CNUT
Connect Container Community

10个精选的
容器应用案例

Ten Container-Related Case Studies

数人云 shurenyun.com

InfoQ

版权声明

InfoQ 中文站出品

10 个精选的容器应用案例

出版：极客邦控股（北京）有限公司

北京市朝阳区洛娃大厦 C 座 1607

欢迎共同参与 InfoQ 中文站的内容建设工作，包括原创投稿和翻译，请联系 editors@cn.infoq.com。

网 址：www.infoq.com.cn

卷首语

自2013年Docker诞生以来，该技术在业界迅速掀起一股热潮。短短几年时间内，Docker生态系统迅猛发展，在企业中的应用遍地开花。Docker为企业级应用的构建、交付和运行带来了革命性的便利，它屏蔽了各种复杂的异构环境，真正做到了“Build Once, Run Anywhere”，极大地降低了企业应用开发者和运维人员的工作复杂度。Docker的火热，也推动了相关技术生态的快速成长。企业的应用需求纷繁复杂，Docker不可能解决企业客户的所有需求，因此Docker相关技术层出不穷，进一步带动了Docker在企业的落地，丰富了Docker的生态圈。

比如，针对Docker的调度，就涌现出了Mesos、Kubernetes等优秀的开源项目。当企业运行成百上千个Docker应用程序的时候，靠人工来调度管理这么多的Docker程序是不现实的，企业需要对Docker程序进行自动化调度管理，而这就是Mesos和Kubernetes显身手的地方。Mesos源自加州伯克利大学的AMP实验室，是Apache旗下的顶级开源项目，其在Twitter、Airbnb、Apple等国外IT巨头已经有大规模应用，Twitter最大的Mesos集群已经有上万台服务器之多。Kubernetes源自Google，虽然诞生时间很短，仅有一两年的时间，但是因为Google巨大的影响力，Kubernetes也受到了广泛的关注和应用。

本书列举了十个Docker及Docker相关技术Mesos和Kubernetes的应用案例，充分展示了这些技术在构建企业级应用的巨大优势：一方面是这些技术在弹性计算方面的

优势，诸如蘑菇街和京东案例；另一方面是这些技术在构建 PaaS 平台方面的优势，诸如阿里百川和 SAE 的案例。特别需要指出的是，Docker、Mesos 和 Kubernetes 等技术，不仅在国外受到热捧，在国内也得到了广泛关注。本书列举的十个 Docker 案例，除了 Apple 来自美国，其它九个都是源自国内的案例。而且，这九个国内的案例，不仅有来自国内互联网巨头，更有来自浙江移动这样的传统企业，这充分显示了国内对于热点新技术的追踪和应用已经不输于国外。相信在不久的未来，国内在 Docker 的相关领域必将出现新的技术，超越国外同行，引领这些领域的发展方向。

数人云 CEO　**王璞**

目录 / CONTENT

1

中国移动浙江公司数据中心操作系统（DCOS）实践

作者 **钟储建**

背景

中国移动浙江公司数据中心自2009年开始从小型机为主的架构开始了X86化、IaaS资源池化、PaaS 资源池化的发展历程，数据中心在向云计算转型过程中软硬件管理的能力和效率上面临着诸多挑战：

1) 应用的快速部署开通受到极大制约：大部分应用系统有开发、测试、准发布和生产四个部署环境，各部署环境不一致，代码从开发到上线环节多、部署复杂、容易出错，无法满足业务快速上线的要求 。

2) 系统弹性扩展能力不足：应用系统部署以虚拟机为单位构建，系统的扩容需要经历虚拟机分配、软件安装、应用部署、测试、割接入网等环节，在业务量突增时无法进行快速的扩展；系统的缩容不能随意进行，导致资源存在一定的预留和浪费。

3) 现有资源利用率较低：资源池 CPU 平均利用率仅为 10-20%左右，显著低于先进数据中心 50-70%的利用率。

4) 应用系统仍旧“烟囱式”的建设：以虚拟机为基础的资源池化在应用系统架构上并没有改变竖井化的建设模式，应用与平台没有解耦，高可用、监控运维等无法标准化。

针对在云化和系统运维中碰到的上述问题，我们在 2014 年 3 月就开始关注 Docker 容器化技术并在核心系统中进行了试点。2015 年业界开始流行数据中心操作系统（DCOS：Data Center Operating System）的概念，正好与我们私有云架构中规划的弹性计算相契合，因而提出以开源技术为核心建设 DCOS 验证网，对新一代云计算技术体系架构下的数据中心解决方案、产品选择、集成交付和运维保障进行全面验证：

1. 为整个数据中心提供分布式调度与协调功能，统一协调各类资源，实现数据中心级的弹性伸缩能力。
2. 提供一个高效率、可靠、安全的管理数据中心的平台，确保各类资源随着应用的需求动态调度，同时简化应用程序的开发、部署难度。

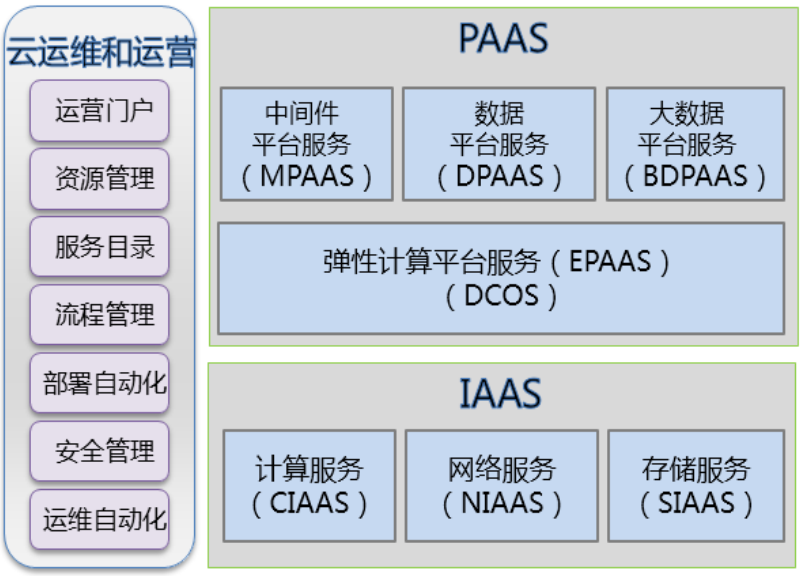


图 中国移动浙江公司私有云架构

技术选型

数据中心操作系统（DCOS）是为整个数据中心提供分布式调度与协调功能，实现数据中心级弹性伸缩能力的软件堆栈，它将所有数据中心的资源当做一台计算机调度。

大规模应用的数据中心操作系统有：Google Borg/Omega 系统和 Twitter、Apple、

Netflix 等公司基于 Mesos 构建的系统。

可以用于数据中心操作系统构建的开源解决方案有：

1) Mesos：Mesos 最早由美国加州大学伯克利分校 AMPLab 实验室开发，后在 Twitter、Apple、Netflix 等互联网企业广泛使用，成熟度高。其中，Mesosphere 公司 DCOS 产品，就是以 Mesos 为核心，支持多领域的分布式集群调度框架，包括 Docker 容器集群调度框架 Marathon、分布式 Cron（周期性执行任务）集群调度框架 Chronos 和大数据的主流平台 Hadoop 和 Spark 的集群调度框架等，实现系统的资源弹性调度。

2) Apache Hadoop YARN：Apache Hadoop YARN 一种新的 Hadoop 资源管理器，它是一个通用资源管理系统，可为上层应用提供统一的资源管理和调度。

3) Kubernetes：Kubernetes 是 Google 多年大规模容器管理技术的开源版本，面世以来就受到各大巨头及初创公司的青睐，社区活跃。

4) Docker Machine + Compose + Swarm：Docker 公司的容器编排管理工具。

5) 此外，CloudFoundry/OpenShift 等 PaaS 产品也可以作为 DCOS 的解决方案。

相关技术在调度级别、生态活跃、适用场景等方面的比较如下表所示：

产品名称	Mesos	Yarn	Kubernetes	Docker Machine+ Compose +Swarm	CF/OpenShift
调度级别	二级调度（Dominant Resource Fairness）	二级调度（FIFO，Capacity Scheduler，Fair Scheduler）	二级调度（基于 Predicates 和 Priorities 两阶段算法）	一级调度 （提供 Strategy 和 Filter 两种调度策略）	CF 一级调度（基于 Highest-scoring 调度策略）/OpenShift 使用 Kubernetes
生态活跃	活跃	活跃	非常活跃	活跃	一般
适用场景	通用性高，混合场景	大数据生态场景	目前较单一	较单一	较单一

成熟度	高	高	中	低	中
应用与平台耦合度	低	中	中	低	高
应用案例分析	Twitter、Apple、Airbnb、Yelp、Netflix、ebay、Verizon	Hadoop 生态圈应用很多	目前快速发展中，生产环境应用较少	很少	较少，应用与平台的耦合度较高

（注：按照公开文档和使用经验做简单比较，未做详细验证）

根据对适合构建 DCOS 的各种技术架构的评估，我们选择以 Mesos 为基础的方案。其优点是成熟度高、使用两级调度框架、适合多种应用场景、支持混合部署、应用与平台耦合度低。

建设历程

2014 年 3 月开始关注 Docker 容器化技术，2014 年 8 月启动 Docker 应用的技术验证。

2014 年 11 月将核心系统 CRM 的一个完整集群迁移到容器运行，Docker 正式投入生产。

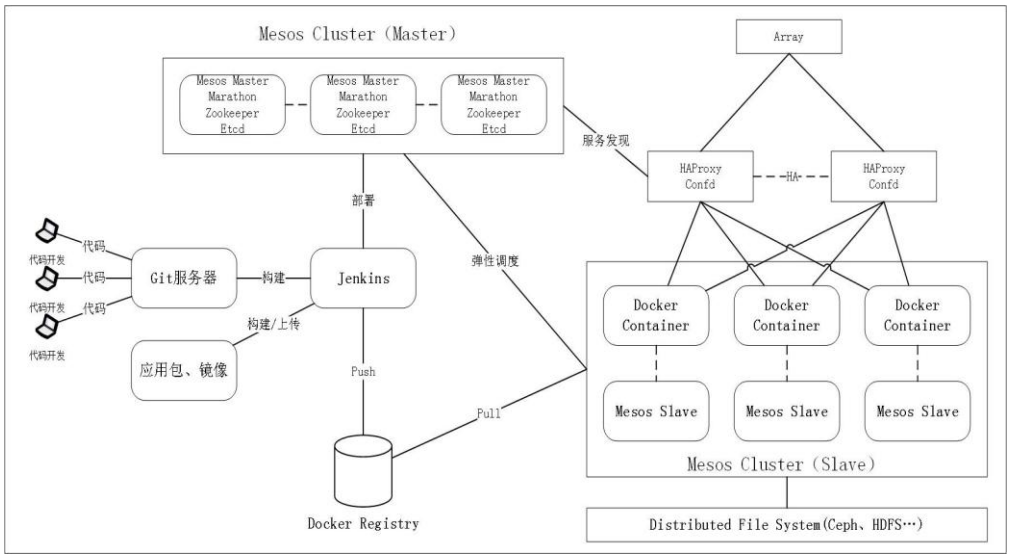
2015 年 8 月提出数据中心操作系统的设想，建设 DCOS 验证网。

2015 年 11 月 4 日中国移动浙江公司 DCOS 验证网上线，成功支撑手机营业厅“双 11”活动，12 月 10 日 CRM 系统试点迁移到 DCOS。

DCOS 技术方案

1. 技术架构

中国移动浙江公司 DCOS 方案采用了以容器为基础封装各类应用和运行环境，以 Mesos、Marathon 为核心实现容器资源的分布式调度与协调，以 Haproxy、Confd、Etcd 实现服务注册和业务的引流。架构如下：



a) 应用封装

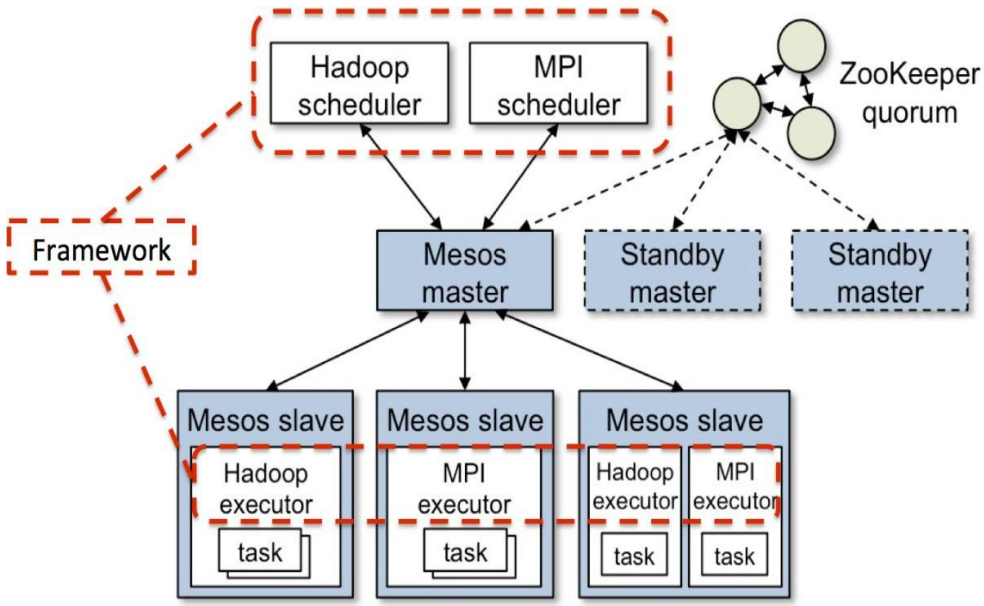
应用封装采用 Docker 做为应用容器引擎，Docker 是轻量级虚拟化技术，它在标准的 LXC 之上融合 AUFS 分层镜像管理机制，并以应用为单元进行“集装封箱”，实现的相关应用封装能力如下：

1. Docker 容器技术可以部署应用到可移植的的容器中，这些容器独立于硬件、语言、框架、打包系统，帮助实现持续集成与部署。一个标准的 Docker 容器包含一个软件组件及其所有的依赖 ，包括二进制文件、库、配置文件、脚本等。

2. Docker 容器可以封装任何有效负载，可以在任何服务器之间进行一致性运行。开发者构建的应用只需一次构建即可多平台运行。

b) 资源调度

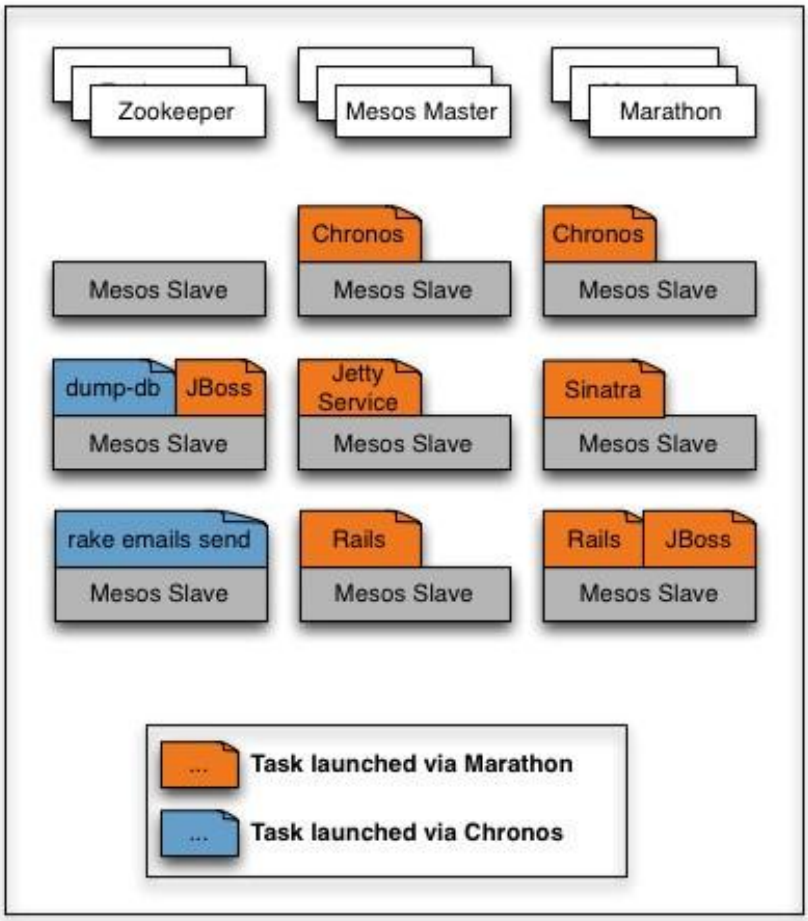
使用 Mesos 作为资源调度框架，其核心工作原理如下：



1. Mesos Master 负责将资源分配给各个框架，而各个框架的 Scheduler 进一步将资源分配给其内部的各个应用程序。
2. Mesos 和不同类型的 Framework 通信，每种 Framework 通过各自的 Scheculer 发起 task 给 Mesos slave，对 Mesos 集群进行调度管理。
3. Framework 的 Executor 执行其 Scheculer 发起的 task。

c) 任务调度

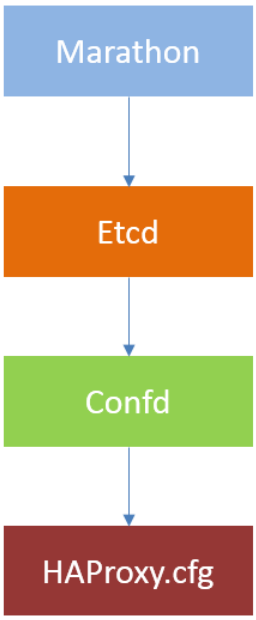
由于 Mesos 仅负责分布式集群资源分配，不负责任务调度。因此，需引入 Marathon 来基于 Mesos 来做任务调度，Marathon 用来调度长期运行的服务。Mesos 集群可以混合运行不同类型的任务 ，其任务调度示意图如下：



1. Marathon 基于 Mesos 的任务调度为动态调度，即每个任务在执行之前是对具体服务器和绑定端口均为未知。
2. Mesos 集群上混合运行着包括 Marathon 在内各种调度框架的任务，当某台服务器宕机以后，Marathon 可把任务迁移到其他服务器上，实现容错。

d) 服务注册及引流

通过 Haproxy、Confd、Etcd 配合实现数据中心应用的动态服务注册与引流，其中 Etcd 是一个高可用的键值存储系统，主要用于共享配置和服务发现。HAProxy 提供高可用性、负载均衡的解决方案。其主要的架构流程如下：



1. Marathon 通过 Mesos 启动 Docker 容器时，Marathon 将容器启动信息通知 Etcd 服务。

2. Etcd 服务将已经启动容器信息注册到 Etcd 键值库中。

3. Confd 监测到 Etcd 中相关的服务变化，Confd 就会根据变化的情况更新 Haproxy 的 cfg 配置文件并执行重新加载命令，使相关变化生效，同样当容器停止时也会触发 Haproxy 更新 cfg 配置文件并重新加载，达到动态服务注册。

