Kubernetes Multi-Node Cluster with: KIND for CKA Exam Preparation, Troubleshooting Included

Connect with me: Amit Singh

Prepare for the CKA Exam by creating a Kubernetes cluster using KIND with 1 control plane and 3 worker nodes. This guide covers hands-on setup, application deployment, and troubleshooting skills essential for passing the certification.

How to Install Kubernetes Locally Using KIND

There are many ways to install Kubernetes locally: (it's important to check kubernetes version while giving CKA Exam)

- **Minikube**
- **k3s**
- **k3d**
- **Kind** (Kubernetes IN Docker)

In this guide, we are using **Kind**, because it's simple and spins up Kubernetes **inside Docker containers**.

Each container = **a node** in your Kubernetes cluster!
(Example: one container acts as a Control Plane, others as Worker Nodes.)

Steps to Set Up KIND

1. Prerequisites

- **Docker** installed
- (Optional) **Go 1.16+** if building from source.

  We already have Docker installed.

2. Install KIND (Kubernetes IN Docker)

- apt install kind

  Linux:
  Use release binaries or install from source.

  After installation, kind command should be available in your terminal.

3. Install kubectl (if not already installed)

kubectl = command-line tool to interact with Kubernetes clusters.

Create Your First KIND Cluster

You can create a simple cluster with:

```
kind create cluster
```

- By default, this uses the latest Kubernetes version.

  BUT WAIT!
  For CKA exam prep, you should match the Kubernetes version used in exams.
  (Current example: Kubernetes **v1.29**)

  So, create cluster with specific version:

```
kind create cluster --name cka-cluster-1 --image kindest/node:v1.29.4
```

✅ This sets up a **single node cluster** (control plane + worker in one).

4. Verify Cluster

```
kubectl get nodes
```

You should see **one node** with:

- STATUS = Ready
- ROLE = Control Plane

Create a Multi-Node Kubernetes Cluster

👉 We don't want only one node!
In real environments, control plane and worker nodes should be **separate**.

Create a YAML configuration file:

config.yaml

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
```

Create cluster with config file:

```
kind create cluster --name cka-cluster-2 --image kindest/node:v1.29.4 --config config.yaml
```

Now, your cluster will have:

- Control Plane node
- Worker nodes

5. Verify Multi-Node Cluster

```
kubectl get nodes
```

You should see **three nodes**, all in Ready state.

Why Multi-Node Clusters?

- In real-world Kubernetes, you have **multiple nodes** (workers) to distribute the workload.
- Control Plane manages the cluster; Worker Nodes run your applications (pods).
- Helps simulate real production architecture.

How to Create a Multi-Node Cluster?

(a) Create a YAML configuration file

Example multi-node-cluster.yaml:

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
```

```
nodes:
- role: control-plane
- role: worker
- role: worker
```

(b) Create the Cluster with Config

```
kind create cluster --name multi-node-cluster --config=multi-node-cluster.yaml
```

Verify Multi-Node Cluster

```
kubectl get nodes
```

You should see 3 nodes:

- control-plane
- workers
  Status: **Ready**

Check Node Roles

```
kubectl get nodes -o wide
```

You can check with this command:

- Node names
- Internal IP addresses
- Roles (master/control-plane, worker)

Delete the Cluster

```
kind delete cluster --name multi-node-cluster
```

Important to clean up when not needed.

Short Commands

- *Create multi-node cluster:* kind create cluster --name <name> --config=<config.yaml>
- *View nodes:* kubectl get nodes
- *View node details:* kubectl get nodes -o wide
- *Switch between clusters:* kubectl config use-context <context-name>
- *Delete multi-node cluster:* kind delete cluster --name <name>

Quick Notes for Multi-Node Cluster

- Kind **uses Docker containers** to simulate real Kubernetes nodes.
- **kubectl** is used to interact with any cluster — local or cloud (AWS, Azure, GCP, on-premises).
- Always **match Kubernetes versions** close to exam versions.
- Multi-node clusters **simulate production environments** better.

