注意 必读!

模拟环境是在 node01 （ 11.0.1.112 ）上做题的。
帐号为 candidate
密码为 123
在此账号下，可以免密 ssh 登录 master01 和 node02 （ ssh master01 或 ssh node02 ）
3 台虚拟机的 candidate 帐号也可以免密切换到 root （ sudo -i ）

考试时，是 candidate 或 student 用户，在 node-1 机器上做题的，也不是在 root 的，也不是在 master 的，要注意。

（请注意，文档里的一些截图还是使用的 student 帐号，没有更新。但这不影响的。只是因为在 v1.23 里用的是 student，截图是当时 v1.23 时候截的，但是在 v1.26 里用的帐号是 candidate 了。）

注意，官方的 CKA 考试环境是有多套的，你考试时，不一定抽到哪套。不同考试环境里面部分题目的内容有很小的变化，但题干都是一样的。在下面的答案解析中也有详细的描述。

注意阅读《K8S 考试流程(报名、约考、注意).pdf》，几乎所有同学都反馈新考试平台非常卡，70%反馈不用 V-P-N 没法做题，20%反馈用了 V-P-N 还是卡，甚至有几个同学没有做完题。
所以日常要练习的很熟练，尽量熟练到 80%的题不参考 K8S 网页!!!
如果太卡，导致可能做不完题，CKA 一定要把“排查集群中故障节点 kubelet”那道题做完，因为它的分值最高，且也操作最简单。

新考试平台，不允许外接第二个屏幕，不允许访问自己的浏览器书签。
新考试平台，更像是一个远程的 Ubuntu 桌面，在这个 Ubuntu 桌面里，你可以用终端做题，也可以用 Ubuntu 自带的火狐浏览器到官网自己查。
另外每道题目都会给你一个官方的参考链接，可以先看这个连接，如果找不到自己需要的信息，再从官网自己查找。
注意考试时搜出来的结果，跟平常在官网搜出来结果排序不一样。还有就是官网网页的最右边，是可以点击选择中文/英文的。
最后，能不查官网就尽量不要查，因为真的很卡，很费时间。

每次还原初始化快照后，开机后等 5 分钟，ssh 登录 node01（11.0.1.112）检查一下所有的 pod，确保都为 Running，再开始练习。
kubectl get pod -A


```
candidate@node01:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   9d    v1.26.0
node01     Ready    <none>          9d    v1.26.0
node02     Ready    <none>          9d    v1.26.0
candidate@node01:~$
candidate@node01:~$ kubectl get pod -A
NAMESPACE          NAME                                       READY   STATUS    RESTARTS          AGE
calico-apiserver   calico-apiserver-5b694c87c8-gtbgc          1/1     Running   7 (6m26s ago)     9d
calico-apiserver   calico-apiserver-5b694c87c8-mwwhd          1/1     Running   7 (6m27s ago)     9d
calico-system      calico-kube-controllers-6b7b9c649d-qjwjx   1/1     Running   7 (6m28s ago)     9d
calico-system      calico-node-4kck4                          1/1     Running   7 (6m26s ago)     9d
calico-system      calico-node-cxbbm                          1/1     Running   7 (6m27s ago)     9d
calico-system      calico-node-kpbwv                          1/1     Running   7 (6m28s ago)     9d
calico-system      calico-typha-7c645fbb95-8bjlj              1/1     Running   7 (6m26s ago)     9d
calico-system      calico-typha-7c645fbb95-bkx8t              1/1     Running   7 (6m27s ago)     9d
calico-system      csi-node-driver-4277d                      2/2     Running   14 (6m28s ago)    9d
calico-system      csi-node-driver-h65tq                      2/2     Running   14 (6m27s ago)    9d
calico-system      csi-node-driver-k9kcl                      2/2     Running   14 (6m26s ago)    9d
cpu-top            nginx-host-56c8f5b697-ltdmd                1/1     Running   4 (6m27s ago)     8d
cpu-top            redis-test-ff9fd7d78-dsbf2                 1/1     Running   4 (6m26s ago)     8d
cpu-top            test0-75d9886fc7-44b28                     1/1     Running   4 (6m26s ago)     8d
default            11-factor-app                              1/1     Running   4 (6m26s ago)     8d
default            foo                                        1/1     Running   4 (6m27s ago)     8d
default            front-end-6d67cb94dc-z4gch                 1/1     Running   4 (6m26s ago)     8d
default            presentation-d9dbc88cb-ff8g5               1/1     Running   4 (6m27s ago)     8d
ing-internal       hello-8544bccfcb-m2v49                     1/1     Running   4 (6m26s ago)     8d
ingress-nginx      ingress-nginx-controller-2qctj             1/1     Running   4 (6m27s ago)     8d
ingress-nginx      ingress-nginx-controller-9g98t             1/1     Running   4 (6m26s ago)     8d
kube-system        coredns-5bbd96d687-2vsp8                   1/1     Running   7 (6m28s ago)     9d
kube-system        coredns-5bbd96d687-7p5mt                   1/1     Running   7 (6m28s ago)     9d
kube-system        etcd-master01                              1/1     Running   7 (6m28s ago)     9d
kube-system        kube-apiserver-master01                    1/1     Running   7 (6m28s ago)     9d
kube-system        kube-controller-manager-master01           1/1     Running   7 (6m28s ago)     9d
kube-system        kube-proxy-fwtxx                           1/1     Running   7 (6m28s ago)     9d
kube-system        kube-proxy-k5pjg                           1/1     Running   7 (6m27s ago)     9d
kube-system        kube-proxy-lvmzt                           1/1     Running   7 (6m26s ago)     9d
kube-system        kube-scheduler-master01                    1/1     Running   7 (6m28s ago)     9d
kube-system        metrics-server-67996f964b-bwbhh            1/1     Running   5 (6m27s ago)     8d
tigera-operator    tigera-operator-54b47459dd-ltj7q           1/1     Running   8 (6m27s ago)     9d
candidate@node01:~$
```

kubectl 备忘单（kubectl Cheat Sheet）
https://kubernetes.io/zh-cn/docs/reference/kubectl/cheatsheet/

这套《题库》跟《真题》的题干是一样的，区别在于里面的一些 pod、deployment、namespace、ServiceAccount 等参数可能不同而已，因为在真实考试中，也是会时常变换里面的这些变量参数的。注意理解这些变量的含义，而不要死记硬背答案。

比如以下截图：

设置配置环境：
[student@node-1] $ kubectl config use-context k8s

Task
如下创建一个新的 nginx Ingress 资源：
名称: ping
Namespace: ing-internal
使用服务端口 5678 在路径 /hello 上公开服务 hello
可以使用以下命令检查服务 hello 的可用性，该命令应返回 hello：
curl -kL <INTERNAL_IP>/hello

设置配置环境：
[student@node-1] $ kubectl config use-context hk8s

Task
创建名为 app-config 的 persistent volume，容量为 1Gi，访问模式为 ReadWriteMany。
volume 类型为 hostPath，位于 /srv/app-config

在新的考试环境里，默认 k8s 命令，已经可以自动补全了。如果考试时不可以补全，则参考下面方法：
在 bash 中设置当前 shell 的自动补全，要先安装 apt-get install bash-completion 包（模拟环境已安装）。
然后执行命令 source <(kubectl completion bash)
其实我觉得没有必要设置的，不会的命令可以 tab 键先试试，如果出不来，就-h 帮助一下即可。
我觉得你练习时，使用-h 是最好的，因为这样记忆度更高。

鸣谢：

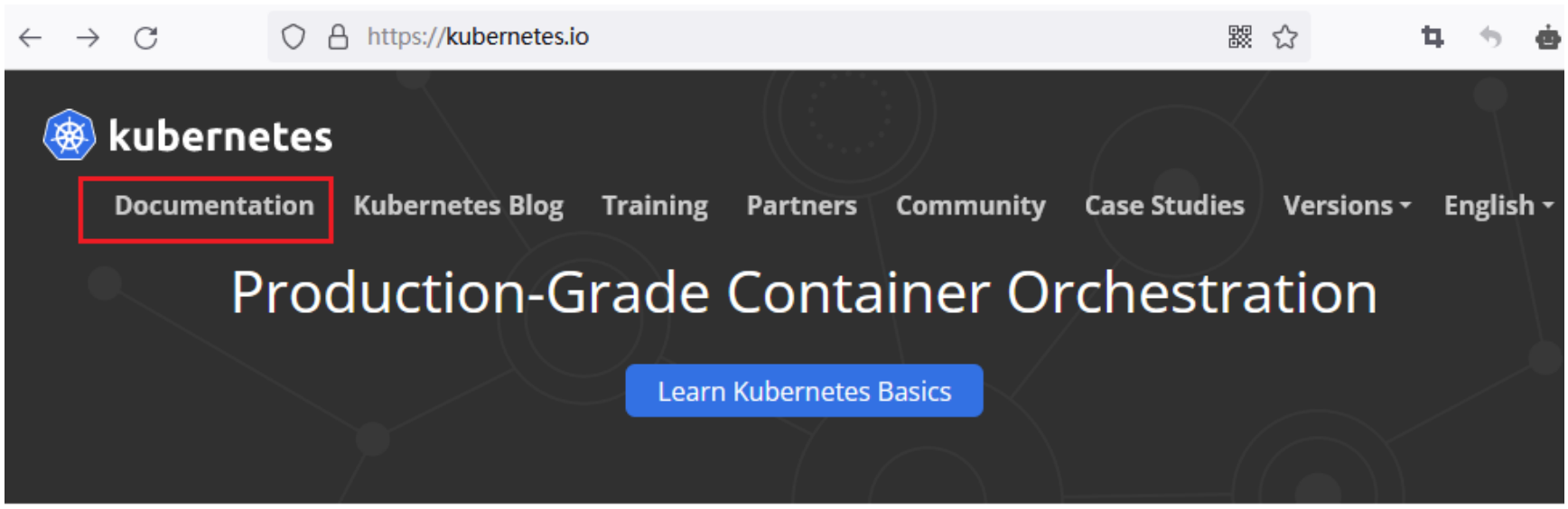
特别鸣谢以下网友（排名不分先后）：

勋（lizhanxun4*）
唐 Jing（sinot*）
gmwinsto* [台]

0、如何一步步找到找官网资料

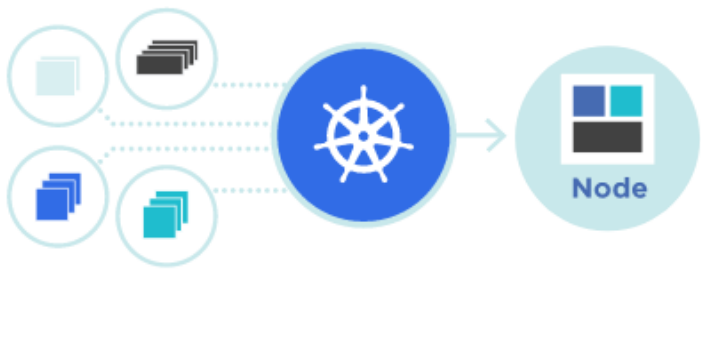
<1> 考试时，打开考试环境 Ubuntu 20.04 桌面上的火狐浏览器，并输入 K8S 官网 https://kubernetes.io/

<2> 点击左上角的“Documentation”



Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

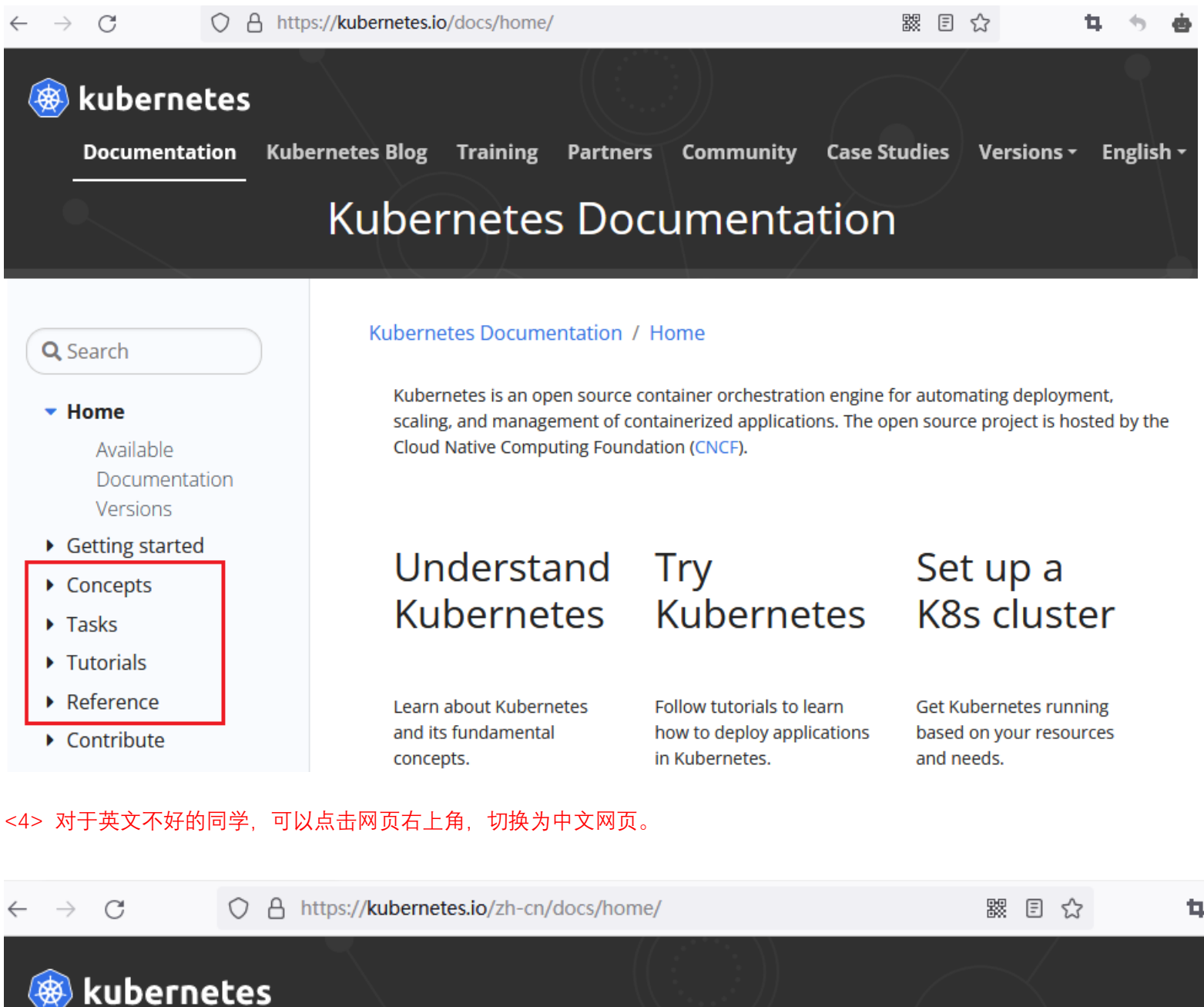
It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon 15 years of experience of running production workloads at Google, combined with best-of-breed ideas and practices from the community.



<3> 后面做题时，如果需要参考官网，我们就在此网址，或者此网址的中文翻译里查找。

https://kubernetes.io/docs/reference/

https://kubernetes.io/zh-cn/docs/reference/



<4> 对于英文不好的同学，可以点击网页右上角，切换为中文网页。

1、权限控制 RBAC

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context k8s

Context
为部署流水线创建一个新的 ClusterRole 并将其绑定到范围为特定的 namespace 的特定 ServiceAccount。

Task
创建一个名为 deployment-clusterrole 且仅允许创建以下资源类型的新 ClusterRole：
Deployment
StatefulSet
DaemonSet
在现有的 namespace app-team1 中创建一个名为 cicd-token 的新 ServiceAccount。
限于 namespace app-team1 中，将新的 ClusterRole deployment-clusterrole 绑定到新的 ServiceAccount cicd-token。

考点：RBAC 授权模型的理解。

参考链接

没必要参考网址，使用-h 帮助更方便。
kubectl create clusterrole -h
kubectl create rolebinding -h

https://kubernetes.io/docs/reference/access-authn-authz/rbac/#command-line-utilities

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context k8s

```
student@node-1:~$ kubectl config use-context k8s
Switched to context "k8s".
student@node-1:~$
```

模拟环境因为没有做相关设置，只有一套集群，所以执行会报错的。
但是建议每道题练习时，还是要输入一遍切换集群的命令，固化思维，防止考试时忘记切换。

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$
```

考试考题会像下图这样子，在考题前面有一个红色的框提醒你，在哪个账号和哪台机器上切换哪个集群的，命令直接都是给你的，照着敲。

设置配置环境：

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

Context

为部署流水线创建一个新的 ClusterRole 并将其绑定到特定的 namespace 的特定 ServiceAccount 。

Task

创建一个名为 deployment-clusterrole 且仅允许创建以下资源类型的新 ClusterRole：

设置配置环境：

