

DAY 4 Tasks – Kubernetes, Kubernetes Architecture, Deploy, Service, Namespace

Kubernetes (K8s)

Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications. The open source project is hosted by the Cloud Native Computing Foundation (CNCF).

It provides a scalable and resilient framework for automating the deployment, scaling, and management of applications across clusters of servers.

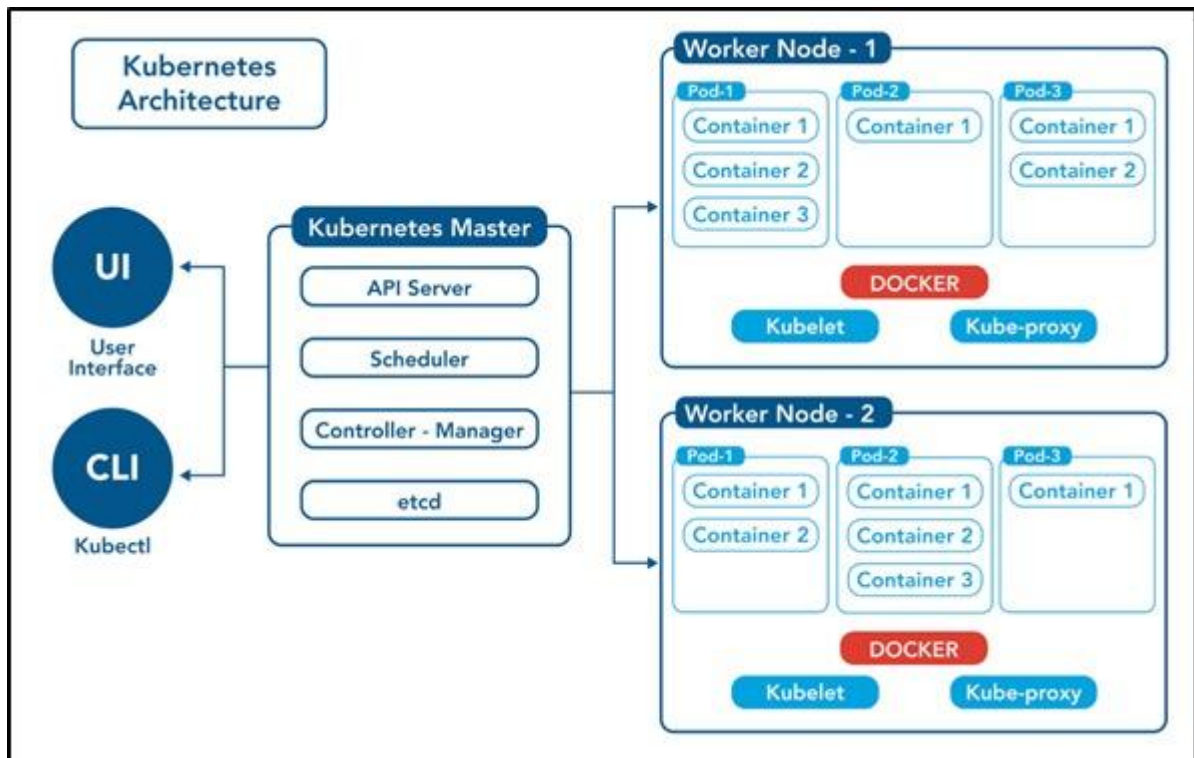
A SMALL HISTORY OF K8S:

- In the early 2000s, Google started developing a system called Borg to manage their internal containerized applications.
- Borg enabled Google to run applications at scale, providing features such as automatic scaling, service discovery, and fault tolerance.
- In 2014, Google open-sourced a version of Borg called Kubernetes.
- Kubernetes was donated to the Cloud Native Computing Foundation (CNCF), a neutral home for open-source cloud-native projects, in July 2015.
- Kubernetes 1.8 added significant enhancements for storage, security, and networking. Key features included the stable release of the stateful sets API, expanded support for volume plugins, and improvements in security policies.
- Check URL: <https://kubernetes.io/releases/> for more release details.

Control Plane /Master Node:

The control plane's components make global decisions about the cluster (for example, scheduling), as well as detecting and responding to cluster events (for example, starting up a new pod when a deployment's replicas field is unsatisfied).

Control plane components can be run on any machine in the cluster. Do not run user containers on this machine.



