# End-to-End CI/CD Automation with Scalable Kubernetes Deployment

5 min read · Just now

Soujitd

In today's fast-paced software landscape, delivering features quickly without sacrificing reliability or security is a top priority. This blog demonstrates how to build a **complete CI/CD pipeline** for a sample Python Flask application, combining automation, security, and scalability — all on a local Minikube cluster.

## Why This Matters ??

Manual deployments are error-prone and slow. As applications grow, teams need automated, consistent workflows to:

1. **Speed:** Build-test-deploy cycles shrink from hours to minutes, giving instant feedback on regressions.

2. **Consistency:** Dockerfiles and Kubernetes YAML in Git ensure identical setups across dev, CI, and production.

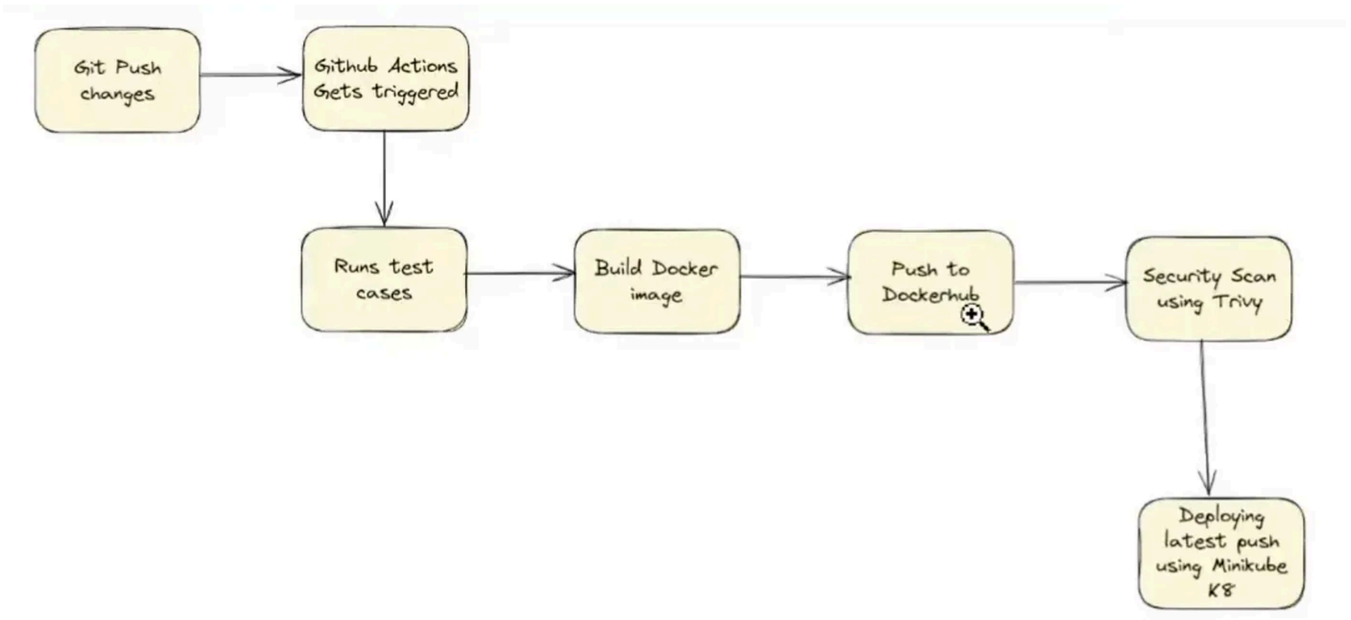
3. **Security:** Early Trivy scans catch vulnerabilities before they reach your registry or cluster.