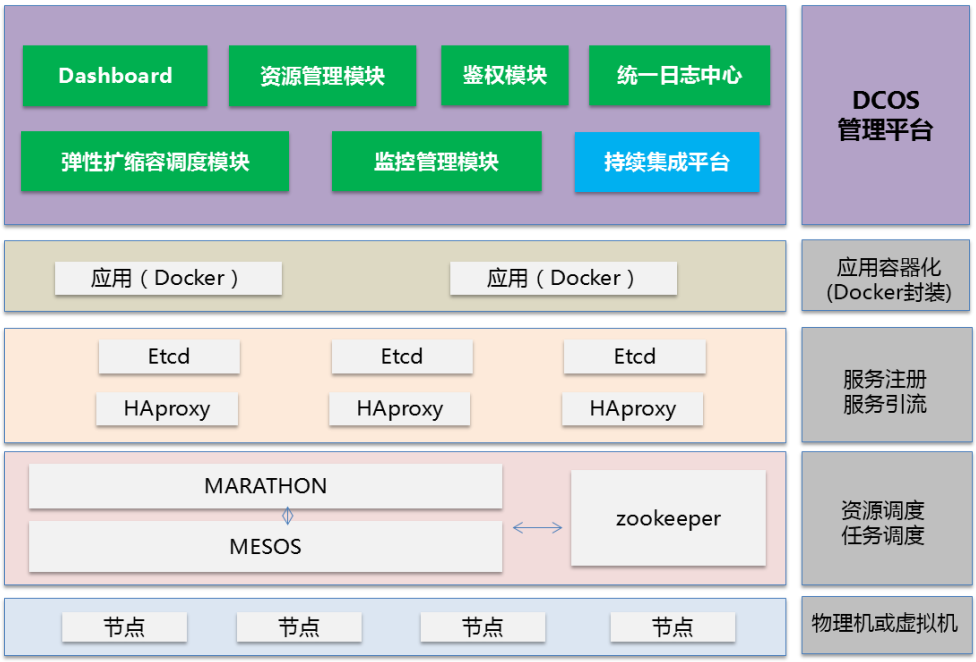
4. 业务请求通过 HAProxy 分发到 Docker 容器中的应用。

e) 自动弹性扩缩容

Marathon 的扩缩容默认只能根据用户需要进行手动调整，我们结合多年的系统运维经验，实现基于并发数、响应时间、CPU 和内存使用率等容量指标进行自动弹性扩缩容调度的模块。结合前面提到的 HAProxy、Confd、Etcd 动态服务注册和引流能力，实现应用的自动弹性扩缩容能力。

2. 功能框架

以开源技术 Mesos 、Marathon 、Docker、HAProxy 为引擎，在其上开发了 DCOS 控制台、资源管理模块、鉴权模块、统一日志中心、弹性扩缩容调度模块、监控管理模块、持续集成平台。DCOS 的功能框架如下：



1. DCOS Dashboard

2. 资源管理模块：服务目录管理、规则管理、CMDB 信息；

3. 监控管理模块：监控数据采集、日志管理、告警管理和事件管理；

4. 弹性扩缩容调度模块：基于 CPU 使用率、内存使用率、服务并发数、响应时间等容量数据，通过定制的调度算法实现服务的自动弹性扩缩容；

5. 统一日志中心：采用 Elasticsearch、Logstash、Graylog 构建，实现对容器日志的统一存储及检索；

6. 鉴权模块：用户管理、用户组管理、权限策略管理和统一认证接口；

7. 持续集成平台：镜像构建、集成测试、流程管理和上线管理。

DCOS 应用

在对 DCOS 平台验证网充分测试验证后，我们选取手机营业厅系统作为业务试点，迁移至 DCOS 平台。

手机营业厅是面向中国移动客户提供快速便捷的查询、办理和交费等自助服务的客户端软件及系统，中国移动浙江公司手机营业厅注册用户 2500 万，日活跃用户数 300 万。

DCOS 平台采用 93 个主机节点，其中平台部分由 5 个节点构成 Mesos Master Cluster，8 个节点构成 HAproxy Cluster，计算节点由 80 个 Mesos Slave 节点组成，平台和计算节点均跨机房部署，该平台可在 1 分钟内轻松扩展到 1000 个以上 Docker 容器。

下图为 DCOS 控制台，手机营业厅 WEB 和 APP 两个应用模块在 DCOS 资源池中动态调度，容器数量的变化显示了两个应用模块的弹性扩缩容情况：

效果总结

1) 高性能高稳定性

双十一期间，运行在 DCOS 架构的浙江移动手机营业厅系统承受的并发数最高峰值接近 6 万次/秒，成为浙江移动首个在单日实现 10 亿级 PV 的业务系统。

2) 高资源利用率

DCOS 相较于虚拟机有着基于 CPU、内存的更细粒度的资源调度，多个计算框架或应用程序可共享资源和数据，提高了资源利用率。

3) 高效的跨数据中心的资源调度

DCOS 平台展现了其在线性动态扩展、异地资源调度等方面的优异性能，1 分钟内快速扩展到 1000+的容器（如果应用更轻量启动速度还可以更快），平台和计算节点完全跨机房分布式调度。

4) 自动弹性扩缩容

彻底解决应用的扩缩容问题，容量管理从“给多少用多少”向“用多少给多少”转变，被动变主动。应用的扩缩容时间从传统集成方式的 2-3 天缩短到秒级分钟级，可以根据业务负载自动弹性扩缩容。

5) 敏捷开发、快速部署

容器和 DCOS 技术的结合通过将应用和它的依赖进行封装，隐藏了数据中心硬件和软件运行环境的复杂性，让开发、测试、生产的运行环境保持一致，降低应用的开发、发布难度。传统的部署模式“安装->配置->运行”转变为“复制->运行”，实现一键部署。

6) 高可用性、容灾

DCOS 平台所有组件采用分布式架构，应用跨机房分布式调度。自动为宕机服务器上运行的节点重新分配资源并调度，保障业务不掉线，做到故障自愈。

问题和经验总结

1. 经验

1) Marathon 自动弹性扩缩容调度

Marathon 的扩缩容默认只能根据用户需要进行手动调整，我们结合多年的系统运维经验，实现基于并发数、响应时间、CPU 和内存使用率等容量指标进行自动弹性扩缩容调度的算法。

2) Marathon Etcd 联动实现服务注册发现

Etcd 只是个独立的服务注册发现组件，只能通过在宿主机上部署 Etcd 发现组件，通过其发现宿主机的容器变化来发现，属于被动的发现，往往会出现发现延迟时间较长的问题，我们通过修改 Etcd 组件的发现接口，实现与 Marathon 的 Event 事件接口进行对接，达到 Marathon 的任何变动都会及时同步给 Etcd 组件，提高了系统的发现速度，并且避免在每个宿主机上部署 Etcd 发现组件。

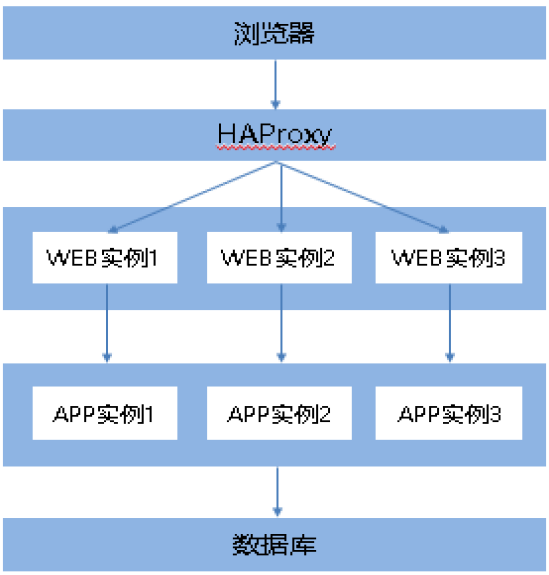
3) DCOS 平台组件容器化改造

为提高 DCOS 平台的可维护性，我们将 DCOS 平台的相关组件全部进行 Docker 化，相关组件运行环境和配置信息都打包到 Docker 镜像中，实现快速部署、迁移和升级。

2. 应用迁移经验

应用要在 DCOS 平台上动态的扩展和伸缩，对应用的要求是无状态化。

以常见的三层架构为例，WEB 层负责展现，APP 层负责处理业务逻辑和数据库进行交互，WEB 层使用负载均衡进行请求分发，WEB 到 APP 层有多种调用方式，如下图所示。



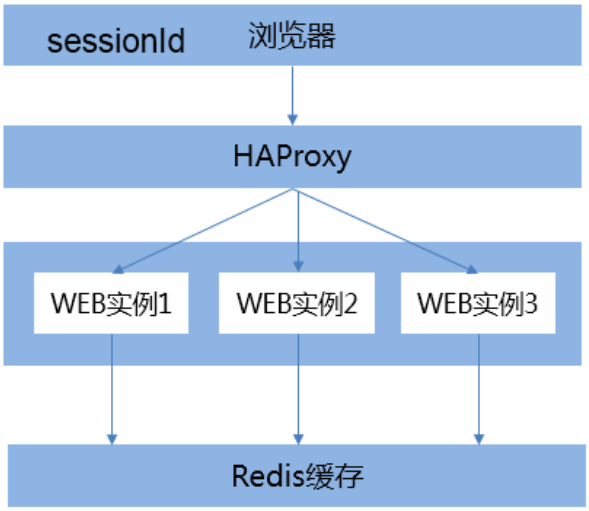
1) WEB 层应用无状态改造

去 HTTP Session：不再采用传统的 HTTP Session 保存会话的方式；

将客户端与 WEB 端的交互改成 http+json 短连接方式；

使用缓存服务器来保存用户的会话信息。

如下所示：



通过以上的应用改造使应用的状态数据与应用分离，WEB 实例的启动和停止不会导致用户会话数据的丢失。

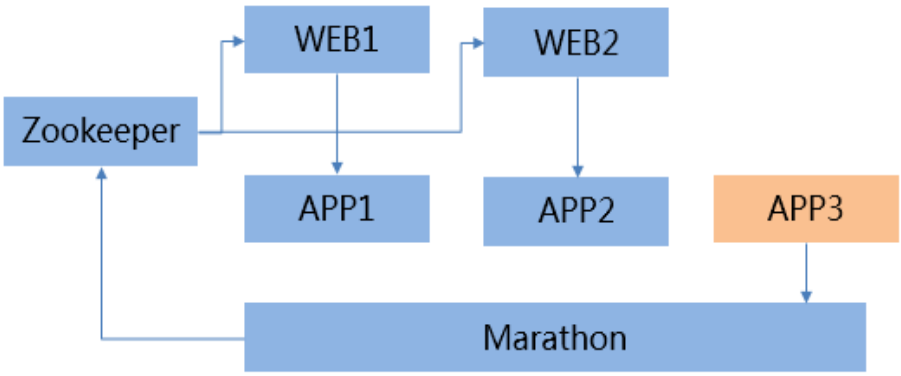
2) APP 层应用改造

APP 层要支持动态的伸缩，除了 APP 层实现无状态化外，取决于 WEB 到 APP 的 RPC 调用方式：

HTTP 协议：同 WEB 层一样使用负载均衡方案 HAProxy+Confd+Etcd；

服务化框架：使用服务化框架服务的发现和注册功能，注意需要将容器外的 IP 和端口上报给配置中心；

未实现服务发现的 RPC 调用：对于没有服务发现和注册功能的传统应用则需进行改造。我们以移动的 CRM 系统为例，CRM 系统使用 EJB 技术实现，APP 层没有服务注册的能力，改造后的架构图如下所示：



Zookeeper 保存 APP 实例的实时分布数据，Marathon 负责监控 APP 实例的运行状态，并在状态发生变化时通知给 Zookeeper 进行修改 APP 实例分布数据，WEB 层根据分组策略获取一组的 APP 实例分布信息，根据该信息轮训调用组内的 APP 实例。

- **分组**

为保证 APP 负载的均衡我们采用分组策略，我们将所有 Zookeeper 内的 APP 实例根据 Hash 算法进行分组，每个组内保持着一定数量的 APP 实例，每个 WEB 请求按照路由策略均衡分发到组内 APP 实例上。

- **扩缩容调度**

垂直调度：因为每次弹性扩缩都会对 WEB 访问 APP 的路由表进行更新，当频繁更新时有可能会造成服务访问的短时异常，为避免该问题我们采用垂直调度机制，每个 APP 组根据弹性调度算法进行垂直扩缩操作，不影响其他组的运行。

水平调度： 对 APP 层整体服务能力进行评估，当能力变化值大于一个组的服务能力时，需要进行水平扩缩操作，以组为单位进行水平扩缩。

3. 问题

1) Marathon Exit 容器清理

弹性扩缩容会导致宿主机上产生大量的 Exit 状态的 Docker 容器，清除时较消耗资

源，影响系统稳定性。默认 Mesos 只有基于时长的清除策略，比如几小时，几天后清除，没有基于指定时间的清除策略，比如在系统闲时清除，也无法针对每个服务定制清除策略。

解决方案：修改 Marathon 的源码程序，添加 Docker 容器垃圾清理接口，可以对不同服务按指定策略将 Exit 状态的 Docker 容器进行清理。

2) Docker DeviceMapper dm-thin 问题

在 CentOS6.5 上，我们发现 Docker 在使用 DeviceMapper 时会不定时出现 Linux Kernel Crash 的情况。

解决方案：通过修改 dm-thin.c 内核源码修复。

后续计划

通过将公司两大核心系统迁移到 DCOS，对于使用 Mesos 和 Docker 来构建企业私有云的弹性计算平台得到了充分的验证，后续将继续完善弹性调度功能、复杂应用编排、持续集成等能力。同时对 Kubernetes、Swarm 与 Mesos 的集成方案进行跟踪、测试和比较，构建高效稳定的 DCOS 平台能力。

2

蘑菇街 11.11：私有云平台的 Docker 应用实践

作者 **郭嘉**

对于蘑菇街而言，每年的 11.11 已经成为一年中最大的考验，考验的是系统稳定性，容灾能力，紧急故障处理，运维等各个方面的能力。蘑菇街的私有云平台，从无到有，已经经过了近一年的发展，生产环境上经历了 3 次大促，稳定性方面得到了初步验证。本文我将从架构、技术选型、应用等角度来谈谈蘑菇街的私有云平台。

蘑菇街的私有云平台（以下简称蘑菇街私有云）是蘑菇街面向内部上层业务提供的基础性平台。通过基础设施的服务化和平台化，可以使上层业务能够更加专注在业务自身，而不是关心底层运行环境的差异性。它通过基于 Docker 的 CaaS 层和 KVM 的 IaaS 层来为上层提供 IaaS/PaaS 层的云服务，以提高物理资源的利用率，以及业务部署和交付的效率，并促进应用架构的拆分和微服务化。

在架构选型的时候，我们觉得 Docker 的轻量化，秒级启动，标准化的打包 / 部署 / 运行的方案，镜像的快速分发，基于镜像的灰度发布等特性，都十分适合我们的应用场景。而 Docker 自身的集群管理能力在当时条件下还很不成熟，因此我们没有选择刚出现的 Swarm，而是用了业界最成熟的 OpenStack，这样能同时管理 Docker 和 KVM 虚拟机。相对来说，Docker 适合于无状态，分布式的业务，KVM 适合对安全性，隔离性要求更高的业务。

对于上层业务来说，它不需要关心是运行在容器中，还是 KVM 虚拟机里。今后的

思路是应用的微服务化，把上层的业务进行拆分，变成一个个微服务，从而对接 PaaS 基于容器的部署和灰度发布。

技术架构

在介绍双十一的准备工作之前，我先简单介绍一下蘑菇街私有云的技术架构。

我们采用的是 OpenStack＋novadocker＋Docker 的架构模式，novadocker 是 StackForge 上一个开源项目，它做为 nova 的一个插件，通过调用 Docker 的 RESTful 接口来控制容器的启停等动作。每个 Docker 就是所谓的“胖容器”，它会有独立的 IP 地址，通过 supervisord 来管理容器内的子进程，常见的如 SSHD、监控 agent 等进程。

我们在 IaaS 的基础上自研了 PaaS 层的编排调度等组件，实现了应用的弹性伸缩、灰度升级，支持一定的调度策略。我们通过 Docker 和 Jenkins 实现了持续集成（CI）。Git 中的项目如果发生了 git push 等动作，便会触发 Jenkins Job 进行自动构建，如果构

建成功便会生成 Docker Image 并 push 到镜像仓库。基于 CI 生成的 Docker Image，可以通过 PaaS 的 API 或界面，进行开发测试环境的实例更新，并最终进行生产环境的实例更新，从而实现持续集成和持续交付。

网络方面，我们没有采用 Docker 默认提供的 NAT 网络模式，NAT 会造成一定的性能损失。通过 OpenStack，我们支持 Linux bridge 和 openvswitch，不需要启动 iptables，Docker 的性能接近物理机的 95%。

准备工作

稳定性

迎战双 11，最重要的当然是确保稳定性。通过近一年的产品化和实际使用，我们积累了丰富的提高稳定性的经验。

对于那些已遇到过的问题，需要及时采用各种方式进行解决或者规避。

比如说，CentOS6.5 对 network namespace 支持不好，在 Docker 容器内创建 Linux bridge 会导致内核 crash，upstream 在 2.6.32-504 中修复了这个 bug，因此线上集群的内核版本，必须升级至 2.6.32－504 或以上。

又比如，CentOS6.5 自带的 device mapper 存在 dm-thin discard 导致内核可能随机 crash，这个问题我们早在四月份的时候已经发现并解决了，解决的办法是关闭 discard support，在 docker 配置中添加“--storage-opt dm.mountopt=nodiscard --storage-opt dm.blkdiscard=false”，并且严格禁止磁盘超配，因为磁盘超配可能会导致整个 device mapper 无法分配磁盘空间，而把整个文件系统变成只读，从而引起严重问题。

监控

我们在双 11 前重点加强的是针对容器的监控。

在此之前，我们已经自研了一套 container tools。主要功能有两个：一是能够以容

器为粒度计算 load 值，可以根据 load 值进行容器粒度的 qps 限流。二是替换了原有的 top、free、iostat、uptime 等命令，确保运维在容器内使用常用命令时看到的是容器的值，而不是整个物理机的值。双十一之后我们还会把 lxcfs 移植到我们的平台上来。

在宿主机上，我们增加了**多维度**的阈值监控和报警，包括对关键进程的存活性监控／语义监控，内核日志的监控，实时 pid 数量的监控，网络连接跟踪数的监控，容器 oom 的监控报警等等。

实时 pid 数量监控

为什么要监控实时的 pid 数量呢？因为目前的 Linux 内核对 pid 的隔离性支持是不完善的。还没有任何 Linux 发行版能做到针对 pid 按照容器粒度进行 pid_max 限制。

曾经发生过一个真实的案例是：某个用户写的程序有 bug，创建的线程没有及时回收，容器中产生了大量的线程，最后在宿主机上都无法执行命令或者 ssh 登陆，报的错是"bash: fork: Cannot allocate memory"，但是此时通过 free 命令看到空闲的内存却是足够的。

为什么会这样呢？根本原因是内核中的 pid_max(/proc/sys/kernel/pid_max)是全局共享的。当一个容器中的 pid 数目达到上限 32768，会导致宿主机和其他容器无法创建新的进程。最新的 4.3-rc1 才支持对每个容器进行 pid_max 的限制。

内存使用监控

值得一提的是，我们发现 cgroup 提供的内存使用值是不准确的，比真实使用的内存值要低。因为内核 memcg 无法回收 slab cache，也不对 dirty cache 量进行限制，所以很难估算容器真实的内存使用情况。曾经发生过统计的内存使用率一到 70-80%，就发生 OOM 的情况。为此，我们调整了容器内存的计算算法，根据经验值，将 cache 的 40% 算做 rss，调整后的内存计算比之前精确了不少。

日志乱序

还有一个问题是跑 Docker 的宿主机内核日志中经常会产生字符乱序，这样会导致日志监控无法取到正确的关键字进行报警。

经过分析发现，这个跟我们在宿主机和 Docker 容器中都跑了 rsyslogd 有关。由于内核中只有一个 log_buf 缓冲区，所有 printk 打印的日志先放到这个缓冲区中，docker host 以及 container 上的 rsyslogd 都会通过 syslog 从 kernel 的 log_buf 缓冲区中取日志，导致日志混乱。通过修改 container 里的 rsyslog 配置，只让宿主机去读 kernel 日志，就能解决这个问题。

