01从单体架构到微服务架构的演进

1. Monolithic单体式架构解析  
   Monolithic单体式架构指的是尽管是模块化逻辑，但是最终还是会打包并且部署为一个单一应用，具体的格式依赖于应用语言和框架，例如，部分java应用会被打包为WAR格式，部署在Tomcat上或者JIT上，而另外一些java应用会被打包为自包含的jar格式，同样，Reals和node.js会被打包为层级目录。  
   2. Monolithic单体式架构的优缺点  
   优点：开发工具IDE和其他工具都擅长开发一个简单应用，这类应用也很易于调试和部署。只需要把打包应用拷贝到服务器端，通过在负载均衡器后端运行多个拷贝，就可以轻松是实现应用扩展。  
   缺点：单体式架构一旦随着时间的推移，逐渐的变大，敏捷开发和部署举步维艰。任何单个开发者都很难搞懂它。修正bug和正确的添加新功能变得非常困难且很耗时。  
   3. Monolithic单体式架构面临的挑战  
   随着市场变化快用户需求变化快、用户访问量增加的同时，单块架构应用的维护成本、人员的培养成本、缺陷修复成本、技术架构演进的成本、系统扩展成本等都在增加。单块架构曾经的优势已逐渐不在适应互联网时代的快速变化。  
   4.微服务架构模式倡导的做法  
   Microservice微服务架构是一种架构模式，提倡将Monolithic单体式架构应用化分为一系列小的服务，服务之间相互协调，相互配合，为用户提供服务。每个服务运行于其独立的进程中，服务之间采用轻量级的协议进行通信，每个服务都围绕着具体业务进行构建，并能够独立部署。  
   微服务架构的优点：每个服务能够内聚，代码容易理解，开发效率高，服务之间可以独立部署，使得持续部署成为可能，容易针对每个服务组件开发团队，容错性也大大提高。  
   5.向微服务架构演进的推荐顺序  
   先规划，然后是中间件和数据库，最后是服务和应用

02 基于Docker的微服务应用架构设计

如今，软件通常会作为一种服务来交付，它们被称为网络应用程序，或“软件即服务”（SaaS）。  
1.微服务架构设计需要遵循的模式  
比较知名的“12-factor”  
2.“12-factor”为构建如下的SAAS应用提供了方法论  
使用标准化流程自动配置，从而使得新的开发者花费最少的成本加入这个项目；和操作系统之间尽可能的划清界限，在各个系统中提供最大的可移植性，适合部署在现在的云计算平台，从而在服务器和系统管理方面节省资源。将开发环境和可生产环境的差异降至最低，并使用实施交付实现敏捷开发。可以在工具、架构、开发流程不发生明显变化的前提下实施扩展。  
3.“12-factor”中比较常用的几个：  
基准代码  
基准代码和应用之间总是保持一一对应的关系：一旦有多个基准代码，就不能称为一个应用，而是一个分布式系统。分布式系统中的每一个组件都是一个应用，每一个应用可以使用“12-factor”进行开发。  
多个应用共享一份基准代码是有悖于“12-factor”原则的，解决方案是将共享的代码拆分为独立的类库，然后使用依赖管理策略去使用管理他们。  
（显示声明）依赖关系  
应用程序不会隐式依赖系统级的类库，他通过依赖清单，确切的声明所有的依赖项。  
此外，在运行过程中通过依赖隔离工具来确保程序不会调用系统中存在但清单中未声明的依赖项。这一做法会统一应用到生产和开发环境。  
配置（config）  
推荐将应用的配置存储于环境变量中，环境变量可以非常方便的在不同的部署之间做修改，却不动一行代码。  
于配置文件不同，一不小心把他们嵌入代码库的概率微乎其微  
与一些传统的解决配置问题的机制（比如java的属性配置文件）相比，环境变量与语言和系统无关。  
后端服务  
“12-factor”应用不会区别对待本地或者第三方服务，对应用程序而言，两种都是附加资源。  
“12-factor”应用支持任意部署，如：可以在不进行惹任何代码改动的情况下，将本地的mysql数据库换成第三方服务。

依赖--显式声明依赖关系

12-Factor规则下的应用程序不会隐式依赖系统级的类库。 它一定通过依赖清单 ，确切地声明所有依赖项。此外，在运行过程中通过 依赖隔离 工具来确保程序不会调用系统中存在但清单中未声明的依赖项。这一做法会统一应用到生产和开发环境。

03 基于容器的微服务架构剖析

1. Docker本质

Docker是一个微容器，一个云计算的微pass容器，类似jvm(JVM是Java Virtual Machine（Java[虚拟机](http://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA)）的缩写),比其更加强大，直接基于Linux内核；

