



下载APP



## 05 | 鉴权：如何保护你的数据安全？

2021-01-29 唐聪

etcd实战课

[进入课程 >](#)**讲述：王超凡**

时长 18:45 大小 17.18M



你好，我是唐聪。

不知道你有没有过这样的困惑，当你使用 etcd 存储业务敏感数据、多租户共享使用同 etcd 集群的时候，应该如何防止匿名用户访问你的 etcd 数据呢？多租户场景又如何最小化用户权限分配，防止越权访问的？

etcd 鉴权模块就是为了解决以上痛点而生。

那么 etcd 是如何实现多种鉴权机制和细粒度的权限控制的？在实现鉴权模块的过程中，核心的挑战是什么？又该如何确保鉴权的安全性以及提升鉴权性能呢？

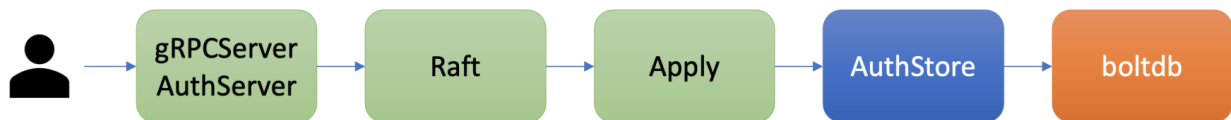


今天这节课，我将为你介绍 etcd 的鉴权模块，深入剖析 etcd 如何解决上面的这些痛点和挑战。希望通过这节课，帮助你掌握 etcd 鉴权模块的设计、实现精要，了解各种鉴权方案的优缺点。你能在实际应用中，根据自己的业务场景、安全诉求，选择合适的方案保护你的 etcd 数据安全。同时，你也可以参考其设计、实现思想应用到自己业务的鉴权系统上。

