# **Flask API CI/CD Pipeline Documentation**

## **Introduction**

This document describes the setup and execution of a CI/CD pipeline for a Flask API. The pipeline automates the process of building, testing, deploying, and monitoring the Flask application. It leverages Jenkins as the CI/CD tool, Docker for containerization, and Kubernetes for orchestration and deployment.

## **Prerequisites**

1. **Jenkins**: Jenkins should be installed and configured with Docker and Kubernetes plugins.
2. **Docker Hub Account**: An account on Docker Hub for pushing Docker images.
3. **Kubernetes Cluster**: A running Kubernetes cluster (local or cloud-based).
4. **Git Repository**: A Git repository that contains the Flask API codebase, including a Dockerfile and a Jenkinsfile.
5. **Access to a Linux-based server**: For running the Jenkins pipeline, Docker, and Kubernetes commands.

## **Tools and Technologies**

* **Jenkins**: Automates the CI/CD pipeline.
* **Docker**: Containerizes the Flask API for consistent deployment.
* **Docker Hub**: Hosts Docker images.
* **Kubernetes**: Manages containerized applications across a cluster of machines.
* **Python (Flask)**: Web framework used for creating the API.
* **Git**: Version control system for managing codebase.

### **Pipeline Overview**

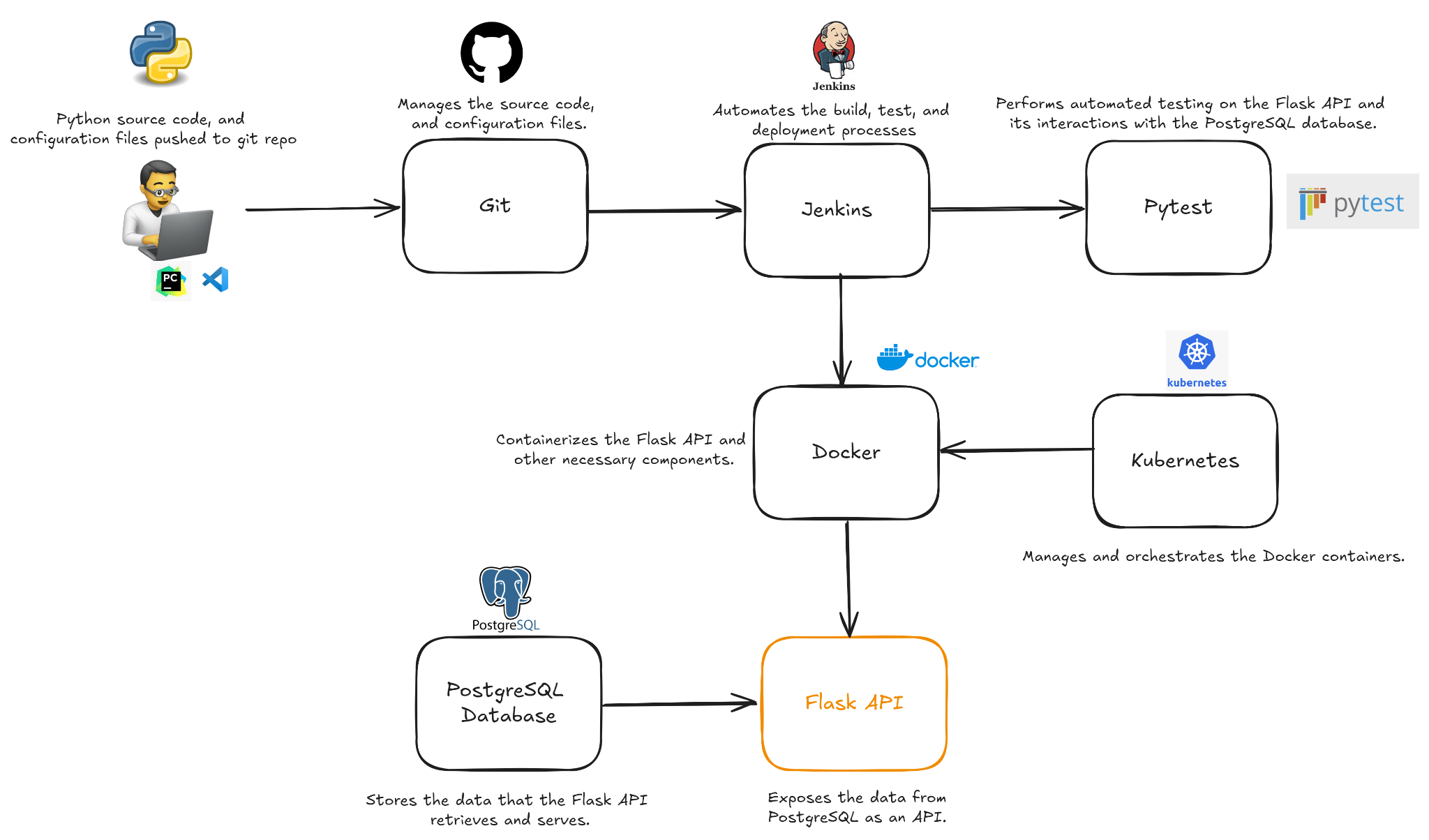


Figure 1 High-level Architecture of the Flask API

The CI/CD pipeline for the Flask API is a comprehensive automation setup that includes the following stages:

1. **Source Code Management:**
   * **Python Source Code:** The development of the Flask API and associated components are done in Python. Developers use tools like Visual Studio Code or PyCharm for code writing.
   * **Git:** The source code and configuration files are managed and version-controlled using Git, ensuring consistent code quality and collaboration.
2. **Continuous Integration and Continuous Deployment (CI/CD):**
   * **Jenkins:** Jenkins automates the build, test, and deployment processes. It pulls the code from the Git repository, triggers tests, and proceeds with the containerization of the application.
   * **Pytest:** Automated tests are executed using Pytest to verify the functionality of the Flask API and its integration with the PostgreSQL database.
3. **Containerization:**
   * **Docker:** Jenkins leverages Docker to containerize the Flask API along with other necessary components. The containerization process ensures consistency across different deployment environments.
4. **Container Orchestration:**
