Page

Introduction to **Kubernetes**

+88013 12131212 | paged.us@gmail.com | www.paged.cloud

Container orchestration

- Container orchestration automates the deployment, management, scaling, and networking of containers across the cluster. It is focused on managing the life cycle of containers.
- Enterprises that need to deploy and manage hundreds or thousands of Linux® containers and hosts can benefit from container orchestration.
- Container orchestration is used to automate the following tasks at scale:
  - ✓ Configuring and scheduling of containers
  - ✓ Provisioning and deployment of containers
  - ✓ Redundancy and availability of containers
  - ✓ Scaling up or removing containers to spread application load evenly across host infrastructure
  - ✓ Movement of containers from one host to another if there is a shortage of resources in a host, or if a host dies
  - ✓ Allocation of resources between containers
  - ✓ External exposure of services running in a container with the outside world
  - ✓ Load balancing of service discovery between containers
  - ✓ Health monitoring of containers and hosts

Swarm vs Kubernetes

Both Kubernetes and Docker Swarm are important tools that are used to deploy containers inside a cluster but there are subtle differences between the both

Features	Kubernetes	Docker Swarm
Installation & Cluster Configuration	Installation is complicated; but once setup, the cluster is very strong	Installation is very simple; but cluster is not very strong
GUI	GUI is the Kubernetes Dashboard	There is no GUI
Scalability	Highly scalable & scales fast	Highly scalable & scales 5x faster than Kubernetes
Auto-Scaling	Kubernetes can do auto-scaling	Docker Swarm cannot do auto-scaling
Rolling Updates & Rollbacks	Can deploy Rolling updates & does automatic Rollbacks	Can deploy Rolling updates, but not automatic Rollbacks
Data Volumes	Can share storage volumes only with other containers in same Pod	Can share storage volumes with any other container
Logging & Monitoring	In-built tools for logging & monitoring	3rd party tools like ELK should be used for logging & monitoring

Kubernetes

- Kubernetes also known as K8s, is an open-source Container Management tool
- It provides a container runtime, container orchestration, container-centric infrastructure orchestration, self-healing mechanisms, service discovery, load balancing and container (de)scaling.
- Initially developed by Google, for managing containerized applications in a clustered environment but later donated to CNCF
- Written in Golang
- 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.

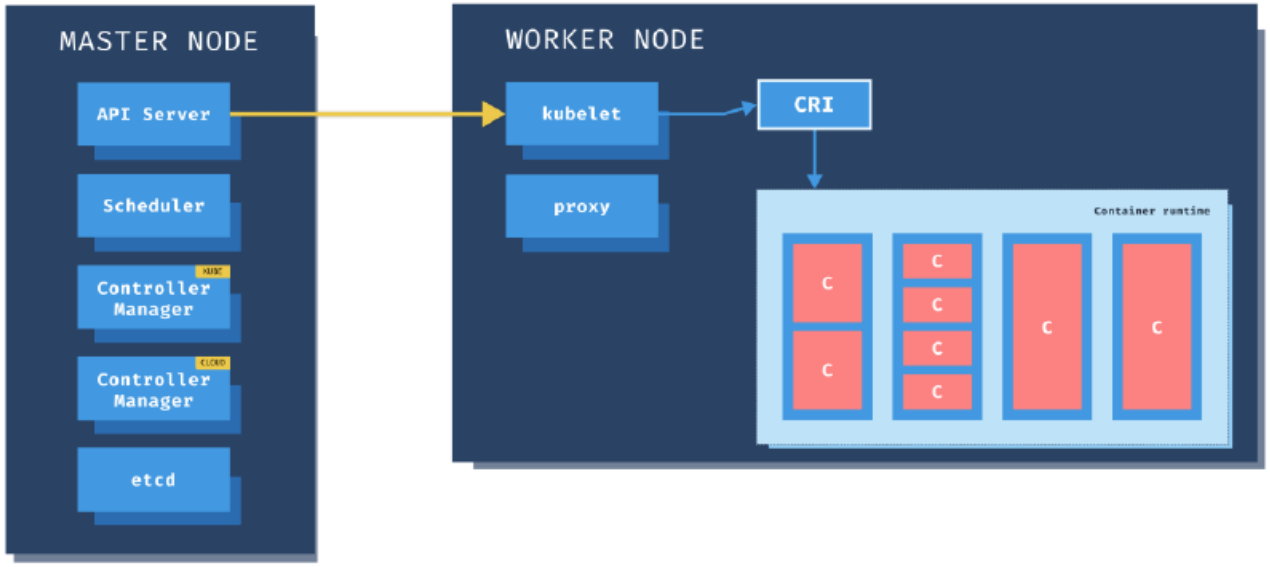
Kubernetes Cluster

A Kubernetes cluster is a set of physical or virtual machines and other infrastructure resources that are needed to run your containerized applications. Each machine in a Kubernetes cluster is called a **node**.

