Kubernetes 安全机制-RBAC 鉴权

前言：

课程名称：Kubernetes 安全机制-RBAC 鉴权

实验环境：

本章节 Kubernetes 集群环境如下：

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
工作节点	192.168.128.21	k8s-node01	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化
工作节点	192.168.128.22	k8s-node02	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

1、初识 RBAC

1.1、什么是 RBAC？

RBAC 是一种最常用的访问控制解决方案，它通过将用户分配到角色，将角色分配给权限实体，从而定义了哪些用户可以访问哪些资源，以及在资源上执行哪些操作。简单来说，RBAC 允许对不同的用户或组分配不同的权限，以实现资源的安全管理。

在 Kubernetes 中，RBAC 用于控制对 API 对象的访问权限，可以限制对集群资源的访问，例如容器、节点、服务以及部署等。

在 Kubernete 中，RBAC 的主要组件包括：

1、Role：定义权限的具体内容，例如获取、创建、修改或删除某个对象。

2、ClusterRole：在整个集群中定义权限。

3、RoleBinding：将 Role 绑定到用户或组，即为用户授权。

4、ClusterRoleBinding：将 ClusterRole 绑定到用户或组，即为用户授权。

其中，Role 和 RoleBinding 控制访问一个 namespace 中的资源，ClusterRole 和 ClusterRoleBinding 控制访问整个集群中的资源。

管理员可以基于自己需求的情况自定义规则。因此，RBAC 为 Kubernetes 的安全管理提供了高度的灵活性和可配置性，非常适合用于企业级和大规模的应用场景。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

1.2、认证、授权、准入控制

Kubernetes 中的 RBAC（基于角色的访问控制）机制提供了三个方面的控制：

1、认证：即控制用户身份的验证工作，Kubernetes 通过使用 OAuth2、OpenID Connect、TLS 等协议来进行客户端身份认证。

2、授权：即控制用户能够访问的资源和操作，使用 RBAC 以及其他方式来实现。

3、准入控制：即对用户请求进行检查，以确保它们符合 Kubernetes 集群的条件和策略。准入控制包括各种策略和机制，如网络策略、pod、服务端点生成、限制 pod 的资源使用、入站和出站流量等等。

现实例子：

假如小美认证小明做自己的男朋友（认证过程）。小美又授权小明晚上可以来自己家（授权过程）。月黑风高夜，此时，小明拥有去小美家的权限，但是这个时候小美父母刚好在家，那么这个时候小美可以拒绝小明来自己家（准入控制）（就等同于命名空间被限制 10vCPU、10Gi 内存，这个时候小美父母在这个空间下，空间占满，那么小明就进不来）。

- 认证基本介绍

kubernetes 主要通过 APIserver 对外提供服务，那么就需要对访问 apiserver 的用户做认证，如果任何人都能访问 apiserver，那么就可以随意在 k8s 集群部署资源，这是非常危险的，也容易被黑客攻击渗透，所以需要我们对访问 k8s 系统的 apiserver 的用户进行认证，确保是合法的符合要求的用户。

- 授权基本介绍

在授权方面，Kubernetes 使用 RBAC 提供了细粒度的授权，它通过 Role 和 Role Binding，ClusterRole 和 ClusterRole Binding 四个重要组件来定义访问控制策略，使得管理员可以很好地控制集群中各个用户或组的相对权限。

- 准入控制基本介绍

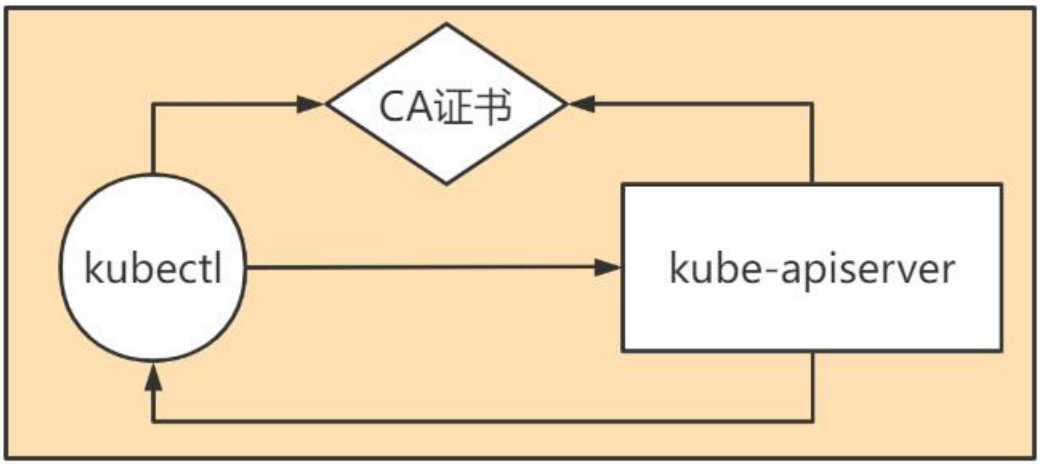
在准入控制方面，Kubernetes 提供了一系列的 admission 控制器，如 NamespaceLifecycle、LimitRanger、ResourceQuota、PodSecurityPolicy、ServiceAccount 等等，通过启用或禁用这些控制器来限制 Kubernetes 对象的创建、修改和删除，从而进一步提高了安全性。

1.2.1、认证

- k8s 客户端访问 apiserver 的几种认证方式

（1）客户端认证

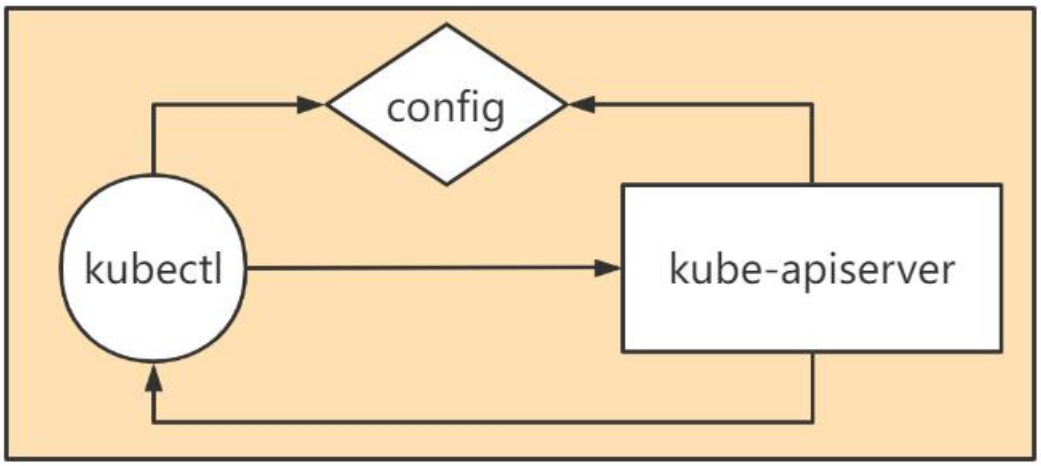
客户端认证也称为双向 TLS 认证，kubectl 在访问 apiserver 的时候，apiserver 也要认证 kubectl 是否是合法的，他们都会通过 ca 根证书来进行验证，如下图：



kubectl 客户端在访问请求 kube-apiserver 组件时，会带着 ca 证书去做请求。这个时候 kube-apiserver 会验证 ca 证书是否正确。正确则能够被 kube-apiserver 所信任。

（2）Bearertoken

Bearertoken 的方式，可以理解为 apiserver 将一个密码通过了非对称加密的方式告诉了 kubectl，然后通过该密码进行相互访问，如下图：



kubectl 访问 k8s 集群，要找一个 config 文件，基于 config 文件里的用户访问 kube-apiserver。

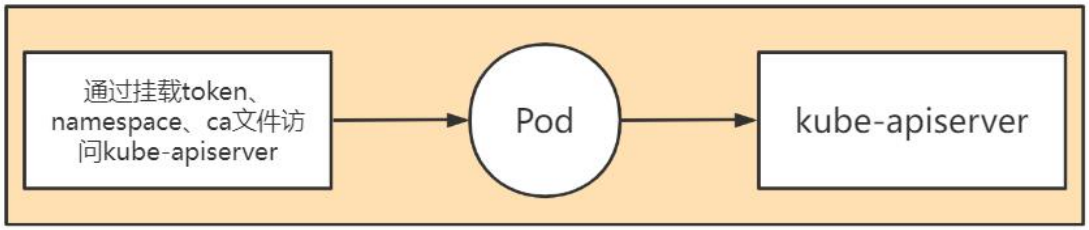
```
    # 初始化 kubernetes 集群时，创建的 config 文件。默认使用的是这个文件访问的 kube-apiserver 组件。
    [root@k8s-master01 ~]# ls .kube/config -l
    -rw------- 1 root root 5638 Nov 17 03:26 .kube/config

    # 集群默认生成的文件，config 文件就是根据 admin.conf 文件拷贝过去的。所以当删除.kube 文件夹时，可以根据这个文件来恢复，如果都被删了，那么 k8s 集群就无法访问了。
```

```
[root@k8s-master01 ~]# cd /etc/kubernetes/
[root@k8s-master01 kubernetes]# ls admin.conf -l
-rw------- 1 root root 5638 Nov 17 03:26 admin.conf
```

（3）Serviceaccount

上面客户端 ca 证书认证和 Bearertoken 的两种认证方式，都是外部访问 kube-apiserver 的时候使用的方式，那么我们这次说的 Serviceaccount 是内部访问 pod 和 kube-apiserver 交互时候采用的一种方式。

Serviceaccount 包括：namespace、token、ca，且通过目录挂载的方式给予 pod，当 pod 运行起来的时候，就会读取到这些信息，从而使用该方式和 kube-apiserver 进行通信。如下图：



● 演示 config 文件认证 apiserver

（1）kubectl config view 命令

在 kubernetes 集群当中，当我们使用 kubectl 操作 k8s 资源时候，需要确定我们用哪个用户访问哪个 k8s 集群，kubectl 操作 k8s 集群资源会去/root/.kube 目录下找 config 文件，可以通过“kubectl config view”查看 config 文件配置，如下：

```
[root@k8s-master01 ~]# kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.128.11:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
```

```
    user:
      client-certificate-data: DATA+OMITTED
      client-key-data: DATA+OMITTED
```

