**AWS Fundamentals**

**Datacenter**

* A Data Center is a facility that houses computing facilities like servers, routers, switches and firewalls, as well as supporting components like backup equipment, fire suppression facilities and air conditioning.
* A Data center may be complex (dedicated building) or simple (an area or room that houses only a few servers).
* We may have one or more Data centres, depending on how large the customer base is.

**Components Of Data center**

* A data center infrastructure may include:
  + **Racks** : Data center hardware is typically mounted into racks that maximize the use of space in the facility.
  + **Network Connectivity** : Data centers often have multiple fiber connections to the internet provided by multiple carriers.
  + **Power** : Each machine in a data center may be dual-power provided with the data center having multiple grid connections.
  + **Energy Production Systems** : A system of backup power such as a generator with fuel storage. It is common for data centers to have a solar panel system on the roof or nearby.
  + **Environment Control** : Systems for cooling hardware and providing heating, ventilation, air conditioning, humidification and dehumidification for the facility.
  + **Physical Security** : Typically monitored with cameras and may have on-site security guards.

**Cons Of Physical Data Center**

* For many organizations, the job of running a Data center is an Expensive and Complicated burden
  + Cost of Building
  + Cost of Administration
  + Cost of Power Generators
  + Cost of Cooling
  + Cost of Cabling
  + Physically securing the place

**Virtualization**

Virtualization is the creation of a virtual, rather than actual version of something, such as an operating system, a server, a storage device or network resources.

You probably know a little about Virtualization if you have ever divided your hard drive into different partitions. A partition is the logical division of a hard disk drive to create, in effect, two separate hard drives.

When people talk about Virtualization, they’re usually referring to **Server Virtualization**, which means partitioning one physical server into several virtual servers, or machines.

**Pre-Virtualization World** - **Why do we need Server Virtualization ??**

In a world before virtualization, servers would traditionally run **one** **application** on **one** **server** with **one** **operating system**. In the old system, the number of servers would continue to mount since every new application required its own server and operating system.

As a result, expensive hardware resources were being purchased, but not used. Each server would only use about **12%** of its resources. **Almost 88% of server resources were completely unutilized**.

Virtualization is possible because of a software layer called a **HYPERVISOR**.

Virtualization allows multiple operating systems to run concurrently on a single host computer.

**Hypervisor**

What is a hypervisor and how does it differ from bare metal?

A **Hypervisor/Virtual Machine Monitor** (**VMM**) is an low-level program/software that can create and run Virtual Machines (VM's) within a bare-metal server.

Let’s have a look at the representation below to better understand the difference between the two.

The first image above represents a traditional bare-metal server. The operating system (CentOS, Debian, Ubuntu, Windows Server, etc.) is installed directly on the server, and applications are running natively in the operating system, **A single OS owns all hardware resources**.

The first step of the virtualization process is installing the hypervisor onto a server.

In the second image above, a bare-metal server installed with a hypervisor provides the user with a management suite to create virtual machines on the server.

After the hypervisor is up and running, multiple “software containers” known as virtual machines (VMs) are built on top of the hypervisor. All of the VMs are isolated from each other.

Once the VMs have been built on top of the hypervisor:

* Applications and operating systems can be added to each VM.
* Every app and operating system placed on the server lives in a separate VM, which means if an app or operating system goes down on one VM, none of the other apps or operating systems on the other VMs are affected.

**So how do the VMs interact with the hardware resources?**

This is where the hypervisor comes in: the hypervisor is able to distribute the underlying resources based on what each VM needs. Resources (like memory, storage, processors, and networking) are pooled together so that every VM can get exactly what it needs for its ideal performance.

You can think of the hypervisor as the traffic cop that controls processor,memory, networking and storage management.

If one VM needs more memory than other apps, the hypervisor can allocate more memory for that VM. If another needs more storage, the hypervisor can allocate more storage. And so on.

Virtualization lets you run more applications on fewer physical servers. Rather than one application running on one server with one operating system, multiple VMs run multiple applications and operating systems on one physical server.

Just in case this is still muddled or confusing, here’s how I would explain virtualization.

Virtualization is like a school bus. Before the school bus was invented, every parent used their own car to drive their kid to school, using extra gas and resources, putting all of the kids into one vehicle wasn’t an option.

One day, the school bus was introduced, exposing the inefficiency of every parent driving their kid to school separately. By using the school bus, parents could use less gas and fewer vehicles, all while transporting more kids.

**Benefits of Virtualization**

* Power Savings
* Cooling Savings
* Hardware Savings
* Network savings, no need of extra network cables
* Space Savings, lower number of physical servers
* Resource Sharing, can create multiple machines on single server, which saves Money by reducing cost.
* Deploy multiple Applications & OS's
* Full utilization of Hardware resources
* Isolation, VM's are isolated from each other as if they are physically separated
* VM's can be migrated between different hosts

With Virtualization solution you can reduce IT costs while increasing the efficiency, utilization and flexibility of their existing computer hardware i.e, simplified management of Data center.

Experts predict that shipping hypervisors on bare metal will impact how organizations purchase servers in the future. Instead of selecting an OS, they will simply have to order a server with an embedded hypervisor and run whatever OS they want.

**Cloud Computing**

* Cloud computing is the on-demand delivery of compute power, database storage, applications, and other IT resources through a cloud services platform via the internet with pay-as-you-go pricing.
* With cloud computing, you don’t need to make large upfront investments in hardware and spend a lot of time on the heavy lifting of managing that hardware. Instead, you can provision exactly the right type and size of computing resources you need to operate your IT department. You can access as many resources as you need, almost instantly, and only pay for what you use.
* Testing of your new ideas for the applications becomes much easier.

**Cloud Computing Offerings**

* **IAAS** - Infrastructure As A Service
* **PAAS** - Platform As A Service
* **SAAS** - Software As A Service

**IAAS**

**PAAS**

**SAAS**

**AWS Account Setup**

**Setting up AWS Account**

* The AWS Free Tier enables you to gain free, hands-on experience with the AWS platform, products, and services.
* These free tier offers are only available to new AWS customers, and are available for 12 months following your AWS sign-up date.
* [FREE TIER FEATURES](https://aws.amazon.com/free/) and you can register an account in same page
* [Visit](https://aws.amazon.com/free/)
* Click the button to “**Create an AWS Account**”
* On the next page, provide your Email Address, Password, and AWS Account Name (you can change this name in your account settings after sign up).
* Click “**Continue**” to proceed
* Next, you’ll provide your contact information. If you’re registering as an individual, select “**Personal**” and if you’re using any business, select “Company”
* Complete the remaining fields with your information. Then click “Create Account and Continue” to proceed.
* Next, you’ll be asked to provide a credit card for your AWS Account.
* Once you’ve completed this information, click the “Secure Submit” button to proceed.
* Next, you’ll be asked to complete a brief phone verification step. Here, you are asked to provide a phone number where you can be reached, and to click the “Call Me Now” button to receive an automated phone call.
* Once you receive the call, you’ll input the number shown on your screen using your dial-pad

AWS Global Infrastructure

Amazon Cloud Computing resources are available across the world. In easy words if we see this then **Amazon Data Centres are available in different geographical locations**.

Organization can register their presence and launch their product using these Data Centre in any Location

* AWS, in terms of its global infrastructure is broken up at the highest level between
* **AWS Regions**
* **AWS Availability Zones**
* **AWS Edge Locations**

**Regions**

Amazon Cloud Computing Resources or Data Centres are available in different Geographical Locations.

* One location is called One AWS Region.
* Each AWS Region is separate from other AWS Region.
* Availability of Services are different for different AWS Regions.
* Regions are designed to service AWS customers (or your users) that are located closest to a region.
* When viewing a region in the console you will only view resources in one region at a time.
* Availability of regions allow the architects to design applications to conform to specific laws and regulations
* Some AWS services work "globally" while some work within a specific region only
* When we provision an EC2 instance or S3 Bucket, then you would select the region and that is where these are provisioned or stored in that region.
* One AWS region is a combination of multiple Availability Zones (AZs).

**Availability Zone**

As per AWS infrastructure, each geographical area is known as AWS Region which is a **logical Data Centre**.

Each Region has multiple Physical Data Centres and these Physical Data Centres are known as AVAILABILITY ZONE or AZs.

* The Availability zone is where the actual data centres are located.
* So within a Region there can be multiple Availability zones which are physically separated but are connected through low latency and high speed internet connections.
* One Availability zone (AZ) is one Physical Data Centre.
* Each AZ has independent Power Supply, Networking, Cooling System, Physical Security.
* Each AZ connected via redundant, ultra-low-latency networks with other AZ in the same AWS Region.
* Properly designed applications will utilize multiple availability zones for High Availability and Fault Tolerance.
* That means if any organization deploys their database in one region then the data is distributed in multiple AZs. In the issue of power outages, lightning strikes, tornadoes, earthquakes, and more at one AZ, Data is safe and accessible from other AZ.
* AZ’s are physically separated by a meaningful distance, many kilometers, from any other AZ, although all are within 100+ km (60+ miles) of each other.
* [VISIT AWS Global Infra](https://aws.amazon.com/about-aws/global-infrastructure/)

**Region & AZ’s**

**Edge Locations**

* An Edge location can be assumed to be a collection of physical servers within a data center to allow for **content distribution to reduce latency for end users**.
* The higher the number of edge locations the better the content is distributed all over the world / region.
* An example would be CloudFront which is a CDN:
  + Cached items such as a PDF file can be cached on the edge location which reduces the amount of "space/time/latency" required for a request from the other part of the world.

VPC - Virtual Private Cloud

* Amazon Virtual Private Cloud (Amazon **VPC**)
  + **Let’s customers(IBM) provision a logically isolated section of the AWS Cloud** where you can **launch AWS resources(Servers) in a VIRTUAL NETWORK** that you define.
* This virtual network closely resembles a traditional network that you'd operate in your own datacenter.
* It is similar to having your own data center inside AWS. The resources are completely isolated from other VPC on AWS
* **VPC is the Backbone of infrastructure of any systems that we decide to build on AWS**.
* Amazon **VPC is the networking layer** for Amazon EC2.

**VPC Features & Benefits**

* You have complete control over your virtual networking environment
  + **selection of your own IP address range**
  + **creation of subnets**
  + **configuration of route tables**
  + **configuration of network gateways.**
* A variety of connectivity options exist for your Amazon VPC. **You can connect your VPC to the Internet, to your data center, or other VPCs**, based on the AWS resources that you want to expose publicly and those that you want to keep private.
* **Layered security**
* **Instance level - Security Groups (firewall on instance level)**
* **Subnet level - Network ACLs (firewall on the subnet level)**

**VPC Connectivity Options**

* Connect directly to the Internet (**public subnets**) – You can launch instances such as web servers into a publicly accessible subnet where they can send and receive traffic from the Internet.
* Connect to the Internet using (**private subnets**) – Private subnets can be used for instances such as database servers that you do not want to be directly addressable from the Internet.
* Connect privately to other VPCs - **Peer VPCs** together to share resources across multiple virtual networks owned by your or other AWS accounts.
* **NOTE** : First thing you need to understand is, VPC within a region spans across Multiple Availability zones because of that it spans across multiple data centres.

**Default VPC**

* Your AWS resources are automatically provisioned in a ready-to-use default VPC that was created for you.
* The default VPC is meant to allow the user easy access to VPC without having to configure it from scratch.
* Default VPC has CIDR, Security Group, NACL and Route Table settings
* Has Internet Gateway created and attached by default
* Each instance launched in the default VPC (by default) has a private and public IP address (defined on the subnet settings).

**VPC Network Routing Basics**

* Now to understand Routing we need to first look into VPC Components
* **Internet Gateway**
* **Route Tables**
* **Subnets**
* **NACL's**
* **Security Groups**
* "**To enable access to or from the internet to an instance in a VPC which resides in a subnet, you must attach an Internet gateway to your VPC, ensure that your subnet route table points to the Internet gateway and ensure that instance has a public IP address or Elastic IP address, and ensure that your network access control and security group rules allow the relevant traffic to your instance**" -- **AWS**

**Internet Gateway**

* Internet Gateway rules and details you need to know:
  + Only 1 IGW can be attached to a VPC at a time.
  + An IGW must be attached to a VPC if the resources inside the VPC need to connect to resources via the open internet.

In the above diagram, Subnet 1 in the VPC is associated with a custom route table that points all internet-bound(0.0.0.0/0) traffic to an Internet gateway. The instance has an Elastic IP address, which enables communication with the internet.

