1. Why we need manager?

We need manager for below:

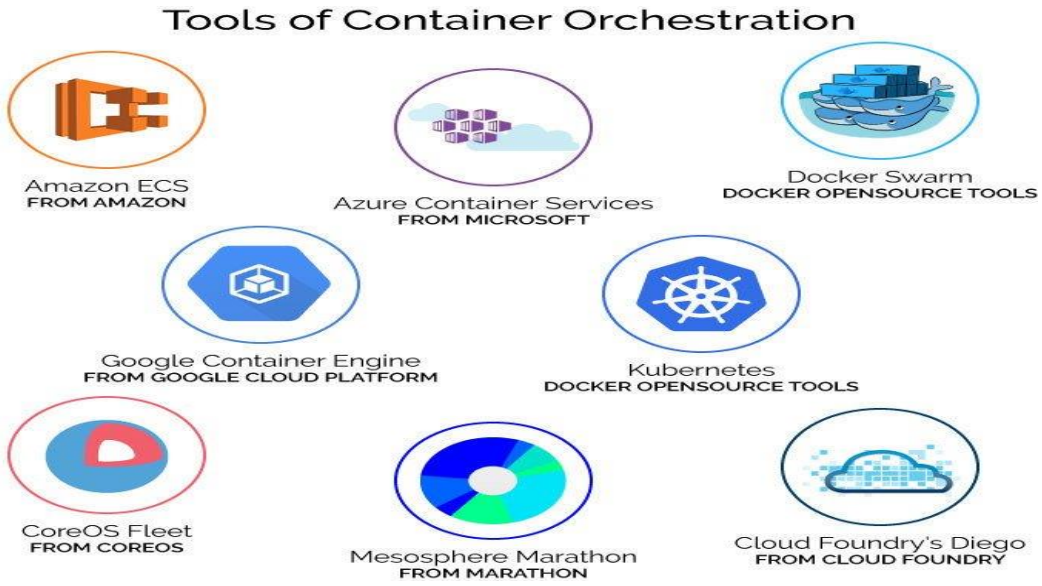
- To assign task
- To get new project
- Leave management
- Work management
- Promotions
- Adding more members
- Monitoring

2. What is container orchestration? Why we need container orchestration?

- Container orchestration automatically provisions, deploys, scales, and manages containerized applications without worrying about the underlying infrastructure.
- Container orchestration automates the deployment, management, scaling, and networking of containers.
- Hosting container based application in different environment brings many challenges to infra & devops team such as below:
    - Which nodes suit the container requirement?
    - Maintain the desire state?
    - How to share load (user request)?
    - Scalability?
    - Auto Healing?
    - Auto scaling?
    - Zero downtime upgrades/rollback
    - Container & nodes communication
    - Security—RBAC & certificates
- All the above features are provided by container orchestration tool, that's why we need this.

3. How many container orchestration software's available in marker?

- In marker below are container orchestration software's available in marker
- Rancher, Nomad, Mesos, Docker Swarm, kubernetes.
- Here kubernetes is open source
- Some of the organization taken kubernetes open source code & added their own flavors & released their own kubernetes at enterprises level & gives support for any issue.
- Amazon AWS cloud give you EKS (Elastic Kubernetes Service)
- Azure gives you AKS(Azure Kubernetes service)
- Google cloud gives you GKE(Google Kubernetes Engin)
- VMware gives you VMware Tanzu
- Redhat gives you Openshift

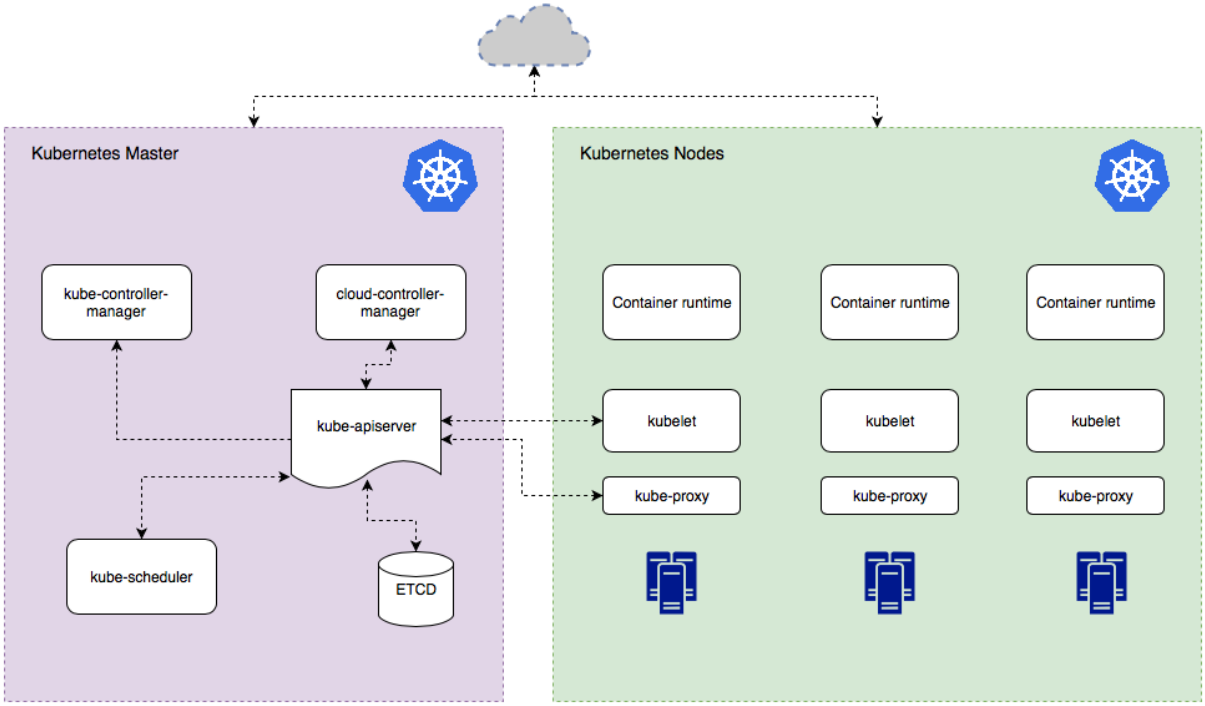


4. What is Kubernetes? What all components are available?

- Kubernetes (K8s in short) is an open-source container orchestration platform introduced by Google in 2014.
- It is a successor of Borg, Google's in-house orchestration system that accumulated over a decade of the tech giant's experience of running large enterprise workloads in production.
- In 2014, Google decided to further container ecosystem by sharing Kubernetes with the cloud native community.
- Kubernetes became the first graduated project of the newly created Cloud Native Community Foundation (CNCF), an organization conceived by Google and the Linux Foundation as the main driver of the emerging cloud native movement.
- Kubernates work on **Master-Node** model.
- **Kubectl** is keyword which is used in Kubernetes to give instructions to master.
- **Master** has below important components

Component	Description
Etcd	This is a data store used by Kubernetes to store all information about the cluster. It's critical for keeping everything up to date
kube-apiserver	This component exposes Kubernetes API to users allowing them to create API resources, run applications, and configure various parameters of the cluster. It is key components which accept all requests.
kube-scheduler	This component is responsible for scheduling user workloads on the right infrastructure. When scheduling applications, kube-scheduler considers various factors such as available node resources, node health, and availability, as well as user-defined constraints.
kube-controller-manager	The component that manages API objects created by users. It makes sure that the actual state of the cluster always matches the desired state

cloud-controller-manager	This component embraces various controllers, all of which interact with the cloud providers' APIs.



- On node (worker node) below components available
- **CRI** : To run container on worker node we need container runtime(CRI), it can be docker or container-d or Crio
- **Kubelet**: It is an agent of API server to get command from Master. And report back to API server.
- If you want to run container on kubernetes below will be steps:
    - We give request through **KUBECTL**
    - **KUBECTL** goes to **API server**, & give request to API server.
    - **API server** create object.
    - **Scheduler** keeps on watching API server & checks any request came to **API server** for container creation.
    - **Scheduler** find which node is appropriate for container creation & it tells to **API server**.
    - **API server** talks to **KUBLET** of worker node to run container.
    - **KUBLET** accept request & uses **CRI,** which will download image, run container & report back to **API Server**.
    - **API server** save that information (image, memory, CPU, number of replica) about container in **ETCD**
    - **Controller manager** keep on watching instances of container and manages desire state.
- We can have **KUBELET & CRI** in master machine also to run container on master machine.

5. How many ways we can install Kubernetes?

- We can install kubernetes in 2 ways
- **Hard way**: In hard way we need to install each & every component one by one by downloading.
- **Kubeadm way**: It is easiest way to install kubernetes, it is utility by using we get all component.

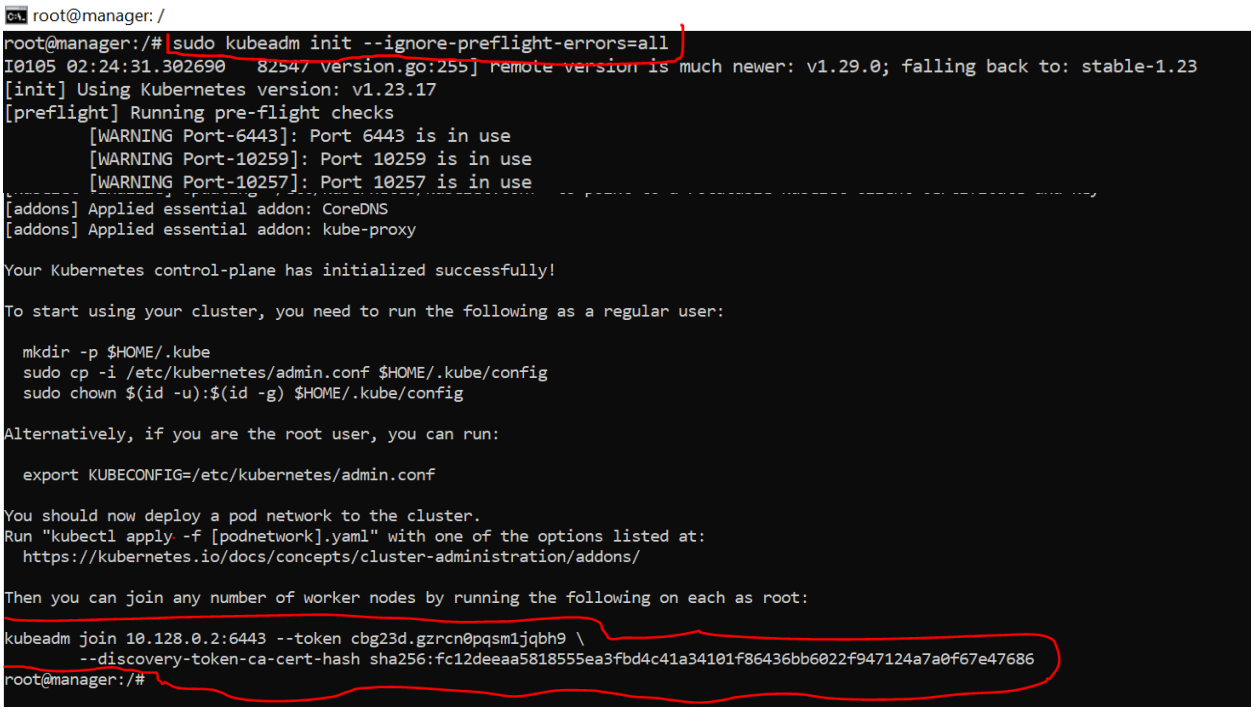
- We need minimum 2 CPU & 4 GB RAM to setup kubernates.
- It is recommended to install Kubernetes on Linux only.

6. How to install Kubernetes?

- First we need to install any container runtime (CRI) from below

Runtime	Path to Unix domain socket
Containerd	unix:///var/run/containerd/containerd.sock
CRI-O	unix:///var/run/crio/crio.sock
Docker Engine (using cri-dockerd)	unix:///var/run/cri-dockerd.sock

- After that we need to install component kubeadm, kubelet and kubectl.
  - **kubeadm:** the command to bootstrap the cluster.
  - **kubelet:** the component that runs on all of the machines in your cluster and does things like starting pods and containers.
  - **kubectl:** the command line util to talk to your cluster.
- Login into manager/master install below
  - Docker, kubeadm,kubelet,kubectl
  - Initialize master get token
  - `Syntax> sudo kubeadm init --ignore-preflight-errors=all`



  - Setup API server home path, so that kubectl can communicate with API server
  - In kubernetes home directory there is file "/etc/kubernetes/admin.conf" in which all API server details are available , so if get invoke get node command without this file path it will give error
  - `Syntax:> kubectl get nodes`
  - So to resolve this we need to execute below command
  - `Syntax:>kubectl get nodes --kubeconfig /etc/kubernetes/admin.conf`

```
root@manager:/# kubectl get nodes --kubeconfig /etc/kubernetes/admin.conf
NAME       STATUS     ROLES                  AGE   VERSION
manager    NotReady   control-plane,master   13h   v1.23.10
root@manager:/#
```

- 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
```

```
root@manager:/# mkdir -p $HOME/.kube
root@manager:/# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
root@manager:/# sudo chown $(id -u):$(id -g) $HOME/.kube/config
root@manager:/# kubectl get nodes
NAME       STATUS     ROLES                  AGE   VERSION
manager    NotReady   control-plane,master   13h   v1.23.10
root@manager:/#
```

```
devops@manager:/$ kubectl get nodes
The connection to the server localhost:8080 was refused - did you specify the right host or port?
devops@manager:/$ mkdir -p $HOME/.kube
devops@manager:/$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
devops@manager:/$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
devops@manager:/$  kubectl get nodes
NAME       STATUS     ROLES                  AGE   VERSION
manager    NotReady   control-plane,master   13h   v1.23.10
devops@manager:/$
```

- Alternatively, if you are the root user, you can run:
- `Syntax>export KUBECONFIG=/etc/kubernetes/admin.conf`
- Kubernetes does not give you networking feature, we need to install separately
- Due to networking not enabled master is showing in not ready state.
- You should need to install networking driver -- Weave/flannel/canal/calico etc... by using below command
- `> kubectl apply -f` https://raw.githubusercontent.com/projectcalico/calico/v3.24.1/manifests/calico.yaml
- Validate same using command: `>kubectl get nodes`

```
root@manager: /
root@manager:/# kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.24.1/manifests/calico.yaml
poddisruptionbudget.policy/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
serviceaccount/calico-node created
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/caliconodestatuses.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipreservations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
deployment.apps/calico-kube-controllers created
root@manager:/# kubectl get nodes
NAME       STATUS     ROLES                  AGE   VERSION
manager    NotReady   control-plane,master   23h   v1.23.10
root@manager:/# kubectl get nodes
NAME       STATUS   ROLES                  AGE   VERSION
manager    Ready    control-plane,master   23h   v1.23.10
root@manager:/#
```

- Now we can join any number of worker nodes by running the following on each as root:
- Login into worker/node install below
    - Docker, kubeadm,kubelet,kubectl
    - Apply token by using below command
    - `> kubeadm join 10.128.0.2:6443 --token cbg23d.gzrcn0pqsm1jqbh9 --discovery-token-ca-cert-hash sha256:fc12deeaa5818555ea3fbd4c41a34101f86436bb6022f947124a7a0f67e47686`

root@worker-1: /

```
root@worker-1:/# kubeadm join 10.128.0.2:6443 --token cbg23d.gzrcn0pqsm1jqbh9 --discovery-token-ca-cert-hash sha25
67e47686
[preflight] Running pre-flight checks
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 24.0.7. Latest
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
W0105 14:49:57.098984   67736 utils.go:69] The recommended value for "resolvConf" in "KubeletConfiguration" is: /r
n/systemd/resolve/resolv.conf
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

root@worker-1:/#
```

root@worker-2: /

```
root@worker-2:/# kubeadm join 10.128.0.2:6443 --token cbg23d.gzrcn0pqsm1jqbh9 --discovery-token-ca-cert-hash sh
67e47686
[preflight] Running pre-flight checks
        [WARNING SystemVerification]: this Docker version is not on the list of validated versions: 24.0.7. Lat
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
W0105 14:51:11.591326   66565 utils.go:69] The recommended value for "resolvConf" in "KubeletConfiguration" is:
n/systemd/resolve/resolv.conf
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

root@worker-2:/#
```

- We can check worker who has join cluster using below command
- `Synatx> kubectl get nodes`

root@manager: /

```
root@manager:/# kubectl get nodes
NAME       STATUS   ROLES                  AGE     VERSION
manager    Ready    control-plane,master   23h     v1.23.10
worker-1   Ready    <none>                 2m54s   v1.23.10
worker-2   Ready    <none>                 100s    v1.23.10
root@manager:/# sudo kubeadm token create --print-join-command
kubeadm join 10.128.0.2:6443 --token j2iva5.dpobern3v30xrfw3 --discovery-token-ca-cert-hash sha256:fc12deeaa5818555ea3fbd4c41a34101f86436bb6022f947124a7a0f67e47686
root@manager:/#
```

- Run Below on Master Node to get join token
- `Syntax> sudo kubeadm token create --print-join-command`

7. What is Pod in Kubernetes?

- In kubernetes we don't run container independently, instead we run container inside POD.
- Pod runs our container in kubernetes
- Pod maintains resources, IP address, storege etc of container.
- It makes sure that container will be up & running always.
- Pod is security layer on top of container.
- If something happened to container pod tries to bring up that container.
- In kubernetes anything we create is called as object.
- Pod is first & smallest object, that we run in kubernetes.

8. What is use of kubectl? What are its different commands?

- In order to perform any task in kubernetes we use client command "**kubectl**"
- Command to start pod
- `Syntax :> kubectl run <pod name> --image <image name>`
- `Ex:> kubectl run pod1 --image nginx`
- Command to check pods
- `Syntax :> kubectl get pods`
- Command to check pods with wider output, which gives you details like name, status, ip, node etc.
- `Syntax :> kubectl get pods –o wide`

```
root@manager: /
root@manager:/# kubectl run pod1 --image nginx
pod/pod1 created
root@manager:/# kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
pod1    1/1     Running   0          64s
root@manager:/# kubectl get pods -o wide
NAME    READY   STATUS    RESTARTS   AGE     IP                NODE       NOMINATED NODE   READINESS GATES
pod1    1/1     Running   0          2m27s   192.168.133.193   worker-2   <none>           <none>
root@manager:/#
```

- Command to check details of a pod, which gives detail like name, namespace, node where it is running, its container details, volumes, events, etc….
- `Syntax :> kubectl describe pod <pod name>`
- `Syntax :> kubectl describe pod pod1`

```
root@manager: /
root@manager:/# kubectl get pods -o wide
NAME    READY   STATUS    RESTARTS   AGE     IP                NODE       NOMINATED NODE   READINESS GATES
pod1    1/1     Running   0          2m27s   192.168.133.193   worker-2   <none>           <none>
root@manager:/# kubectl describe pod pod1
Name:             pod1
Namespace:        default
Priority:         0
Node:             worker-2/10.128.0.5
Start Time:       Sun, 07 Jan 2024 06:53:54 +0000
Labels:           run=pod1
Annotations:      cni.projectcalico.org/containerID: 5b6c3ce9a28eecb96bb205ae54a3defefe3529041d1c3e0dbe09f743d35ba229
                  cni.projectcalico.org/podIP: 192.168.133.193/32
                  cni.projectcalico.org/podIPs: 192.168.133.193/32
Status:           Running
IP:               192.168.133.193
IPs:
  IP:  192.168.133.193
Containers:
  pod1:
    Container ID:   docker://70392234487b8e3cddf0ba6e87a7a24206ac77eaf97ea83d53f2f7a48ca6506c
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:2bdc49f2f8ae8d8dc50ed00f2ee56d00385c6f8bc8a8b320d0a294d9e3b49026
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Sun, 07 Jan 2024 06:53:56 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-vgqfv (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
```

root@manager: /

```
Volumes:
  kube-api-access-vgqfv:
    Type:                    Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  13m   default-scheduler  Successfully assigned default/pod1 to worker-2
  Normal  Pulling    13m   kubelet            Pulling image "nginx"
  Normal  Pulled     13m   kubelet            Successfully pulled image "nginx" in 488.167684ms
  Normal  Created    13m   kubelet            Created container pod1
  Normal  Started    13m   kubelet            Started container pod1
root@manager:/#
```

- Command to delete single pod
- `Syntax :> kubectl delete pod <pod name>`
- `Ex :> kubectl delete pod pod1`
- Command to delete all pods
- `Syntax :> kubectl delete pods --all`

root@manager: /

```
root@manager:/# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          31m
root@manager:/# kubectl delete pod pod1
pod "pod1" deleted
root@manager:/# kubectl get pods
No resources found in default namespace.
root@manager:/# kubectl run pod2 --image alpine
pod/pod2 created
root@manager:/# kubectl get pods -o wide
NAME   READY   STATUS             RESTARTS     AGE   IP               NODE       NOMINATED NODE   READINESS GATES
pod2   0/1     CrashLoopBackOff   1 (6s ago)   10s   192.168.133.194   worker-2   <none>           <none>
root@manager:/# kubectl get pods -o wide
NAME   READY   STATUS             RESTARTS     AGE   IP               NODE       NOMINATED NODE   READINESS GATES
pod2   0/1     CrashLoopBackOff   1 (8s ago)   12s   192.168.133.194   worker-2   <none>           <none>
root@manager:/# kubectl get pods -o wide
NAME   READY   STATUS             RESTARTS      AGE   IP               NODE       NOMINATED NODE   READINESS GATES
pod2   0/1     CrashLoopBackOff   1 (11s ago)   15s   192.168.133.194   worker-2   <none>           <none>
root@manager:/# kubectl get pods -o wide
NAME   READY   STATUS             RESTARTS     AGE   IP               NODE       NOMINATED NODE   READINESS GATES
pod2   0/1     CrashLoopBackOff   2 (3s ago)   19s   192.168.133.194   worker-2   <none>           <none>
root@manager:/# kubectl delete pods --all
pod "pod2" deleted
root@manager:/# kubectl get pods -o wide
No resources found in default namespace.
root@manager:/# kubectl get pods
No resources found in default namespace.
root@manager:/#
```

- Command to enter inside POD
- `Syntax :> kubectl exec -it <pod name> --bash`
- `Ex :> kubectl exec -it pod3 -- bash`
- Command to exit from POD `> exit`
- We can use CTRL+PQ Command to exit from POD

root@manager: /

```
root@manager:/# kubectl exec -it pod3 -- bash
root@pod3:/# exit
exit
root@manager:/#
```

- Command to check log of pod
- `Syntax :> kubectl logs <pod name>`
- `Ex :> kubectl logs pod3`
- `Syntax :> kubectl logs -f <pod name>`
- `Ex :> kubectl logs -f pod3`

```
root@manager: /
root@manager:/# kubectl logs pod3
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/01/11 12:55:06 [notice] 1#1: using the "epoll" event method
2024/01/11 12:55:06 [notice] 1#1: nginx/1.25.3
2024/01/11 12:55:06 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/01/11 12:55:06 [notice] 1#1: OS: Linux 5.15.0-1047-gcp
2024/01/11 12:55:06 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/01/11 12:55:06 [notice] 1#1: start worker processes
2024/01/11 12:55:06 [notice] 1#1: start worker process 29
2024/01/11 12:55:06 [notice] 1#1: start worker process 30
root@manager:/# kubectl logs -f pod3
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/01/11 12:55:06 [notice] 1#1: using the "epoll" event method
2024/01/11 12:55:06 [notice] 1#1: nginx/1.25.3
2024/01/11 12:55:06 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/01/11 12:55:06 [notice] 1#1: OS: Linux 5.15.0-1047-gcp
2024/01/11 12:55:06 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/01/11 12:55:06 [notice] 1#1: start worker processes
2024/01/11 12:55:06 [notice] 1#1: start worker process 29
2024/01/11 12:55:06 [notice] 1#1: start worker process 30
```

- If we run OS container image in pod it's status will go in "CrashLoopBackOff", because in plain OS container there is no back ground running process available, so it is getting stopped & pod is again & again trying to bring it up.
- So in kubernates we never run OS container in pods.
- Command to get all resources available on kubernetes
- `Syntax:> kubectl api-resources`

```
root@manager: /
root@manager:/# kubectl api-resources
NAME                              SHORTNAMES   APIVERSION                             NAMESPACED   KIND
bindings                                       v1                                     true         Binding
componentstatuses                 cs           v1                                     false        ComponentStatus
configmaps                        cm           v1                                     true         ConfigMap
endpoints                         ep           v1                                     true         Endpoints
events                            ev           v1                                     true         Event
limitranges                       limits       v1                                     true         LimitRange
namespaces                        ns           v1                                     false        Namespace
nodes                             no           v1                                     false        Node
persistentvolumeclaims            pvc          v1                                     true         PersistentVolumeClaim
persistentvolumes                 pv           v1                                     false        PersistentVolume
pods                              po           v1                                     true         Pod
podtemplates                                   v1                                     true         PodTemplate
replicationcontrollers            rc           v1                                     true         ReplicationController
resourcequotas                    quota        v1                                     true         ResourceQuota
secrets                                        v1                                     true         Secret
serviceaccounts                   sa           v1                                     true         ServiceAccount
services                          svc          v1                                     true         Service
mutatingwebhookconfigurations                  admissionregistration.k8s.io/v1        false        MutatingWebhookConfiguration
validatingwebhookconfigurations                admissionregistration.k8s.io/v1        false        ValidatingWebhookConfiguration
customresourcedefinitions         crd,crds     apiextensions.k8s.io/v1                false        CustomResourceDefinition
apiservices                                    apiregistration.k8s.io/v1              false        APIService
controllerrevisions                            apps/v1                                true         ControllerRevision
daemonsets                        ds           apps/v1                                true         DaemonSet
deployments                       deploy       apps/v1                                true         Deployment
replicasets                       rs           apps/v1                                true         ReplicaSet
statefulsets                      sts          apps/v1                                true         StatefulSet
tokenreviews                                   authentication.k8s.io/v1               false        TokenReview
localsubjectaccessreviews                      authorization.k8s.io/v1                true         LocalSubjectAccessReview
selfsubjectaccessreviews                       authorization.k8s.io/v1                false        SelfSubjectAccessReview
selfsubjectrulesreviews                        authorization.k8s.io/v1                false        SelfSubjectRulesReview
subjectaccessreviews                           authorization.k8s.io/v1                false        SubjectAccessReview
horizontalpodautoscalers          hpa          autoscaling/v2                         true         HorizontalPodAutoscaler
cronjobs                          cj           batch/v1                               true         CronJob
jobs                                           batch/v1                               true         Job
certificatesigningrequests        csr          certificates.k8s.io/v1                 false        CertificateSigningRequest
leases                                         coordination.k8s.io/v1                 true         Lease
bgpconfigurations                              crd.projectcalico.org/v1               false        BGPConfiguration
bgppeers                                       crd.projectcalico.org/v1               false        BGPPeer
blockaffinities                                crd.projectcalico.org/v1               false        BlockAffinity
```

- Command to get all resources api version in kubernetes
- `Syntax:> kubectl api-versions`

```
root@manager: /
root@manager:/# kubectl api-versions
admissionregistration.k8s.io/v1
apiextensions.k8s.io/v1
apiregistration.k8s.io/v1
apps/v1
authentication.k8s.io/v1
authorization.k8s.io/v1
autoscaling/v1
autoscaling/v2
autoscaling/v2beta1
autoscaling/v2beta2
batch/v1
batch/v1beta1
certificates.k8s.io/v1
coordination.k8s.io/v1
crd.projectcalico.org/v1
discovery.k8s.io/v1
discovery.k8s.io/v1beta1
events.k8s.io/v1
events.k8s.io/v1beta1
flowcontrol.apiserver.k8s.io/v1beta1
flowcontrol.apiserver.k8s.io/v1beta2
networking.k8s.io/v1
node.k8s.io/v1
node.k8s.io/v1beta1
policy/v1
policy/v1beta1
rbac.authorization.k8s.io/v1
scheduling.k8s.io/v1
storage.k8s.io/v1
storage.k8s.io/v1beta1
v1
root@manager:/#
```

- Command to get explanation of any resources/ component of kubernetes
- `Syntax:> kubectl explain <component/component name>`
- `Ex:> kubectl explain pods`
- `Ex:> kubectl explain namespace`

```
root@manager:/# kubectl explain pods
KIND:     Pod
VERSION:  v1

DESCRIPTION:
     Pod is a collection of containers that can run on a host. This resource is
     created by clients and scheduled onto hosts.

FIELDS:
   apiVersion   <string>
     APIVersion defines the versioned schema of this representation of an
     object. Servers should convert recognized schemas to the latest internal
     value, and may reject unrecognized values. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

   kind <string>
     Kind is a string value representing the REST resource this object
     represents. Servers may infer this from the endpoint the client submits
     requests to. Cannot be updated. In CamelCase. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

   metadata     <Object>
     Standard object's metadata. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata

   spec <Object>
     Specification of the desired behavior of the pod. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status

   status       <Object>
     Most recently observed status of the pod. This data may not be up to date.
     Populated by the system. Read-only. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status
```

```
root@manager:/# kubectl explain namespace
KIND:     Namespace
VERSION:  v1

DESCRIPTION:
     Namespace provides a scope for Names. Use of multiple namespaces is
     optional.

FIELDS:
   apiVersion   <string>
     APIVersion defines the versioned schema of this representation of an
     object. Servers should convert recognized schemas to the latest internal
     value, and may reject unrecognized values. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources

   kind <string>
     Kind is a string value representing the REST resource this object
     represents. Servers may infer this from the endpoint the client submits
     requests to. Cannot be updated. In CamelCase. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds

   metadata     <Object>
     Standard object's metadata. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata

   spec <Object>
     Spec defines the behavior of the Namespace. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status

   status       <Object>
     Status describes the current status of a Namespace. More info:
     https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#spec-and-status
```

- We can get inner object detail also using explain command
- `Ex:> kubectl explain pod.spec`
- `Ex:> kubectl explain pod.spec.containers`
- `EX:> kubectl explain pod.spec.containers.resources`

```
root@manager:/# kubectl explain pod.spec.containers.resources
KIND:     Pod
VERSION:  v1

RESOURCE: resources <Object>

DESCRIPTION:
     Compute Resources required by this container. Cannot be updated. More info:
     https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

     ResourceRequirements describes the compute resource requirements.

