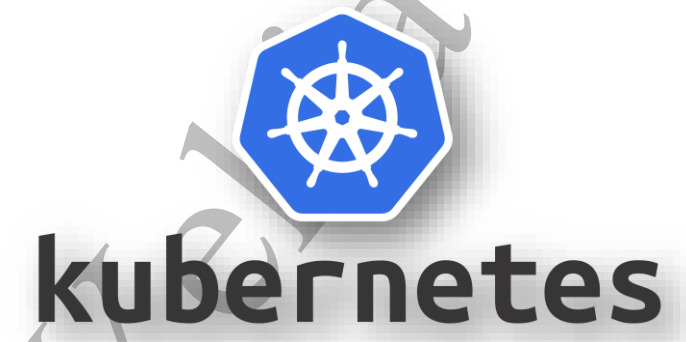



**By: Eng. Sherif Yehia Mostafa**

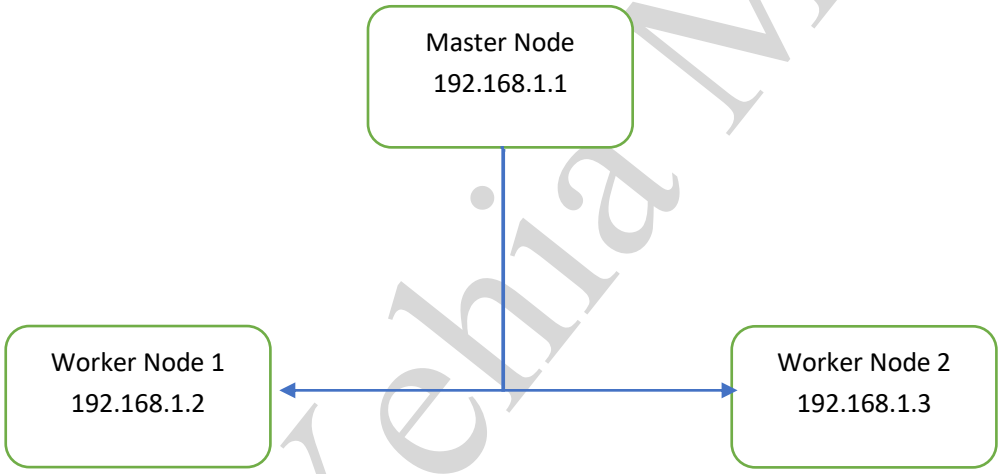
**Email:** sherif.yehiaee@gmail.com

**LinkedIn:** linkedin.com/in/sherif-yehia-389071178/

# Table of Contents.

- **Install k8s from scratch: -**
  - Requirements: -
    1- 3 machines – OS: Ubuntu 20.04.
    2- 2 GB or more of RAM per machine.
    3- 2 CPUs or more.
    4- Full network connectivity between all machines in the cluster, (you can disable firewall just testing environment).
    5- Unique hostname, MAC address, and product_uuid for every node.
    6- Root user.

  - Architecture: -




  - **Installation Steps: -**
    a) **Command run through ( Master node – Worker Node 1 – Worker Node 2 ) :-**
       1- **You MUST disable swap**
       **$$ sudo swapoff -a**

       **Then disable swap as a below**
       **$$ nano /etc/fstab**

```
 GNU nano 4.8                                                /etc/fstab
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/sda5 during installation
UUID=64462a1e-11fe-4011-9fc2-b373aeabcd59 /               ext4    errors=remount-ro 0       1
# /boot/efi was on /dev/sda1 during installation
UUID=3370-2476  /boot/efi       vfat    umask=0077      0       1
#/swapfile                                 none            swap    sw              0       0
```

2- **Set up the IPV4 bridge on all nodes ( run all below at one time).**

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

# sysctl params required by setup, params persist across reboots
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF

# Apply sysctl params without reboot
sudo sysctl --system
```

3- **Verify that the br_netfilter, overlay modules are loaded by running the following commands:**

**$$lsmod | grep br_netfilter**

**$$lsmod | grep overlay**

```
root@( Master-node                :~# lsmod | grep br_netfilter
br_netfilter           28672  0
bridge                249856  1 br_netfilter
root@( Master-node                :~# lsmod | grep overlay
overlay               126976  0
```

4- **Verify that the net.bridge.bridge-nf-call-iptables, net.bridge.bridge-nf-call-ip6tables, and net.ipv4.ip_forward system variables are set to 1 in your sysctl config by running the following command:**

**$$sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables net.ipv4.ip_forward**

```
root@                  :~# sysctl net.bridge.bridge-nf-call-iptables net.bridge.bridge-nf-call-ip6tables net.ipv4.ip_forward
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
```

5- **Disable firewall (to make sure 3 machines can connect together): -**

**$$ufw status**

```
root@ Master-node                    :~# ufw status
Status: inactive
```

6- **Install Containerd from docker website: -**

**$$sudo mkdir /etc/apt/keyrings**

**$$sudo apt-get update**

**$$sudo apt-get install ca-certificates curl gnupg**

**$$sudo install -m 0755 -d /etc/apt/keyrings**

**$$curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg**

**$$sudo chmod a+r /etc/apt/keyrings/docker.gpg**

**$$echo **
**"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg]**
**https://download.docker.com/linux/ubuntu **
**"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | **
**sudo tee /etc/apt/sources.list.d/docker.list > /dev/null**

**$$sudo apt-get update**

**$$sudo apt-get install containerd.io**

**$$systemctl status containerd.service**

**Check Containerd service! Should be as below active!**

```
root@                    l:~# systemctl status containerd.service
● containerd.service - containerd container runtime
     Loaded: loaded (/lib/systemd/system/containerd.service; enabled; vendor preset: enabled)
     Active: active (running) since Sun 2023-11-26 14:41:24 EET; 1 weeks 0 days ago
       Docs: https://containerd.io
    Process: 845276 ExecStartPre=/sbin/modprobe overlay (code=exited, status=0/SUCCESS)
   Main PID: 845277 (containerd)
      Tasks: 185
     Memory: 4.6G
     CGroup: /system.slice/containerd.service
             ├─ 24480 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id dc266593eaa49953dcc95da8a6b0adce7502a80509b7e3b806bfcf438c2>
             ├─ 24481 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id 1abff8f5b814b094ab78a084a099460e866dac7fdb2c0e4a546284659d4>
             ├─ 28444 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id f6c8dd3a33fcca785038c583462282f3473a667f63c7a9761a7af915502>
             ├─611826 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id f4525ddb8fdfb44905a17c6a3cc6b857d3334cab6b78c9505c87bb66396>
             ├─612304 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id 6b0bffb760515f719898f45622ef70ce82578df2b5da87ea6403f534afc>
             ├─731883 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id 4bd8c9f0bb4d8e1a0d3011a5e3400466159aa16bb652007d75d7c0d5c83>
             ├─731892 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id 2c12e90b92ad5b7a86b01ea3c15ed20838cca16712cacdeb53a021a850c>
             ├─732056 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id 6e949bf17ab7eb28c6e9e8f063df04d73be827c5d24f64fde16e947ee1b>
             ├─732130 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id 3170a03b7aa4107cff45522db06f3eb321817222d3d20f35953a12f82fb>
             ├─732204 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id 01305d5e5b20e5758f6e9153346bd2def3660a645e80bbb8b4900885a5d>
             ├─733275 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id 651ccb11e85f9a087ce70c11d11e5838d0afbbbf81ad50945a17ac2eb20>
             ├─845277 /usr/bin/containerd
             └─866624 /usr/bin/containerd-shim-runc-v2 -namespace k8s.io -id 5ff3c0b0009bdcad52c5f53873e5f8ec0fc1cdf9b81b7f3bae60d8098d3>
```

7- **Set up Cgroup as a true (systemd)**

**$$ nano /etc/containerd/config.toml**

```
          NoPivotRoot = false
          Root = ""
          ShimCgroup = ""
          SystemdCgroup = true

    [plugins."io.containerd.grpc.v1.cri".containerd.untrusted_workload_runtime]
```

**Then restart service to read new configuration.**

**$$ systemctl restart containerd.service**

8- **Let's install kubelet, kubeadm, and kubectl to create a Kubernetes cluster. They play an important role in managing a Kubernetes cluster.**

**$$ sudo apt-get update**

**$$ sudo apt-get install -y apt-transport-https ca-certificates curl**

**$$ mkdir -p /etc/apt/keyrings**

**$$ curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-archive-keyring.gpg**

**$$ echo "deb [signed-by=/etc/apt/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list**

**$$ sudo apt-get update**

**$$ sudo apt install -y kubelet=1.25.4-00 kubeadm=1.25.4-00 kubectl=1.25.4-00**

**$$ sudo apt-mark hold kubelet kubeadm kubectl**

**To check the installation of kubelet-kubeadm-kubectl :-**

**$$ kubeadm version**
**$$ kubelet --version**
**$$ kubectl version –short**

```
root@                    :~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"25", GitVersion:"v1.25.4", GitCommit:"872a965c6c6526caa949f0c6ac028ef7aff3fb78", GitTree
State:"clean", BuildDate:"2022-11-09T13:35:06Z", GoVersion:"go1.19.3", Compiler:"gc", Platform:"linux/amd64"}
root@                    :~# kubelet --version
Kubernetes v1.25.4
root@                    :~# kubectl version --short
Flag --short has been deprecated, and will be removed in the future. The --short output will become the default.
Client Version: v1.25.4
Kustomize Version: v4.5.7
The connection to the server localhost:8080 was refused - did you specify the right host or port?
root@                    :~#
```

b) **Command run through ( Master node only – 192.168.1.1 ) :-**

1- **initialize your master node. The --pod-network-cidr flag is setting the IP address range for the pod network (Must use range 10.244.0.0 for flannel network package – Next step).**

- **192.168.1.1 is the IP of master node.**

**$$kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-address=192.168.1.1**

The result of initialize is below:

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.1.1:6443 --token nuz9ws.fvflg8ht7dqg913h \
    --discovery-token-ca-cert-hash sha256:20ea7b03841b072c3b68d6ec14b772efead9054f1accb34b55f0a75911549cd8
```

2- **Set up config file.**

**$$ mkdir -p $HOME/.kube**
**$$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**
**$$ sudo chown $(id -u):$(id -g) $HOME/.kube/config**

c) **Join workers node to the cluster through (WorkerNode1 192.168.1.2 - WorkerNode2 192.168.1.3) in upper photo.**

1- **From machine WorkerNode1 run**

**$$ kubeadm join 192.168.1.1:6443 --token d2tgh8.f9q3vjf1i5t1uneu **
**--discovery-token-ca-cert-hash**
**sha256:59b6ac2294eb69ccf84743fc2b9ea5113b64bbe5ea0d5372938b1e81468c47da**

**Copy certificate from MasterNode path /root/.kube/config**
**Then paste it in WorkerNode1 in same path**
**/root/.kube/config**

2- **From machine WorkerNode2 run**

**$$ kubeadm join 192.168.1.1:6443 --token d2tgh8.f9q3vjf1i5t1uneu **
**--discovery-token-ca-cert-hash**
**sha256:59b6ac2294eb69ccf84743fc2b9ea5113b64bbe5ea0d5372938b1e81468c47da**

**Copy certificate from MasterNode path /root/.kube/config**
**Then paste it in WorkerNode2 in same path**
**/root/.kube/config**

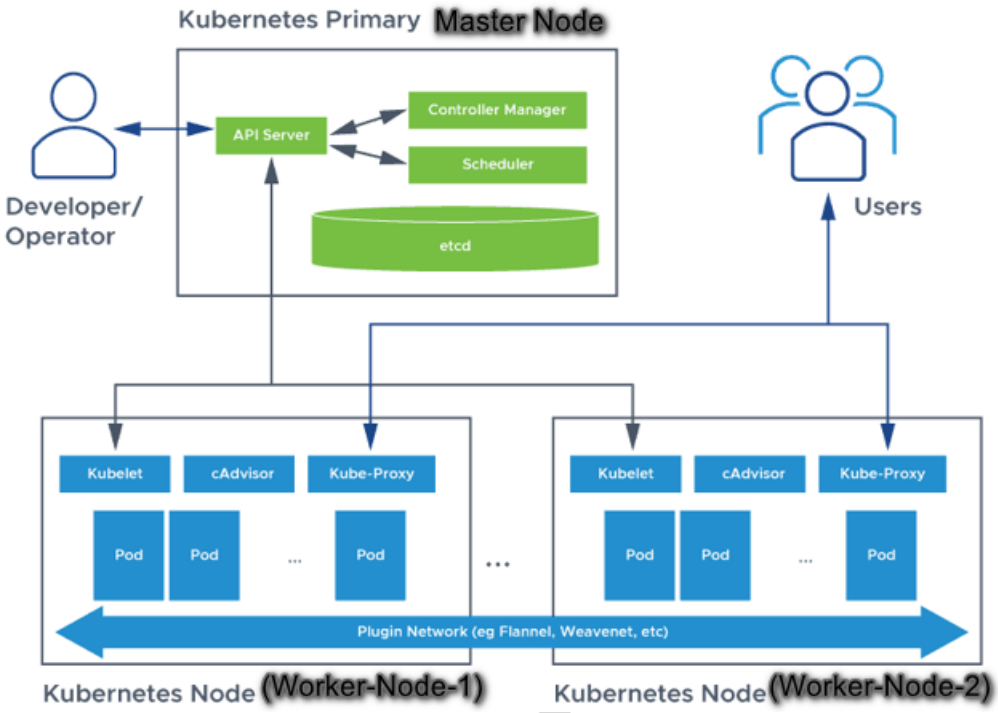
d) **Command run through ( Master node only 192.168.1.1 ) :-**

1- **Install flannel package for network solution.**
**$$ sudo kubectl apply -f**
**https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml**

- **Finally,**
**$$ kubectl get nodes**

```
root@C                    :~# kubectl get nodes
NAME                STATUS   ROLES           AGE   VERSION
Worker-Node-2       Ready    <none>          19d   v1.25.4
Worker-Node-1       Ready    <none>          19d   v1.25.4
Master-Node-1       Ready    control-plane   19d   v1.25.4
root@C                    :~#
```

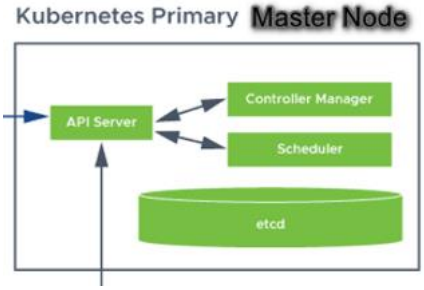
- **What does k8s consist of?**




➢ **Kubernetes Architecture is consisting of Master Node and Worker nodes: -**

- **<u>Master Node</u>: responsible for cluster management and for providing the API that is used to configure and manage resources within the Kubernetes cluster.**
- **<u>Worker Node</u>: are responsible for running the containers and doing any work assigned to them by the master node.**

➢ **Master Node consists of: -**




**1- <u>ETCD CLUSTER</u>: data stores information regarding the cluster such as nodes, pods, configs, secrets, Accounts, roles, bindings, others. When you run the Kube control get command is from Etcd server. Every change in cluster such as adding nodes, deploy pods are updated in the Etcd server.**

**--ETCD listen to port 2379 to connect between ETCD and kube-API.**

**2- Kube-scheduler: scheduler identifies the right node to place a container on based on the containers resource requirements worker node capacity.**

**3- Kube Controller Manager: used to take care of nodes, the responsible for onboarding new nodes to the cluster, handling node become unavailable or destroyed it and the replications nodes.**

**By default, Kube controller management check node every 5 second by using kube-api if it stop receiving heartbeat from node the node will be marked as unreachable but it will waiting 40 second before market it as unreachable then waiting 5 mint as unreachable to comeback up it, if it is not up again ,it will remove the pods=container which creating on that node and put it in another node replication**

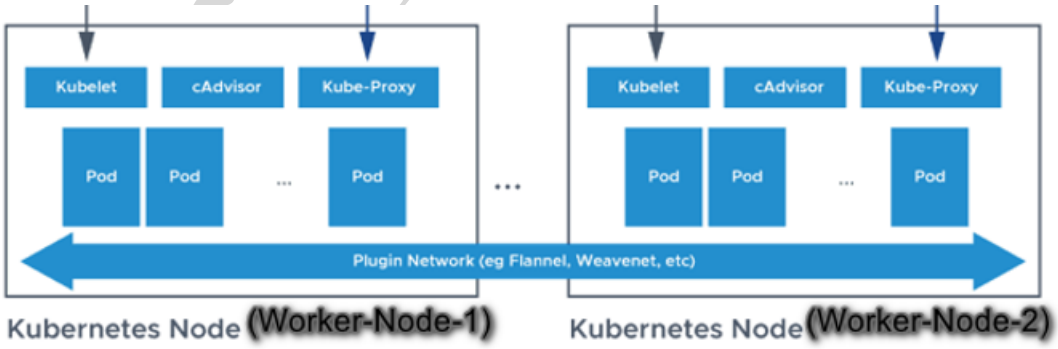
**4- Kube-Apiserver: is APIS used to handle request and response between master node and worker node.**

**Kube-API: by using postman you can post a request (curl –X POST /api/v1/namespaces/default/pods1) to create new pod=container=application; -**

**How kube-API manage that request?**

1- **Kube-api go to ETCD to Authenticate user.**
2- **Then Kube-api validated the request.**
3- **Kube-Api has retrieved data from ETCD and updates new pod in ETCD.**
4- **Kube-Api sent to Kube-scheduler to release there is a new pod need to create to define which node suitable to create on.**
5- **Then Kube-Api gets information from Kube-Scheduler to define the place of new pod and update it in ETCD**
6- **Then Kube-Api sends all information to the Kubelet which manages the place that scheduler defines it to create pod on.**
7- **Kubelet starts to create pod on same node that Kubelet manages it.**