Node Components / Worker Nodes

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

1. **Master Node:** The master node is responsible for managing the cluster and coordinating the overall state of the system. It includes the following components:

- a. **API Server:** The API server is the central control point for all interactions with the cluster. It exposes the Kubernetes API and handles requests from users and other components.
- b. **Scheduler:** The scheduler is responsible for assigning workloads (pods) to individual worker nodes based on resource requirements, constraints, and other policies.
- c. **Controller Manager:** The controller manager runs various controllers that monitor the cluster state and drive it towards the desired state. Examples include the replication controller, node controller, and service controller.
- d. **etcd:** etcd is a distributed key-value store used by Kubernetes to store cluster state and configuration data.

Pod: The basic building block of Kubernetes. A pod represents a single instance of a running process within the cluster. It can encapsulate one or more containers that share the same network and storage resources.

Services (short name = svc):

Service is an abstraction that defines a logical set of pods and a policy to access them. Services enable network connectivity and load balancing to the pods that are part of the service, allowing other components within or outside the cluster to interact with the application.

Service Types: Kubernetes supports different types of services:

1. **NodePort:** Exposes the service on a static port on each selected node's IP. This type makes the service accessible from outside the cluster by the <NodeIP>:<NodePort> combination.
2. **ClusterIP:** Exposes the service on a cluster-internal IP. This type makes the service only reachable within the cluster.
3. **LoadBalancer:** Creates an external load balancer in cloud environments, which routes traffic to the service.

Commands:

1. Create a pod using run command

```
$ kubectl run <pod-name> --image=<image-name> --port=<container-port>
```

```
$ kubectl run my-pod --image=nginx --port=80
```

2. View all the pods

(In default namespace)

```
$ kubectl get pods
```

(In All namespace)

```
$ kubectl get pods -A
```

For a specific namespace

```
$ kubectl get pods -n kube-system
```

For a specific type

```
$ kubectl get pods <pod-name>
```

```
$ kubectl get pods <pod-name> -o wide
```

```
$ kubectl get pods <pod-name> -o yaml
```

```
$ kubectl get pods <pod-name> -o json
```

3. Describe a pod (View Pod details)

```
$ kubectl describe pod <pod-name>
```

```
$ kubectl describe pod my-pod
```

4. View Logs of a pod

```
$ kubectl logs <pod-name>
```

```
$ kubectl logs my-pod
```

5. Execute any command inside Pod (Inside Pod OS)

```
$ kubectl exec <pod-name> -- <command>
```

2. Create ReplicaSet by executing above YAML file

```
$ kubectl create -f rs-test.yml
```

Do necessary modifications if exist, else create new

```
$ kubectl apply -f rs-test.yml
```

Completely Modify Pod Template

```
$ kubectl replace -f rs-test.yml
```

3. View ReplicaSets

```
$ kubectl get replicaset
```

```
$ kubectl get rs
```

```
$ kubectl get rs -o wide
```

```
$ kubectl get rs <replica-set-name> -o json
```

```
$ kubectl get rs <replica-set-name> -o yaml
```

4. View ReplicaSet Description

```
$ kubectl describe rs <replica-set-name>
```

5. We can modify generated/updated YAML file

```
$ kubectl edit rs <replica-set-name>
```

```
## change replicas: count to any other value then (ESC):wq
```

```
# We can modify our YAML file and then execute apply command
```

```
$ kubectl apply -f rs-test.yml
```

```
## We can Even scale using command also
```

```
$ kubectl scale replicaset <replicaset-name> --replicas=<desired-replica-count>
```

6. Delete ReplicaSet

```
$ kubectl delete rs <replica-set-name>
```

```
$ kubectl delete -f rs-test.yml
```

2. Create Deployment by executing above YAML file

```
$ kubectl create -f web-deploy.yml
```

```
# Do necessary modifications if exist, else create new
```

```
$ kubectl create -f web-deploy.yml
```

```
# Completely Modify Pod Template
```

```
$ kubectl replace -f web-deploy.yml
```

3. View Deployments

```
$ kubectl get deployments
```

```
$ kubectl get deploy
```

```
$ kubectl get deploy -o wide
```

```
$ kubectl get deploy <deployment-name> -o json
```

```
$ kubectl get deploy <deployment-name> -o yaml
```

