

TASK 5 :

KUBERNETES

Submitted by:-
Aditya Kumar Verma

Guided by:
Ashish Mittal Sir

What is Kubernetes?

Kubernetes, also known as K8s, is an open source system for managing [containerized applications](#) across multiple hosts. It provides basic mechanisms for the deployment, maintenance, and scaling of applications.

Kubernetes builds upon a decade and a half of experience at Google running production workloads at scale using a system called [Borg](#), combined with best-of-breed ideas and practices from the community.

Kubernetes is hosted by the Cloud Native Computing Foundation ([CNCF](#)). If your company wants to help shape the evolution of technologies that are container-packaged, dynamically scheduled, and microservices-oriented, consider joining the CNCF. For details about who's involved and how Kubernetes plays a role, read the CNCF [announcement](#).

Kubernetes Features

Kubernetes has many features that help orchestrate containers across multiple hosts, automate the management of K8s clusters, and maximize resource usage through better utilization of infrastructure. Important features include:

- **Auto-scaling**

Automatically scale containerized applications and their resources up or down based on usage.

- **Lifecycle management**

Automate deployments and updates with the ability to:

Rollback to previous versions

Pause and continue a deployment

- **Declarative model**

Declare the desired state, and K8s works in the background to maintain that state and recover from any failures.

- **Resilience and self-healing**

Auto placement, auto restart, auto replication and auto scaling provide application self-healing.

- **Persistent storage**

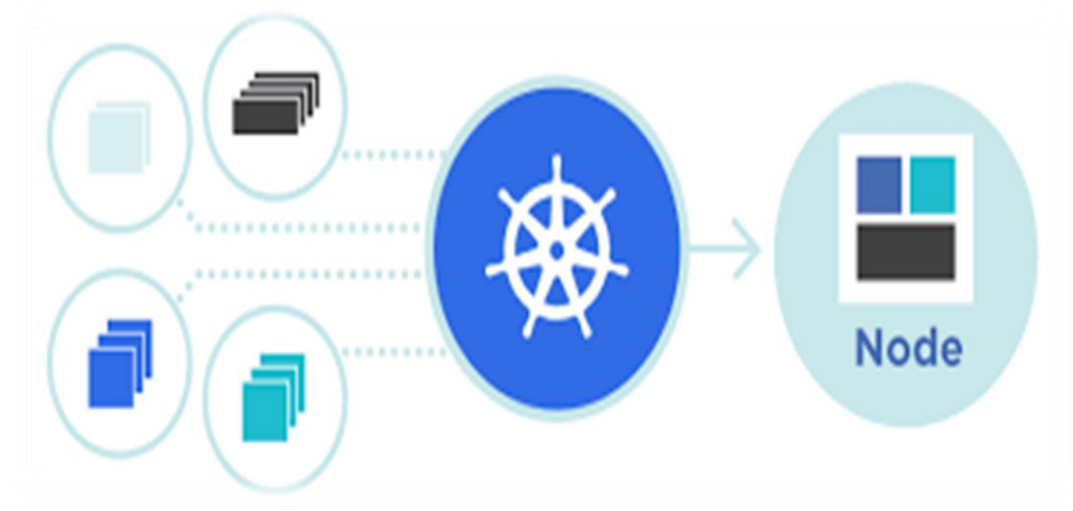
Ability to mount and add storage dynamically.