执行该命令后，您将看到输出显示有关集群、用户和上下文的详细信息，包括当前使用的上下文。

（2）kubectl 指定 config 文件查询集群资源

我们可以使用 kubectl 指定 config 文件进行集群资源查询，当我们有多套 k8s 集群时，我们可以指定不同的 config 文件来查询不同集群的资源。

```
[root@k8s-master01 ~]# kubectl get pods -A --kubeconfig=/root/.kube/config
```

```
[root@k8s-master01 ~]# kubectl get pods -A --kubeconfig=/root/.kube/config
NAMESPACE          NAME                                       READY   STATUS    RESTARTS        AGE
calico-apiserver   calico-apiserver-5858bff4cb-4mndq          1/1     Running   3 (3m32s ago)   18d
calico-apiserver   calico-apiserver-5858bff4cb-7bh97          1/1     Running   5 (3m33s ago)   18d
calico-system      calico-kube-controllers-689465784b-mtnxt   1/1     Running   2 (4m13s ago)   18d
calico-system      calico-node-2n9kd                          1/1     Running   2 (4m17s ago)   18d
calico-system      calico-node-ngkfw                          1/1     Running   2 (4m15s ago)   18d
calico-system      calico-node-npxvl                          1/1     Running   2 (4m13s ago)   18d
calico-system      calico-typha-7d5c457c7d-pm7mq              1/1     Running   2 (4m13s ago)   18d
calico-system      calico-typha-7d5c457c7d-qjdz8              1/1     Running   2 (4m15s ago)   18d
calico-system      csi-node-driver-fxzng                      2/2     Running   4 (4m17s ago)   18d
calico-system      csi-node-driver-q66xf                      2/2     Running   4 (4m15s ago)   18d
calico-system      csi-node-driver-v6m5g                      2/2     Running   4 (4m13s ago)   18d
kube-system        coredns-5bbd96d687-gxhbp                   1/1     Running   2 (4m17s ago)   18d
kube-system        coredns-5bbd96d687-h74qj                   1/1     Running   2 (4m17s ago)   18d
kube-system        etcd-k8s-master01                          1/1     Running   3 (4m17s ago)   18d
kube-system        kube-apiserver-k8s-master01                1/1     Running   3 (4m17s ago)   18d
kube-system        kube-controller-manager-k8s-master01       1/1     Running   3 (4m17s ago)   18d
kube-system        kube-proxy-2snpx                           1/1     Running   2 (4m13s ago)   18d
kube-system        kube-proxy-5gknd                           1/1     Running   2 (4m15s ago)   18d
kube-system        kube-proxy-r2vtg                           1/1     Running   2 (4m17s ago)   18d
kube-system        kube-scheduler-k8s-master01                1/1     Running   3 (4m17s ago)   18d
kube-system        tigera-operator-5d6845b496-969bv           1/1     Running   2 (4m15s ago)   18d
[root@k8s-master01 ~]#
```

1.2.2、授权

如果用户通过认证，什么权限都没有，需要一些后续的授权操作，如对资源的增删该查等，kubernetes 1.6 之后开始有 RBAC（基于角色的访问控制机制）授权检查机制。Kubernetes 的授权是基于插件形成的，其常用的授权插件有以下几种：

1、Node（节点认证）

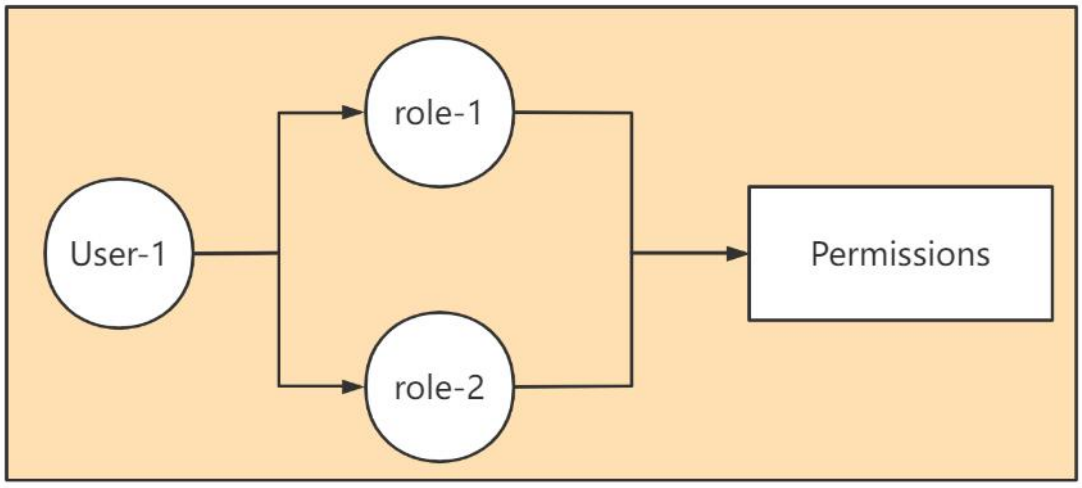
2、ABAC（基于属性的访问控制）

3、RBAC（基于角色的访问控制）

4、Webhook（基于 http 回调机制的访问控制）

- 什么是 RBAC（基于角色的授权）？

让一个用户（Users）扮演一个角色（Role），角色拥有权限，从而让用户拥有这样的权限，随后在授权机制当中，只需要将权限授予某个角色，此时用户将获取对应角色的权限，从而实现角色的访问控制。如图：



在 k8s 的授权机制当中，采用 RBAC 的方式进行授权，其工作逻辑是，把对对象的操作权限定义到一个角色（role）当中，再将用户绑定到该角色，从而使用户得到对应角色的权限。

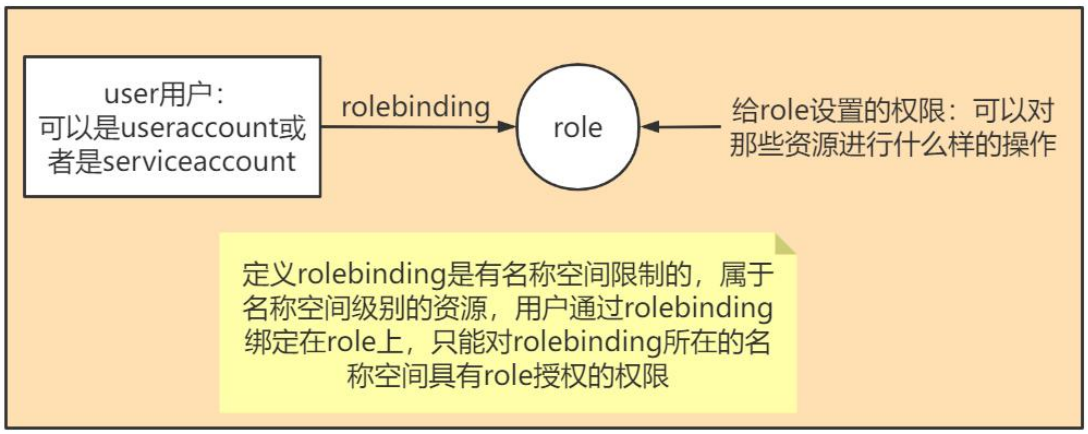
如果通过 rolebinding 绑定 role，只能对 rolebingding 所在的名称空间的资源有权限，上图 user-1 这个用户绑定到 role-1 上，只对 role-1 所在的名称空间资源有权限，对其他名称空间资源没有权限，属于名称空间级别的。

另外，k8s 为此还有一种集群级别的授权机制，就是定义一个集群角色（ClusterRole），对集群内的所有资源都有可操作的权限，从而将 User-2 通过 ClusterRoleBinding 到 ClusterRole，从而使 User-2 拥有集群的操作权限。

● Role、RoleBinding、ClusterRole 和 ClusterRoleBinding 的关系如下图：

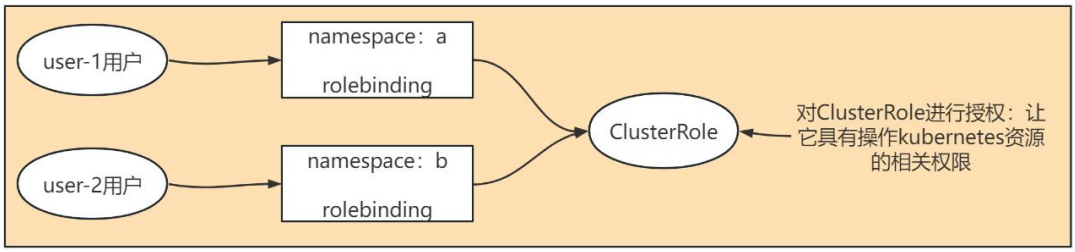
1、用户基于 rolebinding 绑定到 role

限定在 rolebinding 所在的名称空间：



2、用户基于 rolebinding 绑定到 clusterrole

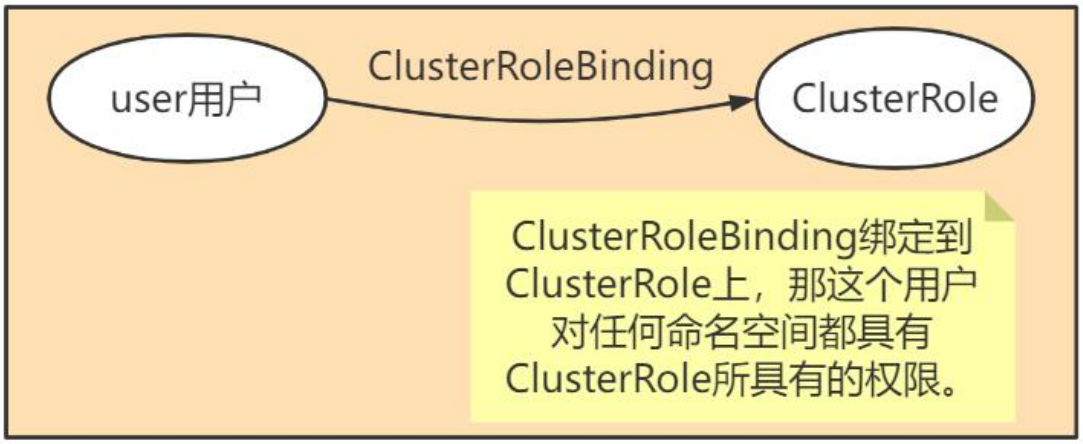
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



rolebinding 绑定 clusterrole 的好处：

假如有 6 个名称空间，每个名称空间的用户都需要对自己的名称空间有管理员权限，那么需要定义 6 个 role 和 rolebinding，然后依次绑定，如果名称空间更多，我们需要定义更多的 role，这个是很麻烦的，所以我们引入 clusterrole，定义一个 clusterrole，对 clusterrole 授予所有权限，然后用户通过 rolebinding 绑定到 clusterrole，就会拥有自己名称空间的管理员权限了。

3、用户基于 clusterrolebinding 绑定到 clusterrole



1.2.3、准入控制

- 什么是准入控制？

准入控制器会在请求通过认证和鉴权之后，对象被持久化之前拦截到达 kube-apiserver 服务的请求。

准入控制器可以限制创建、删除、修改对象的请求。准入控制器不会阻止读取（get、watch 或 list）对象的请求。

- 准入控制阶段

准入控制过程分为两个阶段。第一阶段，运行变更准入控制器。第二阶段，运行验证准入控制器。某些控制器既是变更准入控制器又是验证准入控制器。

如果两个阶段之一的任何一个控制器拒绝了某请求，则整个请求将立即被拒绝，并向最终用户返回错误。

- 如何启用一个准入控制器？

Kubernetes API 服务器的 enable-admission-plugins 参数表示开启的（以逗号分隔的）准入控制插件列表，这些插件会在集群修改对象之前被调用。

```
查看当前启用的准入控制器：
    [root@k8s-master01 ~]# cat /etc/kubernetes/manifests/kube-apiserver.yaml
    ...省略部分内容...
    spec:
      containers:
      - command:
        ...省略部分内容...
        - --enable-admission-plugins=NodeRestriction
        ...省略部分内容...
```

