# 组合Docker, Kubernetes, MongoDB来构建你的微服务

微服务这个词已经出现一段时间了，而且他的概念已经被广大开发者所接受，这篇文章并不是一个解释什么是微服务，而是在进行微服务部署时我们应当选择的技术搭配，应用层、数据层以及运维部署等都是需要考虑的范围。在很多项目开发过程中，我们也许并没有真正理解到微服务的核心概念，在我经历过的大多项目中，在项目初期开发团队热烈的讨论微服务构架，技术选型，大量的框架比较争论在技术部热火朝天的进行着，但是当项目做到快交付的时候，为了加快项目进度，很多最佳实践被抛弃，有些人甚至抛弃了设计阶段服务之间的耦合关系。最终导致的结果是我们搭建了一套微服务构架产品，但是产品内部却存在着几个大型单体应用服务，而这些单体应用服务却被大家当成微服务来处理，严重违背了微服务的理念以及消耗大量的开发资源来对系统进行维护和升级。

那么正确的微服务项目过程应当如何呢？对于一个一般产品周期来说，其必经下面几个阶段：

图1：产品开发周期

* MVP  
  在项目初期大家会对业务进行最小化需求管理，整理出需求以后需要开发快速发布一款MVP产品。在这个阶段时间是最宝贵的资源，尽量快速的发布是产品要求的核心要素。很多投资方会看重你的MVP市场反应，项目组也会根据用户对MVP的反馈来进行下一阶段的部署和策划。
* 迭代更新  
  根据用户反馈，产品会逐渐进入迭代更新阶段，开发人员会按照新的需求进行迭代开发，逐步完善产品功能并持续发布新版本。
* 维护  
  在产品用户数到达一定规模时，开发运维往往占据主要的开发资源，保证产品稳定运行，不断线，问题出现后能够即使修复是开发运维需要解决的首要问题。
* 扩展  
  当产品性能、容量达到一定程度时，我们需要对产品进行扩展，一种方法是增加服务器性能，另一种是增加服务器个数，前者称为垂直扩展后者称为水平扩展，在云平台流行的今天，水平扩展是大家比较喜欢尝试的方案。但是，由于服务的内容没有发生变化，只是在数量上得到了增加，这样做并不能实质上解决产品容量的限制。因此，我们需要引入微服务构架来重构我们的产品。开发团队会对产品服务进行切分，找到服务边界并把相互独立的服务分别部署到不同的服务器上。

以上就是一款产品从零到微服务的发展历程，当然这只是一个概要式说明，不同的产品会有或多或少的区别。在这个里程表中，前三个阶段都可以算是单体应用阶段，就是所有业务全部有一个独立的应用程序所承担。到了微服务阶段我们就需要用到一些工具、框架和方法来适配业务的变化，下面我们从容器、Orchestraion，数据库三个方面来看一下如何选择正确的技术栈来为微服务做支撑。

# 容器（Container）

虚拟操作系统在过去十几年非常流行，它允许用户在一台系统环境上运行多种不同的虚拟操作系统，每个系统都相互独立互不干扰。

容器运行在操作系统之上，并且共享系统内核、库文件和进程文件。一个容器的大小可以小到只有几兆，这使得容器非常轻量，启动一个容器只需要几秒钟的时间。

## 容器 vs 虚拟机

容器听起来很想虚拟机，的确是这样，他们之间有一定的相似度，但是也有很大的区别。正如下图所示，两者在最下面两层都是一样的，采用同样的硬件和操作系统。但是区别发生在第三层，对于虚拟机来讲他需要Hypervisor的支持来虚拟不同的硬件环境和操作系统，但是对于容器来说，它只需要一个Docker Engine就可以启动容器实例，相比Hypervisor，Docker Engine更轻量，消耗更少的硬件资源。在其之上，虚拟机就开始了虚拟操作系统的过程这也是和容器区别最大的地方。当你是用一台虚拟机时，Guest OS被称为目标操作系统，比如在你的Windows操作系统上安装了VMWare Workstation，可以通过VMWare安装其他Linux或者Windows系统，这个在VMWare之上安装的操作系统被称为Guest OS。再往上看，虚拟机把一个Guest OS当作一个全新的操作系统，你可以想象在一台全新的操作系统上安装各种应用程序、库、服务等。可以说虚拟机不光是虚拟了全部操作系统，而且还虚拟了操作系统所需要的虚拟硬件。反观Container，他并不需要虚拟一个操作系统，他会共享你的主机硬件和资源，这也是为什么容器更轻量的原因，它只需要在Docker Engine之上加载一些库、应用，并在其上搭建各种服务。多个服务可以共享一台应用，而每个服务之上就是最终面向用户的容器接口。