**Router**

* It's the central VPC routing function
* It connects different Availability Zones and Subnets together
* It connects VPC to IGW
* Each Subnet will have a Route Table and router uses it to forward the traffic within the VPC i.e SUBNET ASSOCIATION
* Route tables will have entries to destinations

**Route Tables**

* A route table contains a set of rules called routes, that are used to determine where network traffic is directed.
* **Your VPC automatically comes with a main route table**
* You can create additional custom route tables for your VPC.
* Each subnet must be associated with a route table, which controls the routing for the subnet.
* If you don't explicitly associate a subnet with a particular route table, the **subnet is implicitly associated with the main route table**.
* You cannot delete the main route table
* A route table's rules are comprised of two main components:
* Destination: The CIDR block range of the target (where the data is routed to).
* Target: A name identifier of where the data is being routed to.
* By default, all subnets traffic is allowed to each other, available subnet within your VPC which is called the **local route**.
* You cannot modify or delete the local route.
* Unlike an IGW, you can have multiple route tables in a VPC
* NOTE: The "default" VPC already has a "main" route table.

**Main / Default Route Table**

* When you create a VPC, it automatically has a main route table. On the Route Tables page you can view the main route table for a VPC by looking for Yes in the Main column.
* The main route table controls the routing for all subnets that are not explicitly associated with any other route table.

**Custom Route Table**

* Your VPC can have route tables other than the default table.
* Custom route tables ensure that you explicitly control how each subnet routes outbound traffic.
* Route Table characteristics, will decide the Subnet characteristics
  + Public Route Table - Internet Based - igw
  + Private Route Table - Intranet Based - local

**Subnets**

* When you create a VPC, it spans across all of the Availability Zones in the region.
* After creating a VPC, you can add one or more subnets in each Availability zone.
* Each subnet must reside entirely within one availability zone and cannot span zones
* Subnets MUST be associated with a route table.
* A **PUBLIC subnet** HAS a route to the internet.
* It is associated with a route table that has an IGW attached.
* A **PRIVATE subnet** does NOT have a route to the Internet.
  + It is associated with a route table that does NOT have an IGW attached.

**NACL**

* A Network access control list (NACL) acts as a firewall for controlling traffic on one or more subnets. NACL's operate at the subnet level.
* They support allow and deny rules for traffic traveling into or out of a subnet.
* They **process rules in number order** when deciding whether to allow traffic.
* Rules are evaluated in order, starting with the lowest rule number -
  + for Example: if traffic is denied at a lower rule number and allowed at a higher rule number, the allow rule will be ignored and the traffic will be denied.
* NOTE - Your "default" VPC already has a NACL and it is associated with the default subnets.

**Default NACL**

* The default network ACL is configured to allow all traffic to flow in and out of the subnets to which it is associated.
* Each network ACL also includes a rule whose rule number is an asterisk. This rule ensures that if a packet doesn't match any of the other numbered rules, it's denied. You can't modify or remove this rule.
* The following is an example default network ACL for a VPC

**NACL Rules**

* Rules are evaluated from lowest to highest based on "rule #".
* The first rule found that applies to the traffic type is immediately applied, regardless of any rules that come after it
* An NACL allows or denies traffic from entering a subnet. Once inside the subnet, other AWS resources (i.e EC2 instances) may have an additional layer of security (security groups).

**Security Groups**

* A security group acts as a virtual firewall for your instance to control inbound and outbound traffic.
* Security groups are very similar to NACLs, Security groups act at the instance level, whereas NACLs work at subnet level.
* You can specify only allow rules, but not deny rules.
* Your VPC automatically comes with a default security group.
* Each EC2 instance that you launch in your VPC is automatically associated with the default security group if you don't specify a different security group when you launch the instance. You can't delete default Security Group.
* You can't delete the default Security Group.
* Changes to Security Groups take effect immediately
* Default SG, will have inbound rules allowing instances assigned the same SG to talk to one another and also all outbound traffic is allowed
* Custom (non-default) SG will have no inbound rules, basically all inbound traffic is denied by default and all outbound traffic is allowed by default

**VPC & Subnetting**

* When you create a VPC, you must specify a CIDR block for the VPC.
* The allowed block size is between a /16 netmask (65,536 IP addresses) and /28 netmask (16 IP addresses).
* AWS recommends to create a CIDR with 10.0.0.0/16 for future purpose
* The CIDR blocks of the subnets cannot overlap.
* <https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Subnets.html>

The **first four IP addresses** and the **last IP address** in each subnet CIDR block are not available for you to use, and cannot be assigned to an instance. For example, in a subnet with CIDR block 10.0.0.0/24, the following five IP addresses are reserved:

* 10.0.0.**0**: Network address.
* 10.0.0.**1**: Reserved by AWS for the VPC router.
* 10.0.0.**2**: Reserved by AWS for Amazon DNS server.
* 10.0.0.**3**: Reserved by AWS for future use.
* 10.0.0.**255**: Network broadcast address.

**CIDR Chart**

**Subnet Calculator**

[**https://www.site24x7.com/tools/ipv4-subnetcalculator.html**](https://www.site24x7.com/tools/ipv4-subnetcalculator.html)

**VPC Requirement**

**Network Requirement Given by CST**

> CST is going to setup the environment i.e servers on AWS, their **clients are from london**

> They are gonna host around 8000 Servers

> These 8k servers are grouped into two subnets each with 4k servers

> First Subnet is for web servers, around 4k

> Second Subnet is for database servers, around 4k

> In the future, they might add more number of subnets i.e application servers subnets, load balancer subnet

> The web servers needs to be communicated from internet

> The database servers needs to be communicated only from Web Servers, but not from internet

> Coming to security of Web Servers, they should have only

> SSH traffic enabled for administration from internet

> HTTP traffic for clients from internet

> Coming to security of Database Servers, they should have only SSH access from web servers

> SSH traffic enabled from web servers only

**Solution**

> CST is going to setup the environment i.e servers on AWS, their clients are from london

> Select Region : London

> They are gonna host around 8000 Servers

> VPC Capacity : 10.0.0.0/19

> Always select MAX Capacity i.e /16

> VPC Capacity : 10.0.0.0/16

> These 8k servers are grouped into two subnets each with 4k servers

> First Subnet is for web servers, around 4k

> Web Subnet : 10.0.0.0/20

> Second Subnet is for database servers, around 4k

> DB Subnet : 10.0.16.0/20

> In the future, they might add more number of subnets i.e application servers subnets, load balancer subnet

> Future Subnet : 10.0.32.0/X

> The web servers needs to be communicated from internet

> Web Server will be launched in cst-web subnet

> Create Internet Gateway

> Attach IGW to cst vpc

> Route Table pointing to Internet Gateway [ 0.0.0.0/0 ]

> Route Table Association to cst-web subnet

> Enable Public IP address setting on subnet level { cst-web subnet }

> The database servers needs to be communicated only from Web Servers, but not from internet

> create a new route table

> local route is present, no need of internet

> Route Table Association to cst-db subnet

> Coming to security of Web Servers, they should have only

> SSH traffic enabled for administration from internet

> Add SSH i.e port 22 - Source : 0.0.0.0/0

> HTTP traffic for clients from internet

> Add HTTP i.e port 80 - Source : 0.0.0.0/0

> Coming to security of Database Servers, they should have only SSH access from web servers

> SSH traffic enabled from web servers only

> Add SSH i.e port 22 - Source : 10.0.0.0/20

**EC2 - Elastic Cloud Compute**

* Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) cloud.
* Using Amazon EC2 eliminates your need to invest in hardware upfront, so you can develop and deploy applications faster.
* You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.
* I want you to picture EC2 like a computer, and the components that make it up like OS, CPU, HDD, NW, Firewall, RAM etc.

**EC2 - Features**

* Virtual computing environments, known as instances
* Preconfigured templates for your instances, known as Amazon Machine Images (AMIs), that package the bits you need for your server (including the OS and additional software)
* Various configurations of CPU, Memory, Storage and Networking capacity for your instances, known as Instance Types
* Secure login information for your instances using Key Pairs (AWS stores the public key and you store the private key in a secure place)
* Storage volumes for temporary data that's deleted when you stop or terminate your instance, known as Instance Store Volumes
* The instance store is ideal for temporary storage, because the data stored in instance store volumes is not persistent through instance stops, terminations or hardware failures.
* Persistent storage volumes for your data using Amazon Elastic Block Store, known as Amazon EBS volumes
* A firewall that enables you to specify the protocols, ports, and source IP ranges that can reach your instances using security groups
* Static IPv4 addresses for dynamic cloud computing, known as Elastic IP addresses
* Metadata tags, that you can create and assign to your Amazon EC2 resources
* Virtual networks you can create that are logically isolated from the rest of the AWS cloud known as Virtual Private Clouds (VPCs)

**EC2 - Configuration**

* EC2 instances are designed to mimic traditional on-premise servers, but with the ability to be commissioned and decommissioned on-demand for easy scalability and elasticity.
* EC2 instances are primarily comprised of the follow components:
* Amazon Machine Image (AMI): The operating system (and other softwares).
* Instance Type: The hardware (computer power, ram, network bandwidth, etc).
* Network interface: (public, private, or elastic IP addresses).
* Storage: The instances "hard drive" (including two options).
* Elastic Block Store (EBS) - which is "network persistent storage".
* Instance Store - which is "ephemeral storage".

**EC2 Facts**

* A Security group must be assigned to an instance during the creation process.
* Each instance must be placed into an existing VPC, availability zone and subnet.
* Automated (bootstrapping) custom launch commands can be passed into the instance during launch via "user - data" scripts.
* "Tags" can be used to help name and organize provisioned instances.
* Key-pairs are used to manage login authentication.

**EC2 Instance Types**

* When you launch an instance, the instance type that you specify determines the hardware of the host computer used for your instance.
* Each instance type offers different compute, memory, and storage capabilities and are grouped in instance families based on these capabilities.
* Instances types describe the "hardware" components that an EC2 instance will run on:
* Compute power (processor/vCPU)
* Memory (ram)
* Storage Option (hard drive)
* Network Performance (bandwidth)
* As an architect, it's important to use the proper instance type to handle your application's workload.
* There is a collection of pre configured instance types that are grouped into families and types that you can choose from:
* **General Purpose Instances** - General purpose instances provide a balance of compute, memory, and networking resources, and can be used for a variety of workloads.
  + Websites and web applications, Small and medium databases, Development, build, test, and staging environments
  + Check this link<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/general-purpose-instances.html>
* **Compute Optimized Instances** - Compute optimized instances are ideal for compute-bound applications that benefit from high-performance processors. They are well suited for the following applications:
  + High-performance web servers, High-performance computing (HPC), Media transcoding, Scientific modeling
  + Check this link<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/compute-optimized-instances.html>
* **Memory Optimized Instances** - Memory optimized instances are designed to deliver fast performance for workloads that process large data sets in memory.
  + High-performance, relational (MySQL) and NoSQL (MongoDB, Cassandra) databases.
  + In-memory databases using optimized data storage formats and analytics for business intelligence (for example, SAP HANA).
  + Check this link<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/memory-optimized-instances.html>
* **Storage Optimized Instances** - Storage optimized instances are designed for workloads that require high, sequential read and write access to very large data sets on local storage. They are optimized to deliver tens of thousands of low-latency, random I/O operations per second (IOPS) to applications.
  + Massive parallel processing (MPP) data warehouse
  + MapReduce and Hadoop distributed computing
  + Applications that require high-throughput access to large quantities of data
  + Log or data processing applications
  + Check this link<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/storage-optimized-instances.html>
* **Accelerated Computing Instances** - If you require high processing capability, you'll benefit from using accelerated computing instances, which provide access to hardware-based compute accelerators such as Graphics Processing Units (GPUs)
  + Check this link<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/accelerated-computing-instances.html>

EC2 - Purchase Options

Amazon EC2 provides the following purchasing options to enable you to optimize your costs based on your needs:

* **On-Demand Instances** – Pay, by the second, for the instances that you launch.
* **Savings Plans** – Reduce your Amazon EC2 costs by making a commitment to a consistent amount of usage, in USD per hour, for a term of 1 or 3 years.
* **Reserved Instances** – Reduce your Amazon EC2 costs by making a commitment to a consistent instance configuration, including instance type and Region, for a term of 1 or 3 years.
* **Spot Instances** – Request unused EC2 instances, which can reduce your Amazon EC2 costs significantly.
* **Dedicated Hosts** – Pay for a physical host that is fully dedicated to running your instances, and bring your existing per-socket, per-core, or per-VM software licenses to reduce costs.
* **Dedicated Instances** – Pay, by the hour, for instances that run on single-tenant hardware.
* **Capacity Reservations** – Reserve capacity for your EC2 instances in a specific Availability Zone for any duration.

**Instance Lifecycle**

The lifecycle of an instance starts when it is launched and ends when it is terminated. The purchasing option that you choose affects the lifecycle of the instance. For example, an On-Demand Instance runs when you launch it and ends when you terminate it. A Spot Instance runs as long as capacity is available and your maximum price is higher than the Spot price.

