

## **ASSIGNMENT - 5**

**AIM:** To understand the Kubernetes Cluster Architecture, install and Spin Up a Kubernetes Cluster on Linux Machines/Cloud Platforms

**LO2.** To deploy single and multiple container applications and manage application deployments with rollouts in Kubernetes.

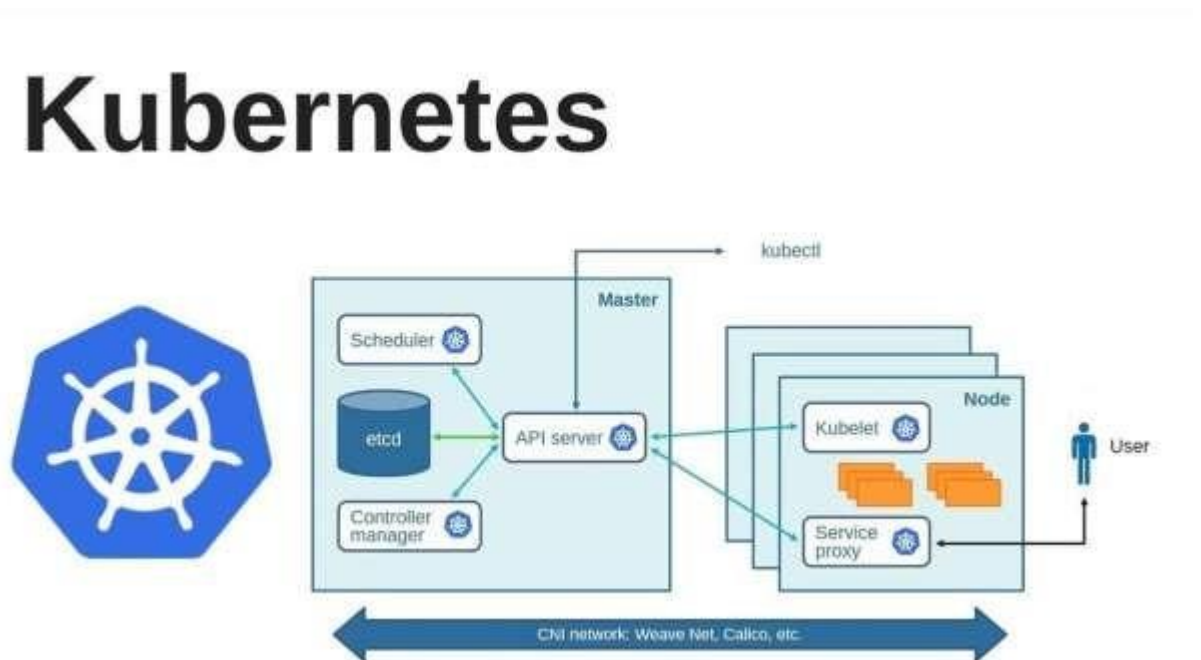
### **THEORY:**

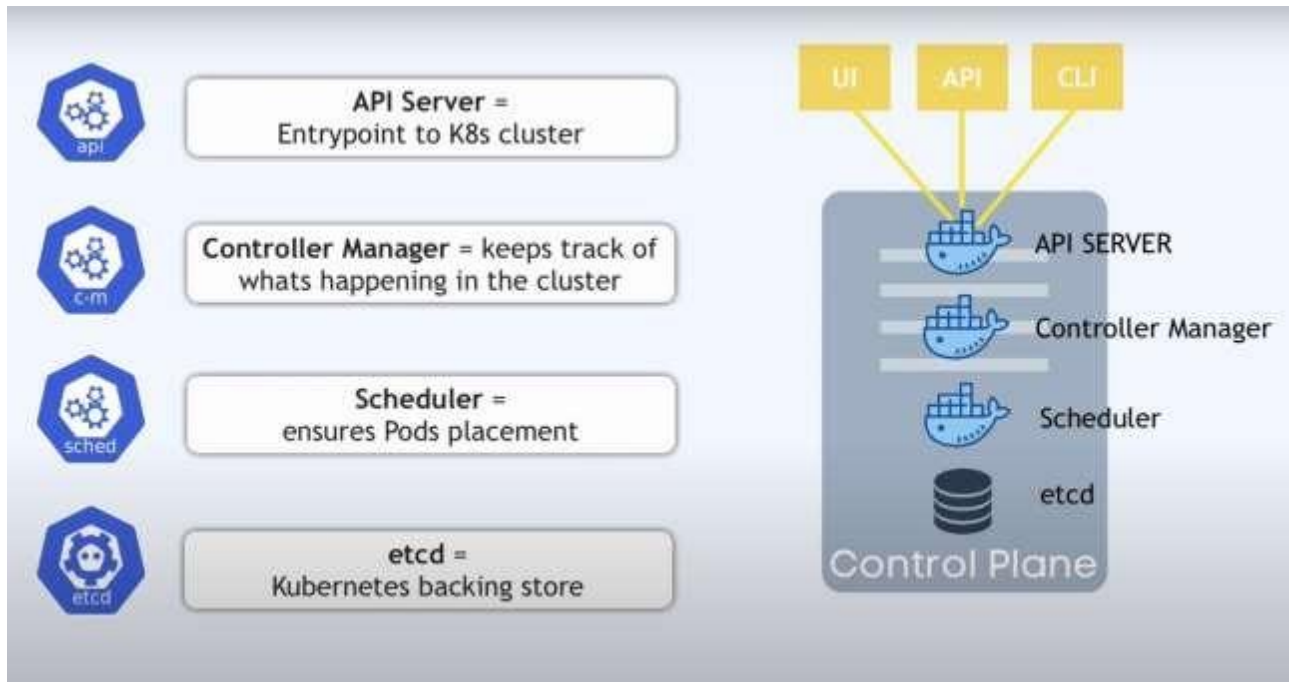
Kubernetes is an open-source container management tool that automates container deployment, scaling & load balancing.

