PROJECT DOCUMENTATION

1. TITLE :

Kubernetes-Based Canary Deployment for Tetris Gaming Application of two versions by using Node.js, Kubernetes(KOPS Cluster) and Istio .

2. INTRODUCTION :

Modern application development demands fast and reliable deployment strategies. Canary deployment is a release technique that reduces the risk of introducing a new software version by gradually shifting traffic from the old version to the new version. This project demonstrates a canary deployment model using kubernetes(KOPS Cluster) and Istio, a service mesh platform, to control traffic flow between different versions of a deployed application. The goal is to simulate real-world DevOps practices with traffic routing, performance monitoring, and rollbacks or promotions based on application behavior.

3. ABSTRACT :

This project simulates a canary deployment strategy on a Kubernetes(KOPS Cluster) environment. Here am creating KOPS Cluster because of high availability to my cluster two versions of a web application are deployed. And also creating a CI/CD pipeline for automate purpose .Using Istio, traffic is split between the stable (v1) and the canary (v2) versions — initially 80% to v1 and 20% to v2. Performance is monitored using Grafana/Prometheus dashboards. Based on performance, the deployment is either rolled back or the canary is promoted to full production. am collecting traffic logs and metrics dashoard .

4. TOOLS USED :

Git, GitHub, Jenkins, kubernetes(KOPS Cluster), Istio tool, Docker, Helm, DockerHub, prometheus & Grafana .

5. STEPS INVOLVED IN BUILDING THE PROJECT :

#STEP-1 : First of all taking a New Server with Server configurations are server_name : project-7 AMI : Amazon linux kernel 5.10, Instance_type : t2.medium, Key_Pair : sai-kp, Security_Group : All-Traffic, EBS : 25GIB . After connecting to Server with SSH .
#STEP-2 : Now To Set-Up the Tools Git, Docker, Jenkins, kubernetes(Kops Cluster), kubectl, Istio Tool, Helm, Promotheus & Grafana .
#STEP-3 : Now am creating a New Repository in GitHub and the Repository_Name is project-7 .
#STEP-4 : After am accessing the Jenkins dash board with public-ip:8080 . 8080 it is a port number . Now am creating a New Job for the pipeline running purpose . So Here am using CI/CD pipeline for some advanced stages because of automate purpose .
#Step-5 : Now am installed some plugins
PLUGIN_NAME : pipeline stage view , node.js, docker pipeline, eclipse temurin installer, sonarqube scanner, OWASP Dependency Check .
- Now writing and implementing CI/CD workflow with the stages .

stage-1 : It is a clean workspace .
stage-2 : To get the source code from github to ci server .
stage-3 : Here am using Sonarqube for the code quality analysis purpose because of to scan the source code .
stage-4 : Next am using quality gates because the purpose of code must pass before it can move forward in the build, test, or deployment process . if the code is incorrect then the pipeline is aborted .
stage-5 : To build the source code .
stage-6 : OWASP Dependency-Check into a CI/CD pipeline helps checks known vulnerabilities in your project dependencies.
stage-7 : To build the dockerfile .then a New docker-image has to come and also To Rename that docker-image .
stage-8 : So here am using Trivy Tool because to scan the New docker-image .
stage-9 : Last stage is new docker-image to push the docker-hub .

- #STEP-6 : After my CI/CD pipeline is successfully executed . Then a New Docker Image is created .

- COMMAND : docker build -t saikumar1817/project-7:v1 - To build the dockerfile first Version .
- COMMAND : docker build -t saikumar1817/project-7:v2 - To build the dockerfile . Second Version .
- COMMAND : docker push saikumar1817/project-7:v1 - To push the image to dockerhub .
- COMMAND : docker push saikumar1817/project-7:v2 - To push the image to dockerhub .

#STEP-7 : So Here am creating KOPS cluster because high availability for my cluster . Now to set-up a Kubernetes Kops Cluster in server by using ,

- awscli commands . (collecting the commands from via browser) .
- kubectl commands . (collecting the commands from via browser) .
- kops cluster commands . (collecting the commands from via browser) .

- To create a kops cluster creation by using command .

