HEALTHCHECKS

HEALTH CHECKS

- If your application malfunctions, the pod and container can still be running, but the application might not work anymore.

- To detect and resolve problems with your application, you can run healthchecks:

- Two different types of HC:

- - Running a command in the container periodically
- - Periodic checks on a URL (HTTP)

- the typical production application behind a load-balancer should always have heathchecks implemented in some way to ensure availability and resiliency of the app

HEALTH CHECKS

kubectl get nodes

kubectl get nodes --show-labels

kubectl create -f <yamlfile>

kubectl describe pod (you can edit the values)

kubectl edit pod <POD-NAME>

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-podmonitor
  labels:
    app: myapp
    type: front-end
spec:
  containers:
  - name: httpd-container
    image: httpd
    ports:
    - name: httpd-port
      containerPort: 8080

    livenessProbe:
      httpGet:
        path: /
        port: 8080
      initialDelaySeconds: 15
      timeoutSeconds: 30
  nodeSelector:
    hardware: high-spec
```

Static Pod

Individual pods on Workers

Static Pods

Static Pods are managed directly by the kubelet daemon on a specific node, without the API server observing them.

Unlike Pods that are managed by the control plane (for example, a Deployment); instead, the kubelet watches each static Pod (and restarts it if it fails).

Static Pods are always bound to one Kubelet on a specific node.

The kubelet automatically tries to create a mirror Pod on the Kubernetes API server for each static Pod. This means that the Pods running on a node are visible on the API server, but cannot be controlled from there.

The Pod names will be suffixed with the node hostname with a leading hyphen.

Static Pods

K8S version >1.21.x:

Put pod.yaml file under below path on node to get it started automatically:
/etc/kubernetes/manifests/

For previous versions:
1) Create a directory i.e. /etc/kubelet.d/
2) Put yaml files into it
3) Run below command:

```
KUBELET_ARGS="--cluster-dns=10.254.0.10 --cluster-domain=kube.local --pod-manifest-path=/etc/kubelet.d/“
```

4) Restart kubelet via

```
systemctl restart kubelet
```

Init Container

Start before others

Init Containers

What if you need one container in pod to run successfully, before other containers get started in it to setup the base (like Filesystem, Binaries availability/Dependency etc)?

A Pod can have multiple containers running apps within it, but it can also have one or more init containers, which are run before the app containers are started.

Init containers are exactly like regular containers, except:

Init containers always run to completion.
Each init container must complete successfully before the next one starts.

If a Pod's init container fails, the kubelet repeatedly restarts that init container until it succeeds.

However, if the Pod has a restartPolicy of Never, and an init container fails during startup of that Pod, Kubernetes treats the overall Pod as failed.

Init Containers

https://kubernetes.io/docs/concepts/workloads/pods/init-containers/

DaemonSet

Run on all Machines

DaemonSets

A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

Some typical uses of a DaemonSet are:

- running a cluster storage daemon on every node
- running a logs collection daemon on every node
- running a node monitoring daemon on every node

https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

SECRETS

SECRETS

- Secrets provides a way in K8's to distribute credentials, keys, passwords or secret data to the pods.

- K8 itself used this mechanism to provide the credentials to access the internal API's

- Secrets is one way to provide secrets, native to K8, other way is by using external vault services

- Secrets can be used as: **environment variables**
- or **as a file in a pod** (this need volume to be mounted and volumes have the files having secrets)

- First generate a Secrets using files:
- echo -n "root" > ./username.txt
- echo -n "password" > ./password.txt

- kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt

- **NOTE: You can also use SSH keys**

SECRETS

```
echo -n "root" | base64
echo -n "password" | base64
```

```
apiVersion: v1
kind: Secret
metadata:
  name: db-secrets
type: Opaque
data:
  username: cm9vdA==
  password: cGFzc3dvcmQ=
```

```
kubectl create -f <yaml>
kubectl get secrets
kubectl describe secrets <Secret-Name>
```

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
  - name: httpd-container
    image: httpd
    volumeMounts:
    - name: cred-volume
      mountPath: /etc/creds
      readOnly: true
  volumes:
  - name: cred-volume
    secret:
      secretName: <Secret-Name>
```

```
kubectl create -f <YAML>
kubectl describe pod <POD-Name>
```

Login into the container and see the secrets will be in:

/etc/creds/username
/etc/creds/password

Execute mount and you will see default secrets shared by Kubernetes: tmpfs on /run/secrets/kubernetes.io/serviceaccount type tmpfs (ro,relatime,seclabel)

SECRETS

Secrets with Environment Variables

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
  - name: httpd-container
    image: httpd
  env:
  - name: SECRET_USERNAME
    valueFrom:
      secretKeyRef:
        name: db-secret
        key: username
  -name: SECRET_PASSWD
............
```

ConfigMaps

ConfigMaps

- Configuration parameters that are not secret, can be put in a configmap

- The input is again key-value pair and can be read from any of the below methods:

- - environments variables
- - volumes
- - Container command line parameters

- A configmap can be a complete configuration file, This file can be then mounted using volumes where applications are expecting the file

- This way we can change the container without changing the container directly (without changing the container image)

- **CREATE A CONFIG FILE**
- **kubectl create configmap <CONFIG_MAP_NAME> --from-file=<file-name>**
- **Then create a pod with volumes**

ConfigMaps

```
/usr/local/apache2/htdocs/index.html

<html><body><h1>It works again with
Configmaps!</h1></body></html>

kubectl create configmap nginx-config
--from-file=index.html

kubectl get configmap

kubectl get configmap -o yaml

(here you will see a key (in our case
index.html) and values)
```

```
apiVersion: v1
kind: Pod
metadata:
  name: helloworld-nginx
  labels:
    app: helloworld-nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    volumeMounts:
    - name: config-volume
      mountPath:
/usr/share/nginx/html
  volumes:
    - name: config-volume
      configMap:
        name: nginx-config
        items:
        - key: index.html
          path: index.html
```

