广西广电网络 GUANGXI RADIO & TV NETWORK 桂广网技专栏

浅谈 Docker 容器技术

赵冠臣 王冬妮 刘至洋 广西广播电视信息网络股份有限公司
孟振江 广西广播电视信息网络股份有限公司河池分公司

摘要：本文介绍了 Docker 容器技术与传统虚拟化技术的区别，Docker 容器的几个关键技术优势，同时简单地介绍了 Docker 容器的应用场景。

关键词：Docker 虚拟化 容器 DevOps
DOI:10.16045/j.cnki.catvtec.2019.09.021

1 Docker 技术介绍

1.1 概述

Docker 的最初版本于 2013 年由 Docker Inc. 公司发布，是基于操作系统层面的虚拟化容器，利用 Linux 内核的资源分离机制以及 Linux 内核的命名空间来建立独立运行的容器。容器间互相隔离，除了内核之外，每个容器可以有自己的库文件、配置文件、工具和应用，并且提供了良好设计的容器间通信机制，Docker 的优势使它在短短几年内成为最流行的容器解决方案，推动了基于云计算平台开发模式的变革和应用部署方式的变革。

1.2 与传统虚拟机的区别

要了解 Docker 与传统的虚拟机的区别，需要了解资源隔离和调度的机制在这两种技术中有何不同。在传统的虚拟机架构中，虚拟机监视器（Hypervisor）负责进行资源的隔离和调度。实现资源隔离的方法为：虚拟机监视器自己运行在硬件层上（1 型虚拟机），或者运行在宿主操作系统之上（2 型虚拟机），然后通过对 CPU、内存、IO 存储设备等进行虚拟化来实现资源的隔离和调度。传统的虚拟机注重的是虚拟 CPU、内存、IO 等设备，然后在其上运行客户机操作系统（Guest OS）。

而在 Docker 容器技术中，资源隔离和调度的角色由 Docker 引擎来承担，Docker 引擎利用 Linux 内核的资源分离机制以及 Linux 内核的命名空间来对容器进行隔离，即利用命名空间实现系统环境的隔离，利用 CGroups 实现资源限制，利用镜像实现运行目录环境的隔离。容器内使用的内核和宿主机是同一个内核。传统虚拟机和 Docker 容器的区别如图 1 所示。

这两个技术的主要区别如下。

（1）传统的虚拟机虚拟的是底层 CPU、内存和 IO 存储等资源的环境隔离，而 Docker 利用的是 Linux 内核命名空间特性来实现资源和环境隔离。

（2）传统虚拟机是属于更高层级的客户机操作系统间的隔离，比容器隔离度更高，而 Docker 容器是对应用程序的隔离，容器和宿主操作系统使用同一内核。

（3）传统虚拟机因为需要安装客户机操作系统，因此所需的资源消耗，如存储、CPU、内存等较多，Docker 容器更为轻量化，启动速度更快，运行效率更高。

1.3 隔离性

隔离技术出现的并不晚，早在 2000 年，FreeBSD 团队就已经将虚拟化技术导入 chroot，开发出了 jail 系统命令，可以让应用程序在沙箱中运行并相互隔离，在早期的 chroot 机制下，运行主进程可以跳出目录限制，因而 2004 年即有开发人员破解此种隔离机制。同时，容器内的应用使用宿主系统的库文件、配置文件等，一旦这些库文件和配置文件遭到破坏，所有容器均不可避免地遭受破坏，使用不当

会存在潜在风险。

而 Docker 使用的是 LXC 用户空间接口或同类型技术，利用 Linux 内核的命名空间（包括 ipc、uts、mount、pid、network 和 user）、CGroups 等特性来实现容器的隔离化，以 LXC 为例，LXC 通常被认为介于“加强版”的 chroot 和完全成熟的虚拟机之间的技术。LXC 的目标是建立一个尽可能与标准安装的 Linux 相同但又不需要分离内核的环境。这种隔离化相对与 chroot 来说隔离性更高，容器运行库文件、配置文件和工具集等和宿主操作系统可以不一致，系统风险与 chroot 相比大大减小。