## 整体架构

在详细介绍 etcd 的认证、鉴权实现细节之前，我先给你从整体上介绍下 etcd 鉴权体系。

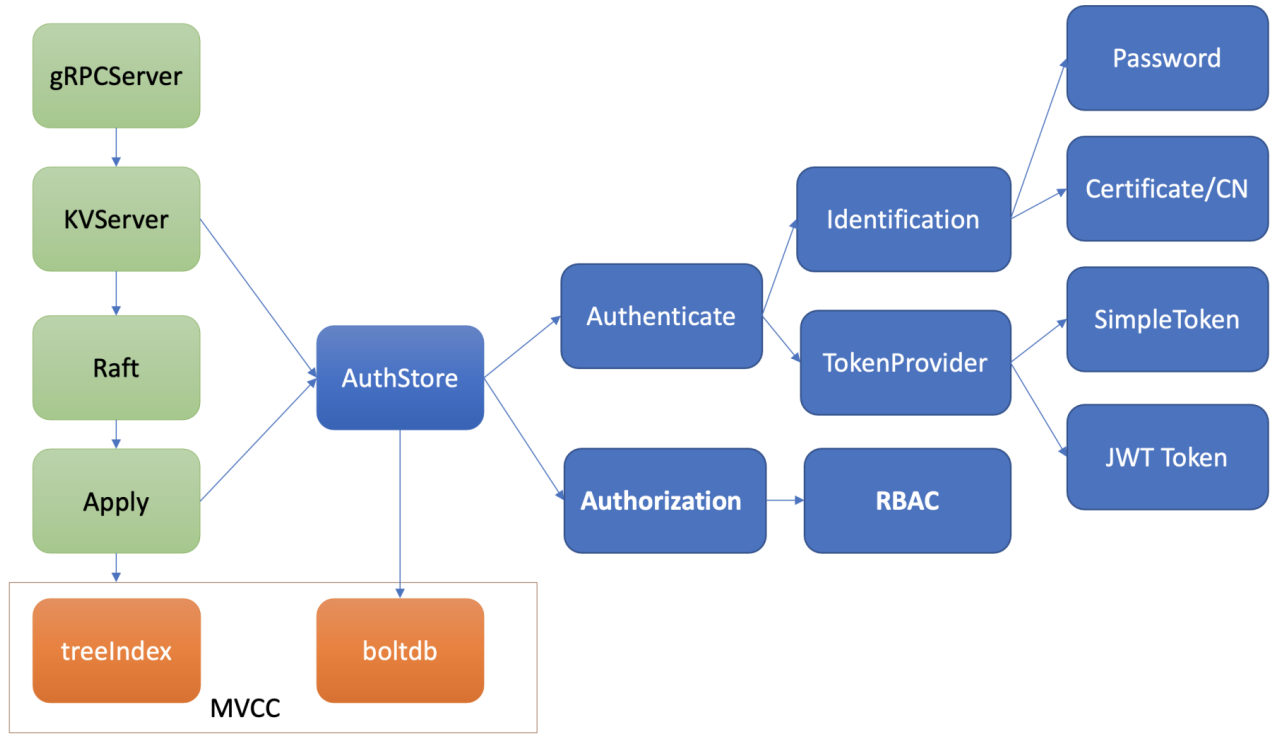
etcd 鉴权体系架构由控制面和数据面组成。



上图是 etcd 鉴权体系控制面，你可以通过客户端工具 etcdctl 和鉴权 API 动态调整认证、鉴权规则，AuthServer 收到请求后，为了确保各节点间鉴权元数据一致性，会通过 Raft 模块进行数据同步。

当对应的 Raft 日志条目被集群半数以上节点确认后，Apply 模块通过鉴权存储 (AuthStore) 模块，执行日志条目的内容，将规则存储到 boltdb 的一系列“鉴权表”里面。

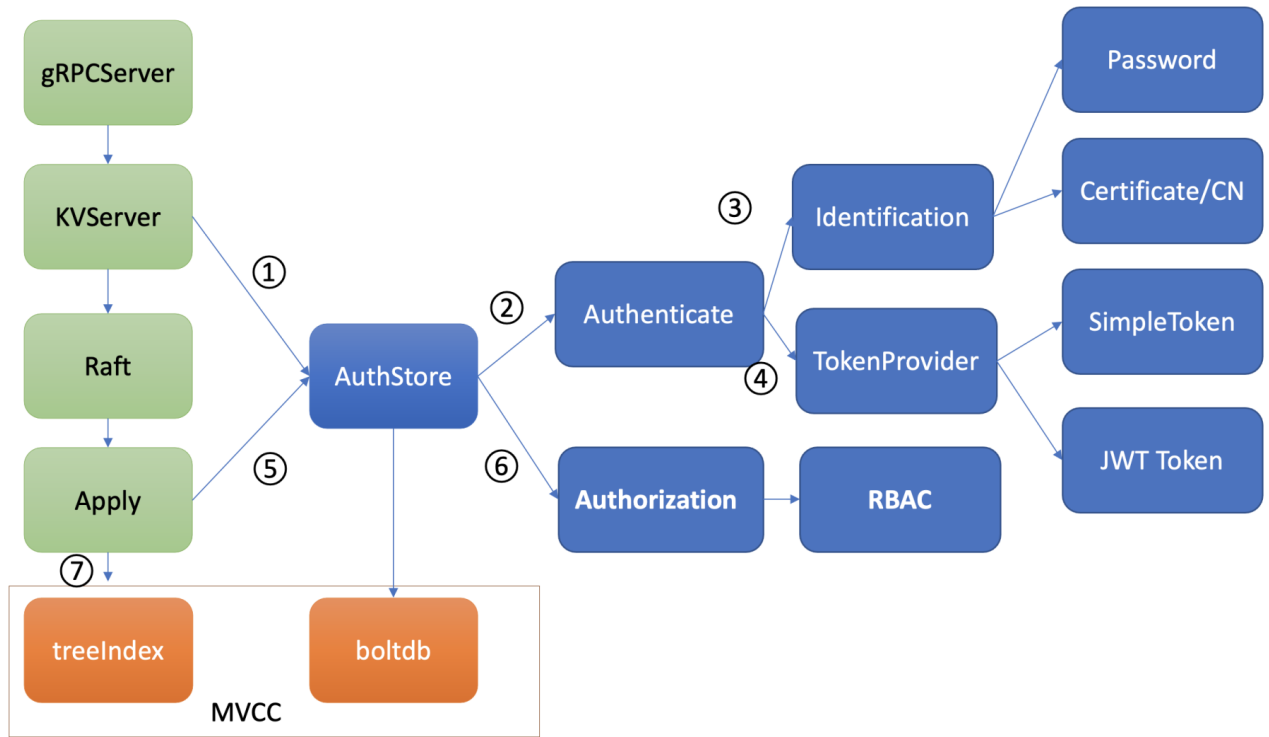
下图是数据面鉴权流程，由认证和授权流程组成。认证的目的是检查 client 的身份是否合法、防止匿名用户访问等。目前 etcd 实现了两种认证机制，分别是密码认证和证书认证。