```
apiVersion: v1
kind: Service
metadata:
  name: helloworld-
nginx-service
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: helloworld-
nginx
  type: NodePort
```

PV and PVC

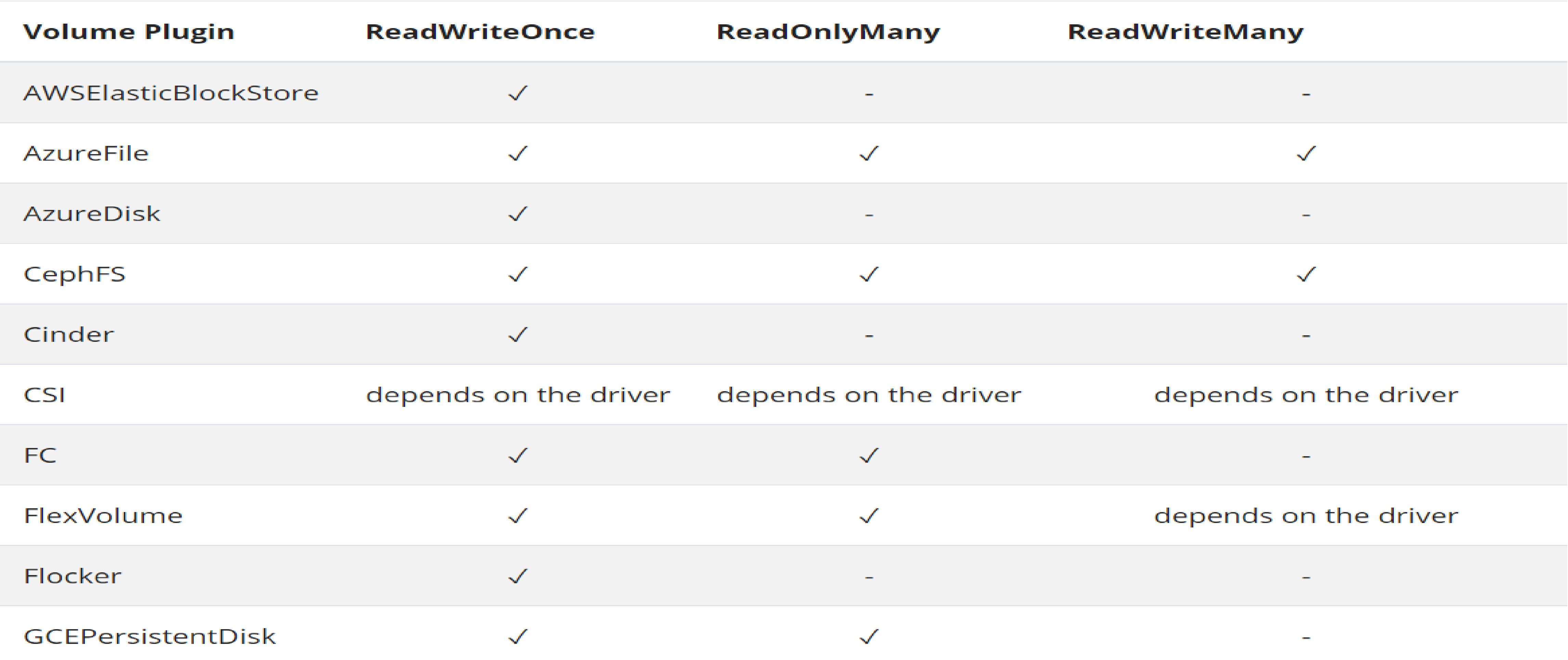
Persistent Storage in Kubernetes

Persistent Storage

A Persistent Volume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes but have a lifecycle independent of any individual Pod that uses the PV.

A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany

Access Types and Methods



Volume Plugin	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWSElasticBlockStore	✓	-	-
AzureFile	✓	✓	✓
AzureDisk	✓	-	-
CephFS	✓	✓	✓
Cinder	✓	-	-
CSI	depends on the driver	depends on the driver	depends on the driver
FC	✓	✓	-
FlexVolume	✓	✓	depends on the driver
Flocker	✓	-	-
GCEPersistentDisk	✓	✓	-

Persistent Volume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```

Persistent Volume Claim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

DashBoards

DashBoards

- K8’s comes with a WebUI you can use instead of the kubectl commands.
- You will get an overview of running applications on your cluster.
- Creating and Modifying individual kubernetes resources and workloads can also be done like KUBECTL create and delete command.
- In general you can access the kubernetes webUI at https://<MASTER>/ui
- Note: Now the process has been secured and changed since 1.8+ version

DashBoards

- https://github.com/kubernetes/dashboard

- cd dashboard-master/ src/deploy/recommended/ kubernetes-dashboard.yaml

- Expose the NodePort for AWS/Public IP’s, and your service should look like:

- spec:
-   **type: NodePort**
-   ports:
-     - port: 443
-       targetPort: 8443
-   selector:
-     k8s-app: kubernetes-dashboard

DashBoards

- kubectl create -f kubernetes-dashboard.yaml
- kubectl get pods -n namespace
- kubectl get pods -n kube-system
- kubectl get service -n kube-system

- Excess the dashboard using WEB with mentioned port

- Now use the below link to create a user and view the dashboard:

- https://github.com/kubernetes/dashboard/wiki/Creating-sample-user

DashBoards

- https://github.com/kubernetes/dashboard/wiki/Creating-sample-user

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kube-system
```

