云生态 专刊

Cloud Ecosystem

第6期
Vol.06

打造中国最优质的云生态媒体

热点回顾

中国移动浙江公司数据中心操作系统（DCOS）实践

UPYUN云CDN架构演进之路

技术热点

从0到N，百度云的精益质量保障

JIMDB：一个大规模分布式内存存储的演进之路

对话大咖

与“品高云之父”聊私有云

观点&趋势

简评云计算过去的这一年

Lars Kurth谈开源安全流程之云安全

生态圈新闻

云生态专刊 06

本期主编 魏星

流程编辑 丁晓昀

发行人 霍泰稳

联系我们

提供反馈 feedback@cn.infoq.com

商务合作 sales@cn.infoq.com

内容合作 editors@cn.infoq.com

卷首语

信任是云计算的核心

2006 年，以 Amazon AWS 公共云为代表的云计算开始进入大众视野。经过 9 年多的发展，云计算在技术上已经进入成熟期。而在国内的市场上，公共云的服务提供商也是真真假假、大大小小、琳琅满目。如果你不熟悉云计算相关专业领域技术，挑花眼是很正常的事情。因此，你需要了解云计算面对的核心问题，了解由其延伸开来的一系列其他问题、标准和规范，并用它们作为选择你的云计算供应商的尺度。到那时，你也许会发现，云计算供应商的名气、规模，并不一定与其技术和服务的成熟度成正比；有时候，反而是不太知名的某个云更能体现真正的云计算特点，更有可能成为你的技术基石，支撑你的业务发展。

实际上，云计算需要解决的核心问题一直没有太大变化——信任问题。客户会问：我们凭什么相信云计算？我们为什么要将我们的数据放在云计算供应商那里？我的数据安全吗？

在云计算刚刚兴起时，对安全问题的回答，常常以银行做比喻——你相信你的银行吗？既然你愿意相信你选择的银行，把真金白银放在与你不相干的人那里保管，觉得那么做安全，为什么就不能相信你的数据放在云计算供应商那里是安全的呢？

银行是一个很好的比喻。但，保证安全只是营造信任的一部分。云计算技术发展到现在，想要完美回答信任问题，需要从安全、经验、分布、效率、产品完备性、合规性、丰富的 API 等多个方面做出考虑。我们接下来借助银行的比喻，一个个分析。

安全：不少人将安全等同于信任，认为只要提供足够的安全就可以获得客户的信任。想象一下，如果你的银行告诉你，我们有世界上最安全的金库，你只要把资金放在这里，不管是虚拟的线上黑客，还是剽悍的现实悍匪，都不能得手！只是，你要是想存钱呢，得等上半年，想取钱，也要提前三个月预约，然后每次还要填写数以百计的表格。这样的银行，你愿意成为它的客户吗？

经验：走进一家银行网点，看到的都是 20 出头的年轻人，虽然他们都很有活力，但是聊

起业务来一问三不知，遇到一些小问题就手忙脚乱。在另一家银行的营业部里，工作人员个个稳重有礼，业务熟练，有问必答，出现异常情况也毫不惊慌，能够迅速审慎地处理。这两家银行，你更信任哪一个？云计算同样如是，运营公共云平台的经验，只能靠一跤一跤摔过来、一个坑一个坑踩过来。公共云在国内大规模推广只有 6、7 年，谁在这些年里积累了血汗泪的经验，谁有久经考验的团队，谁就更值得你信赖。

分布：有一家各方面都堪称完美的银行，但它只有一个营业网点。不管你是开户取钱存钱理财，只能去这唯一的一个营业网点……没有人会选择只有一个数据中心的云计算供应商，就像没有人会选择只有一家网点的银行。供应商的机房部署越多，客户就拥有越大的服务灵活性。何况现在是全球化的时代，每一家有雄心壮志的公司，发展海外市场都属于他们的战略方向，如果云计算供应商有丰富的全球云计算资源和经验，对于需要拓展海外业务的公司来说，可以大幅降低拓展的时间成本、资金成本和试错机会成本。

效率：在市面上所有的银行中，人们最不愿意去的，就是工行和中行，他们的办事效率之低，可以说是“臭名远扬”。等得久不说，真轮到你办业务了，工作人员总是一幅“爱存不存（ICBC）”的样子。如果这样的银行提供云服务，而且是同样的效率，你会选择他们吗？你受得了吗？

产品的完备性：如果你的日常资金存取操作在某家银行，但是他们不提供理财产品，你必须要到另一家才能办理；想买点儿外汇，还得再选另外一家。是不是够让你焦头烂额的？没错，对于云计算的用户来说，云计算产品和服务的完备性，就像银行的金融产品完备性同样重要。一家成熟的云计算公司，能够提供多种配套的组合产品和服务，足以满足客户“一站式”的需要，而不是让客户在多家供应商之间来回奔波，更不用说还得考虑不同服务之间的对接问题。

规章、制度、流程的合规性：金融行业，是一个国家的命脉产业，是国家和政府重点监管的对象。理想情况下，银行和金融企业的一举一动，都要接受相关流程、法律法规的制约。云计算产业同样应该如此，因为云计算供应商每天面对的，都是客户的命脉：数据。现在国际上虽然还没有通用的云计算标准，但是国内有唯一受官方认可的“可信云”标准体系。一家公司的云产品和服务，如果能够通过可信云的认证，至少表明它可以满足合规性的要求。

丰富的 API：国际云计算行业有个不成文的惯例——不提供 API 的云计算称不上公共云。API 就是不同云产品和服务之间的服务接口和协议。你完全可以把它看做银行不同部分之间互相协作时需要满足的特定条件。有了这些服务接口和协议，云平台以及其上的产品就可以互相协作，将低效率的人工介入和干预环节最小化。云计算的 IaaS、PaaS 和 SaaS 这三个自底向上的服务层次，也都是构建于底层的 API 之上，同时，自己也向外提供 API。因此，一家成熟的云计算供应商，应该将自己的云产品和服务的 API 都提供出来，方便自己的用户调用。Uber 和 Airbnb 能够有这么快的发展，云平台成熟的 API 功不可没。

以上这七个因素，足以回答云计算的核心问题：信任问题。也可以用来考察一家云计算供应商是否合格、乃至是否成熟。如果你打算实施云计算，采购云计算供应商的产品和服务，不妨从这七个因素入手试试。

——**于浩**，SpeedyCloud 创始人兼 CEO

中国移动浙江公司数据中心操作系统（DCOS）

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。

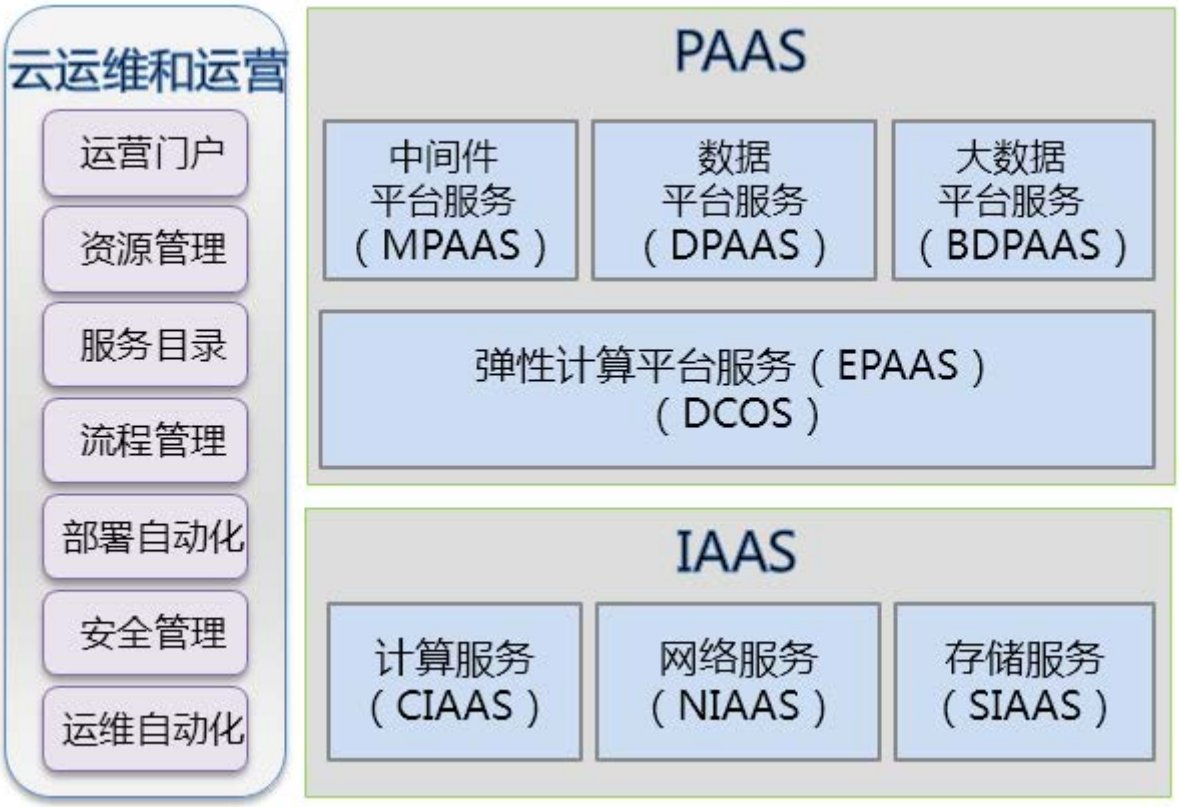


图 1：中国移动浙江公司私有云架构

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来调度。

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。根据对适合构建 DCOS

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）

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。架构如图 2 所示。

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。

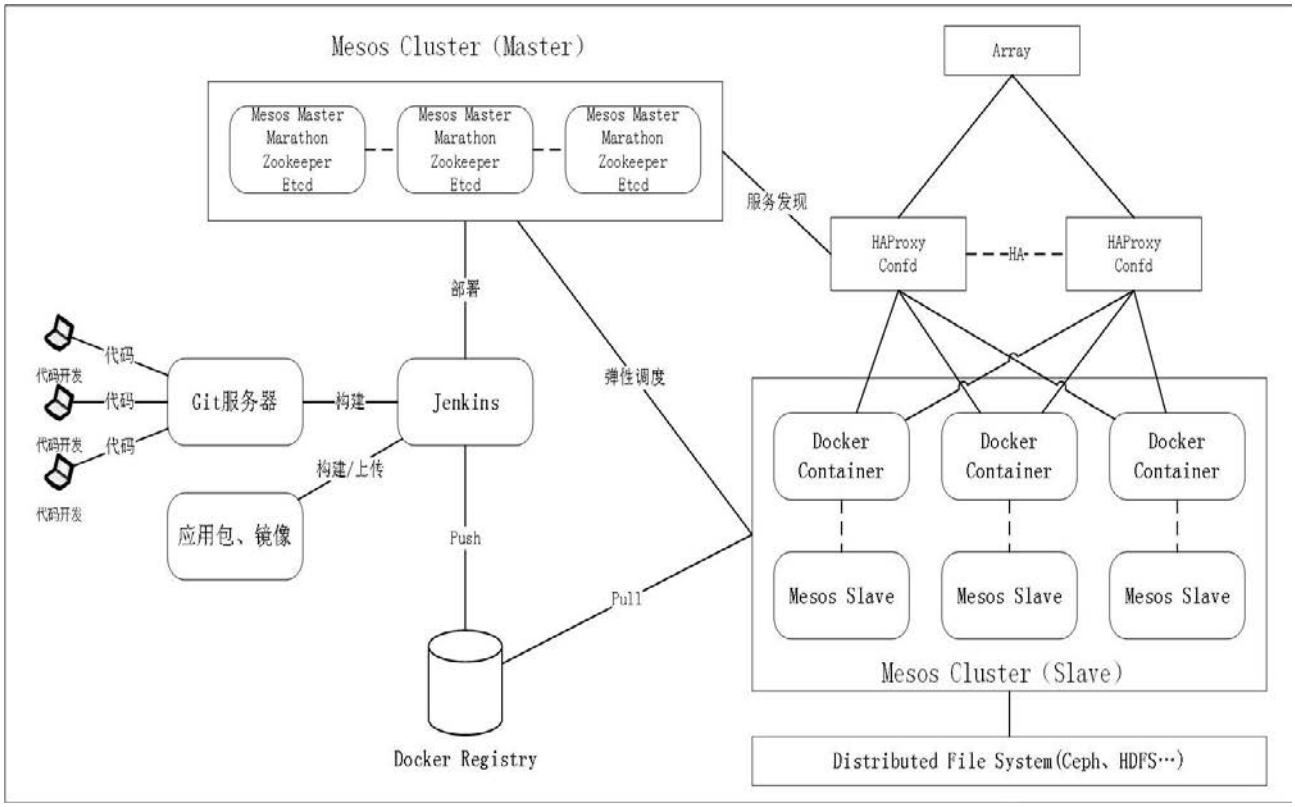

Mesos Cluster (Master)
Mesos Master
Marathon
Zookeeper
Etcd
Mesos Master
Marathon
Zookeeper
Etcd
Mesos Master
Marathon
Zookeeper
Etcd
Array
服务发现
HAProxy
Confd
HA
HAProxy
Confd
部署
代码
代码开发
代码
代码开发
代码
代码开发
Git服务器
构建
Jenkins
弹性调度
构建/上传
应用包、镜像
Push
Pull
Docker
Container
Docker
Container
Docker
Container
Mesos Slave
Mesos Slave
Mesos Slave
Mesos Cluster (Slave)
Docker Registry
Distributed File System(Ceph、HDFS…)

