Automated CI/CD Pipeline for Kubernetes Deployment with HPA and Monitoring

This project explains an end-to-end setup for automating CI/CD pipelines for Kubernetes deployments using Jenkins. The deployment will feature Horizontal Pod Autoscaling (HPA) and monitoring using Prometheus and Grafana. The application being deployed is based on the Docker image `learnitguide/busapp`.

Step 1: Setting Up Kubernetes Cluster

1.1. Create EC2 Instances

1. Launch two EC2 instances (1 master and 1 worker node) with the following specifications:
   - OS: Ubuntu 20.04
   - Instance type: t2.medium (minimum)
   - Ports to open: 22, 80, 3000, 9090, 9100
2. Update and install necessary dependencies on both nodes:

```
3. sudo apt update && sudo apt upgrade -y
4. sudo apt install -y docker.io curl apt-transport-https
5. sudo systemctl enable docker
6. sudo systemctl start docker
```

1.2. Install Kubernetes Components

Install kubeadm, kubelet, and kubectl on both nodes:

```
1. curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo
   apt-key add -
```

echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list sudo apt update sudo apt install -y kubelet kubeadm kubectl sudo systemctl enable kubelet

~~~
2. Disable swap:
```bash
sudo swapoff -a
sudo sed -i '/ swap / s/^/#/' /etc/fstab
~~~

```
   3. Initialize the Kubernetes cluster on the master node:
   4. sudo kubeadm init --pod-network-cidr=192.168.0.0/16
   5. Configure kubectl on the master node:
   6. mkdir -p $HOME/.kube
```

7. `sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config`
8. `sudo chown $(id -u):$(id -g) $HOME/.kube/config`
9. `Install a network plugin (Calico):`
10. `kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml`
11. `Join the worker node to the cluster (run the kubeadm join command provided during initialization on the worker node).`

(Note : You can also use EKS or any cloud provider for creating cluster)

Step 2: Deploying the Application

2.1. Create Deployment YAML File

`Create a busapp-deployment.yaml file:`

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busapp-deployment
  labels:
    app: busapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: busapp
  template:
    metadata:
      labels:
        app: busapp
    spec:
      containers:
      - name: busapp
        image: learnitguide/busapp
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: "100m"
            memory: "128Mi"
          limits:
            cpu: "500m"
            memory: "256Mi"
```

2.2. Create Service YAML File

`Create a busapp-service.yaml file:`

```yaml
apiVersion: v1
kind: Service
metadata:
  name: busapp-service
spec:
  selector:
    app: busapp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: LoadBalancer
```

2.3. Create Ingress YAML File

Create a busapp-ingress.yaml file:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: busapp-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: busapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: busapp-service
            port:
              number: 80
```

2.4. Apply YAML Files

Deploy the application, service, and ingress:

```
kubectl apply -f busapp-deployment.yaml
kubectl apply -f busapp-service.yaml
kubectl apply -f busapp-ingress.yaml
Verify the deployment:
kubectl get pods,svc,ingress
```

Step 3: Horizontal Pod Autoscaler

3.1. Create HPA YAML File

Create a busapp-hpa.yaml file:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
```

```yaml
metadata:
  name: busapp-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: busapp-deployment
  minReplicas: 1
  maxReplicas: 5
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

Apply the HPA manifest: `kubectl apply -f busapp-hpa.yaml`

Check the HPA: `kubectl get hpa`

Step 4: Monitoring with Prometheus and Grafana

4.1. Install Prometheus and Grafana

`Install Prometheus:`

1. `kubectl apply -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/main/bundle.yaml`
2. `Install Grafana:`
3. `helm repo add grafana https://grafana.github.io/helm-charts`
4. `helm repo update`
5. `helm install grafana grafana/grafana --namespace monitoring --create-namespace`

4.2. Access Grafana

`Retrieve Grafana admin password:`

1. `kubectl get secret --namespace monitoring grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo`
2. `Port-forward Grafana service:`
3. `kubectl port-forward svc/grafana -n monitoring 3000:80`
4. `Open Grafana in the browser: http://localhost:3000.`

4.3. Configure Prometheus in Grafana

1. `Add Prometheus as a data source with the URL http://prometheus.default.svc.cluster.local:9090.`
2. `Import Kubernetes dashboards (ID: 6417 or 315).`

Step 5: Jenkins CI/CD Pipeline

5.1. Install Jenkins

Deploy Jenkins using Helm:

1. `helm repo add jenkins https://charts.jenkins.io`
2. `helm repo update`
3. `helm install jenkins jenkins/jenkins --namespace cicd --create-namespace`
4. Access Jenkins:
   - Retrieve admin password:
   - `kubectl exec --namespace cicd -it svc/jenkins -c jenkins -- cat /run/secrets/chart-admin-password`
   - Port-forward Jenkins service:
   - `kubectl port-forward svc/jenkins -n cicd 8080:8080`
   - Open Jenkins: `http://localhost:8080`.

5.2. Configure Jenkins Pipeline

Create a new GitHub repository and push all YAML files and a `Jenkinsfile`:

```
pipeline {
    agent any
    stages {
        stage('Clone Repo') {
            steps {
                git 'https://github.com/busapp.git'
            }
        }
        stage('Deploy to Kubernetes') {
            steps {
                sh 'kubectl apply -f busapp-deployment.yaml'
                sh 'kubectl apply -f busapp-service.yaml'
                sh 'kubectl apply -f busapp-hpa.yaml'
                sh 'kubectl apply -f busapp-ingress.yaml'
            }
        }
        stage('Monitor Deployment') {
            steps {
                echo "Monitor deployment with Grafana and Prometheus."
```

Implemented a robust, scalable, and automated CI/CD pipeline for a Kubernetes application, including monitoring and autoscaling.

--