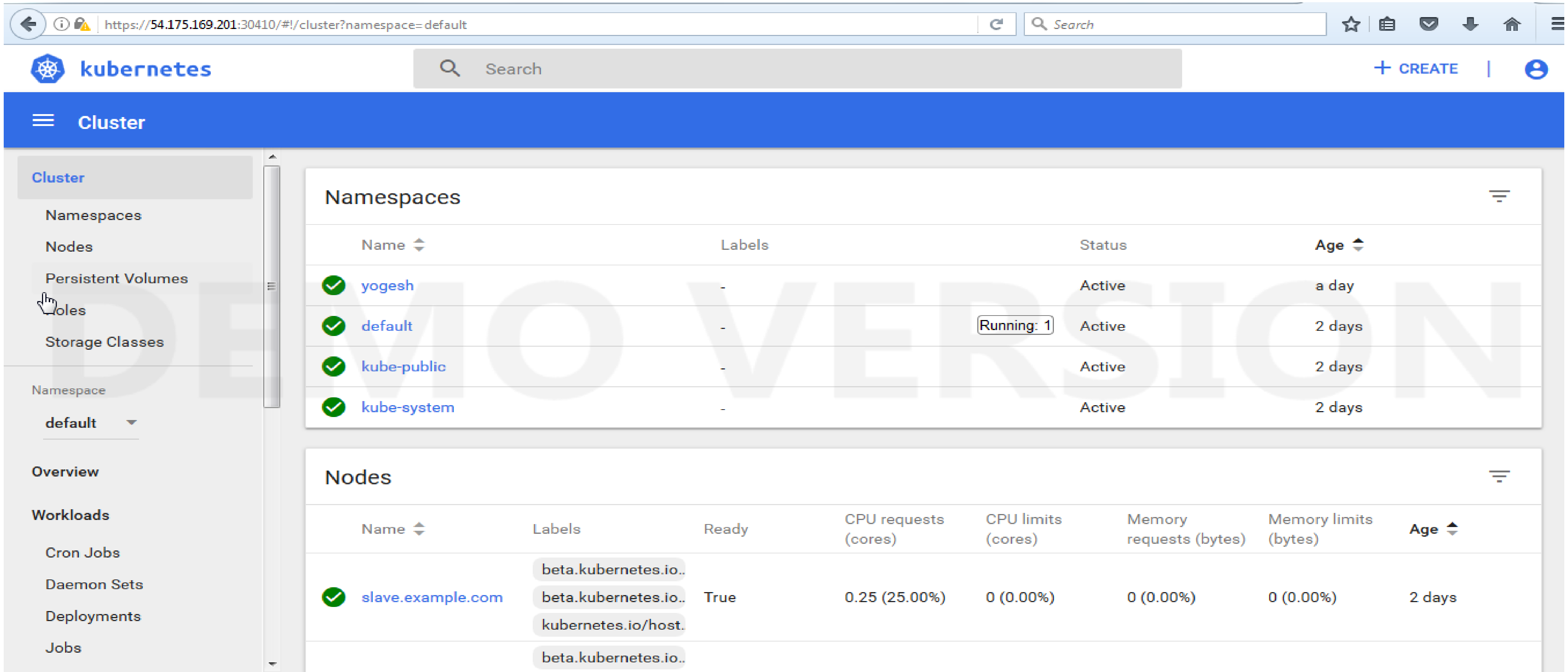
```
apiVersion:
rbac.authorization.k8s.io/v1b
eta1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup:
rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kube-system
```

```
$ kubectl create -f
<yaml.file>

$ kubectl -n kube-system
describe secret $(kubectl -n
kube-system get secret | grep
admin-user | awk '{print
$1}')

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

Service Accounts can be managed with CLI

```
kubectl get serviceaccounts --all-namespaces

kubectl get roles --all-namespaces

kubectl get rolebinding

kubectl create serviceaccount gagan

kubectl create role pod-reader --verb=get --verb=list --verb=watch --resource=pods

kubectl create rolebinding gagan --clusterrole=pod-reader --user=gagan
```

DashBoards



Namespaces

	Name	Labels		Status	Age
✓	yogesh	-		Active	a day
✓	default	-	Running: 1	Active	2 days
✓	kube-public	-		Active	2 days
✓	kube-system	-		Active	2 days

Nodes

	Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age
✓	slave.example.com	beta.kubernetes.io.. beta.kubernetes.io.. kubernetes.io/host.	True	0.25 (25.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	2 days
		beta.kubernetes.io..						

Addition/Deletion of a node to the Cluster

Node Taint & Toleration

Node affinity is a property of Pods that *attracts* them to a set of nodes (either as a preference or a hard requirement).

Taints are the opposite -- they allow a node to repel a set of pods.

Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints.

https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/

Node cordon and draining

```
kubectl cordon my-node                         # Mark my-node as unschedulable

kubectl drain my-node                          # Drain my-node in preparation for

maintenance

kubectl uncordon my-node                       # Mark my-node as schedulable
```

Remove node

To remove a Kubernetes worker node from the cluster, perform the following operations.

Migrate pods from the node

```
$ kubectl drain  <node-name> --delete-local-data --ignore-daemonsets
```

Prevent a node from scheduling new pods use – Mark node as unschedulable

```
$ kubectl cordon <node-name>
```

Revert changes made to the node by ‘kubeadm join‘ – Run on worker node to be removed

```
$ sudo kubeadm reset # run it from worker node
```

Remove the node from cluster

```
$ kubectl delete node <node-name>
```

Node Addition

- Create a New instance and perform all of the pre-requisties.
- Check the token is valid or not by running "kubeadm token list"
- Now generate the token using "kubeadm token create --print-join-command"
- Run "kubeadm reset" on node if you ran the old join command
- Finally run the newly generated token command
- Go to master and run "kubectl get nodes"

NodeSelectors

Node Selector

- You can also use labels to tag your nodes
- Once nodes are tagged, you can use label selectors to let pods only run specific nodes
- there are two steps required to run a pod on a specific set of nodes

- first you tag the node

- then you add a nodeSelector to your pod configuration

Node Selector

```
kubectl get nodes
kubectl get nodes --show-labels
kubectl create -f <yamlfile>
```

Run the shown yaml file and check the pod status you will find the error stating "node(s) didn't match node selector."

