Jenkins

Jenkins

Jenkins is an open-source automation server used in professional software development to build, test, and deploy applications. It enables continuous integration and continuous delivery (CI/CD) by automating the steps in the software development lifecycle, improving code quality and speeding up delivery.

In Short : Jenkins is a CI/CD automation tool that helps teams build, test, and deploy software efficiently.

CI/CD and Testing in the SDLC

In development lifecycle, we follow a structured Software Development Life Cycle (SDLC) process that includes the following phases:

1. Requirement Gathering
2. Planning
3. Designing
4. Development
5. Build Process
6. Testing
7. Code Review
8. Deployment
9. Maintenance

From development to deployment, we implement CI/CD (Continuous Integration and Continuous Deployment) practices using tools like Jenkins. This enables us to automate the building, testing, and deployment of applications efficiently.

Testing Strategy

- **Unit Testing -** Testing individual components or functions of the code in isolation to ensure they work as expected.
- **Integration Testing -** Verifies that different modules or services work together correctly when combined.
- **System Testing -** Tests the complete, integrated system to validate that it meets the specified requirements.

These types of tests are executed within the CI/CD pipeline, typically in development or staging environments.

In addition,

- **Alpha Testing** – Performed internally to validate functionality before external exposure.
- **Beta Testing** – Conducted by a limited group of external users in a controlled setting.

Jenkins CI/CD Workflow (with GitHub Integration)

Here's a typical CI/CD workflow using Jenkins with GitHub:

1. Code Push

   A developer pushes code to a GitHub repository.

2. Trigger Build

   - A webhook is configured in the GitHub repo (under Settings → Webhooks) pointing to:

     `http://<jenkins-server>:8080/github-webhook/`

   - This webhook notifies Jenkins whenever there is a code change (push event).

- Alternatively, Jenkins can be set up to poll the repository at regular intervals to check for changes.

3. Build
    - Jenkins starts a build process using tools like:
        - Maven
        - Gradle
        - Ant
    - These tools compile the code and generate build artifacts. (Artifacts are nothing but the executable code or object (.jar, .war, .exe))
4. Testing
    - Automated tests run as part of the pipeline using tools such as:
        - Selenium (for UI/functional testing)
        - Mocha (for JavaScript unit testing)
    - If the tests pass, the pipeline continues.
    - If the tests fail, Jenkins can:
        - Send failure details to Jira (via plugins or integrations).
        - Notify developers via email or messaging tools.
5. Deployment
    - If the build and tests are successful, Jenkins proceeds to deploy the application to a staging or production environment.
6. Continuous Integration / Continuous Deployment (CI/CD)
    - This entire automated process of building, testing, and deploying is known as CI/CD.
    - It ensures rapid, safe, and consistent delivery of software.

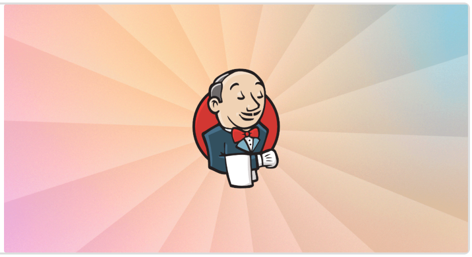
Jenkins Installation

- **Debian/Ubuntu -**

```
sudo apt update
sudo apt install fontconfig openjdk-21-jre openjdk-21-jdk
sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

More Information -

Installing Jenkins
Jenkins – an open source automation server which enables developers around the world to reliably build, test, and deploy their software
https://www.jenkins.io/doc/book/installing/



Jenkins Workspace

The Jenkins workspace is the directory on the Jenkins server where the files related to a specific build job are checked out, built, and tested.

When Jenkins runs a job (a build), it needs a place to work, a folder where it:

- Downloads the code (from GitHub, GitLab, etc.)
- Compiles the code
- Runs tests
- Creates build files (like `.jar`, `.war`, `.zip`)
- Stores temporary files

This working folder is called the workspace.

What Happens Inside the Workspace?

Suppose we have a Jenkins job called `build-java-app`:

1. Developer pushes code to GitHub.
2. Jenkins job is triggered.
3. Inside the workspace:
    - Jenkins pulls the latest code from GitHub.
    - Compiles `.java` files into `.class` files.
    - Runs test cases.
    - Generates `.jar` or `.war` files.
    - Saves the build artifacts.

Getting Started With Jenkins

Deploying a Simple Java Application Using Jenkins

1. Create a Git Repository

- Create a new Git repository and commit a basic Java file named `myfile.java`.

```
public class myfile {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

2. Access Jenkins

- Open your Jenkins server:

`http://<jenkins-server-ip>:8080`

3. Create a New Jenkins Job

- Click New Item
    - Item Name: `java-project-build`
    - Project Type: Freestyle Project

4. General Configuration

- Optionally, add a Description for the project under the General section.

