**微服务与容器架构技术分析和发展应用**

摘 要：微服务作为一种软件开发技术--面向服务架构（SOA）架构风格的变种--提倡将单个应用程序划分为一组小型服务，这些服务相互协调和配合，为用户提供最终价值。每个服务都在自己的独立进程中运行，服务之间使用轻量级的通信机制（通常是基于HTTP的RESTful API）进行通信。每个服务都是围绕着特定的业务建立的，可以独立部署到生产环境、类似生产环境等。此外，应尽量避免使用统一的、集中的服务管理机制，对于特定的服务，应根据具体情况选择合适的语言和工具来构建。作为一种现代化软件架构理念，微服务架构与传统单体结构、集群结构、分布式系统相比，不仅能有效地解决服务器中方案的耦合问题，还能从根本上为各项系统的运转提供方便灵活的服务支持，最终为实现可持续发展目标奠定基础。基于此，笔者主要阐述了设计架构的演变历程，并对微服务特性和容器技术的基本内容进行了全面探析。

关键词：设计架构；演变历程；微服务特性；容器技术

Abstract: Microservices as a software development technology - a variant of the Service Oriented Architecture (SOA) style of architecture - promotes the division of a single application into a set of small services that coordinate and work with each other to provide ultimate value to the user. Each service runs in its own separate process, and the services communicate with each other using a lightweight communication mechanism (usually an HTTP-based RESTful API). Each service is built around a specific business and can be deployed independently to production environments, production-like environments, etc. In addition, uniform, centralised service management mechanisms should be avoided as far as possible, and for specific services, the appropriate language and tools should be chosen to build them on a case-by-case basis. As a modern software architecture concept, microservice architecture, compared with traditional monolithic structure, cluster structure and distributed system, can not only effectively solve the coupling problem of solutions in servers, but also fundamentally provide convenient and flexible service support for the operation of various systems, and ultimately lay the foundation for achieving the goal of sustainable development. Based on this, the author focuses on the evolution of the design architecture and provides a comprehensive exploration of the basic content of microservice features and container technology.

Keywords: design architecture; evolution; microservice features; container technology

## 1软件架构设计的发展历程

大量调研数据分析可知，数据服务接口的开发作业过程历经了分布式系统和集群结构两个阶段，其软件构架从最初的单体结构演变为最后的微服务构架。其中，不同的发展阶段其架构内容也不尽相同 [1]。软件架构的发展经历了从单体架构、垂直架构、SOA架构到微服务架构的过程，一下论述了不同阶段软件的演变并明显表明了软件架构发展历程的不断改进和更新换代。

### 1.1. 单体架构

所谓单体，简单理解就是一个程序里包含了一个系统/产品的所有业务功能，意味着代码被部署并作为单个节点上的单个进程运行，优点呢是小项目开发快 成本低、架构简单、易于测试、易于部署等，但是它的缺点也很容易就能显现出来比如一旦遇到大项目就会模块耦合严重的问题，且不易开发，维护，沟通成本较高。单体架构可能会演变成“大泥球”，又名意大利面条架构。其中包的结构和关系不显式，结构内聚和封装程序低，依赖不遵循规则，很难进行更改和重构。系统是不透明的，粘性的，脆弱的和僵硬的，就像一个大的泥球!单体架构所有的模块都聚合在一起，无法做到单独发版，只能一起发布，且服务存在严重的不稳定性，若服务一旦down机就会导致整个服务无妨使用，因此单体架构目前已经走在了淘汰的淘汰的步伐。

### 1.2. 垂直架构

垂直架构是根据业务把项目垂直切割成多个项目，因此这种架构称之为垂直架构。随着企业业务发展，单体架构转变成了垂直架构，解决了单体架构耦合高且不易开发等问题。系统拆分实现了流量分担，解决了并发问题。还可以针对不同的模块进行优化方便水平扩展，负载均衡，容错率提高，但是缺点也随之显露例如实际应用中服务之间相互调用，如果某个服务的端口或者IP地址发生改变。调用的系统得手动变化。再者搭建集群之后，实现负载均衡比较复杂。比如：内网负载，在迁移得时候会影响调用方的路由，导致线上故障，这对于一些对服务稳定和数据安全要求很高的服务来说无疑是灾难。