## ➢ Worker Nodes consists of: -

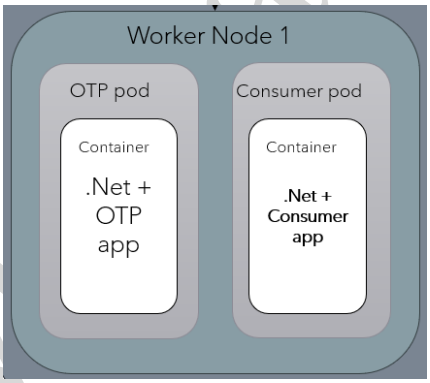



**1- Kubelet: every worker node has its own Kubelet. Kubelet is the captain of Worker nodes which take the order from master node and send reports to master node and status of worker node by using Kube-Apiserver**

**2- Kube-Proxy: Kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.**

- **How to manage k8s?**
  - **What is Pod and how to create one?**

**Pods: Pods are the smallest deployable units of computing that you can create and manage in Kubernetes, pods contain the container that contains the applications + any another dependency.**




**1-create yaml file such as pod-definition.yaml**

**$$nano pod-definition.yaml**

**2-yaml contain: -**

```
apiVersion: v1
kind: Pod
metadata:
   name: myapp-pod
   labels:
      app: myapp
      type: front-end
spec:
   containers:
   - name: nginx-container
     image: nginx
```

apiVersion: v1 ➡ Related to kube-API version on master node (default is v1)

kind: Pod ➡ kind of creation such as (pod – service – ReplicaSet – Deployment)

metadata: ➡ This part is used to give my creation name and labels (using this name and labels to connect service to this pod and many another purpose).

name: myapp-pod ➡ Name of the pod. (up to you)

labels: ➡ It is used to identify this object, so we are using (app - type) to identify this object. (up to you)
- Think about 1000 pods.
- you can add more of labels under app and type

spec: ➡ To specific the container in this Pod we are going to create single container to single pod.

containers: ➡ Part of information about the container of the application

- name: nginx-container ➡ Name of the container of the application (up to you)

image: nginx ➡ Name of the image on docker repository

## 3-Apply yaml file: -

**$$kubectl create -f pod-definition.yaml**

**-f : file name**

**Or**

**--We can use kubectl apply if we have modified yaml file, to terminate old pod and create new one.**

**$$kubectl apply -f pod-definition.yaml**

**--To get status of the new pod**

**$$kubectl get pods -owide**

```
root@C                    :/var/www/demo# kubectl get pods -owide
NAME        READY   STATUS    RESTARTS   AGE   IP             NODE                    NOMINATED NODE   READINESS GATES
myapp-pod   1/1     Running   0          36s   10.244.1.167                           <none>           <none>
```

**--To get more information about the new pod**

**$$kubectl describe pod myapp-pod**

```
root@Co                ch:/var/www/demo# kubectl describe pod myapp-pod
Name:             myapp-pod
Namespace:        default
Priority:         0
Service Account:  default
Node:             worker-node-1/192.168.1.2
Start Time:       Thu, 07 Dec 2023 09:56:52 +0200
Labels:           app=myapp
                  type=front-end
Annotations:      <none>
Status:           Running
IP:               10.244.1.167
IPs:
  IP:  10.244.1.167
Containers:
  nginx-container:
    Container ID:   containerd://36e58f9da77c78233da8f1a90313ecfb422a3bd76a63266a16a2c2c32c0ec17c
    Image:          nginx
    Image ID:       docker.io/library/nginx@sha256:10d1f5b58f74683ad34eb29287e07dab1e90f10af243f151bb50aa5dbb4d62ee
    Port:           <none>
    Host Port:      <none>
    State:          Running
```

**--To open session inside pod to run any command**

**$$ kubectl exec -it myapp-pod -- /bin/bash**

```
root@(                    :/var/www/demo# kubectl exec -it myapp-pod -- /bin/bash
root@myapp-pod:/#
```

## 4-To delete it: -

**$$kubectl delete -f pod-definition.yaml**

## ➢ What is Replication Controller and how to create one: -

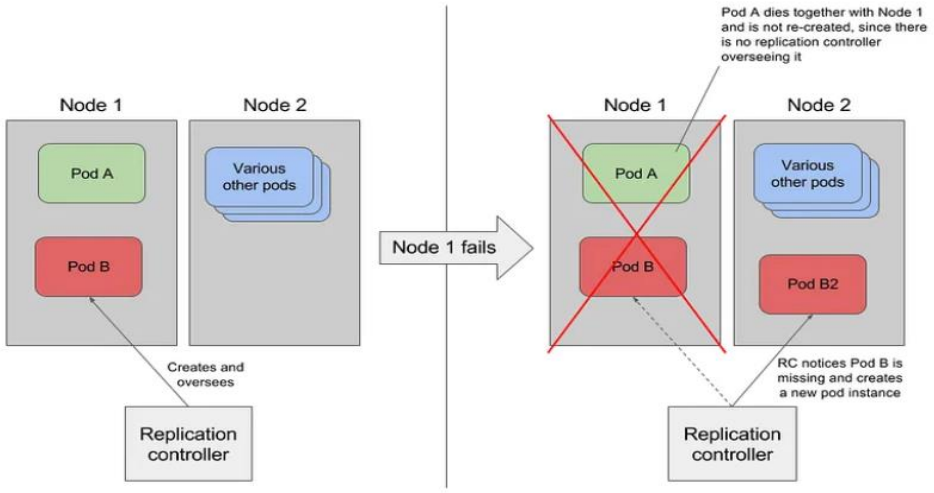



Figure 1 When a node fails, only pods backed by a replication controller are recreated

**Replication Controller: is a Kubernetes resource that ensures a pod (or multiple copies of the same pod) is always up and running. If the pod disappears for any reason (like in the event of a node disappearing from the cluster), the replication controller creates a new pod immediately.**

**Let's create one: -**

**1-create yaml file such as replication-controller-definition.yaml**

**$$nano replication-controller-definition.yaml**

**2-yaml contain: -**

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:
  replicas: 3
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
```

**This part of name and labels related to the ReplicationController**

**This part of name and labels related to the Pod**

**Number of pods is 3**

**Replicationcontroller uses labels to bind pod to its own replication controller.**

**3-Apply yaml file: -**

**$$kubectl apply -f replication-controller-definition.yaml**

**--To get status of the new pod**

**$$kubectl get pods -owide**

```
root@                  :/var/www/demo# kubectl get pods -owide
NAME             READY   STATUS    RESTARTS   AGE   IP              NODE                NOMINATED NODE   READINESS GATES
myapp-rc-88zlp   1/1     Running   0          43m   10.244.2.75     worker-node-1       <none>           <none>
myapp-rc-djhwr   1/1     Running   0          43m   10.244.1.168    worker-node-2       <none>           <none>
myapp-rc-rgj7z   1/1     Running   0          43m   10.244.1.169    worker-node-2       <none>           <none>
```

**$$kubectl get replicationcontroller**

```
root@(                    ı:/var/www/demo# kubectl get replicationcontroller
NAME       DESIRED   CURRENT   READY   AGE
myapp-rc   3         3         3       44m
```

**4-To delete it: -**

**$$kubectl delete -f replication-controller-definition.yaml**

## ➢ What is Replica Set and how to create one: -

**Replication Set: is the same concept of Replication controller but ReplicaSet is new technology of replica than replication controller.**

**Let's create one: -**

**1-create yaml file such as replication-set-definition.yaml**

**$$nano replica-set.yaml**

**2-yaml contain: -**

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-rs
  labels:
    app: myapp
    type: front-end
spec:
  replicas: 3
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  selector:
    matchLabels:
      type: front-end
```

--selector used as a label to match every replica with its pod by using type.

--using to monitor three instances if any instance of 3 down it will redeploy it again

**3-Apply yaml file: -**

**$$kubectl apply -f replica-set.yaml**

**--To get status of the new pod**

**$$kubectl get pods -owide**

```
root@(              h:/var/www/demo# kubectl get pods -owide
NAME              READY   STATUS    RESTARTS   AGE     IP              NODE            NOMINATED NODE   READINESS GATES
myapp-rs-7lhbd    1/1     Running   0          4m55s   10.244.1.170    worker-node-1   <none>           <none>
myapp-rs-lj8nk    1/1     Running   0          4m55s   10.244.2.76     worker-node-2   <none>           <none>
myapp-rs-s77l7    1/1     Running   0          4m55s   10.244.1.171    worker-node-2   <none>           <none>
```

## ➢ How to scale up Replica set: -

**Scale up of replication means increased number of pods.**

**Let's edit uppear file: -**

**1-Edit yaml file replication-set-definition.yaml by changing number of replicas from 3 to 6 .**

**$$nano replica-set.yaml**

**2-yaml contain: -**

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-rs
  labels:
    app: myapp
    type: front-end
spec:
  replicas: 6
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  selector:
    matchLabels:
      type: front-end
```

**3-Apply yaml file: -**

**$$kubectl apply -f replica-set.yaml**

**--To get status of the new pod**

**$$kubectl get pods -owide**

```
root@C                h:/var/www/demo# kubectl get pod -owide
NAME             READY   STATUS    RESTARTS   AGE   IP              NODE            NOMINATED NODE   READINESS GATES
myapp-rs-25f6t   1/1     Running   0          28s   10.244.2.79     worker-node-1   <none>           <none>
myapp-rs-btn6v   1/1     Running   0          28s   10.244.2.77     worker-node-2   <none>           <none>
myapp-rs-n2q2j   1/1     Running   0          28s   10.244.1.191    worker-node-2   <none>           <none>
myapp-rs-rh4kg   1/1     Running   0          28s   10.244.1.190    worker-node-2   <none>           <none>
myapp-rs-slhvp   1/1     Running   0          28s   10.244.2.78     worker-node-1   <none>           <none>
myapp-rs-znfns   1/1     Running   0          28s   10.244.1.192    worker-node-1   <none>           <none>
```

## 4-Another way: -

**$$kubectl scale –replicas=6 -f replica-set.yaml**

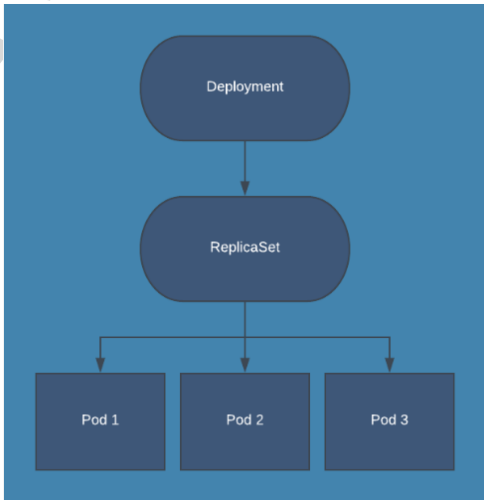
**Or:**

**$$kubectl edit replicaset myapp-rs** **>> myapp-rs is name of replicaset in yaml file**

## ➢ What is Deployment and how to create one: -

**Deployment: is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.**

| Deployments | ReplicaSet |
|---|---|
| High-level abstractions that manage replica sets. | A lower-level abstraction that manages the desired number of replicas of a pod. |
| Deployment manages a template of pods and uses replica sets to ensure that the specified number of replicas of the pod is running. | ReplicaSet only manages the desired number of replicas of a pod. |
| Deployment provides a mechanism for rolling updates and rollbacks of the application, enabling seamless updates and reducing downtime. | Applications must be manually updated or rolled back. |




**Let's create one: -**

**1-create yaml file such as deployment.yaml**

**$$nano deployment.yaml**

**2-yaml contain: -**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-rs
  labels:
    app: myapp
    type: front-end
spec:
  replicas: 3
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  selector:
    matchLabels:
      type: front-end
```

**3-Apply yaml file: -**

**$$kubectl apply -f deployment.yaml**

**--To get status of the new pod**

**$$kubectl get pods -owide**

```
root@                    :/var/www/demo# kubectl get pods -owide
NAME             READY   STATUS    RESTARTS   AGE     IP              NODE                NOMINATED NODE   READINESS GATES
myapp-rs-7lhbd   1/1     Running   0          4m55s   10.244.1.170                        <none>           <none>
myapp-rs-lj8nk   1/1     Running   0          4m55s   10.244.2.76                         <none>           <none>
myapp-rs-s77l7   1/1     Running   0          4m55s   10.244.1.171                        <none>           <none>
```

## ➢ Some Important command: -

**1-$$kubectl run nginx --image=nginx**

It will automatically create a pod by pulling nginx image from docker hub officially image then run the image in pod on worker node **without yaml file**.

**2-$$kubectl get all**

To get all resources (pod – deployment – replicaset - service - daemonset - etc) on default namespace (we will learn more about namespaces next slides)

```
root@                    h:/var/www/demo# kubectl get all
NAME                                 READY   STATUS    RESTARTS   AGE
pod/myapp-rs-7c4d4f7fc6-bjjm6        1/1     Running   0          18m
pod/myapp-rs-7c4d4f7fc6-f9j4d        1/1     Running   0          18m
pod/myapp-rs-7c4d4f7fc6-sbmff        1/1     Running   0          18m

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/jenkins      NodePort                    <none>                         3d20h
service/kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          26d

NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp-rs        3/3     3            3           18m

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-rs-7c4d4f7fc6        3         3         3       18m
```

**3-$$kubectl get all -A**

To get all resources in all namespaces

**4-$$kubectl get all -A -owide**

To get all resources in all namespaces with more information about location of all pods on which workernode , virtual IP of every pod

**5-$$kubectl exec -it myapp-rs-7c4d4f7fc6-bjjm6 -- /bin/bash**

It is used to open the terminal inside the pod to execute any command inside the pod.

**myapp-rs-7c4d4f7fc6-bjjm6 >>> name of the pod**

**6- $$ kubectl describe pod myapp-rs-7c4d4f7fc6-bjjm6**

```
root@                    :/var/www/demo# kubectl describe pod myapp-rs-7c4d4f7fc6-bjjm6
Name:             myapp-rs-7c4d4f7fc6-bjjm6
Namespace:        default
Priority:         0
Service Account:  default
Node:
Start Time:       Mon, 11 Dec 2023 13:18:21 +0200
Labels:           app=myapp
                  pod-template-hash=7c4d4f7fc6
                  type=front-end
Annotations:      <none>
Status:           Running
IP:               10.244.1.194
IPs:
  IP:           10.244.1.194
Controlled By:  ReplicaSet/myapp-rs-7c4d4f7fc6
```

**7-$$kubectl run redis --image=redis123 --dry-run=client -o yaml > redis-definition.yaml**

It is easy way to create yaml file with basic configuration **without run this pod (--dry-run=client).**

The output: -

```
root@                    :/var/www/demo# cat redis-definition.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: redis
  name: redis
spec:
  containers:
  - image: redis123
    name: redis
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

**8-$$ kubectl create deployment --image=nginx nginx --replicas=4 --dry-run=client -o yaml > nginx-deployment.yaml**

```
root@                    :/var/www/demo# cat nginx-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: nginx
  name: nginx
spec:
  replicas: 4
  selector:
    matchLabels:
      app: nginx
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}
```

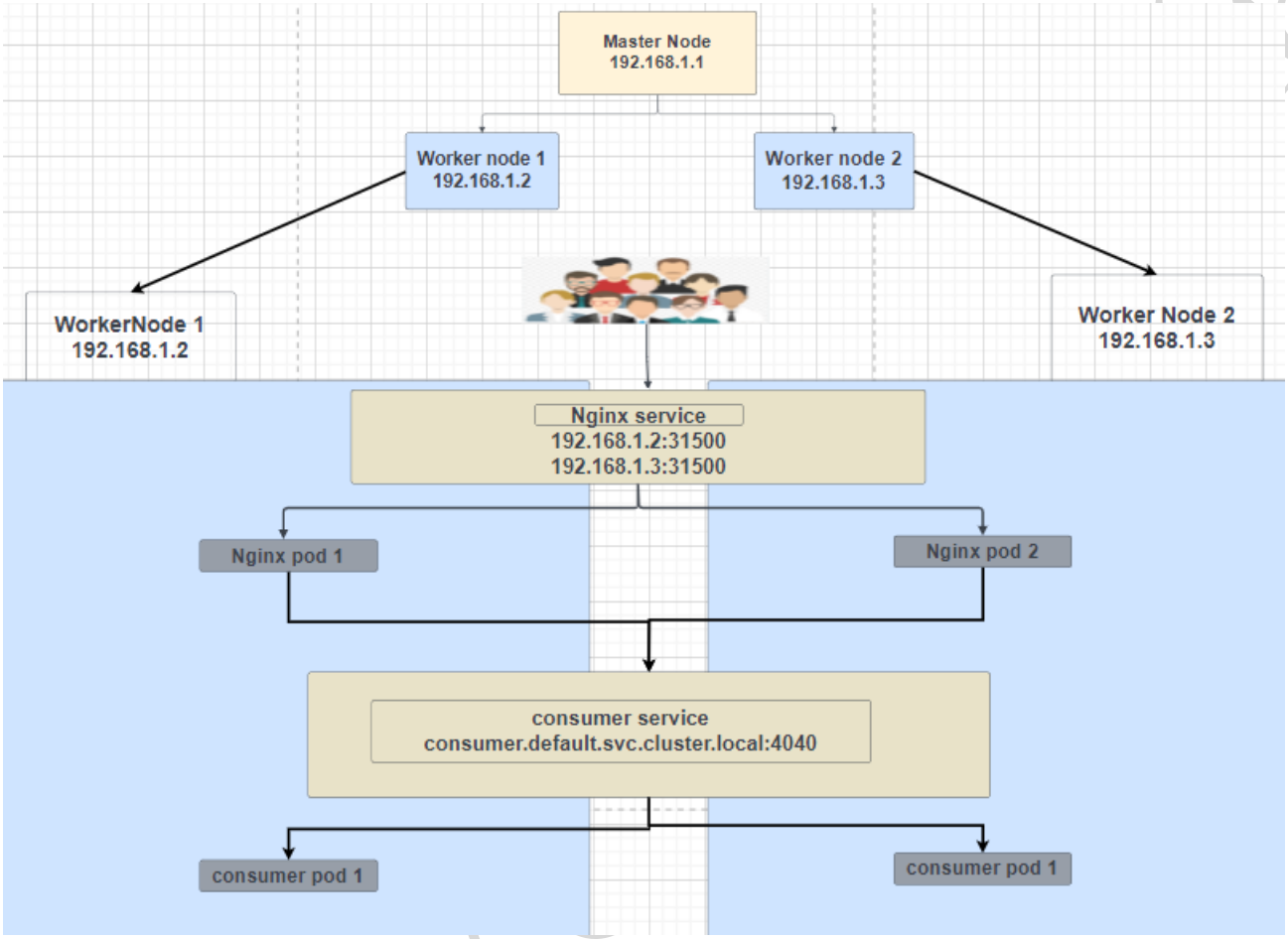
**9-To get logs of any pod**

**$$ kubectl logs consumer-55c97bfbc7-7nfn6 -n backend-dev**

**-n = namespace**