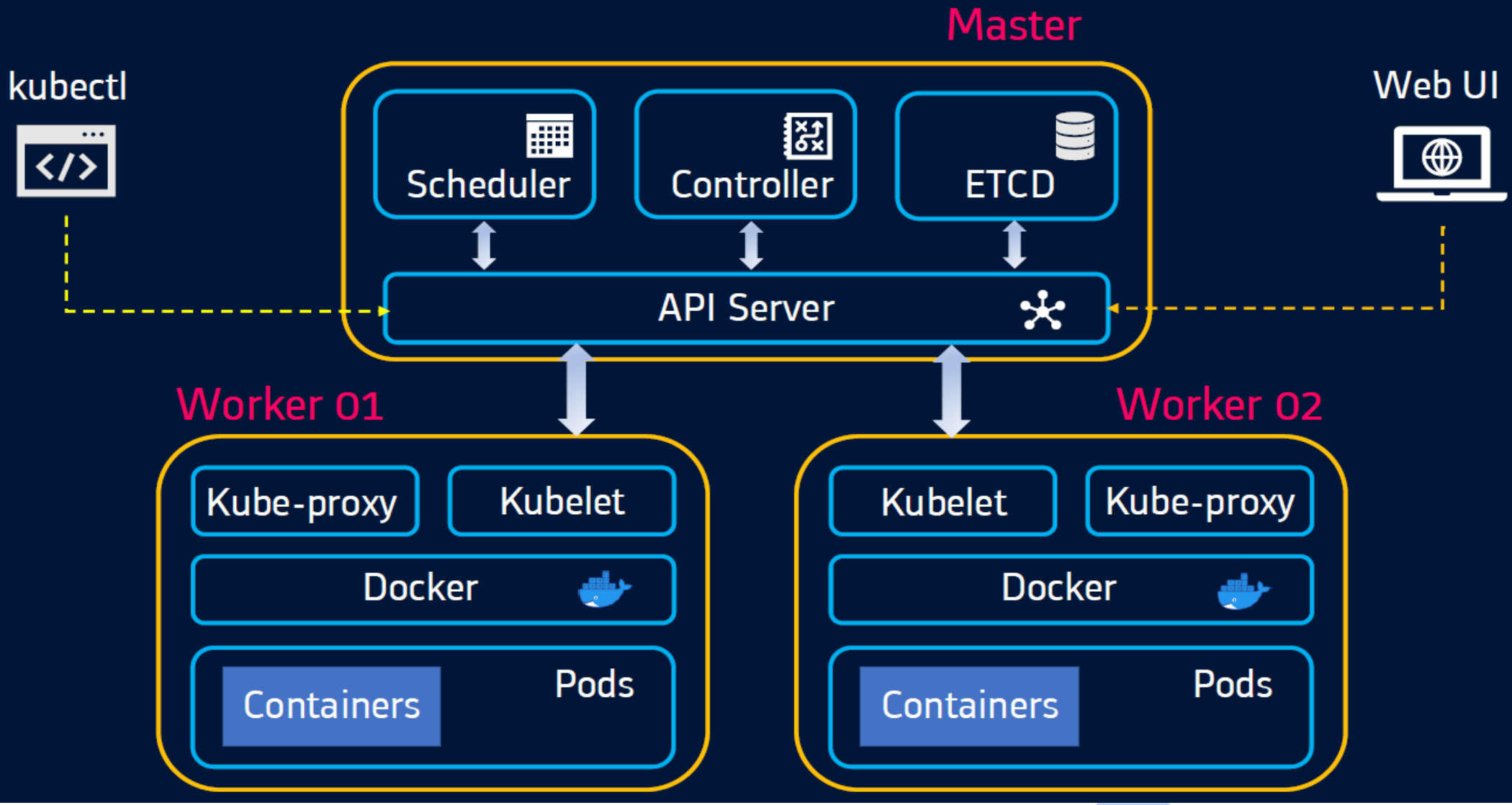
There are two types of node in each Kubernetes cluster:

Master node(s): hosts the Kubernetes control plane components and manages the cluster

