云技术驱动的架构演进与变迁

组件化、系统化、微服务化

云平台和服务模型

社交 / 服务	企业解决方案	地产 / 物业 解决方案	政务解决方案
开发 / 测试	即时通讯	图像 / 图片处理 与检索	数据管理与分析

主机入侵检测系统

KVM/Docker安全组

网络流量监测系统

应用防火墙

数据库审计系统

边界防火墙

堡垒机

安全扫描引擎

Tomcat	Apache	Jetty	NodeJS	RabbitMQ	Spark	Storm
Mysql	MongoDB	Neo4j	Kafka	Zookeeper	Redis	Memcache

自动化部署

监控告警平台

日志分析平台

配置管理数据库

数据库管理平台

自服务平台

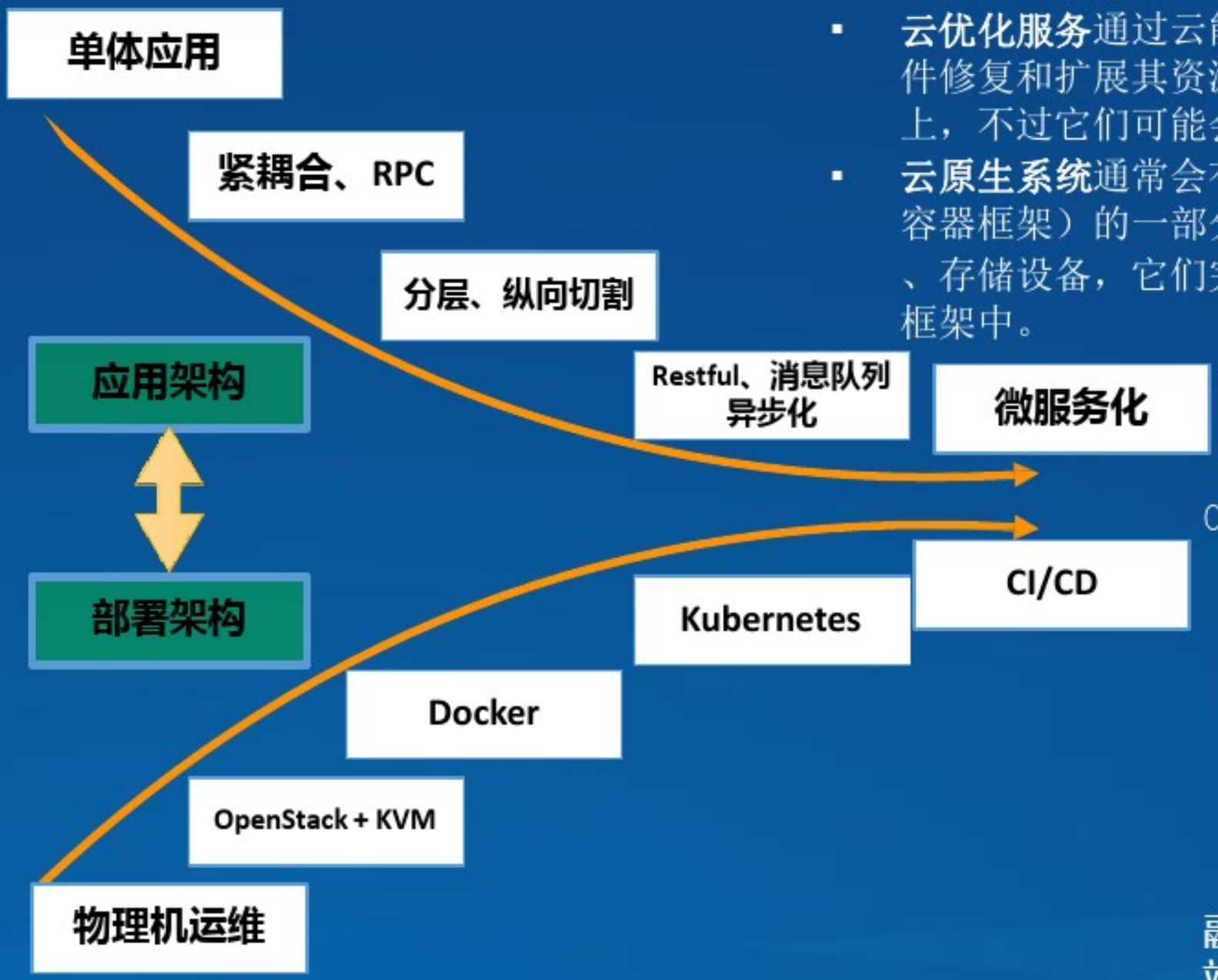
KVM | Docker | OpenStacK Trove(Liberty) | OpenStacK Sahara(Liberty)

OpenStack Core Services(Juno) | Swift | HDFS(HBase)