图 2

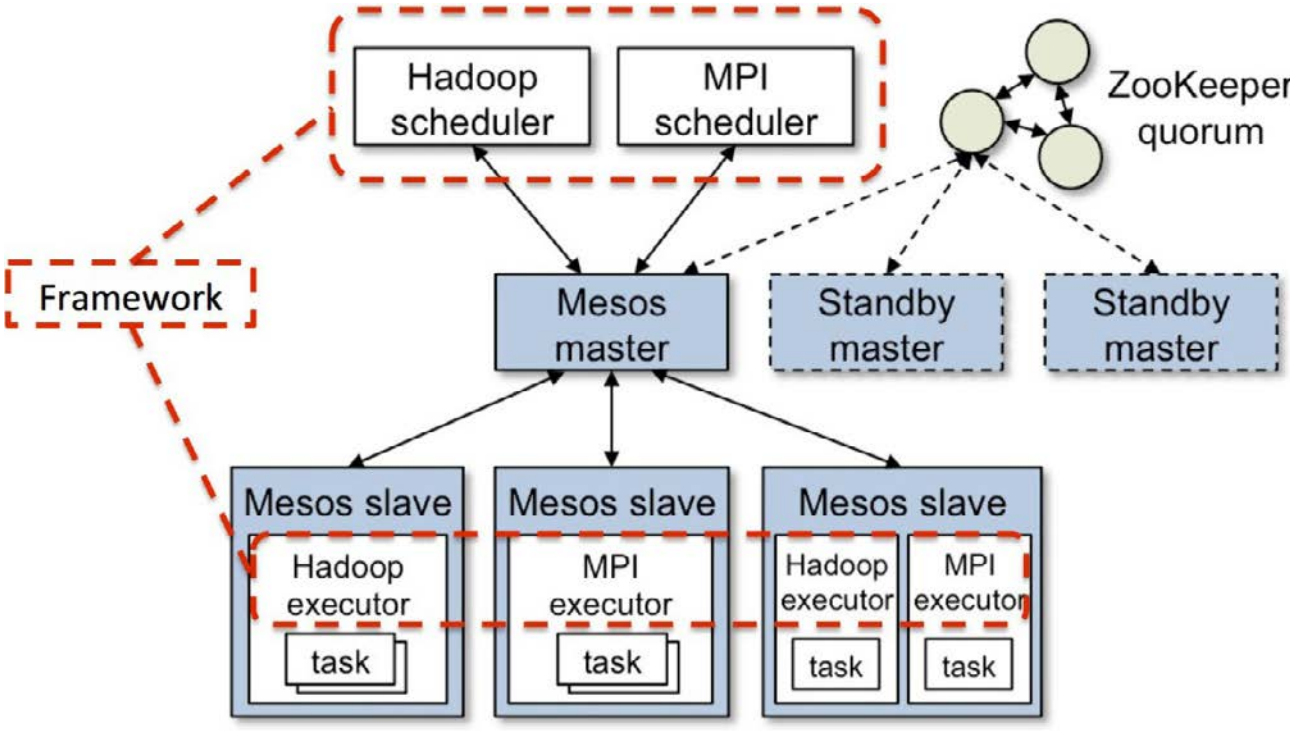

Hadoop
scheduler
MPI
scheduler
ZooKeeper
quorum
Framework
Mesos
master
Standby
master
Standby
master
Mesos slave
Hadoop
executor
task
Mesos slave
MPI
executor
task
Mesos slave
Hadoop
executor
task
MPI
executor
task

图 3

b) 资源调度

使用 Mesos 作为资源调度框架，其核心工作原理如图 3 所示。

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 ，其任务调度示意如图 4 所示。

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图 5 所示。

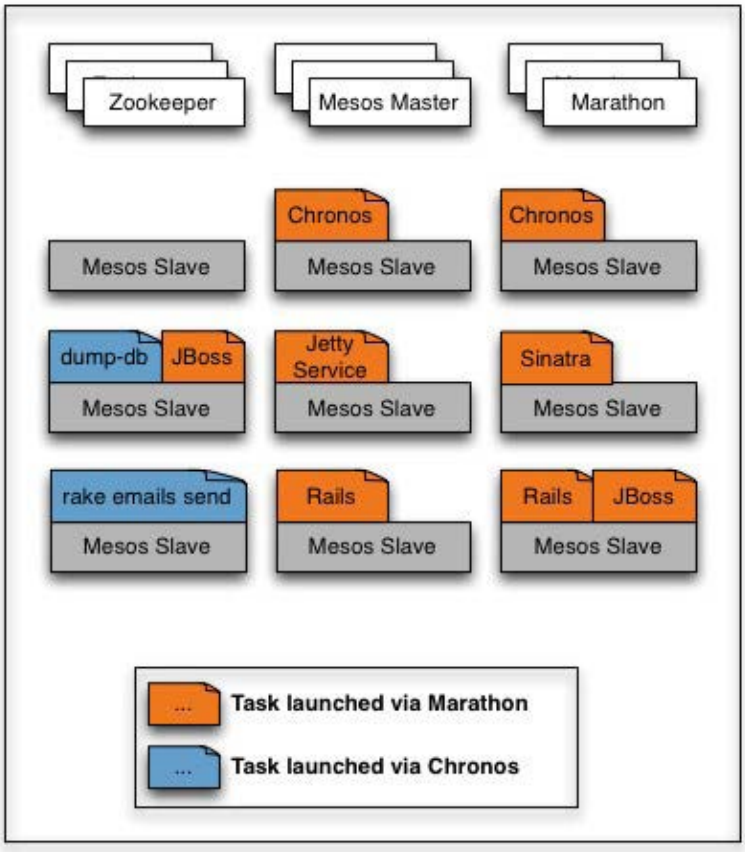


图 4

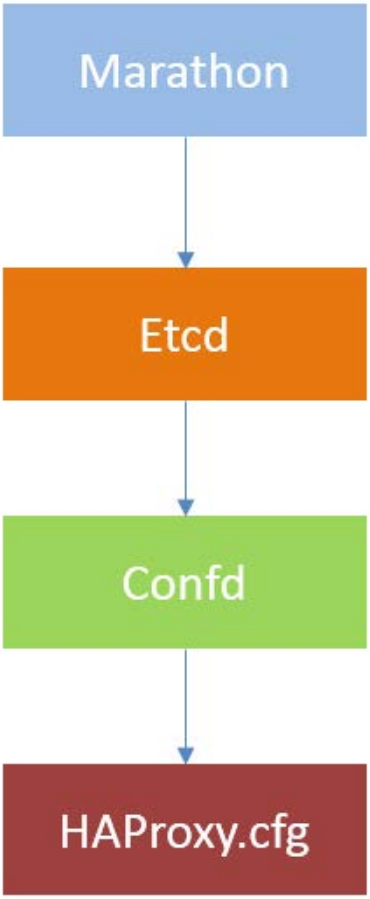


图 5

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 的功能框架如图 6 所示。

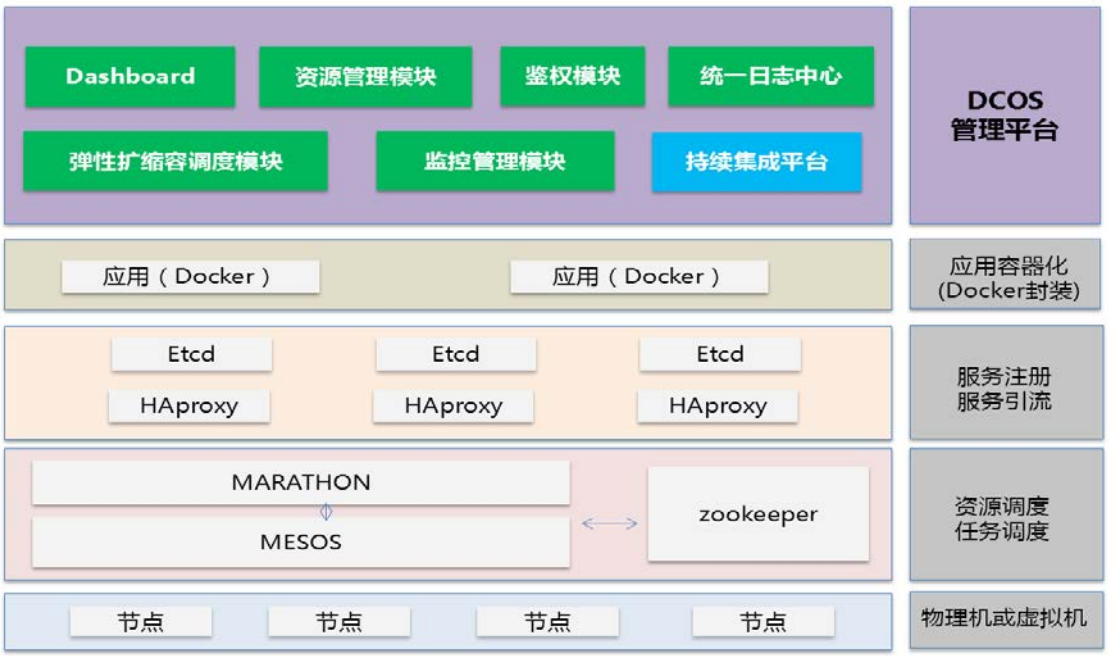


图 6

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。

图 7 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。

图 7

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，如图 8 所示。

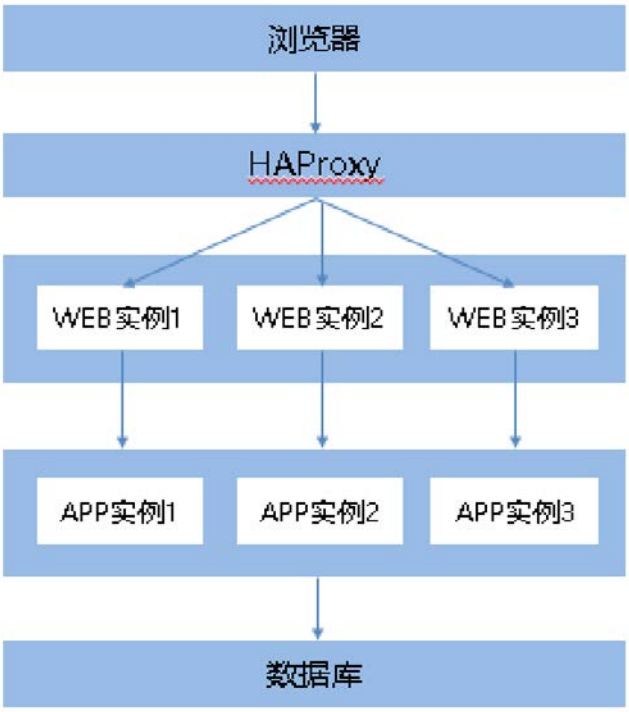


图 8

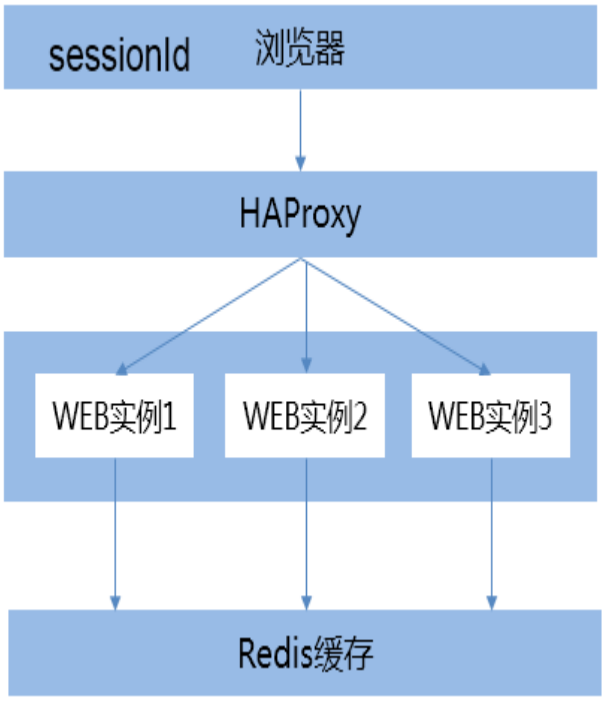


图 9

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。如图 9 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，改造后的架构图如图 10 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。

