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 ）

真正考试时，是 candidate 或 cli 普通账号下，在 node-1 机器上做题的。并不是在 root 下，也不是在 master 机器上，要注意。
所以模拟环境也是用 candidate 账号，在 node01 账号下做题的。

注意，官方的考试环境有多套，考试时，不一定抽到哪套。不同考试环境部分题目的内容有略微变化，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，甚至有少部分同学没有做完题。
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(PSI)，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(PSI)，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 K8S 官网自己查。
另外每道考试题目里，也都会给你一个参考链接，但是这些不是太好，没有参考价值，不建议使用。还是推荐用考试软件 PSI 里的火狐浏览器到官网自己查。
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，再 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)
在新的 PSI 考试环境里，默认 kubectl 命令，已经可以自动补全了。模拟环境的 kubectl 命令也是可以自动补全的。


```
candidate@node01:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE    VERSION
master01   Ready    control-plane   3h45m  v1.29.0
node01     Ready    <none>          3h40m  v1.29.0
node02     Ready    <none>          3h40m  v1.29.0
candidate@node01:~$
candidate@node01:~$ kubectl get pod -A
NAMESPACE          NAME                                       READY   STATUS    RESTARTS      AGE
calico-apiserver   calico-apiserver-556f6d894-7z9xn           1/1     Running   3 (97s ago)   151m
calico-apiserver   calico-apiserver-556f6d894-ncc4w           1/1     Running   3 (96s ago)   151m
calico-system      calico-kube-controllers-687c447f4b-75xfq   1/1     Running   3 (97s ago)   157m
calico-system      calico-node-8xgq7                          1/1     Running   3 (97s ago)   157m
calico-system      calico-node-d8wrr                          1/1     Running   3 (96s ago)   157m
calico-system      calico-node-mzz24                          1/1     Running   3 (97s ago)   157m
calico-system      calico-typha-6d6f5f8cbf-rt545              1/1     Running   3 (97s ago)   157m
calico-system      calico-typha-6d6f5f8cbf-zcjwz              1/1     Running   3 (96s ago)   157m
calico-system      csi-node-driver-29ckz                      2/2     Running   6 (97s ago)   157m
calico-system      csi-node-driver-jwghc                      2/2     Running   6 (97s ago)   157m
calico-system      csi-node-driver-r7xxz                      2/2     Running   6 (96s ago)   149m
cpu-top            nginx-host-846bf49987-mf4mr                1/1     Running   1 (96s ago)   60m
cpu-top            redis-test-799bc675cd-zr7jv                1/1     Running   1 (96s ago)   60m
cpu-top            test0-6c665fdc6c-kbm76                     1/1     Running   1 (96s ago)   60m
default            11-factor-app                              1/1     Running   1 (97s ago)   16m
default            foo                                        1/1     Running   1 (97s ago)   20m
default            front-end-86d6dfc7df-9txjt                 1/1     Running   1 (96s ago)   57m
default            presentation-69bc4b5956-88vmb              1/1     Running   1 (97s ago)   27m
ing-internal       hello-57645c4896-6w46p                     1/1     Running   1 (97s ago)   57m
ingress-nginx      ingress-nginx-controller-ndrqr             1/1     Running   1 (96s ago)   36m
ingress-nginx      ingress-nginx-controller-ql7qz             1/1     Running   1 (97s ago)   36m
kube-system        coredns-857d9ff4c9-7zmvq                   1/1     Running   3 (96s ago)   135m
kube-system        coredns-857d9ff4c9-pk8vm                   1/1     Running   3 (97s ago)   3h44m
kube-system        etcd-master01                              1/1     Running   3 (97s ago)   3h45m
kube-system        kube-apiserver-master01                    1/1     Running   3 (97s ago)   3h45m
kube-system        kube-controller-manager-master01           1/1     Running   3 (97s ago)   3h45m
kube-system        kube-proxy-v9vzq                           1/1     Running   3 (96s ago)   3h40m
kube-system        kube-proxy-vgkdl                           1/1     Running   3 (97s ago)   3h44m
kube-system        kube-proxy-x5qrq                           1/1     Running   3 (97s ago)   3h40m
kube-system        kube-scheduler-master01                    1/1     Running   3 (97s ago)   3h45m
kube-system        metrics-server-5b95796896-59d5v            1/1     Running   1 (97s ago)   58m
tigera-operator    tigera-operator-55585899bf-7f4x5           1/1     Running   4 (97s ago)   159m
candidate@node01:~$
```

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

比较好的命令参考网址
https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands

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：
左边截图是考题，右边截图是模拟环境的题库。

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

鸣谢：

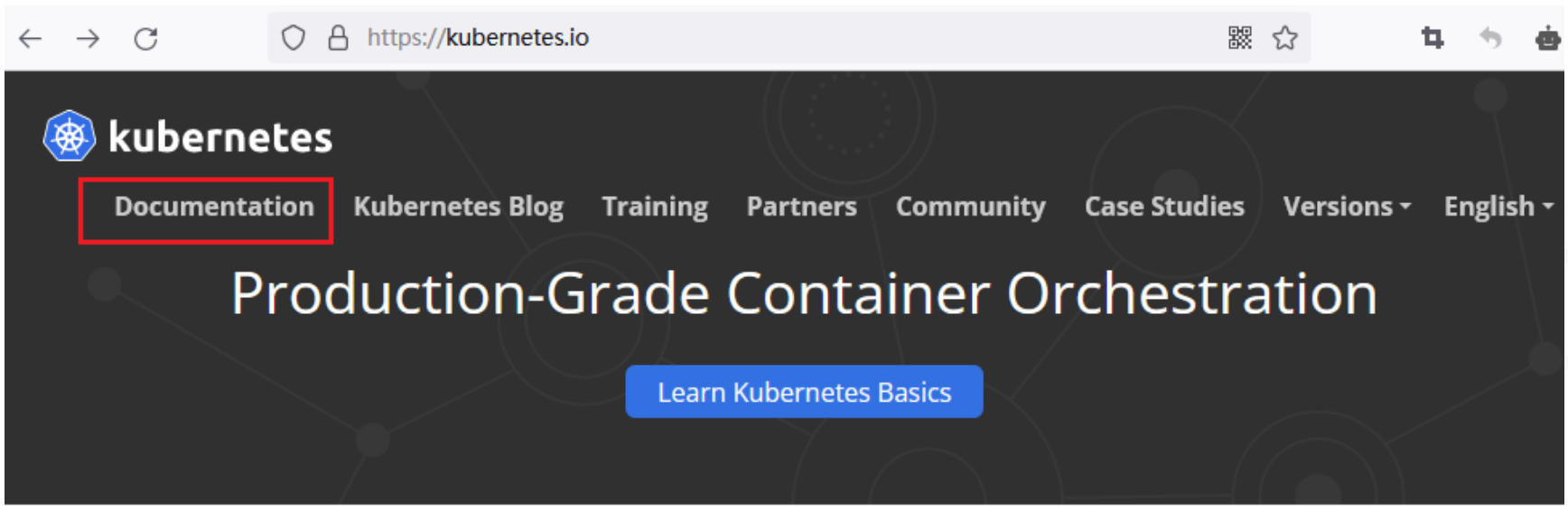
特别鸣谢以下网友（排名不分先后）：

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

0、如何一步步找到找官网资料（这个方法要学会）

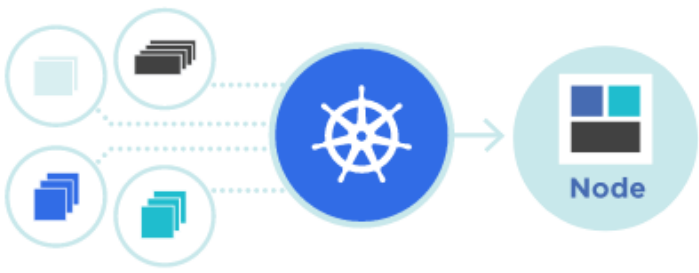
<1> 考试时，打开考试环境 Ubuntu 20.04 桌面上的火狐浏览器，并输入 K8S 官网 https://kubernetes.io/

<2> 点击左上角的“Documentation”

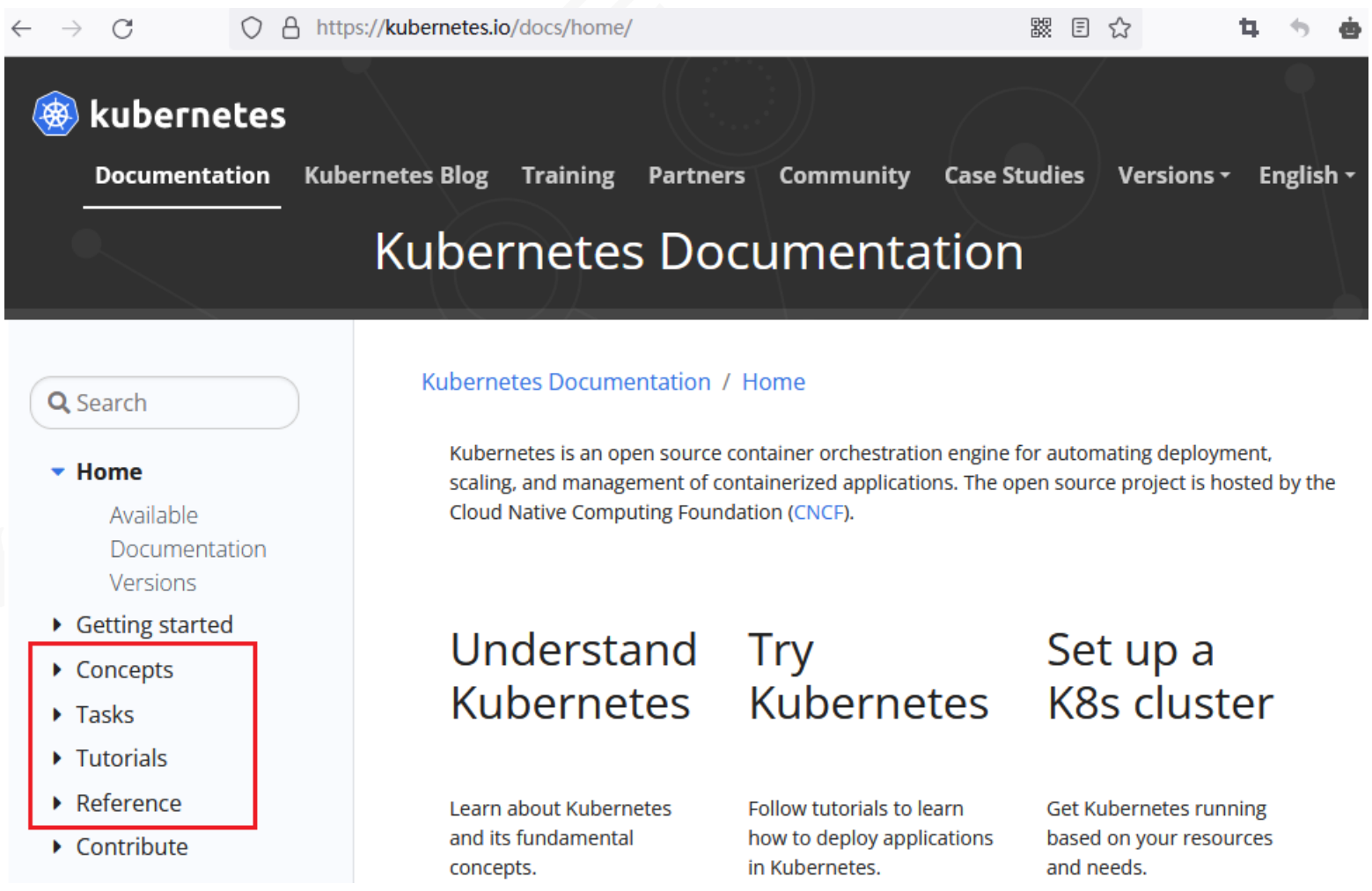


Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

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。

下面的所有题目，需要参考官网的，都是按照这种方式，一步步点出答案来。

1、权限控制 RBAC

考题

（考试的考题内容，只有下面方框里的）

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

强烈建议背过操作命令，如果非要参考，可以使用-h 帮助。
kubectl create clusterrole -h
kubectl create rolebinding -h

解答

先说明一下考试时每道题要切换集群，因为考试时，不同的题，分布在不同集群里，但是都是在 node01 做题。所以要通过切换集群，来做不同的题。

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

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

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

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 ：

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

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

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

kubectl -n app-team1 create serviceaccount cicd-token

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

```
candidate@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
candidate@node01:~$
candidate@node01:~$ kubectl create clusterrole deployment-clusterrole --verb=create --resource=deployments,statefulsets,daemonsets
clusterrole.rbac.authorization.k8s.io/deployment-clusterrole created
candidate@node01:~$ kubectl -n app-team1 create serviceaccount cicd-token
serviceaccount/cicd-token created
candidate@node01:~$
candidate@node01:~$ kubectl -n app-team1 create rolebinding cicd-token-rolebinding --clusterrole=deployment-clusterrole --serviceaccount=app-team1:cicd-token
rolebinding.rbac.authorization.k8s.io/cicd-token-rolebinding created
```

