**What is IaaS, PaaS, SaaS**

**IaaS**

Infrastructure as a Service (IaaS) is a business model that delivers IT infrastructure like compute, storage, and network resources on a pay-as-you-go basis over the internet.

In IaaS basically all IT resources are managed by cloud owner, we need to use them for our purpose.

**PaaS**

PaaS (Platform as a Service) combines servers, stockpiling, and organization framework with the product you want to send applications.

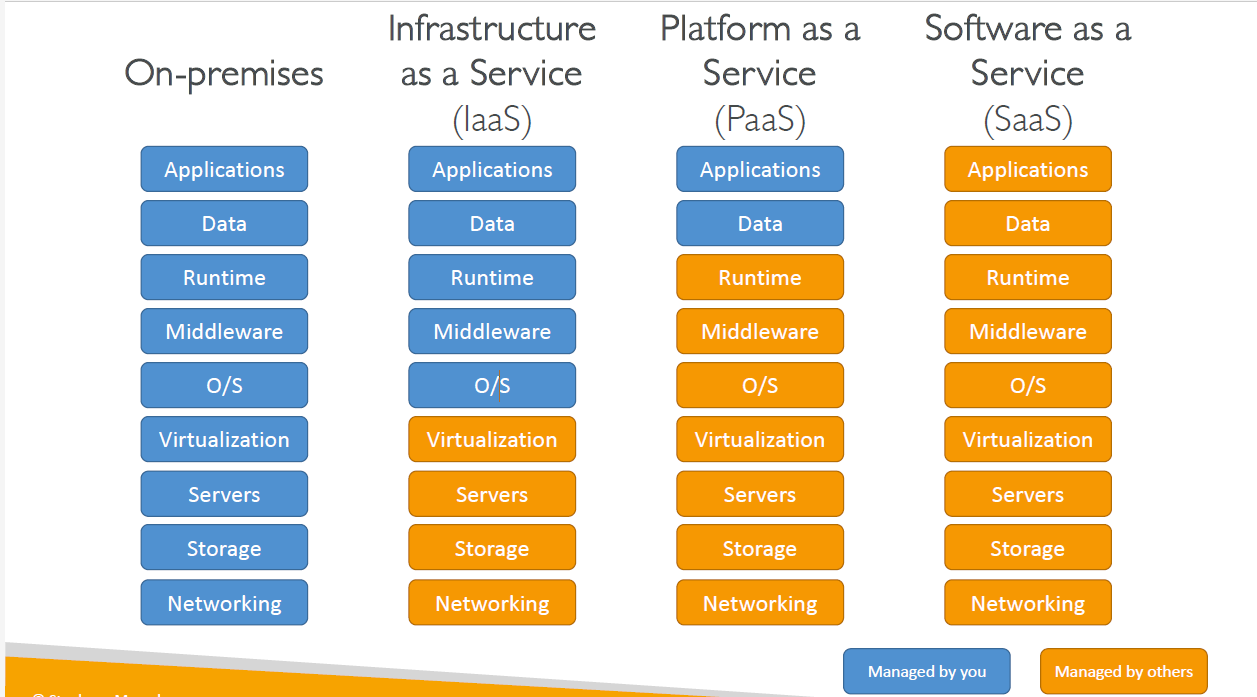
PaaS eliminates the need to invest in on-premises hardware and the anxiety associated with creating a virtual environment in which to run your apps.

In PaaS basically all IT resources and middleware are managed by cloud owner

**SaaS**

Software as a Service (SaaS) is a cloud-based software model that delivers applications to end-users through an internet browser. SaaS vendors host services and applications for customers to access on-demand. With a SaaS offering, you do not have to think about how the service is maintained or how the underlying infrastructure is managed; you only need to think about how you will use the software.

In everything is managed by cloud owner, e.g – gmail, outlook, office365



**AWS Region**

AWS Regions are large and widely dispersed into separate geographic locations, i.e region are geographic location.

**Availability Zone**

Availability Zones are distinct locations within an AWS Region that are engineered to be isolated from failures in other Availability Zones.

In a region each availability zone are isolated in power supply, colling etc but interconnected by high bandwidth link.

**Edge location:**

An edge location is the nearest point to the consumer(user) who is consuming the AWS service. In these locations, the server is not present but a small setup is there.

They are in major cities around the world. Unlike Availability Zones which are Physical Locations where AWS servers Lies, An Edge location is basically a small setup in different locations.AWS Edge location is the place where the data is cached to reduce the latency to the end users.



**What is IAM**

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources that will be used by any users.

You use IAM to control (read access, write access etc ) who is authenticated (signed in) and authorized (has permissions) to use resources.

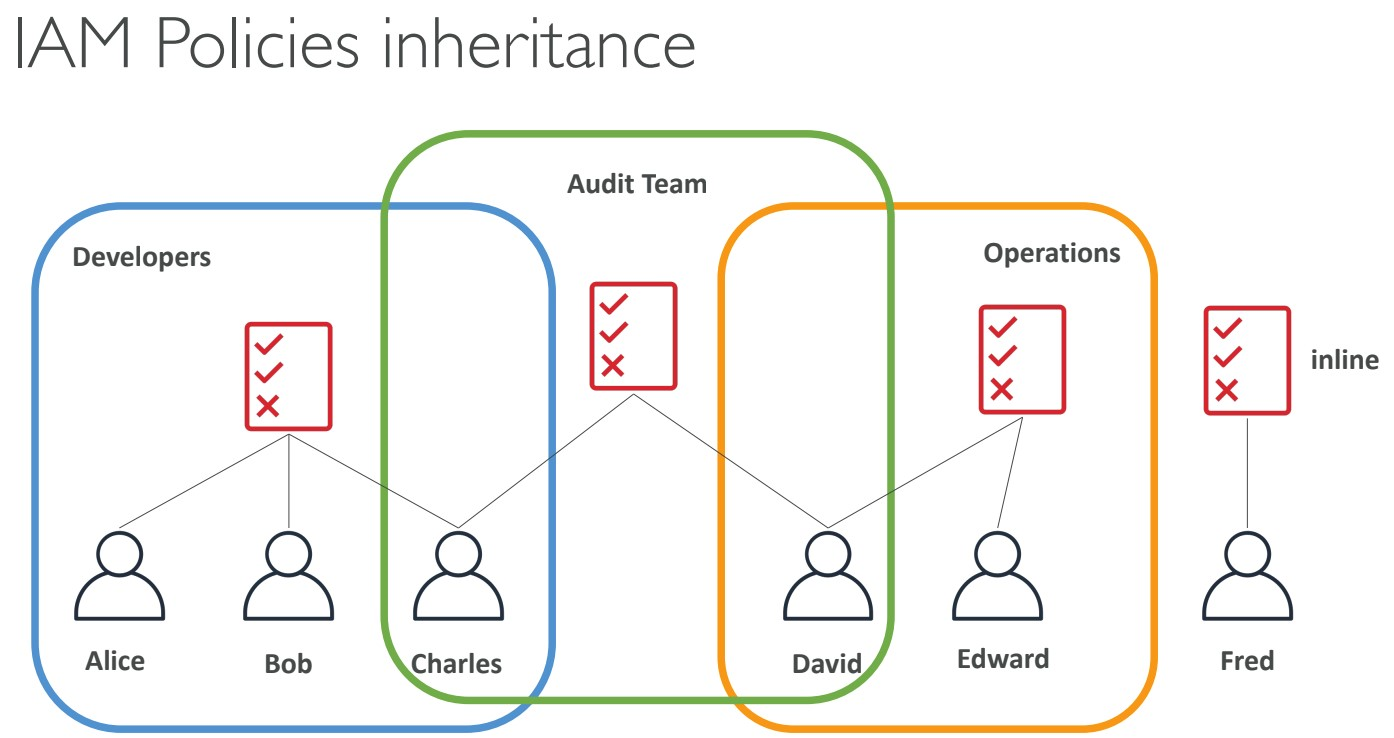
**IAM: Users & Groups**

* **IAM** = Identity and Access Management, Global service
* Root account created by default, should not be used or shared
* Users are people within your organization, and can be grouped
* Groups only contain users, not other groups
* Users don't have to belong to a group, and user can belong to multiple groups

**IAM: Permissions**

* Users or Groups can be assigned JSON documents called policies
* These policies define the permissions of the users
* In AWS you apply the least privilege principle: don’t give more permissions than a user needs

**IAM Policies inheritance**



**IAM Policies Structure**

* **Consists of**

1. **Version**: policy language version, always include “2012-10-17”
2. **Id**: an identifier for the policy (optional)
3. **Statement**: one or more individual statements (**required**)

* **Statements consists of**

1. **Sid**: an identifier for the statement (optional)
2. **Effect**: whether the statement allows or denies access (Allow, Deny)
3. **Principal**: account/user/role to which this policy applied to
4. **Action**: list of actions this policy allows or denies
5. **Resource**: list of resources to which the actions applied to
6. **Condition**: conditions for when this policy is in effect (optional)

Example:



**IAM – Password Policy**

* Strong passwords = higher security for your account
* Allow all IAM users to change their own passwords
* Require users to change their password after some time (password expiration)
* Prevent password re-use
* In AWS, you can setup a password policy

1. Set a minimum password length
2. Require specific character types:
3. including uppercase letters
4. lowercase letters
5. numbers
6. non-alphanumeric characters

**IAM Roles for Services**

* Some AWS service will need to perform actions on your behalf
* To do so, we will assign permissions to AWS services with IAM Roles
* Common roles:

1. EC2 Instance Roles
2. Lambda Function Roles
3. Roles for CloudFormation

**IAM Security Tools**

* IAM Credentials Report (account-level).
* A report that lists all your account's users and the status of their various credentials.
* IAM Access Advisor (user-level).
* Access advisor shows the service permissions granted to a user and when those services were last accessed.
* You can use this information to revise your policies.

**IAM Guidelines & Best Practices**

* Don’t use the root account except for AWS account setup
* One physical user = One AWS user
* **Assign users to groups** and assign permissions to groups
* Create a **strong password policy**
* Use and enforce the use of **Multi Factor Authentication (MFA)**
* Create and use Roles for giving permissions to AWS services
* Use Access Keys for Programmatic Access (CLI / SDK)
* Audit permissions of your account with the IAM Credentials Report
* **Never share IAM users & Access Keys**

**Shared Responsibility Model for IAM**

| **AWS** | **YOU** |
| --- | --- |
| Infrastructure (global network security) | Users, Groups, Roles, Policies management and monitoring |
| Configuration and vulnerability analysis | Enable MFA on all accounts |
| Compliance validation | Rotate all your keys often, Use IAM tools to apply appropriate permissions, Analyze access patterns & review permissions |

## Multi Factor Authentication - MFA

* Users have access to your account and can possibly change configurations or delete resources in your AWS account
* You want to protect your Root Accounts and IAM users
* MFA = password you know + security device you own
* Main benefit of MFA: if a password is stolen or hacked, the account is not compromised

## MFA devices options in AWS

* Virtual MFA device (Support for multiple tokens on a single device.)
  + Google Authenticator (phone only)
  + Authy (multi-device)
* Universal 2nd Factor (U2F) Security Key (Support for multiple root and IAM users using a single security key)
  + YubiKey by Yubico (3rd party)
* Hardware Key Fob MFA Device
* Hardware Key Fob MFA Device for AWS GovCloud (US)

## How can users access AWS ?

* To access AWS, you have three options:
  + AWS Management Console (protected by password + MFA)
  + AWS Command Line Interface (CLI): protected by access keys
  + AWS Software Developer Kit (SDK) - for code: protected by access keys
* Access Keys are generated through the AWS Console
* Users manage their own access keys
* Access Keys are secret, just like a password. Don’t share them
* Access Key ID ~= username
* Secret Access Key ~= password

## What’s the AWS CLI?

* A tool that enables you to interact with AWS services using commands in your command-line shell
* Direct access to the public APIs of AWS services
* You can develop scripts to manage your resources
* It’s open-source <https://github.com/aws/aws-cli>
* Alternative to using AWS Management Console

## What’s the AWS SDK?

* AWS Software Development Kit (AWS SDK)
* Language-specific APIs (set of libraries)
* Enables you to access and manage AWS services programmatically
* Embedded within your application
* Supports
  + SDKs (JavaScript, Python, PHP, .NET, Ruby, Java, Go, Node.js, C++)
  + Mobile SDKs (Android, iOS, …)
  + IoT Device SDKs (Embedded C, Arduino, …)
* Example: AWS CLI is built on AWS SDK for Python

## IAM Section – Summary

* **Users:** mapped to a physical user, has a password for AWS Console
* **Groups:** contains users only
* **Policies:** JSON document that outlines permissions for users or groups
* **Roles:** for EC2 instances or AWS services
* **Security:** MFA + Password Policy
* **AWS CLI:** manage your AWS services using the command-line
* **AWS SDK:** manage your AWS services using a programming language
* **Access Keys:** access AWS using the CLI or SDK
* **Audit:** IAM Credential Reports & IAM Access Advisor

**######################## Amazon Elastic Compute Cloud (EC2) ###############################**

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) Cloud.

Amazon **EC2 provides cloud hosted virtual machines, called "instances", to run applications**.

It includes all basic requirement (hardware, software) – RAM, ROM, CPU etc.