5. Source Code Management

- Under Source Code Management, select Git
    - Paste your repository URL containing `myfile.java`
    - https://github.com/harshkhalkar/node.git

6. Add Build Step

- Scroll to Build section
    - Add a Build Step → `Execute Shell`
    - Enter the following commands:

```
javac myfile.java
java myfile
```

7. Save and Run the Job

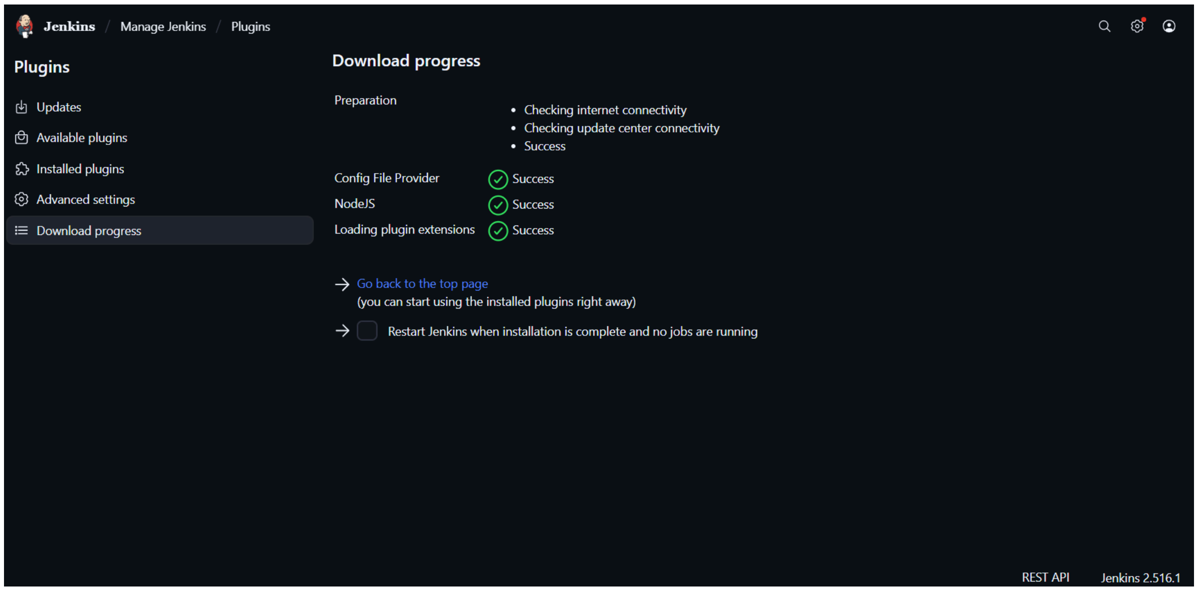
- Click Save
- Click Build Now to trigger the job manually
- Check the Console Output to verify successful compilation and execution

---

Deploying a Node.js Application Using Jenkins

1. Install Node.js Plugin in Jenkins

- To run a Node.js app, first install the NodeJS plugin from Jenkins Plugin Manager.


Jenkins / Manage Jenkins / Plugins

Plugins
- Updates
- Available plugins
- Installed plugins
- Advanced settings
- Download progress

Download progress

Preparation	• Checking internet connectivity • Checking update center connectivity • Success
Config File Provider	✓ Success
NodeJS	✓ Success
Loading plugin extensions	✓ Success

→ Go back to the top page
(you can start using the installed plugins right away)