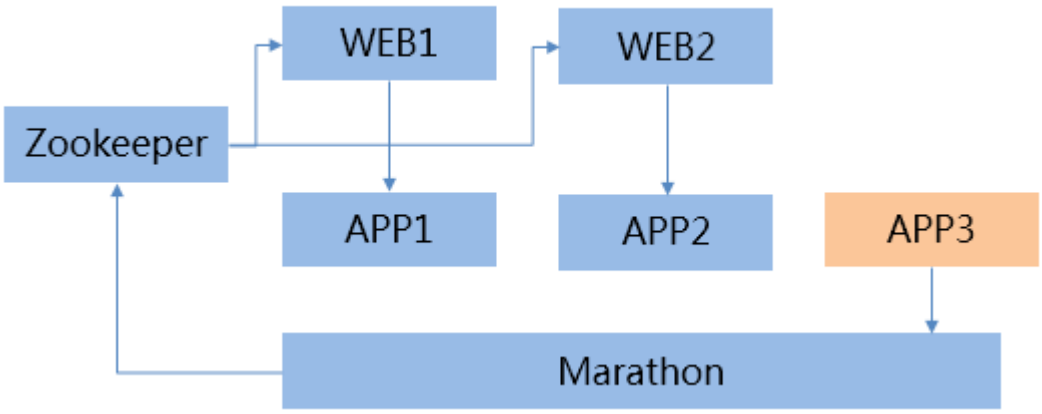


图 10

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。

从起步到爆发，UPYUN云CDN架构演进之路

作者 董志南

云 CDN，是一种依托强大云计算平台而构建的先进的流量分配网络，可有效解决 Internet 网络拥挤的状况，提高用户访问网站等应用服务的响应速度，使内容传输的更快速更稳定，同时提高服务可用性，并改善互联网上的服务质量。如今，CDN 已经成为目前构成互联网基础的环节之一。移动互联网也让它的角色变得更为重要。它所带来的加速效果，在今天以速度构成的用户体验中，可以说直接影响着企业的用户认同度，速度慢则会导致用户的流失。而在对 UPYUN（又拍云）来说，CDN 能做的远不止加速这么简单。CDN 与云的结合使得 CDN 能够提供更加丰富的服务，它为企业带来的功能也更具价值，UPYUN 把这种全新的服务定义为新一代的 CDN 加速服务。

2010 年之前，中国移动的网络基础设施还未大规模兴起，中国联通和中国电信的节点基本上可以覆盖到全国的用户，区区的 60 个节点基本上就可以做到全国覆盖了。从 2010 年开始，中国移动的网络基础设施开始实现大规模覆盖，用户数量有了大规模的增长，因此网络加速需求开始呈现逐年旺盛的态势。然而传统的 CDN 服务 2000 年诞生开始，直到 2010 年，技术并没有得到很大的提升。由于当时 CDN 服务无法满足 UPYUN 大部分的工作需求，因此 UPYUN 开始思考并着手构建自己的云 CDN 服务。在 2015 年之前，UPYUN 对外宣称的品牌形象始终停留在提供“存储”服务的阶段，一直到 2015 年，UPYUN 才将自己对外的品牌形象设为“云服务”，其主打产品为“云 CDN”服务。UPYUN 的云 CDN 系统早在 2010 年就开始研发，一直到 2015 年，总共经历了 4 年的研发周期。在这四年期间，基本上每年都会有一次技术上的迭代。

2010 年，起步阶段

2010 年，UPYUN 的云 CDN 节点数量只有 35 个，处于刚刚起步的阶段，基本和业内持平。前端的边缘节点使用 LVS 来进行负载均衡，下面每一个节点布设 5-10 台服务器，服务器应用系统采用 Nginx 和 ATS 系统。由于对于一个 CDN 系统来说，无法避免的两个功能是流量统计和

内容规则的推送与配置，因此早在 2010 年开始 UPYUN 就针对 Nginx 系统进行了大量的设计开发：使用 C Modules 去做业务功能的插入。

从整体架构的角度来看，从 2010 年起，UPYUN 就开始使用三层架构：

1. 边缘层，包括 40 个节点左右。

2. 中转层，包括 4 个节点。中转层不仅使得前端用户在回源时便于寻找更优的路线到达源站进行内容的拉取；同时还便于合并节点与路线，无需每个边缘节点都直接回归源站，从而避免源站的带宽堵塞；并且随着节点规模增大，中转节点可帮助源站分担压力，利于源站压力的缩减。

3. 数据中心层，该层使用 Nginx 和 ATS 应用系统，并向下连接 UPYUN 云存储系统。因此，整个的数据缓存有三层：数据中心缓存，中转层缓存以及边缘层缓存。

2013 年，基础建设阶段

2013 年，UPYUN 由原来的 C Modules 改为使用 Lua 来开发业务模块，并在系统中引入 Redis 集群来开发整个公网的 Push 推送规则，所有的用户规则配置都集中在用户中心里面，并通过 Redis 将所有规则推送至边缘层，边缘层的每个节点只需到 Redis 中心获取域名的访问规则即可，无需通过 Nginx 与数据中心进行任何交互。

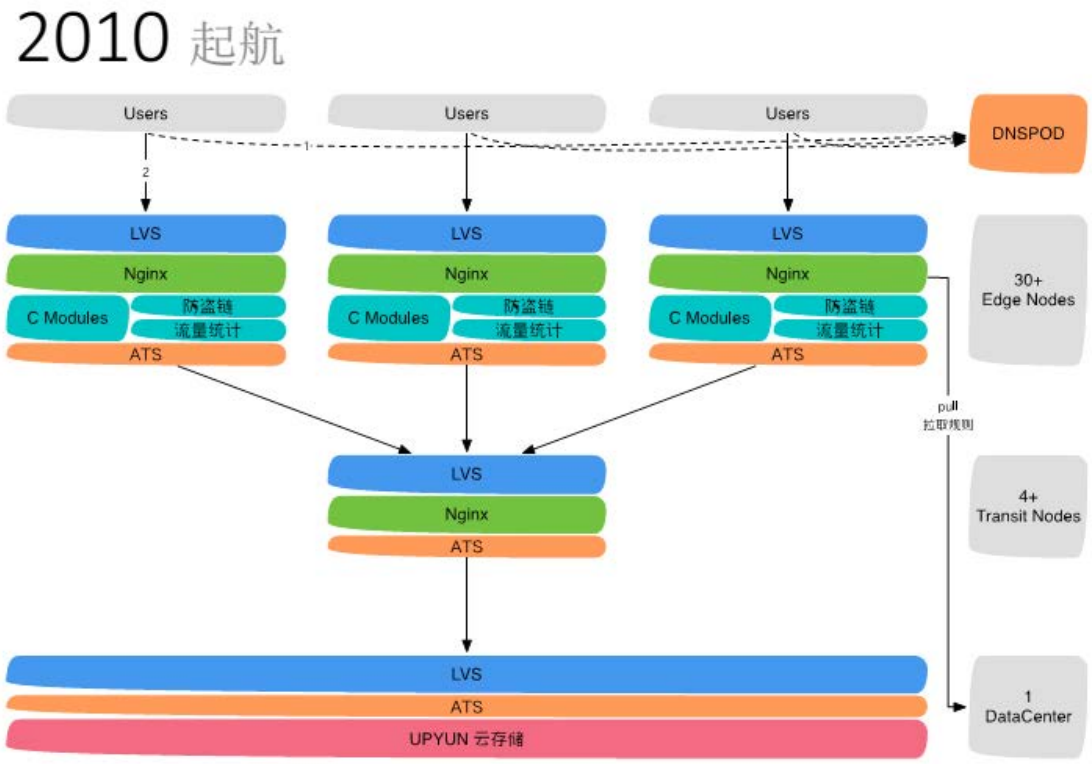


图 1 2010 年，UPYUN 云 CDN 架构（源自 UPYUN 黄慧攀）

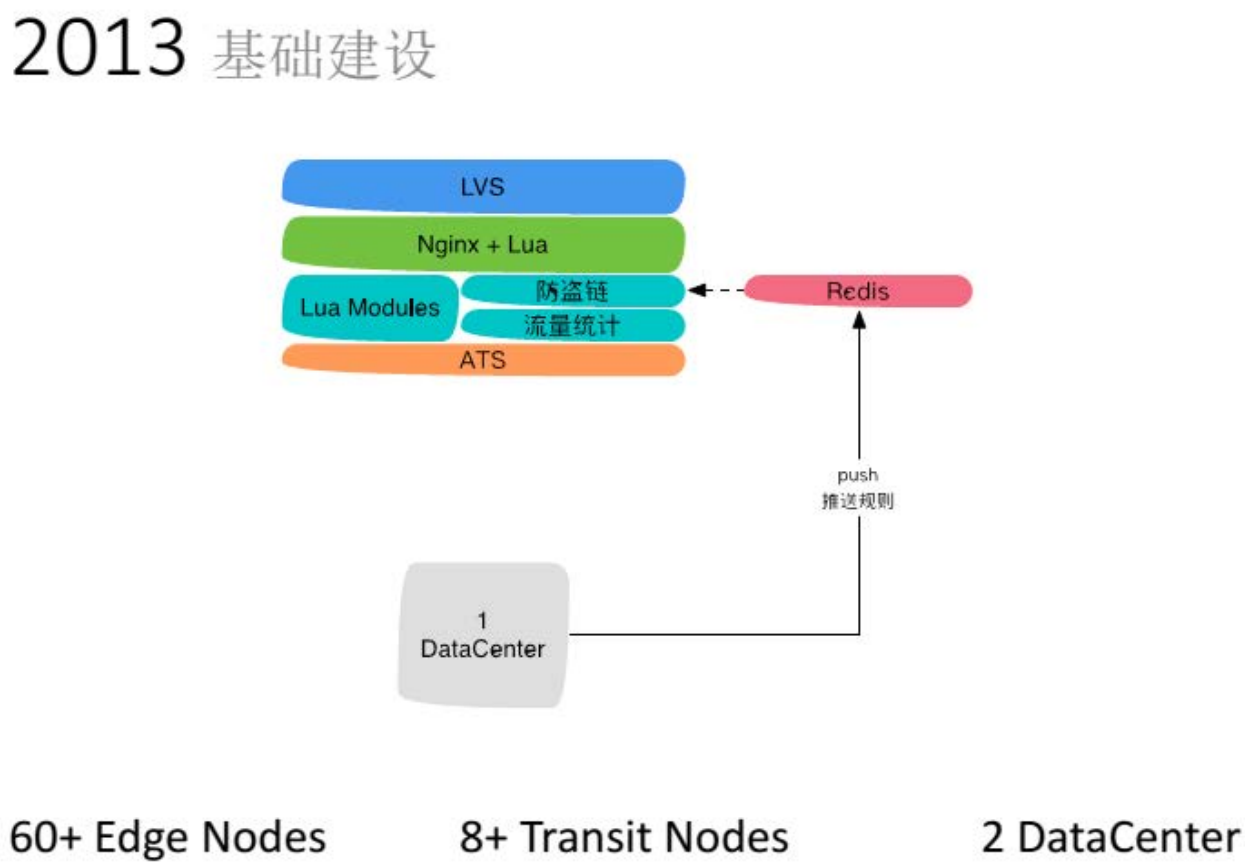


图 2 2013 年，UPYUN 云 CDN 架构（源自 UPYUN 黄慧攀）

2013 年，UPYUN 的主要工作都放在了原有的系统完全改进：即用 Lua Modules 去改写整个 C Modules。这件事成功地使得 UPYUN 面对大量客户化定制需求时，能将原有一个月的开发调试周期缩短为只有一周，系统部署也变得简单灵活了。

2013 年，UPYUN 不仅在软件方面有一个大的飞跃，在硬件方面也由原有的 40 个边缘节点增加为 60 个节点，中转节点也由原有的 4 个增加为 8 个，数据中心由原有的 1 个变为了 2 个。

2014 年，爆发阶段

2014 年对于 UPYUN 来说是业务大规模拓展与增强的一年。其中主要增强的是整个系统中的中转节点这一层。由于在第一代云 CDN 系统和第二代云 CDN 系统中，所需面对的只有云存储系统一个，但在 2014 年，UPYUN 开始尝试向外延更更多的云 CDN 服务，这势必需要连接到客户的源上，并进一步连接到原系统的 Lua 上去，所以 UPYUN 研发团队在中转层中多添加了一层”Nginx + Lua Modules”逻辑控制，这样做的目的主要是为了判断其域名对应的是客户的源还是 UPYUN 的数据中心。当然在其中还会涉及很多功能细节：例如多源站可以支持热备、云群负载以及多个线路的优化等功能。其中多线路优化是指支持判断不同的边缘节点以及所连接的中转节点之间究竟是什么线路，客户源站是什么类型的运营商（电信、联通还是移动），甚至可以将客户源站配置为海外源站。总而言之，UPYUN 会根据客户源站的情况进行特殊路由的设置，只需用户进行自主选择即可，该功能将于 2015 年 12 月对外发布。

