Lab10: Jenkins for DevOps Automation Using Python (CI/CD Pipeline)

Objective

You will learn how to set up a Jenkins pipeline for continuous integration and continuous deployment (CI/CD) using Python. You will understand how to automate testing and deployment processes using Jenkins, integrating these steps into a CI/CD pipeline.

Prerequisites

- Basic understanding of Python programming.
- Familiarity with Git and GitHub (or any Git repository hosting service).
- Jenkins installed on your local machine or accessible via a server.
- Python and pip installed on your machine.

Part 1: Setting Up Jenkins

1. **Install Jenkins**:
    - Ensure Jenkins is installed on your machine. You can download and install Jenkins from the official Jenkins website:

    https://www.jenkins.io/download/thank-you-downloading-windows-installer-stable/

    https://www.jenkins.io/doc/book/installing/windows/

    - After installation, start Jenkins by running the following command:

java -jar jenkins.war

    - Open Jenkins in your browser by navigating to http://localhost:8080/.
2. **Set Up Jenkins**:
    - Follow the initial setup instructions, including unlocking Jenkins, installing suggested plugins, and creating your first admin user.
    - Install the following Jenkins plugins:

- **Git Plugin**: To integrate with Git repositories.
- **Pipeline Plugin**: To enable the use of Jenkins Pipelines.
- **SSH Agent Plugin**: For deploying to remote servers via SSH.

Part 2: Setting Up a Python Project

1. **Create a Python Project**:
    - Set up a new Python project in a directory named jenkins_project.
    - Initialize a Git repository in this directory:

```
git init
```

    - Create a basic Python script, app.py, with the following content:

```
def greet(name):
    return f"Hello, {name}!"

if __name__ == "__main__":
    print(greet("World"))
```

2. **Set Up Unit Tests**:
    - Create a tests/ directory with a basic unit test for app.py:

```
# tests/test_app.py
import unittest
from app import greet

class TestApp(unittest.TestCase):

    def test_greet(self):
        self.assertEqual(greet("World"), "Hello, World from FirstName LastName!")
```

```
if __name__ == "__main__":
    unittest.main()
```

3. **Create a requirements.txt File**:

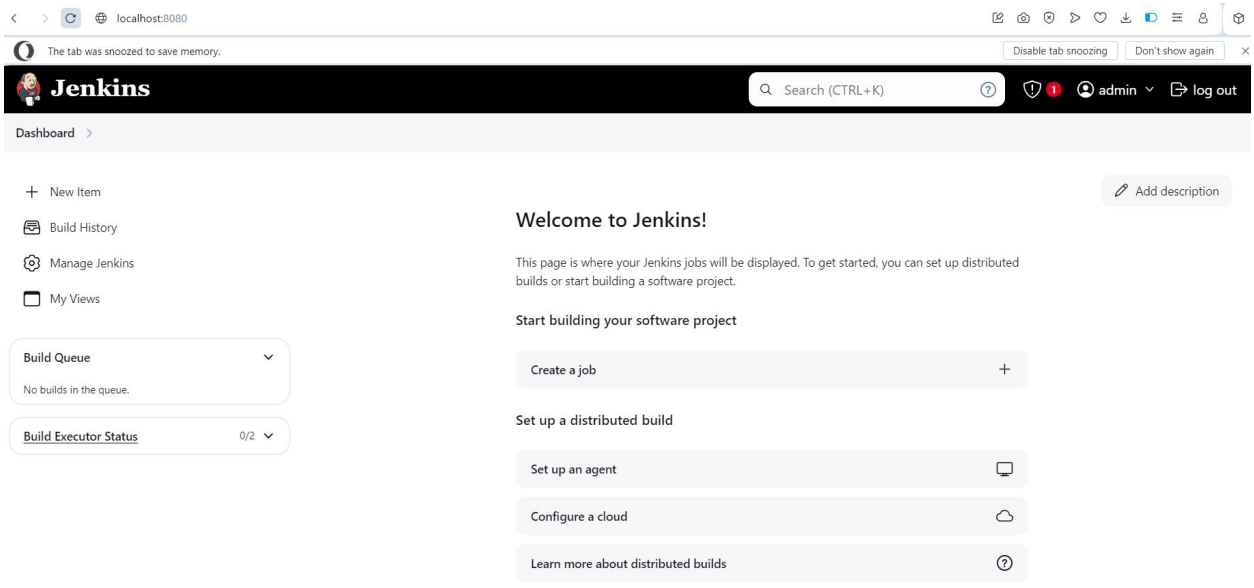
   - Create a requirements.txt file to list the project dependencies:

```
flake8
```