   * **Kubernetes:** Docker containers are managed and orchestrated by Kubernetes, ensuring that the Flask API is efficiently deployed and scaled. Kubernetes provides the necessary environment to manage the Docker containers in production.
5. **Database Integration:**
   * **PostgreSQL Database:** The PostgreSQL database stores the data retrieved and served by the Flask API. The API interacts with the database to fetch and update data, which is then exposed via the API endpoints.

This pipeline ensures that every change to the source code is automatically tested, containerized, and deployed, streamlining the entire software development lifecycle.

## **Jenkinsfile Configuration**

The Jenkinsfile is used to define the CI/CD pipeline in Jenkins. Below is an explanation of the stages:

| pipeline {  agent any  environment {  VENV\_DIR = 'venv'  DOCKER\_IMAGE = 'muyiwao/flask-api:latest'  FLASK\_APP\_PORT = '5310'  SERVER\_IP = '18.132.73.146' // Replace with your server's public IP  }  stages {  stage('Clone Repository') {  steps {  git url: 'https://github.com/muyiwao/APIPython.git', branch: 'main'  }  }  stage('Set Up Virtual Environment') {  steps {  script {  // Create a virtual environment  sh 'python3 -m venv ${VENV\_DIR}'  }  }  }  stage('Install Dependencies') {  steps {  script {  // Activate virtual environment, upgrade pip, and install dependencies  sh '''  source ${VENV\_DIR}/bin/activate  pip install --upgrade pip  pip install -r requirements.txt  '''  }  }  }  stage('Run Tests') {  steps {  script {  // Activate virtual environment and run pytest  sh '''  source ${VENV\_DIR}/bin/activate  pytest test\_app.py --junitxml=test-results.xml  '''  }  }  }  stage('Build Docker Image') {  steps {  script {  sh 'docker build -t ${DOCKER\_IMAGE} .'  }  }  }  stage('Push Docker Image to Docker Hub') {  steps {  script {  // Log in to Docker Hub  withCredentials([usernamePassword(credentialsId: 'muyiwa-hub', usernameVariable: 'DOCKER\_HUB\_USERNAME', passwordVariable: 'DOCKER\_HUB\_PASSWORD')]) {  sh 'echo ${DOCKER\_HUB\_PASSWORD} | docker login -u ${DOCKER\_HUB\_USERNAME} --password-stdin'  }  // Push the image  sh 'docker push ${DOCKER\_IMAGE}'  }  }  }  stage('Verify Deployment Files') {  steps {  script {  // Verify that the deployment files exist  sh 'ls -al k8s/'  }  }  }  stage('Deploy to Kubernetes') {  steps {  script {  // Apply the Kubernetes deployment and service files  sh '''  kubectl apply -f k8s/deployment.yaml  kubectl apply -f k8s/service.yaml  '''  }  }  }  }  post {  success {  // Output the full URL to access the Flask API  echo "Build succeeded. The Flask API is running at http://${SERVER\_IP}:${FLASK\_APP\_PORT}/data"  }  } } |
| --- |

