AWS certified developer associate (DVA-C02)

→ AWS Regions :-
  ↳ AWS Regions all around the world

→ How to choose AWS region :-
  ↳ Compliance with data governence & legal req
  ↳ Proximity to customers
  ↳ Availibility ~~zones~~ services within a ~~region~~
  ↳ Pricing

→ Each region has many availibility zones (min - 3 / max - 6)

→ 400 + point of presence
  30 + Regional Caches in 90 + countries

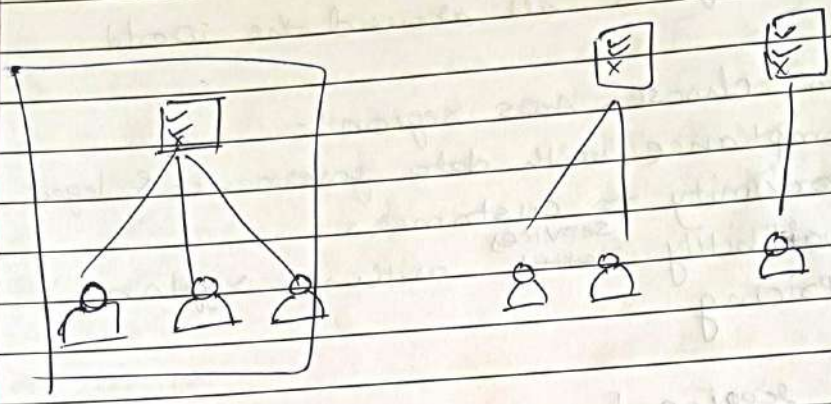
→ IAM : Users & Groups

→ IAM :- Identity and Access Management, Global service
→ Root account :- created by default, shouldn't be used or shared.
→ Users are people within your organization, and can be grouped.
→ Groups only contains users, not other groups.
→ User don't have to belong to a group and user can belong to multiple groups.

→ Users or Groups can be assigned JSON daument called policies.

→ Policies define permissions

→ IAM polices Inheritance



→ IAM. Policy structure

→ consist of
  ↳ version
  ↳ Id
  ↳ statement
    ↳ Sid
    ↳ Effect
    ↳ principal
    ↳ Action
    ↳ Condition

→ To access Aws there is three option
  ↳ Aws Management console
  ↳ Aws Command Line Interface
  ↳ Aws Software development kit

→ AWS CLI

→ Interact with AWS services using commands in your CLI shell.

→ Direct access to public APIs of AWS services

⇒ AWS SDK

→ AWS software development kit

→ Enable access and manage AWS services programmatically

→ Support:- JS, Python, .Net, Ruby and etc.

→ Mobile SDKs (Android, IOS)

→ IAM Roles for services

→ Some AWS services will need to perform actions on your behalf.

→ So we assign permission to AWS services with IAM Roles.

→ Common roles:- EC2 Instance Roles, Lambda function Roles and Roles of cloudformation.

→ IAM Security Tool

↳ IAM Credentials Report (account level)

↳ IAM Access Advisor (user-level)

→ EC2 sizing & configuration option

↳ Os (Mac, linux & windows)
↳ Computer power & core (CPU)
↳ random-access Memory (RAM)
↳ Network - Attached (EBS & EFS)
↳ Hardware attached (EC2 Instance Store)
→ Network card
→ Firewall rules
→ Bootstrap strip

→ Info Instance type ! example
↳ t2.micro, t2.xlarge, c5d.4Xlarge.....

→ t2.micro is free on AWS free tier (750 per month)

→ m5.2Xlarge
↳ m → instance class
↳ 5 → generation
↳ 2Xlarge → size within an instance class.

→ Security Groups

→ security grps are fundamental of network security in AWS.
→ control in and out traffic of our EC2 Instances.
→ It contain allow rules.
→ rules can be refered by IP or security grps.

→ can attached to Multiple Instances.

→ Classic ports

→ 22 → SSH
→ 21 → FTP
→ 22 → SFTP
→ 80 → HTTP
→ 443 → HTTPS
→ 3389 → RDP

→ Ec2 Instances purchasing option

① On demand Instance :- short workload, predictable pricing, pay by second.

② Reserved (1 & 3) years
→ Reserved Instance → long workloads
→ Convertible Reserved Instance → long workload with flexible instance

③ Saving plan :- (1 & 3 years plan) → commitment to an amount of usage, long workloads

④ Spot Instances → short workloads, cheap, can lose instance

⑤ Dedicated Hosts :- book an entire physical server, control Instance placement.

⑥ Dedicated Instances :- no other customers will share your hardware.

⑦ Capacity Reservation - reserve capacity in a Specific AZ for any duration.

* → EC2 on demand

→ pay for what you use
→ Has the highest cost but no upfront payment
→ No long-term commitment

→ Recommended for short-term and un-interrupted workloads.

* → EC2 Reserved Instances

→ 72% discount to on-demand
→ you reserve specific instance attributes.
→ Reservation period → 1 year (+ discount) or 3 years (+++ discount)
→ payment option (No-upfront (+) / partial Upfront (++) and All Upfront (+++)
→ Recommended: for steady state usage
→ buy and sell in the Reserved Instance marketplace.

* → Convertible Reserve Instance
→ can change EC2 Instance type, instance family, os.
→ Upto 66% discount.

* → EC2 saving plans

→ Get a discount on long term usage
→ commit to some certain type of usage
→ Usage beyond EC2 saving plan is billed at the on-demand price.

* → EC2 Spot Instances

→ can get upto 90% compared to on-demand
→ Instance can lose at any time
→ Most cost-effective instance in AWS.
→ Not suitable for critical jobs or databases.

* → EC2 dedicated Hosts

→ A physical server with EC2 instance capacity fully dedicated to us.
→ Allow you to address compliance requirement and use your existing server-bound s/w licenses.
→ purchase option → on demand, reserved
→ most expensive
→ useful for s/w that have complicated licensing model (BYOL)

* EBS Volume :-

→ EBS (Elastic Block Store) Volume is a network drive you can attach to your instance while they run.
→ Allow to persist data, even after termination.
→ only be mounted to one instance at a time.
→ bound to specific availability zone.
→ network USB stick.
→ use network to communicate.
→ quickly attach to another.
→ locked to availability zone.

* EBS Snapshots

→ Make a backup (snapshot) of your EBS volume at a point in time
→ Not necessary to detach volume to do snapshot but recommended.
→ can copy snapshots across AZ or region.
→ EBS Snapshot archive
→ Recyclebin for EBS
→ Fast Snapshot restore.

* AMI (Amazon machine Image)

→ AMI are a customization of an EC2 instance.
→ you can add, own s/w, config, OS, monitoring...
→ AMI is build for specific region (copied across other

→ You can launch EC2 instance from:-
  ↳ A public AMI → Aws provided
  ↳ Your own AMI → own created
  ↳ An Aws Marketplace AMI → Someone else made.

* AMI process
  Start an EC2 instance and customize it
  ↓
  Stop the instance (data integrity)
  ↓
  Build an AMI (create EBS snapshots)
  ↓
  Launch instance from other AMIs

* EC2 Instance Store

→ EBS volume are network drive with good limited performance.
→ If you need a high-performance hardware disk use EC2 Instance Store
  ↳ Better I/O performance
  ↳ Good for buffer / cache / scratch / temporary content
  ↳ Risk of data loss if hardware fails.
  ↳ Backup and replication are your responsibility.

* EBS Volume Types

EBS Volumes comes in 6 types.
↳ gp2 | gp3 (SSD)
↳ io1 | io2 Block Express (SSD
↳ st1 (HDD ~~SSD~~)
↳ sc1 (HDD)

* Amazon EFS - Elastic file System

→ Use cases :- content Management, web serving, data sharing, wordpress.
→ Use NFSv4.1 protocol
→ Uses security group to control access to EFS. ~~NFS~~.
→ Compatible with Linux based AMI (not window)
→ Encryption at rest using KMS.
→ scale automatically, pay-per-use, no capability planning

* Load balancer

↳ Load balancers are servers that forward traffic to multiple servers downstream

→ Elastic Load Balancer

It integrated with many services
↳ EC2, EC2 auto scaling, Amazon ECS
↳ AWS Certificate Manager (ACM), cloudwatch
↳ Route 53, AWS WAF, AWS Global Accelerator

→ Health check are crucial for Load balancer.

→ Types of Load balancer
  ↳ classic Load balancer
    ↳ Application Load balancer
  ↳ Network Load balancer
    ↳ Gateway Load balancer.

* Application Load Balancer (ALV2)
→ Application Load Balancer is layer 7 (HTTP)
→ Load balancing to multiple HTTP application across machines
→ Load balancing to multiple applications on the same machine
→ Support redirects
→ best for micro-services & container based App^n

* Network Load Balancer
  ↳ work at Layer 4
  ↳ Forward TCP & UDP traffic to your Instance
  ↳ Handle millions of Request per second.
  ↳ NLP has 1 static IP / AZ
→ NLP used for extreme performance TCP or UDP traffic

8510087705

* Gateway Load Balancer

→ Deploy, Scale and manage a fleet of 3rd party network virtual application appliances in AWS.

→ Ex:- Firewall, IDS, IPS, Deep packet Inspection systems, Payload manipulation.

→ Operate at layer 3.
→ Combine the following function:-
  ↳ Transparent network gateway
  ↳ Load Balancer.
→ uses the GENEVE protocol on port 6081.

* Sticky Sessions

→ It is possible to implement stickiness so that the same Client is always redirected to the same instance behind a load balancer.

→ works for all the Load balancer.
→ make sure that the user doesn't lose his session data.
→ Types
  ↳ Application based cookies
  ↳ Duration based cookies

* Load Balancer
cross zone load balancer
Each load balancer instance distributed even to all the registered instances in AZ.

→ SSL/TLS

→ An SSL certificate allows traffic between your clients and your load balancer to be encrypted in transit.

→ SSL → Secure Sockets Layer

→ TLS → Transport Layer Security

→ SNI :-

→ SNI helps to solve problem to load multiple SSL certificates onto one web server

→ only works with ALB & NLB, Cloudfront

* Auto Scaling Group

→ manage load of server

→ Scale out (add EC2 Instance) to match increase load

→ Scale in (remove EC2 Instance)

→ Ensure min & Max no of EC2 Instances.

→ Automatically register new instance to load balancer.

→ Recreate automatically.

→ Free

* Amazon RDS Overview

→ RDS → Relational database service
→ It's managed DB services for DB use SQL as a query language.
→ allow to create db in the cloud allowed by AWS
  ↳ Postgres, MS@MySQL, MariaDB, Oracle, Microsoft SQL server, Aurora (AWS)

→ RDS also provide other services
  ↳ Automated provisioning, OS patching, continuous backup and restore at specific time stamp.
→ Monitoring dashboard, read replicas, Multi AZ.
→ Maintaince windows, scaling capability,
→ backed by EBS.

* RDS Read Replicas for read scalability

→ upto 15 read replicas
→ within AZ, cross AZ or cross Region.
→ Replication is ASYNC, so reads are eventually consistant
→ Replicas is promoted to their own DB.
→ App^n must update the connection string to leverage read replicas.

