## Kubernetes Architecture Overview:

At a high level, Kubernetes follows a **Master-Worker** architecture (also called **Control Plane** and **Nodes**).

There are two main parts:

* **Control Plane (Master components)** — manages the cluster.
* **Worker Nodes** — run the actual applications (your containers).

## 1. Control Plane Components (Master) – api server , scheduler , etcd , controller manager

The **Control Plane** makes decisions about the cluster, like:

* scheduling
* monitoring
* maintaining the desired state
* responding to failures

| **Component** | **Role** |
| --- | --- |
| **kube-apiserver** | Central communication hub. **All components talk through the API Server.**   * All component inside kuberntes communicate thorugh kube api server |
| **Etcd** | Key-value database **to store all cluster data (like cluster state**, config). |
| **kube-scheduler** | Decides *which* node a new pod should run on. On which node out pod should run. – on which node our pod run |
| **kube-controller-manager** | **Runs controllers** (for handling replication, endpoints, etc.). **Keeps the cluster in the desired state.** |

## 2. Node Components (Workers)

Each **Node** (a VM or physical server) runs Pods (your apps) and has:

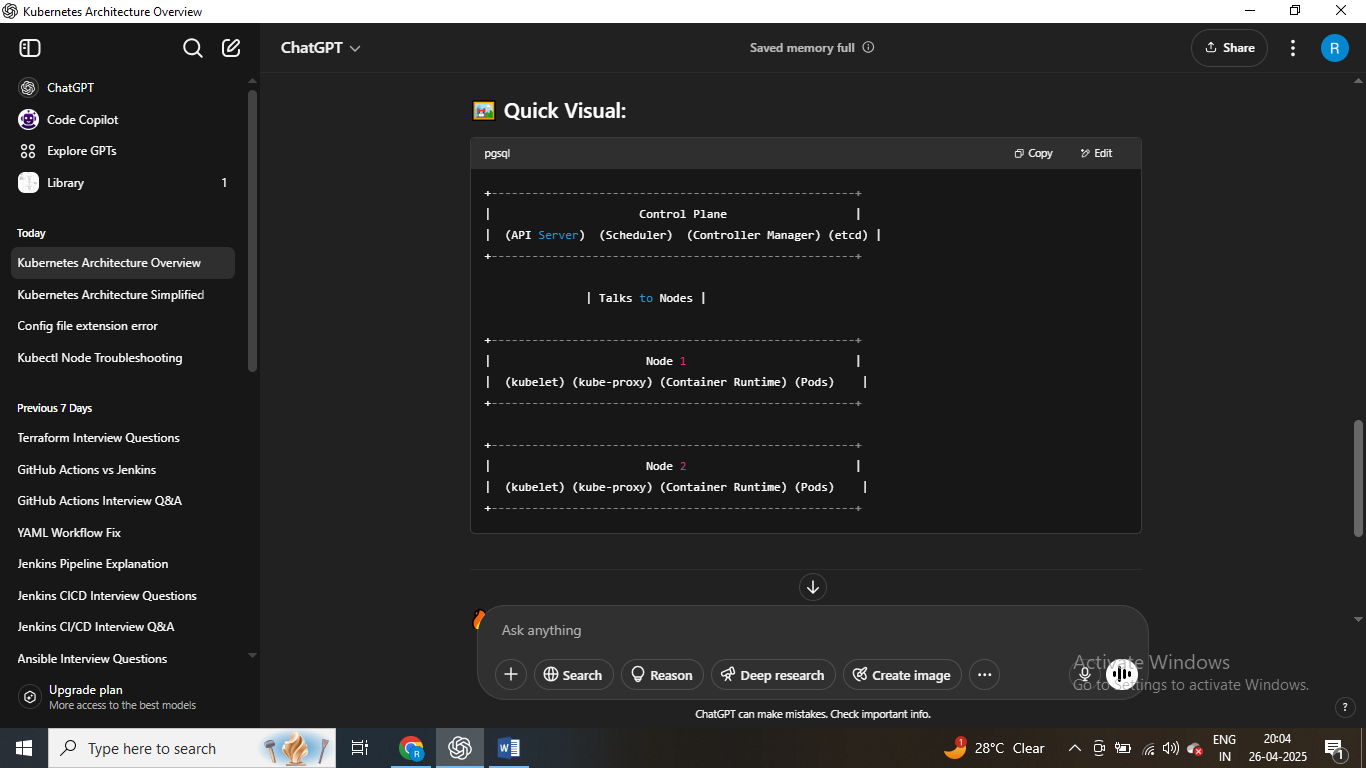
| **Component** | **Role** |
| --- | --- |
| **kubelet** | **Agent running on each node**. Talks to API Server and manages pods on the node. |
| **kube-proxy** | Manages network rules and allows communication between pods/services. |

## Additional Key Terms:

* **Pod**: Smallest deployable unit in Kubernetes. It can have one or more containers.
* **Service**: A stable network endpoint to expose your Pods.
* **Namespace**: Logical division inside the cluster (for separating teams/projects).
* **Deployment/ReplicaSet/DaemonSet**: Controllers to manage how Pods are created and managed.

## 1. 🌟 What is a Pod?

* **Pod** = the **smallest, most basic deployable unit** in Kubernetes.
* It **wraps one or more containers** (usually one) and manages their shared resources.



"**Kubernetes Control Plane manages the cluster, and the Worker Nodes run the actual applications (Pods) based on the Control Plane's instructions.**"

**Why?**  
Because sometimes **containers need to work together** tightly (like a main app + helper/sidecar). They should share the same network/storage.  
That’s why Kubernetes groups them inside a **Pod** instead of treating them individually.

A Pod has:

* **One or more Containers** (usually Docker containers, but not limited to Docker)
* **Storage Volumes** (optional shared storage)
* **A unique IP address**
* **Resource specifications** (CPU, memory requests/limits)
* **Configuration** like environment variables, secrets, etc.

## Characteristics of a Pod

| **Feature** | **Details** |
| --- | --- |
| **Single IP Address** | All containers in a Pod share the same network namespace. (localhost communication between them) |
| **Shared Storage** | All containers can mount the same volumes. |
| **Ephemeral** | Pods are **short-lived**. If a Pod dies, it is not repaired — it is **replaced** by a new one (by Deployments or ReplicaSets). |

## How is Pod Networking different?

* Each Pod gets **its own IP address**.
* Containers inside the Pod **talk on localhost** (127.0.0.1).
* Between Pods: Kubernetes services manage communication.
* **Pod-to-Pod communication** within a cluster is flat (no NAT needed).

|  |  |
| --- | --- |
| **ReplicaSet** | Ensure a fixed number of Pod replicas are running. |

|  |  |
| --- | --- |
| **DaemonSet** | Run one Pod on **each Node** (e.g., logging agent). |

|  |  |
| --- | --- |
| **StatefulSet** | Manage stateful applications with persistent IDs. |

Pods group **one or more containers together into a single unit** with shared networking and storage. **They are ephemeral** and are usually managed by controllers like Deployments. **Pods help Kubernetes treat containers more smartly** and robustly.

## What is a ReplicaSet?

* A **ReplicaSet** is a **Kubernetes controller** whose main job is:

➔ **To ensure a specified number of identical Pods are running at all times.**

If a Pod dies → ReplicaSet automatically creates a new one.  
If extra Pods appear → ReplicaSet deletes the extras.

🔵 **ReplicaSet = Self-healing Pod manager**.

## Why do we need ReplicaSet?

Because **Pods are ephemeral**:

* If a Pod crashes or the node fails, the Pod is gone.
* Kubernetes by itself won’t recreate Pods unless **something** tells it to.

**ReplicaSet** is that "something" that **continuously monitors and maintains** the desired state (desired number of Pods).

## How does ReplicaSet work internally?

* You define a **desired number of replicas**.
* ReplicaSet continuously watches the **current** vs **desired** state.
* If a Pod goes down → ReplicaSet creates a new Pod immediately.
* If too many Pods exist → It deletes the extras.

Key Features of ReplicaSet

| **Feature** | **Description** |
| --- | --- |
| **Self-Healing** | Replaces failed Pods automatically. |
| **Scalability** | Can scale Pods up or down easily. |
| **Label Selector** | Identifies which Pods belong to this ReplicaSet. |

## How is ReplicaSet different from Pod?

| **Feature** | **Pod** | **ReplicaSet** |
| --- | --- | --- |
| Responsibility | Run one instance of your app | Ensure multiple replicas |
| Recovery | No automatic recovery | **Automatic self-healing** |
| Scalability | Manual | **Automatic scaling** (changing replicas) |

## How is ReplicaSet related to Deployment?

* **ReplicaSet** manages Pods.
* **Deployment** manages ReplicaSets.

Think of it like:

"**Deployment → ReplicaSet → Pods**"

**Deployment** gives extra benefits:

* Version control (rollbacks)
* Rolling updates
* Declarative management

**Important:**  
When you create a **Deployment**, **it automatically creates a ReplicaSet** inside.

# Final 1-line Summary

"**ReplicaSet ensures that a specified number of Pods are running at any time, by monitoring and maintaining the Pods automatically. It is mostly used by Deployments in Kubernetes today.**"

| **Feature** | **ReplicaSet** | **Deployment** | **StatefulSet** |
| --- | --- | --- | --- |
| **Purpose** | **Ensure a fixed number of identical Pods** are running | Manage ReplicaSets + handle updates/rollbacks | Manage stateful applications (ordered Pods, persistent storage) |
| **Manages** | Only Pods | **ReplicaSets (and indirectly** Pods) | Stateful Pods (with unique identities) |
| **Pod Identity** | All Pods are identical | All Pods are identical | Each Pod has a **stable, unique identity** (like pod-0, pod-1) |
| **Updates/Rollbacks** | Manual (not easy) | **Easy (rolling updates, rollback to previous version)** | Controlled, ordered updates |
| **Pod Names** | Randomly generated | Randomly generated | Deterministic (fixed names) |
| **Storage** | Volatile, lost if Pod dies | Volatile, lost if Pod dies | Each Pod can have **its own persistent volume** |
| **Ordering of Pods** | No ordering | No ordering | Supports **ordered deployment, scaling, termination** |
| **Use Cases** | Simple apps needing only replication | **Apps that need versioning, updates**, and rollbacks | Databases, distributed systems (e.g., Cassandra, MongoDB) |
| **Common Example** | Simple nginx app | Rolling out new versions of frontend app | Running a database cluster like MySQL, Kafka |

 **ReplicaSet** = "I just want 5 identical Pods running."

 **Deployment** = "I want 5 identical Pods + easy upgrades/rollbacks."

 **StatefulSet** = "I want 5 Pods, each with its own ID and storage (like db-0, db-1)."

## What is a Deployment?

* A **Deployment** is a **higher-level Kubernetes controller**.
* It manages **ReplicaSets** — which in turn manage **Pods**.
* **Main goal:**  
  ➔ **Declaratively manage Pods and ReplicaSets with easy updates, rollbacks, and scaling.**

**You define the desired state**, and Deployment **automatically maintains** it.

"**Deployment = Smart manager for ReplicaSets and Pods.**"

Deployment makes it easy:

* **Rolling updates** without downtime
* **Rollbacks** if something fails
* **Automatic scaling** and management
* **Version history** maintained

## 3. 🔥 Key Features of Deployment

| **Feature** | **Description** |
| --- | --- |
| **Rolling Updates** | Update Pods gradually without downtime |
| **Rollback** | Go back to a previous working version |
| **Scaling** | Scale replicas easily |
| **Self-Healing** | Replaces unhealthy Pods automatically |

## How does Deployment work internally?

1. You define a **Deployment YAML** file (desired state).
2. Deployment creates a **ReplicaSet**.
3. ReplicaSet creates **Pods**.
4. If you **update the Deployment** (new image, new config), it:
   * Creates a **new ReplicaSet** (new version)
   * Gradually shifts traffic from old Pods to new Pods.
   * Deletes old Pods after successful rollout.
5. If something goes wrong → you can **rollback** to a previous ReplicaSet automatically.