隔离开关

平时我们的容器是严格隔离的，我们做的隔离包括 CPU、内存和磁盘 IO，网络 IO 等。但双十一的业务量可能是平时的十几倍或几十倍。我们为双十一做了不少开关，在压力大的情况下，我们可以为个别容器进行动态的 CPU，内存等扩容或缩容，调整甚至放开磁盘 iops 限速和网络的 TC 限速。

健康监测

我们还开发了定期的健康监测，定期的扫描线上可能存在的潜在风险，真正做到提前发现问题，解决问题。

灾备和紧急故障处理

除了稳定性，灾备能力也是必须的，我们做了大量的灾备预案和技术准备。比如我们研究了不启动 Docker Daemon 的情况下，离线恢复 Docker 中数据的方法。具体来说，是用 dmsetup create 命令创建一个临时的 dm 设备，映射到 Docker 实例所用的 dm 设备

号，通过 mount 这个临时设备，就可以恢复出原来的数据。

我们还支持 Docker 容器的冷迁移。通过管理平台的界面可以一键实现跨物理机的迁移。

与已有运维系统的对接

Docker 集群必须能与现有的运维系统无缝对接，才能快速响应，真正做到秒级的弹性扩容 / 缩容。我们有统一的容器管理平台，实现对多个 Docker 集群的管理，从下发指令到完成容器的创建可以在 7 秒内完成。

性能优化

我们从系统层面也对 docker 做了大量的优化，比如针对磁盘 IO 的性能瓶颈，我们调优了像 vm.dirty_expire_centisecs,vm.dirty_writeback_centisecs, vm.extra_free_kbytes 这样的内核参数。还引入了 Facebook 的开源软件 flashcache，将 SSD 作为 cache，显著的提高 docker 容器的 IO 性能。

我们还通过合并镜像层次来优化 docker pull 镜像的时间。在 docker pull 时，每一层校验的耗时很长，通过减小层数，不仅大小变小，docker pull 时间也大幅缩短。

镜像	文件层数	文件大小	docker pull 时间
原镜像	13	1.051 GB	2m13
新镜像	1	674.4 MB	0m26

总结

总的来说，双 11 是对蘑菇街私有云的一次年终大考，对此我们已有了充分的准备。随着 Docker 集群部署的规模越来越大，我们还有很多技术难题有待解决，包括容器本身的隔离性问题，集群的弹性调度问题等等。同时我们也很关注 Docker 相关的开源软件 Kubernetes、Mesos、Hyper、criu、runC 的发展，未来将引入容器的热迁移，Docker Daemon 的热升级等特性。

如果大家想了解更多的话，可以关注我们的技术博客 http://mogu.io/。蘑菇街期待你的加入，一起来建设大规模的 Docker 集群。

作者介绍

郭嘉，蘑菇街平台技术部架构师，虚拟化组负责人。2014 年加入蘑菇街，目前主要专注于蘑菇街的私有云建设，关注 Docker、KVM、OpenStack、Kubernetes 等领域。邮箱：guojia@mogujie.com。

3

Apple 使用 Apache Mesos 重建 Siri 后端服务

作者 **Daniel Bryant** ，译者 **韩陆**

苹果公司宣布，将使用开源的集群管理软件 Apache Mesos，作为该公司广受欢迎的、基于 iOS 的智能个人助理软件 Siri 的后端服务。Mesosphere 的博客指出，苹果已经创建了一个命名为 J.A.R.V.I.S.，类似 PaaS 的专有调度 Framework，由此，开发者可以部署可伸缩和高可用的的 Siri 服务。

集群管理软件 Apache Mesos 将 CPU、内存、存储介质以及其它计算机资源从物理机或者虚拟机中抽象出来，构建支持容错和弹性的分布式系统，并提供高效的运行能力。Mesos 使用与 Linux 内核相同的系统构建原则，只是他们处在不同的抽象层次上。Mesos 内核运行在每台机器上，通过应用程序 Framework，提供跨整个数据中心和云环境进行资源管理和调度的 API。苹果已经创建了自己专有的调度 Framework 以运行 Siri 的后端服务，将其命名为 J.A.R.V.I.S.。

J.A.R.V.I.S.是“一种相当智能的调度器（Just A Rather Very Intelligent Scheduler）”的缩写，这个名字的灵感来自《钢铁侠》电影中的智能化计算机助手。苹果公司使用 J.A.R.V.I.S.作为内部的平台即服务（PaaS）系统，使开发者编写的 Siri 后端应用程序可以部署为可伸缩性和弹性的服务，用于响应 iOS 用户通过个人助理应用程序请求的语音查询。

据 Mesosphere 的博客报道，在苹果公司总部加州库比蒂诺的聚会上，苹果的开发者表示，他们的 Mesos 集群有数千个节点。支持 Siri 应用程序的后台系统包括约 100

种不同类型的服务，应用程序的数据存储在 Hadoop 分布式文件系统（HDFS）中。从基础设施的角度来看，使用 Mesos 有助于使 Siri 具备可伸缩性和可用性，并且还改善了 iOS 应用程序自身的延迟。

Mesos 后端是第三代 Siri 平台，告别了之前部署在“传统的”基础设施的历史。Mesosphere 博客认为，从概念上讲，苹果公司与 Mesos 的合作以及 J.A.R.V.I.S.类似于 Google 的 Borg 项目，领先于其他支持长时间运行应用服务的类 PaaS Framework，比如 Mesospere 数据中心操作系统（DCOS）的相关组件 Mesosphere Marathon 和出自 Twitter 基础设施团队的 Apache Aurora。

Mesosphere 高级研究分析师 Derrick Harris 在 Mesosphere 的博客中表示，关于 Siri 由 Apache Mesos 集群管理软件支撑的公告是对 Mesos 成熟度的证明：

苹果公司能够信任使用 Mesos 支撑 Siri——这是一个复杂的应用程序，用以处理只有苹果知道每天会有多少数量的、来自数以亿计的 iPhone 和 iPad 用户的语音查询。这足以说明 Mesos 的成熟度，Mesos 已经为各种类型的企业带来巨大影响做好了准备。

InfoQ 采访了 Mesosphere 高级副总裁 Matt Trifiro，并询问了这项公告对正在考虑部署应用到 Mesos 的企业和软件开发者会有什么影响。

InfoQ：为什么苹果的这项公告对 Mesos 和 Mesosphere 很重要？

Trifiro：苹果公司宣布，他们完全重建了 Siri，以运行于 Mesos 之上。这再次表明，Mesosphere DCOS 中的分布式内核 Mesos，是编排大规模容器和构建新的分布式系统的黄金标准。

InfoQ：不是每家企业都能达到苹果公司的规模，那么传统企业怎样应用 Mesos 呢？

Trifiro：像苹果和 Twitter 这样的公司，几乎全部的基础设施都使用了这项技术。因为 Siri 和 Twitter 都依赖于 Mesos，可想而知，它必须是可靠的。但是，开源的 Apache Mesos 是一项非常尖端的技术，通过开源工具手工装配，并将 Mesos 用于生产环境是非常困难的。这正是 Mesosphere 产生的原因。任何公司都能使用这项久经考验的技术，

构建完整的数据中心操作系统（DCOS），并具备和 Twitter 或者苹果公司同等的能力和自动化效果，而不必成为 Twitter 或者苹果那样大规模的公司。

InfoQ：苹果公司从 Mesos API 直接实现了一套调度器（J.A.R.V.I.S.），这意味着什么呢？

Trifiro：Mesos 最强大的方面其一就是，它提供了用于构建新的分布式系统的基本功能。如果你去看其它的分布式系统，比如早于 Mesos 出现的 Hadoop，它有几十万行代码，很多地方是在重复制造轮子。所有的失败处理、网络实现、消息传递和资源分配的代码，开发者不应重写这些功能。而为程序员提供了内置这些功能的 Mesos 内核的话，他们就可以快速构建新的高可用性和弹性分布式系统，而无需重复所有基本的功能。他们可以专注于业务逻辑的实现上。

InfoQ：Mesos 和 Mesosphere DCOS 之间是什么关系？

Trifiro：Mesosphere DCOS 是一种新型的操作系统，跨越数据中心或云环境中的所有机器，将他们的资源放到一个资源池中，使他们的行为整体上像一个大的计算机。Apache 的开源项目 Mesos 是这个操作系统里面的内核。我们将其和其他组件包装到一起，包括初始化系统（marathon）、文件系统（HDFS）、应用打包和部署系统、图形用户界面和命令行界面（CLI）。所有这些组件一起构成了 DCOS。这就像苹果公司的 Yosemite 操作系统或者像 Android，他们各有一个内核（分别是 BSD 和 Linux），他们为内核添加了系统服务和工具，使内核成为值得笔记本电脑或者智能手机使用的产品。我们为数据中心所做的工作也是相同的。

更多关于苹果公司宣布使用 Mesos 作为 Siri 后端服务的消息，详见 Mesosphere 的博客。

查看英文原文：Apple Rebuilds Siri Backend Services Using Apache Mesos

4

阿里百川：全架构 PaaS TAE 2.0 的 Docker 实践

作者 **徐川**

随着 Docker 及容器技术的火热发展，PaaS 进入了新时代，最近一些传统 PaaS 开始基于 Docker 及类似技术进行升级改造，也出现了一些专门提供容器托管的平台。淘宝应用引擎 TAE 是国内较早规模化使用 Docker 的 PaaS 平台，它最近推出了 2.0 版，作为阿里百川项目的一部分对外开放。阿里百川是阿里巴巴集团无线开放平台，为移动开发者（涵盖移动创业者）提供快速搭建 APP、加速 APP 商业化、提升用户体验的解决方案。

InfoQ 记者采访了阿里百川的技术负责人云动，探讨了 TAE 在 Docker 和容器技术方面的实践，以及在 PaaS 方面取得的成果。

InfoQ：请介绍一下 TAE 项目的开发历程?

云动：TAE 最开始是服务于店铺开放业务，为了让第三方开发者参与到卖家店铺的装修和页面设计，同时要保证整体安全、性能和稳定性，孵化出淘宝应用引擎服务：TAE（Taobao App Engine），同时带有安全，受控容器的特点。2014 年起，为了更好的服务到阿里百川的移动开发者，TAE 基于 Docker 容器技术的升级，针对移动开发者团队小，迭代快的特点，推出 TAE 2.0 全架构 PaaS，从开发者的系统构建，代码发布到系统运维管理一整套的解决方案。

InfoQ：TAE 2.0 的全架构 PaaS 是什么意思？有哪些特性？

云动：类似 GAE、SAE 这样的传统 PaaS 服务，具备弹性伸缩，分布式，免运维等

优点，但用户想要使用到这些特性，都需要对原有的系统代码做改造，所以传统 PaaS 的门槛比较高，并且不够灵活，开发者很难扩展自己的服务。

另外传统 PaaS 一般只支持 Web+数据库这种两层结构，这个比较适合 Web 网站应用。一旦涉及到更复杂一些的架构，比如 Web+消息队列+缓存+数据库+数据分析的系统架构，甚至 C++或 NodeJS 的服务，传统 PaaS 很难满足。全架构的 PaaS 就是突破传统 PaaS 的这些瓶颈，让开发者的任意系统都能在 TAE 中搭建起来。

首先，TAE 2.0 在 PaaS 功能上，在业界应该是最强的，从开发者系统迁移到 TAE 的工具化支持，到系统可视化创建，到系统可视化发布运维，以及线上系统的监控、系统性能 Profile、日志、压测、慢 SQL 分析、数据源分析，WebIDE 代码管理，定时任务，多媒体云存储和加速，能够覆盖到开发者从 0 到数百万 DAU 成长过程的技术需求。其次，基于 Docker 给开发者带来很好的扩展性，比如开发者可以将自己制作的镜像部署到 TAE 上，也可以通过我们的 WebSSH 或原生 SSH 登录到 Container 中像 IaaS 一样管理容器代码。一句话说，TAE 2.0 的全架构 PaaS 就是兼具 IaaS 的灵活性和 PaaS 的方便易运维。

InfoQ：您提到 TAE 2.0 使用了 Docker 技术，能否讲讲 TAE 采用 Docker 的历程？

云动：TAE 诞生于淘宝店铺开放业务，之前一直在服务淘系内部业务，比如优站，微淘插件，手淘开放等业务，开发者主要的使用场景就是 Web+数据库的两层结构。而百川移动开发者，往往是三层结构甚至 N 层结构。原来的 Web PaaS 很难满足用户了。另外我们发现，现在的移动初创团队，团队小，迭代快，他们迫切需要有一套系统能够像 PaaS 一样去运维他们的生产环境，但是不希望为了使用 PaaS 而修改自己的代码，并且有 IaaS 的灵活性。基于这样的需求，所以我们比较自然的想到到了用 PaaS + Container based IaaS 的方案来满足用户需求，这个正好和 Docker 的的特点比较吻合，所以我们选择了 Docker。

TAE 是从去年 8 月份开始尝试使用 Docker 技术的，经过了近 1 年的产品化和实际使用，目前已经比较稳定了，我们计划七月份正式对外公测。从产品和技术的应用来讲，

TAE 算是比较早的规模化使用 Docker 的 PaaS 平台。

InfoQ：TAE 在容器方面做了哪些优化和调整？

云动：我们在容器的网络，存储，安全上都做了很大的优化和调整：

在安全上，我们已经比较好的解决了 Container 的多租户问题，通过三层安全防护体系来保证用户的多租户安全。

在网络虚拟化上，TAE 底层是阿里云的 ECS，为了解决 Docker 容器的网络互通，我们和阿里云的 VPC（Virtual Private Cloud）团队做了很多网络虚拟化的共建，利用 VPC 来支持 Docker 的网络虚拟化。

在存储虚拟化上，基于阿里云底层的盘古分布式存储，我们可以给 Docker 提供高可用的分布式磁盘产品。

另外，Docker 现在发展得非常快，在存储，网络，热迁移，系统稳定性上，还有大量的问题需要解决。我们在解决 Docker 的网络性能，Docker Daemon 的稳定性等方面，我们做了很多工作。针对不同的镜像服务我们提供了很多产品化的功能，比如 Web 服务的多环境发布，存储服务的主备同步和切换，配置文件的可视化管理，容器日志收集，Docker Web 镜像 Build 环境等。

InfoQ：TAE 2.0 和业界其它 PaaS 产品有什么不同，它的核心竞争力是什么？

云动：TAE 2.0 是针对移动场景的定制化 PaaS，我们除了能够支持上面提到的复杂架构的系统搭建，并且能够做到用户代码零改动，就可以享受到 PaaS 的服务。

具体一些：TAE 支持开发者的系统构建（Build），发布（Deploy），运维管理（Management）三大块功能：

系统构建：基于 Docker 的镜像仓库来实现的，我们官方目前支持了包括 Web 服务，存储服务，消息队列，负载均衡，搜索等等 14 个镜像。如果官方镜像无法满足需求，用户还可以将自己在线制作或者线下制作的镜像，提交到镜像仓库。用户搭建系统，通过 TAE 的可视化界面即可完成参数配置、实例选择、接入层设置等，发布上线。

发布系统：包括打包发布，Git/SVN 发布，历史版本回滚，灰度发布，Beta 发布等

一些列的发布功能，并且 PHP 还能够单文件发布，发布过程中，能够基于健康检查，按批次发布，做到随时发布对用户无影响。

在运维管理方面，基于淘宝多年的系统容量规划、性能和稳定性经验，TAE 提供 应用监控（QPS，RT，URI），容器监控（CPU,内存，带宽）， 数据库监控等一些列的监控功能。还提供日志采集，搜索，查询，实时日志显示，监控日志告警联动，自定义监控，在线 Debug，Web JStack 等功能，保证能够最快的帮助用户定位到系统的问题在哪里。可以这么说，使用 TAE，就可以使用淘宝内部的软件开发流程，来规范化开发和系统运维过程。

另外我们还支持在线 IDE，可以在线编辑，打包发布。支持 WebSSH 登录 Container 管理容器，并且支持传统的 SSH 客户端登录。当然传统的弹性伸缩，故障迁移等特性也支持。

TAE 2.0 的特点可以概括为：像使用 PaaS 一样运维管理系统，还可以像 IaaS 一样登录到服务器去查看管理 Container。开发者不需要学习 Docker，就能够将他的系统在 TAE 上搭建成功。如果开发者懂 Docker，也可以把 TAE 当成是一个 Container AS A Service 的容器服务提供商，管理和运维他的 Docker Container。

和传统 PaaS 或最近新的 CaaS 竞品相比，TAE 已经成熟的服务了大量的百川开发者，并且对客户的需求和痛点，有一整套的解决方案，TAE 的愿景，就是让 3,5 个人的小团队，能够轻松服务海量用户，这就是 TAE 对于开发者的核心价值，也是 TAE 的核心竞争力。

举个实际的例子，我们有一个做母婴社区 APP 的客户，30 多万 DAU 的用户，他们的架构比较典型，既包括 Java 的前台和 PHP 的后台，也包括 ActiveMQ，ZooKeeper 等中间件，并且使用 Dubbo 提供分布式服务，用 Hadoop 做日志和业务数据分析，存储包括 MongoDB，MySQL，Redis 等开源解决方案。他们只用了半天时间，就将他们 8 个应用在 TAE2.0 上搭建成功，用一周时间，完成数据迁移和压测，将他们系统、数据、流量切换到 TAE 平台。迁移后，使用 TAE 的可视化发布，随时回滚，可视化监控和报

警，日志联动，线上问题排查工具等功能做 App 的发布和运维工作，这是开发者用 IaaS 的时候不敢想的。

InfoQ：TAE 在整个阿里百川里是个什么样的定位，跟 ACE 是什么关系，未来是否会独立向外提供服务？

云动：TAE 是从淘宝业务中成长起来的一个云计算服务，同时，又在淘系业务的安全开放上，积累了很多的经验。阿里百川是阿里巴巴针对移动互联网的开放平台，TAE 提供了阿里百川的云服务托管解决方案，我们希望 TAE 能够帮助到创业者，快速完成从零到一的初创期过程，在开发者成长期，TAE 提供的运维、监控、弹性等特性，能够解决开发者的痛点。并且针对的初创团队，百川有部分云资源免费两年的策略，扶持到中小创业者。

和 ACE 的关系，TAE 和 ACE 两个团队合作得非常紧密，可以说是共用一套底层阿里云的基础设施，ACE 是面向公有云用户，而 TAE 是面向移动开发者用户，所以有不同的产品形态。

关于 TAE 开放给公有云用户，我们也在和阿里云计算同学在推进，希望能够基于 TAE 目前的产品，给公有云 ECS 的用户，提供 Docker Container Service 的服务，除了百川开发者，让更多的开发者能够 TAE 的功能。并且通过阿里云镜像市场，从虚拟机镜像向 Docker 镜像转型，形成一个技术闭环的云计算生态环境。

受访嘉宾介绍

云动（真名：陈思淼），阿里巴巴资深技术专家，2008 年加入淘宝网，参与淘宝从集中式系统到大规模分布式系统的演变过程，先后参与淘宝用户中心建立，淘宝商品体系重构，淘宝交易系统拆分，以及稳定性和性能的优化和重构，先后参与了四次双十一技术保障工作。从 2011 年起，负责淘宝&天猫旺铺系统，主导了店铺装修系统，店铺对外开放体系，店铺第三方插件体系等技术上改造工作。从 2014 年起，担任阿里百川的技术负责人。

5

大众点评容器云平台：运营超一年，承载大部分业务

作者 **郭蕾**

Docker 容器的应用场景之一就是构建企业私有云平台，它简单、轻量的特点不仅可以降低私有云的构建难度，而且还能更高效的利用物理资源。同时，容器在安全性方面的短板恰恰因为私有云的应用场景而显得不再重要。但基于容器的私有云也有不少挑战，比如网络、监控、用户习惯等方面都需要有好的解决方案。大众点评在 2014 年 7 月就基于 Docker 搭建了私有云平台，目前平台承担了大部分的线上的业务，实例数 2800 个左右，Docker 物理集群 300 多台。InfoQ 记者采访了大众点评云平台首席架构师盛延敏，听他分享大众点评私有云平台实战过程中踩过的坑和经验教训。另外，盛延敏还将在 8 月 28 日~29 日举办的 CNUTCon 全球容器技术大会上分享题为《大众点评的容器私有云实战》的演讲，敬请关注。