Remember!

- Local clusters are **great for learning**: you can control everything.
- Managed services (EKS, AKS, GKE) are good for **production**, but not for **deep learning** initially.
- Practice troubleshooting **locally first**

Important Points to Remember for Multi-Node Cluster

1. **Learn Kubernetes Locally First**

- Before using managed Kubernetes services like EKS, AKS, or GKE, practice Kubernetes installation and concepts *locally* to fully understand the architecture (Control Plane, Worker Nodes, etc.).
- Managed services hide many components (like the Control Plane), so troubleshooting and deep learning are limited.

2. Kind (Kubernetes in Docker)

- "Kind" spins up Kubernetes clusters **using Docker containers** as nodes.
- Each container acts like a **separate Kubernetes node** (Control Plane or Worker Node).

3. Prerequisites

- Docker must be installed.
- Go language (v1.16+) is needed only if you're building Kind from source (not required for most users).

- kubectl CLI tool must be installed for interacting with the Kubernetes cluster.

4. Installing Kind

- On Mac → brew install kind
- On Windows → choco install kind
- On Linux → download binaries manually.

5. Creating Clusters

- Simple Cluster → kind create cluster
- Custom Version → specify the Kubernetes version image when creating a cluster.
- Example: kind create cluster --image kindest/node:v1.29.4
- Custom Name → kind create cluster --name my-cluster
- Multi-node Cluster → Create a YAML config file specifying multiple nodes (control plane and workers).

  Example file:

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
```

- Create using config →
  kind create cluster --name my-multinode-cluster --config=config.yaml

6. kubectl Commands

- kubectl get nodes → shows nodes in the cluster.
- kubectl cluster-info → displays the cluster's API server and DNS info.
- Make sure your kubectl version matches (or is close to) your Kubernetes cluster version.

7. Cluster Context

- If multiple clusters are running, you can switch between them using kubectl config use-context <context-name>.
- Each cluster you create with Kind sets up its own context automatically.

8. Node Joining

- Worker nodes must “join” the control plane, which happens automatically in Kind setups.
- In real-world installations (e.g., kubeadm), manual joining is necessary.

Scenario-Based Questions for multi-node cluster

Scenario 1:
You have created a single-node Kind cluster, but you need multiple worker nodes for a project simulation. What would you do?
→ Create a multi-node cluster using a YAML config file and specify roles for control plane and workers.

Scenario 2:
After installing a Kubernetes cluster using Kind, kubectl get nodes shows nodes but the status is NotReady. How will you troubleshoot?
→ Check if Docker is running properly, verify if the CNI (Container Network Interface) plugins were installed successfully, and check kubectl describe node <node-name> for errors.

Scenario 3:
You created multiple Kind clusters but are unsure which one your kubectl is connected to. How do you check and switch?
→ Use kubectl config get-contexts to list contexts and kubectl config use-context <context-name> to switch.

Scenario 4:
Your Kubernetes cluster was set up with a wrong version (e.g., v1.30

instead of v1.29 for CKA exam practice). What will you do?
→ Delete the cluster using kind delete cluster --name <cluster-name> and recreate it with the correct image version.

Scenario 5:
You are trying to interact with your Kubernetes cluster but getting authentication errors. What steps will you take?
→ Ensure kubectl is pointing to the right context, your Docker engine is running, and check if the Kind cluster is up with docker ps.

Scenario 6:
You have a single-node cluster with a combined control plane and worker node. Why might this be bad in production?
→ Single node = Single point of failure. If that node fails, the whole cluster is down. Production needs **HA (High Availability)** with multiple control plane nodes and worker nodes.

Example of Advanced Multi-node Config

(Exposing port 30000, setting custom version)

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  image: kindest/node:v1.29.4
  extraPortMappings:
  - containerPort: 30000
    hostPort: 30000
- role: worker
  image: kindest/node:v1.29.4
- role: worker
  image: kindest/node:v1.29.4
```

Troubleshooting Multi-Node Clusters Cheat Sheet ❌

1. Nodes stuck in **NotReady** state

Symptoms:
kubectl get nodes shows worker nodes in **NotReady**.

