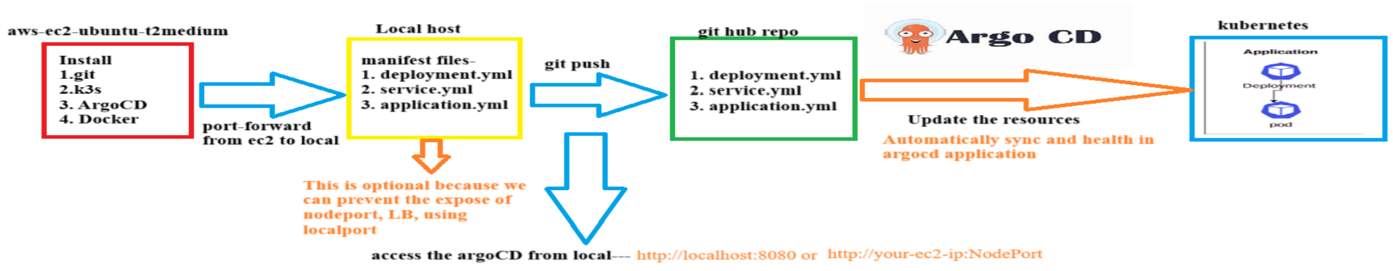
PROJECT-1 GitOps Workflow using ArgoCD on Kubernetes

Objective: Implement GitOps to automate Kubernetes deployments by syncing with a GitHub repository using ArgoCD.

Tools:

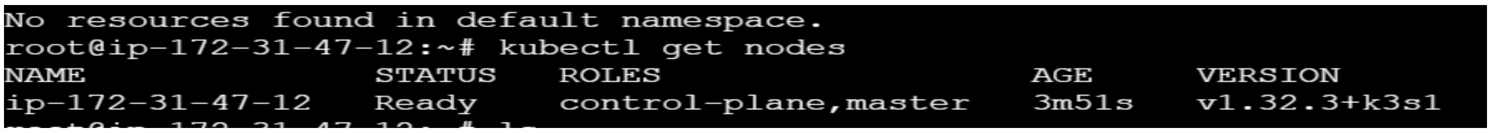
- EC2 (Ubuntu) ---- pick t2.medium instance
- K3s (Lightweight Kubernetes)
- ArgoCD
- GitHub
- Docker (for building and pushing images)
- MetalLB for K3s or Ingress controller (optional)

Architecture



Step-by-Step Guide

- **Set Up K3s (Kubernetes) on EC2 ----->** curl -sfL https://get.k3s.io | sh –
- **Check node status:** sudo kubectl get nodes
- **Set up kubectl for current user:**
  mkdir -p $HOME/.kube
  sudo cp /etc/rancher/k3s/k3s.yaml $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
- **Install ArgoCD in K3s**
  - kubectl create namespace argocd
  - kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
- **Expose the ArgoCD UI:**
  - nohup kubectl port-forward svc/argocd-server -n argocd 8080:443 > portforward.log 2>&1 &
  - ssh -i "C:\Path\To\YourKey.pem" -L 8080:localhost:8080 ubuntu@your-remote-ip # We don't want to expose of server NodePort address of server (optional)
  - kubectl get nodes



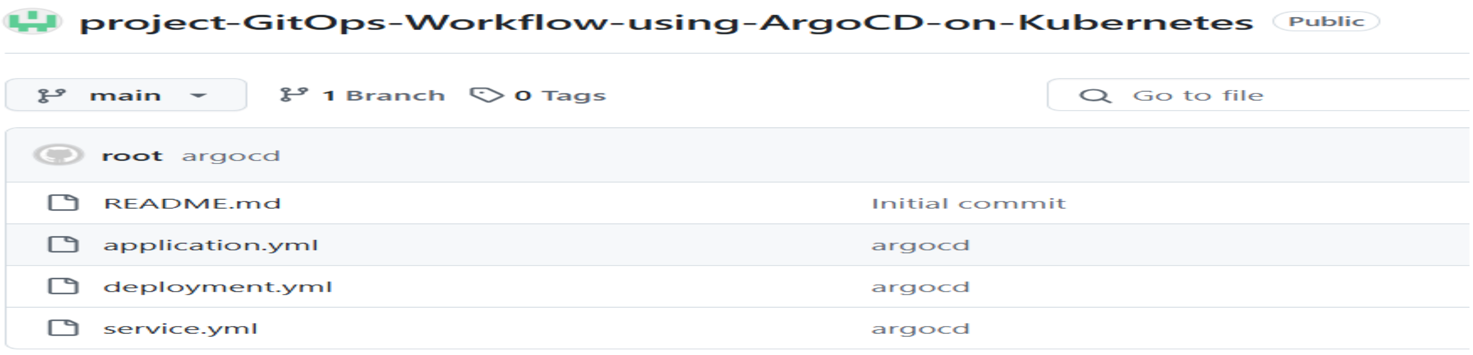
```
No resources found in default namespace.
root@ip-172-31-47-12:~# kubectl get nodes
NAME               STATUS   ROLES                  AGE     VERSION
ip-172-31-47-12    Ready    control-plane,master   3m51s   v1.32.3+k3s1
```

