KodeKloud

Container orchestration

Kubernetes Concepts Explained!

With Docker, you can run a single instance of the application with a simple Docker run command. For instance, to run a Node JS-based application, you run `docker run nodejs` command. But that's just one instance of your application on one Docker host.

What happens when the number of users increases and that instance is no longer able to handle the load? You deploy an additional instance of your application by running the Docker run command multiple times. However, you have to keep a close watch on the load and performance of your application and deploy additional instances yourself. And not just that, you have to keep a close watch on the health of these applications.

Learn more about Docker from this blog: [How to Run a Docker Image as a Container?]

That's why you need container orchestration. In this blog, we'll cover what container orchestration entails, orchestration solutions, and the differences between Docker and Kubernetes.

Kubernetes Concepts Explained

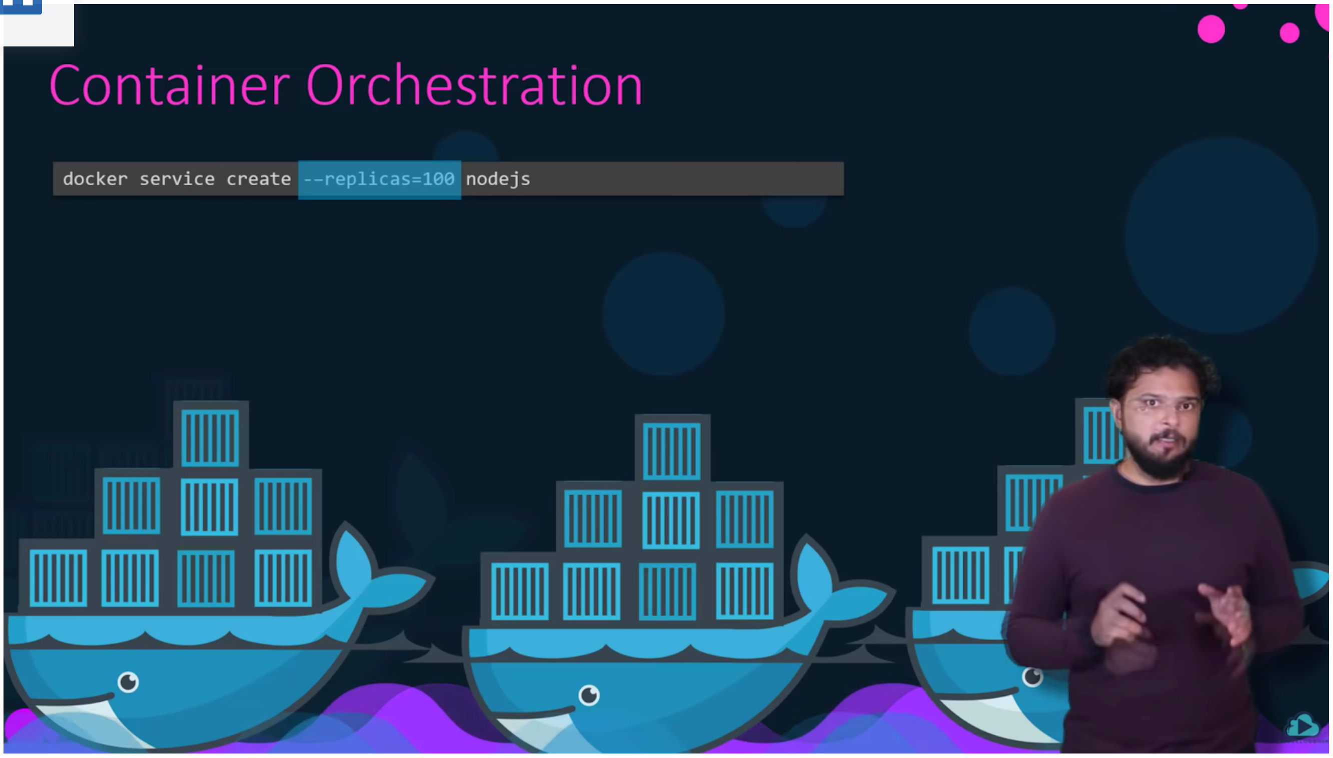
Subscribe

Let's start by diving into what container orchestration entails.

