→ Terraform is an open-source infrastructure as code(IAC).
→ tool developed by HashiCorp.
→ It enables you to define and manage your infrastructure resources using declarative configuration files.
→ It has multiple blocks like possible deployment scenarios:
  • On-premises / VMWare / Openstack.
  • Azure:
    - ARM Templates
    - Azure CLI / Powershell
    - Azure SDK
  • AWS:
    - CloudFormation
    - AWS CLI
    - AWS SDK
  • GCP:
    - GCloudSDK
    - GCloudCLI

★ **Infrastructure as Code (IAC):**

Terraform follows the principles of IAC, allowing you to define your infrastructure resources in human-readable configuration files. This approach provides several benefits, including version control, reproducibility, and ability to treat infrastructure.

* Resource :

→ A resource is an entity that represents a piece of infrastructure that you want to manage

→ This is the infrastructure which you want to create

* Provider :

→ A provider is a plugin that enables terraform to interact with a specific type of infrastructure.

→ Providers are responsible for translating terraform configurations into API calls and managing the lifecycle of resources created by terraform.

→ This refers to where you want to create infrastructure.

* Arguments :

- Arguments are the key value pair which are passed by the user. Arguments are the inputs to create the resources
- The inputs which we express in terraform.

* Variables (.tfvars) :

.tfvars are files the variables are provided/ assigned a value.

* Attributes :

- Attributes holds the output values of the resources that were created.
- The output given by terraform.

* Meta-Arguments :-

① **depends_on :**

The 'depends_on' meta-argument is used to explicitly specify the dependencies b/w resources

② **count :**

The 'count' meta-argument is used to create multiple instances of the resource based on a numeric value. It allows you to create a dynamic no. of similar resources.

③ **'for_each' :**

The 'for_each' meta-argument is used to create multiple instances of a resource based on a map (or) set of strings.

④ **Provider :**

The 'provider' meta-argument is used to configure provider specific settings for the particular block or module

⑤ **lifecycle :**

The 'lifecycle' metaargument is used to configure advanced settings related to the lifecycle of a resource, such as preventing a resource from being destroyed or specifying create before destroy behaviour.

* Null resource :-

It implements standard resource library, but no future action is taken. The triggers argument allows an arbitary set of values that will cause the replacement of resources when changed.

* Terraform lifecycle :

1) terraform init (initialize a work directory)
2) terraform validate (validate configuration files for syntax)
3) terraform plan (creates an execution plan)
4) terraform apply (Executes changes to the actual environment)
5) terraform refresh (update local state files against real resources)
6) terraform fmt (format code as per HCL canonical standard)
7) terraform output (list all the outputs for the root module)
8) terraform graph (creates a source graph listing all resources in your configuration & their dependencies)
9) terraform destroy (delete created resources / destroy / clean up)
10) terraform show (provide human-readable o/p from a state file)

* Terraform Variables :

They constraints are created from a mixture of type keywords and type constructors

- String
- Number
- boolean

The type constructors allow you to specify complex types such as collections

- list (<type>)
- Set (<TYPE>)

- Map (<TYPE>)
- Object ({<AFTER NAME> = <TYPE>, ...}).
- tuple ([<TYPE>, ...]).

* Terraform Command Line:

→ terraform fmt

→ terraform validate (validate code for syntax)

→ terraform validate -backend=false (validate code skip backend validation)

→ terraform init (initialize directory, pull down providers)

→ terraform init -get-plugins=false (initialized directory, do not download plugins)

→ terraform init -verify-plugins=false (initialize directory, do not verify plugins of Hashicorp signature)

→ terraform apply --auto-approve

→ terraform destroy --auto-approve.

* Terraform Module :-

→ Terraform module is a container for organizing and reusing terraform configurations.

→ It allows you to encapsulate a set of resources, input variables, output values, and other configuration elements into a reusable and shareable unit.

→ Modules helps in creating abstraction and separation of concerns in your infrastructure code.

→ They promote code reusability, maintainability, & make it easier to manage complex infrastructure setups.

* Taint :

→ 'terraform taint' command is used to manually mark a resource as tainted.

→ When a resource is tainted, terrafor considers it to be in an "error state" and plans to recreate (or) replace it during the next 'terraform apply' operation.

→ 'terraform untaint' command is used to remove the tainted state of a marked resource.

* Terraform graph :

- 'Terraform graph' command generates a visual representation of the dependancy graph for your infrastructure.
- It helps you visualize the relationships between resource defined in your terraform configuration.

