**Cloud Computing:**

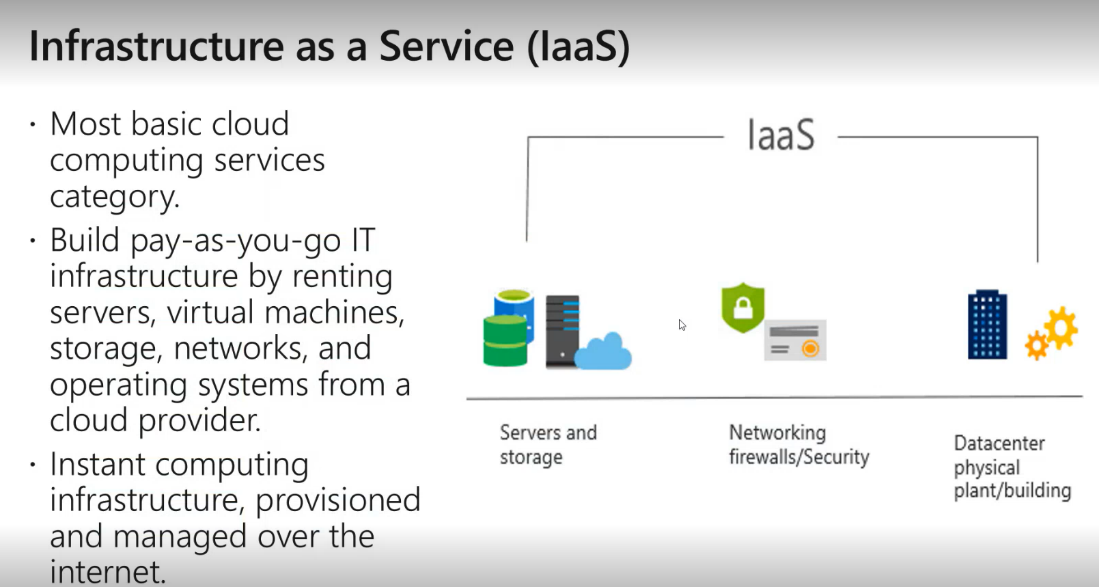
* Cloud computing is like renting a computer or storage space on the internet. Instead of relying on your own computer, you use someone else's computer (servers) that is located in a data center.
  + **Compute power**
  + **Storage**
  + **Networking**
  + **Analytics**

**Types of Cloud Models:**

* **Public Cloud:**
  + Owned by cloud services/hosting provider.­
  + Provides resources/services to multiple organizations & users.
  + Accessed via secure network connection.
* **Private Cloud:**
  + Owned & operated by organizations.
  + Organizations are responsible for operating.
* **Hybrid Cloud:**
  + Combination of public & private clouds to allow apps to run in most appropriate location.
    - Ex: data in private cloud & infrastructure in public cloud.