Container Orchestration

Container orchestration is the process of managing the deployment, scaling, and operation of multiple containers in a distributed computing environment. Container orchestration tools help automate the management of containers, making it easier to deploy and scale containerized applications.

Typically, a container orchestration solution consists of multiple Docker hosts that can host containers. That way, even if one fails, the application is still accessible through the others. The container orchestration solution easily allows you to deploy hundreds or thousands of instances of your application with a single command.



```
docker service create --replicas=100 nodejs
```

Some orchestration solutions can help you automatically scale up the number of instances when users increase and scale down the number of instances when the demand decreases. Some solutions can even help you automatically add additional hosts to support the user load.

And, not just clustering and scaling, the container orchestration solutions also provide support for advanced networking between these containers across different hosts. As well as load-balancing user requests across different hosts. They also provide support for sharing storage between the host, as well as support for configuration management and security within the cluster.

Learn more about Kubernetes networking: Kubernetes Networking: Fundamental Concepts Explained (2023)

Orchestration solutions

There are multiple container orchestration solutions available today – Docker has Docker Swarm, Kubernetes from Google, and Mesos from Apache.

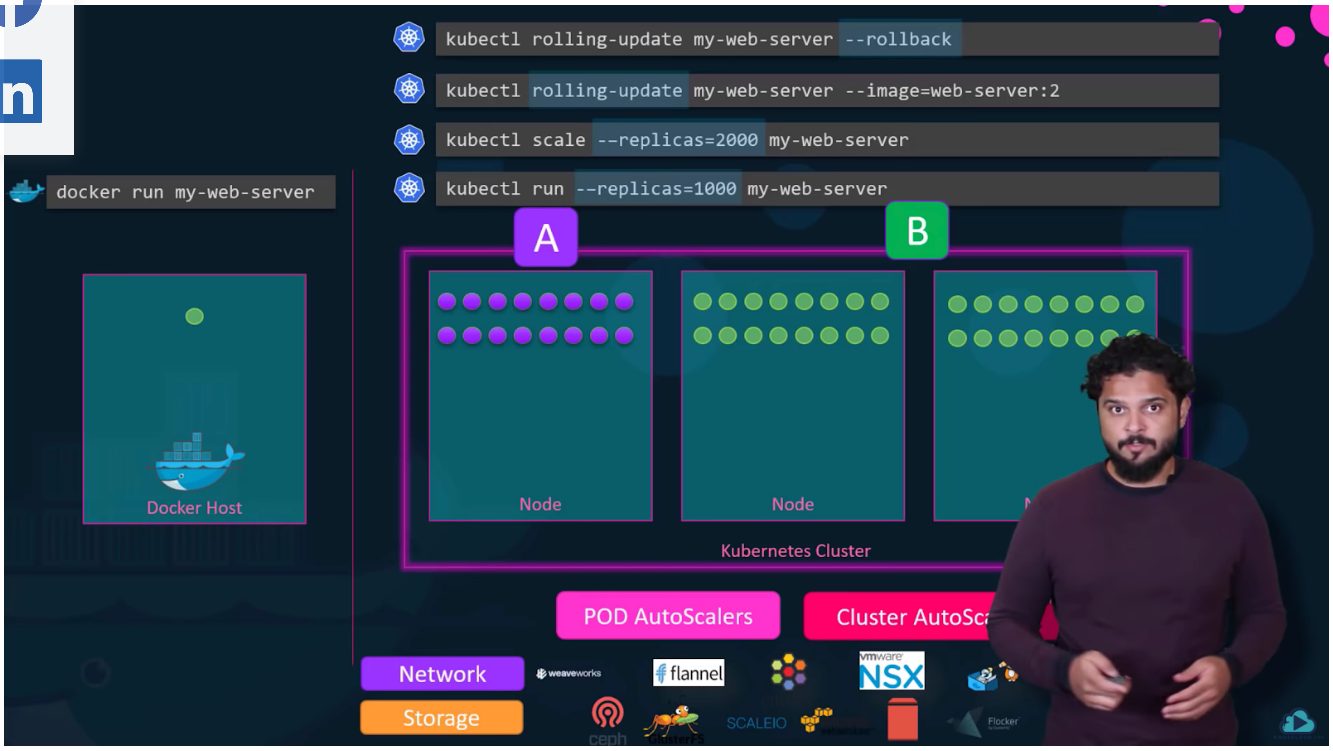
- **Docker Swarm** is really easy to set up and get started. It lacks some of the advanced auto-scaling features required for complex production-grade applications.
- **Mesos** is quite difficult to set up and get started but supports many advanced features.
- **Kubernetes**, arguably the most popular of all, is a bit difficult to set up and get started but provides a lot of options to customize deployments and has support for many different vendors. Kubernetes is now supported on all public cloud service providers like GCP, Azure, and AWS, and the Kubernetes project is one of the top-ranked projects on GitHub.