* EC2 is one of the most popular of AWS offering
* EC2 = Elastic Compute Cloud = **Infrastructure as a Service**
* It mainly consists in the capability of:
  + Renting virtual machines (EC2)
  + Storing data on virtual drives (**EBS - Elastic Block Store** )
  + Distributing load across machines (**ELB – Elastic Load Balancing**)
  + Scaling the services using an auto-scaling group (**ASG – Auto Scaling Group**)
* Knowing EC2 is fundamental to understand how the Cloud works

**Note**:

Amazon EC2 currently supports a variety of **operating systems including**- Linux, Ubuntu, Windows Server, Red Hat Enterprise Linux, SUSE Linux Enterprise Server, openSUSE Leap, Fedora, Fedora CoreOS, Debian, CentOS, Gentoo Linux, Oracle Linux, and FreeBSD.

**Feature/Benefits:**

* **Auto-Scaling** - Providing resources (instances, RAM, cpu etc) on demand **using ASG and ELB**. We can scale up or down bases on demand.
* **Pay-as-you-go** - AWS provides hourly rental and other pricing option based on your usage, that means we will pay only for our usages.
* **Increased Reliability** - We can store our data, host our application on multiple Region, AZ that makes high reliable our application, data.
* **Less procurement time** - In AWS services are just a click away that means we can quickly procure any services.
* **Security, maintenance** - In AWS servers’ maintenance, security etc are maintained by AWS and are highly secure.

**AWS pricing options:**

* On-demand
* Spot instances (discounted short-term capacity if available)
* Savings Plans (commitment to a certain amount of usage)
* Reserved Instances (discounted reserved long-term capacity)
* Dedicated Hosts (on-demand hourly or by reserved instances)

**EC2 sizing & configuration options**

* Operating System (OS): Linux, Windows or Mac OS
* How much compute power & cores (CPU)
* How much random-access memory (RAM)
* How much storage space:
  + Network-attached (EBS & EFS)
  + hardware (EC2 Instance Store)
* Network card: speed of the card, Public IP address
* Firewall rules: **security group**
* Bootstrap script (configure at first launch): EC2 User Data

**What is Bootstrapping in AWS?**

Bootstrapping in AWS simply means to add commands or scripts to AWS EC2’s instance User Data section that can be executed when the instance starts.

**EC2 User Data**

* It is possible to bootstrap our instances using an **EC2 User data** script.
* **bootstrapping** means launching commands when a machine starts
* That script is **only run once** at the instance **first start**
* EC2 user data is used to automate boot tasks such as:
  + Installing updates
  + Installing software
  + Downloading common files from the internet
  + Anything you can think of
* The EC2 User Data Script runs with the root user

### EC2 Instance Types - Overview

* You can use different types of EC2 instances that are optimised for different use cases (<https://aws.amazon.com/ec2/instance-types/>)
  + [General Purpose](https://github.com/kananinirav/AWS-Certified-Cloud-Practitioner-Notes/blob/master/sections/ec2.md#general-purpose)
  + [Compute Optimized](https://github.com/kananinirav/AWS-Certified-Cloud-Practitioner-Notes/blob/master/sections/ec2.md#compute-optimized)
  + [Memory Optimized](https://github.com/kananinirav/AWS-Certified-Cloud-Practitioner-Notes/blob/master/sections/ec2.md#memory-optimized)
  + [Storage Optimized](https://github.com/kananinirav/AWS-Certified-Cloud-Practitioner-Notes/blob/master/sections/ec2.md#storage-optimized)
  + Accelerated Computing
* AWS has the following naming convention: m5.2xlarge
* m: instance class
* 5: generation (AWS improves them over time)
* 2xlarge: size within the instance class

#### **General Purpose**

The computation, memory, and networking resources in general-purpose instances are balanced. General Purpose Instances **can be used for gaming servers, small databases, personal projects, etc**

* Great for a diversity of workloads such as web servers or code repositories
* Balance between:
  + Compute
  + Memory
  + Networking

#### **Compute Optimized**

Compute-optimized instances are appropriate for applications that **require a lot of computation and need high-performance CPUs.**

* Great for compute-intensive tasks that require high performance processors:
  + **Batch processing workloads**
  + Media transcoding
  + High performance web servers
  + High performance computing (HPC)
  + **Scientific modelling & machine learning, data science application**
  + Dedicated gaming servers

#### **Memory Optimized**

Memory-optimized instances are geared for workloads that **need huge datasets/informations/data to be processed in memory (RAM)**.

* Fast performance for workloads that process large data sets in memory
* Use cases:
  + High performance, relational/non-relational databases
  + Distributed web scale cache stores
  + In-memory databases optimized for BI (business intelligence)
  + Applications performing real-time processing of big unstructured data

#### **Storage Optimized**

Storage-optimized instances are made for workloads that demand **fast, sequential read and write access to huge datasets**. Best for - Distributed file systems, data warehousing applications, and high-frequency online transaction processing (OLTP) systems.

* Great for storage-intensive tasks that require high, sequential read and write access to large data sets on local storage
* Use cases:
  + **High frequency online transaction processing (OLTP) systems**
  + Relational & NoSQL databases
  + Cache for in-memory databases (for example, Redis)
  + **Data warehousing applications**
  + **Distributed file systems**

**Accelerated Computing Instancesss**

Coprocessors are used in accelerated computing instances to execute specific operations more effectively than software running on CPUs.

Best for - **Scientific calculation, AI and ML modelling etc**

**#######################################################################################**

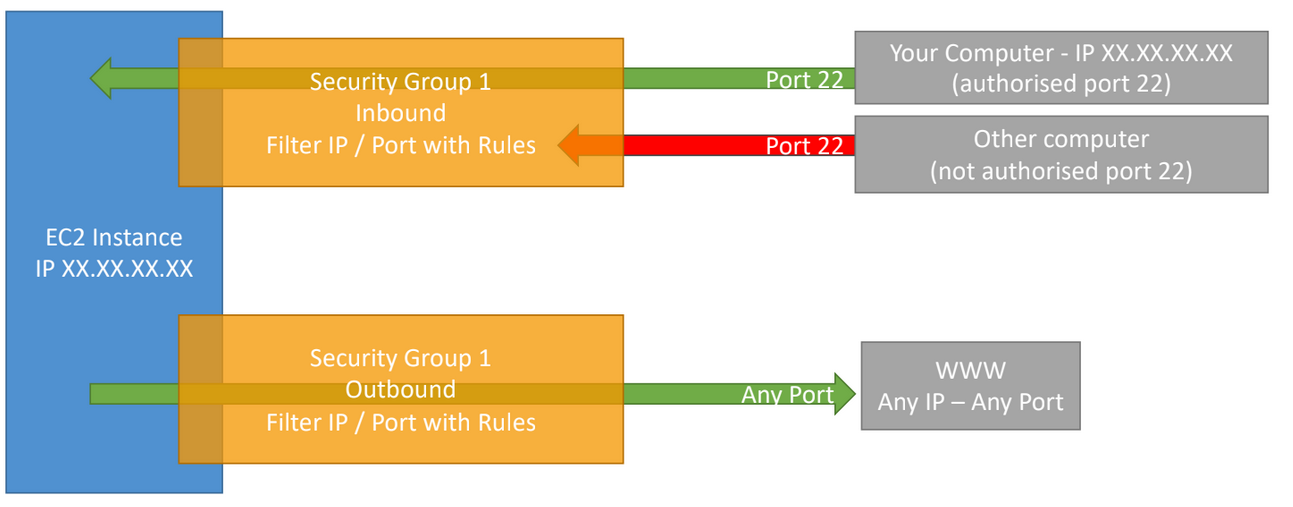
## **Introduction to Security Groups**

* Security Groups are the fundamental of network security (like firewall) in AWS
* They control how traffic is allowed into or out of our EC2 Instances.
* Security groups only contain allow rules
* Security groups rules can reference by IP or by security group

**Deeper Dive**

* Security groups are acting as a “firewall” on EC2 instances
* They regulate:
  + Access to Ports
  + Authorised IP ranges – IPv4 and IPv6
  + Control of inbound network (from other to the instance)
  + Control of outbound network (from the instance to other

### Security Groups Diagram



**Good to know**

* Can be attached to multiple instances
* Locked down to a region / VPC combination
* Does live “outside” the EC2 – if traffic is blocked the EC2 instance won’t see it
* It’s good to maintain one separate security group for SSH access
* If your application is not accessible (time out), then it’s a security group issue
* If your application gives a “connection refused“ error, then it’s an application error or it’s not launched
* All inbound traffic is **blocked** by default
* All outbound traffic is **authorized** by default

## Classic Ports to know

* 22 = SSH (Secure Shell) - log into a Linux instance
* 21 = FTP (File Transfer Protocol) – upload files into a file share
* 22 = SFTP (Secure File Transfer Protocol) – upload files using SSH
* 80 = HTTP – access unsecured websites
* 443 = HTTPS – access secured websites
* 3389 = RDP (Remote Desktop Protocol) – log into a Windows instance

**Amazon S3**

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance.

S3 provides web interfaces to store files. Our data files (image files, video files, text files, etc.) are stored as **objects** within **buckets**. Buckets serve as containers for objects.

**S3 use cases: -**

* Backup and storage
* Disaster Recovery
* Archive
* Hybrid Cloud storage
* Application hosting
* Media hosting
* Data lakes & big data analytics
* Software delivery
* Static website

**Bucket**

A bucket is a container for objects (any file which we can to store) stored in Amazon S3. You can store any number of objects in a bucket and can have up to 100 buckets(can be increased on request) in your account.

* Amazon S3 allows people to store objects (files) in “buckets” (directories)
* Buckets must have a globally unique name (across all regions all accounts)
* Buckets are defined at the region level
* S3 looks like a global service but buckets are created in a region
* Naming convention-
  + No uppercase
  + No underscore
  + 3-63 characters long
  + Not an IP
  + Must start with lowercase letter or number

**Note:**

We can sync buckets in same or different region (CRR) using S3 Batch Replication.

**Objects**

Objects are the fundamental entities stored in Amazon S3. Objects consist of object data and metadata.

The metadata is a set of name-value pairs that describe the object. These pairs include some default metadata, such as the date last modified, and standard HTTP metadata, such as Content-Type. You can also specify custom metadata at the time that the object is stored.

**An object is uniquely identified within a bucket by a key (name) and a version ID**.

**Key**

An object key (or key name) is the unique identifier for an object within a bucket. Every object in a bucket has exactly one key. It is basically full path of object (stored files). E.g-

* //my-bucket/my\_file.txt
* //my-bucket/my\_folder1/another\_folder/my\_file.txt

**The combination of a bucket, object key, and optionally, version ID (if S3 Versioning is enabled for the bucket) uniquely identify each object.**

 The key is composed of **prefix** + **object name**

* s3://my-bucket/my\_folder1/another\_folder/my\_file.txt

 There is no concept of “directories” within buckets (although the UI will trick you to think otherwise)

 Just keys with very long names that contain slashes (“/”)

 Object values are the content of the body:

* Max Object Size is 5TB (5000GB)
* If uploading more than 5GB, must use “multi-part upload”

 Metadata (list of text key / value pairs – system or user metadata)

* Tags (Unicode key / value pair – up to 10) – useful for security / lifecycle
* Version ID (if versioning is enabled)

**S3 Versioning**

You can use S3 Versioning to keep multiple variants of an object in the same bucket. With S3 Versioning, you can preserve, retrieve, and restore every version of every object stored in your buckets. You can easily recover from both unintended user actions and application failures.

**Version ID**

When you enable S3 Versioning in a bucket, Amazon S3 generates a unique version ID for each object added to the bucket. Objects that already existed in the bucket at the time that you enable versioning have a version ID of null.

**S3 Secutiry**

We can configure the security ( read/ write etc access ) of s3 bucket in below way-

* **User based**

In this method we can assign role/access/policies (predefined or user defined) to IAM user who can access s3 buckets (this for user level.)

IAM policies - which API calls should be allowed for a specific user from IAM console

* **Resource Based**

In this we assign which user can access bucket and which object(s) within a bucket by creating policy using JSON file. (this is bucket level)

* + Bucket Policies - bucket wide rules from the S3 console - allows cross account
  + Object Access Control List (ACL) – finer grain(steps- bucket –> permission –> Bucket ACL )
  + Bucket Access Control List (ACL) – less common
* Note: an IAM principal can access an S3 object if-
  + The user IAM permissions allow it OR the resource policy ALLOWS it
  + AND there is no explicit DENY
* Encryption: encrypt objects in Amazon S3 using encryption keys

**S3 Bucket Policies- (kind of resource-based security/policy)**

S3 bucket policies are created in JSON file. This is most common to define access for any user.

These polices/rules are used to control the access of bucket objects and upload action (encryption on object upload ect)

When we create S3 bucket polices which is JSON file it contains below parameter.

* JSON based policies
  + **Resources**: this is basically buckets and objects
  + **Actions**: Set of API(AWS bucket permission e.g- GetObject, GetBucketAcl) to Allow or Deny
  + **Effect**: Allow / Deny
  + **Principal**: The account or user to apply the policy.
* Use S3 bucket for policy to:
  + Grant public access to the bucket
  + Force objects to be encrypted at upload
  + Grant access to another account (Cross Account)

**How to configure bucket policy:**

Go to bucket ---- > permission ------ > Bucket Policy , click on EDIT in front of it ----- > Policy generator ----- > Fill the details of form and JSON policy will be generated ---- > Copy and paste in Bucket Policy and save it.



