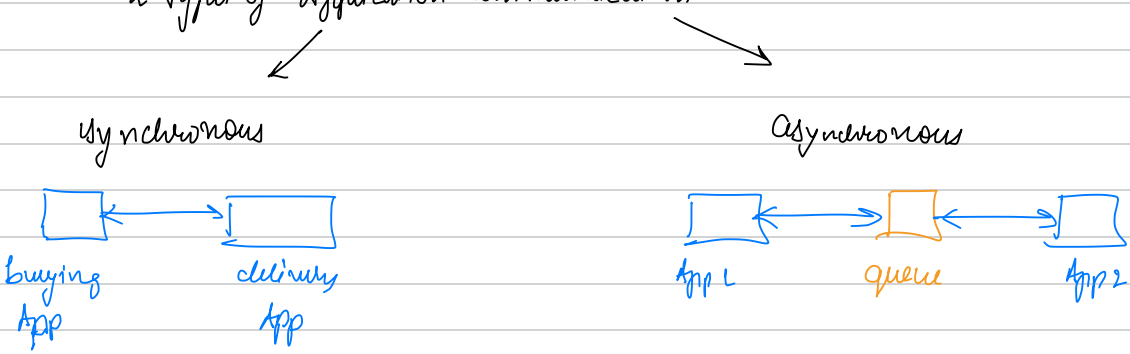
§ Messaging services { decoupling Applications }

2 types of Application communication



→ synchronous is problematic if sudden spike in traffic

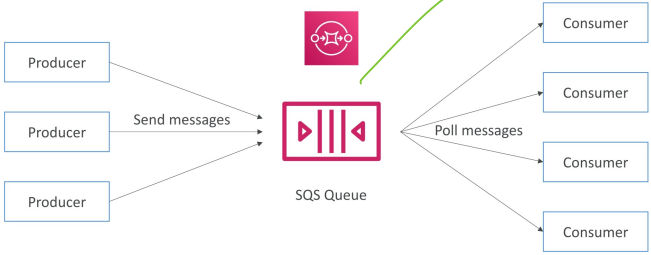
so "decouple" the applications

SQS → queue model
SNS → pub/sub model
Kinesis → real time streaming model

→ these scale independently from the application.

" The beauty of cloud when using microservices "

1 Amazon SQS { queue model }



a queuing service acts as a buffer to decouple the producers with consumers

→ oldest service

→ no limit on throughput, no limit on messages in queue

→ default retention = 4 days { max = 14 days
↳ consumer has to read within the time & delete it.

→ 256 KB / message sent

→ can have duplicate messages.

→ can have "out of order" messages.

Producing messages !

- 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

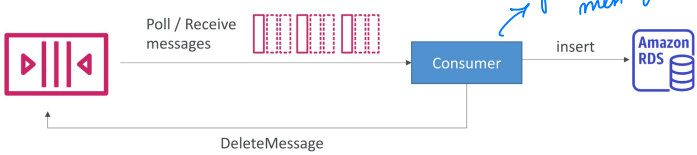
SQS – Consuming Messages

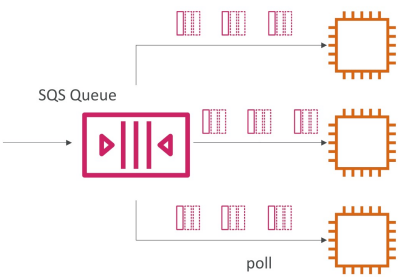
- Consumers (running on EC2 instances, servers, or AWS Lambda)…
- Poll SQS for messages (receive up to 10 messages at a time)
- Process the messages (example: insert the message into an RDS database)
- Delete the messages using the DeleteMessage API

consumer call the poll function.

