04 微服务实战：从架构到发布（二）

MAR 2016 IN 后端编程,架构

引言：上篇文章介绍了微服务和单体架构的区别、微服务的设计、消息、服务间通信、数据去中心化，本篇会继续深入微服务，介绍其它特性。

治理去中心化

通常“治理”的意思是构建方案，并且迫使人们通过努力达到组织的目标。SOA治理指导开发者开发可重用的服务，以及随着时间推移，服务应该怎么被设计和开发。治理建立了服务提供者和消费者之间对于服务的协定，告诉消费者能从服务提供获取到什么样的支持。

SOA中有两种常见的治理：

- 设计时的治理-定义和控制服务的创建、设计和服务策略的实施。
- 运行时的治理-确保执行过程的策略。

那么微服务中的治理是什么意思呢？

在微服务架构中，不同的微服务之间相互独立，并且基于不同的平台和技术。因此，没有必要为服务的设计和开发定义一个通用的标准。

总结微服务的治理去中心化如下：

- 微服务架构，在设计时不需要集中考虑治理。
- 每个微服务可以有独立的设计、执行决策。
- 微服务架构着重培养通用/可重用的服务。
- 运行时的治理，比如安全级别保证（SLA），限制，监控，安全和服务发现，可以在API网关层处理。

服务注册和发现

微服务架构下，有大量的微服务需要处理。由于微服务的快速和敏捷研发，他们的位置可能会动态变化。因此在运行时需要能够发现服务所在的位置，服务发现可以解决这个问题。

服务注册

注册中心有微服务的实例和位置信息，微服务在启动时向注册中心注册自己的信息，关闭时注销。其它使用者能够通过注册中心找到可用的微服务和相关信息。

服务发现

为了能找到可用的服务和他们的位置信息，需要服务发现机制。有两种发现机制，客户端发现和服务端发现。

