Aim: Jenkins installation and setup, explore the environment. Demonstrate continuous integration and development using Jenkins.

Description:

What is Jenkins?

Jenkins is an open-source automation tool written in Java programming language that allows continuous integration.

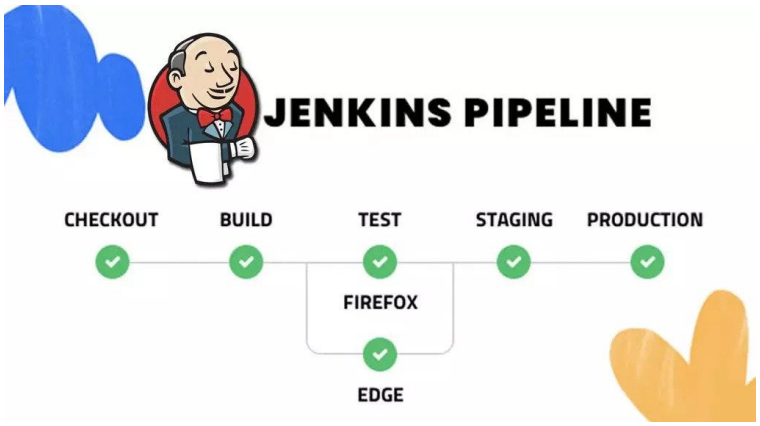
Jenkins **builds** and **tests** our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

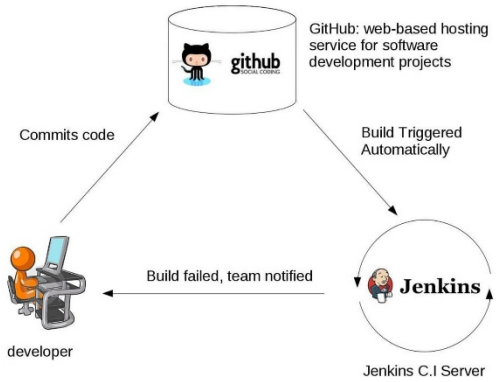
It also allows us to continuously **deliver** our software by integrating with a large number of testing and deployment technologies.

Jenkins offers a straightforward way to set up a continuous integration or continuous delivery environment for almost any combination of languages and source code repositories using pipelines, as well as automating other routine development tasks.

With the help of Jenkins, organizations can speed up the software development process through automation. Jenkins adds development life-cycle processes of all kinds, including build, document, test, package, stage, deploy static analysis and much more.

Jenkins achieves CI (Continuous Integration) with the help of plugins. Plugins is used to allow the integration of various DevOps stages. If you want to integrate a particular tool, you have to install the plugins for that tool. For example: Maven 2 Project, Git, HTML Publisher, Amazon EC2, etc.





Implementation:

Step 1: Install Jenkins on Windows

1. Download and Install Java (Prerequisite)
   Jenkins requires **Java 11 or later**.

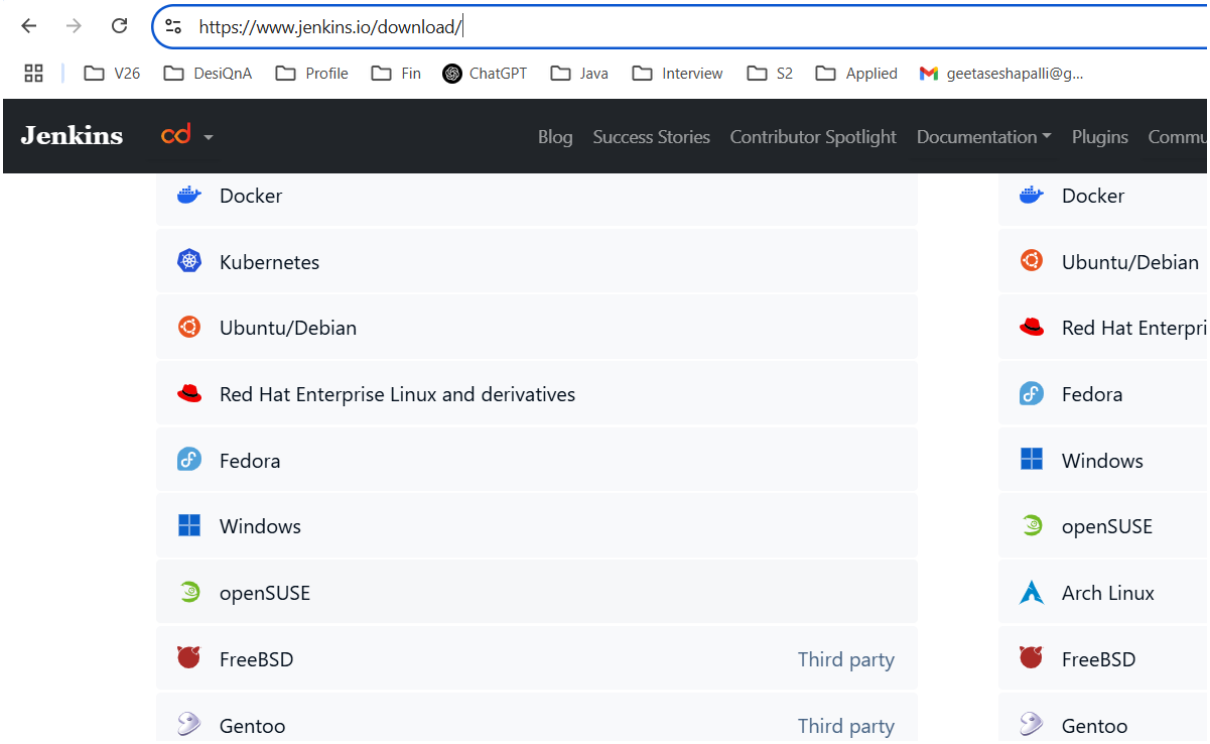
- Download and install Java:
  Oracle JDK
  OR
  OpenJDK
- Verify Java installation: java -version

```
C:\Users\geeta>java -version
java version "17" 2021-09-14 LTS
Java(TM) SE Runtime Environment (build 17+35-LTS-2724)
Java HotSpot(TM) 64-Bit Server VM (build 17+35-LTS-2724, mixed mode, sharing)

C:\Users\geeta>
```