- **Load balancing**

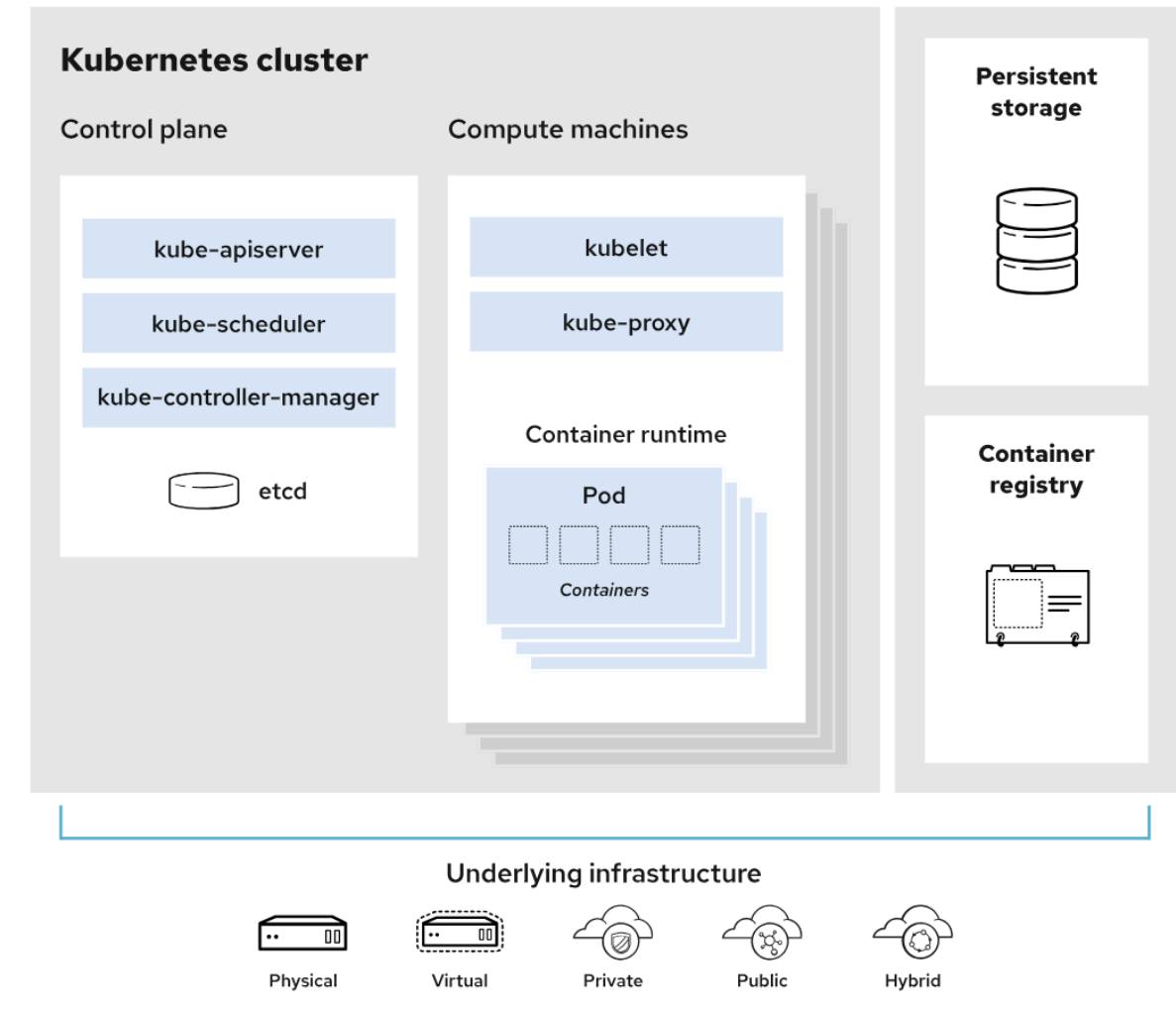
Kubernetes supports a variety of internal and external load balancing options to address diverse needs.

- **DevSecOps support**

[DevSecOps](#) is an advanced approach to security that simplifies and automates container operations across clouds, integrates security throughout the container lifecycle, and enables teams to deliver secure, high-quality software more quickly. Combining DevSecOps practices and Kubernetes improves developer productivity.



Working of Kubernetes






Task 5 :

- kubernetes:-
- killercoda:-playground
- <https://killercoda.com/playgrounds/scenario/kubernetes>
- 1 master and one node
for 60 minutes
- Pods 10:-httpd,caddy,nginx
- 5 from CLI and 5 from Definition-file.
- <https://kubernetes.io/docs/concepts/workloads/pods/>
- name:-every pod different name
label(key=value):-5 types
- 5 rc:-5 label(pod-label)
- each rc should have 4 replicas.
- <https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/>
- deployment:-5

1. Creating a Kubernetes cluster with one master node and one worker node.

K L L R C O D A

 PLUS

AreasAccountCreatorLogout

Kubernetes 1.27

This playground will always have the latest stable Kubernetes version a few weeks after release.

