RBAC权限控制

前言

- 本文主要介绍kubernetes中的认证授权流程，包括授权机制中的几个概念，RBAC的授权插件中的几个重要对象以及它们之间的关联关系，学会使用命令创建角色并赋权。

目标

- 学完本课程后，您将能够：
  - 描述kubernetes授权机制中的几个重要概念
  - 区分kubernetes中的useraccount和serviceaccount
  - 描述什么是RBAC授权插件
  - 学会如何创建账号，角色，并进行赋权

目录

访问控制概述

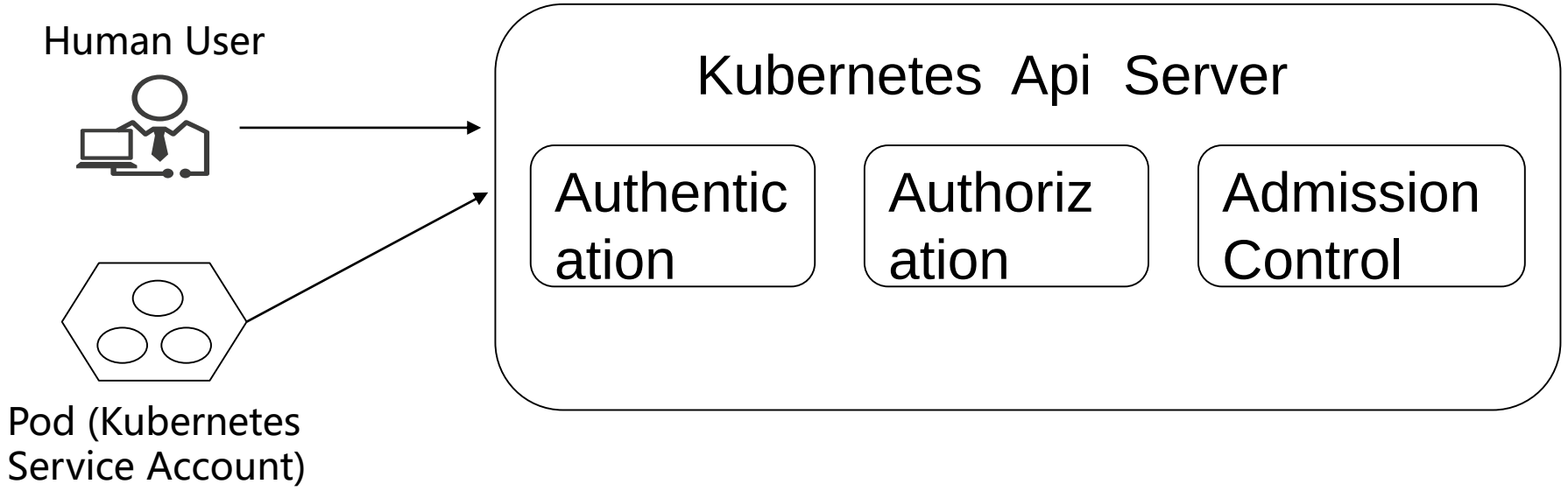
- 在任何提供资源或服务的系统上，认证和授权是两个不可或缺的功能，认证用于身份鉴定，即Authentication；授权用于实现权限分配，即Authorization；kubernetes以插件化的方式实现了这两种功能，常用的授权插件有以下几种：
  - Node（节点认证）
  - ABAC（基于属性的访问控制）
  - RBAC（基于角色的访问控制）
  - Webhook（基于http回调机制的访问控制）

访问控制概述

- API Server作为Kubernetes集群系统的网关，是访问及管理资源对象的唯一入口，其它所有需要访问集群资源的组件，包括kube-controller-manager、kube-scheduler、kubelet和kube-proxy等集群基础组件、CoreDNS等集群的附加组件以及此前使用的kubectl命令等，都需要经过此网关才能进行集群访问和管理。
- 以上这些客户端均要经由API Server访问或改变集群状态并完成数据存储，并由它对每一次的访问请求都需要进行合法性的检验，其中包括身份鉴别、操作权限验证以及操作规范验证等，所有检查均正常完成且对象配置信息合法性检验无误之后才能访问或存入数据于后端存储系统etcd中。