  - kubectl edit svc argocd-server -n argocd #change the clusterIP to nodeport
- **Access vi :** http://<ec2-user-ip>:8080
- **Get ArgoCD Admin Password:** kubectl get secret argocd-initial-admin-secret -n argocd -o jsonpath="{.data.password}" | base64 -d

Create Your GitHub Repo

- Push the following sample YAML files to a public GitHub repo:
  1. deployment.yaml
  2. service.yaml

- **Configure ArgoCD to Sync from Git:** application.yml # Autosync file
- **Apply it:** kubectl apply -f .
- **Verify Deployment:** kubectl get all
- Then push the files into the GitHub repo, check the argocd UI

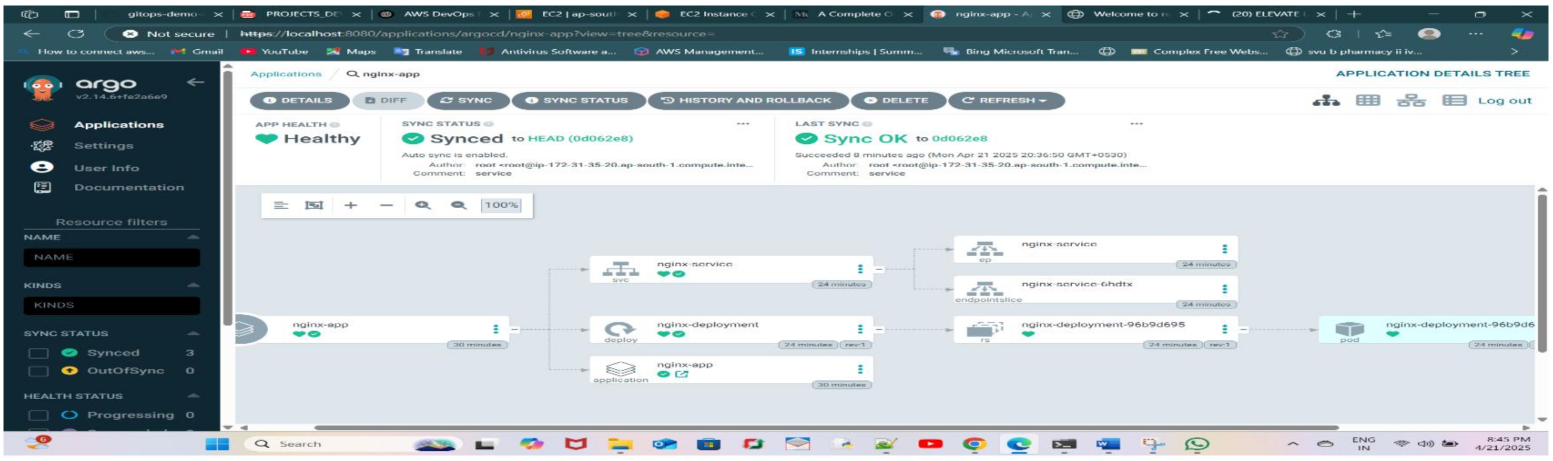


project-GitOps-Workflow-using-ArgoCD-on-Kubernetes Public

main · 1 Branch · 0 Tags · Go to file

root argocd	
README.md	Initial commit
application.yml	argocd
deployment.yml	argocd
service.yml	argocd

