JDOS
容器集群实践

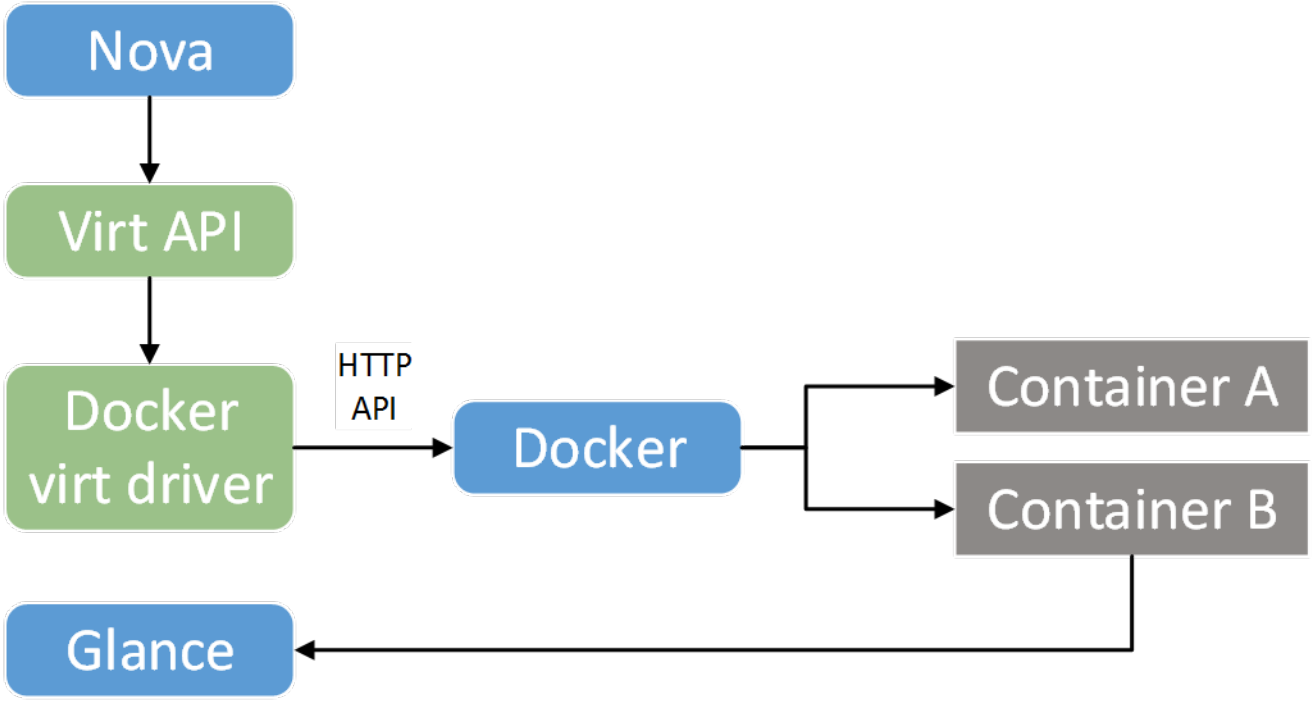
徐新坤

JDOS 1.0

当时的状况

- 应用直接部署在物理机上
- 应用上线从申请资源到最终分配物理机时间平均为一周
- 物理机失效导致的应用实例迁移时间以小时计
- 物理机资源浪费严重
- 多个配套工具系统

openstack


Nova
Virt API
Docker
virt driver
HTTP
API
Docker
Container A
Container B
Glance

成果

- 物理资源池化
- 申请计算资源由之前的一周缩短到分钟级
- 部署应用密度显著提升
- 业务应用的容器化
- 网络、容器等基础技术磨合成熟
- 大规模容器的运维经验
- 监控、巡检等周边系统完善

