[Developer productivity] Nov 29

5. Free Code Camp. dg: Youtube

00:20:25 **Elastic Beanstalk** → PaaS → choose a platform, upload your code and it runs with little knowledge of the infrastructure → Not Recommended for Production applications → powered by CloudFormation → can run Docker in environments

need to choose environment → Web / Worker → ASG / SQS / SQSd on EC2 → cloudwatch Alarm to dynamic scale up based on health

ASG + ELB
auto scaling group

1- load balanced Web env

→ use ASG and set to scale
→ use ELB
→ Design to Scale

2- Single Instance Web Env

→ still use an ASG but to keep desired capacity set to 1 to ensure server is always up & running

× No ELB as 1 server
→ public IP has to be used to route traffic to the server

Policy	Load Blanced Env	Single Env
all at once	✓	✓
Rolling	✓	✗
Rolling with additional batch	✓	✗
Immutable	✓	✓

comes with deployment Rules

- **all at once** :- 1) take all servers down
  2) deploy new app on all server at a time
  3) bring back the servers up

  ✓ fast × dangerous because of unavailable server and in case of major failures

- **Rolling** :- 1) take a batch bring down, deploy and up again
  2) move to next batch until all servers are done

  × slow ✓ safer than all at once, but failure needs rollback
  m= batch is reduced in capacity at any giving time during deploy

- **Rolling with additional batch** : 1) spin up new server batch
  deploy the new app, if good
  take down a batch of old servers
  2) repeat for all batches

  × slow ✓ safer than rolling, no rollback needed
  never reducing the capacity of our servers

• immutable : 1) new ASG with EC2 instances, deploy, point ELB to new ASG and delete the old ASG which will terminate its EC2's.

× slow ✓ safest way to deploy

deployment methods 10:35:19

method	impact of failed deployment	deployment time	No downtime	No DNS change	Roll backpoint	code deployed to instance
all at once	down time	⊕	✗ downtime	✓	manual	existing
Rolling	single batch out of service; any successful batches before failure running new app version	⊕⊕ may vary	No ✓ downtime	✓	manual	existing
Rolling with additional batch	minimal if first batch fails; otherwise similar to rolling	⊕⊕⊕ may vary	No ✓ downtime	✓	manual	New & Existing
Immutable	Minimal	⊕⊕⊕⊕	✓	✓	Terminate New	New
Blue/Green	Minimal	⊕⊕⊕⊕	✓	Yes DNS change ✗	Swap URL	New

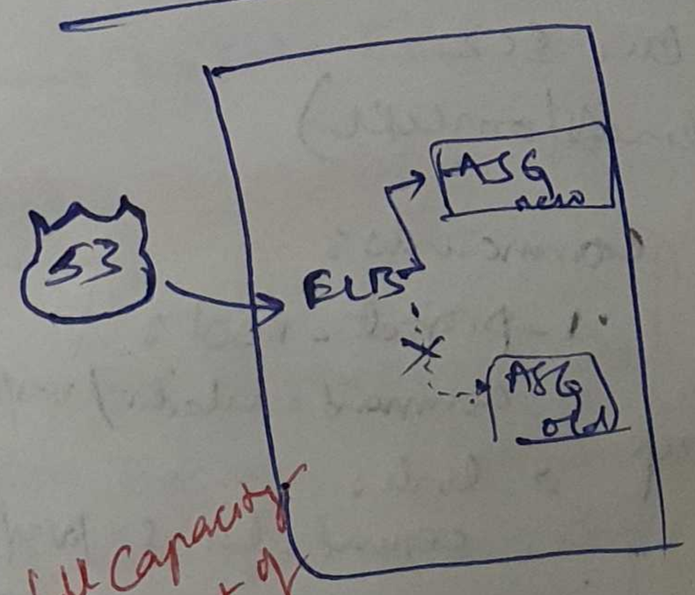
EBS by default performs "in place" updates.

- all at once
- Rolling
- Rolling with additional
- Immutable

* in place : means within the scope of EBS environment - i.e

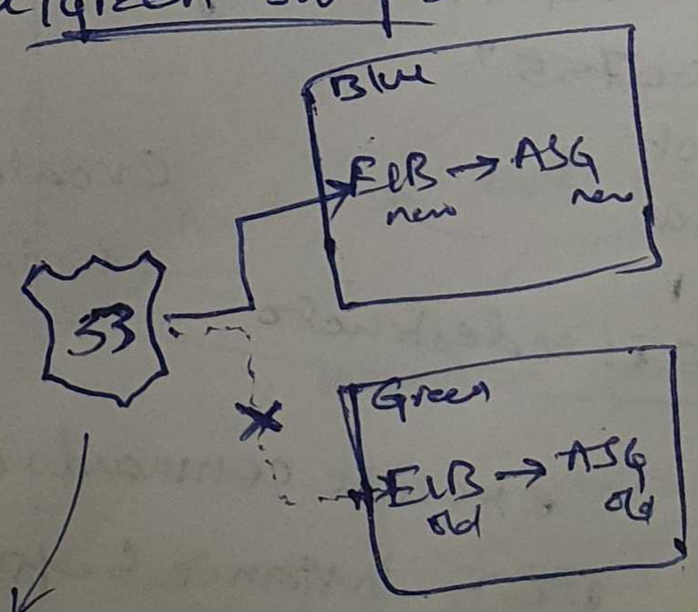
could
- mean with in the scope of same server (not replacing) → (all at once, Rolling)
- mean with in the scope of uninterrupted server (0 downtime) → EB can't do this [Capistrano + Ruby on Rails] [Chef automate this]

immutable (in place)



[maintain full capacity and minimal impact of failed deployment]

Blue/Green Swap Swu URL



EBS defines Blue/Green as swapping EB environments and this occurs at the DNS level swapping out the ELBs

Can give unavailable server due to the swapping

Configuration Files:

EBS env can be customized using config files

- .ebextensions : hidden folder at root of your project which contain config files
- .config is the extension for the config files which need to be store in .ebextensions

These files can → option settings / Linux / Win Server Config / Custom Resources

00:45:25

Env.yml environment manifest, which is stored @ root of your project

```
AWSConfigurationTemplateVersion: 1.1.0.0
EnvironmentName: exapro-prod +
SolutionStack: Ruby
EnvironmentLinks:
    "WORKERQUEUE": "worker +"
OptionSettings:
    aws:elb:loadbalancer:
        CrossZone: true
```

groups are denoted by +

00:47:00

Server Configuration [Linux]

① download packages & install apps & components

```
packages:
  yum:
    libmemcached: []
    ruby-devel: []
```

```
groups:
  groupAdmin: {}
  groupDev:
    gid: "12"
```

creating linux/unix group and to assign group IDs

```
users:
  andrew:
    groups:
      - groupAdmin
    uid: 87
    homeDir: "/var/www"
```

```
files:
  "/home/ec2-user/application.yml":
    mode: "000755"
    owner: root
    group: root
    content: |
      SECRET: 00destructo
```

create files on the EC2 instance (inline/from URL)

execute commands on the EC2 instance before app is setup

```
commands:
  1_project_root:
    command: mkdir /var/app
  2_link:
    command: ln -s /var/app /app
```

```
services:
  sysvinit:
    nginx:
      enabled: true
      ensureRunning: true
```

define which services should be started / stopped when the instance is launched

execute commands that affect your application's src code
run after downloading the src code
Not related to docker commands

Container_commands:
0_collectstatic:
command: "django-admin.py"
1_migrate:
command: "migrate.py
leader_only: true

EB CLI [00:49:20]

install CLI from github

- eb init → configure your project directory
- eb create → create your first env — single to avoid ELB (single instance)
- eb status → see the current status of your env
- eb health → instance/overall health (updates every 10 sec)
- eb events → list of events
- eb logs → pulls logs
- eb open → opens your env website in a browser
- eb deploy → once the env is running, deploy an update
- eb config → take a look at the config options available for running your env
- eb terminate → delete the env

EB Custom Image

: You can specify an AMI to use (amazon Machine Image) instead of standard EB AMI

improve provisioning times
useful if you need to install a lot of files

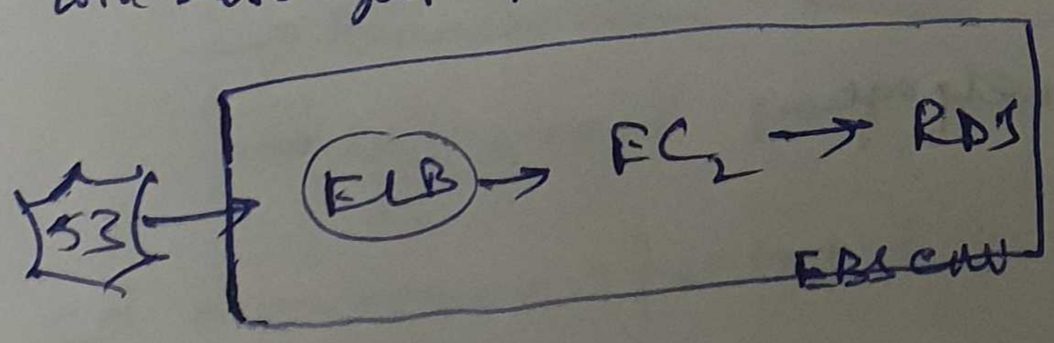
AWS Docs 1→ CLI 2→ describe-platform-version
aws elasticbeanstalk ↓ --region eu-east-1
--platform ...
3↓
ImageId 4→ launch a EC2 5→ Sections 6→ New AMI Manager
7 → update ENV

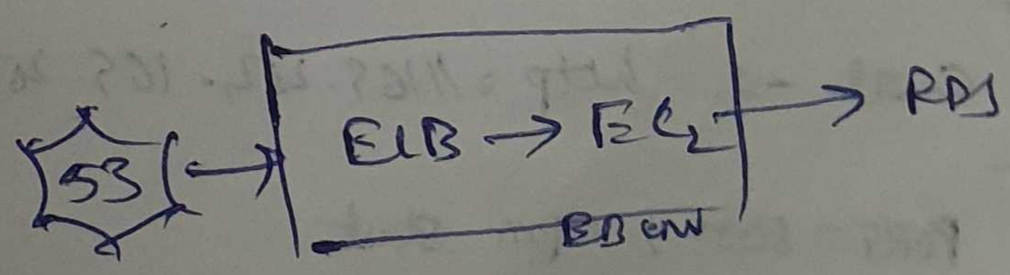
Configuring RDS

: can add a RDS inside & outside EB env

inside → intended for generally dev envs. when EB env is terminated the RDS will also get terminated

outside → intended for prod env





EB cheat sheet: [00:55:12]

EB demo: [00:59:23]

$ npm i cg -g creates webpage
$ cg README.md
$ mkdir ~/environment/study-sync
$ cd study-sync/
$ npm init -y → creates a package.json
$ npm i express --save
$ touch main.js index.js index.html app.js style.css

---

metadata service
$ curl -s http://169.254.169.254/latest/meta-data
↳ standard for EC2

$ _____ /mac (macaddress)
$ _____ 4 _____ /network
$ _____ 4 _____ /network/interface/mac/12:68:de.....
$ _____ 4 _____ /security-group-ids result

$ aws ec2 authorize-security-group-ingress --group-id
--port 8080 --protocol tcp, <ipaddress>/32
--cidr tell only a single ip address

- For a given ec2 instance of this cloud9 it opens inbound port 8080 with TCP under this security group

$ aws ec2 describe-security-groups --group-ids <secu gr>
--output text --filters Name=ip-permission.to-port,
Values 8080

to check the inbound rule on port 8080

$ curl -s http://169.254.169.254/latest/meta-data/public-ipv4 → get ipaddress of cloud9 EC2

$ PORT=8080; npm start
$ http://[illegible]:8080 → opens your webpage

Setup git [01:18:00]

```
$ cd ~/environment/study-sync/
$ touch .gitignore
$ ls -la
$ cg .gitignore ——→ in the editor node_modules → asking to ignore this folder
$ git init
$ ls -la → see .git created due to above command
$ git status
$ git add .
$ git status
$ git commit -m "initial commit"
```

For github and install EB cli

```
$ cd..
$ git clone https://github.com/aws/aws-elastic-beanstalk-cli-setup.git
$ ./aws-elastic-beanstalk-cli-setup/scripts/bundled_installer
$ eb → did find, then add to path
$ echo 'export PATH="/home/ec2-user/.ebcli-virtual-env/executables:$PATH"'
    >> ~/.bash_profile && source ~/.bash_profile
$ echo 'export PATH=/home/ec2-user/.pyenv/versions/3.7.2/bin:$PATH'
    >> /home/ec2-user/.bash_profile && source /home/ec2-user/.bash_profile
$ eb → now eb gives you some info
$ cd ~/environment
```

EB init [01:26:00]

```
$ cd ~/environment/study-sync/
$ eb init
    → give region as eu-east-1
    → new application
    → code commit [Y]
    → Repo name : study-sync
    → branch name : master
    → do you need ssh : [N]
$ cd ~/environment/.elasticbeanstalk/
$ cg config.yml
```

EB Config :

```
$ cd ~/environment/study-sync/
$ mkdir .ebextensions
$ cd .ebextensions
$ touch 001_envar.config
$ touch 002_node_command.config
$ cd ..
$ git add .
$ git commit -m "config for EBS"
$ git push
```

→ pushing to code commit is ing as if these files are not ther it will not allow to create environment

```
option_settings:
  aws:elasticbeanstalk:application:environment
    PORT : 8081
    NODE_ENV : production
```

→ how to start our app

```
option_settings:
  aws:elasticbeanstalk:container:node
    NodeCommand: "npm start"
    NodeVersion : 10.15.1
```

EB Create : New environment

```
$ eb create --single
  enter env name
  enter dns prefix
  would you like to enable spot fleet requests for this env -n
```

→ eb environment in single mode
if you don't it will spin up ELB (can cost money)

```
$ eb status  ---> launching .....> Ready
$ eb logs
$ eb events
```

} just to check how they look like

EBS Immutable : [01:42:12]

```
$ cd ~/environments/study-sync/.ebextensions/
$ touch 000_deploy.config
$ cd ..
$ git add .
$ git commit -m "immutable deploy"
$ git push
```

```
option_settings:
  aws:elasticbeanstalk:command:
    DeploymentPolicy : Immutable
    HealthCheckSuccessThreshold : Warning
    IgnoreHealthCheck : true
    Timeout : "600"
```

```
$ eb deploy
```

→ if any errors give $ eb abort, fix it and then gitadd & commit & push and eb deploy again

EBS ~~blue green~~ deploy:

$ eb clone
enter the clone env name
enter the DNS cname prefix
→ make small change in your web page
git push ib

$ eb deploy study-sync-prod-clone → now the clone will have the new version of the web page. Remember clone & normal env are running different urls

$ eb swap study-sync-prod
--destination-name study-sync-prod-clone

$ eb terminate study-sync-prod → terminating the old version environment

---

EBS single-Container Dockerfile

$ curl http://169.254.169.254/latest/meta-data/public-ipv4 o/p

$ op: 8080
No need of node config as it will be part of docker

$ rm .ebextensions/002_node_command.config
$ git add . && git commit -m "docker" && git push

$ eb create --single
enter env name - study-sym-docker

EBS Single Container with ECR → elastic Container Registry

in above example when placed the dockerfile the EBS did all the work for us, build the image, but we need to actually build the image ourselves and push to docker repository i.e ECR in aws. here we will do this.

→ change version/small text in your website
$ docker build -t study-sync . → building docker image
$ aws ecr get-login-password | docker login --username AWS --password-stdin <accnumber>.dkr.ecr.us-east-1.amazon.com → this basically creates a /home/ec2-user/.docker/config.json file with your password, this will help you to authenticate
$ docker images → take imageid
$ docker tag <imageid> <acc>./study-sync
$ docker push → go to ECR and create a repo study-sym

$ cd ..

$ mkdir study-sync-external

$ cd study-sync-external

$ ~~touch~~ Dockerrun.aws.json

$ cq

$ git init

$ touch .gitignore

$ mkdir .ebextensions

$ touch .ebextensions/001_envvar.config

$ git add . && git commit -m "Dockerrun" && git push

```
{
"AWSEBDockerrunVersion": "1",
"Image": {
    "Name": "<acc>.dkr.ecr.../study-sync
    },
"Ports": [{
    "ContainerPort": 8080,
    "HostPort": 8080
    }]
}
```

copy content from the previous examples

$ eb init → 1, 2, y, 2, study-syn-external, master, n

$ eb create --single → study-syn-external, n

- the deployment of EBS is taking very long time due to resolving dependencies on all of your 100 target EC2 instance ⇒

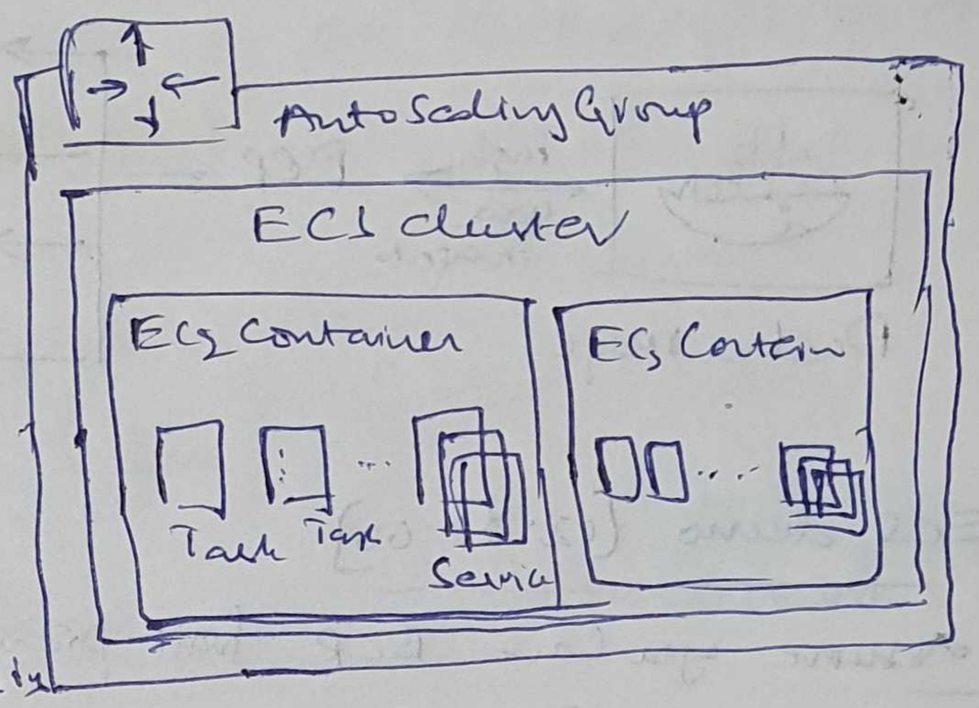
  Bundle the dependencies in the source code during the build step of Code Build

Elastic Container Service: [02:17:12]

Nov 29

Introduction to ECS: - fully managed, container orchestration service, highly secure, reliable and scalable way to run containers

Components in ECS:-



Cluster: multipe EC_2 instances which will house the docker containers

Task Definition: json file that defins the configuration of (upto 10) containers you want to run

Task: launches containers defined in Task Definition. Tasks do not remain running once workload is complete

Service: like task but longrunning for web-apps.

Container Agent:- Binary on each EC_2 instance which monitors, starts and stops tasks

Creating an ECS cluster

EC_2 Linux + Networking

Resources to be created:
(1) Cluster
(2) VPC
(3) Subnets
(4) ASG with Linux AMI

Create cluster: Decide 1) ECS | Fargate
2) Spot (or) On Demand
3) # instance
4) EBS Storage Volume
5) EC_2 → Amazon linux 1/2
6) existing VPC | new
7) Assign an IAM Role
8) Option to turn on Cloudwatch Container Insights
9) * Option to choose keypair → SSH not recommended for ECS

Task def ex:

Create New Task Definition

- Can contain multiple containers in 1 task def
- Docker Images can be provided either by ECR (or) official docker repo like Docker Hub
- must have 1 essential container

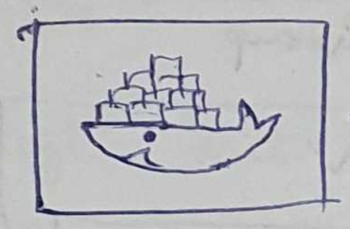
```
{
"containerDefinitions": [
  { "name": "wordpress",
  "links": ["mysql"],
  "image": "wordpress",
  "essential": true,
  "portMappings": [
    { . . . .
      . . . .
    }

  "essential": true

}
```

ECR: Elastic Container Registry

A fully managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images



Docker Image → Push your images → ECR → ECS / Fargate / EKS / On Premise

ECS demo [02:22:00]

Assume you have ECR from previous chapter

Create ecsInstanceRole

{IAM > For Service EC2 > Search (EC2 Container Service for EC2 Role).
> Next > Next > Name: > Create

ECS > Cluster > Create > EC2 Linux + Networking > ClusterName: MyECSCluster >
On Demand > t2.micro > #instance 1 > Create

Task Definition > Create New > EC2 > Task Name: StudyFunc > ~~Task Role~~:
Task Memory: 500 > Task CPU: 1 ~~vcpu~~ vcpu > Container Name: StudyFunc Contain
Image: <Copy from ECR's StudyFunc> Memory Limits: 256 >
port mapping: host 80 — container 8080 > ADD > Create

Environment variables

key	value
PORT	8080
NODE_ENV	production

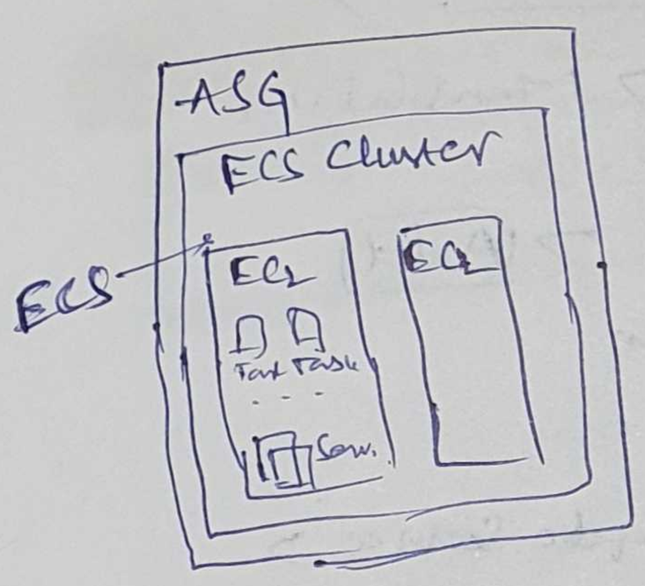
IAM > For ~~service~~ usecase: Elastic Container
Service Task > Search: ~~Amazon~~ ECSTask
Execution Role Policy > Next > Next >
Name: ecsTaskExecutionRole > Create

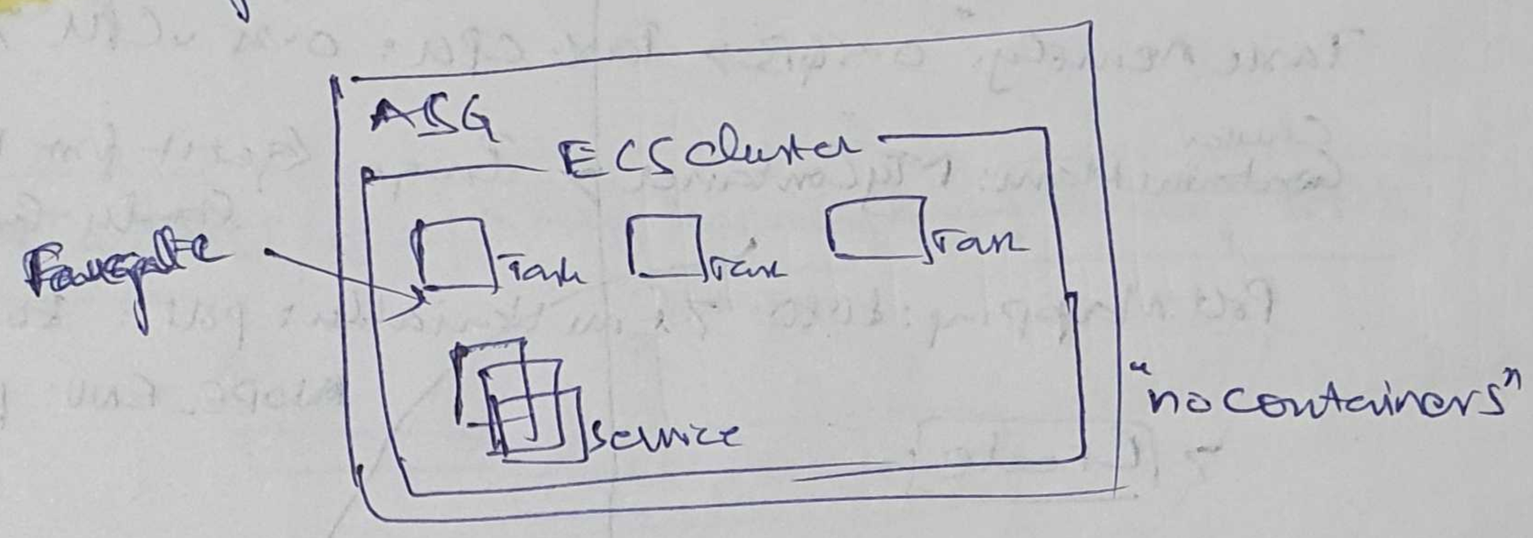
~~Create New Revision >~~

ECS > Cluster > Service > Create > launch type: EC2 > Task def: > — >
cluster: MyECSCluster > ServiceName: MyService > ServiceType: Replica >
task: 1 > Next > Load balancer: None > Next > Create

Fargate [02:37:22]

- Serverless containers. Runs containers, pay by duration & consumption.
- You can create an "empty" ECS cluster, no EC_2's provisioned. and then launch Tasks as "Fargate"
- No need to [provision | configure | scale] clusters of EC_2 instances to run containers
- Charged for atleast 1 min, after that per second





Configure Fargate Tasks

- In your Fargate Task definition you define the "memory" & "vCPU"
- You will then add your containers and "allocate" the memory & vCPU to each
- When you run the task you choose VPC and Subnet
- "Apply security groups to a Task" → only Fargate
- Can apply an IAM role to the Task → This can be done to both ECS & Fargate

Fargate vs Lambda

	Fargate	Lambda
Cold Starts	Yes (Shorter)	Yes
Duration	As long as you want	15 mins (max)
Memory	Upto 30GB	Upto 3GB
Container	You provide your own container	Limited to standard containers
Integration	manual labor	Seamlessly integrates with other serverless services
Pricing	Pay atleast 1min after that per sec	Pay per 100ms

F

Fargate demo [02:42:50]

- ECS > Cluster > [New] > Networking only > Cluster name: MyFargate Cluster
  Powered by Fargate

  > [Create]

- Task def > Fargate > Name: Study-Sync-F > Task Role, ecsTaskExecutionRole >
  Network Mode: awsvpc > Task execution role: ecsTaskExecutionRole >
  Task memory: 0.5GB > Task CPU: 0.25 vCPU > [Add Container] >
  Cluster Container Name: MyContainer > Image: Get it from ECR > Memory limit: 128
  Study-Sync
  Port Mapping: 8080 > Env Variables: port: 8080 > [Add]
  NODE_ENV: production

  > [Create]

- Service > Create > Task def: > Cluster > Service: MyFargate Service >
  # task: 1 > [Next] > Cluster VPC: , Subnet, ELB: None > [Next] > [Create]

ECS & Fargate Cheat Sheet: [02:53:22]

AWS X-Ray: Helps developers analyze and debug applications utilizing "microservices" architecture

→ distribute tracing system / Performance Monitoring System

MicroServices: is an architectural/organizational approach to s/w development where s/w is composed of small independent services that communicate over well-defined API's.

isolated services

S3	Containers Tasks	λ	RDS	Notifications	Queuing	Streaming

Distributed Tracing: also called as distributed req tracing, is a method used to profile and monitor apps, especially those build using a microservices architecture. It helps pinpoint where failures occur and what causes poor performance.

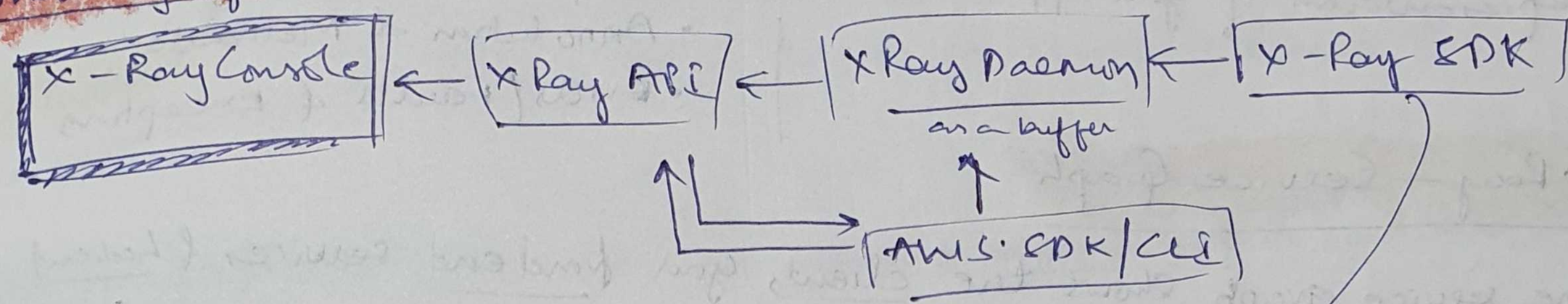
Performance Monitoring: monitoring & management of performance, availability of s/w apps. APM strives to detect & diagnose complex app performance problems to maintain an expected level of service

application Performance Monitoring

3rd party services to X-Ray :- DataDog | New Relic | Signal Fx | Lumigo

- Collects data about API requests that your app serves
- View, filter collected data to identify issues & avenues for optimization
- Also trace the call that your app makes to the downstream AWS resources microservices, dbs & HTTP web APIs.

Anatomy of X-Ray



provides ① Interceptors to add to your code to trace incoming HTTPS requests

② client handlers to instrument AWS SDK clients that your app uses to call other AWS services

③ HTTP Client to use to instrument calls to other internal & external HTTP web services

* XRay SDK supports instrumenting calls to SQL dbs, automate AWS SDK client instrumentation & other features

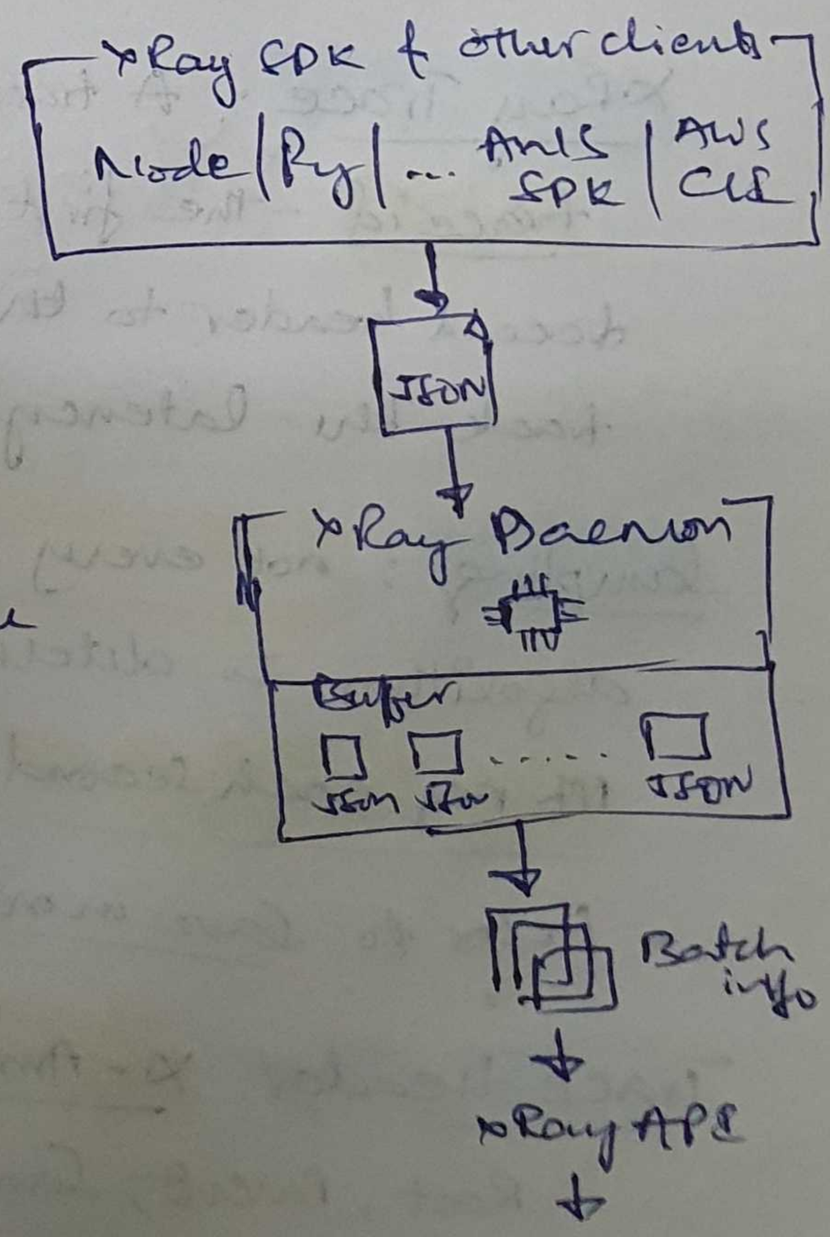
Instrumenting: The ability to monitor / measure the level of a product's performance, to diagnose errors & to write trace information.

X-Ray Daemon:

Instead of sending trace data directly to X Ray, the SDK sends JSON segment docs to a daemon process listening for UDP traffic.

The X Ray daemon buffers segments in a queue and uploads them to X Ray in batch

included in EBS, can install on Win/Lin/Mac & Lambda



X Ray Concepts

- AWS X Ray receives data from services as segments
- X Ray then groups segments that have a common requests into traces
- X Ray process the traces to generate a service graph that provides a visual representation of your app

- Segments
- Subsegments
- Service Graphs
- Traces
- Sampling
- Tracing Header
- Filter Expressions
- Groups
- Annotation & Metadata
- Errors / Faults & Exceptions

X Ray - Service Graph

The service graph shows the client, your frontend services & backend services
- It helps to indentify bottlenecks, latency spikes, & other issue

X Ray Segments:

- The compute resources running your app logic send data about their work as segments
- It contains - Hostname, the request info, the response, the work done

X Ray Sub Segment: - more granular timing information and details about down stream calls that your app made to fulfill the original req & calls
- it contains additional details about AWS service, HTTP API calls

X Ray Trace: A trace collects all the segments generated by single req

trace id - the first supported service the HTTP interacts with adds a traceid header to the request and propagates it downstream to track the latency, disposition and other request data

Sampling: not every request is traced. X Ray SDK uses a sampling algorithm to determin which request get traced. by default 1st req each second and 5% of any additional requests
- helps to save money. by reducing request trace count

Trace Header: X-Amzn-Trace-Id this is the header contains traceid Root, Parent, Sampled

Filter Expression: to narrow down to a path / users

Groups: You can assign a filter expression to a group, you call the group by name / by ARN to generate its own service graph, trace summaries and Amazon CloudWatch metrics.

