

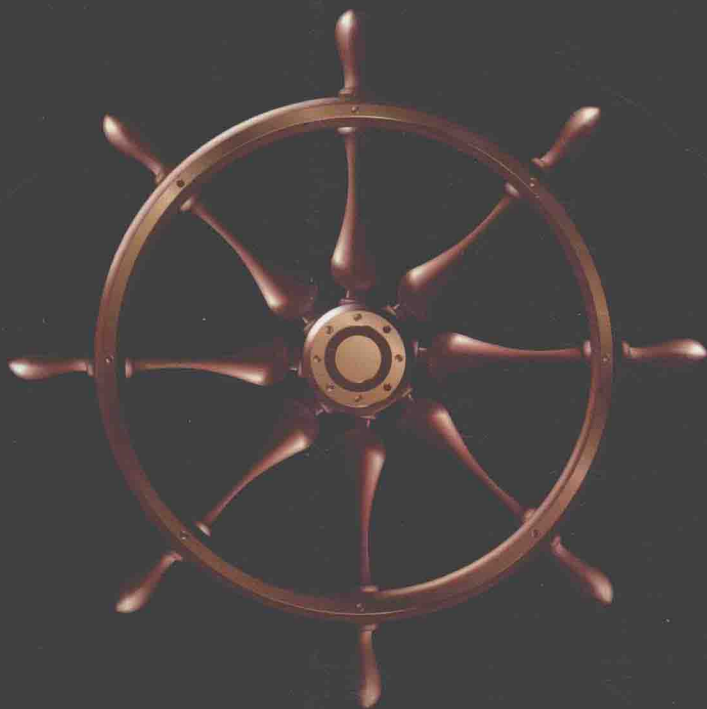
Kubernetes

权威指南

第2版

龚正 吴治辉 王伟
崔秀龙 闫健勇 等编著

从Docker到Kubernetes
实践全接触



下一代分布式架构的王者

Kubernetes: The Definitive Guide



中国工信出版集团



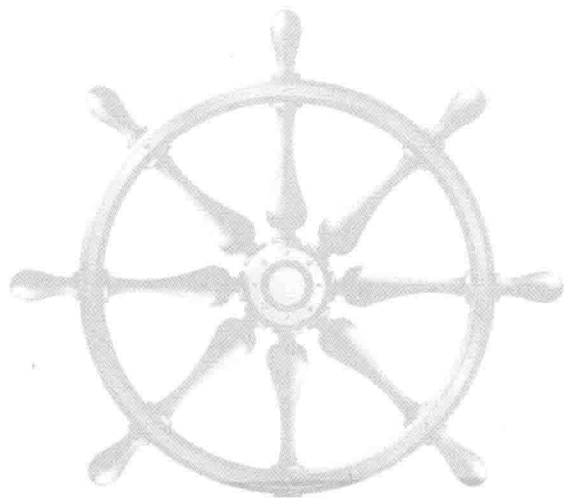
电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Kubernetes

权威指南 第2版

龚正 吴治辉 王伟 / 等编著
崔秀龙 闫健勇

从Docker到Kubernetes
实践全接触



电子工业出版社

Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

Kubernetes 是由谷歌开源的 Docker 容器集群管理系统，为容器化的应用提供了资源调度、部署运行、服务发现、扩容及缩容等一整套功能。本书从一个开发者的角度去理解、分析和解决问题，囊括了 Kubernetes 入门、核心原理、实践指南、开发指导、高级案例、运维指南及源码分析等方面的内容，图文并茂、内容丰富、由浅入深、讲解全面；并围绕着生产环境中可能出现的问题，给出了大量的典型案例，比如安全问题、网络方案的选择、高可用性方案及 Trouble Shooting 技巧等，有很强的可借鉴性。

无论是对于软件工程师、测试工程师、运维工程师、软件架构师、技术经理，还是对于资深 IT 人士来说，本书都极具参考价值。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

图书在版编目（CIP）数据

Kubernetes 权威指南：从 Docker 到 Kubernetes 实践全接触 / 龚正等编著. —2 版. —北京：电子工业出版社，2016.10

ISBN 978-7-121-29941-4

I. ①K… II. ①龚… III. ①Linux 操作系统—程序设计—指南 IV. ①TP316.85-62

中国版本图书馆 CIP 数据核字（2016）第 225554 号

策划编辑：张国霞

责任编辑：徐津平

印 刷：北京天宇星印刷厂

装 订：北京天宇星印刷厂

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×980 1/16 印张：32.5 字数：730 千字

版 次：2016 年 10 月第 1 版

印 次：2016 年 10 月第 1 次印刷

印 数：4000 册 定价：99.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zltz@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819 faq@phei.com.cn。