Possible Reasons:

- kubelet is not healthy inside the node.
- Missing network connectivity (CNI issues).

  Fix:

- Restart the node/container:docker restart <node-container-name>
- Check kubelet logs inside the node:docker exec -it <node-container-name> journalctl -u kubelet

2. Unable to pull images for pods

Symptoms:
Pods are stuck in **ImagePullBackOff**.

Possible Reasons:

- Wrong image name.
- No internet access inside the node.

Fix:

- Double-check image name/tag.
- If using a private registry, add credentials.
- Or pre-pull the image manually into nodes:
- docker pull <image-name>

3. Pod scheduled on wrong node

Symptoms:
You expected pods to run on a worker node, but it went to control-plane.

Reason:

- Taints/Tolerations misconfigured.
- No nodeSelector or Affinity rules used.

Fix:

- Use kubectl describe nodes to check taints.
- Add tolerations or correct nodeSelector in your Deployment YAML.

4. Port Forwarding Issues

Symptoms:
kubectl port-forward doesn’t work or gives connection refused.

Reason:

- Wrong pod name.
- Service/Pod not ready.
- Pod crashed.

Fix:

- Ensure pod is running:kubectl get pods
- Check pod logs:kubectl logs <pod-name>
- Then retry port-forward: kubectl port-forward pod/<pod-name> 8080:80

5. Cluster Creation Fails

Symptoms:
kind create cluster fails with errors.

Reason:

- Old/dangling Docker containers.
- Wrong YAML config.

- Port conflicts (if exposing ports).

Fix:

- Clean up Docker:
- docker system prune -af
- Validate your YAML config properly.
- Change host ports if conflict detected.

6. Cannot reach services inside cluster

Symptoms:
Service IP unreachable.

Reason:

- Service misconfigured.
- Pod label mismatch in selector.
- Missing CNI (Container Network Interface).

Fix:

- Validate service YAML.
- Check selector labels match pods.
- Restart kind cluster if networking failed initially.

7. kubeconfig / Context not switching

Symptoms:
Running kubectl commands hit wrong cluster.

Reason:

- Wrong context selected.

Fix

- List contexts: kubectl config get-contexts
- Switch context:kubectl config use-context kind-<your-cluster-name>

Bonus Tip:

👉 Always inspect the running kind cluster containers if you feel something is off: *docker ps*

You can **exec** into any node like this to troubleshoot: *docker exec -it <node-container-name> bash*

Quick Practical Task

Create a 1 Control Plane + 3 Worker cluster. Deploy an nginx deployment. Check which node your pod is running on.

Step 1: Create Kind Config for 1 Control Plane + 3 Workers

Create a YAML file named kind-multinode.yaml:

```
# kind-multinode.yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
 - role: control-plane
 - role: worker
 - role: worker
 - role: worker
```

Step 2: Create the Cluster

Run:

```
kind create cluster --name multinode-cluster --config kind-multinode.yaml
```

Wait for a few minutes until the cluster is ready! 🚀

Step 3: Check All Nodes

Check if nodes are ready:

```
kubectl get nodes
```

You should see 4 nodes (1 control-plane + 3 workers).

Step 4: Deploy Nginx Deployment

Create an nginx Deployment:

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

OR apply using YAML:

```
# nginx-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx
```

Apply it:

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

Step 5: Verify Pod is Running

Check pod status:

```
kubectl get pods -o wide
```

Quick Tips:

- **Control-plane** node usually has a taint that stops workloads from scheduling there.
- Pods will automatically go to available **worker nodes**.

Connect with me: LinkedIn “Amit Singh”