InfoQ：点评是从什么时候开始使用 Docker 构建私有云平台的？未使用容器云之前，点评内部有基于其他技术搭建私有云吗？基于 Docker 的私有云会有哪些优势？有哪些挑战？

盛延敏：我们从 2014 年开始基于 Docker 搭建点评的私有云平台。原先点评的应用都是部署在 KVM 上的，大量重复性的操作耗费了运维同学不少的时间和精力，特别是扩容和缩容这块。所以就萌发了使用 Docker 搭建一个私有云平台的想法，借助云平台，

将应用标准化，运维自动化。从目前使用情况来看，基本达到了原先设想的目标。

未使用容器云之前，点评内部谈不上使用过私有云。从实践来看，基于 Docker 的私有云可以带来更高效的物理资源使用率，更快的部署以及更加统一规范的标准化运行环境。较虚拟化云，其优势还是很明显的。关于挑战，我的主要体会是安全和规范，Docker 还不能做到完全的多租户环境，这一点对于安全敏感的企业级应用是一个不小的挑战。不过，对于私有云而言，这一点倒不是一个致命的短板。另外一个从实践来看，大家对于从 KVM 过渡到 Docker 的使用环境，使用习惯也是一个不小的挑战。举个例子，在 KVM 时代，应用上线前是一定要分配一些 KVM 实例的，这样可以加速应用上线发布的速度。到了 Docker 时代，Docker 可以支持实例秒级创建，是不是还一定要预先分配实例？这其实是一个使用习惯的问题。

InfoQ：你们的容器云平台使用了哪些开源的技术栈？

盛延敏：底层使用了 Docker，通过 Dockerfile、Docker Registry 统一管理应用的标准化运行环境。这一块是整个容器云的基石。

组件之间的交互使用了 NATS，通过消息的『发布-订阅』模型，将各个组件之间的耦合最小化。这也是参考和学习了 Cloud Foundry 公有云架构。

接入层使用了 Nginx 和 ZooKeeper，对于 Web 类型的应用，通过和 Nginx 暴露的 Restful 接口交互，完成实例在集群内的注册以及注销。对于服务类型的应用，通过在 ZooKeeper 上注册和注销服务 IP 和端口，便于服务客户端发现和更新该服务。

InfoQ：从我的理解来看，基于容器的私有云，需要用户了解 Docker 等技术，并且容器推荐的是使用微服务架构。那你们在内部推进这样的私有云是否困难？架构方面的问题如何解决？

盛延敏：你说的很对。所以我们从一开始就制定了合适的目标，其中最重要的是不要让用户感觉到 KVM 切换到 Docker 这个转换，在这个目标基础上我们做了大量的工作。比如，用户并不需要自己了解 Dockerfile，我们使用 Dockerfile 来定制应用运行的环境，如果需要更新运行时环境，也是由云平台团队来维护。如果你需要 Node.js 的环

境，我们可以帮助定制一个 Node.js 的镜像。再比如，我们更改了 Docker 的代码，让其从推荐的微服务架构演变到目前的『虚拟机』架构。开发和运维可以通过 IP 直接访问到 Docker『虚拟机』，基于 IP 的应用基础架构也不需要开发和运维做剧烈的改变。通过这些方法，加上公司内部的支持，逐渐推动了私有云平台的加速建设。

InfoQ：目前有哪些业务跑在容器上了？怎么样的业务适合容器云？

盛延敏：基本上除搜索和数据库以外，点评现有的业务大多都有跑到容器上的。当然覆盖率还在提升当中。只要是可以标准化的业务，经过改造基本都可以跑到容器云上。当然这里面也有一些特殊情况，例如 IP 本身是不是对外暴露的，是否可以支持横向扩展等等。

InfoQ：监控、网络方面的问题是如何解决的？

盛延敏：监控点评有业绩比较有名的 CAT（Central Application Tracking），所以我们的监控也是接入 CAT 的。通过收集 CGroups 和 Docker 实例的实时信息，将内存、CPU、网络等源源不断的上报到 CAT。再由 CAT 提供查询，检索和展示。当然，也支持告警。

关于网络，主要是采用了 Linux bridge 工作在 level 2 的模式，使公共 IP 得以暴露出来，这部分我们是做了定制的。

InfoQ：目前私有云的有哪些基础功能？可以做什么，不可以做什么？接下来有什么规划？

盛延敏：主要的基础功能包括实例的创建和销毁、缩容和扩容，和 DevOps 工具集成等等。接下来会把精力放到整个平台的演进上，包括模块的进一步拆分，新的镜像的支持，和运维工具 Puppet 的集成等等。

InfoQ：这么长时间的应用，有做过复盘吗？未来有什么计划？

盛延敏：我们还是复盘过的，其实我们自己觉得也比较幸运，刚开始决定做 PaaS 的时候，正是 Docker 技术出现并繁荣的时候，在 Docker 上构建整个私有云平台是一个比较明智的决定。更早一点的话，说不定就使用 LXC 或者 Warden 了。未来的计划还

是跟公司的业务需求紧密结合，最大能力的发挥 PaaS+Docker 的威力。

InfoQ：你将在 CNUTCon 上分享哪些内容？

盛延敏：上述的内容都会有涉及到。另外，还有大量我们的『干货』分享，都是我们曾经踩过的坑哦。欢迎大家捧场。谢谢！

6

京东 618：Docker 扛大旗，弹性伸缩成重点

作者 **郭蕾**

不知不觉中，年中的 618 和年终的 11.11 已经成为中国电商的两大促销日，当然，这两天也是一年中系统访问压力最大的两天。对于京东而言，618 更是这一年中最大的一次考试，考点是系统的扩展性、稳定性、容灾能力、运维能力、紧急故障处理能力。弹性计算云是京东 2015 年研发部战略项目，它基于 Docker 简化了应用的部署和扩容，提高了系统的伸缩能力。目前京东的图片系统、单品页、频道页、风控系统、缓存、登录、团购、O2O、无线、拍拍等业务都已经运行在弹性计算云系统中。

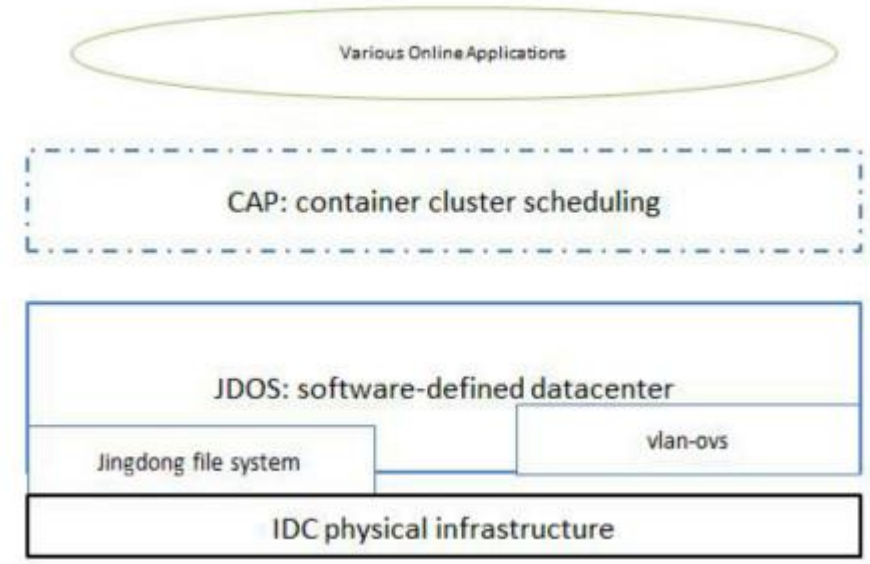
过去的一段时间里，弹性计算云项目在京东内部获得了广泛应用，并且日趋稳定成熟。一方面，这个项目可以更有效地管理机器资源，提高资源利用率；另外还能大幅提高生产效率，让原来的申请机器上线扩容逐渐过渡到全自动维护。京东弹性计算云项目将深刻影响京东未来几年的基础架构。

InfoQ：能否介绍下京东弹性计算云项目的情况，你们什么时候开始使用 Docker 的？目前有多大的规模？

刘海锋：弹性计算云项目在去年第四季度开始研发，今年春节后正式启动推广应用。经过半年多的发展，逐渐做到了一定规模。截至 6 月 17 日，我们线上运行了 9853 个 Docker 实例（注：无任何夸大）以及几百个 KVM 虚拟机。京东主要的一些核心应用比如商品详情页、图片展现、秒杀、配送员订单详情等等都部署在弹性云中。弹性计算云

项目也作为今年 618 的扩容与灾备资源池，这估计是国内甚至世界上最大规模的 Docker 应用之一。随着业务的发展以及 IDC 的增加，预计今年年底规模会翻两番，京东大部分应用程序都会通过容器技术来发布和管理。

系统架构可以这样简洁定义：弹性计算云 = 软件定义数据中心 + 容器集群调度。整个项目分成两层架构，底层为基础平台，系统名 JDOS，通过『OpenStack married with Docker』来实现基础设施资源的软件管理，Docker 取代 VM 成为一等公民，但这个系统目标是统一生产物理机、虚拟机与轻量容器；上层为应用平台，系统名 CAP，集成部署监控日志等工具链，实现『无需申请服务器，直接上线』，并进行业务特定的、数据驱动的容器集群调度与弹性伸缩。



InfoQ：能否谈谈你们的 Docker 使用场景？在 618 这样的大促中，Docker 这样的容器有什么优势？618 中有哪些业务跑到 Docker 中？

刘海锋：目前主要有两类场景：无状态的应用程序，和缓存实例。这两类场景规模最大也最有收益。不同的场景具体需求不同，因此技术方案也不相同。内部我们称呼为“胖容器”与“瘦容器”技术。从资源抽象角度，前者带独立 IP 以及基础工具链如同一台主机，后者可以理解为物理机上面直接启动 cgroup 做资源控制加上镜像机制。

618 这样的大促备战，弹性计算云具备很多优势：非常便捷的上线部署、半自动或全自动的扩容。Docker 这样的操作系统级虚拟化技术，启动速度快，资源消耗低，非常适合私有云建设。

今年 618，是京东弹性计算云第一次大促亮相，支持了有很多业务的流量。比如图片展现 80%流量、单品页 50%流量、秒杀风控 85%流量、虚拟风控 50%流量，还有三级列表页、频道页、团购页、手机订单详情、配送员主页等等，还有全球购、O2O 等新业务。特别是，今年 618 作战指挥室大屏监控系统都是部署在弹性云上的。

InfoQ：你们是如何结合 Docker 和 OpenStack 的？网络问题是如何解决的？

刘海锋：我们深度定制 OpenStack，持续维护自己的分支，称之为 JDOS（the Jingdong Datacenter Operating System）。JDOS 目标很明确：统一管理和分配 Docker、VM、Bare Metal，保证稳定高性能。网络方面不玩复杂的，线上生产环境划分 VLANs + Open vSwitch。SDN 目前没有显著需求所以暂不投入应用。我们以『研以致用』为原则来指导技术选择和开发投入。

InfoQ：能否谈谈你们目前基于 Docker 的 workflow？

刘海锋：弹性计算平台集成了京东研发的统一工作平台（编译测试打包上线审批等）、自动部署、统一监控、统一日志、负载均衡、数据库授权，实现了应用一键部署，并且全流程处理应用接入，扩容、缩容、下线等操作。支持半自动与全自动。

InfoQ：这么多的容器，你们是如何调度的？

刘海锋：容器的调度由自主研发的 CAP（Cloud Application Platform）来控制，并会根据应用配置的策略来进行调度；在创建容器的时候，会根据规格、镜像、机房和交换机等策略来进行创建；创建完容器后，又会根据数据库策略、负载策略、监控策略等来进行注册；在弹性调度中，除了根据容器的资源情况，如 CPU 和连接数，还会接合应用的 TPS 性能等等来综合考虑，进行弹性伸缩。

目前已经针对两大类在线应用实现自动弹性调度，一是 Web 类应用，二是接入内部 SOA 框架的服务程序。大规模容器的自动化智能调度，我们仍在进一步研究与开发。

InfoQ：目前主要有哪些业务使用了 Docker？业务的选择方面有什么建议？

刘海锋：目前有 1000 个应用已经接入弹性云，涵盖京东各个业务线，包括很多核心应用。目前我们主要支持计算类业务，存储类应用主要应用到了缓存。数据库云服务也将通过 Docker 进行部署和管理。

特别强调的是，业务场景不同，技术方案就有差别。另外，有些对隔离和安全比较敏感的业务就分配 VM。技术无所谓优劣和新旧，技术以解决问题和创造业务价值为目的。

InfoQ：你们的缓存组件也跑在 Docker 中，这样做有什么好处？IO 什么的没有问题吗？有什么好的经验可以分享？

刘海锋：我们团队负责一个系统叫 JIMDB，京东统一的缓存与高速 NoSQL 服务，兼容 Redis 协议，后台保证高可用与横向扩展。系统规模增长到现在的三千多台大内存机器，日常的部署操作、版本管理成为最大痛点。通过引入 Docker，一键完成容器环境的缓存集群的全自动化搭建，大幅提升了系统运维效率。

技术上，缓存容器化的平台并不基于 OpenStack，而是基于 JIMDB 自身逻辑来开发。具体说来，系统会根据需求所描述的容量、副本数、机房、机架、权限等约束创建缓存容器集群，并同时在配置中心注册集群相关元数据描述信息，通过邮件形式向运维人员发出构建流水详单，并通知用户集群环境构建完成。调度方面，不仅会考虑容器内进程，容器所在机器以及容器本身当前的实时状况，还会对它们的历史状况进行考察。一旦缓存实例触发内存过大流量过高等扩容条件，系统会立即执行扩容任务创建新的容器分摊容量和流量，为保证服务质量，缓存实例只有在过去一段时间指标要求持续保持低位的情况下才会缩容。在弹性伸缩的过程中，会采用 Linux TC 相关技术保证缓存数据迁移速度。

InfoQ：使用过程中有哪些坑？你们有做哪些重点改进？

刘海锋：坑太多了，包括软件、硬件、操作系统内核、业务使用方式等等。底层关键改进印象中有两个方面：第一，Docker 本地存储结构，抛弃 Device Mapper、AUTFS

等选项，自行定制；第二，优化 Open vSwitch 性能。比如，优化 Docker 镜像结构，加入多层合并、压缩、分层 tag 等技术，并采用镜像预分发技术，可以做到秒级创建容器实例；优化 Open vSwitch 转发层，显著提升网络小包延迟。

受访嘉宾介绍

刘海锋，京东云平台首席架构师、系统技术部负责人。系统技术部专注于基础服务的自主研发与持续建设，包括分布式存储、以内存为中心的 NoSQL 服务、图片源站、内容分发网络、消息队列、内部 SOA 化、弹性计算云等核心系统，均大规模部署以支撑京东集团的众多业务。

7

腾讯游戏如何使用 Docker

作者 **郭蕾**

腾讯第一季度的财报显示，腾讯游戏的收入占腾讯总营收的 59.4%，很显然，腾讯游戏已经是腾讯最赚钱的部门，当然，腾讯也是国内最大的游戏发行商。游戏行业相对于其他行业来说特点非常明显，一是需要同时运营多款游戏，二是游戏业务的生命周期长短不一。不管是从数量还是周期来看，游戏行业特殊的业务都为技术团队提出了更高的要求。腾讯游戏从 2014 年下半年开始就在生产环境中使用 Docker，并取得了不错的成果。目前《我叫 MT2》等多款重量级游戏都跑在容器中，且整体运行良好。在 8 月 28 日的 CNUTCon 全球容器技术大会上，腾讯游戏的高级工程师尹烨将会介绍腾讯游戏业务使用 Docker 的进展及收益，并从内核、网络、存储、运营等方面深入探讨腾讯游戏在实践过程中遇到的问题及解决方案，最后还会复盘反思 Docker 对于游戏业务的价值。本文是会前 InfoQ 记者对尹烨的采访。

InfoQ：腾讯游戏是什么时候开始使用 Docker 的？能介绍下目前的一些应用情况吗？

尹烨：我们是从 2014 年 6 月份开始接触 Docker，那时 Docker 在国内才刚刚开始兴起，了解的人还很少。Docker 让容器的管理变得非常简单，再加上创造性的分层镜像的技术，给人眼前一亮。我们希望通过 Docker，构建腾讯游戏内部的容器平台，一方面通过容器提高资源利用率，另一方面通过镜像分发技术标准化、统一化应用部署流程。

经过半年的调研、各种测试、系统设计和开发，14 年底，整个系统开始上线试运行。但是对于一项全新技术的应用，大家都很谨慎，因为很多游戏业务的在线玩家很多，我们的压力很大。第一个接入我们 Docker 平台的是腾讯的一款游戏，叫《QQ 宠物企鹅》。这款游戏的架构在容灾方面设计得很好，前端可平行扩展，所以就选它作为试金石了。跑了几个星期，运行正常，然后开始慢慢扩展到其它业务。

现在，我们的 Docker 平台上已经跑了数十款端游、页游和手游的各种游戏应用，特别是新上的手游业务，其中，我们代理的一款重量级手游《我叫 MT2》，一个业务就使用了 700 多个容器。现在整个平台总共有 700 多台物理机，3000 多个 Docker 容器。这个数字在业界并不算多，我们自己也没有刻意去追求数量，相对数量，我们更愿意以稳为先。目前，整个平台运行了大半年，整体运行良好。

InfoQ：与其它行业相比，游戏行业有什么特殊性？Docker 在这样的业务中有什么样的优势？它可以发挥怎么样的价值？

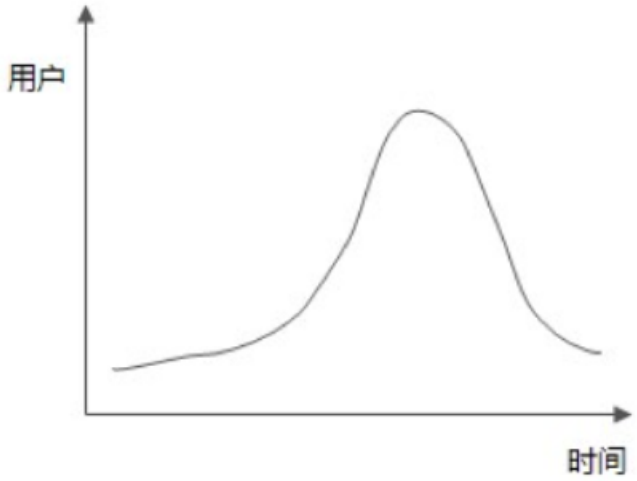
尹烨：相比于其它业务，一是游戏业务更加复杂多样，有端游、手游和页游，有的是分区分服，有的是全区全服；另外，我们又分自研和代理游戏，更加增加了复杂性。这也给业务的运维部署带来了许多不便，尽管我们内部有很成熟的部署平台。而 Docker 统一的镜像分发方式，可以标准化程序的分发部署，解放运维的生产力。特别是代理游戏，如果都以 image 方式交付，可以极大提高效率。

另一方面，一般来说，游戏业务的生命周期长短不一，这需要弹性的资源管理和交付。所以，腾讯游戏很早就开始使用 XEN/KVM 等虚拟机技。相比于虚拟机，容器更加轻量，效率更高，资源的交付和销毁更快。另外，还可以通过修改 cgroup 参数，在线调整容器的资源配额，更加灵活弹性。

