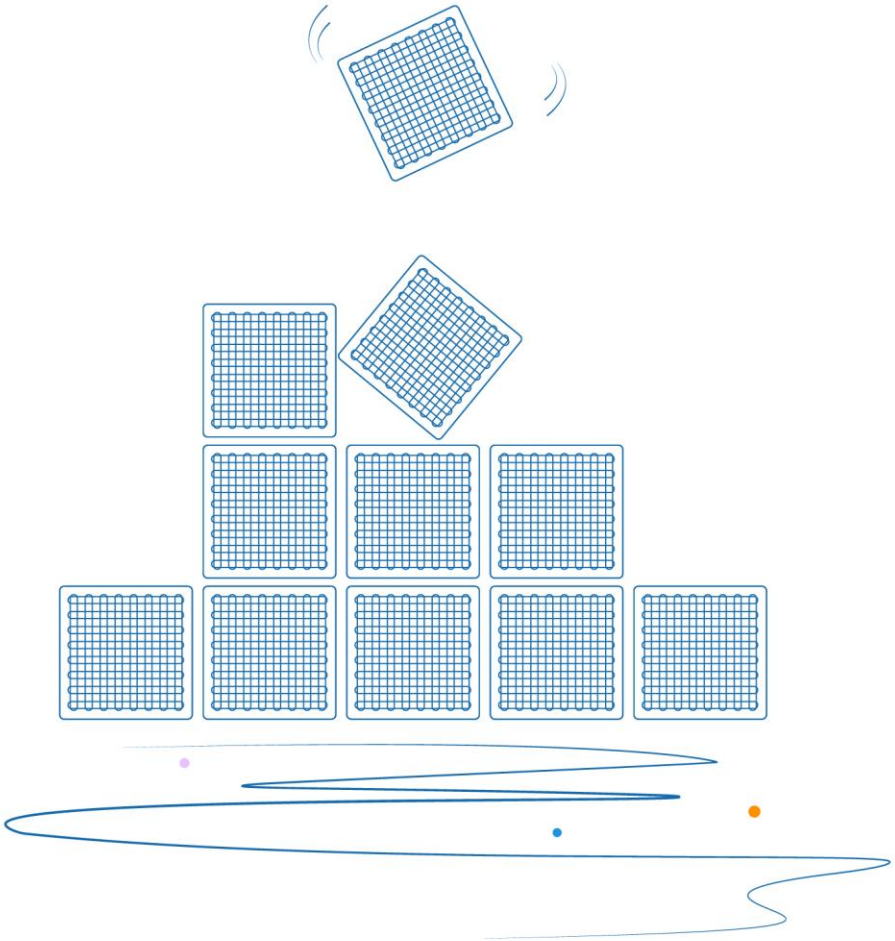
容器技术及其应用
白皮书［V1.0］



指导单位：工业和信息化部信息化和软件服务业司
编写单位：中国开源云联盟容器工作组

2016 年 11 月 7 日发布

内容摘要

近年来，容器技术及相关应用得到了国内外越来越多的关注度，研发和应用推广发展势头迅猛。在国外，容器技术已经形成了较成熟的生态圈；在国内，金融企业、互联网企业、IT 企业积极投入容器技术研发和应用推广，发展势头迅猛。为了积极引导我国容器技术和应用发展，我们编写本白皮书。其主要内容包括：

一、针对容器技术现状进行研究和分析。一是梳理了容器技术从开始到现在的发展历程，对现有容器发展的生态结构进行分析，其中包括开源社区、产业联盟、解决方案厂商等；二是对容器技术框架进行了详细的描述，对技术框架各层涉及的技术点进行了介绍；三是结合已发布的国家和国际标准，将现有容器技术对于参考架构的实现情况进行分析；四是分析了容器技术与大数据、物联网、SDN 之间的关系。

二、容器技术发展路线及技术架构。通过列举容器技术典型 4 个应用场景，包括 PaaS 平台建设、软件定义数据中心、容器即服务、持续集成和发布等，分析了容器技术在各种场景下的关键成功因素。

三、容器未来发展趋势。结合容器发展现状和应用场景应用情况，分析了容器技术在应用过程中面临的问题，同时提出了容器今后发展的生态图，对未来容器技术发展进行了展望。

指导单位

工业和信息化部信息化和软件服务业司

编写单位（排名不分先后）

北京数人科技有限公司（数人云）

中国电子技术标准化研究院（工业和信息化部电子工业标准化研究院）

央视国际网络有限公司（央视网）

Hyper

北京凌云雀科技有限公司（灵雀云）

北京天云融创软件技术有限公司

新华三集团

阿里云计算有限公司

Vmware（威睿）

DockOne 社区

北京国电通网络技术有限公司

去哪儿网

中国国际航空股份有限公司

Intel（英特尔公司）

中国移动苏州研发中心

北京鼎震科技有限责任公司

编写人员

陈志峰　肖德时　周　平　杨丽蕴　赵　鹏　牛继宾　韩　萍

王　雷　周明渡　闫长海　刘　斌　胡　震　王志鹏　吴　涛

张航源　彭丰华　王　旭　李亦波　周小璐　王　锋　张　峻

赖寒雪　王洁萍　何钦钦　徐　磊　李颖杰　吴国华　魏晓铭

吕晓旭　杜永丰　曹高晋　张争艳　徐　超　李　璐

版权声明

目 录

一、概述

1.1 背景

继虚拟化技术出现后，容器技术逐渐成为对云计算领域具有深远影响的变革技术。容器技术的发展和应用，将为各行业应用云计算提供了新思路，同时容器技术也将对云计算的交付方式、效率、PaaS 平台的构建等方面产生深远的影响，具体体现在以下几个方面：

简化部署：容器技术可以将应用打包成单一地址访问的、Registry 存储的、仅通过一行命令就可以部署完成的组件。不论将服务部署在哪里，容器都可以从根本上简化服务部署工作。

快速启动：容器技术对操作系统的资源进行再次抽象，而并非对整个物理机资源进虚拟化，通过这种方式，打包好的服务可以快速启动。

服务组合：采用容器的方式进行部署，整个系统会变得易于组合，通过容器技术将不同服务封装在对应的容器中，之后结合一些脚本使这些容器按照要求相互协作，这样操作不仅可以简化部署难度还可以降低操作风险。

易于迁移：容器技术最重要的价值就是为在不同主机上运行服务提供一个轻便的、一致的格式。容器格式的标准化加快交付体验，允许用户方便地对工作负载进行迁移，避免局限于单一的平台提供商。

为更好地推进容器及相关技术在中国的落地与实践，推动容器技术在国内的落地，并建立顺应国际技术发展趋势、符合中国本地化特征的容器标准体系，中国开源云联盟容器工作组开展了本白皮书的研制工作，白皮书立足于容器技术发展的演进路线图，分析容器技术在应用过程中的应用场景以及面临的具体问题和关键成功因素，描绘容器技术未来的发展趋势和方向。

本白皮书的发布，旨在与业界分享我们在容器技术领域的研究成果和实践经验，呼吁社会各界共同关注容器技术的同时，共同推动容器技术的发展，提升容器技术在云计算领域中实践和服务能力。

1.2 相关术语

表 1.1 术语

术语	定义/解释
镜像	系统文件及其应用文件以特殊的文件形式进行备份制作成单一的文件。
微服务架构	微服务架构是一种特定的软件应用程序设计方式——将大型软件拆分为多个独立可部署服务组合而成的套件方案。
开发运维一体化	可定义为是一种过程、方法、文化、运动或实践，主要是为了通过一条高度自动化的流水线来加强开发和其他 IT 职能部门之间的沟通和协作，加速软件和服务的交付。
运行时引擎	指用户用来运行容器镜像的软件系统。

1.3 缩略语

表 1.2 缩略语

术语	解释
CI/CD	Continuous Integration/Continuous Delivery，持续集成和持续交付
CaaS	Container as a Service，容器即服务
CCRA	Cloud Computing Reference Architecture，云计算参考架构
CLI	command-line interface，命令行界面
DC/OS	DataCenter Operating System，数据中心操作系统
DevOps	Development and Operations，开发运维一体化
DNS	Domain Name System，域名系统
IaaS	Infrastructure as a Service，基础设施即服务
LXC	Linux Container，Linux 容器
OCI	Open Container Initiative，开放容器组织
PaaS	Platform as a Service，平台即服务
SaaS	Software as a Service，软件即服务
SDN	Software Defined Network，软件定义网络
VPS	Virtual Private Server，虚拟机专有服务
VM	Virtual Machine，虚拟机

