**Difference Between Kind, Kubeadm, and Minikube:-**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Kind (Kubernetes in Docker)** | **Kubeadm** | **Minikube** |
| **Purpose** | Ideal for testing Kubernetes in Docker | Used for setting up production-ready Kubernetes clusters | Best for local Kubernetes development |
| **Architecture** | Runs Kubernetes inside Docker containers | Initializes a real Kubernetes cluster on physical/VM nodes | Runs a single-node Kubernetes cluster inside a VM or container |
| **Use Case** | CI/CD testing, lightweight Kubernetes development | Setting up clusters in production & testing environments | Learning and experimenting with Kubernetes locally |
| **Cluster Type** | Multi-node cluster in a single machine using containers | Multi-node cluster across multiple physical/virtual machines | Single-node cluster (by default) but can support multiple nodes |
| **Installation Complexity** | Easy – just need Docker | Medium – requires configuring networking, etc. | Easy – one command setup |
| **System Requirements** | Docker | Requires Linux/Ubuntu (or cloud VMs) | Works on Windows, macOS, and Linux |
| **Speed** | Fast (runs Kubernetes inside containers) | Medium (sets up a real cluster) | Fast (runs inside VM or container) |
| **Production Use** | No, for testing only | Yes, designed for production clusters | No, for development/testing only |
| **Networking** | Uses Docker networking | Requires configuring networking (e.g., Calico, Flannel) | Comes with built-in networking solution |
| **Load Balancer** | Needs manual setup | Supports external load balancers (e.g., MetalLB) | Has a built-in load balancer |

* **Kind**: If you need a quick and lightweight Kubernetes cluster for testing inside Docker (CI/CD, local development).
* **Kubeadm**: If you need a real, multi-node Kubernetes cluster for production or testing.
* **Minikube:** If you want a simple local Kubernetes cluster for development, learning, or experimenting.

**Ports and Protocols: -**

When running Kubernetes in an environment with strict network boundaries, such as on-premises datacenter with physical network firewalls or Virtual Networks in Public Cloud, it is useful to be aware of the ports and protocols used by Kubernetes components.

**Port Ranges:**

**Control Plane: -**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Protocol** | **Direction** | **Port Range** | **Purpose** | **Used By** |
| TCP | Inbound | 6443 | Kubernetes API server | All |
| TCP | Inbound | 2379-2380 | etcd server client API | kube-apiserver, etcd |
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 10259 | kube-scheduler | Self |
| TCP | Inbound | 10257 | kube-controller-manager | Self |

Although etcd ports are included in control plane section, you can also host your own etcd cluster externally or on custom ports.

**Worker Node: -**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Protocol** | **Direction** | **Port Range** | **Purpose** | **Used By** |
| TCP | Inbound | 10250 | Kubelet API | Self, Control plane |
| TCP | Inbound | 10256 | Kube-proxy | Self, Load balancers |
| TCP | Inbound | 30000-32767 | Node Port Services† | All |

**What is a Namespace in Kubernetes?**

A **Namespace** in Kubernetes is a virtual cluster within a Kubernetes cluster that allows you to organize and manage resources efficiently. It helps in **logically separating workloads** in a multi-team or multi-project environment.

**Why Use Namespaces?**

* **Isolation** – Different teams or projects can have separate namespaces.
* **Resource Management** – Allows setting resource quotas for different teams.
* **Access Control** – Role-Based Access Control (RBAC) can restrict user access to specific namespaces.
* **Avoid Naming Conflicts** – Different applications can have the same resource names in separate namespaces.

**When to Use Namespaces?**

* Multi-tenant clusters (different teams or projects using the same cluster).
* Organizing **development, testing, and production** environments.
* Enforcing resource limits and access control for specific teams.

**Difference between Container, Pod, and Deployment.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Container 🛢️** | **Pod 📦** | **Deployment 📜** |
| What it is | A running instance of an application | |  | | --- | | A group of containers sharing network/storage |  |  | | --- | |  | | |  | | --- | | A controller that manages multiple Pods |  |  | | --- | |  | |
| Manages | Application runtime | |  | | --- | | Containers |  |  | | --- | |  | | |  | | --- | | Pods (via Replica Sets) |  |  | | --- | |  | |
| Scalability | Not managed by K8s | |  | | --- | | Single unit, not scalable directly |  |  | | --- | |  | | |  | | --- | | Scalable with replicas |  |  | | --- | |  | |
| Self-healing | No | |  | | --- | | No, but Kubernetes replaces dead Pods |  |  | | --- | |  | | |  | | --- | | Yes, restarts failed Pods automatically |  |  | | --- | |  | |
| Networking | |  | | --- | | Own network namespace |  |  | | --- | |  | | |  | | --- | | Shared across all containers in the Pod |  |  | | --- | |  | | |  | | --- | | Managed by Kubernetes |  |  | | --- | |  | |
| Use case | |  | | --- | | Running a micro-service |  |  | | --- | |  | | |  | | --- | | Running closely coupled containers |  |  | | --- | |  | | Managing scalable applications |