计算 | 存储 | 网络

多数据中心 | 开发、测试、压测、预生产、生产环境

云技术与架构演进之路



Gartner对运行在云端的三大类应用工作负载进行了定义：

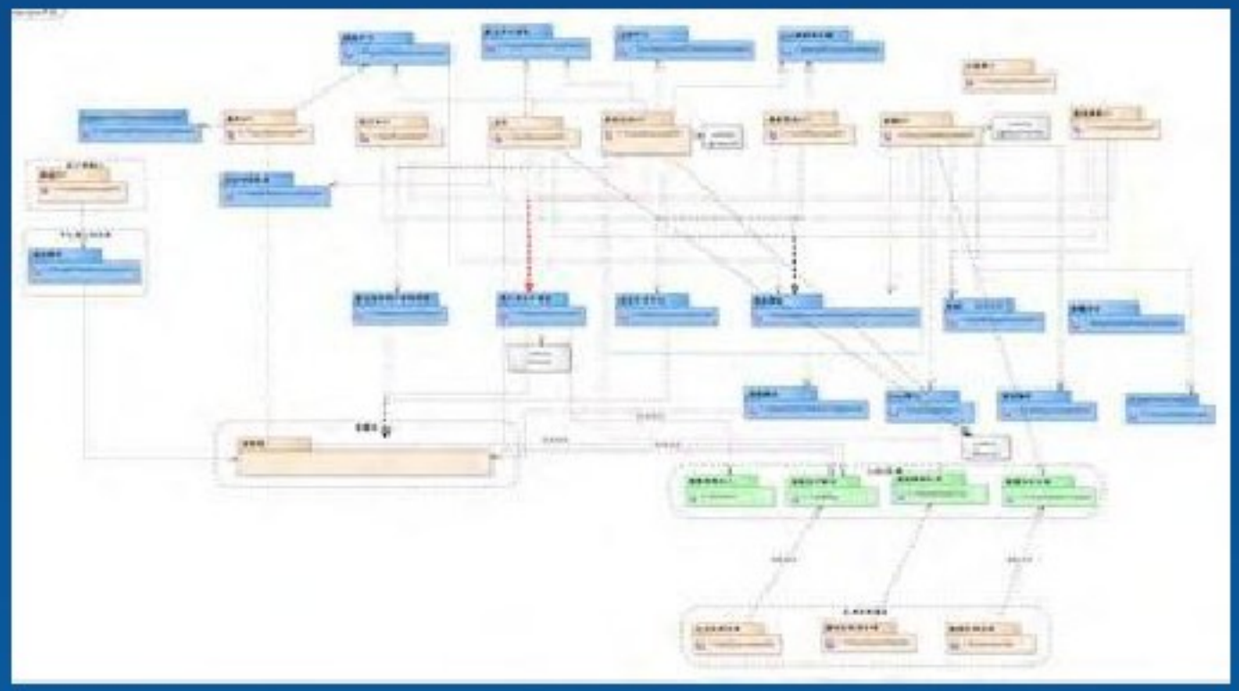
- **云托管系统**被认为是摆脱基于硬件的昂贵解决方案的“初级阶段”。云托管系统从专用的手动管理硬件转移到了云配置的裸机和虚拟服务器。
- **云优化服务**通过云能力支持系统的可用性和性能，例如允许软件修复和扩展其资源。云优化后的工作负载仍然运行在服务器上，不过它们可能会通过编排系统实现虚拟化和自动配置。
- **云原生系统**通常会有其他的抽象层作为应用架构（例如PaaS或容器框架）的一部分。例如，云原生应用不会有专用的服务器、存储设备，它们完全嵌入到了隐匿的传统基础设施资源的云框架中。

OpenStack 的两个发展阶段：

- 第一阶段 OpenStack as IaaS
- 第二阶段 Solutions on OpenStack
  - 用户体验
  - 可管理性
  - 性能
  - 稳定性
  - 扩展性