二、容器技术现状

2.1 容器技术发展演进路径

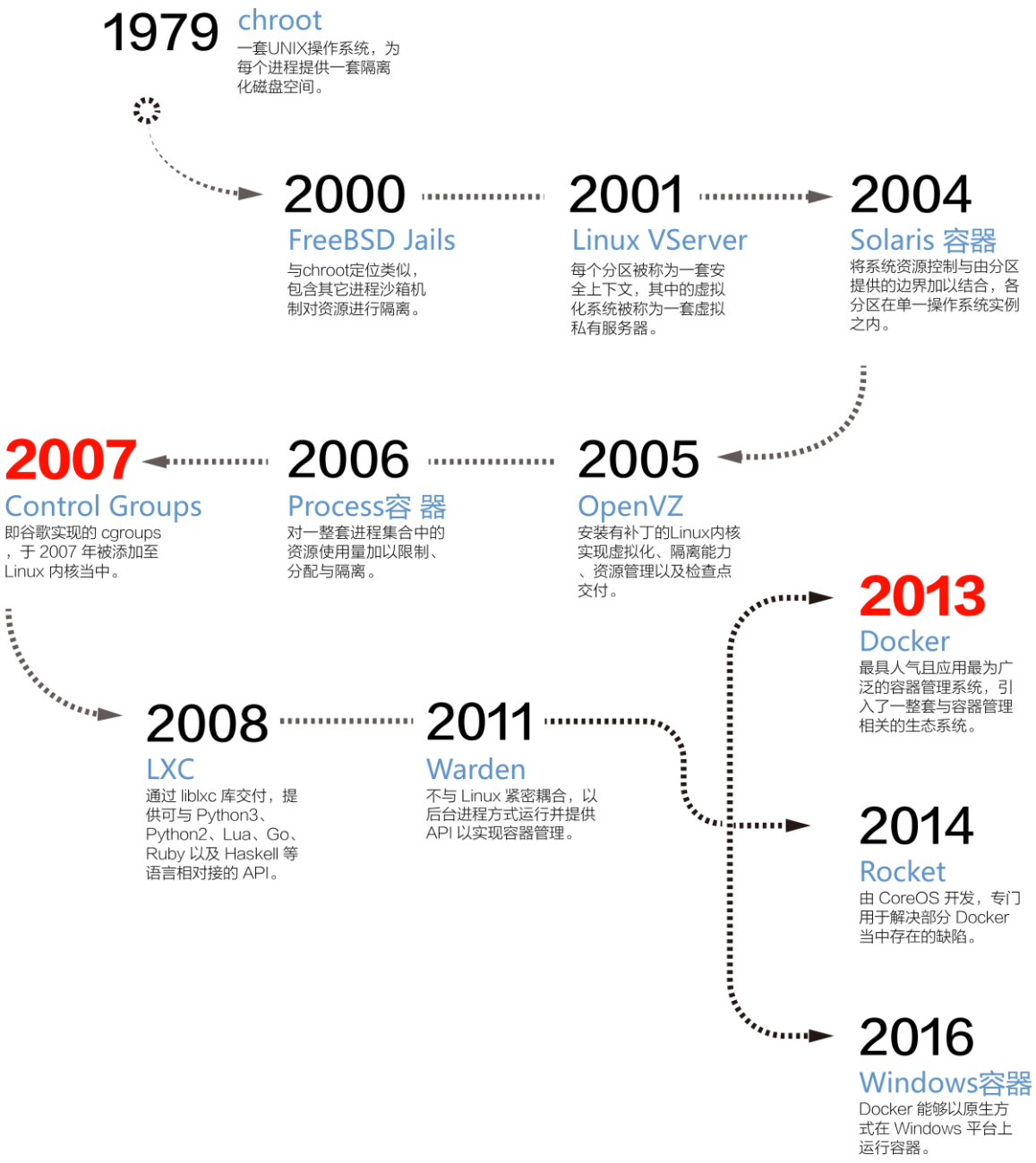


图 2.1 容器技术演变路径

容器技术演变路径如图 2.1。容器技术最早可以追溯到 1979 年 UNIX 系统中的 chroot，最初是为了方便切换 root 目录，为每个进程提供了文件系统资源的隔离，这也是 OS 虚拟化思想的起源。2000 年，BSD 吸收并改进了 chroot 技术，

发布了 FreeBSD Jails。FreeBSD Jails 除文件系统隔离，还添加了用户和网络资源等的隔离，每个 Jail 还能分配一个独立 IP，进行一些相对独立的软件安装和配置。2001 年，Linux 发布了 Linux Vserver，Linux VServer 依旧是延续了 Jails 的思想，在一个操作系统上隔离文件系统、CPU 时间、网络地址和内存等资源，每一个分区都被称为一个 security context，内部的虚拟化系统被称为 VPS。2004 年原 SUN 公司发布了 Solaris Containers，Solaris Containers 作为 Solaris 10 中的特性发布的，包含了系统资源控制和 zones 提供的二进制隔离，Zones 作为在操作系统实例内一个完全隔离的虚拟服务器存在。2005 年 SWsoft 公司发布了 OpenVZ，OpenVZ 和 Solaris Containers 非常类似，通过打了补丁的 Linux 内核来提供虚拟化、隔离、资源管理和检查点。OpenVZ 标志着内核级别的虚拟化真正成为主流，之后不断有相关的技术被加入内核。2006 年 Google 发布了 Process Containers，Process Container 记录和隔离每个进程的资源使用（包括 CPU、内存、硬盘 I/O、网络等），后改名为 cgroups（Control Groups），并在 2007 年被加入 Linux 内核 2.6.24 版本中。2008 年出现了第一个比较完善的 LXC 容器技术，基于已经被加入内核的 cgroups 和 Linux namespaces 实现。不需要打补丁，LXC 就能运行在任意 vanila 内核的 Linux 上。2011 年，CloudFoundry 发布了 Warden，和 LXC 不同，Warden 可以工作在任何操作系统上，作为守护进程运行，还提供了管理容器的 API。2013 年 Google 公司建立了开源的容器技术栈 lmctfy，Google 开启这个项目是为了通过容器实现高性能，高资源利用率，同时接近零开销的虚拟化技术。目前 Kubernetes 中的监控工具 cAdvisor 就起源于 lmctfy 项目，2015 年 Google 将 lmctfy 的核心技术贡献给了 libcontainer。2013 年 Docker 诞生，Docker 最早是 dotCloud（Docker 公司的

前身，是一家 PaaS 公司）内部的项目，和 Warden 类似，Docker 最初也用了 LXC，后来才自己写了 libcontainer 替换了 LXC。和其它容器技术不同的是，Docker 围绕容器构建了一套完整的生态，包括容器镜像标准、容器 Registry、REST API、CLI、容器集群管理工具 Docker Swarm 等；2014 年 CoreOS 创建了 rkt，为了改进 Docker 在安全方面的缺陷，重写的一个容器引擎，相关容器工具产品包括：服务发现工具 etcd 和网络工具 flannel 等。2016 年微软公司发布基于 Windows 的容器技术 Hyper-V Container，Hyper-V Container 原理和 Linux 下的容器技术类似,可以保证在某个容器里运行的进程与外界是隔离的，兼顾虚拟机的安全性和容器的轻量级。

2.2 容器技术发展生态

随着容器技术的演进，越来越多的机构开始重视并参与到容器技术的探索中来。从最初的以 Unix 或 Linux 项目到开源社区，到各种类型的容器技术创业公司、IT 企业及产业联盟，容器技术的发展生态也在逐渐得到发展与丰富。在开源社区方面，附录 A 中列出了国际上的 OCI 和 CNCF（Cloud Native Computing Foundation，简称 CNCF，下同），容器的开源项目也在附录 B 中列出；在国内的 IT 企业和创业公司方面，还提供了多个行业应用案例，这些行业应用案例在附录 D 中列出；产业联盟方面包括国际包括 CNCF，国内包括中国开源云联盟（COSCL）。容器的开源社区、创业公司、IT 企业、产业联盟共同构成容器技术发展的生态圈。

2.3 容器技术框架

通过研究、梳理和分析现有的容器技术，形成容器相关技术的技术架构，如图 2.2 所示。

图 2.2 容器技术框架

2.3.1 服务器层

当运行容器镜像时，容器本身需要运行在传统操作系统之上，而这个操作系统既可以是基于物理机，也可以是基于 VM。服务器层包含了这两种场景，泛指了容器运行的环境，同时容器并不关心服务器层如何提供和管理，它的期望只是能获得这些服务器资源。

2.3.2 资源管理层

资源管理包含了服务器、操作系统等资源的管理。其中如果是物理服务器的话，需要涉及物理机管理系统（例如 Rocks 等）；如果是虚拟机的话，需要使用虚拟化平台。此外，无论是物理服务器还是虚拟机，都需要对其中的操作系统加以管理(例如：Chef、Puppet、Ansible 和 SaltStack 等)。而传统的存储和网络管理也包含在资源管理层。由于存储，网络两者选择众多，不一而足，因此不再列举。

总而言之，资源管理层的核心目标是对服务器和操作系统资源进行管理，以支持上层的容器运行引擎。

2.3.3 运行引擎层

容器运行引擎层主要指常见的容器系统，包括 Docker、rkt、Hyper、CRI-O。这些容器系统的共通作用包括启动容器镜像、运行容器应用和管理容器实例。运行引擎又可以分为管理程序和运行时环境两个模块。

需要注意的是，运行引擎是单机程序，类似虚拟化软件的 KVM 和 Xen，不是集群分布式系统。引擎运行于服务器操作系统之上，接受上层集群系统的管理。

相关开源项目包括：

——资源隔离：Cgroup、Hypervisor；

——访问限制：Namespace、Hypervisor；

——管理程序：Docker Engine、OCID、hyperd，RKT、CRI-O；

——运行时环境：runC(Docker)、runV(Hyper)、runZ (Solaris)

2.3.4 集群管理层

可以把容器的集群管理系统类和针对 VM 的集群管理系统划等号，都是通过对一组服务器运行分布式应用。而这两者的细微区别在于，VM 的集群管理系统需要运行在物理服务器上，而容器集群管理系统既可以运行在物理服务器上，也可以运行在 VM 上。

