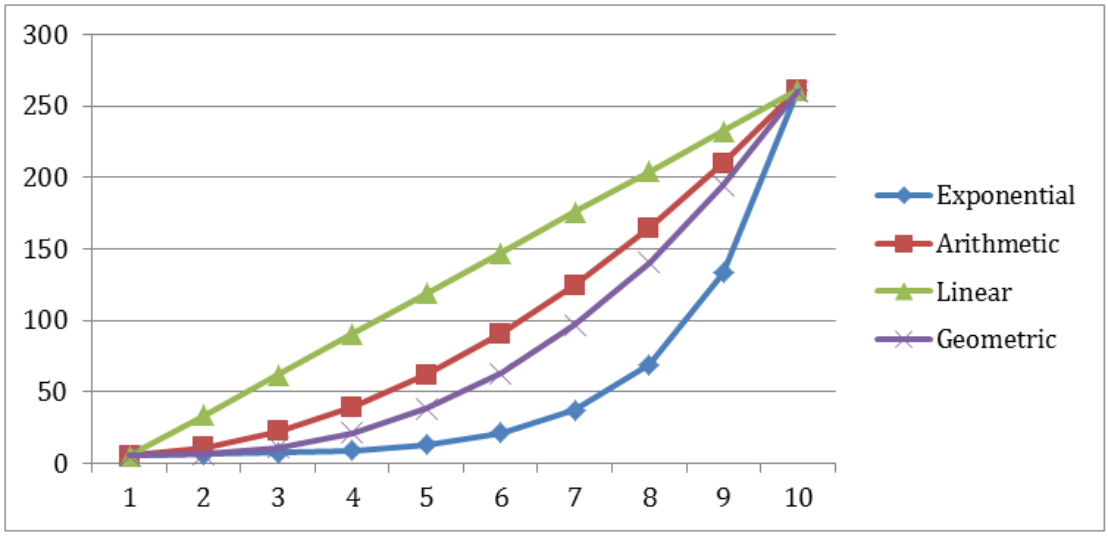
▼ Persistence of Messages in SNS

There are two common event-driven design patterns



- **No persistence**
  Whichever consumer is present at the time of message arrival, get the message and the message is deleted. If no consumers available then the message is lost.
- If SNS can't reach Lambda or the message is rejected, SNS retries at increasing intervals over several hours and then the message is lost after a few retries i.e., there is no guarantee of delivery.
- If Lambda fails an event, that particular event is lost. In the case of DLQ (Dead Letter Queue) configured, provisions need to be made for reading that separately and processing the message.

- But you can increase the persistency of SNS message by efficiently using the **no. of retry** and **dead-letter queue**.
- Insert SQS in between SNS and Lambda , This design is more persistent then the previous design.



This Figure shows how each retry backoff function affects the delay associated with retries during the backoff phase: A delivery policy with the total number of retries set to 10, the minimum delay set to 5 seconds, and the

maximum delay set to 260 seconds. The vertical axis represents the delay in seconds associated with each of the 10 retries. The horizontal axis represents the number of retries, from the first to the tenth attempt.

SNS Security/Authentication

1. **Ensure topics aren't publicly accessible**

   > Don’t use policies with `Principal` set to `""` .

   > Don’t use “*” .

2. **Privilege Access**

   > Publisher , Subscriber IAM policies should be choose wisely. Choose only those policy that you required .

3. **Implement server-side encryption**

   > It remove data leakage issues.

4.

