

# JENKINS ASSIGNMENT

## 1. Build and Deploy a Simple App using Jenkins

### Step 1: Install Required Plugins

1. Open Jenkins Dashboard → Manage Jenkins → Manage Plugins.

2. Install the following plugins:

- Pipeline
- Git Plugin
- Docker Pipeline Plugin (if using Docker)

### Step 2: Create a New Jenkins Pipeline Job

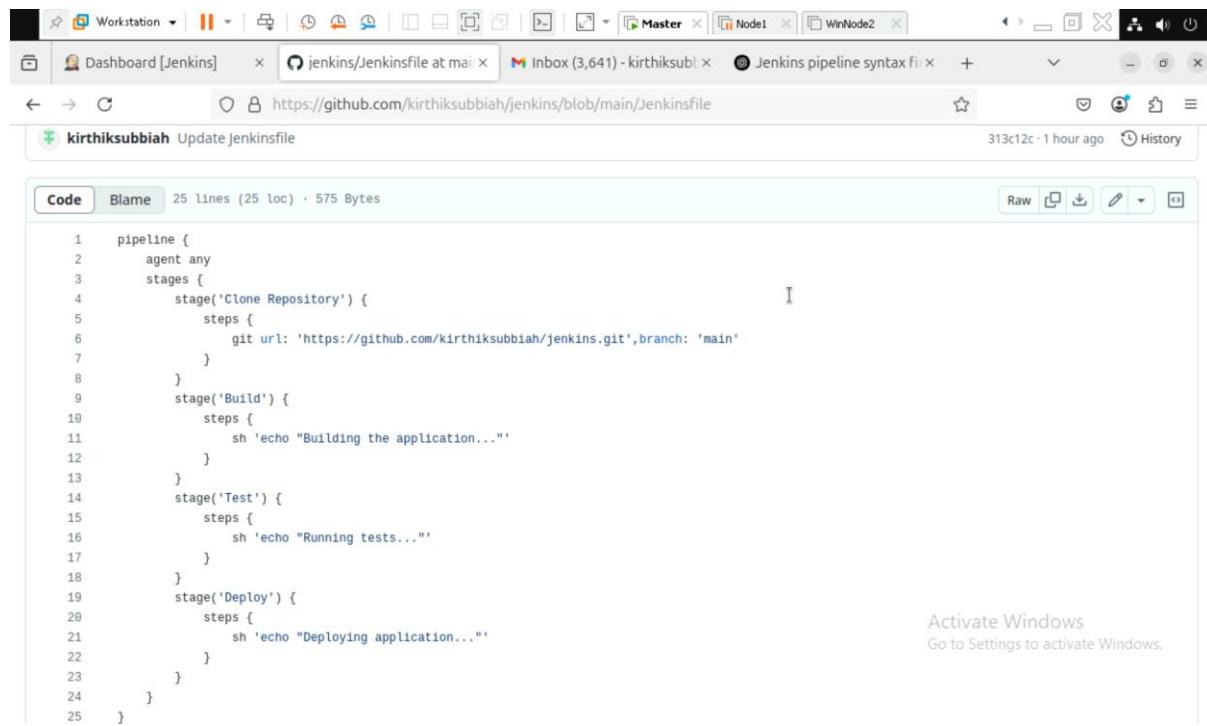
1. In Jenkins, click **New Item** → **Pipeline** → Give it a name → Click **OK**.

2. Scroll down to the **Pipeline** section and select **Pipeline script from SCM**.

3. In the **SCM** field, select **Git**, then enter your **GitHub repository URL**.

4. In the **Branch Specifier**, enter `*/main` or `*/master`.

5. In the **Script Path**, enter `Jenkinsfile` (which we will create in the next step).



The screenshot shows a GitHub code editor window with the URL <https://github.com/kirthiksubbiah/jenkins/blob/main/Jenkinsfile>. The Jenkinsfile contains the following Groovy script:

```
1 pipeline {
2     agent any
3     stages {
4         stage('Clone Repository') {
5             steps {
6                 git url: 'https://github.com/kirthiksubbiah/jenkins.git', branch: 'main'
7             }
8         }
9         stage('Build') {
10            steps {
11                sh 'echo "Building the application..."'
12            }
13        }
14        stage('Test') {
15            steps {
16                sh 'echo "Running tests..."'
17            }
18        }
19        stage('Deploy') {
20            steps {
21                sh 'echo "Deploying application..."'
22            }
23        }
24    }
25 }
```

At the bottom right of the code editor, there is a watermark that says "Activate Windows Go to Settings to activate Windows."

