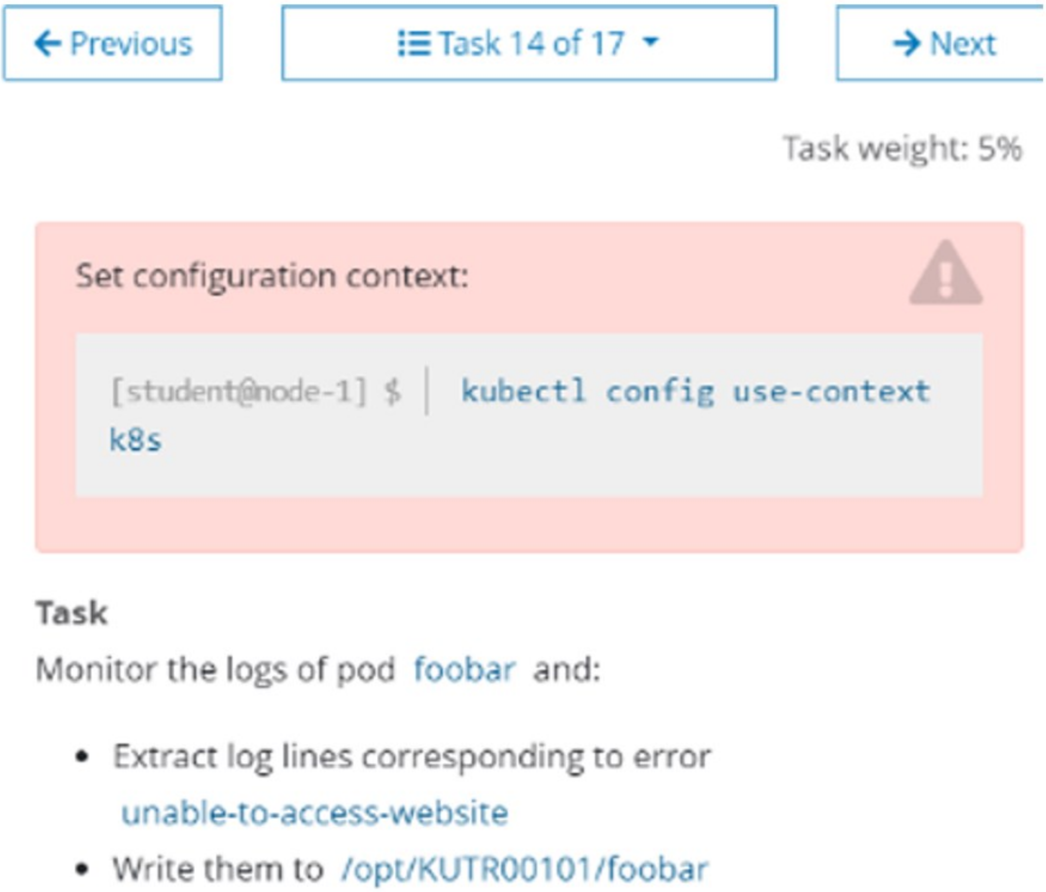
考题由简单到难排列

Kubectl 命令大全 https://kubernetes.io/zh/docs/reference/kubectl/cheatsheet/

考试时先挑简单的做~~~

1.1 监控 Pod 日志



Task weight: 5%

Set configuration context:

```
[student@node-1] $ | kubectl config use-context
k8s
```

Task

Monitor the logs of pod foobar and:

- Extract log lines corresponding to error unable-to-access-website
- Write them to /opt/KUTR00101/foobar

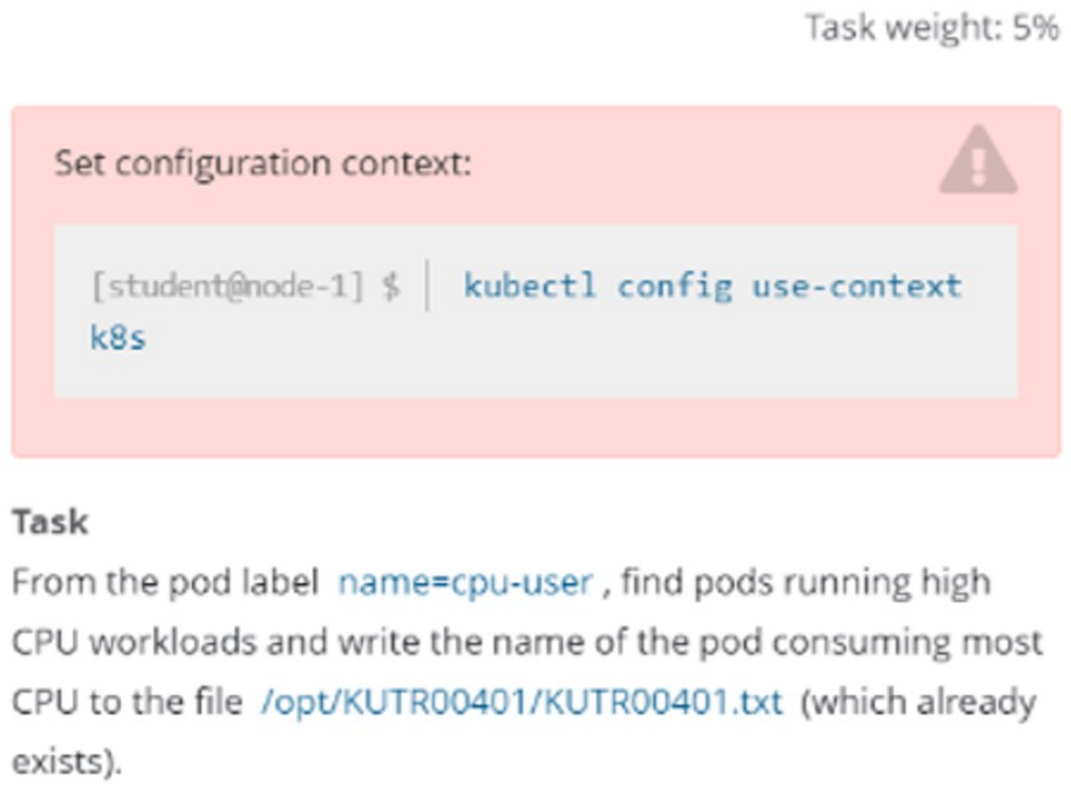
中文解释：

监控名为 foobar 的 Pod 的日志，并过滤出具有 unable-access-website 信息的行，然后将写入到 /opt/KUTR00101/foobar

解题：

```
    $ kubectl config use-context k8s
    $ kubectl logs foobar | grep unable-access-website >
/opt/KUTR00101/foobar
```

1.2 监控 Pod 度量指标



Task weight: 5%

Set configuration context:

```
[student@node-1] $ | kubectl config use-context
k8s
```

Task

From the pod label name=cpu-user , 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).

中文解释：

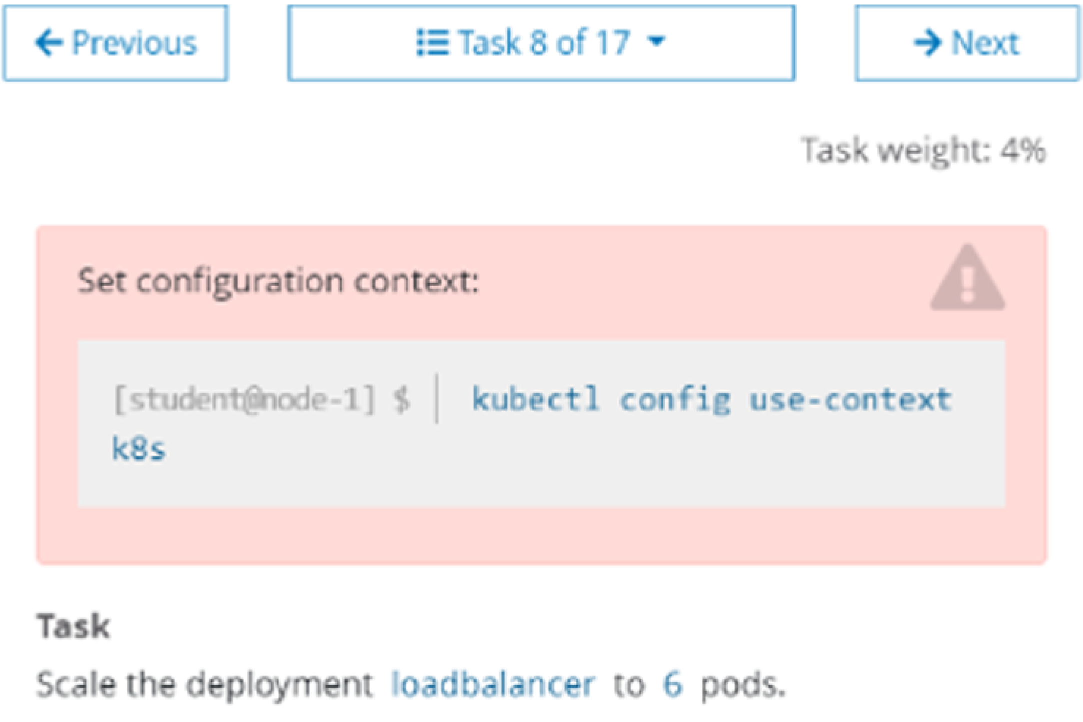
找出具有 name=cpu-user 的 Pod，并过滤出使用 CPU 最高的 Pod，然后把它的名字写在已经存在的/opt/KUTR00401/KUTR00401.txt 文件里（注意他没有说指定 namespace。所以需要使用-A 指定所以 namespace）

解题：

```
    $ kubectl config use-context k8s
    $ kubectl  top po -A -l name=cpu-user
    NAMESPACE     NAME                        CPU(cores)   MEMORY(bytes)
    kube-system   coredns-54d67798b7-hl8xc    7m           8Mi
    kube-system   coredns-54d67798b7-m4m2q    6m           8Mi
    # 注意这里的 pod 名字以实际名字为准，按照 CPU 那一列进行选择一个最大的 Pod，另外如果
CPU 的数值是 1  2  3 这样的。是大于带 m 这样的，因为 1 颗 CPU 等于 1000m，注意要用>>而不是>
```

```
    $ echo "coredns-54d67798b7-hl8xc" >> /opt/KUTR00401/KUTR00401.txt
```

1.3 Deployment 扩缩容



Task weight: 4%

Set configuration context:

```
[student@node-1] $ kubectl config use-context k8s
```

Task

Scale the deployment loadbalancer to 6 pods.

