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~~~

## 监控Pod日志



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

## 监控Pod度量指标



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)

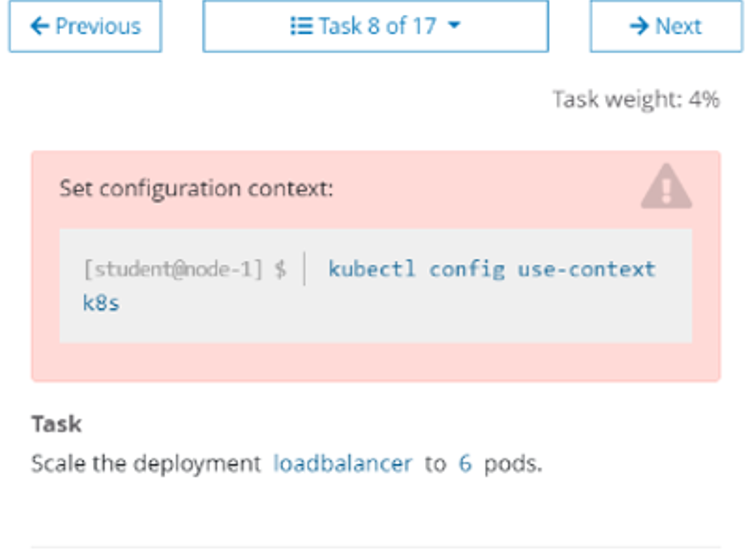
kube-system coredns-54d67798b7-hl8xc 7m 8Mi