Want a comprehensive comparison between Kubernetes and Docker Swarm? Check out this blog: Kubernetes vs. Docker Swarm: A Comprehensive Comparison (2023)

With Kubernetes, you can run 1000 instances of the same application with a single command. It can also scale up to 2000 instances with one command. It can even be configured to do this automatically so that instances and the infrastructure itself scale up and down based on user load.

Kubernetes can upgrade these 2000 application instances in a rolling upgrade fashion or one at a time. If something goes wrong, it can help you roll back the upgrades with a single command. Kubernetes can also help you test new features of your application by only upgrading a percentage of these instances through AB testing methods.

The Kubernetes open architecture provides support for many different network and storage vendors. Any network or storage brand that you can think of has a plugin for Kubernetes. Kubernetes supports a variety of authentication and authorization mechanisms. All major cloud service providers also have native support for Kubernetes.



```
kubectl rolling-update my-web-server --rollback
kubectl rolling-update my-web-server --image=web-server:2
kubectl scale --replicas=2000 my-web-server
kubectl run --replicas=1000 my-web-server
docker run my-web-server
```

The Relationship Between Docker and Kubernetes

The container runtime is a key component in Kubernetes as it is responsible for running and managing containers. Kubernetes supports several container runtimes, including Docker, CRI-O, and containerd. The container runtime is responsible for starting and stopping containers, managing their lifecycle, and providing access to container resources such as storage and networking.

Kubernetes uses the container runtime interface (CRI) to communicate with the container runtime, which allows it to manage containers regardless of the underlying runtime technology being used. Essentially, the container runtime is the engine that powers Kubernetes and enables it to manage containerized applications.