Worker node(s): runs your containerized applications



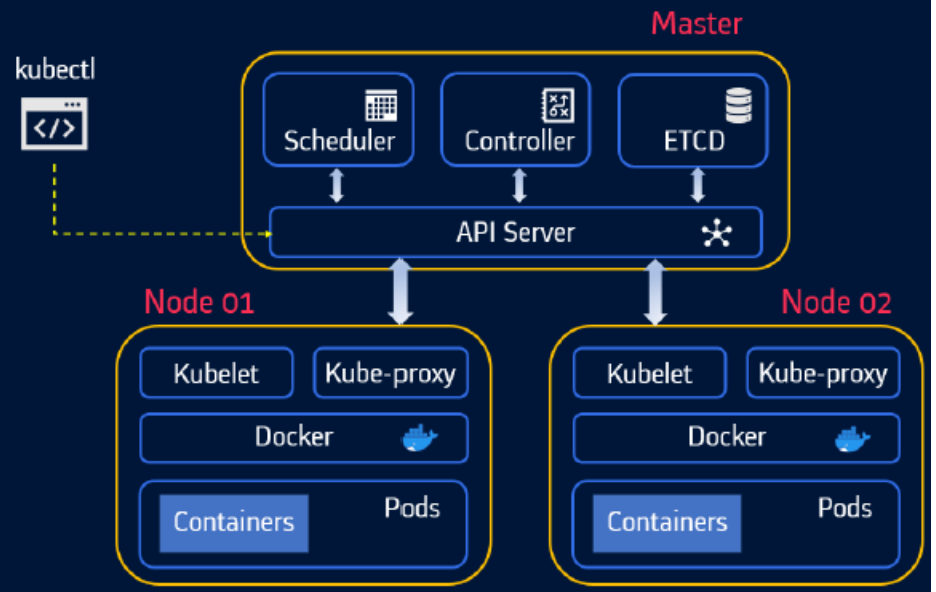
Kubernetes Architecture



Kubernetes Architecture

Kubernetes Master

- Master is responsible for managing the complete cluster.
- You can access master node via the CLI, GUI, or API
- The master watches over the nodes in the cluster and is responsible for the actual orchestration of containers on the worker nodes
- For achieving fault tolerance, there can be more than one master node in the cluster.
- It is the access point from which administrators and other users interact with the cluster to manage the scheduling and deployment of containers.
- It has four components: ETCD, Scheduler, Controller and API Server

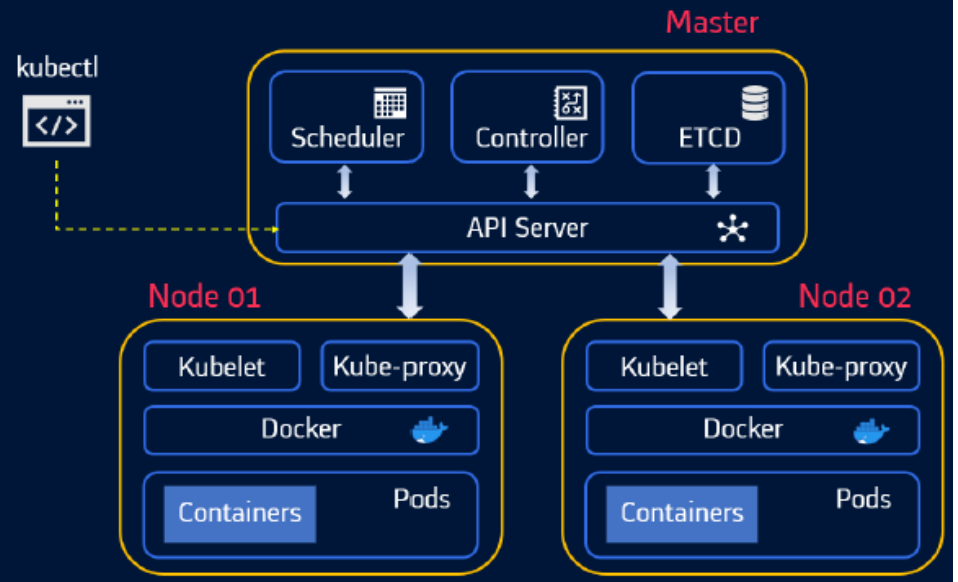


Kubernetes Architecture

Kubernetes Master

ETCD

- ETCD is a distributed reliable key-value store used by Kubernetes to store all data used to manage the cluster.
- When you have multiple nodes and multiple masters in your cluster, etcd stores all that information on all the nodes in the cluster in a distributed manner.
- ETCD is responsible for implementing locks within the cluster to ensure there are no conflicts between the Masters