### 1.3. 分布式架构

相对于垂直软件架构思路，分布式架构是一种面向服务的架构，基于分布式架构，它将不同业务功能按服务进行拆分，并通过这些服务之间定义良好的接口和协议联系起来，系统间相互独立，互不影响，新的业务迭代时更加高效。

分布式架构把单个服务分成多个层级，按照应用、业务服务、基础业务、基础服务、存储层等按照业务性质分层，每一层要求简单和容易维护，服务以接口为粒度，为开发者屏蔽远程调用底层细节且业务分层以后架构更多清晰，并且每个业务模块职责单一，扩展性更强，也实现了数据隔离，权限回收，数据访问都通过接口，让系统更加稳定，安全且服务责任易确定，每个服务可以确定责任人，这样更容易保证服务质量和稳定，但是即使架构演变到了分布式架构仍然会有一些列问题产生如服务粒度控制复杂，如果没有控制号服务的粒度，服务的模块就会越来越多，就会引发超时，分布式事务瓦内特，另外服务接口数量不宜控制，容易引发接口爆炸，所版本升级兼容困难，尽量不要删除方法、字段，枚举类型的新增字段也可能不兼容。对于复杂业务调用链路长，服务质量不可监控，调用链路变长，下游抖动可能会影响到上游业务，最终形成连锁反应，服务质量不稳定，同时链路的变成使得服务质量的监控变得困难以服务接口建议以业务场景进行单位划分，并对相近的业务做抽象，防止接口爆炸，因此，出现了向微服务应用的再次升级转变。

### 1.4. 微服务架构

微服务架构是分布式架构的延伸，是一种架构风格和架构思想，它主张我们在传统软件应用架构的基础上，将系统业务按照功能拆分成更加细化的服务，每个拆分出来的服务都是一个独立的应用，这些应用向外界提供公共API，可以独立承担对外服务的责任。这样开发出来的软件服务实体就是 "微服务"，而围绕微服务思想构建的一系列架构（包括开发、测试、部署等）可以称为 "微服务架构"。

在微服务架构下，技术的选择是多样化的。每个团队都可以根据自己的服务需求和行业发展现状，自由选择最适合的技术。由于每个微服务相对简单，当技术需要升级时，涉及的风险较小，甚至完全重构一个微服务也是可行和容易的。当架构中的一个组件出现故障时，在具有单一进程的传统架构下，故障很可能会在进程中蔓延，导致整个应用程序不可用。在微服务架构下，故障被隔离在单个服务中。如果设计得好，其他服务可以通过重试和平滑降级等机制在应用层面实现容错。单一服务的应用也可以横向扩展，这种扩展可以通过将整个应用完整地复制到不同的节点来实现。当应用程序的不同组件有不同的扩展要求时，每个微服务都有自己的业务逻辑和适配器，一个微服务一般只完成一个特定的功能，例如，商品服务只管理商品，客户服务只管理客户，如此类推。这使得开发人员可以专注于某一特定功能，而不必过多考虑其他功能，从而提高开发效率。

然而，世界上不存在完美的架构，即使软件架构已经迭代到微服务架构仍然有很多问题需要解决例如虽然微服务实现了服务细粒度的隔离然而这对于测试人员来说无疑是加重了负担，测试工作变得更加困难。在微服务架构中，服务数量众多，每个服务都是独立的业务单元，服务主要通过接口进行交互，如何保证依赖的正常，是测试面临的主要挑战。微服务架构用多个服务实例取代了1个单体应用程序实例，如果每个服务都运行在自己的JVM中，那么有多少个服务实例，就会有多少个实例在运行时的内存开销，在部署和管理时，由许多不同服务类型组成的系统的操作比较复杂，这将要求开发、测试及运维人员有相应的技术水平，因此我们需要微服务与容器结合减轻因为服务部署而出现的压力。

