
1.1 CKS 考试题型分析

一、AppArmor

12 分的题目

1、AppArmor 简介：AppArmor 是一个高效和易于使用的 Linux 系统安全应用程序。AppArmor 对操作系统和应用程序所受到的威胁进行从内到外的保护，简单的说，AppArmor 是与 SELinux 类似的一个访问控制系统，通过它你可以指定程序可以读、写或运行哪些文件，是否可以打开网络端口等。作为对传统 Unix 的自主访问控制模块的补充，AppArmor 提供了强制访问控制机制，它已经被整合到 2.6 版本的 Linux 内核中。目前 Ubuntu 自带了 Apparmor。官网：https://wiki.ubuntu.com/AppArmor/

2、AppArmor 两种工作模式：enforcement、complain/learning

Enforcement – 在这种模式下，配置文件里列出的限制条件都会得到执行，并且对于违反这些限制条件的程序会进行日志记录。

Complain – 在这种模式下，配置文件里的限制条件不会得到执行，Apparmor 只是对程序的行为进行记录。例如程序可以写一个在配置文件里注明只读的文件，但 Apparmor 不会对程序的行为进行限制，只是进行记录。

3、访问控制与资源限制：

（1）文件系统的访问控制：

Apparmor 可以对某一个文件，或者某一个目录下的文件进行访问控制，包括以下几种访问模式：

r	Read mode
w	Write mode (mutually exclusive to a)
a	Append mode (mutually exclusive to w)
k	File locking mode
l	Link mode

在配置文件中的写法，如

如 /tmp r, (表示可对/tmp 目录下的文件进行读取)

注意：没在配置文件中列出的文件，程序是不能访问的

2）资源限制

Apparmor 可以提供类似系统调用 setrlimit 一样的方式来限制程序可以使用的资源。要限制资源，可在配置文件中这样写：

set rlimit [resource] <= [value],

其 resource 代表某一种资源，value 代表某一个值，要对程序可以使用的虚拟内存做限制时，可以这样写：

set rlimit as<=1M, （可以使用的虚拟内存最大为 1M）

（3）访问网络

Apparmor 可以程序是否可以访问网络进行限制，在配置文件里的语法是：

network [[domain] [type] [protocol]]

要让程序可以进行所有的网络操作，只需在配置文件中写：

network,

要允许程序使用在 IPv4 下使用 TCP 协议，可以这样写：

network inet tcp,

4、配置文件的编写

编写完配置文件后，要把文件放到/etc/apparmor.d 这个目录下

5、模拟环境：考试不需要做，考试环境已经准备好了

（1）在 k8s 工作节点安装 apparmor 命令

apt-get install apparmor-utils apparmor-profiles apparmor-profiles-extra -y

（2）查看当前 AppArmor 的状态

apparmor_status

注意：Apparmor 的 profile 配置文件均保存在目录/etc/apparmor.d，对应的日志文件记录在/var/log/messages。

（3）在工作节点生成/etc/apparmor.d/nginx_apparmor 文件

```
cat /etc/apparmor.d/nginx_apparmor
#include <tunables/global>
profile nginx-profile   flags=(attach_disconnected) {
  #include <abstractions/base>
  file,
  # Deny all file writes.
  deny /** w,
}
```

(4) 创建一个 pod，不加载 apparmor 配置

```
cat /cks/1/pod1.yaml
apiVersion: v1
kind: Pod
```

```
metadata:
  name: hello-apparmor
  annotations:
spec:
  containers:
  - name: hello
    image: busybox
    command: [ "sh", "-c", "echo 'Hello AppArmor!' && sleep 1h" ]
```

（5）考试题解答

KSSH00401 kssh00401-master kssh00401-worker 1

You can switch the cluster/configuration context using the following command:

```
[candidate@cli] $ kubectl config use-context KSSH00401
```

Context

AppArmor is enabled on the cluster's worker node. An AppArmor profile is prepared, but not enforced yet.

You may use your browser to open **one additional tab** to access the AppArmor documentation.

Task

On the cluster's worker node, enforce the prepared AppArmor profile located at

/etc/apparmor.d/nginx_apparmor 。Edit the prepared manifest file located at /cks/1/pod1.yaml to apply the AppArmor profile。Finally, apply the manifest file and create the pod specified in it.

中文翻译：

任务：

在集群的工作节点上，确保准备好的 AppArmor 文件在/etc/AppArmor.d/目录下，文件名字是 nginx_apparmor。编辑位于/cks/1/pod1.yaml 的准备好的清单文件，以应用 AppArmor 配置。最后，应用清单文件并创建 pod 指定清单文件。

题目解析：

/etc/apparmor.d/nginx_apparmor #配置文件考试已经写好了，不需要自己去配置。

$ kubectl get nodes #查看 k8s 集群节点状态

$ ssh 远程到指定工作节点

$ apparmor_parser -q /etc/apparmor.d/nginx_apparmor #加载 apparmor 配置文件

$ apparmor_status #查看策略名称，结果是 nginx-profile

exit 退出工作节点，回到控制节点

$ vim /cks/1/pod1.yaml

```
annotations:
  container.apparmor.security.beta.kubernetes.io/hello: localhost/nginx-profile
```

hello 是 pod 里容器的名称，nginx-profile 是 apparmor_status 解析出来的结果

$ kubectl apply -f /cks/1/pod1.yaml #更新 yaml 文件就可以了。

参考：https://kubernetes.io/zh/docs/tutorials/clusters/apparmor/

https://kubernetes.io/zh/docs/tutorials/clusters/apparmor/#%E4%BF%9D%E6%8A%A4-pod

二、kube-bench 对 k8s 进行基准测试

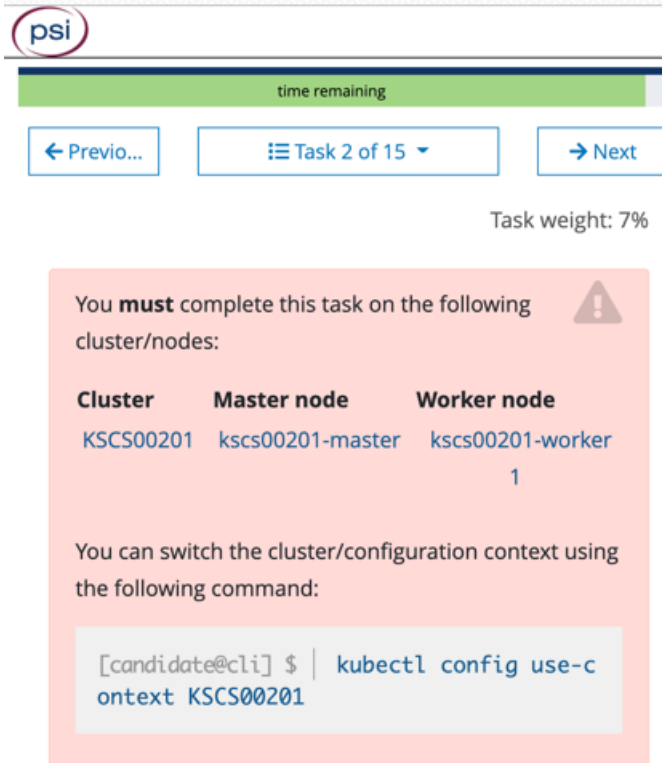
1、Kube-bench 简介：

kube-Bench 是一款针对 Kubernete 的安全检测工具，从本质上来说，Kube-Bench 是一个基于 Go 开发的应用程序，它可以帮助研究人员对部署的 Kubernete 进行安全检测，安全检测原则遵循 CIS Kubernetes Benchmark。

