

SBL

🕒 Created	@November 4, 2025 10:32 PM
☰ Tags	

▼ Main Difference Between Git, Jenkins, and Docker

Simple Explanation

- **Git** → Developers use it to **store and track code versions**.
Example: You and your teammates upload your project code to GitHub using Git.
- **Jenkins** → Automatically **builds and tests** your code from Git.
Example: Jenkins watches your Git repo — whenever new code is pushed, it runs tests and builds automatically.
- **Docker** → Packages and **deploys your built application** into containers.
Example: Jenkins can build a Docker image of your app and deploy it to a production server.

Tool	Purpose	Stage in DevOps Cycle	Simple Analogy / Function
Git	Version Control System	Code Management (Source Control)	Keeps track of code changes — like a “time machine” for your code.

Tool	Purpose	Stage in DevOps Cycle	Simple Analogy / Function
Jenkins	Continuous Integration / Continuous Deployment (CI/CD) Automation Tool	Build, Test, and Deploy Automation	Acts like a "robot" that automatically builds, tests, and deploys your code whenever you update it.
Docker	Containerization Platform	Deployment and Environment Management	Packages your app and dependencies into a "container" that runs anywhere without compatibility issues.

▼ GIT

1 What is Git?

- Git is a **Version Control System (VCS)**.
- It helps you **track changes** in your code, **go back to old versions**, and **collaborate** with others easily.

 *Think of Git as a "time machine" for your code.*

2 What is a Repository?

- A **repo** is like a **folder** that stores your project files **plus** all version history.
- Two types:
 - **Local repo:** On your computer.
 - **Remote repo:** On GitHub, GitLab, etc.

3 Git Installation & Configuration

After installing Git, set your name and email:

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

Check configuration:

```
git config --list
```

4 git init

Creates a new Git repository in your project folder.

```
git init
```

 It creates a hidden `.git` folder that stores version history.

5 git add

Moves files from the **working directory** → **staging area** (preparing them for commit).

```
git add filename
```

or add all:

```
git add .
```

6 git commit

Saves changes permanently in the local repository.

```
git commit -m "your message"
```

 Each commit is like a “save point” in your project.

7 git status

Shows which files are modified, added, or staged.

```
git status
```

8 git log

Displays all commits with author and message.

```
git log
```

Short version:

```
git log --oneline
```

10 git reset / git revert

- **git reset** → Moves HEAD pointer back (undo commits).
- **git revert** → Makes a new commit that undoes changes (safer).

```
git revert <commit_id>
```

1 Branching and Merging

A **branch** is like a separate line of development.

Command	Description
<code>git branch</code>	List branches
<code>git branch newfeature</code>	Create new branch
<code>git checkout newfeature</code>	Switch branch
<code>git merge newfeature</code>	Merge with main branch
<code>git branch -d newfeature</code>	Delete a branch

 Main branch is usually called `main` or `master`.

2 Merge Conflicts

Happens when two people change the same part of a file.

You need to manually decide which changes to keep.

3 git remote

Connects your local repo to a remote one (e.g., GitHub).

```
git remote add origin https://github.com/user/repo.git
```

View remotes:

```
git remote -v
```

4 git push

Uploads commits from **local repo** → **remote repo** (GitHub).

```
git push origin main
```

5 git pull

Downloads and merges changes from remote to local repo.

```
git pull origin main
```

16 git clone

Copies an existing remote repository to your computer.

```
git clone https://github.com/user/repo.git
```

17 git tag

Marks specific commits as versions (like checkpoints).

```
git tag v1.0  
git push origin v1.0
```

18 .gitignore

A file that tells Git which files/folders to ignore (like logs, temporary files).

Example `.gitignore` :

```
*.log  
node_modules/  
__pycache__/
```

19 Forking

- **Fork** = copy someone else's repo to your account (on GitHub).
- Lets you edit and propose changes without affecting the original project.

20 Pull Request (PR)

- After making changes in your forked repo, you create a **Pull Request**.
- It asks the main repo owner to **review and merge** your changes.

