**Kubernetes**

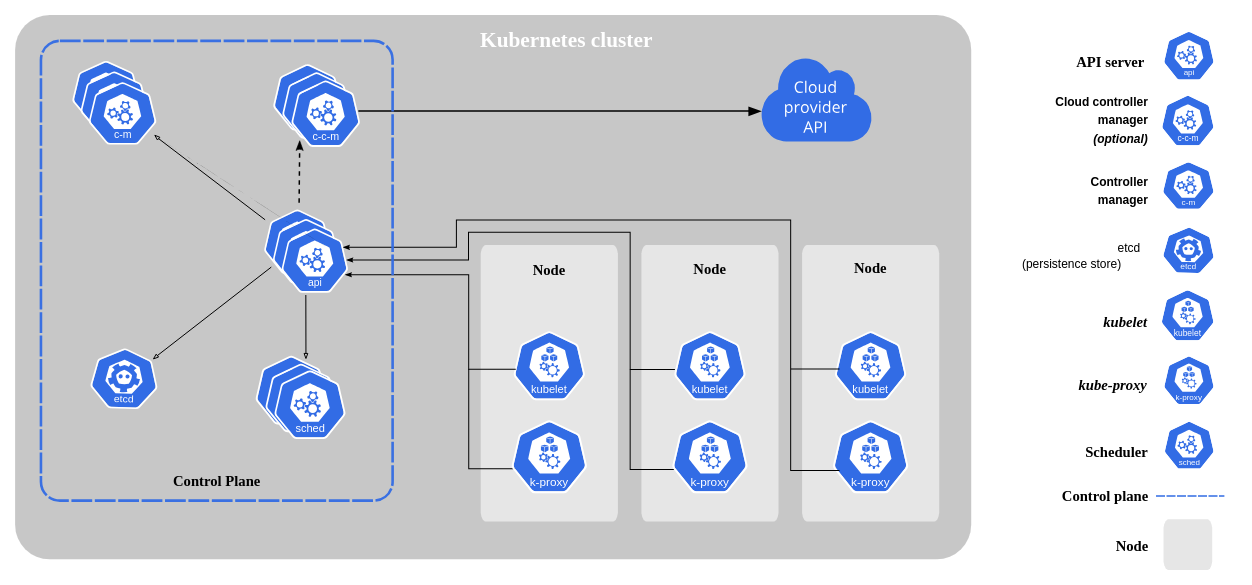
**Kind:**

[kind](https://kind.sigs.k8s.io/) lets you run Kubernetes on your local computer.

This tool requires that you have either [Docker](https://www.docker.com/) or [Podman](https://podman.io/) installed.

**Kubernetes** is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

# **Kubernetes Components**



When you deploy Kubernetes, you get a **cluster.**

A Kubernetes cluster consists of a set of **worker machines, called** [**nodes**](https://kubernetes.io/docs/concepts/architecture/nodes/), that run containerized applications. Every cluster has at least one worker node.

The worker node(s) host the [**Pods**](https://kubernetes.io/docs/concepts/workloads/pods/) that are the components of the application workload. The [**control plane**](https://kubernetes.io/docs/reference/glossary/?all=true#term-control-plane) manages the worker nodes and the Pods in the cluster.

### **kube-apiserver**

The API server is a component of the Kubernetes [control plane](https://kubernetes.io/docs/reference/glossary/?all=true#term-control-plane) that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

### **etcd**

Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.

### **kubelet**

An agent that runs on each [node](https://kubernetes.io/docs/concepts/architecture/nodes/) in the cluster. It makes sure that [containers](https://kubernetes.io/docs/concepts/containers/) are running in a [Po](https://kubernetes.io/docs/concepts/workloads/pods/)d.

### **kube-proxy**

kube-proxy is a network proxy that runs on each [node](https://kubernetes.io/docs/concepts/architecture/nodes/) in your cluster, implementing part of the Kubernetes [Service](https://kubernetes.io/docs/concepts/services-networking/service/) concept.

It maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

### 

### 

### **Container runtime**

A fundamental component that empowers Kubernetes to run containers effectively. It is responsible for managing the execution and lifecycle of containers within the Kubernetes environment.

## **Addons**

Addons use Kubernetes resources ([DaemonSet](https://kubernetes.io/docs/concepts/workloads/controllers/daemonset), [Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/), etc) to implement cluster features.

### **DNS**

While the other addons are not strictly required, all Kubernetes clusters should have [cluster DNS](https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/), as many examples rely on it.

Cluster DNS is a DNS server, in addition to the other DNS server(s) in your environment, which serves DNS records for Kubernetes services.

### **Web UI (Dashboard)**

[Dashboard](https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/) is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage and troubleshoot applications running in the cluster, as well as the cluster itself.

### **Container Resource Monitoring**

[Container Resource Monitoring](https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-usage-monitoring/) records generic time-series metrics about containers in a central database, and provides a UI for browsing that data.

### **Cluster-level Logging**

A [cluster-level logging](https://kubernetes.io/docs/concepts/cluster-administration/logging/) mechanism is responsible for saving container logs to a central log store with search/browsing interface.