中文解释：

扩容名字为 loadbalancer 的 deployment 的副本数为 6

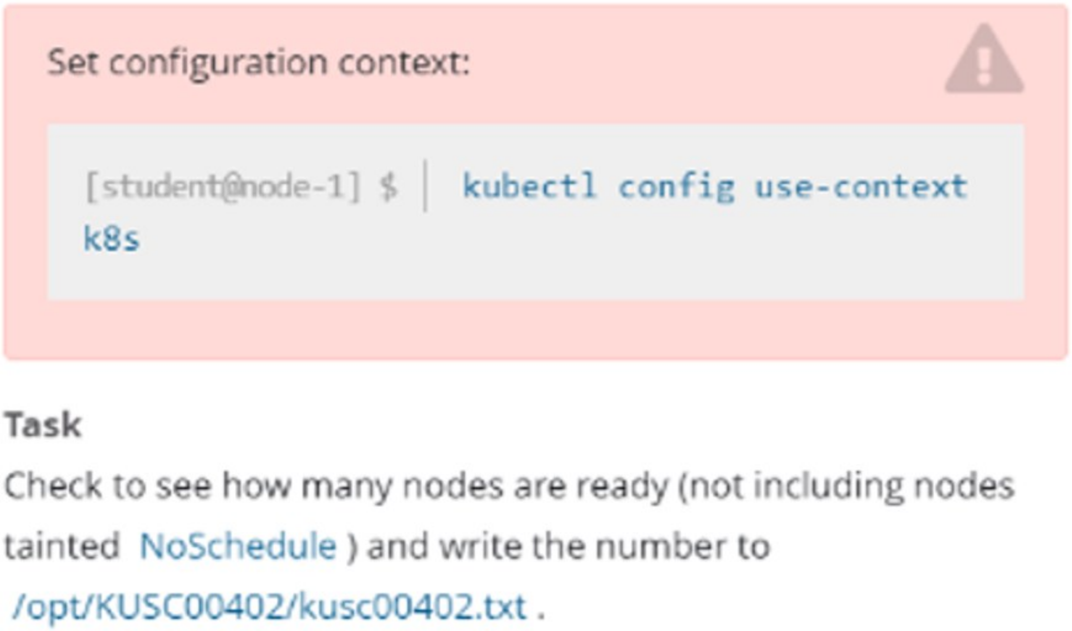
解题：

```
$ kubectl config use-context k8s

$ kubectl scale --replicas=6 deployment loadbalancer
```

$ kubectl edit

1.4 检查 Node 节点的健康状态



Set configuration context:

```
[student@node-1] $ | kubectl config use-context
k8s
```

Task

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

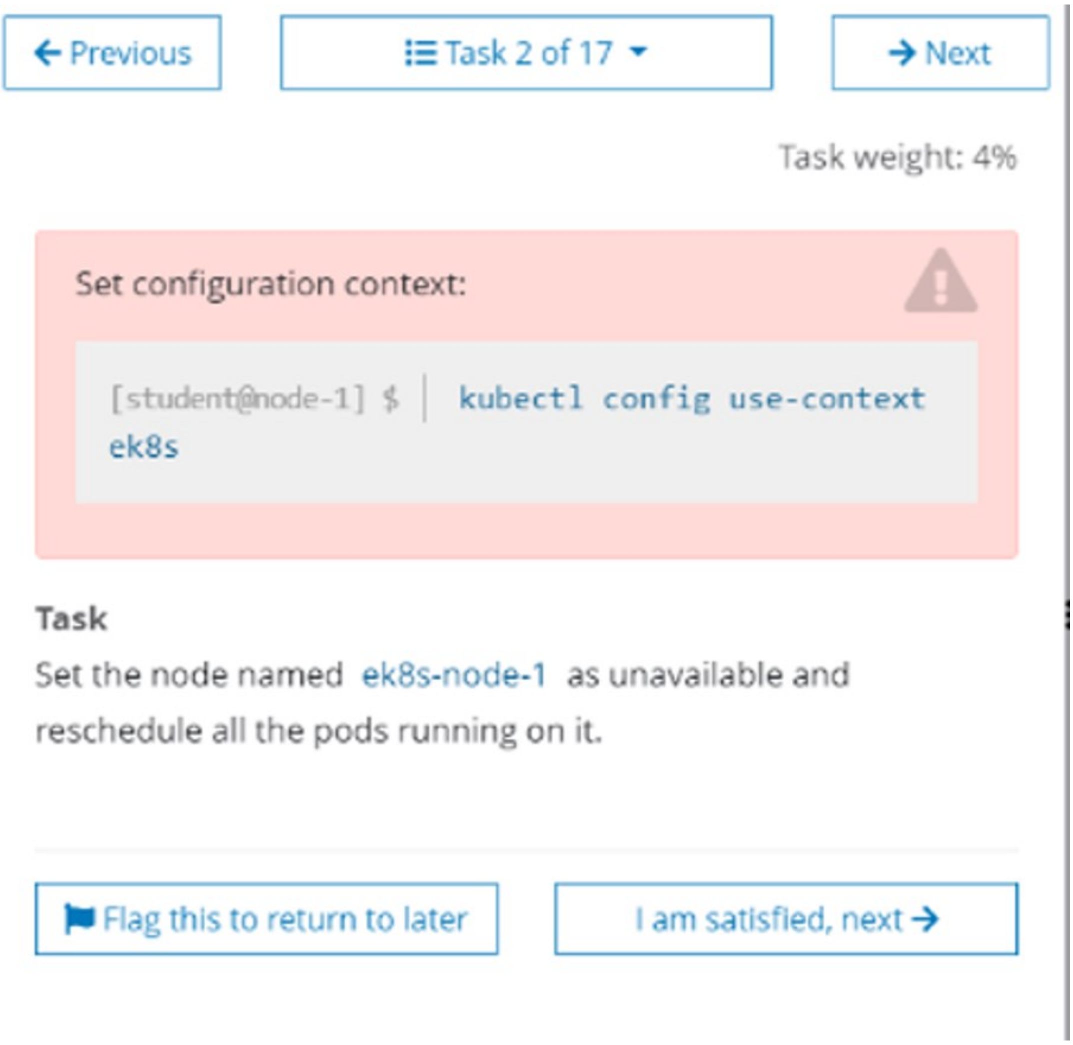
中文解释：

检查集群中有多少节点为 Ready 状态，并且去除包含 NoSchedule 污点的节点。之后将数字写到/opt/KUSC00402/kusc00402.txt

解题：

```
$ kubectl config use-context k8s
$ kubectl get node | grep -i ready   # 记录总数为 A
$ kubectl  describe node | grep Taint | grep NoSchedule  # 记录总数为 B
# 将 A 减 B 的值 x 导入到/opt/KUSC00402/kusc00402.txt
$ echo x >> /opt/KUSC00402/kusc00402.txt
```

1.5 节点维护



Task weight: 4%

Set configuration context:

```
[student@node-1] $ | kubectl config use-context
ek8s
```

Task

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

中文解释：

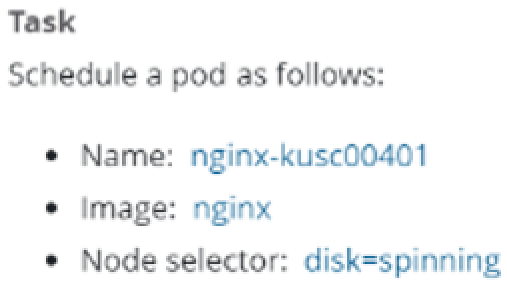
将 ek8s-node-1 节点设置为不可用，然后重新调度该节点上的所有 Pod

解题：

```
    $ kubectl config use-context ek8s
    $ kubectl cordon ek8s-node-1
    $ kubectl drain ek8s-node-1 --delete-emptydir-data --ignore-daemonsets -
-force
```

https://kubernetes.io/zh/docs/tasks/configure-pod-container/

1.6 指定节点部署



Task

Schedule a pod as follows:

- Name: nginx-kusc00401
- Image: nginx
- Node selector: disk=spinning

中文解释：

创建一个 Pod，名字为 nginx-kusc00401，镜像地址是 nginx，调度到具有 disk=spinning 标签的节点上，该题可以参考链接：https://kubernetes.io/zh/docs/concepts/scheduling-eviction/assign-pod-node/