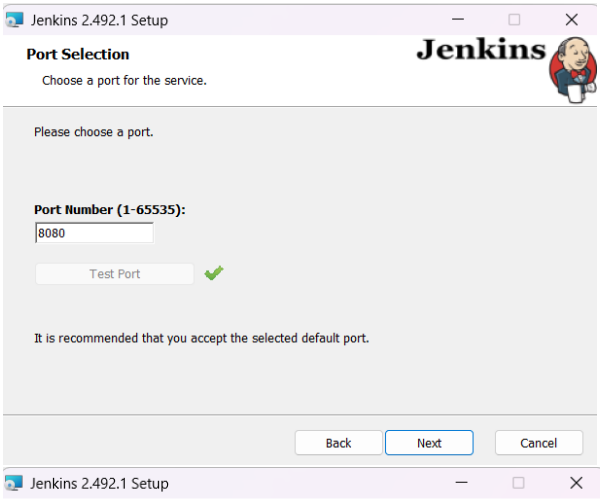
2. Download and Install Jenkins

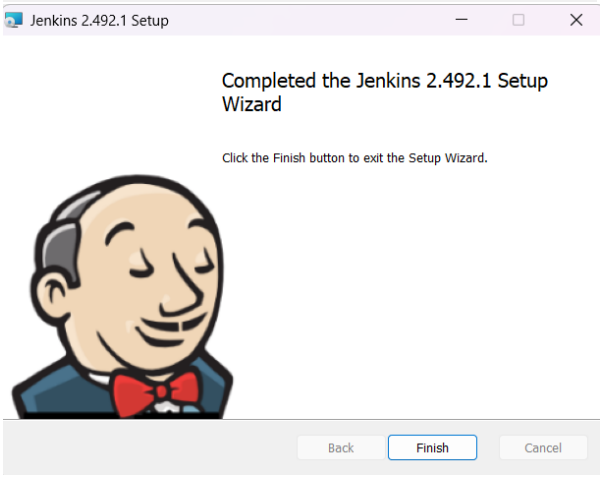
- Download Jenkins **Windows Installer** from: https://www.jenkins.io/download/



- Install Jenkins by running the .msi file.

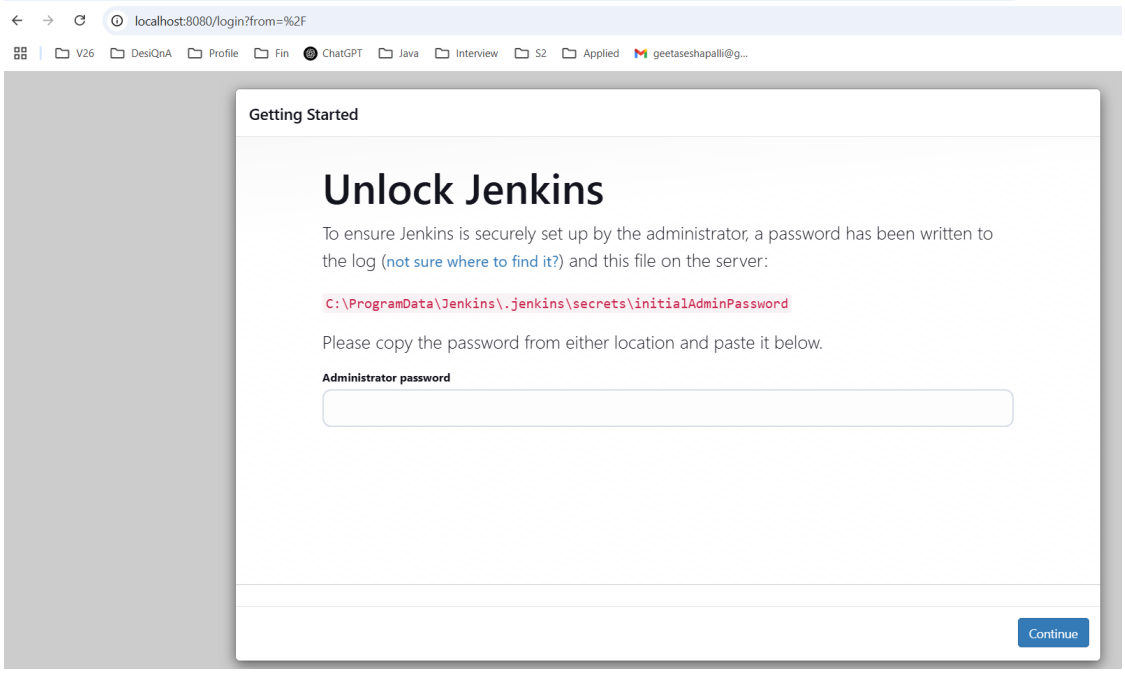
- During installation:
  - Choose **Run as a Service** (recommended).
  - The default installation path is: C:\Program Files\Jenkins
  - Jenkins runs on port **8080** by default.