访问控制概述

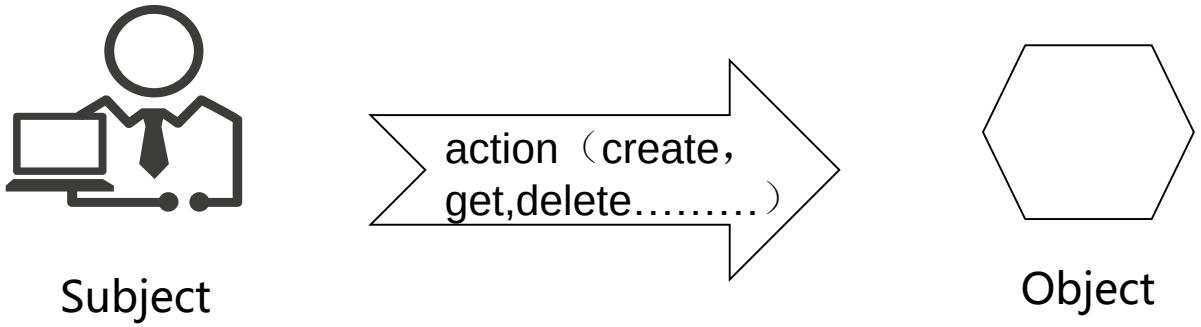
- 客户端认证操作由API Server配置的一个到多个认证插件完成。收到请求后，API Server依次调用为其配置的认证插件来认证客户端身份，直到其中一个插件可以识别出请求者的身份为止。
- 授权操作也由一个到多个授权插件进行，它负责确定那些通过认证的用户是否有权限执行发出的资源操作请求，如创建、读取、删除等。随后，通过授权检测的用户所发出的操作请求还要经由一个或多个准入控制插件遍历检测，比如检查目标Namespace资源对象是否存在、检查是否违反系统资源限制等等，其中任何的检查失败都会导致该操作失败。



User、Permission和Role

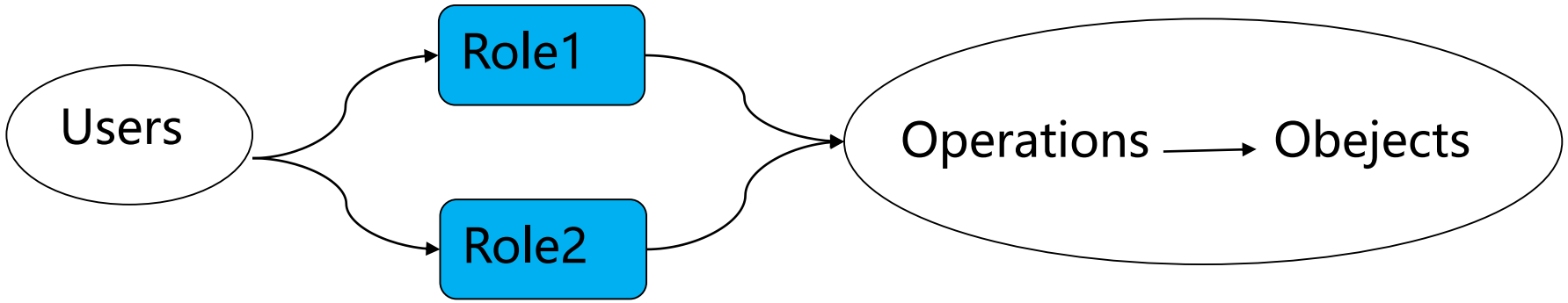
- 几个基础概念：
  - 用户（User）是一个可以独立访问计算机系统中的数据或者用数据表示的其它资源的主体（Subject）。
  - 许可（Permission）就是允许对一个或多个客体（object）执行的操作（action）。
  - 角色（Role）代表了一种权利、资格，可由多种许可构成。

注：请思考，User、permission和Role之间的关系



User、Permission和Role

- 一个User可经授权而拥有多个角色，每个角色可拥有多种许可，每种许可也可授权给多个不同的角色，每个操作可施加给多个客体（object），每个客体也可以接受多个操作。
- 在RBAC中基于“角色”（role）这一核心组件实现了权限指派，它为账号赋予一个到多个角色从而让其具有角色上的权限，其中的账户可能是用户账号、用户组、服务账号及其相关的组等，而同时关联多个角色的账号所拥有的权限是多个角色之上的权限集合。