It schedules, runs, and manages isolated containers that are running on virtual/physical/cloud machines.

All top cloud providers support Kubernetes. One popular name for Kubernetes is K8s.

### **ARCHITECTURE**





## Working with Kubernetes

- We create a Manifest (.yaml) file
- Apply those to cluster (to master) to bring it into the desired state.
- POD runs on a node, which is controlled by the master.

Q.1 What are the various Kubernetes services running on nodes? Describe the role of each service.

In a Kubernetes cluster, several essential services run on nodes to manage containerized applications and maintain the overall health and functionality of the system. These services play crucial roles in orchestrating and maintaining container workloads. Here, we'll describe some of the core Kubernetes services running on nodes and their roles:

### 1. Kubelet:

- The Kubelet is one of the most critical components in a Kubernetes node. It is responsible for ensuring that containers are running in a Pod (the smallest deployable unit in Kubernetes).
- Kubelet communicates with the Kubernetes master to receive Pod specifications, which it then works to realize by starting, stopping, and managing containers.

### 2. Container Runtime:

- Kubernetes is designed to be container runtime-agnostic, which means it can work with various container runtimes such as Docker, containerd, or CRI-O.
- The container runtime is responsible for pulling container images, creating containers, and managing their lifecycles.

### 3. Kube Proxy:

- Kube Proxy is responsible for managing network connectivity and ensuring that network rules (such as Service LoadBalancer and Ingress) are correctly implemented.
- It maintains network rules on each node and facilitates communication between Pods.

### 4. Node Status:

- The Node Status service is responsible for reporting the health of the node to the master. It provides information about the node's resources, capacity, and conditions.
- The master can make scheduling decisions based on this information, ensuring that Pods are placed on nodes with adequate resources.

### 5. cAdvisor (Container Advisor):

- cAdvisor is an open-source resource usage and performance analysis agent. It is integrated into the Kubelet and provides valuable information about resource consumption for each container.
- This information is used for monitoring and scaling decisions, helping to ensure optimal resource allocation.

### 6. Node Problem Detector:

- This service is responsible for detecting and reporting node-level problems, such as hardware or OS issues. It can detect issues like disk failures, memory problems, or kernel crashes.
- Node Problem Detector helps in identifying problematic nodes and allows for proactive maintenance.

### 7. Cluster DNS:

- Kubernetes clusters usually have a built-in DNS service to provide DNS resolution for Pods. This service ensures that Pods can communicate with each other using their names.
- It maintains DNS records for all active Services and Pods in the cluster.

### 8. Node-local DNS Cache:

- This service caches DNS queries on a node to improve DNS resolution performance. It reduces the load on the Cluster DNS service.

#### 9. Container Logging:

- While not a single service, container logging is a crucial part of node-level services. Containers often generate logs that need to be collected and centralized for monitoring, troubleshooting, and auditing purposes.

- Common solutions for container logging include Fluentd, Elasticsearch, and Kibana (EFK) or Fluentd, Logstash, and Kibana (ELK) stacks.

These are the fundamental services running on Kubernetes nodes. Each plays a distinct role in managing containers, networking, resource monitoring, and system health. Together, they ensure the reliable operation of containerized workloads in a Kubernetes cluster.