You have access to an empty Kubeadm cluster with two 2GB nodes. The controlplane node has taint removed to be able to schedule workload as well.

This is just an empty environment, if you're looking for scenarios check [CKS](#), [CKA](#), [CKAD](#) or all [Areas](#).

EditorTab1+

Initialising Kubernetes... done

controlplane \$

52 min

RESTARTSCENARIOS

2. Deploy 5 pods with different names and using different container images (httpd, caddy, nginx). You can use the “kubectl run” command to create the pods directly from the command-line interface (CLI):

```
Editor Tab1 + 57 min
controlplane $ kubectl run httpdpod1 --image=httpd
pod/httpdpod1 created
controlplane $ kubectl run httpdpod2 --image=httpd
pod/httpdpod2 created
controlplane $ kubectl run caddypod1 --image=caddy
pod/caddypod1 created
controlplane $ kubectl run caddypod2 --image=caddy
pod/caddypod2 created
controlplane $ kubectl run nginxpod1 --image=nginx
pod/nginxpod1 created
controlplane $ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
caddypod1     1/1     Running   0           37s
caddypod2     1/1     Running   0           31s
httpdpod1     1/1     Running   0           64s
httpdpod2     1/1     Running   0           52s
nginxpod1     1/1     Running   0           12s
controlplane $ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP           NODE     NOMINATED NODE   READINESS GATES
caddypod1     1/1     Running   0           72s   192.168.1.5  node01   <none>            <none>
caddypod2     1/1     Running   0           66s   192.168.1.6  node01   <none>            <none>
httpdpod1     1/1     Running   0           99s   192.168.1.3  node01   <none>            <none>
httpdpod2     1/1     Running   0           87s   192.168.1.4  node01   <none>            <none>
nginxpod1     1/1     Running   0           47s   192.168.1.7  node01   <none>            <none>
controlplane $
```


3. For the remaining 5 pods, create a YAML definition file for each pod and use the “kubectl create” command to create the pods for the destination files.

Step 1 : `controlplane $ vi httpdpod3.yaml`



```
Editor  Tab1  +
apiVersion: v1
kind: Pod
metadata:
  name: caddypod3
  labels:
    key1 : value1
    key2 : value2
    key3 : value3
    key4 : value4
    key5 : value5
  run : caddy
spec:
  containers:
  - name: caddy3
    image: caddy
-- INSERT (paste) --
```

Step 2 :

Step 3 :

```
Editor  Tab 1  +
caddypod2  1/1    Running    0      17m
httpdpod1  1/1    Running    0      17m
httpdpod2  1/1    Running    0      17m
nginxpod1  1/1    Running    0      16m
controlplane $ kubectl create -f httpdpod3.yaml
pod/httpdpod3 created
controlplane $ kubectl get pods
NAME          READY   STATUS             RESTARTS   AGE
caddypod1     1/1     Running            0          18m
caddypod2     1/1     Running            0          18m
httpdpod1     1/1     Running            0          18m
httpdpod2     1/1     Running            0          18m
httpdpod3     0/1     ContainerCreating  0          7s
nginxpod1     1/1     Running            0          18m
controlplane $ kubectl get pods
NAME          READY   STATUS             RESTARTS   AGE
caddypod1     1/1     Running            0          18m
caddypod2     1/1     Running            0          18m
httpdpod1     1/1     Running            0          19m
httpdpod2     1/1     Running            0          18m
httpdpod3     0/1     ContainerCreating  0          24s
nginxpod1     1/1     Running            0          18m
controlplane $ kubectl get pods
NAME          READY   STATUS             RESTARTS   AGE
caddypod1     1/1     Running            0          19m
caddypod2     1/1     Running            0          18m
httpdpod1     1/1     Running            0          19m
httpdpod2     1/1     Running            0          19m
httpdpod3     1/1     Running            0          43s
nginxpod1     1/1     Running            0          18m
controlplane $
```