但是 Docker 容器的隔离性又较传统虚拟机更低，传统虚拟机是客户机操作系统间的隔离，而 Docker 技术是应用程序容器间的隔离。

1.4 镜像继承和高迁移性

要运行一个 Docker 容器，必须指定该容器使用的镜像（image），镜像就是打包好的具有分层结构的文件，而要创建镜像，则要编写创建该镜像的配置文件（DockerFile），Dockers 通过该文件包含的指令构建一个虚拟容器，实现快速构建和快速部署。DockerFile 一般由基础镜像信息、维护者信息、操作指令和启动指令 4 部分组成，构建过程为先导入基础镜像信息并且运行为容器，执行第一条指令对容器做出修改，修改的结果创建为一个新的镜像层并提交镜像，然后以新提交的镜像为基础继续运行容器，并执行下一条指令对容器做修改，接着创建镜像层提交镜像，周而复始完成所有指令，当执行完毕，对这些镜像层进行打包即成为容器的镜像。

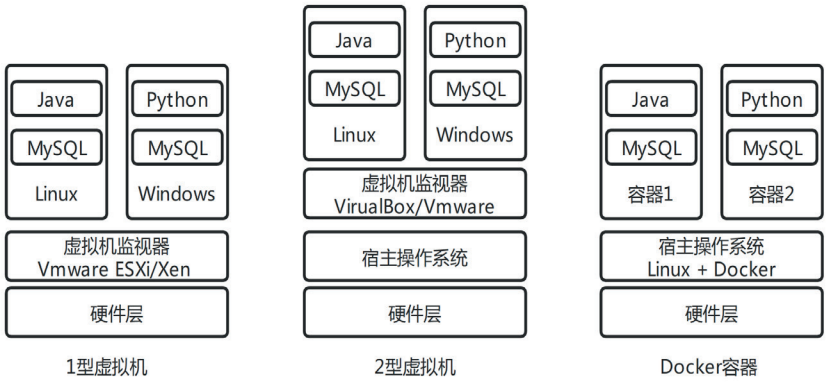

图 1 传统虚拟机和 Docker 容器的区别

由以上的构建过程可以看出来，镜像是可以有继承关系的，只要在 Dockerfile 文件开始部分导入某个镜像，由这个镜像开始去构建后生成的是新的镜像，因而生成的镜像可以再次被导入到其他 Dockerfile 文件中作为基础镜像，这就形成了镜像间的继承关系。

这个分层和打包机制和镜像继承的优点就是可迁移性高，只要对容器进行镜像打包，则无论最终的基础设施是什么，容器只要运行在 Docker 引擎之上，则最终的运行结果是一致的，因而便于构建、迁移和部署。Docker 的镜像继承和分层机制如图 2 所示。