**Service Models:**

* **Iaas:**

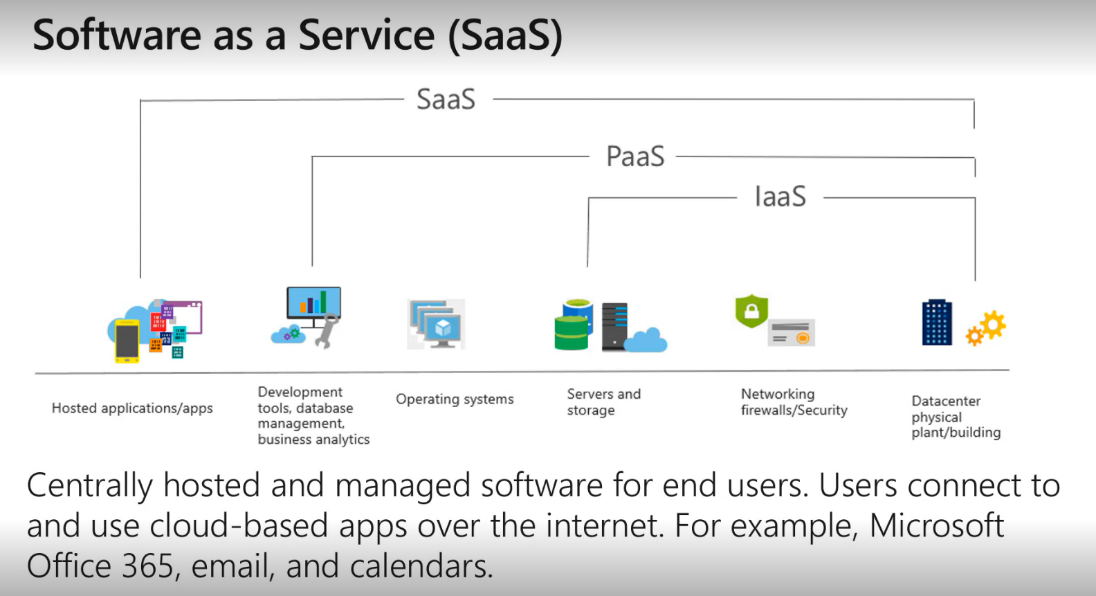
****

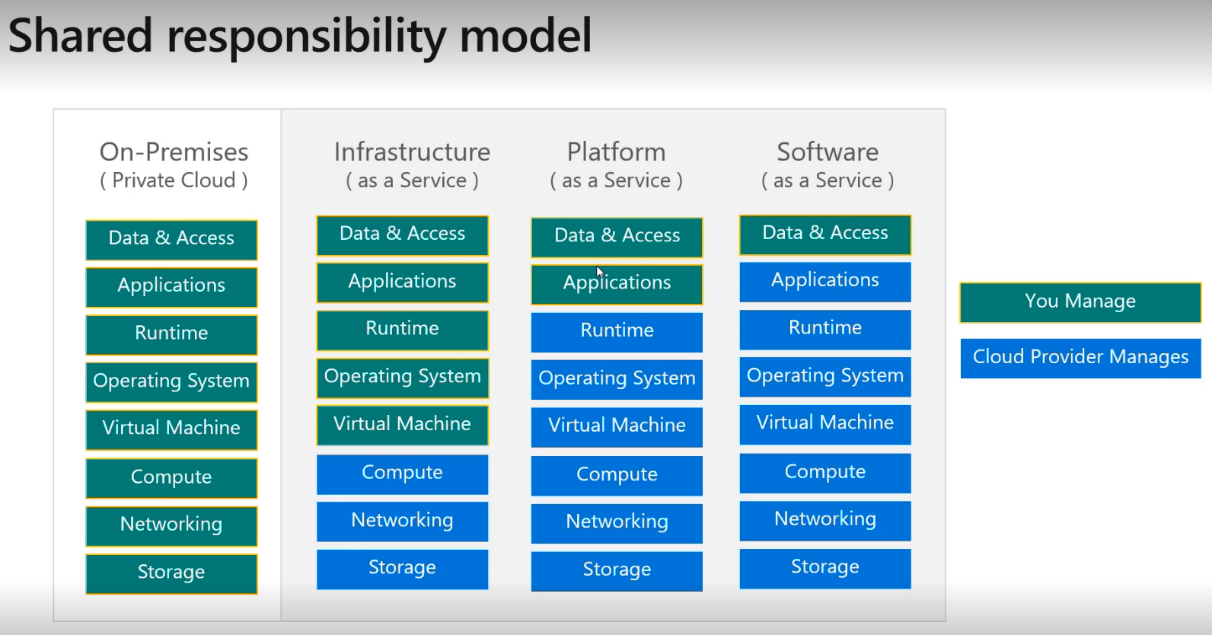
* **Paas:**

**A diagram of a service

Description automatically generated**

* **Saas:**

****

* 

**Feature of AWS:**

**A diagram of a cloud service

Description automatically generated**

**====================================================================================================================================================================================**

