
प्रगत संगणन विकास केंद्र
सी डैक
CDAC
॥ ज्ञानादेव तु कैवल्यम् ॥
CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

Kubernetes

- Kubernetes is a powerful open-source system, initially developed by Google, for managing containerized applications in a clustered environment. It aims to provide better ways of managing related, distributed components and services across varied infrastructure.
- It is a platform designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability.

- As a Kubernetes user, you can define how your applications should run and the ways they should be able to interact with other applications or the outside world. You can scale your services up or down, perform graceful rolling updates, and switch traffic between different versions of your applications to test features or rollback problematic deployments.

Kubernetes Architecture

- Kubernetes can be visualized as a system built in layers, with each higher layer abstracting the complexity found in the lower levels.
- At its base, Kubernetes brings together individual physical or virtual machines into a cluster using a shared network to communicate between each server. This cluster is the physical platform where all Kubernetes components, capabilities, and workloads are configured.
- The machines in the cluster are each given a role within the Kubernetes ecosystem. One server (or a small group in highly available deployments) functions as the **master** server. This server acts as a gateway and brain for the cluster by exposing an API for users and clients, health checking other servers, deciding how best to split up and assign work (known as “scheduling”), and orchestrating communication between other components. The master server acts as the primary point of contact with the cluster and is responsible for most of the centralized logic Kubernetes provides.

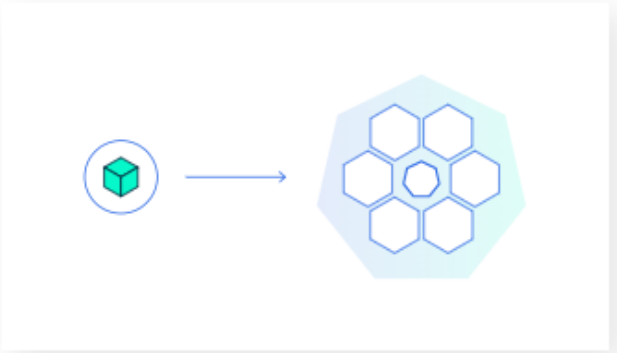
- The other machines in the cluster are designated as **nodes**: servers responsible for accepting and running workloads using local and external resources. To help with isolation, management, and flexibility, Kubernetes runs applications and services in **containers**, so each node needs to be equipped with a container runtime (like Docker or rkt). The node receives work instructions from the master server and creates or destroys containers accordingly, adjusting networking rules to route and forward traffic appropriately.

What can Kubernetes do for you?

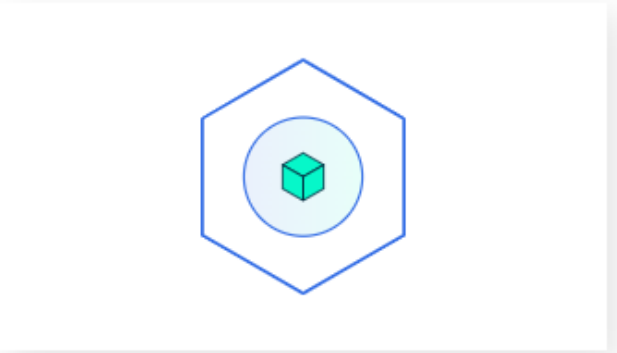
- With modern web services, users expect applications to be available 24/7, and developers expect to deploy new versions of those applications several times a day.
- Containerization helps package software to serve these goals, enabling applications to be released and updated without downtime.
- Kubernetes helps you make sure those containerized applications run where and when you want, and helps them find the resources and tools they need to work.
- Kubernetes is a **production-ready, open source platform** designed with Google's accumulated experience in container orchestration, combined with best-of-breed ideas from the community.