process the message

deleting is necessary तभी तो दूसरा queue में जाएगा



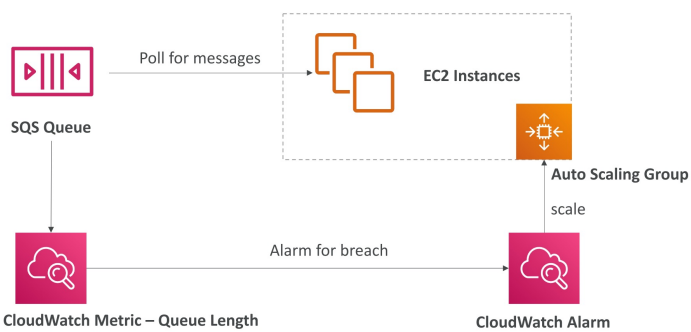


- Consumers receive and process messages in parallel
- At least once delivery
- Best-effort message ordering
- Consumers delete messages after processing them
- We can scale consumers horizontally to improve throughput of processing

say some msg is not processed by the consumer then the other consumers may receive it

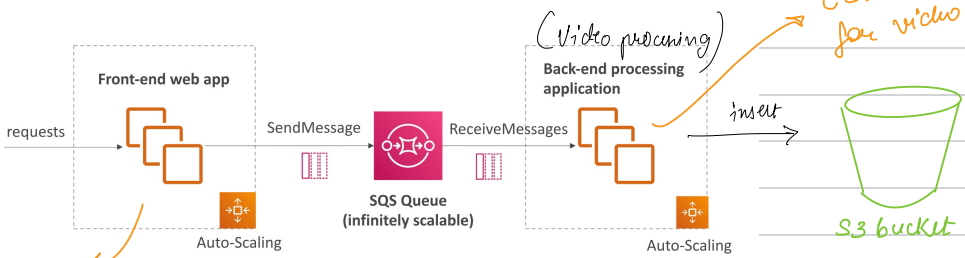
or else other consumers may see

using ASG



metric for ASG to scale according to Queue length.

SQS to **decouple** between application tiers



(Video processing)

EC2 instance GPUs for video processing

EC2 for front end delivery

directly processing & uploading will take time so

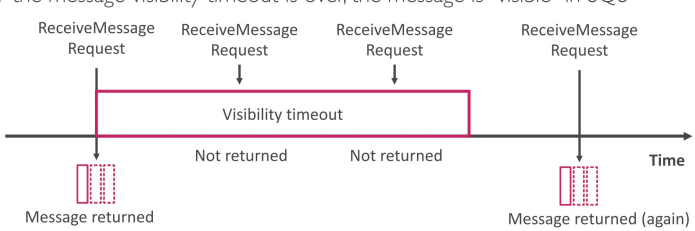
Amazon SQS - Security

- 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

Queue don't & explore this Purge delete all messages

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



After the timeout if the msg is not processed then it will be again in the queue & can't be polled again.

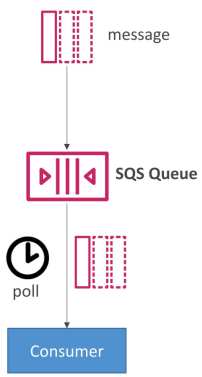
- If a message is not processed within the visibility timeout, it will be processed twice
- A consumer could call the ChangeMessageVisibility API to get more time
- If visibility timeout is high (hours), and consumer crashes, re-processing will take time
- If visibility timeout is too low (seconds),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

so if a msg needs more time to be processed then the consumer can make an API call to get more time

if timeout (↑) then reprocess after crash takes time
time out (↓) then duplicate messages.

