14/06 – Welcome call

https://udacity.zoom.us/rec/share/qcDWWhs11pvaUxGSkGURVC0jeYaSZy09UIQBQHPrSAowxa4BS3ac3cycUCDaSVyb.80frUjIXbYQYssaQ?startTime=1686733457000

Enrolment -
Reset classroom password
NatWest email address

Go through presentation

Tips -
Shut down your aws resources

Reach out for help – gpt/knowledge forum / tech support /mentors /nidhi

Timeline –

5 projs completed over 120 days

1 (Deploy static website on aws)– July 1

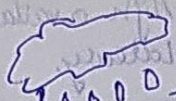
2 (HA web app using cloud formation)– July 24th

3 – Aug 18th

4 – Sept 11

5 – Oct 5th

CLOUD



Delivery of shared pool of on-demand (all software) computing services over Internet (broad network access).
- Rapidly provisioned & released (elastic process).
- Minimal mgmt effort & service prov interaction.

(Transition from traditional datacenters)
→ renting resources vs purchasing H/w

GCP by Ashish
@ 27/3/23

Cloud
- IaaS – H/w managed by provider (Virtualization, servers, storage, N/w), you Manage → O/S, Middleware, Runtime, Data, Appln. e.g. GCP, Azure, AWS
- PaaS – All except Appln & data is managed by provider. e.g. Heroku
- SaaS – e.g. Google sheet → direct software service. eg. Jira, Gmail. e.g. Kubernetes

Cloud Deployment Models.

- Private Cloud → connected via LAN (on-prem) GCP Anthos, AWS outposts, Azure Stack
- Community cloud → Separate threads
- ✓ Public cloud → GCP, AWS – resources over public internet → e.g. Web Appln – Public, Data – Private
- ✓ Hybrid cloud → combination of any two (Public + Private)
- Multi cloud → Using multiple public clouds (AWS, GCP)

Why Cloud? // Cloud Economics.

- Cap Ex ↓ → Capital Expenditure for scaling up, more load on server. (All infra cost completed upfront)
- Ope Ex ↑ → Operational Expenditure → operational, cost maintenance.
  - server lease costs
  - S/w & feature leases
  - demand Scaling costs
  - No upfront cost.
  - Pay as you consume
- Availability ↑
- Latency ↓

Server costs, storage costs, N/w costs, backup & archive costs, data center costs.

Request/Response

Client (Mukul) → req → POP (point of presence) (edge N/w) → req → Google Data Center
Client ← res ← POP ← res ← Google Data Center

Pizza As a Service

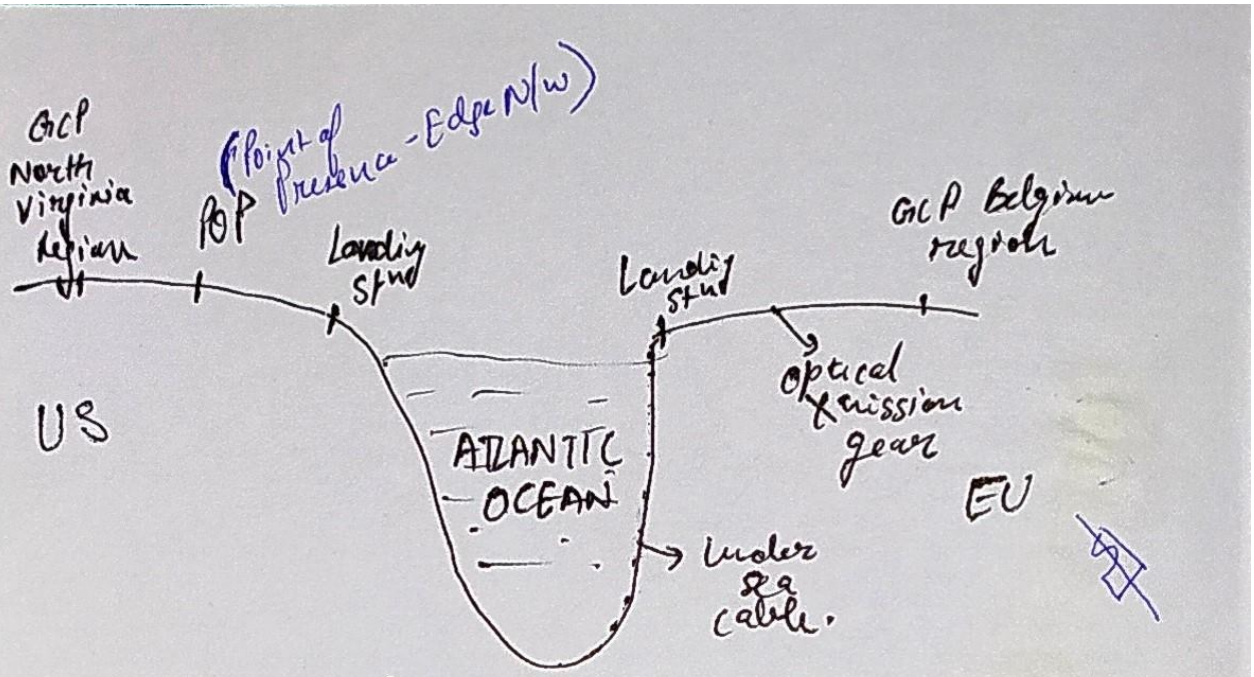
Made at Home (on-Prem)	Take & Bake (IaaS)	Pizza Delivered (PaaS)	Dined Out (SaaS)
Dining Table	Dining Table	Dining Table	Dining Table
Soda	Soda	Soda	Soda
Gas	Gas	Gas	Gas
Oven	Oven	Oven	Oven
Fire	Fire	Fire	Fire
Pizza Dough	Pizza Dough	Pizza Dough	Pizza Dough
Tomato Sauce	Tomato sauce	Tomato Sauce	Tomato Sauce
Toppings	Toppings	Toppings	Toppings
Cheese	Cheese	Cheese	Cheese

You Manage
Vendor Manages

Cloud service Models. XaaS

Traditional	IaaS	PaaS	SaaS
Applications	Applications	Applications	Applications
Data	Data	Data	Data
RunTime	RunTime	RunTime	RunTime
Middleware	Middleware	Middleware	Middleware
O/S	O/S ← Limit of Consumption	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

Infra Stack →



Geography & Regions example	Zone ← Region ← Multiregion ← USA
	us-east4-a us-east4-b us-east4-c
	us-east4
	nam3 N Virginia/ S Carolina

Npoiy — OSI model

7 Application → HTTP / HTTPS / DHCP / DNS / SSH - Telnet
6 Presentation
5 Session
4 Transport → TCP / UDP - IP packets
3 Network → IPv4 / V6 — IP address subnets
2 Data Link
1 Physical

Google Cloud Global Infra

Latency control

Considerations
- cost
- Data locality — High availability, Latency.

Infrastructure — Google resources → Region / Zonal Impact ← Explored

- (1) Multi-Region — (IND/US) - - - 36 countries — Geo redundancy to max distribution.
- (2) Regions — Delhi / Mumbai ... state → 3 zone ... Total 109 zone — collection of zone for HA; Inter communication b/w zones < 5ms within region.
- (3) ZONE — Smallest Unit → a, b, c / b, c, d (sub datacenter) deployment area for GCP resources within a region

Website — (Domain-URL / IP address) // CDN → Content delivery N/W

↓

Load Balancer (Regional / Global)

↓

Appln | Shared VPC → allows an org to connect to proj in a common VPC; contain shared N/W

Networking

(1) VPC → Virtual Private Cloud (inside Public Cloud)

- S/W version of N/W to link resources in a project (no sharing b/w projects) • Core networking service
- global ✓, Transparent, Modifiable, Hybrid route, connectivity, Packet Mirroring
- custom / default → Region wise fixed range
- (No IP address) → custom range

Global resource encapsulated within a project.

VPN → Subnet (SubNetwork) → CIDR . xyz | subnet address size no. after slash

★ Classless Inter-Domain Routing → slash after IP

(zonal) → associated with region

expandable from /20 (low) to /16 (high)

PREFIX ↗

(Subnet - calculator . com)

Subnet Address Space
- Reserved address
- Example subnet - 10.10.0.0/16
- Subnet address - 10.10.0.0
- Gateway address - 10.10.0.1
- 2nd to last address - 10.10.255.254
- Broadcast address - 10.10.255.255

Avoid large subnets to reduce CIDR collisions.

(2) Firewall — ingress / egress (to control traffic)

✓ default → SSH - 22, relp → 3389

✓ custom — app 1 ← - - → app 2 (net tag)

Reserved IP address
- First address - N/W
- 2nd " - Default g/w
- 2nd last - GCP future use
- last - Broadcast

(3) Storage

Routing — static / dynamic

↳ There to direct traffic packets.

VPC cont'd.

- Resources communicate with one another via internal (private) IPv4 addresses
- Only IPv4 addresses support

N/w types → Auto-mode
→ custom-mode.

Networking added

DNS → DNS in GCP (Publish & maintain DNS records CLI/API/SDK)

stored in zone file hosted on Name server

Google Provided DNS

Customer DNS server
- SaaS server with DNS
- Infoblox Virtual Appliance

Internal IP addresses
- Allocated from subnet range via DHCP
- DHCP lease default is 24 hours
- VM name is registered automatically to internal DNS
- 2 types of internal IP address → Static (Assigned until explicitly released. Attached to stopped instances until removed)
- Ephemeral

(used within a N/w; resources communicate privately.)

Internal IP address reservation
- Reserve address & then associate with a resource
- Specify internal IP address then promote to static.

Auto/custom internal IP addressing (Deleting and re-creating instances will change the IP. Stopping or restarting will not)

Naming → [Instance Name].[zone].c.[project-ID].internal

External IP addresses
- Can be assigned to instances or a forwarding rule.
- Needed to communicate to the internet, resources outside of VPC.
- Only resources with an external IP can send & receive traffic directly to & from outside the N/w

(Same as internal IP address reservation)

Types

Static external
- Long-term assignment
- Remain attached to stopped compute instances until explicitly removed.

Ephemeral External
- Remain in place only until compute instances are stopped/restarted/terminated

Regional / Global actors
(Global reserved for global forward rules used by global LB)

New IPs assigned when instances are restarted again.

Hybrid Connectivity — On Premises to GCP connectivity

- **Cloud VPN**
  - IPsec VPN connection
  - Requires Cloud VPN G/w service (Regional)
  - Cloud Router required for BGP
- **Cloud Interconnect**
  - Dedicated or partner-connected service up to 50 Gbps.
  - HA, low latency, enterprise grade connection

In a VPC, route consists of a single dest^n (CIDR) & single next hop.
- All routes stored in VPC routing table.

GCP routes
- System Generated
  - Default
  - Subnet
- Custom
  - Static
  - Dynamic

Static Vs Dynamic Routing → Static routes — you must create or maintain a routing table. A topology change on either N/w requires you to manually update static routes.

Cloud Router : you can use BGP to exchange routing info b/w N/w. Instead of manually configuring static routes, N/w automatically & rapidly discover topology changes through BGP. (Dynamic routing)

VPN → Cloud VPN — connect existing N/w to VPC through IPsec connection

HA-VPN
- 99.9% SLA when configured with 2 interfaces & 2 public IPs.
- Public IPs created from pool.
- No fw rules required.
- Only dynamic routing — BGP
- Supported — 2 tunnels from 1 cloud VPN g/w to the same peer g/w
- API resource — vpnGateway resource

Classic VPN
- 99.9% SLA
- Public IPs & fw rules must be created.
- Static routing - Policy based / route based
- Dynamic routing — BGP
- Not supported.
- target-vpn-gateway resource

Cloud Interconnect

Dedicated

- Direct connection to google.
- BGP configuration required on your on-premises routers & cloud routers.
- Google provided 99.9 SLA

Partner

3rd party – more points of connectivity through one of our supported service providers.

For 2 layer connections – BGP on on-prem routers & cloud routers.
For 3 layer connections – automated configuration of cloud routers & peers.
Google provided SLA b/w google & service provider

Firewall – Every VPC n/w functions as a distributed f/w.
– F/w rules defined at N/w level & connections allowed/denied on a per-instance (VPC) basis.
– GCP f/w rules exists b/w instances & other N/w
└ b/w individual instances within same N/w

F/w rule components →

- Priority – If rule will be applied. Highest priority where (lower no. – higher priority) component match traffic is applied, lower priority ignored.
- Traffic direction – Ingress – incoming traffic from sources to or IP target
  Egress – traffic to destinations from GCP targets. (NOT BOTH)
- Action on Match – Allow / Deny (NOT BOTH)
- Target – Instance to which rule will apply. – (Target tags / Source (ingress) IP ranges / Destination (egress) / Service acc)
- Protocol & Port – TCP, UDP, ICMP (which protocol)
- Enforcement – Enable / Disable

Default rules in default N/w

- default - allow - internal
- default - allow - ssh
- default - allow - rdp
- default - allow - icmp

Implied & Pre-populated rules (can be overridden)

GCP
- Egress rules
  - TCP: port 25 blocked (deny)
- ingress rules
  - only allowed TCP, UDP, ICMP, GRE protocols (allow)

Load Balancing

Distributing workload across multiple instances

HTTP(S) LB

Distribute traffic across regions to ensure req are routed to closest region or in event of disaster to a healthy next closest region instance

Based on content type

N/w LB

Distributes among server instances in same region

Based on incoming IP protocol data - Address / port / protocol

Advanced Connectivity

- Direct Peering - Exchange internet traffic b/w business & google at Google's broad reaching edge N/w locations
- Carrier Peering - Connect your infra to google's N/w edge through HA, low latency connections by service providers

VPC Peering

Private connectivity across 2 VPC N/ws for same or diff proj/org.

Private Google Access

For VM instance that have only internal IP address (no ext) → to reach ext IP address of Google APIs & services.

- can be enabled by subnet basis

Shared VPC → Allows an org to connect resources from multiple proj to common VPC to communicate by using internal IP address from that N/w.

One project is made host & others are made service projects.

VPC flowlogs → Records sample of N/w flows sent & received by VMs
- used for N/w monitoring, security analysis, expense optimization
- Visible in cloud logging. & can be exported to Pub/Sub for analysis
- Enabled for all VMs in a subnet.

Record Format

Core fields
- connection → src-IP, port; dest IP, port; protocol
- start_time
- end_time
- bytes_sent
- packet_sent
- reporter

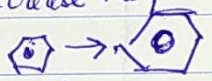
additional fields
- src instance, dest instance } → proj-id, VM name, region, zone
- src VPC, dest VPC } → proj-id, VPC name, subnet name
- src loc, dest loc } → continent, region, country, city
- src gke details, dest gke details } → Cluster / Pod / Service

O'reilly Course

Cloud features
- On demand self service → no human interaction needed to get resources
- Broad N/w access → Access from anywhere via internet
- Resource pooling → Provider shares resources to customers
- Rapid elasticity → Get more resources quickly as needed.
- Measured service → Pay only for what you consume.

Run your app in someone else's datacenter. Cloud providers are responsible for physical H/w and facilities necessary to ~~execute~~ Execute work.

Scalability → ↑/↓ resources based on workload demand

Vertical scaling → Add additional resources to increase the power of H/W workload

eg. Add additional CPUs to a VM.

GCP instance groups.

Horizontal Scaling → Adding more nodes.

↳ better for cloud computing

Google cloud

Services

Compute —	Compute engine ✓	App engine ✓	Container engine ✓	Container registry ✓	Cloud Run ✓	Cloud Functions
Networking —	Cloud Virtual N/w VPC ✓	Cloud Load Balancing ✓	Cloud CDN	Cloud Inter connect ✓	Cloud DNS	

Content delivery — Binary or Obj data (Images, Media, Backups) / Data lakes

Storage & Databases — Cloud Storage, Cloud Bigtable, Cloud Datastore (NoSQL), Cloud SQL (Postgre SQL / MySQL / SQL server, HA across zones), Persistent Disk

Identity & Security — Cloud IAM, Cloud Resource Manager, Cloud Security Scanner, Cloud Platform Security

Big Data — BigQuery (Warehouse – Enterprise Data – Analytics/Dashboard; Enabled via BigQuery data xfer API), Cloud Dataflow, Cloud Dataproc, Cloud Datalab, Cloud Pub/Sub, Genomics

Machine Learning — Cloud Machine Learning, Vision API, Speech API, Natural Lang API, Translation API, Jobs API

Management Tools — Stack Driver, Monitoring, Logging, Error Reporting, Trace, Debugger, Deployment Manager, Cloud Endpoints, Cloud Console, Cloud Shell, Cloud Mobile APP, Billing APP, Cloud APIs (Order Method for console & SDK; Google API for services to be used for proj.; Dashboard)

Developer Tools — Cloud SDK, Deployment Manager, Cloud Source Repositories, Cloud Tools for Android Studio, Cloud Tools for IntelliJ, Cloud Tools for Powershell, Cloud Tools for Visual Studio, Google Plugin for Eclipse, Cloud Test Lab

example gcloud init
gcloud auth login
gcloud config
gcloud components

Format gcloud + component + entity + operation + target + flags

CLI

command line tools
- gcloud for Compute Engine
- gsutil - cloud storage
- bq - BigQuery
- kubectl - Kubernetes

- Infrastructure as code
- Powershell supported
- Can be installed on Windows / Linux / Mac OS

Access GCP
- Cloud Platform Console
- Cloud Shell & SDK
- Cloud Console Mobile App
- Rest API

Browser based CLI via dev tools loaded VM 5GB persistent home directory; code editor built-in pre installed SDK

Types of cloud servics - Compute

Compute decision tree (Know what you're deploying)

Yes
- Day to day computing - General N1
- Web serving, databases - General N2
- HPC, EDA, gaming - C2 compute optimized
- Large in-memory DB - M2 memory optimized

NO

(What do you optimize)
- Cost - General N1
- Best performance to Price - N2 General
- Ultra High performance - C2 compute optimized
- Ultra High memory - M2 memory optimized

Traditional

Virtualization

VM (instances)

App 1 | App 2 | App 3
OS | OS | OS
Hypervisor
Hardware
CPU Memory Disk

Instance grp to manage autoscaling to add/remove capacity.

App 1
OS
Hardware
CPU Memory Disk

Attach/Detach as needed

Containers
- Standardized packaging of s/w & dependencies
- Isolation of apps from each other
- Allows separate apps to share the same OS kernel
- Linux + Windows compatible
- Container registry used to store approved containers.

Speed, Portability, Efficiency ↑

- Choose region & zone
- Choose OS & s/w to put
- Choose image - public/private
- SSH

GCP Compute engine

Machine Type - Virtualized h/w resources available to VM instances grouped for diff workloads & come in variety of families,
N1 - General - best price performance ratio
96 vCPUs, 6.5GB per VCPU
N2 - General - 80 vCPUs, 8GB/vCPU
C2 Compute optimized - High CPU to Mem Ratio
M2 memory optimized - High Mem to CPU ratio
f1 - Shared Core Micro
g1 - Shared core Small
CPU consideration -

Containers

App 1 | App 2 | App 3
Docker
OS
Server

Image, Container, Engine, Registry, Control Plane

Serverless → Run code in an env that doesn't require setting up VM or Kubernetes cluster

google gives 2 serverless options → App Engine
→ Cloud functions

(Single Purpose functions)
(JS/Python/Go/Java)

Short processes running in response to an event in a fully manged env
- No need of infra

Apps & containers running for extended period of time
- Apps can be built in Go/Java/Net/Node PHP/Python/Ruby

VM images (GCP)
① Public images – Provided & maintained by Google & 3rd party. (Google Cloud Marketplace)
② Custom images – Created from boot disks & other images.
③ Shielded vs Non-shielded images – verified images with additional security features.

Redeem of compute.

compute engine	Kubernetes engine (GKE)	App engine	Cloud functions	Managed services.
IaaS	Hybrid (Container as a service) CaaS	PaaS	serverless Logic (Functions As A service) FaaS	Automated elastic (less flexibility)
(More Flexibility)				

Managed Infra → Dynamic Infra

Monolithic App Issues – Minor code requires full re-compile & testing; Appln is single point of failure; Appln is difficult & expensive to scale.
simple deployment; Vertical scaling; inter module refactoring

Partial deployment; storage models bottlenecks; Horizontal scaling → Microservices – Break appln into separate services.
12 Factor apps – Make apps independently scalable, stateless & highly available.

Application → Service 1 – Scaling – Oracle
2 – No scaling – SQL
3 – Scaling – Redis
Functional decomposition → Data De coupling

Storage

GCP

- Object Storage (Cloud storage) — video/photo
  - Provides object storage
  - Not stored in a conventional file system
  - Grouped into buckets
  - Each object is addressable by a URL.
  - no min obj size & unlimited storage
  - Available in multiple flavors
- File Storage
  - NFS – Network File System (NFS v3 compliant)
  - Suitable for appln that require OS type access to files.
  - File system & directories are de coupled from VM
  - Used with VM instances & Kubernetes cluster
  - Platform running apps.
- Block Storage
  - Fixed-Size data structure organized in blocks
  - Attached to VMs in GCP.
  - Persistent or ephemeral
  - Independent life cycle to that of VM.
  - Support file system-level access vs HTTP access to obj storage

Available in zonal & regional options
Flavors: standard, solid state (SSD), low latency

Data
- Structured
- Semi Structured
- Unstructured

GKE — Built on Open-source Kubernetes.
- Flexibility to integrate with on-prem Kubernetes
- Uses Google Compute Engine as nodes in a cluster

Cloud Functions Use cases — Data processing / ETL operations
- Webhooks responding to HTTP triggers.
- APIs that compose loosely coupled logic (Video transcoding)
- Mobile backend functions

Cloud Run (Knative Standard) — Fully managed compute platform for deploying & scaling containerized Apps
Serverless for containers

जे. के. सुपर सीमेन्ट

FaaS - no infra mgmt.
Any language / Any library / library.

Horizontal scaling

12 Factor Apps - Make app independent

Application → Servia 1 — Scaling
2 — ..

Storage

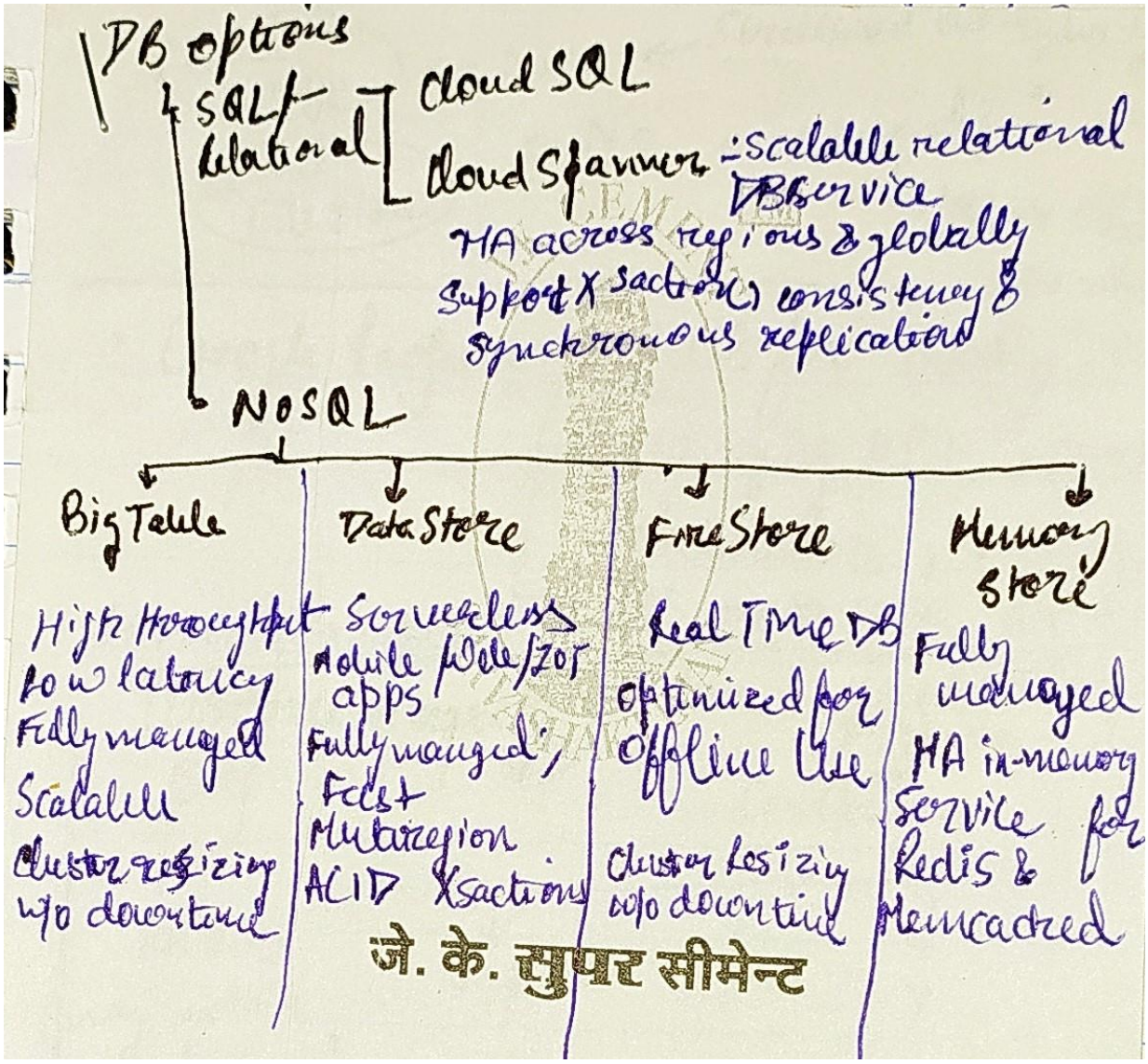
Cloud storage flavors

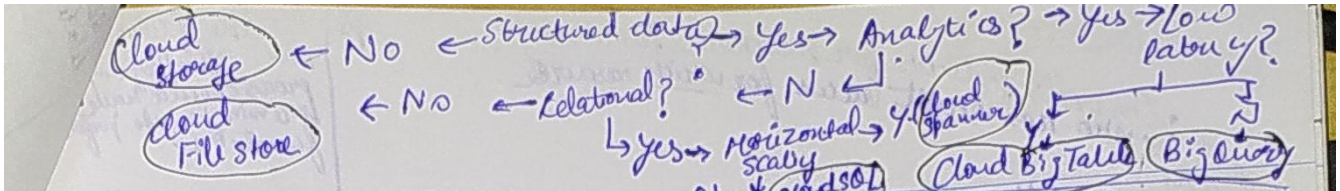
Storage Classes
- Standard — Max availability; no limitations
- Nearline — Accessed < 1/month
- Coldline — Accessed < 1/quarter
- Archive — Accessed < 1/year.

Availability
- Region — 1 in 1 region
- Multi-region — Large geographic area.
- Dual-Region — Pair of regions

जे. के. सुपर सीमेन्ट

→ Kubernetes


DB options
SQL / Relational
Cloud SQL
Cloud Spanner
: Scalable relational DB service
HA across regions & globally
Support X sactions, consistency & synchronous replication
NoSQL
Big Table
High throughput
Low latency
Fully managed
Scalable
Cluster resizing w/o downtime
Data Store
Serverless
Mobile / Web / IOT apps
Fully managed, Fast
Multi region
ACID Xactions
Fire Store
Real Time DB
Optimized for Offline Use
Cluster resizing w/o downtime
Memory Store
Fully managed
HA in-memory service for Redis & Memcached
जे. के. सुपर सीमेन्ट



- Google Projects → Basis for creating, enabling & using GCP services
  → Managing API, enabling billing, add/remove collaborators, managing permissions for GCP resources.
  → All resources are associated with a project.

Hierarchy Example (Resource Hierarchy) – parent child relationship.

Org. — DOMAIN — (Cloud level)

Company.

Folders (may contain proj / sub folders / both)

Dept X

Y

Team A

B

Product 1

2

Shared Infra.

Account level resource

Projects

Dev GCP Proj

Test

Prod.

Resources (Service level resources)

Compute engine instances

App engine services

Cloud storage buckets.

Labels → Key-value pairs to organize GCP resources
- Attach to resources, filter
Use → • Team/cost center labels
- Component label
- Env label
- State/VM labels

→ Billing data can be exported to Big Query (usage cost/pricing)

Cloud Billing A/c
- Linked to a Payments Profile → Individual / Business
- Can be linked to multiple projs
- Subaccounts can be used for resellers
- Managed by IAM – Billing A/c admin / creator / user / viewer / manager

Budget Alerts based on rules customizable

Sustained use Discount (automatic)

Commitment Based Discounts (1/3 yr)
- Spend Based – min amt of service (hrs) – Cloud engine, Cloud SQL
- Resource Based – min amt for compute engine resources – CPU, Mem, GPU,

Reservations → for VM instances you need in future so that these will always be available for you.

- Instance sometimes takes time and are not available for autoscaling wrt of supply/demand

Configuration

- Properties of instance group should exactly match with reservation
- Paired with committed based discount.

Quotas & Limits – Limit on the resource your proj can use
Rate quota → Reset after specified time eg API requests
Allocation quota → Must be explicitly released eg # of load balancer

Why? : resource management
: Availability for others protection

Access quotas & req more quota
① IAM → Quotas
② API dashboard → Quotas

• Controls policies for resource hierarchy.
• who has what access for which resource.

Principle of least priv. → user/program/process should have bare min privilege to perform function.

IAM — Who (Identity) → Google acc / cloud identity user
→ Service acc → user-managed → name@proj-id.iam.gserviceaccount.com
→ Default → Automatic granted editor role
→ Google-managed → used by Google services
→ Google group
→ cloud identity → or G suite domain → Identity as a service for Device mgmt, data wipe
→ All Authenticated Users (IAP auth) → Identity, SSO, Directory mgmt → Via GCDS
→ All Users

managed by servic A/C Keys → Google Managed key
→ User "
for Auth
• Service a/c use access scopes
↳ Allow default access
↳ Allow full access to cloud API

Role Inheritance (WHAT) → Org → folders → project → resources

Role Types → Primitive – Owner/Editor/Viewer/billing admin
Pre-defined → ML.admin / ML.developer
granular access for specific service; managed by GCP.
Custom ← user defined

Google Cloud Directory Sync (GCDS)
↳ Sync data in Google a/c with Microsoft AD / LDAP server → users/groups/shared contacts
On prem to GCP

Condition IAM permissions
<service>.<resource>.<verb>
eg compute.instances.stop

• Org Admin role
• role Admin ~~role~~

Compute Instance Predefined Roles — Compute Engine Admin,
" " N/w "
" " Security "
" " Viewer "

To view policies from Shell
gcloud projects get-iam-policy <proj_id>
gcloud resource-manager folders get-iam-policy <folder_id>
gcloud organizations get-iam-policy <org_id>

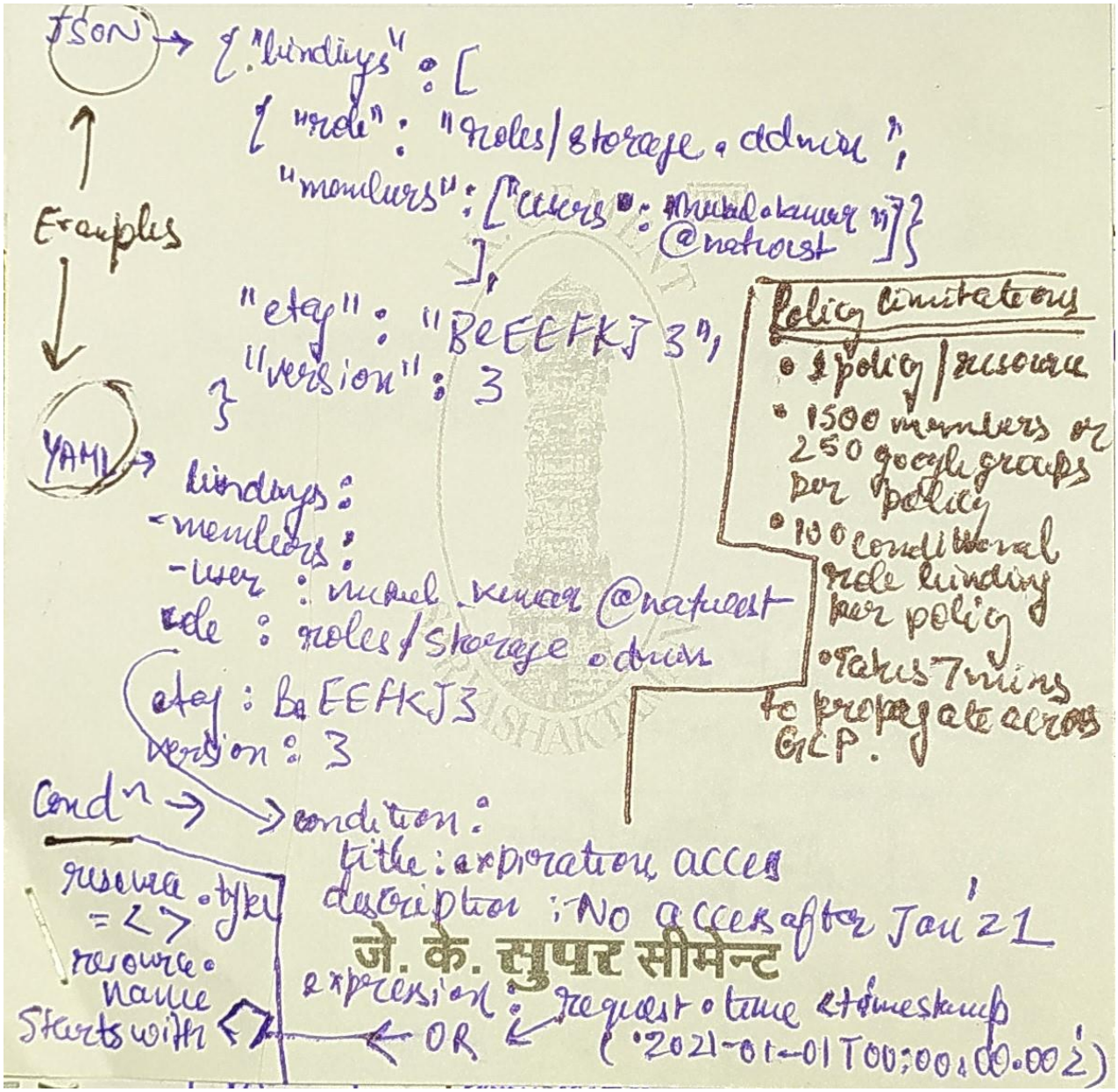
Super Admin Account → irrevocable admin permissions; Grants Org Admin role; Recover a/c at domain level.

Policy Architecture
Member | Condition (Constraint Limit) | Meta data, etags | Version | Role (Permissions) | Audit Config | Binding
Condition → conditional/attribute based access eg timed access for contract employee
Meta data, etags → concurrency control; to avoid race condition when multiple systems change policies.
Version → specifies schema version; to avoid breaking existing integrations & for new feature releases
Audit Config → configure audit logging
Policy statements can be written in JSON/YAML

Google Group ↔ Google A/C
Cloud identity ↔ Service A/C
↔ G Suite

compute.instances.list
.get
.start
one to one REST API methods

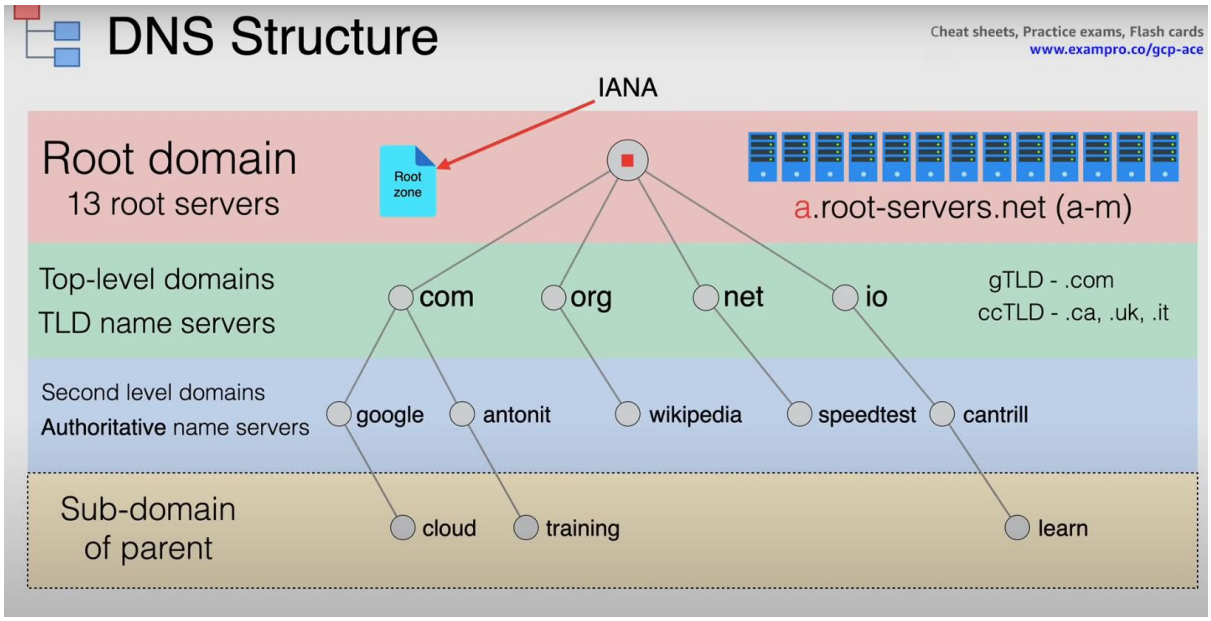
auditConfig:
- auditLogConfig:
- LogType: DATA_READ


JSON
{ "bindings" : [
{ "role" : "roles/storage.admin",
"members" : ["users" : "mukul.kumar@natwest"]}
],
"etag" : "BEEFKJ3",
"version" : 3
}
Examples
YAML
bindings :
- members :
- user : mukul.kumar@natwest
role : roles/storage.admin
etag : BaEEFKJ3
version : 3
Policy limitations
• 1 policy / resource
• 1500 members or 250 google groups per policy
• 100 conditional role binding per policy
• Takes 7 mins to propagate across GCP.
Condn →
condition :
title : expiration access
description : No access after Jan'21
expression : request.time < timestamp ('2021-01-01T00:00:00.00Z')
OR
resource.type = < >
resource.name Starts with < >

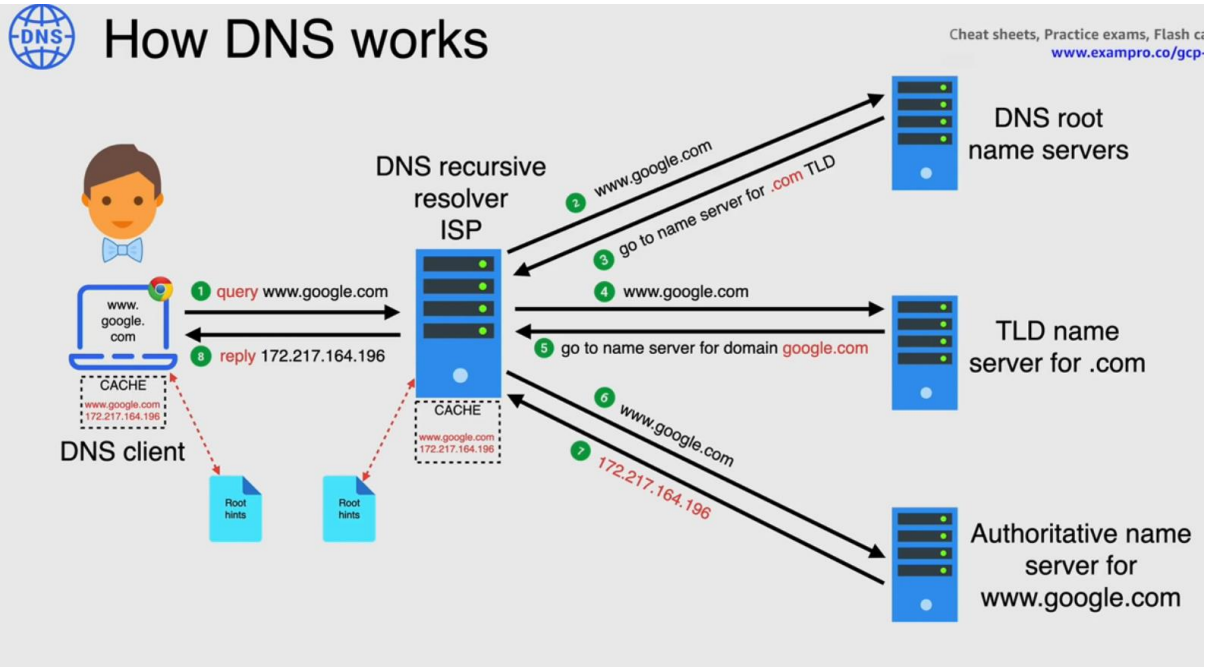
DNS
dns structure diagram -
root domain 13 root servers a to m across the world managed by iana

Tld - top level domain com / org/net /io / global like .com or country level like .ca/.uk.it

Second level domains - authoritative name servers google/Wikipedia …websites
sub domainof parent - cloud/ training /learn



DNS calls diagram
cache is handled by TTL param – time to live



DNS record types

Name server record , a name record, cname record, txt records , mx record for mail exchange , pointer record PTR record, soa record start of authority

Nat address translation nat

Translates local private ip to public ip before packet transfer ; public service only communicates through public ip
types – static : 1 to 1 using nat table

Dynamic 1 to 1 in pool of public address for reuse and efficiency ; nat table with public ip pool list

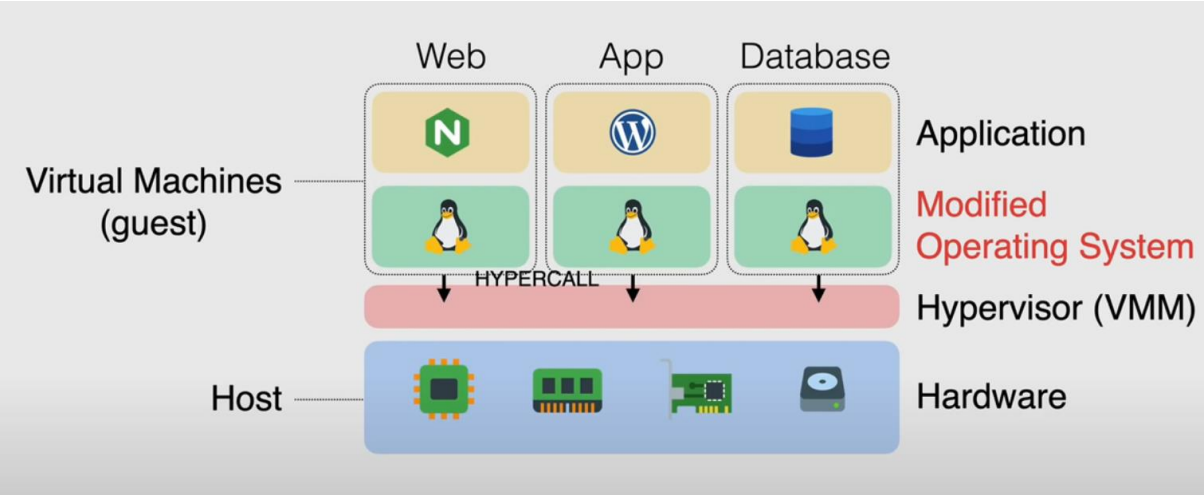
Pat – port address translation – multiple private ips top 1 public using port ; nat table has private ip ,port and public ip,port

Cloud DNS – GCP dns service

Host authoritative name servers - DNS as a service
host zones through managed name servers : public zone(visible on internet) as well as private zone(visible only within nw)

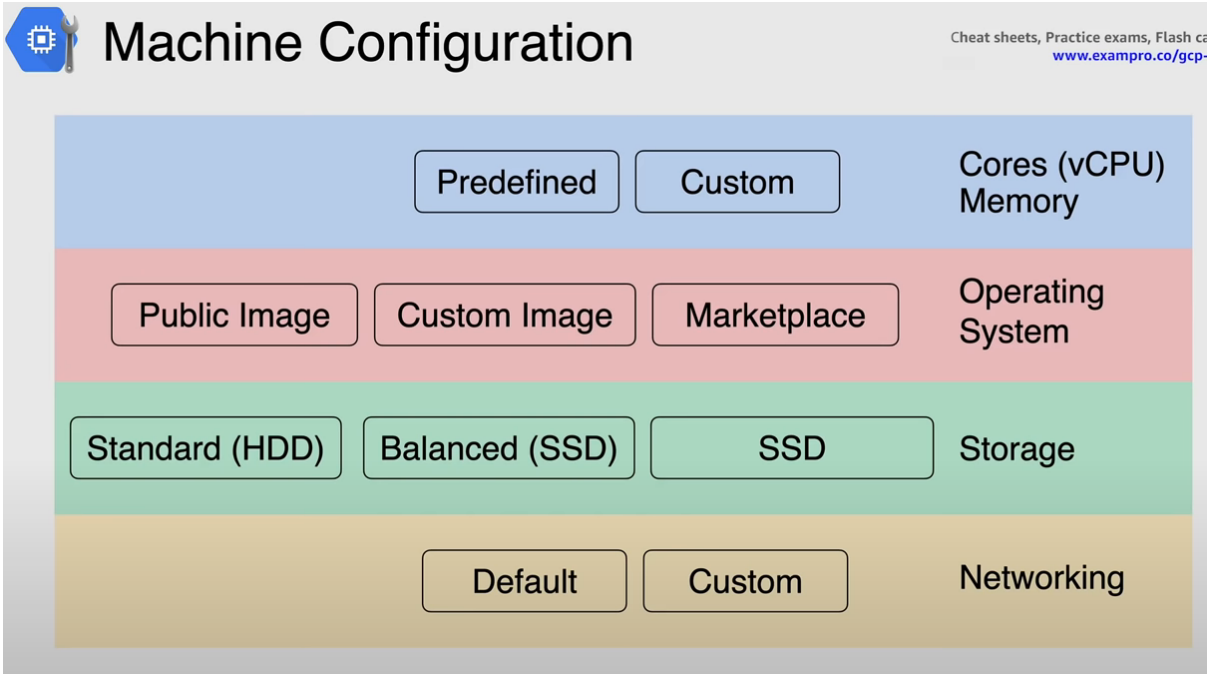
Compute Engine

Virtualization - running multiple OS on same machine
Hypervisor – virtual machine manager - enable multi os to run along side each other sharing same physical computing resources ; modified guest os makes hypercall to hypervisor which in turn communicates to kernel – this is called para virtualization



other types of virtualization -
Emulation virtualization – using binary translation
hardware assisted virtualization – access underlying hw
kernel level virtualization – kernel acts as hypervisor - GCP

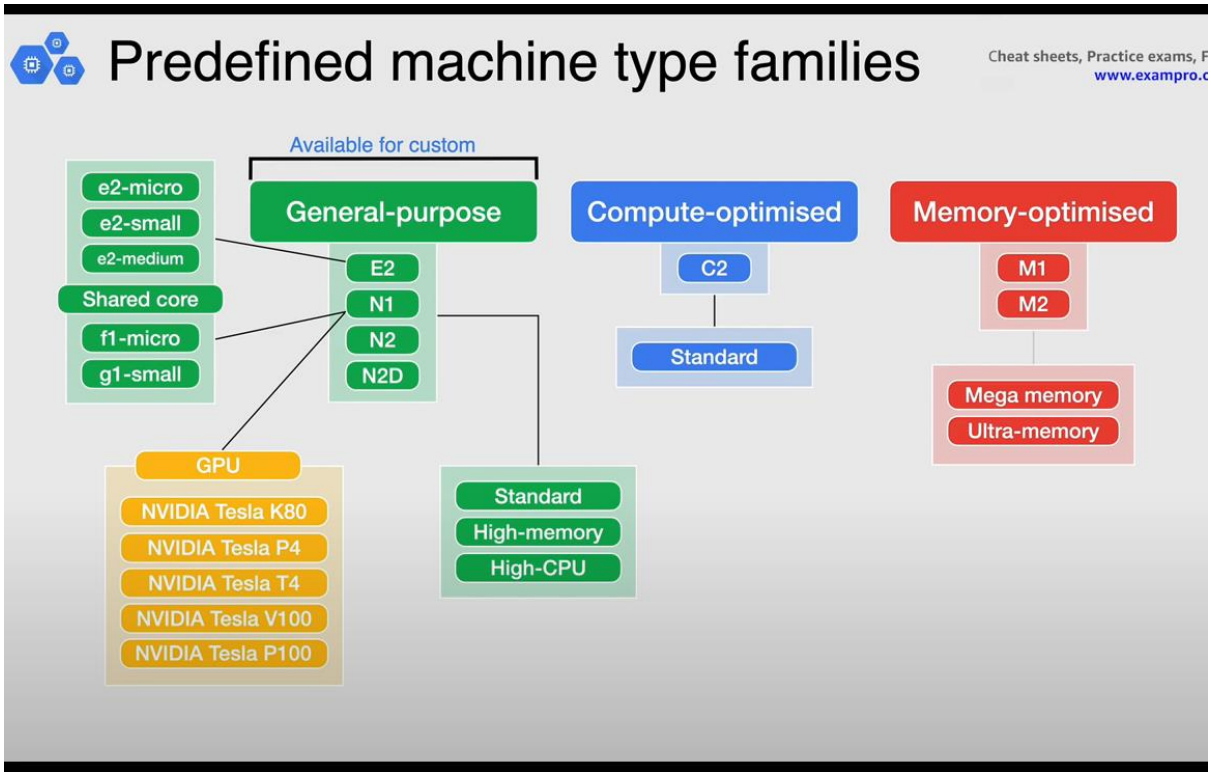
Compute engine
vm machine – instances (IasS) ; charged per second billing ; launched in a vpc network ;
can be multi tenant host or sole tenant node



1. Machine type – predefined/custom – General/compute/memory optimized
   intel/aMD
   vCPU – single hw hyper thread on CPU with nw throughtput of 2Gbps per vcpu
2. OS - public image – linux/ windows
   Custom image – sanpshots/existing disk
   Market place- os +sw eg wordpress on centOS7
3. Storage – performance vs cost
   Persistent disk - standard – spinning hard drive
   Persistent disk - balanced – solid state drive
   Persistent disk - ssd – fastest option
   local-ssd – physically attached (swap disk) – higher throughput, lower latency than pd , data persists until instance is stopped or deleted, used for fast scratch disk or cache, 2 flavors -sci and nvme
4. Network – auto default or custom across many available regions and zones
   ingress/egress fw rules – ip ranges/tags/instances
   n/w load balancer

New instance - name, label, region and zone, family (general purpose/compute optimised/mem optimised)– series and type (predefined) OR custom (set cored and memory), GPU, confidential vm service checkbox, container checkbox , boot disk (public/custom images/snapshots/existing disk) – OS/version and balanced/standard persistent disk , identity and api access , firewall config

Machine Types
format - eg . e2 – standard – 32 : series – type – vcpu count



E2 - day to day computing – web/app serving , back office applications, small/medium databases, microservices, virtual desktop , dev envs
2 to 32 vcpus , 0.5 to 128GB memory config available , standard/high mem/high cpu flavor

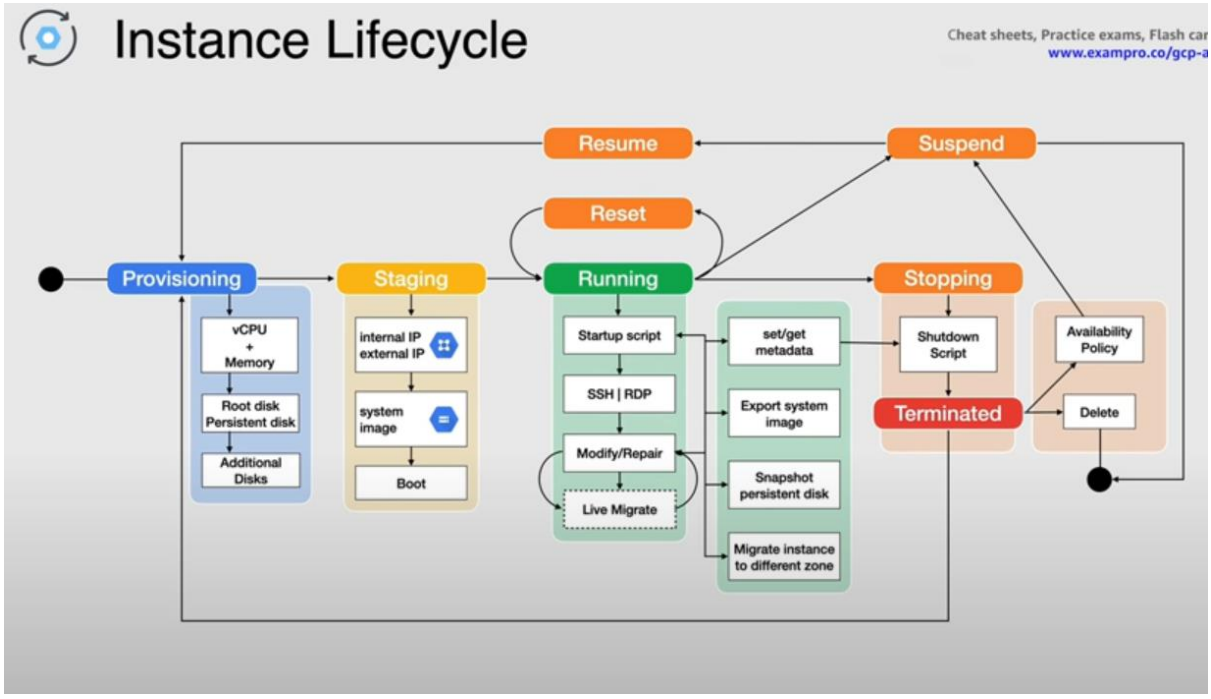
Shared core – cost effective , non resource intensive – 2VCPU and 1 to4 GM mem ; cpu bursting ;physical core available for short period of time
e2-micro/small/medium and n1-f1-micro/g1-small

