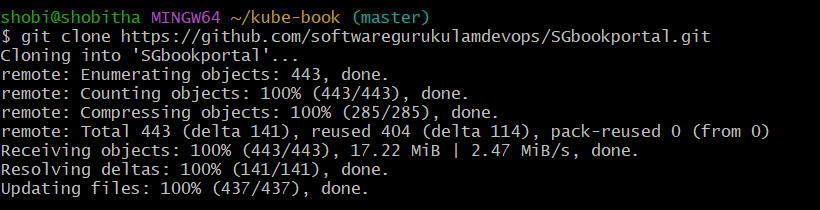
Task:

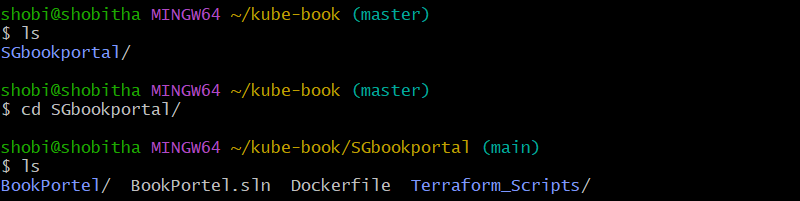
🚀 Deploying .NET Application to AKS Using Kubernetes Manifest Files .

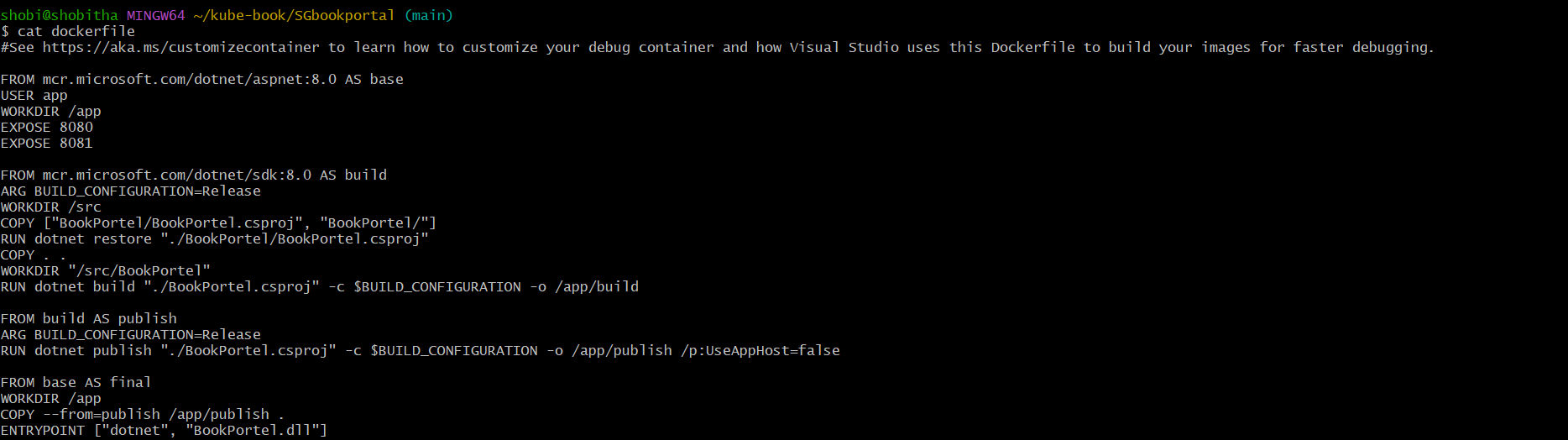
🔧 Prerequisites:

* Docker installed and running
* Kubernetes cluster (Minikube for local, or AKS/GKE/EKS for cloud)
* Docker Hub account
* kubectl and docker CLI configured
* .NET SDK installed (optional, for building locally)