Amazon 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



→ can be enabled @ queue level or its API level

Amazon SQS – FIFO Queue

guarantee ordering but constrains throughput

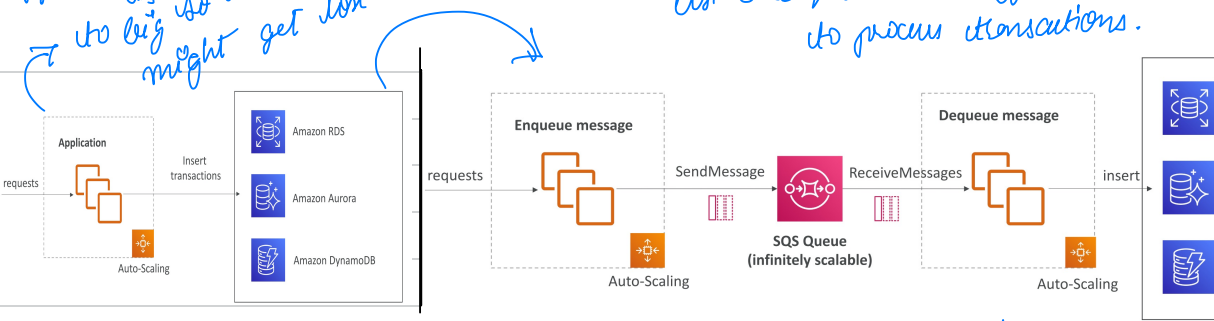
- FIFO = First In First Out (ordering of messages in the queue)

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

→ queue name should end with .fifo

If load is too big so transactions might get lost

Use SQS queue as a buffer to process transactions.

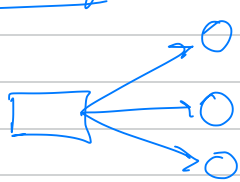


* two bars the producer won't be acknowledged that whether the OLTP transaction is made or not but we may assume that SQS mei chale gaya to ho hi jayegi !

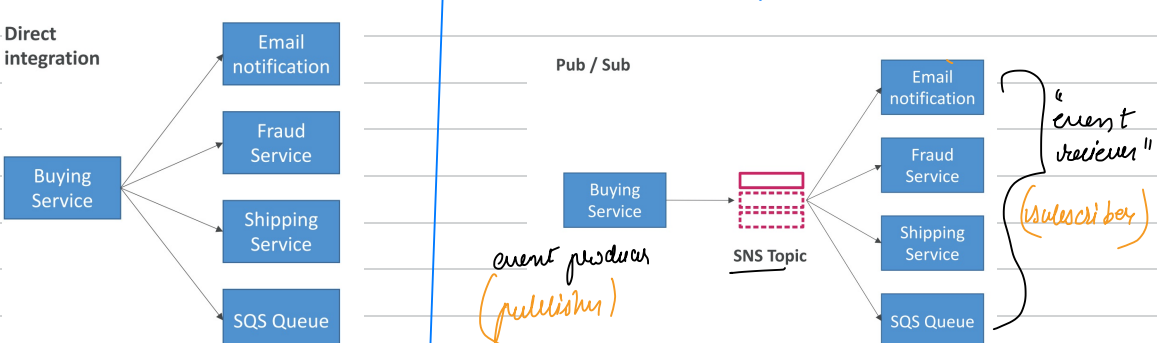
2) Amazon SNS { PUB/SUB model }

one producer → multiple consumers

direct integration can be cumbersome !

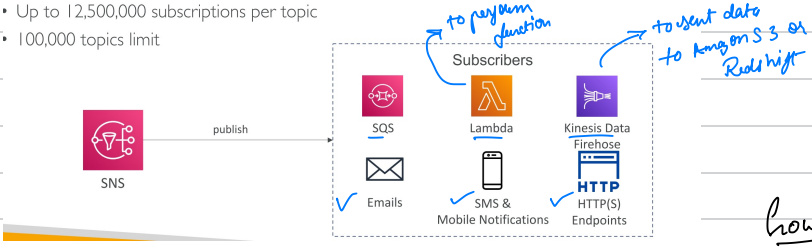


↳ so use the PUB/SUB architecture



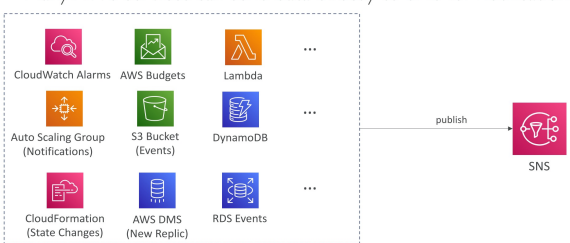
Amazon SNS

- The "event producer" only sends message to one SNS topic
- As many "event receivers" (subscriptions) as we want to listen to the SNS topic notifications
- Each subscriber to the topic will get all the messages (note: new feature to filter messages)
- Up to 12,500,000 subscriptions per topic
- 100,000 topics limit



how to publish ?

