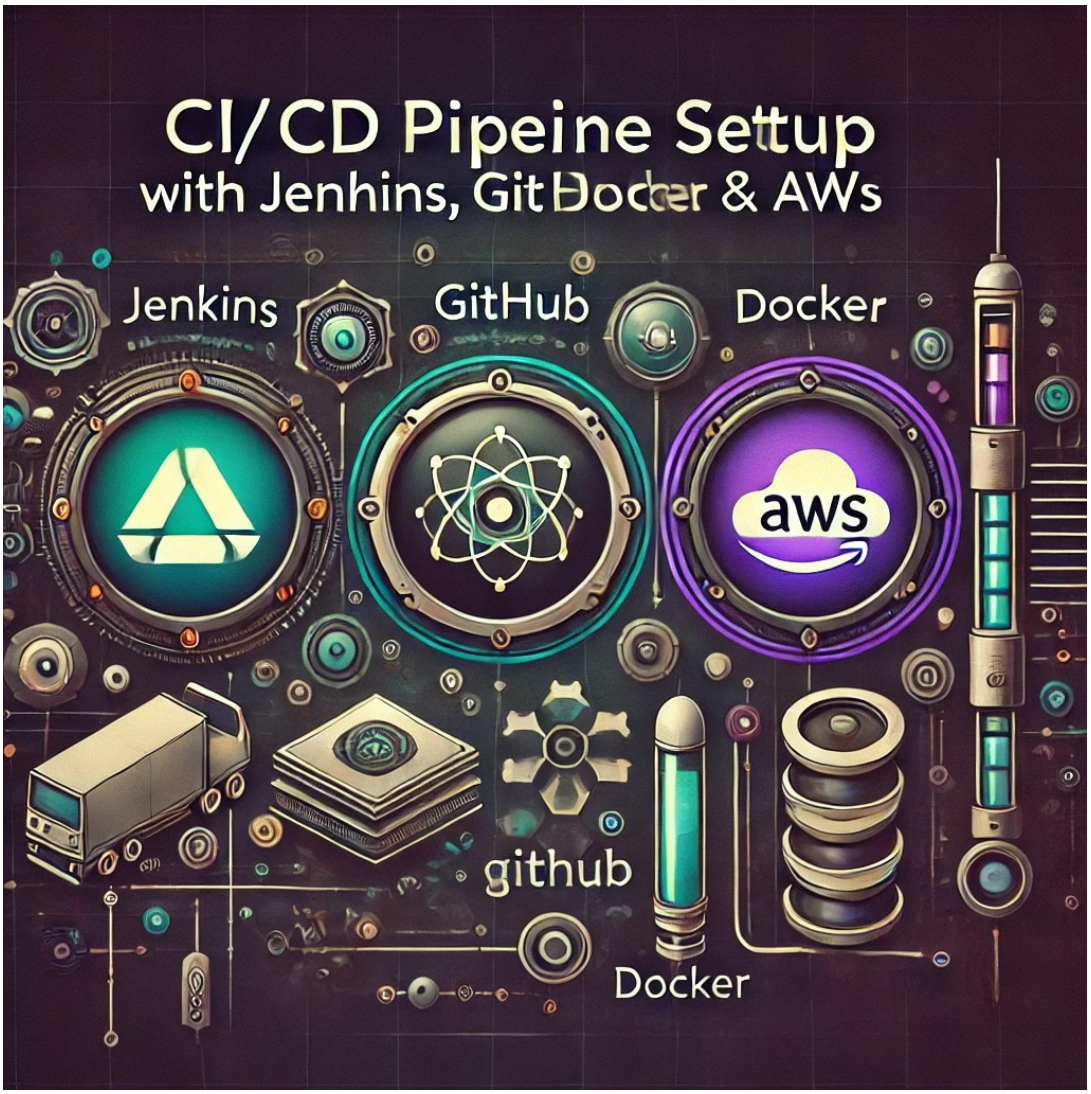
nvndops

Continuous Integration and Continuous Deployment (CI/CD) Pipeline Project



Jenkins CI/CD Pipeline with Docker and GitHub

docker

Jenkins

nvndops

Table of Contents

docker

Jenkins

1. Introduction

Overview of the Project

This project focuses on creating a Continuous Integration and Continuous Deployment (CI/CD) pipeline for a web application using Jenkins, GitHub, and Docker. The objective is to automate the entire process from code changes in the repository to deploying the application on a server, ensuring a seamless and efficient workflow. By integrating Jenkins with GitHub, the project is set up to automatically trigger builds and deployments whenever there is a change in the source code. Docker and Docker Compose are used to manage the application's containerization and ensure that the deployment process is consistent across different environments. This project not only demonstrates the power of CI/CD in modern software development but also serves as a practical example of how to integrate various tools to streamline development and operations.