→ ☐ Restart Jenkins when installation is complete and no jobs are running

REST API Jenkins 2.516.1

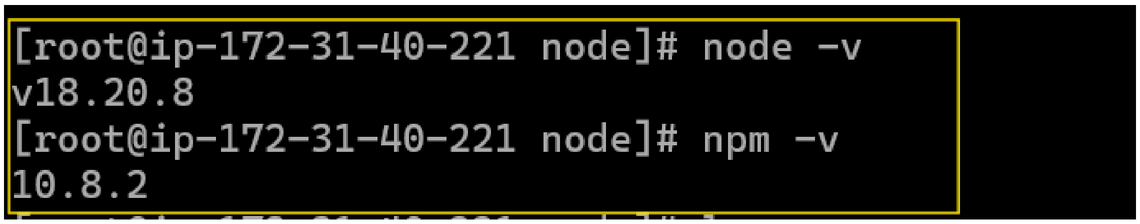
2. Configure Node.js in Jenkins

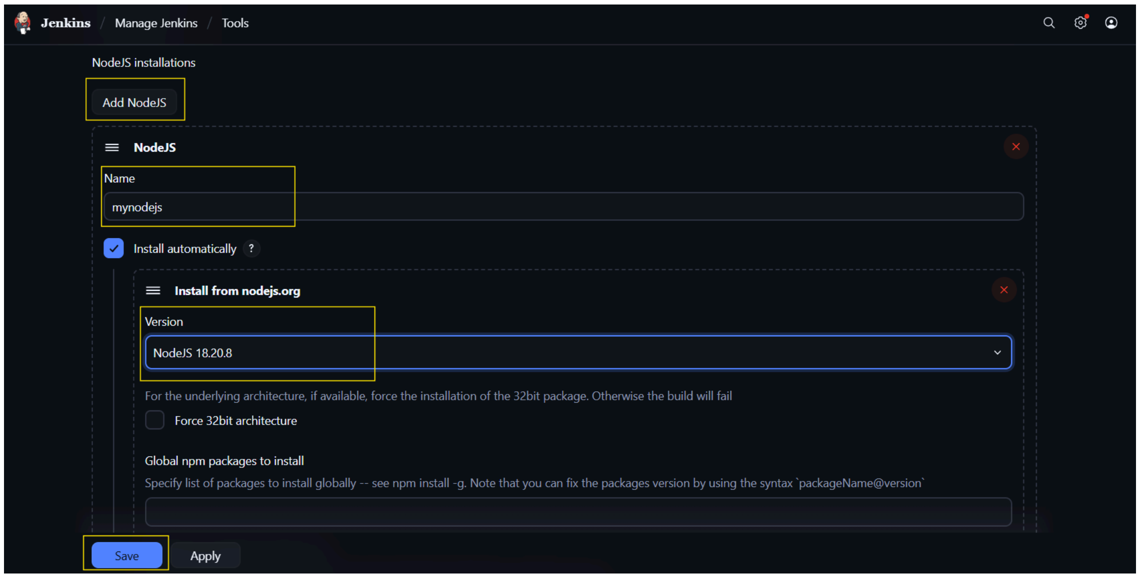
- Navigate to:

  `Manage Jenkins → Tools → NodeJS Installation`

- Click Add NodeJS
  - Name the installation (e.g., `NodeJS 18`)
  - Select the NodeJS version from the dropdown

    (You can verify version on your terminal using `node -v` *)*


```
[root@ip-172-31-40-221 node]# node -v
v18.20.8
[root@ip-172-31-40-221 node]# npm -v
10.8.2
```


Jenkins / Manage Jenkins / Tools

NodeJS installations

Add NodeJS

NodeJS

Name: mynodejs

☑ Install automatically

Install from nodejs.org

Version: NodeJS 18.20.8

For the underlying architecture, if available, force the installation of the 32bit package. Otherwise the build will fail

☐ Force 32bit architecture

Global npm packages to install

Specify list of packages to install globally -- see npm install -g. Note that you can fix the packages version by using the syntax `packageName@version`

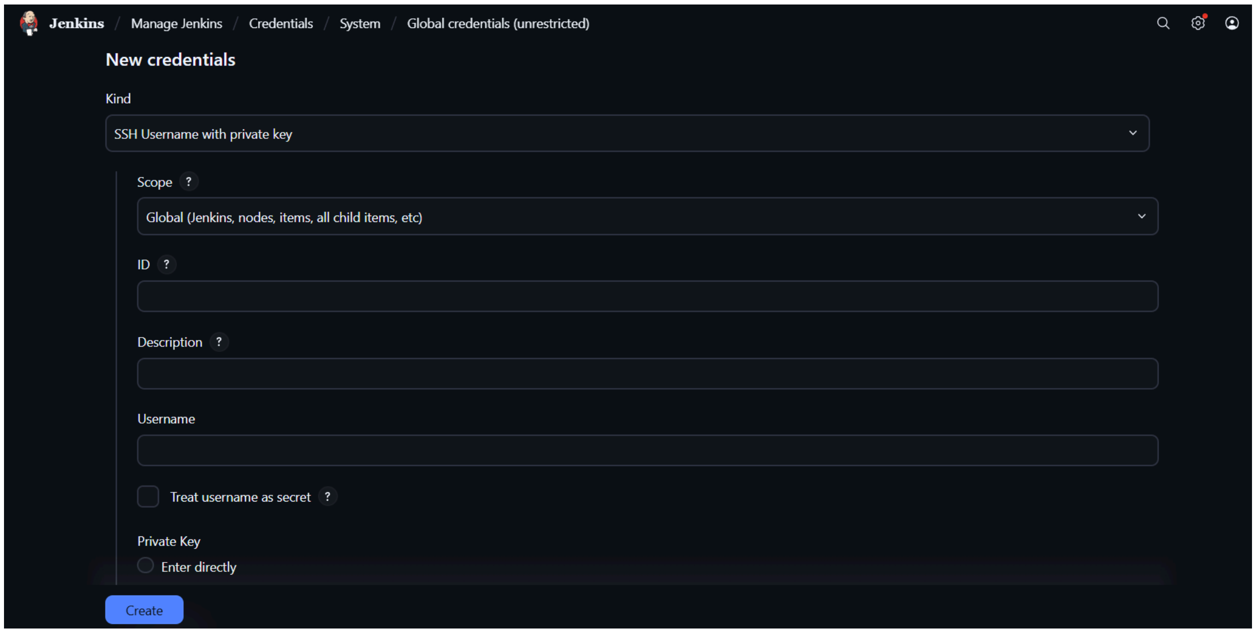
Save Apply

3. Configure SSH Credentials and SSH Site

- Go to:

  `Manage Jenkins → Credentials → System → Add Credentials`

  - Kind: SSH Username with private key
  - Provide your username and private key


Jenkins / Manage Jenkins / Credentials / System / Global credentials (unrestricted)

New credentials

Kind: SSH Username with private key

Scope: Global (Jenkins, nodes, items, all child items, etc)

ID

Description

Username

☐ Treat username as secret

Private Key

○ Enter directly

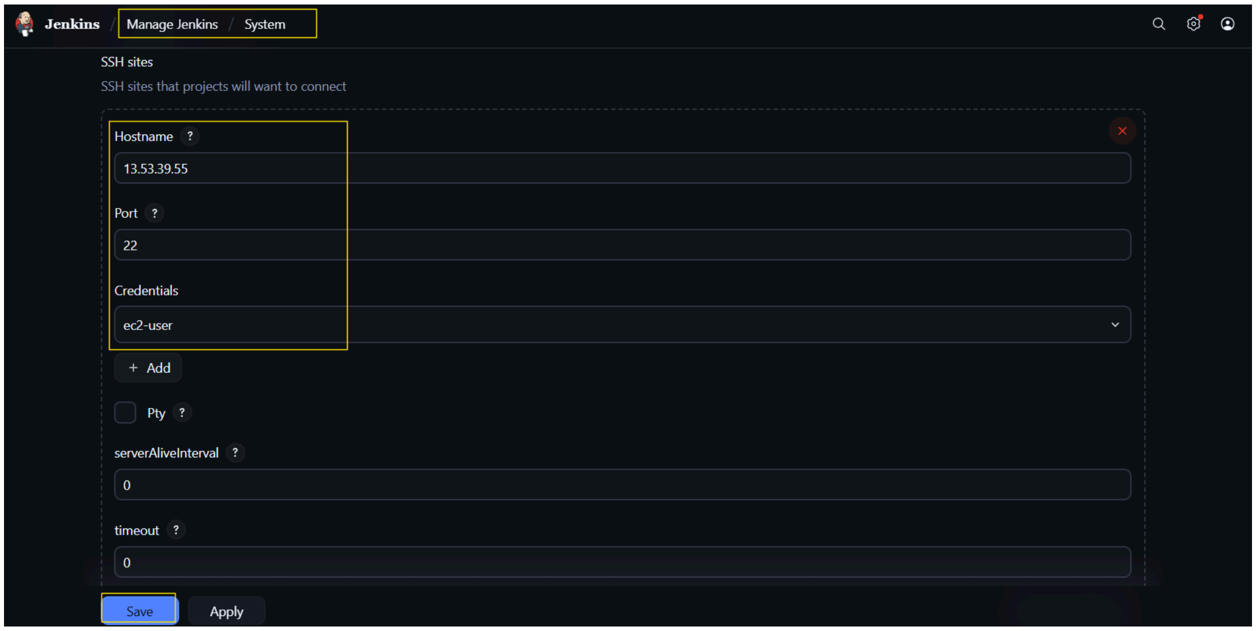
Create

- Now, go to:

  `Manage Jenkins → System`

Scroll down to SSH Sites

- Add Hostname, Port, and select the credential you just added.