## Build History:

The screenshot shows the Jenkins Build History page. On the left, there's a sidebar with links like 'New Item', 'Build History' (which is selected and highlighted in grey), 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'My Views', 'Open Blue Ocean', and 'Lockable Resources'. Below the sidebar, there's a 'Build Queue' section stating 'No builds in the queue.' and a 'Build Executor Status' section. The main area is titled 'Build History of Jenkins' and lists four builds for the 'demo' project:

| S | Build   | Time Since  | Status                  |
|---|---------|-------------|-------------------------|
| 1 | demo #6 | 1 hr 42 min | back to normal          |
| 2 | demo #5 | 2 hr 6 min  | broken for a long time  |
| 3 | demo #4 | 2 hr 11 min | broken for a long time  |
| 4 | demo #3 | 2 hr 13 min | broken since this build |

On the right side of the main area, there's a message: 'Activate Windows Go to Settings to activate Windows.' with a small link icon.

## Status of Each stages of the pipeline:

The screenshot shows the Jenkins Pipeline Stage View for the 'demo' job. On the left, there's a sidebar with links like 'Status' (selected and highlighted in grey), 'Changes', 'Build Now', 'Configure', 'Delete Pipeline', 'Full Stage View', 'Favorite', 'Open Blue Ocean', 'Stages', 'Rename', and 'Pipeline Syntax'. Below the sidebar, there's a 'Builds' section listing builds from today: #7 5:44 PM, #6 3:59 PM, #5 3:35 PM, #4 3:31 PM, and #3 3:28 PM. The main area is titled 'Stage View' and displays a grid of stages for each build. The stages are: Declarative: Checkout SCM, Clone Repository, Build, Test, and Deploy. The 'Build' stage is currently selected. The grid shows run times for each stage across the five builds. A message on the right says: 'Activate Windows Go to Settings to activate Windows.'

Workspace:

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links like 'New Item', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'My Views', 'Open Blue Ocean', and 'Lockable Resources'. The main area displays a table for the 'demo' project. The table has columns for Status (S), Workstation (W), Name (Name), Last Success, Last Failure, Last Duration, and Filter (F). The 'demo' row shows a green checkmark icon, a cloud icon, the name 'demo', 'Last Success: 1 hr 33 min #6', 'Last Failure: 1 hr 57 min #5', 'Last Duration: 9.2 sec', and a green arrow and star icon. Below the table, there are dropdowns for 'Build Queue' (No builds in the queue) and 'Build Executor Status'. A message on the right says 'Activate Windows Go to Settings to activate Windows.'

Status of the Pipeline:

The screenshot shows the Jenkins job details for '#6'. The top navigation bar includes links for 'demo #6 [Jenkins]', 'jenkins/Jenkinsfile at main', 'Inbox (3,641) - kirthiksubbiah', and 'Jenkins pipeline syntax file'. The main content area shows the job status as 'Status #6 (Feb 19, 2025, 3:59:44 PM)'. It includes sections for 'Changes', 'Console Output', 'Edit Build Information', 'Delete build #6', 'Timings', 'Git Build Data', 'Lockable resources', 'Open Blue Ocean', 'Pipeline Overview', and 'Pipeline Console'. A summary section indicates the job was started by user 'admin' 1 hr 8 min ago and took 9.2 sec. It also shows the revision (313c12c67e2dc2433e2c657e56334f8bdec4f5c) and repository (https://github.com/kirthiksubbiah/jenkins.git). A 'Changes' section lists '1. Update Jenkinsfile (details / githubweb)'. A message on the right says 'Activate Windows Go to Settings to activate Windows.'

## Step 3: Create a Jenkinsfile (CI/CD Pipeline)

In your GitHub repository, create a new file named **Jenkinsfile** and add the following code:

```

groovy
CopyEdit
pipeline {
    agent any
    stages {
        stage('Clone Repository') {
            steps {
                git 'https://github.com/your-username/your-repo.git'
            }
        }
        stage('Build') {
            steps {
                sh 'echo "Building the application..."'
            }
        }
        stage('Test') {
            steps {
                sh 'echo "Running tests..."'
            }
        }
        stage('Deploy') {
            steps {
                sh 'echo "Deploying application..."'
            }
        }
    }
}

```

### **Explanation:**

- **Stage 1: Clone Repository** → Pulls the latest code from GitHub.
- **Stage 2: Build** → Builds the application (placeholder command).
- **Stage 3: Test** → Runs tests (placeholder command).
- **Stage 4: Deploy** → Deploys the app (placeholder command).

## **Step 4: Build and Run the Pipeline**

1. Go back to **Jenkins Dashboard** → Click on your job.
2. Click **Build Now**.
3. Click **Console Output** to see the progress.

If everything is successful, you will see all **stages completed** in the output.