2、安装：

docker run --rm -v `pwd`:/host aquasec/kube-bench:latest install

./kube-bench master #仅对 master 节点检查



context

ACIS Benchmark tool was run against the kubeadm-created cluster and found multiple issues that must be addressed immediately.

Task

Fix all issues via configuration and restart theaffected components to ensure the new settings take effect.

Fix all of the following violations that were found against the API server:

Ensure that the 1.2.7 --authorization-mode FAIL argument is not set to AlwaysAllow

Ensure that the 1.2.8 --authorization-mode FAIL argument includes Node

Ensure that the 1.2.9 --authorization-mode FAIL argument includes RBAC

Ensure that the 1.2.18 --insecure-bind-address FAIL argument is not set

Ensure that the 1.2.19 --insecure-port FAIL argument is set to 0

Fix all of the following violations that were found against the kubelet:

Ensure that the 4.2.1 anonymous-auth FAIL argument is set to false

Ensure that the 4.2.2 --authorization-mode FAIL argument is not set to AlwaysAllow

Use webhook authn/authz where possible.

Fix all of the following violations that were found against etcd:

Ensure that the 4.2.1 --client-cert-auth FAIL argument is set to true

---

中文翻译：

背景：
针对 kubeadm 创建的集群运行 ACIS 基准测试工具，发现了多个必须立即解决的问题。

任务：
通过配置修复所有问题，并重新启动受影响的组件，以确保新设置生效。

修复针对 APIserver 发现的以下所有问题：
确保 1.2.7- -authorization-mode 参数不能设置为 AlwaysAllow
确保 1.2.8 --authorization mode 参数包含 Node
确保 1.2.9--authorization mode 参数包含 RBAC
确保 1.2.19 --insecure-port 参数设置为 0

修复针对 kubelet 发现的以下所有 wentire：
确保 4.2.1 anonymous-auth 参数设置为 false
确保 4.2.2 -authorization-mode 参数不能设置为 AlwaysAllow
尽可能使用 webhook authn/authz。

修复针对 etcd 发现的以下所有问题：
确保 4.2.--client-cert-auth 失败参数设置为 true

解析
```
$ ssh root@kscs00201-master
$ vim /etc/kubernetes/manifests/kube-apiserver.yaml
#- --authorization-mode=AlwaysAllow
- --authorization-mode=Node,RBAC
#- --insecure-bind-address=0.0.0.0
- --insecure-port=0

$ vim /var/lib/kubelet/config.yaml  #master 和 node 节点都需要改配置
anonymous:
    enabled: false
authorization:
    mode: Webhook

$ vim /etc/kubernetes/manifests/etcd.yaml
- --client-cert-auth=true

$ systemctl daemon-reload
$ systemctl restart kubelet
```

三、Trivy 进行镜像扫描

Trivy 是一种适用于 CI 的简单而全面的容器漏洞扫描程序。软件漏洞是指软件或操作系统中存在的故障、缺陷或弱点。Trivy 检测操作系统包（Alpine、RHEL、CentOS 等）和应用程序依赖（Bundler、Composer、npm、yarn 等）的漏洞。Trivy 很容易使用，只要安装二进制文件，就可以扫描了。扫描只需指定容器的镜像名称。与其他镜像扫描工具相比，例如 Clair，Anchore Engine，Quay 相比，Trivy 在准确性、方便性和对 CI 的支持等方面都有着明显的优势。

推荐在 CI 中使用它，在推送到 container registry 之前，你可以轻松地扫描本地容器镜像，Trivy 具备如下的特征：

1、检测面很全，能检测全面的漏洞，操作系统软件包（Alpine、Red Hat Universal Base Image、Red Hat Enterprise Linux、CentOS、Oracle Linux、Debian、Ubuntu、Amazon Linux、openSUSE Leap、SUSE Enterprise Linux、Photon OS 和 Distrioless)、应用程序依赖项（Bundler、Composer、Pipenv、Poetry、npm、yarn 和 Cargo)；

2、使用简单，仅仅只需要指定镜像名称；

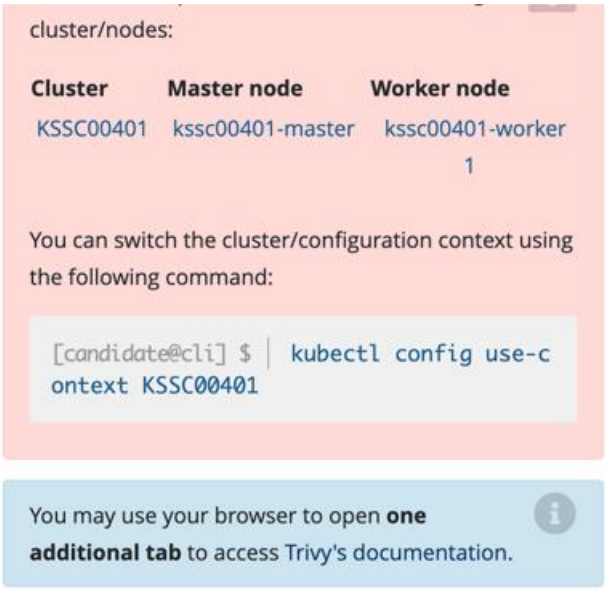
3、扫描快且无状态，第一次扫描将在 10 秒内完成（取决于网络)。随后的扫描将在一秒钟内完成。与其他扫描器在第一次运行时需要很长时间（大约 10 分钟）来获取漏洞信息，并鼓励你维护持久的漏洞数据库不同，Trivy 是无状态的，不需要维护或准备；

4、易于安装测试：

```
docker run   -v /var/run/docker.sock:/var/run/docker.sock -v
$HOME/Library/Caches:/root/.cache/ aquasec/trivy:0.20.2
registry.aliyuncs.com/google_containers/coredns:1.7.0 | grep -i "high"
```

备注：registry.aliyuncs.com/google_containers/coredns:1.7.0 表示要扫描的镜像

解题：



Task

Use the Trivy open-source container scanner to detect images with severe vulnerabilities used by Pods in the namespace yavin.

