**FLASK URL SHORTENER – DOCKERIZED & DEPLOYED VIA JENKINS**

**Table of Contents:**

* Project Summary
* Project Source
* DevOps Objective
* Tech Stack
* Project Structure
* Docker Setup
* Jenkins CI/CD Pipeline
* How to Deploy
* Test Stage
* Screenshots
* Future Enhancements
* Author
* Reference

**Project Summary:**

This project demonstrates the DevOps lifecycle implementation on a Python Flask-based URL Shortener application. The original source code was cloned and enhanced by integrating Docker for containerization and Jenkins for automated CI/CD deployment on an EC2 instance.

**Problem Statement:** The original Flask URL shortener worked locally but lacked automated deployment and environment consistency. Manual setup made it hard to scale, test, or maintain efficiently.

**Solution:** Docker was used to containerize the app, ensuring consistent environments. Jenkins automated the CI/CD pipeline to build, test, push, and deploy the app to an EC2 server, enabling fast, reliable, and repeatable deployments.

**Project Source:**

* The application source was cloned from:

<https://github.com/ezhil56x/URL-Shortener.git>

**DevOps Objective:**

* + - Dockerize the Python Flask application.
    - Build Docker images through Jenkins.
    - Test stage using pytest in Jenkins.
    - Push images to Docker Hub.
    - Deploy the container to an EC2 environment.

**Tech Stack:**

App : Python, Flask, SQlite .

Containerization : Docker.

CI/CD : Jenkins(Pipeline).

Deployment : EC2(Ubuntu).

Source Control : GitHub.

**Project Structure:**

URL\_JenDoc/

│

├── app.py

├── requirements.txt

├── Dockerfile

├── Jenkinsfile

├── test\_app.py

└── templates/

└── index.html

**Docker Setup:**

**Dockerfile:**

FROM python:3.10-slim

WORKDIR /app

COPY requirements.txt .

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

COPY . .

EXPOSE 5000

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

**Jenkins CI/CD Pipeline:**

**Jenkinsfile:**

pipeline {

agent any

environment {

IMAGE = 'clitues2604/url-jendoc' **// DockerHub Username**

PORT = '5000' }

stages {

stage('Test') { agent {

docker {

image 'python:3.10'

args '--user root' }}

steps { sh '''

pip install -r requirements.txt

pip install pytest

pytest || exit 1 '''} }

stage('Build & Push Image') {

steps {

script {

docker.build("${IMAGE}:latest")

docker.withRegistry('https://index.docker.io/v1/', 'dockerhub-creds') {

docker.image("${IMAGE}:latest").push() }}}}

stage('Deploy') {

steps { sh '''

docker rm -f url\_jendoc || true

docker run -d --name url\_jendoc -p ${PORT}:${PORT} ${IMAGE}:latest '''}} }

post {

always {

echo 'Pipeline finished.' }

failure {

echo 'Pipeline failed.'

}

success {

echo 'Application successfully deployed!' }}}

**Jenkins Setup:**

* Job type: Pipeline.
* SCM: GitHub repo.
* Docker installed on EC2.
* Docker Hub credentials set in Jenkins.

**Test Stage:**

**test\_app.py:**

from app import app

def test\_homepage():

response = app.test\_client().get('/')

assert response.status\_code == 200

**How to Process:**

**Step-by-Step:**

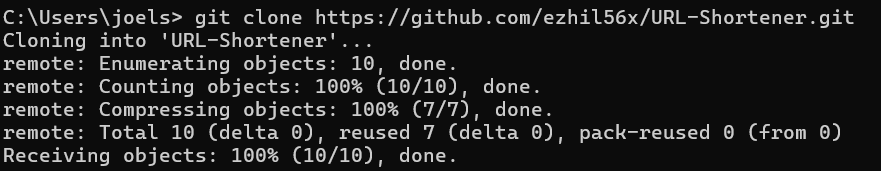
1. Push code to GitHub.
2. Install & Setup Jenkins & Docker.
3. Create & Setup Pipeline.
4. Jenkins pulls repo via SCM.
5. Jenkins will:

* Run tests.
* Build Docker image.
* Push to Docker Hub.
* Deploy container on EC2 instance.