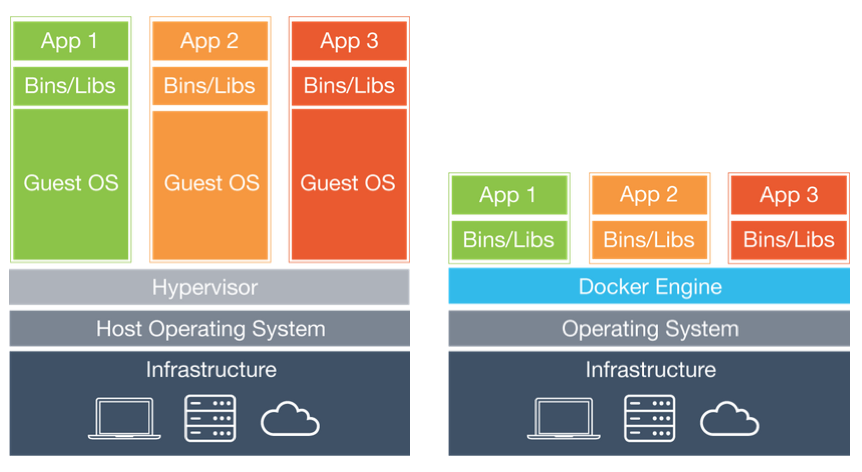


图2：虚拟机和容器的区别

对于容器来说他有很多虚拟机不具备的优势：

* DevOps & 持续发布  
  在开发运维方面，容器可以是维护工作变得越来越简单方便。
* 可重用的应用环境  
  一套应用环境可以方便的克隆给开发、测试、产品环境，而不需要而外的安装配置操作。
* 完善的测试环境  
  很多时候产品的bug只出现在产品环境，开发人员很难在测试环境上重现产品环境的bug。其主要原因是由于环境的不完全匹配。而容器可以保证两套环境是一份完全拷贝，没有任何区别，这样方便我们通过测试环境模拟产品上的缺陷。
* 可扩展性  
  删除、增加容器的实例比重新创建一套虚拟环境简单很多。
* 分离性  
  各个容器之间相互独立，可以单独运行在任何主机上。
* 性能  
  相比虚拟机，容器的性能消耗要小很多，特别是内存方面的消耗。
* 高可用性  
  多个容器可以方便的搭建高可用的产品环境。

# Orchestration

在管理Docker容器方面，很多人都是用Docker内置的Compose，Swarm，它确实方便，不过比较适合小型规模的容器部署，在上百台容器搭建上有些力不从心。Orchestration是指用来管理维护容器的框架。它可以用来启动容器实例，当容器宕机后它可以恢复容器的运行，并且可以用来将若干个容器实例连接在一起构建一个专有网络来为他们提供通讯，在需要扩展容器时Orchestration可以增加新的容器实例到你的系统中。在众多Orchestration产品中，Kubernetes是其中发展比较快、功能较全的框架。Kubernetes对Docker容器提供了很好的支持，如果我们把MongoDB用Docker来搭建，那么Kubernetes可以用来提供管理、监控这些MongoDB数据库，它可以确保你的MongoDB容器在失效时立刻重新创建一个容器来提供服务。下面章节中我会通过实例为大家解释Kubernetes和MongoDB结合如何解决业务上的需求。

# 迁移到微服务之前需要考虑的问题

众所周知，微服务提供了很多有益的方面，但是并不是适用于所有的业务场景。在迁移到微服务构架之前，我们应当考虑下面列出的问题。

* 服务监控  
  微服务构架最有挑战的任务可以说是服务监控，当有成百上千服务部署在产品环境时，监控这些服务的运行状态会变得极其复杂。我们的运维团队在维护系统时不光要监控服务的运行指标，还要分析大量的服务日志，查看网络中可能存在的分区。传统的查看CPU、内存、磁盘、网络等方法需要扩展到一个监控聚合层面，我们要支持自动监控系统运行一段时间所产生的所有与系统相关的数据，并且能够在这些数据中进行相关度比较，从而达到预测风险，减少运维成本的目的。
* 较高的开发技能  
  对比传统的单体应用程序，微服务应用需要分布部署在多台主机上，分布式的开发和调试需要较高的开发水平。开发人员不仅仅是将开发完的产品交给运维团队，更重要的是需要理解运维的方方面面，服务之间的各种耦合，网络消耗等因素都要在开发设计阶段做好充分的考虑。另外，测试团队也需要使用更多的测试技巧来对分布式环境下的产品进行各种级别的测试。所有这些都需要有较高水平的工程师来担任，团队之间的协作也需要跟紧密的配合。
* 服务边界  
  在设计微服务构架前，需要充分了解服务的业务逻辑，找到业务的边界对于划分微服务来说至关重要。在不恰当的微服务设计下，开始运行还算正常，但是当需求一点一点加上来以后，可能会出现分布式的巨大单体应用程序，这会令构架更难维护和开发，并且可能会导致部分服务重新设计再开发再投入等恶性循环。