比VM虚拟机更加轻量，能够在Linux、云计算、IS等平台上直接运行，带着你的应用无缝的移植到各种运行环境。

Docker本质是一个允许创建镜像，并让镜像的实例运行在容器中的软件。

所谓镜像往往包含了很多步骤，就像虚拟机的模板一样。

1. Docker流行因素

依赖设计做的不到位，最后只能在运行阶段依靠docker这样的微容器松耦合，Docker的细粒度松耦合能用一个docker 容器装载一个场景功能。也就是按照功能角色分类。

每个Docker里面装一个服务或应用，一个服务器上可以运行多个Docker。系统级的服务比如Mysql数据库、memcash缓存

1. 集群分布微服务

业务应用系统可以应用微服务进行分类，每个Docker中运行一个微服务，

1. Docker对传统Paas平台的挑战

Docker比java ee服务器更强的地方在于，Docker基于Linux内核，可以装载各种语言应用，是一种崭新的PaaS微平台。

1. Docker的安全性

Docker利用容器将资源进行有效隔离。与Linux操作系统和hypervisor有相同的安全运行管理和配置管理级别。

04微服务架构设计模式

1.客户端如何访问这些服务？

原来的Monolithic方式开发，所有的服务都是本地的，UI可以直接调用，而现在按照功能拆分成独立的服务，通常都是运行在虚拟机上的java进程。后台有N个服务，前台就需要记住管理N个服务，一个服务下线、更新、升级，前台就需要重新部署，这明显不符合拆分的理念。特别当前台是移动应用的时候，通常业务变化的节奏更快。另外，N个小服务的调用也是一个不小的网络开销。所以，后台N个服务和UI之间会通过一个代理，或者叫API GITWAY统一提供服务入口，让服务对前台透明，聚合后台的服务，节省流量，提升性能，提供安全、过滤流控等API管理功能。

2.服务之间如何进行通信？

所有的服务都是独立的java进程，跑在独立的虚拟机上，所以服务间的通信就是IPC（Inter process communication）已经有很多的成熟的方案。

基本最通用的是两种方式：**同步调用REST和IPC异步消息调用**

3.这么多的服务，怎么找？

一般每个服务都有多个拷贝来做负载均衡，一个服务随时可能下线，也可能应对临时访问压力，增加新的服务节点。

如何发现：基本都是通过Zookeeper等类似的技术做服务注册信息的分布式管理。

解释：当服务上限时，服务提供者将自己的服务信息注册到ZK或者类似的管理框架，并通过心跳维持长链接，实时更新链接信息，服务调用者通过ZK寻址，根据可定制算法找到一个服务，还可以将服务信息缓存在本地，以便提高性能。当服务下线时，ZK会发通知给服务的客户端。

4.服务挂了怎么办？

分布式一个最大的缺点是：网络是不可靠的

一个提供高容错性的工具：HYSTRIX

当系统是由一系列的系统调用链组成的时候，我们为您必须确保任一环节出问题，都不至于影响整体的电路，相应的手段有：重试机制、限流、负载均衡、降级（本地缓存）

--------------------------------------------------

05 微服务云架构原理

1. 微服务简化

微服务架构带来了诸多优势，构建、部署、维护分布式的微服务系统并不容易。而容器所提供的的轻量级面向应用的虚拟化运行环境为微服务提供了理想的载体。

基于容器技术的云服务将极大的简化容器化微服务创建、集成、部署、运维的整个流程，从而推动微服务在云端的大规模实践。

1. 微服务创建

假设用户的微服务程序，存储于GitHub等代码托管服务中，用户可以将这个代码仓库构建成容器镜像，并保存在镜像仓库中，用户可以将这个微服务一键部署到容器云平台。

云平台提供了持续集成的功能，用户可以选择是否使用。每当微服务的代码有变化时，就构建一个新的容器镜像以便以后部署使用。

1. 微服务集成

用户可以自由组合、复用数以万计的容器化微服务，像搭积木一样轻松集成应用；

比如，用户需要一个通用的Mysql数据库服务，无需构建镜像，可以直接在镜像仓库中选择适合的数据库服务镜像，并与其微服务链接起来；

1. 微服务部署

微服务由于组件数量众多，云端部署成为实践上的一个难点

容器云平台容器为应用发布的载体，用户不必指定传统部署方式中的=繁琐的步骤，只需提供容器镜像和简单的容器配置，平台会将整个部署流程自动化。

1. 微服务运维

微服务由于独立进程众多，部署后的运维、管理成为实践的另一个难点；

容器云平台完全屏蔽底层云主机和基础架构运维，让用户专注于应用；

通过容器编排，自动恢复，自动扩展，监控日志，等高级应用生命周期服务，实现容器化微服务的智能托管；进一步帮助用户降低运维成本。