大数据、NFV、物联网、区块链、金融企业的核心交易系统、电商企业的核心网站

没有治理的应用架构

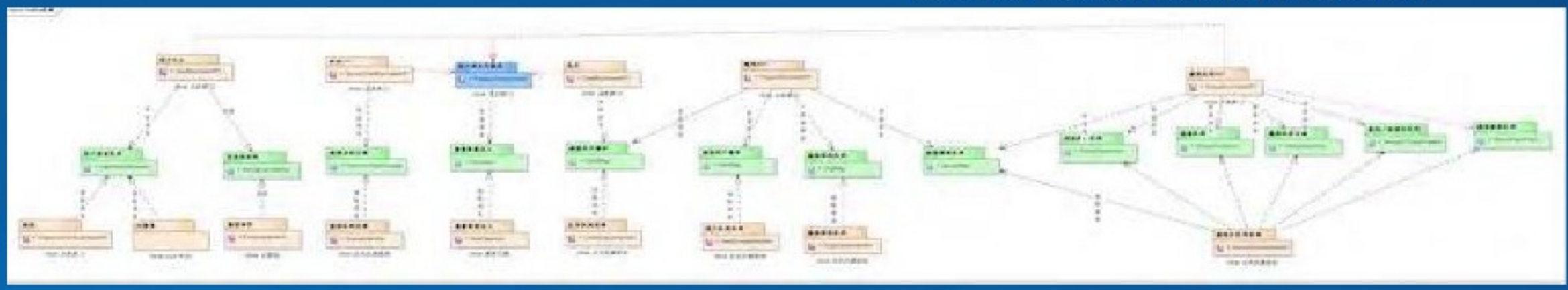


软件设计与架构痛点：

- 大量开源组件的使用：
  - 多种数据库: Mysql、MongoDB
  - 多种中间件：Zookeeper、Kafka、dubbo、Redis、Disconf...
- 中间件使用：
  - 高可用缺失
  - 共享与非共享缺规则
  - 聚合组件的泛用
- 架构设计：
  - 紧耦合
  - 缺乏子系统边界
  - 调用关系复杂
- 应用配置与管理
  - pomvs. Disconf
  - Jekins

传统应用部署的痛点

- 运维人员参与的人工或者半自动化方式，错误率高
- 多环境部署，需要大量时间，效率低下
- 部署过程中，相关数据无法保存，难展现，难追溯
- 需要较多运维人员参与，人力资源成本高
- 重复性劳动，对人员成长及团队稳定不利



云技术的演进

物理化运行环境

- 众多数据库、中间件、缓存、应用“精巧”地运行同一物理服务器

OpenStack+KVM

- 批量应用部署
- 高可用中间件部署

ansible自动化

- 批量应用部署
- 高可用中间件部署

“Docker”化

- Tomcat、 Apache应用Docker化

heat + Cloud-init

- 系统环境定制化
- 应用配置

“环境”自动化

综合ansible + Heat搭建复杂的完整环境

- 高可用数据库集群自动化
- 中间件、缓存集群自动化
- 应用批量部署自动化
- 完整的“压测”、“预生产”环境自动化

Kerbernetes

- 中间件、缓存的Kerbernete适配
- 系统架构“微服务”化

Docker容器技术的使用与推广

Docker OpenStack平台稳定性测试:

OpenStack平台本身是一个SOA的项目，具体服务的参数设置需要依据集群规模，服务搭建架构等进行相关测试和调优。Fake是OpenStack Nova Compute下的一个Driver，绝大多数Compute API走到这里简单处理后返回成功。

通过使用Docker来封装Nova Compute，并在Nova配置中使用fake。这样每个Docker Container便成为一个虚拟的Nova Hypervisor Node，便可以模拟Controller 集群管理超大量Compute节点的状况；同时fake driver收到请求直接返回成功的特性，让我们可以测试超大量的VM同时创建和同时销毁时给控制节点和MQ带来的压力状况。

Docker Nova项目:

服务云化这是第一步。需要其他开发团队逐步熟悉面向容器的开发以及我们对Docker本身逐步的摸索，才敢真正把环境切换到Kubernetes/Mesos上来，进而推进Magnum。

通过Nova API调度Nova Compute生产 Nova instance，而Nova instance的类型由具体配置的hypervisor Driver来决定，这里我们设置Docker作为Driver就可以让Nova Compute节点生产Docker。NovaDocker项目来自于社区，我们结合社区代码进行了一定量的修改，并且在镜像定制，具体使用上有一些自己不同的方式.

Kubernetes & CI/CD:

- Kubernetes内部负载均衡器实现及其访问模式
  - kubeproxy内部负载均衡器实现原理
  - kubeproxy NodePort
  - Kubeproxy ServicePort
- 外部访问k8s Service
  - 直接访问POD
  - 外部负载均衡器
  - 通过NodePort访问内部负载均衡器
  - 内外结合
- Kubernetes对接ELK

OpenStack 自动打包:

私有云平台压力通常没有公有云高，但是个性化定制更强。我们内部的定制化需求也很高（例如集群中计算资源的主机级别和机架级别的反亲和等等）。内部维护了两个OpenStack的版本，大量编译的依赖和依赖的冲突以及编译后的脏数据成为我们的痛点。

将OpenStack所涉及的包括Nova、Nueturn、Glance、Cinder、Trove、Sahara等项目的编译依赖环境统统放进一个Docker Image中。通过参数来指定要编译的OpenStack的版本和组件。