Annotations & metadata: - aggregate at the trace level, can be added to segment & subsegment

- **Indexed** key pair value for use with filter exp → data that you want to use for grouping / filtering so / trace
- **non indexed** key pair value, the values can be any type → data you want to store in the trace, but not need to use for searching traces

Errors / Faults / Exceptions: → X Ray SDK records exception details and the stacktrace

- Client errors: 400 / 4xx
- Server faults: 500 / 5xx
- Throttle errors: 429 too many Requests

AWS Service Integrate

imp * lambda, ELB, SNS, SQS, API gateway, AppMesh, Cloud Trail, Cloudwatch, AWS Config, EBS, ECS, EC2, EKS Fargate

Supported Languages: Go, Java, NodeJS, Ruby, ~~Java~~ .NET, PHP, Py

XRay Cheat Sheet [03:22:04]

Nov 26

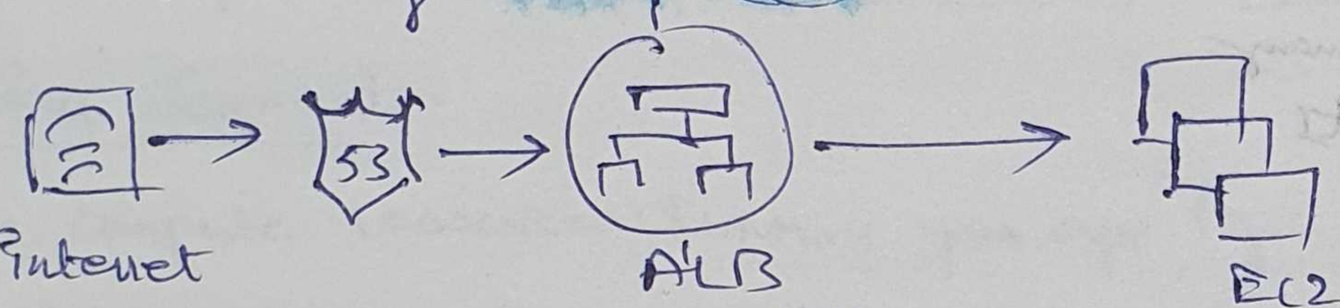
AWS Certified Manager ACM : [03:25:21]

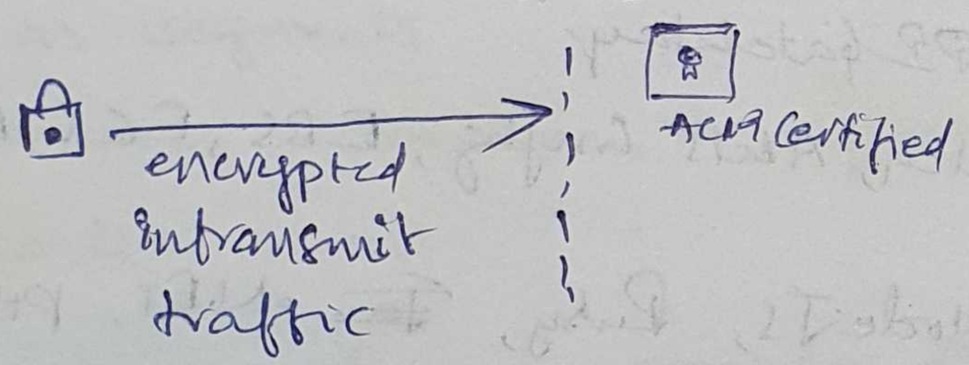
Provision, manage and deploy public and private SSL/TLS certificate for use with AWS services.

- AWS handles 2 kinds of certificates
  - Public – certificates provided by ACM (free)
  - Private – certificates you import ($400/month)
- ACM can handle multiple sub domains and wildcards domains (*)
- ACM can be attached to the following services
  - ELB
  - CloudFront
  - API Gateway
  - EBS (through ELB)

ACM example & SSL Termination

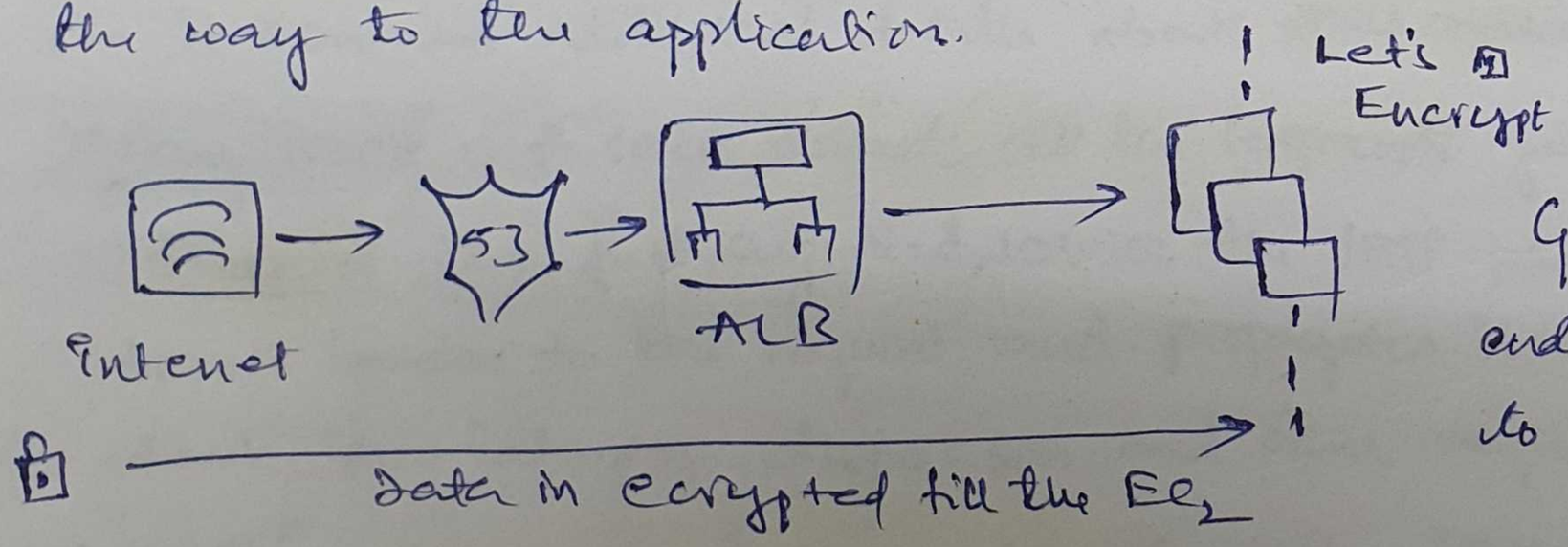
→ Terminating SSL @ALB (application load balancer) → all traffic in transit beyond ALB is unencrypted





↳ you can add as many EC2 instances to the ALB and you don't need to install certificates on each instance.

→ Terminating SSL End-to-End : Traffic is encrypted in-transit all the way to the application.



Guarantees encryption end to end. more complicated to maintain certificates.

Route 53 : highly available and scalable cloud domain name service (DNS)

Register and manage domains,

create DNS routing rules eg. failovers.

Similar to GoDaddy.

→ Register and manage domains

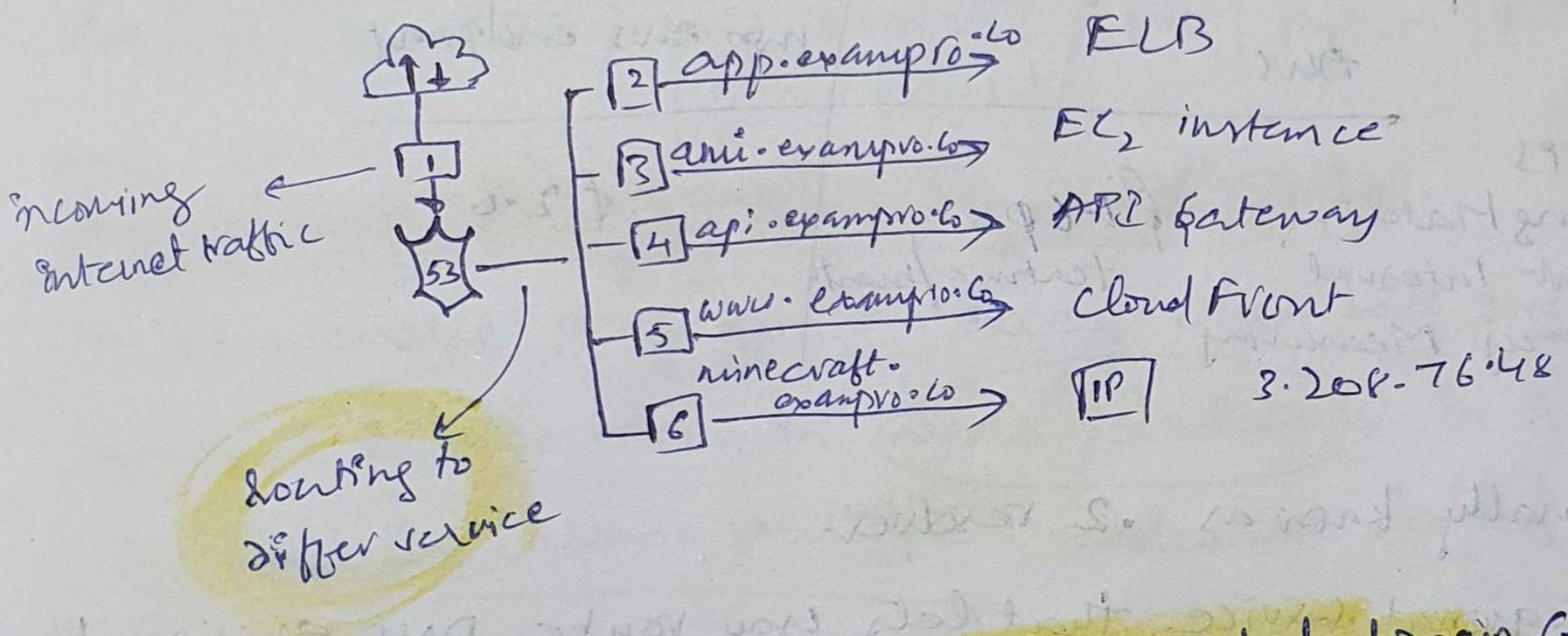
→ create various records sets on a domain

→ Implement complex traffic flows eg: Blue/green deploy, failovers

→ Continuously monitor records via health checks

→ Resolve VPC's outside of AWS

use case: use Route 53 to get your custom domains to point to your AWS Resources



record sets: these allows us to point our naked domain (exampro.co) and subdomains via domain records.

ex: subdomain (www.co) to specific ip of cloud front

alias: instead of ip you can use resource name here. even if ip changes it will point to the right resource

routing policies: 7 different types.

① Simple routing: (default) multiple address result in random selection

② Weighted routing: route traffic based on weighted values to split traffic

③ Latency-Based routing: to region resource with lowest latency

④ Failover Routing: if primary fails/unhealthy send to secondary endpoint

⑤ Geo location routing: based on location of your users

⑥ Geo proximity routing: resources & optionally shift traffic from 1 to other location resource

⑦ Multi-value Answer Routing: respond to DNS queries with upto 8 healthy records selected at random.
[Simple routing + Healthcheck]

Traffic flow: visual editor lets you create sophisticated routing configs for your resources using existing routing types

- supports versioning so you can roll out / roll back updates

Health checks: •) checks every 30s by default. Can be reduce to every 10s
•) it can initial a failover if status is returned unhealthy
•) A cloudwatch alarm can be created to alert you of status unhealthy
•) it can monitor other health checks to create a chain of reaction.

•) Can create upto 50 health checks for AWS endpoints that are within or linked to same aws account

Basic	$0.50 per healthcheck/month AWS		$0.75 non AWS endpoint
Optional features	HTTPS String Matching Fast Interval Latency Measuring	$1.0 per feature/month	$2.0

R53 - Resolver: •) Formally know as .2 resolver.
•) A regional service that lets you route DNS queries b/w your VPCs and your N/w
•) DNS resolution for Hybrid environments (Onpremise - Cloud)

Route53 Cheatsheet: [03:42:53]

AWS CLI : [03:45:00]

- Control multiple AWS services from command line and automate them through script
- Can interact with AWS from anywhere.
- important flags - profile and output (text/table/text)
  to switch b/w aws accounts.
- CLI is installed using a python script

AWS SDK : control multiple AWS services using popular programming languages

it a set of tools and libraries that you can use to create applications for a specific H/w package

in case of AWS SDK - it is a set of API libraries that interact with AWS services

C++ / Go / Java / JS / .NET / NodeJS / PHP / Python / Ruby

Programmatic Access, Access key and Secret :

in IAM enable programmatic Access. which will give you { Access KeyID, Secret access key }

you need to store them in ~/.aws/credentials in home directory

the cred file allows multiple credentials (called profiles)

```
[default]
aws_access_key_id = .....
aws_secret_access_key = .....
[enterprise-d]
[deep-space-9]
```

Cloud 9 & CLI Setup demo

Create new user with programmatic access :-

IAM > add user > username: spock > Access type : ☑ programmatic access >

Add user to group : Developer : poweruser access > take key & secret.

---

cloud 9 > environment > create > linux (t2.micro).

---

in cloud 9 terminal

$ aws ... if installed you will see help.

if not use pip3 install awscli --upgrade --user

$ npm i cg -g

$ cd ~/

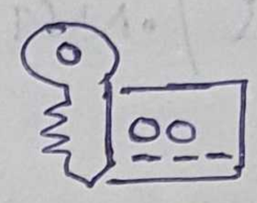
$ ls -la < $ mkdir .aws
$ cd .aws

$ vim credentials → [spock] (profile)
aws_access_key_id= ....
aws_secret_access_key= ....

$ aws s3 ls --profile spock

$ aws s3 cp <bucketname> /home/ec2-user/<objectname.ext>
any object URI

AWS Key Managment Service [KMS]:



- Creates and manages encryption keys, use with a variety of AWS services and in your apps. [checkbox secure ☑ and start encryption]
  when you see this most likely its KMS
- KMS makes it easy for you to create, control, and rotate encryption keys used to encrypt your data on AWS
- KMS is a multitenant hardware security module (HSM)
  - hardware device to store keys 'in' memory' not disk
  - it is very expensive so KMS is multitenant
- Can be used with Cloudtrail to audit access history
- KMS integrates with many services
  [RDS, CodeCommit, S3, CodeDeploy, Galacier, SNS, SQS, DynamoDB, EC, Xray, Elastic Cache, Codebuild .... and many
  Encryption of masterkey [aws/ebs] example.
- multiple customers are utilizing the same piece of hardware. customer are isolated from each other virtually. KMS also provided single tenent/ dedicate HSM

[KMS stores the CMK, and receives data from the clients, which it encrypts and sends back]

Cloud HSM is also a single tenant HSM which gives you full control.
↳ FIPS 140-2 level 3

while KMS is only FIPS 140-2 level 2

KMS - Customer Master Key (CMK)

- cryptographic keys / data keys :- A string of data that is used to lock & unlock cryptographic functions, including authentication / authorization & encryption.
- Master key : These are used to encrypt all other data keys. These are stored in secure hardware HSM
- Envelope encryption : A key used to encrypt another key.

- are the primary resources in AWS KMS. These are logical representation of master key
- CMK includes metadata such as { KeyID, creation date, description, key state
- CMK also contains the key material used to — encrypt, decrypt

AWS KMS supports both symmetric & Asymmetric CMKs

A 256-bit key that is used for encryption & decryption
1 key only
ex: Encrypting S3 bucket with AES-256

an RSA key pair - used for encryption & decryption (or) verification [not both] (technically more secure)
2 keys (public & private)
EC2 key pairs used to SSH into a server

KMS - AWS CLI :

- aws kms create-key → creates CMK in your acc & region
- aws kms encrypt → plaintext to ciphertext using CMK
- aws kms decrypt → decrypts cipher encrypted by CMK
- aws kms reencrypt → decrypts & then re-encrypts (manually rotate a CMK) a secure
- aws kms enable-key-rotation - automatic rotation in same AWS acc symmetric customer master key

AWS Cognito :- Decentralized managed Authentication : Signup/in. integration for your apps. Social identity provider eg: FB/Google

Cognito User Pools : user directory with auth to IdP to grant access to your app

Cognito Identity Pools : Provides temp credentials for user to access AWS services

Cognito Sync : Syncs user data and preference across all devices.

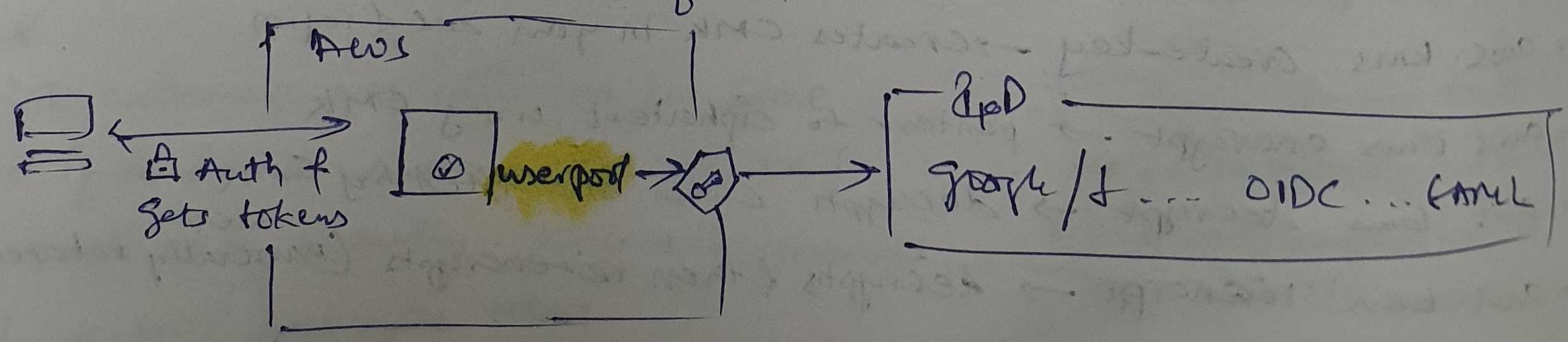
Web Identity Federation (WIF) :- To exchange identity & security information between an identity provider (IdP) and an application

Identity Provider (IdP) :- A trusted provider of your user identity that lets you use authenticate to access other services. ex: google/FB/twitter...

two technologies
- SAML : Security Assertion Markup Language
  - ↳ SSO : Single Signon
- Open ID : OpenID Connect (OIDC)
  - ↳ OAuth - This is for Web Identity Fed

Cognito User Pools are user directories used to manage the actions for web/mobile apps
- Signup/in
- Acc recovery/confirmation.

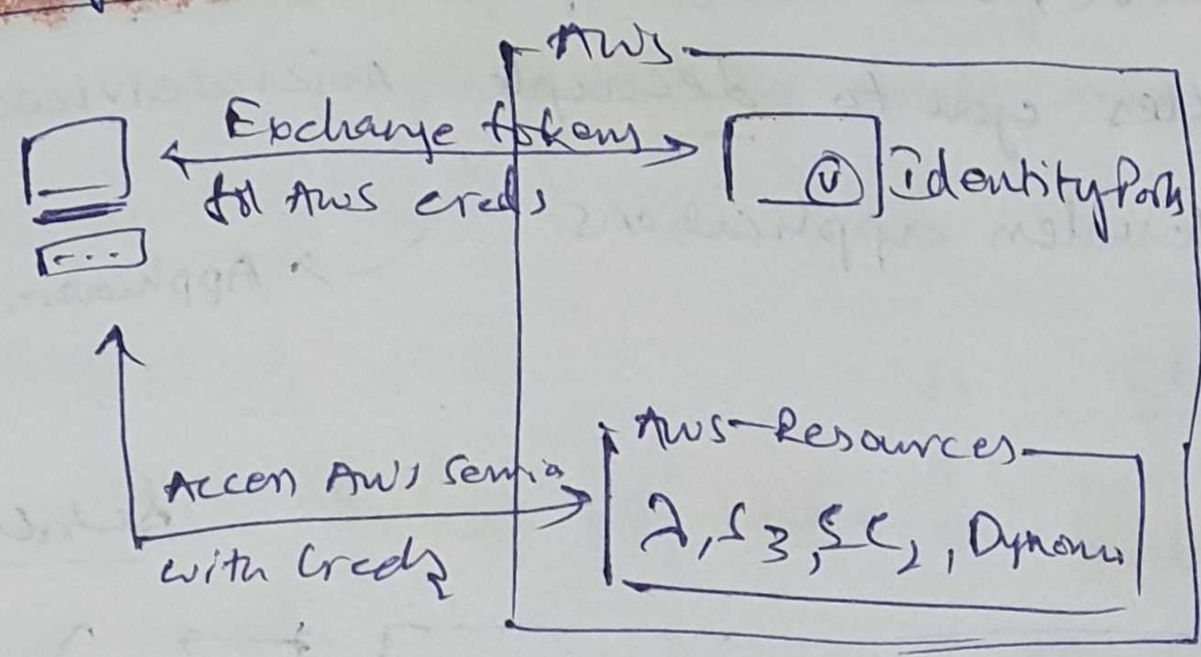
- Allows users to signin to userpool / using WIF
- Cognito acts as broker b/w IdP and AWS.
- Successful user auth generates a JSON Web Token (JWTs)
- User pools can be thought of as the account used to access the system



options { username/email/phone → verification
pswd — length

{ Set DOB in signup
MFA
integrate with user campaign

Identity Pools:



→ First choose the provider — {can be Cognito | FB | Google ... | OpenID | SAML
(or) unauthenticated}

get the creds and use in your SDK.

Cognito Sync: Syncs userdata and preferences across devices with 1 line of code

↳ it uses push synchronization to push updates & sync data.

- uses SNS to send notifications to all devices when data in the cloud changes

Cognito demo: [4:20:06]

Cognito cheat sheet [4:25:00]

Simple Notification Service [SNS]

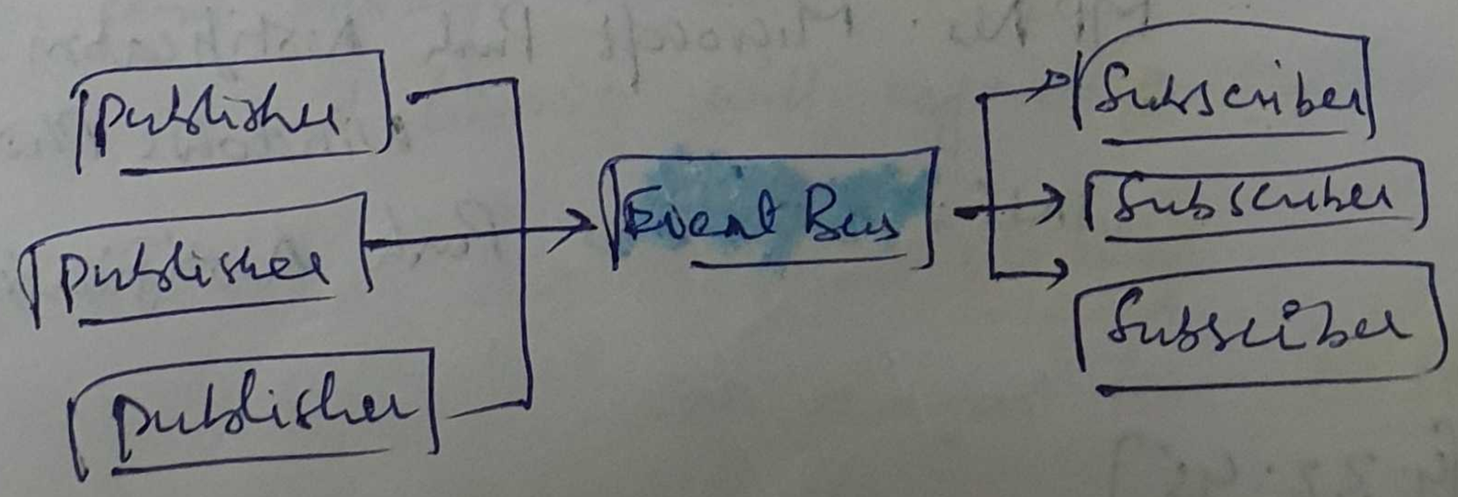
: subscribe and send notification via txt, email, webhooks, λ, SQS, mobile notifications

Pub/Sub :- pattern commonly implemented in messaging system.

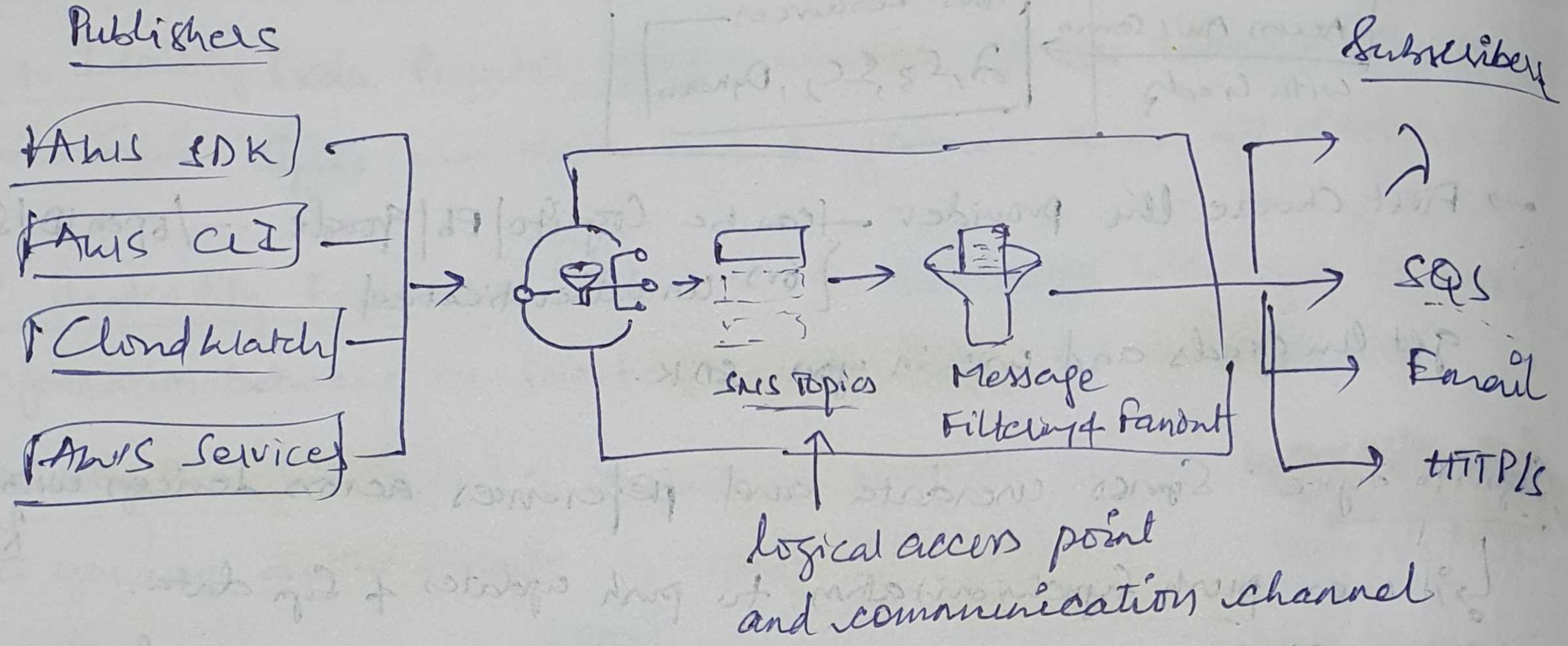
- Sender of message publishes, don't send directly to receivers, sends to Event Bus
- Event bus categorizes their msg into groups. Then sends to subscribers of these groups. msg are immediately delivered.

→ publisher has no idea of subscribers

→ subs don't publish for msgs → msg & events are interchangable in Pub/Sub



SNS :- highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems and serverless applications. → Application Integration



SNS Topic:
- topics allow you to group multiple subscriptions together
- 1 topic can send multiple protocol at once (email, txt...)
- You can encrypt topic via (KMS)

SNS subscriptions:- A subscription can only subscribe to 1 protocol & topic

- HTTP, HTTPS, Email (only plain txt), Email-JSON, Amazon SQS, AWS λ, SMS
- platform application endpoints

platform application endpoints: sends push notification msg directly to apps on mobile devices. → ADM: Amazon Device Messaging
(APNs): Apple Push Notifications Service
Baidu: Baidu Cloud Push
FCM: Firebase Cloud Messaging
MPNs: Microsoft Push Notification Service for Windows Phones
WNS: Windows Push Notification

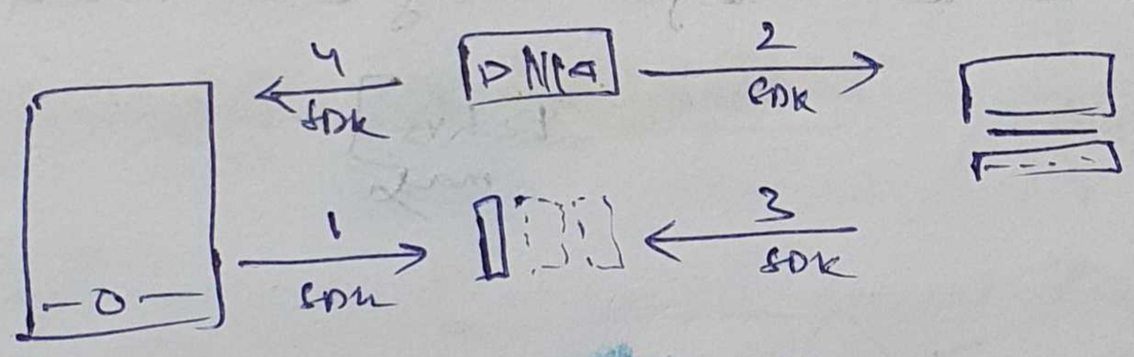
SNS Cheat Sheet: [4:32:45]

Simple Queue Service (SQS) : Fully managed queuing service that enables you to decouple and scale microservices, distributed systems & serverless applications.

Queueing	vs	Streaming
• Generally will delete msg once they are consumed.		• multiple consumers can react to events (msg).
• simple communication		• Real time
• Not Real-time, have to pull		• Event live in the stream for long period of time, so complex operations can be applied
• Not reactive		
SQS / Sidekiq / RabbitMQ		Kineris / Kafka / NATS.

- SQS is for Application Integration
- SQS is pull based NOT push based. { using AWS SDK, write code which publishes msg onto the queue / you pull the queue for msg.

usecase



① mobile send the msg to queue
② Computer pulls the msg periodically
③ Computer sends back task completed and marks msg completion [SQS deletes it from the queue]
④ mobile app pulls the queue and sees the msg is no longer in the queue so consumed.

SQS limit :

- message size : 1 byte to 256 KB
- if more is needed use Amazon SQS extended Client library for JAVA lets you send message 256 KB to 2GB in size. The message will be stored in S3 and library will reference the S3 obj.
- message retention → default : 4 days
  can adjust from 60 sec — 14 days

Delete Queue: Deletes queues, takes 60 sec

Purge Queue: Deletes msg in the queue, But Queue remain same.

SQS Queue Types

1 → Standard
- nearly unlimited
- guarantees delivery atleast 1
- provides best effort in ordering. but can't guarantee
- more than 1 copy of msg could potentially delivered out of order

2. → FIFO Queues:
- limited to 300 transactions/second
- Guarentees the order of the message

SQS Visibility TimeOut :-

this is the period of the time that messages are invisible in the SQS queue, after a reader picks up that msg

- messages will be deleted from the queue after a job picks and process [before the visibility timeout expires]
- If a job is not processed before the visibility time out. the message will become visible again, another reader will process it.
- This can result in the same message being delivered twice!!

