

K8S 用户认证与鉴权

西安电子科技大学 XenoWYC121

1. 操作系统管理员与 K8S 管理员

在 K8S 安装完成后，`kubectl` 命令无法直接使用，需要先执行以下命令，使得 `kubectl` 能够找到并读取配置文件：

```
mkdir -p $HOME/.kube
```

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

或者执行：

```
export KUBECONFIG=/etc/kubernetes/admin.conf
```

但是读取 `admin.conf` 配置文件需要 `root` 权限，而 `kubectl` 需要读取 `admin.conf`，因此执行 `kubectl` 命令也需要 `root` 权限。所以 K8S 的管理员本质上就是集群 `master` 节点的系统管理员，或是由系统管理员授权的用户。

2. K8S 身份认证

所有 Kubernetes 集群都有两类用户：由 Kubernetes 管理的服务账号和普通用户。Kubernetes 假定普通用户是由一个与集群无关的服务通过以下方式之一进行管理的：

- 负责分发私钥的管理员（证书）
- 类似 Keystone 或者 Google Accounts 这类用户数据库（OIDC、webhook）
- 包含用户名和密码列表的文件（静态令牌）

有鉴于此，Kubernetes 并不包含用来代表普通用户账号的对象。普通用户的信息无法通过 API 调用添加到集群中。尽管无法通过 API 调用来添加普通用户，Kubernetes 仍然认为能够提供由集群的证书机构签名的合法证书的用户是通过身份认证的用户。

与此不同，服务账号是 Kubernetes API 所管理的用户。它们被绑定到特定的名字空间，或者由 API 服务器自动创建，或者通过 API 调用创建。服务账号与一组以 `Secret` 保存的凭据相关，这些凭据会被挂载到 Pod 中，从而允许集群内的进程访问 Kubernetes API。

API 请求则或者与某普通用户相关联，或者与某服务账号相关联，亦或者被视作匿名请求。这意味着集群内外的每个进程在向 API 服务器发起请求时都必须通过身份认证，否则会被视作匿名用户。这里的进程可以是在某工作站上输入 `kubectl` 命令的操作人员，也可以是节点上的 `kubelet` 组件，还可以是调用 API 的外部进程。

Kubernetes API Server 可以同时启用多种身份认证方法，并且通常会至少使用两种方法：

- 针对服务账号使用服务账号令牌认证
- 对用户的身份进行认证的方法

当集群中启用了多个身份认证模块时，第一个成功地对请求完成身份认证的模块会直接做出评估决定。API 服务器并不保证身份认证模块的运行顺序。

2.1 静态令牌

当 API 服务器的命令行设置了 `--token-auth-file=SOMEFILE` 选项时，会从文件中读取持有者令牌。目前，令牌会长期有效，并且在不重启 API 服务器的情况下无法更改令牌列表。

令牌文件是一个 CSV 文件，包含至少 3 个列：令牌、用户名和用户的 UID。

在请求中放入持有者令牌

当使用令牌来对 HTTP 客户端执行身份认证时，API 服务器需要一个名为 `Authorization` 的 HTTP 头，其值格式为 `Bearer <token>`，令牌必须是一个可以放入 HTTP 头部值字段的字符序列，其出现在 HTTP 头部时如下所示：

- `Authorization: Bearer 《token》`

优点：

- 简单、部署迅速、无额外依赖

缺点：

- 静态令牌一旦生成，通常不会改变
- 更新令牌文件需要重启 `api server`
- 静态令牌本身由管理员自行填写，安全性取决于令牌本身强度

2.2 服务账号令牌

服务账号（Service Account）是一种自动被启用的用户认证机制，使用经过签名的令牌来验证请求。服务账号通常由 API 服务器自动创建并通过 `ServiceAccount` 准入控制器关联到集群中运行的 Pod 上，允许集群内进程与 API 服务器通信。

