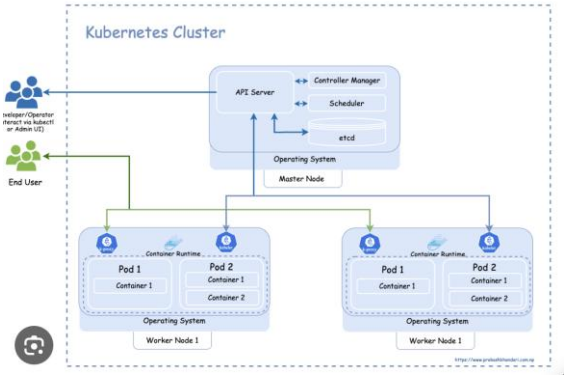
🚀 **What is Kubernetes (K8s)?24/08/2025**

Kubernetes is a **container orchestration platform** that helps you:

- **Deploy applications in containers (like Docker).**
- **Scale them up/down automatically.**
- **Load balance traffic.**
- **Heal itself (restarts crashed containers).**
- **Manage storage and secrets.**

Think of it as a **cluster manager** for containers.



⚙️ **Basic Concepts**

1. **Pod** → Smallest unit. Runs one or more containers. هر پاد یک یا چندین کانتینر دارد
2. **Deployment** → Manages Pods (scaling, rolling updates).
3. **Service** → Exposes Pods to network (**ClusterIP, NodePort, LoadBalancer**).
4. **Namespace** → Logical grouping of resources. مثل networkمشترک
5. **ConfigMap / Secret** → Stores config & sensitive data.
6. **Node** → A worker machine (VM/physical) in the cluster.هر نود یک ویرچوآل برای اجرا کردن پاد ها هست
7. **Cluster** → Group of nodes managed by Kubernetes. هر کلاستر چنتا نود دارد و کلا همه ی تصویر بالا داخل یک کلاستر است

📖 **Common kubectl Commands**

🔍 **Cluster Info**

```
kubectl cluster-info          # Show cluster details
kubectl get nodes             # List all nodes
kubectl get namespaces        # List namespaces
```

📦 **Pods**

```
kubectl get pods                          # List pods in current namespace
kubectl get pods -A                       # List pods in all namespaces
kubectl describe pod <pod-name>           # Detailed pod info
kubectl logs <pod-name>                   # Show logs of a pod
kubectl exec -it <pod-name> -- /bin/sh    # Access pod shell
```

📜 **Deployments**

```
kubectl get deployments                          # List deployments
kubectl create deployment nginx --image=nginx    # Quick create
kubectl scale deployment nginx --replicas=3      # Scale pods
kubectl rollout status deployment/nginx          # Check rollout
kubectl delete deployment nginx                  # Delete deployment
```

🌐 **Services**

```
kubectl expose deployment nginx --port=80 --type=NodePort  # Expose app
kubectl get svc                                            # List services
```

🛠️ **Apply / Delete from YAML**

```
kubectl apply -f app.yaml          # Create/update resource from file
kubectl delete -f app.yaml         # Delete resource from file
```

📝 **Example Workflow**

1. Deploy an Nginx pod:

```
kubectl create deployment nginx --image=nginx
```

2. Expose it as a service:

   kubectl expose deployment nginx --port=80 --type=NodePort

3. Get service info:

   kubectl get svc

4. Open in browser using **minikube service nginx** (if using Minikube locally).

مثال

Docker way (single host) راهکار داکری وبرای مثال است و در اینده با کوبر پیاده شده

If you just want 3 containers running together (like a mini cluster), you'd typically use **Docker Compose**:

```
# docker-compose.yaml
version: "3"
services:
 app1:
  image: nginx
 app2:
  image: redis
 app3:
  image: busybox
  command: sleep 3600
```

Run it:

docker-compose up -d

But here, each container is **separate**. Kubernetes handles this differently.