[0sec min, 30sec default, 12hrs max]

Short vs Long Polling:

→ method in which we retrieve msg from the queue

{ default
- Returns msg immediately even if the msg queue is polled empty

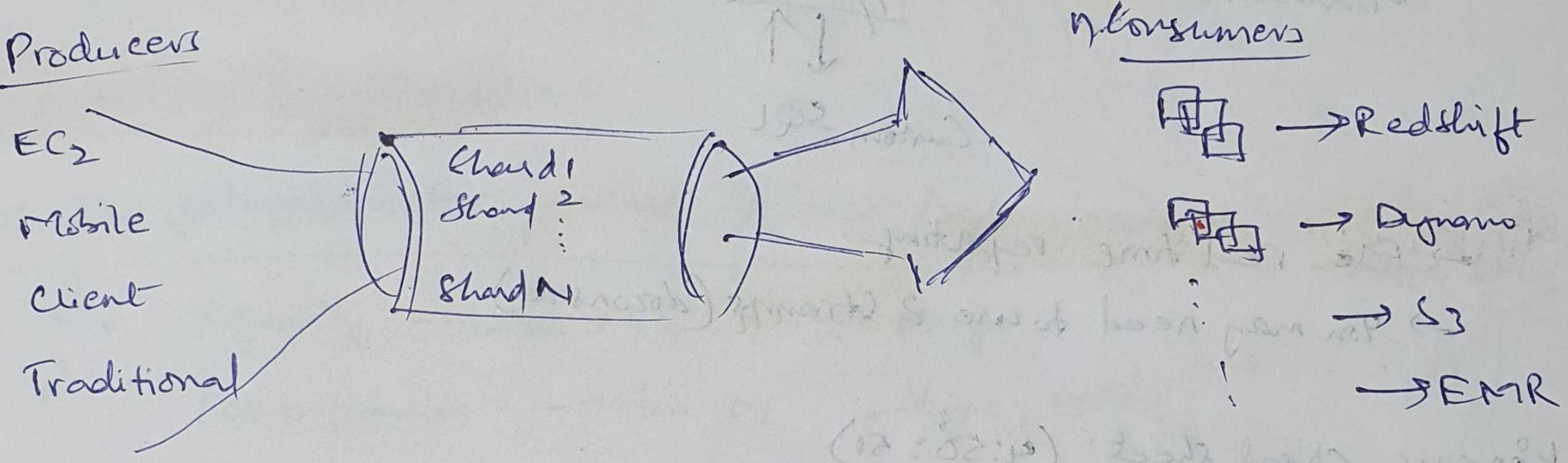
{ waits until the msg arrives in the queue / the long poll timeout expires.
- inexpensive to retrieve msg if available in queue.
- cost effective because no empty receives
- Most use case needs/uses Long Polling

Amazon Kinesis:

is AWS fully managed solution for collecting, processing and analyzing streaming data in the cloud. Real time. It has 4 diff types { Data Stream, Firehouse, Data Analytics, Video Analytics

Can do with SDK but is inefficient need to use KPL for write data to stream
KPL → kinesis producer library

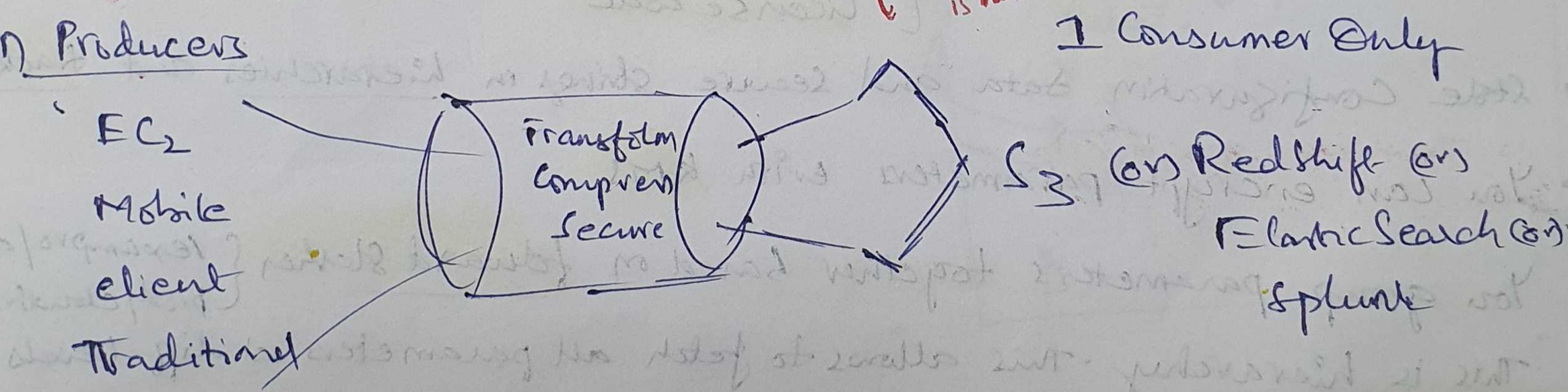
Kinesis Data Streams:



- → You pay per running shard
- → You can have multiple consumers, you must manually configure your consumers
- → Data persist for 24hrs (default) to 168 hrs (7 days) before it disappears from the stream

Kinesis Firehose Delivery Stream

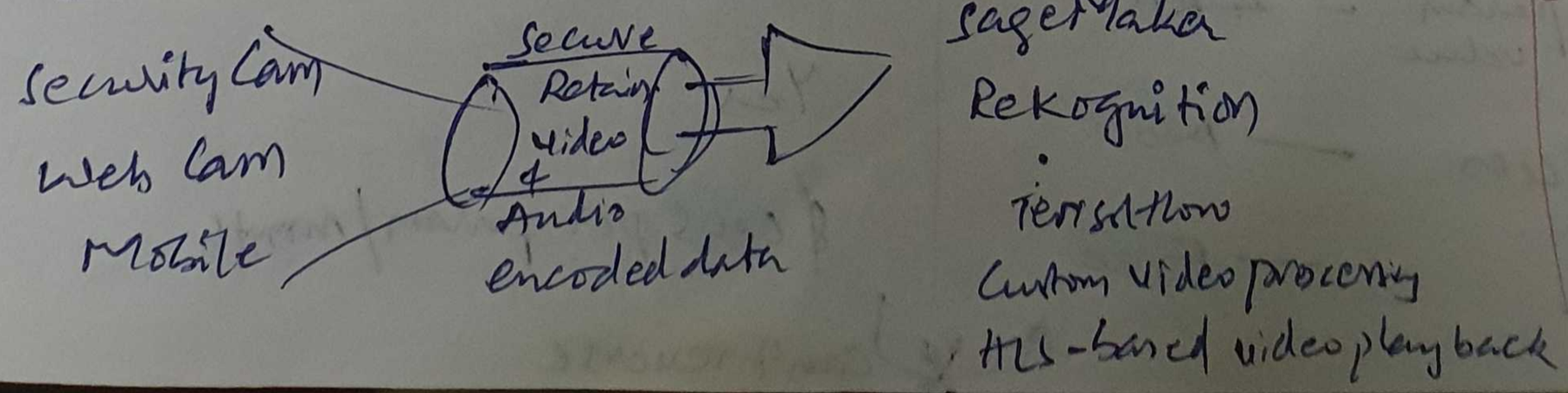
[Amazon Elasticache with S3 as backup] is not a sink type



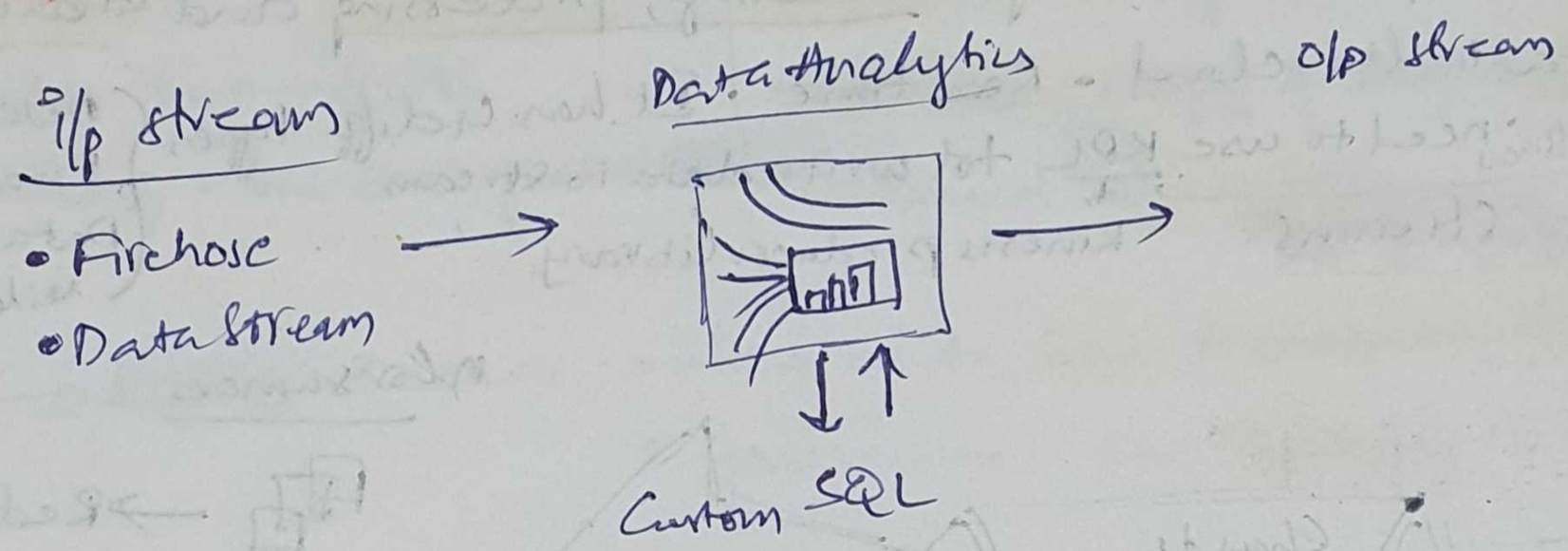
- → only 1 Consumer
- → Data immediately disappears once it's consumed
- → incoming data can be transformed to other format, compress, secure
- → You pay only for data that is ingested

Provisioned Throughput Exceeded
- Configure data producer to retry with an exponential backoff
- Increase # shards with your Data Streams to provide enough capacity

Kinesis Video Streams:-



Kinesis - Data Analytics



→ Fast real time reporting
→ You may need to use 2 streams (downside)

Kinesis cheat sheet! (4:50:51)

System Manager Parameter Store : SSM parameter Store

- Secure, hierarchical storage for configuration data management & secret management
- You can store data such as { passwords, database strings, license code } as Parameter values
- Store configuration data and secure strings in hierarchies and track versions
- You can encrypt parameters with KMS
- You group parameters together based on forward slashes { /example/pro/app/prod/cloudfrnd_key }

  This is hierarchy. This allows to fetch all parameters at diff levels
- Tier — { Standard, Advanced }
- Type — { String, StringList — comma seperated, Secure String — encrypted with KMS }

Tiers	Standard	Advanced
# param/region	10,000	10,000 – 100,000
Max size of param value	4KB	8KB
Param policies	No	Yes
Cost	Free	$0.005 per param/month

Can convert → Advanced

Standard ✗ Can't reverse

Parameter policies: (Only Advanced)

→ help in forcing you to update/delete parameters done asynchronous, periodic scans.

→ You can apply multiple policies → 3 { Expiration, Expiration Notification, No Change Notification.

CLI Hierarchy example:-

- aws ssm put-parameter --name "/planets/vulcan/population" --value 4.9B --type string
- ↳ /gravity --value 9.4G --type string
- ↳ /classification --value M --type string
- every new param is version 1, if you write put on the same name it will update to new version

---

aws ssm get-parameters-by-path / ⇒ --path /planet/vulcan

```
{"Parameters": [
  {"Name": " ",
   "Type": " ",
   "Version": 1,
   "LastModifiedDate": 1559...03,   epoc
   "ARN": ...,
   "Value": ...,
  },
  {
  },
]}
```

Secrets Manager :- protects secrets needed to access your applications and services. Easily rotate, manage and retrieve database credentials API keys and other secrets throughout their lifecycle.

→ Automatically store + rotate db creds – { RDS, Redshift, DocDB, otherdbs
[key/value] → for APIs

maria/mysql/postgres/oracle/sqlserver

need to provide: serveraddress, dbname, Port
127.0.0.1 prapis 5432

→ enforces encryption at-rest by using KMS.

→ price { $0.4 per secret/month
$0.05 per 10,000 API calls

→ Cloud Trail can monitor secret manager access, if you turn on, trail

Automatic rotation :- ⊙ (key/value) doesn't have automatic rotation
Any/other db you can set automatic rotation

⊙ upto 365 days – default 30days

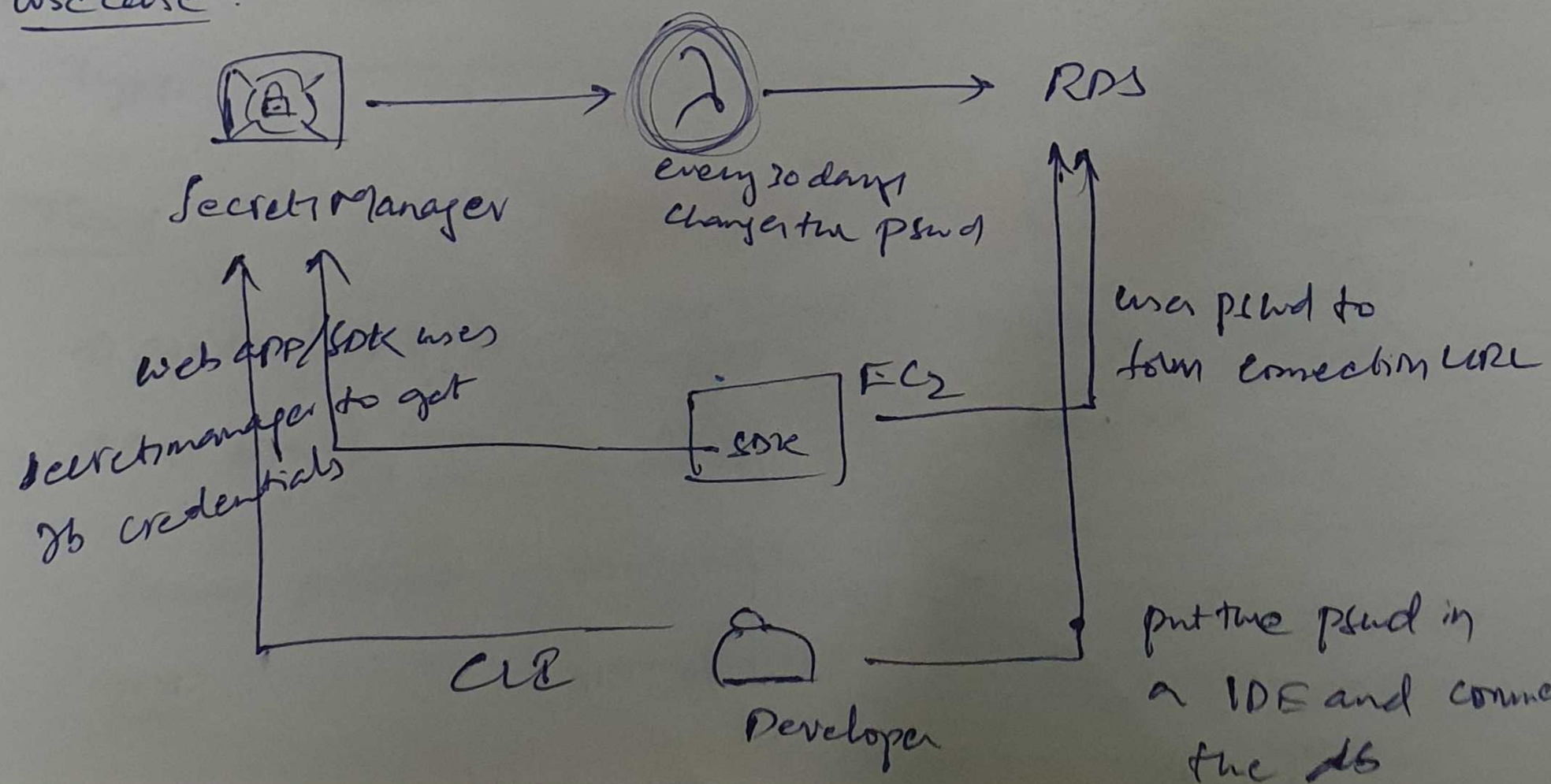
⊙ rotation via λ function

⊙ You can rotate pswd for superuser/developer programmatically accessing the db.

CLI Commands:

- aws secretsmanager describe-secret --secret-id enterprise/ShipDatabase
- aws secretsmanager get-secret-value --secret-id --version-stage AWSCURRENT

use case:



Dynamo DB : [05:05:05]

NOV 28

A Key-value & document NoSQL database which can guarantee consistent reads and writes at any sql.

NoSQL : neither a relational and doesn't use SQL to query data for result

Keyvalue : { Title : "S01E019 DS9 Duet" }

Document Store : A form of data storage which has a nested data structure

```
{ Series : 'DS9',
  Episodes : [
    { Season : 1,
      Episodes : 19,
      Title : 'Duet'.
    }
  ]
}
```

Features : + Fully Managed
+ Multi Region
+ Multi Master
+ Durable database
+ Built in security
+ Backup & restore
+ In memory caching

Provides : - Eventual Consistent Reads [default]
Strongly Consistent Reads

- Can specify the read & write capacity per second.
- the data is stored atleast across '3' different ~~Regions~~ AZ in 'SSD storage'

[Use "Conditional Writes" to avoid overwriting existing items]

Anatomy of Dynamo DB :

items ⇒ rows

Attributes ⇒ Column

Key ⇒ column name

Value ⇒ The actual data / cell

Read Consistency :

- The data is stored in 3 AZs.
- When data needs to be updated it has to write to all copies.

It is possible for data to be inconsistent when reading from a copy which has yet to be updated.

- You have to choose 1 of the 2 read consistency

Eventual Consistent Reads [Default]

- When copies are being updated it is possible for you to read and get inconsistent copy
- Reads are fast but no guarantee of consistency
- Data eventually get consistent within 1 sec

Strongly Consistent Reads

- When copies are being updated, you not get any return to your reads until all copies are updated
- higher latency but guarantees consistency
- All copies of data will be update in 1 sec

Partition: - is an allocation of storage for a table, backed by solid state drives (SSD) and automatically replicated across multiple AZ with an AWS Region. — AWS definition

- A partition is when you slice your table up into smaller chunks of data (a partition). It speeds up reads for very large tables by logically grouping similar data together — In general

- DynamoDB automatically creates partition in 2 cases
  (i) Exceeds every 10 GB of data
  (ii) when you exceeds the RCUs and WCUs for a single partition

dynamodb evenly splits this across the partitions ← { 3,000 max Read Capacity Units; 1,000 max Write Capacity Units

Primary key: •) determines where and how your data will be stored in partition
•) Can't be changed later

→ Simple - you use only partition key
→ Composite - you use both partition & sort key

Primary key*
Partition key [ID] [Number]
Add sort key
[Date] [String / Binary / Number]
•No data format in dynamo

Simple Primary key: -

- In this case partition key has to be unique.
- We don't know which partition dynamodb takes to store an item it depends on secret hash function which works based on the item data

Partition A

ID	date
1000	1993-01-03

Partition B

ID	date
1001	1993-01-04

Partition C

ID	date
1001	1993-01-05

new data

ID	data
1001	1993-01-[illegible]

F(x)

Partition key
[ID] [Number]

Composite Primary key: -

Partition key: Alien
Sort key: Name
the combination has to be unique

new data:

Alien	Name
Romulan	Tomalok

F(x)

Partition A

Alien	Name
Cardassian	Dukat
Cardassian	Dumar

Partition B

Alien	Name
Klingon	Worf
Klingon	Kor

Partition C

Alien	Name
Romulan	Tomalok
Romulan	Toreth
Romulan	Valdore

When using a sort key the records that have the same partition key will be kept together and sorted A-Z

Primary key Design :

Simple Primary key

→ Only Partition key

ID	Date
1000	1993-01-03
✗ 1000	1993-01-04

ID	Date
1000	1993-01-03
1001 ✓	1993-01-04

→ Distinct

→ Try to uniformly distribute the data

Composite Primary key

→ Partition + sort key

Alien	Name
Romulan	Tomalok
✗ Romulan	Tomalok

Alien	Name
Romulan	Tomalok
✓ Romulan	Valdok

Query & Scan :-

dynamodb (items) → query : Table / Partition(userid) = —
sort key(event_at) (=) —
↳ >, <=, between ....

add filters :- sort [Ascending, Descending]
attributes [all, projected]

Start Search

query :- ⊙ allows you to find items in a table based on primary key values
⊙ query any table / secondary index that has a composite primary key
⊙ By default reads as Eventually Consistent
if you need strongly consistent set ConsistenRead = True in table/GSI
⊙ By default all attributes (cols) are returned
if you want to filter use Project Expren = "C1, C2, C3 .."
⊙ By default is sorted in ascending order
if you want you can try ScanIndexForward = False for desc

example of a payload of a query :-

```
{ "ConsistentRead" : true,
  "ProjectionExpression" : "Id, Name, CreatedAt",
  "ScanIndexForward" : true,
  "Limit" : 20
}
```

Scan: • Scan through all items and then return one/more items through filters

- By default it returns all attributes for items
- Scans can be performed on tables and secondary indexes
- Can return specific attributes by using "Project Expression".
- Scan operations are sequential. You can speed up a scan through parallel scans using Segments and Total Segments parameters

Avoid scans because :- • Much less efficient than running a query
• As a table grows, scans take longer to complete
• A large table can eat all your provised throughput

Provisioned Capacity: [Read/Write : capacity mode 1]

* It is the maximum amount of capacity your application is allowed to read/write per second from a table or indexes.

* Throughput is measure in capacity units
  → RCUs - is Read Capacity Units
  → WCUs - is Write Capacity Units

* You can set dynamodb for provisioned to scale capacity up and down based on utilization

* If you go beyond your capacity you will get the error Provisioned Throughput Exceeded Exception.

This is known as throttling, request that are throttled will be dropped (data loss)

Auto Scaling
☑ Read Capacity
Target utilization 70 %
Min provised capacity 5 units
Max 1000 units
☑ Apply same settings to global secondary index

On-Demand Capacity:- [Read/Write : capacity mode 2]

→ Here you pay per request. default upper limit for RCUs/WCUs : 40,000
→ If the traffic is high it will spin up till 40,000 ⇒ might be very costly

usecase :- New tables with unknown workloads
Unpredictable app traffic
The ease of paying for only what you use

✗ No provisioned capacity
✗ No autoscaling

Calculating Reads

RCU: A read capacity unit represents
- 1 strongly consistent read per second
- 2 eventually consistent reads per second

for an item up to 4 KB in size

Strongly Consistent Reads	Eventual Consistent Reads
1. Round the data up to nearest 4 2. Divide the data by 4 3. Times the # reads	1. Round data up to nearest 4 2. Divide data by 4 3. Times the # reads 4. Divide the final by 2, take nearest whole number
50 reads @ 40 KB/item: $(40/4) \times 50 = 500$ RCUs	50 reads @ 40 KB/item $((40/4) \times 50)/2 = 250$ RCUs
10 reads @ 6 KB per item $(6/4) \times 10 = 20$ RCUs $\cong (8/4)$ $\cong 2$	11 reads @ 9 KB/item $[(9/4) \times 11]/2 = \frac{33}{2} = 17$ RCUs $\cong 12/4$ $\cong 3$
33 reads @ 17 KB/item $(17/4) \times 33 = 165$ RCUs $\cong (20/4)$ $\cong 5$	14 reads @ 24 KB/item $[(24/4) \times 14]/2 = 49$ RCUs 7

Calculating Writes:

WCU :- A write capacity unit represents
- 1 write/second

for an item up to 1 KB

50 writes @ 40 KB/item $\Rightarrow 40 \times 50 = 2000$ WCUs

4 writes @ 1 KB/item $\Rightarrow 1 \times 4 = 4$ WCUs

18 writes @ 500 bytes/item $= 18 \times$ 1 KB $= 18$ WCUs

Global Tables:

Amazon DynamoDB global tables provides a fully managed solution for deploying a multi-region, multi-master db, without having to build and maintain your own replication solution.

To create this you need to

1. use KMS CMK
2. Enable Streams
3. Stream type Old/New Image
4. then add a region

Transactions :-

- Represent a change that will occur to the database. If any dependent conditions fail then a transcation will rollback as if the db changes never occured.

- DynamoDB offers the ability to perform transcation at no additional cost using 2 operations { Transact Write Items / Transact Get Items

- DynamoDB transcations allows for all-or-nothing changes to multiple items both within and across tables

- DynamoDB performs 2 underlying reads/writes of every item in the transcation → 1. to prepare the transcation
  → 2. to commit the transcation.

- These 2 underlying r/w operations are visible in your Cloud Watch metrics
  You can use Condition Check with dynamoDB transactions to do preconditional check

Time-To-Live (TTL) :-

- This lets you have items in dynamoDB expire (deleted) at a given time.
- TTL is great for keeping db small & manageable and suited for temp/day conditional data [ex: event logs]

  You need to provide an attribute for TTL attribute in epoch format

Streams :- When you enable a stream on a table, dynamoDB captures every modification U/I/D and sends in near real time to custom. λ fun in the same sequence the event occurred.

Errors :- Throttling exceptions: rate of requests exceeds allowed throughput can happen while CreateTable, UpdateTable, DeleteTable

Provisioned Throughput Exceeds Exception: exceeded max allowed provided RCU/WCU

- In aws SDK, automatic retry with Exponential Backoff when an error occurs. It will attempt 50ms/100ms/200ms/400ms upto a minute before stopping

Index: • A db index is a copy of selected columns of data in a database which is used to quickly fetch

• Dynamo db has 2 types of Indexes
- LSI: Local Secondary Index [can only be created with initial load] — provides strong consistency
- GSI: Global Secondary Index [you should generally use GSI over LSI] — Can't provide strong consistency

Local Secondary Index:

• It's "local" in that every partition of an LSI is scoped to a base table partition. that has the same partition key value

• The total size of indexed items for any partition key value can't exceed 10 GB.

• Shares provisioned throughput settings for r/w activity within the table it is indexing

• limited to 5 per table (default)

• LSI can be created only during creating the initial table, can't be modified / deleted after creation

• LSI needs to have both —
- Partition key — same as initial table
- Sort key — different than initial table [else it will be same as initial table no value add on]

Global Secondary Index:

• It's considered "global" because queries on the index can span all of the data in the base table, across all partitions.

• The Index have no sizes restriction [LSI: 10GB]

• GSI have their own provisioned throughput settings : [LSI: shared with base table]

• limited to 20 per table [LSI: 5]

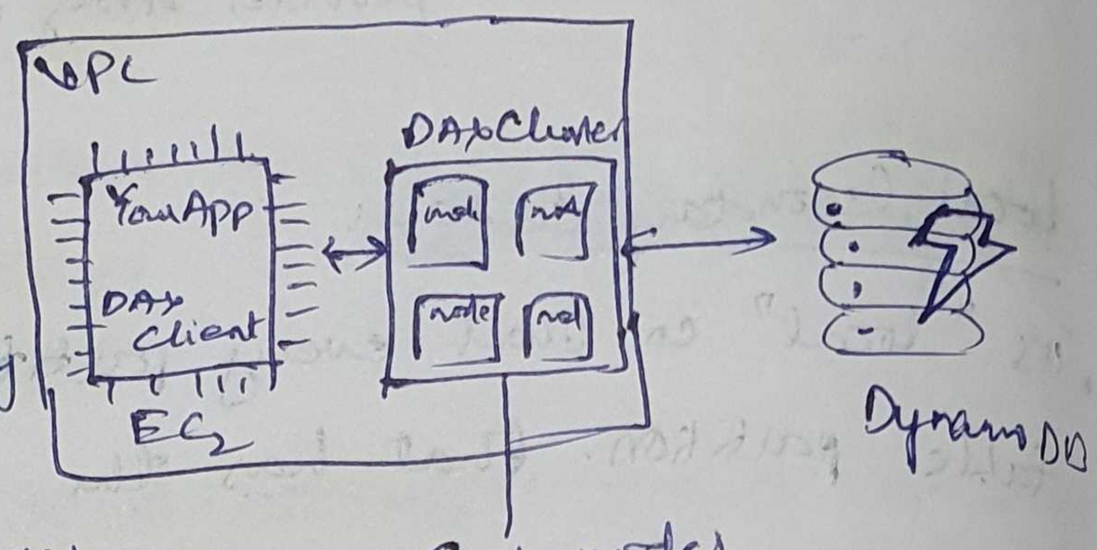
• GSI can be added, modified, deleted at any time [LSI: only during base table creation]

• GSI — partition key can be different than base table [LSI: same]
— Sort key optional [GSI: mandatory different]

DynamoDB - Accelerator (DAX):

- It is a fully managed in-memory cache for dynamoDB that runs in a cluster.
- dynamodb response time is in millseconds. DAX reduces it to Micro seconds.

→ DAX cluster consists of 1/more nodes.
→ each node runs its own instance of DAX caching s/w
→ one of the node serves as the primary node for the cluster.
→ addition nodes (if present) will serve as read replicas



→ Your app can access DAX by specifying the endpoint of the DAX cluster
→ The DAX client s/w works with the cluster endpoint to perform intelligent load balancing & routing.

DAX is ideal for:-

→ Apps that require the fastest possible response time for reads like gaming
→ Apps that read a small ~~#~~ items more frequently than others
→ Apps that are read-intensive but are also cost-sensitive
→ No caching is there on the dynamodb / your own app side

DAX is not ideal for:-

→ App that require strongly consistent reads.
DAX gives eventual consistency.
→ Apps that don't require μsecond response time.
→ Apps that are write intensive
→ If you already have any other caching mechanism in place then no need for DAX.
ex: ElasticCache, [Redis] ↳ for write intensive

Dynamo DB Cheat Sheet [05:49:14]

Dynamo DB demo [06:06:00]

Elastic Compute Cloud EC2 [06:55:31]

- Cloud computing service. Choose your OS, storage, memory, N/w throughput. launch and SSH into your server in minutes
- Resizable compute capacity
- AMI : RedHat, ubuntu, AmazonLinux, Win, SUSE
- Instance type : t2.nano , C4.8xlarge
  - Network card : speed of the card
    Public IP address
- Add Storage : (EBS) (EFS) — EC2 instance store
  - Network attached
  - hardware
- Configuration : Security Groups, KeyPairs, User Data, IAM Roles, Placement Groups

(Can't Reboot the spot Instance in EC2)
(at 5.3 TiB volume gp2 hits max IOPS)

Instance Types :-

General Purpose : Balance of compute/memory/n/w resources
A1, T3, T3a, T2 M5, M5a, M4 [Web-servers, code repositories]

Compute Optimized : Ideal for compute bound, high performance processing.
C5, C5n, C4 [Scientific modeling, dedicated gaming servers, advertising engine]

Memory Optimized : Fast performance for workloads that process large datasets in memory
R5, R5a, X1e, X1, z1d [in memory cache, db, real time big data analytics]

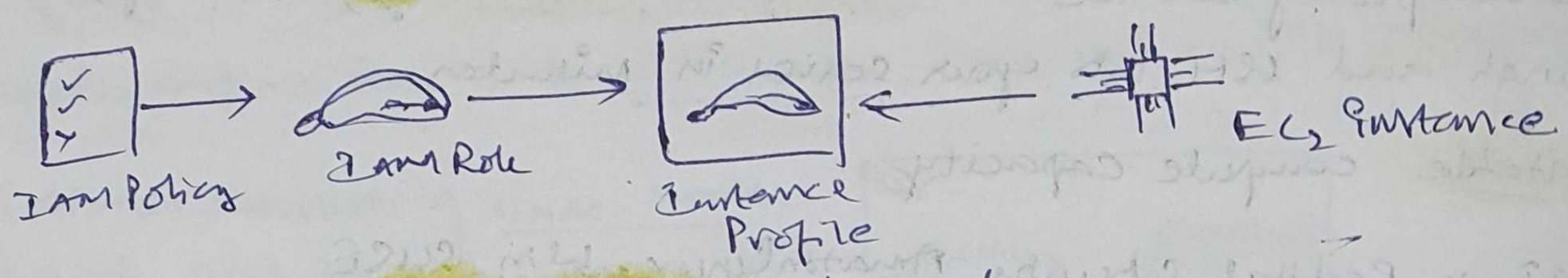
Accelerated Optimized : Hardware accelerators (or) co-processors
P3 P2 G3 F1 [Machine learning, computational finance]

Storage Optimized : High, sequential r/w access to large data sets on local storage
I3, I3en, D2, H1 [No SQL, transaction db, db warehouse]

- Instance size : generally double in price and key attributes

	vCPUs	Ram (GB)	OnDemand /hr	OnDemand / mon
t2.small	1	12	$0.023	$16.79
t2.medium	2	24	$0.0464	$33.87
t2.large	2	36	$0.0928	$67.74
t2.xlarge	4	54	$0.1856	$135.48

Instance Profile :- { You can attach a role to an instance via InstanceProfile
No need to provide [key, secret] in SDKs



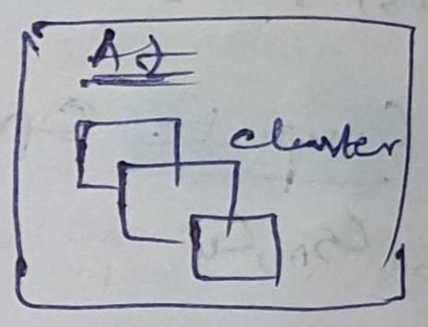
- An Instance Profile holds a reference to a role.
- The EC2 instance is associated with the Instance Profile.
- When you select an IAM role, when launching an EC2 instance AWS will automatically create the instance profile for you.

Instance profiles are not easily viewed via the AWS Console.

Placement Groups (optional) - (free)

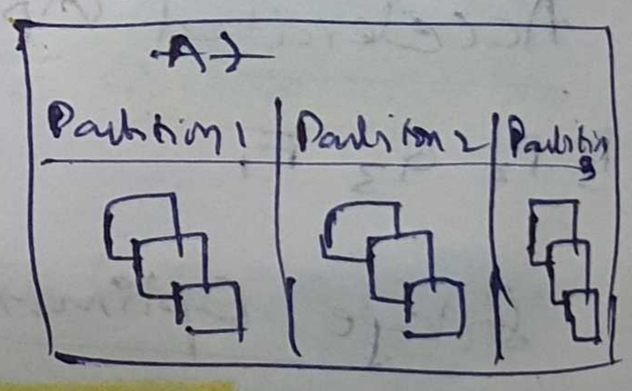
- choose the logical placement of your instance to optimize for communication, performance & durability.

Cluster :-



- packs instances close together inside an AZ
- low latency n/w performance for tightly coupled node-to node communication
- well suited high performance computing (HPC) applications
- clusters cannot be multi AZ

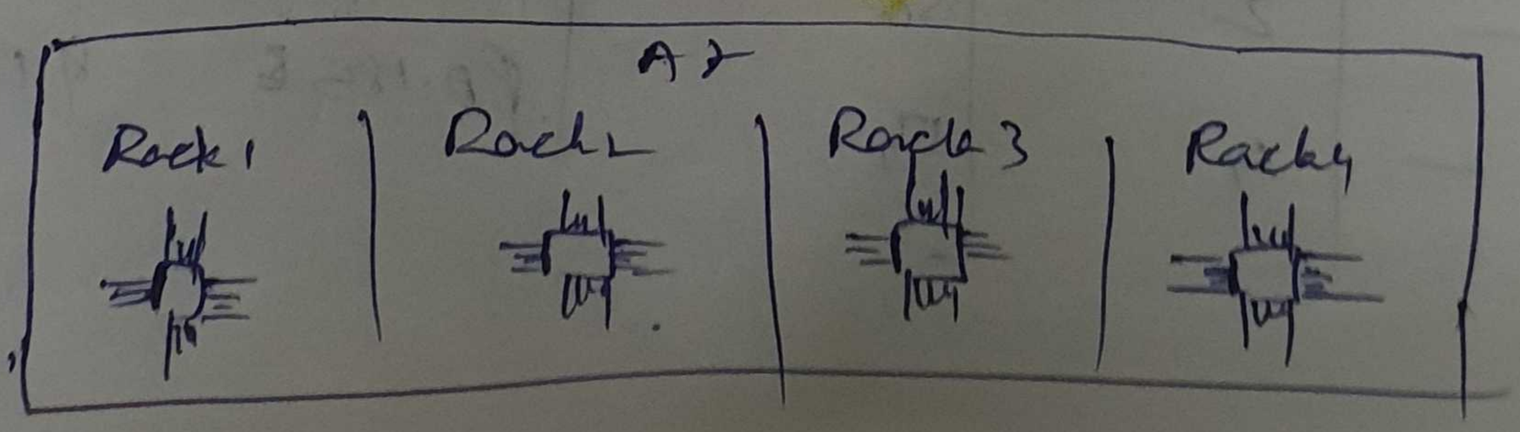
Partition :-



- spreads instances across logical partitions
- each partition do not share the underlying hardware with each other (rack per partition)
- well suite for large distributed and replicated workloads [Hadoop, Cassandra, Kafka]

Spread :-

- each instance is placed on a different rack
- when critical instance should be keep seperate from each other
- You can spread a max of 7 instance spreads. can be multi AZ



User Data:

- This is a script that will be automatically run when launching an EC2.
- You can install packages, apply updates & anything you like

`curl http://169.254.169.254/latest/user-data`

→ to get to know if user data was used & not during launch of an ec2 → run this curl from the ec2

bootstrap script for apache web server

```
#!/usr/bin/env bash
su ec2-user
sudo yum install httpd -y
sudo service httpd start
```

Meta Data: Additional info that you can get about your ec2 instance.

