一、注意 必读　　CKAD　1.32

真正考试时，并不是在 root 下，而是在 candidate 普通账号下，在 base 机器上 ssh 不同的机器后做题的。
所以模拟环境也是用 candidate 账号，在 base 机器上，ssh 不同机器做题，与考试方法一致。

主机名 base（node01）
IP 地址 11.0.1.112
帐号为 candidate
密码为 123

在此账号下，可以免密 ssh 登录 master01 和 node02（ ssh master01 或 ssh node02 ）
3 台虚拟机的 candidate 帐号也可以免密切换到 root（ sudo -i ）

考前务必问问老师，最新考试注意事项!!!

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。

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。尽量熟练到 70%的题不参考 K8S 官网，另外在模拟环境里，熟练到一个半小时内可以做完全部题目，再去约考，成功的把握大。

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，可以用 Ubuntu 自带的火狐浏览器到 K8S 官网查资料。
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。但是如果屏幕太小，可能显示不全，导致看不到网页右边的语言按钮。所以建议平常练习，就使用英文网页。

每次练习时，还原初始化快照后，开机后**等 5 分钟**，再 ssh 登录 base（11.0.1.112）检查一下所有的 pod，确保都 Running 了，再开始练习。（CKAD 题目里 2 个不是 Running 的，忽略）。另外，考试环境和模拟环境的 kubectl 命令，都有自动补全的。


```
candidate@base:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE    VERSION
base       Ready    <none>          7d8h   v1.32.1
master01   Ready    control-plane   7d8h   v1.32.1
node02     Ready    <none>          7d8h   v1.32.1
candidate@base:~$
candidate@base:~$ kubectl get pod -A
NAMESPACE          NAME                                       READY   STATUS      RESTARTS       AGE
calico-apiserver   calico-apiserver-56759c88bb-4dr4z          1/1     Running     2 (48m ago)    7d8h
calico-apiserver   calico-apiserver-56759c88bb-mfqfp          1/1     Running     2 (48m ago)    7d8h
calico-system      calico-kube-controllers-8b49698f4-8w6vh    1/1     Running     2 (48m ago)    7d8h
calico-system      calico-node-brm2q                          1/1     Running     2 (48m ago)    7d8h
calico-system      calico-node-fbs28                          1/1     Running     2 (48m ago)    7d8h
calico-system      calico-node-qsnj4                          1/1     Running     2 (48m ago)    7d8h
calico-system      calico-typha-68f956d696-24fxj              1/1     Running     2 (48m ago)    7d8h
calico-system      calico-typha-68f956d696-mf5cf              1/1     Running     2 (48m ago)    7d8h
calico-system      csi-node-driver-8nb8t                      2/2     Running     8 (48m ago)    7d8h
calico-system      csi-node-driver-j99sw                      2/2     Running     8 (48m ago)    7d8h
calico-system      csi-node-driver-kfcwc                      2/2     Running     8 (48m ago)    7d8h
ckad00015          webapp-754dbdbc58-9bkg6                    1/1     Running     1 (48m ago)    4d21h
ckad00017          ckad00017-deployment-98fbcf678-bc6zl       1/1     Running     1 (48m ago)    4d21h
ckad00018          ckad00018-dmz                              1/1     Running     1 (48m ago)    4d21h
ckad00018          ckad00018-newpod                           1/1     Running     1 (48m ago)    4d21h
ckad00018          db                                         1/1     Running     1 (48m ago)    4d21h
ckad00018          front                                      1/1     Running     1 (48m ago)    4d21h
frontend           frontend-deployment-797bd4bc59-ff7ws       1/1     Running     1 (48m ago)    4d21h
gorilla            honeybee-deployment-58f59d4bf9-z6zkm       1/1     Running     1 (48m ago)    4d21h
goshawk            current-chipmunk-deployment-7ffcbb48f-6b2ph  1/1   Running     1 (48m ago)    4d21h
goshawk            current-chipmunk-deployment-7ffcbb48f-bl9t8  1/1   Running     1 (48m ago)    4d21h
goshawk            current-chipmunk-deployment-7ffcbb48f-glcfp  1/1   Running     1 (48m ago)    4d21h
goshawk            current-chipmunk-deployment-7ffcbb48f-pvnhv  1/1   Running     1 (48m ago)    4d21h
goshawk            current-chipmunk-deployment-7ffcbb48f-vvfbq  1/1   Running     1 (48m ago)    4d21h
haddock            nosql-7b9cd75b9d-5xzzb                     1/1     Running     1 (48m ago)    4d21h
ingress-ckad       nginx-dm-5587f8dbcd-dr7ms                  1/1     Running     1 (48m ago)    4d21h
ingress-ckad       nginx-dm-5587f8dbcd-xmzv4                  1/1     Running     1 (48m ago)    4d21h
ingress-nginx      ingress-nginx-admission-create-m4zmf       0/1     Completed   0              4d21h
ingress-nginx      ingress-nginx-admission-patch-7fwhr        0/1     Completed   0              4d21h
ingress-nginx      ingress-nginx-controller-r7cfg             1/1     Running     1 (48m ago)    4d21h
ingress-nginx      ingress-nginx-controller-sdt9z             1/1     Running     1 (48m ago)    4d21h
kube-system        coredns-6766b7b6bb-26cfm                   1/1     Running     2 (48m ago)    7d8h
kube-system        coredns-6766b7b6bb-m9nrp                   1/1     Running     2 (48m ago)    7d8h
kube-system        etcd-master01                              1/1     Running     2 (48m ago)    7d8h
kube-system        kube-apiserver-master01                    1/1     Running     2 (48m ago)    7d8h
kube-system        kube-controller-manager-master01           1/1     Running     2 (48m ago)    7d8h
kube-system        kube-proxy-2cgdg                           1/1     Running     2 (48m ago)    7d8h
kube-system        kube-proxy-62gnq                           1/1     Running     2 (48m ago)    7d8h
kube-system        kube-proxy-sfzz8                           1/1     Running     2 (48m ago)    7d8h
kube-system        kube-scheduler-master01                    1/1     Running     2 (48m ago)    7d8h
kube-system        metrics-server-8467fcc7b7-6mrlp            1/1     Running     0              3m18s
prod27             probe-http-6747cc4dc5-jsmk9                1/1     Running     1 (48m ago)    4d21h
quetzal            broker-deployment-5cffb94468-8n4zf         1/1     Running     1 (48m ago)    4d21h
tigera-operator    tigera-operator-ccfc44587-scwf5            1/1     Running     2 (48m ago)    7d8h
candidate@base:~$
```

K8S 1.32 最新版本
与考试考题一致
做题需要的Pod

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（这些参数都是蓝色字体）。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。

二、鸣谢

特别鸣谢以下网友（排名不分先后）：

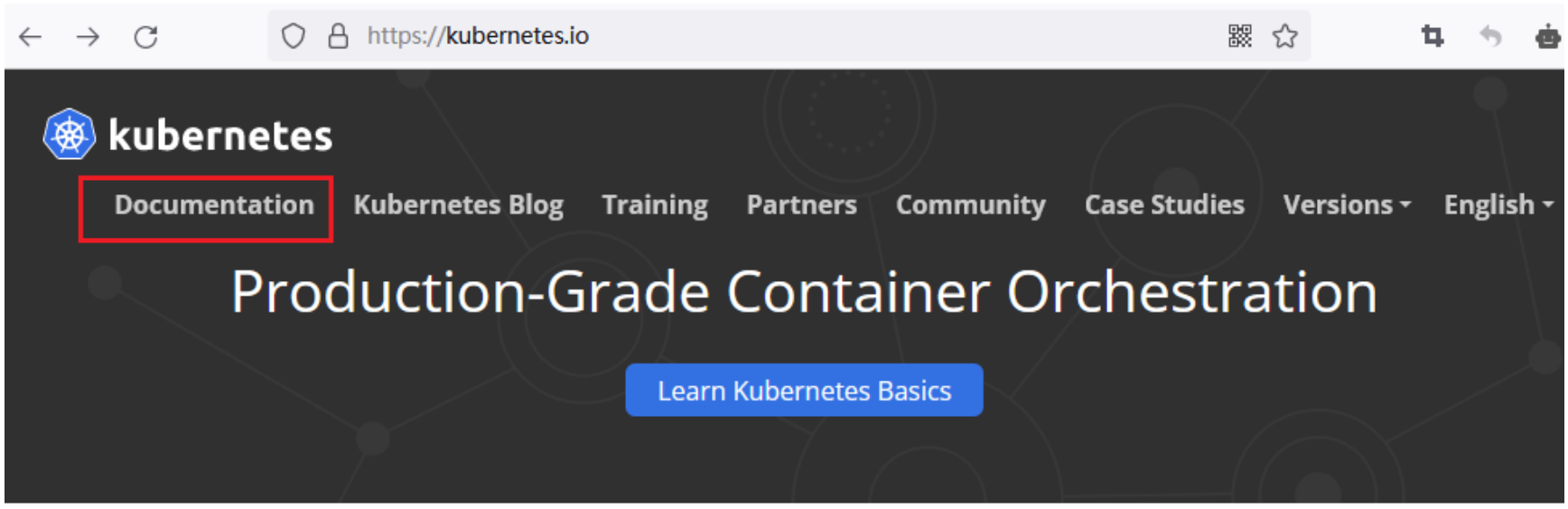
勋（lizhanxun4*）
唐 Jing（sinot*）
gmwinsto* [台]

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。

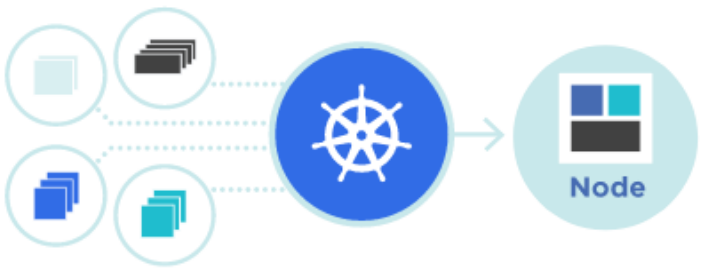
三、如何一步步找到找官网资料

考试时，也这样查官网
<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”

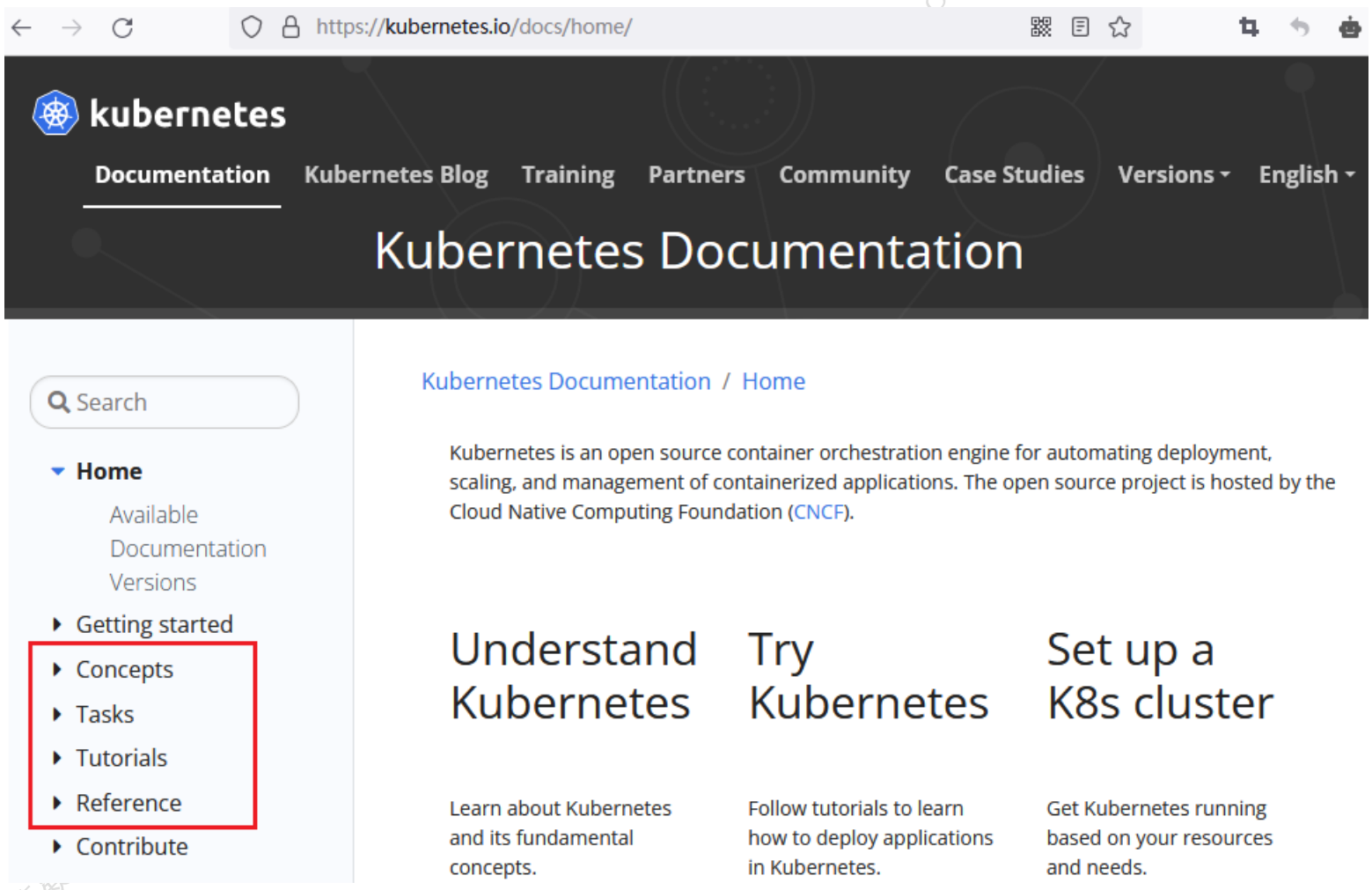


Kubernetes, also known as K8s, is an open-source system for automating deployment, scaling, and management of containerized applications.



