Project 01 - 1 Hour

Deploying a Scalable Web Application with Persistent Storage and Advanced Automation

Objective:

Deploy a scalable web application using Docker Swarm and Kubernetes, ensuring data persistence using a single shared volume, and automate the process using advanced shell scripting.

Overview:

1. **Step 1**: Set up Docker Swarm and create a service.
2. **Step 2**: Set up Kubernetes using Minikube.
3. **Step 3**: Deploy a web application using Docker Compose.
4. **Step 4**: Use a single shared volume across multiple containers.
5. **Step 5**: Automate the entire process using advanced shell scripting.

Step 1: Set up Docker Swarm and Create a Service

1.1 Initialize Docker Swarm

```
# Initialize Docker Swarm
docker swarm init
```

1.2 Create a Docker Swarm Service

```
# Create a simple Nginx service in Docker Swarm
docker service create --name nginx-service --publish 8080:80 nginx
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-5_Kubernetes$ docker service create --name nginx-service
--publish 8081:80 nginx
q4jpec0ir8unt8zvlywmiozbd
overall progress: 1 out of 1 tasks
1/1: running
verify: Service q4jpec0ir8unt8zvlywmiozbd converged
einfochips@AHMLPT1108:~/DevOPs_Training/Day-5_Kubernetes$ docker service ls
ID             NAME            MODE         REPLICAS   IMAGE          PORTS
q4jpec0ir8un   nginx-service   replicated   1/1        nginx:latest   *:8081->80/tcp
einfochips@AHMLPT1108:~/DevOPs_Training/Day-5_Kubernetes$
```

Step 2: Set up Kubernetes Using Minikube

2.1 Start Minikube

```
# Start Minikube
minikube start
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-5_Kubernetes$ minikube start
😄  minikube v1.32.0 on Ubuntu 20.04
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🔄  Restarting existing docker container for "minikube" ...
🎉  minikube 1.33.1 is available! Download it: https://github.com/kubernetes/minikube/releases/tag/v1
.33.1
💡  To disable this notice, run: 'minikube config set WantUpdateNotification false'

🐳  Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.9.4
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20231011-8b53cabe0
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20231011-8b53cabe0
🔎  Verifying ingress addon...
🌟  Enabled addons: storage-provisioner, default-storageclass, ingress
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
einfochips@AHMLPT1108:~/DevOPs_Training/Day-5_Kubernetes$
```

2.2 Deploy a Web App on Kubernetes

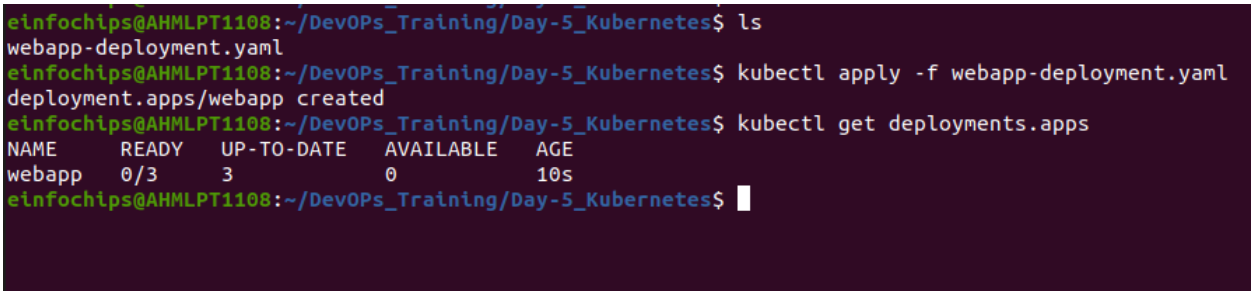
Create a deployment file named `webapp-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
```

```
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: webapp
        image: nginx
        ports:
        - containerPort: 80
```

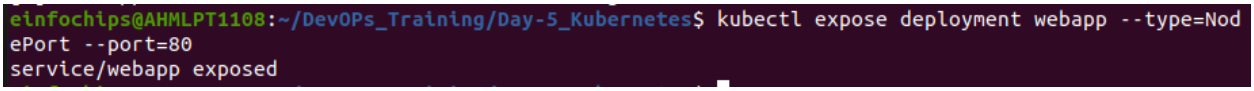
Apply the deployment:

```
kubectl apply -f webapp-deployment.yaml
```



2.3 Expose the Deployment

```
kubectl expose deployment webapp --type=NodePort --port=80
```



Step 3: Deploy a Web Application Using Docker Compose

3.1 Create a `docker-compose.yml` File

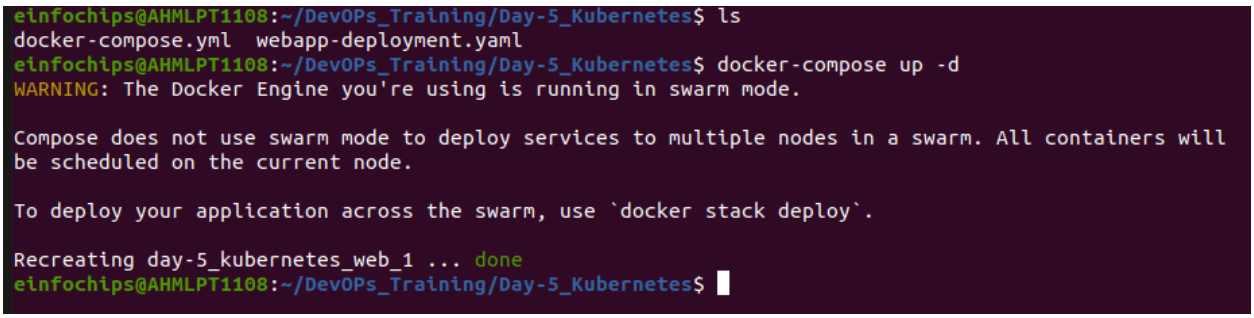
```
version: '3'
services:
```

```
  web:
    image: nginx
    ports:
      - "8080:80"
    volumes:
      - webdata:/usr/share/nginx/html

volumes:
  webdata:
```

3.2 Deploy the Web Application

```
# Deploy using Docker Compose
docker-compose up -d
```



Step 4: Use a Single Shared Volume Across Multiple Containers

4.1 Update `docker-compose.yml` to Use a Shared Volume

```
version: '3'
services:
  web1:
    image: nginx
    ports:
      - "8081:80"
    volumes:
      - shareddata:/usr/share/nginx/html
  web2:
```

```
    image: nginx
    ports:
      - "8082:80"
    volumes:
      - shareddata:/usr/share/nginx/html

volumes:
  shareddata:
```

4.2 Deploy with Docker Compose

```
# Deploy using Docker Compose
docker-compose up -d
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-5_Kubernetes/multiContainer-SharedVolume$ docker-compose up -d
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Creating network "multicontainer-sharedvolume_default" with the default driver
Creating volume "multicontainer-sharedvolume_shareddata" with default driver
Creating multicontainer-sharedvolume_web1_1 ... done
Creating multicontainer-sharedvolume_web2_1 ... done
einfochips@AHMLPT1108:~/DevOPs_Training/Day-5_Kubernetes/multiContainer-SharedVolume$ docker ps
CONTAINER ID   IMAGE                                  COMMAND                  CREATED          STATUS          PORTS
                                    NAMES
618238e20a7a   nginx                                  "/docker-entrypoint.…"   6 seconds ago    Up 5 seconds    0.0.0.0:8083->80/tcp, :::8083->80/tcp
                                    multicontainer-sharedvolume_web2_1
9fa06a654a6c   nginx                                  "/docker-entrypoint.…"   6 seconds ago    Up 5 seconds    0.0.0.0:8082->80/tcp, :::8082->80/tcp
                                    multicontainer-sharedvolume_web1_1
cce9cd59abca   nginx:latest                           "/docker-entrypoint.…"   35 minutes ago   Up 35 minutes   80/tcp
                                    nginx-service.1.i1uzjrq06u1303bzxeyzvt4t5
2d12b42d98cb   gcr.io/k8s-minikube/kicbase:v0.0.42   "/usr/local/bin/entr…"   8 weeks ago      Up 32 minutes   127.0.0.1:32772->22/tcp, 127.0.0.1:3277
9->8443/tcp, 127.0.0.1:32768->32443/tcp   minikube
einfochips@AHMLPT1108:~/DevOPs_Training/Day-5_Kubernetes/multiContainer-SharedVolume$
```

Step 5: Automate the Entire Process Using Advanced Shell Scripting

5.1 Create a Shell Script `deploy.sh`

```
#!/bin/bash

# Initialize Docker Swarm
docker swarm init

# Create Docker Swarm Service
docker service create --name nginx-service --publish 8080:80 nginx
```

```
# Start Minikube
minikube start

# Create Kubernetes Deployment
kubectl apply -f webapp-deployment.yaml

# Expose the Deployment
kubectl expose deployment webapp --type=NodePort --port=80

# Deploy Web App Using Docker Compose
docker-compose -f docker-compose-single-volume.yml up -d

echo "Deployment completed successfully!"
```

5.2 Make the Script Executable

```
# Make the script executable
chmod +x deploy.sh
```

5.3 Run the Script

```
# Run the deployment script
./deploy.sh
```

```
einfochips@AHMLPT1108:~/DevOPs_Training/Day-5_Kubernetes$ ./deploy.sh
Error response from daemon: This node is already part of a swarm. Use "docker swarm leave" to leave t
his swarm and join another one.
Error response from daemon: rpc error: code = InvalidArgument desc = port '8081' is already in use by
 service 'nginx-service' (ibg0erv4o4v022x4wrkktgeiu) as an ingress port
😄  minikube v1.32.0 on Ubuntu 20.04
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🏃  Updating the running docker "minikube" container ...
🐳  Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.9.4
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20231011-8b53cabe0
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20231011-8b53cabe0
🔎  Verifying ingress addon...
🌟  Enabled addons: storage-provisioner, default-storageclass, ingress
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
deployment.apps/webapp created
service/webapp exposed
WARNING: The Docker Engine you're using is running in swarm mode.

