**\*\*\*DevOps full notes\*\*\***

**Git and Github**

**Module 1: Introduction to Version Control**

* What is VCS and its advantages
* Installing and Configuring Git
* Creating Github account
* Familiarizing with git bash

**Module 2 : Demystifying the GIT workflow**

— Working area

— Staging area

— local repository

* **Commands used** – git init

– git add <filename>

– git commit

**Module 3 : Remote repositories**

* **Managing Remote repositories**
* **Pushing code to remote repos**
* **Cloning repositories**
* **Git ignore file**

**Module 4: Branching & Merging Repositories**

* **Creating & managing branches**
* **Merging branches**
* **Resolving “ Merge Conflicts “**

Jenkins

1. Introduction to jenkins & installation
2. Building your first job on jenkins

* Intro to jenkins dashboard
* Creating your first job
* Redirecting your job output to a file

1. Creating users in Jenkins & Role Based access(RBAC)

* Creating various users
* Installing required plugins
* Assigning roles

1. Jenkins with email

* Configuring jenkins to send mail
* Sending mail to multiple mail accounts

1. Trigger Builds & Environment Variables & Parameterised jobs

* Triggering jobs using url
* Remote Triggers
* Global & local variables in jenkins
* Parameterised jobs

1. Upstream & Downstream jobs , Periodic Jobs

* Configuring upstream & downstream jobs in jenkins
* Configuring scheduled jobs in jenkins

1. Jenkins & Git

* Introduction to git plugin
* Git with Poll SCM
* Remote triggers with git

1. Jenkins & build pipeline plugin

* Creating simple pipelines with build pipeline plugins
* Adding steps in the pipeline

1. Jenkins pipeline as code & jenkins file

* Introduction to jenkins file
* Writing first pipeline as a code

1. Jenkins & Terraform pipeline

* Creating an ec2 instance with jenkins using terraform

**Terraform**

1. **Introduction to IAAC**

– Understanding IAAC concepts

– Installing Terraform on servers

– Setting up Visual Studio

1. **Deploying Infrastructure with Terraform**

– Authentication and Authorization

– Launching first VM through terraform

– Provider Tiers

– Creating a Github Repository with terraform

– Terraform Destroy

– Understanding Terraform state files

– Desired and Current States

1. **Read, Generate, Modify Configurations**

– Cross Resource Attributes  
 – Output Values

– Terraform Variables

1. **Modules and Remote State Management**

* Understanding DRY principle
* Implementing EC2 module with Terraform
* Variables and Terraform Modules
* Implementing Remote backend with Terraform s3

**Ansible**

**1. Introduction**

* How ansible works
* Setting up ansible

**2. YAML and INI files**

* yaml 101
* yaml challenge
* inventory files 101
* ini challenge
* writing AWS inventory files

**3. Playbooks**

* Understanding the documentation
* Creating your first playbook
* Running playbooks

1. **Services , Handlers & Shell**

* Using the service module
* Understanding handlers
* Creating handlers
* Using shell & debug modules

1. **Ansible Variables**

* Introduction
* Understanding Jinja 2 templating

**Docker**

1. **Introduction to Containers**

– server vs virtual machines vs containers

– Problems Docker solves

– Installing and configuring

Docker

1. **Docker basics**

– Docker cli

– Images vs Docker containers

– Attached and Detached mode

1. **Image creation**

– working with docker Images

– Overview of Dockerfile

– Writing custom docker files

– building custom Images from dockerfiles

– Variables in Docker-files

— Managing Images with CLI

1. **Docker hub**

– Creating docker hub account

– tagging docker images

– Docker commit

– Pushing Images to Central Repository

**– ECR**

1. **Docker Networking**

— Overview of Docker Networking

— Port binding

— bridge and host networks

**Docker Swarm**

1. **Introduction to Swarm**

– Overview of container Orchestration

– Initializing Docker Swarm

– Master nodes & Worker nodes

1. **Services Tasks & Containers**

– Deploying services

– Scaling Swarm services across nodes

– understanding load balancing in swarm

– Multiple approaches of scaling swarm

– Draining Swarm

– Inspecting Swarm Service and Nodes

Kubernetes

#### **Module 1: Understanding Container Orchestration**

* **Introduction to Kubernetes Architecture**
  + **Overview of Kubernetes components (Master, Nodes, etc.)**
  + **Key concepts: Pods, Services, Deployments, ReplicaSets**
  + **The role of Kubernetes in container orchestration**
* **Installing and Configuring Kubectl**
  + **Introduction to kubectl command-line tool**
  + **Installing kubectl on various operating systems**
  + **Configuring kubectl to interact with Kubernetes clusters**

#### **Module 2: Managed Kubernetes Architecture (Amazon EKS)**

* **Understanding Managed Kubernetes Architecture**
  + **Introduction to Amazon Elastic Kubernetes Service (EKS)**
  + **Benefits of using managed Kubernetes services**
* **Eksctl and Kubeconfig File**
  + **Installing and configuring eksctl tool**
  + **Understanding and managing kubeconfig files**
  + **Configuring kubectl to use EKS clusters**
* **Creating Clusters Using AWS CLI**
  + **Step-by-step guide to creating EKS clusters using AWS CLI**
  + **Configuring and managing EKS clusters from the command line**
* **Understanding Eksctl Tool**
  + **Creating and managing EKS clusters using eksctl**
  + **Advanced configurations and best practices for eksctl**

#### **Module 3: Understanding Pods**

* **Managing Pods Using CLI and YAML Formats**
  + **Introduction to Pods and their lifecycle**
  + **Creating and managing Pods using kubectl commands**
  + **Writing YAML files to define Pods**
  + **Practical examples of Pod management**
* **Replicasets**
  + **Understanding ReplicaSets and their role in ensuring Pod availability**
  + **Creating and managing ReplicaSets using YAML and kubectl**
  + **Scaling applications with ReplicaSets**
* **Deployments**
  + **Introduction to Deployments and their purpose**
  + **Creating and managing Deployments using YAML and kubectl**
  + **Advanced Deployment strategies**
* **Rolling Updates & Rollbacks in Deployments**
  + **Implementing rolling updates to ensure zero downtime**
  + **Rolling back to previous versions in case of issues**
  + **Practical examples of performing updates and rollbacks**

**Empowering DevOps with AI: Advanced Troubleshooting in AWS and DevOps**

–IAM Policies Troubleshooting

–S3 Bucket Access Challenges

–Ansible Configuration Validation

–Kubernetes Yaml files Validation

–Linux System Troubleshooting

– AWS Service Outages and Performance Issues

– Git and Github Repository Management

– InspectingTerraform Infrastructure Failures

–Jenkins Build Pipeline Failures

–Ansible Playbook Execution Issues

–Docker Containerization Challenges

**\*\*\*\*Git and Github\*\*\*\***

**Module 1: Introduction to Version Control**

* What is VCS and its advantages
* Installing and Configuring Git
* Creating Github account
* Familiarizing with git bash

**Module 2 : Demystifying the GIT workflow**

— Working area

— Staging area

— local repository

* **Commands used** – git init

– git add <filename>

– git commit

**Module 3 : Remote repositories**

* **Managing Remote repositories**
* **Pushing code to remote repos**
* **Cloning repositories**
* **Git ignore file**

**Module 4: Branching & Merging Repositories**

* **Creating & managing branches**
* **Merging branches**
* **Resolving “ Merge Conflicts “**

---------------------------------------------------------------

**Module 1: Introduction to Version Control**

* What is VCS and its advantages
* Installing and Configuring Git
* Creating Github account
* Familiarizing with git bash

---------------------------------------------------------------------------------------------------------------------

Version control system:

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time.They are also known as source control or SCM (Source Code Management) tools.

(or)

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

**Examples of version control systems:**

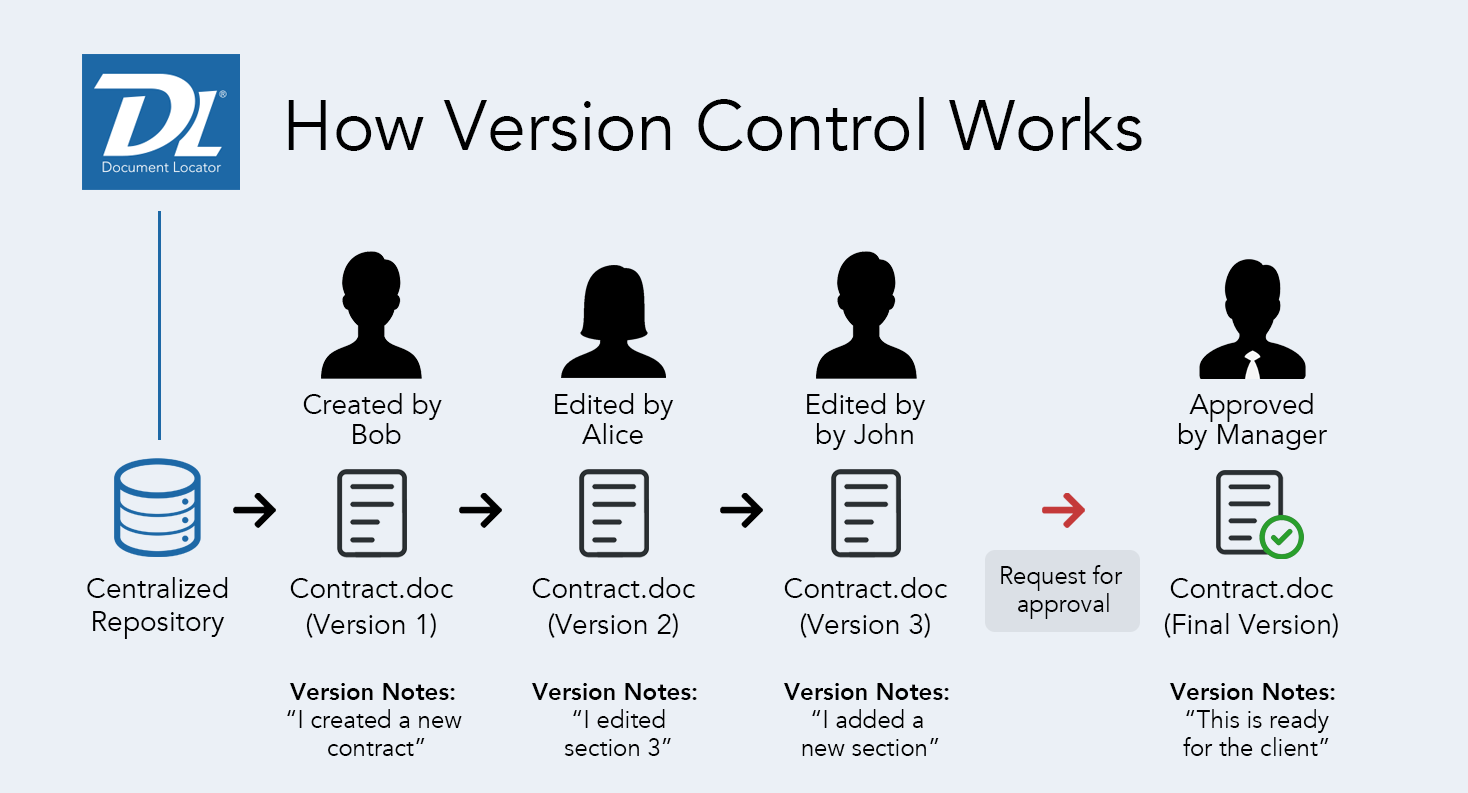
**There are so many version control systems in market in that here….**

**1. Github · 2. GitLab · 3. Beanstalk · 4. PerForce · 5. Apache Subversion · 6. AWS Code Commit · 7. Microsoft Team …….etc**

Why Version Control system is so Important?

As we know that a software product is developed in collaboration by a group of developers they might be located at different locations and each one of them contributes to some specific kind of functionality/features. So in order to contribute to the product, they made modifications to the source code(either by adding or removing). A version control system is a kind of software that helps the developer team to efficiently communicate and manage(track) all the changes that have been made to the source code along with the information like who made and what changes have been made. A separate branch is created for every contributor who made the changes and the changes aren’t merged into the original source code unless all are analyzed as soon as the changes are green signaled they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.

Basically Version control system keeps track on changes made on a particular software and take a snapshot of every modification. Let’s suppose if a team of developer add some new functionalities in an application and the updated version is not working properly so as the version control system keeps track of our work so with the help of version control system we can omit the new changes and continue with the previous version.





Benefits of the version control system:

Enhances the project development speed by providing efficient collaboration,

Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,

Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,

Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this VCS,

For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is Git, Helix core, Microsoft TFS,

Helps in recovery in case of any disaster or contingent situation,

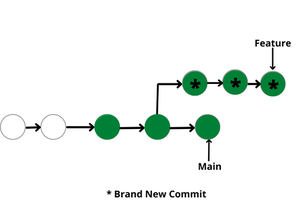
Informs us about Who, What, When, Why changes have been made.

Use of Version Control System:

**A repository:** It can be thought of as a database of changes. It contains all the edits and historical versions (snapshots) of the project.

**Copy of Work (sometimes called as checkout)**: It is the personal copy of all the files in a project. You can edit to this copy, without affecting the work of others and you can finally commit your changes to a repository when you are done making your changes.

**Working in a group**: Consider yourself working in a company where you are asked to work on some live project. You can’t change the main code as it is in production, and any change may cause inconvenience to the user, also you are working in a team so you need to collaborate with your team to and adapt their changes. Version control helps you with the, merging different requests to main repository without making any undesirable changes. You may test the functionalities without putting it live, and you don’t need to download and set up each time, just pull the changes and do the changes, test it and merge it back. It may be visualized as.



Types of Version Control Systems:

Local Version Control Systems

Centralized Version Control Systems

Distributed Version Control Systems

Local Version Control Systems:

 It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

Centralized Version Control Systems:

 Centralized version control systems contain just one repository globally and every user need to commit for reflecting one’s changes in the repository. It is possible for others to see your changes by updating.

**Two things are required to make your changes visible to others which are:**

* **You commit**
* **They update**



The benefit of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what.

It has some downsides as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven’t been kept? You lose absolutely everything.

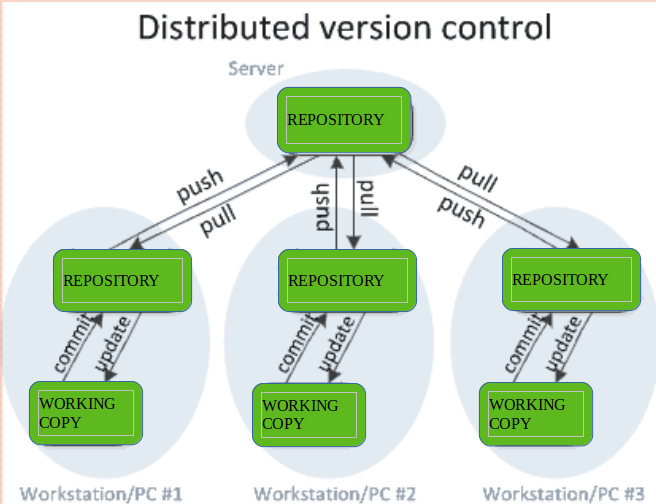
Distributed Version Control Systems:

Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get others’ changes unless you have first pulled those changes into your repository.

**To make your changes visible to others, 4 things are required:**

* **You commit**
* **You push**
* **They pull**
* **They update**

The most popular distributed version control systems are Git, and Mercurial. They help us overcome the problem of single point of failure.



Purpose of Version Control:

* Multiple people can work simultaneously on a single project. Everyone works on and edits their own copy of the files and it is up to them when they wish to share the changes made by them with the rest of the team.
* It also enables one person to use multiple computers to work on a project, so it is valuable even if you are working by yourself.
* It integrates the work that is done simultaneously by different members of the team. In some rare cases, when conflicting edits are made by two people to the same line of a file, then human assistance is requested by the version control system in deciding what should be done.
* Version control provides access to the historical versions of a project. This is insurance against computer crashes or data loss. If any mistake is made, you can easily roll back to a previous version. It is also possible to undo specific edits that too without losing the work done in the meanwhile. It can be easily known when, why, and by whom any part of a file was edited.

Git (Global Information Tracker)

Git: is a free, open-source version control system used to manage source code and track changes in files. It's often used by programmers to collaborate on software development. Git can be used for projects of all sizes, and helps developers work together on non-linear development.

(or)

Git is a an Open Source distributed [version control](https://www.techtarget.com/whatis/definition/version-control) system that is available for free under the [GNU General Public License](https://www.techtarget.com/searchdatacenter/definition/GNU-General-Public-License-GNU-GPL-or-simply-GPL) version 2. The Git source code is hosted on Github, from where it can be downloaded or installed. Users can also utilize one of the binary installers to deploy Git on their systems. For example, they can use [sudo](https://www.techtarget.com/searchsecurity/definition/sudo-superuser-do) to install Git on [Linux](https://www.techtarget.com/searchdatacenter/definition/Linux-operating-system), Homebrew on Mac or a downloadable [executable file](https://www.techtarget.com/whatis/definition/executable-file-exe-file) on Windows.

Git is used primarily by developers to manage their [source code](https://www.techtarget.com/searchapparchitecture/definition/source-code). Git records changes to files over time, while ensuring the integrity of those changes at each stage of processing. Users can [revert to earlier versions](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/remove-revert-discard-local-uncommitted-changes-Git-how-to) and compare different versions at the file level. They can also [branch and merge](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-merge-master-into-any-branch-in-GitLab) files to support independent development efforts and nonlinear [workflows](https://www.techtarget.com/searchcio/definition/workflow). And Git is not limited to source code. It can be used for database [scripts](https://www.techtarget.com/whatis/definition/script), LaTeX documents, [configuration files](https://www.techtarget.com/searchitoperations/definition/configuration-file), content management data or other file types.

Some of its key features include:

Version tracking

Branching for parallel development

Efficient merging of changes

Data integrity

Access controls

Decentralized development:

The history of Git

[Linus Torvalds](https://www.internethalloffame.org/inductee/linus-torvalds/" \t "https://www.techtarget.com/searchitoperations/definition/_blank), creator of [Linux](https://www.techtarget.com/searchdatacenter/definition/Linux-operating-system), along with others in the Linux development community, built and released Git in 2005. They undertook the project because there was a lack of free and open source version control systems that could meet their requirements for Linux [kernel](https://www.techtarget.com/searchdatacenter/definition/kernel) development. They needed a system that could support a large-scale collaborative effort and provide the performance necessary to carry out various tasks, such as applying a [patch](https://www.techtarget.com/searchenterprisedesktop/definition/patch) in a few seconds rather than 30.

Git delivered that performance by providing a solution that did not rely on a centralized repository. Instead, it supported a distributed workflow, while safeguarding against [file corruption](https://www.techtarget.com/searchdatacenter/tip/Use-a-Linux-file-system-journal-for-data-integrity-performance). Git also offered a simple design that supported nonlinear development, making it possible to work on thousands of parallel branches.

Git, as a word, is an alteration of the word get, which itself was shortened from beget. The implicit reference is to illegitimate offspring, and the term is roughly synonymous with twit, dolt, moron or idiot. Within the [open source community](https://www.computerweekly.com/podcast/State-of-open-source-Computer-Weekly-Downtime-Upload-podcast), the significance of the name choice varies. The readme file in the Git source code on Github states this about the name:

The name "git" was given by Linus Torvalds when he wrote the very first version. He described the tool as "the stupid content tracker" and the name as (depending on your mood):

* random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
* stupid, contemptible and despicable. simple. Take your pick from the dictionary of slang.
* "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
* "goddamn idiotic truckload of sh\*t": when it breaks.

How does Git work?

Because Git is a distributed system, it can be used with or without a central [repository](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Create-a-GitHub-repository-with-this-step-by-step-tutorial), unlike centralized version control systems that require a server or [hosting service](https://www.techtarget.com/searchitchannel/definition/hosted-services) to maintain the primary repository. With Git, each user maintains a local copy, or clone, of the repository, including its entire history. As a result, each clone serves as a backup, eliminating any [single point of failure](https://www.techtarget.com/searchdatacenter/definition/Single-point-of-failure-SPOF). Most operations that users perform are carried out locally, so they're not impacted by network [latency](https://www.techtarget.com/whatis/definition/latency) issues. Users can also work out whether or not they're connected to the network.

After installing Git on their systems, users can work with their repositors by entering commands at a terminal or by using a [graphical user interface](https://www.techopedia.com/definition/5435/graphical-user-interface-gui" \t "https://www.techtarget.com/searchitoperations/definition/_blank), such as [Sourcetree](https://www.techtarget.com/searchsoftwarequality/tip/GitKraken-vs-Sourcetree-Pick-a-Git-GUI-that-fits-dev-needs) or [Github Desktop](https://desktop.github.com/" \t "https://www.techtarget.com/searchitoperations/definition/_blank). They can then [set up their own projects](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/sourcetree-tutorial-bitbucket-git-course-branch-merge-conflict-install-download-rebase-reset) or join another project. To join a project, users [clone the repository](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Create-Git-Repo-Repository-Init-New-Clone-Example-Tutorial) from another location to their desktops, where they can modify the existing files or add new ones. The files in a user's local repository are always in one of three states:

Modified. The user has modified one or more files but has not staged or [committed those files](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/git-clone-specific-commit-id-single-branch-reset-depth-head-main-master). Changes occur in the working directory, which is the project folder that hosts the repository. Any files that are changed or added to this folder are considered to be in a modified state until they are staged.