**On-Demand Instances**

* With On-Demand Instances, you pay for compute capacity by the second.
* You have full control over its lifecycle—you decide when to launch, stop, hibernate, start, reboot, or terminate it.
* There is no long-term commitment required when you purchase On-Demand Instances.
* You pay only for the seconds that your On-Demand Instances are in the running state.
* The price per second for a running On-Demand Instance is fixed, and is listed on the [Amazon EC2 Pricing, On-Demand Pricing page](http://aws.amazon.com/ec2/pricing/on-demand/)
* We recommend that you use On-Demand Instances for applications with short-term, irregular workloads that cannot be interrupted.

**Reserved Instances**

* Reserved Instances provide you with significant savings on your Amazon EC2 costs compared to On-Demand Instance pricing.
* With Reserved Instances, you pay for the entire term regardless of actual use.
* You can purchase a Reserved Instance for a one-year or three-year commitment, with the three-year commitment offering a bigger discount.
* **One-year**: A year is defined as 31536000 seconds (365 days).
* **Three-year**: Three years is defined as 94608000 seconds (1095 days).
* Reserved Instances do not renew automatically; when they expire, you can continue using the EC2 instance without interruption, but you are charged On-Demand rates.

* The following payment options are available for Reserved Instances:
* **All Upfront**: Full payment is made at the start of the term, with no other costs or additional hourly charges incurred for the remainder of the term, regardless of hours used.
* **Partial Upfront**: A portion of the cost must be paid upfront and the remaining hours in the term are billed at a discounted hourly rate, regardless of whether the Reserved Instance is being used.
* **No Upfront**: You are billed a discounted hourly rate for every hour within the term, regardless of whether the Reserved Instance is being used. No upfront payment is required.
* Generally speaking, you can save more money making a higher upfront payment for Reserved Instances.

* If your computing needs change, you may be able to modify or exchange your Reserved Instance, depending on the offering class.
* **Standard**: These provide the most significant discount, but can only be modified.
* **Convertible**: These provide a lower discount than Standard Reserved Instances, but can be modified
* <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/reserved-instances-types.html>

**Spot Instances**

* A Spot Instance is an unused EC2 instance that is available for less than the On-Demand price.
* Amazon EC2 Spot Instances are spare EC2 compute capacity in the AWS Cloud that are available to you at savings of up to 90% off compared to On-Demand prices.
* Because Spot Instances enable you to request unused EC2 instances at steep discounts, you can lower your Amazon EC2 costs significantly.
* The hourly price for a Spot Instance is called a Spot price. The Spot price of each instance type in each Availability Zone is set by Amazon EC2, and is adjusted gradually based on the long-term supply of and demand for Spot Instances.
* Your Spot Instance runs whenever capacity is available and the maximum price per hour for your request exceeds the Spot price.
* Spot Instances are a cost-effective choice if you can be flexible about when your applications run and if your applications can be interrupted. For example, Spot Instances are well-suited for data analysis, batch jobs, background processing, and optional tasks.

**Dedicated Hosts**

* An Amazon EC2 Dedicated Host is a physical server with EC2 instance capacity fully dedicated to your use.
* With a Dedicated Host, you have visibility and control over how instances are placed on the server.
* Dedicated Hosts allow you to use your existing per-socket, per-core, or per-VM software licenses, including Windows Server, Microsoft SQL Server etc

**Dedicated Instances**

* Dedicated Instances are Amazon EC2 instances that run in a virtual private cloud (VPC) on hardware that's dedicated to a single customer.

**Dedicated Hosts vs Dedicated Instances**

Dedicated Hosts and Dedicated Instances can both be used to launch Amazon EC2 instances onto physical servers that are dedicated for your use.

There are no performance, security, or physical differences However, there are some differences between the two. The following table highlights them

|  | **Dedicated Host** | **Dedicated Instance** |
| --- | --- | --- |
| **Billing** | **Per-host billing** | **Per-instance billing** |
| **Visibility of sockets, cores, and host ID** | **Provides visibility of the number of sockets and physical cores** | **No visibility** |
| **Host and instance affinity** | **Allows you to consistently deploy your instances to the same physical server over time** | **Not supported** |
| **Targeted instance placement** | **Provides additional visibility and control over how instances are placed on a physical server** | **Not supported** |
| **Automatic instance recovery** | **Supported. For more information, see** [**Host recovery**](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/dedicated-hosts-recovery.html)**.** | **Supported** |
| **Bring Your Own License (BYOL)** | **Supported** | **Not supported** |

**Installing Gitbash**

Once [Git Bash Windows installer](https://git-scm.com/download/win) is downloaded, run the executable file and follow the steps.

**SSH**

* What is **SSH** & **SSH Client** ?
  + SSH (Secure Shell) is a network protocol that gives users, particularly system administrators, a secure way to access a remote computer. An SSH client is a program that allows establishing a secure and authenticated SSH connection to SSH servers.
    - Ex : Putty, GitBash, Terminal etc
* SSH Syntax [ Remote Connection ]

> **ssh -i <key> username@public-ip-address**

> **ssh -i <key> username@public-dns**

**EC2 Server Setup**

Let’s launch an instance i.e server inside AWS, using EC2 service.

**Instance Setup**

> Launch an Instance with **Amazon Linux 2** on AWS

> Login to AWS > Services > Compute Section > EC2 > Launch Instance > Select **Amazon Linux 2** AMI > Choose t2.micro > Config Instance Details {keep all default values} > Add Storage {default} > Add Tags {default} > Configure Security Group > Review & Launch > Launch Instance > In Keypair Section > Create new keypair (cst) > Launch Instance

**Steps to Launch Amazon Linux 2 in AWS**

> Click Services and then EC2

> Click Launch Instance

> Click AWS Marketplace

> Search for Centos

> Select Top Result - Centos7

> Click Continue

> Select your machine type and click Next Configure Instance details.

In our case we will select the **t2.micro** instance as it is free tier eligible.

> Leave the defaults in **Configure Instance Details, Add Storage and Add Tags**

> Click Next: Configure Security Group

> Click review and Launch.

> Review your settings and then click Launch.

> In the drop down menu select create a new key pair, give the key pair a name and Download the Key Pair, then click launch Instances.

> Now scroll down and click view instances

Now in order to communicate with the servers, we need an **ssh client like Putty or Gitbash**,

SSH Syntax

* chmod 400 first.pem
* ssh -i <file.pem> <username>@public-ip-address
* ssh -i first.pem centos@public-ip-address
* Use uname command to verify, if you get Linux, it’s successful

Download Putty

* <https://the.earth.li/~sgtatham/putty/latest/w64/putty.exe>

Download Puttygen

* <https://the.earth.li/~sgtatham/putty/latest/w64/puttygen.exe>

PuTTY uses .ppk files instead of .pem files. If you haven't already generated a .ppk file, do so now. For more information, see [To prepare to connect to a Linux instance from Windows using PuTTY](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html#prepare-for-putty).

> Open puttygen and click Load

> Navigate to where you downloaded your key, click all files, click on your key and click open.

> Now click Save Private key, when prompted click yes you want to save without a passphrase.

> Now open putty and enter your public IP into the host name or IP address field, then expand SSH on the left had side.

> Click auth and then browse, navigate to where you saved your key and select it.

> Now click open

> Click Yes

> Enter **ec2-user** as the username and click enter.

> You will now be logged in

**LAB - Web App**

**Web Server**

A web server is a program which serves web pages to users in response to their requests, which are forwarded by their computers' HTTP clients(Browsers).

**Purpose of Web server**

A web server’s main purpose is to store web site files and broadcast them over the internet for you site visitors to see. In essence, a web server is simply a powerful computer that stores and transmits data via the internet.

Web servers are the gateway between the average individual and the world wide web.

All **computers that host websites must have web server programs**.

**Apache Web Server**

An **open source web server** used mostly for **Unix and Linux platforms**.

It is **fast, secure and reliable**.

* An Open Source Web Server
* Apache is developed and maintained by an open community of developers under the [Apache Software Foundation](https://en.wikipedia.org/wiki/Apache_Software_Foundation).
* The Apache HTTP Server is cross-platform as of 1 June 2017, 92% of Apache Server copies run on Linux distributions.
* Apache played a key role in the initial growth of the World Wide Web.
* The Apache HTTP Server has been the most popular Web Server on the public Internet since April 1996.
* In 2009, it became the first web server software to serve more than 100 million websites.

**Parameters for Apache (httpd)**

* Package - httpd
* Port - 80
* Protocol - http
* Server Root - /etc/httpd
* Main config file - /etc/httpd/conf/httpd.conf
* Configuration Test - httpd -t
* Document root - /var/www/html

**LAB - Setup**

> Launch Linux instance with AMI :: Amazon Linux 2 in web subnet

> Install an web application using the following procedure

> Installing Apache Web Server

-> sudo rpm -qa | grep httpd

-> sudo yum -y install httpd

-> sudo rpm -qa | grep httpd

> Starting the Apache Web Server

-> sudo systemctl status httpd

-> sudo systemctl start httpd

-> sudo systemctl enable httpd

-> sudo systemctl status httpd

> Browse the Public IP of Instance on BROWSER and you should be seeing the sample test app

-> sudo ls /var/www/html

> Generally the code in the organizations will be stored in Source Code Management Tools and for us it is Github

-> sudo rpm -qa | grep git

-> sudo yum -y install git

-> sudo rpm -qa | grep git

-> Git is client and we need client to access github

-> sudo git clone https://github.com/Akiranred/ecomm.git /var/www/html

-> sudo ls /var/www/html

> Browse the Public IP of Instance on BROWSER and you should be seeing the Shopping app

**EC2 - IP Address**

* **Private IP Address**
  + All EC2 instances are automatically created with a PRIVATE IP address.
  + The private IP address is used for internal (inside the VPC) communication between instances.
* **Public IP Address**
* When creating an EC2 instance, you have the option to enable (or auto-assign) a public IP address.
* A public IP address is required if you want the EC2 instance to have direct communication with resources across the open internet, i.e if you want to directly SSH into the instance or have it directly serve web traffic.
* **Auto-assigning** is based on the setting for the **selected subnet** that you are provisioning the instance in.
* **Elastic IP Address (EIP)**
  + An Elastic IP address is a public IPv4 address, which is reachable from the internet.
  + You can mask the failure of an instance or software by rapidly remapping the address to another instance in your account (i.e detaching the EIP from one instance and attaching it to another).
  + Attaching an EIP to an instance will replace it's default public IP address for as long as it is attached.
  + A disassociated Elastic IP address remains allocated to your account until you explicitly release it.
  + To ensure efficient use of Elastic IP addresses, AWS imposes a small hourly charge if an Elastic IP address is not associated with a running instance
  + An Elastic IP address is for use in a specific region only.

**LAB - EC2 - EIP**

> Create an Instance with Amazon Linux 2 As Operating System in Public Subnet and tag it as Web Server

> Attach an Elastic IP to Web Server

> **Steps to attach Elastic IP**

Services -> EC2 -> Left pane -> NETWORK & SECURITY -> Click Elastic IP's -> Allocate New Address -> Amazon Pool -> Select/Checkmark EIP -> Actions > Associate Address > Select Web Server Instance > Associate

> Now stop and start the server back, and see if the Elastic IP got changed ? As you can see it's the same, which is useful in DNS

Deploy the following App -<https://github.com/Akiranred/food.git> /var/www/html

EC2 Storage

EC2 Storage

* EC2 instances support two types for block level storage
  + Elastic Block Store - EBS (Persistent - Network attached drives)
  + Instance Store ( Ephemeral/temporary store)
* EC2 instances can be launched by choosing between AMIs backed by EC2 instance stores and AMIs backed by EBS. However, **AWS recommends use of EBS backed AMIs**, because they launch faster and use persistent storage

EBS - Elastic Block Store

* Amazon Elastic Block Store ( Amazon EBS ) provides block level storage volumes for use with EC2 instances.
* EBS volumes are **highly available and reliable storage volumes** that can be attached to any running instance that is in the same Availability Zone. EBS volumes that are attached to an EC2 instance are exposed as storage volumes that persist independently from the life of the instance. With Amazon EBS, you pay only for what you use.
* Amazon EBS is recommended when data must be quickly accessible and requires long-term persistence. EBS volumes are particularly well-suited for use as the primary storage for file systems & databases.

Root vs Additional Volumes

* Every EC2 instance must have a root volume
* By default, EBS "root" volumes are set to be deleted when the instance is terminated. However, you can choose to have EBS volumes persist after termination.
* You can add additional EBS volumes to instance if needed
* Any additional volume can be attached or detached from instance at any time and is not deleted by default when instance is terminated
* An EBS volume can attach to a single EC2 Instance only at a time
* Both EBS Volume and EC2 instance MUST be in the same AZ
* EBS volumes are persistent, meaning that they can live beyond the life of the EC2 instance they are attached to.
* EBS volumes are network attached storage, meaning they can be attached/detached to or from various EC2 instances.
* However, they can only be attached to ONE EC2 instance at a time.
* EBS volumes have the benefit of being backed up into a snapshot - which can later be restored into a new EBS volume.