* Terraform provisioning :-

Terraform provisioning is used to perform extra operations after resources are created

That could be generally executing scripts.

You can use provisioners to model specific actions on the local machine or on a remote machine in order to prepare servers or other infrastructure objects for service.

There are three provisioners :

1. local-exec
2. remote-exec
3. file

To connect to the remote instance we need connection.

Terraform provisioner will be run only when the resource is created.

① local-exec provisioner

- The local-exec provisioner invokes a local executable after a resource is created.
- This invokes a process on the m/c running terraform, not on the resource.

* Terraform State :

- As of now the terraform state is maintained in the local folder. when we try to execute the same infra from different system it also generates a new statefile which means new resources will be provisioned again.
- Solution for this is provided by terraform backends.
- The state refers to the information and data that terraform uses to manage and track the current state for your infrastroture.
- It represents the recorded state of the resources created and managed by terraform, as well as their attributes and dependencies.
- The statefile is typically stored in a json or binary format, depending on the version.

* Terraform Backend S3 :-

The statefile get stored in aws S3. but terraform S3 backend doesnot support locking. To perform locking terraform expects a dynamodb table to be created.

After adding backend information execute terraform init.

Terraform Workspace :

It allows us to create locking per workspace.
Each workspace can be consider as one environment.
Terraform workspace has multiple commands.

1) delete => terraform workspace delete <name>
2) list => terraform workspace list
3) new => terraform workspace new <name>
4) select => terraform workspace select <name>
5) show => terraform workspace show

Alias (Multiple provider configurations)

- You can optionally define multiple configuration for the same provider, and select which one to use on a per-resource or pre-module basis.
- The primary reason for this is to support multiple regions for a cloud platform.
- An 'alias' is a feature that allows you to assign multiple names or identifiers to a resource. It provides away to the same resource using different names within your terraform configuration.

Terraform Workspace :

It allows us to create locking per workspace.
Each workspace can be consider as one environment.
Terraform workspace has multiple commands.

1) delete => terraform workspace delete <name>
2) list => terraform workspace list
3) new => terraform workspace new <name>
4) select => terraform workspace select <name>
5) show => terraform workspace show

Alias (Multiple provider configurations)

- You can optionally define multiple configuration for the same provider, and select which one to use on a per-resource or pre-module basis.
- The primary reason for this is to support multiple regions for a cloud platform.
- An 'alias' is a feature that allows you to assign multiple names or identifiers to a resource. It provides away to the same resource using different names within your terraform configuration.

* Terraform Backends:

Backends defines where the state has to be stored.

There are two types of backends.

① local-Backend:

This is default backend.

② remote-backend:

Terraform uses persisted state data to keep track of the resources it manages.

=> We have multiple available backends

S3, azurerm, consul, cos, gcs, http, kubernetes, oss, and pg.

=> S3 bucket can be used as terraform backend, S3 backend does not support locking, if you need locking add dynamo db details.

Terraform troubleshooting errors

① Language Errors:

Most errors you run into, will more than likely fall in this category. These are your syntax errors in your configuration that terraform points out the line numbers and a brief explanation of what the error is.

② State Errors:

These are also pretty common errors, Most of these errors come from either your state being locked or out of sync. Terraform may destroy (or) change existing resource if your state is out of sync. Your configuration is always the first place to look when you run into these errors.

③ Core Errors:

These level of errors usually mean you have uncovered a bug and may even produce a panic error from the GO lang. Terraform was written in these errors usually mean you will be dumping your logs to a file, and attaching it to a templated bug report when you submit an issue on Github with the terraform core development team.

④ Provider Errors:

Same can be said for the provider errors. These errors resolve around the providers plugins and possible issues with different versions of the plugin. Usually this leads to submitting an issue in Github for the development team of that provider plugin.