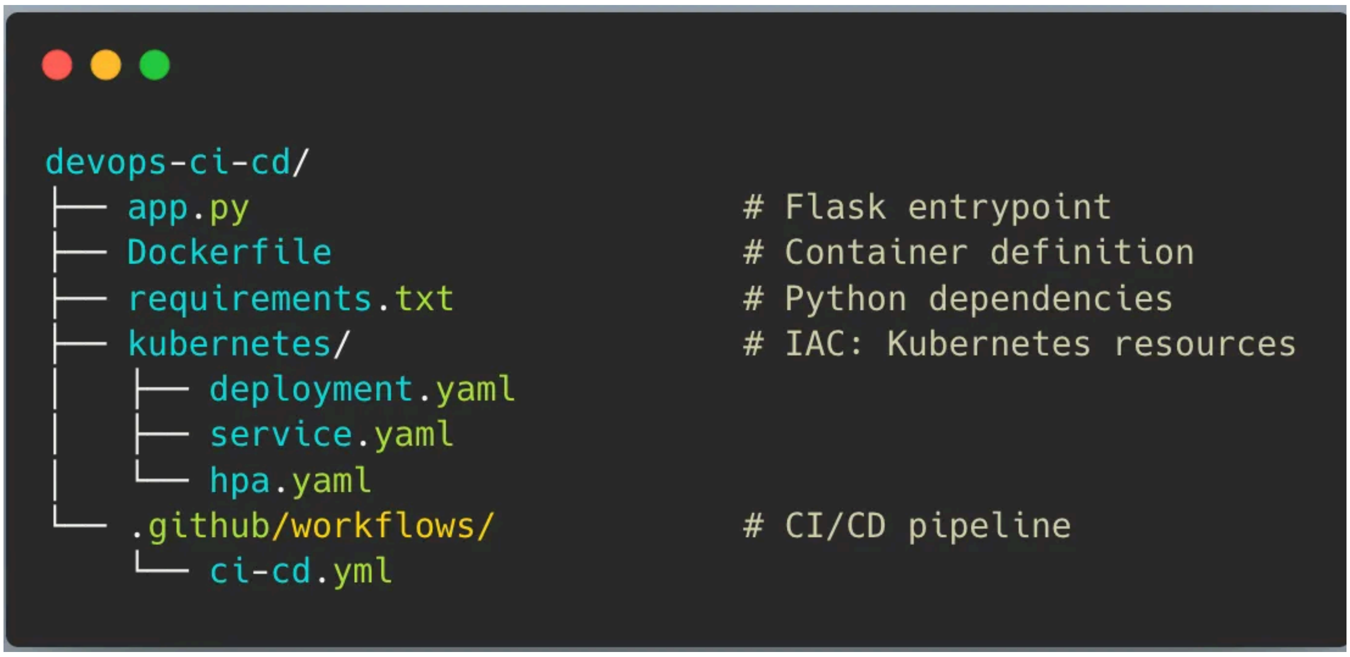
4. **Scalability:** Kubernetes HPA adapts replica counts to real-time load, ensuring performance and cost-efficiency.

## Tools used

- **Python + Flask** — Backend language and lightweight web framework (Flask) for building the application
- **Docker** — Containerization of the application for consistent environments across development and production
- **GitHub Actions** — Automates CI/CD pipeline steps like testing, vulnerability scanning, image building, and deployment
- **Trivy** — Scans Docker images for security vulnerabilities before pushing to the registry
- **Docker Hub** — Container image registry used to store and pull Docker images
- **Minikube** — Local Kubernetes cluster that simulates a real-world production deployment environment
- **kubectl** — Command-line tool for interacting with the Kubernetes cluster and managing deployments
- **YAML** — Declarative syntax used to define GitHub workflows and Kubernetes manifests
- **Git** — Version control system to track changes in source code and CI/CD configurations

## Architecture Overview & Project Structure


Git Push changes
Github Actions Gets triggered
Runs test cases
Build Docker image
Push to Dockerhub
Security Scan using Trivy
Deploying latest push using Minikube K8



```
devops-ci-cd/
├── app.py                          # Flask entrypoint
├── Dockerfile                      # Container definition
├── requirements.txt                # Python dependencies
├── kubernetes/                     # IAC: Kubernetes resources
│   ├── deployment.yaml
│   ├── service.yaml
│   └── hpa.yaml
└── .github/workflows/              # CI/CD pipeline
    └── ci-cd.yml
```

## Key Components

> *1. **Dockerfile:** Defines a minimal, reproducible build environment:*

```
# Use the official Python image as the base
FROM python:3.9

# Set the working directory
WORKDIR /app

# Copy the current directory contents into the container
COPY . .

# Install dependencies
RUN pip install -r requirements.txt

# Expose port 5000
EXPOSE 5000

# Command to run the application
CMD ["python", "app.py"]
```

> *2. **GitHub Actions Workflow** ( `ci-cd.yml` ): Automates the pipeline on every push to `main`*

```
name: CI/CD Pipeline

on:
  push:
    branches:
      - main  # Runs on any push to the main branch
  pull_request:
    branches:
      - main  # Runs on PR to the main branch

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Repository
        uses: actions/checkout@v4  # Checks out your code

      - name: Set Up Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.10"  # Ensure Python is installed

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt

      - name: Run Tests
        run: pytest tests/  # ✅ Runs tests before building the Docker image

      - name: Set Up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Log in to DockerHub
        env:
          DOCKER_USERNAME: ${{ secrets.DOCKER_USERNAME }}
          DOCKER_PASSWORD: ${{ secrets.DOCKER_PASSWORD }}
        run: echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --pas

      - name: Build Docker image
        run: docker build -t ${{ secrets.DOCKER_USERNAME }}/devops-ci-cd:latest

      - name: Push Docker image
        run: docker push ${{ secrets.DOCKER_USERNAME }}/devops-ci-cd:latest

  security_scan:
    runs-on: ubuntu-latest
    needs: build

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Install Trivy
        run: |
          sudo apt-get update
          sudo apt-get install -y curl
          curl -sfL https://raw.githubusercontent.com/aquasecurity/trivy/main/c

      - name: Run Trivy Vulnerability Scanner
        run: |
          trivy image --exit-code 1 --severity CRITICAL ${{ secrets.DOCKER_USER
```

