Experiment No. 10

Aim: To study and implement container orchestration using Kubernetes

Objective: To understand container orchestration using Kubernetes.

Theory:

Container orchestration automates the deployment, management, scaling, and networking of containers. Container orchestration can be used in any environment where you use containers. It can help you to deploy the same application across different environments without needing to redesign it. And microservices in containers make it easier to orchestrate services, including storage, networking, and security. container orchestration to automate and manage tasks such as:

- Provisioning and deployment
- Configuration and scheduling
- Resource allocation
- Container availability
- Scaling or removing containers based on balancing workloads across your infrastructure
- Load balancing and traffic routing
- Monitoring container health
- Configuring applications based on the container in which they will run
- Keeping interactions between containers secure

Kubernetes is an open-source container management (orchestration) tool. Its container management responsibilities include container deployment, scaling & descaling of containers & container load balancing.

Features of Kubernetes

- Automatic Bin packing
- Service Discovery and Load Balancing
- Storage Orchestration
- Self-Healing
- Secrete and configuration management.
- Batch execution
- Horizontal Scaling
- Automatic Rollbacks and Rollouts

Steps:

----Enable Kubernetes---

1. After installing Docker Desktop, you should see a Docker icon in your system tray. Right-click on it, and navigate **Settings** > **Kubernetes**.
2. Check the checkbox labeled **Enable Kubernetes**, and click **Apply & Restart**. Docker Desktop will automatically set up Kubernetes for you. You'll know that Kubernetes has been successfully enabled when you see a green light beside 'Kubernetes *running*' in the **Settings** menu.
3. In order to confirm that Kubernetes is up and running, create a text file called `pod.yaml` with the following content:

```
apiVersion: v1
kind:       Pod
metadata:
  name: demo
spec:
  containers:
  -    name:     testpod
    image: alpine:latest
    command: ["ping", "8.8.8.8"]
```

This describes a pod with a single container, isolating a simple ping to 8.8.8.8.

4. In PowerShell, navigate to where you created `pod.yaml` and create your pod:

```
$ kubectl apply -f pod.yaml
```

5. Check that your pod is up and running:

```
$ kubectl get pods
```

You should see something like:

```
NAME     READY     STATUS   RESTARTS
demo                       Running          4
```

6. Check that you get the logs you'd expect for a ping process:

```
$ kubectl logs demo
```

7. Finally, tear down your test pod:

```
$ kubectl delete -f pod.yaml
```

---Deploy Kubernetes---

Prerequisite

- Download and install Docker Desktop as described in Get Docker.
- Work through containerizing an application in Part 2.
- Make sure that Kubernetes is enabled on your Docker Desktop:
  - **Windows**: Click the Docker icon in the system tray and navigateto **Settings** and make sure there's a green light beside 'Kubernetes'.

Describing apps using Kubernetes YAML

1. Place the following in a file called bb.yaml:apiVersion:

```
apps/v1
kind: Deployment
metadata:
  name:        bb-demo
namespace:     default
spec:
  replicas: 1
  selector:
    matchLabels:
      bb:     web
  template:
  metadata:
     labels:
       bb: web
   spec:
     containers:
     - name: bb-site
       image:    getting-started
       imagePullPolicy: Never
---
apiVersion: v1
kind:   Service
metadata:
  name: bb-entrypoint
  namespace: default
spec:
  type: NodePort
```

```
  selector:
    bb: web
  ports:
  - port: 3000
    targetPort: 3000
    nodePort: 30001
```

--Deploy and check application—

1. In a terminal, navigate to where you created `bb.yaml` and deploy your application to Kubernetes:

```
$ kubectl apply -f bb.yaml
```

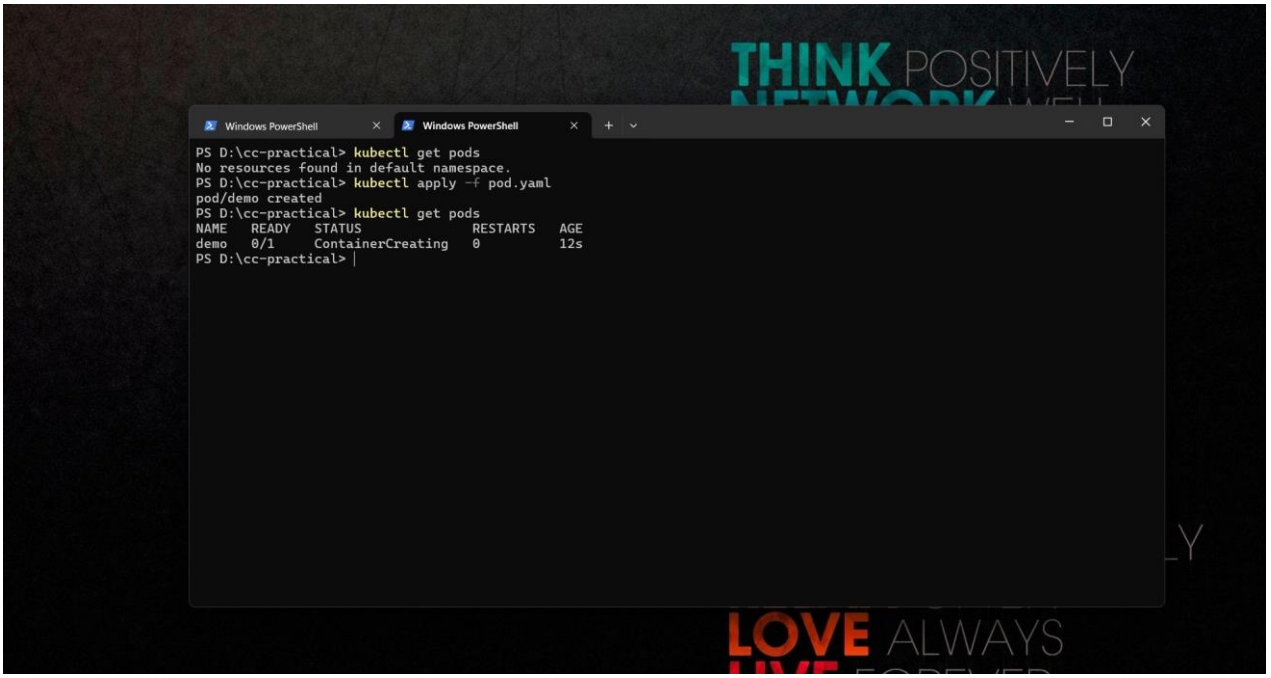
2. Make sure everything worked by listing your deployments:

```
$ kubectl get deployments
```

3. Open a browser and visit your Todo app at `localhost:30001`; you should see yourTodo application.
4. Once satisfied, tear down your application:

```
$ kubectl delete -f bb.yaml
```

Output/Observation:



Conclusion:

Implementing containerization with Kubernetes provides a robust orchestration framework for managing containerized applications at scale. Kubernetes automates deployment, scaling, and management, ensuring high availability and fault tolerance. Its declarative configuration and self-healing capabilities simplify operations and reduce manual intervention. Kubernetes integrates seamlessly with cloud-native technologies, enabling efficient resource utilization and enabling agile development practices.