Day 23 Task: Jenkins Pipeline Project for DevOps Engineers

Table of contents

Step 1: Setup Your Jenkins Environment
Step 2: Prepare Your Project
Step 3: Create a Jenkinsfile
Step 4: Configure Jenkins Pipeline
Run the Pipeline
Explanation
Notes
Conclusion
Show less

The community is absolutely crushing it in the #90daysofdevops journey. Today's challenge is particularly exciting as it entails creating a Jenkins Pipeline Project, an opportunity for DevOps engineers to showcase their skills and push their limits. Who's ready to dive in and make it happen? 😎

What is CI/CD?

Continuous Integration (CI) is the practice of automating the integration of code changes from multiple developers into a single codebase. It involves frequent commits to a central repository, such as GitHub or Bitbucket, followed by automated builds and tests to ensure the new code integrates smoothly. The primary goals of CI are to identify and fix bugs quickly, streamline the integration process for teams, improve software quality, and accelerate the release of new features.

Continuous Delivery (CD) builds on CI by automating the release process, ensuring that code can be deployed to production at any time. CD involves running integration and regression tests in a staging environment that mirrors production to catch issues before deployment. This approach ensures that the product is always in a release-ready state, allowing for frequent and reliable updates.

What is a Pipeline Project?

A Pipeline project in Jenkins allows for building, testing, and deploying software using a series of steps defined in a script. Pipeline projects offer more flexibility and control compared to freestyle projects. They enable complex automation scenarios, including parallel execution, conditional steps, and integration with various tools.

Task Breakdown

Task-01: Creating a Jenkins Pipeline Project

1. **Create an Agent for Your App**: First, ensure you have an agent configured to run your Jenkins jobs. This agent can be a dedicated machine or a Docker container configured to communicate with Jenkins.
2. **Create a New Jenkins Pipeline Project**:

- Open Jenkins and create a new Pipeline project.
- In the project configuration, define your pipeline script.

3. **Build Step to Run** `docker build`:

```
stage('Build Image') {
     steps {
         script {
             dockerImage = docker.build("your-app-
image:${env.BUILD_ID}")
         }
     }
 }
```

4. **Run Step to Start the Container**:

```
stage('Run Container') {
     steps {
         script {
             dockerImage.run("-d --name your-app-container")
         }
     }
 }
```

Task-02: Running Docker Compose

5. **Create Jenkins Project for Docker Compose**:

- Create a new Pipeline project in Jenkins.

- In the pipeline section, add a build script to execute shell commands.

6. **Run** `docker-compose up -d` Command:

```
docker-compose -f path/to/your/docker-compose.yml up -d
```

7. **Set Up Cleanup Step**:

```
docker-compose -f path/to/your/docker-compose.yml down
```

Real-Life Example
Imagine you're working on an e-commerce application with microservices for product catalog, user management, and orders. You've already containerized these services using Docker. Now, you need to automate the build and deployment process to ensure quick and reliable updates.

8. **Pipeline Project for Individual Service**:

- Creating a Jenkins Pipeline project for each microservice, using Docker to build and run each service in its own container, and automating the CI/CD process involves several steps. Here's a step-by-step guide to help you set this up.

Step 1: Setup Your Jenkins Environment

Ensure Jenkins is installed and running, and the necessary plugins are installed:

- Pipeline
- Git
- Docker Pipeline
- NodeJS (if using Node.js microservices)