**S3 Standard General Purpose**

* 99.99% Availability
* Used for frequently accessed data
* Low latency and high throughput
* Sustain 2 concurrent facility failures
* Use Cases: Big Data analytics, mobile & gaming applications, content distribution…

**Types of S3 storage classes**

We can divide s3 storage classes in below types-

1. S3 Infrequent access (IA)
2. One Zone-Infrequent Access (S3 One Zone-IA)
3. S3 Glacier Instant Retrieval
4. S3 Glacier Flexible Retrieval (formerly Amazon S3 Glacier)
5. S3 Glacier Deep Archive - for long term storage
6. S3 Intelligent-Tiering

We can define our storage class while uploading the files/data or can change for existing file/objects in bucket.

**Assign storage class (while uploading objects in bucket)**

Go to bucket --- > upload files ---- > properties (Select storage class)

**Assign storage class (existing objects in bucket)**

Go to bucket and object (click on object) -- > scroll down , click on Edit in front of Storage class and modify.

**S3 Infrequent access (IA)**

S3 Standard-IA **is for data that is accessed less frequently, but requires rapid access when needed.**

* high durability, high throughput, and low latency of S3 Standard, with a low per GB storage price and per GB retrieval charge.
* 99.9% availability.
* Use cases: Disaster Recover, backup.

**Amazon S3 One Zone-Infrequent Access (S3 One Zone-IA)**

* High durability in a single Availability Zone
* 99.5% availability
* Use cases: Storing secondary backup copies of on-premise data, or data you can recreate

**Amazon S3 Glacier Instant Retrieval**

This is mainly for storing the archival data. Great for accessed data once a quarter.

* Millisecond retrieval.
* minimum storage duration of 90 days

**Amazon S3 Glacier Flexible Retrieval (formerly Amazon S3 Glacier)**

S3 Glacier Flexible Retrieval **delivers low-cost storage, up to 10% lower cost (than S3 Glacier Instant Retrieval)**, for archive data that is **accessed 1—2 times per year and is retrieved asynchronously**.

* Expedited (1 to 5 minnures), standard ( 3 to 5 minutes), bulk (5 to 12 hours)
* Minimum storage duration of 90 days

**Amazon S3 Glacier Deep Archive - for long term storage**

S3 Glacier Deep Archive is Amazon S3’s lowest-cost storage class and supports long-term retention and digital preservation for data **that may be accessed once or twice in a year**. It is designed for customers—particularly those in highly-regulated industries, such as financial services, healthcare, and public sectors—that **retain data sets for 7—10 years or longer** to meet regulatory compliance requirements

* Standard (12 hours), Bulk (48 hours)
* Minimum storage duration of 180 days

**S3 Intelligent-Tiering**

* Small monthly monitoring and auto-tiering fee
* Moves objects automatically between Access Tiers based on usage
* There are no retrieval charges in S3 Intelligent-Tiering
* Frequent Access tier (automatic): default tier
* Infrequent Access tier (automatic): objects not accessed for 30 days
* Archive Instant Access tier (automatic): objects not accessed for 90 days
* Archive Access tier (optional): configurable from 90 days to 700+ days
* Deep Archive Access tier (optional): config. from 180 days to 700+ days

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | |  | **S3 Standard** | **S3 Intelligent-Tiering\*** | **S3 Standard-IA** | **S3 One Zone-IA†** | **S3 Glacier Instant Retrieval** | **S3 Glacier Flexible Retrieval** | **S3 Glacier Deep Archive** | | Designed for durability | 99.999999999% (11 9’s) | 99.999999999% (11 9’s) | 99.999999999% (11 9’s) | 99.999999999% (11 9’s) | 99.999999999% (11 9’s) | 99.999999999% (11 9’s) | 99.999999999% (11 9’s) | | Designed for availability | 99.99% | 99.9% | 99.9% | 99.5% | 99.9% | 99.99% | 99.99% | | Availability SLA | 99.9% | 99% | 99% | 99% | 99% | 99.9% | 99.9% | | Availability Zones | ≥3 | ≥3 | ≥3 | 1 | ≥3 | ≥3 | ≥3 | | Minimum  capacity charge per object | N/A | N/A | 128 KB | 128 KB | 128 KB | N/A | N/A | | Minimum storage duration charge | N/A | N/A | 30 days | 30 days | 90 days | 90 days | 180 days | |

**S3 replication**

S3 Replication is a fully-managed feature available for Amazon Simple Storage Service (S3) customers. It can automatically replicate S3 objects to another region’s bucket or same region bucket.

There are two type of Replication-

1. Corss-Region Replication (CRR)

2. Same-Region Replication (SRR)

**Note:**

S3 replication works for any bucket if versioning is enabled for that bucket.

**Cross-Region Replication (CRR)** — copies S3 objects across multiple regions , representing geographically separate Amazon data centres.

**Steps.**

Go to source bucket --- > Management (enable replication) ------> create replication rule ------ > Select bucket in another region

**Same-Region Replication (SRR)** —copies S3 objects between buckets in different availability zones (AZs), which are separate data centres in the same AR.

**Shared Responsibility Model for S3**

| **AWS** | **YOU** |
| --- | --- |
| * Infrastructure (global security, durability, availability, sustain concurrent loss of data in two facilities) | * S3 Versioning, S3 Bucket Policies, S3 Replication Setup |
| * Configuration and vulnerability analysis | * Logging and Monitoring, S3 Storage Classes |
| * Compliance validation | * Data encryption at rest and in transit |

**AWS Snow Family**

AWS Snow Family is a group of devices that transport data in and out of AWS.

The Snow Family (comprised of AWS Snowcone, Snowball, and AWS Snowmobile) oﬀers several physical devices and capacity profiles, most with built-in computing capabilities.

**Data migration**:

* Snowcone
* Snowball Edge
* Snowmobile

**Edge computing**:

* Snowcone
* Snowball Edge

**Snowball Edge (for data transfers)**

* Snowball is device **used for edge computing and data transfer**
* Physical data transport solution: move TBs or PBs of data in or out of AWS
* Alternative to moving data over the network (and paying network fees)
* Pay per data transfer job
* Provide block storage and Amazon S3-compatible object storage
* **Snowball Edge Storage Optimized**
  + 80 TB of HDD capacity for block volume and S3 compatible object storage
* **Snowball Edge Compute Optimized**
  + 42 TB of HDD capacity for block volume and S3 compatible object storage
* Use cases: large data cloud migrations, DC decommission, disaster recovery

Each Snowball Edge device **can transport data at speeds faster than the internet**. This transport is done by shipping the data in the appliances through a regional carrier. The appliances are rugged, complete with E Ink shipping labels.

**AWS Snowcone**

* Small, portable computing, anywhere, rugged & secure, **for edge computing and data transfer**.
* Light (4.5 pounds, 2.1 kg)
* **Device used for edge computing, storage, and data transfe**r
* 8 TBs of usable storage
* Use Snowcone where Snowball does not fit (space-constrained environment)
* Must provide your own battery / cables
* Can be sent back to AWS offline, or connect it to internet and use AWS DataSync to send data

**AWS Snowmobile**

* Transfer exabytes of data (1 EB = 1,000 PB = 1,000,000 TBs)
* Each Snowmobile has 100 PB of capacity (use multiple in parallel)
* High security: temperature controlled, GPS, 24/7 video surveillance
* Better than Snowball if you transfer more than 10 PB

| **Properties** | **Snowcone** | **Snowball Edge Storage Optimized** | **Snowmobile** |
| --- | --- | --- | --- |
| Storage Capacity | 8 TB usable | 80 TB usable | < 100 PB |
| Migration Size | Up to 24 TB, online and offline | Up to petabytes, offline | Up to exabytes, offline |

**Edge Computing**

* Process data while it’s being created on an edge location
  + A truck on the road, a ship on the sea, a mining station underground...
* These locations may have
  + Limited / no internet access
  + Limited / no easy access to computing power
* We setup a Snowball Edge / Snowcone device to do edge computing
* Use cases of Edge Computing:
  + Preprocess data
  + Machine learning at the edge
  + Transcoding media streams
* Eventually (if need be) we can ship back the device to AWS (for transferring data for example)

**Snow Family - Edge Computing**

* Snowcone (smaller)
  + 2 CPUs, 4 GB of memory, wired or wireless access
  + USB-C power using a cord or the optional battery
* Snowball Edge – Compute Optimized
  + 52 vCPUs, 208 GiB of RAM
  + Optional GPU (useful for video processing or machine learning)
  + 42 TB usable storage
* Snowball Edge – Storage Optimized
  + Up to 40 vCPUs, 80 GiB of RAM
  + Object storage clustering available
* All: Can run EC2 Instances & AWS Lambda functions (using AWS IoT Greengrass)
* Long-term deployment options: 1 and 3 years discounted pricing

**AWS OpsHub**

AWS OpsHub is **a graphical user interface you can use to manage AWS Snowball devices(storage, compute)**

* Unlocking and configuring single or clustered devices
* Transferring files
* Launching and managing instances running on Snow Family Devices
* Monitor device metrics (storage capacity, active instances on your device)
* Launch compatible AWS services on your devices (ex: Amazon EC2 instances, AWS DataSync, Network File System (NFS))

**########################################################################################### DataBase and Analytics #####**

**#######################################################################################**

On high level we can classify the database in two types—

1. SQL Database or Relational Database
2. NoSQL database or Non-Relational Database

**SQL Database:**

It is used when we want to store data in tabular form.

In SQL database **data have schema, datatype, length which add a constraint/limitation**.

In almost all situations **SQL databases are vertically scalable** other hand, NoSQL databases are horizontally scalable.

It is best choice when working with related data.

Some popular SQL databases are:-

* MySQL
* MariaDB
* Oracle
* Microsoft SQL server
* Aurora ( AWS proprietary )

**NoSQL database**

A NoSQL database has a dynamic schema(means no schema, data type and length) for unstructured data.

Data can be stored in document, JSON or key-value format.

NoSQL database **are horizontally scalable**.

Best choice when the **dealing with huge data and quick availability**.

**Relational Database Services:**

In AWS we have below advantage of RDS over EC2

* RDS is managed/fully manages serviced that means-
* Automated provisioning, OS patching.
* Continuous backup and restore to specific timestamp.
* Monitoring dashboard
* Read replicas for improved read performance
* Multi AZ setup for DR(disaster recovery)
* maintenance windows for upgrade
* Scaling capability
* Storage backed by EBS

**Drawback:**

We **can’t SSH into our instance**

| **SQL** | **NoSQL** |
| --- | --- |
| RELATIONAL DATABASE MANAGEMENT SYSTEM (RDBMS) | Non-relational or distributed database system. |
| These databases have fixed or static or predefined schema | They have a dynamic schema |
| These databases are not suited for hierarchical data storage. | These databases are best suited for hierarchical data storage. |
| These databases are best suited for complex queries | These databases are not so good for complex queries |
| Vertically Scalable | Horizontally scalable |
| Follows ACID property | Follows CAP(consistency, availability, partition tolerance) |
| **Examples:** [MySQL](https://www.geeksforgeeks.org/mysql-common-mysql-queries/), [PostgreSQL](https://www.geeksforgeeks.org/what-is-postgresql-introduction/), Oracle, MS-SQL Server, etc | **Examples:** [MongoDB](https://www.geeksforgeeks.org/mongodb-tutorial/), HBase, Neo4j, Cassandra, etc |

**Amazon Aurora**

* It is proprietary of AWS
* PostgreSQL and MYSQL are both supported as AWS Aurora
* Aurora is AWS cloud optimized and claims 5X performance improvement over MYSQL on RDX and 3X performance of Postgres on RDX.
* Aurora storage automatically grows in increments of 10GB, upto 128 TB
* Aurora is more cost efficient (20% cheaper than other RDS)

**### RDS Deployment ####**

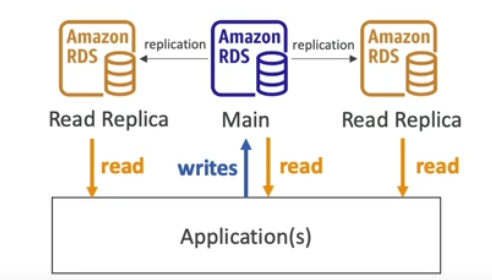
**Read replica**

Read replica is the copy of your DB from where we can read the data.

This is **used to reduce the load on your DB.** **We can create upto 15 read replica**.

While doing the write operation data is always written into main DB.

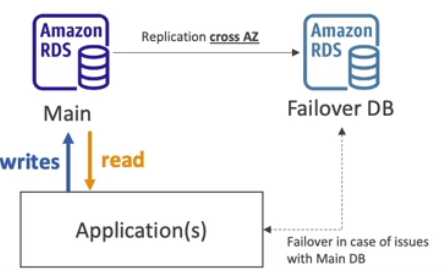
**Read replicas are available for Amazon RDS** ( MySQL, MariaDB, PostgreSQL, Oracle, SQL, Amazon Aurora )



**Multi-AZ**

Multi-AZ is mainly used to be fail safe. (high availability and failover for any worst case)

In this case main database is for read/write. **Read/write happens in failover DB in case of failover situation**.