* Problems with Terraform :

① Terraform communications to a lot of APIs but terraform itself does not have an API

② Terraform State
   - ⓐ Monolithic
   - ⓑ Need locking
   - ⓒ Cannot modify piece of infrastructure

③ Recommends breaking of monolithic statefiles

④ Doesnot react to manual changes made to infrastructure

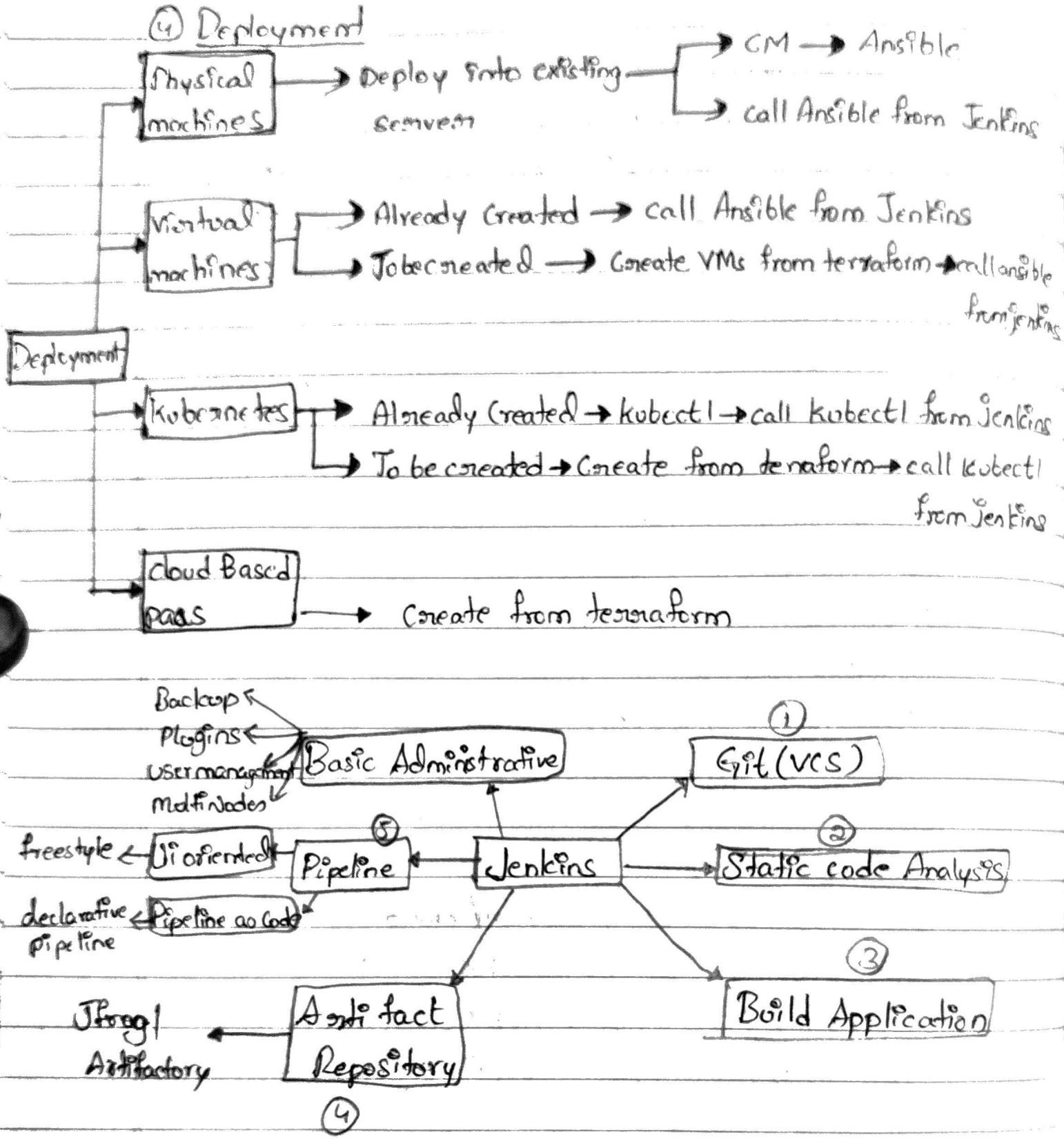
↑ terraform refresh

* Jenkins *

- CI/CD Stands for continuous integration and continuous delivery/Deployment, which are essential practices in the field of DevOps (Development and Operations)
- CI/CD is a set of principles and practices that aim to automate and streamline the software development and deployment process.
- There are 3 major workflows in which most of the projects fit in:
  1. Virtual / Physical Machine Deployment
  2. Infra provisioning and Deployment
  3. Containerized Deployment
- **Application Deployment Options:**

  Applications are deployed on

  1. **Virtual Machies / Physical Machines**
     - i) OnPrem
     - ii) cloud
  2. **Containers**
     - i) kubernetes / Onprem / cloud
  3. **Cloud Native Applications**
     - i) paas (Azure App Service / AWS Elastic Beanstalk)
     - ii) Faas (Azure Functions / AWS Lambda)



Jenkins is an open-source CI/CD engine.