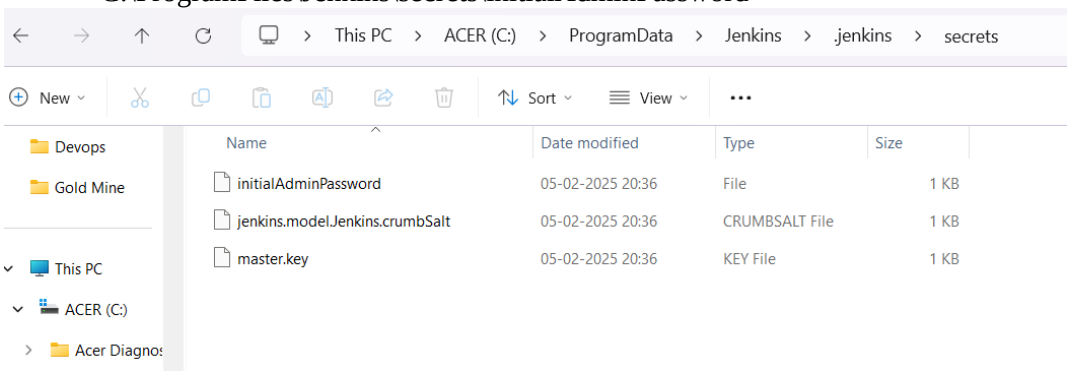


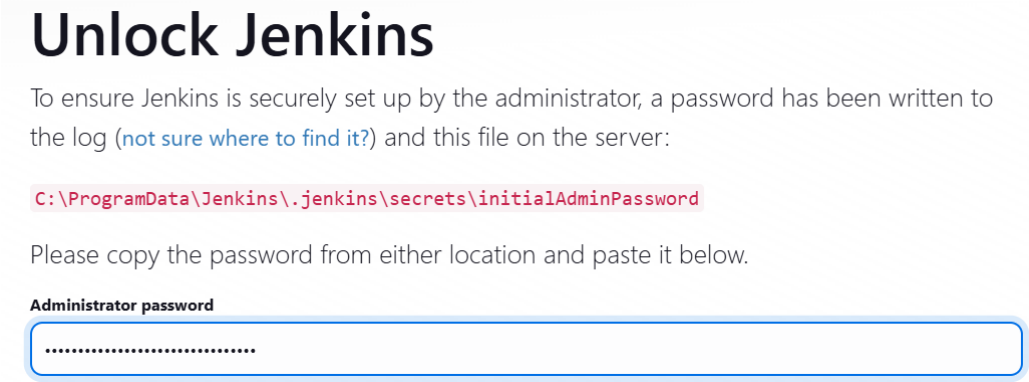
Step 2: Start and Access Jenkins

- Open a browser and go to: http://localhost:8080

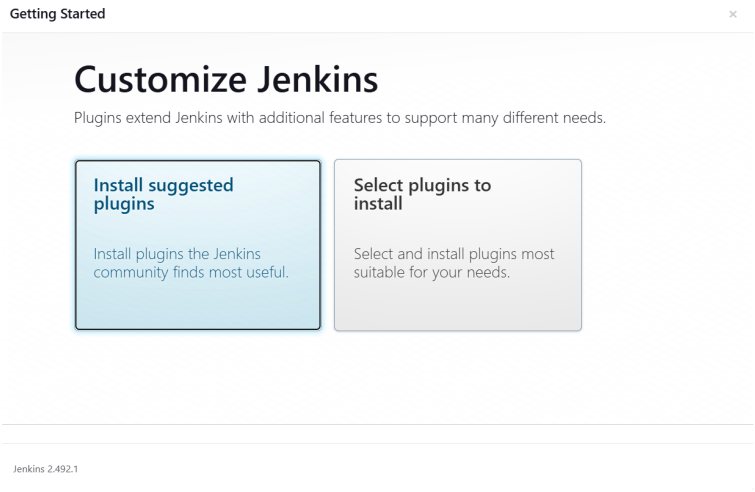


- Unlock Jenkins:
  - Get the **initial admin password** from: C:\ProgramFiles\Jenkins\secrets\initialAdminPassword

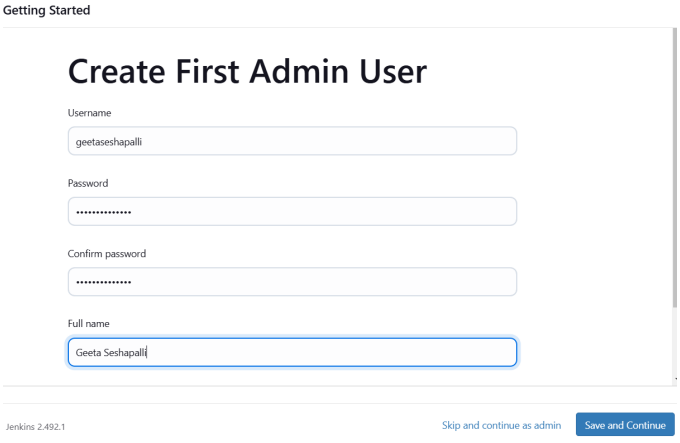


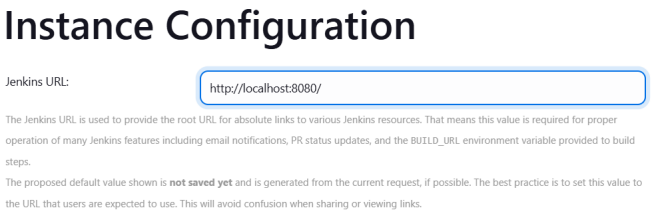


- Install recommended plugins.



- Create an admin user and complete the setup.