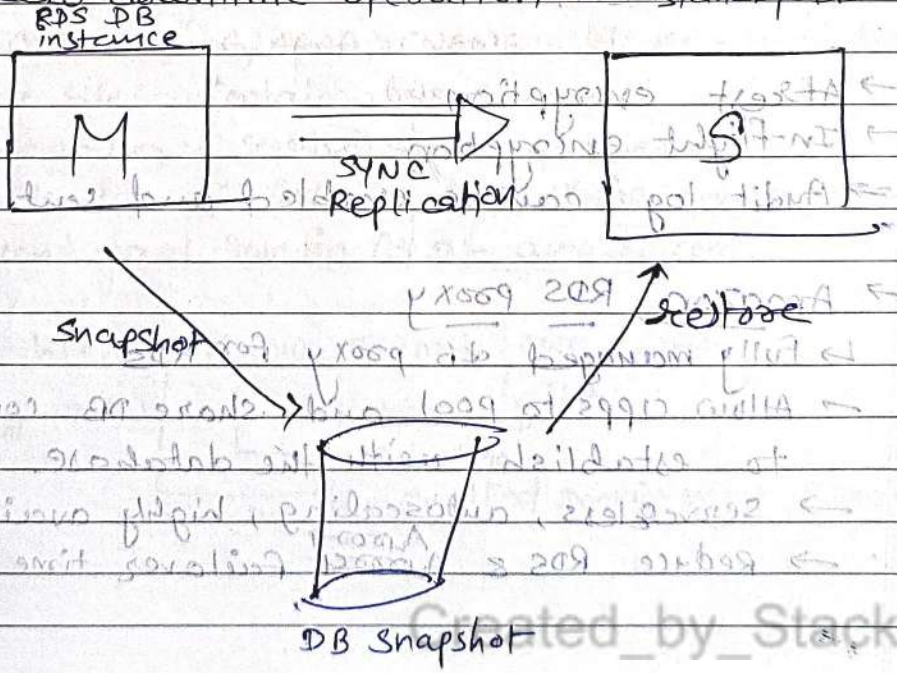
Use case:-
→ you have a production db that is taking on normal load
→ you want to run reporting application to run some analystics.
→ you create a new read replica to run the new workload there.

→ Production workload unaffected
→ Used only select query

→ Cost:-	
→ there is no cost when read replicas within same region	→ there is a cost when read replicas happen at cross-region.

⇒ RDS Multi AZ

→ SYNC replication
→ one DNS name - automatic app failover to standby
→ Increase availability
→ No manual intervention in apps.
→ No used for scaling.
→ Single AZ to multi AZ
→ Zero downtime operation



* Amazon Aurora

→ Aurora is a proprietary technology from Aws.
→ ~~postgreSQL~~ postgres and MySQL are both supported as AuroraDB
→ Aurora is Aws cloud optimized
  ↳ Claims 5x performance improvement over MySQL
  ↳ 3x postgres on RDS
→ Aurora storage automatically grow in increment of 10GB, upto 128 TB.
→ Aurora can have upto 15 replicas and it faster.
→ Failover in Aurora is instaneous.
→ cost more - but more effective

↳ Features of Aurora
  ↳ Automatic failover, Isolation and security
  ↳ Backup & Recovery, Industry compliance
  ↳ Push-button scaling, Advance Monitoring
  ↳ Backtrack.

→ At-rest encryption
→ In-flight encryption
→ Audit log can be enabled and sent to Cloudtrail

→ Amazon RDS proxy
  ↳ Fully managed db proxy for RDS
→ Allow apps to pool and share DB connection to establish with the database.
→ Serverless, autoscaling, highly available
→ Reduce RDS & Aurora failover time by upto 66%

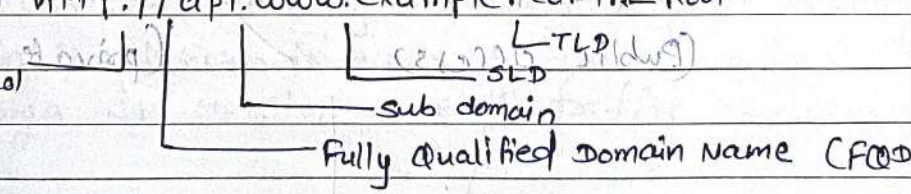
* Amazon Elasticache

→ Same way RDS is to get managed Relational Database.
→ Elasticache is to get managed Redis or Memcached.
→ in memory database help in high performance low latency.
→ Help reduce load off db for read intensive
→ make app^n stateless.
→ take care of OS maintainance / patching, optimization, setup config, monitoring, failure recovery
→ need - heavy app code changes.

* what is DNS

→ Domain name System which translate the human friendly hostname into the machine IP addresses.
→ backbone of the internet
→ Domain Register:- Amazon Route 53, GoDaddy
DNS record:- A, AAAA, CNAME
Zone file:- Contain DNS record
Name Server:- resolve DNS queries.
Top level Domain (TLD):- .com, .us, .in, .gov, .org
second level Domain (SLD):- amazon.com.

http://api.www.example.com. ← Root



* Route 53

→ A highly available, scalable, fully managed and Authoritative DNS
→ Route 53 is also known as Domain Register.
→ Ability to check the health of your resources
→ The only AWS service which provides 100% availability SLA

→ Route 53 - Records :-
  ↳ Domain / subdomin Name :- eg example.com
  ↳ Record Type :- eg. A or AAAA
  ↳ Value
  ↳ Routing policy
  ↳ TTL

→ DNS type :- A / AAAA / CNAME / NS
  → A - maps a hostname to IPV4
  AAAA - maps a hostname to IP V6
  CNAME :- maps a hostname to another hostname
  → NS :- Name Servers for the Hosted Zone.

→ Hosted zones :-

Public Hosted zone (Public access) | Private hosted Zone (private access)

Route 53 - Records & TTL

↳ High TTL (24 hr)
  ↳ Less traffic
  ↳ Possibly outdated records
↳ Low TTL (60 sec)
  ↳ More traffic on Route 53
  ↳ Records are outdated for less time
  ↳ Easy to change records

→ Mandatory

* Routing policies - Simple
→ Typically, route traffic to single resources.
→ Can specify multiple value in same record.
→ If multiple values are returned, a random one is chosen by the client.
→ When alias enabled, specify only one AWS resource

* Routing Policies - Weighted
→ Control the % of the requests that go to each specific resource
→ DNS records must have same name and type.
→ associated with health checks.
→ Use cases:- balancing between region, testing new application versions.
→ Assign 0 to stop sending traffic

* Routing policies – Latency-based

→ Redirect to the resources that has the least latency close to us.
→ Super helpful when latency for user is a priority.
→ Latency is based on traffic between user and AWS Regions.
→ Have associated with health check.

* Route 53 - Health Checks

→ HTTP health checks are only for public resources
→ Health checks ⇒ Automated DNS Failover:-
① Health checks that monitor an endpoint (app, server, AWS resources)
② Health Checks that monitor other health checks
③ Health checks that monitor CloudWatch Alarms

→ Health checks are integrated with CW metrics.
→ About 15 global health checkers will check the endpoint health
Threshold - 3 (default)
Interval - 30 sec
Supported protocol:- HTTP, HTTPs and TCP
If > 18% → healthy
→ Configure your route / firewall to allow incoming requests from Route 53 Health checkers.

→ Route 53 - Calculated Health checks

→ combine the results of multiple Health checks into single health check.
→ you can use OR, AND or NOT
→ can monitor upto 256 child Health checks
→ Specify how many of the health checks need to pass to make the parent pass.
→ perform maintenance to your website without causing all health checks to fail.

→ Health checks - private Hosted zones
→ health checks outside the VPC
→ They cannot access private endpoint
→ you can create cloudwatch metrics and associate a cloudwatch Alarm then create Health check that checks the alarm itself.

* Routing policy - failover

* Routing policies - Geolocation
→ Different from latency based
→ The routing is based on user location.
→ Specify the location by continent, country or by US state
→ should create a default record (if not available spec)
→ Use case :- website localization, restrict content distribution, load balancing.

* Geoproximity Routing policy

→ Route traffic to your resources based on the geographic location of user and resources.
→ Ability to shift more traffic to resource based on the defined bias.
→ To change the size
To extend - (1 to 99) extend traffic
To shrink - (-1 to -99) shrink traffic

→ we use Route 53 traffic flow.

* Routing policies - IP-based Routing

→ Routing is based on clients IP addresses
→ You provide a list of CIDR's for your clients and corresponding endpoint location.
→ Use cost :- Optimize performance, reduce cost

→ Routing policies - Multi Value
→ Use when routing traffic to multiple resources
→ Route 53 return multiple values
→ use & healthy records are returned for each multi value query.
→ Multi value is not substitute for having an ELB.

* VPC & Subnets primer

→ VPC :- private network to deploy your resources
→ Subnets :- allow you to partition your network inside your VPC
→ A public subnet is a subnet that is accessible from the internet
→ A private subnet is a subnet that is not accessible from internet
→ use Route Table to access internet & subnets
→ Internet Gateways helps our VPC instance connect with the internet. public subnet have route to the internet gateway.
→ NAT Gateway & NAT Instance allow your instance in your private subnets to access the internet while remaining private.

* Network ACL & Security Groups

→ NACL
→ A firewall which control traffic from and to subnet
→ Can have allow and Deny rules.
→ Can attached at subnet level / only include IP.

→ Security Grps.
→ A firewall that controls traffic to & from an ENI / an EC2 Instance.
→ Can only Allow rules / Rules includes IP address and other Security grps.

* VPC Peering

→ Connect to VPC, privately using Aws network
→ Make them behave as if they were in the same network.
→ Must not have overlapping CIDR

* VPC ENDpoint :-

→ Endpoints allow you to connect to Aws service using a private network instead of the public www network

→ This gives you enhanced security and lower latency to access Aws service.

→ VPC Endpoint Gateway :- S3 & DynamoDB
→ VPC Endpoint Interface :- the rest
→ only use within your VPC.

→ Site-to-site VPN

→ Connect on-premises VPN to Aws
→ The connection is automatically encrypted.
→ Goes over public internet.

* Direct Connect

→ Establish a physical "connection between" on premises and AWS
→ The connection is private, secure and fast
→ Goes over private networks
→ Takes at least month to establish.
→ VPC Flow Logs = network traffic logs

* LAMP stack on AWS

→ Linux, Apache, MySQL, PHP

* Amazon S3

→ main building block of AWS
→ Use cases :- Backup and storage, disaster recovery, Archive, Hybrid Cloud Storage, Application hosting, media hosting.
→
→ S3 allow people to store objects (files) in buckets
→ Bucket must have globally unique name
→ Buckets are defined at regional level.
→ Naming convention :- No uppercase, No underscore, 3-63 character long, Not an IP, Must start with lowercase or number, Not start with ~~prefix~~ sufix and not end with ~~prefix~~ sufix.
→ Objects have a key.

→ Max object size is 5TB (5000GB)
→ If uploading more than 5GB must use "multi-part upload."