认证通过后，为了提高密码认证性能，会分配一个 Token（类似我们生活中的门票、通信证）给 client，client 后续其他请求携带此 Token，server 就可快速完成 client 的身份校验工作。

实现分配 Token 的服务也有多种，这是 TokenProvider 所负责的，目前支持 SimpleToken 和 JWT 两种。

通过认证后，在访问 MVCC 模块之前，还需要通过授权流程。授权的目的是检查 client 是否有权限操作你请求的数据路径，etcd 实现了 RBAC 机制，支持为每个用户分配一个角色，为每个角色授予最小化的权限。



好了，etcd 鉴权体系的整个流程讲完了，下面我们就以 [第三节课](#) 中提到的 put hello 命令为例，给你深入分析以上鉴权体系是如何进行身份认证来防止匿名访问的，又是如何实现细粒度的权限控制以防止越权访问的。

## 认证

首先我们来看第一个问题，如何防止匿名用户访问你的 etcd 数据呢？

解决方案当然是认证用户身份。那 etcd 提供了哪些机制来验证 client 身份呢？

正如我整体架构中给你介绍的，etcd 目前实现了两种机制，分别是用户密码认证和证书认证，下面我分别给你介绍这两种机制在 etcd 中如何实现，以及这两种机制各自的优缺点。

## 密码认证

首先我们来讲讲用户密码认证。etcd 支持为每个用户分配一个账号名称、密码。密码认证在我们生活中无处不在，从银行卡取款到微信、微博 app 登录，再到核武器发射，密码认证应用及其广泛，是最基础的鉴权的方式。

但密码认证存在两大难点，它们分别是如何保障密码安全性和提升密码认证性能。

## 如何保障密码安全性

我们首先来看第一个难点：如何保障密码安全性。

也许刚刚毕业的你会说直接明文存储，收到用户鉴权请求的时候，检查用户请求中密码与存储中是否一样，不就可以了吗？这种方案的确够简单，但你是否想过，若存储密码的文件万一被黑客脱库了，那么所有用户的密码都将被泄露，进而可能会导致重大数据泄露事故。

也许你又会说，自己可以奇思妙想构建一个加密算法，然后将密码翻译下，比如将密码中的每个字符按照字母表序替换成字母后的第 XX 个字母。然而这种加密算法，它是可逆的，一旦被黑客识别到规律，还原出你的密码后，脱库后也将导致全部账号数据泄密。

那么是否我们用一种不可逆的加密算法就行了呢？比如常见的 MD5，SHA-1，这方案听起来似乎有点道理，然而还是不严谨，因为它们的计算速度非常快，黑客可以通过暴力枚举、字典、彩虹表等手段，快速将你的密码全部破解。

LinkedIn 在 2012 年的时候 650 万用户密码被泄露，黑客 3 天就暴力破解出 90% 用户的密码，原因就是 LinkedIn 仅仅使用了 SHA-1 加密算法。

### 那应该如何进一步增强不可逆 hash 算法的破解难度？

一方面我们可以使用安全性更高的 hash 算法，比如 SHA-256，它输出位数更多、计算更加复杂且耗 CPU。

另一方面我们可以在每个用户密码 hash 值的计算过程中，引入一个随机、较长的加盐 (salt) 参数，它可以让相同的密码输出不同的结果，这让彩虹表破解直接失效。

彩虹表是黑客破解密码的一种方法之一，它预加载了常用密码使用 MD5/SHA-1 计算的 hash 值，可通过 hash 值匹配快速破解你的密码。


最后我们还可以增加密码 hash 值计算过程中的开销，比如循环迭代更多次，增加破解的时间成本。

### etcd 的鉴权模块如何安全存储用户密码？

etcd 的用户密码存储正是融合了以上讨论的高安全性 hash 函数（Blowfish encryption algorithm）、随机的加盐 salt、可自定义的 hash 值计算迭代次数 cost。


下面我将通过几个简单 etcd 鉴权 API，为你介绍密码认证的原理。

首先你可以通过如下的 auth enable 命令开启鉴权，注意 etcd 会先要求你创建一个 root 账号，它拥有集群的最高读写权限。

 复制代码

```
1 $ etcdctl user add root:root
2 User root created
3 $ etcdctl auth enable
4 Authentication Enabled
```