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。

四、yaml 文件格式说明

懂的就不用看了，解释给小白听的 ^_^

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

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

```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。

```yaml
    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。

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。

1、CronJob

（考试的考题内容，只有下面方框里的内容）

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00021

Task
1、创建一个名为 ppi 并执行一个运行以下单一容器的 Pod 的 CronJob：
- name: pi
  image: perl:5
  command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]

配置 CronJob 为：

- 每隔 5 分钟执行一次
- 保留 2 个已完成的 Job
- 保留 4 个失败的 Job
- 永不重启 Pod
- 在 8 秒后终止 Pod

2、为测试目的，从 CronJob ppi 中手动创建并执行一个名为 ppi-test 的 Job。

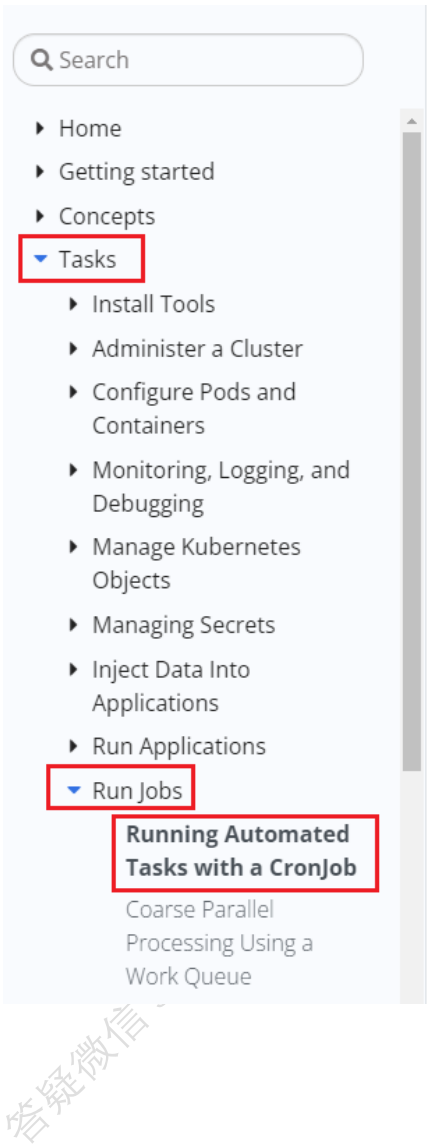
Job的完成或失败并不重要。

**参考地址

创建 CronJob

（需要复制网页内容）

依次点击下图，打开最终网址。（看不懂英文的，可右上角翻译成中文）
请注意：不要直接打开下面网址，而是要学会如何自己在官网查。
https://kubernetes.io/zh-cn/docs/tasks/job/automated-tasks-with-cron-jobs/
https://kubernetes.io/zh-cn/docs/concepts/workloads/controllers/job/
https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/



Kubernetes Documentation / Tasks / Run Jobs / Running Automated Tasks with a CronJob

Running Automated Tasks with a CronJob

CronJobs was promoted to general availability in Kubernetes v1.21. If you are using an older version of Kubernetes, please refer to the documentation for the version of Kubernetes that you are using, so that you see accurate information. Older Kubernetes versions do not support the `batch/v1` CronJob API.

You can use a CronJob to run Jobs on a time-based schedule. These automated jobs run like Cron tasks on a Linux or UNIX system.

Cron jobs are useful for creating periodic and recurring tasks, like running backups or sending emails. Cron jobs can also schedule individual tasks for a specific time, such as if you want to schedule a job for a low activity period.

Cron jobs have limitations and idiosyncrasies. For example, in certain circumstances, a single cron job can create multiple jobs. Therefore, jobs should be idempotent.

For more limitations, see CronJobs.

Before you begin

- You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using minikube or you can use one of these Kubernetes playgrounds:
  - Killercoda
  - Play with Kubernetes

创建 CronJob

CronJob 需要一个配置文件。 本例中 CronJob 的 `.spec` 配置文件每分钟打印出当前时间和一个问好信息：

application/job

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox:1.28
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

command这块内容
根据题目要求写

CronJob 的 spce 解释

时间安排

`.spec.schedule` 是 `.spec` 中的必需字段。它接受 Cron 格式串，例如 `0 * * * *` or `@hourly`，作为它的任务被创建和执行的调度时间。

该格式也包含了扩展的 “Vixie cron” 步长值。 FreeBSD 手册中解释如下:

> 步长可被用于范围组合。范围后面带有 `/<数字>` 可以声明范围内的步幅数值。 例如， `0-23/2` 可被用在小时字段来声明命令在其他数值的小时数执行 （V7 标准中对应的方法是 `0,2,4,6,8,10,12,14,16,18,20,22`）。 步长也可以放在通配符后面，因此如果你想表达 “每两小时”，就用 `*/2` 。

Jobs

A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate. As pods successfully complete, the Job tracks the successful completions. When a specified number of successful completions is reached, the task (ie, Job) is complete. Deleting a Job will clean up the Pods it created. Suspending a Job will delete its active Pods until the Job is resumed again.

A simple case is to create one Job object in order to reliably run one Pod to completion. The Job object will start a new Pod if the first Pod fails or is deleted (for example due to a node hardware failure or a node reboot).

You can also use a Job to run multiple Pods in parallel.

If you want to run a Job (either a single task, or several in parallel) on a schedule, see CronJob.

Running an example Job

Here is an example Job config. It computes π to 2000 places and prints it out. It takes around 10s to complete.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
```

终止 Job 的另一种方式是设置一个活跃期限。 你可以为 Job 的 `.spec.activeDeadlineSeconds` 设置一个秒数值。 该值适用于 Job 的整个生命期，无论 Job 创建了多少个 Pod。 一旦 Job 运行时间达到 `activeDeadlineSeconds` 秒，其所有运行中的 Pod 都会被终止， 并且 Job 的状态更新为 `type: Failed` 及 `reason: DeadlineExceeded`。

注意 Job 的 `.spec.activeDeadlineSeconds` 优先级高于其 `.spec.backoffLimit` 设置。 因此，如果一个 Job 正在重试一个或多个失效的 Pod，该 Job 一旦到达 `activeDeadlineSeconds` 所设的时限即不再部署额外的 Pod， 即使其重试次数还未达到 `backoffLimit` 所设的限制。

例如：

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-with-timeout
spec:
  backoffLimit: 5
  activeDeadlineSeconds: 100
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```

任务历史限制

`.spec.successfulJobsHistoryLimit` 和 `.spec.failedJobsHistoryLimit` 是可选的。 这两个字段指定应保留多少已完成和失败的任务。 默认设置分别为 3 和 1。设置为 `0` 代表相应类型的任务完成后不会保留。

答题

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。

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令

ssh ckad00021

```
candidate@base:~$ ssh ckad00021
candidate@master01:~$
```

【2】 创建 CronJob

vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入

vi cronjob-1.yaml

#根据官网参考链接，复制 yaml 内容，并修改:

注意 command: 前面的空格对齐!!!

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: ppi
spec:
  schedule: "*/5 * * * *"     #CronJob 的运行时间表，根据题目要求修改。这里是每 5 分钟!
```

```
successfulJobsHistoryLimit: 2
failedJobsHistoryLimit: 4
jobTemplate:
   spec:
      activeDeadlineSeconds: 8       #感谢网友一如即往提供的测试反馈
      template:
         spec:
            containers:
            - name: pi
              image: perl:5
              imagePullPolicy: IfNotPresent
              command: ["perl",    "-Mbignum=bpi", "-wle", "print bpi(2000)"]
            restartPolicy: Never       #这里记得修改，永不重启 Pod 为 Never
```

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: ppi
spec:
  schedule: "*/5 * * * *"
  successfulJobsHistoryLimit: 2
  failedJobsHistoryLimit: 4
  jobTemplate:
    spec:
      activeDeadlineSeconds: 8
      template:
        spec:
          containers:
          - name: pi
            image: perl:5
            imagePullPolicy: IfNotPresent
            command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
          restartPolicy: Never
```

创建 CronJob

kubectl apply -f cronjob-1.yaml

```
candidate@master01:~$ vi cronjob-1.yaml
candidate@master01:~$
candidate@master01:~$ kubectl apply -f cronjob-1.yaml
cronjob.batch/ppi created
candidate@master01:~$
```

检查 CronJob

kubectl get cronjob

```
candidate@master01:~$ kubectl get cronjob
NAME   SCHEDULE      TIMEZONE   SUSPEND   ACTIVE   LAST SCHEDULE   AGE
ppi    */5 * * * *   <none>     False     0        <none>          15s
candidate@master01:~$
```

【3】 手动触发一个 cronjob

kubectl create job ppi-test --from=cronjob/ppi

```
candidate@master01:~$ kubectl create job ppi-test --from=cronjob/ppi
job.batch/ppi-test created
candidate@master01:~$
```

查看这个 job

kubectl get jobs

> Job的完成或失败并不重要。

```
candidate@master01:~$ kubectl get jobs
NAME       STATUS     COMPLETIONS   DURATION   AGE
ppi-test   Complete   1/1           6s         18s
candidate@master01:~$
```

【4】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00021 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00021 closed.
candidate@base:~$
```

2、Dockerfile

（考试的考题内容，只有下面方框里的内容）

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00033

Task
一个 Dockerfile 已经存在于 /ckad/DF/Dockerfile

1、使用已存在的 Dockerfile，构建一个名为 centos 和标签为 8.2 的容器镜像。
您可以安装和使用您选择的工具。

原本的系统中预装了多个符合 OCI 标准的镜像构建器和工具，包括：docker 、skopeo 、 buildah 、img 和 podman 。

请不要将构建的镜像推送到镜像仓库、运行容器或以其他方式使用它。

2、使用您选择的工具，以 OCI 格式导出构建的容器镜像，并将其存储在 /ckad/DF/centos-8.2.tar

参考地址

背过。如果考试时忘记了，就使用 docker -h 帮助。

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点
考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

```
确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00033
```

```
candidate@base:~$ ssh ckad00033
```

```
candidate@master01:~$
```

【2】 构建镜像
要切换到 Dockerfile 文件所在的目录做，注意命令最后还有一个小数点，表示当前目录。

```
cd /ckad/DF/
sudo docker build -t centos:8.2 .
```

```
candidate@master01:~$ cd /ckad/DF/
candidate@master01:/ckad/DF$ sudo docker build -t centos:8.2 .
[+] Building 1.1s (7/7) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 233B
 => [internal] load metadata for docker.io/library/centos:8
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/library/centos:8
 => [2/3] RUN useradd shadow
 => [3/3] RUN mkdir /opt/shadow
 => exporting to image
 => => exporting layers
 => => writing image sha256:be1fd83083c69394a020c3d405aceb0e00ee10632f0fa0a6247012da322002bc
 => => naming to docker.io/library/centos:8.2
candidate@master01:/ckad/DF$
```

检查镜像

sudo docker images | grep centos | grep 8.2

```
candidate@master01:/ckad/DF$ sudo docker images | grep centos | grep 8.2
centos       8.2       be1fd83083c6   21 seconds ago   232MB
candidate@master01:/ckad/DF$
```

【3】 导出新镜像 centos:8.2，并保存到/ckad/DF/centos-8.2.tar

sudo docker save centos:8.2 > /ckad/DF/centos-8.2.tar

```
candidate@master01:/ckad/DF$ sudo docker save centos:8.2 > /ckad/DF/centos-8.2.tar
candidate@master01:/ckad/DF$
```

检查存储的镜像

ll /ckad/DF/centos-8.2.tar

```
candidate@master01:/ckad/DF$ ll /ckad/DF/centos-8.2.tar
-rw-rw-r--+ 1 candidate candidate 238904832 Feb 16 12:39 /ckad/DF/centos-8.2.tar
candidate@master01:/ckad/DF$
```