Staged. The user has marked the new or modified files as being ready to commit to the repository. [Staged files](https://www.theserverside.com/tutorial/Five-basic-Git-commands-every-beginner-needs-to-know) are added to the staging area, which is a logical, intermediate area for reviewing the files before committing them. The staging area is more a designation than a physical location. Staged files are tracked in a special file named index.

Committed. The user [commits](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/uncommit-git-last-commit-file-changes-pushed-deleted-message) the staged files to the repository. Each commit operation represents a version of the repository that is assigned a unique checksum (Secure Hash Algorithm 1), which is used to reference the commit in subsequent operations.

After users commit their changes locally, they can [push them out to a central repository or to other users' repositories](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/git-push-new-branch-remote-github-gitlab-upstream-example). Users can also pull changes from a central repository or from other users' repositories. For example, a user might update several files, stage and commit the files, and then push them out to the team's central [repository on Github](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/How-to-clone-a-git-repository-with-submodules-init-and-update). The other team members can then pull the updated files to their own systems to ensure they're working with the latest version. However, there is nothing in Git to prevent users from pushing and pulling changes directly from each other without relying on a central repository.

## **10 Common Git Commands**

1. **git add** – moves changes from the working directory into the staging area
2. **git branch** – a general-purpose branch administration tool for creating isolated development environments within a depository
3. **git checkout** – allows the user to navigate existing branches, view old commits and old file revisions
4. **git clean** – remove untracked files from a working directory
5. **git clone** – clones the existing repository
6. **git commit** – commits the current version of the project into the project history
7. **git config** – configures options for a Git installation
8. **git fetch** – downloads a branch and its associated commits and files from another repository without integrating anything into the local repository
9. **git init** – initializes a new Git repository
10. **git log** – provides several formatting options for exploring previous revisions of a project

How to install and configure git?

Setting up Git

1. Download and install the latest version of Git. Note: Most Chrome OS devices from 2020 onwards now have a built-in Linux environment, which includes Git. To enable it, go to the Launcher, search for Linux, and click Turn on. …
2. Set your username in Git.
3. Set your commit email address in Git.

Where is git installation?

The default path is **“D:\Program Files\Git“**. If you want the software installed in a different location click Browse and specify a different folder.

How is git configured?

The git config command is a convenience function that is used to set Git configuration values on a global or local project level. These configuration levels correspond to .gitconfig text files. Executing git config will modify a configuration text file.

How to configure the git user?

Configure your Git username/email

1. Open the command line.
2. Set your username: git config --global user.name "FIRST\_NAME LAST\_NAME"
3. Set your email address: git config --global user.email ["MY\_NAME@example.com"](mailto:\"MY_NAME@example.com\")

* Creating Github account
* Familiarizing with git bash

## **What is Git Bash?**

Git Bash is a command-line interface (CLI) that combines the functionality of Git and Bash, a popular Unix shell. It offers a user-friendly way to interact with Git repositories, perform version control tasks, and navigate the file system. Git Bash's key features include its extensive Git command support, integration with various operating systems, and the ability to run Unix-like commands on Windows.

Incorporating the power of Git Worktrees can substantially improve your ability to manage multiple branches concurrently, so don't forget to explore our detailed guide on [Git Worktree](https://www.hatica.io/blog/git-worktree/" \t "https://www.hatica.io/blog/what-is-git-bash/_blank) to learn more. Additionally, to expand your Git knowledge and productivity, check out our comprehensive list of [essential Git Commands for Developers](https://www.hatica.io/blog/git-commands-for-developers/" \t "https://www.hatica.io/blog/what-is-git-bash/_self).

## **Installing Git Bash**

To get started with Git Bash, follow the installation instructions for your operating system:

* ****Windows****: Download the installer from the official [Git website](https://git-scm.com/download/win" \t "https://www.hatica.io/blog/what-is-git-bash/_blank) and follow the installation prompts.
* ****macOS****: Install Git using Homebrew by running ****"brew install git****" in the Terminal. Git Bash is included by default.
* ****Linux****: Install Git using your package manager (e.g., "****sudo apt-get install git****" on Ubuntu).

## **Configuring Git Bash**

Once installed, set up your Git Bash user information by running the following commands:

git config --global user.name "Your Name"  
git config --global user.email "your@email.com"

What benefits come with using GIT?

• Data replication and redundancy are both possible.

• It is a service with high availability.

• There can only be one Git directory per repository.

• Excellent network and disc performance are achieved.

• On any project, collaboration is very simple.

What is a repository in Git?

A repository, or repo, is a storage location where a project's files and their revision history are stored. It can be local or remote (on a server..

Which is an example of a version control system?

A specific example of version control. Git is a popular version control system. You can use Git to manage the versions of your computer programs and work on programs in conjunction with other developers. Git is itself a computer program.

Why Git is a version control system?

Git is a distributed version control system that enables software development teams to have multiple local copies of the project's codebase independent of each other.

Git hands-on some commands

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /

$ cd /d

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d (master)

$ ls

**'$RECYCLE.BIN'**/ **Temp**/ **gitproject**/

**'Network - Shortcut.lnk'**\* **WindowsApps**/ 'images (1).jpeg'

**'Program Files'**/ **XboxGames**/ 'ms office id.txt'

**'System Volume Information'**/ **git**/ **'note pad'**/

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d (master)

$ cd git

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (master)

$ echo " this is devops training session " >> ram

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (master)

$ ls

Ram

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git add .

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git status

On branch boys

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: divya

new file: master

modified: ram

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git init

Reinitialized existing Git repository in D:/git/.git/

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git commit -m "this is my third commit in ram " >> ram

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git log

commit b0ddea2355bef47a1cc7e7b9d19d37219621f309 (**HEAD** -> **boys**)

Author: raghavaram <tragavram@gmail.com>

Date: Sun Aug 25 19:36:02 2024 +0530

this is my third commit in ram

commit 35af7fba8d41dfc16efa598369fcda3ebab526cb (**origin1/boys**)

Author: raghavaram <tragavram@gmail.com>

Date: Sun Aug 25 19:25:24 2024 +0530

this is my frst commit doen by divya

commit 64bd25284d8c97007f35608b76f76113ae8a6038 (**origin1/names**, **names**)

Author: raghavaram <tragavram@gmail.com>

Date: Sun Aug 25 19:22:04 2024 +0530

this commit changes by krishna

commit df33b2f65d56e11c5a0f8448e55419a0d2170fdd (**master**)

Author: raghavaram <tragavram@gmail.com>

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git remote add origin1 (paste repository link here ) [https://Github.com/Raghavaram96/practice.git](https://github.com/Raghavaram96/practice.git)

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git config -- globaluser.name "raghavaram"

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git config -- globaluser.gmail "tragavram@gmail.com"

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git push origin1 master

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git branch heros

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (boys)

$ git switch heros

Switched to branch 'heros'

M ram

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (heros)

$ pwd

/d/git

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (heros)

$ git branch -a

boys

\* heros

master

names

remotes/origin1/boys

remotes/origin1/master

remotes/origin1/names

RAGHAVA RAM@DESKTOP-BO31MNR MINGW64 /d/git (heros)

$ git merge heros master

Purpose of git abort (git --abort)

The **git --abort** command isn't actually a valid Git command on its own. However, you might be referring to git merge --abort or git rebase --abort, both of which are used to cancel ongoing operations.

**Here's a quick overview of their purposes:**

git merge --abort: This command is used to stop a merge process that is in progress and revert the working directory to the state it was in before the merge started. This is useful if you encounter conflicts during a merge and decide that you don’t want to proceed with merging the branches.

git rebase --abort: This command is used to cancel an ongoing rebase operation and return the repository to its previous state before the rebase started. It's helpful if you encounter issues or conflicts during a rebase and decide to stop and revert.

In both cases, these commands are helpful for aborting potentially disruptive operations and returning your repository to a stable state.

--------------------------------------------------------------

\*\*\*\*\*Jenkins\*\*\*\*\*

1. Introduction to jenkins & installation
2. Building your first job on jenkins

* Intro to jenkins dashboard
* Creating your first job
* Redirecting your job output to a file.

1. Creating users in Jenkins & Role Based access(RBAC)

* Creating various users
* Installing required plugins
* Assigning roles

1. Jenkins with email

* Configuring jenkins to send mail
* Sending mail to multiple mail accounts

1. Trigger Builds & Environment Variables & Parameterised jobs

* Triggering jobs using url
* Remote Triggers
* Global & local variables in jenkins
* Parameterised jobs

1. Upstream & Downstream jobs , Periodic Jobs

* Configuring upstream & downstream jobs in jenkins
* Configuring scheduled jobs in jenkins

1. Jenkins & Git

* Introduction to git plugin
* Git with Poll SCM
* Remote triggers with git

1. Jenkins & build pipeline plugin

* Creating simple pipelines with build pipeline plugins
* Adding steps in the pipeline

1. Jenkins pipeline as code & jenkins file

* Introduction to jenkins file
* Writing first pipeline as a code

1. Jenkins & Terraform pipeline Creating an ec2 instance with jenkins using terraform

---------------------------------------------------------------------------------------

1. Introduction to jenkins & installation.

(https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/)

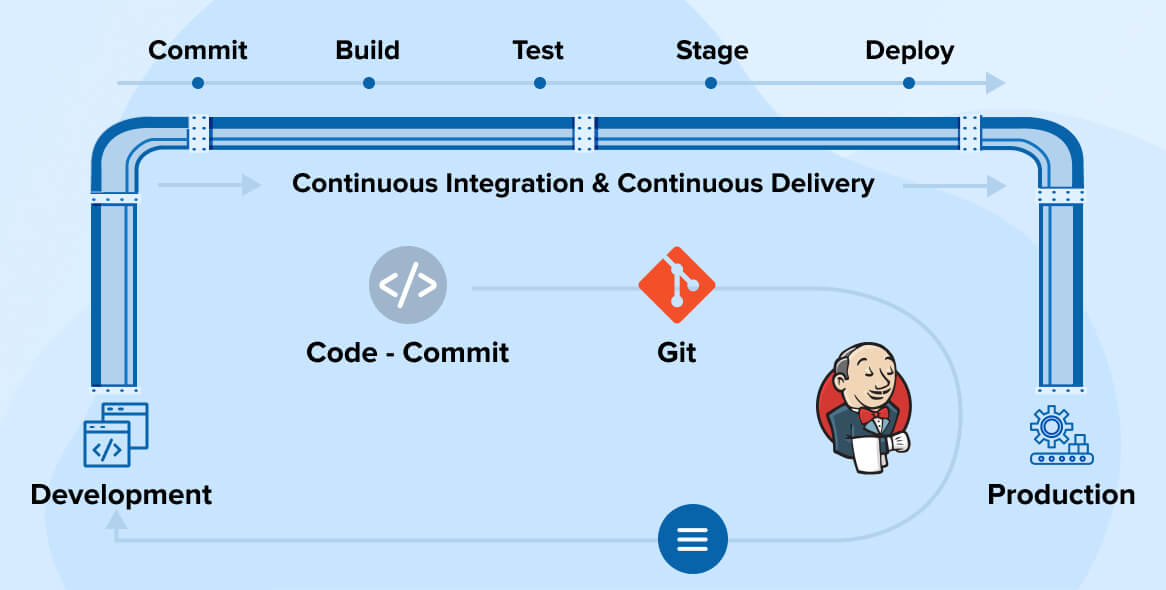
Jenkins :

Jenkins is an [open source](https://en.wikipedia.org/wiki/Open_source" \o "Open source) [automation](https://en.wikipedia.org/wiki/Automation" \o "Automation) [server](https://en.wikipedia.org/wiki/Server_(computing)" \o "Server (computing)). It helps automate the parts of [software development](https://en.wikipedia.org/wiki/Software_development" \o "Software development) related to [building](https://en.wikipedia.org/wiki/Software_build" \o "Software build), [testing](https://en.wikipedia.org/wiki/Test_automation" \o "Test automation), and [deploying](https://en.wikipedia.org/wiki/Software_deployment" \o "Software deployment), facilitating [continuous integration](https://en.wikipedia.org/wiki/Continuous_integration" \o "Continuous integration), and [continuous delivery](https://en.wikipedia.org/wiki/Continuous_delivery" \o "Continuous delivery)(CICD). It is a server-based system that runs in [servlet containers](https://en.wikipedia.org/wiki/Java_Servlet" \l "Container_servers" \o "Java Servlet) such as [Apache Tomcat](https://en.wikipedia.org/wiki/Apache_Tomcat" \o "Apache Tomcat). It supports [version control](https://en.wikipedia.org/wiki/Version_control" \o "Version control) tools, including [AccuRev](https://en.wikipedia.org/wiki/AccuRev_SCM" \o "AccuRev SCM), [CVS](https://en.wikipedia.org/wiki/Concurrent_Versions_System" \o "Concurrent Versions System), [Subversion](https://en.wikipedia.org/wiki/Apache_Subversion" \o "Apache Subversion), [Git](https://en.wikipedia.org/wiki/Git" \o "Git), [Mercurial](https://en.wikipedia.org/wiki/Mercurial" \o "Mercurial), [Perforce](https://en.wikipedia.org/wiki/Perforce" \o "Perforce), [ClearCase](https://en.wikipedia.org/wiki/ClearCase" \o "ClearCase), and [RTC](https://en.wikipedia.org/wiki/Rational_Team_Concert" \o "Rational Team Concert), and can execute [Apache Ant](https://en.wikipedia.org/wiki/Apache_Ant" \o "Apache Ant), [Apache Maven](https://en.wikipedia.org/wiki/Apache_Maven" \o "Apache Maven), and [sbt](https://en.wikipedia.org/wiki/Sbt_(software)" \o "Sbt (software)) based projects as well as arbitrary [shell scripts](https://en.wikipedia.org/wiki/Shell_script" \o "Shell script) and Windows [batch commands](https://en.wikipedia.org/wiki/Batch_file" \o "Batch file).

Builds

[Builds](https://en.wikipedia.org/wiki/Software_build" \o "Software build) can be triggered by various means, for example:

* a webhook that gets triggered upon pushed [commits](https://en.wikipedia.org/wiki/Commit_(version_control)" \o "Commit (version control)) in a version control system
* scheduling via a [cron](https://en.wikipedia.org/wiki/Cron" \o "Cron)-like mechanism
* requesting a specific build [URL](https://en.wikipedia.org/wiki/Uniform_Resource_Locator" \o "Uniform Resource Locator).
* after the other builds in the queue have completed
* invoked by other builds



Jenkins installation :

(https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/)

Installation of Jenkins

Create an ec2 instance & launch instance >> sudo yum update –y >> sudo wget -O /etc/yum.repos.d/jenkins.repo \ <https://pkg.jenkins.io/redhat-stable/jenkins.repo> >> sudo rpm --import <https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key> >> sudo yum upgrade >> sudo yum install java -y >> sudo yum install jenkins -y >> sudo systemctl enable jenkins >> sudo systemctl start jenkins >> sudo systemctl status jenkins >> create security group with port number 8080 >> connect to the instance with the help of I>P address :8080 >> sudo cat /var/lib/jenkins/secrets/initialAdminPassword >> copy the intial admin password paste into the jenkins port >>

Role - based authorized statergy

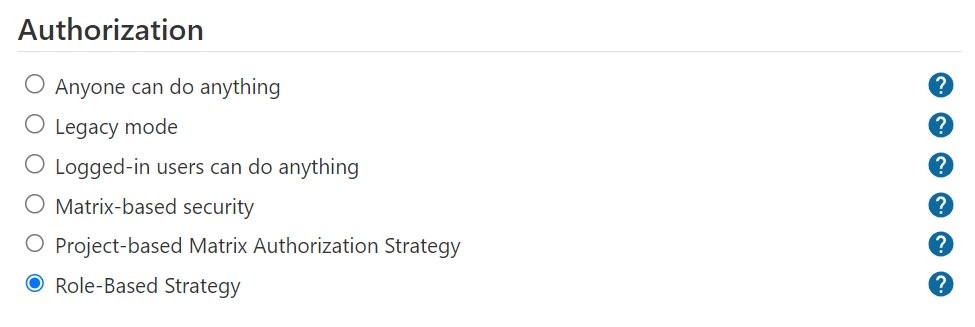
The Role Strategy plugin is meant to be used from [Jenkins](https://jenkins.io/) to add a new role-based mechanism to manage users' permissions. Supported features

* Creating global roles, such as admin, job creator, anonymous, etc., allowing to set Overall, Agent, Job, Run, View and SCM permissions on a global basis.
* Creating item roles, allowing to set item specific permissions (e.g Job, Run or Credentials) on Jobs, Pipelines and Folders.
* Creating agent roles, allowing to set agent specific permissions.
* Assigning these roles to users and user groups
* Extending roles and permissions matching via [Macro extensions](https://github.com/jenkinsci/role-strategy-plugin/blob/master/docs/MACROS.md)

**Usage**

**Installing and enabling the plugin**

The Role Strategy plugin can be installed from any Jenkins installation connected to the Internet using the Plugin Manager screen. Activate the Role-Based Strategy by using the standard Manage Jenkins > Configure Global Security screen:

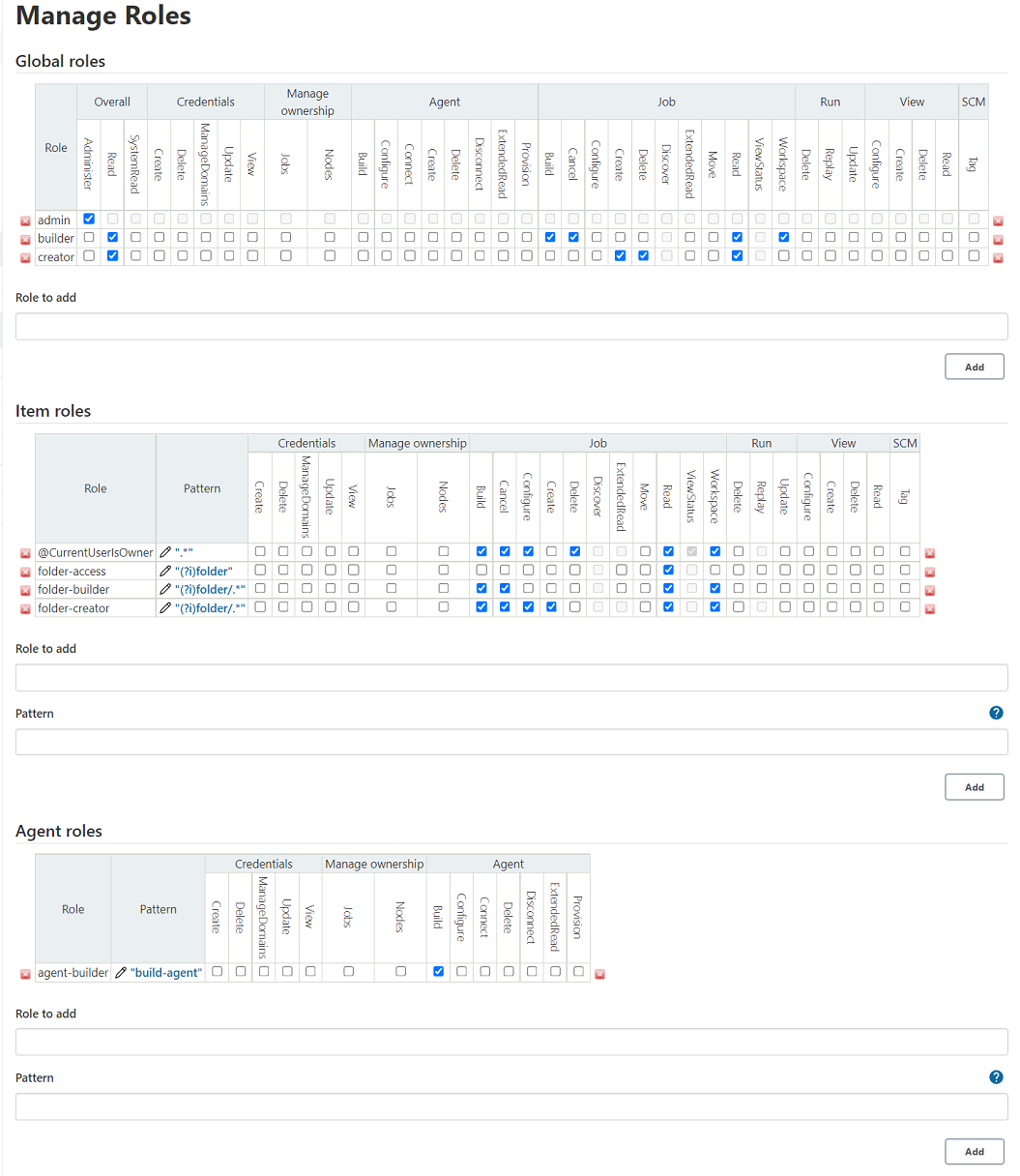
[](https://github.com/jenkinsci/role-strategy-plugin/blob/master/docs/images/configureSecurity.png)

After the installation, the plugin can be configured using the Manage and Assign Roles screen accessible from Manage Jenkins .