启用鉴权后，这时 client 发起如下 put hello 操作时，etcd server 会返回"user name is empty"错误给 client，就初步达到了防止匿名用户访问你的 etcd 数据目的。那么 etcd server 是在哪里做的鉴权的呢？

 复制代码

```
1 $ etcdctl put hello world
2 Error: etcdserver: user name is empty
```

etcd server 收到 put hello 请求的时候，在提交到 Raft 模块前，它会从你请求的上下文中获取你的用户身份信息。如果你未通过认证，那么在状态机应用 put 命令的时候，检查身份权限的时候发现是空，就会返回此错误给 client。

下面我通过鉴权模块的 user 命令，给 etcd 增加一个 alice 账号。我们一起来看看 etcd 鉴权模块是如何基于我上面介绍的技术方案，来安全存储 alice 账号信息。

 复制代码

```
1 $ etcdctl user add alice:alice --user root:root
2 User alice created
```

鉴权模块收到此命令后，它会使用 bcrpt 库的 blowfish 算法，基于明文密码、随机分配的 salt、自定义的 cost、迭代多次计算得到一个 hash 值，并将加密算法版本、salt 值、cost、hash 值组成一个字符串，作为加密后的密码。

最后，鉴权模块将用户名 alice 作为 key，用户名、加密后的密码作为 value，存储到 boltdb 的 authUsers bucket 里面，完成一个账号创建。

当你使用 alice 账号访问 etcd 的时候，你需要先调用鉴权模块的 Authenticate 接口，它会验证你的身份合法性。

那么 etcd 如何验证你密码正确性的呢？

鉴权模块首先会根据你请求的用户名 alice，从 boltdb 获取加密后的密码，因此 hash 值包含了算法版本、salt、cost 等信息，因此可以根据你请求中的明文密码，计算出最终的 hash 值，若计算结果与存储一致，那么身份校验通过。

## 如何提升密码认证性能

通过以上的鉴权安全性的深入分析，我们知道身份验证这个过程开销极其昂贵，那么问题来了，如何避免频繁、昂贵的密码计算匹配，提升密码认证的性能呢？

这就是密码认证的第二个难点，如何保证性能。

想想我们办理港澳通行证的时候，流程特别复杂，需要各种身份证明、照片、指纹信息，办理成功后，下发通信证，每次过关你只需要刷下通信证即可，高效而便捷。

那么，在软件系统领域如果身份验证通过了后，我们是否也可以返回一个类似通信证的凭据给 client，后续请求携带通信证，只要通行证合法且在有效期内，就无需再次鉴权了呢？

是的，etcd 也有类似这样的凭据。当 etcd server 验证用户密码成功后，它就会返回一个 Token 字符串给 client，用于表示用户的身份。后续请求携带此 Token，就无需再次进行密码校验，实现了通信证的效果。

etcd 目前支持两种 Token，分别为 Simple Token 和 JWT Token。

## Simple Token

Simple Token 实现正如名字所言，简单。

Simple Token 的核心原理是当一个用户身份验证通过后，生成一个随机的字符串值 Token 返回给 client，并在内存中使用 map 存储用户和 Token 映射关系。当收到用户的请求时，etcd 会从请求中获取 Token 值，转换成对应的用户名信息，返回给下层模块使用。

Token 是你身份的象征，若此 Token 泄露了，那你的数据就可能存在泄露的风险。etcd 是如何应对这种潜在的安全风险呢？

etcd 生成的每个 Token，都有一个过期时间 TTL 属性，Token 过期后 client 需再次验证身份，因此可显著缩小数据泄露的时间窗口，在性能上、安全性上实现平衡。

在 etcd v3.4.9 版本中，Token 默认有效期是 5 分钟，etcd server 会定时检查你的 Token 是否过期，若过期则从 map 数据结构中删除此 Token。

不过你要注意的是，Simple Token 字符串本身并未含任何有价值信息，因此 client 无法及时、准确获取到 Token 过期时间。所以 client 不容易提前去规避因 Token 失效导致的请求报错。

