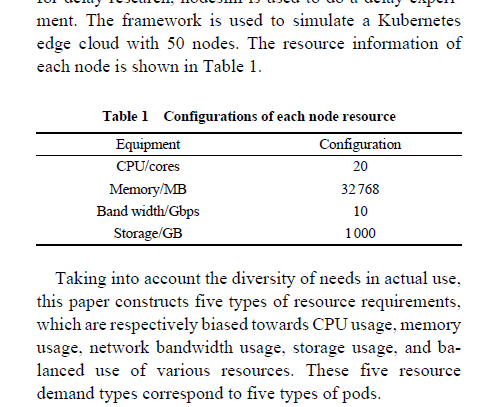
The cluster consists of a master node and several working nodes [11−13]. The cluster master node is the core node of the entire cluster and all commands and operations for the Kubernetes cluster are executed by it. It is responsible for the scheduling and management of the entire cluster,

and is generally an independent server in the cluster. The master node mainly includes Kube-apiserver, Kube-controller, Kube-scheduler and Etcd. Except for the master node in the Kubernetes cluster, the rest of the nodes are called slave nodes. The slave node acts as a real working node, which runs containers for business applications. The components of the worker node mainly include

Kubelet, Kube-proxy and container runtime. The overall architecture of Kubernetes [14] is shown in Fig. 1.



**POD**

* A Pod always runs on a **Node (worker node or master node).**
* **A pod is the smallest building block or basic unit of scheduling in Kubernetes.**
* **In a Kubernetes cluster, a pod represents a running process.**
* **Inside a pod, you can have one or more containers**. **Those containers all share a unique network IP, storage, network and any other specification applied to the pod.**
* POD is a **group of one or more containers** which will be running on some node.
* Pods abstract network and storage away from the underlying container.
* This lets you move containers around the cluster more easily.
* Each Pod has its **unique IP Address** within the cluster.
* Any data saved inside the Pod will disappear without a persistent storage.
* We can have pods with one or more containers and pods with one or more volumes also.
* We can create pod in two ways:
  1. Imperative (using commands)
  2. Declarative (manifest file or yaml file)

**Q. what is common for all the containers which are running in the same pod?**

**Ans: Network and storage** both are common for the containers which are running in a same pod.

**Q. Let’s consider I am creating a pod with two containers, does one container will be running in one node and other container will be running in other node. Does both the containers are running will be in same node for that pod?**

**Ans:** if I have a pod with two containers, **both containers will be running in a same node.**

* we are not going to create or build the images in k8s.
* k8s is only for deployment, we already have a images those images we are going to deploy in this k8s.

**How to Deploy:**

* nodes are basically servers or machines.

**$ kubectl get nodes**

NAME STATUS ROLES AGE VERSION

ip-172-31-34-237 Ready <none> 27d v1.25.4

ip-172-31-36-231 Ready control-plane 27d v1.25.4

ip-172-31-39-236 Ready <none> 27d v1.25.4

**$ kubectl get pods**

No resources found in **default namespace.**

**#: $ kubectl get namespace**

NAME STATUS AGE

default Active 27d

kube-node-lease Active 27d

kube-public Active 27d

kube-system Active 27d

by default, this kubectl pointing to default namespace.

**$ kubectl get all**

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 27d

**creating pod in imperative way:**

**$: kubectl run javawebapp --image=** **rmkrms1519/java-web-app:1 --port=8080**

**(or)**

**Create pod in Declarative (manifest file):**

**we can define what we need in the form of yaml or yml:**

**Attributes:**

apiVersion: kind: metadata: spec:

**Example:**

**apiVersion**: v1

**kind:** Pod (what kind of object you want to create)

**metadata:** (data about this object)

name: javawebapppod (pod name)

labels:

app: javawebapp

**spec:**

containers:

- name: javawebappcontainer (container name)

image: rmkrms1519/java-web-app:1 (from which image we want to create)

ports:

- containerPort: 8080 (container port)

**$:kubectl apply –f <<filename.yaml>>**

**Mostly we are using declarative.**

* **$: kubectl run javawebapp --image=** **rmkrms1519/java-web-app:1 --port=8080 🡪** The moment we execute this command request goes to API server. API server will persist that information in etcd. Scheduler will try to find a node or assign a node for that unscheduled pods. The API server will work with kubelet. The kubelet which is running in that node to create the containers.
* **Request goes to API server 🡪 API server will validate that request and Scheduler will try to find a node or assign a node to that pod and kubelet will start creating the containers.**
* **$: kubectl run javawebapp --image=rmkrms1519/java-web-app:1 --port=8080 –v=7 🡪 we can see what is happening in the background.**
* **$ kubectl get pods**

**NAME READY STATUS RESTARTS AGE**

**javawebapp 0/1 ImagePullBackOff 0 102s**

* **$ kubectl get events**

we can see what is happening in the background:

pod is assigned to some worker node.