**AWS Products:**

**A screenshot of a website

Description automatically generatedA diagram of a cloud

Description automatically generated**

**Elastic Cloud Compute in AWS:**

* Amazon provides resizable compute capacity in the cloud, which is intended to make web-scale computing easier for developers.
  + **Amazon EC2:**
    - Container management service that supports Docker containers & allows us to easily run apps on a managed cluster of Amazon EC2 instances.
    - Highly scalable
    - High performance
  + **AWS EBS:**
    - Easy to use service for deploying & scaling web apps & services.
  + **AWS Lambda:**
    - Let us run the code without provisioning/managing servers.
    - Pay only for the compute & execution time that we use.
    - Serverless computing.
  + **Elastic Load Balancing:**
    - Is a networking service that automatically spreads out incoming application traffic across serveral available EC2 instances.
  + **Amazon EKS:**
    - Is an extension of ECS that runs kubernetes.
    - Is a managed server to handle the installation, deployment, and scaling of kubernetes containers & clusters.

**IAM (Identity & Access Management):**

* With AWS IAM, you can manage users, groups, and permissions to ensure that only authorized individuals or systems have access to your AWS resources. IAM allows you to create and manage AWS users and groups and use permissions to allow and deny their access to AWS resources.
* It is a global service.
* IAM concepts,
  + **Users -** people within the organization & can be grouped.
  + **Groups -** contains users, not other groups.
  + **Roles -** some AWS services will need to perform actions on our behalf, to do so we assign permission to AWS services with IAM roles.

Common roles,

**EC2 instance roles**

**Lambda function roles**

**CloudFormation roles**

* + **Permissions -** users/groups can be assigned JSON docs called policies.
  + **Policies -** it defines the permissions of the users.

**Version - policy language, “2012-10-17”**

**Id - id for policy**

**Statement**

**Sid - id for statement**

**Effect - allows/denies access.**

**Principal - policy applied to which accout/role.**

**Action - list of actions policy allows/denies.**

**Resource - list of resource to which action applied to**

**Condition - condtion to apply the policy.**

**AWS Access Key:**

* **AWS access key, to access AWS we have 3 options,**
  + Generated through AWS console.
  + Users manage their own access key.
  + **Access KeyId - username**
  + **Secret Access Key - password**
  + **AWS Management Console. (protected by password + MFA)**
  + **AWS CLI (protected by access key)**
  + **AWS SDK (protected by access key)**

**AWS CLI:**

* We can develop scripts to manage resources.
* Alternative to using AWS Management Console.
* Setup,
  + Download CLI.
  + **aws --version (checks version)**
  + **aws configure (for configuring accessKeyId, SecretAccessKey, region, oupput formatType)**
  + **aws iam list-users (lists users)**

**AWS SDK:**

* We can develop scripts to manage resources.
* Alternative to using AWS Management Console.
* Language specific APIs, enables us to access & manage AWS services programatically.
* Emkbedded within our application & supports SDK – JS, PHP, JAVA, android, ios, IOT SDKs.

**IAM Best Practices:**

* Don’t use root account except for AWS account setup.
* One physical user = one aws user (never ever share credentials)
* Assign users to group & assign permission to group.
* Use MFA & create strong password policy.
* Create & use roles for giving permission to the aws services.
* Use access key for programmatic access (CLI/SDL).
* Never share IAM users & access key.