**Configuring roles**

You can define roles by using the Manages Roles screen. It is possible to define global, item and agent specific roles.

* Global roles apply to any item in Jenkins and override anything you specify in the Item Roles. That is, when you give a role the right Job/Read in the Global Roles, then this role is allowed to read all Jobs, no matter what you specify in the Item Roles. Giving Job/Create in a global role will allow to create jobs of any name.
* For item and agent roles you can set a regular expression pattern for matching items. The regular expression aimes at matching the full item name.
* For example, if you set the field to Roger-.\*, then the role will match all jobs which name starts with Roger-.
* Patterns are case-sensitive. To perform a case-insensitive match, use (?i) notation: upper, Roger-.\* vs. lower, roger-.\* vs. case-insensitive, (?i)roger-.\*.
* Folders can be matched using expressions like ^foo/bar.\*. To access jobs inside a folder, the folder itself must also be accessible to the user. This can be achieved with a single pattern like (?i)folder($|/.\*) when the permissions on the folder can be the same as for the jobs. If different permissions need to be configured 2 different roles need to be created, e.g. (?i)folder and (?i)folder/.\*. Note that job names inside folders are case-sensitive, though this is probably a bug in the folders plugin [JENKINS-67695](https://issues.jenkins.io/browse/JENKINS-67695). Case sensitivity can be enabled with (?-i), e.g. (?i)folder/(?-i).\*
* Create permissions on item level can only reliably work when the Naming Strategy is set to Role-Based strategy in the global configuration for Restrict project naming. You should see a warning in the administrative monitors if it is not enabled. Only jobs matching the pattern can be created. When granting Job/Create you should also grant Job/Configure and Job/Read otherwise you will be able to create new jobs but you will not be able to configure them. Global Permissions are not required.



Permission Templates

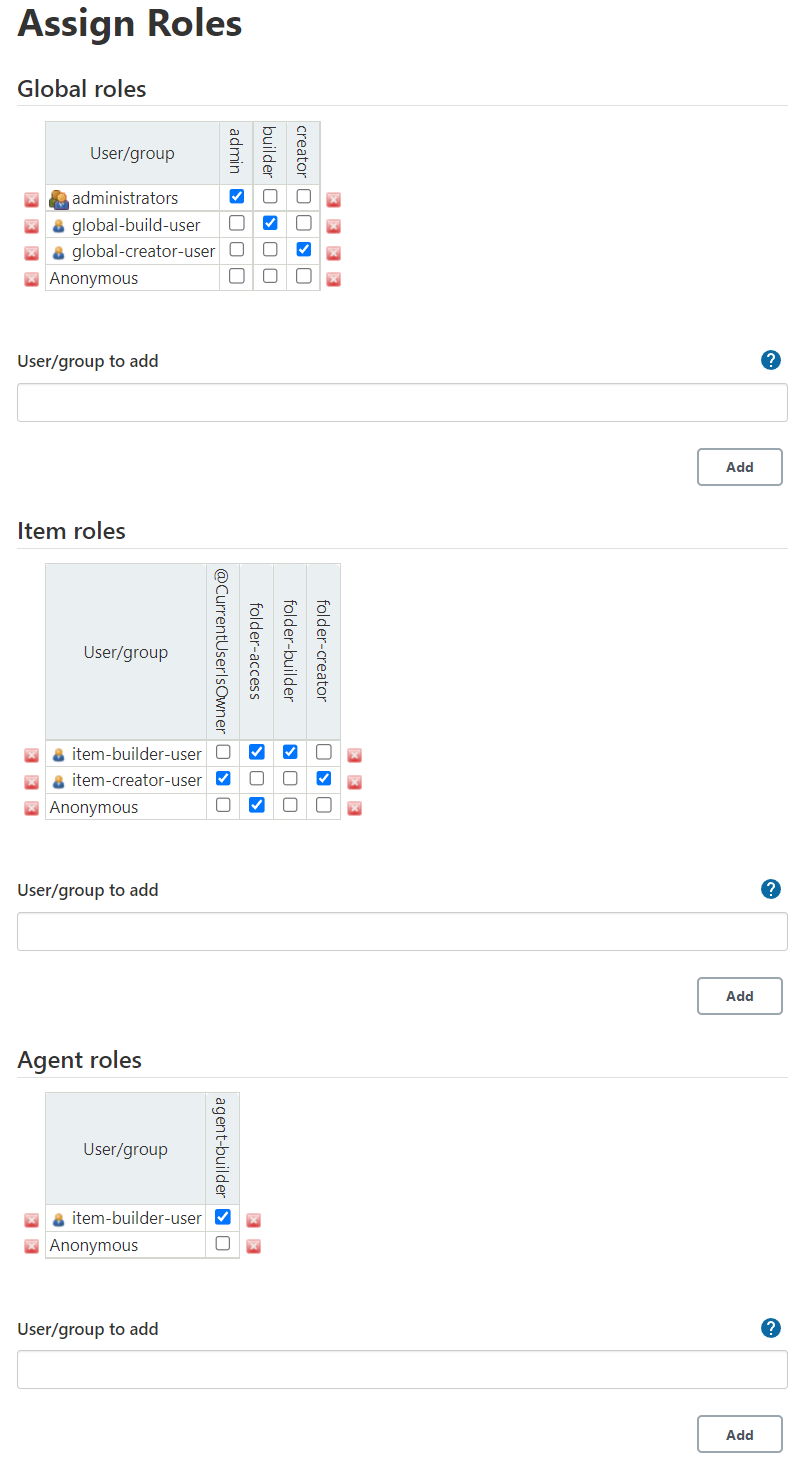
Permission Templates simplify the administration of roles when you need to maintain many roles with identical permissions but different patterns. Templates are only available for Item Roles. The permissions of roles based on a template can't be modified directly. Modifying the template will immediately modify the linked roles after saving the changes.

Deleting a template that is still in use requires confirmation. In case you still delete it, the roles stay with the given permissions but the correlation to the template is removed.

Assigning roles

You can assign roles to users and user groups using the Assign Roles screen

* User groups represent authorities provided by the Security Realm (e.g. Active Directory or LDAP plugin can provide groups)
* There are also two built-in groups: authenticated (users who logged in) and anonymous (any user, including ones who have not logged in)
* Hovering over the header row will show a tooltip with the permissions associated to the role and the pattern.
* Hovering over a checkbox will show a tooltip with role, user/group and pattern.



1. Building your first job on jenkins

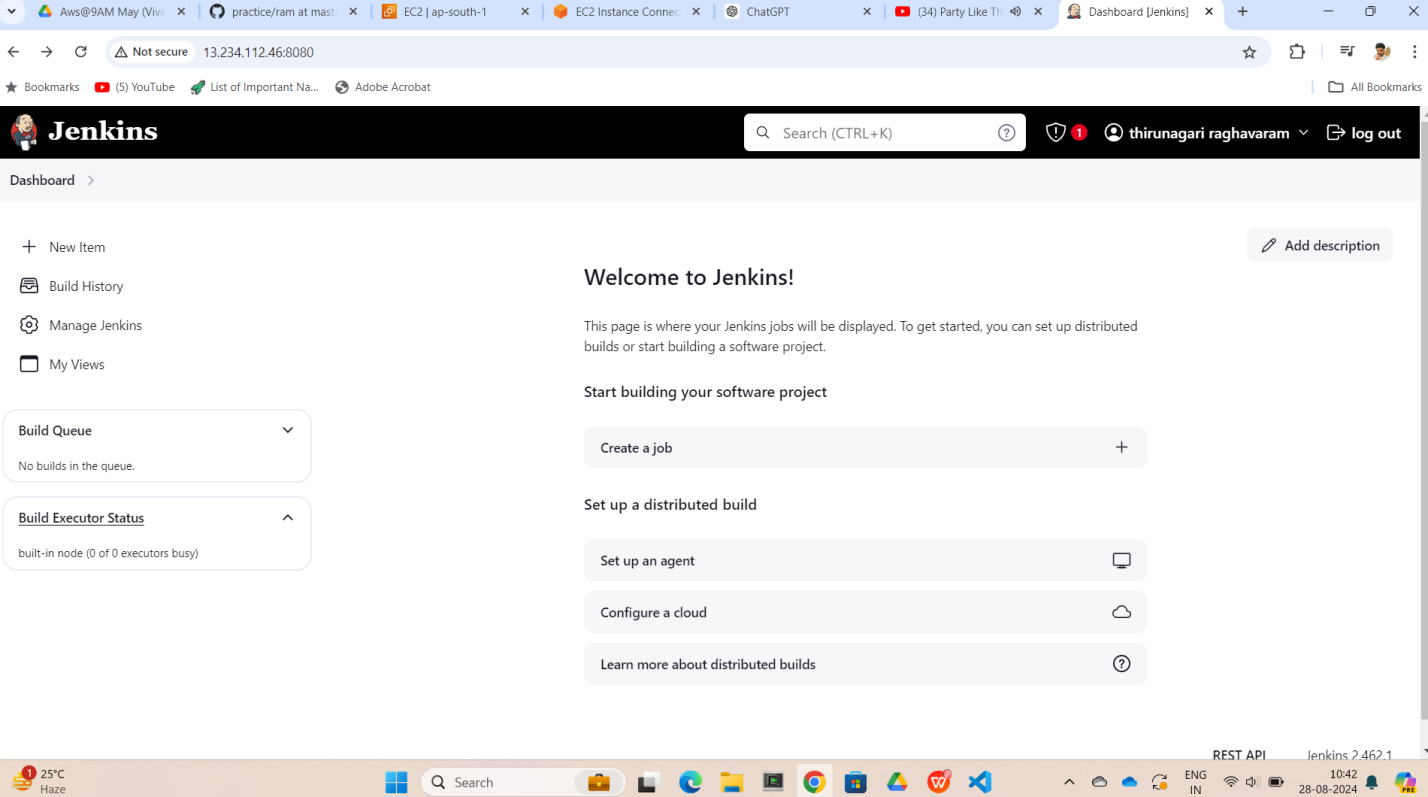
* Intro to jenkins dashboard
* Creating your first job

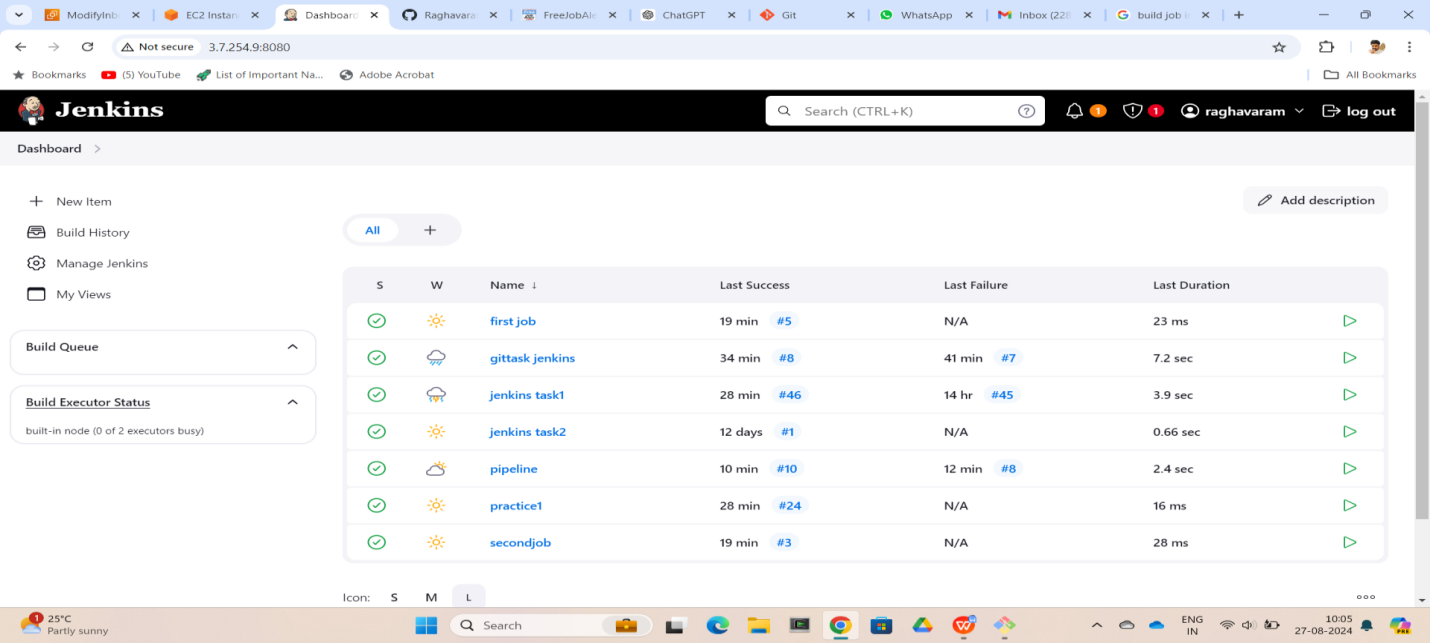
Redirecting your job output to a file.

How do I access Jenkins dashboard?

To see Jenkins, simply bring up a web browser and go to URL http :// myServer :8080 where myServer is the name of the system running Jenkins

\*\*\*\*\*Jenkins dashboard\*\*\*\*\*

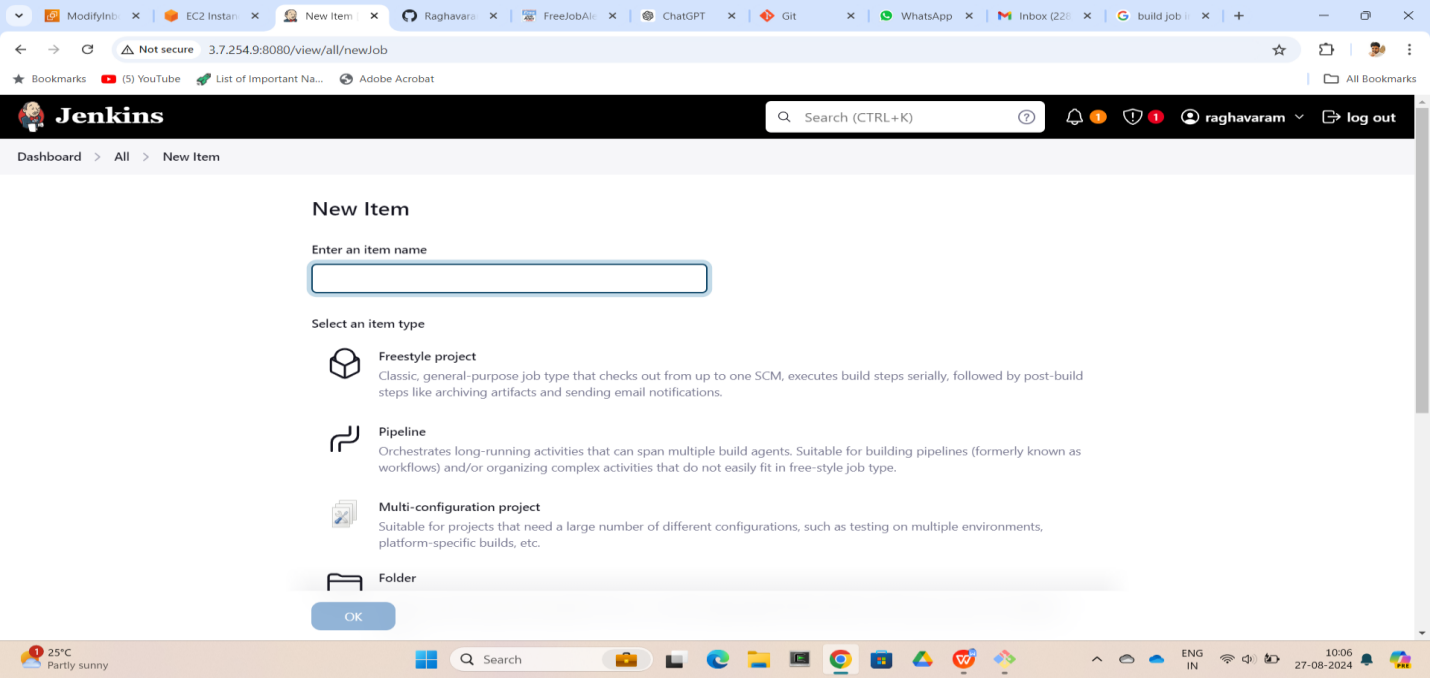


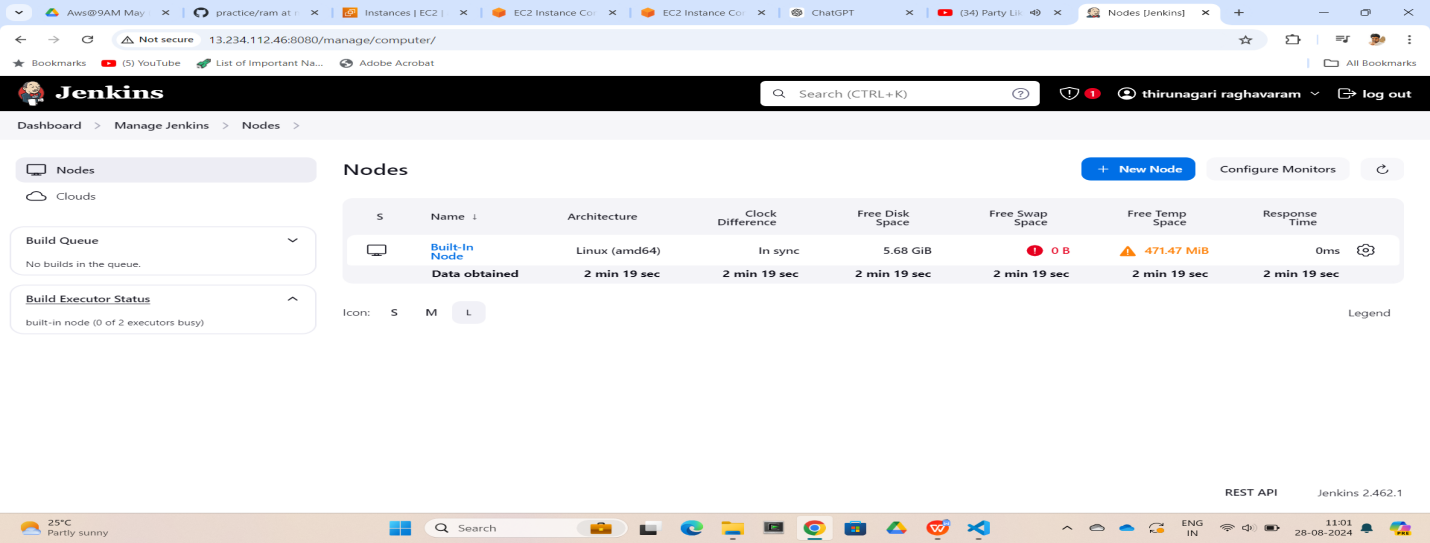


**What is a dashboard in Jenkins?**

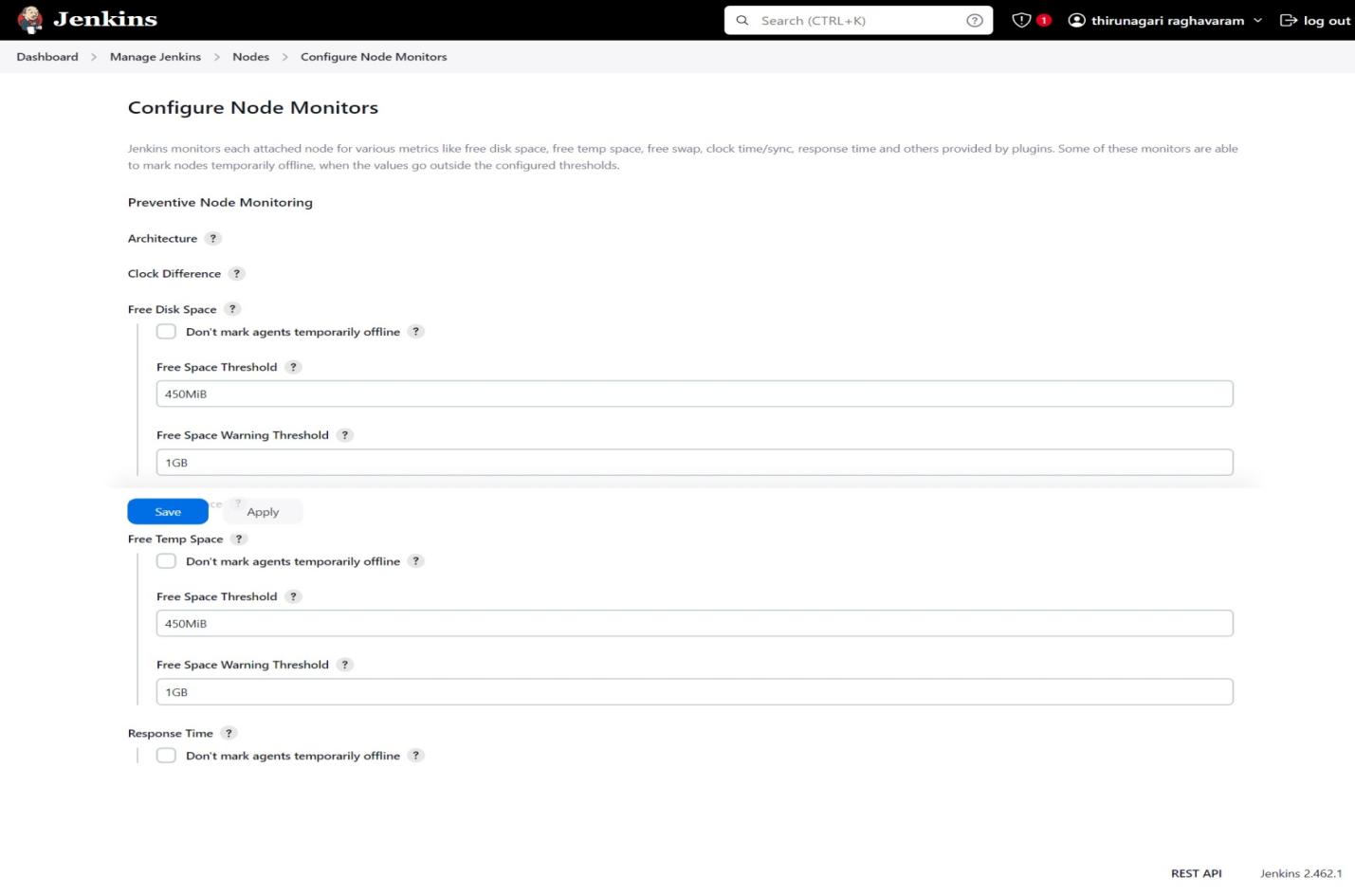
The Jenkins Dashboard serves as the nerve center of your CI/CD environment, providing a unified interface for various tasks, from configuring build jobs to monitoring real-time pipeline executions.