### **Network Plugins**

[Network plugins](https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/network-plugins) are software components that implement the container network interface (CNI) specification. They are responsible for allocating IP addresses to pods and enabling them to communicate with each other within the cluster.

-------------------------------------------------------------------------------------------------------------------------------------

**Pod:**

Smallest Unit of k8s

Abstraction over container

Usually 1 application per pod

Each Pod gets its own IP address

If re-creates the Pod, a new IP-address is re-created.

**Service:**

Static or Permanent IP address

For these, Lifecycle of Pods and Service are not connected. If Pod dies, its service and Ip address will stay. So, no change is needed.

A Load-balancer.

**ConfigMap:** External configuration of your application.

**Secret:** used to store secret data (password, Certificate)

Base64 encoded.

**Volumes:** Storage, to save data.

Can be on **local** machine

Or **Remote** outside the k8s cluster

**K8s doesn’t manage data persistence.**

**Replica is connected to the same service.**

**Deployment:**

Blue-print for my-app Pods

Abstractions of Pods

Database can’t be replicated via Deployment.

Deployment is for stateless App (Does not save Client data)

**StatefulSet:**

For stateful Apps or Databases

**Kubernetes Architecture:**

**Work Node:**

Each Node has multiple Pods on it

3 processes must be installed on every Node.

* Kubelet
* Kube Proxy
* Container Runtime

Worker Nodes do the actual work

Kubelet interacts with both the container and node

Kubelet starts the Pod with a container inside.

Kube Proxy forwards the request.

**Master Node: Master Process**

4 Processes run on every master Node.

* **Api Server:**

Cluster Gateway.

Acts as a gatekeeper for authentication.

This is the only one entry-point to the cluster.

* **Scheduler:**

Just decides on which Node, new Pod should be scheduled.

* **Controller Manager:**

Detects cluster state changes.

When 1 Pod died, controller manager detects that and tries to recover the cluster state as soon as possible. Make a request to the scheduler. Based on the resource calculation, Scheduler decides which work nodes should re-start the Pods again & make request on the Kubelets on those work nodes.

* **Etcd:**

Is the cluster brain.

Cluster changes get stored in the key value store.

All information is stored in etcd. ( But, application data is not stored in etcd)

**How to add new Master/ Node Server:**

* Get new bare server
* Install all the master/worker node processes
* Add it to the cluster.

**Book: Kubernetes Up & Running**

**What is the swap space?**

Swap space is a space on a hard disk that is a substitute for physical memory. It is used as virtual memory which contains process memory images. Whenever our computer runs short of physical memory it uses its virtual memory and stores information in memory on disk.

**Minikube:**

Quickly sets up a local Kubernetes cluster on macOS, Linux, windows.

It is designed to run on a single machine with minimal resource requirements, making it ideal for local development or running Kubernetes on a small scale.

**kind** stands for **Kubernetes IN Docker**

**Imperative configuration** involves creating Kubernetes resources directly at the command line against a Kubernetes cluster.

**Declarative configuration** defines resources within manifest files and then applies those definitions to the cluster.

**The Network File System (NFS)** is a mechanism for storing files on a network. It is a distributed file system that allows users to access files and directories located on remote computers and treat those files and directories as if they were local.

# **Liveness Probe**

Suppose that a Pod is running our application inside a container, but due to some reason let’s say memory leak, cpu usage, application deadlock etc the application is not responding to our requests, and stuck in error state.

Liveness probe checks the container health as we tell it do, and if for some reason the liveness probe fails, it restarts the container.

**Readiness Probe**

In some cases we would like our application to be alive, but not serve traffic unless some conditions are met e.g, populating a dataset, waiting for some other service to be alive etc. In such cases we use readiness probe. If the condition inside readiness probe passes, only then our application can serve traffic.

**Both liveness & readiness probes are used to control the health of an application.**

**Failing liveness probe will restart the container, whereas failing readiness probe will stop our application from serving traffic.**

A Volume separates storage from a container but binds it to a Pod,

while PVs separate storage from a Pod.

The lifecycle of a Volume is dependent on the Pod using it,

while the lifecycle of a PV is not.

The **PersistentVolume** subsystem provides an API for users and administrators that abstracts details of how storage is provided from how it is consumed.

**horizontal** scaling involves adding more machines or nodes to a system,

while **vertical** scaling involves adding more power (CPU, RAM, storage, etc.) to an existing machine.

***Horizontal Pod Autoscaler (HPA)*** *we can scale horizontally by adding additional replicas of a given pod.*

The HPA works with an API resource and a controller, which you link to a Deployment or StatefulSet using selector labels. Next, the HPA controller periodically compares (every 15 seconds by default) the HPA configuration with the pod's metrics. The most obvious metrics are CPU and memory usage.

VPA, also known as the **Vertical Pod Autoscaler**. Rather than increasing the number of replicas, it works based on increasing the pod's configured resources to the recommended value.

a **decoupled** architecture allows for greater flexibility and maintainability,

while a **coupled** architecture is more tightly integrated but can be more difficult to modify.

**Creating Multnode in Minikube Cluster:**

**minikube node add**