微服务架构之「 访问安全 」

Java知音 2019年06月18日 08:46

以下文章来源于不止思考 ， 作者奎哥

不止思考

不止思考，坚持原创。每周分享 个人成长、技术干货 的文章。每一篇都是作者10余年...

> 作者：爱奎哥
>
> 公众号：不止思考

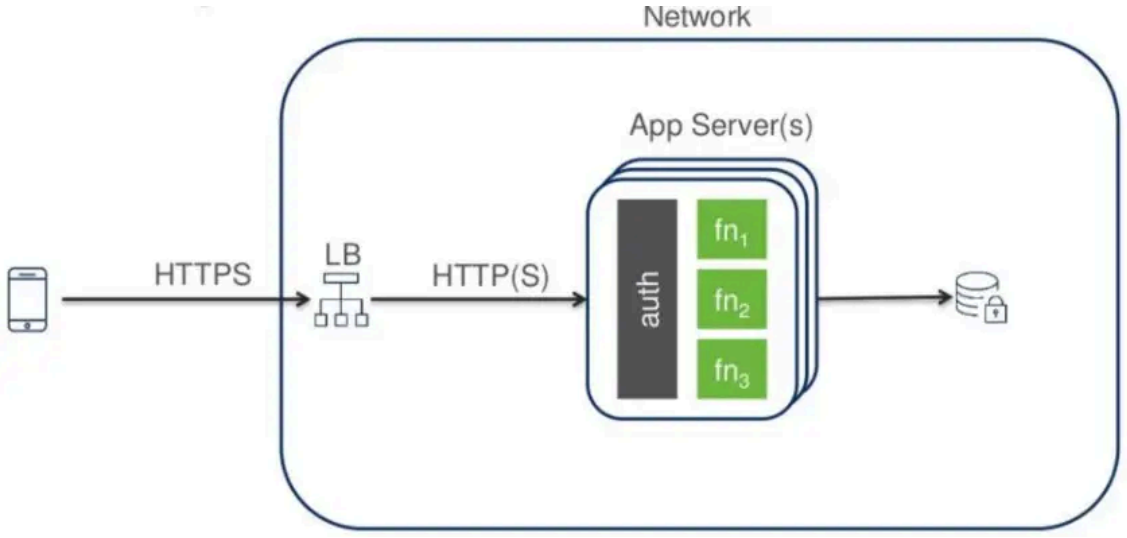
技术经验交流：点击入群

应用程序的访问安全又是我们每一个研发团队都必须关注的重点问题。尤其是在我们采用了微服务架构之后，项目的复杂度提升了N个级别，相应的，微服务的安全工作也就更难更复杂了。并且我们以往擅长的单体应用的安全方案对于微服务来说已经不再适用了。我们必须有一套新的方案来保障微服务架构的安全。

在探索微服务访问安全之前，我们还是先来回顾一下单体应用的安全是如何实现的。

一、传统单体应用如何实现「访问安全」？

下图就是一个传统单体应用的访问示意图：



（图片来自WillTran在slideshare分享）

在应用服务器里面，我们有一个auth模块（一般采用过滤来实现），当有客户端请求进来时，所有的请求都必须首先经过这个auth来做身份验证，验证通过后，才将请求发到后面的业务逻辑。

通常客户端在第一次请求的时候会带上身份校验信息（用户名和密码），auth模块在验证信息无误后，就会返回Cookie存到客户端，之后每次客户端只需要在请求中携带Cookie来访问，而auth模块也只需要校验Cookie的合法性后决定是否放行。

