Docker 的起源、发展现状及对软件工程的影响

陈成

(江苏大学 计算机科学与通信工程学院，江苏 镇江 212000)

The origin of Docker, the status quo of development and the impact on software engineering

CHEN Cheng

(School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang, Jiangsu 212000, China)

Abstract: Docker has been very hot since 2013, has been the support of the majority of users in various scenarios have been widely used, especially for the impact of software engineering is particularly far-reaching. This article introduces the origin of Docker, summarizes the development status of Docker, and analyzes the impact of Docker on software engineering. Emphasis on the revolutionary impact of Docker on test technology.
Key words: Docker; container; software engineering; test technology

摘要：Docker 自 2013 年以来非常火热， 得到了广大用户的支持，在各个场景得到了广泛的应用，特别是对软件工程的影响尤为深远。本文介绍 Docker 的起源，总结 Docker 的发展现状，并分析 Docker 对软件工程的影响。着重分析了 Docker 对测试技术革命性的影响。
关键字：Docker；容器；软件工程；测试技术

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口。

本文介绍 Docker 的起源，总结 Docker 的发展现状，并分析 Docker 对软件工程的影响。

1 Docker 的起源

Docker 是 PaaS 提供商 dotCloud 开源的一个基于 LXC 的高级容器引擎，源代码托管在 Github 上, 基于 go 语言并遵从 Apache2.0 协议开源。

Docker 自 2013 年以来非常火热，无论是从 github 上的代码活跃度，还是 Redhat 在 RHEL6.5 中集成对 Docker 的支持, 就连 Google 的 Compute Engine 也支持 docker 在其之上运行。

一款开源软件能否在商业上成功，很大程度上依赖三件事 - 成功的 user case(用例), 活跃的社区和一个好故事。 dotCloud 自家的 PaaS 产品建立在 docker 之上，长期维护且有大量的用户，社区也十分活跃，接下来我们看看 docker 的故事。

- 环境管理复杂 - 从各种 OS 到各种中间件到各种 app, 一款产品能够成功作为开发者需要关心的东西太多，且难于管理，这个问题几乎在所有现代 IT 相关行业都需要面对。

- 云计算时代的到来 - AWS 的成功, 引导开发者将应用转移到 cloud 上, 解决了硬件管理的问题，然而中间件相关的问题依然存在 (所以 openstack HEAT 和 AWS cloudformation 都着力解决这个问题)。开发者思路变化提供了可能性。
- 虚拟化手段的变化 - cloud 时代采用标配硬件来降低成本，采用虚拟化手段来满足用户按需使用的需求以及保证可用性和隔离性。然而无论是 KVM 还是 Xen 在 docker 看来,都在浪费资源，因为用户需要的是高效运行环境而非 OS, GuestOS 既浪费资源又难于管理, 更加轻量级的 LXC 更加灵活和快速
- LXC 的移动性 - LXC 在 linux 2.6 的 kernel 里就已经存在了，但是其设计之初并非为云计算考虑的，缺少标准化的描述手段和容器的可迁移性，决定其构建出的环境难于迁移和标准化管理(相对于 KVM 之类 image 和 snapshot 的概念)。docker 就在这个问题上做出实质性的革新。这是 docker 最独特的地方。

面对上述几个问题，docker 设想是交付运行环境如同海运，OS 如同一个货轮，每一个在 OS 基础上的软件都如同一个集装箱，用户可以通过标准化手段自由组装运行环境，同时集装箱的内容可以由用户自定义，也可以由专业人员制造。这样，交付一个软件，就是一系列标准化组件的集合的交付，如同乐高积木，用户只需要选择合适的积木组合，并且在最顶端署上自己的名字(最后个标准化组件是用户的 app)。这也就是基于 docker 的 PaaS 产品的原型。

2 Docker 的发展现状

根据 ClusterHQ 联合 DevOps 公布的一项关于 container/Docker 的使用调查报告，报告内容涵盖用户接受程度，应用场景，以及面临的痛点和阻碍，尤其在数据管理和可靠/持久性存储等方面的诉求。我们可以了解到 Docker 的发展现状。