* Amazon S3 Security

→ User-based
  ↳ IAM policies - which API called should be allowed for a specific user from IAM

→ Resource-Based

→ Bucket policies :- bucket wide rules from the S3 console - allows cross account

→ Object Access control list (ACL) :- finer grain
→ Bucket Access control list (ACL) :- less common

→ encrypt object in S3 using Encryption Keys.

* Amazon S3 - Replication (CRR - SRR)

→ Must enable versoning in source and destination buckets.
→ Cross-Region Replication (CRR)
→ Same-Region Replication (SRR)
→ Bucket can be in different AWS account
→ Copying is asynchronous.
→ Must give proper IAM permission to S3.

Use case :-

CRR :- compliance, lower latency access, replication across accounts.

SRR :- log aggregation, live replication between production & test accounts.

→ only new object is replicate, so we need to use S3 batch replication for existing one.
→ can replicate delete markers from source to target
→ There is no chaining of replication

* EC2 Instance Metadata (IMDS)

→ AWS EC2 Instance Metadata (IMDS) is powerful but one of the least known features to developers.
→ Allow EC2 to learn about themselves.
→ Using URL
→ Meta data = info about EC2 Instance
→ user data = launch script of the EC2 Instance.

→ MFA with CLI
→ To use MFA with the CLI, you must create a temporary session.
→ To do so you must run the STS GetSessionToken API.
→ aws sts get-session-token

* SDK

→ Use to perform action on AWS using Software development kit.
→ language support :- Java, .NET, NodeJS, Python, Go...

→ API Rate limit
→ Service Quotas (service limit)
→ Exponential Backup
↳ If you get ThrottlingException intermittently, use exponential backoff.

* Amazon S3 Life Cycle Rules

→ Transition Actions :- configure objects to transition to another storage class
→ Move objects to standard IA class 60 days after creation.
→ Move to Glacier for archiving after 6 months.

→ Expiration actions :- configure objects to expire (delete) after some time.
→ Access log file send to delete after 365 days
→ can be used to delete old versions of files
→ delete incomplete Multi part uploads.

→ Rules can be created for the certain prefix
→ Rules can be created for certain objects Tags

→ S3 Event Notification

→ Use case:- generate thumbnails of image uploaded to S3

→ S3 performance

↳ Multipart Upload
↳ S3 Transfer Acceleration

⇒ S3 Select & Glacier Select

→ Retrieve less data using SQL by performing server-side filtering.
→ Can filter by row & column
→ less network transfer, less CPU cost client side

* Amazon S3 - object Encryption

→ you can encrypt object in S3 using 4 methods

(1) Serverside Encryption (SSE)
→ Serverside Encryption with Amazon S3 Managed Keys (SSE-S3)
→ Serverside Encryption with KMS key stored in AWS KMS (SSE-KMS)
→ Serverside Encryption with Customer provided keys (SSE-C)

(2) Client side Encryption

* Cross origin Resource Sharing (CORS)

→ origin = scheme + host + port
→ Web browser based mechanism to allow requests to other origins while visiting the main origin.
→

* S3 Access log

→ For audit purpose, you may want to log all the access to S3 buckets.
→ Any type of S3 request is logged into another S3 bucket.
→ That data is going to analyze.
→ must be in same AWS region.
→ Not same logging bucket & monitoring bucket
→

* S3 object Lambda

→ use AWS Lambda function to change the object before it is retrieved by the caller application.

* AWS Cloud Cloudfront

→ Content Delivery Network (CDN)
→ Improve read performance, content is cached at the edge.
→ Improves user experience
→ 216 point of presence globally (edge location)
→ DDOS protection, Integration with Shield, AWS web app firewall.
→ Great for static content that must be available everywhere

* Cloudfront Caching

→ The cache live at each cloudfront Edge Location.
→ Cloudfront identifies each object in the cache using the cache key.

→ Cloudfront ALB or EC2 as an origin

→ Cloudfront Signed URL / signed cookies

→ Cloudfront - price classes

→ you can reduce the number of edge location for cost reduction
→ There are three classes
① price Class All :- all region
② price Class 200 :- most region not expensive one
③ Price - Class 100 :- only the least expensive

→ Cloudfront - field level Encryption

→ Cloudfront - Real Time Logs

* Docker

→ Docker is s/w development platform to deploy apps.
→ Apps are packed in containers that can be run on any os.
→ Apps run the same, regardless of where they run
  → Any machine
  → No compatibility issue
  → predictable behaviour
  → less work
  → easier to maintain and deploy
→ work with any language, any os and any technology

Use case:- microservices architecture, lift and shift apps from on-premises to AWS Cloud.

→ Docker Image Store

→ Docker Image stored in Docker repositories
→ Docker HUb:-
  ↳ public repository
→ Amazon ECR (Elastic container Registry)
  ↳ Private repository
  ↳ public repository

→ Docker is sort of a virtulization technology, but not exactly
→ Resource are shared with the host ⇒ many containers are on one server

→ Docker Container Management on AWS

→ Amazon Elastic container service (Amazon own container platform)

→ Amazon Elastic Kubernetes service (EKS) (manage Kubernetes)

→ AWS Fargate
  → Amazon serverless container platform
  → work with ECS & EKS

→ Amazon ECR
  → Store container image

* Amazon ECS - EC2 Launch Type

→ Elastic Container Services
→ Launch Docker container on AWS = Launch ECS Task on ECS Clusters.
→ EC2 Launch type :- you must provision & maintain the infrastructure
→ Each instance must run the ECS Agent to register in ECS Cluster
→ AWS take care of starting / stoping containers

→ Amazon ECS - Fargate Launch type

→ Launch Docker containers on AWS
→ You do not provision the infrastructure
→ It's all ~~serve~~ serverless
→ You just create task definition
→ AWS just run ECS Task for you based on the CPU/RAM you need.
→ To scale, just increase the no of tasks.

→ EC2 Instance profile
→ used by ECS agent

→ Amazon ECS - Data Volume :-
→ Mount EFS File system onto ECS Task.
→ Work for both EC2 & Fargate

→ Fargate + EFS = Serverless
→ Use cases :- persistent multi AZ shared storage for your containers.

→ ECS service Auto Scaling

→ Automatically increase/decrease the desired number of ECS task

→ Amazon ECS Auto scaling use AWS Application Auto scaling
↳ Target scaling → Scale based on target
↳ step Scaling → Scale based on specified Cloudwatch
↳ Scheduled ~~Seal~~ Scaling → scale based on specific date/time.

→ Auto scaling group scaling
→ ECS Cluster capacity provider

* ECS Task Placement

→ when a task of type EC2 is launched, ECS must be determine where to place it, with the constraint of cpu, memory and available port.

→ similary when a service scales in, ECS need to determine which task to terminate

→ so we use task Placement and task placement constraints

→ Binpack:
→ place task based on the least available amount of CPU or memory
→ The minimize the number of instance in use

→ Random:
↳ Place the task randomly

→ Spread:
↳ place the task evenly based on specified value

→ you can also mix them together.

→ distinctInstance :- place each task on different container image.

→ memberOf :- place task on instance that satisfy an expression.

* Amazon ECR

→ ECR = Elastic Container Registry
→ Storage and manage docker image on AWS
→ private and public repository
→ Fully integrated with ECS, backed by Amazon S3.
→ Access is controlled through IAM
→ Support Image Vulnerability Scanning, versoning image tags, image lifecycle.

* AWS Copilot

→ CLI tool to build, release, and operate production-ready containerized apps.
→ Run your apps on Apprunner, ECS and fargate.
→ Help you focus on building apps not more on infrastructure.
→ Automated deployment with one command using Codepipeline
→ Deploy to multiple environment
→ Trouble shooting, logs, health status

* Amazon EKS

→ Elastic Kubernetes service.
→ way to launch managed Kubernetes Clusters on AWS
→ Kubernetes is a open source system for automatic deployment, scaling and management of containerized app.
→ alternative of ECS, diff API
→ use case :- if your company already use Kubernetes on-premises or in another cloud and you want to migrate to AWS using Kuber
→ Kubernetes is Cloud-agnostic

* Elastic Beanstalk

→ Elastic Beanstalk is a developer centric view of deploying an application on AWS.
→ It uses all the components like EC2, ASG, ELB, RDS...
→ Managed services :-
  ↳ Automatically handle capacity provisining, load balancing, scaling, application health monitoring.
→ Just the application code is responsibility of developer
→ have full control over configuration
→ Beanstack is free but you pay for the underlying instances.
→ Support many languages & platforms.
  (Custom platform)

→ Beanstalk Deployment option for updates

→ All at once - fastest, but instance aren't available to serve traffic for a bit

→ Rolling :- Update few instance at a time and then move onto the next.

→ Rolling with additional batches :- like rolling but spins up new instances to move the batch.

→ Immutable :- Spins up new instance in new ASG deploy version to this and then swap when all the things are healthy.

→ Blue Green :- Create a new environment and switchover when ready.

→ Traffic Splitting :- Canary testing - send a small % of traffic to new development.

→ Elastic Beanstalk CLI

* AWS Cloudformation

→ Cloudformation is a declarative way of outlining your AWS Infrastructure, for any resources

→ Then cloudformation creates those for you, in the right order with the exact configuration that you specify.

→ Infrastructure as code :-
  ↳ No resources are manually created, which is excellent to control
  ↳ The code can be version controlled eg. git
  ↳ changes are reviewed

→ Cost :-
→ Each resources within the stack is stagged with an identifier so you can easily see how much a stack cost you.
→ estimate a cost using (template)
→ Saving strategy :- delete template at 5 pm and create template at 8 am

→ Productivity :-

→ Ability to destroy & recreate as infrastructure on the cloud
→ Automated generation of Diagram for your tempelate
→ Declarative programming.

→ Seperation of concern :-

→ create many stacks for many apps, and many layers Ex:- Vpc stacks, Network stacks

→ Don't re-invent the wheel
  → Leverage existing templates on the web
  → Leverage the document

→ YAML File

→ YAML & JSON are the languages you can use for Cloudformation.

→ key value pairs
→ Nested objects
→ Support Array
→ Multi line strings
→ Can include Comments

→ Resources

→ Resources are the core of your Cloudformation template.
→ Represent diff AWS components and config
→ Resources are declared and can reference each other.
→ There are over 224 types of resources
→ type identification
  AWS : aws-product-name :: data-type-name

→ Parameters

→ Parameters are the way to provide input to your AWS cloud formation templete.

→ Outputs :-

→ The output section declares optional output values that we can import into other stacks

→ You can view the outputs in AWS Console & CLI

→ useful for ex:- you define network cloudformation and output the variables such as VPC ID and your subnet IDs.

→ It is a best way to perform some collaboration cross stacks, as you let except handle their own part of the stacks.

→ you can delete cloud formation stack if its outputs are being referenced by another cloudformation stack

→ Fn!! Join
→ Fn:: Ref
→ Fn:: GetAtt
→ Fn:: FindInMap
→ Fn:: ImportValue
→ function Fn:: sub
→ Conditions function
  ↳ Fn! And
    Fn! Equals
    Fn: If
    Fn: OR
    Fn:- Not

→ Cloud formation Rollbacks

→ Stacks creation fails :-
  ↳ everything rollback (get deleted)
  we can look at the log.