Kubernetes way راهکار بالا با کوبر

In Kubernetes, you can put **multiple containers in one Pod**. These containers:

- Share the **same network namespace** (so they talk over localhost).
- Can share **volumes**.
- Useful for "sidecar" patterns (e.g., logging or proxy container).

---

1. A Pod with 3 Containers

Here's a YAML example (**three-containers-pod.yaml**):

```
apiVersion: v1
kind: Pod
metadata:
    name: multi-container-pod
spec:
    containers:
    - name: nginx-container
        image: nginx
        ports:
        - containerPort: 80
    - name: redis-container
        image: redis
        ports:
        - containerPort: 6379
    - name: busybox-container
        image: busybox
        command: ["sh", "-c", "while true; do echo hello; sleep 5; done"]
```

Apply it:

kubectl apply -f three-containers-pod.yaml

kubectl get pods

Check logs of each container:

kubectl logs multi-container-pod -c nginx-container

kubectl logs multi-container-pod -c redis-container

kubectl logs multi-container-pod -c busybox-container

---

2. Running this Pod as a Cluster Deployment مثلا از 3 تا کانتینر بالا که یک پاد شده ما حالا چندین پاد بیاریم بالا و این ها رو در یک کلاستر بزاریم

If you want **multiple copies of this pod across the cluster**, use a **Deployment**:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: multi-container-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: multi-container
  template:
    metadata:
      labels:
        app: multi-container
    spec:
      containers:
        - name: nginx-container
          image: nginx
        - name: redis-container
          image: redis
        - name: busybox-container
          image: busybox
          command: ["sh", "-c", "while true; do echo hello; sleep 5; done"]
```

Apply it:

```
kubectl apply -f multi-container-deployment.yaml
kubectl get pods
```

You'll see **3 Pods**, each running **3 containers** → total **9 containers in your cluster**.

✅ So summary:

- **Pod with 3 containers** → all run together on 1 node.
- **Deployment with replicas=3** → multiple Pods across the cluster → fault-tolerance + scaling.

To **stop / shut it down**, you have a few options depending on what you want:

🟡 **Option 1: Delete the deployment completely**

```
kubectl delete -f multi-container-deployment.yaml
OR
kubectl delete deployment multi-container-deployment
```

🟡 **Option 2: Scale replicas to zero**

```
kubectl scale deployment multi-container-deployment --replicas=0
Later, you can restart it:
kubectl scale deployment multi-container-deployment --replicas=3
```

🟢 **Option 3: Pause / resume rollouts (for updates only)**

```
kubectl rollout pause deployment multi-container-deployment
kubectl rollout resume deployment multi-container-deployment
```

I want me to also show you how to expose one of those containers (like nginx) to the outside world using a Service
اجرای کامند ها به صورت دستی و در ادامه این کامند ها در یک فایل ادغام شده

🔹 **Step 1: Expose Deployment as a Service**

Run this command:

```
kubectl expose deployment multi-container-deployment \
  --name=nginx-service \
  --port=80 \
  --target-port=80 \
```

--type=NodePort

What this does:

- **--port=80** → the port the Service exposes.
- **--target-port=80** → the port inside the nginx container.
- **--type=NodePort** → exposes it on a high port (30000–32767) of your cluster nodes.

🔹 **Step 2: Check the Service**

kubectl get svc nginx-service

Example output:

```
NAME            TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
nginx-service   NodePort   10.96.28.12    <none>        80:31425/TCP   2m
```

- 80 is the service port.
- 31425 is the **NodePort** (random high port assigned).

🔹 **Step 3: Access nginx**

- If you're running Kubernetes with **Minikube**:

minikube service nginx-service

This opens your browser automatically.

- If you're running on a real cluster:
  - Use the **Node IP** + NodePort. Example:
  - http://<node-ip>:31425

🔹 **Step 4 (Optional): Use LoadBalancer instead**

If your Kubernetes cluster is on a cloud (GKE, EKS, AKS), you can make it public with a **LoadBalancer**:

```
kubectl expose deployment multi-container-deployment \
 --name=nginx-lb \
 --port=80 \
 --target-port=80 \
 --type=LoadBalancer
```

Then kubectl get svc nginx-lb will eventually show an **external IP** to access directly.

⚡ Recap:

- NodePort → good for local clusters like Minikube or bare-metal.
- LoadBalancer → best for cloud, gives you a real public IP.

add an updated **YAML file** example that includes the **Service definition** along with your multi-container Deployment (**so you don't have to run separate commands**)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: multi-container-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: multi-container
  template:
    metadata:
      labels:
        app: multi-container
    spec:
      containers:
      - name: nginx-container
        image: nginx
        ports:
        - containerPort: 80
```

```
    - name: redis-container
      image: redis
      ports:
      - containerPort: 6379
    - name: busybox-container
      image: busybox
      command: ["sh", "-c", "while true; do echo hello; sleep 5; done"]

---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
              app: multi-container
  ports:
              - protocol: TCP
                port: 80        # service port
                targetPort: 80    # nginx container port
  type: NodePort       # use LoadBalancer if on cloud (GKE/EKS/AKS)
```

▶️ **Apply it:**

kubectl apply -f multi-container-deployment-with-service.yaml

🔍 **Check resources:**

kubectl get pods

kubectl get svc nginx-service

Example service output:

```
NAME            TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)       AGE
nginx-service   NodePort   10.96.114.233  <none>        80:31425/TCP  1m
```

Now you can reach nginx at:

- **Minikube:**

  minikube service nginx-service

- **Real cluster:**

  Open http://<node-ip>:31425

⚡ Bonus: If you're on the cloud and want an external IP, just change type: NodePort → type: LoadBalancer.

Here's how Kubernetes decides **which container gets the traffic**:

🔹 **How Services Route Traffic in a Multi-Container Pod**

- A **Service doesn't know about containers individually**.
- It only knows about **Pod IP + port**.
- When you create a Service with targetPort: 80, Kubernetes routes traffic to the Pod's IP on port 80.
- Inside the Pod, whichever container has something **listening on port 80** will receive the traffic.

👉 In your case:

- nginx-container listens on **80** → the Service routes traffic here.
- redis-container listens on **6379** → not exposed by this Service.
- busybox-container doesn't expose ports → ignored.

🔹 **What if multiple containers use the same port?**

That would be a conflict ❌, because Pods share a **network namespace**.

- Only **one container** can bind to a specific port (e.g., 80) inside the Pod.
- If two containers try, one will fail to start.

So, when exposing with a Service, you must choose **one targetPort** that maps to **one container's port**.

🔹 **Best Practice**

- Usually, a Pod has **one main application container** (like nginx) and optional **sidecars** (logging, monitoring, proxy, etc).
- Services are almost always pointed at the **main app container**.
- If you need external access to Redis too, you'd create **another Service** targeting port 6379:

```
---
apiVersion: v1
kind: Service
metadata:
 name: redis-service
spec:
 selector:
        app: multi-container
 ports:
        - protocol: TCP
          port: 6379
          targetPort: 6379
 type: NodePort
```

✅ So in summary:

- Your current Service (nginx-service) exposes **only nginx**.
- If you want to expose **redis** too → make a separate Service.
- The **busybox** container is just a sidecar (logs "hello"), so it doesn't need a Service.

---

multi-container-deployment-with-services.yaml

kubectl apply -f multi-container-deployment-with-services.yaml

---

☸️ **What is a Service?**

- A **Service** is an **abstraction that exposes Pods** (your containers) to the network.
- Pods have **dynamic IPs** (they can die and restart with new IPs), so Services give you a **stable address** to access them.
- Services also load-balance traffic across multiple Pods.

🔹 **Service Types**

1. ClusterIP (default)

- Exposes Pods **inside the cluster only**.
- No external access; other Pods/Services can talk to it.
- Useful for backend services (e.g., Redis, databases, internal APIs).

