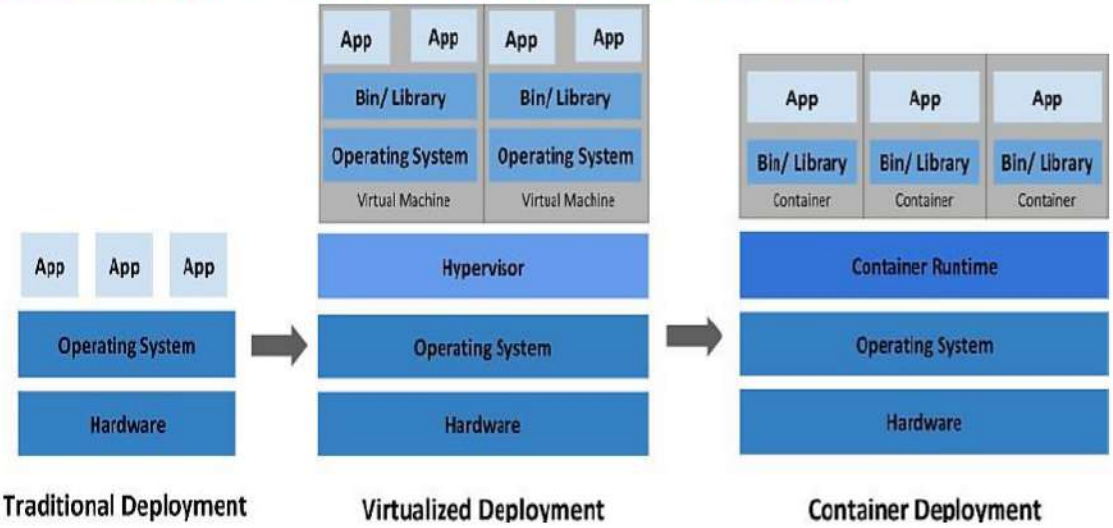
MAIL: Rnraju4u@gmail.com NUMBER: +91 9848363431 YOUTUBE: SYSGEEKS

KUBERNETES

- Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitatesboth declarative configuration and automation.
- Kubernetes also known as k8s or "kube". Kubernetes clusters can span hosts across on-premise, public,private, or hybrid clouds.

TRADITIONAL vs VIRTUALIZATION CONTAINERS:



TRADITIONAL DEPLOYMENT:

- Early on, organizations ran applications on physical servers.
- There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues.
- A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers.

VIRTUALIZED DEPLOYMENT:

- It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU.
- Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.
- Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more.

- Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

CONTAINER DEPLOYMENT:

- Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM, a container has its own filesystem, share of CPU, memory, process space, and more.
- As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

CONTAINERS HAVE BECOME POPULAR BECAUSE THEY PROVIDE EXTRA BENEFITS, SUCH AS:

- Agile application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.
- Continuous development, integration, and deployment: provides for reliable and
- frequent container image build and deployment with quick and efficient rollbacks (due to image immutability).
- Dev and Ops separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
- Observability: not only surfaces OS level information and metrics, but also application health and other signals.
- Environmental consistency across development, testing, and production: Runs the same on a laptop as it does in the cloud.
- Cloud and OS distribution portability: Runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.
- Application-centric management: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- Loosely coupled, distributed, elastic, liberated micro-services: applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
- **Resource isolation:** predictable application performance.
- **Resource utilization:** high efficiency and density.

KUBERNETES FEATURES:

- Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Kubernetes Provides:

SERVICE DISCOVERY AND LOAD BALANCING:

Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.

STORAGE ORCHESTRATION:

Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.

AUTOMATED ROLLOUTS AND ROLLBACKS:

You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.

AUTOMATIC BIN PACKING:

You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.