可见，在传统单体应用中的安全架构还是蛮简单的，对外也只有一个入口，通过auth校验后，内部的用户信息都是内存/线程传递，逻辑并不是复杂，所以风险也在可控范围内。

那么，当我们的项目改为微服务之后，「访问安全」又该怎么做呢。

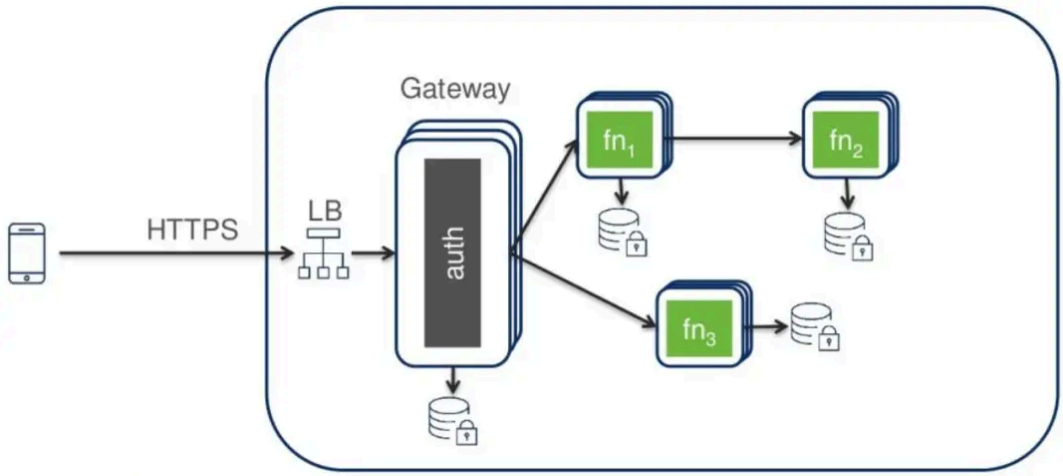
二、微服务如何实现「访问安全」？

在微服务架构下，有以下三种方案可以选择，当然，用的最多的肯定还是OAuth模式。

- **网关鉴权模式（API Gateway）**
- **服务自主鉴权模式**
- **API Token模式（OAuth2.0）**

下面分别来讲一下这三种模式：

1. **网关鉴权模式（API Gateway）**



（图片来自WillTran在slideshare分享）

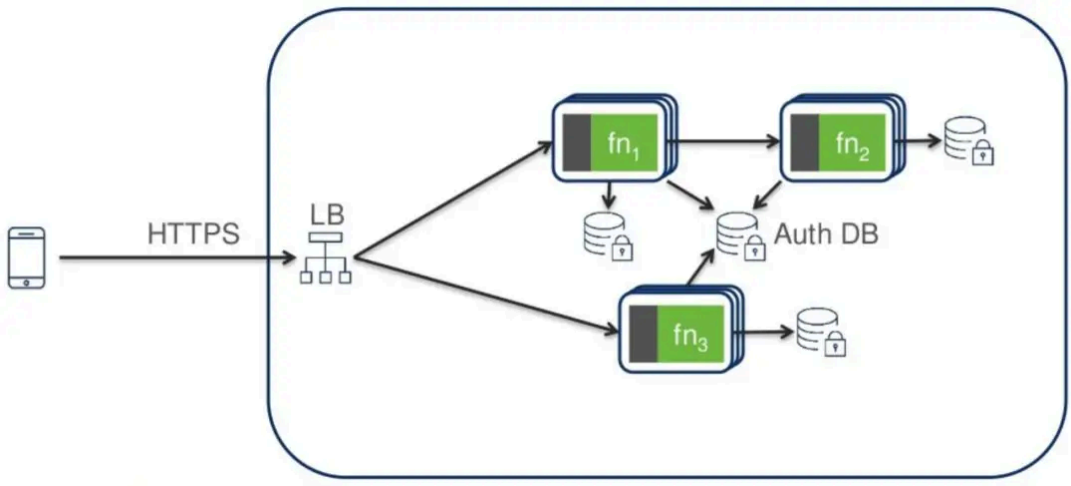
通过上图可见，因为在微服务的最前端一般会有一个API网关模块（API Gateway），所有的外部请求访问微服务集群时，都会首先通过这个API

Gateway，所以我们可以在这个模块里部署auth逻辑，实现统一集中鉴权，鉴权通过后，再把请求转发给后端各个服务。

这种模式的优点就是，由API Gateway集中处理了鉴权的逻辑，使得后端各微服务节点自身逻辑就简单了，只需要关注业务逻辑，无需关注安全性事宜。

这个模式的问题就是，API Gateway适用于身份验证和简单的路径授权（基于URL的），对于复杂数据/角色的授权访问权限，通过API Gateway很难去灵活的控制，毕竟这些逻辑都是存在后端服务上的，并非存储在API Gateway里。

