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.

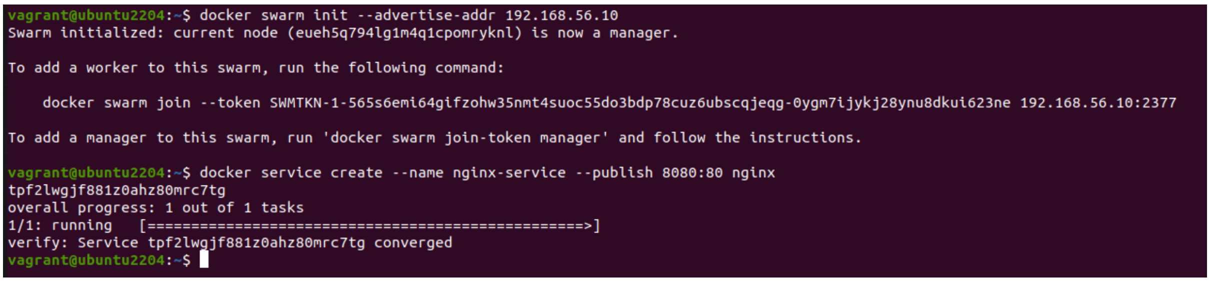
Step 1: Set up Docker Swarm and Create a Service

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
```

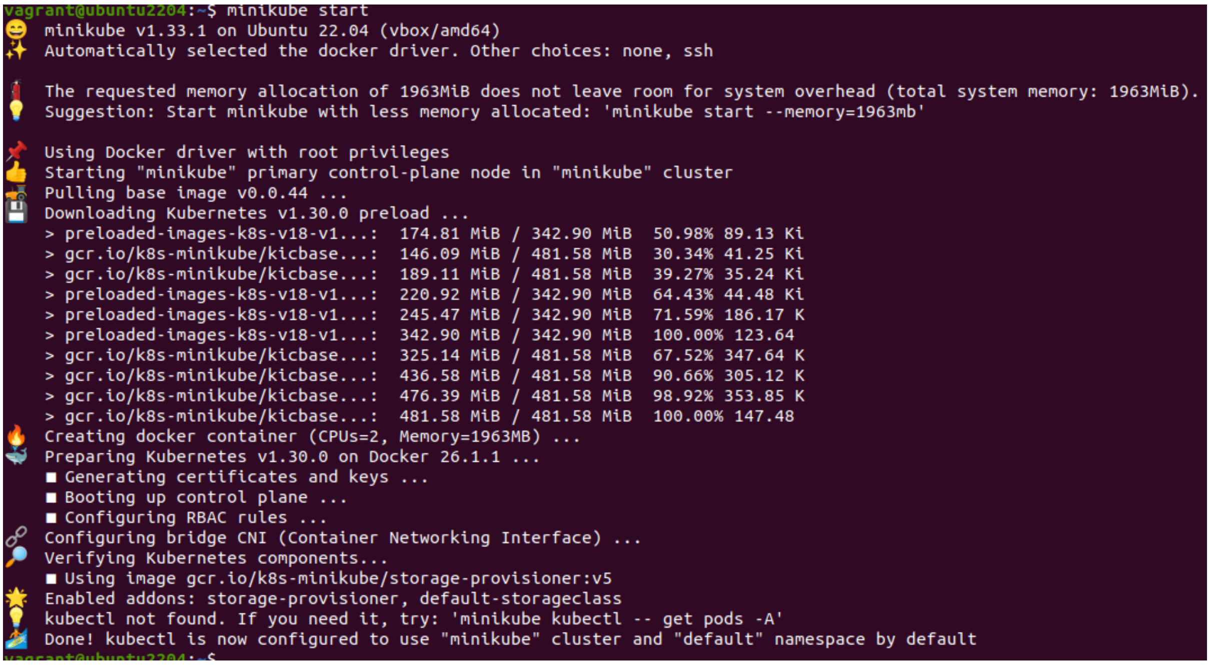


```
vagrant@ubuntu2204:~$ docker swarm init --advertise-addr 192.168.56.10
Swarm initialized: current node (eueh5q794lg1m4q1cpomryknl) is now a manager.

