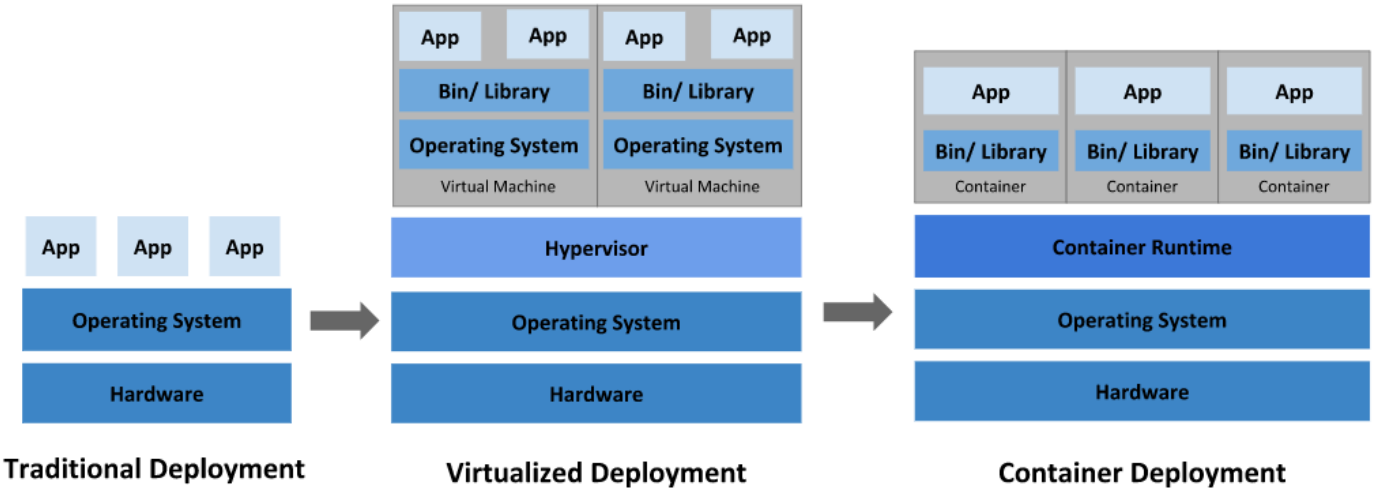
6.1) Kubernetes Detailed notes

Topic:Overview of Containers :-



Why do we need Kubernetes

Question or Concerns about containers

- Containers are a wonderful way of bundling and running your applications, But are they production ready?
- What would happen if the container or the Docker Host goes down?
- How to make containers available 24*7 ?
- How to handle loads during peak time for the applications ?
- How to replace containers without having downtime with new Docker Image based containers ?
- How to monitor containers?

Solution to above Questions or Concerns

Wouldn't it be good if there is some system which can help for handling all the questions/concerns raised in the above section. That is exactly what Kubernetes does.

- Kubernetes takes care of
    - Scaling requirements
    - failover
    - deployment patterns

- Kubernetes Provides
    - **Service Discovery & Load Balancing**
    - **Storage Orchestration**
    - **Automated rollouts and rollbacks**
    - **Automated bin packing**
    - **Self-Healing**
    - **Secret & Configuration Management**
- Kubernetes is not only for the open source community embraced containers, It is deeply embraced by the Cloud Providers.
    - Amazon Web Service offers **Elastic Kubernetes Services(EKS)**
    - Google Cloud platform offers **Google Kubernetes Engine(GKE)**
    - Microsoft Azure offers **Azure Kubernetes Services(AKS)**

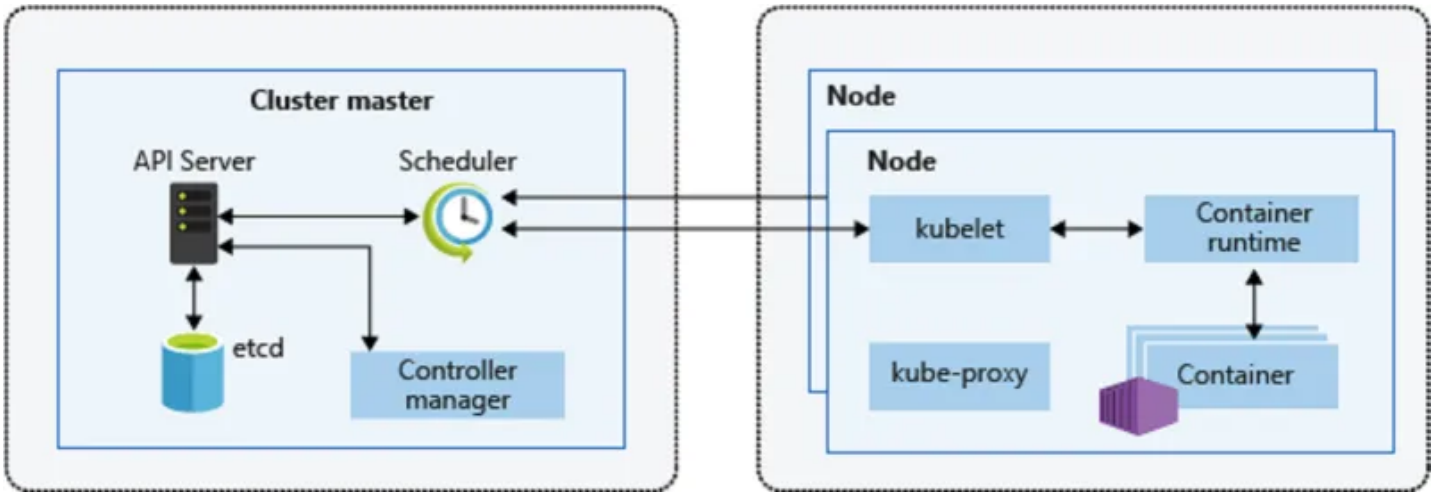
Topic:What is Kubernetes:-

- Kubernetes is a platform that manages container-based applications, their networking and storage components.
- In Kubernetes, we focus on the application workloads rather than the underlying infrastructure.
- Kubernetes provides a declarative approach to deployments, backed by a rich set of APIs for management operations.
- Cluster: Cluster is a collection of compute, storage and networking resources that Kubernetes uses to run workloads.
- Node: It is a single host. Now we can put the cluster as a collection of nodes.

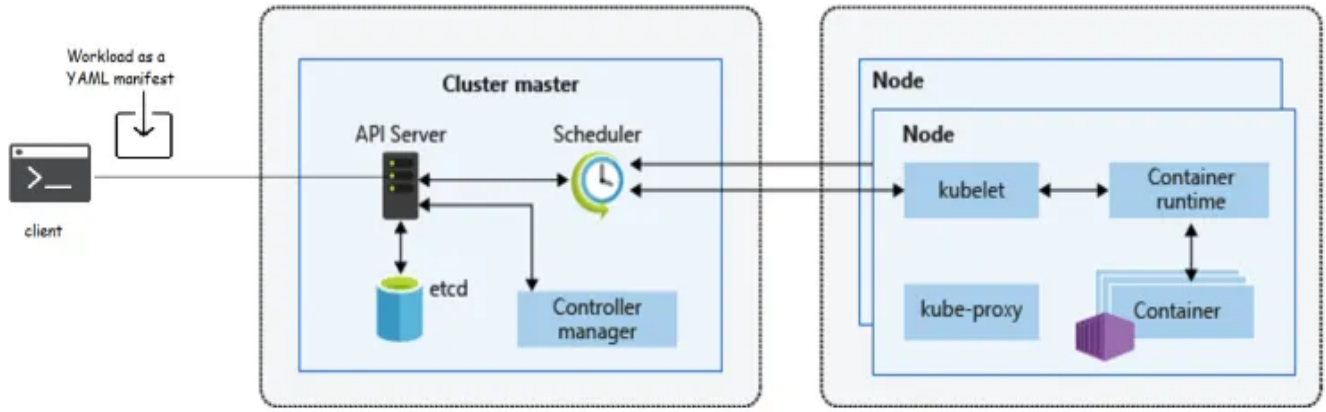
Kubernetes has two kinds of Nodes

- **Master**:
    - Provides core Kubernetes Services and orchestration to application workloads
- **Node**:
    - run your application workloads

Kubernetes Cluster Architecture



What would be our approach with workloads in Kubernetes?



- We would be describing our application as a YAML manifest and pass it to Kubernetes master from some client, and Cluster Master does the rest.

Pets	Cattle
Legacy Infrastructure	Cloud-Friendly Infrastructure
Pets are given names like grumpycat.petstore.com They are unique, lovingly hand raised and cared for When they get ill, you nurse them back to health	Cattle are given numbers like 10200713.cattlerancher.com They are almost identical to other cattle When they get ill, you replace them and get another
Infrastructure is a permanent fixture in the data center	Infrastructure is stateless, ephemeral, and transient
Infrastructure takes days to create, are serviced weekly, maintained for years, and requires migration projects to move	Infrastructure is instantiated, modified, destroyed and recreated in minutes from scratch using automated scripts
Infrastructure is modified during maintenance hours and generally requires special privileges such as root access	Infrastructure uses version-controlled scripts to modify any service without requiring root access or privileged logins
Infrastructure requires several different teams to coordinate and provision the full environment	Infrastructure is self-service with the ability to provision computing, network and storage services with a single click
Infrastructure is static, requiring excess capacity to be dormant for use during peak periods of demands	Infrastructure is elastic and scales automatically, expanding and contracting on-demand to service peak usage periods
Infrastructure is an capital expenditure that charges a fix amount regardless of usage patterns	Infrastructure is a operating expenditure that charges only for services when they are consumed

- k8s is a platform that is used with a huge number of services & capabilities that keeps growing.
- The core functionality of k8s is scheduled workloads in containers across your infra. In addition to this there are many functionalities that k8s brings in
    - Managing storage systems
    - Secret Management
    - Application health & readiness check
    - Replicating application instances
    - Horizontal pod Autoscaling
    - Cluster Autoscaling
    - Naming & Service Discovery
    - Balancing loads
    - Rolling updates
    - Resource Monitoring
    - Log Management
    - Application debugging
    - Authentication & Authorization

Topic:k8s components:-

https://kubernetes.io/docs/concepts/overview/components/

Kubernetes Master

Cluster:

- It is collection of hosts(nodes) that provide compute, memory, storage and networking resources
- k8s uses these resources to run application workloads.

Master Node:

- The master node is a single host in k8s cluster.
- This can be a physical server or virtual machine
- This is the control plane of k8s. This master is responsible for assigning work (container workloads) to be scheduled on nodes
- Kubernetes master runs on Linux node
- Master Components has i) kube-apiserver ii) etct iii) kube-scheduler iv) Controller manager v) Cloud-controller-manager

kube-apiserver

- This component is Central to Kubernetes. All communications between all components goes through the kube-apiserver
- This component is frontend of the Kubernetes control plane.
- This component exposes a REST API.
- We would interact with this component using **kubectl** by using the YAML files, which are also referred as manifests

etcd

- etcd stores the entire configuration and the state of the cluster.
- etcd is a consistent and highly available distributed key-value store.

kube-scheduler

- It watches for new work tasks and assigns them to healthy nodes in the cluster

Controller-Manager

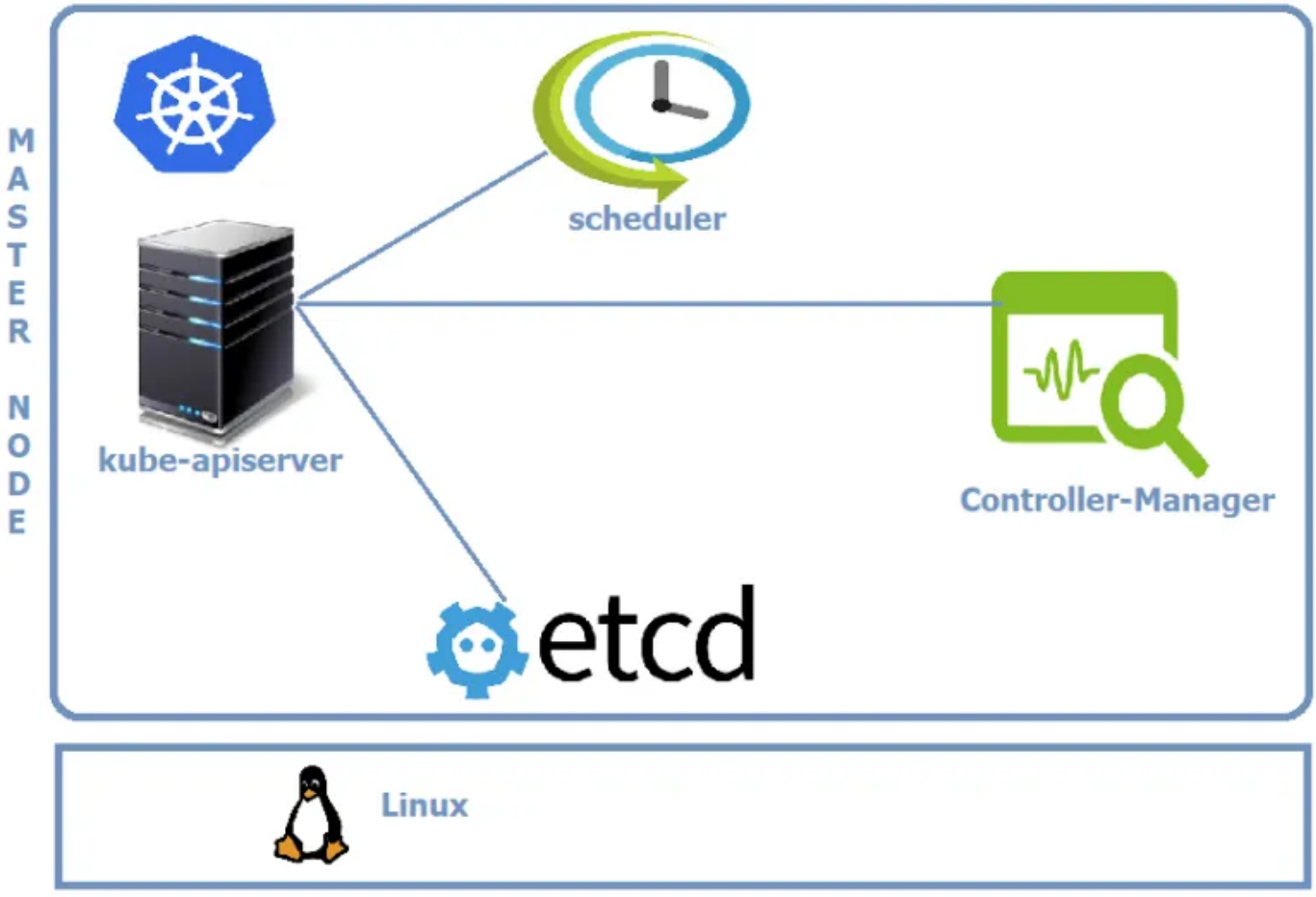
- It is responsible for maintaining desired states mentioned in the manifest.
- It looks like single component, but with in it has
    - Node Controller: for noticing & responding when node goes down

- Replication Controller: for maintaining the correct number of pods for every replication controller object.
- Endpoints Controller: Populates the Endpoints object

Cloud-Controller-manager

- It is responsible for underlying cloud specific controllers.

- If you run the Kubernetes on a supported cloud platform such as AWS, Azure or Google, your control plane runs the Cloud-Controller-Manager.



Node Components

Nodes:

- It is a single host in k8s cluster
- This can be physical server or virtual machine

- These are managed by k8s master and they do the work of executing workloads in k8s (They were referred as minions)
- Node is responsible for doing the work assigned.

kubelet

- This is an agent which runs on each node in the cluster.
- It watches for the instructions from API Server for new work assignments.
- If it can't run the task assigned, it reports back to the master and lets the control plane decide on the actions.
- It is responsible for the node registration process

Container Runtime

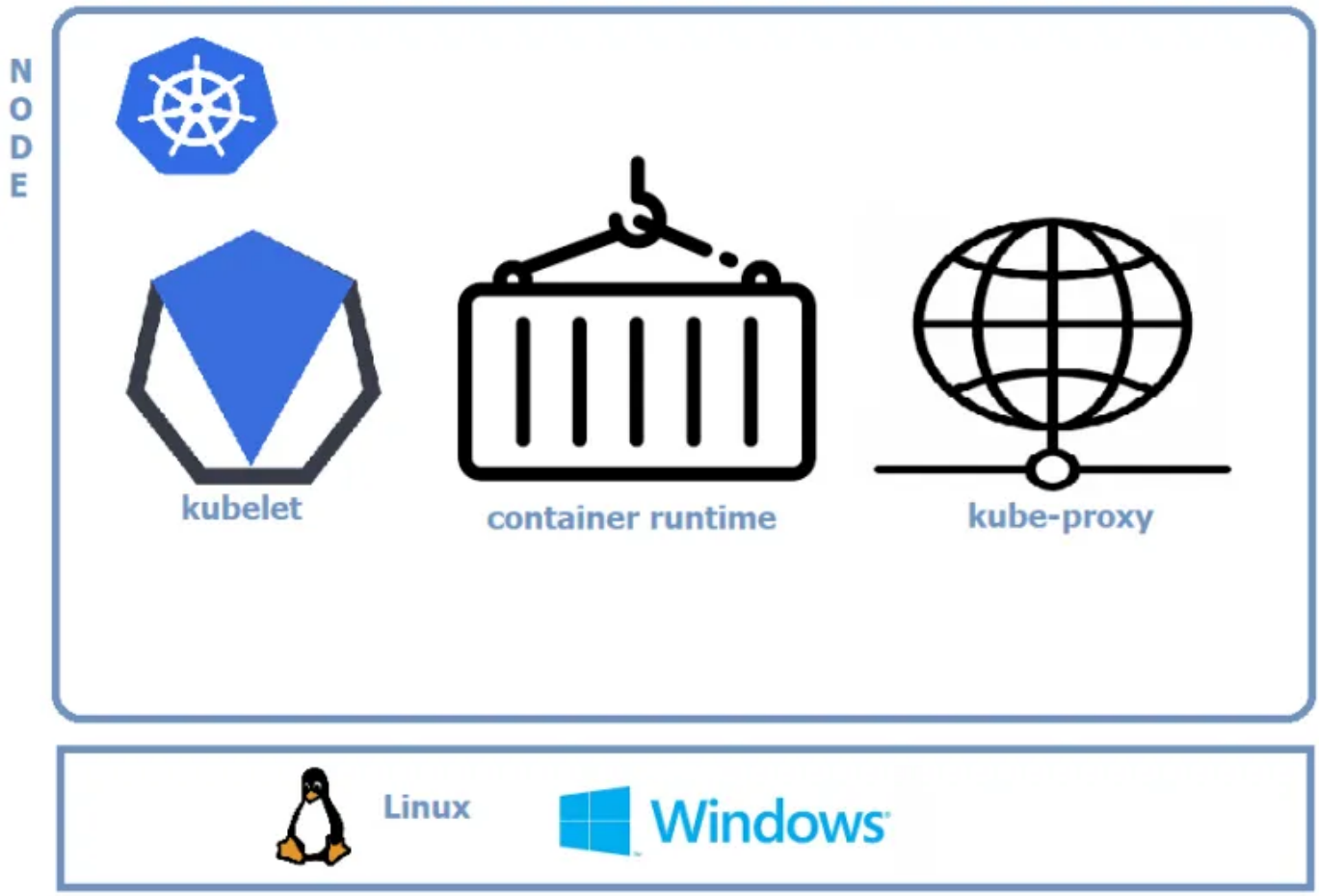
- This is software which is responsible for running containers.
- Some of them are **Docker, containerd, cri-o, rktlet**

kube-proxy

- Maintains the network rules on nodes
- This is responsible for networking on nodes.

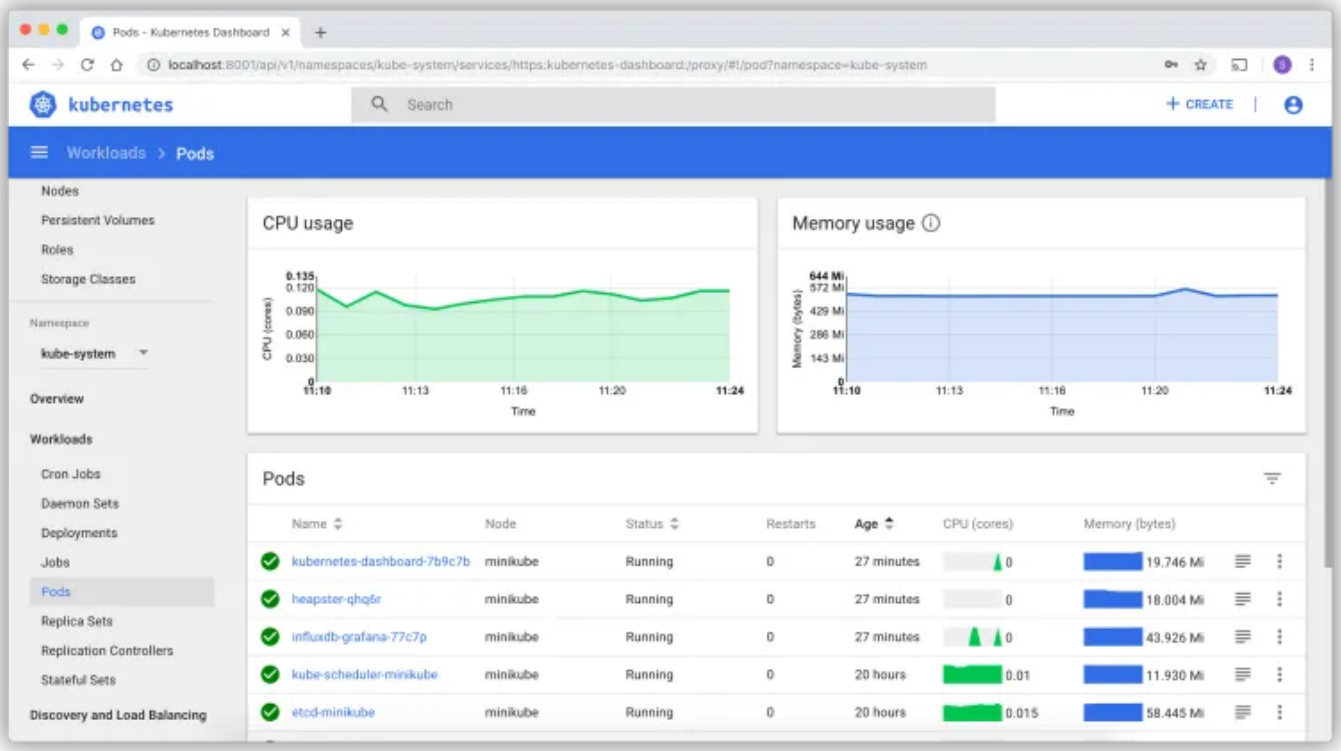
Cluster DNS

- Every Kubernetes Cluster has an internal DNS service
- This has static IP address that is hardcoded into every Pod on the cluster i.e. all Pods now how to find the DNS Server
- Services, Stateful Sets and Pods are registered with Cluster DNS.
- Cluster DNS is based on CoreDNS



Interesting ADD ONS

Web UI (Dashboard)



- Web based Kubernetes user interface
- Capabilities:
    - Deployment
    - View cluster resources
    - Troubleshooting

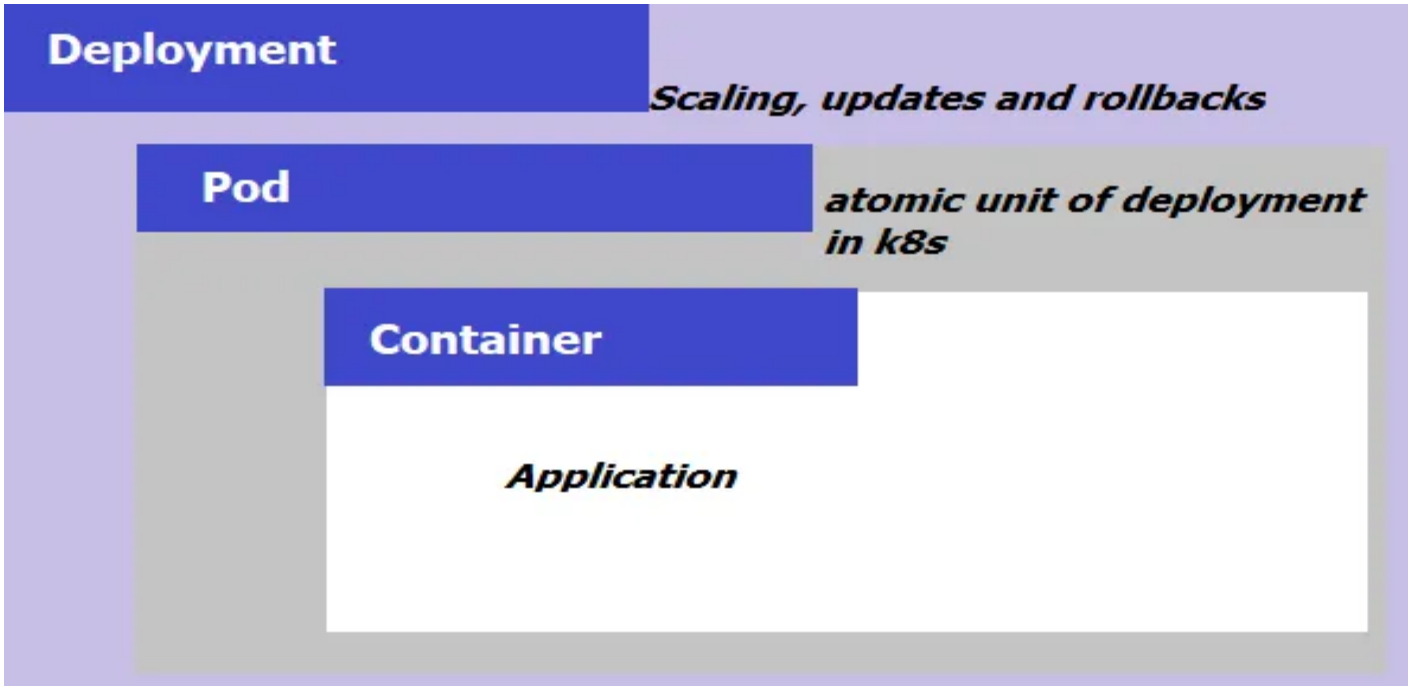
Container Resource Monitoring

- We will discuss about it in next series of articles

Cluster Level Logging

- We will discuss about it in next series of articles

How to Package Apps To Run on Kubernetes



- Package your application as a container image.
    - We would create a Image for the application
    - Push the image to Docker Registry
- Pod will Wrap container
- Pod is deployed into the Kubernetes cluster via Declarative Manifest file.