常见的容器集群管理系统包括：Kubernetes、Docker Swarm、Mesos。这三者各有特色，但随着时间推移，三者的融合将越发明显。Kubernetes 在这三者中比较特殊，它的地位更接近 OpenStack。围绕这 Kubernetes，CNCF 基金会已经建立了一个非常强大的生态体系，这是 Docker Swarm 和 Mesos 都不具备的。而 CNCF 基金会本身也正向着容器界的 OpenStack 基金会发展，值得大家重点关注。

集群管理层涉及到的相关开源软件项目包括：

——指挥调度：Docker Swarm、Kubernetes、Mesos 等

——服务发现：Etcd、Consul、Zookeeper,DNS

——监控：Prometheous

——存储：Flocker

——网络：Calico、Weave、Flannel

2.3.5 应用层

泛指所有运行于容器之上的应用程序，以及所需的辅助系统，包括：监控、日志、安全、编排、镜像仓库等等。

——监控模块，相关开源项目包括：Prometheous、cAdvisor、Sysdig 等；

——日志，相关开源项目包括：Fluented、LogStash 等；

——安全，包括容器镜像的安全扫描，运行环境的安全隔离，集群环境的安全管理等功能；

——编排，相关开源项目包括：Docker Compose、CoreOS Fleet 等；

——CI/CD，相关开源项目包括：Jenkins、Buildbot、Gitlab CI、Drone.io；

——镜像仓库：Docker Hub、VMware Harbor、Huawei Dockyard。

2.4 容器技术对参考架构的实现情况

国家标准 GB/T 32399-2015《信息技术 云计算 参考架构》（简称 CCRA，修改采用 ISO/IEC 17789）是 2015 年发布的国家标准，描述了云计算的利益相关者，云计算系统的基本特征，云计算的基本活动和功能组件，我国是该国际标准的立项推动国之一，积极参与了该国际标准的编制，该标准的诞生标志着国际三

大标准化组织 ISO、IEC 和 ITU 首次在云计算领域统一认识并达成一致，是国际国内云计算领域的最重要的基础性标准。在该标准中描述了云计算的功能架构，功能架构包含了支撑云计算活动所需的功能，如图 2.3。图中标蓝色的部分是现有容器技术已经实现的内容。

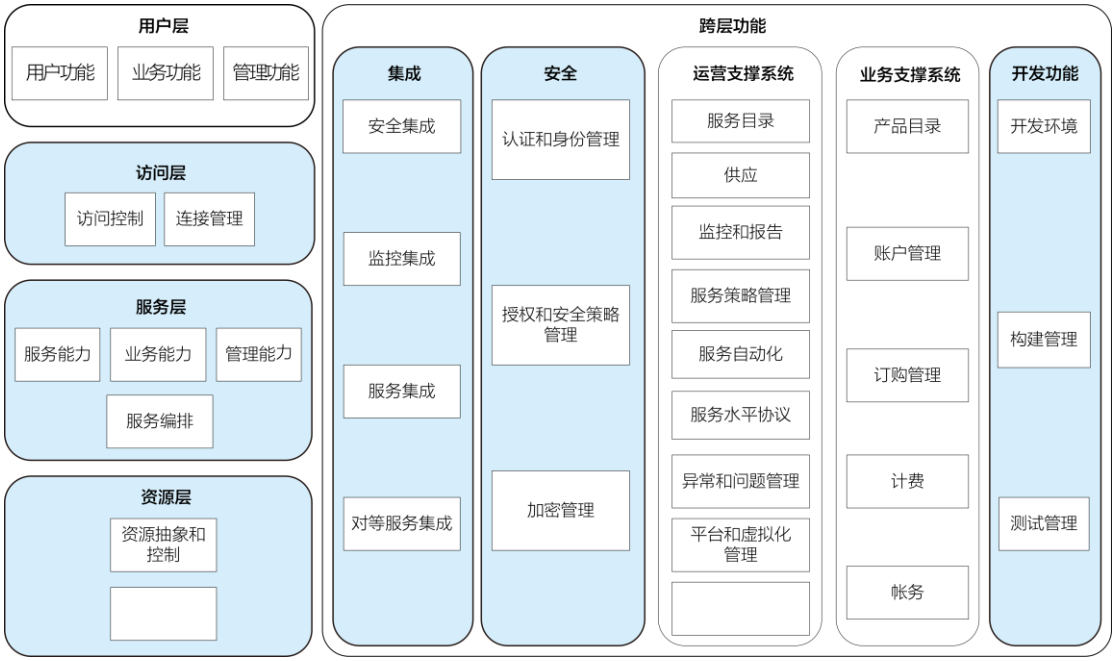


图 2.3 CCRA 功能组件图

2.5 容器技术的关键技术内容

2.5.1 镜像

容器的镜像通常包括操作系统文件、应用本身的文件、应用所依赖的软件包和库文件。为了提高容器镜像的管理效率，容器的镜像采用分层的形式存放。容器的镜像最底层通常是 Linux 的 rootfs 和系统文件，再往上则是各种软件包层。这些文件层在叠加后成为完整的只读文件系统，最终挂载到容器里面。在运行过程中，容器应用往往需要写入文件数据，容器引擎为此需再创建一个可写层，加在镜像的只读文件系统上面。使用分层的容器镜像之后，镜像的下载和传输更加便利，因为只需要在宿主机上把缺少的镜像文件层次下载即可，无需整个镜像传

送。

在 Linux 中，联合文件系统 UnionFS 能够把多个文件层叠加在一起，并透明地展现成一个完整的文件系统。常见的联合文件系统有 AUFS(Another Union File System)，btrfs，OverlayFS 和 DeviceMapper 等。

2.5.2 运行时引擎

容器运行时引擎和容器镜像两者的关系类似于虚拟化软件和虚拟机镜像的关系。容器运行时引擎的技术标准主要是由 OCI 基金会领导社区进行制定。目前 OCI 已经发布了容器运行时引擎的技术规范，并认可了 runC（Docker 公司提供）和 runV（Hyper 公司提供）两种合规的运行引擎。

2.5.3 容器编排

容器编排工具通过对容器服务的编排，决定容器服务之间如何进行交互。容器编排工具一般要处理以下几方面的内容：

1)、容器的启动。选择启动的机器、镜像和启动参数等；

2)、容器的应用部署。提供方法对应用进行部署；

3)、容器应用的在线升级。提供方法可以平滑地切换到应用新版本。

容器的编排一般是通过描述性语言 YAML 或者 JSON 来定义编排的内容。目前主要的编排工具有 Docker compose 和基于 Google 的 Kubernetes helm 等。

2.5.4 容器集群

容器集群是将多台物理机抽象为逻辑上单一调度实体的技术，为容器化的应用提供资源调度、服务发现、弹性伸缩、负载均衡等功能，同时监控和管理整个服务器集群，提供高质量、不间断的应用服务。容器集群主要包含以下技术：

资源调度:主要以集中化的方式管理和调度资源，按需为容器提供 CPU、内

存等资源；

服务发现：通过全局可访问的注册中心实现任何一个应用能够获取当前环境的细节，自动加入到当前的应用集群中；

弹性伸缩：在资源层面，监控集群资源使用情况，自动增减主机资源；在应用层面，可通过策略自动增减应用实例来实现业务能力的弹性伸缩；

负载均衡：当应用压力增加，集群自动扩展服务将负载均衡至每一个运行节点；当某个节点出现故障，应用实例重新部署运行到健康的节点上。

2.5.5 服务注册和发现

容器技术在构建自动化运维场景中，服务注册和发现是重要的两个环节，一般通过一个全局性的配置服务来实现。其基本原理类似公告牌信息发布系统，A 服务（容器应用或者普通应用）启动后在配置服务器（公告牌）上注册一些对外信息（比如 IP 和端口），B 服务通过查询配置服务器（公告牌）来获取 A 注册的信息（IP 和端口）。

2.5.6 热迁移

热迁移（Live Migration），又称为动态迁移或者实时迁移，是指将整容器的运行时状体完整保存下来，同时可以快速地在其他主机或平台上恢复运行。容器热迁移主要应用在两个方面：一是有多个操作单元执行任务，热迁移能迅速地复制与迁移容器，做到无感知运行作业；二是可以处理数据中心中集群的负载均衡，大量数据涌来无法运行计算时，可利用热迁移创建多个容器处理运算任务，调节信息数据处理峰谷，配置管理负载均衡比例，降低应用延迟。

2.6 容器技术与相关技术的关系

2.6.1 容器与云计算

虚拟化是云计算的重要基础，容器定义了一套从构建到执行的标准化体系，改变了传统的虚拟化技术，深度影响了云计算领域，容器是云计算的未来。以 Docker 为代表的容器技术越来越深刻地影响云计算，也改变我们的日常开发、运维和测试。相比于虚拟机，容器的轻量、快速启动和低开销，以及基于此的按业务打包和微服务模式，这些特点被用来改进 DevOps，很多场景下更适合做大规模集群管理和搭建灵活的分布式系统。通过深度整合了 IaaS、PaaS 及容器技术，提供弹性计算、DevOps 工具链及微服务基础设施等服务，帮助企业解决 IT、架构及运维等问题，使企业更聚焦于业务，构建了新一代的云计算生态体系。