N series – balanced price / performance – web/app serving , cache , media streaming , back office applications, medium databases
N1 – 2 to 96 vCPUs and 0.95 to 624GB mem , standard/high mem/high cpu flavor – GPU and TPU support
GPUs are NVIDIA tesla series
N2 – 2 to 80 vCPUs and 0.5 to 640GB mem, standard/high mem/high cpu flavor – Higher per thread performance and higher clock freq
N2D - 2 to 224 vCPUs and 0.5 to 896GB mem, standard/high mem/high cpu flavor – high mem to core ratio, largest gen purpose machine type

C2 – ultar high perf for compute intensive workloads, EDA electronic design automations, Gaming
4 to 60 vCPUs and 16 to 240GB mem – Highest performance per core

M series – ultra high mem workloads; large in mem DB like sap HANA; in mem analytics
M1 – 40 to 160 vCPUs and 32 to 3844GB mem , ultramem/megamem cpu flavor
M2 – 40 to 1600 vCPUs and 32 to 11776 GB mem, ultramem/megamem cpu flavor

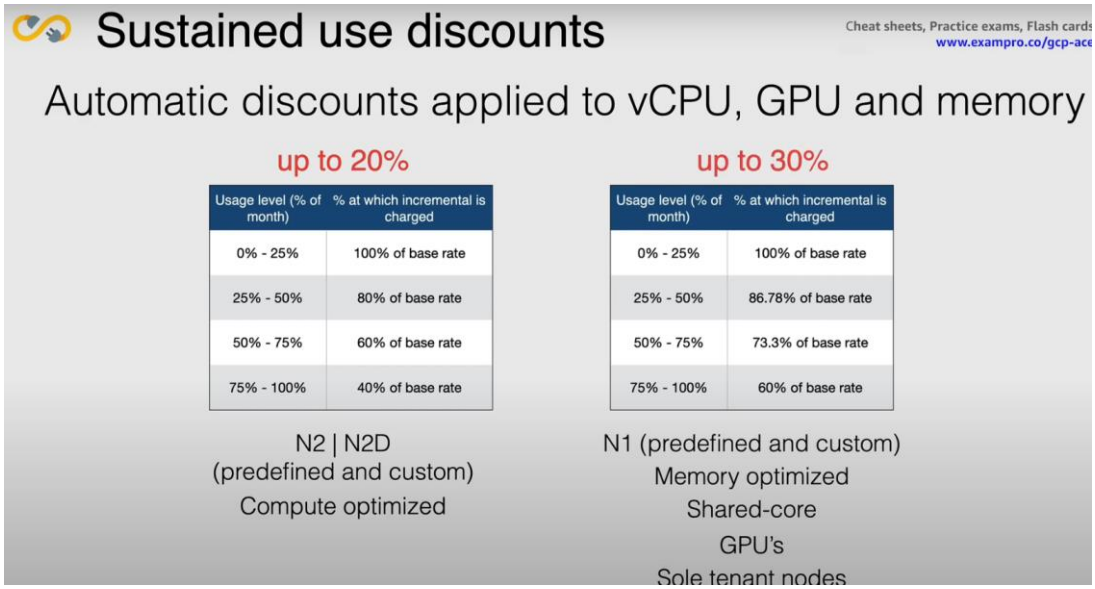
Custom machine types
E2/N2/N2D/N1+GPU - standard/highmem/highcpu – 2 to 224



Provisioning - $:none
Staging – $:none ; Boot – shielded VMs – for added security, process- secure boot-> measured boot ->Integrity monitoring
Running - $$, static IPs ,disks : reset and repair can happen ; Metadata – setup guest environment using key value pairs; start-up script – add as key in metadata and value as script of linux commands eg. apt- get apache3 and this will be setup at time of starting up VM ;Live migration – w/o shutting down VM, can be migrated to diff zones within same region.
suspend/stopping -$$, static IPs ,disks
Terminated - $$, static IPs ,disks

VM access
ssh linux – fw rule – allow tcp:22 from gcp console/cloudshell using cloud SDK
rdp windows – fw rule – allow -tcp:3389 ; or from powershell terminal(fw rule-tcp:5986); rdp chrome extension or 3rd party rdp client ; requires setting windows password

Compute engine billing – resource based – each vCPU and each GB of mem billed separately charged by the second with min of 1 min;



Committed use discount - 1yr/ 3yr 57%(most resources) to 70%(mem optimised machines)
Preemptible VMs – 80% cheaper at fixed price; not always available and max run duration 24 hr; used for fault tolerant applications eg. batch processing

--GCP STORAGE –

Types --BLOCK (data files split into uniquely identifiable evenly sized blocks delivered via spinning hard drive or ssd - high perf storage) / FILE (traditional network file storage nfs – mountable **directory tree** structure- gcp: file store)/OBJECT (unstructured data eg. movies songs etc with metadata and unique ID, infinitely scalable)

Persistent disks – independent nw storage which is resized while running, expandable upto 64TB , encrypted – zonal(better speed)/regional(better availability) configs ; standard(pd-standard, backed by standard hard disk drives {hdd}, large data processing workloads that use sequential i/o, cheapest)/balanced (pd-balanced, balanced perf and cost, gen purpose mid-price)/ssd(pd-ssd, fastest , highest price enterprise applications and high perf DB that demand lower latency) flavor

Persistent disks snapshots – backup and restore PD, global resource across zones and regions ; stored in cloud storage , incremental backups , can be scheduled , retention policy and source disk deletion rule can be setup ; can be used to create new instances ; alternative to snapshots is disk image

--DEPLOYMENT MANAGER--
IaC infrastructure as code ; using yaml/pyhton/jinja code templates ;
Config :: type - api.version.resource / name / properties : zone, machine type,dis, params

```
resources:
- type: compute.v1.instance <-------- base type
  name: instance-1
  properties:
    zone: us-east1-b
    machineType: https://www.googleapis.com/compute/v1/projects/PROJECT_ID/zones/us-east1-b/machineTypes/f1-micro
    disks:
    - deviceName: bootdisk
      type: PERSISTENT
      boot: true
      autoDelete: true
      initializeParams:
        sourceImage: https://www.googleapis.com/compute/v1/projects/debian-cloud/global/images/family/debian-9
```

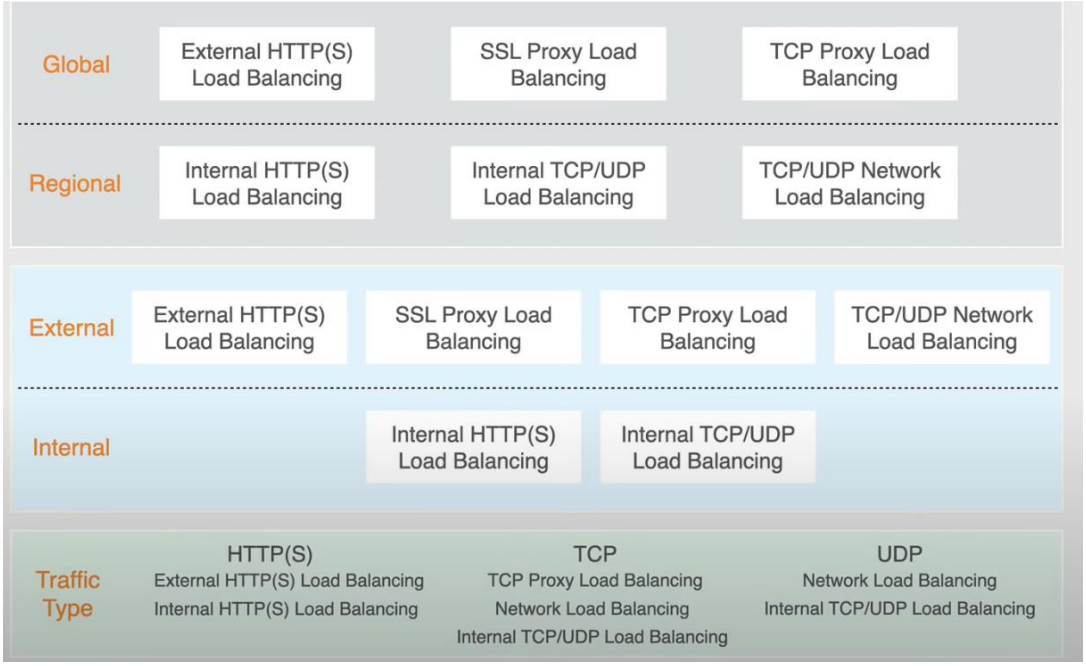
Templates can be imported ; eg cloud deployment-manager deployments create bowtiedeploy –config bowtie-deploy.yaml ; deployment manager api needs to be enabled

-- High Availability and Autoscaling --

LOAD BALANCING

Distributes user traffic across multiple instances ; fully distributed and s/w defined ; backend services decided how load is distributed based on health checks, session affinity, service timeout , traffic distribution- readiness, target capacity and capacity scaling , backends – instance groups available ; global and reginal ;; external(traffic from internet) and internal (traffic from gcp resources);; protocol based configs – https,tcp and udp

Types



Https LB (layer 7 LB, ssl enabled for encryption; IPV4/6 supported {v6 is terminated and served as v4 to backend}) traffic management types : cross-region LB – distributes on basis of location based on forwarding rules created URL maps eg, traffic redirecting to closest targets where the request is originated from ;;; content-based LB – distributed on basis of content type eg. backend service: video to us-west2-b and backend service: images to us-east1-b

SSL Proxy – reverse proxy LB to distribute ssl traffic(client ssl sessions terminated at LB) ; distribution only by location ; IPV4/6 supported {v6 is terminated and served as v4 to backend} ; used for other protocols that use ssl Websockets and IMAP over ssl

TCP proxy LB - reverse proxy LB to distribute tcp traffic(client tcp sessions terminated at LB) ; location based distribution ; layer 4 LB intended for non http traffic , IPV4/6 supported {v6 is terminated and served as v4 to backend}

N/W LB – pass through LB distributes tcp and udp traffic to vms; response from backend go directly to client ; traffic distributed by protocol, scheme and scope , support traffic on ports not supported by tcp and ssl proxy

Internal LB – layer 4 lb for gcp resources traffic between instances, supports either tcp or udp not both

INSTANCE GROUPS

Collection of VM to manage as single entity ; managed (MIG)and unmanaged (UIG)instance groups ; created from instance template ; input – min and max no. of instances

MIG – used for stateless serving backloads(website front end , web servers and web apps), stateless batch (high perf compute workloads)and stateful workloads(DB, legacy applications etc.) – features: auto healing(keeps vm running , recreate not running vm when frozen or crashed) , regional (multi zonal for HA), LB(serve traffic and health checks), autoscaling (dynamically add/remove instances) , auto updating(sw versions and deployments)

UIG – just LB

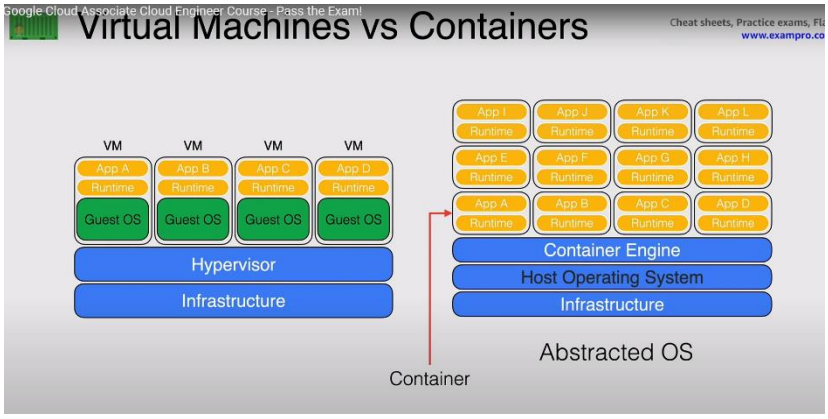
Instance templates – create vm and mig ; defined machine type, boot disk image, container image and label; existing instance template cant be updated

-- Kubernetes Engine Containers—

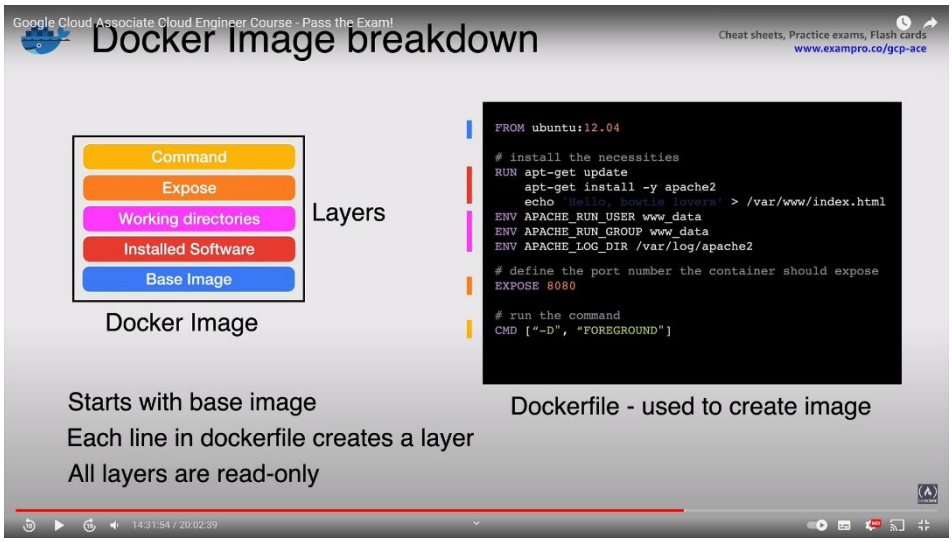
Containers -

Containers

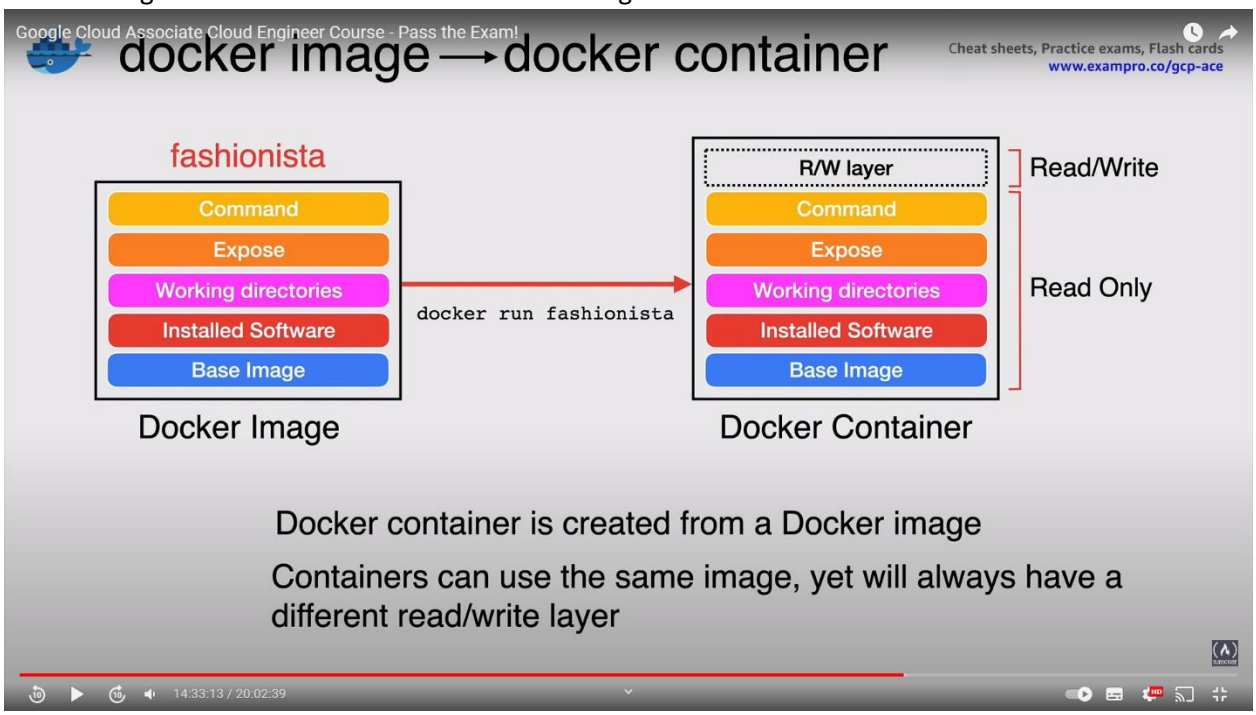
Consistent, efficient, standardized



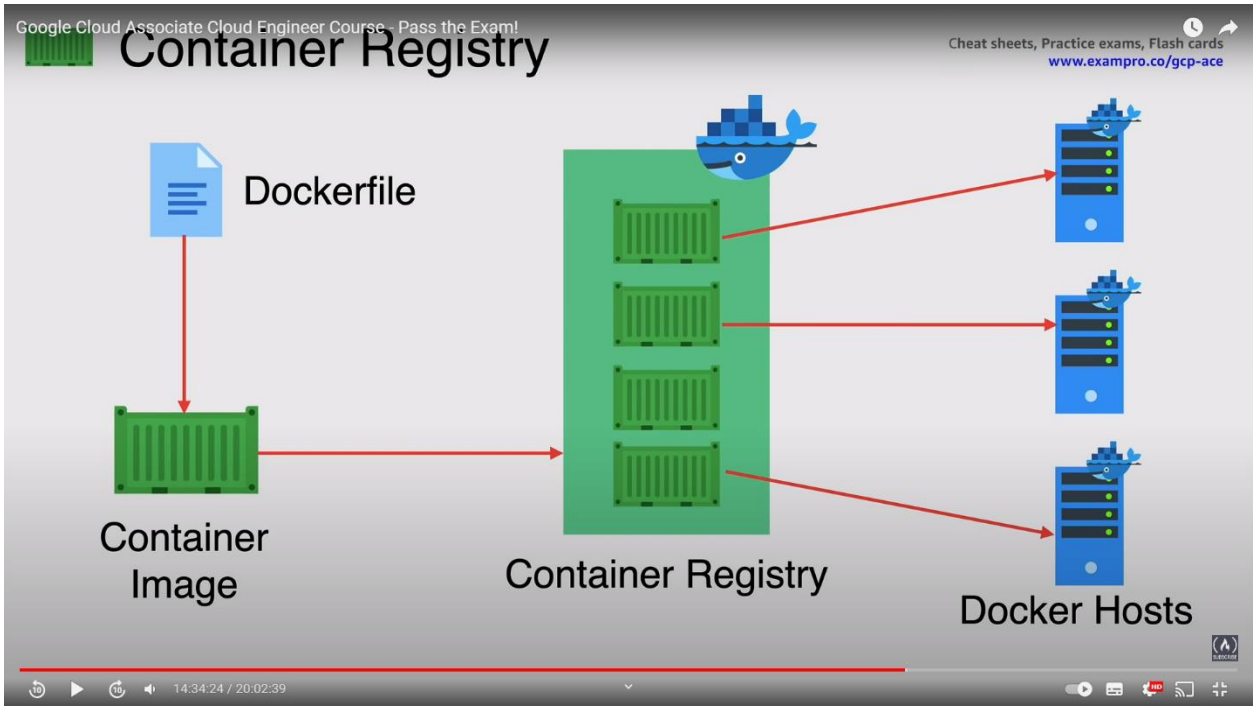
Docker – container engine



Docker Image to docker container – executable image



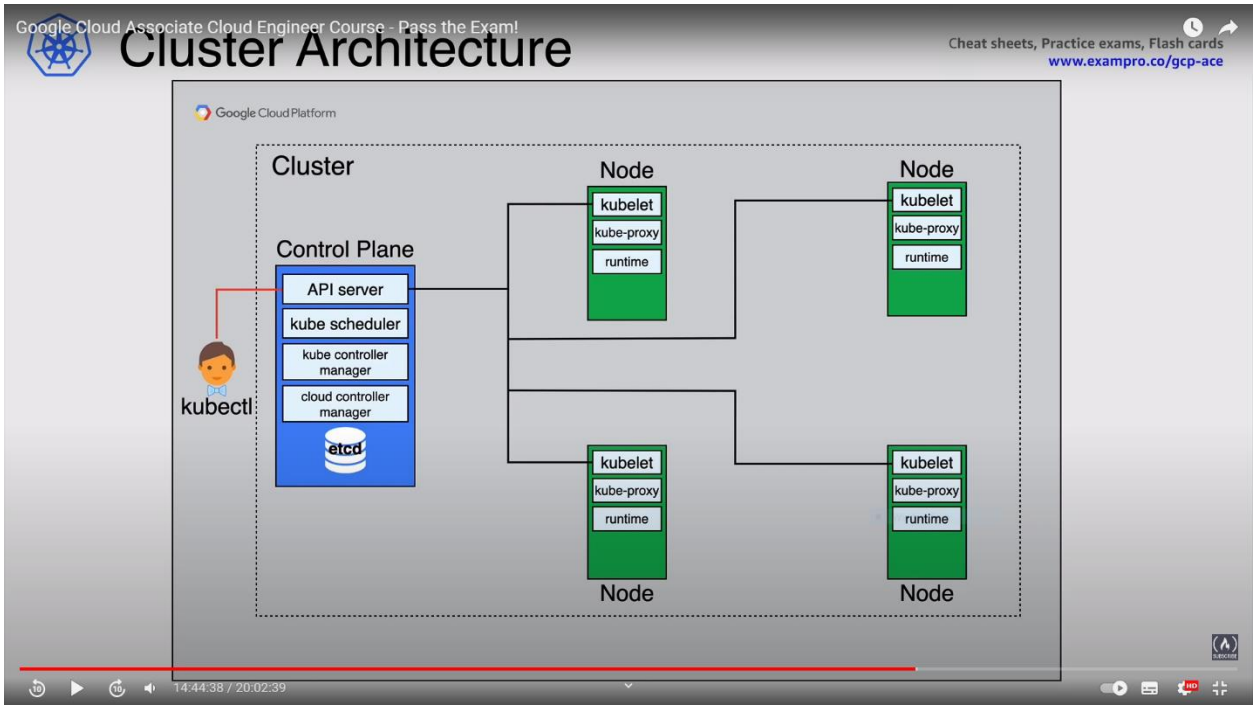
After image is created , stored in container registry – Docker Hub ; which is run on Docker hosts – machine



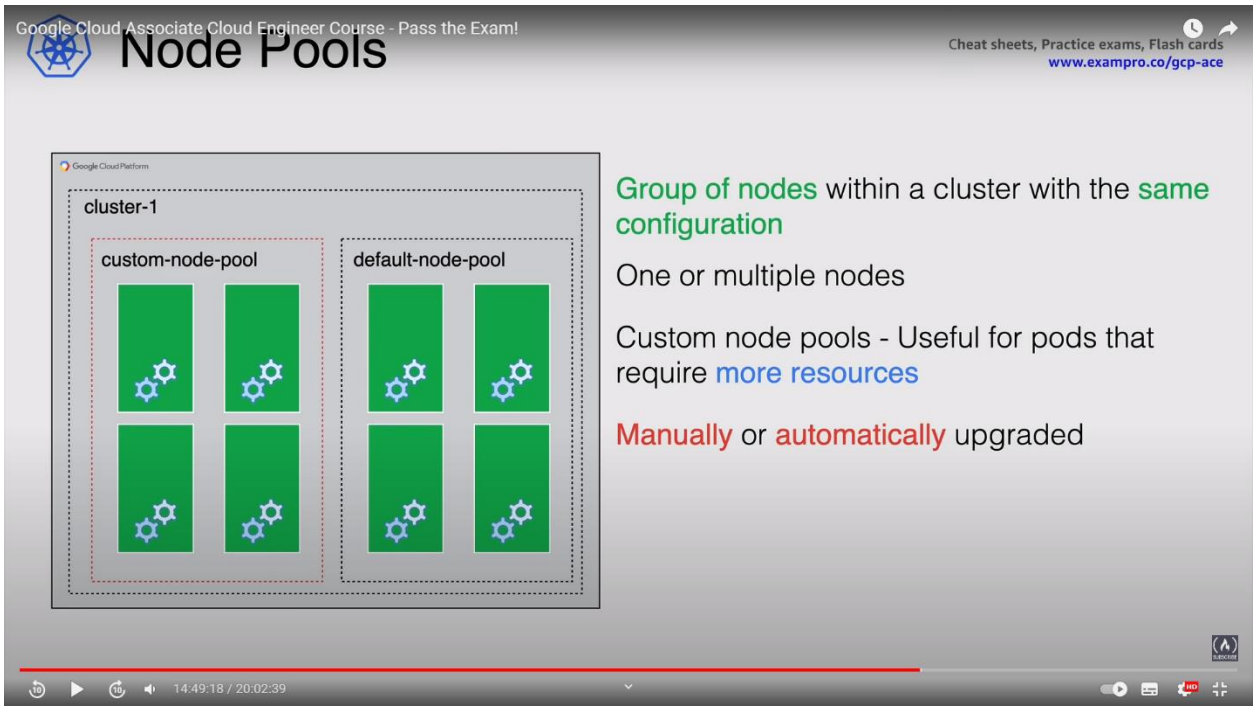
Kubernetes –Orchestration platform for containers ; run containers on clusters of physical or virtual machines

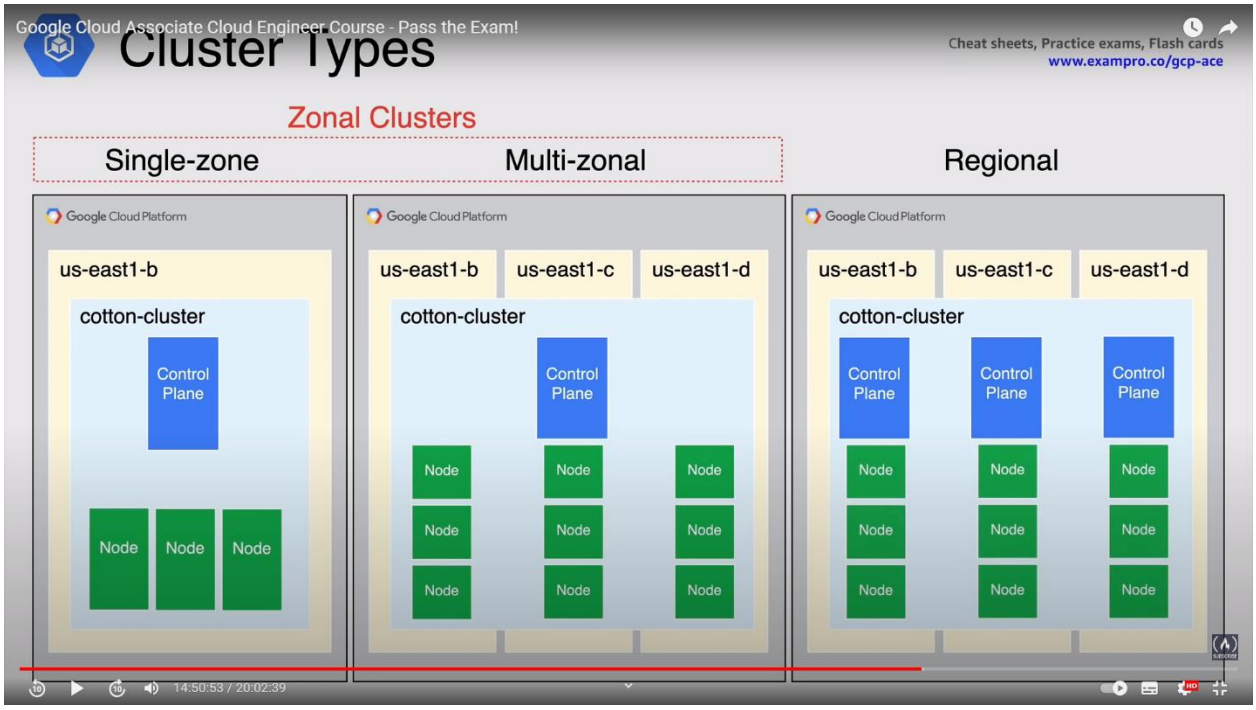
GKE – Running kubernetes in Google cloud instead of on-prem kuberenetes - google's kubernetes service for deploying managing and scaling your containerised application using google infra – compute engine instances grped together to form cluster ; integration with docker for containers created from docker image; other features – cloud load balancing , node pools, autoscaling , upgrades , repair , logging and monitoring

Cluster Architecture –
control pane : atleast one, responsible for coordinating with entire cluster, responsible for scheduling and mgmt of workloads, unified endpoint of cluster, consist of API server{point of interaction with nodes- api calls or kubectl}, Kube scheduler {discover or assign newly created pods}, Kube controller manager {Runs all controller processes}, Cloud controller manager {runs controller specific to cloud provider} , etcd {key value store to store state of cluster}
nodes- one or more, run containerized apps, responsible for docker runtime , consists of kubelet {agent for communication with control pane, starting docker container}, kube proxy {network connetivity}, runtime{runs container}

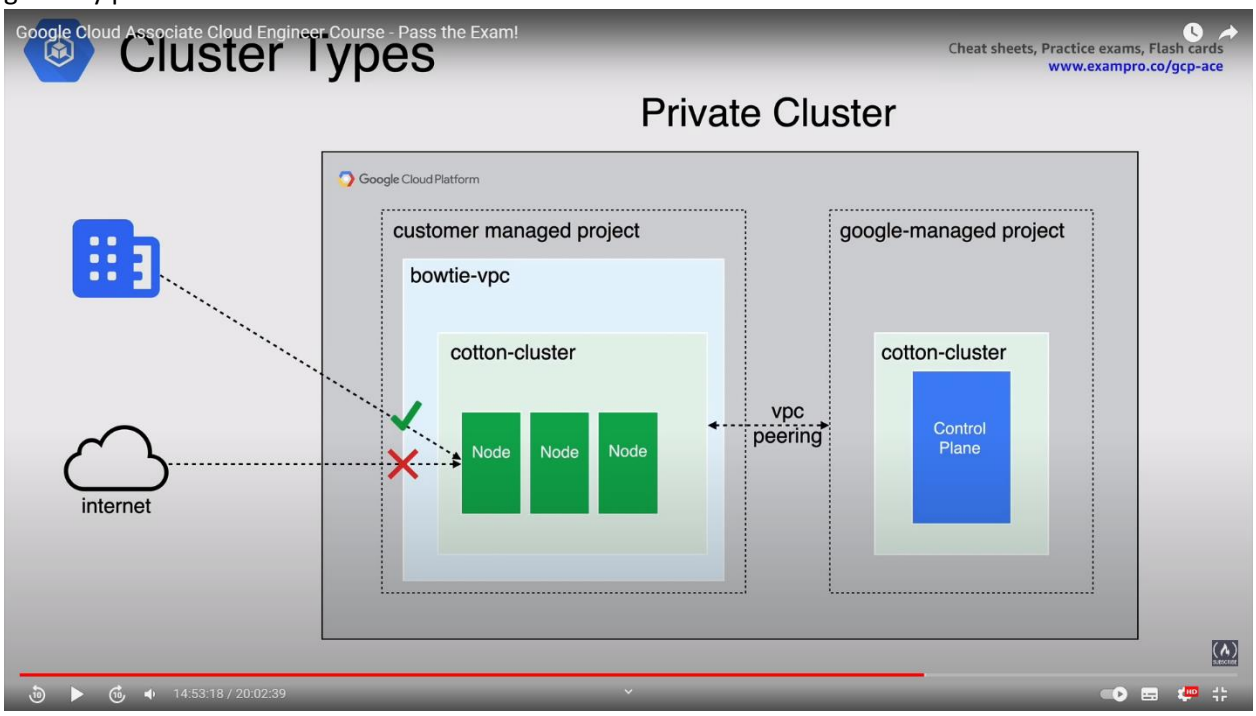

Google Cloud Associate Cloud Engineer Course - Pass the Exam!
Cluster Architecture
Cheat sheets, Practice exams, Flash cards
www.exampro.co/gcp-ace
Google Cloud Platform
Cluster
Control Plane
API server
kube scheduler
kube controller manager
cloud controller manager
etcd
kubectl
Node
kubelet
kube-proxy
runtime
Node
kubelet
kube-proxy
runtime
Node
kubelet
kube-proxy
runtime
Node
kubelet
kube-proxy
runtime
14:44:38 / 20:02:39

Node pools -


Google Cloud Associate Cloud Engineer Course - Pass the Exam!
Node Pools
Cheat sheets, Practice exams, Flash cards
www.exampro.co/gcp-ace
Google Cloud Platform
cluster-1
custom-node-pool
default-node-pool
Group of nodes within a cluster with the same configuration
One or multiple nodes
Custom node pools - Useful for pods that require more resources
Manually or automatically upgraded
14:49:18 / 20:02:39



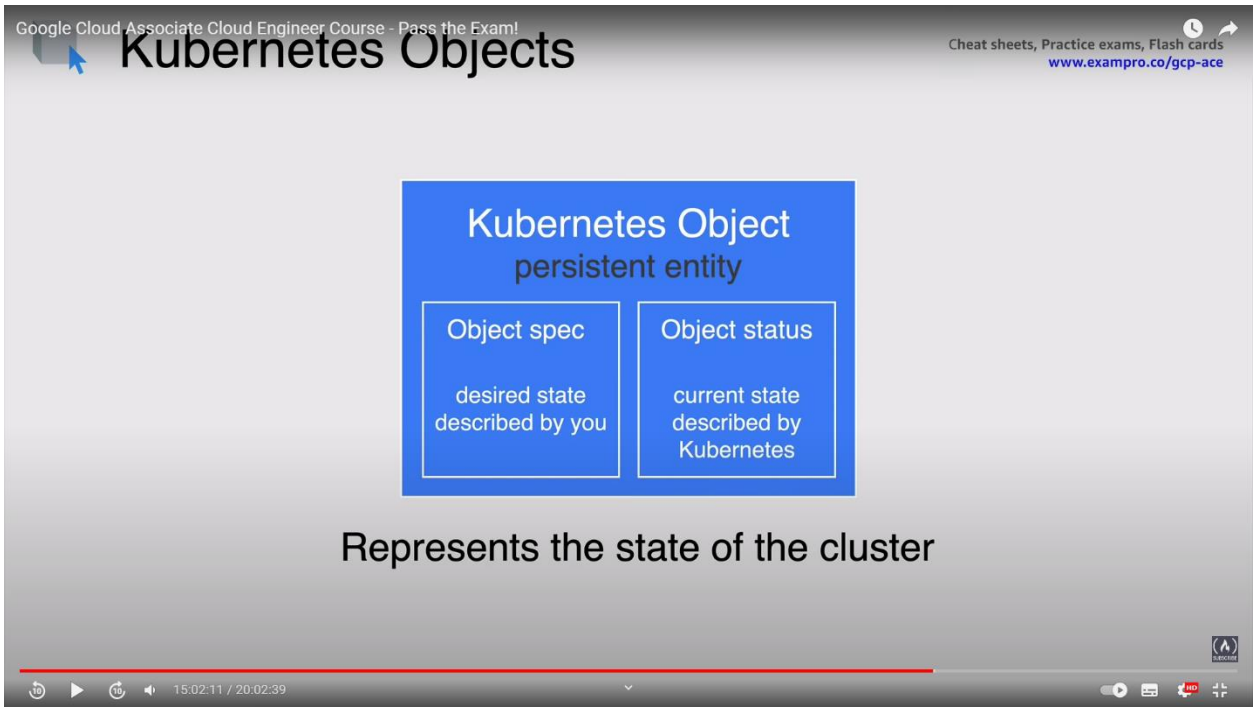
Private clusters – isolation from public internet using internal ip addresses, cloud nat or own nat gateway powered



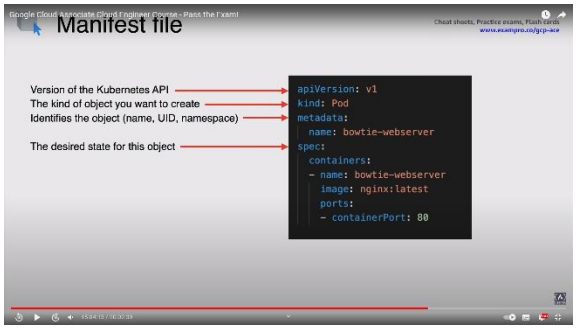
Cluster version - 2 options – Specific version for kubernetes for a given payload - static
Or Release channel for auto upgrade (google managed, best practice) of cluster and nodes – Rapid channel {few weeks after open source Kubernetes releases an upgrade} , Regular channel {default, 2-3 months after releasing in Rapid} , Stable channel {2-3 months after releasing in Regular}

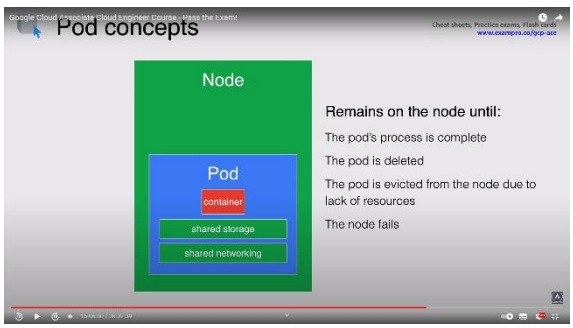
Control plane and nodes donot always run same version at all time, control plane is upgraded before its nodes , zonal cannot launch or edit workloads during upgrade , regional each control plane is upgraded one by one , manual upgrade cannot upgrade more than 1 minor version at a time, surge

upgrades – control no of nodes GKE can upgrade at a time , controlled by params- max_surge_upgrade {more parallel upgrades} and max_unavailable_upgrade {more disruptive}

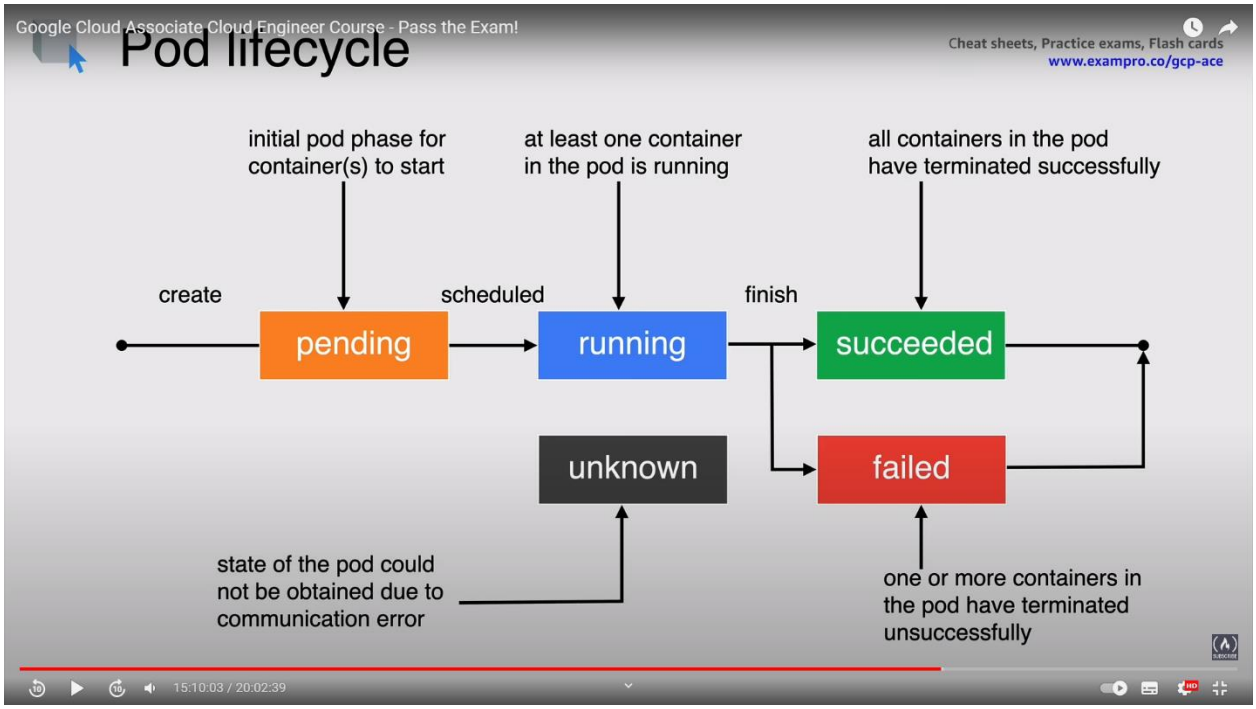


Objects can be pods{smallest, basic deployable objects, represents single instance of running process in cluster, pods control multiple docker containers with shared resources}, Each object has name and uid inside manifest file-yaml/json

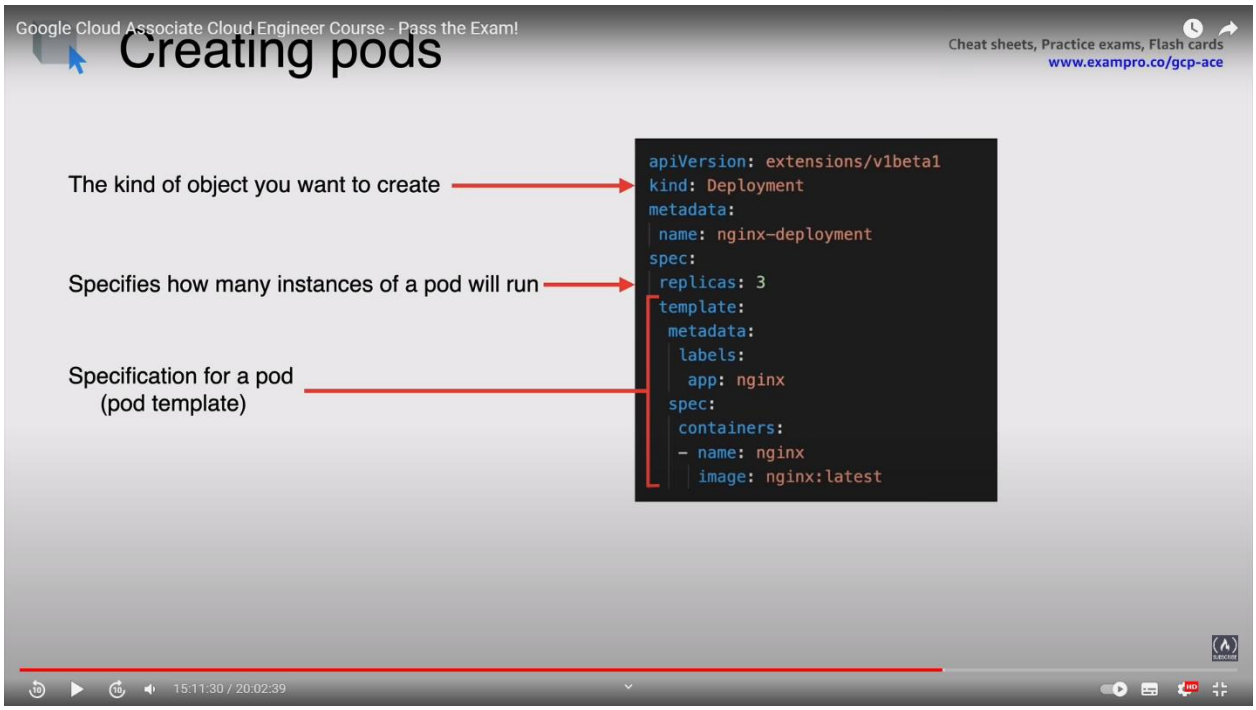




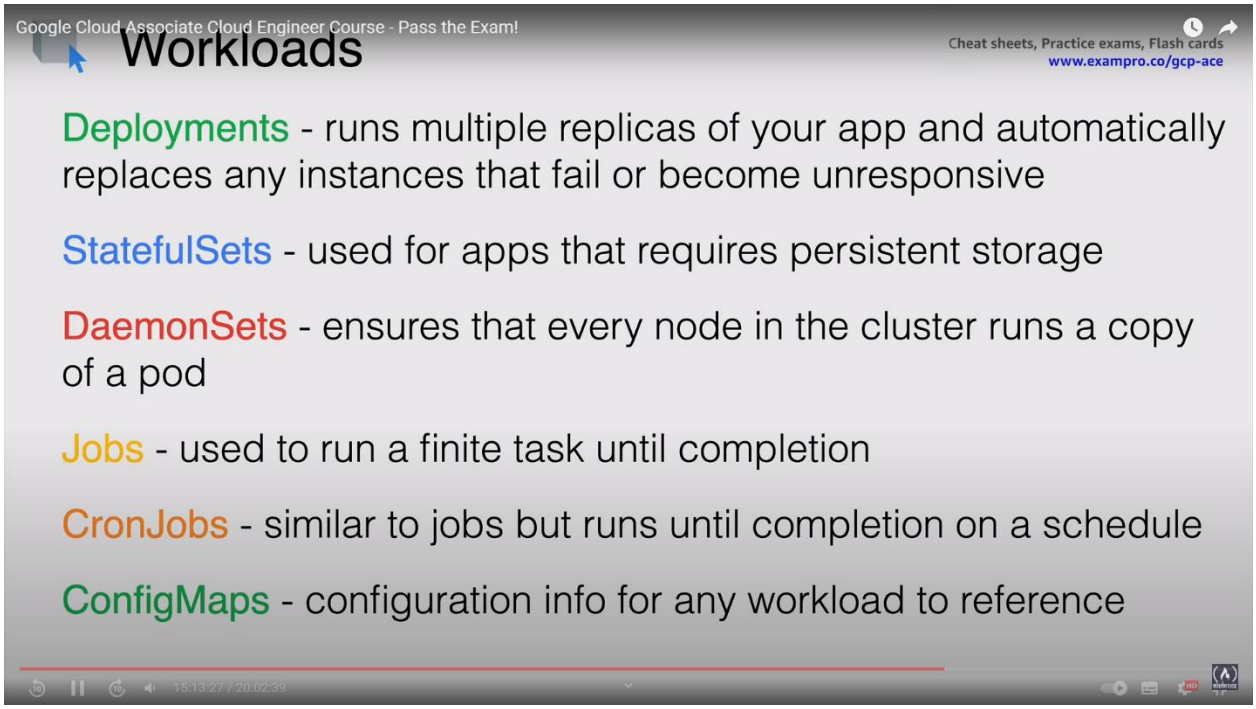
Namespaces , labels are other componnents of kubernetes objects



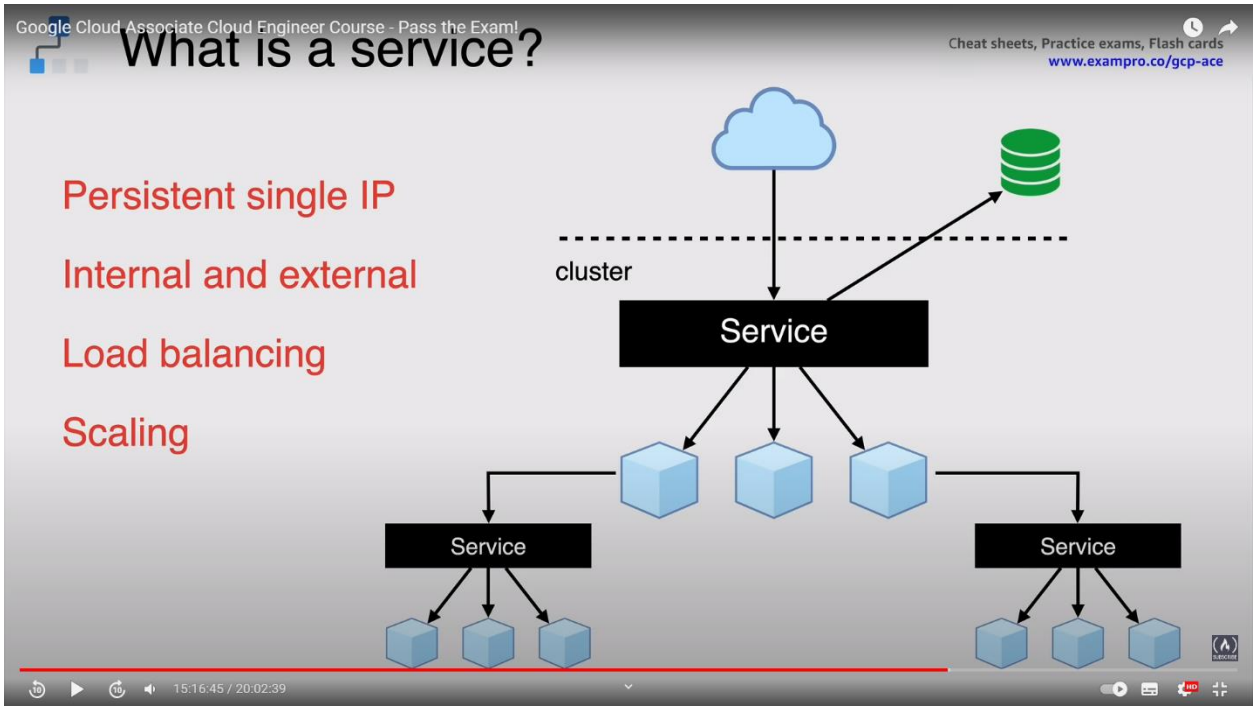
Pods are created using deployments(a workload object type) objects via pod template, manages replica sets



Other workload types



Kubernetes Services –interface how pods communicate with each other amd route traffic to other services, service use selectors to identify which pods they should control – types: clusterIP{default},node port{static port from predefined range}, load balancer, ingress, multiport services, external name, headless



Health checks can be configured using default and inferred health check params defined by using backend config custom resource definition (crd) ; ssl certificates - google managed / self-managed {provision own certs,shared with google cloud} / self managed as secrets {provision own certs}

GCP cont kubernetes service –

Service ip address is static while pods ip address is ephemeral

Key value pair under the selector in services.yaml should match with key value pair under labels under meta-data in deployment.yaml in order to route traffic from service to pods managed by deployment

TYPES of Service

Cluster IP: default , has protocol , port no(host port) exposed internally in cluster and target port that containers are listening on in service . yaml

Node port : static node port from 30000 to 32767 , has appln hosted reached by node ip:node_port , has port no. exposed internally in cluster , port the container is listening on, nodeport within range

Load balancer : for connecting to cloud's load balancer, has port no. exposed internally in cluster , port the container is listening on

Multiport services : multi appln(in container) within same pod exposed by diff ports , similar config as load balancer service type with as many port definition as no. of containers

External name : has internal dns name redirected to an external dns name

Headless : used where no no lb or routing mechanism is needed

INGRESS FOR GKE - - GCP load balancer forwards to ingress which forwards to services which forwards to pods

GKE Storage Options:
DB- Cloud SQL, CloudSpanner , Datastore
NAS – FileStore
Object Storage – CloudStorage
Block Storage – Persistent disk

Volumes – persistence storage for docker containers
Types:
emptyDir – empty directory that containers in the pod can read and write from
ConfigMap – to inject co0nfiguration data into pods
Secret – sensitive data availability to pods
Downward API – make downward api data available to applications
Persistence Volume Claim – provision durable storage to be used by applications , req specific size , access mode{readwriteonce / readonlymany / readwritemany by nodes} and storage class from persistence volume {persistence disk – regional or zonal}

Demo : check default vpc is created -> search for kubernetes engine -> create cluster : name, location – regional/zonal , master version – static for gke or release channel – rapid/release/ stable -> node pool : name, no. of nodes , autoscaling – min and max nodes -> nodes: image type , machine

config , type , boot disk size -> networking and subnets -> monitoring
Can also be done by CLI by gcloud container clusters create
Use kubectl to manage the cluster
clone git repo for application code -> use google's cloud build to build image in google cloud container registry-> gke console deploy tkes image from google container registry ->creates deployment -> expose button creates a service - port, type,name-> gives accessible ip
Rolling update can be scheduled by rebuilding image and using rolling update feature from gke console which makes incremental update of pods

HYBRID CONNECTIVITY – Cloud VPN and Cloud Interconnect and others {direct peering , cdn interconnected }

Cloud VPN

Connecting another nw{ other cloud provider or on prem nw or other vpc } to Google cloud via an IPsec VPN connection (tunnel over public internet encrypted by one vpn gateway and decrypted by another vpn gateway) , when public internet access is needed , peering location is not available
Regional service , site to site only (personal laptop vpn not supported) , upto 3Gbps per tunnel ,dynamic and static routing ; cheaper option
types : 1. Classic VPN{99.9%SLA, static and dynamic routing, 1 ext ip addr for a single interface, depreciated } 2. HA VPN {99.99% sla, only dynamic routing, 2 ext ip configured for 2 interfaces , new default vpn}

CLOUD Interconnect

Low latency , highly available, btw on prem and google cloud vpc nw ; directly accessible internal ip address using private google access ; not traversing through public internet but with a dedicated connection ; not encrypted ; high price ; upto 200Gbps
TYPES : 1. Dedicated interconnect {direct physical connection btw own nw and googles nw - } 2. Partner interconnect {via a supported service provider }

DIRECT PEERING
btw on prem nw and google's edge nw ; available in 100 location in 33 countries ; direct peering connection with google is free

CDN intereconnect
enables third party cdn providers to establish direct peering links with google's edge nw via direct traffic from vpc nw to providers nw , eg, cloudflare or verison

SERVERLESS SERVICEs – APP engine and CLOUD Functions

APP ENGINE

PaaS service , fully managed serverless platform to develop and host web applications ; can run in containers or code python java node.js php go .net ruby ; supports autoscaling and versioning for rollbacks, migration and traffic splitting ; supports connection to ext storage ;
2 env types standard { apps run in sandbox env with specific version of runtime for free or very less cost based on instance hours price ; designed for sudden or extreme traffic spike } and flexible { Apps run in docker containers with any version of runtime used which is designed for consistent traffic; no free quota and pricing depends on VM resources which are managed }
Managed Instances – creates or shuts down instances by specifying no. of instances to run ; Scaling Type – Auto-scaling {based on metric like req rate & response latencies} , Basic scaling {create instances when appln receives req} , Manual scaling {specifies no. of instances that continuously run}
Deployed via app.yaml file to upload file on cloud storage and create service