Pods:-

- A pod is a unit of work in k8s.
- Each pod contains one or more containers.
- Every pod gets a unique ip address and the containers in the pod have the same ip address.

Topic: K8s Installations:-

- Refer Here for devops blog installation steps
- Refer Here for official documentation
- Install docker and kubeadm,kubectl on all the three servers as mentioned in above documentation. Now login into master and execute as Refereed Over Here
- We need to execute “kubeadm init” it will generate a script,we need to follow according to that and need to create a folder for k8s and configure the network driver for that.
- Now login to kubernetes master and execute “kubeadm init”
- Make node kubeadm init commands login as normal user and create .kube folder and config as below

Your Kubernetes control-plane has initialized successfully!

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

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

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

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

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

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

```
kubeadm join 172.31.41.237:6443 --token 9g3h4c.oxuxjih0ee4f98g6 \
    --discovery-token-ca-cert-hash
sha256:95646829cf695d5829e9391e1df0453041d797800118f2c6b344e16c3fce041a
```

- As mentioned above if we execute “kubeadm init” it will display like above we need to execute below commands as normal users

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

- After that as mentioned in the lines go to the url. This url contain network drivers and that drivers installation commands
  {https://kubernetes.io/docs/concepts/cluster-administration/addons/}
- Generally we have network drivers like Flannel,kniter,Romana..etc in above we have selected weave-network
- I wish to install “weave-network” so we will go to the installation they will give the command
- For install weave-wetwork -network driver we need to execute {kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"}
- Then join the worker node by following command for ex

  {kubeadm join 172.31.78.198:6443 --token qbhg9d.v46u4elnoi65bzpp \
  --discovery-token-ca-cert-hash
  sha256:ba79a470521e5f13c4460e7def9a23936890c6bb8a725d86b088d83b334dbd44}

- Now join two nodes to cluster by executing kubeadm join command shown above
- After joining nodes, login into master and execute kubectl get nodes

```
ubuntu@ip-172-31-20-190:~$ kubectl get nodes
NAME                 STATUS   ROLES    AGE    VERSION
ip-172-31-17-40      Ready    <none>   33s    v1.19.4
ip-172-31-20-190     Ready    master   6m3s   v1.19.4
ip-172-31-27-107     Ready    <none>   79s    v1.19.4
ubuntu@ip-172-31-20-190:~$
```

Topic: Kubernetes commands:-

1) Kubectl version (or) kubectl version -o json {to know the version of k8s}
2) kubectl apply -f <yamal file> {To create a POD}
3) kubectl get pods {To know number of pods running}

```
root@ip-172-31-17-61:/mahi/k8s/podspecs# kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
gol-pod                             1/1     Running            0          2m41s
hello-world-rest-api-55c678fc85-fc566   0/1     ImagePullBackOff   0          4d22h
root@ip-172-31-17-61:/mahi/k8s/podspecs#
```

4) kubectl get pods -o wide {It will display pod clearly}

```
root@ip-172-31-17-61:/mahi/k8s/podspecs# kubectl get pods -o wide
NAME                                READY   STATUS             RESTARTS   AGE    IP          NODE               NOMINATED NODE   READINESS GATES
gol-pod                             1/1     Running            0          10m    10.44.0.1   ip-172-31-12-116   <none>           <none>
hello-world-rest-api-55c678fc85-fc566   0/1     ImagePullBackOff   0          4d22h  10.47.0.1   ip-172-31-15-137   <none>           <none>
root@ip-172-31-17-61:/mahi/k8s/podspecs#
```

5) kubectl describe pod gol-pod {It will describe entire pod details}

```
root@ip-172-31-17-61:/mahi/k8s/podspecs# kubectl describe pod gol-pod
Name:         gol-pod
Namespace:    default
Priority:     0
Node:         ip-172-31-12-116/172.31.12.116
Start Time:   Wed, 07 Apr 2021 09:39:41 +0530
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           10.44.0.1
IPs:
  IP:  10.44.0.1
Containers:
  my-gol-cont:
    Container ID:   docker://513a9c89b3e290d5ca3e9c32803add6b6a4fe8c219af9aa9b8baa439dc068b1f
    Image:          qualitythought/gameoflife:07112020
    Image ID:       docker-pullable://qualitythought/gameoflife@sha256:d1e9605cd63903c77ab9cb44a08cb3f5c8da6ec88bbabec2fde4fc0b70b419b7
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Wed, 07 Apr 2021 09:39:52 +0530
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-wltf2 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-wltf2:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-wltf2
    Optional:    false
QoS Class:       BestEffort
```

6) kubectl port-forward --address 0.0.0.0 <pod-name> source port(the port where the pod is running):destination port(port where we have to expose the pod)

```
^Croot@ip-172-31-17-61:/mahi/k8s/git-repo/KubernetesZone# kubectl port-forward --address 0.0.0.0 gameoflife 8080:8080
Forwarding from 0.0.0.0:8080 -> 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
```

NOTE:- To know the pod is running on which port etc details execute "kubectl describe <pod-name>" . When ever we do port-forward, default it will try to map on loopback adapter which will work when we are in local machine but whenever you are working on remote machines especially in cloud don't forget to put '0.0.0.0'

7) kubectl logs <pod-name> {to view the logs of the pod}

```
root@ip-172-31-17-61:/mahi/k8s/git-repo/KubernetesZone# kubectl logs gameoflife
08-Apr-2021 05:03:22.611 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version name:   Apache Tomcat/9.0.39
08-Apr-2021 05:03:22.616 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built:          Oct 6 2020 14:11:46 UTC
08-Apr-2021 05:03:22.617 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version number: 9.0.39.0
08-Apr-2021 05:03:22.617 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name:               Linux
08-Apr-2021 05:03:22.617 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version:            5.4.0-1038-aws
08-Apr-2021 05:03:22.617 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Architecture:          amd64
08-Apr-2021 05:03:22.618 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Java Home:             /usr/local/openjdk-8/jre
08-Apr-2021 05:03:22.618 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Version:           1.8.0_272-b10
08-Apr-2021 05:03:22.618 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Vendor:            Oracle Corporation
08-Apr-2021 05:03:22.618 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_BASE:         /usr/local/tomcat
08-Apr-2021 05:03:22.619 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log CATALINA_HOME:         /usr/local/tomcat
08-Apr-2021 05:03:22.620 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.util.logging.config.file=/usr/local/tomcat/
conf/logging.properties
08-Apr-2021 05:03:22.620 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.util.logging.manager=org.apache.juli.ClassL
oaderLogManager
08-Apr-2021 05:03:22.620 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djdk.tls.ephemeralDHKeySize=2048
08-Apr-2021 05:03:22.621 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Djava.protocol.handler.pkgs=org.apache.catalina.w
ebresources
08-Apr-2021 05:03:22.621 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Command line argument: -Dorg.apache.catalina.security.SecurityListener.UM
ASK=0027
```

8) kubectl exec <pod-name> -- command {to execute commands in pod using exec }

```
root@ip-172-31-17-61:/mahi/k8s/git-repo/KubernetesZone# kubectl exec gameoflife date
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
Thu Apr  8 07:36:33 UTC 2021
root@ip-172-31-17-61:/mahi/k8s/git-repo/KubernetesZone# kubectl exec gameoflife -- date
Thu Apr  8 07:38:16 UTC 2021
root@ip-172-31-17-61:/mahi/k8s/git-repo/KubernetesZone# kubectl exec gameoflife -- ls
BUILDING.txt
CONTRIBUTING.md
LICENSE
NOTICE
README.md
RELEASE-NOTES
RUNNING.txt
bin
conf
lib
logs
native-jni-lib
temp
webapps
webapps.dist
work
root@ip-172-31-17-61:/mahi/k8s/git-repo/KubernetesZone#
```

9) kubectl exec -it <pod-name> -- /bin/bash {TO get inside the docker container }

```
root@ip-172-31-17-61:/mahi/k8s/git-repo/KubernetesZone# kubectl exec -it gameoflife -- /bin/bash
root@gameoflife:/usr/local/tomcat# ls
BUILDING.txt  CONTRIBUTING.md  LICENSE  NOTICE  README.md  RELEASE-NOTES  RUNNING.txt  bin  conf  lib  logs  native-jni-lib  temp  webapps  webapps.dist  work
root@gameoflife:/usr/local/tomcat# ll
bash: ll: command not found
root@gameoflife:/usr/local/tomcat# ls -l
total 152
-rw-r--r-- 1 root root 18982 Oct  6  2020 BUILDING.txt
-rw-r--r-- 1 root root  5409 Oct  6  2020 CONTRIBUTING.md
-rw-r--r-- 1 root root 57092 Oct  6  2020 LICENSE
-rw-r--r-- 1 root root  2333 Oct  6  2020 NOTICE
-rw-r--r-- 1 root root  3257 Oct  6  2020 README.md
-rw-r--r-- 1 root root  6898 Oct  6  2020 RELEASE-NOTES
-rw-r--r-- 1 root root 16262 Oct  6  2020 RUNNING.txt
drwxr-xr-x 2 root root  4096 Nov  6 20:27 bin
drwxr-xr-x 1 root root  4096 Apr  8 05:03 conf
drwxr-xr-x 2 root root  4096 Nov  6 20:27 lib
drwxrwxrwx 1 root root  4096 Apr  8 05:03 logs
drwxr-xr-x 2 root root  4096 Nov  6 20:27 native-jni-lib
drwxrwxrwx 2 root root  4096 Nov  6 20:27 temp
drwxr-xr-x 1 root root  4096 Apr  8 05:03 webapps
drwxr-xr-x 7 root root  4096 Oct  6  2020 webapps.dist
drwxrwxrwx 1 root root  4096 Apr  8 05:03 work
root@gameoflife:/usr/local/tomcat# cd webapps
root@gameoflife:/usr/local/tomcat/webapps# ls
gameoflife  gameoflife.war
```

10) kubectl top nodes {to see the metrics of nodes} && kubectl top pods {to see the metrics of pods}

11) kubectl get rs {to view the replica-set}

```
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get rs
NAME                                DESIRED   CURRENT   READY   AGE
hello-world-rest-api-55c678fc85     1         1         0       6d8h
replocatio-set                      2         2         2       3m39s
```

12) kubectl describe rs replication-set {to describe the replication set}

```
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl describe rs replocatio-set
Name:         replocatio-set
Namespace:    default
Selector:     app=gol,version=1.1
Labels:       <none>
Annotations:  <none>
Replicas:     2 current / 2 desired
Pods Status:  2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=gol
           version=1.1
  Containers:
   gol:
    Image:        qualitythought/gameoflife:07112020
    Port:         8080/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age    From                    Message
  ----    ------            ----   ----                    -------
  Normal  SuccessfulCreate  3m59s  replicaset-controller  Created pod: replocatio-set-d2xv6
  Normal  SuccessfulCreate  3m59s  replicaset-controller  Created pod: replocatio-set-ffpfc
root@ip-172-31-17-61:/mahi/k8s/rs#
```

13) kubectl get rs -o wide {to see rs in detailed}

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get rs -o wide
NAME     DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES                                SELECTOR
gol-rs   2         2         2       80s   gol          qualitythought/gameoflife:07112020   app=gol,version=0711202
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$
```

14) kubectl get pods -o wide {to display pods in wide}

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get rs -o wide
NAME     DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES                                SELECTOR
gol-rs   2         2         2       80s   gol          qualitythought/gameoflife:07112020   app=gol,version=0711202
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE    IP          NODE              NOMINATED NODE   READINESS GATES
gol-rs-4px55  1/1     Running   0          108s   10.47.0.1   ip-172-31-17-40   <none>           <none>
gol-rs-mmx5l  1/1     Running   0          108s   10.44.0.1   ip-172-31-27-107  <none>           <none>
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$
```

15) kubectl get rs -w {to watch the replication set like what is happening}

```
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get rs -w
NAME                                DESIRED   CURRENT   READY   AGE
hello-world-rest-api-55c678fc85     1         1         0       6d9h
replocatio-set                      2         2         2       17m
replocatio-set                      2         2         1       19m
^Croot@ip-172-31-17-61:/mahi/k8s/rs#
```

16) kubectl scale replicaset <rs-name> --replicas=<number>

```
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl scale replicaset replocatio-set --replicas=5
replicaset.apps/replocatio-set scaled
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get ods
error: the server doesn't have a resource type "ods"
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get rs
NAME                                DESIRED   CURRENT   READY   AGE
hello-world-rest-api-55c678fc85     1         1         0       6d9h
replocatio-set                      5         5         5       30m
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
hello-world-rest-api-55c678fc85-ndd85   0/1     ImagePullBackOff   0          42m
replocatio-set-d2xv6                1/1     Terminating        0          30m
replocatio-set-ffpfc                1/1     Running            0          30m
replocatio-set-gmdlz                1/1     Running            0          18s
replocatio-set-pdr6q                1/1     Running            0          18s
replocatio-set-sjwrg                1/1     Running            0          18s
replocatio-set-z5cdk                1/1     Running            0          6m10s
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get pods -o wide
NAME                                READY   STATUS             RESTARTS   AGE     IP          NODE               NOMINATED NODE
DINESS GATES
hello-world-rest-api-55c678fc85-ndd85   0/1     ImagePullBackOff   0          42m     10.44.0.1   ip-172-31-12-116   <none>
ne>
replocatio-set-d2xv6                1/1     Terminating        0          30m     10.47.0.1   ip-172-31-15-137   <none>
ne>
replocatio-set-ffpfc                1/1     Running            0          30m     10.44.0.2   ip-172-31-12-116   <none>
ne>
replocatio-set-gmdlz                1/1     Running            0          26s     10.44.0.4   ip-172-31-12-116   <none>
ne>
replocatio-set-pdr6q                1/1     Running            0          26s     10.44.0.5   ip-172-31-12-116   <none>
ne>
replocatio-set-sjwrg                1/1     Running            0          26s     10.44.0.6   ip-172-31-12-116   <none>
ne>
replocatio-set-z5cdk                1/1     Running            0          6m18s   10.44.0.3   ip-172-31-12-116   <none>
ne>
root@ip-172-31-17-61:/mahi/k8s/rs#
```

17) kubectl autoscale rs <name of replicaset> --min=<number> --max=<number> --cpu-percent=<percentage> {to configure replicaset for auto scaling using imperative commands}

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl autoscale rs gol-rs --min=2 --max=5 --cpu-percent=
80
horizontalpodautoscaler.autoscaling/gol-rs autoscaled
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get rs
NAME     DESIRED   CURRENT   READY   AGE
gol-rs   2         2         2       11m
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get rs
NAME     DESIRED   CURRENT   READY   AGE
gol-rs   2         2         2       11m
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get hpa
NAME     REFERENCE           TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
gol-rs   ReplicaSet/gol-rs   <unknown>/80%   2         5         2          27s
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$
```

18) kubectl get hpa {horizontal pod autoscaler}

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl autoscale rs gol-rs --min=2 --max=5 --cpu-percent=
80
horizontalpodautoscaler.autoscaling/gol-rs autoscaled
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get rs
NAME     DESIRED   CURRENT   READY   AGE
gol-rs   2         2         2       11m
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get rs
NAME     DESIRED   CURRENT   READY   AGE
gol-rs   2         2         2       11m
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get hpa
NAME     REFERENCE           TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
gol-rs   ReplicaSet/gol-rs   <unknown>/80%   2         5         2          27s
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$
```

19) kubectl get pods --show-labels {To view names of the labels}

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get pods --show-labels
NAME                READY   STATUS    RESTARTS   AGE   LABELS
gol-rs-label-2hrnf   1/1     Running   0          62s   app=gol,version=07112020
gol-rs-label-9gkjh   1/1     Running   0          62s   app=gol,version=07112020
gol-rs-label-9l9pg   1/1     Running   0          62s   app=gol,version=07112020
gol-rs-label-ln7p7   1/1     Running   0          62s   app=gol,version=07112020
gol-rs-label-spxqj   1/1     Running   0          62s   app=gol,version=07112020
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$
```

20) kubectl get pods --selector=”app=<name>” {to view the particular pods with selected name/label}

```
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get pods --selector="app=gol"
NAME                READY   STATUS    RESTARTS   AGE
gol-rs-label-2mncp   1/1     Running   0          8m31s
gol-rs-label-dflkp   1/1     Running   0          8m31s
gol-rs-label-gdfj2   1/1     Running   0          8m31s
gol-rs-label-k4rmx   1/1     Running   0          8m31s
gol-rs-label-wrjff   1/1     Running   0          8m31s
root@ip-172-31-17-61:/mahi/k8s/rs#
```

21) Kubectl get pods --selectors in (appname,app-name)

```
ubuntu@ip-172-31-20-190:~/KubernetesZone$ kubectl get pods --selector="app=jenkins"
NAME                   READY   STATUS    RESTARTS   AGE
jenkins-rs-label-h854n   1/1     Running   0          82s
jenkins-rs-label-w2x82   1/1     Running   0          82s
ubuntu@ip-172-31-20-190:~/KubernetesZone$ kubectl get pods --selector="app in (jenkins, gol)"
NAME                   READY   STATUS    RESTARTS   AGE
gol-rs-label-2hrnf       1/1     Running   0          7m59s
gol-rs-label-9gkjh       1/1     Running   0          7m59s
gol-rs-label-9l9pg       1/1     Running   0          7m59s
gol-rs-label-ln7p7       1/1     Running   0          7m59s
gol-rs-label-spxqj       1/1     Running   0          7m59s
jenkins-rs-label-h854n   1/1     Running   0          108s
jenkins-rs-label-w2x82   1/1     Running   0          108s
ubuntu@ip-172-31-20-190:~/KubernetesZone$
```

22) kubectl api-resources {To know available api-resources}

```
root@ip-172-31-17-61:/home/ravi# 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/v1                         true         HorizontalPodAutoscaler
cronjobs                          cj           batch/v1beta1                          true         CronJob
```

23) kubectl get svc {to get service object nodeport, etc}

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ ls
gol-dev-rs.yaml  gol-dev-svc.yaml  jenkins-qa-rs.yaml  jenkins-test-svc.yaml
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl apply -f gol-dev-svc.yaml
service/gol-dev created
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
gol-dev      NodePort    10.100.108.213   <none>        8080:32766/TCP   7s
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          5d23h
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl apply -f jenkins-test-svc.yaml
service/jenkins-test created
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl get svc
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
gol-dev        NodePort    10.100.108.213   <none>        8080:32766/TCP   2m22s
jenkins-test   NodePort    10.96.163.64     <none>        8080:32765/TCP   11s
kubernetes     ClusterIP   10.96.0.1        <none>        443/TCP          5d23h
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$
```

24) kubectl get namespaces (or) kubectl get ns {To view all name-spaces}

```
root@ip-172-31-27-144:/home/ubuntu# kubectl get ns
NAME              STATUS   AGE
default           Active   6h26m
kube-node-lease   Active   6h26m
kube-public       Active   6h26m
kube-system       Active   6h26m
root@ip-172-31-27-144:/home/ubuntu#
```

25) kubectl get sts {to get statefuset}

```
root@ip-172-31-27-144:/mahi/k8s/statefull# kubectl get sts
NAME               READY   AGE
statefulset-demo   3/3     3m3s
root@ip-172-31-27-144:/mahi/k8s/statefull# kubectl get pods
NAME                 READY   STATUS    RESTARTS   AGE
statefulset-demo-0   1/1     Running   0          2m31s
statefulset-demo-1   1/1     Running   0          2m29s
statefulset-demo-2   1/1     Running   0          2m27s
root@ip-172-31-27-144:/mahi/k8s/statefull# kubectl get pods -o wide
NAME                 READY   STATUS    RESTARTS   AGE     IP              NODE                              NOMIN
ATED NODE   READINESS GATES
statefulset-demo-0   1/1     Running   0          2m45s   192.168.49.114   ip-192-168-34-80.ec2.internal   <none
>            <none>
statefulset-demo-1   1/1     Running   0          2m43s   192.168.21.0     ip-192-168-8-224.ec2.internal   <none
>            <none>
statefulset-demo-2   1/1     Running   0          2m41s   192.168.42.229   ip-192-168-34-80.ec2.internal   <none
>            <none>
root@ip-172-31-27-144:/mahi/k8s/statefull#
```

26) kubectl get ds {to get demonset}

```
qtkhaja@Azure:~$ kubectl apply -f daemonset.yaml
daemonset.apps/daemonset-demo created
qtkhaja@Azure:~$ kubectl api-resources | grep daemon
daemonsets                        ds          apps                           true         DaemonSet
qtkhaja@Azure:~$ kubectl get ds
NAME             DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset-demo   3         3         3       3            3           <none>          32s
qtkhaja@Azure:~$ kubectl get pods
NAME                   READY   STATUS    RESTARTS   AGE
daemonset-demo-4vmbn   1/1     Running   0          39s
daemonset-demo-b66cp   1/1     Running   0          39s
daemonset-demo-rwsz9   1/1     Running   0          39s
qtkhaja@Azure:~$ kubectl get pods -o wide
NAME                   READY   STATUS    RESTARTS   AGE   IP           NODE                                NOMINATED NODE   READINESS
GATES
daemonset-demo-4vmbn   1/1     Running   0          46s   10.244.1.3   aks-nodepool1-34786526-vmss000001   <none>           <none>
daemonset-demo-b66cp   1/1     Running   0          46s   10.244.2.3   aks-nodepool1-34786526-vmss000002   <none>           <none>
daemonset-demo-rwsz9   1/1     Running   0          46s   10.244.0.17  aks-nodepool1-34786526-vmss000000   <none>           <none>
qtkhaja@Azure:~$
```

27) kubectl get deploy {to get deployment set}

```
root@ip-172-31-27-144:/mahi/k8s/controller# kubectl get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
gameoflife-deploy   3/3     3            3           54m
root@ip-172-31-27-144:/mahi/k8s/controller#
```

28) Kubectl describe rs <name of rs> {to describe replication set}

```
root@ip-172-31-27-144:/mahi/k8s/controller# kubectl describe rs gameoflife-deploy
Name:           gameoflife-deploy-754b466d7f
Namespace:      default
Selector:       app=gameoflife,pod-template-hash=754b466d7f,version=1.0
Labels:         app=gameoflife
                pod-template-hash=754b466d7f
                version=1.0
Annotations:    deployment.kubernetes.io/desired-replicas: 3
                deployment.kubernetes.io/max-replicas: 4
                deployment.kubernetes.io/revision: 1
Controlled By:  Deployment/gameoflife-deploy
Replicas:       3 current / 3 desired
Pods Status:    3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=gameoflife
           pod-template-hash=754b466d7f
           version=1.0
  Containers:
   gol-dev:
    Image:        qualitythought/gameoflife:07112020
    Port:         8080/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  56m   replicaset-controller  Created pod: gameoflife-deploy-754b466d7f-k7jxd
  Normal  SuccessfulCreate  56m   replicaset-controller  Created pod: gameoflife-deploy-754b466d7f-jjsmg
  Normal  SuccessfulCreate  56m   replicaset-controller  Created pod: gameoflife-deploy-754b466d7f-9dwpf
```

Topic:concepts in kubernetes:-

kubectl

- This is a command-line utility which we would be using to interact with k8s cluster.
- This utility creates api-request internally and forwards them to kube-api server
- Refer Here for kubectl cheat sheet.