【4】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00033 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:/ckad/DF$ exit
logout
Connection to ckad00033 closed.
candidate@base:~$
```

3、限制 CPU 和内存 request

（考试的考题内容，只有下面方框里的内容）

Context

您需要创建一个请求一定量的 CPU 和内存的 Pod。

您必须连接到正确的主机。不这样做可能导致零分。

[candidate@base] $ ssh ckad00003

Task

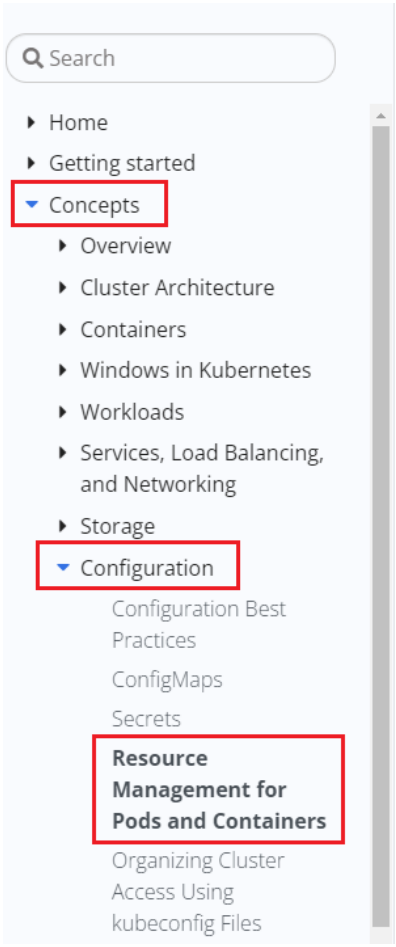
在现有的 namespace pod-resources 中创建一个名为 nginx-resources 的 Pod。用 nginx:1.16 的镜像来指定一个容器.

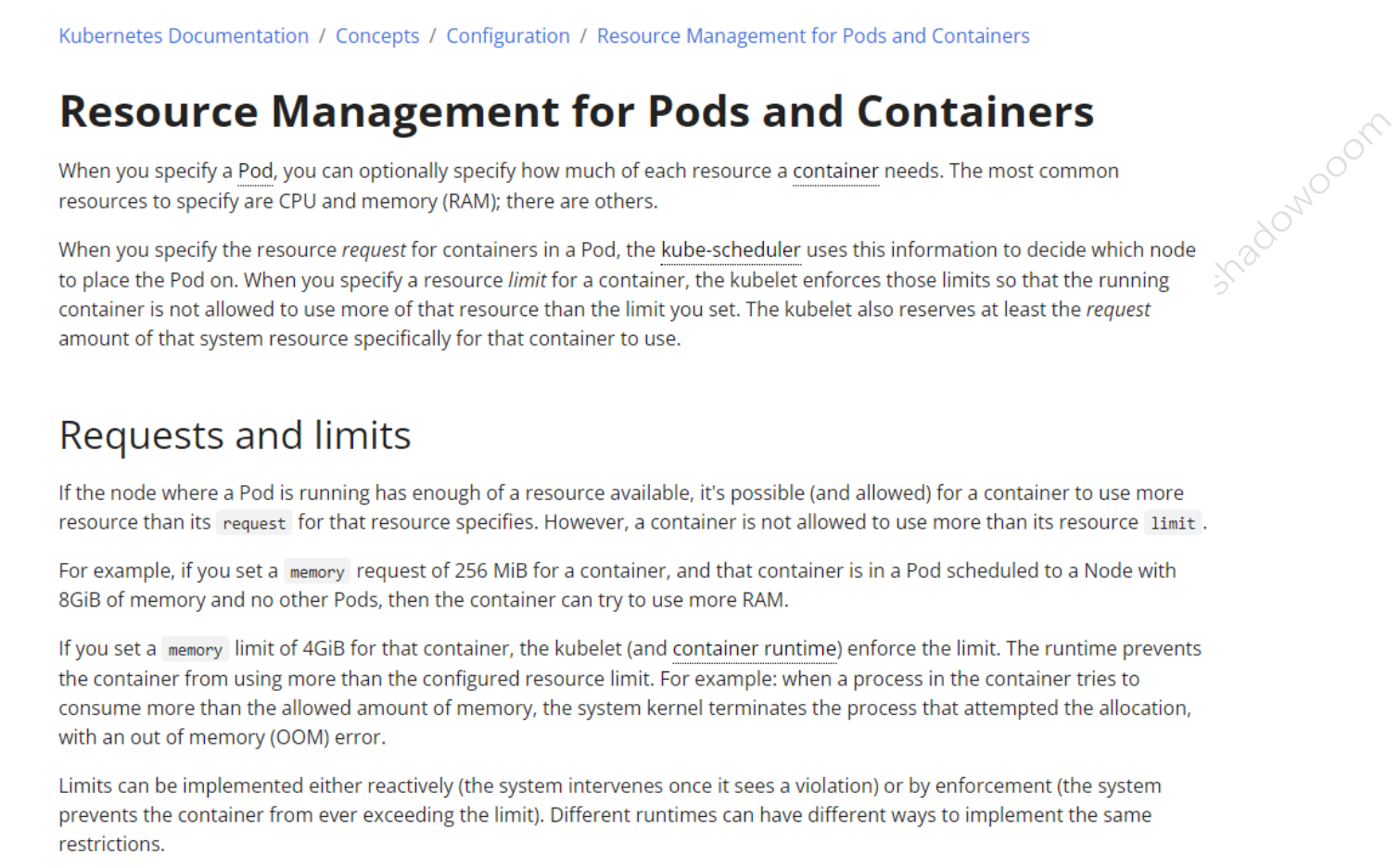
为其容器指定 40m 的 CPU 和 50Mi 的内存的资源请求.

**参考地址

（需要复制网页内容）

依次点击下图，打开最终网址。（看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/concepts/configuration/manage-resources-containers/





Kubernetes Documentation / Concepts / Configuration / Resource Management for Pods and Containers

Resource Management for Pods and Containers

When you specify a Pod, you can optionally specify how much of each resource a container needs. The most common resources to specify are CPU and memory (RAM); there are others.

When you specify the resource *request* for containers in a Pod, the kube-scheduler uses this information to decide which node to place the Pod on. When you specify a resource *limit* for a container, the kubelet enforces those limits so that the running container is not allowed to use more of that resource than the limit you set. The kubelet also reserves at least the *request* amount of that system resource specifically for that container to use.

Requests and limits

If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its `request` for that resource specifies. However, a container is not allowed to use more than its resource `limit`.

For example, if you set a `memory` request of 256 MiB for a container, and that container is in a Pod scheduled to a Node with 8GiB of memory and no other Pods, then the container can try to use more RAM.

If you set a `memory` limit of 4GiB for that container, the kubelet (and container runtime) enforce the limit. The runtime prevents the container from using more than the configured resource limit. For example: when a process in the container tries to consume more than the allowed amount of memory, the system kernel terminates the process that attempted the allocation, with an out of memory (OOM) error.

Limits can be implemented either reactively (the system intervenes once it sees a violation) or by enforcement (the system prevents the container from ever exceeding the limit). Different runtimes can have different ways to implement the same restrictions.

容器资源示例

以下 Pod 有两个容器。每个容器的请求为 0.25 CPU 和 64MiB（2^{26} 字节）内存， 每个容器的资源约束为 0.5 CPU 和 128MiB 内存。 你可以认为该 Pod 的资源请求为 0.5 CPU 和 128 MiB 内存，资源限制为 1 CPU 和 256MiB 内存。

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00003

```
candidate@base:~$ ssh ckad00003

candidate@master01:~$
```

【2】 编辑 yaml 文件

vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入

vi nginx-resources.yaml

```
#添加如下内容
apiVersion: v1
kind: Pod
metadata:
  name: nginx-resources
  namespace: pod-resources
spec:
  containers:
  - image: nginx:1.16
    name: nginx-resources
    resources:
      requests:
        cpu: "40m"
        memory: "50Mi"
```

创建

kubectl apply -f nginx-resources.yaml

```
candidate@master01:~$ vi nginx-resources.yaml
candidate@master01:~$
candidate@master01:~$ kubectl apply -f nginx-resources.yaml
pod/nginx-resources created
candidate@master01:~$
```

【3】 检查

kubectl -n pod-resources get pod

```
candidate@master01:~$ kubectl -n pod-resources get pod
NAME              READY   STATUS    RESTARTS   AGE
nginx-resources   1/1     Running   0          35s
candidate@master01:~$
```

【4】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

退出 ckad00003 节点，返回到 candidate@base 用户和节点下
exit

```
candidate@master01:~$ exit
logout
Connection to ckad00003 closed.
candidate@base:~$
```

4、限制内存 request 和 limit

（考试的考题内容，只有下面方框里的内容）

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00032

Task

haddock namespace 中名为 nosql 的 Deployment 的 Pod 因其容器已用完资源而无法启动。
请更新 nosql Deployment，使 Pod：

- 为其容器请求 15Mi 的内存

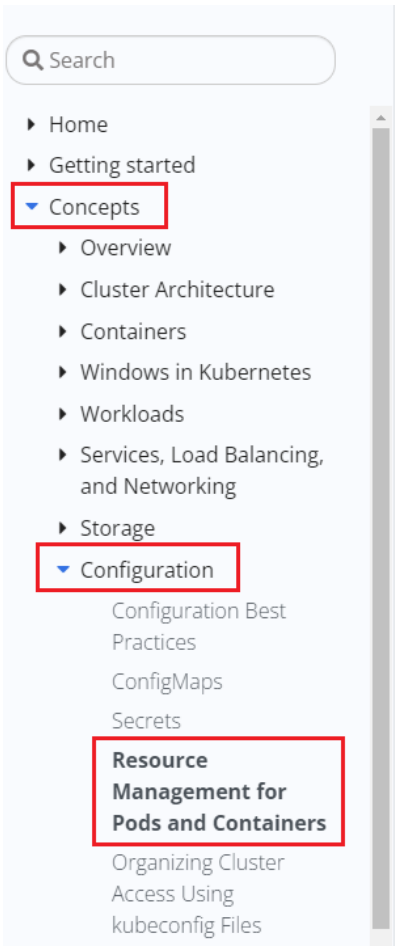
- 将内存限制为 haddock namespace 设置的最大内存容量的一半。

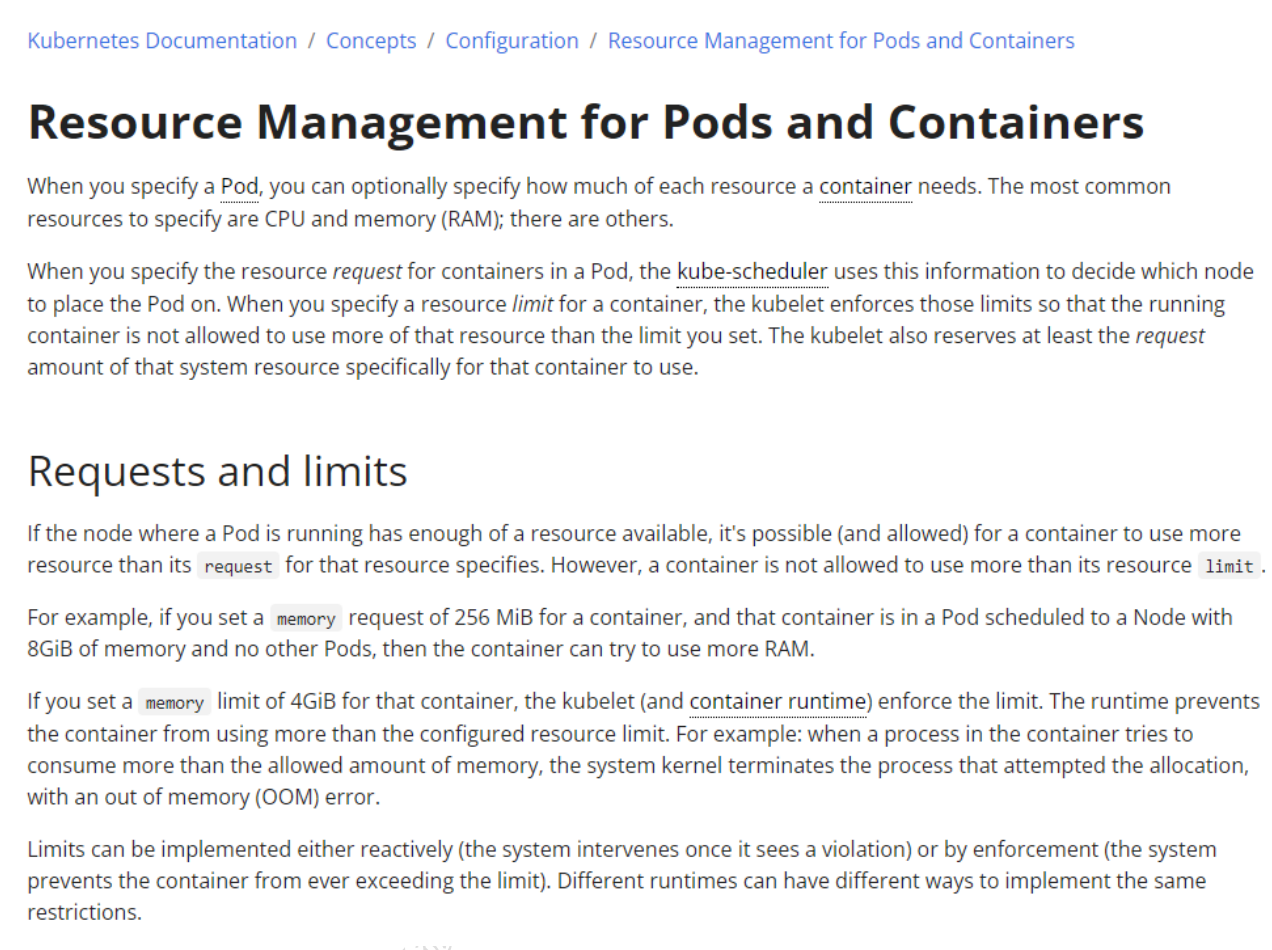
您可以在 /ckad/chief-cardinal/nosql.yaml 找到 nosql Deployment 的配置清单。

**参考地址

（需要复制网页内容）

依次点击下图，打开最终网址。（看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/concepts/configuration/manage-resources-containers/





容器资源示例

以下 Pod 有两个容器。每个容器的请求为 0.25 CPU 和 64MiB（2^{26} 字节）内存， 每个容器的资源约束为 0.5 CPU 和 128MiB 内存。 你可以认为该 Pod 的资源请求为 0.5 CPU 和 128 MiB 内存，资源限制为 1 CPU 和 256MiB 内存。

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00032

```
candidate@base:~$ ssh ckad00032
candidate@master01:~$
```

【2】 先查看 namespace haddock 的 LimitRange 详情

kubectl describe ns haddock

可见 namespace haddock 设置的最大内存为 40Mi，一半就是 20Mi。（注意，是看 Max，不是看 Min。Min 是最小的意思。）
感谢网友 wlc924991*的反馈

```
candidate@master01:~$ kubectl describe ns haddock
Name:         haddock
Labels:       kubernetes.io/metadata.name=haddock
Annotations:  <none>
Status:       Active

No resource quota.

Resource Limits
 Type       Resource  Min  Max   Default Request  Default Limit  Max Limit/Request Ratio
 ----       --------  ---  ---   ---------------  -------------  -----------------------
 Container  memory    -    40Mi  8Mi              16Mi           -
candidate@master01:~$
```

【3】 修改 CPU 和 memory

方法 1：直接在线修改

vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入
kubectl -n haddock edit deployments.apps nosql

注意，在 containers: image 这一小段最后添加。
添加如下内容

```
      spec:
        containers:
        - image: nginx:1.16
          name: nginx
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File      #在 containers: image 这一段最后添加
          resources:     # 删除 resources 后面的 {} ， 然后在下面新增 4 行。
            requests:
              memory: "15Mi"
            limits:
              memory: "20Mi"
```

```
    spec:
      containers:
      - image: nginx:1.16
        imagePullPolicy: IfNotPresent
        name: nginx
        resources:
          requests:
            memory: "15Mi"
          limits:
            memory: "20Mi"
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
```

提示 edited，表示修改成功

```
candidate@master01:~$ kubectl -n haddock edit deployments.apps nosql
deployment.apps/nosql edited
candidate@master01:~$
```

检查一下，确保修改成功了。

kubectl -n haddock describe deployment nosql

```
candidate@master01:~$ kubectl -n haddock describe deployment nosql
Name:                   nosql
Namespace:              haddock
CreationTimestamp:      Sat, 15 Feb 2025 19:41:20 +0800
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 2
Selector:               app=nosql
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nosql
  Containers:
   nginx:
    Image:      nginx:1.16
    Port:       <none>
    Host Port:  <none>
    Limits:
      memory:  20Mi
    Requests:
      memory:      15Mi
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  nosql-5b4984d8c8 (0/0 replicas created)
NewReplicaSet:   nosql-757d774775 (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  22s   deployment-controller  Scaled up replica set nosql-757d774775 to 1
  Normal  ScalingReplicaSet  20s   deployment-controller  Scaled down replica set nosql-5b4984d8c8 to 0 from 1
candidate@master01:~$
```

方法 2：

还有一个用命令修改的方法，比如

kubectl -n xxxx set resources deployment yyyy --limits=cpu=200m,memory=512Mi --requests=100m,memory=256Mi

这道题则为

kubectl -n haddock set resources deployment nosql --limits=memory=20Mi --requests=memory=15Mi

```
candidate@master01:~$ kubectl -n haddock set resources deployment nosql --limits=memory=20Mi --requests=memory=15Mi
deployment.apps/nosql resource requirements updated
candidate@master01:~$
```

【4】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00032 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00032 closed.
candidate@base:~$
```