## 2微服务架构的基本概述

### 2.1微服务架构的应用优势

随着城市化和工业化进程的不断加快，企业的规模和数量不断增加，服务器的压力也越来越大。与单体系统、分布式系统和集群系统相比，微服务通过对单体应用的分解降低了系统的复杂性，同时多个子系统的分离也降低了单个服务的开发和维护成本。另一方面，大量的研究和数据分析表明，每个微服务都有相对独立的运行流程和业务处理能力，与传统的变更操作相比，微服务架构由于单个微服务的独立性，避免了对整个应用的编译和部署。此外，微服务架构中各个服务的独立性意味着在发生影响系统稳定性的系统故障时，故障的影响可以控制在单个应用中，避免了对其他服务的影响，大大保证了软件的稳定性、安全性和可靠性。此外，使用不同技术对微服务进行整合和部署，比其他架构模式的风险要小，这就保证了系统的可扩展性，为所有业务的顺利实施提供了坚实的基础 [3]。

### 2.2微服务架构面临的挑战

从以上分析可以看出，微服务除了具有一定的应用优势外，在应用过程中也面临着一些挑战。首先，由于微服务架构是一个分布式系统，其开发难度和复杂程度相对较高，这不仅增加了服务的复杂性，也导致了各种开发风险的存在。其次，微服务架构的数据库分区需要使用大量的数据库，其中一些数据库不支持分布式交易，这就增加了开发的难度。

2.3微服务技术的具体应用

结合Docker的细粒度、松耦合的微服务架构，具有操作简单、资源共享的优势，同时，通过加载不同场景的容器，在角色分类的基础上，在Docker容器中安装一个服务和应用，从而提供一类资源基础，为后期运营打下基础。负载均衡层、综合业务服务层和单一业务服务层的多层方式是项目架构的一个典型案例，它按照业务逻辑规划细化分散到每个Docker中，并按照Rest接口进行整合和链接，最终为实现轻量级、面向应用的虚拟化运行环境奠定了良好的基础，从而为云中微服务的推广奠定了基础。同时，也为促进微服务在云中的大量应用奠定了良好的基础。 [4]。但是仅仅使用Docker作为服务载体还远远不够，对于业务复杂且业务庞大的系统对于运维工作来说无疑是一个巨大的负担，因为我们需要进行服务的编排和系统维护K8S是第一个将“一切以服务为中心，一切围绕服务运转”作为指导思想的创新型产品，它的功能和架构设计自始至终都遵循了这一指导思想，构建在K8S上的系统不仅可以独立运行在物理机、虚拟机集群或者企业私有云上，也可以被托管在公有云中。

微服务架构的核心是将一个巨大的单体应用拆分为很多小的互相连接的微服务，一个微服务背后可能有多个实例副本在支撑。单体应用微服务化以后，服务之间必然会有依赖关系，在发布时，若每个服务都单独启动会非常痛苦，简单地说包括一些登录服务、支付服务，若想一次全部启动，此时必不可少要用到编排的动作，K8s结合Docker完美地解决了调度，负载均衡，集群管理、有状态数据的管理等微服务面临的问题，成为企业微服务容器化的首选解决方案。使用K8S就是在全面拥抱微服务架构。容器只是具有应用受限制的 Linux 进程。限制的示例包括允许进程使用多少 CPU 或内存。Docker 之类的工具允许开发人员将他们的可执行文件与依赖项和附加配置打包在一起。这些包被称为 镜像，并且经常且容易混淆地也被称为容器。微服务并不新鲜。这是一种旧的软件设计模式，由于互联网公司的规模不断扩大，它越来越受欢迎。微服务不一定要容器化。同样，单体应用程序可以是微服务。

但是对于软件架构对于业务简单我们仍不应该回避整体设计。我们也应当按照实际的需求应用软件架构，而不是都采用最新最高级的架构手段，这样反而会增添架构复杂度和企业成本

## 3容器技术的基本概述

### 3.1. 容器技术在微服务应用中的应用

容器技术在微服务应用中的应用，在一定程度上减轻了微服务器架构的压力，为进一步研究创造了良好的条件。大量的研究数据分析表明，Docker是现阶段常用的容器技术之一，不同的容器可以通过内核机制进行链接，容器之间的资源可以有效隔离。近年来，随着我国电子信息技术的不断发展和广泛使用，容器技术的应用不仅在很大程度上降低了微服务架构的成本，而且为不同容器的统一管理创造了良好的条件。