Cloud Functions

FaaS, supported runtime – Python , Java , node.js , go , .net ; event driven by triggers – http/ pub-sub/cloud storage ; billings depend on time and provisioned resources , free upto 2 mil invocation settings can be done for auth , n/wing , amt of mem etc. ; 1 trigger per function; source code is stored in cloud storage bucket as zip file ; cloud build then builds code into container image and pushes to container registry ; cloud function grabs image from registry and passes along with event data to instance for processing ; functions are stateless
types – http functions { http standard req get post put} and background functions {events from gcp infra}

Storage Services - Cloud Storage and CLOUD SQL and Cloud Spanner and NoSQL DBs

Cloud Storage

Consistent, scalable and large capacity , highly durable object storage – not file or block ; used for data files , text files , photos and videos ; excels for content delivery , big data sets and backups ;
Used using BUCKETS and OBJECTS
BUCKETS : basic container to control access and organize data ; globally unique name ; geo locations – regional / dual region / multi region ; storage class – standard {max availability for hot data , no min storage duration , used for analytical workloads, high storage price and no retrieval price}/ nearline{infrequent accessed hot data , 30 day min storage duration , for data backup and archiving , lower data storage cost but higher data retrieval cost than standard } /coldline{very infrequent cold data , min 90 day storage duration ; data backup and archiving ; lower storage cost but higher retrieval cost than nearline } / archive {lowest cost archival storage for cold line data ; min 365 day storage duration , cold storage data and disaster recovery , lowest storage cost but highest retrieval cost, lowest availability }; access –uniform {iam} /fine grained{iam with acl}
Retention policy – min duration that bucket's obj are protected from deletion
OBJECTS: files inside buckets ; compo – obj data and meta data ; immutable ; lifecycle management using setting time to live , downgrading storage classes and archiving non current versions by specified conditions and rules

CLOUD SQL

GCP DBaaS DB as a service , fully managed RDBMS, low latency transactional relational db workflows , Mysql, postgresql and sql server, read replicas supported from primary instance – which can be promoted as primary if primary corrupts in case of planned regional migration or unplanned disaster recovery , HA , two types of backup – on demand {persist until u delete} and automatic{in backup window} , supports point in time recovery, 30 tb storage capacity with auto increase , both at rest and in transit encryption , billing based on instance, persistence disk and egress traffic , ssd or hdd flavours , connection via cloud sql proxy

Cloud Spanner

Funny managed relational DBaaS , strongly consistent and horizontally scalable with node redundancy , supports schemas, ACID transactions and SQL queries , globally distributed , handles replcias and sharding{db split} ,synchronous data replication , HA, data layer encryption , audit logging and iam integration , designed for financial services, ad tech , retail, gaming and global supply chain

NoSQL DBs

Cloud Bigtable –Fully managed , wide column noSQL db designed for terabyte to petabyte scale workload that offers low latency nd high throughput , built for real time app serving & large scale analytical workloads , regional service ,automated replication , single keyed data in resizable cluster nodes, high priced, integration with big data tools like hadoop, use cases – time series data, marketing data, financial data , iot data, graph data
Cloud Datastore – fully managed highly scalable nosql document db for auto scaling , high performance , HA, atomic transactions, SQL like language gql , datastore emulator to provide local emulation of prod datastore
Firestore for Firebase{mobile app dev platform} – flexible scalable nosql , for client and server side development , data organized in data , document and collection , available for ios , android , c++, node , python , java ; serverless, multi region replication , expressive querying , realtime updates , secure , free tier
MemoryStore – Fully managed service for either redis or memcached in-memory data store to build application caches , HA , secure , up to date, use cases- caching , gaming , stream processing

BIG DATA and MACHINE LEARNING SERVICES

BIG prefix – analytics

FIRE prefix – desktop and mobile application

Data services :

BigQuery – Fully managed , petabyte scale , low cost data warehousing , on demand pricing charged for bytes read, streaming and batch ingest using free bulk loading data transfer service , use cases – real time analytics , HA, automatic backup and restore , standard sql , big data ecosystem integration , data encryption

Pub/Sub – Fully managed real time messaging service to exchange message btw independent applications, publisher publishes a message for a particular topic stored in message storage & subscriber creates a subscription on a topic to receive message

Composer – fully managed workflow orchestration service build on apache airflow

Cloud Dataflow – Fully managed processing service for executing apache Beam pipelines for batch and realtime data streaming ; reads data from source , transforms and write to sink ; serverless

Dataproc - Fully managed Spark and Hadoop service to replace with on prem Hadoop env

Datalab – interactive tool for data exploration , analysis , virtualization and ML ; collect data-organize data-create model – train model – deploy trained model

Dataprep – serverless for cleaning , preparing structured and un structured data for analysis , reporting and ML

ML services :

Vision API – for sight – pre trained machine learning model that allows you to assign label to images and quickly classify into predefined categories

Video intelligence - for sight - pre trained machine learning model to automatically recognize a vast no of objects and places in stored and streaming video

Natural language – for language – derive insights from unstructured text using google ML

Translation – for language – dynamically translate btw languages using googles pre trained or custom ML models

Dialog flow – for conversation – natural language understanding platform to design and integrate conversational user interface

Speech to text and text to speech – for conversation

Operations Suite

Suite of tools for logging {central repo for multiple source log collection, real time mgmt. and export, log viewer , retention policy, types- audit logs, access transparency logs , agent logs } , monitoring{collects metrics to provide insights dashboards and charts , workspace are needed to use cloud monitoring , alerting } and application diagnostics , error reporting, application performance management – debugger , trace {latency data}, profiler {resource consumption} ; available for gcp and aws and on prem env , vm monitoring with agents

Lesson 1 - cloud fundamentals

AWS services

Compute services

- Amazon EC2
- Amazon Elastic Container Service
- AWS Lambda - serverless

Storage

- Amazon EBS (Amazon Elastic Block Store)
- Amazon EFS (Amazon Elastic File Service)

Analytics

- Quick Sight
- Athena
- Redshift – data warehousing

Application integration

- Simple Queue Service (SQS)
- Simple **Notification** Service (SNS)

Cost management

- AWS Budgets

Database management services

- MySQL
- Oracle
- SQLServer
- DynamoDB
- MongoDB

Developer tools

- Cloud 9
- Code Pipeline – Continuous Integration

Security services

- Key Management Service (KMS) – Data encryption
- Shield
- Identity and Access Management (IAM)

Additional Services

- Blockchain
- Machine Learning
- Computer Vision
- Internet of Things (IoT)
- AR/VR

AWS global infrastructure

Region

- A region is considered a geographic location or an area on a map. Each region is located in a **separate geographic area.**

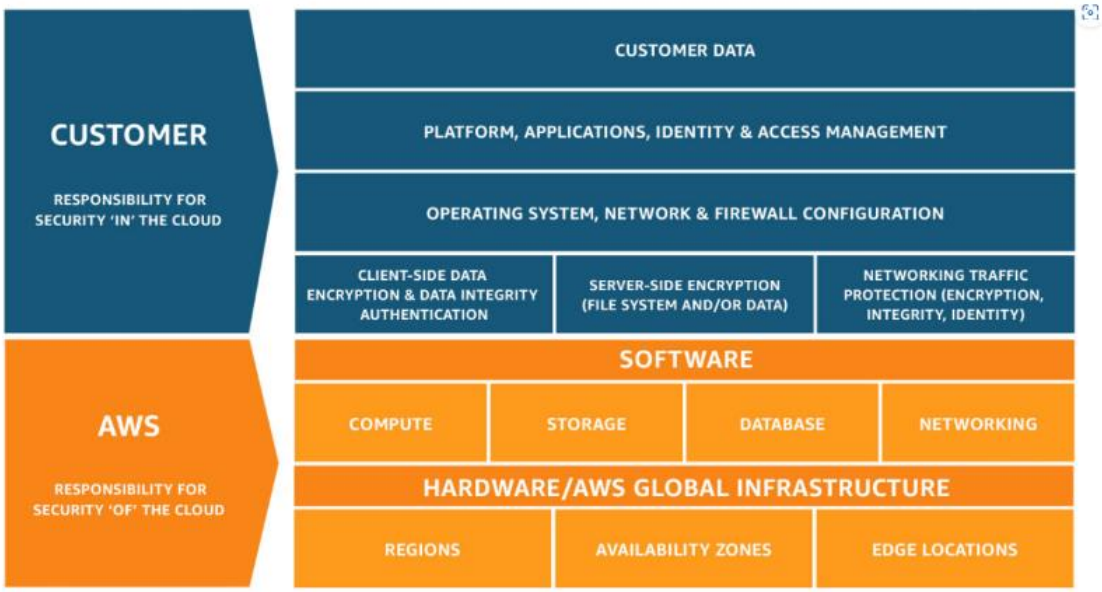
Availability Zone (AZ)

- An availability zone is an isolated location within a geographic region and is a physical **data center** within a specific region. There should be at least two AZs per Region. AZs are distinct locations that are engineered to be isolated from failures

Edge Location

An edge location is as a mini-data center used solely to **cache** large data files closer to a user's location.

Security -= as a shared responsibility model : part by aws part by customer



AWS Responsibility Model

AWS is responsible for:

- Securing edge locations
- Monitoring physical device security
- Providing physical access control to hardware/software
- Database patching
- Discarding physical storage devices

You are responsible for:

- Managing AWS Identity and Access Management (IAM)
- Encrypting data
- Preventing or detecting when an AWS account has been compromised
- Restricting access to AWS services to only those users who need it

--Visual studio code--
VScode can be connected directly to aws via **AWS Toolkit** plugin , setting up **access key ID, secret access key**, and the **session token** via **AWS: Create Credentials Profile.**

Compute Services (Servers in cloud)

EC2 elastic cloud compute – instances in physical servers in AZ ; compute section of aws mgmt console ; The EC2 Dashboard is home to a variety of related services, such as Amazon Machine Images, Elastic Block Store (EBS), Load Balancer, and Auto Scaling ; ec2 security groups – built in firewall for ec2 instances

There are several **pricing** options for EC2.

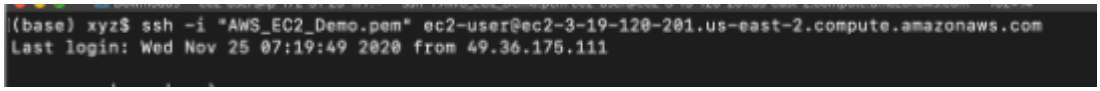
1. On Demand - Pay as you go, no contract.
2. Dedicated Hosts - You have your own dedicated hardware and don't share it with others. For condition where you cannot run that application on a shared server for certain software packages that say no multi-tenancy.
3. Spot - You place a bid on an instance price. If there is extra capacity that falls below your bid, an EC2 instance is provisioned. If the price goes above your bid while the instance is running, the instance is terminated. good if you have an application that has a flexible start and stop time.
4. Reserved Instances - You earn huge discounts if you pay up front and sign a 1-year or 3-year contract. good when you know the steady-state for your applications and you want to pay upfront.

Launch EC2 – **name** and **tags**. **AMI** amazon machine image(template for OS/appln server eg. Linux windows ;; You can build a custom Image by using the EC2 Image Builder), instance **type** (CPU/mem/storage/nw perf/ipv6 availability eg. T2.micro), **key pair** (public -encrypted { stored on ec2 instance }and private { unencrypted .pem file downloaded locally} encryption keys to login to ec2 - name, key type {RSA/ED25519}, o/p - .pem/.ppk [putty private key]), **security group** (for fw rules – allow ssh/http/https ; given security group can be assigned to multiple EC2 instances and you can specify one or more security groups to a VM), **storage**(volumes EBSs --ssd or magnetic storage)

Optional : Advanced Details → User data section, add the following configuration script to run automatically during launch : eg. install configure and launch apache web server { **sudo apt install** apache2 -y }

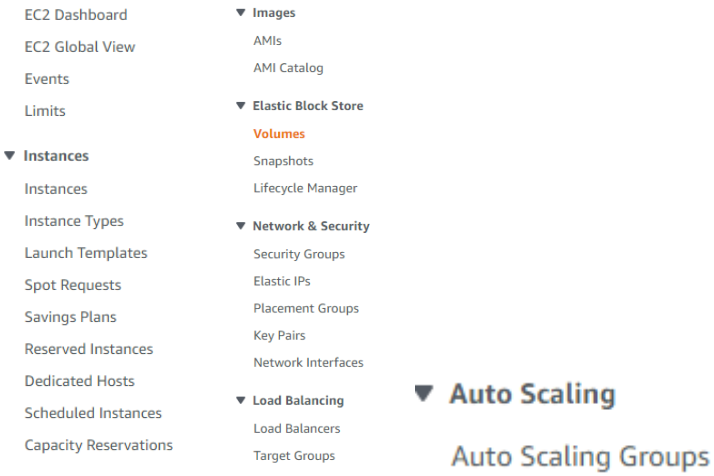
CLI -> aws ec2 create-key-pair --key-name --query –output text > pem
aws ec2 create-security-group –group-name
aws ec2 run-instances --image-id ami-xxxxxxxx --count 1 --instance-type t2.micro --key-name

Connect to Linux EC2 – OPTION 1 - connect button opens **EC2 Instance Connect** wizard opens new browser tab with ec2-user logged in
OPTION 2 – SSH client using private key file(.pem) downloaded from key pair earlier
1. Linux/mac user - chmod 400 <.pem file > ; **ssh** -i <.pem **file**> <public DNS>



2. Windows – from PUTTY – > host = public DNS / IP address ; connection->data Ubuntu username = ubuntu/linux username = ec2-user ; connection → SSH → Auth – upload .ppk file
3. AWS CLI - aws ec2-instance-connect send-ssh-public-key -- instance-id < id goes here> -- availability-zone < zone goes here> -- instance-os-user < user goes here> -- ssh-public-key file://< keyname>.pub

Other EC2 dashboard features -



Launch templates - json/yaml scripts to automate instance launches, simplify permission policies, and enforce best practices .. will be used later in auto scaling
EXAMPLE MISSING!!!

Capacity Reservations - This allows you to reserve the desired capacity (count) of instances in a particular availability zone. The reserved capacity is charged at the selected instance type's On-Demand rate whether an instance is running in it or not.

Elastic Block Store (EBS) – ec2 storage solution ; associated with VM is shown here ; external **physical** hard drive that we attach to the server for additional storage ; persist data after ec2 is terminated ; automatically replicated in AZ

- **Volumes** – block-level storage

CREATE EBS VOL – a) create and attach **while** creating EC2 using Launch instance ; b) empty EBS create and attach to running ec2 ; c) create EBS from prev snapshot and attach to running ec2

Ec2 dashboard->ebs->volumes -> create : type, size (within limit of vol type), AZ(based on region), snapshot ID, Tags , encypt vol checkbox

CLI : aws ec2 create-volume --availability-zone < zone value goes here> --size < value goes here> --tag-specifications 'ResourceType=string,Tags=[{Key=string,Value=string},{Key=string,Value=string}]'

Volume Type	Min (GB)	Max (GB)	I/O per sec	Comments
General Purpose SSD (gp2)	1	16384	[100 - 3000] IOPS	consistent baseline of 3 IOPS/GB.
General Purpose SSD (gp3)	1	16384	[3000-16000] IOPS	
Provisioned IOPS SSD (io1) and (io2)	4	16384	[100-64000] IOPS	best for EBS-optimized instances.
Cold HDD (sc1)	500	16384	Not applicable	predefined throughput (MB/s) with a standard baseline.
Throughput Optimized HDD (st1)	500	16384	Not applicable	predefined throughput (MB/s) with a standard baseline.

Volume Type	Min (GB)	Max (GB)	I/O per sec	Comments
Magnetic (standard)	1	1024	100 IOPS (avg)	

Other options – attach/detach/replace/monitor(states - Okay, Warning, Impaired or insufficient-data ; I/O performance metrics -read/write bandwidth (kB/sec),read/write throughput (Operations/sec),average queue (Operations),Idle time (%),avg read/write size (kB/Operation),avg read/write latency (msec/Operation))/delete
CLI to list vols - aws ec2 describe-volumes -- filters Name=tag:Name,Values=Test* -- query "Volumes[*].{ID:VolumeId,Tag:Tags}"

- **Snapshots** - A snapshot is the saved state of the data in the (existing) volume at a particular moment. Snapshots can be used to transfer volumes from one instance to another or saving the state for future use.
- **Lifecycle Manager** - It helps to schedule and manage the creation and deletion of EBS snapshots.

Elastic IP addresses - static IPv4 address; If ec2 instance fails ,the back-up instance will spin up and have a different IP address, which will require you to update the IP address used in your client application. An Elastic IP address can mask the failure of an instance by **remapping** the current IP address to another instance in your account.

Placement Group - You can imagine the EC2 instances as VMs running on the real servers in a data center. By default, the EC2 instances that you launch will be spread out across underlying hardware. But, sometimes there is a requirement to place the group of interdependent instances to meet the needs of your workload. AWS allows placing the instances based on either of the following **placement strategies** - cluster (tightly packed), partition (logically grouped), or spread evenly across the underlying hardware.

Network Interfaces - represents a virtual network card in a **VPC**, a both private and public IP addresses. you can create and attach additional network interfaces to any instance. An EC2 instance can have multiple network interfaces. ; shows nw here vm is allocated

Load Balancer -distributes the incoming traffic across multiple *targets*, such as EC2 instances in one or more Availability Zones. AWS supports three types of load balancers: **Application Load Balancers** (applications with HTTP and HTTPS traffic , Operating at the request level, provide advanced routing and visibility features targeted at application architectures, including microservices and containers ; allows you to host the different API endpoints of your application on different servers), **Network Load Balancers** (new , balance the load on each individual server , manages volume and latency , TLS offloading at scale, centralized certificate deployment, support for UDP), and **Classic Load Balancers** (might become deprecated soon , previous generation , for existing application running in the EC2-Classic network).

ELASTIC LOAD BALANCER – aws service distributes incoming application traffic across multiple servers ; provides redundancy by adding more servers ; works with EC2 Instances, containers, IP addresses, and Lambda functions
Create Elastic LB – Prerequisite EC2 in running state ; select type of LB -Application LB (configure LB

{scheme -internet facing or internal , n/w – ip add type , vpc, at least two AZ}, security settings & groups, routing{listener eg. http:80} , register targets{target group – instances/ip add(s) / lambda fx}) // N/w LB (Prerequisites – Default VPC with public subnet in each AZ ; create 2 ec2 instances in separate AZs(Configure Security Group step and ensure to have a firewall rule to allow incoming HTTP traffic on port 80.); configure LB(name,scheme-internet facing, vpc,AZs{where you have created instances}), security settings, routing , register targets{new target grp name, type=instance, protovol = tcp ,port = 80}, add the two EC2 instances created previously to the target group) -> o/p=request on NLB goes to instance 1 and instance2 randomly // Classic LB(define LB, assign sec grps, Configure Security Settings,Configure Health Check , Add EC2 Instances)

EC2 Auto Scaling – Elasticity(scale up {vertical – resizing by increasing RAM,CPU,IO} , scale out {horizontal – increased # of resources}) - **service** that automatically launches/terminates EC2 instances based on user-defined scaling policies(conditions), scheduled actions, and health checks. It ensures that you have a specified number of instances always up and running – availability and performance . You can specify the minimum and maximum count of instances. This service uses ***launch templates***, i.e., a script containing the configuration details of the instances that will be launched automatically. ; Integration with SNS – simple notification service (messaging service for alerts) ; other service is AWS auto-scaling (for DynamoDB scaling)
Create EC2 auto-scaling service : Auto Scaling group name, Count of instances(min <= desired <= max) , launch template(auto-scaling group – name, Amazon machine image ID, instance type, key pair, security group,EBS vol, network details-subnet,AZ) , scaling policy(**step/simple/target**), notification setting -> o/p = Activity tab (status-successful/ terminated / in-progress)

VPC

Private network inside aws cloud where services can be launched and public/private subnets can be defined for security measures. Control : IP address ranges, subnets(subset of VPC ; Subnets of a VPC can be present in **different AZs** eg **us-east-1a** and **us-east-1b**), route tables(determine which IP address the network traffic should be directed), nw gateways(communication between resources in VPC and the internet)
VPC spans all the Availability Zones in the **region**
- under Networking & Content Delivery section of AWS console.
- default limit is 5 VPCs per Region; increase can be requested
- no additional charges for creating and using the VPC

Custom VPC –with user-defined subnets, route tables, internet gateways etc.

Each VPC has a valid IPv4/IPv6 CIDR block ; resource in the VPC will have an IP address from the allocated CIDR block minus reserved IPs ;

Create VPC : Resources to create(VPC only/VPC and more) , name, IPV4 cidr block (manual – specify range eg. 10.0.0.0/24 = 256 addresses ; block size bw 16(largest) and 28(shortest)) , IPV6 cidr block, tenancy, tags

CLI : aws ec2 create-vpc --cidr-block 10.0.0.0/16 --tag-specification ResourceType=vpc,Tags=[{Key=Name,Value=MyVpc}]
aws ec2 describe-vpcs

ACL – Access control list ; set of firewall rules for traffic in/out of subnets in VPC : each VPC automatically gets associated with a *modifiable default* network ACL and Each subnet in your VPC must be associated with any *one* network ACL;; a given network ACL can be associated with multiple subnets.

Create ACL, : inbound rules (rule#, type {eg. All traffic} , protocol , port , source , allow/deny)/ outbound rules (-do-)/ subnet association / tags

---ROUTE 53-- - aws cloud dns service ; global,reliable,scalable ; register a domain name (or manage an existing) ; route internet traffic based on the user's geographic location ; dns failover , health check mechanism
The AWS Route 53 DNS service connects user requests to ELB load balancers, Amazon EC2 instances, Amazon S3 buckets, and other infrastructure running on AWS.
he following process occurs when a user accesses a web server via Route 53 DNS:

- A user accesses an address managed by Route 53, www.website.com, which leads to an AWS-hosted machine.
- Typically managed by the local network or ISP, the user's DNS resolver receives the request for www.website.com routed by AWS Route 53 and forwards it to a DNS root server.
- The DNS resolver forwards the TLD name servers for ".com" domains the user requests.
- The resolver acquires the four authoritative Amazon Route 53 name servers that host the domain's DNS zone.
- The DNS resolver selects one of the four AWS Route 53 servers, and requests details for www.website.com.
- The Route 53 name server searches the DNS zone for the www.website.com IP address and other relevant information and returns it to the DNS resolver.
- As specified by the Time to Live (TTL) parameter, the DNS resolver caches the IP address locally, and of course returns it to the user's web browser.
- The browser uses the IP address the resolver provides to contact Amazon-hosted services such as the web server.
- The user's web browser displays the website.

Routing policies types in route 53 – simple , failover , geolocation based,latency based , weighted routing

--COMPUTE POWER –

Serverless ; run code on demand in response to event (eg. S3 file upload or record inserted in DynamoDB); scalability and availability ; pay when code runs

LAMBDA – AWS serverless solution , one specific task(Lambda function) within 15 mins (limit) ; author code locally or via lambda console editor; node.js, python ,ruby,java,go , .net or custom runtime

Create lambda fx: from scratch / blueprint (sample code eg. hello world for node.js)/ browse repo (AWS Serverless Application Repository{container image }) , blueprints - > choose runtime(nodejs12.x) and template name(hello-world), fx name, execution role(new role with basic lambda permissions{eg. upload logs to Amazon CloudWatch Logs}/ existing role/new role from aws policy template), lambda function code, event (trigger code using designer to choose aws services and destination{optional} ; event is json file ; max 10 event per fx)

Fx code :

```
console.log('Loading function');
exports.handler = async (event, context) => {
   //console.log('Received event:', JSON.stringify(event, null, 2));
   console.log('value1 =', event.key1);
   console.log('value2 =', event.key2);
   console.log('value3 =', event.key3);
   return event.key1;  // Echo back the first key value
   // throw new Error('Something went wrong');
};
```

Event json code :

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

Execution result :

```
Test Event Name
mk-lambda-event

Response
"value1"

Function Logs
2023-06-21T08:01:57.337Z    undefined              INFO       Loading function
START RequestId: aa352361-d62d-4127-93ff-e512aeca674e Version: $LATEST
2023-06-21T08:01:57.341Z    aa352361-d62d-4127-93ff-e512aeca674e          INFO       value1 = value1
2023-06-21T08:01:57.342Z    aa352361-d62d-4127-93ff-e512aeca674e          INFO       value2 = value2
2023-06-21T08:01:57.342Z    aa352361-d62d-4127-93ff-e512aeca674e          INFO       value3 = value3
END RequestId: aa352361-d62d-4127-93ff-e512aeca674e
REPORT RequestId: aa352361-d62d-4127-93ff-e512aeca674e          Duration: 3.02 ms   Billed Duration: 4 ms
            Memory Size: 128 MB          Max Memory Used: 57 MB    Init Duration: 176.63 ms

Request ID
aa352361-d62d-4127-93ff-e512aeca674e
```

Eg2. , create s3 bucket(name,region, Copy settings from existing bucket ,allow all public access, versioning , encryption , object lock {objects not deletable/modifiable }), create lambda fx(name,runtime-nodejs12.x , fx code - body: **JSON**.**stringify**('Hello from Lambda!'),), create trigger (service-s3,event type- All object create events), create event(test button,event template - Hello World, event json - "key1": "Place your name here")…and finally adding a file to s3 will trigger this lambda fx

CloudWatch console in the lambda monitoring tab to see metrics, invocation, duration, errors and more.

--ELASTIC BEANSTALK—

orchestration service to deploy **web application** at the touch of a button by spinning up (or provisioning) all of the services that you need to run your application ; automates aws to create ec2, setup auto scaling and LB, runtimes (java,php,python,.net,node.js,ruby,docker and other servers), dbs, vpc, sec grps, CloudWatch alarms ; event logs creates while creating env

Create ElasticBeanstalk env: Env tier (web server or worker{process long running workloads}) , Application name, tags, env name, env domain(<> .us-east-1.elasticbeanstalk.com), platform (.net/go/java/ruby/python etc.), application code (sample or upload{from s3 or local}), presets(single inst,spot, HA, HA with spot etc.) , Optional's(networking, DB, tags, instance traffic, scaling ,root volume ,alerts ,monitoring logging) -→ mk11.us-east-1.elasticbeanstalk.com (instance,sec grp , s3 bucket created)

--STORAGE—

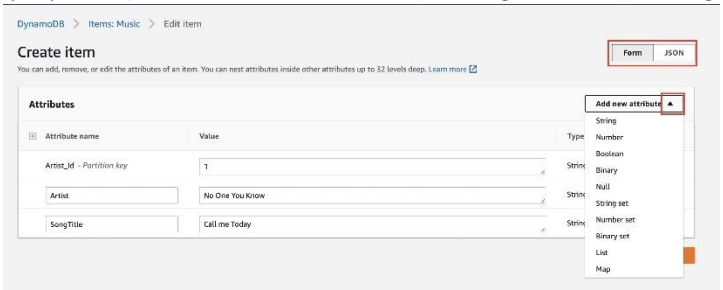
With durability (guarantee of data access), availability(quickly access data) and scalability (meeting demand by adding/removing resources ; vertical {increase server power}/horizontal {adding more servers}/ diagonal)

Storage & Database Services

- Amazon Simple Storage Service **(Amazon S3**) – Object store system (text doc,image,html) ; 1 obj max 5 TB ; gives unique URL inside a bucket within a region across multiple AZ ; s3 acceleration -fast, easy, and secure transfers of files over long distances between your data source and your S3 bucket ; use case – static website, content delivery , backup and recovery , archiving , appln data ; S3 storage classes (standard/glacier/glacier deep / intelligent-tiering / standard infrequent /one zone infrequent)
  Creation of S3 bucket above(near lambda fx)

Sometimes the local machine's network setting or firewall might block, or the browser's Adblocker may prohibit the file upload, such as *buysellads.svg* file. Workarounds -1. Chrome->settings->Security and Privacy >> Site Settings.-> Additional content settings >> Ads 2. Try with CLI

- Amazon Simple Storage Service (Amazon S3) **Glacier** – one of the s3 storage classes for archiving; not so fast accessing speed ; use case – audit log files
- **DynamoDB** - noSQL DB ; quickly scale to handle large amt of data and requests ; schema less (no fixed structure); data stored in json key value pairs ; each row is called document – document data model ; no servers to maintained in aws - full managed serverless ; availability in 3 AZs
  Create DynamoDB table : name, partition key (primary key), sort key (second part of primary key to filter item which use same partition key), default settings (r/w capacity, auto scaling, encryption key), PITR (point in time recovery – continuous backups)
  Actions : edit capacity , create item (item ~ row in rdbms ; item can contain multiple attributes; attributes have name, datatype and value), create index (improve querying performance) , create replica, export to s3 , scan or query items (attribute name, condn , value), set as global table (multi region)



- Relational Database Service (**RDS**) – service for **DB administration** ; to automate mgmt (patching, backup, install, monitoring, performance and security) ; DB engines supported(Oracle , PostgreSQL,MySQL,MariaDB and SQL server)

Create MySQL dB : RDS dashboard -> standard (custom configuration) / easy create(from best practices) , config (amazon aurora/mysql/MariaDB/oracle ..) , DB instance size (PROD-500gb/Test-100gb/Free tier-20gb), db name, master username password
Connect to DB instance using a MYSQL client – eg. Mysql workbench – using RDS DB instance endpoint URL(database-1.us-east-1.rds.amazonaws.com) and port#(3306) and master credentials ; ensure attached sec grp has allowed inbound traffic from anywhere.

- **Redshift -** cloud data warehousing service ; to manage big data and run fast queries using SQL,ETL and BI tools ; stores data in column format ; uses ML ; Redshift Spectrum – run queries in s3 ; supported encryption
- ElastiCache
- Neptune
- Amazon DocumentDB

--CDN-- -content delivery network ; speeds delivery of static and dynamic web content eg. web pages css js images etc ; pulled from cached (from closest edge location {mini data center})to reduce latency and decrease server load
Two types – push cdn (content is received by the CDNs whenever changes occur on the server, responsibility lies in us for uploading the content to CDN) and pull cdn (new content is grabbed from the server when the first user requests the content from the site, lower requests for the first time till the content gets stored/cached on the CDN.)
CloudFront – aws cdn service ; global ; integration with s3 , ES lb , ec2, lambda ; supports GeoIP blocking (specific countries restrictions) ; cache control headers determines how often CloudFront checks source for updated version ; max size of file supported for delivery is 20 GB
Create CloudFront distribution and integration with S3 (non public): create distribution - origins and groups (origin type s3 bucket name) , Origin Access Identity, restrictions , invalidations (to explicitly expire content from cache) ; o/p – so for a sample.html deployed in a private s3 bucket will not be available via object url https://mk-cloudfront-s3-bucket.s3.amazonaws.com/sample.html (as public access restricted) but will be available via Distribution domain name d14muuwiwsgykz.cloudfront.net/Sample.html after cloudFront distribution is deployed and sample.html is cached ~30 min
AWS cloudfront not updating on update of files in S3? Create Invalidation -> In the "object path" text field, you can list the specific files ie /index.html or just use the wildcard /* to invalidate all. This forces cloudfront to get the latest from everything in your S3 bucket.

--SECURITY—

What? Secure data, applications , servers
AWS WAF – Web Application firewall – nw security mech to monitor incoming and outgoing traffic coming to an Amazon API Gateway API, an Amazon CloudFront distribution, or an Application Load Balancer, AWS AppSync using rules ; can stop sql inj, cross site scripting ; ; can also protect websites not hosted in AWS through Cloud Front ;;create *Web access control lists (web ACLs)* to monitor HTTP(S) requests ;
AWS SHIELD - provides continuous DDoS attack detection and automatic mitigations – standard(always running automatically , out of box free service) and advanced(paid, view the threat-event summary such as total events, largest bit rate, largest packet rate, and the largest request rate) configurations.
AWS Firewall Manager: configure and manage firewall rules across accounts and applications centrally.

--**IAM**-- -> least privilege access concept ; global service across all regions ; mfa – multi factor authentication to secure root account ;
IAM user - unique identifier generated by IAM service to grant access to AWS resources ; can be a person/ system / appln ; scope of user access controlled by roles and policies ; ; user consists of name , access credentials – console pswd or access key id
IAM group – collection of iam users ; attach/ detach permissions to a group using access control policies
IAM role – set of policies ;assign role to iam user or service like ec2 ; delegate access with defined permissions without sharing long-term access keys
IAM policy – granular level permissions ;json file defining resource to get access , level of access and allowed actions ; attach policy to user/groups/roles ;predefined as well as user defined; sample IAM policy that allows full EC2 access within a specific AWS region:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Action": "ec2:*",
```

```
            "Resource": "*",              // any resource
            "Effect": "Allow",
            "Condition": {
               "StringEquals": {
                  "ec2:Region": "us-east-2"
               }
            }
         }
      ]
   }
```

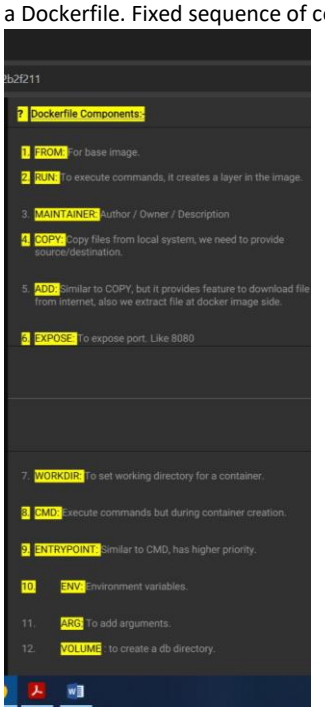
IAM policy simulator - evaluate, and validate the effects of your access control policies ; ; AWS offers other types of policies, such as an S3 Bucket Policy, an SNS Topic Policy, a VPC Endpoint Policy, and an SQS Queue Policy.
Create Policy : Select service (s3/lambda/ec2), Actions allowed(Access level – list/read/write/permission mgmt/tagging) , Resource ARNs, policy name and description -> o/p Json policy
Attach policy to a IAM user : create user : username , programmatic(through access keys) / AWS Management Console access , permissions (Add user to group / copy permissions from a user /attach policy directly)

--MESSAGING--
notification when certain events happen in cloud ; from one system to another ;
SNS – simple notification service – aws service for sending notifications to the users of your applications ; decouple notification login ; uses publish{ to Amazon SQS queues, AWS Lambda functions, and HTTP/S webhooks }/ subscribe{user/applcn} model ; notification via mobile push / email/ text msg ; SNS Topic names are limited to 256 characters.
Create SNS TOPIC : Type - FIFO{ Strictly-preserved message ordering; protocols: SQS }/standard { protocols: SQS, Lambda, HTTP, SMS, email, mobile application endpoints } , name , access policy – basic{ Define who can publish messages to the topic }/ advanced ,
Create SNS subscription : topic , protocol {email/lambda/http/sms} , endpoint {eg. email address} – status goes to pending confirmation -> confirmed
Now when you publish a new message { Subject , Body } to our topic, it gets distributed to subscriptions.

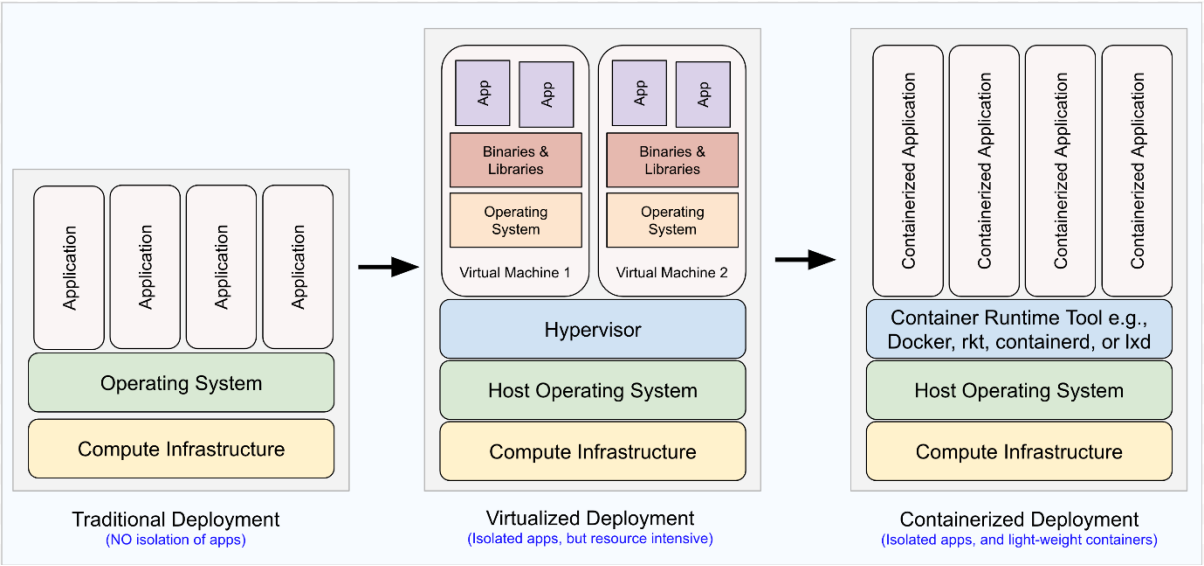
--MESSAGE QUEUEING—
For performance , scalability , async processing and improved user experience ;
SQS – Simple queueing service ; aws managed queue service ; to integrate Q functionality to your application ; send/store/receive msgs ; types – FIFO queue (where ordering is important)/ standard queue { best-effort ordering but no guarantees ; occasionally more than one copy of a message is delivered } ; FIFO queues support up to 3,000 messages per second with batching or up to 300 messages per second without batching whereas standard queue supports an unlimited number of transactions per second (TPS) for each API action (SendMessage, ReceiveMessage, or DeleteMessage).
CREATE QUEUE: type – standard / fifo , name , Visibility timeout 0 seconds - 12 hours , Message retention period - The duration (1 minute - 14 days) , Delivery delay - (0 seconds - 15 minutes) , Maximum message size - between 1 KB and 256 K , access policy – sender / receiver privileges -> o/p = https://sqs.us-east-2.amazonaws.com/014421265158/MyQueue to be used in application url
SEND message – (to Q) : body, delay ,metadata{metadata,geopspatial data,signature, identifier}
RECEIVE message – (from Q) : a consumer *polls* to receive up to 10 number of messages from the queue

--CONTAINERS—
Docker – container technology/runtime tool;; : everything application needs to run ; application code itself + dependencies +libraries + config files +runtimes etc. ; no need to rebuild when changing environment; easily move container across which can run on its own ; use case- microservice arch application working on discrete components
Containers share a single kernel and share application libraries.
Containers cause a lower system overhead as compared to Virtual Machine ; smaller ; run as isolated processes versus virtualized hardware ; faster to boot
Docker Image - portable auto-generated template that contains a set of instructions to create a container. An image can be instantiated multiple numbers of times to create multiple *containers*.

Docker File - text file containing commands to create an image ; Docker generates images by reading the commands from a Dockerfile. Fixed sequence of commands -



Kubernetes – container orchestration service to manage Docker cluster . other is docker swarm.



ECS - ELASTIC CONAINER SERVICE --
AWS container orchestration service to manage docker containers ; launching / stopping containers ; scaling your containerised application ; schedule long-running applications, services, and batch processes using ECS.
ECS also supports other AWS services like Elastic Load Balancing, EC2 security groups, EBS volumes, and IAM roles.
Task definition : application requirements concerning containers, such as the max amount of total CPU and memory used for the task ; ECS offers to create a task definition using either the AWS Fargate { priced on task size } or AWS EC2 priced on computing resource usage ,; Add a container definition – name, image, memory limit
ECS CLUSTER : set of containers running task requests within an AWS region.
create cluster : name, type – ec2Linux+networking / networking only , configure vpc , enable cloudWatch container insights
Container agent : utility that connects container instances to one of your clusters. Each container instance runs a container agent.
container instance is an EC2 instance that is registered into any of your ECS clusters.

---AWS MANAGEMENT—
CLOUD TRAIL - service to log api calls
AWS account audit mechanism ; checks for log in, service accessed, actions performed ; records all AWS API calls (stored in S3 or CloudWatch logs), delivers log file in response ; covers console as well as SDK (and CLI) ; alerts and alarms can be setup ; last 90 days results ; create up to five trails in an AWS region ; changed for usage of s3 bucket
CREATE TRAIL: name, logs storage location (create a new s3 or use existing one), encryption (CloudTrai will encrypt log files by using the AWS Key Management Service) , *Events and Management details (choose type of event to be logged {Management events, Data events (operations performed within a resource), and Insights events (unusual activity, errors, or user behavior)})*

CLOUD WATCH – monitor aws resources ,disk , network activity , databses and volumes
AWS service to monitor resources and applications by collecting logs, metrics and events ; Set alarms and create triggers (eg. for lambda) ; eg. create a Cloud Watch event to notify via an SNS topic when an EC2 instance is created.
CREATE CLOUDWATCH rule : Event -> Rule , rule name, rule type(Rule with event pattern{runs when an event matches the defined event pattern} or schedule) , event source (AWS events type : service : eg. EC2 Instance State-change Notification) , Specific state(s) – running , target – sns topic
Testing : create an ec2 instance and when its state changes to running, you will get an sns notification to email

AWS Config – to find out about configurations and changes in infra

_____INFRASTRUCTURE AS CODE_____
Script to stand up servers , databases , runtime parameters and resources ; manage a collection in a logical unit ; eg, configure vpc security group , launch EC2 instances, create LB , create auto scaling

--CLOUDFORMATION---
AWS IaC service ; uses text file template in jsin/yaml using scratch or blueprint ; can also be created using visual designer (drag and drop interface); version controlled
CREATE CLOUDFORMATION STACK (from designer): select designer tab , drag drop s3-> bucket , change properties json text as needed

```
{
 "AWSTemplateFormatVersion": "2010-09-09",
 "Description": "Basic S3 Bucket CloudFormation template",
 "Resources": {
  "S3BucketCreatedByCloudFormation": {
   "Type": "AWS::S3::Bucket",
   "DeletionPolicy": "Delete",
   "Properties": {
    "AccessControl": "PublicRead"
   }
  }
 },
 "Outputs": {
  "BucketName": {
   "Value": {
    "Ref": "S3BucketCreatedByCloudFormation"
   },
   "Description": "Name of the newly created Amazon S3 Bucket"
  }
 }
}
```

Refresh designer , save to local as json file , validate template , click on create stack to deploy , name , parameters -> o/p once stack reaches CREATE_COMPLETE state, a new s3 bucket will be created

---AWS CLI---
Install AWS CLI v2 in local ; uses python internally ; create iam user from policy {Programmatic access with administrator privileges - using attach existing policies directly option with “AdministratorAccess” policy }and generate access key pair - ; aws configure command with access key id and secret access key {generate an Access key from the AWS IAM service, and specify the level of permissions (authorization) with the help of *IAM Role* } , set aws session token to none , set profile {aws configure --profile new_name} and region ; default o/p format – yaml or json
Create s3 : aws s3api create-bucket --bucket my-033212455158-bucket --acl public-read-write --region us-east-1 --profile UdacityLab
Upload file : aws s3api put-object --bucket my-033212455158-bucket --key Sample.html --body Sample.html --profile UdacityLab
Delete s3 and contents : aws s3api delete-object --bucket my-033212455158-bucket --key Sample.html
aws s3api delete-bucket --bucket my-033212455158-bucket --profile UdacityLab

____STATIC WEBSITE___
Just html, js,css w/o server side processing ; data stored on s3 public bucket and cached using CloudFront CDN ; secure using IAM ;
steps: create s3 , uncheck the “Block all public access” to allow enable the public access to the bucket objects via the S3 object URL (**Hosting requires the content should be publicly readable.**), upload html ,css,js to bucket , add bucket IAM policy -

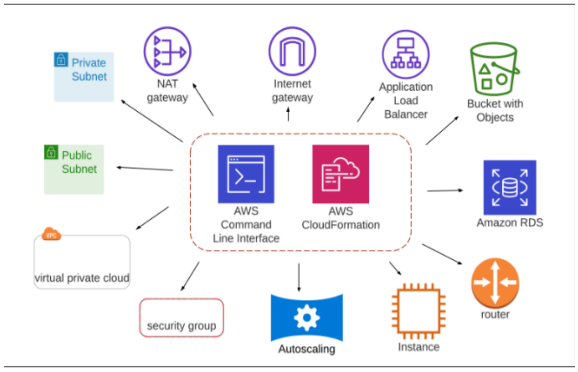
```
{
 "Version":"2012-10-17",
 "Statement":[
  {
   "Sid":"AddPerm",
   "Effect":"Allow",
   "Principal": "*",
   "Action":["s3:GetObject"],
   "Resource":["arn:aws:s3:::your-website/*"]
  }
 ]
}
```

-> O/p = bucket dns url/index.html publically accessible = https://mk-proj1-bucket.s3.amazonaws.com/index.html

If we were not learning about static website hosting, we could have made the bucket private and wouldn't have to specify any bucket access policy explicitly. In such a case, once we set up the **CloudFront distribution**, it will automatically update the current bucket access policy to access the bucket content. The CloudFront service will make this happen by using the **Origin Access Identity** user.
Configure **Static website hosting in s3->o/p=**: http://mk-proj1-bucket.s3-website-us-east-1.amazonaws.com/ , create cloudfront distribution with static website website hosting endpoint -> o/p=CloudFront Distribution domain name: https://d24ygk3svamcxm.cloudfront.net/

-------LESSON 2-----

IaC is capable of



Benefits of cloud DevOps –resolves issues of unpredictable deployments (devops tool manages unpredictable and automated deployment) , mismatch environments , configuration drift(devops tools like puppet , chef and ansible to deploy configurations as script files).
Important Practices are – CICD {Continuous integration is the process flow of testing any change made to your development flow, while continuous deployment tracks those changes through to staging and production systems. Tools - Jenkins / teamCity / Circle CI}, IaC {yaml/json}, microservices , monitoring and logging, Communication and Collaboration

--CLOUD FORMATION—

AWS tool for creating, managing, configuring, and deploying cloud resources ; to provision a set of cloud resources multiple times, at scale. IaC code under version control using Git ; declarative language {like SQL, specifying what you want w/o specifying how you want } ; handles resource dependencies {which resource to start up before another }
Tools – 1. AWS CLI tool {download from exe}, git , code editor like vs code
2. Aws access type - **Programmatic access {** code to interact with AWS **;** this option will Enable the access key ID and secret access key for the AWS CLI – we will use this for CLI**} / AWS management console access** **-** here no region to be specified as users are global
3. This is combined with administrator privileges using attach existing policies directly option with "AdministratorAccess" policy.
4. Generate access key pair (max 2 active keys per user ; best practice to rotate keys every 90 days ; actions to delete or deactivate keys available)

Access key	**Secret access key**
AKIATU64N7XPCTWE5BTX	izHg29jKkgVkiFKY4aUMMOUWSvFoJdhlmfhywjAw

5. aws configure command

F:\>aws configure
AWS Access Key ID [None]: AKIATU64N7XPCTWE5BTX
AWS Secret Access Key [None]: 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
Default region name [None]: us-west-2
Default output format [None]:

6. Verify access – aws s3 ls or aws iam list-users command

YAML is the industry preferred version that's used for AWS and other cloud providers (Azure, Google Cloud Platform ; whitespace indentation matters! recommend that you use **four white spaces for each indentation** in YAML ; cloudformation files can be grouped based on resource type{separate files to organize your code}; don't need to memorize the code that you need for each resource - **AWS CloudFormation - Resource and property types** ; syntax -

Template - describe your AWS resources and their properties ; such as .json, .yaml, .template, or .txt ; For example, in a template, you can describe an Amazon EC2 instance, such as the instance type, the AMI ID, block device mappings, and its Amazon EC2 key pair name. Whenever you create a stack, you also specify a template that CloudFormation uses to create whatever you described in the template ;You can also specify multiple resources in a single template and configure these resources to work together. ; you can also add dynamic input ; At least 1 resource in cloud formation template
For example, if you created a stack with the following template, CloudFormation provisions an instance with an ami-0ff8a91507f77f867 AMI ID, t2.micro instance type, testkey key pair name, and an Amazon EBS volume.

AWSTemplateFormatVersion: 2010-09-09 //optional
Description: A sample template //optional
Resources: //at least 1 -VPC/EC2/DB
MyEC2Instance: // **Name**: A name you want to give to the resource (does this have to be unique across all resource types?)
Type: 'AWS::EC2::Instance' // **Type**: Specifies the actual hardware resource that you're deploying
Properties: // **Properties**: Specifies configuration options for your resource
ImageId: ami-0ff8a91507f77f867
InstanceType: t2.micro
KeyName: testkey
BlockDeviceMappings:

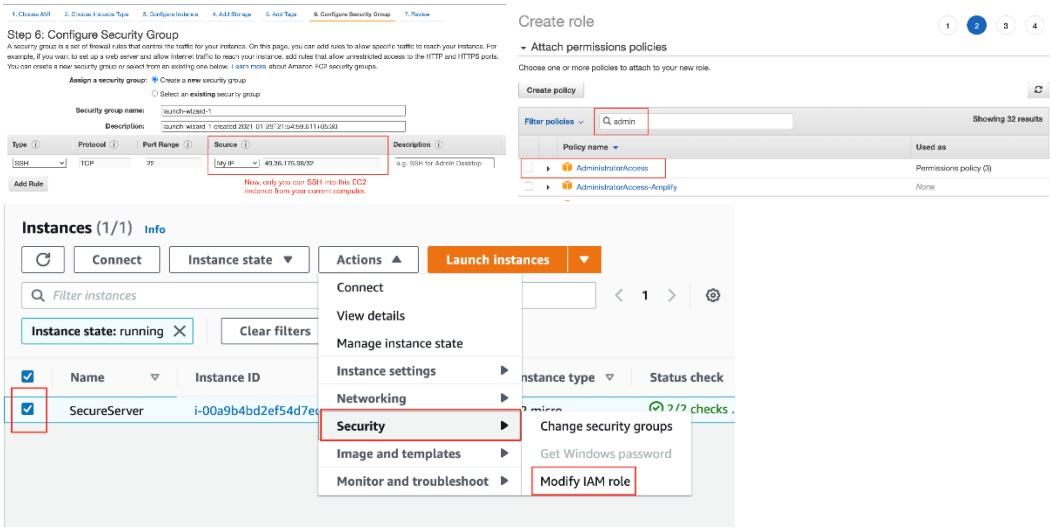
- DeviceName: /dev/sdm
Ebs:
VolumeType: io1
Iops: 200
DeleteOnTermination: false
VolumeSize: 20

Stack: Instance spawned from template ; A stack is a group of resources as a single unitack. These are the resources that you want to deploy, and that are specified in the YAML file ; . All the resources in a stack are defined by the stack's CloudFormation template.
Change sets - make changes to the running resources in a stack, you can generate a change set

* **Run the aws command** :
aws cloudformation create-stack --stack-name myFirstTest --region us-east-1 --template-body file://testcfn.yml
* **Alternate method - Shell Script** Helper scripts ./create.sh stack_name template.yml parameters.json
create.sh = aws cloudformation create-stack --stack-name $1 --template-body file://$2 --parameters file://$3 --capabilities "CAPABILITY_IAM" "CAPABILITY_NAMED_IAM" --region=us-west-2 ``` were `$1`, `$2`, and `$3` can be replaced with the actual values passed as command-line arguments.
* **Alternate method - Batch Script**
actual values can be written as `%1` instead as`$1`.

update-stack used if create-stack used first and then added more resources in yml script
aws cloudformation update-stack --stack-name myFirstTest --region us-east-1 --template-body file://testcfn.yml
aws cloudformation **describe**-stacks *--stack-name myFirstTest*
https://docs.aws.amazon.com/cli/latest/reference/cloudformation/index.html#cli-aws-cloudformation

Other exercise - create an EC2 Instance based on Amazon Linux AMI that you can connect via SSH(limit access to your IP address only, using Security Groups). **The instance will already have the CLI installed by default**. Once the instance is running, create an IAM role with admin access and attach to ec2 ; Connect to your EC2 instance via SSH (from ssh client) = ssh -i "<keypairFile>.pem" ec2-user@<ec2instance> ; aws –version on instance will give CLI version



Other other exercise :D

1. Create a Security Group which only allows inbound access on TCP port 80 and also allows unrestricted outbound access,
2. Create an EC2 instance in a given VPC, 3 public subnet, 4using above sec group, 5 t3.micro, 6 **Amazon Linux 2 AMI (HVM), 7 SSD Volume Type**, 8 on top of which we'll install our web server software,9 your server should get assigned a public IP address that you can reach over the internet to verify it's working correctly, 10 bonus of you use parameters for the Subnet, VPC and AMI

Answer - parameters.json ::=>
[

```
  {
    "ParameterKey": "myVPC",
    "ParameterValue": "vpc-cb298db6"
  },
  {
    "ParameterKey": "PublicSubnet",
    "ParameterValue": "subnet-3425b06b"
  },
  {
    "ParameterKey": "AMItoUse",
    "ParameterValue": "ami-047a51fa27710816e"
  }
]
```

Solution.yml::=>

```
Parameters:    //pass values to yaml script from json, to be declared above any resources ; reusability
  myVPC:
     Description: VPC used to deploy our resources below
     Type: AWS::EC2::VPC::Id
  PublicSubnet:
     Description: Subnet to be used for our Web Server
     Type: AWS::EC2::Subnet::Id
  AMItoUse:
     Description: AMI to use for our base image
     Type: String
     Default: ami-047a51f816e    //Optional - Default values can be used in parameters ; shown for example , not req here
      AllowedValues:  <>    //Optional - Default values can be used in parameters ; shown for example , not req here
Resources:     // mandatory section – servers/gateways/vpn connectors
  myWebAccessSecurityGroup: //Resource name
    Type: AWS::EC2::SecurityGroup  //Resource type
    Properties:  //Resource properties
       GroupDescription: Allow http to our test host
       VpcId:
         Ref: myVPC
       SecurityGroupIngress:
       - IpProtocol: tcp
         FromPort: 80
         ToPort: 80
```

```
      CidrIp: 0.0.0.0/0
    SecurityGroupEgress:
    - IpProtocol: -1
      FromPort: -1
      ToPort: -1
      CidrIp: 0.0.0.0/0
 myWebServerInstance:
  Type: AWS::EC2::Instance
  Properties:
    InstanceType: t3.micro
    ImageId: !Ref AMItoUse    //!Ref for referencing parameters
    NetworkInterfaces:
     - AssociatePublicIpAddress: "true"
       DeviceIndex: "0"
       GroupSet:
        - Ref: "myWebAccessSecurityGroup"
       SubnetId:
        Ref: "PublicSubnet"
    UserData:
     Fn::Base64: !Sub |
      #!/bin/bash
      yum update -y
      yum install -y httpd
      systemctl start httpd
```

How to use Outputs section in yml script – Not related to above exercise ; to sed in another yml file as parameter

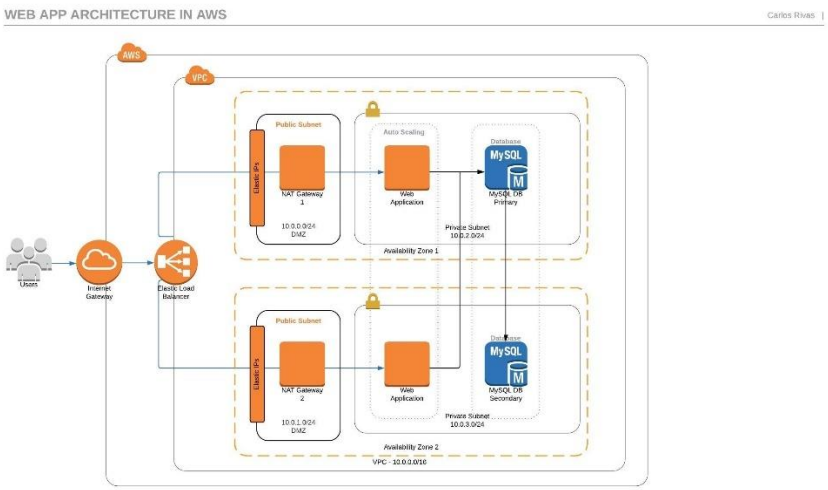
Outputs //declares output values for each resource that you can import into other stacks.