**====================================================================================================================================================================================**

**S3 (Simple Storage):**

* Amazon S3 is one of the main building block of AWS.
* It advertised as infinity scaling storage.
* Many websites use amazon S3 as a backbone.
* Amason S3 allows people to store objects (files) in bucket (repository).
* Buckets must have a globally unique name (across all region all accounts).
* Buckets are defined at region level.
* S3 looks like global service but are creayed in a region.
* Even though the bucket has public access, we can’t access the buckets. Because, Policies should be added for S3 buckets (list bucket, listAllBuckets etc..)
* **Amazon S3-object,**
  + Object (files) have a key.
  + The key is in full path.
    - s3://my-bucket/my\_file.txt
    - s3://my-bucket/my\_folder/my\_file.txt
  + The key is composed of prefix + object name.
    - s3://my\_bucket/my\_folder/my\_file.txt
  + object value is the content of the body.
    - max object size is 5TB.
    - If uploaded more than 5GB must be use multi part upload.
* **Naming convention,**
  + no uppercase, no underscore
  + 3-63 character long
  + Not an IP
  + Must start with lowercase/number
  + Must not start with the prefix xn--
  + Must not end with the suffix -s3alias
* **Usecases,**
  + Backup & storage
  + Disaster revovery
  + Archive purpose
  + Hybrid cloud storage
  + Application hosting
  + Media hosting
  + Big data analytics
  + Software delivery
  + Static website
* **Generating policy using S3 bucket policy generator,s**

{

"Id": "Policy1698120107454",

"Version": "2012-10-17",

"Statement": [

{

"Sid": "Stmt1698120106202",

"Action": "s3:\*",

"Effect": "Allow",

"Resource": "arn:aws:s3:::mavb1/\*",

"Principal": "\*"

}

]

}

**Amazon S3 Security:**

* User based security,
  + IAM policies – which API call should be allowed for a specific user from IAM.
* Resource based security,
  + Bucket policies – bucket wide rules from S3 console
  + Object access control list – finer grain
  + Bucket access control list – less common
* An IAM principal can access an S3 object if,
  + The user IAM permission allow it or the resource policy allow it & there is no explicit deny
* Encryption
  + Encrypt objects in amazon S3 using encryption keys.

**Amazon S3 – static website hosting:**

* S3 can host static websites & have them accessible on the internet.

**Amazon S3 – Bucket Versioning:**

* We can version our files in amazon S3.
* It is enabled at bucket level.
* Same key overwrite will change the version: 1,2,3, . . .
* It is best practice to version our buckets.
  + Protect against unintended deletes.
  + Easy roll back to previous version.
* Any file that is not versioned prior to enabling versioning will have version “nul”.

**Amazon S3 – deploy springboot App:**

* **2 ways to deploy,**
  + EC2 - IaaS
  + Elastic BeanStack - PaaS
* **Steps,**
  + Create a springboot app & create a jar file.
  + Create a linux EC2.
  + ssh into linux instance.
  + sudo -i
  + wget <java-jdk-link> (file should be .rpm)
  + rpm -ivh jdk-17\_linux-x64\_bin.rpm (rpm is a package manager & installs jdk)
  + java --version
  + wget <url of jar file uploaded on S3>
  + ls
  + java -jar <jar file name>
  + modify the security group to allow the relevant port number. (because by default port:22 is allowed)
  + Test the app on browser with public IP address.