# 为什么说MongoDB适合微服务构架

除了应用程序层面的构架之外，数据层面的选择也很关键，如果企业开发团队想对自身的系统进行微服务部署，需要一些必要的技术构架支持，比如灵活的数据模型，系统可扩展性，自动化部署以及容错。下面我们来看一下MongoDB如何支持微服务构架。

* 灵活的数据模型  
  MongoDB灵活的数据模型结构为微服务持续集成提供了必要条件。当添加了新的服务功能而改变了数据库表结构，MongoDB不需要更新现有的记录，而这个数据更新在关系型数据库中可能需要耗费很长时间才能完成。开发人员可以快速的上线部署新的服务从而适应市场的快速变化需求。
* 容错性  
  由于微服务的本质是分布式部署，各种服务分布在不同的主机或者网络环境上，出错的可能性会大大提高，导致错误的原因可能是网络传输、服务发现、数据匹配等。所以在设计微服务构架时，需要考虑容错机制。而MongoDB中的Replica Set提供了强大的容错机制，并且支持高可用以及灾难恢复。在Replica Set中，每一个成员都有一份数据的拷贝，任何一台机器出现故障都可以由另外一个成员接替其工作。此外，每个成员还可以承担不同的角色，例如，可以指定一台服务器专门负责生成报表，其他成员负责日常服务请求，这样指责更加明确。
* 自动化监控  
  在服务数量比较小的阶段，监控系统服务运行状况比较简单，但是随着服务数量的不断增加，手动监控系统状态变得比较笨重。一种自动化的监控方法需要设计进我们的开发运维中。选择一个正确的监控平台是为开发运维提供效率的保障。幸运的是MongoDB提供了MongoDB Ops Manager功能，可以可视化的监控系统性能的各项指标。
* 扩展性  
  微服务的一大特点就是方便的支持服务扩展，MongoDB在这方面也做得很到位，其中Shard Cluster机制提供了MongoDB集群水平扩展的基础构架。当数据量非常大的情况下，我们可以根据自定义的条件将数据进行切分，切分成若干chunk，然后MongoDB会自动将这些chunk均匀分布到不同的Shard集群中，关于MongoDB扩展方面的内容可以参考另一片文章“MongoDB的水平扩展，你做对了吗”。

# 微服务案例分享

在进行微服务搭建的过程中，大家会研究大量的技术栈和最佳实践，网上会有很多文章介绍各种技术组合，但是具体到应用场景的部署方法以及具体问题的解决思路并没有太多可以参考的地方，这里我给大家分析两个成功案例，他们都采用了MongoDB，Kubernetes，Docker作为部署环境。

## fuboTV

fuboTV是一家在纽约的网络视频公司，主要为北美市场提供足球转播的视频服务，从一开始fuboTV就采用MongoDB作为他们的数据存储服务器，随着业务的增长fuboTV已经把他的MongoDB服务器迁移到Docker上，并采用Kubernetes作为管理框架，部署在Google云平台上，今后还会考虑部署在不同云厂商平台。采用MongoDB，Docker和Kubernetes的一大好处是部署时间可以做到非常短，当需要更新服务、升级版本、解决紧急问题时，部署的时间在几秒内就可以完成，这对于一个互联网用户遍布全球的应用来说是一个不可或缺的优势。

fuboTV将MongoDB Replica Set按照Google云的地理位置分布在多个region上，每个实例采用32 CPUs, 28 GB内存和SSDs磁盘。每一个Replica Set包括四个成员，三个成员用来处理日常业务请求，另一个配置成隐藏成员，用来做数据备份、灾难恢复。

在对MongoDB和Kubernetes进行集成时，有一些需要解决的技术问题，我们来看一下fuboTV是如何做到他们的完美集成的。

* MongoDB服务器是有状态的，当一个容器失效，在Replica Set中的其他成员会接替他的工作并对其进行灾难恢复，但是这个恢复过程需要一些时间，而Kubernetes中的存储卷可以映射到MongoDB的数据目录中，这样在容器宕机后磁盘内容仍然可以在网络中访问到。
* MongoDB Replica Set中的各个成员应当能够实时和各自进行通讯，他们之间会进行心跳检测。所以，每一个容器都需要一个固定的IP地址以便于其他成员对其进行访问，而MongoDB容器部署在Kubernetes Pod上，当容器重新启动以后通常会获得一个不同的IP地址，这样就会造成这个重新启动的容器无法加入到Replica Set中。fuboTV为了解决这个问题将Kubernetes DNS服务连接到每一个MongoDB节点上，这样所有容器都可以通过hostname来互相访问，不会收到IP地址变化而引起的丢失成员问题。
* 当MongoDB Replica Set初始化的时候，每个成员节点需要加入到Replica Set中，fuboTV采用CircleCI部署可持续交付环境，CircleCI可以集成Docker和Kubernetes，从而执行Replica Set需要的初始化步骤。

