

- Runs:
- `docker build -t myapp .`

4 □ Jenkins Runs or Deploys Container

Jenkins can run:

```
docker run -d -p 3000:3000 myapp
```

Now the updated app is deployed automatically.

Simple Architecture Flow

Developer → GitHub → Jenkins → Docker Build → Docker Run → Application Live

Important Jenkins Concepts

Job

A task Jenkins performs (build, test, deploy).

Pipeline

A series of steps automated in order (build → test → deploy).

Plugin

Extra tools to connect Jenkins with GitHub, Docker, etc.

Jenkins Installation - (Ubuntu 24.04)

1 Install Java (Required for Jenkins)

Jenkins needs Java to run.

- Update system (already done):
- `sudo apt update`
- `sudo apt upgrade`
- Install Java 21 (Eclipse Temurin)
- Verify:
- `java -version`

2 Add Jenkins Repository & Install

- Add Jenkins GPG key and repository
- Update package list
- Install Jenkins:
- `sudo apt install jenkins`
- Check status:
- `systemctl status jenkins`

Jenkins service should be **active and running**.

3 Customize Jenkins Service

- Stop Jenkins:
- `sudo systemctl stop jenkins`
- Edit service configuration:
- `sudo systemctl edit jenkins`
- Add custom `JAVA_OPTS` or plugin directory if needed.
- Reload and restart Jenkins.

This allows changing:

- Java options
- Temp directory
- Plugin directory

4 Fix Directory Permissions

- Create required directories
- Change ownership:
- `sudo chown -R jenkins:jenkins /path`

Ensures Jenkins has proper access.

5 Start Jenkins & Check Logs

- Start service:
- `sudo systemctl start jenkins`
- View logs:
- `journalctl -u jenkins`

Confirms Jenkins started successfully.

6 Web UI Setup

1. Open browser:
2. `http://your-server-ip:8080`
3. Copy initial admin password from:
`/var/lib/jenkins/secrets/initialAdminPassword`
5. Paste password
6. Install suggested plugins
7. Create admin user

Now Jenkins dashboard is ready.



Sign in to Jenkins

Username

Password

Keep me signed in

sign in

Jenkins Freestyle Job – Step by Step

1 Create Job1 (Normal Freestyle Job)

1. Open Jenkins → Click **New Item**
2. Enter **Job1** as name → Select **Freestyle project** → Click **OK**
3. In **Build** section → Click **Add build step** → **Execute shell**
4. Enter commands:

```
echo "Hello World"  
pwd
```

5. Click **Save**

The screenshot shows the Jenkins interface for job 'Hello' (Build #2). The left sidebar has links for Status, Changes, Console Output (which is selected), Edit Build Information, Delete build '#2', Timings, and Previous Build. The right panel is titled 'Console' and displays the following log output:

```

Started by user SRIMATHI THIRUVELMANI
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/Hello
[Hello] $ /bin/sh -x /tmp/jenkins5819208804984540234.sh
+ echo helloworld
+ pwd
/var/lib/jenkins/workspace/Hello
Finished: SUCCESS

```

2 □ Run Job1

- Click **Build Now**
- Check **Console Output**
- Hello World
- /var/lib/jenkins/workspace/job1

3 □ Trigger Job2 from Job1

1. Open **Job1 → Configure → Post-build Actions**
2. Select **Build other projects**
3. Enter **Job2** as the project to build
4. Save

The screenshot shows the Jenkins interface for job 'job1' (Build #2). The left sidebar has links for Status, Changes, Console Output (selected), Edit Build Information, Delete build '#2', Timings, and Previous Build. The right panel is titled 'Console' and displays the following log output:

```

Started by user SRIMATHI THIRUVELMANI
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/job1
[job1] $ /bin/sh -xe /tmp/jenkins6310502584247468415.sh
+ echo this is job1
this is job1
Triggering a new build of job2
Finished: SUCCESS

```

Pipeline Implementation

Instead of multiple freestyle jobs, we can use a Pipeline.

Example Pipeline Script

```

pipeline {
    agent any
    stages {
        stage('Clone') {
            steps {
                git branch: 'main', url: 'https://github.com/yourrepo.git'
            }
        }
        stage('Build Docker Image') {

```

```

        steps {
            sh 'docker build -t webapp .'
        }
    }
    stage('Run Container') {
        steps {
            sh 'docker run -d -p 5000:5000 webapp'
        }
    }
}
}

```

Possible Output

- Pipeline starts
- Git repository cloned
- Docker image built
- Container started
- Status: SUCCESS

Build Queue Workflow (Chained Jobs)

1. Builder Job

Clones repository:

```
git clone https://github.com/yourrepo.git
cd project-folder
```

2. Image Job

Build Docker image:

```
docker build -t appimage .
```

3. Run Job

Run container:

```
docker run -d -p 8050:5000 appimage
```

This demonstrates job chaining:
Builder → Image → Run