SELF-HEALING:

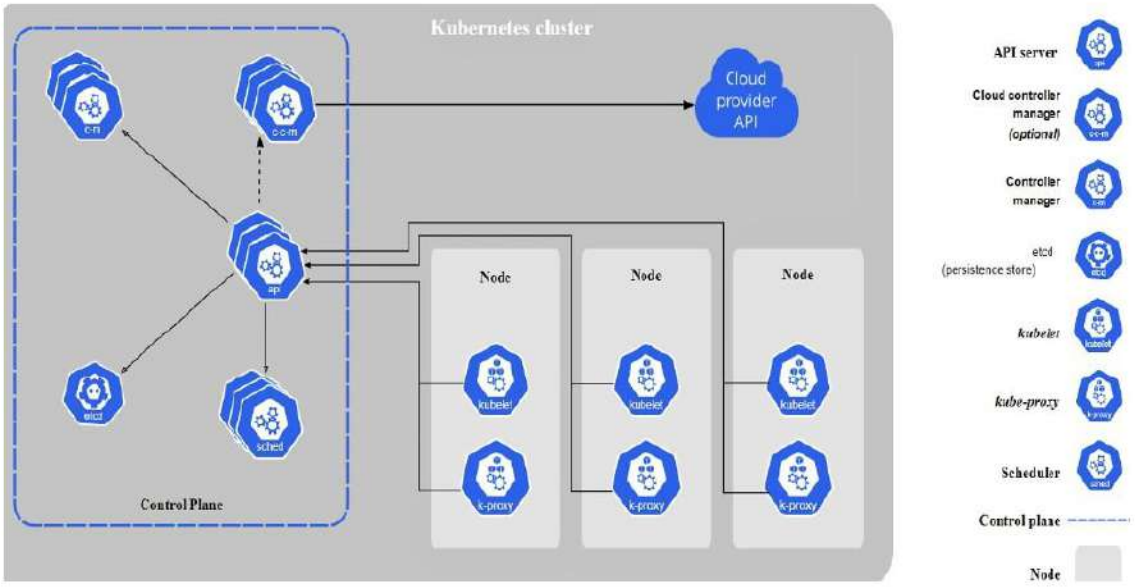
Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.

SECRET AND CONFIGURATION MANAGEMENT:

Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

KUBERNETES ARCHITECTURE & COMPONENTS

- A Kubernetes cluster consists of the components that represent the control plane and a set of machines called nodes.
- A **Kubernetes cluster** consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.
- The **worker node(s)** host the Pods that are the components of the application workload. The control plane manages the worker nodes and the Pods in the cluster. In production environments, the control plane usually runs across multiple computers and a cluster usually runs multiple nodes, providing fault-tolerance and high availability.



KUBERNETES CLUSTER:

- A Kubernetes cluster consists of a set of worker machines, callednodes, that run containerized applications. Every cluster has at least one worker node.
- The Worker Node(s) host the Pods that are the components of the application workload.
- The Control Plane manages the worker nodes and the Pods in thecluster.
- Cluster usually runs multiple nodes, providing Fault-Tolerance and High Availability.

CONTROL PLANE COMPONENTS:

KUBE-APISERVER:

- Kube-APIserver is the main management point of the entire cluster, handling internal and external requests.
- It processes REST operations, validates, and updates thecorresponding objects in etcd.

ETCD:

- Consistent and highly-available key value store used as Kubernetes'backing store for all cluster data.

KUBE-SCHEDULER:

- It considers the resource needs of a pod, such as CPU or memory,along with the health of the cluster.
- It schedules the pod to an appropriate compute node.

KUBE-CONTROLLER-MANAGER:

- Kube-Control-Manager runs the control process.
- A control process is a loop that focuses on making the desired stateequal to the current state for any application in any given instance oftime.

CLOUD-CONTROLLER-MANAGER:

- The cloud controller manager links your cluster into your cloud provider's API, and separates out the components that interact withthat cloud platform from components that only interact with your cluster.
- The following controllers can have cloud provider dependencies:

  - **Node controller:**
    For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
  - **Route controller:**
    For setting up routes in the underlying cloud infrastructure
  - **Service controller:**

    For creating, updating and deleting cloud provider load balancers

NODE COMPONENTS:

KUBELET:

- An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.
- When the control plane needs something to happen in a node, thekubelet executes the action.