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

    docker swarm join --token SWMTKN-1-565s6emi64gifzohw35nmt4suoc55do3bdp78cuz6ubscqjeqg-0ygm7ijykj28ynu8dkui623ne 192.168.56.10:2377

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

vagrant@ubuntu2204:~$ docker service create --name nginx-service --publish 8080:80 nginx
tpf2lwgjf881z0ahz80mrc7tg
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service tpf2lwgjf881z0ahz80mrc7tg converged
vagrant@ubuntu2204:~$
```

Step 2: Set up Kubernetes Using Minikube

2.1 Start Minikube

```
# Start Minikube
minikube start
```



```
vagrant@ubuntu2204:~$ minikube start
😄  minikube v1.33.1 on Ubuntu 22.04 (vbox/amd64)
✨  Automatically selected the docker driver. Other choices: none, ssh

🧯  The requested memory allocation of 1963MiB does not leave room for system overhead (total system memory: 1963MiB).
💡  Suggestion: Start minikube with less memory allocated: 'minikube start --memory=1963mb'

📌  Using Docker driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.44 ...
💾  Downloading Kubernetes v1.30.0 preload ...
    > preloaded-images-k8s-v18-v1...:  174.81 MiB / 342.90 MiB  50.98% 89.13 Ki
    > gcr.io/k8s-minikube/kicbase...:  146.09 MiB / 481.58 MiB  30.34% 41.25 Ki
    > gcr.io/k8s-minikube/kicbase...:  189.11 MiB / 481.58 MiB  39.27% 35.24 Ki
    > preloaded-images-k8s-v18-v1...:  220.92 MiB / 342.90 MiB  64.43% 44.48 Ki
    > preloaded-images-k8s-v18-v1...:  245.47 MiB / 342.90 MiB  71.59% 186.17 K
    > preloaded-images-k8s-v18-v1...:  342.90 MiB / 342.90 MiB  100.00% 123.64
    > gcr.io/k8s-minikube/kicbase...:  325.14 MiB / 481.58 MiB  67.52% 347.64 K
    > gcr.io/k8s-minikube/kicbase...:  436.58 MiB / 481.58 MiB  90.66% 305.12 K
    > gcr.io/k8s-minikube/kicbase...:  476.39 MiB / 481.58 MiB  98.92% 353.85 K
    > gcr.io/k8s-minikube/kicbase...:  481.58 MiB / 481.58 MiB  100.00% 147.48
🔥  Creating docker container (CPUs=2, Memory=1963MB) ...
🐳  Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass
💡  kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
vagrant@ubuntu2204:~$
```

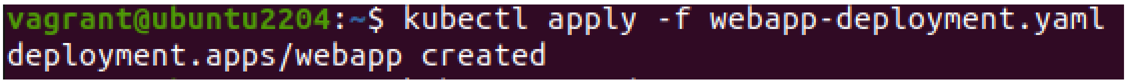
2.2 Deploy a Web App on Kubernetes

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:

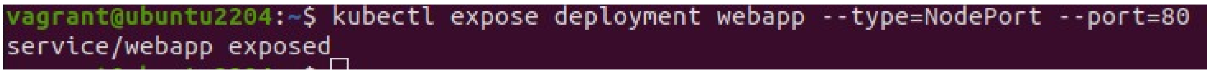
```
kubectl apply -f webapp-deployment.yaml
```



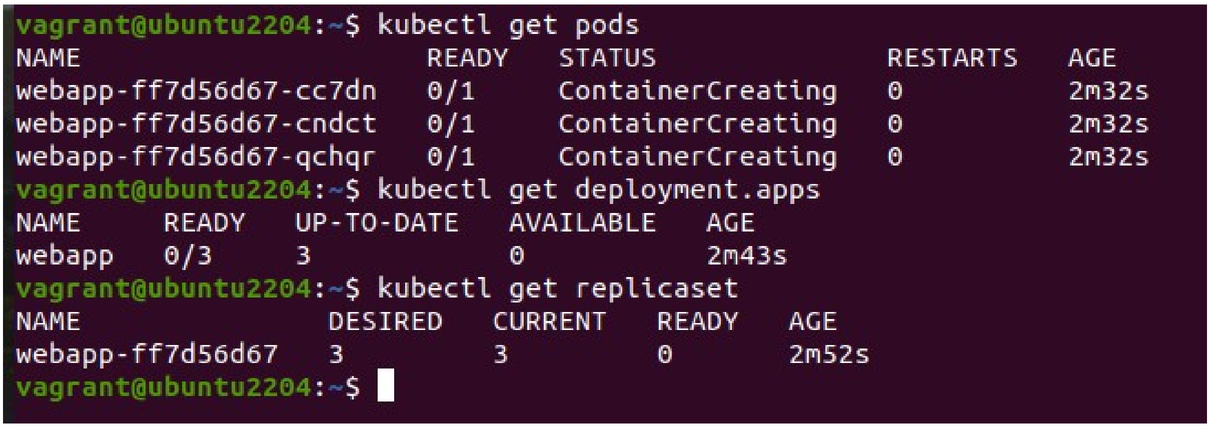
```
vagrant@ubuntu2204:~$ kubectl apply -f webapp-deployment.yaml
deployment.apps/webapp created
```

2.3 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:~$ kubectl get pods
NAME                         READY   STATUS              RESTARTS   AGE
webapp-ff7d56d67-cc7dn       0/1     ContainerCreating   0          2m32s
webapp-ff7d56d67-cndct       0/1     ContainerCreating   0          2m32s
webapp-ff7d56d67-qchqr       0/1     ContainerCreating   0          2m32s
vagrant@ubuntu2204:~$ kubectl get deployment.apps
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
webapp   0/3     3            0           2m43s
vagrant@ubuntu2204:~$ kubectl get replicaset
NAME               DESIRED   CURRENT   READY   AGE
webapp-ff7d56d67   3         3         0       2m52s
vagrant@ubuntu2204:~$
```