\*\*\*create new job in Jenkins\*\*





Following steps (hands-on) : Manage Jenkins >> nodes&clouds >>click on configure monitors >> scroll down to node properties and locate disk space monitoring thresholds>adjust the thresholds as follows >>Free disk space threshold 450MiB, Free disk space warning trash 1GB ,Free temp space threshold 450MiB, Free temp space warning trash 1GB >> then apply and save and then restart your Jenkins “sudo systemctl restart jenkins”



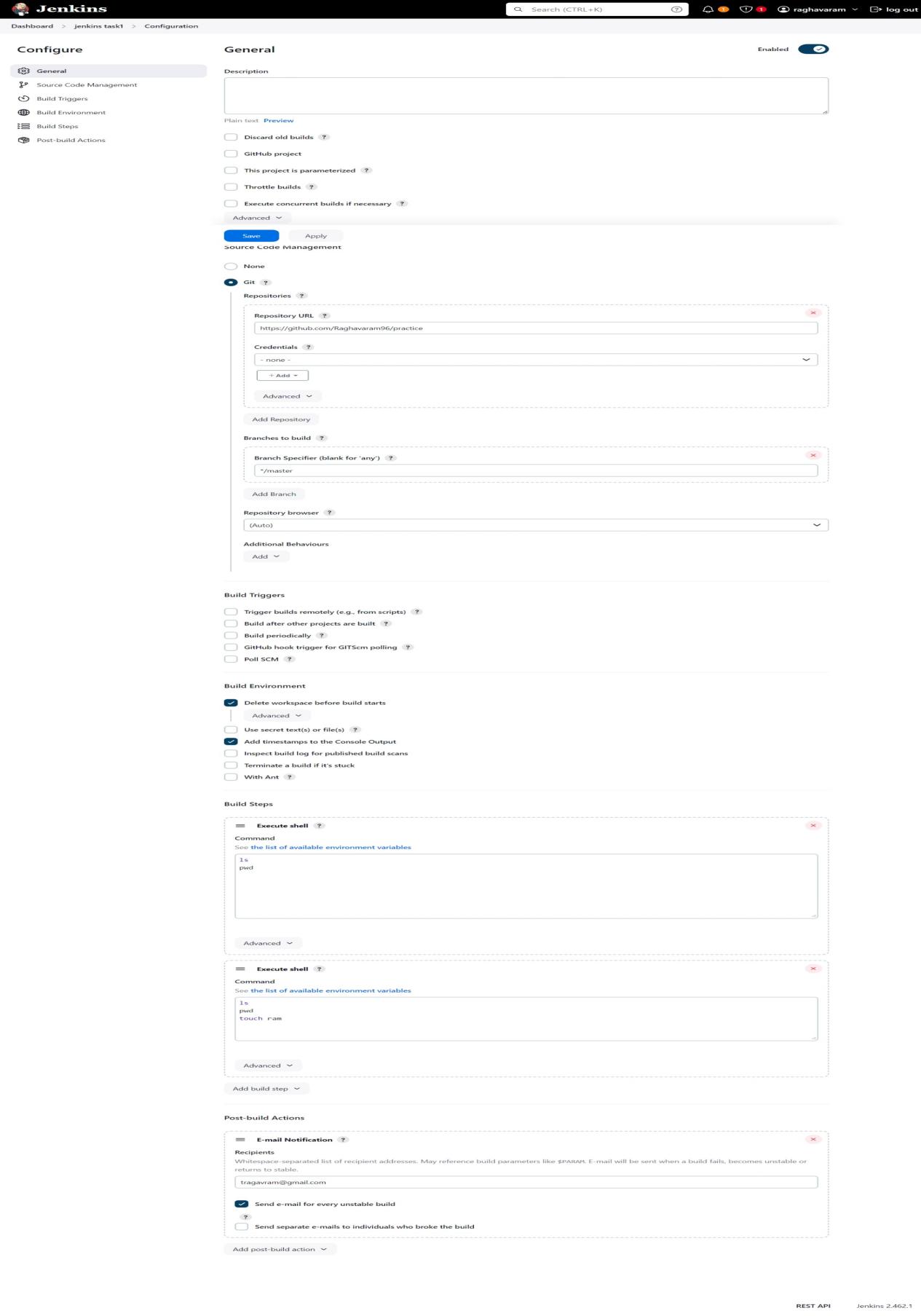
Check weather workspace created or not in Linux V.M

Cd /var/lib/jenkins/workspace

Jenkins work-location path /var/lib/jenkins



\*\*\* build jobs in jenkins \*\*\*



**What is a build job in Jenkins?**

A Jenkins build job contains the configuration for automating a specific task or step in the application building process. These tasks include gathering dependencies, compiling, archiving, or transforming code, and testing and deploying code in different environments.

**How many types of jobs are there in Jenkins?**

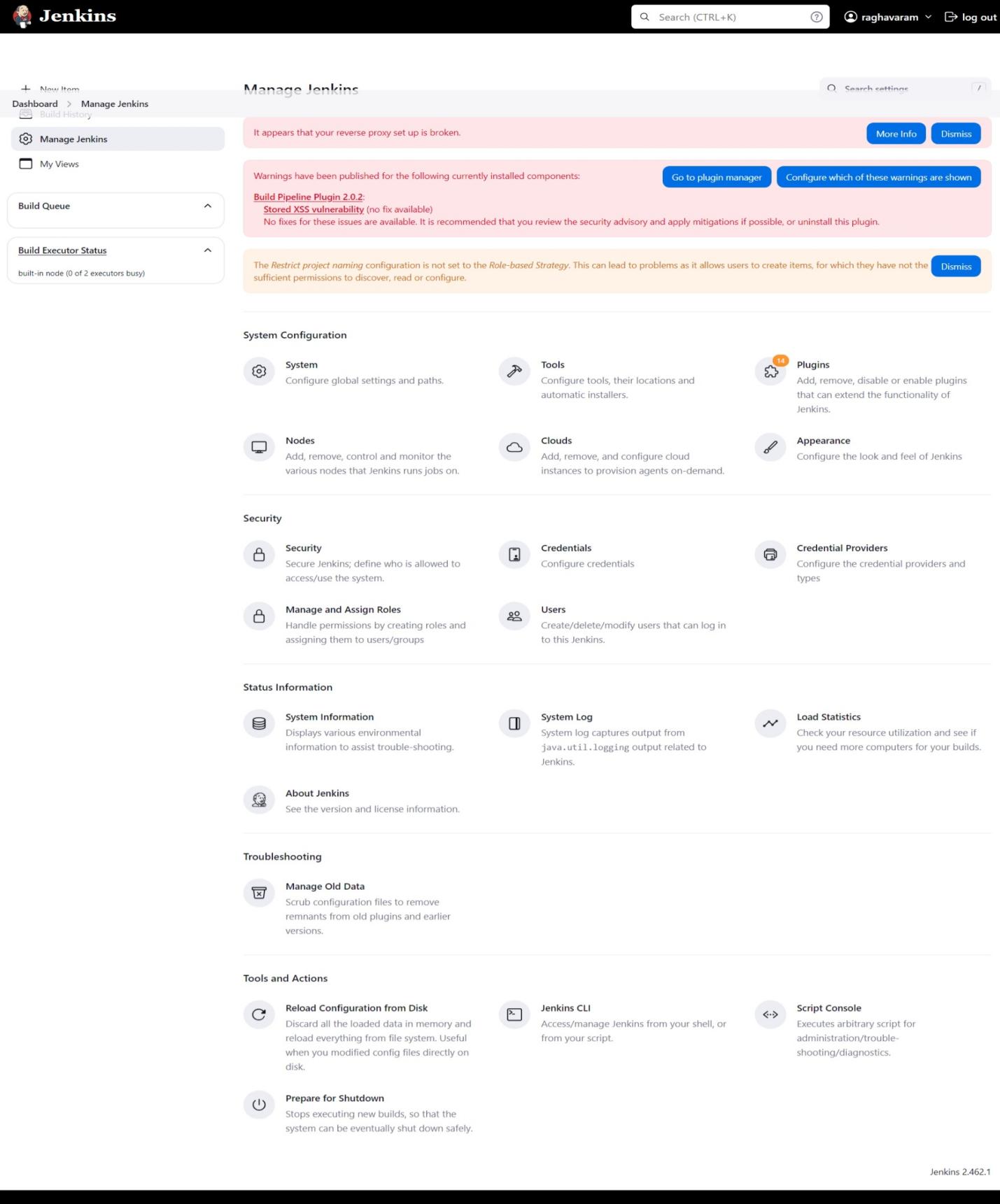
Using Jenkins Job, we can pull the source code from version control, create it build, run tests, verify the status of builds and deploy, etc. Types of Jenkins Jobs are Freestyle Project, Maven Project, Pipeline, Multi-configuration Project, and Github Organization.

1. Creating users in Jenkins & Role Based access(RBAC)

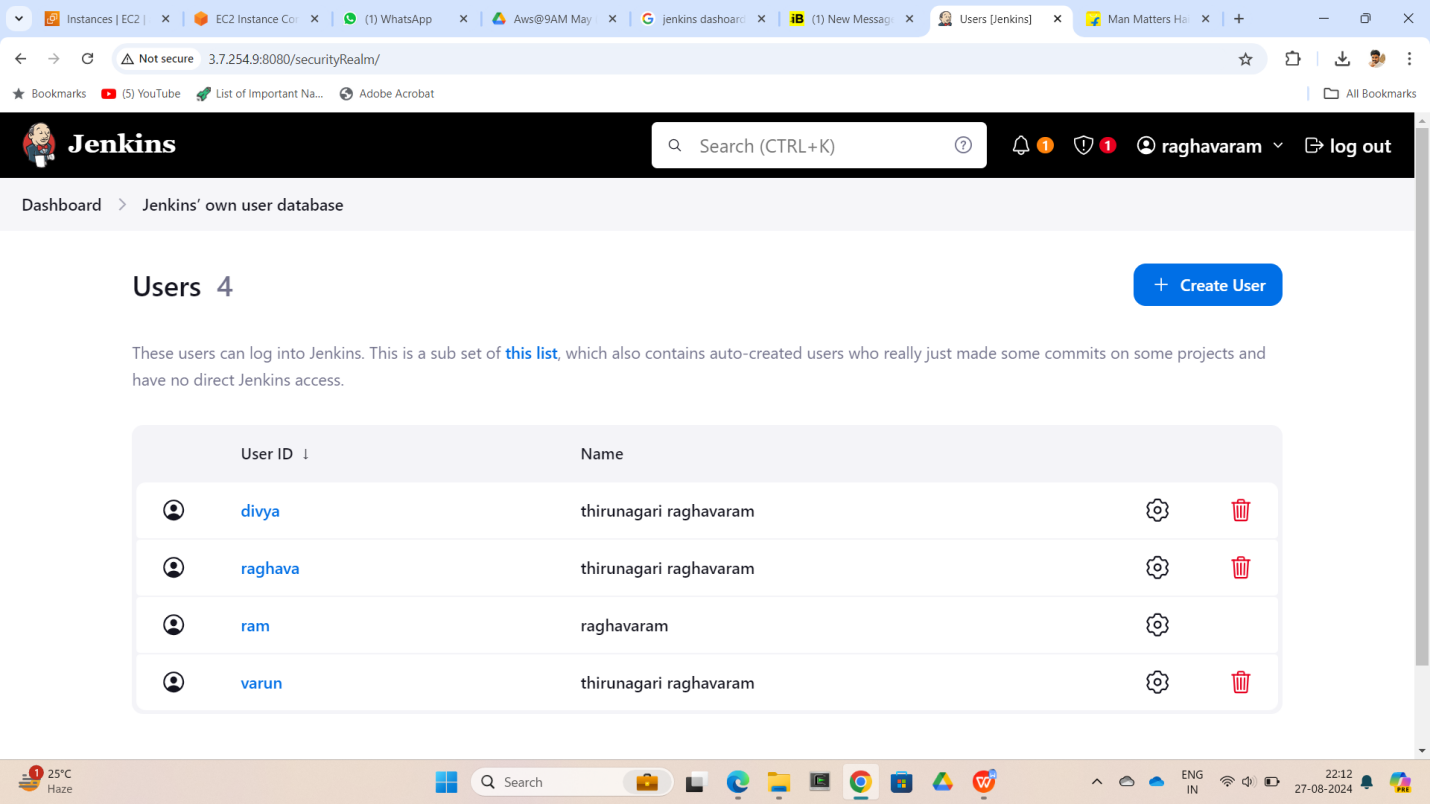
* Creating various users
* Installing required plugins

