01 | 小鲸鱼大事记（一）：初出茅庐

2018-08-28 张磊

01 | 小鲸鱼大事记（一）：初出茅庐
朗读人：张磊　12′11″ | 5.59M

你好，我是张磊。我今天分享的主题是：小鲸鱼大事记之初出茅庐。

如果我问你，现今最热门的服务器端技术是什么？想必你不假思索就能回答上来：当然是容器！可是，如果现在不是 2018 年而是 2013 年，你的回答还能这么斩钉截铁么？

现在就让我们把时间拨回到五年前去看看吧。

2013 年的后端技术领域，已经太久没有出现过令人兴奋的东西了。曾经被人们寄予厚望的云计算技术，也已经从当初虚无缥缈的概念蜕变成了实实在在的虚拟机和账单。而相比于的如日中天 AWS 和盛极一时的 OpenStack，以 Cloud Foundry 为代表的开源 PaaS 项目，却成为了当时云计算技术中的一股清流。

这时，Cloud Foundry 项目已经基本度过了最艰难的概念普及和用户教育阶段，吸引了包括百度、京东、华为、IBM 等一大批国内外技术厂商，开启了以开源 PaaS 为核心构建平台层服务

能力的变革。如果你有机会问问当时的云计算从业者们，他们十有八九都会告诉你：PaaS 的时代就要来了！

这个说法其实一点儿没错，如果不是后来一个叫 Docker 的开源项目突然冒出来的话。

事实上，当时还名叫 dotCloud 的 Docker 公司，也是这股 PaaS 热潮中的一份子。只不过相比于 Heroku、Pivotal、Red Hat 等 PaaS 弄潮儿们，dotCloud 公司实在是太微不足道了，而它的主打产品由于跟主流的 Cloud Foundry 社区脱节，长期以来也无人问津。眼看就要被如火如荼的 PaaS 风潮抛弃，dotCloud 公司却做出了这样一个决定：开源自己的容器项目 Docker。

显然，这个决定在当时根本没人在乎。

“容器”这个概念从来就不是什么新鲜的东西，也不是 Docker 公司发明的。即使在当时最热门的 PaaS 项目 Cloud Foundry 中，容器也只是其最底层、最没人关注的那一部分。说到这里，我正好以当时的事实标准 Cloud Foundry 为例，来解说一下 PaaS 技术。

PaaS 项目被大家接纳的一个主要原因，就是它提供了一种名叫“应用托管”的能力。 在当时，虚拟机和云计算已经是比较普遍的技术和服务了，那时主流用户的普遍用法，就是租一批 AWS 或者 OpenStack 的虚拟机，然后像以前管理物理服务器那样，用脚本或者手工的方式在这些机器上部署应用。

当然，这个部署过程难免会碰到云端虚拟机和本地环境不一致的问题，所以当时的云计算服务，比的就是谁能更好地模拟本地服务器环境，能带来更好的“上云”体验。而 PaaS 开源项目的出现，就是当时解决这个问题的一个最佳方案。

举个例子，虚拟机创建好之后，运维人员只需要在这些机器上部署一个 Cloud Foundry 项目，然后开发者只要执行一条命令就能把本地的应用部署到云上，这条命令就是：

```
$ cf push " 我的应用 "
```

是不是很神奇？

事实上，像 Cloud Foundry 这样的 PaaS 项目，最核心的组件就是一套应用的打包和分发机制。 Cloud Foundry 为每种主流编程语言都定义了一种打包格式，而“cf push”的作用，基本上等同于用户把应用的可执行文件和启动脚本打进一个压缩包内，上传到云上 Cloud Foundry 的存储中。接着，Cloud Foundry 会通过调度器选择一个可以运行这个应用的虚拟机，然后通知这个机器上的 Agent 把应用压缩包下载下来启动。

这时候关键来了，由于需要在一个虚拟机上启动很多个来自不同用户的应用，Cloud Foundry 会调用操作系统的 Cgroups 和 Namespace 机制为每一个应用单独创建一个称作“沙盒”的隔

离环境，然后在“沙盒”中启动这些应用进程。这样，就实现了把多个用户的应用互不干涉地在虚拟机里批量地、自动地运行起来的目的。

这，正是 PaaS 项目最核心的能力。 而这些 Cloud Foundry 用来运行应用的隔离环境，或者说“沙盒”，就是所谓的“容器”。

而 Docker 项目，实际上跟 Cloud Foundry 的容器并没有太大不同，所以在它发布后不久，Cloud Foundry 的首席产品经理 James Bayer 就在社区里做了一次详细对比，告诉用户 Docker 实际上只是一个同样使用 Cgroups 和 Namespace 实现的“沙盒”而已，没有什么特别的黑科技，也不需要特别关注。

然而，短短几个月，Docker 项目就迅速崛起了。它的崛起速度如此之快，以至于 Cloud Foundry 以及所有的 PaaS 社区还没来得及成为它的竞争对手，就直接被宣告出局了。那时候，一位多年的 PaaS 从业者曾经如此感慨道：这简直就是一场“降维打击”啊。

难道这一次，连闯荡多年的“老江湖”James Bayer 也看走眼了么？

并没有。

事实上，Docker 项目确实与 Cloud Foundry 的容器在大部分功能和实现原理上都是一样的，可偏偏就是这剩下的一小部分不一样的功能，成了 Docker 项目接下来“呼风唤雨”的不二法宝。

这个功能，就是 Docker 镜像。

恐怕连 Docker 项目的作者 Solomon Hykes 自己当时都没想到，这个小小的创新，在短短几年内就如此迅速地改变了整个云计算领域的发展历程。

我前面已经介绍过，PaaS 之所以能够帮助用户大规模部署应用到集群里，是因为它提供了一套应用打包的功能。可偏偏就是这个打包功能，却成了 PaaS 日后不断遭到用户诟病的一个“软肋”。

出现这个问题的根本原因是，一旦用上了 PaaS，用户就必须为每种语言、每种框架，甚至每个版本的应用维护一个打好的包。这个打包过程，没有任何章法可循，更麻烦的是，明明在本地运行得好好的应用，却需要做很多修改和配置工作才能在 PaaS 里运行起来。而这些修改和配置，并没有什么经验可以借鉴，基本上得靠不断试错，直到你摸清楚了本地应用和远端 PaaS 匹配的“脾气”才能够搞定。

最后结局就是，“cf push”确实是能一键部署了，但是为了实现这个一键部署，用户为每个应用打包的工作可谓一波三折，费尽心机。

而Docker 镜像解决的，恰恰就是打包这个根本性的问题。 所谓 Docker 镜像，其实就是一个压缩包。但是这个压缩包里的内容，比 PaaS 的应用可执行文件 + 启停脚本的组合就要丰富多了。实际上，大多数 Docker 镜像是直接由一个完整操作系统的所有文件和目录构成的，所以这个压缩包里的内容跟你本地开发和测试环境用的操作系统是完全一样的。

这就有意思了：假设你的应用在本地运行时，能看见的环境是 CentOS 7.2 操作系统的所有文件和目录，那么只要用 CentOS 7.2 的 ISO 做一个压缩包，再把你的应用可执行文件也压缩进去，那么无论在哪里解压这个压缩包，都可以得到与你本地测试时一样的环境。当然，你的应用也在里面！

这就是 Docker 镜像最厉害的地方：只要有这个压缩包在手，你就可以使用某种技术创建一个“沙盒”，在“沙盒”中解压这个压缩包，然后就可以运行你的程序了。

更重要的是，这个压缩包包含了完整的操作系统文件和目录，也就是包含了这个应用运行所需要的所有依赖，所以你可以先用这个压缩包在本地进行开发和测试，完成之后，再把这个压缩包上传到云端运行。

在这个过程中，你完全不需要进行任何配置或者修改，因为这个压缩包赋予了你一种极其宝贵的能力：本地环境和云端环境的高度一致！

这，正是 Docker 镜像的精髓。

那么，有了 Docker 镜像这个利器，PaaS 里最核心的打包系统一下子就没了用武之地，最让用户抓狂的打包过程也随之消失了。相比之下，在当今的互联网里，Docker 镜像需要的操作系统文件和目录，可谓唾手可得。

所以，你只需要提供一个下载好的操作系统文件与目录，然后使用它制作一个压缩包即可，这个命令就是：

```
$ docker build " 我的镜像 "
```

一旦镜像制作完成，用户就可以让 Docker 创建一个“沙盒”来解压这个镜像，然后在“沙盒”中运行自己的应用，这个命令就是：

```
$ docker run " 我的镜像 "
```

当然，docker run 创建的“沙盒”，也是使用 Cgroups 和 Namespace 机制创建出来的隔离环境。我会在后面的文章中，详细介绍这个机制的实现原理。

所以，Docker 项目给 PaaS 世界带来的“降维打击”，其实是提供了一种非常便利的打包机制。这种机制直接打包了应用运行所需要的整个操作系统，从而保证了本地环境和云端环境的高度一致，避免了用户通过“试错”来匹配两种不同运行环境之间差异的痛苦过程。

而对于开发者们来说，在终于体验到了生产力解放所带来的痛快之后，他们自然选择了用脚投票，直接宣告了 PaaS 时代的结束。

不过，Docker 项目固然解决了应用打包的难题，但正如前面所介绍的那样，它并不能代替 PaaS 完成大规模部署应用的职责。

遗憾的是，考虑到 Docker 公司是一个与自己有潜在竞争关系的商业实体，再加上对 Docker 项目普及程度的错误判断，Cloud Foundry 项目并没有第一时间使用 Docker 作为自己的核心依赖，去替换自己那套饱受诟病的打包流程。

反倒是一些机敏的创业公司，纷纷在第一时间推出了 Docker 容器集群管理的开源项目（比如 Deis 和 Flynn），它们一般称自己为 CaaS，即 Container-as-a-Service，用来跟“过时”的 PaaS 们划清界限。

而在 2014 年底的 DockerCon 上，Docker 公司雄心勃勃地对外发布了自家研发的“Docker 原生”容器集群管理项目 Swarm，不仅将这波“CaaS”热推向了一个前所未有的高潮，更是寄托了整个 Docker 公司重新定义 PaaS 的宏伟愿望。

在 2014 年的这段巅峰岁月里，Docker 公司离自己的理想真的只有一步之遥。

总结

2013~2014 年，以 Cloud Foundry 为代表的 PaaS 项目，逐渐完成了教育用户和开拓市场的艰巨任务，也正是在这个将概念逐渐落地的过程中，应用“打包”困难这个问题，成了整个后端技术圈子的一块心病。

Docker 项目的出现，则为这个根本性的问题提供了一个近乎完美的解决方案。这正是 Docker 项目刚刚开源不久，就能够带领一家原本默默无闻的 PaaS 创业公司脱颖而出，然后迅速占领了所有云计算领域头条的技术原因。

而在成为了基础设施领域近十年难得一见的技术明星之后，dotCloud 公司则在 2013 年底大胆改名为 Docker 公司。不过，这个在当时就颇具争议的改名举动，也成为了日后容器技术圈风云变幻的一个关键伏笔。

思考题

你是否曾经研发过类似 PaaS 的项目？你碰到过应用打包的问题吗，又是如何解决的呢？

感谢收听，欢迎你给我留言。

精选留言

李博越 1

后面求加餐能讲讲cncf各个产品的overview以及关系

2018-08-28

旭东 0

Devops和Docker的发展关系会有讲吗？微服务发展需求促成了Debops的发展，而Docker促进了微服务的Devops。
个人的理解不知对否

2018-08-28

江中芦苇 0

沙发

2018-08-28

atompi 0

重复打包，重复配置，换了运行环境，你就不得不再来一遍。我们创建了适用于一个应用的部署模式，但我们仅仅只是创建的它，并不能批量生产它。Docker的出现，好比告诉我们：“你应该用你的模板去快速的批量生产，而不是按照这个模板再‘创造’一个一样的模板”

2018-08-28