Jenkins UI

Manage Jenkins → **System Configuration**
- System
- Nodes
- Tools
- Plugins
- clouds
- Appearance

Security
- Security
- Credentials
- Credential providers
- Users

Troubleshooting
- Manage Old Data

Status Information
- System information
- System log
- load statistics
- About Jenkins

Tools & Actions
- Reload Configuration from Disk
- Jenkins CLI
- Script Console
- Prepare for Shutdown

Plugins

AWS credential plugin

Mailer Plugin

Test runners

Junit (Java)

Pytest (Python)

Jasmine (nodejs)

nunit (c-sharp)

POM (project-object model)

* **Maven phases**
  - Complie
  - test
  - package
  - Install

* Path for initial password

/var/lib/Jenkins/secret/initialadminpassword

* Port forwarding path to file

/etc/default/Jenkins

another way → cd /usr/lib/systemd/system

vi Jenkins.service

Scripted Pipeline	Declarative Pipeline
① Scripted pipeline uses Groovy-based scripting syntax to define the pipeline steps	① Declarative pipeline uses a more structured, human-readable syntax based on the yaml format.
② It provides a lot of flexibility and allows you to write custom logic using groovy scripting	② It aims to provide a simpler and more opinionated way to define pipeline
③ Scripted pipeline provides more flexibility control over the pipeline workflow	③ Declarative pipeline provides simpler & more structured syntax.

* **Path for initial password:**

/var/lib/Jenkins/secret/initialadminpassword

* **Port forwarding path to file**

/etc/default/Jenkins

another way → cd /usr/lib/systemd/system

Vi Jenkins.service

Scripted Pipeline	Declarative Pipeline
① Scripted pipeline uses Groovy-based scripting syntax to define the pipeline steps	① Declarative pipeline uses a more structured, human-readable syntax based on the yaml format.
② It provides a lot of flexibility and allows you to write custom logic using groovy scripting Scripted pipeline provides more flexibility control over the pipeline workflow	② It aims to provide a simpler and more opinionated way to define pipeline ③ Declarative pipeline provides a simpler & more structured syntax.

* Projects in Jenkins:

1. Freestyle project (UI Based)
2. Pipeline → scripted pipeline
   → Declarative pipeline
3. Multi-configuration project
4. folder
5. Multibranch pipeline
6. Organization folder

1. Freestyle Project

A freestyle project in jenkins is a flexible and customizable job type that allows you to define build steps, configure build triggers, and specify post-build actions according to your requirements.

It provides a lot of flexibility and control over the build process.

1. General
2. Source Code Management
3. Build Triggers
4. Build Environment
5. Build steps
6. post-build Actions

② Pipeline:

A pipeline project is a way to define and manage CI, and CDelivery pipelines as code

In pipeline we have two types:

i) Scripted pipeline

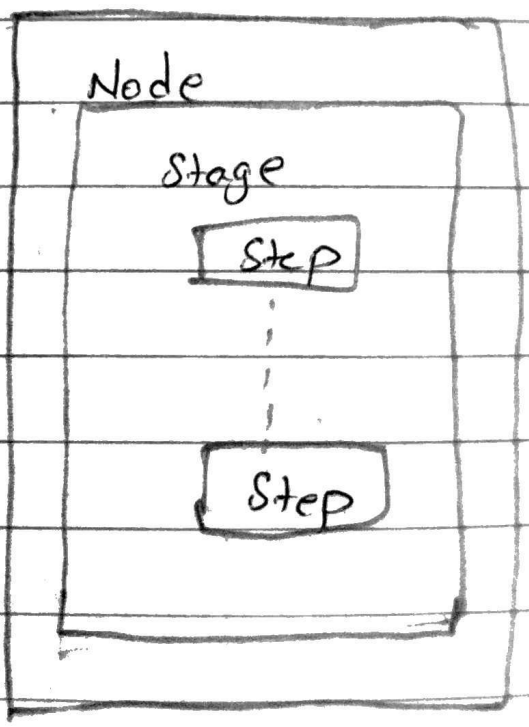
ii) Declarative pipline

i) Scripted pipeline

A scripted pipeline is one of the syntax options for defining pipelines as code. It allows you to write pipeline scripts using groovy, a programming language that runs on the JVM. In scripted pipeline, the pipeline syntax should starts with node