经验与教训

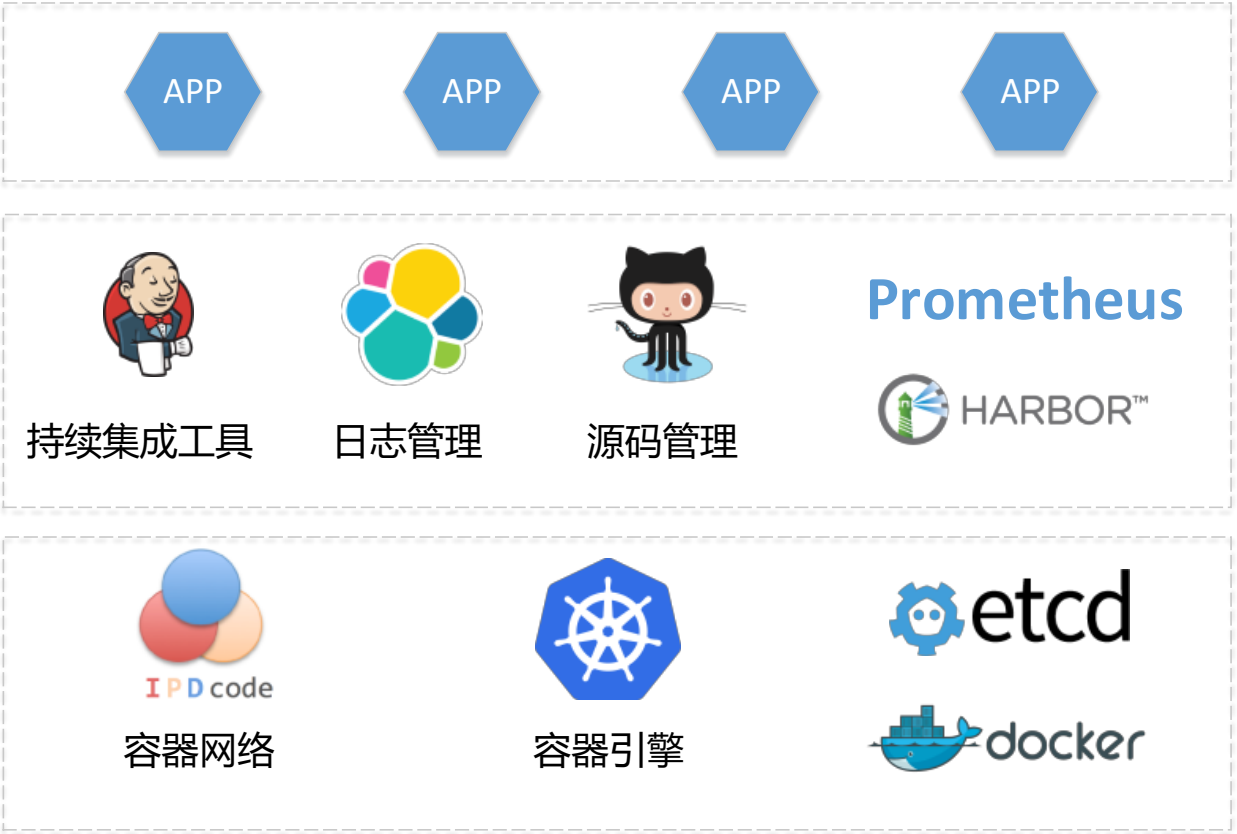
- 容器与虚拟机之辩
- load/top值
- Dm带来的问题
- Jvm的GC延迟问题
- Cpu使用的问题(cpuset与cpushare)