**AWS: RDS + Aurora + Elastic Cache**

* **AWS RDS,**
  + RDS stands for Relational Database Services.
  + It is a managed DB service for DB using SQL as a query language.
  + It allows us to create DB’s in Cloud that are managed by AWS.
    - Postgress, MYSQL, MariaDB, Oracle, MS SQL Server, Aurora (AWS proprietary DB)
  + Advantage over using RDS versus deploying DB on EC2,
    - RDS is a managed Services,
      * Automated provisioning, OS patching.
      * Continious backups & restore to specific timestamp.
      * Monitoring dashboard.
      * Multi AZ setup for disaster recovery.
      * Maintenance windows for upgrades.
      * Scaling capalibility (vertical & horizontal)
      * Storage backed by EBS.
  + We can’t SSH into the instance.
  + Helps us increase storage on our RDS DB instance dynamically,
    - When RDS detects we are running out of free db storage & scales automatically.
    - Avoid manual scaling of our DB storage.
    - You must set max storage threshold. (Max limit for DB storage)
    - Support all RDS database engines.
  + Only master/primary node can read & write, remaining can write to db to maintain consistency in data.
  + Max limit for replicas is 15.
  + No charges if replicas are in the same region vice versa.
  + We can connect to RDS from our local machine using the endpoint of the database.
    - The endpoint can be found in databse configuration.
* **Amazon Aurora,**
  + It is proprietary technology from aws – not open sourced.
  + Postgress & mysql are both supported as aurora db.
  + Aurora is aws cloud optimized & claims 5x performance improvement over the mysql on RDS, over 3x performance improvement on postgress.
  + Aurora storage automatically grows in increment of 10GB up to 128TB.
  + Aurora can have up to 15 replicas & the replication process is faster than MYSQL.
  + Aurora Costs more than RDS (20% more) – but is more efficient.
* **Elastic Cache,**
  + The same way RDS is to get managed relational database.
  + Elastic cache is to get managed redis or memcache.
  + Caches are in-memory database with high performance, low latency.
  + Helps reduce load off database for read intensive workload.
  + Helps make our app stateless.
  + **Redis vs Memcache,**
    - Multi AZ with auto failover.
    - Read replicas to scale read & have high availability.
    - Data durability.
    - Back up & restore features.
    - Support set & sorted set.
    - Multi-node for partitioning of data.
    - No high availability.
    - Non persistence.
    - No backup & restore.
    - Multi threaded architecture.

**====================================================================================================================================================================================**

**ASG – Auto Scaling Group :**

* Scale out (add EC2 instances) to match an increase load.
* Scale in (remove EC2 instances) to match a decrease load.
* Ensure we have min & max number of EC2 instance running.
* Automatically register new instance to a load balancer.
* Re create an EC2 instance in case a previous one is terminated.
* ASG are free (we must pay for the underlying EC2 instance)
* **A launch template,**
  + AMI + instance type
  + EC2 user data (when creating instance, user data executes only once.)
    - **#!/bin/bash**
    - **# Use this for your user data (script from top tom bottom)**
    - **# install httpd (Linux 2 version)**
    - **yum update -y**
    - **yum install -y httpd**
    - **systemctl start httpd**
    - **systemctl enable httpd**
    - **echo "<h1>Hello World from $(hostname -f)</h1>" > /var/www/html/index.html**
  + EBS volume
  + Security Group
  + SSH key pair
  + IAM roles
  + Network + subnet info
  + Load balancer info
* Min size/max size/initial capacity,
  + **Scaling policies**

**====================================================================================================================================================================================**

**Load Balancing:**

* Load balancer are servers that forward traffic to multiple servers (EC2 instances) downstream.
* Spread load across multiple downstream instances.
* Expose a single point of access to your apllication.
* Seamlessly handle failure of downstream instance.
* Do regular health checks to your instances.
* Provides SSL termination of your website.
* Separate public traffic from private traffic.
* High availability across Zones.

**Why we need Elastic Load Balancer?**

* Is a managed load balancer.
  + AWS guarantees that it will be working
* It costs less to setup your own load balancer but it will be a lot more effort on your end.
* It is integrated with many AWS services
  + EC2, EC2 Auto Scaling Group, ECS
  + Route 53, cloud match

**Health Check:**

* It is crucial for load balancer.
* They enable the load balancer to know if instance is forward traffic to are available to reply request.
* It he resoponse is not 200 (OK) then the instance is unhealthy.