5、运行旧版应用程序

（考试的考题内容，只有下面方框里的内容）

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00026

Context
您需要在运行最新版本 Kubernetes 的 cluster 上部署一个为旧版本 Kubernetes 开发的应用程序。

Task
1、修复清单文件 /ckad/credible-mite/www.yaml 中的任何 API 弃用问题，以便可以将应用程序部署在 k8s 集群上。
注意：该应用程序是为 Kubernetes v1.15 开发的。
K8S 集群运行着 Kubernetes v1.31
2、请在 garfish namespace 中部署更新后的清单文件/ckad/credible-mite/www.yaml 中指定的应用程序。

参考地址

建议背过命令，如果忘记了，可以使用如下命令，定位到需要的内容。

kubectl explain deployment.spec

```
student@node01:~$ kubectl explain deployment.spec
KIND:     Deployment
VERSION:  apps/v1

RESOURCE: spec <Object>

DESCRIPTION:
     Specification of the desired behavior of the Deployment.

     DeploymentSpec is the specification of the desired behavior of the
     Deployment.

FIELDS:
   minReadySeconds      <integer>
     Minimum number of seconds for which a newly created pod should be ready
     without any of its container crashing, for it to be considered available.
     Defaults to 0 (pod will be considered available as soon as it is ready)

   paused       <boolean>
     Indicates that the deployment is paused.

   progressDeadlineSeconds      <integer>
     The maximum time in seconds for a deployment to make progress before it is
     considered to be failed. The deployment controller will continue to process
     failed deployments and a condition with a ProgressDeadlineExceeded reason
     will be surfaced in the deployment status. Note that progress will not be
     estimated during the time a deployment is paused. Defaults to 600s.

   replicas     <integer>
     Number of desired pods. This is a pointer to distinguish between explicit
     zero and not specified. Defaults to 1.

   revisionHistoryLimit <integer>
     The number of old ReplicaSets to retain to allow rollback. This is a
     pointer to distinguish between explicit zero and not specified. Defaults to
     10.

   selector     <Object> -required-
     Label selector for pods. Existing ReplicaSets whose pods are selected by
     this will be the ones affected by this deployment. It must match the pod
     template's labels.

   strategy     <Object>
     The deployment strategy to use to replace existing pods with new ones.

   template     <Object> -required-
     Template describes the pods that will be created.
```

kubectl explain deployment.spec.selector

```
student@node01:~$ kubectl explain deployment.spec.selector
KIND:     Deployment
VERSION:  apps/v1

RESOURCE: selector <Object>

DESCRIPTION:
     Label selector for pods. Existing ReplicaSets whose pods are selected by
     this will be the ones affected by this deployment. It must match the pod
     template's labels.

     A label selector is a label query over a set of resources. The result of
     matchLabels and matchExpressions are ANDed. An empty label selector matches
     all objects. A null label selector matches no objects.

FIELDS:
   matchExpressions     <[]Object>
     matchExpressions is a list of label selector requirements. The requirements
     are ANDed.

   matchLabels  <map[string]string>
     matchLabels is a map of {key,value} pairs. A single {key,value} in the
     matchLabels map is equivalent to an element of matchExpressions, whose key
     field is "key", the operator is "In", and the values array contains only
     "value". The requirements are ANDed.
```

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00026

```
candidate@base:~$ ssh ckad00026
```

```
candidate@master01:~$
```

【2】 修改 yaml 配置文件

vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入
vi /ckad/credible-mite/www.yaml

1、修改红色部分
api 版本修改为 apps/v1
并在 spec 下添加或修改 selector 标签，这个标签要和下面的 template 里的标签一致。

```
apiVersion: apps/v1
  selector:
    matchLabels:
      app: nginx
```

2、检查 namespace 是否对，如果不对，则修改为题目要求的 garfish namespace。

```
apiVersion: extention/beta
kind: Deployment
metadata:
  name: www-deployment
  namespace: garfish      #按照题目要求修改 namespace
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: www-deployment
  namespace: garfish
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16
        ports:
          - containerPort: 80
```

创建

kubectl apply -f /ckad/credible-mite/www.yaml

```
candidate@master01:~$ vi /ckad/credible-mite/www.yaml
candidate@master01:~$
candidate@master01:~$ kubectl apply -f /ckad/credible-mite/www.yaml
deployment.apps/www-deployment created
candidate@master01:~$
```

【3】 检查

kubectl -n garfish get deployment

kubectl -n garfish get pod

```
candidate@master01:~$ kubectl -n garfish get deployment
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
www-deployment   1/1     1            1           35s
candidate@master01:~$
candidate@master01:~$ kubectl -n garfish get pod
NAME                              READY   STATUS    RESTARTS   AGE
www-deployment-797697d745-bxr2j   1/1     Running   0          40s
candidate@master01:~$
```

【4】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00026 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00026 closed.
candidate@base:~$
```

6、金丝雀部署

（考试的考题内容，只有下面方框里的内容）

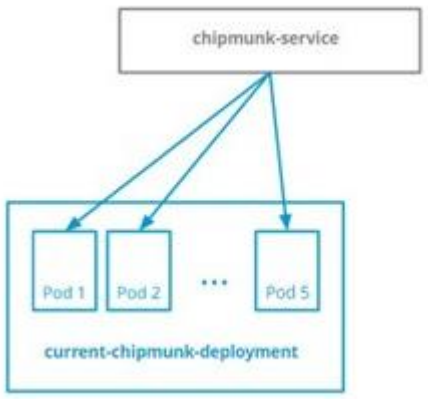
```
您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00023
```

Contex

为了测试新的应用程序发布，您需要准备一个金丝雀部署。

Task

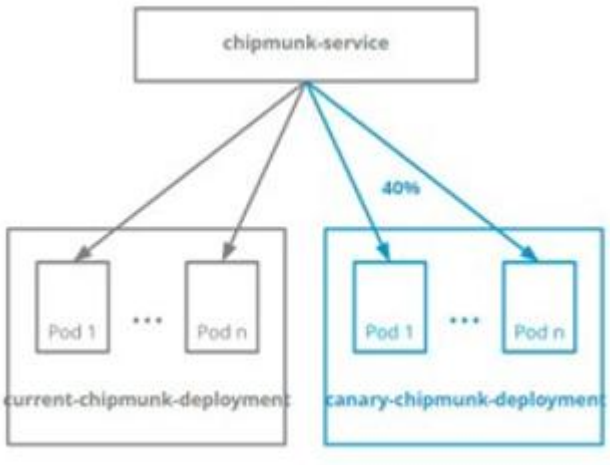
namespace goshawk 中名为 chipmunk-service 的 Service 指向名为 current-chipmunk-deployment 的 Deployment 创建的 5 个 Pod。



你可以在/ckad/goshawk/中找到 current-chipmunk-deployment 的清单文件。

1、在同一 namespace 中创建一个相同的 Deployment，名为 canary-chipmunk-deployment
2、修改 Deployment，以便:

- 在 namespace goshawk 中运行的 Pod 的最大数量为 10 个
- chipmunk.service 流量的 40%流向 Pod canary-chipmunk-deployment



参考地址

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

```
student@node01:~$ kubectl scale -h
Set a new size for a deployment, replica set, replication controller, or stateful set.

 Scale also allows users to specify one or more preconditions for the scale action.

 If --current-replicas or --resource-version is specified, it is validated before the scale is attempted, and it is
guaranteed that the precondition holds true when the scale is sent to the server.

Examples:
  # Scale a replica set named 'foo' to 3
  kubectl scale --replicas=3 rs/foo

  # Scale a resource identified by type and name specified in "foo.yaml" to 3
  kubectl scale --replicas=3 -f foo.yaml

  # If the deployment named mysql's current size is 2, scale mysql to 3
  kubectl scale --current-replicas=2 --replicas=3 deployment/mysql

  # Scale multiple replication controllers
  kubectl scale --replicas=5 rc/foo rc/bar rc/baz

  # Scale stateful set named 'web' to 3
  kubectl scale --replicas=3 statefulset/web
```

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点
考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00023

```
candidate@base:~$ ssh ckad00023
```

```
candidate@master01:~$
```

【2】 开始修改配置文件

备份一下，防止改错

cp /ckad/goshawk/current-chipmunk-deployment.yaml bak.yaml

编辑文件

vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入

vi /ckad/goshawk/current-chipmunk-deployment.yaml

创建金丝雀，修改以下红色字体

……

```
metadata:
  name: canary-chipmunk-deployment          #这个根据题目要求修改
  namespace: goshawk          #检查一下，确保是对的
spec:
  replicas: 5
  selector:
    matchLabels:
      app: canary          #修改为 canary
      run: chipmunk        #这个是 current-chipmunk-deployment 和 canary-chipmunk-deployment 都有的公共标签。
  template:
    metadata:
      labels:
        app: canary       #修改为 canary
        run: chipmunk       #这个是 current-chipmunk-deployment 和 canary-chipmunk-deployment 都有的公共标签。
```

【3】 创建

kubectl apply -f /ckad/goshawk/current-chipmunk-deployment.yaml

```
candidate@master01:~$ kubectl apply -f /ckad/goshawk/current-chipmunk-deployment.yaml
deployment.apps/canary-chipmunk-deployment created
candidate@master01:~$
```

【4】 按照题目要求，修复副本数

总共 10 个 Pod，将 60%流量给当前版本 Pod，就是 current-chipmunk-deployment 扩容至 6 个 pod

kubectl scale deployment current-chipmunk-deployment --replicas=6 -n goshawk

将 40%流量给金丝雀版本 Pod，就是 canary-chipmunk-deployment 扩容至 4 个 pod

kubectl scale deployment canary-chipmunk-deployment --replicas=4 -n goshawk

注意，如果考试时，有可能考将 20%流量给金丝雀版本 Pod，那就是原先为 8 个，金丝雀为 2 个，要会变通。

```
candidate@master01:~$ kubectl scale deployment current-chipmunk-deployment --replicas=6 -n goshawk
deployment.apps/current-chipmunk-deployment scaled
candidate@master01:~$ kubectl scale deployment canary-chipmunk-deployment --replicas=4 -n goshawk
deployment.apps/canary-chipmunk-deployment scaled
candidate@master01:~$
```

【5】 检查

kubectl get pod -n goshawk

```
candidate@master01:~$ kubectl get pod -n goshawk
NAME                                        READY   STATUS    RESTARTS        AGE
canary-chipmunk-deployment-598655dd47-4qsg7    1/1     Running   0               57s
canary-chipmunk-deployment-598655dd47-5lm7g    1/1     Running   0               57s
canary-chipmunk-deployment-598655dd47-f2v4w    1/1     Running   0               57s
canary-chipmunk-deployment-598655dd47-rfj5k    1/1     Running   0               57s
current-chipmunk-deployment-6774785b5c-7qkth   1/1     Running   0               44s
current-chipmunk-deployment-6774785b5c-gq4dv   1/1     Running   2 (5m10s ago)   25h
current-chipmunk-deployment-6774785b5c-hh69h   1/1     Running   2 (5m10s ago)   25h
current-chipmunk-deployment-6774785b5c-kbzdx   1/1     Running   2 (5m9s ago)    25h
current-chipmunk-deployment-6774785b5c-pz46j   1/1     Running   2 (5m10s ago)   25h
current-chipmunk-deployment-6774785b5c-xwqmd   1/1     Running   2 (5m9s ago)    25h
candidate@master01:~$
```

【6】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00023 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00023 closed.
candidate@base:~$
```

扩展： 感兴趣的同学，可以阅读下文。

感谢网友華鴻（horry_tom2*）

这道题其实最后有一个检查方法的，curl svc 会打印流量途径 pod 的 label 标签，如下

为了区分不同流量，可以在部署金丝雀的时候，修改一个 lable 标签，这样，crul svc 时打印标签，会打印原先和金丝雀 pod 不同的标签了。

但是修改金丝雀哪个 label 呢？

我们可以先检查一下 svc，你会发现，svc 暴露的服务（kubectl -n goshawk get svc -o wide），选择的是原先 deployment 中两个标签中的一个，比如是 run: chipmunk（svc 选择是的 pod 的 label，而不是 deployment 的 lable），那么我们只需要部署金丝雀时，修改另一个标签，就是修改 app: current 为 app: canary，这样 curl svc 时，打印的标签就是 current 和 canary，就能够显示出测试的效果了。记住，一定不要修改 svc selcet 的 label，否则这道题，就得 0 分了。

能看懂这一块“扩展”知识的，考试时，可以这样修改。看不懂的，也无需在意，按照前面的答题步骤操作即可。

比如测试环境里：

```
candidate@master01:~$ kubectl get pod -n goshawk --show-labels
NAME                                        READY   STATUS    RESTARTS        AGE     LABELS
canary-chipmunk-deployment-598655dd47-4qsg7    1/1     Running   0               9m12s   app=canary,pod-template-hash=598655dd47,run=chipmunk
canary-chipmunk-deployment-598655dd47-5lm7g    1/1     Running   0               9m12s   app=canary,pod-template-hash=598655dd47,run=chipmunk
canary-chipmunk-deployment-598655dd47-f2v4w    1/1     Running   0               9m12s   app=canary,pod-template-hash=598655dd47,run=chipmunk
canary-chipmunk-deployment-598655dd47-rfj5k    1/1     Running   0               9m12s   app=canary,pod-template-hash=598655dd47,run=chipmunk
current-chipmunk-deployment-6774785b5c-7qkth   1/1     Running   0               8m59s   app=current,pod-template-hash=6774785b5c,run=chipmunk
current-chipmunk-deployment-6774785b5c-gq4dv   1/1     Running   2 (13m ago)     25h     app=current,pod-template-hash=6774785b5c,run=chipmunk
current-chipmunk-deployment-6774785b5c-hh69h   1/1     Running   2 (13m ago)     25h     app=current,pod-template-hash=6774785b5c,run=chipmunk
current-chipmunk-deployment-6774785b5c-kbzdx   1/1     Running   2 (13m ago)     25h     app=current,pod-template-hash=6774785b5c,run=chipmunk
current-chipmunk-deployment-6774785b5c-pz46j   1/1     Running   2 (13m ago)     25h     app=current,pod-template-hash=6774785b5c,run=chipmunk
current-chipmunk-deployment-6774785b5c-xwqmd   1/1     Running   2 (13m ago)     25h     app=current,pod-template-hash=6774785b5c,run=chipmunk
candidate@master01:~$
candidate@master01:~$ kubectl -n goshawk get svc -o wide
NAME              TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   SELECTOR
chipmunk-service  NodePort   10.106.68.199   <none>        80:32074/TCP   44s   run=chipmunk
candidate@master01:~$
candidate@master01:~$ curl 10.106.68.199
Hello World ^_^
candidate@master01:~$ curl 10.106.68.199
Hello World ^_^
candidate@master01:~$ curl 10.106.68.199
Hi , I am Hi ~~~
candidate@master01:~$ curl 10.106.68.199
Hello World ^_^
candidate@master01:~$ curl 10.106.68.199
Hi , I am Hi ~~~
candidate@master01:~$
```