第 1 版推荐序

经过作者们多年的实践经验积累及近一年的精心准备，本书终于与我们大家见面了。我有幸作为首批读者，提前见证和学习了在云时代引领业界技术方向的 Kubernetes 和 Docker 的最新动态。

从内容上讲，本书从一个开发者的角度去理解、分析和解决问题：从基础入门到架构原理，从运行机制到开发源码，再从系统运维到应用实践，讲解全面。本书图文并茂，内容丰富，由浅入深，对基本原理阐述清晰，对程序源码分析透彻，对实践经验体会深刻。

我认为本书值得推荐的原因有以下几点。

首先，作者的所有观点和经验，均是在多年建设、维护大型应用系统的过程中积累形成的。例如，读者通过学习书中的 Kubernetes 运维指南和高级应用实践案例章节的内容，不仅可以直接提高开发技能，还可以解决在实践过程中经常遇到的各种关键问题。书中的这些内容具有很高的借鉴和推广意义。

其次，通过大量的实例操作和详尽的源码解析，本书可以帮助读者进一步深刻理解 Kubernetes 的各种概念。例如书中“Java 访问 Kubernetes API”的几种方法，读者参照其中的案例，只要稍做修改，再结合实际的应用需求，就可以用于正在开发的项目中，达到事半功倍的效果，有利于有一定 Java 基础的专业人士快速学习 Kubernetes 的各种细节和实践操作。

再次，为了让初学者快速入门，本书配备了即时在线交流工具和专业后台技术支持团队。如果你在开发和应用的过程中遇到各类相关问题，均可直接联系该团队的开发支持专家。

最后，我们可以看到，容器化技术已经成为计算模型演化的一个开端，Kubernetes 作为谷歌开源的 Docker 容器集群管理技术，在这场新的技术革命中扮演着重要的角色。Kubernetes 正在被众多知名企业所采用，例如 RedHat、VMware、CoreOS 及腾讯等，因此，Kubernetes 站在了容器新技术变革的浪潮之巅，将具有不可预估的发展前景和商业价值。

如果你是初级程序员，那么你有必要好好学习本书；如果你正在 IT 领域进行高级进阶修炼，那你也有必要阅读本书。无论是架构师、开发者、运维人员，还是对容器技术比较好奇的读者，本书都是一本不可多得的带你从入门向高级进阶的精品书，值得大家选择！

初瑞

中国移动业务支撑中心高级经理



自序

我不知道你是如何获得这本书的，可能是在百度头条、网络广告、朋友圈中听说本书后购买的，也可能是某一天逛书店时，这本书恰好神奇地翻落书架，出现在你面前，让你想起一千多年前那个意外得到《太公兵法》的传奇少年，你觉得这是冥冥之中上天的恩赐，于是果断带走。不管怎样，我相信多年以后，这本书仍然值得你回忆。

Kubernetes 这个名字起源于古希腊，是舵手的意思，所以它的 Logo 既像一张渔网，又像一个罗盘。谷歌采用这个名字的一层深意就是：既然 Docker 把自己定位为驮着集装箱在大海上自在遨游的鲸鱼，那么谷歌就要以 Kubernetes 掌舵大航海时代的话语权，“捕获”和“指引”这条鲸鱼按照“主人”设定的路线巡游，确保谷歌倾力打造的新一代容器世界的宏伟蓝图顺利实现。

虽然 Kubernetes 自诞生至今才 1 年多，其第一个正式版本 Kubernetes 1.0 于 2015 年 7 月才发布，完全是个新生事物，但其影响力巨大，已经吸引了包括 IBM、惠普、微软、红帽、Intel、VMware、CoreOS、Docker、Mesosphere、Mirantis 等在内的众多业界巨头纷纷加入。红帽这个软件虚拟化领域的领导者之一，在容器技术方面已经完全“跟从”谷歌了，不仅把自家的第三代 OpenShift 产品的架构底层换成了 Docker+Kubernetes，还直接在其新一代容器操作系统 Atomic 内原生集成了 Kubernetes。