```
kubectl get pods
kubectl describe pod <pod-name>
kubectl label nodes <nodename> hardware=high-spec
kubectl label nodes <nodename> hardware=low-spec
```

As soon as you label the node you will get the POD in running condition (auto-healing)

CASE2: Change the nodeSelector to other node and re-run

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
  - name: httpd-container
    image: httpd
  nodeSelector:
    hardware: high-spec
```

Setting Role Labels

Adding a Role to a Worker Node:

```
[root@master ~]# kubectl label node worker1 node-role.kubernetes.io/linux-worker=worker1
node/worker1 labeled
[root@master ~]# kubectl get nodes
NAME      STATUS   ROLES          AGE   VERSION
master    Ready    master         12d   v1.19.2
worker1   Ready    linux-worker   12d   v1.19.2
```

Removing a Role Label

```
[root@master ~]# kubectl label node worker1 node-role.kubernetes.io/linux-worker-
node/worker1 labeled
[root@master ~]# kubectl get nodes
NAME      STATUS   ROLES    AGE   VERSION
master    Ready    master   12d   v1.19.2
worker1   Ready    <none>   12d   v1.19.2
[root@master ~]#
```

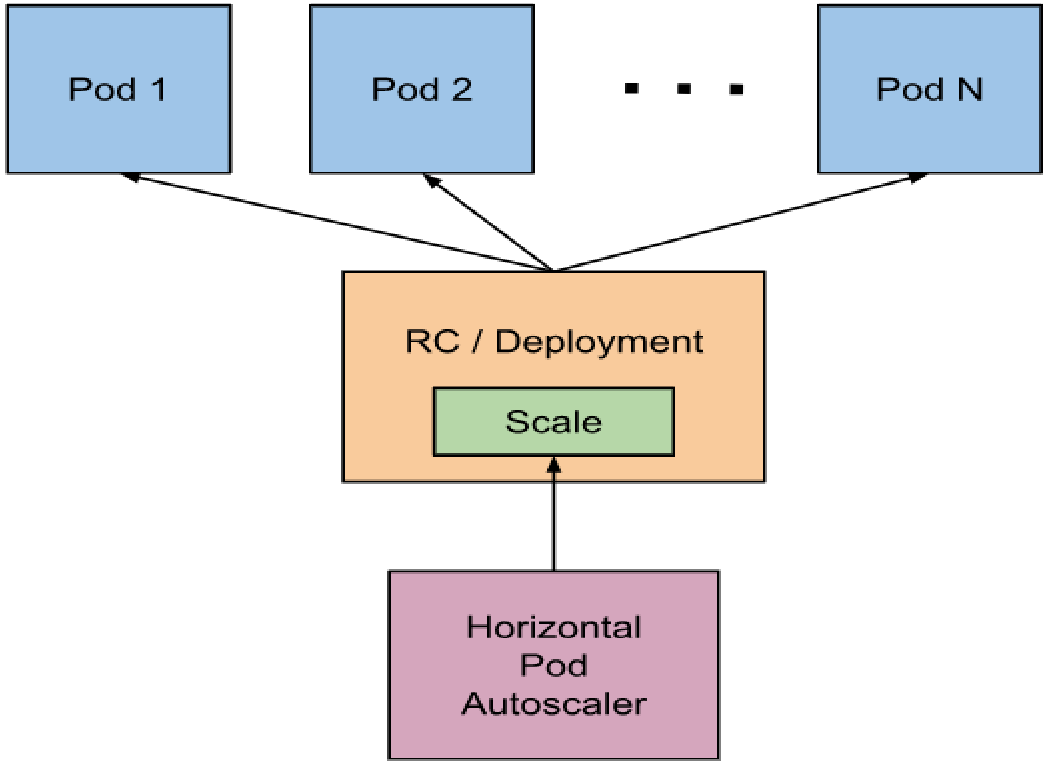
Kubernetes as a Service (PaaS Platform)

