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
```

this command showing error for IP address so user ` docker swarm  --advertise-addr 192.168.56.13 `
this command initialize swarm

```
root@ubuntu2204:/home/vagrant# docker swarm init
Error response from daemon: could not choose an IP address to advertise since this system has multiple addresses on different interfaces (10.0.2.15 on eth0 and 192.168.56.13 on eth1) - specify one with --advertise-addr
root@ubuntu2204:/home/vagrant# docker swarm init --advertise-addr 192.168.56.13
Swarm initialized: current node (kffbjqcbhqxpioqd40h70fr98) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-2159blqedhjjp4ls0qh22fivyq6ty8v73ce8gmsks6qv1rj2fq-cov0rks9w302kdfc87yyh49e3 192.168.56.13:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

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
```

to use this comand first we need nginx image
so use command ` docker pull nginx ` to pull nginx image

```
vagrant@ubuntu2204:~$ sudo docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
f11c1adaa26e: Pull complete
c6b156574604: Downloading   38.2MB
ea5d7144c337: Download complete
1bbcb9df2c93: Download complete
537a6cfe3404: Download complete
c6b156574604: Pull complete
ea5d7144c337: Pull complete
1bbcb9df2c93: Pull complete
537a6cfe3404: Pull complete
767bff2cc03e: Pull complete
adc73cb74f25: Pull complete
Digest: sha256:67682bda769fae1ccf5183192b8daf37b64cae99c6c3302650f6f8bf5f0f95df
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
vagrant@ubuntu2204:~$ sudo docker service create --name nginx-service --publish 8080:80 nginx
f4lpexkyimmrw80g0hafvmh1g
overall progress: 1 out of 1 tasks
1/1: running
verify: Service f4lpexkyimmrw80g0hafvmh1g converged
vagrant@ubuntu2204:~$
```

Step 2: Set up Kubernetes Using Minikube

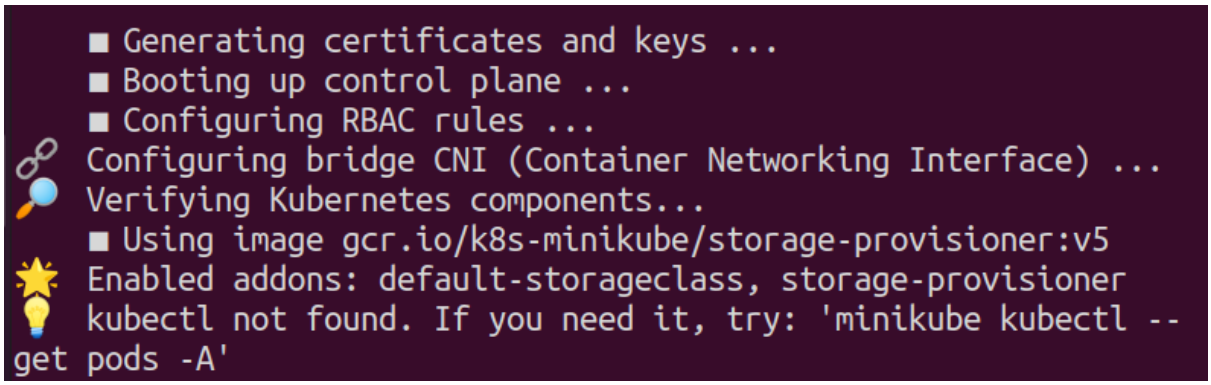
2.1 first downloaded the Minikube using command :

curl -LOhttps://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64

2.2 Start Minikube

```
# Start Minikube
minikube start
```



2.3 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
```

```
vagrant@ubuntu2204:~$ vim webapp-deployment.yaml
vagrant@ubuntu2204:~$ cat 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
vagrant@ubuntu2204:~$
```

Apply the deployment:

kubectl apply -f webapp-deployment.yaml

```
vagrant@ubuntu2204:~$ kubectl apply -f webapp-deployment.yaml
deployment.apps/webapp created
vagrant@ubuntu2204:~$
```

