关于容器化落地转型的思考

技术总体部 刘智勇

摘要： 云计算的发展让很多人从中受益，譬如产品管理的轻量化、即时的可扩展性和更显著的业务提升。而容器化技术作为领先的虚拟化云产品，在企业的数字化转型实践中越来越发挥不可或缺的作用。本文立足于容器化落地转型实践，从建设容器化支撑平台，建立容器迁移指南等多个方面阐述容器化落地转型的一些思考。

关键词：Docker，Kubernetes，流水线，持续交付，云原生。

一、前言

2020 年初以来，总体部会同保障部，陆续在技术总体部，数字化转型小组，业务系统部、监查系统部开展容器化落地转型的实践，取得了一定成果和积累了一些经验。本文主要讲述在整个落地转型过程中的一些思考。

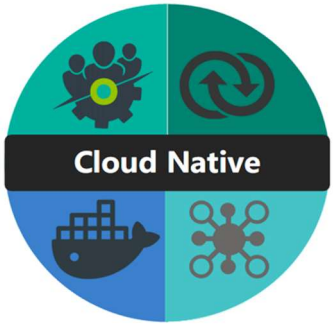

图 1. 云原生

自 2015 年 Pivotal 公司率先提出了云原生应用，之后不久 Google 主导成立云原生计算基金会（CNCF）以来。云原生的概念大火，其生态断壮大，参与者和涵盖应用也愈来愈丰富。本文也将容器化转型实践作为云原生的一个领域来讨论（见图 1）。首先从 docker、k8s 、容器化持续交付三个方面来重点阐述容器化支撑平台的建设。然后阐述企业内部如何建立容器迁移指南，以帮助业务团队快速方便上容器。最后讨论容器化带来的云的变革以及下一步的工作。

二、容器化支撑平台

建设一个稳定，方便且安全的容器化支撑平台是容器化落地转型的第一步。Docker 实践了**“一次构建，处处可用”**的思想，k8s 在 Docker 的基础上实践了一个**“生产级别的容器编排系统”**。而容器化持续交付则构思了一个**“持续构建和发布容器软件”**的方法。将 Docker，k8s 和持续交付流水线技术相结合，给用户提供一个简单易用的容器平台，是容器化支撑平台的首要目的。

2.1 docker 支撑平台

一个 docker 支撑平台需要有 Docker Client，Docker Host，Docker Registry 三大组成部分，下面阐述这三大组成部分的建设概况。

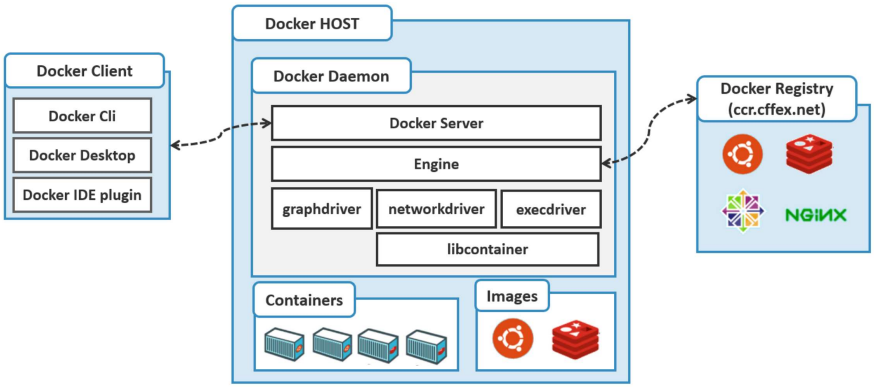

图 2. Docker 架构

2.1.1 Docker Client

如图 2 所示，目前 Docker Client 主要包括 Docker Cli、Docker Desktop、Docker IDE plugin 三类产品。Docker Cli 提供命令行操作 docker 的方式，也是目前主要的 Docker 客户端工具。由于公司内部开发机以虚拟化的形式提供，导致官方的 Docker Desktop 并不能很好地适配内网环境，目前使用 Docker IDE plugin 作为临时替代方案，但不具备通用性。对 Docker Desktop 的重新改造是后期 Docker Client 的研究重点。

2.1.2 Docker Host

Docker Host 是 Docker 各类操作的主要承载体。Docker Host 的初始建立采用 OpenStack 云平台的模板机技术，通过建立 docker 模板机（rhel-docker），免去用户安装和配置 docker 服务的麻烦，从而大大降低了使用 docker 技术的门槛。如图 2 所示，Docker Host 涵盖了 Docker Daemon，Images，Containers 三大模块。Docker Daemon 由 Docker Server（http 服务），Engine（主要处理引擎），多个 driver 和 libcontainer（核心实现技术 Linux namespace，cgroup 等）组成。Images 和 Containers 使用了 UnionFS 的关键存储技术。