Kubernetes Basics Modules

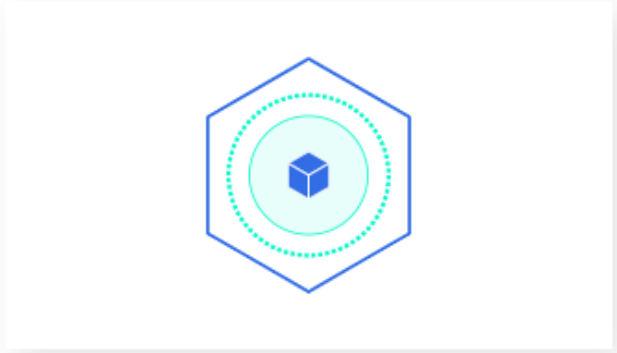
1. Create a Kubernetes cluster



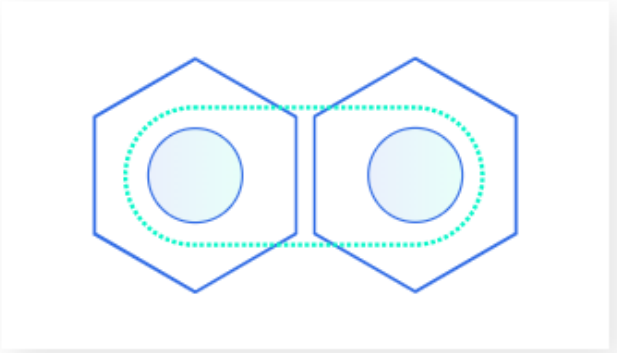
2. Deploy an app



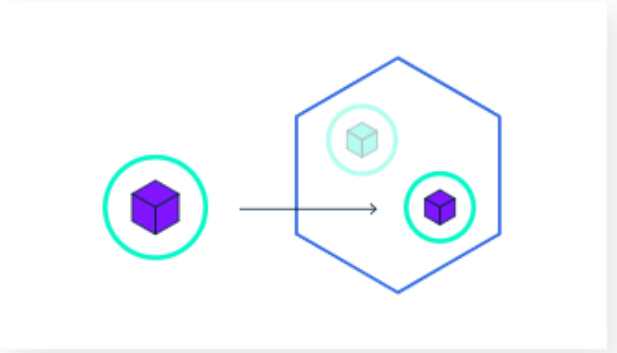
3. Explore your app



4. Expose your app publicly



5. Scale up your app



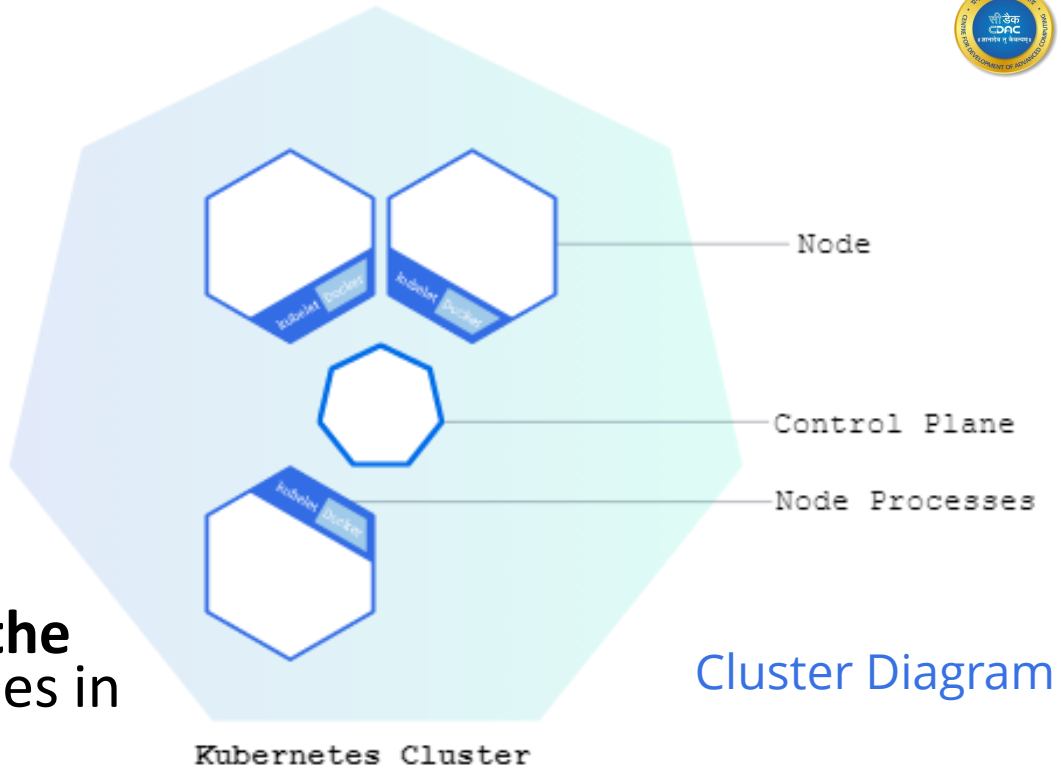
6. Update your app

Kubernetes Clusters

- **Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.** The abstractions in Kubernetes allow you to deploy containerized applications to a cluster without tying them specifically to individual machines.
- Containerized applications are more flexible and available than in past deployment models, where applications were installed directly onto specific machines as packages deeply integrated into the host. **Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way.** Kubernetes is an open-source platform and is production-ready.

Cluster

- A Kubernetes cluster consists of two types of resources:
  - The **Control Plane** coordinates the cluster
  - **Nodes** are the workers that run applications
- **The Control Plane is responsible for managing the cluster.** The Control Plane coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.
- **A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.** Each node has a Kubelet, which is an agent for managing the node and communicating with the Kubernetes control plane. The node should also have tools for handling container operations, such as containerd or Docker.