Ex:

```
node {
    stage('Build') {
        sh 'mvn package'
    }
    stage('Test') {
        sh 'mvn test'
    }
    stage('Deploy') {
        sh 'mvn deploy'
    }
}
```



② Pipeline:

A pipeline project is a way to define and manage CI, and CDelivery pipelines as code

In pipeline we have two types.

i) Scripted pipeline

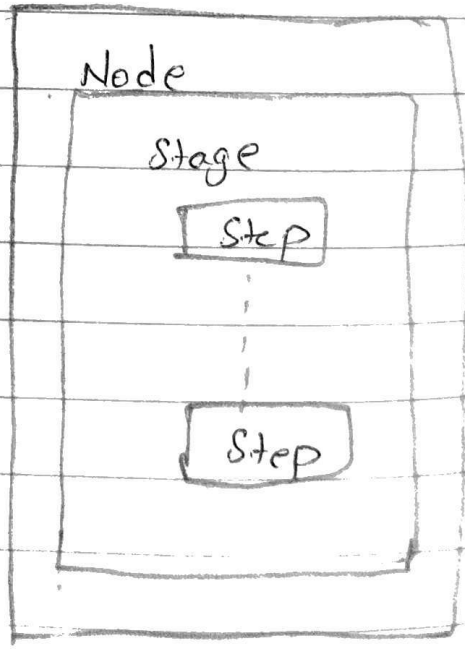
ii) Declarative pipline

i) Scripted pipeline

A scripted pipeline is one of the syntax options for defining pipelines as code. It allows you to write pipeline scripts using groovy, a programming language that runs on the JVM. In scripted pipeline, the pipeline syntax should starts with node

Ex:

```
node {
    stage('Build') {
        sh 'mvn package'
    }
    stage('Test') {
        sh 'mvn test'
    }
    stage('Deploy') {
        sh 'mvn deploy'
    }
}
```



ii) **Declarative pipeline** :-

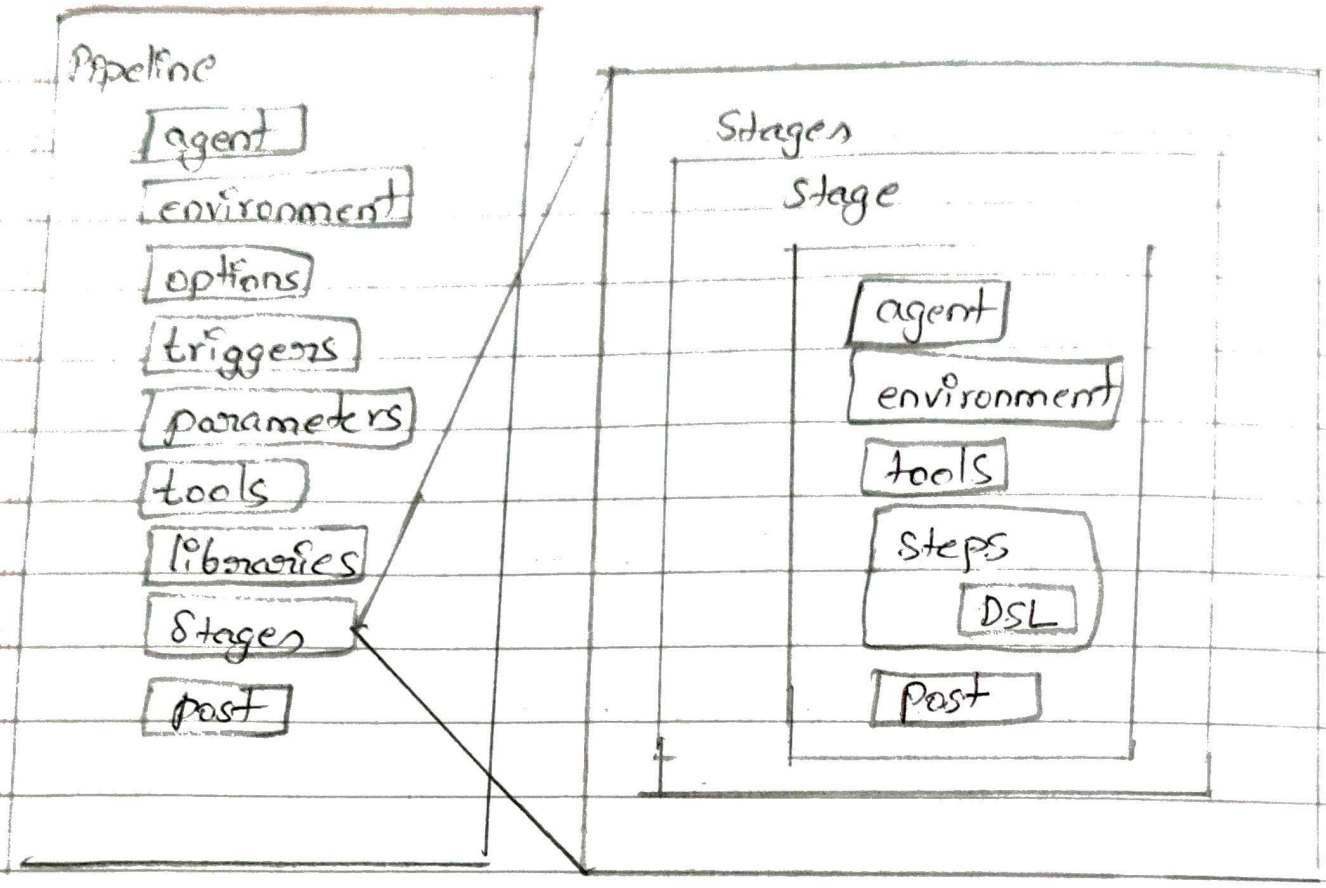
A declarative pipeline is one of the syntax options for defining pipelines as code.