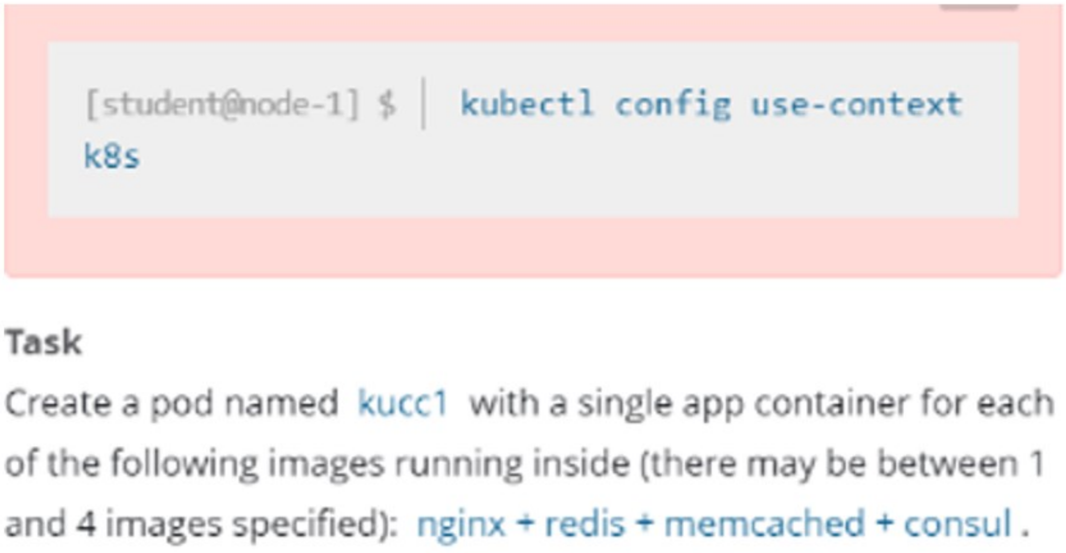
参考：

https://kubernetes.io/zh/docs/tasks/configure-pod-container/assign-pods-nodes/

解题：

```
$ vim pod-ns.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-kusc00401
  labels:
    role: nginx-kusc00401
spec:
  nodeSelector:
    disk: spinning
  containers:
    - name: nginx
      image: nginx
$ kubectl create -f pod-ns.yaml
```

1.7 一个 Pod 多个容器



```
[student@node-1] $ | kubectl config use-context
k8s
```

Task

Create a pod named kucc1 with a single app container for each of the following images running inside (there may be between 1 and 4 images specified): nginx + redis + memcached + consul .

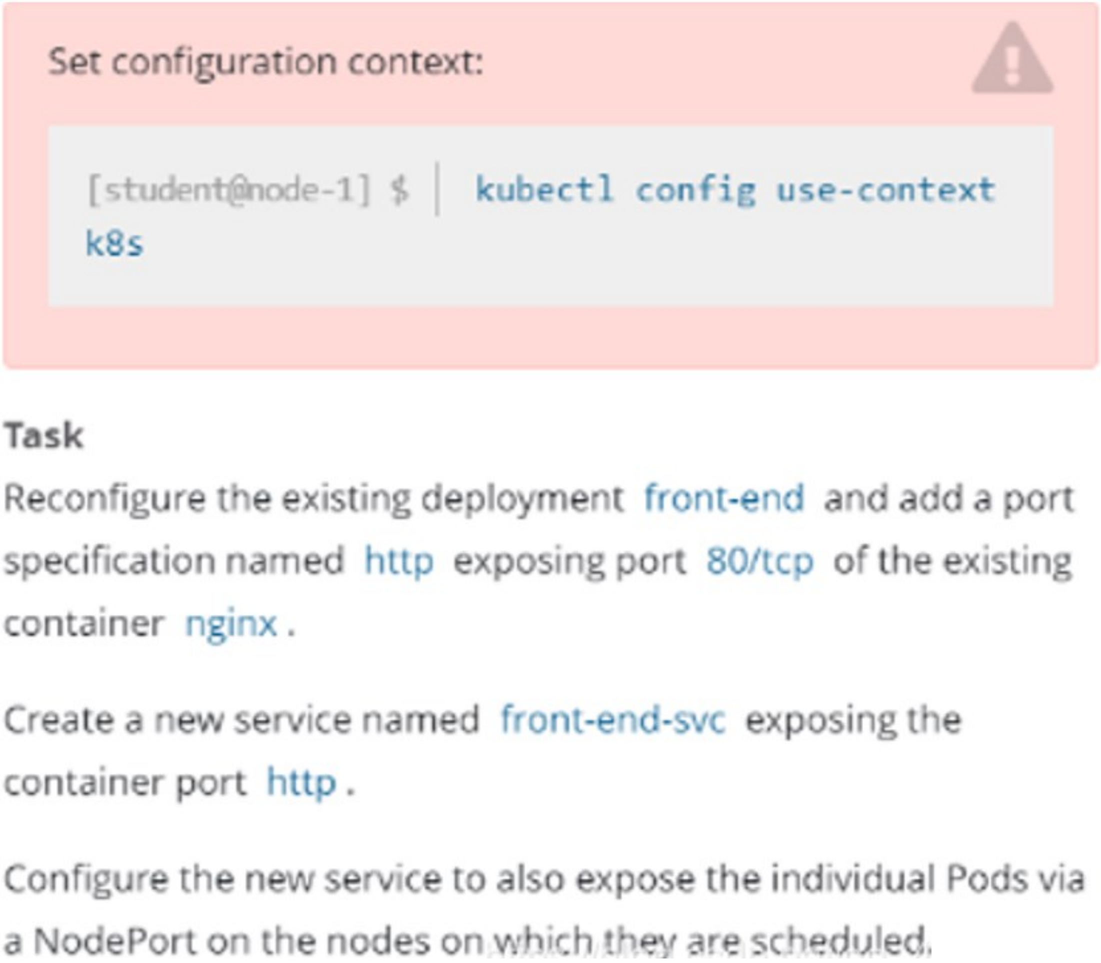
中文解释：

创建一个 Pod，名字为 kucc1，这个 Pod 可能包含 1-4 容器，该题为四个：nginx+redis+memcached+consul

解题：

```
apiVersion: v1
kind: Pod
metadata:
  name: kucc1
spec:
  containers:
  - image: nginx
    name: nginx
  - image: redis
    name: redis
  - image: memchached
    name: memcached
  - image: consul
    name: consul
```

1.8 Service



Set configuration context:

```
[student@node-1] $ | 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.

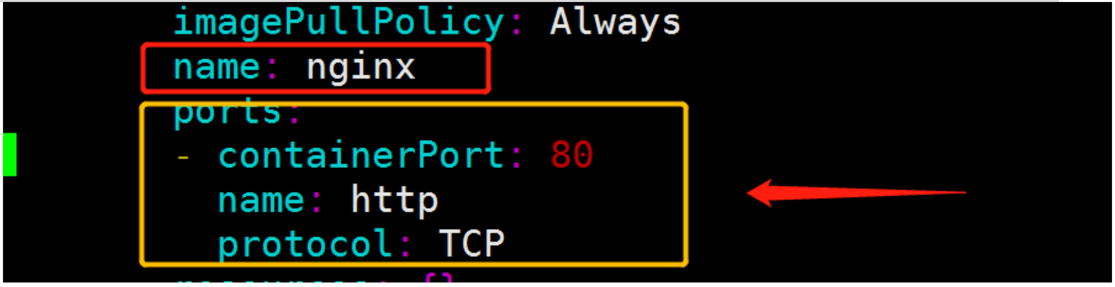
中文解释：

重新配置一个已经存在的 deployment front-end，在名字为 nginx 的容器里面添加一个端口配置，名字为 http，暴露端口号为 80，然后创建一个 service，名字为 front-end-svc，暴露该 deployment 的 http 端口，并且 service 的类型为 NodePort。

解题：

本题可以参考：https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/