#### Q.2 What is Pod Disruption Budget (PDB)?

A Pod Disruption Budget (PDB) is a policy in Kubernetes that allows you to define constraints on how many Pods of a particular type (often referred to as a "workload") can be unavailable simultaneously during voluntary disruptions. These voluntary disruptions can be caused by actions like scaling down or draining nodes, which may result from maintenance tasks, upgrades, or other operational requirements.

The primary purpose of a Pod Disruption Budget is to ensure that critical or sensitive workloads are not disrupted beyond a specified limit, thus maintaining application stability and reliability. Here's a more detailed explanation of PDB:

#### Key Concepts:

1. Target Workload: A Pod Disruption Budget is associated with a specific set of Pods, typically defined by a label selector. These Pods make up the "target workload" for which the PDB is applied.

2. Min Available: The PDB specifies a minimum number (or percentage) of Pods from the target workload that must remain available during disruptions. For example, you can set a PDB to ensure that at least 50% of the Pods are always available.

3. Max Unavailable: This parameter defines the maximum number (or percentage) of Pods from the target workload that can be unavailable during disruptions. For instance, you might limit the unavailability to 1 Pod at a time.

#### Use Cases and Benefits:

- High Availability: PDBs are crucial for maintaining high availability for stateful applications and other workloads where individual Pods may have data that needs to be preserved.

- Rolling Updates: During rolling updates or node maintenance, PDBs help control the rate at which Pods are taken down and new ones are started. This ensures that your application continues to function within defined availability constraints.

- Stateful Workloads: StatefulSets and other controllers for stateful workloads use PDBs to ensure that scaling and updates do not violate data integrity and the order of operations.

#### How PDB Works:

1. Pod Deletion: When a disruption event occurs (e.g., during node maintenance or scaling down), Kubernetes checks the PDB associated with the target workload to see if the disruption is allowed.

2. Evaluation: The PDB evaluates the current state of the target workload and compares it to the defined Min Available and Max Unavailable constraints.

3. Decision: If the disruption event would result in violating these constraints, Kubernetes prevents the action (e.g., node drain) until it's safe to proceed without breaching the PDB.

#### Example:

Suppose you have a critical application with a PDB set to allow a maximum of 1 Pod to be unavailable. When a node running some of your Pods goes offline, Kubernetes will prevent additional Pods from being taken down until there is only 1 Pod remaining on the node. This guarantees that your application remains operational while still allowing maintenance to take place.

Q.3 What is the role of Load Balance in Kubernetes?





The image shows a terminal window titled "Terminal" with a dark background. The terminal output is as follows:

```
prasad@prasad-VirtualBox: -  
  
update      Update configuration of one or more containers  
wait        Block until one or more containers stop, then print their exit codes  
  
Global Options:  
--config string      Location of client config files (default "/home/prasad/.docker")  
--context string      Name of the context to use to connect to the daemon (overrides DOCKER_HOST env var and default context set with "docker context use")  
-D, --debug           Enable debug mode  
-H, --host list       Daemon socket to connect to  
-l, --log-level string Set the logging level ("debug", "info", "warn", "error", "fatal") (default "info")  
    --tls             Use TLS; implied by --tlsverify  
    --tlscacert string Trust certs signed only by this CA (default "/home/prasad/.docker/ca.pem")  
    --tlscert string   Path to TLS certificate file (default "/home/prasad/.docker/cert.pem")  
    --tlskey string    Path to TLS key file (default "/home/prasad/.docker/key.pem")  
    --tlsverify        Use TLS and verify the remote  
-v, --version         Print version information and quit  
  
Run 'docker COMMAND --help' for more information on a command.  
  
For more help on how to use Docker, head to https://docs.docker.com/go/guides/  
prasad@prasad-VirtualBox:~$ sudo docker run hello-world  
Unable to find image 'hello-world:latest' locally  
latest: pulling from library/hello-world  
719385e32844: Pull complete  
Digest: sha256:88ecbaca3ec199d3b7eaf73588f4518c25f9d34f58ce9a0df68429c5af48e8d  
Status: Downloaded newer image for hello-world:latest  
  
Hello from Docker!  
This message shows that your installation appears to be working correctly.  
  
To generate this message, Docker took the following steps:  
1. The Docker client contacted the Docker daemon.  
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.  
(and64)  
3. The Docker daemon created a new container from that image which runs the  
executable that produces the output you are currently reading.  
4. The Docker daemon streamed that output to the Docker client, which sent it  
to your terminal.  
  
To try something more ambitious, you can run an Ubuntu container with:  
$ docker run -it ubuntu bash  
  
Share images, automate workflows, and more with a free Docker ID:  
https://hub.docker.com/  
  
For more examples and ideas, visit:  
https://docs.docker.com/get-started/  
  
prasad@prasad-VirtualBox:~$
```