4. View Deployment Description

```
$ kubectl describe deploy <deployment-name>
```

5. We can modify generated/updated YAML file

```
$ kubectl edit deploy <deployment-name>
```

change replicas: count to any other value then (ESC):wq

We can modify our YAML file and then execute apply command

```
$ kubectl apply -f web-deploy.yml
```

We can Even scale using command also

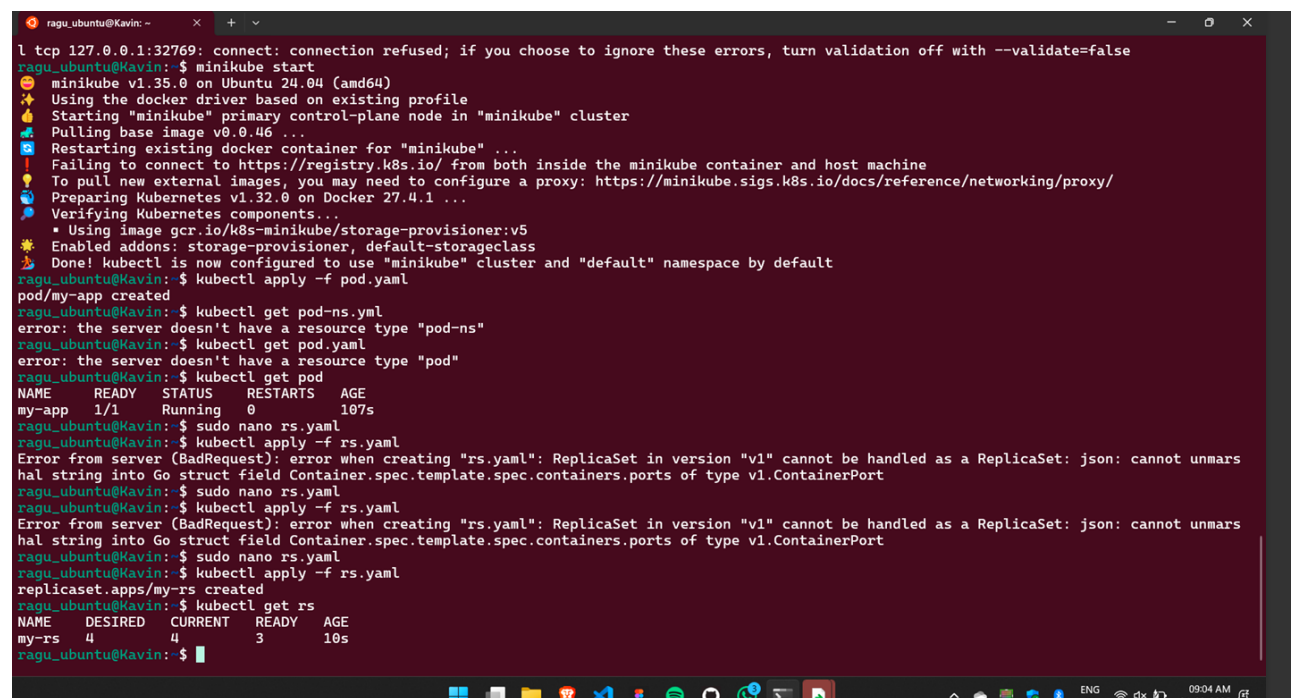
```
$ kubectl scale deploy <deployment-name> --replicas=<desired-replica-count>
```