```
$ kubectl edit deploy front-end
# 添加如下配置，主要是在 name 为 nginx 的容器下
```



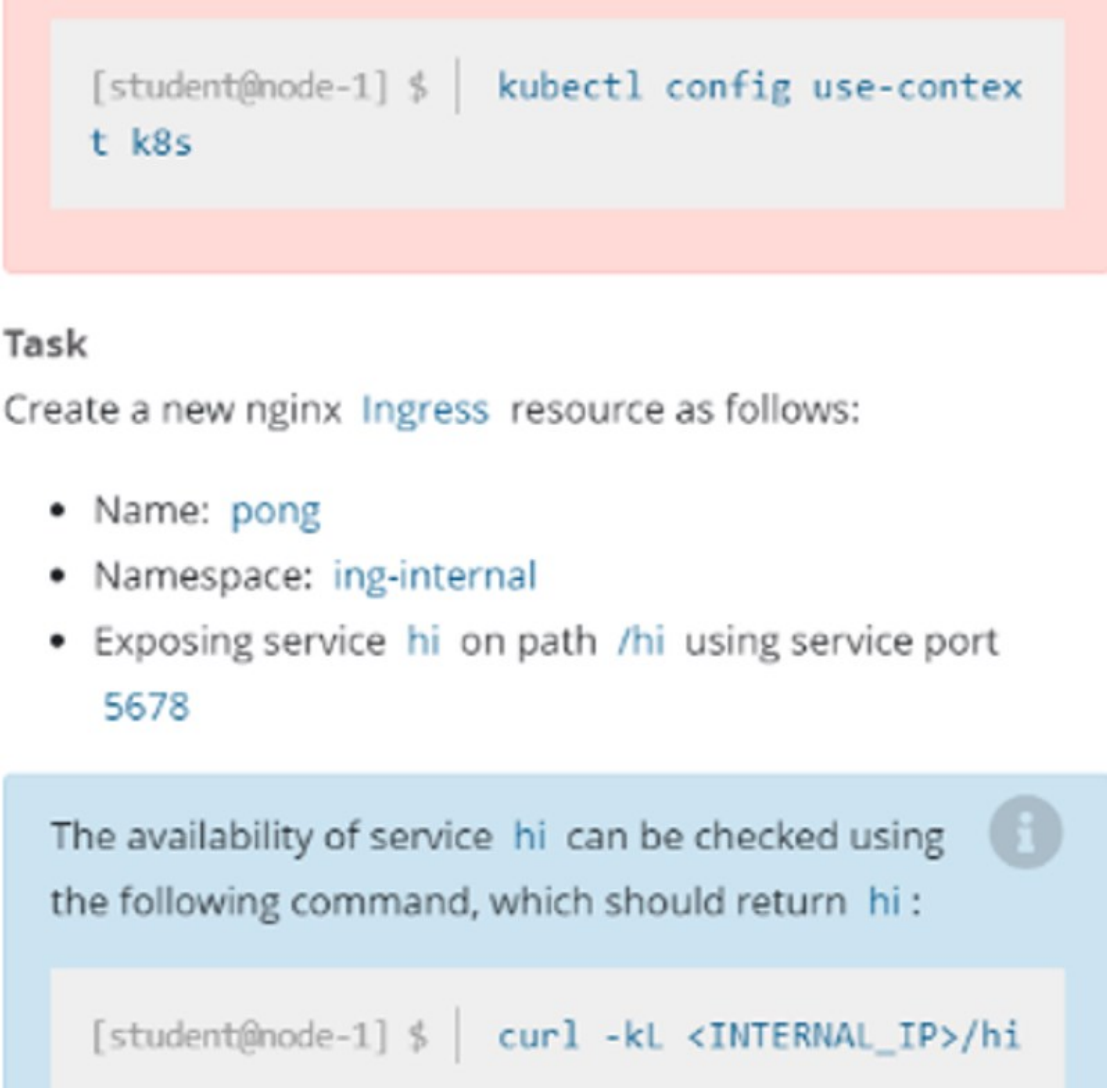
```
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 80
          name: http
          protocol: TCP
        resources: {}
```

添加 service：

```
$
```

```
    kubectl expose deploy front-end --name=front-end-svc  --port=80 --target-port=http --type=NodePort
```

1.9 Ingress



```
[student@node-1] $ | kubectl config use-context k8s
```

Task

Create a new nginx Ingress resource as follows:

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

The availability of service hi can be checked using the following command, which should return hi :

```
[student@node-1] $ | curl -kL <INTERNAL_IP>/hi
```

中文解释：

在 ing-internal 命名空间下创建一个 ingress，名字为 pong，代理的 service hi，端口为 5678，配置路径/hi。

验证：访问 curl -kL <INTERNAL_IP>/hi 会返回 hi

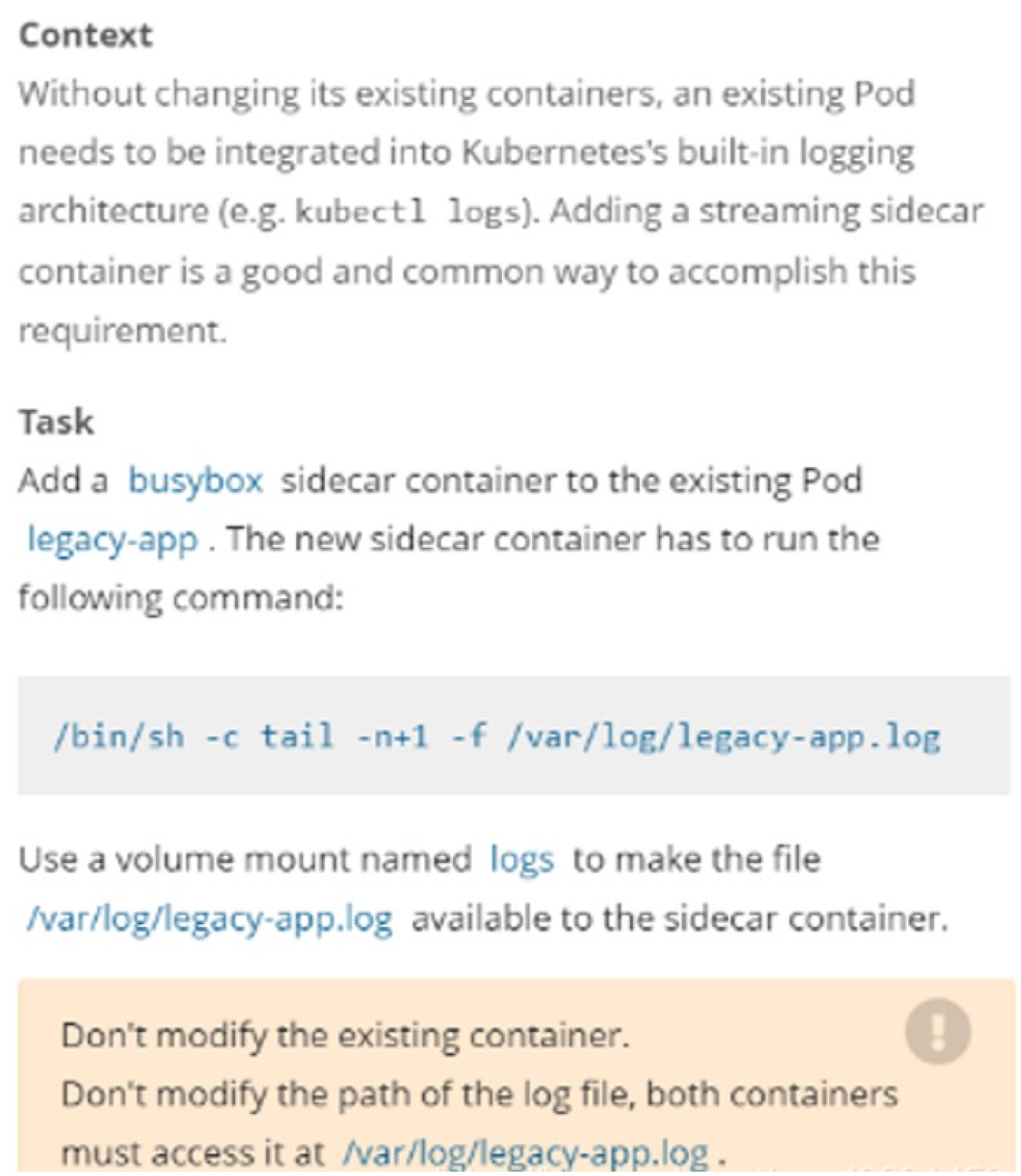
解题：

本地可参考：https://kubernetes.io/zh/docs/concepts/services-networking/ingress/

```
    apiVersion: networking.k8s.io/v1
    kind: Ingress
    metadata:
      name: pong
      namespace: ing-internal
    spec:
      rules:
      - http:
          paths:
```

```
        - path: /hi
          pathType: Prefix
          backend:
            service:
              name: hi
              port:
                number: 5678
```

1.10 Sidecar



Context

Without changing its existing containers, an existing Pod needs to be integrated into Kubernetes's 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 busybox sidecar container to the existing Pod legacy-app . The new sidecar container has to run the following command:

```
/bin/sh -c tail -n+1 -f /var/log/legacy-app.log
```

Use a volume mount named logs to make the file /var/log/legacy-app.log available to the sidecar container.

Don't modify the existing container.

Don't modify the path of the log file, both containers must access it at /var/log/legacy-app.log .

中文解释：

添加一个名为 busybox 且镜像为 busybox 的 sidecar 到一个已经存在的名为 legacy-app 的 Pod 上，这个 sidecar 的启动命令为/bin/sh, -c, 'tail -n+1 -f /var/log/legacy-app.log'。

并且这个 sidecar 和原有的镜像挂载一个名为 logs 的 volume，挂载的目录为/var/log/

解题：

本题答案：https://kubernetes.io/zh/docs/concepts/cluster-administration/logging/

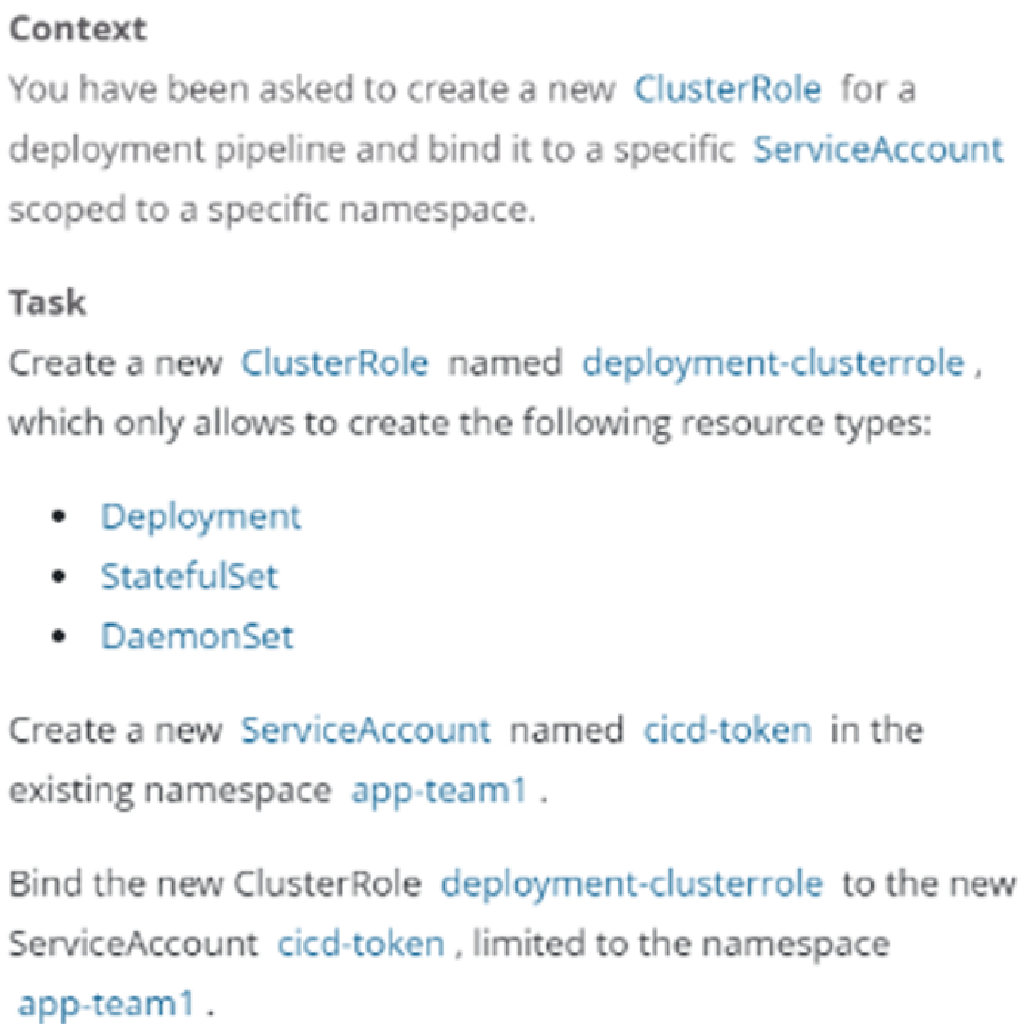
首先将 legacy-app 的 Pod 的 yaml 导出，大致如下：

```
$ kubectl get po legacy-app -oyaml > c-sidecar.yaml
apiVersion: v1
kind: Pod
metadata:
  name: legacy-app
spec:
  containers:
  - name: count
    image: busybox
    args:
    - /bin/sh
    - -c
    - >
      i=0;
      while true;
      do
        echo "$(date) INFO $i" >> /var/log/legacy-ap.log;
        i=$((i+1));
        sleep 1;
      done
```

再此 yaml 中添加 sidecar 和 volume

```
$ vim c-sidecar.yaml
apiVersion: v1
kind: Pod
metadata:
  name: legacy-app
spec:
  containers:
  - name: count
    image: busybox
    args:
    - /bin/sh
    - -c
    - >
      i=0;
      while true;
      do
        echo "$(date) INFO $i" >> /var/log/legacy-ap.log;
        i=$((i+1));
        sleep 1;
      done
    volumeMounts:
    - name: logs
      mountPath: /var/log
  - name: busybox
    image: busybox
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/legacy-ap.log']
    volumeMounts:
    - name: logs
      mountPath: /var/log
  volumes:
  - name: logs
    emptyDir: {}
$ kubectl  delete -f c-sidecar.yaml ; kubectl create -f c-sidecar.yaml
```

1.11 RBAC



Context

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
- StatefulSet
- 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 .

中文解释：

创建一个名为 deployment-clusterrole 的 clusterrole，该 clusterrole 只允许创建 Deployment、Daemonset、Statefulset 的 create 操作

在名字为 app-team1 的 namespace 下创建一个名为 cicd-token 的 serviceAccount，并且将上一步创建 clusterrole 的权限绑定到该 serviceAccount

解题：

可参考：https://kubernetes.io/zh/docs/reference/access-authn-authz/rbac/

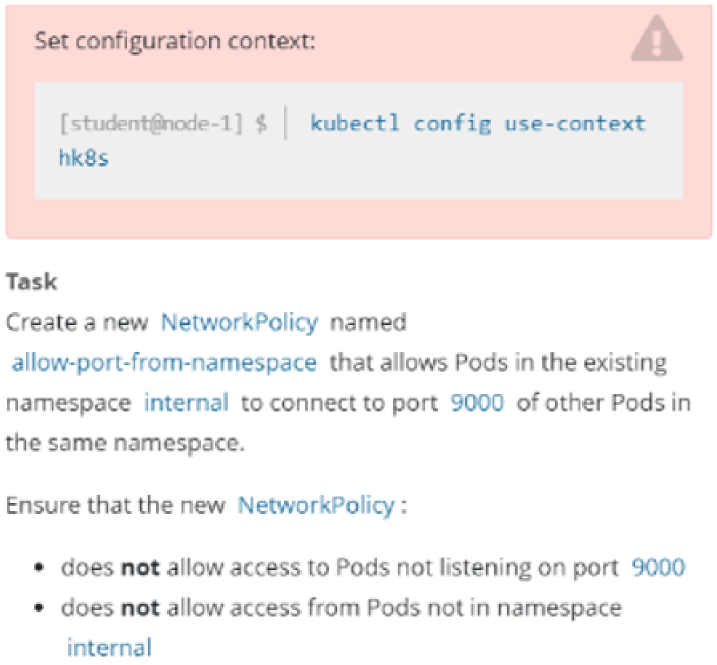
```
创建 clusterrole
[root@k8s-master01 ~]# cat dp-clusterrole.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: deployment-clusterrole
rules:
- apiGroups: ["extensions", "apps"]
  #
  # at the HTTP level, the name of the resource for accessing Secret
  # objects is "secrets"
  resources: ["deployments","statefulsets","daemonsets"]
```

```
      verbs: ["create"]
    [root@k8s-master01 ~]# kubectl create -f dp-clusterrole.yaml
    clusterrole.rbac.authorization.k8s.io/deployment-clusterrole created
    创建 serviceAccount
    # kubectl  create sa cicd-token -n app-team1
    serviceaccount/cicd-token created
```

```
    绑定权限（推荐，节省时间）
    [root@k8s-master01 ~]# kubectl create rolebinding deployment-rolebinding
--clusterrole=deployment-clusterrole --serviceaccount=app-team1:cicd-token -
n app-team1

    或者
    apiVersion: rbac.authorization.k8s.io/v1
    kind: RoleBinding
    metadata:
      name: deployment-rolebinding
      namespace: app-team1
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: ClusterRole
      name: deployment-clusterrole
    subjects:
    - kind: ServiceAccount
      name: cicd-token
      namespace: app-team1
```

1.12 NetworkPolicy



Set configuration context:

```
[student@node-1] $ | kubectl config use-context
hk8s
```

Task

Create a new NetworkPolicy named allow-port-from-namespace that allows Pods in the existing namespace internal to connect to port 9000 of other Pods in the same namespace.

Ensure that the new NetworkPolicy :

- does **not** allow access to Pods not listening on port 9000
- does **not** allow access from Pods not in namespace internal

中文解释：

创建一个名字为 allow-port-from-namespace 的 NetworkPolicy，这个 NetworkPolicy 允许 internal 命名空间下的 Pod 访问该命名空间下的 9000 端口。

并且不允许不是 internal 命令空间的下的 Pod 访问

不允许访问没有监听 9000 端口的 Pod。

解题：

参考：https://kubernetes.io/zh/docs/concepts/services-networking/network-policies/

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-port-from-namespace
  namespace: internal
spec:
  ingress:
  - from:
    - podSelector: {}
    ports:
    - port: 9000
      protocol: TCP
  podSelector: {}
  policyTypes:
  - Ingress
```

1.12.1 NetworkPolicy 此题可能存在的变化

上述的题目是只限制在 internal 命名空间下的，该题可能存在更新。更新如下：

在现有的 namespace my-app 中创建一个名为 allow-port-from-namespace 的 NetworkPolicy 确保这个 NetworkPolicy 允许 namespace my-app 中的 pods 可以连接到 namespace big-corp 中的 8080。