21 Git vs GitHub

Feature	Git	GitHub
Type	Version control tool	Cloud hosting for Git repos
Works	Offline (local)	Online (web-based)
Use	Track versions	Collaborate & share code

22 Key Concepts Summary

Term	Meaning
Repository	Folder containing project + version history
Commit	A saved version of your work
Branch	A separate workspace for features
Merge	Combining branches
Clone	Copy remote repo locally
Push / Pull	Sync between local and remote
Tag	Label for specific version
HEAD	Pointer to your current branch/commit

Viva Short Answer Example

"Git is a version control tool that helps track changes and collaborate.

I can use `git init` to start a repo, `git add` to stage files, `git commit` to save changes, and `git push` to upload to GitHub.

Branching allows parallel development, and merging combines code.

Tags are used for version releases, and `.gitignore` avoids unwanted files being tracked."

▼ JENKINS

1 What is Jenkins?

- Jenkins is an **open-source automation tool** written in **Java**.
- It is used for **Continuous Integration (CI)** and **Continuous Deployment (CD)**.
- It automatically builds, tests, and deploys code whenever changes are made.

 *Think of Jenkins as a "robot" that builds and delivers your code automatically.*

2 CI/CD Explained

Term	Meaning
Continuous Integration (CI)	Automatically testing and integrating code whenever a developer commits changes.
Continuous Deployment (CD)	Automatically deploying tested code to servers or cloud environments.

3 Why Use Jenkins?

 Automates repetitive tasks

 Detects bugs early

- Saves time in builds and deployments
- Integrates with tools like Git, Docker, Maven, and more

4 Jenkins Architecture (Simple View)

Developer → Git → Jenkins → Test/Build → Deploy (Server)

Component	Description
Jenkins Master	Controls and schedules jobs.
Jenkins Agent (Slave)	Executes the actual build tasks.
Plugins	Add extra functionality (e.g., Git plugin, Maven plugin).

5 Installation Steps

1. Install **Java (JDK)** → Jenkins requires Java.
2. Download **Jenkins** ([.war](#) or [.msi](#)) from [jenkins.io](#).
3. Run it and open <http://localhost:8080> in browser.
4. Unlock Jenkins using the admin password (found in [secrets](#) folder).
5. Install suggested plugins → create an admin user → start using Jenkins!

6 Jenkins Dashboard

- After login, you'll see the **dashboard** with all jobs.
- From here, you can:
 - Create jobs or pipelines
 - Manage plugins
 - Check build history
 - Configure global settings

7 Jenkins Job Types

Type	Description
Freestyle Project	Simple job with manual configuration (build, test, deploy).
Pipeline	Code-based automation using a Jenkinsfile .
Multi-configuration	For testing across different environments.
Folder / View	For organizing multiple jobs.

8 Creating a Freestyle Job

1. Click **New Item** → **Freestyle Project**
2. Add **description**
3. Choose **Git** as the source (enter repo URL)
4. Add build steps (e.g., run shell commands or scripts)
5. Save → click **Build Now**

 Jenkins runs the job and shows console output.

9 Jenkins Pipeline (Code-Based Job)

- Pipeline is a script that defines the entire build, test, deploy process.
- Written in **Groovy** and stored in a file called `Jenkinsfile`.

Example:

```
pipeline {  
    agent any  
    stages {  
        stage('Build') {  
            steps {  
                echo 'Building the project...'  
            }  
        }  
        stage('Test') {  
            steps {  
                echo 'Running tests...'  
            }  
        }  
        stage('Deploy') {  
            steps {  
                echo 'Deploying application...'  
            }  
        }  
    }  
}
```

 Pipelines are **automated**, **repeatable**, and **easy to modify**.

10 Jenkins Plugins

- Plugins extend Jenkins functionality.

- Example plugins:
 - **Git Plugin** → connects Jenkins to GitHub.
 - **Maven Plugin** → builds Java projects.
 - **Docker Plugin** → builds/deploys Docker containers.
 - **Email Plugin** → sends build notifications.

Manage plugins:

Dashboard → Manage Jenkins → Manage Plugins

11 Jenkinsfile

- A text file that defines the pipeline code.
- Stored in your project's Git repository.
- Jenkins reads it automatically to build your pipeline.