- `curl http://169.254.169.254/latest/meta-data`

{ /public-ipv4
/ami-id
/instance-type

Stop – No need to pay (compute) but for storage you need to
~~Terminate~~
RAM – erased
Private IP – remain same
Public IP – changes
Instance store volume → erases
Root device volume – Stays

hibernate: saves the contents from instance memory to root memory so that it can start once wake-up

KeyPair: Choose existing key pair or create new key pair
download the file
↓
This has private key file
public is with AWS

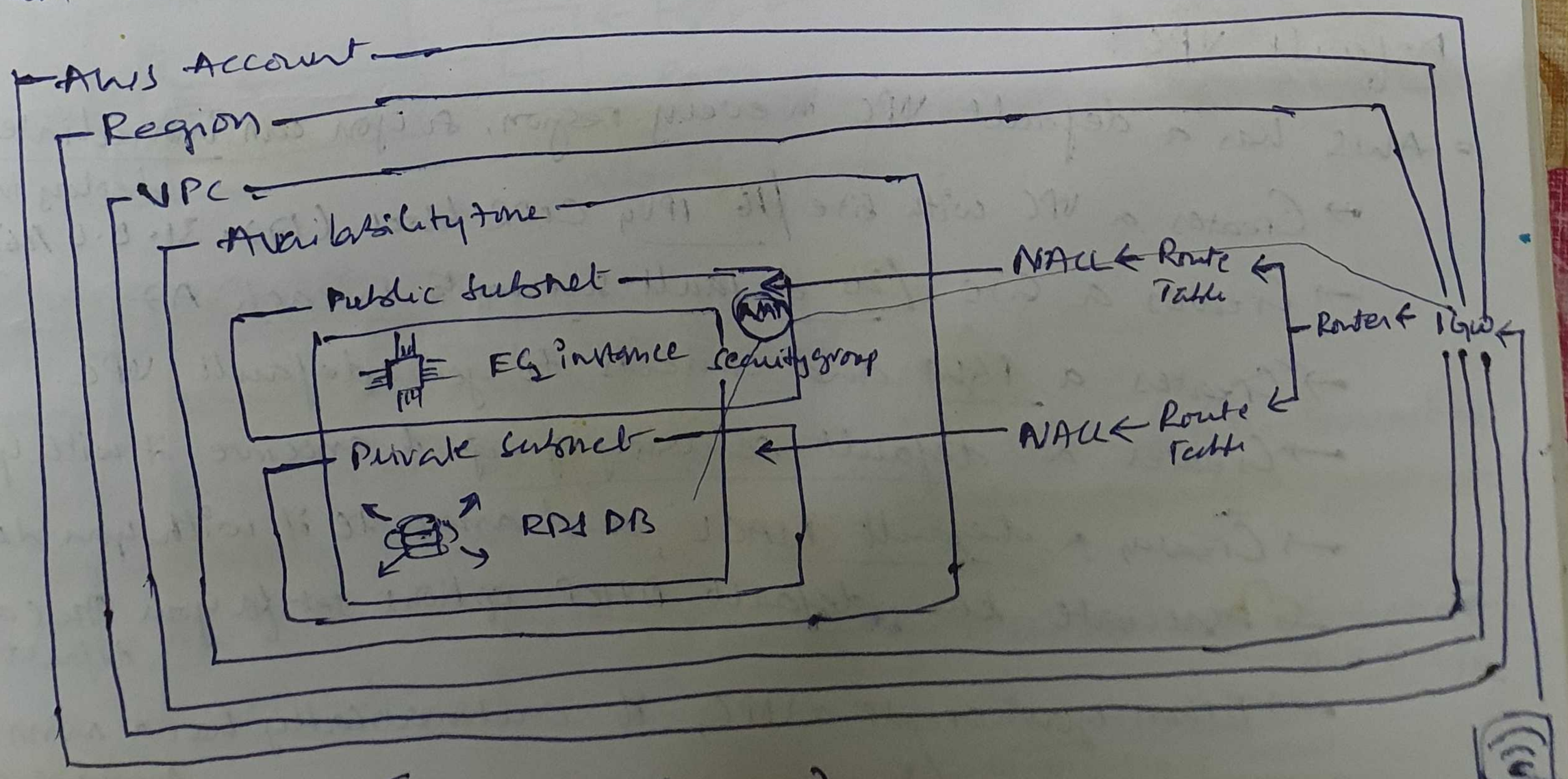
- Combine metadata with user data scripts to perform all sorts of advanced AWS staging automation

EC2 Cheat sheet [7:08:20]

On Demand
Reserved
→ Reserved
→ Convertible reserved
→ Scheduled reserved
(deprecated)
Spot, Dedicated Hosts → compliance, server bound s/w licences
→ Dedicated Instances → run on your h/w

Virtual Private Cloud (VPC) [7:10:20]

- Provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual n/w that you define.
- You can think it like your own personal data center, give control over virtual n/w env.



[IGW: Internet gateway]

Core Components —
- IGW: Internet Gateway
- VPN gateway: Virtual Private Gateway
- Routing tables
- NACLs: Network Access Control Lists (stateless)
- SG: Security Groups (stateful)

- Public Subnets
- Private Subnets
- NAT Gateway
- Customer Gateways
- VPC Endpoints
- VPC peering

Key Features:

→ VPCs are Region Specific, 5 VPCs/region

→ each region has a default VPC

→ 1 VPC ⇒ max 200 subnets

→ Free: VPCs, Route tables, NACLs, IGW, SG, Subnets, VPC peering

→ Cost $: NAT Gateway, VPC endpoints, VPN Gateway, Customer Gateway.

→ DNS: domain name of your instance: public IPV4: ec2-54.....amazonaws.com (default turned off, you need to turn on for your instance like ec2)

Name tag: MyVPC
IPV4 CIDR block: 10.0.0/16
IPV6 CIDR block: ◯ No IPV6 CIDR block
◉ Amazon provided IPV6 CIDR (address of the VPC)
Tenancy: Default
IPV6 CIDR block: 2600:1f16:9e0:8d00::/56

Default VPC:

- AWS has a default VPC in every region, so you can immediately deploy instances.

→ Creates a VPC with size /16 IPV4 CIDR block (172.31.0.0/16)

→ Creates a size /20 default subnets in each AZ

→ Creates a IGW and connects to your default VPC.

→ Creates a default security group & associate it with your default VPC

→ Creates a default NACL, and associate it with your default VPC

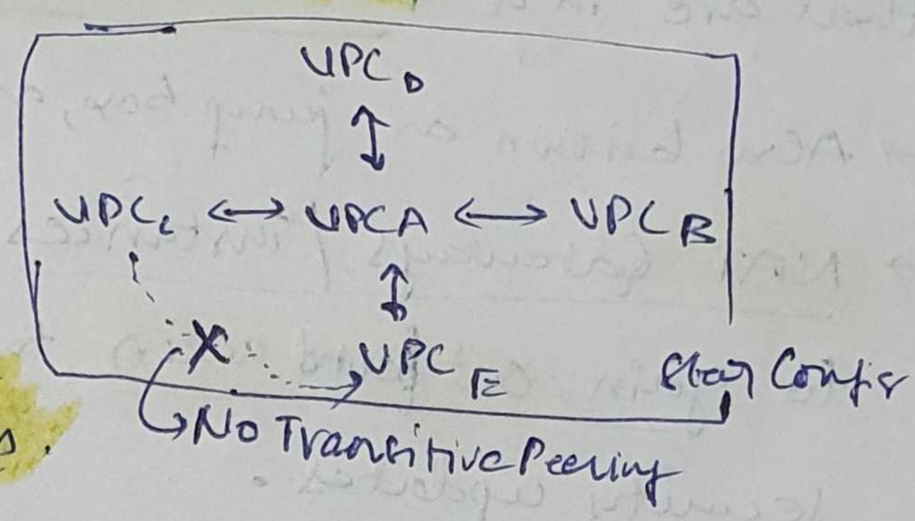
→ Associate the default DHCP options set for your AWS account with default VPC

→ When you create a VPC, it automatically has a main Route table

0.0.0.0/0 : is also known as default
It represents all possible IP address

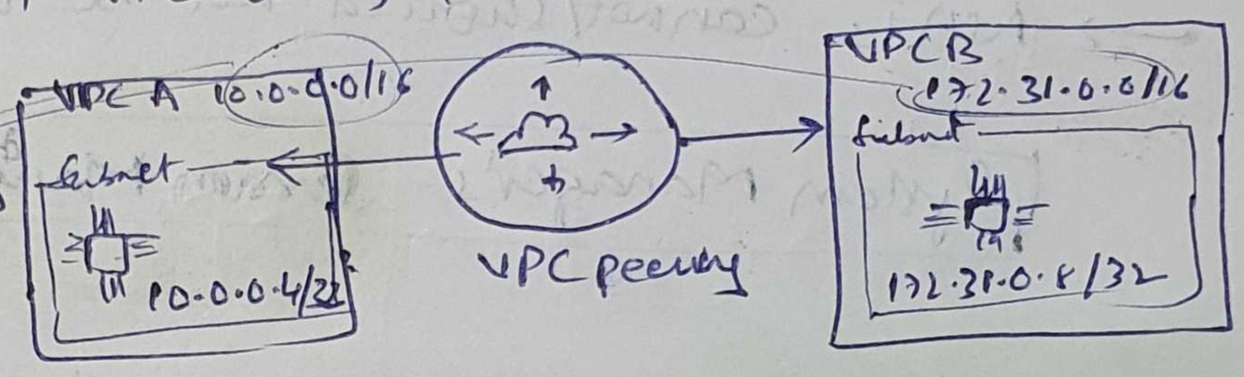
→ When we specify 0.0.0.0/0 → in our route table for IGW, we are allow internet access
→ in our security groups, inbound rules, we are allowing all traffic.
from the internet access our public resources

VPC peering: Allows you to connect 1 VPC with another over a direct n/w
Route using private IP address.



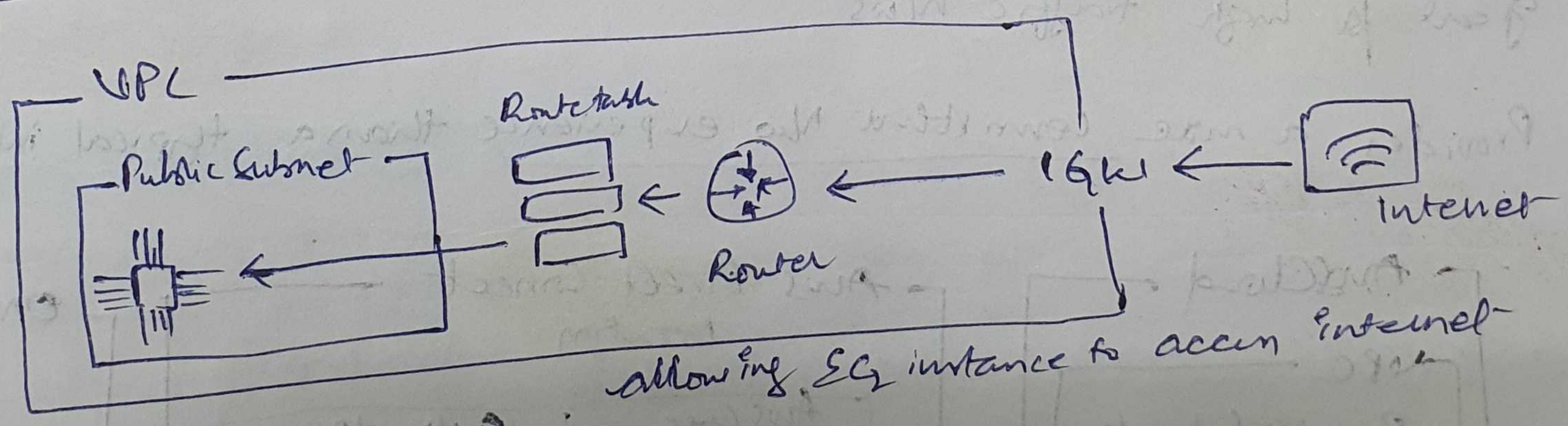
→ instances on peered VPCs behave just like they are on the same n/w.
→ connect VPCs across same/diff AWS acc & regions.

→ Peering uses a Star Configuration 1: Central, 4 - others
→ No Transitive Peering :
→ No overlapping CIDR blocks



Route Table

These are used to determine where n/w is directed
Each subnet in your VPC must be associated with a route table
A subnet can only associate with 1 route table at a time.
1 route table can associate with multiple subnet.



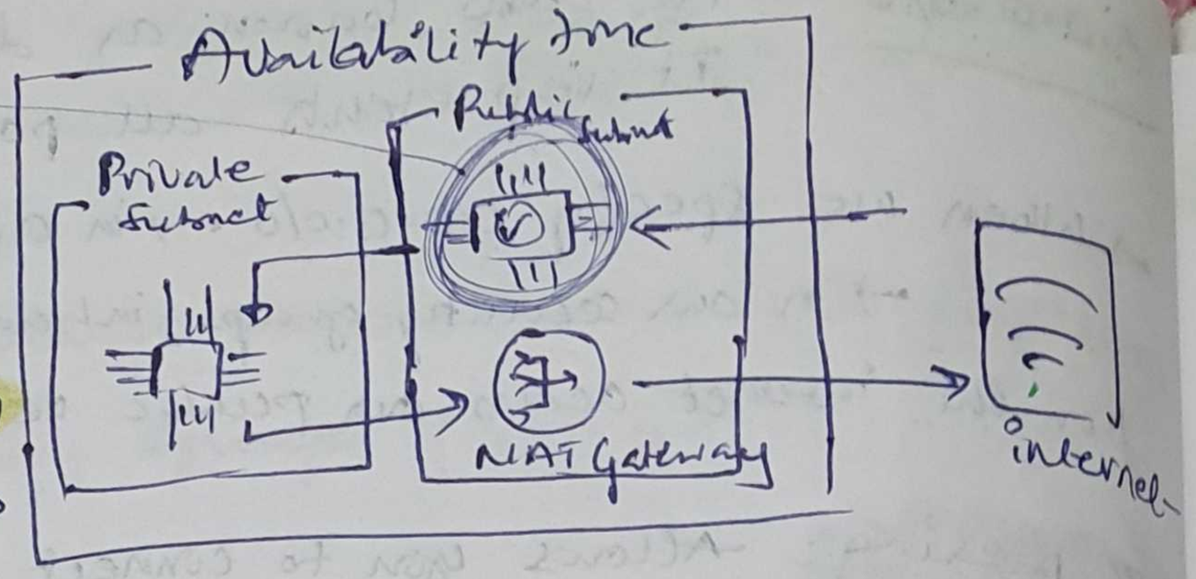
Internet Gateway (IGW):

IGW allows VPC to access to the internet. It does 2 things
① provides target in your VPC route table for internet-routable traffic
② perform N/w add translation (NAT) to instances that have been assigned public IPs

To route out to the internet you need to add in your route tables to the IGW and destination 0.0.0.0/0

destination	target
10.0.0.0/16	local
0.0.0.0/0	igw-id

Bastion / Jumpbox:



→ Bastions are EC2 instances which are security hardened. They are designed to help you gain access to your EC2 instances via SSH/RDP that are in a private subnet.

→ Also known as jump box, as they are jumping from 1 box to other.

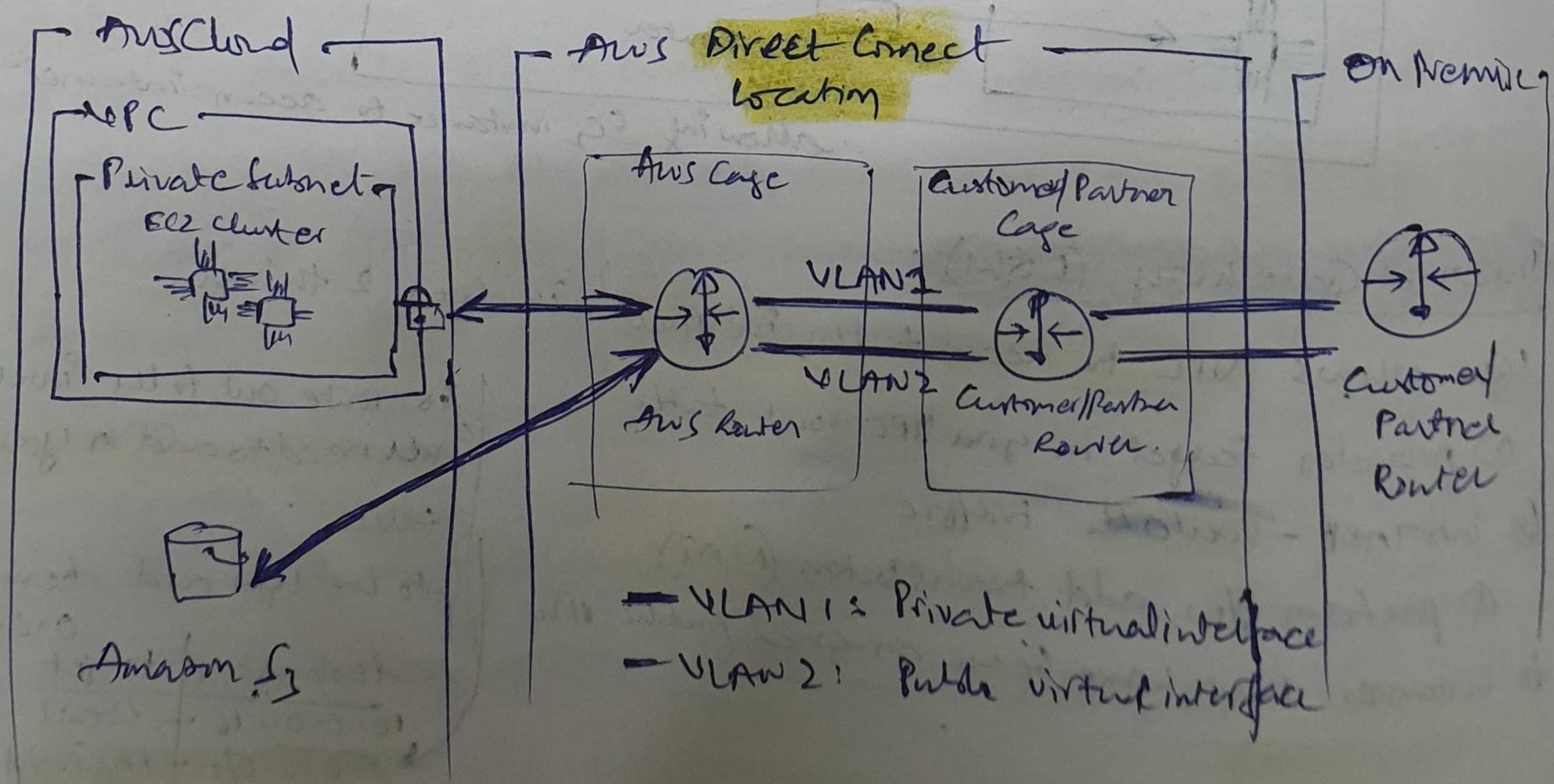
→ NAT Gateways / instances are only intended for EC2 instance to gain outbound access to the internet for things such as security updates.

→ NATs cannot/should not be used as Bastions.

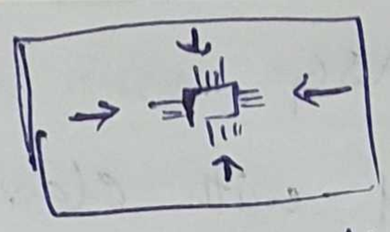
System Manager's Session Manager replaces the need for Bastions

AWS Direct Connect:

- AWS solution for establishing dedicated n/w conn from on-premise to AWS
- Very fast n/w — { Lower bandwidth : 50MB-500MB
  { higher bandwidth : 1GB - 10GB
- Helps to reduce cost & increase bandwidth throughput great for high traffic n/ws
- Provides a more consistent n/w experience than a typical internet based connected



Auto Scaling Groups (ASG)



- ASG set scaling rules which will automatically launch additional EC2 instances / shutdown to meet current demand
- ASG contains a collection of EC2 instances that are treated as a group for the purposes of automatic scaling and management.
- Automatic scaling can occur via
  1. Capacity Settings
  2. Health Check Replacement
  3. Scaling Policies.

Capacity Settings:

Min : # EC2 should at least be running (ASG will always meet this)

Max : # EC2 allowed to be running

Desired : # EC2 you want to ideally run.

Health Check Replacements:

- ASG will perform health check on EC2 to determine hw, hw issues.
- This is based on EC2 status checks. If unhealthy ⇒ ASG will terminates and then launchs a new instance
- Based on ELB health check, by pinging an HTTP(S) endpoint with an expected response. If ELB thinks unhealthy ⇒ sends to ASG ⇒ ASG terminates & launches new

Scaling Policies

→ Scaling out: Adding more instances
→ Scaling In: Removing instances

→ **Traget Tracking Scaling Policy:**

- maintains a specific metric at a target value.

ex: If avg CPU utilization > 75% then add another server
[Network In/out, ALB reqs count per target]

→ **Simple Scaling Policy:** Scales when an alarm is breached
No longer recommeded, its a legacy scaling policy

→ **Scaling policies with steps:** Scales when an alarm is breached, can escalates based on alarm value changing.

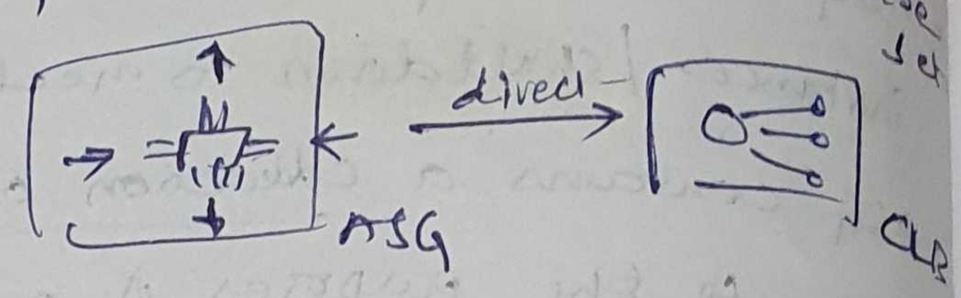
Execute Policy when: New Code Build
Take Action: Add [1] when Sec build < 2
[2] < 3
[3] < 4
add steps

ELB integration with ASG

ASG can be associated with elastic load balancers (ELB).

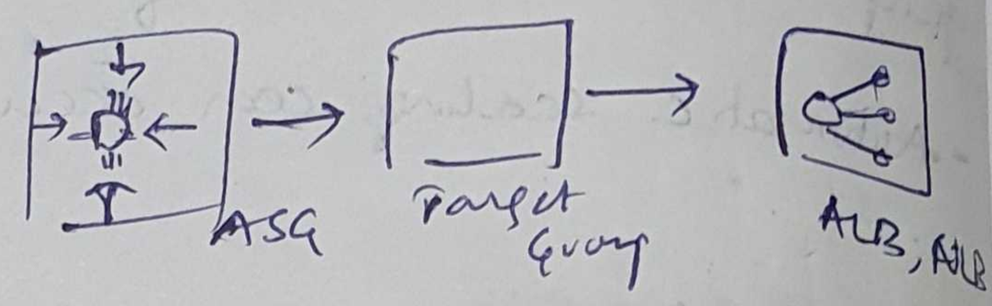
When ASG is associated with ELB, richer health check can be set

- Classic load Blancer are associated **directly** to the ASG

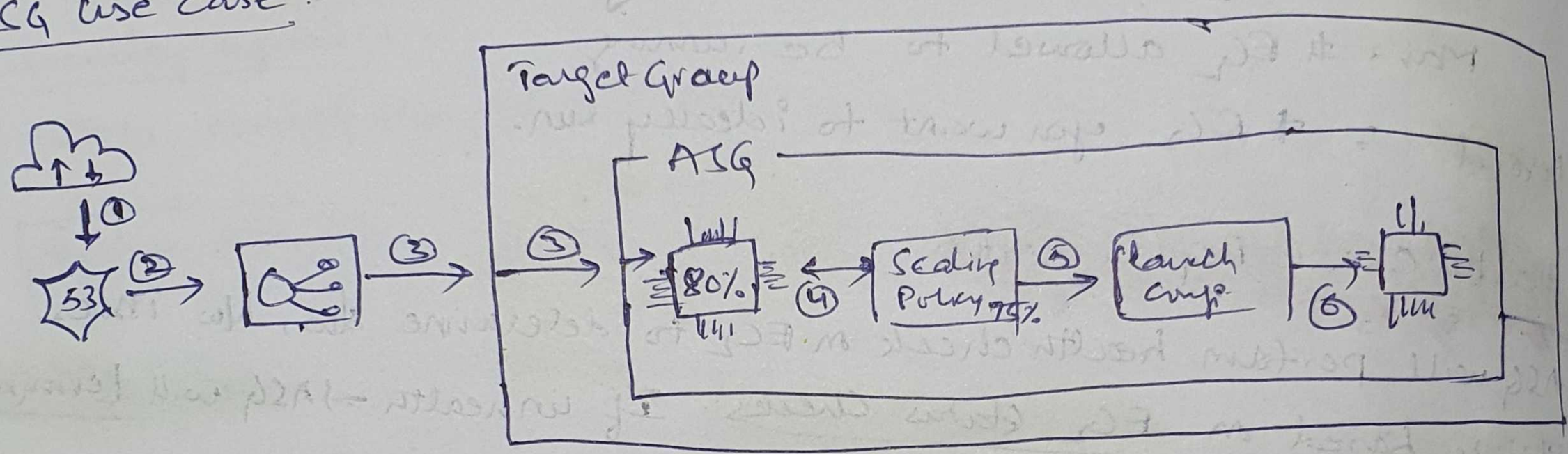


- ALB / NLB are associated
  (application / N/w)

indirectly via thier Target groups



ASG use case:



Launch Configuration.

- A launch configuration is an instance configuration template that an Auto Scaling group uses to launch EC2 instances.
- → A launch Configuration is the same process as launching an EC2 instance except you are saving the conf to launch later.
- → "Can't be edited". You need create a new one / Clone existing and then manually associate the new launch config
- → Launch templates : launch Config with **Versioning**.

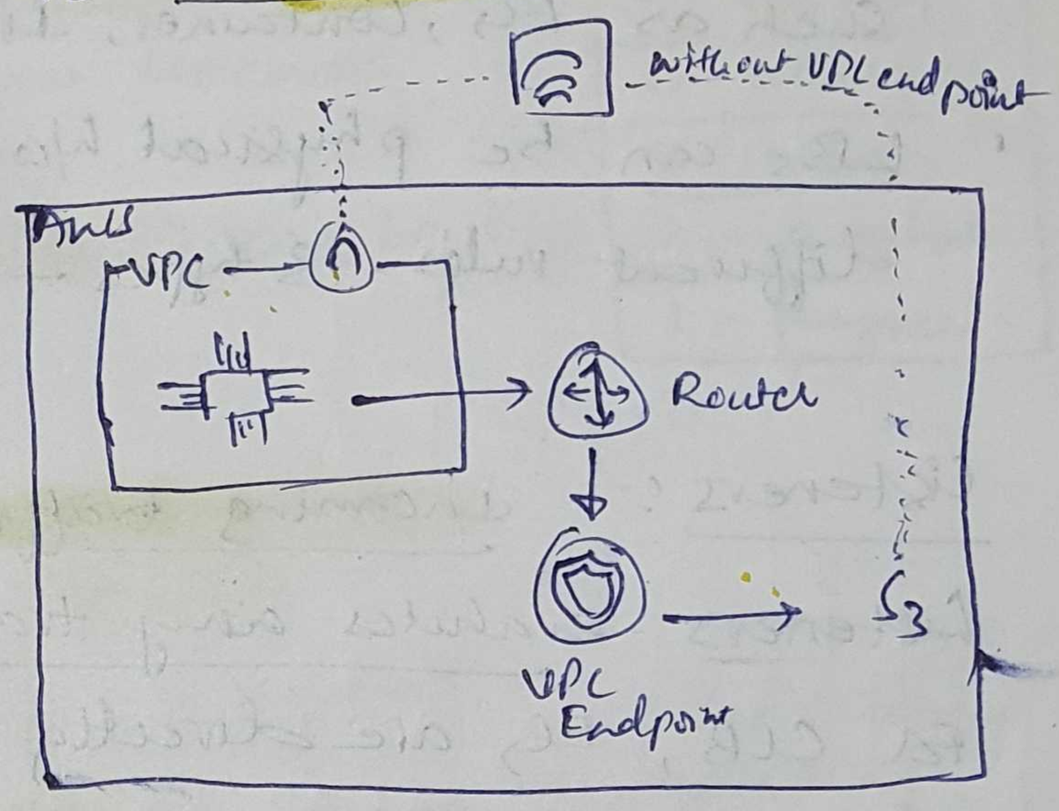
ASG cheat sheet: [7:30:58]

VPC End Points :-

Its like a secret tunnel where you don't have to leave the AWS n/w. For connecting with other services.

→ allows you to privately connect your VPC to other AWS services / VPCs

→ Types — { 1. Interface Endpoints
2. Gateway Endpoint



- Eliminates need for an IGW, NAT devices, VPN connect, AWS Direct Connect.
- instances in VPC No need public IP address to communicate with other AWS services
- Traffic b/w VPC & other AWS services don't leave AWS n/w
- Horizontally scalled, redundant and highly available VPC component
- Secure, no availability risks, no bandwidth constraints of your traffic

Interface ~~Endpoint~~ Endpoint :-

- These are Elastic Network Interface (ENI) with a private IP address.
- They serve as an entry point for traffic going ~~through~~ to a supported service
- powered by AWS Private Link.

Pricing per VPC /AZ /hr : 0.01
GB data processed : 0.01 } ~$7.5/month

Gateway Endpoint :- Free

- This is a gateway that is a target for a specific route in your route table
- used for traffic destinated for a supported 2 AWS services → S3 & DynamoDB
- To create this you need to specify → VPC, → other service (S3/DynamoDB).

VPC Endpoints Cheat Sheet : [7:35:00]

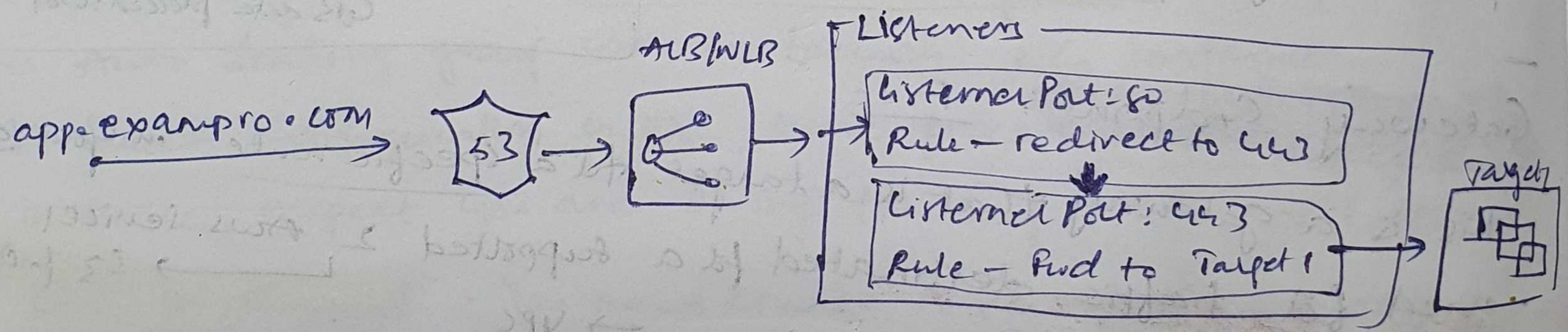
Elastic Load Balancers (ELB):

- Distributes incoming application traffic across multiple targets, such as EC2, container, IP addresses, ...
- LBs can be physical h/w & virtual s/w. They can balance based on different rules. 3 types —
  - ALB — (HTTP/HTTPS)
  - NLB — (TCP/UDP)
  - CLB - legacy

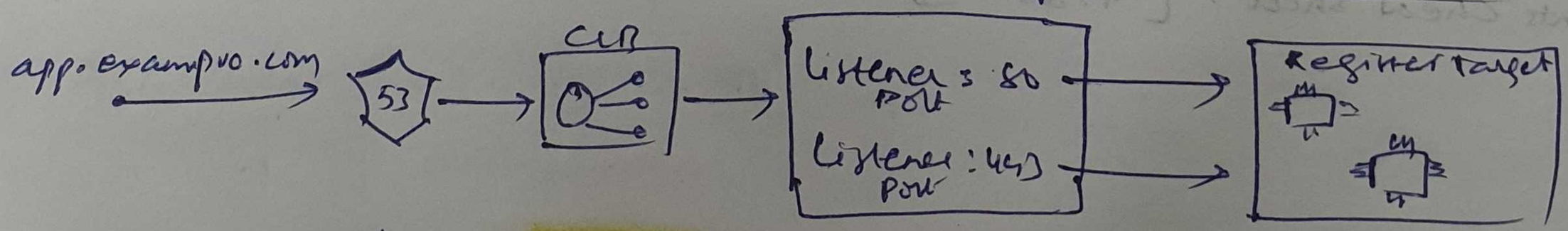
Listeners :- Incoming traffic is evaluated against listeners. Listeners evaluates any traffic that is matches the listeners port. For CLB, EC2 are directly registered to the load balancer

Rules :- (Not available for CLB)
Listeners will then invoke rules to decide what to do with the traffic. Generally the next step is to forward traffic to a target group

Target Group :- (Not available for CLB)
EC2 instances are registered as targets to a Target Group. logical groups of EC2



CLB :- CLB sends traffic to listeners. When port matches, it then forwards to EC2 instances registered. No Rules. No Target group



→ Can balance HTTP(S)/TCP (not all at the same time)
→ Sticky session, cross-zone load balancing
→ Returns (504) timeout error if underlying webserver/db not responding
→ Not recommended (it's a legacy LB).