检查（考试时，所有的检查项，都可以不做）
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
显示 no
candidate@node01:~$ kubectl auth can-i create deployment -n app-team1 --as system:serviceaccount:app-team1:cicd-token
显示 yes

```
candidate@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
candidate@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
```

2、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

强烈建议背过操作命令，如果非要参考，可以使用-h 帮助。
kubectl scale deployment -h

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 deployment 的 pod 数量，显示 1/1，表示 1 个 pod。
kubectl get deployments presentation

```
candidate@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
candidate@node01:~$
candidate@node01:~$ kubectl get deployments presentation
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
presentation   1/1     1            1           14h
candidate@node01:~$
```

扩展成 4 个

kubectl scale deployment presentation --replicas=4

```
candidate@node01:~$ kubectl scale deployment presentation --replicas=4
deployment.apps/presentation scaled
candidate@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 快照的错误操作导致的。

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

kubectl get pod -l app=presentation

显示 4/4，表示 4 个 pod。

```
candidate@node01:~$ kubectl get deployments presentation
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
presentation   4/4     4            4           14h
candidate@node01:~$ kubectl get pod -l app=presentation
NAME                            READY   STATUS    RESTARTS      AGE
presentation-69bc4b5956-88vmb   1/1     Running   1 (49m ago)   14h
presentation-69bc4b5956-8lpxd   1/1     Running   0             2m9s
presentation-69bc4b5956-8w7tg   1/1     Running   0             2m9s
presentation-69bc4b5956-p7wtw   1/1     Running   0             2m9s
candidate@node01:~$
```

2.5、yaml 文件格式说明（这个是为第 3 题做准备的）

（懂的就不用看了，解释给小白听的 ^_^）

```yaml
    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。

```yaml
    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

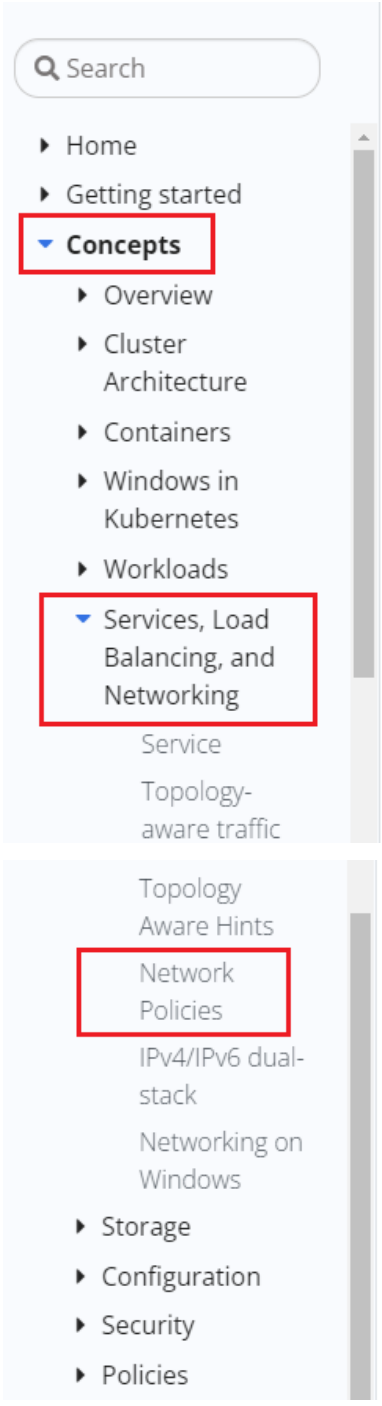
上面解释成白话，双重否定就是肯定，所以最后两句话的意思就是:
仅允许端口为 9000 的 pod 方法。
仅允许 echo 命名空间中的 pod 访问。

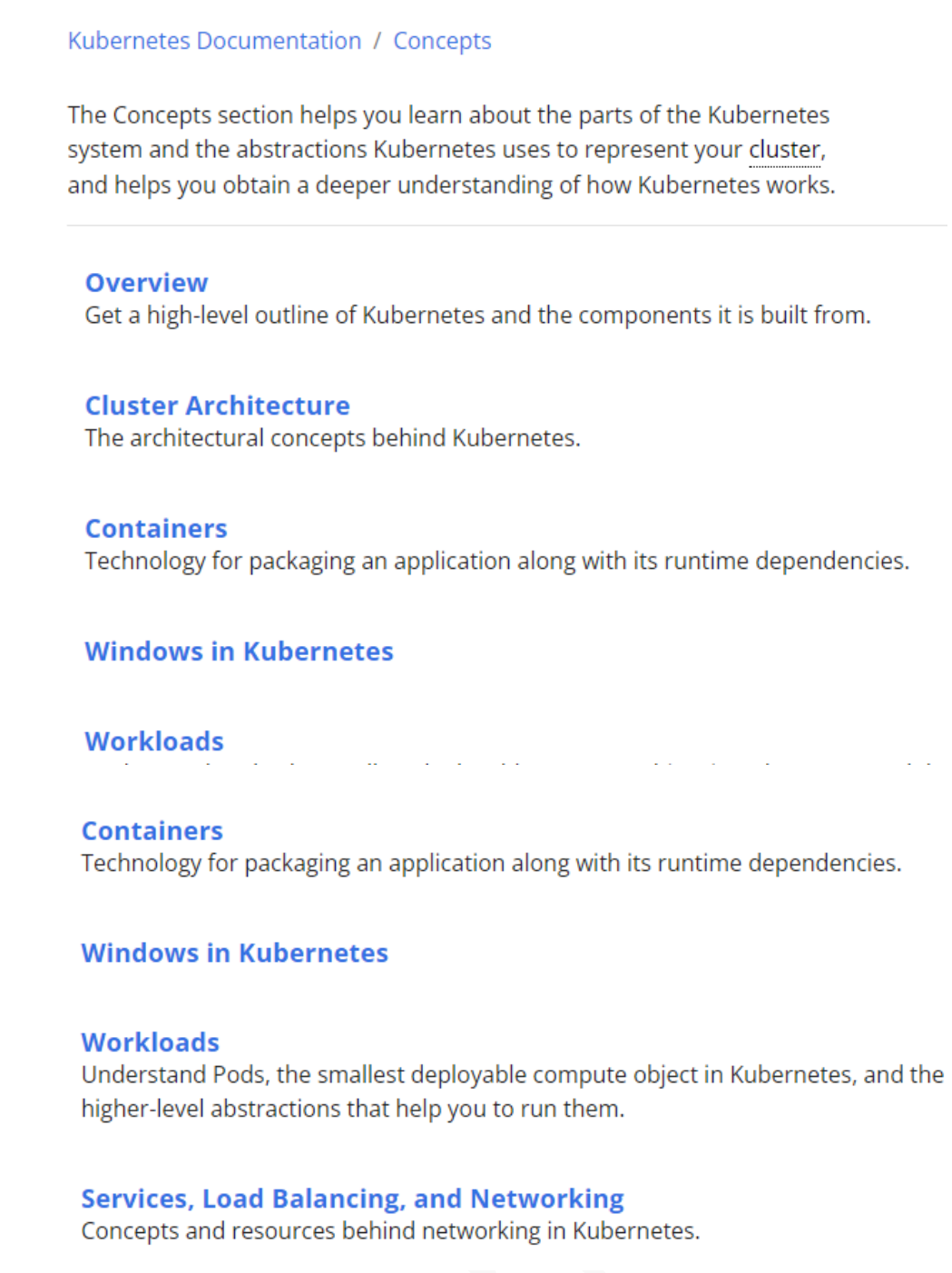
考点：NetworkPolicy 的创建

**参考链接

（需要复制官网网页的内容）

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，考试的考题，可能会故意将 echo 和 my-app 两个调换位置。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，但要分清谁访问谁。
题目“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”，这句话的意思是 echo 访问 my-app。所以 echo 是访问者，my-app 是被访问者。所以下面要创建一个 my-app 的 yaml，里面 ingress 流量控制的 matchLabels 写 project: echo。
翻译成白话，A 访问 B，我们应该在 B 上设置一个防火墙(ingress)，允许 A 来访问。

开始操作

查看所有 ns 的标签 label
kubectl get ns echo --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。（模拟环境其实可以用这个 kubernetes.io/metadata.name=echo 但这个太长了，请按照下面方法打一个）

真实考试环境，是没有标签的，所以需要自己打一个。
kubectl label ns echo project=echo

```
candidate@node01:~$ kubectl config use-context hk8s
error: no context exists with the name: "hk8s"
candidate@node01:~$
candidate@node01:~$ kubectl get ns echo --show-labels
NAME   STATUS   AGE   LABELS
echo   Active   15h   kubernetes.io/metadata.name=echo
candidate@node01:~$
candidate@node01:~$ kubectl label ns echo project=echo
namespace/echo labeled
candidate@node01:~$
candidate@node01:~$ kubectl get ns echo --show-labels
NAME   STATUS   AGE   LABELS
echo   Active   15h   kubernetes.io/metadata.name=echo,project=echo
candidate@node01:~$
```

手动编写一个 yaml 文件，yaml 文件的名字随便起。
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（echo 访问 my-app）
spec:
  podSelector: {}       #这行必须要写
  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（echo 访问 my-app）
    #- 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

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

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

检查

kubectl describe networkpolicy -n my-app

```
candidate@node01:~$ kubectl describe networkpolicy -n my-app
Name:         allow-port-from-namespace
Namespace:    my-app
Created on:   2024-02-16 17:08:34 +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
candidate@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。

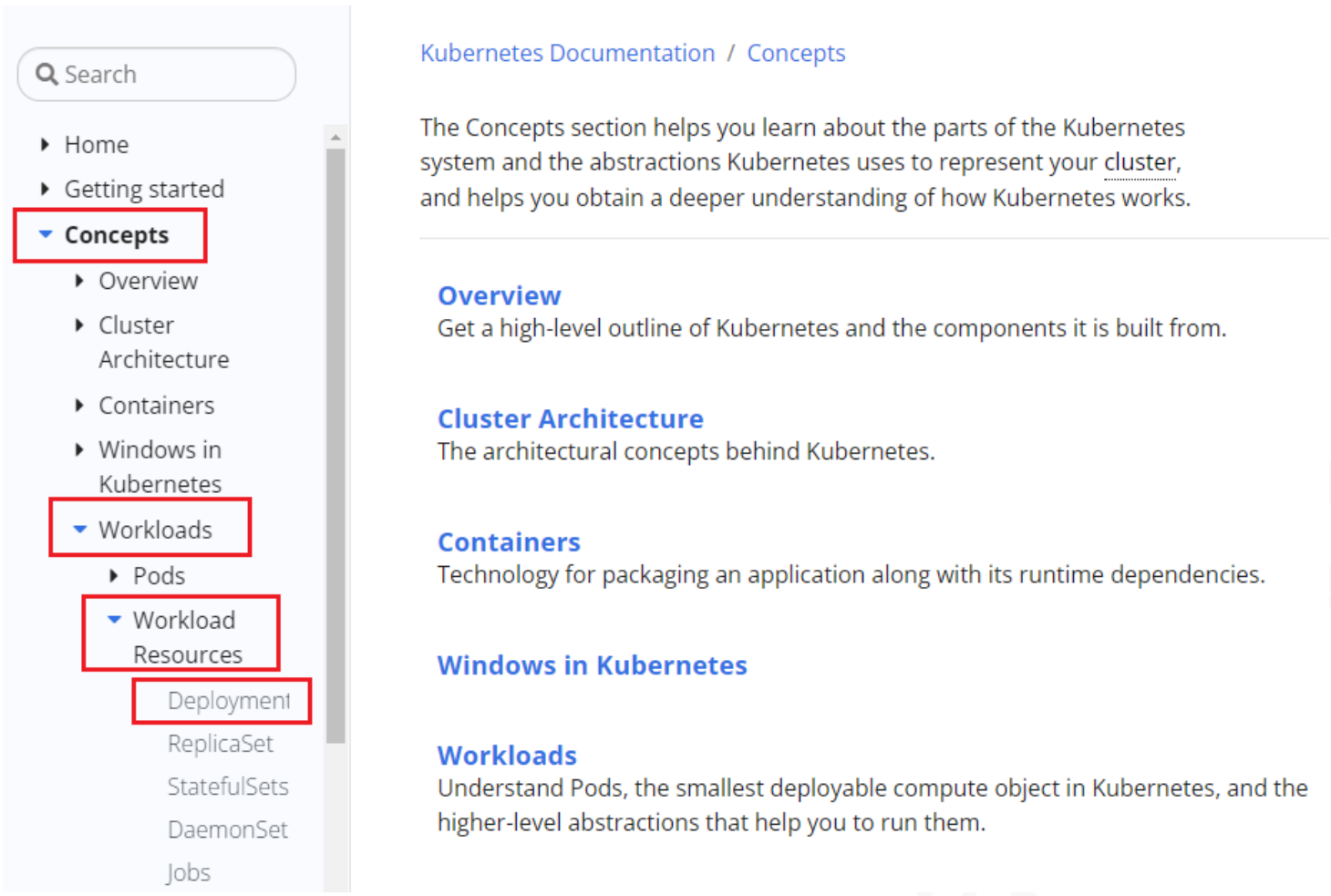
配置此 service，以通过各个 Pod 所在的节点上的 NodePort 来公开他们。

