INDEX

* [Intro 2](#_Toc163677134)
* [Linux 4](#_Toc163677135)
* [GIT & GitHub 5](#_Toc163677136)
* [EC2 7](#_Toc163677137)
* [Elastic IP 9](#_Toc163677138)
* [Web-Servers 10](#_Toc163677139)
* [User Data 11](#_Toc163677140)
* [S3-Bucket 11](#_Toc163677141)
* [VPC 14](#_Toc163677142)
* [Auto-Scaling 17](#_Toc163677143)
* [Elastic Load Balancing 18](#_Toc163677144)
* [Route53 18](#_Toc163677145)
* [JENKINS 20](#_Toc163677146)
* [DOCKER 22](#_Toc163677147)

C:\Users\Admin\Downloads\icons8-aws-logo-48.pngAWS DevOps

# Intro

* **AWS:** **A**mazon **W**eb **S**ervices is a comprehensive and widely used cloud computing platform provided by Amazon.com. It offers a vast array of services including computing power, storage options, networking, databases, machine learning, analytics, security and more. AWS enables businesses and individuals to access and utilize computing resources without the need to invest and maintain the physical infrastructure.
* **SDLC:** Software Development Life Cycle is a structured process used by software developers to design, develop, test, and deploy software applications. The main objective of SDLC is to produce high-quality software.
  + Requirements Gathering: Client needs and requirements are collected and documented.
  + Analysis: Analyses the gathered requirements and need’s.
  + Design: The software architecture and design are created based on requirements and analysis. It involves the system architecture,
  + Data structures, interfaces and other technical specifications.
  + Implementation (Coding): In this phase, Developers write code according to the design specifications.
  + Testing: Once the code is developed, it undergoes various testing phases to identify and fix defects and bugs.
  + Deployment: After successful testing, the software is deployed to production environment. This Involves installing the software on user’s system or servers and making it available for users **(Live).**
  + Maintenance and Support: Even after Deployment, the software requires outgoing maintenance and support to address issues, implement updates, and to add new features.
* Additionally, different methodologies such as Agile, Waterfall or DevOps may be employed to manage the SDLC Process.
* **DevOps:** Addressing all the traditional issues and implementing automation at every stage of software development life cycle by using

Automation tools.

It is a set of practices that combines software Development and IT Operations. It amis to shorten the systems development life cycle and provides continuous delivery with high software quality. Simply DevOps aims to improve the quality, speed and reliability of software delivery. It also involves the use of various tools and technologies to support the DevOps culture and processes.

* + ***Some of the commom DevOps Tools are***:
* **JENKINS**
* **DOCKER**
* **SONAR CLOUD**
* **ANSIBLE**
* **KUBERNATES**
* **TERRAFORM**
* **GIT etc……**
* **AWS DevOps:** The DevOps within the AWS cloud environment (Platform).
* **AWS Management Console** is a web based graphical user interface that allows you to manage and monitor your AWS resources. It provides a single interface to access and manage over 150 AWS services, including Amazon S#, Amazon EC2 and Amazon RDS. AWS console is accessible from any web browser and user friendly i.e, you can simply search for any services that you want to use. And for any help or information regarding AWS you can ask the AWS Assistant Bot (Amazon Q) which was an always-on generative AI assistant.
* **IAM:** **I**dentity and **A**ccess **M**anagement is a web service that helps you securely control access to AWS resources. It is created by the root user. With IAM the root user can centrally manage permissions that control, which AWS resources user can access. Simly IAM is used to control who is authenticated(signed in) and authorized(has permissions) to use resources.
  + **IAM role**: An IAM role is an IAM identity which manages who has access to your AWS resources. An IAM role is similar to an IAM user.
  + **IAM Policy**: is a document with a set of rules that defines permissions for an identity or resource in AWS.

**NOTE**: An IAM role with no IAM policy attached to it won’t have to access any AWS resources and an IAM policy that is not attached to an IAM role is of no use.

* **How to create an IAM user with the IAM roles and policies?**

Sign-in to AWS console and search IAM in the search bar on the console home. After that, go to IAM and select user and assign a user name and then you have to give IAM role and policies (set permissions) as per your requirement and review the created user and press create once you are done with that. You can delete or change the given permissions any time with the root credentials.

* **EC2:** Amazon **E**lastic **C**ompute **C**loud is a web service that allows users to rent virtual servers on which to run their applications without investing in physical hardware. Simply those virtual servers are called **“INSTANCES”.**
* **Key Features of EC2:**
* **Instances:** is a virtual server. It is a core part in the AWS’s cloud Computing platform and provides user with scalable computing capacity. It can be customized based on the various parameters such as memory, computing power, networking capacity.
* **Amazon Machine Images**
* **EBS (Elastic Block Storage)**
* **Network & Security**
* **Load Balancing**
* **Auto Scaling**
* **Server**: which provides functionality to the device or a project.

Working process: CURD

* Create
* Read
* Update
* Delete
* **EC2 Instance creation**:
* **Types of Instances**:
* **Pricing**: AWS offers several pricing models to accommodate different usage patterns and business needs.
* **ON-Demand Pricing**: pay as you go. Suitable for applications with variable workloads and short-term projects.
* **Spot Request**: Allow user to bid the unused ec2 instance capacity at a potentially lower cost (like an offer)
* **Reserved Instances**: It offers cost saving up to 75% as compared to on-demand instances
* **Full Afferent**: One time payment
* **Partial Afferent**: half half payment
* **No Afferent**: Use first and Pay later

# Linux

* **Linux** is an open-source operating system kernel that serves as the foundation for various Unix-like operating systems, commonly referred to as Linux distributions (or distros). Linux is highly customizable and is widely used in server environments, embedded systems, and personal computers.
* **Why DevOps professionals often prefer Linux?**

Compatibility with Tools (Many DevOps tools and platforms are developed primarily for Linux environments.)

Scripting and Automation (Linux provides powerful command-line interfaces and scripting capabilities through shells like Bash. DevOps tasks,

such as provisioning, configuration management, deployment automation, and system monitoring, can be efficiently

performed using shell scripts and command-line tools available in Linux)

Containerization and Orchestration (Linux offers scalability and performance advantages, allowing DevOps teams to efficiently manage large-scale

infrastructure and handle high workloads. Linux-based servers can be optimized for performance, resource

utilization, and reliability, meeting the demands of modern applications and services.)

* **Linux Commands** are instructions or directives given to the Linux operating system through the command-line interface (CLI) or terminal. These commands are used to perform various tasks such as managing files and directories, manipulating processes, configuring system settings, networking etc…

**NOTE: Linux commands are case sensitive**

* + Some of the Linux Commands:
* **ls** : List directory contents.
* **ll** : List all the files including the extension files and hidden files.
* **ls -l** : List directory with details.
* cd : Change directory.

**cd </file/path/>** : To open the directory (folder)

* **pwd** : Print working directory.
* mkdir : Make directory

**mkdir <filename>**: To create a new folder or directory .

* rm : Remove

**rm <file name>** : Remove files or directories.

**rm –rf <file name>** : To force remove.

* cp , mv : Copy , move

**cp </source/path/> </dest/path/>** : Copy files or directories.

**mv </source/path/> </dest/path/>** : Move or rename files or directories.

* **touch <file name>** : To create empty text file (.txt).
* **cat <file name>**  : To view the data in the text files.
* **df -h**  : To know the hard disk storage.
* **du -sh** : To know the folder storage.
* **top**  : To know the CPU utilization.
* **cd ..** : To come one step back.
* **clear** : To clear the screen. (work history obsent)
* **ctrl+l** : Is a shortcut to clear the screen. (work history present)
* sudo : Execute commands with superuser privileges.
* **sudo -s**  : To become a superuser.
* **exit** : To come back or to log out.
* apt : Package manager It is used to install, remove, and manage software packages.