Cluster Diagram

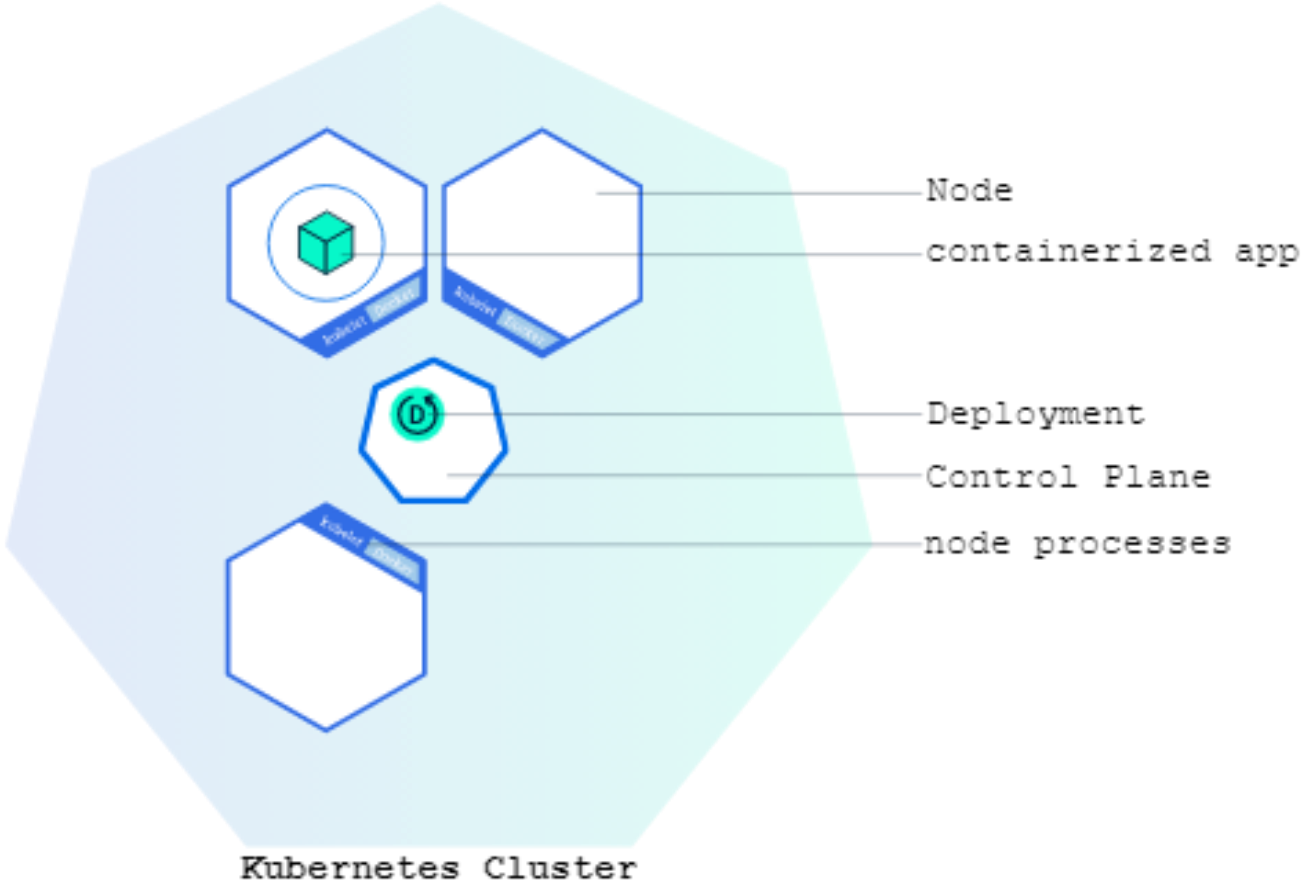
Create a Cluster using Minikube

- When you deploy applications on Kubernetes, you tell the control plane to start the application containers. The control plane schedules the containers to run on the cluster's nodes. **The nodes communicate with the control plane using the Kubernetes API**, which the control plane exposes. End users can also use the Kubernetes API directly to interact with the cluster.
- A Kubernetes cluster can be deployed on either physical or virtual machines. To get started with Kubernetes development, you can use Minikube. Minikube is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only one node.
- The Minikube CLI provides basic bootstrapping operations for working with your cluster, including start, stop, status, and delete.

Deploy an App Using kubectl

- Once you have a running Kubernetes cluster, you can deploy your containerized applications on top of it. To do so, you create a Kubernetes **Deployment** configuration. The Deployment instructs Kubernetes how to create