2.6.2 容器与大数据

大数据平台如果能采用容器方式发布，与 Spark、Hadoop、Cassandra 等相关技术的集成与对接，可降低整个系统的搭建难度，缩短交付和安装周期，减少安装失败风险。容器化后，各类大数据平台组件可以轻松实现迁移的目的，也能实现多复本控制和高可用。

2.6.3 容器与物联网

物联网(IoT)技术发展日新月异，而容器技术刚好遇到这样的机遇，将在几个方面促进物联网的发展。

首先，运用容器技术后，可通过容器封装，可简化下载、安装部署、启动和后续应用更新。这将大大加速物联网应用开发部署。其次，容器技术还可以满足物联网在自动监控，集中式维护管理方面的需求。最后，数据采集端环境千变万

化，如果需要手动适配工作量巨大，如果采用容器化技术，只要打包几类典型的容器镜像，如 ARM,X86,x86_64 等，就可以事半功倍实现终端的发布工作。

2.6.4 容器与 SDN

随着容器部署规模的增大，跨主机、跨网络的容器迁移成为常态。而容器更多地关注于轻量化本身，对于网络架构并没有太多关注。过于复杂的体系结构和管理过程，容易让整个容器网络和系统陷入不可控的非稳定状态。通过 SDN 和 Overlay 网络结合，将控制转发分离、集中控制管理理念应用于容器网络，还可以最大程度增强容器网络的弹性伸缩能力和简化网络管理。

另外，SDN 与容器的配合，是相得益彰、互相促进的。业界的 SDN 控制器和系统一般都比较庞大，安装、运行都极为复杂。通过 Docker 技术，能够实现 SDN 控制器的轻量级快速部署、安装、运行。

三、容器应用

3.1 容器技术应用场景

3.1.1 PaaS 平台建设

最早的 PaaS 平台方案初步解决了很多客户对于应用弹性的需求，但是在容器技术之前，构建一套 PaaS 平台面临着组件多、量级大、改造成本高等挑战，而且对于运行在不同 PaaS 平台上的应用，很难避免应用对平台的深度依赖。譬如，不同的 PaaS 平台对弹性、高可用、性能、监控、日志、版本更新等的实现方式不同，则对其上应用的架构要求也不同；另外，在编程语言和技术栈方面，也会导致应用对平台供应商的深度绑定。总之，在传统 PaaS 平台面前，我们在开发应用时不得不配合平台的要求。

而容器技术的出现，很好的解决了上述问题。容器是以应用为中心的虚拟化

环境，与编程语言、技术栈无关，比传统 PaaS 灵活；对应用的支撑也比底层平台多，可以发挥微服务架构的优势。同时，容器是基于轻量级虚拟化的技术，天生具有高密度的特性，可以更加高效地使用资源。

3.1.2 软件定义数据中心

互联网和海量数据正以前所未有的增长趋势冲击着整个数据中心行业，随着互联网、电子商务、社交网络、移动办公等互联网应用的迅速发展，传统数据中心逐渐难以满足新业务的发展需求，传统数据中心面临不匹配业务的灵活变化、能耗高、运维难、密度低、不满足云和虚拟化弹性伸缩场景，也面临着应用的快速、批量、移动部署慢等问题。

软件定义数据中心负责将存储、计算、网络资源依据策略进行自动化调度与统一管理、编排和监控，同时根据用户需求形成不同的服务并提供计费等功能。容器技术可充分利用底层的各项计算、存储和网络资源，灵活构建容器应用，实现具备应用轻量级的容器承载能力、应用集群的松耦合和资源动态弹性伸缩的能力、实现可视化运维和自动化管控的能力、实现平台自动化部署和升级的能力，从而解决了容器平台对基础设施资源调用的需求，容器平台将数据中心转化成为一个更加灵活高效的业务应用平台，其开放性和兼容性契合了数据中心对异构、大规模、可移植、互操作等方面的需求，容器技术为云计算的实施提供了强有力的支撑。

3.1.3 容器即服务

传统 IaaS 在应用过程中面临运维方面的问题，传统 IaaS 服务没有从根本上加速企业内部的开发运维效率，更多的主要是体现对于 IT 部门的技术优化和提升一定的运维能力，运维和开发人员之间依然存在传统 IT 手段同样的沟通成本。

容器技术三个方面的优势可以有助于解决传统 IaaS 面临的问题。首先，容器的本质是一种操作系统级别的虚拟化，启动一个应用容器其实就是启动一个进程，因此使得容器占用空间小、资源利用率高、本身非常轻，执行起来效率较高。这些是容器技术与传统虚拟机技术的最大差别。其次，容器技术使用镜像方式能够将应用程序和它依赖的操作系统、类库以及运行时环境整体打包，统一交付，使得运维压力大大降低。最后，容器技术与底层所使用的平台无关，容器可以在 Linux 平台各发行版上兼容，这意味着应用架构一旦转换为容器化并且迁移部署之后，就可以在任何云平台之间无缝迁移。

3.1.4 持续集成和发布（CI/CD）

传统软件架构特性是单体应用，开发周期至少以月为单位进行发布和升级，代码一般使用一种语言开发，不同的组件紧耦合，经常依赖于公共的库，部署周期以月为单位，部署依赖人工操作，组件版本复杂，操作风险高，时间管理成本均居高不下。

将容器技术引入到开发和运维环节具有以下几个方面的优势：一是提供了交付环境一致性。从开发到运维的工作流程中，由于基础环境的不一致造成了诸多问题，但通过使用容器技术在不同的物理设备、虚拟机、云平台上运行，将镜像作为标准的交付物，应用以容器为基础提供服务，实现多套环境交付的一致性；二是提供了快速部署。工具链的标准化将 DevOps 所需的多种工具或软件进行容器化，在任意环境实现快速部署。三是轻量和高效。与虚拟机相比，容器仅需要封装应用及相关依赖文件，更加轻量，提高资源利用率。因此企业通过容器技术进行 DevOps 的实践，可较好的缩短软件发布周期，提升产品交付迭代速度，提高生产效率。

3.2 容器技术优势分析

3.2.1 从运维者维度看容器技术优势

快速部署：

使用容器能够利用镜像快速部署运行服务，能够实现业务的快速交付，缩短业务的上线周期，极大地方便运维人员的上线部署工作。

弹性伸缩：

使用容器技术，当遇到高并发、高流量的大活动，容器做到根据业务的负载进行弹性扩容，以提供更好的服务。当访问量降低后，容器平台能够自动缩容，及时释放空闲资源。

可移植性：

容器可以运行于 Linux、Unix、Window 的操作系统之上，可以利用容器将服务移植到不同的操作系统环境下运行。

轻量：

相比传统虚拟机，容器更加轻量，资源消耗更低，镜像体积更小。

高可用：

容器运行的业务通常由一组容器来提供服务，容器平台的服务发现功能可以保证容器实例的副本数量即使在某个主机宕机的情况下也能维持不变。保证服务能够正常提供服务。

资源利用率高：

容器较传统虚拟化有更低资源使用粒度，在一台物理机上可运行上百个容器服务，从而提高服务器硬件资源的利用率。

3.2.2 从开发者维度看容器技术优势

快速构建开发环境：

开发应用除去自身编码工作之外，还需要额外的数据库、缓存或消息队列等组件在本地进行测试，使用容器技术可以快速完成构建，省去了设备申请、采购的流程，简化了开发者组件安装工作，提高了开发效率。

提供一致性的开发环境：

采用容器镜像技术，只需要一次构建就可以实现在不同的项目成员之间快速复制出一套完全一致的开发环境，从而消除因环境异构而导致的不一致性，降低软件缺陷出现的概率。

同时，在开发环境和测试环境中使用同样的镜像，也能保证开发和测试环境的一致性，提前发现软件缺陷，减少对因环境不一致而导致的缺陷的调查成本。

方便开发环境版本管理：

通过对应用程序镜像的版本化管理，可以实现同一套应用程序的多版本共存，尤其是存在对某些组件多版本支持的情况下，通过容器技术，可以轻松支持该组件的多个版本。

对应用程序容器的版本化，在应用程序本身存在多版本的情况下，开发者还能在快速进行版本回溯，提高问题调查和缺陷修复的效率；在发布失败时，也能快速回滚。

3.3 容器技术与其他技术的集成

容器技术与在应用过程中需要与以下几个方面技术进行集成才能在各应用场景中进行应用。一是与 IaaS 管理平台集成。如果容器选择在 VM 中运行，就需要与 VM 的管理平台进行对接。在私有云场景下需要对接例如 OpenStack（容器

技术与 OpenStack 技术集成具体可参考附录 C)、VMware 产品；在公有云场景下，需要与公有云服务提供商，例如阿里云、百度云、AWS 等公司提供的 VM 进行集成。二是与开发工具集成。开发工具相关技术的集成是构建持续集成、持续发布以及 DevOps 环境的必须条件，目前比较常用的开发工具包括 Jenkins、Shippable（for Docker）等。三是与网络进行集成。不同的应用运行于容器集群时有网络的互通需求，因此容器技术在具体场景应用时需要与网络二层或三层功能模块进行集成，以达到互通。四是与存储管理模块集成。存储是应用数据存储的必备，不同的存储在性能、存储空间、易用度等方面差别较大，如果要支持更多的应用，容器平台需要内置外部存储管理插件，与存储平台进行集成。