Step 2: Create a Jenkinsfile for Each Microservice

Each microservice should have its own `Jenkinsfile` in its repository. Here is an example `Jenkinsfile` for a microservice:

```
pipeline {
    agent any
```

```
            environment {
                DOCKER_CREDENTIALS_ID = 'docker-hub-credentials'
// Jenkins ID for Docker Hub credentials
            }

            stages {
                stage('Checkout') {
                    steps {
                        // Checkout code from the repository
        git 'https://github.com/your-repo/microservice-repo.git'
                     }
         }

             stage('Build') {
                steps {
                script {
                  // Build the Docker image
                            docker.build("your-dockerhub-
username/microservice-name:${env.BUILD_NUMBER}")
                        }
                    }
                }

                stage('Test') {
                    steps {
                        script {
                            // Run tests inside the Docker
container
                            docker.image("your-dockerhub-
username/microservice-name:${env.BUILD_NUMBER}").inside {
                                sh 'npm test' // Adjust this
command according to your test command
                            }
                        }
```

```
                }
            }

            stage('Push to Docker Hub') {
                steps {
                    script {

docker.withRegistry('https://index.docker.io/v1/',
DOCKER_CREDENTIALS_ID) {
                                docker.image("your-dockerhub-
username/microservice-name:${env.BUILD_NUMBER}").push()
                            }
                        }
                    }
                }

                stage('Deploy') {
                    steps {
                        // Deploy the Docker container
                        // This step will depend on your
deployment strategy, e.g., Kubernetes, Docker Swarm, etc.
                        // Example for running the container
                        script {
                            sh '''
                            docker run -d --name microservice-
name-${env.BUILD_NUMBER} -p 8080:8080 your-dockerhub-
username/microservice-name:${env.BUILD_NUMBER}

                        }
                    }
                }
            }

            post {
                always {
```

```
                // Clean up workspace after build
                cleanWs()
            }
        }
    }
```

Step 3: Configure Jenkins Pipeline

9. Open your Jenkins dashboard.
10. Create a new pipeline job for each microservice.
11. In the pipeline configuration, select "Pipeline script from SCM".
12. Choose "Git" as the SCM and provide the repository URL where your `Jenkinsfile` is located.
13. Save the configuration.

Step 4: Run the Pipeline

14. Trigger a build manually or configure a webhook to trigger builds automatically on code changes.
15. Monitor the pipeline stages in Jenkins.

Explanation

- **agent any**: Runs the pipeline on any available Jenkins agent.
- **environment**: Sets environment variables, like Docker credentials ID.
- **stages**: Defines the pipeline stages: Checkout, Build, Test, Push to Docker Hub, and Deploy.
- **post**: Specifies actions to be performed after the pipeline run, like cleaning up the workspace.

Notes

- Adjust the commands in the `Jenkinsfile` according to your project's specific needs.
- Ensure your Jenkins instance has Docker installed and configured correctly.
- Use Docker Hub or another Docker registry to store your images.

16. **Pipeline Project for Multi-Service Setup**:

- To create a Jenkins Pipeline project for a multi-service setup using Docker Compose, follow these steps:

Step 1: Setup Your Jenkins Environment

Ensure Jenkins is installed and running, and that the necessary plugins are installed:

- Pipeline
- Git
- Docker Pipeline

Step 2: Prepare Your Project

Ensure your project directory includes a `docker-compose.yml` file that defines all your services. Here is an example `docker-compose.yml`:

```
version: '3.9'

 services:
   web:
     image: "darshif5/simaple-node-app:v1"
     ports:
       - "8000:8000"
```

Step 3: Create a Jenkinsfile

Create a `Jenkinsfile` in the root directory of your project. This file will define the pipeline:

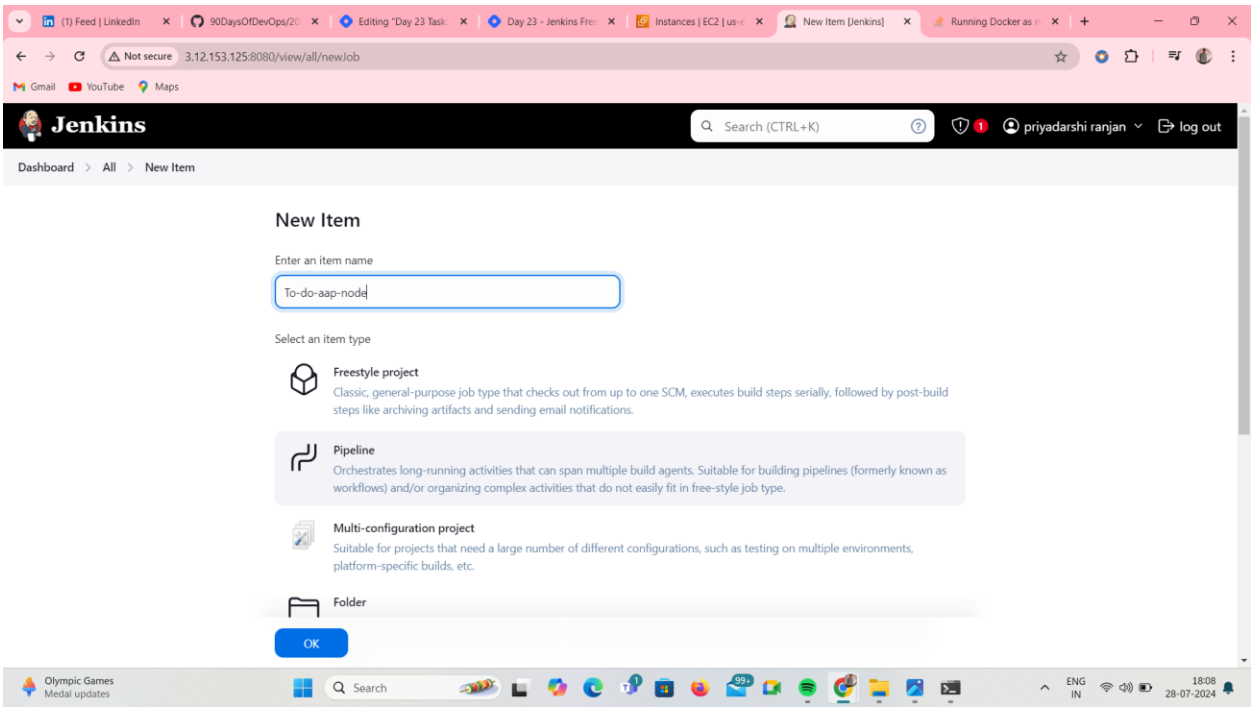
```
pipeline {
     agent any

     stages {
         stage('Code Clone from GitHub') {
```

```
                    steps {
                        echo 'Cloning code from GitHub'
                        git
url:'https://github.com/priyadarshi0811/node-todo-
cicd.git',branch:'Devops'
                    }
                }
                stage('code build') {
                    steps {
                        echo 'code build'
                        sh 'docker build -t simaple-node-app:v1
                    }
                }
                stage('Deploy with Docker Compose') {
                    steps {
                        echo 'Deploying with Docker Compose'
                        sh 'docker-compose down && docker-compos
up -d'
                    }
                }
            }
        }
```

