1.1 RBAC 权限控制

1.1.1 创建 User Account

步骤 1　查看当前用户认证信息

```
[root@k8s-master ~]# kubectl config view
```

```
输出回显:
apiVersion: v1
clusters:                          #集群信息，可能有多个集群
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://192.168.137.11:6443
  name: kubernetes
contexts:                  #context 定义 cluster,user,namespace 元组的名称,用来向指定的
定的集群使用提供的认证信息和命名空间向指定的集群发送请求。可以有多个 context
- context:
    cluster: kubernetes
    user: kubernetes-admin
  name: kubernetes-admin@kubernetes
- context:
    cluster: kubernetes
    user: user1
  name: user1@kubernetes
current-context: kubernetes-admin@kubernetes      #当前使用的账号所在的 context
kind: Config
preferences: {}
users:                             #用户清单
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
- name: user1
  user:
    client-certificate-data: REDACTED
   client-key-data: REDACTED
```

从配置中可以看出当前只有一个 cluster，我们现在使用的是名为 kubernetes-admin 的账户，用户列表中除了 kubernetes-admin，还有一个 user1 的用户。

步骤 2　在 kubernetes 默认存放用户秘钥和证书的目录里，为新用户 user2 账号生成秘钥，

```
[root@k8s-master ~]# cd /etc/kubernetes/pki
[root@k8s-master pki]# (umask 077; openssl genrsa -out user2.key 2048)
```

步骤 3　创建证书签署请求

```
[root@k8s-master pki]# openssl req -new -key user2.key -out user2.csr -subj
"/CN=user2/O=kubeusers"
```

步骤 4　使用 kubernetes 集群创建时候的 CA 为用户颁发证书，设置有效时间

```
[root@k8s-master pki]# openssl x509 -req -in user2.csr -CA ca.crt -CAkey ca.key
-CAcreateserial -out user2.crt -days 365
```

```
[root@master pki]# openssl x509 -req -in user2.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out user2.crt -days 365
Signature ok
subject=/CN=user2/O=kubeusers
Getting CA Private Key
```

1.1.2 使用 User Account 设定 kube-config 配置文件

步骤 1　配置客户端证书及秘钥

```
[root@k8s-master ~]# kubectl config set-credentials user2 --client-
certificate=/etc/kubernetes/pki/user2.crt --client-
key=/etc/kubernetes/pki/user2.key
```

```
User "user2" set.
```

步骤 2　配置 user2 的 context，组合 cluster 和 credentials，使得可以通过 user-context 来切换

```
[root@k8s-master ~]# kubectl config set-context user2@kubernetes --
cluster=kubernetes --user=user2
```

```
Context "user2@kubernetes" created
```

步骤 3　上下文切换

#切换至刚添加的新用户

```
[root@k8s-master ~]# kubectl config use-contex kube-user2@kubernetes
```

#切换回 admin 用户

```
[root@k8s-master ~]# kubectl config use-context kubernetes-admin@kubernetes
```

#临时切换

```
[root@k8s-master ~]# kubectl --context=user2@kubernetes get pods
```

步骤 4　使用刚创建未有任何授权的账户进行测试

```
[root@k8s-master ~]# kubectl --context=user2@kubernetes get pods
Error from server (Forbidden): pods is forbidden: User "user2" cannot list
```

```
resource "pods" in API group "" in the namespace "default"
```

#可以看到没有权限获取该资源

1.1.3 使用内置的 ClusterRole 绑定新建账户

步骤 1 查看 kubernetes 内建 clusterrole

```
[root@k8s-master ~]# kubectl get clusterrole
```

我们选择 vimew 这个内建 clusterrole

步骤 2 查看 vimew 这个 clusterrole 的权限

```
[root@k8s-master ~]# kubectl describe clusterrole vimew
```

```
namespaces/status                    []                 []              [get list watch]
namespaces                           []                 []              [get list watch]
persistentvolumeclaims               []                 []              [get list watch]
pods/log                             []                 []              [get list watch]
pods/status                          []                 []              [get list watch]
pods                                 []                 []              [get list watch]
replicationcontrollers/scale         []                 []              [get list watch]
replicationcontrollers/status        []                 []              [get list watch]
replicationcontrollers               []                 []              [get list watch]
resourcequotas/status                []                 []              [get list watch]
resourcequotas                       []                 []              [get list watch]
serviceaccounts                      []                 []              [get list watch]
```

注：对于大部分资源都是 get、list、watch 的查询权限

步骤 3 在 master 节点创建/labfile/rbac 目录，用于保存配置文件。后续有关本章节的 yaml 文件均保存在此处

```
[root@k8s-master /]# cd /
[root@k8s-master /]# mkdir labfile
[root@k8s-master /]# cd labfile/
[root@k8s-master labfile]# mkdir rbac
[root@k8s-master labfile]# cd rbac/
```

步骤 4 创建一个 ClusterRoleBinding 对象，绑定该 ClusterRole（即 view）

```
[root@k8s-master rbac]# vim ClusterRolebindinguser2.yaml
```

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: clusterrolebindinguser2
subjects:
- kind: User
  name: user2
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: view
  apiGroup: rbac.authorization.k8s.io
```

```
[root@k8s-master rbac]# kubectl apply -f ClusterRolebindinguser2.yaml
```

```
clusterrolebinding.rbac.authorization.k8s.io/clusterrolebindinguser2 created
```

步骤 5　再次测试 user2 账号，已经可以获取到相关信息

```
[root@k8s-master ~]# kubectl --context=user2@kubernetes get pods
NAME                              READY   STATUS    RESTARTS   AGE
httpd-686c9d9595-2mq6t            1/1     Running   2          44d
httpd-686c9d9595-clkjj            1/1     Running   2          44d
httpd-686c9d9595-zf71g            1/1     Running   0          44d
myapp-mychart-7956d98c68-t9kzc    1/1     Running   0          13h
nginx                             1/1     Running   0          19h
pod-first-affinity                1/1     Running   0          24h
pod-second-affinity               0/1     Pending   0          24h
```

1.1.4 创建 Service Account 等相关对象

步骤 1　创建一个名为 mynamespace 的命名空间

```
[root@k8s-master ~]# kubectl create namespace mynamespace
namespace/mynamespace created
```

查看一下 namespace，确认已经创建成功

```
NAME              STATUS   AGE
default           Active   47d
kube-node-lease   Active   47d
kube-public       Active   47d
kube-system       Active   47d
mynamespace       Active   4d
```

步骤 2　编辑 YAML 文件创建一个 Service Account 类型的账户

```
[root@k8s-master rbac]# vim serviceaccount.yaml
```

以下为 yaml 文件中的内容：

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

步骤 3　创建该 Service Account 账户

```
[root@k8s-master rbac]# kubectl apply -f serviceaccount.yaml
serviceaccount/example-sa created
```

步骤 4　确认账号创建成功

```
[root@k8s-master ~]# kubectl get sa -n mynamespace
[root@k8s-master ~]# kubectl get sa -n mynamespace  example-sa -o yaml
```

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"ServiceAccount","metadata":{"annotations":{},"name":"example-sa","
  creationTimestamp: "2019-07-01T07:07:34Z"
  name: example-sa
  namespace: mynamespace
  resourceVersion: "5961988"
  selfLink: /api/v1/namespaces/mynamespace/serviceaccounts/example-sa
  uid: e66695be-9bce-11e9-b39d-525400cbb5d4
secrets:
- name: example-sa-token-lbrnr
```

步骤 5　编辑 Role 的 YAML 文件

```
[root@k8s-master rbac]# vim roletemplate.yaml
```

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

步骤 6　使用该 YAML 文件创建该 Role

```
[root@k8s-master rbac]# kubectl apply -f roletemplate.yaml
```

```
role.rbac.authorization.k8s.io/example-role configured
```

检查 role 对象是否创建成功

```
[root@k8s-master rbac]# kubectl get role example-role -n mynamespace
[root@k8s-master rbac]# kubectl get role example-role -n mynamespace -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"rbac.authorization.k8s.io/v1","kind":"Role","metadata":{"annotations":{},"
e":"mynamespace"},"rules":[{"apiGroups":[""],"resources":["pods"],"verbs":["get","watch","list"
  creationTimestamp: "2019-06-27T06:09:35Z"
  name: example-role
  namespace: mynamespace
  resourceVersion: "5455581"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/mynamespace/roles/example-role
  uid: 23184aa6-98a2-11e9-b39d-525400cbb5d4
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - get
  - watch
  - list
```

步骤 7　编写 Rolebinding 的 YAML 文件来为 Service Account 分配权限。

```
[root@k8s-master rbac]# vim RoleBinding.yaml
```

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

步骤 8　使用该 YAML 创建 RoleBinding。

```
[root@k8s-master rbac]# kubectl apply -f RoleBinding.yaml
rolebinding.rbac.authorization.k8s.io/example-rolebinding configured
```

rolebinding.rbac.authorization.k8s.io/example-rolebinding configured

检查 rolebinding 对象是否创建成功

```
[root@k8s-master rbac]# kubectl get rolebinding -n mynamespace
[root@k8s-master rbac]# kubectl get rolebinding -n mynamespace -o yaml
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"rbac.authorization.k8s.io/v1","kind":"RoleBinding","metadata":{"annotations":{},"n
ing","namespace":"mynamespace"},"roleRef":{"apiGroup":"rbac.authorization.k8s.io","kind":"Role","name":
cts":[{"kind":"ServiceAccount","name":"example-sa","namespace":"mynamespace"}]}
  creationTimestamp: "2019-06-27T13:07:30Z"
  name: example-rolebinding
  namespace: mynamespace
  resourceVersion: "5971562"
  selfLink: /apis/rbac.authorization.k8s.io/v1/namespaces/mynamespace/rolebindings/example-rolebinding
  uid: 85007771-98dc-11e9-b39d-525400cbb5d4
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: example-role
subjects:
- kind: ServiceAccount
  name: example-sa
  namespace: mynamespace
```

1.1.5 创建 Pod 引用该 Service Account

步骤 1　编辑 Pod YAML 文件引用该 Service Account。

```
[root@k8s-master rbac]# vim nginx.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  namespace: mynamespace
  name: nginx-test
spec:
```

```
  containers:
  - name: nginx
    image: nginx:1.7.9
  serviceAccountName: example-sa
```

步骤 2　使用该 YAML 文件创建 Pod

```
[root@k8s-master rbac]# kubectl apply -f nginx.yaml
pod/nginx-test created
```

查看 pod 状态

```
[root@k8s-master ~]# kubectl get pod -n mynamespace
```

步骤 3　查看该 Pod 相关信息

```
[root@k8s-master ~]# kubectl describe pod nginx-test -n mynamespace
Name:               nginx-test
Namespace:          mynamespace
Priority:           0
PriorityClassName:  <none>
Node:               worker02/192.168.137.15
Start Time:         Mon, 01 Jul 2019 15:58:50 +0800
Labels:             <none>
Annotations:        kubectl.kubernetes.io/last-applied-configuration:
                      {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"na
ce"},"spec":{"containers":[{"image"...
Status:             Running
IP:                 10.244.1.15
Containers:
  nginx:
    Container ID:   docker://f1d17cd69f271b41e69b8a441bacc677b4588f1b37d00c0168c48a11d
    Image:          nginx:1.7.9
    Image ID:       docker-pullable://nginx@sha256:e3456c851a152494c3e4ff5fcc26f240206
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Mon, 01 Jul 2019 15:59:33 +0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from example-sa-token-lbrnr (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  example-sa-token-lbrnr:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  example-sa-token-lbrnr
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
```

注意这里使用的 SecretsName 和之前创建的名为 example-sa 的 Service Account 中分配的 secretsname 一致。(使用 kubectl get sa -n mynamespace -o yaml 查看)

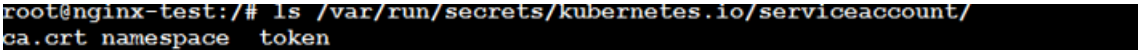
步骤 4　进入之前创建的 Pod 中查看该 secret 对象

#进入容器

```
[root@k8s-master ~]# kubectl exec -it nginx-test -n mynamespace -- /bin/bash
```

#查看容器中对应路径下的文件

```
root@nginx-test:/ls /var/run/secrets/kubernetes.io/serviceaccount/
```



该 secret 对象被挂载在该 Pod 对应的目录下，容器里的应用就可以使用 ca.crt 来访问 API Server，此时能做 GET、WATCH 和 LIST 的操作。因为 example-sa 这个 ServiceAccount 的权限已经被我们用 role 做了绑定

exit 退出