- 官网地址

可以查看如下官网网址，来确认你需要启用的准入控制器：

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

1.3、RBAC 的组成

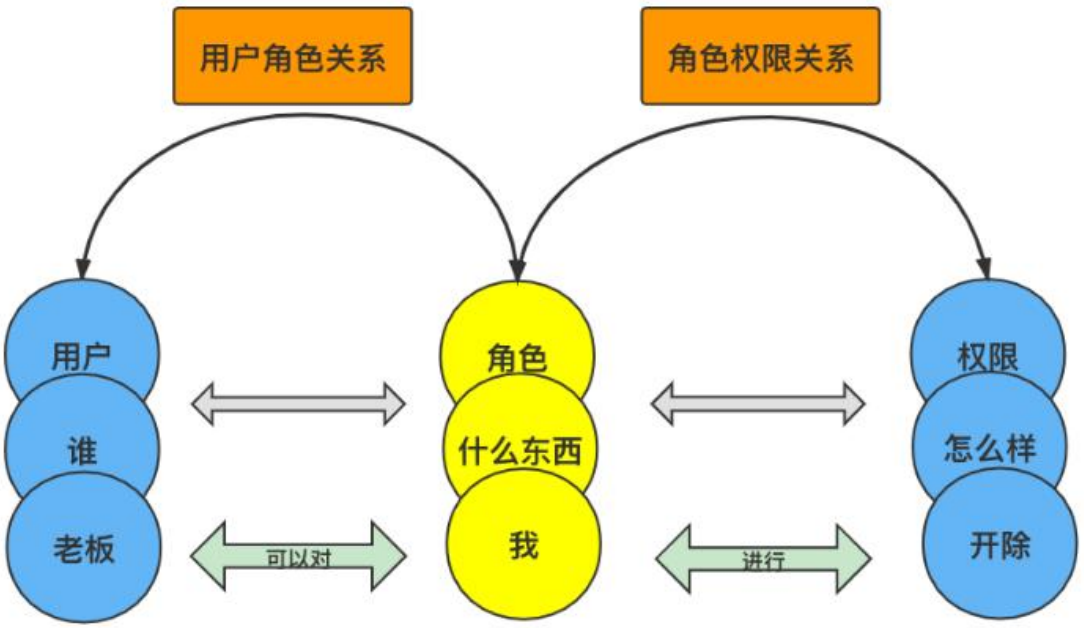
在 RBAC 模型里面，有 3 个基础组成部分，分别是：用户、角色和权限。

RBAC 的组成：

```
    User（用户）：每个用户都有唯一的 UID 识别，并被授予不同的角色。
    Role（角色）：不同角色具有不同的权限。
    Permission（权限）：访问权限。
    用户->角色映射：用户和角色之间的映射关系。
    角色->权限映射：角色和权限之间的映射。
```

图一：用户、角色和权限之间的关系如下图所示：

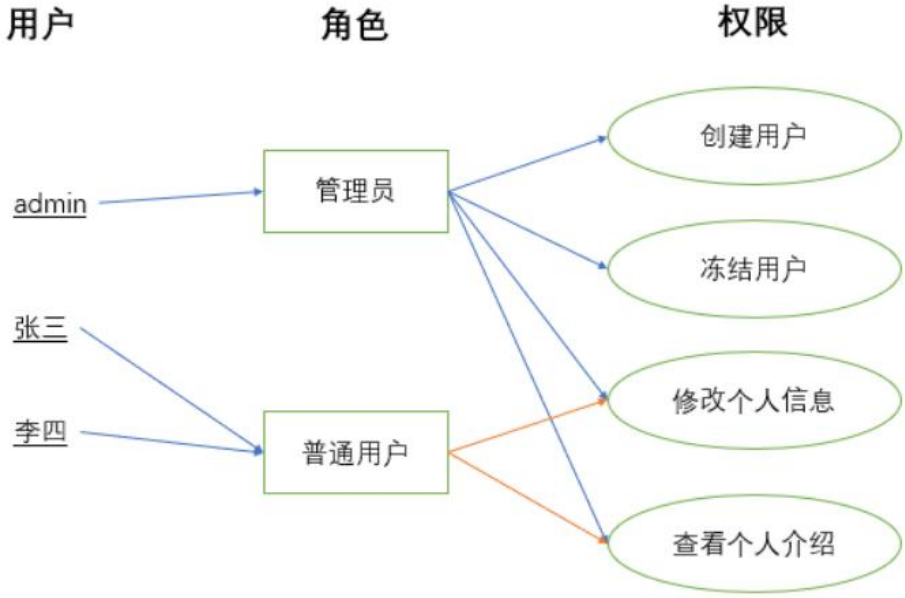
版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



老板可以对我进行开除的决定。我不能对老板的决定进行反对，然后我就被公司人事开除了。

图二：用户、角色和权限之间的关系如下图所示：

管理员和普通用户被授予不同的权限。普通用户只能去修改和查看个人信息，而不能创建创建用户和冻结用户，而管理员由于被授予所有权限，所以可以做所有操作。



2、UserAccount、ServiceAccount 认证

kubernetes 中账户区分为：User Accounts（用户账户）和 Service Accounts

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

（服务账户）两种。

2.1、UserAccount 介绍

UserAccount 是给 kubernetes 集群外部用户使用的，例如运维人员或者集群管理人员，kubeadm 安装的 k8s，默认用户账号是 kubernetes-admin。

k8s 客户端（一般用 kubectl）访问 kube-apiserver 组件。

kube-apiserver 需要对客户端做认证，使用 kubeadm 安装的 K8s，会在用户家目录下创建一个认证配置文件.kube/config 这里面保存了客户端访问 kube-apiserver 的密钥相关信息，这样当用 kubectl 访问 k8s 时，它就会自动读取该配置文件，向 kube-apiserver 发起认证，然后完成操作请求。

后面会实操讲解“限制不同的用户操作 k8s 集群”

2.2、ServiceAccount 介绍

ServiceAccount 是 Pod 使用的账号，Pod 容器的进程需要访问 kube-apiserver 时用的就是 ServiceAccount 账户。

ServiceAccount 仅局限它所在的 namespace 命名空间，每个 namespace 创建时都会自动创建一个 default service account。创建 Pod 时，如果没有指定 Service Account，Pod 则会使用 default Service Account。

- 创建命名空间，查看是否生成有 Service Account

```
[root@k8s-master01 ~]# kubectl create ns test
namespace/test created

[root@k8s-master01 ~]# kubectl get sa -n test
NAME      SECRETS   AGE
default   1         5s
```

- ServiceAccount 应用场景

ServiceAccount 是 Kubernetes 中的一种安全机制，它们用于控制 Pod 访问 Kubernetes API 服务器和其他需要进行身份验证/授权的服务。下面是一些使用 ServiceAccount 的常见应用场景：

```
1、安全访问 Kubernetes API
    管理员可以创建一个有限的、能够访问 Kubernetes API 的 ServiceAccount 来授权操作，从而减少了意外修改或意外删除 Kubernetes 资源的风险。这可以通过把 ServiceAccount 的访问权限限制到仅适用于特定命名空间或资源上实现。

2、微服务架构中的访问控制
```

在应用服务采用微服务架构时，每个服务都需要独立的访问授权，这通常涉及在服务之间传递访问令牌。将每个服务的访问令牌存储在 ServiceAccount 中，能够更好的隔离每个服务，并精确控制对 API 访问的权限。

3、容器中的自动化 API 访问

可以通过创建一个具有访问 API 权限的 ServiceAccount，运行在容器内的进程就可以使用该 ServiceAccount 直接访问 Kubernetes API，从而允许容器自动调整自己的资源、存储、安全以及服务发现等方面。

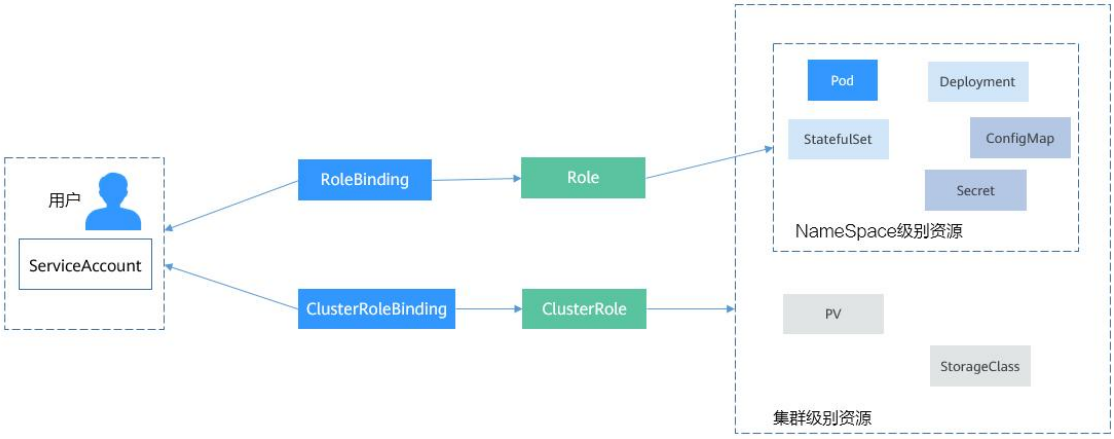
4、在 CI/CD 中自动迭代资源

通过使用 create、apply 和 delete 操作等 Kubernetes API 服务，ServiceAccount 可以在 CI/CD 流水线中自动更新 Kubernetes 资源，如部署、配置映射和服务等。

总之，ServiceAccount 在 Kubernetes 中是一个至关重要的概念，它们提供了可靠的身份验证和授权机制，使管理员可以控制白名单访问和授权机制的限制。

2.3、ServiceAccount 应用案例

目标：创建一个 Service Account 资源，使用 pod 挂载此 sa，在 pod 内通过 sa 访问 apiserver 查看集群资源信息。



（1）创建 Service Account

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi serviceaccount-test.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: test
  namespace: default
  labels:
    sa: test
```

```
2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f serviceaccount-test.yaml
    serviceaccount/test created

3、查看 sa
    [root@k8s-master01 ~]# kubectl get sa -l sa=test
    NAME   SECRETS   AGE
    test   0         46s
```

（2）创建 Pod，挂载 Service Account

```
1、创建资源清单文件
    [root@k8s-master01 ~]# cat serviceaccount-pod.yaml
    apiVersion: v1
    kind: Pod
    metadata:
      name: sa-test
      namespace: default
      labels:
        app: sa-test
    spec:
      serviceAccountName: test
      containers:
      - name: nginx
        ports:
        - containerPort: 80
        image: nginx:latest
        imagePullPolicy: IfNotPresent

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f serviceaccount-pod.yaml
    pod/sa-test created

3、查看 pod
    [root@k8s-master01 ~]# kubectl get pods -l app=sa-test
    NAME      READY   STATUS    RESTARTS   AGE
    sa-test   1/1     Running   0          41s
```

（3）进入容器，使用 sa 访问 kube-apiserver 接口查看命名空间详细信息

```
1、进入容器
    [root@k8s-master01 ~]# kubectl exec -it sa-test -- bash

2、查看挂载进来的文件
```

```
root@sa-test:/# cd /var/run/secrets/kubernetes.io/serviceaccount/
root@sa-test:/var/run/secrets/kubernetes.io/serviceaccount# ls
ca.crt  namespace  token
```

3、通过访问 k8s api 接口查看 kube-system 命名空间详细信息

```
root@sa-test:/var/run/secrets/kubernetes.io/serviceaccount# curl \
--cacert ./ca.crt \
-H "Authorization: Bearer $(cat ./token)" \
https://kubernetes/api/v1/namespaces/kube-system
```