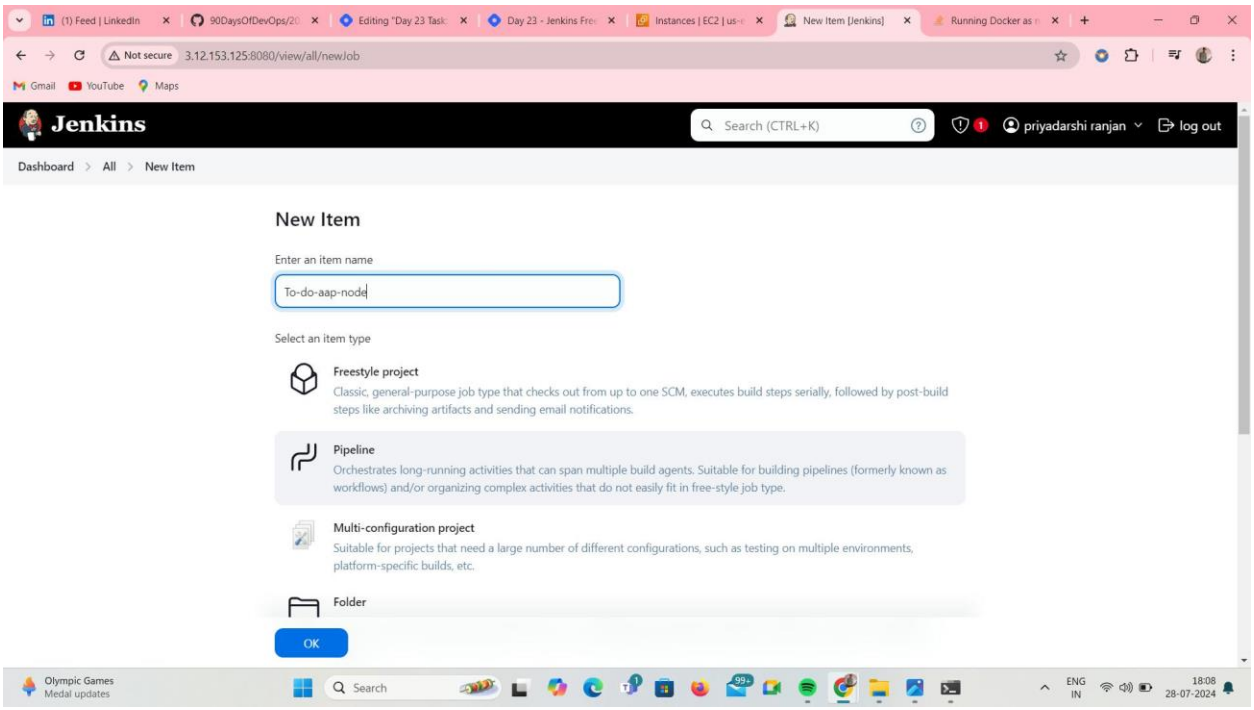
Step 4: Configure Jenkins Pipeline

Open your Jenkins dashboard.