Objective and Goals

- Set up a connection between Jenkins and GitHub: Establish a seamless integration between your Jenkins job and GitHub repository to automate the build and deployment process.

- Add GitHub WebHooks to Jenkins: Configure WebHooks in GitHub to trigger Jenkins builds automatically when there are code changes, ensuring continuous integration.

- Run the application using Docker Compose: In the "Execute Shell" section of the Jenkins job, execute the application using Docker Compose to ensure that all components are correctly deployed.

- Create a Docker Compose file for the project: Develop a Docker Compose file that defines the application's environment, making it easier to deploy and manage the application as an open-source contribution.

- Run the project: Successfully run the project using the CI/CD pipeline, achieving an automated and efficient deployment process.

2. Project Setup

Prerequisites

Before you start, ensure you have the following:

- **Laptop/Desktop:** A system to run Jenkins, Docker, and other necessary tools.

- **GitHub Account:** A GitHub account is essential for storing your source code and integrating with Jenkins.

- **AWS Account:** An AWS account is required for deploying the application and managing the infrastructure.

Tools and Technologies Used

This project utilizes several tools and technologies to create an efficient CI/CD pipeline:

- **Jenkins:** An open-source automation server that facilitates continuous integration and continuous delivery (CI/CD).

- **GitHub:** A version control platform used to store and manage your source code.

- **Docker & Docker Compose:** Tools for containerizing the application and managing multi-container Docker applications.

- **WebHooks:** A mechanism to trigger Jenkins jobs automatically upon changes in the GitHub repository.

- **AWS:** Amazon Web Services (AWS) is used for hosting and managing the application, ensuring scalability and reliability.

3. Step-by-Step Guide

Step 1: Create and Configure an EC2 Instance

1. Create an EC2 Instance:

- Start by creating a new EC2 instance on AWS. This instance will serve as the main server where Jenkins will be installed.

2. Install Java and Jenkins:

- SSH into your EC2 instance.
- Install Java:

```
sudo apt update
sudo apt install openjdk-11-jdk -y
```

- Install Jenkins by following the official Jenkins documentation.

3. Expose Port 8080:

- Ensure that port 8080 is open in your security group settings to access Jenkins.
- You can access Jenkins via `http://<your-ec2-public-ip>:8080`.

4. Set Up Jenkins:

- During Jenkins setup, you'll be prompted to enter an initial password. Copy the password from the following path:

```

***sudo cat /var/lib/jenkins/secrets/initialAdminPassword***

```

- Proceed with the setup:
  - Install the suggested plugins.
  - Configure the admin profile.
  - Set the Jenkins URL and finish the setup.

Now, your first EC2 instance with Jenkins is fully set up.

Step 2: Set Up a Jenkins Agent Node

1. Create a New EC2 Instance:

- Create another EC2 instance on AWS to act as the Jenkins agent node.

2. Generate SSH Key:

- On your first EC2 instance (where Jenkins is installed), generate an SSH key pair:

```

***ssh-keygen***

```

```

- Navigate to the `.ssh` folder and copy the public key:

```

cat ~/.ssh/id_rsa.pub

```

- SSH into the agent EC2 instance and paste the public key into the `authorized_keys` file.

**3. Install Java, Docker, and Docker Compose:**

- On the agent EC2 instance, install the necessary software:

- Java:

```

sudo apt update

sudo apt install openjdk-11-jdk -y

```

- Docker:

```

sudo apt install docker.io -y

```

- Docker Compose:

```

sudo apt install docker-compose -y

```

The second EC2 instance is now ready to serve as a Jenkins agent.

## Step 3: Configure and Add the Agent Node in Jenkins

### 1. Go to Manage Jenkins:

- In your Jenkins dashboard, navigate to “Manage Jenkins” and set up a new agent node.

### 2. Set Up the Agent:

- Name the Node: Enter a name for your agent node and select “Permanent Agent.”

- Add Details:

- Name and Description: Fill in the details as needed.

- Remote Root Directory: On the agent EC2 instance, create a directory named `apps` and paste its path here.

- Labels: Add a label such as `dev`.

- Launch Method: Select “Launch agent via SSH.”

- Host: Enter the IPv4 address of the agent node.

### 3. Add SSH Credentials:

- In the credentials section, select “Add” and choose “SSH Username with Private Key.”

- Fill out the required fields:
```

- ID and Description: Provide identifiers for the credentials.