SSH sites

SSH sites that projects will want to connect

Hostname: 13.53.39.55

Port: 22

Credentials: ec2-user

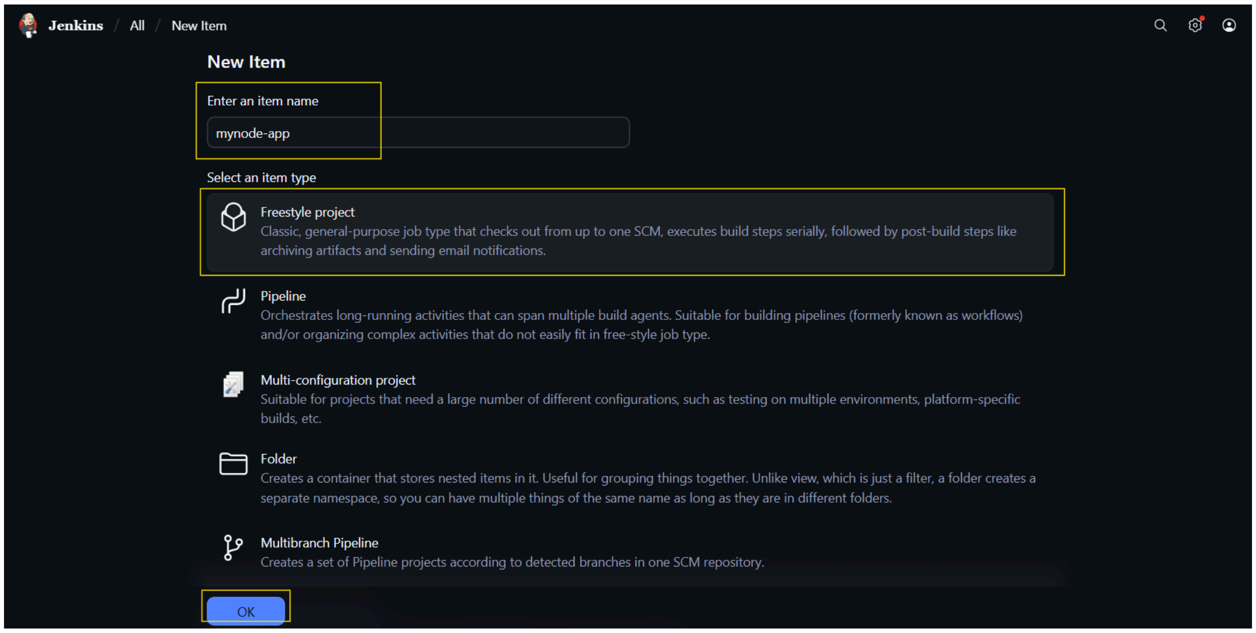
Pty

serverAliveInterval: 0

timeout: 0

4. Create a New Jenkins Job

- Click New Item
    - Item Name: `mynode-app`
    - Project Type: Freestyle Project


Jenkins / All / New Item

New Item

Enter an item name

mynode-app

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.

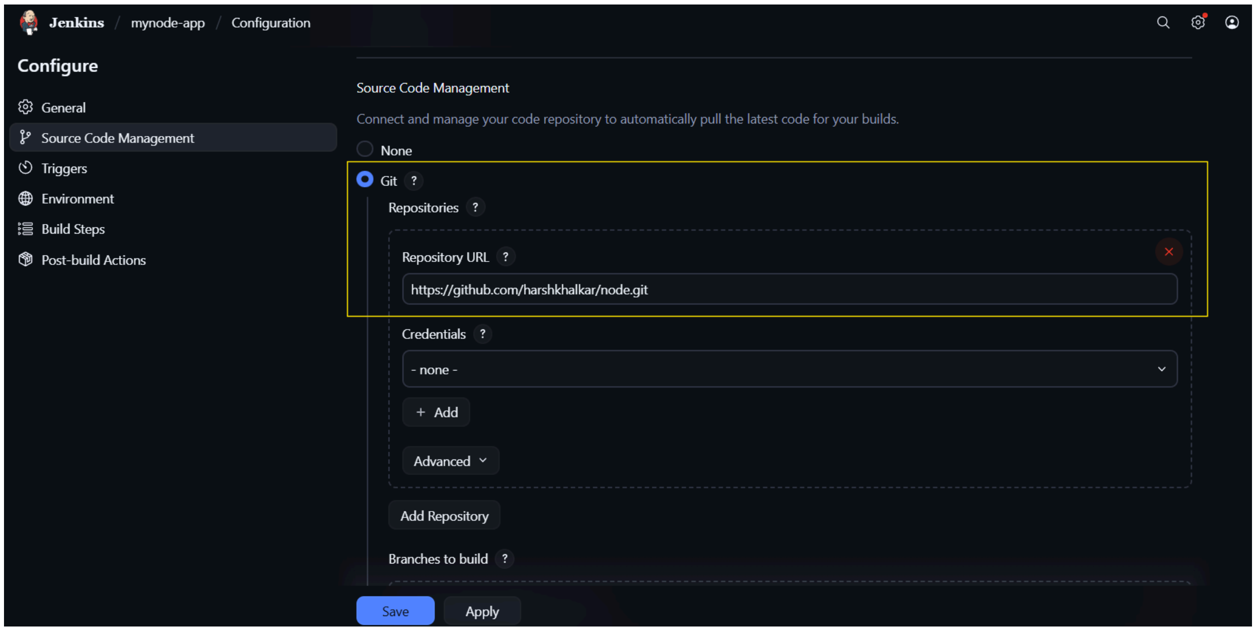
Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

OK

5. Configure Source Code Management (SCM)

- In the job configuration:
  - Add a short Description
  - Scroll down to Source Code Management
    - Select Git
    - Paste your repository URL


Jenkins / mynode-app / Configuration

Configure

- General
- Source Code Management
- Triggers
- Environment
- Build Steps
- Post-build Actions

Source Code Management

Connect and manage your code repository to automatically pull the latest code for your builds.