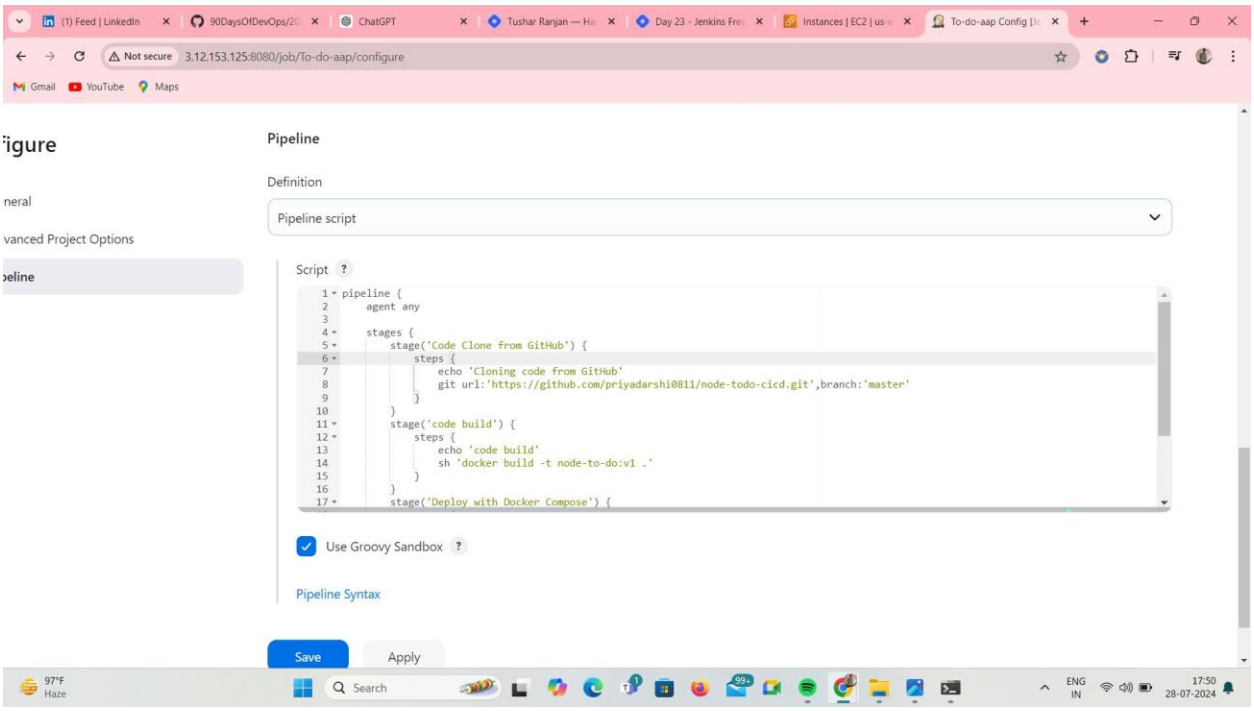


Create a new pipeline job.

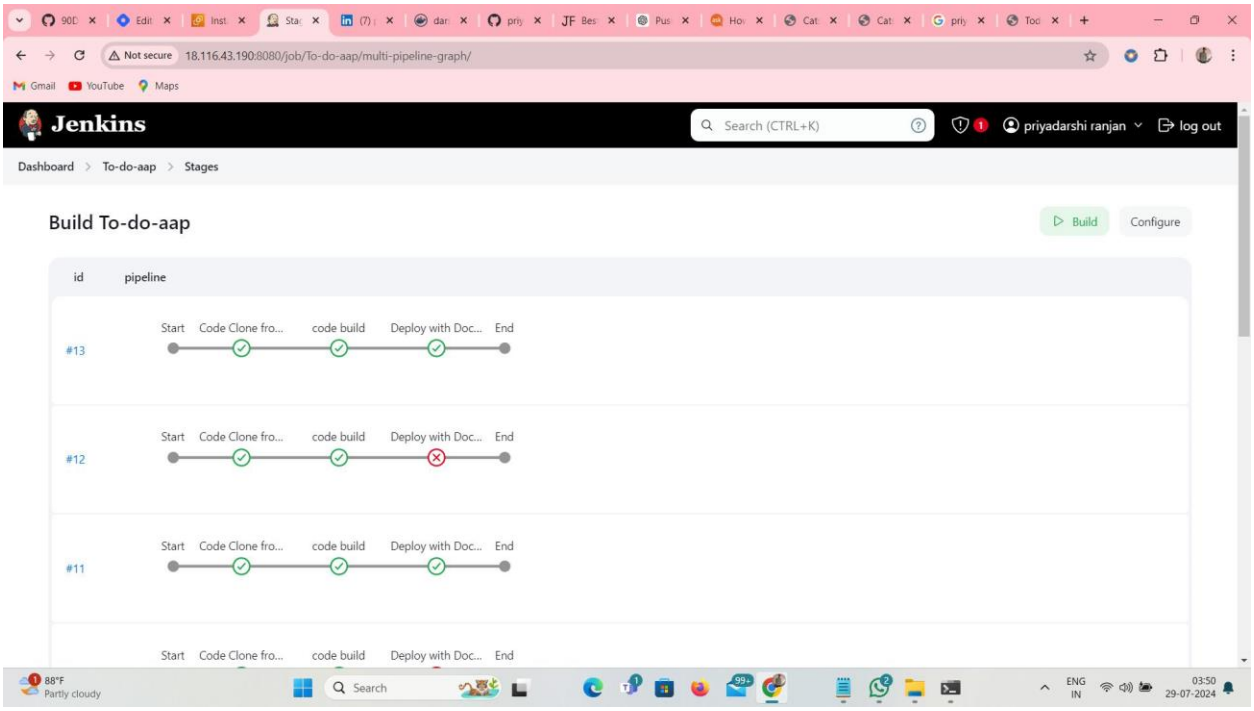
a.