Look for images with High or Critical severity vulnerabilities,and delete the Pods that use those images.

Trivy is pre-installed on the cluster's master node only; it is not available on the base system or the worker nodes. You'll have to connect to the cluster's master node to use Trivy.

翻译：

使用 Trivy 开源容器扫描器检测命名空间 yavin 中 POD 使用的具有严重漏洞的镜像。

查找具有 High 或 Critical 漏洞的镜像，并删除使用这些镜像的 POD。

Trivy 仅预装在集群的主节点上；它在基本系统或工作节点上不可用。必须连接到集群的主节点才能使用 Trivy。

解析：

$ ssh root@kssc00401-master

$ kubectl get po -n yavin -oyaml | grep ‘image’

$ trivy image --skip-update ‘刚才搜到的镜像’ | egrep -i "High|Critical"

$ 把检测出来漏洞的镜像对应的 pod 删除

#--skip-update #跳过镜像更新，考试可以直接跳过。

四、sysdig & falco

Sysdig 官网：www.sysdig.org

sysdig 的定位是系统监控、分析和排障的工具，在 linux 平台上，已经有很多这方面的工具 tcpdump、htop、iftop、lsof、netstat，它们都能用来分析 linux 系统的运行情况，而且还有很多日志、监控工具。为什么还需要 sysdig 呢？ sysdig 的优点可以归纳为三个词语：整合、强大、灵活。

Falco 是一个云原生运行时安全系统，可与容器和原始 Linux 主机一起使用。它由 Sysdig 开发，是 Cloud Native Computing Foundation（云原生计算基金会）的一个沙箱项目。Falco 的工作方式是查看文件更改、网络活动、进程表和其他数据是否存在可疑行为，然后通过可插拔后端发送警报。通过内核模块或扩展的 BPF 探测器在主机的系统调用级别检查事件。Falco 包含一组丰富的规则，您可以编辑这些规则以标记特定的异常行为，并为正常的计算机操作创建允许列表。

`You may use your browser to open one additional tab to access sysdig's documentation or Falco's documentation.`

Task:
Use runtime detection tools to detect anomalous processes spawning and executing frequently in the single container belorging to Pod **redis**.

Two tools are available to use:

- [x] sysdig
- [x] falco

`The tools are pre-installed on the cluster's worker node only; they are not available on the base system or the master node.`

Using the tool of your choice (including any non pre-installed tool), analyse the container's behaviour for at least 30 seconds, using filters that detect newly spawning and executing processes.

Store an incident file at /opt/2/report , containing the detected incidents, one per line, in the following format:

`[timestamp],[uid], [processName]`

`Keep the tool's original timestamp-format as-is.`

`Make sure to store the incident fileon the cluster's worker node.`

You may use your browser to open one additional tab to access sysdig's documentation or Falco's documentation.

Task:

Use runtime detection tools to detect anomalous processes spawning and executing frequently inthe single container belorging to Pod redis.

Two tools are available to use:

sysdig

falco

The tools are pre-installed on the cluster's worker node only; they are not available on the base system or the master node.

Using the tool of your choice (including any non pre-installed tool), analyse the container's behaviour for at least 30 seconds, using filters that detect newly spawning and executing processes.

Store an incident file at /opt/2/report , containing the detected incidents, one per line, in the following format:

[timestamp],[uid], [processName]

Keep the tool's original timestamp-format as-is.
Make sure to store the incident fileon the cluster's worker node.

中文翻译：
你可以使用浏览器打开另一个标签来访问 sysdig 的文档或 Falco 的文档。

任务：
使用运行时检测工具检测 redis 这个 pod 下的单个容器中反常的和频繁发生异常的进程。

有两种工具可供使用：
sysdig
falco

这些工具仅预安装在集群的工作节点上。

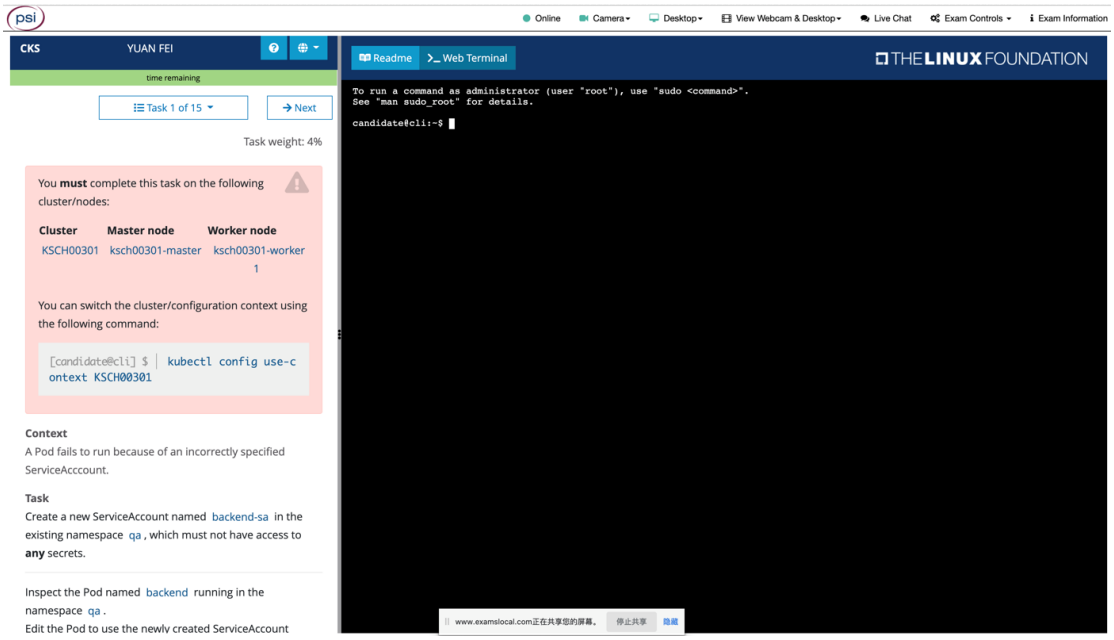
使用你选择的工具（包括任何未预装的工具），至少分析容器 30s，使用过滤器检查最新的异常进程，
将事件文件存储在/opt/2/report 中，其中包含检测到的事件，每行一个事件
按照以下格式保存：

[timestamp],[uid], [processName]

保持工具的原始时间戳格式不变。
确保将事件文件存储在群集的工作节点上。

解析
```
$ ssh root@vms62.rhce.cc
$ docker ps | grep redis
$ sysdig -l | grep time
$ sysdig -l | grep uid
$ sysdig -l | grep proc
$ sysdig -M 30 -p "*%evt.time,%user.uid,%proc.name" container.id=b1dacef30135 > /opt/2/report

#-p：指定打印事件时使用的格式
#-M: 多少秒后停止收集
# evt.time: 事件发生的时间
# user.uid: 用户 uid
# proc.name: 生成事件的进程名字
# container.id: 根据容器过滤
```

五、serviceAccount



context

A Pod fails to run because of an incorrectly specified ServiceAcccount.