JDOS 2.0

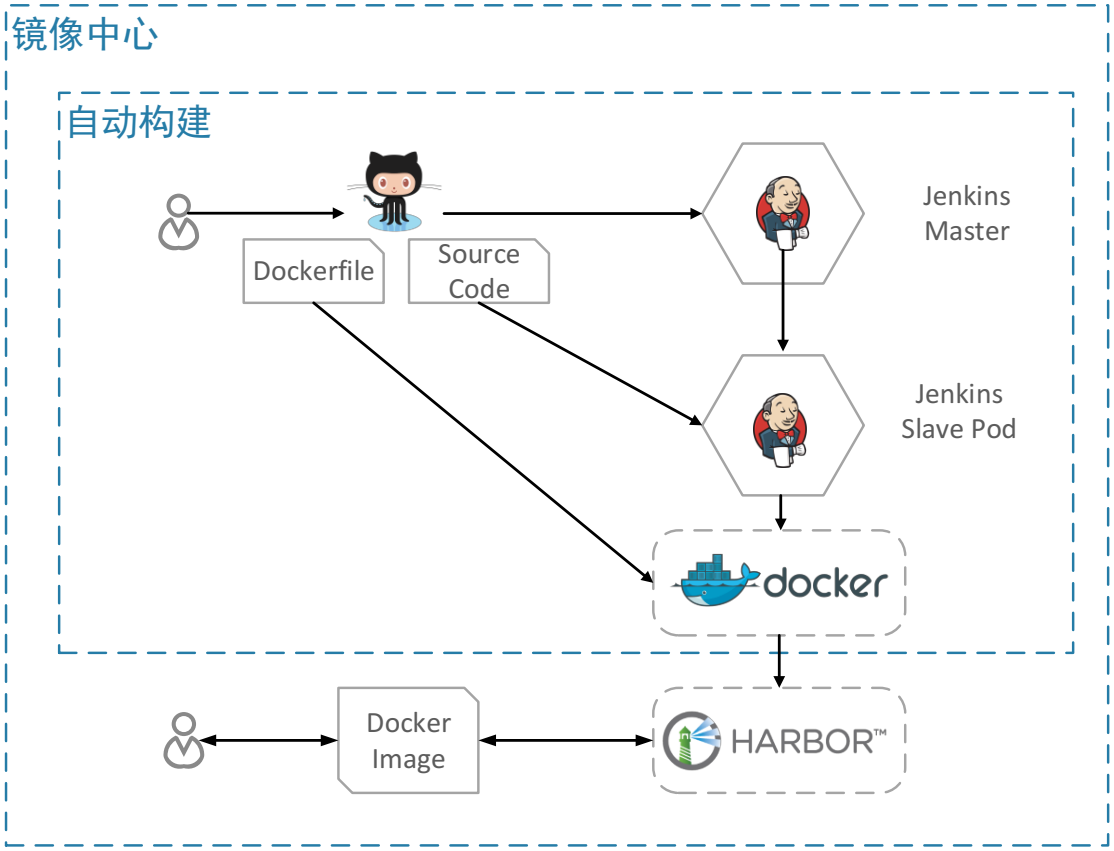
1.0的痛点

- 重IAAS思维
- 容器“开箱后仍不可用”
- 无法达到镜像的“一次构建，随处运行”的理想状态
- 弹性与高可用仍依赖于业务和传统工具
- 调度方式单一