SNS Limitations

- SNS **throughput at scale is limited** and can’t be tweaked or optimized. Throughput of email notifications sent through SNS is limited to 10 emails per second across your entire AWS account while throughput of lambda notification sent through SNS is limited to 1000.
- **High cost at scale**. with a high number of topics, subscribers, and messages the costs of SNS can eventually rise quite high, and operating your own infrastructure for SNS-like load may be more cost-effective in such cases.
- SNS focuses on short (a few KB) self-contained messages and operates at the granularity of one or a few messages (rather than streams of data) when it comes to sending messages to clients. We can’t send large

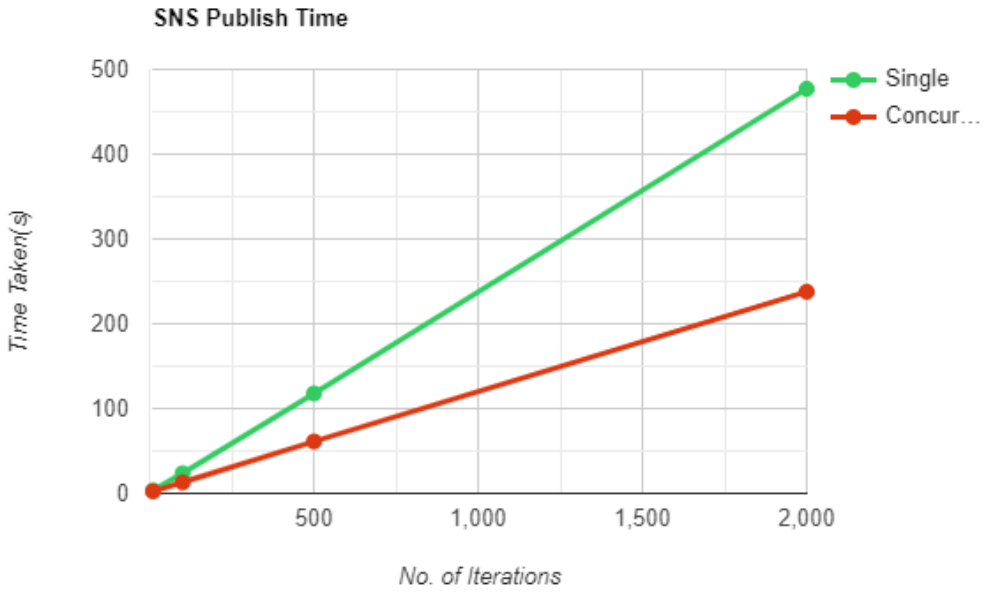
amount of data through SNS(streaming data at very high volumes is not possible).

- **Not much control over performance**. This can be a problem if SNS is a part of your customer-facing response path or if you have certain Service-level agreement to meet.
- Per account topic limit is 1 Lakh and 1.25 crore subscriptions per topic.
- SNS couldn’t Receive events from AWS CloudTrail.
- SNS doesn’t support AWS step function & Kinesis data streams as target.
- SNS **doesn’t support SaaS** integration.
- SNS **doesn’t support Schema Registry** integration.

▼ Visualizing the Publish time

▼ Concurrent v/s Single Publish Time

As you can see **Single lambda execution takes less time compared to Concurrent lambda** execution for less no. of iterations . As soon as we increase no of iterations Concurrent lambda execution takes less time as compared to single lambda execution.



▼ Message Visibility Time

▼ Resiliency in AWS SNS

- You can add resiliency in SNS by **Adding an SQS queue** between the SNS topic and its subscriber bcz messages are durably stored in the SQS queue.

▼ KAFKA

KAFKA Lifecycle

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/33fc47cd-a450-4ed1-a76d-bebc124715e3/KAFKA_Lifecycle.pdf

Benchmarking of AWS Kafka

- [x] ~~Cost~~
- [ ] Through put
- [x] ~~Ensure that message is at least delivered~~
- [ ] message broker is down
- [x] ~~Publisher is down~~
- [x] ~~Subscriber is down~~
- [x] ~~Persistence of messages~~
- [x] ~~Security/Authentication~~
- [ ] Modes of access- API, TCP etc.
- [x] ~~Limitation~~
- [ ] Which one is suitable for a Task queue pattern
- [ ] Task queues let applications perform work, called tasks, asynchronously outside of a user request. If an app needs to execute work in the background, it adds tasks to task queues. The tasks are executed later, by worker services.

