## 🧠 Kubernetes Cluster — Complete Summary

### 🔹 What Is a Kubernetes Cluster?

A **Kubernetes cluster** is the environment where containerized applications are deployed, managed, and scaled. It includes:

#### 1. ****Control Plane**** (Master node):

* **API Server**: Receives commands from kubectl
* **Scheduler**: Decides which node runs which Pod
* **Controller Manager**: Runs background logic like ReplicaSets
* **etcd**: Cluster state storage (key-value store)

#### 2. ****Worker Nodes****:

* Run the **actual applications** (Pods)
* Each has:
  + kubelet: Talks to API server
  + kube-proxy: Manages networking
  + Container runtime (Docker, containerd)

### 🔹 Can You Create Multiple Clusters on One System?

✅ **Yes** — tools like minikube and kind let you create multiple clusters:

bash

CopyEdit

minikube start -p cluster1

minikube start -p cluster2

Each cluster is isolated and has its own control plane and nodes.

Switch between clusters using:

bash

CopyEdit

kubectl config get-contexts

kubectl config use-context <context-name>

### 🔹 Is One Cluster for One App?

🟡 Not necessarily.

* One cluster can run **multiple apps**.
* Each app can include:
  + Frontend (React, Angular)
  + Backend (Node, Django, Spring)
  + DB (Postgres, MongoDB)
* Use **Namespaces** to separate apps logically.
* Use **labels, services, and resource quotas** to manage isolation.

### 🔹 Where Does a Pod Run?

* The **Scheduler** decides which **worker node** the Pod runs on.
* Decision based on:
  + Available CPU/RAM
  + Node selectors or affinity rules
  + Taints/tolerations

Check where a pod is running:

bash

CopyEdit

kubectl get pods -o wide

### 🔹 Can a Cluster Be on One System?

✅ Yes — for development and testing:

* Tools: minikube, kind, MicroK8s
* Single-node cluster: Control plane + worker node run on the same machine

✅ In production:

* Multi-node clusters run on **multiple machines**, often in the cloud.

### 🔹 Common Error: "connection refused" on kubectl apply

If you see:

pgsql

CopyEdit

failed to download openapi: Get "http://localhost:8080/openapi/v2": connection refused

It means:

* The **API server isn’t reachable**.
* Most likely, the **cluster isn’t running** or kubectl is misconfigured.

Fix:

bash

CopyEdit

minikube start

kubectl apply -f pod.yaml

🧠 Can You Run Multiple Pods of the Same App on One Node?

### ✅ YES — It’s Common Practice

* Kubernetes will schedule multiple pods on a node **if there's enough CPU and RAM**
* This is **efficient resource usage**

### ⚠️ BUT in Production:

* **Avoid putting all replicas on the same node**
* If that node fails, your app goes down
* Use features like:
  + **Pod anti-affinity**
  + **Node selectors**
  + **Taints and tolerations**
  + **Multiple nodes or AZs**

## 🖥️ Can a Cluster Be on One System?

### ✅ YES — Single System Cluster (e.g., your laptop)

Tools:

* minikube
* kind
* MicroK8s

These tools simulate a cluster with both:

* Control plane
* Worker node(s)

## 🧰 How Minikube Works Internally

Minikube creates a **local Kubernetes cluster** inside a **VM or container** on your system.

### 🛠️ How it simulates a cluster:

* Starts a **control plane node**
* You can **add multiple nodes** (for testing multi-node clusters):

bash

CopyEdit

minikube start --nodes=3

This runs 3 virtualized nodes inside your system, acting like:

CopyEdit

minikube-m01 → control plane

minikube-m02 → worker

minikube-m03 → worker

You can verify with:

bash

CopyEdit

kubectl get nodes

## 📘 Kubernetes Multi-Node Networking – Summary for Revision

### 🔹 1. ****Single Node vs Multi-Node Networking****

* On a **single node**, Pods can easily communicate using internal IPs.
* On **multiple nodes**, each node might assign the **same internal IPs** to Pods → causing **IP conflicts**.

### 🔹 2. ****Kubernetes Networking Requirements****

Kubernetes expects you to ensure:

1. **Every Pod in the cluster must have a unique IP.**
2. **Pods should communicate across nodes without NAT.**
3. **Pods should be able to talk to Nodes and vice versa.**

**⚠️ Kubernetes does NOT set this up for you.**

### 🔹 3. ****Solution: CNI Plugins (Container Network Interface)****

CNI plugins help establish networking between Pods across nodes.

Popular CNI solutions:

* Flannel – Simple overlay network
* Calico – Advanced, supports routing and network policies
* WeaveNet – Used in play-with-k8s labs
* Cilium, Cisco ACI, VMware NSX-T – Cloud and enterprise setups

These plugins:

* Assign **unique Pod IP ranges** to each node
* Create a **virtual network** to route traffic between Pods across nodes

### 🔹 4. ****Kubernetes Services****

Even though Pods have IPs, they:

* Can **restart and get new IPs**
* Are **scattered across nodes**

A **Service** solves this by:

* Giving a **stable IP/DNS name**
* **Load balancing** traffic to available Pods
* Making it easy to access apps running in Pods

#### Service Types:

| Type | Purpose |
| --- | --- |
| ClusterIP | Internal-only access (default) |
| NodePort | Exposes service on node's port |
| LoadBalancer | Exposes app via cloud load balancer |
| ExternalName | Maps to external DNS |

### 🔹 5. ****How Everything Connects Together****

plaintext

CopyEdit

Client/User

↓

Service (ClusterIP/NodePort)

↓

Picks one of the Pods (using labels/selectors)

↓

Pod might be on another node

↓

CNI plugin handles routing between nodes

↓

Pod receives and responds to request

### ✅ Final Takeaways

* **Don’t fear Pods on different nodes.**
* Just create a **Service**, and Kubernetes with the **CNI plugin** will:
  + Handle networking
  + Ensure communication
  + Load balance across healthy Pods

**You build the app → Kubernetes handles the connections.** 🚀

## ****2. Scheduling and Resource Management****

### ❓ How does Kubernetes decide where to schedule a Pod?

Kubernetes uses the **kube-scheduler**, which follows this process:

1. **Filtering phase**: Eliminate nodes that can’t run the Pod (e.g., not enough CPU/memory, taints, etc.)
2. **Scoring phase**: Assign scores to eligible nodes based on things like:
   * Resource availability
   * Pod affinity/anti-affinity
   * Least loaded node
3. **Bind**: Chooses the **highest scoring node** and binds the Pod to it.