2.1.3 Docker Registry

Docker Registry 是一个镜像生产和消费对接的市场，是整个 Docker 镜像链中的枢纽。目前，使用 Jfrog 公司 artifactory 产品基本建立其企业级别的 docker 镜像中心。提供了 Docker，Helm，AOP-docker 包的托管，分销，升级等功能。有力地支撑了整个 docker 体系的运转。但整个 docker registry 与 CICD 体系的集成、镜像的安全扫描还有很多的工作需要做。整个 Docker Registry 的划分为二方库和三方库建设。

Docker 二方库可以精细化管理企业的内部镜像。如图 3 所示，在二方库的建设上，首先建立 Docker-dev，Docker-test，Docker-prod 库来区分开发，测试和上线阶段。保障从 Docker-prod 获取待上线包，通过部署引擎传输到不同的生产系统，每个生产系统都有 jfrog 公司的 jcr 产品作为镜像的缓存，部署引擎从 jcr 系统中获取对应环境的镜像，并部署到对应的生产系统上。

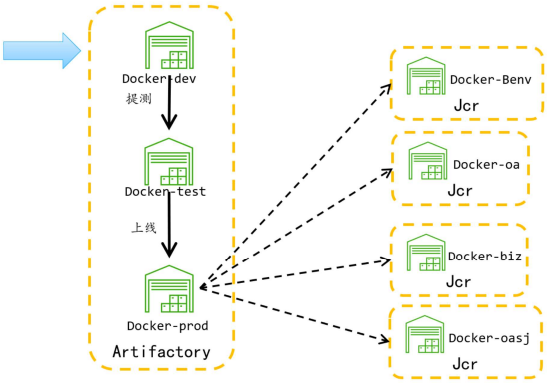

图 3. Docker Registry 二方库建设

Docker 三方库作为引进社区和其他组织的优秀镜像的手段，夯实和丰富容器化的镜像市场。如图 4 所示，在三方库的建设上，通过在内网（cffex.net）和外网（cffex.io）分别搭建 Artifactory 并打通网络，外网 Artifactory 链接到主流的 Docker 镜像中心，由此实现了企业内部拓展使用第三方镜像的目的。三方库现在主要面临的是性能问题，主要原因是：国内互联网并没有一个匹敌 docker.hub 的镜像中心，而直连到 docker.hub 速度比较慢；内网 Artifactory 和外网 Artifactory 的带宽也不足。上述性能问题比较影响 Docker 三方库的用户体验。