**apt install** :To install the packages or softwares

**apt update** : To update the server. (refresh)

**apt upgrade** : To upgrade the packages in the server.

* **kill** : To terminate process.
* **ps** : Process status.
* **whoami** : To display the current username.
* chmod , chown : Change mode, change owner

**Mode numbers**

r — Read — 4 for only read — 400 (owner access) 444 (everyone)

w — Write — 2 for r and w — 600 (owner access) 666 (everyone)

d/x — Execute — 1 foe r, w , d/x — 700 (owner access) 777 (everyone)

**chmod <mode\_num> <file­\_name>** : To change the mode of the file.

**chown <option> <owner\_name> <file\_name>** : To change the owner of the file.

* systemctl : System control.

**systemctl start <service\_name>** : To start a service

**systemctl stop <service\_name>** : To stop a service

**systemctl restart <service\_name>** : To restart a service

**systemctl enable <service\_name>** : To enable a service

**systemctl disable <service\_name>** : To disable a service

**systemctl status <service\_name>** : To check the status

* **lsblk** : List information about all available block devices.
* find : To find the file.

**find / -name <file­\_name>** : To find the file.

* wget : To download from net

**wget <url>** : To download the data or a packages from the net by pasting the url.

* tar : To create and manipulate tar archives.

**tar –zxvf <tar\_file>** : To untar the tar file.

* zip/unzip : To create and extract the files.

**zip –r <filename.zip> <file names to zip>** : Command to zip the files.

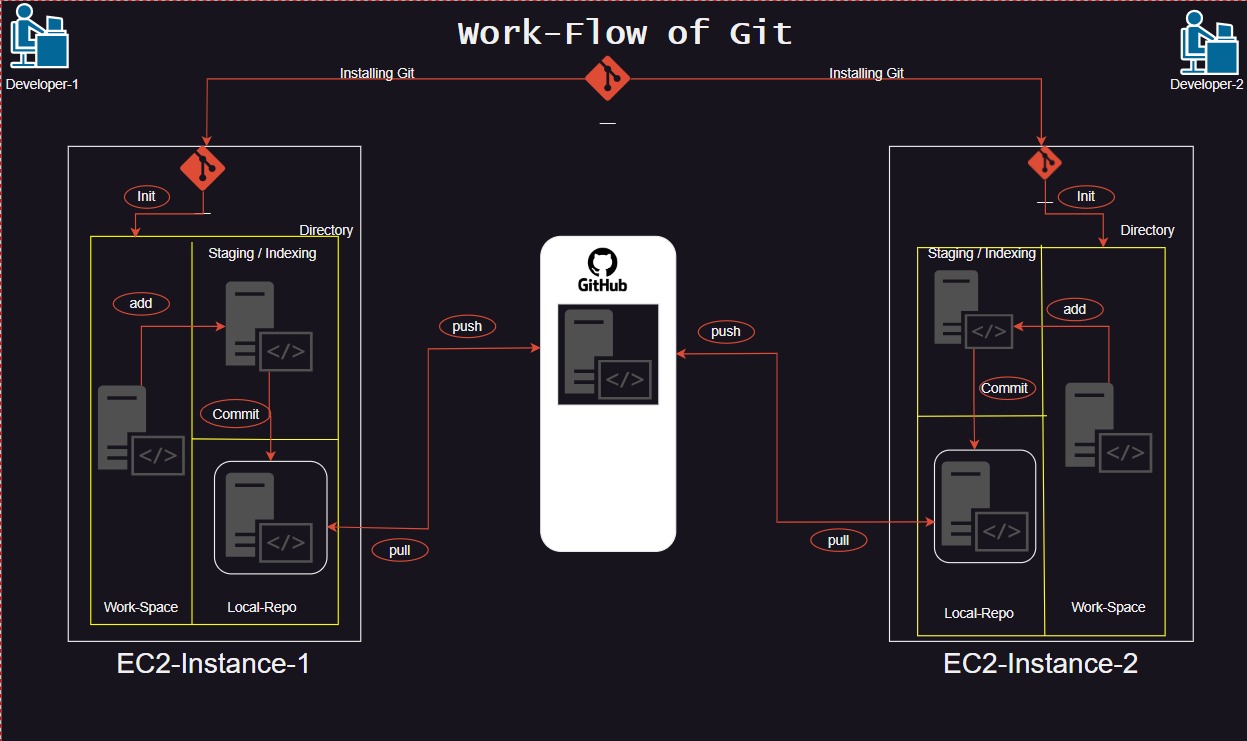
**unzip <.zip\_file­\_name>** : Command used to extract the zip file.

**>filename** is to clear the complete date inside the extension file.

# GIT & GitHub

* **Git** is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
* It's an open-source tool that tracks changes in source code during software development.
* With Git, developers can work collaboratively on projects, track changes, revert to previous versions, and merge changes made by different team members.
* **GitHub** is a web-based platform for version control using Git.
* It offers hosting for software development projects, enabling collaboration between developers, code management, and various project management features

**GIT WORK FLOW:**



**First time:**

* git init
* git add .
* git commit -m "comment"
* git branch -M (branch name)
* git remote add origin (github repo link)
* git push -u origin (branch name)

**Changes:**

* git status
* git add .
* git commit –m “comment”
* git push

**git log** is a command used to display the commit history of a repository.

**SYNTAX**:

* git log
* git log --oneline

It displays

* Commit hash
* Author's name and email
* Date and time of the commit
* Commit message
* **This is the difference between the git log and git log –oneline**



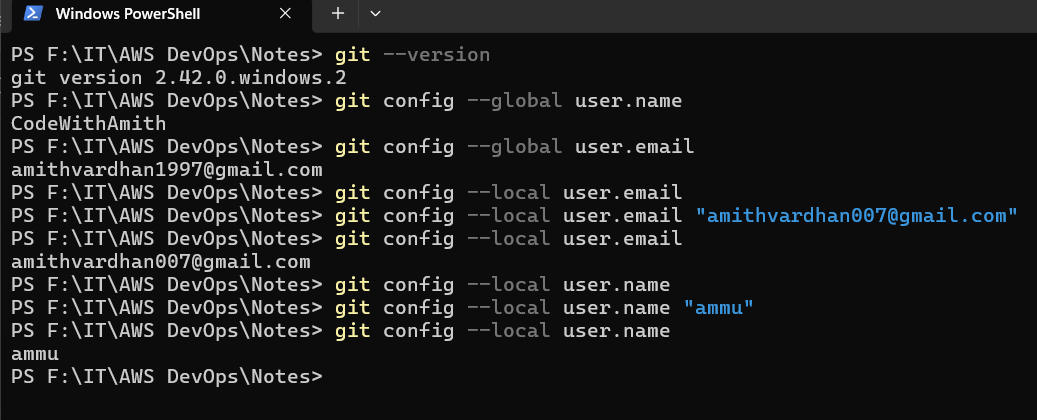
* **git pull** is a Git command used to fetch the latest changes from a remote repository and integrate them into the current branch of your local repository. Essentially, it performs two actions: **git fetch** followed by **git merge**
* **git clone** command is used to create a copy of an existing Git repository. It fetches the entire repository, including all branches, commits, and files, and sets up a new local repository on your machine. This is particularly useful when you want to start working on a project that already exists in a remote Git repository.