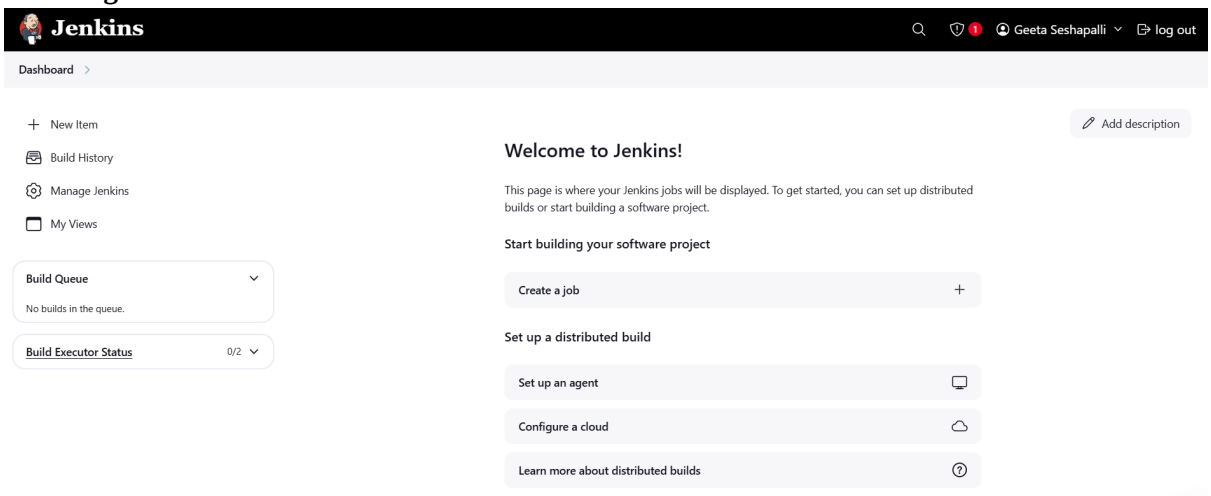
Step 3: Configure Git and GitHub for Jenkins

1. Install Git on Windows

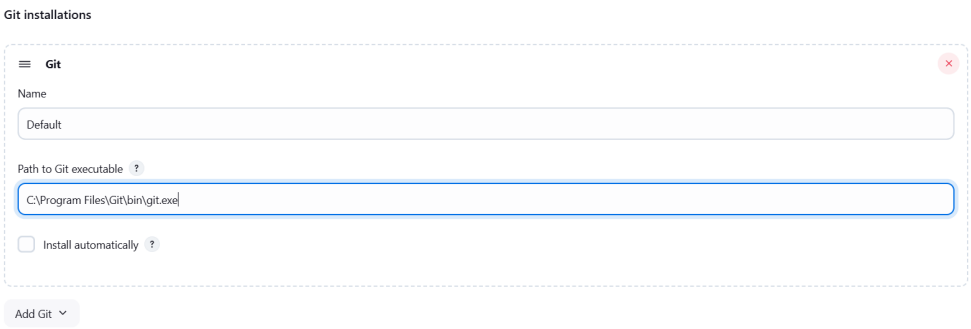
- Download Git: https://git-scm.com/downloads
- Verify installation: git --version

```
PS C:\Users\geeta> git --version
git version 2.47.1.windows.1
PS C:\Users\geeta>
```

2. Configure Git in Jenkins



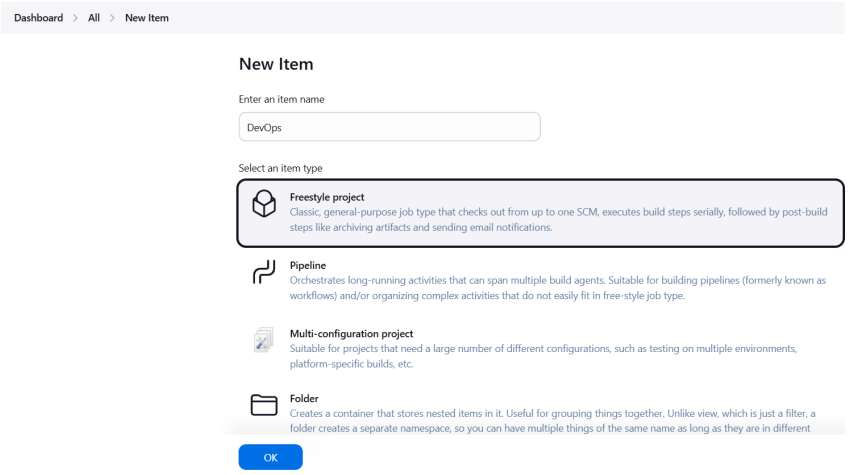
- Go to **Manage Jenkins → Global Tool Configuration**
- Under **Git**, set:
  - **Path to Git executable**:



Step 4: Set Up a Jenkins Job for CI/CD

1. Create a New Job

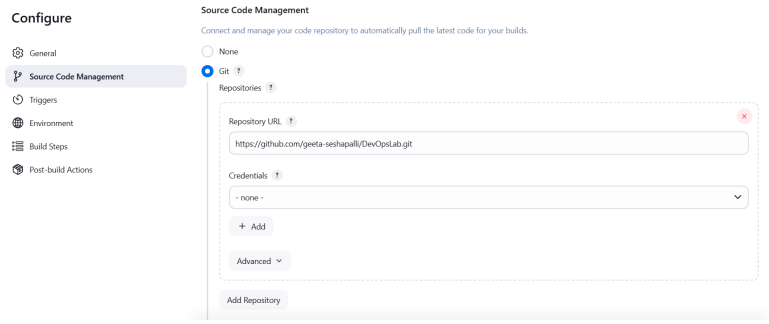
- In Jenkins, go to **Dashboard → New Item → Freestyle Project**.
- Name the project (e.g., DevOps).