Step 3: Deploy a Web Application Using Docker Compose

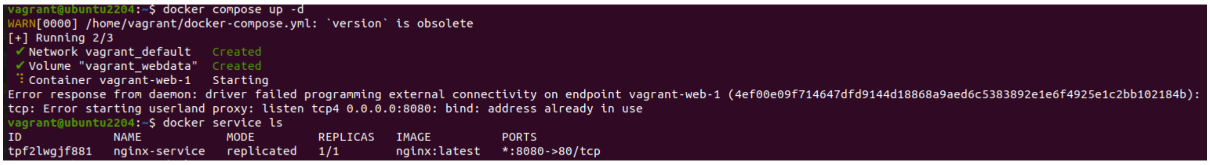
3.1 Create a `docker-compose.yml` File

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

volumes:
  webdata:
```

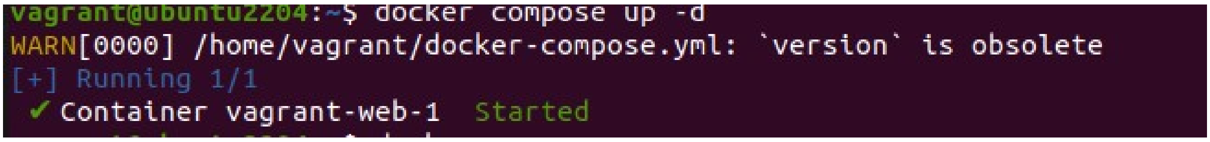
3.2 Deploy the Web Application

```
# Deploy using Docker Compose
```

```
docker-compose up -d
```



```
vagrant@ubuntu2204:~$ docker compose up -d
WARN[0000] /home/vagrant/docker-compose.yml: `version` is obsolete
[+] Running 2/3
 ✔ Network vagrant_default    Created
 ✔ Volume "vagrant_webdata"   Created
 ⠸ Container vagrant-web-1    Starting
Error response from daemon: driver failed programming external connectivity on endpoint vagrant-web-1 (4ef00e09f714647dfd9144d18868a9aed6c5383892e1e6f4925e1c2bb102184b):
tcp: Error starting userland proxy: listen tcp4 0.0.0.0:8080: bind: address already in use
vagrant@ubuntu2204:~$ docker service ls
ID             NAME            MODE         REPLICAS   IMAGE          PORTS
tpf2lwgjf881   nginx-service   replicated   1/1        nginx:latest   *:8080->80/tcp
```