and update instances of your application. Once you've created a Deployment, the Kubernetes control plane schedules the application instances included in that Deployment to run on individual Nodes in the cluster.
- Once the application instances are created, a Kubernetes Deployment Controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. **This provides a self-healing mechanism to address machine failure or maintenance.**

Deploying your first app on Kubernetes



Deploy an App

- ***Applications need to be packaged into one of the supported container formats in order to be deployed on Kubernetes***
- You can create and manage a Deployment by using the Kubernetes command line interface, **Kubectl**. Kubectl uses the Kubernetes API to interact with the cluster.
- When you create a Deployment, you'll need to specify the container image for your application and the number of replicas that you want to run. You can change that information later by updating your Deployment.

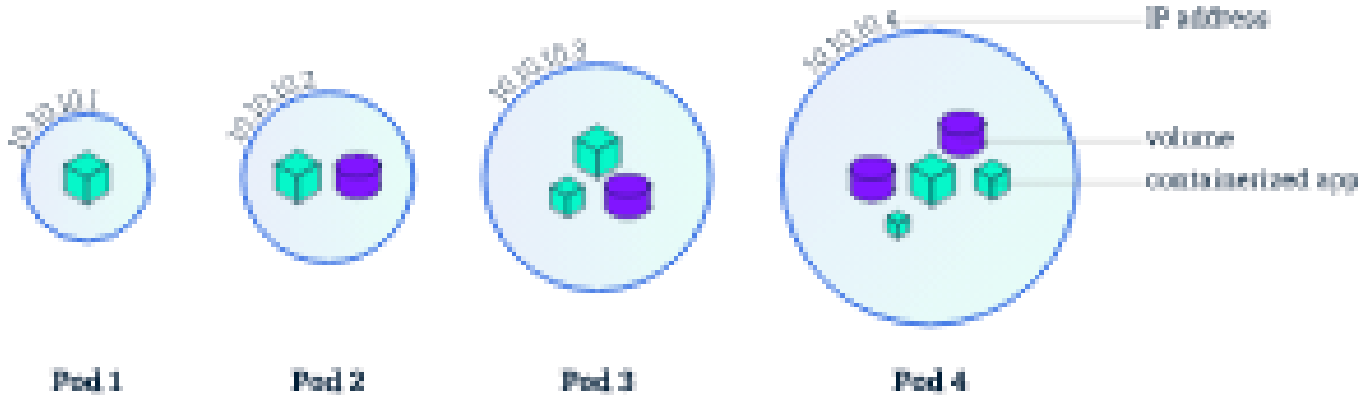
Explore Your App - Viewing Pods and Nodes