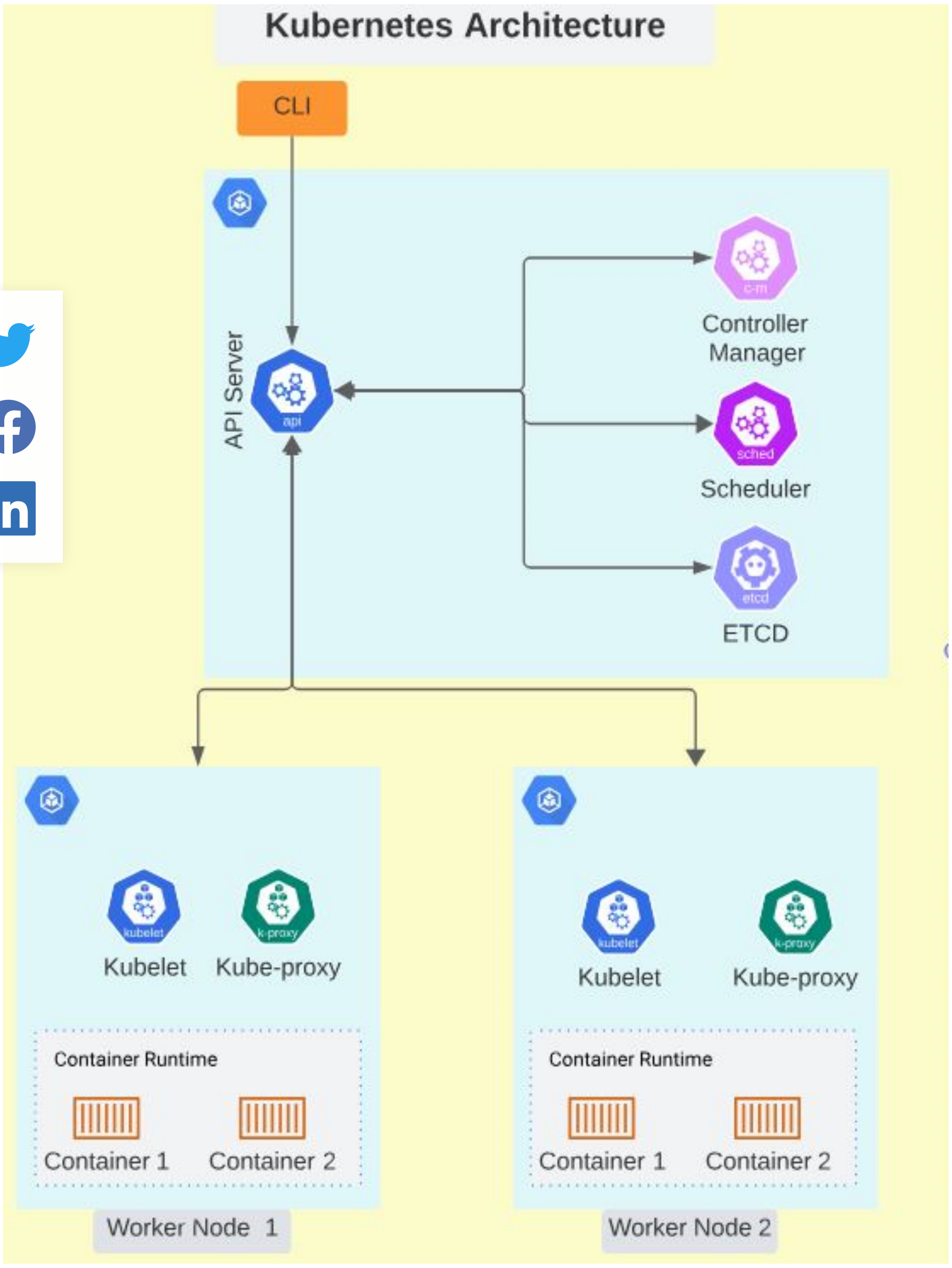
Kubernetes Architecture

In Kubernetes, a physical or virtual node runs applications and services. A cluster has a Master node and worker nodes. Each worker node has its own components, including the kubelet, kube-proxy, and container runtime that facilitate the execution of containers. The Master is a node that carries the Kubernetes control plane components.

Nodes communicate with the master components to receive instructions on which containers to run and how to run them. It's important to have multiple worker nodes in a cluster to ensure high availability and fault tolerance. If a node fails, the containers running on it can be rescheduled on other nodes to maintain application availability.

A Kubernetes cluster consists of several components in different nodes, each with its own specific role. The main components of a Kubernetes cluster are:

- **Master Components:** The master components are responsible for managing the Kubernetes cluster. They include the API server, etcd, the scheduler, and the controller manager.
- **Node Components:** The node components are responsible for running the applications and services in the Kubernetes cluster. They include the kubelet, kube-proxy, and container runtime such as Docker.
- **Add-Ons:** Add-ons are optional components that can be added to the Kubernetes cluster to provide additional functionality. Examples of add-ons include the Kubernetes dashboard, DNS, and the container network interface (CNI).



Kubernetes architecture

When you install Kubernetes on a system, you're actually installing its different components. Below is an explanation of Kubernetes' main components.

API server

The API server is a central management point for the Kubernetes cluster and is responsible for exposing the Kubernetes API. It acts as a gateway between the Kubernetes control plane and the cluster nodes, enabling users and administrators to interact with the cluster using various Kubernetes tools and APIs.

The API server provides a secure and scalable interface for managing the Kubernetes cluster, and it is a critical component of the Kubernetes architecture. It accepts and processes requests from various sources, such as kubectl, the Kubernetes dashboard, and other Kubernetes components, and then communicates with the other components of the control plane to fulfill those requests.

etcd

etcd is a distributed key-value store that is used as the primary data store for Kubernetes. It stores the configuration data and the state of the Kubernetes cluster, including information about the nodes, Pods, services, and other objects in the cluster. The etcd data store is highly available and fault-tolerant, which means that it can continue to function even if some of the nodes in the cluster fail. The etcd cluster also provides strong consistency guarantees, which ensures that all nodes in the cluster have the same view of the data at any given time.