2. **服务自主鉴权模式**



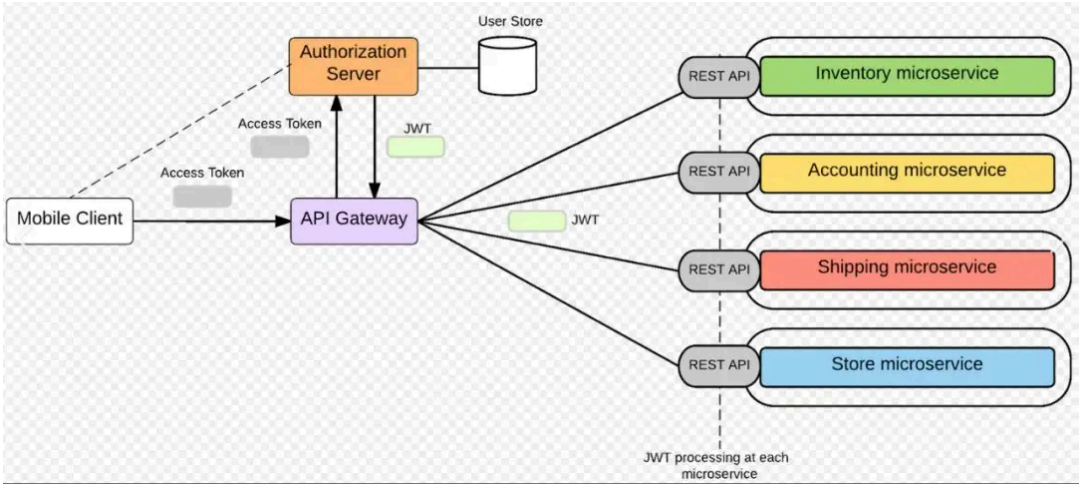
（图片来自WillTran在slideshare分享）

服务自主鉴权就是指不通过前端的API Gateway来控制，而是由后端的每一个微服务节点自己去鉴权。

它的优点就是可以由更为灵活的访问授权策略，并且相当于微服务节点完全无状态化了。同时还可以避免API Gateway 中 auth 模块的性能瓶颈。

缺点就是由于每一个微服务都自主鉴权，当一个请求要经过多个微服务节点时，会进行重复鉴权，增加了很多额外的性能开销。

3. **API Token模式（OAuth2.0）**



（图片来自网络）

如图，这是一种采用基于令牌Token的授权方式。在这个模式下，是由授权服务器（图中Authorization Server）、API网关（图中API Gateway）、内部的微服务节点几个模块组成。

流程如下：

第一步：客户端应用首先使用账号密码或者其它身份信息去访问授权服务器（Authorization Server）获取 访问令牌（Access Token）。

第二步：拿到访问令牌（Access Token）后带着它再去访问API网关（图中API Gateway），API Gateway自己是无法判断这个Access Token是否合法的，所以走第三步。

第三步：API Gateway去调用Authorization Server校验一下Access Token的合法性。

第四步：如果验证完Access Token是合法的，那API Gateway就将Access Token换成JWT令牌返回。

（注意：此处也可以不换成JWT而是直接返回原Access Token。但是换成JWT更好，因为Access Token是一串不可读无意义的字符串，每次验证Access Token是否合法都需要去访问Authorization Server才知道。但是JWT令牌是一个包含JOSN对象，有用户信息和其它数据的一个字符串，后面微服务节点拿到JWT之后，自己就可以做校验，减少了交互次数）。