6. Delete Deployment

```
$ kubectl delete deploy <deployment-name>
```

```
$ kubectl delete -f web-deploy.yml
```

Task – Kubernetes:



```
l tcp 127.0.0.1:32769: connect: connection refused; if you choose to ignore these errors, turn validation off with --validate=false
ragu_ubuntu@Kavin:~$ minikube start
minikube v1.35.0 on Ubuntu 24.04 (amd64)
Using the docker driver based on existing profile
Starting "minikube" primary control-plane node in "minikube" cluster
Pulling base image v0.0.46 ...
Restarting existing docker container for "minikube" ...
Failing to connect to https://registry.k8s.io/ from both inside the minikube container and host machine
To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
Preparing Kubernetes v1.32.0 on Docker 27.4.1 ...
Verifying Kubernetes components...
  * Using image gcr.io/k8s-minikube/storage-provisioner:v5
Enabled addons: storage-provisioner, default-storageclass
Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
ragu_ubuntu@Kavin:~$ kubectl apply -f pod.yaml
pod/my-app created
ragu_ubuntu@Kavin:~$ kubectl get pod-ns.yml
error: the server doesn't have a resource type "pod-ns"
ragu_ubuntu@Kavin:~$ kubectl get pod.yaml
error: the server doesn't have a resource type "pod"
ragu_ubuntu@Kavin:~$ kubectl get pod
NAME      READY   STATUS    RESTARTS   AGE
my-app    1/1     Running   0           107s
ragu_ubuntu@Kavin:~$ sudo nano rs.yaml
ragu_ubuntu@Kavin:~$ kubectl apply -f rs.yaml
Error from server (BadRequest): error when creating "rs.yaml": ReplicaSet in version "v1" cannot be handled as a ReplicaSet: json: cannot unmarshal string into Go struct field Container.spec.template.spec.containers.ports of type v1.ContainerPort
ragu_ubuntu@Kavin:~$ sudo nano rs.yaml
ragu_ubuntu@Kavin:~$ kubectl apply -f rs.yaml
Error from server (BadRequest): error when creating "rs.yaml": ReplicaSet in version "v1" cannot be handled as a ReplicaSet: json: cannot unmarshal string into Go struct field Container.spec.template.spec.containers.ports of type v1.ContainerPort
ragu_ubuntu@Kavin:~$ sudo nano rs.yaml
ragu_ubuntu@Kavin:~$ kubectl apply -f rs.yaml
replicaset.apps/my-rs created
ragu_ubuntu@Kavin:~$ kubectl get rs
NAME      DESIRED   CURRENT   READY   AGE
my-rs     4          4          3       10s
ragu_ubuntu@Kavin:~$
```



```
ragu_ubuntu@Kavin: ~$ kubectl port-forward --namespace default my-service 9000
default      my-service      9000      http://192.168.49.2:30002
Starting tunnel for service my-service.
NAMESPACE    NAME    TARGET PORT    URL
default      my-service      9000      http://127.0.0.1:34289
Opening service default/my-service in default browser...
http://127.0.0.1:34289
! Because you are using a Docker driver on linux, the terminal needs to be open to run it.
^C Stopping tunnel for service my-service.
ragu_ubuntu@Kavin: ~$ curl http://192.168.49.2:30002
<!doctype html><html lang="en"><head><title>HTTP Status 404 - Not Found</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;
} h1, h2, h3, b {color:white;background-color:#525D76;border:none;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {co
lor:black;} .line {height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 404 - Not Found</h1><hr class="line" />
<p><b>Type</b></p><p><b>Status Report</b></p><p><b>Description</b></p> The origin server did not find a current representation for the target resource or is not
willing to disclose that one exists.</p><hr class="line" /><h3>Apache Tomcat/9.0.102</h3></body></html>ragu_ubuntu@Kavin: ~$ curl http://192.168.
49.2:30002/maven-web-app
ragu_ubuntu@Kavin: ~$ curl http://192.168.49.2:30002
<!doctype html><html lang="en"><head><title>HTTP Status 404 - Not Found</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;
} h1, h2, h3, b {color:white;background-color:#525D76;border:none;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {co
lor:black;} .line {height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 404 - Not Found</h1><hr class="line" />
<p><b>Type</b></p><p><b>Status Report</b></p><p><b>Description</b></p> The origin server did not find a current representation for the target resource or is not
willing to disclose that one exists.</p><hr class="line" /><h3>Apache Tomcat/9.0.102</h3></body></html>ragu_ubuntu@Kavin: ~$ curl http://192.168.
49.2:30002/my-app
<!doctype html><html lang="en"><head><title>HTTP Status 404 - Not Found</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;
} h1, h2, h3, b {color:white;background-color:#525D76;border:none;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {co
lor:black;} .line {height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 404 - Not Found</h1><hr class="line" />
<p><b>Type</b></p><p><b>Status Report</b></p><p><b>Description</b></p> The origin server did not find a current representation for the target resource or is not
willing to disclose that one exists.</p><hr class="line" /><h3>Apache Tomcat/9.0.102</h3></body></html>ragu_ubuntu@Kavin: ~$ curl http://192.168.
49.2:30002/my-app
<!doctype html><html lang="en"><head><title>HTTP Status 404 - Not Found</title><style type="text/css">body {font-family:Tahoma,Arial,sans-serif;
} h1, h2, h3, b {color:white;background-color:#525D76;border:none;} h1 {font-size:22px;} h2 {font-size:16px;} h3 {font-size:14px;} p {font-size:12px;} a {co
lor:black;} .line {height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 404 - Not Found</h1><hr class="line" />
<p><b>Type</b></p><p><b>Status Report</b></p><p><b>Description</b></p> The origin server did not find a current representation for the target resource or is not
willing to disclose that one exists.</p><hr class="line" /><h3>Apache Tomcat/9.0.102</h3></body></html>ragu_ubuntu@Kavin: ~$
```