从以上介绍中，你觉得 Simple Token 有哪些不足之处？为什么 etcd 社区仅建议在开发、测试环境中使用 Simple Token 呢？

首先它是有状态的，etcd server 需要使用内存存储 Token 和用户名的映射关系。


其次，它的可描述性很弱，client 无法通过 Token 获取到过期时间、用户名、签发者等信息。

etcd 鉴权模块实现的另外一个 Token Provider 方案 JWT，正是为了解决这些不足之处而生。

## JWT Token

JWT 是 Json Web Token 缩写，它是一个基于 JSON 的开放标准（RFC 7519）定义的一种紧凑、独立的格式，可用于在身份提供者和服务提供者间，传递被认证的用户身份信息。它由 Header、Payload、Signature 三个对象组成，每个对象都是一个 JSON 结构体。

第一个对象是 Header，它包含 alg 和 typ 两个字段，alg 表示签名的算法，etcd 支持 RSA、ESA、PS 系列，typ 表示类型就是 JWT。

 复制代码


```
1 {  
2   "alg": "RS256",  
3   "typ": "JWT"  
4 }
```

第二对象是 Payload，它表示载荷，包含用户名、过期时间等信息，可以自定义添加字段。

 复制代码

```
1 {  
2   "username": username,  
3   "revision": revision,  
4   "exp":      time.Now().Add(t.ttl).Unix(),  
5 }
```

第三个对象是签名，首先它将 header、payload 使用 base64 url 编码，然后将编码后的字符串用"."连接在一起，最后用我们选择的签名算法比如 RSA 系列的私钥对其计算签名，输出结果即是 Signature。

 复制代码

```
1 signature=RSA256(  
2   base64UrlEncode(header) + "." +  
3   base64UrlEncode(payload),  
4   key)  
5
```

JWT 就是由 `base64UrlEncode(header).base64UrlEncode(payload).signature` 组成。

为什么说 JWT 是独立、紧凑的格式呢？

从以上原理介绍中我们知道，它是无状态的。JWT Token 自带用户名、版本号、过期时间等描述信息，etcd server 不需要保存它，client 可方便、高效的获取到 Token 的过期时间、用户名等信息。它解决了 Simple Token 的若干不足之处，安全性更高，etcd 社区建议大家在生产环境若使用了密码认证，应使用 JWT Token (`--auth-token 'jwt'`)，而不是默认的 Simple Token。

在给你介绍完密码认证实现过程中的两个核心挑战，密码存储安全和性能的解决方案之后，你是否对密码认证的安全性、性能还有所担忧呢？

接下来我给你介绍 etcd 的另外一种高性能、更安全的鉴权方案，x509 证书认证。

## 证书认证

密码认证一般使用在 client 和 server 基于 HTTP 协议通信的内网场景中。当对安全有更高要求的时候，你需要使用 HTTPS 协议加密通信数据，防止中间人攻击和数据被篡改等安全风险。

HTTPS 是利用非对称加密实现身份认证和密钥协商，因此使用 HTTPS 协议的时候，你需要使用 CA 证书给 client 生成证书才能访问。

那么一个 client 证书包含哪些信息呢？使用证书认证的时候，etcd server 如何知道你发送的请求对应的用户名称？

我们可以使用下面的 `openssl` 命令查看 client 证书的内容，下图是一个 x509 client 证书的内容，它含有证书版本、序列号、签名算法、签发者、有效期、主体名等信息，我们重点关注的是主体名中的 CN 字段。

在 etcd 中，如果你使用了 HTTPS 协议并启用了 client 证书认证 (`--client-cert-auth`)，它会取 CN 字段作为用户名，在我们的案例中，alice 就是 client 发送请求的用户名。

```
1 openssl x509 -noout -text -in client.pem
```


```
Data:
  Version: 3 (0x2)
  Serial Number:
    5a:89:c3:e6:52:95:09:fe:b3:bd:11:c0:73:0d:ce:3a:
Signature Algorithm: ecdsa-with-SHA256
  Issuer: C=US, ST=CA, L=San Francisco, CN=example.net
  Validity
    Not Before: Oct 31 15:11:00 2020 GMT
    Not After : Aug 23 15:11:00 2310 GMT
  Subject: C=CN, ST=SH, L=Shanghai, CN=alice
  Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
      Public-Key: (256 bit)
      pub:
        04:85:02:4e:1b:02:7d:b8:fc:97:d9:b5:8f:7
        9c:59:98:58:a6:4e:b4:a7:58:15:1c:ef:66:0
        96:e5:b1:ca:6b:46:98:aa:32:76:f6:91:6a:5
        b9:8e:2f:28:ef:1b:28:20:1a:0c:d1:be:c9:5
        f4:07:d9:d5:b0
```

