# README: Flask Web Application Deployment Guide

This guide focuses on implementing core DevOps principles by setting up a simple web application, Dockerizing it, and automating its deployment using Ansible on a local machine. It is tailored to help beginners understand the process and achieve successful deployment with ease.

**Step 1: Set Up the Flask Application**

1. **Create a project directory:** Create a directory named flask\_app where the Flask application will reside.

mkdir flask\_app

cd flask\_app

1. **Create the app.py file:** Use the following command to create the app.py file.

touch app.py

1. **Install Flask:** Install Flask using pip (Python's package manager). If Flask is not already installed, use:

pip install flask

1. **Add Flask Application Code:** Copy the basic Flask app code from GitHub and paste it into the app.py file. You can open the file using:

sudo nano app.py

1. **Set Up Python Virtual Environment:** To manage dependencies, create a virtual environment by running:

python3 -m venv venv

source venv/bin/activate

1. **Create a requirements.txt file:** Create a requirements.txt file and add Flask as a dependency.

echo "flask" > requirements.txt

**Step 2: Dockerizing the Application**

Docker allows you to containerize the Flask app for consistent deployment. Follow these steps to Dockerize your Flask app.

1. **Create Dockerfile:** In your project directory, create a Dockerfile. Add the following content to the Dockerfile:

DFROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt

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

COPY . .

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

This file sets up the base image, installs required dependencies, and runs the Flask app.

1. **Build and Run the Docker Container:** To build the Docker image and run the container, use the following commands:

docker build -t flask-app .

docker run -p 5000:5000 flask-app

This will run your Flask app in a container and expose it on port 5000.

### Step 3: Basic Monitoring

1. **Docker Stats:** Use docker stats to monitor the CPU and memory usage of your running containers

**first check what docker is running by doing following command**.

docker ps

1. then checkCPU and memory usage

docker stats

**Step 4: Application Logs:** Implement logging in your application to track events and debug issues

1. update the app.py

import logging

logging.basicConfig(level=logging.INFO)

1. Then, modify your home() route function to include logging whenever it is accessed:

@app.route('/')

def home():

logging.info("Home page accessed")

return "Hello, World!"

1. **Restart the Flask App:**

* After saving the changes, restart the Flask app so the logging takes effect:

python3 app.py

1. **View Logs in the Terminal:**

* When you access http://localhost:5000 in your browser and hit the home route, you will see the log message in the terminal where your Flask application is running. It will look like this:

INFO:root:Home page accessed

**Step 5: Automating Deployment with Ansible**

**Ansible** is an open-source automation tool that simplifies configuration management, application deployment, and task automation. It uses human-readable YAML playbooks to define automation tasks

Now, let’s automate the deployment process using **Ansible**. Ansible will help us deploy the application to an AWS Kali machine.

1. **Install Ansible on the AWS Kali Machine:** First, update your system and install Ansible by running:

sudo apt update

sudo apt install -y ansible

1. To verify the installation

ansible --version

1. **Set Up the Directory Structure for Ansible:** Create a directory for your Ansible software

mkdir -p ~/ansible/deploy\_flask\_app

cd ~/ansible/deploy\_flask\_app

1. **Upload Application Files to GitHub:** Upload all your Flask application files to a GitHub repository. You will need this repository URL for the Ansible playbook.
2. **Create Ansible Playbook:** Inside the deploy\_flask\_app directory, create a playbook named deploy.yml. This file will contain tasks for deploying your Flask app.

nano deploy.yml

**Update the Ansible playbook content to suit your environment and paste it into the file-**

name: Deploy simple flask application

hosts: localhost

become: true

tasks:

- name: Update the apt package cache

apt:

update\_cache: yes

- name: Install required packages

apt:

name:

- python3

- python3-venv

- git

state: present

- name: Clone Flask app repository from GitHub

git:

repo: 'https://github.com/your\_username/your\_repo.git'

dest: '/home/kali/flask\_app/deploy'

- name: Create a Python virtual environment

command: python3 -m venv /home/kali/flask\_app/venv

args:

creates: /home/kali/flask\_app/venv

- name: Install Python dependencies

command: /home/kali/flask\_app/venv/bin/pip install -r /home/kali/flask\_app/requirements.txt

args:

chdir: /home/kali/flask\_app

- name: Start the Flask application

shell: |

nohup /home/kali/flask\_app/venv/bin/python -m flask run --host=0.0.0.0 --port=5000 &

args:

chdir: /home/kali/flask\_app

1. **Run the Ansible Playbook:** Run the following command to execute the playbook and deploy the application:

ansible-playbook -i inventory deploy.yml

**Summary of What the Ansible Playbook Does**

* **Updates** the system and installs the required packages (Python, Git, etc.).
* **Clones** the Flask application from GitHub.
* **Creates** a Python virtual environment and installs the dependencies listed in requirements.txt.
* **Runs** the Flask app in the background on port 5000.

**Conclusion**

By completing this assignment, you will gain hands-on experience with core DevOps principles such as Dockerization, monitoring, and continuous deployment. This foundational understanding will prepare you for more advanced infrastructure and application management tasks in the future.