```
root@sa-test:/var/run/secrets/kubernetes.io/serviceaccount# curl \
--cacert ./ca.crt \
-H "Authorization: Bearer $(cat ./token)" \
https://kubernetes/api/v1/namespaces/kube-system
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {},
  "status": "Failure",
  "message": "namespaces \"kube-system\" is forbidden: User \"system:serviceaccount:default:test\" cannot get resource \"namespaces\" in API group \"\" in the namespace \"kube-system\"",
  "reason": "Forbidden",
  "details": {
    "name": "kube-system",
    "kind": "namespaces"
  },
  "code": 403
}root@sa-test:/var/run/secrets/kubernetes.io/serviceaccount#
```

从上图结果能看到 sa 能通过 https 方式成功认证 kube-apiserver，但是没有权限访问 k8s 资源，所以 code 状态码是 403，表示没有权限操作 k8s 资源。

（4）对刚才创建的 sa 做授权

1、查看 cluster-admin clusterrole 资源对象详细信息

```
[root@k8s-master01 ~]# kubectl describe clusterrole cluster-admin
```

```
[root@k8s-master01 ~]# kubectl describe clusterrole cluster-admin
Name:         cluster-admin
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
  *.*        []                 []              [*]
             [*]                []              [*]
[root@k8s-master01 ~]#
```

从上图可以看到 cluster-admin clusterrole 拥有对所有资源的权限。

2、将 default 名称空间下的 test sa 通过 clusterrolebinding 绑定到名为 cluster-admin 的 clusterrole 上。

```
[root@k8s-master01 ~]# vi clusterrolebinding-test-admin.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: sa-test-admin
subjects:
- kind: ServiceAccount
  name: test
  namespace: default
roleRef:
```

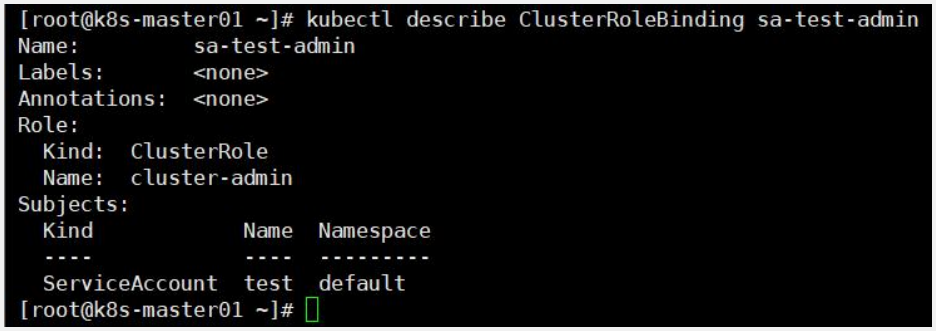
```
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io

[root@k8s-master01 ~]# kubectl apply -f clusterrolebinding-test-admin.yaml
clusterrolebinding.rbac.authorization.k8s.io/sa-test-admin created
```

3、 查看 ClusterRoleBinding

```
[root@k8s-master01 ~]# kubectl get ClusterRoleBinding sa-test-admin
NAME            ROLE                        AGE
sa-test-admin   ClusterRole/cluster-admin   54s
```

4、查看 sa-test-admin ClusterRoleBinding 的详细信息

```
[root@k8s-master01 ~]# kubectl describe ClusterRoleBinding sa-test-admin
```



从上图可以看出名为 test 的 service account 绑定到了名为 cluster-admin 的 clusterrole 上。那么此时 test sa 就拥有了 cluster-admin clusterrole 所对应的权限。

（5）再次使用 sa 访问 kube-apiserver 接口查看命名空间详细信息

1、进入容器

```
[root@k8s-master01 ~]# kubectl exec -it sa-test -- bash
```

2、通过访问 k8s api 接口查看 kube-system 命名空间详细信息

```
root@sa-test:/# cd /var/run/secrets/kubernetes.io/serviceaccount
root@sa-test:/var/run/secrets/kubernetes.io/serviceaccount# curl \
--cacert ./ca.crt \
-H "Authorization: Bearer $(cat ./token)" \
https://kubernetes/api/v1/namespaces/kube-system
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "kube-system",
    "uid": "ce609e28-6652-4648-a0d2-fb20ec8bd165",
    "resourceVersion": "4",
```

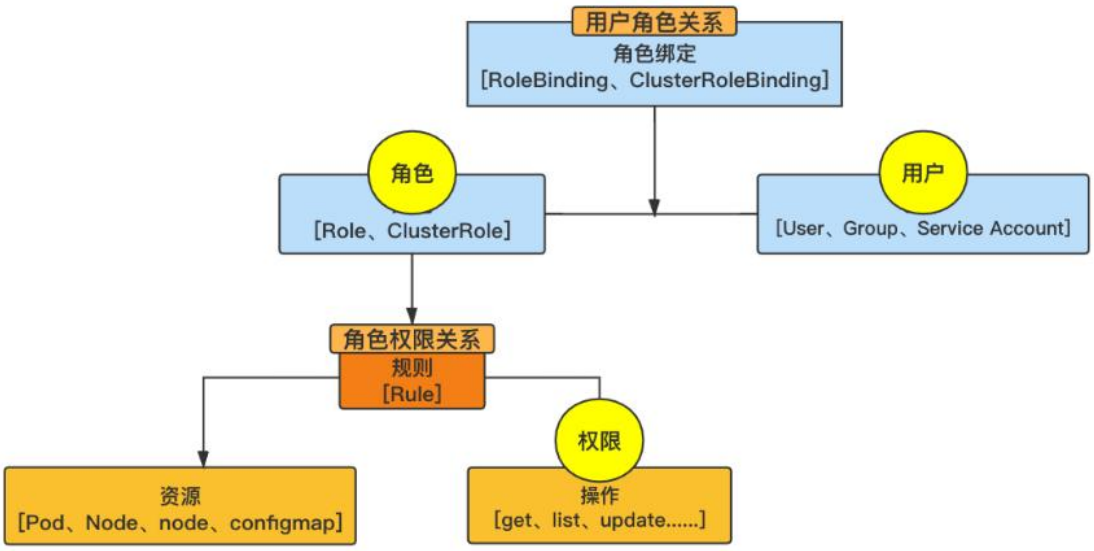
```
        "creationTimestamp": "2023-05-24T10:01:57Z",
        "labels": {
          "kubernetes.io/metadata.name": "kube-system"
        },
        "managedFields": [
          {
            "manager": "kube-apiserver",
            "operation": "Update",
            "apiVersion": "v1",
            "time": "2023-05-24T10:01:57Z",
            "fieldsType": "FieldsV1",
            "fieldsV1": {
              "f:metadata": {
                "f:labels": {
                  ".": {},
                  "f:kubernetes.io/metadata.name": {}
                }
              }
            }
          }
        ]
      },
      "spec": {
        "finalizers": [
          "kubernetes"
        ]
      },
      "status": {
        "phase": "Active"
      }
    }
    # 通过上面输出可以看出，对 sa 做授权之后就有权限访问 k8s 资源了。

3、通过访问 k8s api 接口查看 kube-system 命名空间下所有 pods 的详细信息
    root@sa-test:/var/run/secrets/kubernetes.io/serviceaccount# curl \
    --cacert ./ca.crt \
    -H "Authorization: Bearer $(cat ./token)" \
    https://kubernetes/api/v1/namespaces/kube-system/pods
    ...省略输出...
```

3、角色与角色绑定

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

在 Kubernetes 中，所有资源对象都是通过 API 进行操作。

RBAC API 声明了四种 Kubernetes 对象：Role、ClusterRole、RoleBinding 和 ClusterRoleBinding。可以用 kubectl get 获取到这些资源。



3.1、Role：角色

在 Kubernetes 中，Role 对象表示一组权限，用于授予在特定命名空间中对 Kubernetes 资源的访问权限。Role 对象授予的权限只能限制在特定命名空间中，不能横跨多个命名空间。

● 实战演练

实例 1：定义一个角色，只有对 Pod 资源读取的权限

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi role-read.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: role-read
  namespace: default
rules:
- apiGroups: [""]
  resources: ["pods"]
  resourceNames: []
  verbs: ["get","watch","list"]
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f role-read.yaml
role.rbac.authorization.k8s.io/role-read created
```

对上面的 yaml 文件说明：

这是一个 Kubernetes Role YAML 配置文件的示例，该角色定义可以在默认命名空间中读取 pods 资源。下面是每个字段的解释：

- apiVersion：角色对象所使用的 Kubernetes API 版本。
- kind：角色对象的 Kubernetes 资源类型。
- metadata：元数据部分包含有关角色的基本信息，如名称和命名空间。
- name：定义角色名称。
- namespace：定义角色所属的命名空间。
- rules：定义角色授权规则的列表。

  apiGroups：指定资源所属的 API 组，""表示核心组，核心组就是 v1。

  resources：定义该角色可以访问的 Kubernetes 资源类型列表。

  resourceNames：定义规则适用的资源名称。留空表示适用于所有资源。

  verbs：角色允许的操作列表。get、list 和 watch 动作都可以返回一个资源的完整详细信息。就返回的数据而言，它们是等价的。

实例 2：定义一个角色，只允许访问名称为 my-pod 的 Pod 资源

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi role-read-my_pod.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: role-read-my_pod
  namespace: default
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  resourceNames:
  - my-pod
```

2、更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f role-read-my_pod.yaml
role.rbac.authorization.k8s.io/role-read-my_pod created
```

实例 3：对 apps 这个 API 组中的 deployment 资源拥有获取、删除、更新、创建权限

1、创建资源清单文件

```
[root@k8s-master01 ~]# vi role-deployment.yaml
apiVersion: rbac.authorization.k8s.io/v1
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
  kind: Role
  metadata:
    name: role-deployment
    namespace: default
  rules:
  - apiGroups: ["apps/v1"]
    resources: ["deployments"]
    resourceNames: []
    verbs: ["get","watch","list","create","update","patch","delete"]

2、更新资源清单文件
  [root@k8s-master01 ~]# kubectl apply -f role-deployment.yaml
  role.rbac.authorization.k8s.io/role-deployment created
```

3.2、ClusterRole：集群角色

在 Kubernetes 中，ClusterRole 定义了可以在整个集群中授予的访问控制规则。与 Role 不同，ClusterRole 可以在多个命名空间中定义，使用时需要将其绑定到对应的 Kubernetes 用户或服务账户上。下面是几个关于 ClusterRole 的相关介绍：

1、ClusterRole 是一种 Kubernetes 资源类型，用于控制集群范围内的访问控制。一个 ClusterRole 定义可以控制多个资源的访问权限，包括全局资源和跨命名空间的资源。

2、与 Role 类型相似，ClusterRole 也包含一个规则集列表，规定了资源和操作的权限。

3、集群管理员可以在集群中创建和管理 ClusterRole 对象，它们可以在不同命名空间中共享。

4、要将 ClusterRole 绑定到用户或服务账户上，可以使用 ClusterRoleBinding 对象，将 ClusterRole 与指定的实体绑定起来。