```
vagrant@ubuntu2204:~$ docker compose up -d
WARN[0000] /home/vagrant/docker-compose.yml: `version` is obsolete
[+] Running 1/1
 ✔ Container vagrant-web-1  Started
```

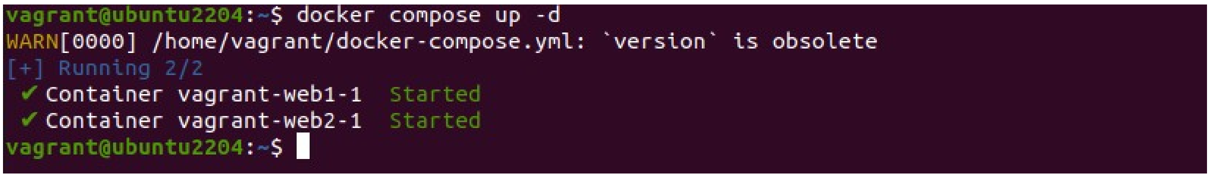
Step 4: Use a Single Shared Volume Across Multiple Containers

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
```



```
vagrant@ubuntu2204:~$ docker compose up -d
WARN[0000] /home/vagrant/docker-compose.yml: `version` is obsolete
[+] Running 2/2
 ✔ Container vagrant-web1-1  Started
 ✔ Container vagrant-web2-1  Started
vagrant@ubuntu2204:~$
```

Step 5: Automate the Entire Process Using Advanced Shell Scripting

5.1 Create a Shell Script deploy.sh

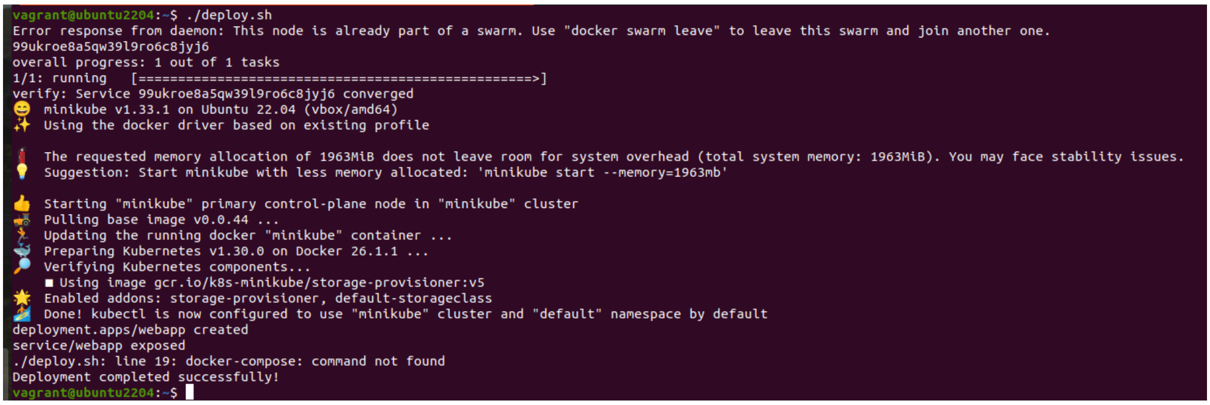
```
#!/bin/bash

# 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
```