[illegible]

## 2. Install minikube using following commands

```

prasad@prasad-VirtualBox:~$ curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
% Total % Received % Xferd Average Speed Time Time Time Current
100 82.4M 100 82.4M 0 0 5315k 0 0:00:15 0:00:15 --:--:-- 5910k
prasad@prasad-VirtualBox:~$ sudo install minikube-linux-amd64 /usr/local/bin/minikube
[sudo] password for prasad:
prasad@prasad-VirtualBox:~$ minikube start --driver=docker
minikube v1.31.2 on Ubuntu 20.04 (vbox/amd64)
Using the docker driver based on user configuration
Exiting due to PROVIDER_DOCKER_NEWGRP: "docker version --format <no value>:<no value>:<no value>" exit status 1: permission denied while trying to connect to the Docker daemon socket at unix:///var/r
un/docker.sock: Get "http://n2fvar2furn2fdocker.sock/v1.24/version": dial unix /var/run/docker.sock: connect: permission denied
Suggestion: Add your user to the 'docker' group: 'sudo usermod -aG docker $USER && newgrp docker'
Documentation: https://docs.docker.com/engine/install/linux-postinstall/

prasad@prasad-VirtualBox:~$ sudo usermod -aG docker $USER && newgrp docker
prasad@prasad-VirtualBox:~$ minikube start --driver=docker
minikube v1.31.2 on Ubuntu 20.04 (vbox/amd64)
Using the docker driver based on user configuration

The requested memory allocation of 1971MiB does not leave room for system overhead (total system memory: 1971MiB). You may face stability issues.
Suggestion: Start minikube with less memory allocated: 'minikube start --memory=1971mb'

Using Docker driver with root privileges
Starting control plane node minikube in cluster minikube
Pulling base image ...
Downloading Kubernetes v1.27.4 preload ...
> preloaded-images-k8s-v18-v1...: 393.21 MiB / 393.21 MiB 100.00% 2.85 Mi
> gcr.io/k8s-minikube/kicbase...: 447.62 MiB / 447.62 MiB 100.00% 2.99 Mi
Creating docker container (CPUs=2, Memory=1971MiB) ...

Docker is nearly out of disk space, which may cause deployments to fail! (94% of capacity). You can pass '--force' to skip this check.
Suggestion:
Try one or more of the following to free up space on the device:
1. Run "docker system prune" to remove unused Docker data (optionally with "-a")
2. Increase the storage allocated to Docker for Desktop by clicking on:
Docker icon > Preferences > Resources > Disk Image Size
3. Run "minikube ssh -- docker system prune" if using the Docker container runtime
Related Issue: https://github.com/kubernetes/minikube/issues/9024

This container is having trouble accessing https://registry.k8s.io
To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
Preparing Kubernetes v1.27.4 on Docker 24.0.4 ...
Generating certificates and keys ...

prasad@prasad-VirtualBox:~$
prasad@prasad-VirtualBox:~$