在处理高并发访问情况下，MongoDB也提供了相应的支持。在一场足球比赛开始前十几分钟的时间内，用户访问fuboTV的频率是平时的十倍以上，为了处理这种高并发请求，fuboTV采用“[Nearest Read Preference](https://docs.mongodb.org/manual/core/read-preference/)” （最近成员） 的Replica Set成员，“Read Preference”是MongoDB Replica Set中读取数据的一个配置选项，当配置成“最近”时应用程序根据用户的地理位置判断那个服务器为其提供服务。

下面我们开一下OTTO如何通过MongoDB来将部署时间从2小时缩短到15分钟的。

## OTTO

OTTO是德国一家时尚品牌零售商，每天需要处理多大两百万次用户访问。OTTO团队有多个业务线组成，每个业务线相对独立，但是又要求快速交付产品。之前OTTO的构架支持2小时发布新产品，但是在使用MongoDB之后，发布部署时间缩短到了15分钟。

对于零售商来说，选择正确的数据库是一件非常重要的决策，OTTO涵盖了多大5000多种品牌的商品，从牛仔裤到沙发再到音响，他们需要存储种类繁多的产品数据，而每一种产品又有不同的属性，例如：人们在选购沙发时关注的是尺寸、重量、面料，而音响的选择主要是声音品质等，此外产品的价格又有着非常灵活的变化范围，随着节假日、新产品发布、促销等不同方式的销售模式，产品价格要随时保持更新的状态。下面是OTTO业务模式需要解决的问题：

* 为了保持产品的竞争力，OTTO需要支持“全渠道零售”营销模式，在互联网和电子商务的当今时代，全渠道零售能通过各种渠道与顾客互动，包括网站、实体店、服务终端、直邮和目录、呼叫中心、社交媒体、移动设备、游戏机、电视、上门服务等。传统商家除非采用全新视角，把各种迥然不同的渠道整合成“全渠道”的一体化无缝式体验，否则就很可能被时代淘汰。
* 发布实时的产品信息和内容更新。
* 为不同的用户定制产品信息内容，根据用户喜好推送相关产品消息。
* 利用社交网络为用户提供分享功能从而扩展产品的影响力。

为了支持以上这些需求，企业需要在数据层面进行创新性选择，不幸的是很多应用还是沿用十几年前的技术（关系型数据库）来存储零售数据，这种情况下数据的多样化很难在关系型数据库中存储，数据的扩展迁移需要耗费大量的时间和人力来完成，有时一个数据库表字段的增加需要几周的时间来更新现有记录，这对与业务逻辑复杂需要快速响应时间的产品来说也许是个噩梦。正是因为这些原因，OTTO选择MongoDB作为自己的存储媒介。MongoDB自身的数据模型设计允许松散的数据集成在一张数据库表（Collection）中，并且不对表进行范式检查，极大方便了数据的扩展并提高了查询数据的性能。关于MongoDB查询性能方面的内容建议大家参考这篇文章： <https://mp.weixin.qq.com/s/xQniuEaUI9g3ICHsI66NRw>。

# 小结

MongoDB提供了多种支持弹性配置，在使用的时候要选择尽量符合你应用场景的配置选项。Docker是一项革新的技术，要选择合适的工具开控制开发部署流程。Kubernetes比较年轻，但是很快达到了产品要求的成熟度，选择好支持他的云平台框架至关重要。经历过交付项目的团队都体会过错误的技术选型所带来的痛苦，为了满足我们的业务增长需要，正确的选型对团队和产品都是至关重要的环节。

# 参考文献

Read Preference: <https://docs.mongodb.com/manual/core/read-preference/?_ga=2.258785674.1422832162.1509151848-573910377.1492591641>

使用MongoDB自带的Explains功能: <https://mp.weixin.qq.com/s/xQniuEaUI9g3ICHsI66NRw>

Kubernetes: <https://kubernetes.io/docs/home/>

# 作者简介

赵翼，毕业于北京理工大学，目前就职于SouthbankSoftware，从事NoSQL，MongoDB方面的开发工作。曾在GE，ThoughtWorks，元气兔担任项目开发，技术总监等职位，接触过的项目种类繁多，有Web，Mobile，医疗器械，社交网络，大数据存储等。