```
[student@node-1] $ kubectl c
onfig use-context ek8s
```

Task

将名为 ek8s-node-1 的 node 设置为不可用，并重新调度该 node 上所有运行的 pods。

我的微信是 shadowooom 有在考试模拟环境里做题问题，考试流程问题，都可以问我的。

开始操作

kubectl create clusterrole deployment-clusterrole --verb=create --resource=deployments,statefulsets,daemonsets

kubectl -n app-team1 create serviceaccount cicd-token

题目中写了“限于 namespace app-team1 中”，则创建 rolebinding。没有写的话，则创建 clusterrolebinding。
kubectl -n app-team1 create rolebinding cicd-token-rolebinding --clusterrole=deployment-clusterrole --serviceaccount=app-team1:cicd-token

rolebinding 后面的名字 cicd-token-rolebinding 随便起的，因为题目中没有要求，如果题目中有要求，就不能随便起了。

检查（考试时，可以不检查的）
kubectl -n app-team1 describe rolebinding cicd-token-rolebinding

网友 Wang_jinpen*提供的检查方法：
candidate@node01:~$ kubectl auth can-i create deployment --as system:serviceaccount:app-team1:cicd-token
no
candidate@node01:~$ kubectl auth can-i create deployment -n app-team1 --as system:serviceaccount:app-team1:cicd-token
yes

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$ kubectl create clusterrole deployment-clusterrole --verb=create --resource=deployments,statefulsets,daemonsets
clusterrole.rbac.authorization.k8s.io/deployment-clusterrole created
student@node01:~$ kubectl -n app-team1 create serviceaccount cicd-token
serviceaccount/cicd-token created
student@node01:~$ kubectl -n app-team1 create rolebinding cicd-token-rolebinding --clusterrole=deployment-clusterrole --serviceacc
ount=app-team1:cicd-token
rolebinding.rbac.authorization.k8s.io/cicd-token-rolebinding created
student@node01:~$ kubectl -n app-team1 describe rolebinding cicd-token-rolebinding
Name:         cicd-token-rolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  deployment-clusterrole
Subjects:
  Kind            Name        Namespace
  ----            ----        ---------
  ServiceAccount  cicd-token  app-team1
student@node01:~$
```

```
candidate@node01:~$ kubectl auth can-i create deployment --as system:serviceaccount:app-team1:cicd-token
no
candidate@node01:~$ kubectl auth can-i create deployment -n app-team1 --as system:serviceaccount:app-team1:cicd-token
yes
candidate@node01:~$
```

2、查看 pod 的 CPU

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context k8s

Task
通过 pod label name=cpu-loader，找到运行时占用大量 CPU 的 pod，
并将占用 CPU 最高的 pod 名称写入文件 /opt/KUTR000401/KUTR00401.txt（已存在）。

考点：kubectl top --l 命令的使用

参考链接

没必要参考网址，使用-h 帮助更方便。
kubectl top pod -h

https://kubernetes.io/docs/reference/kubectl/cheatsheet/#interacting-with-running-pods

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context k8s

开始操作

查看 pod 名称 -A 是所有 namespace
kubectl top pod -l name=cpu-loader --sort-by=cpu -A

将 cpu 占用最多的 pod 的 name 写入/opt/test1.txt 文件
echo "查出来的 Pod Name" > /opt/KUTR000401/KUTR00401.txt

检查
cat /opt/KUTR000401/KUTR00401.txt

```
candidate@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
candidate@node01:~$
candidate@node01:~$ kubectl top pod -l name=cpu-loader --sort-by=cpu -A
NAMESPACE   NAME                          CPU(cores)   MEMORY(bytes)
cpu-top     redis-test-ff9fd7d78-dsbf2    1m           3Mi
cpu-top     nginx-host-56c8f5b697-ltdmd   0m           5Mi
cpu-top     test0-75d9886fc7-44b28        0m           8Mi
candidate@node01:~$ echo redis-test-ff9fd7d78-dsbf2 > /opt/KUTR000401/KUTR00401.txt
candidate@node01:~$
candidate@node01:~$ cat /opt/KUTR000401/KUTR00401.txt
redis-test-ff9fd7d78-dsbf2
candidate@node01:~$
```

2.5、yaml 文件格式说明：

（懂的就不用看了，解释给小白听的 ^_^）

```
    spec:
      containers:
      - image: vicuu/nginx:hello
        imagePullPolicy: IfNotPresent
        name: nginx
        ports:
        - name: http
          containerPort: 80
```

1、先要注意的是 yaml 文件，有严格个空格要求的，yaml 里不同内容的层级，是通过空格来区分的。
因为 html 显示的问题，有些内容看起来对齐的不对（但实际上空格数量是对的）。
如果不了解如何使用空格对齐，可以将 html 的 yaml 内容复制粘贴到 Notepad 等记事本软件里，这样看起来更规整。
2、下面略微解释一下这个 yaml 的部分内容。
从上面我们可以看出 image 和 imagePullPolicy 和 name，这三个字段前的空格是一样的，都是 8 个。(image 前的-可以认为也是一个空格)。而他们上面的 containers 前有 6 个空格，
所以从图中你也能看出来 image 和 imagePullPolicy 和 name 属于 containers 的下一层级，而 image 和 imagePullPolicy 和 name 三个是同层级的。同层级的没有上前顺序，所以下面的书写方式也是对的。

```
    spec:
      containers:
      - image: vicuu/nginx:hello
        name: nginx
        imagePullPolicy: IfNotPresent

    spec:
      containers:
      - name: nginx
        image: vicuu/nginx:hello
        imagePullPolicy: IfNotPresent

    spec:
      containers:
      - imagePullPolicy: IfNotPresent
        image: vicuu/nginx:hello
        name: nginx
```

3、鉴于同级别的没有上下顺序，所以你要加入的绿色三行，你可以在 image 和 imagePullPolicy 和 name 随便一个下写。所以下面的写法也都是正确的。

```
    spec:
      containers:
      - image: vicuu/nginx:hello
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
        name: nginx

    spec:
      containers:
      - image: vicuu/nginx:hello
        imagePullPolicy: IfNotPresent
        ports:
        - name: http
          containerPort: 80
        name: nginx

    spec:
      containers:
      - name: nginx
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
        image: vicuu/nginx:hello
```

4、如果还是不明白，我建议花 1 小时，从网上找个 K8S yaml 文件格式说明的文章或者视频看一下。

3、配置网络策略 NetworkPolicy

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context hk8s

Task
在现有的 namespace my-app 中创建一个名为 allow-port-from-namespace 的新 NetworkPolicy。
确保新的 NetworkPolicy 允许 namespace echo 中的 Pods 连接到 namespace my-app 中的 Pods 的 9000 端口。

进一步确保新的 NetworkPolicy:
不允许对没有在监听 端口 9000 的 Pods 的访问
不允许非来自 namespace echo 中的 Pods 的访问

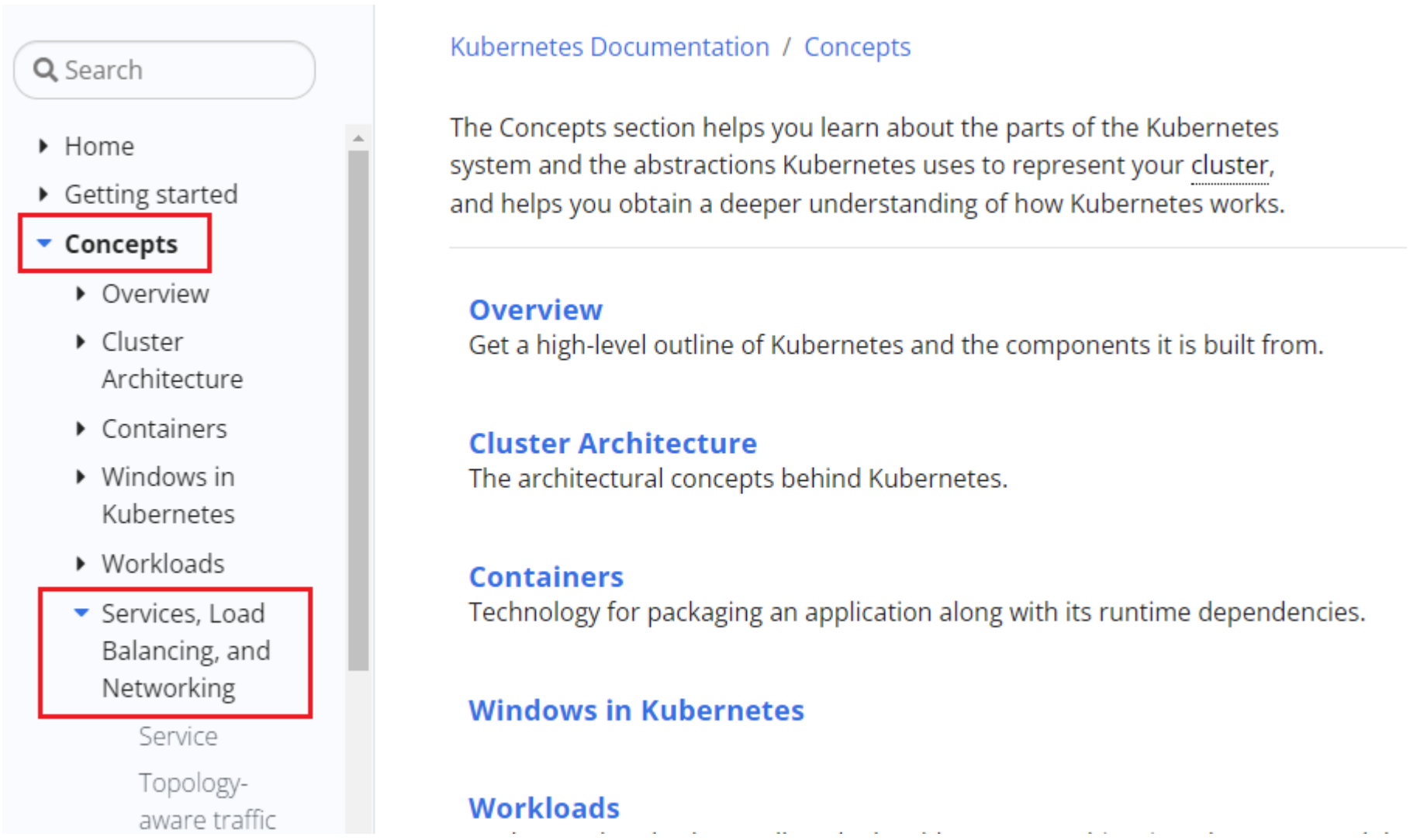
双重否定就是肯定，所以最后两句话的意思就是:
仅允许端口为 9000 的 pod 方法。
仅允许 echo 命名空间中的 pod 访问。

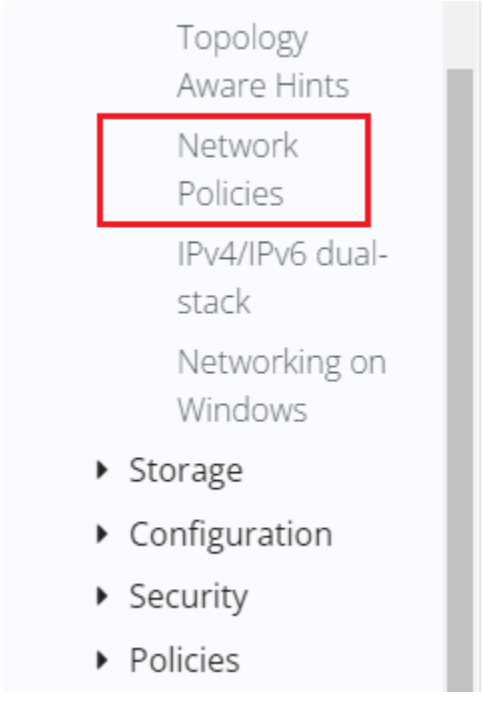
考点：NetworkPolicy 的创建

**参考链接

（需要复制网页内容）

依次点击 Concepts → Services, Load Balancing, and Networking → Network Policies（看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/concepts/services-networking/network-policies/





NetworkPolicy 资源

参阅 NetworkPolicy 来了解资源的完整定义。

下面是一个 NetworkPolicy 的示例:

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```

解答

考试时务必执行，切换集群。模拟环境中不需要执行。

```
# kubectl config use-context hk8s
```

参考官方文档，拷贝 yaml 文件内容，并修改。

这里要注意，模拟题和真题截图里都有提到 echo 这个 namespace，但是和真题的截图比较，你会发现，两个 echo 出现的位置不同，一个作为访问者，一个作为被访问者。

所以不要搞混了。他们其实只是个名字而已，叫什么都无所谓，但要分清访问和被访问。

开始操作

查看所有 ns 的标签 label
kubectl get ns --show-labels
如果访问者的 namespace 没有标签 label，则需要手动打一个。如果有一个独特的标签 label，则也可以直接使用。
kubectl label ns echo project=echo

```
student@node01:~$ kubectl config use-context hk8s
error: no context exists with the name: "hk8s"
student@node01:~$ kubectl get ns --show-labels
NAME              STATUS   AGE    LABELS
app-team1         Active   127d   kubernetes.io/metadata.name=app-team1
cpu-top           Active   127d   kubernetes.io/metadata.name=cpu-top
default           Active   128d   kubernetes.io/metadata.name=default
echo              Active   127d   kubernetes.io/metadata.name=echo
ing-internal      Active   127d   kubernetes.io/metadata.name=ing-internal
ingress-nginx     Active   127d   kubernetes.io/metadata.name=ingress-nginx
internal          Active   127d   kubernetes.io/metadata.name=internal
kube-node-lease   Active   128d   kubernetes.io/metadata.name=kube-node-lease
kube-public       Active   128d   kubernetes.io/metadata.name=kube-public
kube-system       Active   128d   kubernetes.io/metadata.name=kube-system
my-app            Active   127d   kubernetes.io/metadata.name=my-app
student@node01:~$ kubectl label ns echo project=echo
namespace/echo labeled
student@node01:~$ kubectl get ns echo --show-labels
NAME   STATUS   AGE    LABELS
echo   Active   127d   kubernetes.io/metadata.name=echo,project=echo
student@node01:~$
```

编写一个 yaml 文件
vim networkpolicy.yaml
#注意 :set paste，防止 yaml 文件空格错序。

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-port-from-namespace
  namespace: my-app     #被访问者的命名空间
spec:
  podSelector:          #这两行必须要写，或者也可以写成一行为 podSelector: {}
    matchLabels: {}     #  注意 matchLabels:与{}之间有一个空格
  policyTypes:
  - Ingress     #策略影响入栈流量
  ingress:
  - from:     #允许流量的来源
    - namespaceSelector:
        matchLabels:
          project: echo       #访问者的命名空间的标签 label
    #- podSelector: {}       #注意，这个不写。如果 ingress 里也写了- podSelector: {}，则会导致 my-app 中的 pod 可以访问 my-app 中 pod 的 9000 了，这样不满足题目要求不允许非来自 namespace echo 中的 Pods 的访问。
    ports:
    - protocol: TCP
      port: 9000    #被访问者公开的端口
```

创建
kubectl apply -f networkpolicy.yaml

```
student@node01:~$ vim networkpolicy.yaml
student@node01:~$ cat networkpolicy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-port-from-namespace
  namespace: my-app
spec:
  podSelector:
    matchLabels: {}
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          project: echo
    ports:
    - protocol: TCP
      port: 9000

student@node01:~$ kubectl apply -f networkpolicy.yaml
networkpolicy.networking.k8s.io/allow-port-from-namespace created
student@node01:~$
```

检查

kubectl describe networkpolicy -n my-app

```
student@node01:~$ kubectl describe networkpolicy -n my-app
Name:         allow-port-from-namespace
Namespace:    my-app
Created on:   2022-07-01 13:30:38 +0800 CST
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: 9000/TCP
    From:
      NamespaceSelector: project=echo
  Not affecting egress traffic
  Policy Types: Ingress
student@node01:~$
```

4、暴露服务 service

考题

设置配置环境：

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

Task

请重新配置现有的 deployment front-end 以及添加名为 http 的端口规范来公开现有容器 nginx 的端口 80/tcp。

创建一个名为 front-end-svc 的新 service，以公开容器端口 http。

配置此 service，以通过各个 Pod 所在的节点上的 NodePort 来公开他们。

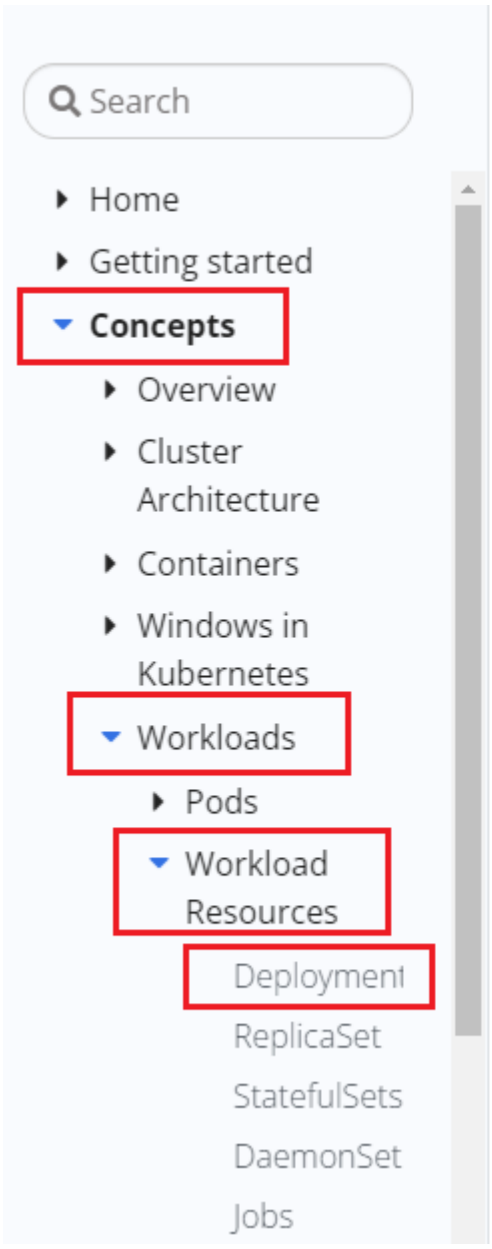
考点：将现有的 deploy 暴露成 nodeport 的 service。

**参考链接

（需要复制网页内容）

依次点击 Concepts → Workloads → Workload Resources → Deployments（看不懂英文的，可右上角翻译成中文）

https://kubernetes.io/zh-cn/docs/concepts/workloads/controllers/deployment/



The Concepts section helps you learn about the parts of the Kubernetes system and the abstractions Kubernetes uses to represent your cluster, and helps you obtain a deeper understanding of how Kubernetes works.

Overview
Get a high-level outline of Kubernetes and the components it is built from.

Cluster Architecture
The architectural concepts behind Kubernetes.

Containers
Technology for packaging an application along with its runtime dependencies.

Windows in Kubernetes

Workloads
Understand Pods, the smallest deployable compute object in Kubernetes, and the higher-level abstractions that help you to run them.

创建 Deployment

下面是 Deployment 示例。其中创建了一个 ReplicaSet，负责启动三个 `nginx` Pods：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

如果protocol: TCP确实忘记怎么写了，
不写也是可以的。因为默认就是TCP。

但是端口的名字 - name: http必须要加上。

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context k8s

开始操作

检查 deployment 信息，并记录 SELECTOR 的 Lable 标签，这里是 app=front-end
kubectl get deployment front-end -o wide