5、ClusterRole 通常用于控制全局操作，如管理节点、访问集群状态、访问各个命名空间中的所有 Pod、网络策略等。

ClusterRole 可用于以下特殊元素的授权。

1、集群范围的资源，例如 Node、PV 等。

2、非资源型的路径，例如：/healthz

3、包含全部命名空间的资源，例如 Pods

- Kubernetes 集群默认存在的 ClusterRole

以下是 Kubernetes 中默认提供的一些 ClusterRole：

1、cluster-admin：该 ClusterRole 授予对集群的完全访问权限，可以控制任何资源，包括集群范围内的操作。

2、admin：该 ClusterRole 授予对所有命名空间内资源的读写权限，但不能够操作集群

范围内的资源。

3、edit：该 ClusterRole 授予对所有命名空间内的资源的读写权限，但不能够操作一些集群范围内的一些资源，如节点等。

4、view：该 ClusterRole 授予对所有命名空间内资源的只读权限，同样不能够操作集群范围内的一些资源。

- 实战演练

实例 1：定义一个集群角色可让用户访问任意 secrets

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi clusterrole-read-secrets.yaml
    apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRole
    metadata:
      name: clusterrole-read-secrets
    rules:
    - apiGroups: [""]
      resources: ["secrets"]
      verbs: ["get","watch","list"]

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f clusterrole-read-secrets.yaml
    clusterrole.rbac.authorization.k8s.io/clusterrole-read-secrets created
```

实例 2：读取核心组的 Node 资源（Node 属于集群级的资源，所以必须存在于 ClusterRole 中）

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi clusterrole-read-nodes.yaml
    apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRole
    metadata:
      name: clusterrole-read-nodes
    rules:
    - apiGroups: [""]
      resources: ["nodes"]
      verbs: ["get","watch","list"]

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f clusterrole-read-nodes.yaml
    clusterrole.rbac.authorization.k8s.io/clusterrole-read-nodes created
```

实例 3：允许对非资源端点“/healthz”及其所有子路径进行 GET 和 POST 操作（必须使用 ClusterRole 和 ClusterRoleBinding）

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi clusterrole-url-healthz.yaml
    apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRole
    metadata:
      name: clusterrole-read-nodes
    rules:
    - nonResourceURLs: ["/healthz","/healthz/*"]
      verbs: ["get","post"]

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f clusterrole-url-healthz.yaml
    clusterrole.rbac.authorization.k8s.io/clusterrole-read-nodes configured
```

3.3、RoleBinding：角色绑定

在 Kubernetes 中，RoleBinding 是一种授权对象，用于绑定一个 Role 或 ClusterRole 对象到一个或多个用户或服务账户。通过 RoleBinding，您可以为 Kubernetes 集群中的任何用户或服务账户授权特定权限，并严格控制其可以访问或操作的资源和操作。

● 实战演练

实例 1：将 default 命名空间中的 role-read 角色与用户 test 进行绑定

```
（1）查看 role
    [root@k8s-master01 ~]# kubectl get role role-read
    NAME        CREATED AT
    role-read   2023-06-12T03:28:04Z

（2）创建用户
    [root@k8s-master01 ~]# useradd test

（3）创建 RoleBinding 资源清单文件
    [root@k8s-master01 ~]# vi rolebinding-1.yaml
    apiVersion: rbac.authorization.k8s.io/v1
    kind: RoleBinding
    metadata:
      name: pod-read-bind
    subjects:
    - kind: User
      name: test
    roleRef:
      kind: Role
```

```
    name: role-read
    apiGroup: rbac.authorization.k8s.io
```

对上面的 yaml 文件说明：

该 RoleBinding 中指定了一个名为 test 的用户作为被授权用户，使用了 RBAC 中的 User 类型。RoleBinding 通过 roleRef 字段引用了一个名为 role-read 的 Role 资源对象，该 Role 对象定义了授权用户读取 Pod 资源的权限。

（4）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f rolebinding-1.yaml
rolebinding.rbac.authorization.k8s.io/pod-read-bind created
```

实例 2：将 view 集群角色与 dev Service Account 进行绑定

（1）查看 clusterrole

```
[root@k8s-master01 ~]# kubectl get clusterrole view
NAME   CREATED AT
view   2023-05-24T10:01:58Z
```

（2）创建 sa

```
[root@k8s-master01 ~]# kubectl create serviceaccount dev
serviceaccount/dev created
```

（3）创建 ClusterRoleBinding 资源清单文件

```
[root@k8s-master01 ~]# vi rolebinding-read.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rolebinding-read
subjects:
- kind: ServiceAccount
  name: dev
  namespace: default
roleRef:
  kind: ClusterRole
  name: view
  apiGroup: rbac.authorization.k8s.io
```

对上面的 yaml 文件说明：

这是一份 Kubernetes 中的 RoleBinding 资源对象的描述文件，用于授权一个名为 dev 的 ServiceAccount 在 default 命名空间中读取 Kubernetes 集群中的节点资源。

该 RoleBinding 中指定了一个 Subject，使用了 RBAC 中的 ServiceAccount 类型。它的 kind 属性为 ServiceAccount，name 属性为 dev，namespace 属性为 default，表示将 ServiceAccount dev 与该 RoleBinding 绑定。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
（4）更新资源清单文件
   [root@k8s-master01 ~]# kubectl apply -f rolebinding-read.yaml
   rolebinding.rbac.authorization.k8s.io/rolebinding-read created

（5）查看 rolebinding
   [root@k8s-master01 ~]# kubectl get rolebinding rolebinding-read
   NAME               ROLE               AGE
   rolebinding-read   ClusterRole/view   20s
```

3.4、ClusterRoleBinding：集群角色绑定

在 Kubernetes 中，ClusterRoleBinding 资源对象用于将一个或多个 Subject 与一个 ClusterRole 绑定。它允许在 Kubernetes 集群范围内分配权限，而不是单个命名空间中的权限。ClusterRoleBinding 是一个集群级的 RBAC 规则，可以控制任何命名空间中的对象所使用的权限。

使用 ClusterRoleBinding 时需要注意以下几点：

1、需要先有一个 ClusterRole 资源对象，该 ClusterRole 对象定义了具体的权限。

2、需要指定绑定到 ClusterRole 的 Subjects。

3、需要指定 ClusterRole 资源对象的 name 和 kind 属性，以及 API 组。

4、与其它 Kubernetes 资源对象一样，ClusterRoleBinding 也需要使用 yaml 或 json 等格式的文件进行编写和创建。

● 实战演练

实例 1：将 cluster-admin 集群角色与 dev Service Account 进行绑定

```
（1）查看 clusterrole
   [root@k8s-master01 ~]# kubectl get clusterrole cluster-admin
   NAME            CREATED AT
   cluster-admin   2023-05-24T10:01:58Z

（2）创建 sa
   [root@k8s-master01 ~]# kubectl create serviceaccount dev
   serviceaccount/dev created

（3）创建 ClusterRoleBinding 资源清单文件
   [root@k8s-master01 ~]# vi clusterrolebinding-admin.yaml
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
     name: clusterrolebinding-admin
   subjects:
```

```
  - kind: ServiceAccount
    name: dev
    namespace: default
  roleRef:
    kind: ClusterRole
    name: cluster-admin
    apiGroup: rbac.authorization.k8s.io
```

对上面的 yaml 文件说明：

这是一个 Kubernetes 集群角色绑定（ClusterRoleBinding）的 YAML 配置文件。

这个配置文件定义了一个名为 clusterrolebinding-admin 的角色绑定资源，它将 dev 命名空间中的 ServiceAccount 对象与一个名为 cluster-admin 的集群角色绑定起来。这意味着，当 dev 命名空间中的任何一个 Pod 使用这个 ServiceAccount 访问 Kubernetes API 时，它们都将被授予 cluster-admin 角色所包含的所有权限。

具体来说，这个 cluster-admin 角色允许用户执行 Kubernetes 集群中的任何操作，包括创建、修改、删除、查看和管理所有资源和对象。这是一个非常强大的权限，需要谨慎使用。

（4）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f clusterrolebinding-admin.yaml
clusterrolebinding.rbac.authorization.k8s.io/clusterrolebinding-admin created
```

3.5、常见的角色绑定示例

Kubernetes 中的角色绑定（Role Binding）是一种将角色与用户或组相关联的机制。下面是一些常见的 Kubernetes 角色绑定示例：

- 1、给命名空间管理员授予权限

此示例将 Namespace 中的用户命名为 namespace-admin，并授予该用户在该命名空间内创建、更新和删除所有资源的权限。如果需要授予更细粒度的权限，则可以创建更多的角色，并将其绑定到具体的用户或组上。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: namespace-admin
  namespace: mynamespace
subjects:
- kind: User
  name: namespace-admin
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: admin
```

```
    apiGroup: rbac.authorization.k8s.io
```

- 2、给组授权

此示例给名为 dev-team 的组授权，使其可以在指定命名空间创建、更新和删除 Pod 和 Service 资源。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-team
  namespace: mynamespace
subjects:
- kind: Group
  name: dev-team
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-service-access
  apiGroup: rbac.authorization.k8s.io
```

- 3、给 Service Account 授权

此示例将名称为 dev-ming 的 Service Account 绑定到集群角色 cluster-admin 上。这将授予该 sa dev-ming 用户 ClusterAdmin 权限，允许它执行集群级别的任何操作。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cluster-admins
subjects:
- kind: ServiceAccount
  name: dev-ming
  namespace: default
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
```

- 4、为 test 命名空间中的所有 Service Account 授权

以下是一个为 test 命名空间中所有 Service Account 进行角色绑定的 Kubernetes YAML 示例。该示例将绑定名为 cluster-admin 的 ClusterRole 到 test 命名空间中所有的 Service Account。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
```

```
  metadata:
    name: test-cluster-admin
  subjects:
  - kind: Group
    name: system:serviceaccounts:test
    apiGroup: rbac.authorization.k8s.io
  roleRef:
    kind: ClusterRole
    name: cluster-admin
    apiGroup: rbac.authorization.k8s.io
```

- 5、为所有命名空间中的 Service Account 授权

以下是一个给所有命名空间中的 Service Account 做角色绑定的 Kubernetes YAML 示例。该示例将绑定名为 namespace-admin 的集群角色到所有命名空间中的 Service Account。

```
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: namespace-admin
  subjects:
  - kind: Group
    name: system:serviceaccounts
    apiGroup: rbac.authorization.k8s.io
  roleRef:
    kind: ClusterRole
    name: namespace-admin
    apiGroup: rbac.authorization.k8s.io
```

4、Kubectl 命令行工具创建资源对象

4.1、Kubectl 对 Service Account 操作

- 语法

```
Usage:
  kubectl create serviceaccount NAME [options]

Options:
  --dry-run：在客户端模式下运行该命令，不会在 Kubernetes 中创建任何资源。
  --namespace string, -n string：指定命名空间，默认为“default”。
  --output string, -o string：输出格式，默认为“yaml”。可以是 json 或 yaml。
```

- 示例

实例 1：创建 ServiceAccount 对象，并将其保存到名为 sa.yaml 的文件中，但不会在 Kubernetes 中创建新的 Service Account。

```
[root@k8s-master01 ~]# kubectl create serviceaccount test -o yaml --dry-run=client > sa.yaml

[root@k8s-master01 ~]# cat sa.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: null
  name: test

[root@k8s-master01 ~]# kubectl get sa test
Error from server (NotFound): serviceaccounts "test" not found
```

下面是命令中使用的参数的详细解释：

```
test：新 Service Account 的名称为 test。
-o yaml：将输出格式设置为 YAML。
--dry-run=client：在客户端模式下运行该命令，不会在 Kubernetes 中创建任何资源。
>：将输出重定向到一个名为 sa.yaml 的文件中。

因此，执行该命令后，你将在当前工作目录中看到一个名为 sa.yaml 的文件，其中包含了 test Service Account 的 YAML 定义。你可以使用 kubectl apply -f sa.yaml 命令来创建该 Service Account。
```