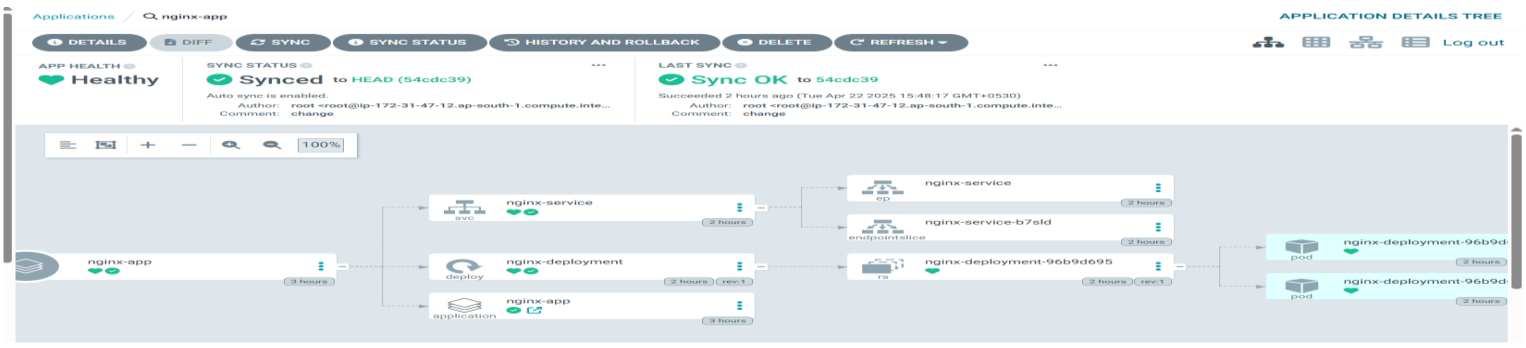
- Click refresh



APP HEALTH Healthy

SYNC STATUS Synced to HEAD (0d062e8)
Auto sync is enabled.
Author: root <root@ip-172-31-35-20.ap-south-1.compute.inte...
Comment: service

LAST SYNC Sync OK to 0d062e8
Succeeded 8 minutes ago (Mon Apr 21 2025 20:36:50 GMT+0530)
Author: root <root@ip-172-31-35-20.ap-south-1.compute.inte...
Comment: service

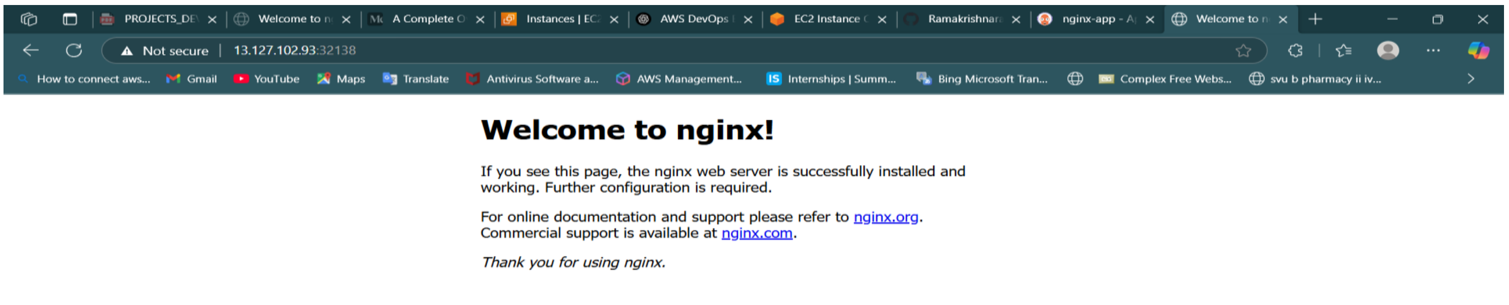


APP HEALTH Healthy

SYNC STATUS Synced to HEAD (54cdc39)
Auto sync is enabled.
Author: root <root@ip-172-31-47-12.ap-south-1.compute.inte...
Comment: change

LAST SYNC Sync OK to 54cdc39
Succeeded 2 hours ago (Tue Apr 22 2025 15:48:17 GMT+0530)
Author: root <root@ip-172-31-47-12.ap-south-1.compute.inte...
Comment: change

- Check the svc of ngnix server NodePort ----> kubectl get svc -n argocd
- Then access it from your local browser: http://your-ec2-ip:NodePort or http://localhost:NodePort



Not secure | 13.127.102.93:32138

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

- Check the EC2 port in security Group ----> if application is not accessing means check the server sg

GitHub Link: https://github.com/Ramakrishnaragi/project-GitOps-Workflow-using-ArgoCD-on-Kubernetes.git

