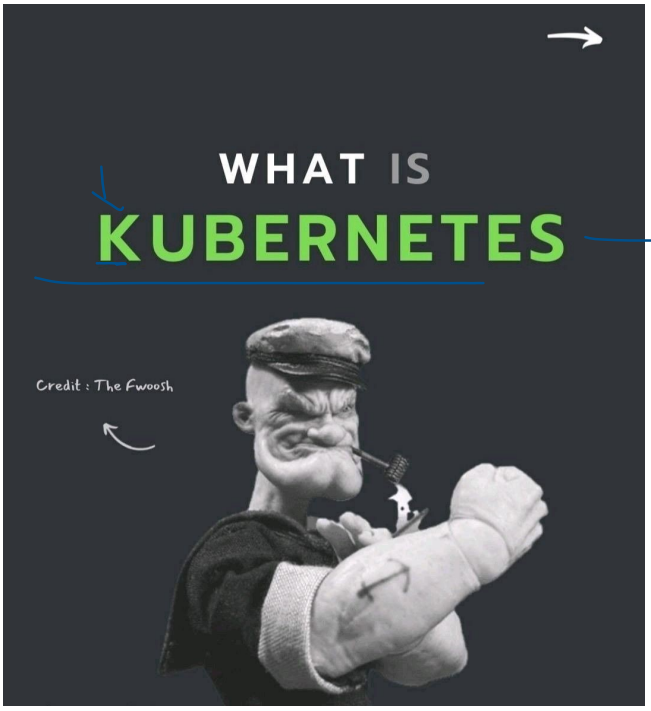
Kebernetes

11:32 AM



APPLICATION

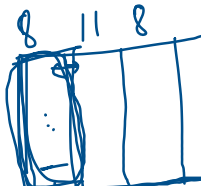
Lets say you have created an application

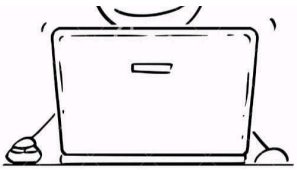
1) repo — Source code
work or not
2 ahoor
2) share docke image
docke build
image → hub

