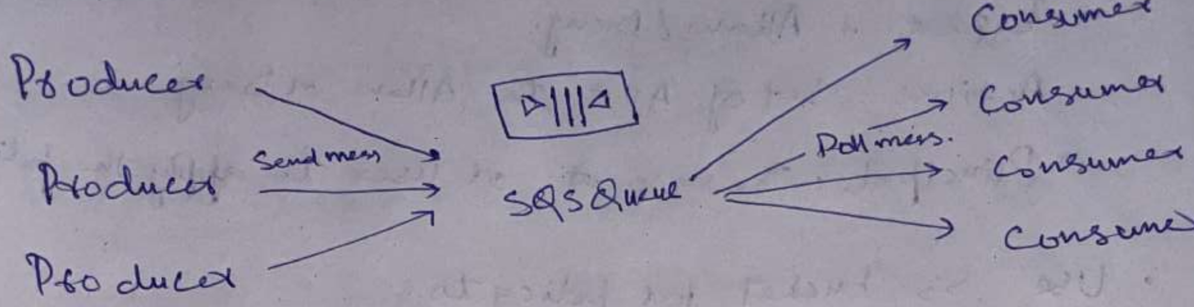
Amazon SQS:

What's a queue?



→ Oldest offering (over 10 years old)

→ Fully managed service, used to decouple application.

Attributes

→ Unlimited Throughput, unlimited numbers of messages

→ Default retention of messages: 4 days, max 14.

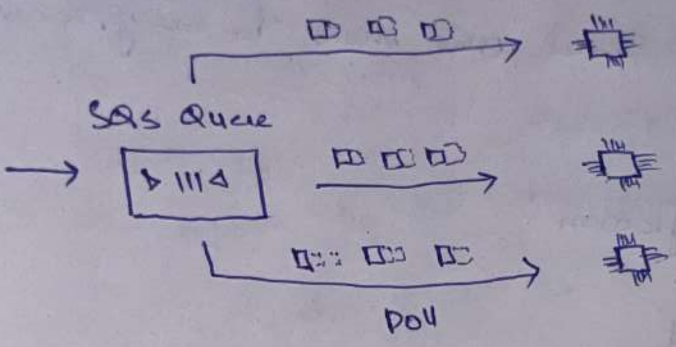
→ Low latency. <10ms

→ Limitation of 1,024 KB per mess. sent.

→ Can have duplicate message.

→ Can have out of order message.

SQS - Multiple EC2 instances Consumers.



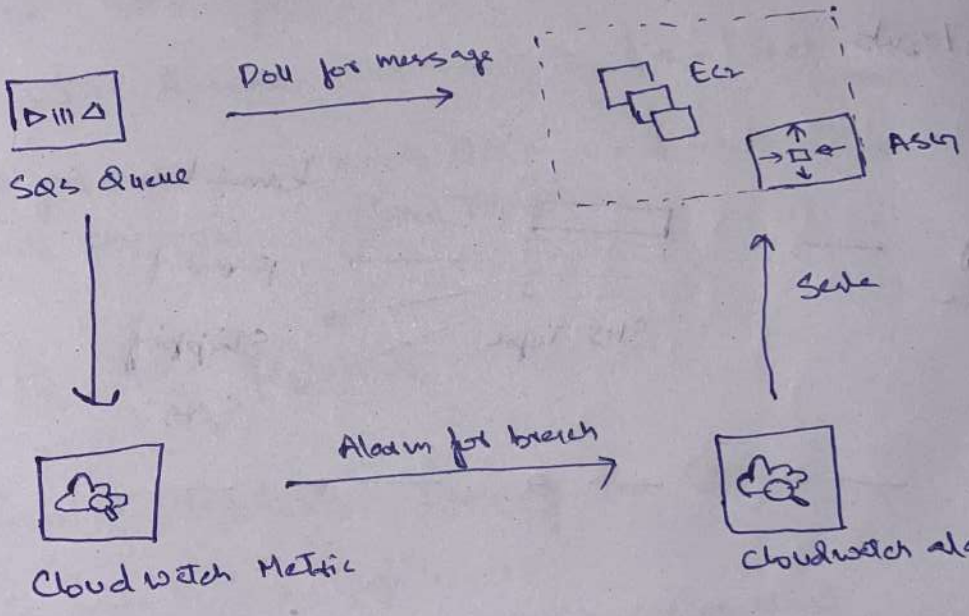
→ Consumers receive & Process message in parallel.