## **Step 5: Deploy with Docker (Optional)**

If you want to deploy your application as a **Docker container**, modify your **Jenkinsfile** like this:

```
groovy
CopyEdit
pipeline {
    agent any
    environment {
        IMAGE_NAME = 'your-dockerhub-username/my-app'
    }
    stages {
        stage('Clone Repository') {
            steps {
                git 'https://github.com/your-username/your-repo.git'
            }
        }
        stage('Build Docker Image') {
            steps {
                sh 'docker build -t $IMAGE_NAME .'
            }
        }
        stage('Push Docker Image') {
            steps {
                withDockerRegistry([credentialsId: 'docker-hub-credentials']) {
                    sh 'docker push $IMAGE_NAME'
                }
            }
        }
        stage('Deploy') {
            steps {
                sh 'docker run -d -p 8080:80 $IMAGE_NAME'
            }
        }
    }
}
```

### Explanation:

- **Builds a Docker image** of your application.
  - **Pushes it to Docker Hub** (requires credentials setup in Jenkins).
  - **Deploys the container** on a local machine or server.
- 
- You have successfully set up **Jenkins CI/CD** to **build, test, and deploy** a simple app!
  - If using **Docker**, your app is now running on **http://localhost:8080**.

## **2. Automating a Python Application Build using Jenkins.**

### **Step 1: Install Jenkins and Required Plugins**

1. Start the Jenkins service and log in to the Jenkins dashboard.
2. Navigate to Manage Jenkins → Plugins.
3. Install the following plugins:
  - o Pipeline Plugin (for defining Jenkins Pipelines)
  - o Git Plugin (for pulling code from GitHub)
  - o Build Tools Plugin (for executing builds)
4. Restart Jenkins to apply the changes.

### **Step 2: Create a New Jenkins Job**

1. Open Jenkins Dashboard.
2. Click on New Item → Enter a project name.
3. Choose **Freestyle Project** or **Pipeline Project**.
4. Click **OK** to create the project.

| S | W | Name ↓                            | Last Success    | Last Failure | Last Duration | F |
|---|---|-----------------------------------|-----------------|--------------|---------------|---|
|   |   | <a href="#">My_First_Pipeline</a> | 18 hr #10       | 18 hr #9     | 26 sec        |   |
|   |   | <a href="#">Python-App-Build</a>  | 6 min 13 sec #1 | N/A          | 8.9 sec       |   |

### **Step 3: Configure Jenkins to Pull Code from GitHub**

1. Open the newly created Jenkins job.
2. Navigate to **Source Code Management**.
3. Select **Git** and enter the repository URL:
  - o Example: <https://github.com/your-repo/python-app.git>
4. Configure GitHub credentials if authentication is required.

The screenshot shows the Jenkins Pipeline configuration page. On the left, there's a sidebar with tabs: General, Triggers, Pipeline (which is selected), and Advanced. The main area is titled 'Pipeline' with the sub-instruction 'Define your Pipeline using Groovy directly or pull it from source control.' Below this is a 'Definition' dropdown set to 'Pipeline script from SCM'. Under 'SCM', it says 'Git'. A 'Repositories' section shows a single entry with 'Repository URL' set to 'https://github.com/Santhosh2010-ramesh/Jenkins.git'. There are 'Save' and 'Apply' buttons at the bottom.

## Step 4: Install Dependencies and Run Tests

Step 1: Adding the desired branch here we use Main branch.

This screenshot shows the 'Branches to build' section of the Jenkins pipeline configuration. It has a 'Branch Specifier (blank for 'any')' input field containing '/main'. A red 'X' icon is visible in the top right corner of the input field.

## Step 5: Pipeline with Python Scripts.

Step 1: Building the Declarative: Checkout SCM with the Script.

This screenshot shows the GitHub repository page for 'Santhosh2010-ramesh/Jenkins'. The 'Jenkinsfile' is open in the code editor. The file contains Groovy declarative pipeline code:

```

1 pipeline {
2   agent any
3
4   stages {
5     stage('Clone Repository') {
6       steps {
7         git url: 'https://github.com/Santhosh2010-ramesh/Jenkins.git', branch: 'main'
8       }
9     }
10
11    stage('Set Up Virtual Environment') {
12      steps {
13        // Ensure python3-venv is available
14        sh 'python3 -m venv venv'
15        sh '. venv/bin/activate' // Activate the virtual environment
16      }
17    }
18
19    stage('Install Dependencies') {
20      steps {
21        // Install dependencies inside the virtual environment
22        sh 'venv/bin/activate && pip install -r requirements.txt'
23      }
24    }
25
26    stage('Run Tests') {

```

A red 'Activate Windows' message is visible at the bottom right of the code editor.

Step 2: Sample Python file.