2.4 Expose the Deployment

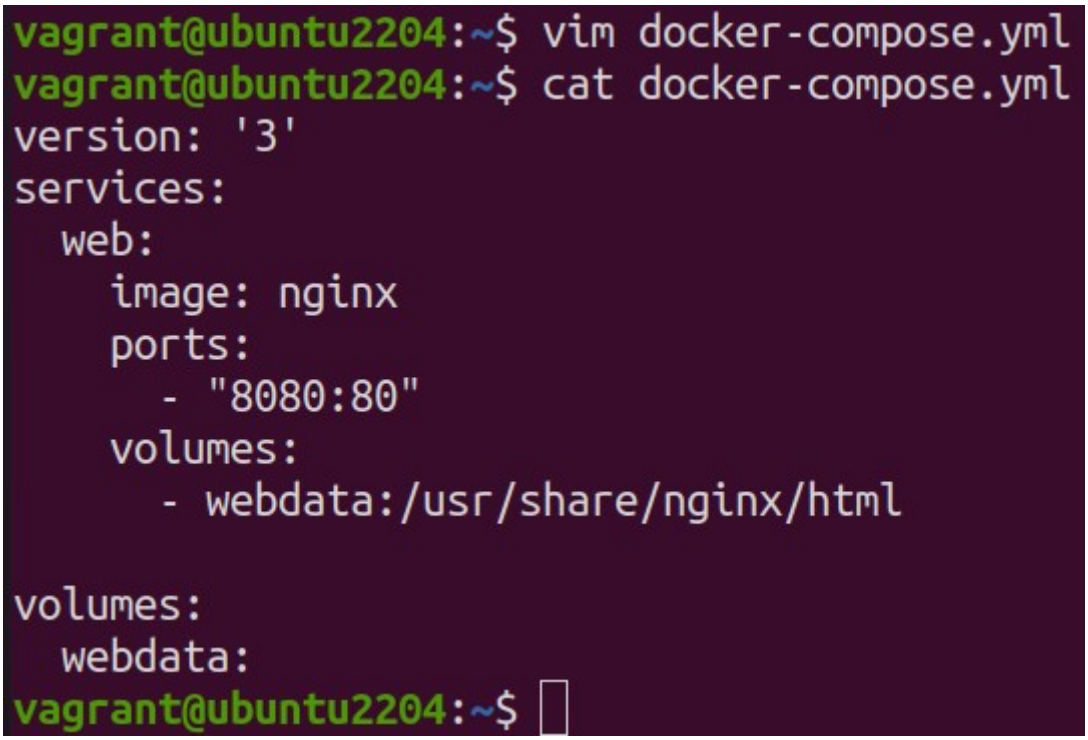
kubectl expose deployment webapp --type=NodePort --port=80

```
vagrant@ubuntu2204:~$ kubectl expose deployment webapp --type=NodePort --port=80
service/webapp exposed
vagrant@ubuntu2204:~$
```

Step 3: Deploy a Web Application Using Docker Compose

3.1 Create a **docker-compose.yml** File

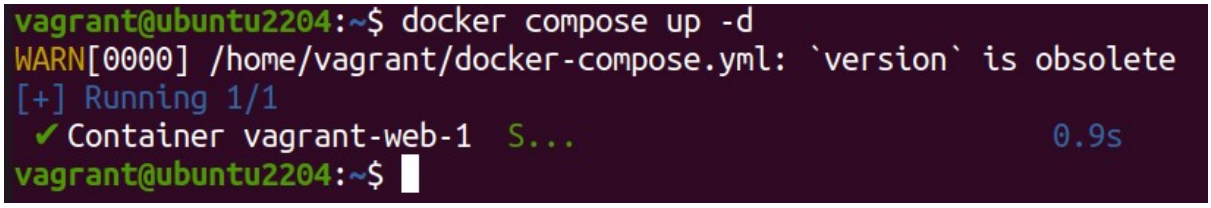
```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "8080:80"
    volumes:
      - webdata:/usr/share/nginx/html

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:
```

```
vagrant@ubuntu2204:~$ vim docker-compose.yml
vagrant@ubuntu2204:~$ cat docker-compose.yml
version: '3'
services:
  web1:
    image: nginx
    ports:
      - "8082:80"
    volumes:
      - shareddata:/usr/share/nginx/html
  web2:
    image: nginx
    ports:
      - "8083:80"
    volumes:
      - shareddata:/usr/share/nginx/html

volumes:
  shareddata:
```