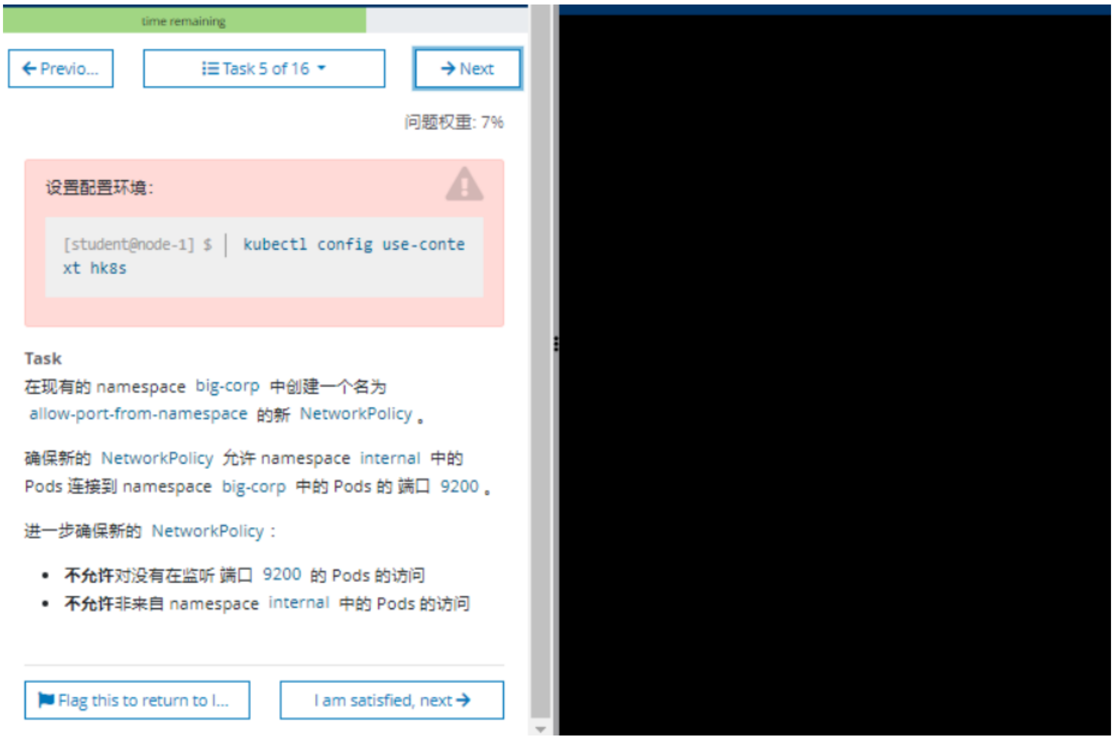
并且不允许不是 my-app 命令空间的下的 Pod 访问

不允许访问没有监听 8080 端口的 Pod。

所以可以拿着上述的答案，进行稍加修改（注意 namespaceSelector 的 labels 配置，首先需要查看 big-corp 命名空间有没有标签：kubectl get ns big-corp --show-labels 如果有，可以更改 name: big-corp 为查看到的即可。如果没有需要添加一个 label：kubectl label ns big-corp name=big-corp）：

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-port-from-namespace
  namespace: my-app
spec:
  egress:
  - to:
    - namespaceSelector:
         matchLabels:
            name: big-corp
    ports:
    - protocol: TCP
      port: 8080
  ingress:
  - from:
    - podSelector: {}
    ports:
    - port: 8080
      protocol: TCP
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

1.12.2 NetworkPolicy 此题可能存在的变化 2



问题权重: 7%

设置配置环境:

```
[student@node-1] $ | kubectl config use-context hk8s
```

Task

在现有的 namespace big-corp 中创建一个名为 allow-port-from-namespace 的新 NetworkPolicy。

确保新的 NetworkPolicy 允许 namespace internal 中的 Pods 连接到 namespace big-corp 中的 Pods 的 端口 9200。

进一步确保新的 NetworkPolicy：

- **不允许**对没有在监听 端口 9200 的 Pods 的访问
- **不允许**非来自 namespace internal 中的 Pods 的访问

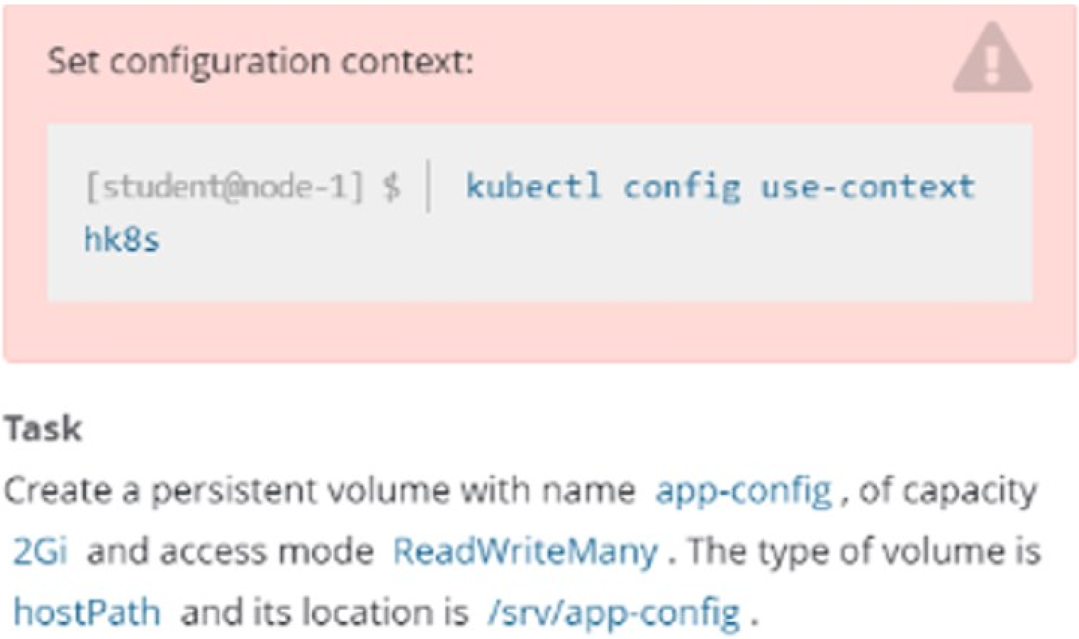
解题：

参考：https://kubernetes.io/zh/docs/concepts/services-networking/network-policies/

此题和上题比较比较简单，只需要允许 internal 命名空间即可

```
  apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-port-from-namespace
    namespace: big-corp
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: internal
      ports:
      - port: 80
        protocol: TCP
    podSelector: {}
    policyTypes:
    - Ingress
```

1.13 PersistentVolume



Set configuration context:

```
[student@node-1] $ | kubectl config use-context
hk8s
```

Task

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

中文解释：

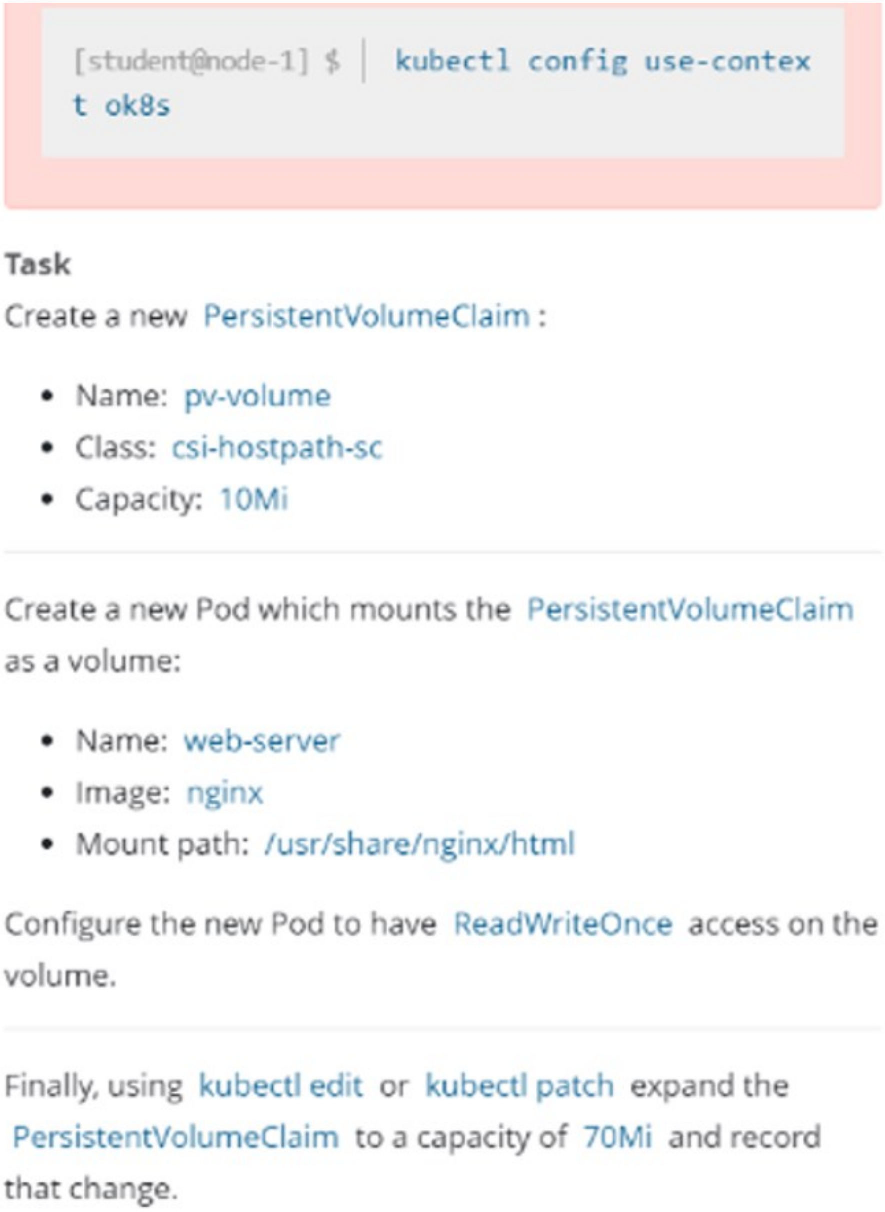
创建一个 pv，名字为 app-config，大小为 2Gi，访问权限为 ReadWriteMany。Volume 的类型为 hostPath，路径为/srv/app-config

解题：

参考：https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: app-config
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/srv/app-config"
```

1.14 CSI & PersistentVolumeClaim