None [✓ Git]

Repositories

Repository URL: https://github.com/harshkhalkar/node.git

Credentials: - none -

Add

Advanced

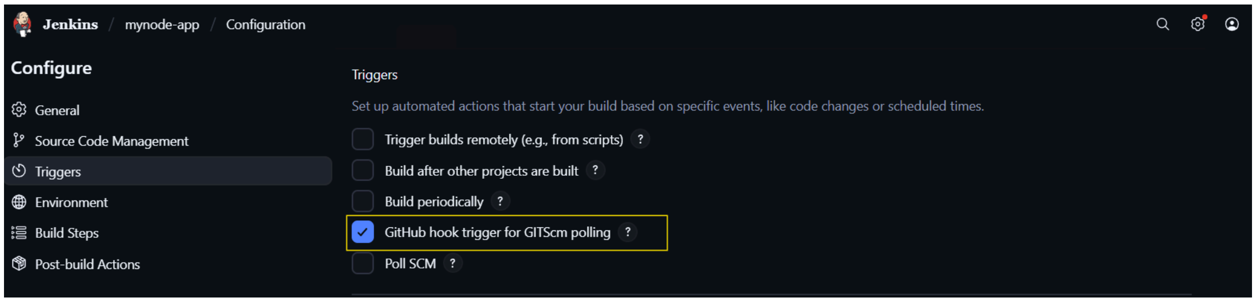
Add Repository

Branches to build

Save Apply

6. Configure Build Triggers

- Under Build Triggers, select: `GitHub hook trigger for GITScm polling`



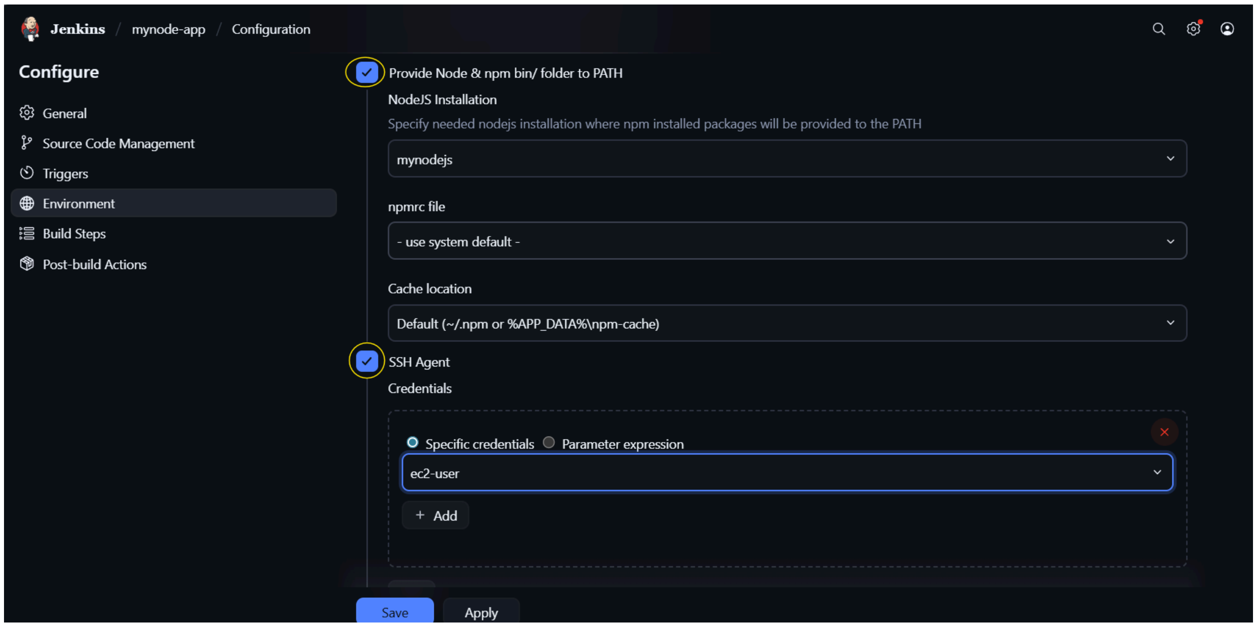
Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [✓] GitHub hook trigger for GITScm polling
- [ ] Poll SCM

7. Configure Build Environment

- Under Build Environment:
  - Check Provide Node & npm bin/folder to PATH
    - Select the NodeJS version you configured earlier
  - Enable SSH Agent
    - Select the SSH credential you added



[✓] Provide Node & npm bin/ folder to PATH

NodeJS Installation

Specify needed nodejs installation where npm installed packages will be provided to the PATH

mynodejs

npmrc file

- use system default -

Cache location

Default (~/.npm or %APP_DATA%\npm-cache)