证书认证在稳定性、性能上都优于密码认证。

稳定性上，它不存在 Token 过期、使用更加方便、会让你少踩坑，避免了不少 Token 失效而触发的 Bug。性能上，证书认证无需像密码认证一样调用昂贵的密码认证操作 (Authenticate 请求)，此接口支持的性能极低，后面实践篇会和你深入讨论。

## 授权

当我们使用如上创建的 alice 账号执行 put hello 操作的时候，etcd 却会返回如下的 "etcdserver: permission denied" 无权限错误，这是为什么呢？

 复制代码

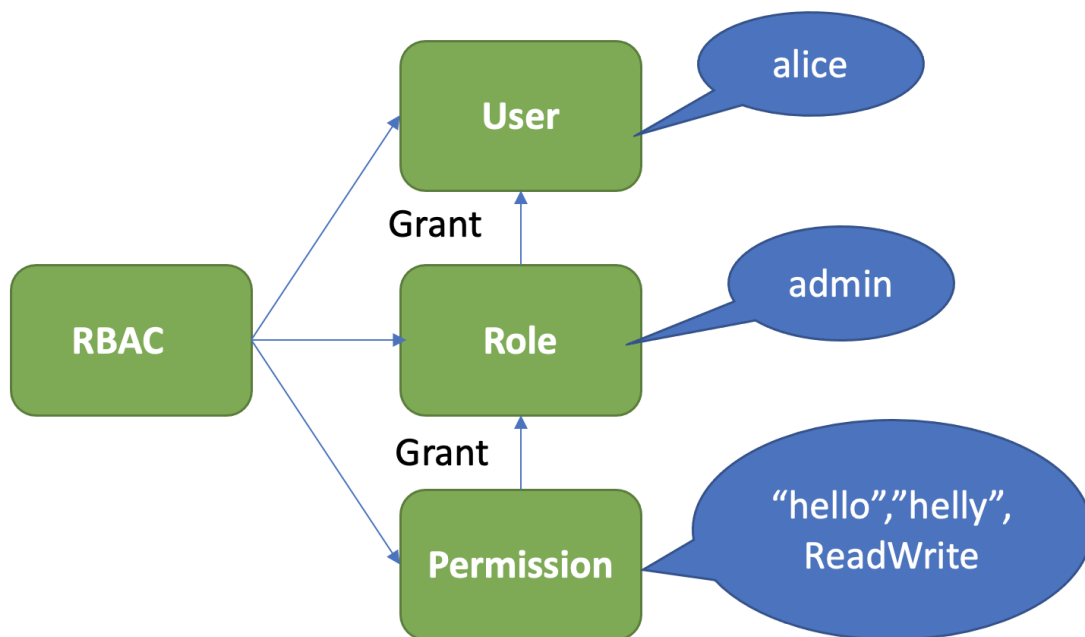
```
1 $ etcdctl put hello world --user alice:alice
2 Error: etcdserver: permission denied
```

这是因为开启鉴权后，put 请求命令在应用到状态机前，etcd 还会对发出此请求的用户进行权限检查，判断其是否有权限操作请求的数据。常用的权限控制方法有 ACL(Access Control List)、ABAC(Attribute-based access control)、RBAC(Role-based access control)，etcd 实现的是 RBAC 机制。

## RBAC


什么是基于角色权限的控制系统 (RBAC) 呢？

它由下图中的三部分组成，User、Role、Permission。User 表示用户，如 alice。Role 表示角色，它是权限的赋予对象。Permission 表示具体权限明细，比如赋予 Role 对 key 范围在[key, KeyEnd]数据拥有什么权限。目前支持三种权限，分别是 READ、WRITE、READWRITE。



下面我们通过 etcd 的 RBAC 机制，给 alice 用户赋予一个可读写[hello,helly]数据范围的读写权限，如何操作呢？

按照上面介绍的 RBAC 原理，首先你需要创建一个 role，这里我们命名为 admin，然后新增了一个可读写[hello,helly]数据范围的权限给 admin 角色，并将 admin 的角色的权限授予了用户 alice。详细如下：