"**Deployment manages ReplicaSets, ReplicaSets manage Pods.**"

| **Feature** | **Description** |
| --- | --- |
| **Rolling Update** | **Slowly replace old Pods with new on**es — no downtime. |
| **Rollback** | **Revert to a previous ReplicaSet version automatically** if the new one fails. |

| **Feature** | **Description** |
| --- | --- |
| **Rolling Update** | **Slowly replace old Pods with new ones** — no downtime. |
| **Rollback** | **Revert to a previous ReplicaSet version automatically** if the new one fails. |

"**Rollout strategy**" is usually **RollingUpdate** (default), but you can configure it.

yaml

CopyEdit

strategy:

type: RollingUpdate

rollingUpdate:

maxUnavailable: 1

maxSurge: 1

* **maxUnavailable**: Number of Pods that can be down during update.
* **maxSurge**: Number of extra Pods that can be created during update.

"**Deployment in Kubernetes is the smart manager for managing Pod lifecycle with rolling updates, version history, rollback support, and scaling features. It abstracts and improves over plain ReplicaSets.**"

(They LOVE asking these 🔥)

| **Question** | **Quick Answer** |
| --- | --- |
| What does a Deployment manage? | ReplicaSets, which manage Pods. |
| Can we do a rolling update with a Deployment? | Yes, by default. |
| Can we rollback a Deployment? | Yes, very easily. |
| What happens internally during an update? | A new ReplicaSet is created, old Pods are replaced gradually. |
| Difference between Deployment and ReplicaSet? | Deployment = manages updates, rollbacks, and versioning. ReplicaSet = only ensures replicas. |

# If we update anything in the YAML file (like image, env, labels) — what happens in a Deployment?

👉 **Answer:**  
**Deployment does NOT update existing Pods directly.  
Instead, it creates a NEW ReplicaSet with updated Pod templates,  
gradually creates new Pods (with updated config), and deletes old Pods.**

# Why not just update existing Pods?

Because:

* **Pods are immutable** in Kubernetes (you can’t modify running Pods for things like images, env variables, etc.).
* For safety and consistency:  
  ➔ "Replace" is better than "mutate."

# If we update anything in the YAML file (like image, env, labels) — what happens in a Deployment?

👉 **Answer:**  
**Deployment does NOT update existing Pods directly.  
Instead, it creates a NEW ReplicaSet with updated Pod templates,  
gradually creates new Pods (with updated config), and deletes old Pods.**

**Whenever a Deployment YAML is updated (changing template spec), Kubernetes creates a new ReplicaSet** with new Pods and gradually replaces the old Pods. It never updates existing Pods directly because Pods are immutable."

## What is NodeSelector?

**NodeSelector** is the **simplest way** to **tell Kubernetes to run a Pod on a specific node (or a group of nodes)** based on **labels**.

You basically say:

"Hey Kubernetes, run this Pod ONLY on a node that has label key=value."

✅ It’s a basic **scheduling constraint**.

## How does NodeSelector work?

* You add **labels** to nodes manually (or during node setup).
* In the Pod's YAML file, you use **nodeSelector** to specify which labeled node the Pod should land on.

## NodeSelector Example

### Step 1: Label a Node

bash

CopyEdit

kubectl label nodes worker-node-1 disktype=ssd

✅ Now worker-node-1 has a label disktype=ssd.

if the interviewer asks **"What if no node matches NodeSelector?"**

✅ Answer: "**Pod will stay Pending forever, Kubernetes won't reschedule automatically unless you change something manually.**"

## 1. 🌟 What is a DaemonSet?

**DaemonSet** ensures that **a copy of a specific Pod is running on ALL (or specific) Nodes** in the Kubernetes cluster.

✅ One Pod **per Node**.  
✅ Automatically adds Pod to **new Nodes**.  
✅ Automatically removes Pod from **deleted Nodes**

## Why do we need DaemonSets?

Think about stuff like:

* Log collection agents (like Fluentd, Logstash).
* Monitoring agents (like Prometheus Node Exporter, Datadog agent).
* Network management components (like CNI plugins, VPN tools).

🔵 **Goal:** Deploy these critical components to **every node** without manual work.

## How DaemonSet is different from Deployment?

| **Aspect** | **DaemonSet** | **Deployment** |
| --- | --- | --- |
| Purpose | Run one Pod **per node** | Run specified **number** of Pods (replicas) |
| Scaling | Automatically matches node count | Manual or auto-scaling (HPA) |
| Node Targeting | All or specific nodes | Any suitable node |
| Use Case | Logs, Monitoring, Storage agents | Applications, APIs, Web Servers |

## How to restrict DaemonSet to specific nodes?

Use:

* **nodeSelector** (simple matching)
* **nodeAffinity** (advanced matching)
* **Tolerations** (to tolerate tainted nodes)

## Can DaemonSet have multiple replicas per node?

❌ No!

DaemonSet is strictly designed to maintain **one Pod per Node**.  
If you want multiple Pods per node, you should **use a Deployment** instead.

"**A DaemonSet ensures that a specific Pod runs on every (or selected) node in the cluster.  
It is used mainly for cluster-wide services like logging agents, monitoring, or networking components.  
It automatically manages Pods as Nodes join or leave the cluster.  
DaemonSets maintain exactly one Pod per node, unlike Deployments.**"

| **Term** | **Meaning** |
| --- | --- |
| **DaemonSet** | **One Pod per node** |
| **Main Usage** | **Monitoring, Logging, Storage agents** |
| **Key Feature** | Auto-schedule on new nodes |
| **Compared to Deployment** | Deployment = replicas, DaemonSet = per node copy |

**Run a copy of each pod on every node** – only single pod copy

## What is Node Affinity?

**Node Affinity** is a way to **control where Pods are scheduled**, based on **rules** about node labels.

✅ It’s **an advanced and flexible version of NodeSelector**.

✅ You tell Kubernetes:  
"Schedule this Pod **preferably or mandatorily** on nodes with these label characteristics."

✅ **In short:**  
**Node Affinity** = **More powerful + more flexible scheduling**.

## Types of Node Affinity

| **Type** | **Meaning** | **requiredDuringSchedulingIgnoredDuringExecution or preferredDuringSchedulingIgnoredDuringExecution** |
| --- | --- | --- |
| **Required (Hard)** | Pod MUST be scheduled ONLY on matching nodes, or stay Pending. | requiredDuringSchedulingIgnoredDuringExecution |
| **Preferred (Soft)** | Pod SHOULD be scheduled on matching nodes if possible, but can be scheduled elsewhere too. | preferredDuringSchedulingIgnoredDuringExecution |

## Node Affinity YAML Example

### Example 1 — Hard Node Affinity

yaml

CopyEdit

spec:

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: disktype

operator: In

values:

- ssd

✅ This says:

"Pod **must** run on nodes where disktype=ssd."

Required must satify all condition

### Example 2 — Soft Node Affinity

yaml

CopyEdit

spec:

affinity:

nodeAffinity:

preferredDuringSchedulingIgnoredDuringExecution:

- weight: 1

preference:

matchExpressions:

- key: disktype

operator: In

values:

- ssd

✅ This says:

"Try to schedule on disktype=ssd nodes.  
But if none available, still run the Pod elsewhere."

While in preffer there is soft

Imagine you have:

* Some nodes labeled with gpu=true
* Others with gpu=false

🎯 You want **ML workloads** to go ONLY on gpu=true nodes.

✅ You would use requiredDuringSchedulingIgnoredDuringExecution with gpu= true!

## What happens if no node matches Node Affinity?

✅ If it’s **required**, Pod stays in Pending state.  
✅ If it’s **preferred**, Pod still gets scheduled **somewhere else** if needed.

**Node Affinity allows Pods to specify rules based on Node labels for scheduling.  
It supports both hard (required) and soft (preferred) rules using flexible match expressions.  
Compared to NodeSelector, Node Affinity is more powerful and is the recommended way for advanced node placement.**"

## What are Resource Requests and Limits?

✅ In Kubernetes, **Requests** and **Limits** are used to **control** how much **CPU** and **Memory** a **Pod/Container** can use.

| **Term** | **Meaning** |
| --- | --- |
| **Request** | The minimum amount of resource that Kubernetes guarantees to the container. |
| **Limit** | The maximum amount of resource the container is allowed to use. |

Request = gruanteee it will use till this

Limit = maximum itne use kar skate hai

 **Prevent Resource Abuse:**  
**Limits** ensure that no single Pod eats up too much CPU or Memory and **affects other Pods**.

 **Stability & Performance:**  
Helps keep the cluster **stable** and **fair** across multiple users.

## 3. 🧩 Real-World Meaning

| **Resource** | **Request** | **Limit** |
| --- | --- | --- |
| CPU | Minimum guaranteed CPU for container. | Max CPU the container can use. |
| Memory | Minimum guaranteed Memory. | Max memory container can use (if exceeded: container may be killed). |

✅ In simple words:  
**Request = Reservation**  
**Limit = Restriction**

## What happens if container uses more than Limit?

* **CPU:**  
  If the container tries to use more CPU than its **limit**, Kubernetes will **throttle** it (slow it down), **not kill** it.
* **Memory:**  
  If it tries to use more **Memory** than the **limit**, the container **will be terminated** (OOMKilled — Out of Memory Kill).

✅ So, **memory** is stricter than **CPU**.

 **CPU Units:**

* 1 CPU = 1 vCPU = 1 Core
* 500m CPU = 0.5 vCPU

 **Memory Units:**

* Common units: Mi (Mebibyte), Gi (Gibibyte)

**Requests and Limits in Kubernetes define the minimum guaranteed and maximum allowed CPU/Memory for a container.  
Requests are used during scheduling, while Limits enforce usage control during runtime.**"

## 1. 🌟 What is a Service in Kubernetes?

✅ A **Service** is a **stable network abstraction** that exposes **Pods** so that:

* Other Pods (internal)
* External users (outside cluster)

can **access them** reliably, even if Pods are **changing**, **dying**, **restarting**, etc.

🔵 **Pod IPs are not permanent**, but **Service IPs are stable**.

## Why do we need a Service?

| **Problem** | **How Service Helps** |
| --- | --- |
| Pods have **dynamic IP addresses** (change on restart). | Service gives a **fixed IP** and **DNS name**. |
| Hard to **load balance** between multiple Pods manually. | Service automatically **load balances** across Pods. |
| Need to **expose** app to outside the cluster. | Service types like **NodePort** or **LoadBalancer** expose apps outside. |

## How does a Service work internally?

✅ A Service **selects Pods** based on **labels** using a **selector**.

✅ It **forwards traffic** to healthy Pods automatically.

✅ Behind the scenes, Kubernetes sets up **iptables rules** or **IPVS** rules for routing.

## YAML Example (ClusterIP Service)

yaml

CopyEdit

apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

selector:

app: my-app

ports:

- port: 80

targetPort: 8080

✅ This creates a Service named my-service that sends traffic to Pods with label: app=my-app on their **port 8080**.  
Clients will access **port 80** of the Service.