**SYNTAX:**

For local repo : git clone <repository\_url>

For instance : git clone <repository url> <file name that u want to save this clone the repo>

* **git fetch** command is used to retrieve new commits from a remote repository without merging them into your local branches. It's essentially a way to update your local repository with the latest changes from the remote repository without automatically merging them into your working branch.
* **git merge is** a command used to merge changes from one branch into another. It combines the changes made in one branch with another branch, resulting in a new commit that reflects the combined history of both branches**.**
* **git --version** is to check the version of the git.
* **which git** is the command to know the git file path.
* **git config --global user.name “give­\_user\_name”** to give the user name in global.
* **git config --local user.name “give­\_user\_name”** to give the user name in local.
* **git config --global user.name** to know the user name in global.
* **git config --local user.name** to know the user name in local.
* **git config --global user.email “**[**give@mail.id**](mailto:give@mail.id)**”** to give the mail id in global.
* **git config --local user.email “**[**give@mail.id**](mailto:give@mail.id)**”** to give the mail id in local.
* **git config --global user.email** to know the mail id in global.
* **git config --local user.email** to know the mail id in local.



# EC2

* **EC2** (**E**lastic **C**ompute **C**loud) is a web service that allows users to rent virtual servers on which to run their applications without investing in physical hardware. Simply those virtual servers are called **“INSTANCES”.**

**EBS- Elastic Block Services:**

**Volumes**:

* Volumes are like pen­-drives, we use pen drive to transfer the data, for extending the storage. In the same way the volumes are used to transfer the data from one instance to another instance and to extend the data storage of the instance if required.

**Snapshots:**

* A snapshot, specifically captures the state of an EBS volume at a particular point in time.
* Snapshots are typically used for backup, data recovery, and disaster recovery purposes.

Devops engineer have to take backup every end of the day. And have to store minimum 1 moth Backups.

**AMI- Amazon Machine Image:**

* An AMI is a pre-configured template that contains the software configuration required to launch an EC2 instance (a virtual server) in AWS.
* It includes not only the data stored on an EBS volume but also the operating system, application server, applications, and other settings required to launch an instance.
* AMIs are used to launch new EC2 instances, providing a starting point for the instance's configuration and setup.

**Difference between the AMI and Snapshots:**

* Both snapshots and AMIs involve capturing data at a specific point in time, snapshots focus on backing up the data stored on EBS volumes, while AMIs encompass a broader set of configurations necessary to launch EC2 instances in AWS. Simply AMI is for server backup from **EC2** and Snapshots are for data backup from the **EBS**.

**Practical:**

**How to Create Volumes:**

EC2 🡪 EBS 🡪 Volumes 🡪 Create Volume 🡪 Choose Volume type 🡪 Size 🡪 Create Volume 🡪 give name

**How to attach EBS Volumes:**

Choose/Create Volume 🡪 Actions 🡪 Attach Instance 🡪 Choose instance 🡪 Proceed

**Next:**

Connect to the server that we have attached the EBS volume

* Sudo –s
* apt update
* apt upgrade –y
* lsblk (command to provides a hierarchical view of block devices and their relationships, including information such as device name, major and minor device

numbers, size, type, mount points, and more.)

* mkfs.ext4 /dev/xvdf (mkfs-make file system To insert the attached volume)
* lsblk (To check whether the volume is inserted or not. You can see the xvdf file at the end without any path)
* mkdir <folder\_name> (We have to create one directory to mount the volume in it)
* cd <folder­\_name>
* pwd (copy the printed work directory path)
* cd ..
* mount /dev/xvdf <paste the coppied path> (To mount the attached volume into that directory)
* lsblk (you will find the path now)

**complete the task given by the developer. After the task is completed we have to un-mount the volume**

* umount /dev/xvdf (command to un-mount)
* lsblk (to check whether the volume was un-mounted or not)
* rm –rf <folder\_name> (Remove the folder once after the work is done)

**Now detach the volume:**

Select volume 🡪 Actions 🡪 Detach the EBS-Volume

If we want to send the work files of one person to another then, we have to attach the volume to that person’s server and we have to copy the files from that server to this volume. After that we have to un-mount and detach to that server and next we have to attach and mount the same volume to another person’s server and paste these copied files inside it. This is how the file transfer works.

**How to create Snapshots:**

Select Volumes 🡪 Actions 🡪 Create Snapshot 🡪 Create 🡪 Give Name

**How to create AMI’s:**

Select instance Actions Image and Templates Create Image Image Name Create Image

**Creating a new instance using the image:**

Launch Instance 🡪 Instance name 🡪 Select the AMI that we have created before 🡪 Launch instance

We can snapshot the copied data inside the volume as a volume backup and that backup will be saved in the s3 bucket.

**How to Create a snapshot?**

Select Volume 🡪 Actions 🡪 Create Snapshot 🡪 Description(opt) 🡪 Create

**How to delete a snapshot?**

Select snapshot 🡪 Actions 🡪 Delete snapshot

**How to create volume using snapshot?**

Select snapshot 🡪 Actions 🡪 Create Volume

**How to create image using snapshot?**

Select snapshot 🡪 Actions 🡪 Create Image

# Elastic IP

* **Elastic IP server capacity**: In any web-application or a streaming application, if the users or the work load increases then we have to increase the EC2 instance capacity to reduce the traffic (to decrease the work load). So that more users can access the application.
* Before doing that we have to inform the team members about this. And take the approval from the networking team, application team and manager.

**Procedure**:

1. Select the EC2 instance in which the application was deployed.
2. **Stop** the server in instance state options.
3. After the instance was stopped, go to **actions** and select the **instance settings** now select the **instance type**.
4. Now, change the instance type as per requirement and **apply changes**.
5. Start the instance now and check weather the application in the server is running stage or not.
6. Check the following commands: **df –h**

**top**

**systemctl status <application­\_name>**

**systemctl restart <application\_name>**

* Every time after stopping and stating the instance the public ip of the instance will be changed. It will be a problem for all the users who are having old ip address.
* **Elastic IP server** is the solution for that problem.
* **Elastic IP** is a service in the EC2 which provides the fixed ip address to the instance.

it is a paid service. It will charge even the instance is not in running stage.

Therefore, it's advisable to use them wisely and release them when not in use, to avoid unnecessary costs.

Each AWS account contains only 5 elastic ip’s.

If we need more me have to request the aws official team.

* How to add the elastic ip:

1. Select the **EC2 instance**: **Network** **and Security**: **Elastic IP’s** : **Allocate the elastic IP**: Choose the **region**: **allocate**.
2. After allocation was done. Select the **elastic ip** and click on **actions** choose the **associate Elastic IP address**: choose **instance**: select the **server**: select the **private ip** of that server: **allocate**.

* How to remove the elastic ip:

1. Select the **instance**: **actions**: **network**: **Disassociate elastic ip address.**
2. After disassociate the elastic ip we have to release the Elastic ip.
3. Select **elastic ip**: **actions**: **release the elastic ip address**.

# Web-Servers

* **web server** is a software application or hardware device responsible for serving web content to clients over the internet.
* It plays a critical role in enabling access to web resources and facilitating communication between clients and servers on the World Wide Web.

Some of the popular web-server software applications:

* Nginx
* Apache
* Node.js
* Caddy
* LiteSpeed
* Microsoft IIS (Internet Information Services) etc..
* **Nginx** is an open source web server known for its high performance, scalability, and efficiency.

syntax to install nginx: **apt install nginx –y**

port number for nginx: **80**   
Path of nginx: **/var/www/html/**

You will find the **web content** (**.html**) file inside this path.

* **Apache** is an open-source web server software developed and maintained by the Apache Software Foundation. It is one of the most widely used web server applications globally.

Syntax to install apache: **apt install apache2 –y**

Port number for apache2: **80**

Path of apache2: **/var/www/html/**

You will find the **web content** (**.html**) file inside this path.