Task

create a new ServiceAccount named backend-sa in the existing namespace qa ,which must not have access to any secrets。Inspect the Pod named backend running in the namespace qa。Edit the Pod to use the newly created serviceAccount backend-sa。You can find the Pod's manifest file at /cks/9/pod9.yaml。Ensure that the modified specification is applied and the Pod is running。Finally, clean-up and delete the now unused serviceAccount that the Pod used initially.

参考官网：https://kubernetes.io/zh/docs/tasks/configure-pod-container/configure-service-account/

中文翻译：

背景：

Pod 无法运行，因为指定的 ServiceAcccount 不正确。

任务：

在现有命名空间 qa 中创建一个名为 backend-sa 的新 ServiceAccount，该帐户不能访问任何 secrets 凭据资源。检查命名空间 qa 中运行的名为 backend 的 Pod。编辑 Pod 使用新创建的 backend-sa 这个 serviceAccount 。你可以在/cks/9/pod9.yaml 上找到 Pod 的清单文件。确保修改之后的 Pod 正在运行。最后，清理并删除未被 Pod 使用的 serviceAccount。

解析

```
$ kubectl create sa backend-sa -n qa --dry-run=client -o yaml > sa.yaml
$ vim sa.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: backend-sa
  namespace: qa
automountServiceAccountToken: false   #禁止访问任何凭据

$ kubectl apply -f sa.yaml

#考试会自动有 backend 的 pod，不需要自己创建
$ vim /cks/9/pod9.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: backend
  name: backend
  namespace: qa
spec:
  serviceAccountName: backend-sa
  containers:
  - image: nginx:1.9
    name: backend
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always

$kubectl apply -f /cks/9/pod9.yaml

可以看看 qa 名称空间有哪些 sa，把除了 backend-sa 的 sa 都删除
kubectl delete sa sa 名字  -n qa
```

六、 Pod 安全策略-PodSecurityPolicy

PSP（Pod Security Policy）：Pod 安全策略，是集群级别的资源，Pod Security Policy 是 Kubernetes 的重要安全措施之一，它首先定义角色，其中规定了对 Pod 行为的限制，其中包括对特权容器、主机网络、Capability、加载卷类型等内容进行了限制，然后通过 RBAC 把 SA-Pod-PSP 三者结合起来，完成对 Pod 权限的限制。注意：PSP 在 1.21 废弃，1.25 版本删除。

Pod 安全策略由设置和策略组成，它们能够控制 Pod 访问的安全特征。这些设置分为如下三类：

基于布尔值控制： 这种类型的字段默认为最严格限制的值。

基于被允许的值集合控制： 这种类型的字段会与这组值进行对比，以确认值被允许。

基于策略控制： 设置项通过一种策略提供的机制来生成该值，这种机制能够确保指定的值落在被允许的这组值中。

1、RunAsUser

1）MustRunAs - 必须配置一个 range。使用该范围内的第一个值作为默认值。验证是否不在配置的该范围内。

2）MustRunAsNonRoot - 要求提交的 Pod 具有非零 runAsUser 值，或在镜像中定义了 USER 环境变量。不提供默认值。

3）RunAsAny - 没有提供默认值。允许指定任何 runAsUser 。

2、SELinux

1）MustRunAs - 如果没有使用预分配的值，必须配置 seLinuxOptions。默认使用 seLinuxOptions。验证 seLinuxOptions。

2）RunAsAny - 没有提供默认值。允许任意指定的 seLinuxOptions ID。

3、SupplementalGroups

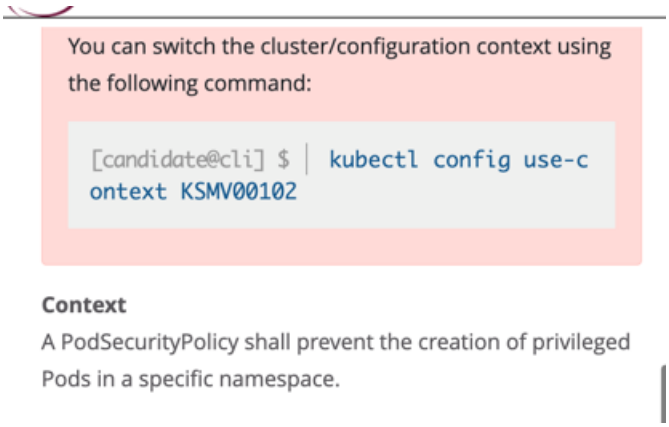
1）MustRunAs - 至少需要指定一个范围。默认使用第一个范围的最小值。验证所有范围的值。

2）RunAsAny - 没有提供默认值。允许任意指定的 supplementalGroups ID。

4、FSGroup

1）MustRunAs - 至少需要指定一个范围。默认使用第一个范围的最小值。验证在第一个范围内的第一个 ID。

2）RunAsAny - 没有提供默认值。允许任意指定的 fsGroup ID。



Context：

A PodsecurityPolicy shall prevent the create on of privileged Pods in a specific namespace.

Task:

Create a new PodSecurityPolicy named prevent-psp-policy, which prevents the creation of privileged Pods.

Create a new ClusterRole named restrict-access-role, which uses the newly created PodSecurityPolicy prevent-psp-policy.

Create a new serviceAccount named psp-denial-sa in the existing namespace development.

Finally, create a new clusterRoleBinding named dany-access-bind, which binds the newly created ClusterRole restrict-access-role to the newly created serviceAccount psp-denial-sa.