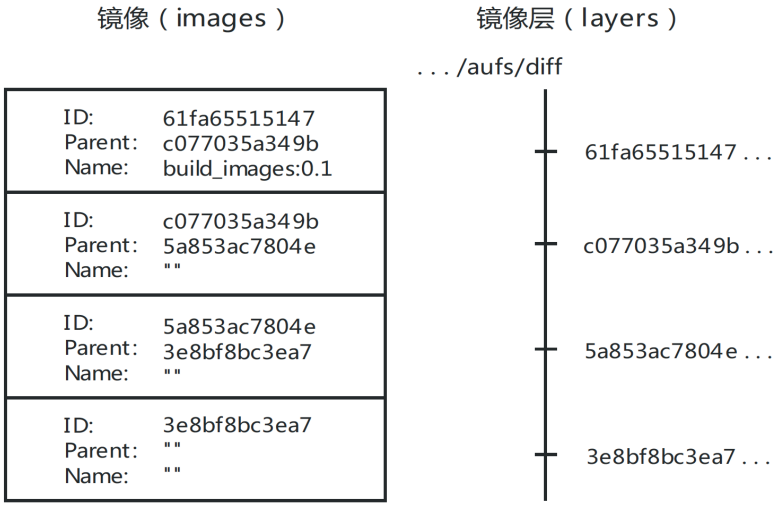

图 2 Docker 的镜像继承和分层机制

2 Docker 技术的应用场景

每个公司的信息化建设程度不同，有些公司采用项目外包的方式，无开发团队但配备运维团队；有些公司自身具备开发能力，有开发团队、测试团队和运维团队；而有些运营商性质的公司可以出售部分服务资源给客户或给服务合作商（SP）。对于以上这些不同的 IT 支撑模式，Docker 都有不同的技术架构来实现相应的解决方案。

2.1 简化运维和提高资源利用率

Docker 技术最初在很长一段时间

内只能单机运行，在 2014 年末才推出相应的容器编排工具和多机运行管理工具，对于不需要涉及容器集群的应用场景，或者刚刚接触 Docker 技术的公司来说，简单运行模式可以是一个适合的起点。

Docker 的容器特性和镜像打包的方式，相当于提供了一种标准化的服务提供方式，可用于快速部署应用的运行容器（环境）和服务，比如 MySQL、RabbitMQ、Java、Redis、MogodDB 等，或者已经打包的好如 Jenkis、GitLab 等应用，只需要写一个配置文件或者直接运行启动命令，就可以快速部署容器，马上向业务提供服务。

通过 Docker 的工具还可以对容器进行编排，比如从上述的各种镜像中取出所需的几个，并像积木一样组合起来，可以编排各个容器的启动顺序，对于运维来说非常方便。同时，DockerFile 这样的文本化配置文件具备基本的追溯能力，对于应用程序的运行环境、部署过程均体现在 DockerFile 中，通过对配置文件进行版本化管理，可以追溯这些容器部署情况的历史更改状态。

Docker 还可以提高单机的计算资源利用率，例如传统的虚拟机模式计算资源是预分配的，虚拟机上的操作系统本身还占用了一定的存储资源，而使用 Docker 本身轻量级的容器机制来运行多些应用，可以在 CPU、内存和 IO 存储上进一步压榨硬件资源，可极大地减少 IT 基础设施的投入成本。

2.2 DevOps

开发团队和运维团队在应用的运行环境上的目标往往不同，开发团队想要的是更符合应用需求、具有某些特性的运行环境，应用可以基于这个特性快速开发相应的业务功能；而维护团队想要的是最稳定的运行环境，使系统宕机时间最短，简言之，开发更偏向业务，运维更偏向基础设施。而缺乏沟通会加剧事态的严重性，开发团队不一定明确环境的变化，而运维团队不一定清楚开发团队在做什么。随着 Docker 技术的出现和该技术所带来的优势，结合自动化手段的介入，使得开发和运维的理解差异变小了，这正是 Docker 能够在短短几年内改变了软件的开发模式的原因，Docker 通过以下几个优势来实现。

（1）Docker 可以做到“一次构建、到处运行，一次配置、任意运行”，这与开发者的景愿——“一次开发、到处运行”相契合。使用容器技术，应用的运行环境在开发阶段、测试阶段和部署阶段没有任何不同。

（2）Docker 镜像在构建的过程中，完整记录了应用程序运行的环境的变量、服务、组件的详尽版本和配置过程，同时分层和继承机制，以及应用和环境打包在一起的方式便于构建、迁移和部署。

（3）Docker 容器较传统虚拟机更轻量和高效，除此之外还解决了底层基础环境的异构问题，无论是物理设备、传统虚拟机还是云计算平台，只要能运行 Docker，最终应用都以容器的形式提供服务。