```
candidate@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
candidate@node01:~$
candidate@node01:~$ kubectl get deployment front-end -o wide
NAME        READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES              SELECTOR
front-end   1/1     1            1           22d   nginx        vicuu/nginx:hello   app=front-end
candidate@node01:~$
```

参考官方文档，按照需要 edit deployment，添加端口信息
kubectl edit deployment front-end
#注意 :set paste，防止 yaml 文件空格错序。

```
    spec:
      containers:
      - image: vicuu/nginx:hello
        imagePullPolicy: IfNotPresent
        name: nginx        #找到此位置。下文会简单说明一下 yaml 文件的格式，不懂 yaml 格式的，往下看。
        ports:             #添加这 4 行
        - name: http
          containerPort: 80
          protocol: TCP
```

```
candidate@node01:~$ kubectl edit deployment front-end
deployment.apps/front-end edited
candidate@node01:~$
```

```
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: front-end
    spec:
      containers:
      - image: vicuu/nginx:hello
        imagePullPolicy: IfNotPresent
        name: nginx
        ports:
        - name: http
          containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
```

暴露对应端口
kubectl expose deployment front-end --type=NodePort --port=80 --target-port=80 --name=front-end-svc
#注意考试中需要创建的是 NodePort，还是 ClusterIP。如果是 ClusterIP，则应为--type=ClusterIP
#--port 是 service 的端口号，--target-port 是 deployment 里 pod 的容器的端口号。

```
student@node01:~$ kubectl expose deployment front-end --type=NodePort --port=80 --target-port=80 --name=front-end-svc
service/front-end-svc exposed
student@node01:~$
```

暴露服务后，检查一下 service 的 selector 标签是否正确，这个要与 deployment 的 selector 标签一致的。
kubectl get svc front-end-svc -o wide
kubectl get deployment front-end -o wide

```
candidate@node01:~$ kubectl get svc front-end-svc -o wide
NAME            TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE     SELECTOR
front-end-svc   NodePort   10.109.159.3   <none>        80:32710/TCP   2m47s   app=front-end
candidate@node01:~$
candidate@node01:~$ kubectl get deployment front-end -o wide
NAME        READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES              SELECTOR
front-end   1/1     1            1           22d   nginx        vicuu/nginx:hello   app=front-end
candidate@node01:~$
```

如果你 kubectl expose 暴露服务后，发现 service 的 selector 标签是空的<none>，或者不是 deployment 的，如下图这样：

```
candidate@node01:~$ kubectl get svc front-end-svc -o wide
NAME            TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE     SELECTOR
front-end-svc   NodePort   10.109.159.3   <none>        80:32710/TCP   9m47s   <none>
candidate@node01:~$
candidate@node01:~$ kubectl get deployment front-end -o wide
NAME        READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES              SELECTOR
front-end   1/1     1            1           23d   nginx        vicuu/nginx:hello   app=front-end
candidate@node01:~$
```

则需要编辑此 service，手动添加标签。（模拟环境里暴露服务后，selector 标签是正确的。但是考试时，有时 service 的 selector 标签是 none）

kubectl edit svc front-end-svc

在 ports 这一小段下面添加 selector 标签

```
  selector:
    app: front-end      #注意 yaml 里是写冒号，而不是等号，不是 app=front-end。
```

```
spec:
  clusterIP: 10.109.159.3
  clusterIPs:
  - 10.109.159.3
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - nodePort: 32710
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: front-end
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

确保 service 的 selector 标签与 deployment 的 selector 标签一致。

```
candidate@node01:~$ kubectl edit svc front-end-svc
service/front-end-svc edited
candidate@node01:~$ kubectl get svc front-end-svc -o wide
NAME            TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE     SELECTOR
front-end-svc   NodePort   10.109.159.3   <none>        80:32710/TCP   8m40s   app=front-end
candidate@node01:~$
candidate@node01:~$ kubectl get deployment front-end -o wide
NAME        READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES              SELECTOR
front-end   1/1     1            1           23d   nginx        vicuu/nginx:hello   app=front-end
candidate@node01:~$
```

最后 curl 检查

kubectl get pod,svc -o wide

curl 所在的 node 的 ip 或主机名:30938

curl svc 的 ip 地址:80

（注意，只能 curl 通 svc 的 80 端口，但是无法 ping 通的。）

```
student@node01:~$ kubectl get pod,svc -o wide
NAME                                READY   STATUS    RESTARTS       AGE    IP            NODE     NOMINATED NODE   READINESS GATES
pod/11-factor-app                   1/1     Running   12 (38h ago)   127d   10.244.2.64   node02   <none>           <none>
pod/foo                             1/1     Running   12 (38h ago)   127d   10.244.1.52   node01   <none>           <none>
pod/front-end-dd8dd7c-ghxb9         1/1     Running   0              118s   10.244.1.56   node01   <none>           <none>
pod/presentation-5649f596b9-jxfnn   1/1     Running   12 (38h ago)   127d   10.244.2.63   node02   <none>           <none>

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE    SELECTOR
service/front-end-svc   NodePort    10.104.247.217   <none>        80:30938/TCP   94s    app=front-end
service/kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP        128d   <none>
student@node01:~$
student@node01:~$ curl node01:30938
Hello World ^_^
student@node01:~$ curl 10.104.247.217:80
Hello World ^_^
student@node01:~$
```

考试时，如果 curl 不通，简单排错后也不通，就不要过于纠结，继续往下做题即可。因为部分同学反馈 curl 不通，不清楚是否为考试集群环境的问题。只要确保都做对了，即使 curl 不通，也最多扣几分而已，是有其他步骤分的。

5、创建 Ingress

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context k8s

Task
如下创建一个新的 nginx Ingress 资源：
名称: ping
Namespace: ing-internal
使用服务端口 5678 在路径 /hello 上公开服务 hello

可以使用以下命令检查服务 hello 的可用性，该命令应返回 hello：
curl -kL <INTERNAL_IP>/hello

考点：Ingress 的创建

**参考链接

（需要复制网页内容）

依次点击 Concepts → Services, Load Balancing, and Networking → Ingress （看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/concepts/services-networking/ingress/

The Concepts section helps you learn about the parts of the Kubernetes system and the abstractions Kubernetes uses to represent your cluster, and helps you obtain a deeper understanding of how Kubernetes works.

Overview
Get a high-level outline of Kubernetes and the components it is built from.

Cluster Architecture
The architectural concepts behind Kubernetes.

Containers
Technology for packaging an application along with its runtime dependencies.

Windows in Kubernetes

Workloads
Understand Pods, the smallest deployable compute object in Kubernetes, and the higher-level abstractions that help you to run them.

Services, Load Balancing, and Networking
Concepts and resources behind networking in Kubernetes.

Storage
Ways to provide both long-term and temporary storage to Pods in your cluster.

Configuration
Resources that Kubernetes provides for configuring Pods.

Ingress 资源

一个最小的 Ingress 资源示例：

service/netwc

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Ingress 需要指定 `apiVersion`、`kind`、`metadata`和 `spec` 字段。 Ingress 对象的命名必须是合法的 DNS 子域名名称。 关于如何使用配置文件，请参见部署应用、 配置容器、 管理资源。 Ingress 经常使用注解（annotations）来配置一些选项，具体取决于 Ingress 控制器，例如重写目标注解。 不同的 Ingress 控制器支持不同的注解。 查看你所选的 Ingress 控制器的文档，以了解其支持哪些注解。

Ingress 规约 提供了配置负载均衡器或者代理服务器所需的所有信息。 最重要的是，其中包含与所有传入请求匹配的规则列表。 Ingress 资源仅支持用于转发 HTTP(S) 流量的规则。

如果 `ingressClassName` 被省略，那么你应该定义一个默认 Ingress 类。

点开

有一些 Ingress 控制器不需要定义默认的 `IngressClass` 。比如：Ingress-NGINX 控制器可以通过参数 `--watch-ingress-without-class` 来配置。 不过仍然推荐 按下文所示来设置默认的 `IngressClass` 。

默认 Ingress 类

你可以将一个特定的 IngressClass 标记为集群默认 Ingress 类。 将一个 IngressClass 资源的 `ingressclass.kubernetes.io/is-default-class` 注解设置为 `true` 将确保新的未指定 `ingressClassName` 字段的 Ingress 能够分配为这个默认的 IngressClass.

> **注意：** 如果集群中有多个 IngressClass 被标记为默认，准入控制器将阻止创建新的未指定 `ingressClassName` 的 Ingress 对象。 解决这个问题只需确保集群中最多只能有一个 IngressClass 被标记为默认。

有一些 Ingress 控制器不需要定义默认的 `IngressClass`。比如：Ingress-NGINX 控制器可以通过参数 `--watch-ingress-without-class` 来配置。 不过仍然推荐 设置默认的 `IngressClass`。

service/networkin

```yaml
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/component: controller
  name: nginx-example
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: k8s.io/ingress-nginx
```

解答

考试时务必执行，切换集群。模拟环境中不需要执行。

```
# kubectl config use-context k8s
```

开始操作

方法 1：（推荐）

拷贝官文的 yaml 案例，修改相关参数即可

```
vim ingress.yaml
#注意 :set paste，防止 yaml 文件空格错序。

apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/component: controller
  name: nginx-example      #考试时，默认是没有 ingressClassName 的，所以我们要先手动建一个 ingressClassName，命名就为 nginx-example 吧
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: k8s.io/ingress-nginx
---       #这 3 个---，必须要写，不能省略
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ping
  namespace: ing-internal
  annotations:
```

```
    nginx.ingress.kubernetes.io/rewrite-target: /          #因为考试环境有多套，不清楚具体抽中的是哪套。在 1.26 的考试里，先写上这行，如果 apply 时报错需要指定域名，则注释这行再 apply，就成功了。
spec:
  ingressClassName: nginx-example   #这里调用上面新建的 ingressClassName 为 nginx-example
  rules:
  - http:
      paths:
      - path: /hello
        pathType: Prefix
        backend:
          service:
            name: hello
            port:
              number: 5678
```

创建

kubectl apply -f ingress.yaml

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$ vim ingress.yaml
student@node01:~$ cat ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ping
  namespace: ing-internal
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /hello
        pathType: Prefix
        backend:
          service:
            name: hello
            port:
              number: 5678

student@node01:~$ kubectl apply -f ingress.yaml
ingress.networking.k8s.io/ping created
student@node01:~$
```

（截图为原先的，截图里的内容不对，请参考上面的问题答案）

最后 curl 检查

通过 get ingress 查看 ingress 的内外 IP，然后通过提供的 curl 测试 ingress 是否正确。

做完题后，略等 3 分钟，再检查，否则可能还没获取 IP 地址。或者可以先去做别的题，等都做完了，再回来检查这道题，一下，记得回来检查时，先使用 kubectl config use-context k8s 切换到此集群。

kubectl get ingress -n ing-internal

curl ingress 的 ip 地址/hello

```
student@node01:~$ kubectl get ingress -n ing-internal
NAME   CLASS    HOSTS   ADDRESS         PORTS   AGE
ping   <none>   *       10.110.175.39   80      38s
student@node01:~$
student@node01:~$ curl 10.110.175.39/hello
Hello World ^_^
student@node01:~$
```

方法 2：（暂不推荐，考试时有问题）

感谢网友 Marlin 提供的另一个方法

kubectl create ingress ping --rule=/hello=hello:5678 --class=nginx --annotation=nginx.ingress.kubernetes.io/rewrite-target=/ -n ing-internal

kubectl get ingress -n ing-internal

curl 10.110.175.39/hello

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$ kubectl create ingress ping --rule=/hello=hello:5678 --class=nginx --annotation=nginx.ingress.kubernetes.io/rewrit
e-target=/ -n ing-internal
ingress.networking.k8s.io/ping created
student@node01:~$ kubectl get ingress -n ing-internal
NAME   CLASS   HOSTS   ADDRESS         PORTS   AGE
ping   nginx   *       10.110.175.39   80      30s
student@node01:~$ curl 10.110.175.39/hello
Hello World ^_^
student@node01:~$
```

```
student@node01:~$ kubectl get ingress ping -n ing-internal -o yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
  creationTimestamp: "2022-07-01T05:40:39Z"
  generation: 1
  name: ping
  namespace: ing-internal
  resourceVersion: "66889"
  uid: 5517800b-037b-41dc-9151-ad2acaa517df
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - backend:
          service:
            name: hello
            port:
              number: 5678
        path: /hello
        pathType: Exact
status:
  loadBalancer:
    ingress:
    - ip: 10.110.175.39
student@node01:~$
```

6、扩容 deployment 副本数量

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context k8s

Task
将 deployment presentation 扩展至 4 个 pods

参考链接

没必要参考网址，使用-h 帮助更方便。
kubectl scale deployment -h

https://kubernetes.io/zh-cn/docs/tasks/run-application/scale-stateful-set/

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context k8s

开始操作

先检查一下现有的 pod 数量（可不检查）
kubectl get deployments presentation -o wide
kubectl get pod -l app=presentation

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$ kubectl get deployments presentation -o wide
NAME           READY   UP-TO-DATE   AVAILABLE   AGE    CONTAINERS   IMAGES              SELECTOR
presentation   1/1     1            1           127d   nginx        vicuu/nginx:hello   app=presentation
student@node01:~$ kubectl get pod -l app=presentation
NAME                            READY   STATUS    RESTARTS       AGE
presentation-5649f596b9-jxfnn   1/1     Running   12 (38h ago)   127d
student@node01:~$
```

扩展成 4 个
kubectl scale deployment presentation --replicas=4

```
student@node01:~$ kubectl scale deployment presentation --replicas=4
deployment.apps/presentation scaled
student@node01:~$
```

检查，如果显示 ContainerCreating，则表示你的集群有问题，99%是因为你对 VMware 快照的错误操作。
请务必确保你的 3 台虚拟机是还原到一个关机状态的快照（如初始化快照）。而不是还原到一个开机状态下打的快照!
kubectl get deployments presentation -o wide
kubectl get pod -l app=presentation

```
student@node01:~$ kubectl get deployments presentation -o wide
NAME           READY   UP-TO-DATE   AVAILABLE   AGE    CONTAINERS   IMAGES              SELECTOR
presentation   4/4     4            4           127d   nginx        vicuu/nginx:hello   app=presentation
student@node01:~$ kubectl get pod -l app=presentation
NAME                            READY   STATUS    RESTARTS       AGE
presentation-5649f596b9-9z758   1/1     Running   0              23s
presentation-5649f596b9-jp2m5   1/1     Running   0              23s
presentation-5649f596b9-jxfnn   1/1     Running   12 (38h ago)   127d
presentation-5649f596b9-txflv   1/1     Running   0              23s
student@node01:~$
```

7、调度 pod 到指定节点

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context k8s

Task
按如下要求调度一个 pod：
名称：nginx-kusc00401
Image：nginx
Node selector：disk=ssd

考点：nodeSelect 属性的使用

**参考链接

（需要复制网页内容）

依次点击 Tasks → Configure Pods and Containers → Assign Pods to Nodes （看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/tasks/configure-pod-container/assign-pods-nodes/

Ingress

FEATURE STATE: `Kubernetes v1.19 [stable]`

An API object that manages external access to the services in a cluster, typically HTTP.

Ingress may provide load balancing, SSL termination and name-based virtual hosting.

Terminology

For clarity, this guide defines the following terms:

- Node: A worker machine in Kubernetes, part of a cluster.
- Cluster: A set of Nodes that run containerized applications managed by Kubernetes. For this example, and in most common Kubernetes deployments, nodes in the cluster are not part of the public internet.

Terminology

For clarity, this guide defines the following terms:

- Node: A worker machine in Kubernetes, part of a cluster.
- Cluster: A set of Nodes that run containerized applications managed by Kubernetes. For this example, and in most common Kubernetes deployments, nodes in the cluster are not part of the public internet.
- Edge router: A router that enforces the firewall policy for your cluster. This could be a gateway managed by a cloud provider or a physical piece of hardware.

创建一个调度到你选择的节点的 pod

此 Pod 配置文件描述了一个拥有节点选择器 `disktype: ssd` 的 Pod。这表明该 Pod 将被调度到 有 `disktype=ssd` 标签的节点。

pods/

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
```

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context k8s

开始操作

先检查一下是否有这个 pod，应该是没有创建的，所以需要创建
kubectl get pod -A|grep nginx-kusc00401

确保 node 有这个 labels，考试时，检查一下就行，应该已经提前设置好了 labels。
kubectl get nodes --show-labels|grep 'disk=ssd'

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$ kubectl get pod -A|grep nginx-kusc00401
student@node01:~$ kubectl get nodes --show-labels|grep 'disk=ssd'
node01     Ready    <none>                 128d   v1.23.1   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,disk=ssd,kuber
netes.io/arch=amd64,kubernetes.io/hostname=node01,kubernetes.io/os=linux
student@node01:~$
```

如果没有设置，则使用 kubectl label nodes node01 disk=ssd 命令来手动自己设置。

拷贝官文案例，修改下 pod 名称和镜像，删除多余的部分即可
vim pod-disk-ssd.yaml
#注意 :set paste，防止 yaml 文件空格错序。

```
apiVersion: v1
kind: Pod
metadata:
   name: nginx-kusc00401
spec:
   containers:
   - name: nginx
      image: nginx
      imagePullPolicy: IfNotPresent       #这句的意思是，如果此 image 已经有了，则不重新下载。考试时写不写这个都是可以的。
   nodeSelector:
      disk: ssd
```

创建

kubectl apply -f pod-disk-ssd.yaml