Kubernetes 是第一个将“一切以服务（Service）为中心，一切围绕服务运转”作为指导思想创新型产品，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 Kubernetes 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。Kubernetes 方案的另一个亮点是自动化，在 Kubernetes 的解决方案中，一个服务可以自我扩展、自我诊断，并且容易升级，在收到服务扩容的请求后，Kubernetes 会触发调度流程，最终在选定的目标节点上启动相应数量的服务实例副本，这些副本在启动成功后会自动加入负载均衡器中并生效，整个过程无须额外的人工操作。另外，Kubernetes 会定时巡查每个服务的所有实例的可用性，确保服务实例的数量始终保持为预期的数量，当它发现某个实例不可用时，会自动

重启该实例或者在其他节点重新调度、运行一个新实例，这样，一个复杂的过程无须人工干预即可全部自动化完成。试想一下，如果一个包括几十个节点且运行着几万个容器的复杂系统，其负载均衡、故障检测和故障修复等都需要人工介入进行处理，那将是多么难以想象。

通常我们会把 Kubernetes 看作 Docker 的上层架构，就好像 Java 与 J2EE 的关系一样：J2EE 是以 Java 为基础的企业级软件架构，而 Kubernetes 则以 Docker 为基础打造了一个云计算时代的全新分布式系统架构。但 Kubernetes 与 Docker 之间还存在着更为复杂的关系，从表面上看，似乎 Kubernetes 离不开 Docker，但实际上在 Kubernetes 的架构里，Docker 只是其目前支持的两种底层容器技术之一，另一个容器技术则是 Rocket，后者来源于 CoreOS 这个 Docker 昔日的“恋人”所推出的竞争产品。

Kubernetes 同时支持这两种互相竞争的容器技术，这是有深刻的历史原因的。快速发展的 Docker 打败了谷歌曾经名噪一时的开源容器技术 lxc，并迅速风靡世界。但是，作为一个已经对全球 IT 公司产生重要影响的技术，Docker 背后的容器标准的制定注定不可能被任何一个公司私有控制，于是就有了后来引发危机的 CoreOS 与 Docker 分手事件，其导火索是 CoreOS 撇开了 Docker，推出了与 Docker 相对抗的开源容器项目——Rocket，并动员一些知名 IT 公司成立委员会来试图主导容器技术的标准化，该分手事件愈演愈烈，最终导致 CoreOS “傍上”谷歌一起宣布“叛逃”Docker 阵营，共同发起了基于 CoreOS+Rocket+Kubernetes 的新项目 Tectonic。这让当时的 Docker 阵营和 Docker 粉丝们无比担心 Docker 的命运，不管最终鹿死谁手，容器技术分裂态势的加剧对所有牵涉其中的人来说都没有好处，于是 Linux 基金会出面调和矛盾，双方都退让一步，最终的结果是 Linux 基金会于 2015 年 6 月宣布成立开放容器技术项目（Open Container Project），谷歌、CoreOS 及 Docker 都加入了 OCP 项目。但通过查看 OCP 项目的成员名单，你会发现 Docker 在这个名单中只能算一个小角色了。OCP 的成立最终结束了这场让无数人揪心的“战争”，Docker 公司被迫放弃了自己的独家控制权。作为回报，Docker 的容器格式被 OCP 采纳为新标准的基础，并且由 Docker 负责起草 OCP 草案规范的初稿文档，当然这个“标准起草者”的角色也不是那么容易担当的，Docker 要提交自己的容器执行引擎的源码作为 OCP 项目的启动资源。

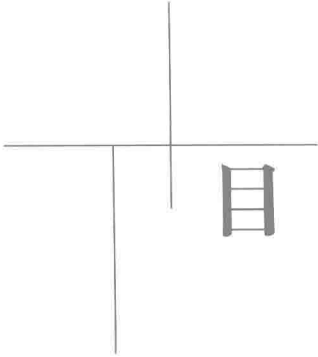
事到如今，我们再来回顾当初 CoreOS 与谷歌的叛逃事件，从表面上看，谷歌貌似是被诱拐“出柜”的，但局里人都明白，谷歌才是这一系列事件背后的主谋，其不仅为当年失败的 lxc 报了一箭之仇，还重新掌控了容器技术的未来。容器标准之战大捷之后，谷歌进一步扩大了联盟并提高了自身影响力。2015 年 7 月，谷歌正式宣布加入 OpenStack 阵营，其目标是确保 Linux 容器及关联的容器管理技术 Kubernetes 能够被 OpenStack 生态圈所容纳，并且成为 OpenStack 平台上与 KVM 虚拟机一样的一等公民。谷歌加入 OpenStack 意味着对数据中心控制平面的争夺已经结束，以容器为代表的形态与以虚拟化为代表的系统形态将会完美融合于 OpenStack 之上，并与软件定义网络和软件定义存储一起统治下一代数据中心。

