基于容器的微服务架构的浅析

◆ 陈春霞

目前关于微服务的讨论日趋火爆，争议不断，褒贬不一。在系统设计与开发中采用微服务方式实现软件系统的松耦合、跨部门开发；通过容器编排、自动修复、自动扩展、监控日志等应用服务实现容器化微服务的智能托管，帮助用户降低运维成本和难度。同时，使用不同的语言和框架来开发各个系统模块，独立进程众多，增加系统集成与测试的难度。尽管一些公司采用了微服务架构，并且取得了良好的效果；但更多公司还是在观望。微服务架构与容器服务正在颠覆传统的系统设计与开发方式。

一、应用设计架构的演变

回顾下 20 年内基于 Web 开发的历史，我们知悉微服务架构为什么可以在 Web 开发领域如此的盛行，同时也顺便了解这个架构可以解决的问题。

在 Web 应用程序开发的早期，应用程序通常使用 Common Gateway Interface（CGI）建立，这个接口为网络服务器提供了处理浏览器发来的 HTTP 请求时执行脚本的能力。后来就出现了比较传统的经典三层架构 Web 方式：表现层（UI）、业务逻辑层（BLL）、数据访问层（DAL）。时至今日，这种多层次的应用构架仍然许多应用的首选：层次划分清晰，层与层之间有比较清晰的接口，每层之间非常清晰。通常情况下，客户端不直接与数据库进行交互，而是通过 COM/DCOM 通讯与中间层建立连接，再经由中间层与数据库进行交互。

在项目初期，多层架构能够使程序员更容易开发、测试和部署。但随着应用程序逐渐增长，业务复杂度会变的越来越高。这种情况下三层构架并不是特别适合业务的继续发展，可维护性和可扩展性都会变差，从而最终影响应用系统的使用。而微服务，就是解决这些问题的一个很好的方案。

二、微服务架构

在微服务架构中，软件开发的方式将有很大变化。应用将由许多微服务组成的，开发软件不再是库而是开发微服务。每个微服务都对应了一个独立的业务功能，只定义了该功能必须的一些操作。整个软件解耦为多个功能相对独立的较小片段，这些功能片段对系统中其它部分的依赖加以限制，能够应对外界的故障。这种特性能够让基于微服务架构的应用在应对 bug 或是对新特性的请求时，能够快速地进行变更。而容器进一步对这种解耦性进行了扩展，并能够将软件从底层的硬件中分离出来，微服务的目的是充分地分解应用程序以促进敏捷开发和部署。

微服务架构通常用于服务之间的交互。微服务与 ESB 有着非常显著的差别，它不包含用于定义服务间接口的同级别控制和规范化数据建模，而 ESB 包含了大量用于模式验证、消息路由、消息翻译和业务规则。因此，对比传统的面向服务架构，微服务架构往往更为简单；通过使用微服务，开发将非常快速，服务的衍变也只需匹配业务的需求。

微服务架构的另一个核心优势就是服务可以基于资源的需求进行独立扩展。取代运行包含大量 CPU 和内存的大服务器，微服务可以被部署在更小的主机上，这些主机只需要满足其部署服务的需求。同时，开发者可以根据业务的需求选择不同的开发语言实现。

三、微服务的特性

1. 复杂度可控：每一个微服务只具有单一功能，它将一个庞大的整体应用分解成一组服务。在整体的功能没有改变的同时，应用程序已经被分解成可管理的模块或服务。通过定义良好的接口清晰表述服务边界，每个服务有以 RPC 或者消息驱动 API 形式定义清楚的界限。在将应用分解的同时，规避了复杂度无止境的积累。微服务架构模式加强了一定程度的模块化，由于体积小、复杂度低，易于保持高可维护性和开发效率。

2. 独立部署：由于微服务具备独立的运行进程，所以每个微服务也可以独立部署。在传统架构中要对应用程序中某个小部分进行变更，就必须对整体架构进行重新构建，并且重新进行部署。而当某个微服务发生变更时无需编译、部署整个应用。由微服务组成的应用相当于具备一系列可并行的发布流程，使得发布更加高效，同时降低对系统环境所造成的风险，最终缩短应用交付周期。

3. 技术选型灵活：微服务架构下，技术选型是去中心化的。可以根据自身服务的需求和行业发展的现状，自由选择最适合的技术栈。由于每个微服务相对简单，当需要对技术栈进行升级时所面临的风险较低，甚至完全重构一个微服务也是可行的。

4. 容错：当某一组应用功能发生故障时，在单一进程的传统架构下，故障很有可能在进程内扩散，形成整个应用的失败。在微服务架构下，故障会被隔离在单个服务中，通过良好的设计，其他服务可平稳退化等机制实现应用层面的容错。

5. 扩展：微服务架构模式使得每一个服务都可以被独立扩展。单块架构应用也可以实现横向扩展，就是将整个应用完整的复制到不同的节点。当应用的不同组件在扩展需求上存在差异时，微服务架构便体现出其灵活性，因为。这些服务通过在不同的基础设施之间实现扩展，能够有效地降低风险。

微服务应用为分布式系统带来的复杂性。开发者需要选择或者实现基于消息或 RPC 的进程间通信机制。因为整体应

用程序中模块间的调用是通过语言层面的方法/程序调用实现的，必须编写处理部分失败的代码，使得整体应用程序更复杂。

微服务的另外一个挑战是分割的数据库架构。在一个数据库中更新处理多个业务事务很容易实现，然后在基于微服务的应用中，你需要更新多个属于不同服务的数据库。分布式事务通常不是最好的选择，而且目前许多高扩展性的NoSQL数据库和消息代理就不支持。