2014 年也是 UPYUN 的云 CDN 在基础设施上爆发的年份。截止于 2014 年，云 CDN 的边缘节点达到了 130 个，中转节点达到了 16 个，这 130 个边缘节点所能够提供的带宽能够达到 1T，即物理服务器的网络处理能力可以达到 1T 的带宽。

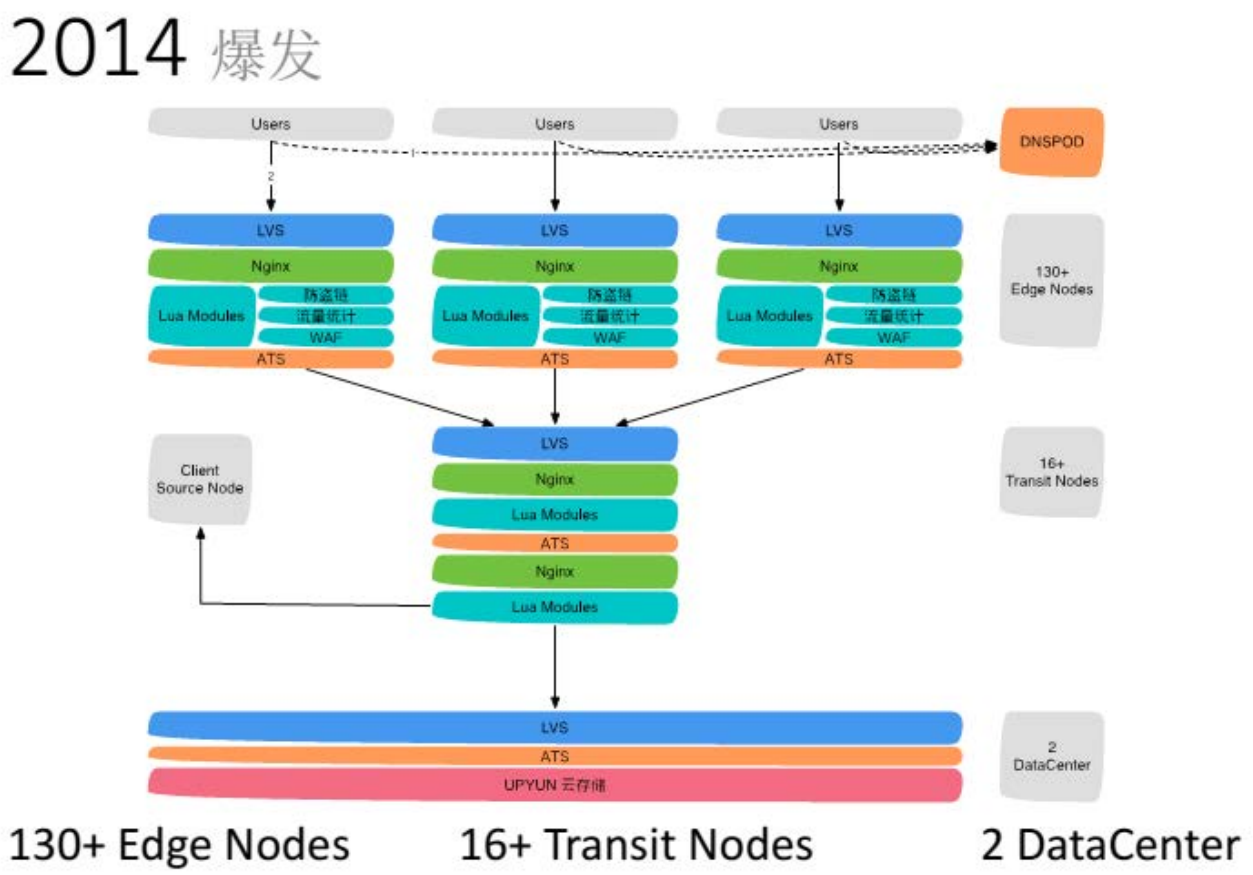


图 3 2014 年，UPYUN 云 CDN 架构（源自 UPYUN 黄慧攀）

2015 年，国际化推进阶段

2015 年，整个 UPYUN 云 CDN 的基础研发已经达到了稳定成熟的状态，并逐步开始进行国际化推进。一方面，UPYUN 公司将工作重心放在了大客户的定制开发和客户对接的工作上来；另一方面， UPYUN 正式开始为国际网络加速进行铺设。首先第一站就打通了从香港到浙江的光纤链路，并成功地针对中国内地的客户进行网络加速。例如在 2015 年 11 月由 UPYUN 承办的两场香港演唱会的大陆直播网络加速服务，首先需要考虑的就是如何把香港的内容快速地推送到内地，目前可以做到仅仅 30 毫秒的延迟就能将香港的内容推送到浙江的用户源站。

随着香港的光纤链路的成功建立，接下来 UPYUN 又着手建立欧美的核心节点，一个节点位于美国的洛杉矶，另一个位于法国，这三个国际节点的落地基本形成了 UPYUN 云 CDN 的国际加速骨干网络。与此同时，UPYUN 还准备于 2016 年在三个国际节点的基础上完成节点星形网络的扩张。迄今为止台湾和新加坡节点建设均已取得不小的进展，而这两个地区正是以香港核心节点为基础的；美国也铺设了 3 个边缘节点，这 3 个边缘节点将首先通过美国骨干节点，再与中国大陆取得通信；欧洲节点扩张工作也将于明年初启动。迄今为止，UPYUN 已经拥有 7 个国际加速节点，3 个国际骨干节点。

2016 年，第二次爆发阶段

2016 年， UPYUN 将此定义为第二次爆发阶段。在这一年，UPYUN 将会持续进行软件层面建设，其中一个比较大的目标就是建立专属的 DNS 调度系统。由于现阶段的 DNS 的主要目的为智能地区调度，该智能调度的技术架构需要进行一个二次铺点的计划——即在同一个省份或线路中会布设多个机房提供服务，为了兼顾到高效率的负载均衡控制，UPYUN 计划于明年开发专属的 DNS 系统。

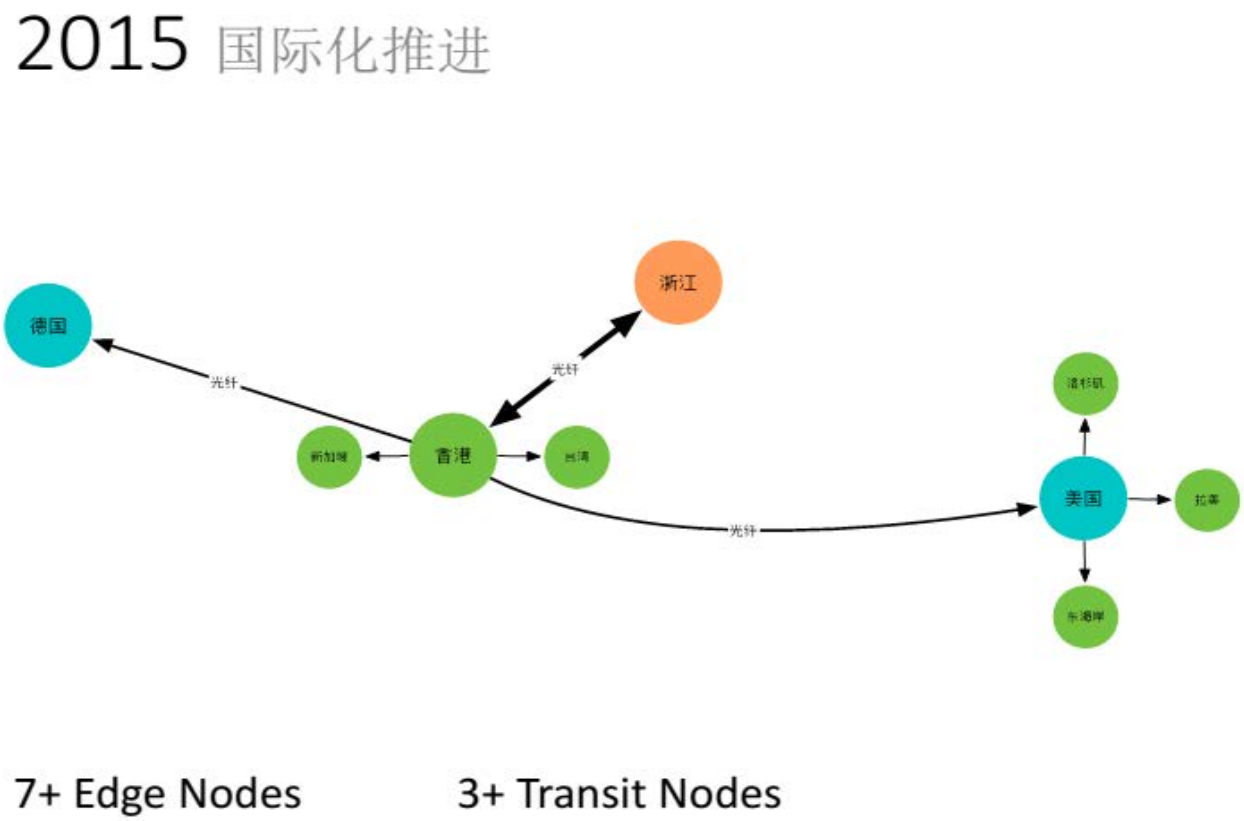


图 4 2015 年，UPYUN 云 CDN 架构（源自 UPYUN 黄慧攀）

图 5 2016 年，UPYUN 云 CDN 展望（源自 UPYUN 黄慧攀）

关于 DDoS/CC/WAF 等安全防护方面，UPYUN 也将会做更大的技术投入和资源投入。UPYUN 的 CTO 黄慧攀称，一旦用户受到了大规模的攻击，UPYUN 会有专人前去与客户进行沟通，并在 10 到 20 分钟内排查具体原因，受到攻击的体量，帮助客户判断是否持续保障服务还是暂停服务。

结语

从 2010 年到 2015 年，短短几年间，UPYUN 的云 CDN 产品经历了从无到有、从小到大、直到迈向国际化舞台的过程，离不开技术在背后的强力支撑，以及整个团队不断总结、思考的态度以及探索的精神。UPYUN 云 CDN 的技术架构演进之路，对于当今不断创新、不断转型的互联网行业也是很好的启示。

与“品高云之父”聊私有云

作者 魏星

写在前面：品高云是一家如此低调的公司，以至于鲜有相关的报道。这家最早成立于 2003 年的公司，起初主要为移动、交通、政府、大型企业等行业龙头客户提供顾问咨询、核心业务系统和办公系统的开发。2008 年品高软件开始研发云计算基础架构平台，两年后国内第一套商用基础云架构产品 BingoCloud 1.0 版正式亮相，2015 年 10 月 21 日，BingoCloud 6.0 发布。笔者与“品高云之父”刘忻有了一番交流，于是有了这篇文字。

品高云是一家如此低调的公司，以至于鲜有相关的新闻报道。这家最早成立于 2003 年的公司，起初主要为移动、交通、政府、大型企业等行业龙头客户提供顾问咨询、核心业务系统和办公系统的开发。2008 年品高软件开始研发云计算基础架构平台，两年后国内第一套商用基础云架构产品 BingoCloud 1.0 版正式亮相。

作为国内首个商用 IaaS 云产品，BingoCloud 自 2012 年 5 月的 3.0 版开始，除了与亚马逊 AWS 接口标准兼容的弹性计算、存贮、弹性计费等功能外，还推出了现在各大云厂商标配的应用软件商店。2015 年 10 月 21 日，“为云而生，因移动而盛”的 BingoCloud 6.0 版发布。

“三个臭皮匠”的北上与南下

在谈及品高云的创始阶段时刘忻说，三位创始人都是做产品出身，和所有创业者一样，满怀信心的他们远赴北京企图闯出一番天地。然而北京的大环境不同于南方，多年后刘忻回忆道，“广州的生意是开放的，同样的事情谁都可以来做。这种开放逼着你去面对各种竞争，也就是说你不能有明显的短板……北京不一样，北京需要关系、需要造势。”就这样，三人回到广州，开始了另一段旅途。

2005 年研发南航电子客票系统时，苦于无法在全球快速部署数据处理中心，同时受 Amazon 和 Google 的影响，2008 年品高投入研发并在两年后成功推出了国内首个商用 IaaS 基础架构云产品。此举奠定了品高云后来的发展之路。多年 IT 核心应用实践使得刘忻坚信，大量的