→ Stack update fails :-
  ↳ The stacks automatically rollback to the previous known working state.
  → Ability to see in the log what happened and error message

→ Cloud formation drift

* Amazon SQS - standard queue service

→ oldest offering
→ fully managed services, decoupled application

Attribute :
→ unlimited throughput, unlimited no of msg in queue
→ Default retension of msg : 4 days, max of 14 days
→ low latency → limitation of 256 KB per msg sent
→ can have duplicate message
→ can have out of order message (best-effort ordering)

→ SQS have unlimited throughput

* SQS - Message Visibility Timeout

→ After the message is polled by a consumer, it become invisible to other consumers.
→ default the message visibity timeout is 30 second

→ Dead letter Queue (DLQ)

→ if a consumer fail to process a msg within a visibility timeout.
the msg goes back to the queue

→ we set threshold of how many time it goes back to queue and After that it goes to dead letter queue (DLQ)

→ DLQ is FIFO queue
→ DLQ of a standard queue
→ Make sure the process the msg before expire
→ expire retension of 14 days in DLQ

→ Amazon SQS - Delay Queue

→ delay a message upto 15 min (consumer don't see immediately)

→ Default is 0 seconds
→ Can set a default at queue level
→ Can override the default on send using the Deploy DelaySeconds parameter

* Amazon SQS - Long polling

→ When a consumer requests message from the queue, it can optionally "wait" for message to arrive if there are none in the queue.

→ This is called Long polling
→ Long polling decrease the number of API calls made to SQS while increasing the efficiency and decreasing the latency of your application.

→ The wait time can be 1 sec to 20 sec
→ Long polling is prefered to short polling
→ enabled at queue level or API level using RecieveMessageWaitTimeSeconds

* SQS Extended Client

→ Message size limit is 256 KB, how to send large messages eg 1 GB?
→ Using the SQS Extended Client (Java library)
→ we save in S3 bucket and send metadata message with S3 bucket link.

* SQS - Must know API

→ CreateQueue (MessageRetentionPeriod), DeleteQueue
→ PurgeQueue: - delete all the message in queue
→ sendMessage (Delay seconds), Recieve Message, Delete Message

→ Max Number Of Messages :- default 1, max 10
→ Recieve Message Wait Time second : Long polling
→ Change Message Visibility : Change the Msg timeout

→ Batch APIs For Send Message, Delete Message, Change Message -Visibility help in decrease in cost.

* Amazon SQS - FIFO Queue

→ FIFO = First In First out (order in the queue)

→ limited throughput : 300 msg/s without batching
3000 msg/s with

→ Exactly once send capability (by removing duplicates)
→ Message are processed in order by the consumer

* SQS FIFO - Deduplication

→ De-duplication interval is 5 minutes
→ Two method :-
↳ Content-based deduplication : will do a SHA-256 hash of the message body.
↳ Explicitly provide a Message Deduplication ID.

* SQS FIFO - Message Grouping

→ If you specify the same value of MessageGroup ID in an SQS FIFO queue, you can only have one consumer and all the message are in order.

→ To get ordering at the level of a subset of msg, Specify different values for MessageGroup ID
→ Each Group ID can have a diff Consumer
→ ordering across group is not guaranteed.

* Amazon SNS

→ Pub / sub pattern
→ The event producer only send message to one SNS topic.
→ As many "event recieves" (subscription) as we want to listen to the SNS topic Notification.
→ Each subscriber to the topic will get the MSG
→ Upto 12,500,000 subscription per topic
→ 100,000 topics limit.

→ SNS —publish→ SQS, Lambda, Kinesis Data Firehose, email, SMS & Mobile, HTTPs end points.

→ Topic publish (using SDK)
→ Direct publish (for mobile app SDK)

→ SNS Security:-

→ Encryption
→ In flight encryption using HTTPS API
→ At rest encryption KMS keys
→ Client side encryption

→ Access control → IAM control
→ SNS Access policies.

* SNS + SQS : Fan out

→ push once in SNS, recieve in all SQS queues that are Subscribers
→ fully decoupled, no data loss
→ SQS allow for : data persistence, delayed, processing and retries of works.
→ Ability to add more SQS & Subscribers over time
→ Make Sure your SQS queue access policy allows for SNS to write.
→ Cross Region delivery : SQS Queues in other Region

→ S3 Events to multiple queues.
→ SNS to Amazon S3 through Kinesis Data Firehose
→ SNS have FIFO capability.
→ SNS FIFO + SQS FIFO : Fan out
→ SNS Message Filtering

* Kinesis

→ Make it easy to collect, process and analyze streaming data in real time.
→ Ingest real-time data such as Application logs, Metrics, website clickstream, IOT telemetry data

→ Kinesis data stream : Capture, process & store
→ Kinesis Data firehose : load data into AWS data store
→ Kinesis Data Analytics → analyze data stream using SQL or Apache Flink
→ Kinesis video streams → capture, process and store video streams.

Kinesis Data Streams

- Retension between 1 day to 365 day
- Ability to Reprocess data.
- once data is inserted in Kinesis, it cannot be deleted
- Data that shared the same potion goes to the same shared.
- ~~producers~~ :-
- producers :- AWS SDL, Kinesis producer library (KPL)
- Consumers :- write your own :- KCL, AWS SDK
  Managed :- AWS Lambda, KDF, KDA

Capacity Mode

- provisioned Mode :-
- you ~~are~~ chose no of shards provisioned, scale manually or using API.
- shards get 1MB/s in (1000 record/ps)
- shards get 2MB/s out (
- pay per hours.

On demand mode :-
- no need to provision or manage the capacity
- Default capacity provisioned (4 MB/s in or 4000 rec
- Scaled automatically based on last 30 days
- pay per ~~stream~~ stream per hours & data in/out per

3)

→ Kinesis producers

→ Put data records into data streams
→ Data record consist of :-
  ↳ sequence no, partition key, Data blob

→ producers :-
  ↳ AWS SDK
  ↳ Kinesis producer library
  ↳ Kinesis Agent

→ write throughput :- 1 MB/sec or 1000 records/sec per shard
→ put Record API
→ Use Batching with putRecords API to reduce costs & increase throughput.

→ Kinesis Consumers

→ Get data records from data streams and process them
→ AWS Lambda
→ Kinesis Data Analytics
→ Kinesis Data Firehose
→ Customer Consumers (AWS SDK)
→ Kinesis client library (KCL)

→ Shared Classic Fan-out consumer - Pull
→ low number of consuming application
→ Read throughput : 2 MB/s per shard across all consumer
→ Max 5 GetRecords API call per sec
→ latency ~200 ms
→ Minimum cost ($)
→ Consumber pull data from Kinesis using GetRecords API calls.

→ Return upto 10 MB or 10000 records

→ Enhanced Fan-out Consumers - Push

→ Multi consuming application for the same stream
→ 2 MB/sec per consumer per shard
→ Latency ~ 70 ms
→ Higher costs ($$$)
→ Kinesis pushes data to consumers over HTTP/2
→ Soft limit of 5 consumers app per data stream

⇒ Kinesis Client Library (KCL)

→ A Java library that help read records from a Kinesis Data stream with distributed app^n sharing the read workload.

→ Each shard is to be read by one KCL Instance
→ 4 shard → 4 KCL
6 Shards → 6 KCL
→ process is checkpointed into DynamoDB
→ Track other workers and share the work amongst shard using ~~Dyno~~ DynamoDB
→ KCL can run on EC2, Elastic Beanstalk and on-prem
→ Records are reads in order at shard level
→ version → KCL 1.x (shared consumer)
KCL 2.x (shared & enhanced fan-out)

* Kinesis Data Firehose

→ Fully managed service, no administration, automatic scaling, serverless → AWS - Redshift, S3, opensearch
→ 3rd party - splunk, MongoDB, DataDog.
→ Send to any HTTP endpoint
→ pay for data going through firehose.
→ near real time
→ 60 sec latency

→ Supports many data formats, conversions, transformations, compression
→ support custom data transformation using AWS lambda
→ can send failed or all data to a backup S3 bucket

* AWS Monitoring, Troubleshooting & Audit

→ AWS CloudWatch :-
→ Metrics :- Collect and track key metrics
→ Logs :- Collect, monitor, analyze and store log files.
→ Events :- Send notifications when certain event happen in AWS
→ Alarm :- React in real-time to metrics / events.

→ AWS X-Ray :-
→ Troubleshooting application performance and errors
→ Distributed tracing of Microservices.

→ AWS CloudTrails :-
→ Internal Monitoring of API calls being made
→ Audit changes to AWS Resources by user.

* Aws Cloudwatch Metrics

→ Cloudwatch provides metrics for every services in Aws
→ Metrics is a variable to monitor
→ Metrics belongs to namespaces
→ Dimension is an attribute of an metric
→ Metrics have timestamps
→ Cloudwatch dashboard for metrics

→ Cloudwatch Custom Metrics

→ Possibility to define and send your own custom metrics to cloudwatch.
→ Ex:- RAM usage, disk space, number of logged in users
→ Use API calls putMetricData
→ Ability to use dimensions → Instance id.
→ Environment.name
→ Metric resolution → Standard
→ High Resolution
→ Important: Accept data point two weeks in past and two hours in the future.

→ Cloudwatch logs

→ Log groups :- arbitary name, usually representing an application.
→ Log stream :- instances within application / log files / containers
→ can define log expiration policy
→ log send to → S3, KDS, KDF, Lambda, opensearch
→ Logs are encrypted by default.
→ can setup KMS-based encryption with your own keys

→ Source :- SDK, Cloudwatch log agent

→ Cloudwatch Logs For EC2

→ By default, no logs from your EC2 machine will go to Cloudwatch
→ You need to run a cloudwatch agent on EC2 to push the log file you want.
→ Make sure IAM permission are correct
→ The Cloudwatch log agent can be setup on-premise too.

→ Cloudwatch Unified Agent
  → CPU, Disk metrics, RAM, Netstat, processes, swap space

→ Cloudwatch Log Metric Filter

→ Cloudwatch logs can use filter expression
  ex :- fing specific IP, logs, errors

→ generate point after filter created
→ ability to specify upto 3 dimensional metric filter.

→ Cloudwatch Alarms

→ Alarm are used to trigger notification for any metric
→ various option
→ Alarm status :- OK, INSUFFICIENT_DATA, ALARM

→ Period :-
→ Length of time in second to evaluate the metric
→ 10 sec, 30 sec, multiples of 60 sec.
→ send notification to SNS

→ Cloudwatch Synthetics Canary

→ configurable script that monitor your APIs, URLs, website
→ Reproduce what your customer do programmatically to find issue before customers are impacted.
→ Integrated with Cloudwatch alarm
→ Script written in Node.js or python
→ programmatially access to headless Chrome
→ Run on schedule
→ Heartbeat Monitor
→ API Canary
→ Broken Links Checker
→ Visual Monitoring
→ canary Recorder
→ GUI workflow Builder

→ AWS EventBridge

→ schedule : Cron jobs (scheduled scripts)

→ Event pattern :- Event rules to react to a service doing something.
→ Trigger lambda function, send SQS | SNS message

→ Event bus can be accessed by other AWS accounts using Resource based policies.
→ You can archive event
→ Ability to ~~archive~~ replay archived events.