→ At least once delivery

→ Best-effort message ordering

→ consumers delete messages after processing them

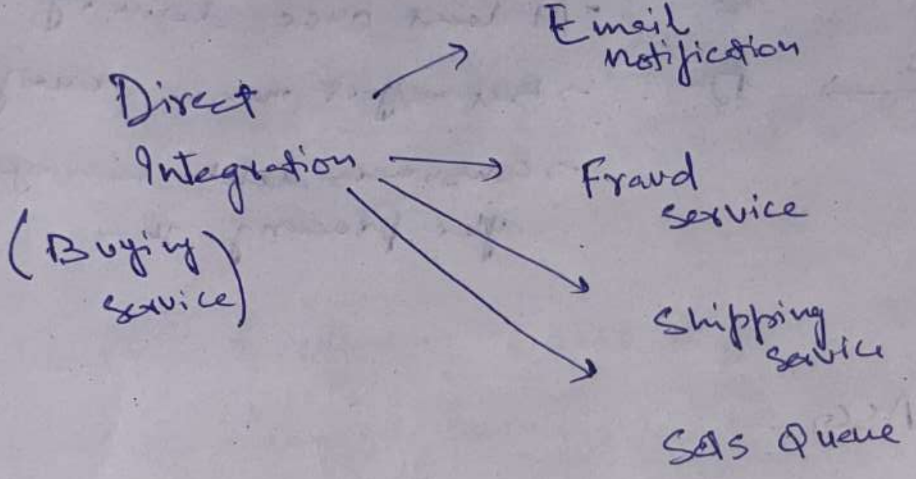
SQS with ASG.



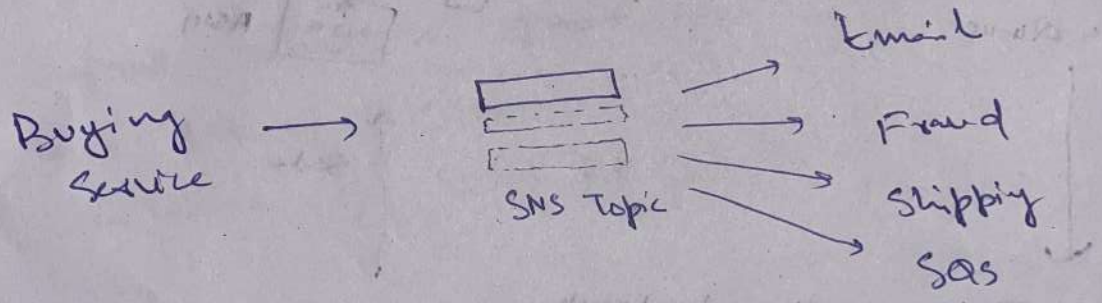
We can use SQS as a Buffer to database write.

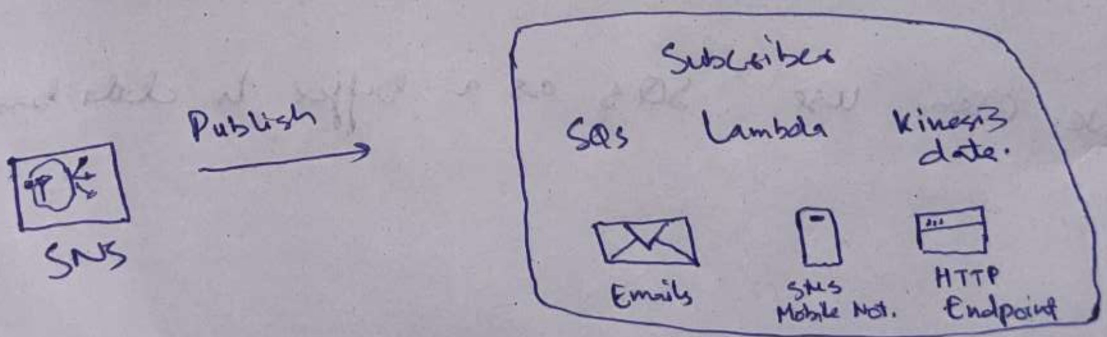
Amazon - SNS

What if we want to send one mess. to many receivers)



Pub/sub





Amazon - SNS

→ The "event producer" only sends message to one SNS Topic
→ As many "event receivers" (subscriptions) as we want to listen to listen to the SNS Topic notifications.
→ Each subscriber to the Topic will get all the messages
→ upto 12,500,000 Subscriptions per topic.
→ 100,000 topic limit.

→ Many AWS services can send data directly to SNS for notification
Cloudwatch Alarms, AWS Budgets, Lambda, ASG, (Noti) S3 Events, dynamoDB
Cloudformation (state changes), AWS DMS (New replic), RDS Events.

How to publish.