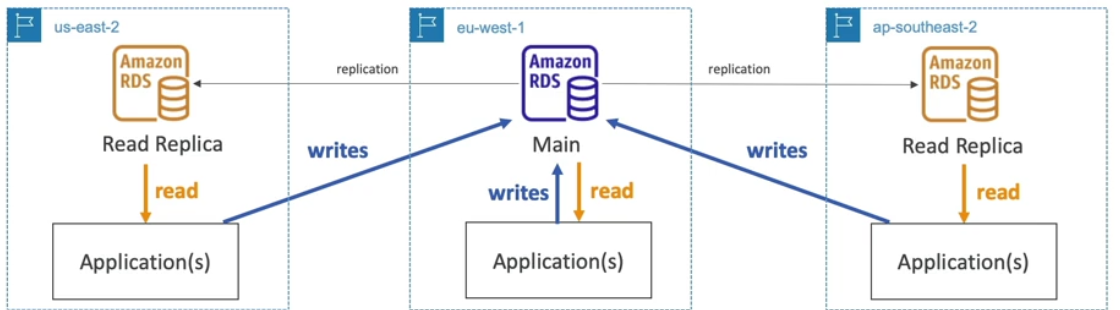


**Multi-Region**

Disaster recovery in case of region issue.

Local performance for global issue.

Involve replication cost

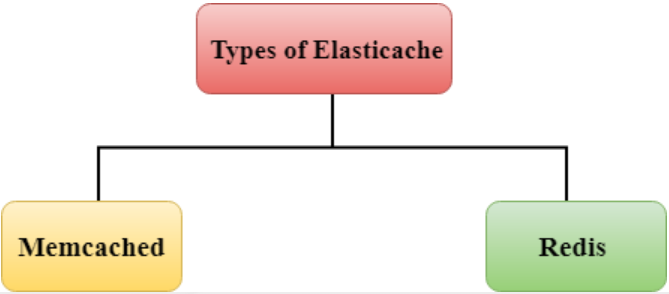


**Amazon ElasticCache**

It this another type of AWS service in AWS.

In elastic cache data are stored in memory instead of disk which make it extremely fast.

* The same way RDS is to get manager **Relation Databases**...
* ElatiCache is way of storing database data in memory to server extremely fast read operation.
* Caches are in-memory database with high performance, low latency.
* Helps to reduce load off database for read intensive workloads.
* AWS takes case of OS Patching/ maintenance/ optimization, setup, configuration, monitoring, failure recovery and backups



Memcached --- Memory cached

Redis --- Remote Dictionary Server

**DynamoDB ---- NoSQL**

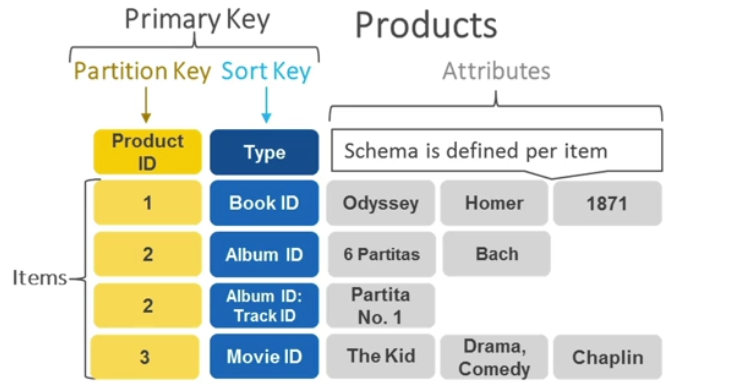
Amazon DynamoDB is a extremely fast (read operation) and flexible NoSQL database services.

In **AWS data are stored in key (partition key, sorting key etc which create primary key ) and value (attributes).**

**Features:**

* Fully Managed Highly available with replication across 3 AZ
* **Scales to massive workloads, distributed “serverless” database**
* Millions of requests per seconds, trillions of row, 100s of TB of storage
* Fast and consistent in performance
* **Single-digit millisecond latency – low latency retrieval**
* Integrated with IAM for security, authorization, and administration
* Low cost and auto scaling capabilities
* Standard & Infrequent Access (IA) Table Class

Example:



**DynamoDB Accelerator – DAX -- (NoSQL databse)**

* Fully managed in-memory cache for single DynamoDB.
* 10X performance improvement - single digit millisecond latency to microsecond latency.
* Secure, highly scalable, and highly available
* Difference with ElastiCache at CCP level- DAX is only used for and is integrated with DynamoBD, while ElastiCache can be used for other databases (SQL)

**Redshift Overview**

Amazon Redshift functions completely on SQL for data exploration and analysis. It uses ANSI SQL to create tables, load data, and perform data analytics

* Redshift is based on PostgreSQL (**relational DB**), **but it’s not used for OLTP (Online Transactional Processing)**
* **It’s for OLAP** – online analytical processing (analytics and data warehousing)
* **Load data once every hour, not every second**
* 10x better performance than other data warehouses, **scale to PBs of data**
* Columnar storage of data (instead of row based)
* Massively Parallel Query Execution (MPP), highly available
* Pay as you go based on the instances provisioned
* Has a SQL interface for performing the queries
* BI tools such as AWS Quicksight or Tableau integrate with it

**Amazon EMR**

Amazon EMR (formerly known as Amazon Elastic Map Reduce) is an **Amazon Web Services (AWS) tool for big data processing and analysis, it is based on Apache Hadoop**.

Amazon **EMR processes big data in Hadoop clusters** of virtual servers on Amazon Elastic Compute Cloud (EC2) and Amazon Simple Storage Service (S3).

Amazon EMR uses Hadoop (Hadoop cluster) on EC2 to distribute your data and processing across resizable clusters of Amazon EC2 instances.

Amazon EMR is a computing framework that runs on Hadoop. It also provides an SQL interface from Apache HIVE to query Amazon S3

**Feature of EMR**

* EMR helps creating Hadoop clusters (Big Data) to analyse and process vast amount of data
* The clusters can be made of hundreds of EC2 instances
* Also supports Apache Spark, HBase, Presto, Flink
* EMR takes care of all the provisioning and configuration
* Auto-scaling and integrated with Spot instances
* Use cases: data processing, machine learning, web indexing, big data

**EMR vs Redshit**

**When to use Redshift**.

Redshift is used when -

* **Traditional data warehouse**
* When you **need the data relatively hot for analytics such as BI**
* When there is no data engineering team
* When your queries require joins
* When you **need a cluster 24X7**
* When you data type are simple, i.e not Arrays, or Structs
* When data has no nested jsons
* When **you have petabyte scale database**
* When you **want analize massive amount of data (spectrum)**
* When you need update/delete
* When you require and ACID DBMS

**When to use EMR**

EMR is used when -

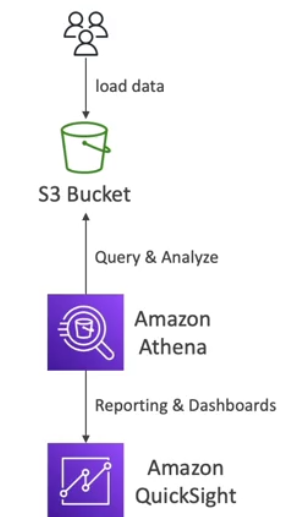
* When you **need a transient cluster, for night or hourly automation**
* When **compute elasticity is important (auto scaling on tasks)**
* When cost is important: spot instances.
* When you **data scales until a few hundred TB’s**
* When you want to decouple compute and storage (external table + task node + auto scaling). this is cloud architecture best practice.
* When you require more flexibility
  + Complex partitions + dynamic partitioning + insert overwrite. click on the link for an example.
  + Complex data type
  + Structs
  + Arrays <–> nested json
* Orchestration built in such as Oozie, although Airflow is more common.
* Notebook built in – mix your code with SQL via Zeppelin

**Amazon Athena**

Athena helps you analyse unstructured, semi-structured, and structured data stored in Amazon S3.

Athena integrates with Amazon QuickSight for easy data visualization. Properties are-

* **Serverless query service to analyse data stored in Amazon S3**.
* Uses standard SQL language to query the files.
* Supports CSV, JSON, ORC, Avro, and Parquet (built on Presto).
* Pricing: $5.00 per TB of data scanned.
* Use compressed or columnar data for cost-savings (less scan).
* Use cases: Business intelligence / analytics / reporting, analyze & query VPC Flow Logs, ELB Logs, CloudTrail trails, etc...
* To analyze data in S3 using serverless SQL, use Athena



**Amazon QuickSight**

Amazon **QuickSight is BI utility to analyse the data and creates the dashboard for visualization** and decision making.

* **Serverless machine learning-powered business intelligence service to create interactive dashboards**.
* Fast, automatically scalable, embeddable, with per-session pricing.
* Integrated (can read data from to create dashboard) with RDS, Aurora, Athena, Redshift, S3…
* Use cases:
  + Business analytics
  + Building visualizations
  + Perform ad-hoc analysis
  + Get business insights using data

**DocumentDB**

Like Auroa which is designed/compatible with PostgreSQL and MySQL, **DocumentDB is mongoDB compatible database in AWS, i.e it is serverless.**

* DocumentDB is for MongoDB (which is a NoSQL database) in AWS.
* MongoDB/DocumentDB is used to store, query, and index JSON data
* Similar “deployment concepts” as Aurora
* Fully Managed, highly available with replication across 3 AZ
* Aurora storage automatically grows in increments of 10GB, up to 64 TB.
* Automatically scales to workloads with millions of requests per seconds

**Amazon Neptune**

Fully managed graph-based database.

* A popular graph dataset would be a social network/media application
* Users have friends
  + Posts have comments
  + Comments have likes from users
  + Users share and like posts…
* Highly available across 3 AZ, with up to 15 read replicas
* Build and run applications working with highly connected datasets – optimized for these complex and hard queries
* Can store up to billions of relations and query the graph with milliseconds latency
* Highly available with replications across multiple AZs
* Great for knowledge graphs (Wikipedia), fraud detection, recommendation engines, social networking

**AWS Glue**

AWS Glue is a **serverless data integration service** that makes it easy **for analytics users to discover, prepare, move, and integrate data from multiple sources**.

AWS Glue is a fully managed ETL (extract, transform, and load) AWS service. One of its key abilities is to analyse and categorize data.

* Managed extract, transform, and load (ETL) service
* Useful to prepare and transform data for analytics
* Fully serverless service
* Glue Data Catalog: catalog of datasets
* Application - can be used by Athena, Redshift, EMR
* **Glue Data Catalog is where permanent metadata is stored.**

**Amazon QLDB**

QLDB stands for ”**Quantum Ledger Database”.** It is **best suites for financial application** where we want to keep each transaction history.

* A ledger is a book recording financial transactions
* Fully Managed, Serverless, High available, Replication across 3 AZ
* Used to review history of all the changes made to your application data over time
* **Immutable system**: if any entry is made then no entry can be removed or modified, cryptographically verifiable
* 2-3x better performance than common ledger blockchain frameworks, manipulate data using SQL
* Difference with Amazon Managed Blockchain: no decentralization component, in accordance with financial regulation

**######################################################################################**

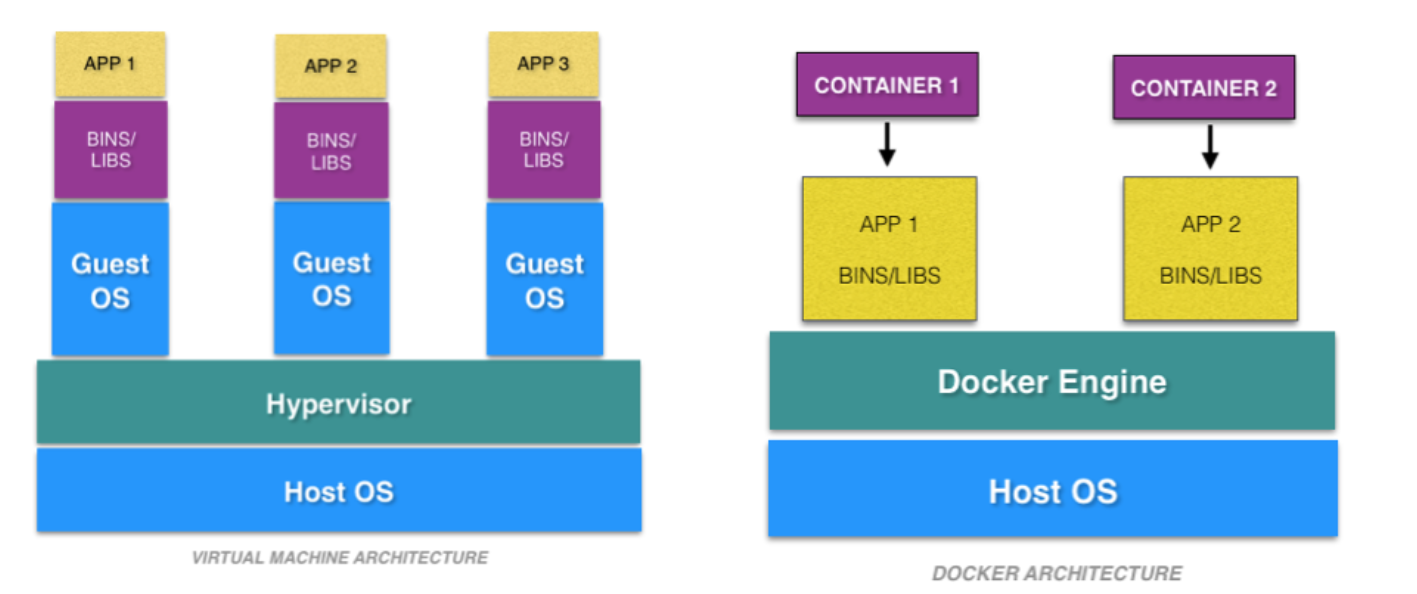
**# Other Compute #**

**######################################################################################**

**Docker**

Docker is **popular virtualization software that helps its users in developing, deploying, monitoring, and running applications in a Docker Container with all their dependencies**.

