

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.

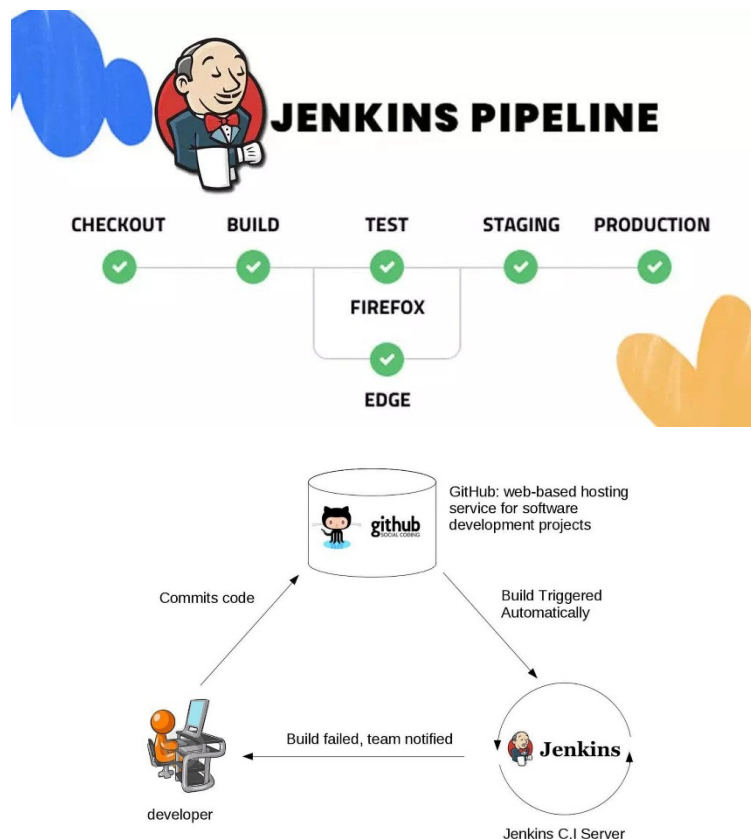
Jenkins **builds** and **tests** our software projects which continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.

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](#)

