**What is Kubernetes ?**

It is a tool for container Orchestration. That means managing & Controlling multiple docker containers as a single service.

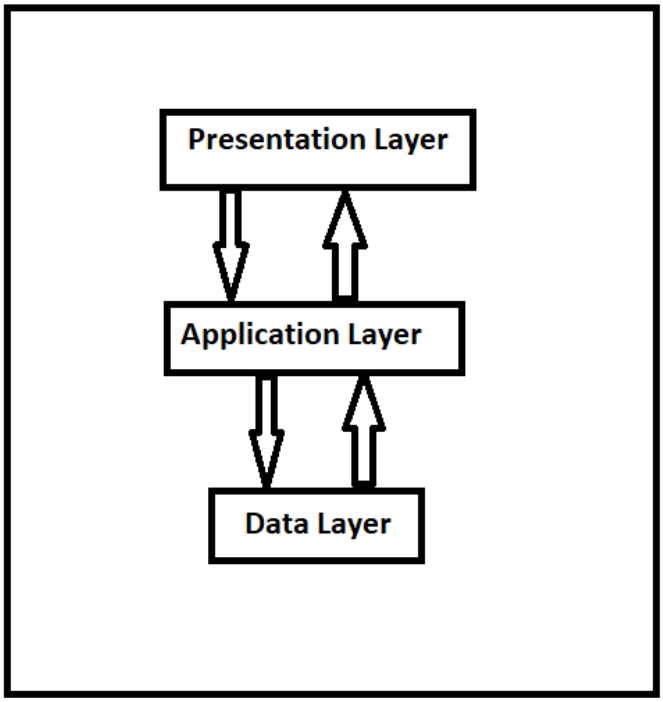
Also it is a tool that automates the deployment, Management, Scaling, Networking and availability of container based applications.

In other words you can cluster together groups of hosts running linux containers and kubernetes helps you easily and efficiently manage those clusters. Kubernetes clusters can span across on-premise, Public Cloud, Private or Hybrid Clouds. For this reason Kubernetes is an ideal platform to host cloud-native applications that required rapid scaling etc.

- Kubernetes was developed by google labs and later donated to CNCF ( Cloud native computing Foundation.
- Kubernetes is the greek word for captian of ship.
- Kubernetes is also refered as K8'S as there are 8 characters between K & S.

In Real world Applications are develop using Monolithic Architecture & Microservices Architecture.

**Monolithic Architecture :**



It Consists of 3 Layers, SO its called as 3 tier Architecture. A monolithic architecture is the traditional unified model for the design of a software program. Monolithic, in this context, means composed all in one piece, Which means all these layers are packaged & deployed together as single unit and that's the reason its called as Monolithic Architecture.

**Advantages :**

1)Easy to Develop, Test and Deploy
2)Greater Compatability

**Disadvantages :**

1)Very Large ( If one component has issues it will impact everything )
2)Locked in with initial decisions and Technology.
3)Implemented using single development stack ( We have to use the same Technology to develop all the other components )
4)Frequent deployments are not possible
5)Need to scale an entire application stack

**What are microservices?**

Microservices (or microservices architecture) are a cloud native architectural approach in which a single application is composed of many loosely coupled and independently deployable smaller components, or services.

These services typically have their own technology stack, inclusive of the database and data management model; communicate with one another over a combination of REST APIs, event streaming, and message brokers; and are organized by business capability, with the line separating services often referred to as a bounded context.

While much of the discussion about microservices has revolved around architectural definitions and characteristics, their value can be more commonly understood through fairly simple business and organizational benefits:

Code can be updated more easily - new features or functionality can be added without touching the entire application
Teams can use different stacks and different programming languages for different components.
Components can be scaled independently of one another, reducing the waste and cost associated with having to scale entire applications because a single feature might be facing too much load.
Microservices might also be understood by what they are not. The two comparisons drawn most frequently with microservices architecture are monolithic architecture and service-oriented architecture (SOA).

The difference between microservices and monolithic architecture is that microservices compose a single application from many smaller, loosely coupled services as opposed to the monolithic approach of a large, tightly coupled application.

We can run these microservices on Physical Machines | Virtual Machines | Containers. It is always recommended to host the containers on Virtual Environment. Containers virtualize at the O/S Level where as Virtual Machines virtualize at the hard ware level. So, If we host our containers on Virtual environment we get the best of both worlds like virtualizing at the hardware level and O/S Level.

**What is container orchestration?**

Container orchestration automates the deployment, management, scaling, and networking of containers. Enterprises that need to deploy and manage hundreds or thousands of Linux® containers and hosts can benefit from container orchestration.

Container orchestration can be used in any environment where you use containers. It can help you to deploy the same application across different environments without needing to redesign it. And microservices in containers make it easier to orchestrate services, including storage, networking, and security.

Containers give your microservice-based apps an ideal application deployment unit and self-contained execution environment. They make it possible to run multiple parts of an app independently in microservices, on the same hardware, with much greater control over individual pieces and life cycles.

Managing the lifecycle of containers with orchestration also supports DevOps teams who integrate it into CI/CD workflows. Along with application programming interfaces (APIs) and DevOps teams, containerized microservices are the foundation for cloud-native applications.

**What are microservices?**

Microservices (or microservices architecture) are a cloud native architectural approach in which a single application is composed of many loosely coupled and independently deployable smaller components, or services.

These services typically have their own technology stack, inclusive of the database and data management model; communicate with one another over a combination of REST APIs, event streaming, and message brokers; and are organized by business capability, with the line separating services often referred to as a bounded context.

While much of the discussion about microservices has revolved around architectural definitions and characteristics, their value can be more commonly understood through fairly simple business and organizational benefits:

Code can be updated more easily - new features or functionality can be added without touching the entire application
Teams can use different stacks and different programming languages for different components.
Components can be scaled independently of one another, reducing the waste and cost associated with having to scale entire applications because a single feature might be facing too much load.
Microservices might also be understood by what they are not. The two comparisons drawn most frequently with microservices architecture are monolithic architecture and service-oriented architecture (SOA).

The difference between microservices and monolithic architecture is that microservices compose a single application from many smaller, loosely coupled services as opposed to the monolithic approach of a large, tightly coupled application.

We can run these microservices on Physical Machines | Virtual Machines | Containers. It is always recommended to host the containers on Virtual Environment. Containers virtualize at the O/S Level where as Virtual Machines virtualize at the hard ware level. So, If we host our containers on Virtual environment we get the best of both worlds like virtualizing at the hardware level and O/S Level.

**What is container orchestration?**

Container orchestration automates the deployment, management, scaling, and networking of containers. Enterprises that need to deploy and manage hundreds or thousands of Linux® containers and hosts can benefit from container orchestration.

Container orchestration can be used in any environment where you use containers. It can help you to deploy the same application across different environments without needing to redesign it. And microservices in containers make it easier to orchestrate services, including storage, networking, and security.

Containers give your microservice-based apps an ideal application deployment unit and self-contained execution environment. They make it possible to run multiple parts of an app independently in microservices, on the same hardware, with much greater control over individual pieces and life cycles.

Managing the lifecycle of containers with orchestration also supports DevOps teams who integrate it into CI/CD workflows. Along with application programming

interfaces (APIs) and DevOps teams, containerized microservices are the foundation for cloud-native applications.

**What is container orchestration used for?**

Use container orchestration to automate and manage tasks such as:

- Provisioning and deployment
- Configuration and scheduling
- Resource allocation
- Container availability
- Scaling or removing containers based on balancing workloads across your infrastructure
- Load balancing and traffic routing
- Monitoring container health
- Configuring applications based on the container in which they will run
- Keeping interactions between containers secure

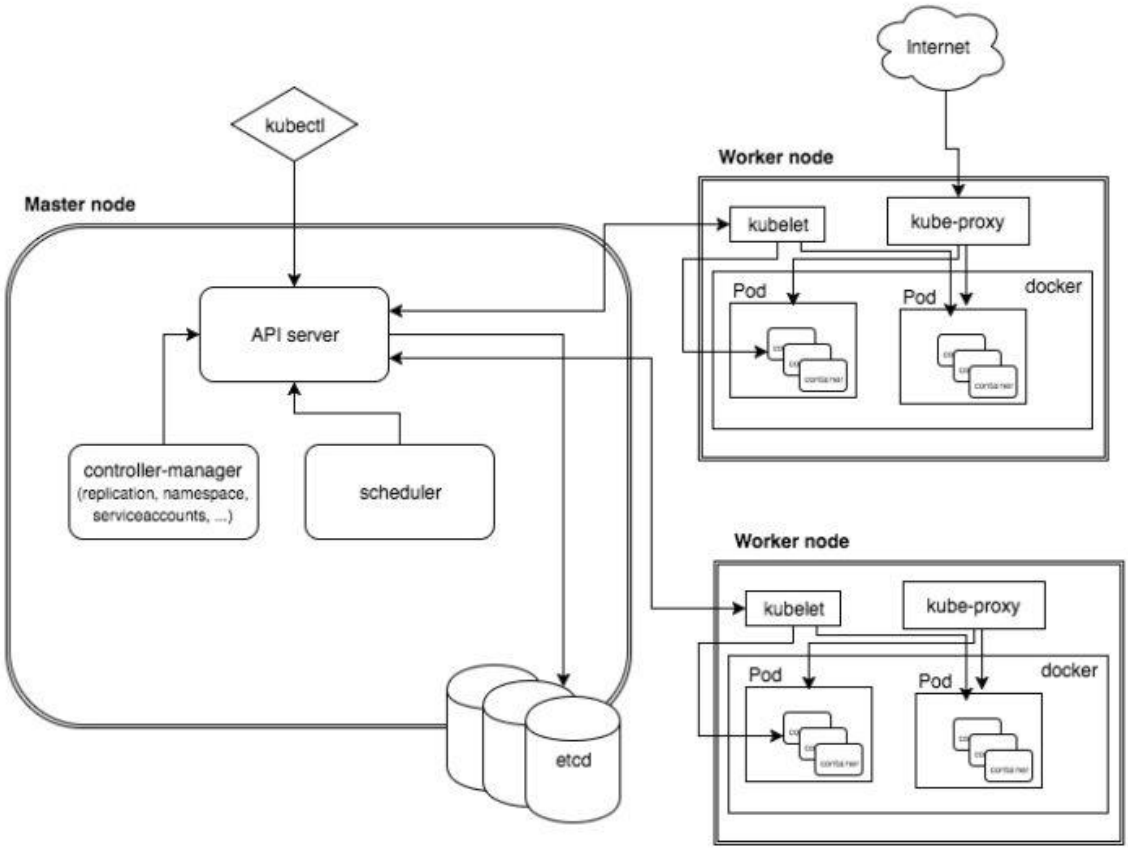
**Container orchestration tools :**

Container orchestration tools provide a framework for managing containers and microservices architecture at scale. There are many container orchestration tools that can be used for container lifecycle management. Some popular options are Kubernetes, Docker Swarm, and Apache Mesos.

Kubernetes is an open source container orchestration tool that was originally developed and designed by engineers at Google. Google donated the Kubernetes project to the newly formed Cloud Native Computing Foundation in 2015.

## Kubernetes Components and Architecture :

Kubernetes follows a client-server architecture. It's possible to have a multi-master setup (for high availability), but by default there is a single master server which acts as a controlling node and point of contact. The master server consists of various components including a kube-apiserver, an etcd storage, a kube-controller-manager, a kube-scheduler, and a DNS server for Kubernetes services. Worker Node components include kubelet and kube-proxy on top of Docker.




**Master Node Components :**

Below are the main components found on the master node:

**kube-apiserver** – Kubernetes API server is the central management entity that receives all REST requests for modifications (to pods, services, replication sets/controllers and others), serving as frontend to the cluster. Also, this is the only component that communicates with the etcd cluster, making sure data is stored in etcd and is in agreement with the service details of the deployed pods.

**kube-controller-manager**

Control Plane component that runs controller processes.
Logically, each controller is a separate process, but to reduce complexity, they are all compiled into a single binary and run in a single process.

Some types of these controllers are:

**Node controller :**

Responsible for noticing and responding when nodes go down.

**Job controller | Replication Controller :**

Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.

**Endpoints controller :**

Populates the Endpoints object (that is, joins Services & Pods).

**Service Account & Token controllers :**

Create default accounts and API access tokens for new namespaces

**kube-scheduler** – helps schedule the pods (a co-located group of containers inside which our application processes are running) on the various nodes based on resource utilization. It reads the service's operational requirements and schedules it on the best fit node. For example, if the application needs 1GB of memory and 2 CPU cores, then the pods for that application will be scheduled on a node with at least those resources. The scheduler runs each time there is a need to schedule pods. The scheduler must know the total resources available as well as resources allocated to existing workloads on each node.

**etcd cluster** – a simple, distributed key value storage which is used to store the Kubernetes cluster data (such as number of pods, their state, namespace, etc), API objects and service discovery details. It is only accessible from the API server for security reasons. etcd enables notifications to the cluster about configuration changes with the help of watchers. Notifications are API requests on each etcd cluster node to trigger the update of information in the node's storage.

**Considerations for large clusters :**

A cluster is a set of nodes (physical or virtual machines) running Kubernetes agents, managed by the control plane. Kubernetes v1.27 supports clusters with up to 5000 nodes. More specifically, Kubernetes is designed to accommodate configurations that meet all of the following criteria:

No more than 110 pods per node
No more than 5000 nodes
No more than 150000 total pods

No more than 300000 total containers
You can scale your cluster by adding or removing nodes. The way you do this depends on how your cluster is deployed

**What is a Pod?**

Pods are the smallest, most basic deployable objects in Kubernetes. A Pod represents a single instance of a running process in your cluster.

Pods contain one or more containers, such as Docker containers. When a Pod runs multiple containers, the containers are managed as a single entity and share the Pod's resources. Generally, running multiple containers in a single Pod is an advanced use case.

Pods also contain shared networking and storage resources for their containers.

**What is Minikube?**

Minikube is a utility you can use to run Kubernetes (k8s) on your local machine. It creates a single node cluster contained in a virtual machine (VM). This cluster lets you demo Kubernetes operations without requiring the time and resource-consuming installation of full-blown K8s.

All you need is Docker (or similarly compatible) container or a Virtual Machine environment, and Kubernetes is a single command away: minikube start.

**Recommended Min Hardware Config Required :**
2 CPUs or more
2GB of free memory
20GB of free disk space
Internet connection
Container or virtual machine manager, such as: **Docker, Hyperkit, Hyper-V, KVM, Parallels, Podman, VirtualBox, or VMWare**

**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. For more information including a complete list of kubectl operations, see the kubectl reference documentation.

**kubectl is installable on a variety of Linux platforms, macOS and Windows**.

**Steps to Config Kubernetes Cluster using MINIKUBE :**

1)Install Virtual Box

2)Install KUBECTL

3)Install MINIKUBE

4)Start MINIKUBE as Non Root User

5) Ensure these all we are doing on Base Operating system ( **Note on Guest OS** )

**Installing Virtual Box on Base OS :**

#yum install wget vim –y ( **Installing wget & Vim softwares** )
#wget https://download.virtualbox.org/virtualbox/rpm/rhel/virtualbox.repo -P /etc/yum.repos.d/ **(Configuring Virtual Box Repository )**

#yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm ( **Configuring EPEL Repository** )

#yum update

#yum install binutils kernel-devel kernel-headers libgomp make patch gcc glibc-headers glibc-devel dkms -y

#Yum install VirtualBox-6.1 ( **Installing Virtual Box** )
#which virtualbox
#reboot
#/sbin/vboxconfig ( **Stopping & STarting the Virtual Box Services** )
#systemctl status vboxconfig ( **Verify Virtual box Service is Runnin** )

**Installing KUBECTL :**

#curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl

#chmod +x ./kubectl

#mv ./kubectl /usr/local/bin/kubectl **(Move the binary in to your PATH )**

**Best to ensure the version you installed is up-to-date:**
#kubectl version --client

**Installing MINIKUBE :**
#curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && chmod +x minikube
#mkdir -p /usr/local/bin/
#Install minikube /usr/local/bin/
$minikube version
$minikube status
**Confirm Installation : (From here you need to do with non root user)**
$minikube start --driver=<driver_name>
virtualbox | vmware ( As per your Hypervisor )

**NOTE :**
1)By default Minikube used 2GB of Ram, 2 V CPU & 20GB of Disk space by default. It is possible to create MiniKube VM with our custom required Hardware configurations
$minikube --memory 8192 --cpus 6 start

**After Installing Test it :**
$minikube status **( Displays the status of MiniKube )**
$kubectl config view **( Displays Kubernetes Cluster configuration Information )**
$kubectl get no **( Displays the nodes available in the Cluster along with its state )**
$kubectl get po **( Displaying the pods Created )**
$kubectl delete po <POD NAME> **( Deleting the Specific POD )**
$minikube stop **( Stopping the Minikube VM )**
$minikube delete **( Deleting the MiniKube VM )**

$minikube status ( **Verify the VM is Deleted or Not** )

## Setting up Kubernetes Cluster using "KUBEADM" – Version 1.25

Docker as an underlying runtime is being deprecated in favor of runtimes that use the Container Runtime Interface (CRI) created for Kubernetes. Docker-produced images will continue to work in your cluster with all runtimes, as they always have.

If you're an end-user of Kubernetes, not a whole lot will be changing for you. This doesn't mean the death of Docker, and it doesn't mean you can't, or shouldn't, use Docker as a development tool anymore. Docker is still a useful tool for building containers, and the images that result from running docker build can still run in your Kubernetes cluster.

If you're using a managed Kubernetes service like AKS, EkS or GKE, you will need to make sure your worker nodes are using a supported container runtime before Docker support is removed in a future version of Kubernetes. If you have node customizations you may need to update them based on your environment and runtime requirements. Please work with your service provider to ensure proper upgrade testing and planning.

If you're rolling your own clusters, you will also need to make changes to avoid your clusters breaking. At v1.20, you will get a deprecation warning for Docker. When Docker runtime support is removed in a future release (currently planned for the 1.22 release in late 2021) of Kubernetes it will no longer be supported and you will need to switch to one of the other compliant container runtimes, like containerd or CRI-O.

Why CRI-O ?
CRI-O is an implementation of the Kubernetes CRI (Container Runtime Interface) to enable using OCI (Open Container Initiative) compatible runtimes.

It is a lightweight alternative to using Docker, Moby or rkt as the runtime for Kubernetes.

CRI-O is a CRI runtime mainly developed by Red Hat folks. In fact, this runtime is used in Red Hat OpenShift now. Yes, they do not depend on Docker anymore.

Interestingly, RHEL 7 does not officially support Docker either. Instead, they provide Podman, Buildah and CRI-O for container environment.

Why CRI-O ?

CRI-O is an implementation of the Kubernetes CRI (Container Runtime Interface) to enable using OCI (Open Container Initiative) compatible runtimes.

It is a lightweight alternative to using Docker, Moby or rkt as the runtime for Kubernetes.

CRI-O is a CRI runtime mainly developed by Red Hat folks. In fact, this runtime is used in Red Hat OpenShift now. Yes, they do not depend on Docker anymore.

Interestingly, RHEL 7 does not officially support Docker either. Instead, they provide Podman, Buildah and CRI-O for container environment.

## Steps to Install Kubernetes CLuster using KUBEADM

we will have 1 Master node and 3 Worker Nodes ( Total Nodes : 4 )

**On all Nodes :**

1)Ensure to have 4 VM's installed with Centos 7 OS.

2)Recommended to have all virtual machines with min 2 CPU & more than 2GB of RAM

3)Ensure swap, SELinux, firewall must be disabled on all the nodes which are a part of k8's cluster .

4)Ensure to install CRI-O runtime on all the nodes which are part of k8s cluster.

5)Ensure to install KUBEADM, KUBECTL, KUBELET on all the nodes which are a part of K8's cluster.

6)Ensure CRI-O & Kubelet service must be started on all the nodes which are part of K8's cluster.

8)Need to configure IP forwarding on all the nodes which are part of K8's cluster.

**Need to be done on Only Master node :**

9)Select any one node out of 4, Initialize as Master node

10)We need to configure the POD networking.

**Only on Worker Nodes :**

10)We need to configure the remaining 3 as worker nodes by joining them to the cluster

## ON ALL Nodes (Manager & Worker Nodes )

**Disabling SWAP :**
#swapoff -a
#swapon -s
**Note : Go to /etc/fstab and add a comment to swap partition for permanently disabling**

**Disable SE Linux :**
#sestatus
#getenforce 0 ( **DIsabling Selinux** )

**To permanently disable SE Linux :**
#vi /etc/selinux/config
SELinux=disabled
:wq!

**Disabling Firewall :**
#systemctl disable firewalld
#systemctl stop firewalld
#reboot

**NOTE : If you don't disable SWAP, SELINUX & Firewall it leads to KUBELET Service failure.**

**Create a enviroment variable for OS.**
#export OS=CentOS_7
**Note: If you are using CentOS 8 then give variable name as CentOS_8**
#export VERSION=1.17

**NOTE : To make it permanent update the variables in /etc/environment , .profile and .bashrc files in the root users home directory.**

**Configuring REPO for cri-o :**

#yum install curl -y

curl -L -o /etc/yum.repos.d/devel:kubic:libcontainers:stable.repo https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/devel:kubic:libcontainers:stable.repo

curl -L -o /etc/yum.repos.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.repo https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:$VERSION/$OS/devel:kubic:libcontainers:stable:cri-o:$VERSION.repo

#cd /etc/yum.repos.d
#ls

#cd

**Installing CRI-O Runtime :**
#yum install cri-o -y

**Start and enable the cri-o service :**

#systemctl start cri-o
#systemctl enable cri-o
#systemctl status cri-o

**Enable Kernel Modules which are prerequisites for Kubernetes:**

#modprobe overlay
#modprobe br_netfilter

**To make it permanent :**
#cat /etc/modules-load.d/k8s.conf
overlay
br_netfilter
:wq!

**Configuring IP forwarding :**

#vi /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
:wq!

**# Apply sysctl params without reboot#**
#sysctl --system

**Setting up Kubernetes Repo :**

#vi /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
enabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
:wq!

#yum install kubeadm-1.25.0 kubectl-1.25.0  kubelet-1.25.0 cri-tools-1.25.0  -y
#systemctl start kubelet
#systemctl enable kubelet

**Only on the node you choose to config as Manager :**

#kubeadm init **( Initializes this node as Manager node and Cluster will be setup)**

**###To start using your cluster, you need to run the following as a regular user##**

#mkdir -p $HOME/.kube
#cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
#chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

#export KUBECONFIG=/etc/kubernetes/admin.conf

**Configuring POD Network :**

We need to install CNI plugin. So that pods can communicate with each other.
There are different CNI plugins for different purpose. Here we are using Weave Net CNI plugin.

kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-daemonset-k8s.yaml

#kubectl get nodes ( **Lists all the available nodes in cluster** )

#kubectl get pods --all-namespaces ( **Verify the POD network we just deployed is RUNNING or not** )

**Setting up Kubernetes Worker Node ( Only on Worker Nodes ) :**

Disabling SWAP :
#swapoff -a
#swapon -s
Note : Go to /etc/fstab and add a comment to swap partition for permanently disabling

**Disable SE Linux :**
#sestatus
#getenforce o ( **DIsabling Selinux** )
**To permanently disable SE Linux :**
#vi /etc/selinux/config
SELinux=disabled
:wq!

**Disabling Firewall :**
#systemctl disable firewalld
#systemctl stop firewalld
#reboot

**NOTE : If you don't disable SWAP, SELINUX & Firewall it leads to KUBELET Service failure.**

**Create a enviroment variable for OS.**
#export OS=CentOS_7
**Note: If you are using CentOS 8 then give variable name as CentOS_8**
#export VERSION=1.17

**NOTE : To make it permanent update the variables in /etc/environment , .profile and .bashrc files in the root users home directory.**

**Configuring REPO for cri-o :**

#yum install curl -y

curl -L -o /etc/yum.repos.d/devel:kubic:libcontainers:stable.repo
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/devel:kubic:libcontainers:stable.repo

curl -L -o /etc/yum.repos.d/devel:kubic:libcontainers:stable:cri-o:$VERSION.repo
https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:$VERSION/$OS/devel:kubic:libcontainers:stable:cri-o:$VERSION.repo

#cd /etc/yum.repos.d
#ls
#cd
**Installing CRI-O RUntime :**
#yum install cri-o -y

**Start and enable the cri-o service :**
#systemctl start cri-o
#systemctl enable cri-o
#systemctl status cri-o

**Enable Kernel Modules which are prerequesites for Kubernetes:**
#modprobe overlay
#modprobe br_netfilter

**To make it permanent :**
#cat /etc/modules-load.d/k8s.conf
overlay
br_netfilter
:wq!

**Configuring IP forwarding :**

#vi /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
:wq!

**# Apply sysctl params without reboot**
#sysctl --system

**Setting up Kubernetes Repo :**

```
#vi /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=0
repo_gpgcheck=0
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
:wq!

#yum install kubeadm-1.25.0 kubectl-1.25.0  kubelet-1.25.0 cri-tools-1.25.0  -y
#systemctl start kubelet
#systemctl enable kubelet
```

**Now this node is ready to join the cluster as Worker Node**

We need to copy the token from the manager node while we got from cluster initialization time, using the same token we can join this node to cluster

```
#kubeadm join 192.168.0.152:6443 --token 1we89d.de34tr34098de345 \
   --discovery-token-ca-cert-hash sha256:d5f47rr6vb4h6y+6fc1g6y7fr14dd879  ( This is just for reference, You have to use the token you have got on your manager node )
```

**Go to Manager Node :**

Verify the Worker node has joined cluster or not

```
#kubectl get nodes
```