## Types of Services (VERY IMPORTANT)

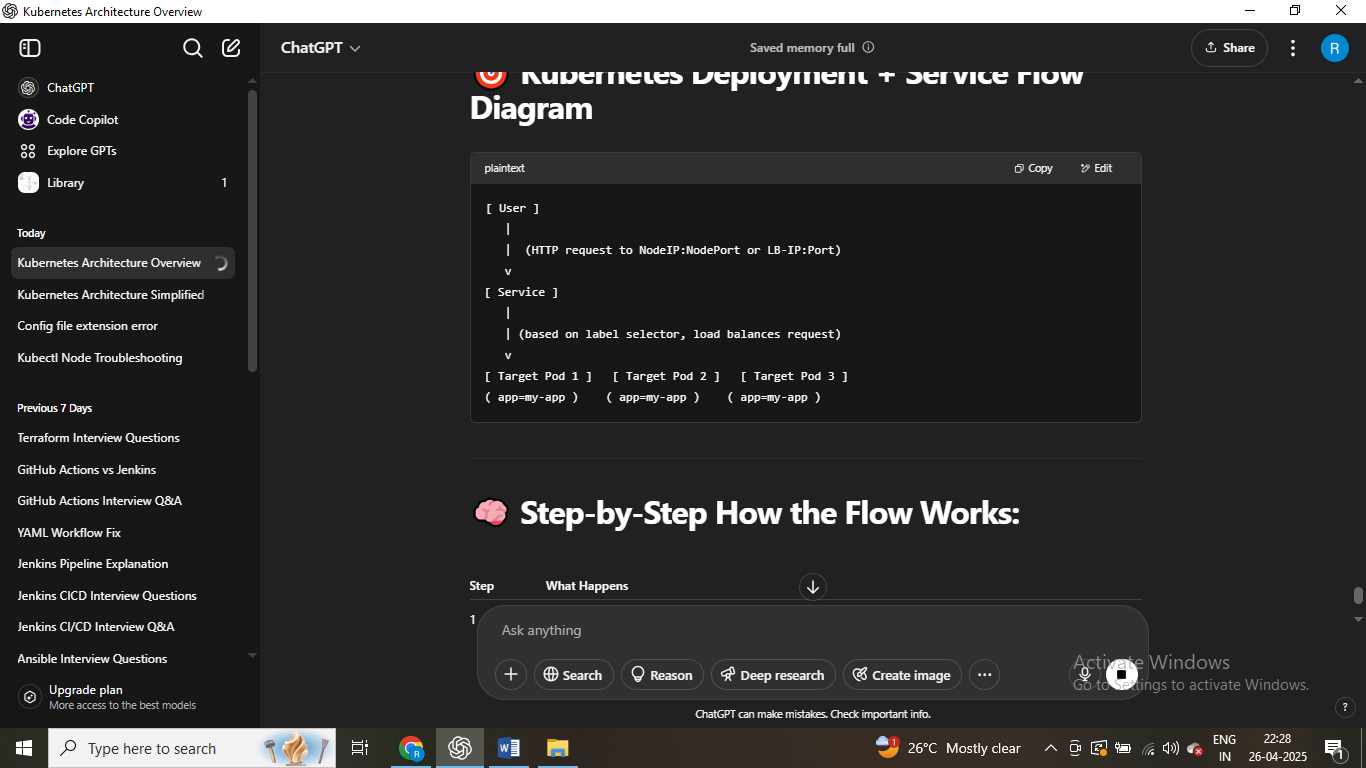
| **Type** | **Purpose** | **Accessibility** |
| --- | --- | --- |
| **ClusterIP** (default) | Expose service **inside** the cluster only. | Internal only. |
| **NodePort** | Expose service on a **port** of each **Node’s IP**. | External (limited). |
| **LoadBalancer** | Expose service through a **cloud provider’s LoadBalancer**. | External, scalable. |

## Interview HOT Questions

| **Question** | **Answer** |
| --- | --- |
| Why do we need a Service? | To provide stable access to changing Pod IPs and load balancing. |
| * **To expose a pod to service** |  |
| What happens if Pod dies? | Service automatically routes to healthy Pods matching the label. |
| Default type of Service? | ClusterIP. |
| What is NodePort Service? | Exposes service at a static port on each Node’s IP. |
| How LoadBalancer Service works? | Provisions a cloud LoadBalancer that routes to Service. |

"**A Kubernetes Service provides a stable IP address and DNS entry to access a set of Pods reliably, even if the Pods' IPs change. It supports internal communication (ClusterIP) and external access (NodePort, LoadBalancer).**"

**Headless Services** exist (by setting ClusterIP: None) when you **don't want load balancing**, but want to discover all Pod IPs directly



## What is NodePort Service?

✅ A **NodePort Service** in Kubernetes **exposes an application outside the cluster** by **opening a port** (called NodePort) **on every Node**.

* The Service listens on a specific **high port (30000–32767)** on the Node.
* When you send a request to **NodeIP:NodePort**, Kubernetes **forwards** it to the **Service**, and then to the **correct Pod**.

## Why use NodePort?

| **Reason** | **Explanation** |
| --- | --- |
| Access app externally | You can reach your app using any Node’s IP and NodePort. |
| Simple exposure | Quick and easy way without setting up a LoadBalancer. |
| Works without cloud provider | Great for bare-metal or local clusters (like Minikube, kind, k3s). |

## Important Concepts

| **Term** | **Meaning** |
| --- | --- |
| **port** | Service port inside the cluster (example: 80). |
| **targetPort** | Pod's container port (example: 8080). |
| **nodePort** | Port exposed on each Node (range 30000–32767). |

|  |  |
| --- | --- |
| What is NodePort? | It exposes a service outside the cluster by opening a specific port on each Node. |

|  |  |
| --- | --- |
| Default NodePort range? | 30000–32767. |

|  |  |
| --- | --- |
| What if you don't specify nodePort? | Kubernetes auto-assigns a random free port in the range. |

|  |  |
| --- | --- |
| Can you have multiple NodePorts? | Yes, for different services or different ports of same service. |

|  |  |
| --- | --- |
| When would you use NodePort? | When you need a simple external access without using LoadBalancer. |

## Expose your service exetranllly with a specific port without load balancer

## Real-world Example

Suppose you have 3 Kubernetes nodes:

| **Node Name** | **IP** |
| --- | --- |
| node-1 | 10.0.0.1 |
| node-2 | 10.0.0.2 |
| node-3 | 10.0.0.3 |

If your **NodePort** is 30036,  
you can hit your app using **any** Node:

* http://10.0.0.1:30036
* http://10.0.0.2:30036
* http://10.0.0.3:30036

✅ Even if the Pod is running on only one node, kube-proxy **forwards the traffic** properly.

## What is a LoadBalancer Service?

✅ A **LoadBalancer** Service **automatically provisions an external Load Balancer** (from your cloud provider — AWS, GCP, Azure)  
✅ This Load Balancer **forwards external traffic** directly to your **Kubernetes Service** inside the cluster.

**User → Cloud Load Balancer → Kubernetes Service → Pods**

✅ It makes your app reachable **from the internet** through a **stable external IP address**.

## 2. 🧠 Why use LoadBalancer?

| **Reason** | **Explanation** |
| --- | --- |
| Publicly expose services | Make your app reachable globally (internet facing). |
| Auto-provisioned Load Balancer | No manual setup needed; Kubernetes and cloud provider create it for you. |

**LoadBalancer Service provisions a public cloud Load Balancer outside the Kubernetes cluster. It forwards incoming external traffic to the Kubernetes Service, and then load-balances it across backend Pods.**"

# NodePort vs LoadBalancer — Quick Comparison Table

| **Feature** | **NodePort** | **LoadBalancer** |
| --- | --- | --- |
| External IP | No (access via NodeIP:Port) | Yes (public IP) |
| Use Case | Testing, internal access, simple clusters | Production-grade external access |
| Provisioned Load Balancer | No | Yes (from cloud) |
| Cost | Free (just node resource cost) | $$$ (Cloud LB cost applies) |
| Suitable For | Local setup, internal services | Public websites, APIs |

| **Service Type** | **One-liner Summary** |
| --- | --- |
| **ClusterIP** | "Default service for internal communication between apps inside the cluster." |
| **NodePort** | "Expose service on each Node’s IP at a static port — simple external access."   * Expose serive on static node port ip 32667 |
| **LoadBalancer** | "Provision external cloud load balancer to expose services publicly." |
| **Headless** | "Expose pods directly without load balancing — useful for StatefulSets and service discovery." |

## What is Ingress?

✅ **Ingress** is a Kubernetes **API object**  
✅ It **manages external access to services** inside a Kubernetes cluster  
✅ It **routes HTTP/HTTPS traffic** to the correct Service **based on rules** (like URL path, hostname).

| **Component** | **Meaning** |
| --- | --- |
| **Ingress** | Defines rules: which URL/host → which Service |
| **Ingress Controller** | Software that reads Ingress objects and processes the traffic (examples: nginx, Traefik) |
| **Services & Pods** | Actual backends that receive the traffic |

## Simple YAML Example

yaml

CopyEdit

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: my-ingress

spec:

rules:

- host: myapp.example.com

http:

paths:

- path: /app

pathType: Prefix

backend:

service:

name: app-service

port:

number: 80

- path: /api

pathType: Prefix

backend:

service:

name: api-service

port:

number: 80

✅ Here:

* Traffic to myapp.example.com/app goes to app-service
* Traffic to myapp.example.com/api goes to api-service

## 7. ✨ Very Important Notes (PRO Level for Interview)

* Ingress **only defines the rules**.  
  **Ingress Controller** actually implements it.
* You **must** install an Ingress Controller manually (unless your cloud does it for you).
* **pathType** (in YAML) can be:
  + Prefix: /app/\* will match /app/page1
  + Exact: must match exactly /app
* You can configure **TLS/SSL** termination at Ingress to handle HTTPS.

**Ingress provides centralized, rule-based routing to multiple Kubernetes services using one public IP, allowing fine-grained control over HTTP/HTTPS traffic inside the cluster.**"

 **NodePort** → Basic way to expose a Service on Node IP.

 **LoadBalancer** → External access through a Cloud LB.

 **Ingress** → Smart routing to multiple services with one IP.

| **Question** | **Quick Tip** |
| --- | --- |
| Does Ingress expose Services directly? | ❌ No, it routes traffic to Services. |
| Is Ingress enough by itself? | ❌ No, needs an Ingress Controller installed. |
| Can Ingress do HTTPS termination? | ✅ Yes, via TLS configuration. |
| Can Ingress route based on domain names? | ✅ Yes (host based routing). |

## 🌟 What is a Service Account?

✅ A **Service Account** is a Kubernetes object that provides an **identity** for **pods** (or processes inside pods) to **interact with the Kubernetes API Server**.

📢 **Simple Words:**  
Service Account =  
**Username/password for Pods** ➔ to **talk to Kubernetes API** ➔ securely.

✅ Example:

* Your Pod wants to **read Secrets**, **list Pods**, **update ConfigMaps** —
* Pod needs an **authentication token** —
* **Service Account** provides it!

## Default Behavior

✅ By default:

* Kubernetes creates a **default** Service Account in every namespace.
* If you don’t specify anything, your Pods use default Service Account.

| **Related Concept** | **Meaning** |
| --- | --- |

|  |  |
| --- | --- |
| **Secret** | Service Account stores a Secret (Token) to authenticate. |

|  |  |
| --- | --- |
| **Role** | Defines *what* actions are allowed (list pods, get services, etc.). |

## Real-World Example

✅ Suppose you are running a **Kubernetes Job** to create some Secrets.

* Without Service Account ➔ **Pod cannot access Kubernetes API** ➔ fails ❌
* With Service Account ➔ **Pod authenticated**, permissions granted ➔ success ✅

| **Question** | **Quick Tip** |
| --- | --- |

|  |  |
| --- | --- |
| What is a Service Account? | Identity for pods to talk to Kubernetes API securely |

|  |  |
| --- | --- |
| How is it different from User Accounts? | Service Accounts are for apps, not humans |

|  |  |
| --- | --- |
| What is the default behavior? | Every namespace has a default Service Account |

|  |  |
| --- | --- |
| How to attach a custom Service Account to a Pod? | Use serviceAccountName: in Pod spec |

|  |  |
| --- | --- |
| What else do you need along with a Service Account? | Roles and RoleBindings (for permissions) |

|  |  |
| --- | --- |
| Where is the token stored inside the pod? | /var/run/secrets/kubernetes.io/serviceaccount |

# What is Helm in Kubernetes?

✅ **Helm** is a **package manager** for Kubernetes.  
(Just like apt for Ubuntu or yum for CentOS.)