```
Outputs:

    VPC:
        Description: A reference to the created VPC
        Value: !Ref VPC
        Export:
          Name: !Sub ${EnvironmentName}-VPCID
```

Description (optional) - A string
Value (required) - The property returned by the aws cloudformation describe-stacks command.
Export (optional) - The name of the resource output to be exported for a cross-stack reference.

Other other other exercise – network scripts – keep diagram open parallel and read section by section



Tip: Don't attempt to write it all at once! just write VPC and Internet Gateway and test it(create-stack), then increment adding more resources(update-stack)
Parameters.json →>

```
[
        {
                "ParameterKey": "EnvironmentName",
                "ParameterValue": "UdacityProject"
        },
        {
                "ParameterKey": "VpcCIDR",
                "ParameterValue": "10.0.0.0/16"
        } ,
        {
                "ParameterKey": "PublicSubnet1CIDR",
                "ParameterValue": "10.0.0.0/24"
        },
        {
                "ParameterKey": "PublicSubnet2CIDR",
                "ParameterValue": "10.0.1.0/24"
        },
        {
                "ParameterKey": "PrivateSubnet1CIDR",
                "ParameterValue": "10.0.2.0/24"
        },
        {
```

```
                "ParameterKey": "PrivateSubnet2CIDR",
                "ParameterValue": "10.0.3.0/24"
        }
]
```

Solution.yml➔>

Parameters:

  EnvironmentName:
    Description: An environment name that will be prefixed to resource names
    Type: String

  VpcCIDR:
    Description: Please enter the IP range (CIDR notation) for this VPC
    Type: String
    Default: 10.0.0.0/16

  PublicSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in the first Availability Zone
    Type: String
    Default: 10.0.0.0/24

  PublicSubnet2CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in the second Availability Zone
    Type: String
    Default: 10.0.1.0/24

  PrivateSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the private subnet in the first Availability Zone
    Type: String
    Default: 10.0.2.0/24

  PrivateSubnet2CIDR:
    Description: Please enter the IP range (CIDR notation) for the private subnet in the second Availability Zone
    Type: String
    Default: 10.0.3.0/24

```
Resources:

  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: !Ref VpcCIDR
      EnableDnsHostnames: true
      Tags:
        - Key: Name
          Value: !Ref EnvironmentName

  InternetGateway:
    Type: AWS::EC2::InternetGateway
    Properties:
      Tags:
        - Key: Name
          Value: !Ref EnvironmentName

  InternetGatewayAttachment:    //additional resource for connecting VPC and internet gateway
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      InternetGatewayId: !Ref InternetGateway
      VpcId: !Ref VPC
PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 0, !GetAZs '' ]      //GetAZs outputs a list of AZ inside VPC and 0 is to select first value from
that list
      CidrBlock: !Ref PublicSubnet1CIDR
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Subnet (AZ1)
```

```
  PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 1, !GetAZs '' ] //]     //GetAZs outputs a list of AZ inside VPC and 1 is to select second value from that list
      CidrBlock: !Ref PublicSubnet2CIDR
      MapPublicIpOnLaunch: true   // enabling auto assign public ip address
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Subnet (AZ2)

  PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 0, !GetAZs '' ]
      CidrBlock: !Ref PrivateSubnet1CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Subnet (AZ1)

  PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 1, !GetAZs '' ]
      CidrBlock: !Ref PrivateSubnet2CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Subnet (AZ2)
NatGateway1EIP:   // NAT gateway elastic IP for static IP allocation if resources restart at any point instead of disposable IP
```

```
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment  //Confirmation clause; this part wont execute unless
InternetGatewayAttachment  id built first; decides ordering
    Properties:
      Domain: vpc

  NatGateway2EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
      Domain: vpc

  NatGateway1:    //NAT Gateway
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId   //Get Attribute
      SubnetId: !Ref PublicSubnet1

  NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
      AllocationId: !GetAtt NatGateway2EIP.AllocationId
      SubnetId: !Ref PublicSubnet2

  PublicRouteTable: //Route table
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Routes

  DefaultPublicRoute: //allow all outbound trafiic routed to Internet Gateway
    Type: AWS::EC2::Route    // Rules(in order of precedence: very specific to least specific) – single rule
    DependsOn: InternetGatewayAttachment
```

```
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway

PublicSubnet1RouteTableAssociation://Attach route table to subnet
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: !Ref PublicRouteTable
    SubnetId: !Ref PublicSubnet1

PublicSubnet2RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
  Properties:
    RouteTableId: !Ref PublicRouteTable
    SubnetId: !Ref PublicSubnet2


PrivateRouteTable1:
  Type: AWS::EC2::RouteTable
  Properties:
    VpcId: !Ref VPC
    Tags:
      - Key: Name
        Value: !Sub ${EnvironmentName} Private Routes (AZ1)

DefaultPrivateRoute1:
  Type: AWS::EC2::Route
  Properties:
    RouteTableId: !Ref PrivateRouteTable1
    DestinationCidrBlock: 0.0.0.0/0  // default rule to route all outbound traffic to nat gateway
    NatGatewayId: !Ref NatGateway1

PrivateSubnet1RouteTableAssociation:
  Type: AWS::EC2::SubnetRouteTableAssociation
```

```
      Properties:
        RouteTableId: !Ref PrivateRouteTable1
        SubnetId: !Ref PrivateSubnet1

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ2)

  DefaultPrivateRoute2:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway2

  PrivateSubnet2RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable2
      SubnetId: !Ref PrivateSubnet2
Outputs:

  VPC:
    Description: A reference to the created VPC
    Value: !Ref VPC
    Export:
      Name: !Sub ${EnvironmentName}-VPCID   // substitution  - UdacityProjectDemo-VPCID as global variable having value
vpc-0aafb849bd17f11d3

  VPCPublicRouteTable:
    Description: Public Routing
```

```
    Value: !Ref PublicRouteTable
    Export:
      Name: !Sub ${EnvironmentName}-PUB-RT

  VPCPrivateRouteTable1:
    Description: Private Routing AZ1
    Value: !Ref PrivateRouteTable1
    Export:
      Name: !Sub ${EnvironmentName}-PRI1-RT

  VPCPrivateRouteTable2:
    Description: Private Routing AZ2
    Value: !Ref PrivateRouteTable2
    Export:
      Name: !Sub ${EnvironmentName}-PRI2-RT

  PublicSubnets:
    Description: A list of the public subnets
    Value: !Join [ ",", [ !Ref PublicSubnet1, !Ref PublicSubnet2 ]]    // Join function to make list comma separated
    Export:
      Name: !Sub ${EnvironmentName}-PUB-NETS

  PrivateSubnets:
    Description: A list of the private subnets
    Value: !Join [ ",", [ !Ref PrivateSubnet1, !Ref PrivateSubnet2 ]]
    Export:
      Name: !Sub ${EnvironmentName}-PRIV-NETS

  PublicSubnet1:
    Description: A reference to the public subnet in the 1st Availability Zone
    Value: !Ref PublicSubnet1
    Export:
      Name: !Sub ${EnvironmentName}-PUB1-SN

  PublicSubnet2:
```

Description: A reference to the public subnet in the 2nd Availability Zone

Value: !Ref PublicSubnet2

Export:

Name: !Sub ${EnvironmentName}-PUB2-SN

PrivateSubnet1:

Description: A reference to the private subnet in the 1st Availability Zone

Value: !Ref PrivateSubnet1

Export:

Name: !Sub ${EnvironmentName}-PRI1-SN

PrivateSubnet2:

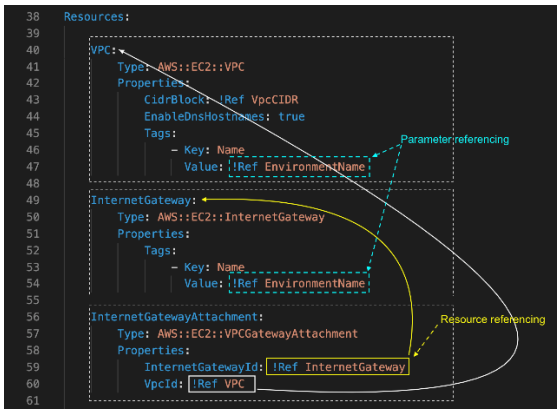
Description: A reference to the private subnet in the 2nd Availability Zone

Value: !Ref PrivateSubnet2

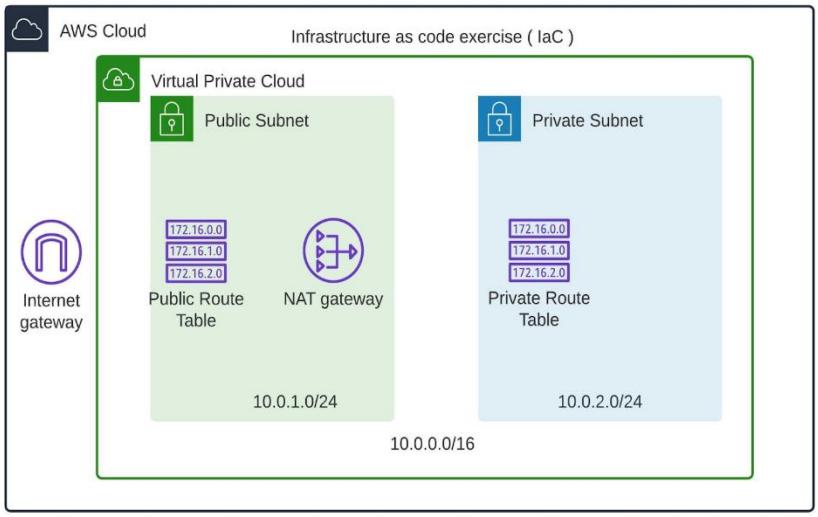
Export:

Name: !Sub ${EnvironmentName}-PRI2-SN

Referencing:



Other other other other exercise



VPC (cidr defined as input parameter) with 1 public (with map public iponlaunch - true)and 1 private subnet(with map pub lic iponlaunch - false) with name tags (and cidr defined as input param and /24 in size)

1 internet gateway attach to vpc

1 nat gateway inside public subnet - with allocated elastic ip

1 public routing table assigned to public subnet

1 private routing table assigned to private subnet

public routing table route to send default traffic to internet gateway

private routing table route to send default traffic to nat gateway

maintain ordering of component creation using depends on attribute

nat and internet gateway attachment

Parameter.json

```
[
  {
    "ParameterKey": "EnvironmentName",
    "ParameterValue": "UdacityCourse2Lesson3Challenge2"
  },
  {
    "ParameterKey": "VpcCIDR",
    "ParameterValue": "10.0.0.0/16"
  },
  {
    "ParameterKey": "PublicSubnet1CIDR",
    "ParameterValue": "10.0.1.0/24"
```

```
  },
  {
    "ParameterKey": "PrivateSubnet1CIDR",
    "ParameterValue": "10.0.2.0/24"
  }
]
```

Solution. Yml➔>

```
Description: >
  MK / Udacity

Parameters:

  EnvironmentName:
    Description: An environment name that will be prefixed to resource names
    Type: String

  VpcCIDR:
    Description: Please enter the IP range (CIDR notation) for this VPC
    Type: String
    Default: 10.0.0.0/16

  PublicSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the public subnet in the first Availability Zone
    Type: String
    Default: 10.0.1.0/24

  PrivateSubnet1CIDR:
    Description: Please enter the IP range (CIDR notation) for the private subnet in the first Availability Zone
    Type: String
    Default: 10.0.2.0/24

Resources:

  VPC:
    Type: AWS::EC2::VPC
```

```
  Properties:
    CidrBlock: !Ref VpcCIDR
#     EnableDnsHostnames: true  wasn't asked here
    Tags:
      - Key: Name
        Value: !Ref EnvironmentName

InternetGateway:
  Type: AWS::EC2::InternetGateway
  Properties:
    Tags:
      - Key: Name
        Value: !Ref EnvironmentName

InternetGatewayAttachment:
  Type: AWS::EC2::VPCGatewayAttachment
  Properties:
    InternetGatewayId: !Ref InternetGateway
    VpcId: !Ref VPC

PublicSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      AvailabilityZone: !Select [ 0, !GetAZs '' ] #could also have been hardcoded AvailabilityZone: 'us-east-1a'
      CidrBlock: !Ref PublicSubnet1CIDR
      MapPublicIpOnLaunch: true
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Subnet (AZ1)

PrivateSubnet1:
  Type: AWS::EC2::Subnet
  Properties:
    VpcId: !Ref VPC
```

```
      AvailabilityZone: !Select [ 0, !GetAZs '' ]
      CidrBlock: !Ref PrivateSubnet1CIDR
      MapPublicIpOnLaunch: false
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Subnet (AZ1)

NatGateway1EIP:
   Type: AWS::EC2::EIP
   DependsOn: InternetGatewayAttachment
   Properties:
   #    Domain: vpc

NatGateway1:   #works for nat gateway attachment
   Type: AWS::EC2::NatGateway
   Properties:
      AllocationId: !GetAtt NatGateway1EIP.AllocationId
      SubnetId: !Ref PublicSubnet1

 PublicRouteTable:
   Type: AWS::EC2::RouteTable
   Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Public Routes

 DefaultPublicRoute:
   Type: AWS::EC2::Route
   DependsOn: InternetGatewayAttachment
   Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway
```

```
  PublicSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PublicRouteTable
      SubnetId: !Ref PublicSubnet1

  PrivateRouteTable1:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ1)

  DefaultPrivateRoute1:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      DestinationCidrBlock: 0.0.0.0/0
      NatGatewayId: !Ref NatGateway1

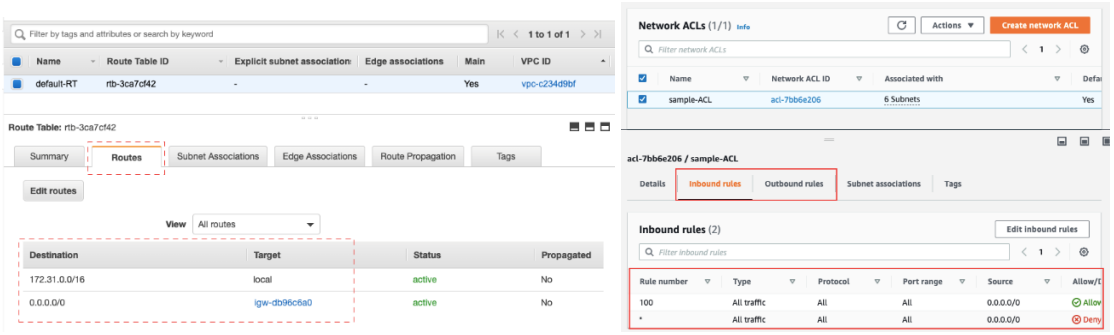
  PrivateSubnet1RouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      RouteTableId: !Ref PrivateRouteTable1
      SubnetId: !Ref PrivateSubnet1
```

____INFRASTRUCTURE DIAGRAMS____
starting point for planning your cloud infrastructure; a visual representation of the required cloud infrastructure before they turn it into code.
Intelligent Diagramming | Lucidchart -> Shapes="AWS*" – LucidPass@1

Containers – **AWS cloud** { represents AWS account and all resources it can access.}, **region** , **AZ**{ set of one or more data centers (physical building)- choose more than one to avoid single point of failure } , **VPC container** – regional – outside AZ

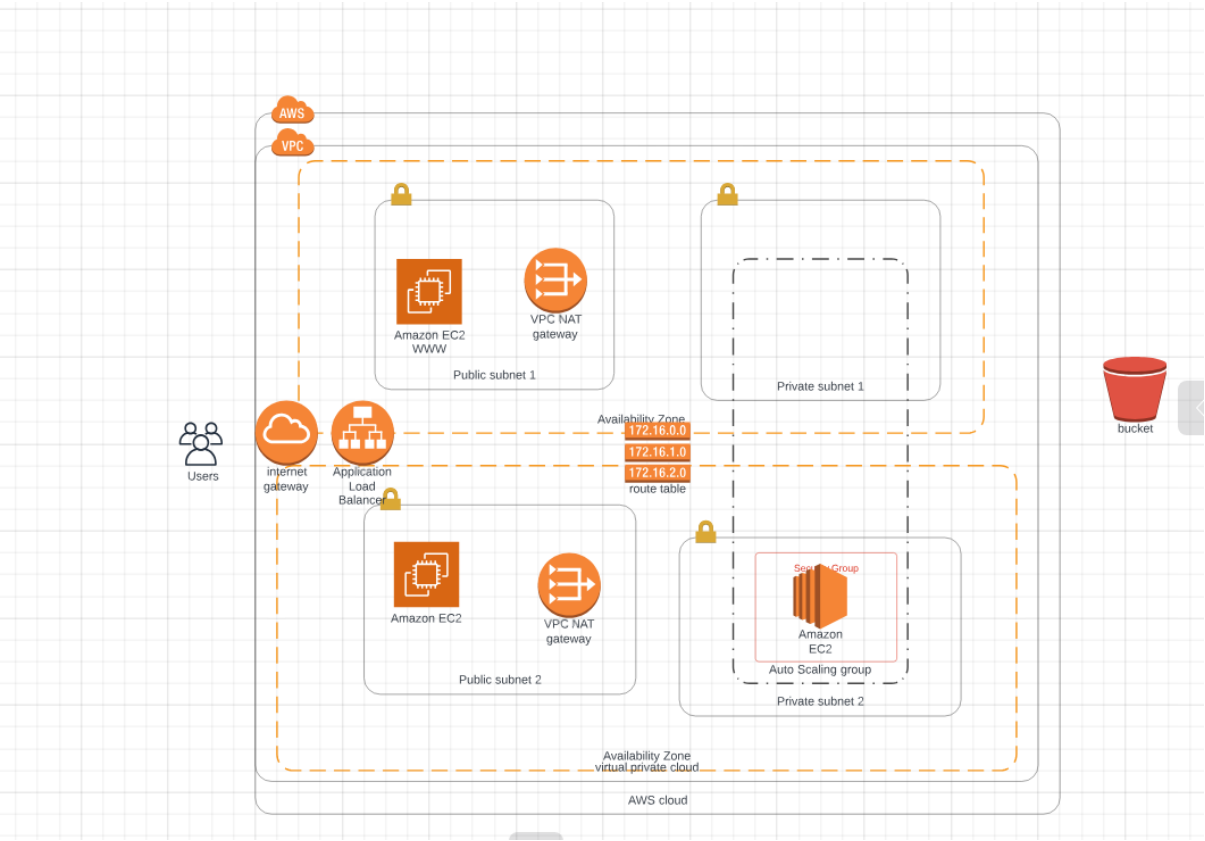
container , **Subnet container** -inside single AZ container { A subnet contains resources, and can be assigned access rights that apply to all resources within that subnet; IP addresses used as the “keys” for routing traffic.}, different for private and public subnet { Public subnets are accessible to external users eg. public webserver/load balancer. Private subnets are only accessed internally by other resources within your cloud container eg. DB/backend appln server} , **autoscaling group container** (spread across subnets {more than 1} for HA and elasticity; running the exact no of VM to meet demand by utomatically starting / stopping the servers)

Other components – **Internet Gateway** (provide inbound and outbound access to VPC by creating a route and associating it with subnet) , **NAT gateway** (1per public subnet ; provides outbound internet access to resources in private subnet { configure the NAT gateway to route the outbound requests to the Internet gateway for the VPC. While routing the outbound requests, the NAT gateway replaces the source instances' (private) IP address with NAT's own (public) IP address} by translating public to private traffic { translates the addresses back to the original source IP address}and keep private instances protected {by not exposing private resource IP address to public}from unsolicited internet inbound connectors eg. servers downloading security patches) , **ec2 instance** ,**Application Load balancer**(distribute work requests to target group{collection of servers for common service}; healthchecks,can be associated to autoscaling grp as well, w/o LB users have to hit diff url for public facing servers in more than 1 AZ), **Security group** (resource level traffic restrictions), **Route Table** (Subnet level how to MOVE TRAFFIC ;rules to allow/deny traffic ranging from entire world to single IP, if a packet arrives for a *destination* {from below ss eg, 0.0.0.0/16 public internet traffic}-it should move toward defined *target*{eg. to internet gateway}; associated with 1 or more subnet ; Network ACL also subnet level but handles what restrictions(allow/deny)are imposed using firewall



Routing rules from ss above- 1 explained above about internet gateway, other is destination 172.31.0.0/16 corresponds to VPC address and it means internal VPC traffic is allowed redirect to local target inside the VPC), **S3** (outside VPC as public service reached via NAT gateways from private subnet)

Default routing rules consists of allowing any traffic inside of VPC (be it public or private subnet resources).



Easy calculation for VPC and Subnets:

VPC CIDR block eg 10.0.0.0/16 -> now 16/8 = 2 which means top 2 fixed octets and 2 free octets -> 2 free = 256*256=65000 IP addresses

subnet – logical separation b/w resources & to allow/block access for group of resources & provide specific services to set of resources – eg. 10.0.1.0/24 which is inside above VPC as starts with 10.0 -> now 24/8 = 3 which means 3 fixed octets and 1 free octet-> 1 free = 255 available addresses in a subnet

create 255 of this smaller subnets within our VPC by doing [10.0.2.0, 10.0.2.1, 10.0.2.2, 10.0.2.3.....10.0.2.255], [10.0.3.0, 10.0.3.1, 10.0.3.2, 10.0.3.3..... 10.0.3.255], and so on

_____Software Defined Networking ____ (place it where you talked about AWS networking or other relevant area)

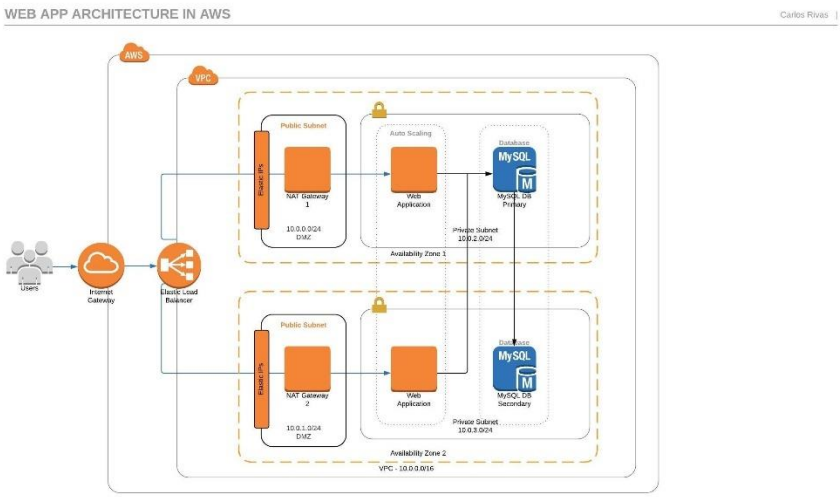
using APIs and already-existing physical infrastructure to create our own networking layer on top, with our own privacy rules, our own routing and our own Private IP Space – Types :

VPN or Virtual Private Network : encrypted connectivity between on-premise data center and Virtual Private Cloud VPC; allows access in and out of AWS VPC in a secure manner, across the internet and using internal, Private IP addresses.

DirectConnect: physical data line that you can purchase directly from AWS or through a telecommunication service provider to access your AWS Cloud without moving your data traffic across the public internet.

Internet Gateway: enables inbound and outbound traffic from the internet to your VPC ; allows external users access to communicate with parts of your VPC(public parts)**.**

____Adding servers and security groups to Cloud Formation YML script (which earlier was just Network)___



Description: >

mk / Udacity 2023

Parameters:

EnvironmentName:

Description: An environment name that will be prefixed to resource names

Type: String

Resources:

LBSecGroup: // Lb sec group

Type: AWS::EC2::SecurityGroup // Sec groups are associated to resources - instance or LB not subnets

Properties:

GroupDescription: Allow http to our load balancer

VpcId:

Fn::ImportValue:

!Sub "${EnvironmentName}-VPCID"

SecurityGroupIngress: // inbound traffic on specific ports - bydefault = deny all

- IpProtocol: tcp

FromPort: 80

ToPort: 80

CidrIp: 0.0.0.0/0 // this rule means allow inbound traffic from anywhere (public internet - 0.0.0.0/0) on port 80 with t cp protocol

SecurityGroupEgress: // outbound traffic from specific ports - - bydefault = allow all

- IpProtocol: tcp

```
      FromPort: 80
      ToPort: 80
      CidrIp: 0.0.0.0/0 // this rule means allow outbound traffic to anywhere (public internet - 0.0.0.0/0)  on tcp:80(http)  pr
otocol
    - IpProtocol: tcp
      FromPort: 8080
      ToPort: 8080 // for tomcat port
      CidrIp: 0.0.0.0/0 // for lb to send response using http as well
  WebServerSecGroup:  //creating another security group for webserver
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Allow http to our hosts and SSH from local only
      VpcId:
        Fn::ImportValue:  // how to import value from outputs defined before
          !Sub "${EnvironmentName}-VPCID"
      SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 8080
        ToPort: 8080
        CidrIp: 0.0.0.0/0
      - IpProtocol: tcp
        FromPort: 22
        ToPort: 22
        CidrIp: 0.0.0.0/0 //allow tcp:22(ssh as linux server) traffic from anywhere
      SecurityGroupEgress:
      - IpProtocol: tcp
        FromPort: 0
        ToPort: 65535
        CidrIp: 0.0.0.0/0  // all ports over tcp to send traffic to public internet(anywhere)
  WebAppLaunchConfig:
    Type: AWS::AutoScaling::LaunchConfiguration      //to tell autoscaling grp what to launch - share common configuration  ;
 you can also use AWS::EC2::LaunchTemplate instead of LaunchConfiguration{defined prior to AutoScaling grp}, AWS::EC2::
LaunchTemplate has LaunchTemplateData property which stores the same info as properties of LaunchConfiguration
    Properties:
      UserData:  // run automaically at time oif instantiating isntances
        Fn::Base64: !Sub |
```

```
        #!/bin/bash

        sudo apt-get update -y

        sudo apt-get install apache2 -
y  // install tomcat apache on ubuntu linux ; in linux you have option of <powershell> or batch_<scripts>

        sudo systemctl start apache2.service

      # ToDo: AMI ID of Ubuntu Linux machine. Too get one, try creating a VM from the web console.

      ImageId: ami-0747bdcabd34c712a

      # ToDo: Change the key-pair name, as applicable to you.

      KeyName: nd9990_keypair //ssh key- optional

      SecurityGroups:

      - Ref: WebServerSecGroup

      InstanceType: t3.medium

      BlockDeviceMappings:

      - DeviceName: "/dev/sdk"

        Ebs:

          VolumeSize: '10' //GB

  WebAppGroup:

    Type: AWS::AutoScaling::AutoScalingGroup   //logical grp of ec2 instance that ajusts automatically by adding/removing in
stances(based on scaling policy - if instance goes down {bad health} or instance's exceedes CPU utilisation or comes down b
elow a threshold or concurrent users or http reqs) to ensure desired no of instances always running

    #eg. create cloudwatch alarm with cutom metrics to count no. of web visitors in last 2 hr, if less than 100- 1 server is eno
ugh, trigger scale down if more than 1 instance

    Properties:

      VPCZoneIdentifier: //list of subnet ids for vpc where autoscaling grp can be created

      - Fn::ImportValue:

          !Sub "${EnvironmentName}-PRIV-NETS"

      LaunchConfigurationName:  //change this to LaunchTemplateId & version if using launch template instead of launch co
nfiguration

        Ref: WebAppLaunchConfig

      MinSize: '3'

      MaxSize: '5'

      TargetGroupARNs: // grp of registered instances(ARN amazon resource names) to whom traffic will be routed ; arn of L
oad balancer target grp

      - Ref: WebAppTargetGroup

  WebAppLB:

    Type: AWS::ElasticLoadBalancingV2::LoadBalancer  // single point of entry for workload distribution via public dns name

    Properties:

      Subnets:   // distributes to mentioned subnets components ; not necessary single subnet
```

```
    - Fn::ImportValue: !Sub "${EnvironmentName}-PUB1-SN"
    - Fn::ImportValue: !Sub "${EnvironmentName}-PUB2-SN"
    SecurityGroups:
    - Ref: LBSecGroup
  Listener:
    Type: AWS::ElasticLoadBalancingV2::Listener // for LB; checks for connection requests on specified protocol
    Properties:
     DefaultActions:
     - Type: forward
       TargetGroupArn:
         Ref: WebAppTargetGroup
     LoadBalancerArn:
       Ref: WebAppLB
     Port: '80' //can use 443 for https but will require certificate
     Protocol: HTTP
  ALBListenerRule:
    Type: AWS::ElasticLoadBalancingV2::ListenerRule //for LB; decides routing to targets based on conditions eg. url path {b
y adding multiple listener rules}
    Properties:
      Actions:
      - Type: forward
        TargetGroupArn: !Ref 'WebAppTargetGroup'
      Conditions:
      - Field: path-pattern
        Values: [/]  //path to listen to ; in this eg. route all req with root endpoint "/" to specified TargetGroupArn
      ListenerArn: !Ref 'Listener'  //listener of LB
      Priority: 1
  WebAppTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup //for LB and AutoScaling grp - mainly for healthcheck; grp of ec2 instan
ces managed bu autoscaling grp
    Properties:
     HealthCheckIntervalSeconds: 10  // checks for retry no. of time mentioned in UnhealthyThresholdCount in intervals of 1
0 sec
     HealthCheckPath: / //Path to check for http 200 ok code
     HealthCheckProtocol: HTTP
     HealthCheckTimeoutSeconds: 8  // wait for 8 sec to receive each response per request
```

HealthyThresholdCount: 2 // declares LB healty after 2 consecutive success health check reqs

Port: 8080

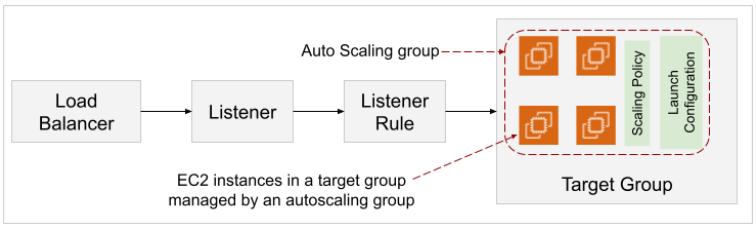
Protocol: HTTP

UnhealthyThresholdCount: 5 // declares LB unhealty after trying 5 times consecutive

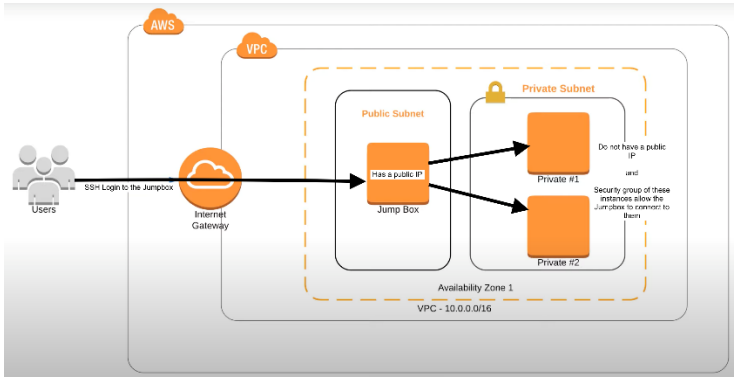
VpcId:

Fn::ImportValue:

Fn::Sub: "${EnvironmentName}-VPCID"



-JUMPBOX---
connect to private subnet servers that don't have ssh open



1. Create **2** key pair jumpbox.pem and private-servers-key.pem – region specific
2. Create jumpbox instance – ec2 on public subnet with public ip , sec grp with ssh from own ip address
3. Use putty or cmder from windows and ssh ec2-user@<public ip> -I jumpbox.pem – test able to login
4. Scp -I private-servers-key.pem ec2-user@<privateServer Private IP>:/home/ec2-user/ private-servers-key.pem
5. ssh ec2-user@< jumpboxServer public ip > -i jumpbox.pem
6. From inside of jumpbox server now , Chmod 400 private-servers-key.pem
7. From inside of jumpbox server now, ssh ec2-user@< privateServer Private IP > -i private-servers-key.pem ➔ works as same vpc

____Additional services using cloudFormation_____

RDS – relational data service : aurora/MySQL/MariaDB/Oracle/PostgreSQL/Microsoft SQL server
Single RDS server can host multiple dBs; usually DB are put in private subnets{via private IP address} with no public accessibility ; Read replica can be used to accommodate statistical reporting and read only queries, served by separate copy of DB – AWS::RDS::DBInstance & AWS::RDS::DBSubnetGroup

Filestores – videos/text docs/config files – S3 – global service
aws s3 ls //lists all buckets
aws s3 ls <link to s3>> //lists all files in bucket
aws s3 cp <filename> <link to s3> //copies a file from local machine to s3

CloudFormation retention policy

DeletionPolicy: retain ; additional prop on any resource to data to persist even if your stack of resources is updated or deleted. useful to assign to your data storage (database, file storage)

Project 2 - **Deploy a high-availability web app using CloudFormation**

dummy application - HTML file to Apache webserver running{application needs to be deployed into private subnets with a Load Balancer located in a public subnet} on ec2 instance (Launch configuration used by autoscaling grp-
deploy four servers, two located in each of your private subnets-
two vCPUs and at least 4GB of RAM, Ubuntu 18, 10GB of disk space - select instance type and AMI accordingly-t3.small)

IAM role - allows your instances to use the S3 Service

Udagram communicates on the default HTTP Port: 80, so your servers will need this inbound port open since you will use it with the Load Balancer and the Load Balancer Health Check. As for outbound, the servers will need unrestricted internet ac cess to be able to download and update their software.

The load balancer should allow all public traffic (0.0.0.0/0) on port 80 inbound. Outbound, it will only be using port 80 to re ach the internal servers.

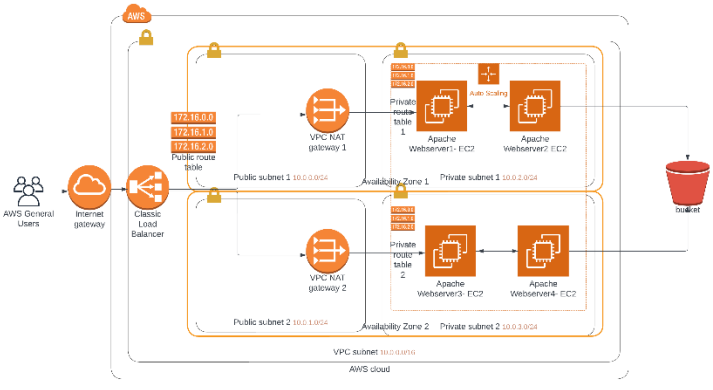
output exports of the CloudFormation script should be the public URL of the LoadBalancer. Bonus points if you add http:// i n front of the load balancer DNS Name in the output

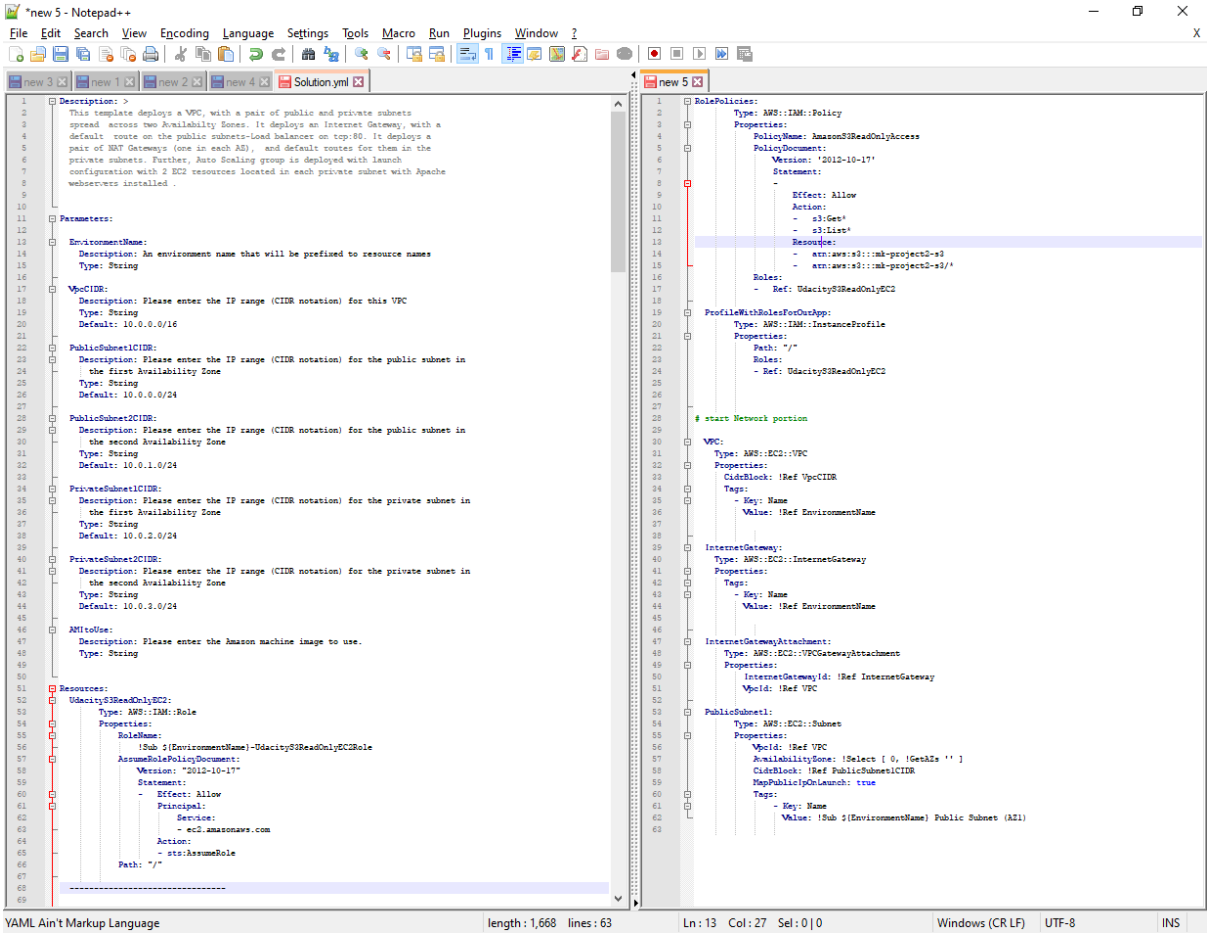
Other considerations

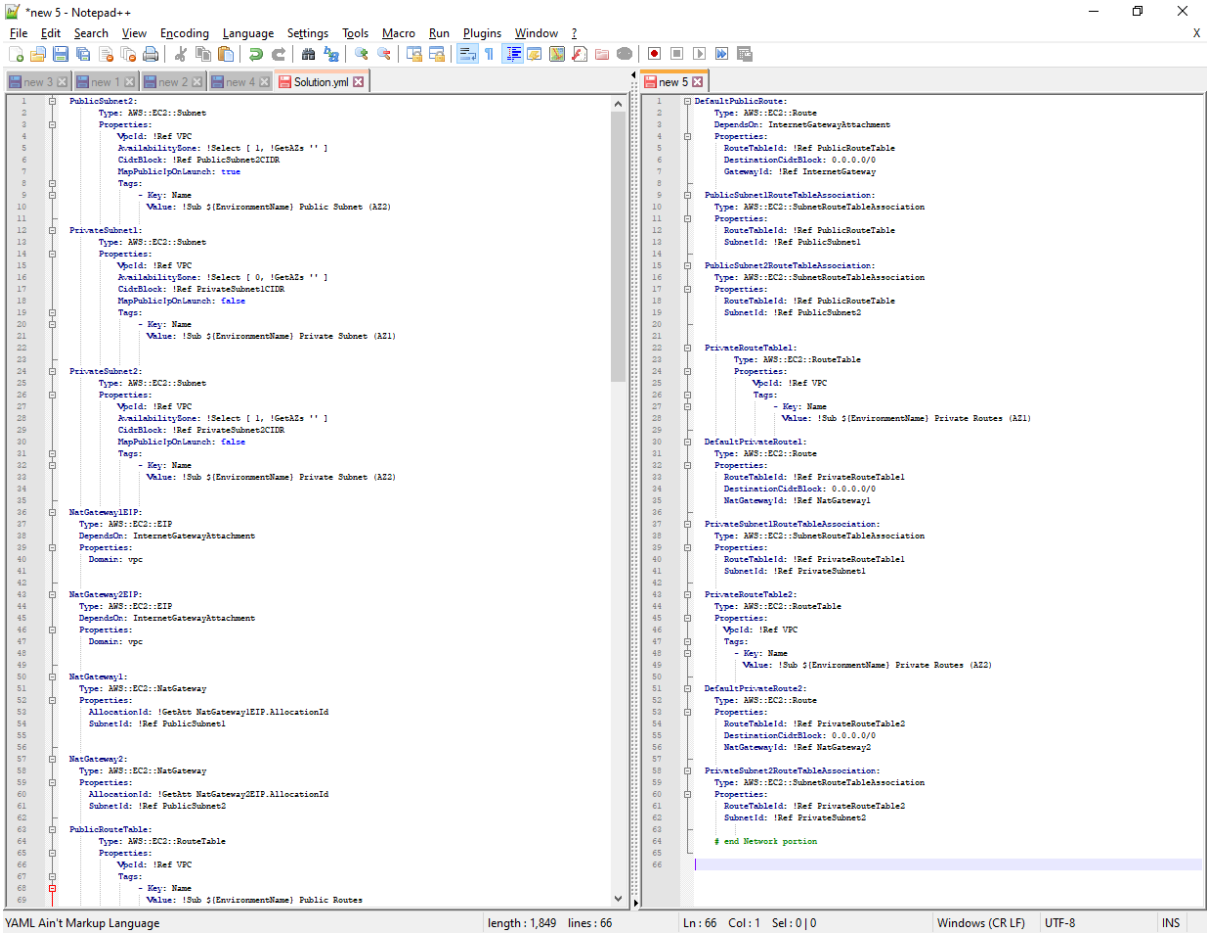
Log information for UserData scripts is located in this file: cloud-init-output.log under the folder: /var/log
&& User Data script for
Apache &&reosurces - LB, launch config, autoscaling grp with health check port 80, sec grps port 80, listener port 80

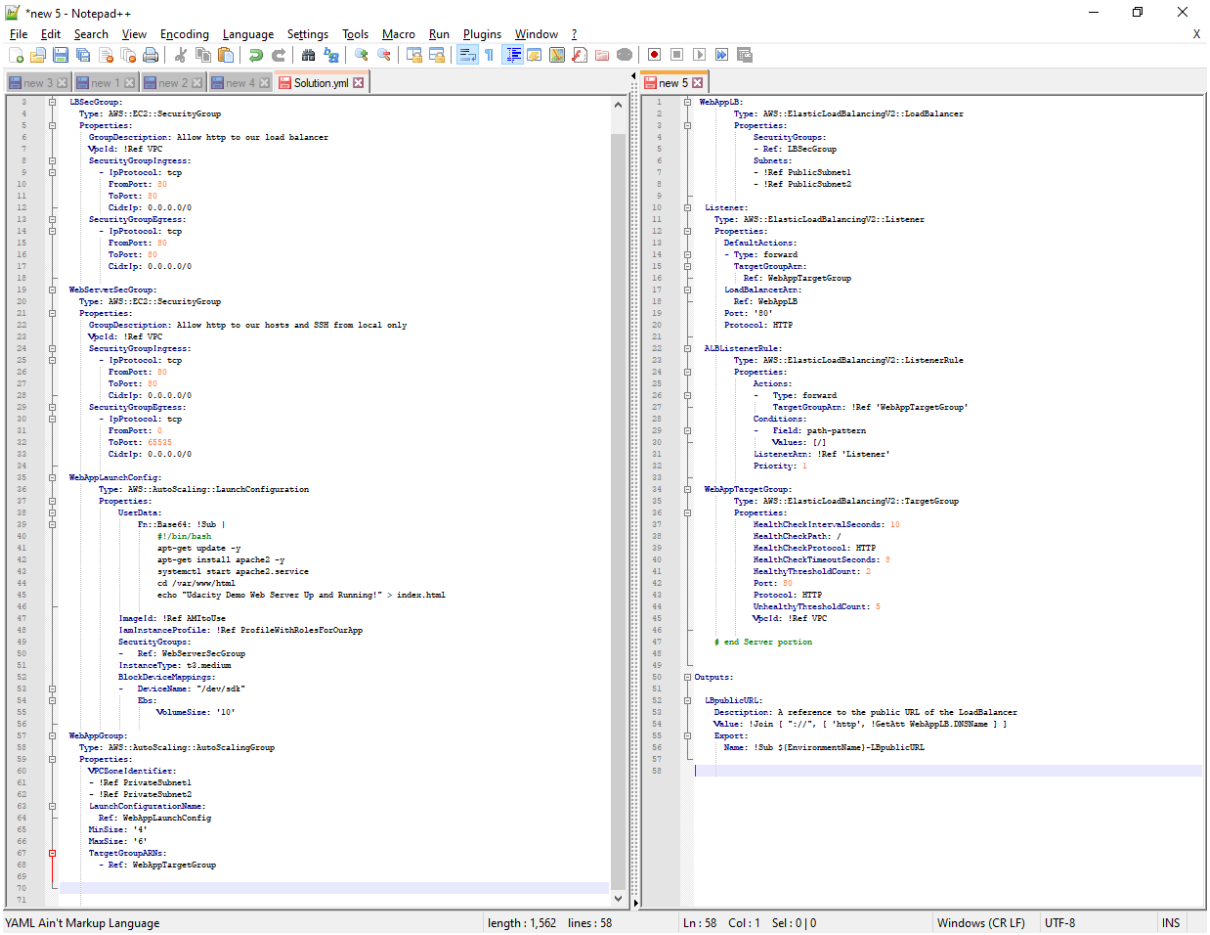
Udagram Infra Diagram

Parameter.json and Script.yml




*new 5 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 3
new 1
new 2
new 4
Solution.yml
new 5
Description: >
This template deploys a VPC, with a pair of public and private subnets
spread across two Availabilty Zones. It deploys an Internet Gateway, with a
default route on the public subnets.Load balancer on tcp:80. It deploys a
pair of NAT Gateways (one in each AZ), and default routes for them in the
private subnets. Further, Auto Scaling group is deployed with launch
configuration with 2 EC2 resources located in each private subnet with Apache
webservers installed .
Parameters:
EnvironmentName:
Description: An environment name that will be prefixed to resource names
Type: String
VpcCIDR:
Description: Please enter the IP range (CIDR notation) for this VPC
Type: String
Default: 10.0.0.0/16
PublicSubnet1CIDR:
Description: Please enter the IP range (CIDR notation) for the public subnet in
the first Availability Zone
Type: String
Default: 10.0.0.0/24
PublicSubnet2CIDR:
Description: Please enter the IP range (CIDR notation) for the public subnet in
the second Availability Zone
Type: String
Default: 10.0.1.0/24
PrivateSubnet1CIDR:
Description: Please enter the IP range (CIDR notation) for the private subnet in
the first Availability Zone
Type: String
Default: 10.0.2.0/24
PrivateSubnet2CIDR:
Description: Please enter the IP range (CIDR notation) for the private subnet in
the second Availability Zone
Type: String
Default: 10.0.3.0/24
AMItoUse:
Description: Please enter the Amazon machine image to use.
Type: String
Resources:
UdacityS3ReadOnlyEC2:
Type: AWS::IAM::Role
Properties:
RoleName:
!Sub ${EnvironmentName}-UdacityS3ReadOnlyEC2Role
AssumeRolePolicyDocument:
Version: "2012-10-17"
Statement:
- Effect: Allow
Principal:
Service:
- ec2.amazonaws.com
Action:
- sts:AssumeRole
Path: "/"
RolePolicies:
Type: AWS::IAM::Policy
Properties:
PolicyName: AmazonS3ReadOnlyAccess
PolicyDocument:
Version: '2012-10-17'
Statement:
-
Effect: Allow
Action:
- s3:Get*
- s3:List*
Resource:
- arn:aws:s3:::mk-project2-s2
- arn:aws:s3:::mk-project2-s2/*
Roles:
- Ref: UdacityS3ReadOnlyEC2
ProfileWithRolesForOurApp:
Type: AWS::IAM::InstanceProfile
Properties:
Path: "/"
Roles:
- Ref: UdacityS3ReadOnlyEC2
start Network portion
VPC:
Type: AWS::EC2::VPC
Properties:
CidrBlock: !Ref VpcCIDR
Tags:
- Key: Name
Value: !Ref EnvironmentName
InternetGateway:
Type: AWS::EC2::InternetGateway
Properties:
Tags:
- Key: Name
Value: !Ref EnvironmentName
InternetGatewayAttachment:
Type: AWS::EC2::VPCGatewayAttachment
Properties:
InternetGatewayId: !Ref InternetGateway
VpcId: !Ref VPC
PublicSubnet1:
Type: AWS::EC2::Subnet
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [0, !GetAZs '']
CidrBlock: !Ref PublicSubnet1CIDR
MapPublicIpOnLaunch: true
Tags:
- Key: Name
Value: !Sub ${EnvironmentName} Public Subnet (AZ1)
YAML Ain't Markup Language
length : 1,668 lines : 63
Ln : 13 Col : 27 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS


*new 5 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 3
new 1
new 2
new 4
Solution.yml
new 5
PublicSubnet2:
Type: AWS::EC2::Subnet
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [1, !GetAZs '']
CidrBlock: !Ref PublicSubnet2CIDR
MapPublicIpOnLaunch: true
Tags:
- Key: Name
Value: !Sub ${EnvironmentName} Public Subnet (AZ2)
PrivateSubnet1:
Type: AWS::EC2::Subnet
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [0, !GetAZs '']
CidrBlock: !Ref PrivateSubnet1CIDR
MapPublicIpOnLaunch: false
Tags:
- Key: Name
Value: !Sub ${EnvironmentName} Private Subnet (AZ1)
PrivateSubnet2:
Type: AWS::EC2::Subnet
Properties:
VpcId: !Ref VPC
AvailabilityZone: !Select [1, !GetAZs '']
CidrBlock: !Ref PrivateSubnet2CIDR
MapPublicIpOnLaunch: false
Tags:
- Key: Name
Value: !Sub ${EnvironmentName} Private Subnet (AZ2)
NatGateway1EIP:
Type: AWS::EC2::EIP
DependsOn: InternetGatewayAttachment
Properties:
Domain: vpc
NatGateway2EIP:
Type: AWS::EC2::EIP
DependsOn: InternetGatewayAttachment
Properties:
Domain: vpc
NatGateway1:
Type: AWS::EC2::NatGateway
Properties:
AllocationId: !GetAtt NatGateway1EIP.AllocationId
SubnetId: !Ref PublicSubnet1
NatGateway2:
Type: AWS::EC2::NatGateway
Properties:
AllocationId: !GetAtt NatGateway2EIP.AllocationId
SubnetId: !Ref PublicSubnet2
PublicRouteTable:
Type: AWS::EC2::RouteTable
Properties:
VpcId: !Ref VPC
Tags:
- Key: Name
Value: !Sub ${EnvironmentName} Public Routes
DefaultPublicRoute:
Type: AWS::EC2::Route
DependsOn: InternetGatewayAttachment
Properties:
RouteTableId: !Ref PublicRouteTable
DestinationCidrBlock: 0.0.0.0/0
GatewayId: !Ref InternetGateway
PublicSubnet1RouteTableAssociation:
Type: AWS::EC2::SubnetRouteTableAssociation
Properties:
RouteTableId: !Ref PublicRouteTable
SubnetId: !Ref PublicSubnet1
PublicSubnet2RouteTableAssociation:
Type: AWS::EC2::SubnetRouteTableAssociation
Properties:
RouteTableId: !Ref PublicRouteTable
SubnetId: !Ref PublicSubnet2
PrivateRouteTable1:
Type: AWS::EC2::RouteTable
Properties:
VpcId: !Ref VPC
Tags:
- Key: Name
Value: !Sub ${EnvironmentName} Private Routes (AZ1)
DefaultPrivateRoute1:
Type: AWS::EC2::Route
Properties:
RouteTableId: !Ref PrivateRouteTable1
DestinationCidrBlock: 0.0.0.0/0
NatGatewayId: !Ref NatGateway1
PrivateSubnet1RouteTableAssociation:
Type: AWS::EC2::SubnetRouteTableAssociation
Properties:
RouteTableId: !Ref PrivateRouteTable1
SubnetId: !Ref PrivateSubnet1
PrivateRouteTable2:
Type: AWS::EC2::RouteTable
Properties:
VpcId: !Ref VPC
Tags:
- Key: Name
Value: !Sub ${EnvironmentName} Private Routes (AZ2)
DefaultPrivateRoute2:
Type: AWS::EC2::Route
Properties:
RouteTableId: !Ref PrivateRouteTable2
DestinationCidrBlock: 0.0.0.0/0
NatGatewayId: !Ref NatGateway2
PrivateSubnet2RouteTableAssociation:
Type: AWS::EC2::SubnetRouteTableAssociation
Properties:
RouteTableId: !Ref PrivateRouteTable2
SubnetId: !Ref PrivateSubnet2
end Network portion
YAML Ain't Markup Language
length : 1,849 lines : 66
Ln : 66 Col : 1 Sel : 0 | 0
Windows (CR LF)
UTF-8
INS



[GitHub - mehmetincefidan/Deploy-a-High-Availability-Web-App-Using-CloudFormation: In this project (Udagram App), I deployed web servers for a highly available web app using CloudFormation.]

[GitHub - salahbeeh/Deploy-a-high-availability-web-app-using-CloudFormation: Deploying web servers for a highly availability web app using CloudFormation.]

-------Course 3------

___CICD____

Continuous Integration + Continuous Deployment = Continuous Delivery (Brilliant maths)

Continuous Integration = practice of merging all developers' working copies to a shared mainline several times a day. (Compile ➔ Unit Test➔Static Analysis➔Dependency vulnerability testing➔Store artifact) ; everything to do with source code. Eg. VCS root in teamcity (fetching code from different gitlab repos) ;

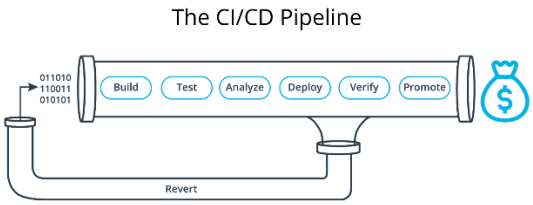
Continuous Deployment = software engineering approach in which the value is delivered frequently through automated deployments.(Creating **infrastructure**➔Provisioning servers➔Copying files➔Promoting to production➔Smoke Testing (aka Verify) ➔Rollbacks) ; everything to do with built code and deployment

Continuous Delivery = engineering practice in which teams produce and release value in short cycles.

8 Principles of Continuous Delivery

1. Repeatable Reliable Process
2. Automate Everything
3. Version Control Everything
4. Bring the Pain Forward
5. Build-in Quality
6. "Done" Means Released
7. Everyone is Responsible
8. Continuous Improvement

Tools – Github, AWS, CircleCI, VS code



----Deployment Strategies -------

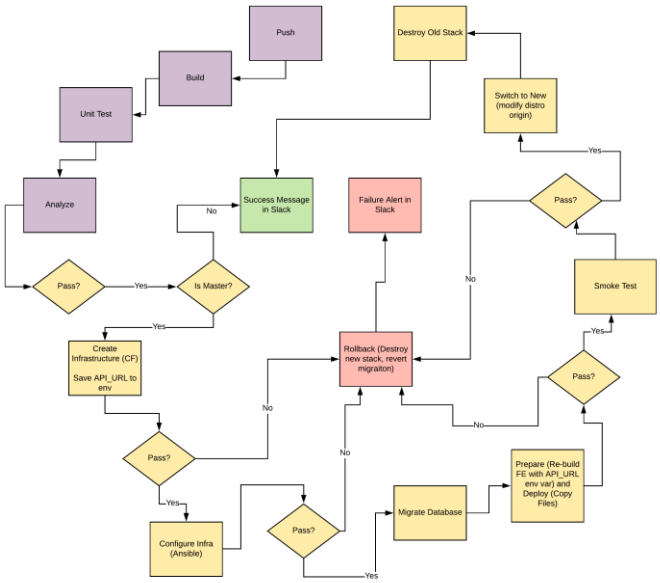
Big-Bang :: Replace A with B all at once.

Blue Green :: Two versions of production: Blue or previous version and Green or new version. Traffic can still be routed to blue while testing green. Switching to the new version is done by simply shifting traffic from blue to green via router (LoadBalancer{instant switch} or CDN{instant switch} or DNS{slow switch because of DNS propagation}) ; compile➔unit tests➔provision➔deploy➔smoke tests➔switch router➔sanity test➔destroy old env

Canary :: Aka Rolling Update, After deploying the new version, start routing traffic to new version little by little until all traffic is hitting the new production. Both versions coexist for a period of time.

A/B Testing:: Similar to Canary, but instead of routing traffic to new version to accomplish a full deployment, you are testing your new version with a subset of users for feedback. You might end up routing all traffic to the new version, but that's always the goal.

Sample pipeline :



Available CICD tools – On prem - Jenkins, Gitlab, teamcity, Microsoft team foundation; Cloud – Bamboo, Travis CI, Gitlab, Circle CI

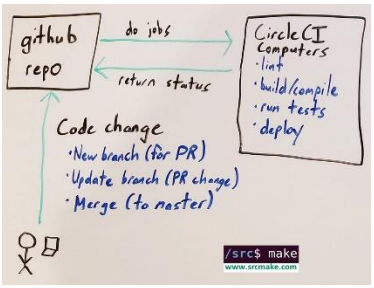
---CI pipeline---

Pipelines have **workflows** that have **jobs** that have **steps.**

Circle CI uses a yaml file to handle the configuration of pipelines and jobs. Various sections of pipeline: Version {circleCI runner version}, jobs, commands {reusable commands to be used in steps of jobs}, orbs {pre written functionality to be included in jobs}, workflows {puts jobs in execution order and setup dependencies [job_name, “requires” field, “filters” field]; 1or more workflows}

Circle CI

Associated with github account ; service that detects when your codebase changes and runs jobs (that you define) on the code; ci {code compile build, runs tests} cd {deploy code to infra aws/heroku}



pipeline – individual trigger or run of one or more workflows ; workflows - one or more jobs {enable running jobs concurrently, sequentially, on a schedule, or manually using an approval job}; jobs – collection of execution steps

Step1 – github repo
Step2 – Circle CI project from github repo containing .circleci folder with config.yml
step3 – commit config.yml and pipeline runs automatically

Eg. Demo react build

version: 2.1 //circle ci RUNNER api version

workflows: //workflow section

test-deploy: // workflow name

jobs: // list of jobs defined in workflow to be executed in order Y

- test

jobs: // jobs section

test: //job name , docker image and Steps

docker: //docker image name for setup environment ;; docker executer to specify what containers to start for this “test” job ; Each docker image comes pre-loaded with utilities and packages you need for different situations.

- image: cimg/node:17.2.0

steps: //single job steps

- checkout

- run: node –version //run command

Other Circle CI features -

Orbs - reusable , sharable package for circle ci configuration – integration eq. aws, azure, docker, slack, node – to use steps without definition in config.yml but just declaring and calling orbs

Filters - run the pipeline on a specified branch or a tag

Requires - when job depends on another one

Commands – reusable block of code to reference them in other places eg, destroy-environment command which takes workflow id and event{eg. On fail} as parameter

Branches – to run a job only on certain branches

Persists to workspace – send certain file or folder to a shared area

Other docker images -

circleci/node	For Node.JS server-side and networking applications.
circleci/postgres	For tasks that require PostgreSQL database functions.

`circleci/python`	For job that need to run Python or pip.
`alpine:latest`	A lightweight container for simple tasks.
`amazon/aws-cli`	For tasks that require the AWS CLI and related tools.

Other steps type -

`checkout`	Checks out the source code. Common to have this in all jobs in CI.
`run`	Runs a shell command. Can name the step or simply execute a script.
`when`	A conditional step that has its own steps that are run if the condition is true.
`save_cache` and `restore_cache`	Save and restore files or folders. Cleaned up after pipeline finishes.
`persist_to_workspace` and `attach_workspace`	Like a cache, but files are available for 15 days after pipeline.
`add_ssh_keys`	Adding some additional ssh keys to the job for a tool that needs them (i.e. Ansible).
`store_artifacts`	Makes an artifact, or file, available for download via CircleCI web app or API.
`store_test_results`	Stores test results from test runner so that results are visible in Circle CI web app in the Test Summary section.
"Orbs"	Orbs, which we talked about already, are used like step types.

Eg2, python project https://www.srcmake.com/home/circleci

MAIN.py