12 Common Jenkins Terms

Term	Meaning
Node	A machine Jenkins runs on (master or agent).
Workspace	Directory where Jenkins builds and stores files.
Build Trigger	Condition that starts a job (like code commit or time).
Post-build Action	What to do after build (e.g., email, deploy).

13 Build Triggers

Jenkins can start jobs automatically using triggers:

Trigger Type	Description
Poll SCM	Check Git repo for new commits.
Build Periodically	Run job at scheduled times (like cron jobs).
After Other Projects	Trigger job after another one finishes.
Webhook	GitHub triggers Jenkins automatically on new code push.

14 Jenkins and Git Integration

- Install **Git Plugin**
- In a Freestyle or Pipeline job:

- Choose “Git” under Source Code Management
 - Add your repository URL
 - Jenkins will **pull latest code** automatically before building
-

15 Jenkins and Docker Integration

- Jenkins can build **Docker images** and deploy **containers**.
- Use **Docker Plugin** or **Docker commands** in the pipeline:

```
stage('Docker Build') {  
    steps {  
        sh 'docker build -t myapp .'  
        sh 'docker run -d -p 8080:80 myapp'  
    }  
}
```

16 Jenkins Security

- Add login users and roles
 - Manage permissions under
Manage Jenkins → Configure Global Security
 - Use authentication for safe CI/CD pipelines
-

17 Key Jenkins Commands

(Usually run through terminal or Jenkins UI)

Command	Description
<code>java -jar jenkins.war</code>	Run Jenkins manually
<code>http://localhost:8080</code>	Open Jenkins Dashboard
<code>jenkins-cli.jar</code>	Command-line interface for Jenkins

18 Jenkins Logs and Reports

- Jenkins stores build logs automatically.
- You can see them in:

```
Dashboard → Job → Console Output
```

19 Backup and Restore

- Backup your Jenkins configuration:
 - `.jenkins` folder (in home directory)
 - You can restore Jenkins by replacing the same folder.
-

20 Advantages of Jenkins

-  Open source & free
 -  Automates CI/CD pipelines
 -  Integrates with almost all DevOps tools
 -  Highly customizable through plugins
 -  Saves time, detects errors early
-

Common Viva Questions (with Short Answers)

Question	Short Answer
1 What is Jenkins?	An automation tool for Continuous Integration and Continuous Deployment.
2 What language is Jenkins written in?	Java.
3 What is CI/CD?	Continuous Integration and Continuous Deployment.
4 What is a Jenkins job?	A task Jenkins performs, like building or testing a project.
5 What is a pipeline?	A script that defines automated steps (build, test, deploy).
6 What is a Jenkinsfile?	A file that contains pipeline code.
7 What is a plugin?	An add-on that extends Jenkins features.
8 What is the default Jenkins port?	8080.
9 What is a build trigger?	An event that automatically starts a job (like code commit).
10 Why is Jenkins important in DevOps?	It automates the entire software build, test, and deployment process.

1-Minute Viva Explanation Script

"Jenkins is an open-source automation tool used for CI/CD.
It automates tasks like building, testing, and deploying code.
I installed Jenkins using Java, unlocked it, installed plugins, and created jobs.
It supports Freestyle and Pipeline jobs — pipelines use a Jenkinsfile written in Groovy.

Jenkins integrates easily with Git and Docker to automate complete DevOps workflows."

▼ DOCKER

◆ 1 What is Docker?

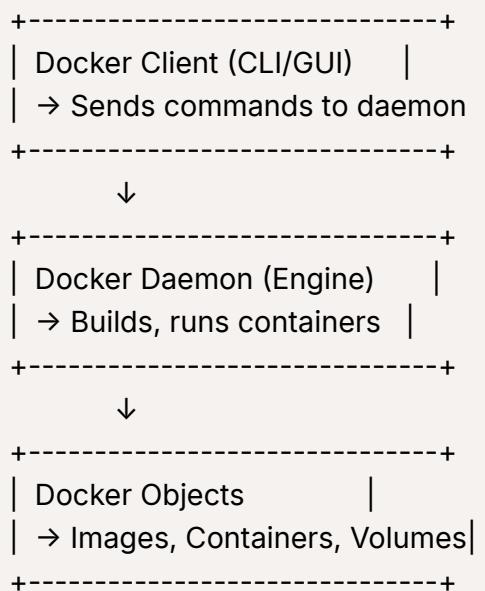
- Docker is a **containerization platform**.
- It lets you **package** your application + its dependencies (libraries, configs) into **containers**.
- Containers run **anywhere** — on any system — without compatibility issues.