It **helps you define, install, and upgrade** complex Kubernetes applications easily.

**Simple Words:**  
Without Helm → You have to write big YAML files manually.  
With Helm → You install apps with **one command** (helm install) and manage configurations smartly.

# Key Concepts in Helm

| **Concept** | **Meaning** |
| --- | --- |
| **Chart** | A Helm package. Contains YAML templates + values. |
| **Release** | A running instance of a chart inside Kubernetes. |
| **Repository** | A place where charts are stored (like DockerHub for images). |
| **Values.yaml** | A file where you define user-specific configuration. |

# How Helm Works (Simple)

1. You create or download a **Chart**.
2. You install it using helm install.
3. Helm converts templates into real Kubernetes YAMLs and applies them.
4. Kubernetes resources (Pods, Services, Ingress, etc.) are created automatically.

# Basic Commands

| **Command** | **Purpose** |
| --- | --- |
| helm repo add | Add a Helm repository (like Bitnami charts) |
| helm search repo | Search for available charts |
| helm install <release-name> <chart-name> | Install a chart |
| helm upgrade <release-name> <chart-name> | Upgrade the release |
| helm rollback <release-name> <revision> | Roll back to previous version |
| helm uninstall <release-name> | Delete the app |

# Helm Advantages

* **Package Management:** Easy install/upgrade/delete.
* **Configuration Management:** Change values.yaml without touching templates.
* **Reuse:** Write a chart once, install it multiple times with different configs.
* **Rollbacks:** Safe undo of bad deployments.

**Helm is a package manager for Kubernetes that simplifies deployment, management, and configuration of applications through reusable charts. It offers templating, version control, and rollback features, reducing complexity and operational risk in Kubernetes environments.**"

mychart/

Chart.yaml # Metadata (name, version, description)

values.yaml # Default configuration values

charts/ # Sub-charts (dependencies)

templates/ # Kubernetes YAML templates (pod.yaml, service.yaml, etc.)

# What is a Namespace in Kubernetes?

✅ A **Namespace** is a way to **divide** a Kubernetes cluster into **multiple virtual clusters** inside the same physical cluster.

* Logical separation
* Isolation between teams, projects, or environments
* Resource grouping and management

Each folder (namespace) can have its own Pods, Services, Deployments, etc.  
They can have same names too inside different namespaces without conflict!

# Why Use Namespaces?

| **Without Namespaces** | **With Namespaces** |
| --- | --- |
| All resources mixed together | Organized resources |
| Hard to manage | Easier for multi-team projects |
| No isolation | Different apps/teams isolated |
| No resource control | Apply quotas per namespace |

# Important Default Namespaces

| **Namespace** | **Purpose** |
| --- | --- |
| default | If you don't specify any namespace, it goes here |
| kube-system | Core Kubernetes system components (kube-dns, kube-proxy) |
| kube-public | **Publicly readable data** (used very rarely) |
| kube-node-lease | Helps nodes send heartbeats **for faster failure detection** |

# Basic Commands for Namespaces

| **Command** | **Purpose** |
| --- | --- |
| kubectl get namespaces | List all namespaces |
| kubectl create namespace <name> | Create a new namespace |
| kubectl delete namespace <name> | Delete a namespace |
| kubectl get pods --namespace=<name> | See pods inside a namespace |
| kubectl config set-context --current --namespace=<name> | Set default namespace for your session |

# Real World Example

| **Team A** | **Team B** |
| --- | --- |
| Namespace: **dev-team** | Namespace: **qa-team** |
| Deployment: nginx | Deployment: nginx |
| Service: backend-service | Service: backend-service |

⏩ Both teams can have apps/services with **same names** without conflict!

# Advantages of Using Namespaces

* Multi-tenant clusters
* Environment separation (dev, staging, production)
* Easy RBAC (Role-Based Access Control) per namespace

| **Question** | **Quick Tip Answer** |
| --- | --- |
| What are namespaces in Kubernetes? | Virtual clusters inside physical cluster for isolation. |
| Why do we need namespaces? | Separation, organization, resource control. |
| What happens if you don’t specify a namespace? | Resources go into the default namespace. |
| Can two pods with the same name exist? | Yes, in different namespaces. |
| What namespaces are created by default? | default, kube-system, kube-public, kube-node-lease. |
| How do you set namespace for kubectl commands? | --namespace flag or set-context. |

# What is Authorization in Kubernetes?

✅ **Authorization** decides **"whether a user is allowed to perform an action"** after **Authentication**.

* **Authentication** = Who are you?
* **Authorization** = Are you allowed to do this?

⏩ In Kubernetes, **after identity is verified (authentication)**, Kubernetes **checks authorization** before allowing access to resources.

# How Authorization Works in Kubernetes

**Step-by-step flow:**

1. User → Sends a request (kubectl, API, dashboard, etc.)
2. Kubernetes API Server → Authenticates the user (who are you?)
3. Authorization → Checks if the user **can perform** the requested action
4. If yes → Request proceeds
5. If no → Request denied (403 Forbidden)

# Types of Authorization in Kubernetes

| **Authorization Mode** | **Meaning** | **Example** |
| --- | --- | --- |
| **RBAC** (Role-Based Access Control) | Assign permissions to users based on roles | Admin can create pods; Developer can only read pods |

# Role-Based Access Control (RBAC) in Kubernetes

RBAC controls **who can perform what action on which resource**.

| **Component** | **Meaning** |
| --- | --- |
| **Role** | Permissions (**within a namespace**) |
| **ClusterRole** | Permissions (**across entire cluster**) |
| **RoleBinding** | Assign a **Role to a user/group within a namespace** |
| **ClusterRoleBinding** | Assign a ClusterRole to user/group for the whole cluster |

| **Question** | **Tip Answer** |
| --- | --- |
| What is Authorization in Kubernetes? | It checks if a user is allowed to perform an action after authentication. |
| Name different authorization modes. | RBAC, ABAC, Webhook, Node Authorization. |
| What is RBAC? | Role-Based Access Control based on user roles and permissions. |
| What is a Role vs ClusterRole? | Role = namespace-specific; ClusterRole = cluster-wide. |
| What is RoleBinding vs ClusterRoleBinding? | RoleBinding = binds Role to user in a namespace; ClusterRoleBinding = binds ClusterRole cluster-wide. |

✅ **Tip:**

* **Role** = Namespace-specific
* **ClusterRole** = Cluster-wide

# What is a **ClusterRole**?

| **Feature** | **Description** |
| --- | --- |
| Scope | Cluster-wide (NOT limited to a namespace) |
| Defines | What **actions** can be performed on **resources** (pods, nodes, namespaces, etc.) |

# What is a **ClusterRoleBinding**?

| **Feature** | **Description** |
| --- | --- |
| Scope | Cluster-wide |
| Purpose | **Binds a ClusterRole** to a **user**, **group**, or **service account** |
| Effect | Grants the permissions of the ClusterRole **across the entire cluster** |

# Table: Role vs ClusterRole vs RoleBinding vs ClusterRoleBinding

| **Feature** | **Role** | **ClusterRole** | **RoleBinding** | **ClusterRoleBinding** |
| --- | --- | --- | --- | --- |
| Scope | Namespace | Cluster | Namespace | Cluster |
| Defines | Permissions | Permissions | Assigns Role | Assigns ClusterRole |
| Example | Read pods in dev namespace | Read pods cluster-wide | Bind Role to user | Bind ClusterRole to user |

| **Question** | **Answer** |
| --- | --- |
| What is a ClusterRole? | **A Role that grants permissions across the entire Kubernetes cluster.** |
| What is a ClusterRoleBinding? | A binding that attaches a ClusterRole to a **user/group/service account cluster-wide.** |
| Can a ClusterRole be used in a namespace? | Yes, using a RoleBinding. |
| When would you use ClusterRole vs Role? | Use ClusterRole when access is needed across all namespaces or for cluster-level resources. |
| Can RoleBinding bind ClusterRole? | Yes, but the access will be limited to the namespace where RoleBinding exists. |

**ClusterRole defines cluster-wide permissions, while ClusterRoleBinding connects users to those permissions across the whole Kubernetes cluster.**"

# Persistent Volumes (PV) and Persistent Volume Claims (PVC)

## 🚀 What is a **Persistent Volume (PV)**?

| **Feature** | **Description** |
| --- | --- |
| Type | A piece of **storage** in the cluster |
| Provisioned By | Admin OR Dynamically |
| Lifespan | **Independent of Pod lifecycle** (survives pod restart/crash) |
| Purpose | Abstract the actual storage (AWS EBS, GCP Disk, NFS, local disk) |
| Managed By | Kubernetes control plane |
| Who creates | Admins or Dynamic provisioners |

✅ **Simple line:** PV is the actual storage resource in the cluster.

## What is a **Persistent Volume Claim (PVC)**?

| **Feature** | **Description** |
| --- | --- |
| Type | A **request** for storage |
| Created By | Developer/User |
| Purpose | Request specific storage requirements (size, access mode) |
| Bound To | A Persistent Volume (PV) |
| Lifespan | Pod-independent, until explicitly deleted |

✅ **Simple line:** PVC is a request that asks Kubernetes for some storage.

# How They Work Together

1. **Admin** creates a PersistentVolume (PV)
2. **Developer** creates a PersistentVolumeClaim (PVC)
3. Kubernetes **binds** the PVC to a matching PV automatically
4. **Pod** uses the PVC to mount storage and access it

| **Question** | **Quick Answer** |
| --- | --- |
| What is PV? | Storage resource provisioned in the cluster |
| What is PVC? | A user request for storage |
| What happens when PVC size > PV size? | Binding fails |
| Can a PVC bind to multiple PVs? | No, one-to-one mapping |
| What happens when Pod is deleted? | PVC remains unless manually deleted |
| What is Reclaim Policy? | Tells what happens to PV after PVC is deleted (Retain, Delete |

**Persistent Volume (PV)** is like a storage 'offer', and **Persistent Volume Claim (PVC)** is a 'request' that binds to an offer. Pods then use the PVC to get persistent storage, even if they crash."

## What is a **ConfigMap**?

| **Feature** | **Description** |
| --- | --- |
| Purpose | To **store non-sensitive configuration data** (key-value pairs) |
| Type | Kubernetes object |
| Scope | Application Configuration (NOT secrets) |

## Why do we need ConfigMaps?

* Imagine you have an app where:
  + Database URL
  + Log level
  + Feature toggles  
    are **configurable** values.

✅ Instead of **hardcoding** them into your app, store them in a **ConfigMap**.  
This way, you can **change configuration without rebuilding the container**.

# Key Points to Remember:

| **Key Feature** | **Description** |
| --- | --- |
| Non-sensitive | Do NOT store passwords or API keys here (use Secrets) |
| Immutable | ConfigMaps can be made immutable (good for stability) |

**Secrets** are for sensitive data (encrypted), **ConfigMaps** are for normal config.

# ConfigMap vs Secret

| **Feature** | **ConfigMap** | **Secret** | |
| --- | --- | --- | --- |
| Data Type | Non-sensitive | Sensitive | |
| Encoding | Plain text | Base64-encoded | |
| Encryption | No | Yes (at rest and in etcd) | |
| Use Case | DB URLs, feature toggles, config files | Passwords, API keys, tokens | |
| **Question** | | | **Quick Answer** | |
| What is ConfigMap? | | | Object to store config data | |
| Difference between ConfigMap and Secret? | | | ConfigMap = non-sensitive, Secret = sensitive and encrypted | |
| How to inject ConfigMap to pod? | | | Environment variable / Volume | |
| Does pod automatically update when ConfigMap updates? | | | No, pod restart is needed | |
| Can ConfigMaps be immutable? | | | Yes, by setting immutable: true | |

"**ConfigMaps** store non-sensitive configuration data separately from application code and are injected into pods via environment variables or mounted volumes, enabling flexible and scalable application management."

Would you also like me to give you **Real-World Scenario Based Questions** like:

* "If you change ConfigMap, how do you update your application without downtime?"