* Deploying a webpage using these web-server softwares in EC2 instance:

Steps-

1. Launch one EC2 instance.
2. Connect to the server and become a root user **sudo –s**  and update the server **apt update .**
3. Now, install the web-server software applications either Apache (**apt install apache2 –y**)

or nginx (**apt install nginx –y**)

1. Go to the web content path **cd /var/www/html**
2. We will find index.html file there.
3. Either we can edit that index.html file with our webpage code given by developers using **vim** or **nano** or we can clone the repository from GitHub (**git clone <repository\_url>**) or we can download the web files using url (**wget <webfile\_url>**).
4. If we download the webpage from the web using wget it will be downloaded as a .zip file. So we have to unzip that file to deploy. For that we need to install the unzip (**apt install unzip –y**). after the installation is done we have to unzip that .zip file by **unzip <.zip\_file­­\_name> .**
5. After the .zip file was extracted we have to remove the .zip file (**rm <.zip\_file\_name>**) just for storage purpose.
6. Now, the extracted files are stored in a new folder in html folder. Open that new folder (**cd <folder­\_name>**) So we have to move all the files in this new folder to the html folder i.e, one step back **mv \* ..**
7. Now, we can remove that empty new folder since we have moved all the files in that folder there is no use of that folder anymore. **rm –rf <new­\_folder\_name>**
8. We have to do the same process except the unzip when we clone from the GitHub.
9. Now, to access this webpage in the internet we have to add the port number 80 (both the nginx and the apache port number is 80).
10. To add the port number: **select the instance** and go to **security** and then open **security groups**

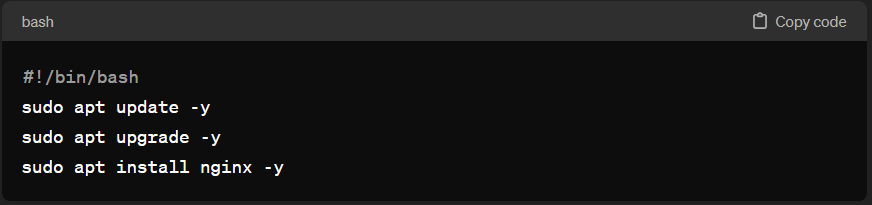
Choose the **edit inbound** **rules** and **add port 80** in it.

1. Now, copy the public id and past it in the browser with :80 at the end (**public\_ip:80**).
2. Hence the web-page was deployed.

# User Data

* **User Data** is a plain text format. It is a feature that allows you to pass initialization scripts or user data to an EC2 instance when it's launched. This user data can be used to automate various tasks such as installing software, configuring the operating system, setting up services, and more, making the instance ready for its intended purpose right after launch.
* When you launch an EC2 instance, you can provide user data in the form of a script .The EC2 instance retrieves this user data and executes it during the boot process. This enables you to customize the configuration of your EC2 instances without manual intervention.

**Syntax:**

****

**We can write the user data in two ways.**

1. While launching the instance we can find the user data in the advance option. We can write the user data there.
2. After launching the instance we can create a text file inserting the user data in it. The text file must be named with .sh at the end. After the file was written we have to give permissions to it by chmod +x <filename.sh>. After that we have to run the file by ./filename.sh this command will run the user data that we have written in that file.

# S3-Bucket

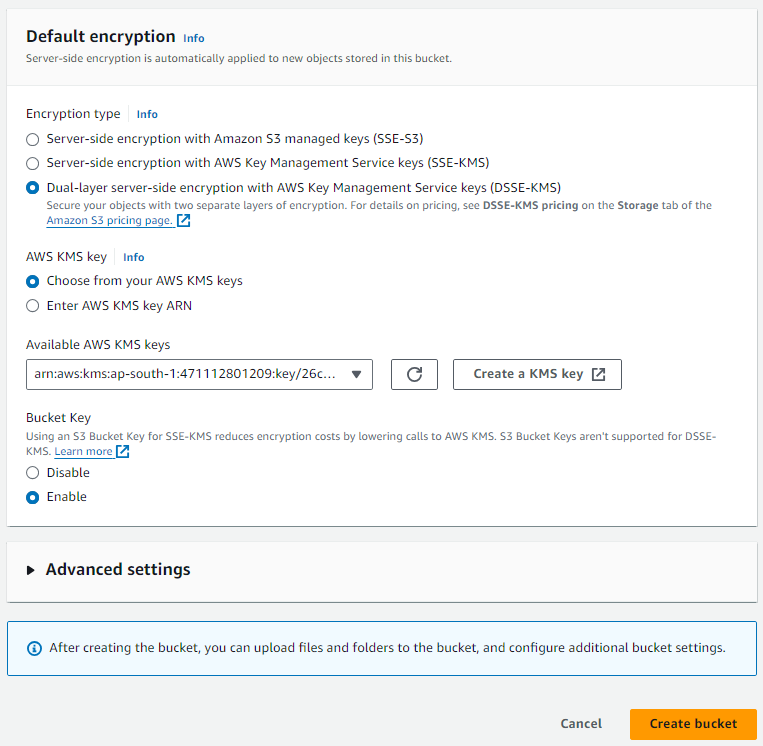
* **S3 (Simple Storage Service) bucket** is a container for storing data objects in the Amazon Web Services (AWS) cloud. It's a fundamental resource provided by AWS for storing and retrieving any amount of data, at any time, from anywhere on the web.
* By default S3 is a global service.
* AWS imposes a soft limit of 100 S3 buckets per region by default. If you require more than 100 S3 buckets, you can request a limit increase from AWS support, providing justification for your needs.
* Each object stored in Amazon S3 can be up to 5 terabytes in size.
* Bucket names must be globally unique across all existing bucket names in the AWS ecosystem. This means that you cannot use a bucket name that is already in use by another AWS account, regardless of region.

Practical:

**How to create the S3 Bucket**:

S3 bucket 🡪 create bucket 🡪 Bucket name 🡪 Region 🡪 ACLs (desible/enable) 🡪 ☑Block all access

🡪Bucket Versioning (enable/disable) 🡪 Default encryption ↴



At last Create Bucket

Choose this for extra extra protection

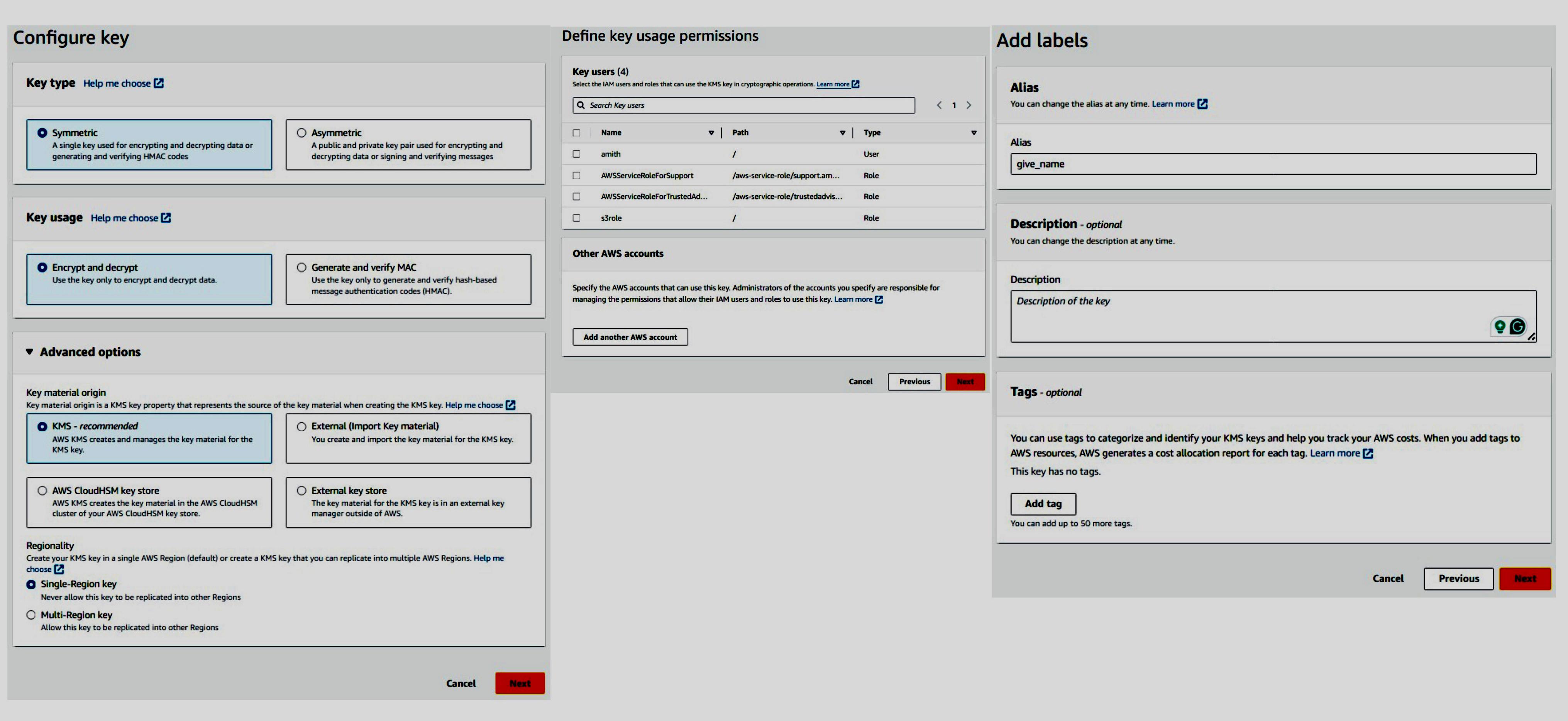
Create KMS key and choose that key here.