7、配置 Container 安全上下文

（考试的考题内容，只有下面方框里的内容）

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00029

Task
修改运行在 namespace quetzal 名为 broker-deployment 的现有 Deployment，使其容器

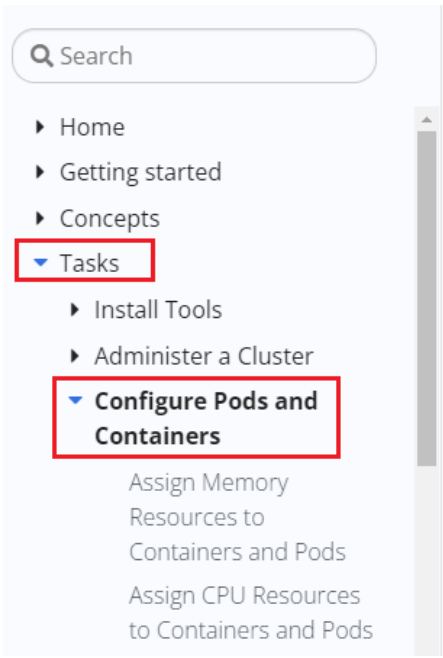
- 以用户 30000 运行
- 禁止特权提升。

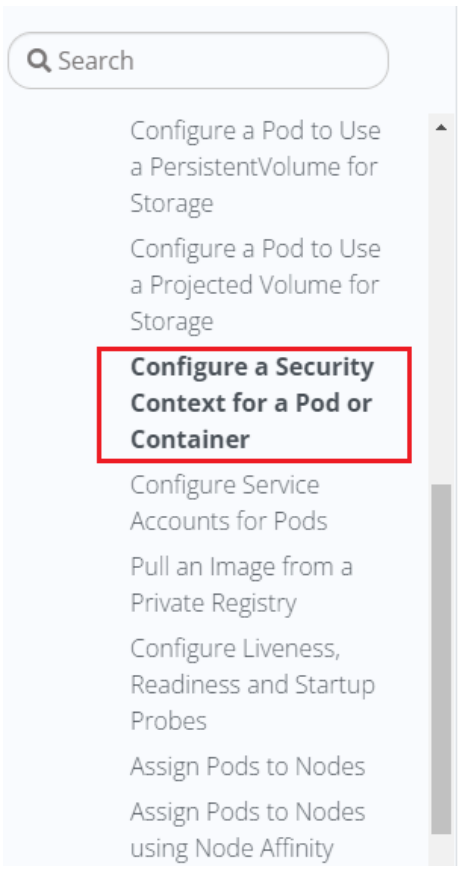
您可以在/ckad/daring-moccasin/broker-deployment.yaml 找到 broker-deployment 的清单文件。（模拟环境无此文件，因为做这道题也不需要此文件）

**参考地址

（需要复制网页内容）

依次点击下图，打开最终网址。（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/security-context/





为 Container 设置安全性上下文

若要为 Container 设置安全性配置，可以在 Container 清单中包含 `securityContext` 字段。 `securityContext` 字段的取值是一个 SecurityContext 对象。你为 Container 设置的安全性配置仅适用于该容器本身，并且所指定的设置在与 Pod 层面设置的内容发生重叠时，会重载后者。Container 层面的设置不会影响到 Pod 的卷。

下面是一个 Pod 的配置文件，其中包含一个 Container。Pod 和 Container 都有 `securityContext` 字段：

pods/security/security-c

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo-2
spec:
  securityContext:
    runAsUser: 1000
  containers:
  - name: sec-ctx-demo-2
    image: gcr.io/google-samples/node-hello:1.0
    securityContext:
      runAsUser: 2000
      allowPrivilegeEscalation: false
```

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00029

```
candidate@base:~$ ssh ckad00029
```

```
candidate@master01:~$
```

【2】 在线编辑

vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入

kubectl edit deployment broker-deployment -n quetzal

在 containers:下的 image:和 name:下面添加：

```
spec:
  containers:
  - image: busybox
    imagePullPolicy: IfNotPresent
    name: sec-ctx-demo
    securityContext:
      runAsUser: 30000
      allowPrivilegeEscalation: false
```

```
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: broker-deployment
    spec:
      containers:
      - command:
        - sh
        - -c
        - sleep 5h
        image: busybox
        imagePullPolicy: IfNotPresent
        name: sec-ctx-demo
        securityContext:
          runAsUser: 30000
          allowPrivilegeEscalation: false
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /data/demo
          name: sec-ctx-vol
      dnsPolicy: ClusterFirst
      restartPolicy: Always
```

提示 edited，表示编辑成功。

```
candidate@master01:~$ kubectl edit deployment broker-deployment -n quetzal
deployment.apps/broker-deployment edited
candidate@master01:~$
```

【3】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00029 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00029 closed.
candidate@base:~$
```

8、创建 Deployment 并指定环境变量

（考试的考题内容，只有下面方框里的内容）

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00014

Task
在现有的 namespace ckad00014 中创建一个运行 6 个 Pod 副本，名为 api 的 Deployment。用 nginx:1.16 的镜像来指定一个容器。
将名为 NGINX_PORT 且值为 8000 的环境变量添加到容器中，然后公开端口 80

参考地址

没必要参考网址，建议背过命令。

如果非要参考，可以按照下面方法。
依次点击下图，打开最终网址。（看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/tasks/inject-data-application/define-environment-variable-container/

Define Environment Variables for a Container

This page shows how to define environment variables for a container in a Kubernetes Pod.

Before you begin

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using minikube or you can use one of these Kubernetes playgrounds:

- Killercoda
- Play with Kubernetes

Define an environment variable for a container

When you create a Pod, you can set environment variables for the containers that run in the Pod. To set environment variables, include the `env` or `envFrom` field in the configuration file.

In this exercise, you create a Pod that runs one container. The configuration file for the Pod defines an environment variable with name `DEMO_GREETING` and value `"Hello from the environment"`. Here is the configuration manifest for the Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
```

为容器设置一个环境变量

创建 Pod 时，可以为其下的容器设置环境变量。通过配置文件的 `env` 或者 `envFrom` 字段来设置环境变量。

本示例中，将创建一个只包含单个容器的 Pod。Pod 的配置文件中设置环境变量的名称为 `DEMO_GREETING`， 其值为 `"Hello from the environment"`。下面是 Pod 的配置清单：

pods/injec

```
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envars
spec:
  containers:
  - name: envar-demo-container
    image: gcr.io/google-samples/node-hello:1.0
    env:
    - name: DEMO_GREETING
      value: "Hello from the environment"
    - name: DEMO_FAREWELL
      value: "Such a sweet sorrow"
```

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令

```
ssh ckad00014
```

```
candidate@base:~$ ssh ckad00014
```

```
candidate@master01:~$
```

【2】 使用 kubeclt 命令，创建需要的 yaml 文件

下面命令背过，如果不会，就使用 kubectl -h 来帮助。

kubectl -n ckad00014 create deployment api --image=nginx:1.16 --replicas=6 --port=80 --dry-run=client -o yaml > api.yaml

```
candidate@master01:~$ kubectl -n ckad00014 create deployment api --image=nginx:1.16 --replicas=6 --port=80 --dry-run=client -o yaml > api.yaml
candidate@master01:~$
```

【3】 编辑 yaml 文件

vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入

vi api.yaml

在 containers:里的 image:和 name:下添加

```
        env:
        - name: NGINX_PORT
          value: "8000"     #注意要加双引号
```

```
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: api
    spec:
      containers:
      - image: nginx:1.16
        name: nginx
        env:
        - name: NGINX_PORT
          value: "8000"
        ports:
        - containerPort: 80
        resources: {}
status: {}
```

【4】 创建

kubectl apply -f api.yaml

```
candidate@master01:~$ kubectl apply -f api.yaml
deployment.apps/api created
candidate@master01:~$
```

【5】 检查

kubectl -n ckad00014 get pod

```
candidate@master01:~$ kubectl -n ckad00014 get pod
NAME                   READY   STATUS    RESTARTS   AGE
api-949766568-9dtg5    1/1     Running   0          25s
api-949766568-bw7t9    1/1     Running   0          25s
api-949766568-clzlj    1/1     Running   0          25s
api-949766568-f5htl    1/1     Running   0          25s
api-949766568-fh8zh    1/1     Running   0          25s
api-949766568-z2b4d    1/1     Running   0          25s
candidate@master01:~$
```

【6】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00014 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00014 closed.
candidate@base:~$
```

9、RBAC 授权

（考试的考题内容，只有下面方框里的内容）

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00028

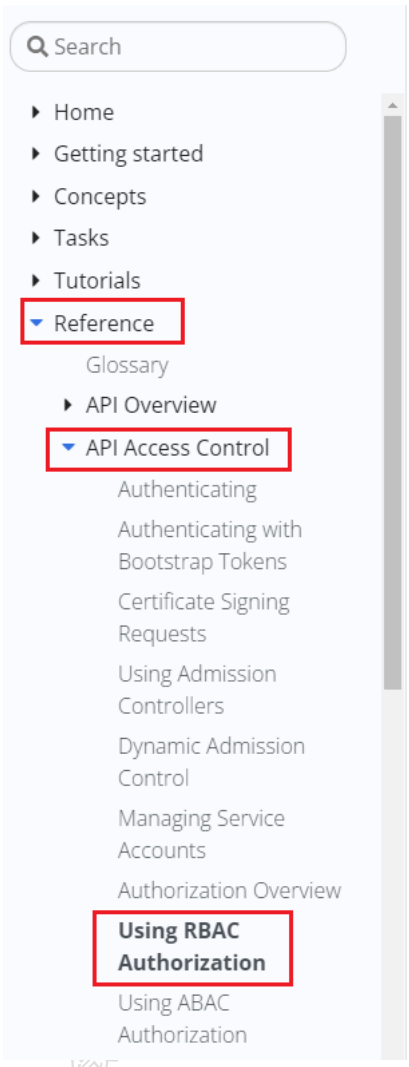
Task
在名为 honeybee-deployment 的 Deployment 和 namespace gorilla 中的一个 Pod 正在记录错误.
1 查看日志以识别错误消息。
找出错误，包括 User “system:serviceaccount:gorilla:default ”cannot list resource “serviceaccounts” […] in the namespace “gorilla”
2 更新 Deployment honeybee-deployment 以解决 Pod 日志中的错误。
您可以在/ckad/prompt-escargot/honeybee-deployment.yaml 中找到 honeybee-deployment 的清单文件

参考地址

没必要参考网址，使用-h 帮助更方便。

如果非要参考，可以按照下面方法。

依次点击下图，打开最终网址。（看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/reference/access-authn-authz/rbac/



Kubernetes Documentation / Reference / API Access Control / Using RBAC Authorization

Using RBAC Authorization

Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within your organization.

RBAC authorization uses the `rbac.authorization.k8s.io` API group to drive authorization decisions, allowing you to dynamically configure policies through the Kubernetes API.

To enable RBAC, start the API server with the `--authorization-mode` flag set to a comma-separated list that includes `RBAC`; for example:

```
kube-apiserver --authorization-mode=Example,RBAC --other-options --more-options
```

API objects

The RBAC API declares four kinds of Kubernetes object: *Role*, *ClusterRole*, *RoleBinding* and *ClusterRoleBinding*. You can describe objects, or amend them, using tools such as `kubectl`, just like any other Kubernetes object.

> **Caution:** These objects, by design, impose access restrictions. If you are making changes to a cluster as you learn, see privilege escalation prevention and bootstrapping to understand how those restrictions can prevent you making some changes.

Role and ClusterRole

An RBAC *Role* or *ClusterRole* contains rules that represent a set of permissions. Permissions are purely additive (there are no "deny" rules).

服务账户权限

默认的 RBAC 策略为控制面组件、节点和控制器授予权限。 但是不会对 `kube-system` 名字空间之外的服务账户授予权限。 （除了授予所有已认证用户的发现权限）

这使得你可以根据需要向特定 ServiceAccount 授予特定权限。 细粒度的角色绑定可带来更好的安全性，但需要更多精力管理。 粗粒度的授权可能导致 ServiceAccount 被授予不必要的 API 访问权限（甚至导致潜在的权限提升）， 但更易于管理。

按从最安全到最不安全的顺序，存在以下方法：

1. 为特定应用的服务账户授予角色（最佳实践）

   这要求应用在其 Pod 规约中指定 `serviceAccountName`， 并额外创建服务账户（包括通过 API、应用程序清单、 `kubectl create serviceaccount` 等）。

   例如，在名字空间 “my-namespace” 中授予服务账户 “my-sa” 只读权限：

```
kubectl create rolebinding my-sa-view \
  --clusterrole=view \
  --serviceaccount=my-namespace:my-sa \
  --namespace=my-namespace
```

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00028

```
candidate@base:~$ ssh ckad00028
```

```
candidate@master01:~$
```

【2】 通过 logs 打印错误日志

kubectl -n gorilla get pod
kubectl -n gorilla logs honeybee-deployment-***

```
candidate@master01:~$ kubectl -n gorilla get pod
NAME                                 READY   STATUS    RESTARTS      AGE
honeybee-deployment-6f6c4b7bf9-7z4l7   1/1     Running   2 (37m ago)   26h
candidate@master01:~$
candidate@master01:~$ kubectl -n gorilla logs honeybee-deployment-6f6c4b7bf9-7z4l7
2025-02-16 13:41:44+00:00
Error from server (Forbidden): serviceaccounts is forbidden: User "system:serviceaccount:gorilla:default" cannot list resource
 "serviceaccounts" in API group "" in the namespace "gorilla"
2025-02-16 13:41:54+00:00
Error from server (Forbidden): serviceaccounts is forbidden: User "system:serviceaccount:gorilla:default" cannot list resource
 "serviceaccounts" in API group "" in the namespace "gorilla"
2025-02-16 13:42:04+00:00
Error from server (Forbidden): serviceaccounts is forbidden: User "system:serviceaccount:gorilla:default" cannot list resource
 "serviceaccounts" in API group "" in the namespace "gorilla"
```

考试时，无论是不能 list pods，还是不能 list deployments，或者不能 list serviceaccounts，做法都一样。

【3】 检查现有的 serviceaccount、rolebinding、role

（感谢网友 wxid_pc6olyf169ek**）

kubectl -n gorilla get sa

```
candidate@master01:~$ kubectl -n gorilla get sa
NAME         SECRETS   AGE
boxweb-sa    0         26h
default      0         26h
gorilla-sa   0         26h
candidate@master01:~$
```