InfoQ：腾讯游戏的 Docker 应用场景是怎么样的？

尹烨： Docker 开创性的提出了『Build、Ship、Run』的哲学。总的来看，现在主要有两种使用 Docker 的方式。一是基于 Docker 搭建 CI/CD 平台，重点放在 Build 和 Ship 上面，一般用于开发、测试环境；另外就是将 Docker 容器当作轻量级虚拟机，更

加关注 Run 的问题，大规模的用于生产环境。个人认为，这两种方式无所谓谁好谁坏，长远来看，二者会渐渐趋于统一。



游戏生命周期

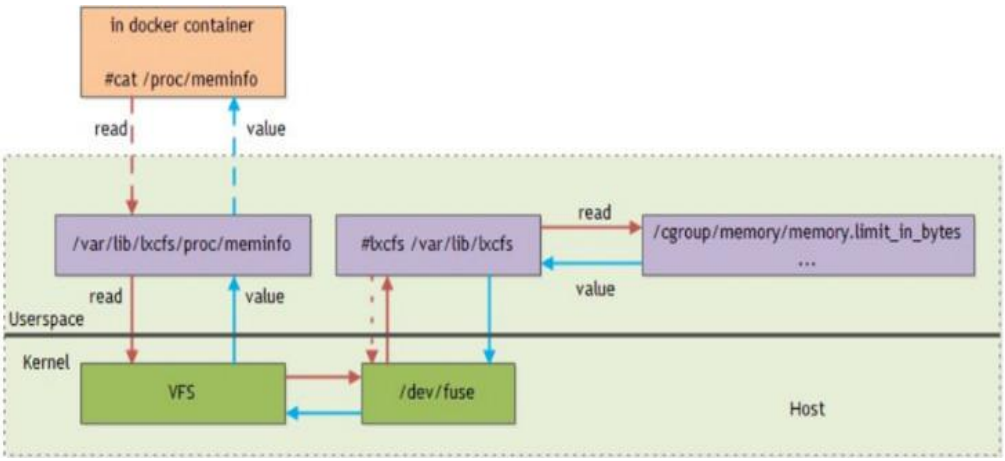
腾讯内部有很成熟的开发、部署工具和流程，我们作为平台支撑部门，去推动业务开发改变原来的开发模式需要的较长的时间周期。所以，我们现在更多的是将 Docker 容器作为轻量级的虚拟机来使用，我们在 Run 上面花了很多时间和精力。同时，我们也在探索通过 Image 方式去标准化业务部署流程。但是，我们不太会去做 CI/CD 的事情，我们更关注提供一个高效的容器资源调度管理平台，然后以 API 的方式对外，提供给开发和运维同学使用，比如，与互娱的蓝鲸平台打通。

InfoQ：能否介绍下你们线上的 Docker 集群所使用的技术栈？

尹烨：我们的使用 Docker 的初衷是替代虚拟机，所以我们直接将 Docker 跑在物理机上。我们使用 Docker 面临的第一个问题就是操作系统内核的问题。腾讯内部一般使用自己的 OS（tlinux）团队维护的内核，这个内核历史比较久，不支持 Docker，我们就选择了 CentOS 6.5 的内核。实际上，由于 CentOS 的内核不像 Ubuntu，演进得很慢，CentOS 6.5 的内核也很老，但基本能把 Docker 跑起来。但在实际使用过程中遇到了一些内核方面问题，现在 tlinux 团队已经提供了 3.10.x 内核的支持，我们也在逐渐往 3.10.x 的内核迁移。

第二个问题就是容器集群的管理调度，那时候虽然出现一些专门针对 Docker 的容器管理工具，比如 Fig、Shipyard 等，但这些工具无法胜任生产环境大规模集群管理调度。刚好那时 Google 开源了 Kubernetes，它是 Borg 的开源版本实现。源于对 Google 的崇敬，我们研究了一下源码，基于 0.4 版本，针对我们的环境做了一些定制修改，用于我们的集群管理调度。现在我们最大的单个 Kubernetes 集群 700 多台物理机、将近 3000 个容器，生产一个容器只需几秒钟。

容器的监控问题也花了我们很多精力。监控、告警是运营系统最核心的功能之一，腾讯内部有一套很成熟的监控告警平台，而且开发运维同学已经习惯这套平台，如果我们针对 Docker 容器再开发一个监控告警平台，会花费很多精力，而且没有太大的意义。所以，我们尽量去兼容公司现有的监控告警平台。每个容器内部会运行一个代理，从 /proc 下面获取 CPU、内存、IO 的信息，然后上报公司的监控告警平台。但是，默认情况下，容器内部的 proc 显示的是 Host 信息，我们需要用 Host 上 cgroup 中的统计信息来覆盖容器内部的部分 proc 信息。我们基于开源的 lxcfs，做了一些改造实现了这个需求。



这些解决方案都是基于开源系统来实现的，当然，我们也会把我们自己觉得有意义的修改回馈给社区，我们给 Docker、Kubernetes 和 lxcfs 等开源项目贡献了一些 patch。融入社区，与社区共同发展，这是一件很有意义的事情。

InfoQ：在我的印象里，游戏还是相对较保守的行业。你们在内部推进 Docker 过

程中遇到过哪些阻力？是如何解决的？

尹烨：首先，我们会在 Docker 新功能接入与业务原有习惯之间做好平衡，尽量降低业务从原来的物理机或虚拟机切换 Docker 的门槛，现阶段业务接入我们的 Docker 平台几乎是“零门槛”。正如前面所述，我们的 Docker 平台上已经跑了数十款端游、页游和手游。

其次，Docker 相对原有的开发部署方式变化很大，与其它新事物一样，让大家全部适应这种方式是需要一些时间，但 Docker 本身的特性是能够在游戏运营的各环节中带来诸多便利，我们的业务主观上对新技术的应用还是欢迎的，双方共同配合，共同挖掘 Docker 在游戏运营的中的优势，所以 Docker 推广目前没有遇到太大的阻力。

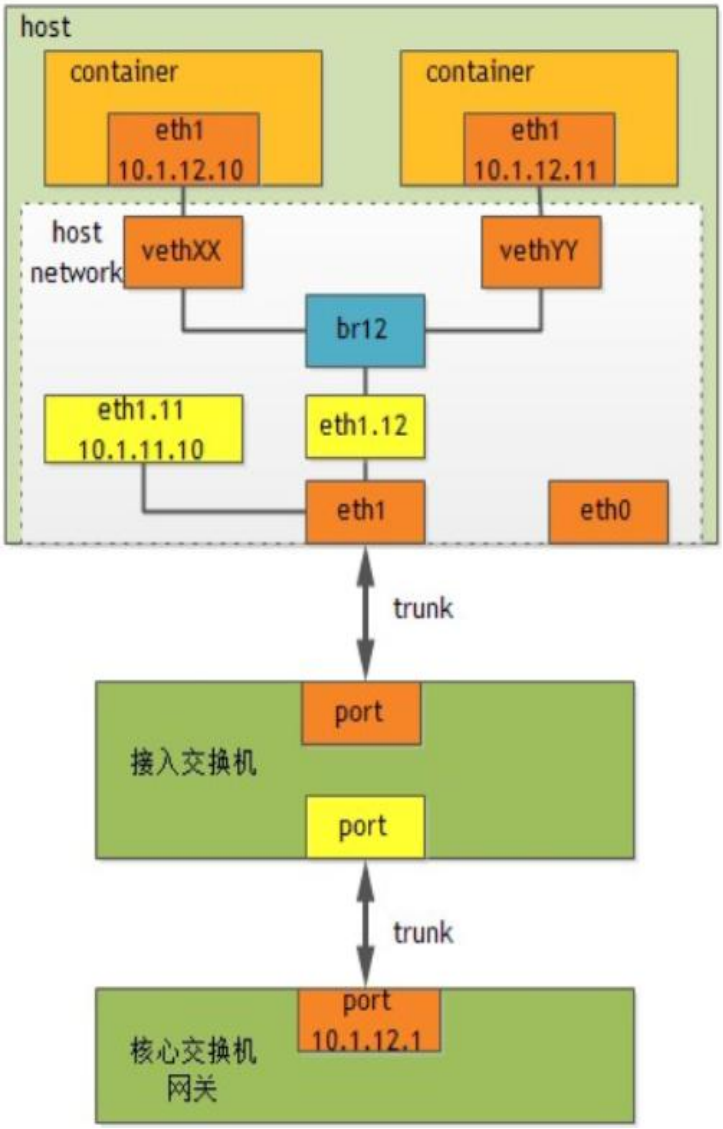
InfoQ：应用过程中，哪些问题是 Docker 目前无法解决的？你们的解决方案是怎样的？

尹烨：我们在实践过程中遇到了很多问题，有些是内核的问题，也有些是 Docker 本身的问题。由于篇幅问题，这里仅举一些比较大的问题。详细的分享留到 8 月底的容器技术大会吧。

我们遇到的第一个大的挑战就是网络的问题，Docker 默认使用 NAT 方式，这种方式性能很差，而且容器的 IP 对外不可见。一般来说，游戏业务对网络实时性和性能要求较高，NAT 这种方式性能损失太大，根本不能用于实际业务中。另外，腾讯内部的很多程序对 IP 都是很敏感的，比如只有特定的 IP 才能拉取用户的资料，如果这些服务没有独立的 IP，是无法正常运行的。

我们针对腾讯的大二层网络环境做了较大的调整，整体架构如下图。

整体架构比较简单，与原来虚拟机的网络架构一致。每个容器都有一个独立可以路由的 IP，网络性能大幅提高，基本能满足业务的需求。而且，每个容器都可以携带 IP 在同一个核心交换机下任意漂移，业务通过 IP 漂移可以做很多有意义的事情，比如 Host 故障快速恢复等。另外，我们针对一些对网络性能要求高的应用，直接使用 SR-IOV，可以完全达到物理机的网络性能。



容器相对于虚拟机，有很多优势，但是也有一些劣势，比如安全性、隔离性等。由于我们是内部业务，所以安全性的问题不是那么突出，但隔离性的问题还是给我们带来了很多麻烦。性能监控我们通过 lxcfs 基本解决，但是还是有一些问题无法解决，比如内核参数的问题。很多内核参数没有实现 namespace 隔离，CentOS 6.5 的内核下，在容器内部修改，会影响整个 Host，我们只能在 Host 上设置一个最优的值，然后告诉业务，让他们不要在容器内部修改内核参数。3.10.x 的内核要好一些，对于没有实现 namespace 的参数，在容器内部不可见，这可以防止业务私自修改内核参数，避免对别的业务造成

影响。但是，有些业务对内核参数有特殊要求，我们只能让业务选择虚拟机。

再举个例子，一些业务会将程序进行 CPU 绑定，这可以避免 CPU 切换带来的性能损失，由于程序无法获取 cgroup 对容器 cpuset 的限制，绑定会失败，这需要业务程序先获取容器的 cpuset，但还是给业务带来了不便。

再比如，现在 cgroup 对 buffer IO 并不能进行 throttle 限制，不过内核社区已经在解决了，但离生产环境使用，可能还需要些时间。

还有 Docker daemon 进程升级的问题，现有 Docker 实现下，Docker daemon 一退出，所有容器都会停止，这会给大大限制 Docker 本身的升级。但最近社区已经在讨论这个问题，希望这个问题在不久的将来得到解决。

InfoQ：使用过程中有哪些坑？你们有做哪些重点改进？

尹烨：上面已经讨论了很多我们在使用 Docker 遇到的问题，当然还有更多，这里不再一一论述。这里再举个简单的例子吧。Docker daemon 进程在退出时，会给所有的容器的 init 进程发送 SIGTERM 信号，但是如果容器的 init 进程忽略了 SIGTERM 信号，就会导致 daemon 进程一直等待，不会结束。我们修改了 Docker，发送 SIGTERM 信号后，等待一段时间，如果容器还是没有退出，就继续发送 SIGKILL 信号，让容器强制退出。我们将这个修改提交到了 Docker 社区，因为一些原因并没有被接受，不过已经有另外的 PR 解决了。

InfoQ：这么长时间的应用，有做过复盘吗？未来有什么计划？

尹烨：的确，相对于 Docker、Kubernetes 的发展速度，大半年的时间已经很长了。我们使用的 Docker 版本还是基于 1.3.2 的，Kubernetes 的版本是基本 0.4 的，已经很老了，但是基本上是满足我们现在的要求，而且系统运行也很稳定，所以，短时间内不会做大的调整。但是，我们也看到最近 Docker 发布一些非常有意思的变化，比如 network plugin、volume plugin，还实现了默认的 overlay network。Plugin 机制会让 Docker 更加开放，生态圈也会发展得更快。但总体来说，这些新特性还处于 experimental 阶段，等这些特性成熟稳定之后，我们会考虑切换我们的 Docker 版本。

但是，我们也不会坐着等待，也会尝试一些我们需要的东西。比如，最近我们正在实现将 Docker 与 Ceph 结合，我们已经实现了 Ceph rbd graph driver，将 Docker 的 rootfs 跑在了 Ceph 存储集群上面，结合 IP 漂移，可以实现更快的故障恢复。我们也将其实现提交给了 Docker 社区，因为一些技术原因，并没有被社区接受。目前看来，以 plugin graph driver 的方式提供会更好，但是现在 Docker 还不支持 plugin graph driver，不过 Docker 社区正在实现，相信不久就会支持。我们已经在 GitHub 上创建一个开源的 plugin graph driver 项目。

另外，对于 Kubernetes 的应用，我们也还没有完全发挥其优势。Kubernetes 的负责人 Brendan Burns 曾说过，“Make writing BigTable a CS 101 Exercise”，Kubernetes 有很多非常超前的设计思想，当然，也会改变业务原有的一些软件架构，真正应用到实际业务中还需要一些时间。

另外，我们也会继续探索与业务开发、运维的结合方式，进一步发挥 Docker 的优势，提高我们的运营效率，更好的支撑游戏业务的发展。

受访嘉宾介绍

尹烨，腾讯互娱运营部高级工程师。2011 年毕业加入腾讯，现在主要负责 Docker 等相关技术在腾讯游戏业务的实践。主要关注 Linux 内核、虚拟化、存储等领域，给 libcontainer/kubernetes 等开源项目贡献过代码。

8

Docker 在蚂蚁金融云平台中的探索与实践

作者 **郭蕾**

蚂蚁金融云是蚂蚁金服推出的针对金融行业的云计算服务，旨在将蚂蚁金服的大型分布式交易系统中间件技术以 PaaS 的方式提供给相应客户。在整个的 PaaS 产品中，蚂蚁金服通过基于 Docker 的 CaaS 层来为上层提供计算存储网络资源，以提高资源的利用率与交付速度，并用来隔离底层 IaaS 的不同，IaaS、CaaS 与 PaaS 三层相互借力，互相配合。为了进一步了解蚂蚁金融云的整个体系架构，InfoQ 采访了蚂蚁金服基础技术部系统组 leader 吴峥涛。另外，吴峥涛也是 QCon 上海《容器与云计算》专题的讲师，他在大会上将分享题为《蚂蚁金服金融云 PaaS Docker 实践》的演讲。

InfoQ：能否介绍下蚂蚁金服的金融云 PaaS 平台？

吴峥涛：金融云 PaaS 是从 2014 年中开始研发的，目前已经承载了网商银行以及另外两个核心业务，后续会以公有云和专有云两种模式对外提供。之所以会有金融云 PaaS 这个项目，是因为蚂蚁这些年来在大型分布式系统领域涉及的 SOA、消息通讯、水平扩展、分库切片、数据一致、监控、安全等技术方向积累了大量的中间件以及与之完整配套的监控运维研发流程体系，这一切在性能和稳定性以及扩展性上做的都不错，能够有效的支撑蚂蚁的业务发展，并应对『双 11』这样的高负荷挑战。很多金融客户与伙伴都对此非常感兴趣，所以我们希望能够把这一整套的技术上云并产品化，以 PaaS 的方式整体对外输出，帮助金融行业的客户使用云计算技术去 IOE，帮助他们解决我们已经解决的技术问题，让他们能专注于业务逻辑。上帝的归上帝，凯撒的归凯撒。

InfoQ：蚂蚁金服想把自己的中间件技术以 PaaS 产品的形式对外输出，这中间碰到了哪些问题？为什么会想到通过 Docker 这样的技术来解决？

吴峥涛：遇到的问题有很多，最主要的问题包括：

1. 金融云 PaaS 是承载在阿里云 IaaS 上的，最初的方案是，用户部署应用的时候，根据所属技术栈，由系统解析软件包（例如 sofa4 依赖 CloudEngine、Tengine、cronolog、JDK 等等），下载安装配置并启动。这样的资源交付、应用部署周期比较长，客户体验不太好。
2. 蚂蚁的应用架构采用的是基于服务发现、消息总线的 SOA 方案，模块间通过服务接口调用；服务可以是本地接口也可以是远程接口，对于调用者是解耦合的。在实际部署的时候，我们会

   根据业务纬度切分大量的服务模块出来，以金融云 PaaS 中枢为例，目前已经有几十种服务模块。这样的话，我们希望资源粒度越小越好，原有以 VM 为粒度的资源分配方式已经不能满足我们的需求。同时资源交付速度慢导致扩容缩容慢，影响资源利用率。
3. 金融云 PaaS 如果对外输出，可能使用非阿里云的 IaaS，所以需要一个中间层屏蔽多种 IAAS 的区别。
4. 镜像管理比较麻烦，无法自动化，也不是白盒。

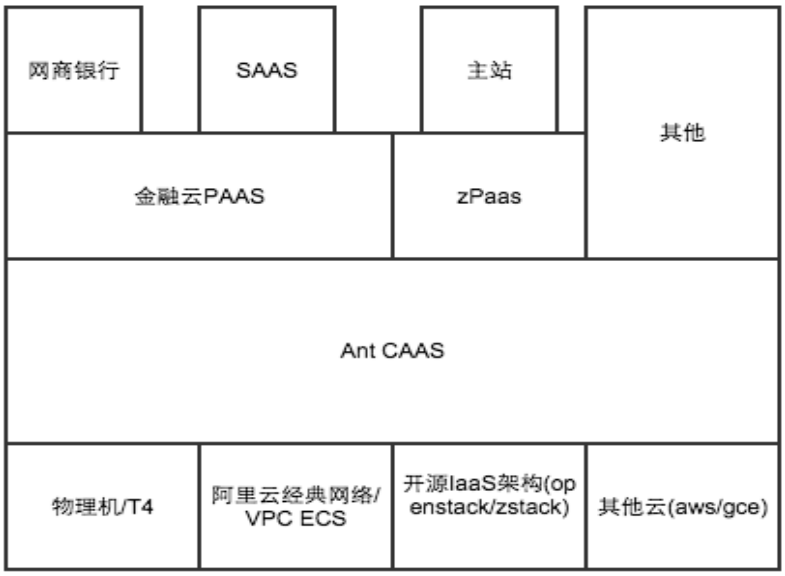
而遇到的这些问题，都可以通过 Docker 来解决。

InfoQ：能否介绍下你们整个平台的系统架构？Docker 在整个架构中扮演怎么样的位置？

吴峥涛：我们在 PaaS 和 IaaS 之间插了一个 AntCaaS 层，这是一个基于 Docker 的管控平台，他的职责是：

1. 以微容器（Docker）为载体，为用户（PaaS、SaaS）按需提供计算存储网络资源，提高资源的利用率与交付速度。
2. 对 PaaS、SaaS 屏蔽 IaaS 实现细节。

3. 实现容器、集群级别的标准化与可复制、可迁移。



AntCaaS 提供 container、app、cluster 三层接口，可以把他当作一个轻量级的 IaaS，区别仅仅是提供 Container 而不是 VM，然后也提供 PaaS 层的接口。

一个比较有特色的功能是，通过『镜像中心+集群 template+环境相关参数列表』，我们实现了集群的快速复制，目的是为了应付突发的负载高峰。

InfoQ：在 PaaS 层面，你们是如何屏蔽底层不同 IaaS 的区别的？

吴峥涛：目前 AntCaaS 可以直接运行在物理机上也可以运行在经典网络的 ECS VM 集群中，VPC、ECS 支持还在开发过程中。在 AntCaaS 中，我们采用三层架构，通过创建不同 pool 来兼容不同的 IaaS。pool 包含了：