```
root@                2:~# kubectl logs consumer-55c97bfbc7-7nfn6 -n backend-dev
[00:01:41 INF] Application Started.
[00:01:41 INF] Application running on environment
[00:01:41 INF] Application version 1.0.0.0
[00:01:41 WRN] Storing keys in a directory '/root/.aspnet/DataProtection-Keys' that may not be persisted outside of the container. Protected data will be unav
ailable when container is destroyed.
[00:01:41 WRN] No XML encryptor configured. Key [redacted] may be persisted to storage in unencrypted form.
[00:47:35 ERR] Unauthorized
[15:52:07 ERR] PleaseLoginAgain
```

- **What is Service and how to create one: -**




**Kubernetes Service: enables communication between various components within and outside of the application, it helps us to connect applications together with other applications or users.**

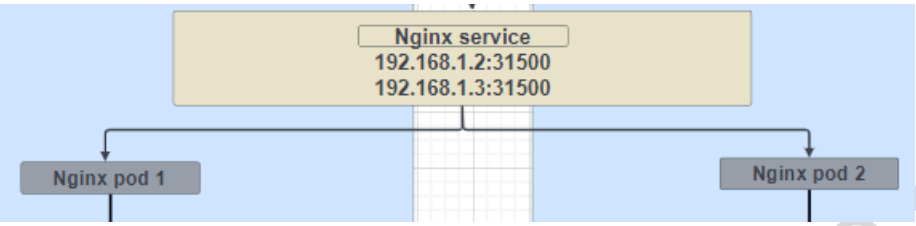
**--There are three types of main service: -**

**1- NodePort-Service: Exposes the service on a static port on each node's IP. It makes the service accessible externally at the specified node port.**

**2- ClusterIP-Service: The default service type, which provides a cluster-internal IP address. It is used for communication between different parts of an application within the cluster.**

**3- LoadBalancer-Service: Exposes the service externally using a cloud provider's load balancer. The external IP is provisioned, and traffic is distributed to the service.**

## 1-- NodePort-Service:



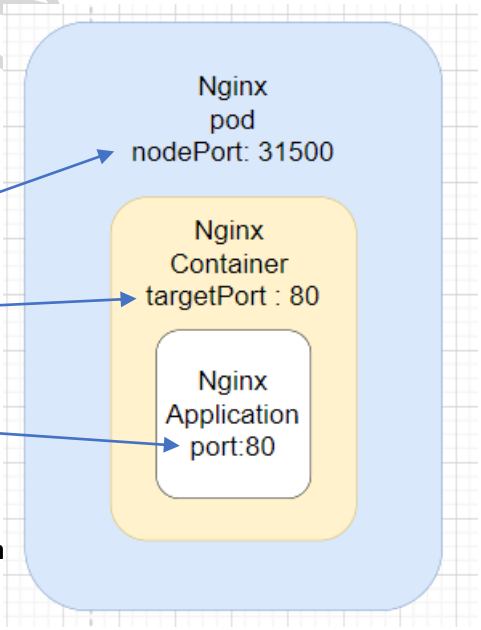

**After creating deployment of nginx as shown Pg:17, We are going to create service (Type: NodePort) to make those pods accessible from out the world.**

**Let's create one: -**

**1-create yaml file such as nginx-service.yaml.**

```
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  ports:
  - targetPort: 80
    port: 80
    nodePort: 31500
  selector:
    type: front-end
    app: myapp
  type: NodePort
```

**- It must be same as labels on Yaml deployment to bind service with pods need to route to.**




**3-Apply yaml file: -**

**$$kubectl apply -f nginx-service.yaml.**

**--To get status of the new pod**

**$$kubectl get all -owide**

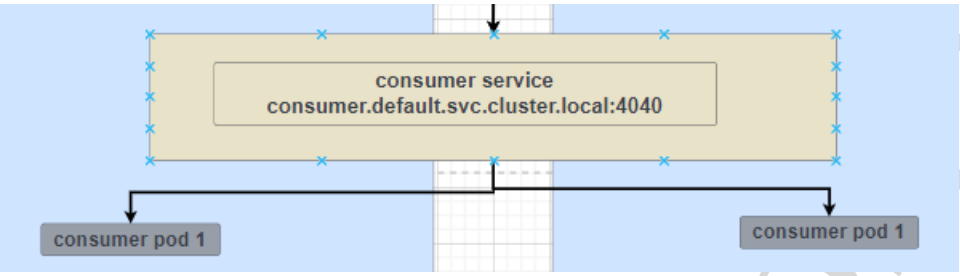
```
root@                    :/var/www/demo# kubectl get all -owide
NAME                                 READY   STATUS    RESTARTS   AGE     IP             NODE                    NOMINATED NODE   READINESS GA
TES
pod/myapp-rs-7c4d4f7fc6-f4zgr        1/1     Running   0          9m39s   10.244.2.91                            <none>           <none>
pod/myapp-rs-7c4d4f7fc6-wwbnn        1/1     Running   0          9m39s   10.244.1.222                           <none>           <none>

NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   SELECTOR
service/kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP        30d   <none>
service/nginx-service   NodePort    10.96.210.200   <none>        80:31500/TCP   11m   app=myapp,type=front-end

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS        IMAGES   SELECTOR
deployment.apps/myapp-rs      2/2     2            2           9m39s   nginx-container   nginx    type=front-end

NAME                                    DESIRED   CURRENT   READY   AGE     CONTAINERS        IMAGES   SELECTOR
replicaset.apps/myapp-rs-7c4d4f7fc6     2         2         2       9m39s   nginx-container   nginx    pod-template-hash=7c4d4f7fc6,type=fr
ont-end
```

- **Now <u>very Important example</u> to understand how NodePort working, we have demo application called consumer, this application consisting of 2 pods on two worker-nodes working on port 4040 and I will expose on port 31040.**



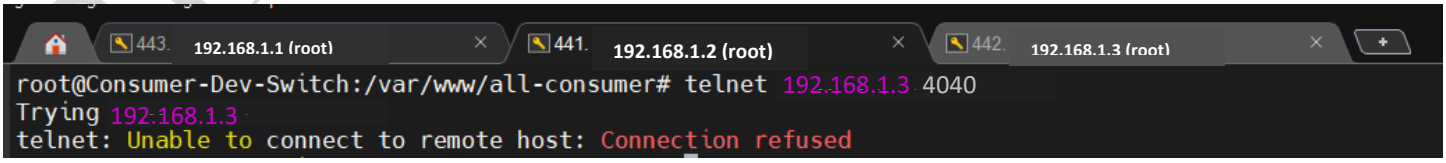

**<u>Let's create one: -</u>**

**1-create yaml file such as consumer-service.yaml and apply it.**

```
---
apiVersion: v1
kind: Service
metadata:
  name: consumer
spec:
  ports:
  - targetPort: 4040
    port: 4040
    nodePort: 31040
  selector:
    app: consumer
  type: NodePort
```

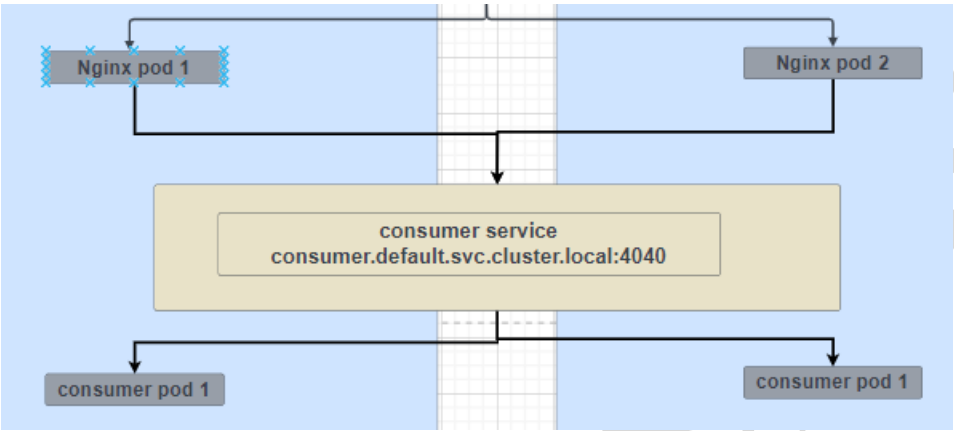
**2- to check connection by telnet.**


**What about check telnet with port 4040**




**So, it is important to know that NodePort service used to export our application out of cluster using nodePort only, but what is targetport used to? We will see next slide.**

- Now <u>very Important</u> to understand how to connect between two pods (nginx application connect to consumer application) [nginx pod ➡ consumer service ➡ consumer pod].




**1- go inside nginx pod (any pod): -**

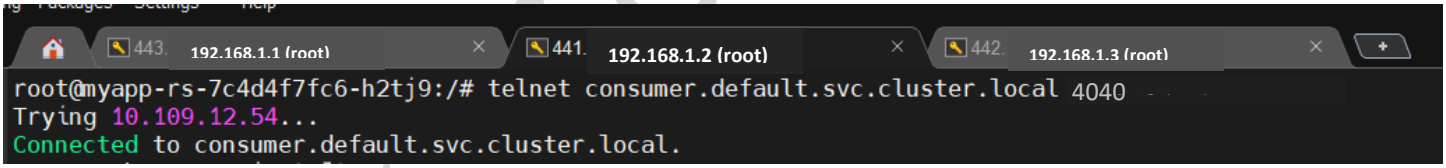
**$$kubectl exec -it myapp-rs-7c4d4f7fc6-h2tj9 -- /bin/bash**

**2- install telnet package: -**

**$$ apt-get update && apt-get install telnet**

**3- check connection from inside nginx: -**

**$$telnet consumer.default.svc.cluster.local 4040**




**What is consumer.default.svc.cluster.local mean ???**

**consumer: name of service Pg:22**
**default: is the default namespace (we will learn more about namespace next slides)**
**svc: referring to Kubernetes services.**
**cluster.local: default cluster**

## Let's have a brief about NodePort service: -

**- The range of nodeport must be between 30000 and 32767.**
**- To connect between out of cluster and pod inside cluster using any IP of worker node and nodePort.**
**- To connect between to pod inside cluster by using name of service and target port.**

## 2-- ClusterIP-Service:

**It is same concept of NodePort service, so it just used to connect between two pod inside cluster only.**

**Let's create one: -**

**1-create yaml file such as ClusterIP-service.yaml and apply it.**

```
---
apiVersion: v1
kind: Service
metadata:
  name: consumer
spec:
  ports:
  - targetPort: 4040
    port: 4040
  selector:
    app: consumer
  type: ClusterIP
```

**2- go inside nginx pod (any pod): -**

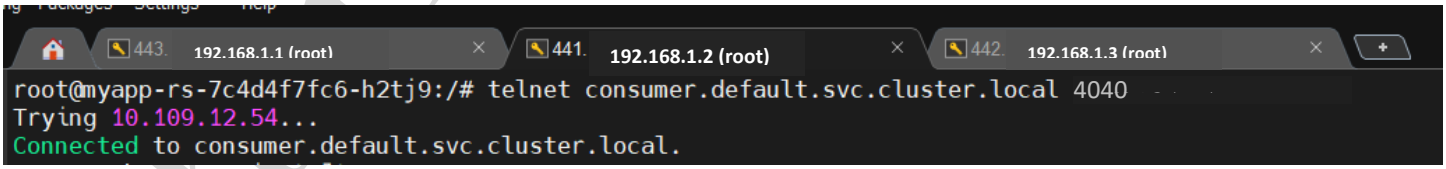
**$$kubectl exec -it myapp-rs-7c4d4f7fc6-h2tj9 -- /bin/bash**

**3- install telnet package: -**

**$$ apt-get update && apt-get install telnet**

**4- check connection from inside nginx: -**

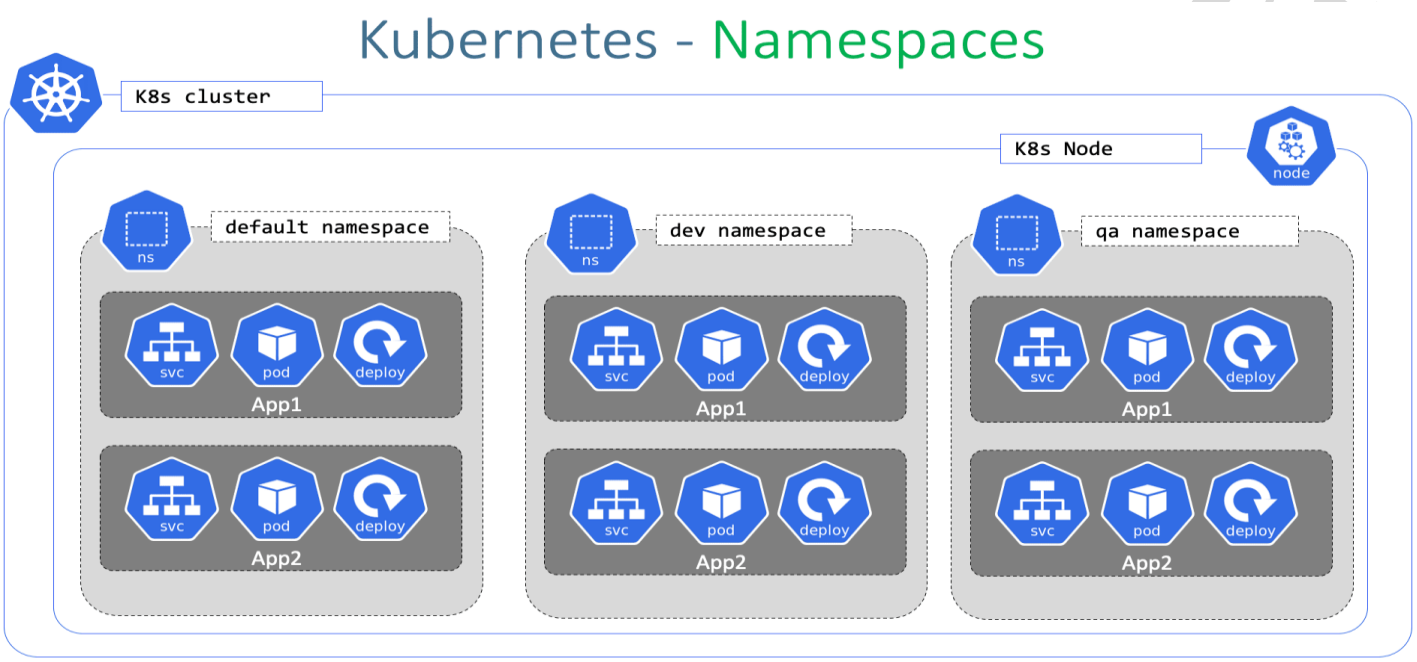
**$$telnet consumer.default.svc.cluster.local 4040**




## 3-- LoadBalancer-Service (working with cloud only):

**For VMware (on premise) we use NodePort-Service to make loadbalancer**
**For cloud AWS , AZURE we use LoadBalancer-service**

➢ **What is namespace and how to create one: -**




**- by default, any pods or service or any other components you created without specify namespace will create in [default namespace] as example Pg: 20**

**- Namespaces: used to divide PODS under difference various Name to avoid accidentally execute modify on wrong production PODS.**

**let's start: -**

**we have namespace > (backend) which consist of consumer app.**
**and namespace > (gateway) which consist of nginx.**

**1—create a namespace called backend.**
**$$ kubectl create namespace backend**

**2--modify yaml file of consumer app to add namespace then apply it.**

```
  GNU nano 4.8                                    consumer-pod.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: consumer
  namespace: backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: consumer
  template:
    metadata:
      labels:
        app: consumer
```

**3 -- modify the service of consumer app to add namespace then apply it.**

```
---
apiVersion: v1
kind: Service
metadata:
  name: consumer
  namespace: backend
spec:
  ports:
  - targetPort: 4040
    port: 4040
    nodePort: 31040
  selector:
    app: consumer
  type: NodePort
```

**3 – create a namespace called gateway.**

**$$ kubectl create namespace gateway**

**4-- modify yaml file of nginx app to add namespace then apply it.**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-rs
  namespace: gateway
  labels:
    app: myapp
    type: front-end
spec:
  replicas: 2
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  selector:
    matchLabels:
      type: front-end
```