这样便不需要再维护一个编译环境了，只需要维护编译镜像和GIT库内部源码。可以任意生成打包环境

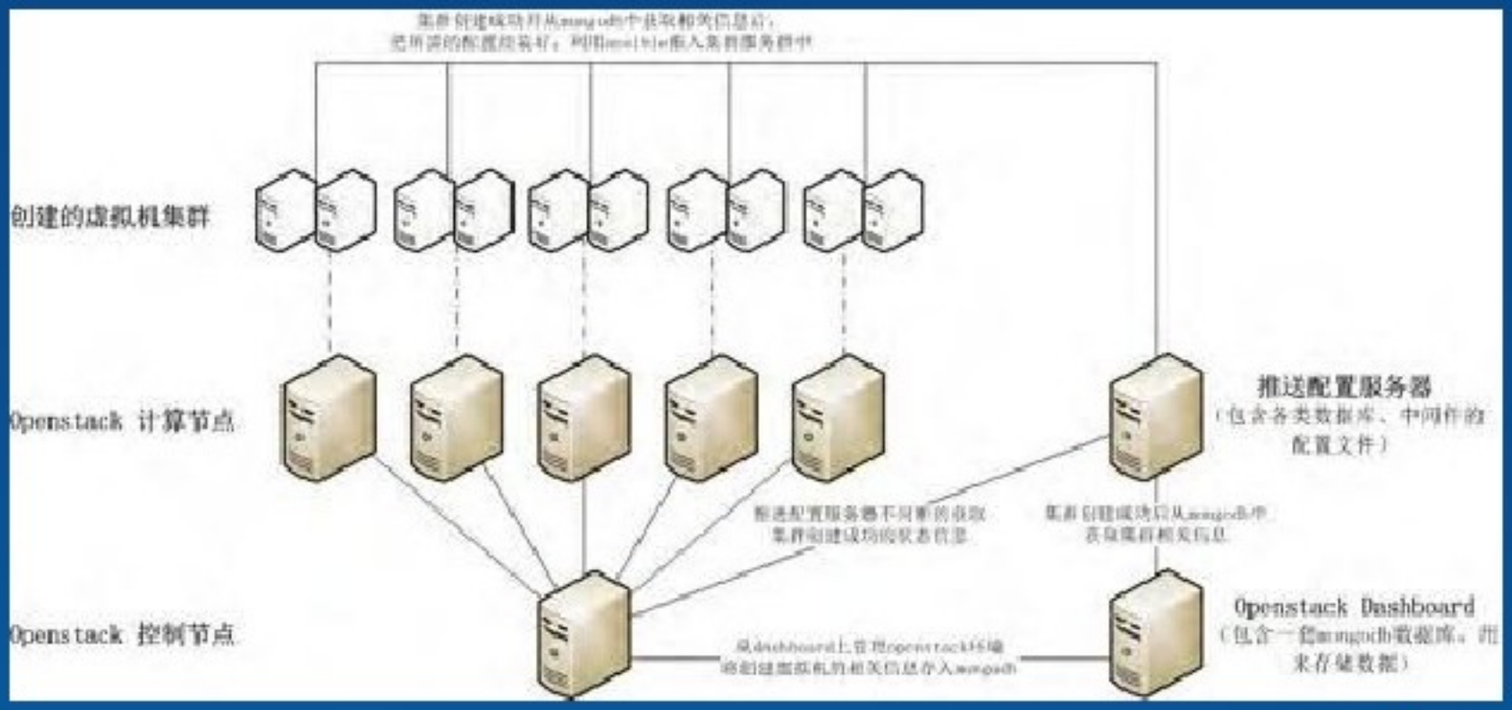
CentOS 6 vs CentOS 7:

在trove项目中，为了安全因素我们希望Mysql实例在Centos6中运行，为了代码兼容以及社区跟进度我们希望Trove在centos7中运行。高版本的Sahara面临同样的问题。OS版本以及OS相关库的冲突成为痛点。

不改变之前任何结构，仅仅将Trove 的相关Agent和Controller。

网络：采用Host模式，Controller直接复用物理机或者VM的网络，不改变网络结构。
启动：Docker 采用autorestart模式，保证agent的自动启动
更新： 通过改变agent Container镜像，简化升级步骤，VM 镜像升级则只有放置Container image不同.
系统：VM、物理机都采用Centos6，Container使用centos 7, 不改变MySQL的运行环境.