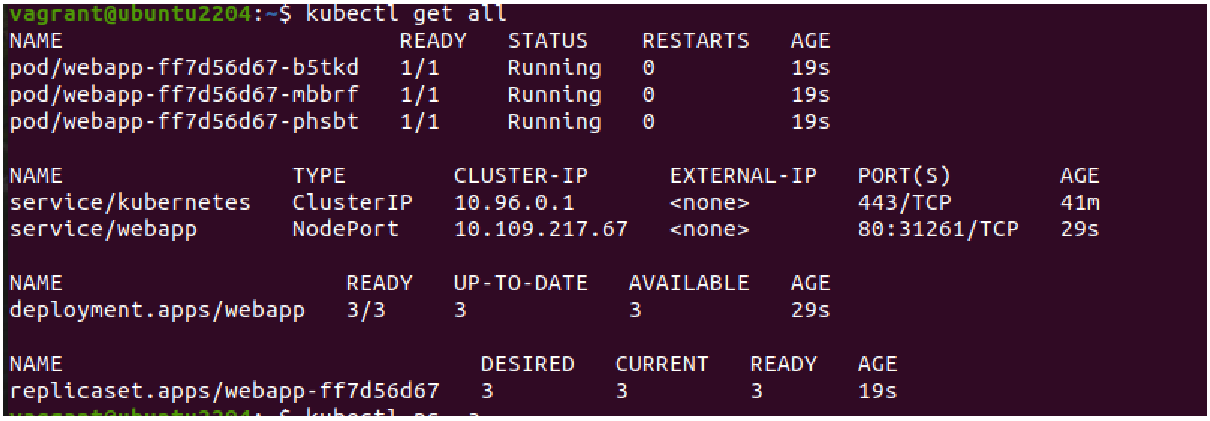
5.3 Run the Script

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

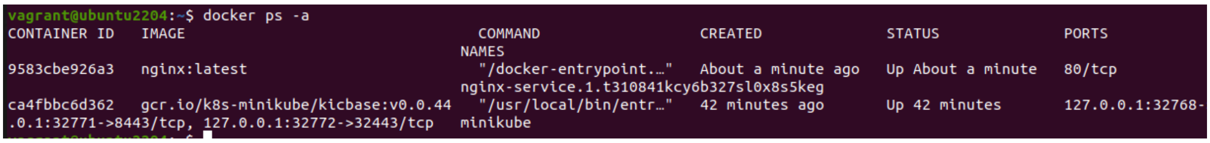


```
vagrant@ubuntu2204:~$ ./deploy.sh
Error response from daemon: This node is already part of a swarm. Use "docker swarm leave" to leave this swarm and join another one.
99ukroe8a5qw39l9ro6c8jyj6
overall progress: 1 out of 1 tasks
1/1: running   [==================================================>]
verify: Service 99ukroe8a5qw39l9ro6c8jyj6 converged
😄  minikube v1.33.1 on Ubuntu 22.04 (vbox/amd64)
✨  Using the docker driver based on existing profile

🧯  The requested memory allocation of 1963MiB does not leave room for system overhead (total system memory: 1963MiB). You may face stability issues.
💡  Suggestion: Start minikube with less memory allocated: 'minikube start --memory=1963mb'

👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.44 ...
🏃  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: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
deployment.apps/webapp created
service/webapp exposed
./deploy.sh: line 19: docker-compose: command not found
Deployment completed successfully!
vagrant@ubuntu2204:~$
```

Project 02 - 1 Hour



```
vagrant@ubuntu2204:~$ kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/webapp-ff7d56d67-b5tkd   1/1     Running   0          19s
pod/webapp-ff7d56d67-mbbrf   1/1     Running   0          19s
pod/webapp-ff7d56d67-phsbt   1/1     Running   0          19s

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        41m
service/webapp       NodePort    10.109.217.67   <none>        80:31261/TCP   29s

NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/webapp   3/3     3            3           29s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/webapp-ff7d56d67   3         3         3       19s
```



```
vagrant@ubuntu2204:~$ docker ps -a
CONTAINER ID   IMAGE                                 COMMAND                  CREATED              STATUS              PORTS
                                                   NAMES
9583cbe926a3   nginx:latest                            "/docker-entrypoint.…"   About a minute ago   Up About a minute   80/tcp
                                                   nginx-service.1.t310841kcy6b327sl0x8s5keg
ca4fbbc6d362   gcr.io/k8s-minikube/kicbase:v0.0.44     "/usr/local/bin/entr…"   42 minutes ago       Up 42 minutes       127.0.0.1:32768-
.0.1:32771->8443/tcp, 127.0.0.1:32772->32443/tcp   minikube
```