```
[student@node-1] $ kubectl config use-context ok8s
```

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.

中文文档：

创建一个名字为 pv-volume 的 pvc，指定 storageClass 为 csi-hostpath-sc，大小为 10Mi 然后创建一个 Pod，名字为 web-server，镜像为 nginx，并且挂载该 PVC 至/usr/share/nginx/html，挂载的权限为 ReadWriteOnce。之后通过 kubectl edit 或者 kubectl path 将 pvc 改成 70Mi，并且记录修改记录。

解题：

参考：https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/

创建 PVC：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-volume
```

```
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 10Mi
  storageClassName: csi-hostpath-sc
```

创建 Pod：

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

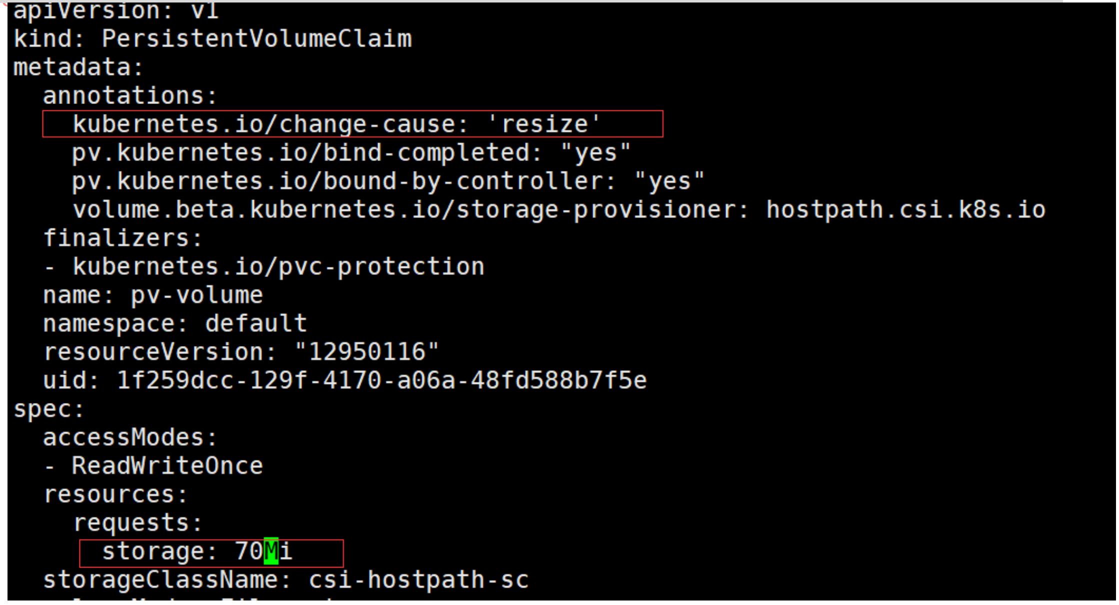
扩容：

方式一 Patch 命令：

```
kubectl patch pvc pv-volume  -p
'{"spec":{"resources":{"requests":{"storage": "70Mi"}}}}' --record
```

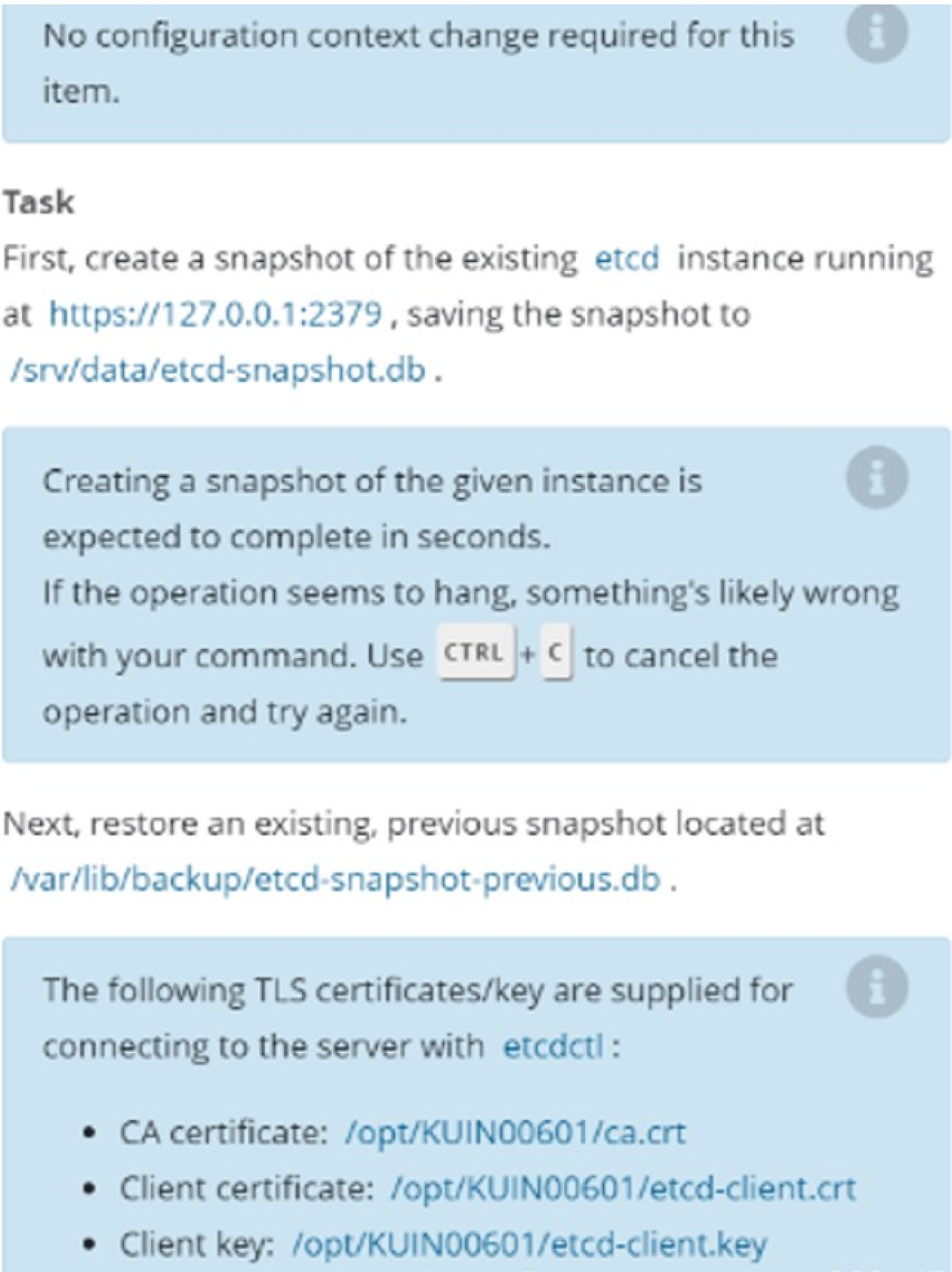
方式二 edit：

```
kubectl  edit pvc pv-volume
```



```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    kubernetes.io/change-cause: 'resize'
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: hostpath.csi.k8s.io
  finalizers:
  - kubernetes.io/pvc-protection
  name: pv-volume
  namespace: default
  resourceVersion: "12950116"
  uid: 1f259dcc-129f-4170-a06a-48fd588b7f5e
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 70Mi
  storageClassName: csi-hostpath-sc
```

1.15 Etcd 备份恢复



No configuration context change required for this item.

Task

First, create a snapshot of the existing etcd instance running at https://127.0.0.1:2379 , saving the snapshot to /srv/data/etcd-snapshot.db .

Creating a snapshot of the given instance is expected to complete in seconds.
If the operation seems to hang, something's likely wrong with your command. Use CTRL + C to cancel the operation and try again.

Next, restore an existing, previous snapshot located at /var/lib/backup/etcd-snapshot-previous.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
- Client key: /opt/KUIN00601/etcd-client.key

中文解释：

针对 etcd 实例 https://127.0.0.1:2379 创建一个快照，保存到/srv/data/etcd-snapshot.db。在创建快照的过程中，如果卡住了，就键入 ctrl+c 终止，然后重试。

然后恢复一个已经存在的快照： /var/lib/backup/etcd-snapshot-previous.db

执行 etcdctl 命令的证书存放在：

ca 证书：/opt/KUIN00601/ca.crt

客户端证书：/opt/KUIN00601/etcd-client.crt

客户端密钥：/opt/KUIN00601/etcd-client.key

解题：

可参考：https://kubernetes.io/zh/docs/tasks/administer-cluster/configure-upgrade-etcd/

```
    $ export ETCDCTL_API=3
    $ etcdctl --endpoints="https://127.0.0.1:2379" --
cacert=/opt/KUIN000601/ca.crt --cert=/opt/KUIN000601/etcd-client.crt --
key=/opt/KUIN000601/etcd-client.key  snapshot save /srv/data/etcd-snapshot.db

    还原
    $ mkdir /opt/backup/ -p
    $ cd /etc/kubernetes/manifests ; mv kube-* /opt/backup
    $ export ETCDCTL_API=3  etcdctl --endpoints="https://127.0.0.1:2379" --
cacert=/opt/KUIN000601/ca.crt --cert=/opt/KUIN000601/etcd-client.crt --
key=/opt/KUIN000601/etcd-client.key  snapshot restore /var/lib/backup/etcd-
snapshot-previous.db --data-dir=/var/lib/etcd-restore
    $ vim etcd.yaml
```

# 将 volume 配置的 path: /var/lib/etcd 改成/var/lib/etcd-restore

```
      volumes:
      - hostPath:
          path: /etc/kubernetes/pki/etcd
          type: DirectoryOrCreate
        name: etcd-certs
      - hostPath:
          path: /var/lib/etcd-restore
```