 复制代码

```
1 $ #创建一个admin role
2 etcdctl role add admin --user root:root
3 Role admin created
4 # #分配一个可读写[hello, helly]范围数据的权限给admin role
5 $ etcdctl role grant-permission admin readwrite hello helly --user root:root
6 Role admin updated
7 # 将用户alice和admin role关联起来，赋予admin权限给用户
8 $ etcdctl user grant-role alice admin --user root:root
9 Role admin is granted to user alice
```

然后当你再次使用 `etcdctl` 执行 `put hello` 命令时，鉴权模块会从 `boltdb` 查询 `alice` 用户对应的权限列表。

因为有可能一个用户拥有成百上千个权限列表，`etcd` 为了提升权限检查的性能，引入了区间树，检查用户操作的 `key` 是否在已授权的区间，时间复杂度仅为  $O(\log N)$ 。

在我们的这个案例中，很明显 `hello` 在 `admin` 角色可读写的 `[hello, helly)` 数据范围内，因此它有权限更新 `key hello`，执行成功。你也可以尝试更新 `key hey`，因为此 `key` 未在鉴权的数据区间内，因此 `etcd server` 会返回 `"etcdserver: permission denied"` 错误给 `client`，如下所示。

 复制代码

```
1 $ etcdctl put hello world --user alice:alice
2 OK
3 $ etcdctl put hey hey --user alice:alice
4 Error: etcdserver: permission denied
```

## 小结

最后我和你总结下今天的内容，从 `etcd` 鉴权模块核心原理分析过程中，你会发现设计实现一个鉴权模块最关键的目标和挑战应该是安全、性能以及一致性。

首先鉴权目的是为了保证安全，必须防止恶意用户绕过鉴权系统、伪造、篡改、越权等行为，同时设计上要有前瞻性，做到即使被拖库也影响可控。`etcd` 的解决方案是通过密码安全加密存储、证书认证、RBAC 等机制保证其安全性。

然后，鉴权作为了一个核心的前置模块，性能上不能拖后腿，不能成为影响业务性能的一个核心瓶颈。etcd 的解决方案是通过 Token 降低频繁、昂贵的密码验证开销，可应用在内网、小规模业务场景，同时支持使用证书认证，不存在 Token 过期，巧妙的取 CN 字段作为用户名，可满足较大规模的业务场景鉴权诉求。

接着，鉴权系统面临的业务场景是复杂的，因此权限控制系统应当具备良好的扩展性，业务可根据自己实际场景选择合适的鉴权方法。etcd 的 Token Provider 和 RBAC 扩展机制，都具备较好的扩展性、灵活性。尤其是 RBAC 机制，让你可以精细化的控制每个用户权限，实现权限最小化分配。

最后鉴权系统元数据的存储应当是可靠的，各个节点鉴权数据应确保一致，确保鉴权行为一致性。早期 etcd v2 版本时，因鉴权命令未经过 Raft 模块，存在数据不一致的问题，在 etcd v3 中通过 Raft 模块同步鉴权指令日志指令，实现鉴权数据一致性。

## 思考题

最后，我给你留了一个思考题。你在使用 etcd 鉴权特性过程中遇到了哪些问题？又是如何解决的呢？

感谢你的阅读，欢迎你把思考和观点写在留言区，也欢迎你把这篇文章分享给更多的朋友一起阅读。

## 04 思考题参考答案

04 讲的思考题 mckee 同学给出了精彩回答，下面是他的回答。

1. 哪些场景会出现 Follower 日志与 Leader 冲突？

leader 崩溃的情况下可能 (如老的 leader 可能还没有完全复制所有的日志条目)，如果 leader 和 follower 出现持续崩溃会加剧这个现象。follower 可能会丢失一些在新的 leader 中有的日志条目，他也可能拥有一些 leader 没有的日志条目，或者两者都发生。

2.follower 如何删除无效日志？

leader 处理不一致是通过强制 follower 直接复制自己的日志来解决。因此在 follower 中的冲突的日志条目会被 leader 的日志覆盖。leader 会记录 follower 的日志复制进度 nextIndex，如果 follower 在追加日志时一致性检查失败，就会拒绝请求，此时 leader 就会减小 nextIndex 值并进行重试，最终在某个位置让 follower 跟 leader 一致。