中文翻译:

背景:

PodsecurityPolicy 应防止在特定命名空间中创建特权 Pod。

任务:

创建一个名为 prevent-psp-policy 的新 PodSecurityPolicy，它阻止创建特权 POD。

创建一个名为 restrict-access-role 的新 ClusterRole，它对新创建的 PodSecurityPolicy prevent-psp-policy 具有 users 权限

在现有命名空间 development 中创建名为 psp-denial-sa 的新 serviceAccount。

最后，创建一个名为 dany-access-bind 的新 clusterRoleBinding，它将新创建的 psp-denial-sa 这个 sa 绑定到新创建的 restrict-access-role 这个 clusterrole 上

参考：https://kubernetes.io/zh/docs/concepts/policy/pod-security-policy/

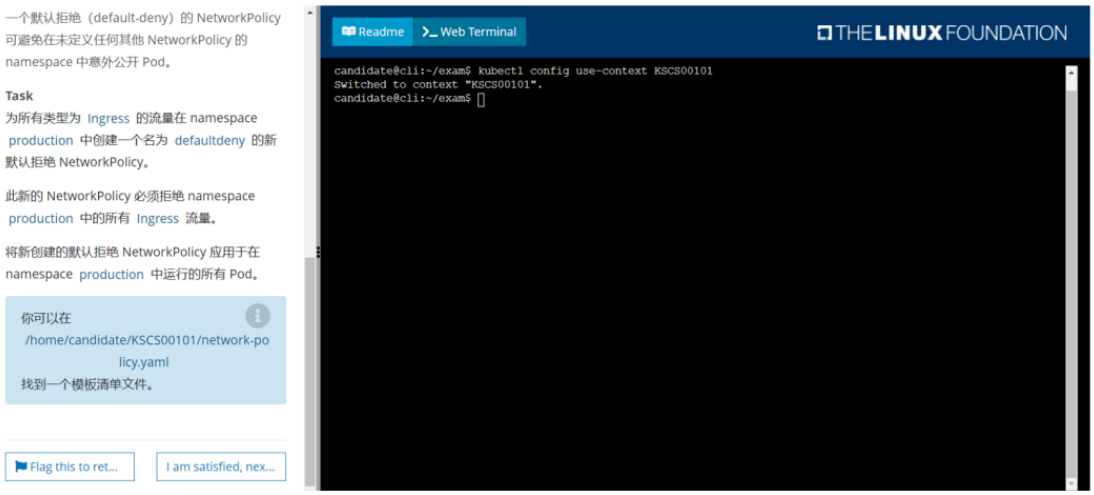
https://kubernetes.io/docs/reference/access-authn-authz/rbac/#command-line-utilities

```
$ vim /etc/kubernetes/manifests/kube-apiserver.yaml
        - --enable-admission-plugins=NodeRestriction,PodSecurityPolicy
```

```
$ systemctl restart kubelet

$ vim 63-psp.yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
    name: prevent-psp-policy
spec:
  privileged: false #禁止容器的特权模式
  seLinux:
    rule: RunAsAny   #设置 selinux 参数，不会校验
  supplementalGroups:
    rule: RunAsAny
  runAsUser:
    rule: RunAsAny   #设置运行容器的用户 ID
  fsGroup:
    rule: RunAsAny #设置组 ID
  volumes:
  - '*'    #可以用任何 volume 卷

$ kubectl apply -f 63-psp.yaml

$ kubectl create clusterrole restrict-access-role --verb=use --resource=psp -- resource-name=prevent-psp-policy


$ kubectl create sa psp-denial-sa -n development

$ kubectl create clusterrolebinding dany-access-bind --clusterrole=restrict-access-role   --serviceaccount=development:psp-denial-sa -n development
```

七、NetworkPolicy 拒绝所有入口流量

、



context

A default-deny NetworkPolicy avoids to accidentally expose a Pod in a namespace that doesn't have any other NetworkPolicy defined.

Task

Create a new default-deny NetworkPolicy named denynetwork in the namespace development for all traffic of type Ingress.

The new NetworkPolicy must deny all Ingress traffic in the namespace development.

Apply the newly created default-deny NetworkPolicy to all Pods running in namespace development.

You can find a skeleton manifest file at /cks/15/p1.yaml

中文翻译：

背景：

默认的拒绝 NetworkPolicy 可避免在未定义任何其他 NetworkPolicy 的命名空间中意外暴漏 Pod。

accidentally：[ˌæksɪˈdɛntəli]，意外的

任务

在命名空间 development 中为所有入口类型的流量创建一个名为 denynetwork 的新默认拒绝网络策略。

新的网络策略必须拒绝命名空间 development 中的所有 Ingress 通信。

将新创建的默认拒绝网络策略应用于命名空间 development 中运行的所有 POD。

您可以在/cks/15/p1.yaml 中找到骨架清单文件

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

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

答案：

```
$ vim /cks/15/p1.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: "denynetwork"
    namespace: "development"
spec:
    podSelector: {}
    policyTypes:
    - Ingress    #拒绝所有入栈流量

$ kubectl apply -f /cks/15/p1.yaml
```

八、NetworkPolicy

Cluster	Master node	Worker node
KSSH00301	kssh00301-master	kssh00301-worker 1

You can switch the cluster/configuration context using the following command:

```
[candidate@cli] $ kubectl config use-context KSSH00301
```

Task

create a NetworkPolicy named pod-access torestrict access to Pod products-service running in namespace development.

only allow the following Pods to connect to Pod products-service :

Pods in the namespace testing

Pods with label environment: staging, in any namespace

Make sure to apply the NetworkPolicy.

You can find a skelet on manifest file at /cks/6/p1.yaml

中文翻译：

任务

创建名为 pod-access 的网络策略，用来限制 development 名称空间的 products-service 这个 pod。

仅允许以下 Pod 连接到 products-service 这个 pod：

在 testing 这个名称空间下的 pod

任何名称空间下带 environment: staging 这个标签的 pod

确保创建网络策略。

可以在清单文件/cks/6/p1.yaml 中找到骨架清单文件

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

https://kubernetes.io/zh/docs/concepts/services-networking/network-policies/#networkpolicy-resource

答案：

#首先查看 testing 名称空间具有的标签

[root@xianchaomaster1 ~]# kubectl get ns testing --show-labels
显示如下:
NAME STATUS AGE LABELS
testing Active 12h name=testing

#查看 products-service 这个 pod 具有的标签
#kubectl get pods products-service -n devlopment --show-lables
NAME READY LABELS
products-service 1/1 run=products-service

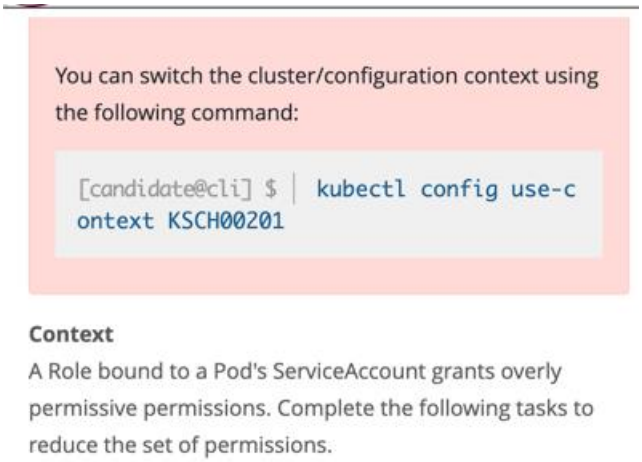
```
$ vim /cks/6/p1.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
    name: "pod-access"
    namespace: "development"
spec:
    podSelector:
      matchLabels:
        run: products-service
    policyTypes:
    - Ingress
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
              name: testing #testing 名称具有的标签
    - from:
      - podSelector:
        matchLabels:
          environment: staging #具有 environment=staging 标签的 pod
        namespaceSelector: {} #所有名称空间
```

$ kubectl apply -f /cks/6/p1.yaml

九、RBAC/clusterrole

参考：https://kubernetes.io/docs/reference/access-authn-authz/rbac/#clusterrole-example

https://kubernetes.io/docs/reference/access-authn-authz/rbac/#kubectl-create-role



context

A Role bound to a Pod's serviceAccount grants overly permissive permissions.