Assigning roles

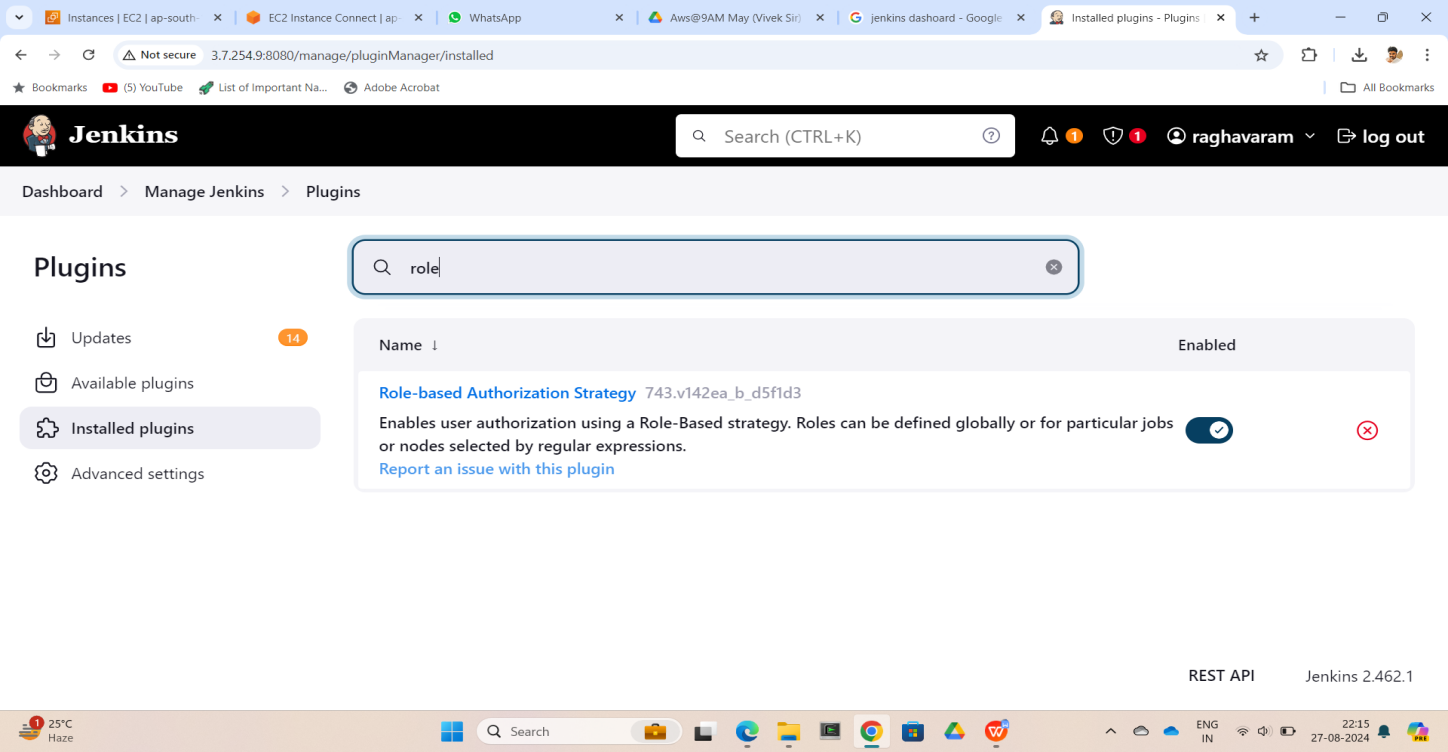
\*\*\* Manage Jenkins \*\*\*



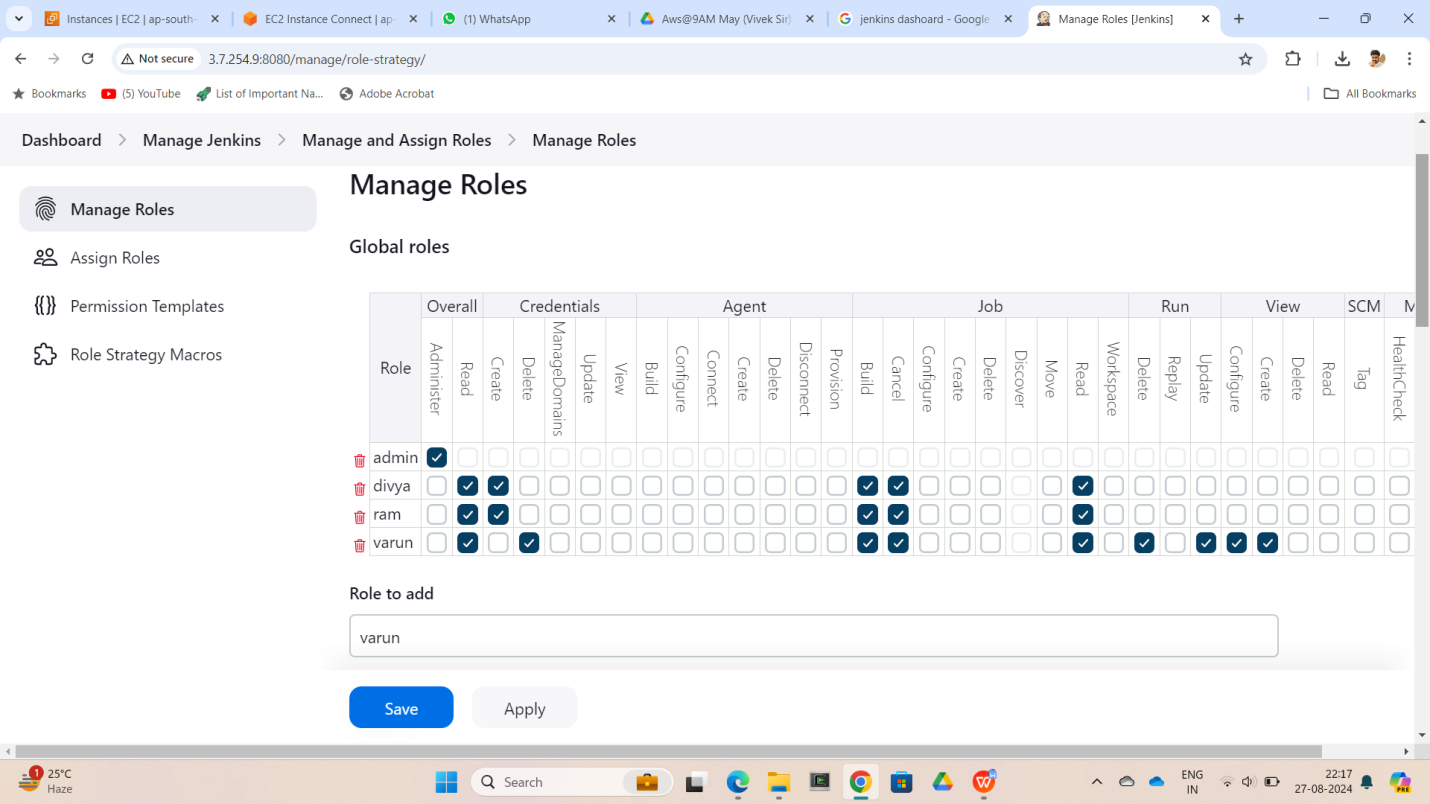
\*\*\*Users list\*\*\*



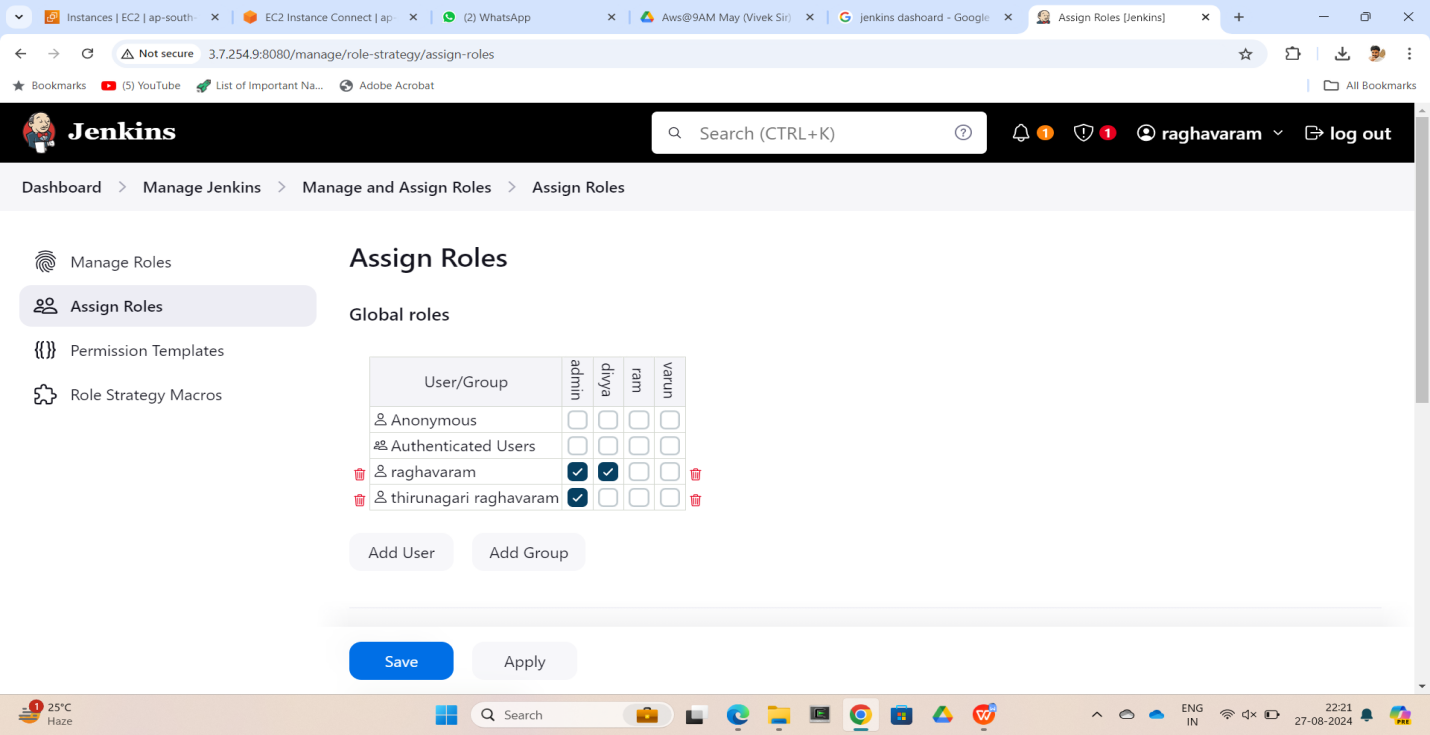
\*\*\*how to create users\*\*\*



\*\*\* Managing roles to user\*\*\*



\*\*\* Roles assigning to users\*\*\*

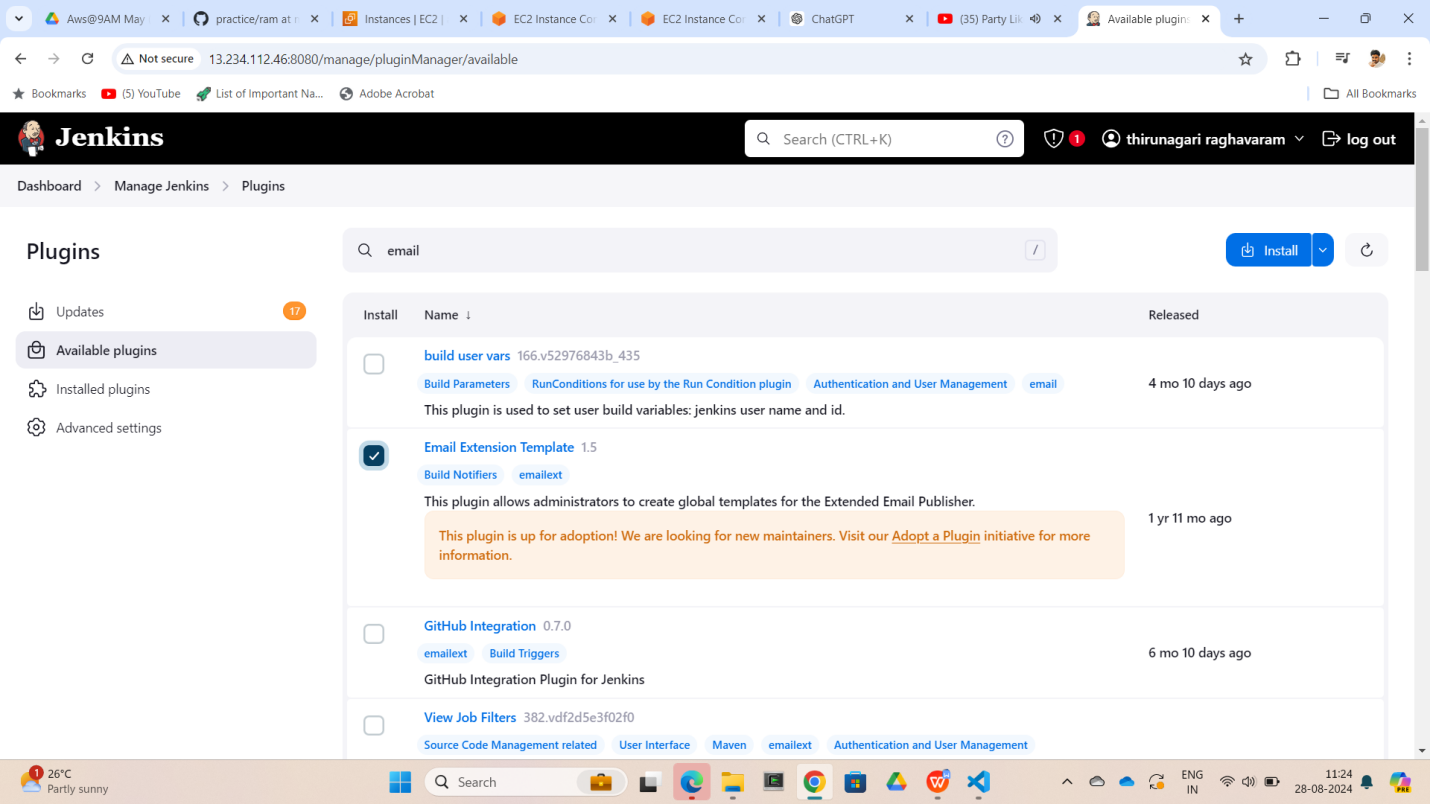


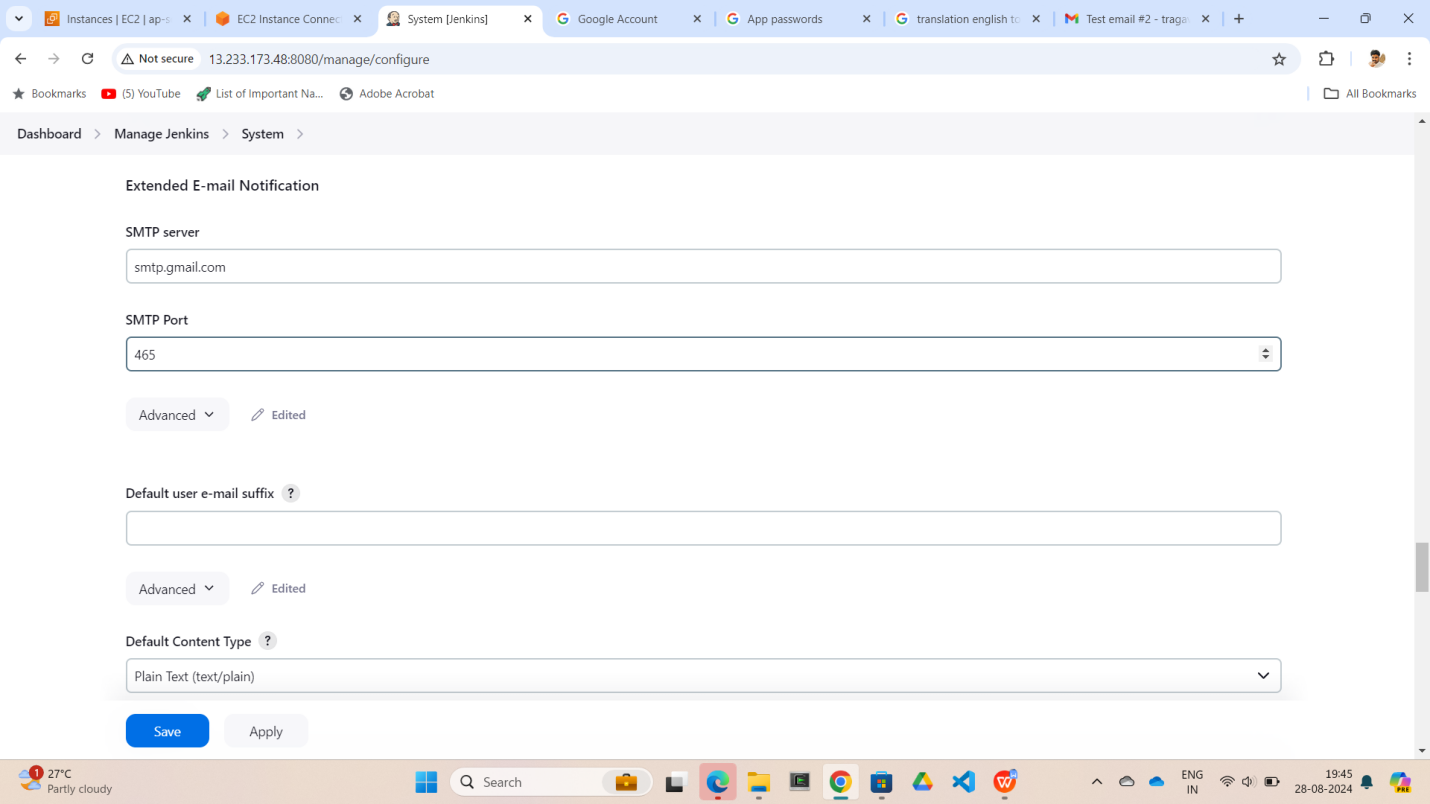
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

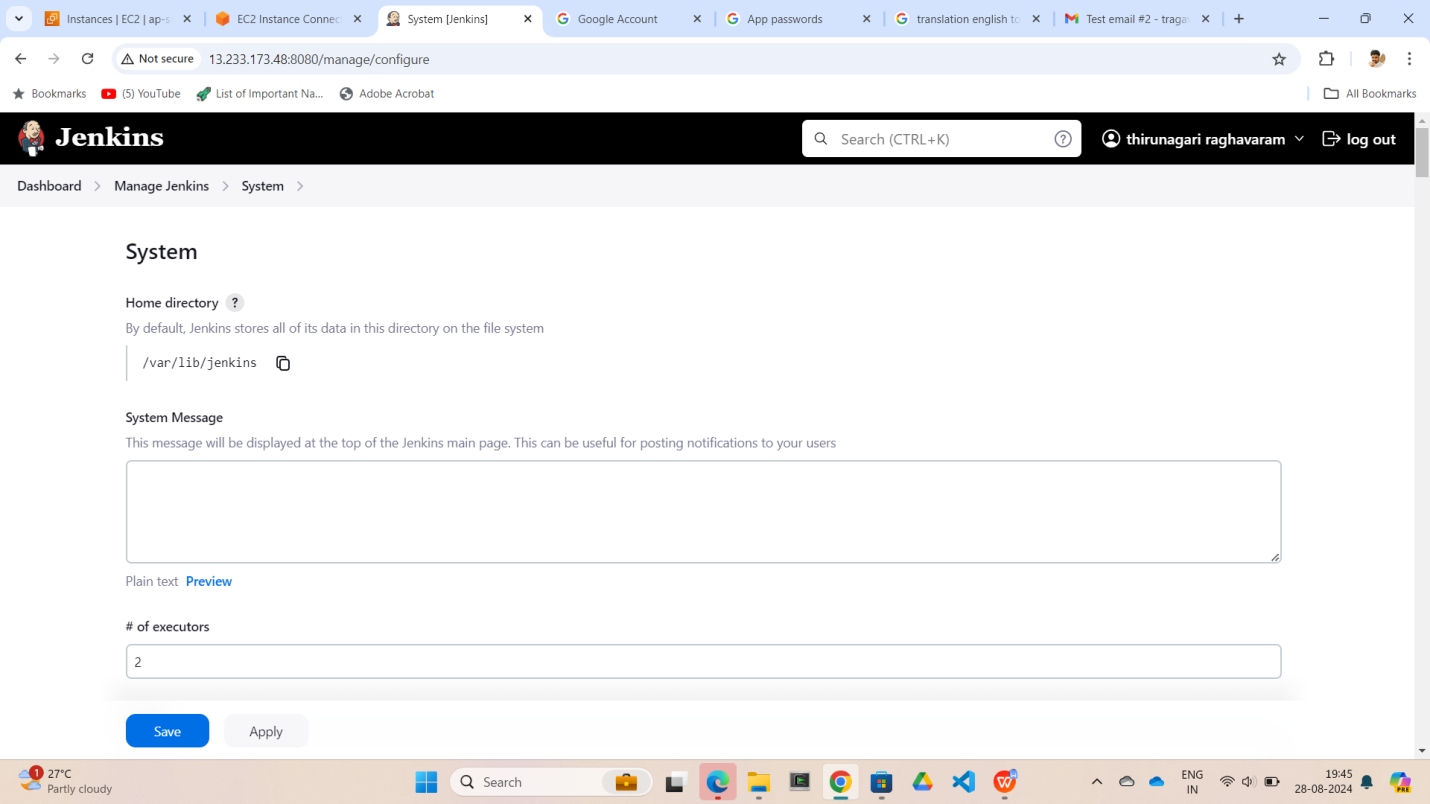
1. Jenkins with email

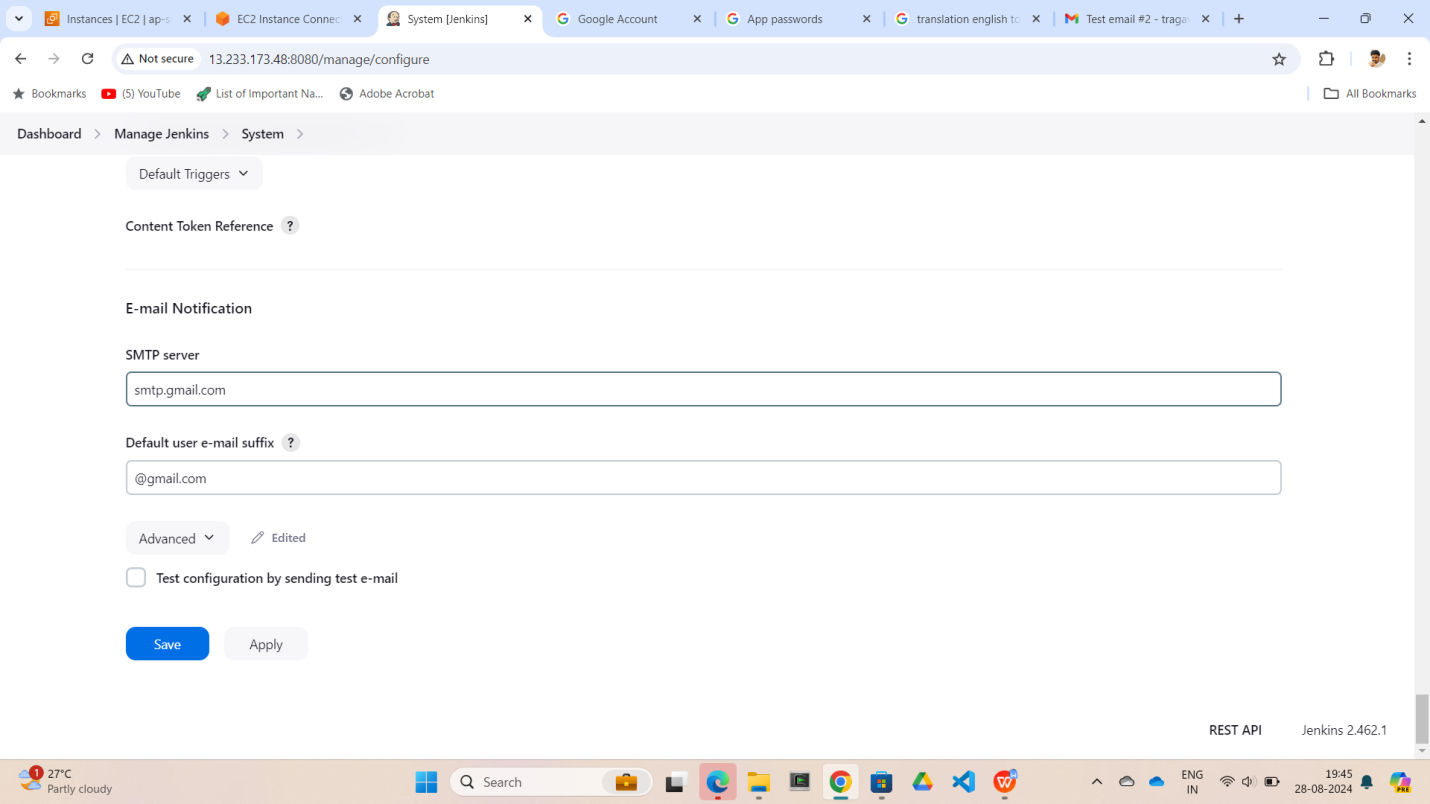
* Configuring jenkins to send mail
* Sending mail to multiple mail accounts

Hands-on: managejenkins >> go to plugins >> install email extensions >> next go to system manage >> (scroll down) >> go to extended email notification >> click n SMTP server >> write(smtp.gmail.com)---(you can search in google) >> write port 465 >> next credentials ADD > domain >> username (gmail.id) >> next go to google account then search and enable 2-step verification ,app password >> write app name jenkins then it displays password copy the password and paste at credentials password >> click On TLS ,SSL >> scroll-down email notification >> smtp server (smtp.gmail.com) >> @gmail.com >>write gmailid, jenkins password copy from google account >> click on SSL,TLS >>port 465>> next click on test configuration mail>>click on configuration mail it displays success >>then click save and apply