FIELDS:
   limits       <map[string]string>
     Limits describes the maximum amount of compute resources allowed. More
     info:
     https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

   requests     <map[string]string>
     Requests describes the minimum amount of compute resources required. If
     Requests is omitted for a container, it defaults to Limits if that is
     explicitly specified, otherwise to an implementation-defined value. More
     info:
     https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
```

- Command to edit pod
- `Syntax :> kubectl edit <pod name>`
- `Ex :> kubectl edit pod3`
- It will open YML file in which we can edit & change image to new image

```
root@manager: /
root@manager:/# kubectl get pods
NAME    READY   STATUS    RESTARTS        AGE
pod1    1/1     Running   1 (35m ago)     42h
pod2    1/1     Running   1 (5h31m ago)   34h
pod3    1/1     Running   1 (35m ago)     10h
root@manager:/# kubectl edit pod3
error: the server doesn't have a resource type "pod3"
root@manager:/# kubectl edit pod pod3
pod/pod3 edited
root@manager:/# kubectl get pods
NAME    READY   STATUS    RESTARTS        AGE
pod1    1/1     Running   1 (37m ago)     42h
pod2    1/1     Running   1 (5h33m ago)   34h
pod3    1/1     Running   2 (6s ago)      10h
root@manager:/#
```

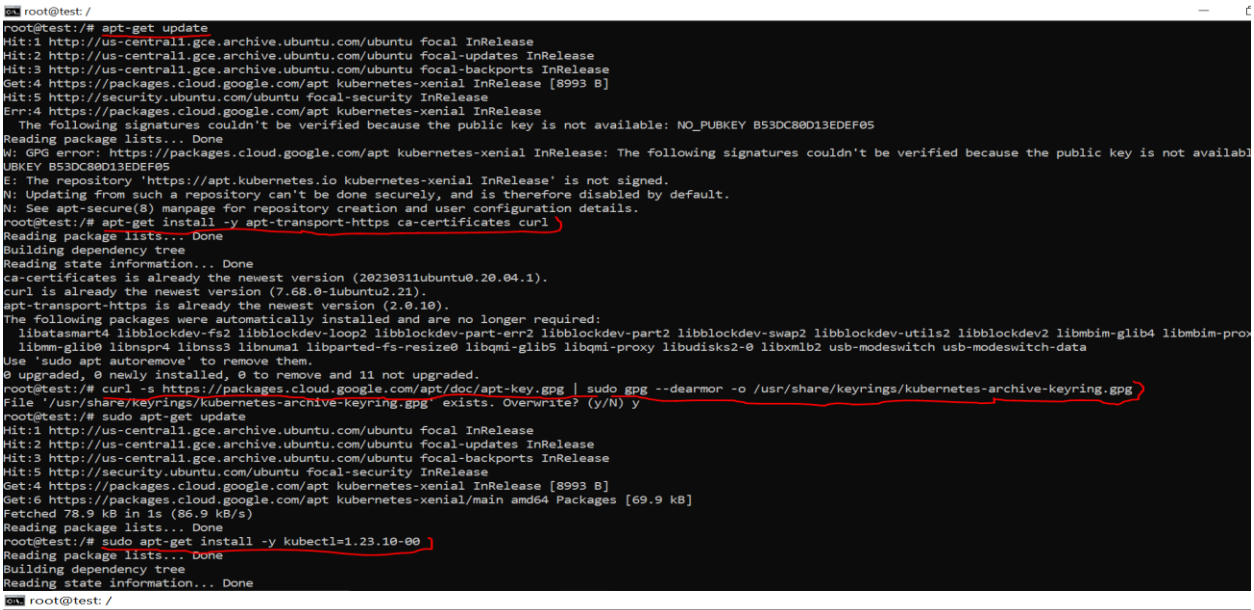
```
                        node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason          Age                From     Message
  ----    ------          ----               ----     -------
  Normal  SandboxChanged  36m (x2 over 37m)  kubelet  Pod sandbox changed, it will be killed and re-created.
  Normal  Pulling         36m                kubelet  Pulling image "nginx"
  Normal  Pulled          36m                kubelet  Successfully pulled image "nginx" in 380.315077ms
  Normal  Killing         77s                kubelet  Container pod3 definition changed, will be restarted
  Normal  Created         76s (x2 over 36m)  kubelet  Created container pod3
  Normal  Started         76s (x2 over 36m)  kubelet  Started container pod3
  Normal  Pulling         76s                kubelet  Pulling image "httpd"
  Normal  Pulled          76s                kubelet  Successfully pulled image "httpd" in 416.25632ms
root@manager:/#
```

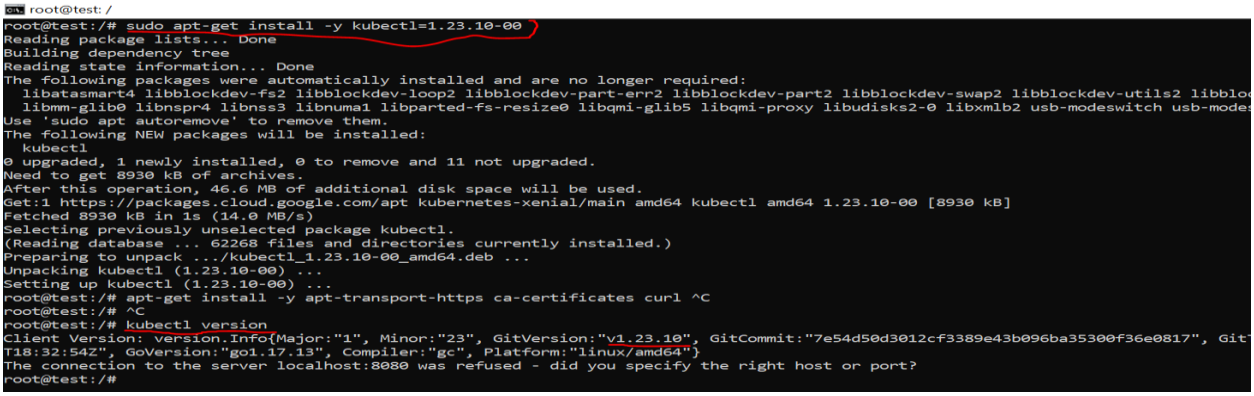
- Command to delete everything i.e. all pods, controllers & services from kubernetes
- `Syntax:> kubectl delete all --all`

```
root@manager-1: /
root@manager-1:/# kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
rs1-2w85d   1/1     Running   0          10h
rs1-j4f98   1/1     Running   0          10h
rs1-wrwjv   1/1     Running   0          10h
root@manager-1:/# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    11h
svc1         ClusterIP   10.106.194.48   <none>        2345/TCP   6h59m
svc2         ClusterIP   10.97.106.220   <none>        2345/TCP   6h54m
root@manager-1:/# kubectl get rc
No resources found in default namespace.
root@manager-1:/# kubectl get rc
No resources found in default namespace.
root@manager-1:/# kubectl get rs
NAME   DESIRED   CURRENT   READY   AGE
rs1    3         3         3       10h
root@manager-1:/# kubectl delete --all
error: at least one resource must be specified to use a selector
root@manager-1:/# kubectl delete all --all
pod "rs1-2w85d" deleted
pod "rs1-j4f98" deleted
pod "rs1-wrwjv" deleted
service "kubernetes" deleted
service "svc1" deleted
service "svc2" deleted
replicaset.apps "rs1" deleted
```

9. How to run kubectl in any machine which will connect to API server which runs on master machine?

- In enterprise we don't have access to master machine of kubernetes cluster.
- So we can install kubectl in any machine & connect to API server of master machine using below steps:
- Create new machine→ install Linux OS → login in
- Install kubectl using below commands
- `>apt-get update`
- `>apt-get install -y apt-transport-https ca-certificates curl`
- `>apt-get update`
- `> sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/kubernetes-archive-keyring.gpg`
- `>apt-get update`
- `>sudo wget https://raw.githubusercontent.com/lerndevops/labs/master/scripts/installK8S-v1-23.sh -P /tmp`
- `>apt-get install -y kubectl=1.23.10-00`





- Copy admin.conf container from manager machine & create config file in new machine using below steps
- Login manager machine open admin.conf & copy content
- `> cat /etc/kubernetes/admin.conf`
- Login new machine execute below command
- `> mkdir .kube`
- `> cd .kube/`
- `> vi config`
- Paste file content & update permission
- `> sudo chown $(id -u):$(id -g) $HOME/.kube/config`

- Now check in new machine using below command
- `> kubectl get nodes`

root@client: /

```
root@client:/# kubectl get nodes
NAME       STATUS   ROLES                  AGE   VERSION
manager    Ready    control-plane,master   28h   v1.23.10
worker-1   Ready    <none>                 27h   v1.23.10
worker-2   Ready    <none>                 27h   v1.23.10
root@client:/#
```

10. What is use of label? How we can use label?

- Label is use to identify nodes.
- We can attach label to pod so that we can identify it easily.
- `Syntax> kubectl run <pod name> --image <image name> --labels='<key value labels>'`
- `Ex:> kubectl run pod2 --image httpd --labels='own=san,env=prod'`
- We can check label of pod using below command
- `Syntax:> kubectl get pods --show-labels`
- We can show pods based on some label condition using below command
- `Syntax:> kubectl get pods -l <condition>`
- `Ex:> kubectl get pods -l own=san`

root@manager: /

```
root@manager:/# kubectl run pod1 --image httpd --labels='own=san,env=prod'
Error from server (AlreadyExists): pods "pod1" already exists
root@manager:/# kubectl run pod2 --image httpd --labels='own=san,env=prod'
pod/pod2 created
root@manager:/# kubectl get pods
NAME   READY   STATUS    RESTARTS   AGE
pod1   1/1     Running   0          8h
pod2   1/1     Running   0          2m35s
root@manager:/# kubectl get pods --show-labels
NAME   READY   STATUS    RESTARTS   AGE     LABELS
pod1   1/1     Running   0          8h      run=pod1
pod2   1/1     Running   0          5m25s   env=prod,own=san
root@manager:/# kubectl get pods -l own=san
NAME   READY   STATUS    RESTARTS   AGE
pod2   1/1     Running   0          10m
```

11. How we can use environment variable?

- We can add environment variable to run pod using below command
- `Syntax>kubectl run <pod name> --image <image name> --env='<key value env variables>'`
- `EX:> kubectl run pod3 --image nginx --env='DBNAME=www.sa.com'`
- We can check environment variable using below command
- `Ex:> kubectl describe pod3`
- We can check environment variable by entering into pod using below command
- `Synatx:> kubectl exec -it <pod name> -- bash`
- `Ex:> kubectl exec -it pod3 - bash`
  `> env`

```
root@manager: /
pod2   1/1     Running   0          24h
root@manager:/# kubectl run pod3 --image nginx --env='DBNAME=www.sa.com'
pod/pod3 created
root@manager:/# kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
pod1    1/1     Running   0          32h
pod2    1/1     Running   0          24h
pod3    1/1     Running   0          15s
root@manager:/# kubectl describe pod3
error: the server doesn't have a resource type "pod3"
root@manager:/# kubectl describe pod pod3
Name:         pod3
Namespace:    default
Priority:     0
Node:         worker-2/10.128.0.5
Start Time:   Thu, 11 Jan 2024 03:08:21 +0000
Labels:       run=pod3
Annotations:  cni.projectcalico.org/containerID: 92848e7ef2af67c99ba388dba60c4b658d1faccc2272e55f27dde808541e56f1
              cni.projectcalico.org/podIP: 192.168.133.195/32
              cni.projectcalico.org/podIPs: 192.168.133.195/32
Status:       Running
IP:           192.168.133.195
IPs:
  IP:  192.168.133.195
Containers:
  pod3:
    Container ID:   docker://d43b71d04a4fd76317e488fd98eccfde4764ded54b05839a4ae9e0cfc53b7bdc
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:2bdc49f2f8ae8d8dc50ed00f2ee56d00385c6f8bc8a8b320d0a294d9e3b49026
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Thu, 11 Jan 2024 03:08:23 +0000
    Ready:          True
    Restart Count:  0
    Environment:
      DBNAME:  www.sa.com
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-9sh2d (ro)
root@manager:/# kubectl exec -it pod3 -- bash
root@pod3:/# env
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_SERVICE_PORT=443
HOSTNAME=pod3
DBNAME=www.sa.com
PWD=/
PKG_RELEASE=1~bookworm
HOME=/root
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
NJS_VERSION=0.8.2
TERM=xterm
SHLVL=1
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PORT=443
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
NGINX_VERSION=1.25.3
_=/usr/bin/env
root@pod3:/# echo JDBC:ODBS:$DBBAME
JDBC:ODBS:
root@pod3:/# echo JDBC:ODBS:$DBNAME
JDBC:ODBS:www.sa.com
root@pod3:/#
```

12. How to create pod using YML file?

- We can give request to kubernetes to create pos using YML file.
- To create any pos using YML file we need to add below tags
- **API version:** APIVersion defines the versioned schema of this representation of an object
- **Kind:** Kind is a string value representing the REST resource this object represents
- **Metadata:** Standard object's metadata
- **spec**: Specification of the desired behavior of the pod.
- We can create YML file like below: p.yml

```
apiVersion: v1
kind: Pod
metadata:
```

```
            name: mypod
            labels:
               env: dev
      spec:
        containers:
          - name: one
            image: tomcat
```

- We can create pod using below command

```
kubectl create -f p.yml
```

```
root@manager: /
root@manager:/# vim p.yml
```

```
root@manager: /
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    env: dev
spec:
  containers:
  - name: one
    image: tomcat
root@manager:/# kubectl create -f p.yml
pod/mypod created
```

```
root@manager:/# kubectl get pods
NAME      READY   STATUS    RESTARTS        AGE
mypod     1/1     Running   0               5m29s
pod1      1/1     Running   1 (29h ago)     2d23h
pod2      1/1     Running   1 (34h ago)     2d15h
pod3      1/1     Running   2 (28h ago)     39h
root@manager:/#
```

- We can check YML file during running pod using below command

Syntax>kubectl run <pod name> --image <image name> --dry-run=client -o yaml
EX:> kubectl run pod4 --image nginx --dry-run=client -o yaml

```
root@manager: /
root@manager:/# kubectl run pod4 --image nginx --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: pod4
  name: pod4
spec:
  containers:
  - image: nginx
    name: pod4
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

- We can check JSON file during running pod using below command

Syntax>kubectl run <pod name> --image <image name> --dry-run=client -o json
EX:> kubectl run pod4 --image nginx --dry-run=client -o json

```
root@manager:/# kubectl run pod4 --image nginx --dry-run=client -o json
{
    "kind": "Pod",
    "apiVersion": "v1",
    "metadata": {
        "name": "pod4",
        "creationTimestamp": null,
        "labels": {
            "run": "pod4"
        }
    },
    "spec": {
        "containers": [
            {
                "name": "pod4",
                "image": "nginx",
                "resources": {}
            }
        ],
        "restartPolicy": "Always",
        "dnsPolicy": "ClusterFirst"
    },
    "status": {}
}
```

- Pod can run multiple containers, but cannot be identical i.e. cannot have same image, port.
- Multiple container use case is tomcat & logger container , tomcat & cache container
- When we have tight dependence of 2 containers, those should run together & die together in pod.
- All containers in pod having same IP address.
- We can create multiple container using YAML file **p.yml** as below

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    env: dev
spec:
  containers:
  - name: one
    image: tomcat
  - name: two
    image: nginx
```

- Here we can denote multiple container with **hyphen -**
- Now create container using below command
- Syntx> kubectl create -f p.yml

root@manager: /

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    env: dev
spec:
  containers:
  - name: one
    image: tomcat
  - name: two
    image: nginx
~
~
```

root@manager: /

```
root@manager:/# kubectl create -f p.yml
pod/mypod created
root@manager:/# kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
mypod   2/2     Running   0          10s
root@manager:/# kubectl get pods -o wide
NAME    READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
mypod   2/2     Running   0          16s   192.168.133.201   worker-2   <none>           <none>
root@manager:/# kubectl describe pod mypod
Name:         mypod
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  72s   default-scheduler  Successfully assigned default/mypod to worker-2
  Normal  Pulling    71s   kubelet            Pulling image "tomcat"
  Normal  Pulled     70s   kubelet            Successfully pulled image "tomcat" in 406.85446ms
  Normal  Created    70s   kubelet            Created container one
  Normal  Started    70s   kubelet            Started container one
  Normal  Pulling    70s   kubelet            Pulling image "nginx"
  Normal  Pulled     70s   kubelet            Successfully pulled image "nginx" in 413.340772ms
  Normal  Created    70s   kubelet            Created container two
  Normal  Started    70s   kubelet            Started container two
```

13. What is request & limit in kubernetes?

- In prod environment we need to restrict/limit resource i.e. CPU & RAM. usage by container, because extreme resource usage by one container may impact on other container running in system. So to avoid this situation request & limits came into picture.
- **Request** is minimum resource we need to run pod/container & perform its task.
- When we give request to API Server via kubectl client to run pod using minimum resource optionally, **scheduler** check that minimum number of resource is available in which node, in that node on pod gets run.
- **Limit** is maximum number of usage, where to stop occupying resource by pod/container.
- When you specify a resource limit for a container to run, the **kubelet** enforces those limits so that the running container is not allowed to use more of that resource than the limit you set.
- If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its **request** for that resource specifies. However, a container is not allowed to use more than its resource **limit**.
- In kubernetes we measure CPU(Computer Processing Unit) in core & RAM(Random Access Memory) in binary system Mebibyte (MiB)
- 1 core = 1000milli CPU
- In binary 1kb =1024byte in decimal 1kb=1000 byte
- We can check CPU using command:> lscpu
- We can check RAM using command:> free -h
- We can create container with request & limit as below YML file l.yml

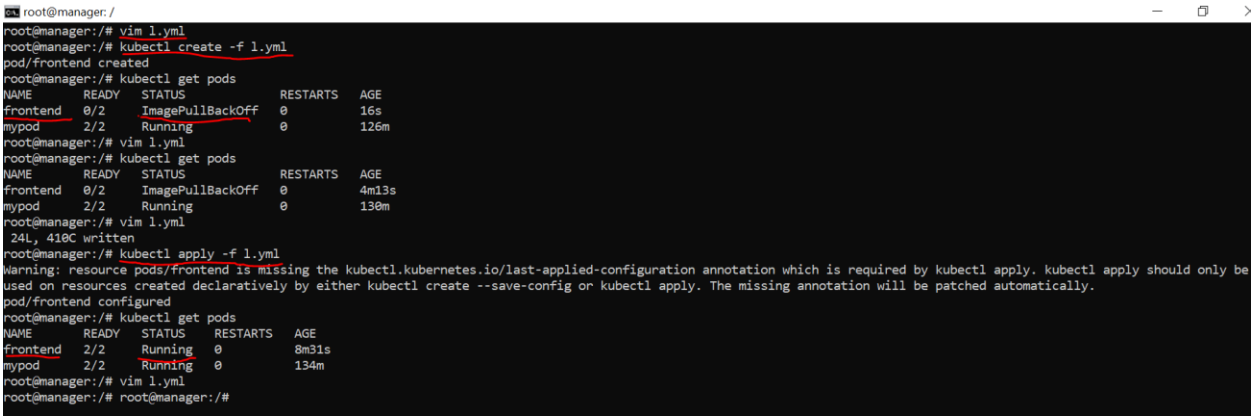
```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "300m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "300m"
```

- Now run container using below command
- `Syntax:> kubectl create -f l.yml`
- We can modify YML file to change image & update existing container using below command
- Change existing YML file l.yml using command > `vim l.yml`

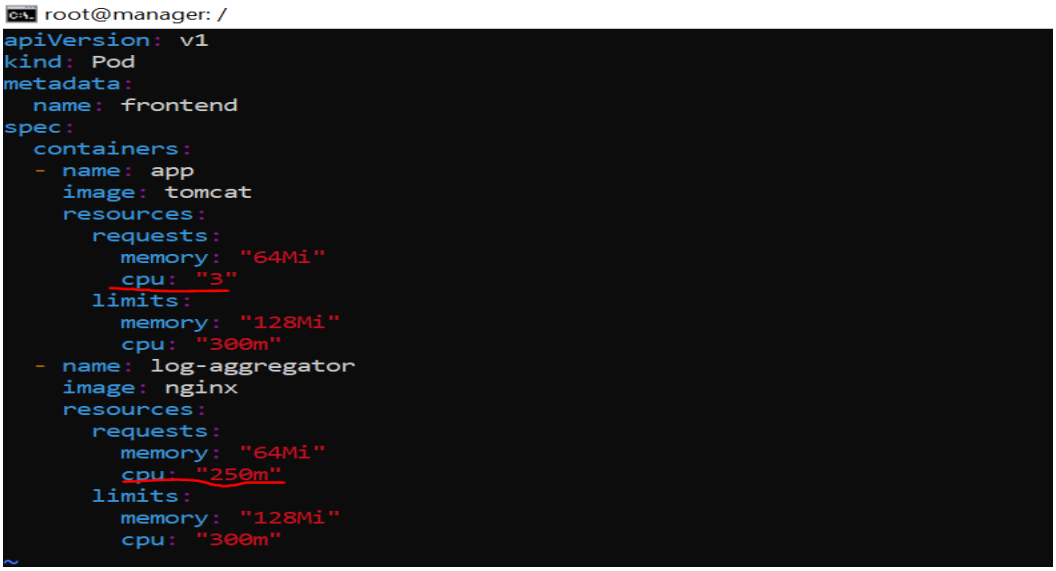
```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
```

```
spec:
  containers:
  - name: app
    image: tomcat
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "300m"
  - name: log-aggregator
    image: nginx
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "300m"
```

- Update container using below command
- `Syntax:> kubectl apply -f <changed file name>`
- `Ex:> kubectl apply -f l.yml`



- **Request** of resource for container should always less that **limit**, else it cannot create container, it will give compilation error while creating container



- If we try to create container using command, give compilation error
- Ex:>kubectl create -f l.yml

```
root@manager: /
root@manager:/# vim l.yml
root@manager:/# kubectl create -f l.yml
The Pod "frontend" is invalid: spec.containers[0].resources.requests: Invalid value: "3": must be less than or equal to cpu limit
root@manager:/#
```

- If we give over limit of resource for container than system capacity it accept request but cannot create container
- Change limit in YML file > vi l.yml

root@manager: /

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: tomcat
    resources:
      requests:
        memory: "64Mi"
        cpu: "2.5"
      limits:
        memory: "128Mi"
        cpu: "3"
  - name: log-aggregator
    image: nginx
    resources:
      requests:
        memory: "64Mi"
        cpu: "200m"
      limits:
        memory: "128Mi"
        cpu: "300m"
~
```

- If we try to create container using command, it creates container but not able to run container because of limited resource
- Ex:>kubectl create -f l.yml

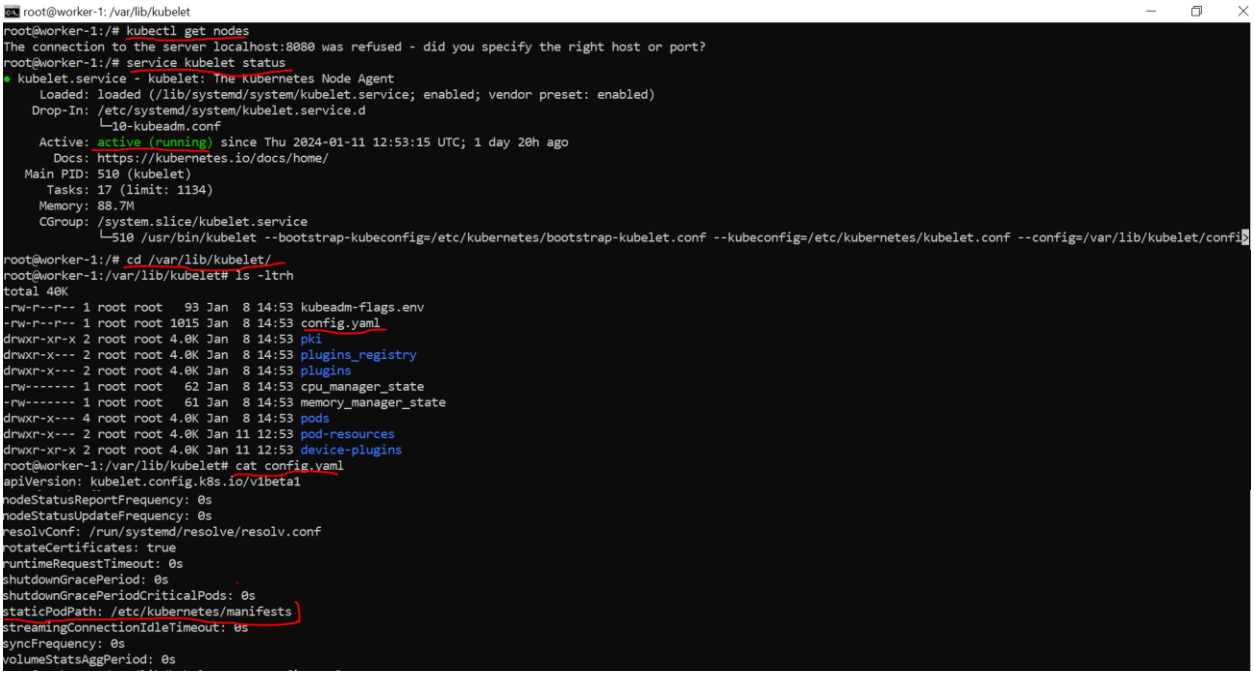
root@manager: /

```
root@manager:/# vim l.yml
root@manager:/# kubectl create -f l.yml
pod/frontend created
root@manager:/# kubectl get pods
NAME       READY   STATUS    RESTARTS   AGE
frontend   0/2     Pending   0          12s
root@manager:/# kubectl describe pod frontend
Name:         frontend
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  20s   default-scheduler  0/3 nodes are available: 1 node(s) had taint {node-role.kubernetes
.io/master: }, that the pod didn't tolerate, 2 Insufficient cpu.
root@manager:/#
```

- If container's RAM limit is reached container will restart automatically
- If container's CPU limit is reached it reduces performance

14. What is static pod in kubernetes?

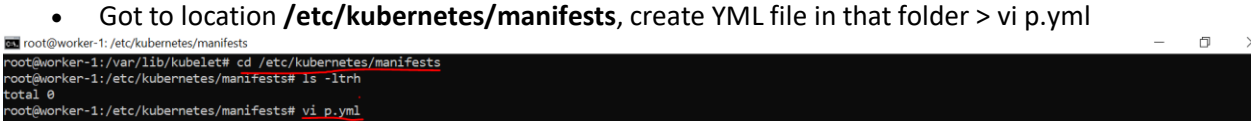
- We can create pod on worker node using kubelet running on it, such pods are called as static pods.
- These pods are managed by kubelet directly
- We can check kubelt service running status using below command
- `Syntax:> service kubelet status`
- Kubelet is install in location `/var/lib/kubelet/`
- We can check **config.yaml** file in above location, in that file we can find location where static pod's YML file to be placed.
- The name of tag to find location is **staticPodPath: /etc/kubernetes/manifests**
- Kubelet keeps watching on static post path i.e. **/etc/kubernetes/manifests**, when it find any YML file it create that POD
- We can also change this location in **config.yaml** file



- Got to location **/etc/kubernetes/manifests**, create YML file in that folder > vi p.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    env: dev
spec:
  containers:
  - name: one
    image: tomcat
  - name: two
    image: nginx
```

- Got to location **/etc/kubernetes/manifests**, create YML file in that folder > vi p.yml



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    env: dev
spec:
  containers:
  - name: one
    image: tomcat
  - name: two
    image: nginx
```

- Now we can check pod is running or not on manager using below command
- `>kubectl get nodes`
- `>kubectl get nodes –o wide`

```
root@manager:/# root@manager:/# kubectl get pods
NAME            READY   STATUS    RESTARTS      AGE
frontend        0/2     Pending   0             103m
mypod-worker-1  2/2     Running   2 (62s ago)   117s
root@manager:/# kubectl get pods -o wide
NAME            READY   STATUS    RESTARTS        AGE    IP              NODE       NOMINATED NODE   READINESS GATES
frontend        0/2     Pending   0               112m   <none>          <none>     <none>           <none>
mypod-worker-1  2/2     Running   2 (9m45s ago)   10m    192.168.226.66  worker-1   <none>           <none>
```

- So whenever we static pod on any of the worker node, its name will be shown like `<pod name>-<node name> i.e. mypod-worker-1`
- Any pod having node name as extension is called as static pods
- If we try to delete this static pod it will be auto created again, that means static pod are **not managed** by **API server**, it is **managed** by **kubelet**
- `Ex:> kubectl delete pod mypod-worker-1`
- `Ex:> kubectl get pods -o wide`

```
root@manager:/# kubectl delete pod mypod-worker-1
pod "mypod-worker-1" deleted
root@manager:/# kubectl get pods -o wide
NAME            READY   STATUS    RESTARTS   AGE    IP       NODE       NOMINATED NODE   READINESS GATES
frontend        0/2     Pending   0          130m   <none>   <none>     <none>           <none>
mypod-worker-1  0/2     Pending   0          3s     <none>   worker-1   <none>           <none>
root@manager:/# kubectl get pods -o wide
NAME            READY   STATUS    RESTARTS      AGE    IP              NODE       NOMINATED NODE   READINESS GATES
frontend        0/2     Pending   0             130m   <none>          <none>     <none>           <none>
mypod-worker-1  2/2     Running   2 (27m ago)   7s     192.168.226.66  worker-1   <none>           <none>
root@manager:/#
```

- If we want to delete static pod we need to remove YML file from worker

```
root@worker-1:/etc/kubernetes/manifests# rm -rf p.yml
root@worker-1:/etc/kubernetes/manifests#


root@manager:/# kubectl get pods -o wide
NAME       READY   STATUS    RESTARTS   AGE    IP       NODE       NOMINATED NODE   READINESS GATES
frontend   0/2     Pending   0          135m   <none>   <none>     <none>           <none>
root@manager:/#
```

15. What is namespace in kubernetes? Why we need namespace?

- Suppose in big house if there are different-different rooms are available, so one room person cannot handle resource of another room.
- Similarly in big kubernetes cluster, which is shared across multiple teams, there should have restriction to access resource/pod of one team from another team directly.
- So to provide this kind of restriction to resource/pod in kubernates **namespace** is used.
- We can create separate spaces to run different pod in different space, which is called as **namespace**.
- It is a way to isolate workloads.
- We can create namespace using below command
- `Syntax:> kubectl create ns <name space name>`
- `Ex:> kubectl create ns ns1`
- `Ex:> kubectl create ns ns2`
- We can create pod to run in above name space using below command
- `Syntax:> kubectl run <pod name> --image <image name> -n <namespace>`

- Ex:> kubectl run pod1 --image nginx -n ns1
- Ex:> kubectl run pod2 --image nginx -n ns2
- Here we have creates 2 pods which run in 2 different namespace i.e. ns1:pod1 & ns2:pod2
- Now if we try to access these pods it is give error “No resources found in default namespace.”