```
student@node01:~$ vim pod-disk-ssd.yaml
student@node01:~$ cat pod-disk-ssd.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-kusc00401
spec:
  containers:
  - name: nginx
    image: nginx
  nodeSelector:
    disk: ssd

student@node01:~$ kubectl apply -f pod-disk-ssd.yaml
pod/nginx-kusc00401 created
student@node01:~$
```

检查

kubectl get pod nginx-kusc00401 -o wide

```
student@node01:~$ kubectl get pod nginx-kusc00401 -o wide
NAME             READY   STATUS    RESTARTS   AGE   IP            NODE     NOMINATED NODE   READINESS GATES
nginx-kusc00401   1/1     Running   0          38s   10.244.1.61   node01   <none>           <none>
student@node01:~$
```

8、查看可用节点数量

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context k8s

Task
检查有多少 nodes 已准备就绪（不包括被打上 Taint：NoSchedule 的节点），
并将数量写入 /opt/KUSC00402/kusc00402.txt

考点：检查节点角色标签，状态属性，污点属性的使用

参考链接

没必要参考网址，使用-h 帮助更方便。

kubectl -h

https://kubernetes.io/zh-cn/docs/concepts/scheduling-eviction/taint-and-toleration/

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context k8s

开始操作

grep 的-i 是忽略大小写，grep -v 是排除在外，grep -c 是统计查出来的条数。
kubectl describe nodes | grep -i Taints | grep -vc NoSchedule
echo "查出来的数字" > /opt/KUSC00402/kusc00402.txt

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$ kubectl describe nodes | grep -i Taints | grep -vc NoSchedule
2
student@node01:~$ echo "2" > /opt/KUSC00402/kusc00402.txt
student@node01:~$
```

其实你直接使用如下命令，就能一眼看出来，几个没有的。

kubectl describe nodes | grep -i Taints

```
student@node01:~$ kubectl describe nodes | grep -i Taints
Taints:             node-role.kubernetes.io/master:NoSchedule
Taints:             <none>
Taints:             <none>
student@node01:~$
```

1个有NoSchedule，2个没有
所以答案是 2

检查

cat /opt/KUSC00402/kusc00402.txt

```
student@node01:~$ cat /opt/KUSC00402/kusc00402.txt
2
student@node01:~$
```

9、创建多容器的 pod

考题

设置配置环境：

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

Task

按如下要求调度一个 Pod：

名称：kucc8

app containers: 2

container 名称/images：

- nginx
- consul

考点：pod 概念

**参考链接

（需要复制网页内容）

依次点击 Concepts → Workloads → Pods （看不懂英文的，可右上角翻译成中文）

https://kubernetes.io/zh-cn/docs/concepts/workloads/pods/

Ingress

FEATURE STATE: `Kubernetes v1.19 [stable]`

An API object that manages external access to the services in a cluster, typically HTTP.

Ingress may provide load balancing, SSL termination and name-based virtual hosting.

Terminology

For clarity, this guide defines the following terms:

- Node: A worker machine in Kubernetes, part of a cluster.
- Cluster: A set of Nodes that run containerized applications managed by Kubernetes. For this example, and in most common Kubernetes deployments, nodes in the cluster are not part of the public internet.

使用 Pod

下面是一个 Pod 示例，它由一个运行镜像 `nginx:1.14.2` 的容器组成。

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

写两个-name和image

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context k8s

可以参考上一题“调度 pod 到指定节点”的 yaml 配置文件。

开始操作

vim pod-kucc.yaml
#注意 :set paste，防止 yaml 文件空格错序。

apiVersion: v1
kind: Pod
metadata:

```
  name: kucc8
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent     #这句的意思是，如果此 image 已经有了，则不重新下载。考试时写不写都可以。但模拟环境里推荐写，pod 启动快。
  - name: consul
    image: consul
    imagePullPolicy: IfNotPresent
```

创建

kubectl apply -f pod-kucc.yaml

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$ vim pod-kucc.yaml
student@node01:~$ cat pod-kucc.yaml
apiVersion: v1
kind: Pod
metadata:
  name: kucc8
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  - name: consul
    image: consul
    imagePullPolicy: IfNotPresent

student@node01:~$ kubectl apply -f pod-kucc.yaml
pod/kucc8 created
student@node01:~$
```

检查

kubectl get pod kucc8

```
student@node01:~$ kubectl get pod kucc8
NAME    READY   STATUS    RESTARTS   AGE
kucc8   2/2     Running   0          39s
student@node01:~$
```

10、创建 PV

考题

设置配置环境：

[candidate@node-1] $ kubectl config use-context hk8s

Task

创建名为 app-config 的 persistent volume，容量为 1Gi，访问模式为 ReadWriteMany。

volume 类型为 hostPath，位于 /srv/app-config

考点：hostPath 类型的 pv

**参考链接

（需要复制网页内容）

依次点击 Tasks → Configure Pods and Containers → Configure a Pod to Use a PersistentVolume for Storage （看不懂英文的，可右上角翻译成中文）

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

This section of the Kubernetes documentation contains pages that show how to do individual tasks. A task page shows how to do a single thing, typically by giving a short sequence of steps.

If you would like to write a task page, see Creating a Documentation Pull Request.

Install Tools
Set up Kubernetes tools on your computer.

Administer a Cluster
Learn common tasks for administering a cluster.

Configure Pods and Containers
Perform common configuration tasks for Pods and containers.

Administer a Cluster
Learn common tasks for administering a cluster.

Configure Pods and Containers
Perform common configuration tasks for Pods and containers.

Monitoring, Logging, and Debugging
Set up monitoring and logging to troubleshoot a cluster, or debug a containerized application.

Manage Kubernetes Objects
Declarative and imperative paradigms for interacting with the Kubernetes API.

Managing Secrets
Managing confidential settings data using Secrets.

Inject Data Into Applications
Specify configuration and other data for the Pods that run your workload.

创建 PersistentVolume

在本练习中，你将创建一个 *hostPath* 类型的 PersistentVolume。 Kubernetes 支持用于在单节点集群上开发和测试的 hostPath 类型的 PersistentVolume。 hostPath 类型的 PersistentVolume 使用节点上的文件或目录来模拟网络附加存储。

在生产集群中，你不会使用 hostPath。 集群管理员会提供网络存储资源，比如 Google Compute Engine 持久盘卷、NFS 共享卷或 Amazon Elastic Block Store 卷。 集群管理员还可以使用 StorageClasses 来设置动态提供存储。

下面是 hostPath PersistentVolume 的配置文件：

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```


解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context k8s

直接从官方复制合适的案例，修改参数，然后设置 hostPath 为 /srv/app-config 即可。

开始操作

vim pv.yaml
#注意 :set paste，防止 yaml 文件空格错序。

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
   name: app-config
   #labels:       #不需要写
      #type: local
spec:
   capacity:
      storage: 1Gi
   accessModes:
      - ReadWriteMany     # 注意，考试时的访问模式可能有 ReadWriteMany 和 ReadOnlyMany 和 ReadWriteOnce，根据题目要求写。
   hostPath:
      path: "/srv/app-config"
```

创建
kubectl apply -f pv.yaml

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$ vim pv.yaml
student@node01:~$ cat pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: app-config
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  hostPath:
    path: "/srv/app-config"

student@node01:~$ kubectl apply -f pv.yaml
persistentvolume/app-config created
student@node01:~$
```

检查

kubectl get pv

RWX 是 ReadWriteMany，RWO 是 ReadWriteOnce。

```
student@node01:~$ kubectl get pv
NAME         CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS      REASON   AGE
app-config   1Gi        RWX            Retain           Available                                      17s
pv01         10Mi       RWO            Retain           Available           csi-hostpath-sc            127d
student@node01:~$
```

11、创建 PVC

考题

设置配置环境：

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

Task

创建一个新的 PersistentVolumeClaim：

名称: pv-volume

Class: csi-hostpath-sc

容量: 10Mi

创建一个新的 Pod，来将 PersistentVolumeClaim 作为 volume 进行挂载：

名称：web-server

Image：nginx:1.16

挂载路径：/usr/share/nginx/html

配置新的 Pod，以对 volume 具有 ReadWriteOnce 权限。

最后，使用 kubectl edit 或 kubectl patch 将 PersistentVolumeClaim 的容量扩展为 70Mi，并记录此更改。

考点：pvc 的创建 class 属性的使用，--record 记录变更

**参考链接

（需要复制网页内容）

依次点击 Tasks → Configure Pods and Containers → Configure a Pod to Use a PersistentVolume for Storage （看不懂英文的，可右上角翻译成中文）

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

This section of the Kubernetes documentation contains pages that show how to do individual tasks. A task page shows how to do a single thing, typically by giving a short sequence of steps.

If you would like to write a task page, see Creating a Documentation Pull Request.

Install Tools
Set up Kubernetes tools on your computer.

Administer a Cluster
Learn common tasks for administering a cluster.

Configure Pods and Containers
Perform common configuration tasks for Pods and containers.

Administer a Cluster
Learn common tasks for administering a cluster.

Configure Pods and Containers
Perform common configuration tasks for Pods and containers.

Monitoring, Logging, and Debugging
Set up monitoring and logging to troubleshoot a cluster, or debug a containerized application.

Manage Kubernetes Objects
Declarative and imperative paradigms for interacting with the Kubernetes API.

Managing Secrets
Managing confidential settings data using Secrets.

Inject Data Into Applications
Specify configuration and other data for the Pods that run your workload.

创建 PersistentVolumeClaim

下一步是创建一个 PersistentVolumeClaim。 Pod 使用 PersistentVolumeClaim
PersistentVolumeClaim，它请求至少 3 GB 容量的卷， 该卷至少可以为一个节点

下面是 PersistentVolumeClaim 的配置文件：

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

题目没有要求，则不写

根据题目要求写

创建 Pod

下一步是创建一个 Pod， 该 Pod 使用你的 PersistentVolumeClaim 作为存储卷。

下面是 Pod 的 配置文件：

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: task-pv-claim
  containers:
    - name: task-pv-container
      image: nginx
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage
```

题目没有要求的话，这块就不用写。

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context ok8s

根据官方文档复制一个 PVC 配置，修改参数，不确定的地方就是用 kubectl 的 explain 帮助。

开始操作

vim pvc.yaml
#注意 :set paste，防止 yaml 文件空格错序。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
   name: pv-volume       #pvc 名字
spec:
   storageClassName: csi-hostpath-sc
   accessModes:
      - ReadWriteOnce   # 注意，考试时的访问模式可能有 ReadWriteMany 和 ReadOnlyMany 和 ReadWriteOnce，根据题目要求写。
   resources:
      requests:
         storage: 10Mi
```

创建 PVC
kubectl apply -f pvc.yaml
如果上面操作失败，则只能将 3 台虚拟机回退快照，重新做这道题，没有修改的机会。所以确保一次做对。
检查
kubectl get pvc

```
student@node01:~$ kubectl config use-context ok8s
error: no context exists with the name: "ok8s"
student@node01:~$ vim pvc.yaml
student@node01:~$ cat pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-volume
spec:
  storageClassName: csi-hostpath-sc
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Mi

student@node01:~$ kubectl create -f pvc.yaml
persistentvolumeclaim/pv-volume created
student@node01:~$
student@node01:~$ kubectl get pvc
NAME        STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS      AGE
pv-volume   Bound    pv01     10Mi       RWO            csi-hostpath-sc   8s
student@node01:~$
```

vim pvc-pod.yaml
#注意 :set paste，防止 yaml 文件空格错序。

```
apiVersion: v1
kind: Pod
metadata:
   name: web-server
spec:
   volumes:
      - name: task-pv-storage      #绿色的两个 name 要一样。
        persistentVolumeClaim:
           claimName: pv-volume    #这个要使用上面创建的 pvc 名字
   containers:
      - name: nginx
        image: nginx:1.16
        volumeMounts:
           - mountPath: "/usr/share/nginx/html"
             name: task-pv-storage      #绿色的两个 name 要一样。
```

创建
kubectl apply -f pvc-pod.yaml

检查
kubectl get pod web-server

```
student@node01:~$ vim pvc-pod.yaml
student@node01:~$ cat pvc-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: web-server
spec:
  volumes:
    - name: task-pv-storage
      persistentVolumeClaim:
        claimName: pv-volume
  containers:
    - name: nginx
      image: nginx:1.16
      ports:
        - containerPort: 80
          name: "http-server"
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage

student@node01:~$ kubectl create -f pvc-pod.yaml
pod/web-server created
student@node01:~$
student@node01:~$ kubectl get pod web-server
NAME         READY   STATUS    RESTARTS   AGE
web-server   1/1     Running   0          35s
student@node01:~$
```

更改大小，并记录过程。

将 storage: 10Mi 改为 storage: 70Mi （模拟环境里会报错，下面有解释。）

注意是修改上面的 spec:里面的 storage:

kubectl edit pvc pv-volume --record #模拟环境是 nfs 存储，操作时，会有报错忽略即可。考试时用的动态存储，不会报错的。

```
student@node01:~$ kubectl edit pvc pv-volume --record
Flag --record has been deprecated, --record will be removed in the future
error: persistentvolumeclaims "pv-volume" could not be patched: persistentvolumeclaims "pv-volume" is forbidden: only dynamically pr
ovisioned pvc can be resized and the storageclass that provisions the pvc must support resize
You can run `kubectl replace -f /tmp/kubectl-edit-973742952.yaml` to try this update again.
student@node01:~$
```

模拟环境使用的是 nfs 做后端存储，是不支持动态扩容 pvc 的**（:wq 保存退出时，会报错）。**

所以最后一步修改为 70Mi，只是操作一下即可。换成 ceph 做后端存储，可以，但是集群资源太少，无法做 ceph。

```
  uid: 45f7925f-35b1-4c42-a1e6-3b6444b31b07
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 70Mi
  storageClassName: csi-hostpath-sc
  volumeMode: Filesystem
  volumeName: pv01
status:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Mi
  phase: Bound
```


12、查看 pod 日志

考题

设置配置环境：

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

Task

监控 pod foo 的日志并:

提取与错误 RLIMIT_NOFILE 相对应的日志行

将这些日志行写入 /opt/KUTR00101/foo

考点：kubectl logs 命令

参考链接

没必要参考网址，使用-h 帮助更方便。
kubectl -h

https://kubernetes.io/docs/tasks/debug/debug_application/debug_running_pod/#examine_pod_logs

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context k8s

开始操作

kubectl logs foo | grep "RLIMIT_NOFILE" > /opt/KUTR00101/foo

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$ kubectl logs foo | grep "RLIMIT_NOFILE" > /opt/KUTR00101/foo
student@node01:~$
```

检查
cat /opt/KUTR00101/foo

```
student@node01:~$ cat /opt/KUTR00101/foo
2022/07/01 06:03:10 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
student@node01:~$
```

13、使用 sidecar 代理容器日志

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context k8s

Context
将一个现有的 Pod 集成到 Kubernetes 的内置日志记录体系结构中（例如 kubectl logs）。
添加 streaming sidecar 容器是实现此要求的一种好方法。

Task
使用 busybox Image 来将名为 sidecar 的 sidecar 容器添加到现有的 Pod 11-factor-app 中。
新的 sidecar 容器必须运行以下命令：
/bin/sh -c tail -n+1 -f /var/log/11-factor-app.log
使用挂载在/var/log 的 Volume，使日志文件 11-factor-app.log 可用于 sidecar 容器。
除了添加所需要的 volume mount 以外，请勿更改现有容器的规格。

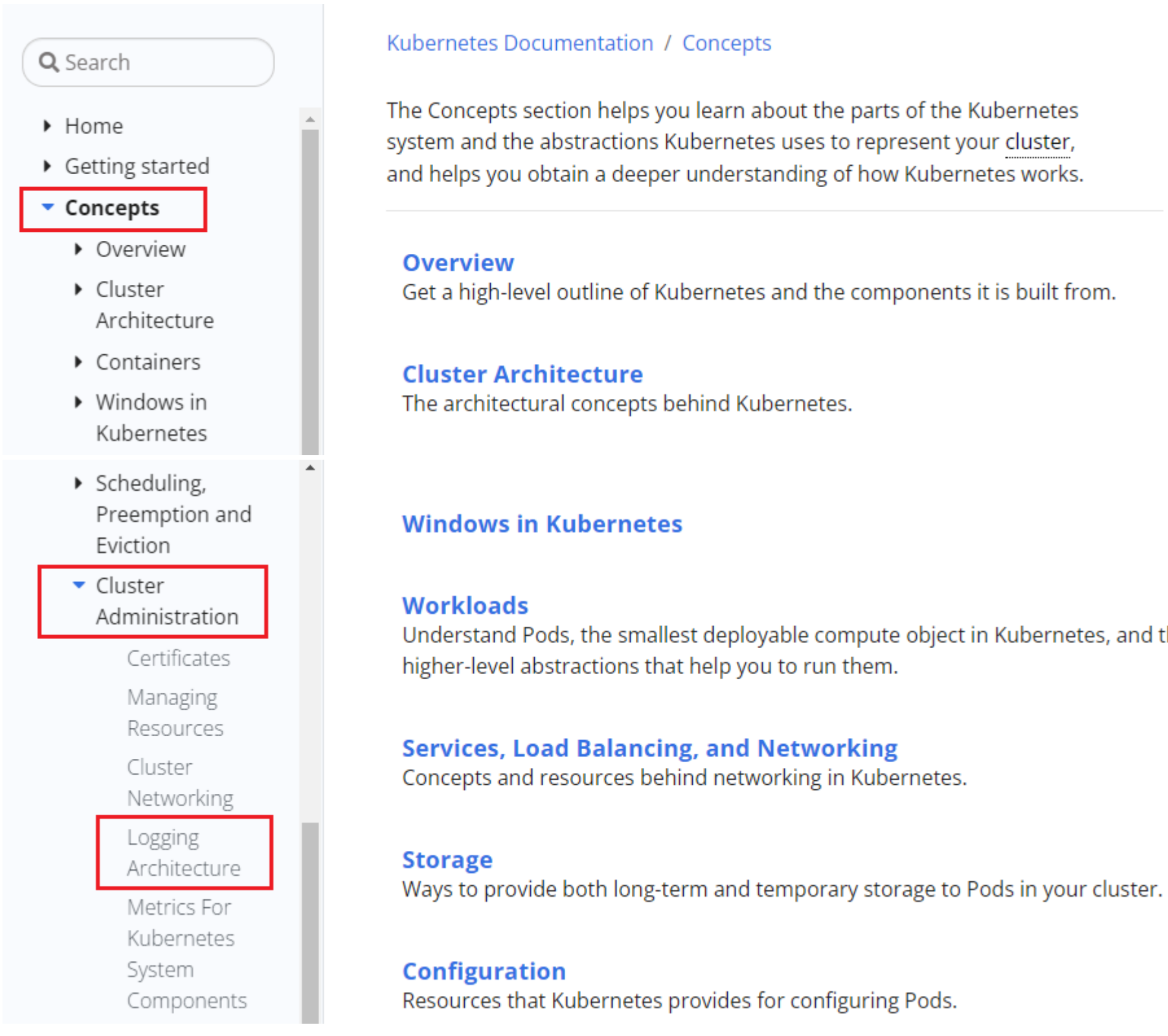
考题翻译成白话，就是：
添加一个名为 sidecar 的边车容器(使用 busybox 镜像)，加到已有的 pod 11-factor-app 中。
确保 sidecar 容器能够输出 /var/log/11-factor-app.log 的信息。
使用 volume 挂载 /var/log 目录，确保 sidecar 能访问 11-factor-app.log 文件