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

**参考链接

（需要复制官网网页的内容）

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/



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           20h   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。
注意，如果新增的内容不对，哪怕错一个空格。第一次:wq 报错，第二次:wq 回滚并退出，也就是不会保存任何你错误修改的内容。

```
    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:~$
```

```
      maxUnavailable: 25%
    type: RollingUpdate
  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
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
status:
  availableReplicas: 1
```

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。

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

最后 curl 检查

考试的时候，在 node01 无法 curl 通，需要 ssh 到这道题的工作节点后，才能 curl 通。所以只需要 get 检查一下，service 有 CLUSTER-IP 就行，不用 curl。

kubectl get svc -o wide

curl service 的 CLUSTER-IP 的 80 端口（curl 10.107.167.39:80）

```
candidate@node01:~$ kubectl get svc -o wide
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   SELECTOR
front-end-svc  NodePort    10.107.167.39   <none>        80:31701/TCP   28m   app=front-end
kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP        23h   <none>
candidate@node01:~$
candidate@node01:~$ curl 10.107.167.39:80
Hello World ^_^
candidate@node01:~$
```

扩展：

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    10.107.167.39   <none>        80:31701/TCP   22m   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           20h   nginx        vicuu/nginx:hello   app=front-end
candidate@node01:~$
candidate@node01:~$ kubectl get pod front-end-55f9bb474b-v8d8d --show-labels
NAME                         READY   STATUS    RESTARTS   AGE   LABELS
front-end-55f9bb474b-v8d8d   1/1     Running   0          23m   app=front-end,pod-template-hash=55f9bb474b
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    10.107.167.39   <none>        80:31701/TCP   20m   <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           20h   nginx        vicuu/nginx:hello   app=front-end
candidate@node01:~$
candidate@node01:~$ kubectl get pod front-end-55f9bb474b-v8d8d --show-labels
NAME                         READY   STATUS    RESTARTS   AGE   LABELS
front-end-55f9bb474b-v8d8d   1/1     Running   0          21m   app=front-end,pod-template-hash=55f9bb474b
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。
```

```
  resourceVersion: "75988"
  uid: 099f989d-1fdf-4c03-907b-bccaf0457bbf
spec:
  clusterIP: 10.107.167.39
  clusterIPs:
  - 10.107.167.39
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - nodePort: 31701
    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:~$
candidate@node01:~$ kubectl get svc front-end-svc -o wide
NAME           TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   SELECTOR
front-end-svc  NodePort   10.107.167.39   <none>        80:31701/TCP   22m   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           20h   nginx        vicuu/nginx:hello   app=front-end
candidate@node01:~$
candidate@node01:~$ kubectl get pod front-end-55f9bb474b-v8d8d --show-labels
NAME                         READY   STATUS    RESTARTS   AGE   LABELS
front-end-55f9bb474b-v8d8d   1/1     Running   0          23m   app=front-end,pod-template-hash=55f9bb474b
candidate@node01:~$
```

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

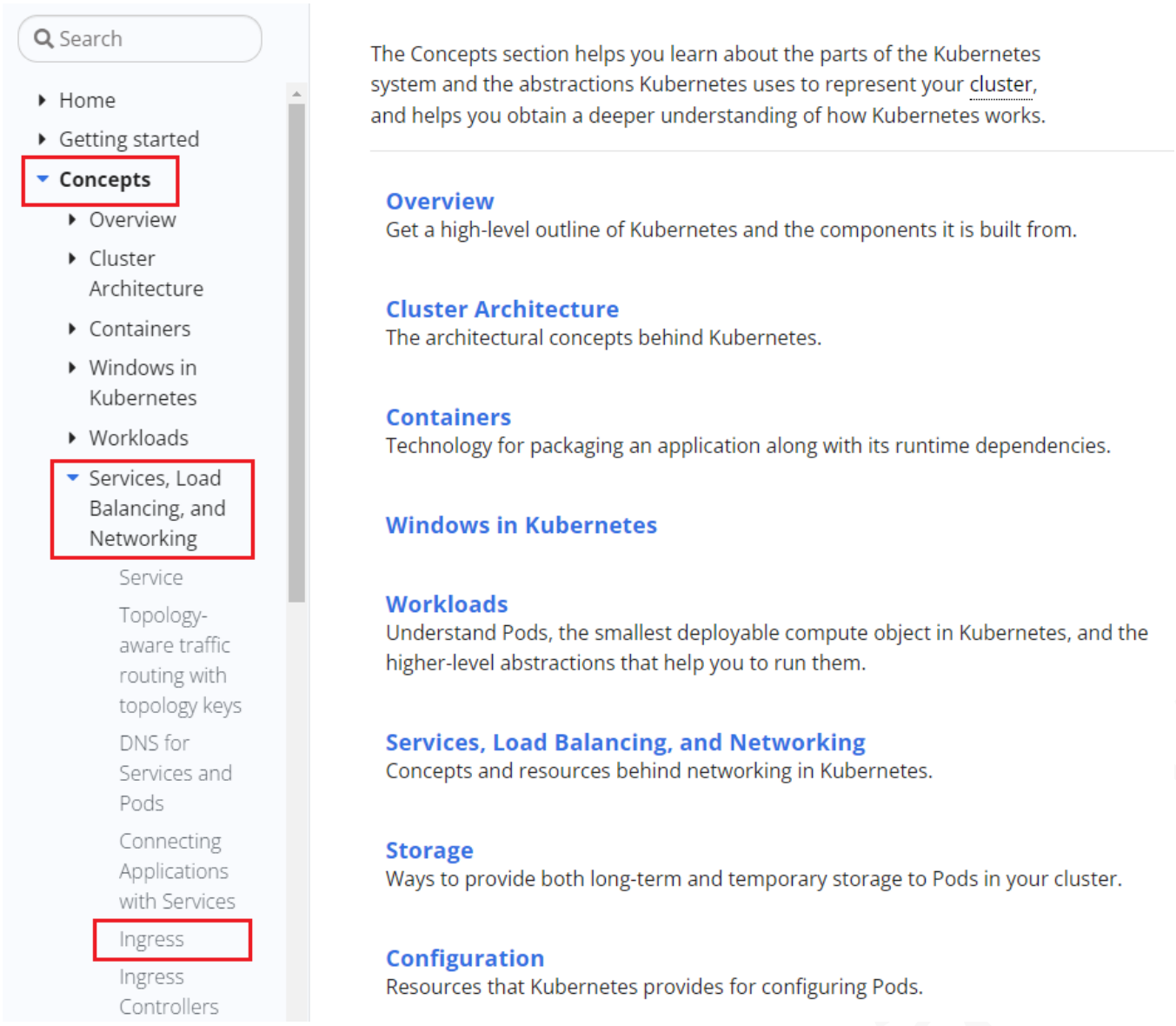
可以使用以下命令检查服务 hello 的可用性，该命令应返回 hello：
curl -kL <INTERNAL_IP>/hello

考点：Ingress 的创建

**参考链接

（需要复制官网网页的内容）

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/



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
```

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

先检查环境里 ingressclass 的名字，通过命令查看到 ingressclass 的名字叫 nginx，下面创建 ingress 需要使用。
kubectl get ingressclass

```
candidate@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
candidate@node01:~$
candidate@node01:~$ kubectl get ingressclass
NAME    CONTROLLER             PARAMETERS   AGE
nginx   k8s.io/ingress-nginx   <none>       23h
candidate@node01:~$
```

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

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ping
  namespace: ing-internal
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx   #这里写上面查出来的 ingressclass 的名字
  rules:
  - http:
      paths:
      - path: /hello
        pathType: Prefix
        backend:
          service:
            name: hello
            port:
              number: 5678
```

创建
kubectl apply -f ingress.yaml

```
candidate@node01:~$ vim ingress.yaml
candidate@node01:~$ cat ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ping
  namespace: ing-internal
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
      - path: /hello
        pathType: Prefix
        backend:
          service:
            name: hello
            port:
              number: 5678

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

检查
查看 ingress 的 IP，然后通过提供的 curl 命令，测试 ingress 是否通。
创建 ingress 后，需要等约 3 分钟后，才会获取到 IP 地址。

kubectl -n ing-internal get ingress
curl ingress 的 ip 地址/hello（curl 10.110.140.170/hello），这个/hello 对应上面创建 ingress 的- path: /hello 字段。

```
candidate@node01:~$ kubectl -n ing-internal get ingress
NAME   CLASS   HOSTS   ADDRESS          PORTS   AGE
ping   nginx   *       10.110.140.170   80      2m9s
candidate@node01:~$
candidate@node01:~$ curl 10.110.140.170/hello
Hello World ^_^
candidate@node01:~$
```

方法 2：（暂不推荐，考试时据说有点问题）
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、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

强烈建议背过操作命令，如果非要参考，可以使用-h 帮助。
kubectl top pod -h

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 的意思
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-799bc675cd-zr7jv   2m           3Mi
cpu-top     nginx-host-846bf49987-mf4mr   0m           2Mi
cpu-top     test0-6c665fdc6c-kbm76        0m           4Mi
candidate@node01:~$
candidate@node01:~$ echo "redis-test-799bc675cd-zr7jv" > /opt/KUTR000401/KUTR00401.txt
candidate@node01:~$
candidate@node01:~$ cat /opt/KUTR000401/KUTR00401.txt
redis-test-799bc675cd-zr7jv
candidate@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

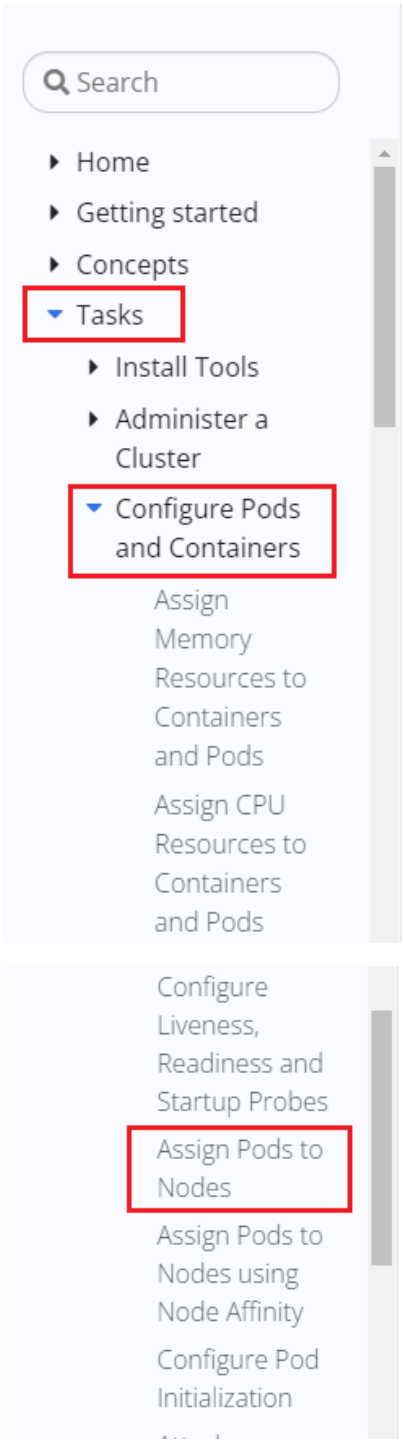
Task
按如下要求调度一个 pod：
名称：nginx-kusc00401
Image：nginx
Node selector：disk=ssd

考点：nodeSelect 属性的使用

**参考链接

（需要复制官网网页的内容）

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，因为没有创建，所以需要创建。
kubectl get pod -A|grep nginx-kusc00401

#检查一下 node 的标签，考试环境和模拟环境，都已经提前设置好 labels 了。
kubectl get nodes --show-labels|grep 'disk=ssd'

```
candidate@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
candidate@node01:~$
candidate@node01:~$ kubectl get pod -A | grep nginx-kusc00401
candidate@node01:~$
candidate@node01:~$ kubectl get nodes --show-labels|grep 'disk=ssd'
node01     Ready    <none>          26h    v1.29.0   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,disk=ssd,kubernetes.io/arch=amd64,kubernetes.io/hostname=node01,kubernetes.io/os=linux
candidate@node01:~$
```

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

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

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

检查
确保 pod 是 running 的
kubectl get pod nginx-kusc00401 -o wide

```
candidate@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          119s   10.244.196.158   node01   <none>           <none>
candidate@node01:~$
```

还有一个方法：
可以先生成 yaml，然后再修改 yaml，最后创建
kubectl run nginx-kusc00401 --image=nginx --dry-run=client -o yaml > pod.yaml
vim pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: nginx-kusc00401
  name: nginx-kusc00401
spec:
  containers:
  - image: nginx
    name: nginx-kusc00401
    imagePullPolicy: IfNotPresent
    resources: {}
  nodeSelector:
    disk: ssd
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
~
```

新增内容

kubectl apply -f pod.yaml

```
candidate@node01:~$
candidate@node01:~$
candidate@node01:~$ kubectl run nginx-kusc00401 --image=nginx --dry-run=client -o yaml > pod.yaml
candidate@node01:~$ vi pod.yaml
candidate@node01:~$
candidate@node01:~$ kubectl apply -f pod.yaml
pod/nginx-kusc00401 created
candidate@node01:~$
candidate@node01:~$ kubectl get pod
NAME                              READY   STATUS    RESTARTS        AGE
11-factor-app                     1/1     Running   2 (7m45s ago)   13h
foo                               1/1     Running   2 (7m45s ago)   13h
front-end-cfdfbdb95-v7q4t         1/1     Running   3 (7m43s ago)   15h
nginx-kusc00401                   1/1     Running   0               31s
presentation-856b8578cd-gd28p     1/1     Running   2 (7m43s ago)   13h
candidate@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