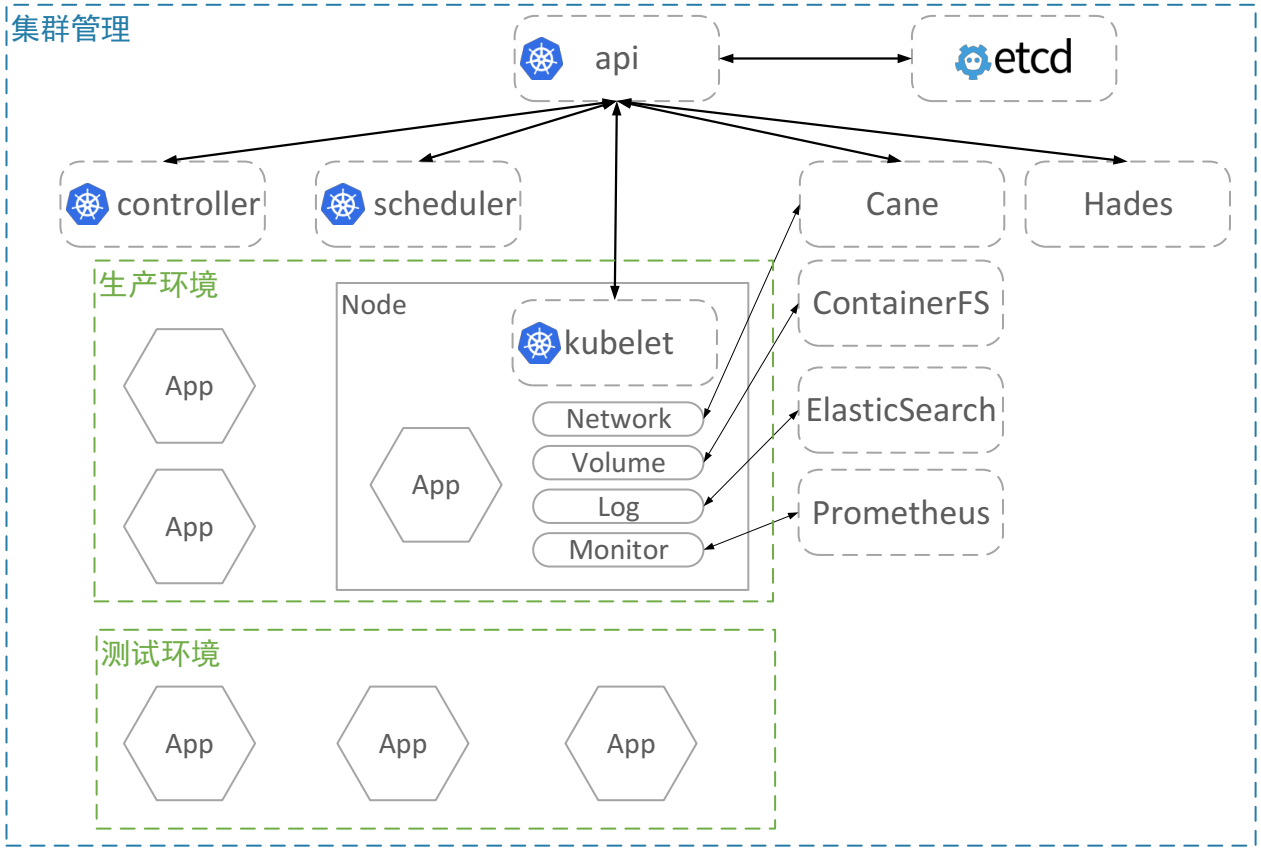
架构


APP
APP
APP
APP
持续集成工具
日志管理
源码管理
Prometheus
HARBOR™
IPD code
容器网络
容器引擎
etcd
docker