→ Schema - Registry
→ Event bus can analyze the events in your bus and infer the schema.
→ It help to generate code
→ It is versioned

→ Resource based policy
→ Manage ~~policy~~ permissions for a specific Event bus.
→ Ex:- allow/ deny events from another AWS account or AWS ~~regr~~ region.
→ Use case:- aggregate all the event from AWS organization.

* AWS X-Ray

→ visual analysis of your application
→ advantage :-
  → Troubleshooting performance
  → understand dependencies in a microservice architecture
  → pinpoint service issue
  → Review request behaviour
  → Find error & exception
  → identified user that are impacted.

→ Compatibility :-
  → Lambda, Elastic Beanstalk, ECS, ELB, API gateway, EC2 Instance

→ AWS x-Ray leverage Tracing
  → Tracing is an end to end way to following a request
→ Tracing is made up of segment (sub segment)

→ X-Ray Instrumentation in your code

→ Instrumentation means the measure of product performance diagnose error and to write trace information.

→ To instrument your application code, you use the X-Ray SDK.

→ X-Ray Concept

→ segment :- each application / service will send them
→ subsegments :- if you need more details in segments
→ Trace :- segment collected together to form an end-to-end trace.
→ Sampling :- decrease the amount of request send to X-Ray reduce cost
→ Annotations :- key value pairs used to index traces and use with filter.
→ Metadata :- key value pairs, not indexed, not use for searching.

→ X-Ray deamon / agent has a config to send traces cross account.

→ AWS Distro for OpenTelemetry

→ secure, production-ready AWS-supported distribution of the open source project open Telemetry project.
→ provide single set of API, library, agent and collector services.
→ Auto-Instrumentation Agents to collect traces without changing your code.

* AWS Cloud Trail

→ Provide, governance, compliance and audit for your AWS account.
→ CloudTrail is enable by default
→ Get an history of events / API calls made within your AWS account
→ Console, SDK, CLI, AWS services

→ Can put logs from CloudTrail to Cloudwatch logs or S3

→ A trail can be applied to All Regions or Sigle Region
→ If all the resources deleted in AWS, investigate CloudTrail first.

→ Events :-

→ Management Events :-
→ Data Events :-
→ CloudTrail Insights

→ CloudTrail Insights to detect unusual activity in your accounts
→ stored for 90 days normally
→ save more then send to S3

* AWS Serverless

→ serverless is new paradigm in which the developers don't have to manage servers more.
→ They just deploy code, just function
→ initially serverless == Faas (Function as a service)
→ before there is only AWS Lambda but now database, messaging, storage etc.
→ It not mean no server it mean you do not need to manage it.

→ AWS Lambda

→ virtual functions - no server to manage
→ limited by time - short executions
→ Run-on demand
→ Scaling is automated.

→ Benefits :-
→ Easy pricing → pay per use
→ Integrated with the whole AWS services
→ Many programming language support
→ Easy monitoring
→ Easy to get more resources.
→ RAM ↑ → cpu and network ↑

→ language support
→ Python, Node.js, Java, C#, Ruby, Custom Runtime API

→ Lambda Container Image
→ The container image must implemented by Lambda API.
→ Ecs /fargate

→ Lambda - Synchronous Invocations

→ Synchronous :- CLI, SDK, API Gateway, ALB
→ ~~Return its result~~ result is returned right away
→ Error handle must happen client side.
→ User Invoked :-
↳ ELB, API Gateway, Cloudfront,
→ Service Invoked :-
↳ Amazon Cognito, step function
→ Other services :-
↳ Lex, Alexa, Kinesis Data Firehose

→ Lambda Integration with ALB

→ To explore a lambda function as an HTTP endpoint
→ you can use ALB or API Gateway.
→ lambda fun^n must be register in target grp.

→ lambda - Asynchronous Invocations
→ S3, SNS, CloudWatch Events
→ The event is placed at event queue.
→ Lambda attempts 3 tries total, 1 min after 1st
2 min after 2nd
→ Make sure the processing is idempotent

→ Services →
→ S3, SNS, CloudWatch Event, EventBridge, codecommit, Codepipeline

→ Lambda - Event and context objects

→ Event objects
→ JSON formated document contain data from the ~~for~~ function to process
→ Contain info" from the invoking service
→ Lambda runtime converts events to an object.
→ Ex:- input arguments, invoking series arguments

→ Context object
→ provides properties and methods that provide the information about the invocation, function and runtime environment.
→ passed to your ~~for~~ f" by lambda at runtime

→ Lambda - Destination

→ can config to send results to destination

→ Asynchronous invocation: can define destination for successful and failed event :-
→ AWS SQS, SNS, lambda & EventBridge bus

→ Now AWS recommends you use destination instead of DLQ now

→ Event source mapping : for discarded events batches
→ SQS, SNS
→ you can send event to a DLQ directly from SQS

→ Lambda Execution Role (IAM Role)

→ Grants the lambda function permission to Aws services / resources.
→ Policies :-
  → AWSLambdaBasicExecutionRole policy
  → " " KinesisExecutionRole
  → " " DynamoDBExecutionRole
  → " " SQSQueueExecutionRole
  → " " VPCAccessExecutionRole
  → " XRayDaemonWriteAccess

→ When we use event source mapping, Lambda use execution Role to read event data.
→ Create lambda execution Role per function

→ Lambda Resource Based Policies

→ use resource based policies to give other accounts and Aws services to use lambda resources
→ give access to all the other services

→ Lambda Environment Variable

→ Environment variable = key / value pair in string
→ Adjust the f^n behavior without updating code footn
→ environment variable are available at code.
→ Lambda services add its own system environment variable as well.
→ Helpful to store ~~keys~~ seecrets.
→ Seecrets can be encrypted

→ Lambda logging & Monitoring

→ Cloudwatch logs :-
→ AWS lambda execution logs are stored in AWS Cloudwatch log.

→ Cloudwatch Metrics :-
→ AWS lambda metrics are displayed in the AWS Cloudwatch Metrics.

→ Lambda Tracing with X-Ray.
→ Enable in lambda configuration
→ Runs the X-Ray Daemon for you
→ environmental variable to communicate with X-Ray
  → _X_AMZN_TRACE_ID
  → AWS_X_RAY_CONTENT_MISSING
  → AWS_XRAY_DAEMON_ADDRESS

→ Lambda@Edge

→ Lambda function is written in Nodejs & python
→ scales to 1000s of request/second.
→ used to change CloudFront request and response.
→ Author your function in one Region and replicate to its location.

* Lambda in VPC

→ You must define VPC ID, the subnet and the security.
→ Lambda will create an ENI (Elastic Network Interface) in your subnet.
→ AWSLAMBDA VPC Access Execution Role.
→ Deploying a Lambda f^n in private subnet gives it internet access if you have NAT Gateway/Instance
→ use VPC endpoint to privately access to Aws services.

→ Lambda function configuration

→ If your application is cpu bound (computation heavy) increase RAM.
→ Timeout :- default 3 second, maximum is 900sec.

→ Lambda layers

→ Custom Runtimes
→ externalize Dependencies to re-use to them

→ Lambda - File System Mounting

→ Lambda function can access EFS File system if they are running on VPC.
→ Config Lambda to mount Efs file system to local directory during initilization
→ Must leverage Efs access point.
→ Limitation:- watch out of the Efs connection limit

→ lambda Concurrency & Throttling

→ Concurrency limit :- upto 1000 concurrent execution
→ can set "reserved concurrency" at the f" level
→ Each invocation over the concurrency limit will trigger a Throttle.
→ If you need higher limit, open a support ticket

→ cold Start :-
→ provisioned Concurrency :-

→ Lambda and cloudformation - inline

→ inline function are very simple.
→ Use the code.zipfile property
→ you cannot include function dependencies with inline functions.

→ other option using S3.

→ you must store the lambda zip in S3
→ you must refer the S3 zip location in the cloudformation code.
  → S3 Bucket
  → S3Key: Full Path to zip
  → S3ObjectVersion: if versioned bucket

→ if you update code in S3 but not the other then Cloudformation won't Update for fn.

→ Lambda container Image

→ Deploy lambda function as container image upto 10 GB from ECR.
→ Pack complex dependencies, large dependencies in container
→ Base image are available for Python, Node.js, Java...
→ can Create your own image as long as it implement the lambda Runtime API.
→ Test the container locally using the lambda Runtime interface Emulator
→ unified workflow to build apps

→ Best practices

→ Use AWS-provided BaseImage
→ Use Multi-stage Build
→ Build from stable to frequently changing
→ Use a single Repository for function with Large Layers
→ Use them to upload large lambda function (10GiB)

→ AWS Lambda versions

→ $LATEST
→ pointer to lambda function version :- Aliases

→ Lambda & CodeDeploy

→ CodeDeploy can help you automate traffic shift for lambda aliases.
→ Features is integrated to SAM framework
→ Linear :- grow traffic every N minutes until 100%.
  → linear 10 percentEvery 3 minutes.
  → " 10 " " 10 minutes
→ Canary :- try X percent then 100%.
  → canary canary 10 percent 5 minutes
  → Canary 10 percent 30 minutes

→ All at once - immediate
→ Can create pre & post traffic hooks to check the health of the lambda function

→ Lambda - function URL

→ Dedicated HTTPS endpoint for your Lambda function
→ A unique URL endpoint is generated for you (never change)
→ Invoke by web browser, Postman, Curl, or HTTP client
→ Access through public Internet only
→ Supports Resources-based policies & CORS configuration.
→ Can be applied to any function Alias or $Latest.
→ Create and config using AWS console & CLI.
→ Throttle using Reserved Concurrency.

→ Security :-

→ Resource based policy
→ CORS

→ AuthType NONE - allow public and unauthenticated access.
→ AuthType AWS_IAM - IAM is used to authenticate and authorize request

→ Lambda and codeGuru profiling

→ Gain insights into runtime performance of your lambda function using codeGuru profiler.
→ CodeGuru creates a profiler Group for your lambda function.
→ support Java & Python

→ AWS lambda limit

→ Execution :-
  → Memory allocation - 128 MB - 10GB (1 MB increment)
  → Maximum execution time - 900 second (15 minutes)
  → Environment Variable (4 KB)
  → Disk capacity in the function container (in /tmp)
    ↳ 512 MB to 10GB
→ concurrency executions :- 1000 (can be increased)

→ Deployment :-

→ lambda function deployment size (.zip) → 50MB
→ size of uncompressed deployment (code + depen) : 250 MB
→ Can use the /tmp directory to load other files at startup.
→ size of environment variable - 4KB

→ AWS Lambda Best practices

→ Perform heavy duty work outside of your function handler
→ Use environment Variable
→ Minimize your development package size to its runtime necessities.
→ Avoid using recursive code, never have a lambda function call itself.

