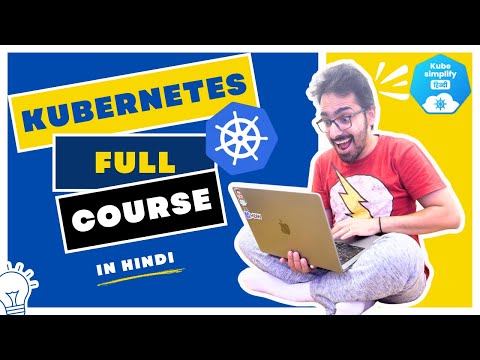
<https://www.youtube.com/watch?v=5NXmbV50IxE> -- Kubernetes Course in Hindi (13 Hours) | Full Hands-On Experience

[](https://www.youtube.com/watch?v=5NXmbV50IxE)

<https://www.youtube.com/watch?v=MGCF6slXG0w>

[](https://www.youtube.com/watch?v=MGCF6slXG0w)

ALMALinux -- Alma Linux is a free and open source linux distribution

Java Programming for Complete Begineers -- By in28Minutes official

Dr. Angela Yu, Developer and Lead Instructor - Udemy Instructor Partner

Terraform on Azure with IaC DevOps SRE | Real-World 25 Demos by Kalyan Reddy Daida

GITLAB CI --- How to integrate.

Ansible for automating tasks.

Monitoring in Linux -- last command is used.

<https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/?couponCode=24T4MT120424>

# **Certified Kubernetes Administrator (CKA) with Practice Tests**

By [**Mumshad Mannambeth**](https://www.udemy.com/user/mumshad-mannambeth/)

<https://www.udemy.com/course/aws-certified-devops-engineer-professional-hands-on/?couponCode=24T4MT120424>

# **AWS Certified DevOps Engineer Professional 2025 - DOP-C02**

<https://www.udemy.com/course/aws-certified-devops-engineer-professional-hands-on/?couponCode=24T4MT120424#instructor-1>

Admiralty is a Kubernetes plugin designed for **multi-cluster management**, allowing users to deploy applications across multiple Kubernetes clusters efficiently. It simplifies the process of managing workloads, traffic routing, and disaster recovery across clusters, making it particularly useful for organizations operating in diverse environments, including cloud and on-premises setups.

## **Key Features of Admiralty**

* **Multi-Cluster Deployment**: Deploy applications across various clusters seamlessly.
* **Traffic Optimization**: Ensures low latency and instant failover.
* **Centralized Management**: Provides a single control plane for managing user access and audit logs.
* **Flexible Architecture**: Supports both centralized and decentralized cluster topologies.
* **Disaster Recovery**: Facilitates active-active disaster recovery strategies.

## **Steps to Install and Configure Admiralty Plugin in Kubernetes**

### **Prerequisites**

1. Ensure you have a working Kubernetes cluster.
2. Install kubectl and helm on your local machine.
3. Have access to the Kubernetes API with appropriate permissions.

### **Installation Steps**

1. **Add the Admiralty Helm Repository**:

helm repo add admiralty oci://public.ecr.aws/admiralty/admiralty  
helm repo update

1. **Install Admiralty in Each Cluster**: Use the following command to install Admiralty in your desired clusters:

for CLUSTER\_NAME in <your-cluster-names>; do  
 sudo helm install admiralty oci://public.ecr.aws/admiralty/admiralty \  
 --kube-context kind-$CLUSTER\_NAME \  
 --namespace admiralty --create-namespace \  
 --version 0.16.0 --wait --debug  
done

Replace <your-cluster-names> with the actual names of your clusters.

1. **Set Up Cross-Cluster Authentication**:
   1. Create a Kubernetes service account in each workload cluster for the management cluster.
   2. Generate a token for this service account.
   3. Obtain a routable Kubernetes API address from the management cluster.
   4. Prepare a kubeconfig file using the token and address, then save this kubeconfig as a secret in the management cluster.
2. **Configure Workloads**: Annotate your pods with multicluster.admiralty.io/elect="" to enable workload distribution across clusters.
3. **Verify Installation**: Check that all components are running correctly by executing:

kubectl get pods -n admiralty

1. **Monitor and Manage**: Use kubectl commands to monitor the status of your deployments and manage workloads across clusters effectively.

By following these steps, you can successfully install and configure the Admiralty plugin in your Kubernetes environment, enabling efficient multi-cluster management and deployment capabilities

Calico is an open-source networking and network security solution designed for Kubernetes and other container orchestration platforms. It provides robust networking capabilities, including IP address management, network policies, and security features, making it a popular choice for organizations looking to enhance their Kubernetes environments.

## **Uses of Calico in Kubernetes**

* **Networking**: Calico enables a flat networking model where each pod receives its own IP address, allowing direct communication without the need for Network Address Translation (NAT).
* **Network Policies**: It supports Kubernetes network policies and extends them with additional features for fine-grained control over traffic between pods, enhancing security through microsegmentation.
* **Performance**: Calico can leverage eBPF (Extended Berkeley Packet Filter) or standard Linux networking for high-performance networking, suitable for large-scale deployments.
* **Multi-Tenancy**: It facilitates network segmentation in multi-tenant environments, allowing organizations to isolate workloads and enforce strict security policies.