```
root@manager: /
root@manager:/# kubectl create ns ns1
namespace/ns1 created
root@manager:/# kubectl create ns ns2
namespace/ns2 created
root@manager:/# kubectl run pod1 --image nginx -n ns1
pod/pod1 created
root@manager:/# kubectl run pod2 --image nginx -n ns2
pod/pod2 created
root@manager:/# kubectl get pods
No resources found in default namespace.
root@manager:/#
```

- So these resource/pods are available in their respective namespaces.
- In this way we can create different-different namespaces & give access to teams, so that they can run their own pods in their respective namespace only.
- Manager can access these pods using below command
- Syntax:> kubectl get pods -n <namespace>
- Ex:> kubectl get pods -n ns1
- Ex:> kubectl get pods -n ns2

```
root@manager:/# kubectl get pods -n ns1
NAME    READY   STATUS    RESTARTS   AGE
pod1    1/1     Running   0          17m
root@manager:/# kubectl get pods -n ns2
NAME    READY   STATUS    RESTARTS   AGE
pod2    1/1     Running   0          17m
root@manager:/#
```

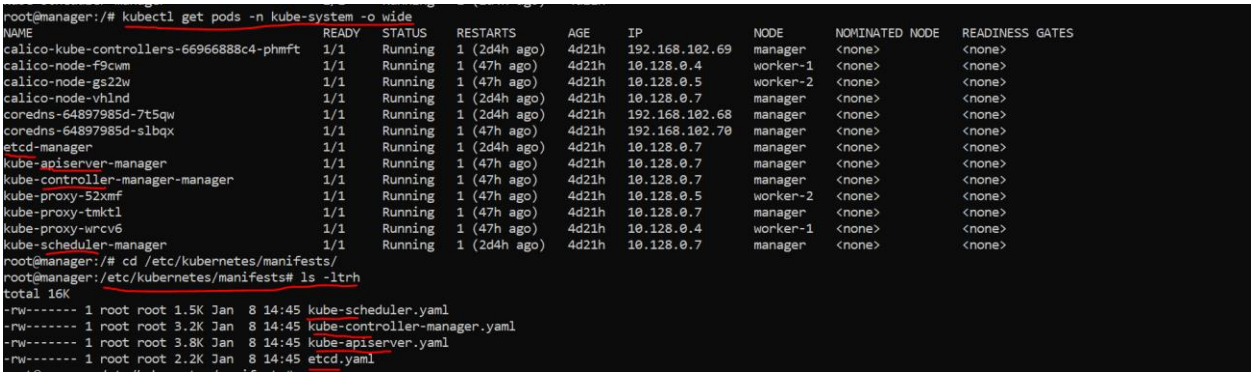
- Manager can delete namespace using below command
- Syntax:> kubectl delete ns <namespace>
- Ex:> kubectl delete ns ns1

```
root@manager:/# kubectl delete ns ns1
namespace "ns1" deleted
```

- Manager can get all namespace using below command
- Syntax:> kubectl get ns

```
root@manager:/# kubectl get ns
NAME                 STATUS   AGE
default              Active   4d20h
kube-node-lease      Active   4d20h
kube-public          Active   4d20h
kube-system          Active   4d20h
ns2                  Active   27m
```

- In kubernetes we have below namespaces inbuilt
- **Default**: If we don’t specify any namespace kubectl will create resource in default namespace.
- **Kube-node-lease**: This namespace is for system health checkup pods, like heart beat messages & all
- **Kube-public:** If we create some resource in public namespace it will be available for all. For ex monitoring pods.
- **Kube-system:** It holds all system level resources, we can get all system level resource using below command
- Syntax:>kubectl get pods -n kube-system



```
root@manager:/# kubectl get pods -n kube-system -o wide
NAME                                   READY   STATUS    RESTARTS      AGE     IP               NODE       NOMINATED NODE   READINESS GATES
calico-kube-controllers-66966888c4-phmft   1/1     Running   1 (2d4h ago)   4d21h   192.168.102.69   manager    <none>           <none>
calico-node-f9cwm                      1/1     Running   1 (47h ago)    4d21h   10.128.0.4       worker-1   <none>           <none>
calico-node-gs22w                      1/1     Running   1 (47h ago)    4d21h   10.128.0.5       worker-2   <none>           <none>
calico-node-vhlnd                      1/1     Running   1 (2d4h ago)   4d21h   10.128.0.7       manager    <none>           <none>
coredns-64897985d-7t5qw                1/1     Running   1 (2d4h ago)   4d21h   192.168.102.68   manager    <none>           <none>
coredns-64897985d-slbqx                1/1     Running   1 (47h ago)    4d21h   192.168.102.70   manager    <none>           <none>
etcd-manager                           1/1     Running   1 (2d4h ago)   4d21h   10.128.0.7       manager    <none>           <none>
kube-apiserver-manager                 1/1     Running   1 (47h ago)    4d21h   10.128.0.7       manager    <none>           <none>
kube-controller-manager-manager        1/1     Running   1 (47h ago)    4d21h   10.128.0.7       manager    <none>           <none>
kube-proxy-52xmf                       1/1     Running   1 (47h ago)    4d21h   10.128.0.5       worker-2   <none>           <none>
kube-proxy-tmktl                       1/1     Running   1 (47h ago)    4d21h   10.128.0.7       manager    <none>           <none>
kube-proxy-wrcv6                       1/1     Running   1 (47h ago)    4d21h   10.128.0.4       worker-1   <none>           <none>
kube-scheduler-manager                 1/1     Running   1 (2d4h ago)   4d21h   10.128.0.7       manager    <none>           <none>
root@manager:/# cd /etc/kubernetes/manifests/
root@manager:/etc/kubernetes/manifests# ls -ltrh
total 16K
-rw------- 1 root root 1.5K Jan  8 14:45 kube-scheduler.yaml
-rw------- 1 root root 3.2K Jan  8 14:45 kube-controller-manager.yaml
-rw------- 1 root root 3.8K Jan  8 14:45 kube-apiserver.yaml
-rw------- 1 root root 2.2K Jan  8 14:45 etcd.yaml
```

- Here we can see **etcd, apiserver, controller-manager & scheduler** all are running as static pods.
- Kubernetes software itself running as container & it manages our application containers

16. What is Namespace Quota in kubernetes?

- When we have multiple namespaces in shared clusters, it is our responsibility to restrict resource usage of namespaces; else there might be chance of consuming all resources by single namespace.
- To avoid such situation Namespace Quota came into picture.
- Whenever any new application come at kuebenates admin for hosting in shared cluster, generally they ask for below details
    - **Object count**: how many deployment/pods/controller & service will run?
    - **Resource utilization**: how much CPU & RAM require for them?
- Based on above question they put limit on namespace, that limit is called as resource quota.
- A resource quota, defined by a ResourceQuota object, provides constraints that limit aggregate resource consumption per namespace.
- It can limit the quantity/number of objects that can be created in a namespace by type, as well as the total amount of compute resources that may be consumed by resources in that project.

Object count quota:

- We can limit the quantity/number of objects that can be created in a namespace by below type : pods, services, replicationcontrollers, configmaps, persistentvolumeclaims, resourcequotas, services.loadbalancers, services.nodeports, secrets
- Let's take example below YML(objquota.yml) in which we have specified limit on object as below on our namespace "prod"
- pods: "4", replicationcontrollers: "2", services: "10", count/deployments.apps: "2", count/replicasets.apps: "2"

```
apiVersion: v1
kind: Namespace
metadata:
  name: prod
  labels:
    name: production
---
apiVersion : v1
kind: ResourceQuota
metadata:
  name: object-counts
  namespace: prod
spec:
  hard:
    pods: "4"
    replicationcontrollers: "2"
```

```
    services: "10"
    count/deployments.apps: "2"
    count/replicasets.apps: "2"
```

root@manager-1: ~

```
apiVersion: v1
kind: Namespace
metadata:
  name: prod
  labels:
    name: production
---
apiVersion : v1
kind: ResourceQuota
metadata:
  name: object-counts
  namespace: prod
spec:
  hard:
    pods: "4"
    replicationcontrollers: "2"
    services: "10"
    count/deployments.apps: "2"
    count/replicasets.apps: "2"
```

- We can create this quota using below command
- Syntax: > kubectl create -f objquota.yml
- We can check quota of our namespace using below command
- Syntax: > kubectl get quota -n <name of namespace>
- Ex :> kubectl get quota -n prod

root@manager-1: ~

```
root@manager-1:~# kubectl create -f objquota.yml
namespace/prod created
resourcequota/object-counts created
root@manager-1:~# kubectl get quota -n prod
NAME            AGE     REQUEST                                                                                      LIMIT
object-counts   4m11s   count/deployments.apps: 0/2, count/replicasets.apps: 0/2, pods: 0/4, replicationcontrollers: 0/2, services: 0/10
```

- Now we can check by deployment over given limit i.e. 2 using below command
- Syntax:> kubectl create deploy dep1 --image nginx -n prod
- Syntax:> kubectl create deploy dep2 --image nginx -n prod
- We can check quota after creating above 2 deployment, it show expected & allowed number of deployment count are equal i.e. 2
- Now if we create one more deployment, then we got error for quota limit exceed
- Syntax:> kubectl create deploy dep3 --image nginx -n prod

root@manager-1: ~

```
root@manager-1:~# kubectl get quota -n prod
NAME            AGE   REQUEST                                                                                      LIMIT
object-counts   11m   count/deployments.apps: 0/2, count/replicasets.apps: 0/2, pods: 0/4, replicationcontrollers: 0/2, services: 0/10
root@manager-1:~# kubectl create deploy dep1 --image nginx -n prod
deployment.apps/dep1 created
root@manager-1:~# kubectl get quota -n prod
NAME            AGE   REQUEST                                                                                      LIMIT
object-counts   14m   count/deployments.apps: 1/2, count/replicasets.apps: 1/2, pods: 1/4, replicationcontrollers: 0/2, services: 0/10
root@manager-1:~# kubectl create deploy dep2 --image nginx -n prod
deployment.apps/dep2 created
root@manager-1:~# kubectl get quota -n prod
NAME            AGE   REQUEST                                                                                      LIMIT
object-counts   14m   count/deployments.apps: 2/2, count/replicasets.apps: 2/2, pods: 2/4, replicationcontrollers: 0/2, services: 0/10
root@manager-1:~# kubectl create deploy dep3 --image nginx -n prod
error: failed to create deployment: deployments.apps "dep3" is forbidden: exceeded quota: object-counts, requested: count/deployments.apps=1, used: count/deployments.apps=2
, limited: count/deployments.apps=2
root@manager-1:~#
```

- Here we can also increase number of pod replica to check limit exceed, we have limit of pod as 4 & if we increase this to 5, using below command than it will create only up to 4 pods.
- Syntax:> kubectl scale deploy dep1 --replicas 5 -n prod

root@manager-1: ~

```
root@manager-1:~# kubectl get quota -n prod
NAME            AGE   REQUEST                                                                                      LIMIT
object-counts   36m   count/deployments.apps: 2/2, count/replicasets.apps: 2/2, pods: 2/4, replicationcontrollers: 0/2, services: 0/10
root@manager-1:~# kubectl scale deploy dep1 --replicas 5 -n prod
deployment.apps/dep1 scaled
root@manager-1:~# kubectl get quota -n prod
NAME            AGE   REQUEST                                                                                      LIMIT
object-counts   38m   count/deployments.apps: 2/2, count/replicasets.apps: 2/2, pods: 4/4, replicationcontrollers: 0/2, services: 0/10
root@manager-1:~# kubectl get deploy -n prod
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
dep1   3/5     3            3           24m
dep2   1/1     1            1           24m
```

- Here we can also change quota limit using below command
- Syntax:> kubectl edit quota -n <name of namespace>
- Ex:> kubectl edit quota -n prod
- After executing above command we can change quota limit, save & check for limit change

```
root@manager-1: ~
root@manager-1:~# kubectl get quota -n prod
NAME            AGE   REQUEST                                                                                 LIMIT
object-counts   59m   count/deployments.apps: 2/2, count/replicasets.apps: 2/2, pods: 4/4, replicationcontrollers: 0/2, services: 0/10
root@manager-1:~# kubectl edit quota -n prod
```

```
root@manager-1: ~
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: ResourceQuota
metadata:
  creationTimestamp: "2024-01-28T16:56:26Z"
  name: object-counts
  namespace: prod
  resourceVersion: "850283"
  uid: a307fdf4-c487-41a0-9bbc-f8bb31c5f7fa
spec:
  hard:
    count/deployments.apps: "2"
    count/replicasets.apps: "2"
    pods: "6"
    replicationcontrollers: "2"
    services: "10"
status:
  hard:
    count/deployments.apps: "2"
    count/replicasets.apps: "2"
    pods: "4"
    replicationcontrollers: "2"
    services: "10"
  used:
    count/deployments.apps: "2"
    count/replicasets.apps: "2"
    pods: "4"
    replicationcontrollers: "0"
    services: "0"
~
resourcequota/object-counts edited
root@manager-1:~# kubectl get quota -n prod
NAME            AGE   REQUEST                                                                                 LIMIT
object-counts   65m   count/deployments.apps: 2/2, count/replicasets.apps: 2/2, pods: 4/6, replicationcontrollers: 0/2, services: 0/10
root@manager-1:~#
```

Resource utilization Quota

- We can limit total amount of compute resources that may be consumed by resources in that project.
- Let's take example below YML(resquota.yml) in which we have specified limit on CPU as **cpu: "2"** & RAM as **memory: 2Gi** on namespace "team1"

```
apiVersion: v1
kind: Namespace
metadata:
  name: team1
  labels:
    name: development
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-demo
  namespace: team1
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
```

```
    limits.memory: 2Gi
```

root@manager-1: ~

```
apiVersion: v1
kind: Namespace
metadata:
  name: team1
  labels:
    name: development
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources-demo
  namespace: team1
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

- We can create this namespace using below command
- `Syntax:> kubectl create -f rsquota.yml`
- We can check quota liming using below command
- `Syntax:> kubectl get quota -n team1`

root@manager-1: ~

```
root@manager-1:~# vim rsquota.yml
root@manager-1:~# kubectl create -f rsquota.yml
namespace/team1 created
resourcequota/compute-resources-demo created
root@manager-1:~# kubectl get quota -n team1
NAME                    AGE     REQUEST                                      LIMIT
compute-resources-demo   2m43s   requests.cpu: 0/1, requests.memory: 0/1Gi   limits.cpu: 0/2, limits.memory: 0/2Gi
root@manager-1:~#
```

17. What is Service in kubernetes?

- When we try to access any container IP from external machine, kubernetes will not allow accessing it from outside of kuberenetes cluster, but it can be accessible from any node of kubernetes cluster i.e. Manager/worker.
- Pod IP & port will not work from outside, but it will work within kubernetes cluster because its scope is within kubernetes network.

root@manager-1: /

```
root@manager-1:/# kubectl run pod1 --image nginx
pod/pod1 created
root@manager-1:/# kubectl get pods -o wide
NAME   READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
pod1   1/1     Running   0          23s   192.168.226.65   worker-1   <none>           <none>
root@manager-1:/#
```

kubernetes/1-intall/install-kube | VM instances – Compute Engin | 192.168.226.65

192.168.226.65

This site can't be reached

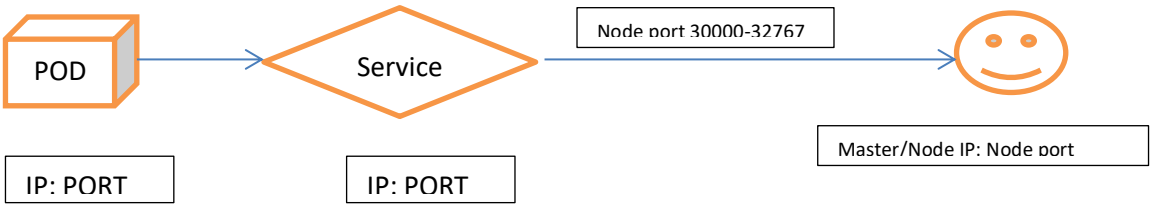
192.168.226.65 took too long to respond.

```
root@manager-1:/# curl 192.168.226.65
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

- So to overcome above situation service concept came into picture in kubernetes.
- Service is an object, which provides you feature to access container from outside kubernetes or from inside kubernetes cluster.
- When we create service, it maps to POD.
- Service is having its own IP & port, but this also works internal to kubernetes only.
- There are 2 type of service available
    - NodePort service
    - ClusterIP service

NodePort Service:

- Service publishes **node port**; using node port & IP of master or worker, end user can access application outside of kubernetes.
- This node port ranges from 30000 to 32767

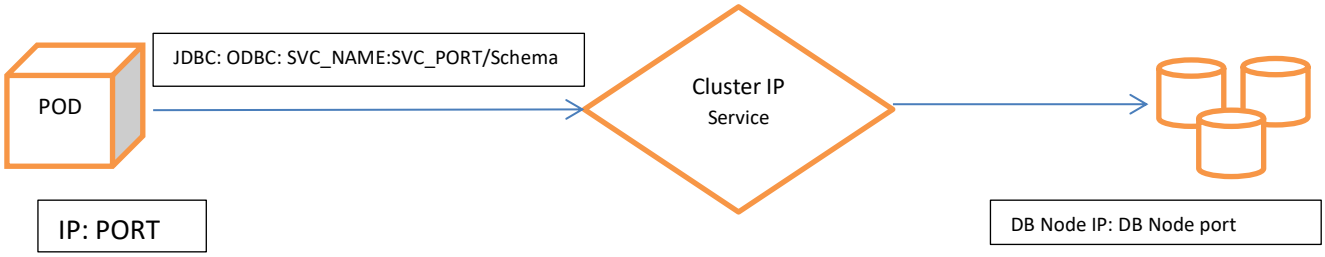


- We can use below command to create service
- `Syntax:>kubectl expose pod <pod name> --name <service name> --port <service port> --target-port <port of container of POD> --type NodePort`
- `Ex:> kubectl expose pod pod1 --name svc1 --port 1234 --target-port 80 --type NodePort`
- We can use below command to list services
- `Synatx:> kubectl get svc`
- `Synatx:> kubectl get service`
- We can access application from browser using Master/Node(worker) IP : NodePort
- `Manager:→ http://34.72.240.122:31501/`
- `Worker:→ http:// 34.27.135.157:31501`

```
root@manager-1:/
root@manager-1:/# kubectl expose pod pod1 --name svc1 --port 1234 --target-port 80 --type NodePort
service/svc1 exposed
root@manager-1:/# kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP          100m
svc1          NodePort    10.97.80.109    <none>        1234:31501/TCP   114s
root@manager-1:/# kubectl get service
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP          100m
svc1          NodePort    10.97.80.109    <none>        1234:31501/TCP   119s
root@manager-1:/#
```

ClusterIP Service:

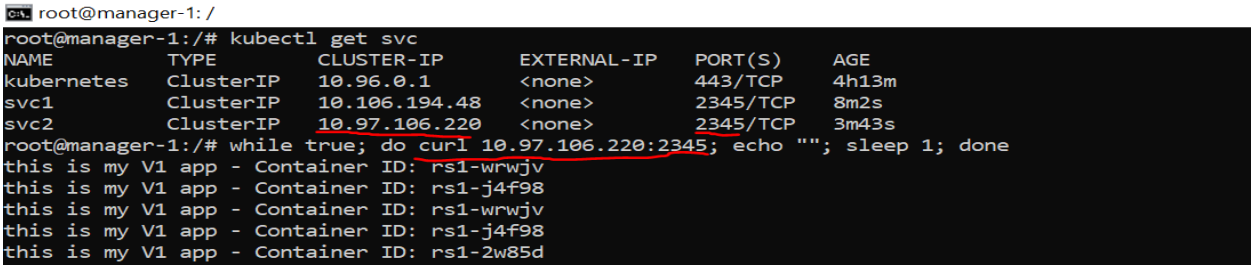
- When there is requirement to connect one application with another application within same cluster like web application connects to DB application which running on different pods, we need to use ClusterIP service.
- To connect to DB from application we use connection string which is having DB IP & DB port, in general we give HOST Name instead of IP, but if DB application is running in different pods, it is difficult to give DB IP/Name. So to deal with this issue we need to use cluster IP service.



- We can create cluster IP service using below command.
- `Syntax:> kubectl expose rs <name of replica set> --name <service name> --port <service port> --target-port <application PORT> --type ClusterIP`
- `Ex:> kubectl expose rs rs1 --name svc1 --port 2345 --target-port 3000 --type ClusterIP`
- IF we do not give any type by default also it will create cluster IP service only
- `Syntax:> kubectl expose rs <name of replica set> --name <service name> --port <service port> --target-port <application PORT>`
- `Ex:> kubectl expose rs rs1 --name svc2 --port 2345 --target-port 3000`

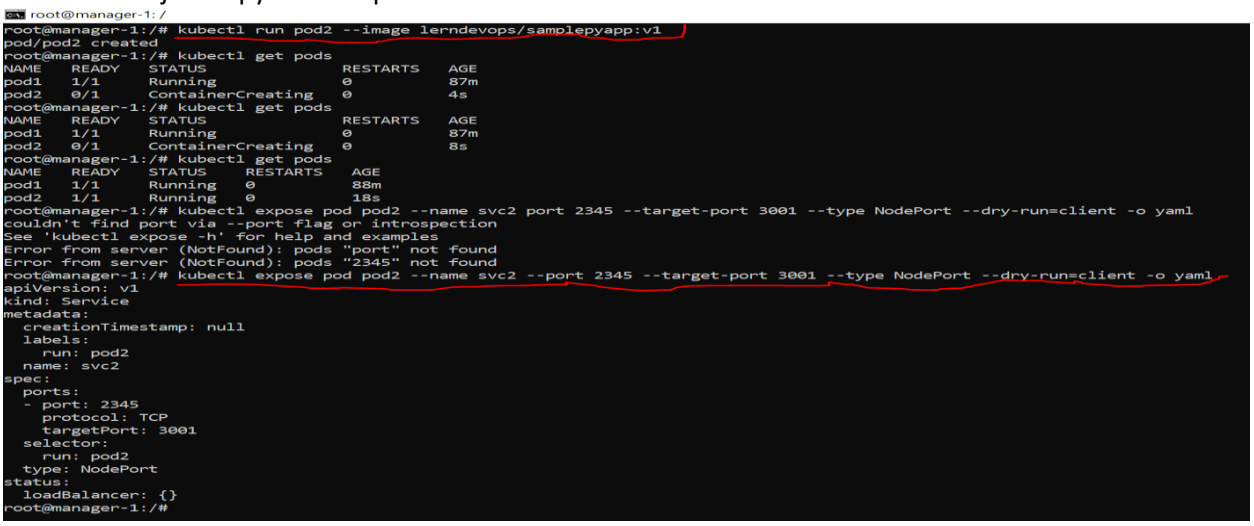
```
root@manager-1: /
root@manager-1:/# kubectl expose rs rs1 --name svc1 --port 2345 --target-port 3000 --type ClusterIP
service/svc1 exposed
root@manager-1:/# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    4h5m
svc1         ClusterIP   10.106.194.48   <none>        2345/TCP   15s
root@manager-1:/# kubectl expose rs rs1 --name svc3 --port 2345 --target-port 3000
service/svc3 exposed
root@manager-1:/# kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    4h9m
svc1         ClusterIP   10.106.194.48    <none>        2345/TCP   4m46s
svc2         ClusterIP   10.97.106.220    <none>        2345/TCP   27s
svc3         ClusterIP   10.108.154.124   <none>        2345/TCP   3s
```

- Now we can directly access this service from kubernates cluster
- Syntax:> `while true; do curl 10.97.106.220:2345; echo ""; sleep 1; done`



18. How can we give our own NodePort to service?

- We can give our own NodePort to access application by specifying in YML file for service creation using below steps.
- Support we want to deploy on python application (lerndevops/samplepyapp) which run on 3000 port, so first we need to run pod of that image
- Ex:> `kubectl run pod2 --image lerndevops/samplepyapp:v1`
- Now using dry-run we can get YML file for service creation by using below command
- **Syntax:>** `kubectl expose pod <pod name> --name <service name> --port <service port> --target-port <application port> --type NodePort --dry-run=client -o yaml`
- **Ex:>** `kubectl expose pod pod2 --name svc2 --port 2345 --target-port 3001 --type NodePort --dry-run=client -o yaml`
- Now just copy YML output & create new YML file



- In YML file we can specify our own highlighted node port as below & create YML File (s.yml)

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    run: pod2
  name: svc2
spec:
  ports:
  - port: 2345
    protocol: TCP
    targetPort: 3000
    nodePort: 30001
```

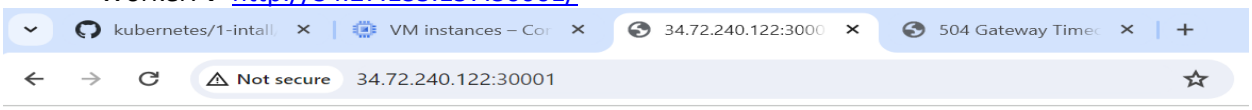
```
  selector:
    run: pod2
  type: NodePort
status:
  loadBalancer: {}
```

- Now create service using below command
- Ex:> kubectl create -f s.yml

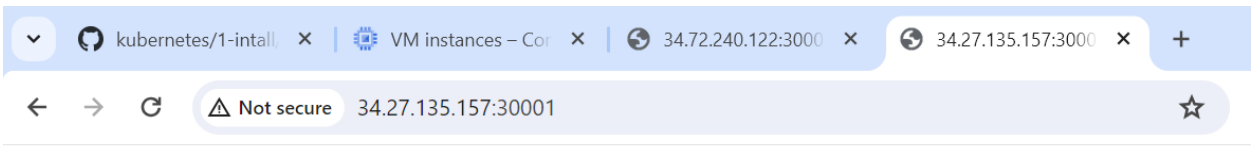
```
root@manager-1: /
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    run: pod2
  name: svc2
spec:
  ports:
  - port: 2345
    protocol: TCP
    targetPort: 3000
    nodePort: 30001
  selector:
    run: pod2
  type: NodePort
```

```
root@manager-1:/# kubectl create -f s.yml
service/svc2 created
root@manager-1:/# kubectl get svc -o wide
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE     SELECTOR
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          3h26m   <none>
svc1         NodePort    10.97.80.109   <none>        1234:31501/TCP   107m    run=pod1
svc2         NodePort    10.109.2.55    <none>        2345:30001/TCP   11s     run=pod2
```

- Now we can access this application using our own port 30001 & IP of manager/worker(nodes)
- Manager : →http://34.72.240.122:30001/
- Worker:→ http://34.27.135.157:30001/



this is my V1 app - Container ID: pod2



this is my V1 app - Container ID: pod2

- We can access this application from manager/worker(node) by using service IP & port

```
root@manager-1:/# kubectl get svc -o wide
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE     SELECTOR
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          3h26m   <none>
svc1         NodePort    10.97.80.109   <none>        1234:31501/TCP   107m    run=pod1
svc2         NodePort    10.109.2.55    <none>        2345:30001/TCP   11s     run=pod2
root@manager-1:/# vi s.yml
root@manager-1:/# curl 10.109.2.55:2345
this is my V1 app - Container ID: pod2root@manager-1:/#
```

```
root@worker-1: /
root@worker-1:/# curl 10.109.2.55:2345
this is my V1 app - Container ID: pod2root@worker-1:/#
```

19. What is controller in kubernetes? Why do we need controller?

- To achieve high availability & work load manage single pod is not sufficient. So we need to run multiple pods at a time.
- This is where controller came into picture.
- Controller helps us to run multiple containers, achieve workload management, high availability, scale up-scale down containers, etc.
- There are many controller in kubernetes
    - Replication controller(RC)
    - Replica set
    - Deployment
    - Daemon set
    - Job
    - Cron job
    - State full set
    - HPA

20) What is ReplicationController in Kubernrtes?

- A ReplicationController ensures that a specified number of pod replicas are running at any one time.
- In other words, a ReplicationController makes sure that a pod or a homogeneous set of pods is always up and available.
- If there are too many pods, the ReplicationController terminates the extra pods. If there are too few, the ReplicationController starts more pods.
- Unlike manually created pods, the pods maintained by a ReplicationController are automatically replaced if they fail, are deleted, or are terminated.
- Now days this controller is deprecated.
- We can create replication controller as below rc.yml

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: k-admins
spec:
  replicas: 4
  selector:
    skill: kube
  template:
    metadata:
      name: nginx
      labels:
        skill: kube
    spec:
      containers:
        - name: one
          image: nginx
```

- Here we have created ReplicationController having name "`k-admins`", having 4 number of pods, with selection criteria who have label `skill: kube`, ReplicationController tries to find pod with matching criteria, if any pod with this criteria is not available ReplicationController will create pod with given criteria.
- Now we can create by using below command:
- `Syntax:> kubectl create -f rc.yml`

```
root@manager-1:/
root@manager-1:/# kubectl create -f rc.yml
replicationcontroller/k-admins created
root@manager-1:/# kubectl get rc
NAME       DESIRED   CURRENT   READY   AGE
k-admins   4         4         4       37s
root@manager-1:/# kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
k-admins-47sbv   1/1     Running   0          75s   192.168.226.70   worker-1   <none>           <none>
k-admins-q7xbc   1/1     Running   0          75s   192.168.226.72   worker-1   <none>           <none>
k-admins-qzsqq   1/1     Running   0          75s   192.168.226.71   worker-1   <none>           <none>
k-admins-td5lb   1/1     Running   0          75s   192.168.226.69   worker-1   <none>           <none>
```

- If we try to delete any one of pod using below command, replication controller will recreate another pod by using desired state
- `Syntax:> kubectl delete pod k-admins-47sbv`

```
root@manager-1:/
root@manager-1:/# kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
k-admins-47sbv   1/1     Running   0          14m   192.168.226.70   worker-1   <none>           <none>
k-admins-q7xbc   1/1     Running   0          14m   192.168.226.72   worker-1   <none>           <none>
k-admins-qzsqq   1/1     Running   0          14m   192.168.226.71   worker-1   <none>           <none>
k-admins-td5lb   1/1     Running   0          14m   192.168.226.69   worker-1   <none>           <none>
root@manager-1:/# kubectl delete pod k-admins-47sbv
pod "k-admins-47sbv" deleted
root@manager-1:/# kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
k-admins-jr4jb   1/1     Running   0          5s    192.168.226.73   worker-1   <none>           <none>
k-admins-q7xbc   1/1     Running   0          15m   192.168.226.72   worker-1   <none>           <none>
k-admins-qzsqq   1/1     Running   0          15m   192.168.226.71   worker-1   <none>           <none>
k-admins-td5lb   1/1     Running   0          15m   192.168.226.69   worker-1   <none>           <none>
```

- We can change number of replicas on the fly i.e. we can increase or decrease number of replicas using below command
- Syntax: kubectl scale rc <replication controller name> --replicas <count>
- Ex:> kubectl scale rc k-admins --replicas 2
- Ex:> kubectl scale rc k-admins --replicas 6

```
root@manager-1:/
root@manager-1:/# kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE    IP              NODE       NOMINATED NODE   READINESS GATES
k-admins-jr4jb   1/1     Running   0          7m43s  192.168.226.73   worker-1   <none>           <none>
k-admins-q7xbc   1/1     Running   0          22m    192.168.226.72   worker-1   <none>           <none>
k-admins-qzsqq   1/1     Running   0          22m    192.168.226.71   worker-1   <none>           <none>
k-admins-td5lb   1/1     Running   0          22m    192.168.226.69   worker-1   <none>           <none>
root@manager-1:/# kubectl scale rc k-admins --replicas 2
replicationcontroller/k-admins scaled
root@manager-1:/# kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
k-admins-qzsqq   1/1     Running   0          23m   192.168.226.71   worker-1   <none>           <none>
k-admins-td5lb   1/1     Running   0          23m   192.168.226.69   worker-1   <none>           <none>
root@manager-1:/# kubectl scale rc k-admins --replicas 6
replicationcontroller/k-admins scaled
root@manager-1:/# kubectl get pods -o wide
NAME             READY   STATUS              RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
k-admins-26rkq   0/1     ContainerCreating   0          4s    <none>          worker-1   <none>           <none>
k-admins-dgsj2   0/1     ContainerCreating   0          4s    <none>          worker-1   <none>           <none>
k-admins-hzkfj   0/1     ContainerCreating   0          4s    <none>          worker-1   <none>           <none>
k-admins-kb8vv   0/1     ContainerCreating   0          4s    <none>          worker-1   <none>           <none>
k-admins-qzsqq   1/1     Running             0          25m   192.168.226.71   worker-1   <none>           <none>
k-admins-td5lb   1/1     Running             0          25m   192.168.226.69   worker-1   <none>           <none>
root@manager-1:/# kubectl get pods -o wide
NAME             READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
k-admins-26rkq   1/1     Running   0          28s   192.168.226.75   worker-1   <none>           <none>
k-admins-dgsj2   1/1     Running   0          28s   192.168.226.76   worker-1   <none>           <none>
k-admins-hzkfj   1/1     Running   0          28s   192.168.226.77   worker-1   <none>           <none>
k-admins-kb8vv   1/1     Running   0          28s   192.168.226.74   worker-1   <none>           <none>
k-admins-qzsqq   1/1     Running   0          25m   192.168.226.71   worker-1   <none>           <none>
k-admins-td5lb   1/1     Running   0          25m   192.168.226.69   worker-1   <none>           <none>
```

21) What is ReplicaSet in kubernetes?

- It is advanced version of ReplicationController, which is having all features of ReplicationController with some extra features.
- In ReplicationController we can give only single match criteria, but in ReplicaSet we can have multiple selection criteria which can be defined under **`matchLabels, matchExpression`** tag.
- We can give complex expression in selection criteria using **`matchExpression`** tag.
- We can create ReplicaSet by using below YAML file i.e. rs.yml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs1
spec:
  replicas: 3
  selector:
    matchLabels:
```

```
      app: sanapp
  template:
    metadata:
      labels:
        app: sanapp
    spec:
      containers:
        - name: abc
          image: lerndevops/samplepyapp:v1
```

- Here we have created ReplicaSet with name “rs1”, having 3 number of replica with selection criteria as label app: sanapp
- Now we can create ReplicaSet using below command
- Syntax:> kubectl create -f rs.yml

root@manager-1: /

```
root@manager-1:/# kubectl create -f rs.yml
replicaset.apps/rs1 created
root@manager-1:/# kubectl get rs
NAME   DESIRED   CURRENT   READY   AGE
rs1    3         3         3       16s
root@manager-1:/# kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
rs1-2vgxd   1/1     Running   0          26s   192.168.226.80   worker-1   <none>           <none>
rs1-c47f7   1/1     Running   0          26s   192.168.226.78   worker-1   <none>           <none>
rs1-xnks5   1/1     Running   0          26s   192.168.226.79   worker-1   <none>           <none>
root@manager-1:/# kubectl describe rs rs1
Name:         rs1
Namespace:    default
Selector:     app=sanapp
Labels:       <none>
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=sanapp
  Containers:
   abc:
    Image:        lerndevops/samplepyapp:v1
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  68s   replicaset-controller  Created pod: rs1-c47f7
  Normal  SuccessfulCreate  68s   replicaset-controller  Created pod: rs1-2vgxd
  Normal  SuccessfulCreate  68s   replicaset-controller  Created pod: rs1-xnks5
```