考点：pod 两个容器共享存储卷

**参考链接

（需要复制网页内容）

依次点击 Concepts → Cluster Administration → Logging Architecture （看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/concepts/cluster-administration/logging/



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
  - name: count
    image: busybox
    args:
    - /bin/sh
    - -c
    - >
      i=0;
      while true;
      do
        echo "$i: $(date)" >> /var/log/1.log;
        echo "$(date) INFO $i" >> /var/log/2.log;
        i=$((i+1));
        sleep 1;
      done
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  - name: count-log-1
    image: busybox
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/1.log']
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  - name: count-log-2
    image: busybox
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/2.log']
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  volumes:
  - name: varlog
    emptyDir: {}
```

解答

考试时务必执行，切换集群。模拟环境中不需要执行。

```
# kubectl config use-context k8s
```

通过 kubectl get pod -o yaml 的方法备份原始 pod 信息，删除旧的 pod 11-factor-app
copy 一份新 yaml 文件，添加 一个名称为 sidecar 的容器
新建 emptyDir 的卷，确保两个容器都挂载了 /var/log 目录
新建含有 sidecar 的 pod，并通过 kubectl logs 验证

开始操作

```
#  导出这个 pod 的 yaml 文件
kubectl get pod 11-factor-app -o yaml > varlog.yaml
#  备份 yaml 文件，防止改错了，回退。
cp varlog.yaml varlog-bak.yaml

#  修改 varlog.yaml 文件
vim varlog.yaml
```

```
student@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
student@node01:~$ kubectl get pod 11-factor-app -o yaml > varlog.yaml
student@node01:~$ cp varlog.yaml varlog-bak.yaml
student@node01:~$ vim varlog.yaml
student@node01:~$
```

根据官方文档拷贝需要的信息，在《下面是运行两个边车容器的 Pod 的配置文件》里。
这个题有点难度，实在搞不定的，可以微信 shadowooom 问我。

```
spec:
◦◦◦◦◦◦
    volumeMounts:     #在原配置文件，灰色的这段后面添加
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-4l6w8
      readOnly: true
    - name: varlog       #新加内容
      mountPath: /var/log     #新加内容
  - name: sidecar       #新加内容，注意 name 别写错了
    image: busybox       #新加内容
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/11-factor-app.log']     #新加内容，注意 文件名 别写错了。另外是用逗号分隔的，而题目里是空格。
    volumeMounts:     #新加内容
    - name: varlog     #新加内容
      mountPath: /var/log     #新加内容
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
◦◦◦◦◦◦
  volumes:     #在原配置文件，灰色的这段后面添加。
  - name: kube-api-access-kcjc2
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
          expirationSeconds: 3607
          path: token
      - configMap:
          items:
          - key: ca.crt
            path: ca.crt
          name: kube-root-ca.crt
      - downwardAPI:
          items:
          - fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
            path: namespace
  - name: varlog     #新加内容，注意找好位置。
    emptyDir: {}      #新加内容
```

```
    imagePullPolicy: IfNotPresent
    name: count
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-kcjc2
      readOnly: true
    - name: varlog
      mountPath: /var/log
  - name: sidecar
    image: busybox
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/11-factor-app.log']
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
```

```
    tolerationSeconds: 300
  volumes:
  - name: kube-api-access-kcjc2
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
          expirationSeconds: 3607
          path: token
      - configMap:
          items:
          - key: ca.crt
            path: ca.crt
          name: kube-root-ca.crt
      - downwardAPI:
          items:
          - fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
            path: namespace
  - name: varlog
    emptyDir: {}
status:
  conditions:
  - lastProbeTime: null
```

删除原先的 pod，大于需要等 2 分钟。
kubectl delete pod 11-factor-app
检查一下是否删除了
kubectl get pod 11-factor-app
新建这个 pod
kubectl apply -f varlog.yaml

```
student@node01:~$ kubectl delete pod 11-factor-app
pod "11-factor-app" deleted
student@node01:~$ kubectl get pod 11-factor-app
Error from server (NotFound): pods "11-factor-app" not found
student@node01:~$ kubectl apply -f varlog.yaml
pod/11-factor-app created
student@node01:~$
```

检查
考试时，仅使用第一条检查一下结果即可
kubectl logs 11-factor-app sidecar
kubectl exec 11-factor-app -c sidecar -- tail -f /var/log/11-factor-app.log
kubectl exec 11-factor-app -c count -- tail -f /var/log/11-factor-app.log

```
student@node01:~$ kubectl logs 11-factor-app sidecar
Fri Jul  1 06:16:44 UTC 2022 INFO 0
Fri Jul  1 06:16:45 UTC 2022 INFO 1
Fri Jul  1 06:16:46 UTC 2022 INFO 2
Fri Jul  1 06:16:47 UTC 2022 INFO 3
Fri Jul  1 06:16:48 UTC 2022 INFO 4
```

```
student@node01:~$ kubectl exec 11-factor-app -c sidecar -- tail -f /var/log/11-factor-app.log
Fri Jul  1 06:17:19 UTC 2022 INFO 35
Fri Jul  1 06:17:20 UTC 2022 INFO 36
Fri Jul  1 06:17:21 UTC 2022 INFO 37
Fri Jul  1 06:17:22 UTC 2022 INFO 38
Fri Jul  1 06:17:23 UTC 2022 INFO 39
Fri Jul  1 06:17:24 UTC 2022 INFO 40
```

```
student@node01:~$ kubectl exec 11-factor-app -c count -- tail -f /var/log/11-factor-app.log
Fri Jul  1 06:17:37 UTC 2022 INFO 53
Fri Jul  1 06:17:38 UTC 2022 INFO 54
Fri Jul  1 06:17:39 UTC 2022 INFO 55
Fri Jul  1 06:17:40 UTC 2022 INFO 56
Fri Jul  1 06:17:41 UTC 2022 INFO 57
```

14、升级集群

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context mk8s

Task
现有的 Kubernetes 集群正在运行版本 1.26.0。**仅将 master 节点**上的所有 Kubernetes 控制平面和节点组件升级到版本 1.26.1。

确保在升级之前 drain master 节点，并在升级后 uncordon master 节点。

可以使用以下命令，通过 ssh 连接到 master 节点：
ssh master01
可以使用以下命令，在该 master 节点上获取更高权限：
sudo -i

另外，在主节点上升级 kubelet 和 kubectl。
请不要升级工作节点，etcd，container 管理器，CNI 插件， DNS 服务或任何其他插件。

（注意，考试敲命令时，注意要升级的版本，根据题目要求输入具体的升级版本!!!）

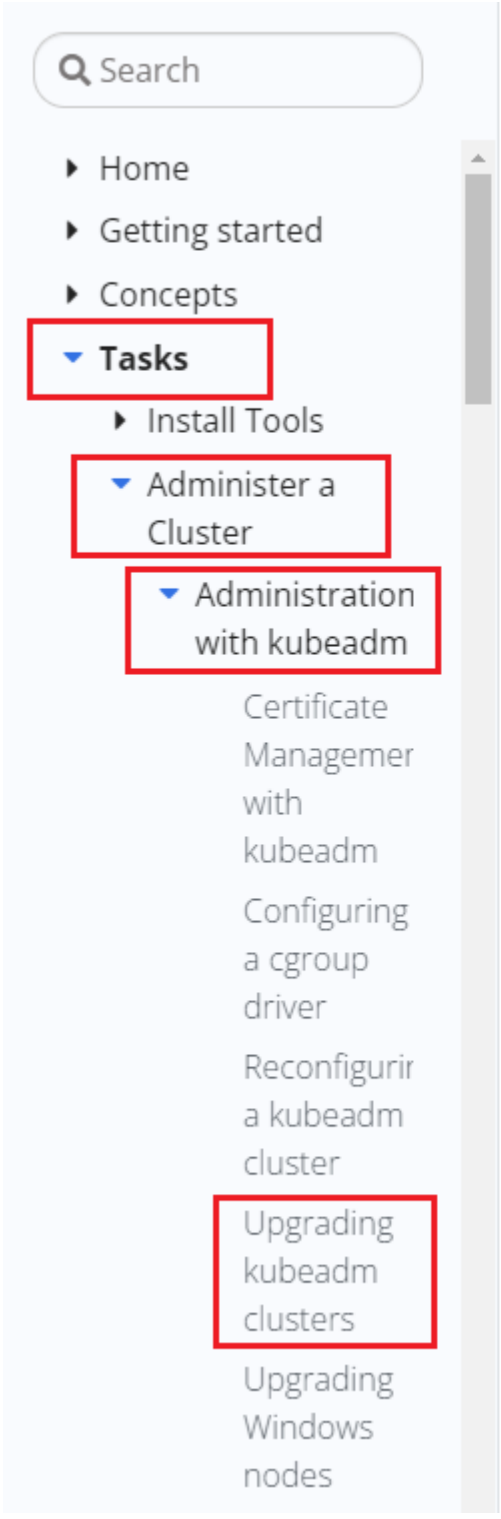
考点：如何离线主机，并升级控制面板和升级节点

**参考链接

没必要参考网址，建议多练习，背过命令就行。
记不清的，可以使用 kubectl -h 来帮助。

如果非要参考，可以按照下面方法。

依次点击 Tasks → Administer a Cluster → Administration with kubeadm → Upgrading kubeadm clusters （看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/



This section of the Kubernetes documentation contains pages that show how to do individual tasks. A task page shows how to do a single thing, typically by giving a short sequence of steps.

If you would like to write a task page, see Creating a Documentation Pull Request.

Install Tools
Set up Kubernetes tools on your computer.

Administer a Cluster
Learn common tasks for administering a cluster.

Configure Pods and Containers
Perform common configuration tasks for Pods and containers.

Monitoring, Logging, and Debugging
Set up monitoring and logging to troubleshoot a cluster, or debug a containerized application.

Manage Kubernetes Objects
Declarative and imperative paradigms for interacting with the Kubernetes API.

Managing Secrets
Managing confidential settings data using Secrets.

腾空节点

- 通过将节点标记为不可调度并腾空节点为节点作升级准备：

```
# 将 <node-to-drain> 替换为你要腾空的控制面节点名称
kubectl drain <node-to-drain> --ignore-daemonsets
```

升级第一个控制面节点

- 升级 kubeadm：

Ubuntu、Debian 或 HypriotOS | CentOS、RHEL 或 Fedora

```
# 用最新的补丁版本号替换 1.23.x-00 中的 x
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm=1.23.x-00 && \
apt-mark hold kubeadm
```

- 验证下载操作正常，并且 kubeadm 版本正确：

```
kubeadm version
```

- 验证升级计划：

```
kubeadm upgrade plan
```

选择要升级到的目标版本，运行合适的命令。例如：

```
# 将 x 替换为你为此次升级所选择的补丁版本号
sudo kubeadm upgrade apply v1.23.x
```

注意添加 --etcd-upgrade=false
可以使用-h帮助，来查找--etcd-upgrade字段

升级 kubelet 和 kubectl

- 升级 kubelet 和 kubectl

Ubuntu、Debian 或 HypriotOS | CentOS、RHEL 或 Fedora

```
# 用最新的补丁版本替换 1.23.x-00 中的 x
apt-mark unhold kubelet kubectl && \
apt-get update && apt-get install -y kubelet=1.23.x-00 kubectl=1.23.x-00 && \
apt-mark hold kubelet kubectl
```

解除节点的保护

- 通过将节点标记为可调度，让其重新上线：

```
# 将 <node-to-drain> 替换为你的节点名称
kubectl uncordon <node-to-drain>
```

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context mk8s

开始操作

kubectl get nodes

```
student@node01:~$ kubectl config use-context mk8s
error: no context exists with the name: "mk8s"
student@node01:~$ kubectl get nodes
NAME       STATUS   ROLES                  AGE    VERSION
master01   Ready    control-plane,master   128d   v1.23.1
node01     Ready    <none>                 128d   v1.23.1
node02     Ready    <none>                 128d   v1.23.1
student@node01:~$
```

cordon 停止调度，将 node 调为 SchedulingDisabled。新 pod 不会被调度到该 node，但在该 node 的旧 pod 不受影响。
drain 驱逐节点。首先，驱逐该 node 上的 pod，并在其他节点重新创建。接着，将节点调为 SchedulingDisabled。
所以 kubectl cordon master01 这条，可写可不写。但我一向做事严谨，能复杂，就绝不简单了事。。。所以就写了。

kubectl cordon master01
kubectl drain master01 --ignore-daemonsets

```
candidate@node01:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   10d   v1.26.0
node01     Ready    <none>          9d    v1.26.0
node02     Ready    <none>          9d    v1.26.0
candidate@node01:~$ kubectl cordon master01
node/master01 cordoned
candidate@node01:~$ kubectl drain master01 --ignore-daemonsets
node/master01 already cordoned
Warning: ignoring DaemonSet-managed Pods: calico-system/calico-node-kpbwv, calico-system/csi-node-driver-4277d,
evicting pod kube-system/coredns-5bbd96d687-7p5mt
evicting pod calico-system/calico-kube-controllers-6b7b9c649d-qjwjx
evicting pod kube-system/coredns-5bbd96d687-2vsp8
pod/calico-kube-controllers-6b7b9c649d-qjwjx evicted
pod/coredns-5bbd96d687-2vsp8 evicted
pod/coredns-5bbd96d687-7p5mt evicted
node/master01 drained
candidate@node01:~$
```

ssh 到 master 节点，并切换到 root 下
ssh master01
sudo -i

```
candidate@node01:~$ ssh master01
```

```
candidate@master01:~$ sudo -i
root@master01:~#
```

apt-get update
apt-cache show kubeadm|grep 1.26.1
apt-get install kubeadm=1.26.1-00

（下面截图是原先旧的，请按照上面最新的命令操作。）
截图是 1.23.1 升级 1.23.2 的

```
root@master01:~# apt-get update
Hit:1 http://mirrors.ustc.edu.cn/ubuntu focal InRelease
Get:2 http://mirrors.ustc.edu.cn/ubuntu focal-updates InRelease [114 kB]
Get:3 http://mirrors.ustc.edu.cn/ubuntu focal-backports InRelease [108 kB]
Get:4 http://mirrors.ustc.edu.cn/ubuntu focal-security InRelease [114 kB]
Get:5 http://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu focal InRelease [57.7 kB]
Get:6 http://mirrors.ustc.edu.cn/kubernetes/apt kubernetes-xenial InRelease [8,993 B]
Get:7 http://mirrors.ustc.edu.cn/ubuntu focal-updates/main amd64 Packages [2,399 kB]
Get:8 http://mirrors.ustc.edu.cn/ubuntu focal-updates/main Translation-en [409 kB]
Get:9 http://mirrors.ustc.edu.cn/ubuntu focal-updates/main amd64 c-n-f Metadata [16.3 kB]
Get:10 http://mirrors.ustc.edu.cn/ubuntu focal-updates/restricted amd64 Packages [1,607 kB]
Get:11 http://mirrors.ustc.edu.cn/ubuntu focal-updates/restricted Translation-en [226 kB]
Get:12 http://mirrors.ustc.edu.cn/ubuntu focal-updates/universe amd64 Packages [1,024 kB]
Get:13 http://mirrors.ustc.edu.cn/ubuntu focal-updates/universe Translation-en [237 kB]
Get:14 http://mirrors.ustc.edu.cn/ubuntu focal-updates/universe amd64 c-n-f Metadata [23.6 kB]
Get:15 http://mirrors.ustc.edu.cn/ubuntu focal-security/main amd64 Packages [1,993 kB]
Get:16 http://mirrors.ustc.edu.cn/ubuntu focal-security/main Translation-en [326 kB]
Get:17 http://mirrors.ustc.edu.cn/ubuntu focal-security/main amd64 c-n-f Metadata [12.2 kB]
Get:18 http://mirrors.ustc.edu.cn/ubuntu focal-security/restricted amd64 Packages [1,495 kB]
Get:19 http://mirrors.ustc.edu.cn/ubuntu focal-security/restricted Translation-en [211 kB]
Get:20 http://mirrors.ustc.edu.cn/ubuntu focal-security/universe amd64 Packages [795 kB]
Get:21 http://mirrors.ustc.edu.cn/ubuntu focal-security/universe Translation-en [154 kB]
Get:22 http://mirrors.ustc.edu.cn/ubuntu focal-security/universe amd64 c-n-f Metadata [17.0 kB]
Get:23 http://mirrors.ustc.edu.cn/ubuntu focal-security/multiverse amd64 Packages [22.9 kB]
Get:24 http://mirrors.ustc.edu.cn/ubuntu focal-security/multiverse Translation-en [5,488 B]
Get:25 http://mirrors.ustc.edu.cn/ubuntu focal-security/multiverse amd64 c-n-f Metadata [528 B]
Get:26 http://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu focal/stable amd64 Packages [24.5 kB]
Fetched 11.4 MB in 2s (7,469 kB/s)
Reading package lists... Done
root@master01:~# apt-cache show kubeadm|grep 1.26.1
Version: 1.26.1-00
Filename: pool/kubeadm_1.26.1-00_amd64_68a7f95a50cafeec7ca7d8b6648253be51f4a79d84a8d9de84aca88a8b2f8c41.deb
root@master01:~# apt-get install kubeadm=1.26.1-00
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be upgraded:
  kubeadm
1 upgraded, 0 newly installed, 0 to remove and 57 not upgraded.
Need to get 9,732 kB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 http://mirrors.ustc.edu.cn/kubernetes/apt kubernetes-xenial/main amd64 kubeadm amd64 1.26.1-00 [9,732 kB]
Fetched 9,732 kB in 0s (22.2 MB/s)
(Reading database ... 110840 files and directories currently installed.)
Preparing to unpack .../kubeadm_1.26.1-00_amd64.deb ...
Unpacking kubeadm (1.26.1-00) over (1.26.0-00) ...
Setting up kubeadm (1.26.1-00) ...
root@master01:~#
```


检查 kubeadm 升级后的版本
kubeadm version

