作者：李晓辉

联系方式：

1. 微信：Lxh_Chat

2. 邮箱：939958092@qq.com

概述

本文介绍了如何在 Kubernetes 环境中允许应用访问 Kubernetes API 的方法和步骤。通过详细的操作示例，展示如何为应用配置必要的权限，确保应用能够安全地与 Kubernetes API 进行交互。文章还讨论了相关的安全考虑和最佳实践，帮助读者在实际部署中实现高效且安全的 API 访问。在 Kubernetes 集群中，某些应用可能需要访问 Kubernetes API 以执行诸如动态配置、资源管理或监控等任务。例如，一个自动化工具可能需要查询 Pod 状态，或者一个监控系统可能需要读取资源使用情况。然而，直接访问 Kubernetes API 存在安全风险，因此需要合理配置权限，确保应用只能访问必要的资源，同时保护集群的安全。

场景

在 Kubernetes 集群中，某些应用可能需要访问 Kubernetes API 以执行诸如动态配置、资源管理或监控等任务。例如，一个自动化工具可能需要查询 Pod 状态，或者一个监控系统可能需要读取资源使用情况。然而，直接访问 Kubernetes API 存在安全风险，因此需要合理配置权限，确保应用只能访问必要的资源，同时保护集群的安全。

访问API的场景如下

以下情况下，Pod 中的应用必须能直接对 Kubernetes 的 API 发起请求：

- **自动扩展（Auto-scaling）**
  Kubernetes Horizontal Pod Autoscaler (HPA) 会根据 Pod 的 CPU 使用率或其他自定义指标，自动扩展或缩减 Pod 的副本数。HPA 需要访问 Kubernetes API 来监控指标并调整资源。
- **监控和日志收集**
  Prometheus 监控系统需要访问 Kubernetes API 获取节点、Pod 和服务的指标信息，从而进行资源监控和告警。Fluentd 等日志收集工具会从各个 Pod 中收集日志，并将其传输到集中式日志存储。它们需要通过 Kubernetes API 获取 Pod 的日志和元数据。

假设你正在管理一个 Kubernetes 集群，其中一个名为 `my-app` 的应用需要定期查询集群中其他 Pod 的状态，以便进行健康检查和自动扩展。为了实现这一功能，你需要为 `my-app` 配置适当的权限，使其能够安全地访问 Kubernetes API。

服务账号概述

服务账号（ServiceAccount）是 Kubernetes 中的一种资源，用于为在 Pod 中运行的应用程序提供身份验证和授权。服务账号允许应用程序以特定的身份访问 Kubernetes API 或其他需要身份验证的服务。每个服务账号都有一个与之关联的密钥对，用于生成用于身份验证的令牌。

服务账号的作用

1. **身份验证**
   服务账号为在 Pod 中运行的应用程序提供了一种身份验证机制，使其能够安全地访问 Kubernetes API 或其他服务。
2. **授权**
   通过与 Role 或 ClusterRole 结合使用，服务账号可以被授予特定的权限，从而实现细粒度的访问控制。
3. **隔离**
   不同的服务账号可以被分配不同的权限，从而实现不同应用程序之间的隔离和安全。

安全最佳实践

1. **最小权限原则**
   为服务账号分配的权限应仅限于其运行所需，避免赋予过多权限。
2. **限制服务账号的使用范围**
   尽量将服务账号的使用限制在特定的命名空间内，避免在集群范围内使用。
3. **定期轮换密钥**
   定期轮换服务账号的密钥，以减少密钥泄露的风险。

如果 Pod 定义不指定服务账号，则 Pod 将会使用 `default` 服务账号。OpenShift 不会向 `default` 服务账号授予任何权限。

授权案例

我们来创建以下资源：

1. **服务账号**
2. **角色/集群角色**
3. **角色绑定**

最后将服务账号分配到应用，用命令测试权限是否足够。

创建服务账号

创建一个名为lxh-serviceaccount的账号

```
[student@workstation ~]$ oc create serviceaccount lxh-serviceaccount
```

创建角色

需要注意的是，角色和集群角色是不同的，角色是本项目内适用，而集群角色是横跨所有项目的。我们这次创建一个本项目内的角色，希望这个角色能查看 Pod 和 Deployment 的权限，本次创建的角色名为 `lxh-role`：

```
[student@workstation ~]$ oc create role lxh-role --verb=get --resource=deployments,pods
role.rbac.authorization.k8s.io/lxh-role created
[student@workstation ~]$ oc describe role lxh-role
Name:         lxh-role
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources         Non-Resource URLs  Resource Names  Verbs
  ---------         -----------------  --------------  -----
  pods              []                 []              [get]
  deployments.apps  []                 []              [get]
```

角色绑定

和角色一样，角色绑定也分为集群角色绑定和角色绑定，我们本次绑定的是本项目内的角色，所以选择普通的绑定即可

```
[student@workstation ~]$ oc create rolebinding lxh-bind --role lxh-role --serviceaccount de
[student@workstation ~]$ oc describe rolebinding lxh-bind
Name:         lxh-bind
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  lxh-role
Subjects:
  Kind            Name                Namespace
  ----            ----                ---------
  ServiceAccount  lxh-serviceaccount  default
```

测试权限

经过测试，权限符合预期

```
[student@workstation ~]$ oc auth can-i get pod --as system:serviceaccount:default:lxh-servi
yes
[student@workstation ~]$ oc auth can-i get deployment --as system:serviceaccount:default:lx
yes
[student@workstation ~]$ oc auth can-i get configmap --as system:serviceaccount:default:lxh
no
```

现有角色绑定

集群中默认也提供了一些角色，可以用以下方法绑定，就不用创建角色了

```
[student@workstation ~]$ oc get clusterrole
...
admin
cluster-admin
edit
view
...
```

有很多角色，可以用以下方法来查询其权限

```
[student@workstation ~]$ oc describe clusterrole view
Name:         view
Labels:       kubernetes.io/bootstrapping=rbac-defaults
              rbac.authorization.k8s.io/aggregate-to-edit=true
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources                                       Non-Resource URLs  Resource Names
  ---------                                       -----------------  --------------
  appliedclusterresourcequotas                    []                 []
  bindings                                        []                 []
  buildconfigs/webhooks                           []                 []
  buildconfigs                                    []                 []
  buildlogs                                       []                 []
  builds/log                                      []                 []
  builds                                          []                 []
  configmaps                                      []                 []
```

绑定方法可以用:

```
[student@workstation ~]$ oc adm policy add-role-to-user admin system:serviceaccount:default
clusterrole.rbac.authorization.k8s.io/admin added: "system:serviceaccount:default:lxh-servi
```

再测试权限，我们已经可以做更多事了

```
[student@workstation ~]$ oc auth can-i get configmap --as system:serviceaccount:default:lxh
yes
[student@workstation ~]$ oc auth can-i create configmap --as system:serviceaccount:default:
yes
```

本文档在线版本：https://www.linuxcenter.cn