强烈建议背过操作命令，如果非要参考，可以使用-h 帮助。
kubectl -h

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

先检查一共有多少就绪节点，状态为 Ready 表示就绪，下图可以看到一共是 3 个就绪节点。
kubectl get nodes

```
candidate@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
candidate@node01:~$
candidate@node01:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   26h   v1.29.0
node01     Ready    <none>          26h   v1.29.0
node02     Ready    <none>          26h   v1.29.0
candidate@node01:~$
```

因为题目要求，不包括被打上 Taint：NoSchedule 的就绪节点，所以要排除 NoSchedule 的。

然后使用如下命令，就能一眼看出来，几个打了 NoSchedule，几个没有打的。
kubectl describe nodes | grep -i Taints

检查查出来的数字写入文件
下图可见 1 个 master01 打了 NoSchedule，所以 3-1=2
echo "查出来的数字" > /opt/KUSC00402/kusc00402.txt

```
candidate@node01:~$ kubectl describe nodes | grep -i Taints
Taints:             node-role.kubernetes.io/control-plane:NoSchedule
Taints:             <none>
Taints:             <none>
candidate@node01:~$
candidate@node01:~$ echo "2" > /opt/KUSC00402/kusc00402.txt
candidate@node01:~$
```

检查
cat /opt/KUSC00402/kusc00402.txt

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

还有一个方法：
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

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

扩展知识：
查看所有节点名字和节点标签的对应情况：

```
candidate@node01:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   27h   v1.29.0
node01     Ready    <none>          26h   v1.29.0
node02     Ready    <none>          26h   v1.29.0
candidate@node01:~$
candidate@node01:~$ kubectl describe nodes | grep -i -e Taints -e 'Hostname:'
Taints:             node-role.kubernetes.io/control-plane:NoSchedule
  Hostname:    master01
Taints:             <none>
  Hostname:    node01
Taints:             <none>
  Hostname:    node02
candidate@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:

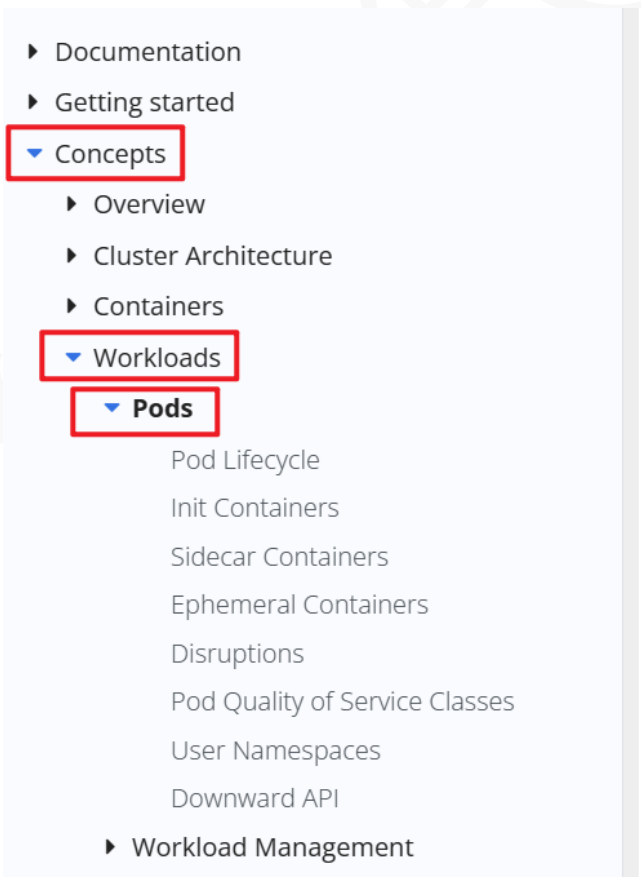
- nginx
- memcached

考点：pod 概念

**参考链接

（需要复制官网网页的内容）

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/



Using Pods

The following is an example of a Pod which consists of a container running the image `nginx:1.14.2`.

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

To create the Pod shown above, run the following command:

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: memcached
      image: memcached
      imagePullPolicy: IfNotPresent
```

创建

kubectl apply -f pod-kucc.yaml

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

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

检查
确保 pod 是 running 的
kubectl get pod kucc8

```
candidate@node01:~$ kubectl get pod kucc8
NAME    READY   STATUS    RESTARTS   AGE
kucc8   2/2     Running   0          20s
candidate@node01:~$
```

10、创建 PV

考题

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

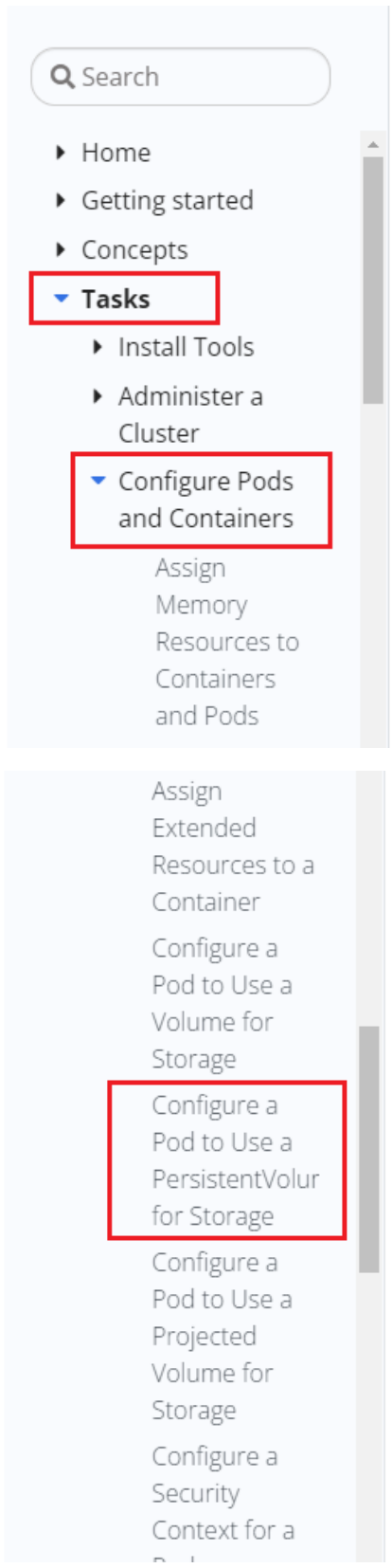
Task
创建名为 app-config 的 persistent volume，容量为 1Gi，访问模式为 ReadWriteMany。
volume 类型为 hostPath，位于 /srv/app-config

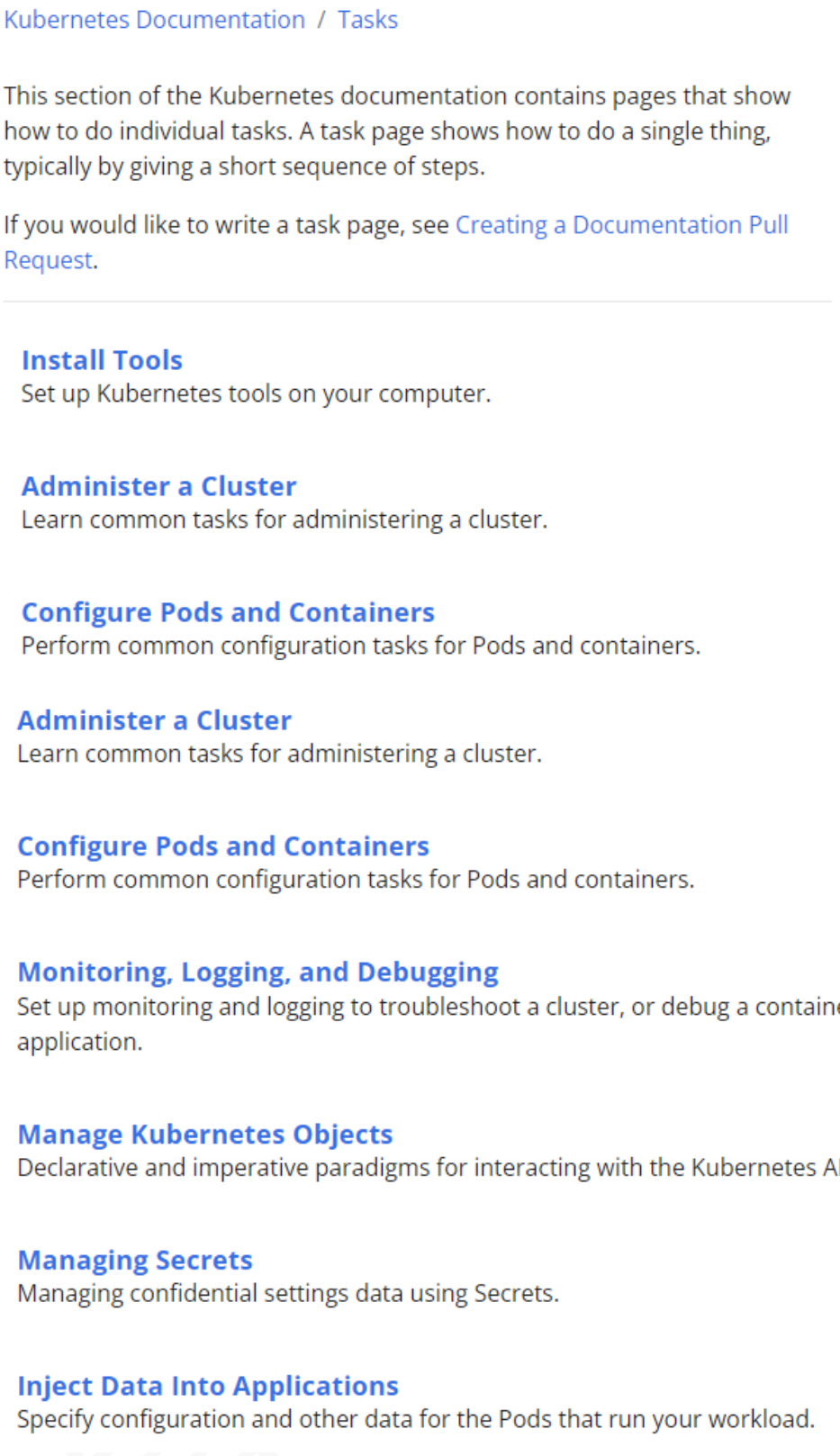
考点：hostPath 类型的 pv

**参考链接

（需要复制官网网页的内容）

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/





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"
```

题目没要求，则不写

根据题目要求写

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 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。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: app-config
spec:
  capacity:
    storage: 1Gi     #按照题目要求的大小写
  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

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

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

检查

kubectl get pv

RWX 是 ReadWriteMany，RWO 是 ReadWriteOnce。

```
candidate@node01:~$ kubectl get pv
NAME         CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS      VOLUMEATTRIBUTESCLASS   REASON   AGE
app-config   1Gi        RWX            Retain           Available                             <unset>                          10s
pv01         10Mi       RWO            Recycle          Available           csi-hostpath-sc   <unset>                          76m
candidate@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。