Complete the following tasks to reduce the set of permissions.

Task

Given an existing Pod named web-pod running in the namespace monitoring. Edit the existing Role bound to the Pod's serviceAccount sa -1 to only allow performing list operations, only on resources of type Endpoints.

create a new Role named role-2 in the namespace monitoring, which only allows performing，update operations, only on resources of type persistentvolumeclaims.

create a new RoleBinding named role-2-binding binding the newly created Role to the Pod's serviceAccount.

Don't delete the existing RoleBinding.

中文翻译：

背景：

绑定到 Pod 的 serviceAccount 的角色授予过度权限。完成以下任务以降低 sa 权限。

任务：

给定的名称空间 monitoring 中已经存在一个名字是 web-pod 的 Pod。在 pod 上有个 sa 是 sa-1，这个 sa 绑定到了已经存在的 role 上，仅允许对 Endpoints 类型的资源执行 list 操作。

在命名空间 monitoring 中创建名为 role-2 的新角色，该角色仅允许执行，仅在 persistentvolumeclaims 类型的资源上更新操作。

创建一个名为 role-2-binding 的新 RoleBinding，将新创建的角色绑定到 Pod 的 serviceAccount。

不要删除现有角色绑定。

解析

```
$ kubectl get po -n monitoring web-pod -oyaml | grep serviceAccountName
    #显示的结果：serviceAccountName: sa-1

$ kubectl get rolebinding -n monitoring -o yaml | grep sa-1 -B 10
显示如下：
    apiGroup: rbac.authorization.k8s.io
    kind: Role
    name: role-1
 subjects:
 - kind: ServiceAccount
   name: sa-1

$ kubectl get role
如果看到 role 是 role-1，可继续
$ kubectl edit role -n monitoring role-1
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
    name: role-1
    namespace: monitoring
    resourceVersion: "9528"
rules:
- apiGroups:
  - ""
  resources:
  - endpoints
  verbs:
  - list

$ kubectl create role role-2 -n monitoring --verb=update --
resource=persistentvolumeclaims

$ kubectl create rolebinding role-2-binding --role=role-2 --
serviceaccount=monitoring:sa-1 -n monitoring
```

十、kube-apiserver 审计日志记录和采集

概念：

Kubernetes 审计（Auditing） 功能提供了与安全相关的、按时间顺序排列的记录集， 记录每个用户、使用 Kubernetes API 的应用以及控制面自身引发的活动。

审计功能使得集群管理员能够回答以下问题：

发生了什么?

什么时候发生的?

谁触发的?

活动发生在哪个（些）对象上?

在哪观察到的?

它从哪触发的?

活动的后续处理行为是什么?

Kube-apiserver 执行审计。每个执行阶段的每个请求都会生成一个事件，然后根据特定策略对事件进行预处理并写入后端。

每个请求都可以用相关的 “stage” 记录。已知的 stage 有：

RequestReceived – 此阶段事件将在审计处理器接收到请求后，并且在委托给其余处理器之前生成。

ResponseStarted - 在响应消息的头部发送后，但是响应消息体发送前。这个 stage 仅为长时间运行的请求生成（例如 watch）。

ResponseComplete - 当响应消息体完成并且没有更多数据需要传输的时候。

Panic - 当 panic 发生时生成。

注意：审计日志记录功能会增加 API server 的内存消耗，因为需要为每个请求存储审计所需的某些上下文。此外，内存消耗取决于审计日志记录的配置。

审计策略：

审计政策定义了关于应记录哪些事件以及应包含哪些数据的规则。处理事件时，将按顺序与规则列表进行比较。第一个匹配规则设置事件的 [审计级别][auditing-level]。已知的审计级别有：

None -符合这条规则的日志将不会记录。

Metadata -记录请求的 metadata（请求的用户、timestamp、resource、verb 等等），但是不记录请求或者响应的消息体。

Request -记录事件的 metadata 和请求的消息体，但是不记录响应的消息体。这不适用于非资源类型的请求。

RequestResponse -记录事件的 metadata，请求和响应的消息体。这不适用于非资源类型的请求。

可以使用 --audit-policy-file 标志将包含策略的文件传递给 kube-apiserver。如果不设置该标志，则不记录事件。注意 rules 字段必须在审计策略文件中提供。

参考：https://kubernetes.io/zh/docs/tasks/debug-application-cluster/audit/

考题解析：

Task

Enable audit logs in the cluster.

To do so, enable the log backend, and ensurethat:

1. logs are stored at /var/log/kubernetes/audit-logs.txt
2. log files are retained for 5 days
3. at maximum, a number of 10 auditlog files are retained

A basic policy is provided at /etc/kubernetes/logpolicy/sample-policy.yaml. g. it only specifies what not to do。

The base policy is located on the cluster's master node.

Edit and extend the basic policy to log:

1. namespaces changes at RequestResponse level
2. the request body of pods changes in the namespace front-apps
3. configMap and secret changes in all namespaces at the Metadata level

Also, add a catch all rule to log all otherrequests at the Metadata level.

Don't forget to apply the modifiedpolicy.