谷歌凭借着几十年大规模容器使用的丰富经验，步步为营，先是祭出 Kubernetes 这个神器，然后又掌控了容器技术的制定标准，最后又入驻 OpenStack 阵营全力将 Kubernetes 扶上位，谷歌这个 IT 界的领导者和创新者再次王者归来。我们都明白，在 IT 世界里只有那些被大公司掌控和推广的，同时被业界众多巨头都认可和支持的新技术才能生存和壮大下去。Kubernetes 就是当今 IT 界里符合要求且为数不多的热门技术之一，它的影响力可能长达十年，所以，我们每个 IT 人都有理由重视这门新技术。

谁能比别人领先一步掌握新技术，谁就在竞争中赢得了先机。惠普中国电信解决方案领域的资深专家团一起分工协作，并行研究，废寝忘食地合力撰写，在短短的 5 个月内完成了这部厚达 500 多页的 Kubernetes 权威指南。经过一年的高速发展，Kubernetes 先后发布了 1.1、1.2 和 1.3 版本，每个版本都带来了大量的新特性，能够处理的应用场景也越来越丰富。本书遵循从入门到精通的学习路线，全书共分为六大章节，涵盖了入门、实践指南、架构原理、开发指南、高级案例、运维指南和源码分析等内容，内容详实、图文并茂，几乎囊括了 Kubernetes 1.3 版本的方方面面，无论是对于软件工程师、测试工程师、运维工程师、软件架构师、技术经理，还是对于资深 IT 人士来说，本书都极具参考价值。

吴治辉

惠普公司系统架构师



目 录

第 1 章	Kubernetes 入门	1
1.1	Kubernetes 是什么	1
1.2	为什么要用 Kubernetes	4
1.3	从一个简单的例子开始	5
1.3.1	环境准备	6
1.3.2	启动 MySQL 服务	7
1.3.3	启动 Tomcat 应用	9
1.3.4	通过浏览器访问网页	11
1.4	Kubernetes 基本概念和术语	12
1.4.1	Master	12
1.4.2	Node	13
1.4.3	Pod	15
1.4.4	Label (标签)	19
1.4.5	Replication Controller (RC)	22
1.4.6	Deployment	25
1.4.7	Horizontal Pod Autoscaler (HPA)	27
1.4.8	Service (服务)	29
1.4.9	Volume (存储卷)	35

1.4.10	Persistent Volume	39
1.4.11	Namespace（命名空间）	40
1.4.12	Annotation（注解）	42
1.4.13	小结	42
第 2 章 Kubernetes 实践指南		43
2.1	Kubernetes 安装与配置	43
2.1.1	安装 Kubernetes	43
2.1.2	配置和启动 Kubernetes 服务	45
2.1.3	Kubernetes 集群的安全设置	51
2.1.4	Kubernetes 的版本升级	57
2.1.5	内网中的 Kubernetes 相关配置	57
2.1.6	Kubernetes 核心服务配置详解	58
2.1.7	Kubernetes 集群网络配置方案	72
2.2	kubectl 命令行工具用法详解	80
2.2.1	kubectl 用法概述	80
2.2.2	kubectl 子命令详解	82
2.2.3	kubectl 参数列表	84
2.2.4	kubectl 输出格式	84
2.2.5	kubectl 操作示例	86
2.3	Guestbook 示例：Hello World	87
2.3.1	创建 redis-master RC 和 Service	89
2.3.2	创建 redis-slave RC 和 Service	91
2.3.3	创建 frontend RC 和 Service	93
2.3.4	通过浏览器访问 frontend 页面	96
2.4	深入掌握 Pod	97
2.4.1	Pod 定义详解	97