✅ Bonus: You can **influence scheduling** using:

* **NodeSelector**
* **Tolerations/Taints**
* **Affinity rules**

### ❓ Difference Between requests and limits

| Term | Meaning |
| --- | --- |
| requests | Guaranteed minimum resource (scheduler uses this) |
| limits | Maximum the container can ever use |

Example:

yaml

CopyEdit

resources:

requests:

cpu: "500m"

memory: "256Mi"

limits:

cpu: "1"

memory: "512Mi"

### ❗ What if a pod exceeds its limit?

* If it exceeds **memory limit** → it gets **killed (OOMKilled)**.
* If it exceeds **CPU limit** → it’s **throttled** (slowed down), not killed.

## ****3. Deployments and Rolling Updates****

### ❓ How do rolling updates work under the hood?

* kubectl apply -f deployment.yaml triggers the Deployment controller.
* It **creates a new ReplicaSet** for the new version.
* Gradually **increases** replicas in the new set and **decreases** old replicas.
* Controlled via:
  + maxUnavailable: how many old Pods can be killed
  + maxSurge: how many extra Pods can be created temporarily

### ❓ How to perform a ****zero-downtime deployment****?

✅ Strategies:

* Use readinessProbes to **only send traffic to ready Pods**
* Set maxUnavailable=0 so no old Pods are killed until new ones are ready
* Use RollingUpdate strategy (default for Deployments)
* Avoid breaking changes in APIs and DB migrations during rollout

### ❓ What are ****readiness**** and ****liveness**** probes?

| Probe | Purpose |
| --- | --- |
| **Readiness** | Determines **if Pod is ready to accept traffic** |
| **Liveness** | Checks **if Pod is still healthy**, else restarts it |

✅ Why important?

* Avoids sending traffic to unready or stuck containers.
* Helps Kubernetes maintain app availability.

## ****4. Debugging & Failures****

### ❓ Pod is in CrashLoopBackOff — how do you debug?

1. Check pod logs:

bash

CopyEdit

kubectl logs <pod-name>

1. Get detailed status:

bash

CopyEdit

kubectl describe pod <pod-name>

1. Exec into pod:

bash

CopyEdit

kubectl exec -it <pod-name> -- /bin/sh

1. Check resource issues (limits/OOM)

### ❓ How to troubleshoot ****network issues between pods****?

* Check if they’re in the same namespace or reachable by ClusterIP
* Use:

bash

CopyEdit

kubectl exec -it <pod> -- ping <pod-ip>

* Check Network Policies (they might be blocking traffic)
* Inspect CNI plugin logs (e.g., Calico or Flannel)
* Use nslookup, curl, or netcat inside the pod

## ****5. Advanced Concepts****

### ❓ What are ****Operators**** and ****CRDs****?

* **CRDs (Custom Resource Definitions)**: Let you define your own resource types, e.g., MySQLCluster.
* **Operators**: Controllers that **automate operations** (backup, scaling, upgrade) for CRDs.

Think of it as:  
🔧 CRD = blueprint  
🤖 Operator = controller logic managing that blueprint

### ❓ Init containers, sidecars, and volumes?

* **Init container**: Runs **before** the main container. Used for setup tasks like DB migration.
* **Sidecar**: Runs **alongside** the main container. Example: logging agent, proxy.
* **Volume**: Shared storage between containers in a Pod or for persistent data.

### ❓ Secrets and ConfigMaps?

* **Secrets**: Store sensitive data (passwords, tokens) → base64-encoded.
* **ConfigMaps**: Store app configs like env vars.

Use with:

* envFrom
* volumeMounts

✅ Use RBAC + avoid exposing secrets via logs

### ❓ ReplicaSet vs StatefulSet vs DaemonSet?

| Type | Use Case |
| --- | --- |
| ReplicaSet | Maintains a fixed # of stateless Pods |
| StatefulSet | For **stateful apps** (e.g., DBs), gives stable identity + storage |
| DaemonSet | Runs **one Pod per node** (e.g., for logging, monitoring) |

## ****6. Security and RBAC****

### ❓ How do you set up RBAC?

* RBAC = **Role-Based Access Control**
* Define:
  + **Role / ClusterRole**: Permissions
  + **RoleBinding / ClusterRoleBinding**: Who gets those permissions

Example:

yaml

CopyEdit

kind: Role

rules:

- apiGroups: [""]

resources: ["pods"]

verbs: ["get", "list"]

### ❓ PodSecurityPolicy / Admission Controllers?

* **PodSecurityPolicy (PSP)**: Legacy method to enforce security settings on Pods.
* **Admission Controllers**: Webhooks that can **mutate** or **validate** Pods on creation.

E.g., prevent privileged containers or enforce specific labels.

### ❓ Namespace isolation?

* Use **Network Policies** to restrict Pod communication across namespaces.
* Use **RBAC** to give separate access to teams for different namespaces.
* Use **ResourceQuotas** to limit how much a team can consume.