```

### 3. Install kubectl



```

prasad@prasad-VirtualBox: ~
$ minikube start
Using Docker driver with root privileges
Starting control plane node minikube in cluster minikube
Pulling base image ...
Downloading Kubernetes v1.27.4 preload ...
> preload-images-k8s-v18-v1...: 393.21 MiB / 393.21 MiB 100.00% 2.85 Mi
> gcr.io/k8s-minikube/kicbase...: 447.62 MiB / 447.62 MiB 100.00% 2.99 Mi
Creating docker container (CPUs=2, Memory=1971MB) ...
Docker is nearly out of disk space, which may cause deployments to fail! (94% of capacity). You can pass '--force' to skip this check.
Suggestion:
Try one or more of the following to free up space on the device:
1. Run "docker system prune" to remove unused Docker data (optionally with "--a")
2. Increase the storage allocated to Docker for Desktop by clicking on:
Docker icon > Preferences > Resources > Disk Image Size
3. Run "minikube ssh -- docker system prune" if using the Docker container runtime
Related issue: https://github.com/kubernetes/minikube/issues/9024
This container is having trouble accessing https://registry.k8s.io
To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
Preparing Kubernetes v1.27.4 on Docker 24.0.4 ...
Generating certificates and keys ...
Booting up control plane ...
Configuring RBAC rules ...
Configuring bridge CNI (Container Networking Interface) ...
Using image gcr.io/k8s-minikube/storage-provisioner:v5
Verifying Kubernetes components...
Enabled addons: default-storageclass, storage-provisioner
kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
prasad@prasad-VirtualBox: ~$ sudo snap install kubectl --classic
[sudo] password for prasad:
kubectl 1.28.2 from Canonical++ installed
prasad@prasad-VirtualBox: ~$ kubectl get po -A
NAMESPACE   NAME                               READY   STATUS    RESTARTS   AGE
kube-system  coredns-5d78c9869d-6wdgp          1/1     Running   0           16m
kube-system  etcd-minikube                     1/1     Running   0           16m
kube-system  kube-apiserver-minikube           1/1     Running   0           16m
kube-system  kube-controller-manager-minikube  1/1     Running   0           17m
kube-system  kube-proxy-snjnt                  1/1     Running   0           16m
kube-system  kube-scheduler-minikube           1/1     Running   0           16m
kube-system  storage-provisioner               1/1     Running   1 (16m ago) 16m
prasad@prasad-VirtualBox: ~$

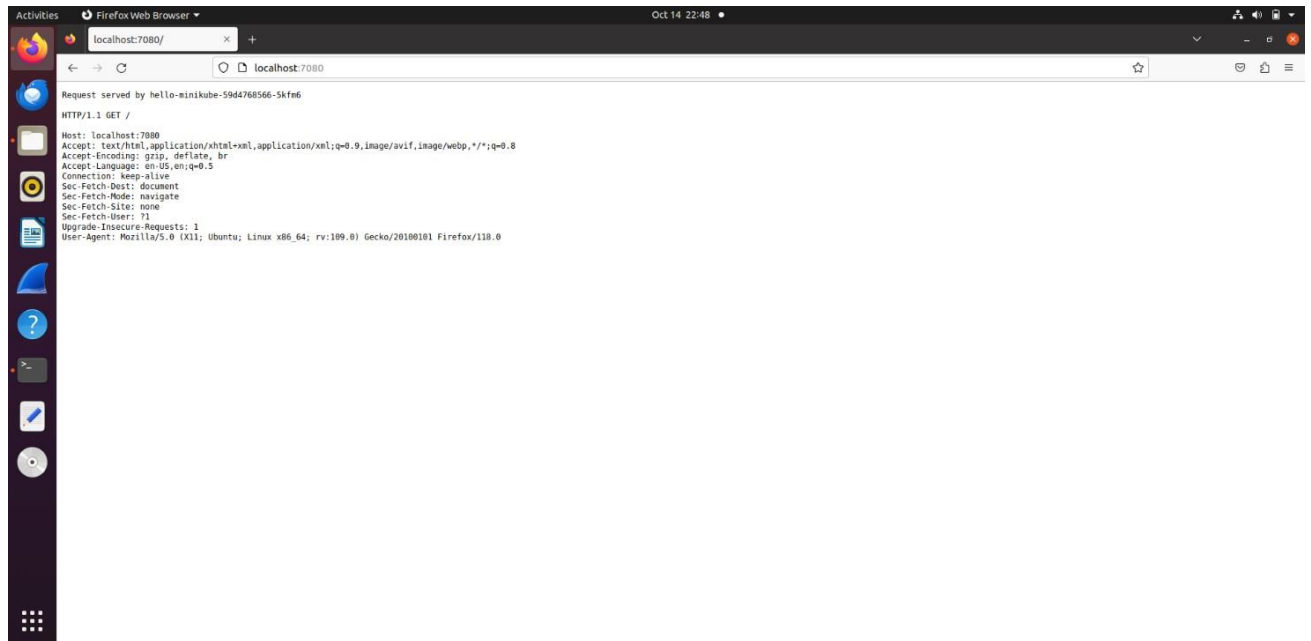
```

#### 4. Create a sample deployment.

```

prasad@prasad-VirtualBox: ~
$ kubectl create deployment hello-minikube --image=kicbase/echo-server:1.0
deployment.apps/hello-minikube created
prasad@prasad-VirtualBox: ~$ kubectl expose deployment hello-minikube --type=NodePort --port=8080
service/hello-minikube exposed
prasad@prasad-VirtualBox: ~$ kubectl get services hello-minikube
NAME      TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
hello-minikube  NodePort  10.106.233.37  <none>        8080:31242/TCP   86s
prasad@prasad-VirtualBox: ~$ kubectl port-forward service/hello-minikube 7080:8080
Forwarding from 127.0.0.1:7080 -> 8080
Forwarding from [::]:7080 -> 8080
Handling connection for 7080

```



## CONCLUSION:

Here we studied Kubernetes cluster architecture in detail. Also we installed Kubernetes in ubuntu machine and created a sample deployment.