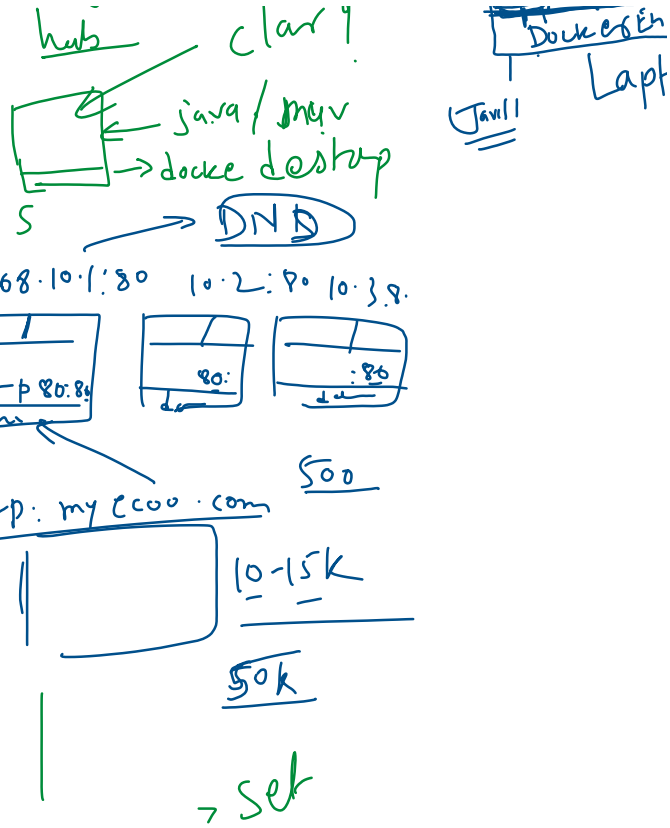
DOCKER

And used Docker containers to package the application

hub
install JDK
mave
copy —
java -jar





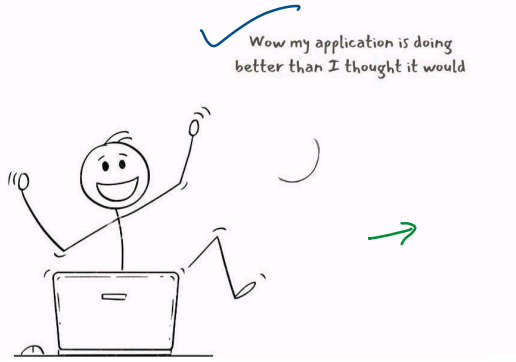


DEPLOYED

Say you have deployed on 3 different servers using Docker

and ...

Your application starts getting massive traffic



SCALING

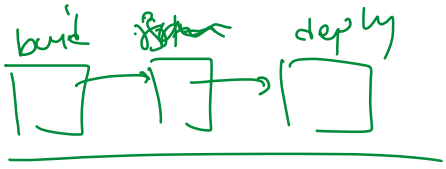
Now you need to scale up fast; how will you go from 3 servers to 40 servers that you may require?

How to decide which container should go where? Monitor all containers ? & make sure they restart if they die?

KUBERNETES

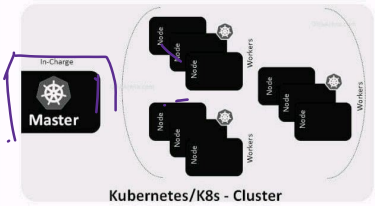
This is where Kubernetes comes into play

Kubernetes (aka k8s or "kube") is an open source container orchestration platform that automates deploying, managing, and scaling containerized applications.



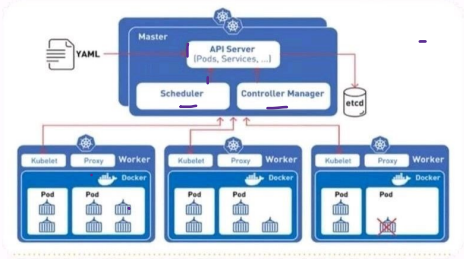
HOW IT WORKS ?

A Kubernetes cluster consists of a set of worker machines, called **nodes**, that run containerized applications



Every cluster has at least one **worker node**. Hence if a node fails, your application will still be accessible from the other nodes as in a cluster, **multiple** nodes are grouped.

ARCHITECTURE



Every node contains a container runtime, **Kubelet** (for starting, stopping, and managing individual containers by requests from the Kubernetes control plane), and **kube-proxy** (for networking and load balancing).

How is Kubernetes related to Docker?

- Docker is a containerization platform and Kubernetes is a container management (orchestrator) tool for container platforms like Docker.
- Kubernetes uses a Container Runtime Interface (CRI) plugin interface which enables kubelet to use a wide variety of container runtimes, without the need to recompile.
- Kubernetes could use any container runtime that implements CRI to manage pods, containers and container images

- From docker images we are creating containers and Kubernetes is used to manages these containers

- Every clustor 1 or more master node (vm)
- Every clustor will have 1 or more worker/slave node
- On **Worker node** our container going to run -> applications.
- On Worker node - service running on -> 1.Docker 2.kube-proxy 3. Kubelet