总体来看，Docker 的发展现状大致如下：

- 关注度: 94% say used, Docker is the most popular choice (90+% choice)
- 使用度: 38% use in production (!). 70% in Dev&op; 65% to use
- 应用环境: 73% in VM env: Vmware (#1: 31%), cloud (20%), KVM (16%)
- IT 架构: Private DC (57%), AWS (52%), Digital Ocean (22%)
- 管理工具: Docker swarm (50%), Kubernetes (38%), Mesos (35%)
- 主要障碍: Security: 61%, Data Management : 53%, Networking: 51%, Persistent store: 48%
- 70% 用户希望运行 stateful services

详细内容如下：

(1) Container 的选择

毫无疑问 Docker, LXC(Linux 自带的 也是 Docker 底层所依赖的容器)位列第二，新型的 Rocket 位居第三。调查情况，如图 2-1 所示。

，

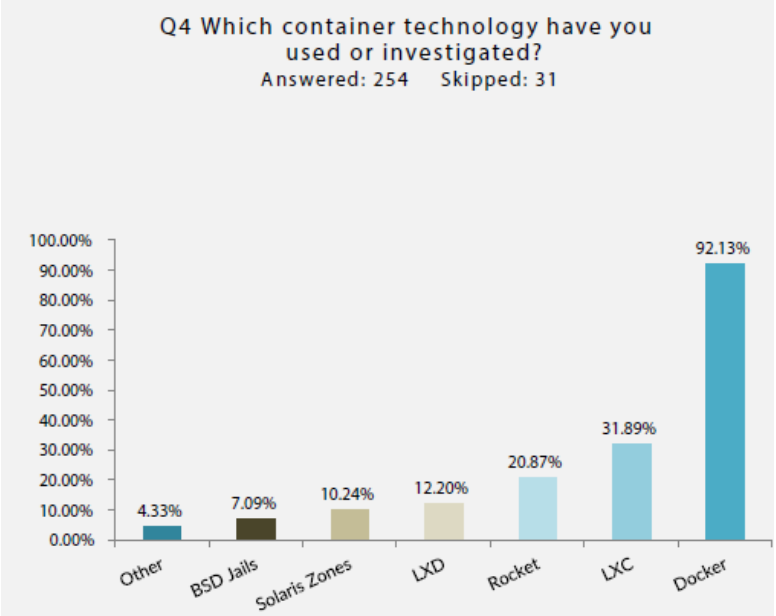

图 2-1 容器选择汇总图

(2) 应用场景

大多数主要用于 Dev/Test 和 PoC，这个可以理解，Docker 做简化开发/测试环境，一致性方面绝对方便，深有体会。比较吃惊的是，有接近 40%已用于生产环境，这个数字还是比较震撼的。猜测主要是互联网用户用于尝鲜，快速迭代，有较为迫切的需求促使技术很快投入实用。接下来，甚至有更多的用户打算用于生产环境(65%)。调查情况，如图 2-2、图 2-3 所示。

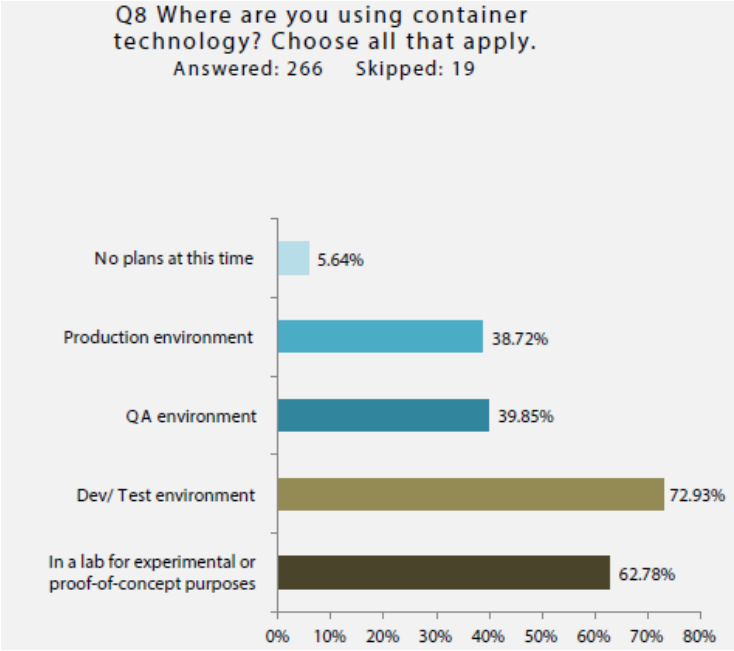

图 2-2 Docker 用途汇总图

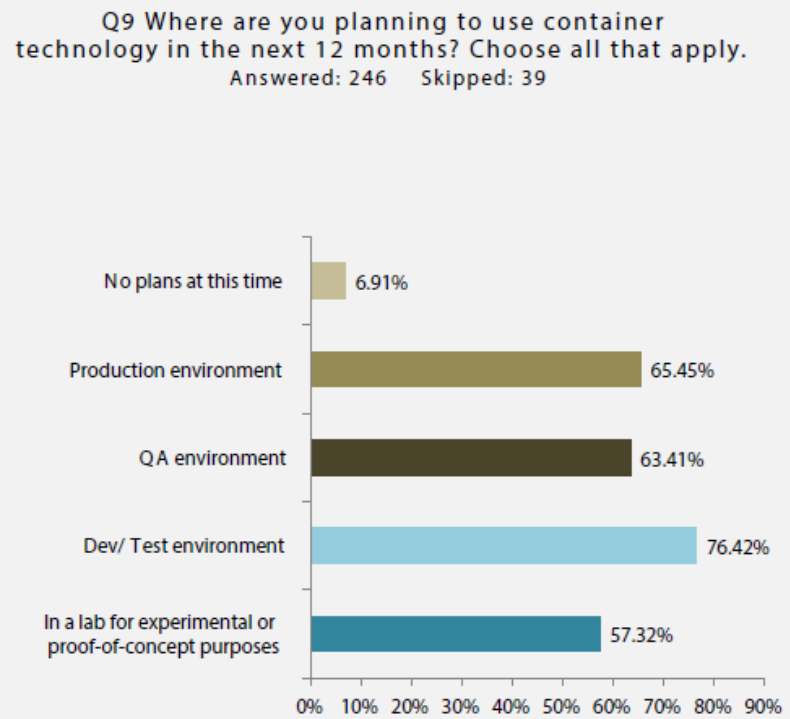

图 2-3 Docker 计划用途汇总图

(3) Docker 与虚拟化 VM

非常值得关注的话题，有虚机还会用 Docker 么？有了 Docker 还要跑在 VM 里面？

调查显示依然有高达 73%的用户会在 VM 里跑 Docker。 Dev&Test 阶段，先部署 VM 给 Dev，然后起 Docker 来开发/测试 App 看起来确实是个比较合理的模式。虚机的使用中，VMWare 依然遥遥领先(30.9%)，其次是 Cloud 环境（19.7%） 和 KVM(15.6)。估计这个次序可能在未来 1-2 年会发生不小的变化，如果 VMWare 还比较保守的话 。

其次，受调查者超过半数是在自己的数据中心里运行，其次是 AWS 。

关于 Docker 的管理工具: 自身的 Swarm 目前仍稍微领先，但 Kubernetes 和 Mesos 已经紧追其后。调查情况，如图 2-4、图 2-5 所示。

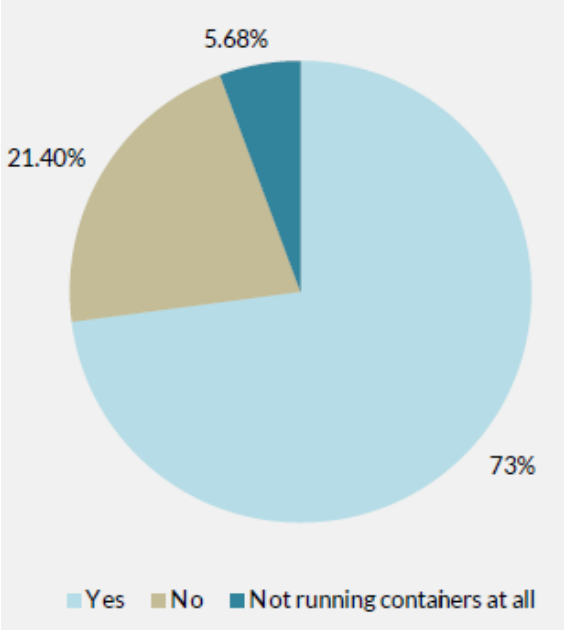

图 2-4 用户选择 Docker/VM 比例图

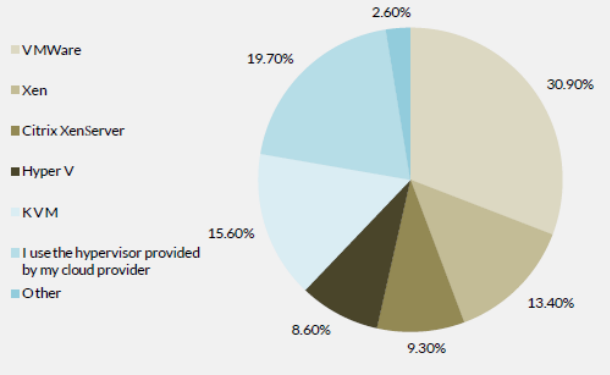

图 2-5 Docker 管理工具选择比例图

(4) 障碍和挑战

安全是心头之痛，后面紧追的还包括数据管理，网络以及可靠存储等。调查情况，如图 2-6 所示。

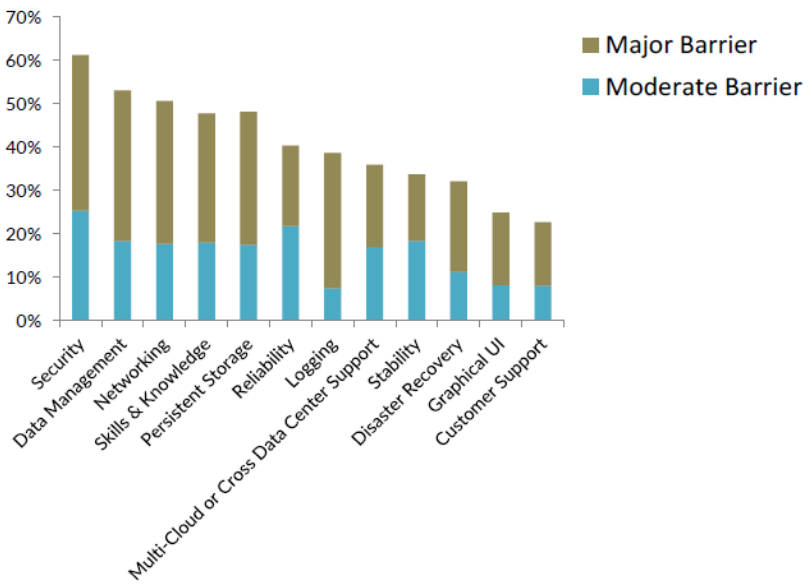

图 2-6 Docker 发展障碍分布图

(5) 数据持久化

ClusterHQ 比较关心存储方面。尽管 Microservice 推崇无状态应用，然而 70%的用户还是想在 Docker 里运行可持久性的数据例如数据库。可能这就是理论和现实间的折中，如果有有效，成熟的底层方案，相信开发者还是希望用比较熟悉的方式。调查情况，如图 2-7 所示。

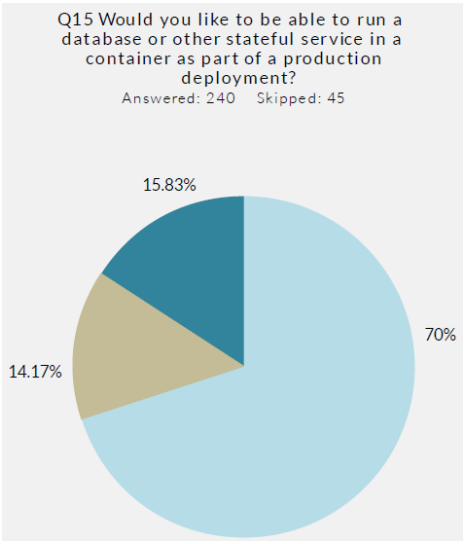

图 2-7 是否选择在 Docker 里运行可持续性数据比例图

3 Docker 对软件工程的影响

3.1 Docker：分布式系统的软件工程革命

Docker 是一个软件系统，实现了一种称为“集装箱”的概念。集装箱类似 Google 机群管理系统 Borg 中的包（package）。

集装箱这个想法已经在深刻地改变传统分布式系统的开发、测试和部署的流程了。传统的做法是，开发者写一个 Makefile（或者其他描述，比如 CMakeList、POM 等）来说明如何把源码编译成二进制文件。随后，开发人员会在开发机上配置并且执行二进制文件，来作测试。测试人员会在测试机群上配置和执行，来作验证。而运维人员会在数据中心里的预发布环境和产品环境上配置和执行，这就是部署。因为开发机、测试机群、和产品环境里机器的数量和质量都不同，所以配置往往很不同。加上每个新版本的软件系统，配置方式难免有所差异，所以经常造成意外错误。以至于绝大部分团队都选择趁夜深人静、用户不活跃的时候，上线新版本，苦不堪言。

而利用集装箱概念的开发流程里，开发者除了写 Makefile，还要写一个 Dockerfile，来描述如何把二进制文件安装进一个集装箱镜像（container image），并且做好配置。而一个镜像就像一台配置好的虚拟机，可以在机群上启动多个实例（instance），而每个实例通常称为一个集装箱（container）。在自测的时候，开发者在开发机上执行一个或者多个集装箱；在验证时，测试人员在测试机群上执行集装箱；在部署时，运维人员在产品环境执行集装箱。因为执行的都是同样地集装箱，所以不容易出错。

这种流程更合理的划分了开发者和其他角色的工作边界，也大大简化了测试和部署工作。

3.2 Docker 对测试技术的革命性影响

1. 更早的发现单元测试中的软件缺陷。

测试驱动开发是软件工程中一个具有里程碑意义的创新，即开发者在提交开发代码的同时也要提供对应的测试代码，在代码提交后系统会自动进行一轮自动化测试。通过 Docker 可以快速部署测试环境，可以有力的支撑自动化测试，从而确保在第一时间发现单元测试中的软件缺陷。

2. 为功能测试和集成测试提供清洁的测试环境。

很多公司由于成本问题，不得不在一个虚拟机中运行不同类型的测试任务。而这些任务在运行时往往会导致环境污染。通过 Docker 技术的隔离性，可以有效地解决测试环境的污染问题。

3. 让测试团队和客户丢掉冗长的配置文档。

开发转测试时往往带有较长的环境部署文档，而在这些文档中往往存在部署过程跳步的问题，测试团队很难一次准确的将环境部署成功。而现在可以通过 Docker 镜像

将配置环境的过程简化，测试团队省去了查阅文档的过程，只需要基于开发团队提供的 Docker 镜像就可以轻松的配置测试环境。

4. 便于复现客户报告的软件缺陷。

当客户使用软件发现缺陷时，可以将其所使用的环境打包成镜像提供给开发团队。开发团队通过镜像即可获取与客户一致的软件环境。

5. 通过 Dockerfile 可以梳理好测试镜像制作的流程。

如果流程步骤需要微调时(如将安装 gcc3.4 改为安装 gcc4.3)，可以将 Dockerfile 中对应的信息进行修改并重新创建新的镜像，不必手动重新配置运行环境。

6. 可以将成熟的测试套或测试工具通过镜像共享。

这样可以支持软件在不同 linux 发行版中成功的运行，软件提供商可以将主要精力放在完善功能上，不必投入过多时间将软件适配到不同的 linux 发行版中。

7. 优越的性能指标。

通过优于虚拟机的性能，Docker 可以提升测试效率。通过“-v”选项可以将主机的目录快速映射到容器中，可以实现测试文件的快速共享。通过“--rm”选项可以在测试完成后第一时间删除容器，以便释放系统资源。

结合 CRIU 技术，可以实现容器运行状态的保存，这项技术也是容器热迁移的基础。