中文翻译：

任务

在群集中启用审计日志。

为此，请启用日志后端，并确保：

1.日志存储在/var/log/kubernetes/audit-logs.txt 中

2.日志文件保留 5 天

3.最多保留 10 个审计日志文件

基本策略在/etc/kubernetes/logpolicy/sample-policy.yaml 中提供。它只指定不记录的日志。

基本策略位于群集的主节点上。

编辑并扩展要记录的基本策略：

1. RequestResponse 级别记录 namespace 的更改
2. Request 级别审计只记录在名称空间 front-apps 中的 pod 更改
3. Metadata 级别记录所有名称空间中的 configMap 和 secret

另外，添加一个全方位的规则来记录 Metadata 级别的所有其他请求。

不要忘记应用修改后的策略。

参考：https://kubernetes.io/zh/docs/tasks/debug-application-cluster/audit/

解析

```
$ vim /etc/kubernetes/logpolicy/sample-policy.yaml
apiVersion: audit.k8s.io/v1   # This is required.
kind: Policy
# Don't generate audit events for all requests in RequestReceived stage.
#不要为 RequestReceived 阶段中的所有请求生成审计事件。
omitStages: #省略阶段
    - "RequestReceived"
rules:
    - level: RequestResponse
      resources:
      - group: ""
        resources: ["namespaces"]
    - level: Request
      resources:
      - group: ""
        resources: ["pods"]
      namespaces: ["front-apps"]
    - level: Metadata
      resources:
      - group: ""
        resources: ["secrets", "configmaps"]
    - level: Metadata
      omitStages:
        - "RequestReceived"

$ vim /etc/kubernetes/manifests/kube-apiserver.yaml
  - --audit-policy-file=/etc/kubernetes/logpolicy/sample-policy.yaml #审计策略文件
  - --audit-log-path=/var/log/kubernetes/audit-logs.txt   #审计日志文件
  - --audit-log-maxage=5    #日志最大保留天数
  - --audit-log-maxbackup=10   #审计日志文件最大保留数据

在 APIServer 的 Pod 上挂载策略文件和日志文件（考试可能会直接挂载好）：
  -   mountPath: /var/log/kubernetes
      name: kubernetes-logs
  -   mountPath: /etc/kubernetes/logpolicy
      name: kubernetes-policy
volumes:
```

```
  - hostPath:
      path: /etc/kubernetes/logpolicy
    name: kubernetes-policy
  - hostPath:
      path: /var/log/kubernetes
    name: kubernetes-logs
```

```
$systemctl daemon-reload
$ systemctl restart kubelet
```

十一、 创建 Secret

Task

Retrieve the content of the existing secret named db1-secret-test in the istio-system namespace. store the username field in a file named /cks/11/old-username.txt , and the password field in a file named /cks/11/old-pass.txt.

You must create both files; they don't exist yet.

Do not use /modify the created files in!the following steps, create new temporaryfiles if needed.

Create a new secret named test in the istio-system namespace, with the following content:

username : thanos
password : hahahaha

Finally, create a new Pod that has access to the secret test via a volume:

pod name | dev-pod
namespace | istio-system
container name | dev-container
image | nginx:1.9
volume name | dev-volume
mount path | /etc/test-secret

中文翻译：

任务

检索 istio-system 名称空间中已经存在的 Secret ，名字是 db1-secret-test。将用户名字段存储在名为/cks/11/old-username.txt 的文件中，将密码字段存储在名为/cks/11/old-pass.txt 的文件中。

你必须创建这两个文件；它们还不存在。

不要使用/modify 创建的文件，按照以下步骤，如果需要，创建新的临时文件。

在 istio-system 名称空间中创建一个名为 test 的 secret，如下所示，最后，创建一个新的 Pod，该 Pod 可以通过卷挂载 test 这个 secret：

pod name | dev-pod
namespace | stio-system
container name | dev-container
image | nginx:1.9
volume name | dev-volume
mount path | /etc/test-secret

解析:

参考：https://kubernetes.io/docs/concepts/configuration/secret/

https://kubernetes.io/docs/concepts/configuration/secret/#using-secrets-as-files-from-a-pod

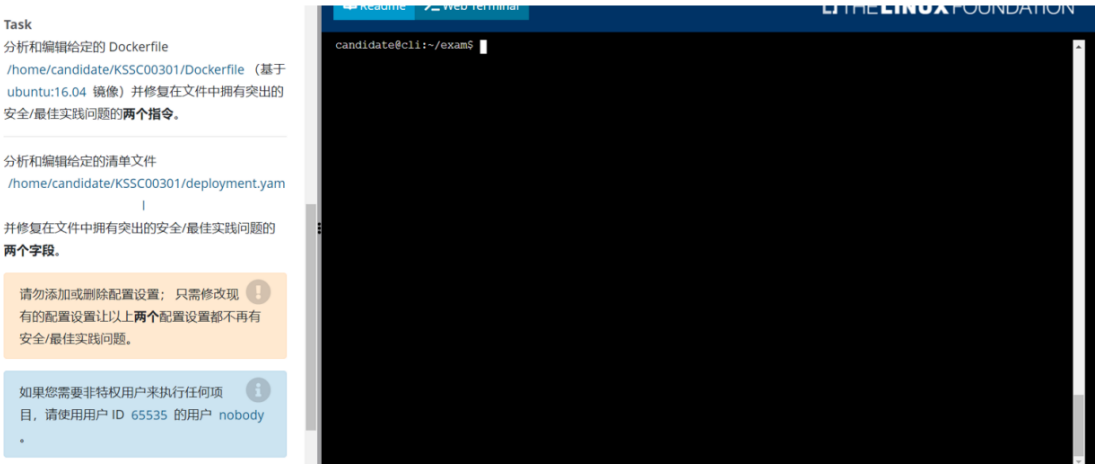
https://kubernetes.io/docs/concepts/configuration/secret/#use-case-pods-with-prod-test-credentials

```
$ kubectl get secrets -n istio-system db1-secret-test -o jsonpath={.data.username} |
base64 -d > /cks/11/old-username.txt
$ kubectl get secrets -n istio-system db1-secret-test -o jsonpath={.data.password} |
base64 -d > /cks/11/old-pass.txt
$ kubectl create secret generic test   -n istio-system --from-
literal=username=thanos --from-literal=password=hahahaha
$ vim k8s-secret.yaml
apiVersion: v1
kind: Pod
metadata:
    name: dev-pod
spec:
    containers:
    - name: dev-container
      image: nginx:1.9
      volumeMounts:
      - name: dev-volume
        mountPath: "/etc/test-secret"
        readOnly: true
    volumes:
```

```
    - name: dev-volume
    secret:
      secretName: test
```

$ kubectl apply -f k8s-secret.yaml
$kubectl get pods dev-pod

十二、dockerfile 和 deployment 优化部分



Task

Analyze and edit the given Dockerfile (based on the ubuntu:16.04 image) /cks/7/Dockerfile fixing two instructions present in the file being prominent security/best-practice issues.

Analyze and edit the given manifest file /cks/7/deployment.yaml fixing two fields present in the file being prominent security/best-practice issues.