Kubernetes Objects

- K8s object is anything which can be stored in etcd, As a key-value pair.
- *k8s objects* are persistent entities in the k8s system.
- These objects are used to represent the state of your cluster
- Every k8s object has

    - Specification: This is also referred as object Spec
    - Status: This is also referred as object status
- How can we describe a k8s object (Specification):
    - K8s objects can be describe in the json format using k8s rest api or in yaml format via kubectl
    - For api version Refer Here
    - Every k8s object can be described using .yaml (specification) or json when using a rest api and passed across to the kube-api server via kubectl or k8s REST API. The cluster responds back with status
    - While describing object spec, we will have some required fields
        - apiVersion: Which version of k8s api are you using to create this object
        - kind: what kind of k8s object do you want to create
        - metadata: Data which is helpful in uniquely identifying the object, here we will be have *name*, *UID* and optional *namespace*
        - spec: What is your desired state

K8s Workloads

- Workload is an application running on k8s cluster.
- A Pod has a defined lifecycle. If the pod fails, we need to create the new pod running the same application, K8s has workload resources which can maintain desired state of pods
- Workload resources are
    - Deployments
    - Replica Sets
    - Stateful Sets
    - Daemon Sets
    - Job and Cron Job

The Pod Manifest (pod yaml file)

- Pods are described in a Pod Manifest which is a text-file representation of k8s API Object.
- Creating a Pod in a imperative kubectl run command

```
ubuntu@ip-172-31-20-190:~$ kubectl run gameoflife --image=qualitythought/gameoflife:07112020
pod/gameoflife created
ubuntu@ip-172-31-20-190:~$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
gameoflife   1/1     Running   0          8s
ubuntu@ip-172-31-20-190:~$ kubectl delete pods gameoflife
pod "gameoflife" deleted
ubuntu@ip-172-31-20-190:~$
```

- Creating a Pod using a manifest file.
- Refer Here for the pod manifest

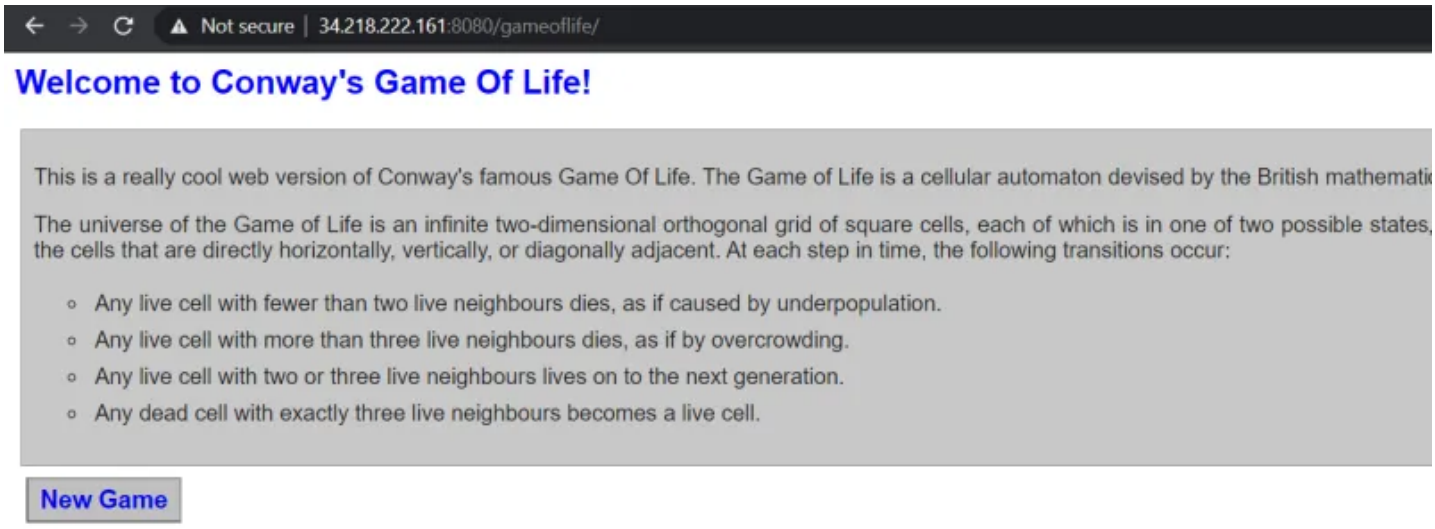
```
 name: gameoflife
spec:
  containers:
    - image: qualitythought.in/gameoflife:07112020
    - image: qualitythought/gameoflife:07112020
      name: gol
      ports:
        - containerPort: 8080
```

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$ kubectl apply -f gameoflife-pod.yaml
pod/gameoflife created
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
gameoflife   1/1     Running   0          5s
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$
```

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$ kubectl describe pods gameoflife
Name:         gameoflife
Namespace:    default
Priority:     0
Node:         ip-172-31-27-107/172.31.27.107
Start Time:   Sat, 14 Nov 2020 02:23:05 +0000
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           10.44.0.1
IPs:
  IP:  10.44.0.1
Containers:
  gol:
    Container ID:   docker://ba9caf8743cb85713bebafb50de1cd413335786ab2a004392a96b54b24f119af
    Image:          qualitythought/gameoflife:07112020
    Image ID:       docker-pullable://qualitythought/gameoflife@sha256:d1e9605cd63903c77ab9cb44a08cb3f5c8da6ec88bbabe
c2fde4fc0b70b419b7
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sat, 14 Nov 2020 02:23:06 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-559v4 (ro)
```

- Accessing your pod: {if we can observe the pod ip address and node ip address is different also those are in two different networks. So to access that pod we need to map the ip to port “kubectl port-forward --address 0.0.0.0 gameoflife 8080:8080”}

```
Usage:
  kubectl port-forward TYPE/NAME [options] [LOCAL_PORT:]REMOTE_PORT
[...[LOCAL_PORT_N:]REMOTE_PORT_N]

Use "kubectl options" for a list of global command-line options (applies to all commands).
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$ kubectl port-forward --address 0.0.0.0 gameoflife 8080:8080
Forwarding from 0.0.0.0:8080 -> 8080
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
```



* Logs information of your pod:

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$ kubectl logs gameoflife
14-Nov-2020 02:23:06.767 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version name:   Apa
che Tomcat/9.0.39
14-Nov-2020 02:23:06.772 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server built:          Oct
 6 2020 14:11:46 UTC
14-Nov-2020 02:23:06.773 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Server version number: 9.0
.39.0
14-Nov-2020 02:23:06.773 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Name:               Lin
ux
14-Nov-2020 02:23:06.773 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log OS Version:            5.4
.0-1029-aws
14-Nov-2020 02:23:06.774 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Architecture:          amd
64
14-Nov-2020 02:23:06.774 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log Java Home:             /us
r/local/openjdk-8/jre
14-Nov-2020 02:23:06.775 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Version:           1.8
.0_272-b10
14-Nov-2020 02:23:06.775 INFO [main] org.apache.catalina.startup.VersionLoggerListener.log JVM Vendor:            Ora
cle Corporation
```

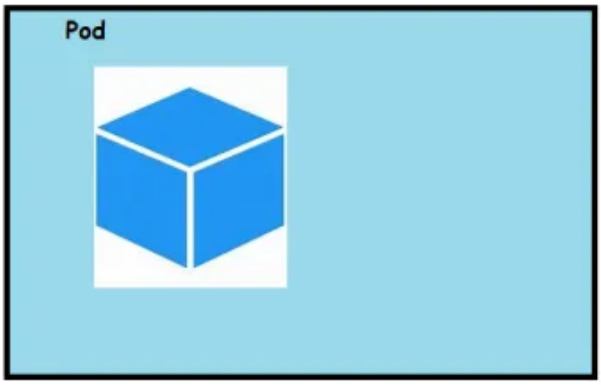
- Running commands in your container with exec

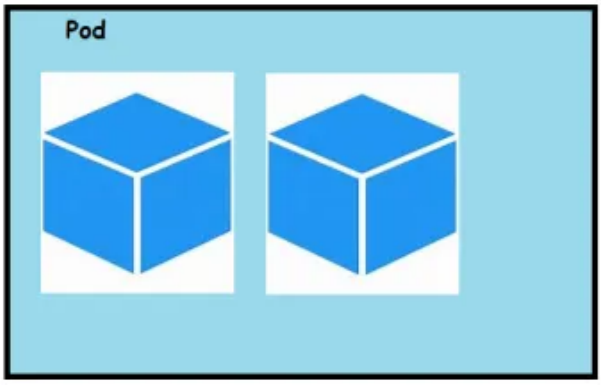
```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$ kubectl exec gameoflife date
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAN
D] instead.
Sat Nov 14 02:33:41 UTC 2020
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$ kubectl exec gameoflife -- date
Sat Nov 14 02:33:57 UTC 2020
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$ kubectl exec gameoflife -- ls
BUILDING.txt
CONTRIBUTING.md
LICENSE
NOTICE
README.md
RELEASE-NOTES
RUNNING.txt
bin
conf
lib
logs
native-jni-lib
temp
webapps
webapps.dist
work
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$
```

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/podspecs$ kubectl exec -it gameoflife -- /bin/bash
root@gameoflife:/usr/local/tomcat# ps
bash: ps: command not found
root@gameoflife:/usr/local/tomcat# ls
BUILDING.txt     LICENSE  README.md      RUNNING.txt  conf  logs           temp     webapps.dist
CONTRIBUTING.md  NOTICE   RELEASE-NOTES  bin          lib   native-jni-lib  webapps  work
root@gameoflife:/usr/local/tomcat# cd webapps
root@gameoflife:/usr/local/tomcat/webapps# ls
gameoflife  gameoflife.war
root@gameoflife:/usr/local/tomcat/webapps#
```

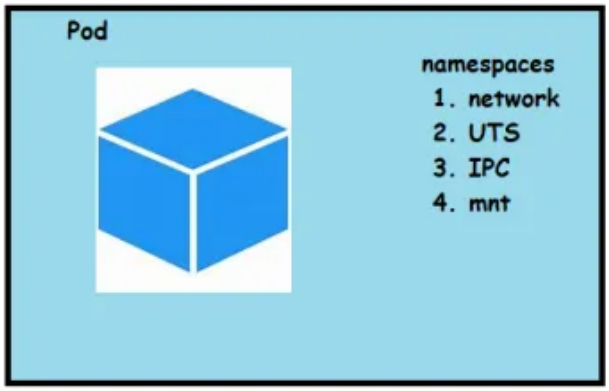
K8s Pods

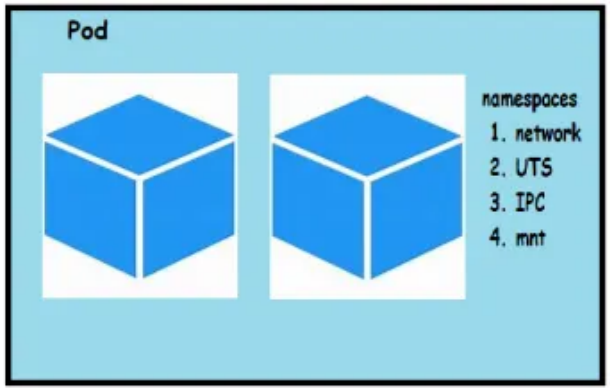
- Pods are the atomic unit of computing that you create in k8s
- A pod represents an application/microservice running in k8s cluster.
- Pods will have containers in it.





- Each container within a pod runs its own cgroup, but they share a number of Linux namespaces like network, mnt, UTS, IPC, etc.
- These namespaces are for pods. Not for containers.





In k8s to create any object we have two ways:-

Imperative:

- We can create objects by executing commands Refer Here

Declarative:

- We create an object specification in a yaml file.

- Send/apply this yaml which has a specification & k8s master will do the rest to create/maintain the specification sent
- To create this specification we will be using k8s api reference

Resource Management:

- Resource Requests for Minimum required resources and maximum limits Refer Here
- https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
- Refer Here for resource limits in a pod.

```
 containers:
  - image: qualitythought/gameoflife:07112020
    name: gol
    resources:
      requests:
        cpu: "500m"
        memory: "128Mi"
      limits:
        cpu: "1000m"
        memory: "512Mi"
    ports:
      - containerPort: 8080
        name: golhttpport
        protocol: TCP
        protocol: TCP
```

- Refer Here for multi container pod

```
---
apiVersion: v1
kind: Pod
metadata:
 name: gameoflife
spec:
 containers:
  - image: qualitythought/gameoflife:07112020
    name: gol
    resources:
      requests:
        cpu: "500m"
        memory: "128Mi"
      limits:
        cpu: "1000m"
```

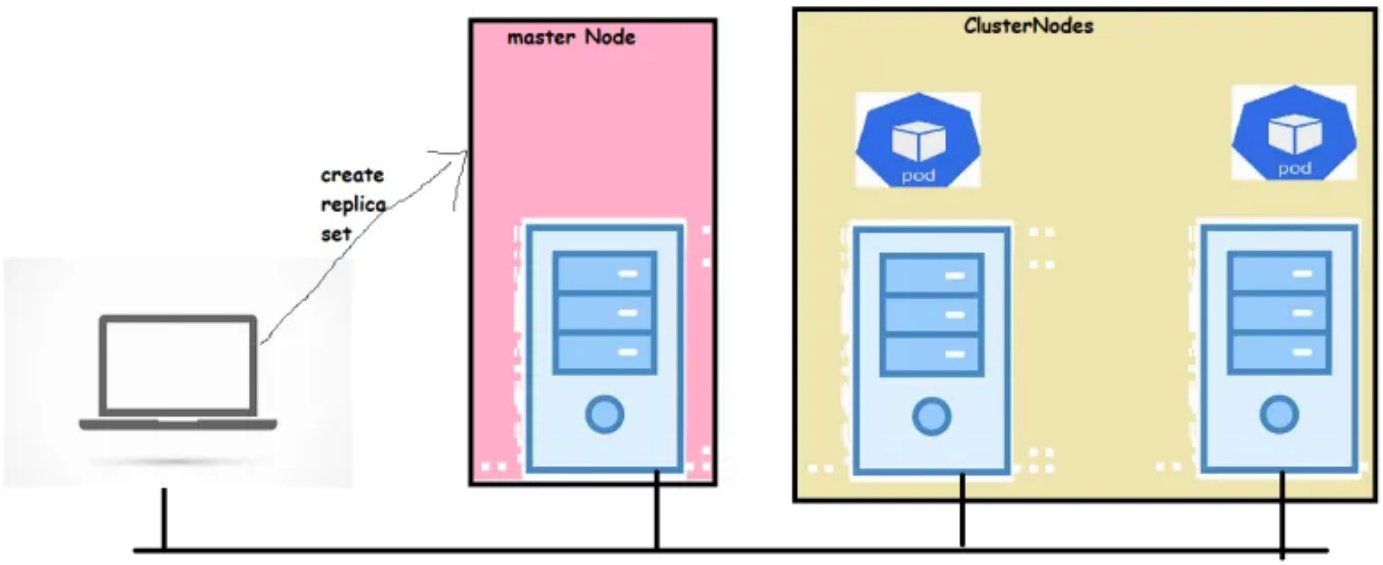
```
      memory: "512Mi"
  ports:
    - containerPort: 8080
      name: golhttpport
      protocol: TCP
- image: httpd
  name: httpd
  ports:
    - containerPort: 80
      name: httpport
      protocol: TCP
```

K8s API Groups

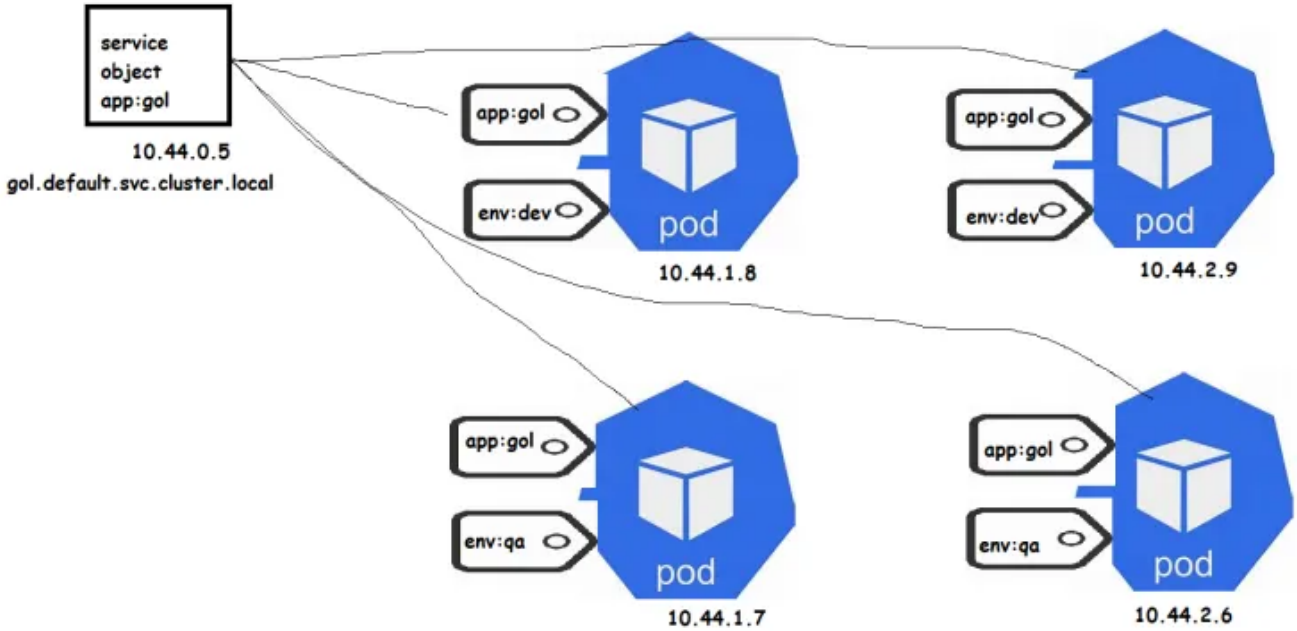
- K8s groups api's as api groups.
- The major purpose is to extend k8s
- Refer Here for api groups
- To use resources from api groups the naming convention is `<GROUP_NAME>/<VERSION>` in yaml file
    - batch/v1
    - extensions/v1beta1
    - rbac.authorization.k8s.io/v1
- To use resource from api groups in the REST api, convention is `/apis/<GROUP_NAME>/<VERSION>`
    - /apis/batch/v1
    - /apis/extensions/v1beta1
    - /apis/rbac.authorization.k8s.io/v1
- If the resources are part of core group we can directly write the version `apiVersion: v1`

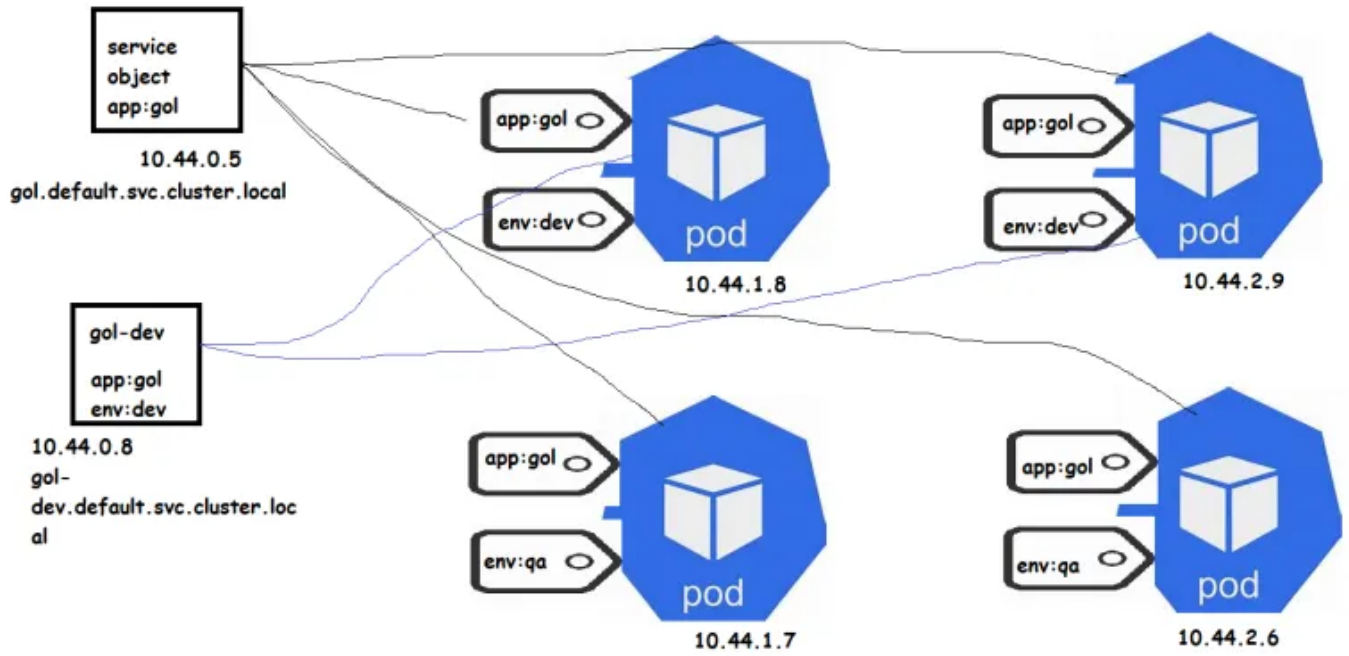
Topic:- K8s Service

- We have created a replica set with n pods. How do we access the application running inside the pods.
- In normal scenarios for these kinds of problems, we use a load balancer.