客户端发现 – 客户端或者API网关通过查询服务注册中心或者服务的位置信息。

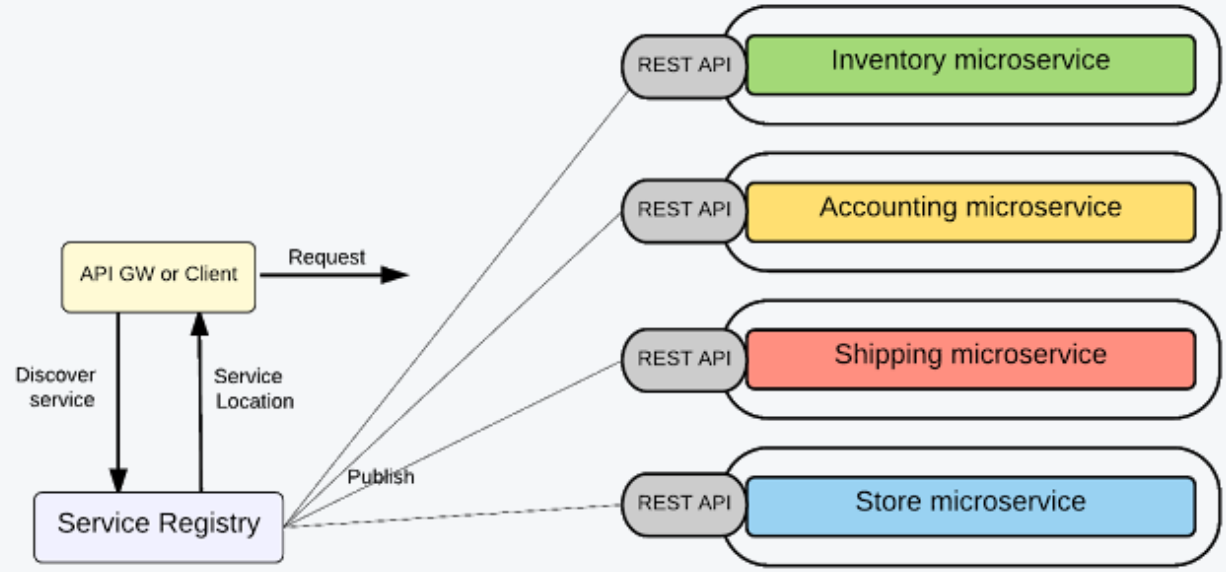

图9：客户端发现

客户端/API网关必须调用服务注册中心组件，实现服务发现的逻辑。

服务端发现 – 客户端/API网关把请求发送到已知位置信息的组件（比如负载均衡器）。组件去访问注册中心，找到微服务的位置信息。

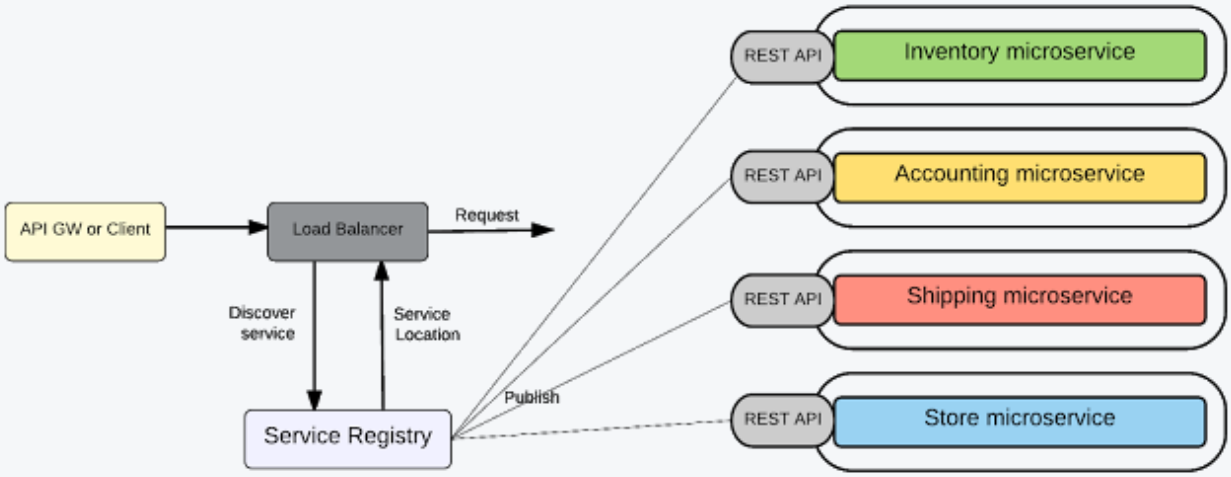

图10：服务端发现

类似Kubernetes（http://kubernetes.io/v1.1/docs/user-guide/services.html ）这种微服务部署解决方案，就提供了服务器端的自动发现机制。

部署

微服务的部署方式也特别重要，以下是关键：

- 能够独立于其他微服务发布或者取消发布
- 微服务可以水平扩展（某一个服务比其他的请求量大）
- 快速构建和发布
- 微服务之间的功能不相互影响

Docker（一个运行在linux上并且开源的应用，能够协助开发和运维把应用运行在容器中）能够快速部署微服务，包括关键几点：

- 把微服务打包成Docker镜像
- 启动容器实例
- 改变实例的数量达到扩容需求

相对于传统的虚拟机模式，利用docker容器，构建、发布、启动微服务将会变得十分快捷。

通过Kubernetes能够进一步扩展Docker的能力，能够从单个linux主机扩展到linux集群，支持多主机，管理容器位置，服务发现，多实例。都是微服务需求的重要特性。因此，利用Kubernetes管理微服务和容器的发布，是一个非常有力的方案。