## What is a Secret?

| **Feature** | **Description** |
| --- | --- |
| Purpose | To **store sensitive data securely** like passwords, API keys, SSH keys, tokens |
| Type | Kubernetes object |
| Scope | Sensitive configuration |
| Lifespan | Lives independently of pods |
| Usage | Injected into pods via env variables or volumes |

✅ **One line:**  
Secrets store ***sensitive information*** that you don't want to expose inside pod specifications or container images.

## Why do we need Secrets?

* Storing passwords, tokens, or keys directly in pod specs is insecure.
* Secrets ensure sensitive data is:
  + **Encoded** (at least base64)
  + **Better protected** (can be encrypted at rest in etcd)
  + **Access controlled** (using RBAC)

✅ This is safer than hardcoding values!

| **Feature** | **Description** |
| --- | --- |
| Encoded | Base64-encoded (not encrypted by default) |
| Encryption | Can be enabled (encrypt at rest using KMS or etcd encryption) |
| Access Control | Controlled by Kubernetes RBAC |
| Size Limit | ~1MB per Secret object |

# Secrets vs ConfigMaps

| **Feature** | **ConfigMap** | **Secret** | |
| --- | --- | --- | --- |
| Use Case | Non-sensitive data | Sensitive data | |
| Encoding | Plain text | Base64-encoded | |
| Encryption | No (by default) | Possible (etcd encryption) | |
| Size Limit | ~1MB | ~1MB | |
| **Question** | | | **Quick Answer** | |
| What is a Secret in Kubernetes? | | | Object to store sensitive data securely | |
| How is a Secret different from a ConfigMap? | | | Secret is encoded & can be encrypted, ConfigMap is not | |
| How are Secrets injected into Pods? | | | As environment variables or volumes | |
| Is base64 encoding secure? | | | No, it is just encoding. Enable encryption at rest for real security. | |
| How to encrypt Secrets? | | | Use encryption providers (like KMS) to encrypt etcd data at rest. | |

**Secrets** securely manage sensitive information like passwords, API keys, and tokens in Kubernetes, allowing injection into pods while ensuring protection through base64 encoding, optional encryption, and RBAC-controlled access."

Nodes **are the machines** (virtual or physical) that run your application workloads.

kubectl run nginx-c1 --image=nginx

This command **creates a new Pod named ngin**x-c1, running a **container using the Nginx image**.

**Command Explanation:**

sh

CopyEdit

kubectl exec -it nginx-c1 -- /bin/bash

This command **opens an interactive shell** inside the nginx-c1 pod, allowing you to run commands inside the running **nginx container**.

is it ubuntu shell provide kubectl exec - like linux env

###### ChatGPT said:

No, **kubectl exec -it nginx-c1 -- /bin/bash** does not necessarily provide an **Ubuntu shell** or a full Linux environment. The shell environment you get depends on the **base image** used in the container.

### ****What Shell Does**** kubectl exec ****Provide?****

* If the **container image** is **Ubuntu** or **Debian**, you get a **full Linux Bash shell**.
* If the image is **Alpine Linux** (e.g., nginx official image), it uses **sh** (BusyBox shell).
* If the image is **CentOS** or **RHEL-based**, it has **bash** available.

apiVersion: v2

kind: Pod

metadata:

  name: nginx

spec:

  containers:

  - name: nginx-container

    image: nginx

#### **What This Manifest Defines:**

* **apiVersion: v1** → Uses the core Kubernetes API version.
* **kind: Pod** → Specifies that this YAML defines a pod.
* **metadata:** → Defines the pod name as nginx.
* **spec:** → Specifies the pod's behavior.
  + **containers:** → Defines a single container inside the pod.
    - **name: nginx-container** → The name of the container.
    - **image: nginx** → Uses the nginx image from Docker Hub.

apiVersion: v1

kind: Pod

metadata:

  name: nginx-2

spec:

  containers:

  - name: nginx-container

    image: nginx

pod.yml

C:\Windows\system32>kubectl run nginx2 --image=nginx --dry-run=client

pod/nginx2 created (dry run)

**$ nginx2 pod never be created , as it just validate resoeurces , in real this pod never be created**

apiVersion: v1

kind: Pod

metadata:

  name: multi-container-pod

spec:

  containers:

  - name: nginx-container

    image: nginx

  - name: redis-container

    image: redis

C:\Windows\system32>kubectl run nginx4 --image=nginx --dry-run=client -o yaml

apiVersion: v1

kind: Pod

metadata:

creationTimestamp: null

labels:

$ **pod is not create but generate a manift file to create a pod , we are gerneatiing a mainst file thorugh a cli command**

**$ this pod is never be created in real just see manifst file as an output**

FROM busybox:latest

CMD ["ping","-c","3","google.com"]

PS E:\CKA BY zeal vora\PracticeME\section2\dockerfile> docker build -t myimage .

[+] Building 58.4s (6/6) FINISHED

docker run myimage

Dockerfile

FROM ubuntu

ENTRYPOINT [ "/bin/echo" ]

CMD [ "hello","world" ]

PS E:\CKA BY zeal vora\PracticeME\section2\sile> docker build -t sile .

[+] Building 14.0s (4/6)

First push the image to docker hub and then use the image to create a pod by pulling the image to docker hub – this Is how it works

Using cmd and arg we **can specify what things to be done when container starts**

cmd-arg.yml

apiVersion: v1

kind: Pod

metadata:

  name: new-ping-pod

spec:

  containers:

  - name: ping-container

    image: busybox

    command: ["ping"]

    args: ["-c","60","google.com"]

###### ChatGPT said:

In Kubernetes, the command and args fields are used to define the behavior of a container when it starts. Here's a breakdown of their differences:

1. **command**:
   * This field **specifies the entrypoint of the container**, essentially the program that will run inside the container when it starts.
   * It **overrides the default entrypoint (ENTRYPOINT) of the container image**.

**Example**: In your configuration:

yaml

CopyEdit

command: ["ping"]

This means the container will start with the ping command.

**args**:

* This field **provides additional arguments to the command specified in the command field** (or the default entrypoint if command is not specified).
* The args field is equivalent to passing arguments to the program when it starts. **It does not override the entrypoint but modifies how the command is executed**.

**Example**: In your configuration:

yaml

CopyEdit

args: ["-c", "60", "google.com"]

These are the arguments that are passed to the ping command. So, the ping command will ping google.com 60 times.

**To summarize:**

* command is **the actual executable that will run**
* In your specific example, the container starts the ping command with the arguments -c 60 google.com to ping Google 60 times.
* So, the final behavior would be similar to running this in a regular terminal:
* bash
* CopyEdit
* ping -c 60 google.com

PS E:\CKA BY zeal vora\PracticeME\section2> kubectl apply -f cmd-arg.yml

pod/new-ping-pod created

apiVersion: v1

kind: Pod

metadata:

  name: new-ping-pod

spec:

  containers:

  - name: ping-container

    image: busybox

    command: ["ping","-c","60","google.com"]

You are specifying the entire command along with its arguments within the command field.

### Explanation:

* The command field here is now **defining the entire command** that will be run when the container starts. It specifies:
  + The program to run: ping
  + The arguments to pass to that program: -c 60 google.com

This configuration means that Kubernetes will run the following command when the container starts:

bash

CopyEdit

ping -c 60 google.com

The ping command will attempt to send 60 ping requests to google.com.

### Key Difference:

* In this case, **the args field is not used** because the command field is already fully defining the entire command with its arguments.
* PS E:\CKA BY zeal vora\PracticeME\section2> kubectl apply -f cmd-arg.yml
* pod/new-ping-pod created
* C:\Windows\system32>kubectl logs new-ping-pod

apiVersion: v1

kind: Pod

metadata:

  name: new-ping-pod

spec:

  containers:

  - name: ping-container

    image: busybox

    command:

    - "ping"

    - "-c"

    - "30"

    - "google.com"

### Explanation of -c:

* **-c** stands for "count" and is **used to limit the number of ping requests** sent.
* PS E:\CKA BY zeal vora\PracticeME\section2> kubectl apply -f cmd-clarity.yml
* pod/new-ping-pod created
* PS E:\CKA BY zeal vora\PracticeME\section2> kubectl get pods
* NAME READY STATUS RESTARTS AGE

C:\Windows\system32>kubectl run custom-pods --image=busybox --dry-run=client -o yaml --command -- echo "Hello World!"

C:\Windows\system32>docker run -d --name=nginx2 -p 80:80 nginx

271979be1a9775f3ecf6fb8870ca44cbf39ae8fb9e20cf4f01328b284426844e

# 80 port on left side is host port ,anyone accessing on host port 80 it will be redirect to nginx port 80 on right side automaiclly on nginx container

### Explanation of -c:

* **-c** stands for "count" and is used to limit the number of ping requests sent.
* **# what if don’t expoese port**
* **Then anyone don’t idnefty who can access or now where is our appn run**
* We can expose the port of out conatienr in our kuberntes

pod-expose.yml

apiVersion: v1

kind: Pod

metadata:

  name: nginx-3

spec:

  containers:

  - name: nginx-container

    image: nginx

    ports:

    - containerPort: 8080

**pods themselves are not directly exposed to the outside world**. You typically **expose a pod's port using a Service.**

### Explanation:

* **apiVersion: v1**: Specifies the version of the Kubernetes API to use.
* **kind: Pod**: Specifies that the resource is a Pod.
* **metadata.name: nginx-3**: The name of the pod is nginx-3.
* **spec.containers**: Defines the containers inside the pod.
  + **name: nginx-container**: The container's name is nginx-container.
  + **image: nginx**: The container uses the nginx image from Docker Hub.
  + **ports: containerPort: 8080**: Exposes port 8080 inside the container.
* PS E:\CKA BY zeal vora\PracticeME\section2> kubectl apply -f pod-expose.yml
* pod/nginx-3 created
* PS E:\CKA BY zeal vora\PracticeME\section2> kubectl get pods
* kubectl describe pods nginx-3

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: nginx-service**

**spec:**

**selector:**

**app: nginx-3**

**ports:**

**- protocol: TCP**

**port: 80 # Port on which the service will be exposed**

**targetPort: 8080 # Port inside the container**

**type: LoadBalancer # If you want to expose it externally, like in cloud environments**

### Explanation of Service YAML:

* **apiVersion: v1**: Specifies the version of the Kubernetes API for the service.
* **kind: Service**: Specifies that the resource is a Service.
* **metadata.name: nginx-service**: The name of the Service is nginx-service.
* **selector**: This matches the label of the Pod you want to expose. It should match the label assigned to the Pod.
* **ports**:
  + **port: 80**: The **port on which the service will be exposed**.
  + **targetPort: 8080**: The **port inside the container to forward traffic** to.
* **type: LoadBalancer**: If you're in a cloud environment, setting the service type to LoadBalancer will provision an external load balancer for you. If you're using a local environment like Minikube or Docker Desktop, you may use NodePort instead.

apiVersion: v1

kind: Pod

metadata:

  name: nginx-3

spec:

  containers:

  - name: nginx-container

    image: nginx

    ports:

    - containerPort: 8080

---

apiVersion: v1

kind: Service

metadata:

  name: nginx-service

spec:

  selector:

    app: nginx-3

  ports:

    - protocol: TCP

      port: 80

      targetPort: 8080

  type: LoadBalancer

**Lablels and selectors**

**Used to identify information reosurces in aws ,** like instance name through we can access to it is known as labels , using label **we can specify the env using pods like which one and what is doing** , like dev and production env and using selector we can select the env

Pod1 - env:dev

Pod2 – env:dev using label specify the env

Selector will look **into the labels which are associated with the objects a**nd all the objects which do

not have the environment of prod.

Will be removed from the list.

This is very similar here **where you have a pod and you have a label** where N is equal to dev.

So that is what labels are.