EBS - Performance

* EBS volumes measure input/output operations in IOPS:
* IOPS are input/output operations per second
* AWS measures IOPS in 256KB chunks (or smaller)
* For example, A 512KB operation would count as 2 IOPS
* The type of EBS volume you specify greatly influences the I/O performance (IOPS) your device
* It is important as architects to understand if your application requires more (or less) I/O when selecting an EBS volume type

EBS - Types

Amazon EBS provides the following volume types, which differ in performance characteristics and price, so that you can tailor your storage performance and cost to the needs of your applications. The volumes types fall into these categories:

* [Solid state drives (SSD)](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html#solid-state-drives) — Optimized for transactional workloads involving frequent read/write operations with small I/O size, where the dominant performance attribute is IOPS.
* [Hard disk drives (HDD)](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html#hard-disk-drives) — Optimized for large streaming workloads where the dominant performance attribute is throughput.
* [Previous generation](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html#ebs-previous-generation-volumes) — Hard disk drives that can be used for workloads with small datasets where data is accessed infrequently and performance is not of primary importance. We recommend that you consider a current generation volume type instead.
* Check the following links for more information
* <https://aws.amazon.com/ebs/volume-types/>
* <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html>

Instance Store

* Instance store volumes are virtual devices whose underlying hardware is physically attached to the host computer that is running the instance.
* Instance store volumes are considered ephemeral data i.e temporary storage, meaning the data on the volumes only exists for the duration of the life of the instance.
* The EC2 instance attached with the instance store can't be stopped, they can only be rebooted or terminated, and termination will erase the data.

| **Instance Store** | **Elastic Block Store** |
| --- | --- |
| Local to Instance | Network Attached Storage |
| Non-persistent storage | Persistent Storage |
| No Snapshot Support | Point in time Snapshot support |

Snapshots

**EBS - Snapshot**

* Frequent snapshots of your data increases **data durability** - so highly recommended.
* When a snapshot is being taken against the EBS volume, it can degrade performance, so snapshots should occur during non-peak load hours
* To take a consistent Snapshot of your **non-root** (**not the boot**) **EBS** Volume:
  + Pause file writes till you snapshot is complete
  + If you can't pause file writes, you need to unmount (detach) the volume from instance, take the snapshot, then re mount the volume to ensure a consistent and complete snapshot
* To create a snapshot for a **root** (**boot**) **EBS** Volume, you should stop the instance first then take the snapshot.
  + Be careful if you have instance store volumes on EC2 Instance , their data will be lost once you stop the instance.

LAB - Snapshots

> Launch Linux instance with AMI :: Amazon Linux 2 in public subnet

> Install an web application using the following procedure

> Installing Apache Web Server

-> sudo rpm -qi httpd

-> sudo yum -y install httpd

-> sudo rpm -qi httpd

> Starting the Apache Web Server

-> sudo systemctl status httpd

-> sudo systemctl start httpd

-> sudo systemctl enable httpd

-> sudo systemctl status httpd

> Browse the Public IP of Instance on BROWSER and you should be seeing the sample test a

-> sudo ls /var/www/html

> Generally the code in the organizations will be stored in Source Code Management Tools and for us it is Github

-> sudo rpm -qi git

-> sudo yum -y install git

-> sudo rpm -qi git

-> Git is client and we need client to access github

-> sudo git clone https://github.com/Akiranred/ecomm.git /var/www/html

-> sudo ls /var/www/html

> Browse the Public IP of Instance on BROWSER and you should be seeing the Shopping app

-> Taking snapshots against volumes

-> Select the volumes that is attached to instance

-> Actions -> Take Snapshot

{ Now a snapshot will be available in Elastic Block Store section of EC2 dashboard }

-> Now your goal is to launch another instance with the same ecomm website from Snapshot, in another availability zone, let's say the first instance was launched in 1A now the new instance we are launching should be in 1B with the ecomm website up and running.

**Limitations Of EBS**

* You cannot get the data across multiple availability zones
* You cannot connect multiple instances to same EBS volume

AMI

* An Amazon Machine Image (AMI) provides the information required to launch an instance. You must specify an AMI when you launch an instance.
* You can launch multiple instances from a single AMI when you need multiple instances with the same configuration.
* You can use different AMIs to launch instances when you need instances with different configurations.
* The following diagram summarizes the AMI lifecycle. After you create and register an AMI, you can use it to launch new instances.
* You can copy an AMI to different AWS Regions for Disaster Recovery. When you no longer require an AMI, you can deregister it.
* You can launch an instance from an existing AMI, customize the instance (for example, install software on the instance), and then save this updated configuration as a custom AMI.
* Instances launched from this new custom AMI include the customizations that you made when you created the AMI.
* The root storage device of the instance determines the process you follow to create an AMI.
* The AWS Marketplace is an online store where you can buy software that runs on AWS, including AMIs that you can use to launch your EC2 instance.
* The AWS Marketplace AMIs are organized into categories, such as Developer Tools, to enable you to find products to suit your requirements.

**Amazon Linux AMI**

Amazon Linux 2 and the Amazon Linux AMI are supported and maintained Linux images provided by AWS. The following are some of the features of Amazon Linux 2 and Amazon Linux AMI:

* A stable, secure, and high-performance execution environment for applications running on Amazon EC2.
* Provided at no additional charge to Amazon EC2 users.
* Repository access to multiple versions of MySQL, PostgreSQL, Python, Ruby, Tomcat, and many more common packages.
* Updated on a regular basis to include the latest components, and these updates are also made available in the yum repositories for installation on running instances.
* Includes packages that enable easy integration with AWS services, such as the AWS CLI, Amazon EC2 API and AMI tools, the Boto library for Python, and the Elastic Load Balancing tools.

LAB - AMI's

> AMI - OS | Apps | Additional S/W's

> Ecomm-AMI - Amazon Linux 2 | Ecomm | Git & HTTPD

LAB - Setup

> Launch Linux instance with AMI :: Amazon Linux 2 in public subnet

> Install an web application using the following procedure

> Installing Apache Web Server

-> sudo rpm -qi httpd

-> sudo yum -y install httpd

-> sudo rpm -qi httpd

> Starting the Apache Web Server

-> sudo systemctl status httpd

-> sudo systemctl start httpd

-> sudo systemctl enable httpd

-> sudo systemctl status httpd

> Browse the Public IP of Instance on BROWSER and you should be seeing the sample test app

-> sudo ls /var/www/html

> Generally the code in the organizations will be stored in Source Code Management Tools and for us it is Github

-> sudo rpm -qi git

-> sudo yum -y install git

-> sudo rpm -qi git

-> Git is client and we need client to access github

-> sudo git clone https://github.com/Akiranred/ecomm.git /var/www/html

-> sudo ls /var/www/html

> Browse the Public IP of Instance on BROWSER and you should be seeing the Shopping app

**AMI Process**

> To create an AMI do the following

-> EC2 Dashboard -> Left side we got AMI's -> Click AMI's

-> select Instance -> Right click -> Image -> Create Image { keep all default }

-> EC2 Dashboard -> Left side we got AMI's -> Click AMI's

-> Checkout Snapshots

-> EC2 Dashboard -> Left side we got AMI's -> Click AMI's -> Select AMI -> Launch Instance

Instance User Data

* **Bootstrapping**
  + Refers to a self-starting process i.e run set of commands without external input.
  + With EC2, we can bootstrap the instance (during the creation process) with custom commands (such as installing software packages, running updates and configuring other various settings).

**User Data**

* When you launch an instance in Amazon EC2, you have the option of passing user data to the instance that can be used to perform common automated configuration tasks and even run scripts after the instance starts.
* If you are familiar with shell scripting, this is the easiest and most complete way to send instructions to an instance at launch. Adding these tasks at boot time adds to the amount of time it takes to boot the instance.
* You should allow a few minutes of extra time for the tasks to complete before you test that the user script has finished successfully.
* User data shell scripts must start with the #! characters and the path to the interpreter you want to read the script (commonly /bin/bash).
* Is data supplied by the user at instance launch in the form of a script to be executed during instance boot
* User data is limited to **16KB**
* User data is not protected by encryption, do not include passwords or sensitive data in your user data scripts
* You can change user data by stopping the instance first, then actions → Instance settings → View/Change user data
* A step/section during the EC2 instance creation process where you can include your own custom commands via a script (i.e a bash script)
* Here is an example of a bash script that will automate the process of updating the yum package installer, install Apache Web Server and start the Apache service.

***#!/bin/bash***

***yum update -y***

***yum install httpd -y***

***service httpd start***

**Elastic File System**

* EFS is a storage option for EC2 that allows for a scalable storage option.
* EFS storage capacity is elastic.
* The storage capacity will increase and decrease as you add or remove files.
* EFS is fully managed (no maintenance required).
* Supports the Network File system version 4.0 and 4.1 (NFSv4) protocols when mounting.
* Best performance when using an EC2 AMI with Linux kernel 4.0 or newer & **EFS is not used as boot volume**

Benefits Of EFS

* The EFS file system can be accessed by one (or more) EC2 instances at the same time
* Shared file access across all your EC2 instances.
* Applications that span multiple EC2 instances can access the same data.
* EFS file systems can be mounted to **on-premise servers** ( when connected to your VPC via **AWS Direct Connect**).
* This allows you to migrate data from on-premise servers to EFS and/or use it as a **backup solution**.
* EFS can **scale to petabytes** in size, while maintaining low-latency and high levels of throughput.
* You pay only for the amount of storage you are using.

**EFS**

* Amazon Elastic File System (Amazon EFS) provides a simple, scalable, fully managed elastic NFS file system.
* EFS can be mounted on EC2 instances or on-premise instances through an AWS Direct connection
* It is built to scale on demand to petabytes without disrupting applications, growing and shrinking automatically as you add and remove files, eliminating the need to provision and manage.
* Amazon EFS can scale up to Petabyte scale, and is designed to provide massively parallel shared access to thousands of Amazon EC2 instances, enabling your applications to achieve high levels throughput.
* It's limited to Linux Instances Only
* You need **NFS client**, to mount the file system on EC2 Instances
* EFS supports Network File System version 4.0 & 4.1
* Multiple EC2 instances in the same region, same VPC and in different AZ's, can access amazon EFS file system at the same time.
* This provides the common data source for workloads and applications running more than one instance
* EFS uses port **2049** for NFS file system not for instances

**EFS Mount Targets**

* To access EFS file system in VPC, you can create one or more mount targets in the VPC
* You can create only one mount target in each availability zone
* If there are multiple subnets in an AZ, you can create a mount target in one of the subnets, then all the instances in that AZ will share the mount target
* Mount targets are also highly available service
* AWS recommends that you create mount targets in all the AZ's, so that you can easily mount the file system on EC2 instances that you might launch in any zone in future, as there are no charges for mount targets

**EFS Use-Cases**

* Amazon EFS enables customers to persist data from their containers and serverless functions, elastic, highly-available and high-performance, cloud-native shared file systems.
* Amazon EFS allows data to be persisted separately from compute, and enables applications to have cross-AZ availability and durability.
* Amazon EFS provides the ease of use, scale, performance, and consistency needed for machine learning and big data analytics workloads. Amazon SageMaker integrates with EFS for training jobs, allowing data scientists to iterate quickly.
* Amazon EFS provides a durable, high throughput file system for content management systems and web serving applications.

**EFS Storage Classes**

Amazon EFS offers two storage classes: the Standard storage class, and the Infrequent Access storage class (EFS IA).

**Standard** : used to store frequently accessed data i.e daily accessed.

**Infrequent Access** : It's a lower cost storage class that's designed for infrequently accessed files(not accessed everyday), IA provides cost-optimization for files not accessed every day.

By simply enabling EFS Lifecycle Management on your file system, files not accessed according to the lifecycle policy you choose will be automatically and transparently moved into EFS IA.

LAB - EFS

> Shared access to multiple instances

> Launch Linux instance with AMI :: Amazon Linux 2 in public subnet tag it as PRIMARY

> Create EFS from Storage Section i.e in Services -> Storage -> EFS

> EFS needs to be launched in Subnets choose the Public Subnets in two diff AZ's

> EFS works on port 2049(NFS), create a security group to allow NFS

> In order to connect to EFS storage we need NFS utilities

> Install NFS utilities on PRIMARY instance, by following the instructions given in EFS page once EFS is in available state

-> Now launch another Linux instance with AMI :: Amazon Linux 2 in public subnet tag it as SECONDARY

> In order to connect to EFS storage we need NFS utilities

> Install NFS utilities on SECONDARY instance, by following the instructions given in EFS page once EFS is in available state

**S3 - Simple Storage Service**

* Cloud storage is a cloud computing model that stores data on the Internet through a cloud computing provider(AWS), who manages and operates
  + **data storage as a service**
* It’s delivered on demand which eliminates buying and managing your own data storage infrastructure. This gives you “**anytime**, **anywhere**” data access.