ALB: ⊙ Is designed to balance HTTP and HTTPS traffic

⊙ It operates at layer 7 (of OSI model)

⊙ ALB has a feature called Request-Routing, which allows you to add routing rules to your listeners based on the http protocol.

⊙ Web Application Firewall (WAF) can be attached to ALB.

use case – Web applications

7 - Application
6 - Presentation
5 - Session
4 - Transport
3 - N/w
2 - Datalink
1 - physical

NLB: ⊙ Are designed to balance TCP/UDP.

⊙ They operate at layer 4 (of OSI model) low latency

⊙ Can handle millions of req/second, while still maintaining.

⊙ Can perform Cross-Zone load balancing

use case: multiplayer vide games.

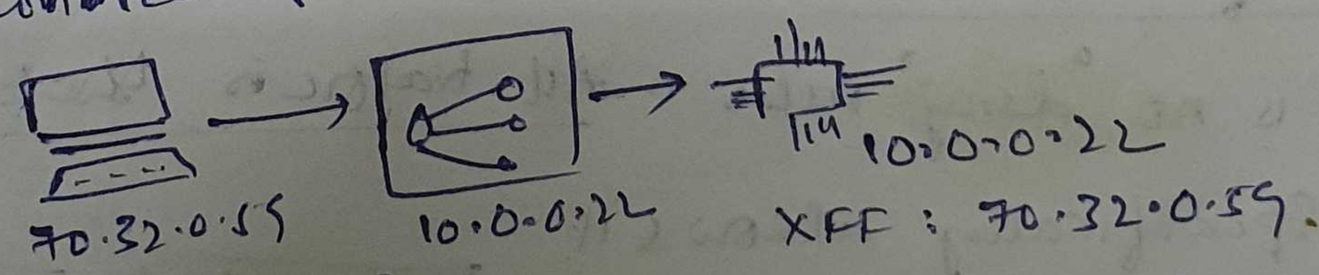
Sticky Sessions: Is an advanced load balancing method that allows you to bind a user session to a specific EC2.

- Ensures all req from the same session are sent to the same instance.
- Typically utilized in CLB, can be enabled in ALB to target group not individual EC2
- Cookies are used to remember which EC2 instance
- useful when specific information is only stored locally on a single instance

(ELB access logs are used to track i/p IP etc. for multi AZ configured targets)

X-Forward-For (XFF) Header:

- If you need IPV4 address of a user, check XFF header.
- The XFF header is a command method for identifying the originating IP address of a client connecting to a web server through an HTTP proxy & a LB.



ELB Health Check:

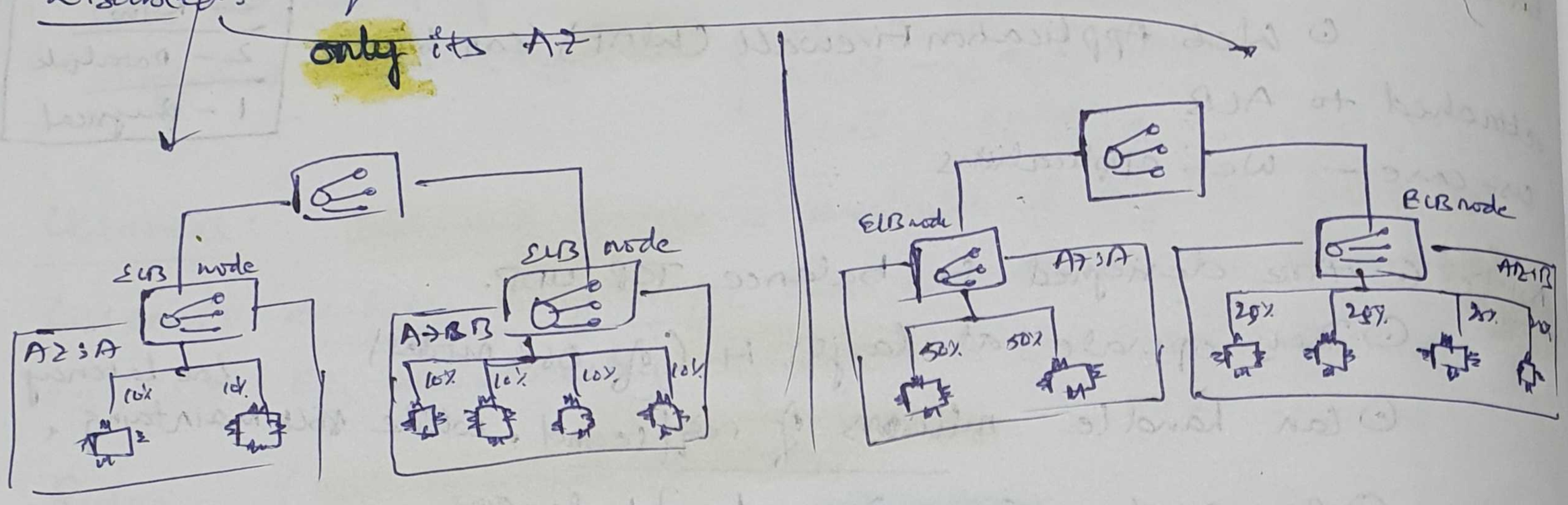
- CLB → you can find in CLB itself
- ALB, NLB → you can find health check on Target Group

- Returns: InService / Out of Service
- ELB doesnot kill unhealthy instance. It will just redirect traffic to healthy instances

Cross-Zone Load Balancing: only available for classic and Network LBs

Enabled : reqs are distributed evenly across the instances in all enabled AZs

Disabled : reqs are distributed evenly across the instances in only its AZ



Request Routing ALB:

- Apply rules to incoming traffic and then forward or redirect traffic (or) authenticate

Rules
- 1. Host Header
- 2. Source IP
- 3. Path
- 4. HTTP header
- 5. HTTP header method
- 6. Query string

ELB Cheat Sheet [07:46:30]

((Connection refused ⇒ SG is ok, instance failed))

((If you are not able access your app and time out error ⇒ mostly SG issue))

Deep

Security Groups:

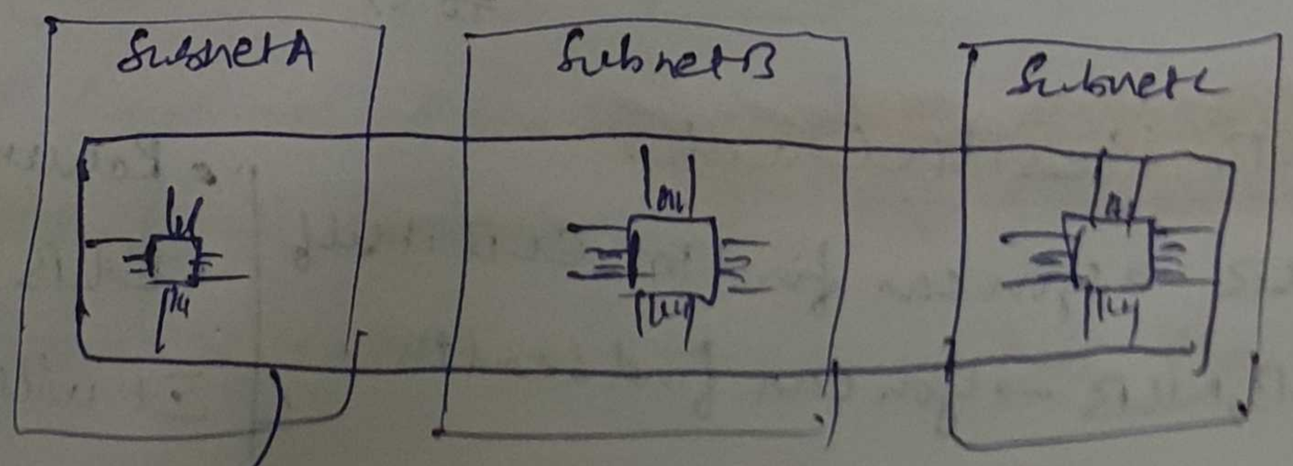
→ Stateful ⇒ inbound rule & outbound rule

A virtual firewall that controls the traffic to and from EC2 instances.

- each security group contains a set of rules that filter coming into and out of EC2 instances

inbound				Outbound
Type	Protocol	Port Range	Source	Des
SSH	TCP	22	MyIP: 23.245.86.48/32	...

- There is no "deny" rules - All traffic is blocked by default unless a rule specifically allows it
- multiple EC2 across diff subnets can belong to one Security Group



Security group : sg-web-servers

- instead of providing i/p address in the rules you can provide another security group as well

Limits: → upto 10,000 Security groups in a Region (default: 2,500) → raise service req to increase this.

→ upto 60 inbound and 60 outbound rules per SG.

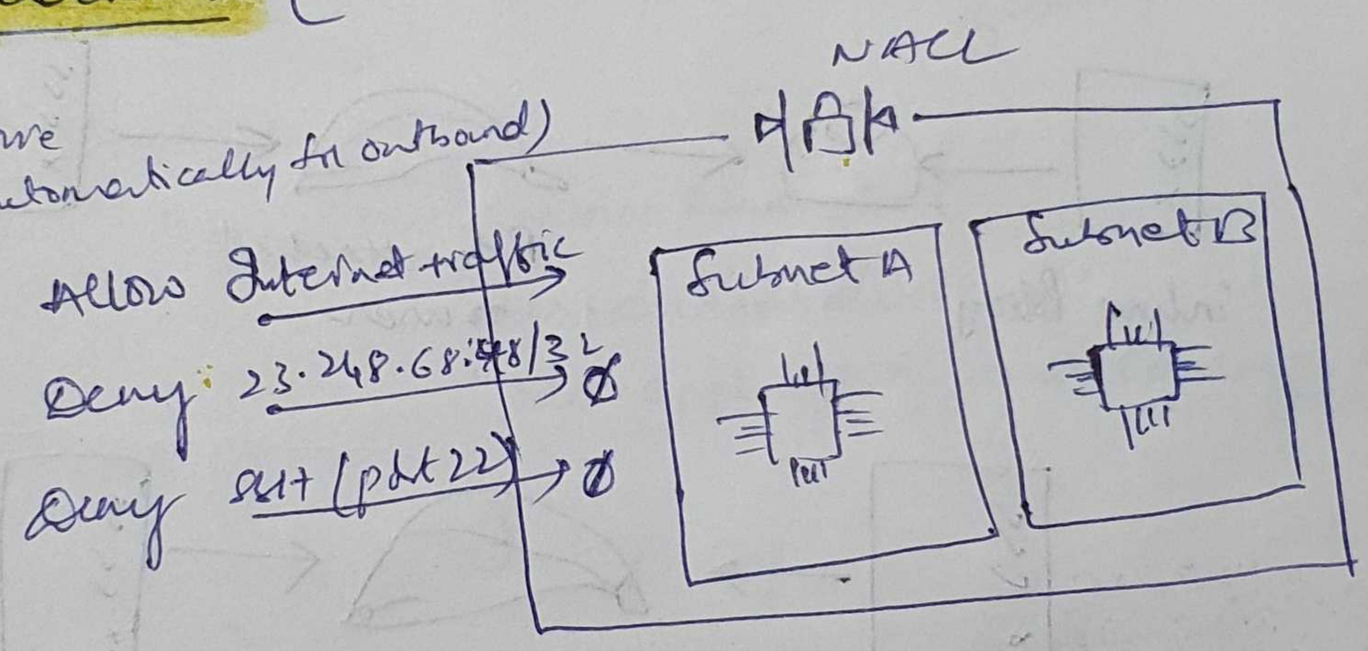
→ 16 SG per Elastic Network Interface ENI default to (5) → ↑

Dec 2

Network Access Control List (NACLs):-

An optional layer of security that acts as a firewall for controlling traffic in and out of subnets.

- VPCs automatically get a default NACL
- Can allow and deny traffic into the subnets
- have Rule #, order of evaluation lowest to highest.
- max rules 32766, Recommend to work in 10/100 increments
- Can block a single IP address [Can't do this in Security Groups]
- Stateless [inbound rule are not applied automatically for outbound]



NACL cheatsheet [7:56:46]

VPC demo [7:58:10]

NACL → { need explicit inbound & outbound rules } → stateless

22 < SSH : secure shell
SFTP : secure file transfer protocol

3389 : RDP remote desktop

21 : FTP : file transfer protocol

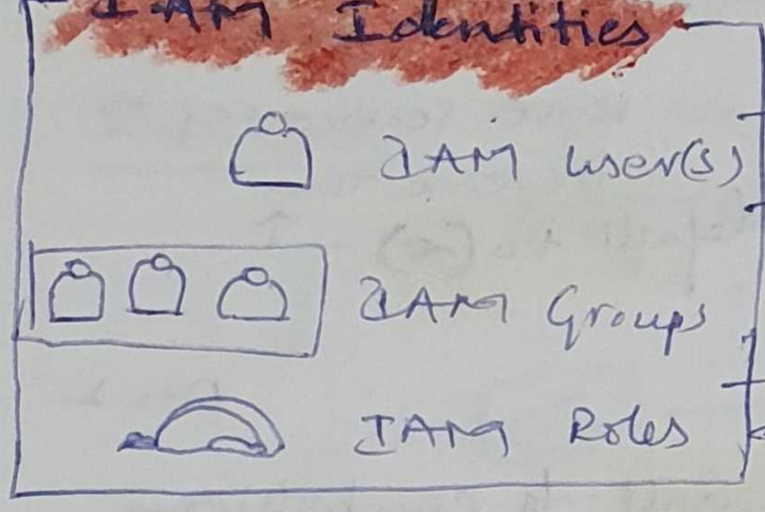
80 : HTTP access unsecured websites

443 : HTTPS

IAM : Identity Access Management

It allows management of access of users and resources

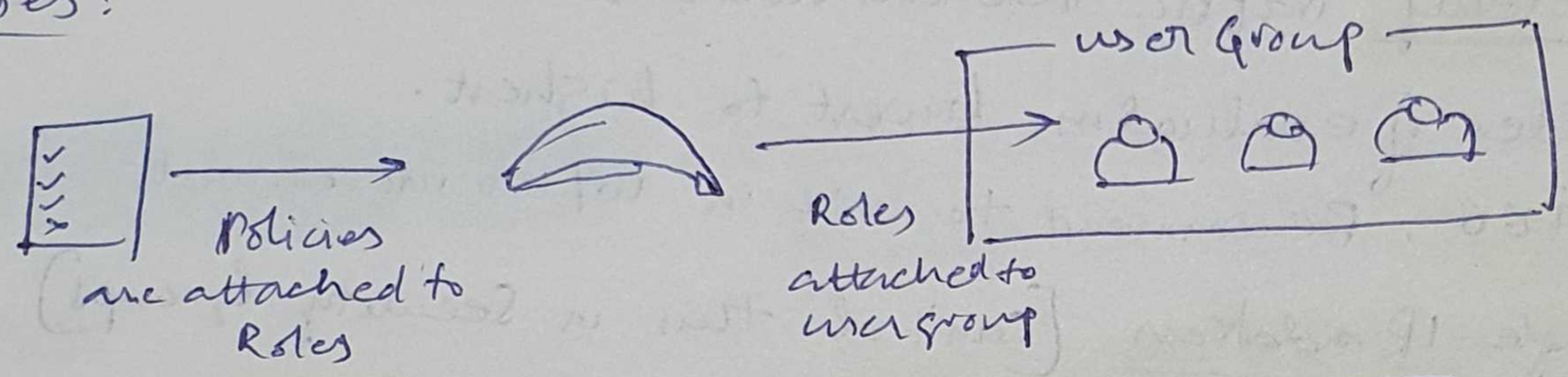
IAM Identities

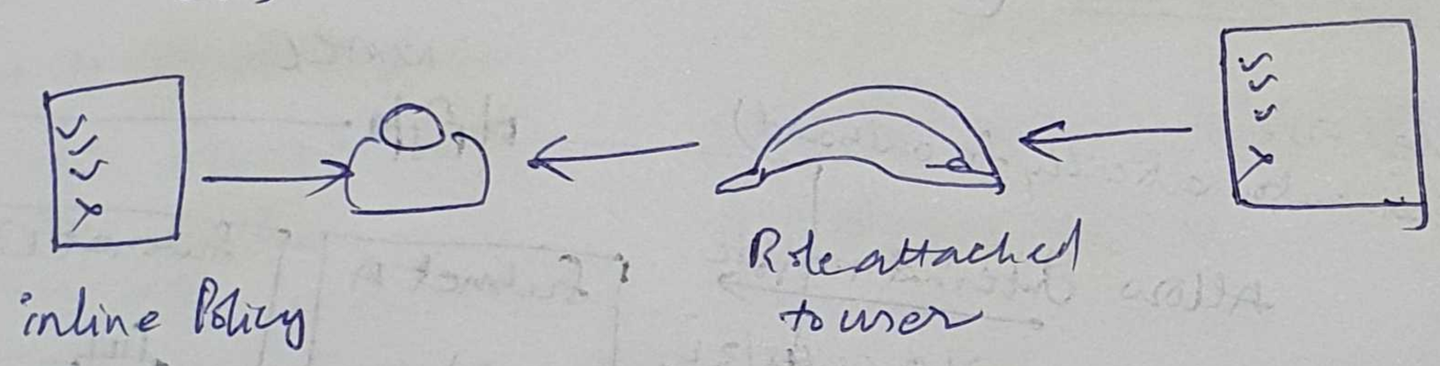


- IAM user(s) — end users [console/access Aws resources programmatically]
- IAM Groups — group of users sharing the same permissions levels. ex: dev/auditors/admins
- IAM Roles — Associate permissions to a Role & the assign it to user/group ~~resources~~.

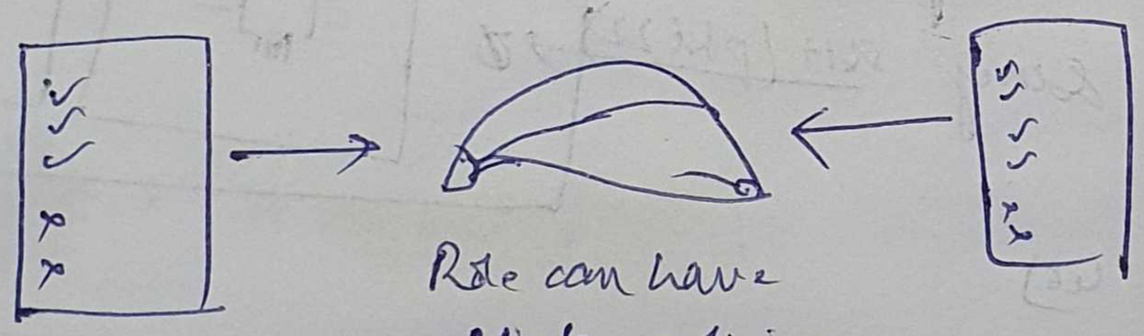
IAM Policies — JSON doc which grant permissions for a specific user, group & role to access services, policies are attached to IAM Identities.

usecases:

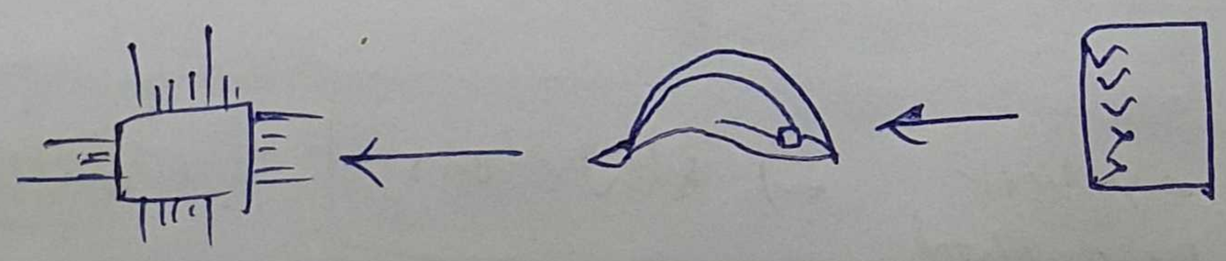




A user can have a role directly attached. A policy can be directly attached to a user (called an Inline Policy)



Role can have multiple policies.



Role can be attached to AWS Resources.

Type of Policies

Managed Policies :- A policy which is created and managed by AWS which you can edit. Managed Policies are labeled with AWS

Customer Managed Policies :- A policy created by the customer which is editable. customer policies have no symbol beside them.

Inline policies :- A policy which is directly attached to the user.

Policy example:

```
{
  "Version": "2012-10-27",
  "Statement": [{,
    "Sid": "Deny-Barcley-S3-Access",
    "Effect": "Deny",
    "Action": "S3: *",
    "Principal": {
        "AWS": ["arn:aws:iam::123456789012:barclay"]
    },
    "Resource": "arn:aws:s3:::my-bucket"
    },,
    {,
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "*",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": [
                "rds.amazonaws.com",
                "rds.application-autoscaling.amazonaws.com"
            ]
        }
    }
  }
]
}
```

Version → policy language version

Statement → container for the policy element you are allowed to have multiples.

Sid → [optional] way of labelling your statements statement identifier

Effect → Set whether policy will Allow/Deny

Principal → account/user/role/federated user to which you would like to allow/deny permission
need to mention role/user or service
"arn:aws:iam::1234...9012:root"

Action → list of actions you want to allow/deny

Resource → The resource to which the action(s) applies
need to mention bucket name
"arn:aws:s3:::mybucket/*"

Condition → [optional] circumstances under which the policy grants permission

denies any requests that don't include the [x-amz-serverside-encrp] requesting with KMS S3

```
"Condition": {"StringNotEquals": {
    "S3:x-amz-server-side-encryption": "aws:kms"
    }
}
```

Password Policy: In IAM you can set a Password Policy. To set the min requirements of a password and rotate passwords so users have to update their passwords after X days.

IAM Access Key:- •) Access keys allow users to interact with AWS service programmatically via the AWS CLI/SDK.

•) You are allowed 2 access keys per user

MFA: → Can be turned on per user
→ The user has to turn on MFA themselves.
Admin can't directly enforce users to have MFA
→ Admin can create a policy requiring MFA to access certain resources

Temporary Security Credentials:

- These are just like Programmatic Access keys except they are temporary.
- These are useful in scenarios like { identity federation, delegation, cross-account access, and IAM roles }
- These last for min to hours
- These are not stored with the user but are generated dynamically & provide to the user when requested.
- These are basis for roles and indentity federations.
  ↓
  We are already using Temporary Security Credentials.
  AWS automatically generates STS token (temp creds) for IAM roles.

Identity Federation:

→ The means of linking a person's electronic identity and attributes, stored across multiple distinct identity management systems.
Simple def: Identity Federation allows users to exist on diff platform
ex: Facebook users to gain access as if they are user in AWS.
→ IAM supports 2 types of Identity Federations

① Enterprise Identity Federation (SSO)
- SAML (microsoft AD)
- Custom Federation broker

② Web Identity Federation
- Amazon
- Facebook
- Google
- OpenID Connect (OIDC) 2.0
  ↓
  OIDC is an authentication layer on top of OAuth 2.0, an authorization Framework

SCP: To apply restrictions across multiple accounts of same org, used service control policy.

Security Token Service (STS)

A web service that enables you to request temporary limited privilege credentials for IAM users / federated user.

→ STS is a global service
all AWS STS requests go to a single endpoint at https://sts.amazonaws.com

→ An STS will return
- Access key ID
- Secret Access key
- Session Token
- Expiration

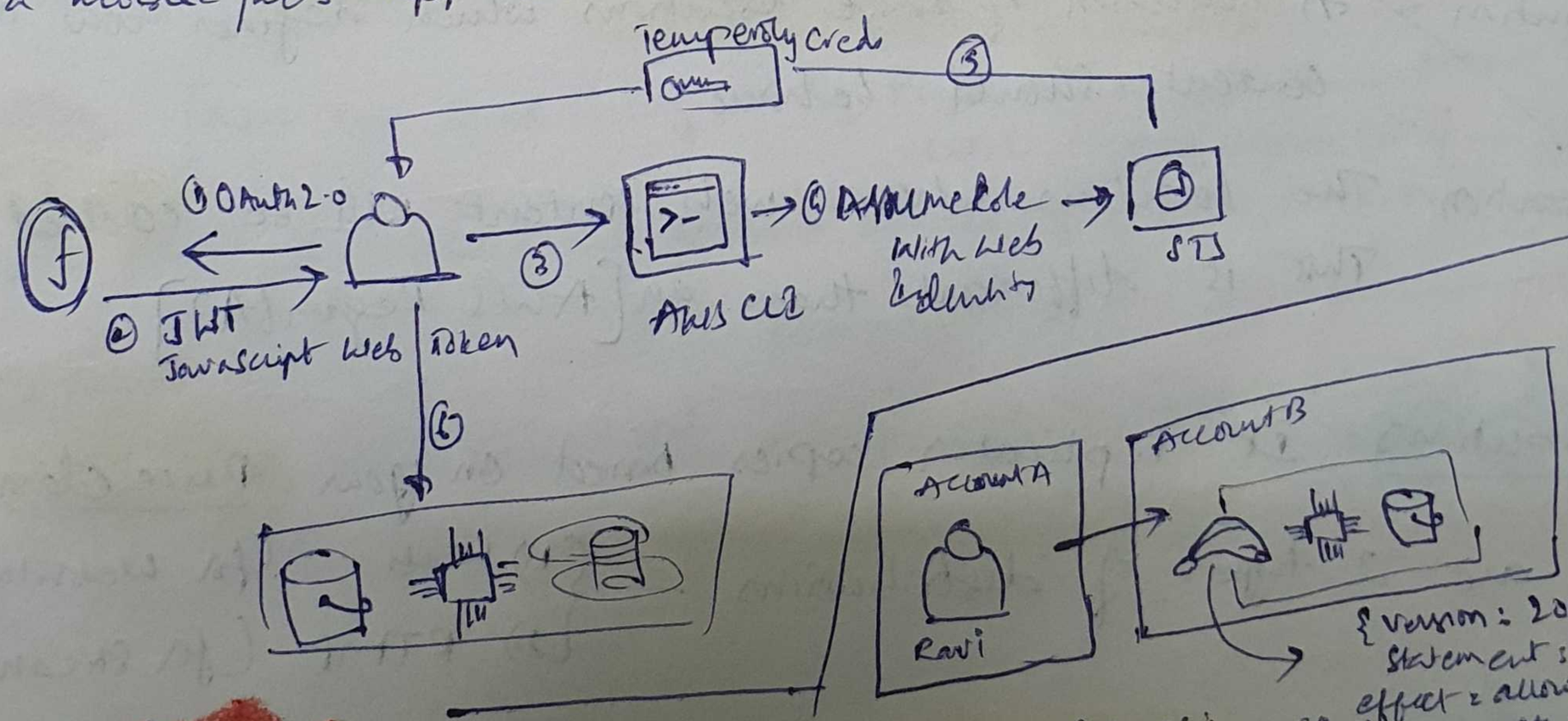
Same as programatic access keys but these are temporary

→ You can use the following API actions to obtain STS:

1. ~~Action~~ Assume Role → when you use Roles, IAM internally (or Cross Account access) use this to get STS tokens
2. Assume Role with SAML
3. Assume Role with Web Identity
4. Decode Authorization Message
5. Get Access key Info
6. Get Caller Identity
7. Get Federation Token
8. Get Session Token.

Assume Role with Web Identity

This action returns a set of temporary security creds for users who have been authenticated in a mobile/web app with a web Identity provider.



CROSS ACCOUNT ROLES:

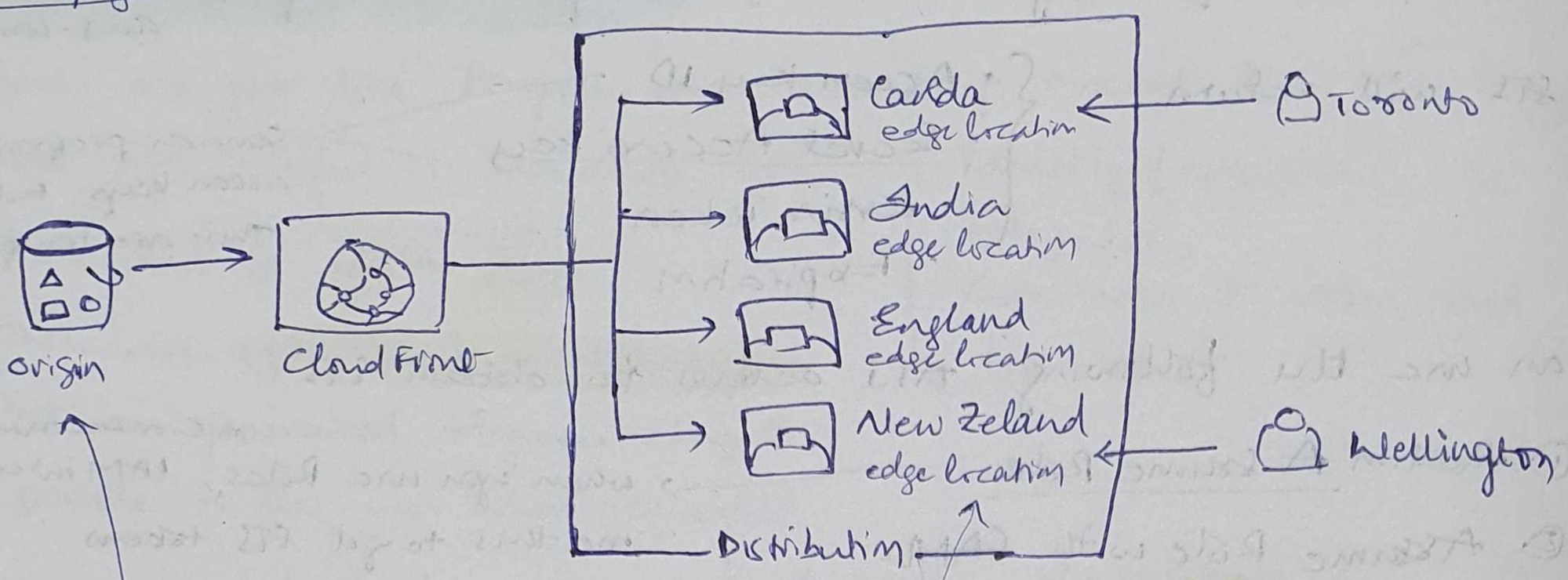
You can grant users from different AWS account access to resources in your account through a "Cross Account Role". This allows you to not to have to create them a user account within your system.

IAM demo: 8:52:08

CloudFront :- Content Distribution Network (CDN)

Creates cached copies of your website at various edge locations around the world

CDN: is a distributed n/w of servers which delivers web pages and content to users based on their geographical location, the origin of the webpage and a content delivery server



- Can be used to deliver an entire website including dynamic & streaming
- Requests for content are served from the nearest "Edge location" for the best possible performance

- when you use the root user to manage CF keypair you can have upto 2 active CF keypairs
- when you create a signer, the public key is with CF & private key is used to sign a portion of URL

Code Component:

Origin :- The location where all the original files are located.
For example an S3 bucket, EC2 instance, ELB & Route53.

Distribution :- A collection of Edge locations which defines how cached content should behave.

Edge location: The locations where web content will be cached.
This is different than an [AWS Region/AZ]

Distributions: It replicates copies based on your Price class

There are 2 types of distributions — (1.) Web - (for websites)
(2.) RTMP (for streaming media)

Behaviours: Redirect to HTTPs, Restrict HTTP methods (Only gets), Restrict Viewer Access, Set TTLs. time to ~~expire~~ live

Invalidations: You can manually invalidate cache on specific files via invalidation

Error Pages: You can serve up custom error pages eg: 404

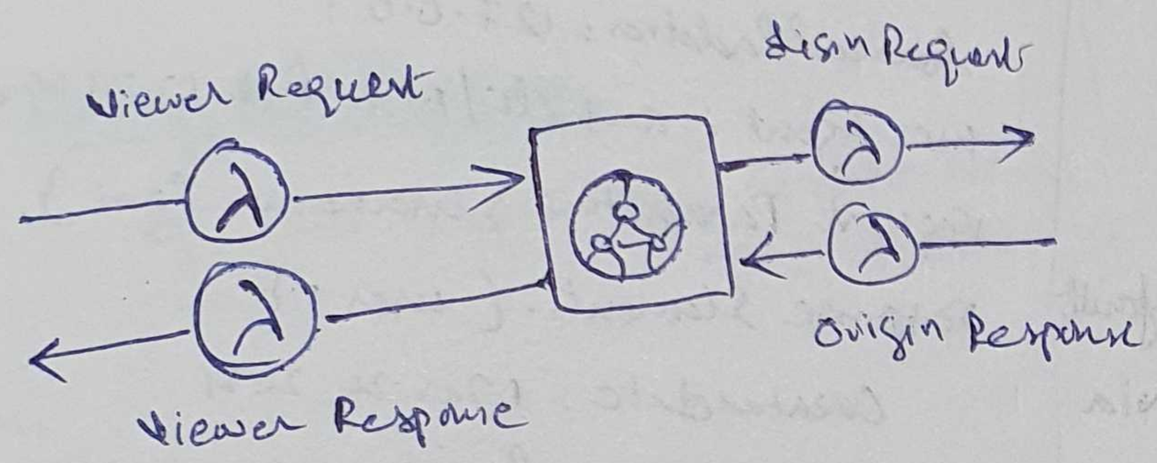
Restrictions: You can use Geo Restriction to blacklist or whitelist specific countries

Lambda @ Edge:

We use lambda @ Edge functions to override the behaviour of request & responses

Following are 4 lambda edge functions

1. Viewer request :- When CloudFront receives a request from a viewer
2. Origin request :- Before cloudfront forwards a request to the origin
3. Origin response :- When cloudfront receives a response from the origin
4. Viewer response :- Before cloudfront returns the response to the viewer



Lambda function Association

Cloud Front Event	Lambda Fun ARN	Include Body
Origin Request	arn:aws:lamb:...	☐
Viewer Response		☐

Protection

By default a distribution allows everyone to have access

OAI original Identity Access: A virtual user identity that will be used to give your CloudFront Distribution permission to fetch a private object

In order to use signed URLs / signed Cookies you need to have an OAI

Signed URLs: (Not the same thing as S3 Presigned URL)

A url with provided temporary access to cached objects

Restrict Viewer Access (use signed URLs/ signed Cookies)	⊙ Yes ○ No

Signed Cookies: A cookie which is passed along with the request to cloudfront. The advantage of using a cookie is you want to provide access to multiple restricted files eg: Video streaming

cloud front distribution demo: 9:11:09.

cheatsheet : 9:19:40

Cloud Trail:

- Log API calls between AWS services, when you need to know who to blame.