- In K8s we use Service Object. A *Service* is a way to create a named label selector. This Service Object will recieve an IP Address and also DNS Name





- So now let's create two replica-sets and two services.
- Lets understand service types
    a. Cluster IP
    b. Node Port
    c. load balancer
    d. external name

EX:-

- now we create 4 objects 2 for replica set in dev environment and test environment. And 2 for services like dev environment and test environment.

```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: gol-dev
spec:
  minReadySeconds: 10
  replicas: 2
  selector:
    matchLabels:
      app: gol
      env: dev
```

```
  template:
    metadata:
      labels:
        app: gol
        env: dev
    spec:
      containers:
        - image: qualitythought/gameoflife:07112020
          name: gol-dev
          ports:
            - containerPort: 8080
              protocol: TCP
##########################################################################


---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: jenkins-qa
spec:
  minReadySeconds: 10
  replicas: 2
  selector:
    matchLabels:
      app: jenkins
      env: qa
  template:
    metadata:
      labels:
        app: jenkins
        env: qa
    spec:
      containers:
        - image: jenkins:2.60.3
          name: jenkins-qa
          ports:
            - containerPort: 8080
              protocol: TCP


##########################################################################
```

- In the below example in the field of ports.

  port belongs to ‘port of your service’
  targetPort belongs to ‘port of your pod’
  nodePort is ‘port of your node. By using which we will access this service’

```
---
apiVersion: v1
kind: Service
metadata:
  name: gol-dev
spec:
  type: NodePort
  selector:
    app: gol
    env: dev
  ports:
    - targetPort: 8080
      port: 8080
      nodePort: 32766

#############################################################


---
apiVersion: v1
kind: Service
metadata:
  name: jenkins-test
spec:
  selector:
    app: jenkins
    env: qa
  type: NodePort
  ports:
    - targetPort: 8080
      port: 8080
      nodePort: 32765
#############################################################
```

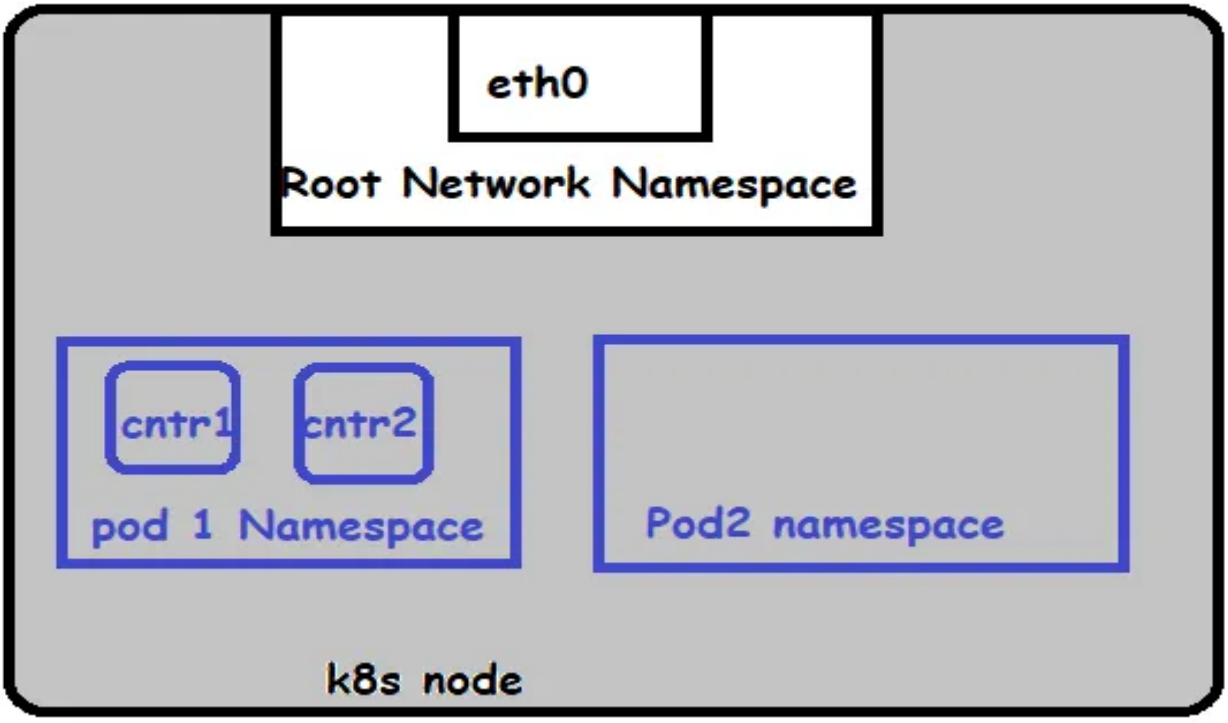
```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl apply -f jenkins-
jenkins-qa-rs.yaml     jenkins-test-svc.yaml
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl apply -f jenkins-qa-rs.yaml
replicaset.apps/jenkins-qa created
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl apply -f gol-dev-rs.yaml
replicaset.apps/gol-dev created
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl get pods
NAME               READY   STATUS    RESTARTS   AGE
gol-dev-pn7xv      1/1     Running   0          6s
gol-dev-wp8fw      1/1     Running   0          6s
jenkins-qa-d77hx   1/1     Running   0          17s
jenkins-qa-zj2fn   1/1     Running   0          17s
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl get rs
NAME         DESIRED   CURRENT   READY   AGE
gol-dev      2         2         2       12s
jenkins-qa   2         2         2       23s
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$
```

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ ls
gol-dev-rs.yaml  gol-dev-svc.yaml  jenkins-qa-rs.yaml  jenkins-test-svc.yaml
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl apply -f gol-dev-svc.yaml
service/gol-dev created
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
gol-dev      NodePort    10.100.108.213   <none>        8080:32766/TCP   7s
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP          5d23h
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl apply -f jenkins-test-svc.yaml
service/jenkins-test created
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$ kubectl get svc
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
gol-dev        NodePort    10.100.108.213   <none>        8080:32766/TCP   2m22s
jenkins-test   NodePort    10.96.163.64     <none>        8080:32765/TCP   11s
kubernetes     ClusterIP   10.96.0.1        <none>        443/TCP          5d23h
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/service$
```

Topic: Kubernetes Networking Model (KNM)

- KNM requirements are
  - all pods can communicate with all other pods without using NAT
  - all Nodes can communicate with all Pods without NAT
  - the IP that Pod sees itself is the same IP that other see it as
- Lets understand the following
  - Container to Container networking
  - Pod-to-Pod Networking
  - Pod-to-svc networking
  - Internet to SVC networking

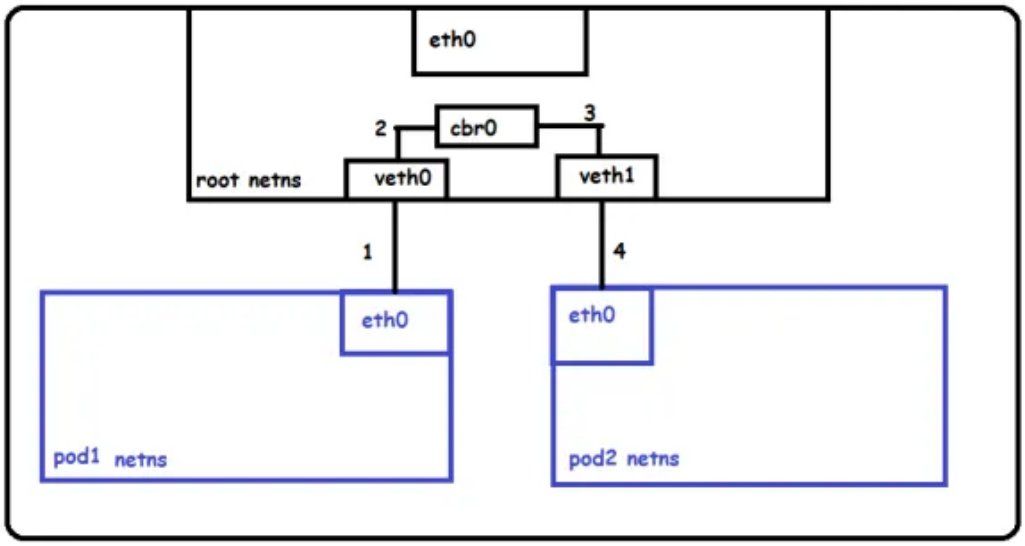
Container to Container Networking



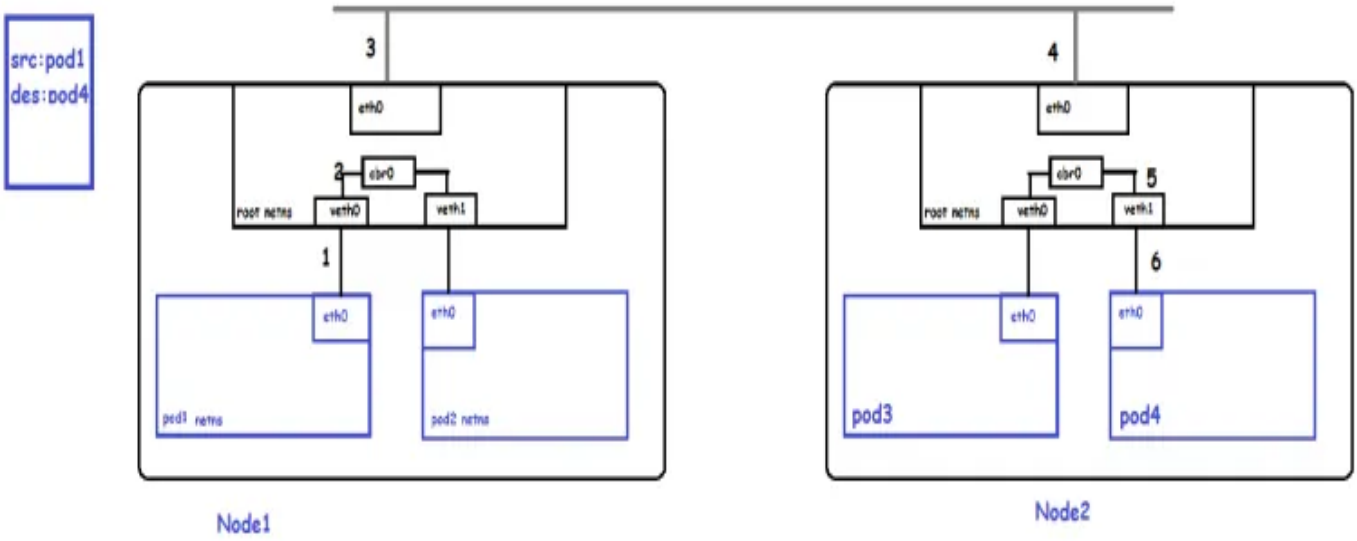
Pod to Pod Networking

- **Both Pods in Same Node**, will communicate like below one container in pod1 is communicating to another container in pod2. For that node it has a root network, each pod has its own network namespace.
- Because of this network namespace each of this pod will get ethernet interface like ‘eth0,eth0’
- This ‘etc’ is connected to virtual-eather net device ‘veth0’
- These all virtual ethernets which are created in our node n/w name spaces are connected to each other via a bridge ‘cbr0’
- Now we need to send a pocket from pod1 to pod2
- Now what will happen is pocket will go to eth interface and reached to pod1’ eth0
- Your eth0 is connected to veth0 and it will reach over there and forward to ‘cbr0’
- Then the container bridge will send an ‘arp’ packet to all other networks in that node except source n/w.
- ‘veth1’ will know that this is pod2’s ip address which we are trying to get and it will responds back to container bridge

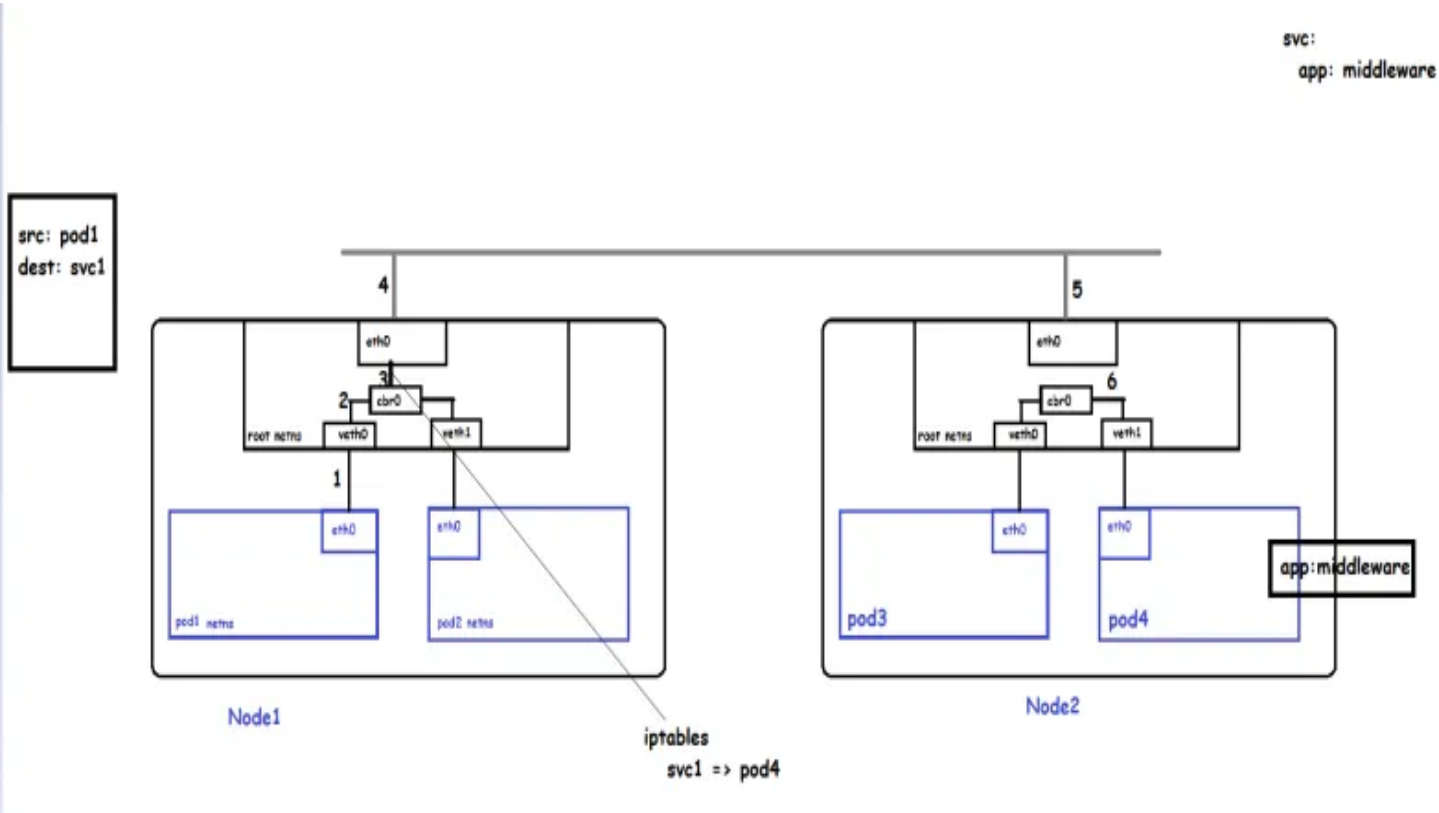
- So this bridge source will send an ‘ARP’ Address resolution protocol to all other networks and any network will respond to that then it will take that and will give to source



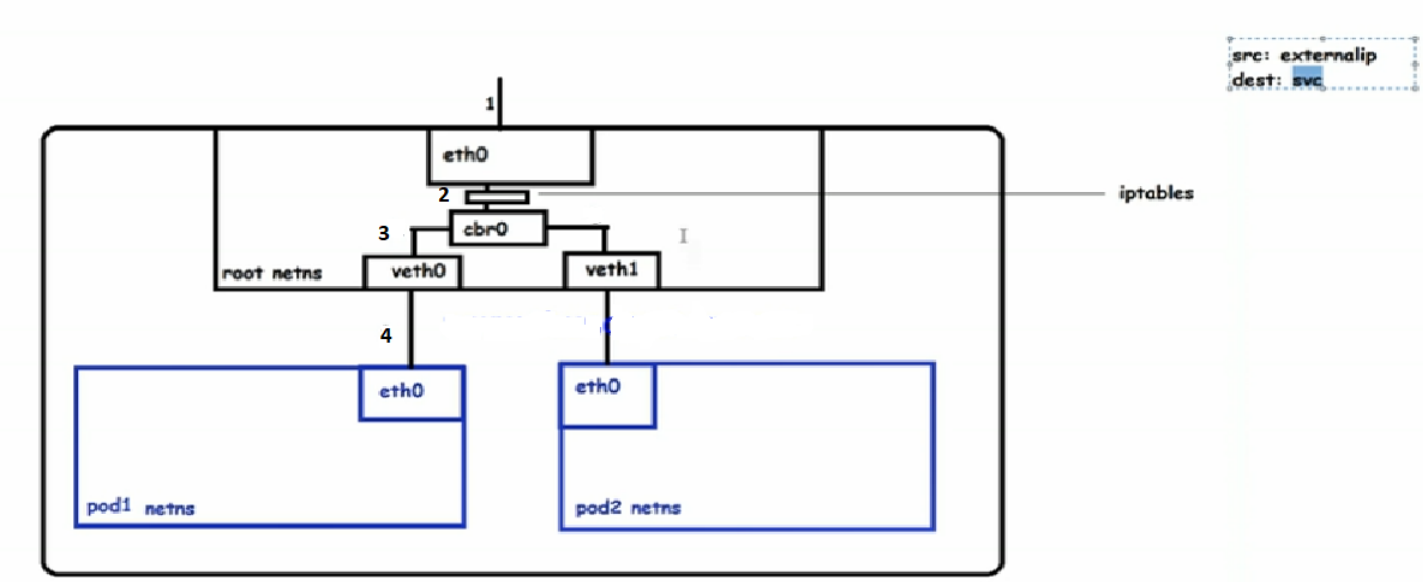
- **Pods on different nodes**, as shown in the below pic, pod1 was trying to communicate with pod4 in node-2.
- For that it will send a packet to eth0 and request will reach to ‘container bridge’ then the bridge will call out for destination ip. If it doesn’t get any response from its own node it will communicate to the external network.
- Basically all nodes are connected via a network. And all the node ethernet will consist of its container's CIDR range. So that it is easy to find out ip of nodes.
- In that pool the matched CIDR node will respond and the packet will reach the destination node. The node-bridge will take that packet to destination pod in this case pod4



- **Pod to Service IP Address**, Hear pod1 wants to communicate to service1
- As same as above concepts the packet is reached to eth0 → cb0→ here it will communicate to ‘iptables’ entries
- Because service is an iptable entry that's why service doesn’t fail. So here this entry will tell that svc1 means pod4
- Then the packet will reach to destination



- From external world to service, request comes to eth0 then it will check in iptables and as we know all service entries are in iptables and then the request is forwarded to that particular service



Health Checks of Pod

- When we run our application in k8s, it is kept alive using process health checks.
- However in most cases a simple process might not be enough
    - process has deadlocked and unable to serve requests (process check will return as application is health)
- To address this, k8s introduced health checks for application *liveness*
- **Liveness Probe:** This probe checks for application specific logic to verify that the application is still running
- Containers that fail liveness probes will be restarted.
- **Readiness probe** describes when a container is ready to serve user requests. Containers that fail readiness check are removed from service (load balancers)
- Refer Here for the liveness probe.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: gameoflife
spec:
  containers:
    - image: qualitythought/gameoflife:07112020
      name: gol
      livenessProbe:
```

```
      httpGet:
        path: /gameoflife
        port: 8080
      initialDelaySeconds: 5
      timeoutSeconds: 1
      periodSeconds: 10
      failureThreshold: 3
    resources:
      requests:
        cpu: "500m"
        memory: "128Mi"
      limits:
        cpu: "1000m"
        memory: "512Mi"
    ports:
      - containerPort: 8080
        name: golhttpport
        protocol: TCP
```

Topic:- Activities

Activity: 1:-

Lets create a pod with the below yaml file.

```
---
apiVersion: v1
kind: Pod
metadata:
  name: gol-pod
spec:
  containers:
    - name: my-gol-cont
      image: qualitythought/gameoflife:07112020
      ports:
        - containerPort: 8080
```

```
ubuntu@ip-172-31-20-190:~/pods$ kubectl apply -f hello-pod.yaml
pod/gol-pod created
ubuntu@ip-172-31-20-190:~/pods$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
gol-pod   1/1     Running   0          16s
ubuntu@ip-172-31-20-190:~/pods$
```

```
ubuntu@ip-172-31-20-190:~/pods$ kubectl get pods -o wide
NAME      READY   STATUS    RESTARTS   AGE     IP          NODE               NOMINATED NODE   READINESS GATES
gol-pod   1/1     Running   0          2m19s   10.44.0.1   ip-172-31-27-107   <none>           <none>
ubuntu@ip-172-31-20-190:~/pods$ kubectl describe pod gol-pod
Name:         gol-pod
Namespace:    default
Priority:     0
Node:         ip-172-31-27-107/172.31.27.107
Start Time:   Fri, 13 Nov 2020 02:56:06 +0000
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           10.44.0.1
IPs:
  IP:  10.44.0.1
Containers:
  my-gol-cont:
    Container ID:   docker://3e2c62abeff5a9858c6b06bad4ad6493d949492ad7f9d896b6e3a510365b8aa5
    Image:          qualitythought/gameoflife:07112020
    Image ID:       docker-pullable://qualitythought/gameoflife@sha256:d1e9605cd63903c77ab9cb44a08cb3f5c8da6ec88bbab
c2fde4fc0b70b419b7
    Port:           8080/TCP
```

```
ubuntu@ip-172-31-20-190:~/pods$ kubectl get pods gol-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"name":"gol-pod","namespace":"default"},"spec":
ontainers":[{"image":"qualitythought/gameoflife:07112020","name":"my-gol-cont","ports":[{"containerPort":8080}]}]}
  creationTimestamp: "2020-11-13T02:56:05Z"
  managedFields:
  - apiVersion: v1
    fieldsType: FieldsV1
    fieldsV1:
      f:metadata:
        f:annotations:
          .: {}
          f:kubectl.kubernetes.io/last-applied-configuration: {}
      f:spec:
        f:containers:
          k:{"name":"my-gol-cont"}:
            .: {}
            f:image: {}
            f:imagePullPolicy: {}
```

```
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: default-token-559v4
    secret:
      defaultMode: 420
      secretName: default-token-559v4
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2020-11-13T02:56:06Z"
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: "2020-11-13T02:56:15Z"
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2020-11-13T02:56:15Z"
    status: "True"
    type: ContainersReady
  - lastProbeTime: null
    lastTransitionTime: "2020-11-13T02:56:05Z"
    status: "True"
    type: PodScheduled
  containerStatuses:
  - containerID: docker://3e2c62abeff5a9858c6b06bad4ad6493d949492ad7f9d896b6e3a510365b8aa5
    image: qualitythought/gameoflife:07112020
```

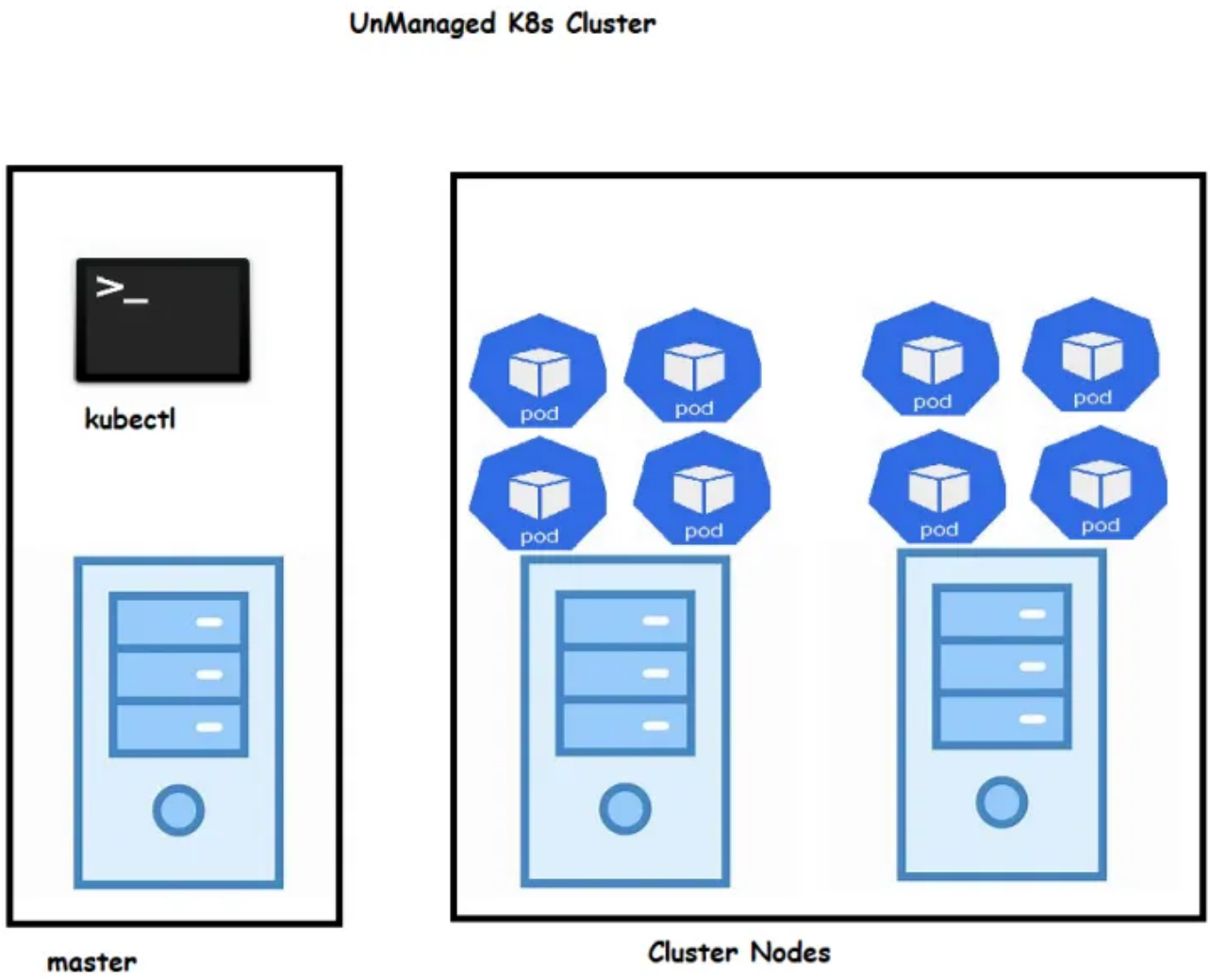
Activity:2:-

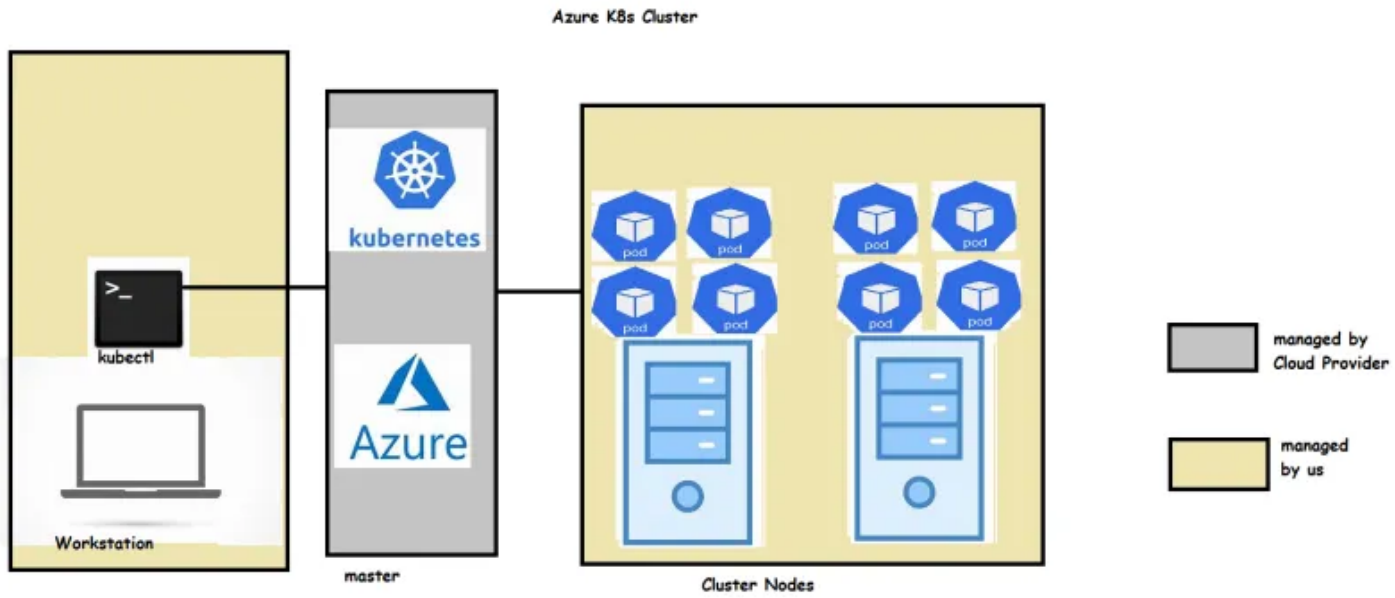
Topic: k8s yaml file

```
---
apiVersion: v1
kind: Pod
metadata:
  name: gol-pod
spec:
  containers:
    - name: my-gol-cont
      image: qualitythought/gameoflife:07112020
      ports:
        - containerPort: 8080
```