DevOps 是一组过程、方法与系统的统称，用于促进开发（应用程序 / 软件工程）、技术运营和质量保障（QA）部门之间的沟通、协作与整合。强调的是结合自动化的工具协作和沟通，完成应用软件整个生命周期的管理，从而更快、更频繁地交付更稳定的软件。DevOps 的定位如图 3 所示。

基于上述优势，特别是镜像打包和高迁移性的优势，Docker 是实现 DevOps 最合适的工具之一，并使 DevOps 成为时下最热门的软件方法论，DevOps 弱化了开发团队和测试运维团队之间的界限，通过自动化的手段来对软件开发交付部署的过程进行优化，降低重复工作的成本、降低人工误操作的概率，提升故障定位的效率，缩短故障恢复的时间，并形成一种良性循环的机制，不断促进 IT 交付的速度和质量，从而提升研发、测试和运维的总体效率。DevOps 流程图如图 4 所示。

2.3 简化版本的 PaaS

PaaS 是容器即服务（Platform as a Service），意思是平台即服务，把服务器平台作为一种服务提供的商业模式。这些服务器平台包含诸如 MySQL、RabbitMQ、Java，并包含了集成工具和部署工具等等，购买这些服务的企业客户只需要关注自身需要哪些服务器平台，以及如何对业务进行开发而不用关心这些服务是如何提供的——他们由服务商提供，作为客户无需关心底层的网络架构及服务间的对接方式。