在集群外部使用服务账号令牌也是完全合法的，且可用来为长时间运行的、需要与 Kubernetes API 服务器通信的任务创建标识，此用法需要手动创建服务账号。

优点：

- 内置于 Kubernetes，易于使用和管理；

缺点：

- 不常用于外部使用，适用于集群内部组件间通信。

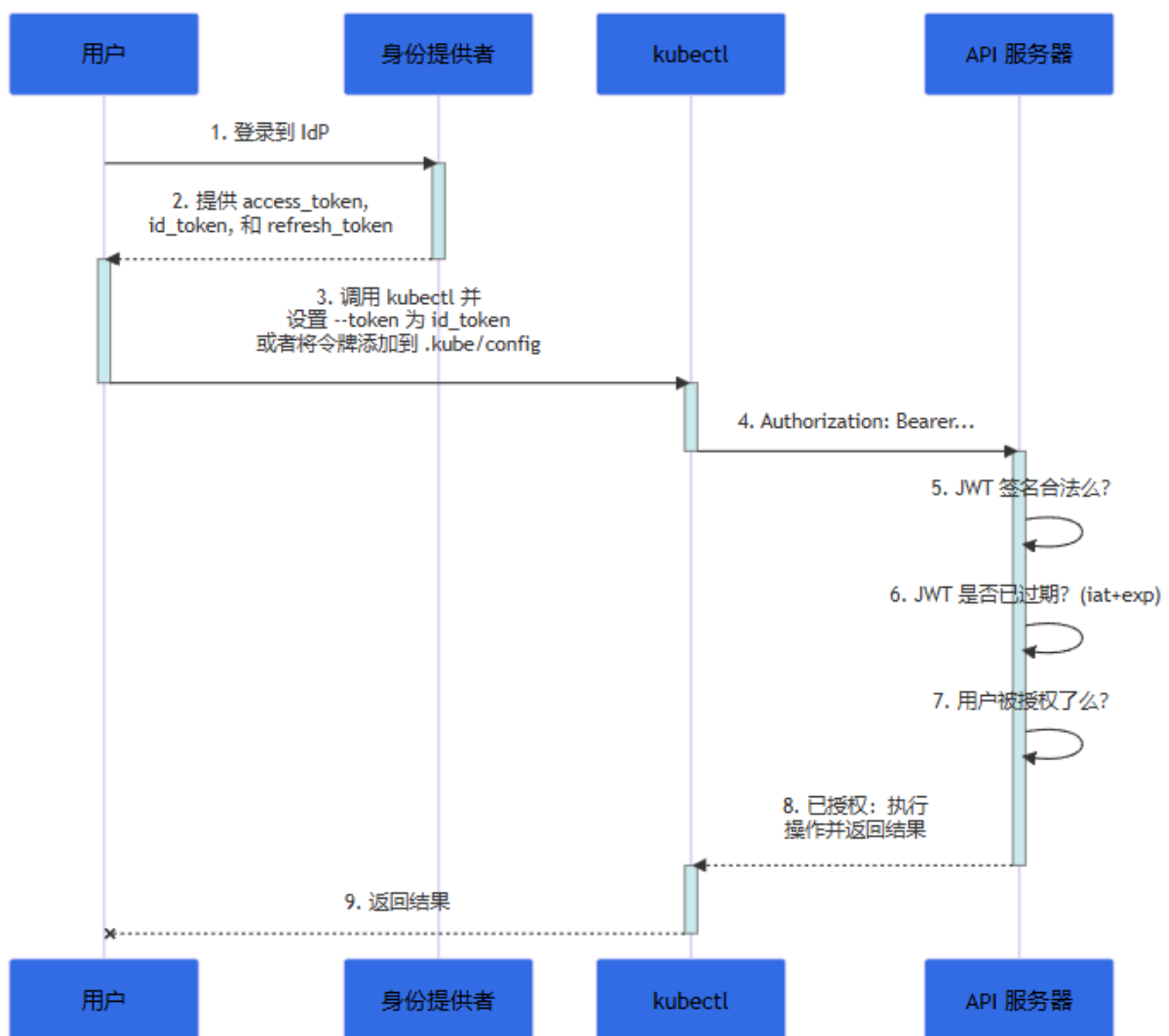
2.3 OpenID Connect (OIDC)

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. 登录到你的身份服务（Identity Provider）

2. 你的身份服务将为你提供 `access_token`、`id_token` 和 `refresh_token`
3. 在使用 `kubectl` 时，将 `id_token` 设置为 `--token` 标志值，或者将其直接添加到 `kubeconfig` 中
4. `kubectl` 将你的 `id_token` 放到一个称作 `Authorization` 的头部，发送给 API 服务器
5. API 服务器将确保 JWT 的签名是有效的
6. 检查确认 `id_token` 尚未过期
7. 如果使用 `AuthenticationConfiguration` 配置了 CEL 表达式，则执行声明和/或用户验证。
8. 确认用户有权限执行操作
9. 鉴权成功之后，API 服务器向 `kubectl` 返回响应
10. `kubectl` 向用户提供反馈信息



优点:

- 高安全性，易于与现有身份验证系统集成。

缺点:

- 配置复杂，需要第三方身份提供者和对 OIDC 协议的了解。

2.4 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。

2.5 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。

3. K8S 鉴权

Kubernetes 鉴权在身份认证之后进行。通常，发出请求的客户端必须经过身份认证与鉴权后才能允许其请求。

3.1 鉴权裁定

Kubernetes 对 API 请求的鉴权在 API 服务器内进行。API 服务器根据所有策略评估所有请求属性，可能还会咨询外部服务，然后允许或拒绝该请求。

API 请求的所有部分都必须通过某种鉴权机制才能继续，换句话说：默认情况下拒绝访问。

当系统配置了多个鉴权模块时，Kubernetes 将按顺序使用每个模块。如果任何鉴权模块**批准或拒绝**请求，则立即返回该决定，并且不会与其他鉴权模块协商。如果所有模块对请求**没有意见**（该模块不负责处理此类请求），则拒绝该请求。总体拒绝裁决意味着 API 服务器拒绝请求并以 HTTP 403（禁止）状态进行响应。