4.2

Deploy with Docker Compose

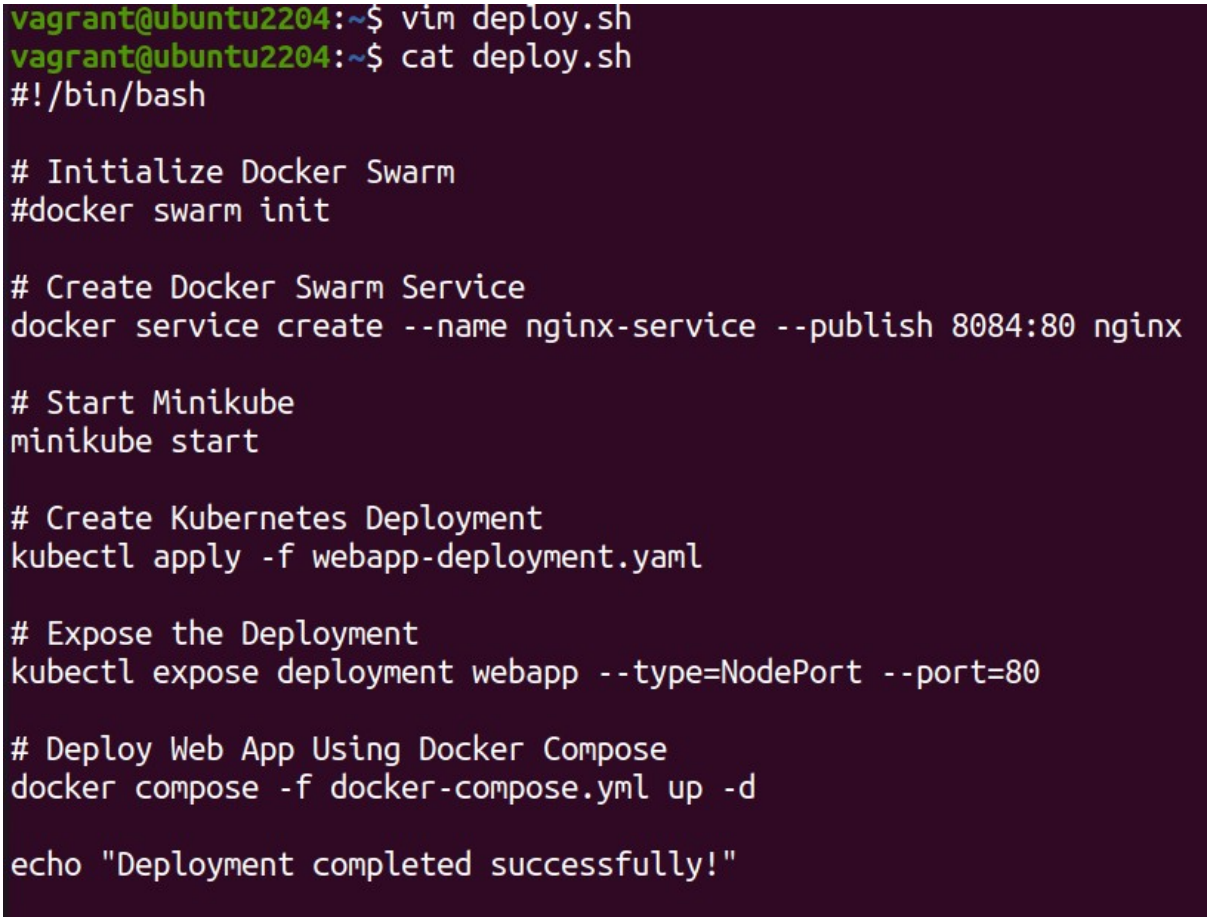
```
# Deploy using Docker Compose
docker-compose up -d
```

```
vagrant@ubuntu2204:~$ docker compose up -d
WARN[0000] /home/vagrant/docker-compose.yml: `version` is ob
solete
WARN[0000] Found orphan containers ([vagrant-web-1]) for thi
s project. If you removed or renamed this service in your co
mpose file, you can run this command with the --remove-orpha
ns flag to clean it up.
[+] Running 2/2
 ✔ Container vagrant-web2-1  Started                    0.7s
 ✔ Container vagrant-web1-1  Started                    0.6s
vagrant@ubuntu2204:~$
```