* AWS DynamoDB

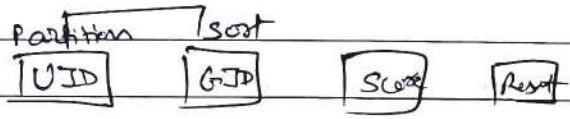
→ NoSQL databases scales horizontally
→ Fully managed, highly available with replication across multi-AZs.
→ No SQL database
→ Scales to massive workloads, distributed database
→ Millions of request per second, trillions of row 100s of TB storage
→ fast & consistance in performance
→ Integrated with IAM securities, authorization and administration.
→ Enable event driven programming with DynamoDB streams
→ low cost and auto-scaling capability
→ Standard & Infrequent Access (IA) Table class

→ DynamoDB Made up of tables
→ Each tables have primary key
→ Can have infinite no of items
→ Each item has attribute
→ Max size of an item is 400KB
→ Data type:-
Scalar type:- String, Number, Binary, Boolean, Null
Document type:- List, Map
Set type:- string set, Number set Binary set

primary key

→ Option 1 :- partition key (HASH)
→ Partition key :- key must be unique for each item
→ Partition key must be diverse so data is distributed
→ Ex :- "USER_ID" for user Table.

option 2 :- partition key + sort key (Hash + Range)
→ The combination must be unique for each item
→ Data is grouped by partition key
→ Ex :- User-game table "User_ID" for partition key and "Game_ID" for sort key.



→ DynamoDB - Read/write capacity Modes

→ control how you manage many msg your table capacity
→ Provisioned Mode (default)
  ↳ you specify the number of read/write per second
  ↳ you need to pay capacity beforehead
  ↳ pay for provisioned read & write capacity unit

→ ON demand Mode
→ Read/write automatically scale up/down with your workloads
→ No capacity planning needed
→ pay for what you use, more expensive

R/W Capacity Modes - provisioned

→ Table must ~~provided~~ provisioned read and write capacity unit
→ Read capacity unit (RCU)
→ Write capacity unit (WCU)
→ option to setup auto-scaling
→ Throughput can be exceeded temporarily using "Burst capacity"
→ advice to do exponential backoff retry.

⇒ WCU

→ one write capacity unit represent one write per second for an item upto 1 KB is size.
→ If item size more it consume more WCU.

Ex-1 :- 10 item per second, with item size 2 KB

→ $10 \times \frac{2\ KB}{1\ KB} = 20$ WCUs

Ex-2 :- 6 item per second, with item size 4.5 KB

$6 \times \frac{5\ KB}{1\ KB} = 30$ WCUs

Ex-3 :- 120 item per minutes, with item size 2 KB

$= \frac{120}{60} \times \frac{2\ KB}{1\ KB} = 4$ WCUs

→ Eventually consistent Read
→ If you read just after a write, its possible we'll get some stale data because of replication

→ Strong Consistent Read

If you read just after write we will get correct data.

→ get consistentRead parameter True

→ Consume twice the RCU.

→ Read Capacity Units

→ One Read capacity unit represent "one" strongly consistent Read per second and "two" Eventually Consistent Reads per second for an item upto 4KB in size

→ If the item are larger than 4KB more RCU are consumed.

Ex:- 10 Strongly Consistent Read per sec, with item size of 4KB

→ $10 \times \frac{4KB}{4KB} = 10 RCUs$

Ex:- 16 Eventually Consistent Read per sec, with item size of 12KB

→ $\left(\frac{16}{2}\right) \times \frac{12KB}{4KB} = 24 RCUs$

Ex:- 10 Strongly consistent Read per sec, with item size of 6KB (6 → 8KB)

→ $10 \times \frac{8KB}{4KB} = 20 RCUs$

→ If we exceed provisioned RCUs and WCUs we get "provisionedThroughputExceededException"

→ Reason:-
→ HOT keys, HOT partition, very large item

→ Solutions :-
→ Exponential backoff, Distribute partition keys
→

→ ON Demand – R/W Capacity Mode

→ Read/write automatically scale up/down with your workloads
→ No capacity planning needed.
→ 2.5X more expensive

→ Dynamo DB – writing Data

→ PutItem :-
→ Create a new item or fully replace an old item (same Primary key)
→ Consume WCUs
→

→ UpdateItem :-
→ Edits an existing items attributes or add new item if it doesn't exist.
→ Can be used to implement Atomic counters :-

→ Conditional writes :-
→ Accept write/update/delete only if conditions are meet otherwise written error
→ Helps with concurrent access to item
→ No performance impact.

→ Reading Data

→ GetItem :-
→ Read based on primary key
→ primary key can be HASH or HASH+Range
→ Eventually consistent read
→ ProjectionExpression can be specified to retrive only certain attributes.

→ Scan :- the entire table and then filter out data
→ for fast we use parallel scan
→ Consume a lot of RCU.
→ Return upto 1MB data.

→ Can use ProjectExpression & filterExpression.

→ Deleting Data

→ DeleteItem :-
→ Delete an individual item
→ Ability to perform a conditional delete

→ DeleteTable :-
→ Delete a whole table and all its items
→ Much quicker deletion than called DeleteItem on all items

→ Batch operation

→ Allow you to save in latency by reducing the no of API calls.
→ Operation are done on parallel

→ BatchWriteItem :-
→ Upto 25 putItem and/or DeleteItem in one call
→ Upto 16MB of data written, 400 KB per item
→ Can't update item
→ UnprocessedItem for file write operation.

→ BatchGetItem :-
→ Return item from one or more tables.
→ Upto 100 items, upto 16MB of data
→ Unprocessed keys for failread operation

→ PartiQL

→ Allow you to insert, select, update and delete data into a dynamoDB using SQL

*→ DynamoDB - Local Secondary Index (LSI)

→ Alternative sort key for your table
→ The sort key consist of one scalar attribute (String, No, B
→ upto 5 local secondary Index per table
→ Must be define at table creation
→ Attribute projection :- Can contain some or all the attributs of the base table

→ Global Secondary Index (GSI)

→ Alternative primary key (Hash or Hash + ~~primary key~~ Range key) from the base table
→ Spead up queries on non-key attributes
→ The index key consist of scalar attribute.
→ Attribute projection :- Some or all the attribute of the base table.
→ Much provision RCUs & WCUs for the index.
→ can be added/modified after table creation.

→ DynamoDB DA Accelerator (DAX)

→ fully managed, highly available, seamless in memory cache for DynamoDB
→ Microsecond latency for cache reads & queries.
→ Don't required application logic modification.
→ Solve Hot key issue
→ 5 minute TTL for cache
→ upto 10 nodes in cluster
→ Multi AZ.
→ Secure

→ DynamoDB Stream
→ ordered Stream of item-level modification in a table
→ Stream records can be
→ Sent to KDS
→ Read by lambda
→ Read by Kinesis Client library application
→ Data Retension 24 hours
→ made up of Shards.
→ Records are not retroactively populated in a stream after enabling it

→ DynamoDB - Time to live (TTL)

→ Automatically delete items after an expiry timestamp

→ DynamoDB CLI

→ --project-expression : one-or more attributes to retrieve
→ --filter-expression : filter item before return to you.

→ AWS CLI Pagination option
- --page-size
- --max-items
- --starting-token

→ DynamoDB Transaction

→ Coordinated, all-or-nothing operations to multiply item across one or more tables..
→ provides ACID
→ Two operations :-
  → Transact GetItems
  → Transact WriteItems

→ DynamoDB Transactions – Capacity Computions

Ex:- 3 Transactional writes per second with item size 5 KB

$= 3 \times \frac{5\,KB}{1\,KB} \times 2 = 30\ WCUs$

Ex:- 5 Transaction read per second with item size 5

$= 5 \times \frac{8\,KB}{4\,KB} \times 2 = 20\ RCUs$

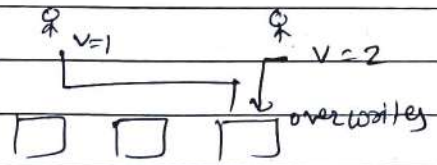
→ DynamoDB write ~~Sharing~~ Sharding

→ Imagine we have a voting app with two candidate A and B

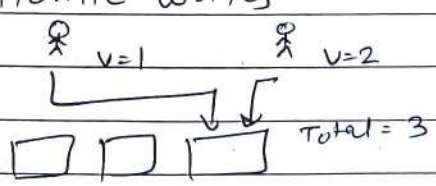
→ A sufix to partition key value
→ Two methods:-
  ↳ Sharding using Random suffix
  → Sharding using calculated suffix

→ Write Types

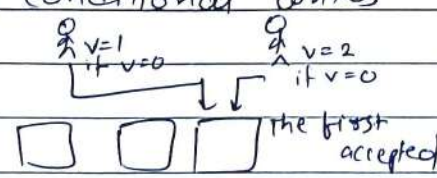
→ Concurrent writes



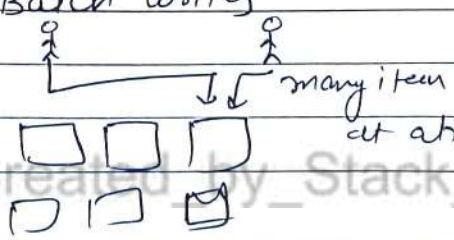
→ Atomic writes



→ Conditional writes



→ Batch writes



* AWS API Gateway

→ Aws lambda + API Gateway: No infrastructure to manage
→ Support for the websocket protocol.
→ Handle API versioning (V1, V2...)
→ Handle different environments (dev, test, prod...)
→ Handle security
→ Create API keys, handle request throttling
→ Swagger / open API import to quickly define API
→ Transform and validate request and response
→ Generate SDK and API specifications
→ Cache API response

→ Lambda Function
  ↳ Invoke lambda function

→ HTTP
  ↳ Expose HTTP endpoint in the backend

→ AWS Service :-
→ Explore any Aws API through the API Gateway.

→ Endpoint Types

→ Edge-optimized :- for global client
→ Regional :- for client within same region
→ private :- can only be using from your VPC

→ API Gateway - Security

→ User Authentication through
  → IAM Roles
  → Cognito
  → Custom Authorizer

→ Custom Domain Name HTTPS

→ API Gateway - Deployment stages

→ Changes are deployed to "stages"
→ you the name you like for stages (dev, test, prod)
→ Each stage have its own parameters
→ stages can be rolled backed as a history of deployment is kept.

→ Stage variable

→ Stage Variable are like environment variable for API Gateway.
→ Use them to change often changing configuration value
→ used in lambda function, ARN, HTTP endpoint.

Use case
→ Configure HTTP endPoint your stages talk to (dev, test, prod)
→
→ Pass configuration parameter to AWS lambda
→ Pass to context object in lambda
→ Format: ${stageVariable.variableName}

→ API Gateway – Canary Deployment

→ possibility to enable canary deployments for any stage. (usually prod)
→ Choose % of traffic the canary channel receive
→ Metrics and logs are seperated.
→ possibility to override stage variable for canary
→ This is blue/green development with AWS lambda & API Gateway.