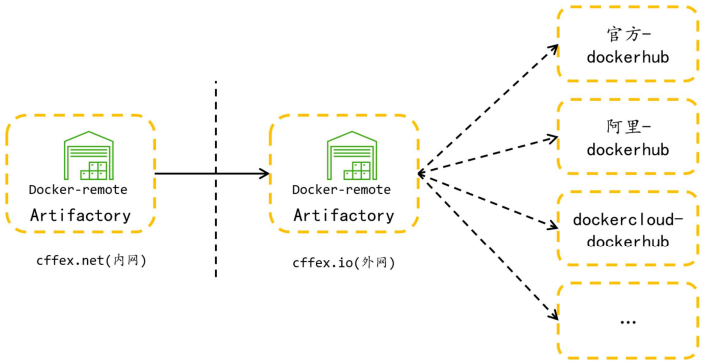

图 4. Docker Registry 三方库建设

2.2 k8s 支撑平台

作为一个生产级别的容器编排系统，k8s 具有非常多强大的功能。其主要功能为自动部署，扩展和编排容器化应用程序。其产品特点为：

（1）服务发现与负载均衡，自我修复，水平扩缩，配置管理；

（2）批量执行，存储编排，自动化上线和回滚；

（3）服务拓扑，自动装箱，端点切片，IPv4/IPv6 双协议栈

为了更加结构化地建设和管理 k8s，制定图 5 所表示地 k8s 分层建设思路图。下面将自低向上逐层分析。

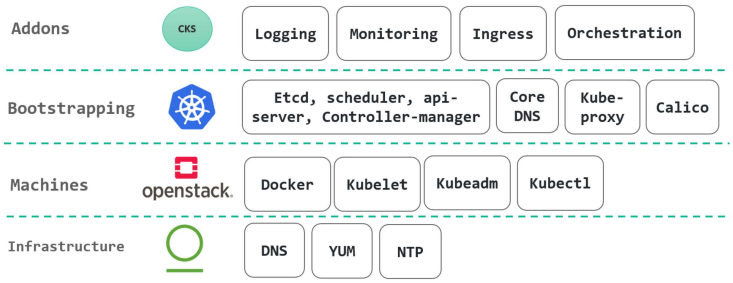

图 5. k8s 分层建设思路图

2.2.1 基础设施层

基础设施层主要是为构建 k8s 支撑平台而衍生的额外 IaaS 层功能。目前我们的 IaaS 层依托于 OpenStack 云平台，提供了基于模板机的虚拟机的管理。但缺乏一些网络和存储的基础设施，譬如 DNS，YUM，NTP 等服务。为了构建容器化支撑平台，首先需要在 IaaS 层补足这些短板。后续这块还需要按需补充，譬如增加 NFS，LoadBalander 等相关服务。

2.2.2 虚拟机层

在基础设施层的基础上，依托 Openstack 的虚拟机，建立涵盖 Docker，Kubelet，Kubeadm，Kubectl 等程序的 k8s 模板机，从而免去运维人员安装和配置 k8s 节点组件的步骤，提升 k8s 集群运维人员的节点管理能力。后续也可以将该层和 Openstack 云平台深度整合，利用 k8s 的 cloud-controller-manager 组件实现节点的自动上下线，甚至是基于流量的弹性节点扩缩容。

2.2.3 平台层

虚拟机层构建了一个机器集合，但还远称不上集群。使用 kubeadm（也可以用其他模式）可以部署一整套 k8s 的基础组件，这些组件将一组机器集合成一个紧密联系的集群。建设到平台层，k8s 支撑平台已经初具服务能力，使用 kubectl 就可以部署一些 k8s 的应用，这些应用自带“服务发现与负载均衡，自我修复，水平扩缩，在线管理配置”等功能。

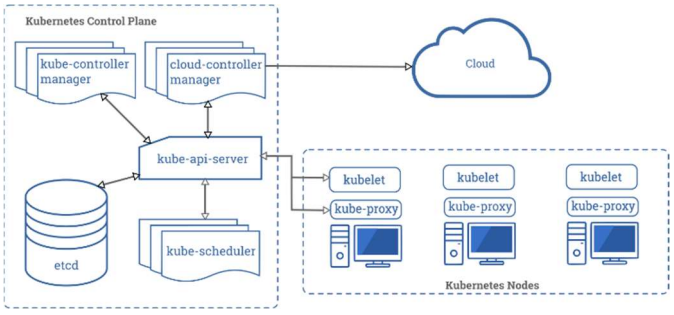

图 6. K8s 架构

整个平台层的 k8s 架构将类似与图 6 所示。由图可知，k8s 是典型的 master-slave 架构，master 包括 etcd、kube-api-server、kube-scheduler、kube-controller-manager、cloud-controller-manager（最近几个版本才有），slave 包括 kubelet、kube-proxy。K8s 使用开发接口 CRI（Container Runtime Interface）、CNI（Container Network Interface）、CSI（Container Storage Interface）来对接 docker 容器引擎。产品本身是一个微服务架构，各个组件之间做到了松耦合。

2.2.3 服务层

单纯地部署 k8s 平台并不能形成战斗力，更多的需要我们需要的是一个 k8s 组合体（见

图 7）。平台层的 k8s 类似于一个裸功能的分布式操作系统，虽然核心功能都已经完备，但使用起来非常得不便，同时也缺乏监控、日志和对外的接口。添加一些 k8s 插件到平台层，补足各个环节，逐步建立起一个完备稳定的可对外服务的 k8s 服务平台。目前看来服务层的工作主要集中在以下几个方面：

- **日志栈**（Logging stack），使用 Elasticsearch，Fluentd，Kibana 技术；
- **监控栈**（Monitoring stack），使用 Prometheus，Alertmanager，Grafana，kube-dashboard 等技术；
- **接口栈**（Ingress stack），使用 NGINX Ingress Controller，cert-manager，OAuth2 Proxy，ExternalDNS 等技术；
- **编排栈**（Orchestration stack），使用 Helm，Kubeapps 等编排工具。