1. 多个 node（container 的载体）。

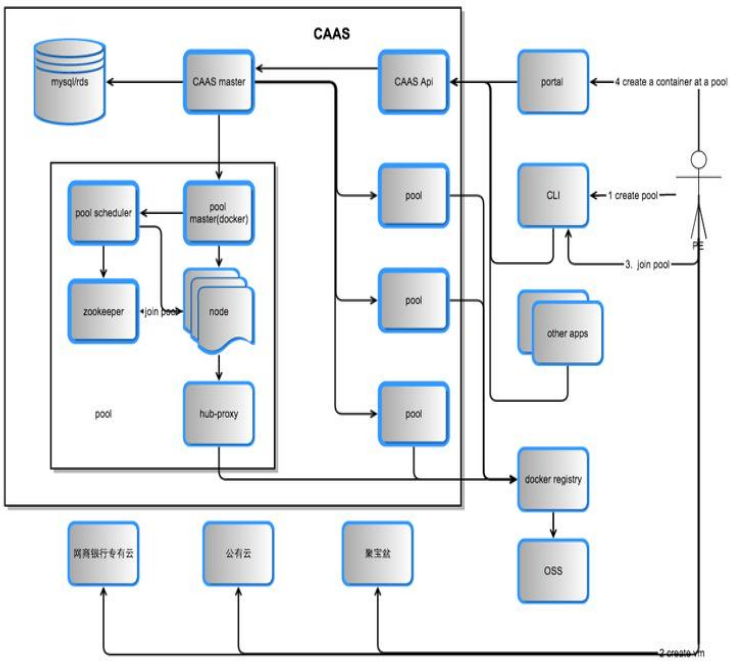
2. ZK 用以监控 node 和 container。

3. scheduler 调度器，负载 container 的创建调度。

4. manager，对 master 提供管控的 HTTP Rest 接口。

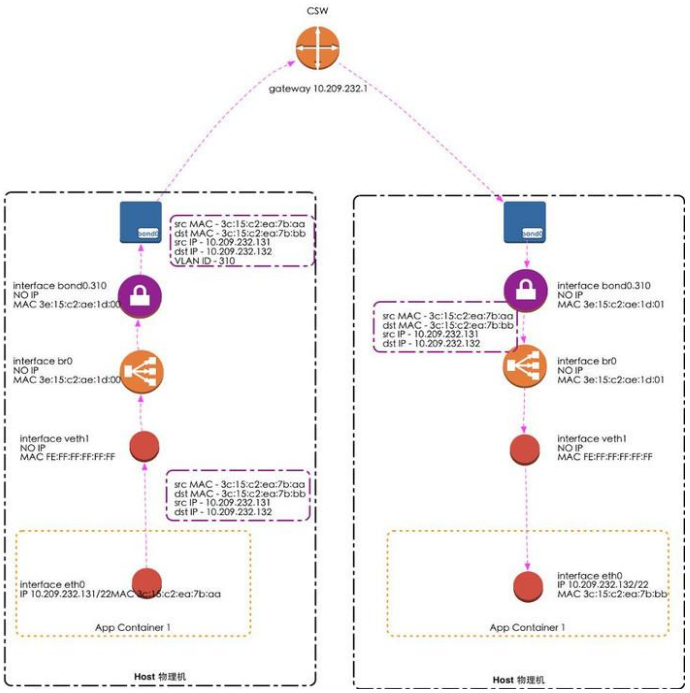
5. registry-proxy 保证网络联通性以及 cache 镜像数据，加速镜像下载。

6. 在 node 内起 cadvisor 监控 container。

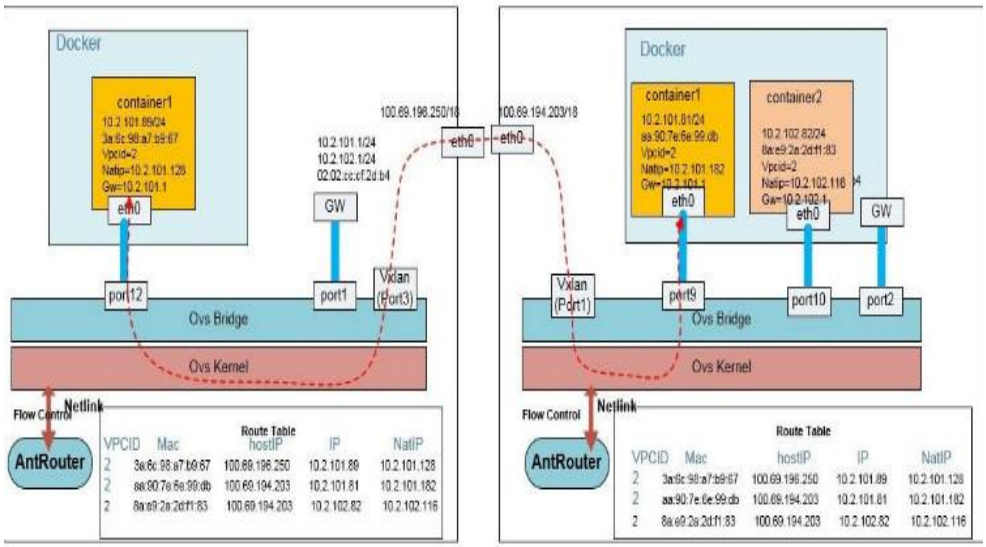


Docker 默认的 bridge/host 网络模式不能满足我们的需要，根据 IaaS 提供的网络功能不同，我们扩展了 vlan/vxlan 两种网络 driver，第三者 vpc driver 还在开发中。

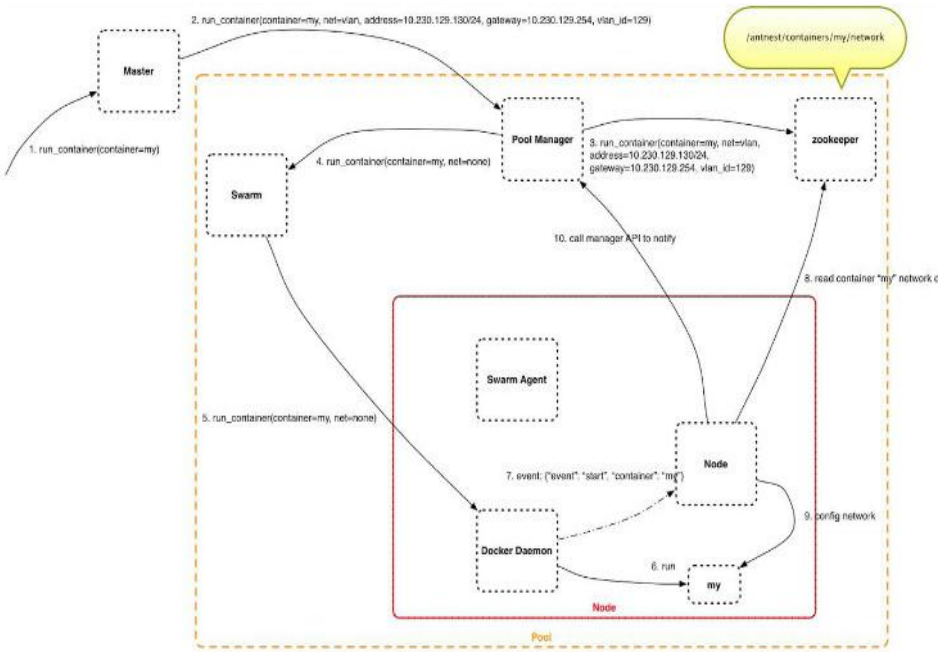
vlan 模式，事先为 container 分配一个与 node 不一样的网段。



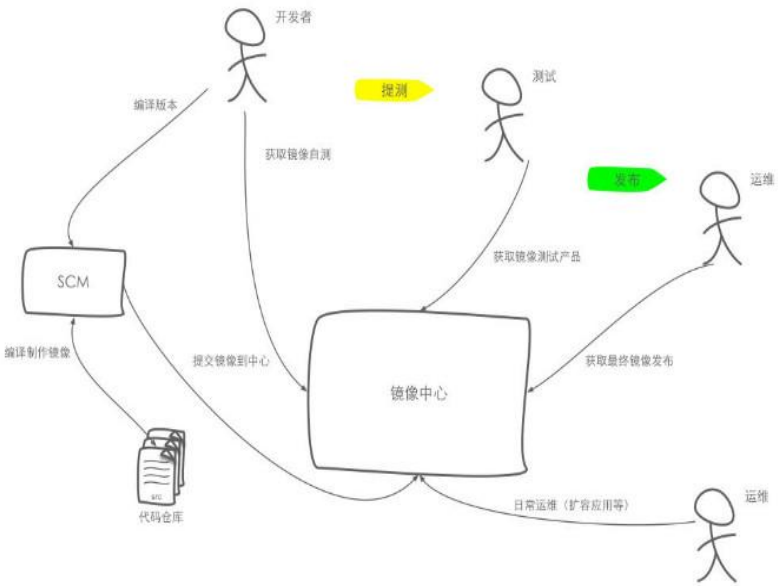
vxlan 模式，首先通过 ovs 实现跨 node 的私有网络，然后通过 zk 缓存同步 vpcid、vpc ip、node ip 的映射关系，极端情况下，如果一个 vpc 有 1000 个 container 分布在 1000 个 node 上，那么新建一个 container 加入这个 vpc 时，需要通知所有的 node。



另外在 VPC 内，我们提供轻量级的 DNS，用于内部域名解析；轻量级的 LB，用于内部的负载均衡。



下图是可扩展的容器创建流程。



InfoQ：容器的安全问题是怎么解决的？

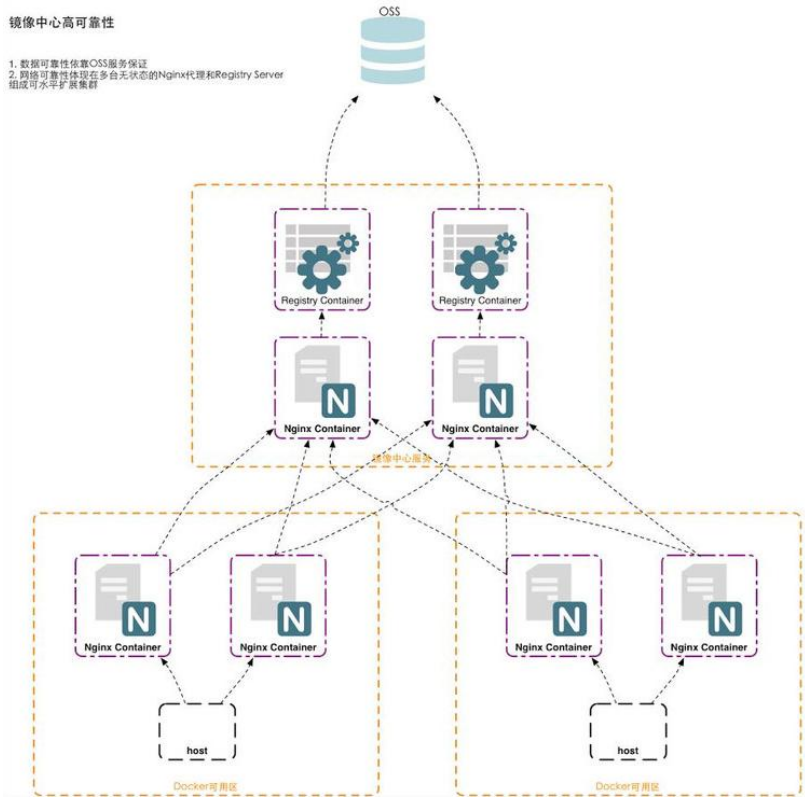
吴峥涛：基于 pool-node-container 的三层解决方案。用户和 pool 是 1 对多的关系，所以 pool 的 node 必然都属于同一个用户，同一个 node 上的 container 属于同一个用户。pool 和 pool 之间根据 IaaS 不同采用不同的隔离方案，经典网络的 ECS 使用安全组隔离，vpc ecs 使用 vpc 隔离。

InfoQ：社区中有反馈说 Docker 会经常无故挂掉，你们有遇到过吗？有做过深入跟进吗？

吴峥涛：碰到过，我们的架构设计允许 Docker Daemon 和 Container 挂掉。使用了集团的一个 Docker Patch，在 Docker Daemon 挂掉后，不影响 container 运行。出现比较多的场景是 docker pull 时候。

InfoQ：你们是如何将基于 Docker 的系统与原有系统对接的？

吴峥涛：为了与现有的运维流程管控 SCM 系统对接，帮助现有系统迁移，以及帮助开发同学转换开发模式，我们做了不少妥协方案。作为 Docker fans，理想的开发模式是下面这样。



但是实际上蚂蚁目前已经有 1000 多个应用，几千的开发人员，很难让他们一下都切换到 Docker 这种以镜像为中心的研发模式上；但是如果一个团队一个团队的推广，那么耗时不可控。并且蚂蚁原有的运维、监控、SCM 等系统都是以 VM 为纬度的，基于 Docker 的运维发布系统需要与原有系统对接集成难度比较大。

所以我们的第一个策略是，首先解决线上环境的 Docker 化部署问题，开发者的本地开发环境 Docker 化问题暂且不管，希望通过线上环境 Docker 化来吸引开发人员学习使用 Docker。

接下来的问题是，谁来写 Dockerfile，首先各个研发团队的人不关心是否使用 Docker 部署，所以不可能写 Dockerfile，由 Docker 团队或者运维同学负责，没有应用代码的编辑权限，同时工作量太大并且不了解应用容易出问题。所幸蚂蚁的业务应用大多数都是基于 sofa3 或 sofa4 框架的 Java 应用，所以我们做了 sofa3/4 的基础镜像以及提供一个辅助工具，使用 sofa3/4 镜像启动一个 container，然后使用原有的发布工具将应用的

tgz 包（类似 war）发布到 container 中。这样就不需要写 Dockerfile，同时原有运维系统也能把 container 当作 VM，无缝对接。

InfoQ：聊聊你们异地多机房的统一镜像中心解决方案？

吴峥涛：关于跨机房镜像中心的解决方案要点：

1. 将镜像数据存在 OSS 写三份，保证数据安全性；

2. Registry 本地不保存数据，是无状态的服务，可以水平扩展；

3. Registry 上跑一个 Nginx，提高镜像数据访问速度；

4. 在每个 pool 中部署一套 Nginx，开启文件缓存，对常见镜像进行预热，构建缓存。

9

Mesos 在去哪儿网的实践之路

作者 **徐磊**

背景

业务线开发环境的困扰

年初的时候机票的同事向我们反馈，希望可以提供 Docker 环境帮助他们快速构建开发环境，加速功能的迭代。正好我们 OpsDev 团队也在为容器寻找试点，双方一拍即合，立即开始了前期的调研工作。

随着交流的深入，我们发现对于一个包含了几十个模块，快速迭代的系统，开发团队想要建立一个相对稳定的，能覆盖周边模块的开发和自测环境是非常困难的，除了要申请虚拟机外，还要新增 profile，创建 jenkins job，发布，服务依赖等一系列的流程。

即使解决了以上问题，运维这套环境又是个大麻烦：项目之间的依赖关系写在配置文件中，切换环境时需要手工修改；多套不同版本的环境维护起来费时费力；对于涉及面较广的联调，需要其他组的同事配合完成，更不用说这些模块间的版本如何有效的保证一致了。

整理问题

经过多次的讨论和调研，最终双方团队确认出几个业务线最关心的功能，优先解决：

1. 版本一致，即代码版本，配置版本和数据库 schema 一致，减少联调时不必要的适配和调整。
2. 快速切换多套环境。
3. 服务依赖，开发新人也可以轻松部署整套复杂的环境。
4. 维护简单，例如新增项目时，自动加入到整套环境中。
5. 低学习成本，节约时间去开发业务。
6. 环境隔离，最好每个人一套完整环境，不互相影响。

暂时性解决 0 和 1 的问题

业务线的同事用 docker-compose 临时搭建了一套开发环境，但是需要手工维护版本以及 nginx 的转发，同时也暴露出了更多的问题：

1. 能支撑如此多模块的 compose，只能是实体机，资源限制较大。
2. 扩容模块时的端口冲突问题。
3. 数据库持续集成
4. 容器固定 IP

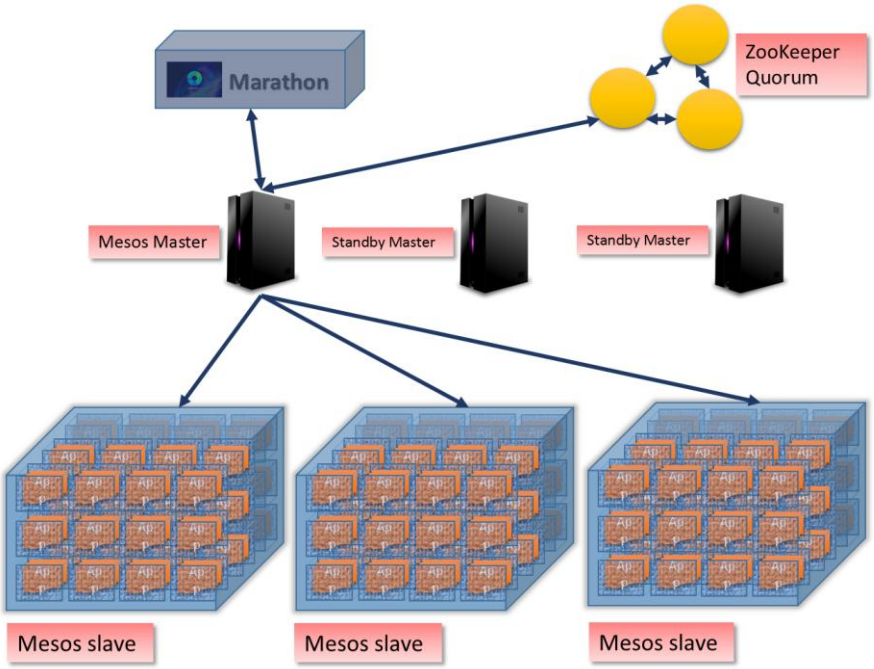
需要找到一个治标又治本的方案解决业务线的问题。

寻求解决之道

参考了现有的容器集群方案后，最终焦点集中在了 Apache Mesos（后简称 Mesos）和 Google Kubernetes 上。Kubernetes 的 pod 和 service 概念更贴近业务线的诉求，同时，Mesos 在资源管理和调度灵活性上显然经得起生产的考验。最终团队决定两者并行测试，在各自的优势方向寻找试点项目做验证。

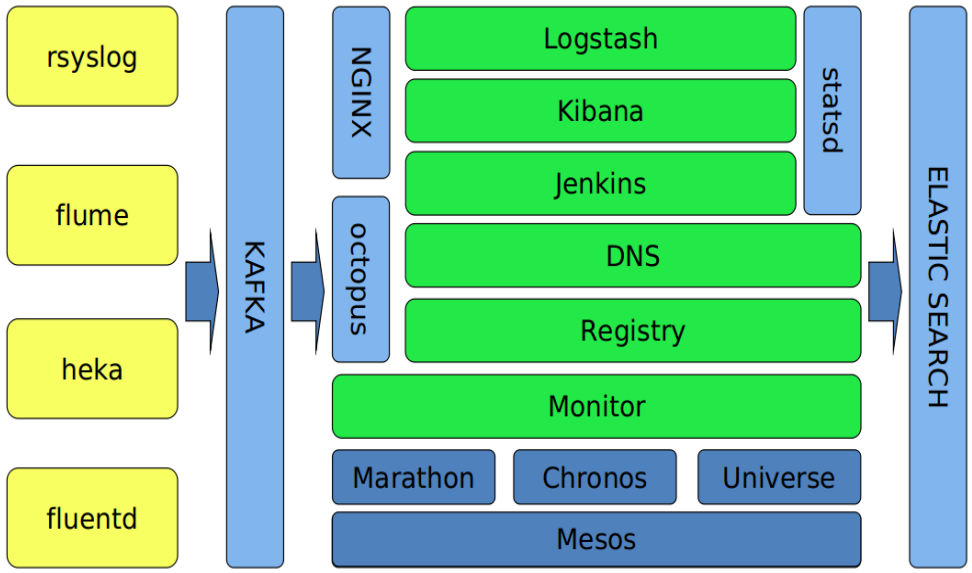
项目试点

仔细考量后，我们选择了基于 ELK 构建的日志平台作为验证 Mesos + Docker 的切入点，积累相关的开发和运维经验。



图一 典型的 Mesos + Docker 结构（source from google)