分布式计算存储群集需要一套机制进行灵活和动态管理，并总结出四则核心产品理念，即可演进、开放性与标准化、技术融合、高性价比与高可用性。

一个行业最重要的是开放标准

“深入了解任何行业都不难发现，标准化是一个行业最重要的东西，标准一定是开放的。”刘忻一字一顿地说，在国内互联互通是最大的难题，即便在国外、即使在 Java 开源世界，这个问题依然存在。还是拿民航电子客票来举例，需要转机的、尤其是国际航班的信息流怎么解决？怎么出票？旅客的行李怎么周转？对标准的认识不像念书时那样只是一个报文、一个格式，“标准必须是从二进制的角度设计，一定要开放。”刘忻解释道，虽然仅仅是从技术需求出发，但品高云服务的南航电子客票系统恰好成了对 OTA 搜索引擎最友好的系统。

因此，基础架构的设计理念要求能够在不受其设计者操控的软硬件环境中进行部署和执行。品高云采用开源的 Linux 作为基础架构云平台的内核，并通过定制化的手段，将各个管理控制子系统客户化进 Linux 发行版本，这样做的好处是：

- Linux 操作系统是目前最活跃的操作系统，开源和云计算都大大促进 Linux 的发展与创新；
- 采用标准的 Linux 内核，可以保持基础架构云平台的核心模块、可扩展性与世界同步；
- 利用其成熟、稳定的社区资源，规避大量的软硬件驱动问题，拓宽基础架构云平台的适用面；
- 与大量优秀的开源软件兼容，可以无需修改直接纳入到基础架构云平台。

以图 2 中的网络子系统为例，其中 SDN 部分自 2013 年开始研发，其网络控制器支持集群部署无单点，完全符合 ONF 标准，并且实现了所有云网络的功能，包括安全组（Security Group）、ACL、Subnet、VPC、VLAN 接 入、NAT、Route 等。目前业内只有如 Vmware（NSX）等少数厂商可做到，而开源社区版本的 OpenStack 自身并没有 SDN 控制器。

图 1

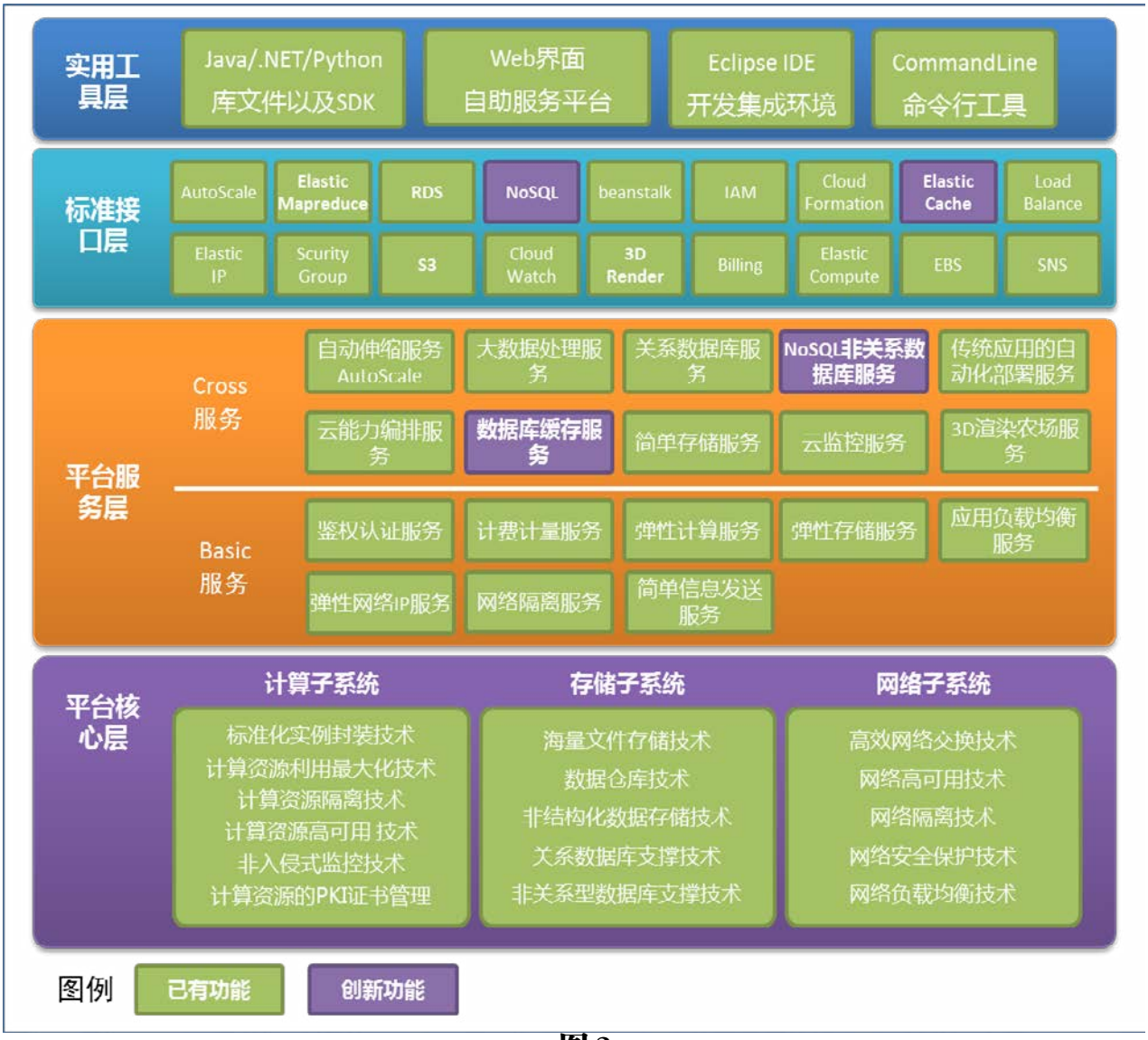


图 2

“我们并不擅长公有云”

云计算的本质是一个集群管理平台，其核心是资源虚拟化。业界公认亚马逊的产品堆叠做的特别好，把最根本的存储服务、网络服务以及应用部署服务都很好地进行了虚拟化，从而开始了搭积木的过程。刘忻认为，亚马逊做的好的地方不仅仅是单机计算资源的虚拟化，更是网络的虚拟化。

混合云最大的一个优势就是可以降低企业的 IT 运维成本。蒋清野先生在《浅谈“中国”语境下的公有云发展》一文中提到，公有云的成长要面临两个问题：一是用户增长，二是财务回报。“品高并不擅长公有云。”刘忻毫不讳言地说，经济离不开上层建筑，中国的经济环境还是以国企、央企为主，大的企业占经济的主体。“现在很多友商也开始转向企业市场了，为什么？技术不光玩的开心就行，要解决实际问题，产生收入才能继续。”

从业务的角度来说，公有云的存储增量是可以预测的，存储服务的产出预测一般是很线性的，因此财务模型很容易计算，这项服务相对比较容易赚钱。但网络就不是这样了，网络规划一定要往大了做，一旦规划做小了会带来无尽的痛苦。蒋清野先生在《浅谈“中国”语境下的公有云发展》同样提到了云产品的弹性，指出云厂商必须把自动伸缩这个概念应用到云主机集群上。刘忻在谈到公有云市场的时候总结说，“你的弹性不叫弹性。所谓弹性计算，是指用户的需求是弹性的，但厂商自己的基础设施从

来不是弹性的。”

正是基于以上的认识，刘忻得出“品高并不擅长公有云”的论断。而创业时候服务的都是大客户，例如一汽大众、南航、中国移动、网易等。“要有比甲方更好的格局和视角去理解其业务。”刘忻强调说，“要看到客户的客户在干嘛。”私有云或者说混合云，才是企业的真实需求。比如，广州地铁用三年时间从 VMWare 迁移到品高云，在迁移过程中，品高云对客户的计算模式分析的比客户还清楚，也因此与客户结下了深厚的友谊。

做一个聪明的跟随者

“目前国内云计算并没有什么自主研发。”刘忻坦言，云厂商要想做大做强，唯有通过做产品提升自己的防御边界。“不管用不用开源技术，你都能跟别人说清楚哪些是自己的东西。这属于自己的部分就是你的优势。”在谈及这个话题时刘忻举例说，在接触过的云厂商中，惠普的 OpenStack 白皮书写的最好，不但拎的清哪些是自己的哪些是开源社区的，甚至承诺负责相关的版权纠纷。这在现有的云厂商中是十分罕见的。

作为一位行业老兵，刘忻表示，“云计算拼的从来不是什么技术的先进程度，而是基础设施及其运营能力。真正做云的底层，远不如想象中那么高大上。云计算厂商要想体现出革命性的突破，需要挑战多年来业界一直坚信和死守的运营实践和技术基础。”而品高云构建之初并没有什么参考者，当时的亚马逊还没有公布其网络部分是如何实现的，“你在 AWS 上起一个主机，把一个简单的网络访问打进去。从响应速度来判断就知道它（AWS）的网络是一个回馈的系统，而不是一个预先配置的系统。”

VPC 作为事实上的行业标准，在面对国内企业的实际需求时是不可行的。在虚拟化引擎技术方面，品高云做一个跟随者无疑是最聪明的选择，刘忻这样解释道。至于基础设施，“机器肯定不值钱，数据才是（最值钱的）。”

云计算的创新点在哪里

云计算在技术上带来的挑战是“堆叠”。以容器为例，Docker 最强的地方是持续交付能力，容器与容器之间的通讯并不是 Docker 所擅长的。数据中心本身东西向的流量就很麻烦，多租户的网络处理、主机端口映射等等更加剧了问题的复杂程度。“让客户又上云、又上 SDN 是不现实的，云厂商一定要在物理层解决 SDN 的问题。”刘忻摇了摇头说，OpenStack 的网络模块比较令人头疼，品高云选择从控制器入手，这也是 SDN 的价值所在。

2013 ～ 2014 年，微信以暴风骤雨的方式教育了大到超大型国企小到创业团队的所有企业，移动化或者说 BYOD 使得企业面临新的办公协同与碎片化的问题，不同企业服务需求的差异直接变身成云厂商必须直面的挑战。所有企业，包括云厂商自己，都必须顺应这一移动化的浪潮。在企业市场，刘忻把服务好一个客户称之为“啃一个骨头”，并笑称“每个你啃完的骨头都是对别人的一个门槛。”例如，异构存储就是品高云一块块骨头啃出来的门槛，在这一方面，品高云显然是洞见了亚马逊在产品和服务方面的承载性和延续性。另一个比较有代表性的例子是 BingoLink，品高 BingoCloud 6.0 发布会上提到的“因移动而盛”实质上指的就是从企业服务中慢慢演化出来的 BingoLink。

“云计算的创新点一定是在业务与技术互相融合的地方。”刘忻一语中的，混合云产品必然出现在业务上下游相关的地方，必然是那些边界应用。以前企业每年花很多钱找人做核心系统；现在不光找技术人才，还要找对业务系统很懂的人来做决策。由此可见，真正的融合才刚刚开始，而这正是创新的起点。

从0到N，百度云的精益质量保障

作者 刘雯雯

导言

2012 年，百度云正式推出，随即推出一系列服务（如数据存储、文件分享、个人主页、图片智能管理、视频播放、离线下载、闪电互传、手机忘带、数据线等），构筑了一套完整的云上生态。与国内外主流厂商如三星、OPPO 等合作让云能力成为终端机标配，3 年实现 3 亿用户的增长。与此同时产品与技术不断创新，对质量保障工作带来了不小的挑战。本文将从测试技术、专项保障，体系经验三个维度讲述了百度云的质量保障工作。

一、质量保障体系

质量保障的核心目标是质量 & 效率并重，对于互联网产品来说诠释如下：

质量

- 不仅仅是功能可用性层面，需要关注用户体验。
- 不仅仅是上线前的质量保证，需要延伸至把关上线中、线上的质量。
- 不仅仅只停留在好坏的感性模糊认识，需要将质量概念量化、可视化。
- 不仅仅光靠抽样个例，需要大数据统计做强有力的支撑。
- 不仅仅只局限自身产品的质量，也需要关心竞品。

效率

- 加快产品迭代，唯快不破。
- 提高问题暴露，定位以及解决速度，快中求稳。

对产品建立质量标准，将其度量化并形成稳定

的、可衡量的产品质量 benchmark，对于个人云存储产品可以列出数据完整性、安全性、传输速度、在线消费体验等最核心的质量维度。线下以此作为发版标准，驱动产品质量迭代越来越接近目标；线上以此作为监控范围，对线上质量问题主动防御，加快应对。

“以质量为中心，以数据为驱动”为宗旨贯穿整个流程，将各种测试工具和方法融入进来，构筑一套全流程质量保障体系，如图 2 所示。

二、测试技术

线下集成持续化、测试服务化，以应用质量（QPS、SLA、性能）、业务指标、过程质量（代码覆盖率，千行 bug 率）一系列发版标准为目标，将自动化测试、性能、单测、异常等工具集成入构建—部署—quickcheck—slowcheck—release 的流程中，快速发现问题并解决，迭代质量。线下需要更多精力关注在异常和性能测试中，这些往往是线上问题多发区。