Kubernetes as a Service

- Many Cloud Service providers provides Kubernetes as a Service.
- They manages Kubernetes master service (Somewhere charged, somewhere no charges)
- Worker nodes are charged and assigned to us.
- We can deploy pods via GUI as well as command line.
- Examples – EKS, GKE, AKS

Autoscaling

Autoscaling



Autoscaling

```
kubectl autoscale deployment  myapp-deployment --cpu-percent=50 --min=1 --max=10
```

```
kubectl get hpa
```

```
#but you need to set a metric deployment first otherwise it'wont be able to collect the metric
```

```
kubectl delete hpa myapp-deployment
```

https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

Helm Charts

Helm

Kubernetes Package Manager

In June 2018, Helm was adopted as an official CNCF project.

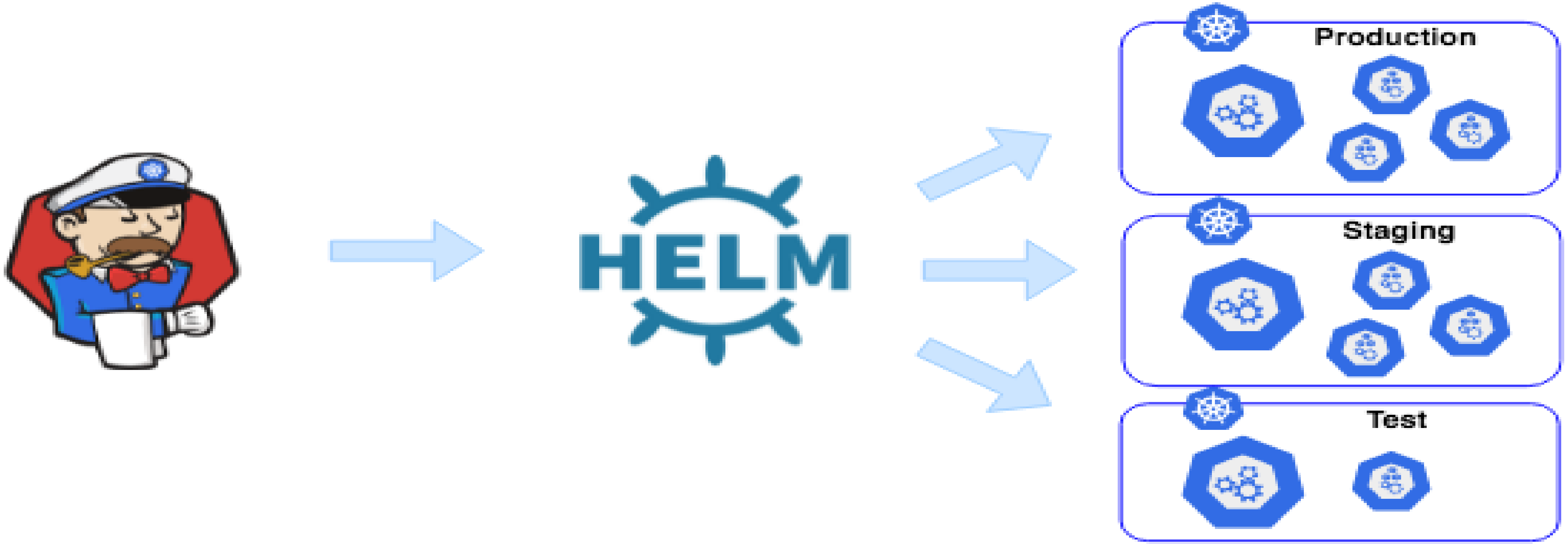
A Helm chart is simply a collection of YAML template files organized into a specific directory structure.

It play a large role in optimizing an organization’s CI/CD integration with Kubernetes.

Manage multiple environments together

With you you’ll have ability to provide application configuration during deployment. Not only you can specify the Kubernetes resources (deployments, services, etc.) that make up your application, but also environment-specific configuration for those resources.