-

2. Configure GitHub Repository

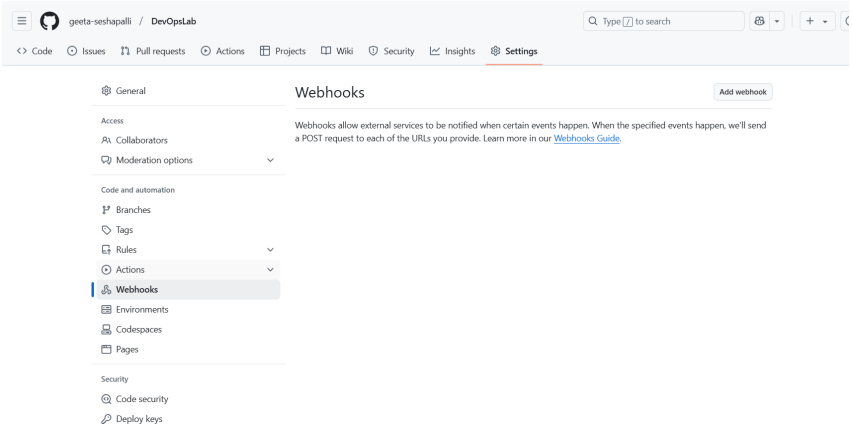
- In the **Source Code Management** section:
  - Select **Git**.
  - Enter your **GitHub repository URL**.



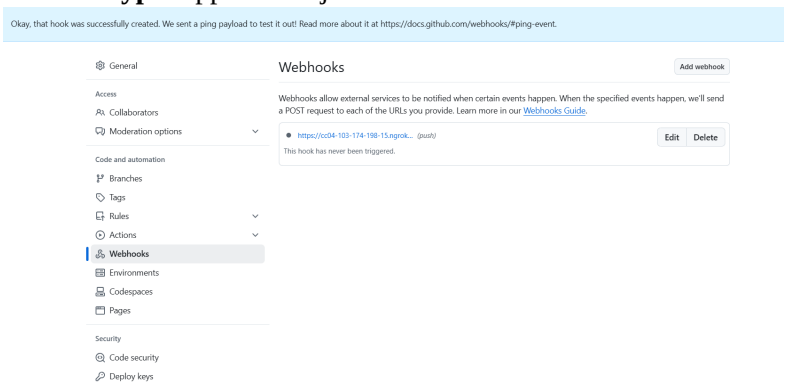
3. Configure Build Triggers

To trigger Jenkins when you push code to GitHub:

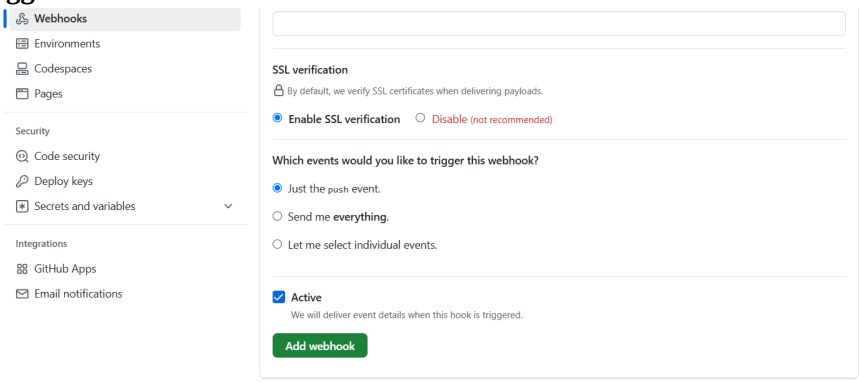
1. In GitHub:
   - Go to **Repository Settings → Webhooks**.



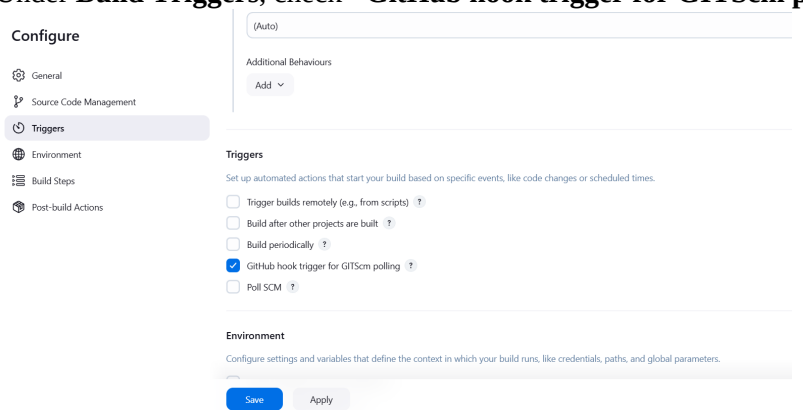
   - Click **Add Webhook**:
     - **Payload URL**:http://your-jenkins-url/github-webhook/
     - **Content type:** application/json



- **Trigger:** Just the "**Push**" event.



2. In Jenkins:
   - Under **Build Triggers**, check **"GitHub hook trigger for GITScm polling"**.

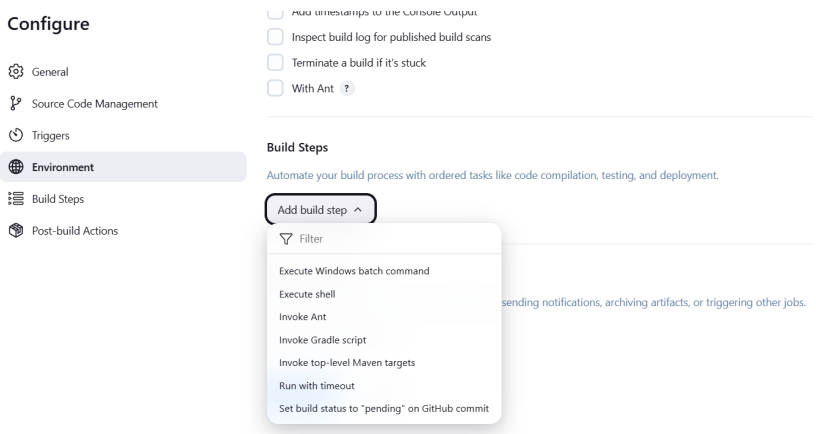


Step 4: Add Build Steps

You can now configure Jenkins to **build, test, or deploy** your project.

1. Add a Simple Build Step

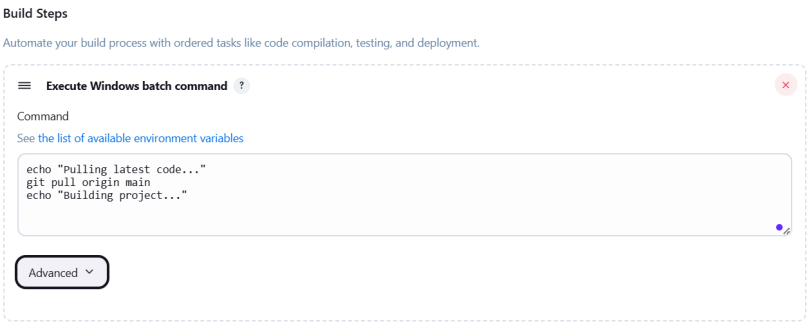
Under **Build → Add build step → Execute Windows Batch Command**, add:



echo "Pulling latest code..."

git pull origin main

echo "Building project..."



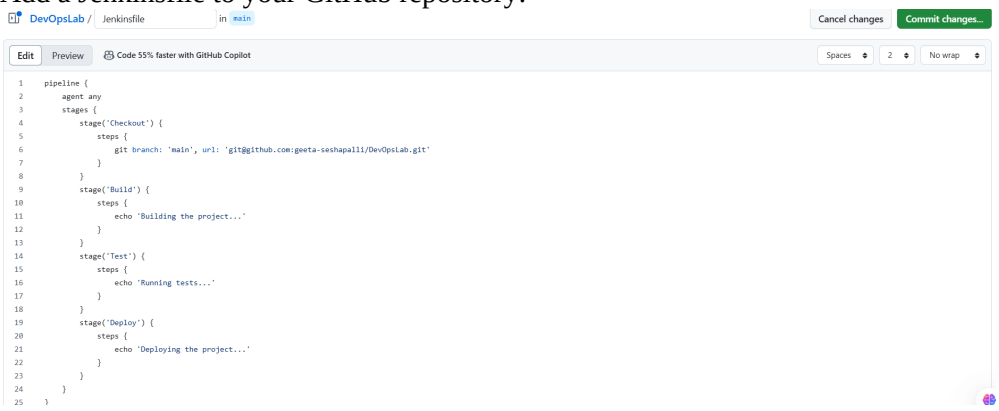
2. Save and Build

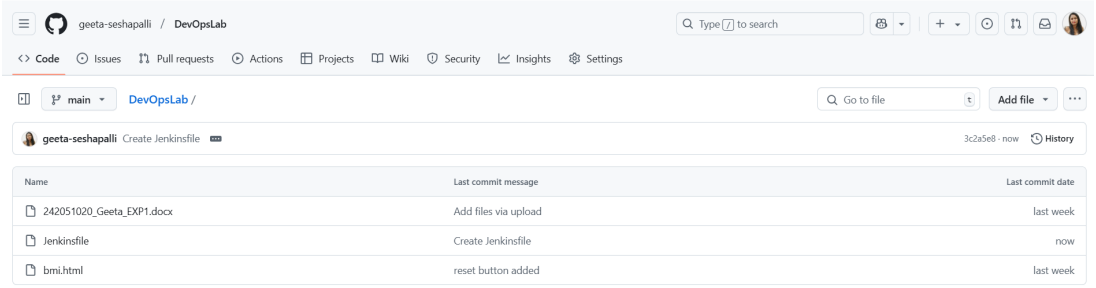
- Click **Save**.
- Click **Build Now** to test the setup.

Step 5: Set Up Jenkins Pipeline

If you want a **Jenkinsfile**-based pipeline instead of freestyle jobs:

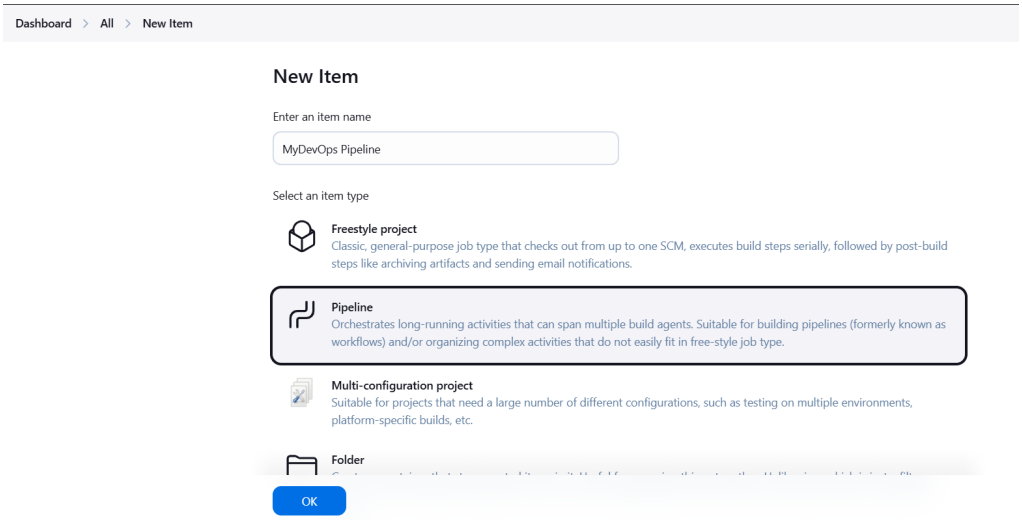
Add a Jenkinsfile to your GitHub repository:





Step 6: Create a New Pipeline Job in Jenkins

1. Open your **Jenkins dashboard**.
2. On the left-hand side, click on **New Item**.
3. In the **Enter an item name** field, give your job a name (e.g., My-Pipeline).
4. Select **Pipeline** from the available options.
5. Click **OK**.



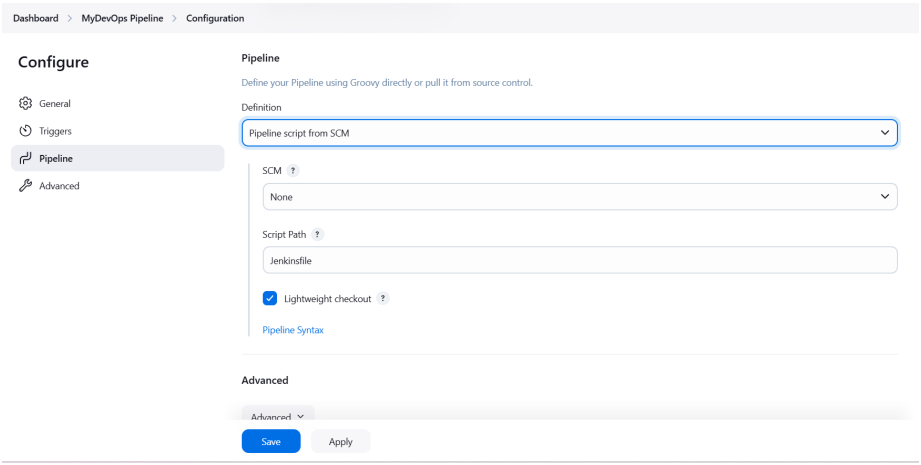
6.

Step 8: Configure the Pipeline Job

1. You will now be directed to the **Configure** page for your new pipeline job.
2. Scroll down to the **Pipeline** section.

Step 9: Set Pipeline Definition to "Pipeline script from SCM"

1. Under the **Pipeline** section, you will see a field labeled **Definition**.
2. Select **Pipeline script from SCM** from the dropdown.

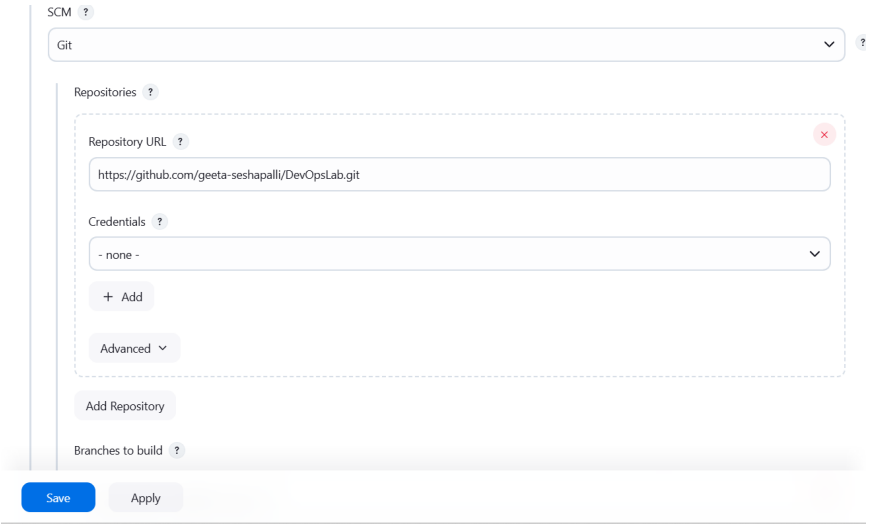


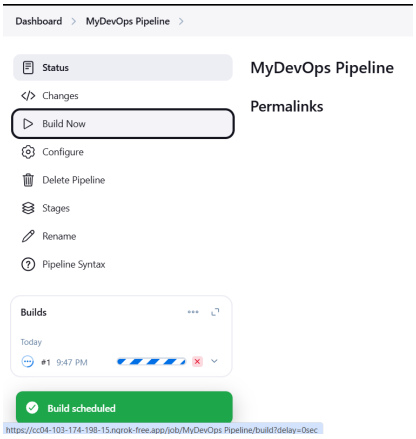
Step 10: Configure SCM Settings

1. Once "Pipeline script from SCM" is selected, the **SCM** section will appear.
2. Select **Git** from the **SCM** dropdown.
3. **Enter the Repository URL**:
   - In the **Repository URL** field, enter the URL of your Git repository

Step 11: Save and Build

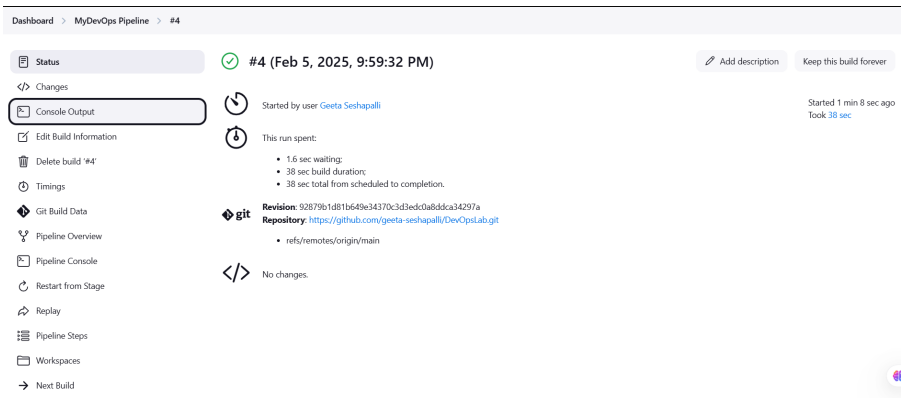
1. After configuring everything, scroll down to the bottom and click **Save**.
2. To build the pipeline, click **Build Now** on the left-hand side.