- AWS cloudTrail is a service that enables governance, compliance, operational auditing and risk auditing of your aws account

- AWS Cloud Trail is used to monitor API calls and Actions made on an AWS acc

- Easily identify which users and acc made the call to AWS eg:

  → Where source IP address
  → When event time
  → Who user, agent
  → What region, resources, action

```
{ Records: [{
    eventVersion : 1.0
    userIdentity : {
        type : IAMUser
        principalId , arn , accId ,
        accKey , userName
        },
    eventTime, eventSource, eventName,
    awsRegion,
    sourceIPAddress : 127.0.0.1
    User Agent : aws-cli/1.3.2 Python/2.7.5 Windows/7
    request Parameters : { Username : Ravi },
    response Elements : { user : {
        CreateDate : Mar 24 2021
        username : Ravi
        arn :
        path :
        userId :
        } }
}]}
```

- Cloud Trail is already logging by default and will collect logs for last 90 days via Event History.

- If you need more than 90 days you need to create a Trail.

- Trails are output to S3 and do not have GUI like Event History. To analyze a Trail you'd have to use 'Amazon Athena'

— options:

→ A trail can be set to log to all regions
→ A trail can be set to action all acc in an Organization
→ You can Encrypt your logs using Server Side Encryption via (SSE-KMS)
→ We can ensure the Integrity of our logs to see if they have been tampered we need to turn on log File Validation
→ Cloud Trail can deliver events to cloud Watch log.

Management Events

Tracks management operations.

Turned on by default.

Can't turn off

ex.

- configuring security
  eg: IAM attachRolePolicy API operation
- Registering devices
  eg: Amazon EC2 CreateDefaultVPC api operation
- Configuring rules for routing data
  eg: Amazon EC2 CreateSubnet API operation
- Setting up logging
  eg: AWS CloudTrail CreateTrail API operation

Data Events

Tracks specific operations for specific AWS services. data events are high volume logging and will result in additional charges.

Turned off by default.

Two services that can be tracked is S3 and lambda.

So it would track actions such as GetObject, DeleteObject, PutObject.

Demo
Checkpoint: 9:34:30.

CloudFormation: (9:35:43)

A templating language that defines AWS resources to be provisioned. automating the creation of resource via code. Infrastructure as a Code.

IaC: Infrastructure as a code, the process of managing and provisioning computer data centers (eg: AWS) through machine-readable definition files (eg: YAML, JSON files) rather than physical hardware configuration or interactive configuration tools (stop doing manual configuration!)

→ CloudFormation can be written in 2 formats { JSON, YAML }

Quick Start: aws quick start are collection of pre-built CloudFormation templates

Stack Updates: when you need to make a change to stack, instead of deleting and recreating the stack. You can modify the CloudFormation template and push a stack update and CloudFormation will intelligently change & delete recreate resources.

① direct update: You directly update when you want to quickly deploy

② executing change set: will give a change to see diff and avoid unintentional changes.

Template Anatomy:-

```
AWSTemplateFormatVersion: 2010-09-09
Description: >-
  Launch an EC2 instance running apache
  and expose default page to the internet
  Hardcoded to work only with us-E1
Parameters:
  InstanceType:
    Description: Webserver EC2 instance type
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.nano
      - t2.micro
Resources:
  WebServer:
    Type: 'AWS::EC2::Instance'
    Properties:
      Tags:
        -
          Key: Name
          Value: Apache Default Webserver
      InstanceType: !Ref InstanceType
      ImageId: 'ami-0b898040...'
      SecurityGroupIds:
        - !GetAtt SecurityGroup.GroupId
      UserData:
        'Fn::Base64':
          !Sub |
            su ec2-user
            sudo yum install httpd -y
            sudo service httpd start
  SecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable internet access
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: 0.0.0.0/0
Outputs:
  PublicIP:
    Value: !Get WebServer.PublicIP
```

Metadata : additional info about the template

Description: a description of what this template is supposed to do

Parameters: Values to pass to your template at runtime

→ doesn't allow conditions here

mappings:- a lookup table. maps keys to values so you change your values to something else

Conditions: whether resources are created & properties are assigned.

Transform: applies macros (like applying some which change the anatomy to be custom)

Resources: a resource you want to create
eg: IAM Role, EC2 Instance, λ, RDS

Output :- values that returned
eg: an ip-address of new server created

cloudformation templates requires you to at least list one resource

Expected o/p values in CF must have unique names within a Single Region

→ When you submit an update in Cloud Formation, it updates differently depending on the state of your resources and circumstances

Updates with No Interruption:

① updates the resources without disrupting operations
② and without changing the resource's physical ID

Updates with Some Interruption:

① updates the resources with some interruption
② and retains the physical ID

Replacement:

① recreates the resource during an update
② also generates a new physical ID

Prevent Stack Updates

You may want certain resources not to be updated.
Preventing a stack-update could be to prevent data loss | interruption to sema

→ Create a StackPolicy

A stack policy is a JSON document that defines the update actions that can be performed on designated resources

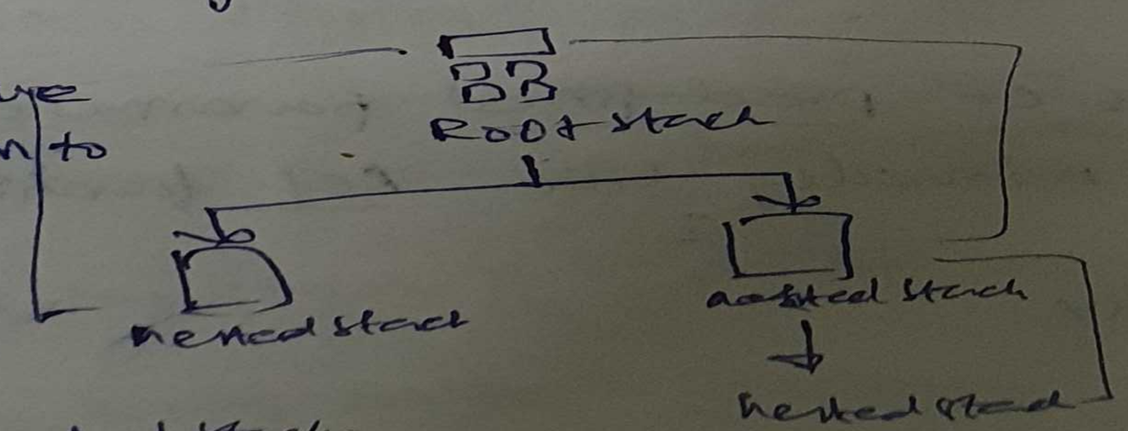
```
{ Statement : [
  { Effect : Deny
    Action : Update:Replace
    Principal : * ,
    Condition : {
      StringEquals : {
        ResourceType : [AWS::RDS::DBInstance]
      } } }
  { Effect : Allow
    Action : Update:* ,
    Principle : * , Resource : *
  } ] }
```

Nested Stacks

Allows you to reference CFN templates inside of another CFN templates:

① Create modular templates (reusability)
② assume large templates (reduce complexity)

Parent stack have immediate access to child stacks



The root stack is accessible by all nested stacks

```
AWSTemplateFormatVersion: 2010-09-09
Resources:
  myStack:
    Type: AWS::CloudFormation::Stack
    Properties:
      TemplateURL: https://S3:.....
      TimeoutInMinutes: 60
```

The nested template needs to be inside an

Drift Detection

→ Drift is when your stack's actual configuration differs (has drifted) by what CF expects.

→ When developers start making manual adhoc changes to the stack (most common example is deleting resources) instead of changing configuration by updating and relaunching the CloudFormation template.

→ CloudFormation can detect drift and tell you if resources deleted & have been modified.

→ When detecting drift on a stack, CF doesn't detect drift on any nested stacks. that belong to that stack. Instead, you can initiate a drift detection operation directly on the nested stack.

•) possible drift status
- DELETED – the resource has been deleted
- MODIFIED – there has been a change to the resource
- NOT_CHECKED – CF hasn't checked resource
- IN_SYNC – everything is as expected

Rollbacks:

When you create, update & destroy a stack you could encounter an error.

eg. Your CF template has a syntax error.

: Your stack is trying to delete a resource which no longer exists

CF will attempt to Rollback in case of an error put your stack back to its previous state.

Rollbacks can fail. Sometimes you'll have to investigate and change resource configuration & possibly need to reach out to paid AWS support to resolve the failed rollback

→ When a rollback is in progress, you'll see ROLLBACK_IN_PROGRESS

successful rollback ← UPDATE_ROLLBACK_COMPLETE

failed rollback ← update_rollback_failed

Pseudo Parameters

There are AWS CF predefined parameters. We don't define them in our template. We use them normally like Ref function

AWS::Partition

AWS::Region

AWS::StackId

AWS::StackName

AWS::URLSuffix

Outputs:
MyStackRegion
Value:
!Ref AWS::Region

Resource Attributes:

→ Creation Policy: prevent its status from reaching create complete until AWS CF receives a specified # of success signals or the timeout period is exceeded.

→ Deletion Policy: Reserve & (in some cases) backup a resource when its stack is deleted [Delete, Retain or Snapshot]

→ Update Policy: How to handle an update for ASG, ElasticCache, Domain & Lambda Alias.

→ Update Replace Policy: To retain & (in some cases) backup the existing physical instance of a resource when it is replaced during a stack update operation [Delete, Retain & Snapshot]

→ DependsOn:- That resource is created only after the creation of the resource specified in the DependsOn attribute

Intrinsic Functions:

Use intrinsic functions in your templates to assign values to properties that are not available until runtime.

Fn::Base64 — Base64 representation of the i/p string
Fn::Cidr — an array of CIDR address blocks
Fn::FindInMap — return value of keys based on mapping section
+ Conditional Functions → Fn::And, Fn::Equals, Fn::If, Fn::Not, Fn::Or
Fn::Transform — a macro to perform custom processing on part of a stack template
Fn::GetAZs — an array that lists AZ for a specified region
Fn::ImportValue — the value of an o/p exported by another stack
Fn::Join — appends a set of values into a single value, separated by delimiter.
Fn::Select — returns a single object from a list of objects by index
Fn::Split — splits the string into a list of string values
Fn::Sub — substitute variables in an i/p string with values you specify
Fn::GetAtt — value of an attribute from a resource in the template.
Ref — the specified parameter or resource

Wait Condition: 2 types of wait

① To coordinate stack resource creation with configuration actions that are external to the stack creation

② To track the status of a configuration process

• Wait Condition is very similar to CreationPolicy
aws recommends creationPolicy for EC2 & ASG

Wait Handle:
Type: AWS::CF::WaitConditionHandle
Wait Condition:
Type: AWS::CF::WaitCondition
DependsOn: "WebServerGroup"
Properties:
Handle:
Ref: "WaitHandle"
Timeout: "300"
Count:
Ref: "WebServerCapacity".

Demo: 10:02:23

Cloud development Kit (CDK) : (10:41:56)

- Write Infrastructure as Code (IaC) using an imperative paradigm with your favourite language

→ Transpiler: A transpiler turns one source code into another.

CDK transpiles into CF templates

Imperative Infrastructure	Declarative Infrastructure
Its implicit what resources will be created in the end state.	It's explicit what resources will be created in the end state.
More Flexible, less Certain, Write less	Less Flexible, Very Certain, Write more
eg: I want EC_2, you fill in all the other details	Eg. I want EC_2, I have to tell you exactly every detail
CDK is Imperative	CF is declarative.

- CDK was first available in TypeScript, now available in JS, Py, Java, .NET.
- CDK's API may yet to have implemented APIs to every AWS resource available in CF
- TypeScript would likely have the most available APIs implemented and another languages would wait to catch up to TypeScript

Serverless Application Model (SAM)

An extension of CF that lets you define serverless applications.

- SAM is both AWS CLI tool and a CF Macro which makes it effortless to define and deploy serverless applications

Macro: A macro allows you to change the rules on how code works, allowing you to embed a language with a language. Macro serves to make code more humanreadable & allow to write less code.

Creating a language within another launch is called DSL. Domain Specific Language

By using macros you are creating a DSL.

CF allows you to specify macros through the Transform attribute.

This is how SAM is used.

Transform: 'AWS::Serverless-2016-10-31'

AWS::Serverless::Function
AWS::Serverless::Api
AWS::Serverless::SimpleTable

SAM CLI Commands:

→ Sam has a CLI which makes it easy to run, package and deploy Serverless applications a lambdas.

sam build - prepares λ src code to be deployed by packaging it for upload.

sam deploy - upload λ package code (artifact) and then deploy

sam init - Initialize a new serverless project, creating all the folders started to get.

sam local generate-event - Generates sample payloads from diff event sources, such as S3, amazon API, Gateway and SNS

sam local invoke - run a single λ locally

sam local start-api - run your serverless app locally for quick development and testing

sam local start-lambda - run a single lambda locally (unchearies)

sam logs - fetches logs that are generated by your lambda function.

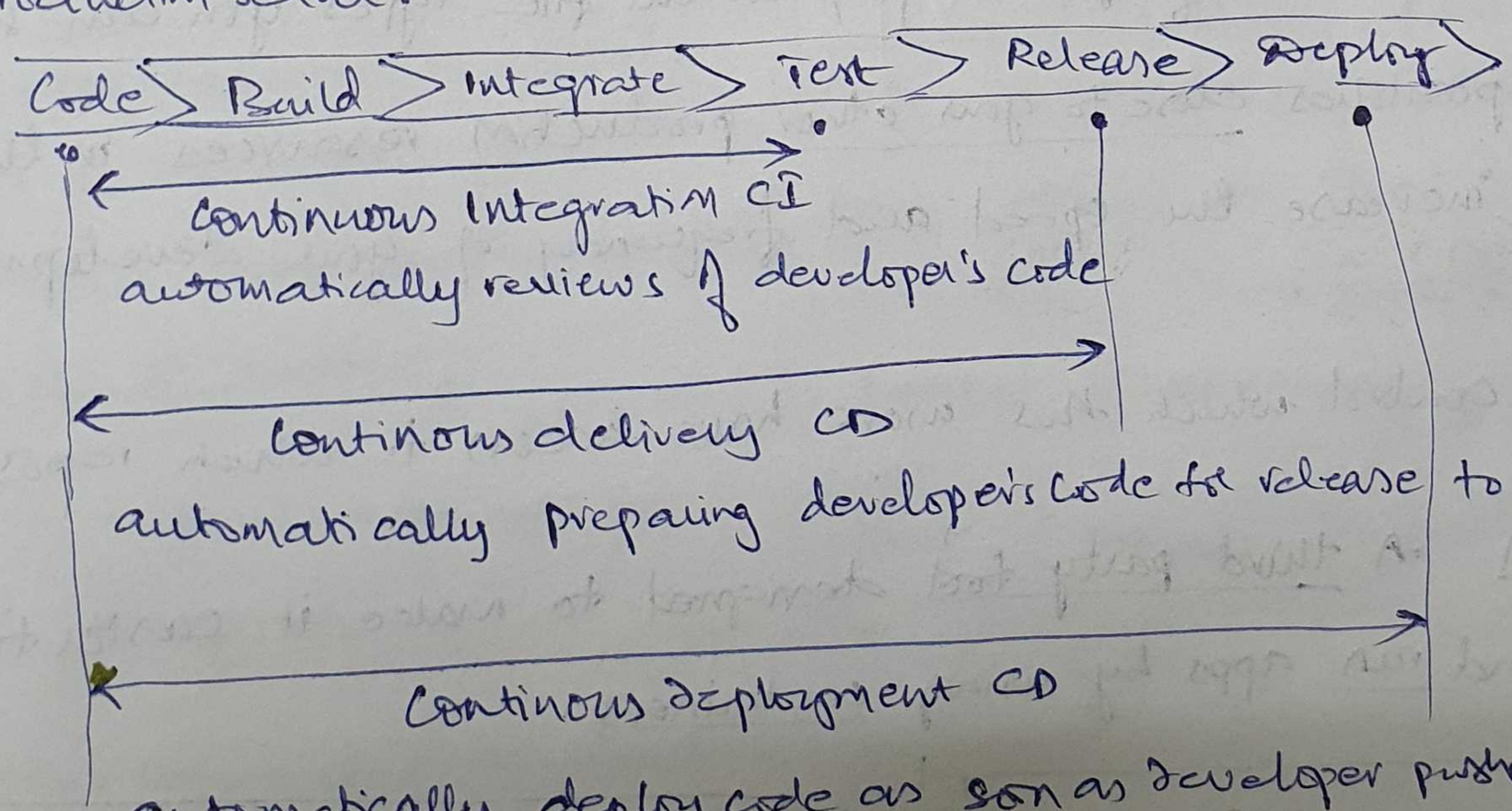
sam package - Packages an AWS SAM application, creates a zip and uploads it to S3

sam publish - Publishes your packaged SAM app to AWS Serverless App Repository

sam validate - Validates an AWS SAM template (great for finding syntax error)

CI/CD : [10:50:05]

→ automated methodologies that prepare, test, deliver & deploy code onto a production server.



CI : the practice of automating the integration of codechanges from multiple contributors into a single software project.

CD: (Continous delivery) : The practice of automating the preparation of code to be released to production & staging branches. You are preparing the codebase for ~~prod~~ deployment. Deployment is still a manual process

CD: (Continous deployment) : Same as continous delivery but automatically deploys changes to Production.

Code Commit: [10:58:35] [IAM usr/pswd can't be used to access CodeCommit → ✓ SSH, Git creds, AWS accounts] Dec

A fully-managed source control service that hosts secure Git-based repos.

Version Control System / Source Control: A system that records to a file a set of files over time so that you can recall specific versions later. [CVS, SVN, mercurial, Git]

Git:) A distributed version-control system.

) Its goals include speed, data-integrity and support for distributed, non-linear workflows.

→ Codecommit is a service which lets you store your Git repos in cloud. Developers can push and pull code from the cloud repo and it has tools resolve conflicts.

Key Features:

) In scope with many compliance programs eg: HIPPA.

) Repositories are encrypted at rest as well as in transit

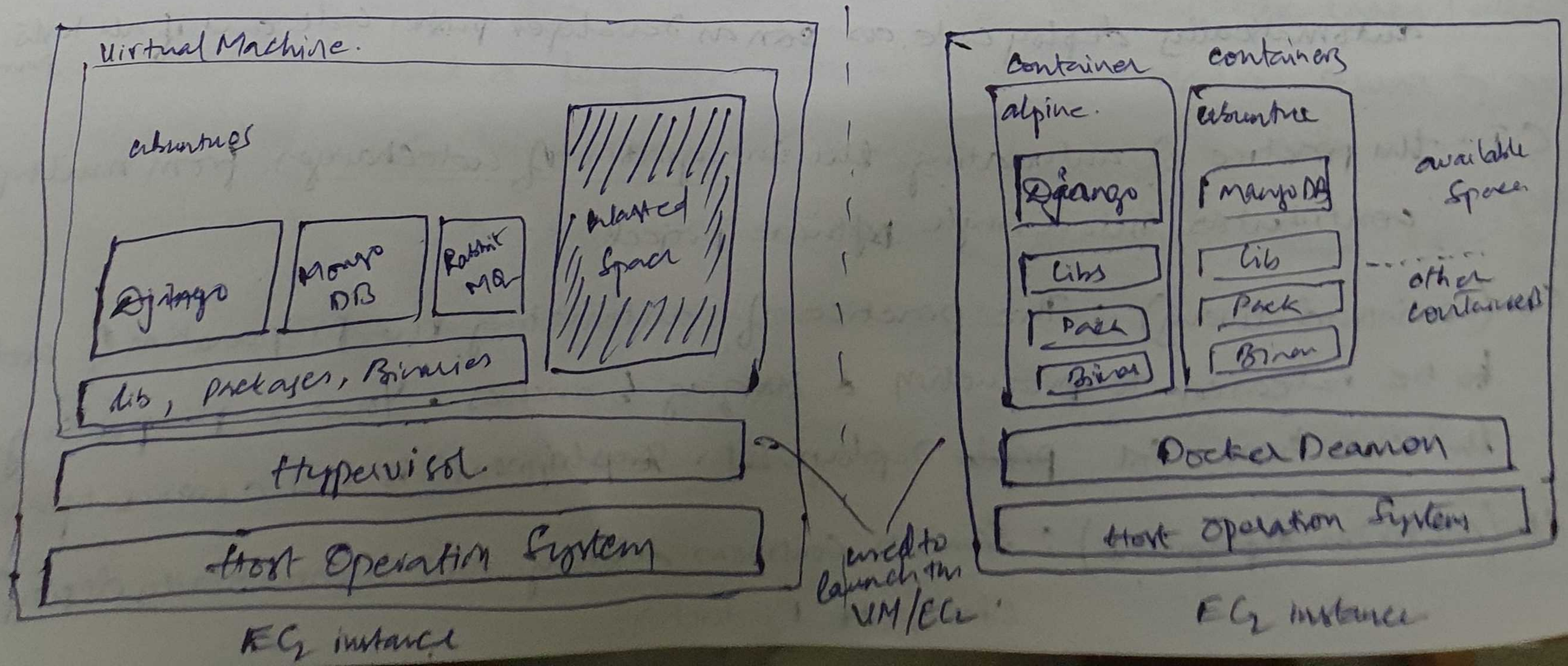
) Can handle repos with large number of files / branches, large file size and lengthy revision histories.

) No limit on the size of your repo / on the file types you can store

) Keeps your repositories close to your other production resources in the AWS, which helps increase the speed and frequency of your development lifecycle.

) Use IAM to control which AWS users have access to which repos.

Docker: [11:02:24] A third party tool designed to make it easier to create, deploy and run apps by using containers.



- VMs do not make best use of space. Apps are not isolated which, ~~that~~ could cause config conflicts, security problems & resource hogging

- Containers allow you to run multiple apps which are virtually isolated from each other.
- launch new containers and configure OS dependencies per container.

Anatomy of a Dockerfile:-

A text document that contains all the commands a user could call on the command line to assemble an image.

FROM: use another docker image as basis for this Dockerfile
else the code will be very large

RUN: execute any kind of bash command. runs on layers on top of current image

WORKDIR: changes default folder for future commands

ENTRY POINT: command executed when the container is started. Can't be overwritten

EXPOSE: listens on the specified n/w ports at runtime.

CMD default arguments for entrypoints.

ex: setting up ruby on rails app, with nodejs / postgres

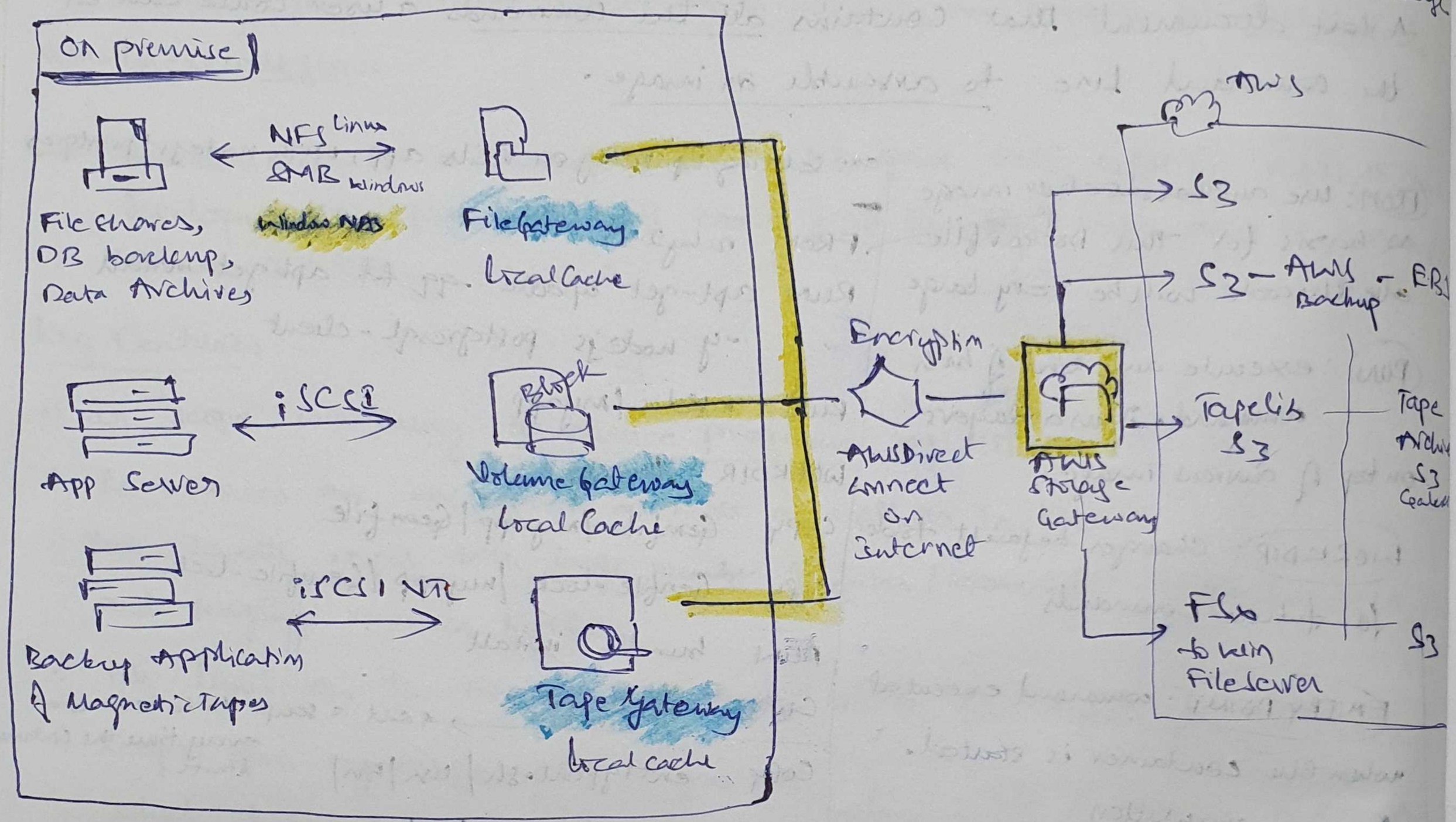
```
FROM ruby2.5
RUN apt-get update -qq && apt-get install
    -y nodejs postgresql-client
RUN mkdir /myapp
WORKDIR /myapp
COPY Gemfile /myapp/Gemfile
COPY Gemfile.lock /myapp/Gemfile.lock
RUN bundle install
COPY ./myapp
# add a script to be executed every time the container starts
COPY entrypoint.sh /usr/bin/
RUN chmod +x /usr/
ENTRYPOINT ["entrypoint.sh"]
EXPOSE 3000
# start the main process
CMD ["rails", "server", "-b", "0.0.0.0"]
```

Docker commands:

Docker build: build an image
Docker ps: list of containers (only running containers are shown here)
Docker images: list of images
Docker run: run a command in a new container
Docker push: push an image to a repo
Docker pull: pull an image from a repo

AWS Storage Gateway:

- provide onpremises apps with access to virtually unlimited cloud storage
- → Low latency data access
- → onpremise access to cloud without disrupting business/workflows
- → capabilities like encryption, audit, logging and Write once Read many (WORM) storage



Use cases:-

① Fill your data lake : store data coming from specialized on premise instruments in Amazon S3 and use AWS for analysis

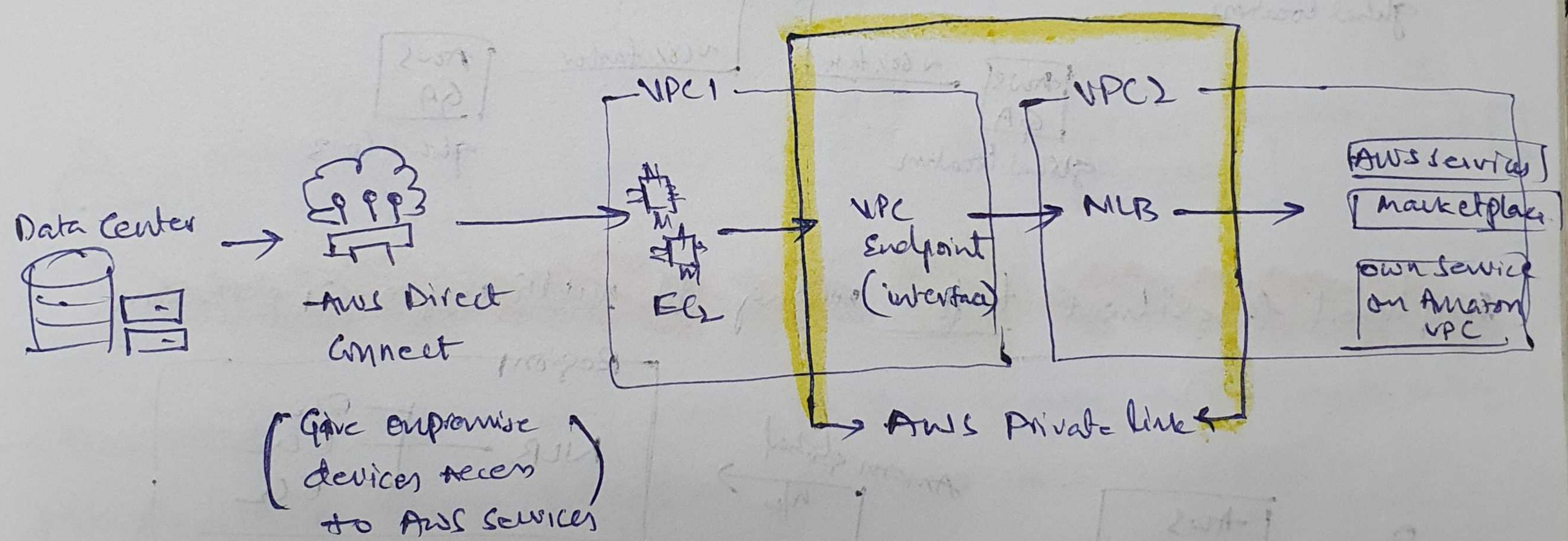
② Modernize interactive file sharing :- Transform your on premise user/group file shares to a hybrid cloud arch for simplified management and cost reduction

③ Backup data to the cloud :- provide cloud-based backup for on-prem files, DB apps for low cost, virtually unlimited scale.

AWS Private link:

- → provides connectivity between VPCs, AWS services, on premise
- → doesn't expose traffic to the public internet
- → easy to connect services across diff account & VPCs to simplify n/w arch
- → Interface VPC endpoints – powered by AWS private link
  Connect you to services hosted by AWS / AWS marketplace
- → Gateway load balancer endpoints – powered by AWS private link.
  security & performance to your virtual n/w app / custom traffic inspection logic.