- Many AWS services can send data directly to SNS for notifications



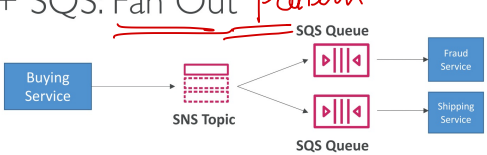
- Topic Publish (using the SDK)
  - Create a topic
  - Create a subscription (or many)
  - Publish to the topic
- Direct Publish (for mobile apps SDK)
  - Create a platform application
  - Create a platform endpoint
  - Publish to the platform endpoint
  - Works with Google GCM, Apple APNS, Amazon ADM...

Amazon SNS – Security

(same as SQS)

- 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 SNS API
- SNS Access Policies (similar to S3 bucket policies)
  - Useful for cross-account access to SNS topics
  - Useful for allowing other services (S3...) to write to an SNS topic

SNS + SQS: Fan Out Pattern



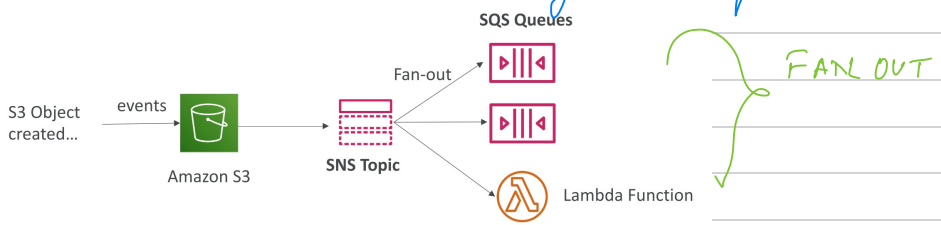
→ SQS queues as subscribers for the SNS topic.

- Push once in SNS, receive in all SQS queues that are subscribers
- Fully decoupled, no data loss
- SQS allows for: data persistence, delayed processing and retries of work
- Ability to add more SQS subscribers over time
- Make sure your SQS queue access policy allows for SNS to write
- Cross-Region Delivery: works with SQS Queues in other regions

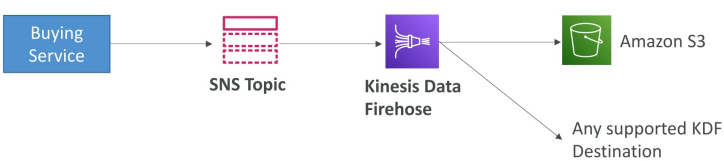
use case for Fan out :

→ For a object & same prefix there can be only one S3 event rule.

And say we want to different jobs for the same event so we can use the fan-out pattern.



* SNS can send to S3 through KDF (Kinesis Data Firehose)