**KMS**-Key Management service is a managed service that make you easy to create and control the cryptographic (encrypts and decrypts the data) keys that are used to protect your data.

**How to create KMS key**:

Search KMS 🡪 Create key ↴ ↱ create.

Bucket is created



**EC2 to S3 bucket connection:**

* Create one fresh EC2 instance and connect to the server.
* sudo –s , apt update and apt upgrade –y (common commands to use after connecting to the server)
* We have to install the aws cli2 (command line interface) in this instance to make a connection between the EC2 and any AWS services like S3.
* But before installing the AWS cli2 we have to install unzip 🡪 apt install unzip -y. Because while installing the cli2 we need to unzip the web downloaded zip files.
* Now, after installing the unzip. Search “install AWS cli2 for ubuntu20.04” in browser. [Install or update to the latest version of the AWS CLI - AWS Command Line Interface (amazon.com)](https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html) (AWS official documentation) select linux and copy the commands.

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

and paste them in the instance.

* After successful installation of AWS cli2. We have to configure the AWS by the command aws configure.
* Now, it will ask for Access key and Private key which you have to create one in the security credentials from your aws account.

**How to create Access key and Private key:**

Click on the profile and select the Security credentials.

Create access key 🡪user case (CLI) 🡪next🡪create

Copy the Access and private key and store them safely

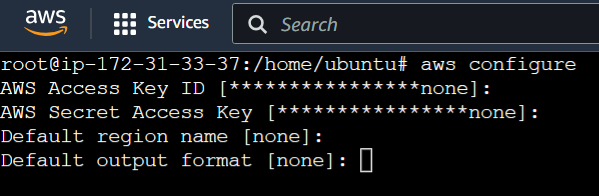
After aws configure command is used

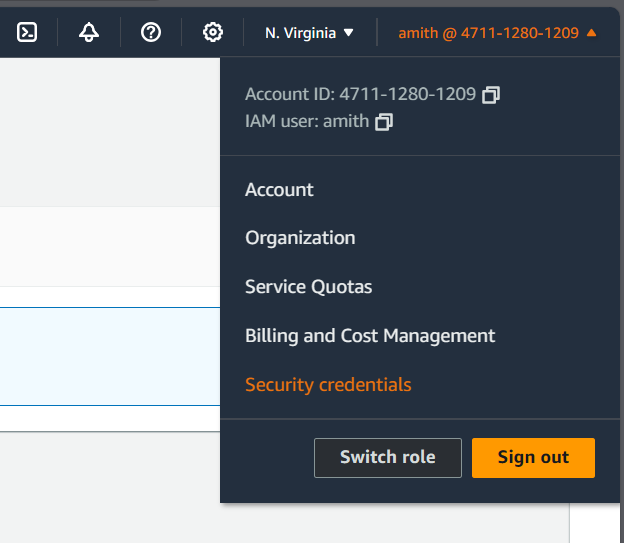
AWS Access Key: (paste access key here)

AWS secret key : (paste the secret key here)

Default region name: (write the region code here)

Default output format : json





Output

**AWS EC2 to S3 (Data copy/move):**

* We have to modify the iam role given to this instance. Create one iam role by giving the S3 full access.
* Now modify the iam role:

Select instance🡪Actions🡪Security🡪Modify IAM role🡪choose role🡪modify

* To check whether the instance was connected with s3 or not, use aws s3 ls command
* If we get the list of your s3 buckets as output then the connection is successful.
* We can add files to s3 using EC2:

mkdir <file\_name>

cd <file\_name>

vi <file\_name>

pwd 🡪 copy the path

**To copy file:**

**from EC2 to S3:**

aws s3 cp <paste/the/pwdpath/filename> s3://<s3\_bucket\_name>

**from S3 to EC2:**

aws s3 cp s3://<s3\_bucket\_name/file\_name> <destination/path>

**To move file:**

**from EC2 to S3:**

aws s3 mv <paste/the/pwdpath/filename> s3://<s3\_bucket\_name>

**from S3 and EC2:**

aws s3 mv s3://<s3\_bucket\_name/file\_name> <destination/path>

(Check the data inside the s3 bucket in AWS and EC2 instance for conformation)

**To create the S3 bucket from EC2:**

aws s3 mb s3://<new\_bucket\_name> mb- make bucket

this will create the bucket in the aws s3 directly.

**Another way of server connection:** ssh –i <\_\_\_\_.pemfile> username@public\_ip

# VPC

* **VPC** stands for Virtual Private Cloud. It's a virtual network dedicated to your AWS account. It enables you to launch AWS resources into a virtual network that you've defined.
* Simply it is a secure isolated private cloud hosted within a public cloud.
* This provides you with control over your virtual networking environment, including the selection of your IP address range, creation of subnets, and configuration of route tables and network gateways.
* In AWS DevOps, understanding how to configure and manage VPCs is essential for designing and deploying applications securely and efficiently within the AWS cloud.

**IP:** IP stands for Internet Protocol. It's a set of rules that govern how devices communicate over the internet or a network. An IP address is a unique identifier assigned to each device connected to a network, allowing them to send and receive data to and from each other. It's like a digital address for devices on the internet, enabling them to find and communicate with each other.

**Types of IP Addresses:** There are two types of IP Addresses. IPV4 and IPV6.

|  |  |
| --- | --- |
| IPV4  🡪 32-bit numerical Addresses. Typically written in a format like n.n.n.n  🡪 Total possible IP’s=2³²  🡪 This type of IP address has been the standard for decades and is still widely used today.  🡪 If number of users increases then IPV6 may be required if the IPV4 ran out. | IPV6  🡪 128-bit numerical addresses. Usually expressed in a hexadecimal format like n.n.n.n.n.n  🡪 Total possible IP’s=2¹²⁸  🡪 IPv6 was developed to address the limitations of IPv4, particularly the exhaustion of available addresses.  🡪 It provides a much larger address space, allowing for trillions of unique addresses, which can accommodate the growing number of devices connecting to the internet. |

IPv4 and IPv6 are both types of IP addresses used for communication between devices on the internet or a network. IPv4 is the older standard with 32-bit addresses, while IPv6 is the newer standard with 128-bit addresses, designed to overcome the limitations of IPv4 and support the growing number of connected devices.