→ API Gateway Integration Type

→ Integration Type Mock
→ API Gateway return to the response without sending the request to the backend.

→ Integration Type HTTP / AWS (lambda / AWS services)

→ you must configure both the integration request and integration response.
→ setup data mapping using mapping templates for the request & response.

→ AWS_PROXY (Lambda proxy)
→ incoming request from the client is the input to lambda.
→ The function is responsible for the logic of request / response.

→ HTTP - PROXY
→ No Mapping template
→ The HTTP request is passed to the backend.
→ The HTTP response from the backend is forwarded by API Gateway.

→ Mapping Templates

→ Mapping template can be used to modify request/ & response.
→ Rename / modify query string parameter.
→ Modify body content, add header.
→ content type set to application/json or application/xml.

→ JSON to XML with SOAP
→ SOAP API are XML based.

→ Extract data from the request: either path, payload or header
→ Build SOAP Message based on request data.
→ Call SOAP service and receive XML response.
→ Transform XML response to desired format and respond to the user.

→ API Gateway - Open API Spec
→ Common way of defining REST APIs using API definition as code
→ Can export current API as OpenAPI spec
→ Can be written in YAML or JSON
→ We can generate SDK for your application

→ Caching API responses

→ Caching reduces the no of calls to backend
→ TTL is 300 second
→ Caches are defined per stage
→ possible to override cache setting per method.
→ Cache encryption method
→ Capacity → 0.5GB to 237GB
→ expensive, make sense in prod, not in dev, test.

→ API Gateway - logging & tracing

→ Cloudwatch Logs
→ log contain info about request/response body
→ Enable cloudwatch login at the stage level
→ can override settings on a per API basis.

X-Ray
→ Enable tracing to get extra info about requests in API Gateway
→ X-Ray + AWS Lambda give you the full pic.

→ Cloudwatch Metrics
→ Metrics are by stage, possibility to enable details Metrics:
→ CacheHITCount & CacheMissCount: efficiency of the cache.
→ Count, IntegrationLatency, Latency
→ 4XXError & 5XXError
(client side) (server side)

→ Throttling

→ Account limit :-
→ API Gateway throttle request at 1000 rps
→ In case of throttling :- 429 Too many requests
→ Can set stage limit & Method limit to improve the performance.
→ you can define Usage plan.

→ CORS

→ CORS must be enable when you recieve API calls from another domain

→ Option pre-flight request contain in header

→ API Gateway - Security

→ IAM permissions
→ Create an IAM policy authorization and attach to User/role.
→ Authentication = IAM
→ Authorization = IAM policy
→ leverage "sig v4" capacity capability where IAM credential are in header.

→ Resource policies

→ Allow for cross Account Access
→ Allow for a specific source of IP Address
→ Allow for the VPC ENDpoint.

→ Cognito user pool

→ Cognito fully manage user life cycle, token exping automatically
→ API Gateway verifies identity automatically from AWS Cognito.
→ Authentication :- Cognito user pools
Authorization :- API Gateway method.

→ lambda Authorizer

→ TOken based authorizer (bearer token).
→ A request parameter base of lambda authorizer
→ lambda must return an IAM policy for the user, result policy is cached.
Authentication - External
Authorization - lambda function

→ WebSocket API

→ Two way interactive communication between a user browser and a server
→ server can push information to the client
→ This enable stutful application use case.

→ real time application such that chat app, collobration platform, multiplayer games and trading platform.

classmate
Date
Page

* AWS CICD :-

AWS CodeCommit → storing our code
AWS codepipeline → automating our pipeline from code to Elastic Beanstalk
AWS CodeBuild → building & testing our code
AWS CodeDeploy → deploy code to Ec2 Instance
AWS Codestar → manage s/w development Activity in one place
AWS CodeArtifact → store, publish
AWS CodeGuru → automated code review using ML

→ CI

→ Developer push the code to a code repository often
→ A testing / build server checks the code as soon as its pushed
→ The developer get feedback about the tests and checks that have passed/failed.
→ find bugs early, then fix bugs.
→ Delivery faster as the code is tested / Deploy often

→ CD

→ Ensure that the s/w can be released reliably whenever needed.
→ Ensure deployments happen often and are quick.
→ Shift away from "one release every 3 months to 5 per release a day.

→ AWS Code Commit

→ version control is ability to understand the various changes that happen to the code over time
→ Version control - Git
→ A Git repository can be synchronized on your computer, but it is online

→ Benefits
→ collobrative with other developer
→ Make sure the code is backed-up somewhere

→ private Git repo
→ No size limit
→ fully managed and highly available
→ Integrated with Jenkins, AWS CodeBuild, and other CI tools.

→ AWS Codepipeline :-

→ Visual workflow to orchestrate your CICD
→ source - Codecommit, ECR, S3, Bitbucket, Github
→ Build - Codebuild, Jenkins, CloudBees, Teamcity
→ Test - codebuild, AWS device Farm
→ Deploy - code deploy, Elastic Beanstalk, cloudform, ECS, S3
→ Invoke - Lambda, Step function

→ Stage
→ Build → Test → Deploy → load testing → ...

→ Each pipeline stage can create artifacts.
→ Artifacts stored in an S3 bucket and passed on to the next stage
→ Code convert convert in artifact

→ Code Build

→ Source → CodeCommit, S3, GitHub
→ Build instructions :- code file buildspec.yml or insert manually in console.
→ output log can be stored in S3 & CloudWatch log.
→ CloudWatch metrics to monitor
→ Event bridge to detect failed builds and trigger notifi
→ Build project can be defined within codepipeline or CodeBuild.

→ buildspec.yml file must be the root of your code
→ env → define environment variable
→ phases → specify command to run
→ artifacts → what to upload to S3
→ cache → files to cache to S3

→ COde Deploy

→ Deployment Service that automates application deployment.
→ Deploy new application version to EC2 instance, on-premise server, lambda function, ECS services
→ Automated Rollback capability in case of failed deployment.
→ Gradual development control.
→ A file named appspec.yml defines how the deployment happen.

→ can deploy to EC2 or on premises servers
→ perform In-place deployment or blue/green deployment
→ development speed :-
→ AllAtOnce, HalfAtATime, OnceAtATime, Custom

→ AWS CodeStar
→ An integration solution that grops : GitHub, codecommit, CodeBuild, Codedeploy, Cloudformation, codepipeline, Cloudwatch.
→ Quickly create CICD - ready project for EC2, lambda, elastic Beanstalk
→ Issue tracking integration with JIRA / GitHub Issue
→ Ability to integrate with Cloud9 to obtain a web IDE
→ One dashboard to views all the components
→ Free services, pay only for the underline usage of other services.
→ limited Customization

→ AWS Artifact

→ s/w packages depend on each other to be built and new once are created.
→ Storing & Retrieving these dependencies is called artifact management.
→ Traditionally, you need to setup your own artifact management system.
→ Work with common dependencies management tool such as Maven, Gradle, npm, yarn, twine, pip and Nuget.

→ Amazon Code Guru

→ An ML powered service for automated code review and application performance recommendation
→ provide two functionalities

→ Code Reviewer :- automatic code review for static code analysis

→ CodeGuru profiler :- visibility/ recommendation about application performance during runtime.

↔ Max Stack Depth.
→ Memory Usage limit Percent
→ Minimum Time For Reporting In Milliseconds → sending reports
→ Reporting Interval In Milliseconds → report profile
→ Sampling Interval In Milliseconds → sample profile

→ AWS Cloud 9

→ Cloud based Integrated Development Environment
→ Code editor, debugger, terminal in a browser
→ work on your project from anywhere
→ prepacked with essential tools for popular programming language.
→ Share your development env with your team.
→ fully integrate with AWS SAM & lambda to easily build serverless application.

* AWS SAM

→ SAM = Serverless Application model
→ Framework for deploying serverless application.
→ All the configuration in YAML code.
→ Generate complex Cloudformation from simple SAM YAML file.
→ Support anything from cloudformation: output, mapping, Parameter and resource.
→ only two commands to deploy to AWS.
→ SAM use codeDeploy for Lambda function
→ can help to run Lambda, API Gateway, Dynamodb

→ SAM - CLI Debugging

→ Locally build, test, and debug your serverless application that are defined using AWS SAM templates.
→ provide lambda like execution env locally
→ SAM CLI + Toolkits ⇒ step throughs and debug your code.
→ Support IDE :- Cloud 9, VScode, JetBrains
→ AWS Toolkits :- IDE plugins which allow you to build, test, debug, deploy and invoke Lambda function built using AWS SAM

→ SAM Policy Templete

→ list of templetes to apply permission to your lambda
→ IMP example :-
S3Read policy :- Give read only permissions to object in S3
SQS pollerpolicy :- Allow to poll an SQS queue
DynamoDBCrud policy :- CRUD = Create read update delete

→ SAM and CodeDeploy

→ SAM framework natively uses CodeDeploy to update lambda function.
→ Traffic shifting features
→ pre-Post traffic hooks features to validate deployment
→ Easy and automated rollbacks using Cloudwatch

→ AutoPublishAlias :-
  → Detect when new code is being deploy.
  → Create and ~~upto~~ publishes an update version of that function with the latest code.
  → points the alias to the updated version of the lambda function

→ Deployment preference
  → Canary, linear, AllAtOnce

→ Alarm
  → Alarm that can trigger a rollback

→ Hooks
  → pre and post traffic shifting lambda function to test your deployment.

→ SAM - Local Capabilities

→ Local start AWS Lambda
→ start a local endpoint that emulates AWS lambda
→ can run automated test against this local endpoint

→ locally invoke lambda function

→ Sam local invoke
→ Invoke lambda function with payload once and quit after lmbd invocation complete.
→ Helpful for generating testcase.
→ It function call API to AWS then correct --profile option.

→ locally start an API Gateway endpoint
→ Sam local start-api
→ start a local HTTP server that host all your f"
→ automatic reloaded

→ Generate AWS event for lambda function
→ sam local generate-event
→ Generate local sample payloads for event sources

Summary
→ SAM is built on Cloud formation
→ SAM required the Transform and Resources Section.
→ Commond to know:-
→ SAM build: fetch dependencies and create local deployment artifacts
→ Sam package: Package and upload to S3, generate CF template
→ Sam deploy: deploy to Cloud formation

→ SAM policy templates for easy IAM policy definations
→ SAM is integrated with Code Deploy to do deploy to lambda aliases.

* AWS Cloud Development Kit

→ Define your cloud infrastructure using similar language:- Javascript/typescript, python, java and .Net

→ Contains high level components called constructs
→ The code is compiled into a cloudformation templates (json/YAML)

→ you can therefore deploy infrastructure and application runtime code together.
  → Great for lambda function
  → " " Docker containers in ECS/EKS.

→ CDK + SAM
→ use you can use SAM CLI to locally test your CDK apps.
→ you must first run cdk synth.