Pulling image

Pulled image

Created container

Started container

* **$ kubectl get pods -o wide**

NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

javawebapp 1/1 Running 0 3h4m 10.36.0.1 ip-172-31-39-236 <none> <none>

**10.36.0.1** 🡪 **This IP is pod IP**

* **To see more details about any k8ss object**

Syn: kubectl describe <<object>> <<Objectname>>

* **$ kubectl describe pod javawebapp**
* **$ kubectl get nodes -o wide**

NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME

ip-172-31-34-237 Ready <none> 27d v1.25.4 172.31.34.237 <none> Ubuntu 20.04.5 LTS 5.15.0-1023-aws **containerd**://1.5.9 🡪 here we are having contained.

**Q. how can we access this pod which is deployed in kubernetes cluster internally or externally?**

Ans: To access the pods we are using one object called “Service” and it is not recommended using pod IP’s.

**Service: to access the pods, within the cluster or outside the cluster.**

**3 kubernetes73:**

**$ kubectl version**

**$ cat ~/.kube/config 🡪** This is kube config file. It has the cluster information. With the help of this kube config file this kubectl is communicating with control plane.

**$ kubectl get nodes**

**$ kubectl get all**

**NAME READY STATUS RESTARTS AGE**

**Pod/javawebapp 1/1 Running 0 102s**

This pod is running default namespace

**$ kubectl get ns 🡪 To get namespaces**

NAME STATUS AGE

default Active 27d

kube-node-lease Active 27d

kube-public Active 27d

kube-system Active 27d

**NameSpace:**

* Name spaces are basically a kind of logical grouping of k8s resources of some applications or some teams or some projects.
* ns is like a cluster inside the cluster which is logically isolated from each other.
* We can use namespaces for managing the pods/applications/managing k8s deployments of a multiple teams in a same k8s cluster.
* We can apply security and resource allocation at namespace level.

**Manifest file for namespace:**

* **apiVersion**: v1
* **kind:** namespace (what kind of object you want to create)
* **metadata:** (data about this object)

name: test-ns (namespace name)

* **spec: 🡪 no spec attribute**

**how to create any k8s resource?**

With the help of kubectl we can create k8s resources.

**How to know apiVersion: V1?**

By executing the below command:

**Kubectl api-resources**

**How containers in same pod can communicate with each other?**

They can use **local host**, because they sharing same network space. <localhost>: portno.

**Service:**

* **How one pod will communicate with other pod?**
* K8's Service ---> In Kubernetes Service makes our pods accessable/discoverable with in the cluster or exposing them to internat. service **will identify pods using it's labels And Selector.** Whenever we create a service a ClusterIP (virtual IP) Address will be allocated for that serivce and DNS entry will be created for that IP. So internally we can access using service name(DNS).
* **Labels:** labels are basically key value pairs.
* If pod label is not matching with selector in service, it will not send the traffic to that pod.
* Types of services: 1 cluster IP 2 Node Port 3 load balancer
* **ClusterIP:** Cluster IP service provides **the communication between the Pods within the cluster** using service name.
* Request goes to service; the service will route the traffic to the respective pods **based on the labels whatever they are matching.**
* If we have multiple pods for that service, then this service acts as a load balancer.
* The default service is ClusterIP service.
* **How service will discover or identify the pods?**
* Based on the labels and selectors.
* Service will have selectors.
* **How can I trouble shoot the object?**
* Kubectl describe <<object>> <<objectname>>
* **How private repo authenticate to remove image pullback off error?**
* we can pass the registry authenticate details as an image pull secret.
* **Secret:** we can create a secret for docker images. These secrets we can pass to pod manager.
* Cmd**: kubectl create secret docker-registry Dockerhubsecret –docker-server=myjfrog.com –docker-username=rmkrms1519 –docker –password=\*\*\*\*\*** [**--docekr-emai=rmkrms1519@gmial.com**](mailto:--docekr-emai=rmkrms1519@gmial.com) **🡪** we crated the secret and then in pod manifest file that secret refer as “imagepullsecret:

-name:regcred”

Docker-registry: secret type

Dockerhubsecret: secret name

Kubelet will authenticate with registry and then pulls image.

* **If one application in one namespace want to communicate with other application in other namespace we are using FQDN (fully qualified domain name).**
* **Syntax: <<service name>>.<<namespace>>.svc.cluster.local**
* Node Port Service: to access the application from outside the cluster. nodeport range:30000-32767.
* From outside the cluster we can reach that pod with the help of <nodeIP>:<nodeport>.

apiVersion: v1

kind: Service

metadata:

name: mavenwebappsvc

spec:

type: ClusterIP

selector:

app: mavenwebapp

ports:

- port: 80

targetPort: 8080

* **kubectl get ns (or) kubectl get namespaces** 🡪 **To get namespaces**
* **Kubectl api-resources🡪 To know apiVersion: V1**
* **kubectl get all**
* **kubectl get pods**
* **kubectl get pods --show-labels**
* **kubectl get pods - o wide**
* **kubectl get pods - o wide --show-labels**
* **kubectl describe pod <podName>**
* **kubectl describe pod <podName> -n <namespace>**

Note: If we don't mention -n <namespace> it will refer default namespace. If required, we can change name space context.

