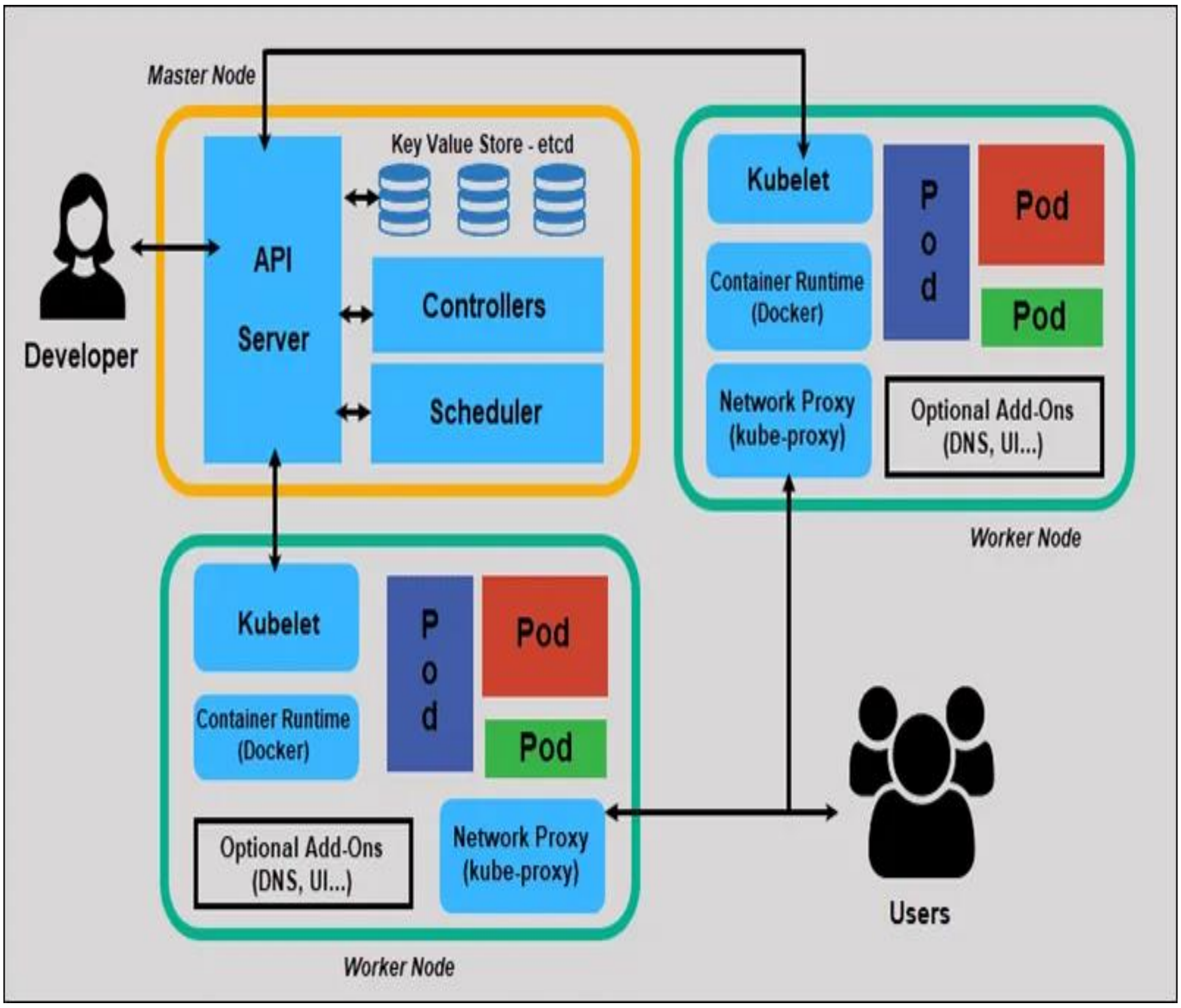
Kubernetes Architecture And Commands

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.



Master Node
Key Value Store - etcd
API Server
Controllers
Scheduler
Developer
Kubelet
Container Runtime (Docker)
Network Proxy (kube-proxy)
Pod
Pod
Pod
Optional Add-Ons (DNS, UI...)
Worker Node
Kubelet
Container Runtime (Docker)
Pod
Pod
Pod
Optional Add-Ons (DNS, UI...)
Network Proxy (kube-proxy)
Worker Node
Users

Master Node :-

It is responsible for managing whole cluster. It monitors the health check of all nodes in the cluster. It stores the members information regarding the different nodes, planning which containers are schedules to which worker nodes, monitoring the containers and nodes, etc

Kubernetes Architecture And Commands

Four basic components of the master node(control plane):

API Server :-
The API server is a centralized component where all the cluster components are communicated.

Scheduler :-
Scheduler is responsible for assigning your application pod to worker node.