Docker containers **include all dependencies (frameworks, libraries, etc.)** to run an application in an efficient and bug-free manner.



VMs have the host OS and guest OS inside each VM. A guest OS can be any OS, like Linux or Windows, irrespective of the host OS. In contrast, Docker containers host on a single physical server with a host OS, which shares among them. Sharing the host OS between containers makes them light and decreases the boot time.

**Virtual environment vs Docker**

|  |  |  |
| --- | --- | --- |
|  | Docker | Virtual Machines (VMs) |
| **Boot-Time** | Boots in a few seconds. | It takes a few minutes for VMs to boot. |
| **Runs on** | Dockers make use of the execution engine. | VMs make use of the hypervisor. |
| **Memory Efficiency** | No space is needed to virtualize, hence less memory. | Requires entire OS to be loaded before starting the surface, so less efficient. |
| **Isolation** | Prone to adversities as no provisions for isolation systems. | Interference possibility is minimum because of the efficient isolation mechanism. |
| **Deployment** | Deploying is easy as only a single image, containerized can be used across all platforms. | Deployment is comparatively lengthy as separate instances are responsible for execution. |
| **Usage** | Docker has a complex usage mechanism consisting of both third party and docker managed tools. | Tools are easy to use and simpler to work with. |

* Docker is a software development platform to deploy apps
* Apps are packaged in containers that can be run on any OS
* Apps run the same, regardless of where they are run
  + Any machine
  + No compatibility issues
  + Predictable behaviour
  + Less work
  + Easier to maintain and deploy
  + Works with any language, any OS, any technology
* Scale containers up and down very quickly (seconds)

**Where Docker images are stored**

* Docker images are stored in Docker Repositories
* **Public: Docker Hub** https://hub.docker.com/
* Find base images for many technologies or OS:
  + Ubuntu
  + MySQL
  + NodeJS, Java…
* **Private: Amazon ECR** **(Elastic Container Registry**)

**ECS -- Amazon Elastic Container Service**

Amazon Elastic Container Service (Amazon ECS) is a container management service that can quickly launch, exit, and manage docker containers in a cluster. Some feature are-

* Launch Docker containers on AWS
* **You must provision & maintain the infrastructure** (the EC2 instances)
* **AWS takes care of starting / stopping containers**
* Has integrations with the Application Load Balancer

**Fargate**

AWS Fargate is a technology that you can use with Amazon ECS to run containers without having to manage servers or clusters of Amazon EC2 instances.

Farget is container management services where we don’t need to maintain infrastructure (EC2)

* Launch Docker containers on AWS
* You do not provision the infrastructure (no EC2 instances to manage) – simpler!
* Serverless offering
* AWS just runs containers for you based on the CPU / RAM you need

**Launch types of AWS ECS**

ECS can be launched in two following modes:-

**Fargate launch:** As discussed above, the fargate launch type, takes most of the responsibility from the users and takes in only the basic inputs like the CPU type**,** memory, and IAM policies from the user to run the application cluster.

**EC2 Launch:** This is a more customizable launch type. Users are responsible for the number of instances in the cluster, scaling their cluster, and more. This allows you to be more in control of your clusters which may be required for security reasons

|  |  |
| --- | --- |
|  |  |

**AWS lambda**

AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part of Amazon Web Services.

In AWS Lambda the code is executed based on the response of events in AWS services such as add/delete files in S3 bucket, HTTP request from Amazon API gateway, etc.

It is used for short time running jobs.

**AWS Lambda vs EC2**

| **EC2** | **Lambda** |
| --- | --- |
| Virtual Servers in the Cloud | Virtual functions – no servers to manage! |
| Limited by RAM and CPU | Limited by time - short executions |
| Continuously running | Run on-demand |
| Scaling means intervention to add / remove servers | Scaling is automated! |

**Benefit of AWS Lambda**

* Easy Pricing:
  + Pay per request and compute time
  + Free tier of 1,000,000 AWS Lambda requests and 400,000 GBs of compute time
* Integrated with the whole AWS suite of services
* Event-Driven: functions get invoked by AWS when needed
* Integrated with many programming languages
* Easy monitoring through AWS CloudWatch
* Easy to get more resources per functions (up to 10GB of RAM!)
* Increasing RAM will also improve CPU and network!

**AWS Batch**

AWS Batch is a service that lets you run batch jobs in AWS.

We can define the time for each job when it should run.

It offers queues where we can send jobs and queue could be configured with a certain priority to run jobs.

**Features of AWS Batch**

* Fully managed batch processing at any scale
* Efficiently run 100,000s of computing batch jobs on AWS
* A “batch” job is a job with a start and an end (opposed to continuous)
* Batch will dynamically launch EC2 instances or Spot Instances
* AWS Batch provisions the right amount of compute / memory
* You submit or schedule batch jobs and AWS Batch does the rest!
* Batch jobs are defined as Docker images and run on ECS
* Helpful for cost optimizations and focusing less on the infrastructure

**AWS lambda vs Batch**

| **Batch** | **Lambda** |
| --- | --- |
| No time limit | Time limit (15 min ) |
| Any runtime as long as it’s packaged as a Docker image | Limited runtime |
| Rely on EBS / instance store for disk space | Limited temporary disk space |
| Relies on EC2 (can be managed by AWS), not serverless | Serverless |
| Not limited by programming language. | Limited by programming language |

**AWS LightSail**

Amazon Lightsail is lightweight AWS services for light weight application, for beginners.

Amazon Lightsail is the easiest way to get started with Amazon Web Services (AWS) for developers who need to build websites or web applications. It includes everything you need to launch your project quickly - instances, container services, managed databases.

**Features of LigthSail:**

* Virtual servers, storage, databases, and networking
* Low & predictable pricing
* Simpler alternative to using EC2, RDS, ELB, EBS, Route 53…
* Great for people with little cloud experience!
* Can setup notifications and monitoring of your Lightsail resources
* Use cases:
  + Simple web applications (has templates for LAMP, Nginx, MEAN, Node.js…)
  + Websites (templates for WordPress, Magento, Plesk, Joomla)
  + Dev / Test environment
* Has high availability but no auto-scaling, limited AWS integrations

**#######################################################################################**

**# AWS Cloud Formation #**

**#######################################################################################**

Cloud formation is AWS service which is used to manage (create, update, delete etc) the service/resource in automated way using the template (text file/yaml format).

In this we need to write the details in text file and upload in cloud formation service, creating updating etc action will be taken care itself.

It is IaaS in cloud technology.

**#######################################################################################**

**# AWS BeanStalk #**

**#######################################################################################**

AWS Elastic Beanstalk is a fully-managed service to deploy, run and scale web applications and services developed in Java, .NET, PHP, Node.js, Python and Ruby and using a selective of servers such as Apache, Nginx and IIS.

AWS Elastic Beanstalk uses AWS CloudFormation underneath for managing the infrastructure and resources required to run your application

This is PaaS cloud technology.

* Elastic Beanstalk is a developer centric view of deploying an application on AWS.
* Beanstalk is free but you pay for the underlying instances
* Beanstalk provides application health-monitoring & responsiveness
* Just the application code is the responsibility of the developer
* It’s managed service that means it can handle/do-
  + Instance configuration / OS is handled by Beanstalk
  + Deployment strategy is configurable but performed by Elastic Beanstalk
  + Capacity provisioning
  + Load balancing & auto-scaling

We have three tier model/structure with beanstalk, as –

* Single Instance deployment: good for dev
* LB + ASG: great for production or pre-production web applications
* ASG only: great for non-web apps in production (workers, etc..)

**Beanstalk vs CloudFormation Comparison:**

AWS Elastic Beanstalk uses AWS CloudFormation underneath for managing the infrastructure and resources required to run your application. Then, what's the difference between them?

| **Parameters** | **AWS CloudFormation** | **AWS Elastic Beanstalk** |
| --- | --- | --- |
| Purpose | Infrastructure as Code | Platform as a Service |
| Deployment | Define and manage AWS infrastructure | Simplified application deployment and scaling |
| Control | High control and flexibility over underlying resources | Simplified management of underlying resources |
| Management | Manages entire stack of resources | Abstracts infrastructure management |
| Granularity | Fine-grained control over individual | Limited configuration of underlying resources |
| Configuration | Uses JSON or YAML templates | Prescriptive configuration and environment setup |
| Use Cases | Complex architectures and multi-service | Web application deployment and scaling |

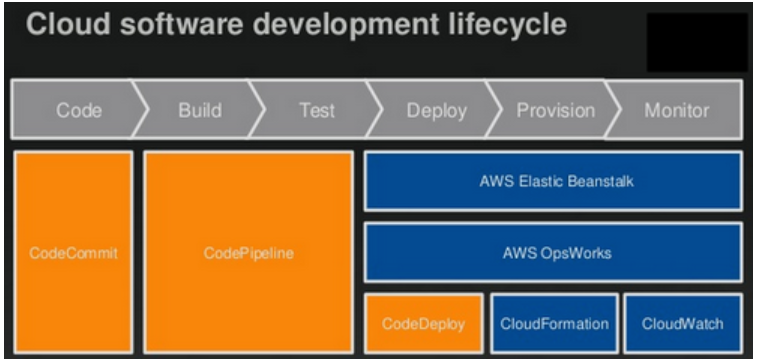
**AWS CodeDeploy**

* We want to deploy our application automatically
* Works with EC2 Instances
* Works with On-Premises Servers
* Hybrid service
* Servers / Instances must be provisioned and configured ahead of time with the CodeDeploy Agent

**CodeDeploy vs ElasticBeanStalk**

We have below major difference between these two-

* **CodeDeploy** is the service that deploys your application to the existing EC2 instance(s). It does not take into account LoadBalancing or scaling etc.
* **ElasticBeanstalk** is more of the PaaS service, that provides you all the wrapping you need to scale your application so you don't need to worry about the DevOps aspect. Like monitoring, scaling etc



**AWS CodeCommit**

Let say you want to manage your code version so in that case we need to use git as version control system.

AWS CodeCommit is AWS Cloud based **public offering for version control system**.

* CodeCommit:
  + Source-control service that hosts Git-based repositories
  + Makes it easy to collaborate with others on code
  + The code changes are automatically versioned
* Benefits:
  + Fully managed
  + Scalable & highly available
  + Private, Secured, Integrated with AWS

**AWS CodeBuild**

AWS CodeBuild is a fully managed continuous integration service that compiles source code, runs tests, and produces ready-to-deploy artifacts. It eliminates the need to maintain and manage build servers, enabling developers to focus on writing code.

* Code building service in the cloud (name is obvious)
* Compiles source code, run tests, and produces packages that are ready to be deployed (by CodeDeploy for example)
* Benefits:
  + Fully managed, serverless
  + Continuously scalable & highly available
  + Secure
  + Pay-as-you-go pricing – only pay for the build time

**AWS CodePipeline**

AWS CodePipeline is a continuous integration and continuous delivery (CI/CD) service that automates the entire release process of your applications. It orchestrates the build, test, and deployment phases, enabling a smooth and reliable release pipeline.

**Features of CodePipeline:**

* Orchestrate the different steps to have the code automatically pushed to production
  + Code => Build => Test => Provision => Deploy
* Basis for CICD (Continuous Integration & Continuous Delivery)
* Benefits:
  + Fully managed, compatible with CodeCommit, CodeBuild, CodeDeploy, Elastic Beanstalk, CloudFormation, GitHub, 3rd-party services (GitHub…) & custom plugins…
  + Fast delivery & rapid updates
* CodePipeline: orchestration layer
  + CodeCommit => CodeBuild => CodeDeploy => Elastic Beanstalk

**Difference between Code Deploy and Code Pipeline**

The main difference between AWS CodeDeploy and AWS CodePipeline is the type of application they deploy.

AWS CodeDeploy is used to deploy applications to Amazon EC2 instances, Amazon ECS containers, or any other instance type supported by AWS. AWS CodePipeline is used to deploy applications to Amazon S3 buckets.

CodeDeploy --- > EC2

CodePipeline --- > S3 bucket

**AWS CodeArtifact**

AWS CodeArtifact is a solution for managing software packages. Artifacts are a collection of files that define an application’s architecture, design, and functionality. It also includes dependencies, library resources, and packages that are required during the software-creating phase of an application. CodeArtifact functions similarly for storing the packages and tools as are necessary for application development.

* Software packages depend on each other to be built (also called code dependencies), and new ones are created
* Storing and retrieving these dependencies is called artifact management
* Traditionally you need to setup your own artifact management system
* CodeArtifact is a secure, scalable, and cost-effective artifact management for software development
* Works with common dependency management tools such as Maven, Gradle, npm, yarn, twine, pip, and NuGet.
* Developers and CodeBuild can then retrieve dependencies straight from CodeArtifact.

**CodeStar**

Unified UI to easily manage software development activities in one place (creating pipeline, build, comit etc).

“Quick way” to get started to correctly set-up CodeCommit, CodePipeline, CodeBuild, CodeDeploy, Elastic Beanstalk, EC2, etc…

Can edit the code ”in-the-cloud” using AWS Cloud9.

**Cloud9**

AWS Cloud9 is a **cloud IDE (Integrated Development Environment) for writing, running and debugging code** in browser.

“Classic” IDE (like IntelliJ, Visual Studio Code…) are downloaded on a computer before being used

A **cloud IDE can be used within a web browser**, meaning you can work on your projects from your office, home, or anywhere with internet with no setup necessary.