四、未来发展趋势

4.1 容器技术问题分析

容器技术拥有快速扩展、灵活性和易用性等诸多优势，但在容器应用过程中会遇到以下几个方面的问题：一是兼容性方面，容器版本在快速更新中，以 Docker 相关技术为例，每隔 1-3 个月左右就有版本的升级，一些核心模块依赖于高版本内核，运维时存在版本兼容问题。二是目前容器技术没有统一的标准，例如：在容器层面有 Docker Container、Mesos Container、rkt、CRI-O 等众多容器技术产品；此外，Google 虽然在联合容器业界相关的厂商制定标准，但目前也未在容器方面进行完全的统一。三是管理容器环境和应用也较为复杂，不仅需要多类技术支撑，包括容器管理、编排、应用打包、容器间的网络、数据快照等，还需要增加对容器的监控。四是容器在应用过程中还需要考虑在容器间、容器与系统间的性能隔离，内核共享带来的安全隔离问题。五是在使用习惯角度上，容器使

用习惯会有别于主机或虚拟机，容器技术在应用过程中大部分用户需要逐步引导才能适应容器的使用方式。

4.2 容器技术未来生态图及展望

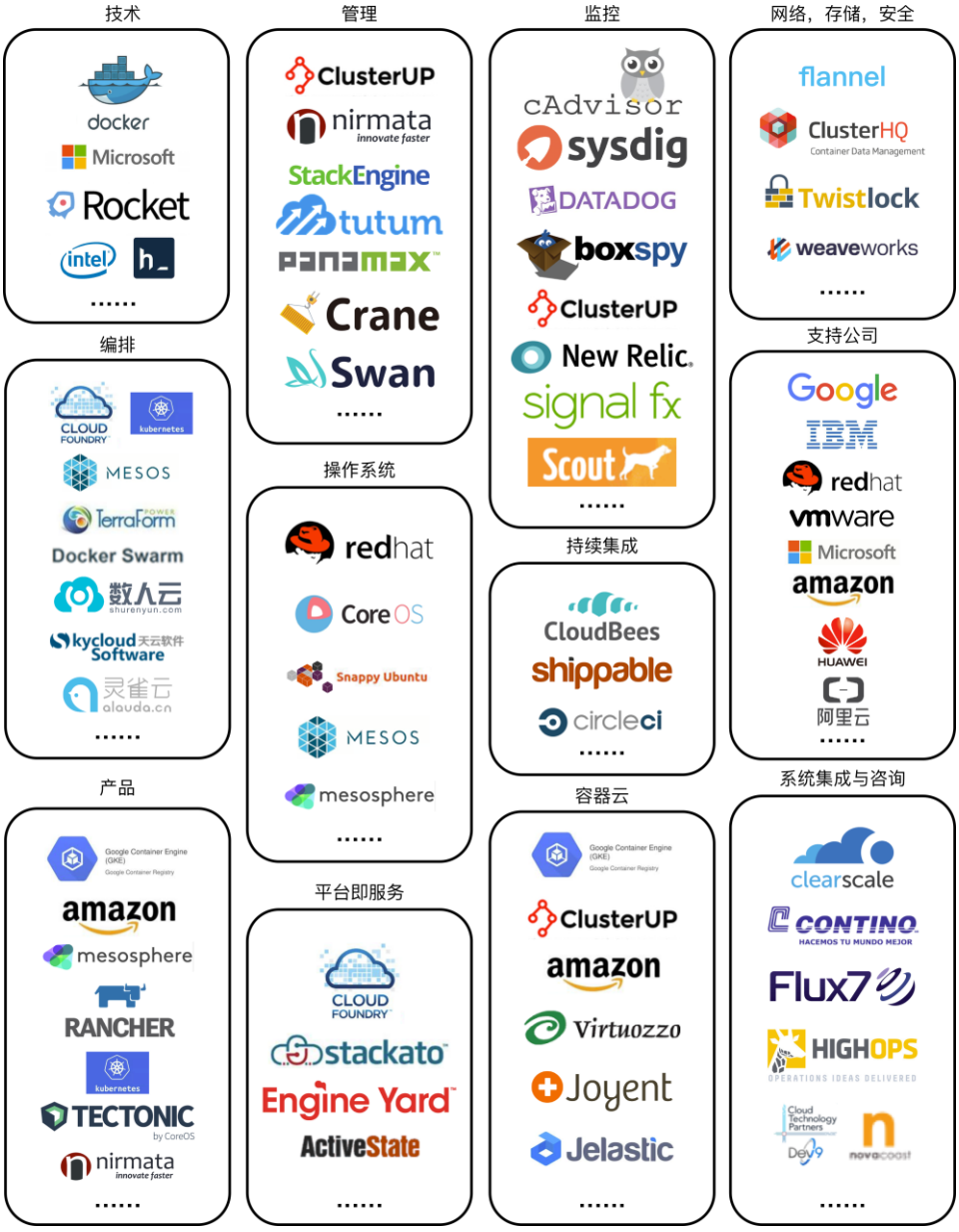


图 4.1 容器未来生态图

容器技术从 2014 年风靡全球信息技术的各个领域开始，从最初的熟悉了解容器技术的应用范围到落地容器实践过程中，业界厂商和用户都不得不思考企业级云计算架构之下，容器技术能解决什么样的问题。从 2016 年的业界发展趋势

来判断，企业用户对应用交付提出了服务化、分布式、容器化需求，通过容器技术实现 IT 应用的快速交付能力是落实企业战略目标的手段之一。

当前容器技术发展速度之快，已经超越当年 Docker 社区对容器技术的定义范畴，它承载着对下一代云计算技术变革的诉求。容器技术主要依赖的 cgroup 和 namespace 内核技术从生产实践中可满足企业用户的需求，并且创造性地定义了以应用为中心的开发模型，伴随着应用集装箱的镜像模式，可以部署到不同的系统平台并提供应用服务。当前的容器技术在不断的扩张自己的应用场景，DevOps、云平台、CI/CD、物联网服务、安全领域、金融领域和电商领域等开始慢慢探索应用实践。当前云计算厂商在众多企业容器需求中创新并实践多种多样的解决方案，容器技术也是当前云计算时代必然经历的一次历程。

正如本白皮书中详细展示的国内外生态图厂商群体一样，在每一个基础领域都在逐步形成容器生态圈，如图 4.1。本白皮书通过容器技术的分析解读帮助各行业用户能理解并参与制定各个细分市场的行业标准实践。在未来的某个时间里，容器技术实践的能力模型会被定义并评级，各行业企业会参考容器应用能力模型定义出自己的容器需求，容器技术提供商在中国开源云联盟的指导下会提供标准化容器解决方案，实现容器技术落地的双赢。中国开源云联盟容器工作组会继续肩负起国内容器技术的落地标准推广工作，并把国内的容器实践标准反馈给国际上容器社区，比如 Linux 基金会下属的 OCI 社区和 CNCF 联盟，为推动制定国际容器技术标准添砖加瓦。

附录 A OCI 和 CNCF 基金会介绍

目前行业里针对容器相关的开源技术标准化组织主要有两个：OCI 和 CNCF。其中 OCI 由 Docker 公司倡议发起，旨在定义容器运行引擎和容器应用镜像的相关技术规范；而 CNCF 由 Google 发起，目标是推动以容器为基础的云原生应用架构模式以及相关的各项技术，其中 Google 已经将 Google 自身的“DCOS”系统 Kubernetes 捐献给 CNCF，Google 每年仍会投入大量工程师全职开发 Kubernetes，但 Kubernetes 已经完全属于 CNCF 基金会，因此可以把 Kubernetes 和 CNCF 基金会的关系类比为 OpenStack 和 OpenStack 基金会的关系。

附录 B 容器开源

B.1 Docker

以 Docker 为代表的容器对应用提供了良好的兼容性，是云计算未来发展的关键技术之一。从 2013 年 Docker 推出以来，容器技术经历了四个阶段：第一个阶段是 2013 年，主要还是应用在开发工具方面；第二个阶段是 2014 年，开发者开始尝试在生产环境中支持容器，提供服务器运行时环境，标志性事件是 2014 年 6 月 Docker1.0 版本的发布以及年底推出的 Docker Machine、Swarm、Compose 三剑客；第三阶段是 2015 年，容器真正部署进入了生产环境中，并出现了各种容器平台和管理工具；第四阶段就是今年，开始实现容器的标准化和微服务化，并且开始全面发力进入企业级市场。图 B.1 是 Docker 截止今年上半年的发展历程。

此外 2016 年下半年 Docker 最大的热点就是 Docker1.12 版本的发布，该版本引入了内置编排机制，Docker1.12 通过 Docker Swarm 命令实现了容器集群，可实现一个大系统的部署。