查看 rolebinding 和 role 的绑定

kubectl -n gorilla get rolebinding

```
candidate@master01:~$ kubectl -n gorilla get rolebinding
NAME                  ROLE                 AGE
boxweb-rolebinding    Role/boxweb-role     26h
gorilla-rolebinding   Role/gorilla-role    26h
candidate@master01:~$
```

【4】 通过 rolebinding，查看 role 和 serviceaccount 的绑定关系

kubectl -n gorilla describe rolebinding

```
candidate@master01:~$ kubectl -n gorilla describe rolebinding
Name:         boxweb-rolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  boxweb-role
Subjects:
  Kind            Name       Namespace
  ----            ----       ---------
  ServiceAccount  boxweb-sa  gorilla


Name:         gorilla-rolebinding
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  gorilla-role
Subjects:
  Kind            Name        Namespace
  ----            ----        ---------
  ServiceAccount  gorilla-sa  gorilla
candidate@master01:~$
```

【5】 检查 role，寻找有 get 或 list 权限的 role

kubectl -n gorilla describe role

```
candidate@master01:~$ kubectl -n gorilla describe role
Name:         boxweb-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names  Verbs
  ---------         -----------------  --------------  -----
  pods              []                 []              [watch]
  serviceaccounts   []                 []              [watch]
  deployments.apps  []                 []              [watch]


Name:         gorilla-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names  Verbs
  ---------         -----------------  --------------  -----
  pods              []                 []              [get list]
  serviceaccounts   []                 []              [get list]
  deployments.apps  []                 []              [get list]
candidate@master01:~$
```

通过上面的一系列操作，检查发现，gorilla-role 符合题目的要求，它对应的 serviceaccount 是 gorilla-sa

【6】 设置 honeybee-deployment 的 serviceaccount

（考试时，其实就是跟题目里的 namespace 相同的那个 sa 是正确的。对应这道模拟题，就是 gorilla-sa 是正确的）

kubectl -n gorilla set serviceaccount deployments honeybee-deployment gorilla-sa

```
candidate@master01:~$ kubectl -n gorilla set serviceaccount deployments honeybee-deployment gorilla-sa
deployment.apps/honeybee-deployment serviceaccount updated
candidate@master01:~$
```

提示 update 表示修改成功

【7】 等 2 分钟，会自动生成一个新 pod，再次检查新 Pod，不报错了，会输出一堆内容。

kubectl -n gorilla logs honeybee-deployment-***

```
candidate@master01:~$ kubectl -n gorilla logs honeybee-deployment-b899d6795-flk4c
2025-02-16 14:24:07+00:00
NAME         SECRETS   AGE
boxweb-sa    0         26h
default      0         26h
gorilla-sa   0         26h
2025-02-16 14:24:17+00:00
NAME         SECRETS   AGE
boxweb-sa    0         26h
default      0         26h
gorilla-sa   0         26h
2025-02-16 14:24:27+00:00
NAME         SECRETS   AGE
boxweb-sa    0         26h
default      0         26h
gorilla-sa   0         26h
```

【8】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00028 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00028 closed.
candidate@base:~$
```

10、Secret

（考试的考题内容，只有下面方框里的内容）

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00004

Task

1、在 namespace default 中创建一个名为 another-secret 并包含以下单个键/值对的 Secret:
key1: value2

2、在 namespace default 中创建一个名为 nginx-secret 的 Pod。用 nginx:1.16 的镜像来指定一个容器。
添加一个名为 COOL_VARIABLE 的环境变量来使用 secret 键 key1 的值。

**参考地址

（需要复制网页内容）

依次点击下图，打开最终网址。（看不懂英文的，可右上角翻译成中文）
https://kubernetes.io/zh-cn/docs/concepts/configuration/secret/

Secrets

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don't need to include confidential data in your application code.

Because Secrets can be created independently of the Pods that use them, there is less risk of the Secret (and its data) being exposed during the workflow of creating, viewing, and editing Pods. Kubernetes, and applications that run in your cluster, can also take additional precautions with Secrets, such as avoiding writing secret data to nonvolatile storage.

Secrets are similar to ConfigMaps but are specifically intended to hold confidential data.

> **Caution:**
> Kubernetes Secrets are, by default, stored unencrypted in the API server's underlying data store (etcd). Anyone with API access can retrieve or modify a Secret, and so can anyone with access to etcd. Additionally, anyone who is authorized to create a Pod in a namespace can use that access to read any Secret in that namespace; this includes indirect access such as the ability to create a Deployment.
>
> In order to safely use Secrets, take at least the following steps:
>
> 1. Enable Encryption at Rest for Secrets.
> 2. Enable or configure RBAC rules that restrict reading and writing the Secret. Be aware that secrets can be obtained implicitly by anyone with the permission to create a Pod.
> 3. Where appropriate, also use mechanisms such as RBAC to limit which principals are allowed to create new Secrets or replace existing ones.

Kubelet 组件会维护一个缓存，在其中保存节点上 Pod 卷中使用的 Secret 的当前主键和取值。 你可以配置 kubelet 如何检测所缓存数值的变化。 kubelet 配置中的 `configMapAndSecretChangeDetectionStrategy` 字段控制 kubelet 所采用的策略。 默认的策略是 `Watch`。

对 Secret 的更新操作既可以通过 API 的 watch 机制（默认）来传播， 基于设置了生命期的缓存获取，也可以通过 kubelet 的同步回路来从集群的 API 服务器上轮询获取。

因此，从 Secret 被更新到新的主键被投射到 Pod 中，中间存在一个延迟。 这一延迟的上限是 kubelet 的同步周期加上缓存的传播延迟， 其中缓存的传播延迟取决于所选择的缓存类型。 对应上一段中提到的几种传播机制，延迟时长为 watch 的传播延迟、所配置的缓存 TTL 或者对于直接轮询而言是零。

以环境变量的方式使用 Secret

如果需要在 Pod 中以环境变量的形式使用 Secret：

1. 对于 Pod 规约中的每个容器，针对你要使用的每个 Secret 键，将对应的环境变量添加到 `env[].valueFrom.secretKeyRef` 中。
2. 更改你的镜像或命令行，以便程序能够从指定的环境变量找到所需要的值。

相关的指示说明， 可以参阅使用 Secret 数据定义容器变量。

点这里，切换网页

非法环境变量

- 在 Pod 规约中定义环境变量：

pods/inject/pod-multiple-secret-env-variable.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: envvars-multiple-secrets
spec:
  containers:
  - name: envars-test-container
    image: nginx
    env:
    - name: BACKEND_USERNAME
      valueFrom:
        secretKeyRef:
          name: backend-user
          key: backend-username
    - name: DB_USERNAME
      valueFrom:
        secretKeyRef:
          name: db-user
          key: db-username
```

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00004

```
candidate@base:~$ ssh ckad00004
```

```
candidate@master01:~$
```

【2】 使用 kubeclt 命令创建 secret

kubectl create secret generic another-secret --from-literal=key1=value2

```
candidate@master01:~$ kubectl create secret generic another-secret --from-literal=key1=value2
secret/another-secret created
candidate@master01:~$
```

【3】 创建题目要求的 Pod

vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入

vi nginx-secret.yaml

添加如下内容

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-secret
spec:
  containers:
  - image: nginx:1.16
    name: nginx-secret
    env:
    - name: COOL_VARIABLE
      valueFrom:
        secretKeyRef:
          name: another-secret
          key: key1
```

创建

kubectl apply -f nginx-secret.yaml

```
candidate@master01:~$ vi nginx-secret.yaml
candidate@master01:~$
candidate@master01:~$ kubectl apply -f nginx-secret.yaml
pod/nginx-secret created
candidate@master01:~$
```

【4】 检查

kubectl get pod nginx-secret

```
candidate@master01:~$ kubectl get pod nginx-secret
NAME           READY   STATUS    RESTARTS   AGE
nginx-secret   1/1     Running   0          33s
candidate@master01:~$
```

【5】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

退出 ckad00004 节点，返回到 candidate@base 用户和节点下
exit

```
candidate@master01:~$ exit
logout
Connection to ckad00004 closed.
candidate@base:~$
```

11、Pod 健康检查 readinessProbe

（考试的考题内容，只有下面方框里的内容）

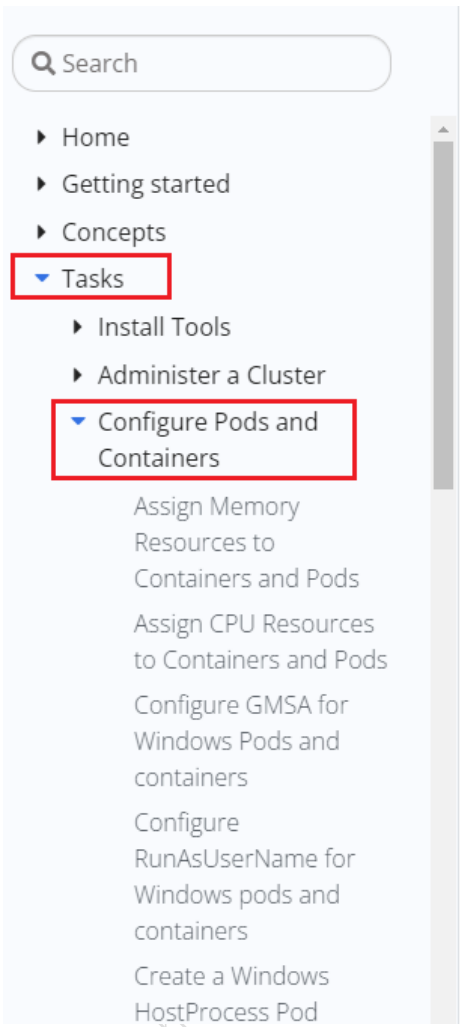
您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00027

在 prod27 namespace 中，名为 probe-http 的 deployment 在端口 80 上运行着 Web 应用程序。
修改 Deployment：
指定 readiness probe 路径为 /healthz/return200
将 initialDelaySeconds 设置为 15，即在执行第一次探测前应该等待 15 秒
将 periodSeconds 设置为 20，即执行探测的时间间隔为 20 秒

**参考地址

（需要复制网页内容）

依次点击下图，打开最终网址。（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-liveness-readiness-startup-probes/



Kubernetes Documentation / Tasks / Configure Pods and Containers / Configure Liveness, Readiness and Startup Probes

Configure Liveness, Readiness and Startup Probes

This page shows how to configure liveness, readiness and startup probes for containers.

The kubelet uses liveness probes to know when to restart a container. For example, liveness probes could catch a deadlock, where an application is running, but unable to make progress. Restarting a container in such a state can help to make the application more available despite bugs.

The kubelet uses readiness probes to know when a container is ready to start accepting traffic. A Pod is considered ready when all of its containers are ready. One use of this signal is to control which Pods are used as backends for Services. When a Pod is not ready, it is removed from Service load balancers.

The kubelet uses startup probes to know when a container application has started. If such a probe is configured, it disables liveness and readiness checks until it succeeds, making sure those probes don't interfere with the application startup. This can be used to adopt liveness checks on slow starting containers, avoiding them getting killed by the kubelet before they are up and running.

Before you begin

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. It is recommended to run this tutorial on a cluster with at least two nodes that are not acting as control plane hosts. If you do not already have a cluster, you can create one by using minikube or you can use one of these Kubernetes playgrounds:

- Killercoda
- Play with Kubernetes

Define a TCP liveness probe

A third type of liveness probe uses a TCP socket. With this configuration, the kubelet will attempt to open a socket to your container on the specified port. If it can establish a connection, the container is considered healthy, if it can't it is considered a failure.

pods/probe

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: registry.k8s.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

定义 TCP 的存活探测

第三种类型的存活探测是使用 TCP 套接字。 使用这种
试在指定端口和容器建立套接字链接。 如果能建立连
是健康的，如果不能则这个容器就被看作是有问题的。

pods

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: goproxy
  labels:
    app: goproxy
spec:
  containers:
  - name: goproxy
    image: k8s.gcr.io/goproxy:0.1
    ports:
    - containerPort: 8080
    readinessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 10
    livenessProbe:
      tcpSocket:
        port: 8080
      initialDelaySeconds: 15
      periodSeconds: 20
```

定义一个存活态 HTTP 请求接口

另外一种类型的存活探测方式是使用 HTTP GET 请求。 下面是一个 Pod 的配置文件，其中运行一个基于 `k8s.gcr.io/liveness` 镜像的容器。

pods/probe/http-

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
    livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
        httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 3
      periodSeconds: 3
```

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00027

```
candidate@base:~$ ssh ckad00027
```

```
candidate@master01:~$
```

【2】 编辑 deployment probe-http

readinessProbe 是就绪探测
vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入

kubectl -n prod27 edit deployment probe-http

在 image 或者 name 下面添加，注意空格对齐位置。

```
        readinessProbe:
          httpGet:
            path: /healthz/return200
            port: 80
          initialDelaySeconds: 15
          periodSeconds: 20
```

```
  template:
    metadata:
      creationTimestamp: null
      labels:
        test: probe
    spec:
      containers:
      - image: vicuu/helloweb:v1
        imagePullPolicy: IfNotPresent
        name: probe
        readinessProbe:
          httpGet:
            path: /healthz/return200
            port: 80
          initialDelaySeconds: 15
          periodSeconds: 20
        ports:
        - containerPort: 80
          protocol: TCP
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
```

提示 edited，表示修改成功

```
candidate@master01:~$ kubectl -n prod27 edit deployment probe-http
deployment.apps/probe-http edited
candidate@master01:~$
```

【3】 检查

kubectl -n prod27 get pod

```
candidate@master01:~$ kubectl -n prod27 get pod
NAME                          READY   STATUS    RESTARTS   AGE
probe-http-6bf8d84dff-2prsg   1/1     Running   0          51s
candidate@master01:~$
```

【4】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

退出 ckad00027 节点，返回到 candidate@base 用户和节点下
exit

```
candidate@master01:~$ exit
logout
Connection to ckad00027 closed.
candidate@base:~$
```

真实考试时的 yaml 文件内容如下：

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app-deployment
  namespace: staging
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
          - containerPort: 8081
            protocol: TCP
        readinessProbe:
          httpGet:
            path: /healthz
            port: 8081
          initialDelaySeconds: 10
          periodSeconds: 5
        volumeMounts:
        - mountPath: /etc/nginx/conf.d/
          name: config
        - mountPath: /usr/share/nginx/html/
```

考试时，是在staging命名空间

要修改的内容

12、升级与回滚

（考试的考题内容，只有下面方框里的内容）

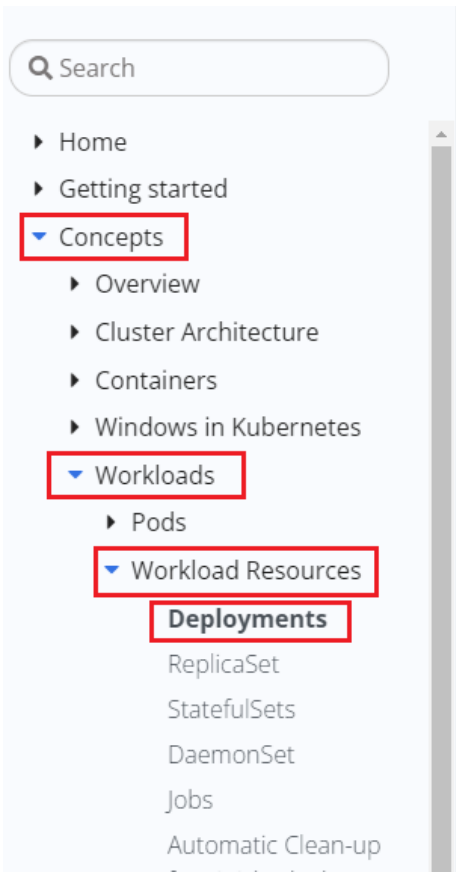
您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00015

Task
1 更新 namespace ckad00015 中的 Deployment webapp 的比例缩放配置，将 maxSurge 设置为 5%，将 maxUnavailable 设置为 5%
2 更新 Deployment webapp 以让容器镜像 lfccncf/nginx 使用版本标签 1.13.7
3 将 Deployment webapp 回滚至前一版本。

参考地址

没必要参考网址，建议背过命令。
如果非要参考，可以按照下面方法。

依次点击下图，打开最终网址。（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/



Kubernetes Documentation / Concepts / Workloads / Workload Resources / Deployments

Deployments

A *Deployment* provides declarative updates for Pods and ReplicaSets.

