Kubernetes for Developers

Kubernetes (**K8s**) is an open source container orchestration platform which automates the manual processes of deployment and managing containerized applications.

DockerSwarm:

A Docker Swarm is a group of either physical or virtual machines that are running the Docker application and that have been configured to join together in a cluster.The activities of the cluster are controlled by a swarm manager, and machines that have joined the cluster are referred to as nodes.

ref. docker run --memory 300M --cpus=0.5 --restart=on-failure

Difference between Docker Swarm and Kubernetes

It's pretty common to compare Kubernetes and Docker, however, a better comparison is Kubernetes vs Docker Swarm.

- Docker Swarm is an orchestration technology similar to Kubernetes and focuses on the clustering of Docker containers.
- A major difference between Docker and Kubernetes is that **Docker runs on a single node**, whereas **Kubernetes is designed to run across a cluster**.
- Another difference between Kubernetes and Docker is that **Docker can be used without Kubernetes**, whereas **Kubernetes needs a container runtime** in order to orchestrate.
- Kubernetes has been widely adopted, and at this point has become **standard for container management and orchestration**. Kubernetes provides an infrastructure-level framework for orchestrating containers at scale, and for managing user interaction with them.
- In much the same way, Docker has become the standard for container development and deployment. Docker provides a platform for developing, deploying, and running containers at a much more basic, nuts-and-bolts level. It is the ground on which the Kubernetes framework sits.

Kubernetes Glossary:

1) **Container**
A lightweight and portable executable image that contains software and all of its dependencies

2) **Cluster**
A set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node

4) **Image**
Stored instance of a Container that holds a set of software needed to run an application.

5) **Kubectl**
A command line tool for communicating with a Kubernetes API server.
You can use kubectl to create, inspect, update, and delete Kubernetes objects.

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

7) **Kubernetes API**
The application that serves Kubernetes functionality through a RESTful interface and stores the state of the cluster.

8) **Label**
Tags objects with identifying attributes that are meaningful and relevant to users

9) **Namespace**
An abstraction used by Kubernetes to support multiple virtual clusters on the same physical cluster

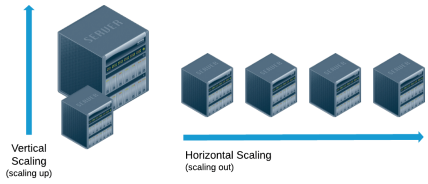
10) **Node**
A node is a worker machine in Kubernetes.
Considering as worker servers

11) **Pod**
The smallest and simplest Kubernetes object. A Pod represents a set of running containers on your cluster.

12) **ReplicaSet**
A ReplicaSet (aims to) maintain a set of replica Pods running at any given time

13) **Service**
An abstract way to expose an application running on a set of Pods as a network service.

14) **Vertical and Horizontal Scaling** (Vertical/Horizontal Pod/Node Autoscaling)



Architecture : Each node has multiple pods running on it.

Fig. One **Node** with two application pods with containers running on it

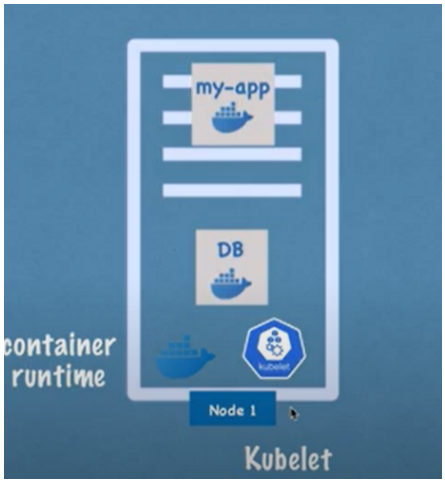
3) Processes must be installed on every node to schedule and manage those pods

Nodes are cluster service that actually does the work (Nodes/ worker nodes)

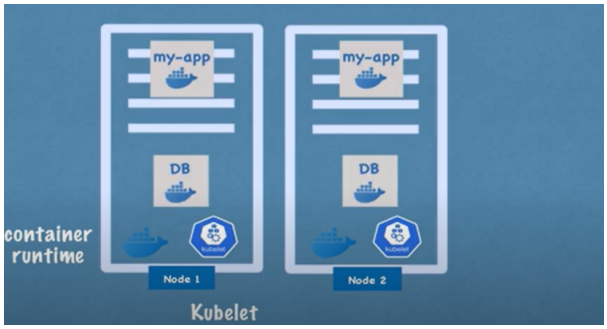
1) 1st process present on container should be **Container runtime** (eg Docker, Container D, Cri-O)
2) 2nd process **Kubelet** (process provided by kubernetes itself)

Kubelet has an interface with container runtime and node (machine) Itself. And is responsible for taking existing configuration and starting the pods With the container inside and assigning resources from the node to container

i.e CPU ram storage etc.



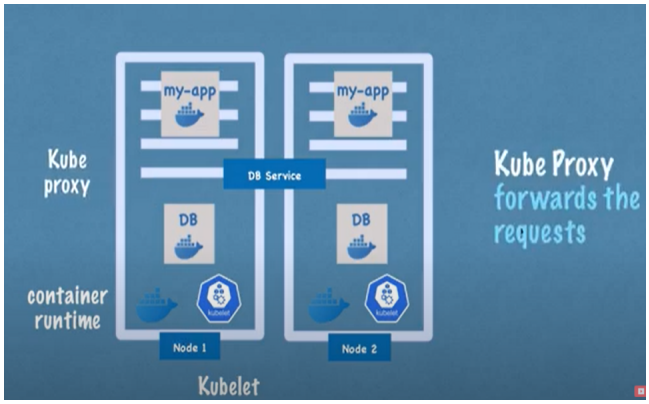
3) 3rd process **Kube Proxy**: Mostly kubernetes cluster comprises Multiple Nodes with Container runtime and kubelet services installed. Multiple Nodes will have replicas of existing pods.



So each worker node should be able to communicate with others using **services**

That should be a kind of **load balancer** that catches the request and forwards it to respective pods.

So the service responsible for forwarding requests from services to pods. Is **Kube Proxy** . That must be installed on every node



a). How to interact with this cluster ?

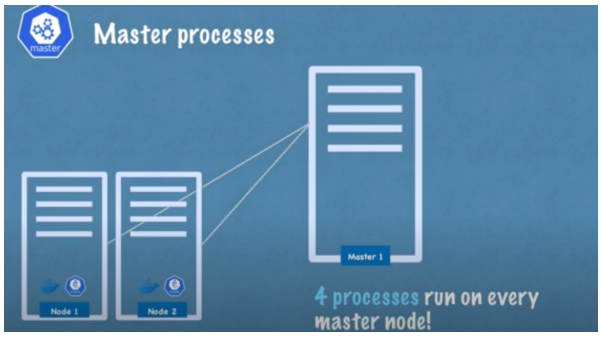
b) How to schedule POD, if replica pod dies how it is monitored or re scheduled/ restart POD

c) If we add a new node how does it join the cluster?

Master Node

All the managing process is done by Master Node

4 Processes to be run on every Master Node. So that master can control the cluster state and worker node itself



Master Node Processes

1) **API server** when we want to deploy new application in kubernetes cluster

We interact using API server through any client ie. command line (Kubelet), Any UI e.g. Rancher, Kubernetes Dashboard or kubernetes API.

API server is like cluster gateway, that gets updates or queries from the cluster

It also has Authentication, if the user is authenticated through API server then request is forwarded to other processes.

Can also query cluster for state, health and other monitoring reports

2) Second Process is **Scheduler.** If we send a request to API server to schedule a new pod after validating request form API server it hands over request to scheduler.

   Scheduler *has intelligence on which specific worker node to start pod/ scheduled* pod/ next component scheduled based on how many resources are available.

   Eg if one worker node is least busy it will schedule a new pod on that.

   NOTE: Scheduler just decides on which node pod will be scheduled.

   But actual work is done by processes running in worker nodes i.e kubelet. Means kubelet gets a request from Scheduler.

3) **Controller manager.** It handles scenarios where if pods die on any node then there should be a way to detect and schedule those pod Asap.

Controller manager detects state changes. Eg crashing of pods.

So it helps recover cluster state. It requests a scheduler to reschedule dead pods.

4) The 4th process is **Etcd.** I.e a key value store of a cluster state.

Consider a cluster brain. And when any state of the cluster changes.i.e. POD dies, created, or scheduled.

All **changes are stored in the key value store**. And is used by scheduler, controller manager etc using it's data.

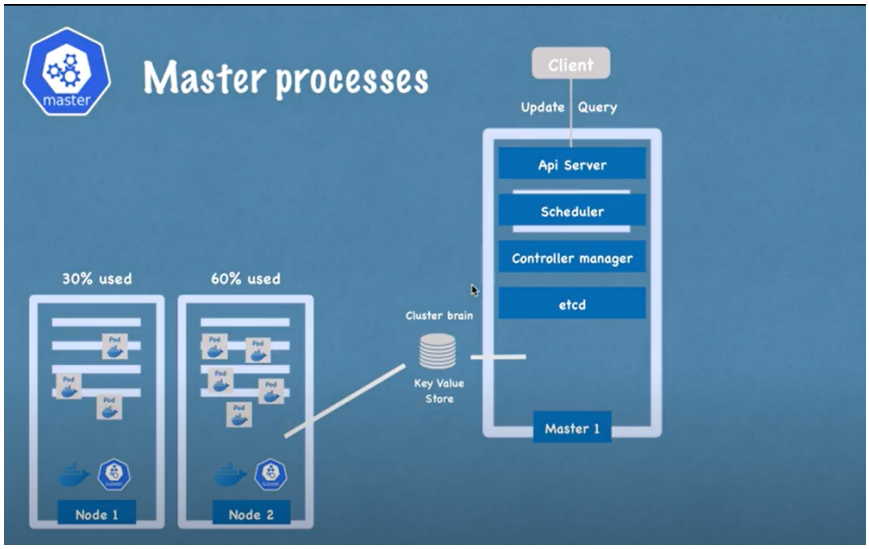
Eg. How the scheduler knows what resources are available in the worker node.As The controller manager knows, the cluster state changed in some way.On queries API server about update query, health data where it gets data from .

All info about nodes, clusters is stored in etcd.

Etcd is used by master processes to communicate with worker processes and vice versa.

Flow:

a) **Controller manager** detects state (of dying pod)
b) **Scheduler** (where the pod should be scheduled, based on resource calculation)
c) Scheduler checks **etcd** for configuration**.**
d) The **Kubelet** on the worker node creates a new pod.



Note: what is not stored in etcd is actual application data that is running on worker node as containerized applications.

Data stored in etcd should be reliably stored and replicated.

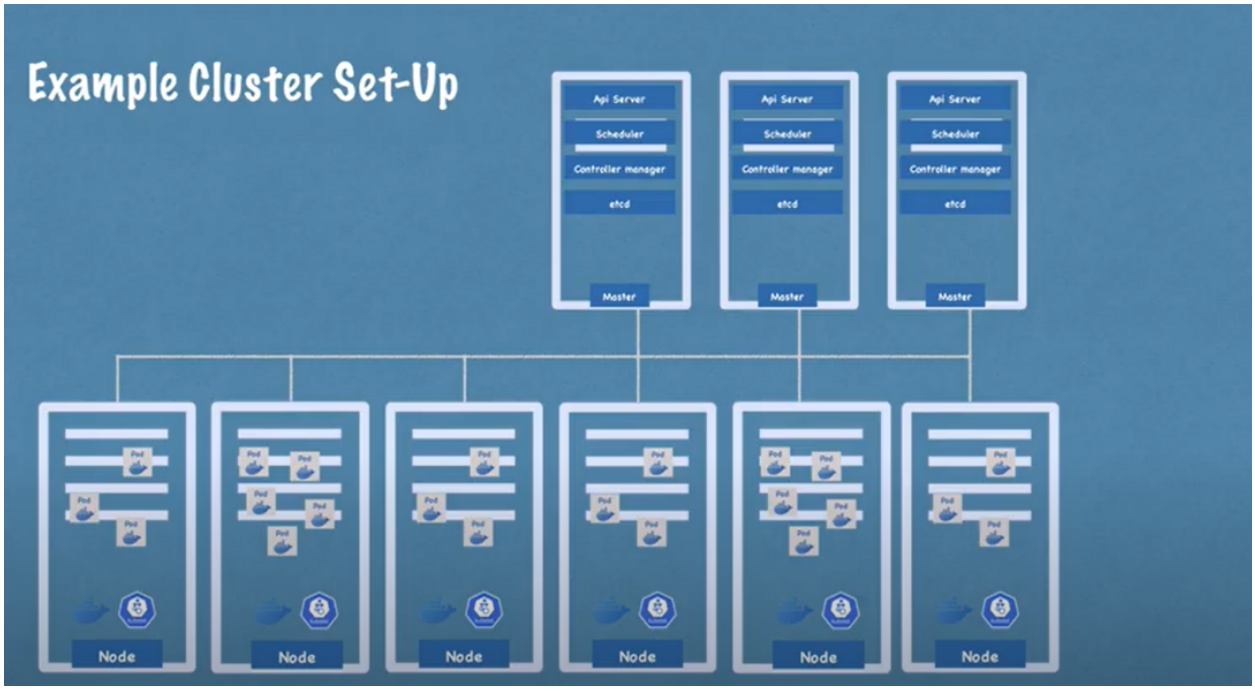
So the kubernetes cluster is made up of multiple masters.

Where the api server has a load balancer. And etcd data set are loaded on each master node

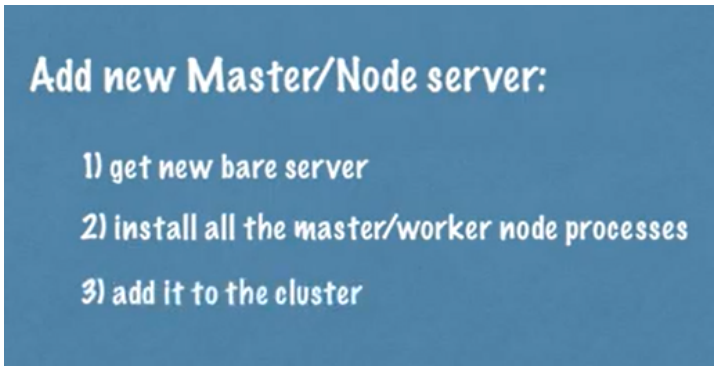
---

setup

1) Generally for small clusters we can have two master nodes, and three worker nodes.
2) Master node has less work to do compared to the worker node. So less resources are allocated to them.'
3) Can have robust cluster for bigger requirements



How to add a new node server?



How to access kubernetes cluster

Using kubectl/minikube

kubectl gets configured to access the kubernetes cluster control plane inside minikube when the `minikube start` command is executed.

However if `kubectl` is not installed locally, minikube already includes kubectl

Differences

- kubectl is a “client-side” tool (binary executable) that can connect to a kubernetes cluster to read, create, modify well known kubernetes resources (like pod, deployment, services)
- minikube is a way to set up a 1 noded kubernetes cluster running in a VM on top of a host (for example your own laptop). It is used to learn kubernetes. Production systems should use kubernetes clusters with 3 master nodes to achieve high availability.

Kubernetes Command Lines

How to install Minikube and Docker/Docker Driver?

- Minikube**:** https://minikube.sigs.k8s.io/docs/start/
- Docker: https://hub.docker.com/search?q=&type=edition&offering=community&sort=updated_at&order=desc
- Docker Driver for Kubernetes: https://minikube.sigs.k8s.io/docs/drivers/docker/

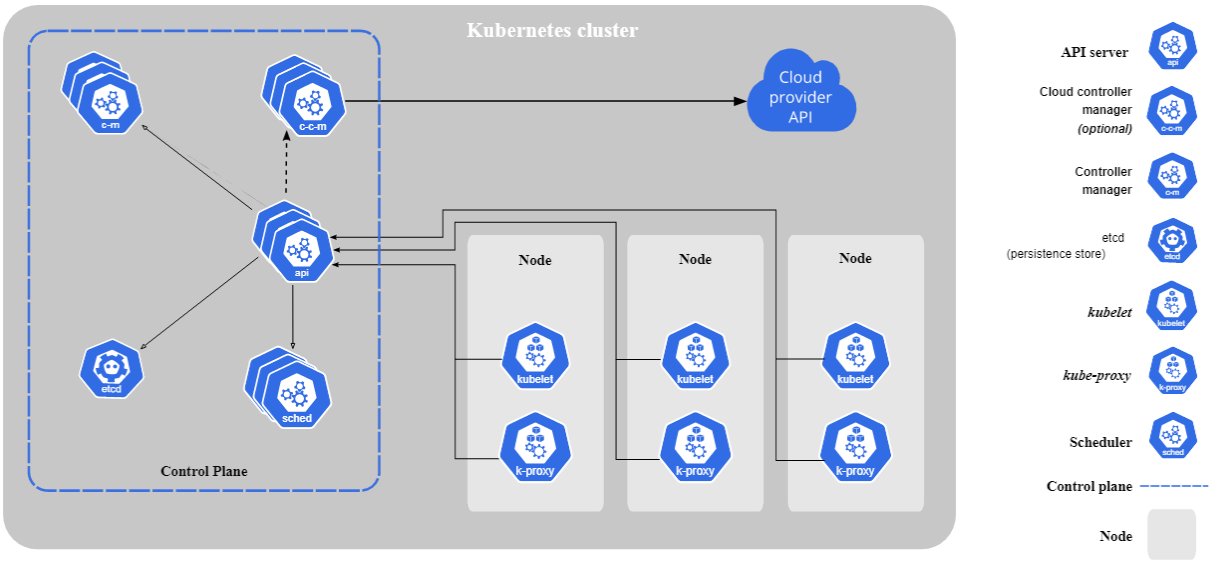
Minikube: Local Kubernetes Engine

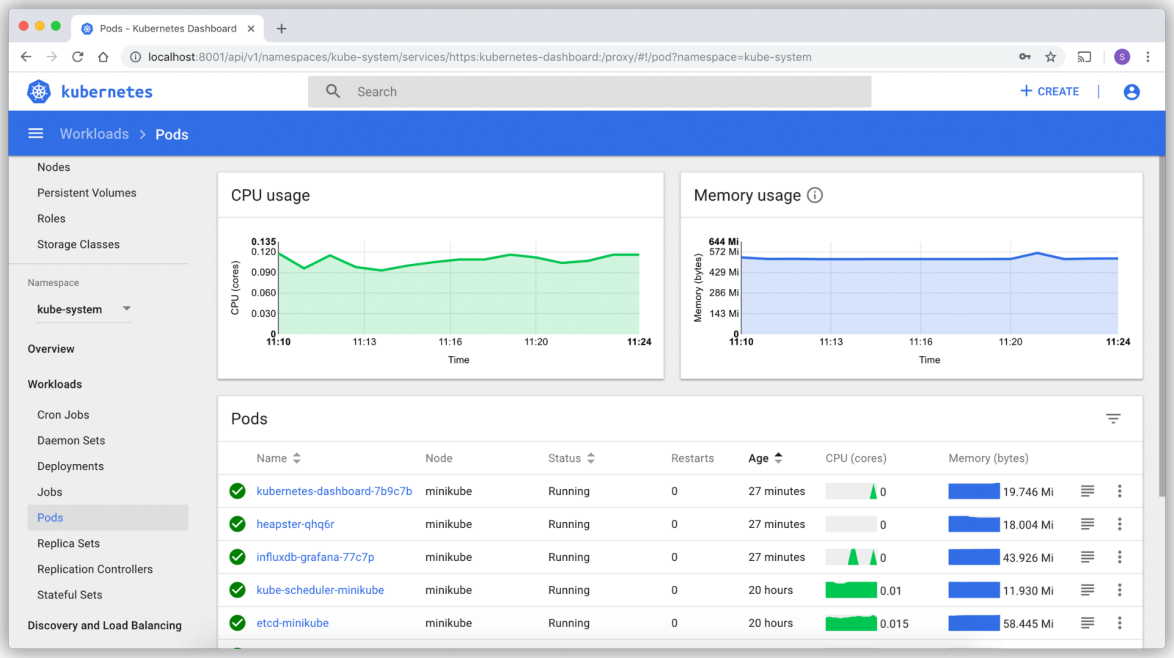
- *minikube start --nodes 2 -p hello-node* // start cluster engine with 2 nodes
- *minikube dashboard start -p hello-node* // view cluster, nodes, pods
- `minikube delete`

Kubernetes: Manage a cluster of Linux containers as a single system to accelerate Dev and simplify Ops

- *kubectl create deployment hello-node --image=k8s.gcr.io/echoserver:1.4*
- *kubectl get deployments*
- *kubectl get pods*
- *kubectl get nodes*
- *kubectl get events*
- *kubectl config view*
- *kubectl expose deployment hello-node --type=LoadBalancer --port=8080*
- *kubectl get services*
- *minikube service hello-node* // start exposed deployment
- --
- *minikube start --nodes 2 -p hello-node*
- *kubectl scale deployments/hello-node --replicas=4*
- *kubectl get deployments*

Kubernetes Dashboard





Command:

- minikube start
- minikube dashboard

Example:
http://127.0.0.1:54804/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/#/overview?namespace=default

What Developers Need to know in Kubernetes:

Developers should know:

- How to write yaml scripts (deployment, service, ingress, config map, secrets etc.)
- How to assign resources in production on kubernetes.
- How to use kubectl commands/ Kubernetes client eg Rancher
- How many pods are needed for a Live environment.
- The Glossary/Terminology of Kubernetes should be known. Eg Pods, Containers, Service yaml.
- When your application is large enough, to serve the clients. Then prefer Kubernetes.
- Microservices should be a preferred candidate for deployment on kubernetes.
- When scale nodes and pods? (by country, region, city)

References

- https://dev.to/techworld_with_nana/full-kubernetes-course-free-24hp
- https://www.katacoda.com/
- https://blog.scaleway.com/understanding-kubernetes-autoscaling/
- https://www.kasten.io/kubernetes/resources/books/kubernetes-backup-and-dr-for-dummies?twclid=11404396198446964744
- https://stackshare.io/stackups/kubernetes-vs-minikube
- https://minikube.sigs.k8s.io/docs/tutorials/multi_node/
- https://kubernetes.io/docs/tutorials/kubernetes-basics/
- https://www.replex.io/blog/how-to-install-access-and-add-heapster-metrics-to-the-kubernetes-dashboard