“环境”On OpenStack - 自动化拉起

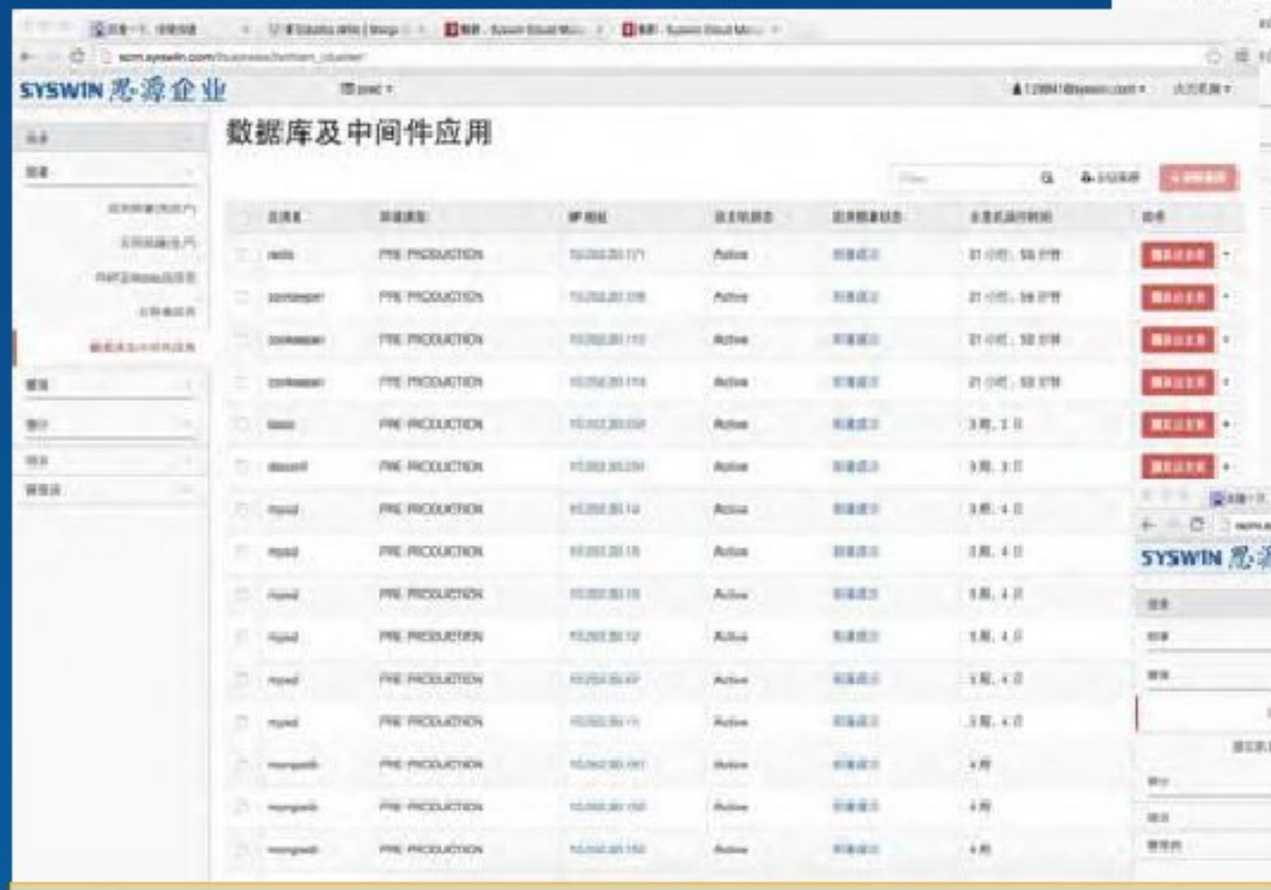


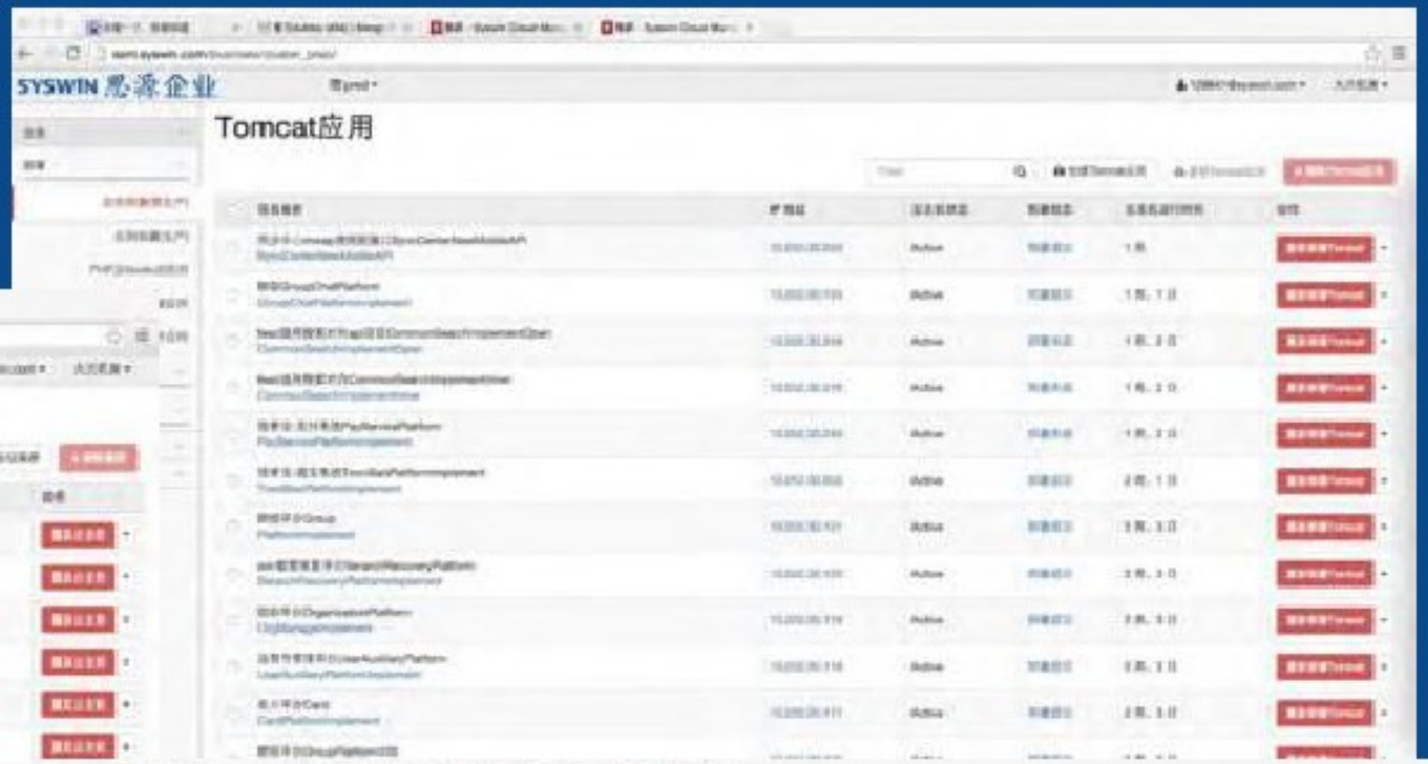
部署项目	部署数量（套）	完成用时
mysql主从集群	3	30分钟
mycat+mysql集群	1	
mha+mysql集群	1	
mongodb集群	1	
zookeeper集群	1	
kafka集群	1	
redis集群	2	
dubbo	1	
disconf	1	
tomcat项目	200	8小时