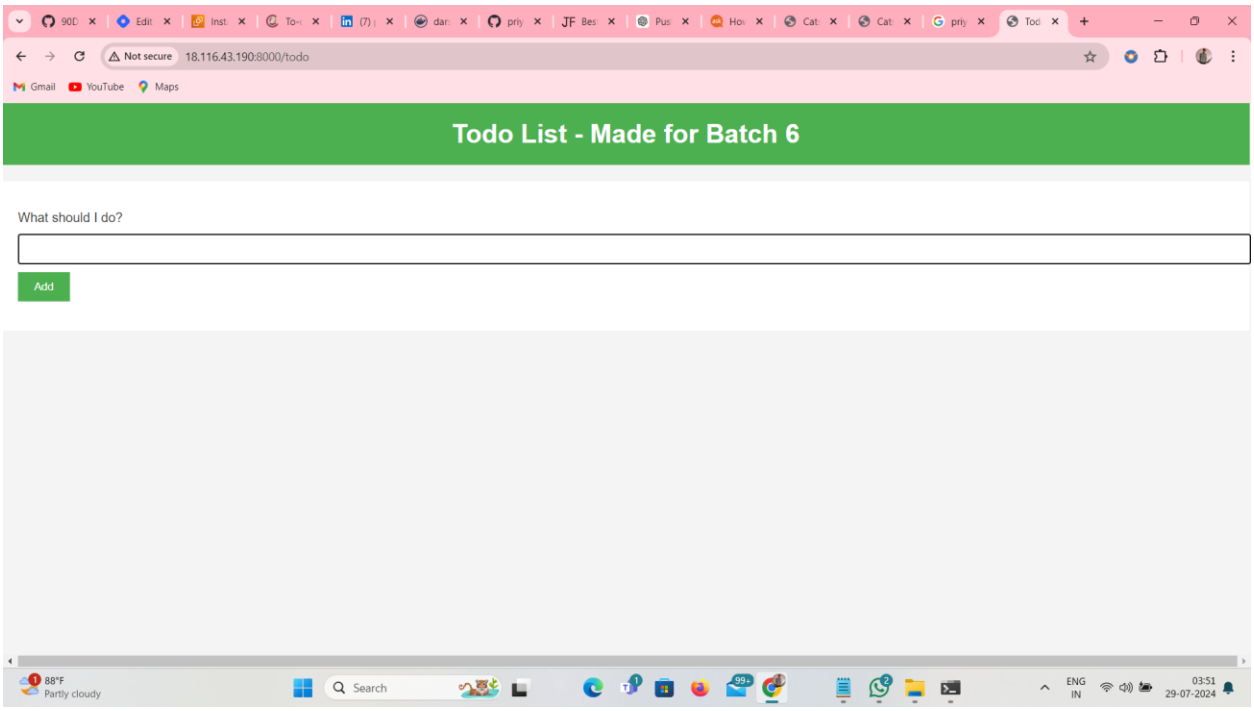


In the pipeline configuration, select "Pipeline script from SCM".



Run the Pipeline





Explanation

- **agent any**: Runs the pipeline on any available Jenkins agent.
- **environment**: Sets environment variables, like Docker credentials ID and Docker Compose project name.
- **stages**: Defines the pipeline stages: Checkout, Build, Test, Push to Docker Hub, and Deploy.
- **post**: Specifies actions to be performed after the pipeline run, like cleaning up the environment.

Notes

- Adjust the commands in the `Jenkinsfile` based on your project's specific needs.
- Ensure your Jenkins instance has Docker installed and configured correctly.
- Use Docker Hub or another Docker registry to store your images.

Conclusion

Creating a Jenkins Freestyle Project provides a hands-on opportunity to understand the fundamentals of CI/CD and automate the build and deployment processes. By breaking down

the tasks into manageable steps, you can efficiently manage individual components and entire application stacks, ensuring quick and reliable software delivery. Keep experimenting, pushing your limits, and learning as you continue your #90daysofdevops journey. Happy DevOps-ing! 🚀

Connect and Follow Me on Socials

LINKDIN | GITHUB |TWITTER