Setting up GitLab and Jenkins with Docker and Configuring a CI/CD Pipeline

Table of Contents

Introduction to Jenkins and GitLab

Jenkins:

Jenkins is a free, open-source tool used to automate tasks in software development. It is mainly used for continuous integration (CI) and continuous delivery (CD). Jenkins helps automatically build, test, and deploy software, which saves time and reduces errors. It can work with many other tools, and users can add extra features through plugins. Jenkins is widely used to help teams catch problems early and keep software development smooth.

GitLab:

GitLab is a platform that provides tools for the entire software development process. It includes features for managing code, tracking tasks, and automating builds and deployments. GitLab combines version control (for managing code changes) with CI/CD tools to make the development process faster and easier. It can be used as a cloud-based service (GitLab.com) or installed on your own servers. GitLab helps teams collaborate and automate many tasks, improving efficiency in software development.

Both Jenkins and GitLab help teams work faster and more efficiently by automating different parts of the software development process. Jenkins is highly customizable with plugins, while GitLab offers an all-in-one solution for managing code and automating tasks.

Prerequisites

- Docker and Docker Compose installed on your machine.
- Basic understanding of Git, Jenkins, and CI/CD pipelines.

Docker Compose Configuration

docker-compose.yml

```yaml
version: '3.8'

services:
  gitlab:
    image: gitlab/gitlab-ce:latest
    container_name: gitlab
    volumes:
      - /srv/gitlab/config:/etc/gitlab
      - /srv/gitlab/logs:/var/log/gitlab
      - /srv/gitlab/data:/var/opt/gitlab
    ports:
      - "443:443"
      - "8080:80"
      - "2222:22"
    restart: always
    networks:
      - gitlab_jenkins_network

  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    privileged: true
    volumes:
      - jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - "8081:8080"
      - "50000:50000"
    environment:
      - no_proxy=127.0.0.1:8080
    restart: always
    networks:
      - gitlab_jenkins_network

volumes:
  jenkins_home:
    driver: local

networks:
  gitlab_jenkins_network:
    driver: bridge
```

Explanation

- **GitLab Service**:
  - Uses the **gitlab/gitlab-ce:latest** image.
  - Stores configuration, logs, and data persistently in **/srv/gitlab**.
  - Maps ports 443, 80 (to 8080), and 22 (to 2222).
  - Automatically restarts if it crashes.
- **Jenkins Service**:
  - Uses the **jenkins/jenkins:lts** image.
  - Maps the Jenkins home directory to a Docker volume.
  - Enables Docker access inside Jenkins using **/var/run/docker.sock**.
  - Maps ports 8081 (to 8080) and 50000 (default Jenkins agent port).
  - Configures no-proxy for localhost.

Setting Up Jenkins

Access Jenkins Container

```
docker exec -it -u root jenkins bash
```

Retrieve Initial Admin Password

```
cat /var/jenkins_home/secrets/initialAdminPassword
```

Install Docker Inside Jenkins

```
apt update
apt install docker.io -y
apt install vim
apt install sudo
groupadd docker
usermod -aG docker jenkins
docker info
```

Grant Jenkins Permission for Docker Commands

Edit the sudoers file:

```
vim /etc/sudoers
```

```
jenkins ALL=(ALL) NOPASSWD: /usr/bin/docker
```

Setting Up GitLab

Retrieve Initial Root Password

```
sudo cat /srv/gitlab/config/initial_root_password
```

Generate Access Token in GitLab

1. Log in to your GitLab instance using the root account.
2. Go to User Settings > Access Tokens.
3. Create a new Personal Access Token with the following scopes:
    - api
    - read_repository
4. Copy the generated token and save it securely (you won't be able to view it again).

Configuring Jenkins to Connect to GitLab

Add Credentials in Jenkins

1. Open Jenkins in your browser.
2. Navigate to Manage Jenkins > Credentials > System > Global credentials (unrestricted).
3. Click Add Credentials and select Username with password.
4. Fill in the following fields:
    - Username: GitLab username or email.
    - Password: The Personal Access Token generated in GitLab.
    - ID: A unique identifier, e.g., **Jenkins-Git**.
5. Click OK to save.

Configuring the Git Repository

Create Repository in GitLab

1. Log in to your GitLab dashboard.
2. Navigate to Projects > Create New Project.
3. Enter a project name (e.g., **quizz**) and set the visibility level.
4. Click Create Project.

Configuring the Git Repository

Initialize Repository

```
git init
```

```
git add .
```

```
git status
```

Add Remote Repository

```
git remote -v
```

If a remote repository already exists, remove it:

```
git remote remove origin
```

Add the new remote repository:

```
git remote add origin http://localhost:8080/root/quizz.git
```

Commit and Push Changes

```
git commit -m "commit message"
git push -u origin main/master
```

Jenkins CI/CD Pipeline

Pipeline Script

The following is a declarative pipeline script to automate building and deploying the project.

```
pipeline {
    agent any

    environment {
        PROJECT_NAME = 'quiz-app'
        IMAGE_NAME = 'quiz-app-image'
        TAG = "${BUILD_NUMBER}"
        CONTAINER_NAME = 'quiz-app-container'
    }

    stages {
        stage('Git Checkout') {
            steps {
                git branch: 'main',
                    credentialsId: 'Jenkins-Git',
                    url: 'http://{base_machine_IP}:8080/root/quizz.git'
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    sh """
                   sudo docker build -t ${IMAGE_NAME}:${TAG} .
                    """
                }
            }
        }

        stage('Remove Old Docker Container') {
            steps {
                script {
                    sh """
                   sudo docker ps -a -q -f name=${CONTAINER_NAME} |
xargs -r sudo docker stop
                   sudo docker ps -a -q -f name=${CONTAINER_NAME} |
xargs -r sudo docker rm
                    """
```

```
                }
            }
        }

        stage('Run New Docker Container') {
            steps {
                script {
                    sh """
                   sudo docker run -d --name ${CONTAINER_NAME} -p
8083:80 ${IMAGE_NAME}:${TAG}
                    """
                }
            }
        }
    }

    post {
        success {
            echo "Pipeline succeeded!"
        }
        failure {
            echo "Pipeline failed!"
        }
    }
}
```

Explanation

- **Git Checkout:** Pulls the latest code from the main branch of the specified Git repository using stored credentials.
- **Build Docker Image:** Builds a Docker image for the project with a unique tag based on the Jenkins build number.
- **Remove Old Docker Container:** Stops and removes any existing containers with the same name as the new container.
- **Run New Docker Container:** Starts a new container from the built image and maps port 8083 to the container's port 80.
- **Post-Build Actions:** Displays success or failure messages based on the pipeline execution.

Additional Considerations

1. **SSL Configuration**: For production environments, ensure you use SSL (HTTPS) for both GitLab and Jenkins. You can use a reverse proxy like Nginx or configure SSL certificates within Docker containers.
2. **Security**: Ensure that your GitLab and Jenkins instances are secured by setting up proper access controls, user roles, and permissions to protect sensitive data.
3. **Webhooks**: Set up webhooks in GitLab to automatically trigger Jenkins builds on commit or push events. This can be configured in **GitLab > Project Settings > Webhooks**.
4. **Monitoring and Alerts**: Use monitoring tools (such as Prometheus, Grafana) to keep track of Docker container health and Jenkins job statuses.