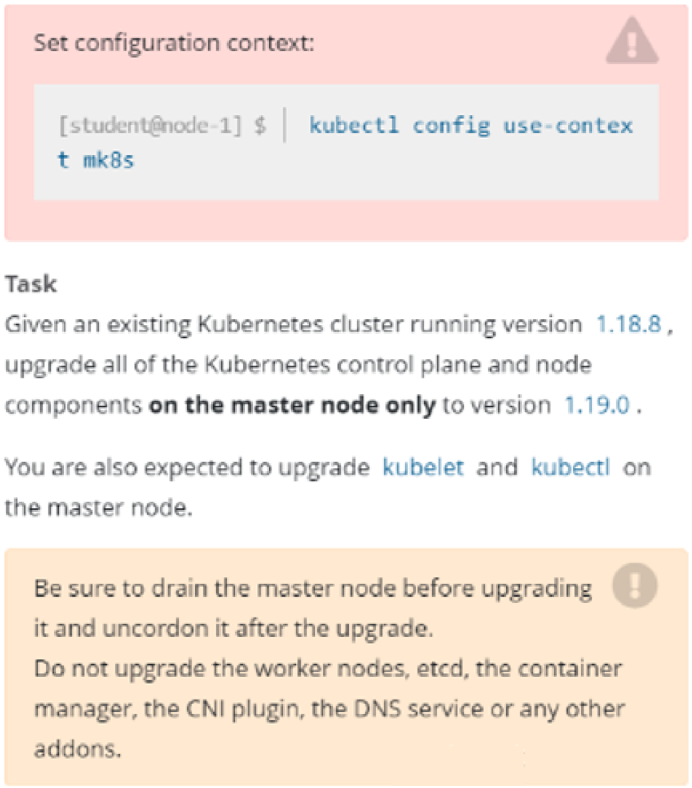
# 还原 k8s 组件

```
    $ mv /opt/backup/* /etc/kubernetes/manifests
    $ systemctl restart kubelet
```

注意

如果是二进制安装的 etcd，考试环境的 etcd 可能并非 root 用户启动的，所以可以先切换到 root 用户（sudo su -）,然后使用 ps aux | grep etcd 查看启动用户是谁和启动的配置文件是谁 config-file 字段指定，假设用户是 etcd。所以如果是二进制安装的 etcd，执行恢复时需要 root 权限，所以在恢复数据时，可以使用 root 用户恢复，之后更改恢复目录的权限：sudo chown -R etcd.etcd /var/lib/etcd-restore，然后通过 systemctl status etcd（或者 ps aux | grep etcd）找到它的配置文件（如果没有配置文件，就可以直接在 etcd 的 service【通过 systemctl status etcd 即可看到】文件中找到 data-dir 的配置），然后更改 data-dir 配置后，执行 systemctl daemon-reload，最后使用 etcd 用户 systemctl restart etcd 即可。

1.16 K8s 升级



Set configuration context:

```
[student@node-1] $ | kubectl config use-contex
t mk8s
```

Task

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.

You are also expected to upgrade kubelet and kubectl on the master node.

Be sure to drain the master node before upgrading it and uncordon it after the upgrade.
Do not upgrade the worker nodes, etcd, the container manager, the CNI plugin, the DNS service or any other addons.

解题：

参考：https://kubernetes.io/zh/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/

首先腾空节点：

```
    # 设置为维护状态
    $ kubectl cordon k8s-master
    # 驱逐 Pod
    $ kubectl drain k8s-master --delete-emptydir-data --ignore-daemonsets --
force
    # 之后需要按照题目提示 ssh 到一个 master 节点
    $ apt update
    $ apt-cache policy kubeadm | grep 1.19.0  # (注意版本的差异，有可能并非 1.18.8
升级到 1.19)
    $ apt-get install kubeadm=1.19.0-00
```

```
    # 验证升级计划
    $ kubeadm upgrade plan
    # 看到如下信息，可升级到指定版本
    You can now apply the upgrade by executing the following command:

        kubeadm upgrade apply v1.19.0

    ________________________________________________________________________
```

```
    # 开始升级 Master 节点
    $ kubeadm  upgrade apply v1.19.0 --etcd-upgrade=false
    [upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.19.0".
Enjoy!

    [upgrade/kubelet] Now that your control plane is upgraded, please
```

```
proceed with upgrading your kubelets if you haven't already done so.
```

# 升级 kubectl 和 kubelet

```
    $ apt-get install -y kubelet=1.19.0-00 kubectl=1.19.0-00
    $ systemctl daemon-reload
    $ systemctl restart kubelet
    $ kubectl uncordon k8s-master
    node/k8s-master uncordoned
    $ kubectl  get node
    NAME          STATUS     ROLES                  AGE   VERSION
    k8s-master01   NotReady   control-plane,master   11d   v1.19.0
    k8s-node01     Ready      <none>                 8d    v1.18.8
    k8s-node02     Ready      <none>                 11d   v1.18.8
    $ kubectl  get node
    NAME          STATUS   ROLES                  AGE   VERSION
    k8s-master01   Ready    control-plane,master   11d   v1.19.0
    k8s-node01     Ready    <none>                 8d    v1.18.8
    k8s-node02     Ready    <none>                 11d   v1.18.8
```

1.16.1 升级到 1.21.1

解题：

参考：https://kubernetes.io/zh/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/

首先腾空节点：

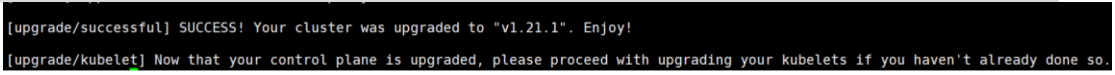
```
    # 设置为维护状态
    $ kubectl cordon k8s-master
    # 驱逐 Pod
    $ kubectl drain k8s-master --delete-emptydir-data --ignore-daemonsets --
force
    # 之后需要按照题目提示 ssh 到一个 master 节点
    $ apt update
    $ apt-cache policy kubeadm | grep 1.21.1  # (注意版本的差异，有可能并非 1.20.1
升级到 1.21.1)
    $ apt-get install kubeadm=1.21.1-00
```

# 验证升级计划

```
    $ kubeadm upgrade plan
    # 看到如下信息，可升级到指定版本
```

# 开始升级 Master 节点，注意看题需不需要升级 etcd

```
    $ kubeadm  upgrade apply v1.21.1 --etcd-upgrade=false -f
```



```
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.21.1". Enjoy!
[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading your kubelets if you haven't already done so.
```

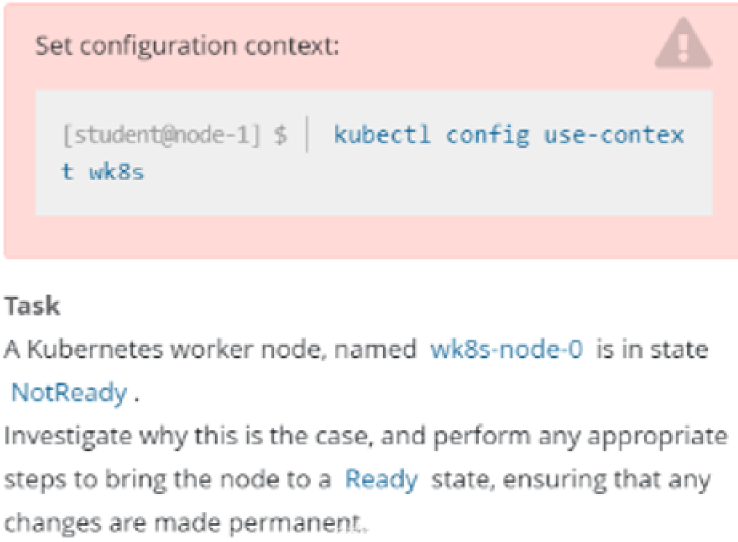
注意

自己的环境升级，可能会报找不到 coredns 的镜像，可以使用如下方法解决：

所有节点 docker pull coredns/coredns:1.8.0 ; docker tag coredns/coredns:1.8.0 registry.cn-hangzhou.aliyuncs.com/google_containers/coredns/coredns:v1.8.0 然后继续就行。1.8.0 改成你自己 CoreDNS 报错的版本

```
# 升级 kubectl 和 kubelet
$ apt-get install -y kubelet=1.21.1-00 kubectl=1.21.1-00
$ systemctl daemon-reload
$ systemctl restart kubelet
$ kubectl uncordon k8s-master
node/k8s-master uncordoned
$ kubectl  get node
NAME          STATUS    ROLES                  AGE   VERSION
k8s-master01   NotReady   control-plane,master   11d   v1.21.1
k8s-node01     Ready      <none>                 8d    v1.12.1
$ kubectl  get node
NAME          STATUS   ROLES                  AGE   VERSION
k8s-master01   Ready    control-plane,master   11d   v1.21.1
k8s-node01     Ready    <none>                 8d    v1.20.1
k8s-node02     Ready    <none>                 11d   v1.20.1
```

1.17 集群故障排查 – kubelet 故障



Set configuration context:

```
[student@node-1] $ 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.

中文解释：

一个名为 wk8s-node-0 的节点状态为 NotReady，让其他恢复至正常状态，并确认所有的更改开机自动完成

解题：

```
$ ssh wk8s-node-0
$ sudo -i
# systemctl status kubelet
# systemctl start kubelet
# systemctl enable kubelet
```

1.18 集群故障排查 – 主节点故障

https://kubernetes.io/zh/docs/tasks/configure-pod-container/static-pod/