Scheduler

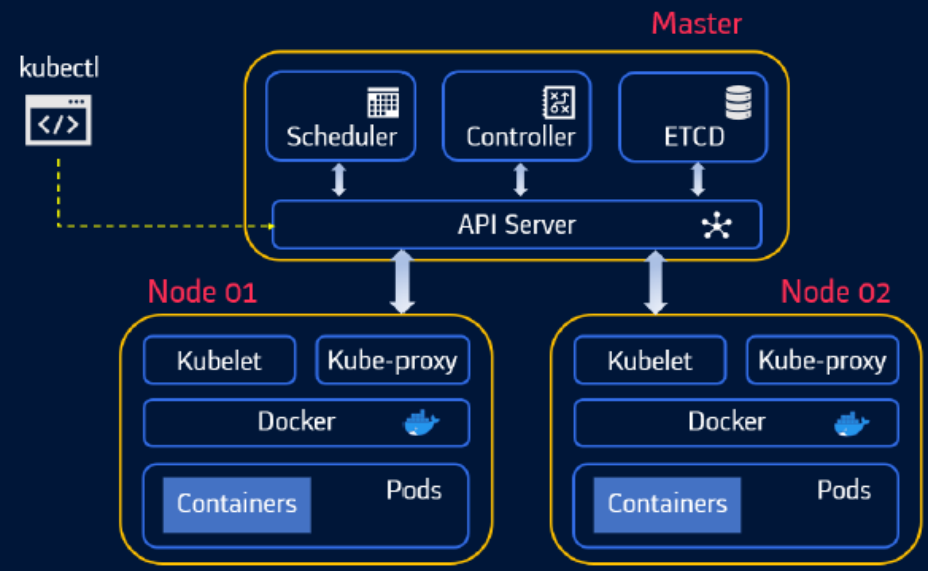
- The scheduler is responsible for distributing work or containers across multiple nodes.
- It looks for newly created containers and assigns them to Nodes.

Kubernetes Architecture

Kubernetes Master

API server manager

- Masters communicate with the rest of the cluster through the kube-apiserver, the main access point to the control plane.
- It validates and executes user's REST commands
- kube-apiserver also makes sure that configurations in etcd match with configurations of containers deployed in the cluster.



Controller manager

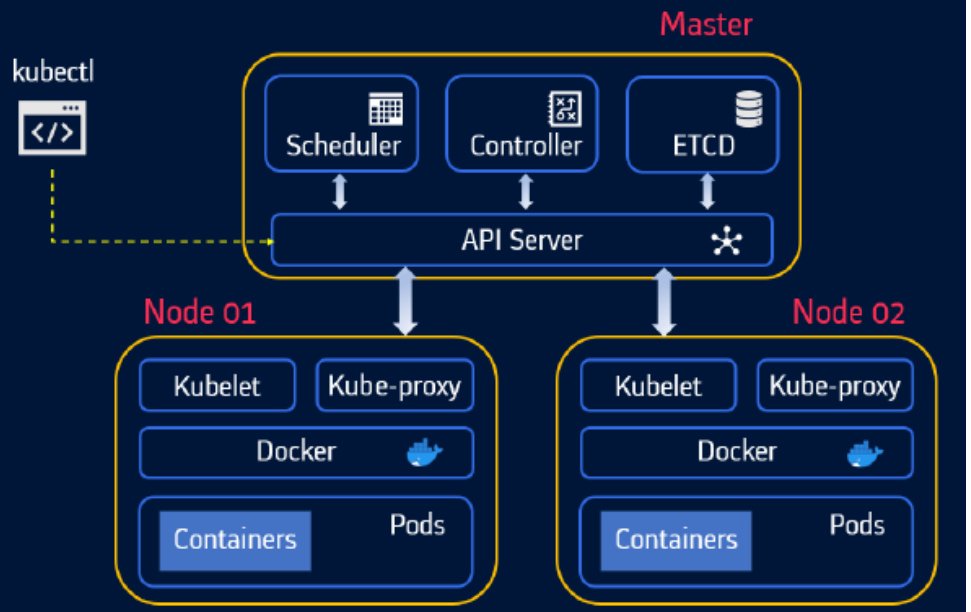
- The controllers are the brain behind orchestration.
- They are responsible for noticing and responding when nodes, containers or endpoints goes down. The controllers makes decisions to bring up new containers in such cases.
- The kube-controller-manager runs control loops that manage the state of the cluster by checking if the required deployments, replicas, and nodes are running in the cluster