图 1

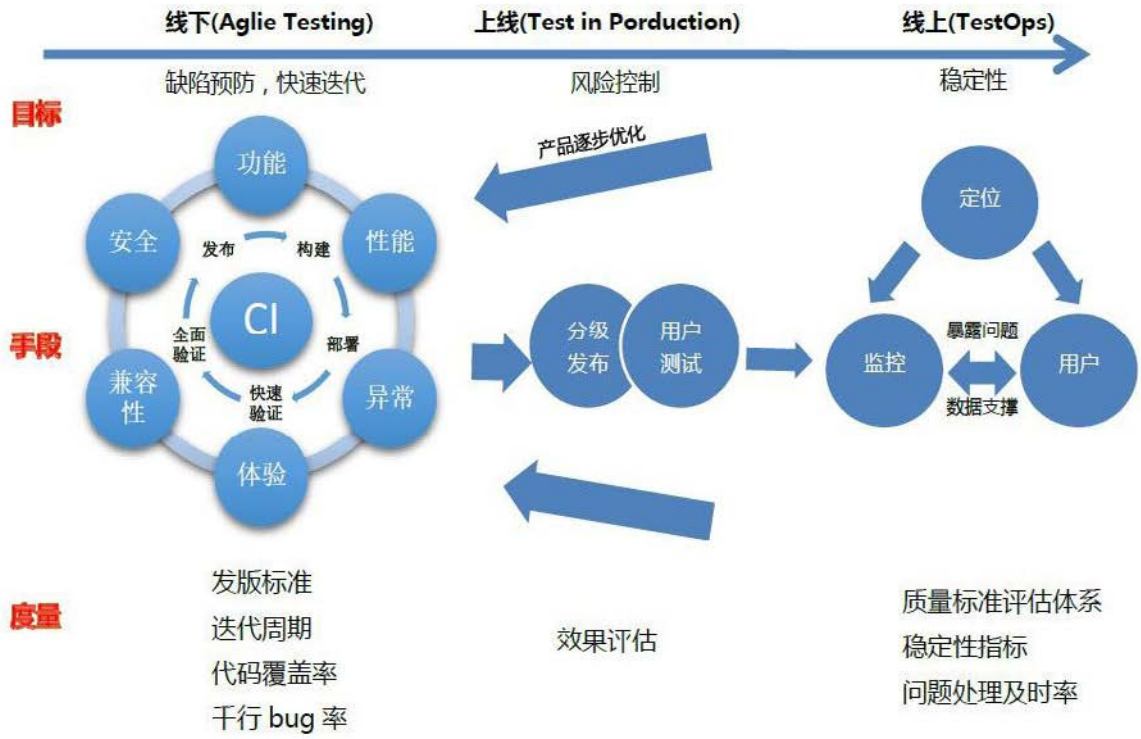


图 2

上线过程中灰度控制，把产品发布过程划分为多个级别，每个级别限制一定的流量和用户范围，并在每个级别对产品进行部署和验证的迭代过程。一方面逐步放量，小心验证，降低上线带来的风险；另一方面开展用户测试，让用户参与产品测试，加强与用户互动。让用户参与 beta 环境分为两种情形：被动命中（将同一特征的用户强制划分至小流量环境中）和主动邀请（邀请粉丝或有偿用户）。对服务器来说架构能够支持逐步放开流量，对客户端发版来说有一个平台支持哪些版本哪些用户能升级到 beta 版本，并且在小流量阶段要密切关注监控和用户反馈，将问题及时扼杀在萌牙阶段，不带到全量阶段。

线上监控 & 定位，从基础拓扑（网络、单机、数据库等底层服务）、服务稳定性（接口成功率、5XX、4XX 非预期返回码的占比等服务器可用性层面）和业务质量（上传、下载的成功率等用户功能层面的易用性）三个核心要素延展开全方位细粒度的监控覆盖，并从质量标准、质量防线和质量闭环三个维度进行质量建设：首先对产品建立一套完善的产品质量标准体系，并将其度量化，固定成 benchmark。紧紧围绕质量数据，组建从用户（舆情热点）、端（产品体验）、服务器（稳定性）到基础网络（SLA）的层层实时防护网，最后通过上线管理—报警中心—智能定位—故障通报的质量闭环环节落地，不断迭代优化，能够快到线上问题快速预警、定位及解决。

三、专项质量保障

（1）多副本分布式存储：旁路测试 & 线上数据检查，以数据完整 & 安全为使命。考虑灾备冗余、成本因素，云存储都会使用多个机房，跨机房的传输相比单机房的数据流动本身即增大了延迟，不同机房网络属性、机器性能等差异更对服务质量的保障提出了挑战。单一的机器性能测试已经不满足需求，需要引入旁路测试：复制线上的部署拓扑，进行等比例缩放，仿真线上的数据，在测试环境里重放，观察复杂部署和网络环境下服务的稳定性，辅佐一定的异常流量，评估系统的容错性以及灾难发生时预案是否能生效等。为更进一步保障数据的安全，对线上每日新增的数据较验各个副本的一致性及完整性。

（2）多机房 & P2P 流量架构：流量 diff 系统 & 实网系统 & 众测测速，传输速度体验。下载由源站 IDC、CDN 和 P2P 三部分承担，用户端、网络端、服务器云端的每一个环节都会影响速度。服务端的流量调度是根据用户地点、运营商网络、请求入口、文件所在机房、资源热度等多重属性对用户分配多个可带优先级的下载域名，让客户端充分并发及容错。多重维度的组合注定了调度策略的复杂性以及验证的难度，流量 diff 系统应运而生：在线下构造两套流量系统，一套线上代码环境，一套测试代码环境。通过回放线下真实流量，diff 前后调度是否符合预期，是否带来了非预期的变化。

P2P 公司测试与用户网络差异大，大规模用户节点的互连测试也受资源所限，只能借助线上真实用户的环境协助，即实网系统。将 SDK 包封装成独立的应用程序下发至众测或是粉丝用户本地进行下载，验证联通率以及联通速度。

对于运营商、第三方 CDN、P2P 这类非自身服务，云存储对他们的服务质量可感知以及可操控能力甚微，需要把控线上用户的传输质量需要邀请全国众测用户为监控节点，定期探测将地域性、运营商性属性、本地接入网络属性、服务

器连接信息并上报服务器，以做全国用户的大数据分析。从运营商、地域、域名、文件大小等多个维度展现网络服务质量，量化了速度大小、失败率质量数据，同时补充了域名联通性、第三方 CDN、骨干网络这类第三方服务监控的空白，为文件传输服务线上问题的监控、定位、解决和服务优化提供全面的数据支撑。

四、百度云 QA 成长史

首先，以百度云为例回顾一下产品从 0 到 N 一路护航过程：

（1）产品从无到有阶段，QA 进入“积本夯木，完善线下测试”重心阶段。12 年底的增设机房和机器的迁移，多机房网络延时大，传输质量差都会加剧服务的稳定性和性能降低的风险，随即推出的一系列大型运营推广活动，对线下测试带来不小的挑战。针对此探索出来的多机房测试方案以及旁路测试、运营活动测试方案很好地保障了质量，也对后续其他产品起到很好的借鉴意义。

（2）产品进入功能迸发式增涨阶段，QA 进入“TestinProduction”重心阶段。此时上线风险把控变得尤为关键，代码的变更是否影响用户的使用，新功能推出时用户的接受度如何？我们在服务器推行分级发布的落地，同时这一思路推广至端的发版，支持了很多 1.0 产品的发布，较快的收集到用户反馈并回馈到需求中迭代，同时也将上线中的问题收敛到小流量阶段，全量回滚数为 0。

（3）产品服务日趋便利强大之后，QA 转移“TestOPs”重心阶段。此时用户进入迸发式增涨阶段，“稳定、速度、安全、成本”成为关注重点。要让线上质量看得见，以质量度量的方式促进服务优化，我们开启了“云图”计划。

该计划经历了四个阶段：

1.0 阶段：利用自动化 Case 定期对线上环境运行的方式进行监控。但经过一段时间的运行，发现两大弊端。其一，单一 Case 对于线上不同的机房，不同的运营商，不同的文件等等维度的爆炸式组合的覆盖面仅仅是九牛一毛；其二，Case 的稳定性维护成本大，并不能协助发现线上问题，需要探索新的监控模式。

2.0 阶段：从线上核心功能数据安全性和数据传输速度入手，利用全量用户真实的端做智能节点做核心质量监控。

3.0 阶段：2014 年 5 月网盘有次故障，经过一段时间才得以恢复，原因是 DB 机房和 PCS 服务机房某一个链路出现丢包严重，导致 PHP-CGI 资源耗尽而拒绝服务。说明承载线上几亿用户的产品在自身的模块以及依赖的第三方服务越来越多的情况下，单一的核心质量监控已经满足不了需求，需要从基础拓扑、服务稳定性和业务质量三个核心要素延展开全方位、细粒度的监控覆盖。

4.0 阶段：从质量标准、质量防线和质量闭环三个维度进行质量建设。首先对产品建立一套完善的产品质量标准体系，并将其度量化，固定成 benchmark。紧紧围绕质量数据，组建从用户（舆情热点）、端（产品体验）、服务器（稳定性）到基础网络（SLA）的实时防线，最后通过“上线管理—报警中心—智能定位—故障通报”的质量闭环环节落地，不断迭代优化。

五、浅谈产品从 0 到 N 的质量保障之路

回顾网盘 QA 伴随着百度云产品的成长之路，完善质量保障体系的征程中多半是问题驱动的，以踩坑、填坑、防止再次入坑的模式前行。分级发布的触发时机也是因为线下测试的不完备导致问题在线上爆发影响用户，速度监控也是应对日益暴涨的用户关于数据传输的体验报怨。除了基本的服务稳定性监控之外开始加速、

做网络等底层。视频卡顿的业务监控同样也是因为几次网盘服务基本不可用的重大故障修复速度未及时跟上所催生。

以百度云三年的经验来看，一个成熟的产品都是经历“目标顾客—小范围实验—反馈修改—产品迭代—获得核心认知—高速增长”的正向良性循环中，从 0 到 1、再从 1 到 N 不断发展壮大的。至关重要的质量保障一环除了线下持续集成能力加快迭代，上线分级发布能力降低风险以及线上业务监控防御能力三类基础的工程能力之外，也需要不断相对调整重心，提早做好准备，跟上产品的节奏。

（1）产品从无到有创立时期，以最快的方式、以最少的精力验证市场需求为目标。质量的重点则是保证核心功能的正确性的前提下，与所有角色达成共识，精简并确定发版标准，招募第一批体验用户，将 MVP（第一个最小化可行化产品）尽早地将产品抵达用户，去接受市场验证，收集有需求的顾客最在意的是什么品质的服务，与此同时核心监控应同重要运营指标统计一起上线，密切关注数据变化。

（2）产品在从 1 进入 N 的发展时期，新功能进入爆发期，技术上需要支持邀请用户来体验新功能，同时产品在此阶段会借助一些运营活动造势，利用传播的力量将产品普及到更多的用户，激发用户更多的使用产品。使应对可能蜂拥而至的流量服务能正常运转成为质量保障的重点。

（3）当产品到达 N，积累到一定量的用户规模后，线上服务的稳定性成为质量的重中之重。建立一套自下而上的系统级、应用级、业务级和用户体验级监控，打造线下持续集成、上线管理、线上监控、用户反馈的质量闭环，事先及时预警发现故障，事后提供翔实的数据用于快速追查问题。

关于作者

刘雯雯，2009 年北京理工大学计算机学院硕士毕业，2010 年加入百度。现任百度云 QA 团队负责人，见证了百度云从 0 到 1 亿再到 3 亿用户的成长。

www.speedycloud.cn
北京迅达云成科技有限公司

扫描微信获得更多详情

SpeedyCloud迅达云

- 全球部署的云计算节点，面向全球用户提供云计算服务
- 性能卓越的云主机，为应用弹性扩展提供无限可能
- 分布式的对象存储服务，提供海量、稳定、安全的存储空间
- 灵活划分的SDN网络，轻松实现网络隔离
- 高效率的云分发服务，让网站平均提速3倍以上
- 精细控制的防火墙策略，全面保护主机安全
- 功能完善的可编程API，使资源可以灵活调度

SpeedyCloud

让云服务加速您的成功！

云主机　云存储　云数据库　云缓存　云分发　SDN方案　云安全　云DNS　负载均衡

JIMDB：一个大规模分布式内存存储的演进之路

作者 魏星

写在前面：Redis 是一个 C 语言编写的开源、支持网络、基于内存、键值对存储、可持久化的高性能 NoSQL 数据库，同时提供多种语言的 API，目前最新版本为 3.0.5。由于性能优异，已被国内外众多公司采用构建缓存集群。其中，京东对 Redis 的应用实践可圈可点。