* **kubctl config set-context --curent --namespace=<namespace>**

ex: **kubectl config set-context --curent --namespace=flipkart**

After setting context by default it will point to that namespace.

Change it to default namespace again if required

ex: **kubectl config set-context --curent --namespace=default**

* **kubectl get events**

we can see what is happening in the background

* **kubectl describe <<object>> <<Objectname>> 🡪**To see more details about any k8ss object (or) To troubleshoot the object in default namespace.
* **kubectl describe <<object>> <<Objectname>> -n <<namespacename>>🡪**To see more details about any k8ss object (or) To troubleshoot the object in specific namespace.
* **Kubectl apply –f <objectmanifestfilename> -v=7 🡪To deploy**

To create k8s object(-v=7🡪print logs/what is happening in the background)

* **Kubectl get resourcequota –n <<namespacename>>** 🡪To see resource quota of a specified namespace
* **Kubectl get all –-all-namespaces (or) Kubectl get all –all-namespaces (or) kubectl get all –A🡪 To see all the objects in all the namespaces.**
* **Kubectl get all –n <<namespace name>>**🡪 To see all the objects in the specific namespace.
* **Kubectl get pods –o wide –n <<specificnamespacename>>**🡪To specific namespace pods only.
* **Kubectl get all** 🡪 To see all the objects in a default namespace.
* **Kubectl get pods –show-labels –n <<namespacename>>** 🡪 To see the labels
* **Kubectl delete pod <<podname>> -n <<namespacename>>** 🡪 To delete pod in specific namespace
* **Kubectl apply –f <<manifestfilename>> --dry-run=client** 🡪 To do the syantax, whether the yaml syntax, the attributes of manifest file is correct or not, we can do some kind of dry run to validate yaml is proper or not.
  + **Error Parsing:** identification not proper
  + **Imagepullbackoff error:**

**1.**registry does’t exist

2.registry is correct and repo is not correct.

3.reg &repo correct but the tag does’t exist

4.for private repo authenticates issues

* **Kubectl get secret** 🡪to get secret in default namespace
* **Kubectl get secret –n <<specific namespace>>**🡪 To get secret in specific namespace
* **kubectl create secret docker-registry Dockerhubsecret –docker-server=myjfrog.com –docker-username=rmkrms1519 –docker –password=\*\*\*\*\*** [**--docekr-emai=rmkrms1519@gmial.com**](mailto:--docekr-emai=rmkrms1519@gmial.com) **🡪 TO create secret in default namespace.**
* **kubectl create secret docker-registry Dockerhubsecret –docker-server=myjfrog.com –docker-username=rmkrms1519 –docker –password=\*\*\*\*\*** [**--docekr-emai=rmkrms1519@gmial.com**](mailto:--docekr-emai=rmkrms1519@gmial.com) **–n <<namespacename>> 🡪 To create secret in specific namespace.**

**Manifest file for Pod in test-ns namespace:**

**apiVersion**: v1 (for creating pod apiVersion is V1)

**kind:** Pod (what kind of object you want to create(P-caps))

**metadata:** (data about this object)

name: mavenwebapppod (pod name—all small letters)

namespace: test-ns (if don’t mention this namespace name(test-ns) then it will consider default namespace—creating pod in test-ns ns)

labels: (any key, any value)

app: mavenwebapp

**spec:**

containers:

- name: mavenbappcontainer (container name)

image: rmkrms1519/maven-web-app:1 (from which image we want to create)

ports:

- containerport: 8080 (container port)

**Imagepullsecret:** the master m/c will passes this infoto kublet

**--namedockerhubsecret**

**------(manifest file for service)**

**apiVersion**: v1 (for creating service apiVersion is V1)

**kind:** service (what kind of object you want to create)

**metadata:** (data about this object)

name: mavenwebappsvc (service name—all small letters)

namespace: test-ns (if don’t mention this namespace name(test-ns) then it will consider default namespace

**spec:**

type: clusterIP

selector:

app: mavenwebapp (selectors should be pod labels)

ports:

- port: 80 (serviceport)

Targetport:8080 (targetport should be same as container port)

(the request goes to the service port 80, then it is going to route the traffic to target port 8080)

**# Multi Container POD**

apiVersion: v1

kind: Pod

metadata:

name: <PODName>

namespace: <nameSpaceName>

labels:

<labelKey>: <labelValue>

spec:

containers:

- name: <nameOftheCotnainer>

image: <imageName>

ports:

- containerPort: <portNumberOfContainer>

- name: <nameOftheCotnainer>

image: <imageName>

ports:

- containerPort: <portNumberOfContainer>