KUBE-PROXY:

- Kube-proxy is a network proxy that runs on each node in your cluster.
- It maintains network rules on nodes. These network rules allow network communication to your Pods of your cluster.

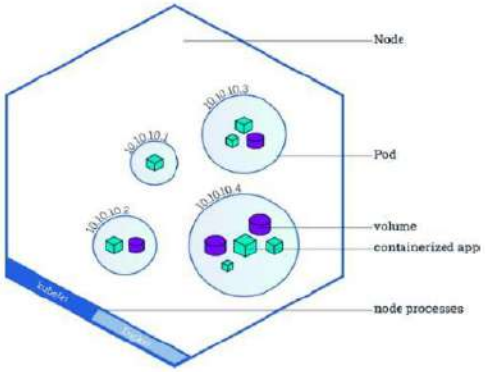
CONTAINER RUNTIME:

- The container runtime is the software that is responsible for running containers.
- Kubernetes supports: Docker, containerd, CRI-O, and KubernetesCRI (Container Runtime Interface).

KUBERNETES NODES & PODS:

NODES:

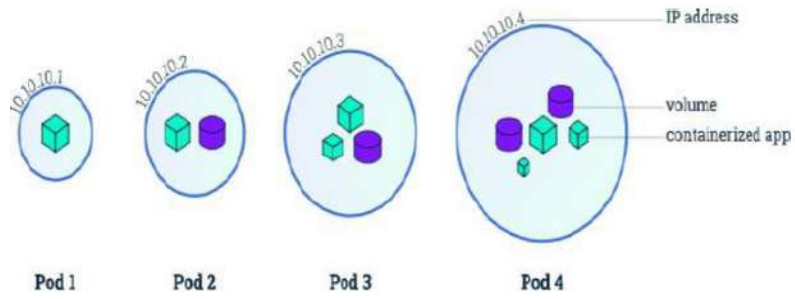
- A node is a worker machine in Kubernetes and may be a VM or physical machine, depending on the cluster.
- Kubernetes runs your workload by placing containers into Pods to runon Nodes. A node can have multiple Pods.
- A node includes the kubelet, a container runtime, and kube-proxy.
- Node names are uniqueness in the cluster.



PODS:

- A pod is the smallest & simplest Kubernetes object. It represents asingle 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 asa single entity and share the Pod's resources.
- Pods also contain shared networking and storage resources fortheir containers

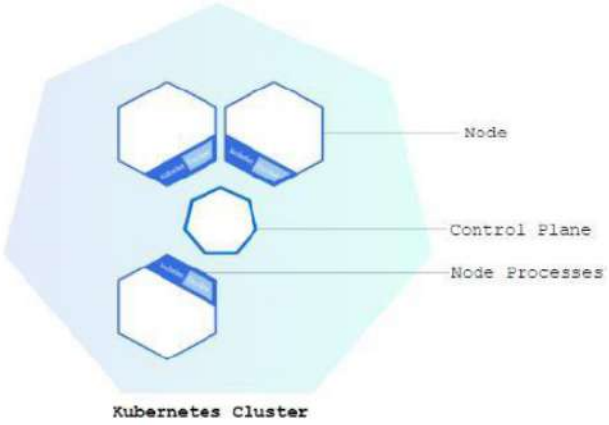
NOTE: Running multiple containers in a single Pod is an advanced usecase.



MINIKUBE

- Minikube is a utility you can use to run Kubernetes (k8s) on yourlocal machine.
- It is a lightweight Kubernetes implementation that creates a VM on your local machine and deploys a simple cluster containing only onenode.
- Minikube CLI provides basic bootstrapping operations for yourcluster, including start, stop, status, and delete.
- Minikube is available for Linux, macOS, and Windows systems.

MINIKUBE ARCHITECTURE:



CONTROL PLANE:

- It is responsible for managing the cluster. The control plane coordinates all activities in your cluster, such as scheduling applications, maintaining applications' desired state, scaling applications, and rolling out new updates.