Useful Links

https://www.serverless.com/guides/amazon-sns#:~:text=Yes%2C SNS supports email%2C SMS,across your entire AWS account.

https://aws.amazon.com/blogs/compute/building-resilient-no-code-serverless-patterns-by-combining-messaging-services/#:~:text=SNS has a robust retry, its subscriber has two benefits.

▼ Cost

Resources we used on AWS

- ec2 (t2.micro)
- MSK (APS3-Kafka.m5.large)
- No. of Broker = 3

- Storage of each broker = 100GB

1 Hour Cost = **$0.662916**

No charge for data transfer

Broker instance pricing

HOURLY INSTANCE USAGE You pay for Apache Kafka broker instance usage on an hourly basis (billed at one second resolution), with varying fees depending on the size of the Apache Kafka broker instance and active brokers in your Amazon MSK clusters. See the Broker Instance Pricing Tables for details.

Broker storage pricing

With Amazon MSK, you pay for the amount of storage you provision in your cluster. This is calculated by adding up the GB per broker each hour and dividing by the number of hours in the month, resulting in a value in "GB-Months," as shown in the pricing example. See the Broker Storage Pricing Tables for details.

Data transfer fees

You are not charged for data transfer between brokers or between Apache ZooKeeper nodes and brokers. You will pay standard EC2 rates for data transferred in and out of Amazon MSK clusters.

▼ Throughput

@https://www.serverless.com/guides/amazon-sns#:~:text=Yes%2C SNS supports email%2C SMS,across your entire AWS account.

https://docs.aws.amazon.com/general/latest/gr/sns.html

- Throughput of email notification sent through SNS is limited to **10/s.**
-

▼ Resiliency & Fault Tolerance

Kafka is known for it’s performance with resiliency & fault tolerance.

Fault tolerance refers to the ability of a system to continue operating without interruption when one or more of it’s components fail. Fault tolerance systems use backup components that automatically take the place of failed components, ensuring no loss of service.

Kafka provides configuration properties in order to handle adverse scenarios.

▼ Broker level fault tolerance

- **Replication Factor:** A replication factor is the number of copies of data over multiple brokers. Topic should have replication-factor >1 (usually 2 or 3) . This helps when a broker is down, so that another can serve the data of a topic.
- `replica.lag.time.max.ms`**:**An in-sync replica is a replica that fully catches up with the leader in the last 10 seconds. The time period can be configured via **replica.lag.time.max.ms.**

▼ Producer level fault tolerance

- **Kafka Producer Delivery Semantics**
  - **At Most Once:** For the at-most-Once delivery semantics it is acceptable to deliver a message either one time only or not at all. For this set Acks=0.
  - **At Least Once:** In at least once delivery semantics it is acceptable to deliver a message more than once but no message should be lost. For this set Acks=1.
  - **Exactly Once:** In exactly once delivery semantics it is acceptable to deliver a message only once and without message loss. For this set Acks=all. The Kafka producer sends the record to the broker and waits for a response from the broker. The producer will retry sending the messages based on retry config n times until received acknowledgement. The broker sends acknowledgment only after replication based on **min.insync.replica** property.
- **Properties to create a safe producer that ensures minimal data loss**
  - **Broker properties**

        - `min.insync.replicas` = 2 (at least 2) — Ensures minimum In Sync replica (ISR).
    - **Producer properties**
        - **Acks** = all (default 1) — Ensures replication before acknowledgement
        - **Retries** = MAX_INT (default 0) — Retry in case of exceptions
        - `Max.in.flight.requests.per.connection` = 5 (default) — Parallel connections to broker

- **Send messages in order**
    - `max.in.flight.requests.per.connection` (default value 5) : It represents the number of unacknowledged requests that are buffering on the producer side. If the retries is greater than 1 and the first request fails, but the second request succeeds, then the first request will be resent and messages will be in the wrong order.
- **Sending messages too fast**
    - When the producer calls send(), the messages will not be immediately sent but added to an internal buffer. The default `buffer.memory` is 32MB. If the producer sends messages faster than they can be transmitted to the broker or there is a network issue, it will exceed `buffer.memory` then the `send()` call will be blocked up to `max.block.ms` (default 1 minute). We can increase the value to mitigate the problem.