实例 2：在 kube-system 命名空间下创建 test-sa serviceaccount

```
[root@k8s-master01 ~]# kubectl create sa test-sa -n kube-system
serviceaccount/test-sa created

[root@k8s-master01 ~]# kubectl get sa test-sa -n kube-system
NAME      SECRETS   AGE
test-sa   0         11s
```

4.2、Kubectl 对 Role 操作

- 语法

```
Usage:
  kubectl create role NAME --verb=verb --resource=resource [--resource-name=resname] [options]

Options:
  --dry-run：在客户端模式下运行该命令，不会在 Kubernetes 中创建任何资源。
  --namespace string, -n string：指定命名空间，默认为“default”。
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    --output string, -o string：输出格式，默认为“yaml”。可以是 json 或 yaml。
    --resource：指定新角色对象允许使用的资源集合。
    --resource-name：允许使用的资源集合中具体的资源项名称。
    --verb：指定新角色对象允许的操作集合。通常使用动词（get、list、watch、create、update、patch、delete）指定。
```

● 示例

实例 1：创建 Role 对象，并将其保存到名为 role.yaml 的文件中，但不会在 Kubernetes 中创建新的 Role。

```
    [root@k8s-master01 ~]# kubectl create role test --verb=get,list,watch --resource=pods,deployments --resource-name=web,storage -o yaml --dry-run=client > role.yaml

    [root@k8s-master01 ~]# cat role.yaml
    apiVersion: rbac.authorization.k8s.io/v1
    kind: Role
    metadata:
      creationTimestamp: null
      name: test
    rules:
    - apiGroups:
      - ""
      resourceNames:
      - web,storage
      resources:
      - pods
      verbs:
      - get
      - list
      - watch
    - apiGroups:
      - apps
      resourceNames:
      - web,storage
      resources:
      - deployments
      verbs:
      - get
      - list
      - watch
```

对上面的命令解释如下：

```
    这个命令会输出一个 YAML 格式的 Role 对象，并将其保存到名为 role.yaml 的文件中，
```

同时不会在 Kubernetes 中创建新的 Role 对象。你可以使用该命令来预览或导出 Role 对象的定义，然后在需要时手动创建或导入。

下面是命令中使用的参数的详细解释：

test：新 Role 对象的名称为 test。

--verb=get,list,watch：新 Role 对象允许的操作集合为 get、list、watch。

--resource=pods,deployments:新 Role 对象允许使用的资源集合为 pods、deployments。

--resource-name=web,storage：允许使用的资源集合中具体的资源项名称为 web、storage。

-o yaml：将输出格式设置为 YAML。

--dry-run=client：在客户端模式下运行该命令，不会在 Kubernetes 中创建任何资源。

>：将输出重定向到一个名为 role.yaml 的文件中。

因此，执行该命令后，你将在当前工作目录中看到一个名为 role.yaml 的文件，其中包含了 test Role 对象的 YAML 定义。你可以使用 kubectl apply -f role.yaml 命令来创建该 Role 对象。

实例 2：在 kube-system 命名空间下创建 test-web role，只能对读取 pods 资源。

```
[root@k8s-master01 ~]# kubectl create role test-web --verb=get,list,watch --resource=pods -n kube-system
role.rbac.authorization.k8s.io/test-web created

[root@k8s-master01 ~]# kubectl get role test-web -n kube-system
NAME       CREATED AT
test-web   2023-06-13T03:12:35Z
```

4.3、Kubectl 对 ClusterRole 操作

- 语法

```
Usage:
  kubectl create clusterrole NAME [--verb=verb] [--resource=resource [--resource-name=resname]] [options]

Options:
  --dry-run：在客户端模式下运行该命令，不会在 Kubernetes 中创建任何资源。
  --namespace string, -n string：指定命名空间，默认为“default”。
  --output string, -o string：输出格式，默认为“yaml”。可以是 json 或 yaml。
  --resource：指定新角色对象允许使用的资源集合。
  --resource-name：允许使用的资源集合中具体的资源项名称。
  --verb：指定新角色对象允许的操作集合。通常使用动词（get、list、watch、create、update、patch、delete）指定。
```

● 示例

实例 1：创建 ClusterRole 对象，并将其保存到名为 clusterrole.yaml 的文件中，但不会在 Kubernetes 中创建新的 ClusterRole。

```
[root@k8s-master01 ~]# kubectl create clusterrole test --verb=* --resource=* -o yaml --dry-run=client > clusterrole.yaml

[root@k8s-master01 ~]# cat clusterrole.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: test
rules:
- apiGroups:
  - ""
  resources:
  - '*'
  verbs:
  - '*'
```

实例 2：在 kube-system 命名空间下创建 web ClusterRole，只能读取 nodes 资源。

```
[root@k8s-master01 ~]# kubectl create clusterrole web --verb=get,list,watch --resource=nodes -n kube-system
clusterrole.rbac.authorization.k8s.io/web created

[root@k8s-master01 ~]# kubectl get clusterrole web -n kube-system
NAME   CREATED AT
web    2023-06-13T03:50:37Z
```

4.4、Kubectl 对 RoleBinding 操作

● 语法

```
Usage:
  kubectl create rolebinding NAME --role=role [--user=username | --group=groupname | --serviceaccount=namespace:serviceaccount ] [options]

Options:
  --dry-run：在客户端模式下运行该命令，不会在 Kubernetes 中创建任何资源。
  --namespace string, -n string：指定命名空间，默认为“default”。
  --output string, -o string：输出格式，默认为“yaml”。可以是 json 或 yaml。
  --role：指定要绑定的角色对象的名称。
```

```
    --user：指定被绑定用户的名称。
    --group：指定被绑定用户组的名称。
    --serviceaccount：指定被绑定 serviceaccount 的信息。
```

● 示例

实例 1：创建 RoleBinding 对象，并将其保存到名为 rolebinding.yaml 的文件中，但不会在 Kubernetes 中创建新的 RoleBinding。

```
    [root@k8s-master01 ~]# kubectl create rolebinding test --role=test
--serviceaccount=default:test -o yaml --dry-run=client > rolebinding.yaml

    [root@k8s-master01 ~]# cat rolebinding.yaml
    apiVersion: rbac.authorization.k8s.io/v1
    kind: RoleBinding
    metadata:
      creationTimestamp: null
      name: test
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: Role
      name: test
    subjects:
    - kind: ServiceAccount
      name: test
      namespace: default
```

对上面的命令解释如下：

```
    这个命令会输出一个 YAML 格式的 RoleBinding 对象，并将其保存到名为 rolebinding.yaml 的文件中，同时不会在 Kubernetes 中创建新的 RoleBinding 对象。

下面是命令中使用的参数的详细解释：
    test：新 RoleBinding 对象的名称为 test。
    --role=test：指定要绑定的 Role 对象名称为 test。
    --serviceaccount=default:test：指定被绑定的 ServiceAccount 名称为 test，所属 Namespace 为 default。
    -o yaml：将输出格式设置为 YAML。
    --dry-run=client：在客户端模式下运行该命令，不会在 Kubernetes 中创建任何资源。
    >：将输出重定向到一个名为 rolebinding.yaml 的文件中。

    因此，执行该命令后，你将在当前工作目录中看到一个名为 rolebinding.yaml 的文件，其中包含了 test RoleBinding 对象的 YAML 定义。你可以使用 kubectl apply -f rolebinding.yaml 命令来创建该 RoleBinding 对象。
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

4.5、Kubectl 对 ClusterRoleBinding 操作

- 语法

```
Usage:
    kubectl    create    clusterrolebinding    NAME    --clusterrole=clusterrole
[--user=username]        [--group=groupname]        [--serviceaccount=ns:name]
[--dry-run=server|client|none] [options]

Options:
    --dry-run：在客户端模式下运行该命令，不会在 Kubernetes 中创建任何资源。
    --namespace string, -n string：指定命名空间，默认为“default”。
    --output string, -o string：输出格式，默认为“yaml”。可以是 json 或 yaml。
    --user：指定被绑定用户的名称。
    --group：指定被绑定用户组的名称。
    --serviceaccount：指定被绑定 serviceaccount 的信息。
```

- 示例

实例 1：创建 ClusterRoleBinding 对象，并将其保存到名为 clusterrolebinding.yaml 的文件中，但不会在 Kubernetes 中创建新的 ClusterRoleBinding。

```
    [root@k8s-master01    ~]#    kubectl    create    clusterrolebinding    test
--clusterrole=test  --serviceaccount=default:test  -o  yaml  --dry-run=client  >
clusterrolebinding.yaml

    [root@k8s-master01 ~]# cat clusterrolebinding.yaml
    apiVersion: rbac.authorization.k8s.io/v1
    kind: ClusterRoleBinding
    metadata:
      creationTimestamp: null
      name: test
    roleRef:
      apiGroup: rbac.authorization.k8s.io
      kind: ClusterRole
      name: test
    subjects:
    - kind: ServiceAccount
      name: test
      namespace: default
```

5、限制不同的用户操作 k8s 集群

- 场景引入

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

场景 1：

公司现在要做一个 CTF 项目平台业务，业务代码需要调用 k8s api 接口去创建 CTF 场景题（一道场景题就是一个 Pod）。

那么就可以使用 k8s RBAC 创建一个 config 授权文件，业务代码指定 config 授权文件就可以调用 k8s 某些资源。

场景 2：

假设公司来了一名新员工 xiaozhang，招的是一名初级运维，没什么生产经验，就给他某个名称空间只读权限。

● 实战

（1）创建证书

1、创建证书存放位置

```
[root@k8s-master01 ~]# mkdir /dev-users
[root@k8s-master01 ~]# cd /dev-users/
```

2、创建 zhangsan 用户相关文件存放位置

```
[root@k8s-master01 dev-users]# mkdir zhangsan
[root@k8s-master01 dev-users]# cd zhangsan/
```

2、给 zhangsan 用户创建一个私钥

```
[root@k8s-master01 zhangsan]# (umask 077; openssl genrsa -out zhangsan.key 2048)
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
```

3、使用我们刚刚创建的私钥创建一个证书签名请求文件：zhangsan.csr，要注意需要确保在-subj 参数中指定用户名和组（CN 表示用户名，O 表示组）

```
[root@k8s-master01 zhangsan]# openssl req -new -key zhangsan.key -out zhangsan.csr -subj "/CN=zhangsan/O=yanfa"
```

4、拷贝 Kubernetes 集群的 ca 根证书

我们使用的是 kubeadm 安装的集群，CA 相关证书位于/etc/kubernetes/pki/目录下面，如果你是二进制方式搭建的，你应该在最开始搭建集群的时候就已经指定好了 CA 的目录，我们会利用该目录下面的 ca.crt 和 ca.key 两个文件来颁发上面的证书请求，生成最终的证书文件，我们这里设置证书的有效期为 3650 天。

```
[root@k8s-master01 zhangsan]# cp /etc/kubernetes/pki/ca.crt .
[root@k8s-master01 zhangsan]# cp /etc/kubernetes/pki/ca.key .
[root@k8s-master01 zhangsan]# ll
total 16
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    -rw-r--r-- 1 root root 1099 Jun 13 03:55 ca.crt
    -rw------- 1 root root 1679 Jun 13 03:55 ca.key
    -rw-r--r-- 1 root root  911 Jun 13 03:51 zhangsan.csr
    -rw------- 1 root root 1675 Jun 13 03:50 zhangsan.key

5、颁发证书
    [root@k8s-master01 zhangsan]# openssl x509 -req -in zhangsan.csr -CA ca.crt
-CAkey ca.key -CAcreateserial -out zhangsan.crt -days 3650
    Signature ok
    subject=/CN=zhangsan/O=yanfa
    Getting CA Private Key

6、查看证书文件
    [root@k8s-master01 zhangsan]# ll
    total 24
    -rw-r--r-- 1 root root 1099 Jun 13 03:55 ca.crt
    -rw------- 1 root root 1679 Jun 13 03:55 ca.key
    -rw-r--r-- 1 root root   17 Jun 13 03:56 ca.srl
    -rw-r--r-- 1 root root  997 Jun 13 03:56 zhangsan.crt
    -rw-r--r-- 1 root root  911 Jun 13 03:51 zhangsan.csr
    -rw------- 1 root root 1675 Jun 13 03:50 zhangsan.key
```

（2）在 kubeconfig 下新增加一个 zhangsan 这个用户

```
1、把 zhangsan 这个用户添加到 kubernetes 集群中，可以用来认证 apiserver 的连接
    [root@k8s-master01 zhangsan]# kubectl config set-credentials zhangsan \
    --client-certificate=./zhangsan.crt \
    --client-key=./zhangsan.key \
    --embed-certs=true
    User "zhangsan" set.