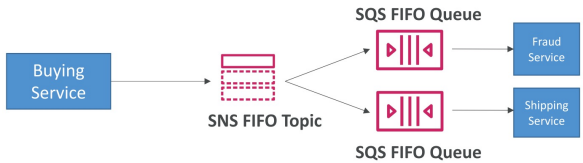


Amazon SNS – FIFO Topic

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

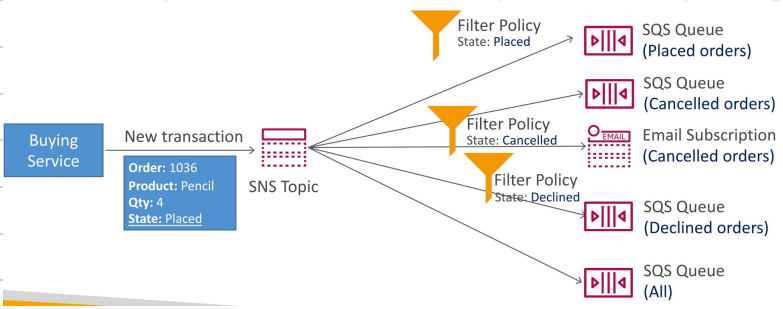
- Similar features as SQS FIFO:
  - Ordering by Message Group ID (all messages in the same group are ordered)
  - Deduplication using a Deduplication ID or Content Based Deduplication
- Can have SQS Standard and FIFO queues as subscribers
- Limited throughput (same throughput as SQS FIFO)

→ for a fanout + ordering + deduplication



SNS – Message Filtering

- JSON policy used to filter messages sent to SNS topic's subscriptions
- If a subscription doesn't have a filter policy, it receives every message



go to SNS > create topic { · Standard · FIFO }

3 Amazon Kinesis

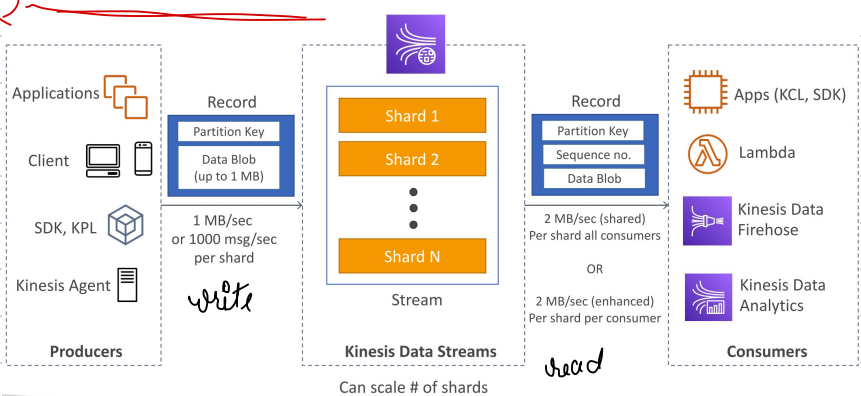
Kinesis Overview

- Makes it easy to collect, process, and analyze streaming data in real-time
- Ingest real-time data such as: Application logs, Metrics, Website clickstreams, IoT telemetry data…

1 2 3 4

- **Kinesis Data Streams:** capture, process, and store data streams
- **Kinesis Data Firehose:** load data streams into AWS data stores and also outside
- **Kinesis Data Analytics:** analyze data streams with SQL or Apache Flink
- **Kinesis Video Streams:** capture, process, and store video streams

a Kinesis Data Streams



Can scale # of shards

→ shards need to be provisioned ahead of time. data is split across shards & shards determine the stream capacity interms of ingestion and consumption rate.

→ how does my every producer relies on SDK

KPL (Kinesis Producer library), KCL (Kinesis Client Library.

producer → KDS → Record { • Partition Key → which shard will record go • Data Blob [≤ 1MB] → data itself. }

rate = 1MB/s or 1000 msg / sec per shard.

KDS → consumer → Record { • Partition Key • Sequence Key → where the record was in shard • Data blob }

shard = 2 MB / s of throughput shared for all the consumers per shard

or

enhanced = 2 MB / sec per shard per consumer
Fan-out

Kinesis Data Streams

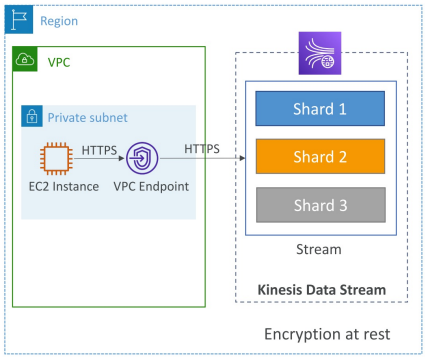
- Retention between 1 day to 365 days
- Ability to reprocess (replay) data
- Once data is inserted in Kinesis, it can't be deleted (immutability)
- Data that shares the same partition goes to the same shard (ordering)
- Producers: AWS SDK, Kinesis Producer Library (KPL), Kinesis Agent
- Consumers:
  - Write your own: Kinesis Client Library (KCL), AWS SDK
  - Managed: AWS Lambda, Kinesis Data Firehose, Kinesis Data Analytics,

