微服务架构简介

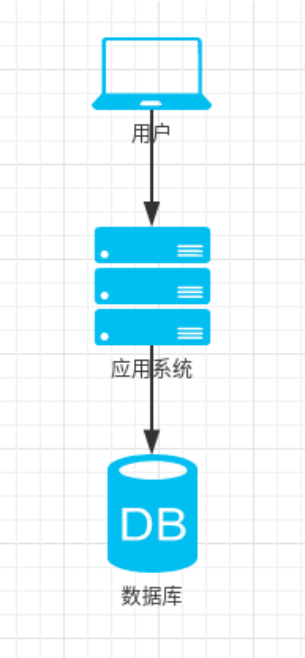
微服务是架构层的一个概念，通过分解（业务单元），将项目拆解出 n 个单元，互相没有强依赖关系（解耦），自我准备需要的依赖条件，进而达到可以独立运行，不再受环境与地点上的限制。

微服务的由来

微服务最早由 Martin Fowler 与 James Lewis 于 2014 年共同提出，微服务架构风格是一种使用一套小服务来开发单个应用的一种方式，每个服务运行在自己的进程中，并使用轻量级机制通信，通常是 HTTP API，这些服务基于业务能力构建，并能够通过自动化部署机制来独立部署，这些服务使用不同的编程语言实现，以及不同数据存储技术，并保持最低限度的集中式管理。

传统的应用模式

我们在以往的传统应用模式下，大致是所有的功能都集成在同一个应用中，这种模式一般被称为单体式开发，即所有的功能打包在一个 war 包内，然后部署在 jee 中，这里面包含了所有的业务逻辑，触发器，大部分的还包含了 ui，如下图：

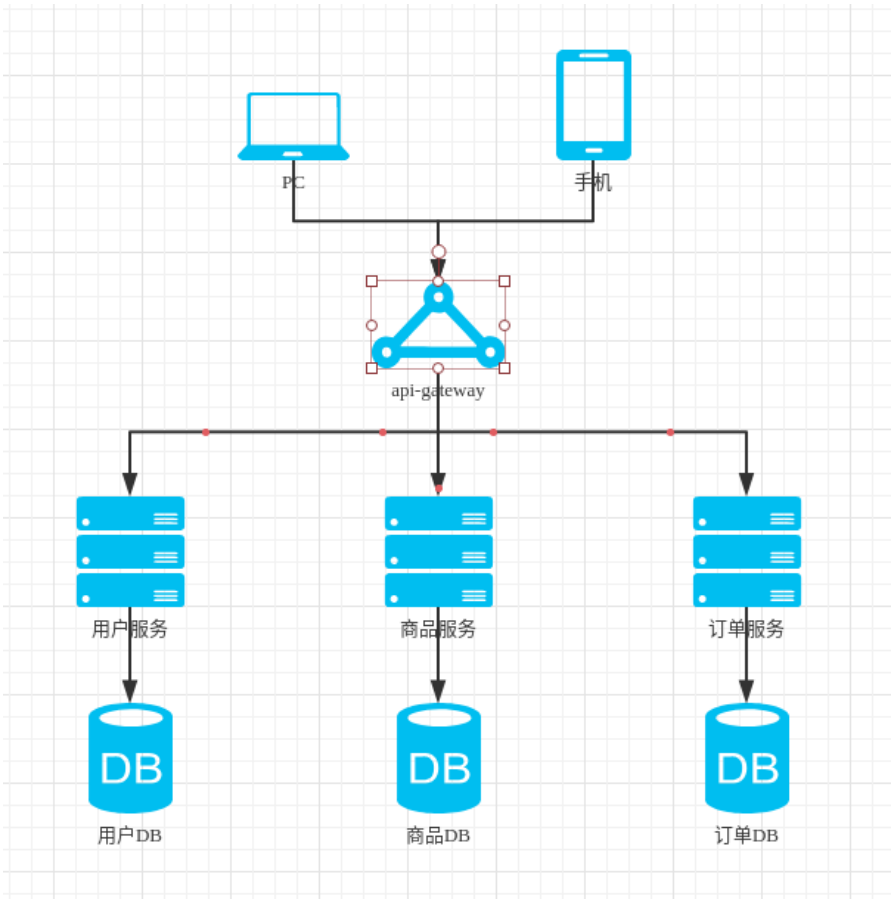


这样的模式下暴露出了他所致命的缺点：

- 代码管理比较难，因为大家都在同一个工程下，会经常的发生冲突
- 新人不太容易上手，因为耦合性太高，调查一个问题时往往会牵扯更多的功能
- 打包危险度大，往往只是提交很小一部分的修改却需要全量打包
- 运维危险度大，可能只是某一个功能崩溃了就会导致整个系统瘫痪
- 不容易扩展，如果只是一个功能的请求量突增，不容易扩展
- 部署要求较高，这样的应用往往需要高配置的主机来承载