```
root@master01:~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"26", GitVersion:"v1.26.1", GitCommit:"8f94681cd294aa8cfd3407b8191f6c70214973a4", GitTree
State:"clean", BuildDate:"2023-01-18T15:56:50Z", GoVersion:"go1.19.5", Compiler:"gc", Platform:"linux/amd64"}
root@master01:~#
```

验证升级计划，会显示很多可升级的版本，我们关注题目要求升级到的那个版本。

kubeadm upgrade plan

排除 etcd，升级其他的，提示时，输入 y。

kubeadm upgrade apply v1.26.1 **--etcd-upgrade=false**

```
root@master01:~# kubeadm upgrade plan
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[preflight] Running pre-flight checks.
[upgrade] Running cluster health checks
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: v1.26.0
[upgrade/versions] kubeadm version: v1.26.1
[upgrade/versions] Target version: v1.26.1
[upgrade/versions] Latest version in the v1.26 series: v1.26.1

Components that must be upgraded manually after you have upgraded the control plane with 'kubeadm upgrade apply':
COMPONENT   CURRENT       TARGET
kubelet     3 x v1.26.0   v1.26.1

Upgrade to the latest version in the v1.26 series:

COMPONENT                 CURRENT   TARGET
kube-apiserver            v1.26.0   v1.26.1
kube-controller-manager   v1.26.0   v1.26.1
kube-scheduler            v1.26.0   v1.26.1
kube-proxy                v1.26.0   v1.26.1
CoreDNS                   v1.9.3    v1.9.3
etcd                      3.5.6-0   3.5.6-0

You can now apply the upgrade by executing the following command:

        kubeadm upgrade apply v1.26.1

_____


The table below shows the current state of component configs as understood by this version of kubeadm.
Configs that have a "yes" mark in the "MANUAL UPGRADE REQUIRED" column require manual config upgrade or
resetting to kubeadm defaults before a successful upgrade can be performed. The version to manually
upgrade to is denoted in the "PREFERRED VERSION" column.

API GROUP                 CURRENT VERSION   PREFERRED VERSION   MANUAL UPGRADE REQUIRED
kubeproxy.config.k8s.io   v1alpha1          v1alpha1            no
kubelet.config.k8s.io     v1beta1           v1beta1             no
_____
```

```
root@master01:~# kubeadm upgrade apply v1.26.1 --etcd-upgrade=false
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[preflight] Running pre-flight checks.
[upgrade] Running cluster health checks
[upgrade/version] You have chosen to change the cluster version to "v1.26.1"
[upgrade/versions] Cluster version: v1.26.0
[upgrade/versions] kubeadm version: v1.26.1
[upgrade] Are you sure you want to proceed? [y/N]: y
[upgrade/prepull] Pulling images required for setting up a Kubernetes cluster
[upgrade/prepull] This might take a minute or two, depending on the speed of your internet connection
[upgrade/prepull] You can also perform this action in beforehand using 'kubeadm config images pull'
```

升级 kubelet

apt-get install kubelet=1.26.1-00

kubelet --version

```
root@master01:~# apt-get install kubelet=1.26.1-00
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be upgraded:
  kubelet
1 upgraded, 0 newly installed, 0 to remove and 56 not upgraded.
Need to get 20.5 MB of archives.
After this operation, 4,096 B of additional disk space will be used.
Get:1 http://mirrors.ustc.edu.cn/kubernetes/apt kubernetes-xenial/main amd64 kubelet amd64 1.26.1-00 [20.5 MB]
Fetched 20.5 MB in 1s (32.0 MB/s)
(Reading database ... 110840 files and directories currently installed.)
Preparing to unpack .../kubelet_1.26.1-00_amd64.deb ...
Unpacking kubelet (1.26.1-00) over (1.26.0-00) ...
Setting up kubelet (1.26.1-00) ...
root@master01:~# kubelet --version
Kubernetes v1.26.1
root@master01:~#
```

升级 kubectl

apt-get install kubectl=1.26.1-00

kubectl version

```
root@master01:~# apt-get install kubectl=1.26.1-00
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be upgraded:
  kubectl
1 upgraded, 0 newly installed, 0 to remove and 55 not upgraded.
Need to get 10.1 MB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 http://mirrors.ustc.edu.cn/kubernetes/apt kubernetes-xenial/main amd64 kubectl amd64 1.26.1-00 [10.1 MB]
Fetched 10.1 MB in 1s (20.1 MB/s)
(Reading database ... 110840 files and directories currently installed.)
Preparing to unpack .../kubectl_1.26.1-00_amd64.deb ...
Unpacking kubectl (1.26.1-00) over (1.26.0-00) ...
Setting up kubectl (1.26.1-00) ...
root@master01:~# kubectl version
WARNING: This version information is deprecated and will be replaced with the output from kubectl version --short.  Use --output=yaml|jso
n to get the full version.
Client Version: version.Info{Major:"1", Minor:"26", GitVersion:"v1.26.1", GitCommit:"8f94681cd294aa8cfd3407b8191f6c70214973a4", GitTreeSt
ate:"clean", BuildDate:"2023-01-18T15:58:16Z", GoVersion:"go1.19.5", Compiler:"gc", Platform:"linux/amd64"}
Kustomize Version: v4.5.7
Server Version: version.Info{Major:"1", Minor:"26", GitVersion:"v1.26.1", GitCommit:"8f94681cd294aa8cfd3407b8191f6c70214973a4", GitTreeSt
ate:"clean", BuildDate:"2023-01-18T15:51:25Z", GoVersion:"go1.19.5", Compiler:"gc", Platform:"linux/amd64"}
root@master01:~#
```

退出 root，退回到 candidate@master01

exit

退出 master01，退回到 candidate@node01

exit

```
root@master01:~# exit
logout
candidate@master01:~$ exit
logout
Connection to master01 closed.
candidate@node01:~$
```

退回到考试的初始节点

恢复 master01 调度

kubectl uncordon master01

```
candidate@node01:~$ kubectl uncordon master01
node/master01 uncordoned
candidate@node01:~$
```

检查 master01 是否为 Ready

kubectl get node

```
candidate@node01:~$ kubectl get node
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   10d   v1.26.1
node01     Ready    <none>          10d   v1.26.0
node02     Ready    <none>          10d   v1.26.0
candidate@node01:~$
```

15、备份还原 etcd

考题

设置配置环境
此项目无需更改配置环境。但是，在执行此项目之前，请确保您已返回初始节点。
[candidate@master01] $ exit　　#注意，这个之前是在 master01 上，所以要 exit 退到 node01，如果已经是 node01 了，就不要再 exit 了。

Task
首先，为运行在 https://11.0.1.111:2379 上的现有 etcd 实例创建快照并将快照保存到 /var/lib/backup/etcd-snapshot.db
（注意，真实考试中，这里写的是 https://127.0.0.1:2379）
为给定实例创建快照预计能在几秒钟内完成。 如果该操作似乎挂起，则命令可能有问题。用 CTRL + C 来取消操作，然后重试。
然后还原位于/data/backup/etcd-snapshot-previous.db 的现有先前快照。
提供了以下 TLS 证书和密钥，以通过 etcdctl 连接到服务器。

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

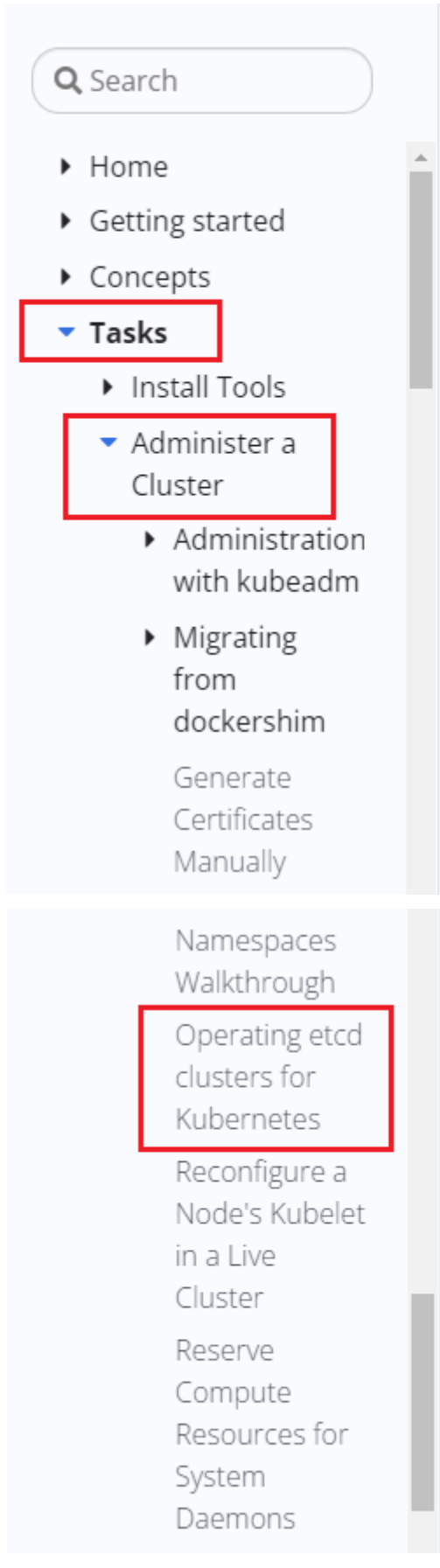
考点：etcd 的备份和还原命令

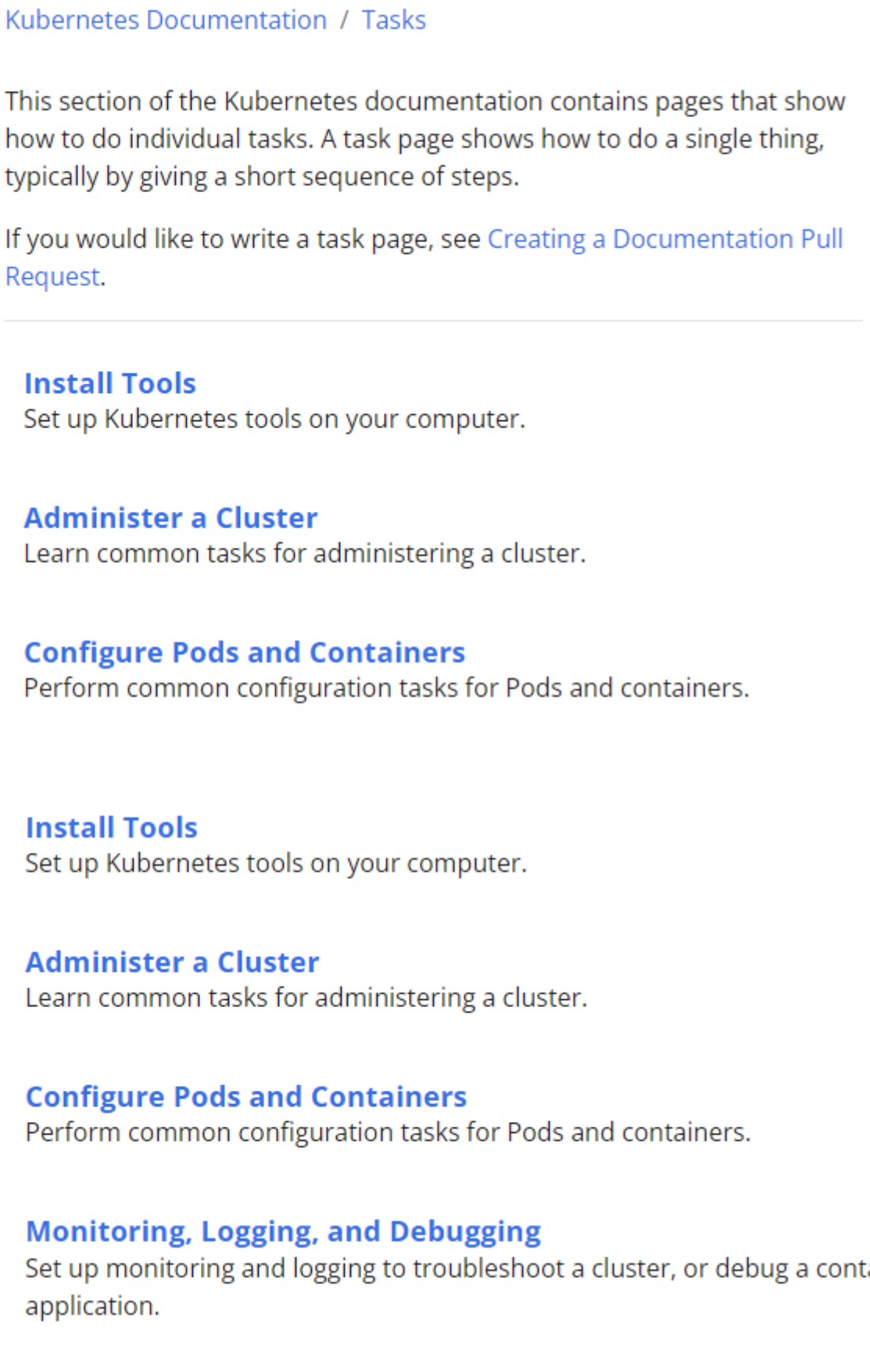
**参考链接

没必要参考网址，建议多练习，背过命令就行。
记不清的，可以使用 etcdctl -h 来帮助，更方便。

如果非要参考，可以按照下面方法。

依次点击 Tasks → Administer a Cluster → Operating etcd clusters for Kubernetes （看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/tasks/administer-cluster/configure-upgrade-etcd/





使用 etcdctl 选项的快照

我们还可以使用 etcdctl 提供的各种选项来拍摄快照。例如：

```
ETCDCTL_API=3 etcdctl -h
```

列出 etcdctl 可用的各种选项。例如，你可以通过指定端点，证书等来拍摄快照，如下所示：

```
ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 \
  --cacert=<trusted-ca-file> --cert=<cert-file> --key=<key-file> \
  snapshot save <backup-file-location>
```

恢复 etcd 集群

etcd 支持从 major.minor 或其他不同 patch 版本的 etcd 进程中获取的快照进行恢复。 还原操作用于恢复失败的集群的数据。

在启动还原操作之前，必须有一个快照文件。它可以是来自以前备份操作的快照文件， 也可以是来自剩余数据目录的快照文件。 例如：

```
ETCDCTL_API=3 etcdctl --endpoints 10.2.0.9:2379 snapshot restore snapshotdb
```

恢复时也可以指定操作选项，例如：

```
ETCDCTL_API=3 etcdctl --data-dir <data-dir-location> snapshot restore snapshotdb
```

解答

注意下面 3 种截然不同的做法：我强烈推荐使用第 1 种方法，稳妥些。
对 k8s 比较熟悉的，也会简单排错的人，可以选择第 2 种或第 3 种方法。

方法 1： （推荐做法）

（非常推荐这样做）
网上流传最多的 etcd 快照还原的步骤:

另外，特别注意，etcd 这道题，在考试时做完后，就不要回头检查或者操作了。因为它是用的前一道题的集群，所以一旦你回头再做时，切换错集群了，且又将 etcd 还原了，反而可能影响别的考题。

注意集群用的是考试时的上一题的集群，所以无需再切换集群了。
但如果事后回来做这道题的话，切记要切换为正确的集群。
kubectl config use-context xxxx

开始操作

确认一下 ssh 终端，是在[candidate@node-1] $ 初始节点。

```
student@node01:~$
```

备份：

```
# 如果不使用 export ETCDCTL_API=3，而使用 ETCDCTL_API=3，则下面每条 etcdctl 命令前都要加 ETCDCTL_API=3。
# 如果执行时，提示 permission denied，则是权限不够，命令最前面加 sudo 即可。
export ETCDCTL_API=3
etcdctl --endpoints=https://11.0.1.111:2379 --cacert="/opt/KUIN00601/ca.crt" --cert="/opt/KUIN00601/etcd-client.crt" --key="/opt/KUIN00601/etcd-client.key"
snapshot save /var/lib/backup/etcd-snapshot.db
```

```
candidate@node01:~$ export ETCDCTL_API=3
candidate@node01:~$ etcdctl --endpoints=https://11.0.1.111:2379 --cacert="/opt/KUIN00601/ca.crt" --cert="/opt/KUIN00601/etcd-client.crt"
--key="/opt/KUIN00601/etcd-client.key" snapshot save /var/lib/backup/etcd-snapshot.db
{"level":"info","ts":"2023-02-14T21:22:33.469+0800","caller":"snapshot/v3_snapshot.go:65","msg":"created temporary db file","path":"/var/
lib/backup/etcd-snapshot.db.part"}
{"level":"info","ts":"2023-02-14T21:22:33.476+0800","logger":"client","caller":"v3@v3.5.6/maintenance.go:212","msg":"opened snapshot stre
am; downloading"}
{"level":"info","ts":"2023-02-14T21:22:33.476+0800","caller":"snapshot/v3_snapshot.go:73","msg":"fetching snapshot","endpoint":"https://1
1.0.1.111:2379"}
{"level":"info","ts":"2023-02-14T21:22:33.603+0800","logger":"client","caller":"v3@v3.5.6/maintenance.go:220","msg":"completed snapshot r
ead; closing"}
{"level":"info","ts":"2023-02-14T21:22:33.615+0800","caller":"snapshot/v3_snapshot.go:88","msg":"fetched snapshot","endpoint":"https://11
.0.1.111:2379","size":"12 MB","took":"now"}
{"level":"info","ts":"2023-02-14T21:22:33.615+0800","caller":"snapshot/v3_snapshot.go:97","msg":"saved","path":"/var/lib/backup/etcd-snap
shot.db"}
Snapshot saved at /var/lib/backup/etcd-snapshot.db
candidate@node01:~$
```

检查：（考试时，这些检查动作，都可以不做）

etcdctl snapshot status /var/lib/backup/etcd-snapshot.db -wtable

```
candidate@node01:~$ etcdctl snapshot status /var/lib/backup/etcd-snapshot.db -wtable
Deprecated: Use `etcdutl snapshot status` instead.