- Now if we delete all pods using below command, ReplicaSet will automatically create number of pod as defined in replicas
- Synatx:>kubectl delete pods –all

root@manager-1: /

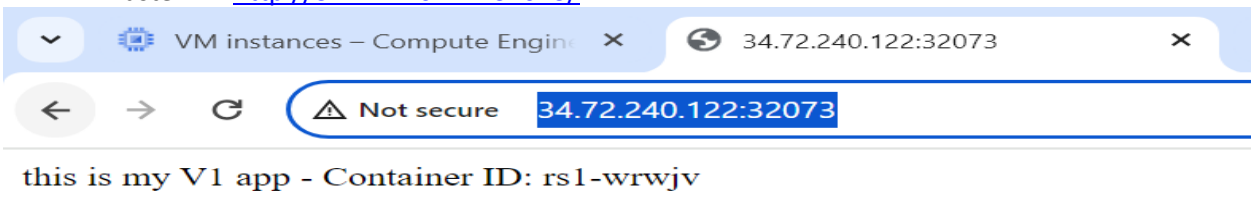
```
root@manager-1:/# kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE    IP               NODE       NOMINATED NODE   READINESS GATES
rs1-2vgxd   1/1     Running   0          9m2s   192.168.226.80   worker-1   <none>           <none>
rs1-c47f7   1/1     Running   0          9m2s   192.168.226.78   worker-1   <none>           <none>
rs1-xnks5   1/1     Running   0          9m2s   192.168.226.79   worker-1   <none>           <none>
root@manager-1:/# kubectl delete pods --all
pod "rs1-2vgxd" deleted
pod "rs1-c47f7" deleted
pod "rs1-xnks5" deleted
root@manager-1:/# kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
rs1-2w85d   1/1     Running   0          6s    192.168.226.81   worker-1   <none>           <none>
rs1-j4f98   1/1     Running   0          6s    192.168.226.82   worker-1   <none>           <none>
rs1-wrwjv   1/1     Running   0          7s    192.168.226.83   worker-1   <none>           <none>
root@manager-1:/#
```

- Now we can expose this replica set using service concept with below command

- Syntax:> kubeclt expose rs <name of replica set> --name <service name> --port <new port> --target-port <application port> --type NodePort
- Ex> kubectl expose rs rs1 --name svc1 --port 1234 --target-port 3000 --type NodePort

```
root@manager-1: /
root@manager-1:/# kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
rs1-2w85d   1/1     Running   0          24m   192.168.226.81  worker-1   <none>           <none>
rs1-j4f98   1/1     Running   0          24m   192.168.226.82  worker-1   <none>           <none>
rs1-wrwjv   1/1     Running   0          24m   192.168.226.83  worker-1   <none>           <none>
root@manager-1:/# kubectl expose rs rs1 --name svc1 --port 1234 --target-port 3000 --type NodePort
service/svc1 exposed
root@manager-1:/# kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP          50m
svc1         NodePort    10.99.61.14    <none>        1234:32073/TCP   15s
root@manager-1:/#
```

- We can access application using service port & any of IP of master/worker(node)
- Master :→ http://34.72.240.122:32073/



- Worker:→ http://34.27.135.157:32073/

- We can do scale update & down for ReplicaSet using below command
- `Syntax: kubectl scale rs <name of replicaset > --replicas <count>`
- `Ex:> kubectl scale rs rs1 --replicas 6`
- `Ex:> kubectl scale rs rs1 --replicas 2`

```
root@manager-1:/# kubectl scale rs rs1 --replicas 6
replicaset.apps/rs1 scaled
root@manager-1:/# kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
rs1-2w85d   1/1     Running   0          40m   192.168.226.81  worker-1   <none>           <none>
rs1-bjlng   1/1     Running   0          9s    192.168.226.84  worker-1   <none>           <none>
rs1-bnts7   1/1     Running   0          9s    192.168.226.86  worker-1   <none>           <none>
rs1-j4f98   1/1     Running   0          40m   192.168.226.82  worker-1   <none>           <none>
rs1-qw6sn   1/1     Running   0          9s    192.168.226.85  worker-1   <none>           <none>
rs1-wrwjv   1/1     Running   0          40m   192.168.226.83  worker-1   <none>           <none>
root@manager-1:/# kubectl scale rs rs1 --replicas 3
replicaset.apps/rs1 scaled
root@manager-1:/# kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
rs1-2w85d   1/1     Running   0          40m   192.168.226.81  worker-1   <none>           <none>
rs1-j4f98   1/1     Running   0          40m   192.168.226.82  worker-1   <none>           <none>
rs1-wrwjv   1/1     Running   0          40m   192.168.226.83  worker-1   <none>           <none>
root@manager-1:/#
```

- We can also access application from kubernete cluster i.e. master/worker node using service IP & service port as well

```
root@manager-1:/# kubectl get svc
NAME         TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1     <none>        443/TCP          75m
svc1         NodePort    10.99.61.14   <none>        1234:32073/TCP   24m
root@manager-1:/# while true; do curl 10.99.61.14:1234; echo ""; sleep 1; done
this is my V1 app - Container ID: rs1-wrwjv
this is my V1 app - Container ID: rs1-j4f98
this is my V1 app - Container ID: rs1-2w85d
this is my V1 app - Container ID: rs1-j4f98
this is my V1 app - Container ID: rs1-wrwjv
```

root@worker-1: /

```
root@worker-1:/# while true; do curl 10.99.61.14:1234; echo ""; sleep 1; done
this is my V1 app - Container ID: rs1-j4f98
this is my V1 app - Container ID: rs1-j4f98
this is my V1 app - Container ID: rs1-j4f98
this is my V1 app - Container ID: rs1-2w85d
this is my V1 app - Container ID: rs1-wrwjv
```

22) How we can update application without down time? What is use of deployment controller?

- By using deployment controller we can achieve 0 down time deployment.
- You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.
- There are 2 deployment models we can use
  - Blue green deployment
  - Rolling updates deployment

23) How we can update application with blue-green deployment?

- To explain this first create on application & deploy that using deployment controller as below
- Create deployment with name `mydeployment` using below YML (dc1.yml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mydeployment
  labels:
    app: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-count
        image: lerndevops/samplepyapp:v1
        ports:
        - containerPort: 3000
```

- Do deployment using below command:
- `Syntax:> kubectl create -f dc.yml`
- We can also give repository raw URL path of YML file as below

- Syntax:> kubectl create -f https://raw.githubusercontent.com/lerndevops/kubernetes/master/3-controllers/deployments/deployment-ex1.yml
- Here even if delete any pod it will comes back, i.e. It will also provide you high availability

root@manager-1: ~

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mydeployment
  labels:
    app: myapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-count
        image: lerndevops/samplepyapp:v1
        ports:
        - containerPort: 3000
```

root@manager-1: ~

```
root@manager-1:~# vim dc.yml
root@manager-1:~# kubectl create -f dc.yml
deployment.apps/mydeployment created
root@manager-1:~# kubectl get deployment
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
mydeployment   3/3     3            3           14s
root@manager-1:~# kubectl get pod -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
mydeployment-b48f5b864-4shfx    1/1     Running   0          33s   192.168.226.94   worker-1   <none>           <none>
mydeployment-b48f5b864-qk9jp    1/1     Running   0          33s   192.168.226.95   worker-1   <none>           <none>
mydeployment-b48f5b864-vfdl7    1/1     Running   0          33s   192.168.226.96   worker-1   <none>           <none>
root@manager-1:~# kubectl delete pod mydeployment-b48f5b864-4shfx
pod "mydeployment-b48f5b864-4shfx" deleted
root@manager-1:~# kubectl get pod -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
mydeployment-b48f5b864-qk9jp    1/1     Running   0          59s   192.168.226.95   worker-1   <none>           <none>
mydeployment-b48f5b864-vfdl7    1/1     Running   0          59s   192.168.226.96   worker-1   <none>           <none>
mydeployment-b48f5b864-xvzcf    1/1     Running   0          7s    192.168.226.97   worker-1   <none>           <none>
```

- Now expose this deployment through service with below command
- Syntax:> kubectl expose deploy <name of deployment> --name <service name> --port <new port> --target-port <application port>
- Syntax:> kubectl expose deploy mydeployment --name svc1 --port 1234 --target-port 3000

root@manager-1: ~

```
root@manager-1:~# kubectl get deployment
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
mydeployment   3/3     3            3           4m43s
root@manager-1:~# kubectl expose deploy mydeployment --name svc1 --port 1234 --target-port 3000
service/svc1 exposed
root@manager-1:~# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    9h
svc1         ClusterIP   10.96.202.254   <none>        1234/TCP   23s
```

- We can access this application using master/node with below command

root@worker-1: ~

```
root@worker-1:~# while true; do curl 10.96.202.254:1234; sleep 1; echo ""; done
this is my V1 app - Container ID: mydeployment-b48f5b864-vfdl7
this is my V1 app - Container ID: mydeployment-b48f5b864-qk9jp
this is my V1 app - Container ID: mydeployment-b48f5b864-qk9jp
this is my V1 app - Container ID: mydeployment-b48f5b864-vfdl7
this is my V1 app - Container ID: mydeployment-b48f5b864-qk9jp
this is my V1 app - Container ID: mydeployment-b48f5b864-qk9jp
this is my V1 app - Container ID: mydeployment-b48f5b864-vfdl7
this is my V1 app - Container ID: mydeployment-b48f5b864-xvzcf
```

- We can see load balancing is happening on all pods.
- Now end user is accessing this application happily & we want to update this to new version, to do this, we create one more deployment with name `green-version` using another YAML file (dc2.yml)

```
apiVersion: apps/v1
```

```
kind: Deployment
metadata:
  name: green-version
  labels:
    app: green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: green
  template:
    metadata:
      labels:
        app: green
    spec:
      containers:
      - name: myapp-cont2
        image: lerndevops/samplepyapp:v2
        ports:
        - containerPort: 3000
```

root@manager-1: ~

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: green-version
  labels:
    app: green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: green
  template:
    metadata:
      labels:
        app: green
    spec:
      containers:
      - name: myapp-cont2
        image: lerndevops/samplepyapp:v2
        ports:
        - containerPort: 3000
```

- Create deployment using below command
- `Syntax:> kubectl create -f dc2.yml`
- We can see that both deployment pods are running

root@manager-1: ~

```
root@manager-1:~# vim dc2.yml
root@manager-1:~# kubectl create -f dc2.yml
deployment.apps/green-version created
root@manager-1:~# kubectl get pods -o wide
NAME                              READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
green-version-5d6f77c778-bdv9r    1/1     Running   0          22s   192.168.226.98  worker-1   <none>           <none>
green-version-5d6f77c778-lgt8v    1/1     Running   0          22s   192.168.226.99  worker-1   <none>           <none>
green-version-5d6f77c778-pk68b    1/1     Running   0          22s   192.168.226.100 worker-1   <none>           <none>
mydeployment-b48f5b864-qk9jp      1/1     Running   0          48m   192.168.226.95  worker-1   <none>           <none>
mydeployment-b48f5b864-vfdl7      1/1     Running   0          48m   192.168.226.96  worker-1   <none>           <none>
mydeployment-b48f5b864-xvzcf      1/1     Running   0          47m   192.168.226.97  worker-1   <none>           <none>
root@manager-1:~#
```

- Now we can edit service to update new application version deployment as below command
- `Syntax:> kubectl edit svc svc1`

root@manager-1: ~

```
root@manager-1:~# kubectl get pod -o wide
NAME                              READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
green-version-5d6f77c778-bdv9r    1/1     Running   0          49m   192.168.226.98  worker-1   <none>           <none>
green-version-5d6f77c778-lgt8v    1/1     Running   0          49m   192.168.226.99  worker-1   <none>           <none>
green-version-5d6f77c778-pk68b    1/1     Running   0          49m   192.168.226.100 worker-1   <none>           <none>
mydeployment-b48f5b864-qk9jp      1/1     Running   0          97m   192.168.226.95  worker-1   <none>           <none>
mydeployment-b48f5b864-vfdl7      1/1     Running   0          97m   192.168.226.96  worker-1   <none>           <none>
mydeployment-b48f5b864-xvzcf      1/1     Running   0          96m   192.168.226.97  worker-1   <none>           <none>
root@manager-1:~# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    11h
svc1         ClusterIP   10.96.202.254   <none>        1234/TCP   92m
root@manager-1:~# kubectl edit svc svc1
 32L, 732C written
service/svc1 edited
```

root@manager-1: ~

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2024-01-27T04:06:20Z"
  labels:
    app: myapp
  name: svc1
  namespace: default
  resourceVersion: "659957"
  uid: 90bcda5d-1704-472f-8f64-984a368a0f61
spec:
  clusterIP: 10.96.202.254
  clusterIPs:
  - 10.96.202.254
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - port: 1234
    protocol: TCP
    targetPort: 3000
  selector:
    app: green
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

- Once we update service we can see those new upcoming requests are going to new deployment pods i.e. `green-version` pods

root@worker-1: ~

```
this is my V1 app - Container ID: mydeployment-b48f5b864-qk9jp
this is my V1 app - Container ID: mydeployment-b48f5b864-vfdl7
this is my V1 app - Container ID: mydeployment-b48f5b864-xvzcf
this is my V1 app - Container ID: mydeployment-b48f5b864-vfdl7
this is my V1 app - Container ID: mydeployment-b48f5b864-qk9jp
this is my V1 app - Container ID: mydeployment-b48f5b864-vfdl7
this is my V2 app - Container ID: green-version-5d6f77c778-bdv9r
this is my V2 app - Container ID: green-version-5d6f77c778-bdv9r
this is my V2 app - Container ID: green-version-5d6f77c778-pk68b
this is my V2 app - Container ID: green-version-5d6f77c778-pk68b
```

- Now we can delete old deployment as below command

```
root@manager-1:~# kubectl get deploy
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
green-version   3/3     3            3           65m
mydeployment    3/3     3            3           114m
root@manager-1:~# kubectl delete deploy mydeployment
deployment.apps "mydeployment" deleted
```

- Drawback of this blue-green deployment model is that at some point of time we need to install both deployment version pods, which will consume more (double) resources of cluster.
- To overcome this rolling-update deployment model came into picture.

24) How we can update application with rolling-update deployment model?

- In rolling-update deployment model we slowly add new version pods & remove old version pods.
- In this deployment at some point of time request is going in both old & new version pods.
- Here also we don't see any down time for deployment.
- In rolling-update at some point of time we cross desire state of pods, i.e. current version running pods + few more pods, this is called as **surge**.
- In rolling-update we need to defined **strategy** under which we need to mention below
- We can mention max **surge number** also, i.e. number of extra pods with new version to be run, by using keyword **maxSurge**

- Also we can mention max number of old version which we want to delete by using keyword **maxUnavailable**.
- For adding new version pods, we need to specify time gap between 2 pods, so that pod will get properly up & ready to server request, this we can specify using key word **minReadySeconds**.
- Let's create new deployment with name "`kubeserve`" & service with name "`kubeserve-svc`" using single YML file(deploy-ru.yml)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubeserve
spec:
  replicas: 4
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 2
  selector:
    matchLabels:
      app: kubeserve
  template:
    metadata:
      name: kubeserve
      labels:
        app: kubeserve
    spec:
      containers:
      - image: lerndevops/kubeserve:v1
        name: app
---
kind: Service
apiVersion: v1
metadata:
  name: kubeserve-svc
spec:
  type: NodePort
  selector:
    app: kubeserve
  ports:
    - port: 80
      targetPort: 80
```

- Now execute below command to create this
- `Syntax:> kubectl create -f deploy-ru.yml`

root@manager-1: ~

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubeserve
spec:
  replicas: 4
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 2
  selector:
    matchLabels:
      app: kubeserve
  template:
    metadata:
      name: kubeserve
      labels:
        app: kubeserve
    spec:
      containers:
      - image: leaddevops/kubeserve:v1
        name: app
---
kind: Service
apiVersion: v1
metadata:
  name: kubeserve-svc
spec:
  type: NodePort
  selector:
    app: kubeserve
  ports:
    - port: 80
      targetPort: 80
```

root@manager-1: ~

```
root@manager-1:~# vim deploy-ru.yml
root@manager-1:~# kubectl create -f deploy-ru.yml
deployment.apps/kubeserve created
service/kubeserve-svc created
root@manager-1:~# kubectl get deploy
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
kubeserve   0/4     4            0           13s
root@manager-1:~# kubectl get pods
NAME                         READY   STATUS              RESTARTS   AGE
kubeserve-6dcd44d8bd-2psg5   0/1     ContainerCreating   0          23s
kubeserve-6dcd44d8bd-2x276   0/1     ContainerCreating   0          23s
kubeserve-6dcd44d8bd-p6mwz   0/1     ContainerCreating   0          23s
kubeserve-6dcd44d8bd-wr5z5   0/1     ContainerCreating   0          23s
root@manager-1:~# kubectl get pods -o wide
NAME                         READY   STATUS              RESTARTS   AGE   IP       NODE       NOMINATED NODE   READINESS GATES
kubeserve-6dcd44d8bd-2psg5   0/1     ContainerCreating   0          31s   <none>   worker-1   <none>           <none>
kubeserve-6dcd44d8bd-2x276   0/1     ContainerCreating   0          31s   <none>   worker-1   <none>           <none>
kubeserve-6dcd44d8bd-p6mwz   0/1     ContainerCreating   0          31s   <none>   worker-1   <none>           <none>
kubeserve-6dcd44d8bd-wr5z5   0/1     ContainerCreating   0          31s   <none>   worker-1   <none>           <none>
root@manager-1:~# kubectl get pods -o wide
NAME                         READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
kubeserve-6dcd44d8bd-2psg5   1/1     Running   0          49s   192.168.226.125   worker-1   <none>           <none>
kubeserve-6dcd44d8bd-2x276   1/1     Running   0          49s   192.168.226.124   worker-1   <none>           <none>
kubeserve-6dcd44d8bd-p6mwz   1/1     Running   0          49s   192.168.226.122   worker-1   <none>           <none>
kubeserve-6dcd44d8bd-wr5z5   1/1     Running   0          49s   192.168.226.123   worker-1   <none>           <none>
root@manager-1:~# kubectl get svc
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes     ClusterIP   10.96.0.1        <none>        443/TCP        3m27s
kubeserve-svc  NodePort    10.110.126.146   <none>        80:30782/TCP   2m36s
```

- Here 4 pods are running, we can access these by using service IP & service port as below
- `Syntax:> while true; do curl 10.110.126.146:80; sleep 1; echo ""; done`

root@worker-1: ~

```
root@worker-1:~# while true; do curl 10.110.126.146:80; sleep 1; echo ""; done
This is v1 pod kubeserve-6dcd44d8bd-2psg5

This is v1 pod kubeserve-6dcd44d8bd-p6mwz

This is v1 pod kubeserve-6dcd44d8bd-2x276

This is v1 pod kubeserve-6dcd44d8bd-p6mwz

This is v1 pod kubeserve-6dcd44d8bd-wr5z5
```

- Here we can see request is going on each pods, so load balancing is working properly.
- Here we can check deployment status by using below command
- `Syntax:> kubectl rollout status deploy <name of deployment>`

- Ex:> kubectl rollout status deploy kubeserve
- We can check rollout history as well using below command, which will give how many version we have deploy so far.
- Syntax:> kubectl rollout history deploy <name of deployment>
- Ex:> kubectl rollout history deploy kubeserve

root@manager-1: ~

```
root@manager-1:~# kubectl get deploy
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
kubeserve   4/4     4            4           18m
root@manager-1:~# kubectl rollout status deploy kubeserve
deployment "kubeserve" successfully rolled out
root@manager-1:~# kubectl rollout history deploy kubeserve
deployment.apps/kubeserve
REVISION  CHANGE-CAUSE
1         <none>
```

- Here we have deployed just 1st version so it is showing REVISION as 1
- Now we can deploy 2nd version either by editing YML file or we can use below command to do same
- Syntax> kubectl set image deploy <name of deployment> app=<image>:<new version>
- Ex:> kubectl set image deploy kubeserve app=leaddevops/kubeserv:v2

```
root@manager-1:~# kubectl rollout status deploy kubeserve
deployment "kubeserve" successfully rolled out
root@manager-1:~# kubectl rollout history deploy kubeserve
deployment.apps/kubeserve
REVISION  CHANGE-CAUSE
1         <none>

root@manager-1:~# kubectl set image deploy kubeserve app=leaddevops/kubeserve:v2
deployment.apps/kubeserve image updated
root@manager-1:~# kubectl rollout status deploy kubeserve
Waiting for deployment "kubeserve" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "kubeserve" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
deployment "kubeserve" successfully rolled out
root@manager-1:~# kubectl rollout history deploy kubeserve
deployment.apps/kubeserve
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
```

- We can see at end use window customer was getting response from v1 pods, than gradually v2 pods are also giving response in between, after some v1 pods are is completely removed & v2 only started serving request

root@worker-1: ~

```
This is v1 pod kubeserve-6dcd44d8bd-zsjm2
This is v1 pod kubeserve-6dcd44d8bd-zsjm2
This is v1 pod kubeserve-6dcd44d8bd-9nxsz
This is v1 pod kubeserve-6dcd44d8bd-pc29k
This is v2 pod kubeserve-7674cb6d4f-4dhqz
This is v1 pod kubeserve-6dcd44d8bd-pc29k
This is v2 pod kubeserve-7674cb6d4f-2pt7h
This is v2 pod kubeserve-7674cb6d4f-4dhqz
```

```
This is v1 pod kubeserve-6dcd44d8bd-8k5fd

This is v2 pod kubeserve-7674cb6d4f-4dhqz

This is v2 pod kubeserve-7674cb6d4f-2pt7h

This is v2 pod kubeserve-7674cb6d4f-2pt7h

This is v2 pod kubeserve-7674cb6d4f-2pt7h
```

- Now consider a scenario, suppose accidently we have deployed some wrong version pod & it started giving error, so in that situation we need to do rollback to previous version or any correct version.
- We have updated wrong version **v3** image in our application as below
- `Syntax:> kubectl set image deploy kubeserve app=leaddevops/kubeserve:v3`

```
root@manager-1:~# kubectl set image deploy kubeserve app=leaddevops/kubeserve:v3
deployment.apps/kubeserve image updated
root@manager-1:~# kubectl rollout status deploy kubeserve
Waiting for deployment "kubeserve" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "kubeserve" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "kubeserve" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
deployment "kubeserve" successfully rolled out
```

- This image started giving error to end user

```
root@worker-1: ~
This is v2 pod kubeserve-7674cb6d4f-4dhqz

This is v3 pod kubeserve-5cdf574594-t25z6

This is v2 pod kubeserve-7674cb6d4f-4dhqz

This is v2 pod kubeserve-7674cb6d4f-n6xw4

Some internal error has occured! This is pod kubeserve-5cdf574594-brskb

Some internal error has occured! This is pod kubeserve-5cdf574594-brskb

This is v3 pod kubeserve-5cdf574594-f8lmx
```

- In this case using below command we can roll back to previous version
- `Syntax :> kubectl rollout undo deploy <deployment name>`
- `Ex:> kubectl rollout undo deploy kubeserve`

```
root@manager-1:~# kubectl rollout undo deploy kubeserve
deployment.apps/kubeserve rolled back
root@manager-1:~# kubectl rollout status deploy kubeserve
Waiting for deployment "kubeserve" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "kubeserve" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "kubeserve" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
deployment "kubeserve" successfully rolled out
```

- Now we can see rollback of previous version happening at end user window

```
root@worker-1: ~
Some internal error has occured! This is pod kubeserve-5cdf574594-t75tj

Some internal error has occured! This is pod kubeserve-5cdf574594-brskb

Some internal error has occured! This is pod kubeserve-5cdf574594-brskb

This is v2 pod kubeserve-7674cb6d4f-rwcmf

This is v2 pod kubeserve-7674cb6d4f-rwcmf

Some internal error has occured! This is pod kubeserve-5cdf574594-f8lmx

Some internal error has occured! This is pod kubeserve-5cdf574594-kplqx
```

```
This is v2 pod kubeserve-7674cb6d4f-9qsj9
This is v2 pod kubeserve-7674cb6d4f-9qsj9
This is v2 pod kubeserve-7674cb6d4f-ws5tx
This is v2 pod kubeserve-7674cb6d4f-s755k
This is v2 pod kubeserve-7674cb6d4f-s755k
This is v2 pod kubeserve-7674cb6d4f-s755k
```

- We can roll-back to specific version also using below command
- `Syntax :> kubectl rollout undo deploy <deployment name> --to-revision=<deployment revision>`
- `Ex:> kubectl rollout undo deploy kubeserve --to-revision=1`

```
root@manager-1:~# kubectl rollout undo deploy kubeserve --to-revision=1
deployment.apps/kubeserve rolled back
root@manager-1:~# kubectl rollout status deploy kubeserve
Waiting for deployment "kubeserve" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "kubeserve" rollout to finish: 3 out of 5 new replicas have been updated...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "kubeserve" rollout to finish: 1 old replicas are pending termination...
deployment "kubeserve" successfully rolled out
```

- We can see at end user window roll-back to specific version i.e. v1 happening slowly

```
root@worker-1: ~
This is v2 pod kubeserve-7674cb6d4f-rwcmf
This is v2 pod kubeserve-7674cb6d4f-rwcmf
This is v2 pod kubeserve-7674cb6d4f-ws5tx
This is v1 pod kubeserve-6dcd44d8bd-9x7gn
This is v1 pod kubeserve-6dcd44d8bd-9x7gn
This is v1 pod kubeserve-6dcd44d8bd-9x7gn
This is v1 pod kubeserve-6dcd44d8bd-wjtsk
This is v1 pod kubeserve-6dcd44d8bd-m6nql
This is v1 pod kubeserve-6dcd44d8bd-m6nql
This is v1 pod kubeserve-6dcd44d8bd-wjtsk
```

- We can see revision history also, it will show new revision number

```
root@manager-1:~# kubectl rollout history deploy kubeserve
deployment.apps/kubeserve
REVISION  CHANGE-CAUSE
3         <none>
4         <none>
5         <none>
```

- Deployment controller internally uses **replica set** only to maintain revision of changes.
- Whenever we do changes in any deployment controller's application version, kubernetes internally creates replica set for that version.
- Whenever we update new image to deployment, a new replica set will be created, that replica set only manages pods.
- By default in kubernetes 10 number of revision history can be remember, we can increase or decrease in that.
- So we can see that we have change our deployment controller with 3 different version of images i.e. v1, v2 & v3, so internally kubernetes has create 3 different replica set.
- As per our desired sate, 5 numbers of pods of version v1 are running.

```
root@manager-1:~# kubectl get rs -o wide
NAME                  DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES                    SELECTOR
kubeserve-5cdf574594   0         0         0       65m   app          leaddevops/kubeserve:v3   app=kubeserve,pod-template-hash=5cdf574594
kubeserve-6dcd44d8bd   5         5         5       89m   app          leaddevops/kubeserve:v1   app=kubeserve,pod-template-hash=6dcd44d8bd
kubeserve-7674cb6d4f   0         0         0       85m   app          leaddevops/kubeserve:v2   app=kubeserve,pod-template-hash=7674cb6d4f
```

- If we rollout to any of specific version, suppose v2, that replica set version of pods will become in ready state gradually.