Controller Manager :-
The Controller Manager maintains the cluster, it handles node failures, replicating components, maintaining the correct number of pods,

ETCD :-
Etcd is a data store that stores the cluster configuration. It is recommended that you have a back-up as it is the source of truth for your cluster.

The Worker Node :-
It is nothing but a virtual machine(VM's) running in cloud or on-prem, a physical server running inside your data center. So, any hardware capable of running container runtime can become a worker node.

Kubernetes Architecture And Commands

Three basic components of the Worker Node(Data plane)

Kubelet :-
The Kubelet runs and manages the containers on node and it talks to API server.

Kube-Proxy :-
The Kube-proxy load balances traffic between application components. It is also called as service proxy which run on each node in the Kubernetes cluster.

Container Runtime :-
Container runtime which runs the containers like Docker, rkt or containerd. Once you have the specification that describe your image for your application, the container runtime will pull the images and and run the containers.

Kubernetes Architecture And Commands

- ## Cluster Management

# kubectl cluster-info

Display endpoint information regarding the services and master in the cluster

# kubectl version

Show the Kubernetes version functioning on the client and server

# kubectl config view

Get the configuration of the cluster

# kubectl api-resources

Make a list of the available API resources

# kubectl api-versions

Make a list of the available API versions

# kubectl get all –all-namespaces

List everything

Kubernetes Architecture And Commands

- ## Node operations

```
# kubectl get node
```

List one or more nodes

```
# kubectl delete node <node_name>
```

Delete a node or multiple nodes

```
# kubectl top node
```

Display Resource usage (CPU/Memory/Storage) for nodes

```
# kubectl describe nodes | grep Allocated -A 5
```

Resource allocation per node

```
# kubectl get pods -o wide | grep <node_name>
```

Pods running on a node

```
# kubectl annotate node <node_name>
```

Annotate a node

```
# kubectl cordon node <node_name>
```

Mark a node as unschedulable

```
# kubectl uncordon node <node_name>
```

Mark node as schedulable

# kubectl drain node <node_name>

Drain a node in preparation for maintenance

# kubectl label node

Add the labels of one or more nodes

- **Namespaces [**Shortcode = ns]

# kubectl create namespace <namespace_name>

Create namespace <name>

# kubectl describe namespace <namespace_name>

Show the detailed condition of one or more namespace

# kubectl delete namespace <namespace_name>

Delete a namespace

# kubectl edit namespace <namespace_name>

Edit and modify the namespace's definition

# kubectl top namespace <namespace_name>

Display Resource (CPU/Memory/Storage) usage for a namespace

Kubernetes Architecture And Commands

- ## Deployments

# kubectl get deployment

List one or more deployments

# kubectl describe deployment <deployment_name>

Show the in-depth state of one or more deployments

# kubectl edit deployment <deployment_name>

Edit and revise the definition of one or more deployment on the server

# kubectl create deployment <deployment_name>

Generate one a new deployment

# kubectl delete deployment <deployment_name>

Delete deployments

# kubectl rollout status deployment <deployment_name>

Check the rollout status of a deployment

- ## Replication Controllers [Shortcode = rc]

# kubectl get rc

Make a list of the replication controllers

# kubectl get rc –namespace=<namespace_name>

Make a list of the replication controllers by namespace

Kubernetes Architecture And Commands

- **ReplicaSets [**Shortcode = rs]

# kubectl get replicasets

List ReplicaSets

# kubectl describe replicasets <replicaset_name>

Show the detailed state of one or more ReplicaSets

# kubectl scale –replicas=[x]

Scale a ReplicaSet [x is a number here]

- **Listing Resources**

# kubectl get namespaces

Create a plain-text list of all namespaces

# kubectl get pods

Create a plain-text list of all pods

# kubectl get pods -o wide

Create a comprehensive plain-text list of all pods

# kubectl get pods–field-selector=spec. nodeName=[server-name]

Create a list of all pods functioning on a certain node server

# kubectl get replicationcontroller [replication-controller-name]

In plain text, make a lst a specific replication controller

```
# kubectl get replicationcontroller, services
```

Generate a plain-text list of all replication services and controllers

- ## Logs

```
# kubectl logs <pod_name>
```

Print the logs for a pod

```
# kubectl logs –since=1h <pod_name>
```

Print the logs for a pod for the last hour

```
# kubectl logs –tail=20 <pod_name>
```

Get the current 20 lines of logs

```
# kubectl logs -f <service_name> [-c <$container>]
```

Get logs from a service and choose which container optionally

```
# kubectl logs -f <pod_name>
```

Adhere to new logs and print the logs for a pod

```
# kubectl logs -c <container_name> <pod_name>
```

For a container in a pod, Print the logs

```
# kubectl logs <pod_name> pod.log
```

Output the logs for a pod into a 'pod.log' file

```
# kubectl logs –previous <pod_name>
```
View the logs for the last failed pod