It provides a more structured & opinionated approach to writing pipeline scripts compared to scripted pipeline syntax.

Declarative pipeline promotes readability, maintainability, and simplicity by enforcing a prefined structure and provisioning a more opinionated approach

Ex:

```
pipeline {
    agent any
    stages {
        stage ('Build') {
            steps {
                sh 'mvn package'
            }
        }
        stage ('Test') {
            steps {
                sh 'mvn test'
            }
        }
    }
}
```



* Day Builds and Night Builds:

Day Builds represent builds during active development time. During this we will have pipelines which will give feedback to developers on their commited codequity. This build will have

- → Build
- → Unit test
- → Package
- → Static Code Analysis

Night Build represents the coork of the team for the entire day. This Build will have

- Build
- Package
- Unit test
- Static Code Analysis
- Store the package into binary repository
- Creating/Update test environments will lastest packages and executing automated tasks
- Generate the reports which QA and management team will be interested.

* **Notifications:**

- For testing, lets setup fake smtp server and inbox in mail trap.io
- let's configure credentials for notifications into jenkins

Navigate to Manage Jenkins

↓

configure system

↓

Email notifications

* Static Code Analysis:

- It is a technique used in software development to analyze source code without executing it.
- It involves examining the code for potential defects, vulnerabilities, style violations, and other issues that could effect the quality, performance, and maintainability of the software
- The analysis is performed by specialized tools known as Static code analyzers (or) static Analysis tools.
- These tools scan the source code & apply a set of predefined rules, standards, or best practices to identify potential issues.

* Sonarqube:

- It is a popular open-source platform for continuous code quality management
- It integrates with various development tools, including jenkins, to provide comprehensive code quality analysis & reporting.
  1. Configure sonarqube server
  2. Install sonarqule scanner
  3. Configure sonarqube in jenkins
  4. Configure build step
  5. Execute build
  6. View analysis results.

③ Molti Configuration project:

- A molti-configuration project in jenkins, also known as a matrix project, is a type of jenkins job that allows you to perform parallel builds across different combinations of axes.
- It is useful when you need to build and test your software on multiple platforms, operation system or configurations.
- With a molti-configuration project, you can define axes that represent different dimensions or variations of your build, such as different operating systems, browsers, versions, (or) any other configuration parameter relevant to your project.

④ folder:-

- In jenkins, a folder is a feature that allows you to organize and categorize your jenkins jobs into logical groups (or) directories.
- Folders provide a hierarchical structure for better management and navigation of jobs within jenkins

① Organizational Structure
② Simplified Navigation
③ Access control
④ Bulk Operations
⑤ Configurable views.

5 Multibranch pipeline:

- A multibranch pipeline is a feature in Jenkins that allows you to automatically create & manage pipelines for multiple branches of a version control repository.
- It is designed to simplify the configuration and management of pipelines for projects with multiple branches, such as GIT or Mercurial repositories.
- With a multi-branch pipeline, Jenkins scans your version control repository for branches & automatically creates & configures pipelines for each branch.