Compose does not use swarm mode to deploy services to multiple nodes in a swarm. All containers will
be scheduled on the current node.

To deploy your application across the swarm, use `docker stack deploy`.

Creating network "day-5_kubernetes_default" with the default driver
Creating day-5_kubernetes_web_1 ... done
Deployment completed successfully!
einfochips@AHMLPT1108:~/DevOPs_Training/Day-5_Kubernetes$
```

Project 02 - 1 Hour

Comprehensive Deployment of a Multi-Tier Application with CI/CD Pipeline

Objective:

Deploy a multi-tier application (frontend, backend, and database) using Docker Swarm and Kubernetes, ensuring data persistence using a single shared volume across multiple containers, and automating the entire process using advanced shell scripting and CI/CD pipelines.

Overview:

6. **Step 1**: Set up Docker Swarm and create a multi-tier service.
7. **Step 2**: Set up Kubernetes using Minikube.
8. **Step 3**: Deploy a multi-tier application using Docker Compose.
9. **Step 4**: Use a single shared volume across multiple containers.
10. **Step 5**: Automate the deployment process using advanced shell scripting.

Step 1: Set up Docker Swarm and Create a Multi-Tier Service

1.1 Initialize Docker Swarm

```
# Initialize Docker Swarm
docker swarm init
```

1.2 Create a Multi-Tier Docker Swarm Service

Create a `docker-compose-swarm.yml` file:

```
version: '3.7'
services:
  frontend:
    image: nginx
    ports:
      - "8080:80"
    deploy:
      replicas: 2
    volumes:
```

```
      - shareddata:/usr/share/nginx/html
  backend:
    image: mybackendimage
    ports:
      - "8081:80"
    deploy:
      replicas: 2
    volumes:
      - shareddata:/app/data
  db:
    image: postgres
    environment:
      POSTGRES_DB: mydb
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    deploy:
      replicas: 1
    volumes:
      - dbdata:/var/lib/postgresql/data


volumes:
  shareddata:
  dbdata:
```

Deploy the stack:

```
# Deploy the stack using Docker Swarm
docker stack deploy -c docker-compose-swarm.yml myapp
```

Step 2: Set up Kubernetes Using Minikube

2.1 Start Minikube

```
# Start Minikube
minikube start
```

2.2 Create Kubernetes Deployment Files

Create `frontend-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: shareddata
          mountPath: /usr/share/nginx/html
      volumes:
      - name: shareddata
        persistentVolumeClaim:
          claimName: shared-pvc
```

Create `backend-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
```

```
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: mybackendimage
        ports:
        - containerPort: 80
        volumeMounts:
        - name: shareddata
          mountPath: /app/data
      volumes:
      - name: shareddata
        persistentVolumeClaim:
          claimName: shared-pvc
```

Create `db-deployment.yaml`:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      labels:
        app: db
```

```
    spec:
      containers:
      - name: db
        image: postgres
        env:
        - name: POSTGRES_DB
          value: mydb
        - name: POSTGRES_USER
          value: user
        - name: POSTGRES_PASSWORD
          value: password
        volumeMounts:
        - name: dbdata
          mountPath: /var/lib/postgresql/data
      volumes:
      - name: dbdata
        persistentVolumeClaim:
          claimName: db-pvc
```

Create `shared-pvc.yaml`:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: shared-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

Create `db-pvc.yaml`:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
```

```
  name: db-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Apply the deployments:

```
kubectl apply -f shared-pvc.yaml
kubectl apply -f db-pvc.yaml
kubectl apply -f frontend-deployment.yaml
kubectl apply -f backend-deployment.yaml
kubectl apply -f db-deployment.yaml
```

Step 3: Deploy a Multi-Tier Application Using Docker Compose

3.1 Create a `docker-compose.yml` File

```
version: '3'
services:
  frontend:
    image: nginx
    ports:
      - "8080:80"
    volumes:
      - shareddata:/usr/share/nginx/html
  backend:
    image: mybackendimage
    ports:
      - "8081:80"
    volumes:
      - shareddata:/app/data
  db:
    image: postgres
    environment:
```

```
      POSTGRES_DB: mydb
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    volumes:
      - dbdata:/var/lib/postgresql/data

volumes:
  shareddata:
  dbdata:
```

3.2 Deploy the Application

```
# Deploy using Docker Compose
docker-compose up -d
```

Step 4: Use a Single Shared Volume Across Multiple Containers

Update `docker-compose.yml` as shown in Step 3.1 to use the `shareddata` volume across the frontend and backend services.