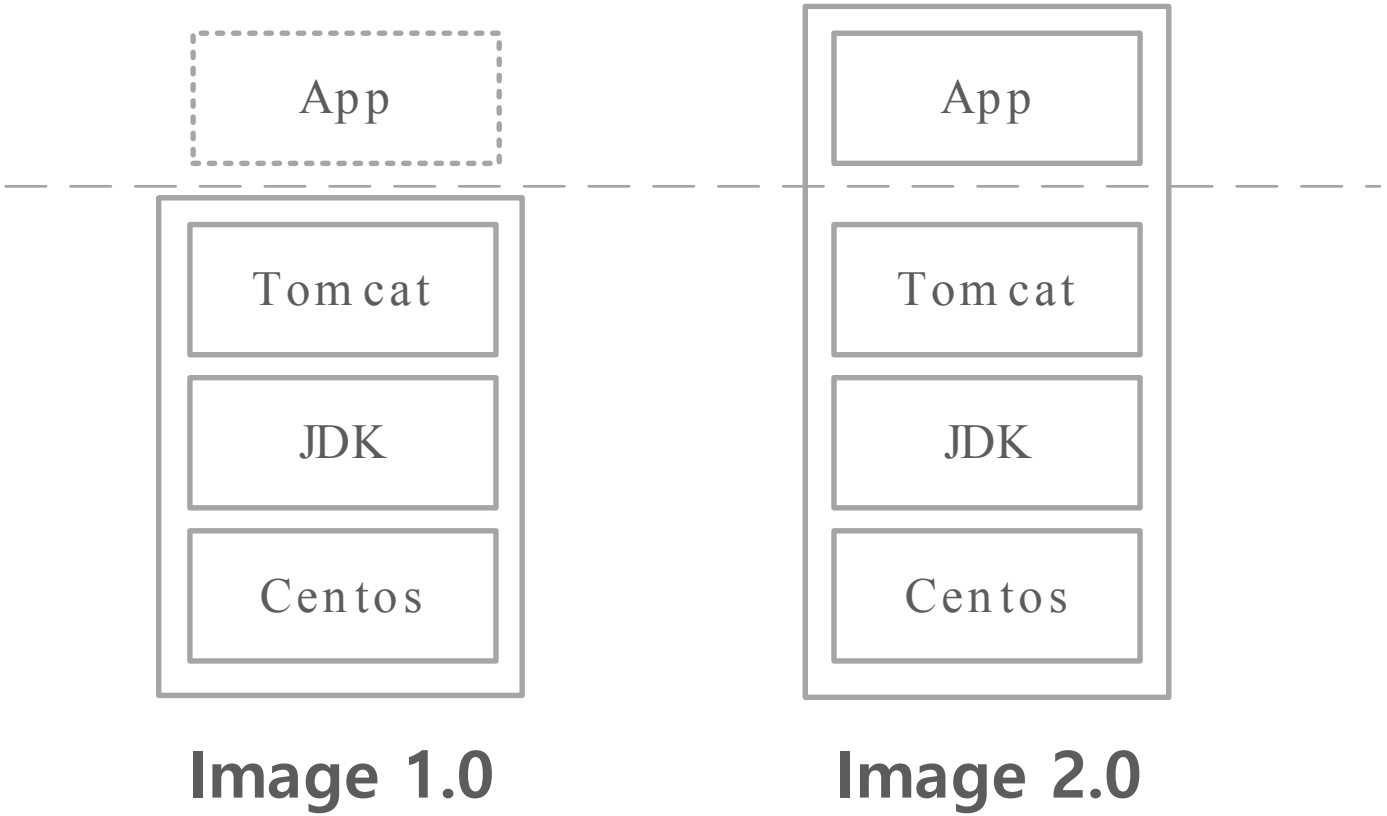
镜像中心与CICD


镜像中心
自动构建
Dockerfile
Source Code
Jenkins Master
Jenkins Slave Pod
docker
Docker Image
HARBOR™

集群管理


集群管理
api
etcd
controller
scheduler
Cane
Hades
生产环境
Node
kubelet
App
App
App
Network
Volume
Log
Monitor
ContainerFS
ElasticSearch
Prometheus
测试环境
App
App
App

镜像


App
Tomcat
JDK
Centos
Image 1.0
App
Tomcat
JDK
Centos
Image 2.0

JDOS 2.0的产品

系统

- 全局概念，不分机房和区域
- 包含若干个相关的应用
- 一般一个部门有一个或者多个系统
- 系统内的所有容器可以互通，不做隔离；系统间可以根据需要进行隔离
- 资源统计和限制的主要层级
- 概念对应
  - harbor中镜像仓库(repository)
  - Kubernetes中的namespace

应用

- 全局概念，不分机房和区域
- 包含提供相同或者相近服务的多个分组
- 提供编译、构建和负载均衡服务
- 概念对应
  - Harbor中的镜像

分组

- 具体到某一机房或区域
- 在某个k8s集群内提供相同服务的若干个Pod
- 每个Pod都是对等可置换的
- 分组内
- 概念对应
  - Kubernetes中的replica set或者多个pod

应用的负载均衡与域名服务

- 应用的负载均衡可以包含一个或者多个分组
- 一个负载均衡由若干个haproxy/nginx的Pod容器实现
- 域名服务会解析到负载均衡的haproxy/nginx的容器上
- 单一分组或者容器均可以独立从负载均衡中摘除
- 负载均衡支持http协议、负载均衡算法等参数配置
- 概念对应
  - Kubernetes中的service(使用svc而不使用ingress进行对应)

Node生命周期管理

- Status
  - 初始化node.status=init
  - 节点就绪node.status=wait
  - 上线状态node.status=online
  - 离线状态node.status=offline

调度管理

- Zone
  - 默认区域zone=default
  - 持续集成区域zone=cicd
  - zone=xxx
- Resource
  - res.ssd=true
  - res.gpu=true
  - res.normal=true
- 标签的设计
  - 相交与互斥

总结

Kubernetes的优势

- 架构简洁
- 运营成本低
- 核心理念(对于资源，任务的理解)
- 灵活的设计(标签)
- 声明式的api

谢谢！