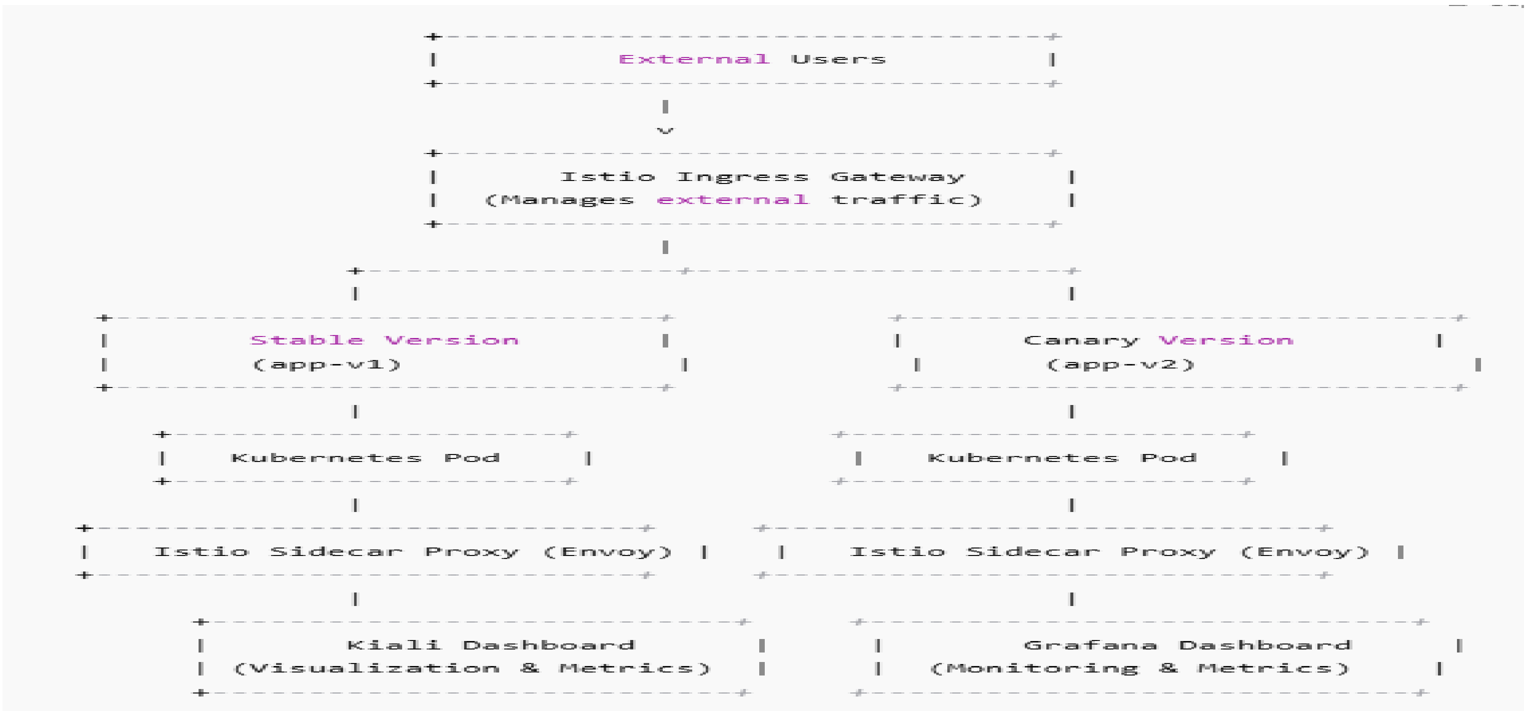
PROJECT-2 Kubernetes-Based Canary Deployment with K3s and Istio

Objective: Simulate modern canary deployments with traffic splitting between stable and new app versions.

Tools:

- K3s: Lightweight Kubernetes distribution
- Istio: Service mesh for traffic control and observability
- Docker: To containerize app
- Helm: (Optional) for managing Kubernetes apps
- App: Node.js or Python (2 versions)