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
```

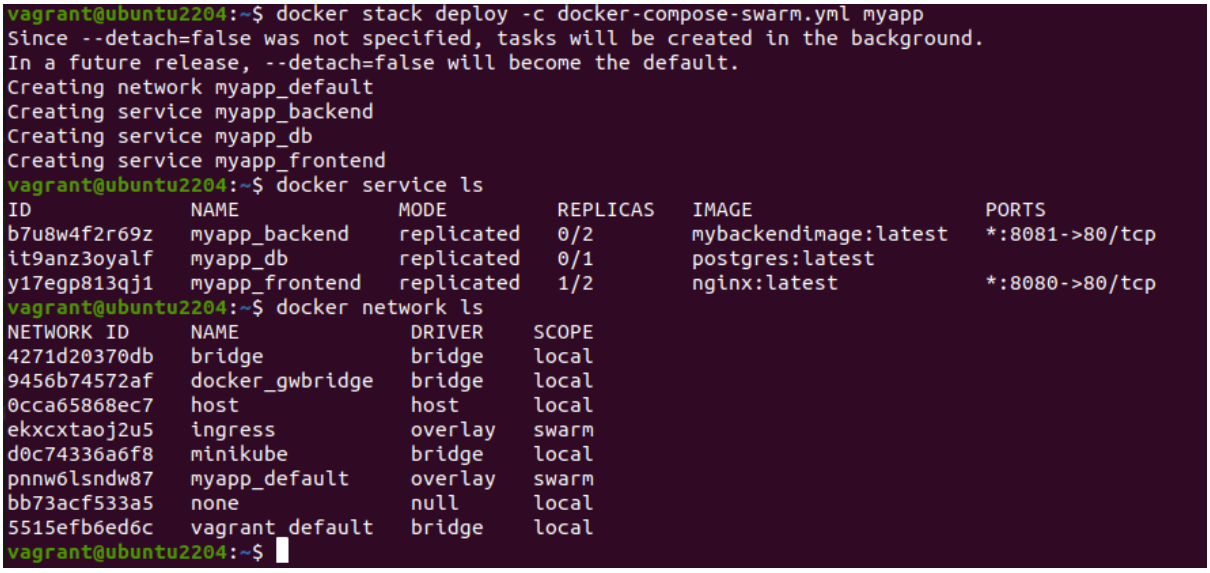
1.2 Create a Multi-Tier Docker Swarm Service

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:
```

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.
Creating network myapp_default
Creating service myapp_backend
Creating service myapp_db
Creating service myapp_frontend
vagrant@ubuntu2204:~$ docker service ls
ID             NAME             MODE         REPLICAS   IMAGE                     PORTS
b7u8w4f2r69z   myapp_backend    replicated   0/2        mybackendimage:latest     *:8081->80/tcp
it9anz3oyalf   myapp_db         replicated   0/1        postgres:latest
y17egp813qj1   myapp_frontend   replicated   1/2        nginx:latest              *:8080->80/tcp
vagrant@ubuntu2204:~$ docker network ls
NETWORK ID     NAME              DRIVER    SCOPE
4271d20370db   bridge            bridge    local
9456b74572af   docker_gwbridge   bridge    local
0cca65868ec7   host              host      local
ekxcxtaoj2u5   ingress           overlay   swarm
d0c74336a6f8   minikube          bridge    local
pnnw6lsndw87   myapp_default     overlay   swarm
bb73acf533a5   none              null      local
5515efb6ed6c   vagrant_default   bridge    local
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
```

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`:

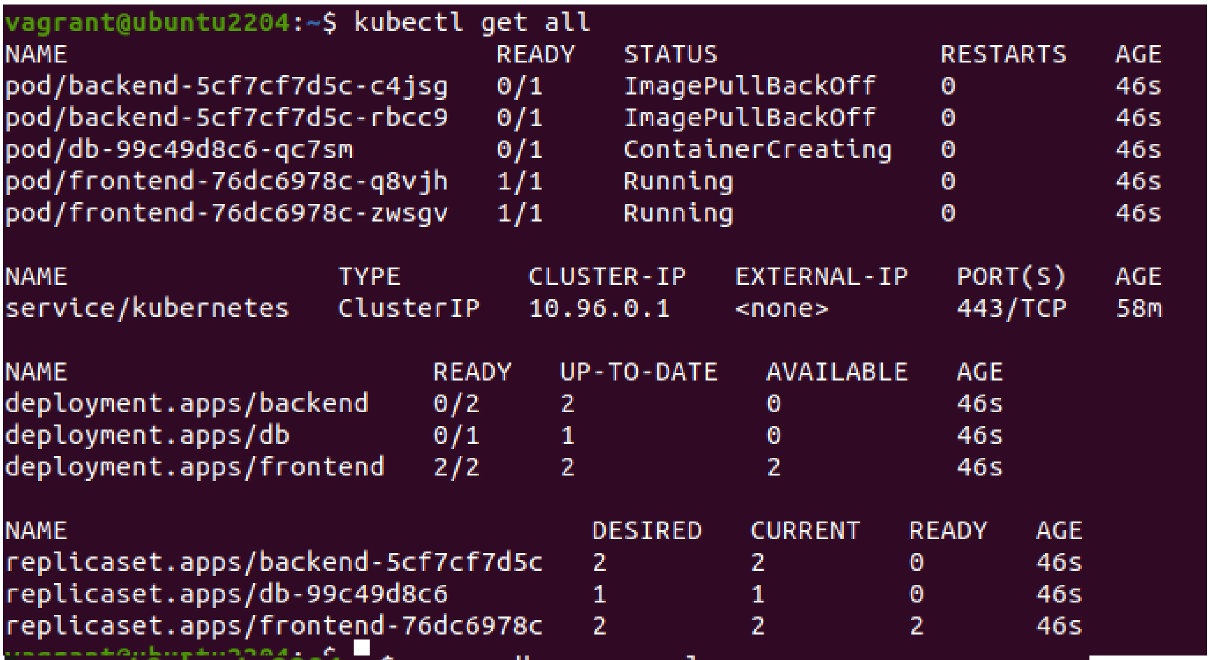
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: shared-pvc
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

Create db-pvc.yaml:

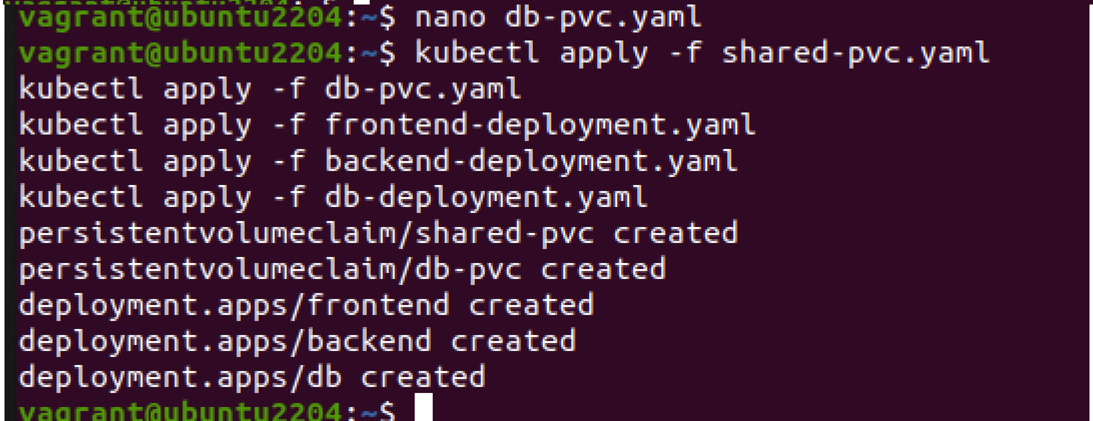
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

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
```



```
vagrant@ubuntu2204:~$ kubectl get all
NAME                               READY   STATUS              RESTARTS   AGE
pod/backend-5cf7cf7d5c-c4jsg       0/1     ImagePullBackOff    0          46s
pod/backend-5cf7cf7d5c-rbcc9       0/1     ImagePullBackOff    0          46s
pod/db-99c49d8c6-qc7sm             0/1     ContainerCreating   0          46s
pod/frontend-76dc6978c-q8vjh       1/1     Running             0          46s
pod/frontend-76dc6978c-zwsgv       1/1     Running             0          46s

NAME                     TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes       ClusterIP   10.96.0.1    <none>        443/TCP   58m

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/backend       0/2     2            0           46s
deployment.apps/db            0/1     1            0           46s
deployment.apps/frontend      2/2     2            2           46s

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/backend-5cf7cf7d5c      2         2         0       46s
replicaset.apps/db-99c49d8c6            1         1         0       46s
replicaset.apps/frontend-76dc6978c      2         2         2       46s
```



```
vagrant@ubuntu2204:~$ nano db-pvc.yaml
vagrant@ubuntu2204:~$ 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
persistentvolumeclaim/shared-pvc created
persistentvolumeclaim/db-pvc created
deployment.apps/frontend created
deployment.apps/backend created
deployment.apps/db created
vagrant@ubuntu2204:~$
```