▼ Producer level fault tolerance

- `fetch.min.bytes` (default value 1MB): It defines max time to wait before sending data from Kafka to the consumer. This results in up to 500 ms of extra latency in case there is not enough data flowing to the Kafka topic to satisfy the minimum amount of data to return. If you want to limit the potential latency you can set `fetch.max.wait.ms` to a lower value.
- `Session.timeout.ms` (default value 10 seconds): Defines how long a consumer can be out of contact with the broker. When session

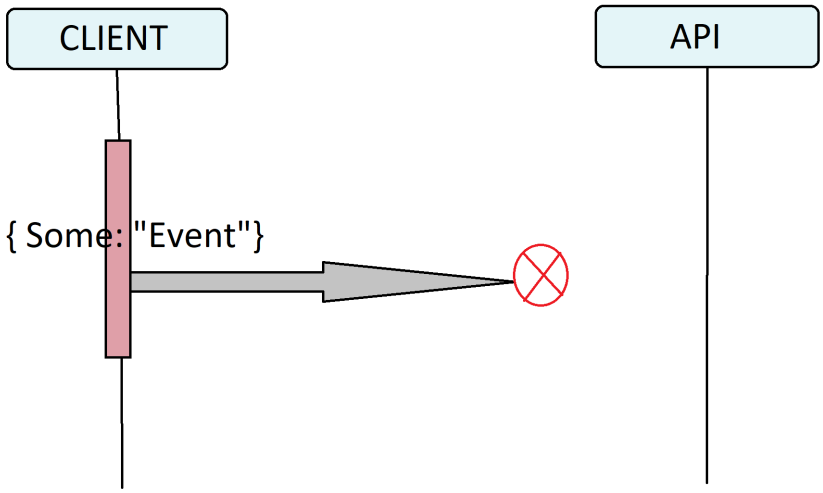
times out consumer is considered lost and rebalance is triggered, while `heartbeat.interval.ms` define how often poll method should send a heartbeat.

- `max.poll.records`: controls the maximum number of records that a single call to `poll()` will return. This is useful to help control the amount of data your application will need to process in the polling loop.
- `auto.offset.reset:` This property controls the behavior of the consumer When reading from the broker for the first time, as Kafka may not have any committed offset value, this property defines where to start reading from. You could set **“earliest”** or **“latest”**, while **“earliest”** will read all messages from the beginning “latest” will read only new messages after a consumer has subscribed to the topic. The default value of `auto.offset.reset` is “latest.”

▼ Ensure that Message is at least Delivered

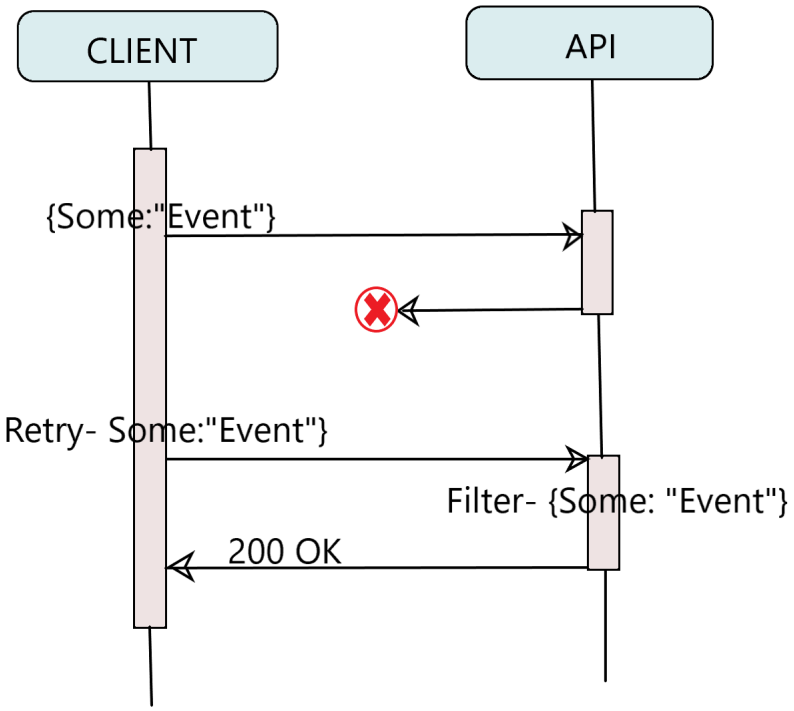
There are 3 types of message delivery semantics in Kafka:

- **At-most-once delivery**: For the at-most-once delivery semantic, a message is delivered either one time only or not at all. Failure to deliver a message is typically due to a communication error or other disruption that causes consumers to not be able to handle an event.



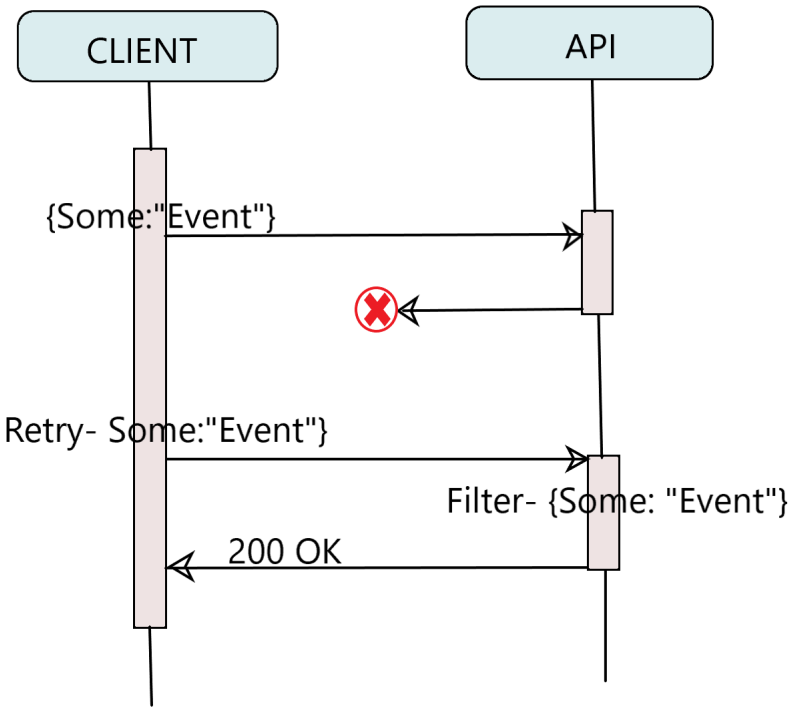
Example of Dropped Event Due to Disrupted Communication

- **At-least-once delivery:** For the at-least-once delivery semantic, a message can be delivered one or more times, but will never be lost. Duplication of events can occur due to the combination of disrupted communication and retrying events.



Example of Event Duplication in At-least-once Delivery Configuration

- **Exactly-once delivery**: For the exactly-once delivery semantic, a message will always be delivered only one time. With this configuration, a message cannot be dropped nor duplicated. Exactly-once delivery is typically achieved by filtering out duplicate events which requires maintaining state on the consumer-side in addition to the producer-side like in at-least-once configuration.



Example of Filtering Out a Duplicate Event in Exactly-once Delivery Configuration

▼ Kafka Message Broker is Down

- During a broker outage, all partition replicas on the broker become unavailable, so the affected partitions' availability is determined by the existence and status of their other replicas. If a partition has no additional replicas, the partition becomes unavailable.
- Kafka Producer library has a retry mechanism built in, however it is turned off by default. We need to change `retries` property in Producer config file to a value bigger than 0 (default value) to turn it on. 2 other parameters we can tune are `retry.backoff.ms`: Time to wait before attempting to retry a failed request to a given topic partition.
  `request.timetout.ms`: The client is going to wait this much time for the server to respond to a request.