And **through selectors you will be able to filter the Kubernetes objects depending on the labels which**

**are associated with them.**

So in one of the organizations where I had recently joined what they used to do, they never had any

of the tags or any of the labels which are associated with the servers there.

So it was very difficult to find which server belonged to which environment.

And in one of the email thread, one of the developer requested the team member to go ahead and delete

the development EC2 instance and what the team member did.

He deleted the EC2 instance which belonged to the staging environment and all of the QA team came running

at the DevOps team asking why did they delete the staging environment?

And this was one of the reason, the reason why this mistake happened was because there was no label

which was associated with the resources in AWS.

And this is the reason why having labels is extremely important not only for the Kubernetes objects,

but also for the non Kubernetes environments like AWS.

#### 1. **Labels**

Labels are key-value pairs that are attached to Kubernetes objects (such as Pods, Services, Deployments, etc.) and are used to identify and group objects.

Labels can be used to identify a set of Pods or other resources.

metadata:

labels:

app: nginx

tier: frontend

n this example:

* app: nginx is a label.
* tier: frontend is another label. These labels can be applied to a Pod, a Deployment, or any Kubernetes object.

#### 2. **Selectors**

Selectors are used to identify a group of resources (like Pods) that match certain labels. When you create a Kubernetes object (e.g., a Service), you can define a **label selector** to determine which Pods should be targeted by that object.

Selectors are used primarily in:

* **Services**: To select the Pods the Service should route traffic to.
* **Deployments**: To associate a Deployment with its Pods.
* C:\Windows\system32>kubectl run pod-1 --image=nginx
* pod/pod-1 created
* C:\Windows\system32>kubectl get pods --show=labels
* **# Attaching label to pods in different pod**
* C:\Windows\system32>kubectl label pod pod-1 env=dev
* pod/pod-1 labeled
* C:\Windows\system32>
* C:\Windows\system32>kubectl label pod pod-2 env=stage
* pod/pod-2 labeled
* C:\Windows\system32>kubectl get pods --show-labels
* NAME READY STATUS RESTARTS AGE LABELS
* pod-1 1/1 Running 0 5m42s env=dev,run=pod-1
* **#selector select a pods based on specify label or filter the data**
* C:\Windows\system32>kubectl get pods -l env=dev
* NAME READY STATUS RESTARTS AGE
* pod-1 1/1 Running 0 10m
* # get pods whose env is dev
* C:\Windows\system32>kubectl run nginx --image=nginx --dry-run=client -o yaml > label

  C:\Windows\system32>notepad label-pod.yaml

apiVersion: v1

kind: Pod

metadata:

  creationTimestamp: null

  labels:

    run: nginx

  name: nginx

spec:

  containers:

  - image: nginx

    name: nginx

    resources: {}

  dnsPolicy: ClusterFirst

  restartPolicy: Always

status: {}

label-pod.yaml

C:\Windows\system32>kubectl apply -f label-pod.yaml

pod/nginx created

**$ replica set ensure right no of pods are always running , if anyone pod delete by mistake then replica set automraticcly create another pods**

apiVersion: apps/v1

kind: ReplicaSet

metadata:

  name: frontend-replicaset

spec:

  replicas: 3

  selector:

    matchLabels:

      tier: frontend

  template:

    metadata:

      labels:

        tier: frontend

    spec:

      containers:

      - name: php-redis

        image: us-docker.pkg.dev/google-samples/containers/gke/gb-frontend:v5

**$labels are associated with each of the pods , pod create by replica set will have label tier frontend**

**Label used to identify objects**

**Selector:**

A selector is used by the ReplicaSet to **identify which Pods it should manage**.

### ****How They Work Together:****

* The **ReplicaSet** creates Pods with the label tier: frontend (defined in template.metadata.labels).
* The **selector** (matchLabels: tier=frontend) ensures the ReplicaSet manages only these Pods.
* If a Pod with this label is deleted, the ReplicaSet will automatically create a new one to maintain the **desired replica count**.
* PS D:\CKA BY zeal vora\PracticeME\section3> kubectl apply -f replica-set.yaml
* replicaset.apps/frontend-replicaset created
* C:\Windows\system32>kubectl get replicaset
* NAME DESIRED CURRENT READY AGE
* frontend-replicaset 3 3 0 48s
* kubectl get pods
* C:\Windows\system32>kubectl scale --replicas=5 rs/frontend-replicaset
* replicaset.apps/frontend-replicaset scaled
* C:\Windows\system32>kubectl get replicaset
* NAME DESIRED CURRENT READY AGE
* frontend-replicaset 5 5 5 11m
* kubectl scale --replicas=1 rs/frontend-replicaset
* replicaset.apps/frontend-replicaset scaled
* now it ensure that one pod is running eveytime if we delete this one pod then it automatically crate another pod

apiVersion: apps/v1

kind: ReplicaSet

metadata:

  name: webserver-replicaset

spec:

  replicas: 3

  selector:

    matchLabels:

      app: webserver

  template:

    metadata:

      labels:

        app: webserver

    spec:

      containers:

      - name: nginx-container

        image: nginx

rs.yaml

PS D:\CKA BY zeal vora\PracticeME\section3> kubectl apply -f rs.yaml

replicaset.apps/webserver-replicaset created

**# replica set not hande regurlar update and cant handle daily challenges**

apiVersion: apps/v1

kind: ReplicaSet

metadata:

  name: webserver-replicaset

spec:

  replicas: 3

  selector:

    matchLabels:

      app: webserver

  template:

    metadata:

      labels:

        app: webserver

    spec:

      containers:

      - name: nginx-container

        image: httpd

PS D:\CKA BY zeal vora\PracticeME\section3> kubectl apply -f rs.yaml

>kubectl describe pod webserver-replicaset-687nr

kubectl delete pods webserver-replicaset-7tdrt

C:\Windows\system32>kubectl scale rs/webserver-replicaset --replicas=0

replicaset.apps/webserver-replicaset scaled

# we only need 0 pod , we don’t need 3 pod

C:\Windows\system32>kubectl scale rs/frontend-replicaset --replicas=0

replicaset.apps/frontend-replicaset scaled

**# replica set use selector to specify what pods it need to manage and selector use labels to identify which pod it should manage**

These label assocaieted with pod spec that;w why imp

apiVersion: apps/v1

kind: ReplicaSet

metadata:

  name: webserver-replicaset

spec:

  replicas: 3

  selector:

    matchLabels:

      app: webserver

  template:

    metadata:

      labels:

        app: webserver

    spec:

      containers:

      - name: nginx-container

        image: httpd

kubectl apply -f rs.yaml

replicaset.apps/webserver-replicaset created

C:\Windows\system32>kubectl run extrnal-pod --image=nginx

pod/extrnal-pod created

C:\Windows\system32>kubectl label pod extrnal-pod app=webserver

pod/extrnal-pod labeled

C:\Windows\system32>kubectl run extrnal-pod --image=nginx

pod/extrnal-pod created

C:\Windows\system32>kubectl label pod extrnal-pod app=webserver

pod/extrnal-pod labeled

C:\Windows\system32>kubectl get pods --show-labels

NAME READY STATUS RESTARTS AGE LABELS

extrnal-pod 1/1 Running 0 63s app=webserver,run=extrnal-pod

C:\Windows\system32>kubectl get pods

NAME READY STATUS RESTARTS AGE

extrnal-pod 1/1 Running 0 2m6s

webserver-replicaset-dwshn 1/1 Running 0 26s

webserver-replicaset-jx29b 1/1 Running 0 26s

**# replica set start managing external pod as it have same label like webserver replicaset**

**#those pod which has same label so replica set manage them**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 1

  selector:

    matchLabels:

      app: nginx  # Fixed the typo

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx

        image: nginx:latest

> kubectl apply -f deployment.yaml

deployment.apps/nginx-deployment created

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 1

  selector:

    matchLabels:

      app: nginx  # Fixed the typo

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx

        image: httpd:latest

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 1

  selector:

    matchLabels:

      app: nginx  # Fixed the typo

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx

        image: httpd:latest

deploy.yaml

kubectl apply –f deployment.yaml

C:\Windows\system32>kubectl rollout history deployment/nginx-deployment

deployment.apps/nginx-deployment

REVISION CHANGE-CAUSE

1 <none>

2 <none>

3 <none>

**# revision or changes 3 reviesorns are there , using deployemtn if we do any change in manifest file then these changes are reflected in pods as well**

Deployement create a replica set and pod and everything

**# we can easily rollback to previous changes at any time**

>kubectl create deployemtn nginx-deployment --image=nginx

C:\Windows\system32>kubectl get deployment

NAME READY UP-TO-DATE AVAILABLE AGE

nginx-deployemnt 1/1 1 1 2m17s

C:\Windows\system32>kubectl get rs

NAME DESIRED CURRENT READY AGE

frontend-replicaset 0 0 0 23h

nginx-deployemnt-86f574f7f5 1 1 1 2m22s

C:\Windows\system32>kubectl get pods

NAME READY STATUS RESTARTS AGE

s

C:\Windows\system32>kubectl rollout history deployment nginx-deployemnt

deployment.apps/nginx-deployemnt

REVISION CHANGE-CAUSE

1. <none>

C:\Windows\system32>kubectl get rs

NAME DESIRED CURRENT READY AGE

frontend-replicaset 0 0 0 23h

nginx-deployemnt-86f574f7f5 0 0 0 15m

nginx-deployemnt-fc76dc788 1 1 1 2m9s

C:\Windows\system32>kubectl describe rs

C:\Windows\system32>kubectl describe rs nginx-deployemnt-86f574f7f5

C:\Windows\system32>kubectl rollout undo deployment nginx-deployemnt

deployment.apps/nginx-deployemnt rolled back

C:\Windows\system32>kubectl scale --replicas=3 deployment nginx-deployemnt

deployment.apps/nginx-deployemnt scaled

**# our pod run inside node and node is run inside docker desktop**

**#one weorker node is running - maybe docker desktop**

**# pod run insidie worker nodes**

### 1. ****ReplicaSet****

A **ReplicaSet** is a Kubernetes controller that ensures a specified number of identical Pods are running at any given time. Its primary goal is to maintain the desired number of replicas of a Pod, even when Pods fail or are deleted.

* **Purpose**: The primary purpose of a ReplicaSet is to ensure that a certain number of identical Pods are running at all times.
* **Direct Control of Pods**: It directly manages Pods and ensures they are kept running in the desired state.
* **Scaling**: You can scale the number of Pods by modifying the replicas field in the ReplicaSet.

### Key Features of ReplicaSet:

* **Self-healing**: If a Pod managed by a ReplicaSet fails or is deleted, the ReplicaSet automatically creates a new Pod to replace it.
* **Scaling**: You can change the number of replicas to scale the application up or down.
* **No Rollouts**: It does not manage rolling updates or versioned deployments. When you need to update the Pods, you would need to manually manage the ReplicaSet.

### 2. ****Deployment****

A **Deployment** is a higher-level abstraction that manages ReplicaSets and provides additional functionality like rolling updates, rollbacks, and version management. It is generally recommended to use **Deployments** rather than **ReplicaSets** directly, because Deployments provide a more flexible and powerful way to manage the lifecycle of applications in Kubernetes.

* **Purpose**: The primary purpose of a Deployment is to manage the lifecycle of applications, including creating, scaling, and updating Pods over time.
* **Indirect Control of Pods**: A Deployment uses a ReplicaSet internally to manage the Pods.
* **Rolling Updates**: A Deployment allows you to perform rolling updates to update Pods incrementally, minimizing downtime.
* **Rollbacks**: Deployments support rolling back to a previous version if something goes wrong during the update process.

**Node selector** –

node for **specific pod to run on which node** we can choose approcaite work node to run our pods

Using node selector

On which node we want to run our pods we can use node selector for it

node selector

###### ChatGPT said:

In Kubernetes, a **Node Selector** is a way to constrain which nodes a Pod can be scheduled on, based on labels attached to the nodes. By using a **Node Selector**, you can ensure that your Pods are scheduled only on nodes that meet certain criteria (i.e., have specific labels). This allows you to control where your workloads run within your cluster, which can be useful for scenarios like resource optimization,

**How Node Selectors Work**

A **Node Selector** uses the labels of nodes to determine where a Pod can be scheduled. T

**basic Structure:**

The nodeSelector is specified in the **Pod spec** and matches the labels on the nodes. If the node doesn't have the labels specified in the nodeSelector, the Pod will not be scheduled on that node.

### Use Cases for Node Selectors:

1. **Resource Optimization**: If certain nodes have specific resources like GPUs, SSDs, or other hardware, you can schedule Pods requiring those resources only on nodes with the appropriate labels.
2. **Workload Isolation**: You can isolate specific workloads to run on certain nodes by labeling the nodes (e.g., labeling a set of nodes for database workloads, and using a nodeSelector to ensure that Pods requiring those resources are scheduled only on those nodes).

C:\Windows\system32>kubectl create deployment tmp-deploy --image=nginx --replicas=3

deployment.apps/tmp-deploy created

kubectl get pods -o wide

C:\Windows\system32>kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

tmp-deploy-568c7959d6-4nsxv 1/1 Running 0 15s 10.1.0.50 docker-desktop <none> <none>

**$ these pods are running in Docker-desktop which is our worker-nodes ,which pod run in which worker node we can use node selector for it**

apiVersion: v1

kind: Pod

metadata:

  name: kplabs-pod

spec:

  nodeSelector:

    size: Large

  containers:

  - name: nginx-pod

    image: nginx

nodeSelector-pod.yaml

**pod goes in worker node whose size is medium here**

apiVersion: v1

kind: Pod

metadata:

  name: nginx-deployment

spec:

  nodeSelector:

    size: medium

  containers:

  - name: nginx-pod

    image: nginx

nodeSelector-deployment.yaml

C:\Windows\system32>kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

kplabs-pod 0/1 Pending 0 4m47s <none> <none> <none> <none>

**# we don’t have any worker nodes whose size is large always**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment

spec:

  replicas: 3

  selector:

    matchLabels:

      app: nginx

  template:

    metadata:

      labels:

        app: nginx

    spec:

      nodeSelector:

        size: medium

      containers:

      - name: nginx-pod

        image: nginx

node-selector-pod.yaml

**# we will place the pod in worker node whose size is large**

**# place in worker node whose size is large , place pod in worker node**

C:\Windows\system32>kubectl get nodes

NAME STATUS ROLES AGE VERSION

docker-desktop Ready control-plane 28h v1.31.4

C:\Windows\system32>kubectl get pods -o wide

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

kplabs-pod 0/1 Pending 0 17s <none> <none> <none> <none>

C:\Windows\system32>kubectl describe pod kplabs-pod

# pod don’t match with the selector large that we have , size == large we have

C:\Windows\system32>kubectl label node docker-desktop size=large

node/docker-desktop labeled

**$after labelling worker node now pod is in running state**

C:\Windows\system32>kubectl describe node docker-desktop

**$ specify the label with our nodes**

PS D:\CKA BY zeal vora\PracticeME\section3> kubectl delete -f nodeSelector-pod.yaml

pod "kplabs-pod" deleted

C:\Windows\system32>kubectl label node docker-desktop size=large

node/docker-desktop labeled

kubectl describe node docker-desktop

kubectl get nodes --show-labels

apiVersion: v1

kind: Pod

metadata:

  name: kplabs-pod

spec:

  nodeSelector:

    size: large  # Changed "Large" to "large"

  containers:

  - name: service-pod

    image: nginx

**# our pod is succeflly attach to large worker node**

**DaemonSets in Kubernetes**

A **DaemonSet** in Kubernetes ensures that a copy of a specific Pod is running on **every node** (or a selected subset of nodes) in your cluster. This is useful for tasks like running monitoring agents, log collectors, networking agents, or any other application that needs to run on every node.

### When to Use DaemonSets:

DaemonSets are typically used when you want to run a specific service or agent on each node in the cluster. Some use cases include:

* **Logging agents** (e.g., Fluentd, Filebeat)
* **Monitoring agents** (e.g., Prometheus node exporter)
* **Networking agents** (e.g., CNI plugins)
* **Custom tools or software** that need to run on every node.

### Key Features of DaemonSets:

* **Pods on Every Node**: A DaemonSet ensures that one Pod is running on each node in your cluster (or only on a specific subset of nodes if you specify node selectors or affinity).
* **Automatic Management**: When new nodes are added to the cluster, the DaemonSet automatically schedules the Pod on those new nodes.
* **Pod Deletion**: If a node is removed from the cluster, the Pod managed by the DaemonSet on that node is also deleted.
* apiVersion: apps/v1
* kind: DaemonSet
* metadata:
* name: anti-virus
* spec:
* selector:
* matchLabels:
* app: nginx
* template:
* metadata:
* labels:
* app: nginx
* spec:
* containers:
* - name: nginx
* image: nginx
* daemonset.yaml
* kubectl get daemonset
* C:\Windows\system32>kubectl get pods
* NAME READY STATUS RESTARTS AGE
* anti-virus-wks8s 1/1 Running 0 43s
* **#dameon set autmatilly create our pod in different worker node**
* # daemon-set use cases learning
* **$ if we have there worker node and we create daemon set , so it will launch in eveyr worker node by default , we create it for only one worker node but it load in three worker node use in prodn environment**

**Node affinity**

@using node selector **we specify label and pod is sehecluede in those nodes**

In prodn we use node affinity

To run pod in which node we can decide using node affinity

**Require all cond need to satfiyt only will execute** and preffred only specific part will be executed

**Node Affinity in Kubernetes**

**Node Affinity** is a set of rules used by Kubernetes to control **where Pods can be scheduled** based on **node labels**. It is a more powerful and flexible alternative to **nodeSelector** for specifying constraints about where Pods should run in the cluster. Node Affinity allows you to define more complex scheduling rules, such as requiring or preferring Pods to run on nodes with specific labels, as well as using operators like In, NotIn, Exists, and DoesNotExist.

apiVersion: v1

kind: Pod

metadata:

  name: nginx

spec:

  affinity:

    nodeAffinity:

      requiredDuringSchedulingIgnoredDuringExecution:

        nodeSelectorTerms:

        - matchExpressions:

          - key: disktype

            operator: In

            values:

            - ssd

  containers:

  - name: nginx

    image: nginx

nodeaffintiy-require.yaml

### Breakdown:

* **nodeAffinity.requiredDuringSchedulingIgnoredDuringExecution**: This section defines the required rules for scheduling the Pod on nodes.
* **nodeSelectorTerms**: This is a list of node selector terms. The Pod must match at least one of these terms for it to be scheduled onto a node.
* **matchExpressions**: This specifies the label key-value pairs the node must have.
  + **key: disktype**: The label key disktype.
  + **operator: In**: The operator In means that the node's label value must be one of the listed values.
  + **values: [ssd]**: The allowed value for the disktype label is ssd. Only nodes with the disktype=ssd label can be selected.

Node Affinity is defined under the affinity field in the Pod specification and is divided into two main types:

* **Required Node Affinity**: These are the rules that must be met for the Pod to be scheduled onto a node.
* **Preferred Node Affinity**: These are the rules that Kubernetes will try to satisfy if possible, but it is not required for scheduling.

**@worker nodes these things to be part as label**

kubectl apply -f nodeAffnity-required.yaml

### Node Affinity Types:

1. **RequiredDuringSchedulingIgnoredDuringExecution**:
   * This type of node affinity is a **hard constraint**. The Pod will only be scheduled onto nodes that meet the criteria defined in this section.
   * If the constraints are not met, the Pod will not be scheduled on any node.
2. **PreferredDuringSchedulingIgnoredDuringExecution**:
   * This is a **soft constraint**. Kubernetes will try to schedule the Pod on a node that satisfies the condition, but if no such node is available, it will schedule the Pod on a node that doesn't meet the condition.

apiVersion: v1

kind: Pod

metadata:

  name: nginx

spec:

  affinity:

    nodeAffinity:

      preferredDuringSchedulingIgnoredDuringExecution:

      - weight: 1

        preference:

          matchExpressions:

          - key: disktype

            operator: In

            values:

            - ssd

  containers:

  - name: nginx

    image: nginx

# NOW pod is in running state after starify all codntion – in prefer – pod will run

**Example 2: Preferred Node Affinity**

Here’s an example of **preferred** node affinity, where the scheduler tries to place the Pod on nodes with the label disktype=ssd, but it will still be scheduled on other nodes if no suitable node is found.

yaml

CopyEdit

### Breakdown:

* **preferredDuringSchedulingIgnoredDuringExecution**: This section defines the preferred rules for scheduling the Pod.
* **weight: 100**: This indicates the importance of this rule (the higher the weight, the stronger the preference). The scheduler will prioritize nodes that match this rule but will still consider nodes that don’t match it if necessary.
* **matchExpressions**: Similar to the required affinity, this specifies that the node must have the label disktype=ssd (in this case, it's just a preference, not a requirement).

kubectl apply -f nodeAffnity-required.yaml

pod/nginx created

C:\Windows\system32>kubectl get nodes

NAME STATUS ROLES AGE VERSION

docker-desktop Ready control-plane 29h v1.31.4

C:\Windows\system32>kubectl label node docker-desktop disktype=ssd

node/docker-desktop labeled

#setting the dikstype of docker-dekstop so that cond meet

kubectl delete -f nodeAffnity-required.yaml

pod "nginx" deleted

request and limit

###### ChatGPT said:

In Kubernetes, **requests** and **limits** are used to manage the resources (CPU and memory) allocated to a container. These **parameters help Kubernetes make better scheduling decisions and e**nsure that containers don’t use more resources than they need, which can affect the performance of other containers running on the same node.

### Key Concepts:

* **Request**: The amount of resource that Kubernetes will **guarantee** for the container. When a container is scheduled on a node, Kubernetes uses **the requested resources to decide whether the node has enough resources available** to run the container.
* **Limit**: The **maximum amount of resource that the container** **can use**. If the container exceeds this limit, Kubernetes may take corrective actions, like throttling CPU usage or killing the container (in case of memory limits).

### How Requests and Limits Work:

1. **Requests**:
   * **CPU Request**: The amount of CPU the container is guaranteed to receive. Kubernetes uses this value when deciding which node to place the container on. If a container tries to use more CPU than its request, it can do so, as long as the node has available CPU.
   * **Memory Request**: The amount of memory the container is guaranteed to receive. Kubernetes uses this value to calculate how much memory is needed on the node for scheduling.
2. **Limits**:
   * **CPU Limit**: The maximum amount of CPU that the container can use. If the container exceeds this limit, Kubernetes can throttle the CPU usage.
   * **Memory Limit**: The maximum amount of memory the container can use. If the container exceeds this limit, it will be terminated (OOMKilled) and may be restarted based on the restart policy.

### Why Are Requests and Limits Important?

* **Efficient Resource Management**: Kubernetes needs to know how much CPU and memory to allocate for each container. By setting **requests** and **limits**, you help Kubernetes make scheduling decisions that optimize resource usage and avoid contention between containers.
* **Quality of Service (QoS)**: Kubernetes uses the requests and limits to classify Pods into three QoS classes:
  1. **Guaranteed**: If both **request** and **limit** are the same for both CPU and memory, the Pod is classified as "Guaranteed."
  2. **Burstable**: If **requests** are less than **limits** (i.e., the container can burst up to the limit), the Pod is classified as "Burstable."
  3. **BestEffort**: If neither **requests** nor **limits** are set, or if only **limits** are set, the Pod is classified as "BestEffort."

apiVersion: v1

kind: Pod

metadata:

  name: kplabs-pod

spec:

  containers:

  - name: kplabs-container

    image: nginx

    resources:

      requests:

        memory: "128Mi"

        cpu: "0.1"

      limits:

        memory: "500Mi"

        cpu: "1"

PS D:\CKA BY zeal vora\PracticeME\section3> kubectl apply -f request-limit.yaml

pod/kplabs-pod created

S D:\CKA BY zeal vora\PracticeME\section3> kubectl delete -f request-limit.yaml

pod "kplabs-pod" deleted

### What Happens When a Container Exceeds Its Limits?

* **CPU Limit**: If the container tries to use more CPU than its **limit**, Kubernetes will **throttle** the container’s CPU usage. It doesn't terminate the container but ensures the container doesn't exceed the specified CPU.
* **Memory Limit**: If the container exceeds its **memory limit**, Kubernetes will **terminate** the container (it will be "OOMKilled" – out of memory killed). The container may be restarted depending on the Pod's restart policy.