微服务的应用模式

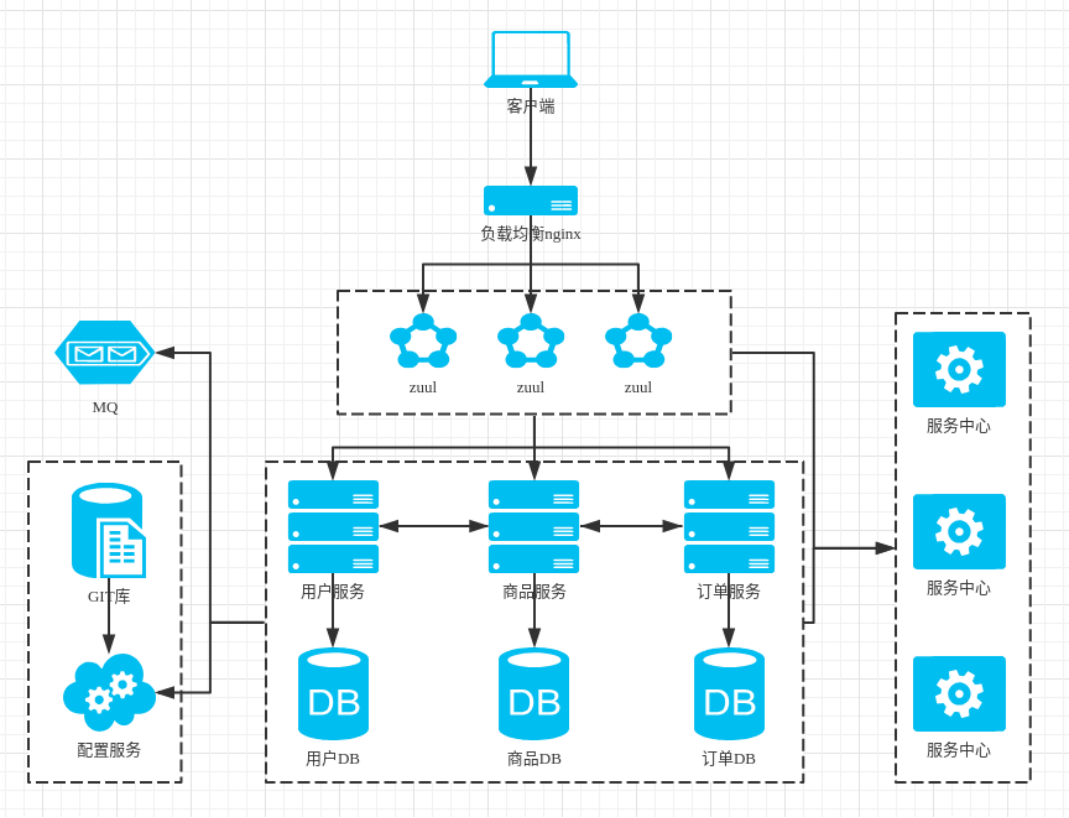
我们在做架构设计的时候，常常需要遵守三个标准（敏捷性、用户体验、成本），基于微服务的架构设计目的就是有效的拆分应用，每个应用单独管理，进而实现敏捷开发和部署，如下图：


PC
手机
api-gateway
用户服务
商品服务
订单服务
DB
DB
DB
用户DB
商品DB
订单DB

由一些独立的应用组合成一个软件系统，每个服务独立运行，跑在自己的微环境中，每个服务独立开发可以按照业务单元进行拆分，实现了跨组织夸地域协同的问题，多个服务采用分布式进行管理，且具有强隔离性。

微服务架构

在微服务架构中，除了每个业务单元的服务外，就是那些服务治理组件了，比如：服务中心、服务消费、负载均衡、断路器、智能路由、配置管理等，这些个组件互相配合再加上业务的各个微服务，共同组建了一个微服务系统，一个简单的微服务系统如下：



用户通过客户端发起请求，nginx 负载到某个 zuul 上然后转发到相应的微服务上，微服务间通过 rpc 或者 mq 进行通信，通过配置服务获取配置数据，最终将整合后的结果返回给用户。

微服务的优缺点

微服务架构之所以流行起来，与他的这些优点密不可分，比如：

1. 他将巨大的单体式应用分解出多个服务，解决了复杂性的问题，在总功能不变的情况下，系统被分解成多个可管理的服务，每个服务都用 rpc/mq 来驱动和定义清晰的 api 边界，为很难实现的功能提供了模块化的解决方案，并且更容易开发和维护。

2. 在这样的架构模式下，可以实现每个服务可由不同的团队来开发，从而放宽了技术选型，只需提供标准的 restapi 即可，在这种自由模式的开发背景下，开发者可以选择较新的技术，由于每个服务的功能很小，所以开发的难度也很低，即使出现了代码重写的问题难度也不是很大。

3. 由于微服务采用的是独立部署，开发者在部署的时候不用考虑其他的服务对自己的影响，这种改变加快了部署速度并减少了部署风险，微服务架构使 ci/cd 成为可能。

但是微服务也存在一些不足，比如：

1. 微服务采用的分布式系统，故而会产生固有的复杂性，开发者需要在 rpc 等消息传递之间完成进程间的通讯，更或者需要用代码来解决消息传递过慢或异常的情况。

2. 来源于数据库事务的分布，以至于我们不能达到强一致性，只能的选择最终一致性

3. 当我们对某个服务的某个 api 进行测试的时候，需要保证这个服务所依赖的其他服务都是正常开启的状态，这给测试带来了复杂性的问题。

Fred Brooks 曾写过 there are no silver bullets，像其他科技一样，微服务也有很多不足，所以在做系统架构之前首先要确定最终的目的，是否有效的拆分应用，是否需要实现敏捷开发、敏捷部署。

思想上的转变

微服务对于我们来说，技术上不是问题，更多的是思维逻辑，对于微服务架构我们应该主动的在思维上进行转变。

主要有一下几点：

- 应用的单元是业务逻辑，按照业务进行服务的拆分
- 做有生命的产品，而不是项目
- 培养团队内部核心人员，微服务架构师，全栈专家
- 单一职责的原则，每个服务只干一件事情
- 部署方式向 docker 转型
- 开发模式向 devops 转型

同时，对于开发同学，有这么多的中间件和强大的 PE 支持固然是好事，我们也需要深入去了解这些中间件背后的原理，知其然知其所以然。

最后，一般提到微服务都离不开 DevOps 和 Docker，理解微服务架构是核心，devops 和 docker 则是工具，是手段。