▼ Kafka Publisher is Down

- In this case you **can’t send message to Kafka broker.**
- You should again send message by **invoking the publisher.**

▼ Kafka Subscriber is Down

- when a subscriber shuts down or crashes; it leaves the group, and the partitions it used to consume will be consumed by one of the remaining subscribers.
- Reassignment of partitions to subscribers also happen when the topics the subscriber group is consuming are modified (e.g., if an administrator adds new partitions).

▼ Persistence of Messages in Kafka

- **No persistence**
  Whichever consumer is present at the time of message arrival, get the message and the message is deleted. If no consumers available then the message is lost.
- But you can increase the persistency of SNS message by efficiently using the **no. of retry** and **dead-letter queue**.

▼ Data Retention in Kafka

Data retention can be controlled by the Kafka server and by per-topic configuration parameters. The retention of the data can be controlled by the following criteria:

- The size limit of the data being held for each topic partition:
  - `log.retention.bytes` property that you set in the Kafka broker properties file
  - `retention.bytes` property that you set in the topic configuration
- The amount of time that data is to be retained:
  - `log.retention.ms`, `log.retention.minutes`, or `log.retention.hours` properties that you set in the Kafka broker properties file

- `retention.ms`, `retention.minutes`, or `retention.hours` properties that you set in the topic configuration

▼ Kafka Security/Authentication

There are three components of Kafka Security:

Encryption of Data In-Flight Using SSL/TLS

It keeps data encrypted between our producers and Kafka, as well as our consumers and Kafka. However, we can say that it is a very common pattern everyone uses when going on the web.

Authentication Using SSL or SASL

To authenticate our Kafka Cluster, SSL and SASL allow our producers and our consumers to verify their identity. It is a very secure way to enable our clients to endorse an identity. That helps tremendously in the authorization.

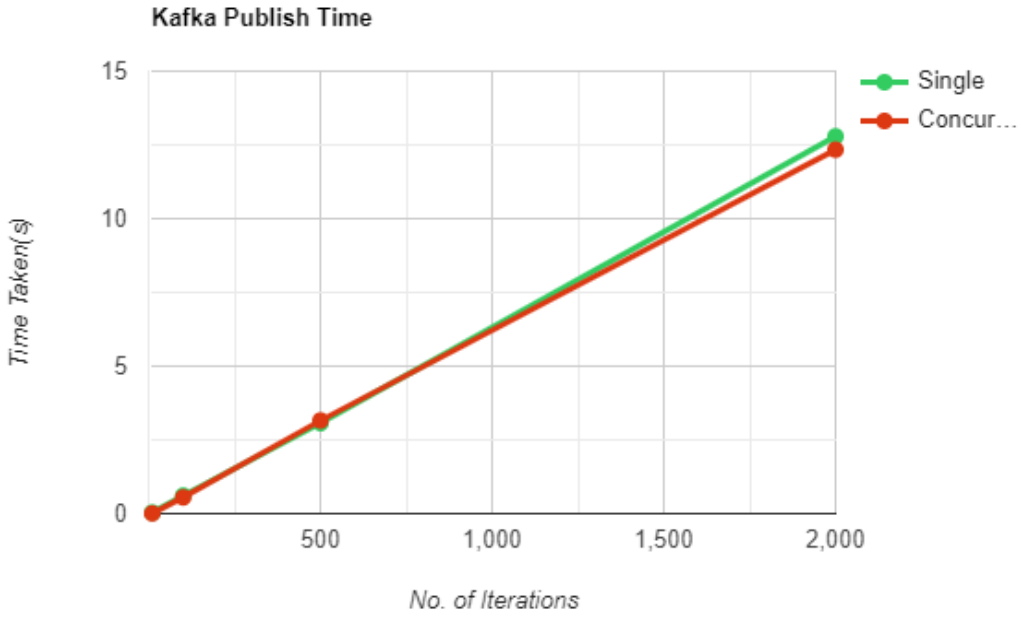
Authorization Using ACLs

In order to determine whether or not a particular client would be authorized to write or read to some topic, the Kafka brokers can run clients against access control lists (ACL).

▼ Kafka Limitations

1. https://www.javatpoint.com/apache-kafka-advantages-and-disadvantages#:~:text=Disadvantages Of Apache Kafka&text=Do not have complete set,deliver messages to the consumer.

▼ Visualizing the Publish time



How to choose which Broker will be best for us

You have to most focus on these 2 things

1. What Kinds of Messages Do You Send?
   - Message size

     Large → Choose Kafka

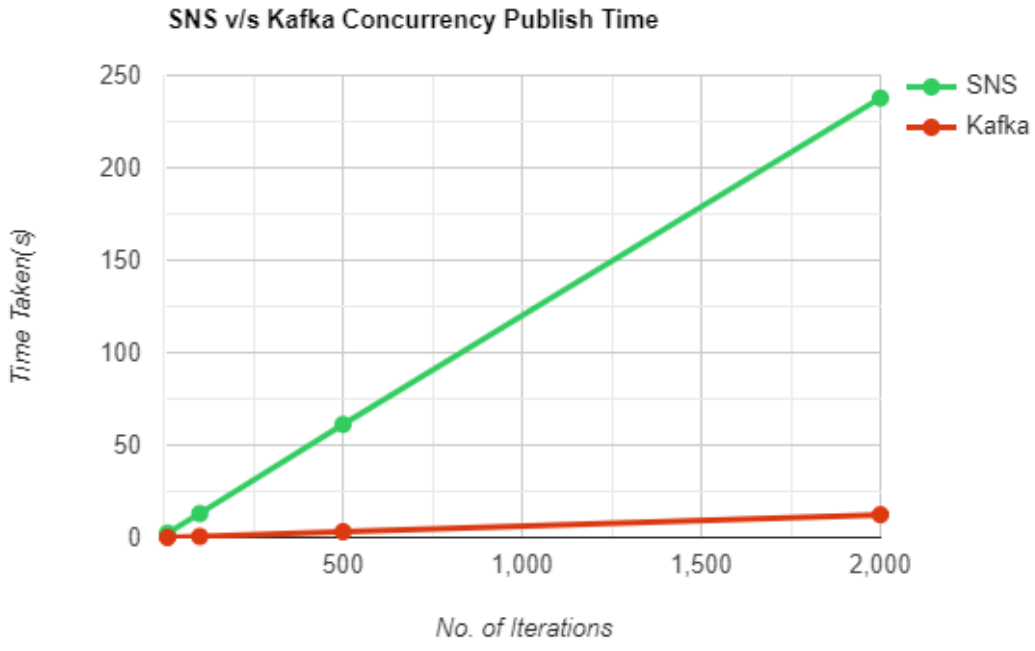
     Small → Choose SNS
2. Work and Applications' Infrastructure
   - Building Application in AWS → Choose SNS
   - Need Message retention and quickly reprocess data → Choose Kafka
   - For small Startups (Quick Setup with minimal overhead ,want good cost structure) → Choose SNS

▼ SNS v/s KAFKA

Properties	SNS	AWS Kafka
Setup-Difficulty	Easy to Setup	Difficult to Setup
Concurrency	Not Available	Available
Distributed System	No	Yes

Properties	SNS	AWS Kafka
Throughput	Limited throughput: you can do batch and concurrent requests, but still achieving high throughputs would be expensive	High Throughput
Fault-Tolerance	Low	Apache Kafka stores streaming data in a fault-tolerant way, providing a buffer between producers and consumers
Setup/maintenance	Not Required	By using ec2
Local Hosting	Not Available	Available
3rd Party Hosting	Not Available	Available
1st Party Hosting	Available	Not Available
Message size effect Cost	Cost Grow as number and size of message increase	Cost doesn't depend on size of the message
Platforms Supported	SaaS	SaaS , Windows , Mac , Linux

Message Publish durations with 2 Publisher



Message Publish durations with 1 Publisher