

# K8S web 权限管理与身份验证

西安电子科技大学 XenoWYC121

## 1. K8S 身份验证方式

### 1.1 客户端证书

K8S 可以签发证书，提供给用户使用。用户拥有 Kubernetes 集群签发的证书，然后将该证书提供给 Kubernetes Api server，就可以调用 Kubernetes 的接口。

流程：

1. 客户端生成证书请求：客户端生成一个公私钥对，并创建一个证书签名请求（CSR）。
2. CA 签发证书：CSR 发送给 CA，CA 对请求进行验证并签署生成客户端证书。
3. 客户端持有证书：客户端持有由 CA 签署的证书和私钥。
4. 服务器验证证书：客户端在请求时提供证书，服务器验证证书的有效性和签名。
5. 建立安全连接：验证成功后，建立安全的 TLS 连接。

优点：

- 高安全性，难以伪造；适用于自动化和机器间通信。

缺点：

- 证书管理复杂，证书更新需要手动分发。

### 1.2 静态用户名与密码

使用预定义的用户名和密码进行身份验证。

流程：

1. 配置用户名和密码：在 Kubernetes Api server 中配置静态用户名和密码。
2. 客户端发送请求：客户端在请求中包含用户名和密码。
3. 服务器验证：服务器验证用户名和密码的匹配。
4. 访问授权：验证成功后，服务器根据用户权限提供相应的资源访问。

优点：

- 配置简单，适用于小规模或测试环境。

缺点：

- 安全性低，不适宜大规模环境

## 1.3 OpenID Connect (OIDC)

使用 OpenID Connect 协议进行身份验证，基于 OAuth 2.0 授权框架，使用 JWT（JSON Web Token）进行身份验证和信息传递。

**流程：**

1. 客户端重定向到身份提供者：客户端向 Kubernetes apiserver 发送请求，重定向到身份提供者（IdP）。
2. 身份提供者认证：用户在 IdP 处进行身份验证（如输入用户名和密码）。
3. 获取 ID Token：认证成功后，IdP 生成一个 ID Token（JWT）并返回给客户端。
4. 客户端携带 ID Token：客户端在后续请求中携带 ID Token。
5. 服务器验证 ID Token：Kubernetes apiserver 验证 ID Token 的签名和有效性。
6. 授权访问：验证成功后，根据 Token 中的信息和 RBAC 配置授权访问资源。

**优点：**

- 高安全性，支持单点登录；

**缺点：**

- 配置复杂，需要第三方身份提供者和对 OIDC 协议的了解。

## 1.4 Webhook Token 认证

通过 HTTP 回调请求到外部服务进行身份验证。

**流程：**

- 配置 Webhook 服务：在 Kubernetes apiserver 中配置 Webhook 认证的 URL。
- 客户端发送请求：客户端在请求中携带 Token。
- 服务器发送 Webhook 请求：Kubernetes apiserver 接收到请求后，将 Token 转发给配置的 Webhook 服务。
- Webhook 服务验证：Webhook 服务验证 Token 的有效性，并返回认证结果。
- 授权访问：根据 Webhook 服务返回的结果，apiserver 决定是否授权访问资源。

**优点：**

- 灵活性高，可以自定义身份验证逻辑；

**缺点：**

- 配置和维护复杂，需要第三方 Webhook 服务；性能可能受到 Webhook 服务的影响。

## 1.5 Service Account Token

Service Account 是 Kubernetes 中的一种内置身份，常用于在 Pod 中运行的应用程序进行身份验证和授权。

**流程：**

1. 创建 Service Account: 在 Kubernetes 中创建 Service Account。
2. 自动生成 Token: Kubernetes 自动为每个 Service Account 生成一个 Token, 并将其挂载到 Pod 中。
3. 客户端携带 Token: 在 Pod 中运行的应用程序在请求中携带 Service Account Token。
4. 服务器验证 Token: Kubernetes apiserver 验证 Token 的有效性和签名。
5. 授权访问: 验证成功后, 根据 Service Account 和 RBAC 配置授权访问资源。

优点:

- 内置于 Kubernetes, 易于使用和管理;

缺点:

- Token 一旦泄露, 需要手动撤销和重新分配; 不常用于外部使用, 适用于集群内部组件。

## 2. 身份验证方式选择

选择客户端证书方式, 原因如下:

1. 用户密码与 Service account 方式安全性欠佳, 仅适用于小规模测试以及 k8s 内部组件使用。
2. OIDC 与 WebHook 认证需要第三方或额外的身份验证服务。
3. 客户端证书方式安全性高, 并且可以使用 K8S 可以直接签发证书, 配置不是很麻烦。

## 3. 签发客户端证书

### 3.1 创建私钥

```
openssl genrsa -out myuser.key 2048
```

```
openssl req -new -key myuser.key -out myuser.csr -subj "/CN=myuser"
```

上述命令使用 openssl 生成了私钥, 并根据私钥生成了 csr (证书签名请求)。其中 CN 参数表示创建的用户名。

### 3.2 创建 CertificateSigningRequest

创建一个 CertificateSigningRequest, 并通过 kubectl 将其提交到 Kubernetes 集群。

以下是 CertificateSigningRequest 定义样例:

```
apiVersion: certificates.k8s.io/v1
```

```
kind: CertificateSigningRequest
```

metadata:

name: myuser

spec:

request: 《CSR base64》

signerName: kubernetes.io/kube-apiserver-client

expirationSeconds: 86400 # one day

usages:

- client auth

需要注意的几点:

- usage 字段必须是 'client auth'
- expirationSeconds 可以设置为更长（例如 864000 是 1 天）或者更短（例如 3600 是一个小时）
- request 字段是 CSR 文件内容的 base64 编码值，要得到该值，可以执行命令：
  - `cat myuser.csr | base64 | tr -d "\n"`

最后使用以下命令将 CertificateSigningRequest 提交给 k8s:

kubectl apply -f (文件名).yaml

### 3.3 批准 CertificateSigningRequest

获取 CSR 列表:

kubectl get csr

批准 CSR:

kubectl certificate approve myuser

### 3.4 获取证书

使用以下命令可以查看 CSR 详情，其中包含已经完成签名的客户端证书:

kubectl get csr myuser -o yaml

证书的内容使用 base64 编码，存放在字段 `status.certificate`。可以使用以下命令从 CSR 中导出颁发的证书：

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

至此，身份验证部分已完成。

## 4. 创建角色与角色绑定

创建了证书之后，为了让这个用户能访问 Kubernetes 集群资源，现在就要创建 Role 和 RoleBinding。下面是为这个新用户创建 Role 的命令：

```
kubectl create role pod-admin --verb=create --verb=get --verb=list --verb=update --verb=delete --resource=pods
```

上述命令创建了一个 pod 管理员角色，拥有 k8s 集群 pod 资源的全部权限。

下面是为这个新用户创建 RoleBinding 的命令：

```
kubectl create rolebinding developer-binding-myuser --role=pod-admin --user=myuser
```

上述命令将 pod-admin 角色绑定给了新用户，使用户拥有操作 pod 的权限。

## 5. 证书使用

### 5.1 CURL 使用示例

```
curl https://<hostname:port>/api/v1/namespaces/{namespace}/{resource} -k \
    --cert myuser.crt --key myuser.key
```

在请求中使用 `--cert` 与 `--key` 参数分别指定签发证书与私钥，即可访问 k8s Restful 接口。

### 5.2 Python 使用示例

```
requests.get("https://<hostname:port>/api/v1/namespaces/{namespace}/{resource}",
             cert=('myuser.crt', 'myuser.key'), verify=False)
```