中文翻译：

任务

分析并编辑给定的 Dockerfile（基于 ubuntu:16.0 镜像）文件/cks/7/Dockerfile，修复文件中存在的两个安全/最佳实践指令。

分析和编辑给定清单文件/cks/7/deployment.yaml，修复文件中存在的两个安全/最佳实践问题字段。

解析

```
$ vim /cks/7/Dockerfile
FROM ubuntu:16.04
#USER root
USER nobody
#把 USER ROOT 变成 USER nobody

$ vim /cks/7/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 labels:
 app: dev
 name: dev
spec:
 replicas: 1
 selector:
 matchLabels:
 app: dev
 template:
   metadata:
   labels:
     app: dev
   spec:
     containers:
     - image: mysql
       name: mysql
       securityContext: {'capabilities':{'add':['NET_ADMIN'],'drop':['all']},'privileged':
False,'readOnlyRootFilesystem': True, 'runAsUser': 65535}

#'privileged'变成 false，readOnlyRootFilesystem'变成 True
```

十三、镜像安全 ImagePolicyWebhook

Task

You have to complete the entire task on the cluster's master node, where all services and files have been prepared and placed.

Given an incomplete configuration in directory /etc/kubernetes/config and a functional container

---

image scanner with HTTPS endpoint https://localhost:8081/image_policy:
1. Enable the necessary plugins to create an image policy
2. validate the control configuration and change it to an implicit deny
3. Edit the configuration to point t the provided HTTPS endpoint correctly.

Finally , test if the configuration is working by trying to deploy the vulnerable resource /cks/1/web1.yaml
You can find the container image scanner's log file at /var/loglimagepolicyiacme.log

中文翻译：
任务：

必须在集群的主节点上完成整个任务，所有服务和文件都已提前准备好并放置在主节点上。

在/etc/kubernetes/config 目录下有不完整的配置

扫描具有 HTTPS 端点的 http://localhost:8081/image_policy 镜像：

1.启用必要的插件以创建镜像策略

2.验证控制平面的配置并将其更改为拒绝

3.编辑配置以正确指向提供的 HTTPS 端点。

最后，通过尝试部署易受攻击的资源/cks/1/web1.yaml 来测试配置是否有效

可以在以下位置找到容器镜像扫描的日志文件: /var/loglimagepolicyiacme.log

解析
参考：https://kubernetes.io/zh/docs/reference/access-authn-authz/admission-controllers/#imagepolicywebhook

https://kubernetes.io/zh/docs/reference/access-authn-authz/admission-controllers/#imagepolicywebhook

```
$ cd /etc/kubernetes/config
$ vim admission_configuration.json
    defaultAllow: false
```

#把 defaultAllow 由 true 改成 false，关闭默认允许

```
$ vim kubeconfig.yaml
    server : https://localhost:8081/image_policy   #要查询的远程服务的 url，必须是 https

$ vim /etc/kubernetes/manifests/kube-apiserver.yaml
- --enable-admission-plugins=NodeRestriction,ImagePolicyWebhook
- --admission-control-config-
file=/etc/kubernetes/config/admission_configuration.json
  ……
volumeMounts:
- mountPath: /etc/kubernetes/config
  name: config
volumes:
- hostPath:
    path: /etc/kubernetes/config
  name: config
$ systemctl restart kubelet
$ kubectl apply -f /cks/1/web1.yaml
```

十四、删除非无状态或非不可变的 pod

context
It is best-practice to design containers to best stateless and immutable.
Task
Inspect Pods running in namespace qa and delete any Pod that is either not stateless or not immutable.
use the following strict interpretation of stateless and immutable:
Pods being able to store data inside containers must be treated as not stateless.
You don't have to worry whether data is actually stored inside containers or not already.
Pods being configured to be privileged in any way must be treated as potentially not stateless and not immutable.

中文翻译：
背景：
最佳实践是将容器设计为无状态和不可变的。
任务：
在命名空间 dev 中检查 running 状态的 pod，测试并删除任何非无状态或非不可变的 Pod。

无状态和不可变的严格解释参考如下：

1、能够在容器中存储数据的 pod 必须被视为非无状态的（非无状态就是有状态）。
2、以任何方式配置为特权的 POD 必须被视为潜在的非无状态或非不可变的。

解析
1. get 所有 pod
2. 查看是否有特权 privi*
3. 查看是否有 volume
4. 把有特权或者 volume 的 pod 都删除

```
$ kubectl get po -n qa #查看 qa 名称空间下的所有 pod
NAME                READY    STATUS      RESTARTS     AGE
qa-pod              1/1       Running     1            19h
qa-pod1             1/1       Running     1            19h
$ kubectl get po -n qa qa-pod   -o yaml | egrep "priv.*: true"
#查看具有特权的 pod
# privileged: true
$ kubectl delete po -n qa qa-pod --force   --grace-period=0
#查看具有 volume 的 pod
$ kubectl get po -n qa qa-pod1 -o jsonpath={.spec.volumes}
$ kubectl delete pod qa-pod1 -n qa
```

十五、gVsior/runtimeclass

context
This cluster uses containerd as CRI runtime. Containerd's default runtime handler is runc.
Containerd has been prepared to support an additional runtime handler , runsc(gVisor).

Task
Create a RuntimeClass named untrusted using the prepared runtime handler named runsc.
Update all Pods in the namespace client to run on gvisor, unless they are already running on anon-default runtime handler.
You can find a skeleton manifest file at /cks/13/rc.yaml

中文翻译：

背景：
此集群使用 containerd 作为 CRI 运行时。Containerd 的默认运行时处理程序是 runc。

Containerd 已经准备好支持一个额外的运行时处理程序 runsc（gVisor）。

任务

使用已经准备好的 runsc，创建一个名字是 untrusted 的 RuntimeClass

在 gvisor 上更新 client 命名空间中的所有 POD，不更新已经通过其他 runc 运行的 pod。

可以在/cks/13/rc.yaml 中找到模板清单文件

解析

参考：

https://kubernetes.io/docs/concepts/containers/runtime-class/

$ vim /cks/13/rc.yaml

```
apiVersion: node.k8s.io/v1  # RuntimeClass is defined in the
node.k8s.io API group
kind: RuntimeClass
metadata:
  name: untrusted # The name the RuntimeClass will be referenced by
  # RuntimeClass is a non-namespaced resource
  handler: runsc # The name of the corresponding CRI configuration
```

apiVersion: node.k8s.io/v1beta1
kind: RuntimeClass
metadata:
** name: untrusted**
handler: runsc

$ kubectl apply -f /cks/13/rc.yaml

$ kubectl get pod -n client #查看 client 名称空间 pod
#假如看到的 pod 有两个：
#pod1
#pod2
看到的 pod，可以通过 kubectl get pod pod1 -n client -o yaml > 1.yaml
把 yaml 文件输出，修改如下部分，在重新 deleye，apply 这个 yaml 文件

```
spec:
    runtimeClassName: untrusted
    containers:
    - image: nginx:1.9
$ kubectl delete -f 1.yaml
$ kubectl apply -f 1.yaml
```