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
```

```
vagrant@ubuntu2204:~$ chmod +x deploy.sh
```

5.3 Run the Script

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

```
vagrant@ubuntu2204:~$ ./deploy.sh
 MiB
    Updating the running docker "minikube" container ...
    Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
    Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:
v5
    Enabled addons: default-storageclass, storage-provisio
ner
    Done! kubectl is now configured to use "minikube" clus
ter and "default" namespace by default
deployment.apps/webapp unchanged
Error from server (AlreadyExists): services "webapp" alrea
dy exists
WARN[0000] /home/vagrant/docker-compose.yml: `version` is
obsolete
[+] Running 2/2
 ✔ Container vagrant-web2-1  Started                  1.1s
 ✔ Container vagrant-web1-1  Started                  1.1s
Deployment completed successfully!
vagrant@ubuntu2204:~$
```

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:

1. **Step 1**: Set up Docker Swarm and create a multi-tier service.
2. **Step 2**: Set up Kubernetes using Minikube.
3. **Step 3**: Deploy a multi-tier application using Docker Compose.
4. **Step 4**: Use a single shared volume across multiple containers.
5. **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:
```

```
vagrant@ubuntu2204:~$ vim docker-compose-swarm.yml
vagrant@ubuntu2204:~$ cat docker-compose-swarm.yml
version: '3.7'
services:
  frontend:
    image: nginx
    ports:
      - "8082:80"
    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
```

```
vagrant@ubuntu2204:~$ docker stack deploy -c docker-compose-swarm.yml myapp
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Updating service myapp_frontend (id: oahye9kz5n4axghtowd7flxl8)
Updating service myapp_backend (id: 88s7w54sekac30zel9l6x9dgs)
image mybackendimage:latest could not be accessed on a registry to record
its digest. Each node will access mybackendimage:latest independently,
possibly leading to different nodes running different
versions of the image.

Updating service myapp_db (id: idv9qkkfg1u9sqwr9ktxdbhem)
vagrant@ubuntu2204:~$
```

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
```

```
vagrant@ubuntu2204:~$ vim frontend-deployment.yaml
vagrant@ubuntu2204:~$ cat 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
vagrant@ubuntu2204:~$
```

Create backend-deployment.yaml:

```
apiVersion: apps/v1
```

```yaml
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
```

```
vagrant@ubuntu2204:~$ vim backend-deployement.yaml
vagrant@ubuntu2204:~$ cat backend-deployement.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
vagrant@ubuntu2204:~$
```

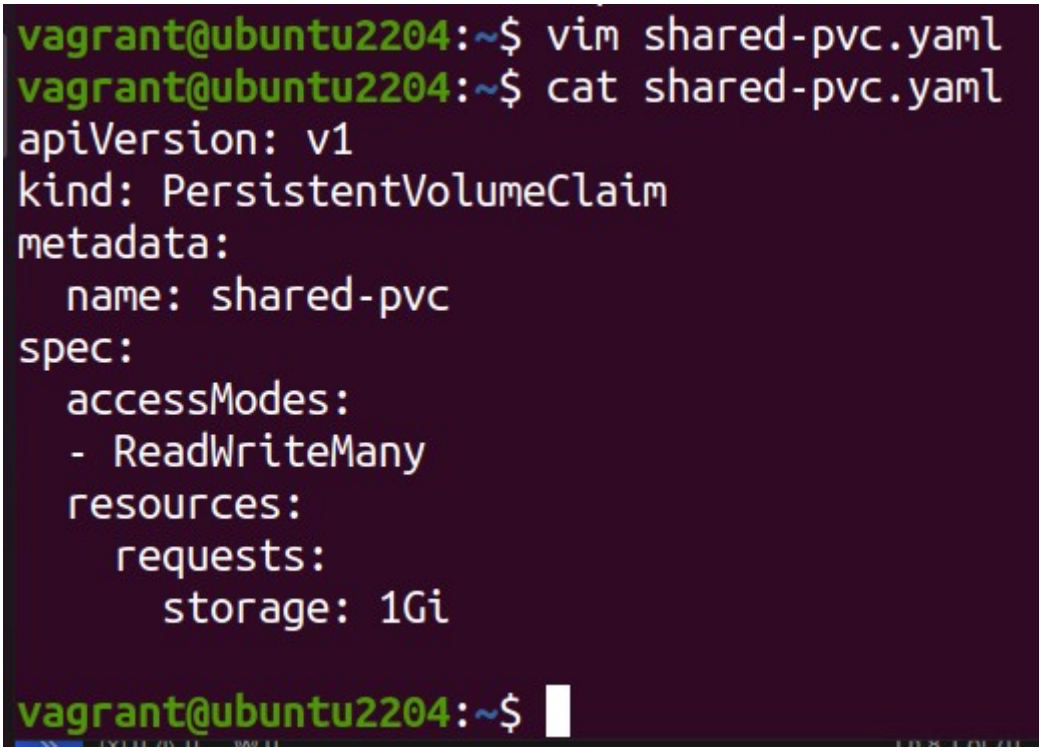
Create db-deployment.yaml:

```
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
```

```
vagrant@ubuntu2204:~$ vim db-deployment.yaml
vagrant@ubuntu2204:~$ cat 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
vagrant@ubuntu2204:~$
```

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
```

```
vagrant@ubuntu2204:~$ vim db-pvc.yaml
vagrant@ubuntu2204:~$ cat db-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
vagrant@ubuntu2204:~$
```

Apply the deployments:

kubectl apply -f shared-pvc.yaml

```
vagrant@ubuntu2204:~$ kubectl apply -f shared-pvc.yaml
persistentvolumeclaim/shared-pvc created
vagrant@ubuntu2204:~$
```

kubectl apply -f db-pvc.yaml

```
vagrant@ubuntu2204:~$ kubectl apply -f db-pvc.yaml
persistentvolumeclaim/db-pvc created
vagrant@ubuntu2204:~$
```

kubectl apply -f frontend-deployment.yaml

```
vagrant@ubuntu2204:~$ kubectl apply -f frontend-deployment.yaml
deployment.apps/frontend created
vagrant@ubuntu2204:~$
```

kubectl apply -f backend-deployment.yaml

```
vagrant@ubuntu2204:~$ kubectl apply -f backend-deployement.yaml
deployment.apps/backend created
vagrant@ubuntu2204:~$
```

kubectl apply -f db-deployment.yaml

```
vagrant@ubuntu2204:~$ kubectl apply -f db-deployment.yaml
deployment.apps/db created
vagrant@ubuntu2204:~$
```

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:
```

```
vagrant@ubuntu2204:~$ vim docker-compose.yml
vagrant@ubuntu2204:~$ cat docker-compose.yml
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.