### **Key Stages**

1. **Clone Repository**:
   * Jenkins clones the repository from GitHub.
   * Ensures that Jenkins has the latest version of the codebase.
2. **Set Up Virtual Environment**:
   * A Python virtual environment is created to isolate dependencies.
   * This ensures that package installations do not affect the global environment.
3. **Install Dependencies**:
   * Installs the required Python packages using pip.
   * The requirements.txt file specifies the necessary dependencies.
4. **Build Docker Image**:
   * A Docker image of the Flask API is built using the Dockerfile.
   * This image encapsulates the application and its environment.
5. **Push Docker Image to Docker Hub**:
   * The Docker image is pushed to Docker Hub for storage.
   * Jenkins uses stored credentials to authenticate with Docker Hub.
6. **Deploy to Kubernetes**:
   * The Docker container is deployed to a Kubernetes cluster.
   * Kubernetes YAML files (deployment.yaml and service.yaml) define the deployment and service.
7. **Post-Deployment**:
   * Cleans up the Docker images locally to save space.
   * Displays the URL of the running Flask API.

## **Kubernetes Deployment**

The Flask API is deployed to Kubernetes using the following YAML files:

### **deployment.yaml**

Defines the deployment configuration, including the Docker image and environment variables.

| apiVersion: apps/v1 kind: Deployment metadata:  name: flask-api-deployment  namespace: default # Ensure the correct namespace is specified or default if no specific namespace is required  labels:  app: flask-api spec:  replicas: 2  selector:  matchLabels:  app: flask-api  template:  metadata:  labels:  app: flask-api  spec:  containers:  - name: flask-api  image: muyiwao/flask-api:latest  ports:  - containerPort: 5310  env:  - name: ENDPOINT  value: "${env.ENDPOINT}"  - name: USER  value: "${env.USER}"  - name: PASSWORD  value: "${env.PASSWORD}"  - name: PORT  value: "${env.PORT}"  - name: DATABASE  value: "${env.DATABASE}" |
| --- |

### **service.yaml**

Defines the service that exposes the Flask API to external traffic.

| apiVersion: v1 kind: Service metadata:  name: flask-api-service  labels:  app: flask-api spec:  selector:  app: flask-api  ports:  - protocol: TCP  port: 80 # The port the service will expose  targetPort: 5310 # The port the Flask API is running on inside the container  type: LoadBalancer # Use LoadBalancer for external access to the service |
| --- |

## **Accessing the Flask API**

Once deployed, the Flask API can be accessed via the external IP address of the Kubernetes service. The service is configured to expose port 5310 to external traffic.

You can retrieve the external IP using the following command:

kubectl get svc flask-api-service

Access the API at:

http://<external-ip>:5310/data

## **Monitoring and Logging**

* **Kubernetes Dashboard**: Can be used for monitoring the deployed services and pods.
* **Jenkins Console**: Provides detailed logs of each stage in the pipeline.
* **kubectl logs**: Retrieves logs from the running containers.

## **Troubleshooting**

* **Pipeline Failures**: Check the Jenkins console output for any errors during the build and deployment stages.
* **Kubernetes Deployment Issues**: Verify the deployment and service YAML files for any misconfigurations.
* **Docker Image Issues**: Ensure that the Docker image builds correctly and is pushed to Docker Hub without errors.

## **Conclusion**

This CI/CD pipeline automates the end-to-end process of building, testing, and deploying a Flask API. It ensures that the API is consistently deployed across different environments with minimal manual intervention. By leveraging Jenkins, Docker, and Kubernetes, the pipeline provides scalability, reliability, and ease of management for modern web applications.