**5-- modify the service of nginx to add namespace then apply it.**

```
---
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: gateway
spec:
  ports:
  - targetPort: 80
    port: 80
    nodePort: 31500
  selector:
    type: front-end
    app: myapp
  type: NodePort
```

**6—let's get all about namespace backend**

**$$kubectl get all -owide -n backend**

**-n = namespaces**

```
root@                    :/var/www/demo# kubectl get all -owide -n backend
NAME                            READY   STATUS    RESTARTS   AGE     IP             NODE                   NOMINATED NODE   READINESS GA
TES
pod/consumer-76fcb5fd44-g87st   1/1     Running   0          2m14s   10.244.1.230                          <none>           <none>
pod/consumer-76fcb5fd44-w6kl7   1/1     Running   0          2m14s   10.244.2.98                           <none>           <none>

NAME               TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE     SELECTOR
service/consumer   NodePort   10.103.251.88   <none>                         2m14s   app=consumer

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS   IMAGES                                                SELECTO
R
deployment.apps/consumer   2/2     2            2           2m14s   consumer                                                           app=con
sumer

NAME                                  DESIRED   CURRENT   READY   AGE     CONTAINERS   IMAGES                                                  S
ELECTOR
replicaset.apps/consumer-76fcb5fd44   2         2         2       2m14s   consumer                                                             a
pp=consumer,pod-template-hash=76fcb5fd44
```

**7-- let's get all about namespace gateway**

**$$kubectl get all -owide -n gateway**

```
root@                    /var/www/demo# kubectl get all -owide -n gateway
NAME                            READY   STATUS    RESTARTS   AGE   IP             NODE                   NOMINATED NODE   READINESS GATE
S
pod/myapp-rs-7c4d4f7fc6-ccmx8   1/1     Running   0          25m   10.244.1.228                          <none>           <none>
pod/myapp-rs-7c4d4f7fc6-jcf9h   1/1     Running   0          25m   10.244.2.95                           <none>           <none>

NAME                    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   SELECTOR
service/nginx-service   NodePort   10.96.217.164   <none>        80:31500/TCP   25m   app=myapp,type=front-end

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS        IMAGES   SELECTOR
deployment.apps/myapp-rs   2/2     2            2           25m   nginx-container   nginx    type=front-end

NAME                                  DESIRED   CURRENT   READY   AGE   CONTAINERS        IMAGES   SELECTOR
replicaset.apps/myapp-rs-7c4d4f7fc6   2         2         2       25m   nginx-container   nginx    pod-template-hash=7c4d4f7fc6,type=fron
t-end
```

- **How to connect between two pods in different namespaces?**

**1- go inside nginx pod (any pod): -**

**$$ kubectl exec -it myapp-rs-7c4d4f7fc6-ccmx8 -n gateway -- /bin/bash**

**-n = namespaces**

**2- install telnet package: -**

**$$ apt-get update && apt-get install telnet**

**3- telnet from nginx pod to consumer pod: -**

**$$ telnet consumer.backend.svc.cluster.local 4040**

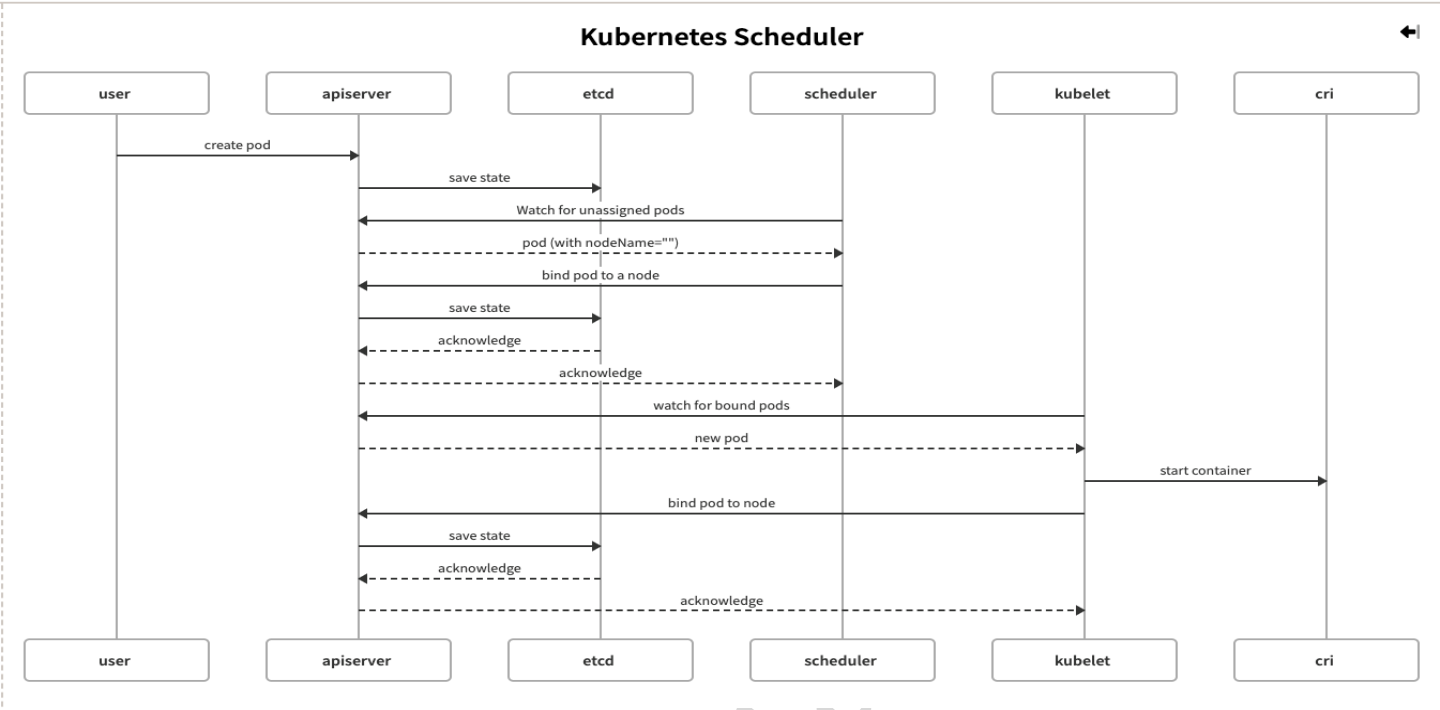
```
root@myapp-rs-7c4d4f7fc6-ccmx8:/# telnet consumer.backend.svc.cluster.local 4040
Trying 10.103.251.88...
Connected to consumer.backend.svc.cluster.local.
```

- **To specify every Namespace with specific resources: -**

**Creart pod-resources.yml and apply it:-**

```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: compute-quota
 namespace: backend
spec:
 hard:
  pods: "10"
  requests.cpu: "4"
  requests.memory: 5Gi
  limits.cpu: "10"
  limits.memory: 10Gi
```

## ➢ What is scheduling and how to create pod on specific node: -

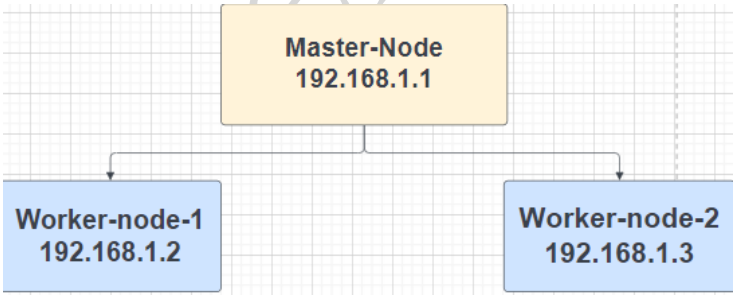



**<u>Scheduling:</u> scheduler identifies the right node to place a pod on, based on the pod's resource requirements and worker node capacity.**

**- By default, scheduling arranged by [Kube-scheduler] components in master node.**

**- You can manually schedule a pod on a specific node.**

**- Create deployment.yaml then apply it.**




```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-rs
  namespace: gateway
  labels:
    app: myapp
spec:
  replicas: 2
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
    spec:
      containers:
      - name: nginx-container
        image: nginx
      nodeName: worker-node-2
  selector:
    matchLabels:
      app: myapp
```

**- We can check it.**

```
root@                  /var/www/demo# kubectl get pods -n gateway -owide
NAME                     READY   STATUS    RESTARTS   AGE     IP             NODE                NOMINATED NODE   READINESS GATES
myapp-rs-7f5cf4d89c-gk8gx   1/1     Running   0          3m41s   10.244.2.126   worker-node-2       <none>           <none>
myapp-rs-7f5cf4d89c-q6h64   1/1     Running   0          3m41s   10.244.2.125   worker-node-2       <none>           <none>
```

## ➢ What is Labels and Selectors and how to create one: -

**Labels and Selectors: Are a standard method to group things together.**

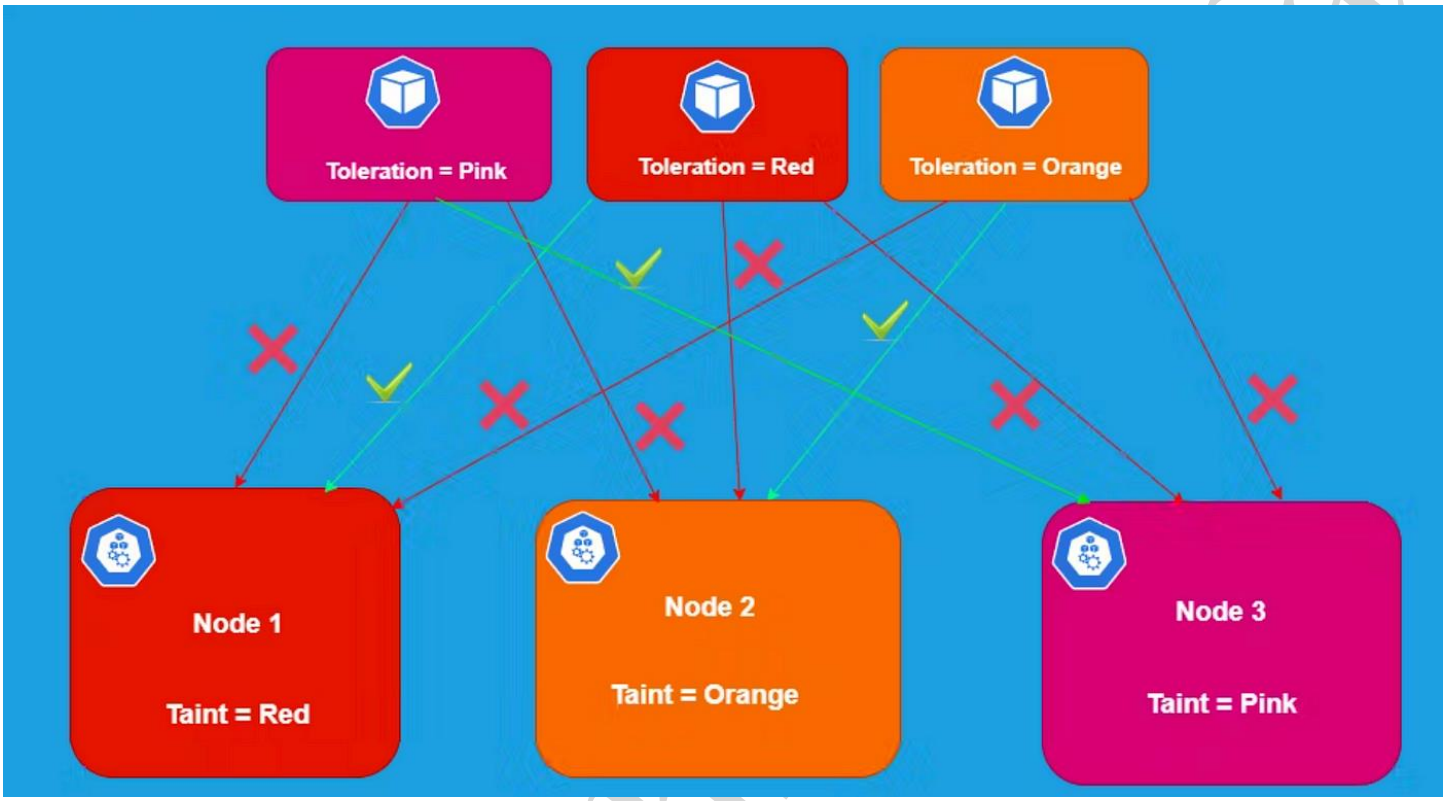
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-rs
  namespace: gateway
  labels:
    app: myapp
    type: front-end
spec:
  replicas: 2
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  selector:
    matchLabels:
      type: front-end
```

**you can search in pods by labels: -**

**$$kubectl get pods --selector app=myapp**

```
root@                  :/var/www/demo# kubectl get pods --selector app=myapp -n gateway
NAME                     READY   STATUS    RESTARTS   AGE
myapp-rs-7f5cf4d89c-gk8gx   1/1     Running   0          46m
myapp-rs-7f5cf4d89c-q6h64   1/1     Running   0          46m
```

## ➢ What is Taints and Tolerations and how to create one: -




**Taints and Tolerations:** **It is the relationship between pods and node, how you can restrict pod on specific node.**

**- As you see in the upper figure, we have 3 pods with 3 Toleration and 3 nodes with 3 Taints.**
**- Any node with Taint will not accept any pod except pod with Toleration same value of Taint.**
**- Important Note: Any pod with Toleration can scheduler on node with Taint same value of Toleration, also it can scheduler on node which have NO Taint, to avoid that we will make something called node affinity we will see it next slides.**
**- Why are pods automatically not deployed on master node?**
**because a Taint is set automatically on master node that prevents any pods to deployed on.**

```
root@                :/var/www/demo# kubectl describe node  Master-node          | grep Taint
Taints:          node-role.kubernetes.io/control-plane:NoSchedule
```

**how to make node as a Taints ?**
**$$kubectl taint nodes node-1 key=Red:taint-effect**
**- There are three types of taint-effect: -**
**NoSchedule : Which means pods will not be scheduled on the node except pod with Toleration same value of Taint of node.**
**PreferNoSchedule : System will try to avoid placing a pod on node but it is not guaranteed.**

**NoExecute: The new pods will not be scheduled on the node and existing pods on the node, if any, will be evicted, if they do not tolerate the taint in case that those pods are applied to node before we Taints the node.**

**1- Assigns node (worker-node-1) with Taints = [myapp=red]**

**$$ kubectl taint nodes worker-node-1 myapp=red:NoSchedule**

```
root@                    :/var/www/demo# kubectl taint nodes    worker-node-1       myapp=red:NoSchedule
node/ worker-node-1          tainted
```

**2- Create pod have Toleration = [myapp=red] then apply it: -**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-rs
  labels:
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
      tolerations:
      - key: "myapp"
        operator: "Equal"
        value: "red"
        effect: "NoSchedule"
  selector:
    matchLabels:
      type: front-end
```

**>> Equal mean that Toleration = [myapp = red]**

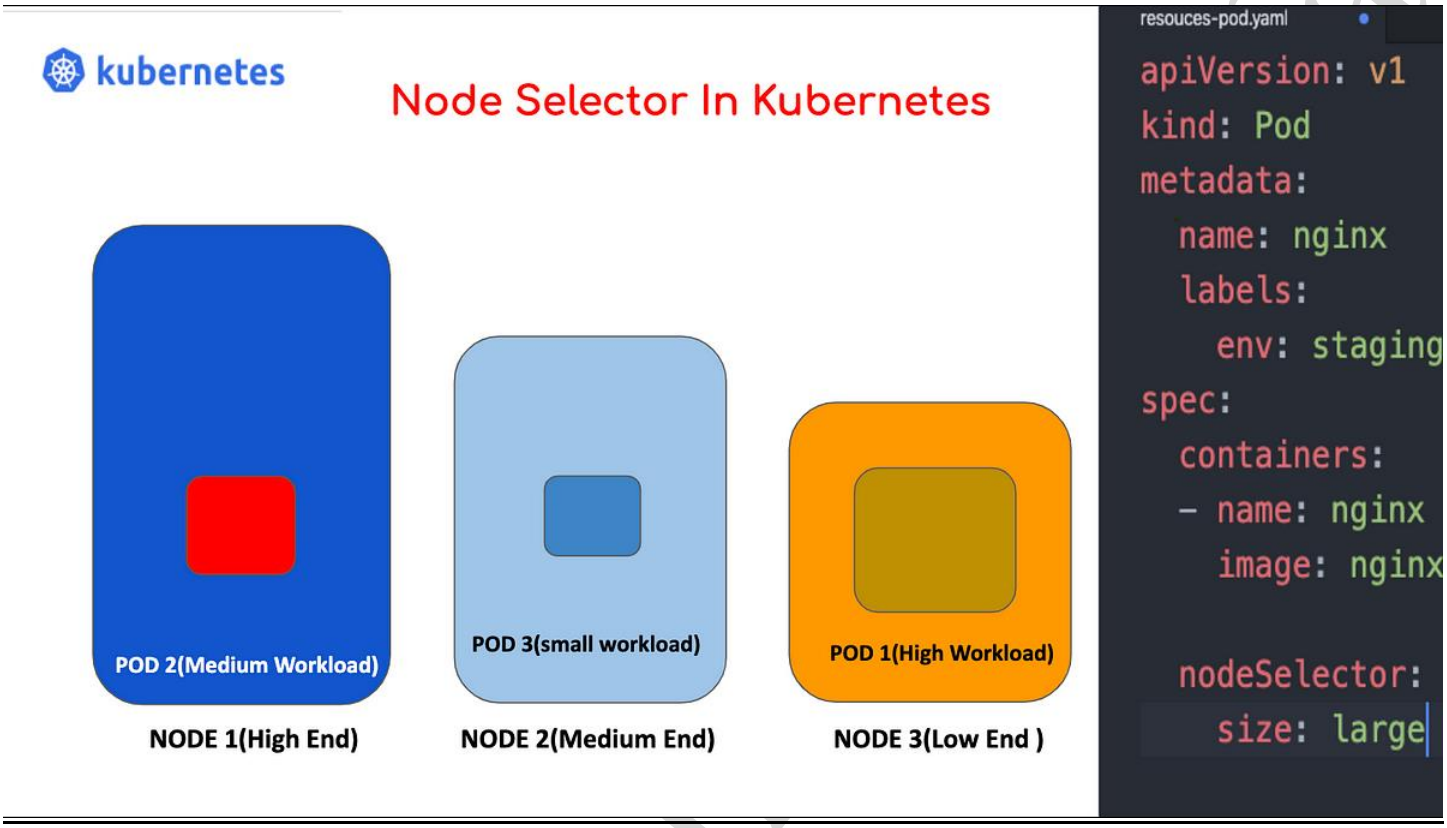
**3- IF you want to remove Taints from node (worker-node-1): -**