Example:

```
apiVersion: v1
kind: Service
metadata:
  name: my-internal-service
spec:
  selector:
    app: myapp
  ports:
    - port: 80
      targetPort: 8080
 type: ClusterIP
```

➡️ Accessible only inside the cluster as my-internal-service:80.

---

2. NodePort

- Exposes Pods on a **static port** (30000–32767) on every Node's IP.
- You can reach it via:

- http://<NodeIP>:<NodePort>
- Commonly used in local clusters (like Minikube).

Example:

```
apiVersion: v1
kind: Service
metadata:
 name: my-nodeport-service
spec:
 selector:
  app: myapp
 ports:
  - port: 80
    targetPort: 8080
    nodePort: 30080
 type: NodePort
```

➡️ If a Node's IP is 192.168.99.100, you access it via http://192.168.99.100:30080.

---

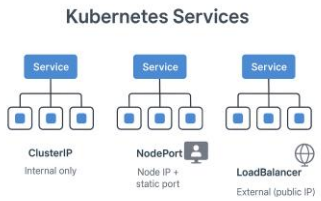
3. LoadBalancer

- Creates a **cloud provider load balancer** (AWS ELB, GCP LB, Azure LB).
- Exposes Pods to the **internet** with an external IP.
- Used in production clusters on the cloud.

Example:

```
apiVersion: v1
kind: Service
metadata:
 name: my-loadbalancer-service
spec:
 selector:
  app: myapp
 ports:
  - port: 80
    targetPort: 8080
 type: LoadBalancer
```

➡️ Cloud provider gives an **external IP** (like 34.120.50.22) → accessible anywhere.



📊 **Quick Comparison**

Service Type	Scope	Typical Use Case
ClusterIP	Internal only	Backend, databases
NodePort	Node IP + static port	Local dev, debugging
LoadBalancer	External (public IP)	Production apps on cloud

✅ So:

- Use **ClusterIP** for internal communication.
- Use **NodePort** for local dev access.
- Use **LoadBalancer** for real-world, internet-facing apps.

---

what is minikube and how to install and run it with docker?

What is Minikube?

- **Minikube** is a lightweight tool that lets you run a **single-node Kubernetes cluster** on your local machine.
- It's great for:
  - Learning Kubernetes
  - Testing apps locally
  - Development before deploying to a real cluster

👉 By default, Minikube runs Kubernetes inside a VM or container runtime like **Docker**.

⚙️ **Installing Minikube with Docker**

1. Prerequisites

- Install **Docker** (community edition):
  - Docker Install Guide
- Install **kubectl** (the Kubernetes CLI):

```
# Linux
curl -LO "https://dl.k8s.io/release/$(curl -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x kubectl
sudo mv kubectl /usr/local/bin/

# Mac (with brew)
brew install kubectl
```

2. Install Minikube

- **Linux:**

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

- **Mac (with brew):**

```
brew install minikube
```

- **Windows (with Chocolatey):**

```
choco install minikube
```

▶️ **Running Minikube with Docker**

1. Start Minikube using Docker as the driver

```
minikube start --driver=docker
```

This spins up a Kubernetes cluster inside Docker.

2. Verify installation

```
kubectl get nodes
```

You should see one node, e.g.:

```
NAME       STATUS   ROLES           AGE   VERSION
minikube   Ready    control-plane   1m    v1.30.0
```

3. Deploy an app (example: Nginx)

```
kubectl create deployment nginx --image=nginx
kubectl expose deployment nginx --port=80 --type=NodePort
```

4. Access the app

```
minikube service nginx
```

This command opens your default browser with the Nginx welcome page 🚀.

📝 **Summary**

- **Minikube** = local Kubernetes cluster for learning/dev.
- Works with **Docker driver** (so no need for VM software).
- minikube start --driver=docker boots the cluster.
- Use kubectl as usual to manage Pods, Deployments, and Services.