4. Assign labels to each pod using key-value pairs.
You can assign five different labels with different values to each pod.

```
Editor  Tab 1  +
controlplane $ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
caddypod1     1/1     Running   0           36m
caddypod2     1/1     Running   0           36m
caddypod3     1/1     Running   0           10m
caddypod4     1/1     Running   0           6m4s
httpdpod1     1/1     Running   0           36m
httpdpod2     1/1     Running   0           36m
httpdpod3     1/1     Running   0           17m
ngiinxpod2    1/1     Running   0           3m47s
ngiinxpod1    1/1     Running   0           35m
ngiinxpod3    1/1     Running   0           2m38s
controlplane $ kubectl get pods --show-labels
NAME          READY   STATUS    RESTARTS   AGE   LABELS
caddypod1     1/1     Running   0           36m   key1=value1,key2=value2,key3=value3,key4=value4,key5=value5,run=caddypod1
caddypod2     1/1     Running   0           36m   key1=value1,key2=value2,key3=value3,key4=value4,key5=value5,run=caddypod2
caddypod3     1/1     Running   0           10m   key1=value1,key2=value2,key3=value3,key4=value4,key5=value5,run=caddy
caddypod4     1/1     Running   0           6m9s  key1=value1,key2=value2,key3=value3,key4=value4,key5=value5,run=caddy
httpdpod1     1/1     Running   0           36m   key1=value1,key2=value2,key3=value3,key4=value4,key5=value5,run=httpdpod1
httpdpod2     1/1     Running   0           36m   key1=value1,key2=value2,key3=value3,key4=value4,key5=value5,run=httpdpod2
httpdpod3     1/1     Running   0           17m   key1=value1,key2=value2,key3=value3,key4=value4,key5=value5
ngiinxpod2    1/1     Running   0           3m52s  key1=value1,key2=value2,key3=value3,key4=value4,key5=value5,run=nginx
ngiinxpod1    1/1     Running   0           35m   key1=value1,key2=value2,key3=value3,key4=value4,key5=value5,run=ngiinxpod1
ngiinxpod3    1/1     Running   0           2m43s  key1=value1,key2=value2,key3=value3,key4=value4,key5=value5,run=nginx
controlplane $
```

5. Create five replication controllers (RC), each with a different label selector matching the labels assigned to the pods. Ensure that each RC has four replicas. You can create the RCs using YAML definition files, or the “kubectl create” command.

Step 1:

```
controlplane $ kubectl get pod httpdpod1 --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
httpdpod1 1/1     Running   0           10m   key1=value1,key2=value2,key3=value3,key4=value4,key5=value5,run=httpdpod1
controlplane $ vi rc1.yaml
```

Step 2:

```
Editor  Tab1  +
apiVersion: v1
kind: ReplicationController
metadata:
  name: rc1
spec:
  replicas: 4
  selector:
    key1: value1
  template:
    metadata:
      labels:
        key1: value1
    spec:
      containers:
      - name: httpdpod1
        image: httpd
```

Step 3:

```
controlplane $ kubectl create -f rc1.yaml  
replicationcontroller/rc1 created
```

6. Finally, create five Deployments, each representing a specific application or scenario. You can use the “Kubectl create” command or create YAML definition files for the deployments.

Step 1:

```
controlplane $ vi httpddeploy1.yaml
```

Step 2:

```
Editor  Tab 1  +
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpddeploy1
spec:
  replicas: 4
  selector:
    matchLabels:
      key1: value1
  template:
    metadata:
      labels:
        key1: value1
    spec:
      containers:
        - name: httpd1
          image: httpd
```

Step 3:

```
controlplane $ kubectl create -f httpddeploy1.yaml
deployment.apps/httpddeploy1 created
```


Once cross check the Deployments :

```
controlplane $ kubectl get deployments
NAME          READY  UP-TO-DATE  AVAILABLE  AGE
httpdeploy1   4/4    4           4          33s
controlplane $ kubectl get deployments -owide
NAME          READY  UP-TO-DATE  AVAILABLE  AGE  CONTAINERS  IMAGES  SELECTOR
httpdeploy1   4/4    4           4          43s  httpdeploy1  httpd   key1=value1
```

----- COMPLETED -----