**$$ kubectl taint nodes worker-node-1 myapp=red:NoSchedule-**

**Just add (-) at end of command**

```
root@                    /var/www/demo# kubectl taint nodes                    . myapp=red:NoSchedule-
node/                        untainted
root@                    :/var/www/demo#
```

- **What is Node Selectors and how to create one: -**




**<u>Node Selectors:</u> if I have container with application need a high node resource such as RAM: 64 , so we used Node Selectors to assign the node with high resources as a node with labels size = large , so we can edit application .yaml to put label size = large , so it make this node deployed on node of high resources**

**1- Assign node as a Size = large: -**

**$$ kubectl label nodes woker-node-1 size=Large**

**2- Edit yaml file with node selector and apply it: -**

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: nginx-container
   image: nginx
 nodeSelector:
   size: Large
```

## ➢ What is Node Affinity and how to create one: -

**<u>Node Affinity:</u> it is the same as node selector but Node Affinity more complex which give you more feature.**

**1- Assign node as a Size = large: -**

**$$ kubectl label nodes worker-node-1 size=Large**

**2-Edit yaml file to deploy on pod with label Large then apply it: -**

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: nginx-container
   image: nginx
 affinity:
  nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
    - matchExpressions:
      - key: size
        operator: In
        values:
        - Large
```

**Or edit yaml file to deploy on pod with label Large or medium then apply it: -**

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: nginx-container
   image: nginx
 affinity:
  nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
    - matchExpressions:
      - key: size
        operator: In
        values:
        - Large
        - Medium
```

**Or edit yaml file to deploy on pod with label small then apply it: -**

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: nginx-container
   image: nginx
 affinity:
  nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
    - matchExpressions:
      - key: size
        operator: NotIn
        values:
        - Small
```

**Example to deploy pod on any node have label size without value just key only: -**

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: nginx-container
   image: nginx
 affinity:
  nodeAffinity:
   requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
    - matchExpressions:
      - key: size
        operator: Exists
```

- **What could happen if there is node with Labels size and someone deleted label size from node which contain pod with nodeAffinity ?**

**- The answer in requiredDuringSchedulingIgnoredDuringExecution**

**- There are Three type: -**

**1- requiredDuringSchedulingIgnoredDuringExecution >>> pod must find node with same labels if not the pod will not deploy.**

**>>>> IgnoredDuringExecution : it mean that any change on node like ( delete label on node ) will not affect after pod had been deployed**

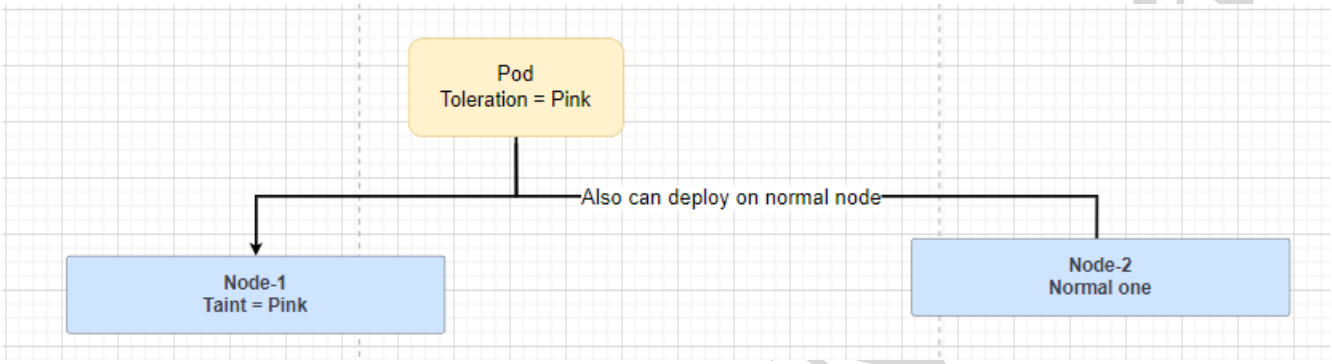
**2- preferredDuringSchedulingIgnoredDuringExecution >>> pod no need to find node with same labels so the pod will deploy on another node.**

**stage one (DuringScheduling)**: which pod does not exist and created for first time.

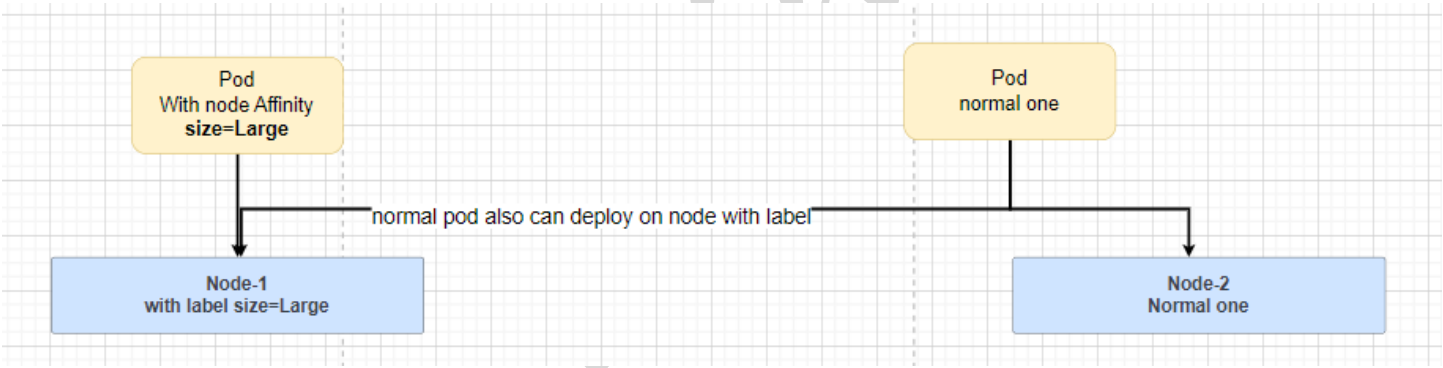
**stage two (DuringExecution):** pod exist and there is a change on environment which affect node affinity such as ( delete label on node ) which already contain pods with node affinity.

## ➢ Taints and Tolerations vs Node Affinity: -

**Taints and Tolerations: as you know the Taints pod will deploy on the Tolerations node but also it can deploy on normal node.**




**Node Affinity: as you know the pod with node Affinity will deploy on Node with same label but also the normal pod can be deployed on node with label.**

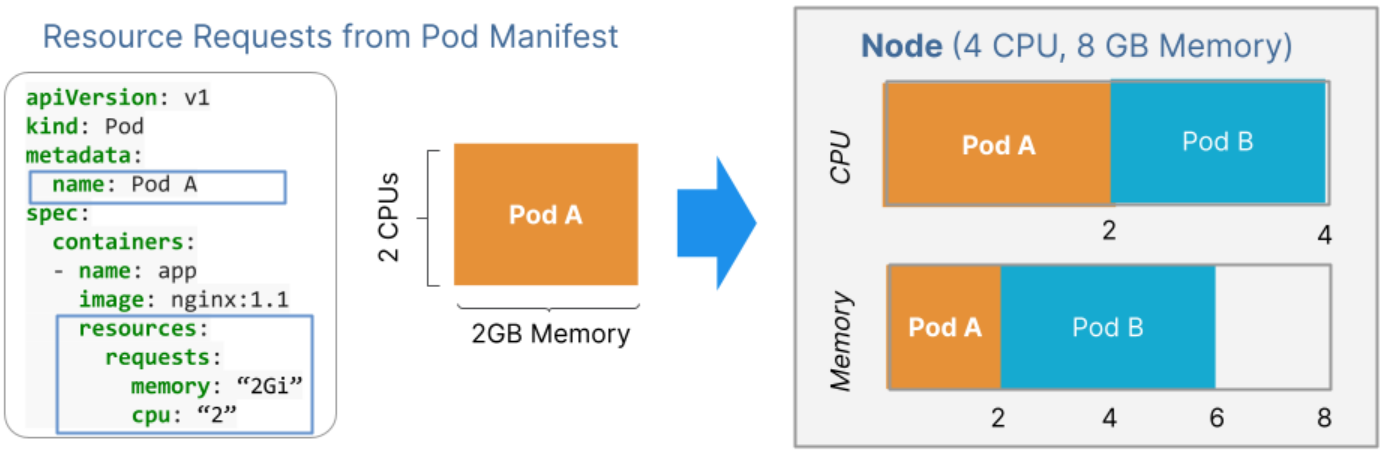



**so if you want to specific pod deployed on specific node it needed combination between Taints and Tolerations / and Node Affinity**

**or we have easy way by adding nodeName to specify node to deploy on: -**

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: nginx-container
   image: nginx
 nodeName: Node-1
```

## ➢ What is Resource Requirements and Limits and how to create one: -




**Resource Requirements and Limits: we are going to specify resources for every POD.**

**Note that by default: pod has no limit of resources as increase on load of application the resource will be increase so it led to suffocate another pod.**

**<u>Let's create one: -</u>**

**1- create yaml file of pod then apply it:-**

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: nginx-container
  image: nginx
  ports:
   - containerPort: 8080
  resources:
   requests:
    memory: "4Gi"          >>>>>
    cpu: 2                 >>>>>
```

**1Gi = 1073741824 bytes / 1Mi = 1048576 bytes / 1Ki = 1024 bytes**
**it can be also 100m or 0.1 AS 1 CPU = 1000m**

**2- Also, we have another example to limit maximum utilization.**

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
spec:
 containers:
 - name: nginx-container
   image: nginx
   ports:
    - containerPort: 8080
   resources:
    requests:
     memory: "4Gi"
     cpu: 2
    limits:
     memory: "8Gi"
     cpu: 4
```

**NOTE: As increase on load the pod never take more than 4 cpu, even it needs 2 more cpu it will not take more than 2 CPU**

**NOTE: AS increase on load the pod will take more than 8Gi RAM but it will be terminated with error OOM (out of memory)**

**So the best practice of (Resource Requirements and Limits) is adding requests without limits BUT you must (put requests for all pods) because pod with no requests may be led to consume all resource:-**

**3- We can Create a LimitRange it is something like service to specify a default for all pods on my cluster of using CPU and RAM: -**

**NOTE this LimitRange will not affect pod that is created it just affect new pods!**

**Let's create one: -**

**- Limit for CPU**

```
apiVersion: v1
kind: LimitRange
metadata:
 name: cpu-resource-constraint
spec:
 limits:
 - default:
    cpu: 500m
   defaultRequest:
    cpu: 500m
   max:
    cpu: "1"
   min:
    cpu: 100m
   type: Container
```

**- Limit for RAM**

```
apiVersion: v1
kind: LimitRange
metadata:
 name: memory-resource-constraint
spec:
 limits:
 - default:
    memory: 1Gi
   defaultRequest:
    memory: 1Gi
   max:
    memory: 1Gi
   min:
    memory: 1Gi
   type: Container
```

**4- Also we can limit all of cpu and ram that is used by all pod (sum of ram and cpu for all pods together) for all namespace:-**

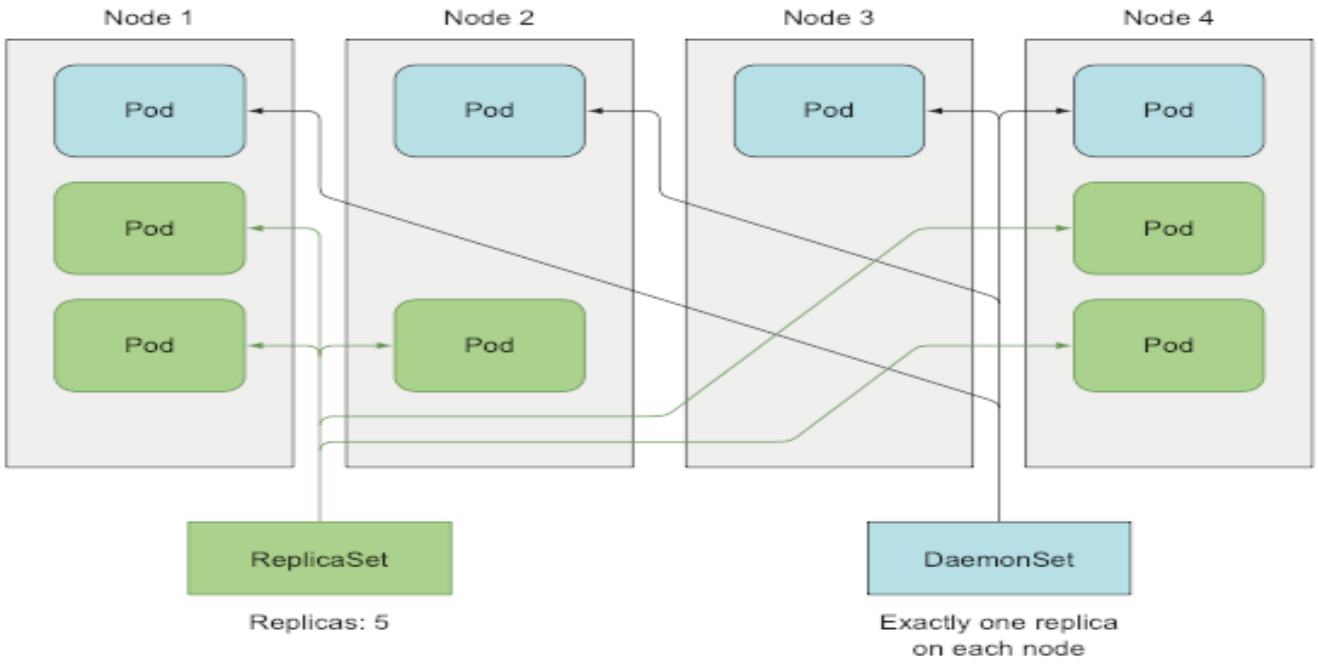
```
apiVersion: v1
kind: ResourceQuota
metadata:
 name: my-resource-quota
spec:
 hard:
  requests.cpu: 4
  requests.memory: 4Gi
  limit.cpu: 10
  limits.memory: 10Gi
```

**Just for your knowledge: -**

**-To extract the pod definition in YAML format to a file using the command**

**$$kubectl get pod webapp -o yaml > my-new-pod.yaml**

- **What is DaemonSets and how to create one: -**




**DaemonSets: Used to deploy one copy of your specific pod on each node in your cluster, If you joined new node in cluster by default, DaemonSets going to deploy specific pod just like monitoring agent**

**Let's say you want to deploy a monitoring agent as a pod or log collector as a pod on each of your nodes in the cluster, to monitor your cluster better, A DaemonSets is perfect for that.**

**what is different between DaemonSets and ReplicaSet ?**

**_____**

**-DaemonSets deployed one copy of your specific pod on each node in your cluster.**
**-ReplicaSet can be deployed 2 or 3 copy your pod on same node in your cluster.**

**Example of DaemonSet: -**

**kube-proxy is deployed on each node of the cluster As a DaemonSets**
**Flannel is deployed on each node of the cluster As a DaemonSets**

**Let's create one: -**

1- **create yaml file (daemonset.yaml) of pod then apply it:-**

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: monitoring-daemon
spec:
  selector:
    matchLabels:
      app: monitoring-agent
  template:
    metadata:
      labels:
        app: monitoring-agent
    spec:
      containers:
      - name: monitoring-agent
        image: monitoring-agent
```

2- **You can check it.**

**$$ kubectl get daemonsets --all-namespaces**