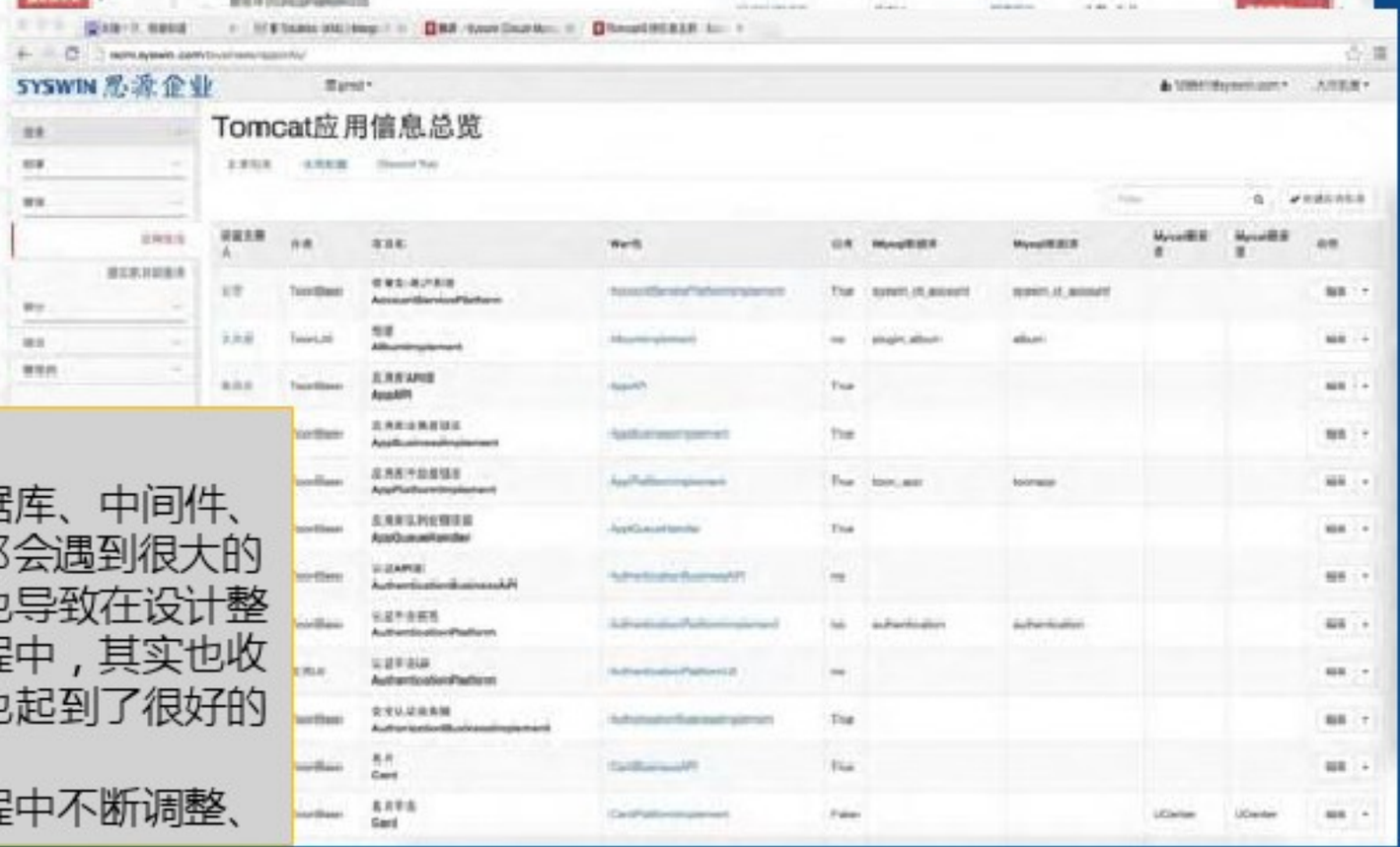
- 实现自动化部署的思想及架构图
  - 要在OpenStack环境中做自动化部署，先决条件就是要创建出虚拟机，并且是一个集群化的多台虚拟机，首先引入OpenStack Heat
  - 虚拟机创建完成后，让程序得知虚拟机已创建完成，并把所需的配置推送到创建出的虚拟机中，推送完成后，重启相应的服务，已到达集群可用状态，基于以上的分析，引入了一台推送配置服务器，并利用ansible作为配置管理工具。
  - 最后，在整个推送配置过程中，所有环节的状态信息，配置产生的输出，日志及错误信息，都要存入到数据库中，并在OpenStack dashboard网站中，全部呈现出来，已方便创建者查看自动化配置的过程及错误解决