- Username: Enter the username for your host EC2 instance (where Jenkins is installed).

- Private Key: Paste the private key from the host EC2 instance.

- Select the added credentials and choose “Non-verifying Verification Strategy.”

4. Finish Setup:

- After saving, your agent node should be added.

- Refresh the Jenkins page. If the agent node shows free space and other details, it has been configured correctly. Otherwise, troubleshoot and reconfigure as needed.

Step 4: Jenkins Job Setup and GitHub Webhook Configuration

1. Fork the Repository:

-Go to therepository: `https://github.com/LondheShubham153/node-todo-cicd`

- Fork the repository to your GitHub account.

2. Create a Jenkins Job:

- Go to Jenkins and create a new job.

- Enter the project name as `node-todo-project`.

- Select `Freestyle project`.

3. Configure GitHub Repository:

- Scroll down to the `Source Code Management` section and select `Git`.

- Add the forked repository URL.

- In the `Branches to build` section, select `main`.

4. Configure Build Triggers:

- Scroll to `Build Triggers` and select `GitHub hook trigger for GITScm polling`.

5. Add Shell Command:

- Scroll down to `Build` and select `Add build step`.

- Select `Execute Shell` and paste the following command:

```
docker buid . -t node-app:latest

docker compose down && docker compose up -d
```

Step 5: Running the Project

1. Testing the WebHook:

- Make changes to the source code in your GitHub repository. Commit and push the changes.

- Jenkins will automatically trigger a build.

2. Verify the Build:

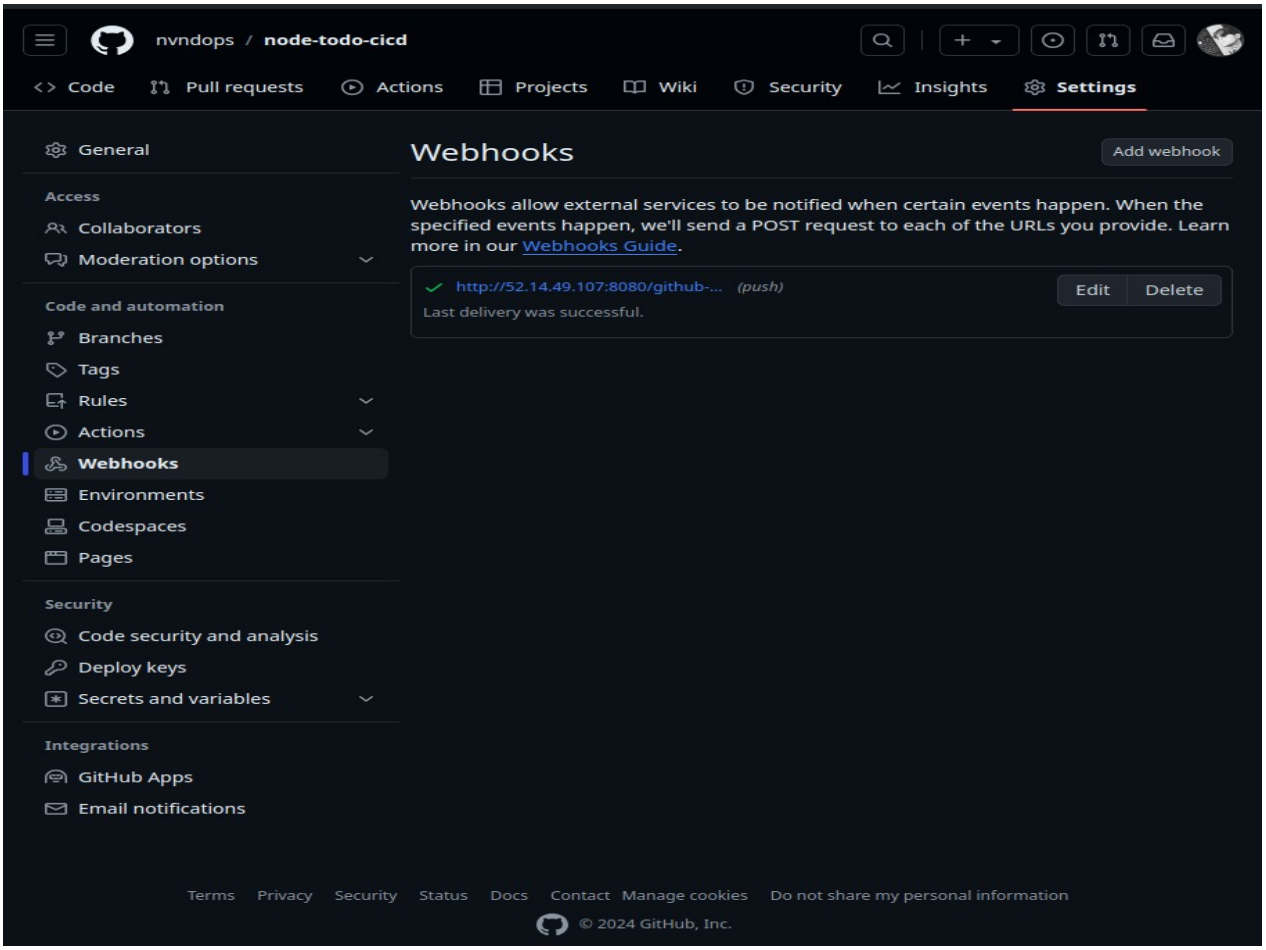
- Go to your Jenkins job and check the build history.