→ Topic Publish (using the SDK)
→ Create a topic
→ Create a subscription (or any)
→ Publish to the topic.

→ Direct Publish (for mobile apps SDK)
→ Create a platform application
→ Create a platform endpoint
→ Publish to the platform endpoint.
→ work with google GCM, Apple APNS, Amazon ADM.

Amazon SNS - 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 Control : 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.

Amazon Kinesis Data streams (No free tier).

→ It is fully managed real-time data streaming service from AWS.

→ It Collects your process, and analyze streaming data (like Logs, Click streams, IOT data, or events) as it arrives, with very low latency.

→ Key points (quick):

→ Handles continuous data streams in real time.

→ Data is split into shards for scalability.

→ Producers send data → Consumers process it.

→ Data can be replayed for Configurable time (upto 365 days)

→ Built for high throughput and fault tolerance.

→ Common use cases:

→ Log and event processing

→ Real-time analytics dashboards.

→ Monitoring & alerting.

→ IoT and click stream analysis.

Think of it as a high - speed, real-time pipeline for streaming data.

$\downarrow$

Prog, bqui.

Multiplayer Gaming.

Live vedio streaming & calls.

ML → dashboard.

data → S3

Amazon Date Firehose.

· Note: Used to be called "Kinesis data Firehose".

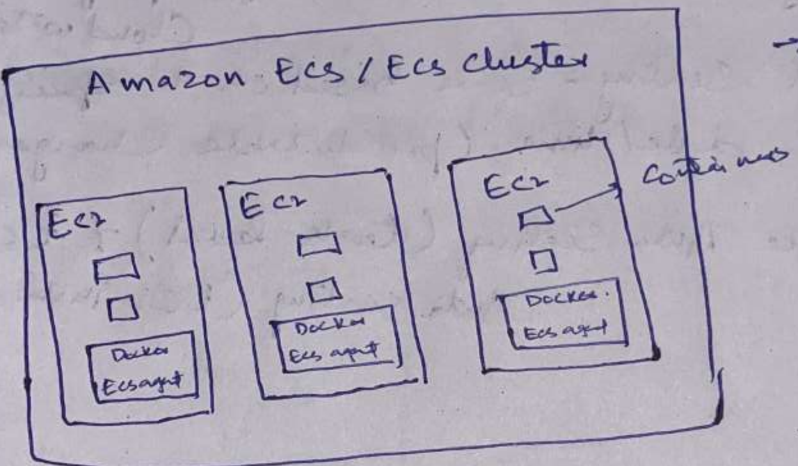
· Fully Managed service.
  → Amazon Redshift / Amazon S3 / Amazon Open Search Service
  → 3rd party: Splunk / MongoDB / Datadog / New relic..
  → Custom HTTP end point.
· Automatic scaling, serverless, Pay for what you use.
· Near Real-Time with buffering capability based on size/time.
· Supports CSV, JSON, Parquet, Avro, Raw text, Binary data.
· Conversions to parquet/ORC, compressions with gzip/snappy.
· Custom data transformation using AWS Lambda.

SQS	SNS	Kinesis
· Consumer "Pull data"	· Push data to many subscribers	· Standard: pull data · 2 MB per shard.
· Data is deleted after being consumed.	· upto 12,500,000 subscribers.	· Enhanced-fan out: push data · 2 MB per customer.
· Can have as many workers (consumers) as we want.	· Data is not persisted (lost if not delivered)	· Possibility to replay data.
· No need to Provision Throughput	· Pub/sub · upto 100,000 topics	· Meant for real-time big data, analytics. ETL
· ordering guarantees only on FIFO queues	· No need to provision through put	· Ordering the shard level · Data expires after x days
· individual message delay capability.	· Integrates with SQS for Fan out arch. pattern	· Provisioned mode for non-demand capacity mode.
	· FIFO Capability for SQS FIFO	

Section 18: Containers on AWS (Docker)
ECS, Fargate, ECR & EKS.

Amazon ECS: EC2 launch Type

ECS = Elastic Container Service.
→ Launch Docker Containers on AWS = launch ECS tasks on ECS clusters.
→ EC2 launch type: you must provision & maintain the infra structure (the EC2 instances).
→ Each EC2 instances must run the ECS agent to register in the ECS cluster.



→ AWS takes care of starting / stopping containers.

Fargate launch type.
- Launch Docker Containers on AWS
- You dont provision the infrastructure
- its all serverless
- You just runs ECS tasks for you based on the CPU / RAM you need.
- To scale, just increase the number of tasks simple - no more EC2 instances.

ECS servic Auto Scaling.