首先容器化的是 Logstash 和 Kibana，Kibana 本身作为 ElasticSearch 的数据聚合展示层，自身就是无状态化的，Logstash 对 SIGTERM 有专门的处理，docker stop 的时候可以从容处理完队列中的消息再退出。而 ElasticSearch 部署在 Mesos 集群外，主要考虑到数据持久化的问题以及资源消耗。采用 Marathon 和 Chronos 调度 Logstash 和 Kibana，以及相关的监控、统计和日志容器。



图二 日志平台的结构

数据来自多种方式，针对不同的日志类型，采取不同的发送策略。系统日志，比如 mail.log、sudo.log、dmesg 等通过 rsyslog 发送。业务日志采用 flume，容器日志则使用 heka 和 fluentd。汇总到各个机房的 Kafka 集群后，粗略的解析后汇总到中央 Kafka，再通过 Logstash 集群解析后存入 ElasticSearch。同时，监控数据通过 statsd 发送到内部的监控平台，便于后续的通知和报警。

随着业务线日志的逐步接入，这个平台已经增长成为单日处理 60 亿条日志/6TB 数据的庞大平台。

问题和经验总结

1. Daemon OOM

最初我们使用的 Docker 版本是 1.6，docker attach 接口存在内存泄露，容器的 stdout 输出较多日志时，比较容易造成 daemon 的 OOM。

```
{code}
fatal error: runtime: out of memory


runtime stack:
runtime.SysMap(0xc2c9760000, 0x7f310000, 0x7f453c96b000, 0x13624f8)
       /usr/local/go/src/runtime/mem_linux.c:149 +0x98
runtime.MHeap_SysAlloc(0x1367be0, 0x7f310000, 0x43b8f2)
       /usr/local/go/src/runtime/malloc.c:284 +0x124
runtime.MHeap_Alloc(0x1367be0, 0x3f986, 0x10100000000, 0x0)
       /usr/local/go/src/runtime/mheap.c:240 +0x66
......
{code}
```

这个问题是比较严重的，daemon 挂掉后容器跟着都宕机了，虽说上层的 Marathon 会重新部署应用，但是频率较高的话容易造成集群不稳定。

首先想到的办法就是用 runsv 启动 daemon，保证进程宕掉后可以重新被拉起。其次，参考了 Kubernetes 的做法，在 daemon 启动后修改 oom_adj 的值为-15，防止 daemon 被最先 kill 掉。

最治标的办法还是升级 Docker 的版本，或者自己 patch 这个 bug（https://github.com/docker/docker/issues/9139）。

2. Heka 的 DockerEventInput 不释放 socket

DockerEventInput 使用的 go-dockerclient 有 bug，heka 异常推出后不会关闭 socket，容易导致文件句柄泄露，最终导致 daemon 不再接受任何命令，这个 BUG 在 v0.10.0b1 仍然还存在。

```
    {code}
    time="2015-09-30T15:25:00.254779538+08:00" level=error msg="attach: stdout: write unix @: broken pipe"
    time="2015-09-30T15:25:00.254883039+08:00" level=error msg="attach: stdout: write unix @: broken pipe"
    time="2015-09-30T15:25:00.256959458+08:00" level=error msg="attach: stdout: write unix @: broken pipe"
    {code}
```

相关问题：https://github.com/fsouza/go-dockerclient/issues/202。

3. 对新加入集群的 slave“预热”

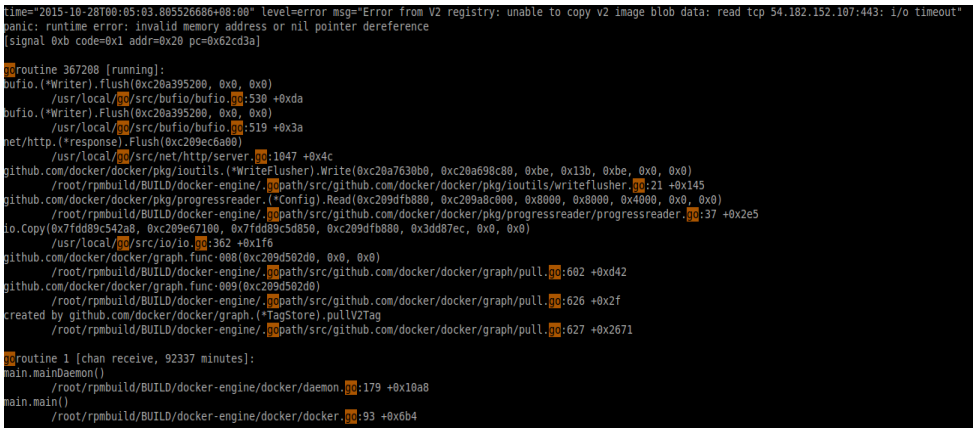
同在局域网内，第一次下载镜像也是比较慢的，推荐在 slave 部署完毕后，主动 pull 一批常用的镜像，减少第一次启动的时间。这个工作我们放在 salt、ansible 脚本里自动部署。另外，对于基础监控类的容器，Marathon 目前还未支持自动 scale，需要自己实现。

相关讨论：https://github.com/mesosphere/marathon/issues/846

4. Distribution 引起的 daemon 宕机

升级 1.7.1 后发现的问题，起因是一个手误导致 Marathon 的配置没有带上自己的 registry，daemon 去 pull 了官方的镜像。这个坑幸好发生在我们的 registry 准备迁移 V2 的之前，相关的代码还没有 patch 到我们自己的 docker 上，暂时还是使用 V1。

相关问题：https://github.com/docker/docker/issues/15724



5. Mesos 的资源抢占

资源抢占是在 Mesos 0.23.0 版本引入的，官方还不建议在生产环境使用，如何有效的抢占资源一直是我们在使用过程中比较关注的。

Mesos 的资源是直接映射到 role 上的，我们以此为切入点，提前划分多个 role，每个 role 分配静态资源。比如，ops 的 role 运行基础服务，每个 slave 上最多占用 4 个 CPU，logstash 则在每台机器上可以占用 32 个 CPU，以这种方式变相超售 CPU 资源。

```
MESOS_resources="cpus(logstash):32;"

MESOS_resources="${MESOS_resources}cpus(common):4;"

MESOS_resources="${MESOS_resources}cpus(kibana):4;"

MESOS_resources="${MESOS_resources}cpus(ops):4;"

MESOS_resources="${MESOS_resources}cpus(spark):16;"

MESOS_resources="${MESOS_resources}cpus(storm):16;"

MESOS_resources="${MESOS_resources}cpus(rebuild):32;"

MESOS_resources="${MESOS_resources}cpus(mysos):16;"

MESOS_resources="${MESOS_resources}cpus(others):16;"

MESOS_resources="${MESOS_resources}cpus(universe):1;"
```

```
MESOS_resources="${MESOS_resources}cpus(test):8;"
MESOS_resources="${MESOS_resources}mem(*):126976;ports(*):[8000-32000]"
```

在使用时，不再根据容器的资源使用情况动态调整实例数量，而是交替发布任务抢占 CPU。比如凌晨 2 点至 6 点是业务低峰，日志量少，许多 logstash 容器并未满负荷工作，正适合发布 Spark 的 job。这种调度方式实现简单，基于时间调度，更容易监控。

缺点也是显而易见的，需要提前规划 role，尽量对每种资源消耗大户都分配到一个对应的 role，扩展性较差，适合上层应用较稳定的系统。等 MESOS-3791 合并后，就可以动态的管理 role，那么 Mesos 的资源的管理就会更加灵活了。

6. 版本升级

主要是 Mesos、Docker 的版本升级，由于众所周知的原因，Docker 的升级是比较痛苦的，需要停止所有的容器后再升级 daemon。我们的线上环境经历了 Mesos 0.22.0 到 0.25.0，Docker 1.4.1 到 1.7.1 的演进，总结出了一套比较有效的升级策略，上层服务无感知。首先 Mesos 要开启白名单（--whitelist）功能：

1. 先将要升级的机器踢出白名单，这一步保证了上层的 Framework 在收到 statusUpdate 不会调度到这台机器上；
2. 然后逐个 stop 容器，容器内的应用建议处理 SIGTERM 信号做清理工作；
3. 接着停止 docker daemon 和 mesos slave；
4. 升级 docker 和 mesos 版本；
5. 重启 docker 和 mesos 并将机器重新加入到白名单。

发环境快速 rebuild

有了日志平台的经验，我们的工作中心开始向实际需求倾斜，尽快满足业务线的环境要求。共经历了三次比较大的变更，主要从兼容性，公司内的发布流程和开发人员易

用性的角度考量，逐步演进：

1. OpenStack + nova-docker + VLAN
2. Mesos + Marathon + Docker(--net=host) + 随机端口
3. Mesos + Marathon + Docker + Calico

第一阶段：容器当作虚拟机用

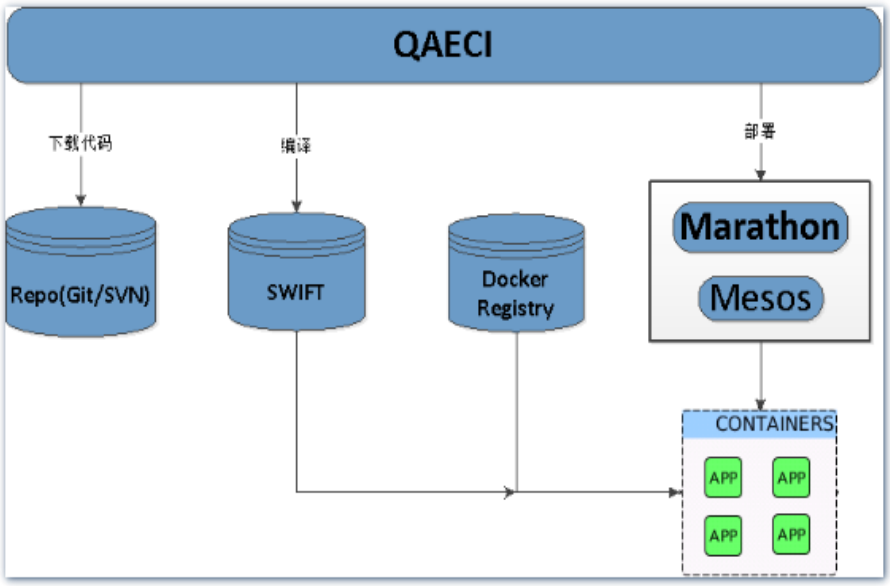
容器的使用和行为尽量模拟虚拟机是我们第一阶段考虑的重点，同时还要考虑到发布系统改造的成本，OpenStack 提供的 nova-docker 自然成了首选。再此基础上，为容器提供外部可访问的独立 IP（VLAN）。nova-docker 和 nova-network 已经提供了大部分功能，整合的速度也比较快。

容器启动后会有多个进程，比如 salt-minion 和 sshd，这样使用者可以 ssh 到容器内 debug，而部署的工作则交给 salt 统一管理。

第二阶段：以服务为核心

逐渐强化以服务为核心的应用发布和管理流程，向统一的服务树靠拢。在第一阶段的成果的基础上，完善服务树的结构和规则，为后面打通监控树，应用树等模块做好充分的准备。

同时，容器开始从 OpenStack + nova-docker 的结构向 Mesos + Marathon + Docker 迁移，整套环境的发布压缩到了 7～9 分钟，其中还包含了 healthcheck 的时间，还有深入优化的空间。



1. 依赖放在 QAECI 中维护，发布时根据拓扑排序后的结果选择自动切换并行，串行发布。
2. 代码和配置在容器启动后再拉取，减少维护镜像的成本，方便升级运行环境，比如升级 JDK 或 Tomcat。
3. 服务端口全部随机生成，并通过环境变量注入到依赖的容器中并替换配置，这样就解决了--net=host 模式下端口分配的问题。dubbo 服务注册的是宿主机的 IP 和 PORT，如果是 bridge 模式的话，记得要注册宿主机的 IP 和映射的 PORT。
4. 适当缓存编译后的代码，减少重复构建的时间浪费。
5. Openresty + lua 脚本动态 proxy_pass 到集群内的 Tomcat，外部即可通过泛域名的方式访问 Marathon 发布的应用，例如 app1.marathon.corp.qunar.com 即可访问到 app1 对应的 WEB 服务。
6. 修改 logback 和 tomcat 的配置，所有日志都输出到 stdout 和 stderr，并附带文件名前缀做区分。并通过 heka，配合 fields_from_env 区分是哪一个 Mesos task 的日志，统一发向日志平台汇总和监控。

第三阶段

为容器分配固定 IP，打通集群内外的服务通信，让开发人员无障碍的访问容器。为此我们引入了 Calico 作为解决方案。Calico 整合 Mesos 比较简单，通过 Mesos slave 启动时指定--modules 和-isolation 即可使用：

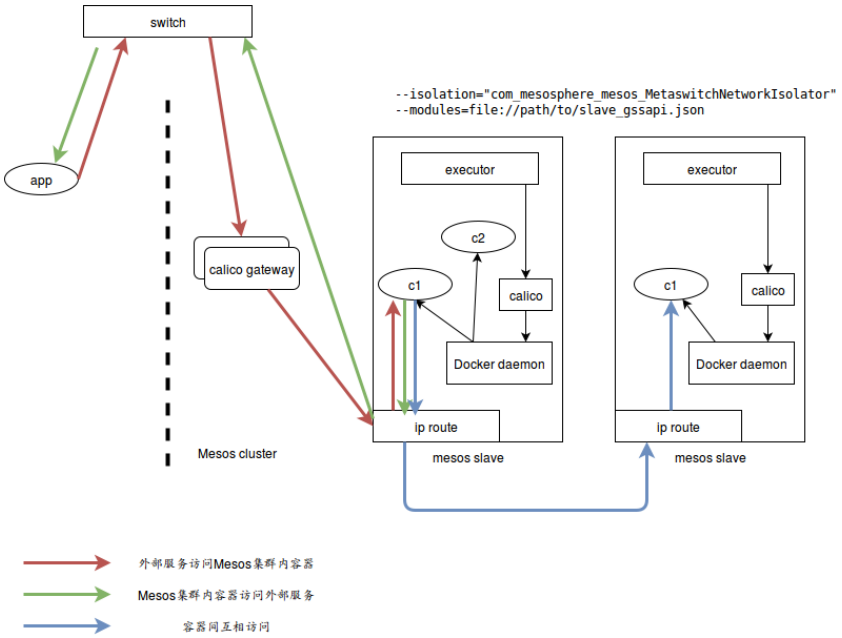
```
    {code}
    ./bin/mesos-slave.sh --master=master_ip:port --namespaces='network' \
--modules=file://path/to/slave_gssapi.json \
--isolation="com_mesosphere_mesos_MetaswitchNetworkIsolator" \
        --executor_environment_variables={“DOCKER_HOST”: “localhost:2377”}


    {   "libraries": [     {      "file":
"/path/to/libmetaswitch_network_isolator.so      "modules":
[        {          "name": "com_mesosphere_mesos_MetaswitchNetworkIsolator",
"parameters": [            {              "key": "initialization_command",
"value": "python /path/to/initialization_script.py arg1 arg2"            },
{              "key": "cleanup_command",              "value": "python
/path/to/cleanup_script.py arg1
arg2"            }          ]        }      ]     }   ] }
    {code}
```

这样 Mesos 在执行 Docker 命令的时候，所有的请求都被 calico 容器劫持并转发给 docker daemon，同时给容器分配 IP，上层的 Marathon 只需要额外添加两个 env 配置：

- CALICO_IP=auto|ip
- CALICO_PROFILE=test

结合我们自身的网络结构，我们在交换机上预留了一个 IP 段，全部指向了 calico 的两台 gateway，转发到 Mesos 集群内部：



同时整合公司内的 DNSDB 服务，将容器的名称和 IP 自动注册到 DNSDB 内，这样全公司的人都可以访问到这个容器，打通集群内外的通信。对于一些有特殊要求的情况，如开发机的名称必须符合一定命名规则，通过传入--hostname 就可以模拟一台开发机。

总结

经过近 1 年来的使用和运维，在 Docker 和 Mesos 上踩了不少的坑，多亏了社区的贡献者们，积累了许多经验。Mesos 表现出的稳定性、可用性和扩展性足够担当生产环境的资源管理者，美中不足的是调度策略略显单一，依赖上层 Framework 的二次调度。

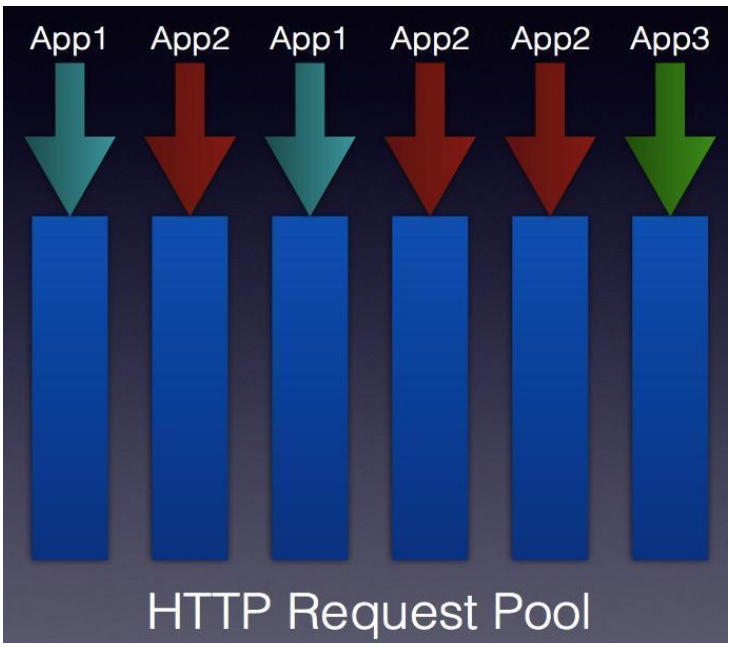
后续我们将考虑在第四阶段调研 Swarm on Mesos，利用 Docker 公司原生的集群方案配合 Mesos 的资源管理，为业务线提供更加稳定，便利的容器环境。

10

基于 Kubernetes 打造 SAE 容器云

作者 **从磊**

作为国内第一个公有云计算平台，SAE 从 2009 年已经走过了 6 个年头，积累了近百万开发者，而一直以来 SAE 一直以自有技术提供超轻量级的租户隔离，这种隔离技术实际是从用户态和内核态 hook 核心函数来实现 HTTP 层的进程内用户隔离：



如图所示，这种方式的好处是：

- 精准的实现租户间 CPU、IO、Memory 隔离

- 可以实现进程多租户复用，从而达到超低成本
- 完全对等部署，管理方便

另外，这种模式的最大好处可以实现完全的无缝扩容，SAE 自动根据 HTTP 等待队列长度进行跨集群调度，用户从 1000PV 到 10 亿 PV 业务暴增，可以不做任何变更，早在 2013 年，SAE 就利用这种机制成功的帮助抢票软件完成 12306 无法完成的任务，12306 抢票插件拖垮美国代码托管站 GitHub。