这里我补充下为什么 WAL 日志模块只通过追加，也能删除已持久化冲突的日志条目呢？其实这里 etcd 在实现上采用了一些比较有技巧的方法，在 WAL 日志中的确没删除废弃的日志条目，你可以在其中搜索到冲突的日志条目。只是 etcd 加载 WAL 日志时，发现一个 raft log index 位置上有多个日志条目时，会通过覆盖的方式，将最后写入的日志条目追加到 raft log 中，实现了删除冲突日志条目效果，你如果感兴趣可以参考下我和 Google ptabor [关于这个问题的讨论](#)。

提建议

12.12 大促

## 每日一课 VIP 年卡

10分钟，解决你的技术难题

¥159/年 ¥365/年

每日一课  
VIP 年卡

仅3天，【点击】图片，立即抢购 >>>

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | Raft协议：etcd如何实现高可用、数据强一致的？

下一篇 06 | 租约：如何检测你的客户端存活？

## 精选留言 (10)

写留言



唐聪

2021-01-29

本讲是一个微型鉴权系统设计与实现分析，曾遇到好几次业务使用密码鉴权出现问题的，原因就在于大家对密码鉴权、适用场景，了解太少导致线上出问题，希望通过本讲帮助大家搞清楚etcd鉴权的一切



11

不瘦二十斤  
不改头像

jeffery

2021-01-31

每章都干货 多租户模式下怎么设置鉴权、etcd不知道后期会不会支持多数据中心！相比Nacos、怎么选型！谢谢老师

展开

作者回复: 感谢jeffery同学的高度认可，如果有收获可以分享给更多人，谢谢你的支持。多租户模式下，如果使用密码鉴权，你可以直接调用API给不同租户添加账号、密码，每个租户分配一个唯一的路径，然后按路径前缀分配权限。如果使用的证书鉴权，你可以为每个用户生成证书，证书的CN字段为用户的名称即可，建议使用证书鉴权。同时重要业务不建议多租户模式哈，多租户场景不同租户可能会相互影响，导致各种稳定性问题，除非各个租户的行为是可控的，可信赖的。etcd选型、多数据中心这些后面实践篇将为你分析。



5



云原生工程师

2021-01-29

密码鉴权简单易用，但是潜在隐患多，证书可能略麻烦，特别是多租户场景，每个用户证书都不一样，需要独立生成，总的而言，还是不能为一时方便偷懒选用密码

作者回复: 是的



5

**领程**

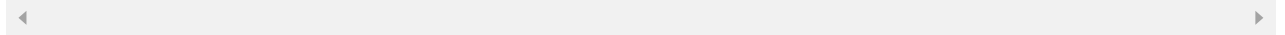
2021-02-03

1) 撞库：黑客通过收集互联网已泄露的用户和密码信息，生成对应的字典表，尝试批量登陆其他网站后，得到一系列可以登录的用户。说的简单一点，就是一个小偷，入室盗窃后偷到了一串钥匙，然后他拿着这串钥匙，在整个小区里面挨家挨户的进行开锁。这个过程就是撞库。

...

展开 ∨

作者回复: 赞



1

**一步**

2021-02-02

对于数据授权，能不能实现 模糊匹配呢？比如 hello\* 可以操作前缀是 hello 而且后面可以跟任意的一个字符的 key

展开 ∨



1

**石小**

2021-02-18

老师过年好，使用jwt token如果server不保存用户名，client 发来的用户名和签名若经过篡改的，服务器怎么知道呢？

展开 ∨

**Alery**

2021-02-07

"鉴权模块首先会根据你请求的用户名 alice，从 boltdb 获取加密后的密码，因此 hash 值包含了算法版本、salt、cost 等信息，因此可以根据你请求中的明文密码，计算出最终的 hash 值，若计算结果与存储一致，那么身份校验通过"

老师，这一段不太理解，hacker如果拿到密码也可知道 "算法版本、salt、cost 等信...

展开 ∨

**akai**

2021-02-06

老师，这里给范围 key 加权限用的[]，我看官方文档给的 [ ],是两种都适用吗

The range can be specified as an interval [start-key, end-key) where start-key should be lexically less than end-key in an alphabetical manner.

展开 ▾

作者回复: 嗯，细心，是)，感谢反馈，已修正



**mckee**  
2021-02-02

遇到的问题(k8s场景):  
主要是可能会出现apiserver etcd client证书可能过期问题，需要有监控手段，需要对其续签。当然方便的话可以直接签很久



**Simon**  
2021-02-01

老师, 上一节的思考题答案还没有更新吗?  
展开 ▾

作者回复: 好的，明天更新到05