Topic:Azure Kubernetes Services (AKS)

- AKS is a managed k8s cluster in Azure.
- Azure manages master node
- Managed vs UnManage k8s cluster





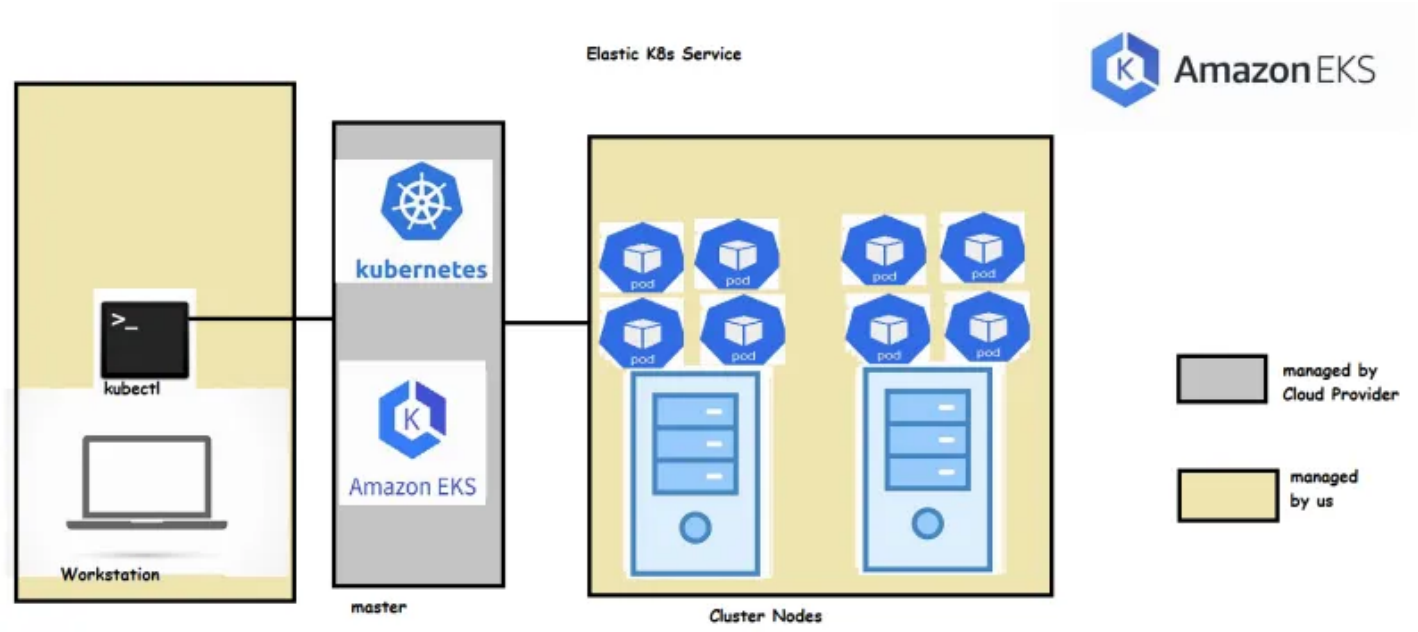
- Lets install AKS Cluster. AKS Cluster can be installed using
  - Portal Refer Here {https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough-portal}
  - CLI Refer Here {https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough }
  - Powershell Refer Here {https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough-powershell}
- For installing Azure CLI/Powershell Refer Here

Topic:Amazon Elastic K8s Service (EKS)

- Lets create a ubuntu ec2 instance
- Install aws cli in the ubuntu
- Refer Here for setting up eks cluster {https://docs.aws.amazon.com/eks/latest/userguide/getting-started.html}
- Install eks ctl by following docs Refer Here {https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html}
- After installing the eks, aws cli and kubectl commands, create k8s cluster by executing the following command

{

```
eksctl create cluster \
--name my-cluster \
--version 1.18 \
--region us-west-2 \
--nodegroup-name linux-nodes \
--nodes 1 \
--nodes-min 1 \
--nodes-max 2 \
--with-oidc
--ssh-access
--ssh-public-key dockerclassroom
--managed
}
```



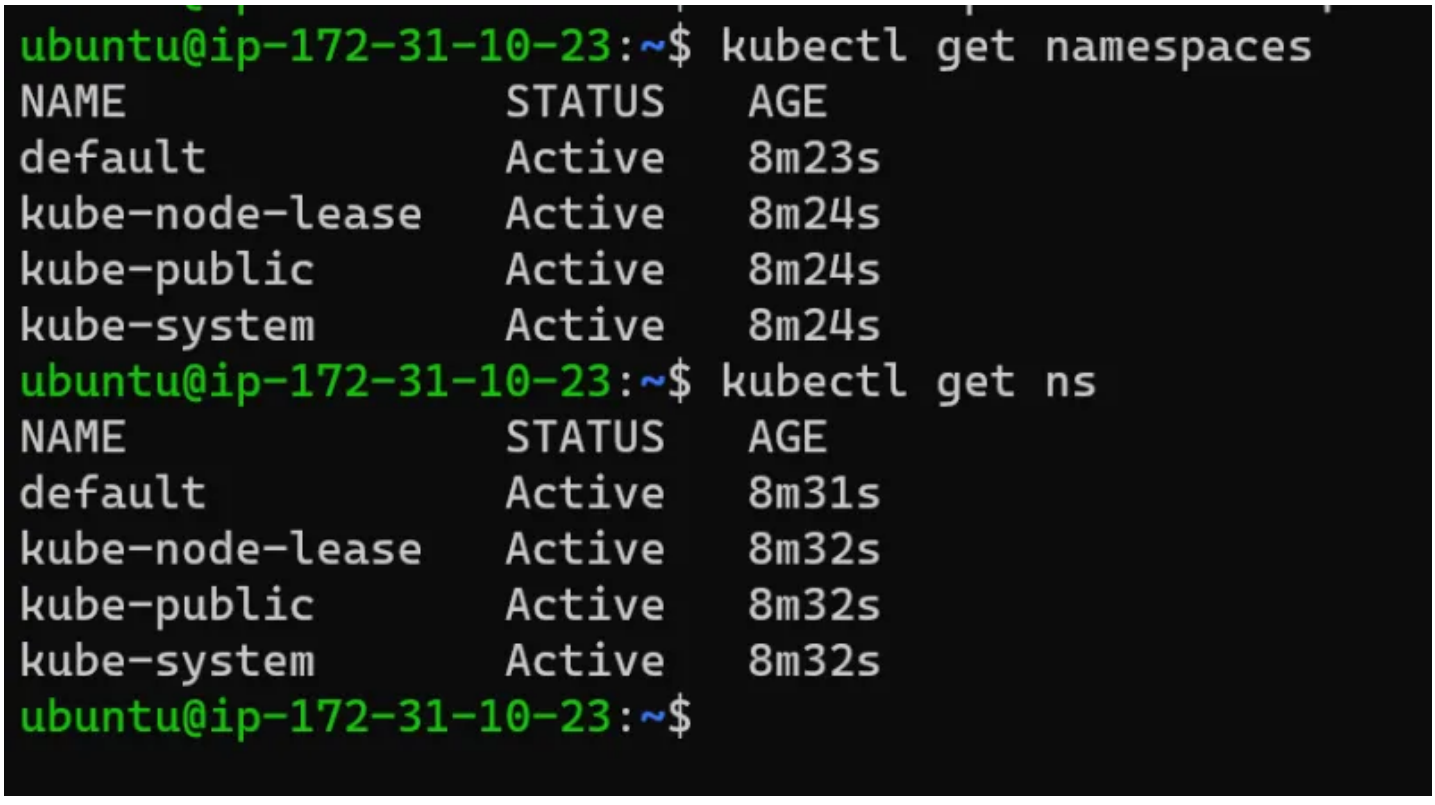
1) Create a vm and install aws cli on that
2) Create a user callead k8s in IAM and give him admin previlises
3) Configure aws cli with the provision of k8s keys
4) Install EKS by aws official documentation
5) Install Kubectl by using official documents
6) Configure managed nodes using docs

Topic: Namespaces

- K8s is one physical cluster with k8s master and cluster nodes & in this cluster if you want to create multiple virtual clusters (like one for dev, staging, qa..) we can use namespaces.
- Execute kubectl api-resources. The resource with value namespaced

- **true** => will be created per namespace {It means when we create two namespaces that will be different. }
- **false** => will be same/shared across namespaces {if it is false across name-spaces or virtual clusters you can't have different name spaces or nodes. Nodes are same across virtual clusters}

```
qtkhaja@Azure:~$ kubectl api-resources
NAME                              SHORTNAMES   APIGROUP                       NAMESPACED   KIND
bindings                                                                      true         Binding
componentstatuses                 cs                                          false        ComponentStatus
configmaps                        cm                                          true         ConfigMap
endpoints                         ep                                          true         Endpoints
events                            ev                                          true         Event
limitranges                       limits                                      true         LimitRange
namespaces                        ns                                          false        Namespace
nodes                             no                                          false        Node
persistentvolumeclaims            pvc                                         true         PersistentVolumeClaim
persistentvolumes                 pv                                          false        PersistentVolume
pods                              po                                          true         Pod
podtemplates                                                                  true         PodTemplate
replicationcontrollers            rc                                          true         ReplicationController
resourcequotas                    quota                                       true         ResourceQuota
secrets                                                                       true         Secret
```



```
ubuntu@ip-172-31-10-23:~$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   8m23s
kube-node-lease   Active   8m24s
kube-public       Active   8m24s
kube-system       Active   8m24s
ubuntu@ip-172-31-10-23:~$ kubectl get ns
NAME              STATUS   AGE
default           Active   8m31s
kube-node-lease   Active   8m32s
kube-public       Active   8m32s
kube-system       Active   8m32s
ubuntu@ip-172-31-10-23:~$
```

- Generally kubectl command will pull the values from namespace *default*. If you want k8s to fetch the information from other namespaces use –namespace flag

```
ubuntu@ip-172-31-10-23:~$ kubectl get pods
No resources found in default namespace.
ubuntu@ip-172-31-10-23:~$ kubectl get pods --namespace kube-system
NAME                       READY   STATUS    RESTARTS   AGE
aws-node-mndqp             1/1     Running   0          3m35s
coredns-559b5db75d-cq59l   1/1     Running   0          10m
coredns-559b5db75d-djrmv   1/1     Running   0          10m
kube-proxy-nckts           1/1     Running   0          3m35s
ubuntu@ip-172-31-10-23:~$
```

NOTE:- If we need to k8s will pull the values from some particular name-space, Then you can set something called as context (context is nothing but some values stored in kubectl config file ~/.kube/config)

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUN5RENDQWJDZ0F3SUJBZ0lCQURBTkJna3Foa2lHOXcwQkFRc0ZBREFWTVJNd0
VRWURWUVFERXdwcmRXSmwKY2O1bGRHVnpNQjRYRFRJeE1EUXhNakV3TWpVek5sb1hEVE14TURReE1ERXdNalV6Tmxvd0ZURVRNQkVHQTFVRQpBeE1LYTNWaVpYSnVaWFJsY3pD
QOFTSXdEUVlKS29aSWh2YO5BUUVCQlFBRGdnRVBBRENDQVFvQ2dnRUJBTOpYCnJpZW9nQlo2bWg4U2RmWVVyNkxIb2lta0xENm44eWVlSlVKcVBOWEUyOGJTb2lQeldiMjYxSE
hFL2RWbVVOdFIKVnNjWG5TQlZuVXM1UnpIYmlOWVZsNm9HY1pqczRjaOxJUEZQcTBWQzFjZVVkQO5TSXBrTUQOY3F4ZTJzQ1FsVAoOd1JGNFFmUOsvQ1RoQUhLNWZrVkdtWU9u
cG9nVG9sMGtjTU1ILzJrakFXbWpvTlJkSWsyMGNDUFV4bDVseG5XCmkzdm1hV1JPZUNUbFZob29mRlk3TkdNUOFMUW1wZ3RXWnMwd2xZdkpUNm1rZWdFZWpjbVYzdXJzV3BDQk
MOVlYKbDUySFdzOHZ5WEZDQjIxQVRRRXFOQzJrazU4M1JmK3RBUWZFS1NEZOppRkpINEFUdlR2WXNyYXNENWtZQXU3dQpTblZIVHZuZzJpc3A5SXUyTHMwQOF3RUFBYU1qTUNF
dORnWURWUjBQQVFILOJBUURBZOtrTUE4ROExVWRFdOVCCi93UUZNQU1CQWY4dORRWUpLb1pJaHZjTkFRUxCUUFEZ2dFQkFOckdISU9sNzljajdMb1hhcllWmRXSkk2ZHIKNW
VlcOY4aWk0VUdjVjdmY1pvQ1M3Vkp3RktveUNQZFF6VUZnbHhZckppRHhsVnVyWHhXMTZubXlNTFY2amViWgpwQThHSkg1L1dzVzdqcnM1S3h2OTl1cFRRUUxrQTZBaHNYTGYy
VTY4OFF6NERzam5rWFpjeGJGUmFLMXl3ekc5CnAxbCtTWUQxdGdsZk5iRVgwSm5Hb2tvcnVYbHBCZVFycOw5R3FnSOFUZllnSnhtdkVXbO5aQmdwSFBiYTFNbWcKSEgzSWU1KO
RlRG8OaW1RV2tneEpRY1NDVkVrQ1JUVkh1L2t5QmRTcDljUy9CdFRTWmZpQzV1RysvcGdTWTduaAp2encOMEROVklYamEveHlsTTlMeGRadDFHZFBpc1EzRWE2djVUVUFKRngz
Yk1lTkxGL1hsdkN3ZOJJUT0KLSOtLS1FTkQgQOVSVElGSUNBVEUtLSOtLQo=
    server: https://039758468CAD1FCDA01552014E982FF5.gr7.us-east-1.eks.amazonaws.com
  name: mahi-k8s.us-east-1.eksctl.io
contexts:
- context:
    cluster: mahi-k8s.us-east-1.eksctl.io
    user: k8s@mahi-k8s.us-east-1.eksctl.io
  name: k8s@mahi-k8s.us-east-1.eksctl.io
current-context: k8s@mahi-k8s.us-east-1.eksctl.io
kind: Config
preferences: {}
users:
- name: k8s@mahi-k8s.us-east-1.eksctl.io
  user:
    exec:
      apiVersion: client.authentication.k8s.io/v1alpha1
      args:
      - eks
      - get-token
      - --cluster-name
      - mahi-k8s
      - --region
      - us-east-1
```

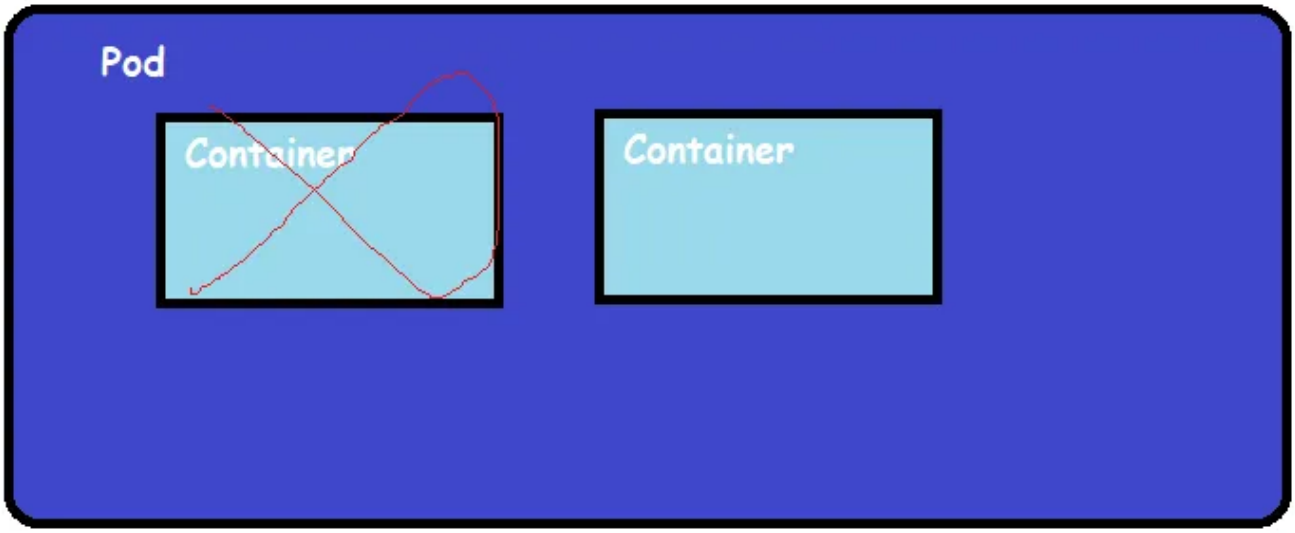
Upgrading

- Upgrading to published versions is easier in the cloud.
    - In AKS, we can upgrade the nodepools and k8s seperately
    - In the case of eksctl

```
ubuntu@ip-172-31-10-23:~$ eksctl upgrade --help
Upgrade resource(s)

Usage: eksctl upgrade [flags]

Commands:
  eksctl upgrade cluster                    Upgrade control plane to the next version
  eksctl upgrade nodegroup                  Upgrade nodegroup

Common flags:
  -C, --color string   toggle colorized logs (valid options: true, false, fabulous) (default "true")
  -h, --help           help for this command
  -v, --verbose int    set log level, use 0 to silence, 4 for debugging and 5 for debugging with AWS debug logging (d
efault 3)

Use 'eksctl upgrade [command] --help' for more information about a command.

ubuntu@ip-172-31-10-23:~$
```

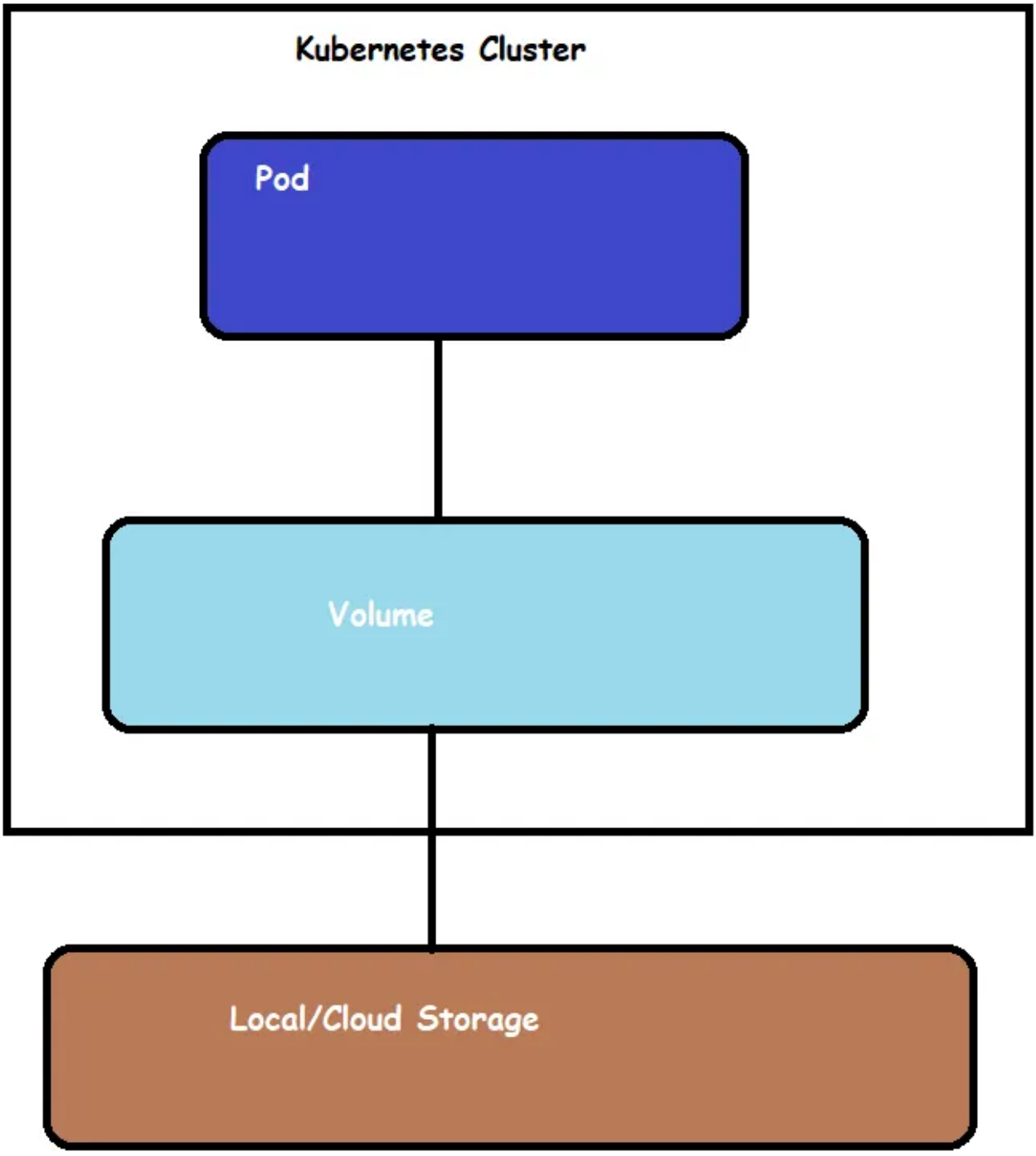
Topic: Storage in K8s

- Both containers & Pods are ephemeral (when we delete the Pods/Containers data generated is deleted)
- Docker Volume separates storage from the life cycle of Container, So even if the container is deleted, data will still be available.
- Since Docker containers are created in the Pod, and the problem of storage has to be solved at Pod level in k8s as Pod provides kernel namespace to docker container.
- Pod has a container. The container in the pod stores the data and let's assume it is crashes, Pod will try to restart the container and in this case data will be lost. The new container will start with empty disk space allocated.



- We cannot rely on containers themselves even for temporary storage of data.

- K8s Volume abstraction solves the problem

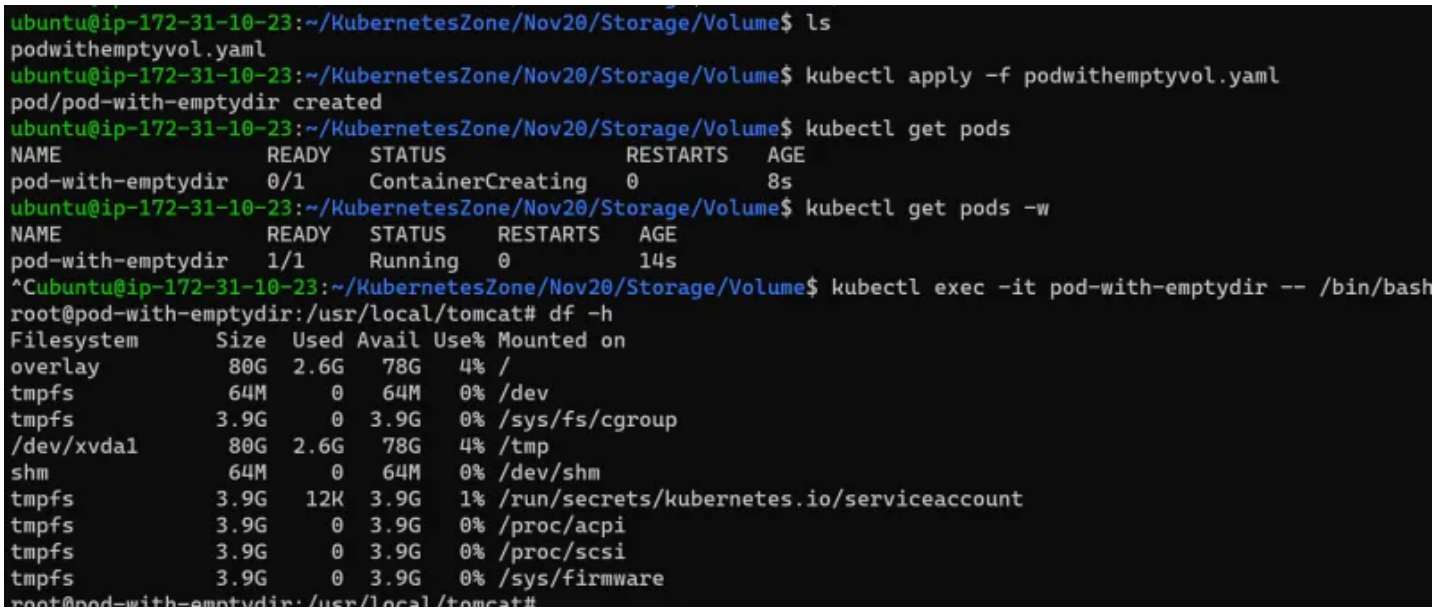


- Volume is exposed to the applications which eventually store data on any type of storage (node/cloud).
- Lifetime of k8s volume is the same as the Pod's lifetime. As long as Pod is alive the containers within POD will get the same volume across restarts or new container Creation.
- How to use K8s Volume

- Volume is defined in Pod Spec
- Apply the k8s pod manifest with empty volume