但是这种模式也有很大的弊端，最主要的弊端就是：

- namespace 独立性不足
- 本地读写支持度不好
- 容易产生用户 lock-in

针对于此，SAE 决定基于 Kubernetes 技术推出以 Docker 容器为运行环境的容器云，下面我们就以下三个方面展开讨论：

1. Kubernetes 的好处是什么
2. Kubernetes 的不足有哪些
3. 针对不足，怎么改进它

一、Kubernetes 的好处

选择一个技术，对于大型业务平台来讲，最重要的就是它的易维护性，Kubernetes 由 Go 语言编写，各个逻辑模块功能比较清晰，可以很快定位到功能点进行修改，另外，Kubernetes 可以非常方便的部署在 CentOS 上，这点对我们来讲也非常重要。

Kubernetes 提出一个 Pod 的概念，Pod 可以说是逻辑上的容器组，它包含运行在同一个节点上的多个容器，这些容器一般是业务相关的，他们可以共享存储和快速网络通信。这种在容器层上的逻辑分组非常适合实际的业务管理，这样用户可以按照业务模块组成不同的 Pod，比如，以一个电商业务为例：可以把 PC 端网站作为一个 Pod，移动端 API 作为另一个 Pod，H5 端网站再作为一个 Pod，这样每个业务都可以根据访问量

使用适当数量的 Pod，并且可以根据自己的需求进行扩容和容灾。

相同的 Pod 可以由 Replication Controller 来控制，这样的设计方便 Pod 的扩容、缩容，特别当有 Pod 处于不健康的状态时，可以快速切换至新的 Pod，保证总 Pod 数不变，而不影响服务。这种模式保证了实际业务的稳定性。

二、Kubernetes 的不足

对于目前的 Kubernetes 来讲，无缝扩容是一个比较致命的问题，不过好在社区有大量的人力已经投入力量在开发了，截止到我写稿的时间，基于 CPU 的扩容代码已经出来，也就是用户可以设定一个平均 CPU 利用率，一旦数值高于某个阈值，就触发扩容。但当我们仔细思考这个模式时，就会发现不合理的地方：CPU 利用率并不能真正反映业务的繁忙程度，也不能反映是否需要扩容。比如某些业务需要大量跟 API 或者后端数据库交互，这种情况下，即使 CPU 利用率不高，但此时用户的请求已经变得很慢，从而需要扩容。

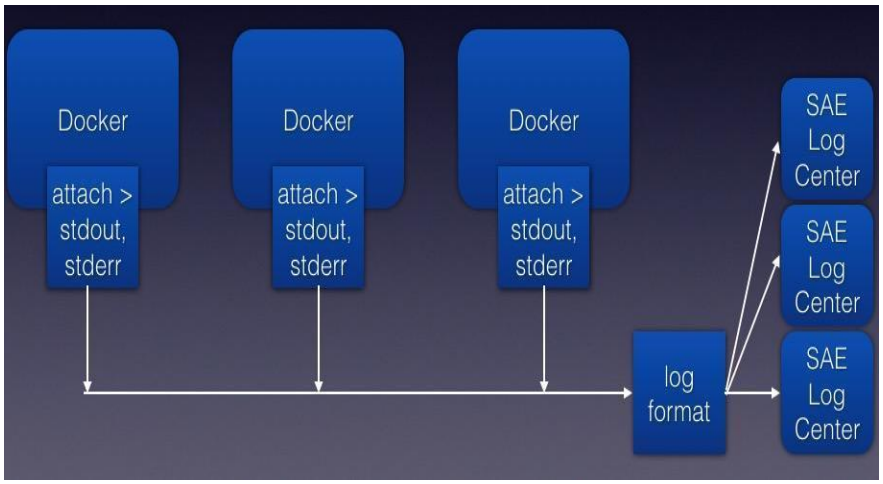
再有，良好的监控一直是一个系统稳定运行的前提，但 Kubernetes 显然目前做的不够，先不说 Kubernetes 自身的监控，就是对于 Pod 的监控，默认也只是简单的 TCP Connect，显然这对于业务层是远远不够的，举个最简单的例子，假如某个业务因为短时间大并发访问导致 504，而此时 TCP 协议层的链接是可以建立的，但显然业务需要扩容。

还有一些其他的功能，不能算是不足，但在某些方面并不适用于 SAE，比如 Kube-Proxy，主要是通过 VIP 做三层 NAT 打通 Kubernetes 内部所有网络通信，以此来实现容器漂 IP，这个理念很先进，但有一些问题，首先是 NAT 性能问题，其次是所有网络走 VIP NAT，但是 Kubernetes 是需要和外部环境通信的，SAE 容器云需要和 SAE 原有 PaaS 服务打通网络，举个例子，用户利用容器云起了一组 Redis 的 Pod，然后他在 SAE 上的应用需要访问这个 Pod，那么如果访问 Pod 只能通过 VIP 的话，将会和原

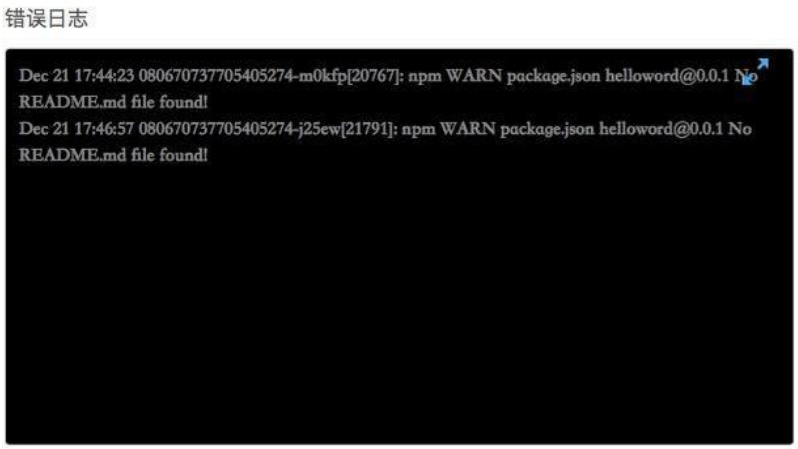
有 SAE 的网络访问产生冲突，这块需要额外的技巧才能解决。所以我们觉得 Kube-Proxy 不太适合。

三、如何改进 Kubernetes

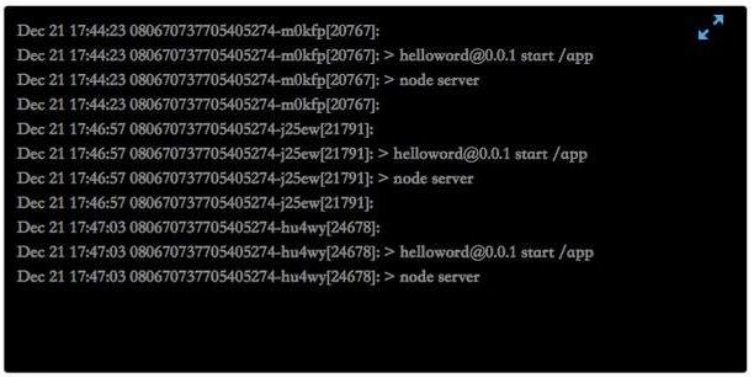
针对 Kubernetes 的不足，我们需要进行改进，首先需要改造的就是日志系统，我们希望容器产生的日志可以直接推送进 SAE 原有日志系统。



如上图所示，我们将容器的 stdout、stderr 通过 log format 模块重定向分发到 SAE 的日志中心，由日志中心汇总后进行分析、统计、计费，并最终展现在用户面板，用户可以清晰的看到自己业务的访问日志、debug 日志、容器启动日志以及容器的操作日志。



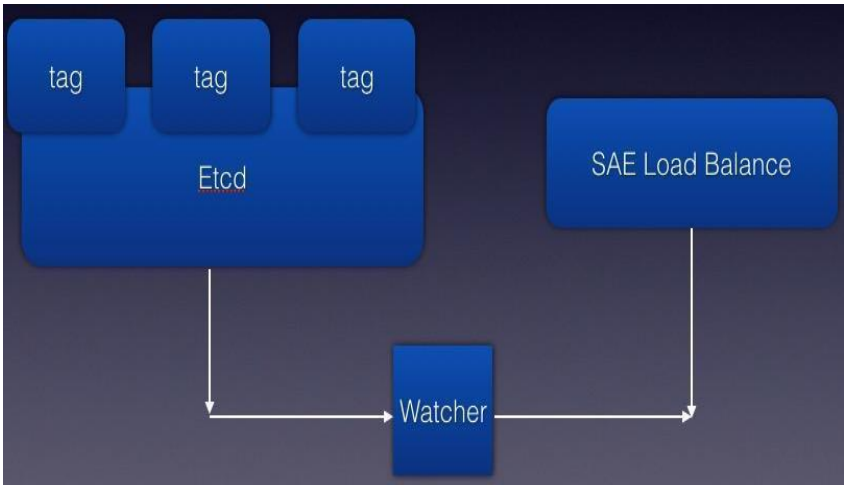
容器日志



访问日志

运维日志

其次，既然我们不使用 Kube-Proxy 的 VIP 模式，那么我们需要将 Pod 对接到 SAE 的 L7 负载均衡服务下面，以实现动态扩容和容灾。



etcd watcher 监控 etcd 里 pod 路径下的文件变更事件，得到事件取配置文件处理，判断配置文件里状态更新，将容器对应关系的变化同步到 SAE Load Balance，Load Balance 会在共享内存中 cache 这些关系，并且根据实际的用户请求的 HTTP Header 将请求 forward 到相应的 Pod 的对外端口上。当用户手工增加 Pod 时，Watcher 会在 100 毫秒内将新增的 Pod 的映射关系同步到 Load Balance，这样请求就会分配到新增的 Pod 上了。

当然，对于云计算平台来讲，最重要的还是网络和存储，但很不幸，Kubernetes 在这两方面都表现的不够好。

网络

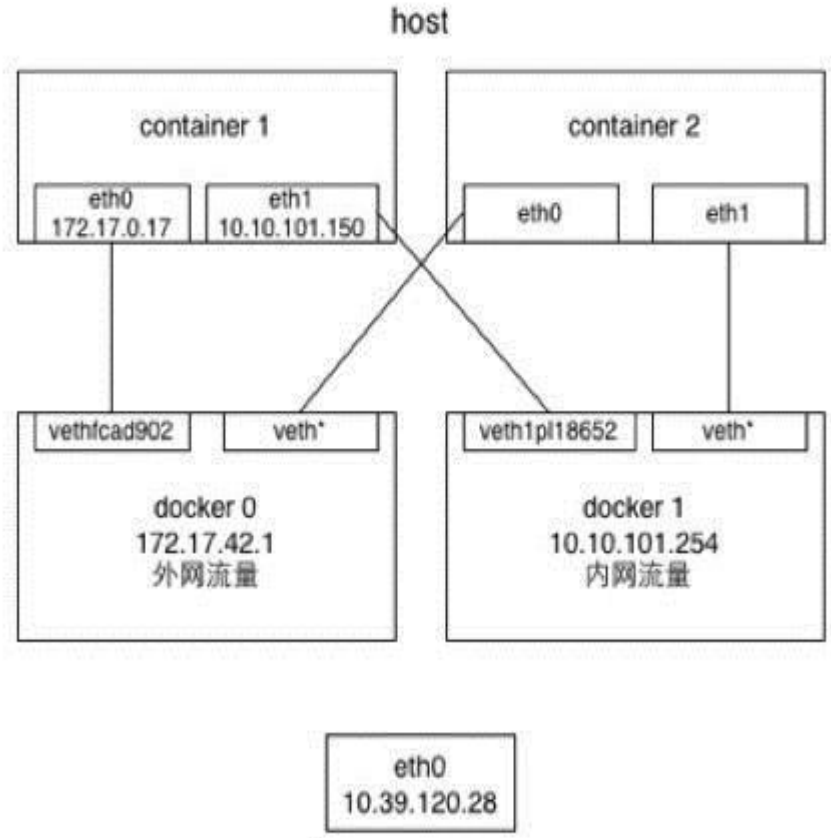
首先，我们来看对于网络这块 PaaS 和 IaaS 的需求，无论是 IaaS 还是 PaaS，租户间隔离都是最基本的需求，两个租户间的网络不能互通，对于 PaaS 来讲，一般做到 L3 基本就可以了，因为用户无法生成 L2 的代码，这个可以利用 CAP_NET_RAW 来控制；而对于 IaaS 来讲，就要做到 L2 隔离，否则用户可以看到别人的 mac 地址，然后很容易就可以构造二层数据帧来攻击别人。对于 PaaS 来讲，还需要做 L4 和 L7 的隔离处理，比如 PaaS 的网络入口和出口一般都是共享的，比如对于出口而言，IaaS 服务商遇到问题，可以简单粗暴的将用户出口 IP 引入黑洞即可，但对于 PaaS 而言，这样势必会影响

其他用户，所以 PaaS 需要针对不同的应用层协议做配额控制，比如不能让某些用户的抓取电商行为导致所有用户不能访问电商网站。

目前主流的 Docker 平台的网络方案主要由两种，Bridge 和 NAT，Bridge 实际将容器置于物理网络中，容器拿到的是实际的物理内网 IP，直接的通信和传统的 IDC 间通信没有什么区别。而 NAT 实际是将容器内的网络 IP:Port 映射为物理实际网络上的 IP:Port，优点是节约内网 IP，缺点是因为做 NAT 映射，速度比较慢。

基于 SAE 的特点，我们采用优化后的 NAT 方案。

根据我们的需求，第一步就说要将内外网流量分开，进行统计和控制。

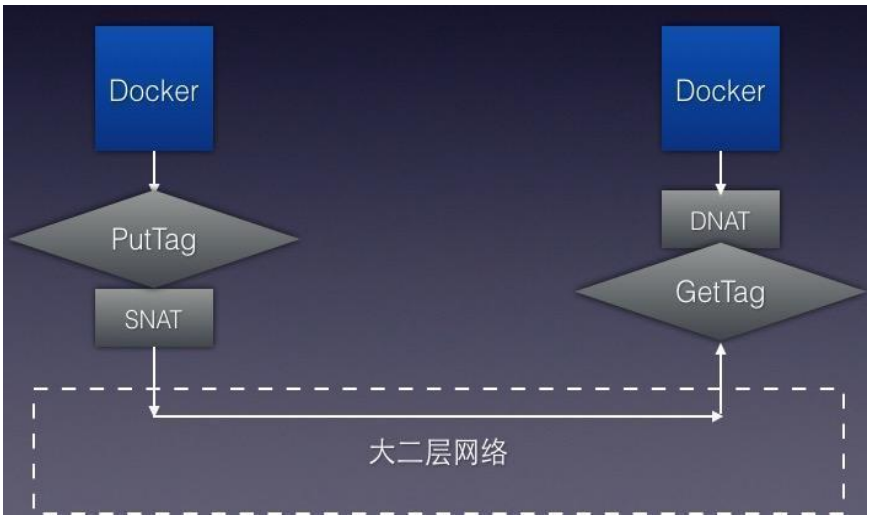


如图所示，在容器中通过 eth0 和 eth1 分布 pipe 宿主机的 docker0 外网和 docker1 内网 bridge，将容器的内外网流量分开，这样我们才能对内外网区分对待，内网流量免费，而外网流量需要计费统计，同时内外网流量都需要 QoS。

第二步就是要实现多租户网络隔离，我们借鉴 IaaS 在 vxlan&gre 的做法，通过对用户的数据包打 tag，从而标识用户，然后在网路传输中，只允许相同在同一租户之间网络包传输。

当网络包从容器出来后，先经过我们自己写的 TagQueue 进程，TagQueue 负责将网络包加上租户 ID，然后网络包会被 Docker 默认的 iptables 规则进行 SNAT，之后这个网络包就变成了一个可以在物理网络中传输的真实网络包，目标地址为目标宿主机 IP，

然后当到达目标时，宿主机网络协议栈会先将该网络包交给运行在宿主机上的 TagQueue 进程，TagQueue 负责把网络包解出租户 ID，然后进行判断，是否合法，如果不合法直接丢弃，否则继续进行 Docker 默认的 DNAT，之后进入容器目标地址。



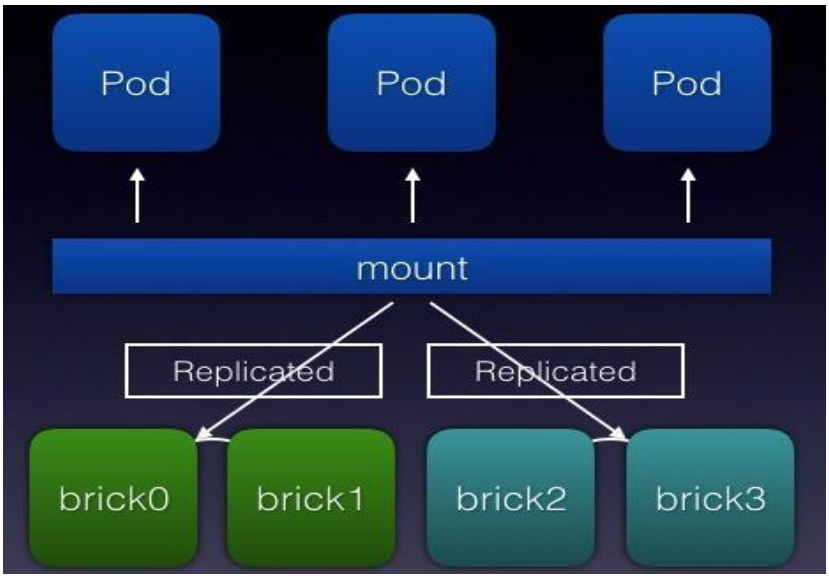
除去 Pod 间的多租户网络，对外网络部分，SAE 容器云直接对接 SAE 标准的流量控制系统、DDOS 防攻击系统、应用防火墙系统和流量加速系统，保证业务的对外流量正常

存储

对于业务来讲，对存储的敏感度甚至超过了网络，因为几乎所有业务都希望在容器之上有一套安全可靠高速的存储方案，对于用户的不同需求，容器云对接了 SAE 原有

PaaS 服务的 Memcache、MySQL、Storage、KVDB 服务，以满足缓存、关系型数据库、对象存储、键值存储的需求。

为了保证 Node.js 等应用在容器云上的完美运行，容器云还势必引入一个类似 EBS 的弹性共享存储，以保证用户在多容器间的文件共享。针对这种需求，Kubernetes 并没有提供解决方案，于是 SAE 基于 GlusterFS 改进了一套分布式共享文件存储 SharedStorage 来满足用户。



如图所示，以 4 个节点（brick）为例，两两一组组成 distributed-replicated volume 提供服务，用户可以根据需求创建不同大小的 SharedStorage，并选择挂载在用户指定的文件目录，mount 之后，就可以像本地文件系统一样使用。我们针对 GlusterFS 的改进主要针对三个方面：1，增加了统计，通过编写自己的 translator 模块加入了文件读写的实时统计；2，增加了针对整个集群的监控，能够实时查看各个 brick、volume 的状态；3，通过改写 syscall table 来 hook IO 操作，并执行容器端的 IO Quota，这样防止某个容器内的应用程序恶意执行 IO 操作而导致其他用户受影响。

通过 SharedStorage 服务，用户可以非常方便的就实现容器热迁移，当物理机宕机后，保留在 SharedStorage 数据不会丢失，Kubernetes 的 replication controller 可以快速在另外一个物理节点上将容器重新运行起来，而这个业务不受影响。

总结

Kubernetes 是一个非常优秀的 Docker 运行框架，相信随着社区的发展，Kubernetes 会越来越完善。当然，因为 SAE 之前有比较完备的技术储备，所以我们有能力针对 Kubernetes 目前的不足做大量的改进，同时，也欢迎大家来 SAE 体验运容器平台，http://www.sinacloud.com/sc2.html。