Architecture



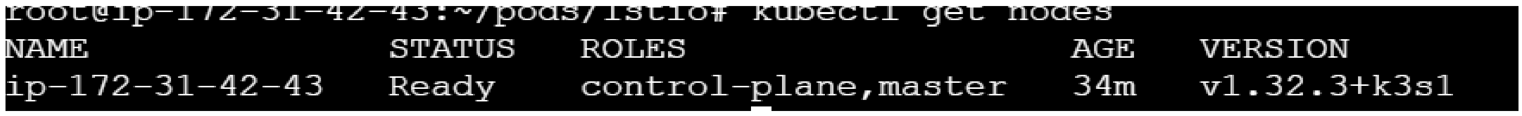
Step 1: Install K3s

curl -sfL https://get.k3s.io | sh -

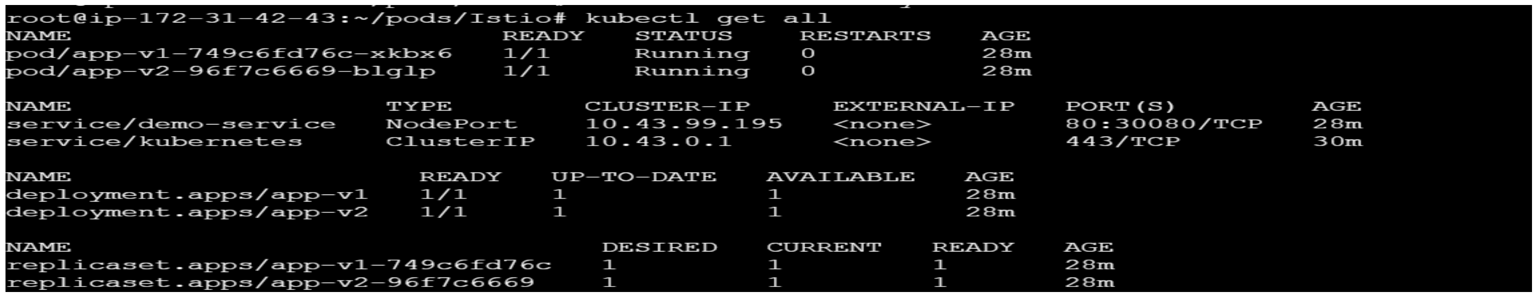
sudo chmod 644 /etc/rancher/k3s/k3s.yaml

export KUBECONFIG=/etc/rancher/k3s/k3s.yaml

kubectl cluster-info



```
root@ip-172-31-42-43:~/pods/Istio# kubectl get nodes
NAME               STATUS   ROLES                  AGE   VERSION
ip-172-31-42-43    Ready    control-plane,master   34m   v1.32.3+k3s1
```



```
root@ip-172-31-42-43:~/pods/Istio# kubectl get all
NAME                          READY   STATUS    RESTARTS   AGE
pod/app-v1-749c6fd76c-xkbx6   1/1     Running   0          28m
pod/app-v2-96f7c6669-blglp    1/1     Running   0          28m

NAME                     TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
service/demo-service     NodePort    10.43.99.195   <none>        80:30080/TCP   28m
service/kubernetes       ClusterIP   10.43.0.1      <none>        443/TCP        30m

NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/app-v1   1/1     1            1           28m
deployment.apps/app-v2   1/1     1            1           28m

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/app-v1-749c6fd76c   1         1         1       28m
replicaset.apps/app-v2-96f7c6669    1         1         1       28m
```

Step 2: Install Istio with Helm

helm repo add istio https://istio-release.storage.googleapis.com/charts

helm repo update

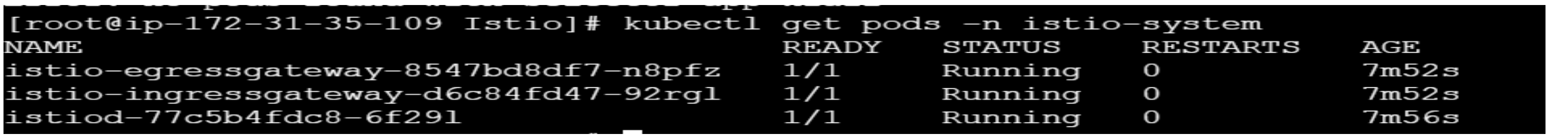
kubectl create namespace istio-system

helm install istio-base istio/base -n istio-system

helm install istiod istio/istiod -n istio-system --wait

helm install istio-ingress istio/gateway -n istio-system –wait

kubectl get pods -n istio-system # Verify Istio installation



```
[root@ip-172-31-35-109 Istio]# kubectl get pods -n istio-system
NAME                                       READY   STATUS    RESTARTS   AGE
istio-egressgateway-8547bd8df7-n8pfz       1/1     Running   0          7m52s
istio-ingressgateway-d6c84fd47-92rgl       1/1     Running   0          7m52s
istiod-77c5b4fdc8-6f291                    1/1     Running   0          7m56s
```

Step 3: Label Namespace for Istio Injection

kubectl create namespace app

kubectl label namespace app istio-injection=enabled

Step 4: Deploy Sample Application

Create a simple Node.js application with two versions (v1 and v2).

