Kubernetes Architecture:

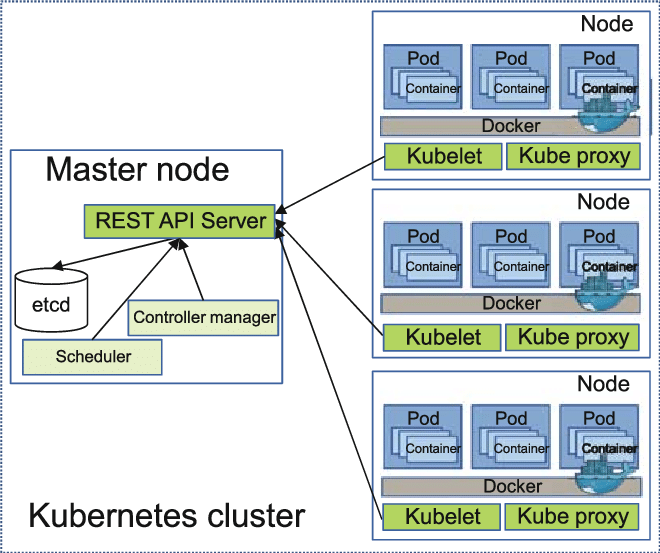
When you deploy Kubernetes, you get a cluster.

A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.

The worker node(s) host the Pods that are the components of the application workload.

The control plane manages the worker nodes and the Pods in the cluster.

In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing high availability.



Master Node:

API server:

The API server is a component of the Kubernetes control plane that exposes the Kubernetes API. The API server is the front end for the Kubernetes control plane.

The main implementation of a Kubernetes API server is designed to scale horizontally that is scales by deploying more instances.

You can run several instances of and balance traffic between those instances.

Etcd:

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

If your Kubernetes cluster uses etcd as its backing store, make sure you have a backup plan for those data.

### scheduler :

Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.

controller-manager

Control plane component that runs controller processes.

Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

Some types of these controllers are:

* Node controller: Responsible for noticing and responding when nodes go down.
* Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
* Endpoints controller: Populates the Endpoints object (that is, joins Services & Pods).

As with the kube-controller-manager, the cloud-controller-manager combines several logically independent control loops into a single binary that you run as a single process.

You can scale horizontally (run more than one copy) to improve performance.

The following controllers can have cloud provider dependencies:

* Node controller: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding.
* Route controller: For setting up routes in the underlying cloud infrastructure.
* Service controller: For creating, updating and deleting cloud provider load balancers.

## Worker Node Components:

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

### Kubelet:

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

### kube-proxy

kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.

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

kube-proxy uses the operating system packet filtering layer if there is one and it's available. Otherwise, kube-proxy forwards the traffic itself.

### Container runtime

The container runtime is the software that is responsible for running containers.

Kubernetes supports several container runtimes: Docker, containerd, and any implementation of the Kubernetes CRI (Container Runtime Interface).

## Addons

Addons use Kubernetes resources (DaemonSet, Deployment, etc) to implement cluster features. Because these are providing cluster-level features, namespaced resources for addons belong within the kube-system namespace.

### DNS

While the other addons are not strictly required, all Kubernetes clusters should have cluster DNS, 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.

Containers started by Kubernetes automatically include this DNS server in their DNS searches.

### Web UI (Dashboard)

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 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 mechanism is responsible for saving container logs to a central log store with search/browsing interface.

Admin flow:

Admin --->cubectl client-->master(api server)---> worker(kubelet------>pods).

Config file will have a secrets,with out configfile you cont connec cluster.

Kubeconfig (its secrets)---k8s server ip (apiserver,port,protocol)

For every cluster have a separate config file.

Kubectl is communicated to the eks cluster server.

End-user flow:

I wanted to run a application in the k8s and I wanted to access from outside as end user.

Enduser---->route53---->ingress controller---->ingress rules---->service---->pod

In kubernetes every thing called as a object.

1.Deployment objects

Pod

Deployment

Repicaset

Daemons

State full set

2.Storage objects

Persistence volume

Persistence volume claim

Storage class

3.Network objects

Service object

Ingress

4.Secrets objects

Configmap

Secrets