Kubernetes Architecture

Kubernetes Master

Kubectl

- kubectl is the command line utility using which we can interact with k8s cluster
- Uses APIs provided by API server to interact.
- Also known as the kube command line tool or kubectl or kube control.
- Used to deploy and manage applications on a Kubernetes



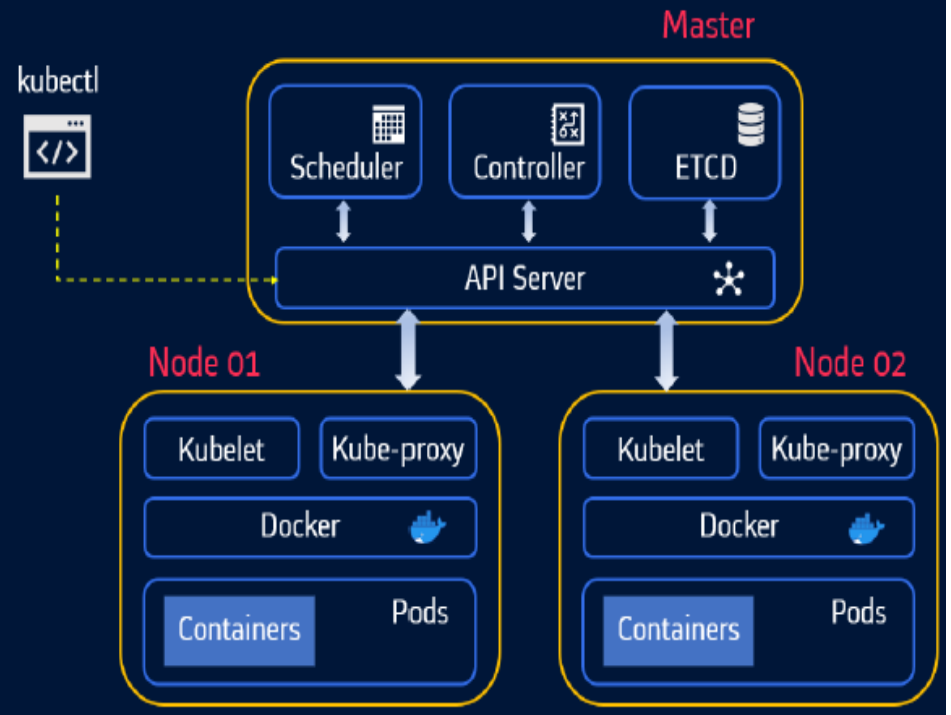
- kubectl run nginx used to deploy an application on the cluster.
- kubectl cluster-info used to view information about the cluster and the
- kubectl get nodes used to list all the nodes part of the cluster.

Kubernetes Architecture

Kubernetes Worker

Kubelet

- Worker nodes have the kubelet agent that is responsible for interacting with the master to provide health information of the worker node
- To carry out actions requested by the master on the worker nodes.



Kube proxy

- The kube-proxy is responsible for ensuring network traffic is routed properly to internal and external services as required and is based on the rules defined by network policies in kube-controller-manager and other custom controllers.