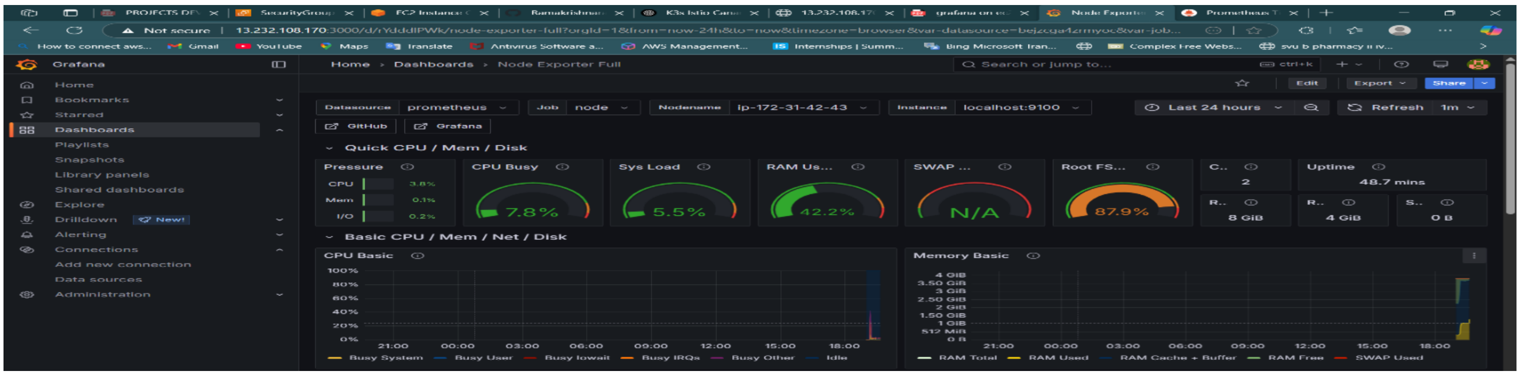
kubectl apply -f app-v1.yaml

kubectl apply -f app-v2.yaml

Step 5: Configure Istio Gateway and VirtualService

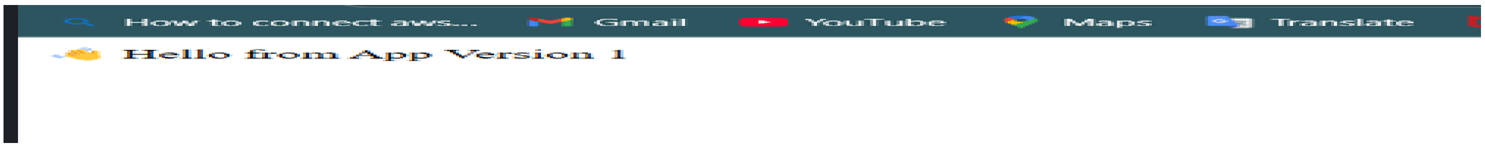
Vi gateway.yaml
Vi virtualservice.yaml
Vi destinationrule.yaml
Apply the Istio configurations:

kubectl apply -f gateway.yaml
kubectl apply -f destinationrule.yaml
kubectl apply -f virtualservice.yaml
Monitoring & Metrics: Istio Telemetry (Prometheus + Grafana pre-installed with demo profile)



Promotion/Rollback Strategy:

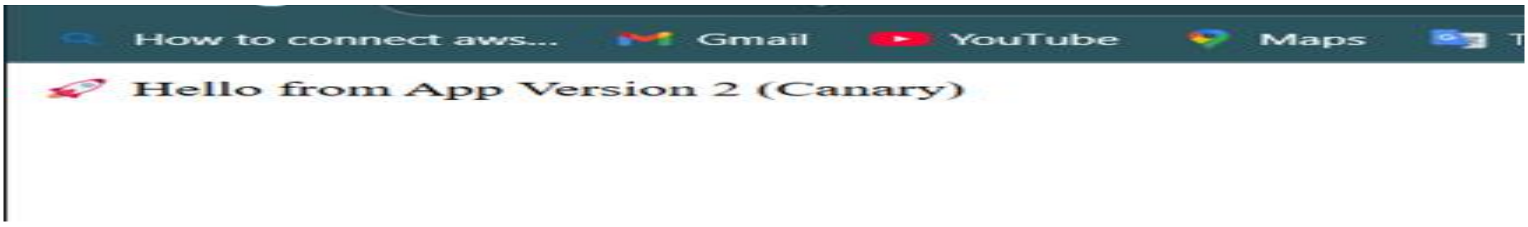
Promote v1:



How to connect aws... Gmail YouTube Maps Translate

Hello from App Version 1

Rollback: Reverse weights or scale down v2 deployment



How to connect aws... Gmail YouTube Maps

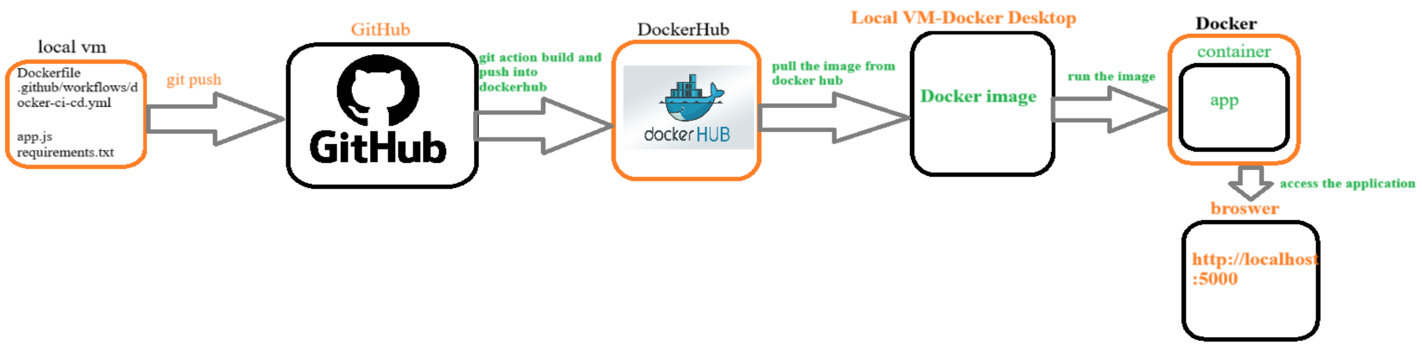
Hello from App Version 2 (Canary)

PROJECT-3 CI/CD Pipeline with GitHub Actions & Docker

Tools

- GitHub Action (CI/CD)
- Docker and DockerHub (Image build and push)
- Local VM or Minikube

Overview:



Step-by-Step Process:

1. **Create Your Application:**

- app.js (python-flask-app)

```
from flask import Flask
import os
app = Flask(__name__)
@app.route("/")
def hello():
    return "Flask sample application!!"
if __name__ == "__main__":
    port = int(os.environ.get("PORT", 5000))
    app.run(debug=True,host='0.0.0.0',port=port)
```

- **requirements.txt:**

```
flask
```

- **Dockerfile:**

```
FROM python:3.6
MAINTAINER veera "Ramakrishna"
COPY . /app
WORKDIR /app
RUN pip install -r requirements.txt
#ENTRYPOINT ["python"]
EXPOSE 5000
CMD ["python", "app.py"]
```

2. **Create GitHub Actions Workflow --- .github/workflows/docker-ci-cd.yml**

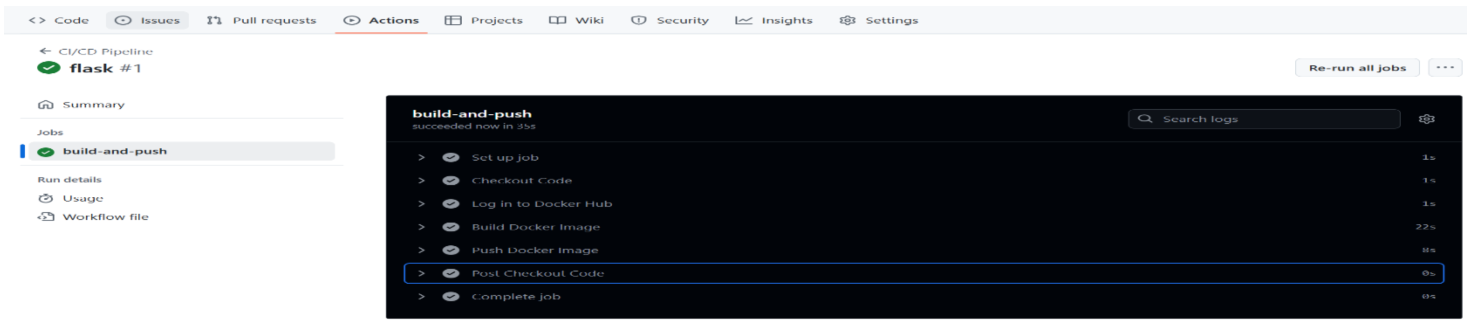
```
name: CI/CD Pipeline

on:

  push:

    branches: [main]
```

```
jobs:
  build-and-push:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout Code
      uses: actions/checkout@v3
    - name: Log in to Docker Hub
      run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u "${{ secrets.DOCKER_USERNAME }}" --password-stdin
    - name: Build Docker Image
      run: docker build -t ${{ secrets.DOCKER_USERNAME }}/myapp:latest .
    - name: Push Docker Image
      run: docker push ${{ secrets.DOCKER_USERNAME }}/myapp:latest
```