2.4.2	Pod 的基本用法	102
2.4.3	静态 Pod	107
2.4.4	Pod 容器共享 Volume	108
2.4.5	Pod 的配置管理	110
2.4.6	Pod 生命周期和重启策略	123
2.4.7	Pod 健康检查	124
2.4.8	玩转 Pod 调度	126
2.4.9	Pod 的扩容和缩容	135
2.4.10	Pod 的滚动升级	139
2.5	深入掌握 Service	143
2.5.1	Service 定义详解	143
2.5.2	Service 基本用法	145
2.5.3	集群外部访问 Pod 或 Service	150
2.5.4	DNS 服务搭建指南	153
2.5.5	Ingress: HTTP 7 层路由机制	161

第 3 章 Kubernetes 核心原理 165

3.1	Kubernetes API Server 原理分析	165
3.1.1	Kubernetes API Server 概述	165
3.1.2	独特的 Kubernetes Proxy API 接口	168
3.1.3	集群功能模块之间的通信	169
3.2	Controller Manager 原理分析	170
3.2.1	Replication Controller	171
3.2.2	Node Controller	173
3.2.3	ResourceQuota Controller	174
3.2.4	Namespace Controller	176
3.2.5	Service Controller 与 Endpoint Controller	176

3.3	Scheduler 原理分析	177
3.4	kubelet 运行机制分析	181
3.4.1	节点管理	181
3.4.2	Pod 管理	182
3.4.3	容器健康检查	183
3.4.4	cAdvisor 资源监控	184
3.5	kube-proxy 运行机制分析	186
3.6	深入分析集群安全机制	190
3.6.1	API Server 认证	190
3.6.2	API Server 授权	192
3.6.3	Admission Control 准入控制	194
3.6.4	Service Account	195
3.6.5	Secret 私密凭据	200
3.7	网络原理	203
3.7.1	Kubernetes 网络模型	203
3.7.2	Docker 的网络基础	205
3.7.3	Docker 的网络实现	217
3.7.4	Kubernetes 的网络实现	225
3.7.5	开源的网络组件	229
3.7.6	网络实战	234

第 4 章 Kubernetes 开发指南 247

4.1	REST 简述	247
4.2	Kubernetes API 详解	249
4.2.1	Kubernetes API 概述	249
4.2.2	API 版本	254
4.2.3	API 详细说明	254

4.2.4	API 响应说明	256
4.3	使用 Java 程序访问 Kubernetes API	258
4.3.1	Jersey	258
4.3.2	Fabric8	270
4.3.3	使用说明	271
第 5 章	Kubernetes 运维指南	292
<hr/>		
5.1	Kubernetes 集群管理指南	292
5.1.1	Node 的管理	292
5.1.2	更新资源对象的 Label	294
5.1.3	Namespace: 集群环境共享与隔离	295
5.1.4	Kubernetes 资源管理	299
5.1.5	Kubernetes 集群高可用部署方案	333
5.1.6	Kubernetes 集群监控	343
5.1.7	kubelet 的垃圾回收 (GC) 机制	361
5.2	Kubernetes 高级案例	362
5.2.1	ElasticSearch 日志搜集查询和展现案例	362
5.2.2	Cassandra 集群部署案例	371
5.3	Trouble Shooting 指导	376
5.3.1	查看系统 Event 事件	377
5.3.2	查看容器日志	379
5.3.3	查看 Kubernetes 服务日志	379
5.3.4	常见问题	381
5.3.5	寻求帮助	384
5.4	Kubernetes v1.3 开发中的新功能	385
5.4.1	Pet Set (有状态的容器)	385
5.4.2	Init Container (初始化容器)	388
5.4.3	Cluster Federation (集群联邦)	391

第 6 章 Kubernetes 源码导读

396

6.1	Kubernetes 源码结构和编译步骤	396
6.2	kube-apiserver 进程源码分析	400
6.2.1	进程启动过程	400
6.2.2	关键代码分析	402
6.2.3	设计总结	417
6.3	kube-controller-manager 进程源码分析	420
6.3.1	进程启动过程	420
6.3.2	关键代码分析	423
6.3.3	设计总结	431
6.4	kube-scheduler 进程源码分析	433
6.4.1	进程启动过程	434
6.4.2	关键代码分析	438
6.4.3	设计总结	445
6.5	kubelet 进程源码分析	447
6.5.1	进程启动过程	447
6.5.2	关键代码分析	452
6.5.3	设计总结	475
6.6	kube-proxy 进程源码分析	476
6.6.1	进程启动过程	476
6.6.2	关键代码分析	478
6.6.3	设计总结	493
6.7	kubectl 进程源码分析	494
6.7.1	kubectl create 命令	495
6.7.2	rolling-update 命令	499
	后记	505

