Question 1: Set the node node1.example.com as unavailable and reschedule all the pods running on it.

```
$ kubectl get nodes
$ kubectl drain node1.example.com --ignore-daemonsets
```

Question 2: Create a snapshot of the existing etcd instance running at https://127.0.0.1:2379, saving the snapshot to /srv/data/demo-snapshot.db

- CA certificate: /opt/pki/ca/crt
- Client Certificate: /opt/pki/client/etcd-client.crt
- Client Key: /opt/pki/etcd-client.key

Next restore an existing , previous snapshot located at /srv/data/etcd-previoussnapshot.db

```
—------- Create a snapshot of etcd —------------

$ ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379
--cacert=/opt/pki/ca/crt --crt=/opt/pki/client/etcd-client.crt
--key=/opt/pki/etcd-client.key snapshot save /srv/data/demo- snapshot.db

—-------- Restore etcd backup —----------------

$ ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 snapshot restore
/srv/data/etcd-previoussnapshot.db
```

Question 3: Create a network policy named "allow-port" in the fubar namespace. Ensure that the new network policy allows pods in namespace project=corp-net to connect to port 9200 of pods in namespace fubar.
Further ensure that the new network policy

- Does not allow access to pods, which do not listen on 9200
- Does not allow access from pods, which are not in namespace corp-net.

Note: Need help :
https://kubernetes.io/docs/concepts/services-networking/network-policies/

```
$ vi network-policy.yml

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-port
  namespace: fubar
spec:
  podSelector: { }
  policyTypes:
```

```
  - Ingress
 ingress:
  - from:
     - namespaceSelector:
        matchLabels:
          project: corp-net
    ports:
     - protocol: TCP
       port: 9200

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

Question 4: 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.

```
——---- Reconfigure the existing deployment——-------------

$ kubectl get deploy
$ kubectl edit deploy front-end
   spec:
     containers:
     - image: nginx
       imagePullPolicy: Always
       name: nginx
       ports:
       - containerPort: 80
         name: http
         protocol: TCP

——----------- Create a Service Named front-end-svc——----------

$ kubectl get deploy
$ kubectl expose deploy front-end --port=80 --target-port=80 --type=NodePort
--name=front-end-svc
$ kubectl get pods -o wide
$ kubectl get svc
$ curl http://< front-end-svc-ip >
```

Question 5: Create a new nginx ingress resource as follows:
Name: pong, Namespace: ing-internal
Exposing service hello on the path /hello using service port 5678

Note: Need help:
https://kubernetes.io/docs/concepts/services-networking/ingress/

```
$ vi ingress.yml

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

$ kubectl create -f ingress.yml
$ kubectl get ingress -n ing-internal
```

Question 6: Scale the deployment presentation to 5 pods.

```
$ kubectl get deployment
$ kubectl describe deployment presentation
$ kubectl scale --replicas=5 deployment presentation
$ kubectl get deploy
```

Question 7: Schedule a pod as follows:

- Name: nginx-prod
- Image: nginx
- Node selector: disk=spinning

```
$ kubectl get nodes --show-labels | grep "disk"

$ vi pod.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-prod
spec:
  nodeSelector:
      disk: spinning
  containers:
  - name: nginx
    image: nginx

$ kubectl create -f pod.yml

$ kubectl get pods -o wide
```

Question 8: Check to see how many nodes are ready (not including nodes tainted NoScheduling)
and write the number in /opt/kubenetes/nodes.txt

```
—------- We need only ready nodes, not tainted nodes—---------

$ kubectl describe nodes | grep "Taint"

Note: Check numbers of nodes in ready state

$ echo '2' >  /opt/kubenetes/nodes.txt
$ cat /opt/kubenetes/nodes.txt
```

Question 9: Create a pod named kucc8 with a single app container for each of the following images running inside: nginx+redis

```
$ vi pod.yml

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

$ kubectl create -f pod.yml
```

```
$ kubectl get pods
```

Question 10: Create a persistent volume with name app-config , of capacity 1Gi and access mode ReadOnlyMany, the type of volume is hostPath and its location is /srv/app-config

Need Help: https://kubernetes.io/docs/concepts/storage/persistent-volumes/

```
$ vi pv.yml

apiVersion: v1
kind: PersistentVolume
metadata:
 name: appconfig
spec:
 capacity:
   storage: 1Gi
 volumeMode: Filesystem
 accessModes:
   - ReadWriteMany
 storageClassName: slow
 hostPath:
   path: /srv/app-config


$ kubectl create -f pv.yml
$ kubectl get pv
$ kubetcl describe pv appconfig
```

Question 11: Create a 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' expand the PersistentVolumeClaim to a capacity of 70Mi and record that change.