> 3. ***Kubernetes Manifests***
>
> ***deployment.yaml:*** *Declares desired replicas, container image, and resource requests/limits.*

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: devops-ci-cd
  labels:
    app: devops-ci-cd
spec:
  replicas: 3  # Set replicas for scalability (you can change this)
  selector:
    matchLabels:
      app: devops-ci-cd
  template:
    metadata:
      labels:
        app: devops-ci-cd
    spec:
      containers:
      - name: devops-ci-cd
        image: dockersd12/devops-ci-cd:latest  # Replace with your Docker Hub
        imagePullPolicy: Always
        ports:
        - containerPort: 5000  # Flask default port
        resources:
          requests:
            memory: "128Mi"
            cpu: "250m"
          limits:
            memory: "256Mi"
            cpu: "500m"
        livenessProbe:   # Self-healing check
          httpGet:
            path: /
            port: 5000
          initialDelaySeconds: 5
          periodSeconds: 10
        readinessProbe:  # Ensures app is ready before traffic
```

```
          httpGet:
            path: /
            port: 5000
          initialDelaySeconds: 5
          periodSeconds: 5
```

> ***service.yaml:*** *Exposes the app on a NodePort for local access.*

```
apiVersion: v1
kind: Service
metadata:
  name: devops-ci-cd
spec:
  selector:
    app: devops-ci-cd  # This should match the labels in your deployment
  ports:
    - protocol: TCP
      port: 80    # Port inside the cluster
      targetPort: 5000  # The port your app is running on
  type: NodePort   # Expose the service as a NodePort
```

> ***hpa.yaml:*** *Configures HPA to scale between 1–3 pods at 50% CPU usage.*

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: devops-ci-cd-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: devops-ci-cd
  minReplicas: 1
  maxReplicas: 3
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
```

## Minikube Configuration setup

### *Install Minikube*

```
brew install minikube #For macOS
```

### *Start Minikube Cluster*

```
minikube start --driver=docker --memory=4096 --cpus=2
```

### *Exposing App locally*

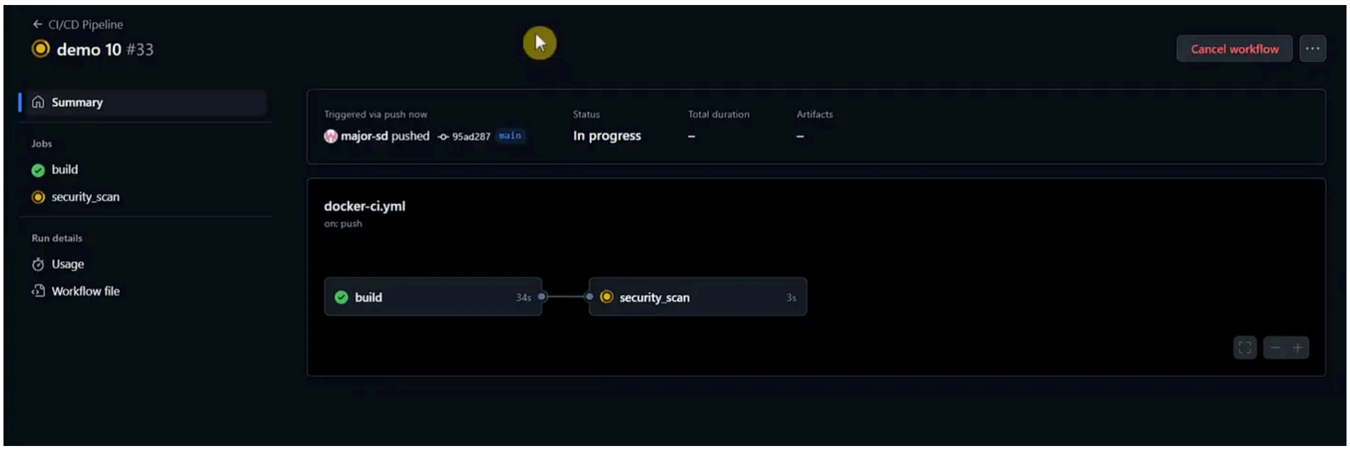
```
minikube service <<service-name>> --url
```