💡 *Think of a container like a lunchbox — it has everything your app needs inside it.*

💡 2 Why Docker?

- ✓ Avoids “works on my machine” problems
- ✓ Lightweight and fast (unlike virtual machines)
- ✓ Easy to deploy and scale apps
- ✓ Perfect for **DevOps CI/CD pipelines**

⚙️ 3 Docker Architecture (Simple View)



Component	Description
Docker Client	The command-line interface users interact with.
Docker Daemon	Runs in the background; manages containers and images.
Docker Image	Blueprint/template for containers.

Component	Description
Docker Container	Running instance of an image.
Docker Hub	Cloud registry to store and share images.

4 Installing Docker

Windows:

1. Download **Docker Desktop** from [docker.com](https://www.docker.com)
2. Install and enable **WSL2** backend
3. Verify:

```
docker --version
docker run hello-world
```

 "Hello from Docker!" → installation successful.

5 Docker Image

- A **read-only template** that contains app code + environment.
- Built from a **Dockerfile**.

Commands:

```
docker pull ubuntu
docker images
docker rmi <image_id>
```

 *Image = recipe / Container = cooked dish.*

6 Docker Container

- A **running instance** of an image.
- Can be started, stopped, deleted anytime.

Commands:

```
docker run -it ubuntu
docker ps      # show running containers
docker ps -a    # show all containers
docker stop <id>  # stop container
docker start <id> # restart stopped container
docker rm <id>   # delete container
```

7 Dockerfile

- A text file with **instructions** to build a Docker image.

Example:

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python3
CMD ["python3", "--version"]
```

Build and run:

```
docker build -t mypython .
docker run mypython
```

8 Docker Networking

Allows containers to **communicate** with each other or with external systems.

Network Type	Description
bridge	Default network; containers communicate via names.
host	Shares host's network (no isolation).
none	No networking; isolated container.
overlay	Used for multi-host networking (Swarm).

Commands:

```
docker network ls
docker network create mynet
docker run -dit --name app1 --network mynet ubuntu
docker exec -it app1 ping app2
```

9 Linking Containers

Older method (now replaced by networks):

```
docker run -dit --name db ubuntu
docker run -dit --name web --link db ubuntu
```

Now containers connect using **custom networks** instead of `--link`.

10 Port Mapping

Used to **expose container services** to the outside world.

Example:

```
docker run -d -p 8080:80 nginx
```

→ Maps container's port 80 to host's port 8080.

Access in browser:

```
http://localhost:8080
```

11 Docker Volumes

Used to **store data permanently** (even if container is deleted).

```
docker volume create myvolume  
docker run -v myvolume:/data ubuntu  
docker volume ls
```

 Without volumes, container data is lost after deletion.

12 Docker Compose

- A tool to manage **multi-container applications**.
- Uses a file called `docker-compose.yml` to define services.

Example:

```
version: '3'  
services:  
  web:  
    image: nginx  
    ports:  
      - "8080:80"  
  db:  
    image: mysql
```

Run all containers:

```
docker-compose up
```

Stop all:

```
docker-compose down
```

13 Docker Registry / Docker Hub

- Central repository to **store and share images**.
- Example:

```
docker login  
docker push username/myimage  
docker pull username/myimage
```

14 Docker Commands Quick Summary

Command	Function
<code>docker pull</code>	Download image from Docker Hub
<code>docker run</code>	Run container from image
<code>docker ps</code>	List containers
<code>docker stop</code>	Stop container
<code>docker rm</code>	Remove container
<code>docker images</code>	List images
<code>docker rmi</code>	Remove image
<code>docker build</code>	Build image from Dockerfile
<code>docker exec -it</code>	Enter container terminal
<code>docker network ls</code>	List networks
<code>docker volume ls</code>	List volumes

15 Docker vs Virtual Machine

Docker	Virtual Machine
Lightweight (shares OS kernel)	Heavy (runs full OS)
Fast startup	Slow startup
Uses fewer resources	Uses more resources
Runs isolated apps	Runs full OS environments
Uses containers	Uses hypervisors (VMware, VirtualBox)

 *Docker = lightweight, faster alternative to VMs.*

16 Advantages of Docker

- Lightweight & portable
 - Faster deployment
 - Easy scaling and replication
 - Works with Jenkins for CI/CD
 - Environment consistency (same everywhere)
-

Viva Questions and Answers (Short)

Question	Answer
1 What is Docker?	A containerization platform to package apps and dependencies together.
2 What is a container?	A lightweight, isolated environment to run applications.
3 What is a Docker image?	A template used to create containers.
4 What is Docker Hub?	A cloud repository to store and share images.
5 What is the difference between Docker and VM?	Docker is lightweight and shares OS; VM runs full OS and is heavy.
6 What is a Dockerfile?	A text file containing instructions to build a Docker image.
7 What is Docker Compose?	A tool to define and run multiple containers using a YAML file.
8 What is a Docker volume?	Storage that persists data even after the container is deleted.
9 What is port mapping?	Mapping container port to host port (e.g., <code>-p 8080:80</code>).
10 How to check running containers?	<code>docker ps</code>

1-Minute Viva Explanation Script

"Docker is a containerization tool that packages an app with its dependencies into a portable container.

Containers are lightweight, fast, and run anywhere.

I learned Docker installation, basic commands like `docker run`, `docker ps`, and `docker build`, networking, volumes, and Docker Compose for multi-container setups.

Docker helps in faster, consistent deployments in DevOps pipelines."

EXP 3

- ◆ What is Jenkins?

Jenkins is an **open-source automation tool** written in Java.

It helps developers **automate** building, testing, and deploying code — a key part of the **DevOps** process.

How Jenkins Works

1. **Developer pushes code** → to GitHub or GitLab.
2. **Jenkins detects changes** → pulls latest code.
3. **Builds** → using tools like Maven or Gradle.
4. **Tests** → using automated testing (e.g., Selenium).
5. **Deploys** → application to server or container.
6. **Notifies** → the team about success or failure.

Term	Description
Continuous Integration (CI)	Automatically integrating and testing code after each commit.
Continuous Deployment (CD)	Automatically deploying tested code to production or servers.
Plugin	Add-on that extends Jenkins' functionality (e.g., Git plugin, Maven plugin).
Build Job	A task Jenkins executes (like building or testing a project).

Advantages of Jenkins

- Free and open source
- Easy to install and configure
- Works on Windows, Linux, and macOS
- Over 1000+ plugins available
- Supports distributed builds (parallel builds on multiple machines)

What is Jenkins Dashboard?

The Jenkins dashboard is the main interface to view, create, manage, and monitor jobs and pipelines.

- **Jenkins Master:** Manages jobs, schedules builds.
- **Jenkins Agent:** Executes builds (can be on different machines).
- **Plugins:** Integrate with tools (Git, Docker, Maven, etc.)

EXP 4

◆ What is a Jenkins Pipeline?

A **pipeline** in Jenkins is a series of **automated steps** that define how your software is built, tested, and deployed.

⚙️ Types of Jenkins Pipelines

Type	Description
Declarative Pipeline	Easier to write, uses a simple structured syntax (recommended for beginners).
Scripted Pipeline	More flexible, uses Groovy scripting language (for advanced users).

Jenkins Pipeline Structure

A simple pipeline script (Declarative format):