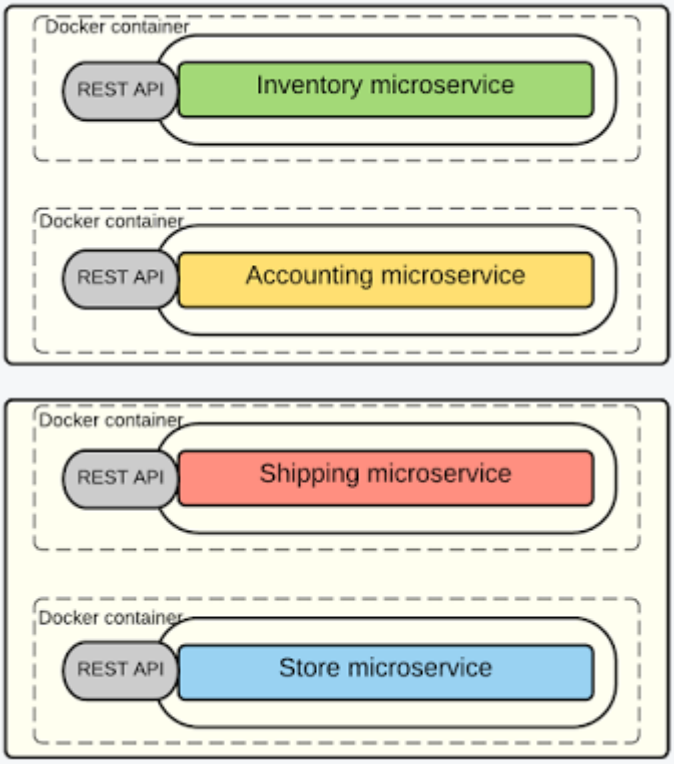

图11：构建和部署服务的容器

图11，展示了零售应用的微服务部署。每个服务都在独立的容器中，每个主机有两个容器，通过kubernetes可以随意调整容器的数量。

安全

在实际运行环境中，微服务的安全也非常重要。我们先看下单体架构下安全是如何实现的。

一个典型的单体应用，安全问题主要是“谁调用”，“调用者能做什么”，“如何处理”。服务器接收到请求后，一般都在处理链条的最开始，通过安全组件来对请求的信息进行安全处理。

我们能直接把这种处理方式应用在微服务架构中吗？答案是可以的，需要买个微服务都实现一个安全组件从资源中心获取对应的用户信息，实现安全控制。这是比较初级的处理方式。可以尝试采用一些标准的API方式，比如OAuth2和OpenID。深入研究之前，可以先概括下这两种安全协议以及如何使用。

OAuth2-是一个访问委托协议。需要获得权限的客户端，向授权服务申请一个访问令牌。访问令牌没有任何关于用户/客户端的信息，仅仅是一个给授权服务器使用的用户引用信息。因此，这个“引用的令牌”也没有安全问题。

OpenID类似于OAuth，不过除了访问令牌以外，授权服务器还会颁发一个ID令牌，包含用户信息。通常由授权服务器以JWT（JSON Web Token）的方式实现。通过这种方式确保客户和服务器端的互信。JWT令牌是一种“有内容的令牌”，包含用户的身份信息，在公共环境中使用不安全。