```
1 def Add(a, b):
2         return a + b
3
4 def SayHello():
5         print("sup world from srcmake")
6
7 if __name__ == '__main__':
```

```
8         SayHello()
```

MAIN-TEST.py

```
1
2 # Import the Add function, and assert that it works correctly.
3 from main import Add
4
5 def TestAdd():
6         assert Add(2,3) == 5
7         print("Add Function works correctly")
8
9 if __name__ == '__main__':
          TestAdd()
```

CONFIG.yml

```
version:
2.1


        jobs:
          build:
            working_directory: ~/circleci-python
            docker:
              - image: "circleci/python:3.6.4"
            steps:
              - checkout
              - run: python3 main.py
          test:
            working_directory: ~/circleci-python
            docker:
              - image: "circleci/python:3.6.4"
            steps:
              - checkout
              - run: python3 main-test.py


        workflows:
          build_and_test:
            jobs:
              - build
              - test:
                  requires:
```

```
      - build
```

Pipeline cotd. -> CI Section

Eg 3 : workflow with 2 jobs to print hello and world

```
# Use the latest 2.1 version of CircleCI pipeline process engine.
See: https://circleci.com/docs/2.0/configuration-reference

version: 2.1
# Use a package of configuration called an orb.
# Orchestrate or schedule a set of jobs
jobs:
  print_hello:
    docker:
      - image: circleci/node:13.8.0
    steps:
      - run: echo hello
  print_world:
    docker:
      - image: circleci/node:13.8.0
    steps:
      - run: echo world
workflows:
  # Name the workflow "welcome"
  welcome:
    # Run the welcome/run job in its own container
    jobs:
      - print_hello
      - print_world:
          requires:
            - print_hello
```

Global values:

Pipeline values: used like My pipeline id is << pipeline.id >> and my git branch is << pipeline.git.branch >>.

pipeline.id	The ID of the currently running pipeline
pipeline.number	An alternative numeric ID for the currently running pipeline
pipeline.project.git_url	The URL of the triggering git event (ex: pull request URL)
pipeline.project.type	Example: “github”
pipeline.git.branch	The branch triggering the pipeline

Env variables: also used to store secrets

SCOPE inheritance : Organisations -> Project-> Pipeline Job

Built in env vars :

`CIRCLE_BRANCH`	The name of the Git branch currently being built

`CIRCLE_WORKFLOW_ID`
 	
`CIRCLE_BUILD_NUM`
	A unique identifier for the workflow instance of the current job
	
	The number of the CircleCI build

`CIRCLE_PR_NUMBER`	The number of the associated GitHub or Bitbucket pull request
`CIRCLE_SHA1`	The SHA1 hash of the last commit of the current build

Eg, set env var in projects settings and use via –

```
      - run:
```

```
          name: "echo an env var that is part of our project"
          command: |
            echo $MY_ENV_VAR
```

Triggering : to run circle Ci pipeline ; mamnaged by webhooks{ for connection between circle ci project and github repo, a webhook is created automatically when you configure your proj in circle ci console to listen to events like push/commit on github repo; set up webhooks requires end point payload url to send events to – circle ci, type of events – selected/all, secrets – optional, content type –application json }

Git Branch Commit	Commit or merge to a branch-like `master` and push changes to the branch in central repository to start a new build.
New Pull/Merge Request	Make changes in a branch or fork and create a pull/merge request to trigger a build.
API	Make a `POST` or `GET` request to an API endpoint to kick off a new build.
Schedule	Run a pipeline at a certain time each day or week based on a schedule.
Other Pipelines	Another pipeline might finish a job and then trigger another pipeline.
Chat Message	Using a chat tool, post a message containing special text in order to trigger a build.

Command-Line Tool	Use a command-line tool to configure and start a new build.

Sharing information : between jobs – 3 ways
//by default files and memory that are used during the job are gone after the job completes. For job isolation
Cache {using keys; immutable }

```
- save_cache:
          key: v1-my-project-{{ checksum "project.clj" }}
          paths:
            - ~/.m2
```

```
  - restore_cache:
          keys:
            - v1-my-project-{{ checksum "project.clj" }}
            - v1-my-project-
```

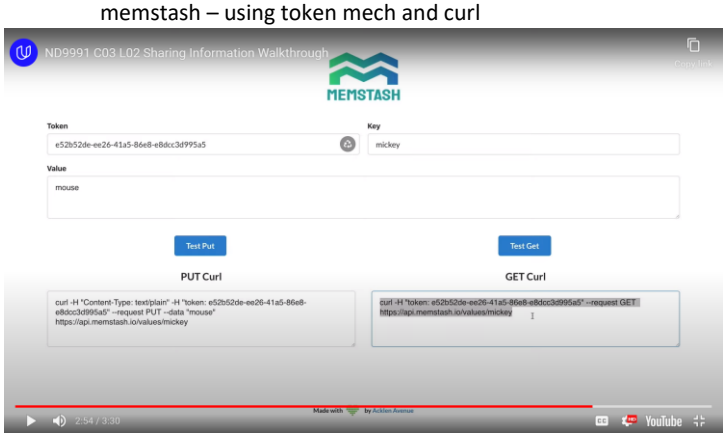
Template variables for cache to form keys - {{ .Branch }} / {{ .BuildNum }} / {{ .Environment.variableName }} / {{ checksum "filename" }} / {{ epoch }}

Workspace {15 days retention; mutable }

```
- persist_to_workspace:
          root: /tmp/workspace
          paths:
            - target/application.jar
            - build/*
```

```
  - attach_workspace:
          at: /tmp/workspace
```

Secret keeper – for bits of information w/o files and folders (unlike cache and workspaces) – 3rd party
harshicorps's vault - web service and command line tool that work together to give you the ability to save bits of information securely and then retrieve it later.
memstash – using token mech and curl



KVdb REST API – storing key values in a bucket

```
curl -d 'email=your-email-ID@domain.com' https://kvdb.io
```
- > create bucket

```
curl https://kvdb.io/[bucket-ID]/[key]  -d '[value]'
```
-> store key value

```
curl --insecure  https://kvdb.io/[bucket-ID]/[key]
```
-> retrieve using key

Eg 4. : persists to workspace for output.txt

```
version: 2.1

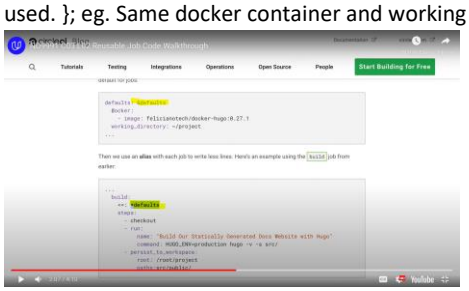
jobs:
  save_hello_world_output:
    docker:
      - image: circleci/node:13.8.0
    steps:
      - run: echo "hello world" > ~/output.txt
      - persist_to_workspace:
          root: ~/
          paths:
            - output.txt

  print_output_file:
    docker:
      - image: circleci/node:13.8.0
    steps:
      - attach_workspace:
          at: ~/
      - run: cat ~/output.txt

workflows:
  my_workflow:
    jobs:
      - save_hello_world_output
      - print_output_file:
          requires:
            - save_hello_world_output
```

Reusable code – common defaults , rollback when a job fails , standard code compilation, same deploymrnt plan for all environments

Anchors and alias mechanism is used for this purpose where anchors define set of reusable yaml syntax {Uses a & symbol to signal that an anchor is being defined. } and aliases are used to call anchors {Uses the * to signal that an alias is being used. }; eg. Same docker container and working directory for all jobs



Commands- to reuse multiple steps in one line , supports parameters , acts like a function

Example:

```
commands:
  sayhello:
    description: "A very simple command for demonstration purposes"
    parameters:
      to:
        type: string
        default: "Hello World"
    steps:
      - run: echo << parameters.to >>
```

```
jobs:
  myjob:
    docker:
      - image: "circleci/node:9.6.1"
    steps:
      - sayhello:
          to: "Lev"
```

o/p – LEV {if not passed anything, then hello world}

```
version: 2.1

commands:
  print_pipeline_id:
    steps:
      - run: echo ${CIRCLE_WORKFLOW_ID}

jobs:
  my_job:
    docker:
      - image: circleci/node:13.8.0
    steps:
      - print_pipeline_id

workflows:
  my_workflow:
    jobs:
      - my_job
```

Job Failure – non 0 exit codes ; pipeline will stop; eg. Send a msg when a job fails, or send any rollback command, or alert another service ; using “when” keyword
when: on_fail // any previous steps fails

```
steps:

  - run:
      name: Testing application
      command: make test
      shell: /bin/bash
      working_directory: ~/my-app
      no_output_timeout: 30m
      environment:
        FOO: bar

  - run: echo 127.0.0.1 devhost | sudo tee -a /etc/hosts

  - run: |
      sudo -u root createuser -h localhost --superuser ubuntu &&
      sudo createdb -h localhost test_db

  - run:
      name: Upload Failed Tests
```

```
      command: curl --data fail_tests.log
http://example.com/error_logs
      when: on_fail
```

eg,

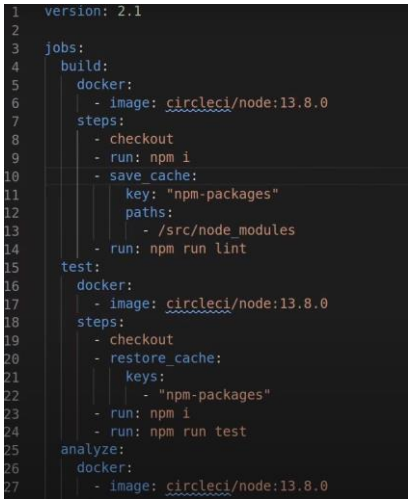
```
version: 2.1

jobs:
  my_job:
    docker:
      - image: circleci/node:13.8.0
    steps:
      - run: exit 1            //simulating error
      - run:
          name: on error
          command: echo "Hello Error!"
          when: on_fail

workflows:
  my_workflow:
    jobs:
      - my_job
```

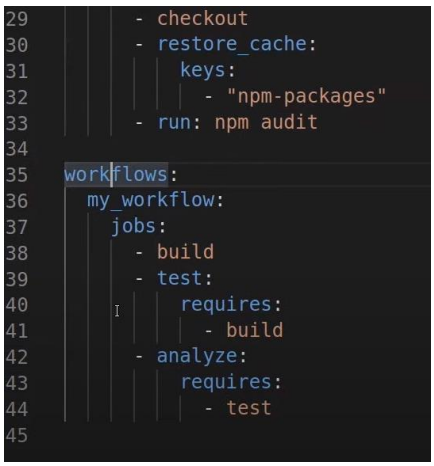
Most common CI stages – compile, test and analyse

Stage	Description
Compile	Convert programmer code from text format to zero's and one's, or whatever is needed by the final runtime.
Test	Run unit or integration tests against the code to make sure it is up to specifications.
Analyze	Perform some deeper tests on the code such as static analysis or a package security audit.



```
version: 2.1

jobs:
  build:
    docker:
      - image: circleci/node:13.8.0
    steps:
      - checkout
      - run: npm i
      - save_cache:
          key: "npm-packages"
          paths:
            - /src/node_modules
      - run: npm run lint
  test:
    docker:
      - image: circleci/node:13.8.0
    steps:
      - checkout
      - restore_cache:
          keys:
            - "npm-packages"
      - run: npm i
      - run: npm run test
  analyze:
    docker:
      - image: circleci/node:13.8.0
```



```
      - checkout
      - restore_cache:
          keys:
            - "npm-packages"
      - run: npm audit

workflows:
  my_workflow:
    jobs:
      - build
      - test:
          requires:
            - build
      - analyze:
          requires:
            - test
```

Build – checkout; install npm packages, save those to cache, run lint
Test – checkout, restore cache for npm packages, install those packages and run test
Analyse - checkout, restore cache for npm packages, run audit

This was all CI,,now lets see CD

After generating application artefact{docker image/zipfile or executable } from CI, CD deploys artefact to reach production with steps like validating artefact, build infra to host artefact, provision servers, copy files to destn

CD metrics

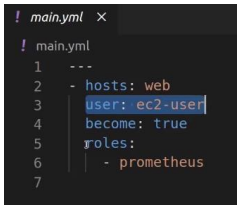
Metric	Formula	Impact
Lead Time to Production	(*Time at Successful Prod Deployment*) - (*Time at Completion of Feature Grooming*)	Shows how CI/CD is impacting overall delivery time, from the point the team hears about the feature to the point at which it is done (deployed to production). Easy metric to collect if using task management system to track feature grooming and deployments.
Rollback Rate	(*Total Rollbacks*) / (*Total Deployments*)	Shows the quality of our deployments. Of course, rate should be low because previous stages should filter out defected builds. This metric is a leading indicator for the confidence of the business in the dev team's ability to delivery.
Time to Failure	(*Time at Failure Discovery*) - (*Time at Build Start*)	Shows how quickly we find failures. The lower the better.
Production Uptime	(*Total Production Working Time*) / (*Total Time*)	Shows the amount of time we are taking production down because of botched deployments or due to our chosen deployment strategy.
Failed Pipeline Cost	Various calculations including job run time and resources created	Shows the estimated amount of money spent on a failed build. Encourages us to put cheaper jobs earlier in the pipeline.

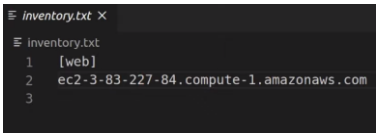
Configuration management - after servers are already spinned up and require internal configuration, brute force – ssh to each of 1000 webservers and execute commands, Automated approach – Chef {depends on agent to be installed} / Puppet { Requires master "puppet master" server}/ saltStack { inventory on a central server }/ ansible {agentless,popular,fast uses ssh via shell and bash to connect to remote machine ; Ansible uses yaml files called Playbooks to manage all the configurations to be made }

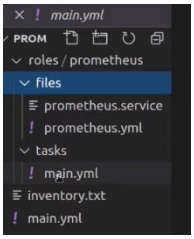
Ansible : :: : npm install ansible
eg, ec2 setup with key pair and downloaded pem file can be installed with prometheus webserver on port 9090
steps – add hostname to inventory file
run ansible playbook - `ansible-playbook -i inventory.txt main.yml --private-key=~/udacity-ansible.pem`
check on public url from browser with port 9090

Playbook {entry point} – plays {instruction for individual step like updating webserver , migrating db, installing dependencies etc. }, auth { Ansible executes bash commands over SSH, username [defaults- ec2-user/ubuntu] and ssh key-key pair or pem file }, host targets {machines to connect to, specify one hostname, a group name, or use a pattern to select multiple hosts from an inventory list}, roles {child playbooks logically divided ; each sub-folder of each role must have a `main.yml` file in it, role components – tasks, files{that role deploys}, handlers, library, defaults vars, templates. meta}, tasks {Contains the ordered series of commands to run on the identified hosts. Sometimes, it contains Modules, which are like library function – shell/apt/npm/file/git/script/copy/systemd}

Playbooks detailed :







In above eg., prometheus role has 2 compo – files {which will get copied – prometheus.service & prometheus.yml }and tasks {main.yml- which roles executes}

Other eg. of roles ,

```
├── main1.yml      # Playbook file (Ignore file name)
└── roles
       ├── configure-prometheus-node-exporter
       │   └── tasks
       │       └── main.yml
       └── configure-server
           └── tasks
               └── main.yml
```

EXERCISE 1:Ansible playbook to print content of env variable to console via role

Main.yml ::

```
- name: Exercise
  hosts: localhost //w/o inventory file
  roles:
    - print
```

roles/print/tasks/main.yml ::

```
---
 - name: Print env variable
    shell: echo $PATH      //env variable to be printed
    register: print_result

 - name: print message
    debug:
      msg: "{{ print_result.stdout_lines }}"
```

Test using : ansible-playbook main.yml command

Inventory file – manage and query server instances , getting list of instances {using public ips} from aws using describe-instances query and append

```
echo "[all]" > inventory
aws ec2 describe-instances \

    --query 'Reservations[*].Instances[*].PublicIpAddress' \
    --output text >> inventory
```

O/p -

```
[all]
3.83.227.84
```

Additional – add tag project= udacity while creating ec2 instance and use aws cli command as `aws ec2`

```
describe-instances \

\
        --query 'Reservations[*].Instances[*].PublicIpAddress' \
      --filters "Name=tag:Project,Values=udacity" \
      --output text >> inventory
```

Excercise2: Install apache , copy html for hello world

Main.yml::

```
# Remote Control Using Ansible

- name: Remote Control Using Ansible

  hosts: ubuntu

  user: ubuntu
```

```
  become: true
  become_method: sudo
  become_user: root
  roles:
    - prepare
    - apache
```

roles/apache/files/index.html :: //file that needs to be copied

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Hello world!
        </title>
    </head>
    <body>
        <h1>
            Hello World, This page is served from Apache!
        </h1>
    </body>
</html>
```

roles/apache/tasks/main.yml ::

```
---
- name: "Install Apache2"   //play1
  become: true //elevating privileges for this command like sudo
  apt:
    name: ["apache2"]
    state: latest
    update_cache: yes


- name: "Install index.html page" //play2
```

```
  copy: //copy module
    src: index.html //ansible knows to look in files/
    dest: /var/www/html/index.html // folder in remote machine
    backup: yes
```

roles/prepare/tasks/main.yml ::

```
---
- name: "updadte apt packages" //play1
  become: true
  become_method: sudo
  apt:
    update_cache: yes

- name: "Remove dependencies that are no longer required" //play2
  become: true
  apt:
    autoremove: yes
```

To test use : sudo ansible-playbook main.yml and browser => local host shows html file

Imp factors :
* public ip addresses of ec2 instances changes after relaunching the instance- there inventory file needs to be managed.
* ec2 instance 's security grp's inbound rule to allow tcp:3000 traffic from all public ips(0.0.0.0/0) for a webserver created on port 3000 created using below code,

```
Create a simple web server file in NodeJS - `roles/setup/files/index.js`  containing the following content.

```javascript
var http = require("http");
var server = http.createServer(function (req, res) {
res.writeHead(200);
res.end("Hello world!");
});
server.listen(3000);
```

& tested using, http://<publicIP>:3000
* accessibility issue in ssh can also be resolved by creating a ansible.cfg file in the top level directory, with the following content:

```
[defaults]

host_key_checking = false
```
```

Exercise : configure ec2 instance with pm2 to run a node server

Step 1 : Ansible work

Directory structure

```
├── main4.yml      # Playbook file
```

```
├── inventory.txt
└── roles
    └── setup
        ├── files
        │   └── index.js
        └── tasks
            └── main.yml
```

Main.yml :

```
 # Exercise for setting up a web server in an EC

- name: Exercise for setting up a web server in an EC

  # hosts: ubuntu

  # Use the public IP if the VM is already created

  hosts: all

  user: ubuntu

  become: true

  become_method: sudo

  become_user: root

  roles:

    - setup
```

roles/setup/files/index.js :

```
var http = require("http");

var server = http.createServer(function (req, res) {

res.writeHead(200);

res.end("Hello world!");

});

server.listen(3000);
```

roles/setup/tasks/main.yml :

```
---

- name: "update apt packages."

  become: yes

  apt:

    update_cache: yes


- name: "upgrade packages"

  become: yes
```

```
  apt:
    upgrade: yes


- name: remove dependencies that are no longer required
  become: yes
  apt:
    autoremove: yes

- name: "install dependencies."
  become: yes
  apt:
    name: ["nodejs", "npm"]
    state: latest
    update_cache: yes

- name: "install pm2"
  become: yes
  npm:
    name: pm2
    global: yes
    production: yes
    state: present

- name: Creates directory
  file:
    path: ~/web
    state: directory

- name: Copy index test page
  template:
    src: "files/index.js"
    dest: "~/web/index.js"

- name: Executing node
  shell: |
    pm2 start ~/web/index.js –f
```

Step2 : aws console ec2 instance – Ubuntu ami , tags, security grp with tcp:3000 open from 0.0.0.0/0, key-pair
Step3 : inventory file ->
[all]
<host name from step2>
Step4: Run ansible playbook ansible-playbook main.yml –I inventory –private-key ~/<key-pair from step2>.pem
Step5: < host name from step2>:3000 from browser

IN BLUE – green deployment, new infrastructure created for each CI/CD run
AFTER CODE RELATED JOBS IN CIRCLE ci are completed,
Infra CREATION JOBS ::: spin new aws resources , which will be later configures, deployed to and tested and promoted to production , basics steps will be :
* aws credentials configured in circle ci project env variables
* docker image with aws cli
* filter for master branch
* code checkout
* cloud formation script execution step

Exercise : circle ci job to create infra – ec2 instance and the associated Security group.
Steps: 1 pre-requite : aws ec2 key pair, github repo for cloudformation script, circle ci account
2. Create circle ci proj with env vars for AWS_ACCESS_KEY_ID, AWS_DEFAULT_REGION, AWS_SECRET_ACCESS_KEY extracted from AWS IAM user.
3. Cloud formation script in git repo

```
AWSTemplateFormatVersion: 2010-09-09
Description: ND9991 C3 L4
Resources:
  EC2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      SecurityGroups:
        - !Ref InstanceSecurityGroup
      # Change this, as applicable to you
      KeyName: udacity
      # Change this, as applicable to you
      # You may need to find out what instance types are available in your region - use https://cloud-images.ubuntu.com/locator/ec2/
      ImageId: 'ami-09e67e426f25ce0d7'
      InstanceType: t3.micro
  InstanceSecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: Enable SSH access via port 22
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: '22'
          ToPort: '22'
          CidrIp: 0.0.0.0/0
```

5. ./circleci/config.yml

```
jobs:
  create_infrastructure:
    docker:
      - image: amazon/aws-cli
    steps:
      - checkout
      - run:
          name: Ensure backend infrastructure exist
          command: |
            aws cloudformation deploy \
              --template-file template.yml \
              --stack-name my-stack
workflows:
  my_workflow:
    jobs:
      - create_infrastructure
```

==----------

After infra deployment from circle ci, next is configuration and deployment of infra using ansible playbook

```
version: 2.1

jobs:
  configure_infrastructure:
    docker:
      - image: python:3.7-alpine3.11
    steps:
      - checkout
      - add_ssh_keys:
          fingerprints: ["06:7c:db:71:49:2f:03:36:60:08:d2:fd:33:
      - run:
          name: Install dependencies
          command: |
            apk add --update ansible # install the dependencies n
      - run:
          name: Configure server
          command: |
            ansible-playbook -i inventory.txt playbook.yml

workflows:
```

*alpine python docker image for ansible

*checkout code and add ssh keys using fingerprint{ 1. project setting->ssh keys->add ssh key ufsing hostname and private key from aws key pair .pem file; 2. Take fingerprint and add to yml}

*install ansible in agent running job and run playbook, public IP EC2 instance added to inventory file

```
└── template.yml        # Change the KeyName and ImageID property value
├── ansible.cfg           # Disables host_key_checking
├── inventory           # Or it could be inventory.txt
├── main.yml            # Playbook file from the Exercise: Remote Control Using Ansible
└── roles
|   └── setup
|       ├── files
|       |   └── index.js
|       └── tasks
|           └── main.yml
└── .circleci
    └── config.yml      # Should have the create_infrastructure Job
```

Gihub repo directory structure, to be committed -> runs circle ci build

Verify using `ssh -i "udacity.pem" ubuntu@ec2-35-170-198-76.compute-1.amazonaws.com`

-------SMOKE TESTING

Eg. Website is responding to request or not

eg., `if curl -s --head "https://google.com" // to see if google is down`

```
then
  echo "It worked!"
else
  echo "It failed"
fi
```

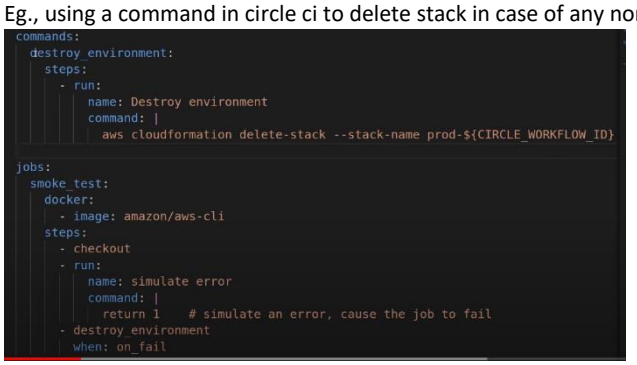
- Requires a light weight image like alpine:latest

From Circle ci – job :

```
smoke_test:
    docker:
      - image: alpine:latest
    steps:
      - run: apk add --update curl
      - run:
          name: smoke test.
          command: |
            URL="https://blog.udacity.com/"
            if curl -s --head ${URL} # test if blog.udacity.com exists
            then
              return 0
            else
              return 1
              fi
```

Rollback – incase smoke tests fail

Eg., using a command in circle ci to delete stack in case of any non 0 return from any job



```
commands:
  destroy_environment:
    steps:
      - run:
          name: Destroy environment
          command: |
            aws cloudformation delete-stack --stack-name prod-${CIRCLE_WORKFLOW_ID}

jobs:
  smoke_test:
    docker:
      - image: amazon/aws-cli
    steps:
      - checkout
      - run:
          name: simulate error
          command: |
            return 1    # simulate an error, cause the job to fail
      - destroy_environment
        when: on_fail
```

destroy-environment command executed on-fail trigger from smoke test job

---

Upto this stage, our green version is up and running with smoke test passed, time to make router switch:: PROMOTE TO PRODUCTION ::

Circle workflow id environment variable used to identify blue or green version of aws stack in create-infra jobs :

```
aws cloudformation deploy
--template-file cloudfront.yml
--stack-name cloudfront
--parameter-overrides PipelineID="${CIRCLE_WORKFLOW_ID}"
```

----------Promote to Production: ----------

Exercise : 1. Create an S3 bucket (Manually) -> upload a sample index.html and Enable the Static website hosting; 2. Create a Cloudformation Stack (Manually) to deploy Cloudfront distribution CDN from s3 bucket -> cloudfront.yml

```
Parameters:

  # Existing Bucket name
  PipelineID:
    Description: Existing Bucket name
    Type: String
Resources:
      CloudFrontOriginAccessIdentity:
        Type: "AWS::CloudFront::CloudFrontOriginAccessIdentity"
        Properties:
          CloudFrontOriginAccessIdentityConfig:
            Comment: Origin Access Identity for Serverless Static
Website
      WebpageCDN:
        Type: AWS::CloudFront::Distribution
        Properties:
          DistributionConfig:
            Origins:
              - DomainName: !Sub "${PipelineID}.s3.amazonaws.com"
```

```
                Id: webpage
                S3OriginConfig:
                  OriginAccessIdentity: !Sub "origin-access-
identity/cloudfront/${CloudFrontOriginAccessIdentity}"
            Enabled: True
            DefaultRootObject: index.html
            DefaultCacheBehavior:
              ForwardedValues:
                QueryString: False
              TargetOriginId: webpage
              ViewerProtocolPolicy: allow-all
    Outputs:
      PipelineID:
        Value: !Sub ${PipelineID}
        Export:
          Name: PipelineID
```

```
aws cloudformation deploy \
 --template-file cloudfront.yml \
 --stack-name production-distro \
 --parameter-overrides PipelineID="${S3_BUCKET_NAME}" \ # Name of
the S3 bucket you created manually.
 --tags project=udapeople &
```

-----BLUE VERSION READY UPTO HERE--

3. Cloudformation template to create a new S3 bucket -> bucket.yml

```
Parameters:
 # New Bucket name
 MyBucketName:
   Description: Existing Bucket name
   Type: String
Resources:
 WebsiteBucket:
   Type: AWS::S3::Bucket
   Properties:
     BucketName: !Sub "${MyBucketName}"
     AccessControl: PublicRead
     WebsiteConfiguration:
       IndexDocument: index.html
       ErrorDocument: error.html
 WebsiteBucketPolicy:
   Type: AWS::S3::BucketPolicy
   Properties:
     Bucket: !Ref 'WebsiteBucket'
     PolicyDocument:
       Statement:
       - Sid: PublicReadForGetBucketObjects
```

```
        Effect: Allow
        Principal: '*'
        Action: s3:GetObject
        Resource: !Join ['', ['arn:aws:s3:::', !Ref
'WebsiteBucket', /*]]
```

4. Update the CircleCI Config file to Write a job named create_and_deploy_front_end that executes the bucket.yml template to: Create a new S3 bucket and Copy the contents of the current repo (production files) to the new bucket. //CDN cache changed after this job runs success

```
# Executes the bucket.yml - Deploy an S3 bucket, and interface with
that bucket to synchronize the files between local and the bucket.
 # Note that the `--parameter-overrides` let you specify a value
that override parameter value in the bucket.yml template file.
 create_and_deploy_front_end:
   docker:
     - image: amazon/aws-cli
   steps:
     - checkout
     - run:
         name: Execute bucket.yml - Create Cloudformation Stack
         command: |
           aws cloudformation deploy \
           --template-file bucket.yml \
           --stack-name stack-create-bucket-
${CIRCLE_WORKFLOW_ID:0:7} \
           --parameter-overrides MyBucketName="mybucket-
${CIRCLE_WORKFLOW_ID:0:7}"
```
// passing CIRCLE_WORKFLOW_ID to form the name of our new bucket
```
     # Uncomment the step below if yoou wish to upload all contents
of the current directory to the S3 bucket
      - run: aws s3 sync . s3://mybucket-${CIRCLE_WORKFLOW_ID:0:7} –
delete
```

5. A)CircleCI job named get_last_deployment_id that performs the query and saves the last successful production release id to a file that we can persist to the workspace - saving the bucket ID to a file and persist the file to the workspace

```
# Fetch and save the pipeline ID (bucket ID) responsible for the
last release.
```

```
 get_last_deployment_id:
   docker:
     - image: amazon/aws-cli
   steps:
     - checkout
     - run: yum install -y tar gzip
     - run:
```

```
        name: Fetch and save the old pipeline ID (bucket name)
responsible for the last release.
        command: |
          aws cloudformation \
          list-exports --query
"Exports[?Name==\`PipelineID\`].Value" \
          --no-paginate --output text > ~/textfile.txt
    - persist_to_workspace:
        root: ~/
        paths:
          - textfile.txt
```

B) job named promote_to_production that executes our cloudfront.yml(used in the manual steps)

```
# Executes the cloudfront.yml template that will modify the existing
CloudFront Distribution, change its target from the old bucket to
the new bucket - `mybucket-${CIRCLE_WORKFLOW_ID:0:7}`.
```

//this is the router switch

```
# Notice here we use the stack name `production-distro` which is the
same name we used while deploying to the S3 bucket manually.
 promote_to_production:
   docker:
     - image: amazon/aws-cli
   steps:
     - checkout
     - run:
         name: Execute cloudfront.yml
         command: |
           aws cloudformation deploy \
           --template-file cloudfront.yml \
           --stack-name production-distro \
           --parameter-overrides PipelineID="mybucket-
${CIRCLE_WORKFLOW_ID:0:7}"
```

------GREEN VERSION UP & router switchhere------

C) Job Named clean_up_old_front_end that uses the pipeline ID to destroy the previous production version's S3 bucket and CloudFormation stack using pipeline id from get_last_deployment_id

```
# Destroy the previous production version's S3 bucket and
CloudFormation stack.
 clean_up_old_front_end:
   docker:
     - image: amazon/aws-cli
   steps:
     - checkout
     - run: yum install -y tar gzip
     - attach_workspace:
         at: ~/
```

```
      - run:
          name: Destroy the previous S3 bucket and CloudFormation
stack.
          # Use $OldBucketID environment variable or
mybucket644752792305 below.
          # Similarly, you can create and use $OldStackID environment
variable in place of production-distro
          command: |
            export OldBucketID=$(cat ~/textfile.txt)
            aws s3 rm "s3://${OldBucketID}" –recursive
```

6. Workflow

```
workflows:
      my_workflow:
        jobs:
          - create_and_deploy_front_end
          - promote_to_production:
              requires:
                - create_and_deploy_front_end
            - get_last_deployment_id
            - clean_up_old_front_end:
              requires:
                - get_last_deployment_id
                - promote_to_production
```

7. Commit to github and circle pipeline will run and cdn reflected in max 30 mins

_____MONITORING AFTER DEPLOYMENT _____

#Network logs
#App logs
#CPU/memory/disk
#Availability/Performance/Capacity/Productivity

Some common areas where you might need monitoring are:

- Database Stats
- Application Logs
- Docker Logs
- Firewall
- Load Balancer
- GPU
- Operating System
- Router

- Cable Modem
- Message Bus
- AWS Services
- Mail Server
- Other 3rd Party Services
- Other Monitoring Systems

Proactive Monitoring - Forecast Infrastructure Costs / Track Bugs to Their True Source / Predict Seasonal Spikes and Trends

Pull monitoring system – pulls data automatically from data source; push system- listens to data source

Components of monitoring system – 1 **Time-Series Data :** data in a series of time intervals; 2 **Data Aggregator:** collects and groups data by type or data source ; 3 **Data Visualizer:** takes data from aggregator and produces useful charts and graphs

Monitoring tools available -

Data Aggregator	Details	Visualizers
Graphite	Mature, Open-Source, Installable	*Grafana*
Loggly	Managed, Cloud-Based, Powerful tooling	*Built-In*
Datadog	Managed, Cloud-Based, Built-in AI/ML	*Built-In*
Prometheus	Open Source, Lightweight, Self-Contained, Installable	*Grafana*
Logstash	Open Source, Cloud-Based or Installable	*Kibana*
CloudWatch	Built-In to AWS	*Built-In*
ElasticSearch	RPA CoE	*Kibana*

----Prometheus----

Ec2 ubuntu instance, port 9090 open to the public, key pair and ssh to machine
ssh -i <.pem file> ubuntu@<EC2dnshostname> ; download and extract the Prometheus server files using wget < linux prometheus download url > command ; **Prometheus.yml** configuration file
Start Prometheus- ./Prometheus -- config.file= Prometheus.yml ➔ browser <EC2dnshostname >:9090

Exporters- lightweight agents installed on data source from where Prometheus "scrapes" data every few seconds – prebuilt for github/slack/jira/aws/node{for cpu/disk/memory} /mongoDB /blackbox {http based application} etc.

Prometheus Node Exporter on AWS EC2 - https://codewizardly.com/prometheus-on-aws-ec2-part2/ - new ec2 ubuntu with port 9090 open to the public, key pair and ssh to machine ➔ wget <linux node exporter download url> ➔ extract and ./node-exporter

ssh to Prometheus server and edit Prometheus.yml to add another scrape config

```
- job_name: 'node_exporter'
```

```
    static_configs:
      - targets: ['ec2-13-58-127-241.us-east-
2.compute.amazonaws.com:9100']
```

./Prometheus.yml - restart

Prometheus Service Discovery on AWS EC2 - new instance don't need to be added manually

```
scrape_configs:
```

```
  - job_name: 'node'
    ec2_sd_configs:
      - region: us-east-1
        access_key: PUT_THE_ACCESS_KEY_HERE
        secret_key: PUT_THE_SECRET_KEY_HERE
        port: 9100
```

Prometheus visualization:

Graph tab ➔ insert metric at cursor {lists from exporters eg. node memory} ➔ graph/console result

Prometheus ALERTS:

Email/chat tool/ desktop notfn / phone calls

0. Port 9093 should be open on Prometheus instance➔ ssh to server and create rules.yml
eg., if free server memory gets below 1mil for a whole 1 min
1. Rules.yml

```
groups:
  - name: AllInstances
    rules:
    - alert: UsingTooMuchMemory
      # Condition for alerting
      expr: node_memory_MemFree_bytes < 1000000
      for: 1m
    # Annotation - additional informational labels to store more information
      annotations:
        title: 'Instance {{ $labels.instance }} is almost out of memory'
        description: '{{ $labels.instance }} of job {{ $labels.job }} has been down f>
    # Labels - additional labels to be attached to the alert
      labels:
        severity: 'critical'
```

2. Prometheus.yml

```
  GNU nano 4.8                    prometheus.yml                          Modified
# my global config
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is >
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 >
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
  - static_configs:
    - targets:
      # - alertmanager:9093

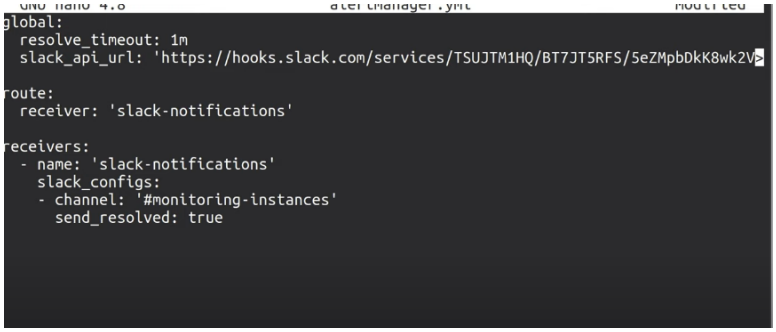
# Load rules once and periodically evaluate them according to the global 'evaluation_>
rule_files:
  - "rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
```

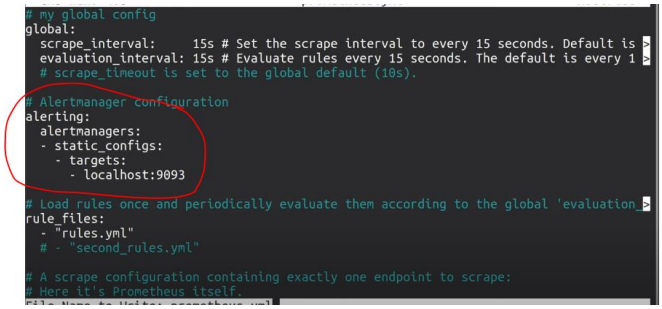
3. Setup alert manager setup – runs on port 9093

wget <linux Prometheus alert manager URL> -> untar -> ./alert-manager -config=alertManager.yml

alertmanager.yml: // configuration made for slack notification, different receiver can be setup for email



4. Prometheus.yml configuration :



and restart Prometheus using ./

5. browser ➔ <Prometheus instance public dns>:9093

PROJECT 3

1. Create a proposal in document or presentation form that "sells" the concept of CI/CD to non-technical decision-makers in the UdaPeople organization.

benefits of CI/CD to achieve, build, and deploy automation for cloud-based software products.

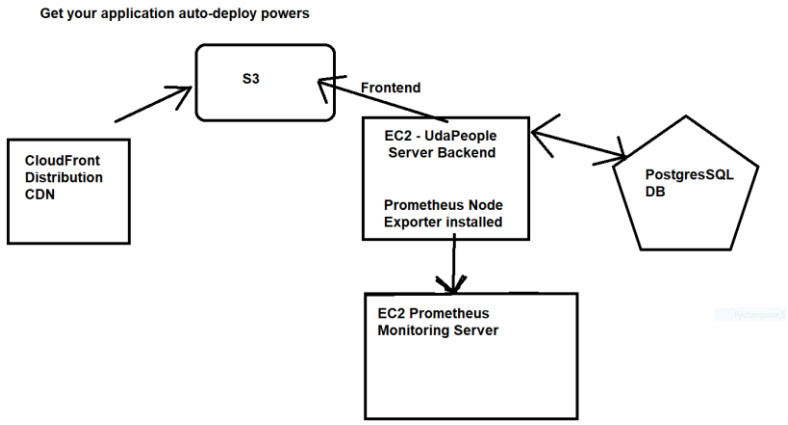
Utilize Deployment Strategies to design and build CI/CD pipelines that support Continuous Delivery processes.

Utilize a configuration management tool to accomplish deployment to cloud-based servers.

Technical language to the business's values

focus on benefits that create revenue, protect revenue, control costs, or reduce costs.

5 slides ppt ==> presentation.pdf



https://github.com/AlbusDumble-dore/cdond-c3-projectstarter

2.Code in github + urls + screenshots

i>Fork github repo - https://github.com/udacity/cdond-c3-projectstarter/tree/master

ii>Set up circleCi project from github repo

iii>Build frontend and backend circleCi job-> npm install & npm run build

iV>Unit test front end & backend circleCi jobs-> npm run test

V>Scan Backend and frontend circleci job->npm audit

Vi>Udacity.pem key pair for EC2 access in AWS manually

Vii>Udapeople IAM user with Admin acces , Access key - AWS manually

Viii>PostgreSQL RDS DB , publically accessibility = True in AWS manually

ix>Public S3 bucket in AWS manually - Blue deployment

X>CloudFront distrubution from Cli and cloud-formation stack(using step vii configuration) using cloudfront.yml - Blue deployment

xi>CircleCI console using SSH and env variables

xii>circleCi job Deploy-Infra:

a>call cloudformatoin backend.yml to create ec2 sec grp and ec2 instance

b>call cloudformation frontend.yml to create new s3 bucket

c>backend ec2 public dns to inventory file

d>destroy env command call

I>Destroy s3

II>Destroy backend.yml stack

IIi>Destroy frontend.yml stack

xiii>CircleCI job Configure Infra

a>checkout, add ssh keys, install aws cli dependencies

b> call ansible playbook configure-server.yml with env variables

I> play 1- configure-server play - npm install nodejs, pm2 --.circleci/ansible/roles/configure-server/tasks

II> play 2- node_exporter play - install node exporter - circleci/ansible/roles/configure-prometheus-node-exporter/tasks/main.yml

c>Destroy env

xiv>CircleCi job - running migration step of deploy application - npm run migration & send to kvdb

xv>circleCi job - deploy application frontend on ec2 backend url - npm run build- copy to new S3 - green deployment

xvi>circleCi job - deploy application backend - install dep,ssh, run deploy-backend.yml playbook which uses deploy play - .circleci/ansible/roles/deploy/tasks - green deployment

xvii>Revert migration (on-fail) command - npm run migration:revert

xviii>Smoke test circleCi job

a>Frontend - s3 public url endpoint grep <keyword>

b>Backend - EC2 public dns:3030/api/status

xix>CloudFront update CircleCi job- aws cloudformation deploy stack using cloudfront.yml to point to new S3

xx> cleanup CircleCi job-

a>get old workflowID= aws cloudformation list-exports

b>get STACKS=awscloudformation list-stacks

c>if(STACK=~oldWorkflowID)

aws s3 rm s:{old WorkflowID}

aws cloudformation delete-stack backend {old WorkflowID}

aws cloudformation delete-stack frontend {old WorkflowID}

xxi> Deploy prometheus ec2 instance with ec2 auto discovery and alert manager

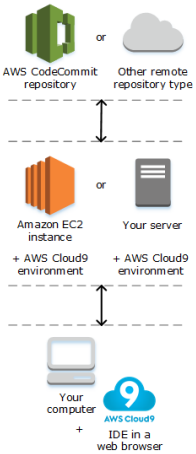
----------Course 4 – Microservice at scale using AWS and Kubernetes – AWS lambda Function as a service ; cloud9{IDE }; http request response ----------

_____Deploying event Driven Microservice _____

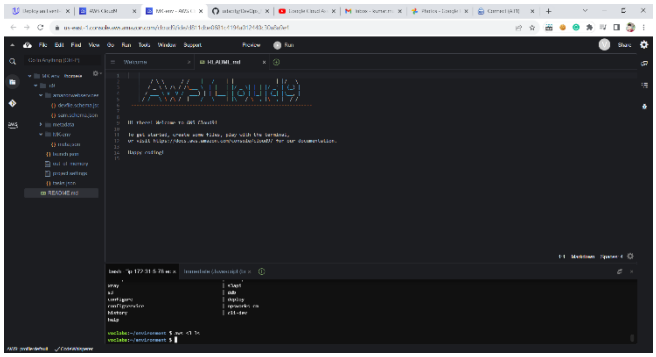
AWS Lambda – serverless compute service to run code without provisioning or managing servers; in response to an event ; scalable and pay as you use; leverage cloud native services {design applications in cloud[public/private/hybrid] to leverage autoscaling ,infinite storage and compute, advanced APIs – Microservice oriented, CD and DevOps supported, fault tolerant }
allows integration with other core services in AWS through triggers
Example AWS Step Functions - to create more sophisticated workflows like polling for a job to finish and performing an action
Example2 - AWS DeepLens -a computer vision device to serve out predictions

AWS SQS – simple queuing service – hosted queue to integrate and decouple distributed software systems

Cloud9 setup



1. Setup cloud9 Environment – name, description, type {direct access- create new ec2 / systems manager – create new non ingress ec2 /ssh connection – create and run in remote server }, instance type{t2/t3..}, platform {amazon linux 1 / 2/ ubuntu }, hibernate setting {cost saving}



feature of cloud 9 – collaboration – invite other users, debugger, terminal, file explorer , source control, aws resources

2.a. Fork github repo to your Github account

2.b Generate SSH key pair (public and private) in cloud9 terminal and save public key to Github repo for *Cloud9 env to be able to push changes to the Github repo*

```
ssh-keygen -t rsa
cat /home/ec2-user/.ssh/id_rsa.pub
```

Save the public key, content of /home/ec2-user/.ssh/id_rsa.pub file, in the Github >> Settings >> SSH & GPG keys.