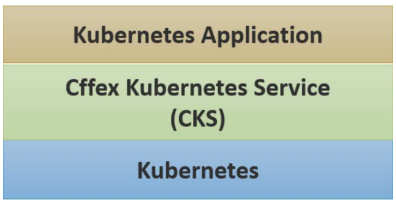

图 7. k8s 组合体-cks

2.3 容器化持续交付

软件开发生命周期会有两次瓶颈，第一次：需求到开发，针对不断变化地需求。第二次：开发到生产，待上线包堆积。第一次瓶颈由 Jira 和 Confluence 为代表的敏捷管理平台实现。容器化持续交付主要解决第二次瓶颈。为了更有效地开展容器化持续交付，建立**三级跃迁流水线模型**（见图 8）。形成以制品库为枢纽，沟通起开发、测试、保障个团队，串联起开发、测试、生产等网络环境，各司其职，相互协作的持续交付流水线。

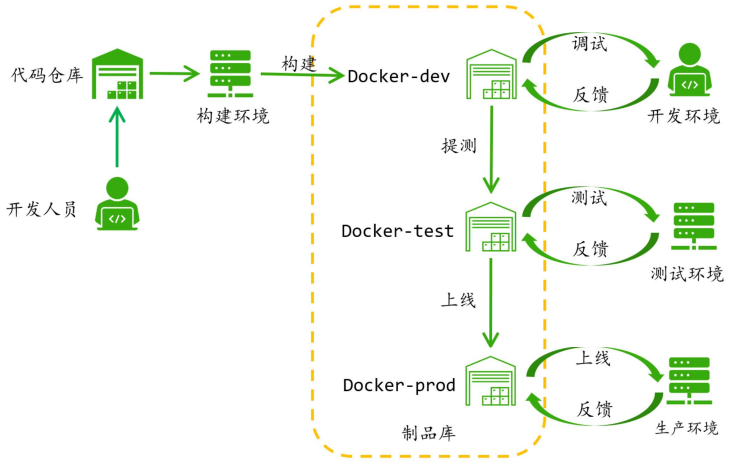

图 8. 三级跃迁流水线模型

2.3.1 构建工程

在构建工程阶段，容器化持续交付主要有两个变革。第一个变革是流水线构建机不需要安装众多的构建程序（诸如 maven，npm，pypi 等），只需要 docker run 特定的镜像即可构建各种语言的程序。这个变革也让流水线有了依据特定版本构建程序构建应用的能力。第二个变革是构建阶段加入 sonar 静态代码扫描和单元测试两个阶段，而且这两个阶段的扫描报告

和测试报告可以通过制品库的元数据功能记录到 Docker registry，使得流水线比较完备和精妙。

2.3.2 测试工程

在测试工程阶段，容器化持续交付重点关注在自动化部署领域。通过不断优化容器应用的编排手段。从 Docker 的命令行编排，到 docker-compose yaml 编排，再到 k8s 的 yaml 编排，一直优化到使用 helm 编排实现一键部署。通过对各种部署场景的分析，实现无状态应用，有状态应用等多种部署模型。

2.3.3 部署工程

在部署工程阶段，容器化持续交付考虑到公司的实际情况，将容器化的部署模型和 AOP 做适配，形成了使用 AOP 上线容器化应用的模型。并基于此模型自研了 AOP 的打包流水线，从打包到部署实现了符合我司运维规范的上线流程。

三、容器化迁移指南

在容器化的旅程中，有些人可能行动迅速，而有些人则可能需要更多一些时间。建立完备的容器迁移指南计划，以消除业务团队的恐惧，帮助业务团队平缓转型。从传统应用到容器应用的变迁有点类似于从骑自行车到开汽车，需要从认知和实践对驾驶行为做大变革。下面将从核心概念和操作、规范和最佳实践、社区和生态三个方面来阐述容器化迁移指南。

3.1 核心概念和操作

对容器应用的编排有多种方式，从原生的 Docker 命令行，docker-compose 的 yaml 文件，到 k8s 的 yaml 编排，再到封装简易的 helm charts 以及适用于 AOP 的 AOP-docker 包。容器化迁移指南为用户提供了在不同场景，不同阶段下的各种编排工具。然后无论是哪种编排工具，封装得有多简单，操作有多方便，了解一些 k8s 的核心概念和操作还是非常有必要的。目前用到的 k8s 的概念主要有：Cluster，Pod，Deployment，Service 等。

- Cluster：计算、存储、网络等资源的集合，k8s 利用这些资源运行容器应用。
- Pod：k8s 的最小调度单元，一个 pod 对应一个或多个 container，同一个 pod 里面的 container 共享一个网络栈和一块存储空间。
- Deployment：最常用的 Controller，实现 Pod 的多个副本，确保 Pod 按照期望的状态运行，集成应用部署、滚动升级、弹性伸缩等功能。
- Service：K8s 通过 service 来解决这些 pod 的 ip 多变的问题，可以把 service 简单理解为一个负载均衡器，也可以说 service 是一组功能相同 pod 的统一入口。