```
root@manager-1: ~
root@manager-1:~# kubectl rollout history deploy kubeserve
deployment.apps/kubeserve
REVISION  CHANGE-CAUSE
3         <none>
4         <none>
5         <none>

root@manager-1:~# kubectl rollout undo deploy kubeserve --to-revision=4
deployment.apps/kubeserve rolled back
root@manager-1:~# kubectl get rs -o wide
NAME                  DESIRED   CURRENT   READY   AGE    CONTAINERS   IMAGES                    SELECTOR
kubeserve-5cdf574594   0         0         0       80m    app          leaddevops/kubeserve:v3   app=kubeserve,pod-template-hash=5cdf574594
kubeserve-6dcd44d8bd   4         4         4       104m   app          leaddevops/kubeserve:v1   app=kubeserve,pod-template-hash=6dcd44d8bd
kubeserve-7674cb6d4f   3         3         3       101m   app          leaddevops/kubeserve:v2   app=kubeserve,pod-template-hash=7674cb6d4f
root@manager-1:~# kubectl get rs -o wide
NAME                  DESIRED   CURRENT   READY   AGE    CONTAINERS   IMAGES                    SELECTOR
kubeserve-5cdf574594   0         0         0       81m    app          leaddevops/kubeserve:v3   app=kubeserve,pod-template-hash=5cdf574594
kubeserve-6dcd44d8bd   1         1         1       106m   app          leaddevops/kubeserve:v1   app=kubeserve,pod-template-hash=6dcd44d8bd
kubeserve-7674cb6d4f   5         5         5       102m   app          leaddevops/kubeserve:v2   app=kubeserve,pod-template-hash=7674cb6d4f
root@manager-1:~# kubectl get rs -o wide
NAME                  DESIRED   CURRENT   READY   AGE    CONTAINERS   IMAGES                    SELECTOR
kubeserve-5cdf574594   0         0         0       82m    app          leaddevops/kubeserve:v3   app=kubeserve,pod-template-hash=5cdf574594
kubeserve-6dcd44d8bd   0         0         0       106m   app          leaddevops/kubeserve:v1   app=kubeserve,pod-template-hash=6dcd44d8bd
kubeserve-7674cb6d4f   5         5         5       102m   app          leaddevops/kubeserve:v2   app=kubeserve,pod-template-hash=7674cb6d4f
```

25) What is DaemonSet in kubernetes? What are its usages?

- When there is requirement to mandatorily run a pod in all nods of kubernetes cluster, than at that time we use Daemon controller.
- DaemonSet ensures that a copy of the Pod is running on all (or some) nodes.
- When a node joins the cluster, a Pod will be added to them.
- When a node is removed from the cluster, these Pods will also be recycled.
- Deleting a DaemonSet will delete all Pods it created.
- Some typical uses of a DaemonSet are:
    - running a cluster storage daemon on every node
    - running a logs collection daemon on every node
    - running a node monitoring daemon on every node
- Create DaemonSet using YML (daemonSet.yml)

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: mydaemonset
  labels:
    app: test
    env: prod
spec:
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      name: mydaemonsetpod
      labels:
        app: test
    spec:
      containers:
       - name: mydaemonsetpod
         image: lerndevops/tomcat:8.5.47
         ports :
```

```
      - containerPort: 8080
    resources:
      limits:
        memory : 200Mi
      requests:
        cpu: 100m
        memory : 200Mi
```

root@manager-1: ~

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: mydaemonset
  labels:
    app: test
    env: prod
spec:
  selector:
    matchLabels:
      app: test
  template:
    metadata:
      name: mydaemonsetpod
      labels:
        app: test
    spec:
      containers:
       - name: mydaemonsetpod
         image: lerndevops/tomcat:8.5.47
         ports :
          - containerPort: 8080
         resources:
           limits:
             memory : 200Mi
           requests:
             cpu: 100m
             memory : 200Mi
```

- Now execute below command
- Syntax:> kubectl create -f daemonSet.yml

root@manager-1: ~

```
root@manager-1:~# vim daemonSet.yml
root@manager-1:~# kubectl create -f daemonSet.yml
daemonset.apps/mydaemonset created
root@manager-1:~# kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
mydaemonset-8flbv   1/1     Running   0          39s
mydaemonset-m94zm   1/1     Running   0          39s
root@manager-1:~# kubectl get nodes
NAME        STATUS   ROLES                  AGE     VERSION
manager-1   Ready    control-plane,master   6d10h   v1.23.10
worker-1    Ready    <none>                 6d9h    v1.23.10
worker-2    Ready    <none>                 101m    v1.23.10
root@manager-1:~# kubectl get ds
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
mydaemonset   2         2         2       2            2           <none>          98s
root@manager-1:~# kubectl get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE    IP               NODE       NOMINATED NODE   READINESS GATES
mydaemonset-8flbv   1/1     Running   0          2m6s   192.168.226.120   worker-1   <none>           <none>
mydaemonset-m94zm   1/1     Running   0          2m6s   192.168.133.195   worker-2   <none>           <none>
root@manager-1:~#
```

- Here in our cluster as there are 2 worker nodes running, so there are only 2 DaemonSet pods are running.
- We can delete all DaemonSet or specific DaemonSet using below command
- Syntax:> kubectl delete ds --all
- Syntax:> kubectl delete ds --<name of DaemonSet>

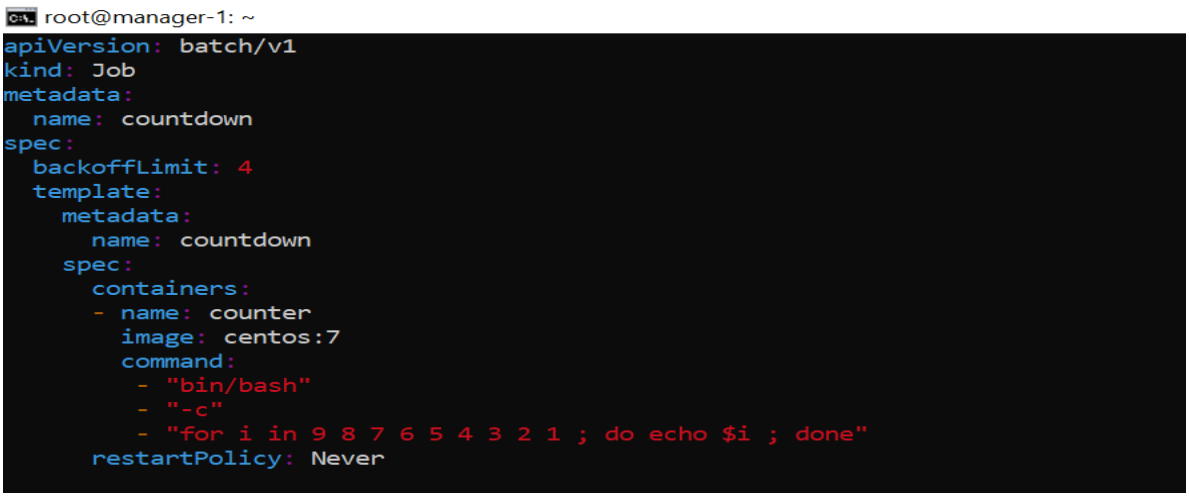
root@manager-1: ~

```
root@manager-1:~# kubectl create -f daemonSet.yml
daemonset.apps/mydaemonset created
root@manager-1:~# kubectl get ds
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
mydaemonset   2         2         2       2            2           <none>          8s
root@manager-1:~# kubectl delete ds --all
daemonset.apps "mydaemonset" deleted
root@manager-1:~# kubectl delete ds mydaemonset
Error from server (NotFound): daemonsets.apps "mydaemonset" not found
root@manager-1:~#
```

26) What is Job Controller in kubernetes? What are its usages?

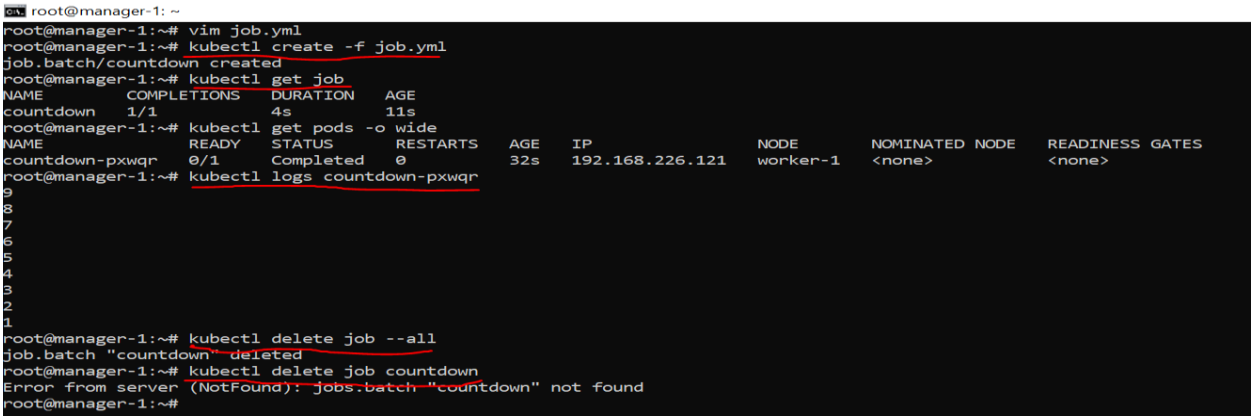
- When we run a pod it continuously in running state, but when there is requirement to complete a task/ job & die after completion, in that situation we use Job Controller.
- It helps us to perform one single activity.
- Job controller ensures that one or more pods execute their commands and exit successfully.
- In Job controller to stop containers to bring it up again & again, we use key word **`restartPolicy: Never`** in specification.
- If Job controller pod is not getting start, kubernetes retries number of times, that we can specify using keyword **`backoffLimit: 4`**
- We can create job controller using below YML(job.yml)

```
apiVersion: batch/v1
kind: Job
metadata:
  name: countdown
spec:
  backoffLimit: 4
  template:
    metadata:
      name: countdown
    spec:
      containers:
      - name: counter
        image: centos:7
        command:
         - "bin/bash"
         - "-c"
         - "for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done"
      restartPolicy: Never
```



- By running below command we can execute job controller
- `Syntax> kubectl create -f job.yml`
- We can check Job using below command
- `Syntax> kubectl get jobs`
- We can check job execution status by checking log of pod
- `Syntax>kubectl get pods -o wide`
- `Syntax> kubectl logs <pod name>`
- `Ex> kubectl logs countdown-pxwqr`
- We can use below command to delete all job
- `Syntax>kubectl delete job --all`
- We can use below command to specific job

- Syntax>kubectl delete job <job name>



27) What is Cron Job Controller in kubernetes? What are its usages?

- Suppose there is requirement to perform some task on regular interval, for example take backup of etcd, scale up some replica at some point of time, report generation etc. in that case we use CronJob controller.
- CronJob creates a schedule based on repeated intervalsJob.
- We have created a CronJob which run every 1 minute & print date-time with some message using YML(cron.yml)

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          name: mypod
          labels:
            app: myapp
        spec:
          restartPolicy: OnFailure
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
```

- Using below command we can create CronJob controller
- Syntax> kubectl create -f cron.yml
- We can get all CronJob controller using below command
- Syntax> kubectl get cj
- We can check log of pod to confirm CronJob controller has executed or not using below
- Syntax:> kubectl get pods
- Syntax:> kubectl logs <pod ID>
- We can delete all or specific CronJob controller using below command
- Syntax> kubectl delete cj --all
- Syntax> kubectl delete cj <CornJob Controller name>

root@manager-1: ~

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        metadata:
          name: mypod
          labels:
            app: myapp
        spec:
          restartPolicy: OnFailure
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
```

root@manager-1: ~

```
root@manager-1:~# vim cron.yml
root@manager-1:~# kubectl create -f cron.yml
cronjob.batch/hello created
root@manager-1:~# kubectl get cj
NAME    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
hello   */1 * * * *   False     0        <none>          30s
root@manager-1:~# kubectl get pods -o wide
NAME                     READY   STATUS      RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
hello-28440245-cfjbm     0/1     Completed   0          23s   192.168.133.199   worker-2   <none>           <none>
root@manager-1:~# kubectl logs hello-28440245-cfjbm
Sun Jan 28 04:05:01 UTC 2024
Hello from the Kubernetes cluster
root@manager-1:~# kubectl get pods -o wide
NAME                     READY   STATUS      RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
hello-28440245-cfjbm     0/1     Completed   0          59s   192.168.133.199   worker-2   <none>           <none>
root@manager-1:~# kubectl get pods -o wide
NAME                     READY   STATUS      RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
hello-28440245-cfjbm     0/1     Completed   0          84s   192.168.133.199   worker-2   <none>           <none>
hello-28440246-nnhzk     0/1     Completed   0          24s   192.168.226.65    worker-1   <none>           <none>
root@manager-1:~# kubectl logs hello-28440246-nnhzk
Sun Jan 28 04:06:02 UTC 2024
Hello from the Kubernetes cluster
root@manager-1:~# vim cron.yml
root@manager-1:~# kubectl delete cj --all
cronjob.batch "hello" deleted
root@manager-1:~# kubectl delete cj hello
Error from server (NotFound): cronjobs.batch "hello" not found
root@manager-1:~#
```

28) What is HPA (Horizontal Pod Auto Scaler) Controller in kubernetes? What are its usages?

- When application received more traffic, pod get in hang state, because of high CPU & memory usage, so at that time HPA (HorizontalPodAutoscaler) automatically updates a workload resource, by increasing number of pods.
- HPA can monitor CPU & memory utilization intermittently & check if CPU/RAM usage crosses particular % (mark) new pods get created & when CPU/RAM usage goes normal, it automatically delete extra pods.
- In HPA we need to configure auto scaling matrix i.e. min & max number of pods to run depending on resource usage condition, & the target value of raising condition.
- We can set HPA running interval by the `--horizontal-pod-autoscaler-sync-period` parameter to the `kube-controller-manager`
- Job of HPA to monitor Deployment or ReplicaSet, so that auto creation or deletion of pods will happen depending on load condition
- HPA can work only with number of pods, not with number of nodes.
- If we want node base auto scaling we need to go with GKE,AKS etc.
- Horizontal scaling means that the response to increased load is to deploy more Pods.

- So to know memory or CPU utilization matrix of pod we need to install external software, which is by default not available in kubernetes.
- We can use below command to check CPU utilization of pod, but without installing matrix software we cannot get that details.
- `Syntax:> kubectl top pods -n <name space>`
- `Ex:> kubectl top pods -n kube-system`

```
root@manager-1: ~
root@manager-1:~# kubectl get pods -n kube-system
NAME                                       READY   STATUS    RESTARTS   AGE
calico-kube-controllers-66966888c4-fkfks   1/1     Running   0          7d2h
calico-node-jfnlr                          1/1     Running   0          7d2h
calico-node-n7598                          1/1     Running   0          17h
calico-node-q5pwm                          1/1     Running   0          7d2h
coredns-64897985d-lsv6p                    1/1     Running   0          7d2h
coredns-64897985d-zz55n                    1/1     Running   0          7d2h
etcd-manager-1                             1/1     Running   0          7d2h
kube-apiserver-manager-1                   1/1     Running   0          7d2h
kube-controller-manager-manager-1          1/1     Running   0          7d2h
kube-proxy-82x7l                           1/1     Running   0          7d2h
kube-proxy-jjcqw                           1/1     Running   0          7d2h
kube-proxy-sm5qc                           1/1     Running   0          17h
kube-scheduler-manager-1                   1/1     Running   0          7d2h
root@manager-1:~# kubectl top pods -n kube-system
error: Metrics API not available
root@manager-1:~#
```

- So 1st step to set up HPA is install matrix server using below YML file
- `Syntax:> kubectl create -f` https://raw.githubusercontent.com/lerndevops/kubernetes/master/3-controllers/hpa/metric-server_1.23.yml
- Now we can get CPU & memory utilization of pods using below command
- `Syntax:> kubectl top pods -n kube-system`
- We can get CPU & memory utilization of nodes using below command
- `Syntax:> kubectl top nodes`

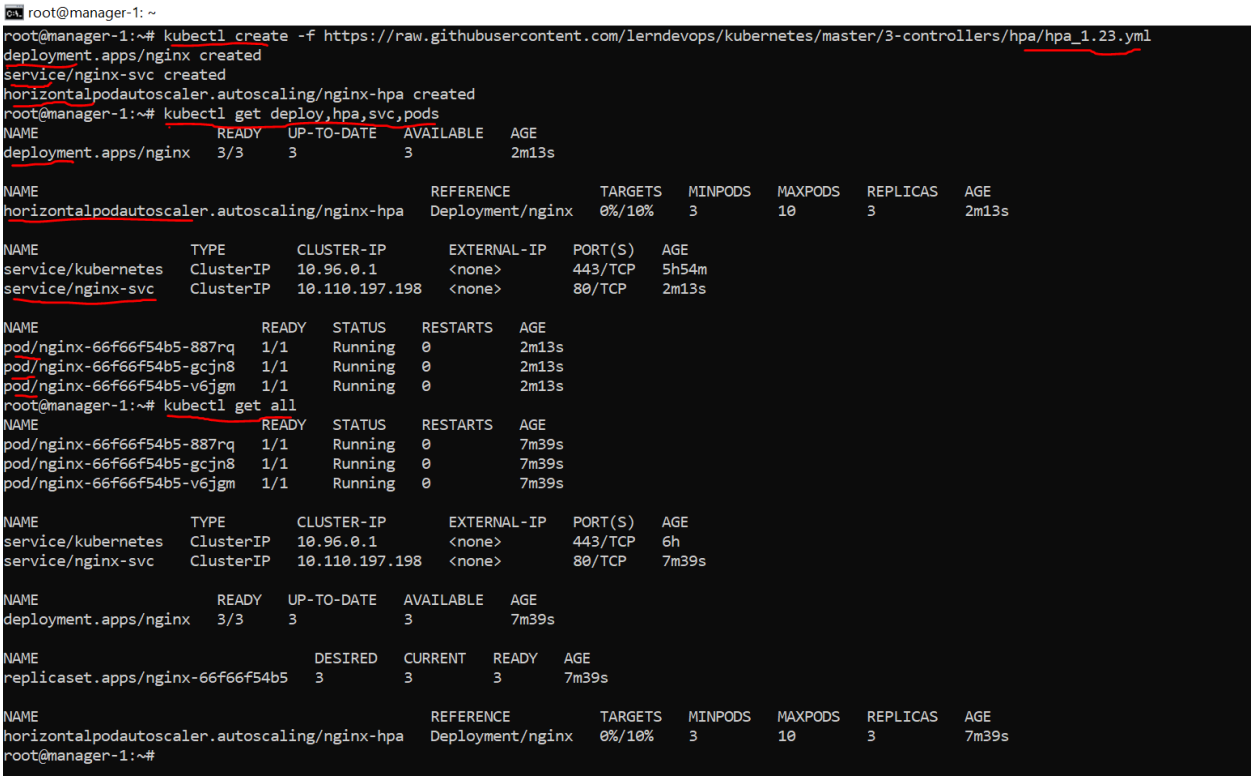
```
root@manager-1: ~
root@manager-1:~# kubectl create -f https://raw.githubusercontent.com/lerndevops/kubernetes/master/3-controllers/hpa/metric-server_1.23.yml
serviceaccount/metrics-server created
clusterrole.rbac.authorization.k8s.io/system:aggregated-metrics-reader created
clusterrole.rbac.authorization.k8s.io/system:metrics-server created
rolebinding.rbac.authorization.k8s.io/metrics-server-auth-reader created
clusterrolebinding.rbac.authorization.k8s.io/metrics-server:system:auth-delegator created
clusterrolebinding.rbac.authorization.k8s.io/system:metrics-server created
service/metrics-server created
deployment.apps/metrics-server created
apiservice.apiregistration.k8s.io/v1beta1.metrics.k8s.io created
root@manager-1:~# kubectl top pods -n kube-system
NAME                                       CPU(cores)   MEMORY(bytes)
calico-kube-controllers-66966888c4-fkfks   4m           23Mi
calico-node-jfnlr                          46m          89Mi
calico-node-n7598                          33m          78Mi
calico-node-q5pwm                          41m          98Mi
coredns-64897985d-lsv6p                    3m           13Mi
coredns-64897985d-zz55n                    2m           12Mi
etcd-manager-1                             19m          53Mi
kube-apiserver-manager-1                   60m          261Mi
kube-controller-manager-manager-1          17m          56Mi
kube-proxy-82x7l                           1m           13Mi
kube-proxy-jjcqw                           5m           15Mi
kube-proxy-sm5qc                           5m           12Mi
kube-scheduler-manager-1                   4m           21Mi
metrics-server-7cb5455c67-8wjdv            65m          13Mi
root@manager-1:~# kubectl top nodes -n kube-system
NAME        CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
manager-1   289m         14%    1363Mi          35%
worker-1    148m         7%     1531Mi          40%
worker-2    146m         7%     1923Mi          50%
root@manager-1:~# kubectl top nodes
NAME        CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
manager-1   224m         11%    1360Mi          35%
worker-1    139m         6%     1530Mi          40%
worker-2    133m         6%     1925Mi          50%
root@manager-1:~#
```

- Now we can create HPA using below YML, in which we have created
  - simple nginx container having name **nginx** with minimum 3 number of replica & given 20m (mili) CPU
  - Service with name **nginx-svc** for above deployment
  - HPA with name **nginx-hpa** for above deployment, with minReplicas: 3 & maxReplicas: 10, & matrix we used as CPU utilization i.e. when average of CPU utilization of 3 pod reached to 10% of allocated CPU 20m = 2m it will create new pods.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginxpod
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:latest
          resources:
            limits:
              cpu: 20m

---

apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec:
  type: ClusterIP  ## this is default if we do not type in service
definition
  selector:
    app: nginx
  ports:
   - protocol: TCP
     port: 80
     targetPort: 80

---

apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  minReplicas: 3
  maxReplicas: 10
  metrics:
```

```
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 10
```

- Now run below command to create above application
- `Syntax:> kubectl create -f https://raw.githubusercontent.com/lerndevops/kubernetes/master/3-controllers/hpa/hpa_1.23.yml`
- Here we have created HPA, Service, deployment, pods, we can check all using below command
- `Syntax:> kubectl get deploy,hpa,svc,pods`
- We can also use below command, but it will not give detail about secrets & config map
- `Syntax:> kubectl get all`



- Now by increasing load we can see 10 pods get auto created, so to do that we use load runner software.
- We can use Apache Bench (ab), it is a load testing and benchmarking tool for Hypertext Transfer Protocol (HTTP) server. It can be run from command line and it is very simple to use. A quick load testing output can be obtained in just one minute.
- Install this using below command
- `Syntax:> apt-get install apache2-utils`
- Install we need to get service IP to start load test first
- `Syntax:> apt-get install apache2-utils`
- Now increate load using below command
- `Syntax:> ab -n 500000 -c 1000 http://10.110.197.198`
- Here **-n** denotes Number of requests to perform & **-c** denotes Number of multiple requests to make at a time
- Here we can check HPA target % of CPU using below command

- `Syntax:> kubectl get hpa`

```
t@manager-1: ~
anager-1:~# kubectl get svc
        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
etes    ClusterIP   10.96.0.1        <none>        443/TCP   6h40m
svc     ClusterIP   10.110.197.198   <none>        80/TCP    47m
anager-1:~# kubectl get hpa
      REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa   Deployment/nginx    0%/10%    3         10        3          47m
anager-1:~# ab -n 500000 -c 1000 http://10.110.197.198/
s ApacheBench, Version 2.3 <$Revision: 1843412 $>
ght 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
ed to The Apache Software Foundation, http://www.apache.org/

arking 10.110.197.198 (be patient)
cket_recv: Connection refused (111)
of 13722 requests completed
anager-1:~# kubectl get hpa
      REFERENCE           TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa   Deployment/nginx    70%/10%   3         10        3          48m
anager-1:~#
```

- We can check pods CPU usage goes high & automatically 10 replica of pods get created

```
root@manager-1: /
Every 2.0s: kubectl top pods                        manager-1: Sun Jan 28 11:28:51 2024

NAME                     CPU(cores)   MEMORY(bytes)
nginx-66f66f54b5-42sbp   8m           2Mi
nginx-66f66f54b5-4txc4   0m           3Mi
nginx-66f66f54b5-77t54   7m           2Mi
nginx-66f66f54b5-887rq   13m          3Mi
nginx-66f66f54b5-gcjn8   16m          3Mi
nginx-66f66f54b5-vlc8m   8m           2Mi
```

```
root@manager-1: ~
very 2.0s: kubectl get pods                         manager-1: Sun Jan 28 11:29:18 2024

AMEy 2.0s: kubectl get pREADY   STATUS    RESTARTS   AGE
ginx-66f66f54b5-42sbp   1/1     Running   0          60s
ginx-66f66f54b5-44rcg   1/1     Running   0          45s
ginx-66f66f54b5-4txc4   1/1     Running   0          13m
ginx-66f66f54b5-77t54   1/1     Running   0          60s
ginx-66f66f54b5-887rq   1/1     Running   0          55m
ginx-66f66f54b5-89xjq   1/1     Running   0          45s
ginx-66f66f54b5-gcjn8   1/1     Running   0          55m
ginx-66f66f54b5-l2kbj   1/1     Running   0          45s
ginx-66f66f54b5-lqdtd   1/1     Running   0          45s
ginx-66f66f54b5-vlc8m   1/1     Running   0          60s
```

- Once load get reduce number of pods get down to min number 3

```
root@manager-1: /
Every 2.0s: kubectl top pods                                  man

NAME                     CPU(cores)   MEMORY(bytes)
nginx-66f66f54b5-4txc4   0m           3Mi
nginx-66f66f54b5-887rq   0m           3Mi
nginx-66f66f54b5-gcjn8   0m           3Mi
```

29) What is dashboard in kubernetes? What are its usages?

- Dashboard is GUI software in kubernates, which is use to manage kubernetes through GUI.
- It is also image prepared by kubernates.
- We can have many customize dashboard in kubernates.
- We can install secure & insecure dashboards.
- We can install insecure dashboard using YML as below.
- `Syntax :> kubectl create -f https://raw.githubusercontent.com/lerndevops/kubernetes/master/ui/dashboard/dashboard-insecure-v2.7.0.yaml`
- We can get all namespace, using which we can get name of our dashboard namespace to check details of that namespase
- `Syntax:> kubectl get ns`
- Now using below commands we can get all component available in dashboard namespace
- `Syntax:>kubectl get all -n kubernetes-dashboard`

```
root@manager-1: ~
root@manager-1:~# kubectl create -f https://raw.githubusercontent.com/lerndevops/kubernetes/master/ui/dashboard/dashboard-insecure-v2.7.0.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```

```
root@manager-1: ~
root@manager-1:~# kubectl get ns
NAME                   STATUS   AGE
default                Active   7d7h
kube-node-lease        Active   7d7h
kube-public            Active   7d7h
kube-system            Active   7d7h
kubernetes-dashboard   Active   77s
sock-shop              Active   10h
root@manager-1:~# kubectl get all -n kubernetes-dashboard
NAME                                             READY   STATUS    RESTARTS   AGE
pod/dashboard-metrics-scraper-6f669b9c9b-cjvrj   1/1     Running   0          112s
pod/kubernetes-dashboard-76958ccd64-lmjbh        1/1     Running   0          112s

NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/dashboard-metrics-scraper   ClusterIP   10.101.173.255   <none>        8000/TCP       113s
service/kubernetes-dashboard        NodePort    10.103.228.218   <none>        80:30009/TCP   113s

NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/dashboard-metrics-scraper   1/1     1            1           113s
deployment.apps/kubernetes-dashboard        1/1     1            1           113s

NAME                                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/dashboard-metrics-scraper-6f669b9c9b   1         1         1       113s
replicaset.apps/kubernetes-dashboard-76958ccd64        1         1         1       113s
root@manager-1:~#
```

- We can access dashboard application using node port service of dashboards
  - http://34.72.240.122:30009/
- We can access all resource of kubenares i.e. jobs, cron jobs, service, pods, replica set, replication controller, namespace etc. in dashboard application.
- We can perform all task in dashboard, whatever we do through command line on all components of kubernates.
- But in organization we don't put edit option in dashboard, it is use to view only.

30) What is probs in kubernetes?

- If pod/container is showing running status, it doesn't mean it is serving request or it is healthy or working fine.
- Sometime pod/container is showing in running status, but internally it will be in hang status.
- To handle such situation in kubernates Porbs will help us.
- Prob is nothing but health check condition of pod/container.
- There are 3 probs available in kubernetes; **Liveness, Readiness and Startup Probes**

31) What is Liveness probs in kubernetes?

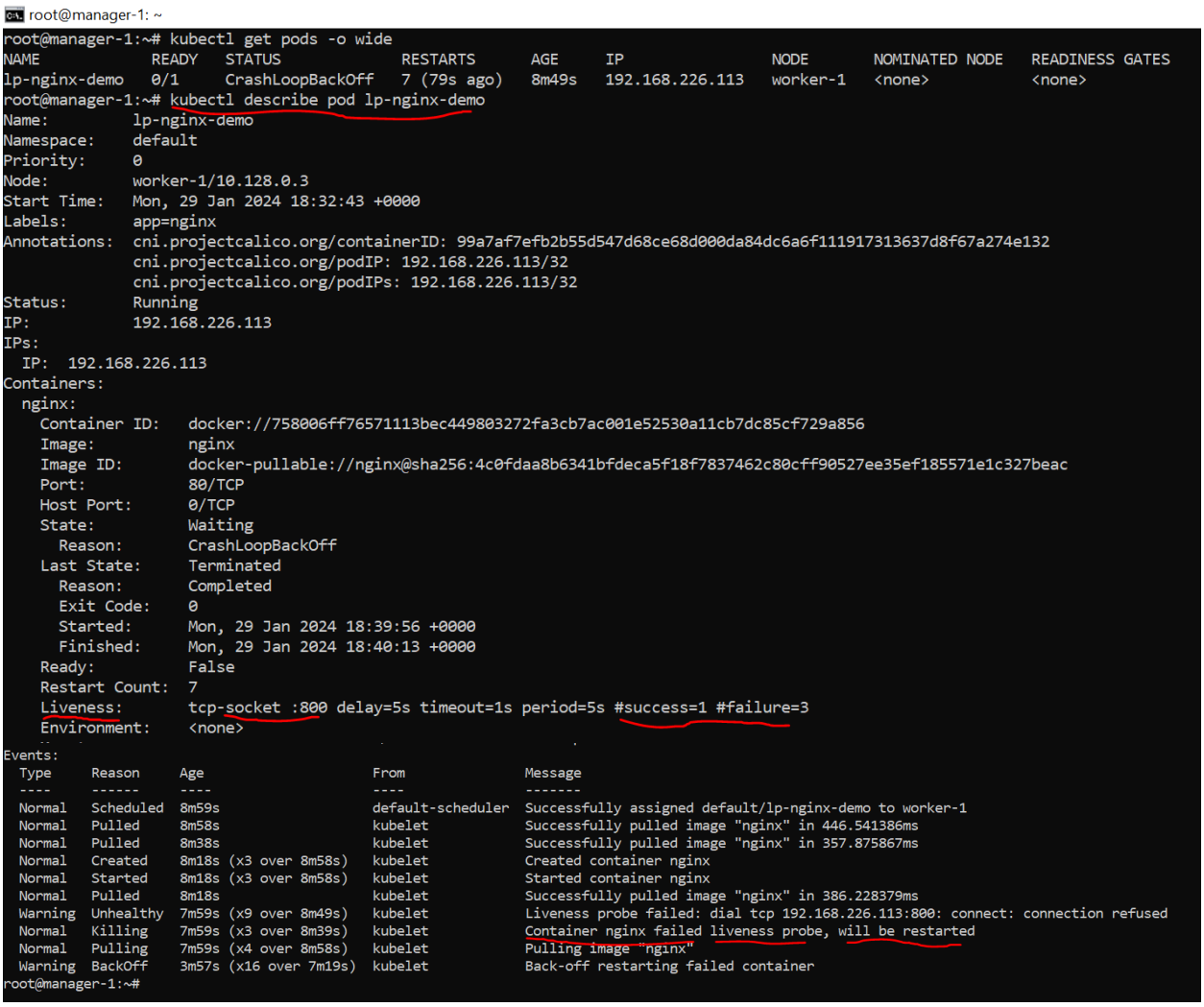
- The Liveness Probe serves as a diagnostic check to confirm if the container is alive.
- When the livenessProbe fails, Kubernetes considers the pod unhealthy, and then, attempts to restart it as a recovery measure.
- Let's take example of pod with nginx image in YML file(lp.yml), in which we put wrong condition to check connectivity on 800 port instead on 80 port

root@manager-1: ~

```
appVersion: v1
kind: Pod
metadata:
  name: lp-nginx-demo
  labels:
    app: nginx
spec:
  terminationGracePeriodSeconds: 0
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    livenessProbe:
      tcpSocket:
        port: 800
      initialDelaySeconds: 5
      periodSeconds: 5
```

- Create pod with below command
- `Syntax: > kubectl create -f lp.yml`
- We can check pods status using below command, here we can see pod is getting auto restarts because of failing liveness condition
- `Syntax: > kubectl get pode -o wide`
- `Syntax: > kubectl describe pod lp-nginx-demo`

```
root@manager-1:~# kubectl create -f lp.yml
pod/lp-nginx-demo created
root@manager-1:~# kubectl get pode -o wide
error: the server doesn't have a resource type "pode"
root@manager-1:~# kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS      AGE   IP               NODE       NOMINATED NODE   READINESS GATES
lp-nginx-demo  1/1     Running   1 (15s ago)   35s   192.168.226.113  worker-1   <none>           <none>
root@manager-1:~# kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS      AGE   IP               NODE       NOMINATED NODE   READINESS GATES
lp-nginx-demo  1/1     Running   2 (11s ago)   51s   192.168.226.113  worker-1   <none>           <none>
```



32) What is Readiness probs in kubernetes?

- Sending request to POD
- Suppose we create DB pod & its status is showing ready, but it is not confirm that it is ready to accept request because it might be its internal DB configuration are yet to start, so in that case we configure readiness prob.
- It will not restart container, it just confirm pod is ready to accept request.
- The kubelet uses readiness probes to know when a container is ready to start accepting traffic.
- A Pod is considered ready when all of its containers are ready.
- One use of this signal is to control which Pods are used as backends for Services. When a Pod is not ready, it is removed from Service load balancers.
- Let's create YML (rp.yml) in which we have set up pod with nginx image & to check readiness prob we have places get request to 800 port instead of 80 port.

root@manager-1: ~

```
apiVersion : v1
kind: Pod
metadata:
  name: rp-nginx-demo
  labels:
    app : nginx
spec:
  terminationGracePeriodSeconds: 0
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    readinessProbe:
      httpGet:
        path: /
        port: 800
      initialDelaySeconds: 5
      periodSeconds: 5
~
```

- Now we can create pod using below command:
- Syntax: > kubectl create -f rp.yml
- Here we can see that it is not ready to accept request, using below command
- Syntax: > kubectl get pods -o wide
- If we have expose this pod through service & check its end point, than we cannot get its end point
- Syntax:> kubectl expose pod rp-nginx-demo --name svc1 --port 80 --target-port 80
- Syntax:> kubectl describe svc svc1

root@manager-1: ~

```
root@manager-1:~# vim rp.yml
root@manager-1:~# kubectl create -f rp.yml
pod/rp-nginx-demo created
root@manager-1:~# kubectl get pods -o wide
NAME            READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
rp-nginx-demo   0/1     Running   0          13s   192.168.226.112   worker-1   <none>           <none>
root@manager-1:~# kubectl expose pod rp-nginx-demo --name svc1 --port 80 --target-port 80
service/svc1 exposed
root@manager-1:~# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   2d
svc1         ClusterIP   10.111.69.204   <none>        80/TCP    17s
root@manager-1:~# kubectl describe svc svc1
Name:              svc1
Namespace:         default
Labels:            app=nginx
Annotations:       <none>
Selector:          app=nginx
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.111.69.204
IPs:               10.111.69.204
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:
Session Affinity:  None
Events:            <none>
root@manager-1:~#
```

33) What is startup probs in kubernetes?

- The kubelet uses startup probes to know when a container application has started.
- If such a probe is configured, liveness and readiness probes do not start until it succeeds.
- It making sure above probes doesn't interfere with the application startup.

34) How many ways we can schedule to run pod in kubernetes?

- We can schedule to run pod in below 3 different ways
  - NodeName: Schedule to run pod on specific node
  - NodeSelector: Schedule to run pod on specific node base on matching labels
  - NodeAffinity: Schedule to run pod base on if else condition.

35) What is NodeName? How can we schedule to run pod on specific node in kubernetes?

- By using **node name** we can deploy our pod in specific node.
- Suppose we have node with name worker-2 in which we want to deploy our pod, so we can write YML as below (spn.yml)

root@manager-1: ~

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubeserv
spec:
  replicas: 2
  selector:
    matchLabels:
      app: kubeserve
  template:
    metadata:
      name: kubeserve
      labels:
        app: kubeserve
    spec:
      nodeName: worker-2
      containers:
      - image: leaddevops/kubeserve:v1
        name: app
```

- Now we create deployment of this using below command
- `Syntax:> kubectl create -f spc.yml`
- Here we can see our pod is running on worker-2 only using below command
- `Syntax:> kubectl get pod -o wide`

Select root@manager-1: ~

```
root@manager-1:~# vim spc.yml
root@manager-1:~# kubectl create -f spc.yml
deployment.apps/kubeserv created
root@manager-1:~# kubectl get node -o wide
NAME        STATUS   ROLES                  AGE     VERSION    INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION    CONTAINER-RUNTIME
manager-1   Ready    control-plane,master   12d     v1.23.10   10.128.0.2    <none>        Ubuntu 20.04.6 LTS   5.15.0-1048-gcp   docker://25.0.0
worker-1    Ready    <none>                 12d     v1.23.10   10.128.0.3    <none>        Ubuntu 20.04.6 LTS   5.15.0-1048-gcp   docker://25.0.0
worker-2    Ready    <none>                 5d22h   v1.23.10   10.128.0.5    <none>        Ubuntu 20.04.6 LTS   5.15.0-1049-gcp   docker://25.0.1
root@manager-1:~# kubectl get pod -o wide
NAME                      READY   STATUS    RESTARTS   AGE     IP               NODE       NOMINATED NODE   READINESS GATES
kubeserv-68c8d6f74c-69vxv   1/1     Running   0          2m53s   192.168.133.230   worker-2   <none>           <none>
kubeserv-68c8d6f74c-jhpfh   1/1     Running   0          2m53s   192.168.133.229   worker-2   <none>           <none>
rp-nginx-demo             0/1     Running   0          43h     192.168.226.112   worker-1   <none>           <none>
```

- Here API server directly tells kubelete where to run pod, hence there is no use to scheduler in node name.
- IF node gets deleted, than that pod will be in pending state.