3.2 资源请求

为了确定资源 API 端点的请求动词，Kubernetes 会映射所使用的 HTTP 动词，并考虑该请求是否作用于单个资源或资源集合：

HTTP 动词	请求动词
POST	create
GET 、 HEAD	get （针对单个资源）、 list （针对集合，包括完整的对象内容）、 watch （用于查看单个资源或资源集合）
PUT	update
PATCH	patch
DELETE	delete （针对单个资源）、 deletecollection （针对集合）

请求动词与 Http 动词的映射关系，K8S 内部权限使用请求动词，而 Restful 只支持 Http 方法，需要进行转换。

3.3 鉴权模式

AlwaysAllow

此模式允许所有请求，但存在安全风险，仅当你的 API 请求不需要鉴权时（例如，用于测试），才使用此鉴权模式。

AlwaysDeny

此模式阻止所有请求。此鉴权模式仅适用于测试。

ABAC（基于属性的访问控制）

Kubernetes ABAC 模式定义了一种访问控制范例，通过使用将属性组合在一起的策略向用户授予访问权限，策略可以使用任何类型的属性（用户属性、资源属性、对象、环境属性等）。

RBAC（基于角色的访问控制）

Kubernetes RBAC 是一种根据企业内各个用户的角色来管理其对计算机或网络资源的访问权限的方法。在此上下文中，访问权限是单个用户执行特定任务（例如查看、创建或修改文件）的能力。在这种模式

下，Kubernetes 使用 `rbac.authorization.k8s.io` API 组来驱动鉴权决策，允许你通过 Kubernetes API 动态配置权限策略。

Node

一种特殊用途的鉴权模式，根据 `kubelet` 计划运行的 Pod 向其授予权限。

Webhook

Kubernetes 的 Webhook 鉴权模式用于鉴权，进行同步 HTTP 调用，阻塞请求直到远程 HTTP 服务响应查询。可以自定义软件来处理这种向外调用，也可以使用已有的解决方案。

3.3.1 ABAC

基于属性的访问控制（Attribute-based access control, ABAC）定义了访问控制范例，ABAC 通过使用将属性组合在一起的策略来向用户授予访问权限。

简单来说就是：某用户对某 namespace 下的某资源有某种操作的权限

例如：

Alice 可以对所有资源做任何事情：

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "alice", "namespace": "*", "resource": "*", "apiGroup": "*"}}
```

kubelet 可以读取所有 Pod：

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet", "namespace": "*", "resource": "pods", "readOnly": true}}
```

kubelet 可以读写事件：

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "kubelet", "namespace": "*", "resource": "events"}}
```

Bob 可以在命名空间 `projectCaribou` 中读取 Pod：

```
{"apiVersion": "abac.authorization.kubernetes.io/v1beta1", "kind": "Policy", "spec": {"user": "bob", "namespace": "projectCaribou", "resource": "pods", "readOnly": true}}
```

3.3.2 RBAC

比 ABAC 优点，能方便给一群用户授权（明确查询一下）

基于角色的访问控制是一种访问控制机制，通过将权限分配给用户的角色，而不是直接分配给用户来管理访问权限。每个角色代表一组特定的权限，用户被分配到这些角色中，从而继承相应的权限。

K8S RBAC API 声明了四种 Kubernetes 对象：Role、ClusterRole、RoleBinding 和 ClusterRoleBinding。**Role** 或 **ClusterRole** 中包含一组代表相关权限的规则。**Role** 用来在某个 namespace 内设置访问权限；在创建 **Role** 时，必须指定该 **Role** 所属的名字空间。**ClusterRole** 则是一个集群作用域的资源，可以设置所有 namespace 的访问权限。

角色绑定（**Role Binding**）是将角色中定义的权限赋予一个或者一组用户。它包含若干主体（**Subject**）（用户、组或服务账户）的列表和对这些主体所获得的角色的引用。**RoleBinding** 在指定的 namespace 中执行授权，而 **ClusterRoleBinding** 在集群范围执行授权。

一个 **RoleBinding** 可以引用同一个 namespace 中的任何 **Role**。并且一个 **RoleBinding** 可以引用某 **ClusterRole** 并将该 **ClusterRole** 绑定到 **RoleBinding** 所在的名字空间。如果需要将某 **ClusterRole** 绑定到集群中所有 namespace，请使用 **ClusterRoleBinding**。

3.3.3 Node 鉴权

节点鉴权是一种特殊用途的鉴权模式，专门对 kubelet 发出的 API 请求进行鉴权。

节点鉴权器允许 kubelet 执行 API 操作。包括：

读取操作：

- services
- endpoints
- nodes
- pods
- 与绑定到 kubelet 节点的 Pod 相关的 Secret、ConfigMap、PersistentVolumeClaim 和持久卷

写入操作：

- 节点和节点状态（启用 NodeRestriction 准入插件以限制 kubelet 只能修改自己的节点）
- Pod 和 Pod 状态（启用 NodeRestriction 准入插件以限制 kubelet 只能修改绑定到自身的 Pod）
- 事件