3.2 规范和最佳实践

正确地构造和使用容器化应用，可以享受到容器化带来的诸多优势。反之，你会觉得容器化应用处处掣肘。正如规范地驾车可以让你更快更安全的到达目的地，而毫无章法的驾驶将会让你变成马路杀手。目前总结的典型容器化应用最佳实践如下：

- 配置与应用分离：应用与环境松耦合，配置在运行时注入。
- 计算与数据分离：数据挂载在主机或者共享存储，日志收集在日志处理引擎中（ELK

or EFK）。

- 服务透明访问：使用域名的方式来访问你的服务（DNS 非常重要）。
- 资源限制：Pod 需要有资源限制，应用本身也需要有资源限制。
- 健康检查：应用的运行状态必须有可信的健康数据。

容器化团队还将会继续总结抽象这方面的知识，形成体系完备，简单易懂，少而精的规范和最佳实践集。

3.3 社区和生态

基于我司现有的网络环境现状，整个容器化生态是内循环和外循环相结合的运行模式。在内循环上以 Docker Registry 二方库和三级跃迁模型为基础，串联起开发，测试和保障等部门。在外循环上，以 Docker Registry 三方库为基础，串联起外部环境的各个组织，引进第三方优质镜像。

然而，整个社区和生态依然有很大改进空间。从内循环来看，丰富的文档，最佳实践案例，技术交流等这些还没有完全建立，整个容器化持续交付流水线运转并不够滑润。从外循环来看，只有引进镜像而无出口镜像，对于外面容器化的生态只有使用而无发明贡献。

总而言之，容器化之路漫漫其修远兮，容器团队需要深炼内功，潜心研究，深入业务一线，将精妙的技术和公司的业务实践有力地结合起来。

四、云的变革

云的变革有助于帮助我们取得长期成功，但也需要阐明这样的现实，很多业务团队在云的过渡和转变中面临很多的困难和问题。推动容器化落地转型，促进云的变革必须从多个维度，以开放的思想来考虑。

4.1 内部服务

我们相信，容器的落地转型会对应用产生诸多影响，不仅仅是生产模式和运营模式，甚至是影响业务的运行模式。保持开放，拥抱云的文化，相信我们在变革过程中投入的时间和精力，都将会以多种方式获得回报。

4.2 商业合作

如果抬起头看看外面，会发现外部世界的云变革异常激烈和快速。很多新兴的软件公司和软件产品都采用订阅的类 SaaS 方式来提供服务。IT 行业的销售、采购等商业行为也在发生变化。在容器化转型甚至是云的变革中，我们必须增强和外部厂商的合作，在与优质厂商的持续商业合作中，双方都会取得共赢。

五、未来和下一步工作

目前来看，容器化落地转型还在一个初级跃进到中级的水平，对于容器化的排障，k8s 基础对象的理解，k8s 管理组件的研究，k8s 衍生插件的理解等方面都需要持续投资。未来容器化团队将会在 k8s 做深做大上持续钻研，逐步帮助业务团队在容器化转型方面走得更稳更快，让每一个人都享受容器的益处。

六、结语

在这个迅速变化的时代，我们感受了软件快速变革带来业务提升的欣喜，也承受着比以往更大的数字化转型压力。没有有效、方便且安全的支撑平台，软件的开发和连接将会是异常艰难的。始终记得我们做 Docker，k8s 和流水线的使命之一是“**又好又快地发布软件**”。持续想象和构建一个世界，在这个世界上，所有软件更新都从开发人员的指尖无缝地一直流到最终用户（Liquid Software）。保持谦卑之心，倾听用户的声音，理解他们并将这种理解转化为产品，最终服务于终端客户。

参考文献：

[1]. Kubernetes in Action 中文版/（美）马尔科-卢克沙著：七牛容器云团队译. —北京：电子工业出版社，2019.1

[2]. JezHumble, DavidFarley, 亨布尔, 法利, & 乔梁. (2011). 持续交付：发布可靠软件的系统方法. 人民邮电出版社.

[3]. 浙江大学 SEL 实验室. Docker：容器与容器云[M]. 人民邮电出版社, 2015.

[4]. Docker. https://docs.docker.com

[5]. Jfrog. https://www.jfrog.com/confluence/display/RTF/Artifactory+User+Guide

[6]. Kubernetes. https://kubernetes.io/docs/home/