

Question 1:

Context :- `$ kubectl config use-context k8s`

You have been asked to create a new ClusterRole for a deployment pipeline and bind it to a specific ServiceAccount scoped to a specific namespace.

Task -

Create a new ClusterRole named deployment-clusterrole, which only allows to create the following resource types:

- Deployment
- Stateful Set
- DaemonSet

Create a new ServiceAccount named cicd-token in the existing namespace app-team1. Bind the new ClusterRole deployment-clusterrole to the new ServiceAccount cicd-token, limited to the namespace app-team1.

```
$ kubectl config use-context k8s
$ kubectl create clusterrole deployment-clusterrole --verb=create
--resource=deployment,statefulset,daemonset
$ kubectl get clusterrole | grep deployment-clusterrole

$ kubectl create serviceaccount cicd-token --namespace=app-team1
$ kubectl get serviceaccount -n app-team1 | grep "cicd-token"

$ kubectl create clusterrolebinding crb1 --clusterrole=deployment-clusterrole
--serviceaccount=app-team1:cicd-token
$ kubectl get clusterrolebinding | grep "crb1"
```

Question 2:

Context :- `$ kubectl config use-context ek8s`

Task -

Set the node named ek8s-node-0 as unavailable and reschedule all the pods running on it

```
$ kubectl config use-context ek8s
$ kubectl get nodes
$ kubectl drain node ek8s-node0 --ignore-daemonsets
$ kubectl get nodes
```

Question 3:

Context :- `$ kubectl config use-context mk8s`

Task -

Given an existing Kubernetes cluster running version 1.22.1, upgrade all of the Kubernetes control plane and node components on the master node only to version 1.22.2.

Be sure to drain the master node before upgrading it and uncordon it after the upgrade.

—————First we need to drain the node then ssh to node, update new version packages and restart kubelet services—————

```
$ kubectl config use-context mk8s
$ kubectl get nodes -o wide
$ kubectl version          #1.22.1
$ kubelet --version        #1.22.1
$ kubeadm version          #1.22.1

$ kubectl drain mk8s-master-0 --ignore-daemonsets
$ kubectl get nodes -o wide
$ ssh root@mk8s-master-0
root@master~# apt update
root@master~# apt install kubeadm=1.22.2 kubelet=1.22.2 kubectl=1.22.2
root@master~# systemctl restart kubelet
root@master~# systemctl status kubelet

EXIT
```

```
$ kubectl uncordon mk8s-master-0
$ kubectl get nodes -o wide
$ kubectl describe node mk8s-master-0
$ kubectl version          #1.22.2
$ kubelet --version        #1.22.2
$ kubeadm version          #1.22.2
$
```

Question 4:

Context :- **\$ kubectl config use-context ek8s**

Task -

First, create a snapshot of the existing etcd instance running at <https://127.0.0.1:2379>, saving the snapshot to `/var/lib/backup/etcd-snapshot.db`.

- CA Certificate= `/opt/KUIN00601/ca.crt`
- Client Certificate= `/opt/KUIN00601/etcd-client.crt`
- Client Key= `/opt/KUIN00601/etcd-client.key`

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

```
$ kubectl config use-context ek8s
```

----- Create a snapshot of etcd instance -----

```
$ 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 /var/lib/backup/etcd-snap
shot.db
```

```
$ 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 status /var/lib/backup/etcd-  
snapshot.db
```

----- Restore existing previous snapshot of etcd -----

```
$ ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 snapshot restore  
/var/lib/backup/etcd-snapshot-previous.db
```

Question 5:

Context :- **\$ kubectl config use-context hk8s**

Task -

Create a new NetworkPolicy named allow-port-from-namespace in the existing namespace fubar.

Ensure that the new NetworkPolicy allows Pods in namespace internal to connect to port 9000 of Pods in namespace fubar.

→ Further ensure that the new NetworkPolicy:

- ◆ does not allow access to Pods, which don't listen on port 9000
- ◆ does not allow access from Pods, which are not in namespace internal

```
$ kubectl config use-context hk8s  
$ kubectl label namespace fubar project=namespace # first we create a label on  
namespace, so we can select namespace as per label  
$ vi network-policy.yml
```

```
apiVersion: networking.k8s.io/v1  
kind: NetworkPolicy  
metadata:  
  name: allow-port-from-namespace  
  namespace: fubar  
spec:  
  podSelector: { }  
  policyTypes:  
    - Ingress  
  ingress:  
    - namespaceSelector:  
        matchLabels:  
          project: myproject  
    - podSelector:  
        matchLabels:  
          role: frontend  
  ports:  
    - protocol: TCP  
      port: 9000
```

```
$ kubectl create -f network-policy.yml
$ kubectl get networkpolicy -n fubar
```

Question 6:

Context :- **\$ kubectl config use-context k8s**

Task -

Reconfigure the existing deployment front-end and add a port specification named http exposing port 80/tcp of the existing container nginx.

Create a new service named front-end-svc exposing the container port http.

Configure the new service to also expose the individual Pods via a NodePort on the nodes on which they are scheduled.

```
$ kubectl config use-context k8s
$ kubectl get deployment
$ kubectl get deployment -o yaml > front-end.yml
$ vi deployment.yml
```

----- Add port section under container section -----

```
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
      name: http
```

Save and exit

```
$ kubectl apply -f front-end.yml
$ kubectl get deployment
$ kubectl describe deployment front-end
```

----- Create a service to expose deployment front-end NodePort -----

```
$ kubectl expose deployment front-end --name=front-end-svc --port=80
--target-port=80 --type=NodePort
$ kubectl get svc
$ kubectl describe svc front-end-svc
$ curl http://<front-end-svc-ip>:portnumber
```

Question 7:

Context :- **\$ kubectl config use-context k8s**

Task -

Scale the deployment presentation to 3 pods.

```
$ kubectl config use-context k8s
$ kubectl get deployment
$ kubectl get pods
$ kubectl scale --replicas=3 deployment presentation
$ kubectl get deployment
$ kubectl get pods
```

Question 8:

Context :- **\$ kubectl config use-context ek8s**

Task -

Schedule a pod as follows:

- Name: nginx-kusc00401
- Image: nginx
- NodeSelector: disk=ssd

```
$ kubectl config use-context ek8s
$ vi pod.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx-kusc00401
spec:
  nodeSelector:
    disk: ssd
  containers:
  - name: nginx
    image: nginx

$ kubectl create -f pod.yml
$ kubectl get pods
```

Question 9:

Context :- **\$ kubectl config use-context ek8s**

Task -

Check to see how many nodes are ready (not including nodes tainted NoSchedule) and write the number to /opt/KUSC00402/kusc00402.txt.

—We will confirm how many nodes are restricted because of Taint restriction—

```
$ kubectl config use-context ek8s
$ kubectl describe nodes | grep "Taint"
```

----- Count the number of ready nodes -----

```
$ echo '2' > /opt/KUSC00402/kusc00402.txt
$ cat /opt/KUSC00402/kusc00402.txt
```

Question 10:

Context :- **\$ kubectl config use-context k8s**

Task -

Schedule a Pod as follows:

- Name: kucc8
- App Containers: 2
- Container Name/Images:
 - nginx
 - consul

```
$ kubectl config use-context k8s
$ vi pod.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: kucc8
spec:
  containers:
  - name: nginx
    image: nginx
  - name: consul
    image: consul
```

```
$ kubectl create -f pod.yml
$ kubectl get pods
```

Question 11:

Context :- **\$ kubectl config use-context hk8s**

Task -

Create a persistent volume with name app-data, of capacity 2Gi and access mode ReadOnlyMany. The type of volume is hostPath and its location is /srv/app- data.

```
$ kubectl config use-context hk8s
$ vi pv.yml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: app-data
spec:
  capacity:
    storage: 2Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  storageClassName: slow
  hostPath:
    path: /srv/app-data
```

```
$ kubectl create -f pv.yml
$ kubectl get pv
$ kubectl describe pv app-data
```

Question 12:

Context :- **\$ kubectl config use-context k8s**

Task -

Monitor the logs of pod foo and:

- Extract log lines corresponding to error file-not-found
- Write them to /opt/KUTR00101/foo

```
$ kubectl config use-context k8s

$ kubectl get pods
$ kubectl logs foo | grep "file-not-found"
$ kubectl logs foo | grep "file-not-found" > /opt/KUTR00101/foo
$ cat /opt/KUTR00101/foo
```

Question 13:

Set Context :- **\$ kubectl config use-context k8s**

Note Context - 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.

Task -

- 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 "
- Use a Volume, mounted at /var/log, to make the log file big-corp-app.log available to the sidecar container.

```
$ kubectl config use-context k8s
$ kubectl get pods
$ kubectl get pod existing-pod1 -o yaml > pod.yml
----- or you can use '< $ kubectl edit pod existing-pod >' directly-----
$ vi pod.yml

apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
  sidecarContainers:
  - name: sidecar
    image: busybox
    command: ["/bin/sh", "-c", "tail -n+1 -f /var/log/big-corp-app.log "]
  volumeMounts:
  - mountPath: /var/log
    name: test-volume
  volumes:
  - name: log-volume
    emptyDir: { }

$ kubectl create -f pod.yml
$ kubectl get pods
```

Question 14:


Context :- **\$ kubectl config use-context ek8s**

Task -

From the pod label name=overloaded-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 config use-context k8s
----- when we use --sort-by=cpu, --no-headers=true output will be -----
$ kubectl top pods -l name=overloaded-cpu --sort-by=cpu --no-headers=true
```


When you use both the `--no-headers=true` and `--sort-by=cpu` flags with the `kubectl top pod` command, the sample output will look like this:

 Copy code


```
overloaded-pod-2      800m      512Mi
overloaded-pod-1      500m      256Mi
overloaded-pod-3      100m      128Mi
```

----- when we don't use `--sort-by=cpu`, `--no-headers=true` output will be -----

```
$ kubectl top pods -l name=overloaded-cpu
```

Certainly! Here's an example sample output of the `kubectl top pod` command when not using the `--no-headers=true` and `--sort-by=cpu` flags:

SCSS

 Copy code

```
NAME                  CPU(cores)   MEMORY(bytes)
overloaded-pod-1      500m         256Mi
overloaded-pod-2      800m         512Mi
overloaded-pod-3      100m         128Mi
```

----- so better will be, to check and save name manually-----

```
$ echo "overloaded-pod-1" > /opt/KUTR00401/KUTR00401.txt
```

Question 15:

Context :- `$ kubectl config use-context wk8s`

Task -

A Kubernetes worker node, named `wk8s-node-0` is in state `NotReady`. Investigate why this is the case, and perform any appropriate steps to bring the node to a `Ready` state, ensuring that any changes are made permanent.

```
$ kubectl config use-context wk8s
```

```
$ kubectl get nodes
```

```
$ kubectl describe node wk8s-node-0
```

----- Kubelet service is stopped in node '`wk8s-node-0`' so restart and enable the service for permanent solution -----

```
$ ssh wk8s-node-0
$ sudo -i

root@master~# systemctl status kubelet.service
root@master~# systemctl restart kubelet.service
root@master~# systemctl enable kubelet.service
root@master~# systemctl status kubelet.service

EXIT

$ kubectl get nodes
```

Question 16:

Context :- **\$ kubectl config use-context ek8s**

Task -

- ❖ Create a new PersistentVolumeClaim:
 - Name: pv-volume
 - Class: csi-hostpath-sc
 - Capacity: 10Mi
- ❖ Create a new Pod which mounts the PersistentVolumeClaim as a volume:
 - Name: web-server
 - Image: nginx
 - Mount path: /usr/share/nginx/html
- Configure the new Pod to have ReadWriteOnce access on the volume.
- Finally, using kubectl edit or kubectl patch expand the PersistentVolumeClaim to a capacity of 70Mi and record that change.

```
$ kubectl config use-context ok8s
```

-----**(Step-1) First we will create PVC with given info in question**-----

```
$ vi pvc.yml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-volume
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
resources:
  requests:
```

```
storage: 10Mi
storageClassName: csi-hostpath-sc
```

```
$ kubectl create -f pvc.yml
$ kubectl get pvc | grep 'pv-volume'
```

—(Step-2) Now after PVC creation, we create a pod and use PVC as a volume—--

```
$ vi pod.yml

apiVersion: v1
kind: Pod
metadata:
  name: web-server
spec:
  containers:
    - name: my-pod
      image: nginx
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: myvol
  volumes:
    - name: myvol
      persistentVolumeClaim:
        claimName: pv-volume

$ kubectl create -f pod.yml
$ kubectl get pods
$ kubectl describe pod web-server
```

----- (Step-3) Now we will extend the PVC volume and apply changes -----

```
$ vi pvc.yml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-volume
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 70Mi  # extent volume amount from 10Mi to 70Mi in PVC
  storageClassName: csi-hostpath-sc
```

```
$ kubectl apply -f pvc.yml
$ kubectl get pvc
$ kubectl describe pods
```

Question 17:

Context :- **\$ kubectl config use-context k8s**

Task -

Create a new nginx Ingress resource as follows:

- Name: pong
- Namespace: ing-internal
- Exposing service hello on path /hello using service port 5678

```
$ kubectl config use-context k8s
$ vi ingress.yml
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: pong
  namespace: ing-internal
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /hello
        pathType: Prefix
        backend:
          service:
            name: hello
            port:
              number: 5678
```