* **Private IP:** A private IP address is used within a local network, such as a home or business network, and is not directly

accessible from the internet.

Eg: 10.x.x.x x ranges from 0 to 255

172.y.x.x where y ranges from 16 to 31

192.192.x.x rest all are public IP’s

* **Public IP**: A public IP address is assigned to a device that is directly accessible from the internet and are globally reachable.

It is used for communication between devices over the internet.

* **SUBNET** is short for subnetwork, is a logical subdivision of an IP network. It allows you to divide a single network into

smaller, more manageable parts. Subnetting is typically done to improve network performance, security, and organization.

Subnets are of two types: Public Subnet and Private Subnet

**Public Subnet:** are subnets within a VPC that have a route to the Internet Gateway (IGW).

**Private Subnet:** are subnets within a VPC that do not have a route to the Internet Gateway (IGW).

They are isolated from the internet.

The subnets will wary from region to region.

**Mumbai Region** ap-south-1  **US East (N. Virginia)** us-east-1

• ap-south-1a • us-east-1a • us-east-1d

• ap-south-1b • us-east-1b • us-east-1e

• ap-south-1c • us-east-1c • us-east-1f

No. of IP’s in a given subnet (n.n.n.n/x) can be calculated by 2³²-ˣ for IPV4 and 2¹²⁸-ˣ for IPV6.

Where n.n.n.n is the IP address and x is the subnet mask.

* **INTERNET GATEWAY** in AWS VPC facilitates internet connectivity for resources within the VPC, allowing them to

communicate with services on the internet and vice versa. It is a crucial component for building scalable and flexible architectures in AWS.

* **NAT GATEWAY** (Network Address Translation Gateway), is a managed AWS service that allows instances in private

subnets within an VPC to access the internet or other AWS services while

remaining private.

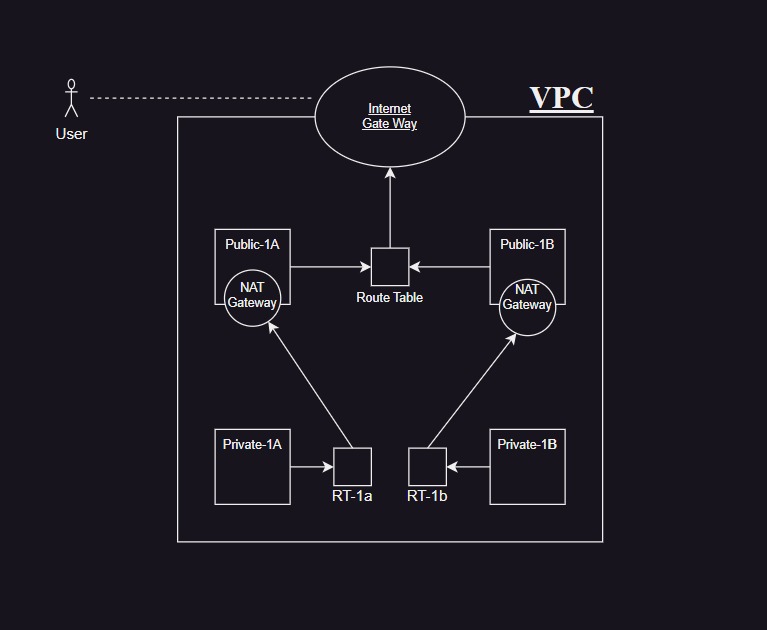
* **ROUTE TABLE** is a fundamental component of a VPC in AWS that controls the routing of network traffic within the VPC. It

determines how traffic is directed between different subnets, internet gateways, virtual private gateways,

NAT gateways, and other network devices.

* **HA Set-up (High Availability):**

**VPC Structure:**



* **Steps to create VPC:**
* **Step-1:** VPC creation

VPC 🡪 Create VPC 🡪 Name 🡪 IPV4 CIDR (10.1.0.0/16) 🡪 Create VPC

* **Step -2:** Create Internet Gateway

Internet Gateways 🡪 Create Internet Gateway 🡪 Name 🡪 Create

* **Step-3:** Connect the VPC to Internet Gateway

Select Internet Gateway 🡪 Actions 🡪 Attach VPC

* **Step-4:** Now, create 2 Public subnets

Subnets 🡪 Create Subnet 🡪 Select VPC 🡪 Name 🡪 Availability Zone 🡪 IPV4 CIDR block 🡪 Create **CIDR Examples**

10.1.1.0/24

10.1.2.0/24

* **Step-5:** Create one common route table for public subnets.

Route Table 🡪 Create Route Table 🡪 Name 🡪 Select VPC 🡪 Create

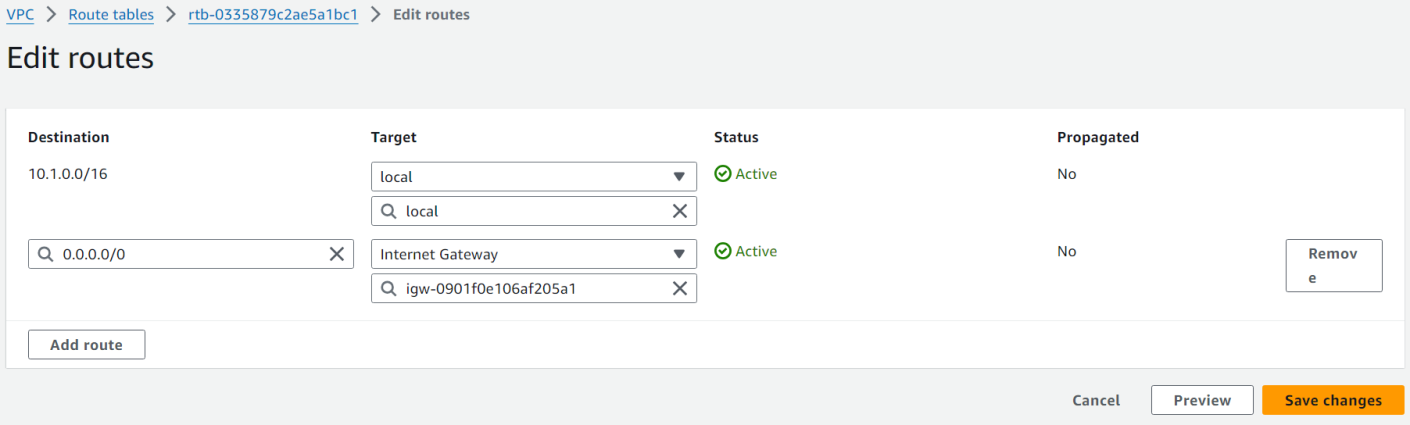
* **Step-6:** Now, we have to Associate the subnets to the route table.

Select Route table🡪 Actions 🡪 Edit subnet associations 🡪 Select public subnets 🡪 Save association

* **Step-7:** Now, we have to give routing to the Internet gateway.