You describe a *desired state* in a Deployment, and the Deployment Controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments.

> **Note:** Do not manage ReplicaSets owned by a Deployment. Consider opening an issue in the main Kubernetes repository if your use case is not covered below.

Use Case

The following are typical use cases for Deployments:

- Create a Deployment to rollout a ReplicaSet. The ReplicaSet creates Pods in the background. Check the status of the rollout to see if it succeeds or not.
- Declare the new state of the Pods by updating the PodTemplateSpec of the Deployment. A new ReplicaSet is created and the Deployment manages moving the Pods from the old ReplicaSet to the new one at a controlled rate. Each new

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令

ssh ckad00015

```
candidate@base:~$ ssh ckad00015
```

```
candidate@master01:~$
```

【2】 在线编辑 deployment

vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入

kubectl -n ckad00015 edit deployment webapp

按照题目要求，修改红框中的内容

maxSurge 和 maxUnavailable 可以是百分比（5%），可以可以是数字（比如 5），请根据题目的要求写。

```
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: webapp
  strategy:
    rollingUpdate:
      maxSurge: 5%
      maxUnavailable: 5%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: webapp
    spec:
      containers:
```

提示 edited，表示修改成功。

```
candidate@master01:~$ kubectl -n ckad00015 edit deployment webapp
deployment.apps/webapp edited
candidate@master01:~$
```

【3】 先检查一下之前的 image 版本，为 lfccncf/nginx:1.12.2

kubectl -n ckad00015 get deployments webapp -o yaml|grep -e image -e name

```
candidate@master01:~$ kubectl -n ckad00015 get deployments webapp -o yaml|grep -e image -e name
      {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"name":"webapp","namespace":"ckad00015"},"spec"
:{"replicas":1,"selector":{"matchLabels":{"app":"webapp"}},"template":{"metadata":{"labels":{"app":"webapp"}},"spec":{"contain
ers":[{"image":"lfccncf/nginx:1.12.2","name":"nginx"}]}}}}
  name: webapp
  namespace: ckad00015
      - image: lfccncf/nginx:1.12.2
        imagePullPolicy: IfNotPresent
        name: nginx
candidate@master01:~$
```

【4】 更新为 lfccncf/nginx:1.13.7，这里的 nginx=其中的 nginx 是 containers 的 name 字段，也就是图绿框里的。

kubectl set image deployment webapp nginx=lfccncf/nginx:1.13.7 -n ckad00015

```
candidate@master01:~$ kubectl set image deployment webapp nginx=lfccncf/nginx:1.13.7 -n ckad00015
deployment.apps/webapp image updated
candidate@master01:~$
```

提示updated，表示修改成功

或者 kubectl set image deployment webapp nginx=lfccncf/nginx:1.13.7 -n ckad00015 --record

注意，这里加不加--record 都行，加上，则下面步骤的 rollout history 描述里会是命令，不加，则下面第 3 步的 rollout history 描述里就是<none>

【5】 再次检查，image 为 1.13.7 了。

kubectl -n ckad00015 get deployments webapp -o yaml|grep -e image -e name

```
candidate@master01:~$ kubectl -n ckad00015 get deployments webapp -o yaml|grep -e image -e name
      {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"name":"webapp","namespace":"ckad00015"},"spec"
:{"replicas":1,"selector":{"matchLabels":{"app":"webapp"}},"template":{"metadata":{"labels":{"app":"webapp"}},"spec":{"contain
ers":[{"image":"lfccncf/nginx:1.12.2","name":"nginx"}]}}}}
  name: webapp
  namespace: ckad00015
      - image: lfccncf/nginx:1.13.7
        imagePullPolicy: IfNotPresent
        name: nginx
candidate@master01:~$
```

如果需要看历史，则可以使用下面这条命令。(考试不需要查)

kubectl -n ckad00015 rollout history deployment webapp

【6】 回滚到上一个版本

kubectl -n ckad00015 rollout undo deployment webapp

```
candidate@master01:~$ kubectl -n ckad00015 rollout undo deployment webapp
deployment.apps/webapp rolled back
candidate@master01:~$
```

或者 kubectl rollout undo deployment webapp -n ckad00015 --to-revision=1

#--to-revision 指定回滚的版本，其实，这里指不指定都行，一共就两个版本，1 为 lfccncf/nginx:1.12.2，而 2 为 lfccncf/nginx:1.13.7。

【7】 再检查，image 为 1.12.2 了。

kubectl -n ckad00015 get deployments webapp -o yaml|grep -e image -e name

```
candidate@master01:~$ kubectl -n ckad00015 get deployments webapp -o yaml|grep -e image -e name
      {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"name":"webapp","namespace":"ckad00015"},"spec"
:{"replicas":1,"selector":{"matchLabels":{"app":"webapp"}},"template":{"metadata":{"labels":{"app":"webapp"}},"spec":{"contain
ers":[{"image":"lfccncf/nginx:1.12.2","name":"nginx"}]}}}}
  name: webapp
  namespace: ckad00015
      - image: lfccncf/nginx:1.12.2
        imagePullPolicy: IfNotPresent
        name: nginx
candidate@master01:~$
```

【8】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00015 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00015 closed.
candidate@base:~$
```

13、Deployment 使用 ServiceAcount

（考试的考题内容，只有下面方框里的内容）

Contest
您的应用程序需要使用特定的 ServiceAccount

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00005

Task
更新在 namespace frontend 中的 Deployment，使其使用现有的 ServiceAccount app

参考地址

没必要参考网址，建议背过命令。

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点
考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00005

```
candidate@base:~$ ssh ckad00005
```

```
candidate@master01:~$
```

【2】 更新 deployment
检查 deployment 的名字
kubectl -n frontend get deployment

开始更新
kubectl -n frontend set serviceaccount deployments frontend-deployment app

```
candidate@master01:~$ kubectl -n frontend get deployment
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
frontend-deployment   1/1     1            1           26h
candidate@master01:~$
candidate@master01:~$ kubectl -n frontend set serviceaccount deployments frontend-deployment app
deployment.apps/frontend-deployment serviceaccount updated
candidate@master01:~$
```

提示update，表示更新成功

【3】 检查
kubectl -n frontend get deployment frontend-deployment -o yaml |grep -i serviceaccount

```
candidate@master01:~$ kubectl -n frontend get deployment frontend-deployment -o yaml |grep -i serviceaccount
      serviceAccount: app
      serviceAccountName: app
candidate@master01:~$
```

【4】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

退出 ckad00005 节点，返回到 candidate@base 用户和节点下
exit

```
candidate@master01:~$ exit
logout
Connection to ckad00005 closed.
candidate@base:~$
```

14、更新 Deployment 标签，并暴露 Service

（考试的考题内容，只有下面方框里的内容）

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00017

Task
1 首先，更新在 namespace ckad00017 中的 Deployment ckad00017-deployment ：

- 以使其运行 3 个 Pod 的副本
- 将以下标签添加到 Pod
  tier: dmz

2 然后，在 namespace ckad00017 中创建一个名为 rover 的 NodePort Service 以在 TCP 端口 81 上公开 Deployment ckad00017-deployment

参考地址

没必要参考网址，建议背过命令。

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点
考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00017

```
candidate@base:~$ ssh ckad00017
```

```
candidate@master01:~$
```

【2】 第一个 Task，在线编辑 deploymen
修改副本数为 3，
在 template 下的 labels 里面，添加另一个标签 tier: dmz

vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入
kubectl -n ckad00017 edit deployments.apps ckad00017-deployment

```
  generation: 1
  labels:
    app: ckad00017-deployment
  name: ckad00017-deployment
  namespace: ckad00017
  resourceVersion: "83285"
  uid: 64b0dcaa-49c4-4fe1-97cc-610af05b06f8
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: ckad00017-deployment
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: ckad00017-deployment
        tier: dmz
    spec:
      containers:
      - image: vicuu/nginx:hello81
        imagePullPolicy: IfNotPresent
        name: nginx
        ports:
        - containerPort: 81
          protocol: TCP
        resources: {}
```

提示 edited，表示编辑成功

```
candidate@master01:~$ kubectl -n ckad00017 edit deployments.apps ckad00017-deployment
deployment.apps/ckad00017-deployment edited
candidate@master01:~$
```

等 1 分钟，检查新 pod 的标签，可以发现新增的 tier: dmz 标签

kubectl -n ckad00017 get pod --show-labels

```
candidate@master01:~$ kubectl -n ckad00017 get pod --show-labels
NAME                                    READY   STATUS    RESTARTS   AGE   LABELS
ckad00017-deployment-d5f5d9899-bkccf    1/1     Running   0          28s   app=ckad00017-deployment,pod-template-hash=d5f5d9899,tier=dmz
ckad00017-deployment-d5f5d9899-dc5r2    1/1     Running   0          30s   app=ckad00017-deployment,pod-template-hash=d5f5d9899,tier=dmz
ckad00017-deployment-d5f5d9899-kk5s7    1/1     Running   0          29s   app=ckad00017-deployment,pod-template-hash=d5f5d9899,tier=dmz
candidate@master01:~$
```

【3】 第二个 Task，创建 service

先确认 pod 容器的端口，就是下面命令--target-port=的端口。

kubectl -n ckad00017 get deployments ckad00017-deployment -o yaml|grep -i "containerPort:"

通过检查发现，容器端口为 81。

```
candidate@master01:~$ kubectl -n ckad00017 get deployments ckad00017-deployment -o yaml|grep -i "containerPort:"
        - containerPort: 81
candidate@master01:~$
```

创建 service，

注意如果上一步查出来的是 80 端口，则下面的两个 port 都改成 80

kubectl -n ckad00017 expose deployment ckad00017-deployment --name=rover --protocol=TCP --port=81 --target-port=81 --type=NodePort

```
candidate@master01:~$ kubectl -n ckad00017 expose deployment ckad00017-deployment --name=rover --protocol=TCP --port=81 --target-port=81 --type=NodePort
service/rover exposed
candidate@master01:~$
```

最好背过，命令太长记不住，就使用 kubectl -h 和 kubectl expose -h 来帮助。

说明：

--port=81　是暴露的 service 的端口为 81。

--target-port=81　是 Deployment ckad00017-deployment 内 Pod 容器使用的端口，这个需要检查 yaml 文件。如果检查完，发现容器端口为 containerPort: 80，或者没有 containerPort，则也是用默认的 80。则暴露服务的命令需要写成。

kubectl -n ckad00017 expose deployment ckad00017-deployment --name=rover --protocol=TCP --port=81 --target-port=80 --type=NodePort

检查 service
kubectl -n ckad00017 get svc
curl 10.103.95.21:81　　#curl 查出来的 IP 的 81 端口

```
candidate@master01:~$ kubectl -n ckad00017 get svc
NAME    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
rover   NodePort   10.105.226.47   <none>        81:31383/TCP   50s
candidate@master01:~$
candidate@master01:~$ curl 10.105.226.47:81
Hello World ^_^
Port 81
candidate@master01:~$
```

【4】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00017 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00017 closed.
candidate@base:~$
```

15、NetworkPolicy 网络策略

（考试的考题内容，只有下面方框里的内容）

Contest
您需要让一个 Pod 只与其他两个 Pod，而不是和这两个以外的，进行通信。

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00018

Task
更新在 namespace ckad00018 中的 Pod ckad00018-newpod ，
使其使用一个**只**允许 此 Pod 与 Pod front 和 db 之间收发流量的 Networkpolicy。

所有必要的 NetworkPolicy 均已被创建。

在完成此项目时，请勿创建、修改或删除任何 NetworkPolicy。您只能使用现有的 NetworkPolicy。

**参考地址

（需要复制网页内容）

依次点击下图，打开最终网址。（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/

Network Policies

If you want to control traffic flow at the IP address or port level (OSI layer 3 or 4), then you might consider using Kubernetes NetworkPolicies for particular applications in your cluster. NetworkPolicies are an application-centric construct which allow you to specify how a pod is allowed to communicate with various network "entities" (we use the word "entity" here to avoid overloading the more common terms such as "endpoints" and "services", which have specific Kubernetes connotations) over the network. NetworkPolicies apply to a connection with a pod on one or both ends, and are not relevant to other connections.

The entities that a Pod can communicate with are identified through a combination of the following 3 identifiers:

1. Other pods that are allowed (exception: a pod cannot block access to itself)
2. Namespaces that are allowed
3. IP blocks (exception: traffic to and from the node where a Pod is running is always allowed, regardless of the IP address of the Pod or the node)

When defining a pod- or namespace- based NetworkPolicy, you use a selector to specify what traffic is allowed to and from the Pod(s) that match the selector.