```
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-emptydir
spec:
  containers:
    - image: qualitythought/gameoflife:07112020
      name: gol-container
      volumeMounts:
        - mountPath: /tmp
          name: tmp-volume
  volumes:
    - name: tmp-volume
      emptyDir: {}
```

```
qtkhaja@Azure:~$ kubectl apply -f podwithemptyvol.yaml

pod/pod-with-emptydir created
qtkhaja@Azure:~$
qtkhaja@Azure:~$ kubectl get po
NAME                READY   STATUS              RESTARTS   AGE
pod-with-emptydir   0/1     ContainerCreating   0          12s
qtkhaja@Azure:~$ kubectl get po -w
NAME                READY   STATUS    RESTARTS   AGE
pod-with-emptydir   1/1     Running   0          17s
^Cqtkhaja@Azure:~$ kubectl exec -it pod-with-emptydir -- /bin/bash
root@pod-with-emptydir:/usr/local/tomcat# df -h
Filesystem      Size  Used Avail Use% Mounted on
overlay         124G   21G  104G  17% /
tmpfs            64M     0   64M   0% /dev
tmpfs           3.5G     0  3.5G   0% /sys/fs/cgroup
/dev/sda1       124G   21G  104G  17% /tmp
shm              64M     0   64M   0% /dev/shm
tmpfs           3.5G   12K  3.5G   1% /run/secrets/kubernetes.io/serviceaccount
tmpfs           3.5G     0  3.5G   0% /proc/acpi
tmpfs           3.5G     0  3.5G   0% /proc/scsi
tmpfs           3.5G     0  3.5G   0% /sys/firmware
root@pod-with-emptydir:/usr/local/tomcat#
```



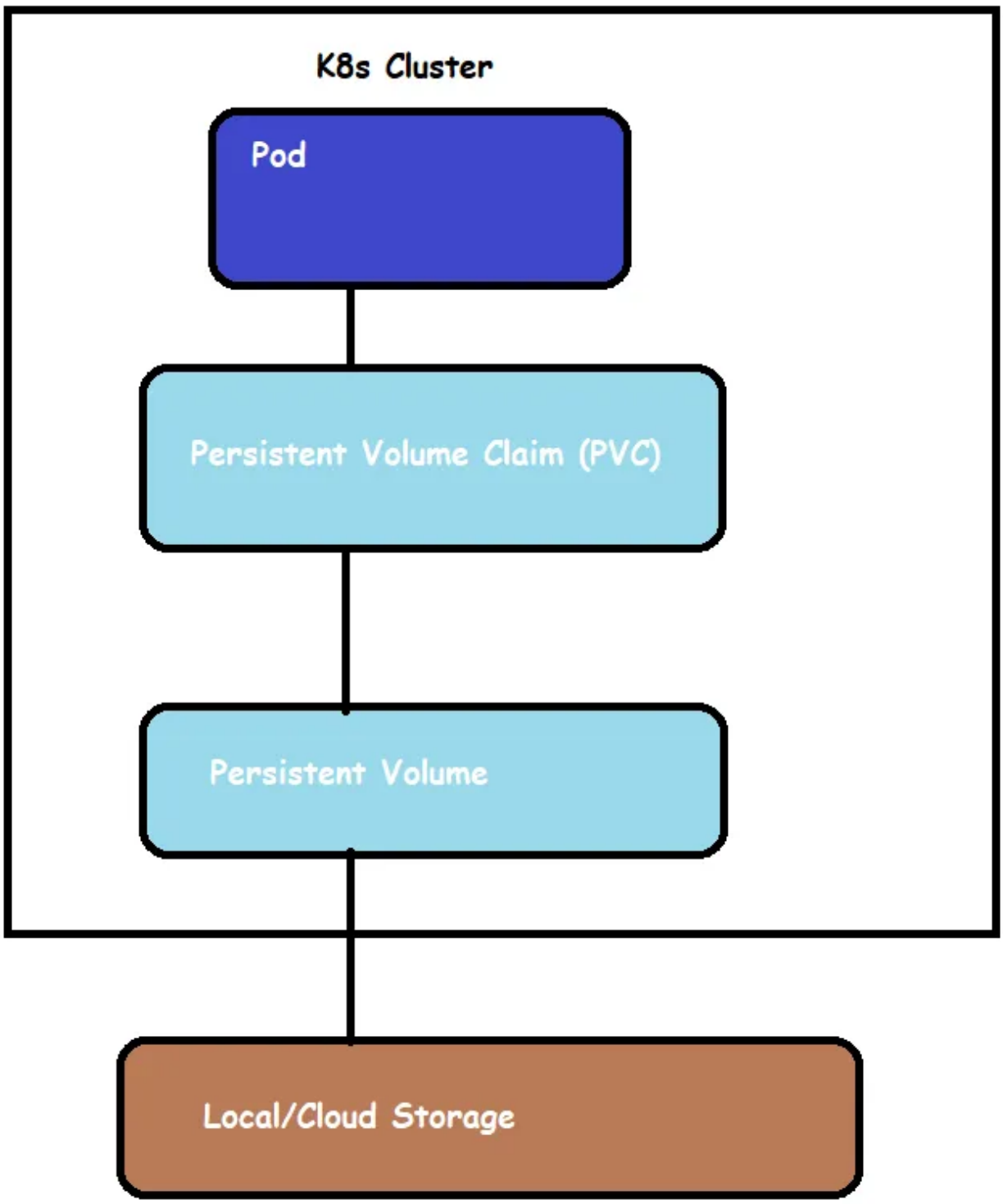
- Refer Here for official docs {https://kubernetes.io/docs/concepts/storage/volumes/}
- Lets use one more example of hostPath (Type of volume) Refer Here

```
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-hostpath
spec:
  containers:
    - image: qualitythought/gameoflife:07112020
      name: gol-container
      volumeMounts:
        - mountPath: /tmp
          name: tmp-volume-host
  volumes:
    - name: tmp-volume-host
      hostPath:
        path: /tmp
        type: Directory
```

- Volumes such as emptyDir and hostPath get deleted when the pod using them is deleted or the pod is restarted. So we need to find a solution Where the data/storage lifetime is different than Pod's lifetime
- To Solve this Problem, K8s has Persistent Storage in the form of Persistent Volume (PV).

Topic: Persistent Volumes & Persistent Volume Claims

- A PV is a K8s object that represents a block of storage in the cluster. This can be provisioned before hand by cluster admins or be dynamically provisioned.
- In Order to use PV, a Persistent Volume Claim (PVC) needs to be created. A PVC is a request for storage by user or by Pod. This request can be of specific size or specific access mode.



- Each PV belongs to a certain storage class. A Storage Class is a K8s Object that provides a way for administrators to describe different types of Storage.
- Volume mode has two Values *Block*, *FileSystem*
- Access Modes:
    - ReadWriteOnce (RWO): Mounted as read-write by a single node
    - ReadOnlyMany (ROX): Mounted as read-only by many nodes
    - ReadWriteMany (RWX): Mounted as Read-write by many nodes
- persistentVolume Reclaim Policy
    - **Retain** (Even if u delete your persistent volume the data will still be there)
    - **Recycle** (when we delete your persistent volume the store will be there but the data in the storage will be cleaned)
    - **Delete** (when ever we delete the persistence volume the storage/volume and data in that volume will be deleted)

PVC in Azure

- Dynamic Provision Azure Disk Refer Here Azure Files Refer Here
- Static Provision Azure Disk Refer Here Azure Files Refer Here
- NFS in Azure Refer Here
- Azure Netapp Refer Here
- AWS Refer Here
- Refer Here for AWS Storage Classes Documentation

Storage Classes in Azure (In azure we have 4 different classes)

```
qtkhaja@Azure:~$ kubectl get sc
NAME                PROVISIONER                RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
azurefile           kubernetes.io/azure-file   Delete          Immediate           true                   104m
azurefile-premium   kubernetes.io/azure-file   Delete          Immediate           true                   104m
default (default)   kubernetes.io/azure-disk   Delete          Immediate           true                   104m
managed-premium     kubernetes.io/azure-disk   Delete          Immediate           true                   104m
```

PVC in AWS

Storage Class in AWS (we have 1 storage class in aws)

```
ubuntu@ip-172-31-10-23:~$ kubectl get sc
NAME            PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
gp2 (default)   kubernetes.io/aws-ebs   Delete          WaitForFirstConsumer   false                  100m
ubuntu@ip-172-31-10-23:~$
```

Activity:

- Now let's create a 'my-sql' container, my-sql is a database so it require some kind of storage, so we will create a pod in that we will create a my-sql,
- Lets understand how to use mysql container in a Pod with Persistent Volume (in AWS and Azure)
- Refer Here to mysql Dockerfile which has volume at /var/lib/mysql

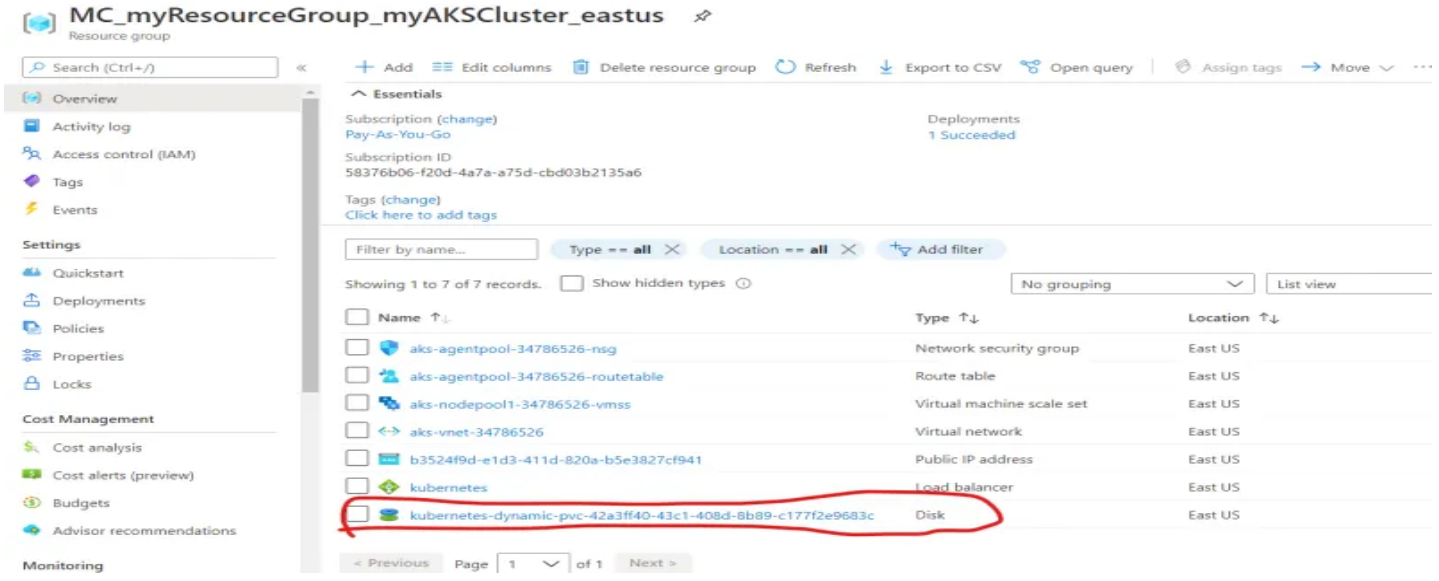
In azure:

- Lets use existing storage classes in Azure to create mysql-pod Refer Here

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-azure-pvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: managed-premium
  resources:
    requests:
      storage: 1Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: mysql-azure
spec:
  containers:
    - name: mysql-cont
      image: mysql:8
      volumeMounts:
        - mountPath: /var/lib/mysql
```

```
      name: mysql-volume
    env:
      - name: MYSQL_ROOT_PASSWORD
        value: 'rootroot'
      - name: MYSQL_DATABASE
        value: 'openmrs'
      - name: MYSQL_USER
        value: qtdevops
      - name: MYSQL_PASSWORD
        value: qtdevops
    ports:
      - name: mysqlport
        containerPort: 3306
        protocol: TCP
volumes:
  - name: mysql-volume
    persistentVolumeClaim:
      claimName: mysql-azure-pvc
```

```
qtkhaja@Azure:~$ kubectl apply -f mysql-azure.yaml
persistentvolumeclaim/mysql-azure-pvc unchanged
pod/mysql-azure created
qtkhaja@Azure:~$ kubectl get pods -w
NAME                READY   STATUS              RESTARTS   AGE
mysql-azure         0/1     ContainerCreating   0          8s
pod-with-emptydir   1/1     Running             0          69m
^Cqtkhaja@Azure:~$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                   STORAGECLASS
    REASON   AGE
pvc-42a3ff40-43c1-408d-8b89-c177f2e9683c   1Gi        RWO            Delete           Bound    default/mysql-azure-pvc   managed-premi
m           6m33s
qtkhaja@Azure:~$
```



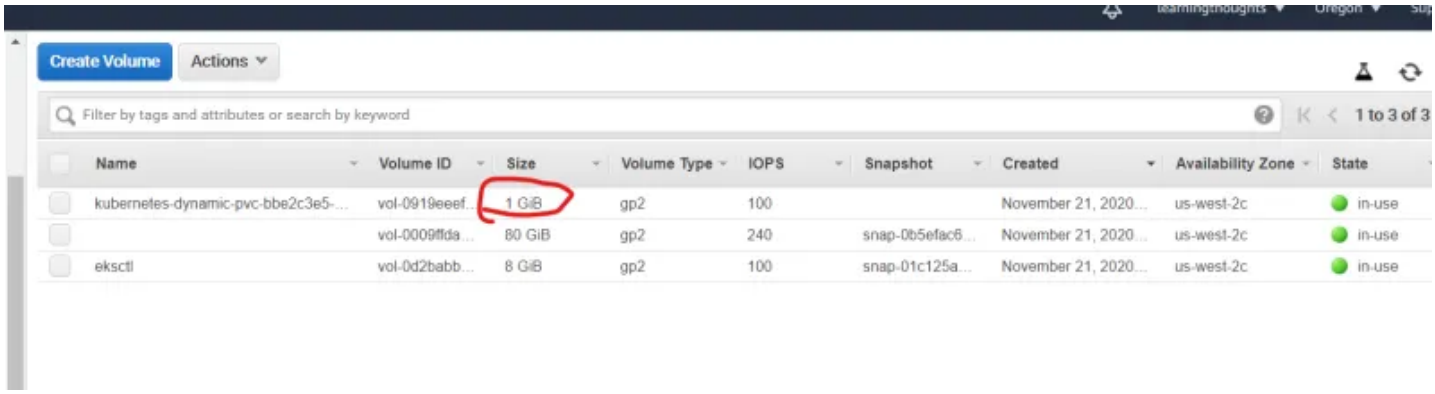
In AWS:

- for the manifest Refer Here

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pvc-aws
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: gp2
  resources:
    requests:
      storage: 1Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: mysql-pvc-aws
spec:
  containers:
    - name: mysql-cont
      image: mysql:8
      volumeMounts:
        - mountPath: /var/lib/mysql
          name: mysql-volume
      env:
        - name: MYSQL_ROOT_PASSWORD
          value: 'rootroot'
        - name: MYSQL_DATABASE
          value: 'openmrs'
        - name: MYSQL_USER
          value: qtdevops
        - name: MYSQL_PASSWORD
          value: qtdevops
      ports:
```

```
      - name: mysqlport
        containerPort: 3306
        protocol: TCP
  volumes:
    - name: mysql-volume
      persistentVolumeClaim:
        claimName: mysql-pvc-aws
```

```
ubuntu@ip-172-31-10-23:~/KubernetesZone/Nov20/Storage/PersistentVolume$ kubectl apply -f mysql-aws.yaml
persistentvolumeclaim/mysql-aws-pvc created
pod/mysql-aws created
ubuntu@ip-172-31-10-23:~/KubernetesZone/Nov20/Storage/PersistentVolume$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM
  STORAGECLASS   REASON   AGE
pvc-bbe2c3e5-e93a-4287-b1be-48c6ef0c8add   1Gi        RWO            Delete           Bound    default/mysql-aws-pvc
  gp2                    1s
ubuntu@ip-172-31-10-23:~/KubernetesZone/Nov20/Storage/PersistentVolume$ kubectl get pods
NAME                READY   STATUS              RESTARTS   AGE
mysql-aws           0/1     ContainerCreating   0          14s
pod-with-emptydir   1/1     Running             0          70m
ubuntu@ip-172-31-10-23:~/KubernetesZone/Nov20/Storage/PersistentVolume$
```

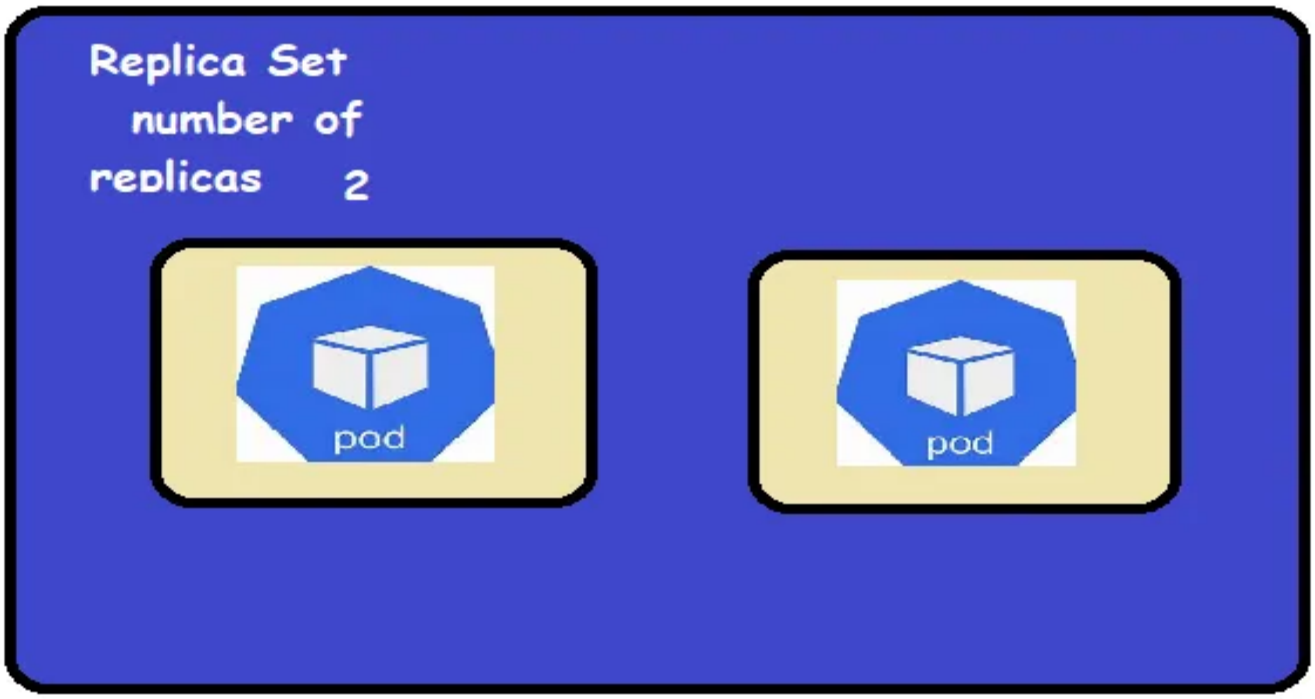


K8s Volume Snapshots & Clones

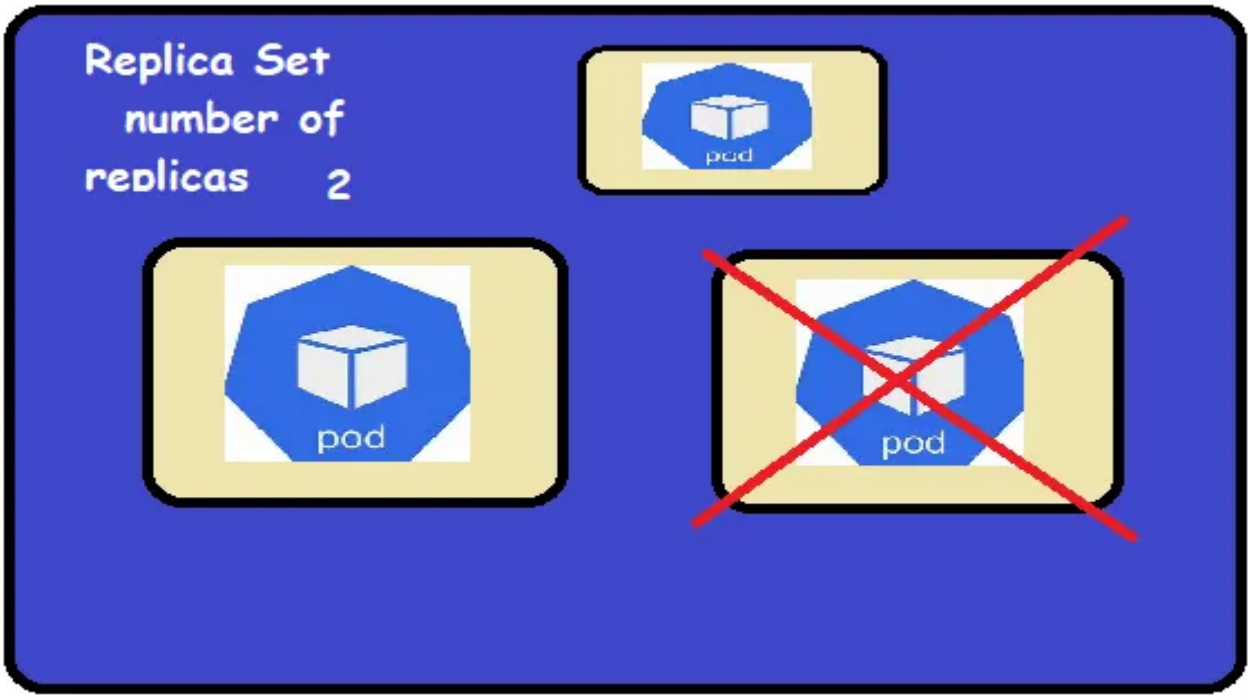
- K8s Volumesnapshot represents the snapshot of volume on the storage system.
- Source can be specified as PVC Refer Here
- CSI Volume Cloning features add support for existing PVCs to clone the volume. Refer Here

Topic: Replica set (K8s controller type-1)

- Replica Set will ensure that the number of replicas specified in the spec is maintained by k8s cluster.



- K8s will try to maintain the desired state which is the number of replicas. If some pod gets deleted/crashed etc, k8s will create one more pod to maintain the desired state.



- Pods might be allocated on different cluster nodes.
- So lets create a replica set with 2 replicas for the game of life. Refer Here for the changeset

---
apiVersion: apps/v1