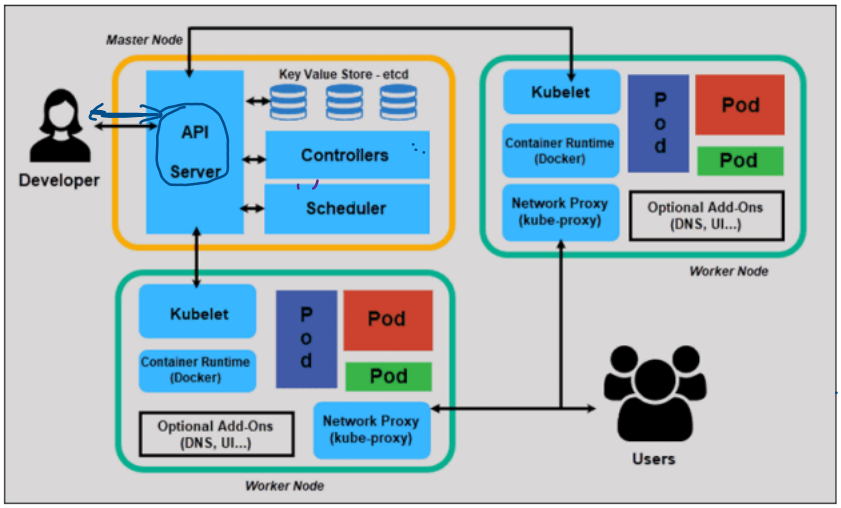
---

Master node:

API server, controllers, scheduler, etcd

Master nodes

Worker nodes



Master Node

- The management of a cluster is the responsibility of the master node as it is the first poir for almost all administrative tasks for the cluster.
- Depending upon the setup, there will be one or more master nodes in a cluster. This is d an eye on the failure tolerance.
- master node comprises different components such as Controller-manager, ETCD, Sched Server.
  1. **API Server**: It is the first point of contact for the entirety of the REST commands, v to manage and manipulate the cluster.
  2. **Scheduler**: The scheduler, as its name suggests, is responsible for scheduling tasks t nodes. It also keeps the resource utilization data for each of the slave nodes.
  3. **ETCD**: It is majorly employed for shared configuration, as well as for service disco basically a distributed key-value store.
  - Kubernetes uses etcd as a key-value database store. It stores the configuration of the cluster in etcd.
  - It also stores the actual state of the system and the desired state of the system in etcd
  - It then uses etcd's watch functionality to monitor changes to either of these two thing diverge, Kubernetes makes changes to reconcile the actual state and the desired state

4. **Controller-manager**: It is a daemon that is responsible for regulating the cluster in and it also manages various other control loops that are non-terminating.
   - ❖ **Replication controller:** Manages pod replication

- **Node controller**: Responsible for noticing and responding when nodes go down
- **Job controller**: Watches for Job objects that represent one-off tasks, then create those tasks to completion.
- **Endpoints controller**: Populates the Endpoints object (that is, joins Services &
- **Service Account & Token controllers**: Create default accounts and API access new namespaces