AWS Cloud9 also allows for code collaboration in real-time (pair programming)

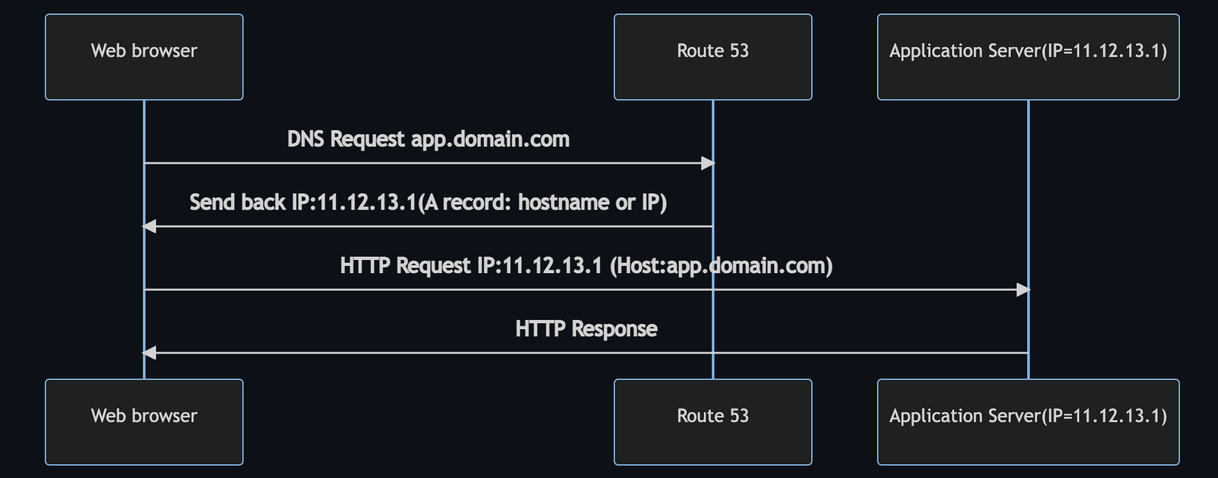
**#######################################################################################**

**# AWS GLOBAL INFRA #**

**#######################################################################################**

**Route 53**

The development of AWS Route 53 aimed to give developers and businesses an extremely reliable and cost-effective way to route end users to Internet applications by translating names like www.abc.com into numeric IP addresses like 192.0.1.2. The computers use this IP address to connect.



**CloudFront**

Amazon CloudFront is a fast content delivery network (CDN) service that securely delivers data, videos, applications etc. It is kind of caching of data on user request and then providing from cache.

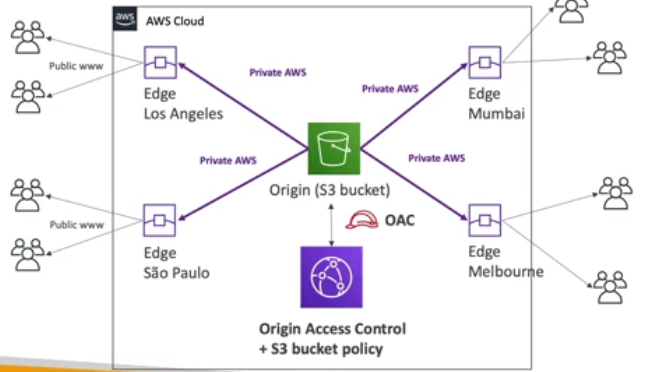
Amazon CloudFront is a web service that speeds up distribution of your static and dynamic web content, such as .html, .css, .js, and image files, to your users. CloudFront delivers your content through a worldwide network of data centres called edge locations.

When a user requests content that you are serving with CloudFront, the request is routed to the edge location that provides the lowest latency (time delay), so that content is delivered with the best possible performance.

* Content Delivery Network (CDN)
* Improves read performance, content is cached at the edge
* Improves users experience
* 216 Point of Presence globally (edge locations)
* DDoS (global server attach) protection integration with Shield, AWS Web Application Firewall

**Mechanism of working**

* If requested object is available in edge location (already cached) then it is served from edge location else if gets retrieved from origin, cached into edge and server from edge.
* Edge location will be decided as nearest location to end user.



**S3 Transfer Acceleration**

S3 Transfer Acceleration is a service that speeds up, simplifies, and secures file transfers between one client and an S3 bucket.

Transfer Acceleration uses Amazon CloudFront's globally dispersed edge sites to accelerate file transfers. When data arrives at an edge point, it is routed through an efficient network path to the Amazon S3 service. When the Transfer acceleration service is employed, the user incurs additional expenses.

I simple terms it is basically a way of transferring file from client to browser and vice versa to a temporary location(using users network) and then from there to required region (using AWS network) to make the file transfer quick.

**Example:**

Let say we want to transfer file to bucket in Australia region using S3 transfer acceleration. It will happened in below way-

**File in USA -> Edge Location(USA) -> S3 Bucket(Australia)**

**AWS Global Accelerator**

Let take a case that we have deployed our application in India and it’s global application. Now if any client who is accessing the application is as much far from India will experience that much of latency. To overcome this AWS introduced global accelerator.

AWS Global Accelerator is a service that allows you to route traffic to your applications using the AWS global network (AWS infrastructure) instead of the internet.

Global Accelerator (GA) provides you with two public static IP addresses that users can connect to in order to reach your application endpoint. GA then optimises the path from those IP addresses to your application which results in lower latency and better network performance.

* Improve global application availability and performance using the AWS global network
* Leverage the AWS internal network to optimize the route to your application (60% improvement)
* 2 Anycast IP are created for your application and traffic is sent through Edge Locations
* The Edge locations send the traffic to your application

**AWS Global Accelerator vs CloudFront**

* They both use the AWS global network and its edge locations around the world
* Both services integrate with AWS Shield for DDoS protection.
* **CloudFront – Content Delivery Network**
  + Improves performance for your cacheable content (such as images and videos)
  + Content is served at the edge
* **Global Accelerator**
  + No caching, proxying packets at the edge to applications running in one or more AWS Regions.
  + Improves performance for a wide range of applications over TCP or UDP
  + Good for HTTP use cases that require static IP addresses
  + Good for HTTP use cases that required deterministic, fast regional failover

**AWS OutPosts**

AWS outposts is hybrid cloud service. In this AWS installs the server (racks) on-premise and these on-premise servers works like AWS servers.

AWS Outposts are “server racks” that offers the same AWS infrastructure, services, APIs & tools to build your own applications on-premises just as in the cloud

* AWS will setup and manage “Outposts Racks” within your on-premises infrastructure and you can start leveraging AWS services on-premise.
* You are responsible for the Outposts Rack physical security.

**Benefits of Outposts**

* Low-latency access to on-premises systems
* Local data processing
* Data residency
* Easier migration from on-premises to the cloud
* Fully managed service
* Some services that work on Outposts are-
  + EC2
  + EBS
  + S3
  + EKS
  + ECS
  + RDS
  + EMR

**AWS WaveLength**

* WaveLength Zones are infrastructure deployments embedded within the telecommunications providers’ datacenters at the edge of the 5G networks
* Brings AWS services to the edge of the 5G networks
* Example: EC2, EBS, VPC…
* Ultra-low latency applications through 5G networks
* Traffic doesn’t leave the Communication Service Provider’s (CSP) network
* High-bandwidth and secure connection to the parent AWS Region
* No additional charges or service agreements
* Use cases: Smart Cities, ML-assisted diagnostics, Connected Vehicles, Interactive Live Video Streams, AR/VR, Real-time Gaming

**#######################################################################################**

**# Cloud Integration #**

**#######################################################################################**

Let say we deployed application then they need to communicate with one another, communication between them can be categorised in two type-

1. **Synchronous communication** (application to application)
2. **Asynchronous/event-based** Communication (application -- que -- application)

If communication is synchronous then problematic in some cases, like- if there are sudden spikes of traffic. Thus, we need to decouple synchronous communication and use some other models-

1. **SQS**: queue model

* FIFO (First In First Out queue)
* Standard Queue (message are delivered in random order)

1. **SNS**: pub/sub model
2. **Kinesis**: real-time data streaming model (out of scope for the exam)

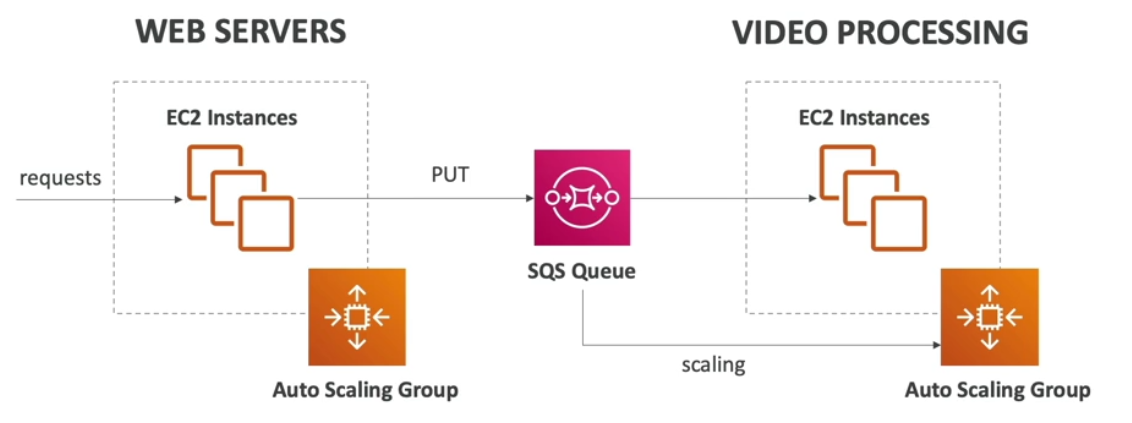
**SQS Model**

Amazon SQS is a distributed queue system that enables web service applications to quickly and reliably queue messages that one component in the application generates to be consumed by another component where a queue is a temporary repository for messages that are awaiting processing.

With the help of SQS, you can send, store, and receive messages between software components at any volume without losing messages.

Once the message is read/delivered then that message is deleted from queue.

**SQS in webserver/application example**



**Kinesis**

* Kinesis = real-time big data streaming
* Managed service to collect, process, and analyze real-time streaming data at any scale
* Too detailed for the Cloud Practitioner exam but good to know:
  + Kinesis Data Streams: low latency streaming to ingest data at scale from hundreds of thousands of sources
  + Kinesis Data Firehose: load streams into S3, Redshift, ElasticSearch, etc…
  + Kinesis Data Analytics: perform real-time analytics on streams using SQL
  + Kinesis Video Streams: monitor real-time video streams for analytics or ML

**SNS (Simple Notification Service)**

* What if you want to send one message to many receivers?
* Amazon Simple Notification Service is a notification service provided as part of Amazon Web Services since 2010. It provides a low-cost infrastructure for mass delivery of messages, predominantly to mobile users.
* The “event publishers” only sends message to one SNS topic
* As many “event subscribers” as we want to listen to the SNS topic notifications
* Each subscriber to the topic will get all the messages
* Up to 12,500,000 subscriptions per topic, 100,000 topics limit

**Amazon MQ**

* SQS, SNS are “cloud-native” services, and they’re using proprietary protocols from AWS.
* Traditional applications running from on-premise may use open protocols such as: MQTT, AMQP, STOMP, Openwire, WSS
* When migrating to the cloud, instead of re-engineering the application to use SQS and SNS, we can use Amazon MQ
* Amazon MQ = managed Apache ActiveMQ
* Amazon MQ doesn’t “scale” as much as SQS / SNS
* Amazon MQ runs on a dedicated machine (not serverless)
* Amazon MQ has both queue feature (~SQS) and topic features (~SNS)

**#######################################################################################**

**# Monitoring #**

**#######################################################################################**

**CloudWatch**

Amazon CloudWatch is a service used for monitoring and observing resources/services (all AWS services) in real-time.

CloudWatch collects monitoring and operational data in the form of logs, metrics, and events, providing its users with an aggregated view of AWS resources, applications, and services that run on AWS.

The CloudWatch can also be used to detect anomalous behaviour in the environments, set warnings and alarms, visualize logs and metrics side by side, take automated actions and troubleshoot issues.

* CloudWatch provides metrics for every services in AWS
* Metric is a variable to monitor (CPUUtilization, NetworkIn, etc..)
* Metrics have timestamps
* Can create CloudWatch dashboards of metrics

**Important Metrics**

* EC2 instances: CPU Utilization, Status Checks, Network (not RAM)
  + Default metrics every 5 minutes
  + Option for Detailed Monitoring ($$$): metrics every 1 minute
* EBS volumes: Disk Read/Writes
* S3 buckets: BucketSizeBytes, NumberOfObjects, AllRequests
* Billing:Total Estimated Charge (only in us-east-1)
* Service Limits: how much you’ve been using a service API
* Custom metrics: push your own metrics

**Terminology of AWS CloudWatch Metrics**

#### **Metrics**

* It represents a time-ordered set of data points that are published to Amazon CloudWatch
* All data point is marked with a timestamp
* Metric is a variable that is monitored and data points are the value of that variable over time
* They are uniquely defined by a name, namespace, and zero or more dimensions
* Metric math is used to query multiple cloudwatch metrics and use math expressions to create new time-series based on these metrics

#### **Dimensions**

* A dimension is a name/value pair which uniquely identifies a metric
* Dimensions are the unique identifiers for a metric, so whenever you add a unique name/value pair to one of the metrics, you are creating a new variation of that metric.