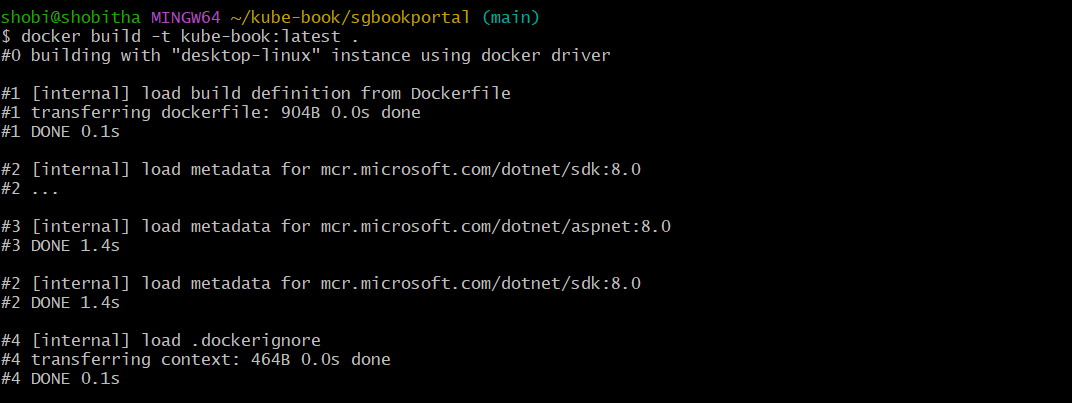
**✅ STEP 1: Clone Your .NET App to local repo**

**✅ STEP 2: Create a Dockerfile in Your Project Root**

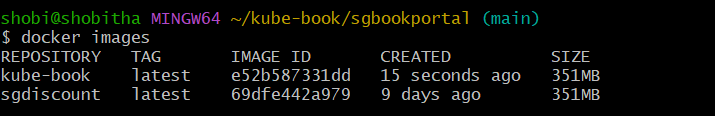




**✅ STEP 3: Build the Docker Image**



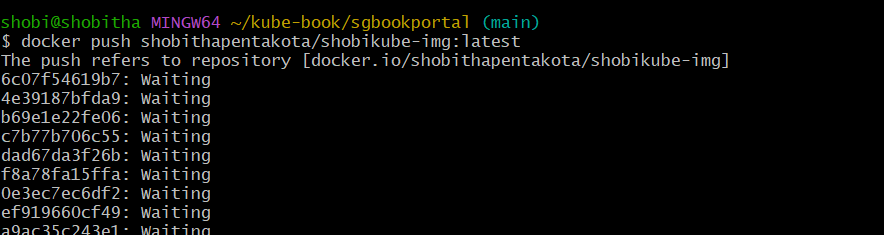
* Our image build successfully.



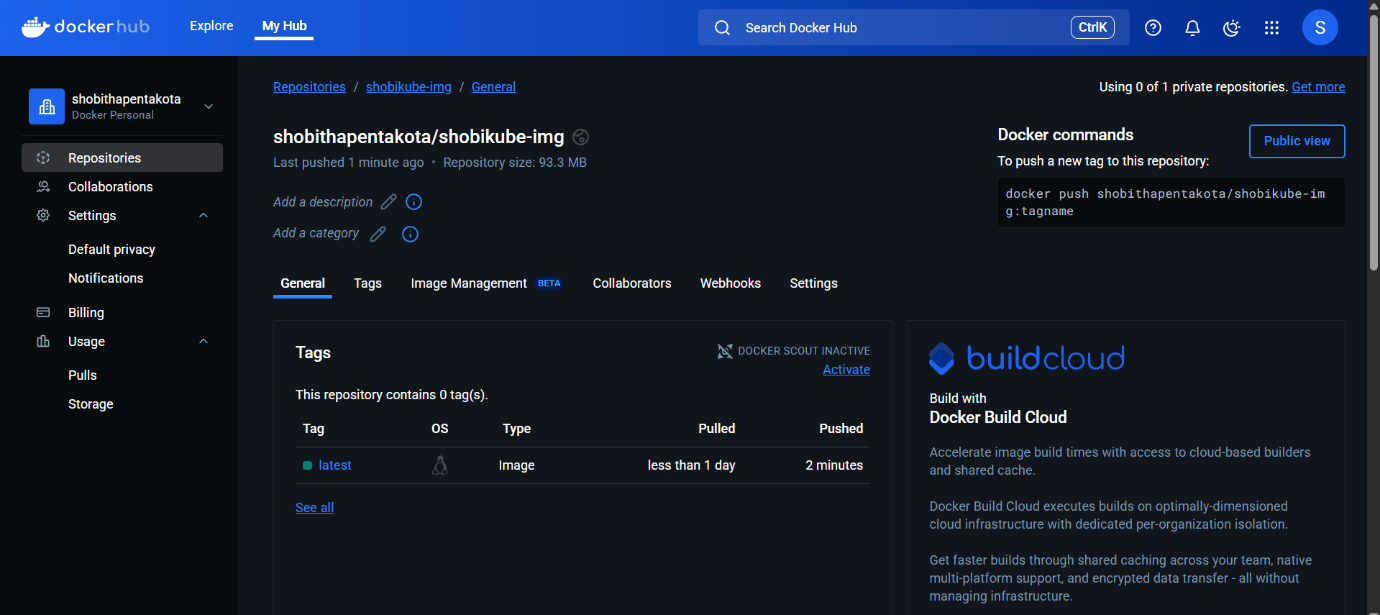
**✅ STEP 4: Push the Image to Docker Hub**