```
kind: ReplicaSet
metadata:
  name: gol-rs
spec:
  minReadySeconds: 5
  replicas: 2
  template:
    metadata:
      labels:
        app: gol
        version: "07112020"
    spec:
      containers:
        - image: qualitythought/gameoflife:07112020
          name: gol
          ports:
            - containerPort: 8080
```

- Now lets add selector for the labels


```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: replocatio-set
spec:
  replicas: 2
  minReadySeconds: 10
  template:
    metadata:
      labels:
        app: gol
        version: "1.1"
```

```
  spec:
    containers:
      - image: qualitythought/gameoflife:07112020
        name: gol
        ports:
          - containerPort: 8080
selector:
  matchLabels:
    app: gol
    version: "1.1"
```

- apply the replica set and explore
- kubectl get rs - to view the replica-set
- kubectl describe rs replication-set - to describe the replication set

```
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get rs
NAME                                DESIRED   CURRENT   READY   AGE
hello-world-rest-api-55c678fc85     1         1         0       6d8h
replocatio-set                      2         2         2       3m39s
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl describe rs replocatio-set
Name:         replocatio-set
Namespace:    default
Selector:     app=gol,version=1.1
Labels:       <none>
Annotations:  <none>
Replicas:     2 current / 2 desired
Pods Status:  2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=gol
           version=1.1
  Containers:
   gol:
    Image:        qualitythought/gameoflife:07112020
    Port:         8080/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age    From                    Message
  ----    ------            ----   ----                    -------
  Normal  SuccessfulCreate  3m59s  replicaset-controller  Created pod: replocatio-set-d2xv6
  Normal  SuccessfulCreate  3m59s  replicaset-controller  Created pod: replocatio-set-ffpfc
root@ip-172-31-17-61:/mahi/k8s/rs#
```

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get rs -o wide
NAME     DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES                              SELECTOR
gol-rs   2         2         2       80s   gol          qualitythought/gameoflife:07112020   app=gol,version=0711202
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE    IP          NODE               NOMINATED NODE   READINESS GATES
gol-rs-4px55   1/1     Running   0          108s   10.47.0.1   ip-172-31-17-40    <none>           <none>
gol-rs-mmx5l   1/1     Running   0          108s   10.44.0.1   ip-172-31-27-107   <none>           <none>
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$
```

Now lets shutdown one node and see the result

```
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get rs -w
NAME                          DESIRED   CURRENT   READY   AGE
hello-world-rest-api-55c678fc85   1         1         0       6d9h
replocatio-set                2         2         1       22m
replocatio-set                2         1         1       24m
replocatio-set                2         2         1       24m
replocatio-set                2         2         2       24m
replocatio-set                2         2         2       24m
^Croot@ip-172-31-17-61:/mahi/k8s/rs# kubectl get pods -o wide
NAME                                READY   STATUS             RESTARTS   AGE   IP          NODE              NOMINATED NODE   READI
NESS GATES
hello-world-rest-api-55c678fc85-ndd85   0/1     ImagePullBackOff   0          37m   10.44.0.1   ip-172-31-12-116  <none>           <none
>
replocatio-set-d2xv6                1/1     Terminating        0          26m   10.47.0.1   ip-172-31-15-137  <none>           <none
>
replocatio-set-ffpfc                1/1     Running            0          26m   10.44.0.2   ip-172-31-12-116  <none>           <none
>
replocatio-set-z5cdk                1/1     Running            0          90s   10.44.0.3   ip-172-31-12-116  <none>           <none
>
```

We can even scale the pods in replica set with out modifying yml file
kubectl scale replica set replication-set --replicas=5
NOTE:- it is not a recommended way

```
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl scale replicaset replocatio-set --replicas=5
replicaset.apps/replocatio-set scaled
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get ods
error: the server doesn't have a resource type "ods"
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get rs
NAME                          DESIRED   CURRENT   READY   AGE
hello-world-rest-api-55c678fc85   1         1         0       6d9h
replocatio-set                5         5         5       30m
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
hello-world-rest-api-55c678fc85-ndd85   0/1     ImagePullBackOff   0          42m
replocatio-set-d2xv6                1/1     Terminating        0          30m
replocatio-set-ffpfc                1/1     Running            0          30m
replocatio-set-gmdlz                1/1     Running            0          18s
replocatio-set-pdr6q                1/1     Running            0          18s
replocatio-set-sjwrg                1/1     Running            0          18s
replocatio-set-z5cdk                1/1     Running            0          6m10s
root@ip-172-31-17-61:/mahi/k8s/rs# kubectl get pods -o wide
NAME                                READY   STATUS             RESTARTS   AGE     IP          NODE              NOMINATED NODE
DINESS GATES
hello-world-rest-api-55c678fc85-ndd85   0/1     ImagePullBackOff   0          42m     10.44.0.1   ip-172-31-12-116  <none>
ne>
replocatio-set-d2xv6                1/1     Terminating        0          30m     10.47.0.1   ip-172-31-15-137  <none>
ne>
replocatio-set-ffpfc                1/1     Running            0          30m     10.44.0.2   ip-172-31-12-116  <none>
ne>
replocatio-set-gmdlz                1/1     Running            0          26s     10.44.0.4   ip-172-31-12-116  <none>
ne>
replocatio-set-pdr6q                1/1     Running            0          26s     10.44.0.5   ip-172-31-12-116  <none>
ne>
replocatio-set-sjwrg                1/1     Running            0          26s     10.44.0.6   ip-172-31-12-116  <none>
ne>
replocatio-set-z5cdk                1/1     Running            0          6m18s   10.44.0.3   ip-172-31-12-116  <none>
ne>
root@ip-172-31-17-61:/mahi/k8s/rs#
```

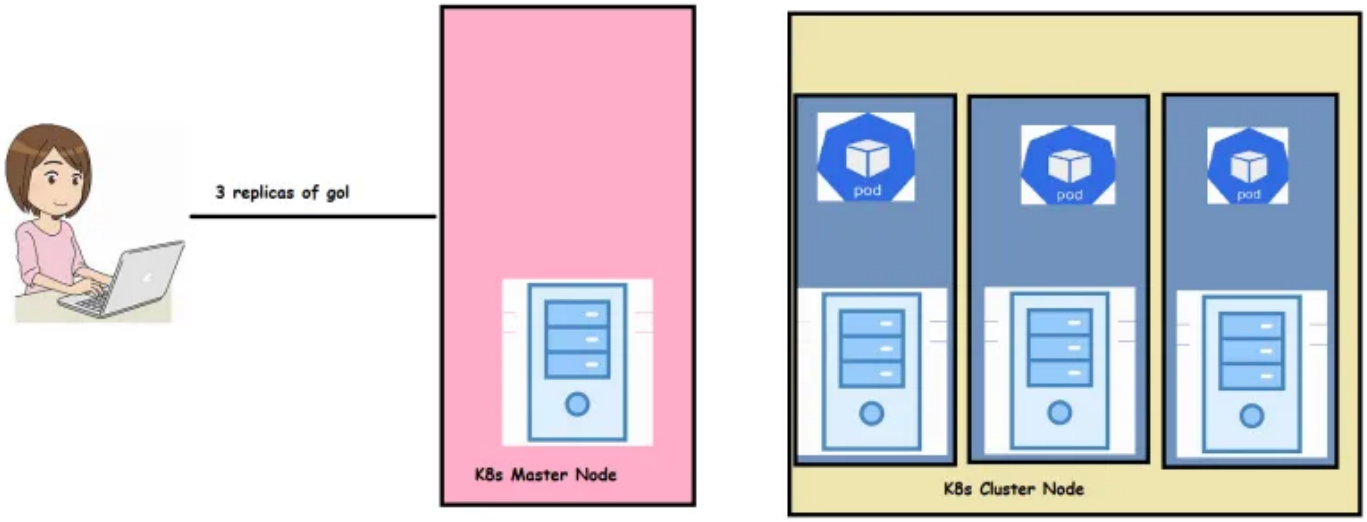
- when ever we need to scale the replicas is in two ways i) chaining the yaml file ii) using auto scaling imperative commands

EX:- kubectl autoscale rs replocatio-set --min=2 --max=8 --cpu-percent=80

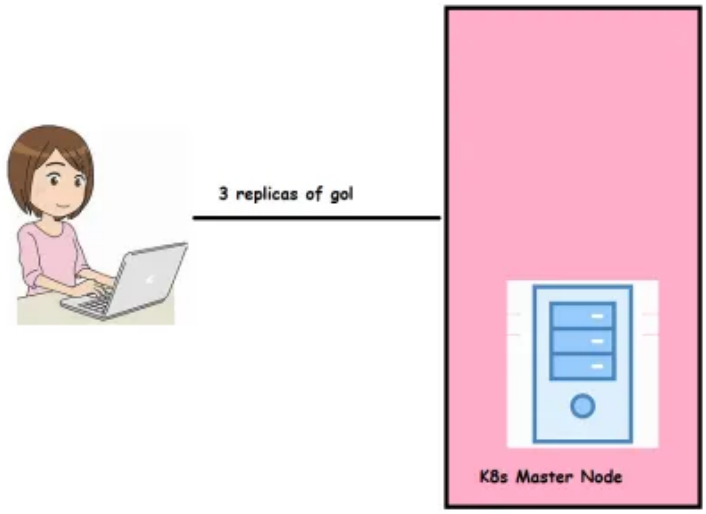
```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl autoscale rs gol-rs --min=2 --max=5 --cpu-percent=
80
horizontalpodautoscaler.autoscaling/gol-rs autoscaled
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get rs
NAME     DESIRED   CURRENT   READY   AGE
gol-rs   2         2         2       11m
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get rs
NAME     DESIRED   CURRENT   READY   AGE
gol-rs   2         2         2       11m
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get hpa
NAME     REFERENCE           TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
gol-rs   ReplicaSet/gol-rs   <unknown>/80%   2         5         2          27s
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$
```

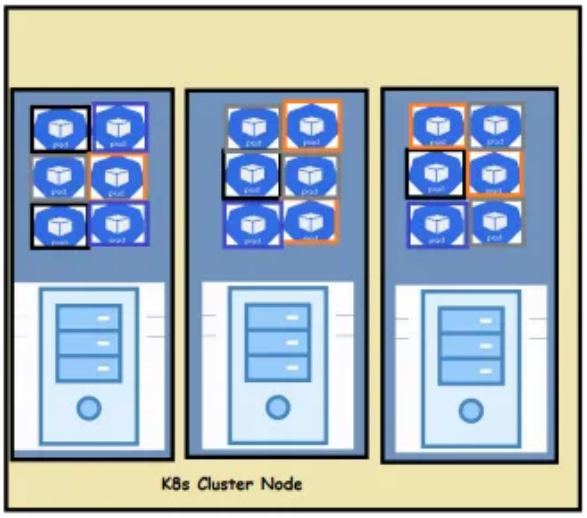
Labels:-

- Basic workflow of creating replicas



- Giving any pod ip address to access the application is not good caz any pod can be killed any time and any pod can create at any time.
- As k8s is used and application microservices run in the cluster, the number of pods running of nodes might grow in size and complexity.
- As shown in the below pic orange circled pods are belongs to one service for ex web server, Black circled pods are belongs to another service for ex app server..etc.,





- K8s has added foundational concepts of Labels and annotations to organization, mark and cross-index all of your resources to represent the groups.
- *Labels* are key/value pairs that can be attached to k8s objects such as pods and ReplicaSets. Labels are used by k8s cluster to group pods/other objects
- *_Annotations* provide storage mechanism similar to labels which information that can be used by tools & libraries
- Labels provide identifying metadata


```
---
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: gol-rs-label
spec:
  minReadySeconds: 10
  replicas: 5
  selector:
```

```
  matchLabels:
    app: gol
template:
  metadata:
    labels:
      app: gol
      version: "07112020"
  spec:
    containers:
      - image: qualitythought/gameoflife:07112020
        name: gol
        ports:
          - containerPort: 8080
            protocol: TCP
```

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl apply -f gol-rs-withlabels.yaml
replicaset.apps/gol-rs-label created
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get rs
NAME           DESIRED   CURRENT   READY   AGE
gol-rs-label   5         5         5       6s
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get pods
NAME                 READY   STATUS    RESTARTS   AGE
gol-rs-label-2hrnf   1/1     Running   0          13s
gol-rs-label-9gkjh   1/1     Running   0          13s
gol-rs-label-9l9pg   1/1     Running   0          13s
gol-rs-label-ln7p7   1/1     Running   0          13s
gol-rs-label-spxqj   1/1     Running   0          13s
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get pods -owide
NAME                 READY   STATUS    RESTARTS   AGE   IP          NODE               NOMINATED NODE   READINESS GA
ES
gol-rs-label-2hrnf   1/1     Running   0          20s   10.47.0.1   ip-172-31-17-40    <none>           <none>
gol-rs-label-9gkjh   1/1     Running   0          20s   10.44.0.1   ip-172-31-27-107   <none>           <none>
gol-rs-label-9l9pg   1/1     Running   0          20s   10.44.0.2   ip-172-31-27-107   <none>           <none>
gol-rs-label-ln7p7   1/1     Running   0          20s   10.47.0.2   ip-172-31-17-40    <none>           <none>
gol-rs-label-spxqj   1/1     Running   0          20s   10.47.0.3   ip-172-31-17-40    <none>           <none>
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$
```

```
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$ kubectl get pods --show-labels
NAME                 READY   STATUS    RESTARTS   AGE   LABELS
gol-rs-label-2hrnf   1/1     Running   0          62s   app=gol,version=07112020
gol-rs-label-9gkjh   1/1     Running   0          62s   app=gol,version=07112020
gol-rs-label-9l9pg   1/1     Running   0          62s   app=gol,version=07112020
gol-rs-label-ln7p7   1/1     Running   0          62s   app=gol,version=07112020
gol-rs-label-spxqj   1/1     Running   0          62s   app=gol,version=07112020
ubuntu@ip-172-31-20-190:~/KubernetesZone/Nov20/replicaset$
```

- Pods can be selected by writing selectors

Some selector examples:

```
---
selector:
  matchLabels:
    app: jenkins
```

```
---
selector:
  matchExpressions:
    - {key: app, operator: In, values: ['jenkins', 'gol']}
---
selector:
  matchExpressions:
    - {key: ver, operator: NotIn, values: ['1.0', '2.0']}
```

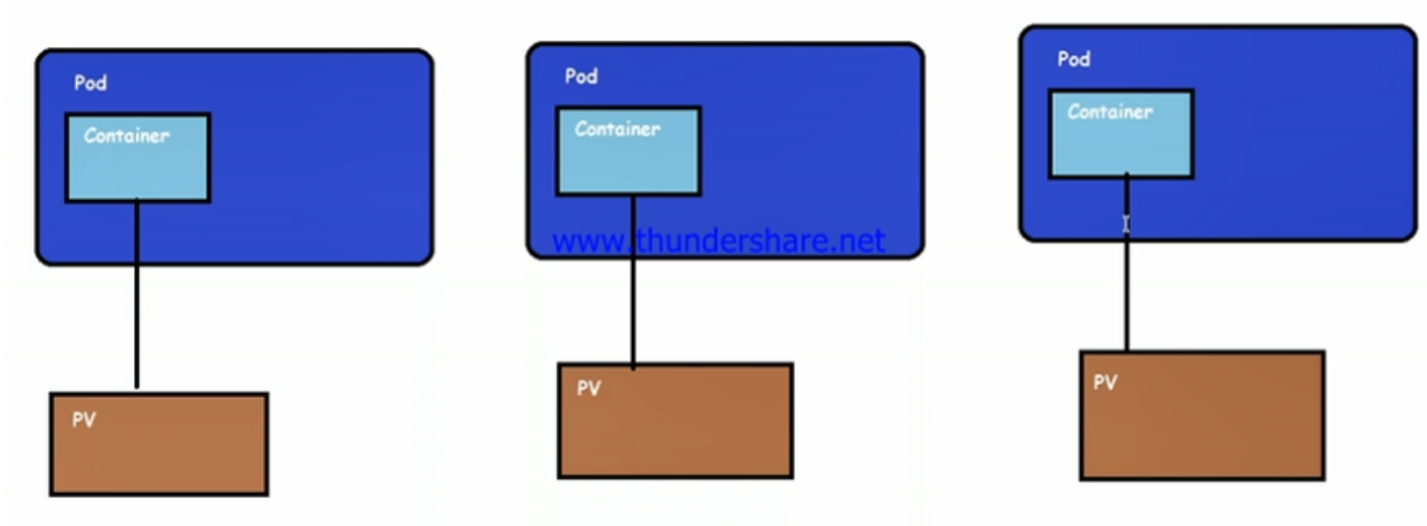
- If you want to all the api-resources in k8s cluster

Service Discovery

- We are able to run our applications in pods
- We can dynamically scale pods by using autoscale command
- How do i access my application which is running in pods
- Service-discovery helps solving the problem of finding which processes are listening at which address for each service.
- We need to expose our services to the external world. As of now w.r.t networking we know each pod is having its own ip address.

Topic: StatefulSet (K8s Controller type-2)

- Generally we map same storage disk/location to a application running in different pos/containers
- But if we each pod/container running for same application can maintain separate pvc/storage as shown in the below fig then we need stateful-set
- Stateful sets are used to stateful replicas



Stateful sets are used to stateful replicas Refer Here

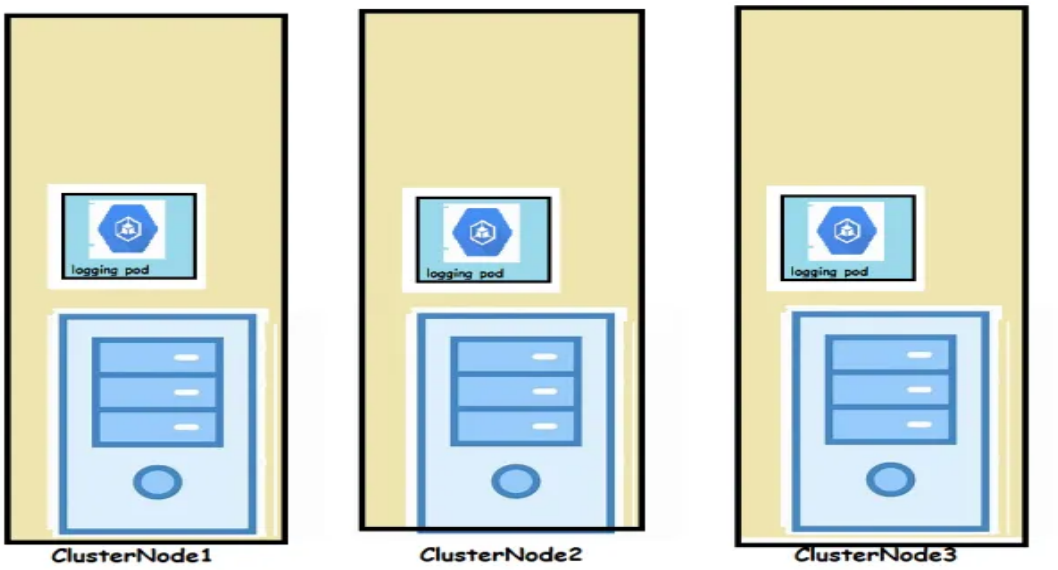
```
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: stateful-demo
spec:
  replicas: 3
  serviceName: jenkins-svc
  selector:
    matchLabels:
      app: jenkins
  template:
    metadata:
      labels:
        app: jenkins
        version: "2.60.3"
    spec:
      containers:
        - image: jenkins:2.60.3
          name: jenkins
          ports:
            - containerPort: 8080
              protocol: TCP
```

```
qtkhaja@Azure:~$ kubectl apply -f Statefulset.yaml
error: error validating "Statefulset.yaml": error validating data: ValidationError(StatefulSet.spec): missing required field "serviceNa
me" in io.k8s.api.apps.v1.StatefulSetSpec; if you choose to ignore these errors, turn validation off with --validate=false
qtkhaja@Azure:~$ kubectl apply -f Statefulset.yaml
statefulset.apps/stateful-demo created
qtkhaja@Azure:~$ kubectl api-resources | grep State*
qtkhaja@Azure:~$ kubectl api-resources | grep State
statefulsets                    sts         apps                        true        StatefulSet
healthstates                                azmon.container.insights    true        HealthState
qtkhaja@Azure:~$ kubectl get sts
NAME           READY   AGE
stateful-demo   1/3     31s
qtkhaja@Azure:~$ kubectl get sts -w
NAME           READY   AGE
stateful-demo   3/3     36s
```

```
root@ip-172-31-27-144:/mahi/k8s/statefull# kubectl get sts
NAME               READY   AGE
statefulset-demo   3/3     3m3s
root@ip-172-31-27-144:/mahi/k8s/statefull# kubectl get pods
NAME                 READY   STATUS    RESTARTS   AGE
statefulset-demo-0   1/1     Running   0          2m31s
statefulset-demo-1   1/1     Running   0          2m29s
statefulset-demo-2   1/1     Running   0          2m27s
root@ip-172-31-27-144:/mahi/k8s/statefull# kubectl get pods -o wide
NAME                 READY   STATUS    RESTARTS   AGE     IP               NODE                                NOMIN
ATED NODE   READINESS GATES
statefulset-demo-0   1/1     Running   0          2m45s   192.168.49.114   ip-192-168-34-80.ec2.internal   <none
>            <none>
statefulset-demo-1   1/1     Running   0          2m43s   192.168.21.0     ip-192-168-8-224.ec2.internal   <none
>            <none>
statefulset-demo-2   1/1     Running   0          2m41s   192.168.42.229   ip-192-168-34-80.ec2.internal   <none
>            <none>
root@ip-172-31-27-144:/mahi/k8s/statefull#
```

Topic: DaemonSet (K8s Controller type-3)

- Daemonsets are used to manage creation of a Particular Pod on all or selected nodes in k8s cluster
- Use Cases:
    - Logging agent
    - Monitoring agent



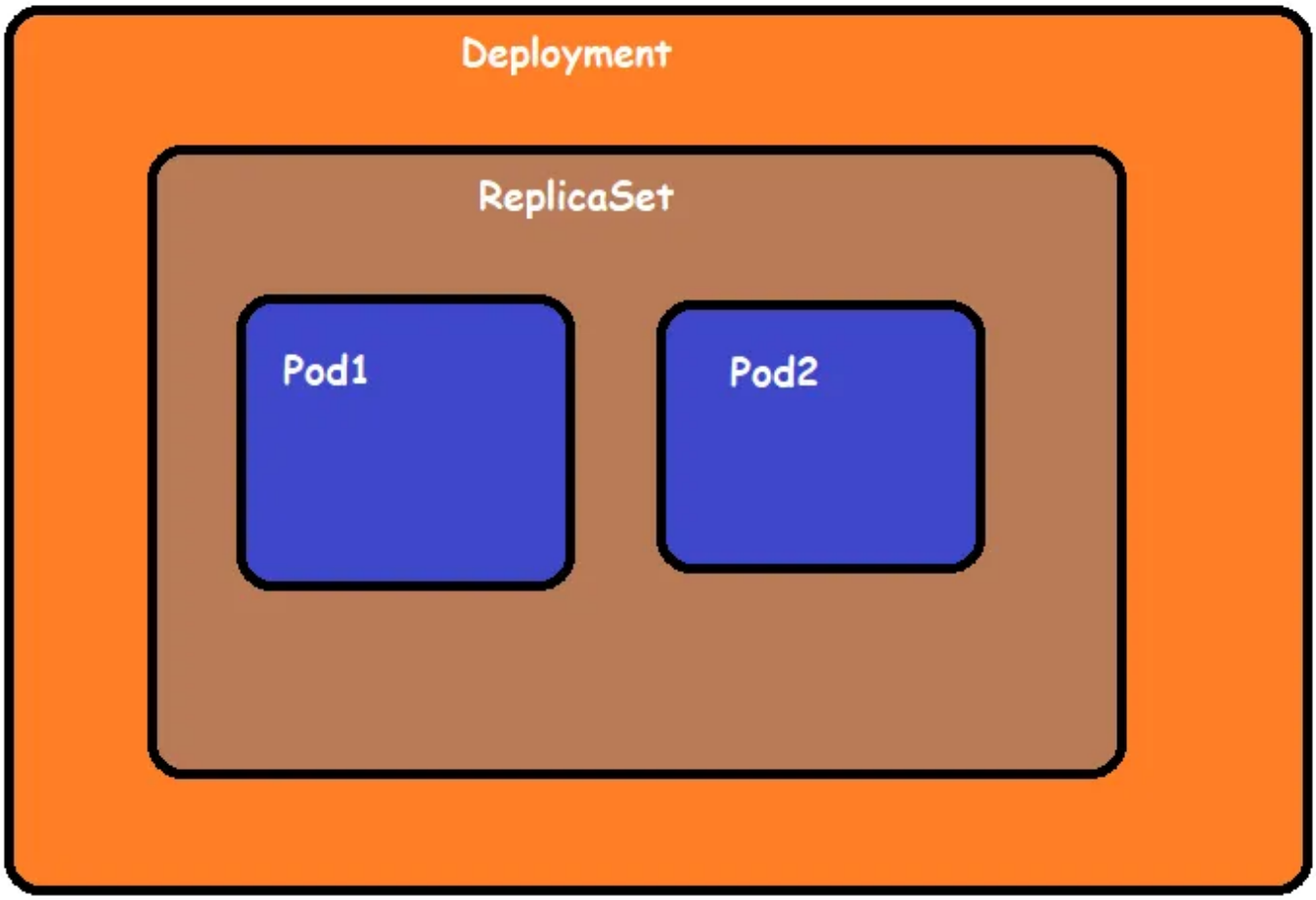
Apply the spec Refer Here