+--------+----------+------------+------------+
|  HASH  | REVISION | TOTAL KEYS | TOTAL SIZE |
+--------+----------+------------+------------+
| 98bb29 |    49632 |       1714 |      12 MB |
+--------+----------+------------+------------+
candidate@node01:~$
```

还原：

考试时，/data/backup/etcd-snapshot-previous.db 的权限应该是只有 root 可读，所以需要使用 sudo 命令。

可以 ll /data/backup/etcd-snapshot-previous.db 检查一下读写权限和属主。

```
student@node01:~$ ll /data/backup/etcd-snapshot-previous.db
-rw-------+ 1 root root 4026400 Feb 24 00:33 /data/backup/etcd-snapshot-previous.db
student@node01:~$
```

不加 sudo 会报错 permission denied,

上面执行了 export ETCDCTL_API=3 了，下面就可以不写 ETCDCTL_API=3d 的，另外还原也是可以不写证书的。

sudo ETCDCTL_API=3 etcdctl --endpoints=https://11.0.1.111:2379 --cacert="/opt/KUIN00601/ca.crt" --cert="/opt/KUIN00601/etcd-client.crt" --key="/opt/KUIN00601/etcd-client.key" snapshot restore /data/backup/etcd-snapshot-previous.db

```
candidate@node01:~$ sudo etcdctl --endpoints=https://11.0.1.111:2379 snapshot restore /data/backup/etcd-snapshot-previous.db
Deprecated: Use `etcdutl snapshot restore` instead.

2023-02-14T21:25:47+08:00       info    snapshot/v3_snapshot.go:248     restoring snapshot      {"path": "/data/backup/etcd-snapshot-prev
ious.db", "wal-dir": "default.etcd/member/wal", "data-dir": "default.etcd", "snap-dir": "default.etcd/member/snap", "stack": "go.etcd.io/
etcd/etcdutl/v3/snapshot.(*v3Manager).Restore\n\tgo.etcd.io/etcd/etcdutl/v3@v3.5.6/snapshot/v3_snapshot.go:254\ngo.etcd.io/etcd/etcdutl/v
3/etcdutl.SnapshotRestoreCommandFunc\n\tgo.etcd.io/etcd/etcdutl/v3@v3.5.6/etcdutl/snapshot_command.go:147\ngo.etcd.io/etcd/etcdctl/v3/ctl
v3/command.snapshotRestoreCommandFunc\n\tgo.etcd.io/etcd/etcdctl/v3/ctlv3/command/snapshot_command.go:129\ngithub.com/spf13/cobra.(*Comma
nd).execute\n\tgithub.com/spf13/cobra@v1.1.3/command.go:856\ngithub.com/spf13/cobra.(*Command).ExecuteC\n\tgithub.com/spf13/cobra@v1.1.3/
command.go:960\ngithub.com/spf13/cobra.(*Command).Execute\n\tgithub.com/spf13/cobra@v1.1.3/command.go:897\ngo.etcd.io/etcd/etcdctl/v3/ctl
v3.Start\n\tgo.etcd.io/etcd/etcdctl/v3/ctlv3/ctl.go:107\ngo.etcd.io/etcd/etcdctl/v3/ctlv3.MustStart\n\tgo.etcd.io/etcd/etcdctl/v3/ctlv3/c
tl.go:111\nmain.main\n\tgo.etcd.io/etcd/etcdctl/v3/main.go:59\nruntime.main\n\truntime/proc.go:225"}
2023-02-14T21:25:47+08:00       info    membership/store.go:141 Trimming membership information from the backend...
2023-02-14T21:25:47+08:00       info    membership/cluster.go:421       added member    {"cluster-id": "cdf818194e3a8c32", "local-member-
id": "0", "added-peer-id": "8e9e05c52164694d", "added-peer-peer-urls": ["http://localhost:2380"]}
2023-02-14T21:25:47+08:00       info    snapshot/v3_snapshot.go:269     restored snapshot       {"path": "/data/backup/etcd-snapshot-prev
ious.db", "wal-dir": "default.etcd/member/wal", "data-dir": "default.etcd", "snap-dir": "default.etcd/member/snap"}
candidate@node01:~$
```

```
student@node01:~$ sudo ETCDCTL_API=3 etcdctl --endpoints=https://11.0.1.111:2379 --cacert="/opt/KUIN00601/ca.crt" --cert="/opt/KUIN0
0601/etcd-client.crt" --key="/opt/KUIN00601/etcd-client.key" snapshot restore /data/backup/etcd-snapshot-previous.db
Deprecated: Use `etcdutl snapshot restore` instead.

2022-07-01T14:49:28+08:00       info    snapshot/v3_snapshot.go:251     restoring snapshot      {"path": "/data/backup/etcd-snapshot
-previous.db", "wal-dir": "default.etcd/member/wal", "data-dir": "default.etcd", "snap-dir": "default.etcd/member/snap", "stack": "g
o.etcd.io/etcd/etcdutl/v3/snapshot.(*v3Manager).Restore\n\t/tmp/etcd-release-3.5.1/etcd/release/etcd/etcdutl/snapshot/v3_snapshot.go
:257\ngo.etcd.io/etcd/etcdutl/v3/etcdutl.SnapshotRestoreCommandFunc\n\t/tmp/etcd-release-3.5.1/etcd/release/etcd/etcdutl/etcdutl/sna
pshot_command.go:147\ngo.etcd.io/etcd/etcdctl/v3/ctlv3/command.snapshotRestoreCommandFunc\n\t/tmp/etcd-release-3.5.1/etcd/release/et
cd/etcdctl/ctlv3/command/snapshot_command.go:128\ngithub.com/spf13/cobra.(*Command).execute\n\t/home/remote/sbatsche/.gvm/pkgsets/go
1.16.3/global/pkg/mod/github.com/spf13/cobra@v1.1.3/command.go:856\ngithub.com/spf13/cobra.(*Command).ExecuteC\n\t/home/remote/sbats
che/.gvm/pkgsets/go1.16.3/global/pkg/mod/github.com/spf13/cobra@v1.1.3/command.go:960\ngithub.com/spf13/cobra.(*Command).Execute\n\t
/home/remote/sbatsche/.gvm/pkgsets/go1.16.3/global/pkg/mod/github.com/spf13/cobra@v1.1.3/command.go:897\ngo.etcd.io/etcd/etcdctl/v3/
ctlv3.Start\n\t/tmp/etcd-release-3.5.1/etcd/release/etcd/etcdctl/ctlv3/ctl.go:107\ngo.etcd.io/etcd/etcdctl/v3/ctlv3.MustStart\n\t/tm
p/etcd-release-3.5.1/etcd/release/etcd/etcdctl/ctlv3/ctl.go:111\nmain.main\n\t/tmp/etcd-release-3.5.1/etcd/release/etcd/etcdctl/main
.go:59\nruntime.main\n\t/home/remote/sbatsche/.gvm/gos/go1.16.3/src/runtime/proc.go:225"}
2022-07-01T14:49:28+08:00       info    membership/store.go:141 Trimming membership information from the backend...
2022-07-01T14:49:28+08:00       info    membership/cluster.go:421       added member    {"cluster-id": "cdf818194e3a8c32", "local-me
mber-id": "0", "added-peer-id": "8e9e05c52164694d", "added-peer-peer-urls": ["http://localhost:2380"]}
2022-07-01T14:49:28+08:00       info    snapshot/v3_snapshot.go:272     restored snapshot       {"path": "/data/backup/etcd-snapshot
-previous.db", "wal-dir": "default.etcd/member/wal", "data-dir": "default.etcd", "snap-dir": "default.etcd/member/snap"}
student@node01:~$
```

PS：扩展阅读
考试时，etcd 是在 node01 上运行的，所以考试时，考题里的网址写的是 https://127.0.0.1:2379。但是模拟环境里，虽然也模拟在 node01 上面执行数据恢复，但 etcd 实际是在 master01 节点，所以还原实际没有真正效果的。
真正恢复的时候 是必须要先将 etcd data 目录清理掉或者移走，然后再去执行 snapshot restore，这时集群的数据就一定和 snapshot 备份中的数据统一了。
比如方法 3。
感谢网友善先生

方法 2：（了解即可）

（建议了解 K8S 原理的这么做，但前提是 etcd 使用的是 systemd 服务，而不是 etcd pod。）
（真实考试时，使用的是 systemd 服务的。但模拟环境使用的是 etcd pod。无法在模拟环境模拟练习这道题的。）
方法 2 有一个前提，就是 etcd 使用的是 systemd 服务，而不是 etcd 容器。
1.22 和 1.23 的考试里，有人反馈 etcd 是服务，有人反馈 etcd 是 pod。我怀疑 CKA 可能有多套考试环境，所以你做的时候，需要确认是否为 etcd 服务。
这个模拟环境，使用的是 etcd pod 的方式搭建的，所以不能用此方法测试。

方法 2 和方法 3 类似，或者说原理是相同的。而且，是可以实现真正 etcd 快照还原的步骤：
首先说一下，这种做法稍后有风险的，一旦操作错误，集群出问题，很可能影响同集群的其他考题，所以不建议冒险。

注意集群用的是考试时的上一题的集群，所以无需再切换集群了。
但如果事后回来做这道题的话，切记要切换为正确的集群。
kubectl config use-context xxxx

开始操作

确认一下 ssh 终端，是在[candidate@node-1] $ 下

备份：
#如果不使用 export ETCDCTL_API=3，而使用 ETCDCTL_API=3，则下面每条 etcdctl 命令前都要加 ETCDCTL_API=3。
export ETCDCTL_API=3
etcdctl --endpoints=https://11.0.1.111:2379 --cacert="/opt/KUIN00601/ca.crt" --cert="/opt/KUIN00601/etcd-client.crt" --key="/opt/KUIN00601/etcd-client.key" snapshot save /var/lib/backup/etcd-snapshot.db

检查：（考试时，这些检查动作，都可以不做）
etcdctl snapshot status /var/lib/backup/etcd-snapshot.db -wtable

还原：
考试时，/data/backup/etcd-snapshot-previous.db 的权限应该是只有 root 可读，所以需要使用 sudo 命令。
可以 ll /data/backup/etcd-snapshot-previous.db 检查一下读写权限和属主。

1、先检查一下考试环境，使用的 etcd 是服务还是容器。
kubectl get pod -A
# 注意加 sudo，因为你用的 candidate 帐号，而非 root 帐号。
sudo systemctl status etcd
如果是 systemd 服务，则继续往下操作。（模拟环境使用的 etcd 不是服务，而是方法 3 的 pod。）

2、确认 etcd 数据目录（--data-dir 值）

ps -ef |grep etcd
一般默认为/var/lib/etcd

3、停止 etcd 服务
sudo systemctl stop etcd

4、先移动备份 etcd 原目录
sudo mv /var/lib/etcd /var/lib/etcd.bak

5、开始还原（还原时，可以不加证书和秘钥）
sudo ETCDCTL_API=3 etcdctl --data-dir=/var/lib/etcd snapshot restore /data/backup/etcd-snapshot-previous.db

6、更改文件属主
sudo chown -R etcd:etcd /var/lib/etcd

7、启动 etcd 服务
sudo systemctl start etcd

方法 3：（了解即可）

方法 3 和方法 2 类似，或者说原理是相同的。而且，是可以实现真正 etcd 快照还原的步骤：
首先说一下，这种做法稍后有风险的，一旦操作错误，集群出问题，很可能影响同集群的其他考题，所以不建议冒险。

注意集群用的是考试时的上一题的集群，所以无需再切换集群了。
但如果事后回来做这道题的话，切记要切换为正确的集群。
kubectl config use-context xxxx

开始操作

确认一下 ssh 终端，是在[candidate@node-1] $ 下

备份：
#如果不使用 export ETCDCTL_API=3，而使用 ETCDCTL_API=3，则下面每条 etcdctl 命令前都要加 ETCDCTL_API=3。
export ETCDCTL_API=3
etcdctl --endpoints=https://11.0.1.111:2379 --cacert="/opt/KUIN00601/ca.crt" --cert="/opt/KUIN00601/etcd-client.crt" --key="/opt/KUIN00601/etcd-client.key" snapshot save /var/lib/backup/etcd-snapshot.db

检查：（考试时，这些检查动作，都可以不做）
etcdctl snapshot status /var/lib/backup/etcd-snapshot.db -wtable

还原：
考试时，/data/backup/etcd-snapshot-previous.db 的权限应该是只有 root 可读，所以需要使用 sudo 命令。
可以 ll /data/backup/etcd-snapshot-previous.db 检查一下读写权限和属主。

测试环境的 etcd 在 master01 节点，所以需要切换到 master 节点。
但考试时，从考试给出的题目“为运行在 https://11.0.1.111:2379 上的现有 etcd 实例创建快照”可知，etcd 是运行在本节点 node-1 上的。
所以考试时，不需要切到 master。

```
student@node01:~$ kubectl get pod -A -o wide|grep -i etcd
kube-system     etcd-master01                      1/1     Running   9 (16h ago)     2d3h   11.0.1.111    master01   <none>           <none>
student@node01:~$
```

ssh master01 #仅测试环境下执行切到 master

```
student@node01:~$ ssh master01
```

移除且备份 /etc/kubernetes/manifests 目录
sudo mv /etc/kubernetes/manifests /etc/kubernetes/manifests.bak
查看 kube-apiserver、etcd 镜像是否停止
crictl ps|grep etcd && crictl ps|grep kube-apiserver
备份现有 Etcd 数据
sudo mv /var/lib/etcd /var/lib/etcd.bak
##开始还原（还原时，可以不加证书和秘钥）
sudo 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 /data/backup/etcd-snapshot-previous.db --data-dir=/var/lib/etcd/
（之前备份的 HASH 为 68a972d6）
恢复 Kube-Apiserver 与 Etcd 镜像
sudo mv /etc/kubernetes/manifests.bak /etc/kubernetes/manifests
退回 node01
exit #仅测试环境下执行，因为考试时，你没有切到 master 下，而是一直在 node-1 下的。

```
student@master01:~$ sudo mv /etc/kubernetes/manifests /etc/kubernetes/manifests.bak
student@master01:~$ sudo mv /etc/kubernetes/manifests /etc/kubernetes/manifests.bak
student@master01:~$ docker ps|grep etcd && docker ps|grep kube-apiserver
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1
.24/containers/json: dial unix /var/run/docker.sock: connect: permission denied
student@master01:~$ sudo mv /var/lib/etcd /var/lib/etcd.bak
student@master01:~$ sudo ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert="/opt/KUIN00601/ca.crt" --cert="/opt/KUIN00601/etcd-cli
ent.crt" --key="/opt/KUIN00601/etcd-client.key" snapshot restore /data/backup/etcd-snapshot-previous.db --data-dir=/var/lib/etcd/
Deprecated: Use `etcdutl snapshot restore` instead.

