**Final Project Report: CI/CD Pipeline with Kubernetes Deployment**

**Project Flow Overview**

This project automates the lifecycle of a Flask app using Docker, Jenkins, Docker Hub, and Kubernetes. Each stage serves a real world DevOps need: from source code management to containerization, continuous integration, and production-grade deployment.

**Flow Summary:**

* **GitHub**: Version control system for our source code
* **Docker**: Containerizes our app to make it portable and platform-independent
* **Docker Hub**: Stores our ready-to-run container image publicly
* **Jenkins**: Automates build + push steps every time the code changes
* **Kubernetes**: Deploys the app at scale with networking and container orchestration

**PHASE 1: GitHub Setup**

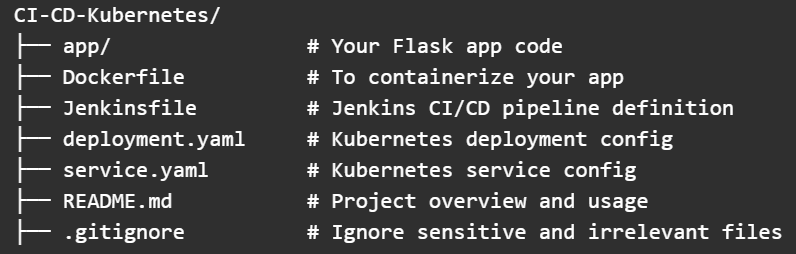
**Why We Created MSTIP-DevOps Repo**

* Version control for DevOps project
* Organized structure for internships and job showcase
* Centralized repo for Jenkins to pull code from

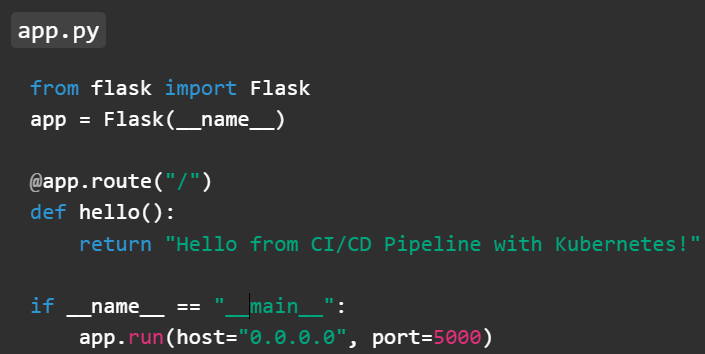
**Why We Used .gitignore**

* Prevents uploading junk files like:
  + \_\_pycache\_\_ and .pyc
  + IDE configs like .vscode/, .idea/
  + Environment files like .env
* Keeps repo clean and secure

**PHASE 2: Project Folder Structure**



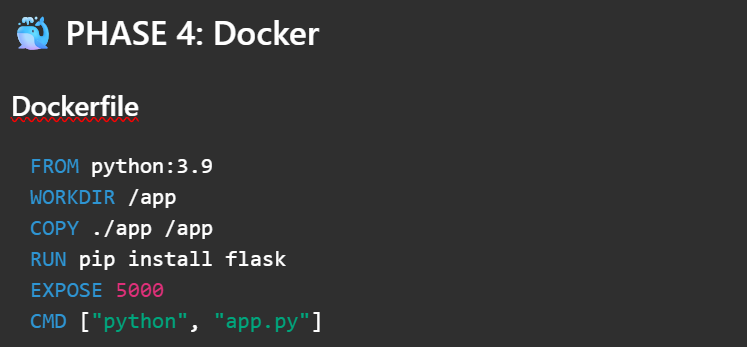
**PHASE 3: Flask App**



host="0.0.0.0": Makes the app accessible from outside Docker container

port=5000: Standard Flask port; used consistently in Docker + K8s

**PHASE 4: Docker**



**FROM:** Uses Python base image

**WORKDIR:** Default working directory in container

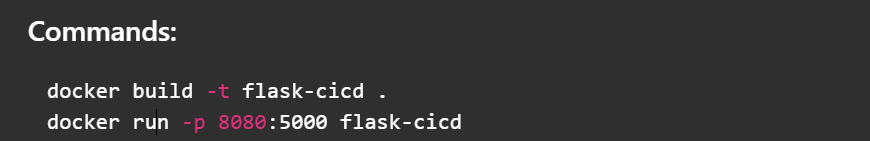
**COPY:** Adds Flask code to the image

**RUN:** Installs Flask

**EXPOSE:** Informs Docker that app uses port 5000

**CMD:** Runs app when container starts

**PHASE 5: Build & Test Locally with Docker**



Sometimes port 5000 is blocked. 8080 is more reliably accessible on most systems.