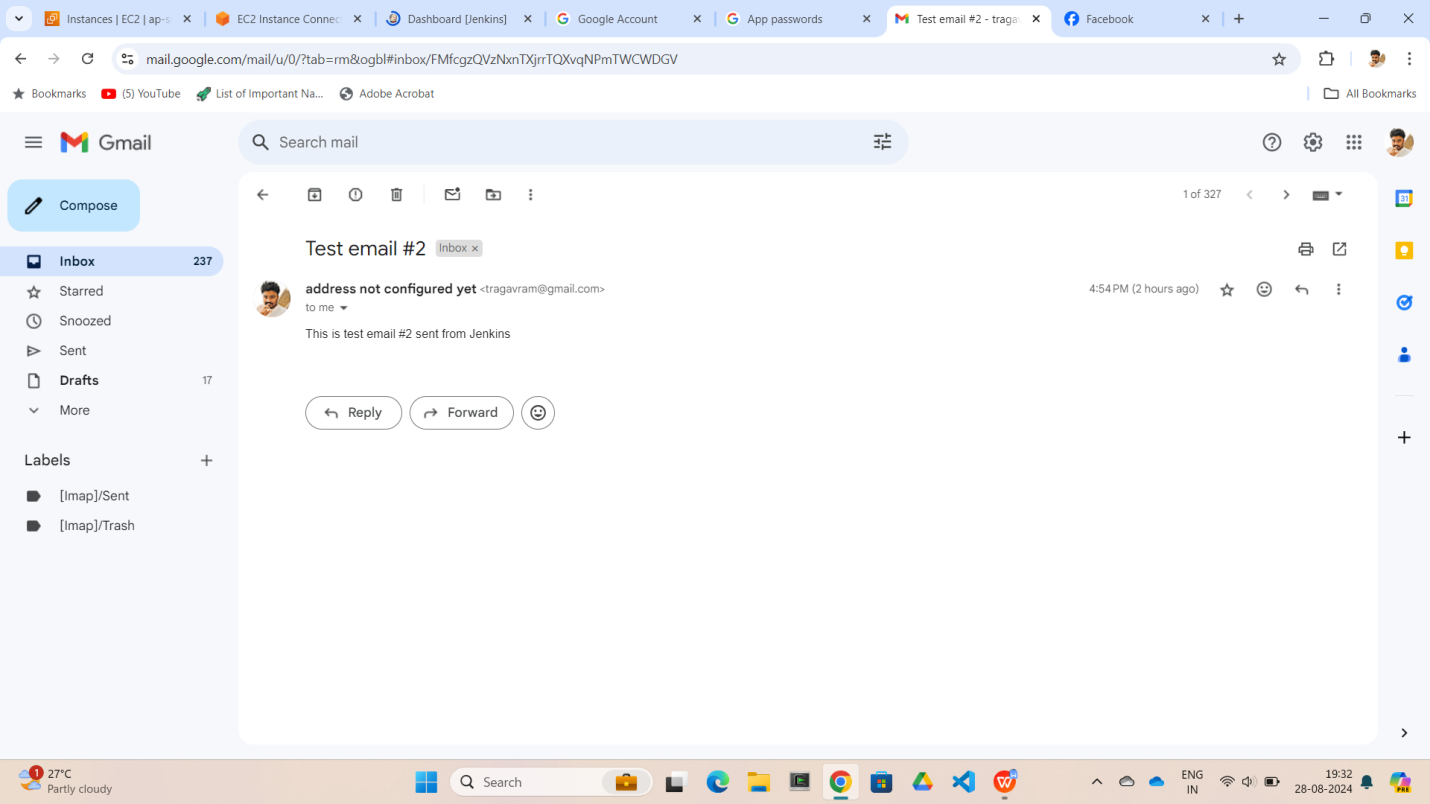






Final result after build :

mail successfully sent to gmail



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Trigger Builds & Environment Variables & Parameterised jobs

* Triggering jobs using url
* Remote Triggers
* Global & local variables in jenkins
* Parameterised jobs

Hands on : first create a new job >> configure the project >> click on the git link >> copy and paste the github repository link >> save and apply >> buildnow

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Upstream & Downstream jobs , Periodic Jobs

* Configuring upstream & downstream jobs in jenkins
* Configuring scheduled jobs in jenkins

Upstream jobs and Downstream jobs In jenkins

An upstream job is a configured project that triggers a project as part of its execution.

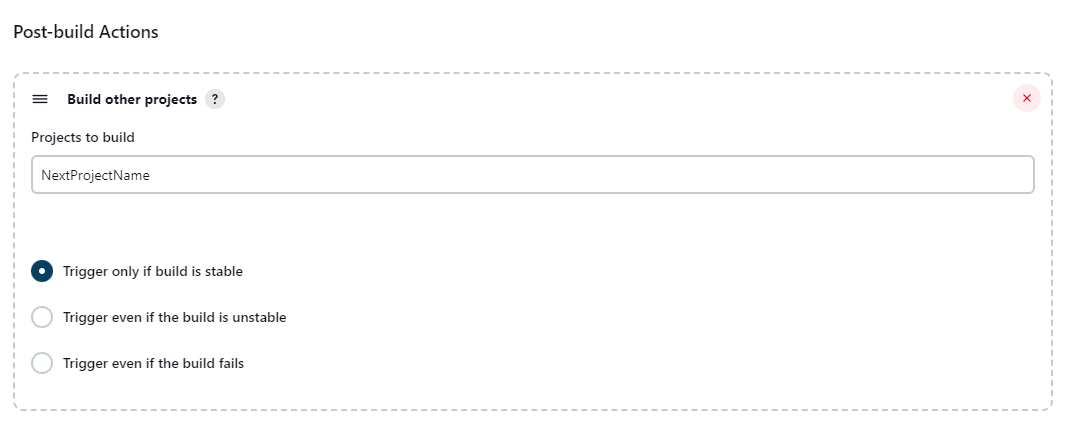
A downstream job is a configured project that is triggered as part of a execution of pipeline.

In simple words, upstream project triggers another project after build and downstream project triggers itself when particular project build completes

## Steps for Normal Configuration:

### Upstream Project:

Configure Project->Post-Build Action->Build Other Projects->Give Next Project Name and Select Trigger->Save



Upstream Project Config

### Downstream Project:

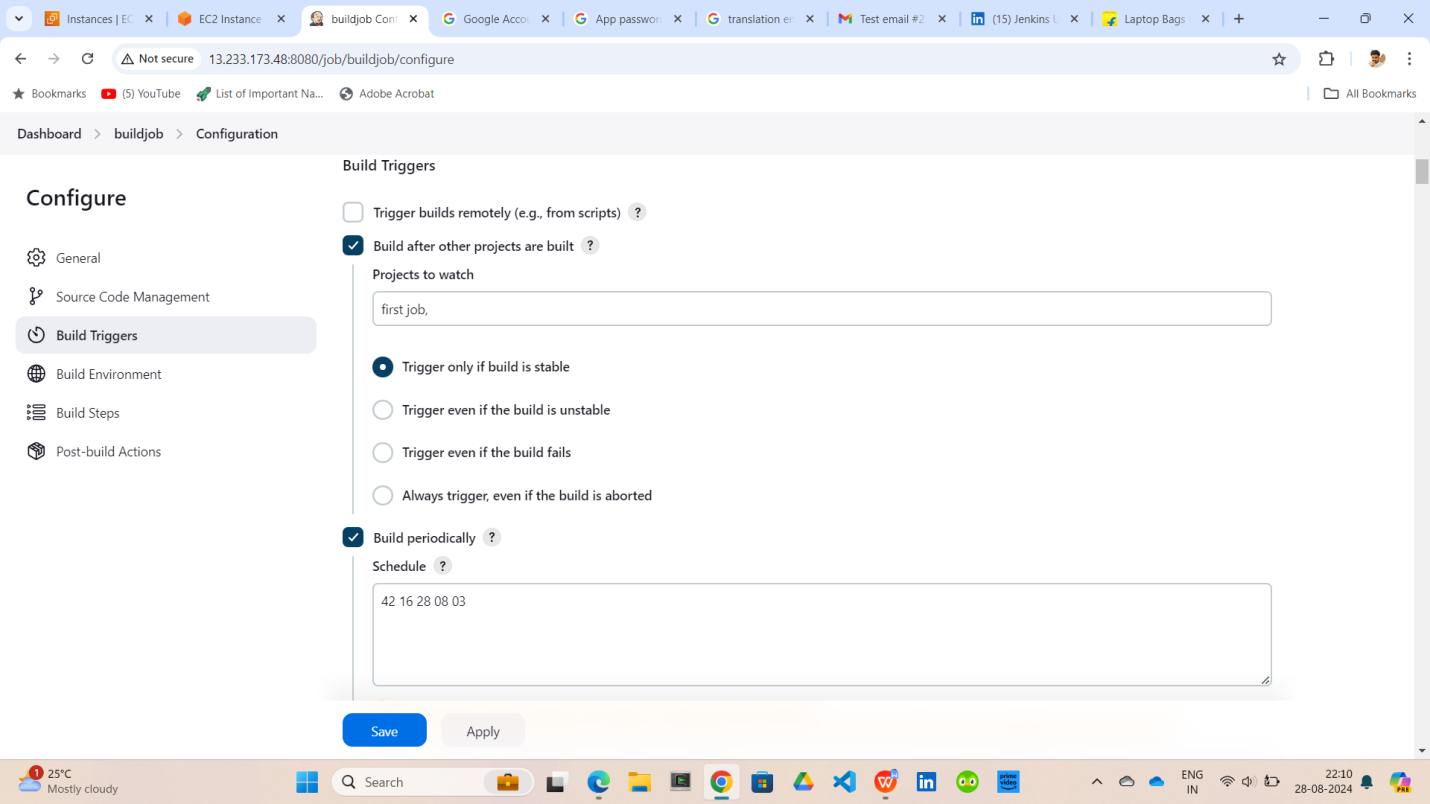
Configure Project->Build Triggers->Build after other projects are built->Give Previous Project Name and Select Trigger->Save



Downstream Project Config

Build periodically

Pattern : minute hour(0-23) date(0-31) month(1-12) day(0-7)

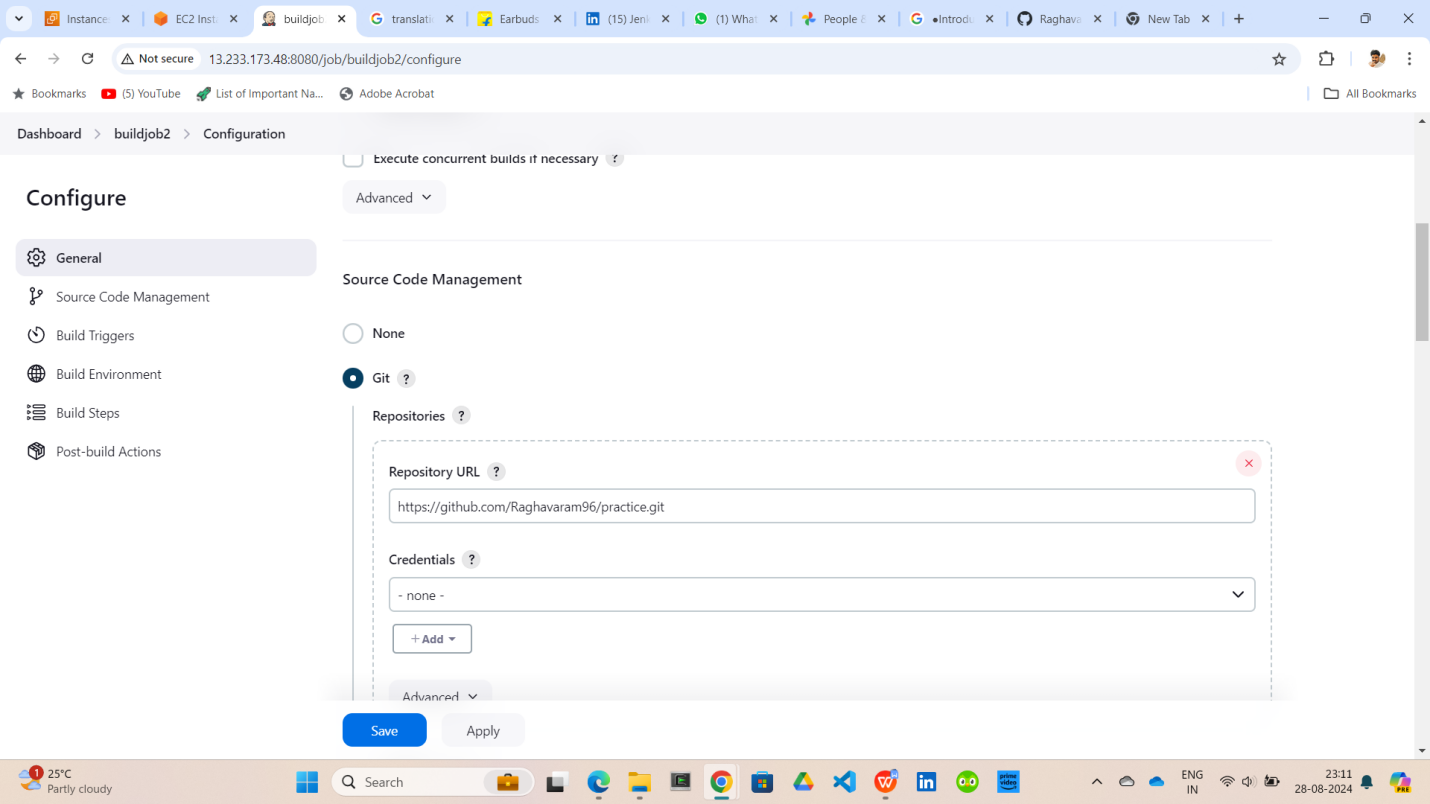


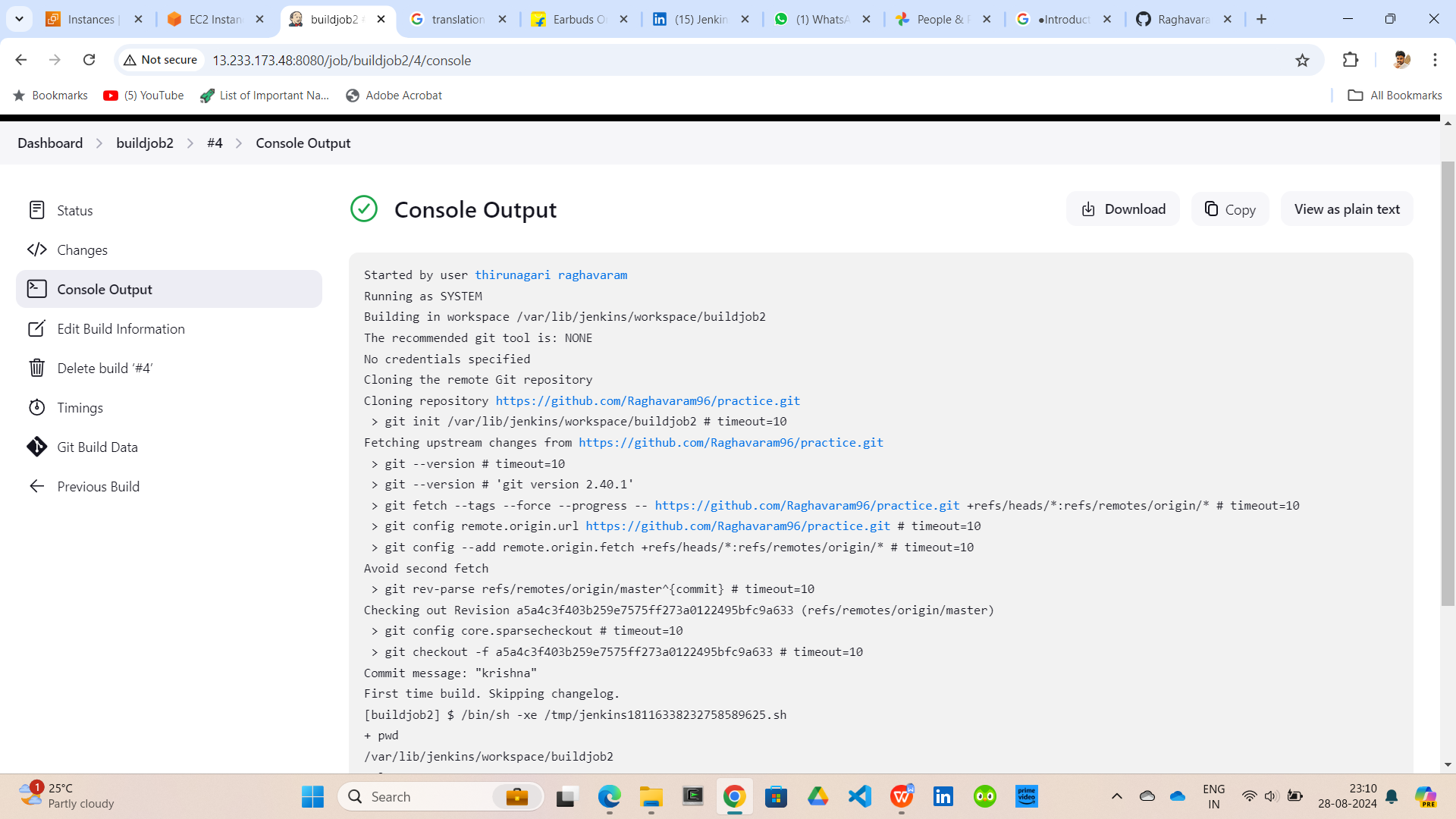
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Jenkins & Git

* Introduction to git plugin
* Git with Poll SCM
* Remote triggers with git

Hands-on : first install git into the instance (sudo yum install git -y ) , then open our Github account and copy the repository ink paste into the jenkins project then save & apply >> build now





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Jenkins & build pipeline plugin

* Creating simple pipelines with build pipeline plugins
* Adding steps in the pipeline

Hands-on: first create a new item and click on pipeline then save the job >> then go to plugins then install build pipeline (user interface) plugin and pipeline (stage view) >> then configure the project and go to stages click on to the hello-world yaml syntax

Edit the syntax then >> save and apply then build now the project

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Jenkins pipeline as code & jenkins file

* Introduction to jenkins file
* Writing first pipeline as a code

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Jenkins & Terraform pipeline Creating an ec2 instance with jenkins using terraform

Terraform

1. Introduction to IAAC

– Understanding IAAC concepts

– Installing Terraform on servers

– Setting up Visual Studio

1. Deploying Infrastructure with Terraform

– Authentication and Authorization

– Launching first VM through terraform

– Provider Tiers

– Creating a Github Repository with terraform

– Terraform Destroy

– Understanding Terraform state files

– Desired and Current States

1. Read, Generate, Modify Configurations

– Cross Resource Attributes  
 – Output Values

– Terraform Variables

1. Modules and Remote State Management

Understanding DRY principle

Implementing EC2 module with Terraform

Variables and Terraform Modules

Implementing Remote backend with Terraform s3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Introduction to IAAC

– Understanding IAAC concepts

– Installing Terraform on servers

– Setting up Visual Studio

IAAC --> Infrastructure As A code

Terraform :

Terraform is an infrastructure as a code **(IAC)** tool that allows you to build,change, and version infrastructure safely and efficiently.

In Infrastructure as Code **(IaaC),** the environment servers as distinct configurations or contexts within which infrastructure resources are provisioned, managed, and operated. Environment typically corresponds to the software development life-cycle stages: development, testing, staging, and production

**What is the basic definition of infrastructure?**

Infrastructure, or information technology infrastructure, is a collection of hardware, software, and networks that enable a company to run its applications and business. It's the foundation for a company's technology resources and operations.

**Here are some things to know about IT infrastructure:**

Components

IT infrastructure includes hardware, software, networks, facilities, and related services. Hardware includes physical components and devices, such as servers, storage systems, networking devices, graphics cards, hard disk drives, and RAM. Software includes operating systems, business applications, and containers.

***The three main components of IT infrastructure are hardware, software and networking.***

Security

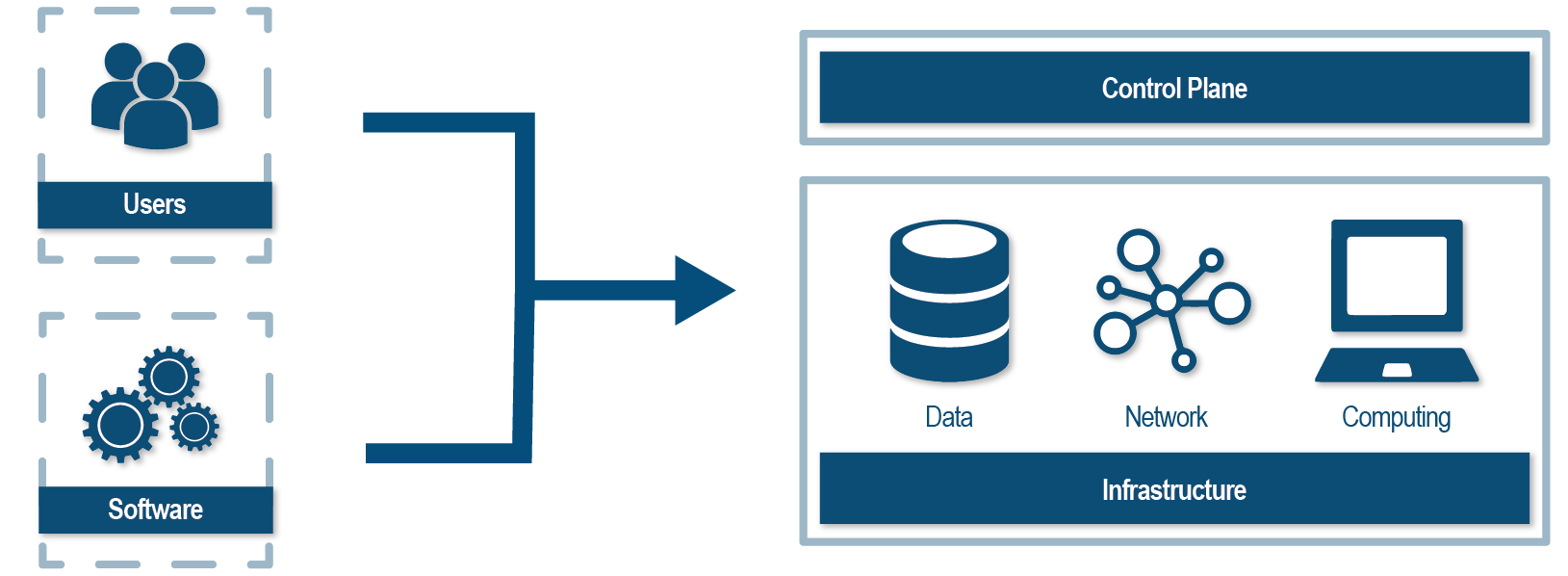
A robust IT infrastructure includes security measures like firewalls, encryption, and access controls to protect sensitive data.

Cloud infrastructure

Modern IT infrastructure includes a mix of cloud environments, on-premise data centers, and edge computing devices.

ITIL framework

The ITIL framework for IT service delivery and service management includes managing infrastructure as a core practice.



**Older version newer version**

Physical machine >> changed to virtual machine

Physical server >> changed to virtual server

Installing Terraform on servers

Browse in google >> terraform.io >> go to download >>select your operating system(linux) >> then click on amazon linux copy the code >> execute this commands into the ec2-instance>>(**sudo yum install -y yum-utils shadow-utils sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo**

**sudo yum -y install terraform**)>> check the terraform version **(terraform --version)** >> next create a folder (mkdir foldername) >> then enter into the folder (cd foldername) >> then again create a newfolder into the folder (mkdir folder2 >> create one file (touch filename) >> edit this file (vim filename >> in this file copy the code from visual code) >>go to visualcode studio copy the code and save the code(esc +shift + :wq!) >> if the code is correct or not you will check this command (**terraform validate**) >>

provider "AWS" {

  region     = "ap-south-1"

  access\_key = "AKIA6ODU673LOCMJTIYI"

  secret\_key = "a6Qg4bMTtzrIOYRFYBjrcXmX0n1xA2E0WL5mJbDo"

}

resource "AWS\_instance" "myfirstserver" {

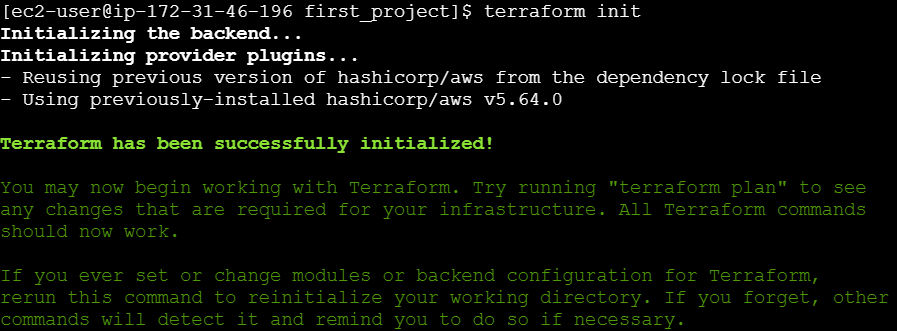
    ami = "ami-02b49a24cfb95941c"

    instance\_type = "t2.micro"

    key\_name = "versionit-class"

}

go to the instance apply this command (**terraform init**) >> then it will be display like this >>



After completing the initialization you have to plan the terraform with (**terraform plan**) with this command. >> After completing the plan you have to apply the terraform with (**terraform apply**) with this command.then it will shows like below.



Setting up Visual Studio

Install visual studio on your server (www.vs studio ) click on download and install >> after installing the visual studio code select folder or create a new folder for your work then do your works into that folder >>

1. Deploying Infrastructure with Terraform

Authentication and Authorization

Setup the authentication and authorization:

Authentication : it is a process of verifying who user is

Authorization : it is a process of verifying what they have access to

Access key

Access keys are long-term credentials for an IAM user or the AWS account root user. You can use access keys to sign programmatic requests to the AWS CLI or AWS API (directly or using the AWS SDK).

What is secret access key?

The secret access key is available only at the time you create it. If you lose your secret access key, you must delete the access key and create a new one. For more details, see Reset lost or forgotten passwords or access keys for AWS. You can have a maximum of two access keys per user.

How to give access key and secret key in Terraform?

Note that the usual and recommended way to authenticate to AWS when using Terraform is via the AWS CLI, rather than any of the provider options listed above. To do this, first, install the AWS CLI, then type AWS configure . You can then enter your access key ID, secret access key, and default region.

access key and secret key in terraform

Go to the IAM user >> click on users >> click on which user you want >> next go to security credentials >> then click on create access key >> in the use case you click on (CLI) >> click on understand >>give a template name >> save the details

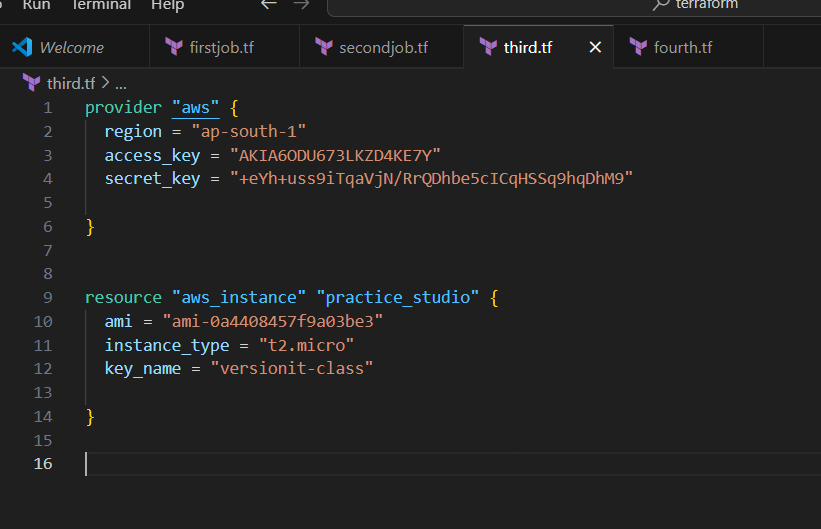
>> copy the access key and security key into the vs studio

– Launching first VM through terraform

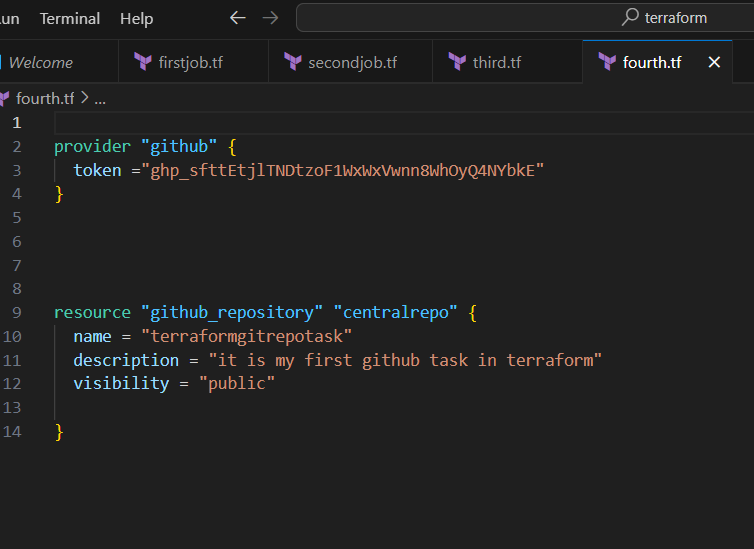
– Provider Tiers

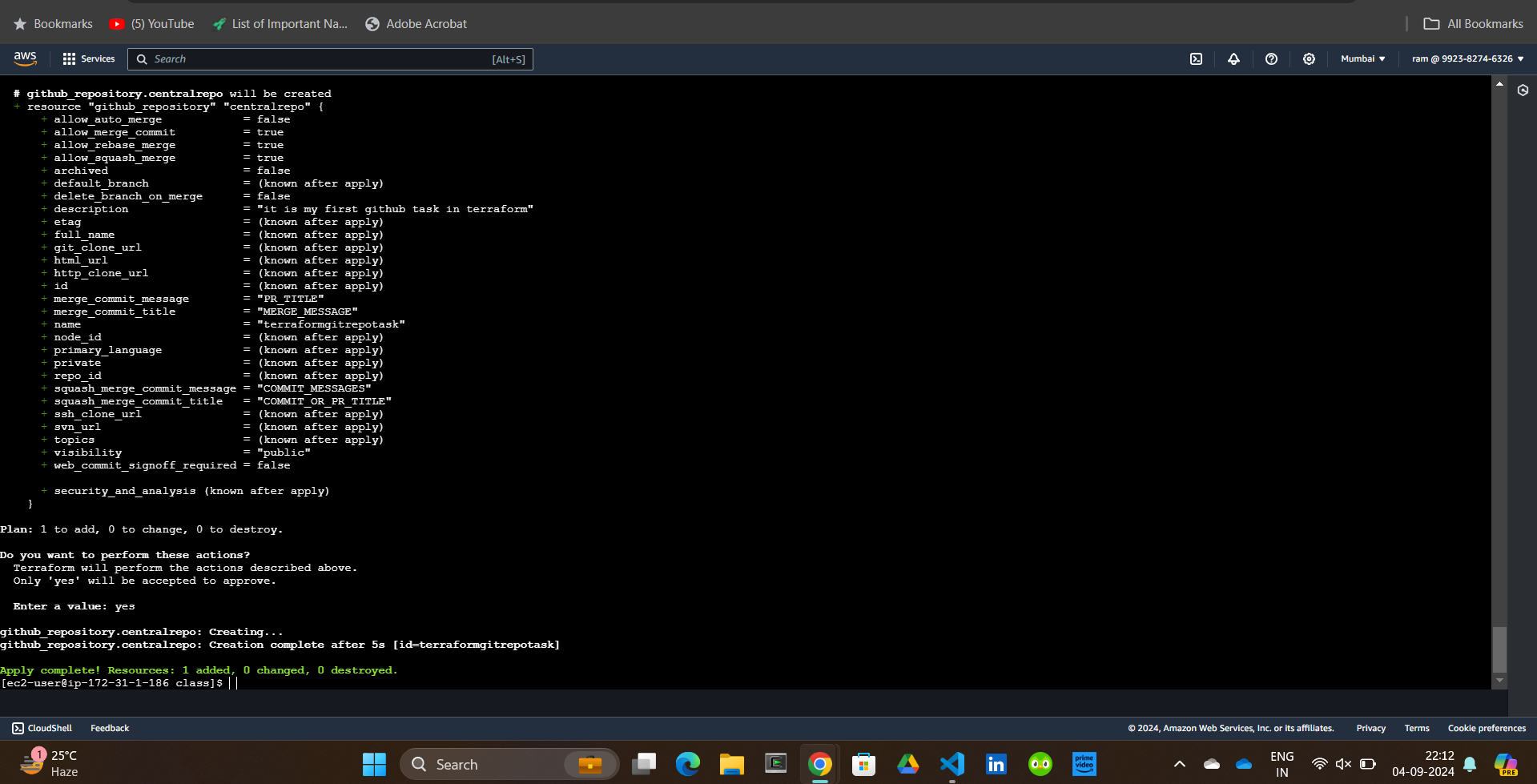
– Creating a Github Repository with terraform

Search Terraform integrations >> github >> copy the code >>into VS studio >> copy the code into any newly created file >> terraform init >> terraform plan >> terraform apply >>

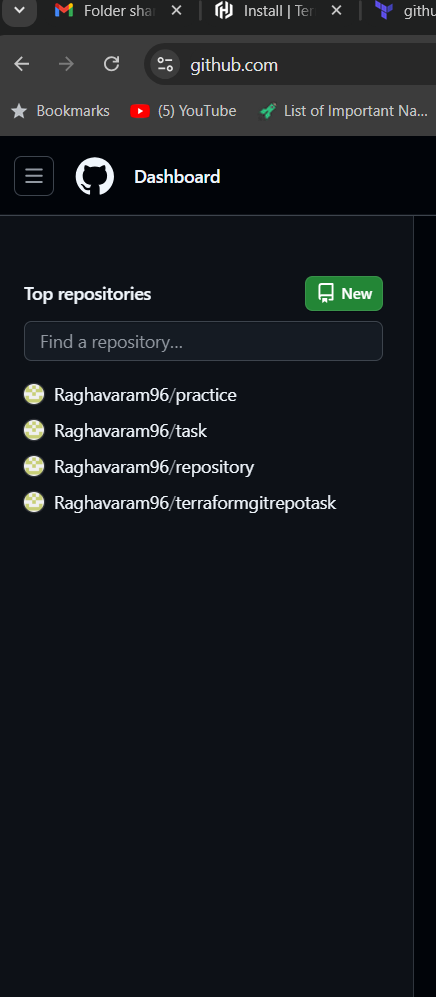


here you will create a new token into the git hub >> go to settings >> developer settings >> create new token >> name to token >> save the token >> copy the token link into the VS studio >>





After connect the token link it will show like this ,it creates a new repository



Terraform Destroy

Terraform destroy is a command used in Terraform, an Infrastructure as Code (IaC) tool, to delete all the resources defined in your Terraform configuration files. It effectively tears down the infrastructure you've created, allowing you to clean up your environment.

**How to Use terraform destroy**

1. **Navigate to Your Project Directory:** Open your terminal and change to the directory where your Terraform configuration files are located.
2. **Run the Command:**

**terraform destroy**

1. **Confirm the Action:** Terraform will show you a list of resources that will be destroyed and prompt you for confirmation. You can bypass the prompt by using the -auto-approve flag:

**terraform destroy -auto-approve**

Important Considerations

**Data Loss:** This action is irreversible; once you destroy the resources, any data stored within them will typically be lost unless you have backups.

**Dependencies:** Be aware of dependencies between resources. Destroying one resource may affect others.

**State File:** Terraform keeps track of resources in a state file. Running terraform destroy updates this state file accordingly.

Understanding Terraform state files

The terraform state file, by default, is named terraform. tfstate and is held in the same directory where Terraform is run. It is created after running terraform apply . The actual content of this file is a JSON formatted mapping of the resources defined in the configuration and those that exist in your infrastructure.

Desired and Current States

In Terraform, the concepts of desired state and current state are crucial for managing infrastructure as code. Here’s a breakdown of both:

Desired State:

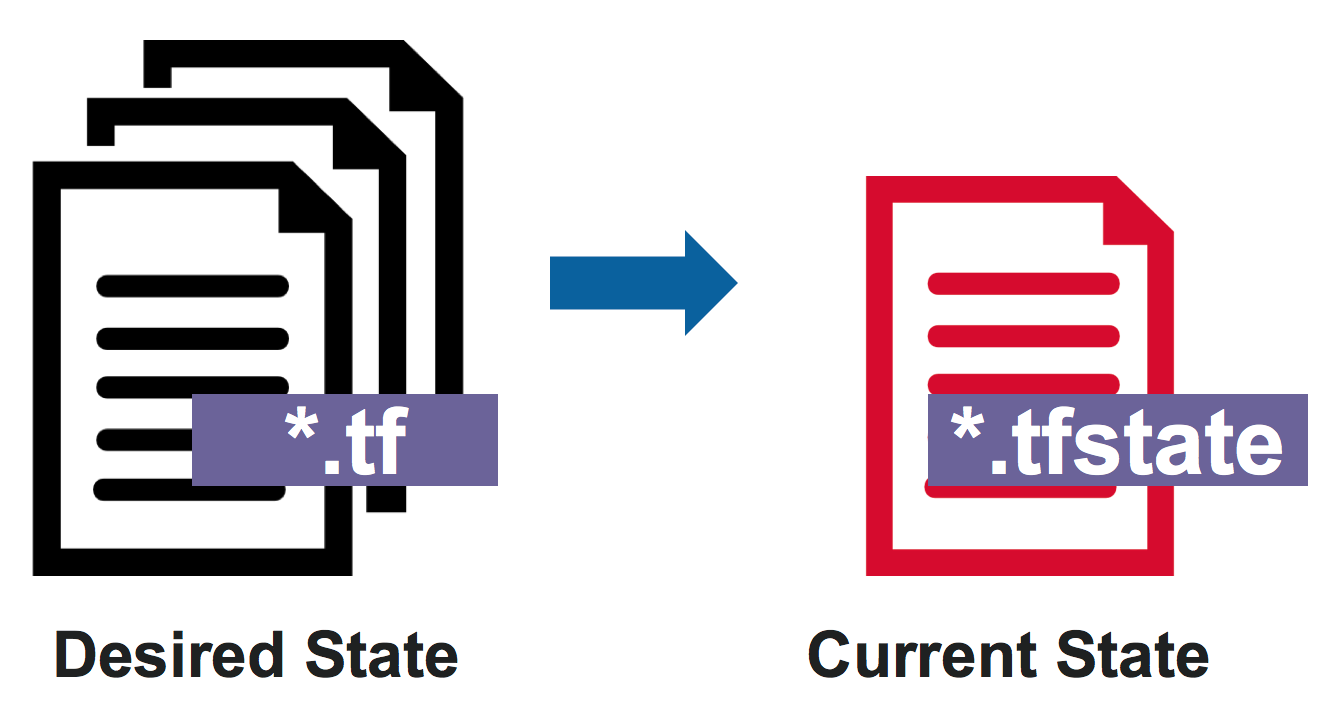
**Definition:** The desired state refers to the configuration you define in your Terraform files (usually **.tf files)**. It describes what your infrastructure should look like, including resources, their properties, and any dependencies.

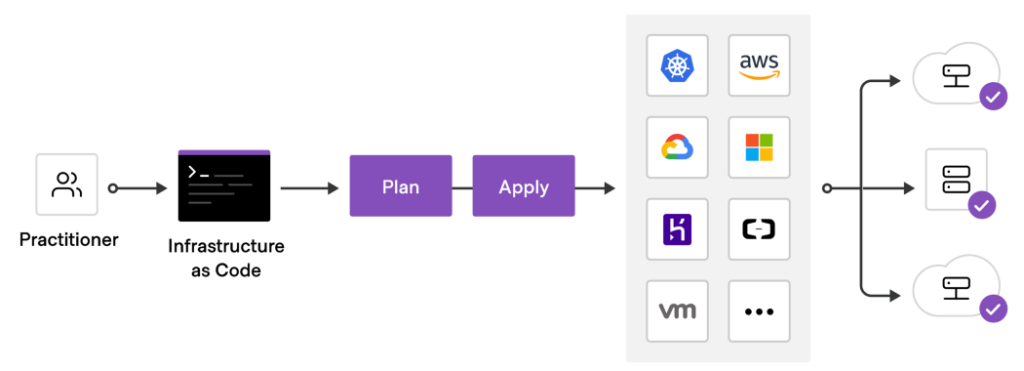
**Manifested by:** This state is expressed through the Terraform code you write. For example, if you specify an AWS EC2 instance in your configuration, that becomes part of your desired state.

Current State:

**Definition:** The current state reflects the actual state of your infrastructure at a given moment in time. This state is captured in the Terraform state file (usually named **terraform.tfstate)**.

**Managed by:** Terraform automatically updates this file whenever you apply changes. It contains details about resources that have been created, modified, or destroyed.





3.Read, Generate, Modify Configurations

– Cross Resource Attributes  
 – Output Values

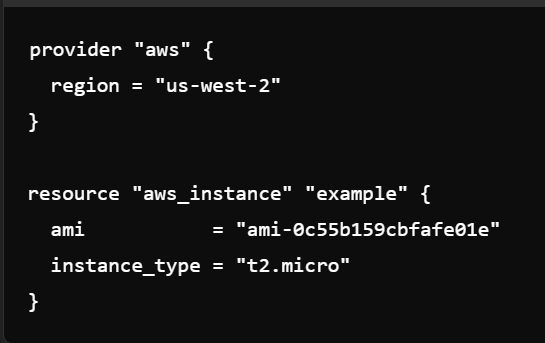
– Terraform Variables

Read, Generate, Modify Configurations

Terraform is a powerful tool for managing infrastructure as code. You can read, generate, and modify configurations in Terraform using its HashiCorp Configuration Language (HCL). Below are some examples of how you can achieve each of these tasks.