→ CDK Constructs

→ CDK construct is a components that encapsulates everything CDK need to create the final cloudformation stack.
→ can represent a single AWS resource or multiple related resources.
→ AWS construct Library
  → A collection of constructs included in AWS CDK which contain constructs for every AWS resources.
  → Contains 3 different levels of construct available (L1, L2, L3)

→ construct Hub:- Contains additional constructs from AWS, 3rd parties, and open-source CDK community.

→ Layer 1 → CFN Resources which represent all resources directly available in CF.

Layer 2 → higher level AWS resources but with a higher level.

Layer 3 :- Can called patterns which represent multiple related resources.

→ CDK Bootstrapping

→ The process of provisioning resources for CDK before you can deploy CDK apps into an AWS environment.
→ AWS environment = account & region
→ Cloudformation stacks called CDKToolkit is created
  → S3 Bucket
  → IAM Role

→ Testing

→ CDK Assertions Module
→ Two types :-
  → Fine-grained Assertions
  → Snapshot Tests.

→ Template.fromStack(MyStack) → inside
→ Template.fromString(MyString) → outside.

* Amazon Cognito

→ Give user an identity to interact with our web or mobile application.

→

→ Cognito user pool :-
- → Sign in functionality for app user.
- → integrate with API Gateway & Application load balancer.

→

→ Cognito Identity pool (Federated Identity)
- → provide AWS Credential to user so they can access AWS resources directly.
- → integrate with Cognito user pools as an identity provider.

→ Cognito VS IAM :-
- → "hundred of user", "mobile user", "authenticate with SAML"

→ Cognito User pools (CUP) - User features

→ Create a serverless database of user for your web and mobile apps.
→ Simple login : Username & password
→ password rest, phoneno & Email verification, MFA
→ Federated Identities
→ Feature :- block user if there Credential are compromised elsewhere.
→ login send back to JSON web tocken.
→ Integrate with API Gateway & Application load Balancer.

→ Cognito Identity Pool

→ Get identities for "users" so they obtain temporary AWS credentials.
→ your identity pool include:-
→ public provider, user of CUP., OpenID Connect provide
→ Cognito Identity pool allow for unauthenticated access.
→ Then user can access aws services directly through API Gateway.

* AWS Step Functions

→ Model your workflow as state machines (one per workflo
→ order Fulfillment, Data processing
→ web application, Any workflow

→ Visulation Visulization of workflow, execution of workflow, as well as history
→ start workflow with SDK call, API Gateway, Eventbridge

→ Error handling

→ Any state can encounter runtime error for various reasons:-
→ state machine definition issue, Task failure, Transient issue
→ use Retry and Catch
→ predefined as error code:- States.All, states.Timeout, State.TaskFailed, State.permissions

→ ErrorEquals
→ IntervalSeconds
→ BackoffRate
→ MaxAttempts
→ max attempts then enter ~~cache~~ catch
→ Error Equals
→ Next
→ Resultpath → define the input for next state/field

→ Activity Task
→ Enable you to have the task work performed by an Activity worker.
→ Activity worker apps can be running on EC2, lumbda, mobile device.
→ Activity worker pool for a task using GetActivityTask API.
→ After Activity worker complete is work, it send a response of its success/failure using sendTaskSuccess or SendTaskFailure
→ wait time → Timeoutsecond
heartbeat from activity worker → SendTaskHeartBeat
HeartBeatSeconds
→ define time using Timeoutsecond and
Max time → 1 year.

→ Aws AppSync.
→ AppSync is managed service that uses GraphQL
→ GraphQL make it easy for application to get exactly the data they need.
→ Combination of data.
→ Start with one GraphQL schema

→ Amplify studio
→ Amplify CLI
→ Amplify Libraries
→ Amplify Hosting

* AWS STS - Security Token Service

→ Allow to grant limited and temporary access to AWS resources.
→ AssumeRole :- Assume role within your account or cross account.
→ Assume Role with SAML — credential of user logged in with SAML.
→ AssumeRole With WebIdentity :- return cred for user logged it with an IDP.
→ GetSessionToken :- for MFA.
→ GetFederationToken :- obtain temp cred for federated user.
→ GetCallerIdentity : return details about the IAM user or role used in API calls.
→ DecodeAuthorizationMessage :- decode error message when AWS API denied.

→ AWS Directory services

→ Aws managed Microsoft AD
→ Create your own AD in AWS, manage user locally, support MFA.

→ AD Connector :-
→ Directory Gateway to redirect to an premise AD, support MFA.
→ user are managed on the on-premise AD

→ Simple AD
→ AD-compatible managed directory on AWS
→ Cannot be joined with on-premise AD

* AWS Security and Encryption

→ AWS KMS

→ Aws manage encryption key for us
→ Fully integrated with IAM for authorization
→ Easy way to control access to your data.
→ Able to audit KMS key usage using CloudTrail
→ Seamlessly integrated into most Aws service
→ Never ever store your secrets in plaintext, especially in your code.
→ also available through API calls.
→ Encryption ~~key~~ secret can be stored in code / env variable

→ KMS Keys Types

→ Symmetric (AES-256 keys)
  → Single encryption key that use for Encrypt & decrypt
→ Aws Service that are integrated with KMS use symmetric KMS CMKs
→ you never get access to the KMS keys unencrypted (through API only)

→ Asymmetric (RSA & ECC key pairs)
→ public (Encrypt) and public key (Decrypt) pair
→ used for Encrypt / Decrypt or sign / verify operation
→ The public key is downloadable, but you can't access the private key unencrypted.
→ encryption outside aws

→ Types of KMS Key

→ Aws owned keys (free)
→ Aws managed key (free) (eg:- aws/service-name)
→ Customer managed keys created in KMS: $1/month
→ Customer managed keys imported
→ + pay for API calls to KMS

→ Automatic key rotation
→ Aws managed KMS key: auto every 1 year
→ Customer-managed KMS keys = 11
→ Imported KMS keys: manual rotation

→ Envelope Encryption

→ KMS Encryption API calls has limit of 4KB.
→ If we need to encrypt > 4KB we need to use Envelope Encryption.
→ The main API helps is GenerateDataKey API

for exam :- anything over 4KB of data that needs to be encrypted must use the Envelope Encryption
== GenerateDataKey API

→ The Aws Encryption SDK implemented envelope Encryption for us.

→ Data key caching
→ re use keys instead of creating new all the time on encryption

→ Use LocalCryptoMaterials cache

→ S3 Bucket Key for SSE-KMS encryption.

→ New setting to decrease
  → No of api call made to KMS from S3 by 99%.
  → Cost of ~~Overall~~ overall KMS encryption with Amazon S3 by 99%.

→ A S3 bucket key is generated
→ That key is used to encrypt KMS object with new data keys

→ You will see less KMS cloudtrail events in cloudtrail

→ CloudHSM

→ AWS manage the s/w for encryption.
→ CloudHSM → aws provision encryption hardware
→ HSM is tamper resistent
→ Support both type of encryption
→ good for SSE-C encryption

→ SSM parameter store

→ Secure storage for configuration and secrets
→ Optional seamless encryption using KMS
→ Serverless, Scalable, durable, easy SDK
→ Version tracking of configuration / secrets
→ Security through IAM
→ Notification with Amazon EventBridge
→ Integration with Cloudformation

* Aws Secrets ~~Keys~~ Manager

→ Newer service, meant for storing secrets
→ Capability to Fource rotation of secrets every X days
→ Automatic generation of secrets or rotation
→ Integration with Amazon RDS and many
→ Secrets are encrypted using KMS
→ Replicate secrets among multi Aws Regions
→ Make read Replica in sync with primary secrets

* AWS SES Simple Email service

→ send email to pepple using:-
→ SMTP interface → or aws SDK

→ Ability to recieve ~~msg~~ email. integrates with
→ S3, → SNS → Lambda

→ ~~Integrated~~ with IAM for allowing to send emails

* Amazon Opensearch service

→ Amazon ~~Successor~~ Opensearch is successor to Amazon Elastic search
→ In DynamoDB, queries only exist by primary key or ~~indies~~ indexes. with opensearch, you can search any field, even partially matches.
→ Two modes:- managed cluster or serverless cluster
→ Does not natively support SQL.
→ Ingestion from Kinesis Data Firehose, Aws IOT, and Cloudwatch logs.

→ Security through Cognito & IAM, KMS encryption, TLS
→ Comes with opensearch Dashboards

* Amazon Athena

→ serverless query service to analyze data stored in S3.
→ Uses standard SQL query language to query the files. (built on presto)
→ Support CSV, JSON, ORC, Avro and Parquet
→ Commonly used with Amazon Quicksight for reporting / dashboards.

→ Use case :- business Intelligence, Analytics, reporting, analyze & query VPC Flow logs, ELB logs...

→ Exam Tip :- Analyze data in S3 using serverless SQL, use Athena.

→ Use columnar data for cost saving
→ Compress data
→ partition datasets
→ large file >128 KB

→ Use datasource connector that run on Aws lambda to run federated Queries

* Amazon Managed streaming for Apache Kafka (Amazon MSK)

→ Alternative to Amazon Kinesis
→ Fully managed Apache Kafka on AWS
  → Allow to create, update, delete cluster
→ MSK create and manage Kafka brokers nodes
→ Deploy the MSK Cluster in your VPC, multi AZ
→ Automatic recover from failure.
→ Data is stored on EBS volume for long as you want.

MSK Serverless

→ Run Apache Kafka on MSK without managing the capacity.
→ MSK automatically provisions resources and scale Compute & Storage.

* AWS Certificate Manager (ACM)

→ let's you easily provision, manage, and deploy SSL/TLS certificates
→ used to provide in-flight encryption for website
→ Support both public and private TLS certificates
→ Free of charge for public TLS certificates.
→ Automatic TLS Certificate renewal
→ Integration with
  → ELB, CF Distribution, API on API Gateway

→ AWS private Certificate Authority (CA)

→ Manage service allow you to create private certificate authority (CA) including root and subordinates CA.
→ can issue and deploy end-entity X.509 certificates
→ Certificates are trusted by only your organization
→ Work for AWS services which are integrated with ACM.

→ Use case :- Encrypted TLS communication, Cryptographically signing code, Authenticate user, Computer, API, IoT.

* Amazon Macie

→ Amazon Macie is fully managed data security and data privacy service that uses ML and pattern matching to discover and protect your sensitive data in AWS.

→ Macie help identify and alert you to sensitive data such as personally identifiable information (PII).

* AWS AppConfig

→ Configure, validate and deploy dynamic configuration.
→ Deploy dynamic configuration changes to your application independently of any code deployment
→ Feature flag, application tuning, allow/block listing
→ use in EC2, lambda, ECS, EKS.
→ Gradually deploy the config changes and rollback if issue occurs.
→ Validate configuration changes before deployment :-
  → JSON schema
  → Lambda function

* → CloudWatch Evidently

→ ~~Safly~~ Safely validate new features by serving them to a specified % of your user.
→ Launches (= Feature flag): enable and disable features for a subset for user.
→ Experiments (= A/B testing): Compare multiple versions of the same features
→ Overrides :- pre-define a validation for a specific user.
→ Store evaluation events in cloudwatch logs ~~in~~ or S3.