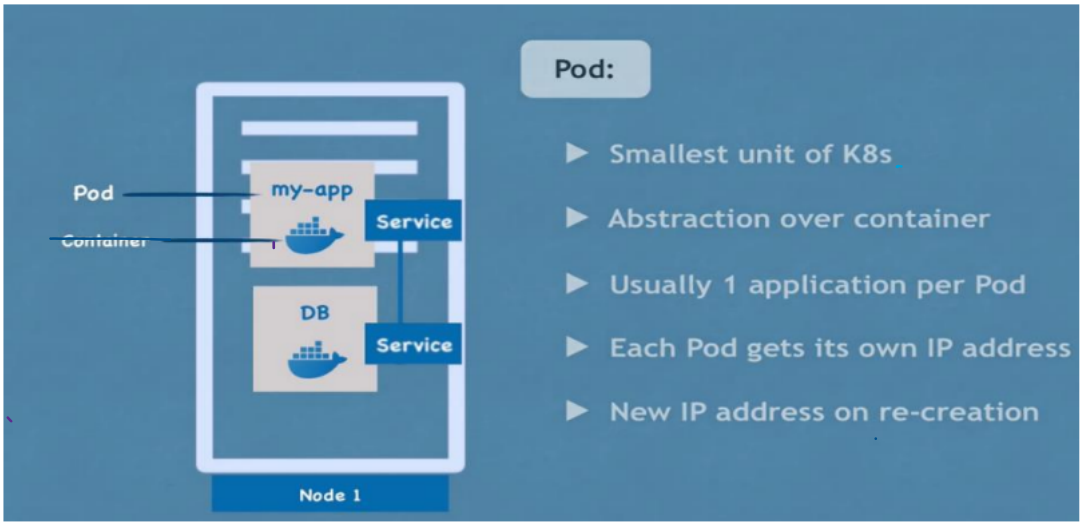
new namespaces

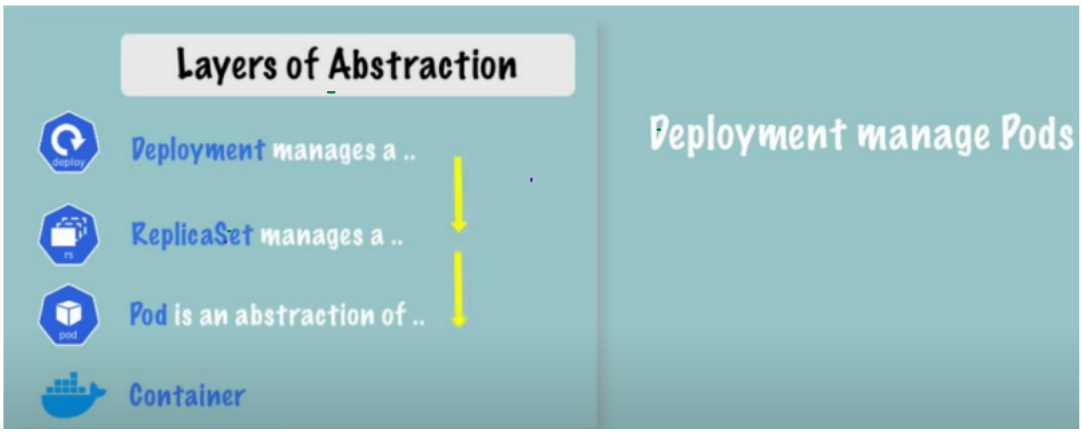
Worker/Slave Nodes

- Worker or slave nodes consist of all the needed services that are required to manage networking among containers.
- The services communicate with the master node and allocate resources to scheduled containers.
- worker nodes have the following components:

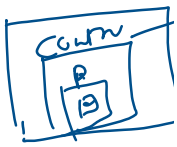
1. **Docker container**
   - Docker must be initialized and run on each worker node in a cluster.
   - Docker containers run on each worker node, and they also run the pods that are configured
2. **Kubelet**
   - The job of kubelet is to get the configuration of pods from the API server.
   - It is also used to ensure that the mentioned containers are ready and running.
3. **Kube-proxy**
   - Kube-proxy behaves like a network proxy and act as a load balancer for a service on any single worker node for pods running on the node by implementing east/west load-balancing using NAT in iptables
   - The kube-proxy handles network communications inside or outside the cluster
4. **CAdvisor:**
   - Used for monitoring resource usage and performance
5. **Label:**
   - Used to identify pods
6. **Pods**
   - A pod can be thought of as one or more containers, which can logically run on nodes together.

What is a pod?





- A group of one or more containers is called a Pod.
- Pods are also the simplest unit in the Kubernetes object model, which we can create and deploy.
- Pods life is short in nature, and they do not have the capability to self-heal by themselves. That is why we use them with controllers, which can handle a Pod's replication, fault tolerance, self-heal, etc.
- Examples of controllers are Deployments, Replica Sets, Replication Controllers, etc. We attach the Pod's specification to other objects using Pod Templates



Limitations of POD

- A pod is a single entity, and if it fails, it cannot restart itself. This won't suit most use cases, as we want our applications to be highly available. But Kubernetes has this issue solved by using with controllers

Value	Description
Pending	The pod is accepted by the Kubernetes system, but one or more of the container images are not created. This includes the time before it is scheduled and the time spent downloading images over the network, which could take a while
Running	The pod is bound to a node, and all of the containers are created. At least one container is still running or is in the process of starting or restarting
Succeeded	All containers in the pod have terminated in success and will not be restarted
Failed	All containers in the pod have terminated, and at least one container has terminated in failure, i.e., the container either exited with a non-zero status or was terminated by the system
Unknown	For some reason, the state of the pod could not be obtained, e.g., due to an error in communication with the host of the pod

Key feature of Kubernetes:

- Horizontal Scaling
- Auto Scaling
- Health check & Self-healing
- Load Balancer
- Service Discovery
- Automated rollbacks & rollouts
- Canary Deployment