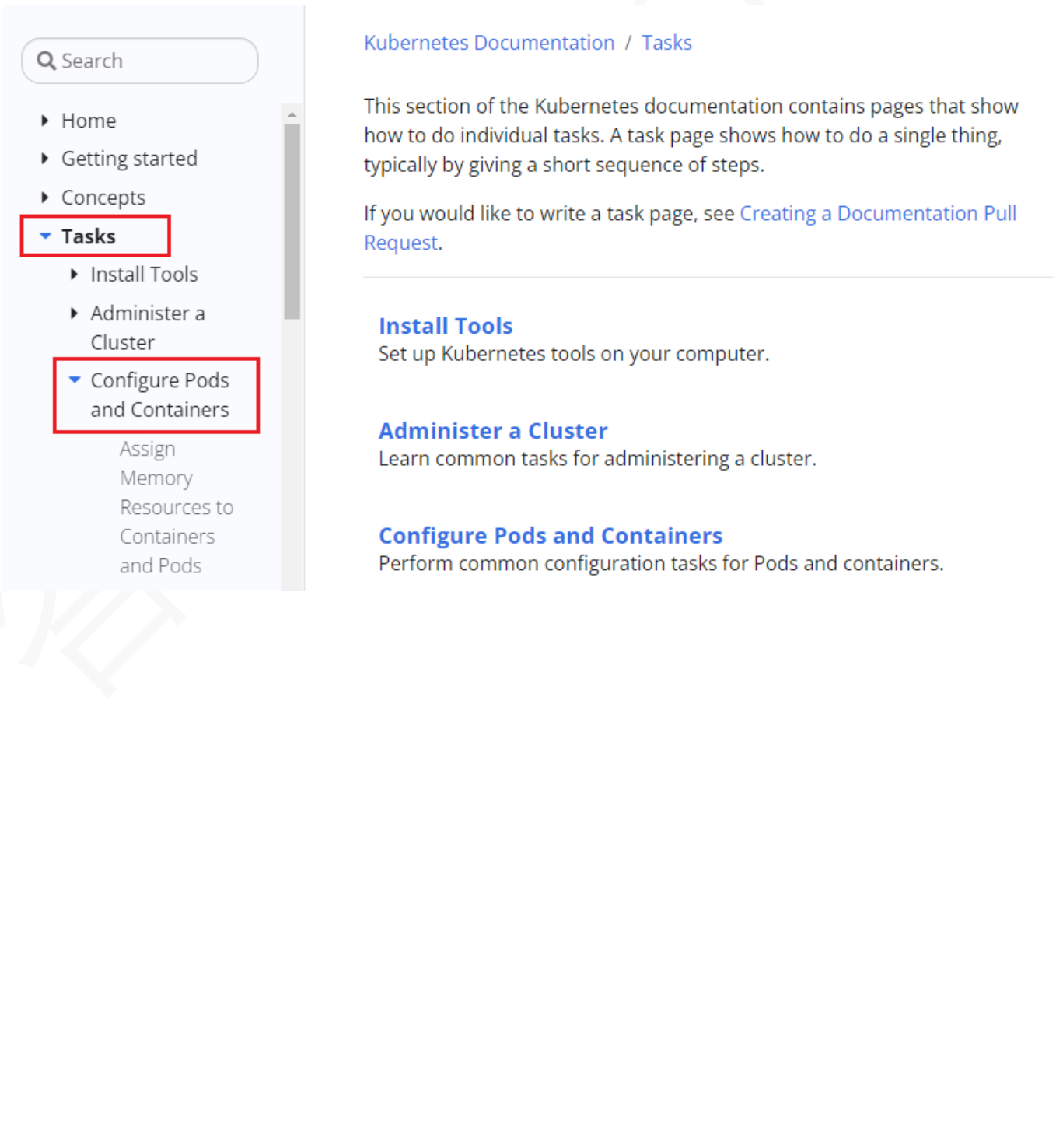
最后，使用 kubectl edit 或 kubectl patch 将 PersistentVolumeClaim 的容量扩展为 70Mi，并记录此更改。

考点：pvc 的创建 class 属性的使用，--record 记录变更

**参考链接

（需要复制官网网页的内容）

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/



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 来请求物理存储。 在本练习中，你将创建一个 PersistentVolumeClaim，它请求至少 3 GB 容量的卷， 该卷一次最多可以为一个节点提供读写访问。

下面是 PersistentVolumeClaim 的配置文件：

pods/

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

根据题目要求修改

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。
请注意，这道 pvc 的题，和上面 pv 的题，是两道不同的题。两者之间没有任何关联。

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     #模拟环境里没有使用支持动态扩容的存储，如 CEPH，所以没有 storageClass，但是这里还是要按照题目要求写上。
  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     #按照题目要求的大小写
```

创建 PVC
有些同学 apply 创建后，发现不对，就 delete 了 pvc，修改 yaml 后重新创建，这样在模拟环境是不行的，因为模拟环境没有使用支持动态扩容的存储，导致 delete 后会造成 pvc 找不到 pv，pvc 状态一直是 Pending。
所以如果创建的有问题，那么只能将 3 台虚拟机回退快照，重新开机后，再重新做这道题。

kubectl apply -f pvc.yaml

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

candidate@node01:~$ kubectl apply -f pvc.yaml
persistentvolumeclaim/pv-volume created
candidate@node01:~$
```

检查 pvc，确保状态是 Bound
kubectl get pvc

```
candidate@node01:~$ kubectl get pvc
NAME        STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS      VOLUMEATTRIBUTESCLASS   AGE
pv-volume   Bound    pv01     10Mi       RWO            csi-hostpath-sc   <unset>                 5s
candidate@node01:~$
```

创建 pod
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

```
candidate@node01:~$ vim pvc-pod.yaml
candidate@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
      volumeMounts:
        - mountPath: "/usr/share/nginx/html"
          name: task-pv-storage

candidate@node01:~$ kubectl apply -f pvc-pod.yaml
pod/web-server created
candidate@node01:~$
```

检查
确保 pod 是 running 的
kubectl get pod web-server

```
candidate@node01:~$ kubectl get pod web-server
NAME         READY   STATUS    RESTARTS   AGE
web-server   1/1     Running   0          54s
candidate@node01:~$
```

更改大小，并记录过程。
将 storage: 10Mi 改为 storage: 70Mi　（这一步，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:

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。
kubectl edit pvc pv-volume --record

```
candidate@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 prov
isioned pvc can be resized and the storageclass that provisions the pvc must support resize
You can run `kubectl replace -f /tmp/kubectl-edit-640170612.yaml` to try this update again.
candidate@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。如果换成 ceph 做后端存储，可以动态扩容了。但是集群资源太少，无法做 ceph。

```
  resourceVersion: "124192"
  uid: 180a90fc-b89d-4871-8da0-10f3073533fc
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

强烈建议背过操作命令，如果非要参考，可以使用-h 帮助。
kubectl -h

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

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

检查
cat /opt/KUTR00101/foo

```
candidate@node01:~$ cat /opt/KUTR00101/foo
2024/02/16 03:08:07 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
candidate@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。

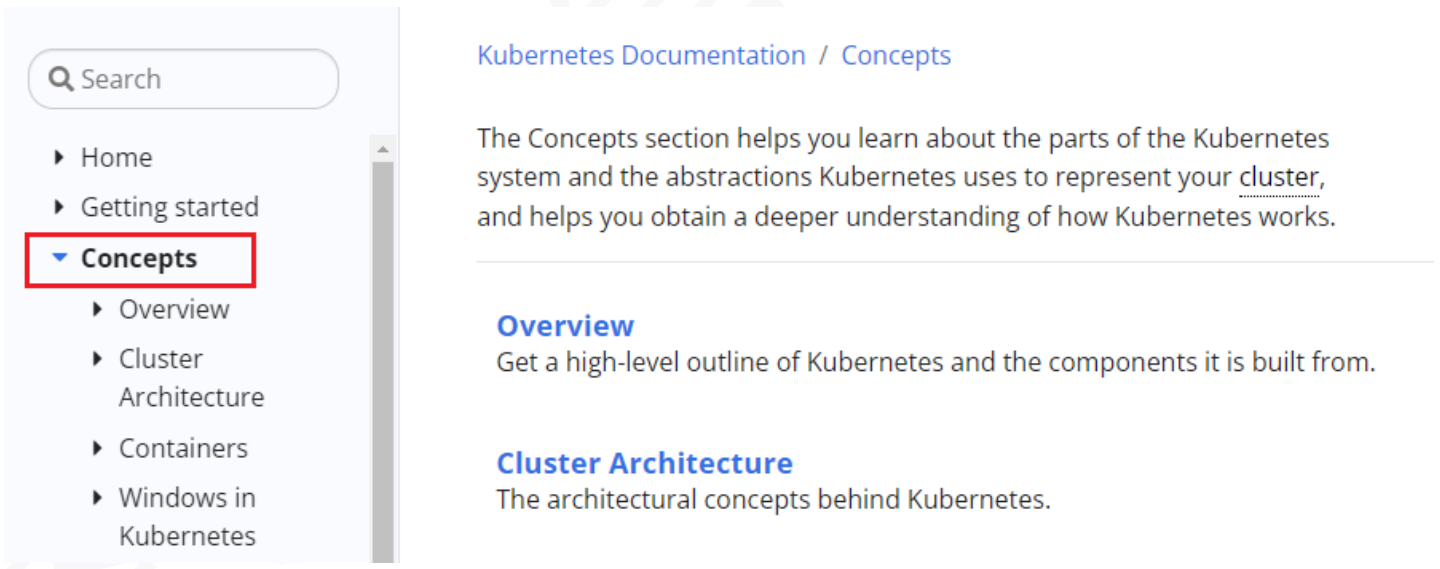
考题翻译成白话，就是:
添加一个名为 sidecar 的边车容器(使用 busybox 镜像)，加到已有的 pod 11-factor-app 中。
确保 sidecar 容器能够输出 /var/log/11-factor-app.log 的信息。
使用 volume 挂载 /var/log 目录，确保 sidecar 能访问 11-factor-app.log 文件

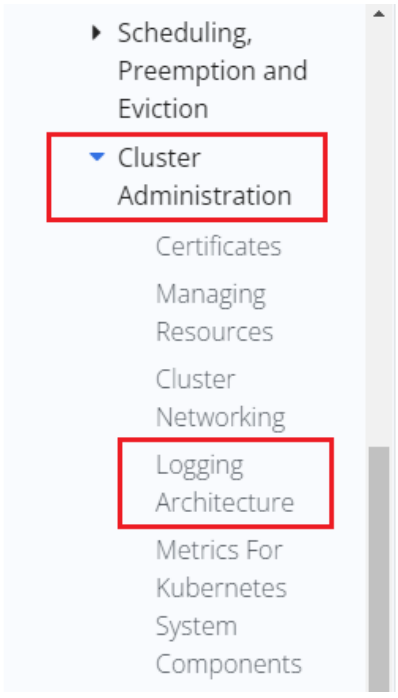
考点：pod 两个容器共享存储卷

**参考链接

（需要复制官网网页的内容）

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/





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.

```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

这道题的大体流程为：

1、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

2、复制一份新 yaml 文件，添加 一个名称为 sidecar 的容器

3、挂载 emptyDir 的卷，确保两个容器都挂载了 /var/log 目录

4、创建含有 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

```
candidate@node01:~$ kubectl config use-context k8s
error: no context exists with the name: "k8s"
candidate@node01:~$
candidate@node01:~$ kubectl get pod 11-factor-app -o yaml > varlog.yaml
candidate@node01:~$
candidate@node01:~$ cp varlog.yaml varlog-bak.yaml
candidate@node01:~$
candidate@node01:~$ vim varlog.yaml
candidate@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: kube-api-access-gt98r
      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
    imagePullPolicy: IfNotPresent    #模拟环境里，建议加上这句，可以直接使用环境里的 busybox 镜像。因为国内网络不好，从网络可能下载不了。
    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 volumes 下面添加。
  - name: varlog    #新加内容，注意找好位置。
    emptyDir: {}      #新加内容
  - name: kube-api-access-gt98r
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
          expirationSeconds: 3607
```

```yaml
  resourceVersion: "24841"
  uid: 26b79d99-719a-4ed5-8da7-e704c26d1e1d
spec:
  containers:
  - args:
    - /bin/sh
    - -c
    - |
      i=0; while true; do
        echo "$(date) INFO $i" >> /var/log/11-factor-app.log;
        i=$((i+1));
        sleep 1;
      done
    image: busybox
    imagePullPolicy: IfNotPresent
    name: count
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: kube-api-access-gt98r
      readOnly: true
    - name: varlog
      mountPath: /var/log
  - name: sidecar
    image: busybox
    imagePullPolicy: IfNotPresent
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/11-factor-app.log']
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  nodeName: node01
  nodeSelector:
    kubernetes.io/hostname: node01
  preemptionPolicy: PreemptLowerPriority
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: varlog
    emptyDir: {}
  - name: kube-api-access-gt98r
    projected:
      defaultMode: 420
      sources:
      - serviceAccountToken:
          expirationSeconds: 3607
          path: token
      - configMap:
          items:
          - key: ca.crt
            path: ca.crt
          name: kube-root-ca.crt
      - downwardAPI:
          items:
```

删除原先的 pod，大约需要等 2 分钟，中途不要 CTRL+C 终止命令。

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

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

```
candidate@node01:~$ kubectl logs 11-factor-app sidecar
Fri Feb 16 14:55:27 UTC 2024 INFO 0
Fri Feb 16 14:55:28 UTC 2024 INFO 1
Fri Feb 16 14:55:29 UTC 2024 INFO 2
Fri Feb 16 14:55:30 UTC 2024 INFO 3
Fri Feb 16 14:55:31 UTC 2024 INFO 4
Fri Feb 16 14:55:32 UTC 2024 INFO 5
Fri Feb 16 14:55:33 UTC 2024 INFO 6
Fri Feb 16 14:55:34 UTC 2024 INFO 7
Fri Feb 16 14:55:35 UTC 2024 INFO 8
Fri Feb 16 14:55:36 UTC 2024 INFO 9
Fri Feb 16 14:55:37 UTC 2024 INFO 10
Fri Feb 16 14:55:38 UTC 2024 INFO 11
Fri Feb 16 14:55:39 UTC 2024 INFO 12
Fri Feb 16 14:55:40 UTC 2024 INFO 13
Fri Feb 16 14:55:41 UTC 2024 INFO 14
Fri Feb 16 14:55:42 UTC 2024 INFO 15
```

```
candidate@node01:~$ kubectl exec 11-factor-app -c sidecar -- tail -f /var/log/11-factor-app.log
Fri Feb 16 15:00:12 UTC 2024 INFO 285
Fri Feb 16 15:00:13 UTC 2024 INFO 286
Fri Feb 16 15:00:14 UTC 2024 INFO 287
Fri Feb 16 15:00:15 UTC 2024 INFO 288
Fri Feb 16 15:00:16 UTC 2024 INFO 289
Fri Feb 16 15:00:17 UTC 2024 INFO 290
Fri Feb 16 15:00:18 UTC 2024 INFO 291
Fri Feb 16 15:00:19 UTC 2024 INFO 292
Fri Feb 16 15:00:20 UTC 2024 INFO 293
Fri Feb 16 15:00:21 UTC 2024 INFO 294
Fri Feb 16 15:00:22 UTC 2024 INFO 295
```