Page

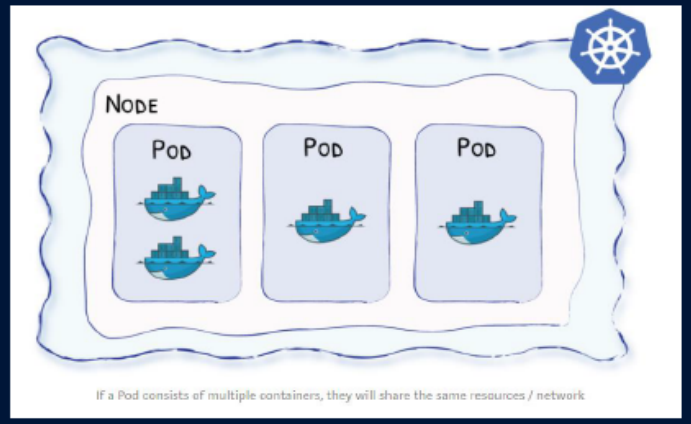
Kubernetes Pods

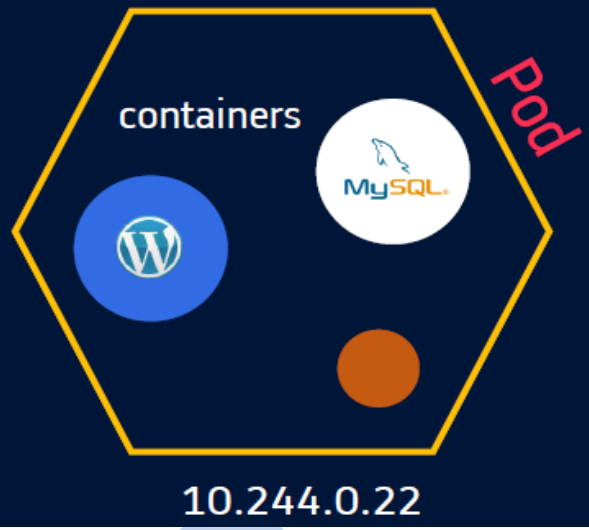
Kubernetes

Pods

- Basic scheduling unit in Kubernetes. Pods are often ephemeral
- Kubernetes doesn't run containers directly; instead it wraps one or more containers into a higher-level structure called a pod
- It is also the smallest deployable unit that can be created, schedule, and managed on a Kubernetes cluster. Each pod is assigned a unique IP address within the cluster.
- Pods can hold multiple containers as well, but you should limit yourself when possible. Because pods are scaled up and down as a unit, all containers in a pod must scale together, regardless of their individual needs. This leads to wasted resources.

Ex: nginx, mysql, wordpress..