36) What is NodeSelector? How can we schedule to run pod on specific node based on some condition in kubernetes?

- In node name if node gets deleted, than that pod will be in pending state, because we wanted to go in specific pod.
- So to solve above issue **node selector** came into picture, suppose we have multiple nodes in kubernetes system with their own property such as environment like DEV, UAT & PROD, so we can assign label to each node as env:DEV, env:UAT, env:PROD & depending on label condition we can deploy our application using node selector.
- For example lets create label color=green on one node i.e. worker-1, first we can check labels of all node using below command

- `Syntax:> kubectl get nodes --show-labels`
- We can set label using below command:
- `Syntax:>kubectl label node <node name> <label>`
- `Ex:> kubectl label node worker-1 color=green`

```
root@manager-1:~# kubectl label node worker-1 color=green
node/worker-1 labeled
root@manager-1:~# kubectl get nodes --show-labels
NAME       STATUS   ROLES                  AGE    VERSION    LABELS
manager-1  Ready    control-plane,master   13d    v1.23.10   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=mana
ger-1,kubernetes.io/os=linux,node-role.kubernetes.io/control-plane=,node-role.kubernetes.io/master=,node.kubernetes.io/exclude-from-external-load-balancers=
worker-1   Ready    <none>                 13d    v1.23.10   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,color=green,kubernetes.io/arch=amd64,kubernetes.io/h
ostname=worker-1,kubernetes.io/os=linux
worker-2   Ready    <none>                 7d3h   v1.23.10   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=work
er-2,kubernetes.io/os=linux
root@manager-1:~#
```

- We can create deployment with selector condition to run our pod on specific node using below YML file

root@manager-1: ~

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubeserv
spec:
  replicas: 2
  selector:
    matchLabels:
      app: kubeserv
  template:
    metadata:
      name: kubeserv
      labels:
        app: kubeserv
    spec:
      nodeSelector:
        color: "green"
      containers:
      - image: leaddevops/kubeserve:v1
        name: app
```

- Using below command we can run & check pods is running on which node
- `Syntax:> kubectl create -f nodeSelector.yml`
- `Syntax:> kubectl get pod -o wide`

root@manager-1: ~

```
root@manager-1:~# vim nodeSelector.yml
root@manager-1:~# kubectl create -f nodeSelector.yml
deployment.apps/kubeserv created
root@manager-1:~# kubectl get pod -o wide
NAME                       READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
kubeserv-68dfd659df-42mjc  1/1     Running   0          61s   192.168.226.127 worker-1   <none>           <none>
kubeserv-68dfd659df-sq2zn  1/1     Running   0          61s   192.168.226.66  worker-1   <none>           <none>
root@manager-1:~#
```

- When we want to deploy our pod on multiple nodes we can use node selector.

37) What is NodeAffinity?

- Schedule to run pod base on if else condition is as NodeAffinity.
- We can apply condition as `requiredDuringSchedulingIgnoredDuringExecution` This means that the pod will get scheduled only on a node that satisfy condition
- Also we can apply condition as `preferredDuringSchedulingIgnoredDuringExecution` This means that the pod will get scheduled on a node that satisfy condition if not satisfied we can schedule on another node.

38) What is Taint & Toleration in Kubernetes?

39) What is Volumes in Kubernetes?

- Disk inside container are lasting for a very short time, due to which there are some of the problem for running application within container
- One problem when container crash or stopped all files inside that were created or modified during the lifetime of the container are lost because during a crash, kubelet restarts the container with a clean state
- Another problem occurs when multiple containers are running in a Pod and need to share files, it can be challenging to setup and access a shared filesystem across all of the containers.
- To resolve above problem Volumes concept is introduced.
- Kubernetes supports many types of volumes, **Ephemeral volume** types have a lifetime of a pod, but **persistent volumes** exist beyond the lifetime of a pod.
- At its core, a volume is a directory, possibly with some data in it, which is accessible to the containers in a pod.
- Below are types of volumes used in kubernetes
    - Empty Directory
    - Host Path
    - Persistent Volume(PV)
    - Persistent Volume Claim(PVC)
    - Config map
    - Secret

40) What is Empty Directory Volumes in Kubernetes?

- It is a type of volume that is created when a Pod is first assigned to a Node.
- It remains active as long as the Pod is running on that node.
- The volume is initially empty and the containers in the pod can read and write the files in the emptyDir volume.
- Each container in this volume can share files using mount path
- Once the Pod is removed from the node, the data in the emptyDir is erased.
- Let's create one pod having 3 different containers images
    - Centos having mount path name "/demo1"
    - Ubuntu having mount path name "/demo2"
    - Alpine having mount path name "/demo3"
- All 3 different containers share common mount, so that they can share files with each other
- Create pod using YML & below command
- `Syntax:> kubectl create -f https://raw.githubusercontent.com/lerndevops/kubernetes/master/5-storage/volumes/emptyDir-vol-ex2.yml`
- We can enter into each container & do read/write files of each other by entering into container using below command
- `Syntax:> kubectl exec -it <pod name> -c <container name> -- bash`
- First login in centos container & create file in share mount using below command
- `Syntax:> exec -it myvolumes-pod  myvolumes-container-1 -- bash`
- `Syntax:> echo centos >/demo1/t1.txt`
- `Syntax:> ls /demo1/`
- Now login in Ubuntu container & create file in share mount using below command
- `Syntax:> exec -it myvolumes-pod  myvolumes-container-2 -- bash`
- `Syntax:> echo ubuntu >/demo2/t2.txt`

- `Syntax:> ls /demo2/`
- Now login in Alpine container & create file in share mount using below command
- `Syntax:> exec -it myvolumes-pod  myvolumes-container-3 -- bash`
- `Syntax:> echo alpine >/demo3/t3.txt`
- `Syntax:> ls /demo3/`

root@manager-1: ~

```
apiVersion: v1
kind: Pod
metadata:
  name: myvolumes-pod
  namespace: default
spec:
  containers:
  - command:
    - sh
    - -c
    - echo The Bench Container 1 is Running ; sleep 3600
    image: centos
    imagePullPolicy: IfNotPresent
    name: myvolumes-container-1
    volumeMounts:
    - mountPath: /demo1
      name: demo-volume
  - command:
    - /bin/bash
    - -c
    - echo The Bench Container 2 is Running ; sleep 3600
    image: ubuntu
    imagePullPolicy: IfNotPresent
    name: myvolumes-container-2
    volumeMounts:
    - mountPath: /demo2
      name: demo-volume
  - command:
    - sh
    - -c
    - echo The Bench Container 3 is Running ; sleep 3600
    image: alpine
    imagePullPolicy: IfNotPresent
    name: myvolumes-container-3
    volumeMounts:
    - mountPath: /demo3
      name: demo-volume
  volumes:
  - emptyDir: {}
    name: demo-volume
```

root@manager-1: ~

```
root@manager-1:~# kubectl get pod -o wide
NAME            READY   STATUS    RESTARTS          AGE     IP               NODE       NOMINATED NODE   READINESS GATES
myvolumes-pod   3/3     Running   213 (6m40s ago)   2d23h   192.168.226.126   worker-1   <none>           <none>
root@manager-1:~# kubectl exec -it myvolumes-pod -- bash
Defaulted container "myvolumes-container-1" out of: myvolumes-container-1, myvolumes-container-2, myvolumes-container-3
[root@myvolumes-pod /]# ld /demo1/
/demo1/: file not recognized: Is a directory
[root@myvolumes-pod /]# ls /demo1/
[root@myvolumes-pod /]# echo cento >t1.txt
[root@myvolumes-pod /]# ls /demo1/
[root@myvolumes-pod /]# echo cento > /demo1/t1.txt
[root@myvolumes-pod /]# ls /demo1/
t1.txt
[root@myvolumes-pod /]# exit
root@manager-1:~# kubectl exec -it myvolumes-pod -c myvolumes-container-2 -- bash
root@myvolumes-pod:/# ls /demo2/
t1.txt
root@myvolumes-pod:/# echo cento > /demo1/t2.txt
bash: /demo1/t2.txt: No such file or directory
root@myvolumes-pod:/# echo cento > /demo2/t2.txt
root@myvolumes-pod:/# ls /demo2/
t1.txt  t2.txt
```

41) What is Host Path Volumes in Kubernetes?

- A hostPath volume mounts a file or directory from the host node's file system into your Pod.
- To use hostPath volume we need to configure 2 properties :
    - Path : directory location on host machine
    - Type: this is optional, if we have not specified any value, kubernetes will automatically create folder on given path of host machine where pod will run.
    - If we have specified `type: Directory`, it is must to have directory on given path
    - If we have specified `type: DirectoryOrCreate`, if nothing exists at the given path, an empty directory will be created on given path of host machine where pod will run.
    - Other Type are `File, Socket, CharDevice, BlockDevice`
- Some uses for a hostPath are:

- Running a container that needs access to node-level system components (such as a container that transfers system logs to a central location, accessing those logs using a read-only mount of /var/log).
- Making a configuration file stored on the host system available read-only to a static pod; unlike normal Pods, static Pods cannot access ConfigMaps.
- let's create nginx container whose log folder path we mount in host machine path as below YML

root@manager-1: ~

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath-vol-pod1
spec:
  containers:
    - name: test-container
      image: nginx
      volumeMounts:
        - name: hostpath-volume
          mountPath:  /var/log/nginx
  volumes:
    - name: hostpath-volume
      hostPath:
        path: /tmp/logs
```

- Now create POD using below command

Select root@manager-1: ~

```
root@manager-1:~# vim hp.yml
root@manager-1:~# kubectl create -f hp.yml
pod/hostpath-vol-pod1 created
root@manager-1:~# kubectl get pods -o wide
NAME                READY   STATUS    RESTARTS      AGE    IP               NODE       NOMINATED NODE   READINESS GATES
hostpath-vol-pod1   1/1     Running   0             15s    192.168.133.240  worker-2   <none>           <none>
myvolumes-pod       3/3     Running   6 (46m ago)   166m   192.168.226.65   worker-1   <none>           <none>
root@manager-1:~# kubectl exec -it hostpath-vol-pod1 -- bash
root@hostpath-vol-pod1:/# ls /var/log/nginx/
access.log  error.log
root@hostpath-vol-pod1:/#
```

root@worker-2: ~

```
root@worker-2:~# ls /tmp/logs/
access.log  error.log
root@worker-2:~#
```

- We can see pod is running on worker-2 node & check all logs available in container location(/var/log/nginx) by entering into pod container are mounted in host machine(worker-2) path (/tmp/logs)

42) What is Persistent Volume (PV), Persistent Volume (PVC) & Storage Class in Kubernetes?

PersistentVolume (PV)

- A **PersistentVolume (PV)** is a piece of storage in the cluster that has been created by an administrator or dynamically provisioned using Storage Classes.
- It is a resource/object in the cluster just like a node is a cluster resource.
- Persistent volumes are independent of the lifecycle of the pod that uses it, meaning that even if the pod shuts down, the data in the volume is not erased.
- In order for a Pod to start using these volumes, it must request a volume by issuing a **persistent volume claim (PVC)**.
- **PVCs** describe the storage capacity and characteristics a pod requires, and the cluster attempts to match the request and provision the desired persistent volume.
- The PV having below specification attributes
- **Capacity**: this is used to set PV capacity
- **VolumeMode**: Kubernetes supports two volume modes of persistent volumes. A valid value for volume mode can be either Filesystem or Block. Filesystem is the default mode if the volume mode is not defined.

- **AccessMode**:
    - ReadOnlyMany(ROX) allows being mounted by multiple nodes in read-only mode.
    - ReadWriteOnce(RWO) allows being mounted by a single node in read-write mode.
    - ReadWriteMany(RWX) allows multiple nodes to be mounted in read-write mode.
- A volume can only be mounted using one access mode at a time, even if it supports many access modes.
- **Class:** A PV can specify a StorageClass to dynamically bind the PV and PVC, where the specific StorageClass is specified via the storageClassName property. If no PV is specified with this property, it can only bind to PVC that does not require a specific class.
- **Reclaim Policy:** When the node no longer needs persistent storage, the reclaiming strategies that can be used include:
    - **Retain** - meaning the PV, until deleted, is kept alive.
    - **Recycle** - meaning the data can be restored later after getting scrubbed.
    - **Delete** - associated storage assets (such as AWS EBS, GCE PD, Azure Disk, and OpenStack Cinder volumes) are deleted.
- **Mount Option**: Kubernetes administrators can specify mount options for mounting persistent volumes on a node. Not all PV types support mount options. Common types of mount options supported are:
    - gcePersistentDisk
    - awsElasticBlockStore
    - AzureDisk
    - NFS
    - RBD (Rados Block Device)
    - CephFS
    - Cinder (OpenStack volume storage)
    - Glusterfs

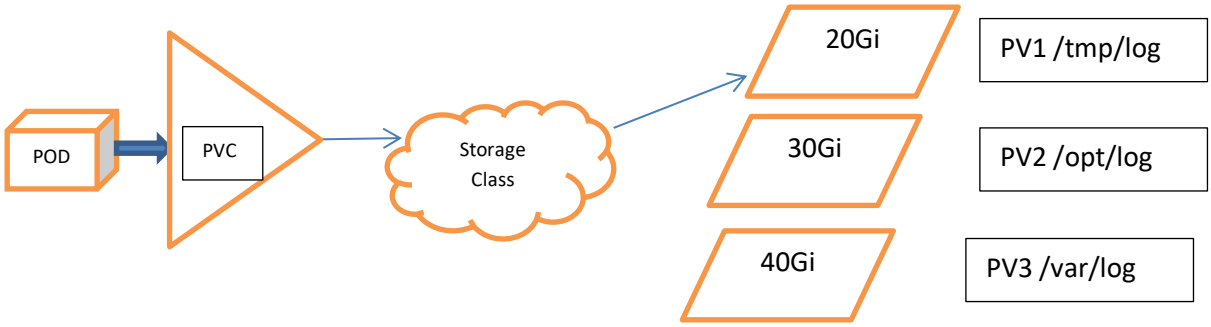
PersistentVolumeClaim (PVC)

- This is a request sent by a Kubernetes node for storage.
- Persistent volume claim is a request for storage usage by a Kubernetes developer.
- The claim can include specific storage parameters required by the application—for example an amount of storage, or a specific type of access (read/write, read-only, etc.).

Storage Class

- It is an object in kubernaters which is help to manage PV allocation to pods.
- The StorageClass object allows cluster administrators to define PVs with different properties, like performance, size or access parameters.
- It lets you expose persistent storage to users while abstracting the details of storage implementation.
- There are many predefined StorageClasses in Kubernetes (see the following section), or you can create your own.

43) What is Persistent Volume lifecycle in Kubernetes?



- The interaction between PVs and PVCs follows a distinct lifecycle, starting with provisioning and including binding, using, and reclaiming.

Provisioning

Here are the two main types of provisioning:

- Static - PVs created by a cluster administrator. These PVs are located in the Kubernetes API and contain information about storage resources available to cluster users.
- Dynamic - to meet PVC demands, clusters can attempt to dynamically provision a volume. This typically occurs when available static PVs do not match the PVC. In this case, the cluster can optionally be configured to use the default StorageClasses.

Binding

- A user creates a PVC request, specifying a certain amount of storage and the required access modes.
- A control loop, located in the master, looks for new PVCs, and then:
- **Static** - the control loop attempts to find a matching PV and then binds it to the PVC.
- **Dynamic** - if a PV was already dynamically provisioned for the PVC, the control loop binds them together. After PVC and a PV are bound, they remain exclusive.
- PVC to PV binding is a one-to-one mapping. The process uses a ClaimRef, which creates a bi-directional binding between the PV and the PVC.

Using

- Pods use claims as volumes.
- The cluster inspects the claim to find the bound volume and mounts that volume for a Pod.
- Once a user has a claim and that claim is bound, the bound PV belongs to the user for as long as they need it.

Reclaiming

- A reclaim policy for PVs tells the cluster what to do with the volume after the claim is released, and volumes can be Retained, Recycled, or Deleted.
- Once users do not need their volume anymore, they can delete the PVC objects from the API that allows reclamation of the resource.

Example

- Let's create one sample application which is having tomcat container, write log on PV using PVC
- Create PV using below YML(`my-pv.yml`)
- `Syntax:> kubectl create -f my-pv.yml`
- Check PV using below command
- `Syntax:> kubectl get pv`

root@manager-1: ~

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  storageClassName: standard
  persistentVolumeReclaimPolicy: Recycle
  capacity:
    storage: 1000Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /tmp/pvdata
```

root@manager-1: ~

```
root@manager-1:~# vim my-pv.yml
root@manager-1:~# kubectl create -f my-pv.yml
persistentvolume/my-pv created
root@manager-1:~# kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
my-pv   1000Mi     RWO            Recycle          Available           standard                3m50s
root@manager-1:~#
```

- Create PVC using below YML(my-pvc.yml), in which we have ask(request) for minimum 500Mi size storage

root@manager-1: ~

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mv-pvc
spec:
  storageClassName: standard
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
~
```

- `Syntax:> kubectl create -f my-pvc.yml`
- We can check pvc using below command
- `Syntax:> kubectl get pvc`
- We can check both pv & pvc using below command
- `Syntax:> kubectl get pv,pvc`

root@manager-1: ~

```
root@manager-1:~# vi my-pvc.yml
root@manager-1:~# kubectl create -f my-pvc.yml
persistentvolumeclaim/my-pvc created
root@manager-1:~# kubectl get pvc
NAME     STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
my-pvc   Bound    my-pv    1000Mi     RWO            standard       11s
root@manager-1:~# kubectl get pv,pvc
NAME                        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM            STORAGECLASS   REASON   AGE
persistentvolume/my-pv      1000Mi     RWO            Recycle          Bound    default/my-pvc   standard                95s

NAME                           STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/my-pvc   Bound    my-pv    1000Mi     RWO            standard       21s
root@manager-1:~#
```

- Now let's run pod using below yml (pod-pv.yml) which is having tomcat container that uses this PVC (my-pvc), which is already bound to PV (my-pv).
- So on whatever node this pod gets run, PV will bind logs of tomcat on host path (/tmp/pvdata).

root@manager-1: ~

```
apiVersion: v1
kind: Pod
metadata:
  name: pvc-pod
spec:
  containers:
    - name: my-pvc-container
      image: tomcat:8.5
      volumeMounts:
        - name: pv
          mountPath: "/usr/local/tomcat/logs"
  volumes:
    - name: pv
      persistentVolumeClaim:
        claimName: my-pvc
```

- Now run pod & check on host where this pod gets run.
- Syntax: > kubectl create -f pod-pv.yml

root@manager-1: ~

```
root@manager-1:~# vim pod-pv.yml
root@manager-1:~# kubectl create -f pod-pv.yml
pod/pvc-pod created
root@manager-1:~# kubectl get pod -o wide
NAME      READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
pvc-pod   1/1     Running   0          11s   192.168.133.244   worker-2   <none>           <none>
root@manager-1:~#
```

- We can see pod is running on worker-2, so we can check on host path (/tmp/pvdata)on that to get log by login in worker-2

root@worker-2: /

```
root@worker-2:/# ls -ltrh /tmp/pvdata/
total 8.0K
-rw-r----- 1 root root    0 Feb 18 08:30 manager.2024-02-18.log
-rw-r----- 1 root root    0 Feb 18 08:30 localhost.2024-02-18.log
-rw-r----- 1 root root    0 Feb 18 08:30 host-manager.2024-02-18.log
-rw-r----- 1 root root    0 Feb 18 08:30 localhost_access_log.2024-02-18.txt
-rw-r----- 1 root root 5.0K Feb 18 08:30 catalina.2024-02-18.log
```

- Here in above example we have created PV which uses host path storage, it has many disadvantages :
- As pod run on multiple nodes, we cannot access all pod data at single point.
- If pod is reschedule to run on other node its previous node data is gone.
- So to overcome above it is recommended to use common storage i.e. NFS(Network File Storage)
- We need to create NFS server → create PV by using that NFS mount instead of host path → create PVC → create POD

44) How can we supply dynamic data to pod in Kubernetes?

- By using **ConfigMap & Secret** we can supply dynamic data in kubernetes.
- Both are kubernetes object

45) What is ConfigMap in Kubernetes? How can we create & use it?

- Many applications rely on configuration which is used during either application initialization or runtime.
- Most of the times there is a requirement to adjust values assigned to configuration parameters.
- **ConfigMaps** is the kubernetes way to inject application pods with configuration data.
- ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.

- A ConfigMap is an API object used to store non-confidential data in key-value pairs in the cluster store (ETCD).
- Pods can consume ConfigMaps as
    - environment variables,
    - command-line arguments
    - As configuration files in a volume.
- We can use ConfigMap to setup different environment i.e. DEV/QC/UAT/PROD with same image but different configuration file.
- We can create ConfigMap using below command
- `Syntax:> kubectl create configmap <map-name> <data-source>`
- `Syntax:> kubectl create cm <map-name> <data-source>`
- We can create 2 different configMap for 2 different environment as below
- `Ex:> kubectl create cm cm1 --from-literal=username=san-dev`
- `Ex:> kubectl create cm cm2 --from-literal=username=san-uat`

root@manager-1: ~

```
root@manager-1:~# kubectl create cm cm1 --from-literal=username=san-dev
configmap/cm1 created
root@manager-1:~# kubectl create cm cm2 --from-literal=username=san-uat
configmap/cm2 created
root@manager-1:~# kubectl get cm
NAME               DATA   AGE
cm1                1      42s
cm2                1      5s
kube-root-ca.crt   1      64s
```

- We can create multiple properties configuration in file(config) & create configMap as below

root@manager-1: ~

```
username: san-dev
port: 1234
host: www.myhost.com
~
```

- `Ex:> kubectl create cm cm3 --from-file=config`
- We can describe ConfigMap as below
- `Ex:> kubectl describe cm cm3`

root@manager-1: ~

```
root@manager-1:~# vim config
root@manager-1:~# kubectl create cm cm3 --from-file=config
configmap/cm3 created
root@manager-1:~# kubectl describe cm cm3
Name:         cm3
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
config:
----
username: san-dev
port: 1234
host: www.myhost.com


BinaryData
====

Events:  <none>
```

- We can create ConfigMap using YML as below

Select root@manager-1: ~

```
apiVersion: v1
data:
  config: |
    username: san-dev
    port: 1234
    host: www.myhost.com
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: cm3
```

- Let's create nginix container in which we initialize index.html file using config map using below YML file

root@manager-1: ~

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dev-html
data:
  index.html: |
      <H1> Hello Dev ENV ConfigMap </H1>
---
kind: Pod
apiVersion: v1
metadata:
  name: cm-pod-ex1
spec:
  containers:
    - name: cm-pod-cont
      image: nginx
      ports:
        - containerPort: 80
      volumeMounts:
        - name: html-from-cm
          mountPath: /usr/share/nginx/html/
          readOnly: true
  volumes:
    - name: html-from-cm
      configMap:
        name: dev-html
```

- We can create ConfigMap & pod using below command

Select root@manager-1: ~

```
root@manager-1:~# vim cm-demo1.yml
root@manager-1:~# kubectl create -f cm-demo1.yml
configmap/dev-html created
pod/cm-pod-ex1 created
root@manager-1:~# kubectl get cm
NAME               DATA   AGE
cm1                1      58m
cm2                1      58m
cm3                1      52m
dev-html           1      47s
kube-root-ca.crt   1      59m
root@manager-1:~# kubectl decribe cm dev-html
error: unknown command "decribe" for "kubectl"

Did you mean this?
        describe
root@manager-1:~# kubectl describe cm dev-html
Name:         dev-html
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
index.html:
----
<H1> Hello Dev ENV ConfigMap </H1>


BinaryData
====

Events:  <none>
```

- We can check in POD, our index.html is used or not by entering into it as below

root@manager-1: ~

```
root@manager-1:~# kubectl get pod -o wide
NAME         READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
cm-pod-ex1   1/1     Running   0          17m   192.168.133.246   worker-2   <none>           <none>
root@manager-1:~# kubectl exec -it cm-pod-ex1 -- bash
root@cm-pod-ex1:/# cat /usr/share/nginx/html/index.html
<H1> Hello Dev ENV ConfigMap </H1>
```

- Let’s create nginix container in which we use some environment variables using config map using below YML file & command

root@manager-1: ~

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: env-cm
data:
  database: mongodb
  database_uri: mongodb://localhost:27017
---
apiVersion: v1
kind: Pod
metadata:
  name: cm-env-pod
spec:
  containers:
    - name: env-var-configmap
      image: nginix
      envFrom:
        - configMapRef:
            name: env-cm
```

root@manager-1: ~

```
root@manager-1:~# vim cm-demo2.yml
root@manager-1:~# kubectl create -f cm-demo2.yml
configmap/env-cm created
pod/cm-env-pod created
root@manager-1:~# kubectl get cm
NAME               DATA   AGE
cm1                1      94m
cm2                1      94m
cm3                1      87m
dev-html           1      36m
env-cm             2      19s
kube-root-ca.crt   1      95m
root@manager-1:~# kubectl describe cm env-cm
Name:         env-cm
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
database:
----
mongodb
database_uri:
----
mongodb://localhost:27017

BinaryData
====

Events:  <none>
```

- We can check environment variable value by entering inside containers & exceuting commands as below

```
root@manager-1:~# kubectl get pod -o wide
NAME         READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
cm-env-pod   1/1     Running   0          16s   192.168.133.248   worker-2   <none>           <none>
cm-pod-ex1   1/1     Running   0          45m   192.168.133.246   worker-2   <none>           <none>
root@manager-1:~# kubectl exec -it cm-env-pod -- bash
root@cm-env-pod:/# env
KUBERNETES_SERVICE_PORT_HTTPS=443
database_uri=mongodb://localhost:27017
KUBERNETES_SERVICE_PORT=443
HOSTNAME=cm-env-pod
PWD=/
PKG_RELEASE=1~bookworm
HOME=/root
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
database=mongodb
NJS_VERSION=0.8.3
TERM=xterm
SHLVL=1
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PORT=443
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
NGINX_VERSION=1.25.4
_=/usr/bin/env
root@cm-env-pod:/# echo jdbc:odbc:$database_uri
jdbc:odbc:mongodb://localhost:27017
root@cm-env-pod:/#
```

46) What is secret in Kubernetes? How can we create & use it?

- A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key.
- Such information might otherwise be put in a Pod specification or in a container image.
- Using a Secret means that you don't need to include confidential data in your application code.
- By default in kubernetes it used base 64 encoding to store secretes, once we used secret in pod it will automatically decode this value.
- We can create secretes by command line as below
- `Syntax:> kubectl create secret generic <secrete name> <data source key-value>`
- `Ex:> kubectl create secret generic sc1 --from-literal=pass=abc@123`
- `Ex:> kubectl get secret`
- `Ex:> kubectl describe secret sc1`

```
root@manager-1: ~
root@manager-1:~# kubectl create secret generic sc1 --from-literal=pass=abc@123
secret/sc1 created
root@manager-1:~# kubectl get secret
NAME                  TYPE                                  DATA   AGE
default-token-ssgb6   kubernetes.io/service-account-token   3      2m4s
sc1                   Opaque                                1      22s
root@manager-1:~# kubectl describe secret sc1
Name:         sc1
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
pass:  7 bytes
root@manager-1:~#
```

- We can create secretes & use that by below YML & command

```
root@manager-1: ~
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: ZXhhbXBsZS11c2Vy
  password: ZXhhbXBsZS1wYXNzd29yZA==
---
apiVersion: v1
kind: Pod
metadata:
  name: sec-pod-demo
spec:
  containers:
  - name: mycontainer
    image: redis
    volumeMounts:
    - name: myvol
      mountPath: "/apps/secrets"
      readOnly: true
  volumes:
  - name: myvol
    secret:
      secretName: mysecret
```

```
root@manager-1: ~
root@manager-1:~# vim sec-demo.yml
root@manager-1:~# kubectl create -f sec-demo.yml
secret/mysecret created
pod/sec-pod-demo created
root@manager-1:~# kubectl get secret
NAME                  TYPE                                  DATA   AGE
default-token-kzn7t   kubernetes.io/service-account-token   3      3m43s
mysecret              Opaque                                2      32s
root@manager-1:~# kubectl explain secret mysecret
error: We accept only this format: explain RESOURCE
root@manager-1:~# kubectl describe secret mysecret
Name:         mysecret
Namespace:    default
Labels:       <none>
Annotations:  <none>

Type:  Opaque

Data
====
password:  16 bytes
username:  12 bytes
root@manager-1:~#
```

- We can check secret value by login in side container as below

root@manager-1: ~

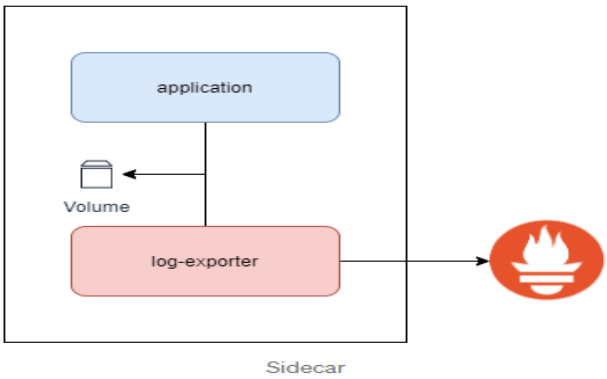
```
root@manager-1:~# kubectl exec -it sec-pod-demo -- bash
root@sec-pod-demo:/data# cd /
root@sec-pod-demo:/# ls /apps/secrets/
password  username
root@sec-pod-demo:/# cat /apps/secrets/username
example-userroot@sec-pod-demo:/#
root@sec-pod-demo:/# cat /apps/secrets/password
example-passwordroot@sec-pod-demo:/#
root@sec-pod-demo:/#
```

47) What is multi container Pod in Kubernetes? How can we create & use it?