对上面的命令说明如下：
    该命令用于为 Kubernetes 集群中的一个用户创建一组客户端证书和密钥，并将这些证书和密钥保存到本地 kubeconfig 文件中，以便进行身份验证。
    下面是命令中使用的参数的详细解释：
    ● zhangsan：指定要创建的用户名称为 zhangsan。
    ● --client-certificate=./zhangsan.crt：指定客户端证书文件的路径，该文件包含了经过签名的证书，用于认证该用户。
    ● --client-key=./zhangsan.key：指定客户端密钥文件的路径，该文件包含了用于签名和加密通信的私钥。
    ● --embed-certs=true：指定是否将客户端证书嵌入到 kubeconfig 文件的 credentials 条目中，在该条目中引用证书文件的路径。如果设置为 false，则将在 credentials 条目中只引用证书文件的路径。
```

2、上条命令成功执行后，kubeconfig 文件会更新为包含新的用户信息。可以使用如下命令来查看和验证是否正确添加了该用户的证书和密钥。

```
[root@k8s-master01 zhangsan]# kubectl config view
```

```
[root@k8s-master01 zhangsan]# kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.128.11:6443
  name: kubernetes
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
current-context: kubernetes-admin@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED
- name: zhangsan
  user:
    client-certificate-data: DATA+OMITTED
    client-key-data: DATA+OMITTED
[root@k8s-master01 zhangsan]#
```

3、在 kubeconfig 下新增加一个 zhangsan 这个账号

```
[root@k8s-master01 zhangsan]# kubectl config set-context \
zhangsan@kubernetes \
--cluster=kubernetes \
--user=zhangsan
Context "zhangsan@kubernetes" created.
```

对上面的命令说明如下：

该命令用于为 Kubernetes 集群中的一个用户创建一个上下文，以便在使用该用户进行身份验证时可以轻松地切换到该用户的上下文进行工作。

下面是命令中使用的参数的详细解释：

- zhangsan@kubernetes：指定要创建的上下文名称为 zhangsan@kubernetes，其中 zhangsan 是用户的名称，kubernetes 是集群的名称。
- --cluster=kubernetes：指定该上下文所对应的集群名称为 kubernetes，该集群名称需要与 kubeconfig 文件中的集群名称一致。
- --user=zhangsan：指定该上下文所对应的用户名称为 zhangsan，该用户需要在 kubeconfig 文件中事先定义。

执行该命令后，kubeconfig 文件中就会创建一个新的上下文，其中包含了与该用户相关联的所需信息，如证书、密钥、集群信息等等。接下来，你可以使用 kubectl config use-context zhangsan@kubernetes 命令来切换到该用户的上下文进行操作。

（3）赋予 zhangsan 用户对 dev 名称空间的 pods、deployments、services 资源有所有权限。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

1、创建命名空间

```
[root@k8s-master01 zhangsan]# kubectl create ns dev
namespace/dev created

[root@k8s-master01 zhangsan]# kubectl get ns dev
NAME   STATUS   AGE
dev    Active   4s
```

2、创建一个 role 角色，设置对应权限

```
[root@k8s-master01 zhangsan]# vi zhangsan-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: zhangsan-role
  namespace: dev
rules:
- apiGroups: ["", "apps"]
  resources: ["deployments", "replicasets", "pods", "services"]
  verbs: ["*"]

[root@k8s-master01 zhangsan]# kubectl apply -f zhangsan-role.yaml
role.rbac.authorization.k8s.io/zhangsan-role created
```

3、创建 rolebinding，将角色绑定给用户

```
[root@k8s-master01 zhangsan]# vi zhangsan-rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: zhangsan-rolebinding
  namespace: dev
subjects:
- kind: User
  name: zhangsan
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: zhangsan-role
  apiGroup: rbac.authorization.k8s.io

[root@k8s-master01 zhangsan]# kubectl apply -f zhangsan-rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/zhangsan-rolebinding created
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

（4）测试权限

1、切换集群上下文，切换为我们刚才创建的集群上下文

```
[root@k8s-master01 zhangsan]# kubectl config use-context zhangsan@kubernetes
Switched to context "zhangsan@kubernetes".
```

2、测试查看权限

```
# 测试查看 kube-system 命名空间下的 pod，发现没权限
[root@k8s-master01 zhangsan]# kubectl get pods -n kube-system
```

```
[root@k8s-master01 zhangsan]# kubectl get pods -n kube-system
Error from server (Forbidden): pods is forbidden: User "zhangsan" cannot list resource "pods" in API group "" in the namespace "kube-system"
[root@k8s-master01 zhangsan]#
```

```
# 测试查看 dev 命名空间下的 pod，拥有权限
[root@k8s-master01 zhangsan]# kubectl get pods -n dev
No resources found in dev namespace.
```

3、创建测试 pod 和 service

```
[root@k8s-master01 zhangsan]# cat zhangsan-pod-svc.yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  namespace: dev
  labels:
    app: myweb
spec:
  containers:
  - name: myweb
    image: nginx
---
apiVersion: v1
kind: Service
metadata:
  name: test-pod
  namespace: dev
  labels:
    app: myweb
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
```

```
      nodePort: 30880
    selector:
      app: myweb

[root@k8s-master01 zhangsan]# kubectl apply -f zhangsan-pod-svc.yaml
pod/test-pod created
service/test-pod created

4、查看 pod 和 svc
[root@k8s-master01 zhangsan]# kubectl get pods -n dev
NAME       READY   STATUS    RESTARTS   AGE
test-pod   1/1     Running   0          4m12s

[root@k8s-master01 zhangsan]# kubectl get svc -n dev
NAME       TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
test-pod   NodePort   10.10.42.4    <none>        80:30880/TCP   4m15s

5、测试删除 pod 和 svc
[root@k8s-master01 zhangsan]# kubectl delete -f zhangsan-pod-svc.yaml
pod "test-pod" deleted
service "test-pod" deleted

6、切换到 kubernetes-admin@kubernetes 集群上下文
[root@k8s-master01 zhangsan]# kubectl config use-context
kubernetes-admin@kubernetes
Switched to context "kubernetes-admin@kubernetes".
```

（5）创建实体化用户的登陆权限

```
1、添加一个 zhangsan 的普通用户
[root@k8s-master01 zhangsan]# useradd zhangsan
[root@k8s-master01 zhangsan]# passwd zhangsan

2、拷贝集群认证文件到用户家目录下
[root@k8s-master01 zhangsan]# cp -ar /root/.kube/ /home/zhangsan/
[root@k8s-master01 zhangsan]# chown -R zhangsan.zhangsan /home/zhangsan/

3、为了提高安全性，进入到 config 中删除掉 kubernetes-admin 信息
[root@k8s-master01 zhangsan]# vi /home/zhangsan/.kube/config
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
contexts:
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
- context:
    cluster: kubernetes
    user: xiaozhang
  name: xiaozhang@kubernetes
current-context: xiaozhang@kubernetes
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
```

```
users:
- name: kubernetes-admin
  user:
    client-certificate-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURFekNDQWZ1Z0F3SUJBZ0lJYVNJY
zVWL2hQR1F3RFFZSktvWklodmNOQVFFTEJRQXdGVEVUTUJFR0ExVUUKQXhNS2EzVmlaWEp1WlhSbGN6QWVGdzB5TVRFeU1EWXh
OREl6TlRoYUZ3MHlNakV5TURZeE5ESTBNREJhTURReApGekFWQmdOVkJBb1REbk41YzNSbGJUcHRZWE4wWlhKek1Sa3dGd1lEV
lFRREV4QnJkV0psY201bGRHVnpMV0ZrCmJXbHVNSUlCSWpBTkJna3Foa2lHOXcwQkFRUZBQU9DQVE4QU1JSUJDZ0tDQVFFQXl
DOXZKWVdZTjA3VDZJTVoKb2ExRXUwa0ZpL1FsZnNjcC9ENzNqbW5PR2s1dGpvazRSQzR6dGlXSjJ2R3BqVjRCV1cwM1NDUlViT
HNXWnVNMgptUjRVbFlXMC80eHI2dEJRQmx3a2wzWjJkUEpCT3krNS80T1lxdElrSjVuRE9uOVBGOTh1djJReUNCMkF2QVFQCk0
1SmhiNC8rV3VuUHVYbGJPT1o3WndWNzF1TmtMcVlTR1RMdUhoaEVDUm1HNDFSQmZINVJVRGlhN2tzdmcySWoKazZtc0xoZ1NzV
XU1ZGZuRjF4aFF1OS9mMGxJaU90TWJuaHl4YVNGZ09RYkQxOEFzaGdLcnNzVlI2elcyTXBsaQozN0pyOGF5UFNhZGlQd0x1aUt
Qd0hqcHE2ZkNxRHR3emdRZEtTQVZyOTNvWmRoVTRmOXRwNG1zRnNUMjhnNmtQCkRDbTQ0d0lEQVFBQm8wZ3dSakFPQmdOVkhRO
EJBZjhFQkFNQ0JhQXdFd1lEVlIwbEJBd3dDZ1lJS3dZQkJRVUgKQXdJd0h3WURWUjBqQkJnd0ZvQVUwTXl4MU9JU2E3S0xWc2o
vZEo0TnFndW02bTh3RFFZSktvWklodmNOQVFFTApCUUFEZ2dFQkFIM2JSNjExYVZrcUQ5ZW11cWR3eHFZbDIzVFUyNzFhcHpKT
UM5dThPUWZYei9WV1doN2d0UFN6Cmo4L1RjeTlFdW5EMnBHYjNEQTVRbGFrenRGYTJkVHdicW5RbUNHeGRPVFJja0dobW9XRDM
5dVRNNnRlRXVuamcKZW1rclY2a3VPS3Zha2Rib2VlbDRVTm93eWlNTmZocUlTdmhBZCtGNVkxcmZWSFc1REpUeGg0dG5QMHNOS
21lNQpFc1RZSkY2cHVqZU12bUZQQkZWOTQ3NURtWW00TnZHZ1d6TGliY3VVU2V6RjFYNVE2eEhWWU53SUFGQThFWGx1Cjh0MWZ
wTlNxTm5GWCtMbFZLU1V4aktKa1pNYWdBakRmc3R0eks0ZzZnZTFOSEJTeEdwVWFTSnBoeWNVenBLL3gKd1hqZERhV0hHbUczc
XlXTFB1aTJRcEVMOTYyRWtpRT0KLS0tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
```