The data stored in etcd is accessed and manipulated by the Kubernetes control plane components, such as the API server, the scheduler, and the controller manager. For example, when a new Pod or service is created in the cluster, the API server stores its configuration data in etcd, which is then used by other components to schedule the Pod, manage its life cycle, and ensure that it is running correctly.

Scheduler

In Kubernetes, the scheduler is responsible for assigning Pods to nodes in the cluster based on the resource requirements of the Pod and the available resources on the node. It ensures that the Pods are distributed across the nodes in a way that maximizes resource utilization and avoids overloading any particular node.

The scheduler continuously monitors the state of the cluster and identifies new Pods that need to be scheduled. It then selects a suitable node for each Pod based on factors such as the available CPU, memory, and storage resources on the node, as well as any specific requirements or constraints specified by the Pod.

Once the scheduler has identified a suitable node for a Pod, it updates the API server with the scheduling decision, which triggers the creation of the Pod on the selected node. The scheduler then continues to monitor the state of the cluster and makes scheduling decisions as needed to ensure that the Pods are running on the most suitable nodes.

Controllers

In Kubernetes, controllers are responsible for maintaining the desired state of the cluster by managing the lifecycle of various objects, such as Pods, services, and replication controllers. There are several built-in controllers in Kubernetes, including the Replication Controller, ReplicaSet, Deployment, StatefulSet, and DaemonSet.

- The Replication Controller ensures that the specified number of replicas of a Pod are running at all times. If a Pod fails or is terminated, the Replication Controller creates a new Pod to maintain the desired number of replicas.
- The ReplicaSet is similar to the Replication Controller but provides more advanced features, such as the ability to specify more complex deployment strategies.
- The Deployment controller manages the rollout and rollback of changes to a set of Pods. It ensures that the desired state of the deployment is achieved by creating new replicas and terminating old ones as needed.
- The StatefulSet controller manages stateful applications by ensuring that Pods are created and terminated in a specific order. It also provides stable network identities and persistent storage for each Pod.
- The DaemonSet controller ensures that a specific set of Pods is running on all nodes in the cluster. It is often used for system-level services, such as logging or monitoring agents.

Controllers continuously monitor the state of the cluster and make changes as needed to ensure that the desired state is achieved. They use the Kubernetes API to interact with the cluster and communicate with other controllers to coordinate their actions.

Kubelet

In Kubernetes, a kubelet is an agent that runs on each node in the cluster and is responsible for managing the state of the node. It ensures that the containers are running and healthy by starting, stopping, and restarting them as needed.

The kubelet communicates with the Kubernetes API server to receive instructions about which containers to run and how to configure them. It monitors the state of the containers and reports any issues back to the API server.

The kubelet also manages the storage and networking for the containers. It mounts the required volumes to the containers and sets up the network interfaces so that the containers can communicate with each other and with the outside world.

In addition to managing the containers, the kubelet also monitors the health of the node itself. It reports back to the API server about the node's status, including its available resources, such as CPU and memory.

Container Runtime

In Kubernetes, a container runtime is responsible for running the containers that make up the applications in the cluster. It provides an environment for the containers to run in and manages their lifecycle, including starting, stopping, and monitoring them.

Kubernetes supports several container runtimes, including Docker, containerd, and CRI-O. The choice of container runtime depends on the specific requirements of the application and the environment in which it will run.

The container runtime is responsible for creating and managing the container images, which are the blueprints for the containers. It pulls the images from a container registry, such as Docker Hub or Google Container Registry, and stores them locally on the node.

When a Pod is scheduled on a node, the container runtime creates a container for each container in the Pod. It mounts the required volumes and sets up the network interfaces so that the containers can communicate with each other and with the outside world.

The container runtime also monitors the health of the containers and reports any issues back to the Kubernetes control plane. If a container fails, the container runtime can automatically restart it or take other actions to ensure that the application continues to run correctly.

Kubectl

In Kubernetes, kubectl is a command-line tool that is used to interact with the Kubernetes API server and manage the state of the cluster. It allows developers and administrators to deploy, inspect, and manage applications running on the cluster.

With kubectl, you can create, update, and delete Kubernetes objects, such as Pods, services, deployments, and config maps. You can also view the status of the objects and get detailed information about their configuration and health.

Kubectl also provides a powerful interface for managing the cluster itself. You can use it to view and modify the configuration of the API server, scheduler, and other control plane components. You can also view and modify the configuration of the nodes in the cluster, such as adding labels, taints, and tolerations.