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 1.29.0。**仅将 master 节点**上的所有 Kubernetes 控制平面和节点组件升级到版本 1.29.1。

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，注意题目要求你升级的版本，根据题目要求输入正确的版本号!!!）
（比如考题是，正在运行的版本是 1.29.1，要求升级到 1.29.2， 下面答案的命令也应该做相应的改变。）

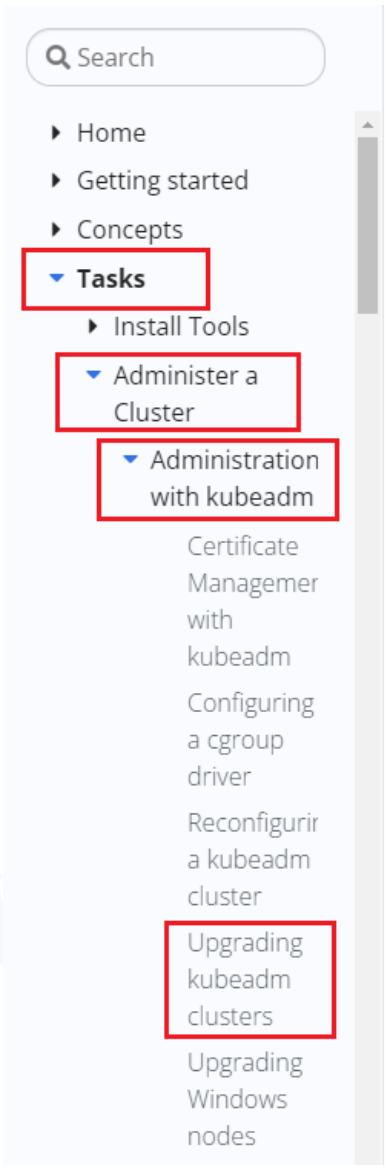
考点：如何离线主机，并升级控制面板和升级节点

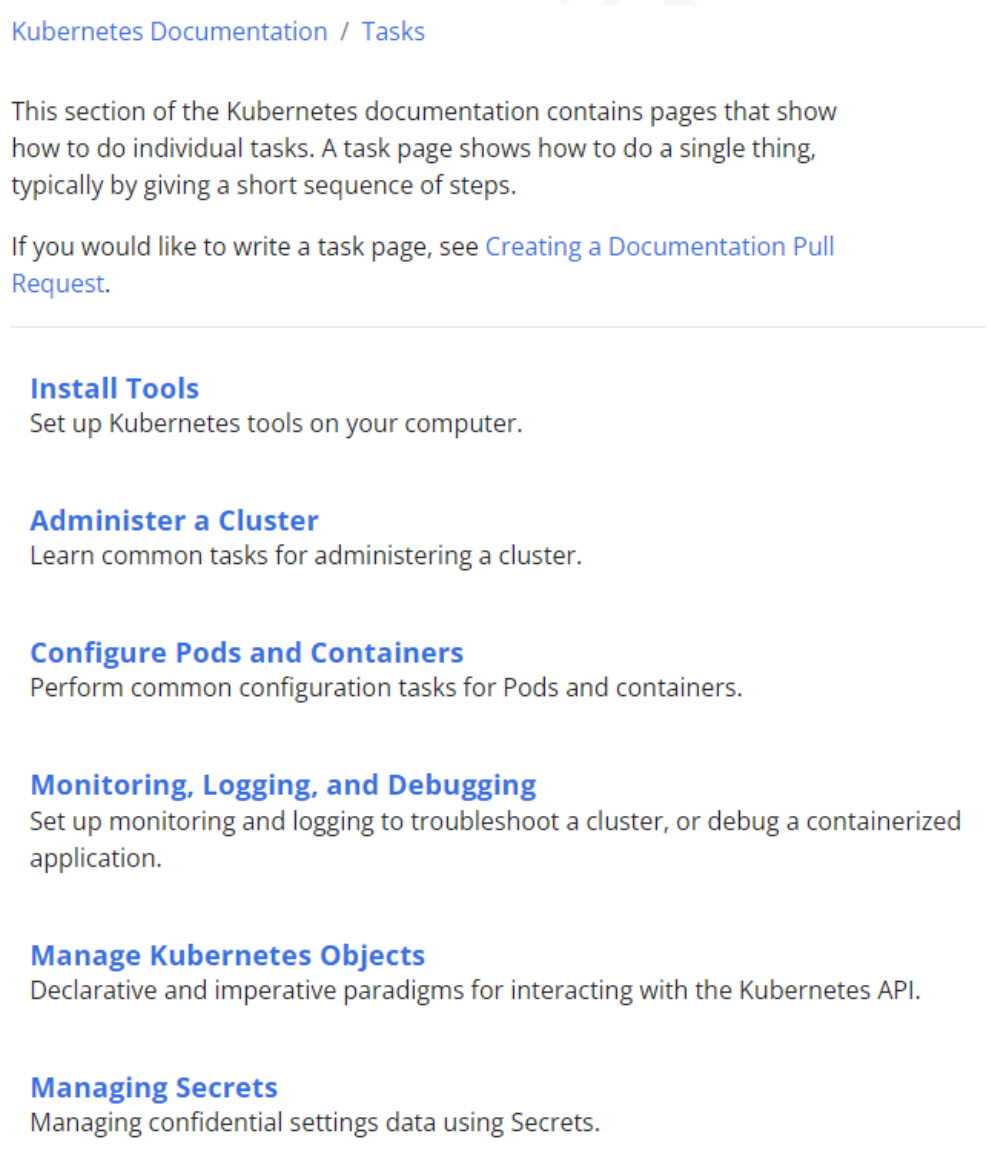
**参考链接

没必要参考官网，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/





腾空节点

将节点标记为不可调度并驱逐所有负载，准备节点的维护：

```
# 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
```

执行 “kubeadm upgrade”

对于第一个控制面节点

1. 升级 kubeadm：

Ubuntu, Debian or HypriotOS | CentOS, RHEL or Fedora

```
# 用最新的补丁版本号替换 1.29.x-* 中的 x
apt-mark unhold kubeadm && \
apt-get update && apt-get install -y kubeadm='1.29.x-*' && \
apt-mark hold kubeadm
```

2. 验证下载操作正常，并且 kubeadm 版本正确：

```
kubeadm version
```

3. 验证升级计划：

```
kubeadm upgrade plan
```

4. 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 v1.29.x
```

升级 kubelet 和 kubectl

1. 升级 kubelet 和 kubectl：

Ubuntu, Debian or HypriotOS | CentOS, RHEL or Fedora

```
# 用最新的补丁版本替换 1.29.x-* 中的 x
apt-mark unhold kubelet kubectl && \
apt-get update && apt-get install -y kubelet='1.29.x-*' kubectl='1.29.x-*' && \
apt-mark hold kubelet kubectl
```

2. 重启 kubelet：

```
sudo systemctl daemon-reload
sudo systemctl restart kubelet
```

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

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

从 1.27.2 升级到 1.27.3，查出来的是 v1.27.,2
从 1.28.0 升级到 1.28.1，查出来的是 v1.28.0
从 1.29.0 升级到 1.29.1，查出来的是 v1.29.0

```
candidate@node01:~$ kubectl config use-context mk8s
error: no context exists with the name: "mk8s"
candidate@node01:~$
candidate@node01:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   28h   v1.29.0
node01     Ready    <none>          28h   v1.29.0
node02     Ready    <none>          28h   v1.29.0
candidate@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 cordon master01
node/master01 cordoned
candidate@node01:~$
candidate@node01:~$ kubectl drain master01 --ignore-daemonsets
node/master01 already cordoned
Warning: ignoring DaemonSet-managed Pods: calico-system/calico-node-8xgq7, calico-system/csi-node-driver-29ckz, kube-system/kube-proxy-vgkdl
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:~#
```

更新 apt 仓库源
apt-get update

通过下面命令，可以查出来具体要升级的版本号，包括后面的版本号数字。比如 1.28.1-00 1.29.1-1.1 等等。这样后续升级命令才能写出具体版本号数字。
这里要注意，如果考题要求是从 1.27.2 升级到 1.27.3，这里就是写 apt-cache show kubeadm|grep 1.27.3
如果考题是从 1.28.0 升级到 1.28.1，这里就是写 apt-cache show kubeadm|grep 1.28.1

apt-cache show kubeadm|grep 1.29.1

```
root@master01:~# apt-cache show kubeadm|grep 1.29.1
Version: 1.29.1-1.1
Filename: amd64/kubeadm_1.29.1-1.1_amd64.deb
```

注意 kubeadm=后面写上一步查出来的具体版本号数字，比如 1.27.3-00 1.28.1-00 1.29.1-1.1 ，以查出来的为准。

apt-get install kubeadm=1.29.1-1.1

```
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]
Hit:3 http://mirrors.ustc.edu.cn/ubuntu focal-backports InRelease
Hit:4 http://mirrors.ustc.edu.cn/ubuntu focal-security InRelease
Hit:5 http://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu focal InRelease
Get:6 http://mirrors.ustc.edu.cn/ubuntu focal-updates/main amd64 Packages [3,095 kB]
Get:8 http://mirrors.ustc.edu.cn/ubuntu focal-updates/multiverse amd64 Packages [26.1 kB]
Hit:7 https://prod-cdn.packages.k8s.io/repositories/isv:/kubernetes:/core:/stable:/v1.29/deb  InRelease
Fetched 3,235 kB in 3s (972 kB/s)
Reading package lists... Done
root@master01:~#
root@master01:~# apt-cache show kubeadm|grep 1.29.1
Version: 1.29.1-1.1
Filename: amd64/kubeadm_1.29.1-1.1_amd64.deb
root@master01:~#
root@master01:~# apt-get install kubeadm=1.29.1-1.1
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 80 not upgraded.
Need to get 0 B/10.1 MB of archives.
After this operation, 8,192 B of additional disk space will be used.
(Reading database ... 111141 files and directories currently installed.)
Preparing to unpack .../kubeadm_1.29.1-1.1_amd64.deb ...
Unpacking kubeadm (1.29.1-1.1) over (1.29.0-1.1) ...
Setting up kubeadm (1.29.1-1.1) ...
root@master01:~#
```

如果是 1.28.1，则查出来的是 1.28.1-00

```
root@master01:~# apt-cache show kubeadm|grep 1.28.1
Version: 1.28.1-00
Filename: pool/kubeadm_1.28.1-00_amd64_9cc8ccc25cf248e11868d29c3b7a38a147113088ddc8bb065bfc872475ea5a9a.deb
root@master01:~#
```

检查 kubeadm 升级后的版本

kubeadm version

```
root@master01:~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"29", GitVersion:"v1.29.1", GitCommit:"bc401b91f2782410b3fb3f9acf43a995c4de90d2", GitT
reeState:"clean", BuildDate:"2024-01-17T15:49:02Z", GoVersion:"go1.21.6", Compiler:"gc", Platform:"linux/amd64"}
root@master01:~#
```

验证升级计划，

注意，这里只会显示最后一个版本，但是我们具体要升级到哪个版本，要按照考试题目里要求的版本号数字写。

kubeadm upgrade plan

```
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: v1.29.0
[upgrade/versions] kubeadm version: v1.29.1
[upgrade/versions] Target version: v1.29.2
[upgrade/versions] Latest version in the v1.29 series: v1.29.2

Components that must be upgraded manually after you have upgraded the control plane with 'kubeadm upgrade apply':
COMPONENT   CURRENT       TARGET
kubelet     3 x v1.29.0   v1.29.2

Upgrade to the latest version in the v1.29 series:

COMPONENT                 CURRENT    TARGET
kube-apiserver            v1.29.0    v1.29.2
kube-controller-manager   v1.29.0    v1.29.2
kube-scheduler            v1.29.0    v1.29.2
kube-proxy                v1.29.0    v1.29.2
CoreDNS                   v1.11.1    v1.11.1
etcd                      3.5.10-0   3.5.10-0

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

        kubeadm upgrade apply v1.29.2

Note: Before you can perform this upgrade, you have to update kubeadm to v1.29.2.