#### **Statistics**

* Statistics are metric data aggregations over specified periods of time
* The few available statistics on Cloudwatch are maximum, minimum, sum, average, and sample count.

#### **Alarm**

* It is used  to automatically initiate actions on our behalf
* It watches a single metric over a specified time period and performs one or more specified actions based on the value of the metric
* The estimated AWS charges can also be monitored using the alarm

#### **Percentiles**

* It represents the relative weightage of the data in a dataset
* It helps the user to get a better understanding of the distribution of metric data

#### **Cloudwatch dashboard**

* A user-friendly Cloudwatch console is available which is used for monitoring resources in a single view.
* There is no limit on the number of cloudwatch dashboards you can create.
* These dashboards are global and not region-specific

#### **Cloudwatch agent**

* It is required to be installed
* It collects logs and system-level metrics from EC2 instances and on-premises servers

#### **Cloudwatch Events:**

* Cloudwatch events help you to create a set of rules that match with any event(i.e stopping of EC2 instance).
* These events can be routed to one or more targets like AWS Lambda functions, Amazon SNS Topics, Amazon SQS queues, and other target types.
* Cloudwatch Events observes the operational events continuously and whenever there is any change in the state of the event, it performs the action by sending notifications, activating lambda, etc.
* An **event** indicates a change in the AWS environment. Whenever there is a change in the state of AWS resources, events are generated.
* **Rules** are used for matching events and routing to targets.
* **Target** process events. They include Amazon EC2 instances, AWS Lambda functions, etc. A target receives the events in JSON format.

#### **Cloudwatch logs:**

* Amazon Cloudwatch logs enable you to store, monitor, and access files from AWS resources like Amazon EC2 instances, Route53, etc.
* It also helps you to troubleshoot your system errors and maintain the logs in highly durable storage.
* It also creates log of information about the DNS queries that Route 53 receives

**CloudWatch Alarm**

CloudWatch Alarm is used to send notification and take action(if defined) whenever a required condition is achieved in AWS.

* Alarms are used to trigger notifications for any metric
* Alarms actions…
  + Auto Scaling: increase or decrease EC2 instances “desired” count
  + EC2 Actions: stop, terminate, reboot or recover an EC2 instance
* SNS notifications: send a notification into an SNS topic
* Various options (sampling, %, max, min, etc…)
* Can choose the period on which to evaluate an alarm
* Example: create a billing alarm on the CloudWatch Billing metric
* Alarm States: OK. INSUFFICIENT\_DATA, ALARM

**CloudWatch Logs**

CloudWatch Logs enables you to centralize the logs from all of your systems, applications, and AWS services that you use, in a single, highly scalable service

* CloudWatch Logs can collect log from:
  + Elastic Beanstalk: collection of logs from application
  + ECS: collection from containers
  + AWS Lambda: collection from function logs
  + CloudTrail based on filter
  + CloudWatch log agents: on EC2 machines or on-premises servers
  + Route53: Log DNS queries
* Enables real-time monitoring of logs
* Adjustable CloudWatch Logs retention

**CloudWatch Logs for EC2**

* By default, no logs from your EC2 instance will go to CloudWatch
* You need to run a CloudWatch agent on EC2 to push the log files you want
* Make sure IAM permissions are correct
* The CloudWatch log agent can be setup on-premises too

**Amazon EventBridge (Amazon CloudWatch Events)**

* EventBridge is the next evolution of CloudWatch Events
* Default event bus: generated by AWS services (CloudWatch Events)
* Partner event bus: receive events from SaaS service or applications (Zendesk, DataDog, Segment, Auth0…)
* Custom Event buses: for your own applications
* Schema Registry: model event schema
* EventBridge has a different name to mark the new capabilities
* The CloudWatch Events name will be replaced with EventBridge

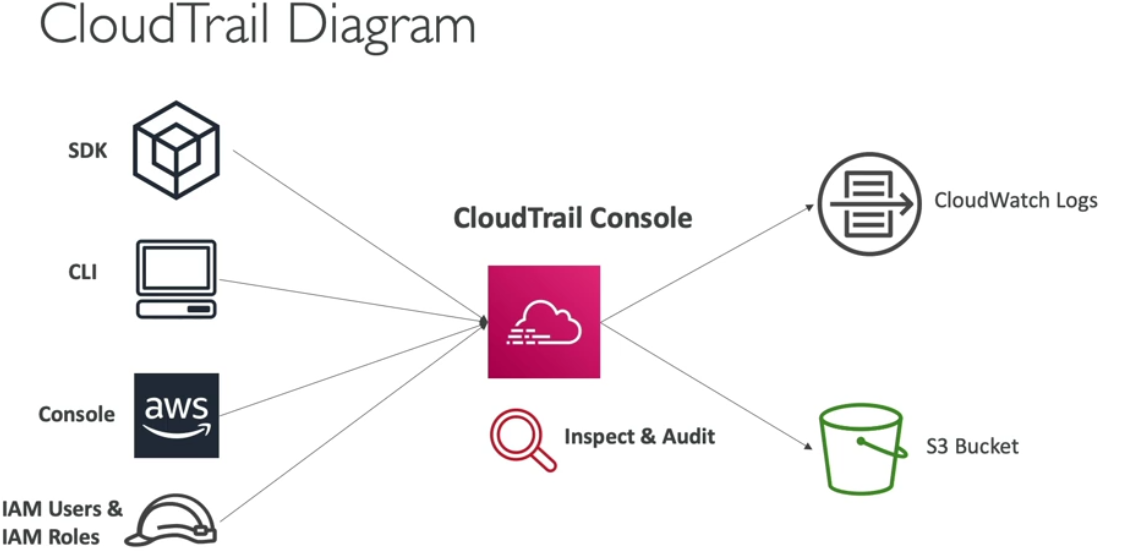
**Event vs Alarm**

Events can self-trigger based on a schedule; alarms don't do this. Alarms invoke actions only for sustained changes.

Alarms watch a single metric and respond to changes in that metric; events can respond to actions (such as a lambda being created or some other change in your AWS environment

**AWS CloudTrail**

* Provides governance, compliance and audit log for your AWS Account
* CloudTrail is enabled by default and capture all logs.
* Get an history of events / API calls made within your AWS Account by:
  + Console
  + SDK
  + CLI
  + AWS Services
* Can put logs from CloudTrail into CloudWatch Logs or S3
* A trail can be applied to All Regions (default) or a single Region.
* If a resource is deleted in AWS, investigate CloudTrail first!



**CloudTrail Events**

* **Management Events:**
  + Operations that are performed on resources in your AWS account
  + Examples:
    - Configuring security (IAM AttachRolePolicy)
    - Configuring rules for routing data (Amazon EC2 CreateSubnet)
    - Setting up logging (AWS CloudTrail CreateTrail)
  + By default, trails are configured to log management events.
  + Can separate Read Events (that don’t modify resources) from Write Events (that may modify resources)
* **Data Events:**
  + By default, data events are not logged (because high volume operations)
  + Amazon S3 object-level activity (ex: GetObject, DeleteObject, PutObject): can separate Read and Write Events
  + AWS Lambda function execution activity (the Invoke API)

**AWS X-Ray**

AWS X-Ray helps developers analyze and debug production, distributed applications, such as those built using a microservices architecture.

With X-Ray, you can understand how your application and its underlying services are performing to identify and troubleshoot the root cause of performance issues and errors.

**AWS X-Ray advantages**

* Troubleshooting performance (bottlenecks)
* Understand dependencies in a microservice architecture
* Pinpoint service issues
* Review request behaviour
* Find errors and exceptions
* Are we meeting time SLA?
* Where I am throttled?
* Identify users that are impacted

**AWS Code Guru**

* An ML-powered service for automated code reviews and application performance recommendations
* Provides two functionalities
* **CodeGuru Reviewer**: automated code reviews for static code analysis (development)
* **CodeGuru Profiler**: visibility/recommendations about application performance during runtime (production)

**CodeGuru Reviewer**

* Identify critical issues, security vulnerabilities, and hard-to-find bugs
* Example: common coding best practices, resource leaks, security detection, input validation
* Uses Machine Learning and automated reasoning
* Hard-learned lessons across millions of code reviews on 1000s of open-source and Amazon repositories
* Supports Java and Python
* Integrates with GitHub, Bitbucket, and AWS CodeCommit

**CodeGuru Profiler**

Amazon CodeGuru Profiler helps developers find an application's most expensive lines of code(issue in code) by helping them understand the runtime behaviour of their applications, identify and remove code inefficiencies, improve performance, and significantly decrease compute costs.

* Helps understand the runtime behavior of your application
* Example: identify if your application is consuming excessive CPU capacity on a logging routine
* Features:
  + Identify and remove code inefficiencies
  + Improve application performance (e.g., reduce CPU utilization)
  + Decrease compute costs
  + Provides heap summary (identify which objects using up memory)
  + Anomaly Detection
* Support applications running on AWS or on- premise
* Minimal overhead on application

**AWS Services Health Dashboard**

AWS Health Shows all regions, all services health, service events, planned changes, and account notifications to help you manage and take actions.

To view it , sign in to account and we would be able to see it.

* Shows all regions, all services health
* Shows historical information for each day
* Has an RSS feed you can subscribe to
* <https://status.aws.amazon.com/>

**AWS Personal Health Dashboard**

Personal Health Dashboard gives you a personalized view into the performance and availability of the AWS services underlying your AWS resources.

* AWS Personal Health Dashboard provides alerts and remediation guidance when AWS is experiencing events that may impact you.
* While the Service Health Dashboard displays the general status of AWS services, Personal Health Dashboard gives you a personalized view into the performance and availability of the AWS services underlying your AWS resources.
* The dashboard displays relevant and timely information to help you manage events in progress and provides proactive notification to help you plan for scheduled activities.
* Global service https://phd.aws.amazon.com/
* Shows how AWS outages directly impact you & your AWS resources
* Alert, remediation, proactive, scheduled activities

**#######################################################################################**

**# VPC and Networking #**

**#######################################################################################**

**What is VPC**

Amazon Virtual Private Cloud (Amazon VPC) provides a logically isolated area of the AWS cloud where you can launch AWS resources in a virtual network that you define.

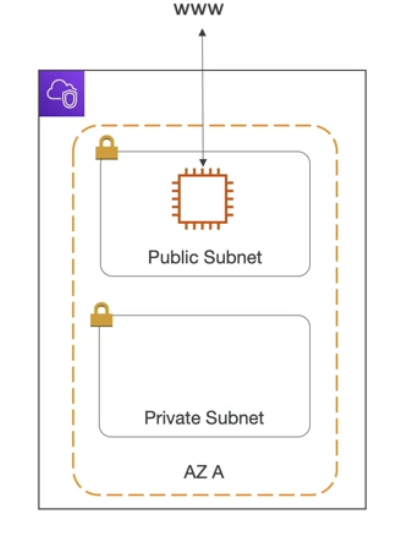
We have complete control over your virtual networking environment, including a selection of your IP address range, the creation of subnets, and configuration of route tables and network gateways.

**IP Address**



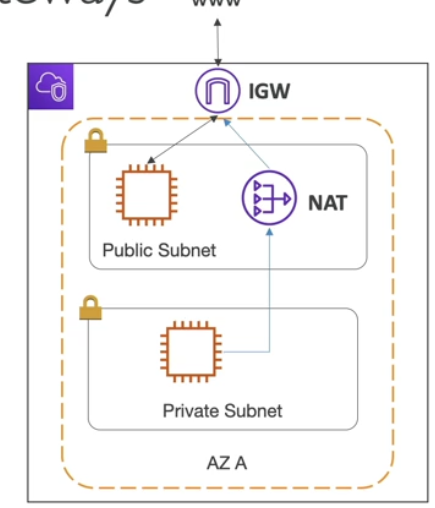
**VPC & Subnets Primer**

* VPC -Virtual Private Cloud: private network to deploy your resources to specific region(regional resource)
* **Subnets** allow you to partition your network inside your VPC (Availability Zone resource)
  + A **public subnet** is a subnet that is accessible from the internet
  + A **private subnet** is a subnet that is not accessible from the internet
* To define access to the internet and between subnets, we use Route Tables.



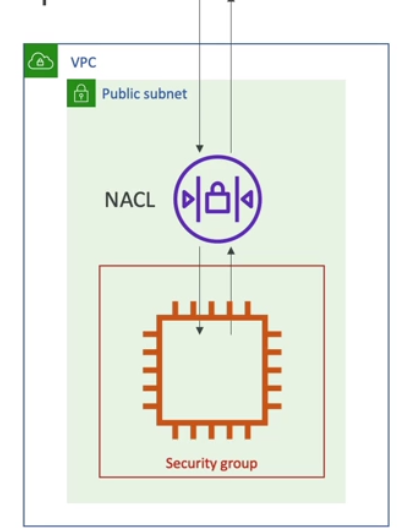
**Internet Gateway (IGW) & NAT Gateways**

* Internet Gateways helps our VPC instances connect with the internet
* Public Subnets have a route to the internet gateway.
* NAT Gateways (AWS-managed) & NAT Instances (self-managed) allow your instances in your Private Subnets to access the internet while remaining private



**Network ACL & Security Groups**

* NACL (Network ACL)
  + A firewall which controls traffic from and to subnet
  + Can have ALLOW and DENY rules
  + Are attached at the Subnet level
  + Rules only include IP addresses
* Security Groups
  + A firewall that controls traffic to and from an ENI / an EC2 Instance
  + Can have only ALLOW rules
  + Rules include IP addresses and other security groups



**Network ACLs vs Security Groups**

| **Security Group** | **Network ACL** |
| --- | --- |
| Operates at the instance level | Operates at the subnet level |
| Supports allow rules only | Supports allow rules and deny rules |
| Is stateful: Return traffic is automatically allowed, regardless of any rules | Is stateless: Return traffic must be explicitly allowed by rules |
| We evaluate all rules before deciding whether to allow traffic | We process rules in number order when deciding whether to allow traffic |
| Applies to an instance only if someone specifies the security group when launching the instance, or associates the security group with the instance later on | Automatically applies to all instances in the subnets it's associated with (therefore, you don't have to rely on users to specify the security group) |

**#######################################################################################**

**# Machine Learning #**

**#######################################################################################**

For AWS CCP certification we need to know about below AWS machine learning services.