## **Installation and Configuration of Calico in Kubernetes**

### **Prerequisites**

* A running Kubernetes cluster.
* Access to kubectl command-line tool.

### **Installation Steps**

1. **Download the Calico Manifest**: Use the following command to download the Calico manifest file:

curl <https://docs.projectcalico.org/manifests/calico.yaml> -O

1. **Apply the Manifest**: Deploy Calico by applying the downloaded manifest:

kubectl apply -f calico.yaml

1. **Verify Installation**: Check that the Calico node agent is running on all nodes:

kubectl get pods -n kube-system -l k8s-app=calico-node

1. **Create a Sample Deployment**: For demonstration, create a simple Nginx deployment:

kubectl create deployment nginx --image=nginx

1. **Expose the Deployment**: Expose the Nginx deployment as a ClusterIP service:

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

1. **Verify Pod Networking**: Check that Calico has assigned an IP address to the Nginx pod and set up necessary routes:

kubectl get pods -l app=nginx -o wide

### **Example of Using Network Policies**

To demonstrate Calico's network policy capabilities, you can create a policy that restricts access to the Nginx pod:

1. **Define a Network Policy**: Create a YAML file named nginx-network-policy.yaml with the following content:

apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
 name: deny-all-ingress  
 namespace: default  
spec:  
 podSelector:  
 matchLabels:  
 app: nginx  
 policyTypes:  
 - Ingress  
 ingress: []

1. **Apply the Network Policy**: Apply the policy to enforce restrictions on incoming traffic to the Nginx pod:

kubectl apply -f nginx-network-policy.yaml

This configuration establishes a basic setup of Calico in your Kubernetes environment, showcasing its networking capabilities and how to implement network policies for enhanced security.

Kubernetes with Kind

Etcd stores the data in Key-Value format. And stores the status of pods.

All the request comes to API Server. As scheduler contineously runs and check the status of nodes. It will check which node is used less. So he can transfer new request to him.

API-Gateway talks to etcd and tell them to create pods. Kubeproxy is used for running POD and also check its status. IP tables handle by kubeproxy.

Application runs inside PODs.

Deployments controlled by Deployment Controller. ReplicaSet controlled by Replicaset controller.

Install “Kind” on Windows for Kubernetes.

To install Kind (Kubernetes in Docker) on Windows and create a running pod with master-slave architecture, follow these detailed steps:

## **Step 1: Install Prerequisites**

1. **Install Docker**:
   1. Download Docker Desktop for Windows from the official Docker website.
   2. Follow the installation prompts, ensuring that WSL 2 is enabled if you are using Windows 10 Home.
2. **Install Chocolatey** (optional but recommended):
   1. Open PowerShell as an administrator and run:Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))

## **Step 2: Install Kind**

### **Method 1: Using Chocolatey**

1. Open a new PowerShell window and run:choco install kind -y

### **Method 2: Direct Download**

1. In PowerShell, run the following command to download Kind:curl.exe -Lo kind-windows-amd64.exe <https://kind.sigs.k8s.io/dl/v0.14.0/kind-windows-amd64>
2. Move the executable to a directory in your PATH:Move-Item .\kind-windows-amd64.exe C:\some-dir-in-your-PATH\kind.exe

## **Step 3: Verify Installation**

Confirm that Kind is installed correctly by running:

kind --version

## **Step 4: Create a Kubernetes Cluster**

To create a Kubernetes cluster with Kind, execute:

kind create cluster

This command will create a single-node cluster named "kind" by default.

## **Step 5: Create a Pod**

To create a running pod, you can use a simple YAML configuration file. Create a file named pod.yaml with the following content:

apiVersion: v1  
kind: Pod  
metadata:  
 name: my-pod  
spec:  
 containers:  
 - name: my-container  
 image: nginx  
 ports:  
 - containerPort: 80

Then, apply this configuration to create the pod:

kubectl apply -f pod.yaml

## **Step 6: Create a Master-Slave Setup**

To set up a master-slave architecture (for example, using two pods), you can define a Deployment with replicas. Create a file named deployment.yaml with the following content:

apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: my-deployment  
spec:  
 replicas: 2 # This creates one master and one slave pod  
 selector:  
 matchLabels:  
 app: my-app  
 template:  
 metadata:  
 labels:  
 app: my-app  
 spec:  
 containers:  
 - name: my-container  
 image: nginx  
 ports:  
 - containerPort: 80

Apply this deployment configuration with:

kubectl apply -f deployment.yaml

## **Step 7: Verify Pods**

To check if your pods are running, use the following command:

kubectl get pods

You should see your master and slave pods listed.

## **Conclusion**

You have successfully installed Kind on Windows, created a Kubernetes cluster, deployed a pod, and set up a master-slave architecture using Kubernetes Deployments. This setup is ideal for local development and testing purposes.