Route table 🡪 Select route table 🡪 Edit routes 🡪 Add Routes 🡪 Select Internet gateway and select our Internet gateway that you have created 🡪 Save changes

**Check out the image for reference:**



* **Step-8:** Create 2 private subnets

Subnets 🡪 Create Subnet 🡪 Select VPC 🡪 Name 🡪 Availability Zone 🡪 IPV4 CIDR block 🡪 Create

**CIDR Examples**

10.1.101.0/24

10.1.102.0/24

* **Step-9:** Create two Route tables for private subnets

Route Table 🡪 Create Route Table 🡪 Name 🡪 Select VPC 🡪 Create

* **Step-10:** Create two NAT Gateways in the Public Subnets

NAT gateway 🡪 Create NAT gateway 🡪 Name 🡪 Select Subnet 🡪 Allocate Elastic IP 🡪 Create

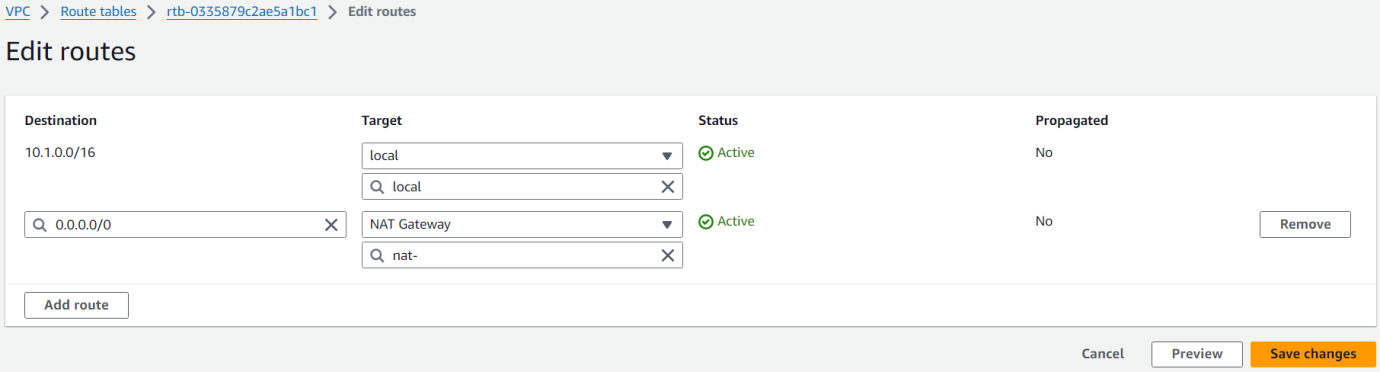
* **Step-11:** Now, Associate the private subnets to the private route tables which we have created in step-9.

Select Route table 🡪 Actions 🡪 Edit subnet associations 🡪 Select public subnets 🡪 Save association

* **Step-12:** And now we have to add NAT gateway to route tables that we have created in step-9.

Select route table 🡪 Actions 🡪 Edit routes 🡪 Add routes 🡪 Select NAT gateway and select the created NAT gateway 🡪 Create

**Check out the image for reference:**



# Auto-Scaling

* Auto-Scaling automatically adjust the number of resources such as server and instances, allocated to an application based on its work load or demand.
* Eg: while amazon sale the number of users will increase. So that the traffic of the application will be very high. At that time to decrease the traffic we have to increase the instances so that the traffic will become normal and the application can be accessed my many users. This will be done by Auto-Scaling.
* **Types:**
  + - **Vertical Scaling**
    - **Horizontal Scaling**
  + **Vertical Scaling** is also known as scaling up. It increases the size or capacity of the instance as per demand. It can

become expensive or impractical as resource requirements grow.

* + **Horizontal Scaling** is also known as Scaling out. It increases the number of instances (not the size or capacity of

the instance). Under the Load Balancer as per requirement.

Practical

* **Steps to create Auto-Scalling:**

Step-1: Create image of the EC2 instance.

Select EC2 instance 🡪 Actions 🡪 Image and Templates 🡪 Create Image 🡪 Name 🡪 Create

Step-2: Create Launch Template from AMI

Launch Template 🡪 Create Launch Template 🡪 Name 🡪 Choose My AMI’s 🡪 select image created in step-1 🡪Select Instance type and key pair 🡪 Select Subnet and common security group 🡪 create

Step-3: now, we have to create Auto scaling Group.

Auto scaling group 🡪 Create Auto Scaling Group 🡪 Name 🡪 Choose Launch Template (create in step-2)

🡪Next

🡪 Select VPC 🡪 select Zones 🡪 Next

🡪Health check period 🡪 Enable cloud watch 🡪 Next

🡪 select desired size 🡪 Min and Max desired capacity 🡪 Next 🡪 Next 🡪 Next 🡪 Create

After these 3 steps the Auto scaling group will be created. And we can see the changes in the EC2 instance. The new instances will be created automatically as per demand.

# Elastic Load Balancing

* **Elastic Load Balancing (ELB)** is a load balancing service provided by AWS that automatically distributes incoming application or network traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses within one or more Availability Zones.
* ELB helps ensure high availability, fault tolerance, and scalability for applications by evenly distributing traffic and seamlessly handling failovers.

**Types:** AWS offers multiple types of load balancers under the Elastic Load Balancing service,

* + - Application Load Balancer
    - Network Load Balancer
    - Classic Load Balancer
* **Application Load Balancer:** Routes traffic based on content at the application layer (Layer 7), ideal for HTTP and

HTTPS traffic, with support for advanced routing features.

* **Network Load Balancer:** Handles high volumes of traffic and operates at the transport layer (Layer 4), suitable for

TCP, UDP, and TLS traffic, offering ultra-low latency.

* **Classic Load Balancer:** Provides basic load balancing across multiple EC2 instances, operating at both the

application and transport layers which is suitable for distributing HTTP, HTTPS, TCP and SSL traffic and offers basic functionality without some of the more advanced features available in the newer Load Balancing types.

Practical:

**Steps to create Load Balancer:**

Step-1: Create two or more instances with same application or web page in it.

Step-2: Create Target Group

Target groups 🡪 Create Target Group 🡪 Instance 🡪 Target Group Name 🡪 VPC 🡪 Next

🡪Select Instances that we have created in step-1 🡪 Create

Step-3: Now, we have to create the Load Balancer.

Load Balancer 🡪 Create load balancer 🡪 Application Load Balancer (create) 🡪 Name 🡪 VPC 🡪 Mapping 🡪 Security group 🡪 Listeners and routing (add Target group) 🡪 create

(Note: Add the instance security groups to the load balancer)

Step-4: After the load balancer is available then copy the DNS and paste it in the browser to see the application. The

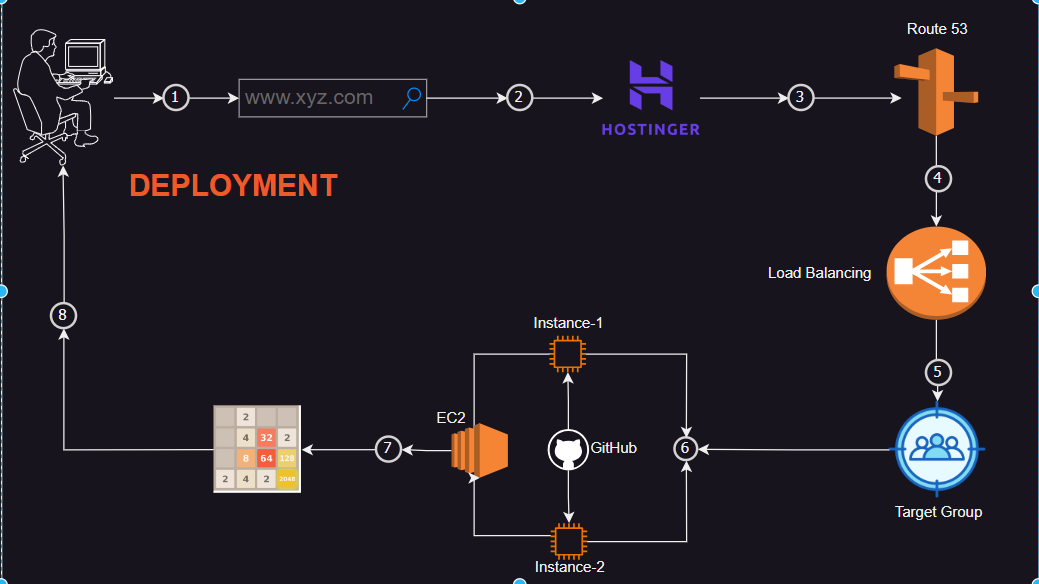
applications will change frome time to time for balancing the load.