  - COMMAND : kops create cluster --name=saikumar.k8s.local --zones=us-east-1a,us-east-1b --master-size=t2.medium --master-count=1 --master-volume-size=30 –node-size=t2.medium –node-count=3 –node-volume-size=30 .
  - COMMAND : kops get cluster - To check the list of clusters .
  - Now KOPS cluster set-up is completed .

#STEP-8 : Now am installing Istio Tool with Helm .

```
 INSTALL HELM:
    curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
    chmod 700 get_helm.sh
   ./get_helm.sh
```

COMMAND : helm version

INSTALL ISTIO :

```
curl -L https://istio.io/downloadIstio | sh -
cd istio-1.26.2
export PATH=$PWD/bin:$PATH
istioctl install --set profile=demo
kubectl label namespace default istio-injection=enabled .
```

#STEP-9 : Now am writing two files one is deployment YAML file and another one is Service YAML file .

- Some Commands are there to Apply & Access .
- COMMAND : kubectl apply -f deployment .yaml - To Executed the deployment file .
- COMMAND : kubectl apply -f svc.yaml - To Executed the service file .

NOTE : am taking the screenshots of deployment and service YAML files and sending to my GitHub Repository into Deliverables folder .

#STEP-10 : Configure Istio Gateway , VirtualService , DestinationRule .

- Now am writing gateway.yaml , VirtualService.yaml, DestinationRule.yaml files .

NOTE : am taking the screenshots of gateway, virtualservice and destinationrule YAML files and sending to my GitHub Repository into

Deliverables folder .

#STEP-11 : Now Monitoring Prometheus & Grafana .

INSTALL K8S METRICS SERVER:

kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml

Verify that the metrics-server deployment is running the desired number of pods

- COMMAND : kubectl get pods -n kube-system
- COMMAND : kubectl get deployment metrics-server -n kube-system .

- INSTALL PROMETHEUS :

COMMAND : helm repo add prometheus-community https://prometheus-community.github.io/helm-charts .
COMMAND : helm repo update .
COMMAND : kubectl create namespace prometheus
COMMAND : helm install prometheus prometheus-community/prometheus --namespace prometheus --set alertmanager.persistentVolume.storageClass="gp2" --set server.persistentVolume.storageClass="gp2"
COMMAND : kubectl get all -n Prometheus

- INSTALL GRAFANA :

COMMAND : helm repo add grafana https://grafana.github.io/helm-charts .
COMMAND : helm repo update .
COMMAND : kubectl create namespace grafana .
COMMAND : helm install grafana grafana/grafana --namespace grafana --set persistence.storageClassName="gp2" --set persistence.enabled=true --set adminPassword=   --set service.type=LoadBalancer
COMMAND : kubectl get all -n Grafana .

#STEP-11 : Now am accessing Grafana dashboard for monitoring purpose .

-So here am Testing & Observing the Traffic . Refresh multiple times and observe which version serves the request.

#STEP-12 : Rollbacking the Versions by using commands .

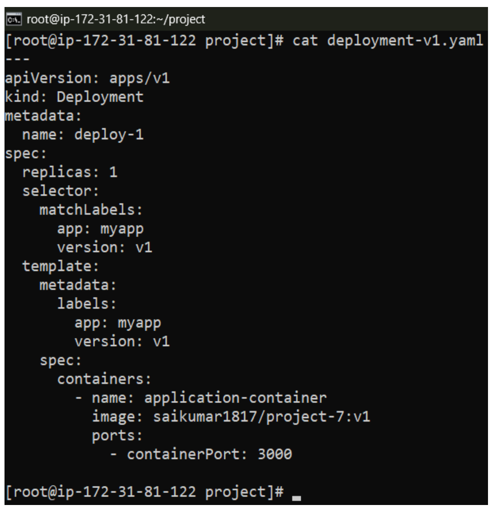
COMMAND : kubectl rollout undo deployment deployment_name –to-revision=3 .

COMMAND : kubectl rollout history deployment deployment_name .