→ Auto matically inc/dec the derised no. of ECS tasks.

→ Amazon ECS Auto Scaling uses AWS app Auto Scaling.
  → ECS service Average CPU utilization
  → ECS Service Average Memory utilization Scale on RAM
  → ALB Request count per target.

→ Target tracking - Scale based on target value for a specific CW metric

→ step scaling - Scale based on ~~target~~ specified Cloud watch Alarm.

→ Scheduled Scaling - Scale based on a specified date/time (predictable changes)

→ ECS Service Auto Scaling (tasks level) ≠ EC2 Auto scaling (EC2 instance level).

Amazon ECR (Docker hub)

ECR = Elastic Container Registry

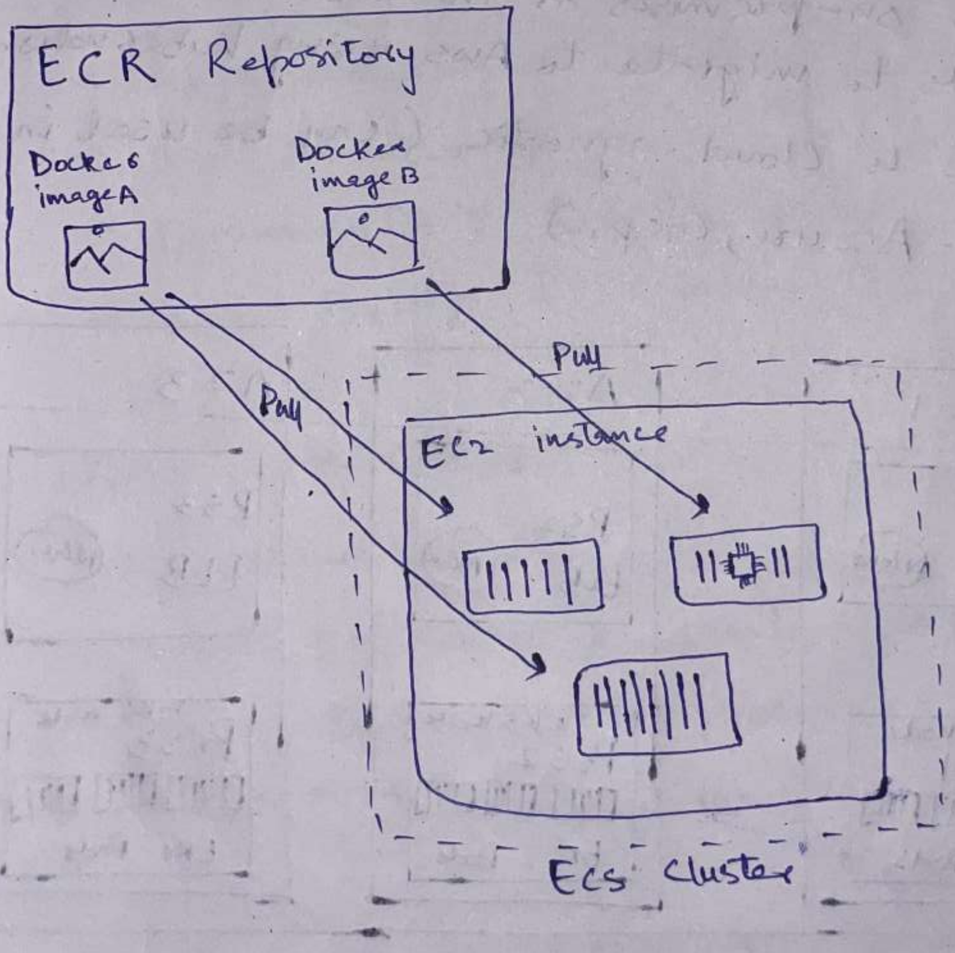
→ Store and manage Docker images on AWS

→ Private & Public repo. (Amazon ECR public Gallery)

→ Fully integrated with ECS, backed by Amazon S3

→ Access is Controlled through IAM (permission errors ⇒ policy)

→ Supports image vulnerability scanning, versioning, image tags, image life cycle...



