

# Kubernetes API的访问控制

当需要调用kubernetes的API时，无论普通用户还是ServiceAccount，都需要通过认证和授权。

client-->Authentication-->Authorization-->AdmissionControl-->Resource

## 传输安全

通过TLS实现数据的加密传输，TLS双向认证

## 认证

建立TLS之后，惊醒HTTP请求认证，检查客户端证书和令牌等

## 鉴权

依据用户，判断所拥有的权限

请求动词对应表：

HTTP 动词	请求动词
POST	create
GET, HEAD	get（针对单个资源）、list（针对集合）
PUT	update
PATCH	patch
DELETE	delete（针对单个资源）、deletecollection（针对集合）

## 鉴权模块

RBAC，基于角色的访问控制

要素：

- serviceAccount
- Role/ClusterRole
- RoleBonding/ClusterRoleBonding

两类用户：

- 普通用户
- serviceAccount

## 普通用户的证书获取

### 创建私钥生成CSR

```
openssl genrsa -out liuzhi.key 2048
openssl req -new -key liuzhi.key -out liuzhi.csr
```

### 创建CSR资源

创建一个 CertificateSigningRequest，并通过 kubectl 将其提交到 Kubernetes 集群。下面是生成 CertificateSigningRequest 的脚本。

```
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: liuzhi
spec:
  groups:
    - system:authenticated
  request:
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSBRSRVFVRVNULS0tLS0KTU1JQ1ZqQ0NBVDQVFBd0VURVBNTBHQTFVRUF3d
0dZVzVuWld4aE1JSUJJakFOQmdrcWhraUc5dzBCQVFFRgpBQU9DQVE4QU1JSUJDZ0tDQVFFQTBByczhJTHRHDYxak
x2dHhWTTJSVlRWMDNHw1JTww0dWluVWo4RElajB0CnR2MUZtRVFSd3VoaUZsOFEzcWl0Qm0wMUFSMkNJVBGd2Z
zSjZ4MXF3ckJzVkhZbGlBNVhwRVpZM3ExcGswSDQKM3Z3aGJlK1o2MVNrVHF5SVBYUWUwTWMT1Nsbm0xb0R2N0Nt
SkZNMU1MRVI3QTVGZnZKOEEdFRjJ6dHBoaU1FMwpub1dtdHNZb3JuT2wzc2lHQ2ZGZzR4Zmd4eW8ybm1neFNVEk11b
XNnVm9PM2ttT0x1RVF6cXpkakJ3TFJXbW1Eck1mMXBMWnoyaVnaId4UkhCM1gyWnVVV1d1T09PZnpXM01LaE8ybH
EvZi9DdS8wYk83c0x0Mct3U2ZMSU91TFcKcW90blZtRmxMMytqTy82WDNDKzBERHk5aUtwbXJjVDBnWGZLemE1dHJ
RSURBUUFcb0FBd0RRWUpLb1pJaHZjTGpBUUVMQlFBRGdnRUJBR05WdmVIOGR4ZzNvK21VeVRkbmFjVmQ1N24zSkEx
dnZEU1JWREkyQTZ1eXN3ZFp1L1BVcKkwZXpZWV0RVNnSk1IRmQycVVMjNuNVJsSXJ3R0xuUXFISU5VStWWhsd
nZsRnpNOVpEW1lSTmU3Q1JvYXgKQV1EdUI5STZXT3FYbkFvczFqRmxNUG5NbFpqdU5kSGxpT1BjTU1oNndLaTZzZF
hpVStHTYT2RUUVl0Y1jSVUyRgpvU2djUWdMYTtk0aEpacGk3ZnNmM10QUxoT045UHdNMGM1dVJVeVjV4T0dGMUtdCbWR
SeEgVbUNOS2JKYjFRQm1HCkkwYitEUEdaTktXTU0xMzhIXQdoV0tkNjVoVHdYOW14V3ZHMkh4TG1WQzg0L1BHT0tW
QW9FNkpsYWFDHd1QVmkKdj10SjVaZlZrcXdCd0hkZbZXdK9xVlA3SVFjZmg3d0drWm89Ci0tLS0tRU5EIENFU1RJR
klDQVRFIjFjFUVVU1Q1tLS0tLQo=
  signerName: kubernetes.io/kube-apiserver-client
  usages:
    - client auth
```