7. CONCLUSION :

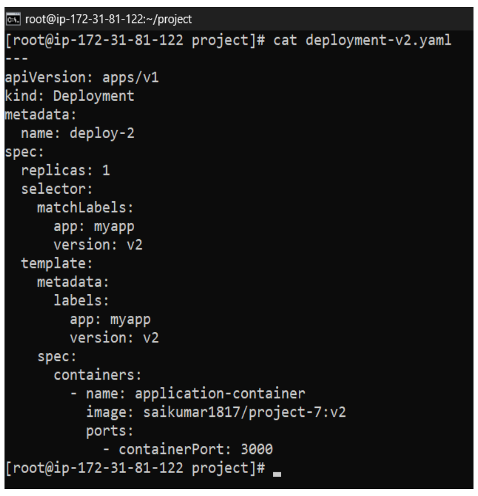
This project effectively demonstrates a canary deployment workflow in a controlled Kubernetes environment using Kops Cluster and Istio. By gradually introducing new app versions and monitoring their behavior and DevOps engineers can ensure safer, incremental releases.
Tools like Istio, Prometheus, and Grafana provide the necessary control and visibility to make data-driven deployment decisions.

NOTE : All screenshots am taken and sending to my GitHub Repository into Deliverables folder .



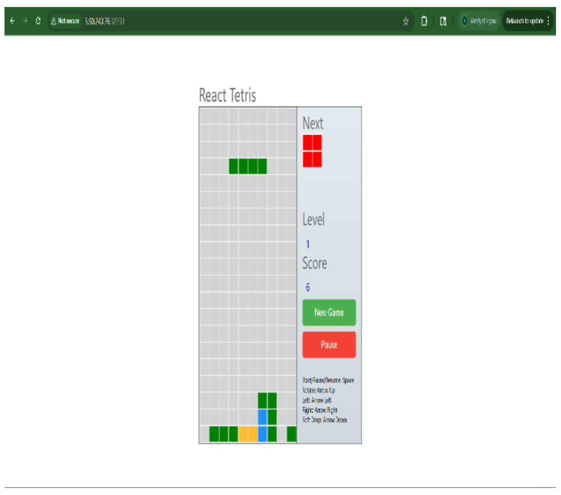
```
root@ip-172-31-81-122:~/project
[root@ip-172-31-81-122 project]# cat deployment-v1.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
      version: v1
  template:
    metadata:
      labels:
        app: myapp
        version: v1
    spec:
      containers:
        - name: application-container
          image: saikumar1817/project-7:v1
          ports:
            - containerPort: 3000

[root@ip-172-31-81-122 project]# _
```

This is the screenshot of deployment Version-V1 YAML file .

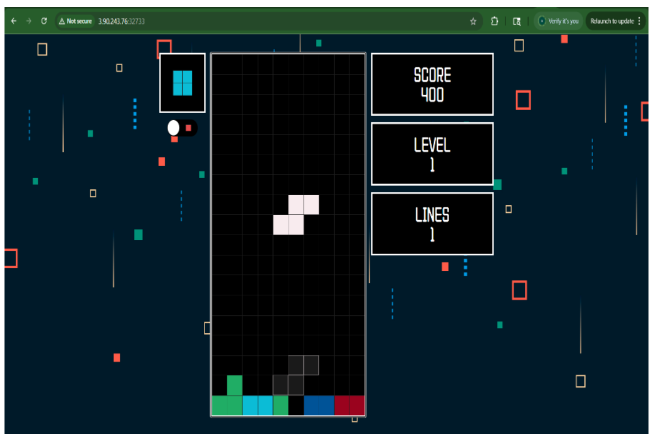


```
root@ip-172-31-81-122:~/project
[root@ip-172-31-81-122 project]# cat deployment-v2.yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
      version: v2
  template:
    metadata:
      labels:
        app: myapp
        version: v2
    spec:
      containers:
        - name: application-container
          image: saikumar1817/project-7:v2
          ports:
            - containerPort: 3000
[root@ip-172-31-81-122 project]# _
```

This is the screenshot of deployment Version-V2 YAML file .



This is the screenshot of Tetris gaming Application Version-V1 .



This is the screenshot of Tetris gaming Application Version-V2 .