**Python Flask CI/CD with Jenkins and Docker**

This guide provides the necessary files and steps to set up a CI/CD pipeline for a simple Python Flask application using Jenkins and Docker.

**1. Prerequisites**

Before you begin, ensure you have Docker installed. To run Jenkins and allow it to manage Docker containers on your host, you need to mount the Docker socket.

Run the following command to start a Jenkins container:

docker run -d \

--name jenkins \

-v /var/run/docker.sock:/var/run/docker.sock \

-p 8080:8080 \

-p 5000:5000 \

jenkins/jenkins:lts

* -v /var/run/docker.sock:/var/run/docker.sock: This volume mount allows the Jenkins container to use the Docker daemon from the host.
* -p 8080:8080: Exposes the Jenkins web UI.
* -p 5000:5000: Exposes the Flask app's port from the Docker container to the host.

**2. Application Files**

The app.py, requirements.txt, and Dockerfile define your application and its container image. The Jenkinsfile orchestrates the automation.

**3. Setting up the Jenkins Job**

1. Access the Jenkins UI at http://localhost:8080 and complete the initial setup.
2. Install the "Pipeline" plugin if it's not already installed.
3. Create a new job and select "Pipeline".
4. In the "Pipeline" section of the job configuration, select "Pipeline script" and copy the contents of the Jenkinsfile below into the text area.
5. Save the job.

**4. Executing the Pipeline**

Click "Build Now" to run the pipeline. Jenkins will automatically execute the stages defined in the Jenkinsfile:

* **Build**: Creates the Docker image.
* **Test**: A placeholder for running tests (you can add your own test script).
* **Deploy**: Stops any existing container and runs a new one from the built image.

**5. Verification**

Once the pipeline completes successfully, your Flask application will be running on port 5000 of your host machine. You can verify this by visiting http://localhost:5000 in your web browser.

You should see the message: "Hello, Jenkins CI/CD!"

# app.py

# A simple Flask application to demonstrate CI/CD.

from flask import Flask

# Create the Flask app instance.

app = Flask(\_\_name\_\_)

# Define the root route.

@app.route('/')

def home():

    """

    Returns a simple greeting for the main page.

    """

    return "Hello, Jenkins CI/CD!"

# Check if the script is the main entry point and run the app.

if \_\_name\_\_ == '\_\_main\_\_':

    # Run the app on all available network interfaces on port 5000.

    app.run(host='0.0.0.0', port=5000)

**Dockerfile**

**A multi-stage Dockerfile to create a small image for the Flask application.**

**Use a lightweight Python base image.**

FROM python:3.9-slim

**Set the working directory inside the container.**

WORKDIR /app

**Copy the requirements file and install dependencies.**

**This step is cached, so it only runs if requirements.txt changes.**

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

**Copy the rest of the application code.**

COPY . .

**Expose the port the Flask application runs on.**

EXPOSE 5000

**Define the command to run the application when the container starts.**

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

// Jenkinsfile

// A declarative pipeline script for building and deploying a Flask application with Docker.

pipeline {

// Agent is set to 'any', meaning the pipeline can run on any available Jenkins agent.

// For more complex setups, you might use a specific label or Docker image.

agent any

// Define environment variables for the project.

environment {

// The name of the Docker image to be built.

IMAGE\_NAME = 'my-flask-app'

// The name for the running container.

CONTAINER\_NAME = 'my-flask-app-container'

}

stages {

// Stage to build the Docker image for the application.

stage('Build') {

steps {

script {

// Use the `sh` command to execute the Docker build command directly.

// The `-t` flag tags the image with the specified name.

// The `.` indicates that the Dockerfile is in the current directory.

echo "Building Docker image..."

sh "docker build -t ${IMAGE\_NAME} ."

echo "Docker image built successfully."

}

}

}

// Stage to run tests. This is a placeholder; you would add your test commands here.

stage('Test') {

steps {

echo "Running tests..."

// Example:

// sh 'python -m pytest'

echo "Tests passed."

}

}

// Stage to deploy the application by running the Docker container.

stage('Deploy') {

steps {

script {

echo "Deploying application..."

// Check if a container with the same name already exists and stop it if it does.

// The `|| true` prevents the script from failing if the container doesn't exist.

echo "Stopping existing container..."

sh "docker stop ${CONTAINER\_NAME} || true"

// Remove the old container to avoid naming conflicts.

echo "Removing old container..."

sh "docker rm ${CONTAINER\_NAME} || true"

// Run the new container in detached mode (`-d`).

// Assign a name (`--name`) and map the ports (`-p`).

// The image is specified by its tag.

echo "Starting new container..."

sh "docker run -d --name ${CONTAINER\_NAME} -p 5000:5000 ${IMAGE\_NAME}"

echo "Deployment complete! Application is running on port 5000."

}

}

}

}

}