Run on **all nodes** (master & workers):

1. OS: Ubuntu 20.04 or 22.04

Ensure you are using a fresh installation of Ubuntu.

2. Update & Install Dependencies

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y apt-transport-https ca-certificates curl gnupg
lsb-release
```

✅ Step 2: Install Container Runtime (containerd recommended)

```
sudo apt install -y containerd

# Create default config file
sudo mkdir -p /etc/containerd
containerd config default | sudo tee /etc/containerd/config.toml

# Restart and enable
sudo systemctl restart containerd
sudo systemctl enable containerd
```

✅ Step 3: Disable Swap (Kubernetes requirement)

```
sudo swapoff -a
sudo sed -i '/ swap / s/^/#/' /etc/fstab
```

✅ Step 4: Add Kubernetes Repo & Install

```
# Add GPG key
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key |
sudo gpg --dearmor -o
/usr/share/keyrings/kubernetes-archive-keyring.gpg

# Add repo
echo "deb
[signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.30/deb/ /" | sudo tee
/etc/apt/sources.list.d/kubernetes.list

# Install
sudo apt update
sudo apt install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

✅ Step 5: Initialize Kubernetes (only on master node)

```
sudo kubeadm init --pod-network-cidr=192.168.0.0/16
```

After it completes, follow the output steps:

```
# Example:
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

✅ Step 6: Install Pod Network Add-on (e.g., Calico or Flannel)

Example: Calico

```
kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manif
ests/calico.yaml
```

If using Flannel instead, use its manifest.

✅ Step 7: Join Worker Nodes (on each worker node)

From the `kubeadm init` output, use the `kubeadm join` command. Example:

```
sudo kubeadm join <master-ip>:6443 --token <token>
--discovery-token-ca-cert-hash sha256:<hash>
```

If you lost this command, regenerate a token with:

```
kubeadm token create --print-join-command
```

✅ Step 8: Verify Cluster

On master:

```
kubectl get nodes
kubectl get pods -A
```

To start using your cluster, you need to run the following as a regular user:

```
  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Alternatively, if you are the root user, you can run:

```
  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.1.13:6443 --token d7i2jj.cdeh2zle872ji8x4 \
        --discovery-token-ca-cert-hash
sha256:3c09ef534578557eb50e77b5f4e59de33a018c8e10c18efb8c8ad4cf4bb8bd20
```

```
sudo kubeadm join <master-ip>:6443 --token <token> --discovery-token-ca-cert-hash
sha256:<hash>

kubeadm join 192.168.1.13:6443 --token j5d6si.nhos71d4a4axvq3c
--discovery-token-ca-cert-hash
sha256:3c09ef534578557eb50e77b5f4e59de33a018c8e10c18efb8c8ad4cf4bb8bd20
```

❌ Errors explained:

- `[ERROR FileAvailable--etc-kubernetes-kubelet.conf]`: The kubelet config file already exists.
- `[ERROR Port-10250]`: The port used by the kubelet is already in use.
- `[ERROR FileAvailable--etc-kubernetes-pki-ca.crt]`: TLS certificates already exist.

✅ Solution

You **must reset the node** before trying to rejoin:

Step 1: Reset the Node

```
sudo kubeadm reset -f
```

Step 2: Clean up config

```
sudo rm -rf /etc/cni/net.d
sudo rm -rf $HOME/.kube
sudo ip link delete cni0 || true
sudo ip link delete flannel.1 || true
sudo systemctl restart kubelet
```

If using `containerd`:

```
sudo ctr containers list
sudo ctr containers rm <container_id>  # if needed
```

Step 3: Retry `kubeadm join`

```
sudo kubeadm join 192.168.1.13:6443 --token j5d6si.nhos71d4a4axvq3c
--discovery-token-ca-cert-hash
sha256:3c09ef534578557eb50e77b5f4e59de33a018c8e10c18efb8c8ad4cf4bb8b
d20
```

What is Kubernetes?

- **Kubernetes** (k8s) is a system to manage and run applications inside containers at scale.
- It handles **container orchestration**: deploying, scaling, and managing containerized apps.

- Kubernetes has a **cluster architecture**: one or more **control-plane nodes** (masters) and many **worker nodes**.

What is kubeadm?

- kubeadm is a tool to **set up a Kubernetes cluster easily**.
- It automates tasks like generating certificates, configuring components, and bootstrapping the control-plane node.
- After running kubeadm init on a machine, that machine becomes the **control-plane (master) node**.

Control-plane (Master) node

- Runs key components:
  - **API Server** (port 6443): main entry point for Kubernetes API.
  - **Controller Manager**: manages controllers that regulate the cluster state.
  - **Scheduler**: assigns workloads (pods) to nodes.
  - **etcd**: distributed key-value store for cluster data.
- You run kubeadm init to install and configure these.

What happens during kubeadm init?

1. **Preflight checks**: Makes sure required ports are free, necessary files don't exist, and your system is ready.

2. **Generating certificates**: For secure communication between components.

3. **Starting control-plane components**: API server, controller manager, scheduler, and etcd start as static pods.

4. **Configuring kubelet**: The node agent that runs pods on nodes.

5. **Deploying core addons**: such as CoreDNS (for service discovery) and kube-proxy (networking).