Benifits:-
1. Secure traffic
2. Simplify n/w architecture : diff acc's, VPCs, (CIDRS, No Firewalls, RT, Peering VPC)
3. Accelerate cloud migration : onpremise → AWS services



To use AWS private link, create an interface VPC endpoint for a service in VPC, This creates Elastic n/w interface (ENI) in your subnet with a private IP add that servers as an entry point for traffic destined to the service. Service endpoints available over AWS private link will appear as ENIs with private IPs in your VPCs

AWS Global Accelerator:-

→ n/w'ing service that improves perf of your users traffic by 60% using amazon web services' global n/w infrastructure. [using 90+ global edge locations]

→ optimizes the path to your applications

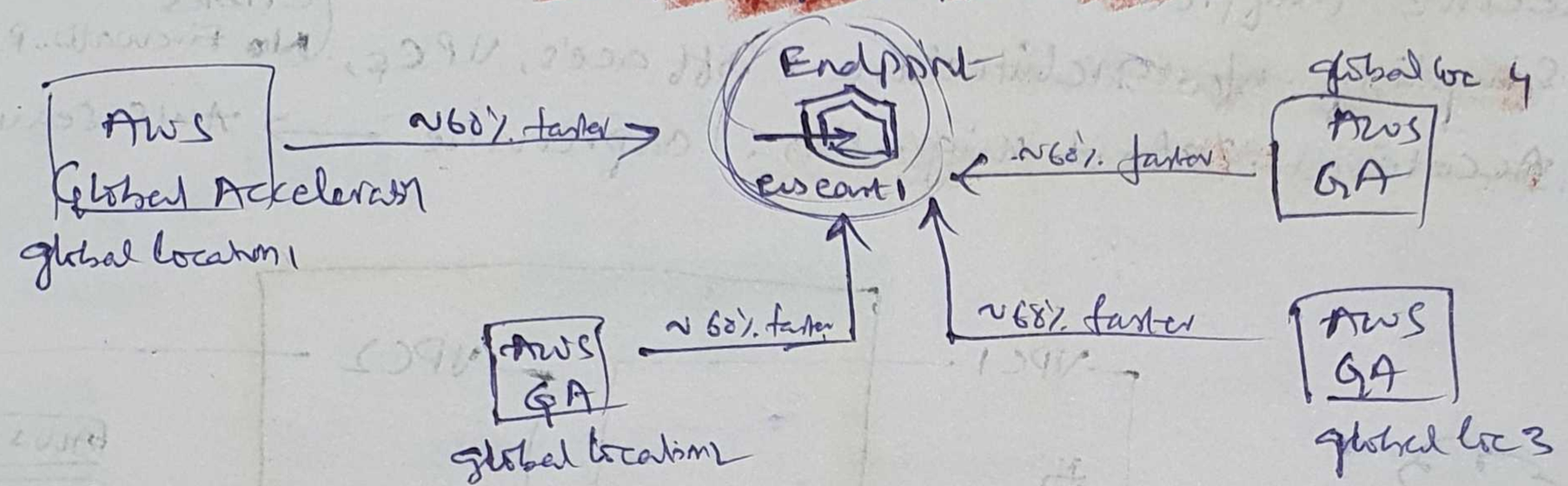
→ provides 2 global static public IPs for your applications in multiple regions for ↑ Availability

→ backend add/removes — appendpoints — { ALB, NLB, EC2 / Elastic IPs

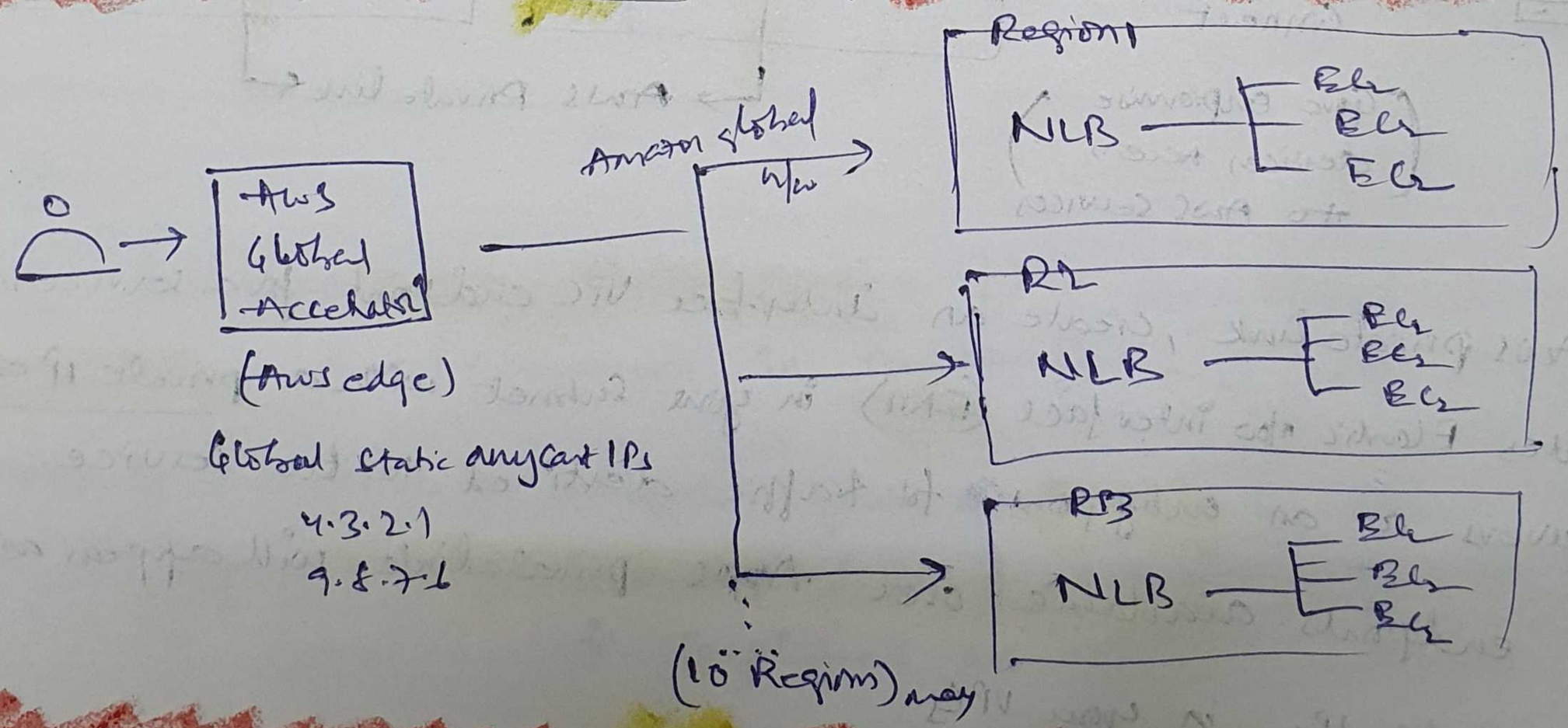
→ automatically routes to your nearest healthy available endpoint to mitigate endpoint failure.

use cases:

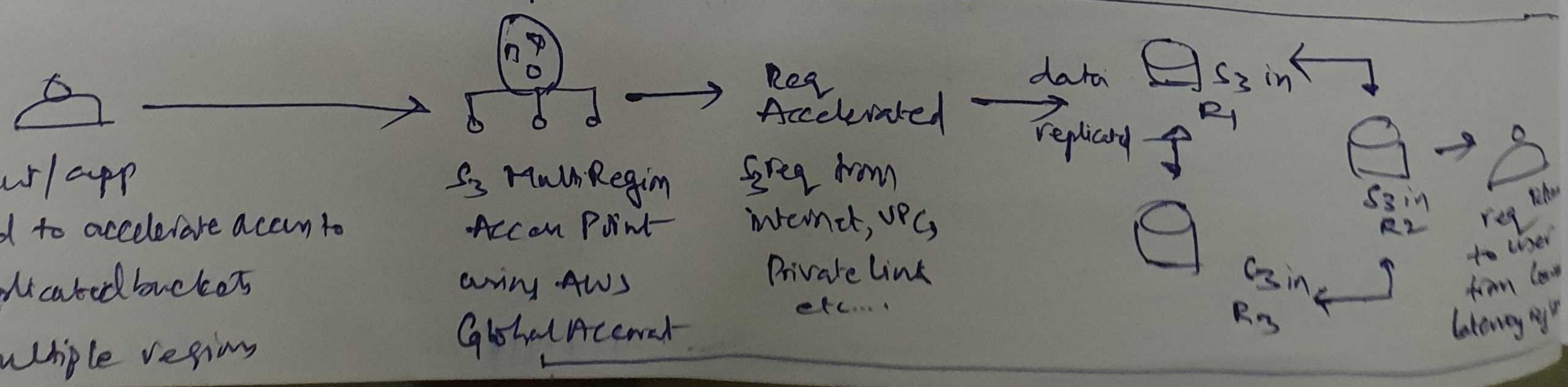
Single region application



Simplified & resilient traffic routing for multi region applications

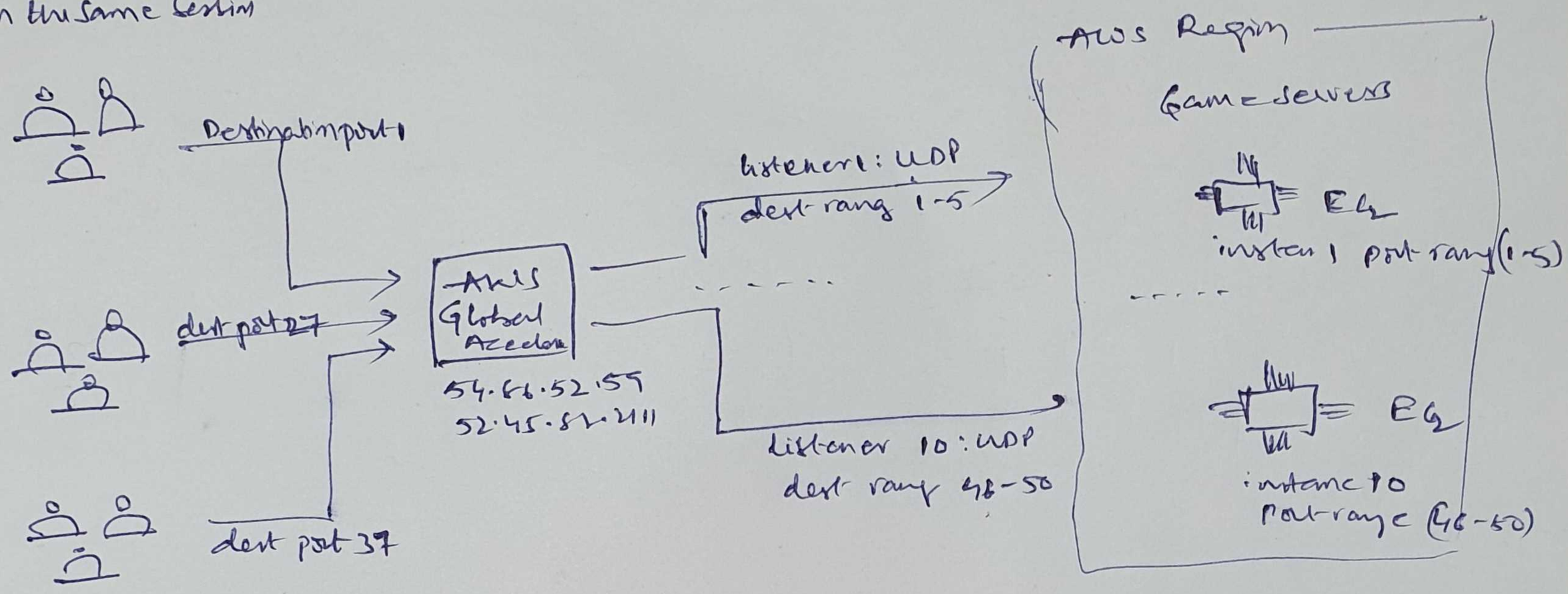


Accelerated & simplified storage for multi region applications

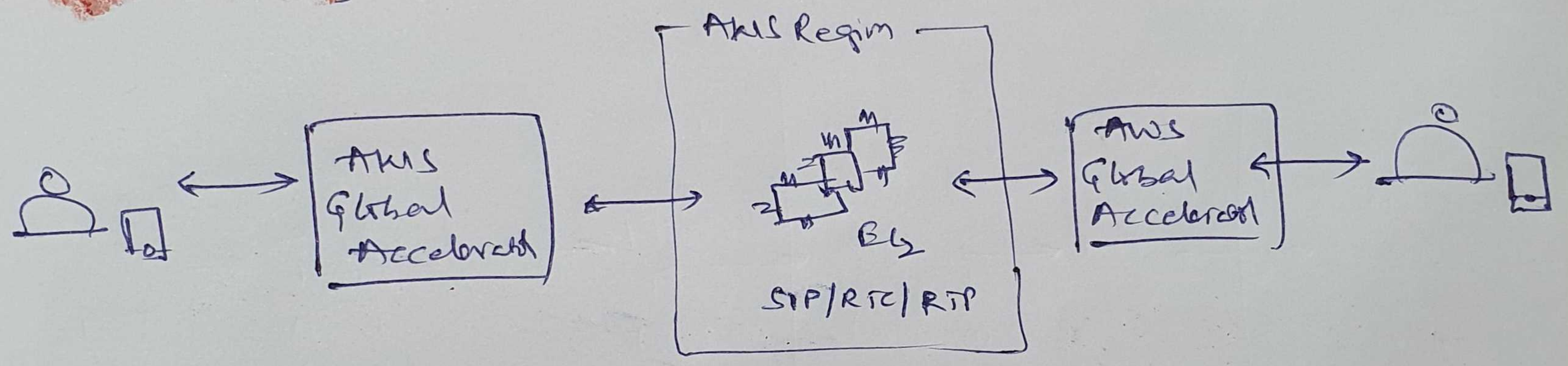


Enhanced performance for gaming:

Gamers attacked on the same Session



Running real time communication as a service & voice over IP on AWS RTCs



→ AWS global accelerator allows along the AWS network to run application protocols like Session Initiation; Real time and Web RTC.

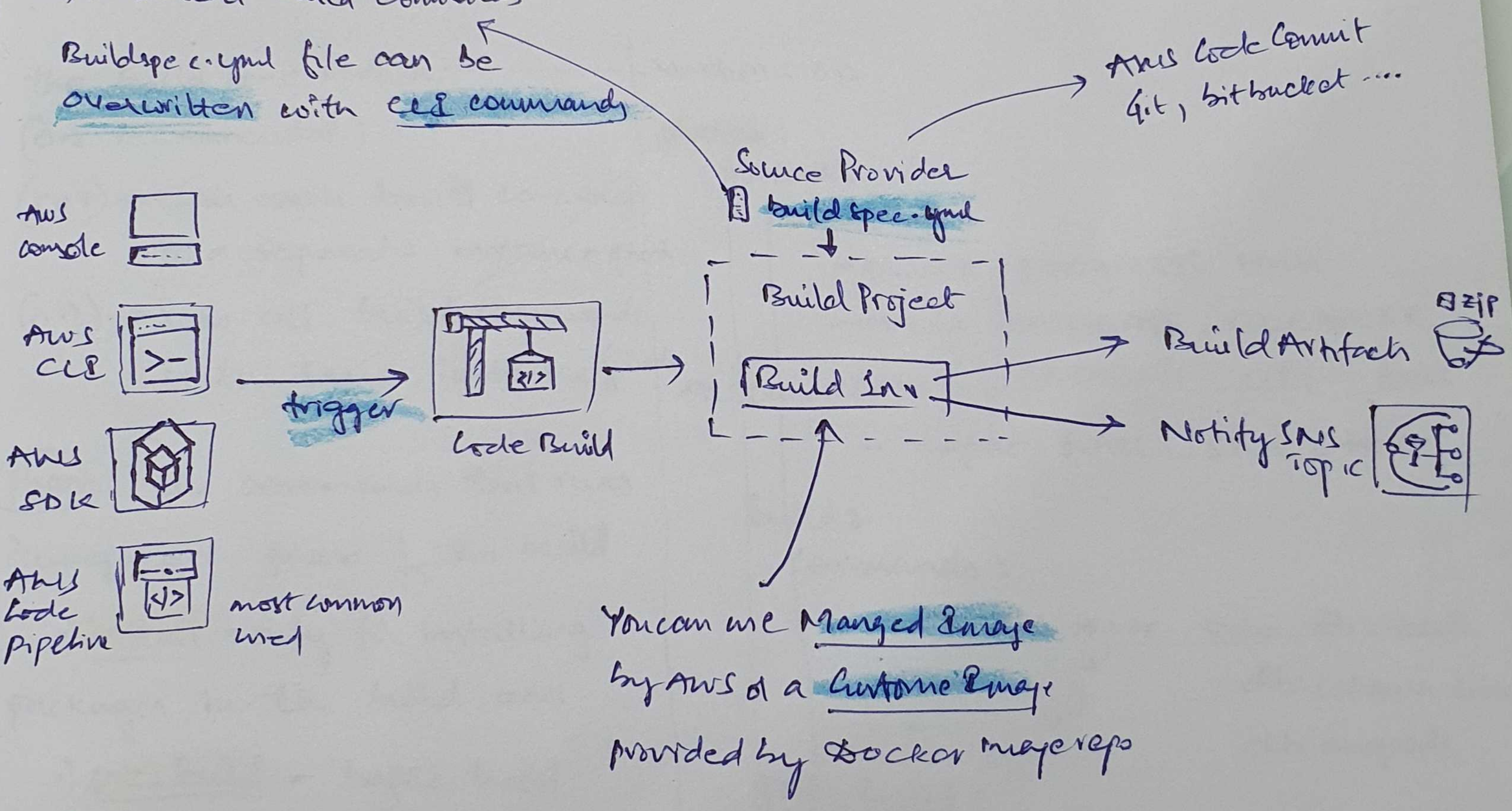
Code Build: [11:10:16]

- A fully managed build pipeline to create temporary server to build and test code projects.
- compiles your source code, runs unit tests, and produces artifacts that are ready to deploy.
- eliminates the need to provision, manage and scale your own build servers.
- provides prepackaged build envs for popular programming languages and build tools such as Apache Maven, Gradle & more
- You can also customize build envs to use your build tools
- Scales automatically to meet peak build requests

Work Flow

2 Option
- use a buildspec file
- Insert build commands

Buildspec.yml file can be overwritten with cli commands



Build Environments:

Docker images managed by Code Build. Check the images to see what comes pre-installed.

platform:
- Amazon Linux 2
- Ubuntu 18.04
- Windows Server Core 2016

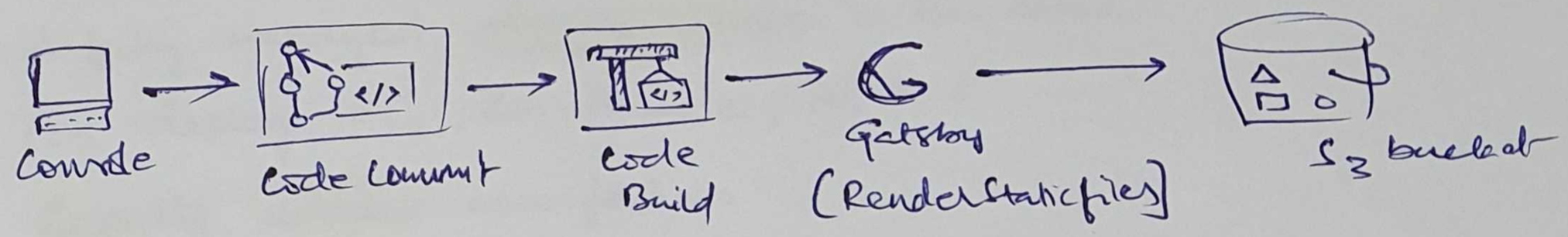
Environment Image
- Managed Image
- Custome Image

OS:-
- Amazon Linux 2
- Ubuntu
- Windows Server

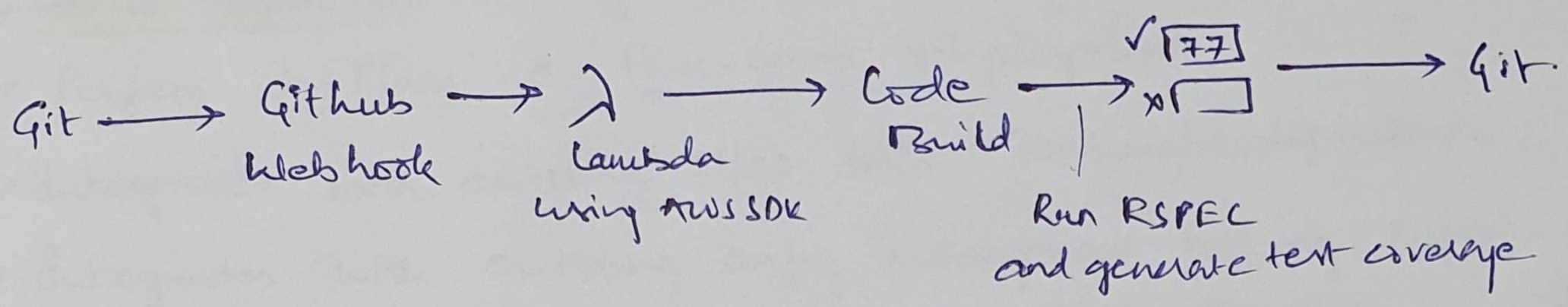
Use cases:

→ Java Application Management

1) Generate out static Pages from a JAM stack



2) Run test Code and report test coverage



Build spec.yml :-

This provides the build instructions. this need to be root folder of your project

the build spec version.
(0.2 recommended)
(0.1) → runs each build command in a seperate instance of bash
(0.2) → runs all build commands in the same instance of bash

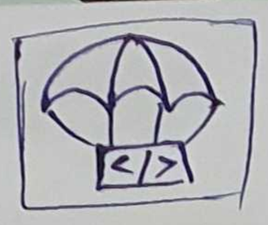
phase: the commands that runs during each phase of the build

- install : only for installing packages in the build env
- pre_build - before build
- build - during build
- post_build - after build

```
version : 0.2
phases :
  install :
    commands :
      - service postgresql start
      - export DATABASE_CLEANER_ALLOW_REMOTE_DATABASE_URL = true
      - export RAILS_ENV = test
  build :
    commands :
      - bundle exec rake db:create
      - db:schema:load
      - db:migrate
  post_build :
    commands :
      - bundle exec rspec
      - cucumber
```

artifacts: where we can find the build ~~artifacts~~ o/p and how to prepare it for uploading to the S3 o/p bucket.

Code Deploy:



A fully managed deploy pipeline to deploy to staging / prod envs.

- → A fully managed deploy service in the cloud.
- → Can deploy EC2, On-Premise, λ, ECS
- → Rapidly release new features
- → Update AWS λ function versions.
- → Avoid downtime during application deployment.
- → Perform In-Place & Blue-Green deployment
- → Integrates with existing CI/CD tools: Jenkins/ Codepipeline.
- → Integrates with existing config managment tool eg: Puppet, Chef, Ansible.

Core Components:

Application: the unique identifier for the application being deployed. Ensures the correct combination of revision, deployment config and groups are referenced during deployment

Deployment Groups: A set of EC2 / λ funs where the new version is deployed to

Deployment: This is the process and components used to apply a new app version

Deployment Configuration: A set of deployment rules used during deployment. with success / failure conditions included

App Spec File: Contains the deployment actions that CodeDeploy should execute during deployment.

Revision: This is everything required to deploy a new version. App spec file, app file, config files, executables

In Place deployment:

Choose how to deploy your app
✓ In place ○ Blue/green

↓

Environment configuration
- ☐ Amazon EC2 Auto Scaling Group
- ☐ Amazon EC2 Instances
- ☐ On premise instances

- The app on each instance in the deployment group is stopped.
- The latest app version is installed, new version of the app is started & validated
- You can use a load balancer so that each instance is deregistered during its deployment and then restored to service after the deployment is complete.
- Only deployment that use EC2/On premise can use In place

No λ func

Blue/Green deployments:

Choose how to deploy your app
- Inplace ✓ Blue-green

Environment configuration
- Automatically copy amazon EC2 auto scaling group
- Manually provision instances

- Instances are provisioned for the replacement environment (ASG) & app required instances
- The latest app revision is installed on the replacement instances
- An optional wait time occurs for activities such as application testing and system verification
- Instances in the replacement environment are registered with an ELB, causing traffic to be rerouted to them.
- Instances in the original environment are deregistered and can be terminated & kept running for other uses.

App spec.yml - responsible for where code to be installed how to get the application running.

Choose one of linux/Win

where .zip file of your code is present

```
version: 0.0
os: linux
files:
  - sources: /
    destination: /home/ec2-user/app
permissions:
  - objects: /home/ec2-user/app
    owner: ec2-user
    group: ec2-user
hooks:
  ApplicationStop:
    - location: aws/deployment/.... /a.sh
      timeout: 20
      runas: ec2-user
  BeforeInstall:
    - {location:
       timeout: runas:
  AfterInstall:
    - {   - {   - {
  ApplicationStart:
    - {
```

Commands which should stop the application

Code deploy Agent: You will need to install the codedeploy agent to the EC2 instance can report back to Code deploy.

Code deploy Service Role: You may need to create a codedeploy service role based on your deployment strategy

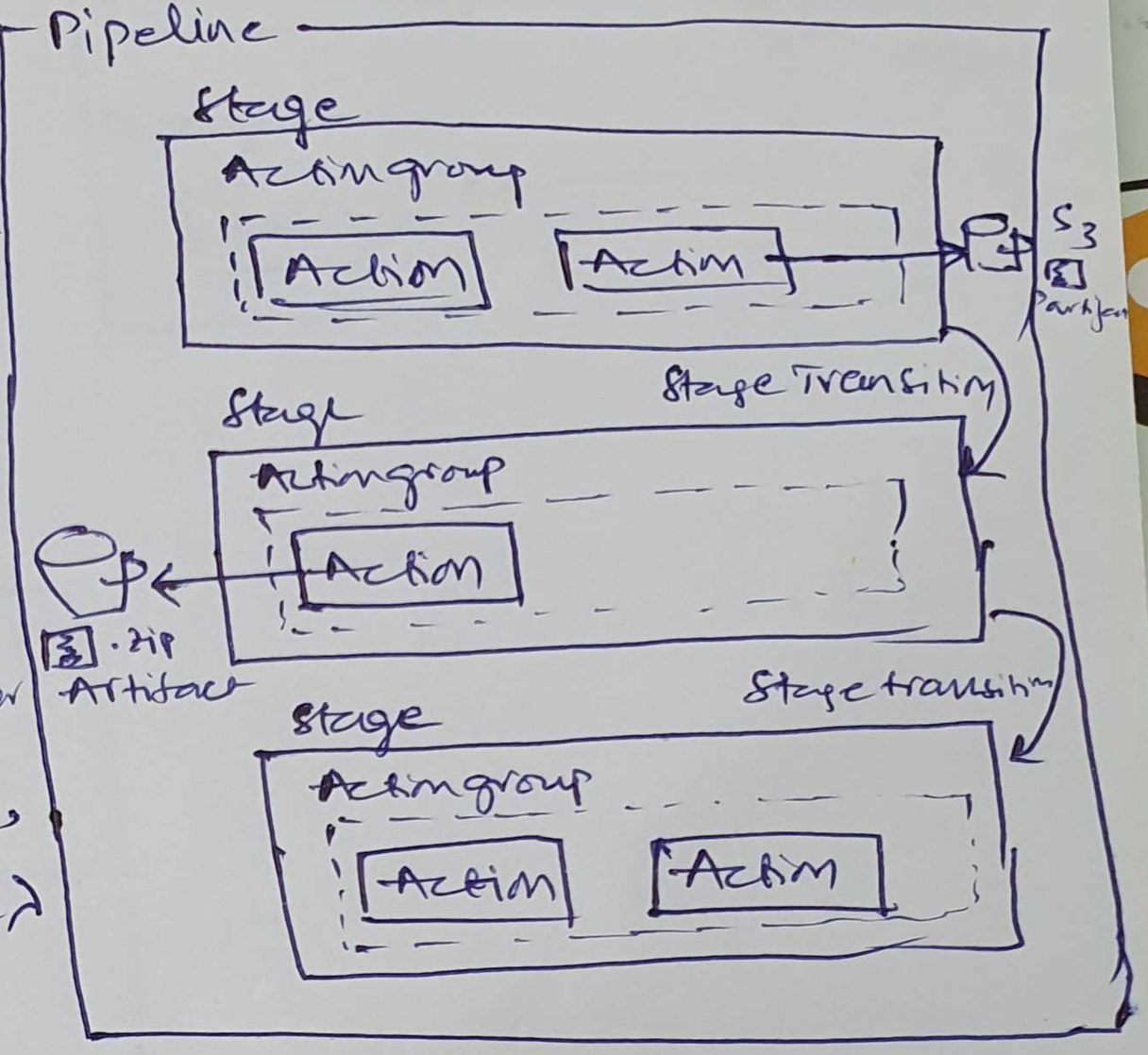
Code Pipeline: A fully managed CI/CD pipeline to setup automatic deployments

Pipeline:- is what encompasses all components

Stage:- represents a step in the pipeline the most common steps would be Source > Build > Deploy.
You can add many stages, so use your creativity.

Action Group:- groups several actions together

Action:- does something eg: pull scr code, build & run tests, deploy to prod, invoke / invoke CF template



Artifact:- is a zip containing files outputted by an action and stored in S3. An artifact can be passed to any future actions and some actions require to i/p an artifact and can optional o/p an artifact.

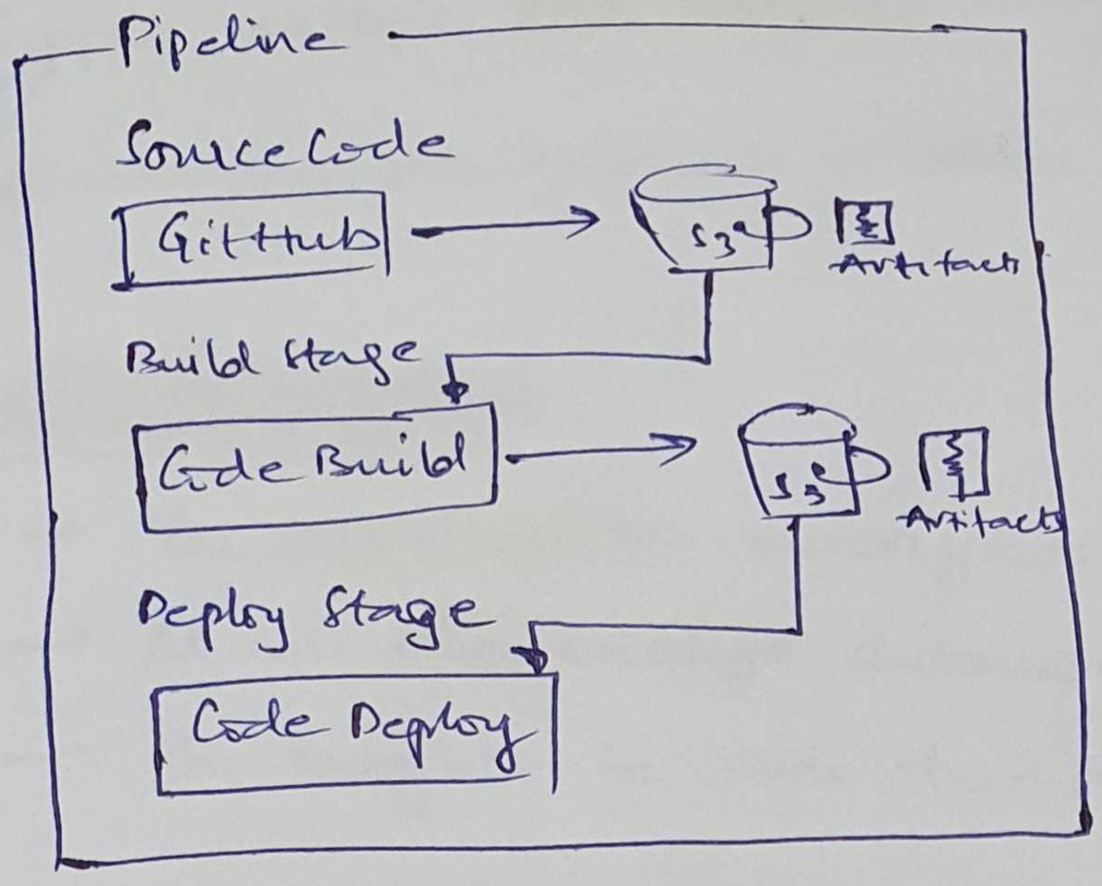
Actions:

- These are all the possible actions. Actions are not restricted to specific stages. Eg: You can use deploy actions in a build stage. pipelines are only limited by your creativity.
- You can create customactions if you need functionality beyond this list

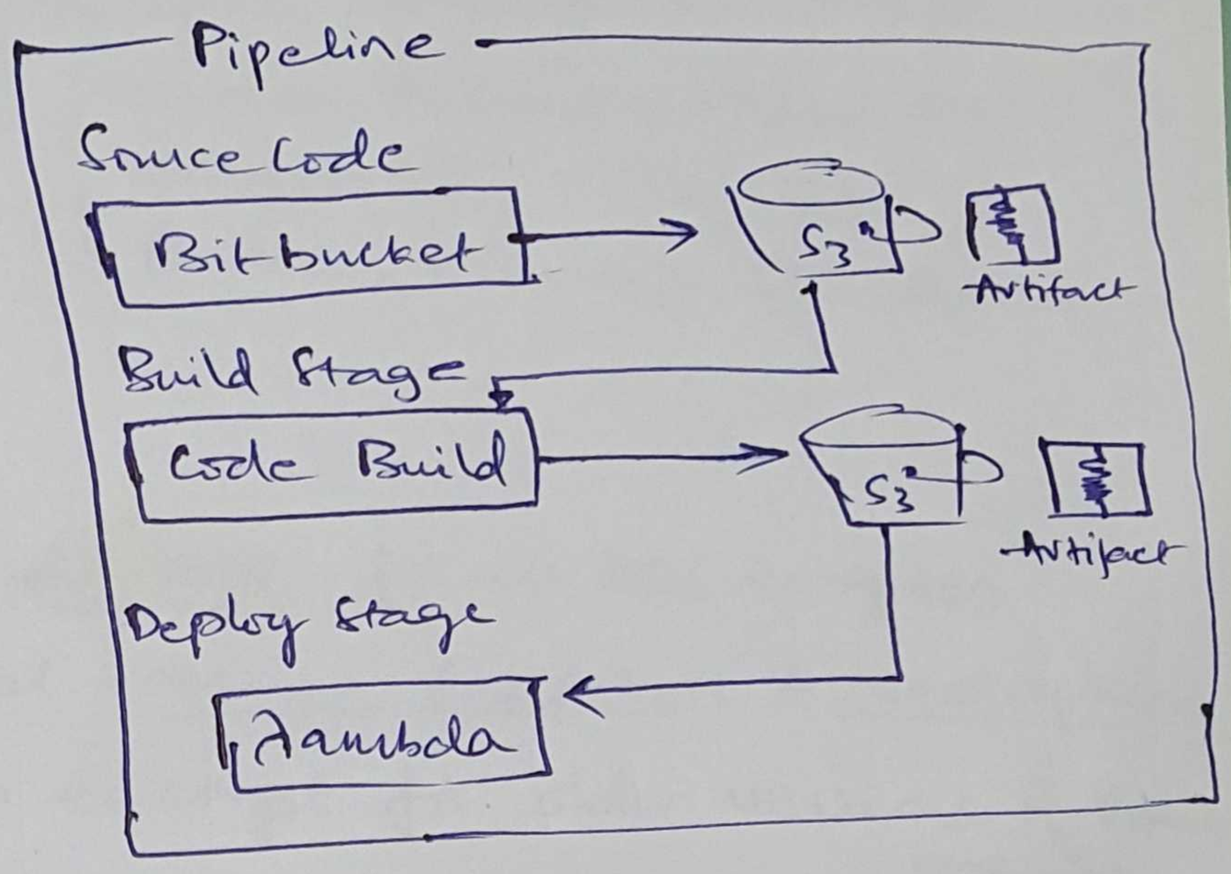
Source	Test	Deploy	Invoke
AWS CodeCommit	AWS Codebuild	AWS CF	AWS lambda
Amazon ECR	AWS Device Farm	Codedeploy	
S3	Add Jenkins	Elastic Beanstalk	
Bitbucket Cloud	Blaze Meter	AWS OpsWorks Stacks	
Github	Ghost Inspector UI testing	AWS Service Catalog	
	Micro Focus Stam Runner	ECS, S3	
	Runscope API Monitoring		

Use Cases:

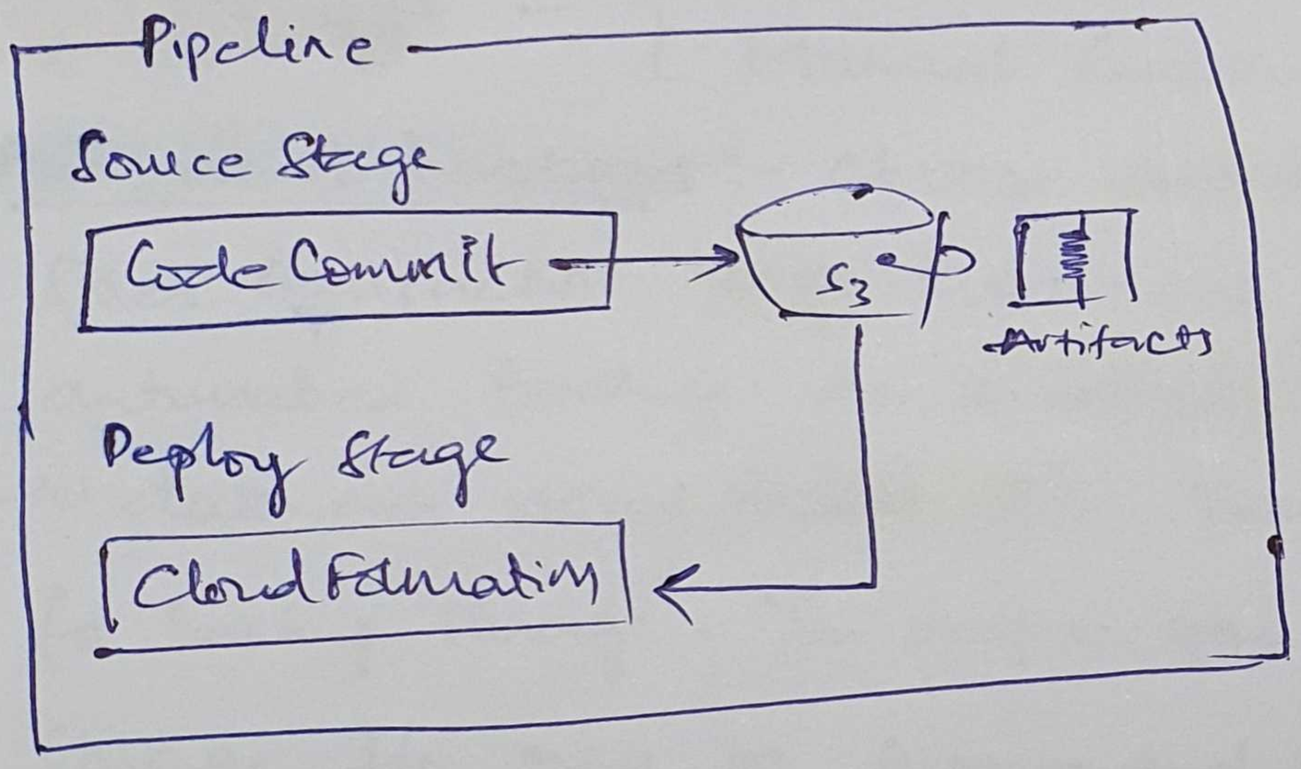
Traditional Monolith Application:



Static S3 Website



Serverless Functions



Code Star: [24:32] Quickly develop, build and deploy apps on AWS.