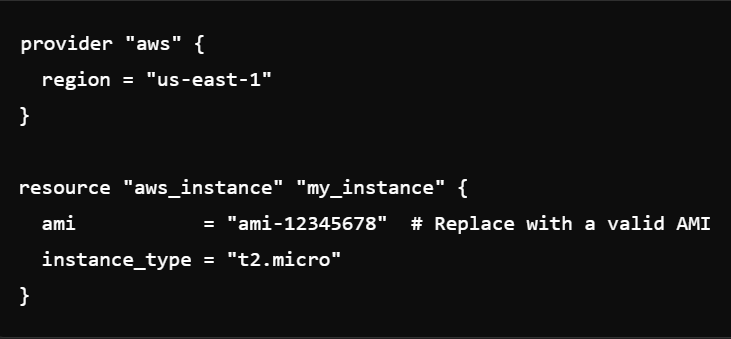
**1. Reading Terraform Configurations**

To read and understand existing Terraform configurations, you typically look at .tf files. For example:



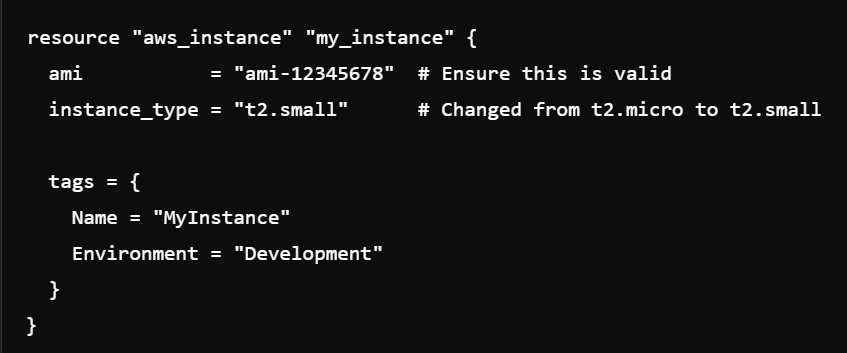
**2. Generating Terraform Configurations**

You can create new configurations from scratch or by modifying existing ones. For instance, generating a new EC2 instance:



**3. Modifying Terraform Configurations**

You can easily modify existing configurations. Here’s how you might change the instance type and add tags:



Cross Resource Attributes

Cross-resource attributes in Terraform allow you to reference attributes from one resource in another resource's configuration. This is crucial for establishing relationships and dependencies between resources. Here's how to work with cross-resource attributes effectively.  
Output Values

Output values in Terraform are a way to extract and display information from your Terraform configurations after they have been applied. Outputs are particularly useful for sharing information between different modules or simply to present the results of your infrastructure deployment.

Terraform Variables

Terraform variables allow you to parameterize your Terraform configurations, making them more flexible and reusable. You can define variables for different purposes, such as to customize resource attributes, make configurations environment-specific, or to simplify your code.

1. Modules and Remote State Management

Understanding DRY principle

Implementing EC2 module with Terraform

Variables and Terraform Modules

Implementing Remote backend with Terraform s3

**Ansible**

1. Introduction

* How ansible works
* Setting up ansible

1. YAML and INI files

* yaml 101
* yaml challenge
* inventory files 101
* ini challenge
* writing AWS inventory files

1. Playbooks

* Understanding the documentation
* Creating your first playbook
* Running playbooks

1. Services , Handlers & Shell

* Using the service module
* Understanding handlers
* Creating handlers
* Using shell & debug modules

1. Ansible Variables

* Introduction
* Understanding Jinja 2 templating

**-------------------------------------------------------------------------**

1. Introduction

How ansible works

Setting up ansible

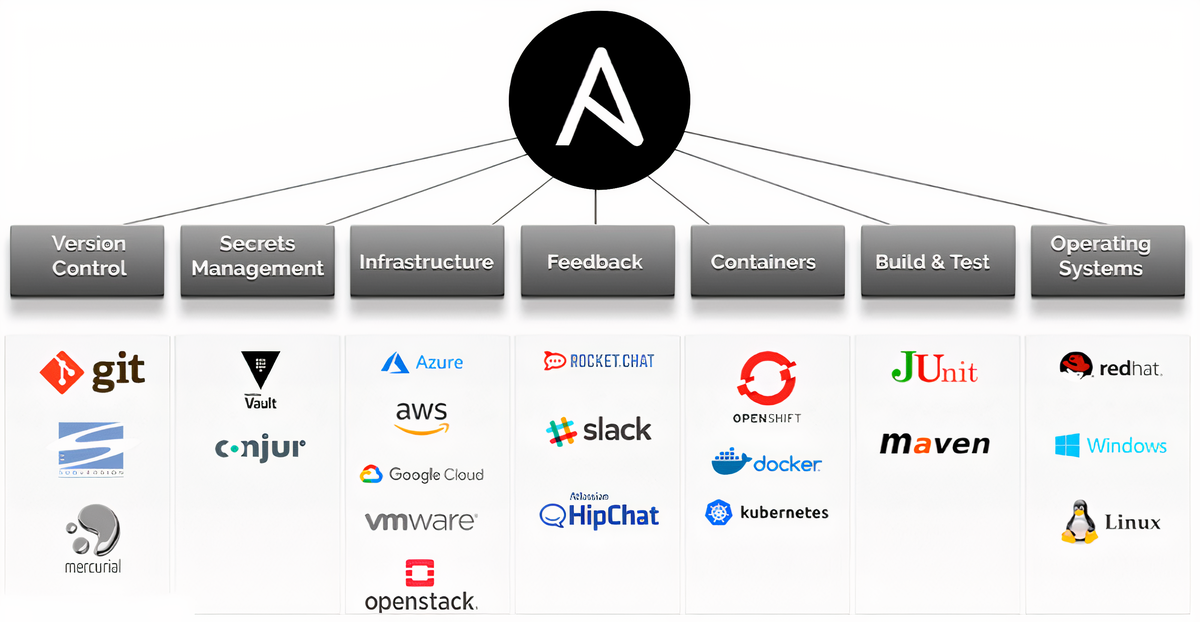
Ansible

Ansible is open source software that automates software provisioning, configuration management, and application deployment. Ansible connects via SSH, remote Power-Shell or via other remote APIs.

(or)

Ansible is an open-source automation tool that simplifies IT tasks like configuration management, application deployment, and task automation. It uses a simple, declarative language (YAML) to define automation tasks in playbooks, making it accessible even for those new to programming.

* Ansible is IT automation, configuration management and provisioning tools
* Ansible is a configuration management, deployment & orchestration tool
* It is a **“push-bashed”** configuration management tool
* It automates your entire IT infrastructure by providing large productivity gains
* It uses “playbook” to deploy, manage, build, test and configure anything from full server environment to websites to custom complied source code for applictions.



Key Features:

Agentless: Ansible doesn’t require any agents on the target machines. It connects over SSH (or WinRM for Windows).

Playbooks: Ansible uses YAML files called playbooks to define automation tasks in a human-readable format.

Modules: Ansible has a rich library of modules that can perform various tasks, such as managing packages, files, services, and more.

Inventory: Ansible manages the servers it controls through an inventory file, which can be static or dynamic.

Idempotency: Ansible ensures that repeated executions of playbooks yield the same result, making it safe to run tasks multiple times.

Extensibility: Ansible can be extended with custom modules and plugins to fit specific needs.

**Features of Ansible**

* Simple to install and setup
* Very easy to learn
* No need of any agent or client software to manage the node
* Capabilities to model complex IT workflow and orchestrate you entire IT infrastructure
* Extensible with models written in any programming language
* Build on the top of Python and hence provides lot of Python’s functionality
* Uses SSH for secure connections
* Follows push bashed architecture for sending configurations.

Orchestration:

**Definition:** Orchestration is the coordination of multiple automated processes or tasks to work together as a unified system. It ensures that different components interact effectively and efficiently.

**Focus:** It emphasizes the integration and management of multiple automated tasks, ensuring they follow a sequence and interact properly.

**Scope:** Orchestration manages workflows, scheduling, and dependencies among various automated processes, often across different systems.

Automation

**Definition:** Automation refers to the use of technology to perform tasks with minimal human intervention. It involves creating processes that can execute specific functions automatically.

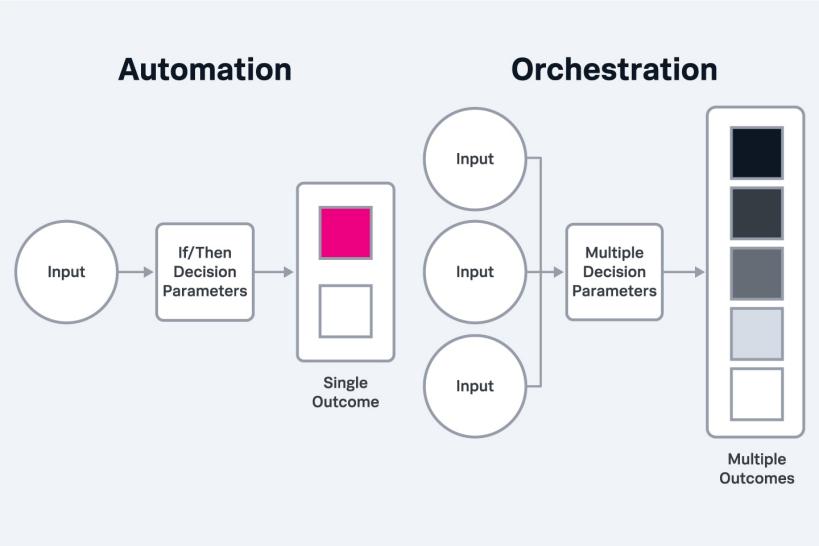
**Focus:** It typically focuses on individual tasks or functions. For example, automating data entry or software testing.

**Scope:** Automation can be simple, like scripts that run tasks, or complex, like robotic process automation (RPA) systems.

Key Differences (orchestration vs automation)

**Level of Coordination:** Automation handles individual tasks, while orchestration coordinates multiple automated tasks to achieve broader objectives.

**Complexity:** Orchestration typically involves more complexity because it deals with interdependencies and interactions between various processes.

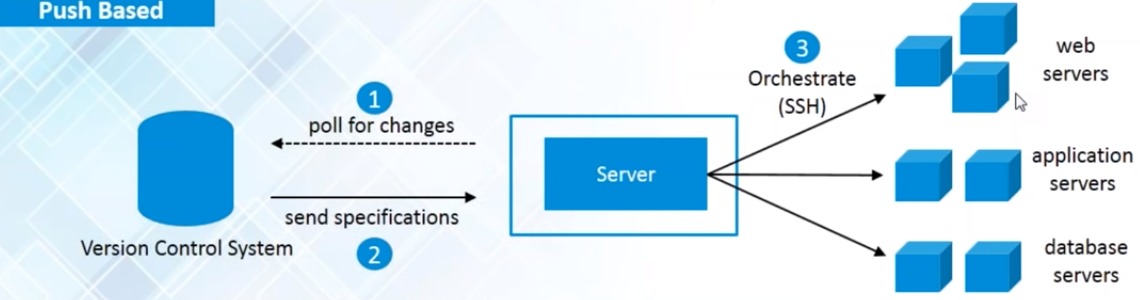


Push-Based And Pull-Based Tools

**Push-based tool**

Tool like Puppet, Chef and Ansible are push based tool

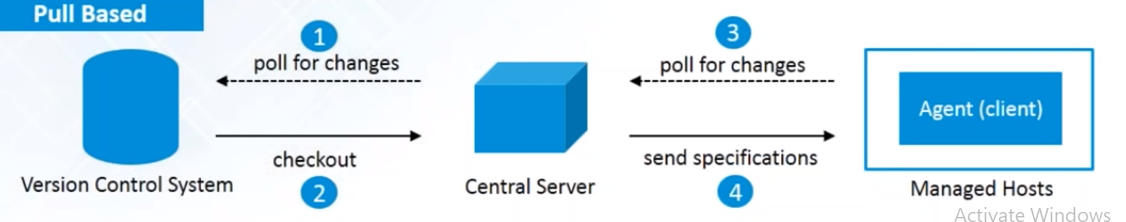
Central server pushes the configuration information on target servers



**Pull-Based Tools**

Tool like Puppet and Chef are pull based tools

Agent on the sever periodically checks for the configuration information from central server(Master)



Basic Components:

Inventory: A file that lists the hosts or machines to manage. It can be in INI or YAML format.

Playbook: A YAML file where you define tasks and roles. Each playbook consists of plays that map hosts to tasks.

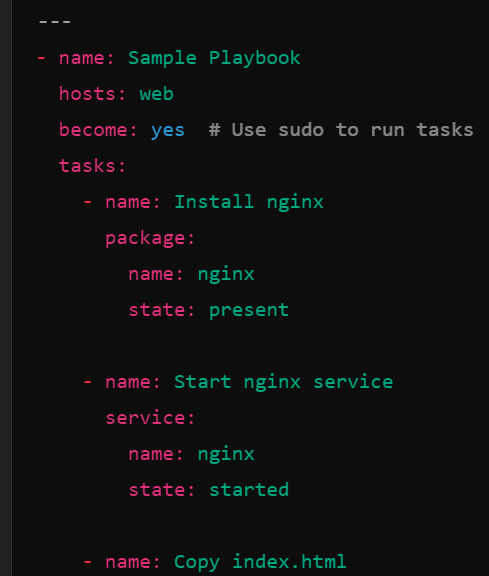
Roles: A way to organize playbooks into reusable components. Roles can include variables, tasks, handlers, and files.

Tasks: Individual actions defined in a playbook, executed in order.

Ansible playbooks

Ansible playbooks are YAML files that define a series of tasks to be executed on specified hosts. They allow you to automate complex workflows by organizing your tasks in a readable and structured way.

Basic structure of Ansible playbook



Key Components of a Playbook

1. ---: This signifies the beginning of a YAML document.
2. - name:: A description of the playbook or individual tasks.
3. hosts:: Specifies the group of hosts on which the playbook will run. This can refer to groups defined in your inventory file.
4. become:: Allows you to run tasks with elevated privileges (similar to using sudo).
5. tasks:: A list of tasks to be executed. Each task consists of:

* name:: A description of what the task does.
* Module name: Indicates what action to perform (e.g., package, service, copy, etc.).
* Parameters: Specific parameters for the module, like name, state, src, dest, etc

Installation of Ansible:

Before installing ansible you have to install python in your instance.

**Hands-on:**

**Login to your instance >> sudo yum install python -y >> sudo yum install ansible -y >>**

2.YAML and INI files

* yaml 101
* yaml challenge
* inventory files 101
* ini challenge
* writing AWS inventory files

YAML:**Yet Another Markup Language**

YAML (YAML Ain't Markup Language) is a human-readable data serialization format often used for configuration files and data exchange between languages with different data structures. Here are some key features and uses of YAML:

Key Features

* **Human-Readable:** Designed to be easy to read and write, making it accessible for both humans and machines.
* **Hierarchical Structure:** Uses indentation to represent nested data structures (like objects and arrays) in a clear manner.
* **Data Types:** Supports various data types, including scalars (strings, integers, booleans), lists, and dictionaries (hashes).
* **Comments:** Allows comments with the # symbol, making it easy to annotate configurations.
* **Multiline Strings:** Supports multiline strings using | (literal block) or > (folded block).

Common Uses

**Configuration Files:** Frequently used in applications (e.g., Docker, Kubernetes, CI/CD tools) for configuration settings.

**Data Serialization:** Often used for data exchange in APIs, especially when the data structure is complex.

**Infrastructure as Code:** Used in tools like Ansible and Terraform for defining infrastructure configurations.

Advantages

**Simplicity:** Easier to read and write compared to JSON or XML.

**Flexible:** Can represent complex data structures with ease.

**Widely Supported:** Many programming languages have libraries for parsing and generating YAML.

Disadvantages

**Indentation Sensitivity:** Requires consistent indentation, which can lead to errors if not managed properly.

**Limited Data Types:** While it supports various data types, some programming languages might not directly map to YAML types.

Basic Syntax of YAML:

1. Key-Value Pairs

key: value

1. Nested Structures Use indentation (typically 2 spaces) to represent hierarchy.

parent:

child1: value1

child2: value2

1. Lists Use hyphens (-) for list items.

fruits:

- apple

- banana

- cherry

1. Dictionaries Dictionaries can contain key-value pairs.

person:

name: John

age: 30

email: john@example.com

1. Multiline Strings Use | for literal blocks (maintains line breaks) and > for folded blocks (converts newlines to spaces).

description: |

This is a multiline

string that retains

line breaks.summary: >

This is a folded

string that will be

a single line.

1. Comments Comments start with #.

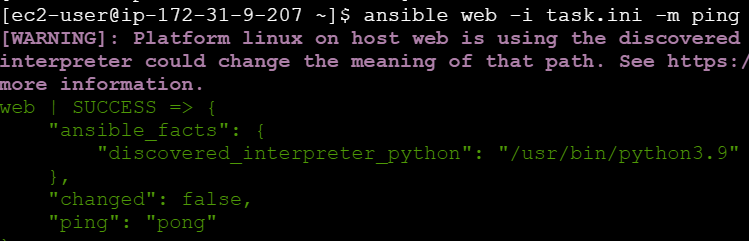
# This is a comment

key: value # Inline comment

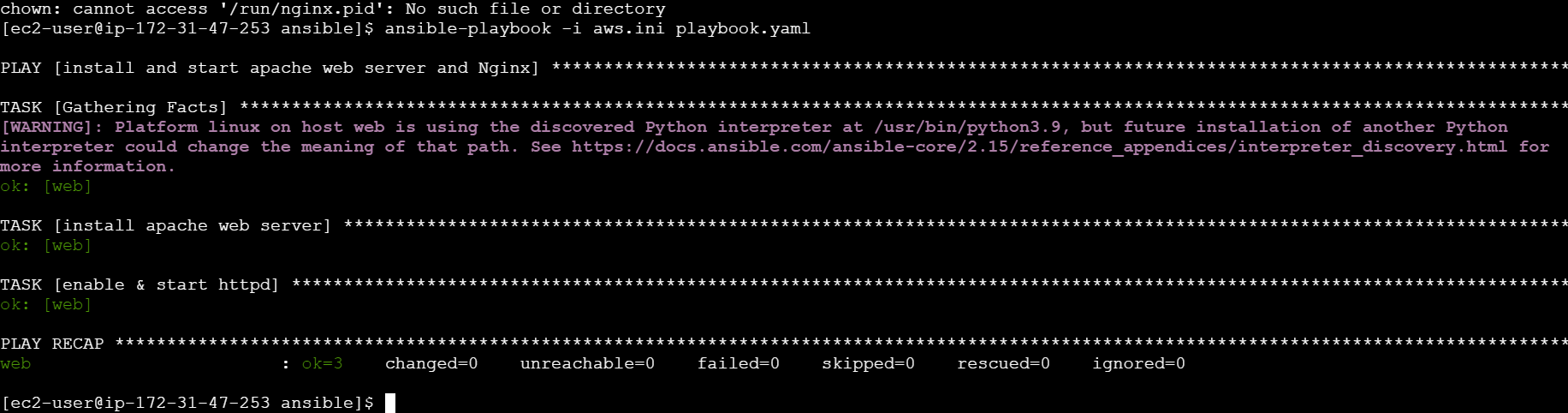
YAML hands-on:

Connect with your instance >> install python **(sudo yum install python -y)** >> install ansible **(sudo yum install ansible -y)** >> create a folder **(mkdir ansible)** >> **cd ansible** >> **touch keyname.pem** file >> **vim keyname.pem** >> give permissions to that .pem file **chmod -v 400 keyname.pem** >> **touch aws.ini** >> **vim aws.ini** >> write the **(web ansible\_host="13.201.42.41" ansible\_port="22" ansible\_user="ec2-user" ansible\_ssh\_private\_key\_file="/home/ec2-user/ansible/VIT.pem")** >> check for ping message **(ansible web -I aws.ini -m ping)** >> create a playbook .yaml file >> **touch playbook.yaml** >> **vim playbook.yaml** >> insert code >> **ansible-playbook -I aws.ini playbook.yaml**

**Check for ping pong message:**

****

**Installing httpd with the help of ansible playbook yaml file:**

****

INI files

In Ansible, INI files can be used for inventory management, where you define hosts and groups in a structured format. File extension name should be .ini (task.ini)

**Structure of INI Inventory Files**

An INI-style inventory file consists of sections that define groups of hosts. Each section is prefixed with a group name in square brackets. Below is a simple example:



In this image:

(1,2,3) lines indicates --> individual hosts.

(6,7,10,11) lines indicates --> grouping servers

(14,15) lines indicates --> parent :child (or) associate group in the group

**Key Components:**

**Groups:** Each group is defined in a section using the **[group\_name]** syntax.

**Hosts:** Below each group header, you list the hostnames or IP addresses of the servers.

**Group Variables:** You can define variables specific to a group using the **:vars** syntax

Using INI Inventory in Ansible

**Running Playbooks:** You can specify the inventory file when running Ansible commands using the -i option.

**ansible-playbook -i inventory.ini playbook.yml**

**Ad-hoc Commands:** You can also use ad-hoc commands with the inventory file.

**ansible -i inventory.ini webservers -m ping**

1. Playbooks

* Understanding the documentation
* Creating your first playbook
* Running playbooks

1. Services , Handlers & Shell

* Using the service module
* Understanding handlers
* Creating handlers
* Using shell & debug modules

1. Ansible Variables

* Introduction
* Understanding Jinja 2 templating

Incryption,descryption