#### signerName:

- `kubernetes.io/kube-apiserver-client`：签名的证书将被 API 服务器视为客户证书。 kube-controller-manager 不会自动批准它。
- `kubernetes.io/kube-apiserver-client-kubelet`：签名的证书将被 kube-apiserver 视为客户证书。 kube-controller-manager 可以自动批准它。
- `kubernetes.io/kubelet-serving`：签名服务证书，该服务证书被 API 服务器视为有效的 kubelet 服务证书， 但没有其他保证。 kube-controller-manager 不会自动批准它。

## 批准证书请求

```
kubectl certificate approve liuzhi
```

## 获取证书

```
kubectl get csr/liuzhi -o yaml  
kubectl get csr liuzhi -o jsonpath='{.status.certificate}' | base64 -d > liuzhi.crt
```

## 创建角色和角色绑定

```
kubectl apply -f role.yaml
```

```
kubectl create rolebinding developer-binding-myuser --role=reader --user=liuzhi
```

## 添加到 kubeconfig

```
kubectl config set-credentials liuzhi --client-key=liuzhi.key --client-certificate=liuzhi.crt --embed-certs=true
```

## 添加上下文

```
kubectl config set-context liuzhi --cluster=kubecamp --user=liuzhi
```

## 切换上下文

```
kubectl config use-context liuzhi
```

## ServiceAccount的证书获取

查看系统已有用户相关信息，并分析作用

1. 创建一个 `serviceaccount`，在 `default namespace`

```
[root@kube01 ~]# kubectl create serviceaccount liuzhi-sa  
[root@kube01 ~]# kubectl get sa  
NAME          SECRETS  AGE  
default       1        14d  
liuzhi-sa     1        26m
```

## 2. 创建一个deployment使用 serviceAccount: liuzhi-sa

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: centos
  labels:
    app: centos
spec:
  replicas: 1
  selector:
    matchLabels:
      app: centos
  template:
    metadata:
      labels:
        app: centos
    spec:
      containers:
        - name: centos
          image: centos
          command: ['sleep', '3600']
          serviceAccount: liuzhi-sa
```

## 3. 登录并安装curl测试工具

```
[root@k8s01 ~]# kubectl exec -it centos-58b9d9bdbc-fdqgz -- bash

[root@centos-58b9d9bdbc-fdqgz /]# ls -l /var/run/secrets/kubernetes.io/serviceaccount/
total 0
lrwxrwxrwx 1 root root 13 Jul 22 02:12 ca.crt -> ..data/ca.crt
lrwxrwxrwx 1 root root 16 Jul 22 02:12 namespace -> ..data/namespace
lrwxrwxrwx 1 root root 12 Jul 22 02:12 token -> ..data/token

# 自动挂载serviceAccount访问集群需要三个核心:
# ca.crt
# namespace
# token

[root@centos-58b9d9bdbc-fdqgz /]# yum install -y curl
```

## 4. 访问Kubernetes APIServer

```
[root@centos-58b9d9bdbc-fdqgz /]# curl
https://kubernetes.default.svc/api/v1/namespaces/default/pods --cacert
/var/run/secrets/kubernetes.io/serviceaccount/ca.crt --header "Authorization: Bearer
$TOKEN"
{
  "kind": "Status",
```

```

    "apiVersion": "v1",
    "metadata": {

    },
    "status": "Failure",
    "message": "forbidden: User \"system:anonymous\" cannot get path \"/v1\"",
    "reason": "Forbidden",
    "details": {

    },
    "code": 403
  }
}

```

缺乏访问权限

## 5. 我们为liuzhi-sa绑定role

创建Role

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: reader
  namespace: default
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - list
  - watch
  - get

```

```

[root@kube01 ~]# kubectl create rolebinding read-pods --role=reader --
serviceaccount=default:liuzhi-sa
rolebinding.rbac.authorization.k8s.io/read-pods created

```

重新尝试访问API

```

[root@centos-c6647f6b9-j95wd /]# curl \
https://kubernetes.default.svc/api/v1/namespaces/default/pods \
--cacert /var/run/secrets/kubernetes.io/serviceaccount/ca.crt \
--header "Authorization: Bearer $TOKEN"
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "263482"
  },
  "items": [

```

```
{  
    ....
```

这就是serviceAccount注入到POD中的意义,ClusterRole与Role类似, 但没有namespace的限制

## 准入控制

---

准入控制器是一段代码, 它会在请求通过认证和授权之后、对象被持久化之前拦截到达 API 服务器的请求。

开启准入控制

```
kube-apiserver --enable-admission-plugins=NamespaceLifecycle,LimitRanger ...
```