4、修改 config 中默认上下文信息

```
[root@k8s-master01 zhangsan]# vi /home/zhangsan/.kube/config
```

```
contexts:
- context:
    cluster: kubernetes
    user: zhangsan
  name: zhangsan@kubernetes
current-context: zhangsan@kubernetes
kind: Config
preferences: {}
users:
- name: zhangsan
```

4、下载改好的 config 文件，可以发给研发部，这个时候在程序代码中指定 config 文件，就可以调用集群 api 接口了。

```
[root@k8s-master01 zhangsan]# sz /home/zhangsan/.kube/config
```

（6）以 zhangsan 这个用户登录到集群进行测试

1、将 yaml 文件拷贝到 zhangsan 家目录下

```
[root@k8s-master01 zhangsan]# cp zhangsan-pod-svc.yaml /home/zhangsan/
[root@k8s-master01 zhangsan]# chown -R zhangsan.zhangsan /home/zhangsan/
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
2、以 zhangsan 身份登录
    [C:\~]$ ssh zhangsan@192.168.128.11

3、查看 dev 命名空间下的 pod
    [zhangsan@k8s-master01 ~]$ kubectl get pods -n dev
    No resources found in dev namespace.

4、创建资源
    [zhangsan@k8s-master01 ~]$ kubectl apply -f zhangsan-pod-svc.yaml
    pod/test-pod created
    service/test-pod created

5、查看资源
    [zhangsan@k8s-master01 ~]$ kubectl get pods -n dev
    NAME       READY   STATUS    RESTARTS   AGE
    test-pod   1/1     Running   0          62s
    [zhangsan@k8s-master01 ~]$ kubectl get svc -n dev
    NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
    test-pod   NodePort   10.10.193.102   <none>        80:30880/TCP   65s

6、删除资源
    [zhangsan@k8s-master01 ~]$ kubectl delete -f zhangsan-pod-svc.yaml
    pod "test-pod" deleted
    service "test-pod" deleted
```

至此，如果公司来了新员工，我们可以通过这样的方法，让新员工只拥有一些低权限。

6、资源定义详解

6.1、ServiceAccount 定义详解

ServiceAccount 资源可以通过如下命令查看相关语法：

```
[root@k8s-master01 ~]# kubectl explain ServiceAccount
```

- ServiceAccount 资源说明

属性名称	取值类型	取值说明
apiVersion	string	Api 版本
kind	string	资源类型
metadata	Object	元数据
metadata.name	String	控制器的名称
metadata.namespace	String	控制器所属的命名空间，默认值为 default

metadata.labels[]	List	自定义标签列表
automountServiceAccountToken	boolean	automountServiceAccountToken 被设计用来控制 Pod 是否可以自动挂载 ServiceAccount Token。通常情况下，Pod 需要自动挂载该 Token，以使 Kubernetes API 通过该 Token 进行身份验证和授权，并且也便于用户在 Pod 内进行开发和测试。 当设置为`false`时,Pod 将无法自动挂载 ServiceAccount Token，需要手动挂载它。
imagePullSecrets.name	[]Object	它用于指定要使用的 Secret 对象的名称，以便从私有 Docker Registry 中拉取镜像。

6.2、Role 定义详解

Role 资源可以通过如下命令查看相关语法：

```
[root@k8s-master01 ~]# kubectl explain Role
```

- Role 资源说明

属性名称	取值类型	取值说明
apiVersion	string	Api 版本
kind	string	资源类型
metadata	Object	元数据
metadata.name	String	控制器的名称
metadata.namespace	String	控制器所属的命名空间，默认值为 default
metadata.labels[]	List	自定义标签列表
rules	[]Object	rules 描述了可以由该 Role 进行管理的资源类型以及允许进行的操作。
rules.apiGroups	[]string	它指定了哪些 API 组可以被 Role 对象管理。如果值是*，则表示所有 API 组都可以被管理。

rules.nonResourceURLs	[]string	用于指定可以访问的非资源型 API 路径，即不指向 Kubernetes 资源的 URL。
		
		例如，下面的 Role 对象定义允许访问 Kubernetes 中/version 和/healthz 这两个非资源型 API 路径：
		apiVersion: rbac.authorization.k8s.io/v1
		kind: Role
		metadata:
		  name: non-resource-reader
		  namespace: default
		rules:
		- nonResourceURLs: ["/version", "/healthz"]
		  verbs: ["get"]

rules.resourceName	[]string	它表示可以指定哪些资源对象的名称可以被一个 Role（或

s		ClusterRole）对象访问。如果不指定 resourceNames，则该 Role（或 ClusterRole）对象允许访问该资源类型的所有资源对象。
rules.resources	[]string	它指定了哪些资源类型可以被 Role 对象管理。如果值是*，则表示所有资源类型都可以被管理。
rules.verbs	[]string	它指定了 Role 所允许进行的操作。常见的操作包括 get、list、create、update、delete、watch 等等。

6.3、ClusterRole 定义详解

ClusterRole 资源可以通过如下命令查看相关语法：

```
[root@k8s-master01 ~]# kubectl explain ClusterRole
```

- ClusterRole 资源说明

属性名称	取值类型	取值说明
apiVersion	string	Api 版本
kind	string	资源类型
metadata	Object	元数据
metadata.name	String	控制器的名称
metadata.namespace	String	控制器所属的命名空间，默认值为 default
metadata.labels[]	List	自定义标签列表
rules	[]Object	rules 描述了可以由该 Role 进行管理的资源类型以及允许进行的操作。
rules.apiGroups	[]string	它指定了哪些 API 组可以被 Role 对象管理。如果值是*，则表示所有 API 组都可以被管理。

rules.nonResourceURLs	[]string	用于指定可以访问的非资源型 API 路径，即不指向 Kubernetes 资源的 URL。
		
		例如，下面的 ClusterRole 对象定义允许访问 Kubernetes 中/version 和/healthz 这两个非资源型 API 路径：
		apiVersion: rbac.authorization.k8s.io/v1
		kind: ClusterRole
		metadata:
		  name: non-resource-reader
		  namespace: default
		rules:
		- nonResourceURLs: ["/version", "/healthz"]
		  verbs: ["get"]

rules.resourceNames	[]string	它表示可以指定哪些资源对象的名称可以被一个 Role（或 ClusterRole）对象访问。如果不指定 resourceNames，则该 Role（或 ClusterRole）对象允许访问该资源类型的所有资源对象。
rules.resources	[]string	它指定了哪些资源类型可以被 Role 对象管理。如果值是*，则表

		示所有资源类型都可以被管理。
rules.verbs	[]string	它指定了 Role 所允许进行的操作。常见的操作包括 get、list、create、update、delete、watch 等等。

6.4、RoleBinding 定义详解

RoleBinding 资源可以通过如下命令查看相关语法：

```
[root@k8s-master01 ~]# kubectl explain RoleBinding
```

● RoleBinding 资源说明

属性名称	取值类型	取值说明
apiVersion	string	Api 版本
kind	string	资源类型
metadata	Object	元数据
metadata.name	String	控制器的名称
metadata.namespace	String	控制器所属的命名空间，默认值为 default
metadata.labels[]	List	自定义标签列表
roleRef	Object	在 Kubernetes 中，ClusterRoleBinding 对象定义了一组 Subject 对象与一个 Role 或 ClusterRole 之间的关系。Subject 对象可以是 User、Group 或 ServiceAccount。而 roleRef 是指定 Subject 对象使用的 Role 或 ClusterRole 的引用。
roleRef.apiGroup	string	Role 或 ClusterRole 所属的 API 组。 通常设置为“rbac.authorization.k8s.io”
roleRef.kind	string	资源的类型，通常为 Role 或 ClusterRole。
roleRef.name	string	要绑定的 Role 或 ClusterRole 的名称，这个名称指定了一个角色集合。
subjects	[]Object	subjects 字段用于指定一组 Subject 对象，可以是 User、Group 或 ServiceAccount。 将 Subject（主体）与 ClusterRole 绑定在一起，以便授予它们对 Kubernetes 资源的访问权限。
subjects.apiGroup	string	apiGroup 字段则是指定 Subject 对象所属的 API 组。默认为“rbac.authorization.k8s.io”。
subjects.kind	string	kind 字段指定 Subject 对象的类型。 Kubernetes 支持三种类型的 Subject： ● User：表示集群中的用户。可以使用用户名或 UID 来指定用户。 ● Group：表示集群中的用户组。可以使用组名或 GID 来指定用户组。 ● ServiceAccount：表示在 Kubernetes 中使用的服务账户。
subjects.name	string	name 属性的值取决于该对象的类型。例如，当 Subject 对象的类型为 User 时，name 属性表示用户名或用户 UID。当 Subject 对象

		是 Group 时，则表示组名或 GID。当 Subject 对象为 ServiceAccount 时，则代表 ServiceAccount 的名称。
subjects.namespace	string	namespace 用于指定该对象的类型所在的命名空间。

6.5、ClusterRoleBinding 定义详解

ClusterRoleBinding 资源可以通过如下命令查看相关语法：

```
[root@k8s-master01 ~]# kubectl explain ClusterRoleBinding
```

- ClusterRoleBinding 资源说明

属性名称	取值类型	取值说明
apiVersion	string	Api 版本
kind	string	资源类型
metadata	Object	元数据
metadata.name	String	控制器的名称
metadata.namespace	String	控制器所属的命名空间，默认值为 default
metadata.labels[]	List	自定义标签列表
roleRef	Object	在 Kubernetes 中，ClusterRoleBinding 对象定义了一组 Subject 对象与一个 ClusterRole 之间的关系。Subject 对象可以是 User、Group 或 ServiceAccount。而 roleRef 是指定 Subject 对象使用的 ClusterRole 的引用。
roleRef.apiGroup	string	ClusterRole 所属的 API 组。 通常设置为“rbac.authorization.k8s.io”
roleRef.kind	string	ClusterRole 的类型，通常为 ClusterRole。
roleRef.name	string	要绑定的 ClusterRole 的名称，这个名称指定了一个角色集合。
subjects	[]Object	subjects 字段用于指定一组 Subject 对象，可以是 User、Group 或 ServiceAccount。 将 Subject（主体）与 ClusterRole 绑定在一起，以便授予它们对 Kubernetes 资源的访问权限。
subjects.apiGroup	string	apiGroup 字段则是指定 Subject 对象所属的 API 组。默认为“rbac.authorization.k8s.io”。
subjects.kind	string	kind 字段指定 Subject 对象的类型。 Kubernetes 支持三种类型的 Subject： ● User：表示集群中的用户。可以使用用户名或 UID 来指定用户。 ● Group：表示集群中的用户组。可以使用组名或 GID 来指定用户组。 ● ServiceAccount：表示在 Kubernetes 中使用的服务账户。
subjects.name	string	name 属性的值取决于该对象的类型。例如，当 Subject 对象的类型为 User 时，name 属性表示用户名或用户 UID。当 Subject 对象是 Group 时，则表示组名或 GID。当 Subject 对象为

		ServiceAccount 时，则代表 ServiceAccount 的名称。
subjects.namespace	string	namespace 用于指定该对象的类型所在的命名空间。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。