3. Git clone repo in Cloud9 terminal

4. Increase cloud9 memory

```
df -h
```

cd DevOps_Microservices/Supporting-material
./resize.sh

5. Initialise new lambda fx – sam init cmd from cloud9 terminal
template source – AWS quick start template -yes or custom template location
package type – zip {artifact is zip uploaded in s3} or Image{artifact is image uploaded ECR [elastic container registry]image repository - yes}

Base image – amazon/python3.8- base
Project name - [Application-name]
AWS quick start application template – Hello World Lambda image-yes/TensorflowML API/scikit ML interfact API…

6. Sam build command to build hell world application {Check the [Application-name]/README file instructions }

cd [Application-name]

sam build
O/p - Dockerize the Lambda function in the Cloud9 environment - *Application-name]/hello world/app.py – shows output of lambda function*

7. Run the application locally (in Cloud9)

sam **local** invoke – output of app.py fx

8. **Deploy the application to an ECR image repository**
Create a cloudformation stack to deploy the application image in the ECR image repository

sam deploy –guided
params- stack name, aws region, other permissions prompt
The sam deploy --guided command will create a Cloudformation stack comprising of an IAM role, Lambda function, and an API Gateway. This command will push the the Docker image from local (Cloud9) to the ECR service.

o/p - API gateway endpoint URL in outputs of cloudformation stack

9. Test <manually>: **curl** [API gateway endpoint URL] or directly paste in browser or from AWS console Lambda service test

Note lambda_handler accepts event and context as input

10. Deployment and testing by tri gger using API g/w webservice-

Sam local start-api – gives api gw endpint which you can curl from another cloud9 terminal

Other example is every time a new file is placed in S3 or CloudWatch event timer [scheduled].
Use case – Lambda fx that scrapes competitor website for similar product that they are selling , running every night using CloudWatch event and put data to S3. Another lambda fx triggered when s3 received data, extarcts pricing info from HTML and writes to DynamoDB if lower than current value in DB, Third lambda fx that uses API gw to serve out companies current pricing which should be always lower than competition.

Logs can be seen in cloudWatch logs or monitor section of lambda in AWS console

11. Creating a virtual environment - Python env to isolate python interpreter to a specific directory for conditions where multiple applications are running and require different and conflicting version of dependencies.
python3 -m venv ~/.<env_name> - to create env in users home directory
source ~/.<env_name>/bin/activate – to activate

12. Making change in lambda fx code –

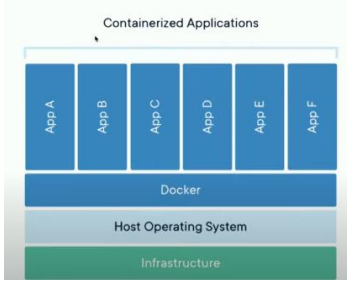
Modify the code in app.py and other files requirement.txt to include dependencies required by new application-> create new virtual env : python3 -m venv ~/.mynewenv ; source ~/. mynewenv /bin/activate ; python3 -m pip install -r requirements.txt -> create payload.json file if required -> Build, test and run application locally – sam build ; sam local invoke -e payload.json ; sam local start api- > deploy app – sam deploy –guided -> test lambda fx on aws console

Event handling - change trigger on aws lambda console and select s3 – bucket name and event name [all object create event] – upload a picture in s3 – it invokes lambda fx to identify tags using object recognition and writes to some other location // other triggers are timer using cloudWatch or http request on api gtw
If an event has body, to retrieve and save json body in lambda fx code – body = json.loads(event[“body”])

Another example – Wikipedia lambda fx to summarise passed topic content – sam init : aws quick start template – hello world , package type image amazon/python3.8-base – paste the code in hello_world/app.py – add Wikipedia in requirements.txt – create a virtual environment & activate – Build fx : sam build - test locally : crate payload.json , sam local invoke -e payload.json - Deploy : sam deploy –guided - test using lambda console on aws console

_____DOCKER_____

Docker container- single purpose VM, package dependencies ,code and OS runtime in a bundle, take less space and time , multiple container can run on a machine, built on IaC as defined as file and checkedin along side code



Example- developer sharing local proj – dev 1 works on a flask application with os and configuration handled by docker container file, another dev2 checkout hre code and run docker run command to run the application

2 components of Docker-

Docker desktop – local development workflow: container runtime {allows container to execute}, dev tool, socker app, Kubernetes{managing containerised applications}; docker desktop works with docker containers to allow local use and development
DockerHub – Collaboration workflow: team& org, pull and use certified imgs, automated build of container images via github, private and public repos ; Docker hub allows developer to share docker containers to use as base image for building new solutions

Steps:
1. Setup virtual env in cloud9– python3 -m venv venv ; source venv/bin/activate ; pip install –upgrade pip; install pylint, black code formatter, pytest testing lib, ipython interactive interpreter -> pip install pylint/black/pytest/ipython

2. Makefile setup – sections : setup - python3 -m venv venv ; install: pip install -r requirements.txt{requirements.txt has all pylint,black,pytest etc. } ; test: python -m pytest -vv –cov=myrepolib tests/*.py , python -m pytest –nbval notebook.ipynb ; lint – pylint –diable=R,C myrepolib cli web ; all- install lint test
Run make setup / install/ test / lint /all

3. Check docker desktop is installed – terminal : docker –version

4. linting: Docker file linter is hadolint to check bugs in dockerfile :hadolint <dockerfile>

.circleci/config.yml:

```
#
version: 2
jobs:
  build:
    docker:
    # Use the same Docker base as the project
      - image: python:3.7.3-stretch

    working_directory: ~/repo

    steps:
      - checkout

      # Download and cache dependencies
      - restore_cache:
          keys:
            - v1-dependencies-{{ checksum "requirements.txt" }}
            # fallback to using the latest cache if no exact match is found
            - v1-dependencies-

      - run:
          name: install dependencies
          command: |
            python3 -m venv venv
            . venv/bin/activate
            make install
            # Install hadolint
            wget -O /bin/hadolint https://github.com/hadolint/hadolint/releases/download/v1.16.3/hadolint-Linux-x86_64 &&\
                chmod +x /bin/hadolint

      - save_cache:
          paths:
            - ./venv
          key: v1-dependencies-{{ checksum "requirements.txt" }}

      # run lint!
      - run:
          name: run lint
          command: |
            . venv/bin/activate
            make lint
```

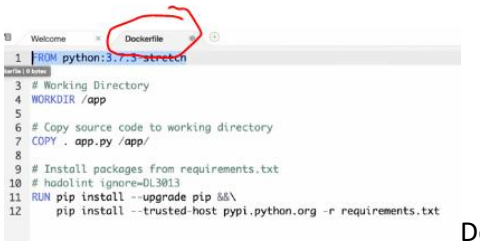
Use makefile with CircleCI step :
```
validate-circleci:
	circleci config process .circleci/config.yml

Run-circleci-local:
	circleci local execute

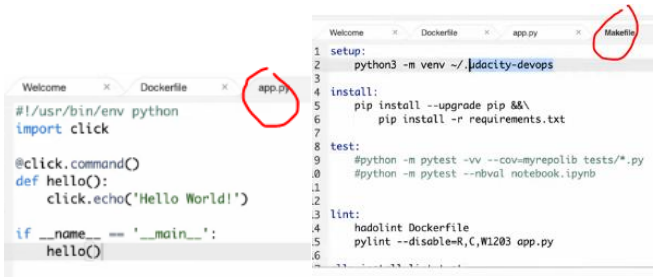
lint:
	hadolint demos/flask-sklearn-student-starter/Dockerfile
	pylint --disable=R,C,W1203 demos/**/**.py
```

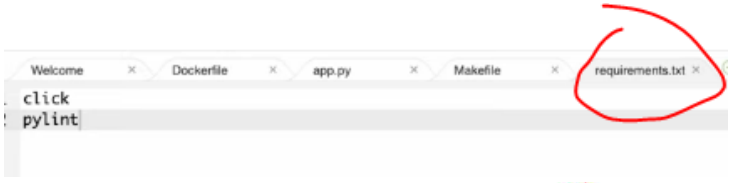
execute make run-circleci-local command

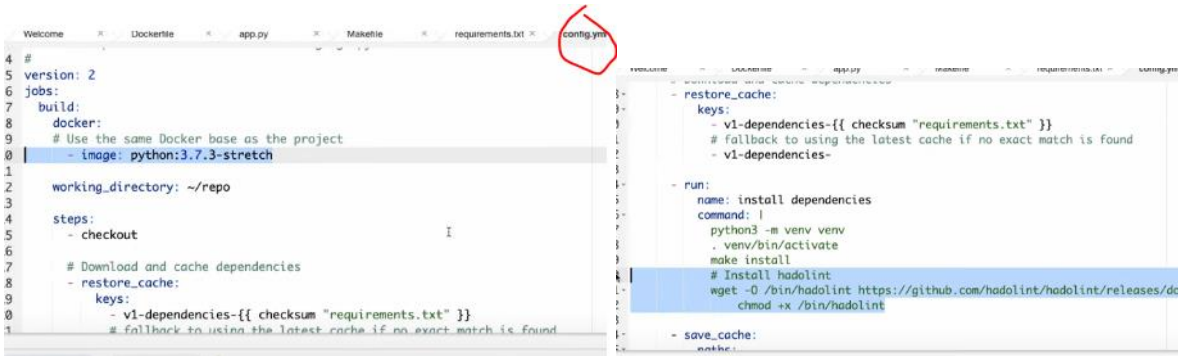
All steps end2end for cloud9 env – create cloud9 env on aws console -> new github repo-> ssh key-gen from cloud9 and add in github repo-> git clone from cloud 9 terminal- >cd directory and touch Makefile,requirements.txt,Dockerfile,app.py-> chmod +x app.py -> paste code for Dockerfile , app.py , Makefile, requirements.txt - > create virtual env-> make install -> docker build –tag=app command to build an image called app-> docker tun -it app bash command -> setup circleCi job from circleCi console-> git push



Docker File - From base image , setup working dir , copy app to working dir , upgrade pip and install requirements - docker image ls to see all of your created Docker images







Deploying [environment and microservice] to Amazon ECR : ecr console create repo -> authenticate from cloud9 console: $(aws ecr get-login –no-include-email –region us-east-1) -> Build container

locally : docker build -t <img_name> . -> tag image : docker tag <image_name>:latest <repo_url>:<image_name>:latest -> docker push <repo_url>:<image_name>:latest

---------Containerisation of existing applications----------

Steps:
0. Cloud9 env from aws console , ssh key exchange to github
1. Go through code in github repo for python{flask} application
2. Install packages – Dockerfile: RUN pip install –upgrade pip &&\
pip install –trusted-host pypi.python.org -r requirements.txt

3. Virtual env creation : python3 -m ~/.<env_name> ; source ~/.< env _name>/bin/activate → this is done as specific applications require specific version of dependencies and virtual env ensures that dep are installed inside the scope of virtual environment.

4. optional: create alias to shortcut to virtual env. Edit ~/.bashrc file and append
alias <alias_name> =”cd /home/ec2-user/environment/< env _name>”

This lets u cd into the github repo by just typing <alias_name>

5. Makefile scaffolding:

```
install:
    pip install --upgrade pip &&\
        pip install -r requirements.txt

test:
    #python -m pytest -vv --cov=myrepolib tests/*.py
    #python -m pytest --nbval notebook.ipynb


lint:
    pylint --disable=R,C hello.py

all: install lint test
```

6. Requirement file creation – requirements.txt
7. Make install
8. Make lint
9. Git push all- makefile & requirement files

10. Make .circleci/config.yml {make install and make lint steps}and Circleci console setup – commit and push executes the circleci pipeline
11. Copying an application to docker -> dockerfile :

COPY . flask_app/web.py /app/

COPY . nlib /app

12. App setup - exposed port 80 to host computer- for flask app to listen to http requests -> dockerfile: EXPOSE 80

13. App start- dockerfile: CMD [“python”,”web.py”]
//other docker commands – docker volume -> creates a volume and mount to container ; configure logging -> `> /tmp docker run -it --log-driver json-file --log-opt mode=non-blocking ubuntu`

`root@551f89012f30:/#` ; map port to ext host ; configuring memory cpu and gpu

14. Docker image built locally - `docker build --tag=api`
Docker run container `docker run -p 8000:5001 api`
5001 was mentioned in web.py for application port which is exposed on port 800 on host computer

Access the container by http://localhost:8000 in browser

----------Kubernetes----------

Installed with Docker desktop, kubectl cmd line tool installed with that, already present in cloud9 env

- Open source orchestration system for working with containerised applications
- Amazon eks, google gke, azure aks
- HA architecture, auto scaling , service discovery , health and config mgmt

Core operations – creating Kubernetes cluster, deploying application into cluster, exposing application ports, scaling application and updating application

Cluster include cluster master and cluster nodes, Kubernetes node can contain multiple pods which contain multiple containers/volumes

Kubernetes cluster can be setup using local – Docker desktop or cloud cluster – eks/gke/aks

Kubernetes.io tutorial – creating cluster : minikube version check -> minikube start [launch Kubernetes cluster to run containers against] -> kubectl version -> kubectl cluster-info [gives master ip addr and master dns] -> kubectl get nodes

Launch Containers in a Kubernetes Cluster :
`docker stack deploy --compose-file`

Yaml file-

```
version: '3.3'

services:
  web:
    image: dockersamples/k8s-wordsmith-web
    ports:
     - "80:80"

  words:
    image: dockersamples/k8s-wordsmith-api
    deploy:
      replicas: 5
      endpoint_mode: dnsrr
      resources:
        limits:
          memory: 50M
        reservations:
          memory: 50M

  db:
    image: dockersamples/k8s-wordsmith-db
```

This could be deployed with the following command:

```
docker stack deploy --namespace my-app --compose-file /path/to/docker-compose.yml mystack
```

Creating EKS cluster from Web console :

Pre-requisite: Default VPC & subnet {/16 ip4 cidr block providing 65536 addrs},auto created default rote gw and internet gw , IAM role for cluster {aws service as trusted entity , service EKS-cluster, policy -AmazonEKSClusterPolicy}, IAM role for worker nodes {for giving permissions to kubelet running on worker node , sercice EC2 , policies – AmazonEKSWorkerNodePolicy , AmazonEC2ContianerRegistryReadOnly, AamazonEKS_CNI_Policy} , SSH key

EKS cluster has 2 compo – Control plane-EKS cluster{nodes running Kubernetes sw like etcd and Kubernetes api server} and Data plane{worker nodes[VM] for running pods }

Create EKS cluster – name, default Kubernetes version, attach iam role created for cluster, attach VPC, subnets, security group , Cluster endpoint access – public

Create Node group – from above cluster properties-> compute -> add node group , name , attach iam role with worker node , node grp compute and scaling config – ami type amazon linux2 , on demand capacity type , instance type t3.micro , 20gb disk size , min/max/desired nodes , node grp ntw config – subnets, allow remote access to nodes , ssh key pair , allow access from all

Eksctl – tool to create aws eks service from aws cli , uses cloudformation internally ; auto generating resources in default region – VPC,subnets ,node groups , decide ami and type of worker node , Kubernetes api endpoint
steps : install eksctl -`curl --silent --location "https://github.com/weaveworks/eksctl/releases/download/latest_release/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp` , create basic cluster : eksctl create cluster {autogenerated name, 2 m5.large worker nodes, linux ami, default region , advanced config : eksctl create cluster --name myCluster --nodes=4 or eksctl create cluster --config-file=<path>}, list cluster : eksctl get cluster [--name=<name>][--region=<region>] , delete cluster : eksctl delete cluster --name=<name> [--region=<region>]
Two separate CloudFormation stacks - eksctl-eksctl-demo-cluster for the cluster, and eksctl-eksctl-demo-nodegroup-ng-7f9854c3 for the initial nodegroup will be created

Kubectl - cli tool for interacting with Kubernetes cluster and to deploy the application, kubectl version –client to check version

Fetch details of cluster : eksctl get cluster --name=eksctl-demo --region=us-east-2 [--profile <profile-name>]
Health check of cluster : kubectl get nodes [--profile <profile-name>]

Deploy App:

```
## Ensure Docker Desktop is running locally

docker --version
## Build an image using the Dockerfile in the current directory
docker build -t python-helloworld .
docker images
## Run a container
docker run -d -p 5000:5000 python-helloworld
## Check the output at http://localhost:5000/ or http://0.0.0.0:5000/ or
http://127.0.0.1:5000/
docker ps
## Now, stop the container.
## Tag locally before pushing to the Dockerhub
## We have used a sample Dockerhub profile /sudkul
## Replace sudkul/ with your Dockerhub profile
docker tag python-helloworld sudkul/python-helloworld:v1.0.0
docker images
## Log into the Dockerhub from your local terminal
docker login
## Replace sudkul/ with your Dockerhub profile
docker push sudkul/python-helloworld:v1.0.0
## Check the image in your Dockerhub online at
https://hub.docker.com/repository/docker/sudkul/python-helloworld
```

Now deploy publicly available docker image to Kubernetes cluster :

```
## Assuming the Kubernetes cluster is ready

kubectl get nodes
## Deploy an App from the Dockerhub to the Kubernetes Cluster
kubectl create deploy python-helloworld --image=sudkul/python-
helloworld:v1.0.0
## See the status
kubectl get deploy,rs,svc,pods
## Port forward
kubectl port-forward pod/python-helloworld-84857d9565-2598m --address
0.0.0.0 5000:5000
```

Access the app locally at http://127.0.0.1:5000/

Deletion: delete stack from cloudformation or Delete using the EKSCTL: eksctl delete cluster --region=us-east-2 --name=eksctl-demo [--profile <profile-name>]

Logging and monitoring can be setup by Prometheus via Prometheus operator integration with Kubernetes: wget <prom_url>, update Prometheus.yml, add rule for opening port 9090 in aws console, execute , check <prom_instance_public_dns>:9090 Another way to get application logs - kubectl logs <pod_name>

Debugging

```
# Run in Docker Hub container with kubernetes

kubectl run flaskskearlndemo\
    --generator=run-pod/v1\
    --image=$dockerpath\
    --port=80 --labels app=flaskskearlndemo

# List kubernetes pods
kubectl get pods

# Forward the container port to host
kubectl port-forward flaskskearlndemo 8000:80
```

kubectl describe pod {POD NAME}

Autoscaling with CPU or Memory

kubectl scale {deployment name} --replicas={desired number of replicas}

The Kubernetes HPA (Horizontal Pod Autoscaler) will automatically scale the number of pods (*remember they can contain multiple containers*) in a replication controller, deployment or replica set. The scaling is based on CPU utilization, memory or custom metrics defined in the Kubernetes Metrics Server.

----------Operationalising microservice----------

workflow – application stored in git -> changes in git trigger cd server to test and deploy code to new env where env is configured as IaC -> microservice could be containerised service running in Kubernetes or FaaS on Lambda -> load testing tool like locust for performance and autoscaling verification -> merge to prod
Alert and incident response – Prometheus, custom alerts, etc.; Kubernetes alerts – application layer metrics , services running on Kubernetes , Kubernetes arch ; ;

DR – backup and restore cluster; copy cluster resources to other cluster ; replicate rod for dev and test envs

Load test using locust – cloud9 env -> run the flask application -> add locust in requirement.txt -> execute locust app on port 8089 -> open <cloud9 ec2 public dns>:8089 and fire load testing from ui by inputting no of users to simulate , hatch rate and host

----PROJECT4---- operationalising a ML microservice API

Sklearn pretrained model to predict housing prices ; python flask app app.py operationalising using Kubernetes – test proj code using linting, containerise using dockerfile , deploy docker container & test, configure Kubernetes and deploy cluster, deploy kubernetes container & test , configure cicd using circleci

proj files-> github repo : https://github.com/AlbusDumble-dore/Operationalize-a-Machine-Learning-Microservice-API

Steps:
1. Git clone repo

2. Create Virtual environment -
python3 –m venv ~/.devops
source ~/.devops/bin/activate

3. Installing dep via makefile listed in requirements.txt – make install

4. Linting pylint for python and hadolint for DockerFile – make lint
https://tcoil.info/how-to-install-hadolint-on-linux/ -> sudo wget -O /bin/hadolint
https://github.com/hadolint/hadolint/releases/download/v1.16.3/hadolint-Linux-x86_64

```
sudo chmod +x /bin/hadolint
/bin/hadolint Dockerfile
```

5. Minikube installation for k8 setup
https://crishantha.medium.com/running-minikube-on-aws-ec2-e845337a956
https://minikube.sigs.k8s.io/docs/start/ ->
2 cpu or more
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

Kubectl install - https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/?source=post_page-----e845337a956----
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
chmod +x kubectl
mkdir -p ~/.local/bin
mv ./kubectl ~/.local/bin/kubectl

6. Dockerfile complete

7. run_docker.sh – package image and test locally by running docker container and make_prediction.sh
Binding: [host port]:[container port]

8. upload to dockerhub – upload_docker.sh
https://forums.docker.com/t/docker-push-error-requested-access-to-the-resource-is-denied/64468
`docker login -u "myusername" -p "mypassword" docker.io`

`docker push myusername/myimage:0.0.1`
https://hub.docker.com/r/mukulkmr/project4

9. configure k8 to run locally – minikube start

10. deploy with k8 – run_kubernetes.sh {including port fwding} and make-prediction.sh

11. circleci automation
https://s3.amazonaws.com/video.udacity-data.com/topher/2019/May/5cda0d76_config/config.yml

Git push issue https://dev.to/shafia/support-for-password-authentication-was-removed-please-use-a-personal-access-token-instead-4nbk - create personal access token in github

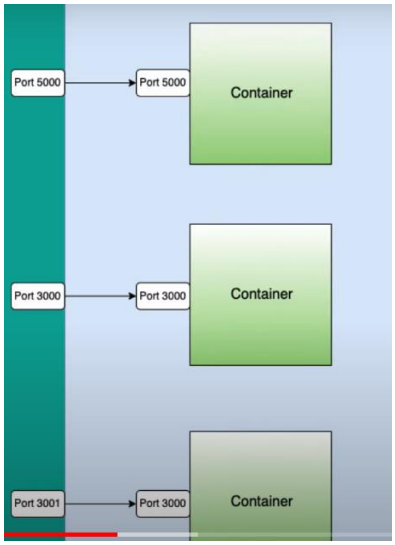
---

TECH WITH NANA

Container - package with dependencies and configuration and start script , portable – can be moved from one env to another easily , container live in container repository (public{eg. Docker hub for docker containers} or private) , run one command to fetch and run docker container to setup entire development environment – docker run <image_name>:<version> //first tries locally and then from image repository ; technical description of containers image artifact – layers of linux based image{mostly alpine} , application image{eg. postgres} and configuration ; container is running instance of image
docker ps – to see running containers – gives container id
docker ps –a –> all containers running or not
docker ps –d <image_name> -> detach mode ; gives container id
docker pull <image_name> - only fetch image not run -> followed by docker run command
docker start/stop <running container id >
docker images - > shows all images list available locally

Container post vs Host port binding

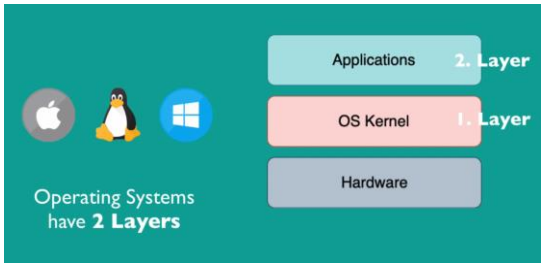
Second and third container both running on port 3000 but that's okay as the binded host port is different. Host port is port of virtual machine on which docker is running

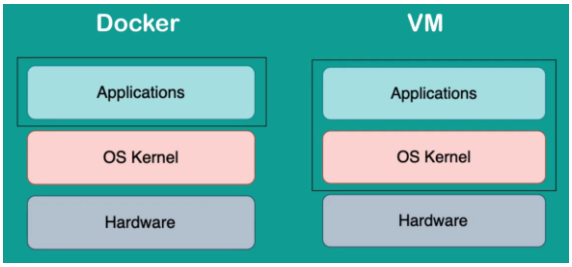


How to bind ? docker stop <id> -> docker run -p<host_port>:<container_port> <container_name> docker start <id>

Docker troubleshooting commands:

Docker logs <contianer_id>/<image_name>
docker run –d –p<hostport>:<cont_port> --name <set_cont_name> <image_name> ➔ docker run in detach mode {run container in background and print container id} from specified image and apply specified port binding and set container name
docker exec –it <cont_id> /bin/bash ➔ opens container in interactive mode logged in as root user to access file system





Docker virtualises just the applications layer of OS and runs on host OS kernel, whereas VM virtualises the complete os system – applications and kernel meaning doesn't use host kernel but boots its own
That's why docker Is light weight fast and more compatible

Docker Workflow: Developer running a js application which uses mongoDB on his local machine , application code committed to git which triggeres cicd pipline to produce artefacts like js build and docker image from js build , this docker image is pushed to a private docker repo, dev/test server pulls js image form private repo and public mongodb image from docker hub for testers to work

Development with docker for above use case -
1. Docker pull mongo {for DB} & docker pull mongo-express {shows DB as UI by running on a sep port }
2. Connection between mongo and mongo-express by docker network
docker network create mongo-network
docker ls

3. run mongo container
docker run –d \ //detach mode
-p 27017:27017 \ // port binding
-e MONGO_INITDB_ROOT_USERNAME=admin \ // root user settings environment variables
-e MONGO_INITDB_ROOT_PASSWORD=admin \
--name mongodb \ //set container name
--net mongo-network \ // use created network
mongo // image chosen

4. start mongo-express container

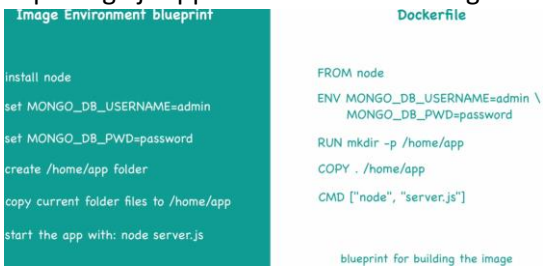
docker run –d \ //detach mode
-p 8081:8081 \ // port binding
-e ME_CONFIG_MONGODB_ADMINUSERNAME=admin \ // root user settings environment variables
-e ME_CONFIG_MONGODB_ADMINPASSWORD =admin \
-e ME_CONFIG_MONGODB_SERVER=mongodb \ // env var to choose mongodb cont name
--name mongo-express \ //set container name
--net mongo-network \ // use created network
mongo-express // image chosen

check localhost:8081 is running -> create DB form ui

5. connect nodejs with mongo -> node module in nodejs configured with ort 27017

6. node application already running on localhost:3000 is able to connect to mongo and app is able to update make changes in db which is reflected on mongo-express ui running on localhost:8081

7. package js application as docker image using Dockerfile



From base image {node}, optional – set mongo params , make dir and copy files, CMD acts as entry point TO START SERVER.JS FILE
docker build –t <image:vers> .
view in docker images command op
docker run <image:vers>
docker exec –it <cont_id> /bin/sh and verify server.js is present in /home/app

to make changes in app/dockerfile and remake image – docker rm <cont_id> and docker rmi <image_name>

8. Push js docker image to aws private docker repo
aws-> ecr -> create repo ; aws repo 1 per image , different versions -> view push commands
docker login step – for aws use command : aws ecr get-login –no-include-email –region us-east-1
tag image : docker tag <local_Imgname:tag> <registryDomain/imageName:tag>
docker push <registryDomain/imageName:tag>

9. Deploy image to test server from ecr {for js image} and docker hub {for mongo and mongo-express images} using docker compose file [defined after this step]

```yaml
version: '3'
services:
  my-app:
    image: 664574038682.dkr.ecr.eu-central-1.amazonaws.com/my-app:1.0
    ports:
    - 3000:3000
  mongodb:
    image: mongo
    ports:
    - 27017:27017
    environment:
    - MONGO_INITDB_ROOT_USERNAME=admin
    - MONGO_INITDB_ROOT_PASSWORD=password
  mongo-express:
    image: mongo-express
    ports:
    - 8080:8081
    environment:
    - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
    - ME_CONFIG_MONGODB_ADMINPASSWORD=password
    - ME_CONFIG_MONGODB_SERVER=mongodb
```

Use this docker –compose file on test server VM; have to use docker login to ecr
docker-compose –f <composeFile> up

Docker compose : yaml File containing individual docker commands if we have to execute again

Since same docker compose file for both docxker urn commands, no need to define docker network , created by default
docker-compose –f <file path> up - > to start containers from compose file
docker-compose –f <file path> down -> to stop all containers defined in compose file and removes network

Data is lost when container is restarted // like fresh installation ; but can be persisted by volumes

DOCKER VOLUMES :: data gets wiped after container restarts ; volumes used to persist data in mongoDB
docker-compose file – define vol and map to mongodb image{map data path}

```yaml
version: '3'
services:
  # my-app:
  # image: ${docker-registry}/my-app:1.0
  # ports:
  # - 3000:3000
  mongodb:
    image: mongo
    ports:
      - 27017:27017
    environment:
      - MONGO_INITDB_ROOT_USERNAME=admin
      - MONGO_INITDB_ROOT_PASSWORD=password
    volumes:
      - mongo-data:/data/db
  mongo-express:
    image: mongo-express
    ports:
      - 8080:8081
    environment:
      - ME_CONFIG_MONGODB_ADMINUSERNAME=admin
      - ME_CONFIG_MONGODB_ADMINPASSWORD=password
      - ME_CONFIG_MONGODB_SERVER=mongodb
volumes:
  mongo-data:
    driver: local
```

docker volumes located at c/programData/docker/volumes or /var/lib/docker/volume

Kubernetes

Open source container orchestration tool – to manger containers (can be docker containers or other) – applications can consist of multiple micro services running in containers - HA, scalability, DR

Architecture – 1) master node{low resources} and 2)multiple worker nodes(with each node running kubelet process{interaction between node and container to start pod within container} and container runtime{to run containers} and kube proxy {to fw the traffic between pods}) , worker nodes may contain docker containers(multiple) deployed for multiple applications ; 3)Virtual network – enables interaction between worker nodes and master node

Master node runs 1)API server process (running through container, entry point to Kubernetes cluster- for UI access, API access , CLI access, also cluster configuration goes through api server via api/cli/ui – request format yaml/json) , 2)control manager (running through container , mgmt process for cluster – repairs , restart etc.) , 3)scheduler (running through container, workload distribution, which worker node will get next process) , 4)etcd (key value store - state management process to store current cluster configuration, backup and restore process)

Worker nodes –each worker node can have multiple A)pods , each pods can have multiple containers{ helper containers to application container} , pod is smallest unit of K8{abstraction over container} (usually 1 pod per application) which has a ephemeral ip address {internal} assigned by virtual network ; B)service (different component) used as substitute of ip address as it remains in place and also acts as load balancer , service types internal {only accessible by other service – DB service}and external {application accessible from browser – js service}, external service is exposed via C)ingress which accepts reqs and forward to external service ; D) config maps – stores external configuraton of application {eg if DB user name/password/url if changed, only config map is adapted and not the entire rebuild ,push, pull steps} ; E) Secret- encoded base64 format to store credentials ; F)Volumes – attaches physical storage{local/remote outside of k8 cluster} to pods for persisting data after pods restart ; F) DEPLOYMENT – blue print for application pods for scaling up and down & HA replication to other nodea , pods not created directly instead deployment configured to create pods ; G) StatefulSet - DB replica to other nodes also done for HA but needs to be consistent using same data store,synchronized so cant be replicated by deployment , for applications like mysql, mongodb, elasticsearch

Working explained: For HA, scalability – for worker setup with 2 nodes having replication of application pods and DB pods, requests when visits the app in browser it is received on ingress {loadbalanced} which forwards the req to app servie {load balanced for 2 app pods replica}, if the req wants to access db , it is forwarded to DB service {loadbalanced , handled by any one replica } ; For DR – if 1 node is crashed , master node-> control manager schedules new replica of node {with new replica of application pods and DB pods} //smart scheduling , master-> etcd stores cluster state of worker pods and create backup to store on remote storage //self healing

Minikube – testing k8 cluster setup with 1 mode where master and worker both run on same node{need virtualization/hypervisor}; already installed docker container runtime and creates virtual box on laptop ; uses kubectl – cli tool {for api server of master node} for k8 cluster to setup services and pods on minikube [configuring minikube cluster]; will use minikube commands to just start and delete the cluster, everything else done by kubectl
minikube start –vm-driver=hyperkit -> start up minikube k8 cluster
minikube status -> checks if host,kubelet,api-server,kubeconfig services of minikube are running or not
kubectl get nodes - > gives status of all nodes {only minikube node in this case which is master}

kubectl version
kubectl get services / kubectl get pods / kubectl get deployment / kubectl get replicaset
kubectl create deployment <name> --image=<docker_img_name> {as we don't create pods directly}
Abstraction layer : deployment ->replicaset->pod->container ; all below deployment are managed by k8 automatically
kubectl edit deployment <name> -> opens the auto generated config file in vi editor-> change and save
kubectl logs <pod_name> & kubectl describe pod <pod_name>-> logging and debugging
kubectl exec –it <pod_name> -- bin/bash -> login to container terminal with root user , can see file system
kubectl delete deployment <dep_name> -> deletes deployment and replica set
kubectl apply –f <file_name.yaml> -> apply changes to deployment explicitly from config file
kubectl delete –f <file_name.yaml> -> delete deployment explicitly from config file
Config File details : yaml
3 parts –0th - api version, kind of component , 1st metadata – name, Labels {labels are used as connectors key value pair which are called from specification->selector section to build connection}, 2nd specification (specific to kind {service or deployment or etc..}) -# of replica, pod specification , image , port binding from service config etc.., 3rd Status – auto generated to adjust specified config{from config file} with actual running component{from etcd – kubectl get deployment <dep_name> -o yaml > output_file.yaml}
For a deployment kind config file, inside specification section -> template -> you can find pod meta data and specification

End2end complete application setup with K8 cluster compo uding config file: Web service which uses MongoDB(and mongo-express for DB visualization) : We will create following – a)MongoDB pod which will be connected via a internal service only, no ext req, b)mongo express deployment which needs db url (through aa config map) and db username,password (thorugh secrets) accessible through browser using a ext service(node's ip and port of ext service) : req flow- browser-> mongo express ext service-> mongo-express pod-> mongodb int service (using config map db url) -> mongodb pod (auth using secrets)
Steps – minikube cluster already running – 1) create mongodb deployment from mongo-config.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
  labels:
    app: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
      - name: mongodb
        image: mongo
        ports:
```

```yaml
        ports:
        - containerPort: 27017
        env:
        - name: MONGO_INITDB_ROOT_USERNAME
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: mongo-root-username
        - name: MONGO_INITDB_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mongodb-secret
              key: mongo-root-password
```

username,password will be taken from secrets
2) create mongo-secret.yaml which takes base64 encoded value that can be taken from terminal using command echo –n 'admin' | base64

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
type: Opaque
data:
  mongo-root-username: dXNlcm5hbWU=
  mongo-root-password: cGFzc3dvcmQ=
```

kubectl apply –f mongo-secret.yaml & kubectl get secret

3)kubectl –f mongo-config.yaml & kubectl get all

4)create mongodb internal service – see port config and selector for connecting to mongo pod

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongodb-service
spec:
  selector:
    app: mongodb
  ports:
    - protocol: TCP
      port: 27017
      targetPort: 27017
```

kubectl apply –f mongo-service.yaml & kubectl get service & kubectl describe service <service-name> for getting service endpoint ip and port

5) mongo-express deployment using mongo-express.yaml – username password from secrets and db config from config map

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-express
  labels:
    app: mongo-express
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo-express
  template:
    metadata:
      labels:
        app: mongo-express
    spec:
      containers:
      - name: mongo-express
        image: mongo-express
        ports:
        - containerPort: 8081
        env:
```

```yaml
env:
- name: ME_CONFIG_MONGODB_ADMINUSERNAME
  valueFrom:
    secretKeyRef:
      name: mongodb-secret
      key: mongo-root-username
- name: ME_CONFIG_MONGODB_ADMINPASSWORD
  valueFrom:
    secretKeyRef:
      name: mongodb-secret
      key: mongo-root-password
- name: ME_CONFIG_MONGODB_SERVER
  valueFrom:
    configMapKeyRef:
      name: mongodb-configmap
      key: database_url
```

6) mongo-express config map for db endpoint using mongo-express-configmap.yaml – takes mongo service name

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongodb-configmap
data:
  database_url: mongodb-service
```

kubectl apply –f mongo-express-configmap.yaml
kubectl apply –f mongo-express.yaml

7) mongo-express ext service from mongo-express-service.yaml by type:loadBalancer and nodeport{port tobe put in browser between 30000 and 32767}

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mongo-express-service
spec:
  selector:
    app: mongo-express
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
      nodePort: 30000
```

kubectl apply –f mongo-express-service.yaml

8)miikube service mongo-express-service to get ext ip of mongo=express service – launches browser

K8 namespaces – grouping: organise compo inside a cluster by acting as a virtual cluster
eg. 1 for db, 1 for monitoring compos, 1 for nginx-ingress
By default 4 available kubectl get namespace : kube-system : Do not modify- for int system processes ,kube-public: publically accessible data stored in a config map, kube-node-lease:info about heartbeats of node ,default: resources cerated here if not defined elsewhere
to crate namespaces – kubectl create namespace <name> or using config file
to use resources from other namespace- componame.namespacename
some compo like vol and nodes cant be created in namespaces but should be placed globally in k8 cluster
to create compo in specific namespaces – kubectl –f <config.yaml> --namespace=<namespace_name> or inside config file under metadata : namespace: =<namespace_name>
Change user defined namespace to make it as default for all compo : install kubens , kubens <user_defined_nmspace>

Ingress{uses website url} : Alternative to external service for accepting incoming traffic for webservice accessible through website{which uses ip:port}
flow:request comes to website->ingress->int service ->pod
create ingress compo thorugh config file pointing to int service

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: myapp-ingress
spec:
  rules:
  - host: myapp.com
    http:
      paths:
      - backend:
          serviceName: myapp-internal-service
          servicePort: 8080
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-internal-service
spec:
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
```

Also need to install ingress controller to manage redirection and evaluating rules – third party implementations available – eg on minikube : use command minikube addons enable ingress
Steps : 0) install ingress controller , 1) define ingress rules in config file : kubectl –f apply ingress.yaml , 1) kubectl get ingress –n <namespace_nm> --watch to extract host ip address and input in /etc/hosts
Default ingess backend can also be defined for all cases where no backend rules fits called path ;; multiple path{google.com/maps, google.com/gmail} can be defined for same host using multiple path in config file pointing to different int services ;; subdomain can also be declared using ingress using multiple hosts in config file pointing to different int services (gmail.google.com , maps.google.com) ;; tls certificate (https) can be configed as well using tls tag inside spec level of config file

Helm: Pakcage manager for k8 (package yaml files in helm charts {bundle} for common used appln like db, elasticsearch, monitoring apps and distribute In public{helmhub}/private repo) , acts as templating engine {standard yaml replaced by placeholders that can be imported from values.yaml}

Kubernetes Volume : data persistence , available to all nodes that services even if entire cluster clashes ; persistence vol pv component mapped with actual storage –available types: nfs storage or cloud storage or local storage ; not namespaced-global
persistence vol claim componentpvc is attached to k8 compo that needs storage and claims the vol mapped in persistence vol component
Sorage class compo – provisions persistence vol dyna,mically when peristencevolumeclaim claims it – pod claims storage via pvc->pvc requests storage from storage class-> storage class created pv-> that meets the needs of the claim
Config maps and secrets can be created as kubenetes volumes as well

How to pull image from private k8 repo – 1) create secret compo containing cred for deocker registry: ecr(aws) login though commands available on ecr {usrename, psswd , private repo url} from minikube ssh-> create secret using config file from docker config json
2) configure deployment/pod using secret through imagePullSecrets tag : using config file

Statefulset : for appln/compo that stores data to keep track of its state eg db ; eg node js application getting req(stateless) accesing mongoDB to update/query data based on prev state (stateful) – to avoid data inconsistency – if multiple replicas of stateful application pods are running , 1 becomes master where both read and write are permitted and all other becomes slave where only read is permitted and data is continuously synced from master to all slaves ; data persistence is configured as best practise for stateful sets

Operator : used for stateful appln ; for managing sync lifecycyle, identity and state of replicas ; acts as human operator to automate using control loop mechanism

Prometheus for k8 cluster : using helm chart{initiate setup} to deploy prometheus operator{manage setup} – helm install Prometheus stable/Prometheus-operator -> creates statefulset , replicas, deployments, services and pods , config maps , secrets for Prometheus, grafanna, alert manager

Services : to avoid prob of ephermal ip addr of pod, enables load balancing ; types – cluster ip type {default , internal, needs ingress,forwards req to pods based on selectors} , headless type {for situations if req is targeted to 1 pod , for stateful appln , eg. Master only who is allowed to write data} , nodePort type{ext traffic has access to fix port on each worker node} , loadbalncer type {accessible externally through cloud provider load balancer – node port and cluster ip created automatically}

----------Capstone ----------

3 sections:
pipeline: github public project + docker image repo
docker container : ci pipeline linting step – 2 screenshot (fail + success) + build docker container from dockerfile
Deployment : 1. CD pipeline: push docker container to docker repo – ecr , docker container deployed to k8 cluster – eks / cloudformation or ansible to build own k8 cluster
2. Blue green or rolling deployment – screenshot – circle ci + ec2 instance as eks cluster nodes + kubectl cmd for deployment, pod, ext ip port service + application hosted

Application – nginx hello world my name is MK
Zip – text file with github repo + all screenshots

Design – docker nginx image pull -> change file (my name is MK) -> build application and containerise using dockerfile -> push docker image to ecr -> docker container deployed to eks / own k8 cluster(cloudformation/ansible) -> github check in -> circle ci setup (+ linting + other manual steps automated) and pipeline execution

Solution steps:
https://medium.com/edureka/amazon-eks-ac646c23abf8
https://medium.com/nerd-for-tech/kubernetes-how-to-deploy-an-nginx-image-using-aws-eks-df9ca5d0356d

1. vpc & subnets in us west1

2. cloud 9 env

3. Github repo

4. docker run nginx

5. index.html custom - <body>MK's capstone</body>

6. Docker file

```
FROM nginx:stable-alpine
RUN rm –rf /usr/share/nginx/html/*
COPY ./index.html /usr/share/nginx/html/index.html
```

7. docker build -t nginx:latest .

Test by binding port and curl – docker run -d -p 8080:80 nginx & curl http://localhost:8080

8. make ecr repo on aws and execute push commands – login , tag , push

9. IAM-> eks role

10. eks on aws console

11. eks-> compute-> node group & verify in ec2 it is running

12.install eksctl & kubectl ; update kube-config

13. kubectl create deployment , ext service

14. automate using circle ci

Automation CircleCi - https://github.com/FelixLutze/Cloud-DevOps-Engineer-Capstone-Project

Job 1 test-build : docker image for circleci job –python3.7 ; steps-checkout git code, restore cache, virtual env, make install (from makefile – upgrade pip, install requirements.txt – flask and pylint,wget hadolint), make lint(from makefile – hadolint dockerfile, pylint app.py), save cache

Job 2 upload-docker to docker hub : docker image for circleci job –golang1.15; steps : checkout, docker build, docker login, docker tag, push (pre-req: dockerhub repo)

Job 3 deploy- infrastructure : docker image for circleci job- amazon/aws-cli ; steps : checkout , install dep- tar,gzip , cloudformation deploy network{vpc,internet gw, public route table, public route , nat gws*2 , elastic ips*2,public subnets*2, public subnets route tble association *2 , sec grp},cluster{role,eks cluster},nodegroup{role, nodegrp- worker nodes},management{2 ec2 hosts to manage eks cluster-master nodes} (pre-req: iam user , circleci proj access key and secret key from iam user cred) , extract ip of mgmt. nodes for ansible – aws describe ec2 to inventory.txt

Job 4 configure-infrastructure : docker image for circleci job- python3.7/alpine ; steps: checkout , add ssh key {from pem file of 2 mgmt. nodes, added to circleci project settings, fingerprint added to circleci config.yml} , install dep ansible , ansible configure-server playbook {install aws cli and kubetl , update kube-config command}

Job 5 configure-cluster : docker image for circleci job- python3.7/alpine ; steps: checkout , add ssh key {from pem file of 2 mgmt. nodes, added to circleci project settings, fingerprint added to circleci config.yml} , install dep ansible , ansible configure-cluster playbook {k8 deployment and service from k8 config file}, ansible save-lb-dns-name playbook{kube tl get svc}

Job 6 deploy-docker: docker image for circleci job- python3.7/alpine ; steps: checkout , add ssh key {from pem file of 2 mgmt. nodes, added to circleci project settings, fingerprint added to circleci config.yml} , install dep ansible , ansible deploy-app playbook{}

NETWORKING----------

3 compo – Medium (how you are connecting), Addressing (how you are identifying), Content (what are you sharing)

Public network - Internet
Private network - Reserved range from public cidr – RFC 1918 standard – 10.0.0.0/8 , 172.16.0.0/12 , 192.168.0.0/16

IPV4 addressing – 32 bit , 4 octet – available addresses = 2^32 =4.2B

Classless interdomain routing – eg. /24 notation = 2^(32-24) = 2^8 = 256 addresses //24 means 3 octet are fixed {prefix-24 bits fixed} and only 1 octet is variable -> 192.168.112.{0-255} //out of which first 2 and last is reserved - .0 , .1 and .255

Cidr.xyz

eg.
10.88.135.144
first octet fixed - first 8 bit fixed , else 3 octet variable – available ip addresses count = 2^(32-8) = 2^24
first usable ip = 10.0.0.1 , last usable ip – 10.255.255.254

OSI model – breakdown of various aspects of networking communication

Layer 7 – Application – http/https / dns / ssh – telnet / dhcp / dns/
client server protocols(http{1 connection is created},ftp{2 connections control and data},smtp{ised with po and imap},websockets{websockets used for 2 way communication btw client and server }) and p2p protocols(webRtc {user underlying UDP})

Layer 6 – Presentation – ssl/ tls – data representation , encryption

Layer 5 – Session –socket – interhost communication

Layer 4 – Transport – tcp(reliable, acknowledgement mechanism) and udp(real time) – end to end connections and reliability – segments layer
TCP(sequence and acknowledgement mechanism) and UDP(no ordering , fast) – IP Packets

Layer 3 – Network – ip and icmp – path determination-ipv4/v6 – packet layer

Layer 2 - Data Link - arp , mac address – frames layer

Layer 1 – Physical – 802.11a/b/g/n wire/fiber/bits - media signal and binary transmiision

Encapsulation: layer 7→1 : data is packed up on sending device ;each layer an header is added

Decapsulation: layer 1→7 : data is unpacked on receiving device ; each layer headers are removed

Load balancing Approaches. -

1. Round Robin – sequentially DUMB :p

2. BGP anycast – bgp allows multiple servers to advertise same IP and dns servers responds to queries with same ip address. The domain is resolved based on DNS nameserver that is topographically the closest.

3. Policy based DNS load balancing – eg. client's resolver decides which servers to forward the request to.

4. dedicated load balancing – custom lb within VPC sits in line btw client resolver and nameserver.

Network Virtualization - Software Defined Networking (SDN) – programmatically, offer centralized control, separation of control and forwarding functions

and Network Functions Virtualization (NFV) - offer centralized control, separation of control and forwarding functions, Virtual Machines (VMs) can run network functions such as routing, load balancing and firewalls.

Nameserver - Nameservers help connect URLs with the IP address of web servers. Nameservers are an important part of DNS. Nameservers are the physical phone book itself, **DNS** records are the individual entries in the phone book. Eg. You type “google.com” into the address bar and hit enter-> Your browser uses DNS to retrieve the domain’s nameservers->Your browser asks for the A record that contains the IP address of the web server (a specific DNS record)->The nameservers provide the IP address from the A record->Your browser requests the website content from that IP address->Your browser retrieves the content and renders it in your browser

DNS lookup step – user enters website address-> browser checks in cache-> if not frwds req to main dns resolver which first check in cache -> if not routes req to root server(top level)-> returns top level domain details to dns resolver which makes another req to TLD server-> returns info for .com/.net etc back to dns resolver -> dns calls authoritative name servers-> returns full info to dns resolver which stores in cache and sends ack to web browser which stores in cache and returns back to user
P.S. also explained where talked about route 53

DHCP – Automatically assign IP add to a new joiner computer in network

LAN fundas – switch, router, gateway

Bridge is a networking device that connects the larger LAN networks with the group of smaller LAN networks -

Switch – forwarding traffic within a same network – school peon tells other class student your message

Inter n/w communication::: Router - forwarding traffic outside a network – school security guard tells outside bacha your message ; gateway – ip address of Router ,takes info from route table, 1 n/w gtw connects to another nw gtw

WAN fundas –

Router core functionality – NAT , DMZ, Firewall, Port forwarding
NAT – network address translation of internet addreese to private address space– N/w address of outside component is translated whenever a lan n/w device communicates with it via router
Firewall – outside device and lan n/w device cant talk directly to each other because of firewall, set of rules
DMZ – demilitarized zone – area created within LAN n/w where outside devices can communicate directly bypassing the firewall – eg. parent meets student in school’s reception
Port forwarding – for similar functionality offered by dmz but configured via port

VPN – reach to office system from home – makes a tunnel btw client and server over public internet , encyption

A **proxy server** is a system(application level) or router that provides a gateway between users and the internet. Therefore, it helps prevent cyber attackers from entering a private network. It is a server, referred to as an “intermediary” because it goes between end-users and the web pages they visit online; proxy sends own ip address and hides the client ip address, supports anonymity, cache, security
types - Forward Proxy(client <->proxy<->internet<->server), Transparent Proxy, Anonymous Proxy, High Anonymity Proxy, Distorting , Data Center Proxy, Residential , Public , Shared , SSL , Rotating , Reverse(client <-> internet<->proxy<->server eg. Cdn , helps in ddos and load balancing)

LINUX----------

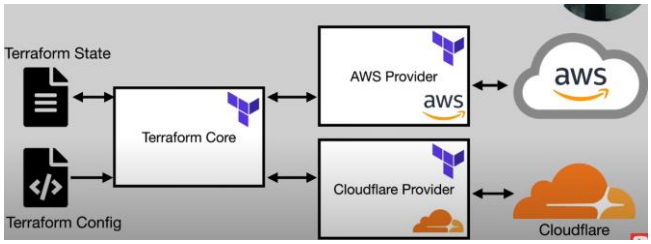
Ls – list files and folders – grp/org/user ; r/w/x
mkdir – make directory
touch – create empty file
nano <filename> / vi – text editor
cat <filename> - read a file
rm – remove/delete
cd change directory
-r recursive
man is for user manual
cp copy
mv move or rename
install packages – apt(ubuntu), yum(aws linux), brew (mac) etc.. package manager
sudo apt install <packagename>
sudo apt purge <packagename> - remove package
sudo super user do
ssh remote login into instance
history command
pwd present working directory
whoiam
sudo su – change to sudo user
which git or which docker - > where is git or docker present
ping - check if coonection can be established
ip addr – returns ip address
nslookup – resolves hostname to ip address nslookup google.com
curl transfers data to get content
wget – download files

Terraform----------

OVERVIEW

Categories of IaC tools: Ad hoc scripts, config mgmt. tools, server templating tools, orch tools, provisioning tools

Hashicorp ; Automate provisioning infra and services on infra- open source and declarative(only tell what to do, not how to do….imperative on the other hand is you tell ehat you want to happen, how, what api calls) ;if you run terraform apply commands 2 times consequently, 1st will provision resources but 2nd wont change a thing
and if you remove a resource from .tf file and rerun terraform apply command, it will delelte that resource from aws
Terraform arch - 1. Terraform core : takes config (what infra to create-desired state) and state(existing state of infra) to compare between current infra and desired infra and creates a plan on what needs to be created/ updated / deleted ; 2. Providers – AWS(IaaS) / Kubernetes(PaaS) / Fastly(SaaS) etc..



Terraform Commands
Refresh – query infra provider to get current state
Plan – create an execution plan – preview
Apply – execute the plan
Destroy – destroy the infra/resources – checks what running using refersh and creates a plan

https://courses.devopsdirective.com/terraform-beginner-to-pro/lessons/00-introduction/01-main

Setup

Steps – 1. Download and install terraform
2. AWS user with programmatic access – roles : rds,ec2, iam, s3 ,dynamodb,route53 all Full access
3. AWSCLI install and configure using access key and secret key -> this will create a credentials file in aws home dir
Alternatively, access key and secret key can be passed in provider section of .tf file
4. Most basic Terraform config file – main.tf // kinda like terraform hello world