Kubernetes中的用户和用户组

- 在Kubernetes中，客户端访问API服务的途径通常有三种，kubectl、客户端库或者直接使用REST接口进行请求，故可以执行这几类请求的主体（subject）也被kubernetes分为两类：
  - User Account（用户账号）：一般是指由独立于Kubernetes之外的其它服务管理的用户账号，Kubernetes中不存在表示此类用户账号的对象，作用于系统全局，名称必须全局唯一。
  - Service Account（服务账号）：为Pod中的进程调用Kubernetes API设计，通常需要绑定于特定的名称空间，作用仅局限在他们所在的Namespace，他们由API Server创建，或者通过API调用手动创建，附带着一组存储为Secret的用于访问API Server的凭据。
- 这两类账号都可以隶属于一个或多个用户组，用户组本身没有操作权限，但是附加在组上的权限可由其内部的所有用户继承，实现高效的授权管理机制。

创建UserAccount

- 在/etc/kubernetes/pki专用目录下，为用户（user1）生成私钥文件：

```
[root@master pki]# (umask 077;openssl genrsa -out user1.key 2048)
Generating RSA private key, 2048 bit long modulus
.....................................................+++
.+++
e is 65537 (0x10001)
```

- 接着用此私钥创建一个csr(证书签名请求)文件，其中我们需要在subject里带上用户信息(CN为用户名，O为用户组)，/O可多次出现，即可以属于多个组：

```
openssl req -new -key user1.key -out user1.csr -subj
"/CN=user1/O=kubeusers"
```

创建UserAccount

- 通过集群的CA证书和之前创建的csr文件，来为用户颁发证书，-CA和-CAkey参数需要指定集群CA证书所在位置（因该环境中证书就在当前路径下，故没有指定），-days参数指定此证书的过期时间，这里为365天。

```
openssl x509 -req -in user1.csr -CA ca.crt -CAkey ca.key -CAcreateserial
-out user1.crt -days 365
```

回显：

```
Signature ok
subject=/CN=user1/O=kubeusers
Getting CA Private Key
```

创建UserAccount

- User1用户已经创建好了，现在我们想要通过kubectl以user1的身份来操作集群，需要将user1的认证信息添加进kubectl的配置，即~/.kube/config中：

```
kubectl config set-credentials user1 --client-
certificate=/etc/kubernetes/pki/user1.crt --client-
key=/etc/kubernetes/pki/user1.key
```

- 配置context上下文，组合cluster和credentials，方便后续可以通过命令切换来使用user1管理指定的cluster：

```
kubectl config set-context user1@kubernetes --cluster=kubernetes --
user=user1
```

创建UserAccount

- 切换user1访问集群：

```
[root@master pki]# kubectl  config use-context user1@kubernetes
Switched to context "user1@kubernetes".
```

- 因为此时没有经过授权，故出现如下错误回显：

```
[root@master pki]# kubectl get pods
Error from server (Forbidden): pods is forbidden: User "user1" cannot
list resource "pods" in API group "" in the namespace "default"
```

注：如果需要切换至管理员账号，则可以使用“kubectl config use-context kubernetes-admin@kubernetes”命令完成

创建ServiceAccount

- 大部分时候我们很少接触UserAccount这种类型用户，更多使用和操作的是ServiceAccount这类用户。
- 通过yaml文件定义一个最简单的serviceaccount，一个最简单的ServiceAccount对象只需要Name和Namespace这两个最基本的字段：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  namespace: mynamespace
  name: example-sa
```

ServiceAccount

- Service account是为了方便Pod里面的进程调用Kubernetes API或其他外部服务而设计的。它与User account不同:
  - User account是为人设计的，而service account则是为Pod中的进程调用Kubernetes API而设计
  - User account是跨namespace的，而service account则是仅局限它所在的namespace
  - 每个namespace都会自动创建一个default service account
  - Token controller检测service account的创建，并为它们创建secret

ServiceAccount

- 开启ServiceAccount Admission Controller后
  - 每个Pod在创建后都会自动设置spec.serviceAccount为default（除非指定了其他ServiceAccout）
  - 验证Pod引用的service account已经存在，否则拒绝创建
  - 如果Pod没有指定ImagePullSecrets，则把service account的ImagePullSecrets加到Pod中
  - 每个container启动后都会挂载该service account的token和ca.crt到/var/run/secrets/kubernetes.io/serviceaccount/

ServiceAccount

- 当创建 pod 的时候，如果没有指定一个 service account，系统会自动在与该pod相同的namespace下为其指派一个default service account。而pod和apiserver之间进行通信的账号，称为serviceAccountName。

```
[root@master RBAC]# kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
httpd-686c9d9595-2mq6t        1/1     Running   2          38d
httpd-686c9d9595-clkjj        1/1     Running   2          38d
httpd-686c9d9595-zf7lg        1/1     Running   0          38d
test-mysql-575697c6f7-bw84f   1/1     Running   2          3d1h
[root@master RBAC]#
[root@master RBAC]# kubectl get pods httpd-686c9d9595-2mq6t -o yaml|grep serviceAccountName
  serviceAccountName: default
```

ServiceAccount

- 每个Pod无论定义与否都会有个存储卷，这个存储卷是名为为default-token-***token令牌，这就是pod和serviceaccount认证信息。通过secret进行定义，由于认证信息属于敏感信息，所以需要保存在secret资源当中，并以存储卷的方式挂载到Pod当中。从而让Pod内运行的应用通过对应的secret中的信息来连接apiserver，并完成认证。

```
kubectl describe pods XXX
```

回显：

```
Volumes:
  default-token-6wm9r:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-6wm9r
```

ServiceAccount

- 每个namespace中都有一个默认的叫做default的ServiceAccount资源。这个secret，就是对应的ServiceAccount对应的、用来跟APIServer进行交互的授权文件，我们一般称为：Token。Token文件的内容一般是证书或者密码，它以一个Secret对象的方式保存在Etcd中。
- 进行查看名称空间内的secret，也可以看到对应的default-token。

```
[root@master RBAC]# kubectl get sa
NAME      SECRETS   AGE
default   1         44d
[root@master RBAC]# kubectl get sa -n mynamespace
NAME      SECRETS   AGE
default   1         25h
[root@master RBAC]# kubectl get secrets
NAME                  TYPE                                  DATA   AGE
default-token-6wm9r   kubernetes.io/service-account-token   3      44d
test-mysql            Opaque                                2      3d1h
[root@master RBAC]# kubectl get secrets -n mynamespace
NAME                  TYPE                                  DATA   AGE
default-token-8dlsg   kubernetes.io/service-account-token   3      25h
```

ServiceAccount

- 默认的ServiceAccount并没有关联任何Role，也就是说，此时它有访问APIServer的绝大多数权限，但无法观察到其他名称空间Pod的相关属性信息。假设有一个Pod需要用于管理其他Pod或者是其他资源对象，是无法通过自身的名称空间的ServiceAccount进行获取其他Pod的相关属性信息的，此时就需要进行手动创建一个ServiceAccount，并在创建Pod时进行引用。

```
apiVersion: v1
kind: Pod
metadata:
……
spec:
  containers:
  - name: myapp
    image: myapp:v1
    ports:
    - name: http
      containerPort: 80
  serviceAccountName: admin
```

目录

RBAC是什么

- 在Kubernetes中，负责完成授权（Authorization）工作的机制，就是RBAC：基于角色的访问控制（Role-Based Access Control）。
- 它用于界定“主体”（subject）能够或不能够“操作”（operate）哪个或哪类“对象”（object）。动作的发出者是subject，通常是常规用户（User Account），也可以是服务账户（Service Account）。“操作”（operate）用于表明要执行的具体操作，包括创建、删除、修改和查看（create、apply、delete、update、patch、edit和get等）。而“客体”（object）则是指操作施加的资源对象，对Kubernetes API来说主要指各类的资源对象以及URL。

RBAC中的对象

- RBAC授权插件支持Role和ClusterRole两类角色，他们本质上都是标准的Kubernetes的API对象：
  - Role作用于名称空间级别，用于定义名称空间内的资源权限集合，可以通过编写yaml文件来创建一个Role对象。也可以直接使用“kubectl create role”的命令创建。

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: mynamespace
  name: example-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

```
kubectl create role example-role --verb="get,watch,list" --
resource="pods" -n mynamespace
```

RBAC中的对象

- Role中重要字段语意：
  - kind：定义对象类型，这里是Role
  - apiVersion：api的版本号
  - metadata：对象元数据
    - namespace：该role作用的范围
    - name：该role的名称
  - rules：定义了权限的集合
    - apiGroups：指定对哪些API群组内的资源操作，空串(“”)代表核心组。
    - resources：规则应用的目标资源类型组成的列表，ResourceAll表示所有资源。
    - Verbs：该规则下的操作列表，包括get，list，create，update，watch等。

RBAC中的对象

- RBAC授权插件支持Role和ClusterRole两类角色，他们本质上都是标准的Kubernetes的API对象：
  - ClusterRole作用于集群级别，用于集群级组件授权，可以通过编写yaml文件来创建一个Role对象。也可以直接使用“kubectl create clusterrole”的命令创建。

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: example-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

注：与role中的字段相比，基本保持一致，但因为ClusterRole属于集群级别资源，不属于名称空间，故在此处其配置不添加metadata.namespace字段

RBAC中的对象

- Role和ClusterRole只负责定义权限和作用范围，但角色本身并不是动作执行的主体，所以他们需要“绑定”（binding）到主体（如user，group或service account）上，这个绑定过程其实就是一个授与主体权限的过程。
- Kubernetes提供了两种绑定类型RoleBinding和ClusterRoleBinding，他们本质上也是kubernetes的API对象。
  - RoleBinding用于将Role或ClusterRole中定义的权限赋予一个或一组用户，但RoleBinding仅能够引用同一名称空间中的Role对象完成授权。RoleBinding可以通过编写yaml文件来创建，也可以直接使用“`kubectl create rolebinding`”的命令创建。

RBAC中的对象

- 使用yaml文件创建RoleBinding对象：

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: example-rolebinding
  namespace: mynamespace
subjects:
- kind: User
  name: user1
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: example-role
  apiGroup: rbac.authorization.k8s.io
```

- 使用命令创建RoleBinding对象：

```
kubectl create rolebinding example-rolebinding --role=example-role
--user=user1 --namespace=mynamespace
```

RBAC中的对象

- RoleBinding中重要字段语意：
  - Subjects下的字段：
    - apiGroup：保存了subject的API组，“ ”代表ServiceAccout默认组，“ rbac.authorization.k8s.io”为user或group默认组
    - kind：引用的主体资源类型，可以是”User”，“ Group”和”ServiceAccount”三个中的一个，必选
    - name：引用的主体的名称，必选
    - namespace：引用的主体所属的名称空间，对于User和Group这种全局类型的主体，namespace为空
  - RoleRef下的字段：
    - apiGroup：引用的资源（Role或ClusterRole）所属的API群组，必选
    - kind：引用的资源所属的类别，可以为Role或者ClusterRole，必选
    - name：引用的资源的名称

RBAC中的对象

- 使用yaml文件创建ClusterRoleBinding对象：

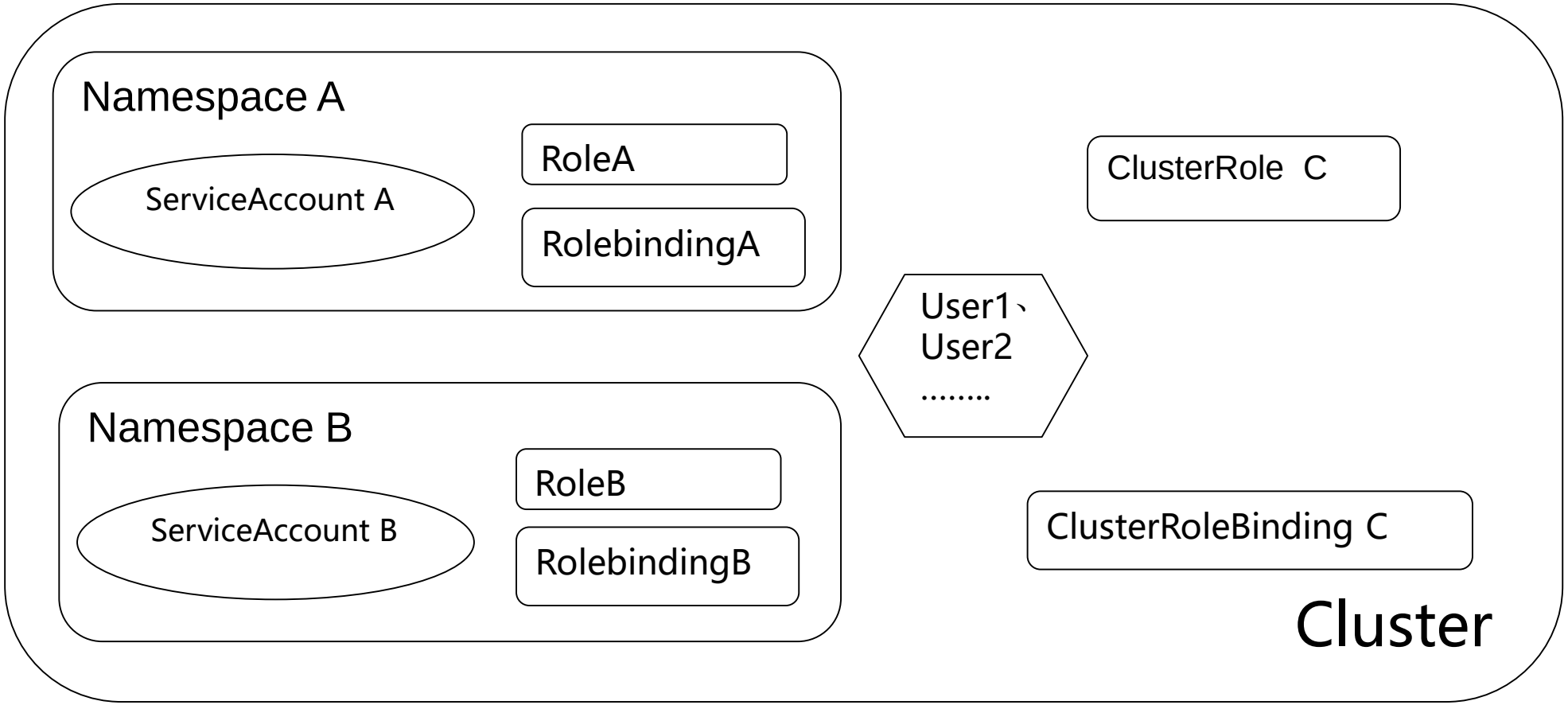
```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: example-clusterrolebinding
subjects:
- kind: User
  name: user1
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: example-clusterrole
  apiGroup: rbac.authorization.k8s.io
```

- 使用命令创建ClusterRoleBinding对象：

```
kubectl create clusterrolebinding example-clusterrolebinding --
clusterrole=example-clusterrole --user=user1
```

RBAC中的对象

- ClusterRolebinding，Rolebinding，ClusterRole，Role四种对象之间的关系如下图



内置ClusterRole和ClusterRolebinding

- 在Kubernetes中已经内置了很多个为系统保留的ClusterRole，他们的名字都是以system:开头，你可以通过`kubectl get clusterroles`和`kubectl get clusterrolebinding`查看他们。

```
[root@master RBAC]# kubectl get clusterroles
NAME                                                                   AGE
admin                                                                  44d
cluster-admin                                                          44d
edit                                                                   44d
elasticsearch-logging                                                  2d21h
example-clusterrole                                                    17h
flannel                                                                44d
system:aggregate-to-admin                                              44d
system:aggregate-to-edit                                               44d
system:aggregate-to-view                                               44d
system:auth-delegator                                                  44d
system:aws-cloud-provider                                              44d
system:basic-user                                                      44d
system:certificates.k8s.io:certificatesigningrequests:nodeclient       44d
system:certificates.k8s.io:certificatesigningrequests:selfnodeclient   44d
system:controller:attachdetach-controller                              44d
system:controller:certificate-controller                               44d
system:controller:clusterrole-aggregation-controller                   44d
system:controller:cronjob-controller                                   44d
system:controller:daemon-set-controller                                44d
system:controller:deployment-controller                                44d
system:controller:disruption-controller                                44d
system:controller:endpoint-controller                                  44d
```

内置ClusterRole和ClusterRolebinding

- 一般来说，这些系统ClusterRole是绑定给Kubernetes系统组件对应的ServiceAccount使用的，比如，其中一个名叫system:kube-scheduler的定义的权限规则是kubernetes调度器运行所需要的必要权限，你可以通过`kubectl describe clusterrole NAME`查看对应权限列表

```
[root@master RBAC]# kubectl describe clusterrole system:kube-scheduler
Name:         system:kube-scheduler
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources                                  Non-Resource URLs  Resource Names    Verbs
  ---------                                  -----------------  --------------    -----
  events                                     []                 []                [create patch update]
  bindings                                   []                 []                [create]
  endpoints                                  []                 []                [create]
  pods/binding                               []                 []                [create]
  tokenreviews.authentication.k8s.io         []                 []                [create]
  subjectaccessreviews.authorization.k8s.io  []                 []                [create]
  pods                                       []                 []                [delete get list watch]
  endpoints                                  []                 [kube-scheduler]  [delete get patch update]
  nodes                                      []                 []                [get list watch]
  persistentvolumeclaims                     []                 []                [get list watch]
  persistentvolumes                          []                 []                [get list watch]
  replicationcontrollers                     []                 []                [get list watch]
```

内置ClusterRole和ClusterRolebinding

- 这个system:kube-scheduler的ClusterRole，就会被绑定给kube-system Namespace下名叫kube-scheduler的ServiceAccount，它正是Kubernetes调度器的Pod声明使用的ServiceAccount。

```
[root@master RBAC]# kubectl describe clusterrole system:kube-scheduler
Name:         system:kube-scheduler
Labels:       kubernetes.io/bootstrapping=rbac-defaults
Annotations:  rbac.authorization.kubernetes.io/autoupdate: true
PolicyRule:
  Resources                                  Non-Resource URLs  Resource Names    Verbs
  ---------                                  -----------------  --------------    -----
  events                                     []                 []                [create patch update]
  bindings                                   []                 []                [create]
  endpoints                                  []                 []                [create]
  pods/binding                               []                 []                [create]
  tokenreviews.authentication.k8s.io         []                 []                [create]
  subjectaccessreviews.authorization.k8s.io  []                 []                [create]
  pods                                       []                 []                [delete get list watch]
  endpoints                                  []                 [kube-scheduler]  [delete get patch update]
  nodes                                      []                 []                [get list watch]
  persistentvolumeclaims                     []                 []                [get list watch]
  persistentvolumes                          []                 []                [get list watch]
  replicationcontrollers                     []                 []                [get list watch]
```

内置ClusterRole和ClusterRolebinding

- Kubernetes还提供了四个预先定义好的ClusterRole来供用户直接使用
  - cluster-admin：整个kubernetes中权限最高的角色，请谨慎使用
  - admin：规定了大部分资源（不包括node）的权限
  - edit：规定了大部分资源的修改权限的角色
  - view：规定了大部分资源只读权限的角色
- 掌握这些默认的内建ClusterRole对后期按需创建用户并快速配置权限带来极大便利。

思考题

1. 请思考：

对于以下这个Pod的yaml文件，它的权限是什么？什么时候对它进行了授权？

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
   name: httpd
spec:
   replicas: 3
   template:
     metadata:
       labels:
         run: httpd
     spec:
       containers:
       - name: httpd
         image: httpd
         ports:
         - containerPort: 80
```

实验&实训任务

- 实验任务
  - 请按照手册2.17章节完成RBAC相关实验，包括：
    - 创建User Account并设定kube-config文件
    - 为新建的User赋权
    - 创建Service Accout
    - 创建Pod并引用该Service Account
- 实训任务
  - 请灵活使用本章课程及实验手册中学到的知识，按照2.17.6章节完成实训任务

本章总结

- 本章介绍了kubernetes中有关认证和授权的相关知识，包括如下：
  - kubernetes中的访问控制流程
  - kubernetes中RBAC插件中的几个重要对象
  - 如何创建useraccount和serviceaccount
  - 如何创建role&clusterrole、rolebinding和clusterrolebinding


谢 谢
www.huawei.com