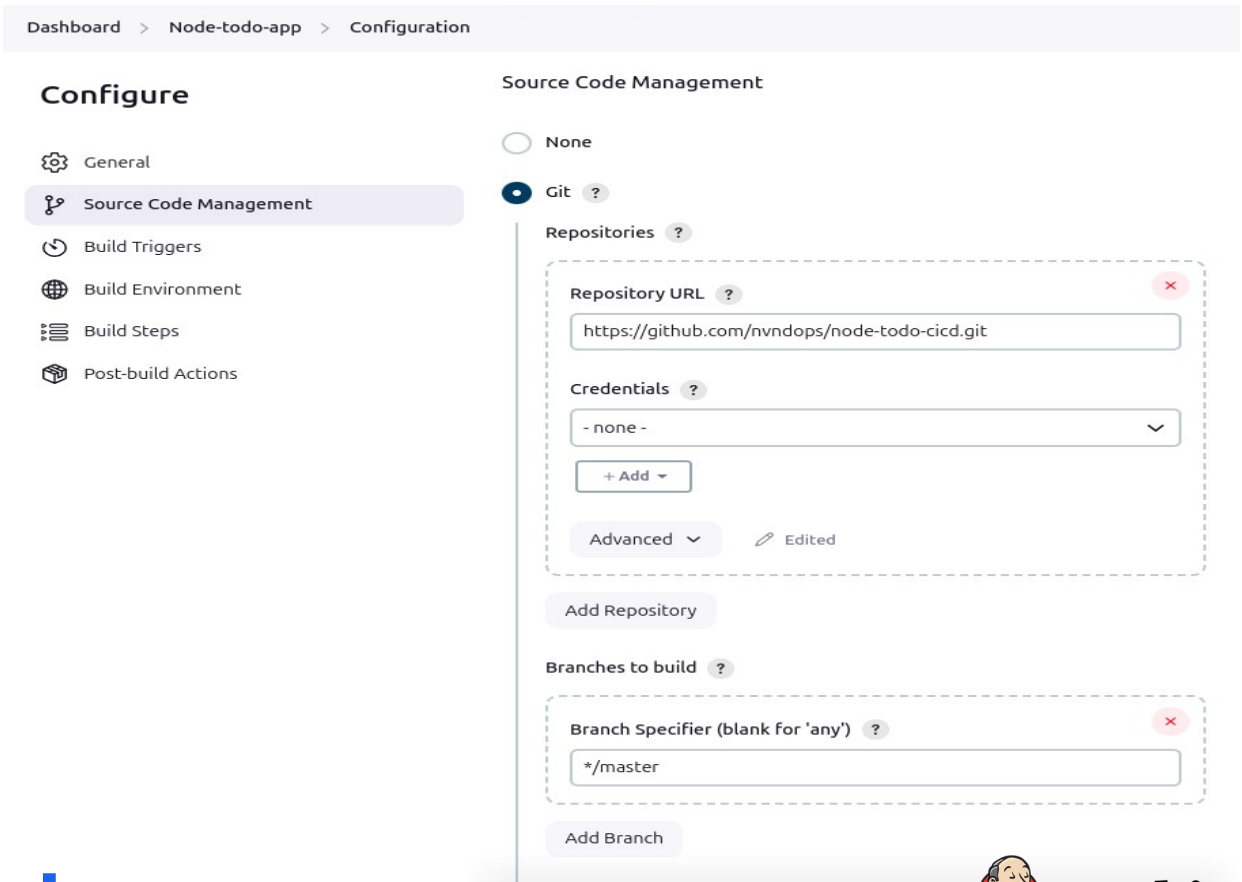
- If the build is successful, your application should be running on the server.