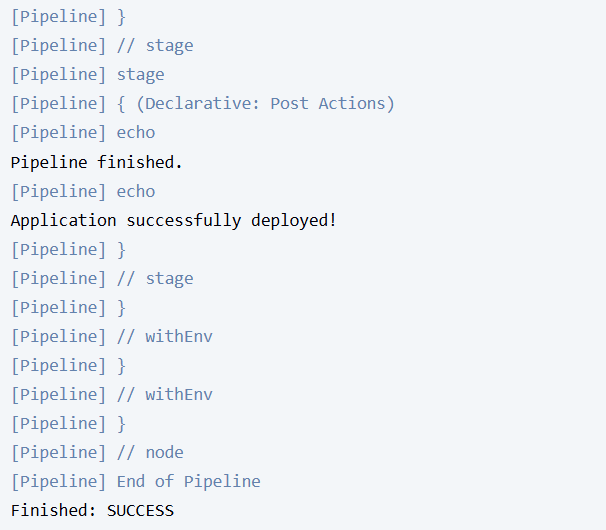
1. Access the app on: **http://<ec2-public-ip>:5000**

**Screenshots:**

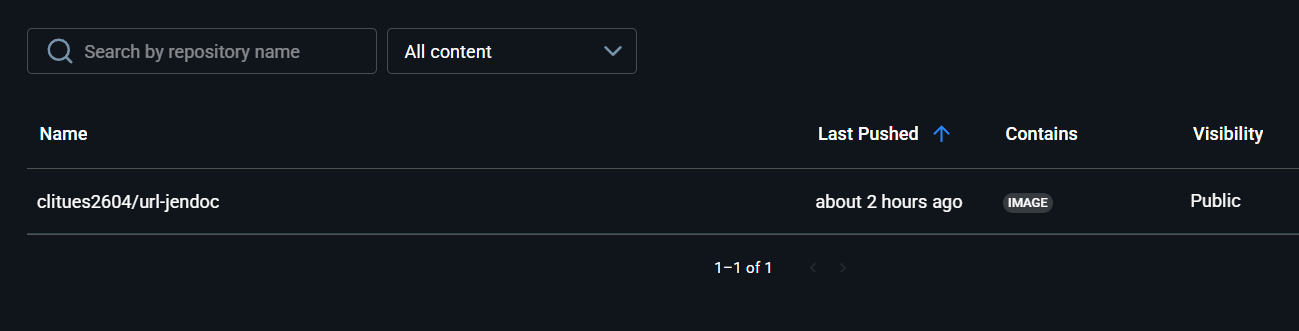
1. **GitClone:**



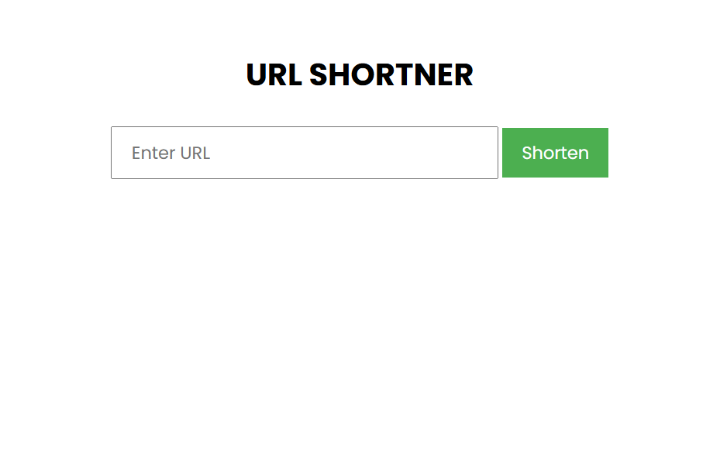
1. **Pipeline:**

****

1. **DockerHub:**

****

1. **App:**

****

**Author:**

**DevOps Work:** Joel Seba Clitues J

**References:**

**Jenkins Docs:** <https://www.jenkins.io/doc/>

**Docker Docs:** <https://docs.docker.com/>