2022-02-24T18:20:22+08:00       info    snapshot/v3_snapshot.go:251     restoring snapshot      {"path": "/data/backup/etcd-snapshot-previous.db",
 "wal-dir": "/var/lib/etcd/member/wal", "data-dir": "/var/lib/etcd/", "snap-dir": "/var/lib/etcd/member/snap", "stack": "go.etcd.io/etcd/etcdutl/v
3/snapshot.(*v3Manager).Restore\n\t/tmp/etcd-release-3.5.1/etcd/release/etcd/etcdutl/snapshot/v3_snapshot.go:257\ngo.etcd.io/etcd/etcdutl/v3/etcdu
tl.SnapshotRestoreCommandFunc\n\t/tmp/etcd-release-3.5.1/etcd/release/etcd/etcdutl/etcdutl/snapshot_command.go:147\ngo.etcd.io/etcd/etcdctl/v3/ctl
v3/command.snapshotRestoreCommandFunc\n\t/tmp/etcd-release-3.5.1/etcd/release/etcd/etcdctl/ctlv3/command/snapshot_command.go:128\ngithub.com/spf13
/cobra.(*Command).execute\n\t/home/remote/sbatsche/.gvm/pkgsets/go1.16.3/global/pkg/mod/github.com/spf13/cobra@v1.1.3/command.go:856\ngithub.com/s
pf13/cobra.(*Command).ExecuteC\n\t/home/remote/sbatsche/.gvm/pkgsets/go1.16.3/global/pkg/mod/github.com/spf13/cobra@v1.1.3/command.go:960\ngithub.
com/spf13/cobra.(*Command).Execute\n\t/home/remote/sbatsche/.gvm/pkgsets/go1.16.3/global/pkg/mod/github.com/spf13/cobra@v1.1.3/command.go:897\ngo.
etcd.io/etcd/etcdctl/v3/ctlv3.Start\n\t/tmp/etcd-release-3.5.1/etcd/release/etcd/etcdctl/ctlv3/ctl.go:107\ngo.etcd.io/etcd/etcdctl/v3/ctlv3.MustSt
art\n\t/tmp/etcd-release-3.5.1/etcd/release/etcd/etcdctl/ctlv3/ctl.go:111\nmain.main\n\t/tmp/etcd-release-3.5.1/etcd/release/etcd/etcdctl/main.go:
59\nruntime.main\n\t/home/remote/sbatsche/.gvm/gos/go1.16.3/src/runtime/proc.go:225"}
2022-02-24T18:20:22+08:00       info    membership/store.go:141 Trimming membership information from the backend...
2022-02-24T18:20:22+08:00       info    membership/cluster.go:421       added member    {"cluster-id": "cdf818194e3a8c32", "local-member-id": "0",
 "added-peer-id": "8e9e05c52164694d", "added-peer-peer-urls": ["http://localhost:2380"]}
2022-02-24T18:20:22+08:00       info    snapshot/v3_snapshot.go:272     restored snapshot       {"path": "/data/backup/etcd-snapshot-previous.db",
 "wal-dir": "/var/lib/etcd/member/wal", "data-dir": "/var/lib/etcd/", "snap-dir": "/var/lib/etcd/member/snap"}
student@master01:~$
student@master01:~$ sudo mv /etc/kubernetes/manifests.bak /etc/kubernetes/manifests
student@master01:~$ exit
logout
Connection to master01 closed.
student@node01:~$
```

稍等 3 分钟，再检查 pod。
因为太快检查，可能会提示“The connection to the server 11.0.1.111:6443 was refused - did you specify the right host or port?”
成功还原到了一个比较早的 etcd 快照，之前的很多 pod 都没有了。比如查看 pud 的 CPU 的那道题的 pod，redis-test 和 nginx-host 等都没有了。
kubectl get pod -A

```
student@node01:~$ kubectl get pod -A
NAMESPACE       NAME                                READY   STATUS    RESTARTS       AGE
ing-internal    hello-6db876d977-pqbls              1/1     Running   0              18h
ingress-nginx   ingress-nginx-controller-58cbl      1/1     Running   0              18h
ingress-nginx   ingress-nginx-controller-fss75      1/1     Running   0              18h
kube-system     coredns-6d8c4cb4d-p68v8             1/1     Running   10 (19h ago)   2d3h
kube-system     coredns-6d8c4cb4d-t5n94             1/1     Running   10 (19h ago)   2d3h
kube-system     etcd-master01                       1/1     Running   0              2d3h
kube-system     kube-apiserver-master01             1/1     Running   0              2d3h
kube-system     kube-controller-manager-master01    1/1     Running   0              2d3h
kube-system     kube-flannel-ds-p7v8r               1/1     Running   8 (19h ago)    2d3h
kube-system     kube-flannel-ds-qlcbv               1/1     Running   8 (19h ago)    2d3h
kube-system     kube-flannel-ds-vmd6h               1/1     Running   8 (19h ago)    2d3h
kube-system     kube-proxy-wgj54                    1/1     Running   8 (19h ago)    2d3h
kube-system     kube-proxy-wjgd7                    1/1     Running   8 (19h ago)    2d3h
kube-system     kube-proxy-zbdn9                    1/1     Running   8 (19h ago)    2d3h
kube-system     kube-scheduler-master01             1/1     Running   0              2d3h
student@node01:~$
```

16、排查集群中故障节点

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context wk8s

Task
名为 node02 的 Kubernetes worker node 处于 NotReady 状态。
调查发生这种情况的原因，并采取相应的措施将 node 恢复为 Ready 状态，确保所做的任何更改永久生效。

可以使用以下命令，通过 ssh 连接到 node02 节点：

ssh node02
可以使用以下命令，在该节点上获取更高权限：
sudo -i

参考链接

没必要参考网址，记住先 restart 再 enable 就行。

https://kubernetes.io/zh-cn/docs/setup/production-environment/tools/kubeadm/kubelet-integration/

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context wk8s

通过 get nodes 查看异常节点，登录节点查看 kubelet 等组件的 status 并判断原因。
真实考试时，这个异常节点的 kubelet 服务没有启动导致的，就这么简单。

执行初始化这道题的脚本 a.sh，模拟 node02 异常。
（考试时，不需要执行的！考试时切到这道题的集群后，那个 node 就是异常的。）

在 candidate@node01 上执行
sudo sh /opt/sh/a.sh

执行完 a.sh 脚本后，等 3 分钟，node02 才会挂掉。

```
candidate@node01:~$ sudo sh /opt/sh/a.sh
Removed /etc/systemd/system/multi-user.target.wants/kubelet.service.
```

开始操作

检查一下 node，发现 node02 异常
kubectl get nodes

```
candidate@node01:~$ kubectl get nodes
NAME       STATUS     ROLES           AGE   VERSION
master01   Ready      control-plane   10d   v1.26.1
node01     Ready      <none>          10d   v1.26.0
node02     NotReady   <none>          10d   v1.26.0
candidate@node01:~$
```

ssh 到 node02 节点，并切换到 root 下
ssh node02
sudo -i

```
candidate@node01:~$ ssh node02
```

```
candidate@node02:~$ sudo -i
root@node02:~#
```

检查 kubelet 服务，考试时，会发现服务没有启动
systemctl status kubelet

```
root@node02:~# systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/lib/systemd/system/kubelet.service; disabled; vendor preset: enabled)
    Drop-In: /etc/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: inactive (dead) since Fri 2022-07-01 14:56:09 CST; 3min 1s ago
       Docs: https://kubernetes.io/docs/home/
   Main PID: 873 (code=exited, status=0/SUCCESS)

Jul 01 14:52:58 node02 kubelet[873]: E0701 14:52:58.253654     873 remote_runtime.go:479] "StopContainer from runtime service failed" >
Jul 01 14:52:58 node02 kubelet[873]: E0701 14:52:58.253671     873 kuberuntime_container.go:719] "Container termination failed with gr>
Jul 01 14:52:58 node02 kubelet[873]: E0701 14:52:58.253681     873 kuberuntime_container.go:744] "Kill container failed" err="rpc erro>
Jul 01 14:52:58 node02 kubelet[873]: E0701 14:52:58.254818     873 kubelet.go:1777] [failed to "KillContainer" for "sidecar" with Kill>
Jul 01 14:52:58 node02 kubelet[873]: E0701 14:52:58.254907     873 pod_workers.go:918] "Error syncing pod, skipping" err="[failed to \>
Jul 01 14:52:58 node02 kubelet[873]: I0701 14:52:58.257172     873 kubelet_volumes.go:160] "Cleaned up orphaned pod volumes dir" podUI>
Jul 01 14:56:09 node02 systemd[1]: Stopping kubelet: The Kubernetes Node Agent...
Jul 01 14:56:09 node02 kubelet[873]: I0701 14:56:09.010896     873 dynamic_cafile_content.go:170] "Shutting down controller" name="cli>
Jul 01 14:56:09 node02 systemd[1]: kubelet.service: Succeeded.
Jul 01 14:56:09 node02 systemd[1]: Stopped kubelet: The Kubernetes Node Agent.
lines 1-18/18 (END)
```

启动服务，并设置为开机启动
systemctl start kubelet
systemctl enable kubelet

```
root@node02:~# systemctl start kubelet
root@node02:~# systemctl enable kubelet
Created symlink /etc/systemd/system/multi-user.target.wants/kubelet.service → /lib/systemd/system/kubelet.service.
root@node02:~#
```

检查
systemctl status kubelet

```
root@node02:~# systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
    Drop-In: /etc/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Fri 2022-07-01 14:59:43 CST; 16s ago
       Docs: https://kubernetes.io/docs/home/
   Main PID: 34060 (kubelet)
      Tasks: 14 (limit: 2531)
     Memory: 31.6M
     CGroup: /system.slice/kubelet.service
             └─34060 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubel>

Jul 01 14:59:45 node02 kubelet[34060]: I0701 14:59:45.024923   34060 reconciler.go:216] "operationExecutor.VerifyControllerAttachedVol>
Jul 01 14:59:45 node02 kubelet[34060]: I0701 14:59:45.024960   34060 reconciler.go:216] "operationExecutor.VerifyControllerAttachedVol>
Jul 01 14:59:45 node02 kubelet[34060]: I0701 14:59:45.024974   34060 reconciler.go:216] "operationExecutor.VerifyControllerAttachedVol>
Jul 01 14:59:45 node02 kubelet[34060]: I0701 14:59:45.024989   34060 reconciler.go:216] "operationExecutor.VerifyControllerAttachedVol>
Jul 01 14:59:45 node02 kubelet[34060]: I0701 14:59:45.025006   34060 reconciler.go:216] "operationExecutor.VerifyControllerAttachedVol>
Jul 01 14:59:45 node02 kubelet[34060]: I0701 14:59:45.025020   34060 reconciler.go:216] "operationExecutor.VerifyControllerAttachedVol>
Jul 01 14:59:45 node02 kubelet[34060]: I0701 14:59:45.025034   34060 reconciler.go:216] "operationExecutor.VerifyControllerAttachedVol>
Jul 01 14:59:45 node02 kubelet[34060]: I0701 14:59:45.025049   34060 reconciler.go:216] "operationExecutor.VerifyControllerAttachedVol>
Jul 01 14:59:45 node02 kubelet[34060]: I0701 14:59:45.025055   34060 reconciler.go:157] "Reconciler: start to sync state"
Jul 01 14:59:53 node02 kubelet[34060]: I0701 14:59:53.732169   34060 prober_manager.go:255] "Failed to trigger a manual run" probe="Re>
lines 1-22/22 (END)
```

退出 root，退回到 candidate@node02
exit

退出 node02，退回到 candidate@node01
exit

```
root@node02:~# exit
logout
candidate@node02:~$ exit
logout
Connection to node02 closed.
candidate@node01:~$
```

再次检查节点， 确保 node02 节点恢复 Ready 状态
kubectl get nodes

```
candidate@node01:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   10d   v1.26.1
node01     Ready    <none>          10d   v1.26.0
node02     Ready    <none>          10d   v1.26.0
candidate@node01:~$
```

17、节点维护

考题

设置配置环境：
[candidate@node-1] $ kubectl config use-context ek8s

Task
将名为 node02 的 node 设置为不可用，并重新调度该 node 上所有运行的 pods。

考点：cordon 和 drain 命令的使用

参考链接

没必要参考网址，使用-h 帮助更方便。
kubectl -h

https://kubernetes.io/zh-cn/docs/tasks/administer-cluster/safely-drain-node/

解答

考试时务必执行，切换集群。模拟环境中不需要执行。
kubectl config use-context ek8s

开始操作

kubectl get node

```
candidate@node01:~$ kubectl get node
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   10d   v1.26.1
node01     Ready    <none>          10d   v1.26.0
node02     Ready    <none>          10d   v1.26.0
candidate@node01:~$
```

kubectl cordon node02
kubectl get node

```
candidate@node01:~$ kubectl cordon node02
node/node02 cordoned
candidate@node01:~$ kubectl get node
NAME       STATUS                     ROLES           AGE   VERSION
master01   Ready                      control-plane   10d   v1.26.1
node01     Ready                      <none>          10d   v1.26.0
node02     Ready,SchedulingDisabled   <none>          10d   v1.26.0
candidate@node01:~$
```

kubectl drain node02 --ignore-daemonsets
注意，还有一个参数--delete-emptydir-data --force，这个考试时不用加，就可以正常 draini node02 的。
但如果执行后，有跟测试环境一样的报错（如下截图），则需要加上--delete-emptydir-data --force，会强制将 pod 移除。
kubectl drain node02 --ignore-daemonsets --delete-emptydir-data --force

```
candidate@node01:~$ kubectl drain node02 --ignore-daemonsets
node/node02 already cordoned
error: unable to drain node "node02" due to error:[cannot delete Pods with local storage (use --delete-emptydir-data to override): defaul
t/11-factor-app, cannot delete Pods declare no controller (use --force to override): default/kucc8], continuing command...
There are pending nodes to be drained:
 node02
cannot delete Pods with local storage (use --delete-emptydir-data to override): default/11-factor-app
cannot delete Pods declare no controller (use --force to override): default/kucc8
candidate@node01:~$
candidate@node01:~$ kubectl drain node02 --ignore-daemonsets --delete-emptydir-data --force
node/node02 already cordoned
Warning: ignoring DaemonSet-managed Pods: calico-system/calico-node-4kck4, calico-system/csi-node-driver-k9kcl, ingress-nginx/ingress-ngi
nx-controller-9g98t, kube-system/kube-proxy-7jb2h; deleting Pods that declare no controller: default/11-factor-app, default/kucc8
evicting pod kube-system/coredns-5bbd96d687-gsmfz
evicting pod calico-apiserver/calico-apiserver-5b694c87c8-gtbgc
evicting pod cpu-top/redis-test-ff9fd7d78-dsbf2
evicting pod cpu-top/test0-75d9886fc7-44b28
evicting pod default/11-factor-app
evicting pod default/front-end-5d546b68bb-pr7pj
evicting pod default/kucc8
evicting pod default/presentation-d9dbc88cb-jb5p9
evicting pod default/presentation-d9dbc88cb-qzvpd
evicting pod ing-internal/hello-8544bccfcb-m2v49
pod/front-end-5d546b68bb-pr7pj evicted
I0214 21:31:04.879492   43044 request.go:682] Waited for 1.109296166s due to client-side throttling, not priority and fairness, request:
GET:https://11.0.1.111:6443/api/v1/namespaces/default/pods/presentation-d9dbc88cb-jb5p9
pod/hello-8544bccfcb-m2v49 evicted
pod/redis-test-ff9fd7d78-dsbf2 evicted
pod/presentation-d9dbc88cb-jb5p9 evicted
pod/test0-75d9886fc7-44b28 evicted
pod/calico-apiserver-5b694c87c8-gtbgc evicted
pod/presentation-d9dbc88cb-qzvpd evicted
pod/kucc8 evicted
pod/coredns-5bbd96d687-gsmfz evicted
pod/11-factor-app evicted
node/node02 drained
candidate@node01:~$
```

检查

kubectl get node

kubectl get pod -A -o wide|grep node02

正常已经没有 pod 在 node02 上了。但测试环境里的 ingress、calico、kube-proxy 是 daemonsets 模式的，所以显示还在 node02 上，忽略即可。

```
candidate@node01:~$ kubectl get node
NAME       STATUS                     ROLES           AGE   VERSION
master01   Ready                      control-plane   10d   v1.26.1
node01     Ready                      <none>          10d   v1.26.0
node02     Ready,SchedulingDisabled   <none>          10d   v1.26.0
candidate@node01:~$ kubectl get pod -A -o wide|grep node02
calico-system      calico-node-4kck4                          1/1     Running   7 (90m ago)    10d     11.0.1.113       node02     <none>
calico-system      csi-node-driver-k9kcl                      2/2     Running   14 (90m ago)   10d     10.244.140.100   node02     <none>
ingress-nginx      ingress-nginx-controller-9g98t             1/1     Running   4 (90m ago)    9d      11.0.1.113       node02     <none>
kube-system        kube-proxy-7jb2h                           1/1     Running   0              27m     11.0.1.113       node02     <none>
candidate@node01:~$
```

我的微信是 shadowooom 有在考试模拟环境里做题问题，考试流程问题，都可以问我的。

更新内容

2023 年 2 月 27 号

1、优化答案备注

2023 年 2 月 13 号

1、更新 K8S v1.26 版本题目答案和备注

2022 年 11 月 5 号

1、更新 K8S v1.25 版本题目答案和备注

2022 年 9 月 7 号

1、更新 ingress 题目答案和备注

2022 年 8 月 10 号

1、补充暴露服务和 ingress 两道题的答案

2022 年 7 月 19 号

1、更新资料为 v1.24
2、更新官网参考地址

2022 年 7 月 1 号

1、根据新考试平台，更新答案和参考网页

2022 年 6 月 5 号

1、优化答案

2022 年 5 月 26 号

1、修改第 5 题 ingress 答案

2022 年 5 月 5 号

1、优化答案

2022 年 4 月 12 号

1、更改调度 pod 那道题的参考链接地址

2022 年 4 月 3 号

1、优化内容

2022 年 3 月 24 号

1、修改参考链接
2、添加参考链接截图

2022 年 3 月 20 号

修改 pvc 做题方法

2022 年 3 月 13 号

增加 ingress 做题方法

2022 年 3 月 6 号

更新“5、创建 Ingress”这道题，优先推荐加 kubernetes.io/ingress.class: "nginx"

2022 年 3 月 2 号

1、优化答案内容

2022 年 2 月 24 号

1、2022 年 2 月 21 号，考试中心已更题库 CKA v1.23，模拟环境和答案也更新为 CKA 1.23
2、优化题目标准答案。

2022 年 2 月 16 号

1、优化答案内容

2022 年 1 月 10 号

1、最近一个同学在 v1.22 版本中，考了 98 分，根据他的反馈，更新了一些题目答案。
主要为第 4 题“创建 service”，第 13 题“使用 sidecar 代理容器日志”，第 16 题“备份还原 etcd”

2022 年 1 月 7 号

1、根据近几次考试结果反馈，CKA 是有多套考试环境的，差别不大。主要的差别是在“创建 Ingress”那道题，有的考试环境需要加 kubernetes.io/ingress.class: "nginx"，
有的考试环境则不需要加 kubernetes.io/ingress.class: "nginx"。
这个注意事项，在模拟题里已经用红色字体备注了。