**K8S CKA**

**1. Create new ClusterRole and binding it to a ServiceAccount**

1- Create a new ClusterRole named deployment-clusterone, which only allows to create the following resources type:

* Deployment, StatefulSet DaemonSet

**Answers: k create clusterRole deployment-clusterone --verb=create --resource=deployment,statefulset,deamonset**

2- Create a new ServiceAccount named “**cicd-token**” in the existing namespace “**app-team1**”.

Answer: k create namespace app-team1

Exam: k create serviceaccount cicd-token –n app-team1

3- Bind the new ClusterRole “**deployment-clusterone**” to the new ServiceAccount “**cicd-token**”, limited to the namespace “**app-team1**”

Answer: : k create clusterrolebinding **deployment-clusterbinding --clusterrole=deployment-clusterone --serviceaccount=app-team1:cicd-token**

# 2 Set node name as unavailable and reschedule its pods/ tape cheat sheet to search for interactive node

Set the node named “**ek8s-node-1**“ as unavailable and reschedule all the pods running on it

**NB: ek8s-node-1 is already in the cluster exam.choose it then run the following command**. But for your practice use your nodes ip then run THE FIIRST COMMAND TO UNCORDON: K UNCORDON +YOUR IP NODE

**Kubectl cordon ek8s-node-1**

**Kubectl drain ek8s-node-1 --ignore-daemonsets**

# 3 Upgrade kubeadm cluster/ tape upgrade on kube.io (copy and paste into any vim file you will create to be able to modified the version then paste the modified )

Given an existing kubernetes cluster running version 1.18.8. Upgrade all of the kubernetes control plane and node components on the MASTER NODE ONLY to version 1.19.0

* Ssh manager
* Sudo –i

apt-mark unhold kubeadm && **\**

apt-get update && apt-get install -y kubeadm=1.27.x-00 && **\**

apt-mark hold kubeadm

kubeadm upgrade plan

sdo kubeadm upgrade apply v1..19.0

# 4.Create/restore a snapshot of the etcd instance/ keywords is in backup etcd instance/ on snapshot using etcd options, copy the command and go run it into a fictive vim file to be able to modify it.

**Task:**

1- Create a snapshot of the existing etcd instance running at [https://127.0.0.1:2379](https://127.0.0.1:2379/) saving the snapshot to **/data/backup/etcd-snapshot.db.**

The following TLS certificates/key are supplied for connecting to the server with etcdctl:

CA certificate: /opt/KUIN00601/ca.crt

Client certificate: /opt/KUIN00601/etcd-client.crt

Clientkey:/opt/KUIN00601/etcd-client.key

2- Next, restore an existing, previous snapshot located at **/var/lib/backup/etcd-snapshot-previous.db**

**Login into the master node and gain sudo access**

* **Ssh masternode-machine**
* **Sudo -i**
* **Insall etcd -client**
* ETCDCTL\_API=3 etcdctl --endpoints=https://127.0.0.1:2379 **\**

--cacert=/opt/KUIN00601/ca.crt \

--cert=/opt/KUIN00601/etcd-client.crt \

--key=/opt/KUIN00601/etcd-client.key **\**

snapshot save /data/backup/etcd-snapshot.db.

systemctl stop etcdctl

* Restoring the snapshot:

ETCDCTL\_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \

--cacert=/opt/KUIN00601/ca.crt \

--cert=/opt/KUIN00601/etcd-client.crt \

--key=</opt/KUIN00601/etcd-client.key> **\**

snapshot restore /var/lib/backup/etcd-snapshot-previous.db

rm -rf /var/lib/etcd/member/\*

cp -rvf \* /var/lib/etcd/member/

systemctl start etcd

systemctl enable etcd

**press exit- exit**

# 5 Create new NetworkPolicy /networking for searching/networkingpolicy resource

### **Tasks:**

1- Create a new NetworkPolicy named**allow-port-from-namespace** in the existing namespace **big-corp**

2- Ensure that the new NetworkPolicy allows Pods in namespace **internal** to connect to port **9200/tcp** of pods in namespace **big-corp**

3- Further ensure that the new NetworkPolicy

* Does not allow access to Pods which don’t listen on port **9200/tcp**
* Does not allow access from pods which are not in namespace **internal**

**For the practice**

* **Kubectl create namespace big-corp**
* **Kubectl create internal**

**In the exam**

* **Kubectl get namespace --show-labels ( here it will open the internal and big-corp label copy and paste on your vim file**
* **Vim NetworkPolicy.yaml**