Amazon EKS overview (K8) AWS

→ Amazon EKS = Amazon Elastic Kubernetes Service

→ It is a way to launch managed Kubernetes cluster on AWS

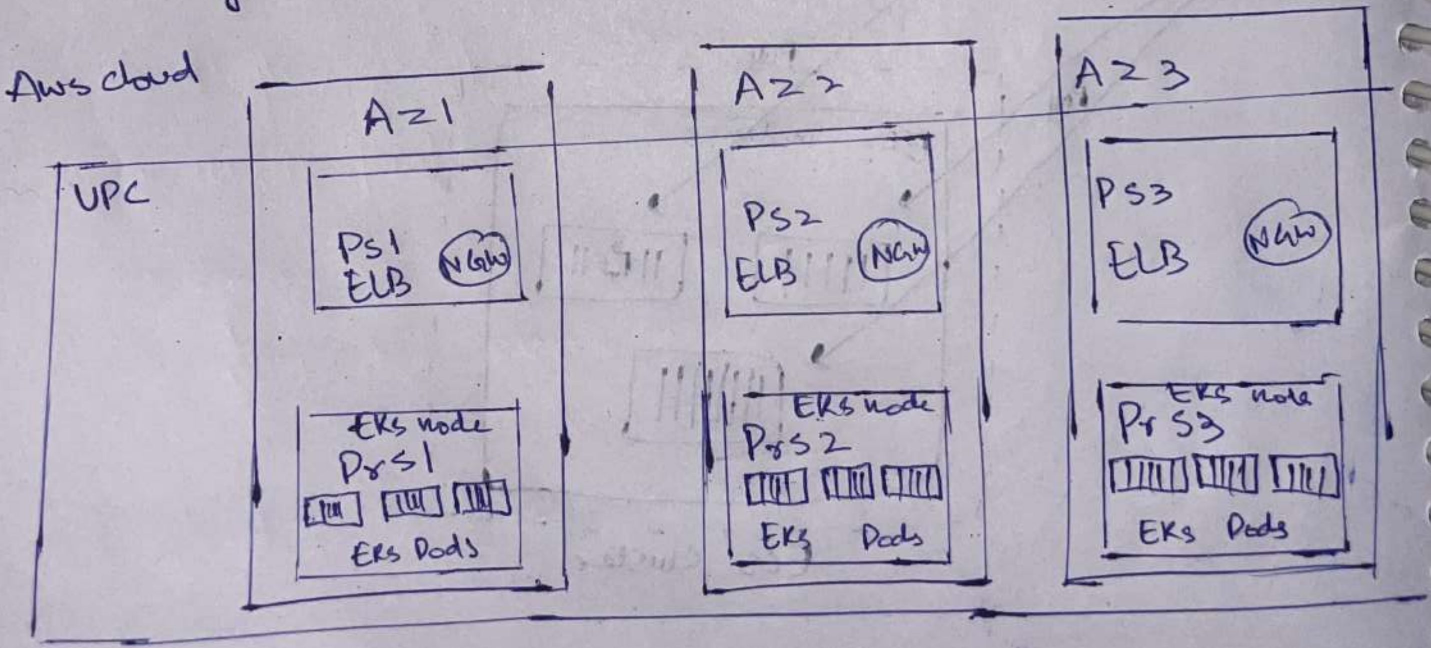
→ Kubernetes is an open-source system for automatic deployment, scaling and management of containerized (usually) Docker) application

→ It's an alternative to ECS, similar goal but different API.

→ EKS supports EC2 if you want to deploy worker nodes or Fargate to deploy serverless containers.

→ Use Case: if your company is already using Kubernetes on-premises or in another cloud, and wants to migrate to AWS using Kubernetes.

→ Kubernetes is cloud-agnostic (can be used in any cloud - Azure, GCP...)



Section -19

LAMBDA →

It is a serverless Compute service that lets you run code without managing servers.

→ Run Code only when ~~A~~ triggerd (event, API calls, Stream, Schedules)

→ No server provisioning or maintenance.

→ Automaticaly scales with traffic.

→ You pay only for Execution time.

→ Supports multiple languages (Py. Nodejs, Java, Go etc).

Common triggers:

→ API Gateway (Rest, HTTP APIs)

→ S3 (file uploads)

→ Dynamono DB & Kinesis streams.

→ Event Bridge (Schaduled jobs)

Typical use Cases:

→ Backend API's

→ Data processing & transformations.

→ Real-time stream processing.

→ Automation & Cron jobs.

Write code → Upload → Lambda fun.

DynamoDB:

It is a fully managed NO SQL value and document databases on AWS

↳ Serverless (no servers to manage)
- → Single -digit millisecond latency at any scale.
- → Automatically scales up or down.
- → Highly durable and available (multi-AZ)
- → Supports Key-Value and JSON document data models.

↳ Tables with primary-key
- → On demand or provisional Capacity.
- → Optional indexes (GSI / LSI) for flexible queries.

Common use Cases:
- → High -traffic web & mobile apps
- → User profiles and sessions storage.
- → Gaming leaderboards.
- → IoT and event data.

⇒ fast, scalable, serverless NO SQL database