This enables you to have separate pipelines for each branch, facilitating continuous integration & delivery for each branch independently.

1. Configure the multi-branch pipeline project
2. Scan for branches
3. Automatic pipeline creation.
4. Branch-specific pipeline
5. Automatic Triggering
6. Visibility & Reporting

Organization Folders:

An organization folder in Jenkins is a feature that allows you to organize and manage multiple projects or repositories belonging to an organization or a group of related projects.

It provides a higher-level organizational structure for managing jobs & pipelines within Jenkins.

1. Automatic Job Creation
2. Repository / Project Synchronization
3. Visibility & Monitoring
4. Fine-grained Access Control
5. Configuration Code

To create an organization folder in Jenkins.

i) Install the "CloudBees folder" plugin from the Jenkins plugin manager if it's not already installed.

ii) From the jenkins dashboard, click on the "New Item" link

iii) Configure the organization folder, specifying the source repositories or projects, and any additional settings such as accesscontrol or sychronization options.

iii) Enter a name for the organization folder & select the "Organization folder" option.

iv) Save the organization folder Configuration.

* Jenkins stash & unstash:

Stash Step:

The 'Stash' Step is used to store files or directories in a temporary location during the pipeline execution. It allows you to specify a name for the stash & define the files or directories to be stashed.

Unstash Step:

The "unstash" step is used to retrieve the stashed files or directories later in the pipeline. It allows you to specify the name of the stash to be retrieved & Optionally specify a target directory where the files should be unarchived.

* Jenkins Shared library:

- Shared library is something which makes your jenkins code reusable
- Directory Structure of a shared library repository
- A "shared library" in jenkins is a reusable collection of groovy scripts that can be used by multiple jenkins job
- This allows you to share code and functionality b/w diff jobs, which can make your builds more efficient and easier to maintain.

GIT

* Version Control System:

- This is a software which helps organizations to maintain the source code.
- VCS helps in maintaining history of changes.

VCS allows us to maintain track of different releases which we give to customers

VCS allows parallel development by multiple developers.

Types of VCS:

1. local Version Control Systems
2. Centralized VCS
3. Distributed VCS

* GitHub:

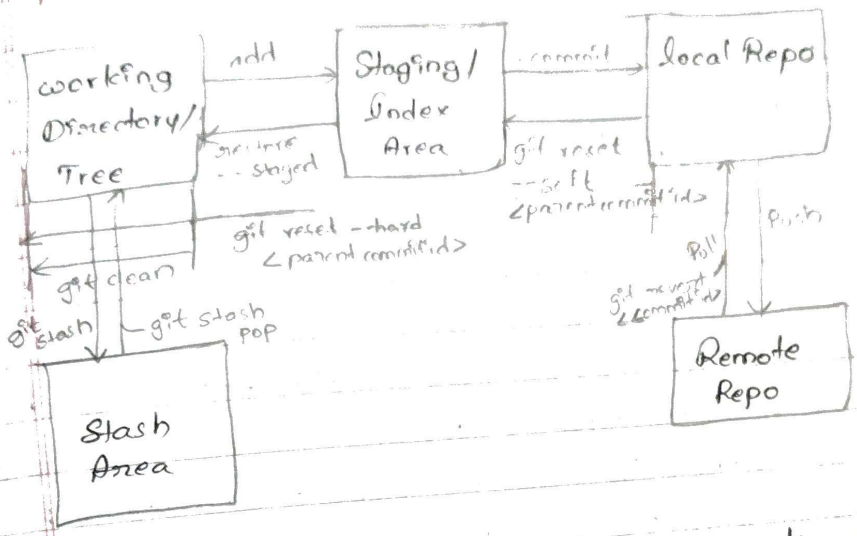
Github is a web-based platform that provides hosting for git repositories which are primarily used for Version Control.

Git is a distributed Version control system that allows multiple people to collaborate on a project by tracking changes to files over time.