**apiVersion: networking.k8s.io/v1**

**kind: NetworkPolicy**

**metadata: allow-port-from-namespace**

**name:**

**namespace: big-corp**

**spec:**

**podSelector: {}**

**policyTypes:**

**- Ingress**

**ingress:**

**- from:**

**- namespaceSelector:**

**matchLabels:**

**name: internal (area=two)**

**- podSelector: {}**

**ports:**

**- protocol: TCP**

**port: 9200**

* **K labels ns internal area=two**
* **K apply networkpolicy..yaml**

# **6** Reconfigure a deployment

### **Task:**

1) Reconfigure the existing deployment “**front-end**” and add a port specification named “**http**” exposing port **80/tcp** of the existing container “**nginx**“

2) Create a new service named **front-end-svc** exposing the container port “**http”**

3) Configure the new service to also expose individual pods via a **NodePort** on the nodes which

they are scheduled.

* **K create deployment front-end –image=nginx –repilas=1 > front-end.yml**
* **Vi front-end.yml**
* **Kubectl create –f front-end.yml**
* **Kubectlget deplyments**
* **1/Kubectl edit deployment front-end ( under specification name:nginx add containerPort: 80** (under containerport addname: http Protocol: tcp

-port: 80

Protocol: tcp

Name: http

* kubectl describe deployment front-end
* 2/Kubectl expose deploy front-end --name=front-end-svc -- port=80 --type=**nodeport**
* kubectl describe svc front-end-svc to check

# 7Create ingress resource

**Task:**

Create a new nginx “**ingress**“resource as follows:

* Name: **ping**
* Namespace: **internal**

Exposing service “**hello**“on path **/hello** using **service port** **5678**

* **Vim ping- ingress.yaml**

**apiVersion: networking.k8s.io/v1**

**kind: Ingress**

**metadata:**

**name: ping**

**namespace: internal**

**spec:**

**rules:**

**- http:**

**paths:**

**- path: /hello**

**pathType: Prefix**

**backend:**

**service:**

**name: hello**

**port:**

**number: 5678**

* kubectl create -f ping-ingress.yaml

# 8. Scale a deployment/ tape cheat sheet on kub.io “scalling resources”

# Scale the deployment front-end to 4 pods

* kubectl scale deployment front-end --replicas=4

# kubectl get deployment

# 9. Schedule a pod / ( nodeselector for searche /assign pods to nodes

**Task:**

 Schedule a pod as follows:

* Name: nginx-kusc007
* Image: nginx
* Node selector: disk=spinning
* vim nodeselector-pod.yml

apiVersion: v1

kind: Pod

metadata:

name: nginx-kusc007

labels:

env: test

spec:

containers:

- name: nginx

image: nginx

imagePullPolicy: IfNotPresent

nodeSelector:

disktype: spinning

* -save and run k create -f nodeselector-pod.yml
* Check where the pod will be scheduled to make sure it is placed on the labeled node
* kubectl get pods -o wide
* kubectl describe pod nginx-kusc007

# 10. Check how many nodes are ready

**Task:**

Check how many nodes are ready (not including tainted nodes) and write the number to **/some/file/path**

* kubectl get nodes to see the status of the nodes
* K describe node node01 | grep –i taint

**At the end, sum up the number of nodes that are ready and not tainted and paste the number in the file given**

echo number > /some/file/path

# 11 Create pod with name and images

Create a pod named “**kucc1**“ with a single app container for each of the following:

Images: **nginx + redis**

* Vim kucc1.yaml

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: kucc1

**spec**:

**containers**:

- **name**: nginx

**image**: nginx:1.14.2

- **name**: redis

**image**: redis

* Kubectl create –f kucc1.yaml

# 12Create a persistent volume

**Task:**

Create a persistent volume with name “**app-config**“ of capacity **2Gi** and access mode “**ReadOnlyMany**“. The type of volume is **HostPath** and its location is **/srv/app-confi**

* **Vim app-config.yaml go to persistent volume choose** [Configure a Pod to Use a **PersistentVolume** for Storage | Kubernetes](https://www.google.com/url?client=internal-element-cse&cx=013288817511911618469:elfqqbqldzg&q=https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/&sa=U&ved=2ahUKEwjDlNag0vv_AhV5SDABHbrHC9wQFnoECAkQAg&usg=AOvVaw3KupcXLNjgX9Ck9veTU2uT)

**apiVersion**: v1

**kind**: PersistentVolume

**metadata**:

**name**: app-config

**labels**:

**type**: local

**spec**:

**storageClassName**: manual

**capacity**:

**storage**: 2Gi

**accessModes**:

- ReadOnlyMany

**hostPath**:

**path**: "/srv/app-config"

* **Kubectl create –f app-config.yaml**

# 13 Create persistent volume claim/ search persistentvolumeclaim,then clike on persistent volume .scroll to persistentvolume claim down after phase

### **Task:**

1- Create a persistent volume claim

* Name: **pv-volume**
* Class: **csi-hostpath-sc**
* Capacity: **10Mii**
* **Vi pv-volume.yaml**

**apiVersion**: v1

**kind**: PersistentVolumeClaim

**metadata**:

**name**: pv-volume

**spec**:

**accessModes**:

- ReadWriteOnce

**volumeMode**: Filesystem

**resources**:

**requests**:

**storage**: 10Gi

**storageClassName**: csi-hostpath-sc

* Kubectl create –f pv-volume.yaml

2- Create a new pod which mounts the persistent volume claim as a volume/**tape persistent volume with pod on the search scroll on create a pod**.

* Name: **web-server**
* Image: **nginx**
* Mount Path: **/usr/share/nginx/html**
* Vi web-server.yaml

**apiVersion**: v1

**kind**: Pod

**metadata**:

**name**: web-server

**spec**:

**volumes**:

- **name**: pv-volume

**persistentVolumeClaim**:

**claimName**: pv-volume

**containers**:

- **name**: nginx

**image**: nginx

**ports**:

- **containerPort**: 80

**name**: "http-server"

**volumeMounts**:

- **mountPath**: "/usr/share/nginx/html"

**name**: pv-volume

* Kubectl create –f web-server.yaml

3- Configure the new pod to have **Readwriteonce** access to volume / nothing to do

4- Finally, use the kubectl edit or kubectl patch to expand the persistent volume claim to **70Mi** and record that change. Only in the exam this should be apply

* Kubectl edit pvc pv-volume --record=true

# 14Monitor the logs and extract some lines/ this will worked only on the exam

**Task:**

Monitor the logs of pod **foobar** and extract logs lines corresponding to error – **file-not-found** and write them to **/some/file/path**

* **Kubectl logs foobar | grep -i “file-not-found” > /some/file/path**

# 15 Adding a streaming sidecar container /search with sidecar then, logging architecture

1-An existing Pod needs to be integrated into the Kubernetes built-in logging architecture (e. g. kubectl logs). Adding a streaming sidecar container is a good and common way to accomplish this requirement.

* Kubectl get pod
* kubectl edit pod legacy-app

containers:

- name: sidecar

image: busybox

command: ["/bin/sh", "-c", "tail -n+1 /var/log/legacy-app.log"]

volumeMounts:

- name: logs

mountPath: /var/log/legacy-app.log

volumes:

- name: logs

emptyDir: {}

2-Add a sidecar container named sidecar, using the busybox Image, to the existing Pod big-corp-app. The new sidecar container has to run the following **command: /bin/sh -c tail -n+1 -f /var/log/big-corp-app.log**  
3-Use a Volume, mounted at **/var/log**, to make the log file **big-corp-app.log** available to the sidecar container.

Don’t modify the specification of the existing container other than adding the required volume mount.

# 16 11Find pods running high CPU workloads

**Task:**

From the pod label “**name = overload-cpu**”, find pods running high CPU workloads and write the name of the pod consuming most cpu to the file **/opt/kutr00401/kutr00401.txt** (Which already exists)

Kubectl top pods -l “ ”

Echo name > /opt…….

* **K get pod**
* **K top pod**
* Kubectl top pod –l “overload-cpu”
* Echo name > /opt/kutr00401/kutr00401.txt

# 17Troubleshoot worker node

A kubernetes worker node “wk8s-node-0“ is in state NotReady. Troubleshoot it and make it ready

* **Here, they have stopped the kubelet process on the specified machine. So all you have to do is to restart it. Here you will need to ssh to that node and start the kubelet. Also enable it.**
* kubectl get nodes
* ssh node1 (use the credentials they will give)
* sudo -i
* systemctl restart kubelet
* Systemctl enable kubelet
* systemctl status kubelet
* exit -exit