**TEST:** Visit http://localhost:8080

**The app should print “Hello from CI/CD Pipeline with Kubernetes!”**

**PHASE 6: Push to Docker Hub**

**docker tag flask-<docker-id>**

**docker login**

**docker push <docker-id>**

**Why Docker Hub?**

* Makes your image globally accessible
* Required for Jenkins + Kubernetes deployment

Image tags and SHA checksums are **safe to share**

.env, tokens, credentials should **never be committed**

**PHASE 7: Jenkins Setup & CI/CD Pipeline**

**Installed Jenkins via Docker:**

* Accessed via: <http://localhost:8081>
* Installed Suggested Plugins
* Created Admin User

**Setup Credentials:**

* Jenkins → Manage Jenkins → Credentials → Global
* Type: Username/Password
* ID: docker-hub-creds
* Used inside Jenkinsfile

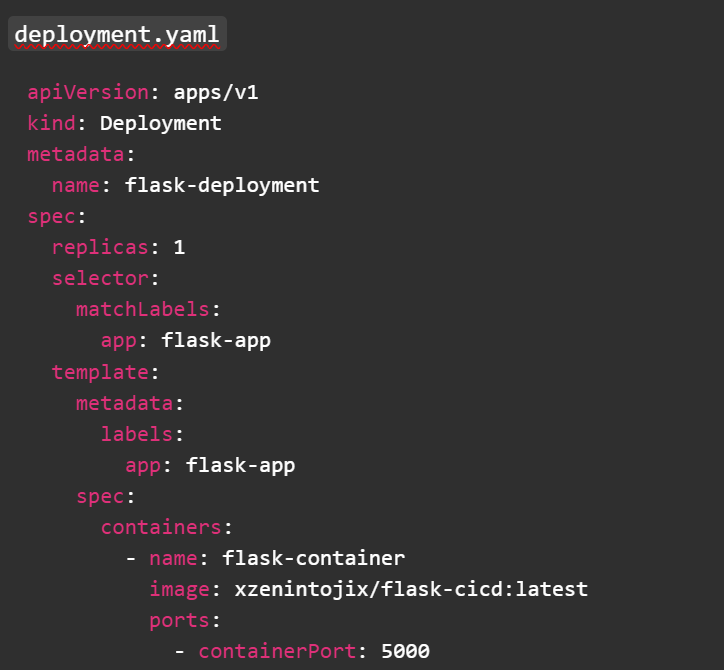


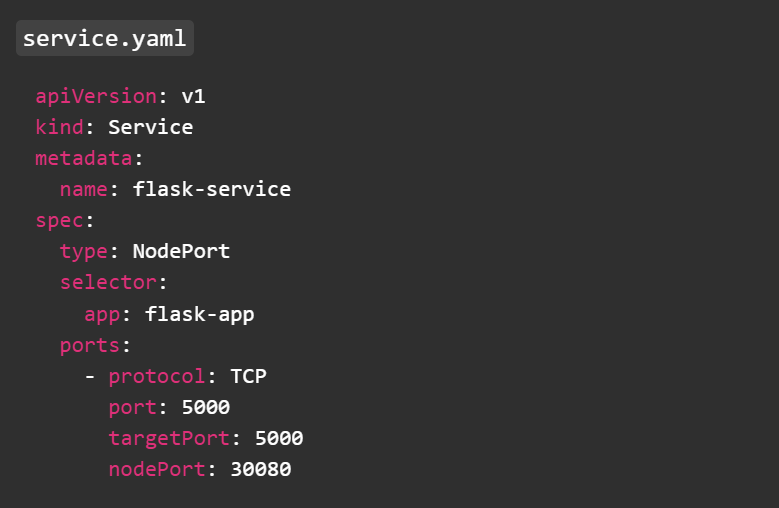
This automated our CI/CD pipeline — Jenkins now builds and pushes our image.

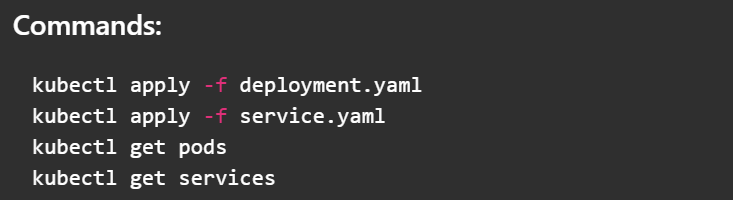
**PHASE 8: Kubernetes Deployment**

**Kubernetes Enabled via Docker Desktop:**

* Docker Desktop → Settings → Kubernetes → ✅ Enable Kubernetes
* Status: Running







**Visit:** <http://localhost:30080>

You’ll see: Hello from CI/CD Pipeline with Kubernetes!

**Common Questions Answered**

**What is the difference between GitHub and Docker Hub?**

* GitHub stores **source code**
* Docker Hub stores **built images** (ready to run)

**Why Docker if I already have Python installed?**

* Docker ensures consistency and portability across any system

**Why push image to Docker Hub?**

* Jenkins and Kubernetes need a **remote image** to pull

**What is the purpose of each stage?**

|  |  |
| --- | --- |
| **Stage** | **Purpose** |
| Docker | Packages app + environment |
| Jenkins | Automates build and push steps |
| Docker Hub | Public image registry |
| Kubernetes | Production-grade deployment & scaling |

**Is my Docker image private?**

* Your current image is public unless you configure Docker Hub for private repositories

**Final Outcome!**

You now have a fully functional DevOps pipeline:

