

└─ Pods
└─ Containers

4. Quick analogy

Think of it like this:

- **Cluster = City**
- **Node = Building**
- **Pod = Apartment**
- **Container = People living in that apartment**

You don't put buildings inside apartments — same way, you don't put nodes inside pods.

Final Answer

- **Cluster** → has control-plane and worker nodes
- **Worker nodes** → run pods
- **Pods** → run containers

If you want, I can now explain:

- What exactly runs on control-plane nodes
- Why pods always run on workers
- How scheduling works (how K8s decides which node should run your pod)

Just tell me.

can you explain me the full hierarchy of k8s

Absolutely — let's build the **full hierarchy of Kubernetes (K8s)** from the *largest outermost level* to the *smallest component that actually executes code*.

I'll make it extremely clear, layered, visual, and interconnected so the architecture becomes unforgettable.

LEVEL 1 — The Kubernetes Cluster (the whole universe)

A **cluster** is the complete Kubernetes environment.

It contains:

- Control-plane nodes (brains)

- Worker nodes (muscle)
- Networking + storage
- All workloads (pods, services, deployments)

CSS

```
[ KUBERNETES CLUSTER ]
      /      \
[ Control Plane ] [ Worker Nodes ]
```

■ LEVEL 2 — Control Plane vs Worker Nodes

1. Control Plane Nodes

These nodes run the components that *manage and operate the cluster itself*.

Core components inside control-plane:

- **kube-apiserver** (main entry point for all commands)
- **etcd** (database storing cluster state)
- **kube-scheduler** (decides which node gets a pod)
- **kube-controller-manager**
- **cloud-controller-manager** (optional)

Diagram:

nginx

```
CONTROL PLANE NODE
├── kube-apiserver
├── etcd
├── scheduler
└── controller-manager
```

2. Worker Nodes

These nodes run your **applications**.

Each worker node has:

- **kubelet** (manages pods on this node)
- **kube-proxy** (manages networking rules)
- **Container runtime** (containerd, Docker, etc.)
- **Pods** (your actual workloads)

Diagram:

```
nginx
```

```
WORKER NODE
├── kubelet
├── kube-proxy
├── container runtime
└── Pods...
```

■ LEVEL 3 — Pods (smallest deployable unit)

A **Pod** is the basic execution environment.

A pod:

- Has **one IP**
- Runs **1 or more containers**
- Shares network + volume space between containers
- Always runs on a worker node
- Is not self-healing by itself

Diagram:

```
css
```

```
POD
├── Container A
└── Container B (optional)
```

■ LEVEL 4 — Containers

Inside a pod are **containers**:

- Docker or containerd images
- Running your actual application code
- Each container has its own filesystem
- Containers inside a pod share:
 - network namespace
 - volumes

```
ini
```

```
CONTAINER = your actual code running
```

This is where Python, Node.js, Nginx, Go, etc. run.

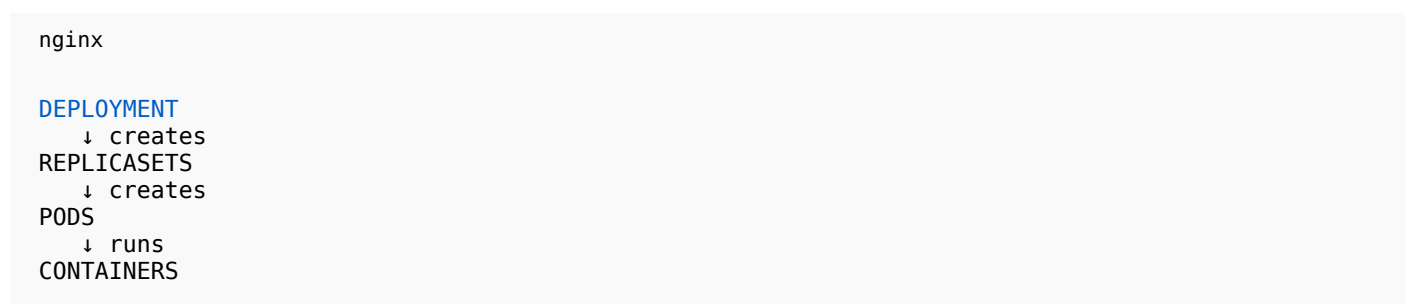
■ LEVEL 5 — Controllers (manage pods)

Kubernetes has “controllers” that watch and maintain the desired state. These are **API-layer abstractions**.

Common controllers:

- ✓ **Deployment** → manages stateless pods (web apps)
- ✓ **ReplicaSet** → maintains number of pod replicas
- ✓ **StatefulSet** → for stateful apps (databases)
- ✓ **DaemonSet** → 1 pod per node (log agents, monitoring)
- ✓ **Job / CronJob** → run-to-completion tasks

Example hierarchy:



LEVEL 6 — Services (expose pods)

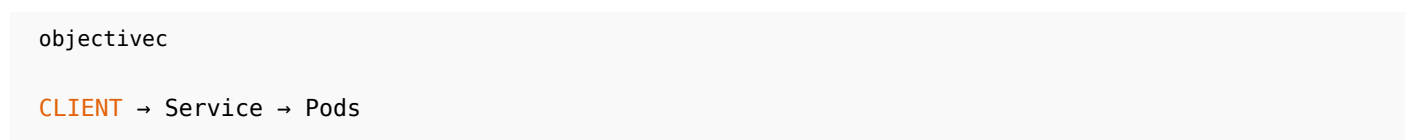
Pods die and restart, so their IPs change.

Services give a stable network identity.

Types:

- **ClusterIP** → internal access
- **NodePort** → exposed on each node
- **LoadBalancer** → cloud external load balancer
- **Headless Service** → for StatefulSets

Diagram:



LEVEL 7 — Higher Abstractions

These are optional but used in real deployments:

- **Ingress** → routes HTTP traffic to services

- **Ingress Controller** → Nginx, Traefik, HAProxy
- **ConfigMaps** → configuration files
- **Secrets** → sensitive data (passwords/tokens)
- **PersistentVolumes (PV)**
- **PersistentVolumeClaims (PVC)**
- **StorageClass**

LEVEL 8 — Storage Hierarchy

Storage in Kubernetes works like:

```
css

StorageClass
  ↓ provisions
PersistentVolume
  ↓ bound to
PersistentVolumeClaim
  ↓ mounted in
Pod
```

LEVEL 9 — Networking Hierarchy

Networking flows like:

```
nginx

Ingress → Service → Pod → Container
```

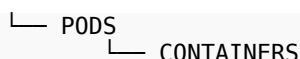
Every pod gets:

- Its own IP
- Can talk to any other pod (Flat network model)

THE FULL HIERARCHY (One Big Diagram)

```
arduino

CLUSTER
├── CONTROL PLANE NODES
│   ├── API Server
│   ├── etcd
│   └── Scheduler
│       └── Controllers
└── WORKER NODES
    ├── Kubelet
    ├── Kube-proxy
    └── Container Runtime
```



Controllers and higher-level resources (outside nodes):

```

sql

DEPLOYMENT
  ↓ creates
REPLICASET
  ↓ creates
POD
  ↓ contains
CONTAINERS
  
```

Networking & Storage:

```

nginx

INGRESS → SERVICE → POD → CONTAINER

STORAGECLASS → PV → PVC → POD
  
```

In One Sentence

Kubernetes hierarchy is:

Cluster → Nodes → Pods → Containers
and above pods: Deployments/ReplicaSets/Services/Ingress/Storage.

This is the complete mental map.

If you want, I can draw a **full ASCII architecture diagram**, or explain:

- how the scheduler decides which node gets a pod
- how controllers maintain the desired state
- how traffic flows from internet → ingress → pod
- or a real example of deploying an app and watching the whole hierarchy interact

Just tell me!

ok so how do you create a cluster in kind , with 10 pods , each with a different set of services like nginx+python , or java+maven or something, with and without deployments

Let's **build the understanding from the ground up**, because "Kind cluster with 10 pods each running different stacks" sounds simple but hides *several* Kubernetes concepts.

Below is a **complete, deeply explained, beginner-friendly walkthrough**, showing both:

1. **With Deployments** (recommended, scalable, self-healing)

2. **Without Deployments** (raw Pods, never used in real production but good for learning)

We'll create a Kind cluster, then create 10 Pods—some running **nginx+python**, some **java+maven**, etc.

✓ **STEP 1 — Create a Kind Cluster**

You can create a basic cluster like this:

```
bash

kind create cluster --name multi-stack-cluster
```

Or if you want multiple worker nodes:

```
bash

cat <<EOF > kind-config.yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
  - role: worker
  - role: worker
  - role: worker
EOF

kind create cluster --name multi-stack-cluster --config kind-config.yaml
```

✓ **STEP 2 — Option A: Create 10 Pods WITHOUT Deployments**

This is the raw, direct way: 10 YAML files or 1 file with 10 pod objects.

Here is a **single YAML** defining 10 pods with different stacks:

```
yaml

apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx-python
spec:
  containers:
    - name: nginx
      image: nginx:latest
    - name: python
      image: python:3.10-slim
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-java-maven
spec:
  containers:
    - name: java
      image: openjdk:17
    - name: maven
      image: maven:3.9
---
apiVersion: v1
kind: Pod
metadata:
```

```

name: pod-go
spec:
  containers:
    - name: go
      image: golang:1.22
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-node
spec:
  containers:
    - name: node
      image: node:20
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-redis
spec:
  containers:
    - name: redis
      image: redis:7
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-postgres
spec:
  containers:
    - name: postgres
      image: postgres:16
      env:
        - name: POSTGRES_PASSWORD
          value: root
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-rabbitmq
spec:
  containers:
    - name: rabbit
      image: rabbitmq:3
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-mongo
spec:
  containers:
    - name: mongo
      image: mongo:7
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-flask
spec:
  containers:
    - name: flask
      image: python:3.10-slim
      command: ["python", "-m", "flask", "run", "--host=0.0.0.0"]
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-nginx-only
spec:
  containers:
    - name: nginx
      image: nginx

```


Apply all:

```
bash

kubectl apply -f multi-pods.yaml
```

Check:

```
bash

kubectl get pods
```

- **This creates 10 pods exactly as specified.**
- But if ANY pod crashes → Kubernetes does NOT restart it.
- This method is only good for *learning*.

✓ STEP 3 — Option B: Create 10 Pods WITH Deployments (Recommended)

Deployments automatically handle:

- ✓ Restart on crash
- ✓ Scaling
- ✓ Replica management
- ✓ Rollouts

Below is a **clean example** of 10 Deployments, each representing a tech stack.

Here is **2 example deployments**. You can easily extend to 10.

📌 Deployment 1: Nginx + Python (multi-container pod)

```
yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-nginx-python
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx-python
  template:
    metadata:
      labels:
        app: nginx-python
    spec:
      containers:
        - name: nginx
          image: nginx:latest
        - name: python
          image: python:3.10-slim
```

📌 Deployment 2: Java + Maven

```
yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-java-maven
spec:
  replicas: 1
  selector:
    matchLabels:
      app: java-maven
  template:
    metadata:
      labels:
        app: java-maven
    spec:
      containers:
        - name: java
          image: openjdk:17
        - name: maven
          image: maven:3.9
```

Apply these:

```
bash
```

```
kubectl apply -f deployments.yaml
```

You can scale any Deployment, for example:

```
bash
```

```
kubectl scale deploy nginx-python --replicas=3
```

Summary (Very Beginner Friendly)

Method	What it Means	Use Case
Pods (No Deployment)	You create raw pods	Only education/debugging
Deployments	Kubernetes manages pods for you	Real apps + stability
Multi-container pods	nginx + python in 1 pod	When containers are tightly coupled
Separate pods	nginx in one pod, python in another	Real microservices

Bonus: If You Want Exactly “10 Pods Total” No Matter What

Choose:

- **10 raw Pods** (Option A)
or
- **10 Deployments with replicas=1** (Option B)

Both result in exactly 10 pods.

If you want, I can also generate:

- ✓ one YAML file containing **10 Deployments** (all different stacks)
- ✓ services to expose each
- ✓ NodePort or Ingress
- ✓ load-balanced architecture
- ✓ multi-node Kind setup

Just tell me “generate full file” and I’ll do it.