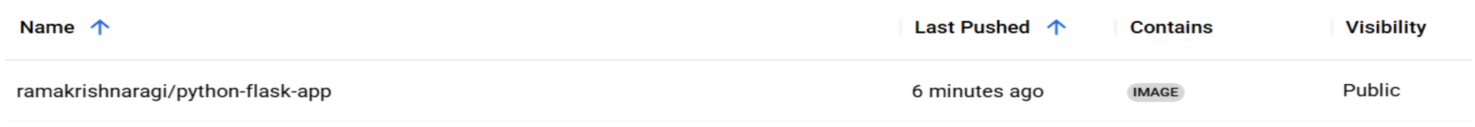
Set GitHub Secrets:

- DOCKER_USERNAME
- DOCKER_PASSWORD

3. **Push Code to GitHub Repo**

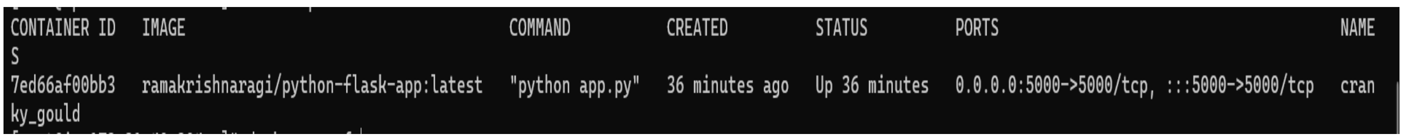


```
build-and-push
succeeded now in 35s
Set up job              1s
Checkout Code           1s
Log in to Docker Hub    1s
Build Docker Image      22s
Push Docker Image       8s
Post Checkout Code      0s
Complete job            0s
```

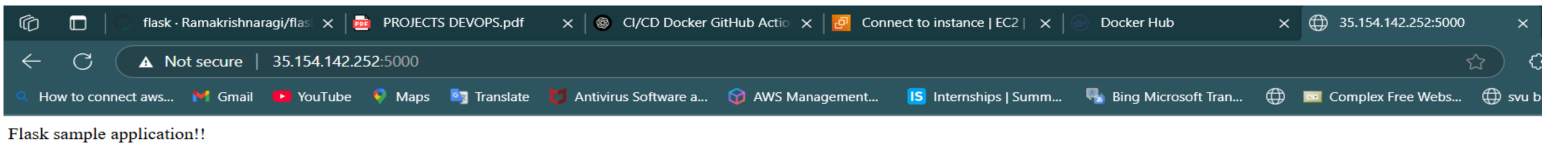


Name	Last Pushed	Contains	Visibility
ramakrishnaragi/python-flask-app	6 minutes ago	IMAGE	Public

4. **Pull & Run Docker Image:**
   - **docker pull your-docker-username/myapp:latest -----** https://hub.docker.com/repository/docker/ramakrishnaragi/python-flask-app/tags/latest/sha256:d7fe0f422c9e6e427f89f571beb7e6664a5d61bd721de223d86a943abbeed754 (docker hub link)
   - **docker run -p 5000:5000 your-docker-username/myapp:latest**



```
CONTAINER ID   IMAGE                                   COMMAND           CREATED          STATUS          PORTS                                           NAME
S
7ed66af00bb3   ramakrishnaragi/python-flask-app:latest "python app.py"   36 minutes ago   Up 36 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp     cran
ky_gould
```



Flask sample application!!