Kinesis Data Streams – Capacity Modes

- Provisioned mode:
  - You choose the number of shards provisioned, scale manually or using API
  - Each shard gets 1MB/s in (or 1000 records per second)
  - Each shard gets 2MB/s out (classic or enhanced fan-out consumer)
  - You pay per shard provisioned per hour
- On-demand mode:
  - No need to provision or manage the capacity
  - Default capacity provisioned (4 MB/s in or 4000 records per second)
  - Scales automatically based on observed throughput peak during the last 30 days
  - Pay per stream per hour & data in/out per GB

Kinesis Data Streams Security

- Control access / authorization using IAM policies
- Encryption in flight using HTTPS endpoints
- Encryption at rest using KMS
- You can implement encryption/decryption of data on client side (harder)
- VPC Endpoints available for Kinesis to access within VPC
- Monitor API calls using CloudTrail



go to Amazon Kinesis > create a data stream >

after creating data stream we can edit the no. of shards min = 1, max = 2 { limit increase request }

to test we use SDK as consumer & producer!
{ kinesis-data-streams.sh follow this }

on CLI:

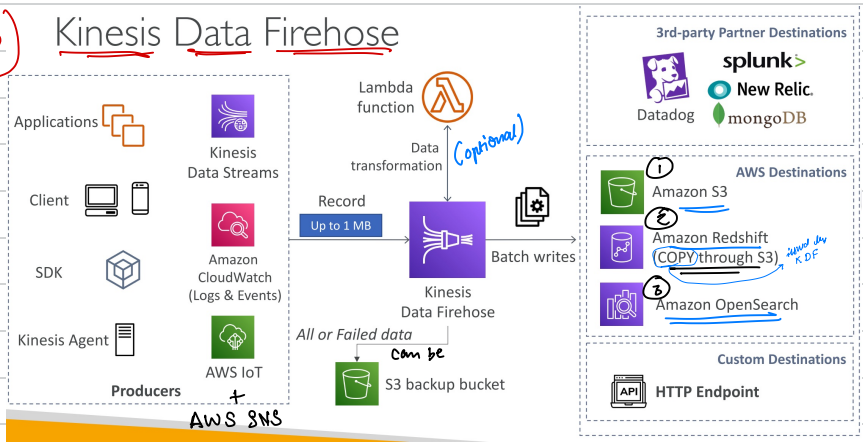
producer
1. To put data into KDS { api call put-record
2. specify partition key for shard & the data.

* to be able to read we need to know which shard ID to fetch from { describe-stream api call }
* but if using KCL then all of this is handled

consumer
1. consumer { get-shard-iterator api } with stream name
2. shard ID
3. how do you want to iterate the shard { TRIM-HORIZON } (from beginning) → shard iterator type.

(4) to read records (get - record api call)

(5) Kinesis Data Firehose



take data from producers and write the data to destination!

Kinesis Data Firehose

- Fully Managed Service, no administration, automatic scaling, serverless
  - AWS: Redshift / Amazon S3 / OpenSearch
  - 3rd party partner: Splunk / MongoDB / DataDog / NewRelic / …
  - Custom: send to any HTTP endpoint
- Pay for data going through Firehose
- Near Real Time
  - Buffer interval: 0 seconds (no buffering) to 900 seconds
  - Buffer size: minimum 1MB max = 128 MB
- Supports many data formats, conversions, transformations, compression
- Supports custom data transformations using AWS Lambda
- Can send failed or all data to a backup S3 bucket

if buffer is not filled in this time then the buffer is flushed.