Key Terms in Your Output

- **Pod network CIDR (--pod-network-cidr)**:
  - Kubernetes needs a range of IP addresses for pods (containers inside Kubernetes).
  - This CIDR (Classless Inter-Domain Routing) tells Kubernetes what IP range pods will use.
  - Must match the network add-on you plan to install later.
- **Ports in use**:
  - Kubernetes uses several ports:
    - 6443: Kubernetes API Server
    - 2379-2380: etcd database
    - 10250, 10257, 10259: kubelet and controller manager communication
  - If these ports are busy, Kubernetes can't start.
- **Manifest files**:

    - Static pods for core components are described in YAML files under /etc/kubernetes/manifests/.
    - These files tell kubelet what pods to run for the control-plane.
- **kubeconfig file**:
    - Configuration file for kubectl (Kubernetes command-line tool).
    - Contains credentials and cluster info to connect to your cluster.
    - Copied to your home directory so you can use kubectl as a normal user.

What do you do after kubeadm init?

1. **Set up kubeconfig**:
    - Copy the admin config so you can run kubectl commands.
2. **Deploy a pod network add-on**:
    - Essential for pod-to-pod communication across the cluster.
    - Popular options: Calico, Flannel, Weave Net.
3. **Add worker nodes**:
    - Run kubeadm join on worker machines with the token and certificate hash given by kubeadm init.
    - Worker nodes will then connect and join the cluster.

Why reset or clean up sometimes?

- If Kubernetes is already partially installed or running, kubeadm init will fail because:
    - Ports are busy.
    - Kubernetes static pod manifests exist.
    - Data directories are not empty.

Running:

```perl
sudo kubeadm reset
```

- cleans these so you can try initializing fresh.

Summary

Concept	What it means
kubeadm init	Command to create the Kubernetes control-plane node.
Control-plane components	API server, scheduler, controller manager, etcd.
Pod Network CIDR	IP address range for pods (containers)
kubeconfig	Config file for kubectl to connect to the cluster
Ports 6443, 2379, etc.	Ports used by Kubernetes system components
kubeadm reset	Command to undo cluster initialization and cleanup

Kubernetes Deployment & How It Works

🧠 Conceptual Overview

Kubernetes runs applications inside **containers** (usually Docker containers) and manages them through a cluster of machines (nodes). The **goal** is to automate deployment, scaling, and management.

🚀 Steps to Deploy and Work on Kubernetes

---

1. You Write a Manifest (YAML File)

- Describes the desired state of your application.
- Common Kubernetes resources:
    - Deployment: Defines how many replicas (pods) you want, what container to run, update strategy, etc.
    - Service: Exposes your app to other pods or the outside world.
    - Pod: Smallest deployable unit (usually created by a Deployment).
    - ConfigMap / Secret: Stores configuration and sensitive data.

📄 Example: deployment.yaml

yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: myapp:latest
          ports:
            - containerPort: 80
```

---

2. Apply the Manifest Using kubectl

bash

```bash
kubectl apply -f deployment.yaml
```

Kubernetes reads the manifest and **creates the desired state** in the cluster.

---

3. Kubernetes Scheduler Places Pods on Nodes

- The **Deployment** creates **ReplicaSets**, which create **Pods**.

- The **Scheduler** assigns pods to suitable **nodes** (machines in the cluster).
- The **kubelet** on each node downloads the container image and runs the pod.

---

4. Networking – Pod-to-Pod and External Access

- All pods in the cluster can reach each other (flat networking).
- To expose your app:
  - **ClusterIP**: internal access only.
  - **NodePort**: exposed on each node at a static port.
  - **LoadBalancer**: used in cloud environments (AWS, GCP, etc.).
  - **Ingress**: smart routing layer (like a reverse proxy).

🛠️ Example: Expose a service

bash

```
kubectl expose deployment my-app --type=NodePort --port=80
```

---

5. Kubernetes Continuously Maintains Desired State

- If a pod crashes or a node fails, Kubernetes **automatically restarts** or **reschedules** pods.
- If you change the deployment (e.g., new version), Kubernetes does a **rolling update**.

---

6. You Can Scale, Update, or Roll Back Easily

Scale Up:

```bash
kubectl scale deployment my-app --replicas=5
```

-

Update Image:

```bash
kubectl set image deployment/my-app my-container=myapp:v2
```

-

Roll Back:

```bash
kubectl rollout undo deployment my-app
```

-

⚙️ Components Involved in Deployment

Component	Role
kubectl	CLI to interact with the cluster
kube-apiserver	Accepts commands (via kubectl)
Controller Manager	Ensures desired state is achieved
Scheduler	Places new pods on appropriate nodes

kubelet	Node agent that runs the pods
Container Runtime (Docker/Containerd)	Runs the actual container

---

🧩 How Everything Works Together

1. You describe what you want in a YAML file (desired state).
2. Kubernetes accepts this instruction and stores it in etcd.
3. The controller checks the actual state and makes it match the desired state.
4. The scheduler decides where to run new pods.
5. kubelet runs the containers and monitors them.

🧪 Common Commands

Task	Command
Apply a deployment	kubectl apply -f deployment.yaml
See all pods	kubectl get pods
View deployment	kubectl get deployments
Check logs	kubectl logs <pod-name>
SSH into a container	kubectl exec -it <pod-name> -- bash
Delete a deployment	kubectl delete -f deployment.yaml

🧠 Summary Diagram

```css
[You] --> [kubectl apply -f app.yaml]
             ↓
    [API Server] --- stores in ---> [etcd]
             ↓
 [Controller Manager] checks for differences
             ↓
     [Scheduler] picks a node
             ↓
      [kubelet on Node] runs container
             ↓
     [Your App] is live!
```