In addition to its core functionality, kubectl can be extended with plugins and custom commands. This allows developers and administrators to add new features and automate common tasks.

Conclusion

Well, that’s all we have for now. A quick introduction to Kubernetes and its architecture. These components work together to provide a powerful and flexible platform for deploying, scaling, and managing containerized applications.

Explore our Kubernetes courses:

- Kubernetes for the absolute beginners
- Certified Kubernetes Administrator(CKA)
- Kubernetes – Absolute beginner to expert
- Certified Kubernetes application developer(CKAD)

More on Kubernetes:

- Getting Started with Kubernetes: A Beginners Guide
- 6 Features of Kubernetes Every Beginner Must Know
- 10 Best Kubernetes Courses to Start Learning Online in 2023
- Kubernetes Learning Curve: How Hard It is to Learn K8s?
- What Is Kubernetes Ingress?
- Kubernetes Architecture Explained: Overview for DevOps Enthusiasts
- Kubernetes Terminology: Pods, Containers, Nodes & Clusters

Mumshad Mannambeth
Sep 21, 2021 • 9 min read

Member discussion

0 comments

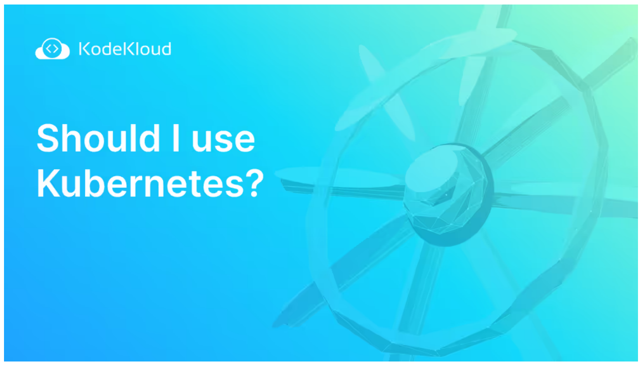
Start the conversation

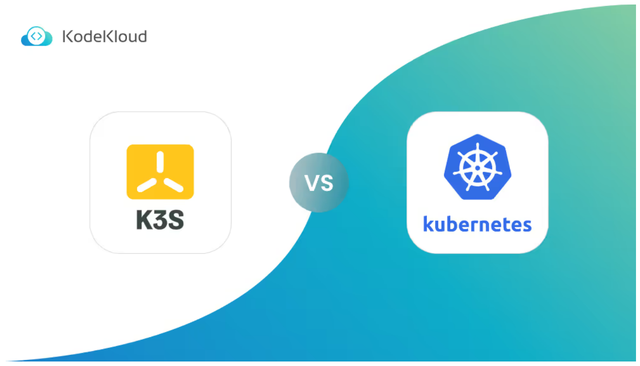
Become a member of **DevOps Blog** to start commenting.

Sign up now

Already a member? Sign in

Recommended





Kubernetes

Should I Use Kubernetes?

In the DevOps world, container orchestration is often synonymous with…

17, 2024 — 7 min read

K3s

K3s vs K8s: What are the Differences & Use Cases

key advantage of K3s over K8s is its focus on simplifying day-to-day…

Oct 16, 2023 — 9 min read

LEARNING PATHS

DevOps

Kubernetes

Docker

Linux

IaC

AWS

GCP

Azure

COURSES

Certified Kubernetes Administrator

Certified Kubernetes Application Developer

Certified Kubernetes Security Specialist

AWS Cloud Practitioner

Microsoft Azure Solutions Architect Expert

Microsoft Azure Administrator

COMMUNITY

Join our community

Teach with Us

Write with Us

Ambassadors

Academia

Affiliates

ABOUT

About Us

Success Stories

Our Values

Careers at KodeKloud

Privacy Policy

Terms of Service

Business Terms of Service

HELP

Contact Us

Support

Give us feedback

Request a Course

YOUR ACCOUNT

Sign In

Register

KodeKloud

Zaurac Technologies Pte Ltd
14 Robinson Road #08-01A
Singapore 048545

Zaurac Consulting Pvt Ltd

MI Zone, 446, Mangattuparamba,

Kalliasseri, Kannur, Kerala 670 567