* Before going push our image to docker hub we need to create a tag.

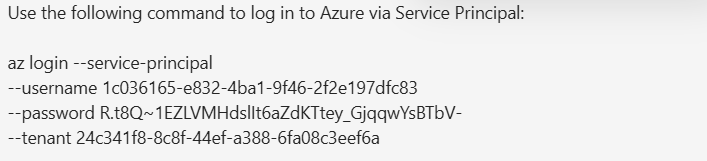


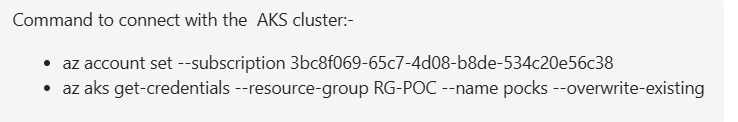


* Image pushed to docker hub…..we can see in our repo tags.



* Login into azure portal….. connect to Aks cluster.

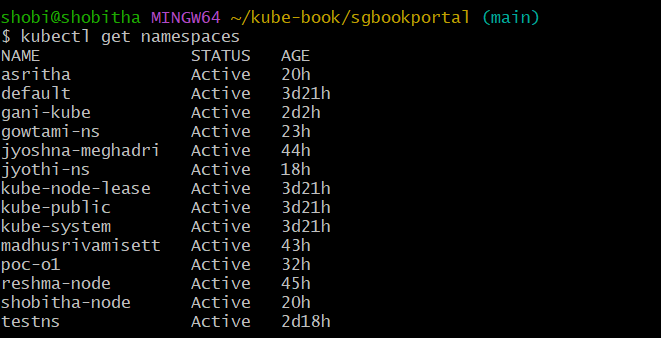




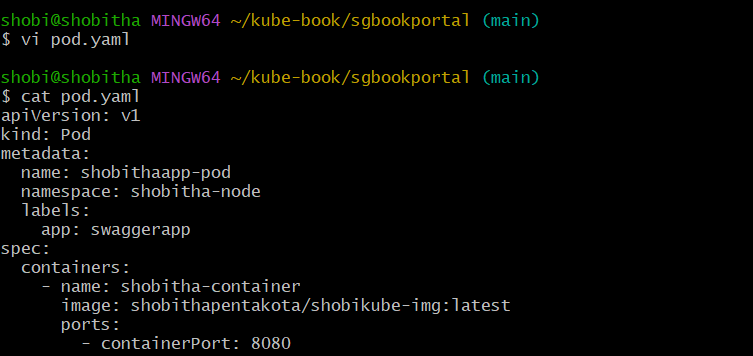
**✅ STEP 5: Write Kubernetes Manifest Files:**

**Create a Namespace**

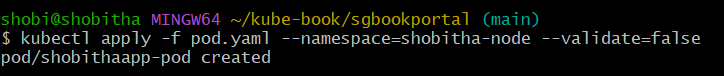
* Create a Kubernetes namespace using your name.