4. Summary

In this project, we successfully set up a CI/CD pipeline using Jenkins, GitHub, and Docker. The integration of these tools allowed us to automate the process of building, testing, and deploying our application. By configuring a Jenkins agent node and setting up a WebHook with GitHub, we ensured that our application was automatically deployed whenever changes were made to the source code. The use of Docker and Docker Compose further streamlined the deployment process, making it easy to manage and scale our application.

nvndops





docker

Jenkins

nvndops

Dashboard > Node-todo-app > Configuration

Configure

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

With Ant ?

Build Steps

Execute shell ?

Command

See the list of available environment variables

```
docker build . -t node-app:latest
docker compose down && docker compose up -d
```

Advanced

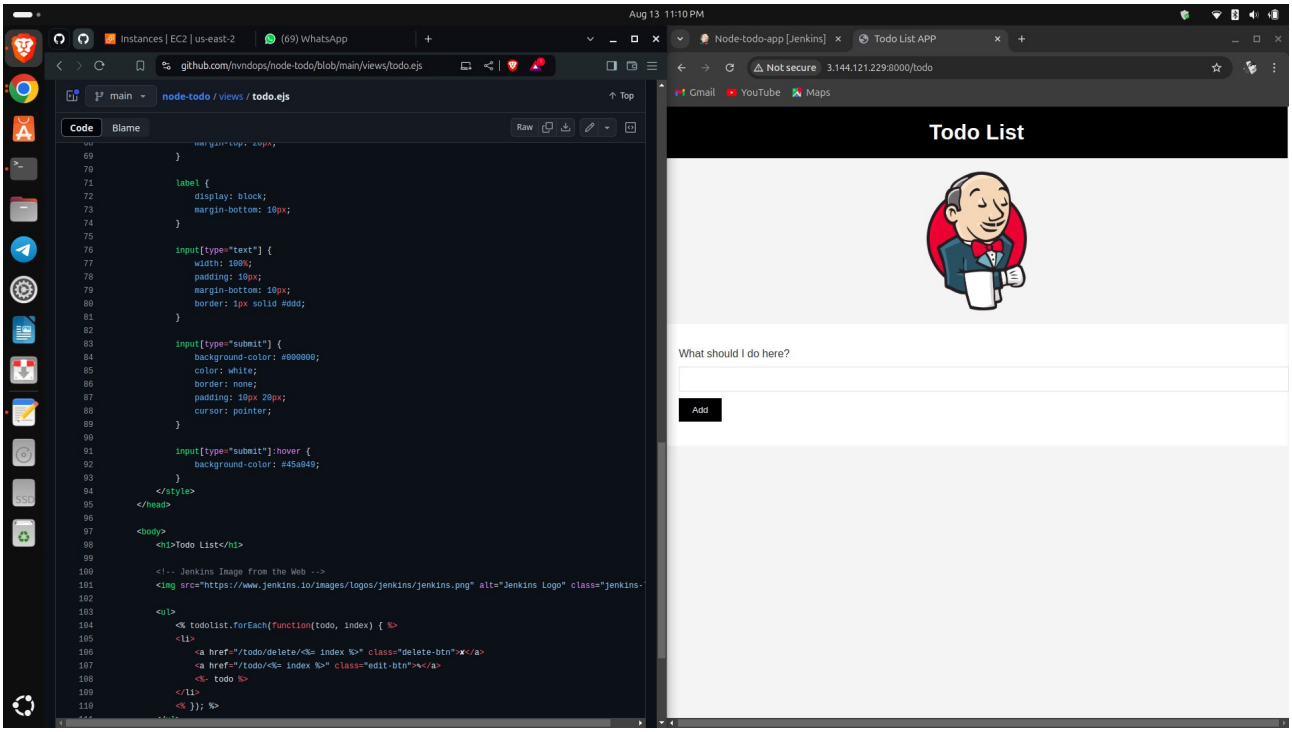
Add build step

Post-build Actions

Add post-build action

Save Apply

REST API Jenkins 2.462.1



docker

Jenkins