```
root@                  :/var/www/demo# kubectl get daemonsets --all-namespaces
NAMESPACE                NAME                                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
cattle-monitoring-system   rancher-monitoring-prometheus-node-exporter   3         3         3       3            3           kubernetes.io/os=linux   2d17h
default                  monitoring-daemon                           2         2         0       2            0           <none>                   6s
kube-flannel             kube-flannel-ds                             3         3         3       3            3           <none>                   2d18h
kube-system              kube-proxy                                  3         3         3       3            3           kubernetes.io/os=linux   2d18h
```

- **Logging & Monitoring.**

### ➢ Kubernetes Dashboard Solution: -

**-- Kubernetes does not come with a full-featured built-in monitor solution, so you can use open-source tool like Prometheus, elastic, Datadog, Dynatrace.**

**-- Kubernetes is going to implement a tool called (heapster) but it is deprecated.**

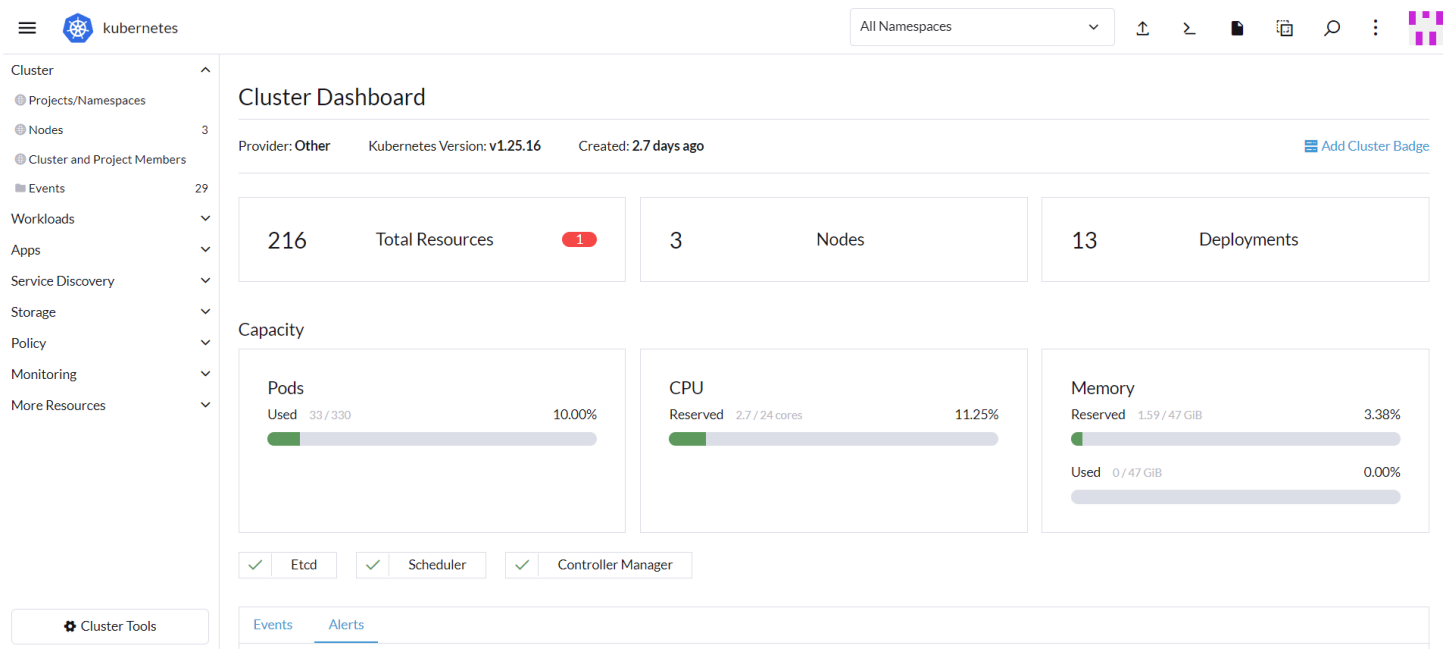
**-- as you know Kubernetes monitoring pods in case that one is down it automatic run new one, that is happened because of Kubelet have small agent called cAdvisor.**

**-- cAdvisor is responsible for retrieving performance metrics from pod and exposing them through kubelet API to (metrics server).**

**-- every cluster Kubernetes have (metrics server) which responsible for retrieves metrics from each of nodes and pods , but it cannot store the metrics data on the disk so you cannot see historical performance data , so you must implement open source tool.**

### ➢ What is Rancher: -

**Rancher is a complete software stack for teams adopting containers. It addresses the operational and security challenges of managing multiple Kubernetes clusters, while providing DevOps teams with integrated tools for running containerized workloads.**



## ➢ How to Implement rancher: -

**1- Install Docker CLI on any WorkerNode of your cluster for example server (192.168.1.2): -**

**$$ sudo apt-get install docker-ce docker-ce-cli**

**2- We will run rancher as a docker image (Containerized App) separated from kubernetes cluster: -**

**$$ docker run -d --restart=unless-stopped -p 80:80 -p 443:443 --privileged rancher/rancher:latest**

**3- To get password for user: admin run below command on same server:-**

**$$ docker logs 9503e066bb38 2>&1 | grep "Bootstrap Password:"**

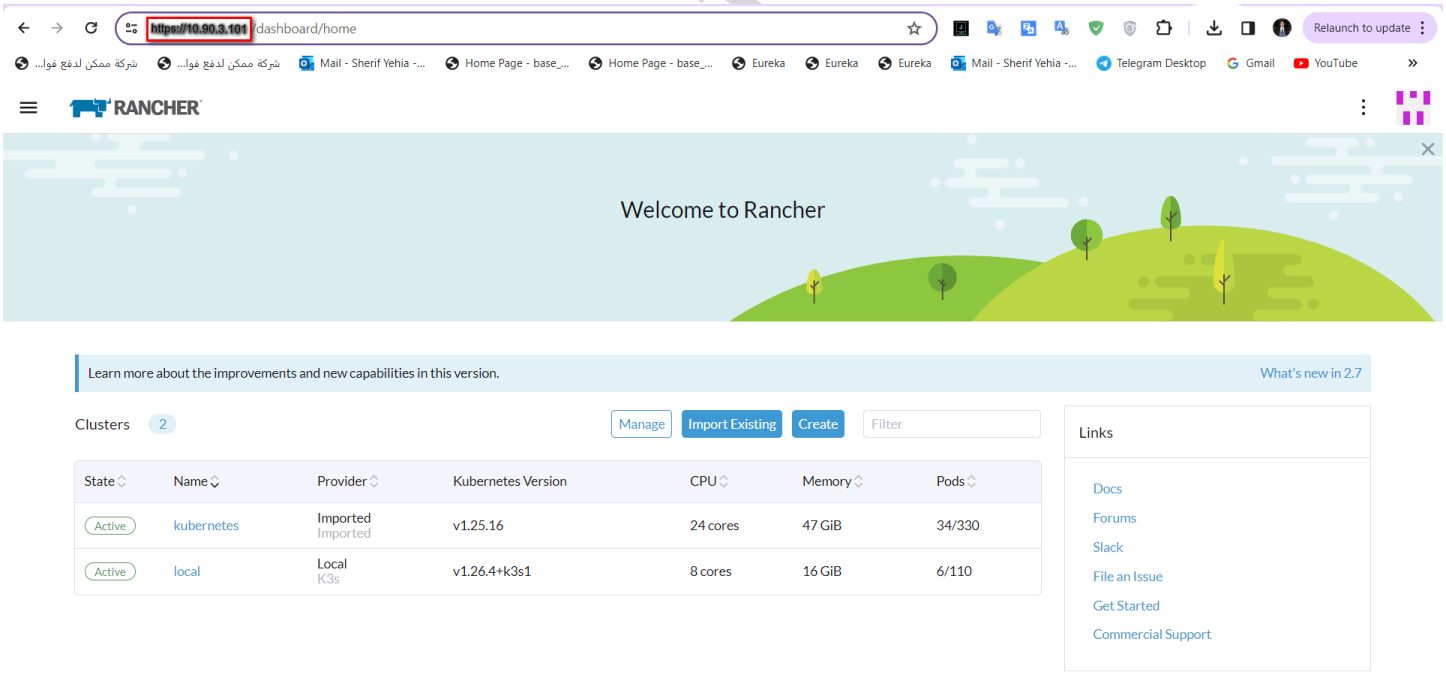
**9503e066bb38 >>> is the container ID**

**To get the container ID run below.**

**$$docker ps**

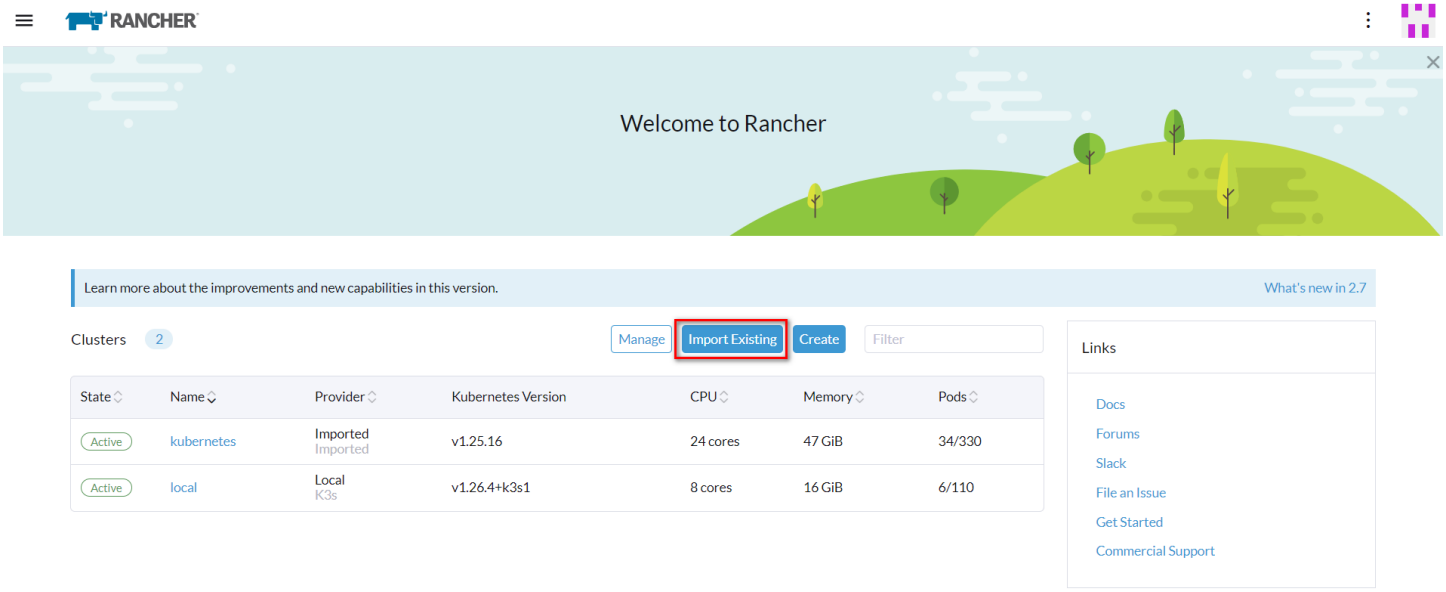
**Then take the password result from command docker logs and add it in dashboard.**

**4- From your PC go to browser and access rancher using https://192.168.1.2 .**

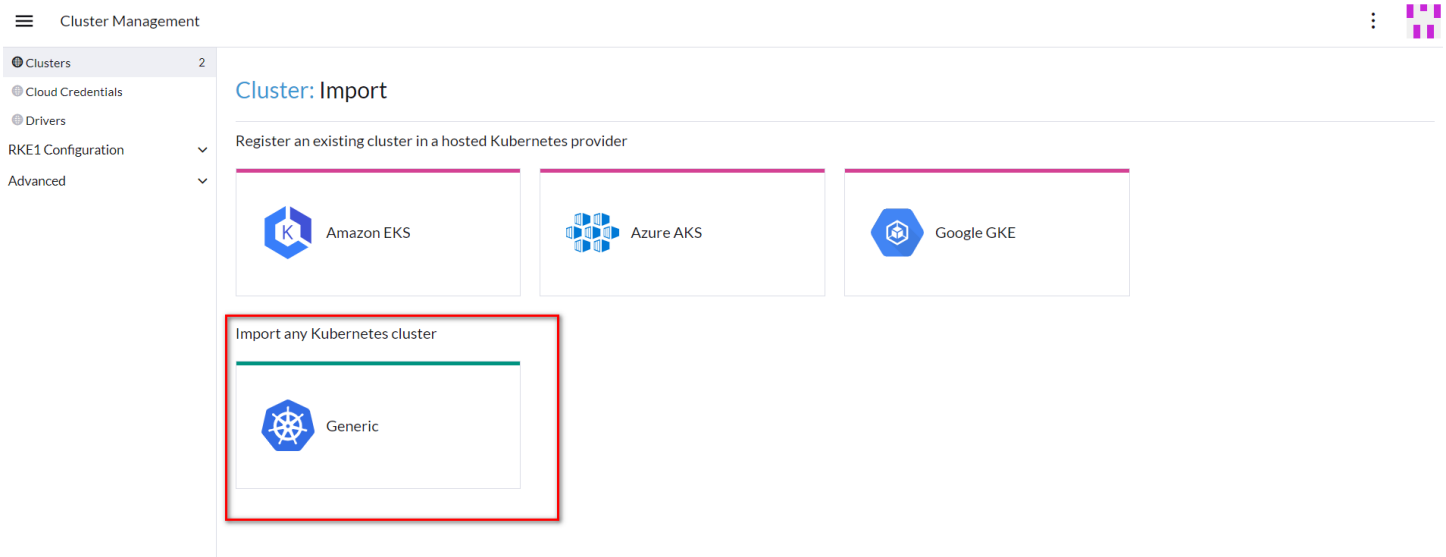


## ➢ How to import our cluster in rancher dashboard: -

### 1- Import Existing Cluster as you see below.




### 2- Choose generic one.

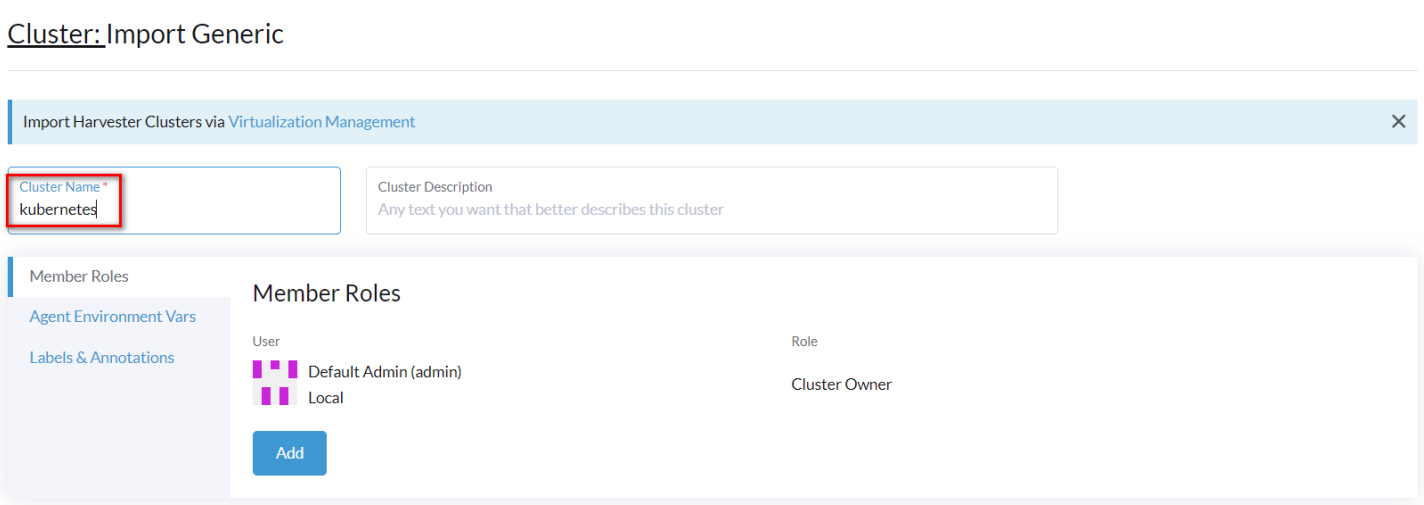



### 3- To get the name of the cluster.

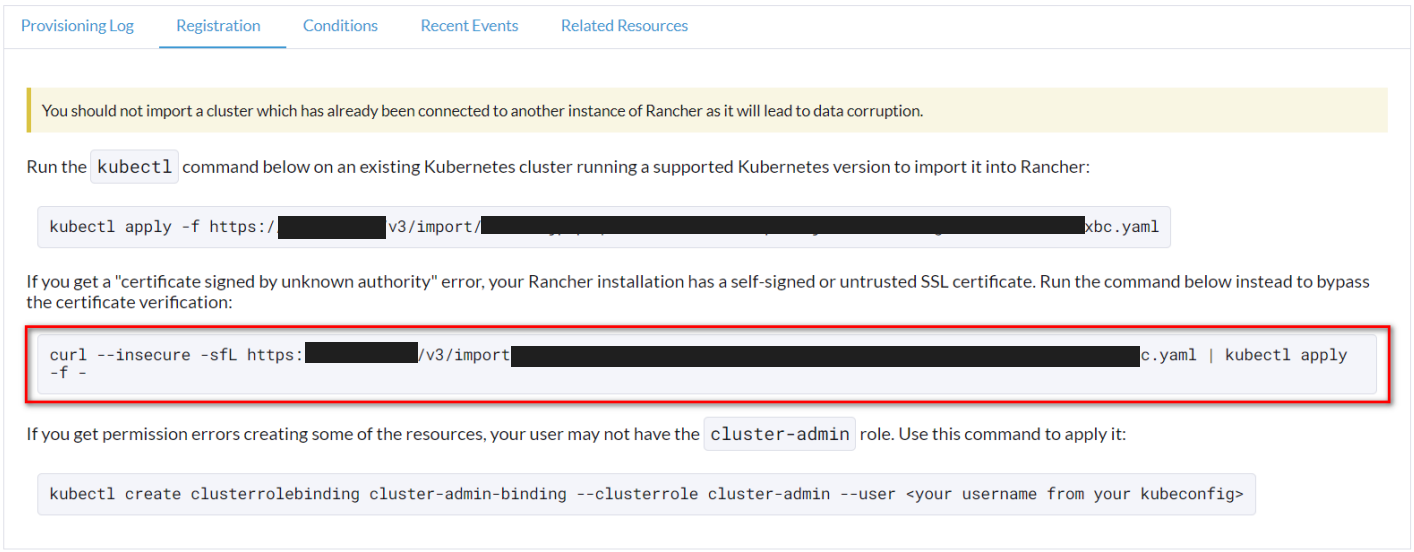
**$$ kubectl config view**