_____


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:~#
```

注意，不要直接使用。具体版本号，以题目要求的为准

开始应用升级，不能直接写上面的，因为 v1.29.2 不是题目要求的，题目要求的是 v1.29.1。升级时有提示，要输入 y。根据各自网络情况，大约要等 5 分钟。
考试时，不加--etcd-upgrade=false，etcd 也不会升级，但能想着的建议加上--etcd-upgrade=false，因为默认这个 etcd 是 true。其他都是 false，不会升级。
kubeadm upgrade apply v1.29.1 --etcd-upgrade=false

```
root@master01:~# kubeadm upgrade apply v1.29.1
[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 "v1.29.1"
[upgrade/versions] Cluster version: v1.29.0
[upgrade/versions] kubeadm version: v1.29.1
[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'
```

输入 y

```
[upgrade/staticpods] Component "kube-scheduler" upgraded successfully!
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with the configuration for the kubelets in the cluster
[upgrade] Backing up kubelet config file to /etc/kubernetes/tmp/kubeadm-kubelet-config1834263585/config.yaml
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "super-admin.conf" kubeconfig file
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get nodes
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate cr
edentials
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.29.1". Enjoy!   升级完成

[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading your kubelets if you haven't already done so.
root@master01:~#
```

升级 kubelet

注意，版本号数字，也要使用上面查到的，比如　1.27.3-00　1.28.1-00　1.29.1-1.1

apt-get install kubelet=1.29.1-1.1

```
root@master01:~# apt-get install kubelet=1.29.1-1.1
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 80 not upgraded.
Need to get 0 B/19.8 MB of archives.
After this operation, 4,096 B of additional disk space will be used.
(Reading database ... 111141 files and directories currently installed.)
Preparing to unpack .../kubelet_1.29.1-1.1_amd64.deb ...
Unpacking kubelet (1.29.1-1.1) over (1.29.0-1.1) ...
Setting up kubelet (1.29.1-1.1) ...
root@master01:~#
```

重启 kubelet

systemctl daemon-reload

systemctl restart kubelet

```
root@master01:~# systemctl daemon-reload
root@master01:~# systemctl restart kubelet
```

查看版本

kubelet --version

```
root@master01:~# kubelet --version
Kubernetes v1.29.1
root@master01:~#
```

升级 kubectl

注意，版本号数字，也要使用上面查到的，比如　1.27.3-00　1.28.1-00　1.29.1-1.1

apt-get install kubectl=1.29.1-1.1

```
root@master01:~# apt-get install kubectl=1.29.1-1.1
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 80 not upgraded.
Need to get 0 B/10.5 MB of archives.
After this operation, 0 B of additional disk space will be used.
(Reading database ... 111141 files and directories currently installed.)
Preparing to unpack .../kubectl_1.29.1-1.1_amd64.deb ...
Unpacking kubectl (1.29.1-1.1) over (1.29.0-1.1) ...
Setting up kubectl (1.29.1-1.1) ...
root@master01:~#
```

查看版本

kubectl version

```
root@master01:~# kubectl version
Client Version: v1.29.1
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.29.1
root@master01:~#
```

切记，做完后，要退回到初始节点

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:~$
```

恢复 master01 调度，这一步务必要做，否则前功尽弃

kubectl uncordon master01

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

检查 master01 是否为 Ready，并且为题目要求升级到的版本

kubectl get node

```
candidate@node01:~$ kubectl get node
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   30h   v1.29.1
node01     Ready    <none>          30h   v1.29.0
node02     Ready    <none>          30h   v1.29.0
candidate@node01:~$
```

15、备份还原 etcd

考题

设置配置环境：

[candidate@node-1] $ kubectl config use-context xk8s

Task
您必须从 master01 主机执行所需的 etcdctl 命令。

首先，为运行在 https://127.0.0.1:2379 上的现有 etcd 实例创建快照并将快照保存到 /var/lib/backup/etcd-snapshot.db

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

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。

然后通过位于/data/backup/etcd-snapshot-previous.db 的先前备份的快照进行还原。

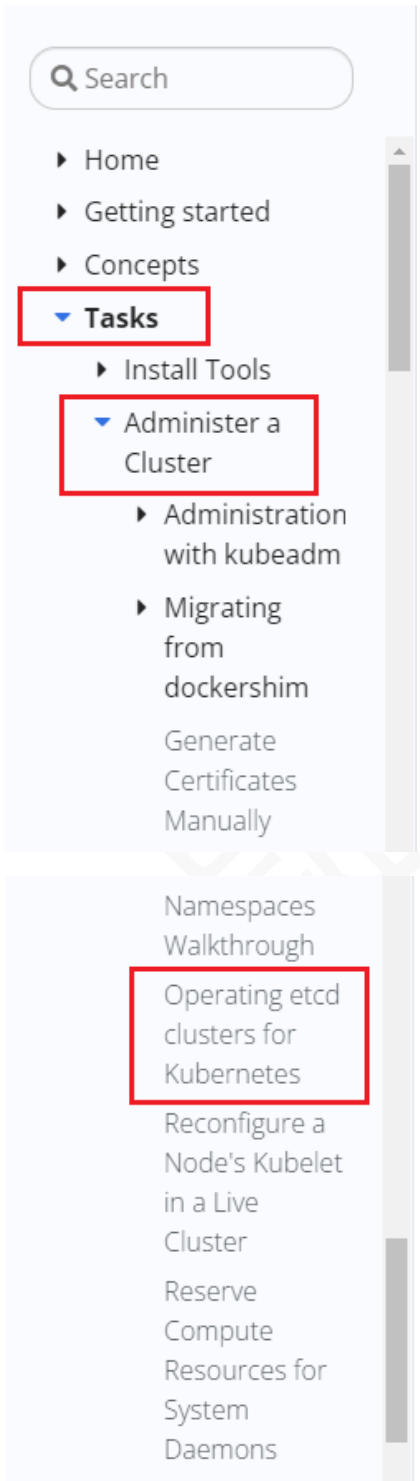
考点：etcd 的备份和还原命令

**参考链接

没必要参考官网，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

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 xk8s

```
candidate@node01:~$ kubectl config use-context xk8s
error: no context exists with the name: "xk8s"
candidate@node01:~$
```

开始操作

ssh 到 master 节点，并切换到 root 下

ssh master01

sudo -i

```
candidate@node01:~$ kubectl config use-context xk8s
error: no context exists with the name: "xk8s"
candidate@node01:~$
candidate@node01:~$ ssh master01
```

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

检查一下当前的 pod

模拟环境可以通过还原前后这些 pod 数量比较，看到还原是否成功。

kubectl get pod -A

```
root@master01:~# kubectl get pod -A
NAMESPACE          NAME                                       READY   STATUS    RESTARTS      AGE
calico-apiserver   calico-apiserver-556f6d894-7z9xn           1/1     Running   3 (13h ago)   29h
calico-apiserver   calico-apiserver-556f6d894-ncc4w           1/1     Running   3 (13h ago)   29h
calico-system      calico-kube-controllers-687c447f4b-75xfq   1/1     Running   3 (13h ago)   29h
calico-system      calico-node-8xgq7                          1/1     Running   3 (13h ago)   29h
calico-system      calico-node-d8wrr                          1/1     Running   3 (13h ago)   29h
calico-system      calico-node-mzz24                          1/1     Running   3 (13h ago)   29h
calico-system      calico-typha-6d6f5f8cbf-rjhzx              1/1     Running   0             10m
calico-system      calico-typha-6d6f5f8cbf-zcjwz              1/1     Running   3 (13h ago)   29h
calico-system      csi-node-driver-29ckz                      2/2     Running   6 (13h ago)   29h
calico-system      csi-node-driver-jwghc                      2/2     Running   6 (13h ago)   29h
calico-system      csi-node-driver-r7xxz                      2/2     Running   6 (13h ago)   29h
cpu-top            nginx-host-846bf49987-mf4mr                1/1     Running   1 (13h ago)   27h
cpu-top            redis-test-799bc675cd-zr7jv                1/1     Running   1 (13h ago)   27h
cpu-top            test0-6c665fdc6c-kbm76                     1/1     Running   1 (13h ago)   27h
default            11-factor-app                              2/2     Running   0             103m
default            foo                                        1/1     Running   1 (13h ago)   27h
default            front-end-55f9bb474b-v8d8d                 1/1     Running   0             7h24
default            kucc8                                      2/2     Running   0             178m
default            nginx-kusc00401                            1/1     Running   0             3h30
default            presentation-69bc4b5956-88vmb              1/1     Running   1 (13h ago)   27h
default            presentation-69bc4b5956-8lpxd              1/1     Running   0             12h
default            presentation-69bc4b5956-8w7tg              1/1     Running   0             12h
default            presentation-69bc4b5956-p7wtw              1/1     Running   0             12h
default            web-server                                 1/1     Running   0             56m
ing-internal       hello-57645c4896-6w46p                     1/1     Running   1 (13h ago)   27h
ingress-nginx      ingress-nginx-controller-ndrqr             1/1     Running   1 (13h ago)   27h
ingress-nginx      ingress-nginx-controller-ql7qz             1/1     Running   1 (13h ago)   27h
kube-system        coredns-857d9ff4c9-7zmvq                   1/1     Running   3 (13h ago)   28h
kube-system        coredns-857d9ff4c9-pk8vm                   1/1     Running   3 (13h ago)   30h
kube-system        etcd-master01                              1/1     Running   3 (13h ago)   30h
kube-system        kube-apiserver-master01                    1/1     Running   0             20m
kube-system        kube-controller-manager-master01           1/1     Running   0             19m
kube-system        kube-proxy-524b2                           1/1     Running   0             18m
kube-system        kube-proxy-wzsjz                           1/1     Running   0             18m
kube-system        kube-proxy-x8ntg                           1/1     Running   0             18m
kube-system        kube-scheduler-master01                    1/1     Running   0             19m
kube-system        metrics-server-5b95796896-59d5v            1/1     Running   1 (13h ago)   27h
tigera-operator    tigera-operator-55585899bf-7f4x5           1/1     Running   6 (20m ago)   29h
root@master01:~#
```

备份：

ETCDCTL_API=3 必须加，因为考试时，默认的 ETCDCTL_API 不是 3，所以执行命令会报错。所以必须指定 ETCDCTL_API=3
#注意，下面是 1 条命令，只是比较长，所以显示成 2 行了。

ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert="/opt/KUIN00601/ca.crt" --cert="/opt/KUIN00601/etcd-client.crt" --key="/opt/KUIN00601/etcd-client.key" snapshot save /var/lib/backup/etcd-snapshot.db

```
root@master01:~# ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --cacert="/opt/KUIN00601/ca.crt" --cert="/opt/KUIN00601/etcd
-client.crt" --key="/opt/KUIN00601/etcd-client.key" snapshot save /var/lib/backup/etcd-snapshot.db
{"level":"info","ts":"2024-02-17T00:35:16.586838+0800","caller":"snapshot/v3_snapshot.go:65","msg":"created temporary db file","path":
"/var/lib/backup/etcd-snapshot.db.part"}
{"level":"info","ts":"2024-02-17T00:35:16.606282+0800","logger":"client","caller":"v3@v3.5.10/maintenance.go:212","msg":"opened snapsh
ot stream; downloading"}
{"level":"info","ts":"2024-02-17T00:35:16.606449+0800","caller":"snapshot/v3_snapshot.go:73","msg":"fetching snapshot","endpoint":"htt
ps://127.0.0.1:2379"}
{"level":"info","ts":"2024-02-17T00:35:16.727881+0800","logger":"client","caller":"v3@v3.5.10/maintenance.go:220","msg":"completed sna
pshot read; closing"}
{"level":"info","ts":"2024-02-17T00:35:16.735824+0800","caller":"snapshot/v3_snapshot.go:88","msg":"fetched snapshot","endpoint":"http
s://127.0.0.1:2379","size":"8.0 MB","took":"now"}
{"level":"info","ts":"2024-02-17T00:35:16.735944+0800","caller":"snapshot/v3_snapshot.go:97","msg":"saved","path":"/var/lib/backup/etc
d-snapshot.db"}
Snapshot saved at /var/lib/backup/etcd-snapshot.db
root@master01:~#
```

检查：（考试时，这些检查动作，都可以不做）
ETCDCTL_API=3 etcdctl snapshot status /var/lib/backup/etcd-snapshot.db -wtable

```
root@master01:~# ETCDCTL_API=3 etcdctl snapshot status /var/lib/backup/etcd-snapshot.db -wtable
Deprecated: Use `etcdutl snapshot status` instead.

+----------+----------+------------+------------+
|   HASH   | REVISION | TOTAL KEYS | TOTAL SIZE |
+----------+----------+------------+------------+
| 41184e2f |   131416 |       1484 |     8.0 MB |
+----------+----------+------------+------------+
root@master01:~#
```

还原：

还原比较难，耗时也长，分值也低，风险还高。非老鸟，建议直接放弃还原这部分，应该只会扣 4 分的。

新建一个临时目录

mkdir -p /opt/bak/

将/etc/kubernetes/manifests/kube-*文件移动到临时目录里

mv /etc/kubernetes/manifests/kube-* /opt/bak/

```
root@master01:~# mkdir -p /opt/bak/
root@master01:~# mv /etc/kubernetes/manifests/kube-*  /opt/bak/
```

 # 还原可以不加证书

 # 需要指定 etcd 还原后的新数据目录，随便写一个系统里没有的目录，比如/var/lib/etcd-restore

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

```
root@master01:~# ETCDCTL_API=3 etcdctl --endpoints=https://127.0.0.1:2379 --data-dir=/var/lib/etcd-restore snapshot restore /data/back
up/etcd-snapshot-previous.db
Deprecated: Use `etcdutl snapshot restore` instead.

2024-02-17T00:58:54+08:00       info    snapshot/v3_snapshot.go:260     restoring snapshot      {"path": "/data/backup/etcd-snapshot-p
revious.db", "wal-dir": "/var/lib/etcd-restore/member/wal", "data-dir": "/var/lib/etcd-restore", "snap-dir": "/var/lib/etcd-restore/me
mber/snap"}
2024-02-17T00:58:54+08:00       info    membership/store.go:141 Trimming membership information from the backend...
2024-02-17T00:58:54+08:00       info    membership/cluster.go:421       added member    {"cluster-id": "cdf818194e3a8c32", "local-memb
er-id": "0", "added-peer-id": "8e9e05c52164694d", "added-peer-peer-urls": ["http://localhost:2380"]}
2024-02-17T00:58:54+08:00       info    snapshot/v3_snapshot.go:287     restored snapshot       {"path": "/data/backup/etcd-snapshot-p
revious.db", "wal-dir": "/var/lib/etcd-restore/member/wal", "data-dir": "/var/lib/etcd-restore", "snap-dir": "/var/lib/etcd-restore/me
mber/snap"}
root@master01:~#
```