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

## Deployment扩缩容



中文解释：

扩容名字为loadbalancer的deployment的副本数为6

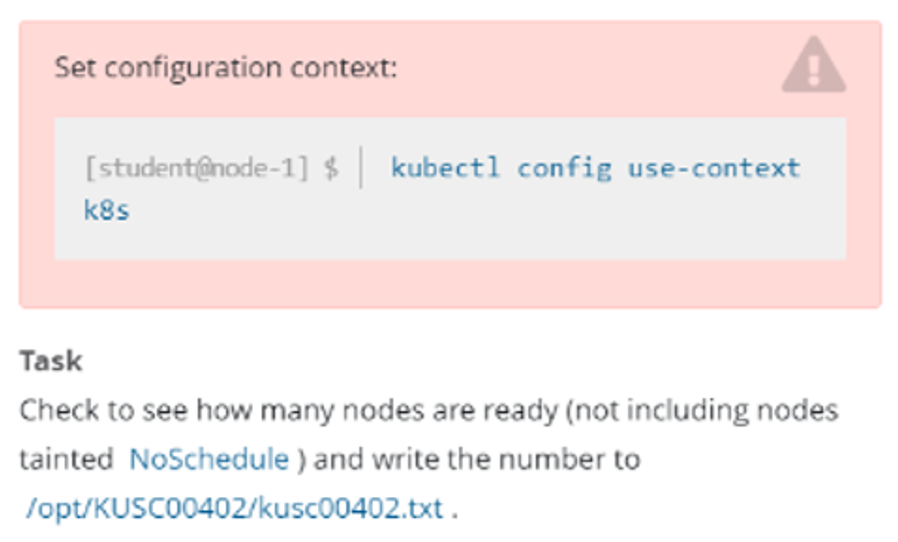
解题：

$ **kubectl config use-context k8s**

$ kubectl scale --replicas=6 deployment loadbalancer

$ kubectl edit

## 检查Node节点的健康状态



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

## 节点维护



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：

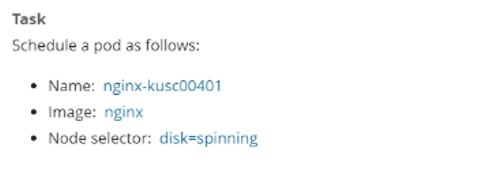
$ **kubectl config use-context ek8s**

$ 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/>

## 指定节点部署



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

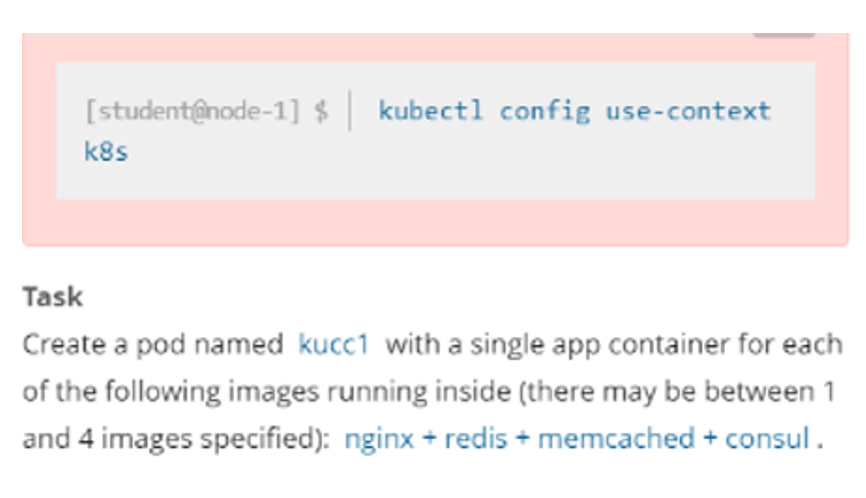
containers:

- name: nginx

image: nginx

$ kubectl create -f pod-ns.yaml

## 一个Pod多个容器



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

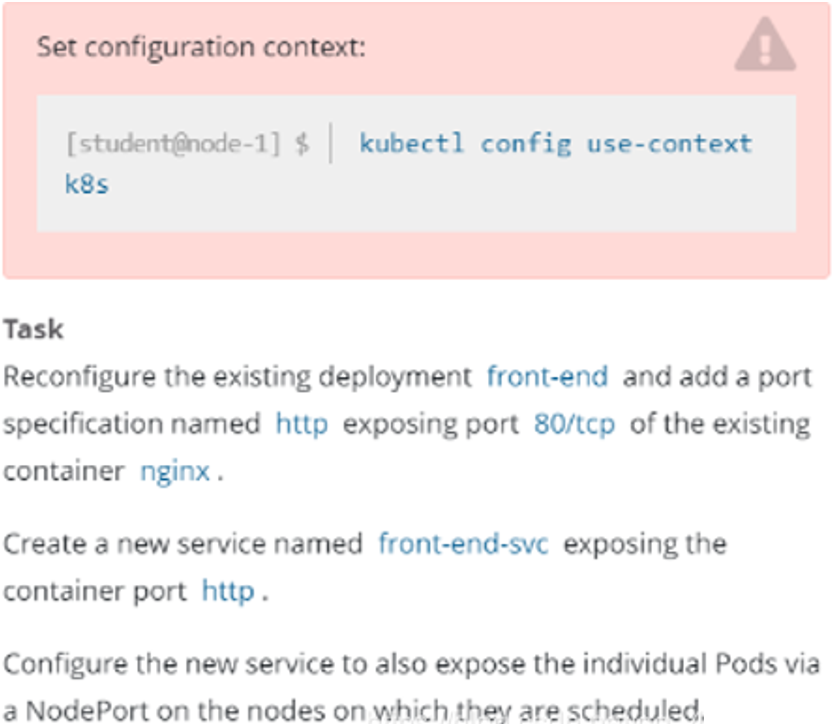
- image: memchached

name: memcached

- image: consul

name: consul

## Service



中文解释：

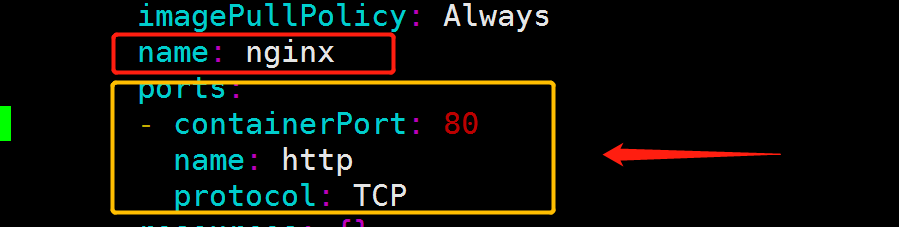
重新配置一个已经存在的deployment front-end，在名字为nginx的容器里面添加一个端口配置，名字为http，暴露端口号为80，然后创建一个service，名字为front-end-svc，暴露该deployment的http端口，并且service的类型为NodePort。

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

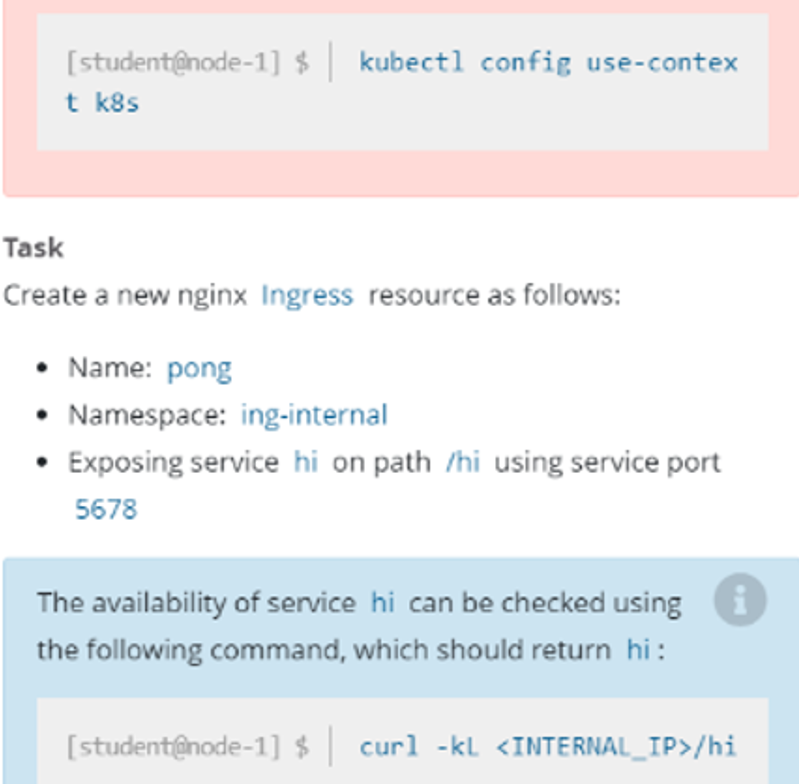


添加service：

$

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

## Ingress



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:

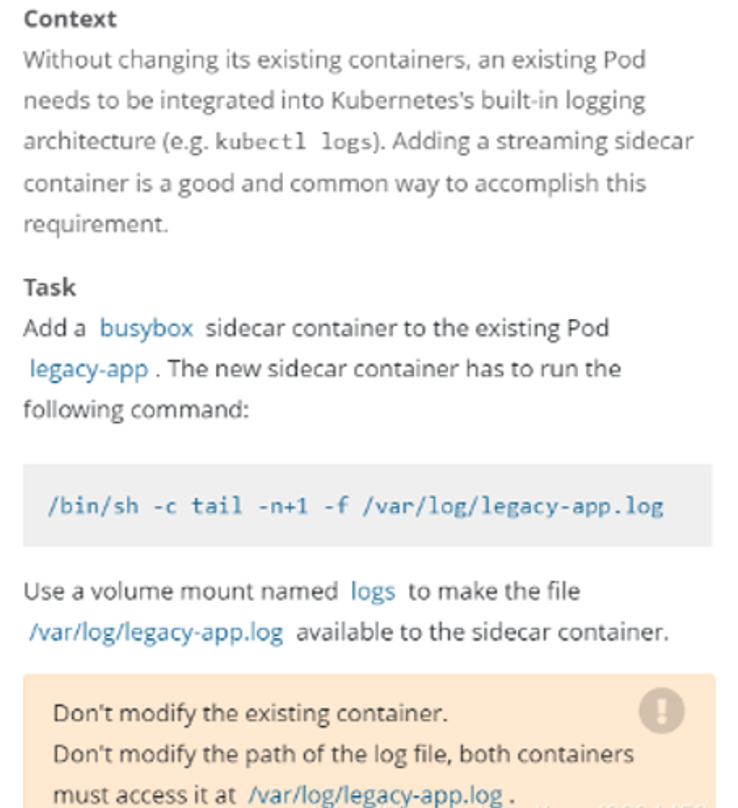
service:

name: hi

port:

number: 5678

## Sidecar



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

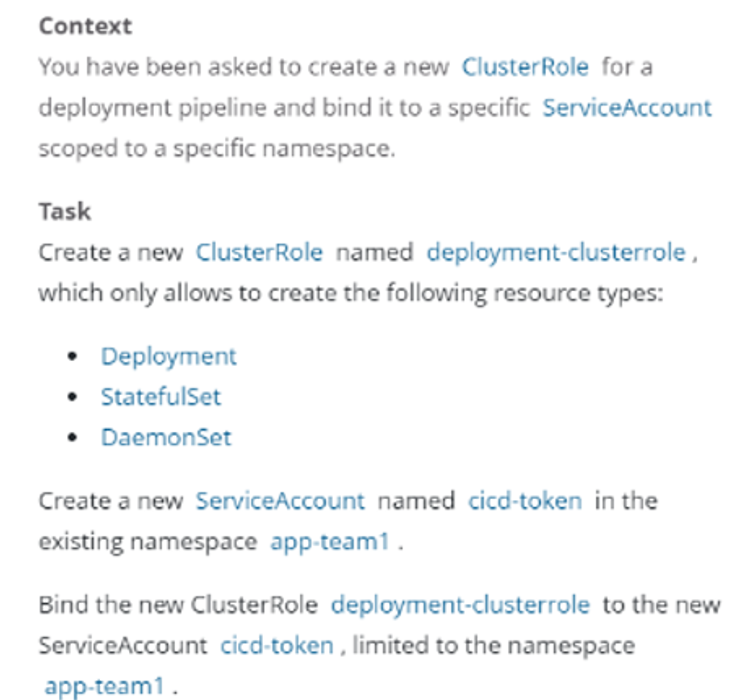
volumes:

- name: logs

emptyDir: {}

$ **kubectl delete -f c-sidecar.yaml ; kubectl create -f c-sidecar.yaml**

## RBAC



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

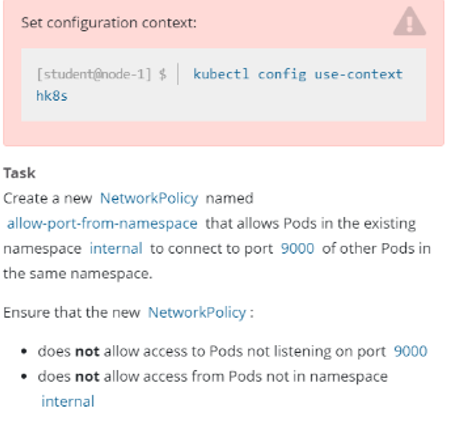
subjects:

- kind: ServiceAccount

name: cicd-token

namespace: app-team1

## NetworkPolicy



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

### 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

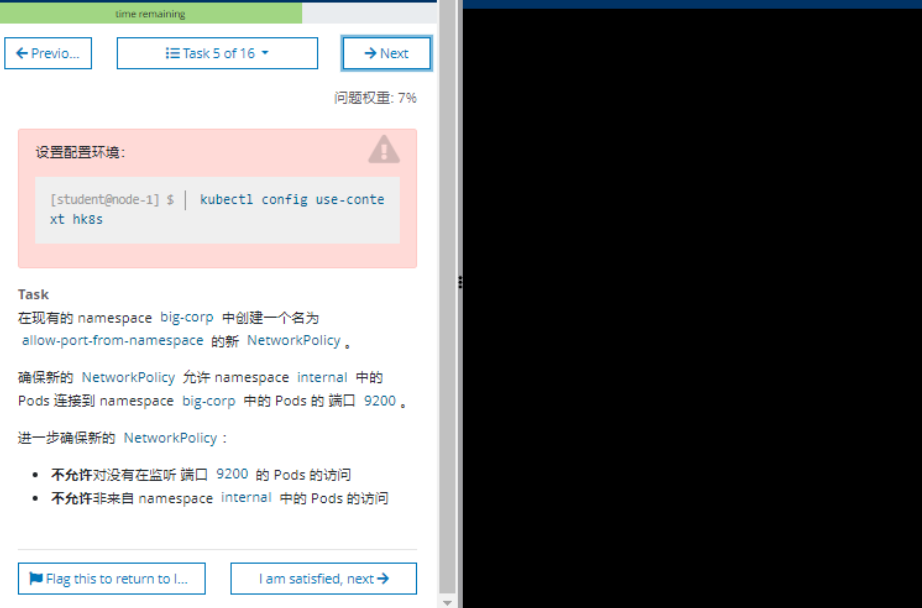
podSelector: {}

policyTypes:

- Ingress

- Egress

### NetworkPolicy此题可能存在的变化2



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**

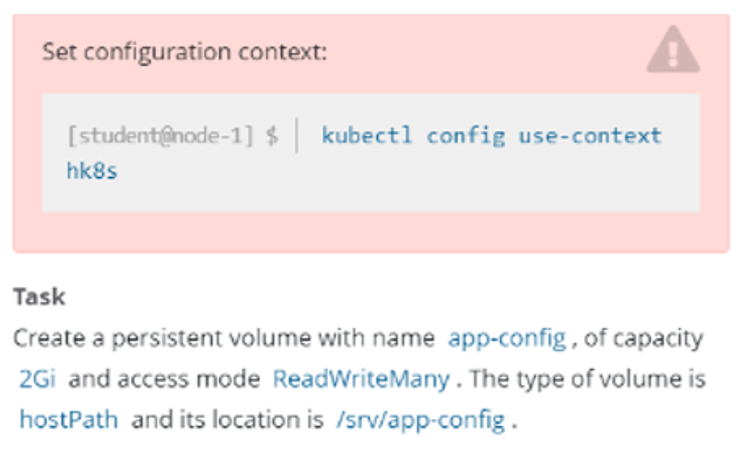
**protocol: TCP**

podSelector: {}

policyTypes:

- Ingress

## PersistentVolume



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

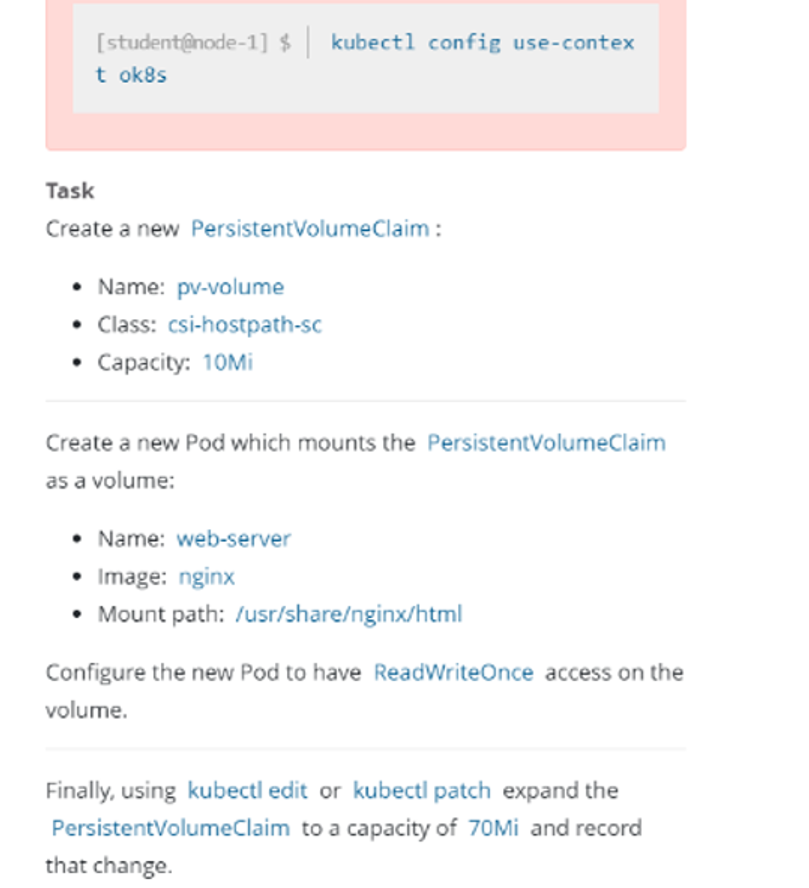
accessModes:

- ReadWriteMany

hostPath:

path: "/srv/app-config"