```
root@                    :~# kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server:
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

**4- Then import it in below then click next: -**




**5- Run this command on master node (192.168.1.1):-**

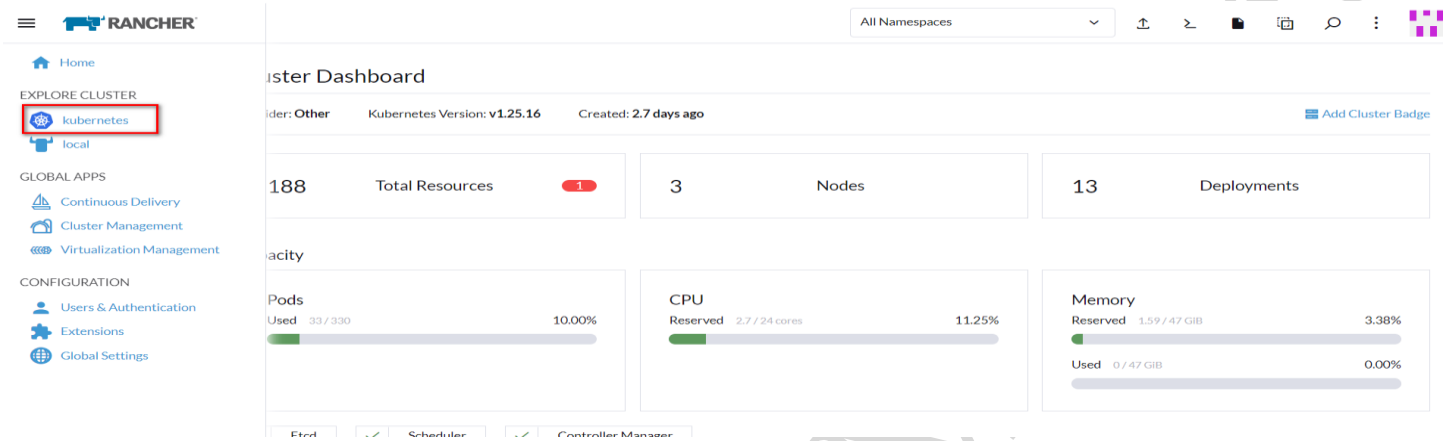



**After that you MUST wait about 20 Mint until see all pods running: -**

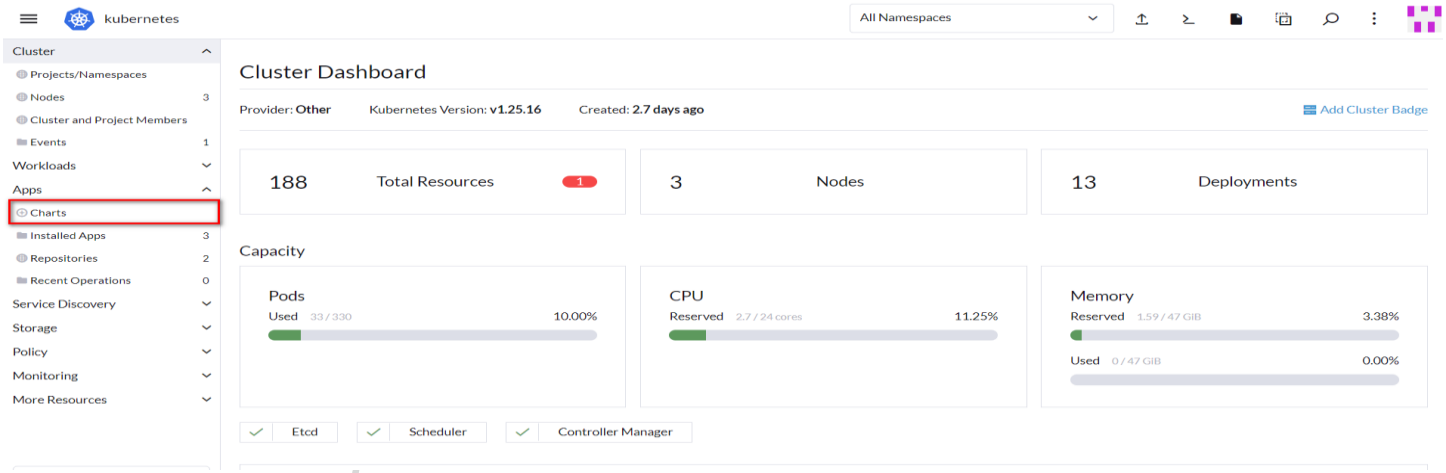
```
cattle-fleet-system          pod/fleet-agent-7d964bc469-ks82v                    0/1     CrashLoopBackOff   786 (2m ago)    2d18h
cattle-monitoring-system     pod/alertmanager-rancher-monitoring-alertmanager-0  2/2     Running            0               2d18h
cattle-monitoring-system     pod/prometheus-rancher-monitoring-prometheus-0      3/3     Running            0               2d18h
cattle-monitoring-system     pod/rancher-monitoring-grafana-5f86d88688-bpggt     4/4     Running            0               2d18h
cattle-monitoring-system     pod/rancher-monitoring-kube-state-metrics-65cd94fffc-p54d8   1/1     Running   0               2d18h
cattle-monitoring-system     pod/rancher-monitoring-operator-5bc75dbdd5-27d7h    1/1     Running            0               2d18h
cattle-monitoring-system     pod/rancher-monitoring-prometheus-adapter-786df96896-jf8gb   1/1     Running   0               2d18h
cattle-monitoring-system     pod/rancher-monitoring-prometheus-node-exporter-l7ssq   1/1     Running   0               2d18h
cattle-monitoring-system     pod/rancher-monitoring-prometheus-node-exporter-rtd7g   1/1     Running   0               2d18h
cattle-monitoring-system     pod/rancher-monitoring-prometheus-node-exporter-th9ng   1/1     Running   0               2d18h
cattle-system                pod/cattle-cluster-agent-64496bb898-ljq2p           1/1     Running            0               2d18h
cattle-system                pod/cattle-cluster-agent-64496bb898-z2shg           1/1     Running            0               2d18h
cattle-system                pod/rancher-webhook-964b85bb4-zd5zm                 1/1     Running            0               2d18h
ingress-nginx                pod/ingress-nginx-admission-create-sx9k6            0/1     Completed          0               2d18h
ingress-nginx                pod/ingress-nginx-controller-5d449ff4c4-fzrf6       1/1     Running            0               2d18h
kube-flannel                 pod/kube-flannel-ds-cqpzk                           1/1     Running            0               2d18h
kube-flannel                 pod/kube-flannel-ds-pb87f                           1/1     Running            0               2d19h
kube-flannel                 pod/kube-flannel-ds-q4796                           1/1     Running            1 (2d19h ago)   2d19h
kube-system                  pod/coredns-565d847f94-b6brh                        1/1     Running            0               2d19h
kube-system                  pod/coredns-565d847f94-pdr8k                        1/1     Running            0               2d19h
kube-system                  pod/etcd-consumer-dev-switch                        1/1     Running            1               2d19h
kube-system                  pod/kube-apiserver-consumer-dev-switch              1/1     Running            0               2d19h
kube-system                  pod/kube-controller-manager-consumer-dev-switch     1/1     Running            0               2d19h
kube-system                  pod/kube-proxy-4fxsh                                1/1     Running            0               2d19h
kube-system                  pod/kube-proxy-n24kq                                1/1     Running            0               2d18h
kube-system                  pod/kube-proxy-tb2c2                                1/1     Running            0               2d19h
kube-system                  pod/kube-scheduler-consumer-dev-switch              1/1     Running            0               2d19h
```

## ➢ How to Implement Grafana and prometheus using Rancher Dashboard: -

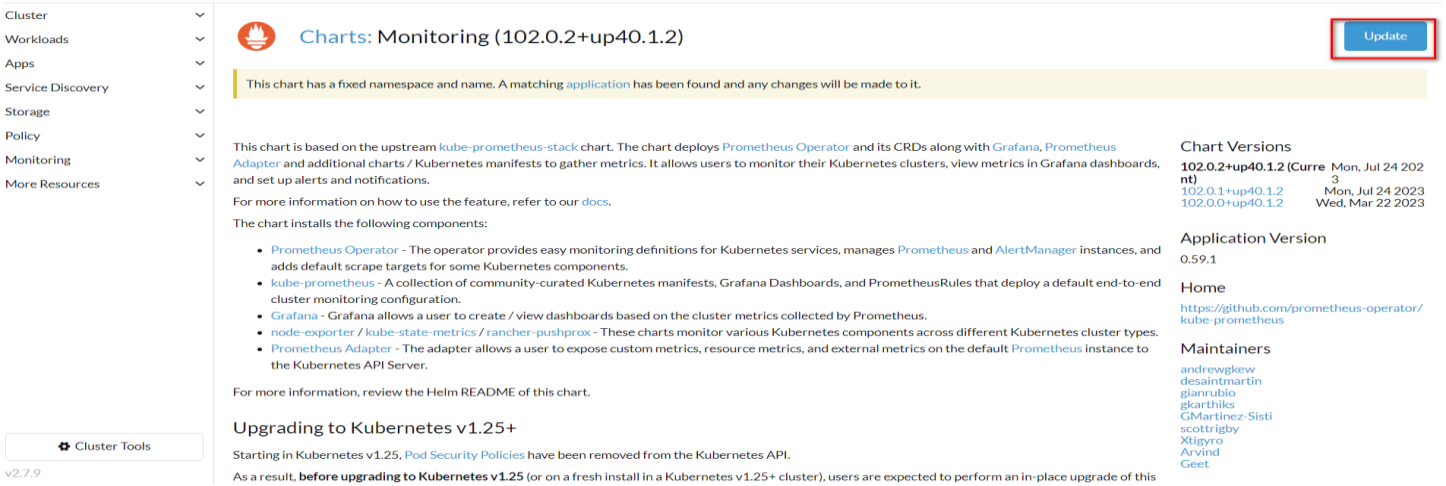
### 1- Access the cluster named Kubernetes.



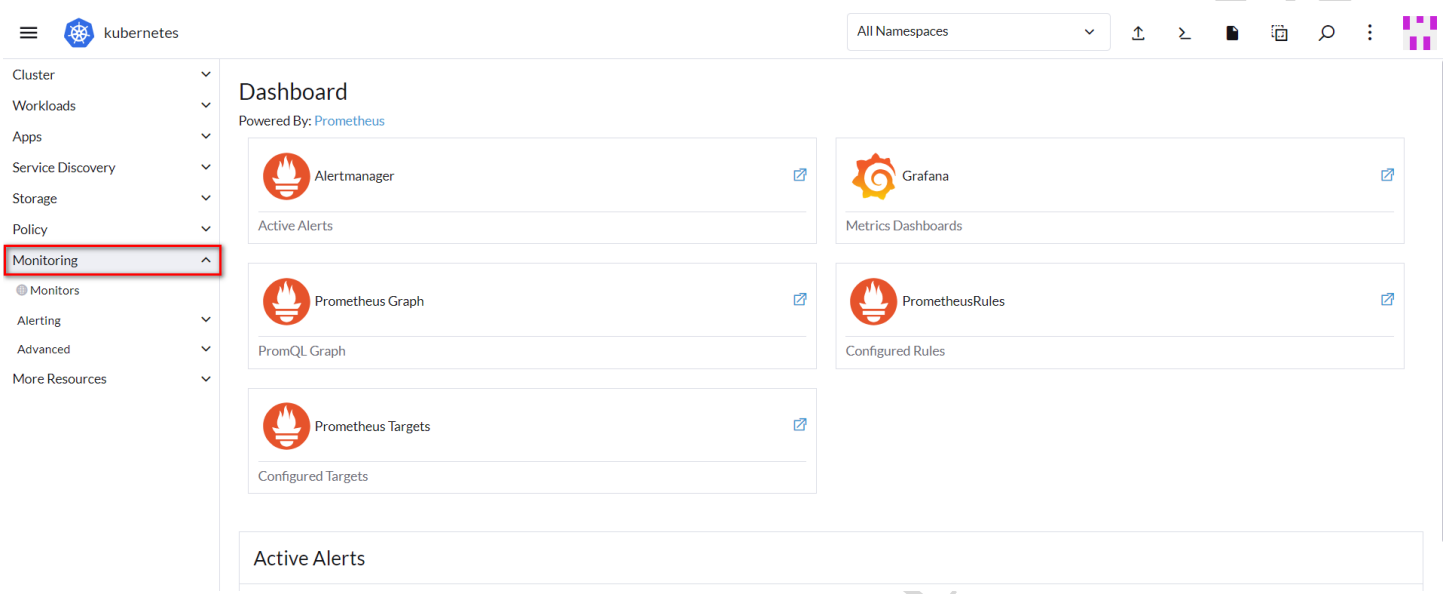

### 2- Click to charts one.




### 3- Then search for Grafana then click to install then wait about 10 Mint until pod up and running.



**4- Then access Grafana by clicking Monitoring itself then click Grafana one.**




**5- Congratulations.**

- **Configure Environment Variables Vs ConfigMap Vs Secrets.**
  - **Configure environment variables in applications: -**

```
apiVersion: v1
kind: Pod
metadata:
 name: simple-webapp-color
spec:
 containers:
 - name: simple-webapp-color
   image: simple-webapp-color
   env:
   - name: APP_COLOR
     value: pink
```

  - **Configuring ConfigMaps in Applications: -**

**ConfigMaps: it is same as environment variable but in separated yaml file when we start pod the ConfigMap will inject into pod as environment variable.**

**Let's create one: -**

1- **Create yaml file of the ConfigMap then apply it: -**

```
apiVersion: v1
kind: ConfigMap
metadata:
 name: app-config
data:
 APP_COLOR: blue
 APP_MODE: prod
```

**2- Then bind the pod to ConfigMap.**

```
apiVersion: v1
kind: Pod
metadata:
 name: simple-webapp-color
 labels:
  name: simple-webapp-color
spec:
 containers:
 - name: simple-webapp-color
   image: simple-webapp-color
 ports:
 - containerPort: 8080
 envFrom:
 - configMapRef:
   name: app-config
```

**-In case we need to retrieve only one variable APP_COLOR, check below.**

```
apiVersion: v1
kind: Pod
metadata:
 name: simple-webapp-color
 labels:
  name: simple-webapp-color
spec:
 containers:
 - name: simple-webapp-color
   image: simple-webapp-color
 ports:
 - containerPort: 8080
 env:
 - name: APP_COLOR
   valueFrom:
     configMapKeyReF:
     name: app-config
     key: APP_COLOR
```

**-You can get all ConfigMap on the cluster.**

**$$kubectl get configmaps -A** **>>> for all namespaces.**

**$$kubectl get configmaps -n ingress-nginx** **>>> for namespace called ingress-nginx.**

**$$kubectl describe configmap app-config**

## ➢ Configure Secrets in Applications: -

**I have application python connect to SQL database, host, user, password is appeared in my code and that is not a good idea, so we are going to Configure secrets: -**

**Let's create one: -**

**1- Create yaml file of the Secrets then apply it: -**

```
apiVersion: v1
kind: Secret
metadata:
 name: app-secret
data:
 DB_Host: bXlzcWw=
 DB_User: cm9vdA==
 DB_Password: cGFzd3Jk
```

**>>> on linux run $$echo -n 'mysql' | base64 >>> it will give you bXlzcWw=**
**>>> on linux run $$echo -n 'root' | base64 >>> it will give you cm9vdA==**
**>>> on linux run $$echo -n 'paswrd' | base64 >>> it will give you cGFzd3Jk**

[ **bXlzcWw=** ] is equal [ **mysql** ] using encoded base 64
[ **cm9vdA==** ] is equal [ **root** ]
[ **cGFzd3Jk** ] is equal [ **paswrd** ]

-You can get secrets

**$$kubectl get secrets**

**$$kubectl describe secrets app-secret**

**$$kubectl get secret app-secret –o yaml**

**2- Then bind the pod to Secret.**

```
apiVersion: v1
kind: Pod
metadata:
 name: simple-webapp-color
 labels:
  name: simple-webapp-color
spec:
 containers:
 - name: simple-webapp-color
   image: simple-webapp-color
 ports:
 - containerPort: 8080
 envFrom:
 - secretRef:
   name: app-secret
```

**-In case we need to retrieve only one secret DB_Password, check below.**

```
apiVersion: v1
kind: Pod
metadata:
 name: simple-webapp-color
 labels:
  name: simple-webapp-color
spec:
 containers:
 - name: simple-webapp-color
   image: simple-webapp-color
 ports:
 - containerPort: 8080
 env:
 - name: DB_Password
   valueFrom:
   secretKeyReF:
    name: app-secret
    key: DB_Password
```

**Important note: -**

**_____**

**-- Secrets are not encrypted. only encoded.**

**-- Anyone able to create pods/deployments in the same namespace can access the secrets**
**so, you must configure least-privilege access to Secrets - RBAC**
**-- you can manage the secrets from external provider such as Vault providers**

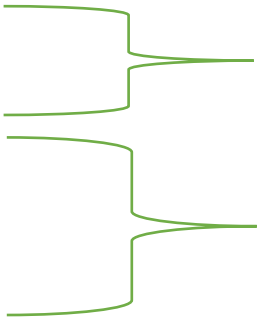
- **Storage in Kubernetes.**

**- Kubernetes is same as docker it saves data while pod is running, if the pod is terminated the data will be deleted!!**