- 采用不同配置策略应对多环境需求
- 引入OpenStack Heat实现虚拟机集群创建、管理不同环境的资源配置、及简化虚拟机的创建
- 引入ansible实现配置管理
- 开发应用信息管理界面，形成应用部署核心数据库
- 引入python selenium，实现自动化测试及大规模部署

复杂的自动化过程





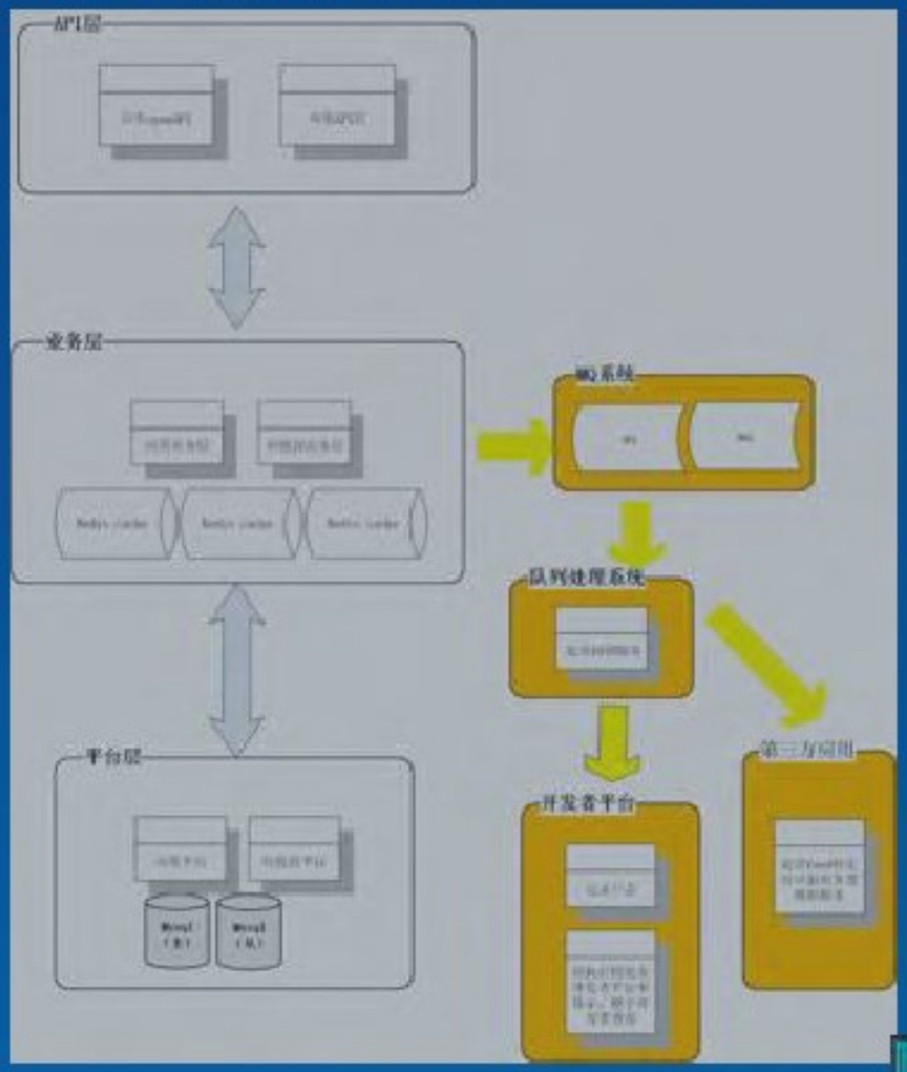


总结：

自动化部署整套环境是一项非常庞大的工程，涉及到从数据库、中间件、到应用的方方面面，有一个环节没有处理好，自动化项目都会遇到很大的挑战。在整个实践中，研发人员对运维工作的认识不足，也导致在设计整个部署流程时，走过一段很长的弯路。但就在走弯路的过程中，其实也收获了大量的经验，加深了很多认识，对后来的调整、改进也起到了很好的作用。