Types Of Storage

AWS provides three popular services :

* **Simple Storage Service (S3)**
* **Elastic Block Store (EBS)**
* **Elastic File System (EFS)**
* Above services work quite differently and offer different levels of performance, cost, availability and scalability.
* **AWS EBS** provides persistent block storage which offer higher performance than object storage. You need to mount EBS onto an Amazon EC2 instance. Use cases include transactional databases management, business continuity.
* **AWS EFS** is a shared, elastic file storage system that grows and shrinks as you add and remove files. You can mount EFS onto several EC2 instances at the same time.
* **Amazon S3** provides simple object storage, useful for hosting website images and videos. You can access the S3 service from anywhere on the internet.
* **AWS EBS** is scalable up or down. EBS is cheaper than EFS, you can use it for database backups and other low-latency interactive applications that require consistent, predictable performance.
* **AWS EFS** is best used for large quantities of data, such as large analytic workloads. Data at this scale cannot be stored on a single EC2 instance. The EFS service allows concurrent access to thousands of EC2 instances, making it possible to process and analyze large amounts of data seamlessly.
* **Amazon S3** cheapest for data storage and can be accessed from anywhere. EBS and EFS are both faster than Amazon S3, with high IOPS and lower latency.

Object Storage

* **Object Storage** stores the object (file), it's metadata and global unique ID
* In object storage there is no limit on type or amount for objects. Examples of object Storage : S3, Drop Box, Facebook { videos images }
* Object storage cannot be mounted as drive or directory to EC2 instance.
* Object storage is perfect solution for data growth storage problems
* Companies today need the ability to simply and securely collect, store, and analyze their data at a massive scale. Amazon **S3** is object storage built to store and retrieve any amount of data from anywhere.
* With S3, you manage your storage in one place with an easy-to-use application interface i.e AWS Management Console.
* You can use S3 to optimize storage costs, tiering between different storage classes automatically. AWS makes storage easier to use to perform analysis, gain insights, and make better decisions faster.

S3 Essentials

* As AWS main storage service, S3 can serve many purposes when designing highly available, fault tolerant and secure application architecture including:
  + Bulk (basically unlimited) static object storage.
  + Various storage classes to optimize cost vs needed object availability/durability
  + Object versioning
  + Access restrictions via S3 bucket policies
  + Object management via lifecycle policies
  + Hosting static files & websites
  + File shares and backup/archiving for hybrid networks (via AWS Storage gateway)
* Amazon S3 is Global Service.
* Objects stay within an AWS region and are synced across all Az's for extremely high availability and durability.
* You should always create an S3 bucket in a region that makes sense to its purpose:
* For better performance, lower latency and to minimize costs, create an S3 bucket closer to client location.

Buckets

* Data is stored in Buckets, Buckets are the main storage containers of S3.
* You can store unlimited objects in a bucket, but an object cannot exceed 5 TB
* S3 bucket is Region specific. **Each bucket must have a unique name across ALL of AWS.**
  + http://**s3**.amazonaws.com/[bucket\_name]
* Bucket names cannot be changed once created and ownership is not transferable.
* By default Buckets are private.

Objects

* By default, all objects are private.
* Objects stored in S3 bucket in a region will never leave that region unless we specify by enabling Cross Region Replication.
* S3 provides high availability, Objects are redundantly stored on multiple devices across multiple facilities(AZ's) in an region where bucket exists.

Managing Access

* By default, all amazon S3 resources are private.
  + Only a resource owner can access the resources.
* A bucket owner can grant cross-account permissions to another AWS accounts (users in another account) to upload objects.
* Managing access refers to granting (AWS Accounts & Users) permissions to perform the resource operations by writing an access policy.
* You can grant S3 bucket/object permissions to:
  + Individual Users, AWS Accounts & Make resources public (grant permissions to everyone)
* Access policy describes who has access to what. You can associate an access policy with a S3 resource(Bucket & Objects) or a User.
* Amazon S3 access policies are as follows:
  + Resource based policies
    - ACL's (Bucket & Object ACL)
    - Bucket Policy
  + User Access policies (IAM)
* Bucket Policies and ACL's are resource based because you attach them to Amazon S3 resources.
* ACL's (Bucket & Object ACL's)
  + Each bucket & object can have ACL associated with it, You can use ACL's to grant basic read/write permissions to other accounts and public.
* Bucket Policy
  + For your bucket, you can add bucket policy to grant other aws accounts, to have access on bucket and objects inside it.
  + Bucket policies are preferred over ACL's(Legacy)

Bucket ACL

* S3 access control lists (ACL's) enable you to manage access to buckets and objects. Each bucket and object can have acl attached.
* ACL's define which AWS accounts are granted access and type of access.
* You cannot provide permissions to individual IAM users here.

Storage Classes

* A Storage Class represents the classification assigned to each object in S3. Amazon S3 offers a range of storage classes designed for different use cases.
* Each storage class has varying attributes that dictate things like:
  + Storage cost
  + Object availability
  + Object durability
  + Frequency of access (to the object)
* Current Storage Class Types include:
  + **Amazon S3 Standard**
  + **Amazon S3 Intelligent-Tiering**
  + **Amazon S3 Standard-Infrequent Access**
  + **Amazon S3 One Zone-Infrequent Access**
  + **Amazon Glacier**
  + **Amazon Glacier Deep Archive**

Amazon S3 offers a range of storage classes designed for different use cases. These include

* S3 Standard for general-purpose storage of frequently accessed data
* S3 Intelligent-Tiering for data with unknown or changing access patterns
* S3 Standard-Infrequent Access (S3 Standard-IA) and S3 One Zone-Infrequent Access (S3 One Zone-IA) for long-lived, but less frequently accessed data
* Amazon S3 Glacier (S3 Glacier) and Amazon S3 Glacier Deep Archive (S3 Glacier Deep Archive) for long-term archive and digital preservation.

Amazon S3 also offers capabilities to manage your data throughout its lifecycle. Once an S3 Lifecycle policy is set, your data will automatically transfer to a different storage class without any changes to your application.

Amazon S3 Standard (S3 Standard)

S3 Standard offers high durability, availability, and performance object storage for frequently accessed data. Because it delivers low latency and high throughput, S3 Standard is appropriate for a wide variety of use cases, including cloud applications, dynamic websites, content distribution, mobile and gaming applications, and big data analytics.

Key Features:

* Low latency and high throughput performance
* Designed for durability of 99.999999999% of objects across multiple Availability Zones
* Resilient against events that impact an entire Availability Zone
* Designed for 99.99% availability over a given year
* Backed with the [Amazon S3 Service Level Agreement](https://aws.amazon.com/s3/sla/) for availability
* Supports SSL for data in transit and encryption of data at rest
* S3 Lifecycle management for automatic migration of objects to other S3 Storage Classes

Unknown or changing access

Amazon S3 Intelligent-Tiering (S3 Intelligent-Tiering)

The S3 Intelligent-Tiering storage class is designed to optimize costs by automatically moving data to the most cost-effective access tier, without performance impact or operational overhead. It works by storing objects in two access tiers: one tier that is optimized for frequent access and another lower-cost tier that is optimized for infrequent access. For a small monthly monitoring and automation fee per object, Amazon S3 monitors access patterns of the objects in S3 Intelligent-Tiering, and moves the ones that have not been accessed for 30 consecutive days to the infrequent access tier. If an object in the infrequent access tier is accessed, it is automatically moved back to the frequent access tier. There are no retrieval fees when using the S3 Intelligent-Tiering storage class, and no additional tiering fees when objects are moved between access tiers. It is the ideal storage class for long-lived data with access patterns that are unknown or unpredictable. S3 Storage Classes can be configured at the object level and a single bucket can contain objects stored in S3 Standard, S3 Intelligent-Tiering, S3 Standard-IA, and S3 One Zone-IA. You can upload objects directly to S3 Intelligent-Tiering, or use S3 Lifecycle policies to transfer objects from S3 Standard and S3 Standard-IA to S3 Intelligent-Tiering. You can also archive objects from S3 Intelligent-Tiering to S3 Glacier.

Key Features:

* Same low latency and high throughput performance of S3 Standard
* Small monthly monitoring and auto-tiering fee
* Automatically moves objects between two access tiers based on changing access patterns
* Designed for durability of 99.999999999% of objects across multiple Availability Zones
* Resilient against events that impact an entire Availability Zone
* Designed for 99.9% availability over a given year
* Backed with the [Amazon S3 Service Level Agreement](https://aws.amazon.com/s3/sla/) for availability
* Supports SSL for data in transit and encryption of data at rest
* S3 Lifecycle management for automatic migration of objects to other S3 Storage Classes

Infrequent access

Amazon S3 Standard-Infrequent Access (S3 Standard-IA)

S3 Standard-IA is for data that is accessed less frequently, but requires rapid access when needed. S3 Standard-IA offers the high durability, high throughput, and low latency of S3 Standard, with a low per GB storage price and per GB retrieval fee. This combination of low cost and high performance make S3 Standard-IA ideal for long-term storage, backups, and as a data store for disaster recovery files. S3 Storage Classes can be configured at the object level and a single bucket can contain objects stored across S3 Standard, S3 Intelligent-Tiering, S3 Standard-IA, and S3 One Zone-IA. You can also use S3 Lifecycle policies to automatically transition objects between storage classes without any application changes.

Key Features:

* Same low latency and high throughput performance of S3 Standard
* Designed for durability of 99.999999999% of objects across multiple Availability Zones
* Resilient against events that impact an entire Availability Zone
* Data is resilient in the event of one entire Availability Zone destruction
* Designed for 99.9% availability over a given year
* Backed with the [Amazon S3 Service Level Agreement](https://aws.amazon.com/s3/sla/) for availability
* Supports SSL for data in transit and encryption of data at rest
* S3 Lifecycle management for automatic migration of objects to other S3 Storage Classes

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

S3 One Zone-IA is for data that is accessed less frequently, but requires rapid access when needed. Unlike other S3 Storage Classes which store data in a minimum of three Availability Zones (AZs), S3 One Zone-IA stores data in a single AZ and costs 20% less than S3 Standard-IA. S3 One Zone-IA is ideal for customers who want a lower-cost option for infrequently accessed data but do not require the availability and resilience of S3 Standard or S3 Standard-IA. It’s a good choice for storing secondary backup copies of on-premises data or easily re-creatable data. You can also use it as cost-effective storage for data that is replicated from another AWS Region using S3 Cross-Region Replication.

S3 One Zone-IA offers the same high durability†, high throughput, and low latency of S3 Standard, with a low per GB storage price and per GB retrieval fee. S3 Storage Classes can be configured at the object level, and a single bucket can contain objects stored across S3 Standard, S3 Intelligent-Tiering, S3 Standard-IA, and S3 One Zone-IA. You can also use S3 Lifecycle policies to automatically transition objects between storage classes without any application changes.

Key Features:

* Same low latency and high throughput performance of S3 Standard
* Designed for durability of 99.999999999% of objects in a single Availability Zone†
* Designed for 99.5% availability over a given year
* Backed with the [Amazon S3 Service Level Agreement](https://aws.amazon.com/s3/sla/) for availability
* Supports SSL for data in transit and encryption of data at rest
* S3 Lifecycle management for automatic migration of objects to other S3 Storage Classes

† Because S3 One Zone-IA stores data in a single AWS Availability Zone, data stored in this storage class will be lost in the event of Availability Zone destruction.

Archive

Amazon S3 Glacier (S3 Glacier)

S3 Glacier is a secure, durable, and low-cost storage class for data archiving. You can reliably store any amount of data at costs that are competitive with or cheaper than on-premises solutions. To keep costs low yet suitable for varying needs, S3 Glacier provides three retrieval options that range from a few minutes to hours. You can upload objects directly to S3 Glacier, or use S3 Lifecycle policies to transfer data between any of the S3 Storage Classes for active data (S3 Standard, S3 Intelligent-Tiering, S3 Standard-IA, and S3 One Zone-IA) and S3 Glacier. For more information, visit the [Amazon S3 Glacier page »](https://aws.amazon.com/glacier/)

Key Features:

* Designed for durability of 99.999999999% of objects across multiple Availability Zones
* Data is resilient in the event of one entire Availability Zone destruction
* Supports SSL for data in transit and encryption of data at rest
* Low-cost design is ideal for long-term archive
* Configurable retrieval times, from minutes to hours
* S3 PUT API for direct uploads to S3 Glacier, and S3 Lifecycle management for automatic migration of objects

Amazon S3 Glacier Deep Archive (S3 Glacier Deep Archive)

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 the Financial Services, Healthcare, and Public Sectors — that retain data sets for 7-10 years or longer to meet regulatory compliance requirements. S3 Glacier Deep Archive can also be used for backup and disaster recovery use cases, and is a cost-effective and easy-to-manage alternative to magnetic tape systems, whether they are on-premises libraries or off-premises services. S3 Glacier Deep Archive complements Amazon S3 Glacier, which is ideal for archives where data is regularly retrieved and some of the data may be needed in minutes. All objects stored in S3 Glacier Deep Archive are replicated and stored across at least three geographically-dispersed Availability Zones, protected by 99.999999999% of durability, and can be restored within 12 hours.