- A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker), and some shared resources for those containers. Those resources include:
  - Shared storage, as Volumes
  - Networking, as a unique cluster IP address
  - Information about how to run each container, such as the container image version or specific ports to use
- A Pod models an application-specific "logical host" and can contain different application containers which are relatively tightly coupled. For example, a Pod might include both the container with your Node.js app as well as a different container that feeds the data to be published by the Node.js webserver. The containers in a Pod share an IP Address and port space, are always co-located and co-scheduled, and run in a shared context on the same Node.

Explore Your App - Viewing Pods and Nodes

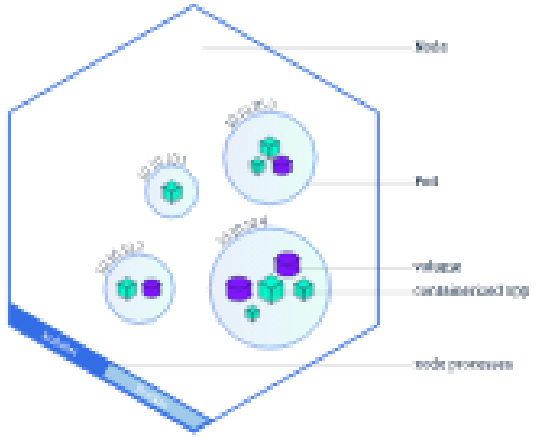
- Pods are the atomic unit on the Kubernetes platform. When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them (as opposed to creating containers directly). Each Pod is tied to the Node where it is scheduled, and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster.

Pods



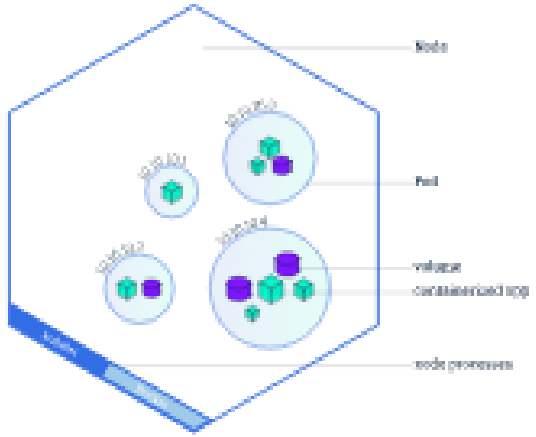
Nodes

- A Pod always runs on a **Node**. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the control plane. A Node can have multiple pods, and the Kubernetes control plane automatically handles scheduling the pods across the Nodes in the cluster. The control plane's automatic scheduling takes into account the available resources on each Node.
- Every Kubernetes Node runs at least:
  - Kubelet, a process responsible for communication between the Kubernetes control plane and the Node; it manages the Pods and the containers running on a machine.
  - A container runtime (like Docker) responsible for pulling the container image from a registry, unpacking the container, and running the application.
- The most common operations can be done with the following kubectl commands:
  - **kubectl get** - list resources
  - **kubectl describe** - show detailed information about a resource
  - **kubectl logs** - print the logs from a container in a pod
  - **kubectl exec** - execute a command on a container in a pod



Expose Your App

- A Pod always runs on a **Node**. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the control plane. A Node can have multiple pods, and the Kubernetes control plane automatically handles scheduling the pods across the Nodes in the cluster. The control plane's automatic scheduling takes into account the available resources on each Node.
- Every Kubernetes Node runs at least:
  - Kubelet, a process responsible for communication between the Kubernetes control plane and the Node; it manages the Pods and the containers running on a machine.
  - A container runtime (like Docker) responsible for pulling the container image from a registry, unpacking the container, and running the application.
- The most common operations can be done with the following kubectl commands:
  - **kubectl get** - list resources
  - **kubectl describe** - show detailed information about a resource
  - **kubectl logs** - print the logs from a container in a pod
  - **kubectl exec** - execute a command on a container in a pod



Refer Further