四、容器的虚拟化技术

容器技术的使用可以很大程度上缓解微服务架构所带来的问题。容器技术使用了内核接口，它允许不同容器共享相同的内核，同时容器之间还进行了完全的隔离。Docker是基于容器的微服务架构比较成功的案例。

2015年是容器技术快速发展的一年，Docker和Kubernetes等容器技术将分别成为容器化和编制的事实标准，并且引领新一代云计算解决方案。Linux容器将服务导向架构(SOA)的一些功能，而规模更容易管理。标准化的应用将整合在一起，以“微服务”的形式交付，并将实现生产化。通过利用Kubernetes等技术将这些应用跨主机编制在一起，将创建新一类的应用和用例，实现弹性、跨地理界限扩展的架构，从而轻松应对不断变化的业务系统。

Docker容器是以资源分割和调度的基本单位，并封装整个软件运行时环境。它是一个跨平台、可移植并且简单易用的容器解决方案。可在容器内部快速自动化地部署应用，并通过操作系统内核技术为容器提供资源隔离与安全保障。

以Docker为代表的容器技术，标准化、高可用的镜像构建，存储服务，大规模、可伸缩的容器托管服务，及自有主机集群混合云服务；它实现应用的镜像构建、发布、持续集成/交付、容器部署、运维管理的新一代云计算平台。优化开发运维环节，提高业务效率，降低IT成本，实现持续创新。

Docker执行环境的接口实现了标准化，为容器镜像提供了一个类的资源库，让容器的共享和发布非常简单，容器隔离使得不同语言开发的微服务代码更加容易部署。Docker有两个特点快速和可移植性。

1. 快速

普通的虚拟机在每次开机时都需要启动一个完整的新操作系统实例，而Docker的容器能够通过内核共享的方式，共享一套托管操作系统。这意味着，Docker容器的启动和停止不需要几分钟，只要几百毫秒就足够了。

更快的速度就意味着，使用Docker容器创建的软件系统比起使用基于虚拟机的解决方案能够实现更高级别的敏捷性，即使将那些基于虚拟机的解决方案通过基于微服务的架构进行组织也是一样。此外，“容器化”的应用比起虚拟机和裸机的性能更好，在2014年IBM发布的一份研究报告中表明：“在几乎所有情况下，容器都能表现出与虚拟机相等、或者是更好的性能。”

2. 可移植性

在基于虚拟机的解决方案中，应用的可移植性通常来说会受到云提供商所支持的地区的限制，如果是在自托管的环境中运行企业软件，那么可移植性就限制在数据中心内。原因在于，不同的云提供商通常会提供不同格式的虚拟机。如果使用Packer这样的工具，那么在不同的云服务中使用运行相同的虚拟机镜像文件也是可以的，但需要进行许多额外的工作。虽然可行，但它也将用户限制在一个单一的平台中。

五、基于容器的微服务的应用

容器技术的使用可以很大程度上缓解微服务架构所带来的问题。Linux容器技术允许不同容器共享相同的内核，同时容器之间还进行了完全的隔离。Docker执行环境模块使接口标准化，为容器镜像提供了一个类的资源库，让容器的共享和发布非常简单，也正是这种相同主机上的容器隔离简易了不同语言开发的微服务代码部署。

虽然微服务架构带来了诸多优势，但构建、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。同样，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. 标准化。利用镜像技术构建标准开发环境，通过封装着完整环境和应用的镜像进行迁移，Docker消除了线上线下的环境差异，保证了应用环境一致性和标准化。由此，测试和运维人员可以直接部署软件镜像来进行测试和发布，大大简化了持续集成、测试和发布的过程。

2. 跨云平台支持。Docker的适配性，使越来越多的云平台都支持Docker，同时也让应用多平台混合部署成为可能。目前支持Docker云平台包括亚马逊云平台（AWS）、Google云平台（GCP）、微软云平台（Azure）等，还包括如Ansible、Chef、Puppet等配置管理工具。

3. 高资源利用率与隔离。Docker容器与底层共享操作系统，系统负载更低，性能更加优良，更充分地利用系统资源，在同等条件下可以运行更多的应用实例。同时，容器具有资源隔离与限制能力，可以精确地对应用分配CPU、内存等资源，保证了应用间不会相互影响。

4. 容器跨平台性与镜像。Docker在原有Linux容器的基础上进行大胆革新，为容器设定了一整套标准化的配置方法，将应用及其依赖的运行环境打包成镜像，真正实现了“构建一次，到处运行”的理念，大大提高了容器的跨平台性。

5. 应用镜像仓库。构建了一个镜像仓库，开发者可以自由下载微服务组件，开发将更加快捷高效。

2015年微服务架构和基于容器的虚拟化技术得到迅猛的发展，将微服务与容器的结合视为一种新的设计理念，应用程序利用微服务架构和容器的虚拟化技术的优点得到充分体现，它能够将软件从底层的硬件中分离出来。以运维为主导的容器使微服务架构大大简化，应用程序能够快捷创建、更加完善、易于维护，微服务架构与容器服务带来的系统设计与开发革命。

（作者单位：河南省信息中心）