在 2014 年的 QCon 上海大会上，京东云平台首席架构师刘海锋介绍了京东分布式内存存储平台（RAM store platform）。经过两年多的演进，这套系统已经支撑起京东几乎所有的在线业务。近日，InfoQ 采访了刘海锋，再探这个分布式内存存储平台的全貌。以下是采访实录：

InfoQ：你能简单介绍一下 JIMDB 这两年来的研发过程吗？

刘海峰：JIMDB 是一个以内存为中心的键值数据库，目前在我们京东多个数据中心部署了几千台机器，支撑了京东几乎所有的在线业务。JIMDB 的研发是一个逐步演进的过程，最初是从 Redis 改进而来，现在的数据模型是兼容但不仅限于 Redis，可以理解为 Redis 的一个超集。JIMDB 在京东的应用也不限于缓存，我们很多业务线已经把它当成唯一的存储。

具体来说，研发过程经历了三个阶段。

第一个阶段是从 2013 年底开始。当时我们在使用 Redis 时遇到了一些问题，为了解决这些问题，我们把 Redis 这个优秀的单机软件变成一个健壮的分布式系统。比如我们要做故障的手动、自动切换；要实现横向扩展，动态分片并且不能影响网络流量；要实现灵活的异步

复制、同步复制。虽然每个实例仍然是单个的Redis，但整个的集群能够实现故障的切换、横向的扩展和比较灵活的复制。

第二个阶段从2014年下半年开始。用一句话来概括这个阶段就是把JIMDB变成了一个完全托管、自管理、自运维的存储服务。因为Redis的引擎比较简单，为了实现这个目标，我们重写了整个内存存储的引擎，使之可以支持更细粒度的分片；复制协议部分也作了改动，不但支持全量复制、增量复制，还支持了任意局部复制。例如，我们经常遇到某个分片上的热数据过载的问题，这时候的解决方案是把这部分数据复制到其他的实例上，从而分散流量，这样我们就实现了动态的分裂、合并。在运维管理上我们引入了Docker，部署环节实现了整体的调度。

第三个阶段在2015年上半年启动规划，目前这个阶段还在进行中。我们已经把很多实现陆续推到了线上。即，不仅仅支持原来的哈希分片和原有的数据类型，我们还实现了在内存中支持B+树、支持按范围的划分、支持排序的遍历、复制协议也更健壮、更可靠，等等。第三阶段实现的特性将比Redis更加强大。当一个分布式存储支持按范围分片和复制的时候，这个系统无疑是更健壮和更可靠的。当然，这个阶段也存在很多的挑战，例如，哈希的分片相对来说比较简单，但我们要实现按范围、跨数据中心的异步和同步复制时，这个挑战就大的多了。

纵观这两年的发展，JIMDB从最初的几台服务器到现在的几千台、从单机Redis系统到自管理的分布式存储系统、用户只需要一个认证授权即可使用、支持京东线上一千多个业务（集群），是一个不断挑战的过程。

InfoQ：市面上已经有了大量的键值对存储数据库，并且已经在大公司的生产环境投入使用。为什么京东还要开发JIMDB？

刘海峰：这更多的是业务的需求。公司原来的存储应用百花齐放，有人用Redis、有人用MongoDB、有人用memcached等等。但现在的情况是，数据库用MySQL比较多；NoSQL几乎99%的都在用JIMDB。原因在于JIMDB有人维护、自管理、性能稳定，俨然已成为京东内部垄断性的应用。但在项目开始的时候我们也没想到会变成今天的规模，随着业务的发展，JIMDB一步一步演进成了今天的样子。

InfoQ：JIMDB在京东的主要应用场景是哪些业务领域？在这些应用场景里，跟其他内存存储相比JIMDB有哪些优势？

刘海峰：目前JIMDB的应用场景十分广泛。比如，现在依然有很多同事把JIMDB当Redis来用，很多人拿来当缓存用，也有很多人把JIMDB当唯一存储在用，用法不一而足。甚至，很多离线计算MapReduce的结果也直接存在JIMDB里；对京东的访问用户来说，大家看到的商品详情整个页面的所有元素、第三方show的广告、搜索推荐的结果等等，都是存在JIMDB里面。我们支撑了无线端、PC端很多业务。但总的来说，目前用量最大的业务是搜索和推荐的集群，另外单品页的用量也很大。因此这个系统的压力也很大。

跟其他内存存储相比，首先这个系统是从Redis演进过来的，Redis有的功能我都有，但是JIMDB在很多方面都比Redis更健壮、更稳定。其次，从Redis的视角来说，这是一个经过大规模生产环境验证的、更好的Redis。目前我们有六、七个人在维护这个系统。

InfoQ：对冷热数据的处理以及数据持久化JIMDB是怎么做的？在算法和数据结构方面JIMDB做了哪些优化设计？

刘海峰：持久化还是采用经典的方式——记日志、定期做快照。虽然这是一个很简单的方式，但是十分的有效。对于可预测的热数据我们会预先做分片，尤其是秒杀活动的内容；而对瞬时的热数据做动态调整是 JIMDB 最优先的特性实现，这跟我们电商的特性相关。目前京东的老机房大量部署了 JIMDB，新机房也部署了很多。

算法和数据结构方面，刚才已经举过一个例子就是 B+ 树。我们实现了 KV 的有序排列和按任意范围的分割、复制。这是我们新加的数据类型，其他的支持已经十分强有力了。

InfoQ：据我所知 Redis 所在机器物理内存使用率并不高（貌似没有超过实际内存总量的 60%）。针对京东这么大规模的在线存储，内存管理方面 JIMDB 是怎么做的？

刘海峰：这一方面主要做的工作是内存分配器的优化。Redis 的内存分配器是 jemalloc，我们依然沿用了 jemalloc，但是我们作了很多优化。此外，我们对比较大的 Value 做了特殊处理，比如我们不让大 Value 的存储占据内存，而把它写到磁盘中去。但是很多人会问 SSD 的问题，为什么不采用混合存储的架构？

其实在 2014 年的时候我们专门在线上测试过这个方案，当时大概对 10% 的集群采用了混合存储的方案。经过一段时间的测试发现：

- 首先，虽然 SSD 的成本有一定的优势，但实施工程是有成本的。基于磁盘去写一个有丰富类型的键值数据库其实是比较复杂的，即使是简单的键值都比较复杂，在遇到 Map、List、B+ 树等等，用磁盘的方式实现起来其实更为复杂、成本更高。也就是说，其实现和维护的成本太高，并且性能也不占优，整体工程成本远远超过内存存储。
- 第二个原因在于，内存的价格在逐渐降低，我们是能承受这个成本的。目前我们使用的内存也越来越大，预计明年我们将在单机上使用 1T 的内存。
- 第三个原因是系统架构方面的考虑。我们的数据中心有很多的机器，很多应用服务器和私有云服务器多数时候的内存利用率并不高，我们可以利用这些机器的资源做 JIMDB 的动态资源池，只需要在相应的机器上部署一个 JIMDB 的程序就好。这也极大地提高了整个数据中心的资源利用率。

基于以上三点考虑，我们选择用内存做存储。随着京东业务的发展，未来 1 ～ 2 年我们 JIMDB 的规模将达到 1 万台机器，当然，这 1 万台机器并不是专用的，而是 JIMDB 在整个数据中心的部署量。

InfoQ：你能谈谈 JIMDB 在“双 11”大考中的表现吗？

刘海峰：JIMDB 在“双 11”的表现很好，系统稳定，性能平稳。我们的新机房启用了大量万兆网卡，以前千兆网卡被打满的情况终于得到缓解。但任何系统都不可能没有任何瑕疵，现在看来 JIMDB 的问题更多的不是技术性的，而是系统到了一定量级怎么去运维，尤其是系统预警和运维操作流程方面的工作需要加强。随着公司业务的发展，运维正变得越来越重要。

我们花了很多精力来做监控与运维。JIMDB 核心的部分是一个个兼容 Redis 协议的 Server，除了网络部分，我们几乎重写了整个存储引擎。还是拿一个具体的应用场景来说吧，假设我一台机器上跑了 15 个实例，但是发现其中一个实例的流量极大，把其他实例的带宽都给占了。这个时候我需要对这个有问题的实例进行分裂和迁移。但是迁移的时候存在一个问题，由于进出流量都很大，迁移有可能是迁不动的。这个时候我们有一个限流措施，保证在不影响其

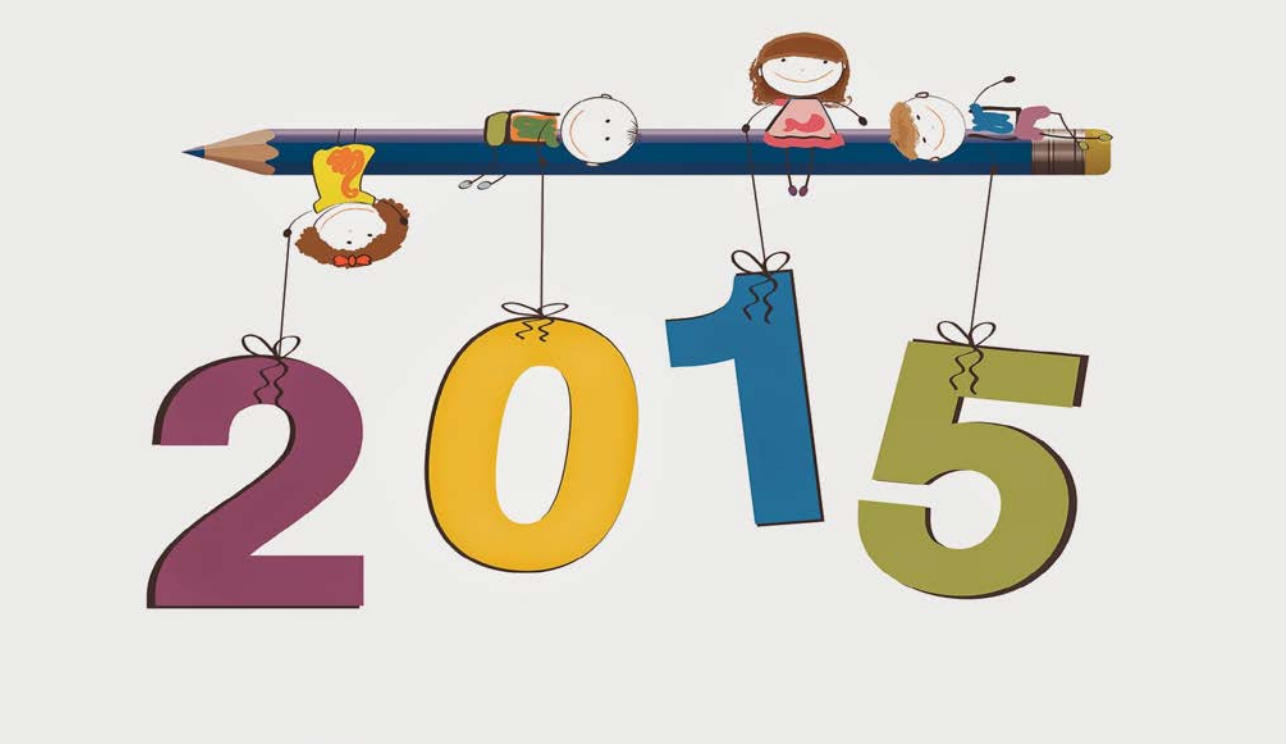
他实例带宽的情况下把有问题的实例迁走。

InfoQ: 你怎么看待 JIMDB 未来的发展趋势？或者，你能否谈一下内存云存储（RAMCloud）的未来？

刘海峰：我非常坚信的一点是，内存是存储的未来，特别是对一些有结构化的数据来说。随着内存成本的降低，以及在内存上实现存储的简洁和高效，这个趋势势不可挡。而且我认为 JIMDB 这两年做的工作是一种更务实的 RAMCloud 实现，它是业务驱动的、经过大规模生产环境验证的。

未来随着系统的发展，在功能上我们会以内存为中心，做有序的、KeyValue 的、丰富数据类型的大表支持。JIMDB 未来有可能会加入一些 SQL 的支持。目前要先把规模、稳定和运维做好。

简评云计算过去的这一年



作者 谢丽

过去十年来，谷歌一直在促进云计算生态圈的发展，也见证了这个领域的许多变化，过去的12个月也不例外。从容器的广泛应用到多重云应用程序，2015年真可谓是是云计算的转型年。近日，谷歌将云计算这一年的发展变化作了如下总结。

企业认识了云：对于大部分组织而言，云计算不再是“是否” 的问题，而是“何时”的问题。据最新估计，34%的企业计划在接下来的两年中将超过60%的应用托管到云上。大部分供应商估计也已经采取了措施支持企业负载。

容器迅速成为主流：甚至在一年之前，许多开发人员都还没用过容器。但在2015年，容器不仅用于测试，而且还在生产环境中得到了广泛应用。最近的一项调查显示，2015年容器应用增长了5倍。这部分得益于Docker和Kubernetes这类开源技术。