Part 3: Creating a Jenkins Pipeline Exercise to be graded

1. **Create a Jenkins Job**:

   - Go to Jenkins Dashboard, click on "New Item", and create a new **Pipeline** job named Python-CI-CD.



   - In the job configuration, under the **Pipeline** section, select **Pipeline script from SCM**.
   - Set **SCM** to **Git** and provide the URL of your Git repository (e.g., https://github.com/yourusername/jenkins_project.git).
   - Specify the branch to build, usually main or master.

2. **Write a Jenkinsfile**:

- In the root of your project directory, create a file named Jenkinsfile_FirstName_LastName:

```
pipeline {
    agent any

    environment {
        VIRTUAL_ENV = 'venv'
    }

    stages {
        stage('Setup') {
            steps {
                script {
                    if (!fileExists("${env.WORKSPACE}/${VIRTUAL_ENV}")) {
                        sh "python -m venv ${VIRTUAL_ENV}"
                    }
                    sh "source ${VIRTUAL_ENV}/bin/activate && pip install -r requirements.txt"
                }
            }
        }

        stage('Lint') {
            steps {
                script {
                    sh "source ${VIRTUAL_ENV}/bin/activate && flake8 app.py"
                }
            }
        }
```

```
        stage('Test') {
            steps {
                script {
                    sh "source ${VIRTUAL_ENV}/bin/activate && pytest"
                }
            }
        }

        stage('Deploy') {
            steps {
                script {
                    // Deployment logic, e.g., pushing to a remote server
                    echo "Deploying application..."
                }
            }
        }
    }

    post {
        always {
            cleanWs()
        }
    }
}
```

- **Explanation**:
  - **Setup**: Creates a Python virtual environment and installs dependencies.
  - **Lint**: Runs flake8 to check code style.
  - **Test**: Runs the unit tests using pytest.
  - **Deploy**: Placeholder for deployment logic.

3. **Run the Pipeline**:
    - Commit your Jenkinsfile to your repository and push it to GitHub.
    - Trigger a build in Jenkins either manually or by pushing changes to the repository.
    - Observe the pipeline running through the stages: Setup, Lint, Test, and Deploy.
    - Submit your github link to moodle in a text file.

Part 4: Exercise to be graded

1. **Objective**:
    - Modify the existing Jenkins pipeline to include additional steps such as security scanning and code coverage analysis.
2. **Instructions**:
    - **Step 1: Code Coverage**:
        - Add code coverage analysis to the pipeline using the coverage.py tool.
        - Modify the Jenkinsfile to include a new stage named Coverage that runs coverage and generates a report.
    - **Step 2: Security Scanning**:
        - Integrate a security scanning tool like bandit into the pipeline.
        - Modify the Jenkinsfile to include a new stage named Security Scan that runs bandit and checks for vulnerabilities.
    - **Step 3: Deployment**:
        - Implement a basic deployment script to deploy the application to a remote server or a simple local deployment.
3. **Expected Output**:
    - A modified Jenkinsfile that includes the additional stages.
    - A successful Jenkins build that passes all stages, including the new code coverage and security scan stages.
4. **Submission**:
    - Submit the modified Jenkins file, along with a brief report explaining the changes made and the output of the Jenkins pipeline after running the build.

Expected Outcome

By the end of this lab, you should have:

- A working Jenkins pipeline that automates testing and deployment for a Python project.
- Experience in integrating additional steps such as code coverage and security scanning into a CI/CD pipeline.
- An understanding of how to use Jenkins for automating DevOps tasks in Python projects.

Resourses:

https://www.jenkins.io/doc/pipeline/tour/hello-world/