```
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: daemonset-demo
spec:
  selector:
    matchLabels:
      app: alpine
  template:
    metadata:
      labels:
        app: alpine
    spec:
      containers:
        - name: alpine-cont
          image: alpine
          args:
            - /bin/sh
            - -c
            - sleep 1d
```

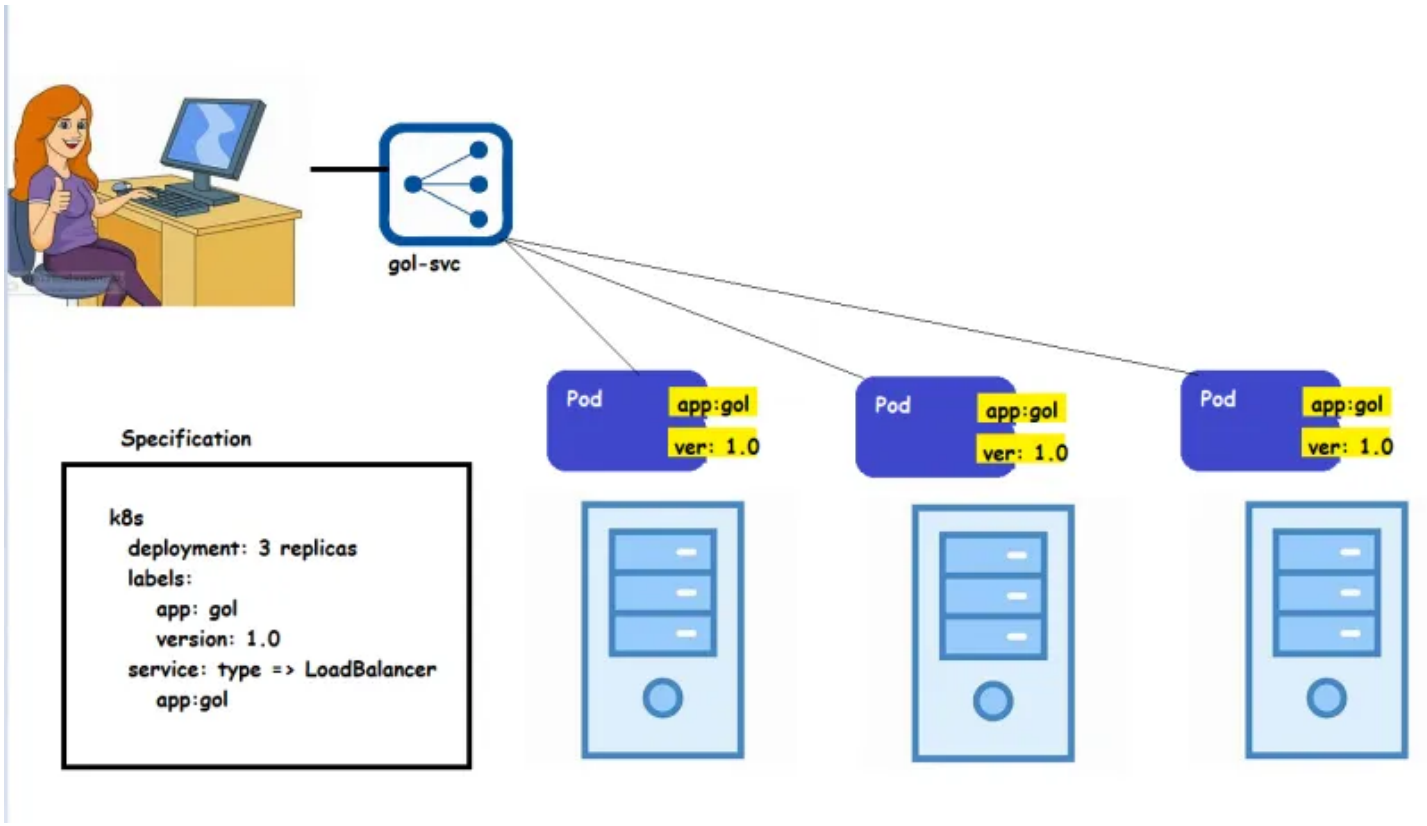
```
qtkhaja@Azure:~$ kubectl apply -f daemonset.yaml
daemonset.apps/daemonset-demo created
qtkhaja@Azure:~$ kubectl api-resources | grep daemon
daemonsets                        ds           apps                           true         DaemonSet
qtkhaja@Azure:~$ kubectl get ds
NAME            DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
daemonset-demo   3         3         3       3            3           <none>          32s
qtkhaja@Azure:~$ kubectl get pods
NAME                   READY   STATUS    RESTARTS   AGE
daemonset-demo-4vmbn   1/1     Running   0          39s
daemonset-demo-b66cp   1/1     Running   0          39s
daemonset-demo-rwsz9   1/1     Running   0          39s
qtkhaja@Azure:~$ kubectl get pods -o wide
NAME                   READY   STATUS    RESTARTS   AGE   IP            NODE                                NOMINATED NODE   READINESS
GATES
daemonset-demo-4vmbn   1/1     Running   0          46s   10.244.1.3    aks-nodepool1-34786526-vmss000001   <none>           <none>
daemonset-demo-b66cp   1/1     Running   0          46s   10.244.2.3    aks-nodepool1-34786526-vmss000002   <none>           <none>
daemonset-demo-rwsz9   1/1     Running   0          46s   10.244.0.17   aks-nodepool1-34786526-vmss000000   <none>           <none>
qtkhaja@Azure:~$
```

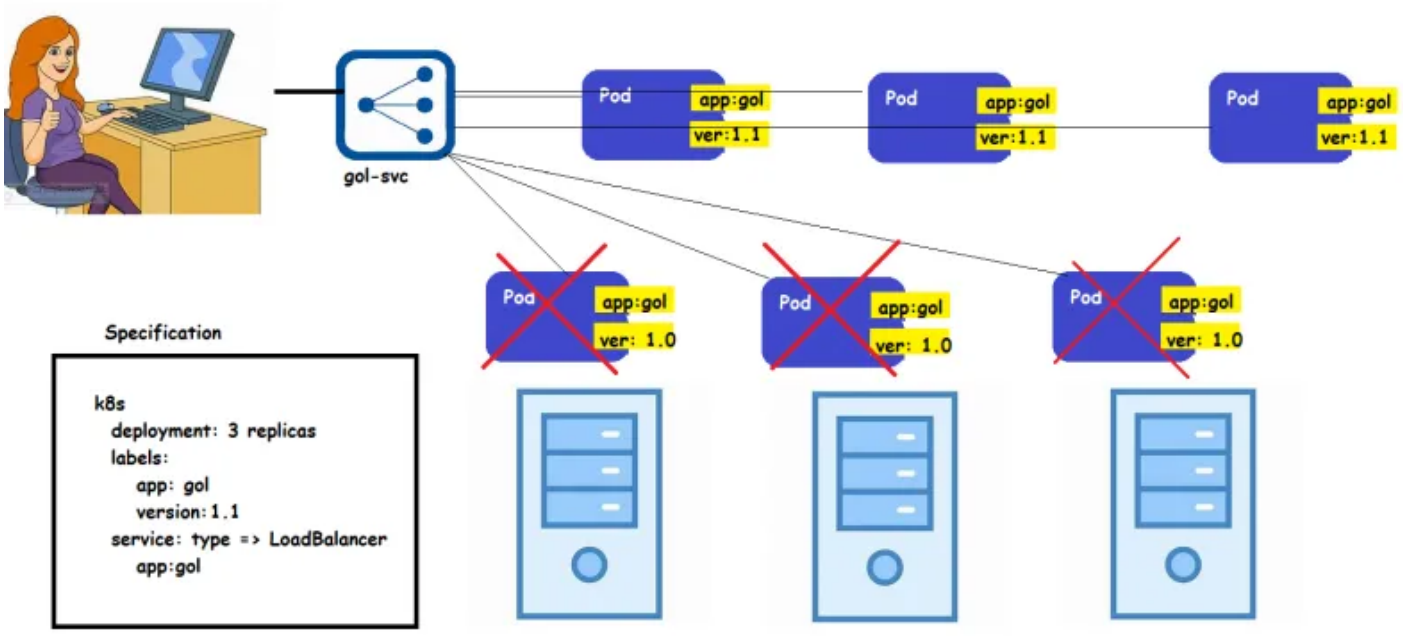
Topic: K8S Deployments (K8s Controller type-4)

Deployment is a k8s object that acts as a wrapper around ReplicaSet and makes it easier to use



- The major reason for Deployment is
  - it maintains the history of revisions (i.e. history of your application Deployments)
  - it makes it easy to roll back to previous revisions
  - it gives a way to do zero downtime deployments
- Deployment Strategy:
  - Rolling Update:
    - This give an option to deploy your application without having downtime
    - This gives maxUnavailable, maxSurge
  - Recreate:
    - This option kills all the running pods and recreates new pods





- Refer Here for deployment commands
- Now lets write a sample deployment Refer Here

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gameoflife-deploy
  labels:
    app: gameoflife
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  selector:
    matchLabels:
      app: gameoflife
      version: "1.0"
  template:
    metadata:
      labels:
        app: gameoflife
        version: "1.0"
    spec:
      containers:
        - image: qualitythought/gameoflife:07112020
          name: gol-dev
          ports:
            - containerPort: 8080
              protocol: TCP


---
apiVersion: v1
kind: Service
metadata:
  name: gol-dev
spec:
  type: LoadBalancer
  selector:
    app: gameoflife
  ports:
    - targetPort: 8080
      port: 80
```

```
qtkhaja@Azure:~$ kubectl apply -f gol-deployment.yaml
deployment.apps/gameoflife-deploy created
qtkhaja@Azure:~$ kubectl get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
gameoflife-deploy   1/3     3            1           11s
qtkhaja@Azure:~$ kubectl get deploy -w
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
gameoflife-deploy   2/3     3            2           16s
gameoflife-deploy   3/3     3            3           16s
^Cqtkhaja@Azure:~$ kubectl get pods
NAME                                 READY   STATUS    RESTARTS   AGE
gameoflife-deploy-754b466d7f-hlk9b   1/1     Running   0          33s
gameoflife-deploy-754b466d7f-krflj   1/1     Running   0          33s
gameoflife-deploy-754b466d7f-xkvwl   1/1     Running   0          33s
qtkhaja@Azure:~$ kubectl apply -f gol-svc.yaml
service/gol-dev created
qtkhaja@Azure:~$ kubectl get svc -w
NAME         TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
gol-dev      LoadBalancer   10.0.83.222   <pending>      80:31136/TCP   6s
kubernetes   ClusterIP      10.0.0.1      <none>         443/TCP        15h
gol-dev      LoadBalancer   10.0.83.222   52.151.219.235   80:31136/TCP   14s
```

```
qtkhaja@Azure:~$ kubectl describe rs gameoflife-deploy
Name:           gameoflife-deploy-754b466d7f
Namespace:      default
Selector:       app=gameoflife,pod-template-hash=754b466d7f,version=1.0
Labels:         app=gameoflife
                pod-template-hash=754b466d7f
                version=1.0
Annotations:    deployment.kubernetes.io/desired-replicas: 3
                deployment.kubernetes.io/max-replicas: 4
                deployment.kubernetes.io/revision: 1
Controlled By:  Deployment/gameoflife-deploy
Replicas:       3 current / 3 desired
Pods Status:    3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=gameoflife
           pod-template-hash=754b466d7f
           version=1.0
```

```
qtkhaja@Azure:~$ kubectl rollout history deployment gameoflife-deploy
deployment.apps/gameoflife-deploy
REVISION  CHANGE-CAUSE
1         <none>

qtkhaja@Azure:~$
```

```
qtkhaja@Azure:~$ kubectl apply -f gol-deployment.yaml --record
deployment.apps/gameoflife-deploy created
qtkhaja@Azure:~$ kubectl get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
gameoflife-deploy   3/3     3            3           8s
qtkhaja@Azure:~$ kubectl apply -f gol-deployment.yaml --record
deployment.apps/gameoflife-deploy configured
qtkhaja@Azure:~$ kubectl get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
gameoflife-deploy   2/3     2            2           81s
qtkhaja@Azure:~$ kubectl get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
gameoflife-deploy   2/3     2            2           103s
qtkhaja@Azure:~$ kubectl get deploy -w
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
gameoflife-deploy   2/3     3            2           106s
gameoflife-deploy   3/3     3            3           106s
gameoflife-deploy   3/3     3            3           106s
gameoflife-deploy   2/3     3            2           106s
gameoflife-deploy   3/3     3            3           108s
```

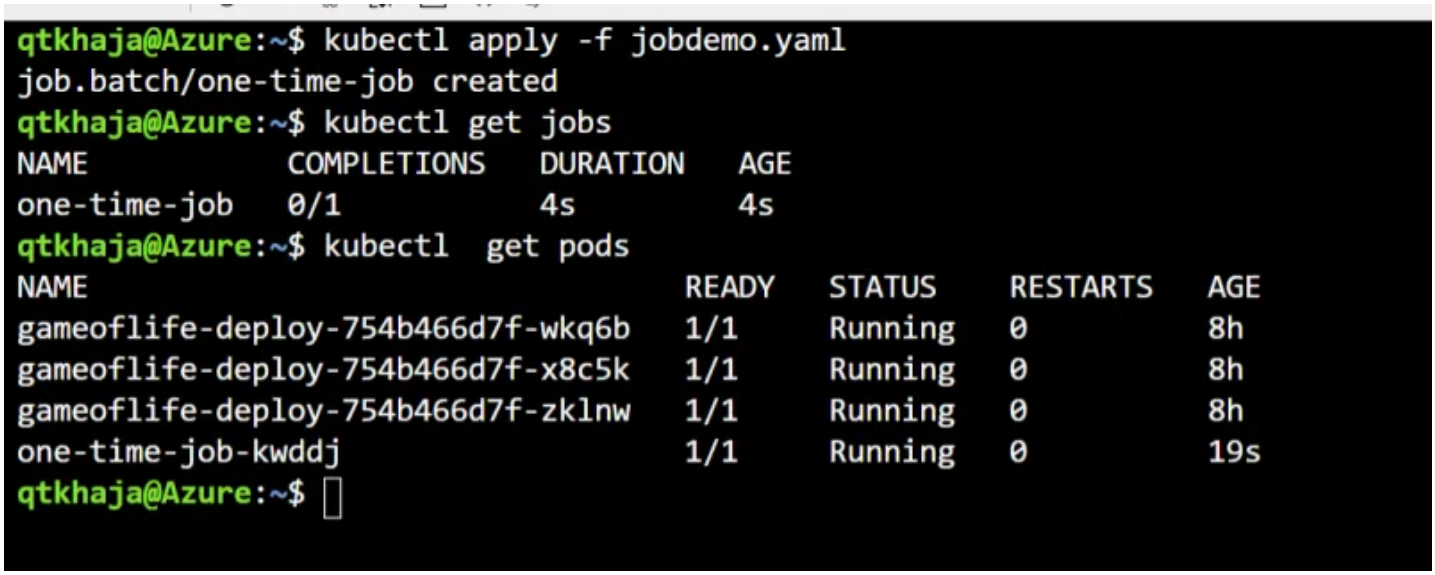
```
qtkhaja@Azure:~$ kubectl rollout history deployment gameoflife-deploy
deployment.apps/gameoflife-deploy
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=gol-deployment.yaml --record=true
2         kubectl apply --filename=gol-deployment.yaml --record=true

qtkhaja@Azure:~$ kubectl rollout undo deployment gameoflife-deploy --to-revision=1
deployment.apps/gameoflife-deploy rolled back
qtkhaja@Azure:~$ kubectl rollout status
error: required resource not specified
qtkhaja@Azure:~$ kubectl rollout status deployment gameoflife-deploy
deployment "gameoflife-deploy" successfully rolled out
qtkhaja@Azure:~$ kubectl get deploy
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
gameoflife-deploy   3/3     3            3           7m55s
qtkhaja@Azure:~$
```

- Follow the complete application creation in k8s Refer Here

Topic:- K8s Jobs and CronJobs

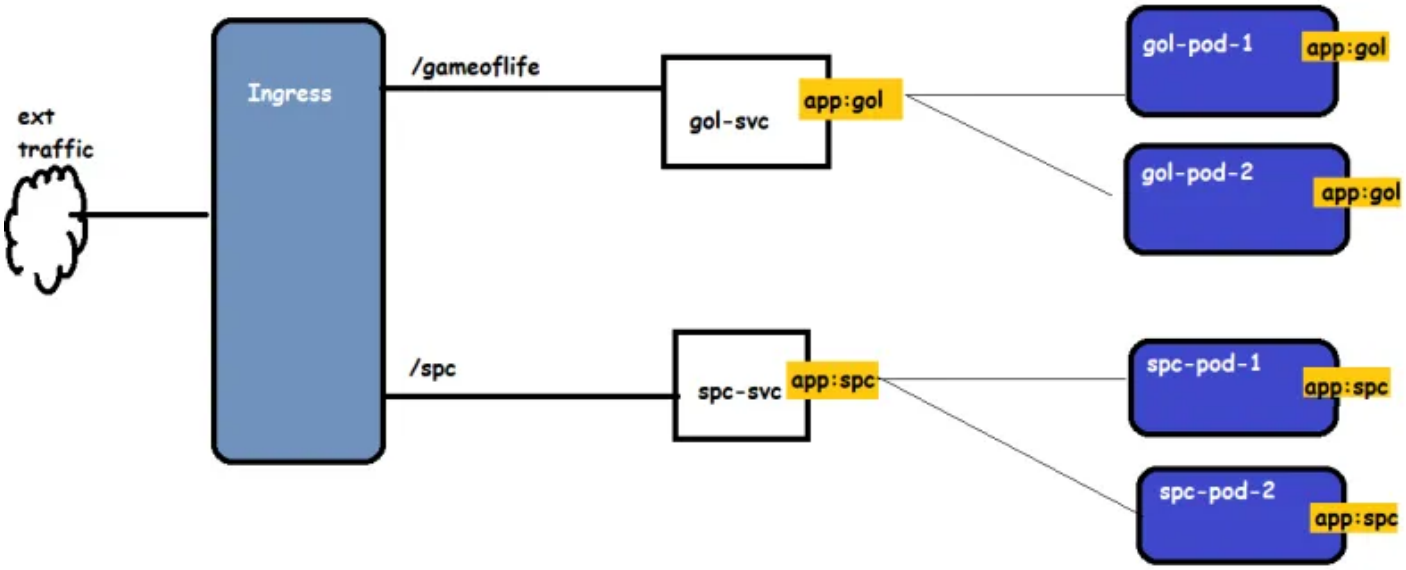
- Let's assume you want to run a Pod which runs a task and then terminates gracefully, To solve these kind of Problems k8s has Jobs & CronJobs
- The difference b/w k8s job & CronJob is CronJobs can be scheduled to run on a particular time frame
- Run the job as mentioned in the changeset Refer Here

```
---
apiVersion: batch/v1
kind: Job
metadata:
  name: one-time-job
spec:
  template:
    spec:
      restartPolicy: OnFailure
      containers:
        - name: ubuntu-container
          image: ubuntu
          args:
            - /bin/bash
            - -c
            - sleep 10m
```



Topic: K8s Ingress

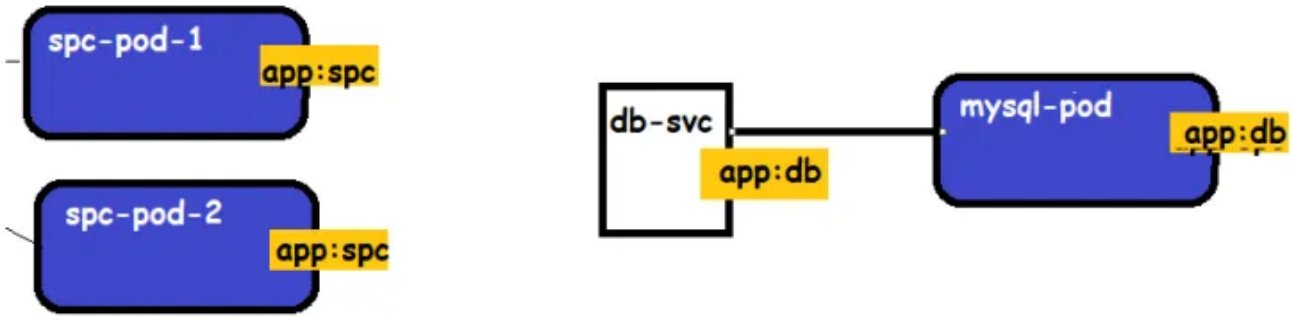
- Ingress is an object in k8s that is used manage external access to services in the k8s cluster. Ingress provides layer 7 load balancing, so it will be aware of url's
- Refer Here {https://docs.microsoft.com/en-us/azure/aks/ingress-basic}



Scenario:

- I have a application component which is running in a pod
- I have a database pod
- Application Pod is configured to scale automatically

- Now application pod needs to access db
- Approach 1:
    - Create a service for db



- Whenever you add a service an environmental variables will be added in all pods containing svc ip and port

```
DIRSTACK=()
EUID=0
GOL_DEV_PORT=tcp://10.0.83.222:80
GOL_DEV_PORT_80_TCP=tcp://10.0.83.222:80
GOL_DEV_PORT_80_TCP_ADDR=10.0.83.222
GOL_DEV_PORT_80_TCP_PORT=80
GOL_DEV_PORT_80_TCP_PROTO=tcp
GOL_DEV_SERVICE_HOST=10.0.83.222
GOL_DEV_SERVICE_PORT=80
GPG_KEYS='05AB33110949707C93A279E3D3EFE6B686867BA6 07E48665A34DCAFAE522
1FBE7D8F78B25E055DDEE13C370389288584E7 61B832AC2F1C5A90F0F9B00A1C506407
B966EBA586F72C284D731FABEE A27677289986DB50844682F8ACB77FC2E86E29AC A9C
B21E8933C60243 F3A04C595DB5B6A5F1ECA43E3B7BBB100D811BBE F7DA48BB64BCB84
GROUPS=()
HISTFILE=/root/.bash_history
```

Approach 2:

- Use a Config Map. *Config Map* is a k8s object that allows us to define the application-related data.
- Config maps are key value pairs
- You can create config maps from
    - literal value

    - file
    - directory
- Lets create a sample config map

```
kubectl create configmap demovalue-map --from-literal=db-ip=10.0.83.222
```

---
apiVersion: v1
kind: Pod
metadata:
  name: configmapdemo
spec:
  containers:
    - name: configmapdemo
      image: alpine
      command: ["/bin/bash", "-c", "sleep 1d"]
      envFrom:
        - configMapRef:
            name: demovalue-map

```
qtkhaja@Azure:~$ kubectl exec -it configmap-pod -- /bin/sh
/ # set
GOL_DEV_PORT='tcp://10.0.83.222:80'
GOL_DEV_PORT_80_TCP='tcp://10.0.83.222:80'
GOL_DEV_PORT_80_TCP_ADDR='10.0.83.222'
GOL_DEV_PORT_80_TCP_PORT='80'
GOL_DEV_PORT_80_TCP_PROTO='tcp'
GOL_DEV_SERVICE_HOST='10.0.83.222'
GOL_DEV_SERVICE_PORT='80'
HISTFILE='/root/.ash_history'
HOME='/root'
HOSTNAME='configmap-pod'
IFS='
```

NOTE:- In K8s we also have secret which are much like config maps but they can store secret of size 1kb in base 64 encoded form

Topic: Helm

- Helm is a Package manager for k8s
- Helm charts represent a k8s package
- Installing helm Refer Here
- Lets execute an example using helm in AKS Refer Here
- In k8s we write yaml manifests, helm simplifies and makes manifests reusable
- Refer Here
- Journey of App from docker to k8s Refer Here