**📄 Documentation: Deploying and Managing Applications in Kubernetes with Minikube**

**🎯 Objective**

To deploy and manage a sample application using a local Kubernetes cluster created with Minikube and control it using kubectl.

**🛠️ Tools Used**

* **Minikube** – to run a Kubernetes cluster locally.
* **kubectl** – to interact with the Kubernetes cluster.
* **Docker Desktop** – as the driver for running containers.
* **Git** – to push the project files and screenshots to GitHub.

**✅ Steps Performed**

**🔹 1. Installing Tools**

| **Tool** | **How It Was Installed** | **Purpose** |
| --- | --- | --- |
| **Docker Desktop** | Downloaded from <https://www.docker.com> and installed. | Provides container engine needed for Minikube. |
| **Minikube** | Downloaded .exe from <https://minikube.sigs.k8s.io> and added to PATH. | Sets up and runs the Kubernetes cluster. |
| **kubectl** | Installed using Minikube’s built-in version. | To control Kubernetes using commands. |

📌 *Why?*  
All these tools work together to simulate a real Kubernetes environment on your laptop.

**🔹 2. Starting the Cluster**

bash

CopyEdit

minikube start --driver=docker

✅ **What happened**:  
This command starts a Kubernetes cluster using Docker.

❌ **Error Faced**:

arduino

CopyEdit

apiServer.certSANs: Invalid value: ""

🛠️ **Fix Applied**:  
We added a valid hostname with:

bash

CopyEdit

minikube start --driver=docker --apiserver-name=localhost

✅ This successfully booted up the control plane and enabled required addons.

📌 *Why?*  
Minikube needs a valid hostname to generate certificates. Adding --apiserver-name=localhost fixes the invalid DNS issue.

**🔹 3. Verifying Cluster is Working**

bash

CopyEdit

minikube status

minikube kubectl -- get pods -A

📌 *Why?*  
We check the status of the cluster and ensure Kubernetes components are working properly.

**🔹 4. Creating Deployment YAML**

A file named deployment.yaml was created with the following content:

yaml

CopyEdit

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello-deployment

spec:

replicas: 2

selector:

matchLabels:

app: hello

template:

metadata:

labels:

app: hello

spec:

containers:

- name: hello-container

image: nginx

ports:

- containerPort: 80

✅ This was applied using:

bash

CopyEdit

kubectl apply -f deployment.yaml

📌 *Why?*  
A Deployment manages multiple instances (replicas) of the application and ensures availability.

**🔹 5. Creating a Service YAML**

A file service.yaml was created to expose the app:

yaml

CopyEdit

apiVersion: v1

kind: Service

metadata:

name: hello-service

spec:

type: NodePort

selector:

app: hello

ports:

- protocol: TCP

port: 80

targetPort: 80

Applied with:

bash

CopyEdit

kubectl apply -f service.yaml

📌 *Why?*  
A Service exposes the app on a port so it can be accessed in a browser or via curl.

**🔹 6. Verifying Pods and Services**

bash

CopyEdit

kubectl get pods

kubectl get services

📌 *Why?*  
To make sure our deployment and service are running successfully.

**🔹 7. Scaling the Deployment**

bash

CopyEdit

kubectl scale deployment hello-deployment --replicas=4

📌 *Why?*  
To increase the number of pods and handle more traffic or add high availability.

**🔹 8. Describing Resources and Viewing Logs**

bash

CopyEdit

kubectl describe deployment hello-deployment

kubectl logs <pod-name>

📌 *Why?*  
To debug or inspect the current state and issues inside the pods.

**🔹 9. Fixing YAML Apply Error**

❌ **Error Faced**:

bash

CopyEdit

error: the path "deployment.yaml" does not exist

✅ **Fix Applied**:  
We navigated properly to the folder using:

bash

CopyEdit

cd "E:\E Drive\Raichal\Elevate labs\k8s-demo"

📌 *Why?*  
You must run kubectl apply from the folder where the YAML file is saved, or give the full path.

**🔹 10. Adding Screenshots**

All screenshots (pods, services, scaling, etc.) were saved inside the same folder k8s-demo for submission.

📌 *Why?*  
To show proof of working steps and for visual clarity in the project report.

**🔹 11. Pushing to GitHub**

bash

CopyEdit

git init

git remote add origin https://github.com/your-username/k8s-demo.git

git add .

git commit -m "Added k8s project with YAML and screenshots"

git push -u origin main

📌 *Why?*  
This helps in version control, cloud backup, and easy project sharing.

**🧠 Summary**

| **Step** | **What You Did** | **Why You Did It** |
| --- | --- | --- |
| Installed Minikube & Docker | To create a local Kubernetes cluster | Simulate production environment |
| Fixed start error | Used --apiserver-name=localhost | Fix certificate issue |
| Created deployment.yaml | Deployed nginx container | Manage replicas |
| Created service.yaml | Exposed app on port | Access from browser |
| Verified with kubectl | Checked pod & service status | Ensure things work |
| Scaled deployment | Increased pods | High availability |
| Used describe/logs | Debug and inspect pods | Get internal info |
| Pushed to GitHub | Uploaded project | Submit or share |

✅ **Project Folder Now Includes:**

* deployment.yaml
* service.yaml
* All relevant screenshots
* GitHub link for submission

**✅ Kubernetes Interview Questions & Answers**

**1. What is Kubernetes?**

**Answer**:  
Kubernetes is an open-source container orchestration platform that helps manage, deploy, scale, and automate applications in containers.

📌 *Why it's useful:*  
It handles large containerized apps in production easily by taking care of scaling, healing, and load balancing.

**2. What is the role of kubelet?**

**Answer**:  
Kubelet is an agent that runs on each node in a Kubernetes cluster. It ensures the containers are running as described in the Pod specifications.

📌 *Example:*  
If a container crashes, kubelet restarts it.

**3. Explain Pods, Deployments, and Services.**

* **Pod**: The smallest unit in Kubernetes. It can contain one or more containers that share storage, network, and lifecycle.
* **Deployment**: It manages pods, allowing you to easily scale and update your application.
* **Service**: It exposes pods to the network, helping different parts of your app talk to each other.

📌 *In simple words:*  
Pod = container(s), Deployment = controller, Service = connector.

**4. How do you scale in Kubernetes?**

**Answer**:  
You can scale by increasing or decreasing the number of pod replicas in a deployment using:

bash

CopyEdit

kubectl scale deployment my-app --replicas=5

📌 *Why scale?*  
To handle more traffic or reduce unused resources.

**5. What is a namespace?**

**Answer**:  
Namespaces are used to organize resources in a Kubernetes cluster into virtual groups.

📌 *Example:*  
You can have dev, test, and prod environments separated using namespaces.

**6. Difference between ClusterIP, NodePort, LoadBalancer**

| **Type** | **Purpose** | **Access Type** |
| --- | --- | --- |
| **ClusterIP** | Default | Internal cluster only |
| **NodePort** | Opens a port on node | External access using node IP |
| **LoadBalancer** | Uses cloud provider to expose service | External access with public IP |

📌 *Use Case:*  
Use ClusterIP for internal services, NodePort for local testing, LoadBalancer for production.

**7. What are ConfigMaps?**

**Answer**:  
ConfigMaps are used to store configuration data (like key-value pairs) separately from code.

📌 *Why?*  
You can change config without touching the app code — good for flexibility.

**8. How do you perform rolling updates?**

**Answer**:  
Kubernetes performs rolling updates automatically with deployments. You just update the image:

bash

CopyEdit

kubectl set image deployment/my-app my-container=my-image:v2

📌 *Why rolling?*  
It updates pods one by one with zero downtime.