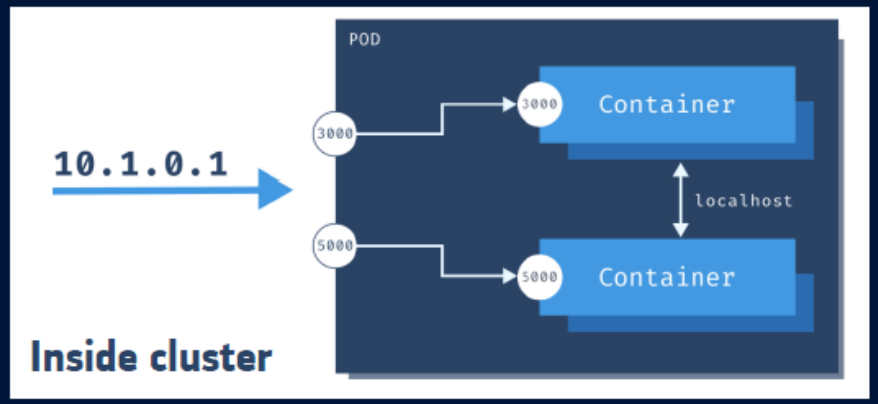




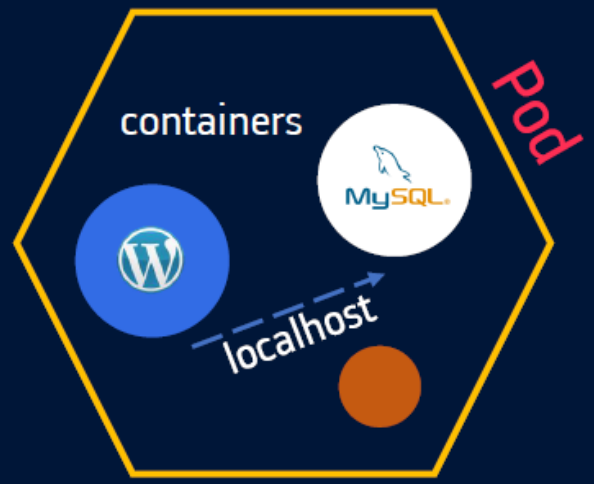
Kubernetes

Pods

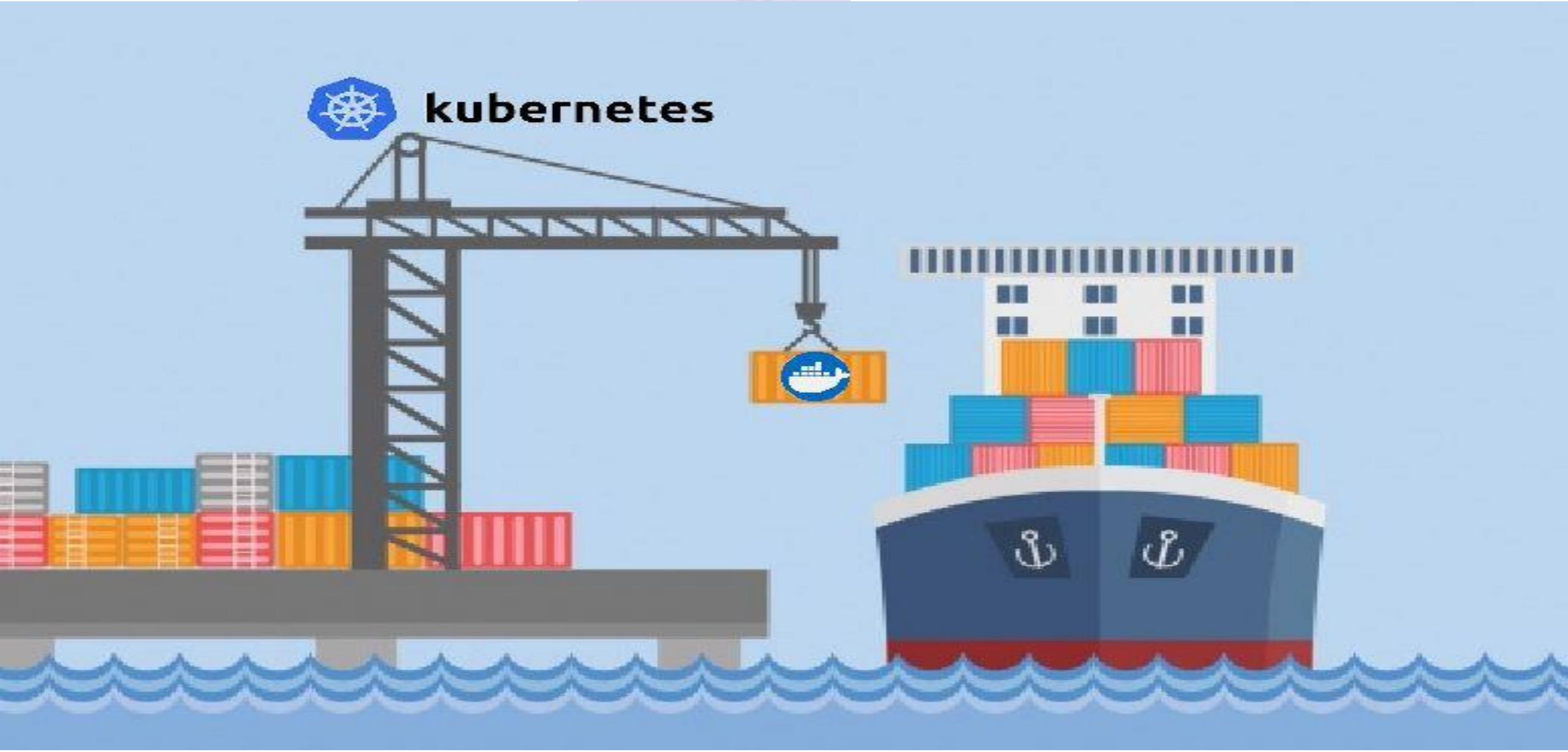
- Any containers in the same pod will share the same storage volumes and network resources and communicate using localhost
- K8s uses YAML to describe the desired state of the containers in a pod. This is also called a Pod Spec. These objects are passed to the kubelet through the API server.
- Pods are used as the unit of replication in Kubernetes. If your application becomes too popular and a single pod instance can't carry the load, Kubernetes can be configured to deploy new replicas of your pod to the cluster as necessary.



Using the example from the above figure, you could run curl 10.1.0.1:3000 to communicate to the one container and curl 10.1.0.1:5000 to communicate to the other container from other pods. However, if you wanted to talk between containers - for example, calling the top container from the bottom one, you could use http://localhost:3000.