NODE:

- Node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.
- Each node has a Kubelet and container run time for managing the node and handling container application

MINIKUBE INSTALLATION

PRE-REQUISITES:

- 2 CPUs or more
- 2GB of free memory
- 20GB of free disk space
- Good Internet connection
- Container or virtual machine manager (Docker, Hyper-V, KVM, VirtualBox, orVMware...etc.)

STEP 1: Setting Up Hostname:

#hostname Minikube

#vim /etc/hostname

Minikube

#bash

STEP 2: Download & Install Minikube:

#curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

#install minikube-linux-amd64 /usr/local/bin/minikube

#minikube start

STEP 3: Interact with your cluster:

#kubectl get po -A

#minikube kubectl -- get po -A

#alias kubectl="minikube kubectl --"

#minikube dashboard

STEP 4: Deploy applications:

#kubectl create deployment hello-minikube image
image=k8s.gcr.io/echoserver:1.4

#kubectl expose deployment hello-minikube --type=NodePort --port=8080
#kubectl get services hello-minikube

#minikube service hello-minikube

#kubectl port-forward service/hello-minikube 7080:8080

Application is now available at http://localhost:7080

STEP 5: Manage your cluster:

Pause Kubernetes without impacting deployed applications:

#minikube pause

Unpause a paused instance:

#minikube unpause

Halt the cluster:

#minikube stop

Change the default memory limit (requires a restart):

#minikube config set memory 9001

Browse the catalog of easily installed Kubernetes services:

#minikube addons list

Create a second cluster running an older Kubernetes release:

#minikube start -p aged --kubernetes-version=v1.16.1

Delete all of the minikube clusters:

#minikube delete --all

KUBEADM INSTALLATION

- Kubeadm is a tool built to provide kubeadm init and kubeadm join as best-practice "fast paths" for creating Kubernetes clusters.
- kubeadm performs the actions necessary to get a minimum viable cluster up and running.

KUBEADM-PREREQUISITES:

- A compatible Linux host. (Debian and Red Hat)
- 2 GB or more of RAM per machine
- 2 CPUs or more.
- Full network connectivity between all machines in thecluster
- Unique hostname, MAC address, and product_uuid forevery node.
- Certain ports are open on your machines.
- MUST disable swap

LAB-SETUP:

- **Master** Hostname: Master (10.10.10.100) 2GB Ram, 2vcpus
- **Worker1** Hostname: Node1 (10.10.10.101) 1GB Ram, 1vcpus
- **Worker2** Hostname: Node2 (10.10.10.102) 1GB Ram, 1vcpus

VM'S-NETWORK-SETUP:

STEP 1: Seting up Hostname on each vm based on LAB-SETUP

#hostname Master
#vim /etc/hostname
Master
#vim /etc/hosts
Master-IP Master
Node1-IP Node1
Node2-IP Node2
#bash

#hostname Node1 **[FROM Node1]**
#vim /etc/hostname
Node1
#vim /etc/hosts
Node1-IP Node1
Master-IP Master

#hostname Node2 **[FROM Node2]**
#vim /etc/hostname
Node2
#vim /etc/hosts
Node2-IP Node2
Master-IP Master
#bash

STEP 2: Turnoff swap on all Master & Worker Nodes:
#swapoff -a
#vim /etc/fstb
Comment a swap file system (#)
#systemctl daemon-reload

STEP 3: Disable SE-Linux firewalls on all master & Worker Nodes:
#setenforce 0
#vim /etc/selinux/config
SELINUX=disabled
#reboot

INSTALLING A CONTAINER RUN TIME ON ALL:

To run containers in Pods, Kubernetes uses a container runtime.

STEP 4: Set up the repository

#yum install -y yum-utils

#yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo

STEP 5: Install Docker Engine

#yum install docker-ce -y

NOTE: Getting any error, plese change repo lines

#vim /etc/yum.repos.d/docker-ce.repo

```
[docker-ce-stable]
name=Docker CE Stable - $basearch
baseurl=https://download.docker.com/linux/centos/$releasever/$basearch/stable
enabled=1
gpgcheck=1
gpgkey=https://download.docker.com/linux/centos/gpg
```

#yum install docker-ce -y

#docker --version

STEP 6: Start and Enable docker service

#systemctl start docker

#systemctl enable docker

#systemctl status docker

INSTALLING KUBEADM, KUBELET AND KUBECTL:

- You will install these packages on all of your machines:

kubeadm: The command to bootstrap the cluster.
kubelet: The component that runs on all of the machines in your cluster and does things like starting pods and containers.
kubectl: The command line util to talk to your cluster.