Key Features:

* Designed for durability of 99.999999999% of objects across multiple Availability Zones
* Lowest cost storage class designed for long-term retention of data that will be retained for 7-10 years
* Ideal alternative to magnetic tape libraries
* Retrieval time within 12 hours
* S3 PUT API for direct uploads to S3 Glacier Deep Archive, and S3 Lifecycle management for automatic migration of objects

S3 Lifecycle policies

* An object lifecycle policy is a set of rules that automate the migration of an object's storage class to a different storage class (or deletion) based on specified time intervals.
* By default, lifecycle policies are disabled on a bucket.
* Are customizable to meet your company's data retention policies.
* Great for automating the management of object storage and to be more cost efficient.
* Example:
* I have a work file that I am going to access everyday for the next 30 days.
* After 30 days, I may only need to access that file once a week for the next 60 days.
* After which (90 days total) I will probably never access the file again but want to keep it just in case.

S3 Versioning

* S3 versioning is a feature to manage and store versions of an object
* S3 versioning protects your data against accidental deletion.
* By default, versioning is disabled on all buckets.
* Once versioning is enabled, you can only "suspend" versioning. It cannot be fully disabled.
* Suspending versioning only prevents new versions from being created. All objects with existing versions will maintain their older versions.
* Versioning can only be set on the **bucket level** and applies to ALL objects in the bucket.
* Versioning and lifecycle policies can both be enabled on a bucket at the same time.
* Versioning can be used with lifecycle policies to create a great archiving and backup solution in S3.

S3 Web Hosting

* Amazon S3 provides an option for low-cost, highly reliable web hosting service for static websites (content that does not change frequently).
* When enabled, static web hosting will provide you with an unique endpoint (url) that you can point to any properly formatted file stored in an S3 bucket. Supported formats include:
* HTML
* CSS
* JavaScript
* Amazon Route S3 can also map human-readable domain names to static web hosting buckets, which are ideal for DNS failover solutions.

> Ecommerce Application Code is hosted in

[**https://github.com/Akiranred/ecomm**](https://github.com/Akiranred/ecomm)

S3 Cross Region Replication

* Cross-region replication is a bucket-level configuration that enables automatic copying of objects across buckets in different AWS Regions.
* We refer to these buckets as **source bucket** and **destination bucket**.
* To activate this feature, you add a replication configuration to your source bucket. In the replication configuration, you provide information such as the following:
  + The **destination bucket** where you want Amazon S3 to replicate the objects.
* You can replicate objects from a source bucket to only one destination bucket i.e you cannot replicate to multiple buckets.
* The source and destination buckets must have **versioning enabled**.
* The source and destination buckets must be in different AWS Regions.
* FIles in an existing bucket are not replicated automatically, all subsequent updated files will be replicated automatically.

IAM - Identity & Access Management

**Account & Services Layer**

**Root User**

* The user created when you first create your AWS account is called the "root" user.
* It's credentials are the email address and password used when signing up for an AWS account.
* By default, **the root user has FULL administrative rights** and access to every part of the account.
* **Best practices for Root user**
* You should not use the root user for daily works and AWS administration. You should create a secondary user(IAM User) that has admin rights and sign in with that user for daily work.
* You should always protect your user account with **MFA**.

**AWS Users / IAM Users**

* This represents an AWS users that you may create (in IAM), who will have varying degrees of access to the AWS account
* We also have a different set of users like Developer users that have access to the dev user account.
* This is how organizations keep their accounts seperate accounts for users.

**Access Ways**

* The lines coming down from AWS users represent the two main ways of connecting to AWS.
* AWS **Console** - **GUI Based**
* AWS **Programmatic** - **CLI/SDK/API**

**AWS Management Console**

* The AWS **Management Console** (generally referred to as the "console") is the primary means for which we will access and interact with AWS.
* Access and manage Amazon Web Services through a simple and intuitive web-based user interface.
* All actions done in the console are API Calls.
* Features
  + Administer your AWS account
  + Finding Services
    - *Recently visited services* section, or expand the *All services*
    - list of all services, either grouped, or arranged alphabetically
* Pin Service Shortcuts

**IAM Components**

* AWS - IAM(Identity & Access management) helps you securely control access to AWS resources. You use IAM to control who is authenticated (sign in) and authorized (has permissions) to use resources.
* IAM is where you manage your AWS users, groups, roles and their access to AWS accounts and services:
* IAM provides access and access permissions to AWS resources (such as EC2, S3 etc)
* IAM is global to all AWS regions, creating a user account will apply to all the regions.
* The common use of IAM is to manage:
* **Users**
* **Groups**
* **Roles**
* **Policies**
* By default, any new IAM new user you create in an AWS account is created with NO access to any AWS services. This is a **non-explicit deny rule set on all new IAM users**.
* For all the users (besides the root user), permissions must be given, that grant access to AWS services

**Security Checks**

* When a new AWS root account is created, it is a "**best practice**" to complete the tasks listed in IAM under "Security Status" - which includes:
  + Delete your root access keys
  + Activate MFA on your root account
  + Create individual IAM users
  + Users groups to assign permissions
  + Apply an IAM password policy
* **Best practice is to login and do daily work as an IAM user - NOT as the root user.**

**Creating IAM Users (Console)**

You can use the AWS Management Console to create IAM users.

**To create one or more IAM users (console)**

1. Sign in to the AWS Management Console and open the IAM console at<https://console.aws.amazon.com/iam/>.
2. In the navigation pane, choose **Users** and then choose **Add user**.
3. Type the user name for the new user. This is the sign-in name for AWS. If you want to add more than one user at the same time, choose **Add another user** for each additional user and type their usernames. You can add up to 10 users at one time.  
    **Note** User names can be a combination of up to 64 letters, digits, and these characters: plus (+), equal (=), comma (,), period (.), at sign (@), underscore (\_), and hyphen (-). Names must be unique within an account. They are not distinguished by case. For example, you cannot create two users named *TESTUSER* and *testuser*.
4. Select the type of access this set of users will have. You can select programmatic access, access to the AWS Management Console, or both.

* Select **Programmatic access** if the users require access to the API, AWS CLI, or Tools for Windows PowerShell. This creates an access key for each new user. You can view or download the access keys when you get to the **Final** page.
* Select **AWS Management Console access** if the users require access to the AWS Management Console. This creates a password for each new user.

1. For **Console password**, choose one of the following:
   * **Autogenerated password**. Each user gets a randomly generated password that meets the account password policy in effect (if any). You can view or download the passwords when you get to the **Final** page.
   * **Custom password**. Each user is assigned the password that you type in the box.
2. Choose **Next: Permissions**.
3. On the **Set permissions** page, specify how you want to assign permissions to this set of new users. Choose one of the following three options:
   1. **Add user to group**. Choose this option if you want to assign the users to one or more groups that already have permissions policies.
   2. **Copy permissions from existing users** Choose this option to copy all of the group memberships, attached managed policies from an existing user to the new users.
   3. **Attach existing policies to users directly**. Choose this option to see a list of the AWS managed and customer managed policies in your account. Select the policies that you want to attach to the new user
4. (Optional) Set a [permissions boundary](https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_boundaries.html). This is an advanced feature.
5. Choose **Next: Tags**.
6. (Optional) Add metadata to the user by attaching tags as key-value pairs.
7. Choose **Next: Review** to see all of the choices you made up to this point. When you are ready to proceed, choose **Create user**.
8. To view the users' access keys (access key IDs and secret access keys), choose **Show** next to each password and access key that you want to see. To save the access keys, choose **Download .csv** and then save the file to a safe location.  
    **Important** This is your only opportunity to view or download the secret access keys, and you must provide this information to your users before they can use the AWS API. Save the user's new access key ID and secret access key in a safe and secure place. **You will not have access to the secret keys again after this step.**
9. Provide each user with his or her credentials. On the final page you can choose **Send email** next to each user. Your local mail client opens with a draft that you can customize and send. The email template includes the following details to each user:

* User name
* URL to the account sign-in page. Use the following example, substituting the correct account ID number or account alias:
* https://*AWS-account-ID or alias*.signin.aws.amazon.com/console

**LAB - Create IAM Users**

* AWS strongly recommends that you do not use the root user for your everyday tasks, even the administrative ones.
* Instead, adhere to the [**best practice of using the root user only to create your first IAM user**](http://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#create-iam-users)**.**
* So let's create a user called admin and will use this user as our daily driver.
  + Services → IAM → Users → Add User( name: admin) → Check ✅ both Programmatic access and Management console access → Custom Password → Next → Review → Says User has no permissions → Create User
  + I'll not set the permissions right away, will set the permissions later on

**Groups**

* Similarly, if a person changes jobs in your organization, instead of editing that user's permissions, you can remove him or her from the old groups and add him or her to the appropriate new groups.
* An IAM Group, is a collection of IAM users.
* Groups let you specify permissions for multiple users, which can make it easier to manage the permissions for those users.
* For example, you could have a group called Admins and give that group the types of permissions that administrators typically need.
* Any user in that group automatically has the permissions that are assigned to the group. If a new user joins your organization and needs administrator privileges, you can assign the appropriate permissions by adding the user to that group.
* If a person changes jobs in your organization, instead of editing that user's permissions, you can remove him or her from the old groups and add him or her to the appropriate new groups.
* Now groups are a great way to simplify the process of granting or restricting access.

The following diagram shows a simple example of a small company. The company owner creates an **Admins** group for users to create and manage other users as the company grows. The **Admins** group creates a **Developers** group and a **Test** group. Each of these groups consists of users (humans and applications) that interact with AWS (Jim, Brad, DevApp1, and so on). Each user has an individual set of security credentials. In this example, each user belongs to a single group. However, users can belong to multiple groups.

**Creating IAM Group (Console)**

1. Sign in to the AWS Management Console and open the IAM console at<https://console.aws.amazon.com/iam/>.
2. In the navigation pane, click Groups and then click Create New Group.
3. In the **Group Name** box, type the name of the group and then click **Next Step**.  
    **Note** Group names can be a combination of up to 64 letters, digits, and these characters: plus (+), equal (=), comma (,), period (.), at sign (@), underscore (\_), and hyphen (-). Names must be unique within an account. They are not distinguished by case. For example, you cannot create groups named both **ADMINS** and **admins**.
4. In the list of policies, select the check box for each policy that you want to apply to all members of the group. Then click **Next Step**.
5. Click **Create Group**.

**IAM Policies**

* You manage authorization in AWS by creating policies and attaching them to IAM identities (users, groups of users, or roles)
* A policy is an object in AWS that, when associated with an identity or resource, defines their permissions.
* AWS evaluates these policies when an IAM principal (user or role) makes a request. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents.
* More than one policy can be attached to a principal (or identity), at the same time.
* Policies **cannot be directly attached to AWS resources** ( such as EC2 instance).
* It is not necessary for you to understand the JSON syntax. You can use the visual editor in the AWS Management Console to create and edit customer managed policies without ever using JSON.

**JSON Policy Structure**

A JSON policy document includes these elements:

* **Version** – Specify the version of the policy language that you want to use. As a best practice, use the latest 2012-10-17 version.
* **Statement** – Use this main policy element as a container for the following elements. You can include more than one statement in a policy.
* **Sid** (Optional) – Include an optional statement ID to differentiate between your statements.
* **Effect** – Use Allow or Deny to indicate whether the policy allows or denies access.
* **Action** – Include a list of actions that the policy allows or denies.
* **Resource** – If you create an IAM permissions policy, you must specify a list of resources to which the actions apply. If you create a resource-based policy, this element is optional. If you do not include this element, then the resource to which the action applies is the resource to which the policy is attached.
* **Condition** (Optional) – Specify the circumstances under which the policy grants permission.

**AWS Programmatic Access**

**Programmatic access:** The IAM user might need to make API calls, use the AWS CLI, or use the SDK Tools. In that case, create an access key (access key ID and a secret access key) for that user.

* **Important API Access Key Facts**:
  + Secret keys are only available ONE time, when a new user is created OR when you reissue a new set of keys
  + AWS will Not regenerate the same set of access keys again
  + In the AWS console you can only see the Access Key ID - never the Secret Key ID
  + If you require new API Key credentials, you can generate new ones.