**Types of Load Balacner:**

* **Class Load Balancer**
  + **HTTP, HTTPS – deprecated**
* **Application Load Balancer (ALB)**
  + **V2 – new generation -2016 – HTTP, HTTPS, Websocket**
* **Network Load Balancer**
  + **V2 – TCP, TLS, UDP**
* **Gateway Load Balancer**
  + **IP Protocol**

**Application Load Balancer:**

* example.com/users - target group1
* example.com/posts - target group2
* path in url
* hostname in url
* ALB are a great fit for micro services 7 container-based app (example docker)

**Network Load Balancer (V2):**

* Network load balancers (layer 4) allow to,
  + Forward TCP & UDP traffic to our instance.
  + Handle millions of requests per second.
  + Less latency – 100ms.
* NLB has one static IP per AZ & supports assigning elastic IP.
* NLB are used for extreme performance, TCP/UDP traffic.
* Not included in AWS free tier.
* EC2 instances are the target group.
* IP address – must be private IP.
* We can also use Application Load Balancer.
* Health check supports TCP, HTTP & HTTPS protocol.

**====================================================================================================================================================================================**

**Data:**

* SQL
* NoSQL
* Structure - table - RDS
* Semi-structured - NoSQL - MongoDB - DynamoDB (fully managed service by AWS)
* UnStructured - S3

**DynamoDb:**

* Fully managed service by AWS.
  + Tables - collection
  + Rows - document
  + Column - field
* Max size of an item (row) – 400KB
  + string, number, binary, boolean, null
  + list, map
* primary key,
  + option1 - partition key(single column as PK) (user\_id)
  + option2 - partition key+sort\_key = PK (2 columns) (user\_id + game\_id)
* DynamoDB Operations,
  + DynamoDB – writing data,
    - putItem - create a new item/fully replace old item (same primary) - consumes WCU (write compute units)
    - updateItem - edit/add.
    - conditional writes - if condition is true, then only operation (put/update) is made.
  + DynamoDB - reading data,
    - getItem - using primary key.
    - Query
      * KeyConditionExpression
  + DynamoDB – deleting data,
    - deleteItem
    - deleteTable
  + can perform Batch Operations.

**====================================================================================================================================================================================**

**CloudFront:**

* CDN - content delivery network (highly available servers, data is cached at edge locations for faster access)
* Improves reading performance, content is cached at the edge locations.
* Improves user experience.
* 225+ edge locations.
* cloudFront – origins (source of content)
  + S3
  + Custom origin (http)
* **Before creating cloudFront origin is needed (i.e; S3 bucket / custom origin)**