```
——————————— Create a PVC named "pv-volume" ——————————————————

$ vi pvc.yml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvvolume
spec:
 accessModes:
  - ReadWriteOnce # As mentioned:  PVC mount in a pod  with RWO volumeMode:
Filesystem
 resources:
  requests:
    storage: 10Mi
 storageClassName: csi-hostpath-sc


$ kubectl create -f pvc.yml
$ kubectl get pvc

——-------- Create a pod and mount pvc with RWO mode  ---------------

$ vi pod.yml

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


$ kubectl create -f pod.yml
$ kubectl describe pod web-server

——---------------- Expend PVC Capacity to 70Mi ——------------------

$ kubectl get pvc
$ kubectl describe pvc pvvolume
$ vi pvc.yml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: pvvolume
```

```
spec:
 accessModes:
  - ReadWriteOnce # As mentioned:  PVC mount in a pod  with RWO volumeMode:
Filesystem
 resources:
  requests:
    storage: 70Mi
 storageClassName: csi-hostpath-sc


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

Question 12: Monitor the logs of pod foobar and:
Extract log lines corresponding to error file-not-found
Write them to /opt/kutr/foobar

```
$ kubectl get pods
$ kubectl logs foobar | grep 'file-not-found' > /opt/kutr/foobar
```

Question 13: From the pod label 'overloaded-cpu', find pods running high CPU workload and write the name of the pod consuming most CPU to the file /opt/kutr0401.txt

```
—------ 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, –bo-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 14: A kubernetes worker node name worker1.example .com is in state NotReady. Investigate why this is the case & perform any appropriate steps to bring the node to a Ready state, ensuring that any changes are made permanent.
[you can ssh to the failed node]

—------ Because of kubelet service down, node worker1.example.com is in state NotReady, We need to restart and enable service —-------

$ kubectl get nodes -o wide
$ kubectl describe node worker1.example.com
$ ssh root@worker1.example.com
root@worker1 ~# systemctl status kubelet
root@worker1 ~# systemctl restart kubelet
root@worker1 ~# systemctl enable kubelet
root@worker1 ~# systemctl status kubelet
EXIT

$ kubectl get nodes -o wide

Question 15: Create a service account named cicd-token in 'app-team1' namespace.

- Create a clusterrole which allow to only create resources Deployment, DaemonSet & statefulSet.
- Bind that clusterrole with the service account 'cicd-token' create in 'app-team1' namespace.

—---------- Create a service account—----------

```
$ kubectl create serviceaccount cicd-token --namespace= app-team1

$ kubectl get serviceaccount
$ kubectl get serviceaccount -n app-team1 | grep cicd-token

——----------------- Create a clusterrole ——----------------------------

$ kubectl create clusterrole clustername --verb=create
--resource=deployment,daemonset,statefulset

$ kubectl get clusterrole | grep clustername

——---------------Create a clusterrolebinding——----------------------
$ kubectl create clusterrolebinding --help

$  kubectl create clusterrolebinding crbname1 --clusterrole=clustername
--serviceaccount=app-team1:cicd-token

$ kubectl get clusterrolebinding | grep crbname1
```

Question 16: Upgrade the kubeadm version from 1.22.1 to 1.22.2 along with kubectl & kubelet only on master node (Never update anything on worker nodes.)

```
--First we will check versions, ssh the node, drain the master node, update new
packages, restart, enable kubelet services--

$ kubeadm version
$ kubectl version
$ kubelet --version


$ kubectl drain masternode --ignore-daemonsets
$ ssh root@masternode
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 enable kubelet
EXIT
$ kubectl version
$ kubelet --version
$ kubeadm version

$ kubectl uncordon masternode
$ kubectl get nodes
```

Question 17: Create a deployment named ku8s-deploy with httpd image & upgrade the version of this deployment.

```
—---------------- Create a deployment with httpd:2.4 version

$ kubectl create deploy  --help | less
$ kubectl create deployment ku8s-deploy --image=httpd:2.4 --replicas=3
$ kubectl get deploy
$ kubectl describe deploy ku8s-deploy

—----------------- Update httpd image with latest version
$ kubectl set image --help | less
$ kubectl set image deployment ku8s-deploy httpd=httpd:latest
$ kubectl describe deployment ku8s-deploy

—-----------------------------------------------------------------------------
—---------------Or you can use thismethod—------------------------
—-----------------------------------------------------------------------------



—----------------- Create a deployment
$ vi deployment.yml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ku8s-deploy
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: httpd
        image: httpd:2.4


$ kubectl create -f deployment.yml
$ kubectl get pods
$ kubectl get deploy

—---------------- upgrade image version—------------------------
```

```
$ vi deployment.yml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ku8s-deploy
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: httpd
        image: httpd:latest


$ kubectl apply -f deployment.yml
$ kubectl describe deploy ku8s-deploy
$ kubectl get deploy
```