第 1 章

Kubernetes 入门

1.1 Kubernetes 是什么

Kubernetes 是什么？

首先，它是一个全新的基于容器技术的分布式架构领先方案。这个方案虽然还很新，但它是谷歌十几年以来大规模应用容器技术的经验积累和升华的一个重要成果。确切地说，Kubernetes 是谷歌严格保密十几年的秘密武器——Borg 的一个开源版本。Borg 是谷歌的一个久负盛名的内部使用的大规模集群管理系统，它基于容器技术，目的是实现资源管理的自动化，以及跨多个数据中心的资源利用率的最大化。十几年来，谷歌一直通过 Borg 系统管理着数量庞大的应用程序集群。由于谷歌员工都签署了保密协议，即便离职也不能泄露 Borg 的内部设计，所以外界一直无法了解关于它的更多信息。直到 2015 年 4 月，传闻许久的 Borg 论文伴随 Kubernetes 的高调宣传被谷歌首次公开，大家才得以了解它的更多内幕。正是由于站在 Borg 这个前辈的肩膀上，吸取了 Borg 过去十年间的经验与教训，所以 Kubernetes 一经开源就一鸣惊人，并迅速称霸了容器技术领域。

其次，如果我们的系统设计遵循了 Kubernetes 的设计思想，那么传统系统架构中那些和业务没有多大关系的底层代码或功能模块，都可以立刻从我们的视线中消失，我们不必再费心于负载均衡器的选型和部署实施问题，不必再考虑引入或自己开发一个复杂的服务治理框架，不必再头疼于服务监控和故障处理模块的开发。总之，使用 Kubernetes 提供的解决方案，我们不仅节省了不少于 30% 的开发成本，同时可以将精力更加集中于业务本身，而且由于 Kubernetes 提供了强大的自动化机制，所以系统后期的运维难度和运维成本大幅度降低。

然后，Kubernetes 是一个开放的开发平台。与 J2EE 不同，它不局限于任何一种语言，没有

限定任何编程接口，所以不论是用 Java、Go、C++ 还是用 Python 编写的服务，都可以毫无困难地映射为 Kubernetes 的 Service，并通过标准的 TCP 通信协议进行交互。此外，由于 Kubernetes 平台对现有的编程语言、编程框架、中间件没有任何侵入性，因此现有的系统也很容易改造升级并迁移到 Kubernetes 平台上。

最后，Kubernetes 是一个完备的分布式系统支撑平台。Kubernetes 具有完备的集群管理能力，包括多层次的安全防护和准入机制、多租户应用支撑能力、透明的服务注册和服务发现机制、内建智能负载均衡器、强大的故障发现和自我修复能力、服务滚动升级和在线扩容能力、可扩展的资源自动调度机制，以及多粒度的资源配额管理能力。同时，Kubernetes 提供了完善的管理工具，这些工具涵盖了包括开发、部署测试、运维监控在内的各个环节。因此，Kubernetes 是一个全新的基于容器技术的分布式架构解决方案，并且是一个一站式的完备的分布式系统开发和支撑平台。

在正式开始本章的 Hello World 之旅之前，我们首先要学习 Kubernetes 的一些基本知识，这样我们才能理解 Kubernetes 提供的解决方案。

在 Kubernetes 中，Service（服务）是分布式集群架构的核心，一个 Service 对象拥有如下关键特征。

- ❶ 拥有一个唯一指定的名字（比如 mysql-server）。
- ❷ 拥有一个虚拟 IP（Cluster IP、Service IP 或 VIP）和端口号。
- ❸ 能够提供某种远程服务能力。
- ❹ 被映射到了提供这种服务能力的一组容器应用上。