Docker是现阶段最常见的容器技术之一，与其他容器技术相比，一方面，该技术的应用不仅从根本上实现了不同存储方式对不同资源的存储，而且面对当前繁重的业务量和用户量，使计算机更加规范化、科学化，被广泛应用于大型托管服务中[5]。另一方面，Docker容器是标准的镜像结构，可以在云服务平台上构建和分发主机集群，从而为云服务计算技术的发展、云计算平台的建立以及运行环境的优化和自动化维护创造条件，从而不断提高企业的工作质量和效率。Docker Register和Docker Engine是Docker容器技术的两个关键要素，前者主要用于发布应用镜像和构建，后者用于构建容器，两者的结合就是云服务软件的服务理念。两者的结合就是云服务软件的服务理念。在具体操作过程中，为了满足用户大量的应用扩展需求，建立了私有用户集群，不同的用户可以在自己的数据中心建立私有的Docker Register，然后在其他技术的帮助下，整合各种工作，保证在满足企业工作需求的基础上，将微服务架构的功能发挥到极致，为行业的可持续发展奠定坚实基础。从而保证微服务架构的功能在满足企业作业需求的基础上得到最大化的发挥，为行业的可持续发展奠定坚实的基础[6]。

### 3.2. 容器编排技术的扩展

Kubernetes（简称k8s），是一个基于容器技术的分布式架构的新的领先解决方案。它是谷歌秘密已久的秘密武器Borg的开源版本，使用容器技术来管理谷歌内部庞大的集群管理系统。2015年，Kubernetes和Borg论文首次公开，世界首次揭开了它神秘的面纱。从那时起，喜欢它的人对他爱不释手；讨厌它的人，呃，讨厌它的人？

如果用一句话来说明k8s的作用，那就是k8s的目的是实现资源管理的自动化，尤其是在大规模集群中。对于操作人员来说，使用k8s将大大减少工作量，因为大多数任务k8s都会自动完成。对于开发人员来说，可以把更多的精力放在打磨业务逻辑上。总之，k8s提供了强大的自动化能力，大大降低了以后系统运行和维护的难度和成本。

运营和维护的难度大大降低。在一个团队中，只需要少数成员进行项目的部署和运营，其他成员可以专门打磨业务逻辑。k8s可以完全拥抱微服务。微服务的核心是将一个庞大的系统分解成许多相互关联的小微服务，一个微服务可以由多个实例副本来支持，副本的数量可以随着系统的负载而调整，k8s几乎天然地支持微服务。系统可以随时整体搬迁到公有云上。目前，国内几个主流云（华为云、阿里云、腾讯云）都相继宣布支持k8s。同时，由于k8s屏蔽了底层网络的细节，基于虚拟IP的设计思想使得k8s独立于底层硬件拓扑结构，允许系统在不改变运行时配置文件的情况下进行迁移。 k8s具有超强的横向扩展能力。

## 4 结语

随着经济全球化进程的加快，软件架构从最初的单体结构发展到现在的微服务架构，不仅有效解决了微服务架构中面临的负载问题，还能更有效地支持微服务。Docker容器和Kubernates的加入不仅为研究人员开拓了新的思路，最重要的是 Docker容器和Kubernates的加入不仅为研究人员开拓了新的思路，而且从根本上加快了微服务的开发，有效解决了用户资源隔离问题和复杂环境问题，最终为实现气象业务系统可持续发展的目标奠定了坚实的基础。

## 参考文献

[1]杨鸥, 张羿涛, 耿贞伟. 微服务架构在容器云中的应用实践 [J]. 电脑与电信 ,2017,15(7):79-81.

[2]张晶晶 , 黄小锋 , 李春阳 . 微服务框架的设计与实现 [J].

计算机系统应用 ,2017,15(6):259-262.

[3]王纪军, 张斌, 顾永生. 云环境中 Web 应用的微服务架构评估 [J]. 计算机系统应用 ,2017,22(5):109-115.

[4]王健, 韩志国, 李冬睿. 从单一模式系统架构往微服务架构迁移转化技术研究 [J]. 科教导刊 ,2016,12(22):111-113.

[5]夏鹏, 王文文, 周瑞康. 基于开源框架及容器技术的微服务架构 [J]. 电子技术与软件工程 ,2018,15(20):146-148.

[6]刘辉军, 刘培锋, 邱钰锋. 基于开源框架及容器技术的微服务架构研究 [J]. 电力信息与通信技术 ,2018,16(6):90-94.