**IAM Roles**

* An IAM role is similar to a user, in that it is an AWS identity with permission policies that determine what the identity can and cannot do in AWS.
* However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it.
* Users and roles use policies for authorization. Keep in mind that user and role can't do anything until you allow certain actions with a policy.
* Also, a role does not have standard long-term credentials (password or access keys)
* In the context of this course, "entities" that can assume a role includes AWS resources (such as an EC2 instance)
* Roles must be used because policies cannot be directly attached to AWS services such as EC2 instances.
* **If you are using an EC2 instance and it need to access an S3 buckets, You "can" but should never pass or store credentials in or to an EC2 instance - so roles are used instead**
* I AM User - Long lived credentials
* I AM Role - Short lived credentials

**AWS CLI VPC**

> The current options we have to create resources

> Management Console [ GUI / Browser ]

> **AWS CLI [ Commands ]**

> AWS Cloudformation [ Code ]

> VPC setup with CLI

> aws ec2 create-vpc --cidr-block 10.0.0.0/16

> aws ec2 create-tags --resources vpc-028ec66787ef8a909 --tags Key=Name,Value=IBM

> aws ec2 create-internet-gateway

> aws ec2 create-tags --resources igw-09149c3058881ca09 --tags Key=Name,Value=IBM-IGW

> aws ec2 attach-internet-gateway --internet-gateway-id igw-09149c3058881ca09 --vpc-id vpc-028ec66787ef8a909

> aws ec2 create-subnet --vpc-id vpc-028ec66787ef8a909 --cidr-block 10.0.0.0/24

> aws ec2 create-tags --resources subnet-0914930263dc2e820 --tags Key=Name,Value=IBM-PUB

> aws ec2 create-route-table --vpc-id vpc-028ec66787ef8a909

> aws ec2 create-tags --resources rtb-0b574098dc526fbad --tags Key=Name,Value=IBM-PUB-RT

> aws ec2 create-route --route-table-id rtb-0b574098dc526fbad --destination-cidr-block 0.0.0.0/0 --gateway-id igw-09149c3058881ca09

> aws ec2 associate-route-table --route-table-id rtb-0b574098dc526fbad --subnet-id subnet-0914930263dc2e820

> aws ec2 modify-subnet-attribute --subnet-id subnet-0914930263dc2e820 --map-public-ip-on-launch

> aws ec2 create-subnet --vpc-id vpc-028ec66787ef8a909 --cidr-block 10.0.1.0/24

> aws ec2 create-tags --resources subnet-0399ca9a2b77f0dca --tags Key=Name,Value=IBM-PVT

> aws ec2 create-security-group --group-name IBM-SSH --description "IBM SSH" --vpc-id vpc-028ec66787ef8a909

> aws ec2 authorize-security-group-ingress --group-id sg-0d9ed2aa62ea6f552 --protocol tcp --port 22 --cidr 0.0.0.0/0

> aws ec2 run-instances --image-id ami-0dc2d3e4c0f9ebd18 --instance-type t2.micro --key-name kiran --subnet-id subnet-0914930263dc2e820 --security-group-ids sg-0d9ed2aa62ea6f552

AWS CloudFormation

* AWS CloudFormation is a service that helps you model and set up your AWS resources so that you can spend less time managing those resources and more time focusing on your applications that run in AWS.
* You create a template that describes all the AWS resources that you want (like Amazon EC2 instances or Amazon S3 Buckets), and AWS CloudFormation takes care of provisioning and configuring those resources. You don't need to individually create and configure AWS resources.

**How CloudFormation can help/When to use Cloud Formation ??**

**Simplify Infrastructure Management**

* For a scalable web application that also includes a back-end database, you might use an Auto Scaling group, an Elastic Load Balancing load balancer, and an Amazon Relational Database Service database instance. Normally, you might use each individual service to provision these resources. And after you create the resources, you would have to configure them to work together. All these tasks can add complexity and time before you even get your application up and running.
* Instead, you can create or modify an existing AWS CloudFormation template. A template describes all of your resources and their properties. When you use that template to create an AWS CloudFormation stack, AWS CloudFormation provisions the Auto Scaling group, load balancer, and database for you.
* By using AWS CloudFormation, you easily manage a collection of resources as a single unit.

**Quickly Replicate Your Infrastructure**

* If your application requires additional availability, you might replicate it in multiple regions so that if one region becomes unavailable, your users can still use your application in other regions. The challenge in replicating your application is that it also requires you to replicate your resources. Not only do you need to record all the resources that your application requires, but you must also provision and configure those resources in each region.
* When you use AWS CloudFormation, you can reuse your template to set up your resources consistently and repeatedly.

**Easily Control and Track Changes to Your Infrastructure**

* When you provision your infrastructure with AWS CloudFormation, the AWS CloudFormation template describes exactly what resources are provisioned and their settings. Because these templates are text files, you simply track differences in your templates to track changes to your infrastructure, similar to the way developers control revisions to source code.
* For example, you can use a version control system with your templates so that you know exactly what changes were made, who made them, and when. If at any point you need to reverse changes to your infrastructure, you can use a previous version of your template.

**Tit-Bits**

* Templates can be stored in Version Control System
* Track all changes made to infrastructure stack
* Create and update resources in a controlled and predictable way
* CF is declarative and flexible, meaning just choose the resources and configurations you need

AWS CloudFormation Concepts

* When you use AWS CloudFormation, you work with **templates** and **stacks**.
* You create templates to describe your AWS resources and their properties.
* Whenever you create a stack, AWS CloudFormation provisions the resources that are described in your template.

**Templates**

**https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/template-formats.html**

* An AWS CloudFormation template is a JSON or YAML formatted text file. You can save these files with any extension, such as .json, .yaml, .template, or .txt.
* AWS CloudFormation uses these templates as blueprints for building your AWS resources.
* For example, in a template, you can describe an Amazon EC2 instance, such as the instance type, the AMI ID, block device mappings, and its Amazon EC2 key pair name. Whenever you create a stack, you also specify a template that AWS CloudFormation uses to create whatever you described in the template.
* *For example, if you created a stack with the following template, AWS CloudFormation provisions an instance with user choice to select Key Pair Name, Instance Type etc*

**Stacks**

* When you use AWS CloudFormation, you manage related resources as a single unit called a stack. You create, update, and delete a collection of resources by creating, updating, and deleting stacks.
* All the resources in a stack are defined by the stack's AWS CloudFormation template. Suppose you created a template that includes an Auto Scaling group, ELB, and an Amazon RDS database instance.
* To create those resources, you create a stack by submitting the template that you created, and AWS CloudFormation provisions all those resources for you.

**Templates Anatomy**

A template is a **JSON or YAML-formatted text file** that describes your AWS infrastructure. The following examples show an AWS CloudFormation template structure and its sections.

**JSON**

**YAML**

**LAB - VPC**

| AWSTemplateFormatVersion: "2010-09-09"  Description: A VPC Template  Resources:  VPC: # IBM VPC Resource  Type: "AWS::EC2::VPC"  Properties:  CidrBlock: 10.0.0.0/16  InstanceTenancy: default  Tags:  - Key: Name  Value: IBM  InternetGateway: # IBM Internet Gateway  Type: AWS::EC2::InternetGateway  Properties:  Tags:  - Key: Name  Value: IBM-IGW  AttachGateway: # Attach Internet Gateway - IBM VPC  Type: AWS::EC2::VPCGatewayAttachment  Properties:  VpcId:  Ref: VPC  InternetGatewayId:  Ref: InternetGateway  PubSubnet1: # Public Subnet  Type: AWS::EC2::Subnet  Properties:  VpcId:  Ref: VPC  CidrBlock: 10.0.0.0/24  AvailabilityZone: "us-east-1a"  MapPublicIpOnLaunch: 'true'  Tags:  - Key: Name  Value: IBM-Pub-Subnet1  PvtSubnet1: # Private Subnet  Type: AWS::EC2::Subnet  Properties:  VpcId:  Ref: VPC  CidrBlock: 10.0.1.0/24  AvailabilityZone: "us-east-1b"  MapPublicIpOnLaunch: 'false'  Tags:  - Key: Name  Value: IBM-Pvt-Subnet1  PublicRouteTable: # Public Route Table  Type: AWS::EC2::RouteTable  Properties:  VpcId:  Ref: VPC  Tags:  - Key: Name  Value: IBM-Pub-RT  PrivateRouteTable: # Private Route Table  Type: AWS::EC2::RouteTable  Properties:  VpcId:  Ref: VPC  Tags:  - Key: Name  Value: IBM-Pvt-RT  PublicRoute: # Route To IGW  Type: AWS::EC2::Route  DependsOn: AttachGateway  Properties:  RouteTableId:  Ref: PublicRouteTable  DestinationCidrBlock: 0.0.0.0/0  GatewayId:  Ref: InternetGateway  PubSubnetRouteTableAssociation: # Pub Sub Association  Type: AWS::EC2::SubnetRouteTableAssociation  Properties:  SubnetId:  Ref: PubSubnet1  RouteTableId:  Ref: PublicRouteTable  PvtSubnetRouteTableAssociation: # Pvt Sub Association  Type: AWS::EC2::SubnetRouteTableAssociation  Properties:  SubnetId:  Ref: PvtSubnet1  RouteTableId:  Ref: PrivateRouteTable |
| --- |

ADVANCED VPC NETWORKING

VPC Peering

* A VPC peering connection is a networking connection between two VPCs that enables you to route traffic between them using private IP addresses
* Instances in either VPC can communicate with each other as if they are within the same network.
* You can create a VPC peering connection between
  + your own VPCs
  + with a VPC in another AWS account
  + VPC's can be in different regions
* You can also use a VPC peering connection to allow other VPCs to access resources you have in one of your VPCs.

**Peering Basics**

* To establish a VPC peering connection, you do the following:
* The owner of the requester VPC sends a request to the owner of the accepter VPC to create the VPC peering connection. The accepter VPC can be owned by you, or another AWS account, and **cannot have a CIDR block that overlaps with the requester VPC's CIDR block**.
* The owner of the accepter VPC accepts the VPC peering connection request to activate the VPC peering connection.
* To enable the flow of traffic between the VPCs using private IP addresses, the owner of each VPC in the VPC peering connection must manually add a route to one or more of their VPC route tables that points to the IP address range of the other VPC (the peer VPC).

**Rules**

* A VPC peering connection is a one to one relationship between two VPCs.
* You can create multiple VPC peering connections for each VPC that you own, but transitive peering relationships are not supported.
* You do not have any peering relationship with VPCs that your VPC is not directly peered with.

**Peering Limitations**

* You cannot create a VPC peering connection between VPCs that have matching or overlapping CIDR blocks
* You have a limit on VPC peering connections that you can have per VPC
  + Active VPC peering connections per VPC is 50
* VPC peering does not support transitive peering relationships.

**Security For Instances**

* So when we are talking about increased security, what we want to focus on is, placing of EC2 instances that are holding our data in private subnets.
* However this causes some issues, we cannot serve traffic from private instances as there is no route to the open internet and also we cannot access or ssh into EC2 instances that are in private subnets or install/update the software on those EC2 instances.
* However with using a Bastion Host and a NAT Gateway we can actually accomplish the above tasks and have our EC2 instances protected as well.
* Setting up a Bastion Host will allow you to SSH and access the EC2 instances, and a NAT Gateway, will allow EC2 instances to reach the open internet and install the software packages, but before that let's talk about new concepts like **Bastion Host** and **NAT Gateway**.

Bastion Host

* A **Bastion Host** is an EC2 instance that lives in a public subnet, and is used as a "**gateway**" for traffic that is destined for instances that live in private subnets.
* This means that we can use a bastion host as a "portal" to access EC2 instances that are located in a private subnet.
* A bastion host is considered the "**critical strong point**" of the network - as all traffic must pass through it first.
* Taking a look at the diagram, traffic coming from AWS users from open internet, through SSH, coming down through the IGW, and into the Bastion Host
* Coz, Bastion Host will be in our public subnet, that is associated with Route Table with IGW attached, the Bastion Host then will act as a portal for us to access any other internal resources, since we are inside the VPC N/W, so if you recall, all the instances within a VPC regardless of whether they are in public or private subnets can communicate with each other.
* So if we were able to access the bastion host, then we can access the instances that are in private subnets.
* A bastion host should have increased and extremely tight security
* A bastion host can be used as an access point to "SSH" into an internal network (to access private resources) without a VPN (virtual private network).
* A bastion host is a system identified by the firewall administrator as a **critical strong point** in the **network's security**. Generally, bastion hosts will have some degree of extra attention paid to their security and may undergo regular audits
* So Bastion Host is going to be a access point for us to reach other resources in private parts of AWS VPC network
* Now once we have access to the private instances i.e ssh into private instances. We still won't be able to install any software packages or update the softwares, coz this is just one way connection
* We cannot send traffic from these private instances to open internet, so in order to solve that problem, we will go with **NAT Gateway**.