# Route53

* Route 53 is a scalable Domain Name System (DNS) web service provided by Amazon Web Services (AWS). It is designed to route end users to internet applications by translating human-readable domain names (e.g., www.example.com) into the numeric IP addresses required for locating and identifying computer services and devices with the underlying network protocols.
* Route 53 is a highly available and scalable Domain Name System (DNS) web service. You can use Route 53 to perform three main functions in any combination: domain registration, DNS routing, and health checking.

**Small real time project:**

Deploying an application in using **EC2**, **Route53**, **Load balancer** and **Git**.



* First of all launch 2 instances and do necessary steps that we used to do while and after connecting to the server. (i.e, adding port numbers and all)
* sudo –s   
  apt update && apt upgrade -y && apt install nginx –y (we can also install apache2)
* Now,we have to do like we have done in the webpage deployment.

open the nginx path cd /var/www/html

* Clone the source code given my the developer from the

git repository. Git clone <repo\_url>

cd <repo\_name>

mv \* .. (Do this to both the instances)

* Now we have to add loadbalancer, for that we have to create the target group

Target groups 🡪 Create Target Group 🡪 Instance 🡪 Target Group Name 🡪 VPC 🡪 Next

🡪Select Instances that we have created 🡪 Create

* Now, create the Load balancer

Load Balancer 🡪 Create load balancer 🡪 Application Load Balancer (create) 🡪 Name 🡪 VPC 🡪 Mapping 🡪 Security group 🡪 Listeners and routing (add Target group) 🡪 create

(NOTE: we have to add the instance security groups to the load balancer)

* Now, we have to purchase one domain from any sites like hostinger , godaddy, etc…
* Now, create one Route53

Route53 🡪 get started 🡪 Create hosted zones 🡪 get started 🡪 domain name 🡪 Create hosted zone

* Copy all the 4 name servers and paste them one by one in the hostinger domain that we have purchased before in the hostinger.
* After the connection was done. Now, we have to create the record in the route53,

Create record 🡪 Record name 🡪 turn on alias for adding Load balancing (off for direct connection to the instance by giving the ip address) 🡪 Choose Application Load Balancer 🡪 Region 🡪 Select the created load balancer 🡪 Create record.

* It will take some time to go online. We can check that in the DNS checker website.
* Once the application become online. Any one can accesss our application by just typing (record\_name.domain\_name).
* Hence the application become live and deployed.

# JENKINS

* **Jenkins** is an open-source automation server that helps automate various tasks in software development, such as building, testing, and deploying applications.
* It's widely used for implementing continuous integration and continuous delivery (CI/CD) pipelines, enabling teams to deliver software changes more rapidly and reliably.
* Java is the prerequisite for Jenkins.
* Jenkins port is 8080.

**CICD:**

* + **Continuous integration CI**: Jenkins can be used to automate the process of building and testing code every time a change is

made to the source repository. This ensures that new code changes are integrated into the main

codebase frequently and tested for any issues, thus improving the overall code quality.

* + **Continuous Delivery CD :** Jenkins can automate the process of deploying applications to AWS infrastructure once they pass

the CI stage. This involves tasks such as packaging the application, deploying it to AWS services like

EC2, ECS, or Lambda, and running any necessary tests in the deployment environment.

**Difference between Continuous Delivery and Continuous Deployment**

**Continuous Deployment:** Involves automatically deploying every successful build to production without manual intervention, assuming it passes all tests and quality checks.

**Continuous Delivery:** Involves the automation of the entire software release process up to the point of deployment to production.

However, the decision to deploy to production is still manual, allowing teams to perform additional checks or

approvals before releasing new changes.

* **Installing Jenkins in EC2:**
* Create one EC2 instance and connect to the server.

sudo –s

apt update && apt upgrade –y

* Install java jre and jdk as java is the prerequisite for Jenkins. (Search in browser)

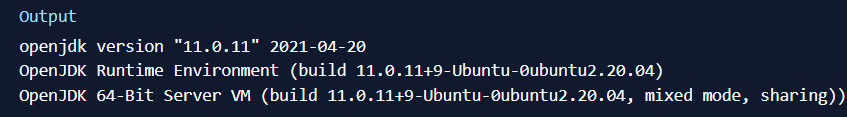
Java –version



Copy one command and paste

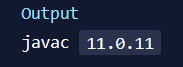
sudo apt install default-jre -y

java –version



sudo apt install default-jdk -y

javac –version



* Now, install Jenkins (Search in browser)

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

https://pkg.jenkins.io/debian/jenkins.io-2023.key

echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]" \

https://pkg.jenkins.io/debian binary/ | sudo tee \

/etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install Jenkins -y

* Check whether the Jenkins got installed or not, systemctl status jenkins
* Open the port 8080 in the security groups of the EC2 instance in which the Jenkins was installed.
* Now, copy the public IP and paste on browser **public\_IP:8080** at the end.
* Now, it will ask for the password and the path of the password is also given there. Copy that path and open it by sing cat command in the terminal. Copy the password and paste in the Jenkins site.
* Install some default plugin’s option and create the account.
* This is how we install Jenkins.
* **Master and Slave Architecture:**
  + **Master:** The Jenkins master is the central server that manages the entire Jenkins environment. It handles tasks such

as scheduling builds, monitoring the execution of jobs, managing plugins, and providing the web interface for users to interact with Jenkins. The master is responsible for coordinating and distributing build jobs to the slave nodes.

* + **Slave:** A Jenkins slave (or agent) is a worker node that performs the actual build jobs assigned by the master.

Slaves can be machines with different operating systems, hardware configurations, or software environments, allowing Jenkins to distribute build workloads across multiple nodes. Slaves connect to the master and wait for instructions to execute build jobs. They report back to the master with build results and logs once the job is completed.

Overall, the master-slave architecture enhances the flexibility, scalability, and reliability of Jenkins as a continuous integration and delivery platform.

**PRACTICAL**

* Create one new instance and connect to the server.

apt update && apt upgrade –y

Now, install java jre and jdk,

apt install default-jre –y

apt install default-jdk –y

* Create one folder mkdir slave (we can give any folder name)
* Now change the owner from root to ubuntu,

chown –R ubuntu:ubuntu slave

cd slave

pwd (copy the path)

* Start the Jenkins installed instance and open Jenkins from browser.

Open Jenkins 🡪 Manage Jenkins 🡪 Nodes 🡪 New node 🡪 Name 🡪 Permanent Agent 🡪 No. of executors🡪

Remote root directory (paste the pwd path) 🡪 give any label name 🡪 launch method (SSH) 🡪

Host (paste the private ip of slave instance) 🡪 Credentials 🡪 add (Jenkins) 🡪

Kind (SSH user name and private key) 🡪 any ID 🡪 Description 🡪 user name🡪 ☑private key enter directly 🡪 add 🡪

(open the .pem key from the linux using cat and copy it) paste here 🡪 add 🡪 host key verification (Manually trusted key)

* Now we can see the nodes online. We can change the options in the slave by configure the slave.

# DOCKER

* **Docker** is a platform that allows developers to package their applications and all of their dependencies into a single unit called a container. This **containerization** technology provides consistency across different environments, making it easier to develop, test, and deploy applications.

**Steps to install docker in EC2 instance:**

Step-1: Create one EC2 instance first and then connect to that instance.

Step-2: sudo –s , apt update , apt upgrade –y (if required)

Step-3: Now, we have to install Docker.

Apt install docker.io –y

Step-4: To check whether the docker is running are not,

Systemctl status docker