[✓] SSH Agent

Credentials

[✓ Specific credentials] Parameter expression

ec2-user

8. Add Build Step

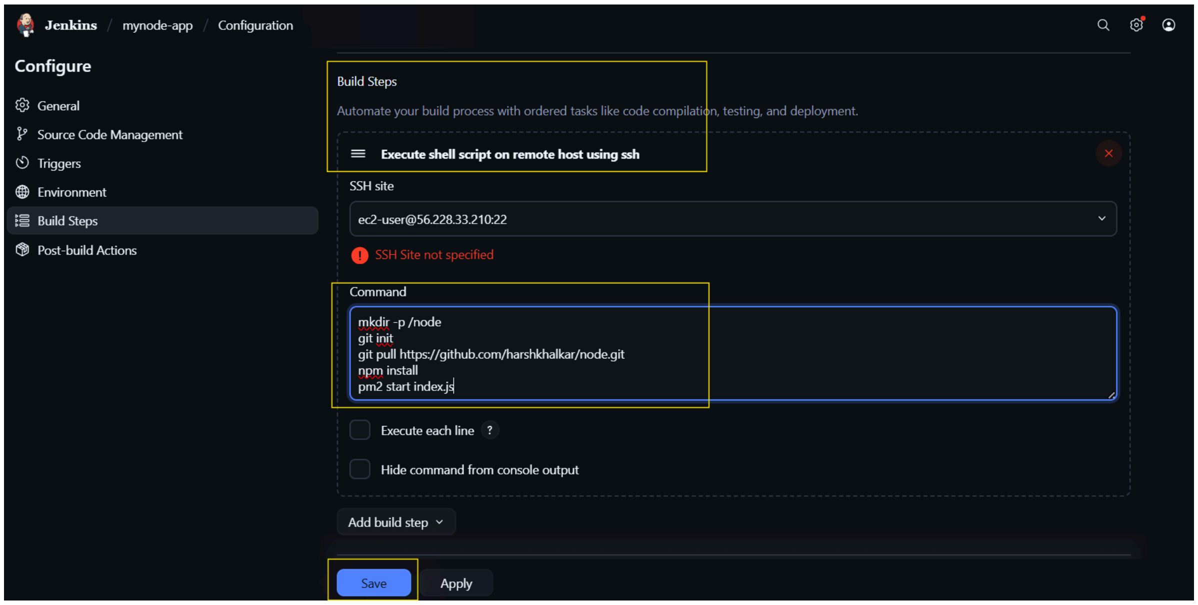
- Add Build Step:

  `Execute shell script on remote host using ssh`

  - Choose SSH Site
  - Add the following command:

Non-Containerized App:

```
mkdir -p /node
git init
git pull <github-repo-link>
sudo npm install
pm2 start index.js || pm2 restart index.js
```


Jenkins / mynode-app / Configuration
Configure
General
Source Code Management
Triggers
Environment
Build Steps
Post-build Actions
Build Steps
Automate your build process with ordered tasks like code compilation, testing, and deployment.
Execute shell script on remote host using ssh
SSH site
SSH Site not specified
Command
Execute each line
Hide command from console output
Add build step
Save
Apply

SSH site: ec2-user@56.228.33.210:22

```
mkdir -p /node
git init
git pull https://github.com/harshkhalkar/node.git
npm install
pm2 start index.js
```

Containerized App:

```
mkdir -p /node
git init
git pull <github-repo-link>
sudo docker build -t myapp .
sudo docker run -d -p 3000:3000 --name nodeapp1 myapp
```

Note: Give Jenkins permission to run Docker commands:

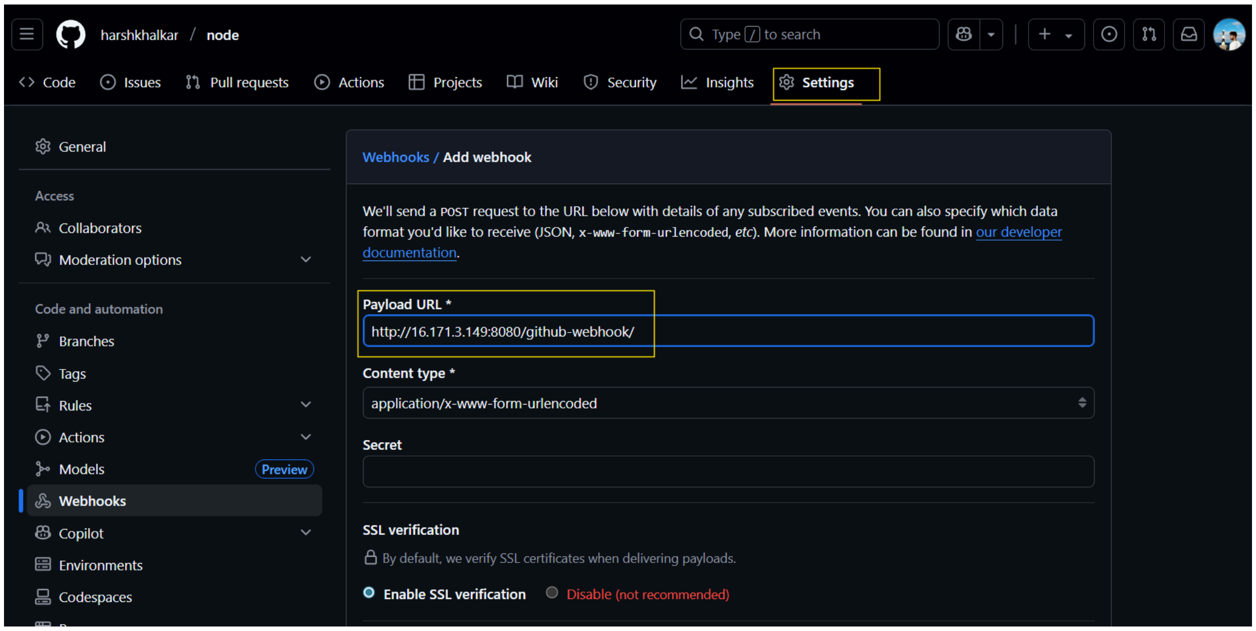
```
sudo usermod -aG docker jenkins

# OR

sudo visudo
# Add the following line:
jenkins ALL=(ALL) NOPASSWD:ALL
```