Service 的服务进程目前都基于 Socket 通信方式对外提供服务，比如 Redis、Memcache、MySQL、Web Server，或者是实现了某个具体业务的一个特定的 TCP Server 进程。虽然一个 Service 通常由多个相关的服务进程来提供服务，每个服务进程都有一个独立的 Endpoint（IP+Port）访问点，但 Kubernetes 能够让我们通过 Service（虚拟 Cluster IP +Service Port）连接到指定的 Service 上。有了 Kubernetes 内建的透明负载均衡和故障恢复机制，不管后端有多少服务进程，也不管某个服务进程是否会由于发生故障而重新部署到其他机器，都不会影响到我们对服务的正常调用。更重要的是这个 Service 本身一旦创建就不再变化，这意味着，在 Kubernetes 集群中，我们再也不用为了服务的 IP 地址变来变去的问题而头疼了。

容器提供了强大的隔离功能，所以有必要把为 Service 提供服务的这组进程放入容器中进行隔离。为此，Kubernetes 设计了 Pod 对象，将每个服务进程包装到相应的 Pod 中，使其成为 Pod 中运行的一个容器（Container）。为了建立 Service 和 Pod 间的关联关系，Kubernetes 首先给每个 Pod 贴上一个标签（Label），给运行 MySQL 的 Pod 贴上 name=mysql 标签，给运行 PHP 的

Pod 贴上 `name=php` 标签, 然后给相应的 Service 定义标签选择器 (Label Selector), 比如 MySQL Service 的标签选择器的选择条件为 `name=mysql`, 意为该 Service 要作用于所有包含 `name=mysql` Label 的 Pod 上。这样一来, 就巧妙地解决了 Service 与 Pod 的关联问题。

说到 Pod, 我们这里先简单介绍其概念。首先, Pod 运行在一个我们称之为节点 (Node) 的环境中, 这个节点既可以是物理机, 也可以是私有云或者公有云中的一个虚拟机, 通常在一个节点上运行几百个 Pod; 其次, 每个 Pod 里运行着一个特殊的被称之为 Pause 的容器, 其他容器则为业务容器, 这些业务容器共享 Pause 容器的网络栈和 Volume 挂载卷, 因此它们之间的通信和数据交换更为高效, 在设计时我们可以充分利用这一特性将一组密切相关的服务进程放入同一个 Pod 中; 最后, 需要注意的是, 并不是每个 Pod 和它里面运行的容器都能“映射”到一个 Service 上, 只有那些提供服务 (无论是对内还是对外) 的一组 Pod 才会被“映射”成一个服务。

在集群管理方面, Kubernetes 将集群中的机器划分为一个 Master 节点和一群工作节点 (Node)。其中, 在 Master 节点上运行着集群管理相关的一组进程 `kube-apiserver`、`kube-controller-manager` 和 `kube-scheduler`, 这些进程实现了整个集群的资源管理、Pod 调度、弹性伸缩、安全控制、系统监控和纠错等管理功能, 并且都是全自动完成的。Node 作为集群中的工作节点, 运行真正的应用程序, 在 Node 上 Kubernetes 管理的最小运行单元是 Pod。Node 上运行着 Kubernetes 的 `kubelet`、`kube-proxy` 服务进程, 这些服务进程负责 Pod 的创建、启动、监控、重启、销毁, 以及实现软件模式的负载均衡器。

最后, 我们再来看看传统的 IT 系统中服务扩容和服务升级这两个难题, 以及 Kubernetes 所提供的全新解决思路。服务的扩容涉及资源分配 (选择哪个节点进行扩容)、实例部署和启动等环节, 在一个复杂的业务系统中, 这两个问题基本上靠人工一步步操作才得以完成, 费时费力又难以保证实施质量。

在 Kubernetes 集群中, 你只需为需要扩容的 Service 关联的 Pod 创建一个 Replication Controller (简称 RC), 则该 Service 的扩容以至于后来的 Service 升级等头疼问题都迎刃而解。在一个 RC 定义文件中包括以下 3 个关键信息。

- ① 目标 Pod 的定义。
- ② 目标 Pod 需要运行的副本数量 (Replicas)。
- ③ 要监控的目标 Pod 的标签 (Label)。

在创建好 RC (系统将自动创建好 Pod) 后, Kubernetes 会通过 RC 中定义的 Label 筛选出对应的 Pod 实例并实时监控其状态和数量, 如果实例数量少于定义的副本数量 (Replicas), 则会根据 RC 中定义的 Pod 模板来创建一个新的 Pod, 然后将此 Pod 调度到合适的 Node 上启动运行, 直到 Pod 实例的数量达到预定目标。这个过程完全是自动化的, 无须人工干预。有了 RC,