大数据需要大见识：2015年，大数据名实不符。调查显示，77%的组织都认为他们的大数据&分析部署失败了或者没有达到预期。原因很清楚，就是大数据复杂度高。当然，孤立的团队、高维护成本的设备和缺少更好的工具也是问题的一部分原因。这一问题的解决方案，可能就是要有更便于数据科学家使用的工具和数据——他们的领域知识可以释放大数据的真正价值。

机器学习人人可用：机器学习的潜在好处早就得到证明了。现在，计算机和数据中心处理能

力的增加最终使这种潜能成为现实。开源软件库（如 TensorFlow）也推动了机器学习的发展。

IoT 的未来：谈及 IoT，大多数人都会想到消费者。但实际上，企业才是最大的 IoT 使用者。据估计，到 2019 年，在 233 亿连接设备中，企业市场将占 91 亿。这意味着，基于流的数据处理将成为任何 IT 战略的关键部分，人们对 Google Cloud Dataflow 和 Apache Spark 这类技术的兴趣也会相应地增加。

API 作为一种商业模式壮大起来：向开发人员按需提供应用程序服务现在已经成为一种经过验证的商业模式，诸如 Twilio 和 Okta 这类“独角兽”公司的出现就是证明。

混合云出现：多重云架构已不新鲜，新鲜的是多重云编排工具（如 Kubernetes 和 Spinnaker）的部署速度。据估计，到 2017 年，50% 的企业将采用混合云。

此外，随着云平台生态圈的发展，市场合并加剧。例如，Racespace 证实，将从提供云服务向为第三方提供云基础设施转型；而惠普将关停 Helion。最后还有一点，就是环保潮流。客户希望他们的云绿色环保，而争论点在于，如何将大型的、泛区域数据中心的环境效益带到本地数据中心。

Lars Kurth谈开源安全流程之一：云安全

作者 张天雷

【编者的话】随着云平台的日渐流行，其面对的安全问题也越来越严重。近日，Xen Project 的顾问委员会主席 Lars Kurth 分享了其对开源安全流程的理解和看法。作为该系列文章四部分的第一篇，本文介绍了 Lars 对于云安全背后的理论。

随着去年的 Heartbleed 以及最近的 VENOM 漏洞的出现，运行互联网服务和云系统的软件遇到了前所未有的威胁。很多人认为，如果要保证软件尽可能的安全，就需要减少程序暴露给攻击者的攻击机会。然而，现实情况远非如此。其关键在于 IT 团队真的需要判断攻击者是否知道一个可被利用的漏洞的概率。接下来，本文就通过介绍虚拟和云环境相关风险的本质，来详细阐述上面的观点。一旦有了这样的框架，文章再讨论如何把理论变为现实，从而分析现实问题。

风险的含义以及它和漏洞 / 利用的关系

当人们讨论一个系统是否安全时，非常容易跌进一个二元的陷阱：一个系统要么处于可以被攻破的非安全状态，要么处于完全无法被攻破的安全状态。接下来，为了便于理解，文章反过来先谈论风险的事情。在实际生活中，任何事情都可能存在风险。软件也面临着同样的情况。因此，一个安全的系统指的应该是该系统被攻破的风险相对较低；而一个非安全的系统则意味该系统被攻破的风险相对较高。无论是何种系统，（安全）风险肯定存在，只是不同

系统对其容忍程度不同。那么，究竟风险的本质是什么呢？风险又从而来呢？

在云计算和虚拟环境中，系统的输入和输出是系统恶意负载的主要来源。然而，系统的负载是多样的，而且与用户行为有关。我们不能假设所有的云用户都可以做正确的事情。因此，迁移风险需要依赖另外两项技术：划分（compartmentalization ）和最小特权原则（Principle of least privilege）。

划分是将对虚拟机、进程、用户和数据等资源的访问区分开来，而且在发生问题时帮助隔离。而最小特权原则是将访问权限限制到能保证系统正常运作的最低水平。例如，一个服务器上的常规用户并不需要 root 访问权限，或者在某些时候不需要拥有安装软件的权利。

在云计算平台和数据中心中，虚拟机和容器就是划分的最基本形式。它们就是依赖 Hypervisor 或 Linux 来强制分离最基本权利的“可信域”。接下来，本文就尝试评估攻破虚拟层、访问其他虚拟机或其他容器中的数据或资源的风险。尽管“攻破”可能是最好的描述，它也可能引起误导——从字面上理解，好像需要暴力手段来攻破软件的防御策略。

在现实情况中，这类风险的来源就是漏洞（Vulnerability）。漏洞指的就是攻击者在一个可信域内能够利用的代码 bug 或者配置 bug。攻击者在利用漏洞时所使用的代码或者技术就叫做利用（Exploit）。如果系统中存在漏洞，而攻击者又正好知道，他就可以进入系统。而如果系统中不存在漏洞或者攻击者不知道漏洞的存在，他就无法进入系统。因此，“攻破”需要的并不是力量，而是对于漏洞存在的感知和掌握。

评估漏洞以保护系统

如上所述，一个软件漏洞就是一个错误。它可以是代码中的错误（例如，软件行为与程序员预期不相同）或者配置中的错误（例如，软件配置错误导致其行为不符合预期）。当评估系统安全时，这两类错误都需要认真考虑。

以编号为 CVE-2015-3456 的 VENOM 漏洞为例。VENOM 是一个存在于 QEMU 的虚拟软盘驱动器（FDC）代码中的安全漏洞。该代码存在于 Xen、KVM 以及 Virtualbox 等多个计算机虚拟化平台之中。VENOM 漏洞可允许攻击者从受感染虚拟机中摆脱访客身份限制进行 DoS 攻击，并很有可能获取主机的代码执行权限。为了规避该漏洞，Xen 工具栈自动配置 QEMU，使得 FDC 被直接关闭；而 KVM 用户可以手动配置系统，从而不使用 FDC。然而，即使是在这种情况下，QEMU 还存在另外一个 bug，使得 QEMU 中的 FDC 并没有实际关闭。

简而言之，管理员想要避免漏洞攻击就需要关闭一切可以被关闭且没被使用的选项。当然，这样的经验同样适用于软件：为了避免 VENOM 这样的漏洞，Xen 工具栈需要关闭不被使用的大量 QEMU 设备。

IT 团队可以从《行尸走肉》中学习到的东西

在当今复杂的软件环境中，安全漏洞已经成为生活的一个方面。IT 部门需要时刻警惕攻击者进行漏洞的识别和发掘。在全世界任何行业的公司中，该风险都真实存在，并将一直存在。尽管《行尸走肉》中电视剧的场景与现实的技术世界相距甚远，它却能够给公司以强烈的震撼，令其采取行动加强系统防御。

正如电视所演，假设你和你的同事是目前所知的最后存活的人类。你们需要四处逃窜，在旧

文明中残喘生存。一旦到一个地方，你们就一直待到该地方的资源使用完毕为止。
以下就是行尸的行动规则：

- 他们昼夜行动，通常会被声音所吸引。由此，小股行尸渐渐汇聚，最终形成一大波行尸群。
- 他们非常强壮，可以轻易破门或破窗而入。
- 然而，他们无法思考，不能识别门、窗户或者墙壁。只有一大波行尸聚集时，才能根据声音自动汇聚到门窗附近或者直接打破墙壁或围栏。

因此，残余的人类就需要保持安静，并确保每一个门、窗户或者任何开放的出入口处于关闭状态，而且防御的墙或围栏足够的坚固和高大。一旦留下任何一个破绽，并被行尸发现，那么故事就结束了。

尽管保卫人员早已知道保证门窗的安全并不是很难，但是人总是会累或者处于焦急状态。因此，即使付出全部的能力，也不可避免会出现一个门或窗忘记关闭。如果够幸运，行尸可能并没有发现。就如行尸一样，计算机的攻击者也在寻找任何可能攻入的“门”，而防御者很难顾及到所有的“门窗”。

小或者简单的门窗是很容易保卫的，而大的门窗就很难去保卫了，对于围栏也存在同样的道理。在给定时间内，修复五个小的窗户可能比修复一个大的窗户要简单很多，因而窗户（漏洞）越小越好。

多层保护，又叫“纵深防御”（defense-in-depth），是最好的防御手段。如果你可以保卫附近建筑或围栏的安全，并关闭房屋内的所有门窗，黑客就需要找到若干漏洞才能攻入系统。而且，如果在这段时间内，你还可以加强安全机制，或者添加新的门进行防御。这就是系统架构中“划分”的应用。

在下一篇文章中，Lars 将进一步挖掘安全漏洞以及他们在 Hypervisor 和容器中的区别。读者可继续阅读《开源安全流程——第二部分：容器 vs. Hypervisor——保护你的攻击面》。

编后语

《他山之石》是 InfoQ 中文站新推出的一个专栏，精选来自国内外技术社区和个人博客上的技术文章，让更多的读者朋友受益，本栏目转载的内容都经过原作者授权。文章推荐可以发送邮件到 editors@cn.infoq.com。

作者 魏星

微软 Azure 安全中心正式启用

近日，微软宣布正式启用 Azure 安全中心公开预览版。作为微软为企业提供全面灵活的安全平台这一愿景的一部分，新服务将为 Azure 用户提供云资源的安全监控和管理功能。按照微软的说法，他们的目标是让 Azure 成为一个可信赖的云，用户可以放心地部署。

Google Cloud Shell 继续免费到 2016 年底

今年 10 月，Google 发布了 Google Cloud Shell，它是 Google 云平台的一个功能，能够帮助开发者在任何浏览器中使用命令行管理基础设施和应用。发布时，Google 宣布了 Google Cloud Shell 的 Beta 版本将在 2015 年全年免费。在 2015 即将结束的时候，Google 宣布将免费日期延伸到 2016 年底。

VMware 宣布开源“Photon Controller”基础设施控制器

在 2015 年 DockerCon EU 中，VMware 通过企业的 Github 账号开源了他们的 Photon Controller 产品。Photon Controller 是 VMware 中 Photon platform 的一个组件。它为了优化“容器与云”工作负载，作为一个基础结构堆栈被设计出来。

Docker 增强容器技术的安全性

2015 年 11 月 16 ～ 17 日，在巴塞罗那举行的欧洲 DockerCon 上，Docker 宣布了一系列新的安全增强。这些增强功能包括容器镜像的硬件签名、镜像内容扫描和漏洞检测、用户细粒度的访问控制策略等。此外，Docker 还在此次的欧洲 DockerCon 上宣布了容器用户命名空间的可用性，允许容器和 Docker 守护进程

之间拥有不同的权限。

UnitedStack 有云宣布完成 C 轮融资

12 月 16 日，UnitedStack 有云宣布完成 C 轮融资，该轮融资由思科和红杉资本投资，具体数额未公布。UnitedStack 有云成立于 2013 年 2 月，致力于为全球数据中心打造统一开放的云平台。是中国市场上第一个提供托管私有云（Managed Private Cloud）服务的云公司，也是中国云计算领域第一个提供公有云和托管私有云区域节点完全一致的高可靠云服务的云公司，2015 年 5 月成为第一个通过 OpenStack 基金会互操作性测试的中国云计算服务商。

云智慧 Cloudwise

云智慧（北京）科技有限公司

应用全生命周期的性能管理解决方案

监控宝 | 透视宝 | 压测宝

应用性能管理(APM)
面向多平台和多语言，快速、持续分析应用性能瓶颈

移动端真实用户体验
实时分析移动端性能及用户行为

浏览器端真实用户体验
深入分析浏览器端的真实用户行为与性能体验

基础设施性能分析
面向应用，全局透视服务器、服务和数据库性能

应用监控
应用的全栈系统级监控，确保系统级SLA

网站监控
7×24全天候持续监控网站，故障信息快速定位、秒级告警，工作效率提高10倍

API监控
中国首家API监控，关注面向业务流程的用户体验

真实用户压力测试(TEST)
基于SaaS的真实分布式规模压力测试平台

云智慧

为您提供极致应用性能解决方案

北京 010-62680693　　上海 021-61728826

广州 020-87503676　　成都 028-66824386

InfoQ 中文站　2015迷你书

云生态专刊 2015年05期

《云生态专刊》是InfoQ为大家推出的一个新产品，目标是“打造中国最优质的云生态媒体”。

开源启示录 第二季

开源软件的未来在于建立一个良性循环，以参与促进繁荣，以繁荣促进参与。在这里，我们为大家呈现本期迷你书，在揭示些许开源软件规律的之外，更希望看到有更多人和企业参与到开源软件中来。

顶尖技术团队访谈录 第四季

《中国顶尖技术团队访谈录》·第四季挑选的六个团队虽然都来自互联网企业，却是风格各异。希望通过这样的记录，能够让一家家品牌背后的技术人员形象更加鲜活，让更多人感受到他们的可爱与坚持。

架构师 月刊

《架构师》月刊是由InfoQ中文站针对高级技术开发和管理人员所推出的电子刊物。