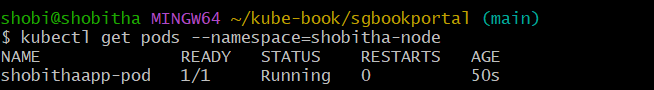


1. Pod.yaml:

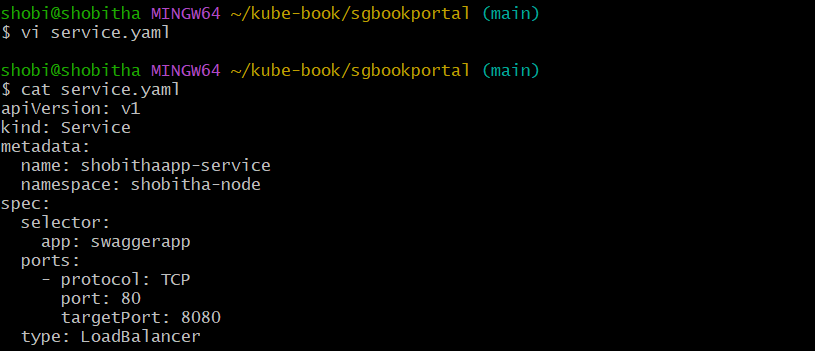


Step 6:Deploy to Kubernetes:

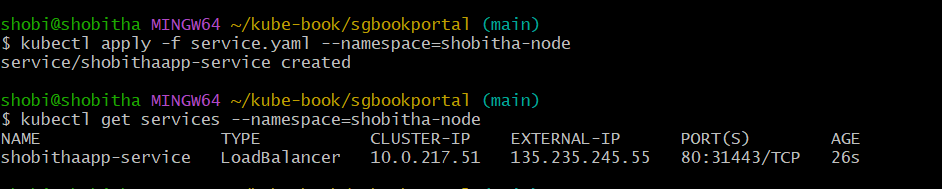




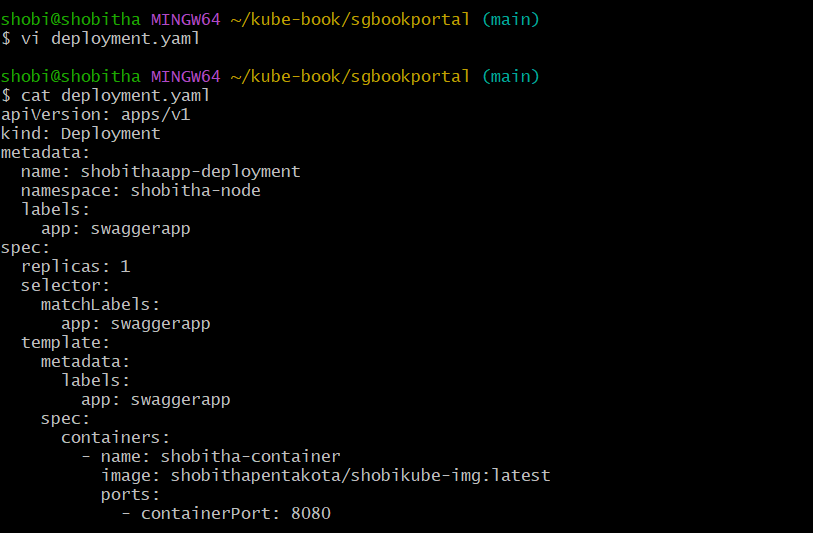
2)service.yaml:

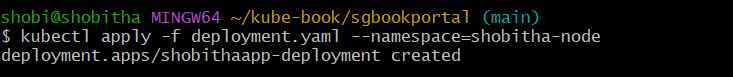


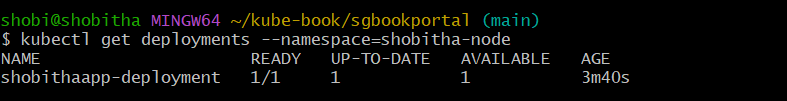
Deploy to Kubernetes:



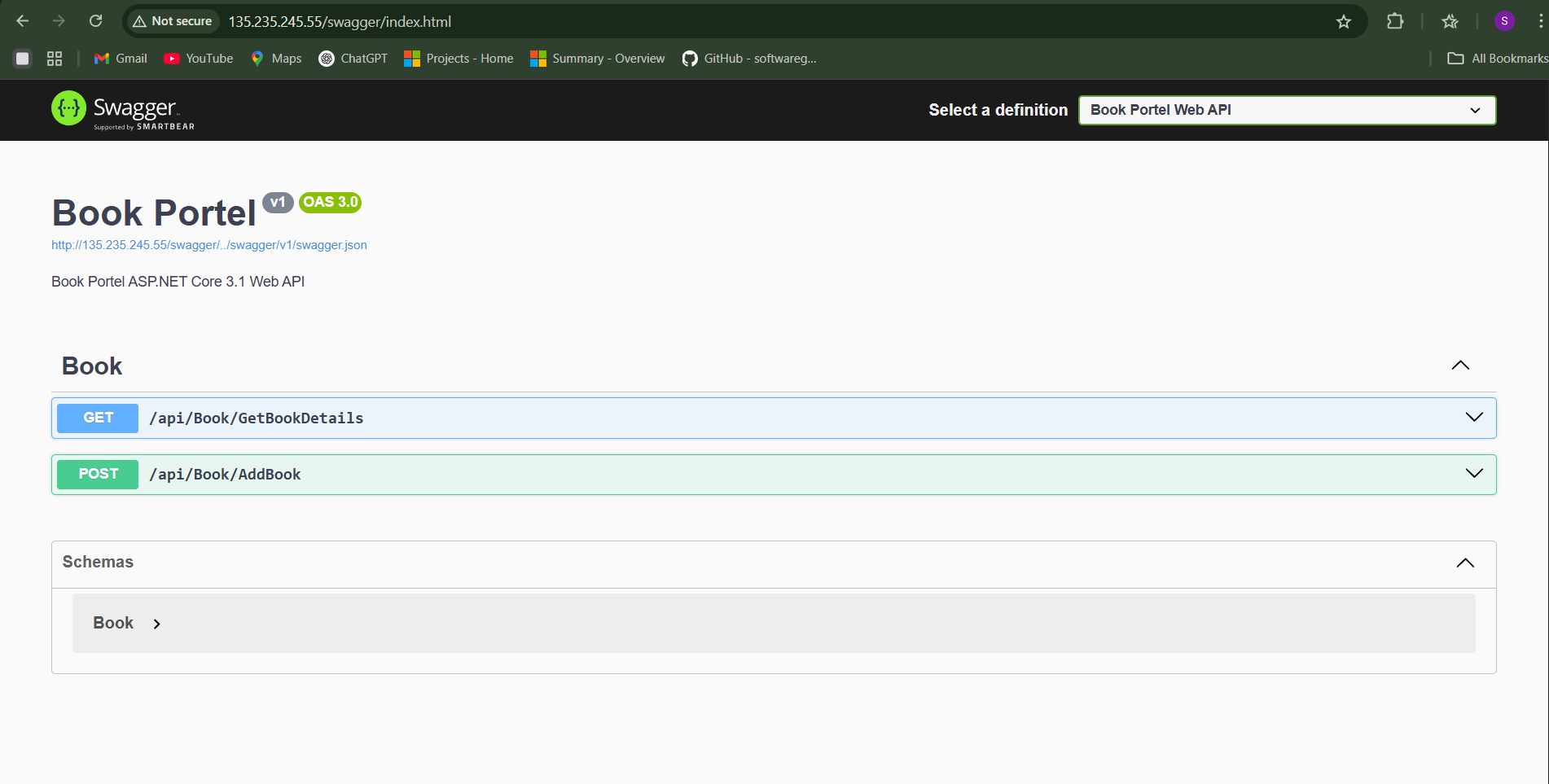
3)Deployment.yaml:



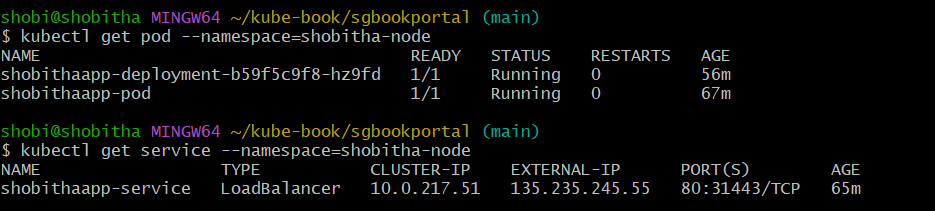


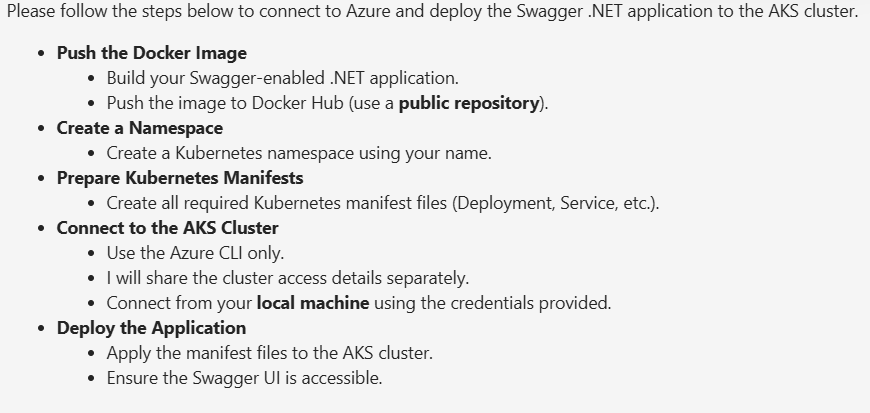


✅ STEP 7: Access the Application with Swagger:



* We successfully deploy .net application .





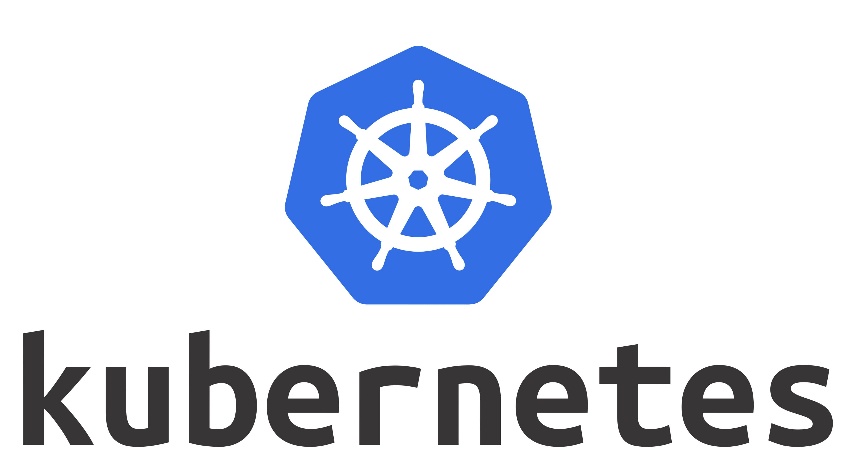
Kubernetes

**📌 What is Kubernetes?**

Kubernetes is an open-source container orchestration platform. It is used to automate the deployment, scaling, and management of containerized applications.

It handles:

* Container deployment
* Load balancing
* Auto-healing (restart crashed containers)
* Scaling (up/down based on load)
* Rolling updates and rollbacks



**📦 Why Kubernetes?**

Kubernetes helps to automate and scale the hosting of applications using container images (e.g., Docker images).

**✨ Key Features:**

✅ Developed by Google (now maintained by the Cloud Native Computing Foundation - CNCF).

✅ Automatically reschedules containers if they crash.

✅ Tracks application versions during deployment.

✅ Supports advanced deployment strategies like:

* Blue-Green Deployment
* Rolling Updates
* Canary Deployment

**🚀 Blue-Green Deployment Strategy:**

**Blue Environment**: Current stable version (e.g., v1.5)

**Green Environment**: New version to be tested (e.g., v1.6)

After validation, 100% of traffic is moved from Blue ➡ Green.

Helps avoid downtime and enables quick rollback if the new version fails.