Meanwhile, when IP based NetworkPolicies are created, we define policies based on IP blocks (CIDR ranges).

Prerequisites

Network policies are implemented by the network plugin. To use network policies, you must be using a networking solution which supports NetworkPolicy. Creating a NetworkPolicy resource without a controller that implements it will have no effect.

The Two Sorts of Pod Isolation

There are two sorts of isolation for a pod: isolation for egress, and isolation for ingress. They concern what connections may be established. "Isolation" here is not absolute, rather it means "some restrictions apply". The alternative, "non-isolated for $direction", means that no restrictions apply in the stated direction. The two sorts of isolation (or not) are declared

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
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
```

这个就是需要的标签

egress下要写的内容，可以参考上面的ingress的内容

答题

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点

考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00018

```
candidate@base:~$ ssh ckad00018
```

```
candidate@master01:~$
```

虽然这道题不需要你写 NetworkPolicy，
但是考试中会有多个 NetworkPolicy 供你选择出正确的。
在所有的 NetworkPolicy 的 yaml 文件中导找，
找 ingress 和 egress 都有标签 podSelector 的，
那么这个 NetworkPolicy 就是正确的。
那么 podSelector 里面的标签就是要给 pod 打上的正确标签。

【2】 先检查所有的 networkpolicy 和 pod 标签

通过下面的 3 条查询命令，可以得知所有 networkpolicy 中，
POD front 和 db 各有一个 networkpolicy，ckad00018-newpod 没有 networkpolicy。
所以查看 POD front 和 db 的 networkpolicy 标签，把可以跟他们各自互通的标签，设置给 ckad00018-newpod，就是这道题的做法。

检查 POD front 和 db
kubectl -n ckad00018 get pod --show-labels

```
candidate@master01:~$ kubectl -n ckad00018 get pod --show-labels
NAME               READY   STATUS    RESTARTS       AGE   LABELS
ckad00018-dmz      1/1     Running   2 (132m ago)   27h   app=ckad00018-dmz,item=CKAD00018
ckad00018-newpod   1/1     Running   2 (132m ago)   27h   app=ckad00018-newpod,item=CKAD00018
db                 1/1     Running   2 (132m ago)   27h   app=db,item=CKAD00018
front              1/1     Running   2 (132m ago)   27h   app=front,item=CKAD00018
candidate@master01:~$
```

查看所有的 networkpolicy
kubectl -n ckad00018 get networkpolicy

```
candidate@master01:~$ kubectl -n ckad00018 get networkpolicy
NAME           POD-SELECTOR      AGE
access-db      app=db            27h
access-front   app=front         27h
all-access     all-access=true   27h
default-deny   <none>            27h
candidate@master01:~$
```

查看 networkpolicy access-front
kubectl -n ckad00018 describe networkpolicy access-front

```
candidate@master01:~$ kubectl -n ckad00018 describe networkpolicy access-front
Name:         access-front
Namespace:    ckad00018
Created on:   2025-02-15 20:39:17 +0800 CST
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     app=front
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      PodSelector: front-access=true
  Allowing egress traffic:
    To Port: <any> (traffic allowed to all ports)
    To:
      PodSelector: front-access=true
  Policy Types: Ingress, Egress
candidate@master01:~$
```

查看 networkpolicy access-db
kubectl -n ckad00018 describe networkpolicy access-db

```
candidate@master01:~$ kubectl -n ckad00018 describe networkpolicy access-db
Name:         access-db
Namespace:    ckad00018
Created on:   2025-02-15 20:39:17 +0800 CST
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     app=db
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      PodSelector: db-access=true
  Allowing egress traffic:
    To Port: <any> (traffic allowed to all ports)
    To:
      PodSelector: db-access=true
  Policy Types: Ingress, Egress
candidate@master01:~$
```

【3】 给 ckad00018-newpod 打正确的标签

```
kubectl -n ckad00018 label pod ckad00018-newpod front-access=true
kubectl -n ckad00018 label pod ckad00018-newpod db-access=true
```

```
candidate@master01:~$ kubectl -n ckad00018 label pod ckad00018-newpod front-access=true
pod/ckad00018-newpod labeled
candidate@master01:~$
candidate@master01:~$ kubectl -n ckad00018 label pod ckad00018-newpod db-access=true
pod/ckad00018-newpod labeled
candidate@master01:~$
```

【4】 再次检查标签，ckad00018-newpod 有标签了，也就是匹配了 access-front 和 access-db 两条网络策略。

```
kubectl -n ckad00018 get pod --show-labels
```

```
candidate@master01:~$ kubectl -n ckad00018 get pod --show-labels
NAME              READY   STATUS    RESTARTS       AGE   LABELS
ckad00018-dmz     1/1     Running   2 (135m ago)   27h   app=ckad00018-dmz,item=CKAD00018
ckad00018-newpod  1/1     Running   2 (135m ago)   27h   app=ckad00018-newpod,db-access=true,front-access=true,item=CKAD00018
db                1/1     Running   2 (135m ago)   27h   app=db,item=CKAD00018
front             1/1     Running   2 (135m ago)   27h   app=front,item=CKAD00018
candidate@master01:~$
```

【5】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00018 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00018 closed.
candidate@base:~$
```

16、Ingress 排错

（考试的考题内容，只有下面方框里的内容）

您必须连接到正确的主机。不这样做可能导致零分。
[candidate@base] $ ssh ckad00034a

Task
在 namespace ingress-ckad 下，名为 nginx-dm 的 Deploymen 通过 Ingress nginx-ingress-test 公开。
您可以在目录/ckad/CKAD202206 中找到 deploymen、service、ingress 的清单文件。
Deployment 本应可以正常访问，但是目前返回错误。
通过更新相关资源，来识别并修复问题，以便 Deploymen 按计划可以被正常访问。

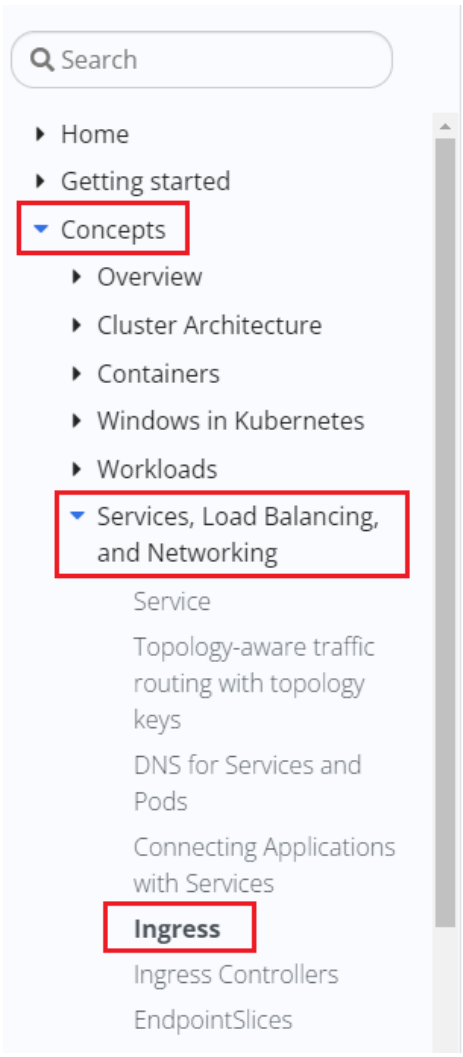
请注意，这道题的 deployment 是正确的，请不要修改 deployment。

请勿修改 Deployment 。 您可以假设 Deployment 是正确且功能正常的。

参考地址

没必要参考网址，建议记住操作方法。
如果非要参考，可以按照下面方法。

依次点击下图，打开最终网址。(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/



Kubernetes Documentation / Concepts / Services, Load Balancing, and Networking / Ingress

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.
- 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.
- Cluster network: A set of links, logical or physical, that facilitate communication within a cluster according to the Kubernetes networking model.
- Service: A Kubernetes Service that identifies a set of Pods using label selectors. Unless mentioned otherwise, Services are assumed to have virtual IPs only routable within the cluster network.

What is Ingress?

Ingress 资源

一个最小的 Ingress 资源示例：

```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
```

Hostname wildcards

Hosts can be precise matches (for example "`foo.bar.com`") or a wildcard (for example "`*.foo.com`"). Precise matches require that the HTTP `host` header matches the `host` field. Wildcard matches require the HTTP `host` header is equal to the suffix of the wildcard rule.

Host	Host header	Match?
`*.foo.com`	`bar.foo.com`	Matches based on shared suffix
`*.foo.com`	`baz.bar.foo.com`	No match, wildcard only covers a single DNS label
`*.foo.com`	`foo.com`	No match, wildcard only covers a single DNS label

service/networking

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-wildcard-host
spec:
  rules:
  - host: "foo.bar.com"
    http:
      paths:
      - pathType: Prefix
        path: "/bar"
        backend:
          service:
            name: service1
            port:
              number: 80
  - host: "*.foo.com"
    http:
      paths:
      - pathType: Prefix
        path: "/foo"
        backend:
          service:
            name: service2
            port:
              number: 80
```

答题

说明:
port 是 service 端口，即 k8s 中服务之间的访问端口
targetport 是 pod（也就是容器）的端口

【1】 按照题目要求，在 base 节点上执行，切换到题目要求节点
考试时务必先按照题目要求，ssh 到对应节点做题。做完后，务必要 exit 退回到 candidate@base 初始节点。

确保在 candidate@base:~$下，再执行切换集群的命令
ssh ckad00034a

```
candidate@base:~$ ssh ckad00034a
```

```
candidate@master01:~$
```

【2】 因为题目告知了 deployment 是正确的，所以我们先看一下 deployment 配置信息。
cat /ckad/CKAD202206/deployment.yaml

内容如下
apiVersion: apps/v1
kind: Deployment
metadata:

```
    name: nginx-dm
    namespace: ingress-ckad
spec:
    replicas: 2
    selector:
        matchLabels:
            name: nginx-ing
    template:
        metadata:
            labels:
                name: nginx-ing          ## 1 ##  记住 Deployment 的 labels 标签，因为题目说明了 Deployment 是正确的。
        spec:
            containers:
            - name: nginx
              image: vicuu/nginx:hello81
              imagePullPolicy: IfNotPresent
              ports:
                - containerPort: 81       ## 2 ##  记住 Deployment 的容器的端口号，因为题目说明了 Deployment 是正确的。
```

【3】 修改 service 配置文件

特别注意红色字体，是错误的地方。其他颜色字体，也要关注一下，确保是对的。
vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入

vi /ckad/CKAD202206/service.yaml

修改内容为下面红色字体

```
apiVersion: v1
kind: Service
metadata:
    name: nginx-ing-svc             ## 3 ##  记住 Service 的 name 名字，下面修改 Ingress 要用到
    namespace: ingress-ckad
spec:
    ports:
    - port: 80                      ## 4 ##  检查一下，确保这个 Service 的 port，和下面的 Ingress 的 port:number 端口一致
      targetPort: 81                ## 2 ##  修改 Service 的 targetPort 端口与上面 Deployment 的 containerPort 端口号一致，即修改为 81。
      protocol: TCP
    selector:
      name: nginx-ing               ## 1 ##  修改 Service 的 selector 为上面 Deployment 的 labels 标签，即修改为 name: nginx-ing，注意是 name 冒号空格 nginx-ing
```

【4】 修改 ingress 配置文件

修改 ingress 配置文件，并重新创建 ingress。
vim 和 kubectl edit 后，要先执行 :set paste 再按 i 插入

vi /ckad/CKAD202206/ingress.yaml

修改内容为下面红色字体

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
    name: nginx-ingress-test
    namespace: ingress-ckad
    annotations:
        nginx.ingress.kubernetes.io/rewrite-target: /
spec:
    ingressClassName: nginx-example123
    rules:
    - http:
          paths:
          - path: /hello
            pathType: Prefix
            backend:
                service:
                  name: nginx-ing-svc          ## 3 ##  修改 Ingress 的 service 为上面 Service 的 name 名字，即修改为  name: nginx-ing-svc
                  port:
                    number: 80                 ## 4 ##  检查一下，确保 Ingress 的 port:number 和 Service 的 port 端口一致
```

【5】 创建 service 和 ingress

kubectl apply -f /ckad/CKAD202206/service.yaml

kubectl apply -f /ckad/CKAD202206/ingress.yaml

```
candidate@master01:~$ vi /ckad/CKAD202206/service.yaml
candidate@master01:~$ vi /ckad/CKAD202206/ingress.yaml
candidate@master01:~$
candidate@master01:~$ kubectl apply -f /ckad/CKAD202206/service.yaml
service/nginx-ing-svc configured
candidate@master01:~$
candidate@master01:~$ kubectl apply -f /ckad/CKAD202206/ingress.yaml
ingress.networking.k8s.io/nginx-ingress-test configured
candidate@master01:~$
```

【6】 检查：

kubectl -n ingress-ckad get pod,svc,ingress -o wide

curl 10.102.84.200:80　　#curl service 的 IP 和端口

curl 10.110.54.196:80/hello　　#curl ingress 的 IP 和端口+地址

```
candidate@master01:~$ kubectl -n ingress-ckad get pod,svc,ingress -o wide
NAME                                READY   STATUS    RESTARTS      AGE   IP              NODE     NOMINATED NODE   READINESS GATES
pod/nginx-dm-5f66878554-bq8k5       1/1     Running   2 (151m ago)  27h   10.244.140.102  node02   <none>           <none>
pod/nginx-dm-5f66878554-wvfql       1/1     Running   2 (151m ago)  27h   10.244.14.122   base     <none>           <none>

NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE   SELECTOR
service/nginx-ing-svc   ClusterIP   10.102.84.200   <none>        80/TCP    27h   name=nginx-ing

NAME                                        CLASS             HOSTS   ADDRESS          PORTS   AGE
ingress.networking.k8s.io/nginx-ingress-test   nginx-example123   *       10.110.54.196    80      27h
candidate@master01:~$
candidate@master01:~$ curl 10.102.84.200:80
Hello World ^_^
Port 81
candidate@master01:~$
candidate@master01:~$ curl 10.110.54.196:80/hello
Hello World ^_^
Port 81
candidate@master01:~$
```

deployment的Pod是81端口

【7】 别忘记，做完后，退回到 base 节点，这样下一道题才能继续切节点。

```
# 退出 ckad00034a 节点，返回到 candidate@base 用户和节点下
exit
```

```
candidate@master01:~$ exit
logout
Connection to ckad00034a closed.
candidate@base:~$
```

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。