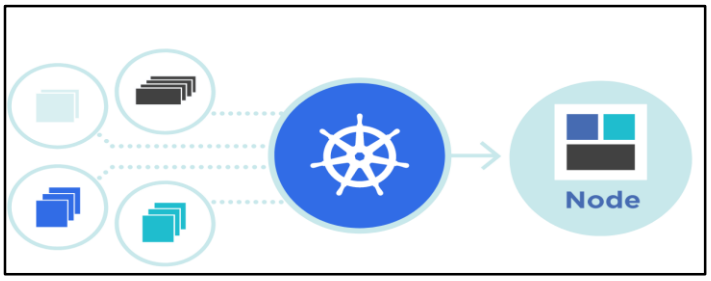
Experiment-10

Aim: To study and implement container orchestration using Kubernetes
Prerequisites: Kubernetes installation
Theory -
Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.



Kubernetes features

Automated rollouts and rollbacks
Kubernetes progressively rolls out changes to your application or its configuration, while monitoring application health to ensure it doesn't kill all your instances at the same time. If something goes wrong, Kubernetes will rollback the change for you. Take advantage of a growing ecosystem of deployment solutions.
Service discovery and load balancing
No need to modify your application to use an unfamiliar service discovery mechanism. Kubernetes gives Pods their own IP addresses and a single DNS name for a set of Pods, and can load-balance across them.
Storage orchestration
Automatically mount the storage system of your choice, whether from local storage, a public cloud provider such as GCP or AWS, or a network storage system such as NFS, iSCSI, Gluster, Ceph, Cinder, or Flocker.
Secret and configuration management
Deploy and update secrets and application configuration without rebuilding your image and without exposing secrets in your stack configuration.
Automatic bin packing
Automatically places containers based on their resource requirements and other constraints, while not sacrificing availability. Mix critical and best-effort workloads in order to drive up utilization and save even more resources.
Batch execution
In addition to services, Kubernetes can manage your batch and CI workloads, replacing containers that fail, if desired.
IPv4/IPv6 dual-stack
Allocation of IPv4 and IPv6 addresses to Pods and Services.
Horizontal scaling
Scale your application up and down with a simple command, with a UI, or automatically based on CPU usage.
Self-healing
Restarts containers that fail, replaces and reschedules containers when nodes die, 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.

Designed for extensibility
Add features to your Kubernetes cluster without changing upstream source code.

Creating Kubernetes cluster locally

Local Kubernetes clusters can be created with various Kubernetes helper programs such as minikube or Kind. For this experiment, we'll use Kind. Kind uses docker containers to run Kubernetes clusters locally.

Kind can be installed on Linux with the following commands.

1. curl -L https://github.com/kubernetes-sigs/kind/releases/download/v0.8.1/kind-linux-amd64 -o kind
2. chmod +x ./kind
3. sudo mv ./kind /usr/local/bin/kind

Step 1: Stop any running docker containers
In order to ensure that no running docker containers cause port conflicts, it is best to stop any running docker containers. This can be done with,
docker stop $(docker ps -a -q)

Step 2: Create a configuration for the docker node
Nodes will require a configuration to work with. We can use a sample YAML configuration given below.

```
# Save this to 'kind.config.yaml'
kind: Cluster
apiVersion: kind.sigs.k8s.io/v1alpha3
nodes:
- role: control-plane
extraPortMappings:
  - containerPort: 30080
        hostPort: 80
        listenAddress: "0.0.0.0"
        protocol: TCP
```

Step 3: Create the cluster
The following command will create a cluster named "mycluster" with the configuration file that we created above. This cluster will be running locally on port 80.
kind create cluster --name mycluster --config config/kind.config.yaml --wait 5m

Deploy applications on Kubernetes

Create services in Kubernetes

A service is an abstraction layer over Pods. It defines a logical set of Pods. It provides a single IP address and DNS name by which pods can be accessed. It is used to expose the pods.

To create a service in Kubernetes, we must first deploy some pods. We can do that as described in the earlier section.
After this, to create a service, we must first create a YAML file detailing the structure of the service.

Create a file called *my-service.yml*and make sure it has the following content.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
externalTrafficPolicy: Local
  ports:
  - name: http
    port: 80
    protocol: TCP
targetPort: 80
  selector:
   app: nginx
  type: NodePort
```

To finally deploy the service, we can use
kubectl create -f my-service.yml

to check up on the status or details of the service, we can do,
kubectl get service | grep nginx
and
kubectl describe service nginx

Conclusion - Understand the steps to deploy Kubernetes Cluster on local systems, deploy applications on Kubernetes, create a service in Kubernetes, develop Kubernetes configuration files in YAML and create a deployment in Kubernetes using YAML.

Sign and Remark:

R1 (3 Marks)	R2 (2 Marks)	R3 (5 Marks)	R4 (5 Marks)	Total (15 Marks)	Signature