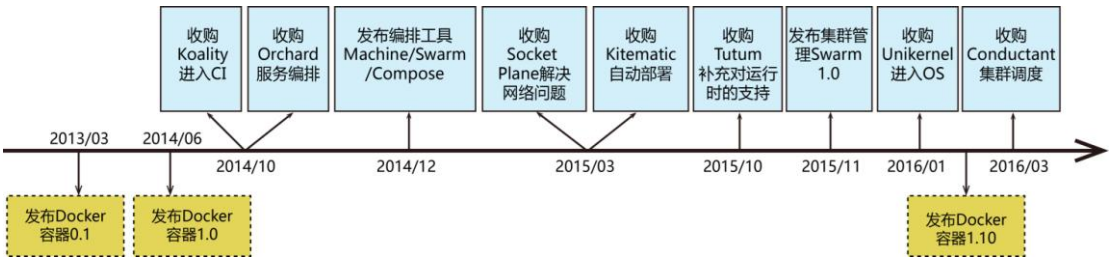


图 B.1 Docker 发展历程

B.2 Mesos

Apache Mesos 系统作为一个集群资源管理调度系统，可以用来管理 Docker 集群。同其他大部分分布式系统一样，Mesos 为了简化设计，也是采用了 Master/Slave 结构；为了解决 Master 单点故障，将 Master 做得尽可能地轻量级，其上面所有的元数据可以通过各个 Slave 重新注册而进行重构，故很容

易通过 Zookeeper 解决该单点故障问题。

Mesos 可以轻松支持 Hadoop，MPI，Spark 等计算框架，并支持在同一集群下的资源混用；同时，Mesos 和 Docker 可以做到无缝结合，通过 Marathon 可提交服务性应用，通过 Chronos 可提交批处理性应用。

B.3 Kubernetes

Kubernetes 是 Google 开源的容器集群管理系统。它构建在 Docker 技术之上，为容器化的应用提供资源调度、部署运行、服务发现、扩容缩容等整一套功能，本质上可看作是基于容器技术的 mini-PaaS 平台。

其优点包括：应用轻量级；支持公有云、私有云、混合云部署；功能模块化，各功能模块可插拔、可挂接、可组合；支持容器自动恢复、自动重启、自动复制。

B.4 Docker Swarm

Swarm 项目是 Docker 公司发布的容器管理工具，用来提供容器集群服务，目的是更好的帮助用户管理多个 Docker Engine，方便用户使用，像使用 Docker Engine 一样使用容器集群服务。

Swarm 是一套较为简单的工具，用以管理 Docker 集群，使得 Docker 集群对外以 Docker API 接口呈现，这样带来的好处是，如果现有系统使用 Docker Engine，则可以平滑将 Docker Engine 切到 Swarm 上，无需改动现有系统。Swarm 对用户来说，之前使用 Docker 的经验可以继承过来。同时 Swarm 本身专注于 Docker 集群管理，非常轻量，占用资源也非常少。

B.5 Hyper

Hyper 是一家中国国内的开源软件创业企业。Hyper 的主要项目包括：

——HyperContainer：Linux 上的虚拟化容器，结合了虚拟化和容器两者的优势，其特点可以概括为“Fast as Container, Secure as VM”；

——runV：OCI 认可的虚拟化容器实现，受到了行业的广泛支持，其中包括华为，Google，IBM，Intel，思科等等；

——Hypernetes：融合了 Kubernetes 和 OpenStack 的多租户、安全的 Kubernetes 版本，可以用来提供公有容器云服务。

B.6 Harbor

Harbor 是由 VMware 中国研发团队负责开发的开源企业级容器 Registry 项目。Harbor 可帮助用户迅速搭建企业级的 Registry 服务。它提供了简单易用的管理图形界面,基于角色的访问控制(Role Based Access Control)、镜像远程复制（同步）、AD/LDAP 集成，以及审计日志等企业用户需求的功能，同时还原生支持中文，深受中国用户的喜爱。

该项目推出仅仅 5 多个月，在 GitHub 获得了较多的点赞和下载数，社区积极参与开发,有 2/3 的代码贡献着来自全球各地。Harbor 采用商业友好的 Apache 2 许可协议，许多厂商和用户已经把 Harbor 开源项目集成到了自己的解决方案中，大大推动了容器应用在国内的普及。项目地址：https://github.com/vmware/harbor。

B.7 Open DC/OS

Open DC/OS（https://dcos.io）是 Mesosphere 公司发布的商业 DC/OS（DataCenter Operating System）的开源版本，使用 Mesos 作为其内核。DC/OS 以现代操作系统为设计原型，旨在为数据中心提供一个开箱即用，以及和现代操作系统一样方便易用的接口：CLI、GUI、软件包管理工具、服务发现等。

Open DC/OS 的架构也和现代操作系统极为相似。以 Mesos 为内核，拥有一

个 Marathon 作为 meta-framework，其作用相当于现代操作系统中的 init 或 systemd，包含容器编排、服务发现、网络、存储、大数据等一系列开源或商业的解决方案，并可通过一键部署的方式安装这些服务。

Open DC/OS 合作伙伴包括：HP、Microsoft、Oracle、IBM、EMC 等，社区主要的代码贡献者现在主要为 Mesosphere、IBM 等。

附录 C 容器和 OpenStack 的关系

OpenStack 致力于物理资源的编排调度，容器可以可以看作轻量级的计算资源，所以 OpenStack 欲将容器治于帐下。先有 NOVA DOCKER DRIVER 的先行者，再有 NOVA HEAT 插件，但是这些项目都有缺陷没法实现对容器的完美调度编排，乃至最新 Newton 上的 Magnum 项目，支持 Kubernets、Mesos、Docker Swarm 集群，OpenStack 正在一步步接近对容器编排调度的目标。

反过来，容器致力于云计算时代新式应用的打包封装和运行管理，OpenStack 本身就是一个非常典型的云应用，天生微服务化和组件化，先后有多家社区和公司宣布将对 OpenStack 进行容器化发布，预计不久的将来，容器仓库中就将出现各版本的 OpenStack 镜像。这对 OpenStack 本身的发展也是一个有利的助推器，将享受容器化带来的种种便利，如一键实现部署更新，集群高可用性，等等。容器和 OpenStack 将形成你中有我，我中有你，齐头并进的良性发展局面。

附录 D 容器应用案例

D.1 容器技术在金融行业解决方案

（本案例由北京数人科技有限公司提供）

案例介绍：

金融客户业务系统数量众多，环境管理问题日益突出。Docker 作为开源容器虚拟化技术，具有轻量级、标准化的特点，发展势头迅猛。由 Docker 发展到容器云，某金融客户的目标是建设有效的环境治理机制，规范环境的使用过程。同时，针对各业务系统运维需求，从交付、部署到监控等各个方面实现标准化运维，做到自动化运维。

需求分析：

项目需求具体包括：

——基于容器云形成有效的环境治理方案；

——实践 DevOps，建立高效的自动化软件研发平台；

解决方案：

客户结合容器及分布式资源调度技术，构建了轻量级云平台，实现统一的资源管理、统一的软件发布管理、统一的软件监控平台和统一的日志中心。

目前容器云平台已完成开发、集成、仿真、预发布、生产五大环境建设；实现界面化部署和细粒度的资源分配；做到对集群、主机、进程的多层监控体系。该平台上已发布高效研发质量平台和业务监测平台，实现应用的稳定运行。

在容器技术落地的过程中，建设云平台只是个开端，在使用云平台的过程中，

对软件交付和开发的技术又提出了新的要求——那就是云原生应用。客户对应用进行了无状态改造，采用前后端分离的轻架构开发模式，最终演变为微服务架构。

D.2 容器技术在能源行业解决方案

（本案例由北京数人科技有限公司提供）

案例介绍：

某国有大型能源行业 IT 基础架构部署方式上采取按项目集中式部署。因此软硬件紧密耦合，导致业务部署周期长、资源利用率低。伴随物联网和互联网应用的快速发展，又推出了智能电表、手机买电客户端等互联网特点的应用，并且大力引入大数据服务。而基础架构支撑业务负载变化能力不足，动态扩展和回收能力无法满足业务全天候弹性需求，运维复杂度和成本居高不下的问题凸现。容器改造后的系统提供了 PaaS 平台技术支撑，在原有容器云产品的基础上与 OpenStack 进行了深入集成，提供了完整的云环境治理管控平台能力，在一个系统界面下就可以完成所有操作，不仅提高了国网的 IT 管理运维能力，也提升了国网信息化项目交付的自动化、标准化水平。

需求分析：

——实践 DevOps，建立高效的自动化软件研发平台

——统一技术路线，规范软件开发过程，优化资源配置

——落地微服务架构，利用先进架构促进业务创新

——基于微服务架构，建设业务监测平台，监测业务流程点，保障安全运行

解决方案：

一、与 OpenStack 集成容器集群管理

通过 OpenStack Magnum 组件，调用 Heat 编排模板，创建虚拟资源环境，调用 API，完成对集群的部署操作。

二、平台账户统一管理

提供多租户能力，使得每个用户可管理自己的虚拟资源、容器集群以及应用。

三、多环境、多应用的 Devops 流水线

借助云操作系统打通开发、测试和生产环境的流水线。在云操作系统环境下，开发人员提交的测试环境的代码，自动化构建成生产标准容器镜像；同时快速构建拟真的测试环境，自动部署进行应用测试，保证测试效果。测试验证后的镜像一键发布到生产环 境，并且支持灰度发布和快速回滚，大大降低了发布的风险。

D.3 容器技术应用在央视网 API 项目

（本案例由央视国际网络有限公司提供）

案例介绍：

API 项目是央视网的一个重要的应用，作为央视影音项目中的数据提供系统，承载全网对外动态数据，融合来自不同平台、不同类型的数据，以丰富的形式对外提供数据接口支持。支撑着央视网 旗下的央视网门户、央视影音、央视新闻、央视体育等众多业务。

