Index

01. Getting started with Kubernetes (K8S)
    - 01.1. Overview on Docker Containers
    - 01.2. Overview on Kubernetes (Container Orchestration)
02. Kubernetes Architecture
    - 02.1. Kubernetes Cluster
    - 02.2. Kubernetes Components
03. Setup Kubernetes Cluster
    - 03.1. Install Single Node Cluster (Minikube)
    - 03.2. Install Multi-Node Kubernetes Cluster (Weave Net CNI)
    - 03.3. Install Multi-Node Kubernetes Cluster (Calico CNI)
    - 03.4. Install Multi-Node Kubernetes Cluster (ContainerD)
    - 03.5. Install Kubernetes Cluster on Amazon EC2
    - 03.6. Deploy EKS Cluster on AWS with Terraform
04. Kubernetes Namespace
    - 04.1. Understanding Kubernetes Namespaces
    - 04.2. Why do we need Namespaces?
    - 04.3. List available Namespaces
    - 04.4. Creating Namespaces
    - 04.5. Creating Resource Objects in other Namespaces
    - 04.6. Terminating Namespaces
05. Kubernetes PODs
    - 05.1. Different methods to create objects in Kubernetes
    - 05.2. Overview on Kubernetes PODs
    - 05.3. Creating PODs using YAML file
    - 05.4. Check status of POD
    - 05.5. Check status of the container from the POD
    - 05.6. Connecting to the POD
    - 05.7. Perform port forwarding using Kubectl
    - 05.8. Understanding multi-container PODs
    - 05.9. Deleting PODs
    - 05.10. Running POD instances in a Job
06. Kubernetes init Containers
    - 06.1. Overview on Kubernetes init containers

06.2. Create a POD with init containers

06.3. How init containers work?

06.4. How init containers are different from normal containers?

07. Deploying Software Strategies

07.1. Canary Deployment strategy

07.2. Basic Deployment strategy

07.3. Rolling Update or Incremental Deployment strategy

07.4. Blue/Green Deployment strategy

07.5. A/B Testing Deployment strategy

08. Perform Rolling update with Kubernetes Deployments

08.1. Overview of Kubernetes Deployment

08.2. Create Kubernetes deployment resource

08.3. Understanding the Naming syntax of PODs as a part of Deployment

08.4. Understanding Kubernetes Labels, Selectors and Annotations

08.5. Using Kubernetes rolling update

08.6. Check Rollout History

08.7. Monitor the rolling update status

08.8. Pause and resume rollout process

08.9. Rolling back (undo) an update

09. Kubernetes Replica Set and Replication Controller

09.1. Overview on Replication Controllers

09.2. How does a Replication Controller works?

09.3. Creating a Replication Controller

09.4. Deleting a Replication Controller

09.5. Using Replica Sets instead of Replication Controller

09.6. Comparing a Replica Sets to Replication Controller

09.7. Example 1: Create Replica Set using match Labels

09.8. Example 2: Create Replica Set using match expressions

09.9. Delete Replica Set

10. Kubernetes Services

10.1. Why we need Kubernetes Services?

10.2. Overview on Kubernetes Services

10.3. Understanding difference Kubernetes Service types

10.4. Create Kubernetes Service

10.5. Delete Kubernetes Service
11. Kubernetes DaemonSet
11.1. Overview on Kubernetes DaemonSets
11.2. When to use DaemonSet and ReplicaSet?
11.3. Using a DaemonSet to run a POD on every Node
11.4. Limiting DaemonSets to specific Nodes
11.5. Deleting DaemonSets
12. Scheduling Jobs in Kubernetes
12.1. Structure of Scheduling Cron Job
12.2. Get details of Cron Job in Kubernetes
12.3. Create Cron Job
12.4. List available Cron Jobs
12.5. Monitor status of Cron Job
12.6. Delete Kubernetes Cron Job
13. Managing Kubernetes Storage
13.1. Overview on Kubernetes Volumes
13.2. Using Kubernetes Persistent Volumes (PV) with Persistent Volume Claims (PVC)
13.3. Using Local Persistent Volumes with storage class
13.4. Kubernetes ConfigMaps
13.5. Using Kubernetes secrets to pass sensitive data to containers
14. Kubernetes StatefulSets
14.1. Overview of Kubernetes StatefulSets
14.2. Limitations of Kubernetes StatefulSets
14.3. Create Kubernetes StatefulSets resources
14.4. Delete Kubernetes StatefulSets resources
15. Securing Kubernetes
15.1. Understanding Kubernetes API server
15.2. Using Host Nodes NameSpace in a POD
15.3. Configure container security context
15.4. Kubernetes Authentication
15.5. Authentication Strategies
15.6. Kubernetes Authorization
15.7. Authorization Modes

15.8. Configure RBAC (Role Based Authentication Control) to define New role with “modify” permission
15.9. Configure RBAC to define new role with “View” only permission
15.10. Deleting Contexts
15.11. Deleting Role
15.12. Deleting Role Binding
16. Limit Kubernetes Resources
16.1. Resource Quota for NameSpaces
16.2. Resource Quota types
16.3. Understanding Limit Range
16.4. Different Kubernetes resource types
16.5. Resource request and limits for POD & Containers
16.6. Understanding Resource units
16.7. How PODs with resource limits are managed
16.8. Example: Define CPU & Memory Limit for Containers
16.9. If don’t specify CPU limit what happens?
17. Expose Containers to external networks (Kubernetes Networking)
17.1. Perform Port Forwarding using Kubectl
17.2. Kubernetes services to expose applications
17.3. Kubernetes Ingresss

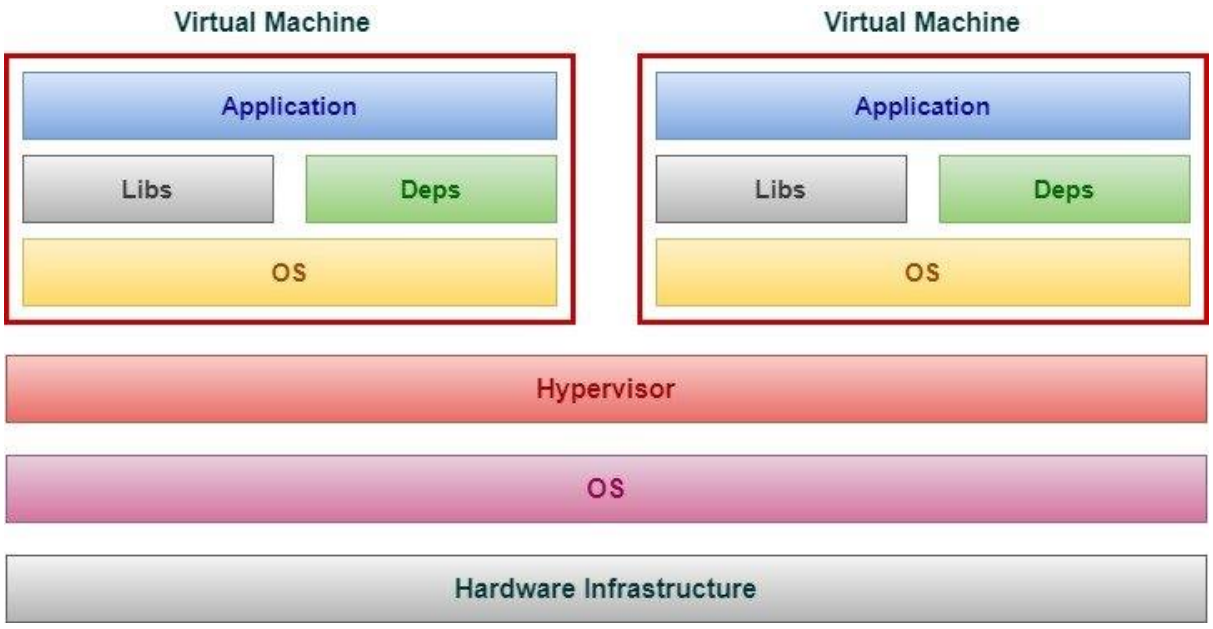
1. Getting started with Kubernetes (K8S)

Kubernetes also known as K8s was created by Google based on their experience from containers in production. It is an open source project and one of the best and most popular container orchestration technology.

Before we understand Kubernetes, we must be familiar with Docker containers or else you if you are a beginner then you may find Kubernetes a little confusing to understand.

1.1. Overview on Docker Containers:

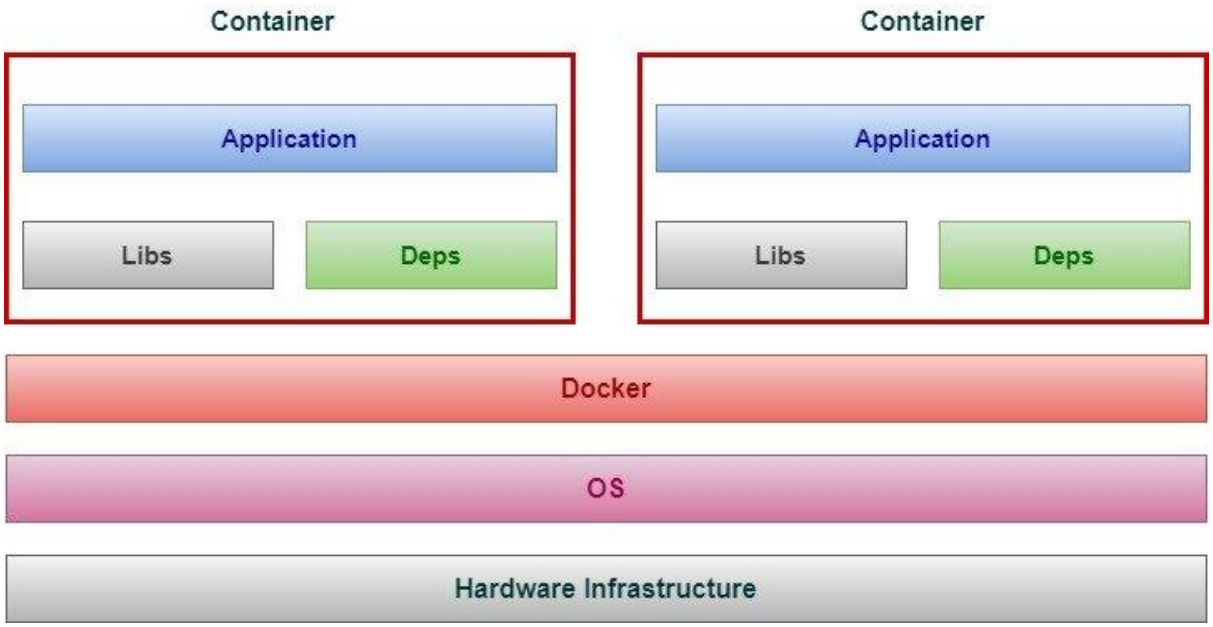
Using Virtual Machine (VMs):



- Instead of running multiple applications, all on the same server, companies would package and **run a single application on each VM**
- With this, all the compatibility problems were gone and life seemed to be good again.
- But this comes with it's own set of demerits where each VM needs a lot of resources where most is used by underlying system OS.

Using Docker containers

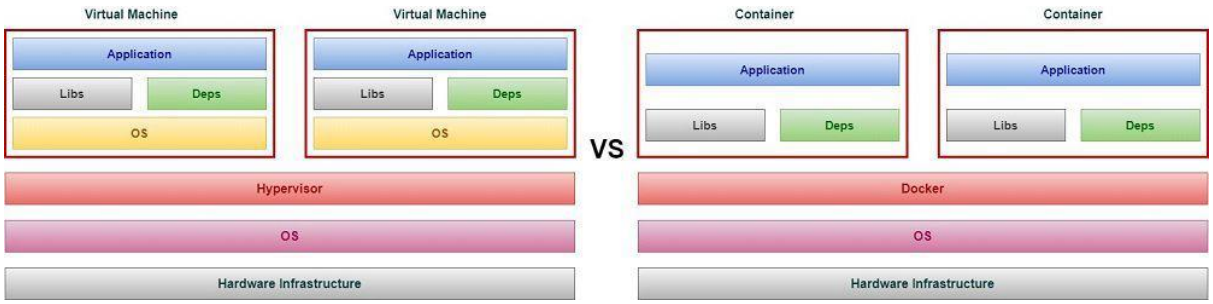
The ultimate solution to this problem was to provide something that is much more lightweight than VMs - **Docker container** to the rescue.



- Instead of virtualizing hardware, containers rest on top of single Linux instance. This allows Docker to leave behind a lot of bloat associated with full hardware hypervisor.
- **Don't mistake** Docker Engine (or the LXC process) as the equivalent of a hypervisor in more traditional VM, it is simply encapsulating process on the underlying system.
- Docker utilizes the **namespace** feature of Linux kernel wherein the namespaces will make the processes that are running within one container are invisible for processors, or users running within another container
- With docker, developers would now package their application, dependent libraries, framework in a container to the testers or operation engineer.
- To testers and operations engineers, a container is just a black box, and all they need is a Linux OS with Docker running and they can easily deploy the container without having to worry about configuring the application as these containers already contain an up and running application.

Virtual Machine vs Docker Containers

The image should be self explanatory to understand the difference between **Docker and VMware** architecture.

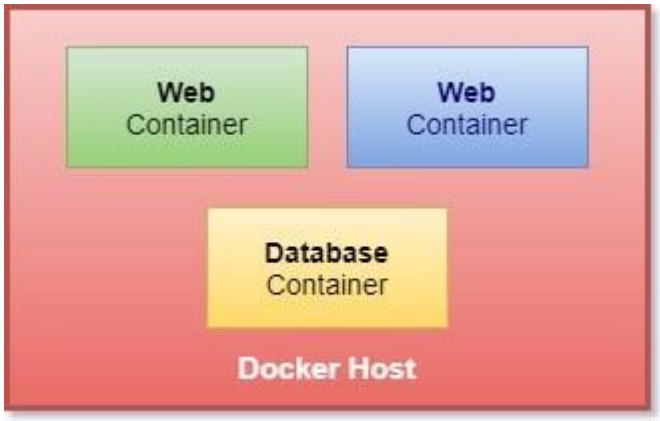


- VM requires an Hypervisor which can be either installed on an operating system or directly on the hardware while a container can be deployed after installing docker.

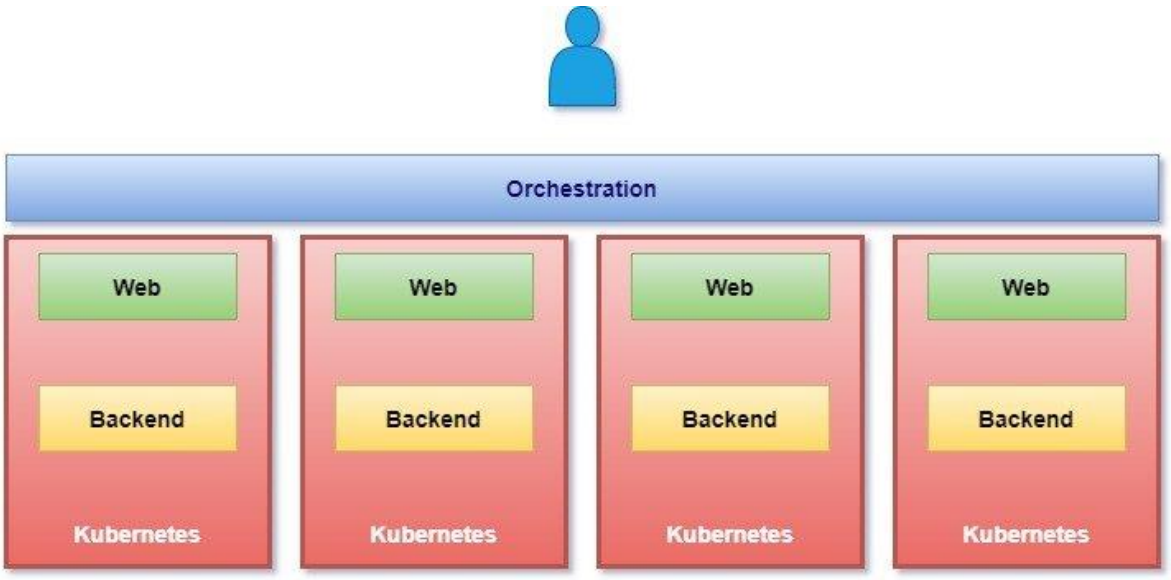
- VM requires a separating OS to be installed to deploy your application while Docker containers share the host operating system, and that is why they are lightweight
- Since Docker shares OS with host, the boot up time of docker container is very less while it is comparatively higher for VMs
- The docker containers share Linux kernel so it would be a good fit if you are planning to run multiple applications on the same Linux kernel but if you have applications that require different operating system then you will have to go for VM
- Since VM does not share the host OS it is comparatively more secure than Docker containers. An attacker may exploit all the containers if it gets access to the host or any single container.
- Since containers don't have OS they use comparatively very less resources to execute the application and you can utilize the underlying resources more effectively

1.2. Overview on Kubernetes (Container Orchestration):

Now that you are familiar with containers, next we need to learn about **container orchestration**. Just to summarize we have a docker container with certain applications running inside the container.



- It is possible that your application from container 1 is dependent on some other application from another container such as database, message, logging service in the production environment.
- You may also need the ability to scale up the number of containers during peak time, for example I am sure you must be familiar with Amazon sale during holidays when they have a bunch of extra offers on all products. In such case they need to **scale up** their resources for applications to be able to handle more number of users. Once the festive offer is finished then they would again need to **scale down** the amount of containers with applications.
- To enable this functionality we need an underlying platform with a set of resources and capabilities. The platform needs to **orchestrate** the connectivity between the containers and automatically scale up or down based on the load.
- This while process of deploying and managing containers is known as **container orchestration**
- **Kubernetes is thus a container orchestration technology** used to orchestrate the deployment and management of hundreds and thousands of containers in a cluster environment.
- There are multiple similar technologies available today, docker has it's own orchestration software i.e. Docker Swarm, Kubernetes from Google and Mesos from Apache.



- Your application is now **highly available** as now we have multiple instances of your application across multiple nodes
- The user traffic is load balanced across various containers
- When demand increases deploy more instances of the applications seamlessly and within a matter of seconds and we have the ability to do that at a service level when we run out of hardware resources then scale the number of underlying nodes up and down without taking down the application and this all can be done easily using a set of declarative object configuration file.

History:

- Google developed an internal system called "**BORG**" (later renamed as "**OMEGA**") to deploy and manage thousands of google applications and services on their cluster.
- In 2014 google introduced Kubernetes as an Open Source Platform written in "GoLang" and later donated to "**CNCF**"(Cloud Native Computing Foundation)

Online Platform for K8S(Kubernetes):

1. Kubernetes Play Ground
2. Play with K8S
3. Play with Kubernetes classroom

Cloud based K8S services:

1. GKE (Google Kubernetes Services)
2. AKS (Azure Kubernetes Services)
3. Amazon EKS (Elastic Kubernetes Services)

Kubernetes Installation Tools:

1. Minicube
2. Kubeadm

Problem with scaling up the containers:

1. Containers cannot communicate with each other.
2. Auto scaling and Load Balancing was not possible
3. Containers have to be managed carefully

Features of Kubernetes:

- Orchestration (Cluster of any number of containers running on different networks)
- Auto Scaling (Vertical and Horizontal)
- Auto Healing
- Load Balancing
- Platform Independent (Cloud/Virtual/Physical)
- Fault Tolerance (Node/POD Failure)
- Rollback (Going back to previous version)
- Health Monitoring of Containers
- Batch execution (one time, sequential, parallel)

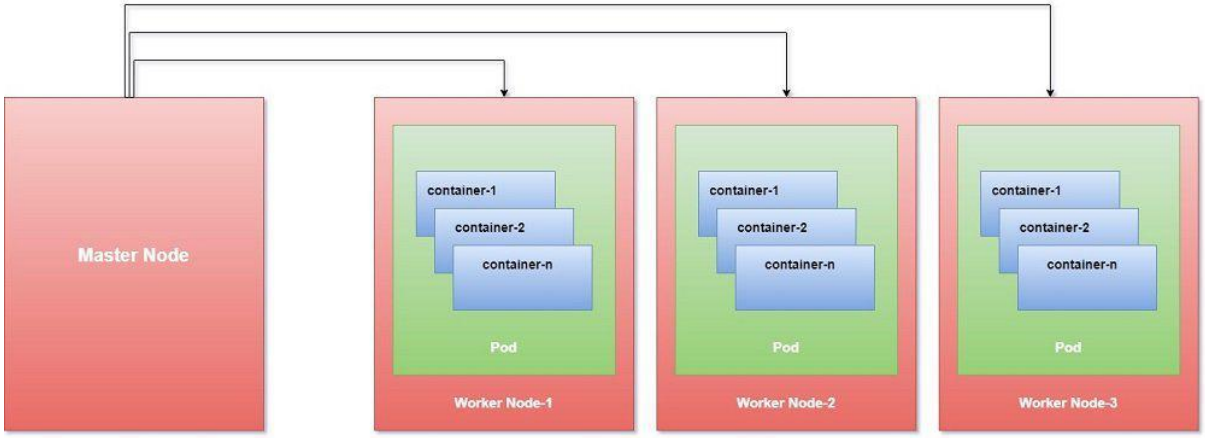
Kubernetes Vs Docker Swarm:

S.No	Features	Kubernetes	Docker Swarm
1	Installation and Cluster configuration	Complicated and time consuming	Fast and Easy
2	Supports	K8S can work with almost all container types like: rocket, docker, ContainerD.	Works with Docker only
3	GUI	GUI is available.	Not available
4	Data Volumes	Can only shared with containers in same POD	Can be shared with any other container
5	Updates and Rollback	Process scheduling to maintain services while updating	Progressive updates and service health monitoring throughput the update.
6	Auto Scaling	Supports Vertical and Horizontal auto scaling	No support
7	Logging and Monitoring	In-built tool present for monitoring	Uses third party tools like splunk

2. Kubernetes Architecture

A **Kubernetes Cluster** consists of Master and Client node setup where we will have one Master or **Controller** node along with multiple Client nodes also referred as **worker** nodes or in minions.

A **Master** is a node with Kubernetes installed and is responsible for the actual orchestration of containers on the worker nodes. It will contain all the information of cluster nodes, monitor each node and if a worker node fails then it will move the workload from the failed node to another worker node.



A **Node** is a worker machine which can be a physical or virtual machine on which Kubernetes is installed.

Kubernetes does not deploy containers directly into the worker nodes, the containers are encapsulated into a Kubernetes object known as **Pods**.

A pod is a single instance of an application and they are the smallest deployable units of computing that you can create and manage in Kubernetes.

You will deploy containers inside these pods where you will deploy your application

Following is a basic architectural diagram of Kubernetes and the relationship between containers, pods, and physical worker nodes which were referred as Minion in past.

2.1. Kubernetes Cluster:

There are three main components in the Kubernetes Cluster i.e. Nodes, Pods and Containers.

Master

- The master is the control plane of Kubernetes.
- It consists of several components, such as an **API server, a scheduler**, and a **controller manager**.
- The master is responsible for the global state of the cluster, cluster-level scheduling of pods, and handling of events. Usually, all the master components are set up on a single host.
- When considering high-availability scenarios or very large clusters, you will want to have master redundancy.

Nodes (Minion)

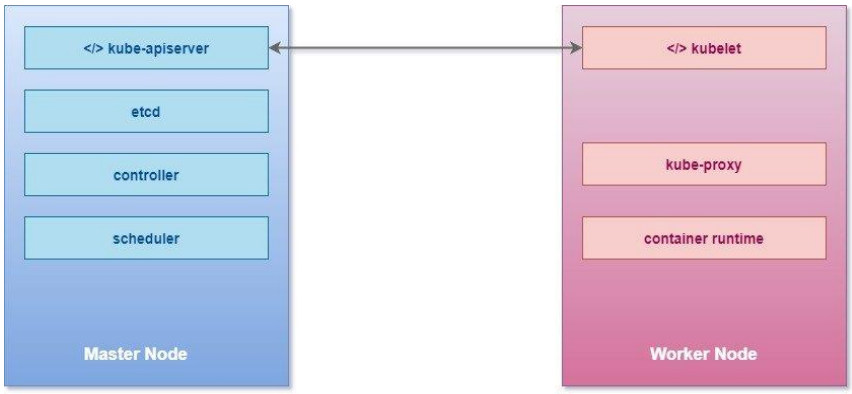
- You can think of these as container clients.
- These are individual hosts (physical or virtual) on which Docker would be installed to host different containers within your managed cluster
- Each Node will run ETCD (key pair management and communication service, used by Kubernetes for exchanging messages and reporting Cluster status) as well as the Kubernetes proxy

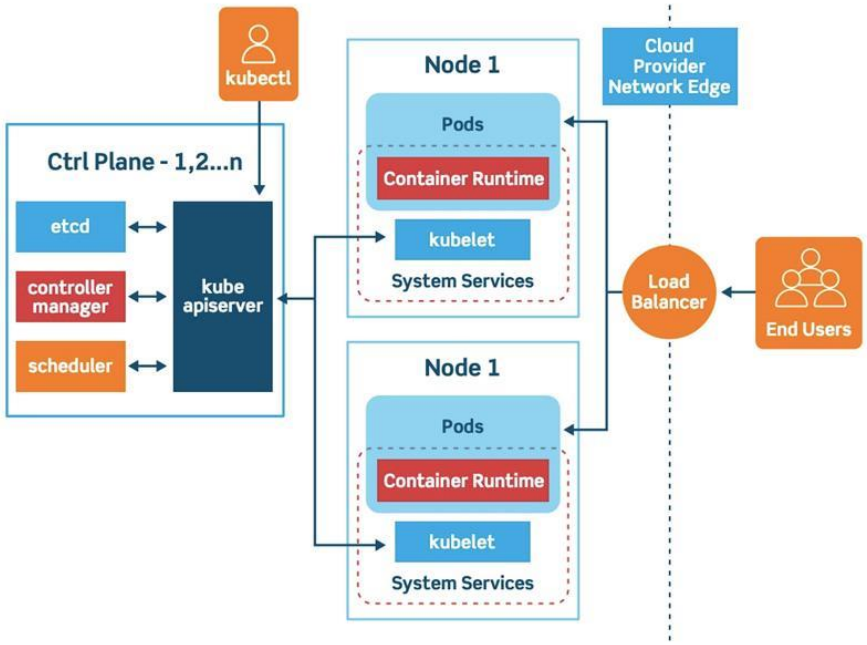
Pods

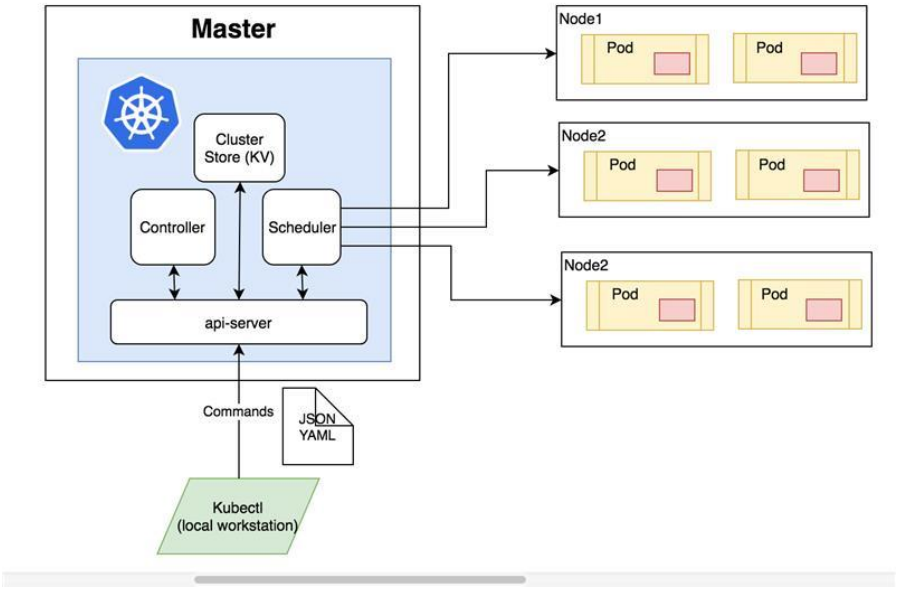
- A Pod is a group of one or more containers, with shared storage/network resources, and a specification for how to run the containers.
- We're not implying that a pod always includes more than one container—it's common for pods to contain only a single container.
- The key thing about pods is that when a pod does contain multiple containers, all of them are always run on a single worker node—it never spans multiple worker nodes
- These containers are guaranteed (by the cluster controller) to be located on the same host machine in order to facilitate sharing of resources
- Pods are assigned unique IP address within each cluster. These allow an application to use ports without having to worry about conflicting port utilization
- Pods can contain definitions of disk volumes or share and then provide access from those to all the members (containers) with the pod.

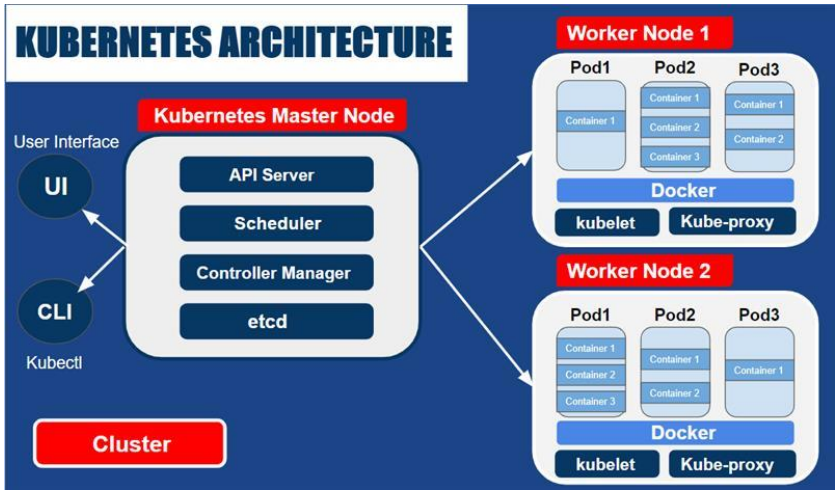
2.2. Kubernetes Components

When you are installing Kubernetes, basically you are working with following components:




kubectl
Ctrl Plane - 1,2...n
etcd
controller manager
scheduler
kube apiserver
Node 1
Pods
Container Runtime
kubelet
System Services
Node 1
Pods
Container Runtime
kubelet
System Services
Cloud Provider Network Edge
Load Balancer
End Users


Master
Cluster Store (KV)
Controller
Scheduler
api-server
Commands
JSON YAML
Kubectl (local workstation)
Node1
Pod
Pod
Node2
Pod
Pod
Node2
Pod
Pod



API Server

- The API server acts as a frontend for Kubernetes
- It can easily scale horizontally as it is stateless and stores all the data in the etcd cluster.
- The users, management devices, command line interfaces, all talk to API server to interact with Kubernetes cluster

etcd key store

- It is a distributed reliable key value store
- Kubernetes uses it to store the entire cluster state
- In a small, transient cluster a single instance of etcd can run on the same node with all the other master components.
- But for more substantial clusters, it is typical to have a three-node or even five-node etcd cluster for redundancy and high availability.
- It is responsible for implementing locks within the clusters to ensure that there are no conflicts between the masters

Scheduler

Kube-scheduler is responsible for scheduling pods into nodes. This is a very complicated task as it needs to consider multiple interacting factors, such as the following:

- Resource requirements
- Service requirements
- Hardware/software policy constraints
- Node affinity and anti-affinity specifications
- Pod affinity and anti-affinity specifications
- Taints and tolerations
- Data locality
- Deadlines

Controllers

- The controllers are the brain behind Orchestration.
- They are responsible for noticing and responding when Nodes, containers or end points goes down
- The controller makes decision to bring up new containers in such case

Controller Runtime

It is the underlying software that is used to run containers. In our case we will be using Docker as the underlying container but there are other options as well such as:

- Docker (via a CRI shim)
- rkt (direct integration to be replaced with Rktlet)
- CRI-O
- Frakti (Kubernetes on the Hypervisor, previously Hypernetes)
- rktlet (CRI implementation for rkt)
- CRI-containerd

The major design policy is that Kubernetes itself should be completely decoupled from specific runtimes. The **Container Runtime Interface (CRI)** enables it.

kubelet

It is the agent that runs on each nodes in the cluster. The agent is responsible for making sure that the containers are running on the nodes as expected. That includes the following:

- Receiving pod specs
- Downloading pod secrets from the API server
- Mounting volumes
- Running the pod's containers (via the configured runtime)
- Reporting the status of the node and each pod
- Running the container startup, liveness, and readiness probes

kube-proxy

- Kube-proxy does low-level network housekeeping on each node
- Containers run on the server nodes, but they interact with each other as they are running in a unified networking setup.
- kube-proxy makes it possible for containers to communicate, although they are running on different nodes.
- It reflects the Kubernetes services locally and can perform TCP and UDP forwarding.
- It finds cluster IPs via environment variables or DNS.

Conclusion

- Linux containers provide much the same benefits as virtual machines, but are far more lightweight and allow for much better hardware utilization.
- Docker improved on existing Linux container technologies by allowing easier and faster provisioning of containerized apps together with their OS environments.
- Kubernetes exposes the whole datacenter as a single computational resource for running applications.
- Developers can deploy apps through Kubernetes without assistance from sysadmins.
- Sysadmins can sleep better by having Kubernetes deal with failed nodes automatically.

3. Setup Kubernetes Cluster

3.1. Install Single Node Cluster (Minikube):

- If you are using physical (bare-metal) servers or **virtual machines** (**VMs**), Kubeadm is a good fit.
- If you're running on cloud environments, kops and Kubespray can ease Kubernetes installation, as well as integration with the cloud providers.
- If you want to drop the burden of managing the Kubernetes control plane, almost all cloud providers have their Kubernetes managed services, such as **Google Kubernetes Engine** (**GKE**), **Amazon Elastic Kubernetes Service** (**EKS**), **Azure Kubernetes Service** (**AKS**), and **IBM Kubernetes Service** (**IKS**).
- If you just want a playground to study Kubernetes , **Minikube** and **Kind** can help you spin up a Kubernetes cluster in minutes.

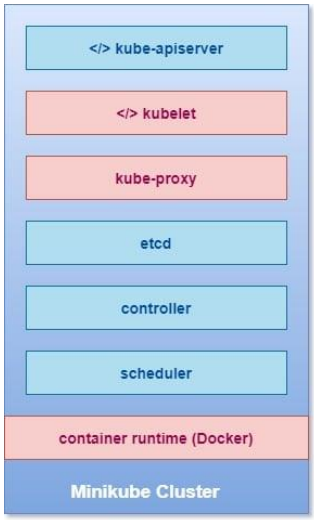
So, as you see you have a bunch of options to choose from to deploy your first Kubernetes Cluster. We will cover the steps to **install kubernetes cluster on CentOS 8 using Minikube**.

Overview on Minikube

Minikube is a tool that can be used to set up a **single-node cluster**, and it provides handy commands and parameters to configure the cluster. It primarily aims to provide a local testing environment. It packs a VM containing all the core components of Kubernetes that get installed onto your host machine, all at once. This allows it to support any operating system, as long as there is a virtualization tool (also known as a **Hypervisor**) pre-installed.

Minikube Architecture

A Kubernetes cluster consists of a controller and worker node where both node types have their own set of components. But since Minikube is a single node cluster, it will contain all the cluster components inside this single node, which would look something like following:



Pre-requisites

The **minimum resource requirement** for your physical host:

- 2 CPUs or more
- Minimum 2GB of free memory
- Minimum 20GB of free disk space

Additionally **your host must have:**

- Working internet connection
- Virtualization technology must be enabled in BIOS to support hypervisor
- Anyone of the supported hypervisor

The following are the most common **Hypervisors** supported by Minikube:

- VirtualBox (works for all operating systems)
- KVM (Linux-specific)
- Hyperkit (macOS-specific)
- Hyper-V (Windows-specific)

Lab Environment

I will use my Laptop which is installed with **Windows 10** and **Oracle VirtualBox** to setup Minikube.

Install Oracle VirtualBox

You may also choose to install different hypervisor based on your environment, we will use **Oracle VirtualBox**. Download VirtualBox software from their official repository.

Download and install kubectl

The Kubernetes command-line tool, `kubectl`, allows you to run commands against Kubernetes clusters. You can use `kubectl` to deploy applications, inspect and manage cluster resources, and view logs.

- If you are using a different platform other than Windows then you can follow official guide to download the kubectl for your platform
- I have created a folder inside my C drive "`C:\Kubernetes`" where I will be storing these software.
- As the time of writing this tutorial, **kubectl 1.19** was the latest stable release available.
- Or if you have curl installed, use this command:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.19.0/bin/windows/amd64/kubectl.exe
```

- To find out the latest stable version (for example, for scripting), take a look at https://storage.googleapis.com/kubernetes-release/release/stable.txt.
- Test to ensure the version of `kubectl` is the same as downloaded:

```
C:\Kubernetes>kubectl version --client
```

Output from my setup:

```
c:\Kubernetes>kubectl version --client
Client Version: version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.0", GitCommit:"e1996418337
7d0ec2052d1f1fa930c4d7575bd50", GitTreeState:"clean", BuildDate:"2020-08-26T14:30:33Z", GoVersio
n:"go1.15", Compiler:"gc", Platform:"windows/amd64"}
```

Download and install Minikube

- You can follow the official guide to get the latest available minikube package for your respective platform. For Windows you can download the latest available minikube from https://storage.googleapis.com/minikube/releases/latest/minikube-installer.exe
- I have downloaded minikube and placed it inside `C:\Kubernetes` folder which I created in the previous step.
- Next you can double click on the downloaded installer and just follow the on screen instructions to install the software.
- Once installed, open a command prompt and enter "`minikube version`" to make sure it is working as expected.

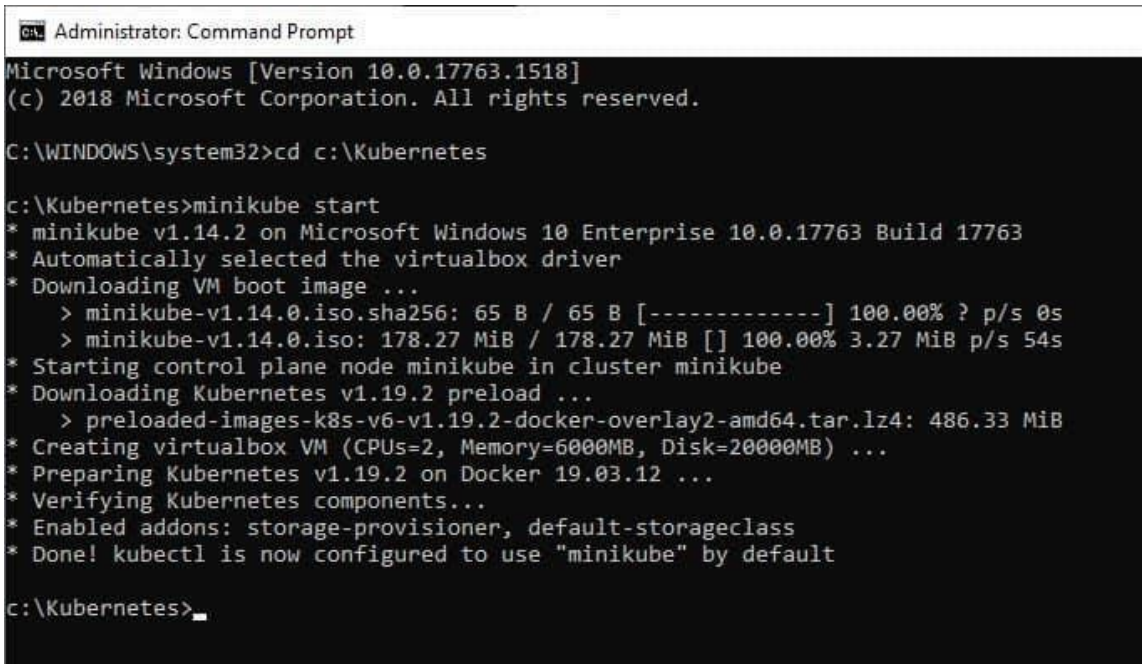
```
c:\Kubernetes>minikube version
minikube version: v1.14.2
commit: 2c82918e2347188e21c4e44c8056fc80408bce10
```

Build and start your minikube cluster

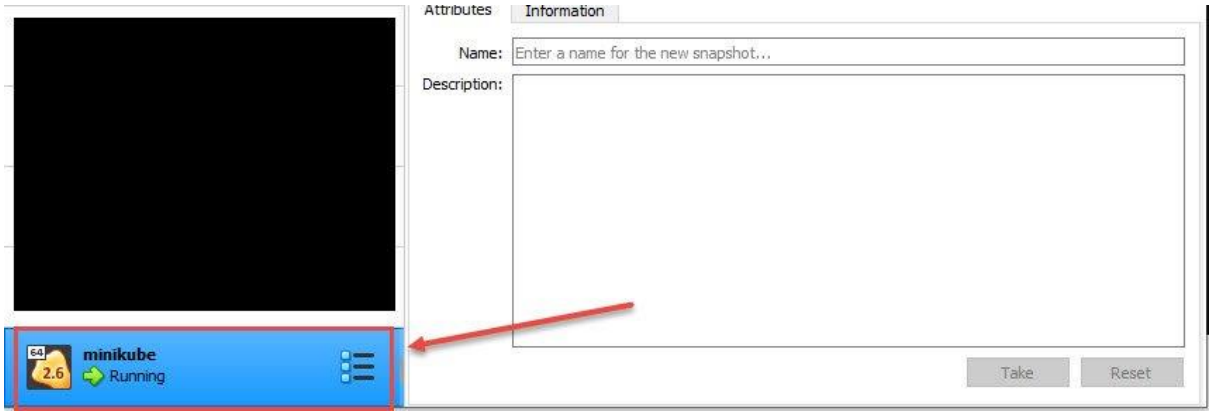
Next we will deploy our Kubernetes Cluster for which we will execute "**minikube start**". This command will download the minikube ISO image and deploy a new Virtual Machine using Oracle VirtualBox which will take some time depending upon your network speed.

NOTE:

You don't need to create any VM, this step itself will create and deploy a new VM on Oracle VirtualBox. Moving forward after you shutdown your VM, you should again use `minikube start` to start the cluster services before you can access it.



minikube has successfully deployed our single node cluster, you can verify the same on your Oracle VirtualBox where you should have a new VM in running state.



Interact with your cluster

The following commands should help establish that the Kubernetes cluster that was started by Minikube is running properly.

```
C:\Kubernetes>minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

Now, let's look at the version of the kubectl client and Kubernetes server:

```
C:\Kubernetes>kubectl version --short
Client Version: v1.19.0
Server Version: v1.19.2
```

To check the machines comprise the cluster and get some basic information about them:

```
C:\Kubernetes>kubectl get node
NAME       STATUS   ROLES    AGE    VERSION
minikube   Ready    master   112m   v1.19.2
```

Now you should have Minikube set up with a **single-node Kubernetes cluster**.

Deploy applications

Now that our cluster node is UP and running, we can create our first Pod (which is basically a container), but before that verify if there is already any pod available on the minikube cluster:

```
C:\Kubernetes>kubectl get pods
No resources found in default namespace.
```

Since the cluster is freshly installed, we did not had any pods. I will create a new pod "`hello-minikube`"

```
C:\Kubernetes>kubectl create deployment hello-minikube --
image=k8s.gcr.io/echoserver:1.4
pod/hello-minikube created
```

Now verify the available pods, currently the status shows as "`ContainerCreating`"

```
C:\Kubernetes>kubectl get pods
NAME                              READY   STATUS              RESTARTS
AGE
hello-minikube-6ddfcc9757-gtr9s   1/1     ContainerCreating   1
19h
```

Wait for sometime and execute the same command, now the status of the pod shows as running:

```
C:\Kubernetes>kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
hello-minikube-6ddfcc9757-gtr9s   1/1     Running   1          3m2s
```

This is basically a webserver and to be able to access it we need to execute following command:

```
C:\Kubernetes>kubectl expose deployment hello-minikube --type=NodePort
--port=8080
service/hello-minikube exposed
```

To get the URL of the web server

```
C:\Kubernetes>minikube service hello-minikube --url
http://192.168.99.100:31068
```

Now we can use this URL to access our application

We are done with this demonstration so I will delete the deployment

```
C:\Kubernetes>kubectl delete deployment hello-minikube
deployment.apps "hello-minikube" deleted
```

Now we don't have any more pods running:

```
C:\Kubernetes>kubectl get pods
No resources found in default namespace.
```

Manage your cluster

To pause Kubernetes without impacting deployed applications use:

```
minikube pause
```

Lastly I will stop my minikube cluster which will power off the virtual machine:

```
C:\Kubernetes>minikube stop
* Stopping node "minikube"  ...
* 1 nodes stopped.
```

To delete all the minikube clusters

```
minikube delete --all
```

Conclusion

Minikube is mostly used to setup (Proof-of-Concept) POC environment to learn Kubernetes Cluster. Although I would be using kubeadm to setup multi-node Kubernetes Cluster in my next article which I will use to demonstrate the entire Kubernetes Tutorial. In this article, you learned the steps to deploy a cluster using minikube on your WIndows host machine, similarly you can deploy cluster on any other host environment such as Ubuntu, CentOS etc.

3.2. Install Multi-Node Kubernetes Cluster (Weave Net CNI)

3.3. Install Multi-Node Kubernetes Cluster (Calico CNI)

3.4. Install Multi-Node Kubernetes Cluster (ContainerD)

3.5. Install Kubernetes Cluster on Amazon EC2:

3.6. Deploy EKS Cluster on AWS with Terraform