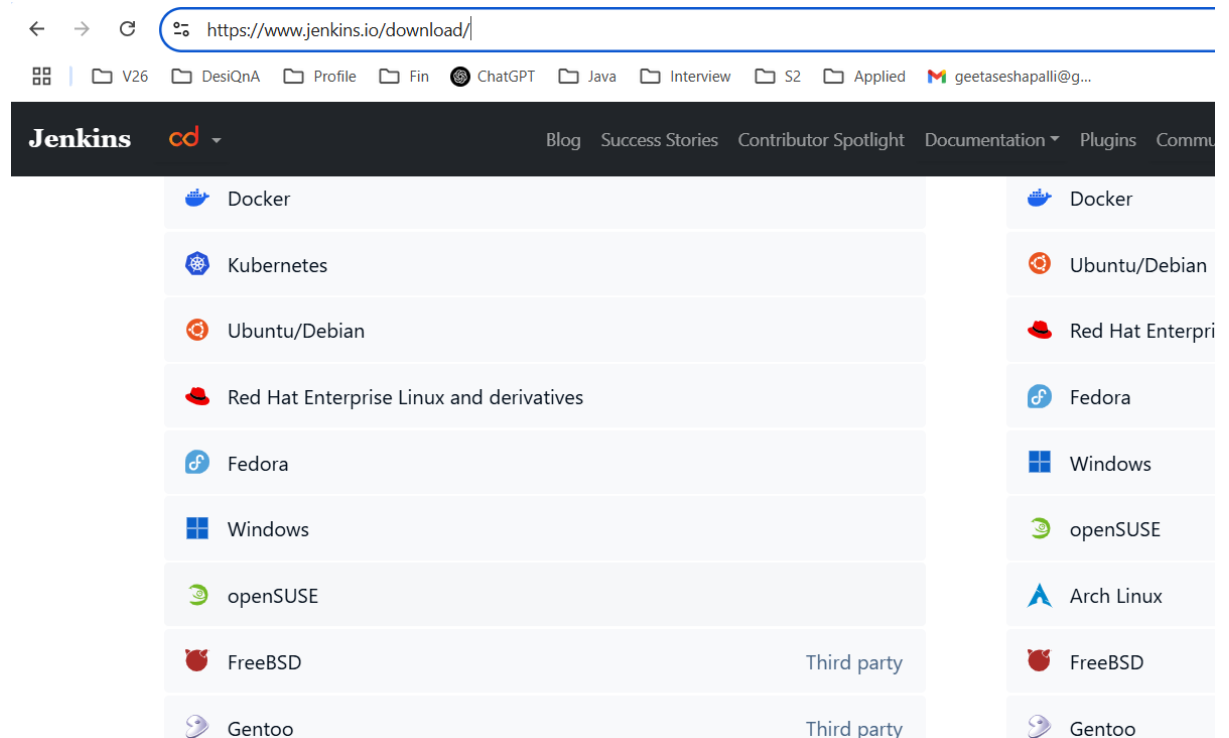
- Verify Java installation: `java -version`

```
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

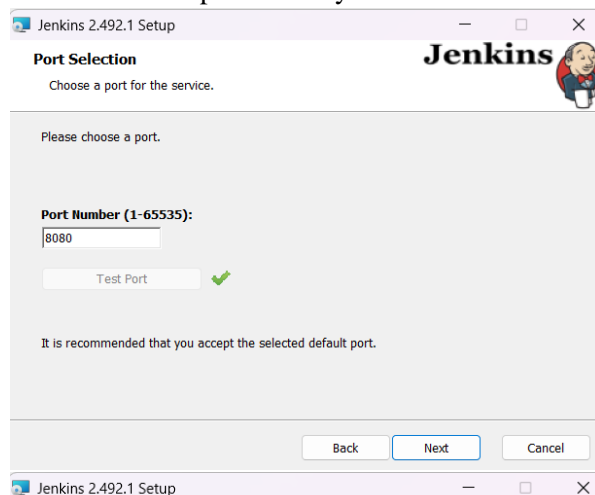
- Download Jenkins **Windows Installer** from: <https://www.jenkins.io/download/>



- Install Jenkins by running the .msi file.

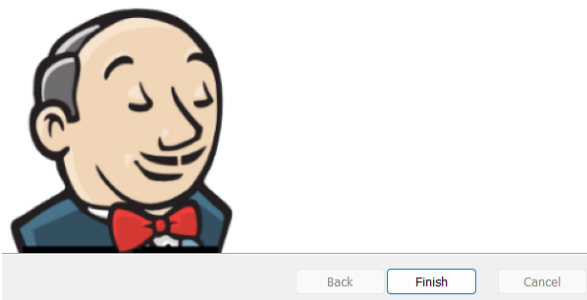


- During installation:
 - Choose **Run as a Service** (recommended).
 - The default installation path is:
C:\Program Files\Jenkins
 - Jenkins runs on port **8080** by default.



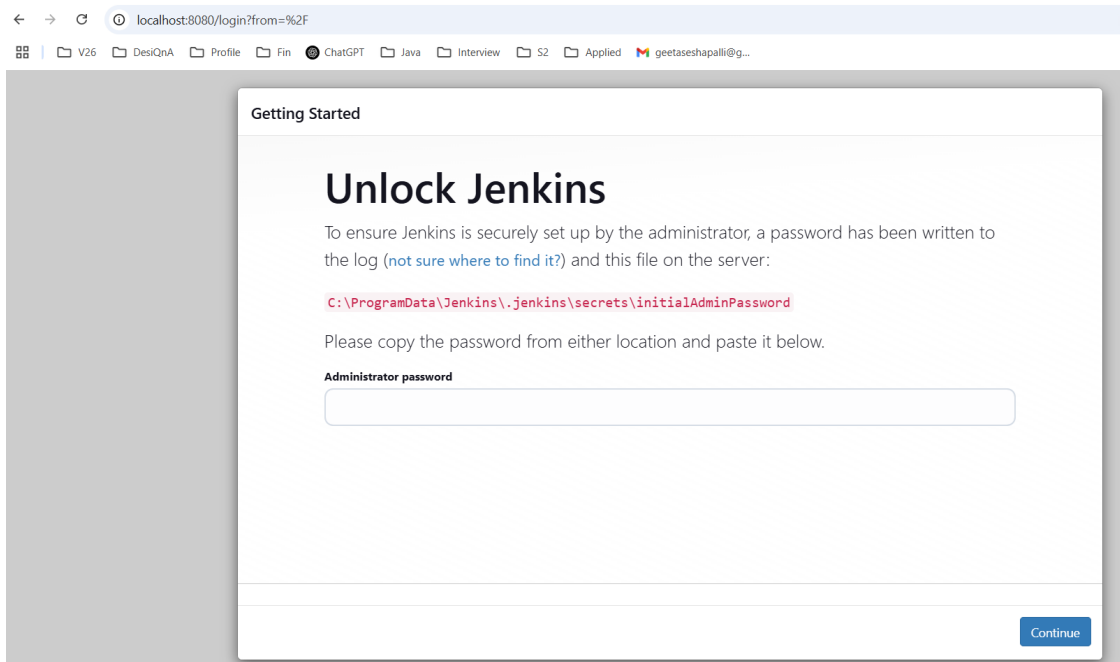
Completed the Jenkins 2.492.1 Setup Wizard

Click the Finish button to exit the Setup Wizard.

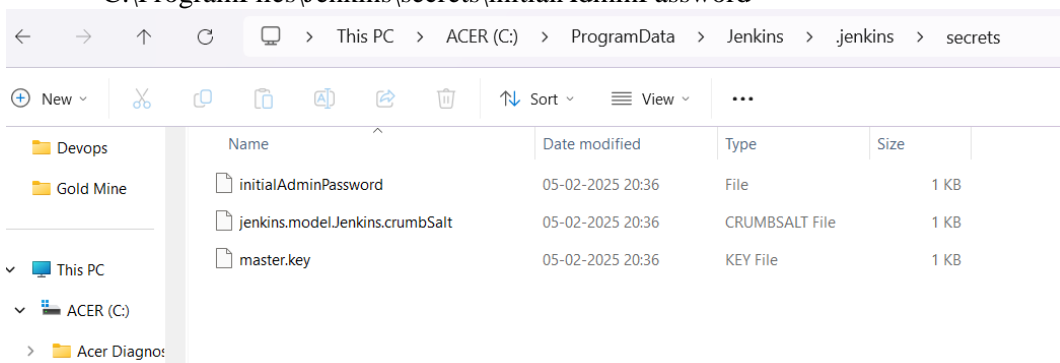


Step 2: Start and Access Jenkins

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



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



Unlock Jenkins

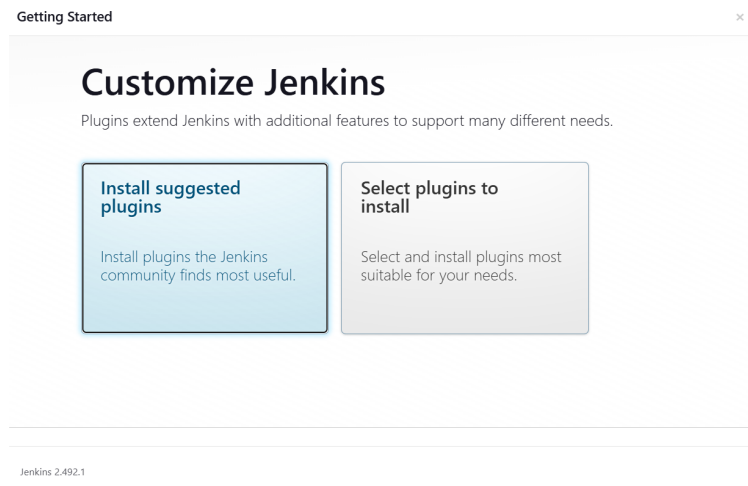
To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

C:\ProgramData\Jenkins\.jenkins\secrets\initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

- Install recommended plugins.



- Create an admin user and complete the setup.

Getting Started

Create First Admin User

Username

Password

Confirm password

Full name

Jenkins 2.492.1 [Skip and continue as admin](#) [Save and Continue](#)

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

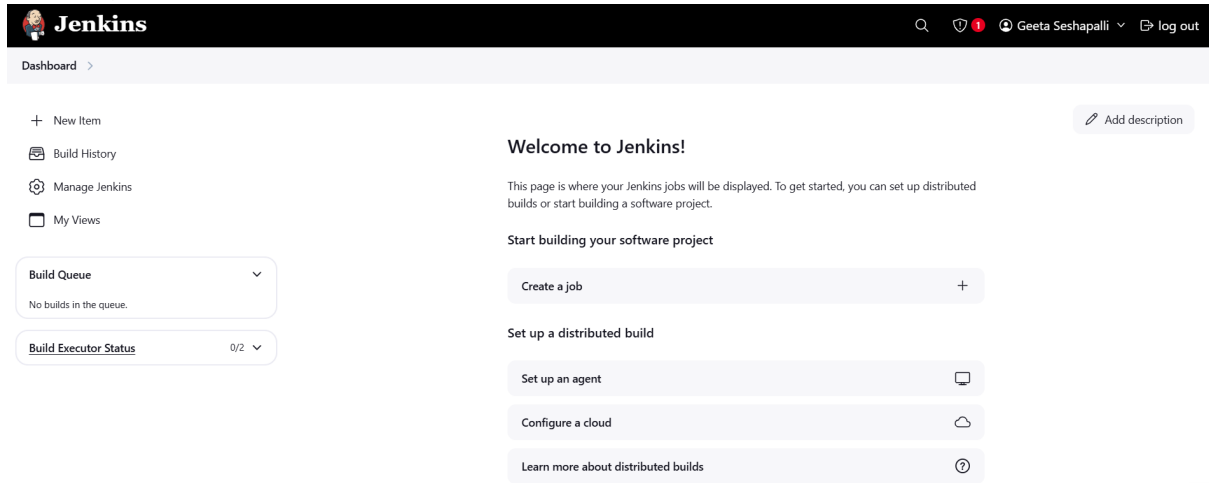
Step 3: Configure Git and GitHub for Jenkins

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> |
```

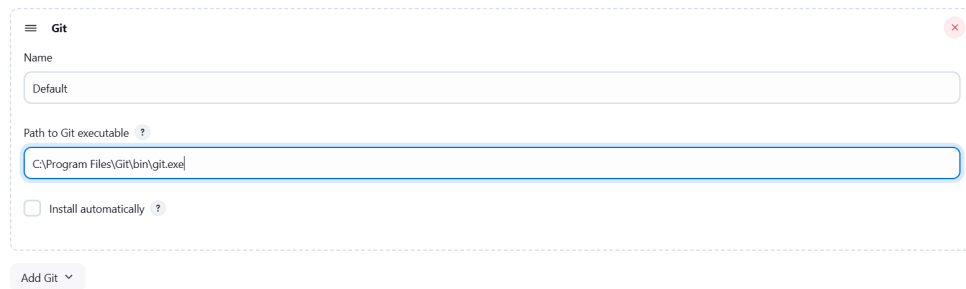
2. Configure Git in Jenkins



The screenshot shows the Jenkins Dashboard. On the left, there's a sidebar with links: New Item, Build History, Manage Jenkins, and My Views. Below these are two status boxes: 'Build Queue' (No builds in the queue) and 'Build Executor Status' (0/2). The main area has a 'Welcome to Jenkins!' message and a 'Start building your software project' section with buttons: 'Create a job', 'Set up an agent', 'Configure a cloud', and 'Learn more about distributed builds'.

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

Git installations

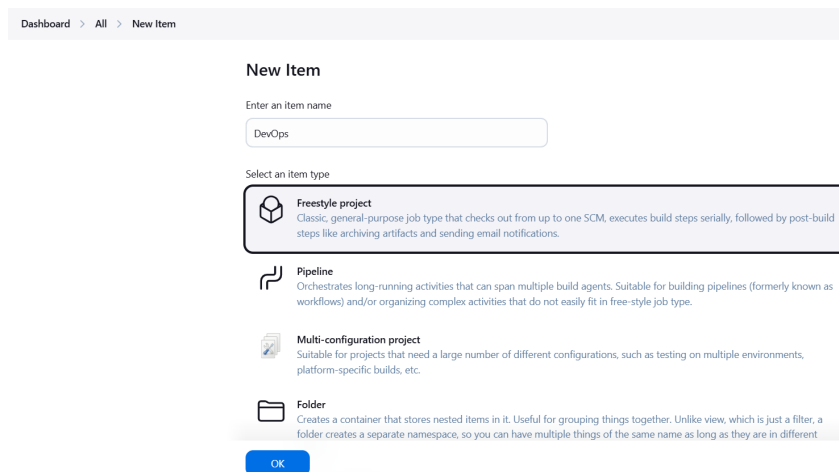


The screenshot shows the 'Git installations' form. It has a 'Name' field with 'Default' entered. The 'Path to Git executable' field is highlighted with a blue border and contains 'C:\Program Files\Git\bin\git.exe'. There is an 'Install automatically' checkbox which is unchecked. At the bottom, there is an 'Add Git' button.

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

1. Create a New Job

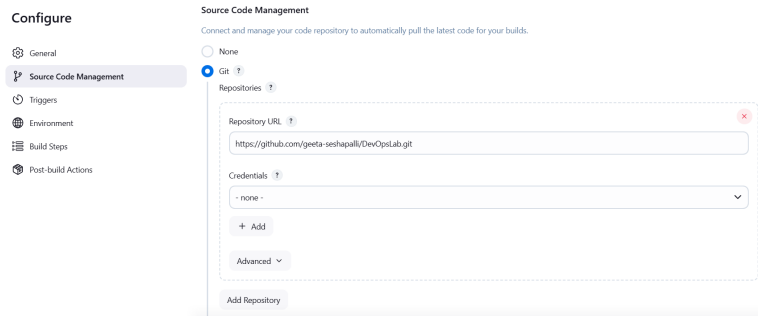
- In Jenkins, go to **Dashboard** → **New Item** → **Freestyle Project**.
- Name the project (e.g., DevOps).



The screenshot shows the 'New Item' form. The 'Enter an item name' field contains 'DevOps'. Under 'Select an item type', the 'Freestyle project' option is selected and highlighted. It has a description: 'Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.' Other options include 'Pipeline', 'Multi-configuration project', and 'Folder'. At the bottom, there is an 'OK' button.

2. Configure GitHub Repository

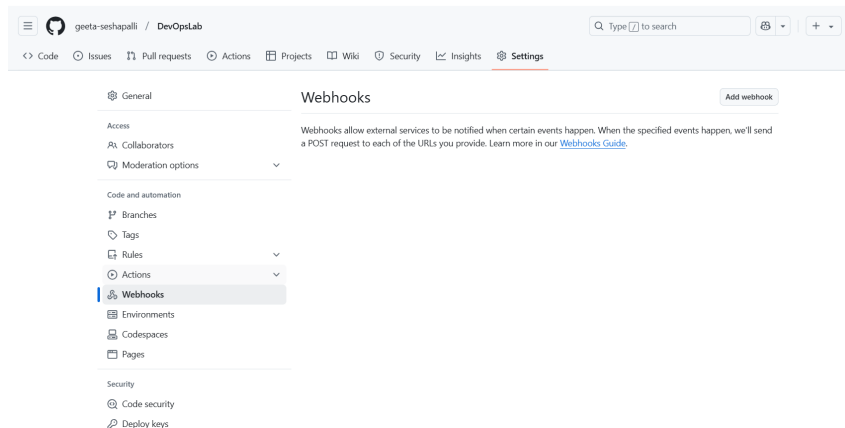
- In the **Source Code Management** section:
 - Select **Git**.
 - Enter your **GitHub repository URL**.



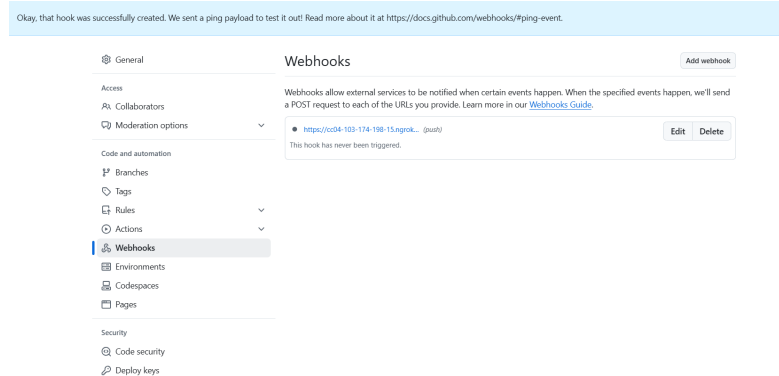
3. Configure Build Triggers

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..."

Build Steps

Automate your build process with ordered tasks like code compilation, testing, and deployment.

Execute Windows batch command ?

Command


See [the list of available environment variables](#)


```
echo "Pulling latest code..."
git pull origin main
echo "Building project..."
```

Advanced ▾


2. Save and Build

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

 Started by user [Geeta Seshapalli](#)

 This run spent:

- 66 ms waiting;
- 2.6 sec build duration;
- 2.7 sec total from scheduled to completion.

 No changes.

Started 7.5 sec ago
Took [2.6 sec](#)

Step 5: Set Up Jenkins Pipeline

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

Add a Jenkinsfile to your GitHub repository:

DevOpsLab / Jenkinsfile in main

Cancel changes Commit changes...

Spaces 2 No wrap

Edit Preview Code 55% faster with GitHub Copilot

```
1 pipeline {
2   agent any
3   stages {
4     stage('Checkout') {
5       steps {
6         git branch: 'main', url: 'git@github.com:geeta-seshapalli/DevOpsLab.git'
7       }
8     }
9     stage('Build') {
10      steps {
11        echo 'Building the project...'
12      }
13    }
14    stage('Test') {
15      steps {
16        echo 'Running tests...'
17      }
18    }
19    stage('Deploy') {
20      steps {
21        echo 'Deploying the project...'
22      }
23    }
24  }
25 }
```

geeta-seshapalli / DevOpsLab

Q Type [7] to search

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main DevOpsLab /

geeta-seshapalli Create Jenkinsfile 3c2a5e8 · now History

Name	Last commit message	Last commit date
242051020_Geeta_EXP1.docx	Add files via upload	last week
Jenkinsfile	Create Jenkinsfile	now
bmi.html	reset button added	last week

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**.

Dashboard > All > New Item

New Item

Enter an item name

MyDevOps Pipeline

Select an item type

Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder

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.

Dashboard > MyDevOps Pipeline > Configuration

Configure

General
Triggers
Pipeline
Advanced

Pipeline

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script from SCM

SCM ?

None

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

Advanced

Advanced

Save Apply

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:**
 - o 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.

SCM ?

Git

Repositories ?

Repository URL ?

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

Credentials ?

- none -

+ Add

Advanced

Add Repository

Branches to build ?

Save Apply

Dashboard > MyDevOps Pipeline >

MyDevOps Pipeline

Status

Changes

Build Now

Configure

Delete Pipeline

Stages

Rename

Pipeline Syntax

Builds

Today

#1 9:47 PM

Build scheduled

https://cd4-103-174-198-15.ngrok-free.app/job/MyDevOps_Pipeline/build?delay=0sec

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

#4 (Feb 5, 2025, 9:59:32 PM)

Started by user Geeta Seshapalli

This run spent:

- 1.6 sec waiting;
- 38 sec build duration;
- 38 sec total from scheduled to completion.

Revision: 92879b1d81b649e34370c3d3edc0a8ddca34297a

Repository: https://github.com/geeta-seshapalli/DevOpsLab.git

- refs/remotes/origin/main

No changes.

Started 1 min 8 sec ago

Took 38 sec

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

DownloadCopyView 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

VJTI FY MTECH CE

GEETA SESHAPALLI

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>