Namespace (short name = ns):

namespace is a virtual cluster or logical partition within a cluster that provides a way to organize and isolate resources. It allows multiple teams or projects to share the same physical cluster while maintaining resource separation and access control.

Task – Namespace:

```
ragu_ubuntu@Kavin: ~$ cat ns.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
  labels:
    name: my-namespace
ragu_ubuntu@Kavin: ~$ kubectl apply -f ns.yaml
namespace/my-namespace created
ragu_ubuntu@Kavin: ~$ kubectl get ns
NAME                STATUS   AGE
default             Active   45h
kube-node-lease     Active   45h
kube-public         Active   45h
kube-system         Active   45h
my-namespace        Active   1m
ragu_ubuntu@Kavin: ~$ kubectl create ns my-deploy
namespace/my-deploy created
ragu_ubuntu@Kavin: ~$ kubectl apply -f deploy.yaml -n my-deploy
deployment.apps/my-deploy created
The Service "my-service" is invalid: spec.ports[0].nodePort: Invalid value: 30002: provided port is already allocated
ragu_ubuntu@Kavin: ~$ sudo nano deploy.yaml
ragu_ubuntu@Kavin: ~$ kubectl apply -f deploy.yaml -n my-deploy
deployment.apps/my-deploy configured
service/my-service created
ragu_ubuntu@Kavin: ~$ kubectl get all --namespace my-deploy
NAME                                READY   STATUS    RESTARTS   AGE
pod/my-deploy-75c67497bc-4gzhd      1/1     Running   0           2m38s

NAME                                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/my-service                 NodePort    10.99.254.94 <none>        9000:30003/TCP   2m5s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/my-deploy          1/1     1             1           2m38s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/my-deploy-75c67497bc 1         1         1       2m38s
ragu_ubuntu@Kavin: ~$
```



```
ragu_ubuntu@Kavin: ~$ kubectl apply -f deploy.yaml -n my-deploy
namespace/my-deploy created
ragu_ubuntu@Kavin: ~$ kubectl apply -f deploy.yaml -n my-deploy
deployment.apps/my-deploy created
The Service "my-service" is invalid: spec.ports[0].nodePort: Invalid value: 30002: provided port is already al
located
ragu_ubuntu@Kavin: ~$ sudo nano deploy.yaml
ragu_ubuntu@Kavin: ~$ kubectl apply -f deploy.yaml -n my-deploy
deployment.apps/my-deploy configured
service/my-service created
ragu_ubuntu@Kavin: ~$ kubectl get all --namespace my-deploy
NAME                                READY    STATUS    RESTARTS    AGE
pod/my-deploy-75c67497bc-4gzhd      1/1      Running   0            2m38s

NAME                                TYPE     CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
service/my-service                  NodePort  10.99.254.94   <none>         9000:30003/TCP   2m5s

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/my-deploy           1/1      1              1            2m38s

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/my-deploy-75c67497bc 1          1          1        2m38s
ragu_ubuntu@Kavin: ~$ kubectl apply -f ns.yaml
namespace/my-demo-ns created
ragu_ubuntu@Kavin: ~$ kubectl apply -f pod-ns.yaml
pod/my-pod created
ragu_ubuntu@Kavin: ~$ kubectl get pod my-demo-ns
Error from server (NotFound): pods "my-demo-ns" not found
ragu_ubuntu@Kavin: ~$ sudo nano ns.yaml
ragu_ubuntu@Kavin: ~$ sudo nano pod-ns.yaml
```