Code Blame 5 lines (4 loc) · 92 Bytes

```
1 def hello_world():
2     print("Hello, World!")
3
4 if __name__ == "__main__":
5     hello_world()
```

### Step 3: Creating Requirements.txt file.

Files Jenkins / requirements.txt

Santhosh2010-ramesh Update requirements.txt 90d1105 · 6 minutes ago History

Code Blame 3 lines (3 loc) · 27 Bytes

```
1 requests
2 pytest
3 setuptools
```

### Step 4: Creating a Setup.py file.

Jenkins / setup.py

Santhosh2010-ramesh Update setup.py 9ed3fed · 6 minutes ago History

Code Blame 10 lines (9 loc) · 174 Bytes

```
1 from setuptools import setup, find_packages
2
3 setup(
4     name='python-app',
5     version='0.1',
6     packages=find_packages(),
7     install_requires=[
8         'pytest',
9     ],
10 )
```

### Step 5: Creating a sample test.py for checkrun.

The screenshot shows a GitHub file viewer for the file `test/test_app.py`. The left sidebar shows a tree view of the repository structure, including `main`, `test` (which contains `test_app.py`), `Jenkinsfile`, `Readme`, `app.py`, `requirements.txt`, and `setup.py`. The right panel displays the content of `test_app.py`:

```

1 # test_app.py
2 from app import hello_world
3 import pytest
4
5 def test_hello_world(capfd):
6     hello_world()
7     captured = capfd.readouterr()
8     assert captured.out == "Hello, World!\n"

```

Below the code, there are buttons for `Raw`, `Copy`, `Edit`, and `Download`. At the bottom right of the page, there is a message: "Activate Windows Go to Settings to activate Windows."

## Step 6: Build and Archive Artifacts with Declarative: Checkout SCM, Clone Repository, set up Virtual Environment, Install Dependencies, Run Tests, Build Artifact, Archive Artifact.

The screenshot shows the Jenkins Pipeline Stage View for the pipeline `Python-App-Build`. The top navigation bar includes the Jenkins logo, a search icon, a user icon for `admin`, and a `logout` link. The left sidebar contains links for `Status`, `Changes`, `Build Now`, `Configure`, `Delete Pipeline`, `Full Stage View`, `Favorite`, `Open Blue Ocean`, and `Stages`. The main area shows the `Stage View` for the `Python-App-Build` pipeline. The stages are listed in a grid:

|   | Declarative: Checkout SCM | Clone Repository | Set Up Virtual Environment | Install Dependencies | Run Tests | Build Artifact | Archive Artifact |
|---|---------------------------|------------------|----------------------------|----------------------|-----------|----------------|------------------|
| Average stage times:<br>(full run time: ~13s) | 1s                        | 1s               | 3s                         | 5s                   | 539ms     | 205ms          | 356ms            |
| #15<br>12:52<br>1 commit                      | 1s                        | 1s               | 4s                         | 1s                   | 735ms     | 746ms          | 1s               |

Jenkins / test / test\_app.py

Santhosh2010-ramesh Update and rename test.py to test\_app.py a0917e1 · 5 minutes ago History

Code Blame 8 lines (7 loc) · 191 Bytes

```
1 # test_app.py
2 from app import hello_world
3 import pytest
4
5 def test_hello_world(capfd):
6     hello_world()
7     captured = capfd.readouterr()
8     assert captured.out == "Hello, World!\n"
```

## Step 7: Run the Jenkins Job

1. Click **Build Now** to trigger a build.
2. Monitor logs in **Console Output**.

The screenshot shows the Jenkins interface with the URL `localhost:8080/job/Python-App-Build/15/console`. The left sidebar has links like Status, Changes, Console Output (which is selected), Edit Build Information, Delete build '#15', Timings, Git Build Data, See Fingerprints, Open Blue Ocean, Pipeline Overview, Pipeline Console, Restart from Stage, and Replay. The main area is titled 'Console Output' and shows the build log:

```
Started by user admin
Obtained Jenkinsfile from git https://github.com/Santhosh2010-ramesh/Jenkins.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on jenkins-slave1 in /opt/jenkins-slave1/workspace/Python-App-Build
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
Selected Git installation does not exist. Using Default
The recommended git tool is: NONE
No credentials specified
Fetching changes from the remote Git repository
> git rev-parse --resolve-git-dir /opt/jenkins-slave1/workspace/Python-App-Build/.git # timeout=10
> git config remote.origin.url https://github.com/Santhosh2010-ramesh/Jenkins.git # timeout=10
Fetching upstream changes from https://github.com/Santhosh2010-ramesh/Jenkins.git
Go to Settings to activate Windows.
> git --version # timeout=10
> git --version # 'git version 2.43.0'
> git fetch --tags --force --progress -- https://github.com/Santhosh2010-ramesh/Jenkins.git +refs/heads/*
:refs/remotes/origin/* # timeout=10
```