```
terraform {

  required_providers {
    aws = {                  // added aws provider
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "example" {
  ami           = "ami-011899242bb902164" # Ubuntu 20.04 LTS // us-east-1
  instance_type = "t2.micro"
}
```

5. Initialise – terraform init command from directory - sets up backend and state storage, creates .terraform folder with provider's exe
5. terraform plan(delta b/w desired and actual calculation) , apply(api call to provider to create/update that delta) and destroy command

6. terraform state list command to output all resources created , and terraform state show <resource from list command> to show details of created resource individually

State file : json file with rosources and data objects deployed using terraform - ext : .tfstate ; can be stored in local backend(working dir of project) or remote backend(eg. Terraform cloud – free upto 5 user/ aws s3{for storage}+dynamoDB{for locking} / gcp cloud storage etc.) These are configured in main.tf files as below,

```
backend "s3" {

  bucket       = "devops-directive-tf-state" # REPLACE WITH YOUR BUCKET NAME

  key          = "03-basics/import-bootstrap/terraform.tfstate"   // tfstate file  location in s3

  region       = "us-east-1"

  dynamodb_table = "terraform-state-locking"

  encrypt      = true
```

OR

```
terraform {

 backend "remote" {   // Terraform cloud
```

```
 organization = "devops-directive"

 workspaces {

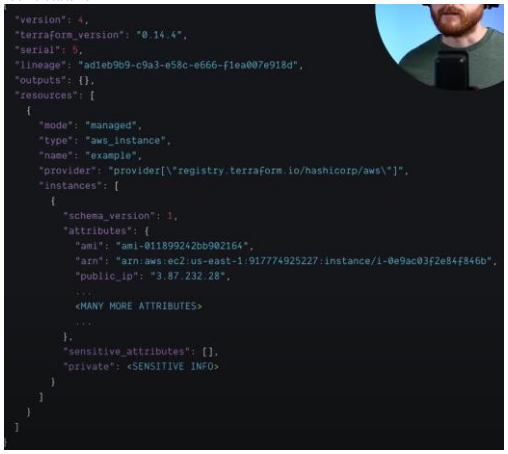
  name = "devops-directive-terraform-course"

 }

}

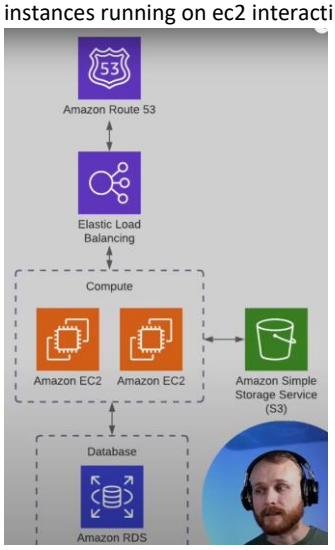
# }
```

Below is .**tfstate** file :



```
{
  "version": 4,
  "terraform_version": "0.14.4",
  "serial": 5,
  "lineage": "ad1eb9b9-c9a3-e58c-e666-f1ea007e918d",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "aws_instance",
      "name": "example",
      "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
      "instances": [
        {
          "schema_version": 1,
          "attributes": {
            "ami": "ami-011899242bb902164",
            "arn": "arn:aws:ec2:us-east-1:917774925227:instance/i-0e9ac03f2e84f846b",
            "public_ip": "3.87.232.28",
            ...
            <MANY MORE ATTRIBUTES>
            ...
          },
          "sensitive_attributes": [],
          "private": <SENSITIVE INFO>
        }
      ]
    }
  ]
}
```

Reference architecture : Basic web application with infra using AWS, DNS using route 53, Elastic load balncing multiple instances running on ec2 interacting with s3, using RDS for DB and default VPC



Main.tf file for above architecture,

```
terraform {

 # Assumes s3 bucket and dynamo DB table already set up // for backend configuraytion – storing state file

 backend "s3" {  // backend configuration goes within the top level terraform {}

  bucket        = "devops-directive-tf-state"

  key           = "03-basics/web-app/terraform.tfstate"

  region        = "us-east-1"

  dynamodb_table = "terraform-state-locking"

  encrypt       = true

 }
```

```
  required_providers {  // where terraform operates
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "instance_1" {  // 2 ec2s are created with basic python webserver
//format:: resource “provider_resourceType” “name” {key value pairs}
  ami           = "ami-011899242bb902164" # Ubuntu 20.04 LTS // us-east-1
  instance_type   = "t2.micro"
  security_groups = [aws_security_group.instances.name]
  user_data       = <<-EOF
            #!/bin/bash
            echo "Hello, World 1" > index.html
            python3 -m http.server 8080 &
            EOF
}

resource "aws_instance" "instance_2" {
  ami           = "ami-011899242bb902164" # Ubuntu 20.04 LTS // us-east-1
  instance_type   = "t2.micro"
  security_groups = [aws_security_group.instances.name]
  user_data       = <<-EOF
            #!/bin/bash
            echo "Hello, World 2" > index.html
            python3 -m http.server 8080 &
            EOF
}
resource "aws_security_group" "instances" {  // sec grp to allow inbound traffic to webservers
```

```
  name = "instance-security-group"
}

resource "aws_s3_bucket" "bucket" { // bucket and its configuartions
  bucket_prefix = "devops-directive-web-app-data"
  force_destroy = true
}

resource "aws_s3_bucket_versioning" "bucket_versioning" {
  bucket = aws_s3_bucket.bucket.id
  versioning_configuration {
    status = "Enabled"
  }
}

resource "aws_s3_bucket_server_side_encryption_configuration" "bucket_crypto_conf" {
  bucket = aws_s3_bucket.bucket.bucket
  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}

data "aws_vpc" "default_vpc" { //Reference Default VPC and Subnet – since already exist, data onbject is used instead of resource to just retrieve info but not manage by terraform
  default = true
}

data "aws_subnet_ids" "default_subnet" {
  vpc_id = data.aws_vpc.default_vpc.id
}

resource "aws_security_group_rule" "allow_http_inbound" {  //sec grp rule to specify that inbound traffic on port 8080 can be routed to our virtual machines
  type            = "ingress"
```

```
  security_group_id = aws_security_group.instances.id

  from_port   = 8080
  to_port     = 8080
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
}

resource "aws_lb_listener" "http" { // setup LB to split traffic , attach 2 ec2s
  load_balancer_arn = aws_lb.load_balancer.arn

  port = 80

  protocol = "HTTP"

  # By default, return a simple 404 page
  default_action {
    type = "fixed-response"

    fixed_response {
      content_type = "text/plain"
      message_body = "404: page not found"
      status_code  = 404
    }
  }
}

resource "aws_lb_target_group" "instances" {
  name     = "example-target-group"
  port     = 8080
  protocol = "HTTP"
  vpc_id   = data.aws_vpc.default_vpc.id

  health_check {
    path                = "/"
```

```
    protocol            = "HTTP"
    matcher             = "200"
    interval            = 15
    timeout             = 3
    healthy_threshold   = 2
    unhealthy_threshold = 2
  }
}

resource "aws_lb_target_group_attachment" "instance_1" {
  target_group_arn = aws_lb_target_group.instances.arn
  target_id        = aws_instance.instance_1.id
  port             = 8080
}

resource "aws_lb_target_group_attachment" "instance_2" {
  target_group_arn = aws_lb_target_group.instances.arn
  target_id        = aws_instance.instance_2.id
  port             = 8080
}

resource "aws_lb_listener_rule" "instances" {
  listener_arn = aws_lb_listener.http.arn
  priority     = 100

  condition {
    path_pattern {
      values = ["*"]
    }
  }

  action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.instances.arn
  }
```

```
}

resource "aws_security_group" "alb" {
  name = "alb-security-group"
}

resource "aws_security_group_rule" "allow_alb_http_inbound" {
  type              = "ingress"
  security_group_id = aws_security_group.alb.id

  from_port   = 80
  to_port     = 80
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]

}

resource "aws_security_group_rule" "allow_alb_all_outbound" {
  type              = "egress"
  security_group_id = aws_security_group.alb.id

  from_port   = 0
  to_port     = 0
  protocol    = "-1"
  cidr_blocks = ["0.0.0.0/0"]

}

resource "aws_lb" "load_balancer" {
  name               = "web-app-lb"
  load_balancer_type = "application"
  subnets            = data.aws_subnet_ids.default_subnet.ids
  security_groups    = [aws_security_group.alb.id]
```

```
}

resource "aws_route53_zone" "primary" { //access application via route53 dns record for domain devopsdeployed.com iunstead of autogenerated domain of LB
  name = "devopsdeployed.com"
}

resource "aws_route53_record" "root" {
  zone_id = aws_route53_zone.primary.zone_id
  name    = "devopsdeployed.com"
  type    = "A"

  alias {
    name                   = aws_lb.load_balancer.dns_name
    zone_id                = aws_lb.load_balancer.zone_id
    evaluate_target_health = true
  }
}

resource "aws_db_instance" "db_instance" {  //RDS
  allocated_storage = 20
  # This allows any minor version within the major engine_version
  # defined below, but will also result in allowing AWS to auto
  # upgrade the minor version of your DB. This may be too risky
  # in a real production environment.
  auto_minor_version_upgrade = true
  storage_type               = "standard"
  engine                     = "postgres"
  engine_version             = "12"
  instance_class             = "db.t2.micro"
  name                       = "mydb"
  username                   = "foo"
  password                   = "foobarbaz"
```

skip_final_snapshot = true

This is followed by executing terraform init, plan and apply command, tested by accessing LB url or route53 dns url

Variables
Types:
1. Input variable – input arguments for a function(can be defined within a separate variables.tf or within main.tf) – declared as

```
variable "instance_type" {

  type = string
  default = "t2.micro"
}
```

called as ”var.<var_name>” eg . var.instance_type
2. Local variable – temp vars within scope of a function(in your main.tf file) – declared as

```
locals {

  service_name = "example-service"
  owner = "your_name"
}
```

called as “local.<var_name>” eg . local.owner
3. Output variable - returns value from a function (can be in separate tf file or main tf file)
Syntax :

```
output "instance_ip" {

  value = aws_instance.example.public_ip
} //outputs on the cmd line where apply command is executed from
```

Terraform refresh command – down’t apply the tf file but executes to display output if outputs section added freshly

Input variables can be setup based on following precedence low to high :
1. Terraform CLI prompts: If you don't specify a variable anywhere and there's no default value, the Terraform CLI will prompt you for a value. – manually enter during plan/apply

2. Default value in the dceclaration block: You can specify a default value when declaring the variable.

3. Environment variables: Use the prefix TF_VAR_ followed by the variable name.

4. Terraform .tfvars files: Store values in .tfvars files for non sensitive data eg. bucket_prefix = "devops-directive-web-app-data"
Bydefault terraform.tfvars file is applied , if explicit apply another tfvars file – terraform apply –var-file=<file_name>.tfvars

5. Auto-applied .auto.tfvars files: These files will be applied over the .tfvars files.

6. Cmd line : -var or -var-file options: Pass values when issuing the terraform plan or terraform apply commands for sensitive data
variable will be defined without value like .. as value passed during runtime
variable "db_pass" {

description = "Password for DB"

type = string

sensitive = true // LOOK AT THAT

}
eg. terraform apply -var "db_user=my_user" -var "db_pass=something_super_secure"

Types :
Primitive : String , number , Boolean
Complex : list , set, map, object, tuple

Sensitive data can be handled in variables by setting “sensitive = true” attribute when defining the variable. This will cause those data to be masked in the Terraform plan output to prevent leaking credentials. Other alternatives for using sensitive data is env vars / -var command(retrieved from secret manager at riuntime) / external secret stores like AWS Secrets Manager or HashiCorp Vault.

Terraform advanced features

1. Template strings: Similar to JavaScript, you can use template strings with curly braces to reference variables within a string.
2. Operators: Arithmetic and logical operators like multiplication, division, and checking equality are available.(!,-,*,/,%,>,== etc)
3. Conditionals: Ternary syntax can be used for conditional expressions.(cond?true:false)
4. For loops: for loops can be used to loop over a list of configurations. (for o in var.list : i.id)
5. Splat expressions: This expands values in a list.(var.list[*].id)
6. Functions: Math functions, date and time functions, and hash/crypto functions can be used in your code. – numeric/ string/ collection /encoding/ filesystem/ ip network/ type conversion
7. Meta Arguments :to control behaviour of resuources ;
   “depends on” – to enforce depeneency ordering, where 2 resources depends on each other (but not on each others data – no direct connection within confiug) eg. if an instance needs access to an S3 bucket, you need to provision the role policy first

```
resource "aws_instance" "example" {

  # ...

  depends_on = [aws_iam_role_policy.example]
}
```

   “count” – to create multiple copies of a nearly identical resource.

```
resource "aws_instance" "example" {

  # ...

  count = 4
  tags = {
    Name = "server-${count.index + 1}"
  }
        }
```

“for_each” - loop through an iterable and create resources based on its values.

```
```json
locals {
  subnets = ["subnet-abc123", "subnet-def456"]
}

resource "aws_instance" "example" {

  # ...

  for_each = toset(local.subnets)
  subnet_id = each.value
        }
```

   “Lifecycle”- specifies order in ehich terraform does changes

```
resource "aws_instance" "example" {
```
```

```
  # ...

  lifecycle {
    create_before_destroy = true    //deployments zero downtime
    ignore_changes = [tags]  //prevents to revert metadata being set elsewhere
    prevent_destroy = true  //reject any plan to destroy this resource
  }
        }
```

8. Provisioners – perform actions on local or remote machine
standard – file, local-exec ,remote-exec
Vendor specific- ansible/ chef/puppet provisioners

Modules
Containers for multiple resources used together – collection of .tf and tf.json files – better organisation
to package and reuse resource configuration within terraform ; abstraction , grouping into logical units

Types : Root module - default module condidting of every .tf file in main working dir ; Child module – separate module referenced from root module, can receive input variables from root module and use meta args
Sources : *Local* (reference using relative path, eg breakup config into diff compos compute , db, storage and ntwing

```
module "web-app" {

  source = "../web-app-modules-dir"
}
```

```
❯ tree ./06-organization-and-modules
.
├── README.md
├── web-app
│   └── main.tf
└── web-app-module
    ├── compute.tf
    ├── database.tf
    ├── dns.tf
    ├── main.tf
    ├── networking.tf
    ├── outputs.tf
    ├── storage.tf
    └── variables.tf
```

```
variable "db_pass_1" {
  description = "password for database #1"
  type        = string
  sensitive   = true
}

variable "db_pass_2" {
  description = "password for database #2"
  type        = string
  sensitive   = true
}

module "web_app_1" {
  source = "../web-app-module"

  # Input Variables
  bucket_prefix    = "web-app-1-data"
  domain           = "devopsdeployed.com"
  app_name         = "web-app-1"
  environment_name = "production"
  instance_type    = "t2.micro"
  create_dns_zone  = true
  db_name          = "webapp1db"
  db_user          = "foo"
  db_pass          = var.db_pass_1
}
```

folder hierarchy in above picture, and in main.tf, variables declarartion part and calling the entire module ”web-app-module” with variables which has all the dependent .tf files)

/ *Terraform registry* (eg. Consul module from terraform registry would provision 52 diff resources win AWS account - ec2 , ntw, iam just by calling module in main .tf file and executing init and plan command

```
module "consul" {

  source = "hashicorp/consul/aws"
  version = "0.1.0
}
```

)

/ *Github* (using http or ssh link-

```
module "example" {

  source = "github.com/hashicorp/example?ref=v1.2.0"
})
```

/ *Generic Git repo*(url with username and pswd

```
module "example" {

  source = "git::ssh://username@example.com/storage.git"
})
```

Multiple environment mgmt for prod staging and dev

2 approaches -
Terrraform workspaces – multiple named sectoins within a single backend; terraform.workspace expression used to reference workspace name in .tf file

```
locals {
  environment_name = terraform.workspace
}

resource "aws_s3_bucket" "bucket" {
  bucket = "web-app-data-${local.environment_name}"
}
```

; min code duplication

```
→  ~ terraform workspace new dev
Created and switched to workspace "dev"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
→  ~ terraform workspace list
  default
* dev
  production
  staging
```

new command to create new workspace ; terraform workspace select <workspace name> to select a workspace

Subdirectories – isolations of backend but more code duplication

```
├── dev
│   ├── main.tf
│   └── terraform.tfvars
├── production
│   ├── main.tf
│   └── terraform.tfvars
└── staging
    ├── main.tf
    └── terraform.tfvars
```

further separation at logical components groups can be done for larger projects using modules

Testing Terraform code

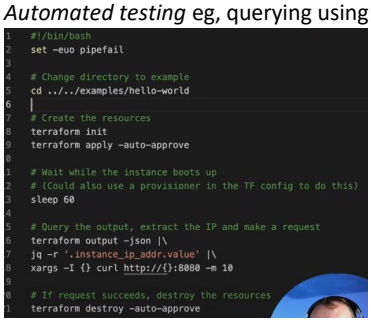
Static checks : built in(terraform fmt -check for checking formatting of .tf file , terraform validate, terraform plan, custom rules –

```
variable "short_variable" {
  type = string

  validation {
    condition = length(var.short_variable) < 4
    error_message = "The short_variable value must be less than 4 characters!"
  }
}
```

) / external (tflint, checkov, terrascan , terraform-compliance, synk, terraform sentinal, terratest package form gruntwork using go lang)

Automated testing eg, querying using http req on provisioned infra ip address using shell script



```
#!/bin/bash
set -euo pipefail

# Change directory to example
cd ../../examples/hello-world

# Create the resources
terraform init
terraform apply -auto-approve

# Wait while the instance boots up
# (Could also use a provisioner in the TF config to do this)
sleep 60

# Query the output, extract the IP and make a request
terraform output -json |\
jq -r '.instance_ip_addr.value' |\
xargs -I {} curl http://{}:8080 -m 10

# If request succeeds, destroy the resources
terraform destroy -auto-approve
```

auto-approve flag used to avoid prompt

END2end workflow of terraform development proj
write updated code-> run locally for dev env ->pull req in version control system-> running tets via CI -> deploy to staging via CD on merge to main -> deploy to prod via CD on release
makefiles can be used to prevent human error
CICD tools supported – github actions, circleci, gitlab etc

Refrencing within terraform file

```
resource "aws_vpc" "first-vpc" {
  cidr_block = "10.0.0.0/16"
  tags = {
    Name = "production"
  }
}

resource "aws_subnet" "subnet-1" {
  vpc_id     = aws_vpc.first-vpc.id
  cidr_block = "10.0.1.0/24"