使用容器化平台来运行 API 业务后，实现了 API 的快速部署，秒级扩容，极大的方便了大活动场景下的扩容需求。容器化较传统虚拟化颗粒度更小，提高了服务器资源的利用率，降低公司的运营成本。同时容器化平台实现 Devops 流程，持续集成、持续发布，简化了运维和开发人员的工作流程，减少了业务的交付周期，提高了业务的扩展性。

需求分析：

央视网每年都要经历多次重要活动的播出重保工作，例如阅兵、世界杯、奥运会、春晚等。活动期间系统的重要保障：“高安全、高可用、弹性扩容”。我们前期需要对业务进行大规模扩容并且在活动期间可能会遇到的资源问题做出快速响应，所以需要快速的应对这种扩展需求。我们急切需求一种快速的扩容，缩容方式，来应对活动期间的资源需求及活动后的资源回收。

解决方案：

1. API 逻辑架构和规模

逻辑构架是五层结构，分为前端、接口层缓存、接口、数据库缓存、数据库，如图 D.1。

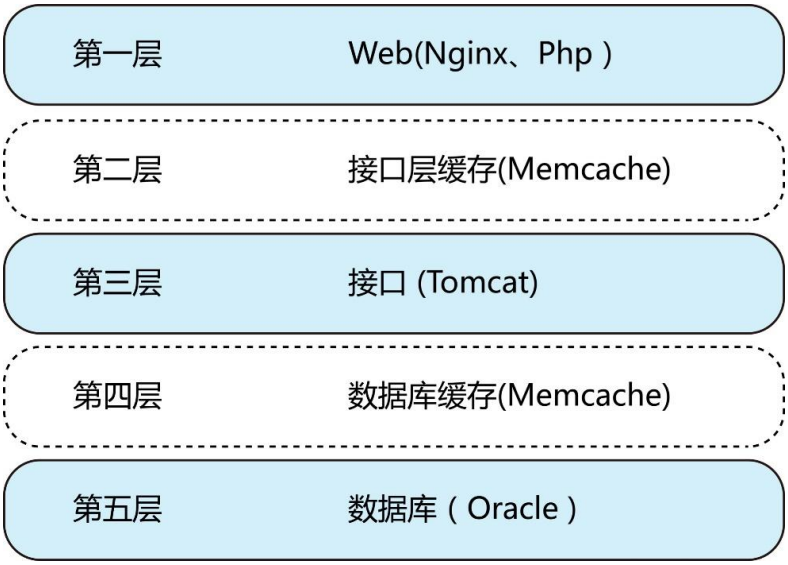


图 D.1 API 逻辑架构

2. 容器化目标

- 对业务进行解耦，将业务拆分成独立的模块。
- 秒级横向扩容能力，快速响应活动期间的突发性流量。
- 屏蔽底层异构，实现混合云无缝部署。

- 无手工操作，从代码到线上环境发布，上线时间大大缩短。
- 提供一致性的测试和开发环境。
- 灰度发布、A/B 测试降低快速发布带来的风险。

3. 容器化管理

（1） 平台管理

利用 mesos 资源管理框架及 marathon 调度框架打造出高可用的容器化管理平台，实现容器的秒级扩容，快速部署。

（2） 日志管理

利用 ELK 日志管理平台将容器日志统一的存储与分析，构建出可视化的日志分析平台。

（3） 服务注册

利用 Bamboo 和 zookeeper 构建服务注册中心，实现节点和容器的自动发现及注册，配置的自动加载，无需人工干预。

D.4 Hyper 公有容器云解决方案

（本案例由 Hyper 提供）

案例简介：

Hyper.sh 是一个基于 Hyper 虚拟化容器技术的公有容器云服务。相比基于 VM 的 IaaS,Hyper CaaS 有以下优势:

- 秒级应用部署：在 IaaS 里，VM 启动一般需要几十秒，而 Hyper CaaS 里只需要 3-5 秒；
- 跨云：不同的 IaaS 使用的 VM 镜像格式往往不同，这使得用户不能通过

统一的 VM 镜像实现通用跨云部署，而 Hyper CaaS 由于借助了 Docker 镜像，可以使用户在任何云中实现完全无缝的迁移；

- 极致弹性：在 IaaS 里，实现 VM 的弹性扩展需要分钟级，否则客户就必须实现启动 VM 等待（白花钱），而 Hyper CaaS 完全满足秒级弹性扩展，帮助应用在最快时间内扩充容量，无缝化解流量高峰；
- 成本低：相比传统 IaaS，Hyper CaaS 只需 1/3-1/2 的成本；
- 极低运维：Hyper CaaS 能够极大简化运维工作，降低管理复杂度，使得用户能够专注于业务应用本身。

相比传统 PaaS，Hyper CaaS 有一下优势：

- 安全：由于 Hyper 拥有 VM 级别的安全隔离性，相对于传统 PaaS，这最大程度保证了用户数据和应用的安全；
- 兼容性：PaaS 一直以来的弱项在于对应用的兼容性，往往需要应用代码进行修改，以适配 PaaS 的限制；Hyper CaaS 支持任何 Docker 镜像，对应用没有任何限制，也无需修改任何代码。

需求分析：

传统上，我们经常把云计算分为 IaaS，PaaS 和 SaaS 三大类。而容器技术的出现催生了 Container-as-a-Service(CaaS)服务的兴起和火爆。

现有 CaaS 一般是将容器运行在 IaaS 平台的 VM 之上，通过 VM+Container 的组合提供服务。这种架构在真正生产环境中会提高运维复杂度，不仅需要管理 IaaS 的 VM 资源，同时还要管理 VM 中的容器。

解决方案：

Hyper 这种新型的安全容器能够解决 VM+Container 的问题，使得 CaaS 达到

IaaS 的安全性兼容性，同时兼具 PaaS 的敏捷性易用性，如图 D.2。Hyper 是基于虚拟化技术，不仅具有容器的敏捷、轻量、快速启动的特性，还具备与传统 VM 相同的安全性。

此外，Hyper 的虚拟化本质还使得它能很容易和其他虚拟化生态的模块进行集成，例如分布式存储、SDN 网络系统，从而打造面向容器，原生的容器 IaaS 服务。

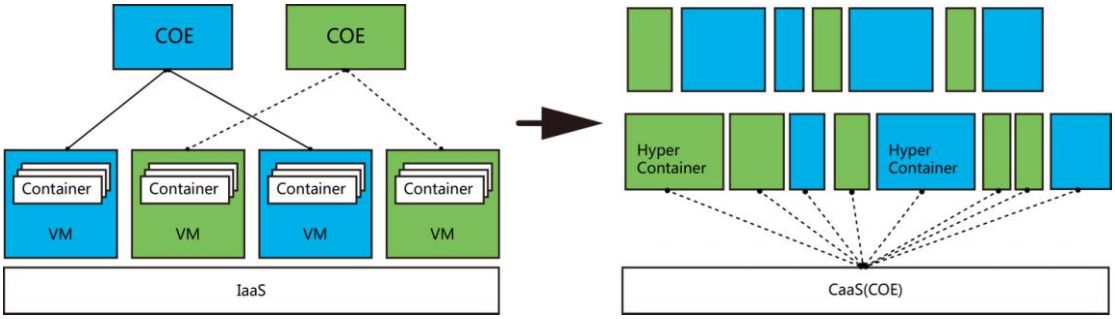


图 D.2 容器云架构对比图

D.5 容器技术在电信运营商容器项目案例

（本案例由北京天云融创软件技术有限公司提供）

案例介绍：

中国移动某省随着电信能力逐步对外开放以及互联网化的“秒杀”、“抢红包”等常态业务需求增多，原有系统存在应用环境需独立安装、应用无法动态扩展、无法多应用共享和自动扩展、应用无法快速部署迭代等诸多问题。

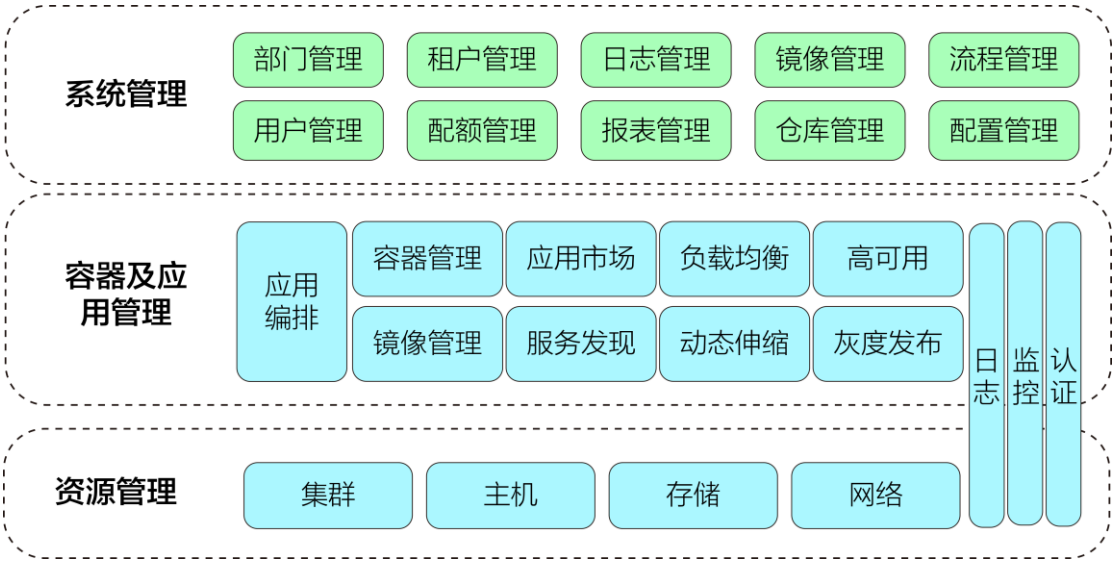
该项目采用容器技术对运营商包括掌上营业厅、网上营业厅、网上商城等业务进行优化改造，对能力开放平台、Web 门户以及其他平台门户进行整合，提高了系统在秒杀、团购等业务场景下的快速扩展能力和降低大规模集群管理复杂度的能力，实现了更快速的交付和部署、更高效的资源利用率与及时的业务响应、更自动化的运维、更高的业务可用性，改善了用户体验。