❌ Not directly available in Docker, but ✅ supported in Kubernetes.

**📌 What is a Kubernetes Cluster?**

In Kubernetes, a cluster is the fundamental unit that represents a group of machines (nodes) working together to run containerized applications.

**🧱 Kubernetes Node Types**

**1. Master Node (Control Plane)**

Controls and manages the entire Kubernetes cluster.

Kubernetes works in a **cluster**, and a cluster has **two main types of nodes**:

| **Component** | **Role** |
| --- | --- |
| **kube-apiserver** | Entry point to the cluster; all commands and communications go through it. |
| **etcd** | Key-value store that holds the cluster state. |
| **kube-scheduler** | Decides which node should run a new pod. |
| **kube-controller-manager** | Maintains cluster state (e.g., ensuring desired replica count). |
| **cloud-controller-manager** (optional) | Integrates with cloud providers. |

**2. Worker Node**

Runs the **actual applications** (pods/containers).

| **Component** | **Role** |
| --- | --- |
| **kubelet** | Communicates with the API server and manages Pods on the node. |
| **kube-proxy** | Manages network rules and routing to ensure services can reach Pods. |
| **Container Runtime** | Starts and stops containers (e.g., containerd, CRI-O). |

**📦 What is a Pod in Kubernetes?**

A **Pod** is the **smallest deployable unit** in Kubernetes. It represents:

* One or more containers
* Shared storage and network
* A specification for how to run the containers

🧠 Think of a Pod as a wrapper around one or more containers that run together and share resources.

**📊 Pod Lifecycle**

1. **Pending** – Pod is accepted but not yet scheduled.
2. **Running** – Pod is running on a node.
3. **Succeeded/Failed** – Containers inside exited normally/with error.
4. **CrashLoopBackOff** – Repeated failures.

**☸️ Kubernetes Architecture**

Kubernetes follows a **client-server architecture** that consists of:

* A **Control Plane** (Master Node)
* Multiple **Worker Nodes**
* And internal communication using APIs

**1. Control Plane (Master Node)**

The Control Plane is the **brain** of Kubernetes. It is responsible for **managing the state** of the cluster and making decisions like scheduling and scaling.

**Key Components:**

* **kube-apiserver**:  
  Acts as the main **communication hub** for the cluster. All commands (from kubectl or other tools) go through the API server.
* **etcd**:  
  A **distributed key-value store** that stores **all cluster data** (like config, state, secrets, etc.).
* **kube-scheduler**:  
  Decides **which worker node** should run a new Pod, based on available resources and constraints.
* **kube-controller-manager**:  
  Runs controllers that maintain the **desired state** (e.g., making sure the correct number of pods are running).
* **cloud-controller-manager** *(optional)*:  
  Used when running Kubernetes on a cloud platform. Handles cloud-specific tasks like managing load balancers, storage, etc.

**🧱 2. Worker Nodes**

Worker nodes are where your **applications actually run** inside containers.

**Each worker node includes:**

* **kubelet**:  
  Talks to the control plane. Makes sure containers are running as expected on the node.
* **kube-proxy**:  
  Manages **network routing** for pods and services. Helps pods talk to each other and to the internet.
* **Container Runtime**:  
  Software that actually runs the containers. Examples: containerd, CRI-O, Docker (deprecated in newer versions).

**📦 3. Pods (Smallest Unit)**

A **Pod** is the **smallest deployable unit** in Kubernetes.  
It can contain **one or more containers** that share:

* The same IP address
* Storage (volumes)
* Network

Example: A web server container + a logging sidecar container in the same Pod.

**🔄 How It All Works (Step-by-Step)**

1. You create a deployment using a YAML file or kubectl.
2. The request goes to the **API Server**.
3. The **Scheduler** finds the best Worker Node.
4. The **Kubelet** on that node pulls the container image.
5. The container runtime runs the containers inside a **Pod**.
6. **kube-proxy** ensures the Pod is reachable by other Pods or services.