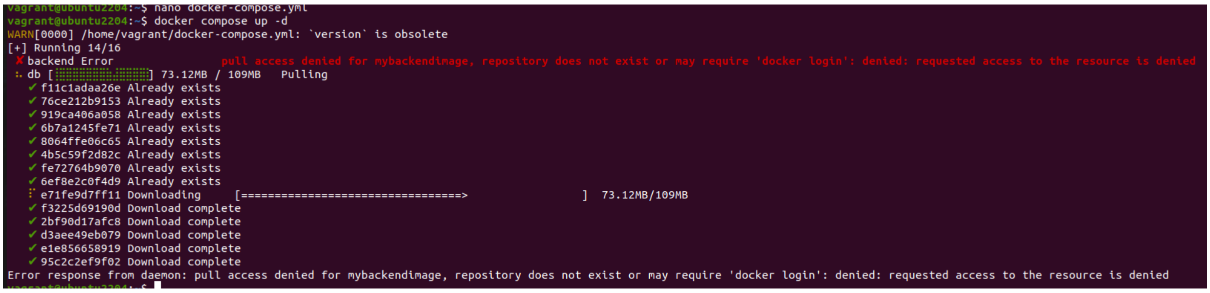
Step 3: Deploy a Multi-Tier Application Using Docker Compose

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:
```

3.2 Deploy the Application

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



```
vagrant@ubuntu2204:~$ nano docker-compose.yml
vagrant@ubuntu2204:~$ docker compose up -d
WARN[0000] /home/vagrant/docker-compose.yml: `version` is obsolete
[+] Running 14/16
 ✘ backend Error                pull access denied for mybackendimage, repository does not exist or may require 'docker login': denied: requested access to the resource is denied
 ⠼ db [⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣀⣿⣿⣿⣿⣿⣿⣿] 73.12MB / 109MB   Pulling
   ✔ f11c1adaa26e Already exists
   ✔ 76ce212b9153 Already exists
   ✔ 919ca406a058 Already exists
   ✔ 6b7a1245fe71 Already exists
   ✔ 8064ffe06c65 Already exists
   ✔ 4b5c59f2d82c Already exists
   ✔ fe72764b9070 Already exists
   ✔ 6ef8e2c0f4d9 Already exists
   ⠏ e71fe9d7ff11 Downloading     [=================================>                 ]  73.12MB/109MB
   ✔ f3225d69190d Download complete
   ✔ 2bf90d17afc8 Download complete
   ✔ d3aee49eb079 Download complete
   ✔ e1e856658919 Download complete
   ✔ 95c2c2ef9f02 Download complete
Error response from daemon: pull access denied for mybackendimage, repository does not exist or may require 'docker login': denied: requested access to the resource is denied
vagrant@ubuntu2204:~$ 
```

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.