## CSI & PersistentVolumeClaim



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

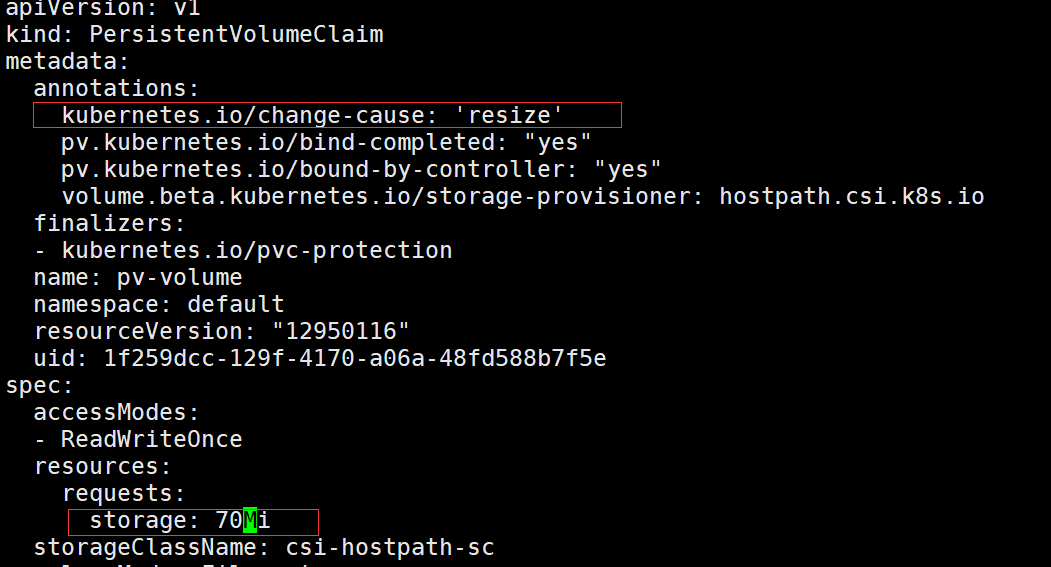
扩容：

方式一Patch命令：

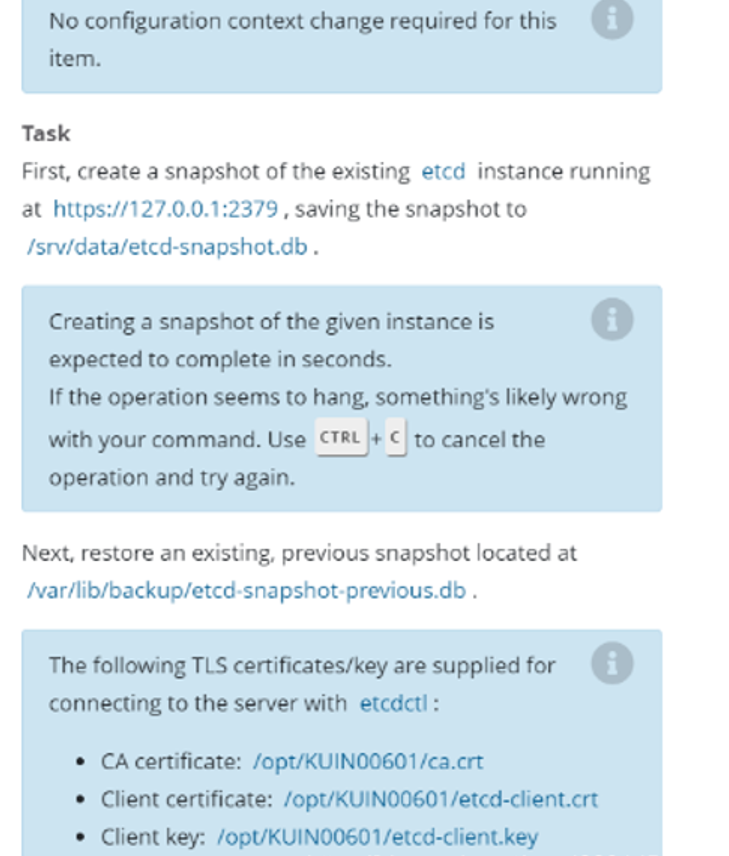
kubectl patch pvc pv-volume -p '{"spec":{"resources":{"requests":{"storage": "70Mi"}}}}' --record

方式二edit：

kubectl edit pvc pv-volume



## Etcd备份恢复



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

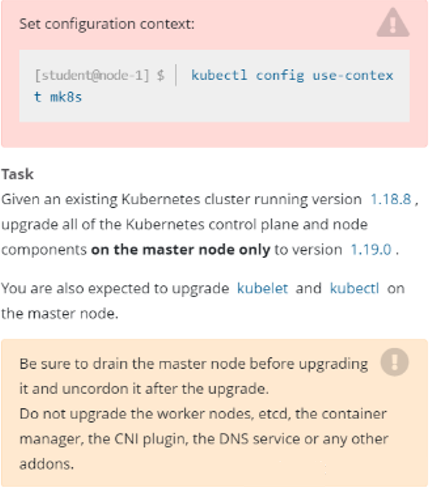
$ mv /opt/backup/\* /etc/kubernetes/manifests

$ 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。

## K8s升级



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

### 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

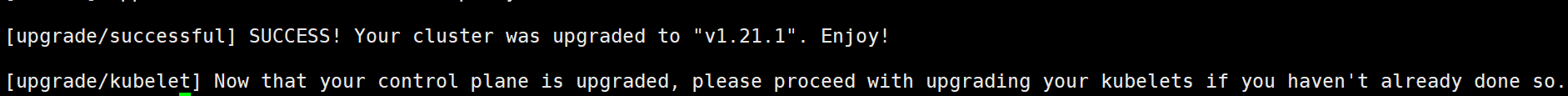
# 验证升级计划

$ 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**



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

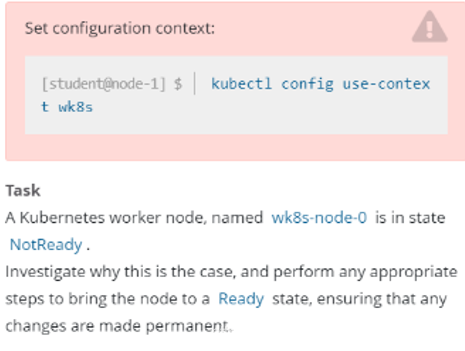
NAME STATUS ROLES AGE VERSION

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

## 集群故障排查 – kubelet故障



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

## 集群故障排查 – 主节点故障

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