检查一下新目录，有数据了

ls /var/lib/etcd-restore

```
root@master01:~# ls /var/lib/etcd-restore
member
```

修改 etcd 配置文件

vim /etc/kubernetes/manifests/etcd.yaml

```
root@master01:~# vim /etc/kubernetes/manifests/etcd.yaml
```

将 volume 下的 hostPath 配置里的 path: /var/lib/etcd 改成 path: /var/lib/etcd-restore

```
      periodSeconds: 10
      timeoutSeconds: 15
    volumeMounts:
    - mountPath: /var/lib/etcd
      name: etcd-data
    - mountPath: /etc/kubernetes/pki/etcd
      name: etcd-certs
  hostNetwork: true
  priority: 2000001000
  priorityClassName: system-node-critical
  securityContext:
    seccompProfile:
      type: RuntimeDefault
  volumes:
  - hostPath:
      path: /etc/kubernetes/pki/etcd
      type: DirectoryOrCreate
    name: etcd-certs
  - hostPath:
      path: /var/lib/etcd-restore
      type: DirectoryOrCreate
    name: etcd-data
status: {}
```

修改这里

将临时目录的文件，移动回/etc/kubernetes/manifests/目录里
mv /opt/bak/kube-* /etc/kubernetes/manifests/

```
root@master01:~# mv /opt/bak/kube-*  /etc/kubernetes/manifests/
```

重启 kubelet
systemctl daemon-reload
systemctl restart kubelet

```
root@master01:~# systemctl daemon-reload
root@master01:~# systemctl restart kubelet
```

再次检查集群状态，和现有的所有 pod。
可以看到很多 pod 没有了。这是因为我在做这个 etcd 备份做的时候，很多 pod 都还没有创建。
模拟环境里，有一些 pod 非 Running，也是因为我在做这个 etcd 备份时，整个集群还没有完全搭建好。

kubectl get nodes
kubectl get pod -A

```
root@master01:~# kubectl get nodes
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   30h   v1.29.1
node01     Ready    <none>          30h   v1.29.0
node02     Ready    <none>          30h   v1.29.0
root@master01:~#
root@master01:~# kubectl get pod -A
NAMESPACE          NAME                                       READY   STATUS             RESTARTS          AGE
calico-apiserver   calico-apiserver-556f6d894-7z9xn           0/1     CrashLoopBackOff   9 (4m36s ago)     29h
calico-apiserver   calico-apiserver-556f6d894-ncc4w           0/1     CrashLoopBackOff   9 (3m58s ago)     29h
calico-system      calico-kube-controllers-687c447f4b-75xfq   0/1     Running            8 (2m16s ago)     29h
calico-system      calico-node-8xgq7                          1/1     Running            3 (14h ago)       29h
calico-system      calico-node-d8wrr                          1/1     Running            2 (28h ago)       29h
calico-system      calico-node-mzz24                          1/1     Running            2 (28h ago)       29h
calico-system      calico-typha-6d6f5f8cbf-rt545              1/1     Running            2 (28h ago)       29h
calico-system      calico-typha-6d6f5f8cbf-zcjwz              1/1     Running            2 (28h ago)       29h
calico-system      csi-node-driver-29ckz                      2/2     Running            6 (14h ago)       29h
calico-system      csi-node-driver-jwghc                      2/2     Running            4 (28h ago)       29h
calico-system      csi-node-driver-r7xxz                      2/2     Running            4 (28h ago)       29h
kube-system        coredns-857d9ff4c9-7zmvq                   1/1     Running            2 (28h ago)       29h
kube-system        coredns-857d9ff4c9-pk8vm                   1/1     Running            2 (28h ago)       30h
kube-system        etcd-master01                              1/1     Running            0                 75s
kube-system        kube-apiserver-master01                    1/1     Running            0                 28s
kube-system        kube-controller-manager-master01           1/1     Running            0                 28s
kube-system        kube-proxy-v9vzq                           1/1     Running            2 (28h ago)       30h
kube-system        kube-proxy-vgkdl                           1/1     Running            0                 30h
kube-system        kube-proxy-x5qrq                           1/1     Running            2 (28h ago)       30h
kube-system        kube-scheduler-master01                    1/1     Running            0                 28s
tigera-operator    tigera-operator-55585899bf-7f4x5           0/1     Error              7 (12m ago)       29h
root@master01:~#
```

切记，做完后，要退回到初始节点
 # 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:~$
```

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

没必要参考官网，记住先 restart 再 enable 就行。

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

```
candidate@node01:~$ kubectl config use-context wk8s
error: no context exists with the name: "wk8s"
candidate@node01:~$
```

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 服务没有运行导致的，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
执行完 a.sh 脚本后，等 3 分钟，node02 才会挂掉。

sudo sh /opt/sh/a.sh

```
candidate@node01:~$ kubectl config use-context wk8s
error: no context exists with the name: "wk8s"
candidate@node01:~$
candidate@node01:~$ sudo sh /opt/sh/a.sh
Removed /etc/systemd/system/multi-user.target.wants/kubelet.service.
candidate@node01:~$
```

开始操作

检查一下 node，发现 node02 异常，执行完 a.sh 脚本后，等 3 分钟，node02 才会挂掉。
kubectl get nodes

```
candidate@node01:~$ kubectl get nodes
NAME       STATUS     ROLES           AGE   VERSION
master01   Ready      control-plane   31h   v1.29.1
node01     Ready      <none>          31h   v1.29.0
node02     NotReady   <none>          31h   v1.29.0
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，会发现服务没有运行，也没有加入开机启动。
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: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: inactive (dead)
       Docs: https://kubernetes.io/docs/

Feb 17 01:19:44 node02 kubelet[1335]: E0217 01:19:44.462580    1335 pod_workers.go:1298] "Error syncing pod, skipping" err="failed to>
Feb 17 01:19:44 node02 kubelet[1335]: E0217 01:19:44.810573    1335 server.go:310] "Unable to authenticate the request due to an erro>
Feb 17 01:19:45 node02 kubelet[1335]: W0217 01:19:45.866084    1335 reflector.go:539] object-"ingress-nginx"/"kube-root-ca.crt": fail>
Feb 17 01:19:45 node02 kubelet[1335]: E0217 01:19:45.866131    1335 reflector.go:147] object-"ingress-nginx"/"kube-root-ca.crt": Fail>
Feb 17 01:19:46 node02 kubelet[1335]: E0217 01:19:46.461013    1335 dns.go:153] "Nameserver limits exceeded" err="Nameserver limits w>
Feb 17 01:19:46 node02 kubelet[1335]: E0217 01:19:46.485638    1335 desired_state_of_world_populator.go:320] "Error processing volume>
Feb 17 01:19:48 node02 kubelet[1335]: E0217 01:19:48.462219    1335 dns.go:153] "Nameserver limits exceeded" err="Nameserver limits w>
Feb 17 01:19:49 node02 systemd[1]: Stopping kubelet: The Kubernetes Node Agent...
Feb 17 01:19:49 node02 systemd[1]: kubelet.service: Succeeded.
Feb 17 01:19:49 node02 systemd[1]: Stopped kubelet: The Kubernetes Node Agent.
lines 1-17/17 (END)
```

此处按 q 退出

运行 kubelet 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:~#
```

重新检查
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: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Sat 2024-02-17 01:25:06 CST; 23s ago
       Docs: https://kubernetes.io/docs/
   Main PID: 208636 (kubelet)
      Tasks: 15 (limit: 2530)
     Memory: 79.4M
     CGroup: /system.slice/kubelet.service
             └─208636 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kub>

Feb 17 01:25:12 node02 kubelet[208636]: I0217 01:25:12.965096  208636 kubelet_volumes.go:161] "Cleaned up orphaned pod volumes dir" p>
Feb 17 01:25:12 node02 kubelet[208636]: I0217 01:25:12.966138  208636 kubelet_volumes.go:161] "Cleaned up orphaned pod volumes dir" p>
Feb 17 01:25:14 node02 kubelet[208636]: E0217 01:25:14.152644  208636 dns.go:153] "Nameserver limits exceeded" err="Nameserver limits>
Feb 17 01:25:14 node02 kubelet[208636]: E0217 01:25:14.805401  208636 server.go:310] "Unable to authenticate the request due to an er>
Feb 17 01:25:15 node02 kubelet[208636]: E0217 01:25:15.090636  208636 dns.go:153] "Nameserver limits exceeded" err="Nameserver limits>
Feb 17 01:25:25 node02 kubelet[208636]: E0217 01:25:25.679271  208636 dns.go:153] "Nameserver limits exceeded" err="Nameserver limits>
Feb 17 01:25:25 node02 kubelet[208636]: I0217 01:25:25.914262  208636 prober_manager.go:312] "Failed to trigger a manual run" probe="">
Feb 17 01:25:26 node02 kubelet[208636]: E0217 01:25:26.129834  208636 dns.go:153] "Nameserver limits exceeded" err="Nameserver limits>
Feb 17 01:25:29 node02 kubelet[208636]: E0217 01:25:29.572093  208636 dns.go:153] "Nameserver limits exceeded" err="Nameserver limits>
Feb 17 01:25:29 node02 kubelet[208636]: E0217 01:25:29.795935  208636 server.go:310] "Unable to authenticate the request due to an er>
lines 1-22/22 (END)
```

切记，做完后，要退回到初始节点
 # 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:~$
```

再次检查节点状态， 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   31h   v1.29.1
node01     Ready    <none>          31h   v1.29.0
node02     Ready    <none>          31h   v1.29.0
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

强烈建议背过操作命令，如果非要参考，可以使用-h 帮助。
kubectl -h

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

```
candidate@node01:~$ kubectl config use-context ek8s
error: no context exists with the name: "ek8s"
candidate@node01:~$
```

开始操作

先检查节点状态
kubectl get node

```
candidate@node01:~$ kubectl config use-context ek8s
error: no context exists with the name: "ek8s"
candidate@node01:~$
candidate@node01:~$ kubectl get node
NAME       STATUS   ROLES           AGE   VERSION
master01   Ready    control-plane   31h   v1.29.1
node01     Ready    <none>          31h   v1.29.0
node02     Ready    <none>          31h   v1.29.0
candidate@node01:~$
```

暂停节点 node02 调度
kubectl cordon node02

再次检查节点状态
kubectl get node

```
candidate@node01:~$ kubectl cordon node02
node/node02 cordoned
candidate@node01:~$
candidate@node01:~$ kubectl get node
NAME       STATUS                     ROLES           AGE   VERSION
master01   Ready                      control-plane   31h   v1.29.1
node01     Ready                      <none>          31h   v1.29.0
node02     Ready,SchedulingDisabled   <none>          31h   v1.29.0
candidate@node01:~$
```

驱逐节点 node02（因为 daemonsets 是无法驱逐的，所以要忽略它）
kubectl drain node02 --ignore-daemonsets

```
candidate@node01:~$ kubectl drain node02 --ignore-daemonsets
node/node02 already cordoned
Warning: ignoring DaemonSet-managed Pods: calico-system/calico-node-d8wrr, calico-system/csi-node-driver-r7xxz, kube-system/kube-proxy
-v9vzq
evicting pod kube-system/coredns-857d9ff4c9-7zmvq
evicting pod calico-apiserver/calico-apiserver-556f6d894-ncc4w
evicting pod calico-system/calico-typha-6d6f5f8cbf-zcjwz
pod/calico-apiserver-556f6d894-ncc4w evicted
pod/coredns-857d9ff4c9-7zmvq evicted
pod/calico-typha-6d6f5f8cbf-zcjwz evicted
node/node02 drained
candidate@node01:~$
```

注意，如果执行上面驱逐命令，有下图这样的报错，无法驱逐，则需要加上 --delete-emptydir-data --force 参数
 # 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:~$
```

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

2024 年 04 月 21 号

1、优化部分题目的参考截图

2024 年 02 月 17 号

1、升级模拟环境 K8S 版本为 1.29.0
2、etcd 备份恢复，改为在 master01 上操作，并修改操作命令。
3、升级集群那道题添加注意事项，通过 apt-cache show kubeadm|grep 1.29.1 命令，可以查出来具体要升级的版本号，包括后面的版本号数字。比如 1.28.1-00 1.29.1-1.1 等等。这样后面的命令才知道具体升级的版本号数字。

2023 年 10 月 5 号

1、etcd 这道题添加注意事项

2023 年 8 月 27 号

1、升级为 K8S v1.28 版本

2023 年 7 月 2 号

1、优化 ingress

2023 年 6 月 17 号

1、13 题增加　　imagePullPolicy: IfNotPresent

2023 年 5 月 20 号

1、优化第 7、8 题答案

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。