STEP 7: Setup a repository:

```
cat <<EOF | sudo tee /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
exclude=kubelet kubeadm kubectl
EOF
```

STEP 8: Installing and Enable Kubelet:

```
#yum install -y kubelet kubeadm kubectl --disableexcludes=Kubernetes
#systemctl start kubelet
#systemctl enable --now kubelet
```

CG GROUP DRIVERS:

- Both the container runtime and the kubelet have a property called "cgroup driver", which is important for the management of cgroups on Linux machines.

```
#mkdir /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
#systemctl enable docker
#systemctl daemon-reload
#systemctl restart docker
```

CREATING A CLUSTER WITH KUBEADM:

- **The kubeadm tool is good if you need:**
  - A simple way for you to try out Kubernetes, possibly for the first time.
  - A way for existing users to automate setting up a cluster and test their application.
  - A building block in other ecosystem and/or installer tools with a larger scope.

Before You Begin:

- One or more machines running a deb/rpm-compatible Linux OS; for example: Ubuntu or CentOS.
- 2 GiB or more of RAM per machine--any less leaves little room for your apps.
- At least 2 CPUs on the machine that you use as a control-plane node.
- Full network connectivity among all machines in the cluster. You can use either a public or a private network.

INITIALIZE KUBERNETES CLUSTER (FROM MASTER):

The control-plane node is the machine where the control plane components run, including etcd (the cluster database) and the API Server (which the kubectl command line tool communicates with).

To initialize the control-plane node run:

#kubeadm init --pod-network-cidr=10.10.0.0/16 --apiserver-advertise-address=10.10.10.100

To start using your cluster, you need to run the following as aregular user:

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

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

#KUBECONFIG=/etc/kubernetes/admin.conf

POD NETWORK ADD ON:

Installing pod network add on: [Flannel Network]
#kubectl apply -f
https://github.com/coreos/flannel/raw/master/Documentation/kube-flannel.yml [For Flannel network 10.10.0.0/16]

(or)

#kubectl --kubeconfig=/etc/kubernetes/admin.conf create -f
https://docs.projectcalico.org/v3.14/manifests/calico.yaml [Calico network]

#kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

[For Network 192.168.0.0/16]

#kubectl get pods --all-namespaces#kubectl get nodes

JOINING NODES:

The nodes are where your workloads (containers and Pods, etc) run.
To add new nodes to your cluster do the following for each machine:

You can join any number of worker nodes by running the following on each as root:

```
#kubeadm join 10.10.10.10:6443 --token tpj9f5.ikl2z77ufimmwos3 \
    --discovery-token-ca-cert-hash
    sha256:dd6c10b0bb9efa3062017926806e77173baf5b80ee1ee7486867ddc
```

If you do not have the token, you can get it by running the following command on the control-plane node:
#kubeadm token list

By default, tokens expire after 24 hours. If you are joining a node to the cluster after the current token has expired, you can create a new token by running the following command on the control-plane node:
#kubeadm token create

After Joining nodes from the Master:
#kubectl get nodes
#kubectl get pods --all-namespaces
#kubectl get nodes -o wide

TROUBLE SHOOTING:

Before removing the node, reset the state installed bykubeadm:
#kubeadm reset [From worker node]

Remove the Nodes [From Master]
#kubectl drain <node name> --delete-local-data --force --ignore-daemonsets

Now remove the node:
#kubectl delete node <node name>

To get a hash again to join nodes:
#kubeadm token create --print-join-command

CLEAN UP THE CONTROL PLANE:

Use **kubeadm reset** on the control plane host to trigger a best-effort clean up

#kubeadm reset