Exp6

1. `docker run --name mycontainer -it ubuntu:16.04 /bin/bash`

bash Copy code

```bash
root@abc123:/#
```

This will open an interactive shell inside a new container named `mycontainer` using the `ubuntu:16.04` image.

2. `docker start mycontainer`

bash Copy code

```bash
mycontainer
```

This starts the previously stopped container named `mycontainer`.

3. `docker stop mycontainer`

bash Copy code

```bash
mycontainer
```

This stops the container named `mycontainer`.

4. `docker rm mycontainer`

bash Copy code

```bash
mycontainer
```

This removes the container named `mycontainer`.

5. `docker ps`

bash Copy code

```bash
CONTAINER ID   IMAGE          COMMAND       CREATED       STATUS       PORTS   NAMES
```

(No running containers if none are active.)

6. `docker images`

bash Copy code

```bash
REPOSITORY     TAG       IMAGE ID       CREATED        SIZE
ubuntu         16.04     abcdef123456   3 days ago     123MB
```

7. `docker pull ubuntu:16.04`

bash Copy code

```bash
16.04: Pulling from library/ubuntu
abc123: Already exists
def456: Pull complete
Digest: sha256:abcdef1234567890
Status: Downloaded newer image for ubuntu:16.04
```

This pulls the Ubuntu 16.04 image.

8. `docker push myimage`

bash Copy code

```bash
The push refers to repository [docker.io/library/myimage]
abc123: Layer already exists
def456: Layer already exists
myimage: Digest: sha256:abcdef1234567890 size: 1234
```

Exp 7

1. Write a `Dockerfile`

The `Dockerfile` content:

bash Copy code

```bash
# Use the official Python image as the base image
FROM python:3.9

# Set the working directory
WORKDIR /app

# Copy the Python script into the container
COPY hello.py /app

# Run the Python script when the container starts
CMD ["python", "hello.py"]
```

---

2. Build the Docker Image

Command:

bash Copy code

```bash
docker build -t myimage .
```

```
Sending build context to Docker daemon  3.072kB
Step 1/4 : FROM python:3.9
 ---> abc123456789
Step 2/4 : WORKDIR /app
 ---> Running in 123abc456def
Removing intermediate container 123abc456def
 ---> abc123456789
Step 3/4 : COPY hello.py /app
 ---> Using cache
 ---> abc123456789
Step 4/4 : CMD ["python", "hello.py"]
 ---> Running in 123abc456def
Removing intermediate container 123abc456def
 ---> abc123456789
Successfully built abc123456789
Successfully tagged myimage:latest
```

3. Run the Docker Container

Command:

bash Copy code

```bash
docker run --name mycontainer myimage
```

Terminal Output:

Copy code

```
Hello, World!
```

---

4. Verify the Output

Command:

bash Copy code

```bash
docker logs mycontainer
```

Terminal Output:

Copy code

```
Hello, World!
```

Exp8

D

1. Build a Docker Image

Command:

bash Copy code

```bash
docker build -t myapplication .
```

Output:

vbnet Copy code

```vbnet
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM python:3.9
 ---> abc123456789
Step 2/3 : COPY app/ /app
 ---> Using cache
Step 3/3 : CMD ["python", "/app/app.py"]
 ---> Running in 123abc456def
Successfully built abc123456789
Successfully tagged myapplication:latest
```

2. Push the Docker Image to a Registry

Command:

bash Copy code

```bash
docker tag myapplication:latest myrepo/myapplication:latest
docker push myrepo/myapplication:latest
```

Output:

vbnet Copy code

```vbnet
The push refers to repository [docker.io/myrepo/myapplication]
abc123: Pushed
def456: Pushed
Digest: sha256:abcdef123456789 size: 1234
```

3. Deploy the Image to Kubernetes

Create a Deployment YAML file (deployment.yaml):

yaml Copy code

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapplication-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapplication
  template:
    metadata:
      labels:
        app: myapplication
    spec:
      containers:
      - name: myapplication
        image: myrepo/myapplication:latest
        ports:
        - containerPort: 80
```

Apply the Deployment:

bash Copy code

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

Output:

bash Copy code

```bash
deployment.apps/myapplication-deployment created
```

---

4. Monitor the Deployment

Check the Deployment Status:

bash Copy code

```bash
kubectl get deployments
```

Output:

mathematica Copy code

```mathematica
Application started successfully
```

Exp 9

1. Create a Kubernetes Cluster

Use tools like Minikube, Google Kubernetes Engine (GKE), or Amazon Elastic Kubernetes Service (EKS) to create a cluster.

Example Command (Minikube):

bash Copy code

```bash
minikube start
```

Output:

vbnet Copy code

```vbnet
😄  minikube v1.30.0 on Linux
🏃  Starting control plane node minikube in cluster minikube
🔗  Configuring CNI (Container Networking Interface) ...
✅  Kubernetes 1.27.4 is now running!
```

---

2. Push Docker Image to a Registry

Push your Docker image to a public or private registry.

Commands:

bash Copy code

```bash
docker tag myapp myrepo/myapp:latest
docker push myrepo/myapp:latest
```

Output:

vbnet Copy code

```vbnet
The push refers to repository [docker.io/myrepo/myapp]
abc123: Layer already exists
def456: Layer already exists
Digest: sha256:abcdef1234567890 size: 1234
```

3. Create a Deployment

Write a deployment YAML file (`deployment.yaml`).

Example Deployment YAML:

yaml Copy code

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp
        image: myrepo/myapp:latest
        ports:
        - containerPort: 80
```

Apply the Deployment:

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

Output:

```bash
deployment.apps/myapp created
```

---

4. Create a Service

Write a service YAML file (`service.yaml`).

Example Service YAML:

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

Apply the Service:

```bash
kubectl apply -f service.yaml
```

Output:

```bash
service/myapp-service created
```

5. Verify the Deployment and Service

Check Pods:

bash Copy code

```bash
kubectl get pods
```

Output:

sql Copy code

```sql
NAME                    READY   STATUS    RESTARTS   AGE
myapp-abc123            1/1     Running   0          3m
myapp-def456            1/1     Running   0          3m
myapp-ghi789            1/1     Running   0          3m
```

Check Services:

bash Copy code

```bash
kubectl get services
```

Output:

scss Copy code

```scss
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
myapp-service   ClusterIP   10.96.0.1      <none>        80/TCP    5m
```

Exp 10

1. Install the Java Development Kit (JDK)

Download and install JDK from the official Oracle website.

Verify Installation:

bash Copy code

```bash
java -version
```

Output:

scss Copy code

```scss
java version "11.0.18" 2024-01-01 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.18+10)
Java HotSpot(TM) 64-Bit Server VM (build 11.0.18+10, mixed mode)
```

Compile the Java Code:

bash Copy code

```bash
javac Main.java
```

Run the Java Program:

bash Copy code

```bash
java Main
```

Expected Output:

csharp Copy code

```csharp
Page title is: Google
```