最后，总结起来就是，架构流程先行，代码实现跟进，过程中不断调整、

面向“微服务化” - Kubernetes



平台层:

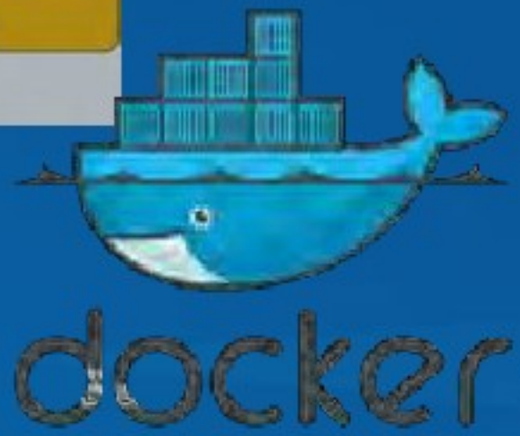
- KVM、Docker、物理机
- 数据库高可用，应用读写分离；尝试Mycat解决分库分表；
- 缓存支持高可用，应用通过Zookeeper注册与发现；
- Kafka集群、Zookeeper集群；
- Dubbo、Disconf
- Tomcat、Apache、NodeJs
- 日志、监控、安全
- …

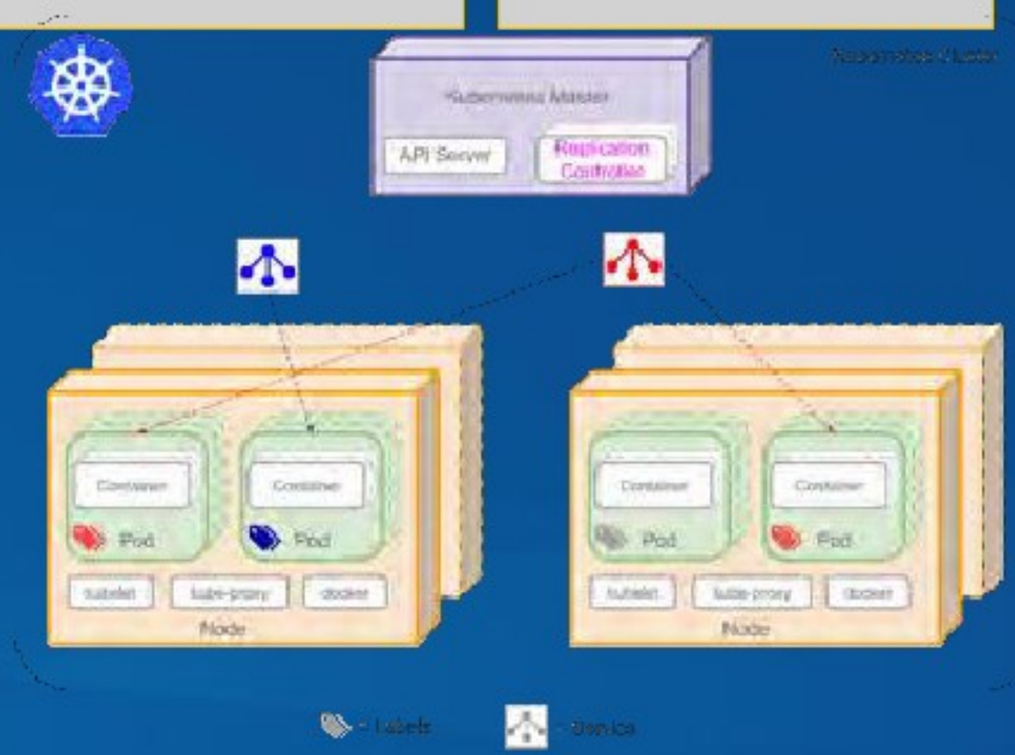
部署规范

- 数据库使用规范
- 缓存使用规范
- 应用配置规范
- 架构规范
- 品质规范
- …

应用架构规范

- 高内聚、低耦合
- 分层、纵向切分
- 明确子系统边界
- Restful
- 微服务化改造
- …





云技术驱动的架构演进与变迁 - 心得

OpenStack 的两个发展阶段：

- 第一阶段 OpenStack as IaaS
  - OpenStack 作为 IaaS 支撑已经很成熟了
- 第二阶段 Solutions on OpenStack
  - 应用架构的多样性与异构是客观存在的现实
  - 应用架构的治理至关重要
  - 应用架构治理需要云技术团队与业务团队紧密合作
  - 应用架构治理需要一个过程
    - 基于Kubernetes(Mesos)的微服务架构
    - “Environment on Demand”，复杂架构很难段时间内符合“微服务”精神
    - 基于Ansible/Heat方式的业务 / 服务部署
    - …
- 外围系统
  - 持久化和共享的中间件仍是难点
  - 运维支撑系统的跟进：日志分析系统、监控系统、安全、自研的其他运维服务平台
  - …

Thank you