**CI/CD for Python Application Using Jenkins and Docker**

By Anu John

In this article, we will cover how to set up CI/CD for Python applications using Jenkins and Docker.

In today’s fast-paced software development environment, continuous integration and continuous deployment (CI/CD) pipelines have become essential for delivering high-quality applications efficiently. By automating the build, test, and deployment processes, CI/CD enables developers to focus on coding while ensuring seamless integration and deployment of updates.

This article provides a step-by-step guide on setting up a CI/CD pipeline for a Python application using Jenkins and Docker Hub. We will explore how to configure Jenkins to automate the building and testing of your Python application and utilize Docker Hub for containerizing and deploying it. Whether you’re new to CI/CD or looking to streamline your existing workflow, this guide will help you integrate these powerful tools effectively.

**Prerequisites**

Install Jenkins, Docker, flask and pytest in a single server

**Jenkins Setup for a Python Application CI/CD**

Download **Docker Pipeline** and **Pipeline Stage view** plugins

* Add GitHub and Docker Hub credentials by selecting the “Username with password” and “Secret text” options. To do this, navigate to **Manage Jenkins** -> **Manage Credentials** -> **System (Global credentials)** -> **Add Credentials**. Ensure you note down the ID assigned to each credential, as these will be referenced in the Jenkins pipeline script.
* Set the PATH variable in Jenkins to match the one on your local machine, including the module’s directory. To do this, go to **Manage Jenkins** -> **Configure System**. Under **Global Properties**, check the **Environment variables** option and add a new variable with the name PATH and its corresponding value as shown below

Global Properties

Enviornmental variables

Name :githubcredentials

Value:github

Name:registry

Value:anusoften/python-jenkins

Name: registryCredentials

Value:dockerhub

**Use below commands to allow Jenkins users to access the docker socket:**

sudo usermod -aG docker jenkins

After that restart jenkins:

sudo systemctl restart jenkins

Without the above configurations, Jenkins will not be able to access Docker Hub.

**CI/CD for a Python Application Using Jenkins and Docker Hub**

**Python Codes:**

**Create files**

* 1. **app.py:**

from flask import Flask app = Flask(\_\_name\_\_) @app.route('/') def home(): return "<h1>Welcome to Jenkins Tutorials</h1>" if \_\_name\_\_ == '\_\_main\_\_': app.run(debug=True, host='127.0.0.1') # Host set to 0.0.0.0 to make the app accessible from any IP

**2.testRoutes.py:**

import pytest

from app import app

@pytest.fixture

def client():

with app.test\_client() as client:

yield client

def test\_home(client):

"""Test the home route"""

response = client.get('/')

assert response.status\_code == 200

assert b"Welcome to Jenkins Tutorials" in response.data

* 1. **Dockerfile:**

# Use Python 3.7 image as the base image

FROM python:3.7-slim

# Set the working directory in the container

WORKDIR /app

# Copy the requirements file into the container

COPY requirements.txt /app/

# Install dependencies

RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container

COPY . /app/

# Expose the port that the Flask app will run on

EXPOSE 5000

# Set the command to run the Flask app

CMD ["python", "app.py"]

* 1. **requirements.txt:**

Flask

pytest

**Jenkins Pipeline Code:**

pipeline {

environment {

registry = "anusoften/python-jenkins" // Docker Hub image repository

registryCredential = 'dockerhub' // Docker Hub credentials ID in Jenkins

githubCredential = 'github' // GitHub credentials ID in Jenkins

}

agent any

stages {

stage('Checkout') {

steps {

// Checkout the GitHub repository

git branch: 'main',

credentialsId: githubCredential,

url: https://github.com/softencloud/Python-d0cker-jenkinpipeline.git’

}

}

stage('Test') {

steps {

// Run tests using pytest

sh "pytest testRoutes.py"

}

}

stage('Clean Up') {

steps {

// Stop and remove any containers related to the job

sh returnStatus: true, script: 'docker stop $(docker ps -a -q --filter "name=${JOB\_NAME}")'

sh returnStatus: true, script: 'docker rm $(docker ps -a -q --filter "name=${JOB\_NAME}")'

// Remove Docker images related to the registry

sh returnStatus: true, script: 'docker rmi $(docker images -q --filter "reference=${registry}") --force'

}

}

stage('Build Image') {

steps {

script {

// Construct the image name with the registry and build ID

img = "${registry}:${env.BUILD\_ID}"

println("Building Docker image: ${img}")

// Build the Docker image

dockerImage = docker.build("${img}")

}

}

}

stage('Push To DockerHub') {

steps {

script {

// Push the Docker image to Docker Hub

docker.withRegistry('https://registry.hub.docker.com', registryCredential) {

dockerImage.push()

}

}

}

}

stage('Deploy') {

steps {

// Remove any existing container with the same name and run a new container

sh returnStatus: true, script: 'docker rm -f ${JOB\_NAME} || true'

sh label: '', script: "docker run -d --name ${JOB\_NAME} -p 5000:5000 ${img}"

}

}

}

}

Use the above Pipeline code in Jenkins Configuration and modify **Environment variables** and **Repositories** and click on **Build**