需求分析：

为完善用户第三代业务支撑系统弹性部署方案，提升系统弹性伸缩能力，本容器解决方案满足了用户业务需求，性能上实现对200万注册用户及10-50万并发用户的支撑。主要建设内容包括：应用容器化、数据库分布式改造；容器管理平台建设；云管理（资源调度）平台配合扩容改造；硬件扩容改造

解决方案：

容器管理平台整体分为三层，分别为系统管理门户、容器及应用管理平台及底层资源管理，如图D.3。



图D.3 容器管理平台功能架构图

容器管理平台核心基于Kubernetes构建，外围模块采取开源+自研开发，主要包括以下子系统：

- 应用、容器管理：基Kubernetes增强，主要提供集群管理，资源调度，应用管理，应用编排，容器扩缩容，灰度发布，应用监控等核心功能；
- 镜像管理：基于开源Docker Registry构建，支持多仓库、私有/公有仓库部署，提供镜像上传、存储、下载，分发及版本管理等功能；

- 运维与监控管理：基于 OpenStack ceilometer 等增强开发；
- 日志管理：主要面向主机、应用、容器及各种系统服务的日志统一收集、存储和检索系统，基于 ELK 方案实现；
- 租户/用户、权限管理：基于 OpenStack KeyStone 增强开发，与其他子系统包括底层 IaaS 平台集成。
- 计算资源、存储、网络管理：基于 Kubernetes on Mesos 方案实现，小规模系统部署直接使用 Kubernetes，采用 OpenStack cinder 来动态管理容器的持久化存储；网络采用 Calico 实现网络隔离、IP 管理等功能。

D.6 容器技术在教育行业应用案例

（本案例由北京凌云雀科技有限公司提供）

案例介绍：

该教育集团是中国最大的网络教育运营机构，和全国 50 多个大学、400 多家教育机构合作，为超过 50 万名学员提供在线学历教育、职业教育等各种在线课程。为了给用户提供更全面和优质的体验，该教育集团希望搭建一个教育云平台、以便合作院校、培训机构能够将其应用部署在该平台，该平台需要具备高弹性、高动态的特性，可以秒间对容器进行横向扩展，轻松应对流量高峰，满足线上业务需要。

通过容器技术，该项目在以下方面取得了显著的效果：

- 交付质量大幅提高： 从代码开发、测试到上线平滑过渡， 基本消除了因为环境的不统一而造成的事故。
- 快速部署测试和生产环境：原来部署一个新环境要花 2-5 个小时，现在基本降低为 0，每个团队都可以快速按照自己的需求部署属于自己团队的

开发、测试环境。

- 运维效率大幅提高：服务自动扩容，无需做流量预案，无需人工盯防，公有云成本降低约三分之一，极大的提高了扩容的效率。

需求分析：

该教育机构提供的业务是在线教育服务，业务模式决定了用户的访问量波动非常大，只有 IT 系统具备高可扩展性才能够给业务提供强大的支撑，这不但要求在软件架构设计上保证可扩展，同时要求运维能力也做到“可扩展”，需要运维团队能够在短时间内对服务进行快速扩容和缩容。

采用变成微服务架构后，代码的变更和部署更加频繁，经常因为测试和生产环境不一致而发生故障，严重影响了产品的用户体验。系统由多达几十个微服务组成。100 多人的研发团队同时进行十几个项目，每个项目都需要至少三个环境：开发、集成测试和 UAT 环境。搭建、维护众多的开发、测试环境运维工作量巨大。 该教育机构需要快速通过测试环境平滑迁移到生产环境，且能够做灰度发布，监控到系统异常能够快速回滚到之前的版本。

解决方案：

在互联网+的背景下，利用快速迭代和持续创新保持领先地位，这家教育机构开始转向微服务架构，使用 Docker 镜像作为交付标准，通过编写 Dockerfile，开发和运维人员共同定义编译和生产运行环境，容器云平台提供的持续集成 / 持续发布功能统一生成和发布镜像，使用容器的服务编排功能搭建各种版本集的运行环境，通过容器云平台把所有线上服务进行统一运维、管理。

D.7“互联网+”容器云解决方案

（本案例由阿里云计算有限公司公司提供）

项目介绍：

客户是互联网在线教育领先企业。该公司的业务目前处于急速扩张期，对后台系统的快速弹性、运维自动化程度、系统耦合程度都提出了较大的挑战。传统方式下，需要新建运维系统、业务系统重构等方式才能解决，而引入阿里云的容器云解决方案后，该公司 2 名运维工程师基于阿里云容器服务构建统一的开发测试运维系统并支撑新的业务系统在 1 个多月的时间上线，实现了向微服务架构的快速转型；通过持续集成和持续部署，提升了开发运维的自动化水平，提高了产品开发迭代的速度，提高了业务系统的高可用性。同时资源利用率从 30%左右提高到 60%左右，极大降低了公司 IT 成本。

需求分析：

通过对公司的业务进行分析，挖掘出该公司的业务痛点主要聚焦在以下 3 方面：a）运维自动化水平。希望能借助第三方服务完成自动化运维系统的构建，提高 DevOps 水平。b）业务系统之间的耦合度。希望通过容器来推动微服务的改造，以便更好地支持公司业务发展需要。c）资源的弹性伸缩。希望能够构建一个弹性伸缩的开发测试管理平台，既满足高峰期对资源的需求，又提高整体的资源利用率，降低公司成本。

解决方案：

通过对客户需求的分析调研，我们提出了完整的基于阿里云容器服务的微服务架构来解决应用系统紧耦合的问题，提出了基于阿里云容器服务的 DevOps 方案来解决持续集成和持续交付的问题。其中，基于阿里云容器服务的微服务架构

如图 D.4 所示：

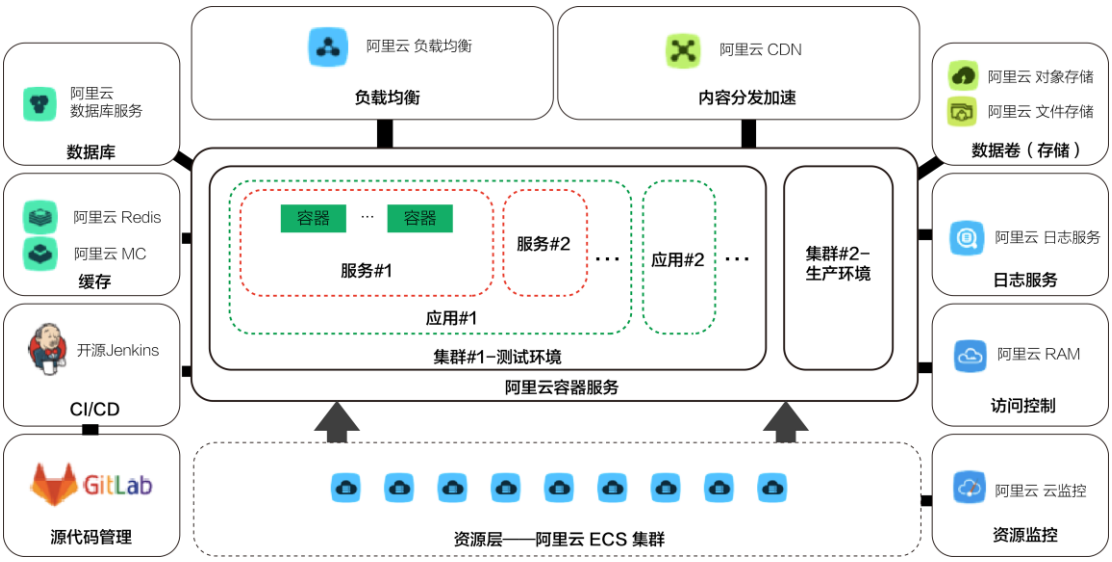


图 D.4 基于容器服务的微服务架构

为了帮助客户提升开发效率，优化开发流程，提供了基于阿里云容器服务的 DevOps 方案，通过内置的容器化 Jenkins 模版，为客户提供了从开发、构建到运维的持续集成持续交付的能力支撑。

总的来说，根据客户应用系统的实际状况，我们总结了多种模式，通过基础镜像的构建、利用 DevOps 流程自动构建应用镜像，应用的微服务化拆分等，完成了客户已有应用系统服务化改造，根据现存服务和新开发服务的不同业务属性提供了相应的服务间依赖定义和路由方案。容器服务提供了和阿里云平台产品的深度集成和优化，充分利用的阿里云平台丰富强大的能力，为客户提供了完整的端到端的和开发语言无关的微服务解决方案。