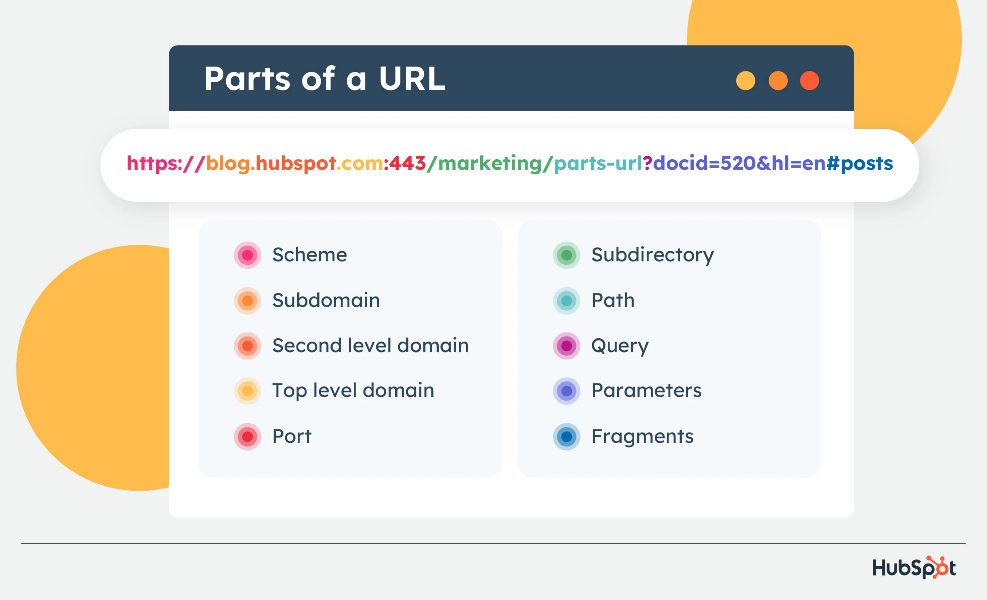
**Monitoring:**

* devOps
* why monitoring,
  + application latency
  + application outages
  + users contacting the IT department/complaining is not a good outcome.
  + Troubleshooting & remediation
* Internal monitoring,
  + Can we prevent the issue from happening?
  + Performance & cost
  + Trends
  + Learning & improvement
* AWS CloudWatch
  + Provides metric for every service in AWS.
    - Metric is a variable to monitor (CPU utilization, networking)
    - Dimension is an attribute of a metric.
      * Upto 30 dimenstions per metric.
  + Metrics - collect & track key metrics.
  + Logs - collect, monitor, analyze & store log files.
  + Events - send notification when certain events occur.
  + Alarms - react in real time to metrics/event.
* AWS X-Ray
  + Troubleshooting application performance & error
  + Distributed tracing of microservices.
* AWS CloudTrail
  + Internal monitoring of api calls being made.
  + Audit chnages to AWS resources by your users.

**====================================================================================================================================================================================**

**Route 53: (Domain Registrar)**

* **53 is a reference to the traditional DNS port.**
* **Supports,**
  + **A ( maps hostname to ipv4)**
  + **AAAA (maps hostname to ipv6)**
  + **CName (maps hostname to hostname)**
  + **NS-NameServer ()**
* DNS – domain name system.
* DNS is the backbone of the internet.
* Domain registrar
  + GoDaddy, Route 53, godly
* DNS records: A (ipv4), AAAA (ipv6), CName, NS
* Zone file (file contains DNS records)
* Name server
* TLD (top level domain) - .com, .us, .co.in, .in, .gov
* SLD (secondary level domain) - amazon.com, google.com
* Parts of an URL,

****

**====================================================================================================================================================================================**

**Progress:**

* **Day1 - creating Linux, Windows instances.**
* **Day2 - creating user, groups, permissions, policies, Roles, account alias, password policy, MFA, AWS CLI, AWS CloudShell (creating, downloading file).**
* **Day3 - static webpage hosting, object uploading, s3 bucket policies, bucket versioning, deploy springboot app, s3 security.**
* **Day4 - creating RDS db, connect to RDS local machine using db endpoint AWS, elastic cache (redis cluster creation), create ASG (Auto Scaling Group).**
* **Day5 - application balancer create, network load balancer create.**
* **Day6 - DynamoDB table creation, cloudFront, cloudwatch, cloudtrail, aws x-ray, Route 53, how an url processed in search engine.**

**====================================================================================================================================================================================**

**Notes:**

* **CapEx -** spend on physical infrastructure upfront. (Capital Expenditure)
* **OpEx -** spend on services/products as needed. (Operational Expenditure)
* **Regions -** there is always a minimum of two availability zones per region. (Zones – datacentres/cluster of datacentres)
* **Edge Locations -** located in major cities around the world to distribute content to end users with reduced latency. (Global reach - site)
* **AMI -** amazon machine image. (OS for an instance)
* **don’t store data in a root volume in EC2.**
* **IAM is a global service.**
* **Root account created by default, shouldn’t be used/shared.**
* **IAM tags to categorize users based on their department for better resource management and access control.**
* **Even though the bucket has public access, we can’t access the buckets. Because, Policies should be added for S3 buckets (list bucket, listAllBuckets etc.)**
* **Any file in S3, that is not versioned prior to enabling versioning will have version “nul”.**
* **We can use NLB (Network Load Balancer) infront of ALB (Application Load Balancer)**