**How to solve it?**
**----------**

**As you see below a demo pod which generates random numbers in text file on path /opt/number.out inside the pod.**

```
apiVersion: v1
kind: Pod
metadata:
  name: random-number-generator
spec:
  containers:
  - image: alpine
    name: alpine
    command: ["/bin/sh","-c"]
    args: ["shuf -i 0-100 -n 1 >> /opt/number.out;"]
  volumeMounts:
  - mountPath: /opt
    name: data-volume
  volumes:
  - name: data-volume
    hostPath:
      path: /data
      type: Directory
```



**>>> volumeMounts: used to Indicates to path of data inside pod.**

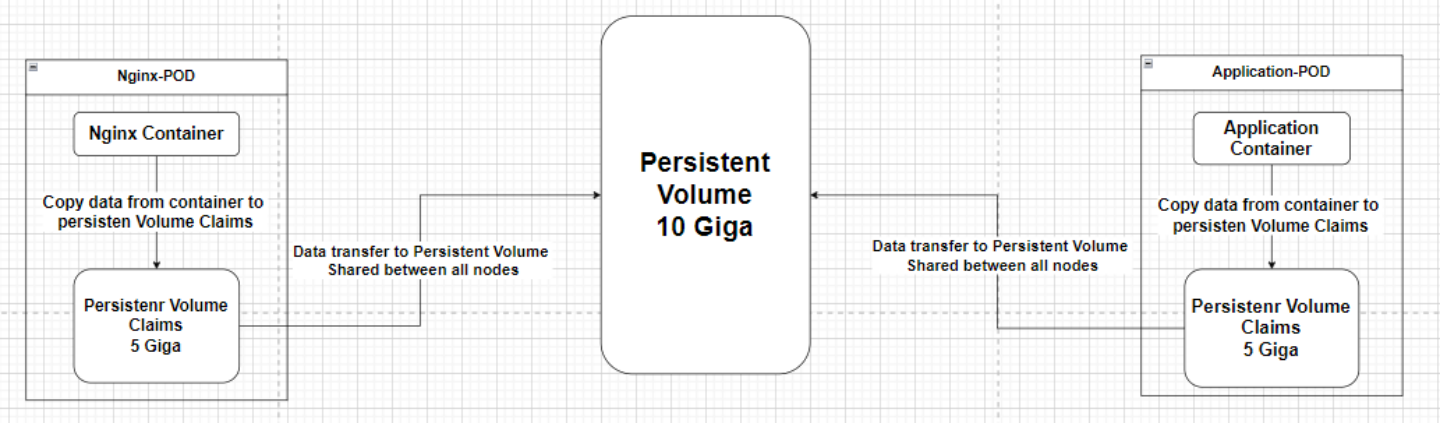
**>>> volumes: used to take data inside volumeMounts copy it to path /data on the node.**

**- for Amazon web services just edit volumes part: -**

```
volumes:
- name: data-volume
  awsElasticBlockStore:
    volumeID:  <volume-Id>
    fsType: ext4
```

**>>> ID of the volume**

## ➢ Persistent Volumes and Persistent Volumes Claims: -




**- Persistent Volumes Claims: Is object used to bind PersistentVolume to pods.**

**- Persistent Volumes: volume which takes disk space from all nodes.**

**- if there is no PersistentVolume the PersistentVolumeClaims will be pending until find the PersistentVolume to bind with.**

**Let's create one: -**

**1- Create yaml file of the PersistentVolume then apply it: -**

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: pv-vol1
 labels:
  name: my-pv
spec:
 accessModes:
 - ReadWriteOnce
 persistentVolumeReclaimPolicy: Retain
  #persistentVolumeReclaimPolicy: Delete
 capacity:
  storage: 1Gi
 hostPath:
  path: /data
```

accessModes:

- ReadWriteOnce >>> There are three type of access mode (ReadOnlyMany-ReadWriteOnce-ReadWriteMany) . accessMode means how to access data mount to that volume – Ready data only – Read and edit data one time – Read and edit data many times.

persistentVolumeReclaimPolicy: Retain >>> when you delete PVC the volume that is used by PVC will not be automatically deleted or released , you must clean up manual before used by another PVC.

#persistentVolumeReclaimPolicy: Delete >>> when you delete pvc the volume that is used by pvc will be automatically deleted, so you can used by another PVC.

**To get it**
**$$kubectl get persistentvolume**

**2- Create yaml file of the PersistentVolumeClaim then apply it: -**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: myclaim
spec:
 accessModes:
 - ReadWriteOnce
 resources:
  requests:
   storage: 500Mi
 selector:
  matchLabels:
   name: my-pv
```

**>>> access mode must be same as persistent volume that will bind with**

**>>> storage must be less than persistent volume that will bind with**

**>>> using label to bind PVC with PV**

**To get it**
**$$kubectl get persistentvolumeclaim**

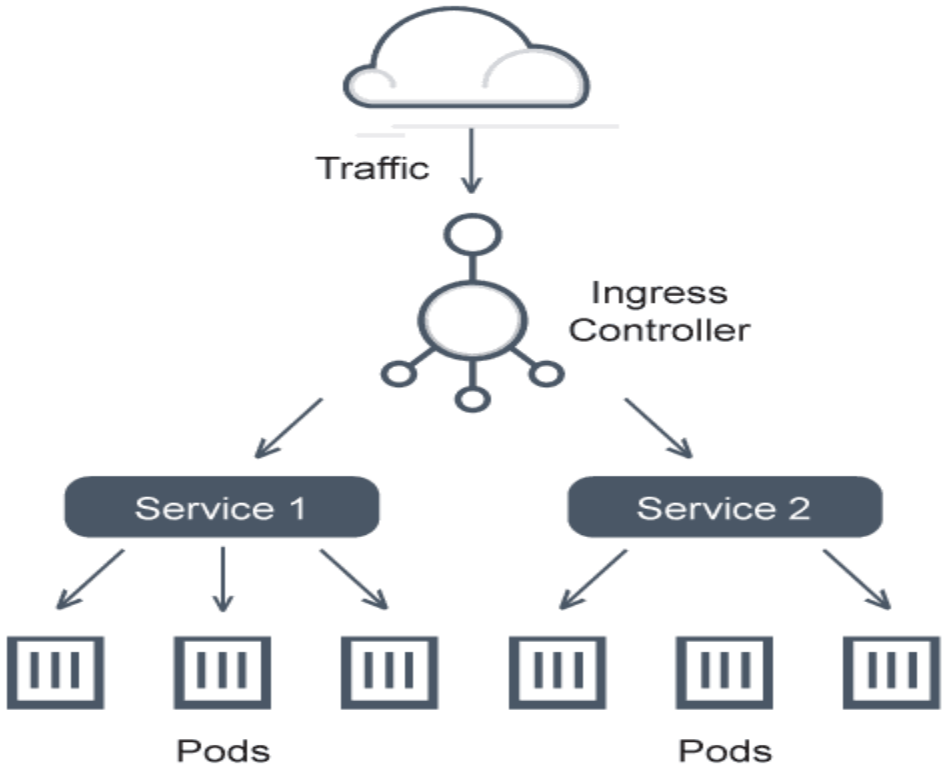
**3- Create yaml file of the Pod to mount data to PVC then apply it: -**

```
apiVersion: v1
kind: Pod
metadata:
 name: mypod
spec:
 containers:
 - name: myfrontend
   image: nginx
   volumeMounts:
   - mountPath: "/var/www/html"
     name: mypd
 volumes:
 - name: mypd
   persistentVolumeClaim:
    claimName: myclaim
```

**>>> path of data on pod that I want to mount**

**Same name of PVC (PersistentVolumeClaim)**

- **Ingress.**
  - **What is Ingress: -**




**- Ingress: an API object that helps developers expose their applications and manage external access by providing http/s routing rules to the services within a Kubernetes cluster.**

**- What does ingress consist of?**

**1-ConfigMap: is used to config parameters just like "err-log-path","keep-alive","ssl protocols" same as nginx configuration.**

**2-ServiceAccount: is used to set permissions and roles, clusterRoles, RoleBindings to access all of this pods (ConfigMap-ingress, ingress-deployment, nodeport service)**

**3-ingress deployment: it is ingress itself.**

**4-NodePort service: create service to expose the ingress controller to the external world**

**5-ingress resource: used to config routing to my application service**

**- Let's start?**

**1-Create ConfigMap-ingress.yaml with your custom "err-log-path","keep-alive","ssl protocols" and apply it: -**

```
kind: ConfigMap
apiVersion: v1
metadata:
 name: nginx-configuration
```

**2-Create ServiceAccount-ingress.yaml with your custom Roles, ClusterRoles, RoleBindings and apply it: -**

```
apiVersion: v1
kind: ServiceAccount
metadata:
name: nginx-ingress-serviceaccount
```

**3-Create ingress-deployment.yaml and apply it: -**

```
apiVersion: networking.k8s.io/v1
kind: Deployment
metadata:
  name: nginx-ingress-controller
  spec:
  replicas: 1
  selector:
   matchLabels:
    name: nginx-ingress
  template:
   metadata:
    labels:
     name: nginx-ingress
   spec:
    containers:
     - name: nginx-ingress-controller
       image: quay.io/kubernetes-ingressSTX
            controller/nginx-ingress-controller:0.21.0
    args:
     - /nginx-ingress-controller
     - --configmap=$(POD_NAMESPACE)/nginx-configuration
    env:
    - name: POD_NAME
      valueFrom:
       fieldRef:
        fieldPath: metadata.name
    - name: POD_NAMESPACE
      valueFrom:
       fieldRef:
        fieldPath: metadata.namespace
    ports:
    - name: http
      containerPort: 80
    - name: https
      containerPort: 443
```

POD_NAME : this variable takes  from ConfigMap.
POD_NAMESPACE : this variable takes from ConfigMap.

**4-Create NodePort-ingress.yaml service and apply it: -**

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-ingress
spec:
 type: NodePort
 ports:
 - port: 80
   targetPort: 80
   protocol: TCP
   name: http
 - port: 443
   targetPort: 443
   protocol: TCP
   name: https
 selector:
   name: nginx-ingress
```

**5-Create ingress-resources.yaml and apply it:-**

**namespace: critical-space >>> in case you want a specific name space**

---

**/wear >>> to access using application through ingress**

**http://**{IP_OF_ANY_NODE}:{PORT_OF_INGRESS}**/wear**

**wear-service >>> name of service of your application.**

**number: 80 >>> port of service of your application.**

---

**/watch >>> to access using application through ingress**

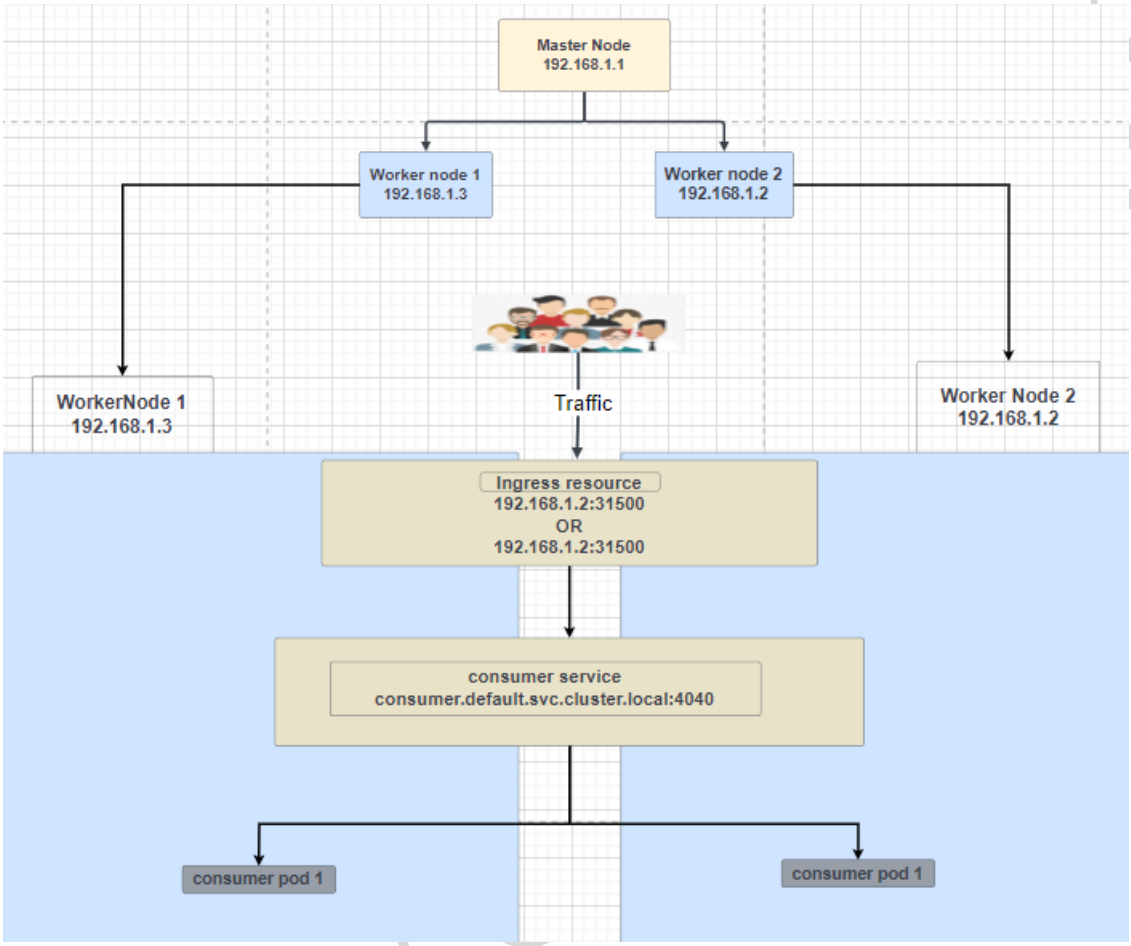
**http://**{IP_OF_ANY_NODE}:{PORT_OF_INGRESS}**/watch**

**watch-service >>> name of service of your application.**

**number: 80 >>> port of service of your application.**

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: ingress-wear-watch
 namespace: critical-space
spec:
 rules:
 - http:
    paths:
    - path: /wear
      pathType: Prefix
      backend:
      service:
       name: wear-service
       port:
        number: 80
    - path: /watch
      pathType: Prefix
      backend:
       service:
        name: watch-service
        Port:
        number: 80
```

## ➢ Another Easy way: -




- From any of worker nodes run below which create configMap – service account – ingress deployment – ingress service: -

$$kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.3/deploy/static/provider/baremetal/deploy.yaml

-Then delete ValidatingWebhookConfiguration by run below: -

$$kubectl delete -A ValidatingWebhookConfiguration ingress-nginx-admission

-Check all is running by run below: -

$$kubectl get pods --all-namespaces -l app.kubernetes.io/name=ingress-nginx
$$kubectl get services ingress-nginx-controller --namespace=ingress-nginx

-To get port of ingress to route (31500): -
$$kubectl get all -n ingress-nginx

```
root@                  l1:~# kubectl get all -n ingress-nginx
NAME                                            READY   STATUS      RESTARTS   AGE
pod/ingress-nginx-admission-create-sx9k6        0/1     Completed   0          7d15h
pod/ingress-nginx-controller-5d449ff4c4-fzrf6   1/1     Running     0          7d15h

NAME                                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
service/ingress-nginx-controller             NodePort                    <none>        80:31500/TCP,443:31501/TCP   7d15h
service/ingress-nginx-controller-admission   ClusterIP                   <none>        443/TCP                      7d15h

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1           7d15h

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-5d449ff4c4   1         1         1       7d15h

NAME                                       COMPLETIONS   DURATION   AGE
job.batch/ingress-nginx-admission-create   1/1           13s        7d15h
job.batch/ingress-nginx-admission-patch    0/1           7d15h      7d15h
```

-Then create yaml for ingress routing and apply it: -

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-backend-dev
  annotations:
    nginx.ingress.kubernetes.io/error-page: "/(/|$)(.*)"
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/use-regex: "true"
#    nginx.ingress.kubernetes.io/force-ssl-redirect: "false"
spec:
  rules:
  - http:
      paths:
      - path: /consumer(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: consumer
            port:
              number: 4040
```

/consumer used to route just like http://192.168.1.2:31500/consumer

- consumer is the name of the service of application pod want to route to.

- It is the port of the service 4040

It was an amazing journey!!!

"In the intricate dance of containers and orchestration, where pods waltz and services harmonize, Kubernetes emerges as the maestro orchestrating a symphony of scalability and resilience. As I navigate this ever-evolving landscape, I am reminded that in the world of distributed systems, Kubernetes is the conductor that transforms chaos into seamless serenity."

"I am Sherif Yehia, a DevOps engineer. I trust that this reference proves beneficial to DevOps engineers navigating the dynamic realm of technology. I am enthusiastic about sharing insights on various technologies, and you can explore more of my posts and papers on my LinkedIn profile. Thank you for your time and consideration.

Best regards, **Sherif Yehia**"



Reference: -


1- https://www.udemy.com/course/certified-kubernetes-administrator-with-practice-tests/
2- https://kubernetes.io/docs/home/
3- https://www.ibm.com/topics/kubernetes
4- https://www.redhat.com/en/topics/containers/what-is-kubernetes
5- https://medium.com/@marko.luksa/kubernetes-in-action-introducing-replication-controllers-aaa2c05e0b4e
6- https://www.baeldung.com/ops/kubernetes-deployment-vs-replicaset
7- https://avinetworks.com/glossary/kubernetes-ingress-services/
8- https://cloud.google.com/learn/what-is-kubernetes