Namespace

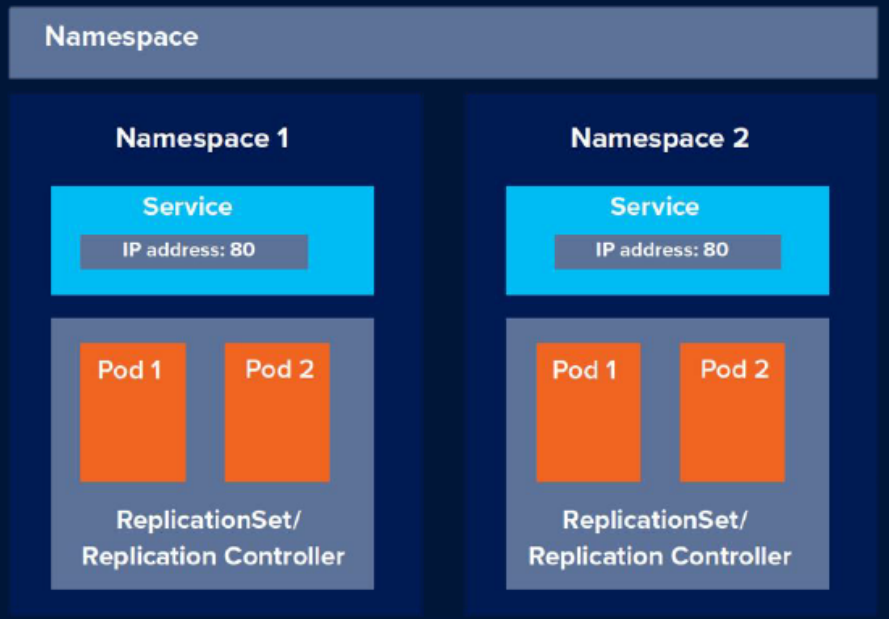


Kubernetes

Namespaces

Namespaces are Kubernetes objects which partition a single Kubernetes cluster into multiple virtual clusters

- Kubernetes clusters can manage large numbers of unrelated workloads concurrently and organizations often choose to deploy projects created by separate teams to shared clusters.
- With multiple deployments in a single cluster, there are high chances of deleting deployments belong to deff prohjects.
- So namespaces allow you to group objects together so you can filter and control them as a unit/group.
- Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces.
- So each Kubernetes namespace provides the scope for Kubernetes Names it contains; which means that using the combination of an object name and a Namespace, each object gets a unique identity across the cluster

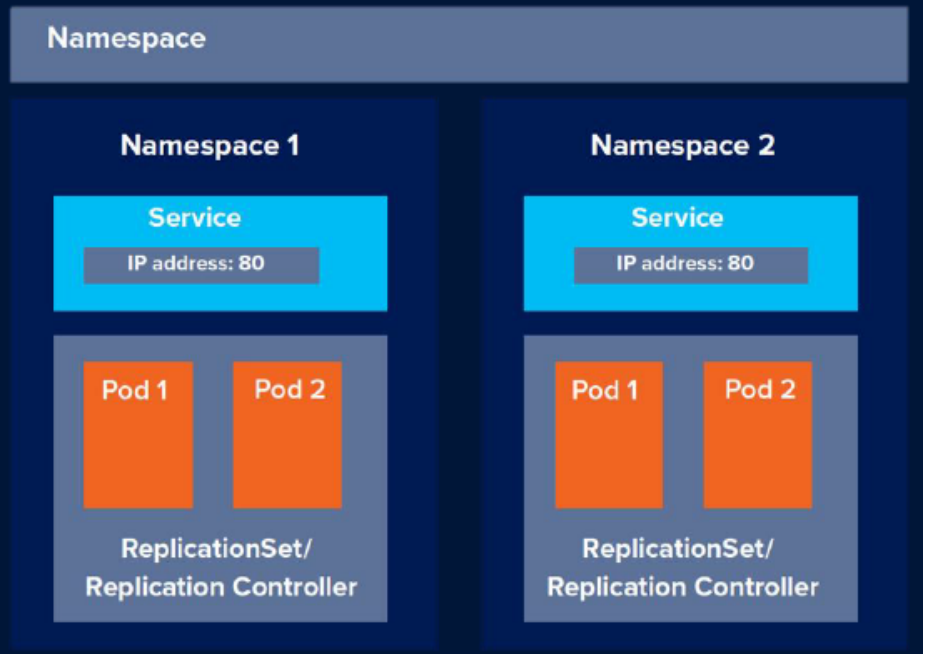


Kubernetes

Namespaces

By default, a Kubernetes cluster is created with the following three namespaces:

- default: It's a default namespace for users. By default, all the resource created in Kubernetes cluster are created in the default namespace
- Kube-system: It is the Namespace for objects created by Kubernetes systems/control plane. Any changes to objects in this namespace would cause irreparable damage to the cluster itself
- kube-public: Namespace for resources that are publicly readable by all users. This namespace is generally reserved for cluster usage like Configmaps and Secrets



Kubernetes

Namespaces

kubectl get namespaces

```
root@k8s-master:/home/osboxes# kubectl get ns
NAME                   STATUS   AGE
default                Active   68m
kube-node-lease        Active   68m
kube-public            Active   68m
kube-system            Active   68m
```

Kubernetes Command Sheet

https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands