开发运维云平台介绍

我们所做的开发运维云平台，基于Openshift及WebIDE等技术，集成了在线开发、构建部署、运维监控、微服务部署、数据持久化等功能，并能够实现一键构建源代码并部署运行，一键部署热门应用等功能，以及持续集成、滚动更新、灰度更新等高级功能。利用现今最流行的Docker容器技术，Kubernetes容器编排技术，大大节省了开发和运维的成本。

什么是容器？使用容器有什么好处？

容器，可以理解为一种轻量化的虚拟机。Docker[[1]](#footnote-1)已经成为了现阶段容器技术的代名词。Docker容器通过使用进程空间技术、UnionFS文件系统，将程序运行需要的所有依赖软件甚至是操作系统软件，打包到一个镜像当中，并在独立的进程空间中运行程序，高效的利用系统资源。几乎所有热门的语言、操作系统及运行环境均有其Docker镜像[[2]](#footnote-2)，如CentOS、Ubuntu，以及Java和Python等等，满足所有运行环境的需求。开发人员只需编写配置文件，构建镜像，就能够实现一次构建，多次部署。

传统的软件构建，使用虚拟机或云主机，使用容器的构建部署方式的区别，如下图所示。

图1 软件构建方式对比

图2 扩容方式的对比

构建部署扩展就是这么简单！

不仅如此，Docker还提供了出色的，接近原生系统的运行效率。下面的系统结构图，展示了用户应用程序运行系统的整体结构。

图3 操作系统结构[[3]](#footnote-3)

很多情况下，单个程序部署在一台物理服务器，如图3中第一列的结构，虽然性能良好，但资源使用效率不高。使用容器技术时，去除了虚拟化所需要的Hypervisor层，完全消除了由于虚拟化带来的性能损耗。使用独立的进程空间，意味着用户的应用程序跟物理系统的应用程序处在同一层，只不过该应用程序所使用的所有依赖软件来自Docker的镜像。

除此以外，使用容器还有以下的优势[[4]](#footnote-4)：

* 更快的启动方式：相比虚拟机启动需要数分钟，镜像的启动只需几秒钟。
* 一致的运行环境：镜像包含了所有依赖文件，而与服务器本身的系统环境无关，理论上任何运行Docker的服务器均能够运行该镜像。
* 持续交付和部署：开发人员只需编写Dockerfile[[5]](#footnote-5)，就能使用Docker构建镜像。通过对镜像打标签，还能够构建运行不同版本的镜像，以运行不同版本的程序。
* 轻松迁移：只需要在新的服务器上拉取并运行镜像，就完成了迁移工作。并且，Docker能在几乎所有的平台上运行。
* 更少的磁盘使用：镜像通常只有几百MB的大小，而虚拟机的镜像则数十GB。
* 单台服务器资源使用更高效：Docker提供了相互隔离的沙箱环境，并且拥有出色的资源调度技术，单台机器能运行上千容器。而单机部署，不同软件对环境要求不同，容易出现冲突的情况，使得单机无法部署过多的应用程序。

我们的平台有什么优势？

Docker虽好，但是却没有满足一点很重要的需求——分布式计算。现阶段的软件都是分布式的，多节点部署，去完成同一个任务，亦或是把系统的各部分功能分别做成一个子系统，也就是现阶段流行的微服务设计。Docker只在一台机器上运行，不能满足分布式系统对于高可用的需求。虽然能够部署Docker到多台服务器上实现分布式，但是，容器的管理、负载均衡、Docker网络管理就会变得异常复杂。

因此，我们的平台，使用了基于当今最流行的容器编排技术Kubernetes[[6]](#footnote-6)的Openshift[[7]](#footnote-7)平台，在Kubernetes提供的基于Docker的分布式计算功能的基础上，提供了多租户管理、网络管理、构建部署自动化、流水线集成等功能。

对于开发人员来说，就能够实现选择镜像->输入Git代码库地址->自动构建部署，或者选择应用模板->填入模板参数->部署模板应用，这些快速便捷的功能。

比如，开发人员有一个使用Maven管理的Java项目，要部署到我们的平台运行，只需要选择Java Maven构建镜像->输入他的Git代码库地址->确认即可，平台会拉取该代码，使用Java Maven构建镜像来自动构建该代码，然后部署运行。另外，我们的平台也提供了Python、Ruby、Nodejs、PHP、Apache等构建镜像，如果有需要，还能添加更多的构建镜像。

如果开发人员需要使用Mysql数据库，他要做的是->选择Mysql模板->填入模板参数，如用户名、密码、root用户名等->确定即可，平台会根据该开发人员的设置，部署一个Mysql应用到平台中，使用用户密码，即可登录。另外，我们的平台还提供了MongoDB、Postgres、MariaDB、Redis等数据库模板，还有其他的应用模板，只要有需求，就能添加进去。

另外，平台支持Webhooks，Git仓库有任何的新推送，就会发送一条信息给平台，平台会自动拉取最新的代码构建并部署。

除了上述的自动化构建部署功能，我们平台还有更多功能可以使用。

滚动更新功能。每当新版本的镜像构建完毕后，平台会同时部署几个新的容器，运行该镜像，在该新容器启动完成后，自动停止旧版本的容器，逐渐的替换容器。而由于平台使用了Service对外提供服务，不是由容器直接提供，在使用该Service的用户看来，这个更新是无缝的。从而实现了软件的不停机更新。

灰度更新功能。有时候我们只希望部分使用新版本，部分使用旧版本，以查看更新版本后是否会出现性能损耗或者其他的异常，这时候我们可以使用平台的灰度更新功能。通过配置Service或者Route去匹配新版和旧版的容器，实现同时使用新版和旧版的容器，并通过调整新版和旧版容器的数量比，来实现从旧版向新版过渡的更新功能。

微服务构建部署。我们可以在同一个项目中，部署整个系统各个微服务的容器，平台会在容器中自动的注入有关其他微服务运行的地址等的相关环境变量，我们的程序只需取用该环境变量，就能够实现快速的微服务发现和整合。

持续集成与持续交付，流水线构建部署等更多的功能，不一一在此介绍。可以参考我们的用户使用手册进行操作。

本地开发&在线开发

我们的平台还提供了本地Eclipse的插件和Intelllij IDEA的插件，方便我们在本地IDE中开发完成后，直接在该插件中管理代码在我们平台上的构建部署。

另外，我们还提供了一个支持主流语言的WebIDE，直接使用浏览器打开，在上面编写和修改Git库的代码，并且直接推送到Git库，方便在没有开发环境的电脑上修改代码。

1. Docker官方网站：<https://www.docker.com/> [↑](#footnote-ref-1)
2. 具体镜像的列表，可以去<https://hub.docker.com/>具体了解 [↑](#footnote-ref-2)
3. 图片参考来源：<https://yeasy.gitbooks.io/docker_practice/content/introduction/what.html> [↑](#footnote-ref-3)
4. 资料来源：<https://yeasy.gitbooks.io/docker_practice/content/introduction/why.html> [↑](#footnote-ref-4)
5. Dockerfile说明：<https://docs.docker.com/engine/reference/builder/> [↑](#footnote-ref-5)
6. Kubernetes官网：<https://kubernetes.io/> [↑](#footnote-ref-6)
7. Openshift官网：<https://www.openshift.com/> [↑](#footnote-ref-7)