- Pods can contain multiple containers, for some excellent reasons — primarily, the fact that containers in a pod get scheduled in the same node in a multi-node cluster. This makes communication between them faster and more secure.
- Containers within a pod can interact with each other in various ways:
- **Network**: Containers can access any listening ports on containers within the same pod, even if those ports are not exposed outside the pod.
- **Shared Storage Volumes:** Containers in the same pod can be given the same mounted storage volume, which allows them to interact with the same files.
- **Shared Process Namespace:** Process namespace sharing can be enabled by setting `shareProcessNamespace true` in the pod spec. This allows containers within the pod to interact with, and signal, one and-other's processes.
- We can run multi container pod using below pattern :
    - Sidecar
    - Ambassador
    - Adapter

48) What is Side Car Pattern in multi container Pod in Kubernetes? How can we create & use it?

- Sidecars derive their name from motorcycle sidecars. While your motorcycle can work fine without the sidecar, having one enhances or extends the functionality of your bike, by giving it an extra seat.
- Similarly, in Kubernetes, a sidecar pattern is used to enhance or extend the existing functionality of the container.
- Your container works perfectly well without the sidecar, but with it, it can perform some extra functions.
- So, a sidecar container enhances the main container in some way, adding functionality to it.
- Some examples are :
    - Using a sidecar for monitoring and logging and adding an agent for these purposes
    - A sidecar periodically syncs files in a webserver container's file system from a Git repository.
- Let's create enginx container as main container which displays HTML page with current date time, and its side car container debian OS whose role is to create HTML file with current date time,
- Debian OS will share HTML file to Nginix using enptyDir shared volume concept.



root@manager-1: ~

```
apiVersion: v1
kind: Pod
metadata:
  name: mc1
spec:
  volumes:
  - name: html
    emptyDir: {}
  containers:
  - name: 1st
    image: nginx
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  - name: 2nd
    image: debian
    volumeMounts:
    - name: html
      mountPath: /html
    command: ["/bin/sh", "-c"]
    args:
    - while true; do
      date >>/html/index.html
      sleep 1;
      done
```

- Create pod using below command:
- `Syntax: kubectl create -f sidecar.yml`
- Create svc to access nginx using below command
- `Syntax: kubectl expose pod mc1 --name svc1 --port 1234 --target-port 80 --type NodePort`

root@manager-1: ~

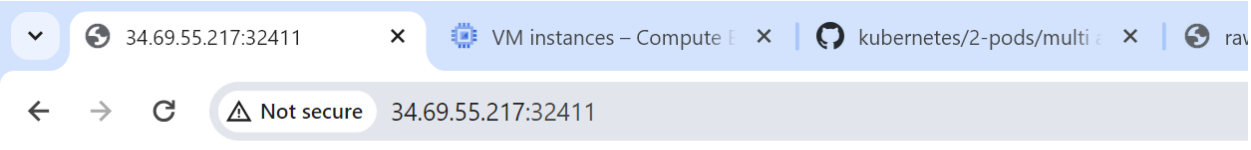
```
root@manager-1:~# vim sidecar.yml
 26L, 436C written
root@manager-1:~# kubectl create -f sidecar.yml
pod/mc1 created
root@manager-1:~# kubectl get pod -o wide
NAME   READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
mc1    2/2     Running   0          30s   192.168.226.84   worker-1   <none>           <none>
root@manager-1:~# kubectl label pod mc1 app=myapp
pod/mc1 labeled
root@manager-1:~# kubectl expose pod mc1 --name svc1 --port 1234 --target-port 80 --type NodePort
service/svc1 exposed
root@manager-1:~# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          5m26s
svc1         NodePort    10.101.116.38   <none>        1234:32411/TCP   32s
root@manager-1:~#
```

- We can login into debian container & create “index.html” file in shared path /html/index.html

root@manager-1: ~

```
root@manager-1:~# kubectl exec -it mc1 -c 2nd -- bash
root@mc1:/# echo "<h1>Hello Kubernet</h1>" >> /html/index.html
root@mc1:/#
```

- We can check this html page create by debins OS is used to display nginx page by accessing in browser by using port & host IP http://34.69.55.217:32411/

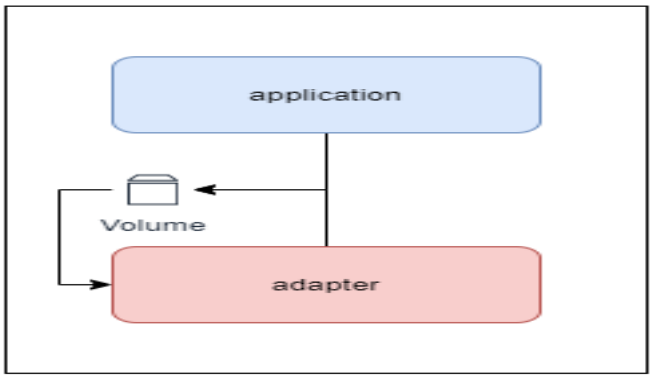


Hello Kuberneter

Hello Kubernet

49) What is Adapter Pattern in multi container Pod in Kubernetes? How can we create & use it?

- The Adapter is another pattern that you can implement with multiple containers.
- The adapter pattern helps you standardize something heterogeneous in nature.
- For example, you're running multiple applications within separate containers, but every application has a different way of outputting log files.
- Now, if you have a centralized logging system that accepts logs in a particular format only, what can you do in such a situation? Well, you can either change the source code of each application to output a standard log format or use an adapter to standardize the logs before sending it to your central server. That's where the adapter pattern comes in.
- So, an adapter container transforms the output of the main container.
- Some great examples are: An adapter container reads log output from the main container and transforms it.



Adapter

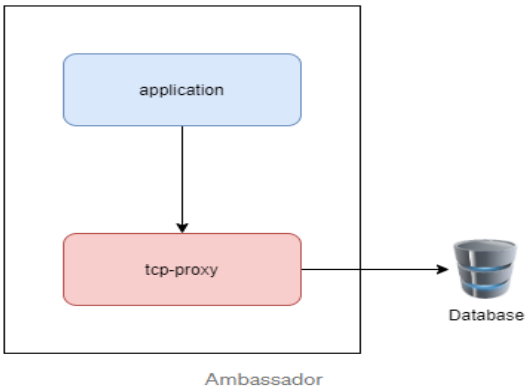
- Let's create pod using below YML having 2 apline container 1st one i.e. **app-container** will generate log, 2nd one i.e. **adapter-container** take log of 1st & format in stander format.
- After running this pod we can login in **adapter-container** & check log in location /var/log, we can found **app-container** logs available in **adapter-container**

```
root@manager-1: ~
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-adapter
spec:
  volumes:
  - name: shared-logs
    emptyDir: {}
  containers:
  - name: app-container
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "while true; do date > /var/log/top.txt && top -n 1 -b >> /var/log/top.txt; sleep 5;done"]
    volumeMounts:
    - name: shared-logs
      mountPath: /var/log
  - name: adapter-container
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "while true; do (cat /var/log/top.txt | head -1 > /var/log/status.txt) && (cat /var/log/top.txt | head -2 | tail -1 | grep -o -E '\\d+\\w' | head -1 >> /var/log/status.txt)
 && (cat /var/log/top.txt | head -3 | tail -1 | grep -o -E '\\d+%' | head -1 >> /var/log/status.txt); sleep 5; done"]
    volumeMounts:
    - name: shared-logs
      mountPath: /var/log
```

```
root@manager-1: ~
root@manager-1:~# vim adapter.yml
root@manager-1:~# kubectl create -f adapter.yml
pod/pod-with-adapter created
root@manager-1:~# kubectl get pod -o wide
NAME               READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
pod-with-adapter   2/2     Running   0          9s    192.168.226.69   worker-1   <none>           <none>
root@manager-1:~# kubectl exec -it pod-with-adapter -c adapter-container -- sh
/ # ls /var/log/
status.txt  top.txt
/ # cat /var/log/status.txt
Sun Mar  3 11:57:38 UTC 2024
1346768K
0%
/ #
```

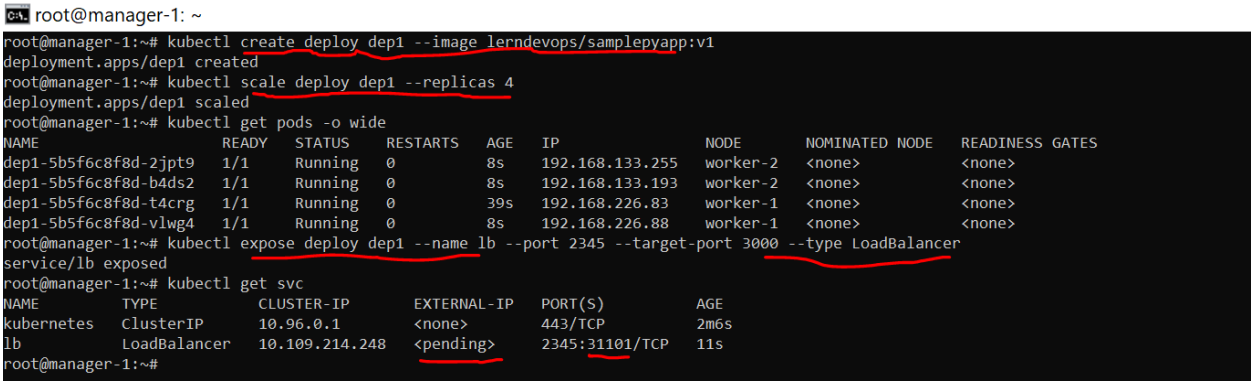
50) What is Ambassador Pattern in multi container Pod in Kubernetes? How can we create & use it?

- The ambassador pattern derives its name from an Ambassador, who is an envoy and a person a country chooses to represent their country and connect with the rest of the world.
- Similarly, in the Kubernetes perspective, an Ambassador pattern implements a proxy to the external world.
- Let's look at an example — if you build an application that needs to connect with a database server, the server configuration, etc, changes with the environment.
- Now, the official recommendation to handle these is to use Config Maps, but what if you have legacy code that is already using another way of connecting to the database. Maybe, a properties file, or even worse, a hardcoded set of values. What if you want to communicate with localhost, and you can leave the rest to the admin? You can use the Ambassador pattern for these kinds of scenarios.

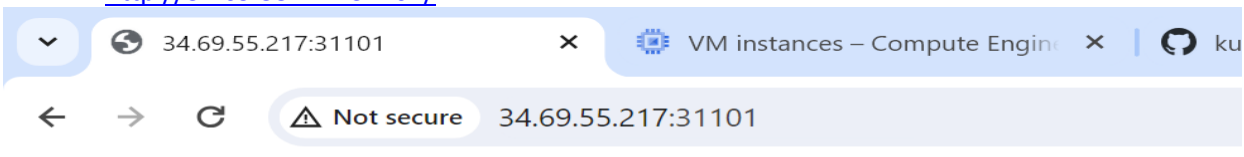


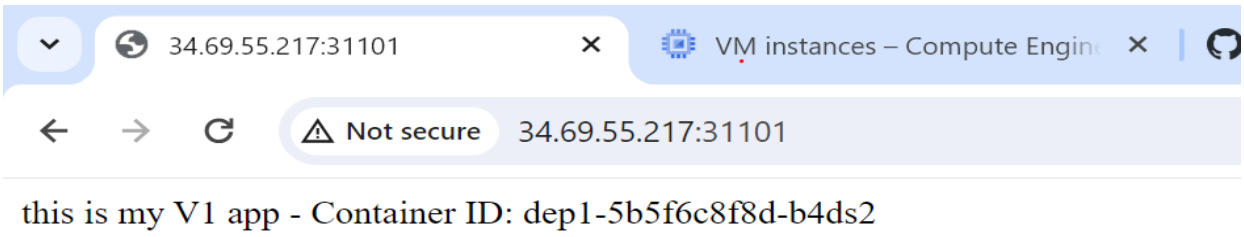
51) What is Service Load Balancing in Kubernetes?

- Load Balancer service is an extension of NodePort service. NodePort and ClusterIP Services, to which the external load balancer routes are automatically created.
- It integrates NodePort with cloud-based load balancers.
- It exposes the Service externally using a cloud provider's load balancer.
- Each cloud provider (AWS, Azure, GCP, etc) has its own native load balancer implementation. The cloud provider will create a load balancer, which then automatically routes requests to your Kubernetes Service.
- Traffic from the external load balancer is directed at the backend Pods. The cloud provider decides how it is load balanced.
- The actual creation of the load balancer happens asynchronously.
- Every time you want to expose a service to the outside world, you have to create a new LoadBalancer and get an IP address.
- Let's create on one deployment having some python application image & we scale this application up to 4 replicas
- Now expose this application using Load balancer service, if this application could have been running on Google Kubernete Engine at External IP instead of pending we could seen virtual IP of Google cloud.



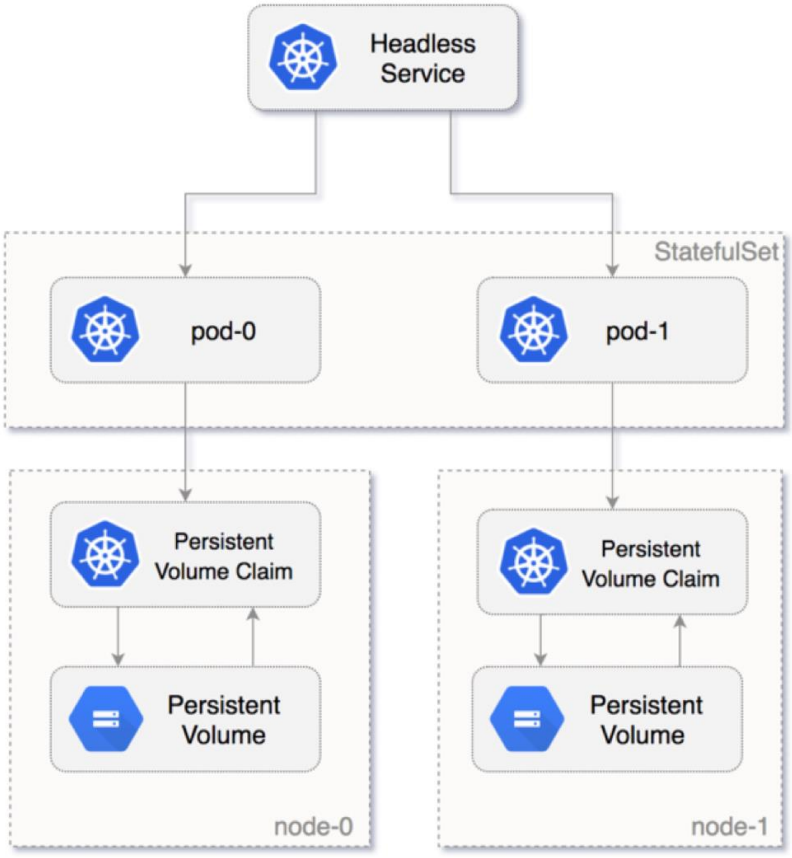
- But we can access this application using our master/worker nodes IP & highlighted port http://34.69.55.217:31101/





52) What is Statefull sets in Kubernetes?

- A stateful application is a program that saves client data from the activities of one session for use in the next session. The data that is saved is called the application's state. OR
- Stateful applications require that data that is used or generated by the application is persisted, retained, backed up and accessible outside of the particular hosts that run the application
- Statefull applications may expect to have static hostname of its replicas
- Statefull applications may expect to start from a start point not as a whole.
- All databases, such as MySQL, Oracle, and PostgreSQL, are examples of stateful applications.
- Kubernetes is a container orchestration tool that uses many controllers to run applications as containers (Pods). One of these controllers is called StatefulSet, which is used to run stateful applications. i.e. Data Base
- A StatefulSet is the Kubernetes controller used to run the stateful application as containers (Pods) in the Kubernetes cluster.
- StatefulSets assign a sticky identity—an ordinal number starting from zero—to each Pod instead of assigning random IDs for each replica Pod.
- A new Pod is created by cloning the previous Pod's data. If the previous Pod is in the pending state, then the new Pod will not be created.
- If you delete a Pod, it will delete the Pod in reverse order, not in random order. For example, if you had four replicas and you scaled down to three, it will delete the Pod numbered 3.



When to Use StatefulSets :

- There are several reasons to consider using StatefulSets. Here are two examples:
- Assume you deployed a MySQL database in the Kubernetes cluster and scaled this to three replicas, and a frontend application wants to access the MySQL cluster to read and write data. The read request will be forwarded to three Pods. However, the write request will only be forwarded to the first (primary) Pod, and the data will be synced with the other Pods. You can achieve this by using StatefulSets.
- Deleting or scaling down a StatefulSet will not delete the volumes associated with the stateful application. This gives you your data safety. If you delete the MySQL Pod or if the MySQL Pod restarts, you can have access to the data in the same volume.

53) How Pod to Pod communications work in Kubernetes?

Networking in Kubernetes:

- Since a Kubernetes cluster consists of various nodes and pods, understanding how they communicate between them is essential. The Kubernetes networking model supports different types of open source implementations.
- The Kubernetes networking model, on the other hand, natively supports multi-host networking in which pods are able to communicate with each other by default, regardless of which host they live in.
- Kubernetes does not provide a default network implementation; it only enforces a model for third-party tools to implement.

Pod to Pod communication:

- Kubernetes follows an “IP-per-pod” model where each pod get assigned an IP address and all containers in a single pod share the same network namespaces and IP address. Containers in the same pod can therefore reach each other’s ports via localhost.
- However, it is not recommended to communicate directly with a pod via its IP address due to pod’s volatility (a pod can be killed and replaced at any moment).
- Instead, use a Kubernetes service which represents a group of pods acting as a single entity to the outside. Services get allocated their own IP address in the cluster and provide a reliable entry point.
- Kubernetes services allow grouping pods under a common access policy (for example, load-balanced). The service gets assigned a virtual IP which pods outside the service can communicate with. Those requests are then transparently proxied (via the **kube-proxy** component that runs on each node) to the pods inside the service.
- Different proxy-modes are supported:
- **iptables**: kube-proxy installs iptables rules trap access to service IP addresses and redirect them to the correct pods.
- **userspace**: kube-proxy opens a port (randomly chosen) on the local node. Requests on this “proxy port” get proxied to one of the service’s pods (as retrieved from Endpoints API).
- **ipvs** (from Kubernetes 1.9): calls netlink interface to create ipvs rules and regularly synchronizes them with the Endpoints API.
- These kube-proxy must be running in all node of cluster
- We can see all kube proxy pod running in system namespace of cluster using below command
- `Syntax:> kubectl get pods -o wide -n kube-system`

root@manager-1: ~

```
root@manager-1:~# kubectl get pods -o wide -n kube-system
NAME                                       READY   STATUS    RESTARTS       AGE   IP               NODE        NOMINATED NODE   READINESS GATES
calico-kube-controllers-66966888c4-fkfks   1/1     Running   3 (8d ago)     48d   192.168.11.141   manager-1   <none>           <none>
calico-node-jfnlr                          1/1     Running   3 (8d ago)     48d   10.128.0.2       manager-1   <none>           <none>
calico-node-n7598                          1/1     Running   3 (6d9h ago)   41d   10.128.0.5       worker-2    <none>           <none>
calico-node-q5pwm                          1/1     Running   5 (6d9h ago)   48d   10.128.0.3       worker-1    <none>           <none>
coredns-64897985d-lsv6p                    1/1     Running   3 (8d ago)     48d   192.168.11.139   manager-1   <none>           <none>
coredns-64897985d-zz55n                    1/1     Running   3 (6d9h ago)   48d   192.168.11.140   manager-1   <none>           <none>
etcd-manager-1                             1/1     Running   3 (6d9h ago)   48d   10.128.0.2       manager-1   <none>           <none>
kube-apiserver-manager-1                   1/1     Running   3 (8d ago)     48d   10.128.0.2       manager-1   <none>           <none>
kube-controller-manager-manager-1          1/1     Running   3 (6d9h ago)   48d   10.128.0.2       manager-1   <none>           <none>
kube-proxy-82x7l                           1/1     Running   5 (6d9h ago)   48d   10.128.0.3       worker-1    <none>           <none>
kube-proxy-jjcqw                           1/1     Running   3 (6d9h ago)   48d   10.128.0.2       manager-1   <none>           <none>
kube-proxy-sm5qc                           1/1     Running   3 (6d9h ago)   41d   10.128.0.5       worker-2    <none>           <none>
kube-scheduler-manager-1                   1/1     Running   3 (6d9h ago)   48d   10.128.0.2       manager-1   <none>           <none>
metrics-server-7cb5455c67-8wjdv            1/1     Running   3 (8d ago)     41d   192.168.133.251  worker-2    <none>           <none>
root@manager-1:~#
```

54) What is DNS in Kubernetes? How it is used for service & pods?

- Kubernetes provides its own DNS service to resolve domain names inside the cluster in order for pods to communicate with each other.
- This is implemented by deploying a regular Kubernetes service which does name resolution inside the cluster, and configuring individual containers to contact the DNS service to resolve domain names.
- Note that this “internal DNS” is compatible and expected to run along with the cloud provider’s DNS service.
- Every service gets assigned a DNS name which resolves to the cluster IP of the service. The naming convention includes the service name and its namespace.
- For example: `my-service.my-namespace.svc.cluster.local`
- A pod inside the same namespace as the service does not need to fully qualify its name, for example a pod in my-namespace could lookup this service with a DNS query for my-service, while a pod outside my-namespace would have to query for my-service.my-namespace.
- We can see DNS software (pod) in side system namespace, which is used for all DNS communication within cluster i.e. pod name resolution & service name resolution.

root@manager-1: ~

```
root@manager-1:~# kubectl get pods -o wide -n kube-system
NAME                                       READY   STATUS    RESTARTS       AGE   IP               NODE        NOMINATED NODE   READINESS GATES
calico-kube-controllers-66966888c4-fkfks   1/1     Running   3 (8d ago)     48d   192.168.11.141   manager-1   <none>           <none>
calico-node-jfnlr                          1/1     Running   3 (8d ago)     48d   10.128.0.2       manager-1   <none>           <none>
calico-node-n7598                          1/1     Running   3 (6d9h ago)   41d   10.128.0.5       worker-2    <none>           <none>
calico-node-q5pwm                          1/1     Running   5 (6d9h ago)   48d   10.128.0.3       worker-1    <none>           <none>
coredns-64897985d-lsv6p                    1/1     Running   3 (8d ago)     48d   192.168.11.139   manager-1   <none>           <none>
coredns-64897985d-zz55n                    1/1     Running   3 (6d9h ago)   48d   192.168.11.140   manager-1   <none>           <none>
etcd-manager-1                             1/1     Running   3 (6d9h ago)   48d   10.128.0.2       manager-1   <none>           <none>
kube-apiserver-manager-1                   1/1     Running   3 (8d ago)     48d   10.128.0.2       manager-1   <none>           <none>
kube-controller-manager-manager-1          1/1     Running   3 (6d9h ago)   48d   10.128.0.2       manager-1   <none>           <none>
kube-proxy-82x7l                           1/1     Running   5 (6d9h ago)   48d   10.128.0.3       worker-1    <none>           <none>
kube-proxy-jjcqw                           1/1     Running   3 (6d9h ago)   48d   10.128.0.2       manager-1   <none>           <none>
kube-proxy-sm5qc                           1/1     Running   3 (6d9h ago)   41d   10.128.0.5       worker-2    <none>           <none>
kube-scheduler-manager-1                   1/1     Running   3 (6d9h ago)   48d   10.128.0.2       manager-1   <none>           <none>
metrics-server-7cb5455c67-8wjdv            1/1     Running   3 (8d ago)     41d   192.168.133.251  worker-2    <none>           <none>
root@manager-1:~# kubectl get rs -o wide -n kube-system
NAME                                 DESIRED   CURRENT   READY   AGE   CONTAINERS                IMAGES                                           SELECTOR
calico-kube-controllers-66966888c4   1         1         1       48d   calico-kube-controllers   docker.io/calico/kube-controllers:v3.24.1        k8s-app=calico-kube-controllers,pod-templa
te-hash=66966888c4
coredns-64897985d                    2         2         2       48d   coredns                   k8s.gcr.io/coredns/coredns:v1.8.6                k8s-app=kube-dns,pod-template-hash=6489798
5d
metrics-server-7cb5455c67            1         1         1       41d   metrics-server            k8s.gcr.io/metrics-server/metrics-server:v0.5.0  k8s-app=metrics-server,pod-template-hash=7
cb5455c67
root@manager-1:~# kubectl get deploy -o wide -n kube-system
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS                IMAGES                                           SELECTOR
calico-kube-controllers   1/1     1            1           48d   calico-kube-controllers   docker.io/calico/kube-controllers:v3.24.1        k8s-app=calico-kube-controllers
coredns               2/2     2            2           48d   coredns                   k8s.gcr.io/coredns/coredns:v1.8.6                k8s-app=kube-dns
metrics-server        1/1     1            1           41d   metrics-server            k8s.gcr.io/metrics-server/metrics-server:v0.5.0  k8s-app=metrics-server
root@manager-1:~#
```

- Let’s take example to check how service & pod can be access :
- Step1: create pod nginx & service nginx in a namespace called "testns"

```
root@manager-1: ~
root@manager-1:~# kubectl create namespace testns
namespace/testns created
root@manager-1:~# kubectl -n testns run nginx --image=nginx --expose --port=80
service/nginx created
pod/nginx created
root@manager-1:~# kubectl get all -n testns -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
pod/nginx   1/1     Running   0          24s   192.168.133.195   worker-2   <none>           <none>

NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE   SELECTOR
service/nginx   ClusterIP   10.98.252.58   <none>        80/TCP    24s   run=nginx
root@manager-1:~#
```

- Step2: create pod mynginx & service mynginx in a namespace called "default"

```
root@manager-1: ~
root@manager-1:~# kubectl -n default run mynginx --image=nginx --expose --port=80
service/mynginx created
pod/mynginx created
root@manager-1:~# kubectl get all -n default -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
pod/mynginx   1/1     Running   0          15s   192.168.226.87   worker-1   <none>           <none>

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE   SELECTOR
service/kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   34h   <none>
service/mynginx      ClusterIP   10.105.57.127   <none>        80/TCP    15s   run=mynginx
root@manager-1:~#
```

- Step3: create another pod in any namespace which has netutils so that we can run a nslookup command

```
root@manager-1: ~
root@manager-1:~# kubectl -n testns run dnstest --image=lerndevops/samples:netshoot --rm -it -- /bin/bash
If you don't see a command prompt, try pressing enter.
bash-5.1#
```

- The above command will create new pod called dnstest in testns namespace & get you inside it, from inside pod we can try accessing pods using their service dns & pod dns
- Step4: test Service DNS:
  - if we try to access **nginx** service using name, we can see it will work because **nginx** service is available in local

```
root@manager-1: ~
root@manager-1:~# kubectl -n testns run dnstest --image=lerndevops/samples:netshoot --rm -it -- /bin/bash
If you don't see a command prompt, try pressing enter.
bash-5.1# cat /etc/resolv.conf
nameserver 10.96.0.10
search testns.svc.cluster.local svc.cluster.local cluster.local us-central1-a.c.capable-argon-411714.internal c.capable-argon-411714.internal google.internal
options ndots:5
bash-5.1# nslookup nginx
Server:         10.96.0.10
Address:        10.96.0.10#53

Name:   nginx.testns.svc.cluster.local
Address: 10.98.252.58
```

  - if we try to access **mynginx** service using name which in in some other namespace i.e. default, we can see it will not work because **mynginx** service is not available in local, so to access this service we need to namespace name also i.e. **mynginx.default.svc**

```
bash-5.1# nslookup mynginx
Server:         10.96.0.10
Address:        10.96.0.10#53

** server can't find mynginx: NXDOMAIN

bash-5.1# nslookup mynginx.default.svc
Server:         10.96.0.10
Address:        10.96.0.10#53

Name:   mynginx.default.svc.cluster.local
Address: 10.105.57.127

bash-5.1#
```

- Step4: test Pod DNS:
    - to make pod DNS we need get the podip & replace "." with "-" then add .namespace.pod
    - Example: covnert 1.2.3.4 to 1-2-3-4 & add .namespace.pod
    - > 1-2-3-5.testns.pod
    - So we can access local **nginx** pod whose IP is : 192.168.133.195 we can use DNS as **192-168-133-195.testns.pod**
    - Similarly we can access default namespace's pod **mynginx** whose IP is : 192.168.226.87 using DNS as **192-168-226-87.default.pod**

root@manager-1: ~

```
bash-5.1# nslookup 192-168-133-195.testns.pod
Server:         10.96.0.10
Address:        10.96.0.10#53

Name:   192-168-133-195.testns.pod.cluster.local
Address: 192.168.133.195

bash-5.1# nslookup 192-168-226-88.default.pod
Server:         10.96.0.10
Address:        10.96.0.10#53

Name:   192-168-226-88.default.pod.cluster.local
Address: 192.168.226.88
```

55) What is network policy in Kubernetes?How it is used for pods communication?