Github add a layer of functionality on top of git, making it easier to work with repositories, collaborate with others, and manage s/w development projects.

repositories

Git flow :

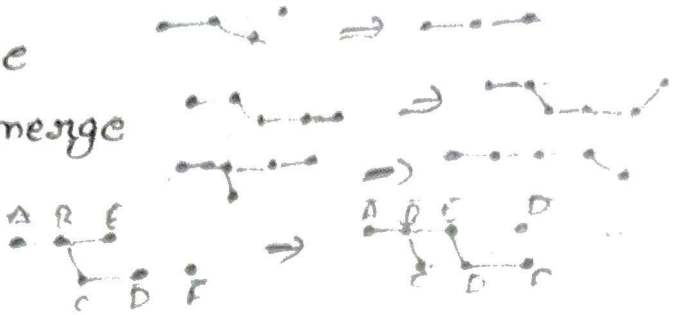


- git reset (unstage a file while retaining the changes in coorking directory)
- git log (or) git log --oneline
- git branch, git branch [branch-name], git checkout
- git status
- get rebase -i
- git merge [branch]
- git init
- git clone [url]
- git rm [filename]
- git stash
- git stash list
- git remote add [alias] [url]
- git push [alias] [branch
- git pull

* **Merge**:

Three ways to merge

1) No fast forward merge
2) Rebase
3) Cherrypick



Centralized VCS	Distributed VCS
① In a centralized VCS there is a single central repository that stores the entire version history of project	① In a distributed VCS, each developer has a complete copy of the repository, including the entire version history on their local machine
② The central repo acts as a server & developers interact with it by checking out files, making changes, & committing them back to the central repository	② Every developers have local copy is a fully functional repository with the ability to perform version control operations independently
③ Network Dependency	③ Offline Operations
④ File locking	④ Branching and merging
⑤ Access Control	⑤ Flexibility
⑥ Collaboration is not possible	⑥ Collaboration

* what is the git command that downloads any repo from github to your computer
  git clone

* Using Standard way
  git init

  Using the bare way
  git init --bare

① You create a working directory with git init	① Does not contain any working or checked out copy of source files.
② .git subfolder is created with all the git related revision history	② Bare repo's store git revision history in the root folder of ur repo instead of .git subfolder

* what is the process to revert a commit that has already been pushed and made public.
  → git revert <commit id>
  (create a new commit that undoes all the changes that were made in the bad commit)

* Git fetch	Git Pull
① Git fetch for only downloads new data from a remote repo.	① Git pull updates the current head branch with the latest changes from the remote server.
② Does not integrate any of this newdata into ur working files	② It downloads new data and integrate it with the current working files

② Git fetch can be done any time to update the remote-tracking branches

`git fetch origin`
`git fetch --all`

③ It tries to merge remote changes with your local ones

`git pull origin master`

Git Merge

① Creates an extra merge commit every time you need to incorporate changes

② Pollutes your feature branch history

Git rebase

① Incorporate all the new commits in master branch

② Re-writes the project history by creating brand new commits for each commit in the original branch

* How do you find a list of files that has been changed in a particular commit.

→ `git diff-tree -r {commit hash}`

∴ "-r" → flag allows the command to list individual files.

"commit hash" → will list all the files that were changed or added in that commit

* What is a merge conflict in git and how can it be resolved?

It arises when you have merge branches that have connecting commits and git needs your help to decide which changes to incorporate in the final merge.

1) Under your repository name, click 'Pull request'
2) In 'Pull requests' list. Click the 'pull request' with a merge conflict that you'd like to resolve
3) Near the button of your pull request, click Resolve conflicts
4) Decide if you want to keep only your branches changes keep only the other branch's changes or make a brand new change, which may incorporate changes from both branches
5) Delete the conflict markers and make changes you want in the final merge.
6) Once you have resolved all the conflicts in the file click 'Mark as resolved'.
7) Once you have resolved all your merge conflicts, click 'Commit Merge'. This merges the entire base branch into your head branch.
8) To merge your pull request, Click 'Merge Pull Request'

→ git rebase -i HEAD ~n

p, pick <commit> = use commit

r, reword <commit> = use commit, but edit the commit message

e, edit <commit> = use commit, but stop for on edit

s, squash <commit> = use commit, but melt into previous commit

Git	GitHub
① Installed locally on the system	① Hosted on the cloud
② Git can be used offline & does not need an internet connection for use	② Github cannot be used offline and needs an internet connection
③ Git can be used w/o Github	③ Github cannot be used w/o git
④ Used for version control & code sharing	④ Used for centralised source code hosting
⑤ There is no GUI	⑤ It provides an easy to use GUI
⑥ Code changes like commit, merge, etc are done using commands from the command line	⑥ Everything is done through a web-based interface
⑦ Open source licensed	⑦ Includes a free and pay for users tiers