  tags = {
    Name = "prod-subnet"
  }
}
```

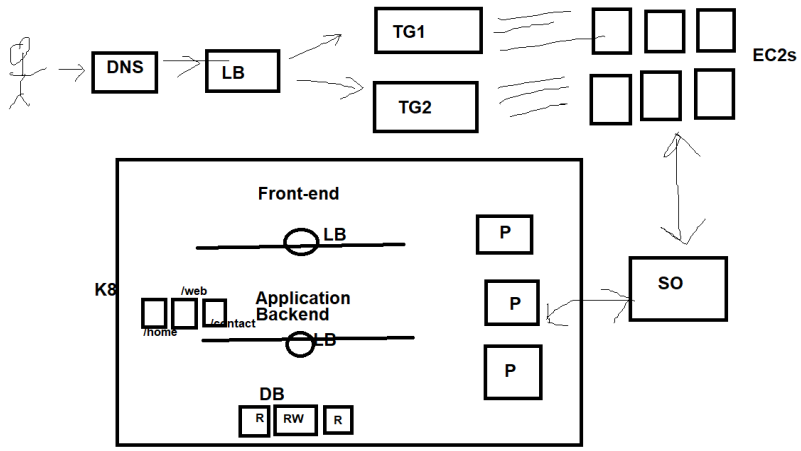
Also, order of declaring resources in tf file doesn't matter .
terraform destroy/apply –target <resource reference> just creates/destroys a specific resource keeping everything else untouched

SYSTEM DESIGN _----------

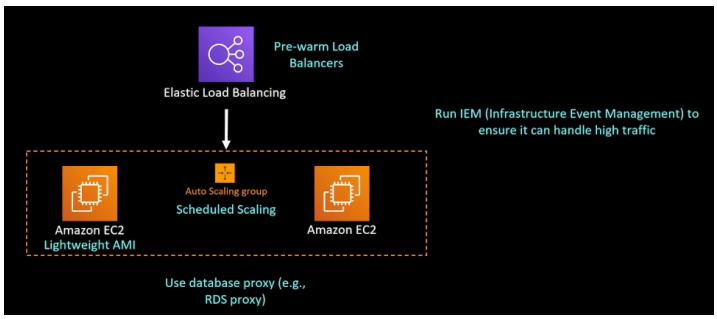
Top 5 components - Microservices (LB or API gtw; sync vs async patterns), DB selection (sql vs nosql), Caching (DB, cdn), Security(authentication, autherisation, encryption at rest and in transit), scalable and HA

1. Scaling
A)system able to scale up and down based on real time traffic

Solution – user is accessing the service via a dns url provided by route 53 whch forwards the request to a Load balancer, Target groups(collection of machines) are attached to Lb hosted in diff availability zones which refer to individual EC2 nodes. AWS provides autoscaling for ec2 instances.
Target group nodes are managed by K8 cluster service object SO (kinda like LB) which interacts with diff pods on which application is deployed. Application generally has frontend, backend layer for application and DB which are containerised. Load balancer may be placed btw front end and application layer; autoscaling of DB can be configured using a messaging layer where read operation is given to 1 replica of DB (primary). Kubernetes also supports HPA horizontal pod scaling where more pods are added to vm's capacity , and when vm capacity reached more vm is added {horizontal cluster scaling} with more pods (suggested for application mainly) and vertical pod scaling (suggested mainly for DB).

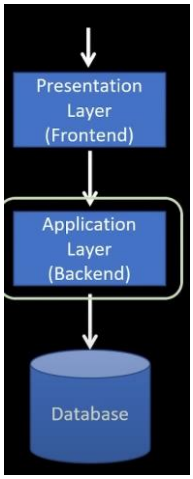


B) Vertical scaling – moving a server to more powerful server - scaling up and dwon take longer with chances of missing transaction and expensive, limit on scaling because of hardware , fast interprocess communication
Horizontal scaling – adding new servers -faster, massively scalable , cost effectivr as better utlization ; applications needs to be compatible eg. Lambda ; slow network calls communication – remote procedural calls; data consistency limitations, resilient

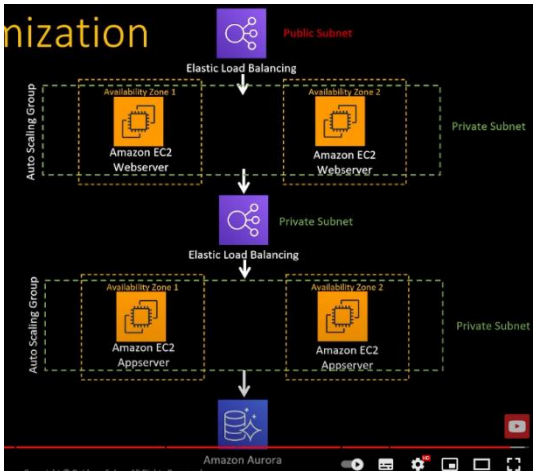


solution for black Friday sale explained – using a pre warmed load balancer with target as a auto saling group configured with scheduled scaling, vms needs to have a light weight AMI and if DB , use a DB proxy for multiple fast connection attempt,; also use an IEM service and break application into smaller microservices as just going into K8 or serverless doesn't eliminate challenges

C) 3 tier architecture

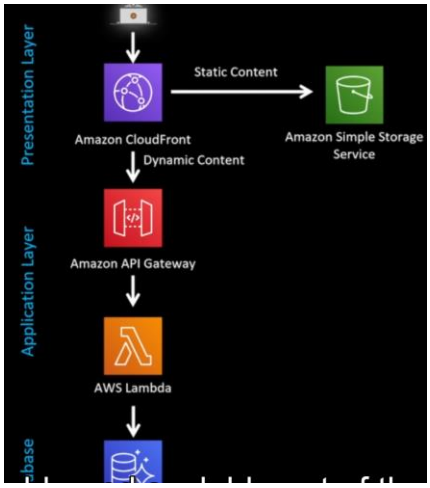


Front end – website using nginx or apache webserver ; backend – business logic running java code in jar file on apache tomcat srever or oracle webserver or api running in container/ lambda; db- oracle, mysql,postgreSQL, dynamoDB
AWS soln –

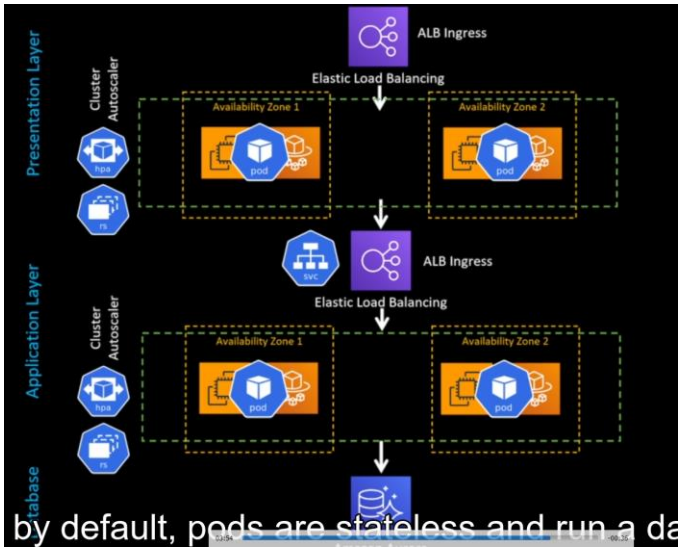


Frontend running on ec2 vmswith webserver installed reached via elastic load balancer dns url, backend running on ec2 vmswith appserver installed reached via elastic load balancer
Scaling – elb is alrady scalable, ec2 vms needs to be in autoscaling group across 2 diff availability zones
network security – only front end elb in public subnet and everuthing else in private subnet , ntw access control list nacl , sec groups and WAf can be implemnetted
DB- sql or nosql selection , aws native dbs selection – dynamoDB/Aurora which are already resilient with multi AZ and global db replication – read replica and caching layer for db optimization

3 tier arch using serverless: static content stored in s3 bucket exposed via cloudfront , dynamic content called via api gtw apis with business logic at aws lambda which can interact with DB services like dynamoDB/Aurora , availability and scalability out of box



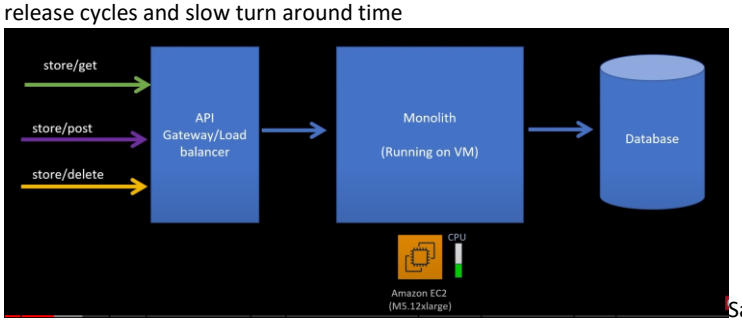
3 tier arch using kubernetes – app server running in a container within a pod – ha via 2 pods in 2 AZ , scalability via k8 concepts of replica set , hpa horizontal pod autocaler



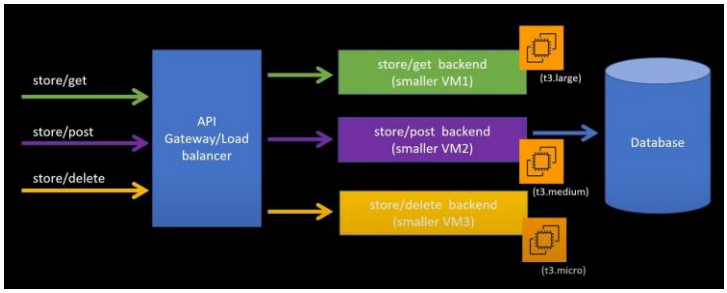
Scaling techniques – lb separating multiple app servers, db replication & scaling, cache, cdn, multiple data centres,message queue,

2. Monolith vs Microservice arch

Monolith – single tiered appln where user interface and data access code is combined into a single platform eg . Java .jar file ; simple, resource efficient at some scale ; no modularity , scaling challenge , all or nothing deployment with long release cycles and slow turn around time

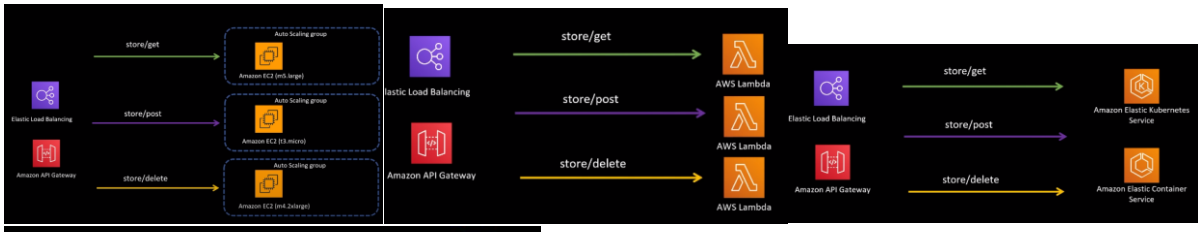


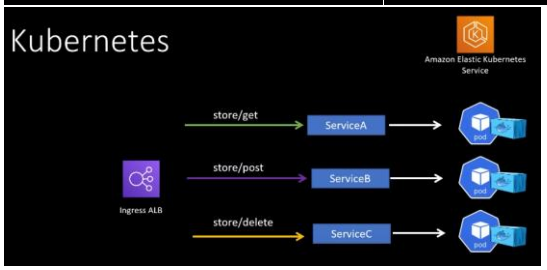
Same backend for entire application, if a particular end point receives more traffic, entire application needs to be scaled up.



Application divided into smaller components. Even DB can be separate for different backends. If a particular end point receives more traffic, just 1 corresponding component needs to be scaled up.

Microservvices on AWS can be hosted on diff autoscaling grps with elastic load balancing or api gateway as entry points ; can be used for serverless backend using lambda or containerised backedn using eks and ecr ;





Mix and matched as well!

Limitations of bad microservices arch – transaction management,latency, tightly coupled arch(all compo needs to be scaled together)

How to divide into microservices (phases) -

1. patterns– Decomposition : division based on business capability or subdomain domain driven design

Stangler pattern – eg for a big monolithic application which has many workflows , some of its functionality is migrated to microservices architecture but rest is not yet migrated- adding a controller interface which receives all the traffic and redirects(to microservice or monolithitic workflow) based on if its migrated yet or not.. eventually all traffic redirects to microservices

SAGA pattern – sequence of local transactions on db for each service mechanism where service interacts with other service to fetch other db data but cannot touch other db directly

2. Database – db division strategy – partitioning , sharding, read replicas , db for each service (better scaling) or shared db (better quering and transactional properties)etc.

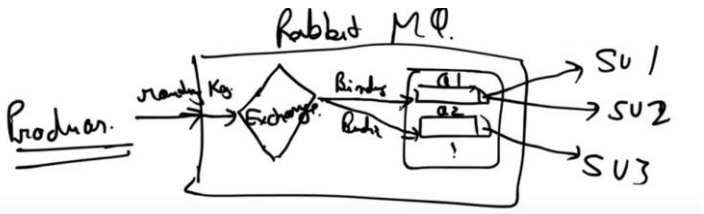
3. Communication – api or events
4. Integration – api gtw
5. Observability – monitoring

3. **Load balancer** –elastic load balancing service on lambda or k8 or ec2s ; types – Application LB (distribute tarrffic to multiple target groups, operates on OSI layer 7 and routes traffic based on url path, ssl supported, supports ec2, ip addr and lambda,aws waf supported, http and https supported), and Network LB (operates on osi layer 4, routes traffic based on protocol and port of incoming traffic, exposes static ip address, supports ec2, ip addr)

4. **API** Gatweay– interface btw application and request source ; aws serverless solution is aws api gateway ; aws waf supported, not possible to get static ip address for endpoint just url, accepts https traffic and able to do request validation and req/rees mapping , able to integrate with lambda ; cache respose functionality

3. Synchoromnous vs event driven (async) architecture -
Sync: response is awaited in same req call invocation eg. Req calls to amazon api gtw which redirectes to aws lambda which invokes amazon dynamoDB, all the components needs to be scaled together (to handle load) and in case of failure at any call, consumer needs to resed transaction for re processesing which is expensive
Async: components decoupling to buffer messages until services are available to process eg. User calls to api gateway which rediuects to amazon sqs simple queue service which loads up aws lambda which calls amazon dynamo dB, so if traffic increases only api gtw needs to be scaled up , each components can scale independently with built in retry mech, cost effective as user doesn't need ot resend complete req
Both architectures can be used hand in hand – eg ordering system where order insers can be event driven and order retrieval can work synchronously

4. Queues vs PubSub –
Queue: pool architecture, one consumer as soon as messaged is retriueved from queue and processed, its deleted; Pull model



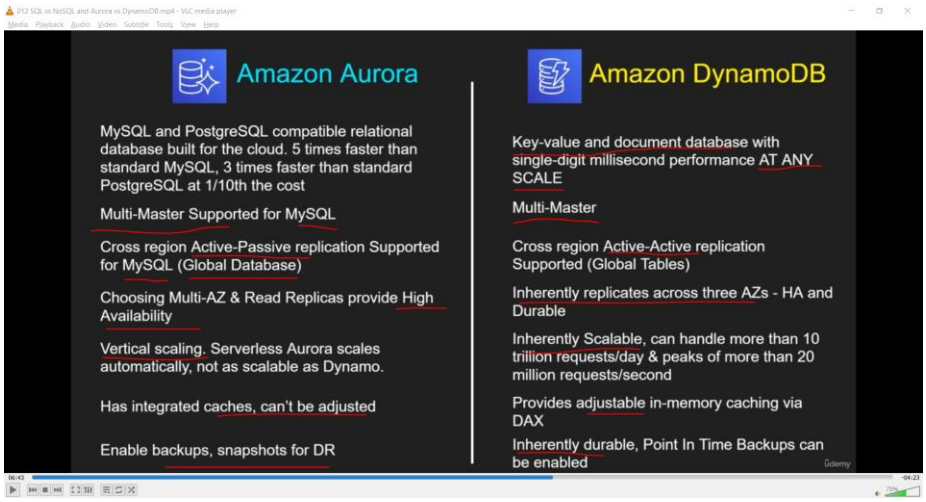
Rabit MQ arch -
Excahnge mechanism 1, direct – routing key matches to binding key and topic is pushed to associated queue ; 2 fanout – exchange forwards to all queues and irrelevant parties ignore the message;

PubSub: Publisher Subscriber arch; push model where publisher publishes a topic which can be pushed to all subscriber to topic ; multiple consumers of same message

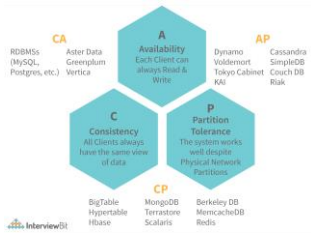
5. Messaging vs streaming -
Messaging: single message, eg. SQS/SNS/eventbridge , gets removed after processing , no need to track the position
Streaming: stream of messages constant flow , 1 event doesn't give the complete picture eg website click stream kinesis / MSK/ amazon streaming service/ Kafka ; stays in stream even after processing , positioning to be tracked for consistency

6.DBs

SQL vs NoSQL
SQL: predefined schema to hold structured data, index joins complex queries supported gracefully, ACID properties atomicity consistency isolation durability , generally vertical scaling eg oracle, db2 msSQL,Aws rds , aws aurora
NoSQL: no predefined schema with unstructured data , not good fit for comple multi table queries, Brewers cap theorem – consiustency availability and partition tolerance, generally horizonatal scaling eg aws dynamoDB, mongoDB,Cassandra

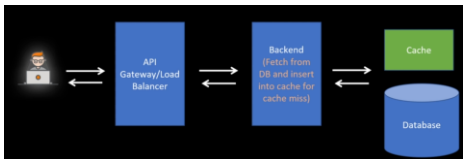


Segway - CAP theorem : a distributed system cannot guarantee C, A and P simultaneously. It can at max provide any 2 of the 3 guarantees. Eg . for a distributed Db system

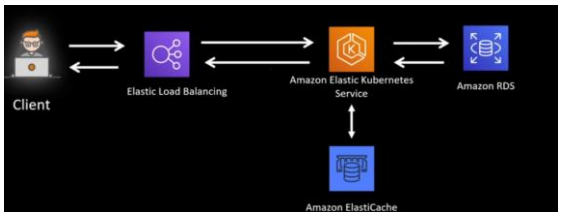


7. Websockets – req res method where only client can invoke server and not vice versa BUT using websocket server can initiate conncetion to client eg Chatbot application whatsapp, telegram , connection remails open for by direction calls

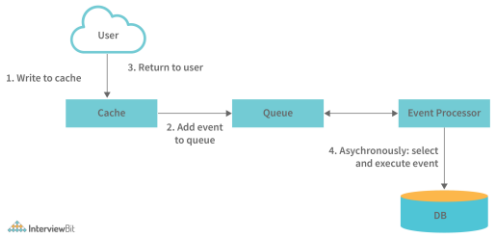
8. Caching – to reduce no. of server calls as backend/ DB are slower to process query



Cache stores a subset of datain memory which has lower latency in contast to Db which is complete and durable. Cache uses TTL time to live parameter to delete cache enteries ; caching can be done in diff parts of system like api gw (aws api cache)/ backend /DB (aws dax dynamo cache)/cloudfront ; if in service not available use cache Db or caching services like amazon elasticache for redis(advanced) or memcached(simpler)



Caching strategies:
Lazy loading (also called cache aside)– user requests data from service which checks if cache hit or miss, if miss it retrieves data from Db and writes cache and returns to user, only requested data is cached ; request time increases if data not cached as store mechanism also; data inconsisitency when if data is already cached and Db is updated
Write through – cache done at time of db interaction/updation
Write-behind (write-back) - Add or update an entry in the cache -> Write the entry into the data store asynchronously for improving the write performance



Refresh-ahead- Using this strategy, we can configure the cache to refresh the cache entry automatically before its

expiration. InterviewBit

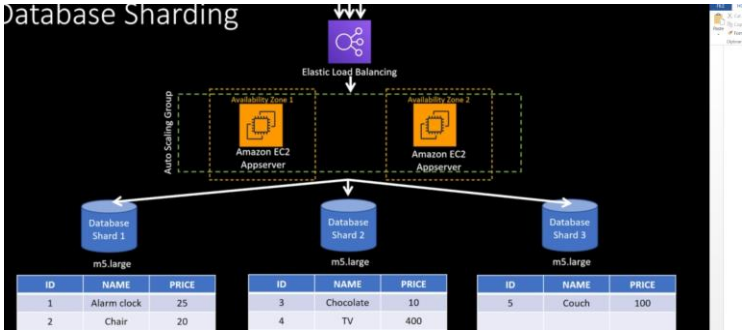
9. HA – system functioning even if some componnets fail by achieveing replication or redundancy in system design; guarentees certain uptime by identifying single point of failure eg. Elastic load balancer (inherently highly available) poiniting to ec2s under autoscaling group and diff availability zones (only auto scaling group gives scalability but not HA) ; aws lambda is inherently HA ; eg 2, Elastic load balancer poiniting to k8 cluster where cluster nodes are in diff availability zones

Data resiliency to achieve 99.99999%(5 9s) availability to avoid single point of failure – active active (using multiple read/w nodes and bidirectional syncup eg postgreSQL or mysql)or active passive arch (using multiple read nodes and unidirectional syncup eg Cassandra db)

10. Fault tolerance arch – HA with no degration of performance even temporarily eg. If need to maintain 100 transaction per second and 2 ec2s defined in 2 AZ each serving 50TPS , if ec2 in 1 AZ goes down , service continues at 50TPS, soln - deploy 3 ec2s in 3 AZ each that can serve 50tps ; fault tolerance is generally expensive that HA

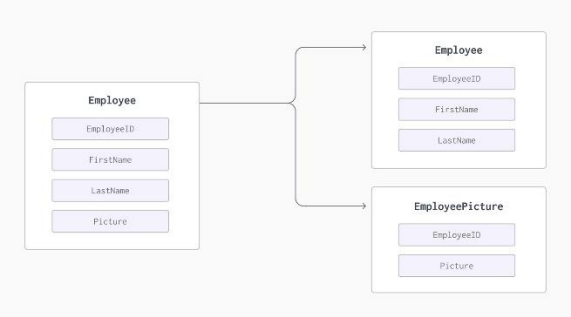
11. Centralised system – client server arch to process all the requests, single point of failure , only vertical scaling, eg. Apps on local machine, ibm db2 on mainframe , appln running non single backend server
Distributed system – multiple appln running on diff servers , horizontal scaling

12. Hashing – Feeding a msg of any length(unique id) into hash function to produce a fixed sized string ; same inout creates same output, fast ; for system design – tables have to be scalable so partitioned into shards/partitioned ; hashing done to decide which partitioned to interact with eg in dynamoDB
issue- if one server(partition) goes down or aserver is added, remapping is done which can create inconsistency ; soln – consistent hashing uses a hash loop supported by server hashing along with request hashing; lb uses consistent hashing
DB sharding – horizontal partitioning for relational DBs; if traffic increases on DB, either vertically scale the DB Or DB sharding – DB arch pattern to create multiple partitions having same schema(columns) but different set of rows(data)



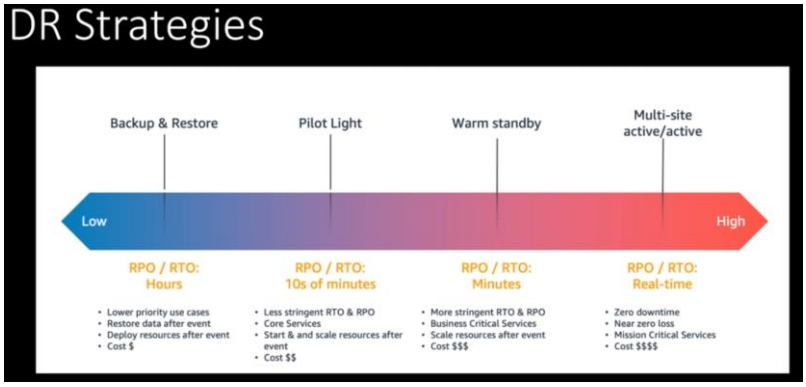
Faster query mechanism, limited blast radius during outage – reliability ; shards may become unbalanced for which reshardigng needs to be done which reduces logic overhead

Sharding vs partitioning - Partitioning is the database process where very large tables are divided into multiple smaller parts. The main goal of partitioning is to aid in maintenance of large tables and to reduce the overall response time to read and load data for particular SQL operations. – Vertical division



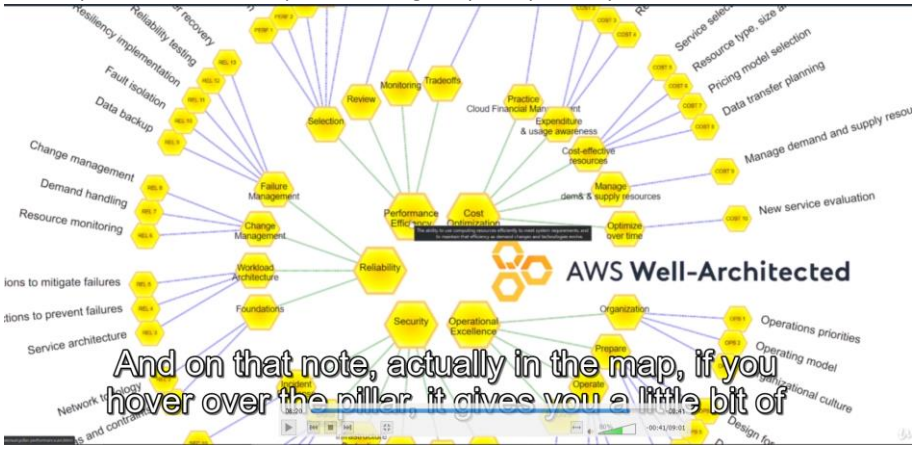
Sharding is horizonatal division also called horizontal partiotining ; both things can be implemented simultaneously for best results

13. DR – approaches dependes on RPO and RTO; RPO – recovery point objective – amount of data allowed to be lost during a disaster measured in time eg. Backup happening every 1 hr , backed up at 1AM and disaster at 1:59AM – can be reduced by taking freq backups but not completely ; Real time RPO can be achieved by storage replication ;;RTO – recovery time objective – amount of time appln can be down in case of disasters

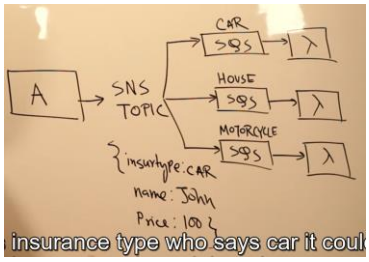


Backup and restore - nothing deployed before hand – not functional DR site
Pilot light – small set of resources already running at DR site which are scaled up at time of disaster ; core components are running but other components needs to be deployed – partially functional DR site
Warm standby – scaled down but fully functional DR site running
Multi site active/active – fully scaled fully functional DR site running

14. 5 pillars of AWS well architected framework - operational excellence , security , reliability , performance , cost optimization –each pillar have dubparts- have to prioritized based on use case – aws well architected tool : set of multi choice questions based on 5 pillars which gives you report on your arch

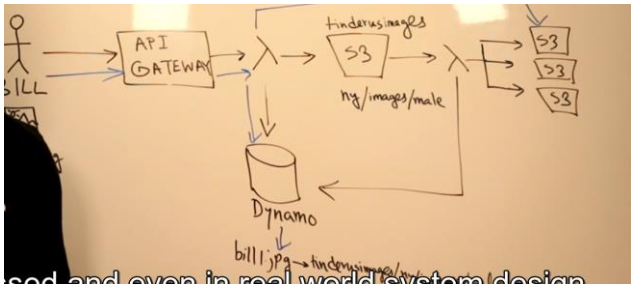


15. Content based message routing system – eg. Insurance company processing car or house or motorbike insurance base don req type
bruteforce – individual apis fro each subtype – not scalable as for any req fail leads to resend complete req from scratch
bit optimise – sqs forwards all req to a lambda which categorises based on type – not have to resend complete req again but lambda becomes bottleneck
ideal solution 1– source publishes the rewquests to sns topic with attached metadata , sns can filter based on metadata and forwards to each sqs which processes the request using lambda fx

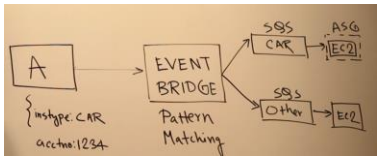


Ideal soln2 – use amazon event bus in place of sns topic in above which filters using same mechanism of categorising based on req body and forwards to each sqs and then to lambda fx – costlier

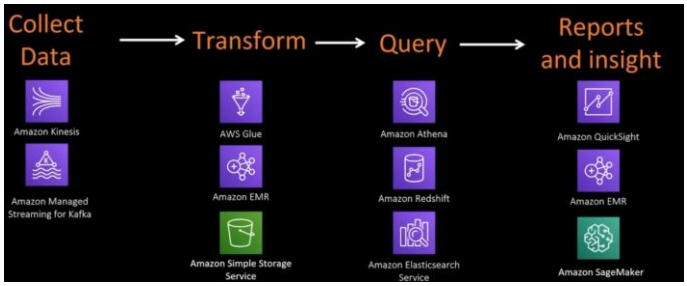
16. image based storage and retrieval arch – tinder/ amazon etc.. - Images can be stored using DB blob storage (disk based or mem based, expensive, faster iops) or object storage like s3(preffered mechanism as cheaper, secure, scalable). Transformation of image also takes place while processing like downgrading quality at time of retrieval etc.
soln for storing img – user calls API gtw with API to put image which calls a lambda fx to trigger s3 upload functionality in a directory structure, lambda also triggers a dynamoDB call in which image name points to path in s3 directory structure where that image is placed for retireval optimization; another lambda can be called to process/categorize images in different quality(downscaled) and save to diffrenet s3s ; retrieval : requester calls the api gtw which forwards the request to lambda to check for position of image in s3 and makes a download call



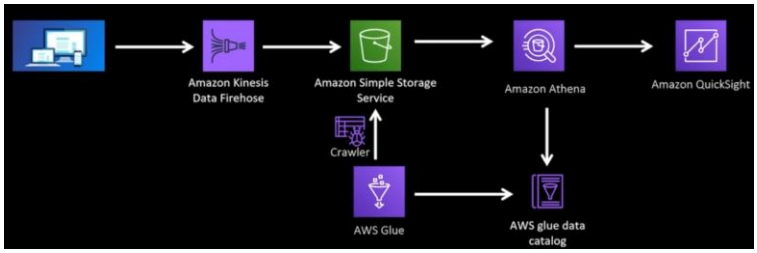
17. High priority queueing messaging architecture – eg. insurance system receiving multiple types of insurance request where car insurance has ighest priority and needs to be processed first
soln – bruteforce : priority based queueing mechanism
optimised : 2 separate sqs mechanism, one for car type requests which forwards message to a autoscaling ec2 group for max performance and other for all other types of request which forwards to a single ec2 instance ; if want to optimise more a event bridge for pattern matching can be placed in between to classify request type



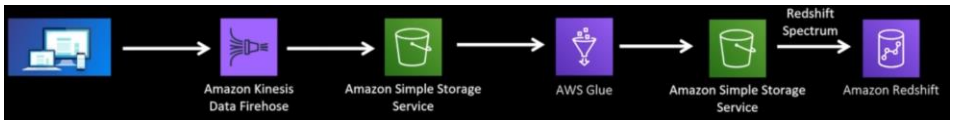
18. Data analytics system design on AWS – Steps:



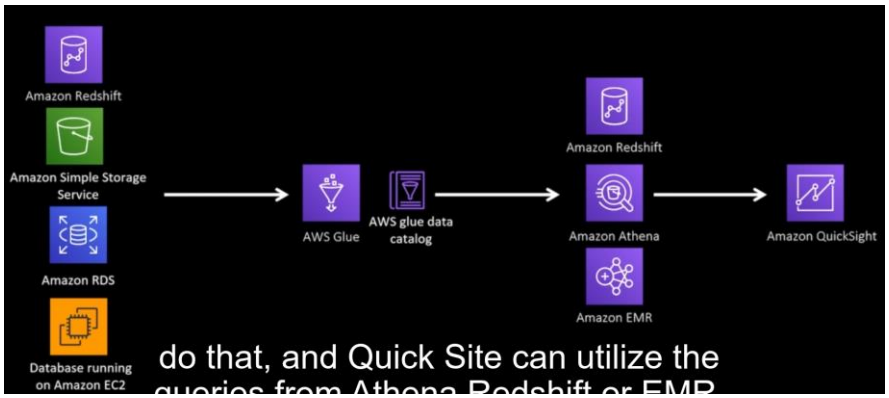
a) query and report on click stream (of a website)- collect data using amazon kinesis data firehose or kafka for stream input and dump to s3, s3 data is unstructured for which aws glue(serverless data integration tool to prepeare and combine data for analytics, ml and analytics, can create etl flow by supporting python,spark, scala) uses crawler based on glue data catalog to identify metadata- defines a structure that can be queried using Athena service and dashboard can be made from amazon quick sight



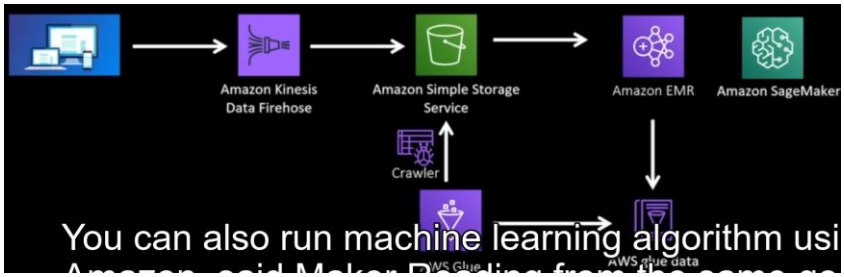
b) etl and data warehouse – extract transform load – collection part same, aws glue reads the data from 1 s3 bucket, transforms the data to change some values and store to another s3 bucket, data warehousing can be done using amazon redshift which works on redshift spectrum that runs sql queries directly on s3 (Athena queries). Alternatively aws glue loads data to redshift tables and use redshift queries



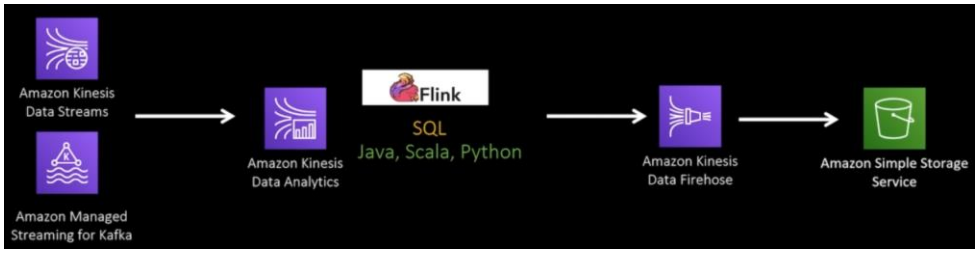
c)unified data across multiple data stores – data from various sources can be transformed using aws glue using glue data catalog which can be queried using redshift / Athena / amazon emr (elastic map reduce , managed big data platform , can runs oien source tools lke apache spark, hide etc. , runs on ec2 or eks) and dashboarding can be done using amazon quicksight



d) big data analysis on click stream data – stream data collected using kinesis stored in s3 which is structured using glue and emr queries the big data to prepare dashboard using sagemaker



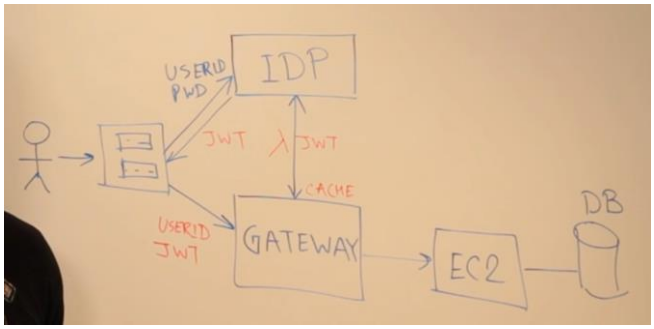
e) instream quering and ETL – analysis done before storing the data to s3 by collecting data from kinesis data stream of kafka and using kinesis data analytics to transform data using flink (sql, java python supported) and send to firehose which dumps to s3



Data lake is superset of all above cases which stores massive amount of data supported by s3

19. Security –

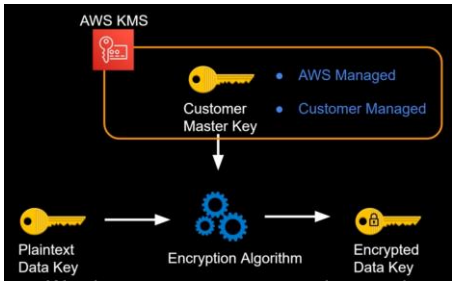
a) authentication(let in or not) and authorisation(accesses or priviledges) :



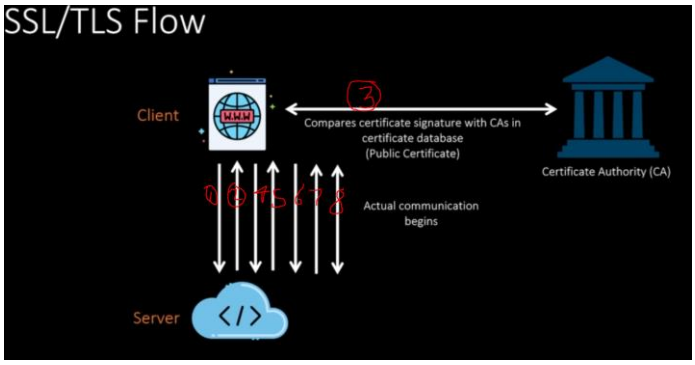
User inputs the credential to a front end screen which forwards it to a IDP(identity provider eg. Active directory) which responses using a jwt token on successful password matching, jwt and user id are forwarded to api gateway which validates jwt by quering it with idp again, next for authorisation a lambda layer can be placed btw api gtw and idp which shares list of policies that are applicable(authorisation) ; cache can be used with set expiry period to skip the gateway->idp call to speed things, if same jwt and userid is again requested it is checked in cache first

b) Encyption ::-
i) At rest :

Client side encryption – client encrypts the file using encryption key and algorithm and then transferred to aws storage - s3/ebs/rds ;Server side encryption – client transfers the raw file over a secured medium(https) and encryption happends in storage ie. 2 parts, in transit and in storage
Encyption key is to be rotated periodically to reduce risk- envelope encryption(manage yourself): loop process of using a new key to encrypt old key using same algo and master key, track to be kept; aws kms: key management service – managed, centralised key mgmt. to track keys ,can be integrated with other aws services where data is at rest – dynamoDB/s3/aurora/elastic search / ebs / redis etc. and needs iam role to access kms

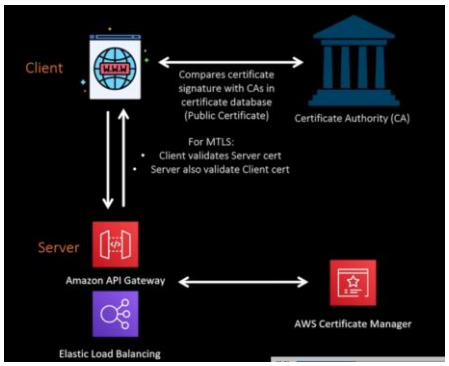


ii) In transit: http was used with info sent in plain text ; https is used in real world in which info is encrypted using either ssl protocol or tls protocol (1 way tls or mutual tls ; faster; also built on ssl)



Steps 1 – client initiate connection to server with info of tls versoin it supports , 2 – server agrees on tl version and sends public key, 3 – client takes public key to CA and compares signature with whats in their DB to say its authorised by CA, 4 – if okay, client uses the public key to send a pre master key, 5- server uses its private key to decrypt pre master key which means both client and master right now uses pre master key to compute a shared scret key, 6 – client shares a encryption text message using shared key 7. Server if able to decrypt it, then shares a mencypted message using shared scret which client decrypts , 8- after this actual communication begins

Mutual TLS – like in single tls (above) only client checks with a CA, but in mutual TLS – server also checks with a CA for client cert – single tls used in thin clients web browsers while mutual tls is used in b2b ;



by default in aws arch, client to api gtw communications is over https with api gtw always using aws certificate manager , commmunicaiton btw apigtw and ec2 is http but can be encyptted using https ; for ntw load balancer ssl/tls is not termninated by default which means https is used for traffic between network load balancer and ec2 but it can be terminated using ssl/tls passthrough feature ; for application load abalncer ssl/tls terminates automatically which means alb to ec2 tarffic is http, https can be enabled

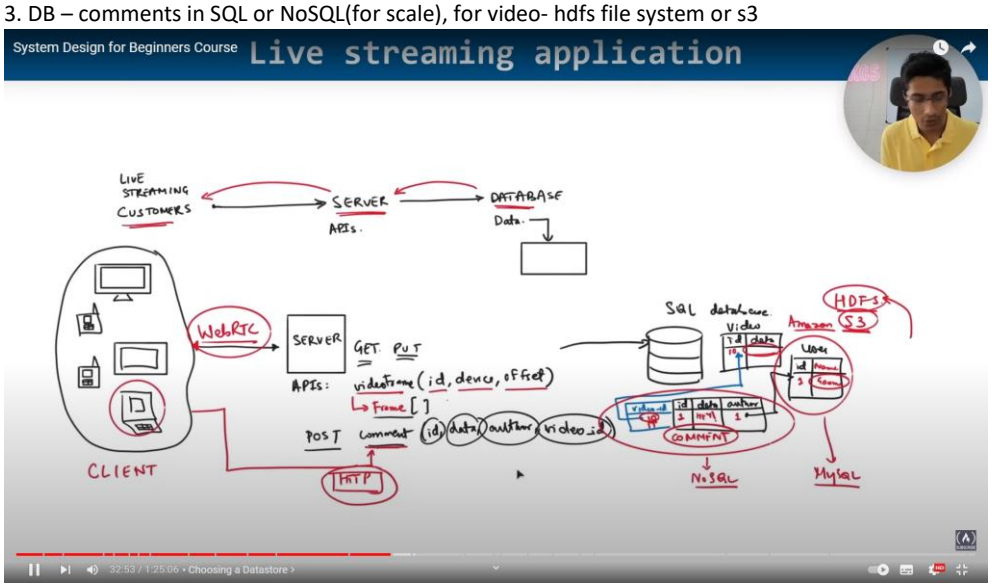
-20. ----------Live streaming system design –

Step 1- define requirement : ability to watch video , ability to handle outage , data duplication based on geography ; Core requirement – streaming video, Additional requireemt - processing video , broadcasting ,fail proof , advertisement , reaction, disclaimer/ news flashes , graceful degradation of videoi quality , multiple device support
Step2 –reduce the features to data definition – likes/comments – to be mapped to objects and later to DB

Step3 – define endpoints to query this data – APIs for each feature
Step4 – Testing of designed system – common cases, edge cases , load test for capacity estimation

High level :

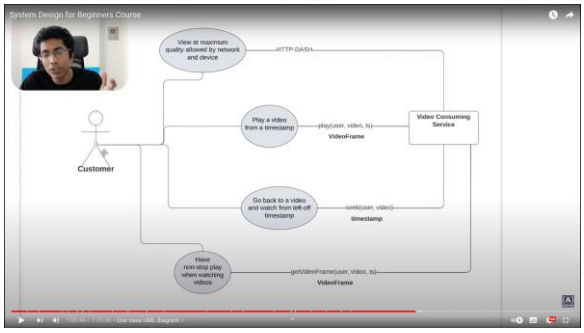
1. API level – get video frame api, post comment api
2. Network – http protocol for comment as stateless(not real time, don't have to store info-server has no context, everything in request from client side or db) , webRTC for video as protocol specially for video, based on TCP as reliable / RTMP – real time media protocol / MPEG dash
3. DB – comments in SQL or NoSQL(for scale), for video- hdfs file system or s3



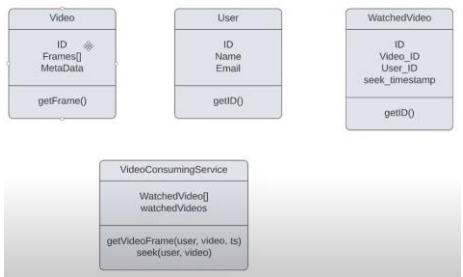
4. Video transformation – raw video source , divided into segments of 10 secs, converted to specific resolution – 144p, 240p, 480p etc. , diff formats for android / apple devices eg h.264 ; map reduce design pattern can be used to take inout from a source and send to diff servers based on req type and receive diff output, convert to data streams form data lake
5. Cache – last 10 sec of video can be cached at server ; cdn can be used for static data for performance

Low Level:

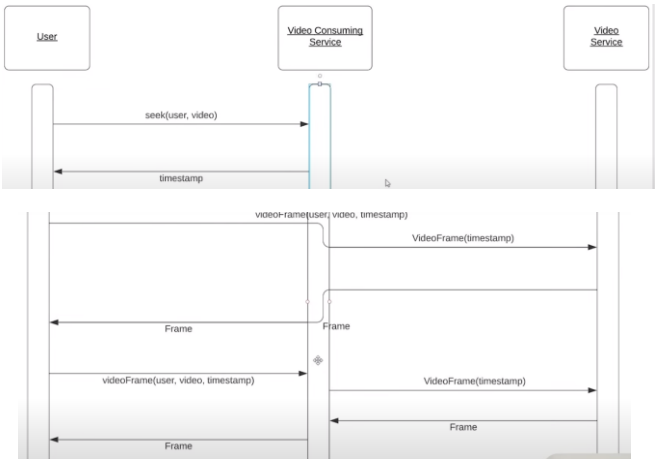
1. use case diagram



2. Class UML diagram

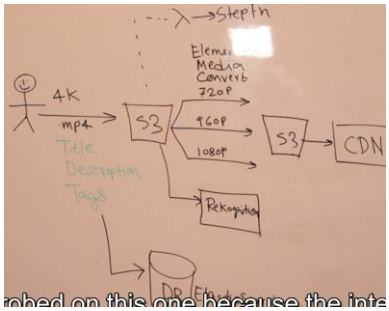


3. Sequence diagram

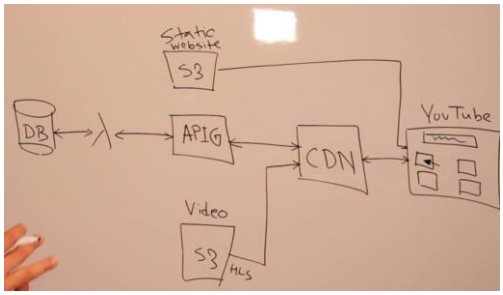


21. Netflix /Youtube/ Amazon prime – on demand video platform

Features considerations - upload , search, view, adult content detection
Design considerations – Scalable, resilient , secure , cost effective
solution – i> videos in mp4 format will be stored in s3 and title,description, tags, s3 directory structure stored in elastic search nosql DB for faster retrieval and search ; ii>video to be encoded in diff resolutions at time of upload so that it doesn't buffer at retrieval will high traffic volume using elementmedia convert service , stored in s3s and linked in elasticsearch, iii>adult content can be filtered using amazon recognition (managed ml model service) and a flag can be updated in DB, all this processing (quality conversion and recognition) can happen in parallel after mp4 upload to s3 by triggering a lambda step function; iv>video to be stored in a cdn for faster access and geographic scalability

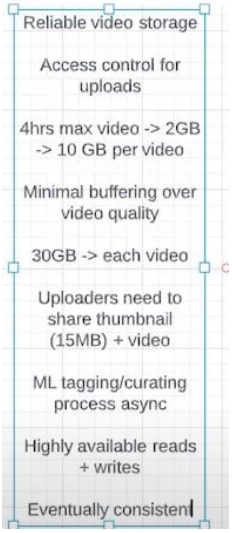


CDN mechanism :website when searched for a video checks cdn, if not present there goes to api gateway to triggere a lambda fx which fetches video address from DB, it is returned to user from s3 and stored in cache for next retrieval, encoded video stored in hls format http live stream in s3 which breaks video into chunks which are delivered sequenctially for optimisation



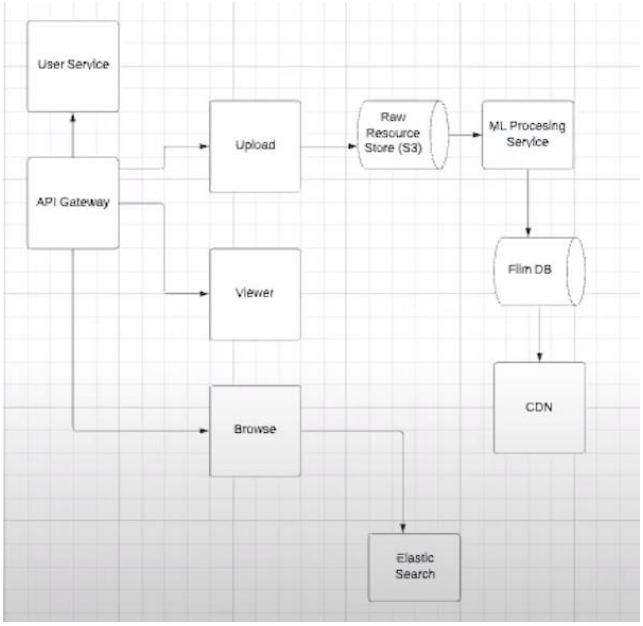
v> cost can be optimised by changing tier of infrequently viewed videos from standard tier to infrequent access tier by setting a s3 lifecycle policy, followed by moving it to glacier ; vi> security – authentication can be implemented using a idp identity access provider mechaniosm, authorisation can be configured using iam roles AWS IAM, encryption for in transit data using https/tls and at rest using kms

Other things – requirements



2gb per hour = max 10 gb video; each video stored in 3 formats = 30gb; trade off between video quality and buffering latency ;; ml used for recommendation engine , eventually consistent used as no req for strong consistency

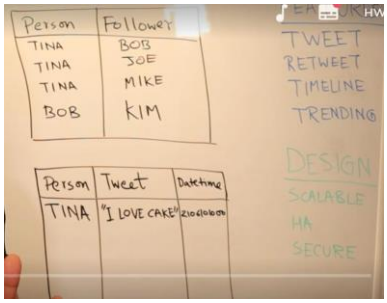
Roles – viewer, uploader , browser



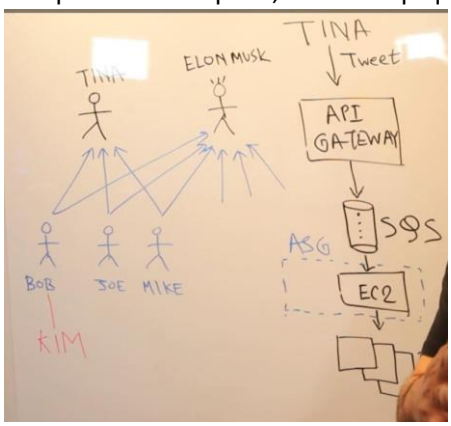
Api gtw architecture with user service separately for authorization and authentication; upload service pointing to s3 bucket which takes content to ml processing to extract info, analyse safety and later list in viewable Db from which it can be cached to cdn ;; browse service using elastic search db to retrieve searched keys fast

22. Twitter

Features – tweet , retweet, timeline , trending
Design – Scalable, HA, Secure
soln – i> follower mechanism – multiple rows as easy to parse without much processing rather than single row with comma separated followers



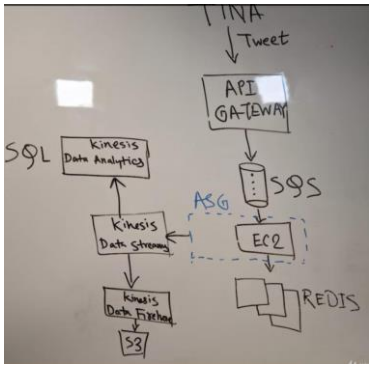
ii> tweet mechanism (precompute mech)– person from follower table, tweet txt and timestamp ; twitter is a read heavy platform where no. of people writing is way less than no. of people reading, so timeline is created at write time(when user tweets something and entry is created in tweet table) rather than read time(when followers login to their account) {1 timeline for 1 follower based on timestamp}; timeline created for each person based on who they follow and only for active followers (last login < 10 hrs) ;tweet is a put api so apigtw, scale so much so async design so retry is not a complete new request, instead sqs queue used and ec2 placed in autoscaling group



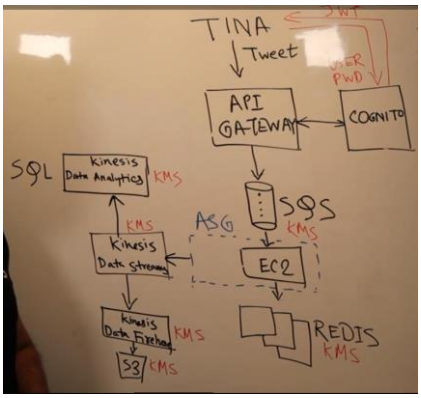
iii> DB issued is mem based (not disk based which is slow) cache DB with 3 replicas (for timeline) – redis is used , cost is expensive as low response time ; DynamoDB can also be used for cost effective design with DAX DynamoDB accelerator (cache)

iv>tweet mech (post compute mech) – if a famous person(elon musk) tweets then all its followers timeline is computed which is very high load in case of millions of followers, in this case post compute mech is adapted where a followers timeline is precomputed normally for someone who is not famous(certain no. of followers condition) and when he logs in and elon musk has tweeted something (less than a timelimit say 24 hrs) , a recomputation of time is done based on already existing timeline and elon's tweet

v>trending mech – bruteforce – check tags or keywords in each tweet and display highest ones (not scalable as millions of tweets) ; optimized – kinesis data stream takes tweets and puts into analytics which can be queried using sql, kinesis firehose used to archive to s3 glacier



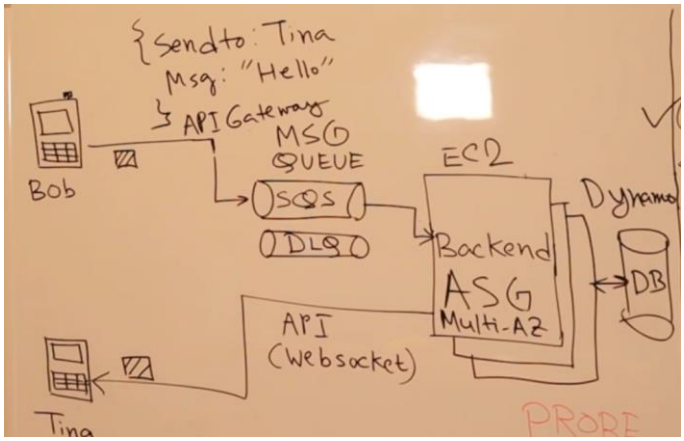
vi> security : auth(n/z) -aws congnito identity provider service using jwt token ; encryption (rest/transit) – kms encryption key , https/tls protocol



23. Whatsapp/telegram/snapchat

Features- one to one messaging, sent /delivered/ read recepits , last online
Design – scalable, resilient , secure

Solution : i> one to one messaging mechanism



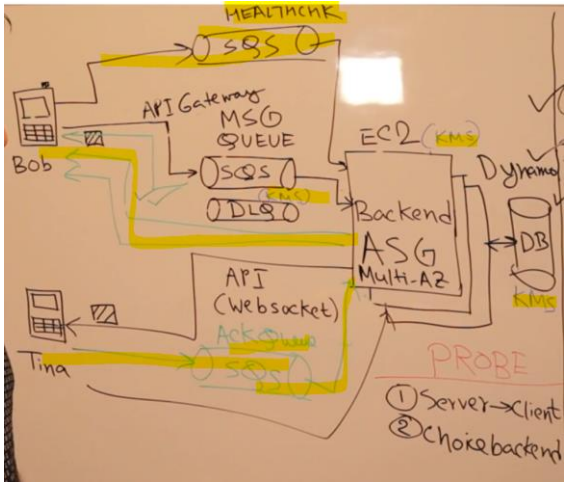
Bob sends a message with json fields having actual message and recipient address to a API gateway(microservice architecture with diff apis for different fucntionality), api is a sync design solution which requires all systems to be scaled together for high traffic and if any one system fails to process req, complete req needs to be retransmitted; there use async design approach using SQS service message queueing mechanism and a DLQ queue (dead letter queue, special type of message queue that temporarily stores messages that a software system cannot process due to errors.) sqs and dlq are inherently scalable and resilient as replicated. Strict ordering can be maintained by FIFO sqs which is not as scalable. Backend can be configured scalable by using autoscaling group based on volume of sqs messages and resilient by using multiple availability zones. Lambda can be used instead of EC2 to reduce idle cost . Load balancer can be used instead of api gateway but it will interact with a compute resource ec2/lambda which will then populate a sqs; api gateway directly integrates with sqs .DynamoDB can be used as scale is required and dynamoDB scales horizontally very well for storing the messages.For next part, Backend calls Tina(Server to client call) using websocket API(bidirectional api where server can also send a message to client using active connection) thorugh API gateway to relay the message.

ii>sent/delivered/read receipts mechanism

sent (single tick) – when apigtw relays message to sqs successfully, it gives ack back to bob
delivered (double tick) – when backend successfully calls the websocket api to tina, it uses another websocket api to send ack back to bob
read receipts (blue tick) – when tina opens whatsapp on her phone, another microservice call is made to ack sqs queue which is relayed to backend and backend uses another websocket api to send ack back to bob <diagram combined after below section >

iii>last online mechanism :distributed system good design is when central backend responsisbilites are distributed to edge devices ; so instead of backend creating schedule to probe each no for last online status, bob calls a healthcheck api every 10 sec which populates another sqs queue which relays message to backend and backend stores in DB, now when tina wants to check bobs online status, she makes another api call to backend , if diff between and time.now() and bob's DB timestamp record is less than 10 sec bob is online, if more show DB record

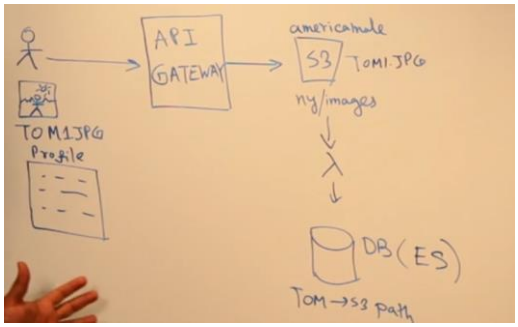
security – in transit : ssl/tls supported at api gtw ; at rest – sqs, dynamoDB, ec2 storage supports kms



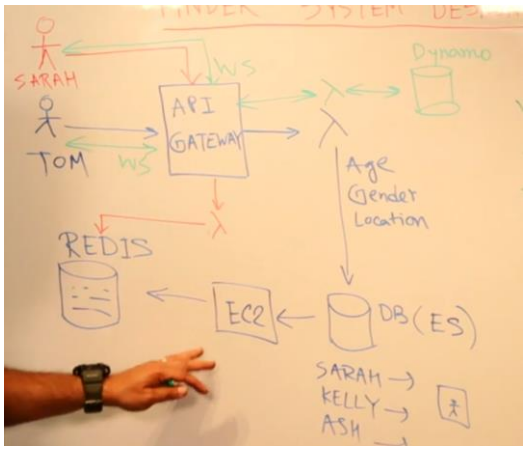
Other things - Make use of push notifications for notifying the members even if they are online.

24. Tinder ;p ::

Features- image store, matcing, recommendation, chat
Design- scalable , ha , cost effective
i> Image store – can store in DB(disk based , expensive) as blob storage or obj storage like s3 (bit slower than disk based but since this is no transactional process its fine) ; tom uploads picture , calls api on api gtw which points to s3 , image is placed at a specific path on s3 bucket which triggeres a lambda fx to store path on elastic search DB for retrieval



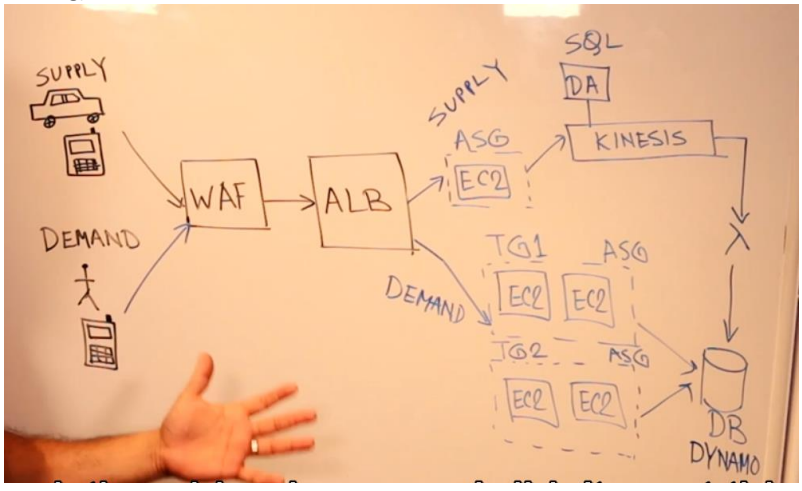
ii> matching – tom logs in which triggers req to api gtw which triggers lambda to search based on age, gender location filters from profiles with info in ES DB, images are pulled from s3 . Performance can be optimised using cache mechanism for frequent users to pre compute their feed by fetching info from DB and compute using ec2 to store in redis, so for toms request a lambda redirects to cache redis DB



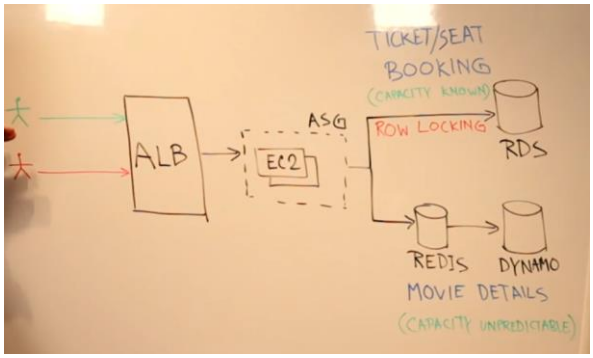
iii> recoomendation – collaborative filtering for next set of profiles (if mukul and Ramesh both likes timmy and Vinnie, and if mukul likes sangeeta , suggest sangeeta to Ramesh as he might like her too) and content filtering for first set of profiles (profile is analysed using NLP to extract keywords which are matched)

iv> chat: websockets apis between tom , Sarah and apigtw ; message stored in DB

25. Uber -
parties – driver, rider
features – location based matching
design - scalability
Uber uses googles s2 library where all the globe is divided into smaller boxes with unique id based on latitude and longitude.Drivers periodically send their location to backend via WAF layer (for security, verifies ip addressees) which is passed to application load balancer whichforwards to a stream via ec2 autoscaling group (as no direct integration ; apigtw has direct integration), stream is used for async design (uber uses kafka but aws's streaming is kinesis) which supports 2 functionality, one for analytics using kinesesis DA data anytics and sql to query ; other is processing location data using lambda and insert into dynamoDB(for horizontal scaling)
for demand side, rider makes a req via waf which passes through alb to a compute layer containing multiple target grps across diff geo loactions(geographical unique id) inside auto scaling group for scalability which fetches supply from dynamoDB and communicated back to rider using websocket response(uber uses ring pop flagship compute layer for this which is a ring of compute resources connected in loop with each vm handling a particular region, using consistent hashing)



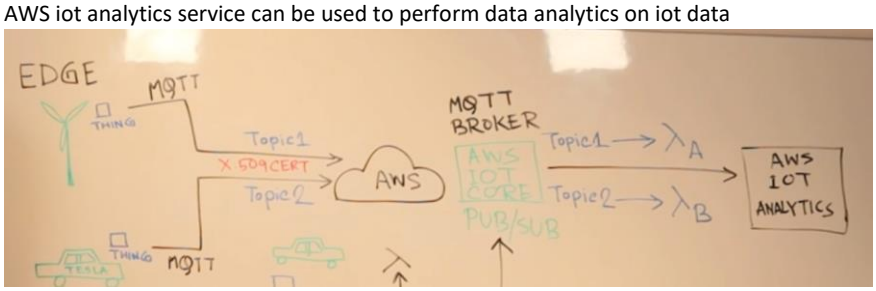
26. ticketing reservation system eg. Book my show



Features – fetching movie details , booking seats (finite no. of available seats)
solution – i> fetching movie details (traffic volume unpredictable) : standard 3 tier arch, requester makes req which passes thorugh alb which forwards to ec2 inside auto scaling group which queries dynamoDB , cache for most searched movies can be maintained in redis
ii> ticket reservation - multiple req should not reserve the same seat, hence locking mmechanism required which is done using relation database (eg. Frontend shows you have 5 min to completethe transaction for which Db row is locked for a user); and since limited seats, scalability is enough vertically as not much traffic

27. IoT architecture

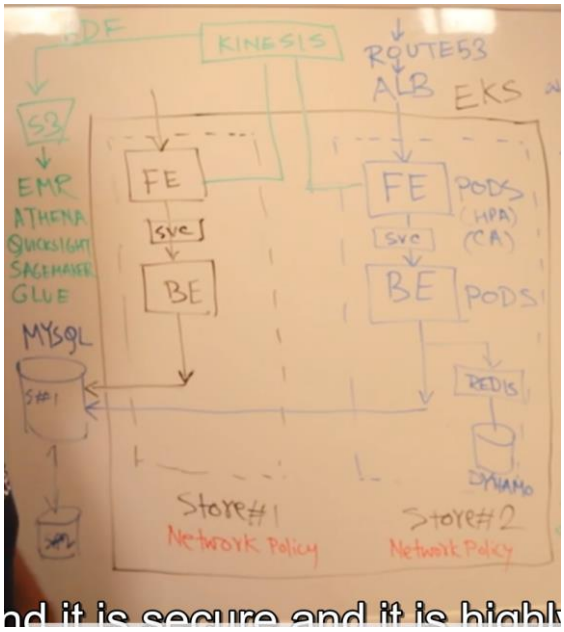
Since iot devices are lightweight(less memory and processing power) themselves, a light weight messssaging protocol required rather than http, there fore mqtt protocol which uses pub sub based messaging where messages are published to diff topics and pushed to backend called mqtt broker like aws iot core which can trigger lambda fx. Authentication works using x.509 certificates which are installed on both mqtt client and mqtt broker. OS for mqtt clients – FreeRTOS (requires 64kb RAM) and IoT greengrass(requires 128 Mb RAM), edge computing is required sometimes to compute solutions on Iot devices rather than cloud backend
AWS iot analytics service can be used to perform data analytics on iot data



28. Shopify – create ecommerce website

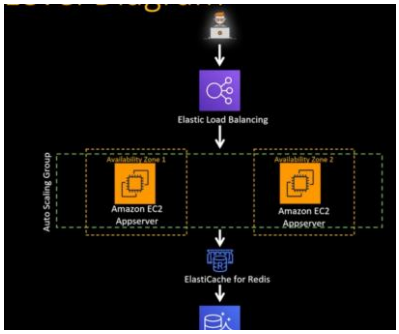
Features- design online store, isolation of customers, analytics
design- scalable, ha, secure, cost effective

Soluition – Shopify uses k8 , so ecr elastic container registry images of front end pods and backend pods which can be deployed using cloudformation when there's a new req for setup of a ecommerce website; user requests website from route 53 dns which routes to application load balancer which redirects to frontend which interacts with backedn via k8 service.
DB can be optimised using caching for storing detail of catalogue implemnneted using redis and dynamoDb; actual transactions can be done on relations db lik mysql which can be common for multiple ecommerece stores as vertical scaling is easily implemented, also sql db is outside of k8 as managed by aws and can be rebalanced based on traffic multiple ecommerce stores can run in same k8 cluster (eks elastic kubernets service) using isolation from k8 namespaces, communication can be isolated using k8 network policies , and data is encypted at source by separate kms keys for relational DB; K8 can be scale using hpa horizontal pod autosclaling and cluster autoscaling
analytics cab be done using kinesis stream which stores data in s3 via kinesis data firehose



29. URL shortner design

Features- character length saving , retrieving , no. of clicks
solution :
Long URL: https://www.google.com
Short url - https://shorturl.at/doTUX saved to amazon aurora DB using primary key “doTUX”
Normal 3 tier arch –elastic load balancer pointing to auto scaling grps ec2s which fetches first form cache redis and then from aurora sql db



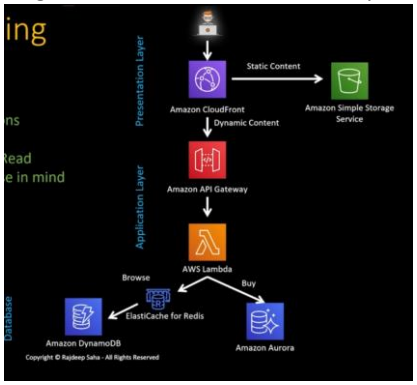
Creation of short url approaches : randomiser(a-z, A-Z, 0-9 which means 26+26+10=62 chars, and for a 5 length primary key 62^5 available options) / hash function (uses hashing algorithm eg. md5) / incremental approach using a counter to generate sequential shorthand , this counter can use amazon emr running apache zookeeper

Other things - Multithreading concept for handling multiple requests at the same time.

30 . Flipkart :D not amazon

Features – product catalog , shopping cart, buy product , product recommendation
design – scalable , ha, secure, cost effective
solution – 3 tier arch using ec2/serverless/k8 with multiple microservices
i> product catalog - no. of people browsing products more than buying product , so nosql use for browsing and relationsal for buying(locking functionality, acid properties), cache used for popular products as well

images store and retrieve functionality from above examples



To manage super high traffic, some microservices can using async arch using sqs

ii> cart – product table – product id, name, price, available count
cart table – card id, product id ,user account id ,

iii> buy product – 3rd party apis are called for integrating 3rd party payment services from lamda fx using authentication, header and body

iv> recommendation - collaborative filtering

31. Consistency patterns : Weak consistency: After a data write, the read request may or may not be able to get the new data. This type of consistency works well in real-time use cases like VoIP, video chat, real-time multiplayer games etc. For example, when we are on a phone call, if we lose network for a few seconds, then we lose information about what was spoken during that time.

Eventual consistency: Post data write, the reads will eventually see the latest data within milliseconds. Here, the data is replicated asynchronously. These are seen in DNS and email systems. This works well in highly available systems.

Strong consistency: After a data write, the subsequent reads will see the latest data. Here, the data is replicated synchronously. This is seen in RDBMS and file systems and are suitable in systems requiring transactions of data

32. Parking garage

Requirement –

Product Requirements

- Need to be able to reserve a parking spot and receive some kind of ticket or receipt

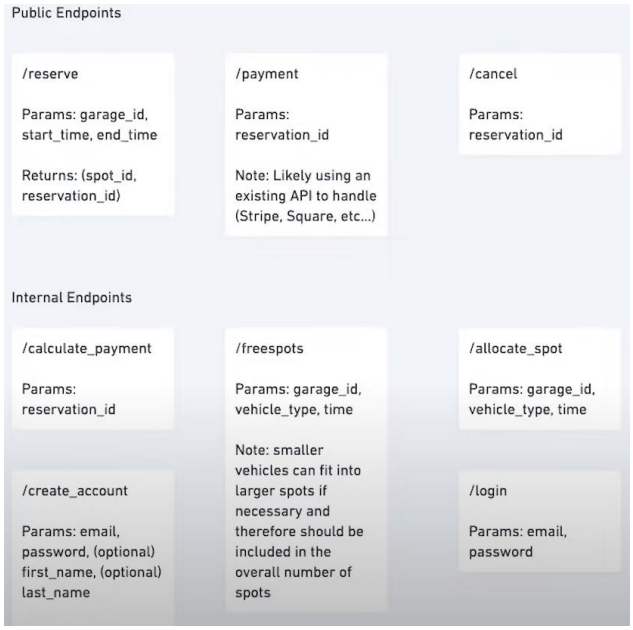
- Need to be able to pay for a parking spot

- System needs to have a high consistency (no two people should be able to reserve the same spot at the same time)

- 3 types of vehicles (compact, regular, and large)

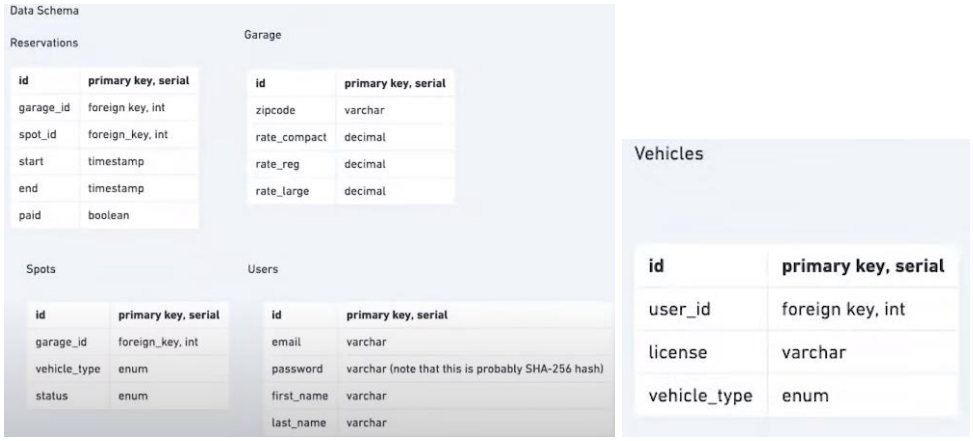
- flat rate based on time, but different rates depending on the type of parking

Endpoints –

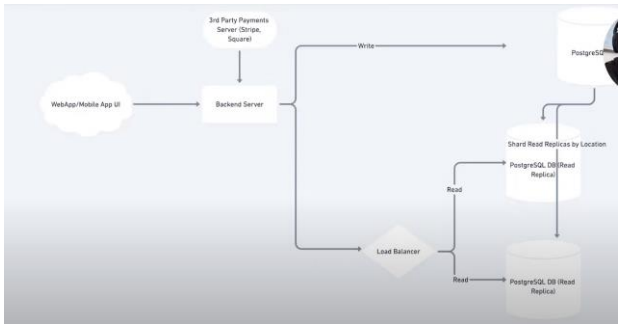


Scale- not so much of a problem as limited parking
consistency – biggest concern as 2 users can't book same parking spot

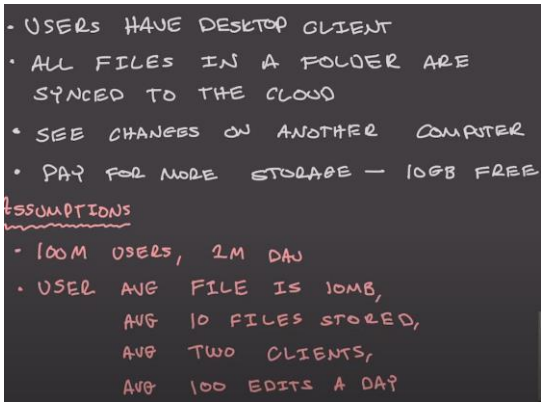
Data types –



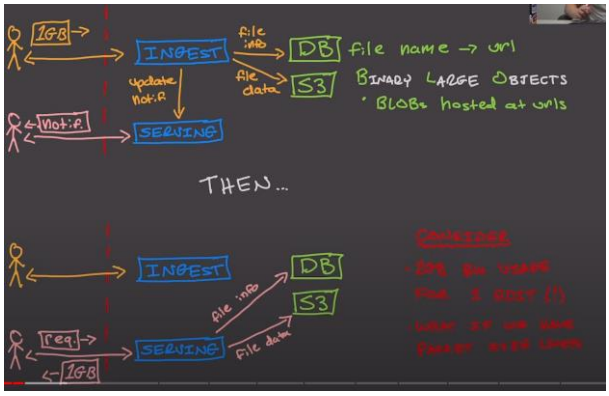
Design – standard 3 tier arch, more reads than write so multple read replicas and a locking mechanism / sharding can be done on zipcode
3rd party payment api integration



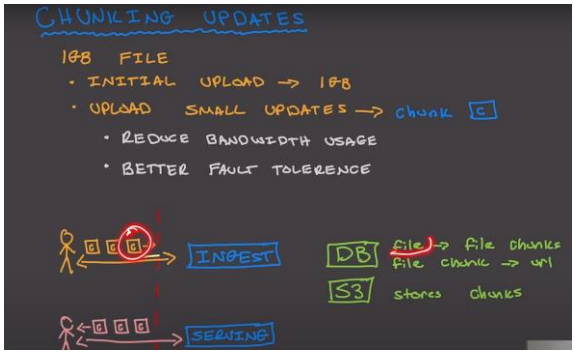
32. Google drive



Dau –daily active users

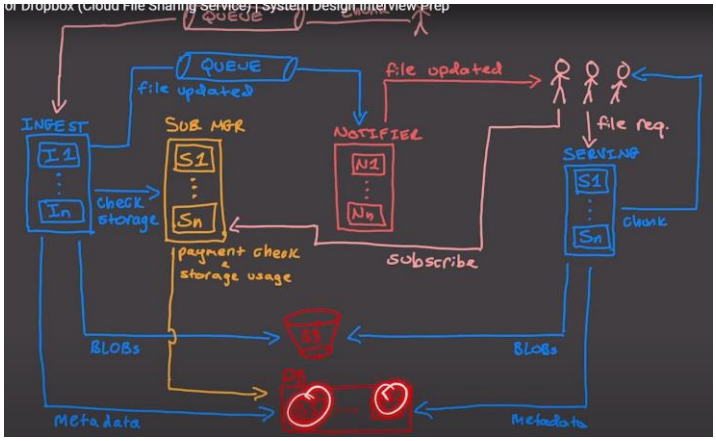


Bruteforce approach – to store file content in s3 and path to file in db ; limitation is if request fails, entire 2 gb bandwidth is wasted for a 1 gb file (in case a new update is to be updated, 1 gb for ingest upload and 1 gb for serving to download file) which is not scalable ; can also fail in case of packet size restriction



Initially 1 gb entire upload; then in case of future edits, only updated chunk ingest

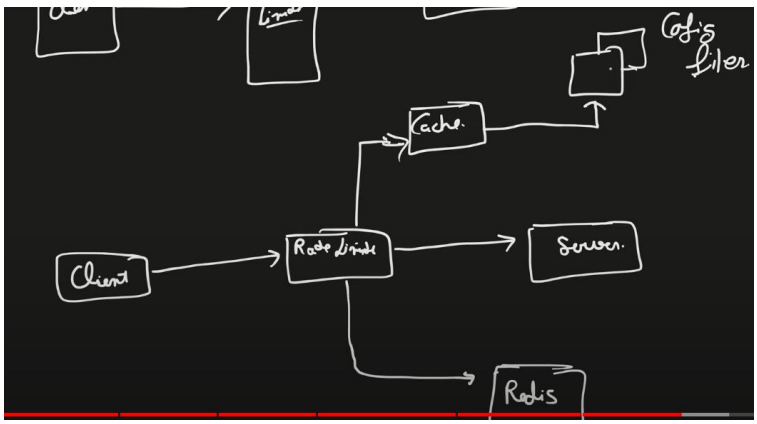
Db optimisation – read replicas, sharding



User upload req in sqs queue for async req which are forwarded to ingest service which checks storage capacity of a particular useraccount from subscription manager based on subscription type , next upload in s3 and metadata stored in DB and later notified to users via notifier service which sends file updated message to users. When a user wants to access file, it requests the serving server which checks path from db and sends the path file in s3 to user back

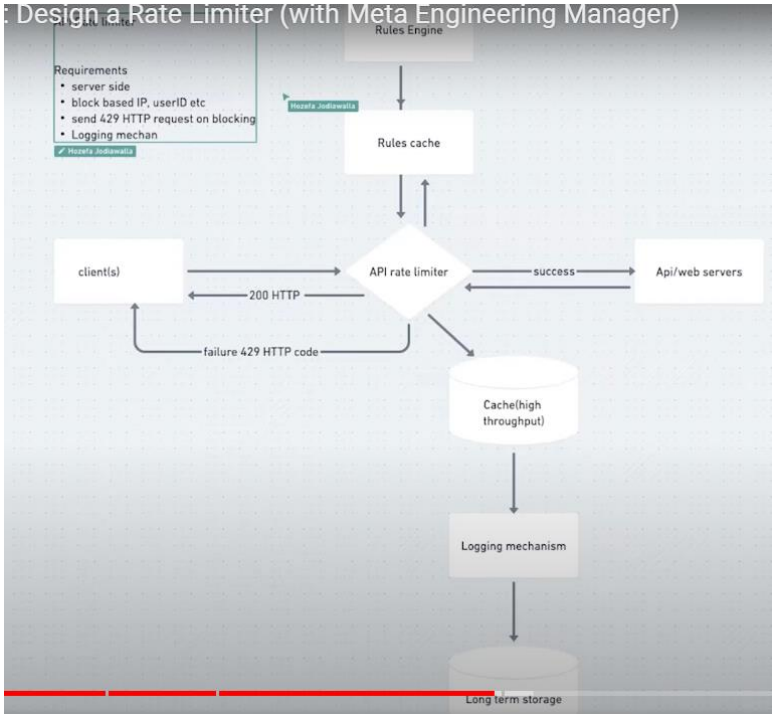
33. Rate limiter – to avoid Ddos attack

Algos: token bucket algo{fixed no. of tokens}, Leaking bucket {req processed at fixed rate using queue}, fixed window counter {capacity divided into windows of time frame which can handle limited no. of req at a time}, sliding window {combines fixed wndow counter and sliding window log}



Rate limiter as an api gtw btw client and server which uses cached configuration file(rule engine) and redis for counter mechanism.

Additional features – logging , long term storage

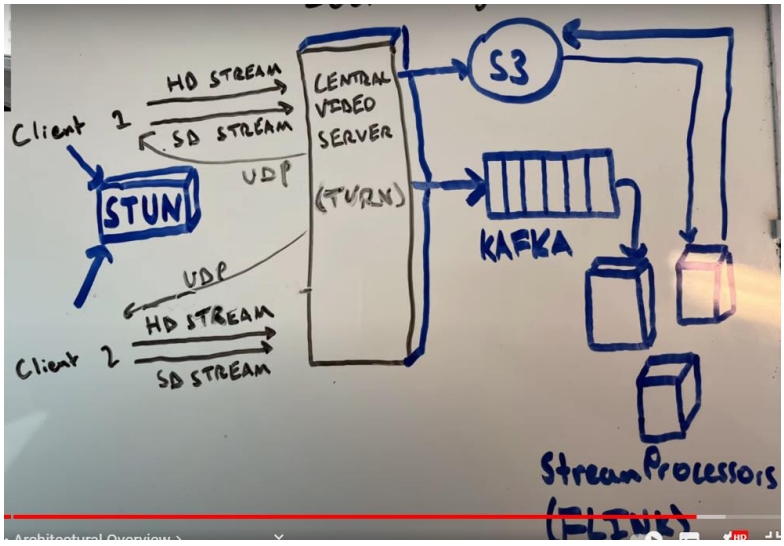


For the distributed system arch, load balancer can be used in between which redirects to multiple instances of backend.

34. Zoom /skype video chat service

Features – real time video and audio chat over internet with low latency, group chat, server side chat recording

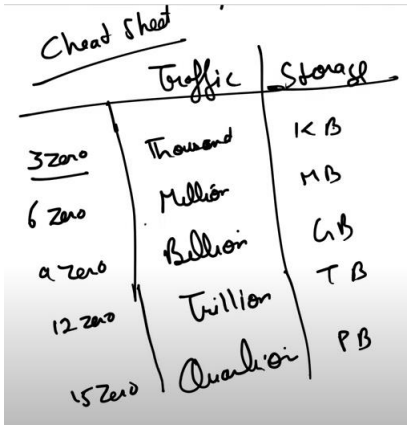
Api – initiate_call(userId_list)
DB - user table , recording table to store s3 path



Stun server – client to client based on public ip address
turn server – central video server to relay network packets
udp protocol used for faster transmission (but less reliability)
clients sending hd stream to central server for high resolution video of speaker on other clients , and

sd stream to central server for low solution video of other non speaking participants just for small thumbnail boxes to other clients - optimization
recording stored in s3, decoupling processing engine(flink -open source apache stream processing framework) from central server which takes input from kafka message queue

35. Capacity planning estimation – no. of servers, ram, storage, trade off CAP theorem



Eg . facebook
traffic = 1 billion, daily actiuve user 25% = 250 million
5read operation + 2 write operation per day per user = 250 *7 / seconds# = 18k query per second

Storage every dau posts 2 times of 250 char each =2*250*2 byte per char =1kb
250 mil user = 250 gb/day
and 10% of user upload a 300kb photo = 7TB/day
if 5yr day needs to be stored = 1825*(250gb+7tb) = 16PB

RAM for each dau, last 5 post are cached = 5*500=2500bytes
250mil*2500bytes = 750GB, or 10 machines with 75 GB RAM each
if 95% requests are completed in 500ms, and each server has 50 threads, no. of servers for 18k requests = 18k/(50*2)=180 machines

C is skipped but A & P is req for fb

36. MISC aws – Place it where it belongs

Q-How to distribute load to multiple nodes for distributed systems(stateless arch – data not stored on server file system, but on DB & user requests)?
A-
1. Pull method – Queues(SQS), stream(Kinesis)
2- Push method – elastic load balancer

Q –automated compute resource initiation
A- 3 ways:
1. Bootstrap- using scripts to copy data to a default EC2- own scripts/chef/puppet/cloudFormation
2. Golden images – snapshot of ec2/rds/volumes and stored as AMI to launch new instances
3- Hybrid – mix of bootstrapping and golden images

Q- Data Replication mechanism for durable data storage
A- 3 ways:
1. Synchronous – write operation synced b/t primary and secondary – where low latency – needs ack
2. Async – decoupled relication – w/o ack mechanism
3. Quoram – mix of sync and async- partial ack mech from limited set of nodes

Q- use case – notifying users about a newsletter
A- setup ec2->goto sns & create topic(public) -> add subscribers -> created s3 bucket -> s3 event mapped with sns -> sync s3 and aws

Q- IAM lifecycle
A-
Principal(user/groups/role/appln service account) -> authentication(confirming principal's identity) -> request(api req or cli req)- > authorisation (match policy to allow/deny request) -> actions (view/edit/delete/create) -> resources(s3/ec2/db)

Q- Type of IAM policies
A- 2 types
1. Managed –i)default policy that is attached to multiple entities(users/grps/roles); ii) aws managed; iii) customer managed
3. Inline – manage your own policy embedded directly into single entity(user/grp/role)

Q- Type of storages
A-
s3- elastic OBJECT storage – max file size 5TB
EBS- elastic BLOCK storage- SSD hard drive with ec2 like c drive/d drive
EFS- elastic FILE storage- almost like EBS but shared, can be accessed by AWS env but EBS can be only accessed by attached ec2

Glacier – archive solution
storage gtw – migrate data from on prem to cloud and maintain a local copy
snowball – 50tb and 82 tb versions – data import and export system to send data to aws -hardware
snowball edge – 100tb
snow mobile – 100,000tb – data center on wheels- actual truck with storage , power and ac – bigger form of snowball

Q- NAT
A- allow instances in private subnet to connect to internet/aws resources but restrict inbout access from internet – have to be places in public internet to get internet – needs elastic ip address- 2 types i) nat gtw – managed service; ii) nat instances

A- lambda run in containers

A- fargate - compute engine in ecs that allows a user to launch containers w/o having to monitor clusters – tasks are launched using fargate – compatible with ecs and eks

A- how to add existing instances to a new autoscaling grp- > ec2 select instance- settings- attach to autoscaling group

Q aws vs gcp vs azure
A-

	AWS	Azure	GCP
Origin	2006, supportive with open source , large and complex scale services	2009, tense relation with open source community, comparatively low quality support	2008, tightly linked open source services, monthly support is quite expensive
Service integration	Simple to combine services – ec2, s3 beanstalk	Combine azure vms , azure app service, sql databases	Cloud engine, clous stotage, cloud sql
AZ	66az with 12 more in pipeline	140 countries 54 regions	20 global areas with 73 zones and 3 more on the way
Compute	Ec2 supporting windows and linux; container services has docket and kubernetes	Vm machines on linux and windows with Microsoft program integration; virtual machine scale sets of azure container services	Compute services with linux and windows support with per second invoicing ; kubernetes engine

storage	S3, ebs, efs, rds, dynamodb	Blob storage, queue storage, disk storage , sql/mysql/postgresql db	Cloud storage, cloud sql, cloud spanner
N/W	Amazon virtual private cloud vpc	Azure virtual network vnet	Cloud virtual network
Market share	32% ; yearly charging with basic instance 69$ per month and largest 3.97$ per hr ;; spot instances, reserved instances and dedicated hosts	19% ; charging per minute basis : smallest 70$ per month, largest 6.79$ per hr ;; developer discount and hybrid benefits	7% ; minute based charging; smallest 52$ per month and largest 5.32$ per hr ;; sustained discount and preemptible discount
Clients	Netflix, airbnb , unilever, bmw, Samsung, mi	Johnsons control, polycom, Fujifilm, hp, Honeywell, apple	Hsbc, paypal, 20th century fox, Bloomberg, dominos
Main pluses	Mature, dominates cloud domain with features such as configuration, security, auto scaling ; better offering for enterprise friendly services , more open source tools and global reach	Reliable to integrate with Microsoft tools, best for development and testing tools, also provides hybrid cloud	Expertise in devops , flexible discounts and contracts,
Main minuses	More expensive	Less efficient mgmt. tooling , less enterprise ready	Less data centers and fewer services