### *Visualise Minikube dashboard*

```
minikube dashboard
```

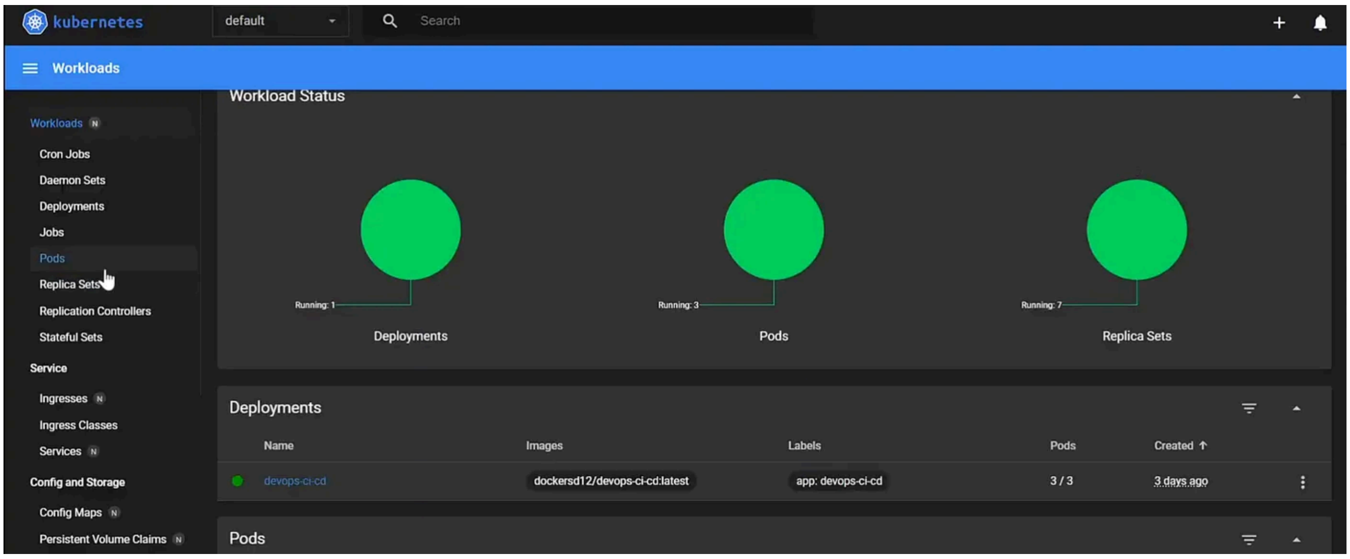
## Workflow Walkthrough

1. **Push Code:** Merging to `main` triggers GitHub Actions.
2. **Build & Scan:** Docker Buildx creates the image; Trivy scans for CVEs.
3. **Publish:** Securely authenticate and push the image to Docker Hub.
4. **Deploy:** `kubectl apply` reconciles the manifests on Minikube.
5. **Auto-Scale:** HPA observes CPU metrics and adjusts pod counts automatically.



| Triggered via push now | Status | Total duration | Artifacts |
|---|---|---|---|
| major-sd pushed 95ad287 main | In progress | – | – |

Triggered Pipeline on push



| Name | Images | Labels | Pods | Created ↑ |
|---|---|---|---|---|
| devops-ci-cd | dockersd12/devops-ci-cd:latest | app: devops-ci-cd | 3 / 3 | 3 days ago |

Minikube dashboard

## Real-World Best Practices

- **Immutable Builds:** Docker images ensure identical runs everywhere
- **Security-First:** Trivy catches vulnerabilities early, reducing risk
- **Infrastructure as Code:** Version-controlled manifests offer auditability and repeatability
- **Local-First Development:** Minikube simplifies experimentation without cloud costs

## Future Scopes

- **Helm Charts** for parameterized deployments
- **GitOps** with Flux or Argo CD for declarative Git-driven ops
- **Blue/Green & Canary Releases** for zero-downtime rollouts

- **Monitoring & Observability:** Prometheus, Grafana, and Loki
- **Secrets Management:** Vault or Kubernetes Secrets Encryption

This guide provides a complete blueprint for automating, securing, and scaling your deployments locally. Clone the repo, follow the steps, and transform your manual processes into a resilient CI/CD pipeline!

**Author:** Soujit Das
**Repo:** https://github.com/major-sd/devops-ci-cd
Medium Link: ***End-to-End CI/CD Automation with Scalable Kubernetes Deployment***

Deployment Minikube Trivy Docker Github Actions

S

Edit profile

## Written by Soujitd

0 followers · 1 following

## No responses yet

S Soujitd

What are your thoughts?