```
pipeline {  
    agent any  
    stages {  
        stage('Build') {  
            steps {  
                echo 'Building the application...'  
            }  
        }  
        stage('Test') {  
            steps {  
                echo 'Running tests...'  
            }  
        }  
        stage('Deploy') {  
            steps {  
                echo 'Deploying application on server...'  
            }  
        }  
    }  
}
```

🧠 Explanation:

- `pipeline {}` – Defines the start of the pipeline.
- `agent any` – Tells Jenkins to run on any available node.
- `stages {}` – Contains different steps (like build, test, deploy).
- `echo` – Displays text in the Jenkins console output.

◆ What is the difference between Scripted and Declarative Pipeline?

Scripted uses Groovy language (flexible but complex), Declarative uses simple structure (easier to use).

◆ What is the use of Jenkinsfile?

A Jenkinsfile is a text file containing pipeline code written using a domain-specific language (DSL) to define automation steps.

◆ Differentiate Declarative and Scripted Pipeline.

Declarative pipelines use a structured syntax for simplicity, while scripted pipelines offer more flexibility using Groovy scripting.

◆ What are the main stages in a Jenkins Pipeline?

Common stages include Build, Test, Deploy, and Monitor — each representing a distinct part of the CI/CD process.

◆ What is a Node in Jenkins?

A Node represents a machine that executes the Jenkins job. The master node coordinates while agent nodes perform tasks

◆ What is the difference between Freestyle job and Pipeline job?

Freestyle is manual and simple; Pipeline automates multiple stages and is code-based.

EXP 5

◆ What is Tomcat?

Apache Tomcat is an **open-source web server and servlet container** developed by the Apache Software Foundation.

It is used to **deploy Java-based web applications** (WAR files).

◆ Why Install Jenkins on Tomcat?

Normally, Jenkins runs on its **own internal web server** (default port 8080).

But sometimes, organizations prefer running Jenkins on **Tomcat** for:

- Better management and integration with existing web apps.
- Using **application server security** (Tomcat user roles).
- Centralized control over multiple applications.

◆ What is a WAR file?

A Web Application Archive (WAR) file packages web applications for deployment on servers like Tomcat.

◆ What file is used to start Tomcat?

`startup.bat` (Windows) or `startup.sh` (Linux).

◆ What is a Web Server?

A: A web server processes HTTP requests and serves web content (HTML, JS, CSS) to clients.

◆ List other web servers used for deployment.

A: Some common web servers are Apache HTTP Server, Nginx, Jetty, and Microsoft IIS

 key points

- Jenkins can run as a **standalone server** or inside **Tomcat**.
 - Installing Jenkins as a **WAR** in Tomcat gives more flexibility and integration.
 - Java and Tomcat must be properly configured for Jenkins to run smoothly.
-

EXP 6

EXP 7

◆ What is Docker?

Docker is an open-source **containerization platform**.

It allows you to **package an application** and its **dependencies** (libraries, configurations) into a **container** — a lightweight, portable unit that runs the same way anywhere.

 Why Docker is Used in DevOps

- Ensures apps **run identically** on any system.
- Helps developers and operations teams work smoothly together.
- Forms the base for **CI/CD automation**.
- Eliminates the “works on my machine” problem.

Q: What is a Docker Container?

A: A Docker container is a lightweight runtime environment created from a Docker image to run isolated applications.

Q: What is a Docker Image?

A: A Docker Image is a template that defines the environment and dependencies for a container.

Q: What is a Dockerfile?

A: A Dockerfile is a script that defines the steps required to build a Docker image, such

as base OS, dependencies, and commands.

Q: What is Docker Hub?

A: Docker Hub is a public cloud-based registry for storing, sharing, and managing Docker images.

Q: Explain Docker Architecture.

A: Docker architecture includes a client, a Docker daemon (server), and registries. The daemon builds, runs, and manages containers.

Q: Differentiate between Container and Virtual Machine.

A: A container shares the host OS kernel, making it lightweight; a VM runs a full OS, consuming more resources

Basic Docker Commands (with Meaning)

Command	Description
<code>docker --version</code>	Check Docker version
<code>docker pull <image></code>	Download an image from Docker Hub
<code>docker images</code>	List all downloaded images
<code>docker ps</code>	List all running containers
<code>docker ps -a</code>	List all containers (running + stopped)
<code>docker run <image></code>	Run a new container from an image
<code>docker stop <container_id></code>	Stop a running container
<code>docker start <container_id></code>	Restart a stopped container
<code>docker rm <container_id></code>	Remove a container
<code>docker rmi <image_id></code>	Remove an image
<code>docker build -t <name> .</code>	Build a custom image from Dockerfile
<code>docker exec -it <container_id> bash</code>	Access a running container's terminal
<code>docker logs <container_id></code>	View container logs

EXP 8

EXP 9