Week3-day3

Jenkins CI/CD Pipeline with Python Project

It includes creating a project directory, initializing a repository, writing a Jenkins pipeline, and automating testing and deployment.

Project Setup

Step 1: Create Project Directory & Initialize Git

```
mkdir my_python_project && cd my_python_project
git init
```

Step 2: Create `pyproject.toml`

```
cat << EOF > pyproject.toml
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[project]
name = "my_python_project"
version = "0.1.0"
dependencies = []

[project.scripts]
myapp = "my_python_project.main:main"
EOF
```

Step 3: Create Source Code and Entry Point

```
mkdir -p src/my_python_project
cat << EOF > src/my_python_project/main.py
def main():
    print("Hello from Jenkins CI/CD Pipeline!")

if __name__ == "__main__":
    main()
EOF
```

Step 4: Write Unit Tests

```
mkdir tests
cat << EOF > tests/test_main.py
import unittest
from my_python_project.main import main
import io
import sys

class TestMain(unittest.TestCase):
    def test_main(self):
        captured_output = io.StringIO()
        sys.stdout = captured_output
        main()
        sys.stdout = sys.__stdout__
```

```
        self.assertEqual(captured_output.getvalue().strip(), "Hello from Jenkins
CI/CD Pipeline!")

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

Python Project Structure

- `pyproject.toml`: Defines dependencies and build settings.
- `src/`: Contains project source code.
- `tests/: Stores unit tests.`

- `pytest`: Runs all test cases in `tests/` directory.
- `Ensures code correctness before deployment.`

Step 5: Create Dockerfile

```
cat << EOF > Dockerfile
FROM python:3.9-slim

WORKDIR /app

COPY dist/*.whl .
RUN pip install *.whl

CMD ["myapp"]
EOF
```

Docker

- `Dockerfile`: Defines the image.
- `docker build -t <tag> .`: Builds a Docker image.
- `docker run -d --name <container-name> <image>: Runs a container.`

Step 6: Create `.gitignore`

```
cat << EOF > .gitignore
__pycache__/
*.py[cod]
*$py.class
dist/
build/
*.egg-info/
venv/
EOF
```

Step 7: Build the Python Wheel

```
pip install build
python -m build --wheel
```

Step 8: Commit to Git

```
git add .
git commit -m "Initial project setup with Python code and Dockerfile"
```

Jenkins Pipeline Setup

Jenkins CI/CD Pipeline

- **Stages:**
    - `Checkout`: Clones the repository.
    - `Build Wheel`: Builds the Python package.
    - `Test`: Runs unit tests using `pytest`.
    - `Build Docker Image`: Creates a containerized app.
    - `Deploy`: Stops old containers and runs the new one.
- **Post Actions:**
    - `success`: Displays a success message.
    - `failure`**: Displays an error message.**

Step 9: Create `Jenkinsfile`

```
cat << EOF > Jenkinsfile
pipeline {
    agent any

    environment {
        DOCKER_IMAGE = 'my-python-project:latest'
    }

    stages {
        stage('Checkout') {
            steps {
                dir('/home/user/my_python_project') {
                    git branch: 'main', url:
'file:///home/user/my_python_project'
                }
            }
        }

        stage('Build Wheel') {
            steps {
                sh 'pip install build'
                sh 'python -m build --wheel'
            }
        }

        stage('Test') {
            steps {
                sh 'pip install pytest'
                sh 'pytest tests/'
            }
        }

        stage('Build Docker Image') {
            steps {
                sh 'docker build -t ${DOCKER_IMAGE} .'
            }
        }

        stage('Deploy') {
            steps {
```

```
                sh 'docker stop my-python-container || true'
                sh 'docker rm my-python-container || true'
                sh 'docker run -d --name my-python-container ${DOCKER_IMAGE}'
            }
        }
    }

    post {
        success {
            echo 'Pipeline completed successfully!'
        }
        failure {
            echo 'Pipeline failed!'
        }
    }
}
EOF
```

Step 10: Commit the `Jenkinsfile` to Git

```
git add Jenkinsfile
git commit -m "Add Jenkinsfile for CI/CD pipeline"
```

Step 11: Display Project Structure

```
echo "Project structure created:"
ls -R
```