→ A wizard for getting common applications projects setup.

→ You get a deployment pipeline, access management and a project dashboard to monitor infrastructure

- Choose a template { Ruby on Rails / Python Django / ... }
- ~~Choo~~ Create a new git repo in Code Commit / Github
- It will create a pipeline

~~Section~~. Relational Database Service RDS :- (26:14)

- A managed relational database service.

Support multiple SQL engines, easy to scale, backup and secure

6 relational dbs available in AWS { Amazon Aurora, MariaDB, PostgreSQL, MySQL, Oracle, Microsoft SQL Server

RDS Encryption:

→ You can turn on encryption at-rest for all RDS engines

→ It will also encrypt automated backups, snapshots & readreplicas

→ You may not be able turn on encrypt for older versions of some engines.

Encryption is handled using AWS key managment service (KMS)

RDS backups: — { Automated backups, Manual snapshots

Automated backups:- choose retention period (1-35) days

Stores transactions logs throughout the day.

automated backups are enabled by default.

All data are stored inside S3. There is no additional charge for backup storage. You define your backup window.

Storage I/O may be suspended during backup.

Manual snapshots:- Taken by user. backup persists even if you delete the original RDS instance.

RDS Restoring Backup:-

- When recovering AWS will take the most recent daily backup, and apply transaction log data relavent to that day. This allows point in time recovery down to a second inside the retention period

→ Backup data is never restored overtop of an existing instance.

→ When you restore an RDS instance from automated backup/manual snapshot a new instance is created for the restored db.

→ Restored RDS instances will have a new DNS endpoint.

RDS Multi AZ: Ensures db remains available if another AZ becomes unavailable

- Makes an exact copy of your db in another AZ. AWS automatically syncs changes in the db over to the standby copy
- Automatic Failover protection. if one AZ goes down failover will occur and the standby slave will be promoted to master.

RDS Read Replicas:-

- Read replicas allow you to run multiple copies of your db, these copies only allow reads (no write) and is intended to alleviate the workload of your primary db to improve performance.
- You must have automatic backups enabled to use Read replicas
- Asynchronous replication happens btw the primary RDS and the replicas.
- You can have upto 5 read replicas, each will have its own DNS endpoint
- You can have multi AZ replicas, replicas in another region or even replicas of other read replicas
- Replicas can be promoted to their own db but this breaks replication.
- No automatic failover, if primary copy fails you must manually update urls to point at copy.

Multi AZ	Read Replicas
synchronous replication, durable↑ scalable↓	Asynchronous replication, durable↓ scalable↑
Only db engine on primary is active	All replicas are active, read scaling
Automatic backups are taken from standby	No backups by default
within same region 2 AZs	multi AZ, same AZ / cross region,
version upgrade on primary	upgrade is independent from source instance
automatic failover to standby in problem	promotion happens manually

Simple Storage Service (S3)

Object-based storage service. Serverless storage in the cloud.

- Data storage architecture that manages data as objects as opposed to other storage architectures
  - file systems which manages data as a file and file hierarchy
  - block storage which manages data as blocks within sectors and tracks

- objects contains your data and consists of
  - Key : name
  - Value : data itself
  - VersionID : if enabled then versioning
  - Metadata : additional info attached to the object
- You can store from 0 bytes → 5 TB
- S3 is an universal namespace, so bucket name must be unique

Storage Classes

Standard :- Fast retrieval, 99.99% Availability
11,9's durability, replicated across atleast 3 AZs

Intelligent Tiering :- use ML and decide appropriate storage class.
Data is moved to most cost-effective tier, no performance overhead.

Standard Infrequently Accessed (IA) :- Still Fast Retrieval,
Cheaper if you access files less than a month.
Additional retrieval fee is applied.
50% less than standard (reduced availability)

One Zone IA :- Still fast, objects available in only 1 AZ
availability (is 99.5%) but cheaper than standard IA
by 20% less. Retrieval fee is applied
Reduced durability, data could get destroyed.

Glacier :- For long term cold storage. Very cheap.
Retrieval is charged and takes mins to hours

Glacier deep archieve :- The lowest cost storage class.
Data retrieval is charged and is 12 hrs time taking

S3 Guarantees
- Platform is built for 99.99% availability
- Amazon guarantees
  - 99.9% availability
  - 11's of durability

	Standard	Intelligent Tiering	Standard IA	One Zone IA	Glacier	Glacier deep Archive
Durability	11 9's	11 9's	11 9's	11 9's	11 9's	11 9's
Availability	99.99%	99.99%	99.99%	99.5%	N/A	N/A
Availability SLA	99.99%	99%	99%	99%	N/A	N/A
AZs	>3	>3	>3	1	>3	>3
Min Capacity charge per Object	N/A	N/A	128 KB	128 KB	40KB	40KB
Min Storage duration charge	N/A	30 days	30 days	30 days	90 days	180 days
Retrieval fee	N/A	N/A	per GB	per GB	per GB	per GB
First byte latency	ms	ms	ms	ms	min to hours	min to hours

S3 Security:-

→ All buckets are private by default

- logging per request can be turned on a bucket.
- log files are generated and saved in a different bucket.
- (even a bucket in a diff acc if desired)
- Access control can be done by —
  - Access Control list: legacy (not deprecated) feature simple way of granting access
  - Bucket Policies:- use a policy to define complex rule access.

- ✓ block all public access
  - → ○ block public access and objects granted access through new ACLs
  - → ○ block public access in any ACLs
  - → ○ block public access to buckets and objects granted through new bucket policies
  - → ○ block public and cross-acc access to buckets & obj through any public bucket policies

```
{
  version: "2012-10-17",
  statement: [
    { "Sid": "PublicReadGetObj",
      "Effect": "Allow", "Principal": "*",
      "Action": "S3:GetObject",
      "Resource": "arn:aws:S3:::www.example.com/*"
}]}
```

Encryption:

- In transit - Traffic b/w your local host & S3 is achieved via SSL/TLS
- Service Side Encryption SSE : Encryption at Rest -

  can be done through
  - SSE AES - S3 handles the keys, uses AES 256
  - SSE KMS - Envelope encryption, AWS KMS
  - SSE C - Customer provided key

  you manage the key

- client side Encryption :-

  You encrypt your data files before uploading to S3.

Data Consistency:

New Objects (PUTS)

- Read After Write Consistency
- When you upload a new S3 obj you are able read immediately after writing

Overwrite (PUTS) / Delete Objects.

→ Eventual Consistency

- When you overwrite/delete it takes time for S3 to replicate versions to AZs
- If you were to read immediately S3 may return older version. Need to wait for few seconds

CRR - Region Replication:

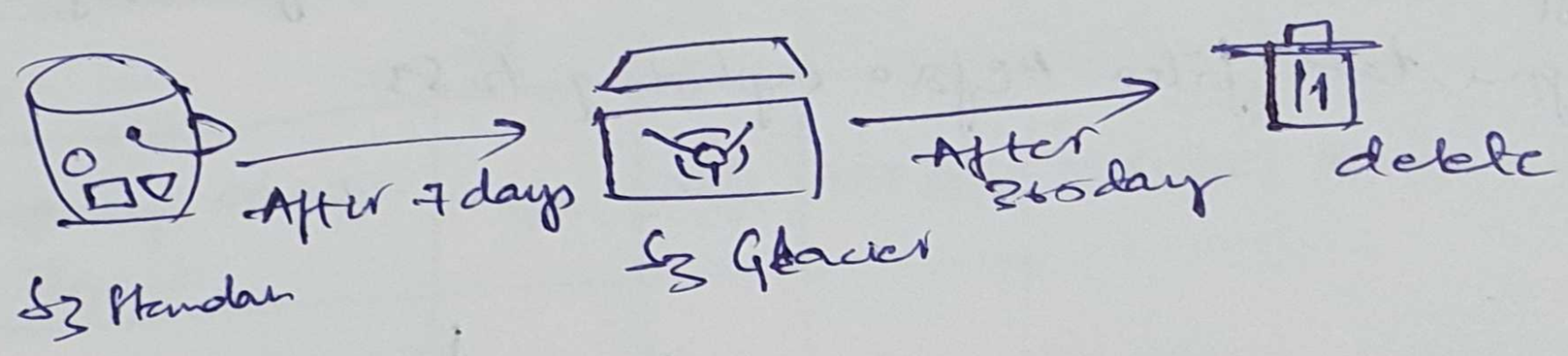
- When enabled, any obj that is uploaded will be automatically replicated to another region(s). Provides higher durability and potential disaster recovery for objects
- You must have versioning turned on both source & destination
- You can have CRR to another AWS acc.

Versioning:

- Stores all versions of an object in S3
- Once enabled it can't be disabled. only suspended in the bucket
- Fully integrates with S3 lifecycle rules
- MFA delete feature provides extra protection against deletion of your data.

Lifecycle Management :-

- Automate the process of moving objects to different storage classes / deleting objects all together.
- Can be used together with versioning
- Can be applied to both current & previous versions



Transfer Acceleration :-

- Fast and secure transfer of files over long distances b/w you end users and an S3 bucket
- utilizes cloud Front's distributed Edge locations
- Instead of uploading to S3, user uploads to distinct URL for an edge location.
- As data arrives at the edge location, it is automatically routed to S3 over a specially optimized path (Amazon's backbone n/w)

Pre Signed URLs :-

- ~~Generally~~ Generate a url which provides you temporary access to an object to either upload / download object data.
- Presigned URLs are commonly used to provide access to private objects. You can use CLI / SDK to generate them

[aws S3 presign s3://bucket/object --expire-in 300]
⇓
[https://bucket.s3.amazonaws.com/object?AWSAccessKeyId=...
&Expires=1503602631&Signature=.....]

- Commonly used to download files in your web apps.

MFA delete:

- MFA delete ensures users cannot delete objects from a bucket, unless they provide MFA code.
- pre requisites — { Versioning enabled on buckets
  AWS CLI must be ~~enabled~~ used to turn MFA on

[aws s3api put-bucket-versioning
— bucket bucket name
— versioning-configuration Status = Enabled,
MFA Delete = Enabled —mfa "your mfa serial number"]

- Only bucket owner logged in as Root user can delete objects from the bucket.

S3 follow along: [1:25:11]

Elastic Cache:- managed caching service which either runs Redis & Memcached.

Caching: It is the process of storing data in a cache, A cache is a temporary storage area. Caches are optimized for fast retrieval with the trade off that data is not durable.

In Memory Data Store :- When data is stored in-memory (think of RAM). The trade off is high volatility (low durability. Like if data lost) but access to data is very fast.

- Elastic cache helps to deploy, run and scale popular open source compatible in-memory data stores.
- Frequently identical queries are stored in the cache.
- Elastic Cache is only accessible to resource operating with the same VPC to ensure low latency.
- Elastic Cache supports 2 open-source caching engines:
  1. Memcache
  2. Redis

-) Memcache is generally preferred for caching HTML fragments.
-) Memcache is a simple key/value store.
-) Trade off simple but very fast

-) Redis can perform many diff kinds of operations on your data.
-) It's very good for leaderboards, keep track of unread notification data.
-) fast but not as fast as Memcache

	MemCache	Redis
Sub-millisecond latency	✓	✓
Developer ease of use	✓	✓
Data partitioning	✓	✓
Support for a broad set of prog languages	✓	✓
Adv data structures	✗	✓
Multithread architecture	✓	✗
Snapshots	✗	✓
Replication	✗	✓
Transactions	✗	✓
Pub/Sub	✗	✓
Lua scripting	✗	✓
Geospatial support	✗	✓

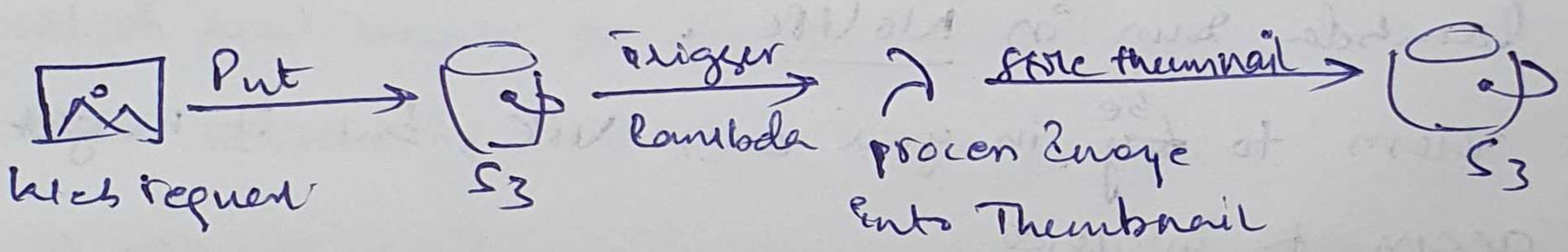
Lambda :- Run code without provisioning/managing servers.
Servers automatically start and stop when needed.
Serverless functions. Pay per invocation.

- λ executes your code only when needed and scales automatically to a few to a 1000 λ functions concurrently in seconds.
- You pay only for the compute time you consume, there is no charge when your code is not running
- λ is { Cheap, Serverless, Scales automatically
- λ supports 7 runtime languages { Py, Go, Java, Ruby, Node JS, C#, powershell
- You can also create your own custom runtime envs.

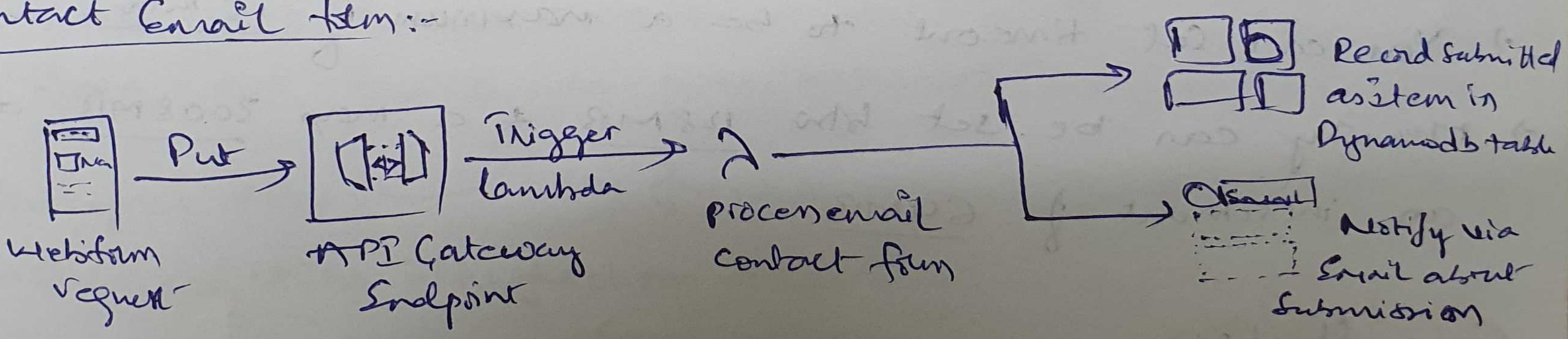
lambda is commonly used to glue different services together so the use cases are endless.

[Combine AWS to manage λ provisioned concurrency on a schedule]

Processing thumbnails:



Contact Email form:-



Trigger : lambda can be triggered/invoked by SDK/other services

few popular services to trigger lambda are

- API Gateway
- AWS Iot, Alexa...
- ALB
- CloudFront
- CloudWatch Events
- CloudWatch Logs
- Code Commit
- Cognito Sync Trigger
- DynamoDB
- Kinesis
- S3
- SNS
- SQS

Pricing :-

- First 1 million reqs/month are free
- Thereafter $0.20 per additional 1 million requests

- 400,000 GB seconds free per month
- Thereafter $0.0000166667 for every GB second

** This price will vary on the amount of memory you allocate

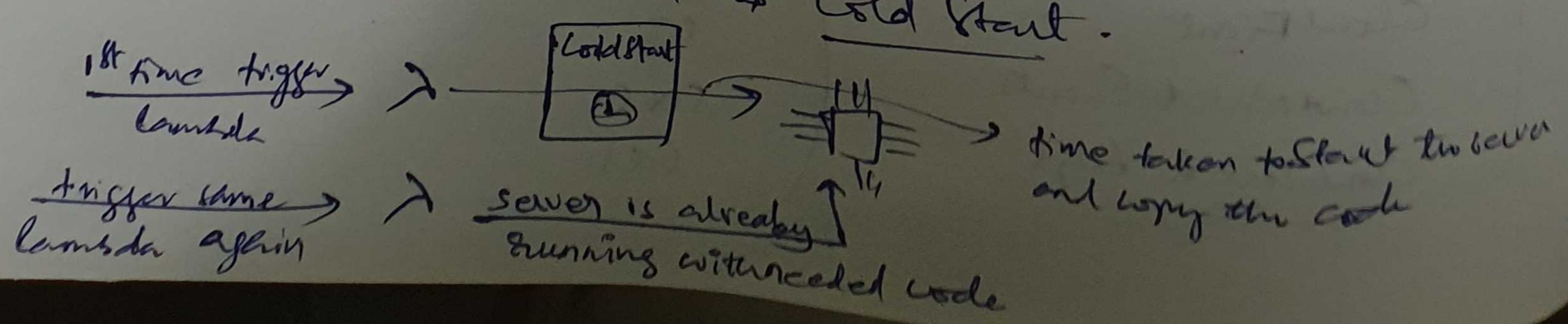
128 MB of memory × 30M executions per month × 200 ms run time per invoke = $5.83 — example

Defaults and limits :-

- By default you can have 1000 lambda running concurrently [ask AWS support for limit increase]
- /tmp directory can contain upto 500MB
- By default lambda run in No VPC. you can set them to be in your own VPC, but it might limit your access to internet
- You can set timeout to be a maximum of 15 min
- Memory can be set b/w 128MB to a max 3008MB at an increment of 64MB.

Cold Starts :-

- AWS has servers preconfigured (just sitting around turned off) for your runtime environment.
- When a λ is invoked those servers need to be turned on and your code needs to be copied over.
- During the time there will be a delay when the function will initially run which is called as Cold Start.



- If the same lambda is invoked and the server is still running it will use the server again, so there will be little to delay to running that function. This is what we call a Warm Server.

- Serverless functions are cheap, but everything comes with a trade off. Serverless functions cold starts can cause delay in User Experience. If your web application relies on being very responsive, than you want to reconsider serverless funs.

- There are strategies around Cold Start such as Pre-Warming which keep servers continuously running. Cloud Providers are always looking for ways to reduce cold starts.

Lambda Versioning :-

You can use versions to manage the deployment of your λ function

publish new version > choose version

each version has its own ARN

- arn : aws : lambda : us-east-1 : 1111111231 : function : MyFun : $LATEST
- arn : aws : lambda : us-east-1 : 1111111231 : function : MyFun : 2
- arn : aws : lambda : us-east-1 : 1111111231 : function : MyFun : 1

Qualified ARN → the function ARN with the version suffix

Unqualified ARN : the function ARN without the version suffix

- With unqualified you can't create alias, these always points to the latest.

Lambda Aliases:

Allows you to give a specific versions a friendlier name when you are accessing the λ programmatically

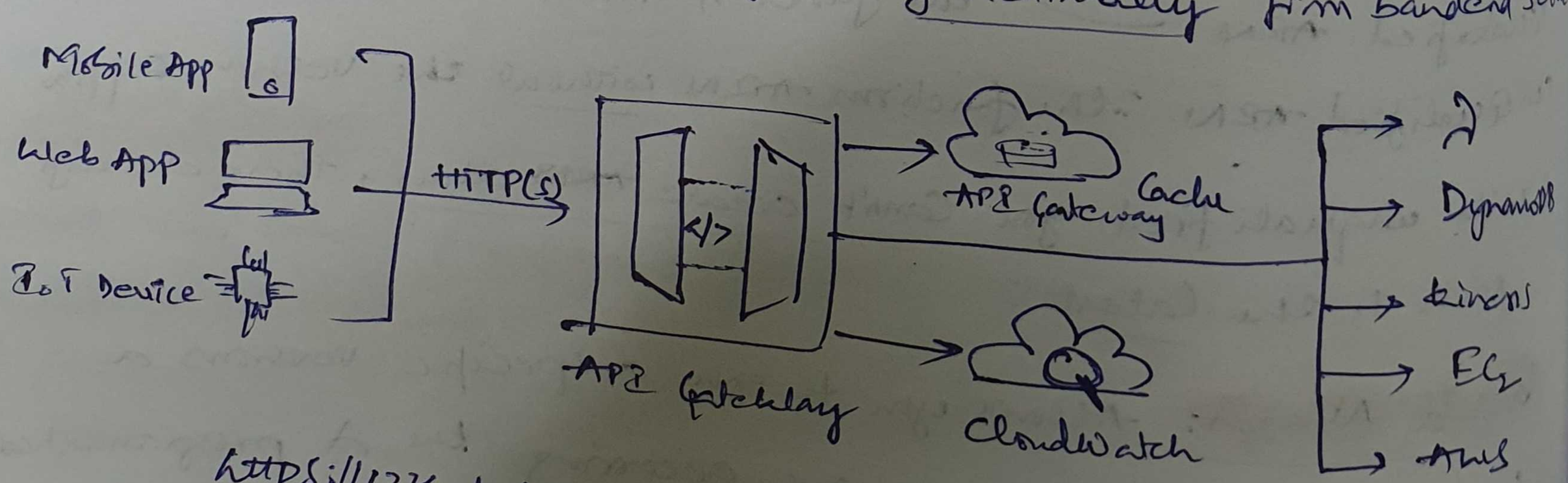
> Create alias > give name > choose version
also allow to give 2 versions (weightage %)

Lambda layers :-

- Pull in additional code and content in the form of layers.
- A layer is a zip archive, that contains libraries, a custom runtime, & other dependencies
- You can use libraries in your functions without needing to include them in your deployment package
- You can have up to 5 layer attached to a function. all layers can't exceed the unzipped deployment package size limit of 250 MB.

API Gateway :-

- Fully managed service to create, publish, maintain, monitor and secure APIs at any scale
- API Gateway is a solution for creating secure APIs in your cloud environment at any scale
- Create APIs that act as a front door for applications to access data, business logic & functionally from backend services



https://1234abcf.execute-api.us-east-1.amazonaws.com/prod

- Get /tasks
- POST /tasks/:id
- delete /tasks/:id

Key features:

- API Gateway handles all the tasks involved in accepting and processing up to 100s of 1,000s of concurrent API calls, including traffic management, authorization and monitoring.

→ Allows you to track and control any usage of the API.

Throttle requests to help prevent attacks.

→ Expose HTTPS endpoints to define a RESTful API

→ Highly scalable (happens automatically) and cost effective.

→ Send each API ~~endpoint~~ request to a different target

→ Maintain Multiple versions of your APIs.

API Gateway Configurations:

Access — { IAM roles/policies, Cognito User Pools, Lambda authorizers } for establishing access control

STS access control NOT supported

Resources: Resources are the URLs you define ex: /projects

When you create an API you need to also create multiple Resources

Resources can have child resources eg: /projects/-id-/edit

Methods: You need to define Methods on Resources.

You can define multiple Methods on a Resource.

Methods allow you to make API calls that resources work with that protocol eg: GET project/-id-

POST project/-id-

Stages: In order to use your API you need to deploy it to stages

Stages are versions of your API

Invoke API: For each stage AWS provides you a Invoke URL

This is where you'll make your API calls. It is possible to use a custom domain for your invoke URL

Deploy API: Everytime you make a change to your API you need to deploy it via the deploy API action. When you deploy you choose the stage

- When you create an API method on a resource you need to choose the integration type.
- The most common integration type is lambda.

Integration type
λ, HTTP, Mock
AWS service,
VPC link

- You can have fine tune control over the Request and Response for the method execution.

API Gateway Caching:

- API caching can be enabled to cache your endpoints response to API calls
- When enabled on a stage, API Gateway caches responses from your endpoint for a specified time-to-live (TTL) period.
- API Gateway responds to requests by looking up the response from the cache (instead of making a request to the endpoint)
- Reduces the # of calls made to your endpoint
- Improves latency of the requests made to your API.

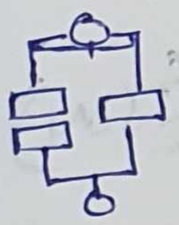
Cross-Origin Resource Sharing (CORS)

- CORS is a way that the server at the other end (not client code in the browser) can relax a same-origin policy.
- Allows restricted resources (ie Fonts) on a webpage to be requested from a different domain than the initial resource that it came from.
- Should always be enabled if using Javascript/AJAX that uses multiple domains with an API gateway.
- CORS is always enforced by the client.

Same Origin Policy :-

•) SOP is a concept in the application security model, where a web browser permits scripts contained in a first web page, to access data in a second webpage.

→ Same Origin Policy, is used to help Cross Site Scripting (XSS) attacks
→ They only works if both web pages have the same origin.
→ They are enforced at the web browser level.
→ They ignore tools such as Postman / Curl.



Step Functions:

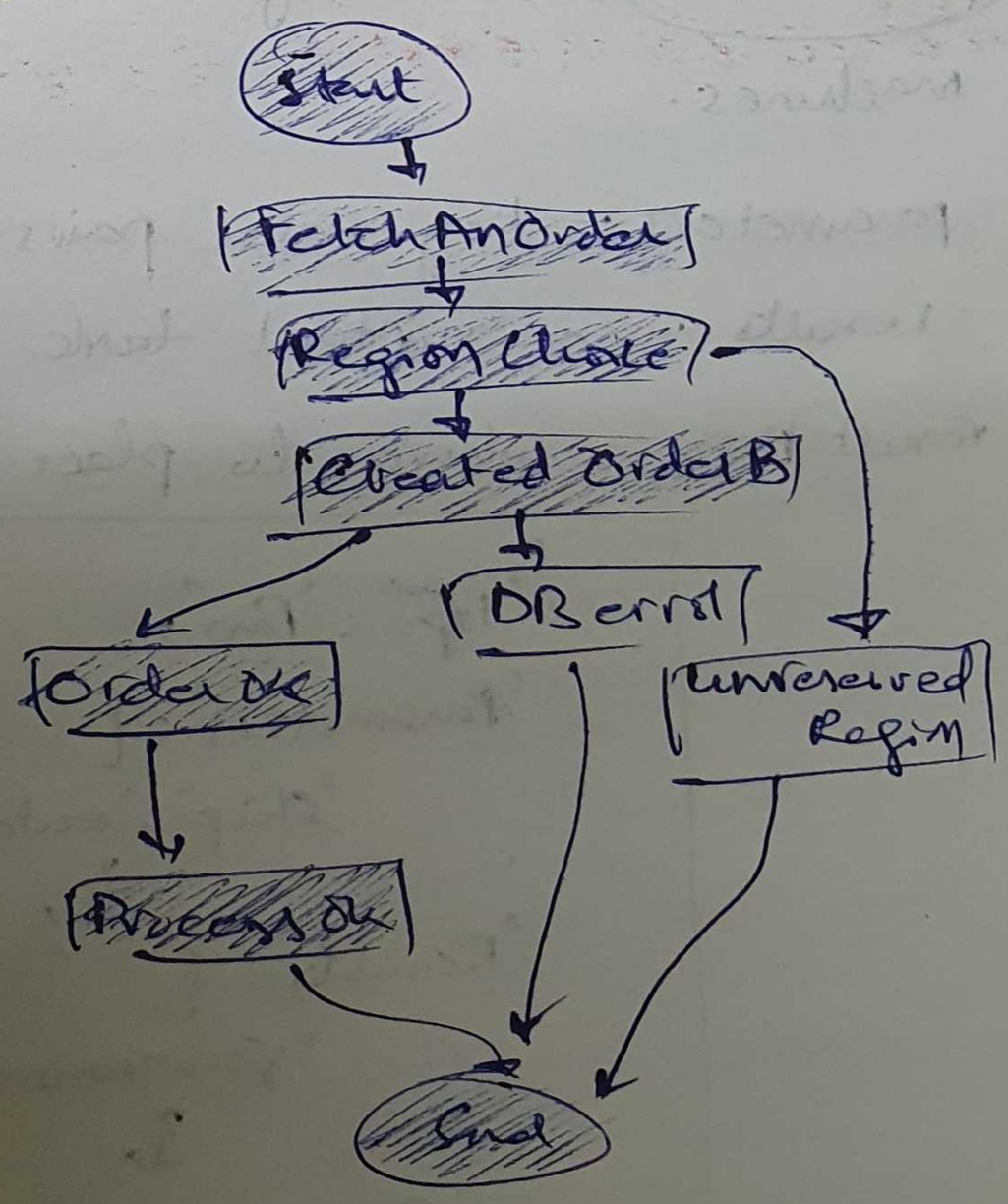
- {You can use step Funs to co-ordinate and manage the components of order management and inventory tracking workflow

coordinates multiple AWS services into serverless workflows.
Creating a state machine with micro services.

State machine :- It is a abstract model which decides how one state moves to another based on a series of conditions.
"Thinks of a state machine like a flow chart."

Step functions:

•) Coordinates multiple AWS services into a serverless workflow.
•) A graphical console to visualise the components of your application as a series of steps
•) Automatically triggers and tracks each step, and retries when there are errors, so your app executes in order and as expected, every time
•) logs the state of each step, so when things go wrong, you can diagnose and debug problems quickly.



•) Step functions has 2 types of state machine { Standard [General purpose], Express [Streaming data] }

durable, checkpointed workflows for ML, order fullfillment, IT/DevOps, ETL, long duration workloads

Event driven workflows for streaming data processing, microservices orchestration, IoT data ingestion, mobile backends and other short duration, high event rate workloads.

•) You can execute steps in parallel.

•) States are configured through Amazon States language (JSON)

Use Cases:-

→ Manage a batch Job / Fargate Container

→ Transfer Data Records.

States: passes its i/p to its o/p, without performaning work (dummy) Pass State

Pass states are useful when constructing and debugging state machines.

parameters :- key-value pairs that will be passed as i/p

results :- a virtual task to be passed to the next state

result Path :- where to place the "o/p" of the virtual task

```
{"Type": "Pass",
 "Parameters": {
        "ship": "enterprise"
        },
 "Results": {
        "Government": "federation"
        },
 "Result Path": "$.politics",
 "Next" : "End"
}
```

o/p

```
{"ship": "enterprise",
 politics: {
 "Government": "federation
        }
}
```

Task State: represents a single unit of work performed by a state machine. A task performs work by

1. a AWS lambda function
2. or passing parameters to the API actions of other services
3. or using an activity.

λ function task:

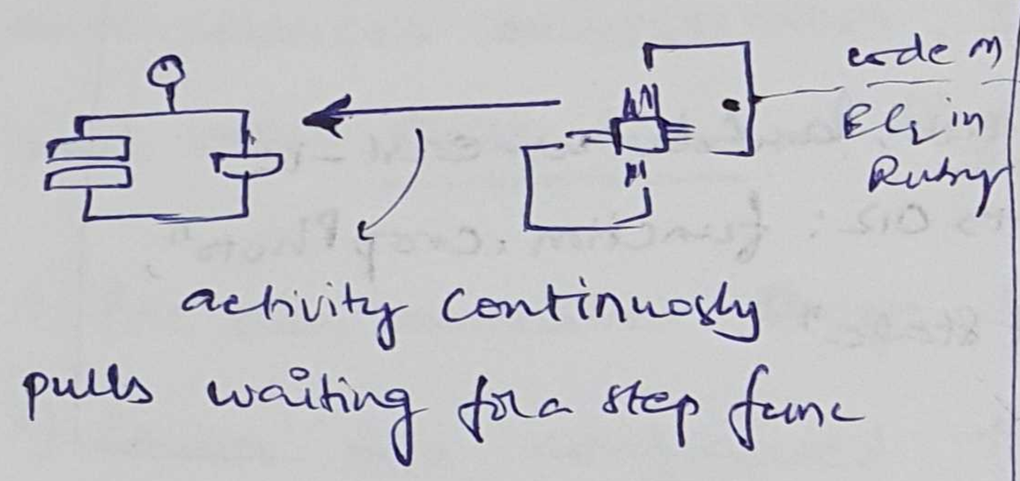
```
"lambdaState": {
    "Type": "Task",
    "Resource": "arn:aws:lambda:us-east-1:
                 123456012:function:cropPhoto",
    "Next": "NextState"
}
```

Supported AWS Services
ex: an batch job
λ, AWS Batch, DynamoDB
ECS/Fargate,
SNS, SQS,
SageMaker,
EMR, Step Functions

```
{"StartAt": "BATCH_JOB",
 "States": {
   "BATCH_JOB": {
     "Type": "Task"
     "Resource": "arn:aws:states:::batch:
                  submitJob.sync",
     "Parameters": {
       "JobDefinition": "preprocessing",
       "JobName": "PreprocessingBatchJob",
       "JobQueue": "SecondaryQueue",
       "Parameters.$": "$.batchjob.
                        parameters",
       "RetryStrategy": {
                  "attempts": 5
       }
     }, "End": true
   }
 }
}
```

Task - Activities

- enables you to have a task in your state machine where the work is performed by a worker that can be hosted on anywhere eg: EC2, ECS, mobile phones.
- when step functions reaches an activity task state, the workflow waits for an activity worker to poll for a task




```
activity = stepFunctions::Activity.new(
  credentials: credentials,
  region: region,
  activity_arn: activity_arn,
  workers_count: 1
  pollers_count: 1,
  heartbeat_delay: 30
  })
activity.start do|input|
  # do something
  {result:: SUCCESS, echo: input['value']}
end
```

choice state:-

- adds branching logic to a state machine.

Wait state:-

- delays the state machine from continuing for a specified time

Succeed state:

- stops an execution successfully

Fail state:-

- stops the execution of the state machine and marks it as a failure

Parallel states:-

- Can be used to create parallel branches of execution in your state machine. The state machine doesn't move forward until both states complete

Map Static:

allows you to iterate through an array and perform operations.