而对比 Docker 技术的标准化和容

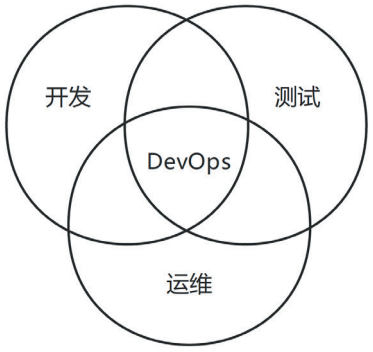

图 3 DevOps 的定位

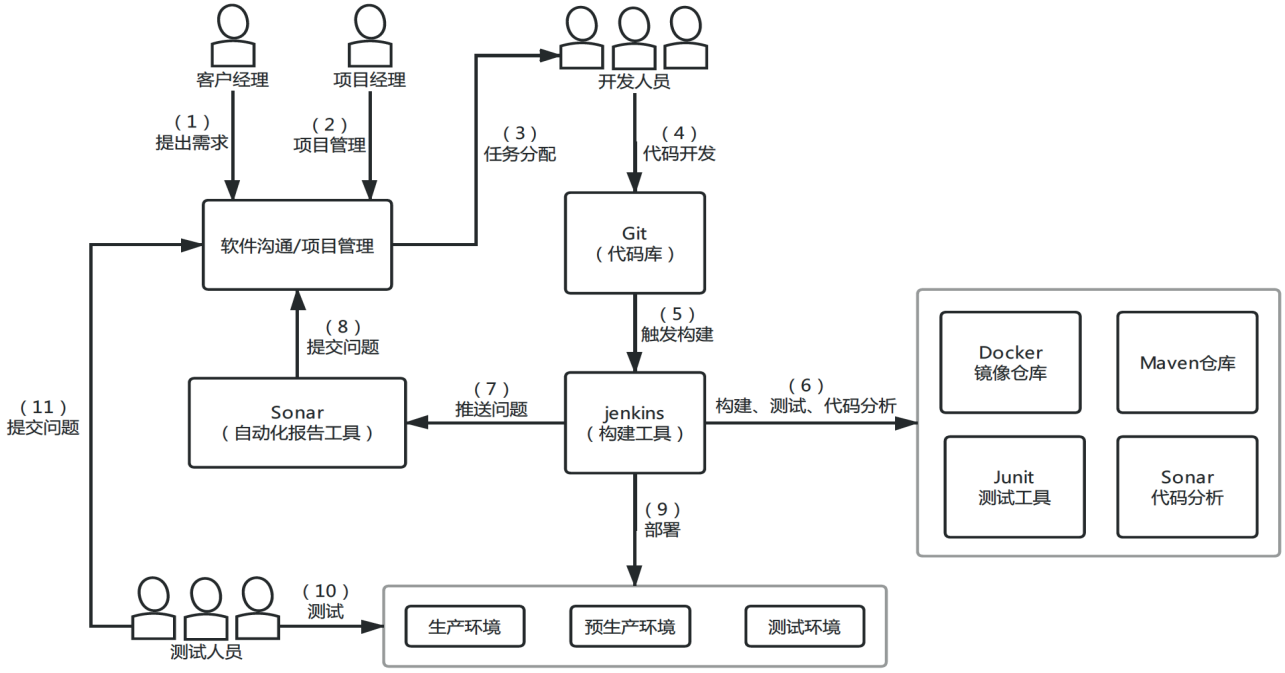

图 4 DevOps 流程图

器化，其在某些方面和 PaaS 是一致的，Docker 就有 MySQL、RabbitMQ、Java 等镜像，结合一些自动化的工具，也可以快速进行集成和部署。但是 Docker 又不完全是 PaaS，因为要达成集成和部署，企业自己还需进行相对底层的服务配置，比如服务编排、容器间互联规划和配置等，因此 Docker 可以视为某种程度的 PaaS，但并不完全是 PaaS。

现在，称通过提供 Docker 等容器的商业模式是 CaaS，意为容器即服务 (Containers as a Service)，相比传统的 PaaS 优势是可以利用大量的镜像来自定义所需服务。比如目前流行的 Kubernetes 是 Google 开源的容器集群管理系统，构建于 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，而其他大型的云计算公司提供容器服务，并且逐渐成为云计算服务提供的“标配”，可见 Docker 容器对 PaaS 或者 CaaS 商业模式的影响之大。

2.4 Docker 应用的缺陷

当然 Docker 容器还是属于年轻的生力军，在涉及一些历史遗留项目或系统来说仍然会受到某些方面的限制，比如 Docker 引擎的安装要求是 Linux 内核版本号不低于 3.10 版本，因而对于一些使用内核版本较低的应用无法迁移到 Docker。某些旧版本的应用程序不能通过 CGroups 配置来正确识别资源限制，当部署应用时，如果对容器进行资源限制，则该容器很容易超过限制策略而被 Docker 引擎回收资源，导致无法提供服务。比如 Java 直到 Java SE 8u131 或者 JDK 9 版本之后，启动 JVM 的特定参数才能正确识别容器内的资源限制。这类情况下，由于应用存在兼容性问题，也是不适合迁移到 Docker 容器中的。

3 结论

对企业来说，由于 Docker 容器技术的优点可以极大地对应用的执行环境进行打包，不仅方便迁移，同时可以反过来降低 IT 基础设施的复杂性，应用程序隔离特性还可以提高应用之间的安全性。同时配合自动化的工具（这些工具就可以方便地从容器中运行），可以极大地促进企业 IT 交付的速度和质量，因而企业可以利用这个优势来优化自己的 IT 基础设施改造和 IT 支撑团队改造。对于广电这类提供部分能力给服务提供商（SP）的场景，则可以提供传统虚拟机来隔离 SP，然后提供 Docker 引擎环境和私有镜像仓库，让 SP 自由组合所需的镜像服务，在增加 SP 满意度的同时可以提高硬件资源利用率，并简化运维团队的维护内容。

参考文献

[1]Wes Felter,Alexandre Ferreira,Ram Rajamon,Juan Rubio.An Updated Performance Comparison of Virtual Machines and Linux Containers.2014.

[2] 丁海斌，崔隽，陆凯．基于 Docker 的 DevOps 系统设计与实现 [J]. 指挥信息系统与技术 ,2017,8(3). CATV