9. Setup GitHub Webhook

- In your GitHub repository:
  - Go to `Settings → Webhooks → Add Webhook`
  - Payload URL: `http://<jenkins-ip>:8080/github-webhook/`
  - Save the webhook



Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, *etc*). More information can be found in our developer documentation.

Payload URL *: http://16.171.3.149:8080/github-webhook/

Content type *: application/x-www-form-urlencoded

Secret:

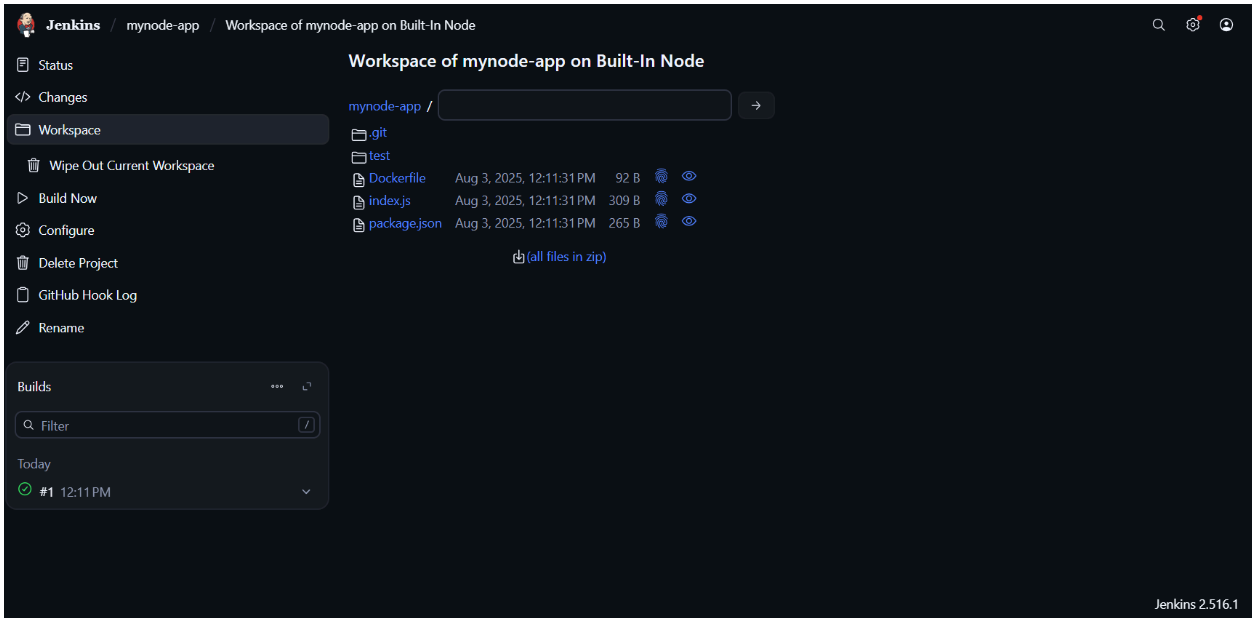
SSL verification

By default, we verify SSL certificates when delivering payloads.

[✓ Enable SSL verification] Disable (not recommended)

10. Push Code and Trigger Build

- Push your code to GitHub. Jenkins will automatically trigger a build via the webhook.


Jenkins / mynode-app / Workspace of mynode-app on Built-In Node

Workspace of mynode-app on Built-In Node

mynode-app /

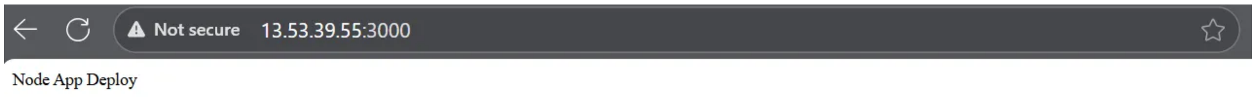
```
.git
test
Dockerfile      Aug 3, 2025, 12:11:31 PM   92 B
index.js        Aug 3, 2025, 12:11:31 PM   309 B
package.json    Aug 3, 2025, 12:11:31 PM   265 B
```

(all files in zip)

11. Access Your App

- Use your server IP and port to access the app:

  `http://<server-ip>:3000/`



Node App Deploy

On LiveServer -

```
yum install nodejs -y
node --version
npm --version
npm install -g pm2
```