**🗺️ Kubernetes Architecture:**



🧩 **Basic kubectl Commands:**

| **Action** | **Command Example** |
| --- | --- |
| Check cluster info | kubectl cluster-info |
| View nodes | kubectl get nodes |
| View pods | kubectl get pods |
| View deployments | kubectl get deployments |
| View services | kubectl get svc |
| Create resource | kubectl apply -f filename.yaml |
| Delete resource | kubectl delete -f filename.yaml |
| Describe resource | kubectl describe pod pod-name |
| Get resource in all namespaces | kubectl get pods --all-namespaces |
| Switch context | kubectl config use-context context-name |

🛠️ **Pod Management Commands:**

| **Action** | **Command Example** |
| --- | --- |
| Run a pod | kubectl run nginx --image=nginx |
| Get logs of a pod | kubectl logs pod-name |
| Execute inside pod | kubectl exec -it pod-name -- /bin/bash |
| Delete a pod | kubectl delete pod pod-name |

🚀 **Deployment Commands :**

| **Action** | **Command Example** |
| --- | --- |
| Create deployment | kubectl create deployment my-app --image=nginx |
| Scale deployment | kubectl scale deployment my-app --replicas=3 |
| Update deployment | kubectl set image deployment/my-app nginx=nginx:1.25 |
| Rollback deployment | kubectl rollout undo deployment my-app |

|  |
| --- |
|  |

**📡 Service & Networking:**

| **Action** | **Command Example** |
| --- | --- |
| Expose pod as service | kubectl expose pod nginx --type=NodePort --port=80 |
| View services | kubectl get services |
| Port forward a pod | kubectl port-forward pod-name 8080:80 |

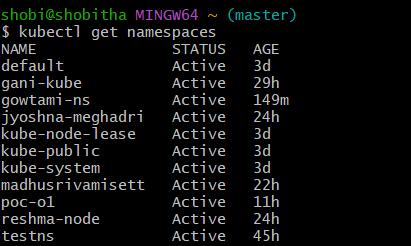
📁 **Config & Contexts:**

| **Action** | **Command Example** |
| --- | --- |
| View current context | kubectl config current-context |
| View all contexts | kubectl config get-contexts |
| Set a namespace | kubectl config set-context --current --namespace=my-namespace |

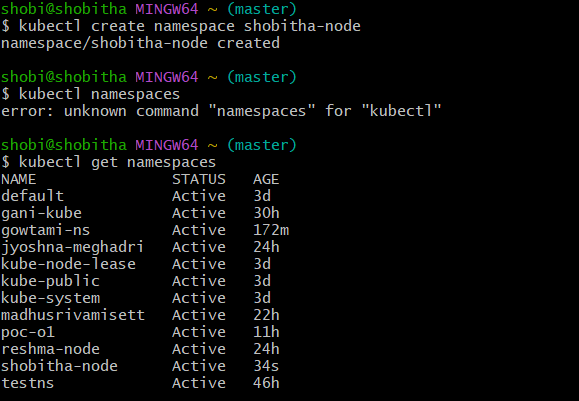
🧭 **Namespace Commands in Kubernetes:**

| **📌 Action** | **🧪 Command** |
| --- | --- |
| ✅ List all namespaces | kubectl get namespaces |
| 🔍 View resources in a namespace | kubectl get pods -n my-namespace |
| ➕ Create a namespace | kubectl create namespace my-namespace |
| ❌ Delete a namespace | kubectl delete namespace my-namespace |
| ✏️ Set default namespace for kubectl | kubectl config set-context --current --namespace=my-namespace |
| 🔍 Describe a namespace | kubectl describe namespace my-namespace |

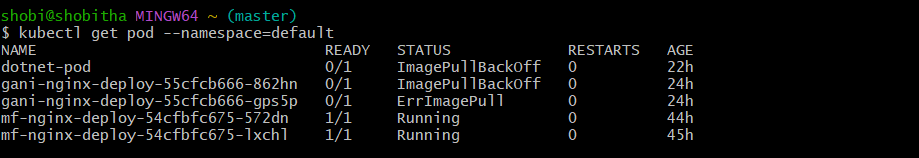
**✅ List all namespaces :**

****

**➕ Create a namespace :**

****

* **To get pods in default namespaces:**

****