Helm Charts



Working with Helm

Do the Installation by running below or reach out to https://helm.sh/docs/intro/install:

```
https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash
```

Create chart with:

```
helm create [chartname]
```

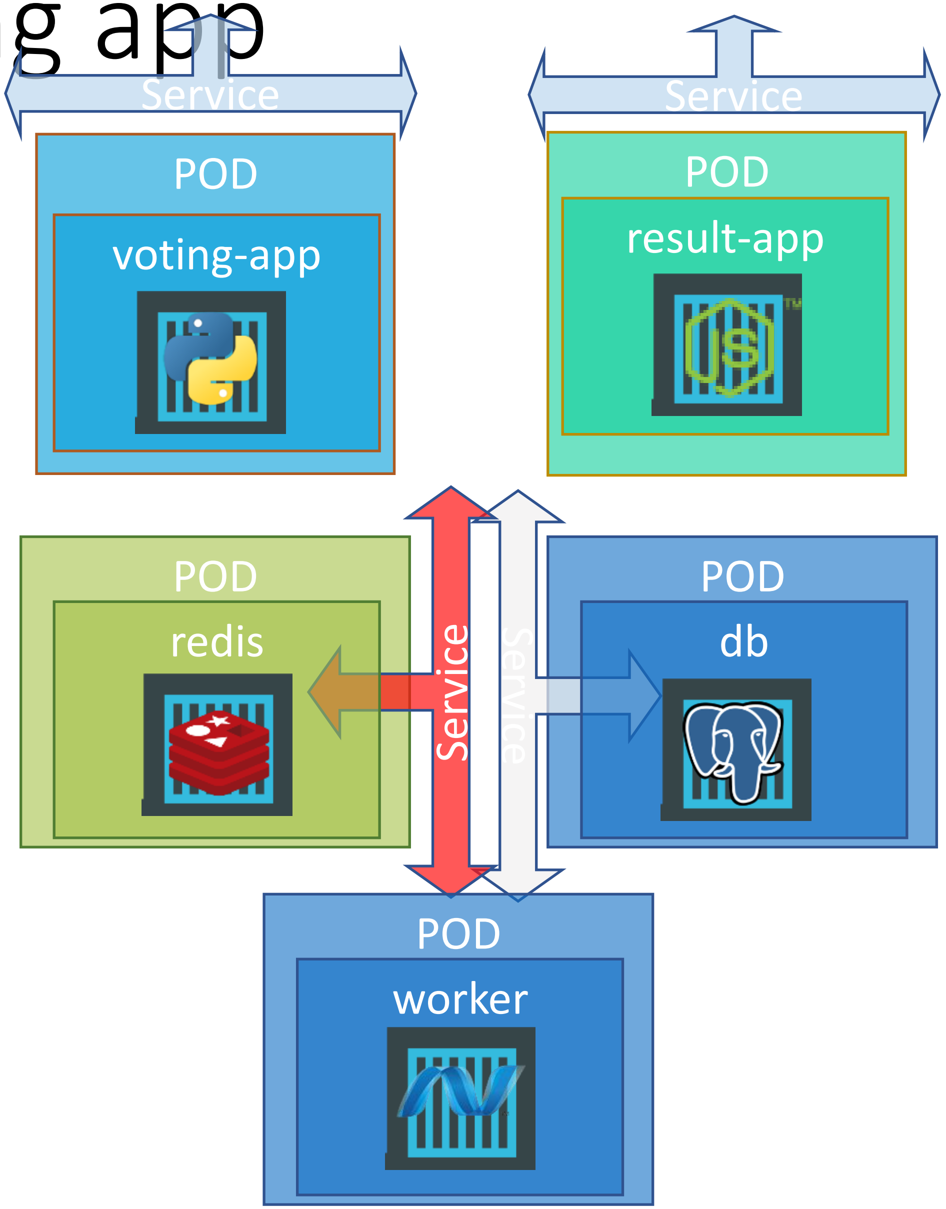
Modify the values.yaml and put entries as per your requirement

```
helm package [chartname]   // to create package
helm install [chartname]  // to Install a chart
Helm ls    // to list charts
Helm upgrade [generated-dep-name] chart-name   // to upgrade
Helm rollback [generated-dep-name] version         // to rollback
Helm delete [generated-dep-name]                                     // to delete the deployed packages
```

Example voting app

Goals:

1. Deploy Containers
2. Enable Connectivity
3. External Access



Steps:

1. Deploy PODs
2. Create Services (ClusterIP)
3. Create Services (LoadBalancer)

Stay connected

Sandeep.kumar@techlanders.com

contactus@techlanders.com