if buffering then near Real time! → data is provided if the buffer size hits

Kinesis Data Streams vs Firehose

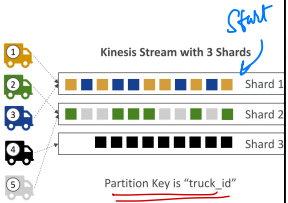
Kinesis Data Streams	Kinesis Data Firehose
• Streaming service for ingest at scale	• Load streaming data into S3 / Redshift / OpenSearch / 3rd party / custom HTTP
• Write custom code (producer / consumer)	• Fully managed
• Real-time (~200 ms)	• Near real-time
• Manage scaling (shard splitting / merging)	• Automatic scaling
• Data storage for 1 to 365 days	• No data storage
• Supports replay capability	• Doesn't support replay capability

how is ordering KDS different from SQS?

Ordering data into Kinesis

- Imagine you have 100 trucks (truck_1, truck_2, ..., truck_100) on the road sending their GPS positions regularly into AWS.
- You want to consume the data in order for each truck, so that you can track their movement accurately.
- How should you send that data into Kinesis?

- Answer: send using a "Partition Key" value of the "truck_id"
- The same key will always go to the same shard



→ Kinesis will hash the partition key to know which shard it is going to

→ data is in order @ shard level

Ordering data into SQS

- For SQS standard, there is no ordering.
- For SQS FIFO, if you don't use a Group ID, messages are consumed in the order they are sent, with only one consumer

- You want to scale the number of consumers, but you want messages to be "grouped" when they are related to each other
- Then you use a Group ID (similar to Partition Key in Kinesis)

→ if group ID is not used then there will be just one consumer.

→ But we use group IDs then we can group the FIFO messages and also send it to multiple consumers!

→ similar to how we write into shards in Kinesis

Kinesis vs SQS ordering

- Let's assume 100 trucks, 5 kinesis shards, 1 SQS FIFO
- Kinesis Data Streams:
  - On average you'll have 20 trucks per shard
  - Trucks will have their data ordered within each shard
  - The maximum amount of consumers in parallel we can have is 5
  - Can receive up to 5 MB/s of data
- SQS FIFO
  - You only have one SQS FIFO queue
  - You will have 100 Group ID
  - You can have up to 100 Consumers (due to the 100 Group ID)
  - You have up to 300 messages per second (or 3000 if using batching)

SQS vs SNS vs Kinesis

SQS:

- Consumer "pull data"
- Data is deleted after being consumed
- Can have as many workers (consumers) as we want
- No need to provision throughput
- Ordering guarantees only on FIFO queues
- Individual message delay capability

SNS:

- Push data to many subscribers
- Up to 12,500,000 subscribers
- Data is not persisted (lost if not delivered)
- Pub/Sub
- Up to 100,000 topics
- No need to provision throughput
- Integrates with SQS for fan-out architecture pattern
- FIFO capability for SQS FIFO

Kinesis:

- Standard: pull data
  - 2 MB per shard
- Enhanced-fan out: push data
  - 2 MB per shard per consumer
- Possibility to replay data
- Meant for real-time big data, analytics and ETL
- Ordering at the shard level
- Data expires after X days
- Provisioned mode or on-demand capacity mode

(4) Amazon MQ

71

Amazon MQ

- SQS, SNS are "cloud-native" services: proprietary protocols from AWS
- Traditional applications running from on-premises may use open protocols such as: MQTT, AMQP, STOMP, Openwire, WSS
- When migrating to the cloud, instead of re-engineering the application to use SQS and SNS, we can use Amazon MQ
- Amazon MQ is a managed message broker service for

Managed RabbitMQ & ActiveMQ.

- Amazon MQ doesn't "scale" as much as SQS / SNS
- Amazon MQ runs on servers, can run in Multi-AZ with failover
- Amazon MQ has both queue feature (~SQS) and topic features (~SNS)

Amazon MQ – High Availability archi.