* Amazon Rekognition
* Amazon Transcribe
* Amazon Polly
* Amazon Translate
* Amazon Lex & Connect
* Amazon Lex: (same technology that powers Alexa)
* Amazon Connect
* Amazon Comprehend
* Amazon SageMaker
* Amazon Forecast
* Amazon Kendra
* Amazon Personalize
* Amazon Textract

**####################**

**# Amazon Recognition #**

**###################**

* Find objects, people, text, scenes in images and videos using ML
* Facial analysis and facial search to do user verification, people counting
* Create a database of “familiar faces” or compare against celebrities
* **Use cases:**
  + Labeling
    - Content Moderation
    - Text Detection
    - Face Detection and Analysis (gender, age range, emotions…)
    - Face Search and Verification
    - Celebrity Recognition

**###################**

**# Amazon Transcribe #**

**###################**

* Automatically convert **speech to text**.
* Uses a deep learning process called **automatic speech recognition (ASR) to convert speech to text** quickly and accurately.
* **Use cases-**
  + transcribe customer service calls
  + automate closed captioning and subtitling
  + generate metadata for media assets to create a fully searchable archive

**##############**

**# Amazon Polly #**

**##############**

* Turn text into lifelike speech using deep learning
* Allowing you to create applications that talk

**##################**

**# Amazon Translate #**

**##################**

* Natural and accurate **language translation**.
* Amazon Translate allows you to **localize content translation**- such as websites and applications - for international users, and to easily translate large volumes of text efficiently.

**######################**

**# Amazon Lex & Connect #**

**######################**

* Automatic Speech Recognition (ASR) to **convert speech to text**
* **Natural Language Understanding to recognize the intent of text, callers**
* Helps build chatbot, call centre bots

**#################**

**# Amazon Connect #**

**#################**

* Receive calls, create contact flows, cloud-based virtual contact centre
* Can integrate with other **CRM systems or AWS**
* No upfront payments, 80% cheaper than traditional contact centre solutions

**#####################**

**# Amazon Comprehend #**

**#####################**

* For Natural Language Processing – NLP
* Fully managed and serverless service
* Uses machine learning to find insights and relationships in text
  + Language of the text
  + Extracts key phrases, places, people, brands, or events
  + Understands how positive or negative the text is
  + Analyses text using tokenization and parts of speech
  + Automatically organizes a collection of text files by topic
* **Sample use cases-**
  + analyze customer interactions (emails) to find what leads to a positive or negative experience
  + Create and groups articles by topics that Comprehend will uncover

**###################**

**# Amazon SageMaker #**

**###################**

* Fully managed service for developers / data scientists to build ML models
* Typically, difficult to do all the processes in one place + provision servers
* Machine learning process (simplified): predicting your exam score

**#################**

**# Amazon Forecast #**

**#################**

* Fully managed service that uses ML **to deliver highly accurate forecasts**
* Example: predict the future sales of a raincoat
* 50% more accurate than looking at the data itself
* Reduce forecasting time from months to hours
* Use cases: Product Demand Planning, Financial Planning, Resource Planning,etc.

**################**

**# Amazon Kendra #**

**################**

* Fully managed document search service powered by Machine Learning
* **Extract answers from within a document** (text, pdf, HTML, PowerPoint, MS Word, FAQs…)
* Natural language search capabilities
* Learn from user interactions/feedback to promote preferred results (Incremental Learning)
* Ability to manually fine-tune search results (importance of data, freshness, custom,etc..)

**###################**

**# Amazon Personalize #**

**###################**

* Fully managed **ML-service to build apps with real-time personalized recommendations**
* Example: personalized product recommendations/re-ranking, customized direct marketing
* Example: User bought gardening tools, provide recommendations on the next one to buy
* Same technology used by Amazon.com
* Integrates into existing websites, applications, SMS, email marketing systems, …
* Implement in days, not months (you don’t need to build, train, and deploy ML solutions)
* Use cases: retail stores, media and entertainment

**#################**

**# Amazon Textract #**

**################**

* Automatically extracts text, handwriting, and data from any scanned documents using AI and ML
* Extract data from forms and tables
* Read and process any type of document (PDFs, images, …)
* Use cases:
  + Financial Services (e.g., invoices, financial reports)
  + Healthcare (e.g., medical records, insurance claims)
  + Public Sector (e.g., tax forms, ID documents, passports)

**#######################################################################################**

**# boto3 #**

**#######################################################################################**

Boto3 is python library for working with AWS services.

Boto is a software development kit (SDK) designed to improve the use of the Python programming language in Amazon Web Services.

**Client in boto3:**

Clients provide a low-level interface to AWS whose methods map close to 1:1 (each AWS services can be used/accessed using boto3) with service APIs. All service operations are supported by clients.

For each AWS resource we can created client in below way –

**client = boto3.client( service\_name )** ---- Creating boto3 client for any service.

* Typically **yields primitive, non-marshalled data** (e.g. DynamoDB attributes are dicts representing primitive DynamoDB values)

**Resource in boto3:**

Resources represent an object-oriented interface to Amazon Web Services (AWS). They provide a higher-level abstraction than the raw, low-level calls made by service clients. Not all AWS services can be exposed/used by resource.

We can create boto3 resource in below way for supportable AWS resource-

**resource = boto3.resource ( service\_name )** ----- Creating boyo3 resource

* Typically **yields marshalled data, not primitive AWS data** (e.g. DynamoDB attributes are native Python values representing primitive DynamoDB values)

**Paginators**

Some AWS operations return results that are incomplete and require subsequent requests in order to attain the entire result set. In such case we need to use paginator.

Paginator can be **created by using boto3 client or resource as** -

**paginator= client.get\_paginator(operation\_name)**

**paginator= resource.get\_paginator(operation\_name)**

* **operation\_name**

This is the same name as the method name of the client/resource.

If method of client/resource is 'foo()' then it will be accessed like –

* + **resource.get\_paginator(foo)**
  + **clinet.get\_paginator(foo)**

**################################ AWS S3 client #########################################**

For boto3 clinet of S3 object we have below methods-



**Commonly used methods are:**

1. **S3.Client.get\_object(\*\*kwargs)**

Retrieves objects from Amazon S3. To use GET, you must have READ access to the objects.

Return type is python dictionary.

**kwarg** – these are keyword argument **of request format**.

**Request Syntax/Format**

Request/syntax accepts below parameters**-**

response = client.get\_object(

Bucket='string',

IfMatch='string',

IfModifiedSince=datetime(2015, 1, 1),

IfNoneMatch='string',

IfUnmodifiedSince=datetime(2015, 1, 1),

Key='string',

Range='string',

ResponseCacheControl='string',

ResponseContentDisposition='string',

ResponseContentEncoding='string',

ResponseContentLanguage='string',

ResponseContentType='string',

ResponseExpires=datetime(2015, 1, 1),

VersionId='string',

SSECustomerAlgorithm='string',

SSECustomerKey='string',

RequestPayer='requester',

PartNumber=123,

ExpectedBucketOwner='string',

ChecksumMode='ENABLED'

)

**Description of request parameter:**

* **Bucket**=string -- Name of bucket in which file is present
* **Key** = string -- key/full path of file (excluding bucket name) in bucket

**Response Syntax**

Response is **always a dict type**, having below parameters-

{

'Body': StreamingBody(), # this is response in botocore

'DeleteMarker': True|False,

'AcceptRanges': 'string',

'Expiration': 'string',

'Restore': 'string',

'LastModified': datetime(2015, 1, 1),

'ContentLength': 123,

'ETag': 'string',

'ChecksumCRC32': 'string',

'ChecksumCRC32C': 'string',

'ChecksumSHA1': 'string',

'ChecksumSHA256': 'string',

'MissingMeta': 123,

'VersionId': 'string',

'CacheControl': 'string',

'ContentDisposition': 'string',

'ContentEncoding': 'string',

'ContentLanguage': 'string',

'ContentRange': 'string',

'ContentType': 'string',

'Expires': datetime(2015, 1, 1),

'WebsiteRedirectLocation': 'string',

'ServerSideEncryption': 'AES256'|'aws:kms'|'aws:kms:dsse',

'Metadata': {

'string': 'string'

},

'SSECustomerAlgorithm': 'string',

'SSECustomerKeyMD5': 'string',

'SSEKMSKeyId': 'string',

'BucketKeyEnabled': True|False,

'StorageClass': 'STANDARD'|'REDUCED\_REDUNDANCY'|'STANDARD\_IA'|'ONEZONE\_IA'|'INTELLIGENT\_TIERING'|'GLACIER'|'DEEP\_ARCHIVE'|'OUTPOSTS'|'GLACIER\_IR'|'SNOW',

'RequestCharged': 'requester',

'ReplicationStatus': 'COMPLETE'|'PENDING'|'FAILED'|'REPLICA'|'COMPLETED',

'PartsCount': 123,

'TagCount': 123,

'ObjectLockMode': 'GOVERNANCE'|'COMPLIANCE',

'ObjectLockRetainUntilDate': datetime(2015, 1, 1),

'ObjectLockLegalHoldStatus': 'ON'|'OFF'

}

**########################### botocore.response #############################**

class botocore.response.StreamingBody(raw\_stream, content\_length)

Wrapper class for an http response body.

It have some methods to work with StreamingBody data.

Streaming Body class methods are:-

* **read(amt=None)**

Read at most amt bytes from the stream.If the amt argument is omitted, read all data.

* **close()**

Close the underlying http response stream

* **readable()**

Return whether object was opened for reading.If False, read() will raise OSError.

**########################### s3 client methods #############################**

1. **client.get\_paginator(operation\_name)**

Creates a paginator for operation.

Parameters are:

**operation\_name** -- this will be boto3 client method name for which we want to create pagination.

1. client.list\_objects(\*\*kwargs)

Returns some or all (up to 1,000) of the objects in a bucket. You can use the request parameters as selection criteria to return a subset of the objects in a bucket. It’s return type is dictionary.

**Request (\*\*kwargs) parameters are:**

response = client.list\_objects(

Bucket='string',

Delimiter='string',

EncodingType='url',

Marker='string',

MaxKeys=123,

Prefix='string',

RequestPayer='requester',

ExpectedBucketOwner='string',

OptionalObjectAttributes=[

'RestoreStatus',

]

)

**Return type/format:**

This method returns the result in dict format having below parameter-

{

'IsTruncated': True|False,

'Marker': 'string',

'NextMarker': 'string',

'Contents': [

{

'Key': 'string',

'LastModified': datetime(2015, 1, 1),

'ETag': 'string',

'ChecksumAlgorithm': [

'CRC32'|'CRC32C'|'SHA1'|'SHA256',

],

'Size': 123,

'StorageClass': 'STANDARD'|'REDUCED\_REDUNDANCY'|'GLACIER'|'STANDARD\_IA'|'ONEZONE\_IA'|'INTELLIGENT\_TIERING'|'DEEP\_ARCHIVE'|'OUTPOSTS'|'GLACIER\_IR'|'SNOW',

'Owner': {

'DisplayName': 'string',

'ID': 'string'

},

'RestoreStatus': {

'IsRestoreInProgress': True|False,

'RestoreExpiryDate': datetime(2015, 1, 1)

}

},

],

'Name': 'string',

'Prefix': 'string',

'Delimiter': 'string',

'MaxKeys': 123,

'CommonPrefixes': [

{

'Prefix': 'string'

},

],

'EncodingType': 'url',

'RequestCharged': 'requester'

}

**Contents (list)**

Metadata about each object returned. it will be list of dictionary-

An object consists of data and its descriptive metadata.

* + Key (string) –

The name that you assign to an object. You use the object key to retrieve the object.

* + LastModified (datetime) –

Creation date of the object.

1. **clinet.list\_objects\_v2(\*\*kwargs)**

this is exactly same as list\_objects(\*\*kwargs) expect list\_objects\_v2(\*\*kwargs) takes some addition parameter in input-

|  |  |
| --- | --- |
| list\_objects\_v2 | list\_objects |
| response = client.list\_objects\_v2(  Bucket='string',  Delimiter='string',  EncodingType='url',  MaxKeys=123,  Prefix='string',  # Replace marker to list continuous page  ContinuationToken='string',  # set to True to fetch key owner info. Default is False.  FetchOwner=True|False,  # This is similar to the Marker in list\_object()  StartAfter='string'  ) | response = client.list\_objects(  Bucket='string',  Delimiter='string',  EncodingType='url',  #Marker to list continuous page  Marker='string',  MaxKeys=123,  Prefix='string'  ) |