Step 12: Verify CI/CD Workflow

- Push a commit to your **GitHub repository**.
- Check Jenkins → The job should trigger automatically.
- View **Console Output** to verify the build logs.



Dashboard > MyDevOps Pipeline > #4

- Status
- Changes
- Console Output
- Edit Build Information
- Delete build '#4'
- Timings
- Git Build Data
- Pipeline Overview
- Pipeline Console
- Restart from Stage
- Replay
- Pipeline Steps
- Workspaces
- Next Build

Console Output

Download Copy View as plain te

```
Started by user Geeta Seshapalli
Obtained Jenkinsfile from git https://github.com/geeta-seshapalli/DevOpsLab.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in C:\ProgramData\Jenkins\.jenkins\workspace\MyDevOps Pipeline@2
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Declarative: Checkout SCM)
[Pipeline] checkout
The recommended git tool is: NONE
No credentials specified
Cloning the remote Git repository
Cloning repository https://github.com/geeta-seshapalli/DevOpsLab.git
 > C:\Program Files\Git\bin\git.exe init C:\ProgramData\Jenkins\.jenkins\workspace\MyDevOps Pipeline@2 # timeout=10
Fetching upstream changes from https://github.com/geeta-seshapalli/DevOpsLab.git
 > C:\Program Files\Git\bin\git.exe --version # timeout=10
 > git --version # 'git version 2.47.1.windows.1'
 > C:\Program Files\Git\bin\git.exe fetch --tags --force --progress -- https://github.com/geeta-seshapalli/DevOpsLab.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
 > C:\Program Files\Git\bin\git.exe config remote.origin.url https://github.com/geeta-seshapalli/DevOpsLab.git # timeout=10
 > C:\Program Files\Git\bin\git.exe config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* # timeout=10
Avoid second fetch
 > C:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/main^{commit}" # timeout=10
Checking out Revision 92879b1d81b649e34370c3d3edc0a8ddca34297a (refs/remotes/origin/main)
```

Dashboard > MyDevOps Pipeline > #4

```
 > C:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
 > C:\Program Files\Git\bin\git.exe checkout -f 92879b1d81b649e34370c3d3edc0a8ddca34297a # timeout=10
Commit message: "Update Jenkinsfile"
First time build. Skipping changelog.
[Pipeline] }
[Pipeline] // stage
[Pipeline] withEnv
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Checkout)
[Pipeline] git
The recommended git tool is: NONE
Warning: CredentialId "your-credentials-id" could not be found.
 > C:\Program Files\Git\bin\git.exe rev-parse --resolve-git-dir C:\ProgramData\Jenkins\.jenkins\workspace\MyDevOps Pipeline@2\.git #
timeout=10
Fetching changes from the remote Git repository
 > C:\Program Files\Git\bin\git.exe config remote.origin.url https://github.com/geeta-seshapalli/DevOpsLab.git # timeout=10
Fetching upstream changes from https://github.com/geeta-seshapalli/DevOpsLab.git
 > C:\Program Files\Git\bin\git.exe --version # timeout=10
 > git --version # 'git version 2.47.1.windows.1'
 > C:\Program Files\Git\bin\git.exe fetch --tags --force --progress -- https://github.com/geeta-seshapalli/DevOpsLab.git
+refs/heads/*:refs/remotes/origin/* # timeout=10
 > C:\Program Files\Git\bin\git.exe rev-parse "refs/remotes/origin/main^{commit}" # timeout=10
Checking out Revision 92879b1d81b649e34370c3d3edc0a8ddca34297a (refs/remotes/origin/main)
 > C:\Program Files\Git\bin\git.exe config core.sparsecheckout # timeout=10
```

Dashboard > MyDevOps Pipeline > #4

```
 > C:\Program Files\Git\bin\git.exe checkout -f 92879b1d81b649e34370c3d3edc0a8ddca34297a # timeout=10
 > C:\Program Files\Git\bin\git.exe branch -a -v --no-abbrev # timeout=10
 > C:\Program Files\Git\bin\git.exe checkout -b main 92879b1d81b649e34370c3d3edc0a8ddca34297a # timeout=10
Commit message: "Update Jenkinsfile"
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] echo
Building the project...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] echo
Running tests...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Deploying the project...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
```

Dashboard > MyDevOps Pipeline > #4

```
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] echo
Running tests...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Deploying the project...
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
Pipeline execution completed successfully.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Conclusion:

In this experiment, we have successfully set up Jenkins on Windows and integrated it with an existing GitHub repository for Continuous Integration (CI) and Continuous Deployment (CD). Here's a summary of the key steps and outcomes:

1. Jenkins Installation:
    - Installed Jenkins on a Windows machine and verified its successful startup.
    - Configured the necessary Java and Git prerequisites for Jenkins to function smoothly.
2. GitHub Integration:
    - Linked Jenkins to GitHub using SSH keys, allowing Jenkins to securely pull code from the repository.
    - Configured GitHub Webhooks to trigger Jenkins jobs automatically whenever new code is pushed to the repository.
3. CI/CD Pipeline:

- Set up a **Pipeline** using a Jenkinsfile, which defines various stages like **Build**, **Test**, and **Deploy**.
- This allows for automated testing and deployment whenever changes are pushed to the GitHub repository.

4. Continuous Integration (CI) Benefits:

- With each code push, Jenkins triggers the build process, ensuring that the latest code is continuously integrated and tested.
- Helps catch issues early in the development cycle through **automated testing** and **build verification**.

REPOSITORY LINK https://github.com/geeta-seshapalli/DevOpsLab.git