**Internet Gateway**

* An Internet Gateway (IGW) is a logical connection between an Amazon VPC and the Internet. It is not a physical device. Only one can be associated with each VPC. It does not limit the bandwidth of Internet connectivity. (The only limitation on bandwidth is the size of the Amazon EC2 instance, and it applies to all traffic — internal to the VPC and out to the Internet.)
* If a VPC does not have an Internet Gateway, then the resources in the VPC cannot be accessed from the Internet (unless the traffic flows via a corporate network and VPN/Direct Connect).
* An Internet Gateway allows resources within your VPC to access the internet, and vice versa. In order for this to happen, there needs to be a routing table entry allowing a subnet to access the IGW.
* That is to say — an IGW allows resources within your public subnet to access the internet, and the internet to access said resources.
* A subnet is deemed to be a Public Subnet if it has a Route Table that directs traffic to the Internet Gateway.

**NAT Gateway**

* A NAT Gateway does something similar, but with two main differences:
* It allows resources in a private subnet to access the internet (think yum updates, external database connections, wget calls, OS patch, etc).
* It only works one way. The internet at large cannot get through your NAT to your private resources unless you explicitly allow it.
* AWS introduced a NAT Gateway Service that can take the place of a NAT Instance. The benefits of using a NAT Gateway service are:
* It is a fully-managed service — just create it and it works automatically, including fail-over.
* A NAT gateway supports 5 Gbps of bandwidth and automatically scales up to 45 Gbps. (a NAT Instance is limited to the bandwidth associated with the EC2 instance type).

LAB - Bastion & NAT Gateway

> Launch instance in public subnet with Amazon Linux 2 and tag it as Bastion

> Allow only ssh from DL-Infra { Network } to Bastion i.e in Security Group of Bastion only SSH from DL N/W i.e search for my ip in google

> Launch instance in public subnet using Amazon Linux 2 tag it as Web Server

> Allow only ssh from Bastion i.e private ip of bastion

> Install httpd on Web Server

> Web Server works on port 80, Allow port 80 from anywhere

> Launch instance in private subnet using Amazon Linux 2 and tag it as DB Server

> Allow only ssh from Bastion i.e private ip of bastion

> Download MYSQL RPM { executable }

> wget<http://repo.mysql.com/mysql-community-release-el7-5.noarch.rpm>

> Gets failed, create NAT gateway in public subnet and attach the routing in PVT Route table

> Steps to create NAT Gateway: VPC Dashboard > NAT Gateways > Create NAT Gateway > Select the Public Subnet > Elastic IP allocation: create new EIP > Create NAT Gateway

Once NAT Gateway is created, attach the routing in the PVT RTB i.e

0.0.0.0/0 -> NAT-GW-ID

<https://docs.aws.amazon.com/vpc/latest/userguide/VPC_NAT_Instance.html#NATInstance>

RDS

* A Database is a store for datasets where :
  + Data access (reads and writes) is needed on a recurring basis
  + It allows multiple user access for reads & writes

**Relational DB**

* A Relational Database (concept) is a data structure that allows you to link information from **different tables**.
* It normalizes data into structures (rows & columns)
* A schema is used to strictly define tables and relations between tables.
* Structured data, same items in tables are stored in the same table locations and can save data in multiple joined tables.
* All Relational Databases use Structured Query Language (SQL).
* Best Suited for OLTP (On line Transaction Processing) (ATM is example).
* Examples - Oracle, MySQL, DB2 etc

**Non Relational DB**

* A Non-Relational Database store data without a structured mechanism means it's a one big giant table.
* Non-Relational Databases are non-schema based unlike Relational Database.
* Use non-structured data.
* Storage and retrieval of data is modeled without tabular relations as in SQL Databases.
* Non-Relational or No-SQL databases use a variety of data models including documents (JSON/XML), graph based, key-value etc.
* Non-Relational databases meets today's needs in social media, Analytics, big data and IOT.

**RDS**

* Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate and scale a relational database in the cloud.
* It provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching and backups.
* It frees you to focus on your applications so you can give them the fast performance, high availability, security and compatibility they need.
* Amazon RDS is available on several database instance types - optimized for memory, performance or I/O
* RDS provides you with six familiar database engines to choose from
  + Amazon Aurora
  + PostgreSQL
  + MySQL
  + MariaDB
  + Oracle
  + Microsoft SQL Server
* One imp thing you need to know about Database section is, we are not going to see, how to use these various DB's, knowing exactly how to use a SQL or NoSQL DB is not the job of Solutions Architect.
* Your job is to understand what are the various offerings, so when a customer or your organization comes with a requirement for DB's you will be able to select the right type of the DB, for your requirement of the application as well as security, cost benefits.

**RDS Essentials**

* RDS is a fully managed Relational Database Service.
* Does not allow access to the underlying operating system (fully-managed).
* You connect to the RDS database server in the same way you would connect to a traditional on-premise database instance (i.e MySQL command line).
* RDS has the ability to provision/resize hardware on demand for scaling.
* Every DB Instance has a weekly maintenance window
* You can enable Multi-AZ deployments for backup and high availability.
* Utilize Read Replicas (MySQL/postgreSQL/Aurora) - to help offload hits on your primary database.
* Relational database are the databases that organize stored data into tables.
* The associated tables have defined relationships between them.

**RDS Benefits**

* Benefits of running RDS instead of a database on your own instance:
* Automatic updates.
* Automatic backups
* Not required to manage operating system
* Multi-AZ
* Automatic recovery in event of a failover.

**RDS Multi AZ Failover**

* From above architecture diagram, we have primary DB instance, when we enable multi AZ failover, which you should do for any kind of production environment, So what will happen is at anytime you write data to primary instance, it is going to synchronously copy that over to a standby instance, in another AZ.
* So as we know with EC2 instances application architecture, we always want to have multiple EC2 instances running our application in multiple AZ's as this provides high availability and fault tolerant, so it's the same concept here, this is how we create high availability and fault tolerance within our DB architecture.

**RDS Backups**

* AWS provides automated point-in-time backups against the RDS database instance.
* Automated backups are deleted once the database instance is deleted and cannot be recovered (but you can take your own snapshots of backups before deleting).

**RDS READ REPLICAS**

* Read replicas are a synchronous copies of the primary database that are used for read only purposes (only allow "read connections").
* When you write new data to the primary database, AWS copies it for you to the read replica.
* You can create, and have multiple read replicas for a primary database.
* Read replicas can be created from other replicas (so no performance hit on the primary database).
* MySQL, MariaDB, PostgreSQL and Aurora currently support read replicas.
* Read Replicas allow for all read traffic to be redirected from the primary database to the read replica. This will greatly improve performance on the primary database.
* Read replicas allow for elasticity in RDS - you can add more read replicas as demand increases.
* You can promote a read replica to a primary instance.

**Application Server Setup**

> Launch an instance in Public Subnet with Amazon Linux 2 OS tag as App Server

> Like we installed Apache Web Server to deploy the website, we need to install **Apache Tomcat** to go with dynamic applications

> Tomcat requires Java to function

> Install Java

> java

> sudo yum -y install java-1.8.0 java-1.8.0-devel

> Download Tomcat Binary

> wget<https://archive.apache.org/dist/tomcat/tomcat-7/v7.0.94/bin/apache-tomcat-7.0.94.tar.gz>

> Extract Tomcat

> tar xvf apache-tomcat-7.0.94.tar.gz

> cd apache-tomcat-7.0.94

> Starting the server

> Tomcat runs on port 8080

> sudo netstat -ntpl | grep 8080

> cd bin

> ./startup.sh { Hit enter }

> sudo netstat -ntpl | grep 8080

> Browse the tomcat server by public-ip:8080, you will be able to see the Tomcat page

> Also apply a custom tcp rule with port 8080 from anywhere in **security group**, as tomcat works on port **8080** by default

> Go back to app server where tomcat is installed and perform below tasks

-> Install Git

> sudo yum -y install git

> git --version { confirm }

-> Install Maven

> sudo yum -y install maven

> mvn --version { confirm }

-> Fetch Application Code

> cd /home/ec2-user

> git clone -b aws https://github.com/Akiranred/aws-rds-java.git

> cd aws-rds-java

> vim src/main/webapp/login.jsp

Change the line no 6 that says

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jwt",

"Akiranred", "Admin123\*");

Connection con = DriverManager.getConnection("jdbc:mysql://**db-server-private-ip**:3306/jwt",

"Akiranred", "Admin123\*");

> vim src/main/webapp/userRegistration.jsp

Change the line no 9 that says

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jwt",

"Akiranred", "Admin123\*");

Connection con = DriverManager.getConnection("jdbc:mysql://**db-server-private-ip**:3306/jwt",

"Akiranred", "Admin123\*");

> mvn package

> cp /home/ec2-user/aws-rds-java/target/LoginWebApp.war /home/ec2-user/apache-tomcat-7.0.94/webapps

> Browse : http://App-Server-Public-IP:8080/LoginWebApp

Now you can Register a user and verify the same by logging in

RDS LAB - PAAS

-> Services -> RDS -> List Left side menu -> CLick Subnet Groups -> Create DB Subnet Group -> Choose the VPC -> Select the Subnets -> Create

-> Databases -> Create Database -> Select MySQL -> Scroll down check Only > enable options eligible for RDS Free Usage Tier -> Give some name : DB instance identifier > username : Akiranred > password : Admin123\* > Select VPC -> select subnet group created > Public accessibility : no > uncheck Enable deletion protection at end -> Create Database

* In the Create database section, choose Create database.
* You now have options to select your engine. For this tutorial, click the MySQL icon, select the value of edition and engine version as any 5.6.X, and select the Free Tier template.
* You will now configure your DB instance. The list below shows the example settings you can use for this tutorial:

Settings:

* DB instance identifier: Type a name for the DB instance that is unique for your account in the Region that you selected. For this setup, we will name it **lamp**.
* Master username: Type a username that you will use to log in to your DB instance. We will use username as **root** for this setup
* Master password: Type a password that contains from 8 to 41 printable ASCII characters (excluding /,", and @) for your master user password.
* Confirm password: Retype your password
* DB instance class: Select db.t2.micro --- 1vCPU, 1 GIB RAM. This equates to 1 GB memory and 1 vCPU.
* Storage type: Select General Purpose (SSD).
* Allocated storage: Select the default of 20 to allocate 20 GB of storage for your database. You can scale up to a maximum of 64 TB with Amazon RDS for MySQL.
* Enable storage autoscaling: If your workload is cyclical or unpredictable, you would enable storage autoscaling to enable RDS to automatically scale up your storage when needed. This option does not apply to this tutorial.
* Multi-AZ deployment: Note that you will have to pay for Multi-AZ deployment. Using a Multi-AZ deployment will automatically provision and maintain a synchronous standby replica in a different Availability Zone.
* VPC security groups: Select Create new VPC security group. This will create a security group that will allow connection from the IP address of the device(web server) that you are currently using to the database created.
* Keep everything else default
* Click Create Database.
* It could take several minutes for the new DB instance to become available.
* The new DB instance appears in the list of DB instances on the RDS console.
* The DB instance will have a status of creating until the DB instance is created and ready for use. When the state changes to available, you can connect to a database on the DB instance.
* Once the DB instance becomes available, copy the endpoint of RDS and connect it in code.

-> endpoint { DNS }

-> Replace the DNS in following files

> vim src/main/webapp/login.jsp

Change the line no 6 that says

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jwt",

"Akiranred", "Admin123\*");

TOO

Connection con = DriverManager.getConnection("jdbc:mysql://db-server-pvt-dns:3306/jwt",

"Akiranred", "Admin123\*");

> vim src/main/webapp/userRegistration.jsp

Change the line no 9 that says

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/jwt",

"Akiranred", "Admin123\*");

TOO

Connection con = DriverManager.getConnection("jdbc:mysql://db-server-pvt-dns:3306/jwt",

"Akiranred", "Admin123\*");

-> on App server

mysql -h endpoint-rds -u Akiranred -p

create database jwt;

use jwt;

CREATE TABLE `USER` (

`id` int(10) unsigned NOT NULL auto\_increment,

`first\_name` varchar(45) NOT NULL,

`last\_name` varchar(45) NOT NULL,

`email` varchar(45) NOT NULL,

`username` varchar(45) NOT NULL,

`password` varchar(45) NOT NULL,

`regdate` date NOT NULL,

PRIMARY KEY (`id`)

) ENGINE = InnoDB DEFAULT CHARSET = latin1;

> mvn package

> cp /home/ec2-user/aws-rds-java/target/LoginWebApp.war /home/ec2-user/apache-tomcat-7.0.105/webapps

> Browse : [http://App-Server-Public-IP:8080/LoginWebApp](http://app-server-public-ip:8080/LoginWebApp)