第五步：API Gateway有了JWT之后，就将请求向后端微服务节点进行转发，同时会带上这个JWT。

第六步：微服务节点收到请求后，读取里面的JWT，然后通过加密算法验证这个JWT，验证通过后，就处理请求逻辑。

这里面就使用到了OAuth2.0的原理，不过这只是OAuth2.0各类模式中的一种。

由于OAuth2.0目前最为常用，所以接下来我再来详细讲解一下OAuth2.0的原理和各类用法。

三、详解 OAuth2.0 的「 访问安全 」？

OAuth2.0是一种访问授权协议框架。它是基于Token令牌的授权方式，在不暴露用户密码的情况下，使 应用方 能够获取到用户数据的访问权限。

例如：你开发了一个视频网站，可以采用第三方微信登陆，那么只要用户在微信上对这个网站授权了，那这个网站就可以在无需用户密码的情况下获取用户在微信上的头像。

OAuth2.0 的流程如下图：

OAuth2.0 里的主要名词有：

- **资源服务器**：用户数据/资源存放的地方，在微服务架构中，服务就是资源服务器。在上面的例子中，微信头像存放的服务就是资源服务器。
- **资源拥有者**：是指用户，资源的拥有人。在上面的例子中某个微信头像的用户就是资源拥有者。
- **授权服务器**：是一个用来验证用户身份并颁发令牌的服务器。
- **客户端应用**：想要访问用户受保护资源的客户端/Web应用。在上面的例子中的视频网站就是客户端应用。
- **访问令牌**：Access Token，授予对资源服务器的访问权限额度令牌。
- **刷新令牌**：客户端应用用于获取新的 Access Token 的一种令牌。
- **客户凭证**：用户的账号密码，用于在 授权服务器 进行验证用户身份的凭证。

OAuth2.0有四种授权模式，也就是四种获取令牌的方式：授权码、简化式、用户名密码、客户端凭证。

下面来分别讲解一下：

1. **授权码（Authorization Code）**

   授权码模式是指：客户端应用先去申请一个授权码，然后再拿着这个授权码去获取令牌的模式。这也是目前最为常用的一种模式，安全性比较高，适用于我们常用的前后端分离项目。通过前端跳转的方式去访问 授权服务器 获取授权码，然后后端再用这个授权码访问 授权服务器 以获取 访问令牌。

   流程如上图。

   第一步，客户端的前端页面(图中UserAgent)将用户跳转到 授权服务器(Authorization Server)里进行授权，授权完成后，返回 授权码(Authorization Code)

   第二步，客户端的后端服务(图中Client)携带授权码(Authorization Code)去访问授权服务器，然后获得正式的 访问令牌(Access Token)

   页面的前端和后端分别做不同的逻辑，前端接触不到Access Token，保证了Access Token的安全性。

2. **简化式（Implicit）**

简化模式是在项目是一个纯前端应用，在没有后端的情况下，采用的一种模式。

因为这种方式令牌是直接存在前端的，所以非常不安全，因此令牌的有限期设置就不能太长。

其流程就是：

第一步：应用（纯前端的应用）将用户跳转到 授权服务器(Authorization Server)里进行授权，授权完成后，授权服务器 直接将 Access Token 返回给 前端应用，令牌存储在前端页面。

第二步：应用（纯前端的应用）携带 访问令牌(Access Token) 去访问资源，获取资源。

在整个过程中，虽然令牌是在前端URL中直接传递，但注意，令牌在HTTP协议中不是放在URL参数字段中的，而是放在URL锚点里。因为锚点数据不会被浏览器发到服务器，因此有一定的安全保障。

3. **用户名密码（Resource Owner Credentials）**

这种方式最容易理解了，直接使用用户的用户名/密码作为授权方式去访问 授权服务器，从而获取Access Token，这个方式因为需要用户给出自己的密码，所以非常的不安全性。一般仅在客户端应用与授权服务器、资源服务器是归属统一公司/团队，互相非常信任的情况下采用。

4. **客户端凭证（Client Credentials）**

这是适用于服务器间通信的场景。客户端应用拿一个用户凭证去找授权服务器获取Access Token。

以上，就是对微服务架构中「访问安全」的一些思考。

在微服务架构的系列文章中，前面已经通过文章介绍过了「服务注册」、「服务网关」、「配置中心 」、「 监控系统 」、「调用链监控」，「容错隔离」大家可以翻阅历史文章查看。

推荐阅读

1. SpringBoot 整合篇
2. ***手写一套迷你版HTTP服务器***
3. 记住：永远不要在MySQL中使用UTF-8
4. ***Springboot启动原理解析***

看完本文有收获？请转发分享给更多人