- A network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints.
- Network policies allow you to specify which pods can talk to other pods.
- This helps when securing communication between pods, allowing you to identify **ingress (**incoming traffic**)** and **egress (**outgoing traffic**)** rules.
- Network policy is created at namespace level, not cluster level.
- We can apply a network policy to a pod by using **labels** or namespace **selectors**.
- We can even choose a CIDR block range to apply the network policy.
- By Default, all pods in the cluster can communicate with any other pod and reach out to any available IP.
- Network Policies allow you to limit what network traffic is allowed to and from pods in your cluster.
- Let's create one ngixn pod

root@manager-1: ~

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: network-policy-secure-pod
  labels:
    app: secure-app
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

root@manager-1: ~

```
root@manager-1:~# vim a.yml
root@manager-1:~# kubectl create -f a.yml
pod/network-policy-secure-pod created
root@manager-1:~# kubectl get pods -o wide
NAME                      READY   STATUS    RESTARTS   AGE     IP              NODE       NOMINATED NODE   READINESS GATES
network-policy-secure-pod   1/1     Running   0          4m11s   192.168.226.89   worker-1   <none>           <none>
```

- Now create test pod which will connect to above pod

root@manager-1: ~

```
apiVersion: v1
kind: Pod
metadata:
  name: network-policy-test-pod
spec:
  containers:
  - name: busybox
    image: radial/busyboxplus:curl
    command: ["/bin/sh", "-c", "while true; do sleep 3600; done"]
root@manager-1:~# kubectl create -f te.yml
pod/network-policy-test-pod created
root@manager-1:~# kubectl get pods -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
network-policy-secure-pod   1/1     Running   0          10m   192.168.226.89   worker-1   <none>           <none>
network-policy-test-pod     1/1     Running   0          5s    192.168.226.91   worker-1   <none>           <none>
```

- We can check by login inside **test-pod** & do curl to IP of **secure-pod (10.168.226.89),** it will work fine

```
root@manager-1:~# kubectl get pods -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
network-policy-secure-pod   1/1     Running   0          10m   192.168.226.89   worker-1   <none>           <none>
network-policy-test-pod     1/1     Running   0          5s    192.168.226.91   worker-1   <none>           <none>
root@manager-1:~# kubectl exec -it network-policy-test-pod -- sh
sh: shopt: not found
[ root@network-policy-test-pod:/ ]$ curl 192.168.226.89
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[ root@network-policy-test-pod:/ ]$
```

- Create a network policy that restricts all access to the secure pod, except to and from pods which bear the allow-access: "true" label:
- We can check all network policy using below command
- `Syntax: > kubectl get networkpolicies`
- `Syntax: > kubectl get netpol`
- We can get any network policy details using below command
- `Syntax:> kubectl describe networkpolicy <my-network-policy name>`

root@manager-1: ~

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
spec:
  podSelector:
    matchLabels:
      app: secure-app
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          allow-access: "true"
    ports:
    - protocol: TCP
      port: 80
  egress:
  - to:
    - podSelector:
        matchLabels:
          allow-access: "true"
    ports:
    - protocol: TCP
      port: 80
```

root@manager-1: ~

```
root@manager-1:~# vim np.yml
 27L, 479C written
root@manager-1:~# kubectl create -f np.yml
networkpolicy.networking.k8s.io/my-network-policy created
root@manager-1:~# kubectl get netpol
NAME                POD-SELECTOR     AGE
my-network-policy   app=secure-app   7m3s
root@manager-1:~# kubectl describe netpol my-network-policy
Name:         my-network-policy
Namespace:    default
Created on:   2024-03-10 08:20:07 +0000 UTC
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     app=secure-app
  Allowing ingress traffic:
    To Port: 80/TCP
    From:
      PodSelector: allow-access=true
  Allowing egress traffic:
    To Port: 80/TCP
    To:
      PodSelector: allow-access=true
  Policy Types: Ingress, Egress
```

- Now we can check curl again by login in test-pod, we cannot connect to secure-pod now.

root@manager-1: ~

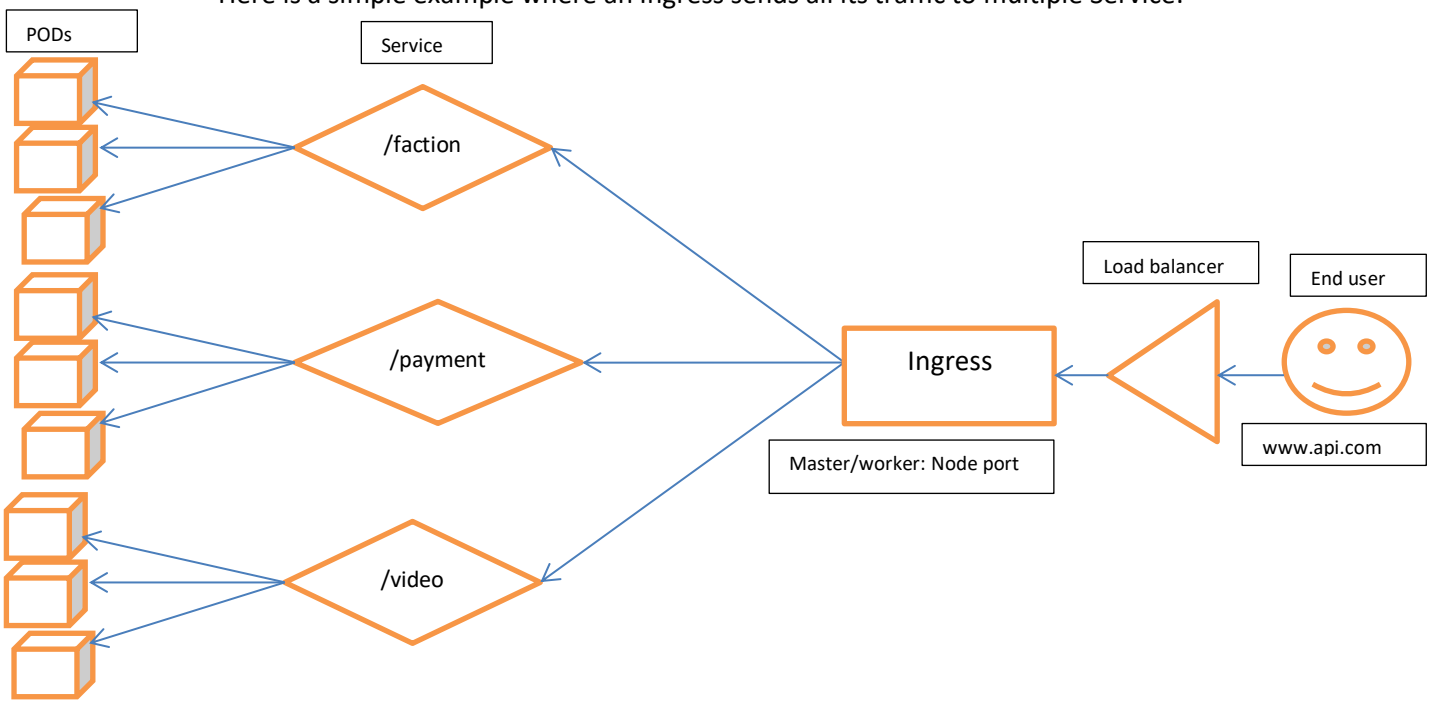
```
root@manager-1:~# kubectl get pods -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES
network-policy-secure-pod   1/1     Running   0          57m   192.168.226.89   worker-1   <none>           <none>
network-policy-test-pod     1/1     Running   0          46m   192.168.226.91   worker-1   <none>           <none>
root@manager-1:~# kubectl exec -it network-policy-test-pod -- sh
sh: shopt: not found
[ root@network-policy-test-pod:/ ]$ curl 192.168.226.89
curl: (7) Failed to connect to 192.168.226.89 port 80: Connection timed out
[ root@network-policy-test-pod:/ ]$
```

- To connect to secure-pod we need to add label to test-pod & again login & check with curl command
- `Syntax:> kubectl label pod network-policy-test-pod allow-access=true`

```
root@manager-1: ~
root@manager-1:~# kubectl get pods --show-labels
NAME                      READY   STATUS    RESTARTS   AGE   LABELS
network-policy-secure-pod   1/1     Running   0          68m   app=secure-app
network-policy-test-pod     1/1     Running   0          57m   <none>
root@manager-1:~# kubectl label pod network-policy-test-pod allow-access=true
pod/network-policy-test-pod labeled
root@manager-1:~# kubectl get pods --show-labels
NAME                      READY   STATUS    RESTARTS   AGE   LABELS
network-policy-secure-pod   1/1     Running   0          69m   app=secure-app
network-policy-test-pod     1/1     Running   0          59m   allow-access=true
root@manager-1:~# kubectl get pods --show-labels -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP              NODE       NOMINATED NODE   READINESS GATES   LABELS
network-policy-secure-pod   1/1     Running   0          69m   192.168.226.89   worker-1   <none>           <none>            app=secure-app
network-policy-test-pod     1/1     Running   0          59m   192.168.226.91   worker-1   <none>           <none>            allow-access=true
root@manager-1:~# kubectl exec -it network-policy-test-pod -- sh
sh: shopt: not found
[ root@network-policy-test-pod:/ ]$ curl 192.168.226.89
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[ root@network-policy-test-pod:/ ]$
```

56) What is ingress in Kubernetes?

- It is an API object that manages external access to the services in a cluster, typically HTTP.
- Ingress can also provide load balancing, SSL termination and name-based virtual hosting.
- Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.
- Traffic routing is controlled by rules defined on the Ingress resource
- Here is a simple example where an Ingress sends all its traffic to multiple Service:



- To use ingress we need to install software which is called **ingress controller**, there are many ingress controller available in market i.e. nginx, HA-proxy, Envoy, traefik etc.
- Best used ingress controller in industry is nginx ingress controller.
- To route traffic to specific service depends on requirement, we need to configure set of rules in ingress controller, which is called as **ingress resources/rules**
- We can install nginx ingress using below, all ingress object are install in namespace "**ingress-nginx**"
- `Syntax:> kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.3.0/deploy/static/provider/baremetal/deploy.yaml`
- We can check what all object are available in "**ingress-nginx**" namespace
- `Syntax:> kubectl get all -n ingress-nginx`
- Here we can see many pods & service are created

```
root@manager-1: ~
root@manager-1:~# kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.3.0/deploy/static/provider/baremetal/deploy.yaml
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
configmap/ingress-nginx-controller created
service/ingress-nginx-controller created
service/ingress-nginx-controller-admission created
deployment.apps/ingress-nginx-controller created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
root@manager-1:~# kubectl get all -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
pod/ingress-nginx-admission-create-g2x84    0/1     Completed   0          46s
pod/ingress-nginx-admission-patch-p8cgd     0/1     Completed   1          46s
pod/ingress-nginx-controller-77cb5dbf4d-smngt   1/1     Running     0          46s

NAME                                         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
service/ingress-nginx-controller             NodePort    10.106.133.51   <none>        80:31697/TCP,443:32755/TCP   46s
service/ingress-nginx-controller-admission   ClusterIP   10.109.191.72   <none>        443/TCP                      46s

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1           46s

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-77cb5dbf4d   1         1         1       46s

NAME                                       COMPLETIONS   DURATION   AGE
job.batch/ingress-nginx-admission-create   1/1           8s         46s
job.batch/ingress-nginx-admission-patch    1/1           8s         46s
```

- Let's take example where we have created 3 application with 3 different URL i.e. /app1, /app2 & /app3 using https://raw.githubusercontent.com/lerndevops/kubernetes/master/6-networking/ingress/deploy-app.yml

```
root@manager-1: ~
root@manager-1:~# kubectl create -f https://raw.githubusercontent.com/lerndevops/kubernetes/master/6-networking/ingress/deploy-app.yml
deployment.apps/app1 created
service/app1-svc created
deployment.apps/app2 created
service/app2-svc created
deployment.apps/app3 created
service/app3-svc created
root@manager-1:~# kubectl get deploy,svc
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/app1    2/2     2            2           32s
deployment.apps/app2    2/2     2            2           32s
deployment.apps/app3    2/2     2            2           32s

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
service/app1-svc     ClusterIP   10.105.131.209   <none>        80/TCP    32s
service/app2-svc     ClusterIP   10.111.127.156   <none>        80/TCP    32s
service/app3-svc     ClusterIP   10.110.72.121    <none>        80/TCP    32s
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP   12m
```

- Now create ingress rule using YML file
- `Syntax:> kubectl apply -f https://raw.githubusercontent.com/lerndevops/educka/master/6-networking/ingress/ingress-rule1.yml`
- We can check all ingress rule using below command
- `Syntax:> kubectl get ing`

root@manager-1: ~

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-rule1
  annotations:
    ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
        - path: /app1
          pathType: Prefix
          backend:
            service:
              name: app1-svc
              port:
                number: 80
        - path: /app2
          pathType: Prefix
          backend:
            service:
              name: app2-svc
              port:
                number: 80
        - path: /app3
          pathType: Prefix
          backend:
            service:
              name: app3-svc
              port:
                number: 80
```

root@manager-1: ~

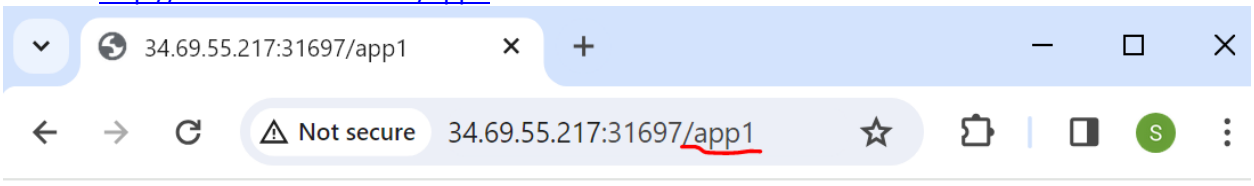
```
root@manager-1:~# vim ing.yml
root@manager-1:~# kubectl create -f ing.yml
ingress.networking.k8s.io/ingress-rule1 created
root@manager-1:~# kubectl get ing
NAME             CLASS    HOSTS    ADDRESS        PORTS    AGE
ingress-rule1    nginx    *        10.128.0.3     80       2m23s
root@manager-1:~#
```

- Now to access application, we need to get nginx ingress controller service node port using below command
- Syntax:> kubectl get services -n ingress-nginx

```
root@manager-1:~# kubectl get pods -n ingress-nginx -o wide
NAME                                        READY   STATUS      RESTARTS   AGE   IP               NODE       NOMINATED NODE   READINESS GATES
ingress-nginx-admission-create-g2x84        0/1     Completed   0          43m   192.168.226.93   worker-1   <none>           <none>
ingress-nginx-admission-patch-p8cgd         0/1     Completed   1          43m   192.168.226.90   worker-1   <none>           <none>
ingress-nginx-controller-77cb5dbf4d-smngt   1/1     Running     0          43m   192.168.226.81   worker-1   <none>           <none>
root@manager-1:~# kubectl get services -n ingress-nginx
NAME                                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
ingress-nginx-controller             NodePort    10.106.133.51   <none>        80:31697/TCP,443:32755/TCP   44m
ingress-nginx-controller-admission   ClusterIP   10.109.191.72   <none>        443/TCP                      44m
```

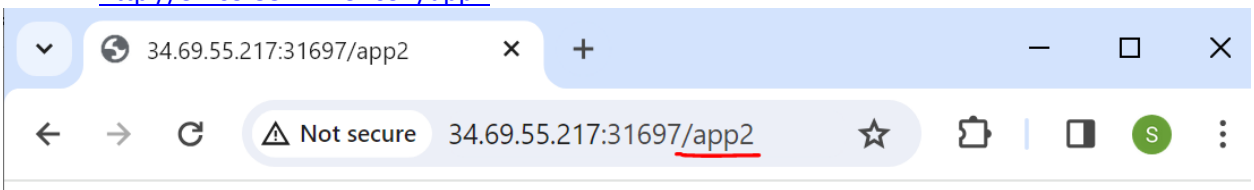
- By using IP of master/worker node using below command & ingress node port we can access our application.
- If we will not specified any path http://34.69.55.217:31697/ it will give 404, because in ingress controller we have not specified default path (/) rout.

- So if we use ingress controller configured path as below we can see out put
- http://34.69.55.217:31697/app1



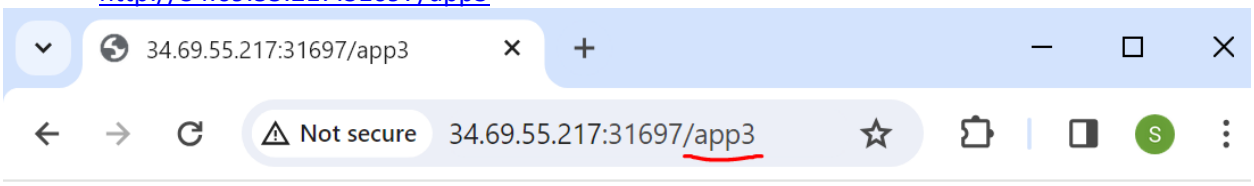
hey there, you reached app1

- http://34.69.55.217:31697/app2



hey there, You reached app2

- http://34.69.55.217:31697/app3



hey there, You reached app3

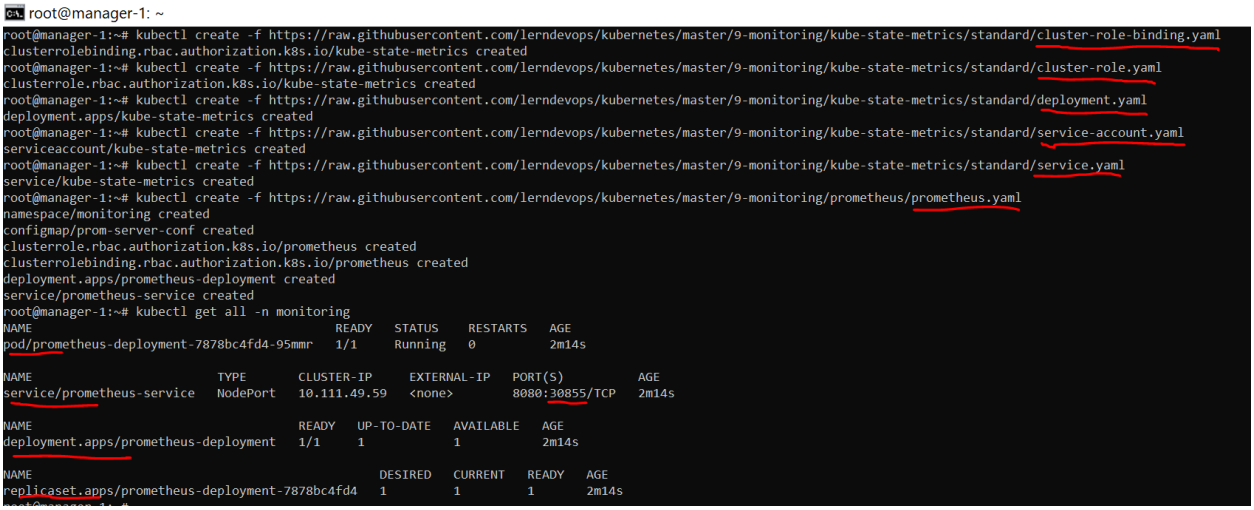
57) Why we need monitoring? How many types of monitoring available in Kubernetes?

- We need monitoring to make sure everything is running fine. If something is not working we should get alert.
- Monitoring is divide into 3 type
  - **System Level** : Monitoring CPU, RAM, speed & disk utilization
  - **Service/Process Level**: Monitoring software application which is needed to run application for ex. Docker software, DB software, and application server software.
  - **Application Level:** Monitoring application is running fine, by hitting application URL or login in application using some dummy credentials, also by using some monitoring tools.

58) How we can monitor Kubernetes?

- To monitor utilization of Pods we need to install software name as matrix server. It shows current time utilization of CPU & RAM.
- We can install matrix server using below command
- `Syntax:> kubectl create -f https://raw.githubusercontent.com/lerndevops/kubernetes/master/9-monitoring/metrics-server/metrics-server.yam`
- If we want to see utilization over period of time & top utilization we need to use “Prometheus”
- Prometheus is an open-source software application used for event monitoring and alerting. It records real-time metrics in a time series database (allowing for high dimensionality) built using a HTTP pull model, with flexible queries and real-time alerting.

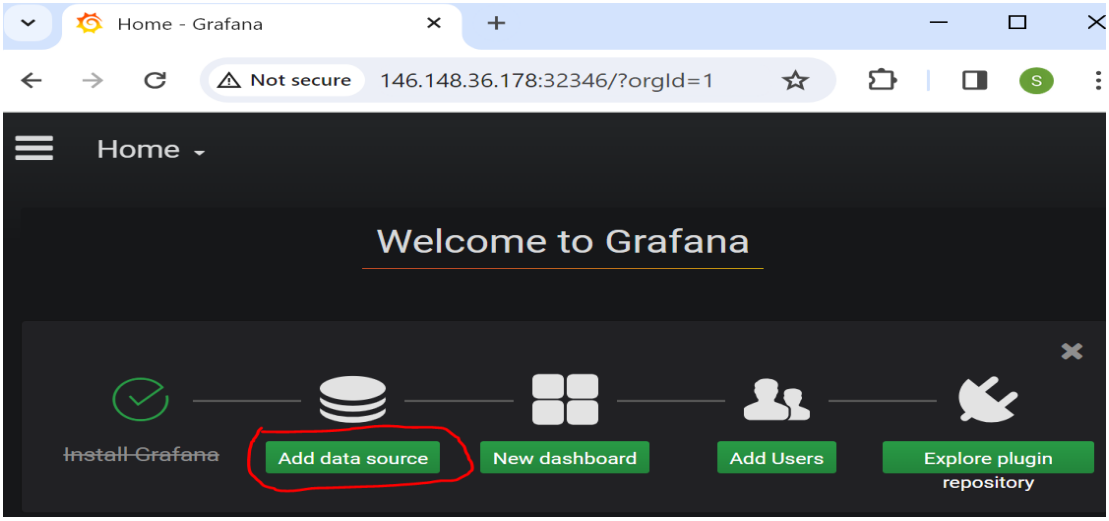
- The Prometheus server works on the principle of scraping, i.e., invoking the metrics endpoints of the various nodes that it is configured to monitor. It collects these metrics at regular intervals and stores them locally.
- It collects data & with the help of Grafana GUI software it display on UI, in the form of Graph, chart etc.
- Prometheus can configure rules to trigger alerts using PromQL, alert manager will be in charge of managing alert notification, grouping, inhibition, etc.
- The alert manager component configures the receivers, gateways to deliver alert notifications.
- Grafana can pull metrics from any number of Prometheus servers and display panels and Dashboards.
- To work with Prometheus we need to install kube-state-metrics first
    - `https://raw.githubusercontent.com/lerndevops/kubernetes/master/9-monitoring/kube-state-metrics/standard/cluster-role-binding.yaml`
    - `https://raw.githubusercontent.com/lerndevops/kubernetes/master/9-monitoring/kube-state-metrics/standard/cluster-role.yaml`
    - https://raw.githubusercontent.com/lerndevops/kubernetes/master/9-monitoring/kube-state-metrics/standard/deployment.yaml
    - https://raw.githubusercontent.com/lerndevops/kubernetes/master/9-monitoring/kube-state-metrics/standard/service-account.yaml
    - https://raw.githubusercontent.com/lerndevops/kubernetes/master/9-monitoring/kube-state-metrics/standard/service.yaml
- Now we need to install Prometheus using below YML
    - https://raw.githubusercontent.com/lerndevops/kubernetes/master/9-monitoring/prometheus/prometheus.yaml
- We can see namespace monitoring is created in which pod, service, deployment & replicaset is created.
- We can access Prometheus using service port & IP of cluster http://146.148.36.178:30855/



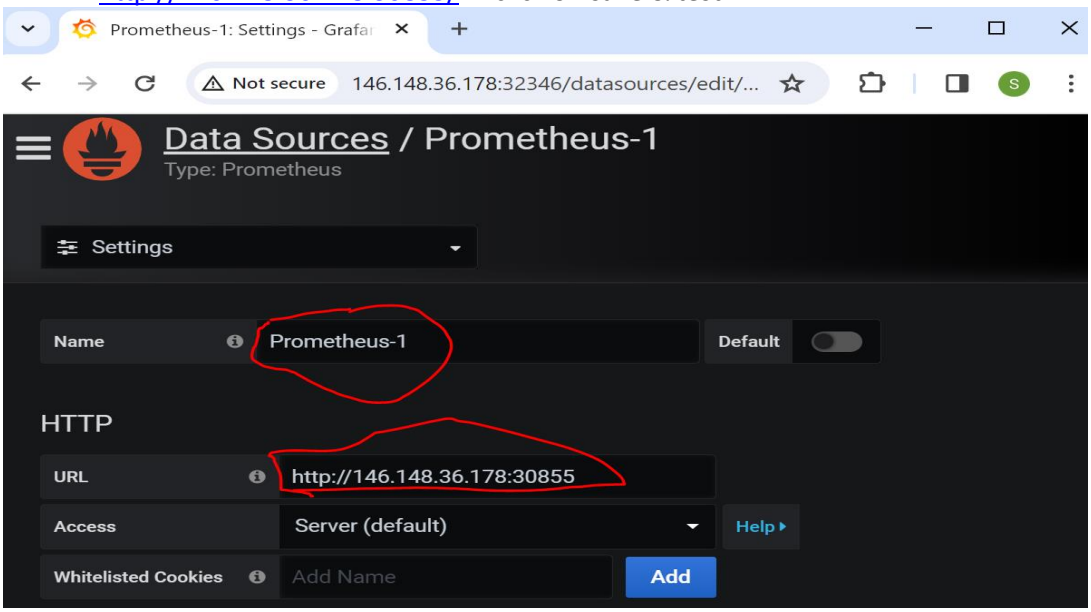
- But this GUI is not so useful, so we need to install **Grafana**
- `kubectl create -f` https://raw.githubusercontent.com/lerndevops/kubernetes/master/9-monitoring/prometheus/grafana.yml
- This will also create pod, service, deployment & replicaset.
- We can access this **Grafana** GUI using service port & cluster IP http://146.148.36.178:32346/login
- We can login using default credentials
    - `user : admin`
    - `Password: admin`

```
root@manager-1: ~
root@manager-1:~# kubectl create -f https://raw.githubusercontent.com/lerndevops/kubernetes/master/9-monitoring/prometheus/grafana.yml
deployment.apps/grafana created
service/grafana-ui-svc created
Error from server (AlreadyExists): error when creating "https://raw.githubusercontent.com/lerndevops/kubernetes/master/9-monitoring/prometheus/grafana.yml":
exists
root@manager-1:~# kubectl get all -n monitoring
NAME                                         READY   STATUS    RESTARTS   AGE
pod/grafana-65cc8d46dd-zdh6h                 1/1     Running   0          23s
pod/prometheus-deployment-7878bc4fd4-95mmr   1/1     Running   0          19m

NAME                         TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
service/grafana-ui-svc       NodePort   10.99.141.51   <none>        3000:32346/TCP   23s
service/prometheus-service   NodePort   10.111.49.59   <none>        8080:30855/TCP   19m

NAME                                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/grafana                 1/1     1            1           23s
deployment.apps/prometheus-deployment   1/1     1            1           19m

NAME                                               DESIRED   CURRENT   READY   AGE
replicaset.apps/grafana-65cc8d46dd                 1         1         1       23s
replicaset.apps/prometheus-deployment-7878bc4fd4   1         1         1       19m
root@manager-1:~#
```

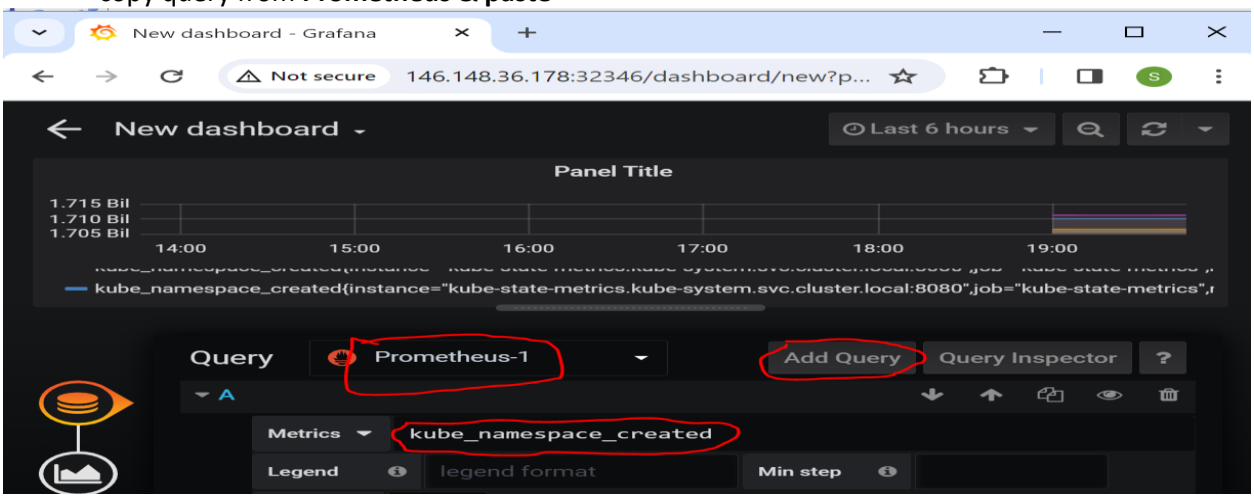
- After login we need to configure data source as **Prometheus** in **Garafana**



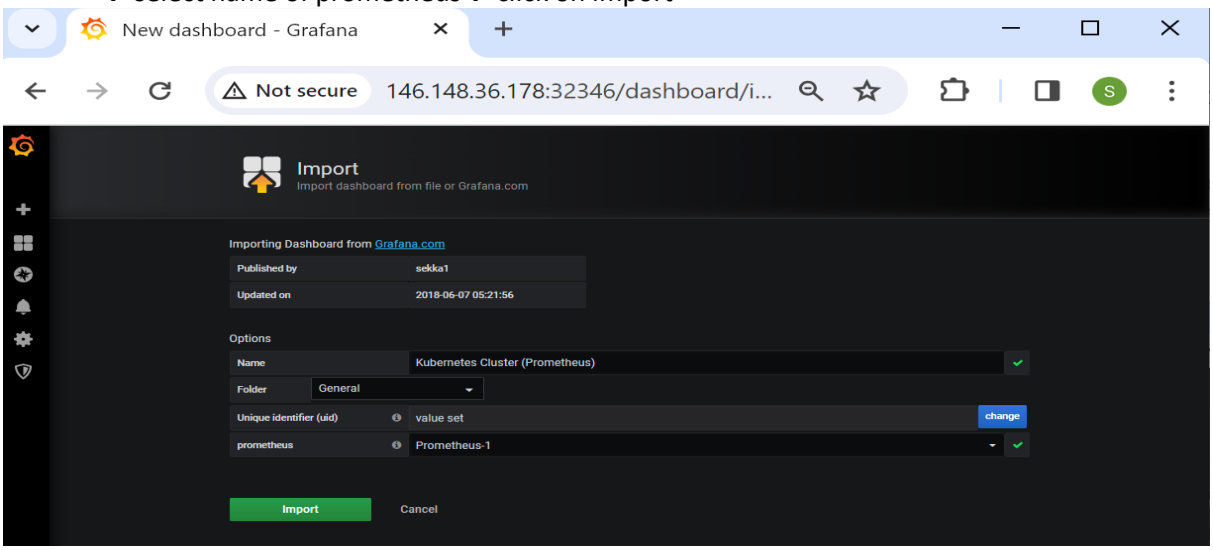
- Go to add data source → select Prometheus → configure URL for Prometheus http://146.148.36.178:30855/ →click on save & test



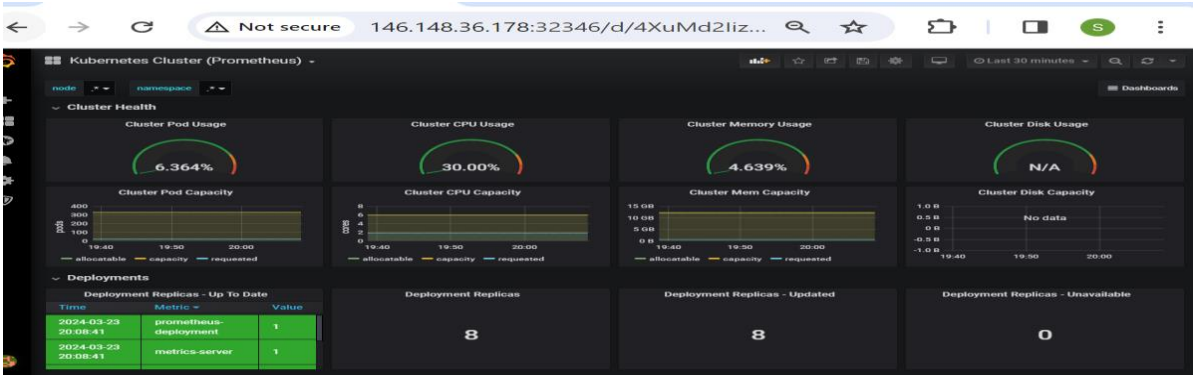
- Now we can create new dashboard using Dashboard→click on New Dashboard → Add Query → copy query from **Prometheus & paste**



- We can create use ready-made dashboard by taking from **Grafana** official web site https://grafana.com/grafana/dashboards/6417-kubernetes-cluster-prometheus/
- Copy ID of Dashboard → go to Garafana → New Dashboard → Import Dashboard → paste ID → select name of prometheus→ click on import



- It will give you GUI for you Kubernete cluster



- We can also give dashboard JSON to create Dashboard

59) What is Kubernetes Security?

- Kubernetes Security is based on the 4C's of cloud native security: Cloud, Cluster, Container, and Code
- **Cloud (or Corporate Datacenter/Colocation facility)**: The underlying physical infrastructure is the basis of Kubernetes security. Whether the cluster is built on one's own datacenter or a cloud provider, basic cloud provider (or physical security) best practices must be observed.
- **Cluster:** Securing a Kubernetes cluster involves both the configurable components such as the Kubernetes API and security of all the applications that are part of the cluster. Since most cloud-native applications are designed around micro services and APIs, applications are only as secure as the weakest link in the chain of services that comprise the entire application.
- **Container:** Container design best practices consist of: starting with the smallest code base possible (excluding unnecessary libraries or functions), avoiding granting unnecessary privileges to users in the container, and ensuring that containers are scanned for vulnerabilities at build time.
- **Code:** Code presents a major attack surface for any Kubernetes environment. Simple policies such as encrypting TCP using TLS handshakes, not exposing unused ports, scanning, and testing regularly can help prevent security issues from arising in a production environment.