现在我们看下如何在网络零售网站中应用这些协议保障微服务的安全。

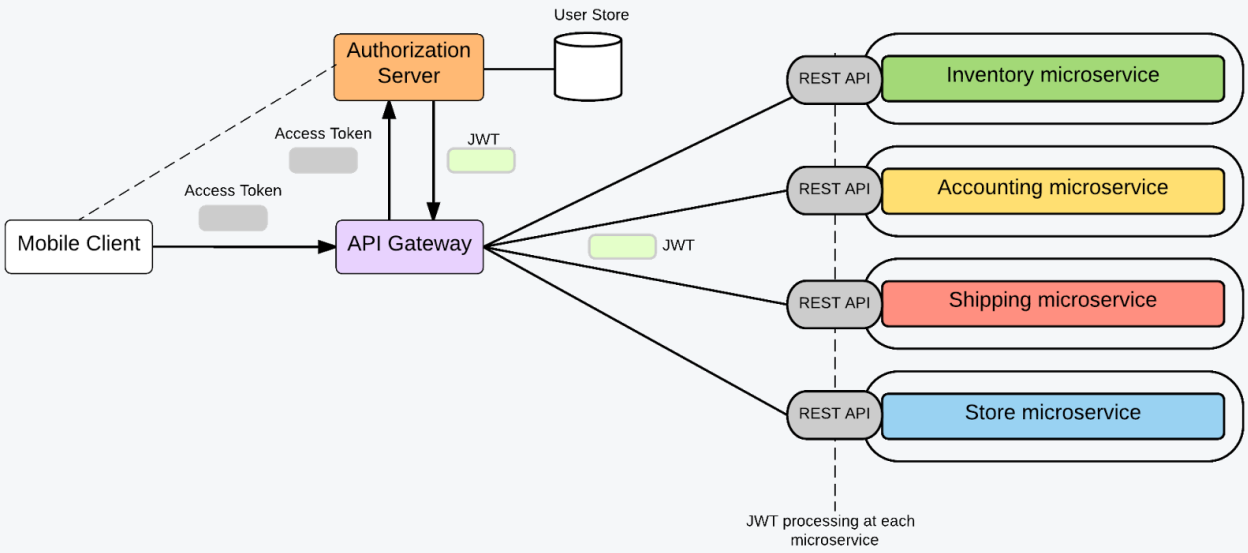

图12：通过OAuth2和OpenID解决安全问题

图12中所示，是实现微服务安全的关键几步：

- 所有的授权由授权服务器，通过OAuth和OpenID方式实现，确保用户能访问到正确的数据。
- 采用API网关方式，所有的客户端请求有唯一入口。
- 客户端通过授权服务器获得访问令牌，把令牌发送到API网关。
- 令牌在网关的处理 – API网关得到令牌后，发送到授权服务器获得JWT。
- 网关把JWT和请求一起发送到微服务中。

JWT包含必要的用户信息，如果每个微服务都能够解析JWT，那么你的系统中每个服务都能处理身份相关的业务。在每个微服务中，可以有一个处理JWT的轻量级的组件。

事务

在微服务中怎么支持事务呢？事实上，跨多个微服务的分布式事务支持非常复杂，微服务的设计思路是尽量避免多个服务之间的事务操作。

解决办法是微服务的设计需要遵循功能自包含和单职责原则。跨越多个微服务支持分布式事务在微服务架构中不是一个好的设计思路，通常需要重新划定微服务的职责。某些场景下，必须要跨越服务支持分布式事务，可以在每个微服务内部利用“组合操作”。

最关键的事情是，基于单职责原则设计微服务，如果某个服务不能正常执行某些操作，那么这个服务是有问题的。那么上游的操作，都需要在各自的微服务中执行回滚操作。

容错设计

微服务架构相比较单体的设计而言，引入了更多服务，在每个服务级别会增加发生错误的可能性。一个服务可能由于网络问题、底层资源等各种问题导致失败。某个服务的不可能不应该影响整个应用的崩溃。因此，微服务系统必须容错，甚至自动回复，对客户端无感知。

任何服务在任何时间都有可能出问题，监控系统需要能够发现问题，并且自动恢复。微服务环境下有不少常用的模式。

线路中断

微服务中请求的失败率达到一定程度后，系统中的监控可以激活线路中断。当正常请求的数量恢复到一定程度后，再关闭线路中断的开关，使系统回复到正常状态。

这个模式可以避免不必要的资源消耗，请求的处理延迟会导致超时，借此可以把监控系统做的更完善。

防火墙

一个应用会有很多微服务租车，单个微服务的失败不应该影响整个系统。防火墙模式强调服务直接的隔离性，微服务不会受到其它微服务失败的影响。

处理超时

超时机制是在确定不会再有应答的情况下，主动放弃等待微服务的响应。这种超时应该是可配置的。

哪些情况下，如何使用这些模式呢？大多数情况，都应该在网关处理。当微服务不可用或者没有回复时，网关能够决定是否执行线路中断或者启动超时机制。防火墙机制同样重要，网关是所有请求的唯一入口，一个微服务的失败不应该影响到其它微服务。网关也是获得微服务状态、监控信息的中心。

微服务，企业集成，API管理

我们已经讨论了微服务的架构和各种特性，以及如何应用在一个现代的IT系统中。同时也需要意识到，微服务不是解决所有问题的灵丹妙药。盲目追求流行的技术概念并不能解决掉企业IT系统的问题。

微服务有很多优势，但是仅靠微服务不能解决企业IT中的所有问题。例如，微服务需要去除ESB，但是现实的IT系统中，大量的应用和服务是基于ESB而不是微服务。集成现有的系统，需要一些集成总线。实际情况是，微服务和其它企业架构并存。

原文作者：Kasun Indrasiri，软件架构师，WSO2
原文链接：https://dzone.com/articles/microservices-in-practice-1
翻译自MaxLeap团队_云服务研发成员：Frank Qin

如果您正在学习移动研发和云服务等方面的讯息，不妨关注我们的微信服务号MaxLeapSvc，我们将不定期推送相关干货。敬请期待！

DOCKER JAVA SOA 分布式 微服务 架构