### Why Set Requests and Limits?

* **Efficient Resource Allocation**: Requests help Kubernetes schedule Pods effectively by ensuring that Pods are scheduled only on nodes with enough available resources.
* **Avoid Resource Starvation**: Setting requests and limits ensures that no single container consumes excessive resources, potentially starving other containers.

**# Service**

**Pod can communicate with each other**

**# frontend pod communicate with bakcend pod**

**Service can distribute traffic to wide varity of pod s**

C:\Windows\system32>kubectl get services

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

**$ throught service we can distribute traffic , we are sending traffic from frenotend pod to service and service sending traffic to bakcedn pod**

# service automatically send traffic to backend pod , from frontned we are making request to service

**ClusterIP (default)**:

* Exposes the service on a cluster-internal IP.
* The service is only accessible within the cluster (i.e., it cannot be accessed from outside the cluster).
* This is the default type of service.

**NodePort**:

* Exposes the service on each Node’s IP at a static port (the NodePort).
* You can access the service from outside the cluster by requesting <NodeIP>:<NodePort>.
* It is often used when combined with an external load balancer.

**LoadBalancer**:

* Creates an external load balancer (if supported by the cloud provider) and assigns a public IP address.
* This type is used to expose a service externally with a cloud-based load balancer.

**ExternalName**:

* Maps a service to an external DNS name (i.e., allows access to an external resource by DNS).
* This type doesn’t expose Pods but instead maps the service to an external domain.

### Key Features of a Kubernetes Service:

* **Stable Endpoint**: Unlike Pods, which can be ephemeral and change IP addresses, a Service provides a stable endpoint to access Pods.
* **Load Balancing**: A Service automatically balances traffic across Pods that match its selector.
* apiVersion: v1
* kind: Service
* metadata:
* name: simple-service
* spec:
* ports:
* - port: 80
* targetPort: 80
* service.yaml
* PS D:\CKA BY zeal vora\PracticeME\section4> kubectl create -f service.yaml
* service/simple-service created
* PS D:\CKA BY zeal vora\PracticeME\section4> kubectl get service
* NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
* kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 73m
* simple-service ClusterIP 10.100.246.207 <none> 80/TCP 23s
* **# wehenever frontend pod want to communicate with backend pod it need to communicate through that ip address only**
* $ through service we can setup communication between frontend and backend pod
* **# backend are nothing but endpoints , endpoints receieve request from service , endpoints have ip address of backendpod**
* So **that communication can setup easily between frontend pod to backend pod**
* # we are sending request to service . and tehn service sending request to backend pod ,service has ip add of backedn pod .
* $ fornend point connect to simple serveice but fails , we need to add enpoint manully
* apiVersion: v1
* kind: Endpoints
* metadata:
* name: simple-service
* subsets:
* - addresses:
* - ip: 10.244.0.4  # IP address of backend pod
* ports:
* - port: 80
* Endpoints.yaml
* PS D:\CKA BY zeal vora\PracticeME\section4> kubectl create -f endpoints.yaml
* endpoints/simple-service created
* C:\Windows\system32>kubectl describe service simple-service
* Name: simple-service
* **$ now we have endpoints ip , if service get request then it forward to enpoints ip – in this way we can setup commn between frontend to backend pod**

apiVersion: v1

kind: Service

metadata:

  name: simple-service

spec:

  ports:

  - port: 8080

    targetPort: 80

service.yaml

#### port: 8080

* This is the **port on which the service is exposed**.
* Any external application or pod trying to access this service will do so via 8080.

#### targetPort: 80

* This is the **port on the container (inside the pod) where the application is running**.
* It means that traffic arriving at 8080 on the service will be forwarded to port 80 inside the container.
* **# when forentne pod make request on port 8080 of service that request send on port 80 of backend port 80**
* **Service become port and backend become target port**

D:\CKA BY zeal vora\PracticeME\section4>kubectl create service clusterip test-service --tcp=80:80

service/test-service created

D:\CKA BY zeal vora\PracticeME\section4>kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 5h13m

apiVersion: v1

kind: Service

metadata:

  name: backend-nodeport-service

spec:

  type: NodePort

  selector:

    app: backend  # Ensure this matches the backend pod label

  ports:

    - protocol: TCP

      port: 80       # The service port

      targetPort: 80 # The backend pod's port

      nodePort: 30008 # NodePort (must be in range 30000-32767)

node-port.yaml

D:\CKA BY zeal vora\PracticeME\section4>kubectl create service clusterip test-service --tcp=80:80

service/test-service created

Service type load balancer

**Nodeport expose out envirnomet to external world** ,nodeprot not used for esxtrnal env prod

Load balancer intermediate between user and node port , load balancer send traffic to node port

In Kubernetes, a **NodePort** service **exposes a Kubernetes service on each node’s IP address** at a static port. This **allows external traffic to access your service through any of the nodes** in the cluster at the specified NodePor

### Key Points About NodePort:

* **Accessible on any node's IP**: Once you create a NodePort service, you can access the service from outside the cluster using <NodeIP>:<NodePort>. Any of the nodes in the cluster can be used to reach the service.
* **Port range**: The default NodePort range is between 30000 and 32767. Kubernetes will automatically assign a port from this range unless you specify one explicitly.
* **Port forwarding**: A NodePort service works as a kind of port forwarding. When an external request hits the node on the NodePort, the request is forwarded to one of the Pods that the service selects.

### How It Works:

1. **Accessing the Service**: After the service is created, you can access it using any of the nodes in the cluster on the given NodePort. For example, if the node's IP is 192.168.1.100, you can access the service externally via http://192.168.1.100:30007.
2. **Load Balancing**: Kubernetes will automatically load balance the incoming traffic across the Pods selected by the service, even though the traffic is coming through the NodePort.

**$ Bydfeilult node port automatically create , load balancer forward traffic to node port of worker node , and then it directly reach the traffic to pod**

Ingrees

**Whenve a load balancer receive request for any service , like ex.internal - it will go to ingrees component , ingress comp through it logic will forware to ex service based on rules**

**We define rule in ingress that route traffic**

**Load balacenr sned traffic to ingrees controller ,controller recive reqeuset from laod balancer and cotroller based on rules route traffic to approacpirate service**

**Controller based on rules send traffic to services**

$ ingress route traffic to service

D:\CKA BY zeal vora\PracticeME\section4>kubectl get ingress

No resources found in default namespace.

D:\CKA BY zeal vora\PracticeME\section4>kubectl create ingress --help

Create an ingress with the specified name

D:\CKA BY zeal vora\kuberntes\_practice\_23 mar\part-15-ingress>kubectl create ingress first-ingress-rule --rule="example.internal/\*=example-service:80"

ingress.networking.k8s.io/first-ingress-rule created

C:\>kubectl get service

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.109.0.1 <none> 443/TCP 26d

C:\>kubectl describe ingress first-ingress

**$ If any traffic come for ex.internal then it redirect to ex-service**

C:\>kubectl create ingress first-ingress --rule="example.internal/\*-example-service: 80" --rule="kplabs.internal/\*-kplabs-serv

ice:80"

C:\>kubectl describe ingress second-ingress

D:\CKA BY zeal vora\PracticeME\section4>kubectl create ingress second-ingress --rule="example.internal/\*=example-service:80" --rule="kplabs.internal/\*=kplabs-service:80"

ingress.networking.k8s.io/second-ingress created

D:\CKA BY zeal vora\PracticeME\section4>kubectl describe ingress second-ingress

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: path-based-ingress

spec:

  rules:

    - http:

        paths:

          - path: /app-1

            pathType: Exact

            backend:

              service:

                name: app-1-service

                port:

                  number: 80

          - path: /app-2

            pathType: Exact

            backend:

              service:

                name: app-2-service

                port:

                  number: 80

path.yaml

D:\CKA BY zeal vora\PracticeME\section4>kubectl create -f path-based.yaml

ingress.networking.k8s.io/path-based-ingress created

D:\CKA BY zeal vora\PracticeME\section4>kubectl describe ingress path-based

Rules:

Host Path Backends

---- ---- --------

\*

/app-1 app-1-service:80 (<error: services "app-1-service" not found>)

/app-2 app-2-service:80 (<error: services "app-2-service" not found>)

Annotations: <none>

Events: <none>

**$ If any request come on this path /app-1 then it redirect to app1-service**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  creationTimestamp: null

  name: named-path

spec:

  rules:

  - host: example.internal

    http:

      paths:

      - backend:

          service:

            name: app-1-service

            port:

              number: 80

        path: /app-1

        pathType: Exact

      - backend:

          service:

            name: app-2-service

            port:

              number: 80

        path: /app-2

        pathType: Exact

name-path.yaml

**1. host: example.internal**

yaml

CopyEdit

rules:

- host: example.internal

* The host specifies the **domain name** that this Ingress rule applies to.
* When a user accesses **http://example.internal**, this Ingress rule determines **which backend service** will handle the request.
* If host is omitted, the rule applies to **all hosts (\*)**.

✅ **Example Request Matching**

* A request to http://example.internal/app-1 → Matches the rule and routes traffic to **app-1-service**.
* A request to http://example.internal/app-2 → Routes traffic to **app-2-service**.
* A request to http://different-domain.com/app-1 → ❌ **Does not match** this Ingress rule.

**2. path: /app-1 and /app-2**

yaml

CopyEdit

- backend:

service:

name: app-1-service

port:

number: 80

path: /app-1

pathType: Exact

* The path defines the **specific URL path** that must be requested for this rule to apply.
* It works **with the host** to determine where traffic should be sent.

**Example Request Matching**

| **Request URL** | **Matched Path** | **Routed To** |
| --- | --- | --- |
| http://example.internal/app-1 | /app-1 | app-1-service:80 |
| http://example.internal/app-2 | /app-2 | app-2-service:80 |
| http://example.internal/app-3 | ❌ No match | ❌ 404 Not Found |

**pathType: Exact**

yaml

CopyEdit

pathType: Exact

* This means the path **must match exactly** (/app-1 or /app-2).
* A request to http://example.internal/app-1/extra **will not match** /app-1 because it's not an exact match.
* If you want to match **any subpath**, use pathType: Prefix instead.

### **Final Example**

🔹 If a user visits **http://example.internal/app-1**, it gets routed to app-1-service.  
🔹 If they visit **http://example.internal/app-2**, it gets routed to app-2-service.  
🔹 If they visit **http://example.internal/app-3**, they get a **404 Not Found**.

**HELM**

Deploy worddpress chart in kuberntes cluster using helm

Package your whole appn In helm charts

Deploy helm chart in wordpress in kuberntes cluster

**Namespaces**

Used to islolate resources in kuberntes

C:\Windows\system32>kubectl get pods --all-namespaces

NAMESPACE NAME READY STATUS RESTARTS AGE

kube-system coredns-668d6bf9bc-qw7vf 1/1 Running 1 (3h35m ago) 21h

C:\Windows\system32>kubectl get configmaps

NAME DATA AGE

kube-root-ca.crt 1 21h

C:\Windows\system32>kubectl get configmaps --all-namespaces

NAMESPACE NAME DATA AGE

default kube-root-ca.crt 1 22h

# **never tuch kube system otherwise functionality of kuberntes cluster might be impactes**