QCon[上海站]
全球软件开发大会2016

企业容器云平台实践

构建以应用为中心的企业云平台

SPEAKER

刘　相

International Software
Development Conference

主办方 Geekbang. 极客邦科技 InfoQ

自我介绍

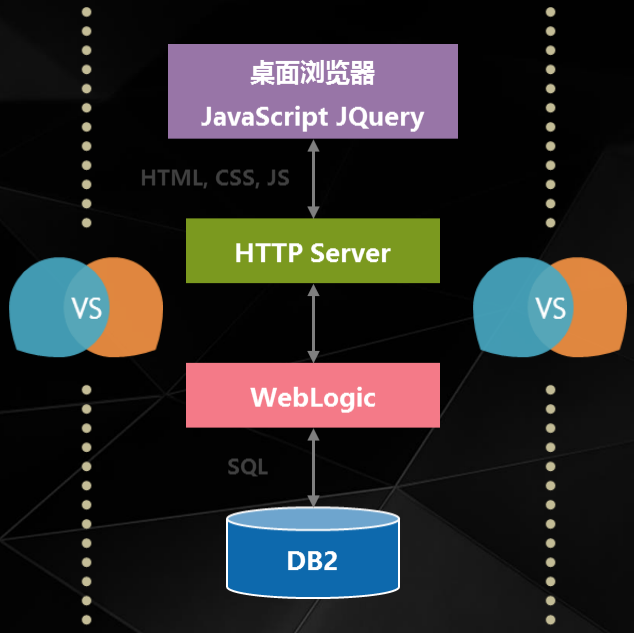
- 刘相
- 10年•专注于企业应用架构
- 普元•加速企业数字化转型•SOA、大数据、云计算

大纲

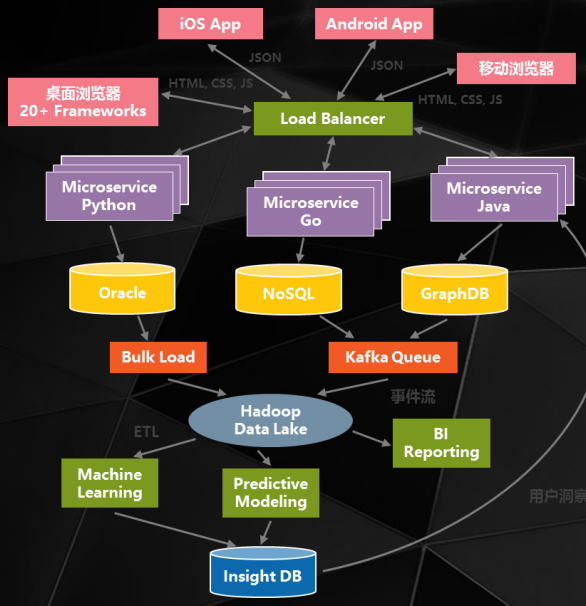
- 企业对容器的认知
- 企业云平台实践
- 关键设计与思考

企业应用架构日益复杂化

2010

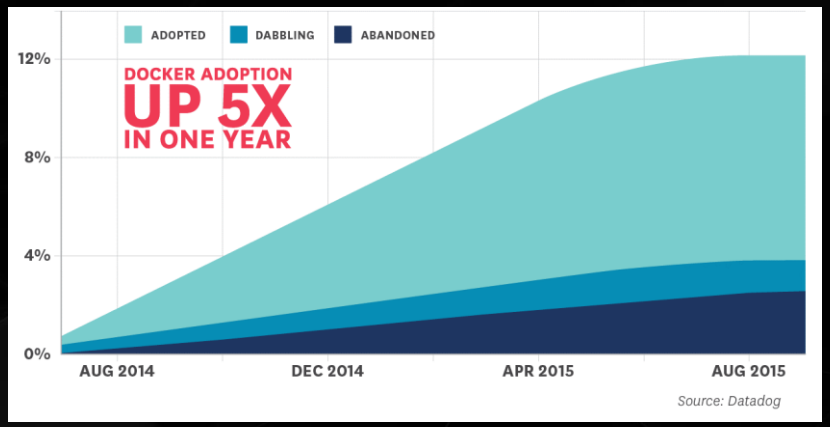


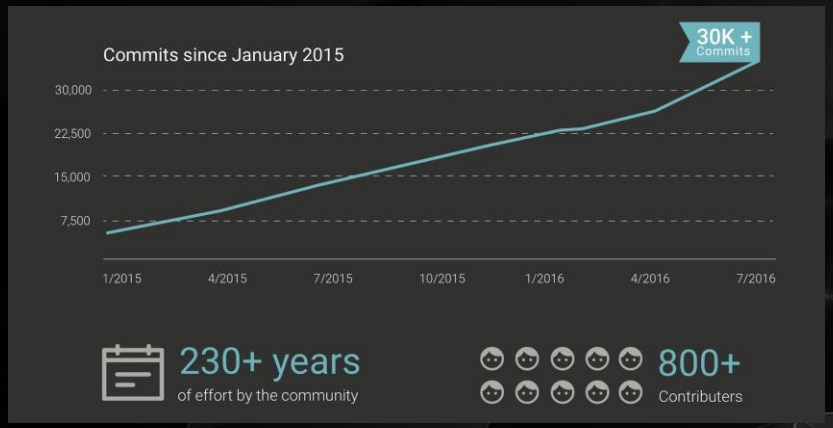
2013



2016

容器技术2015、2016年的双子星





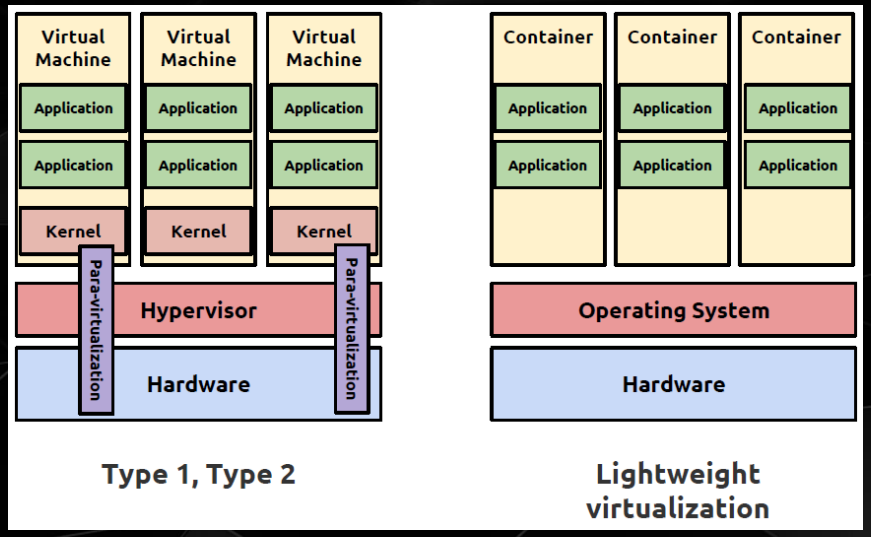
Docker

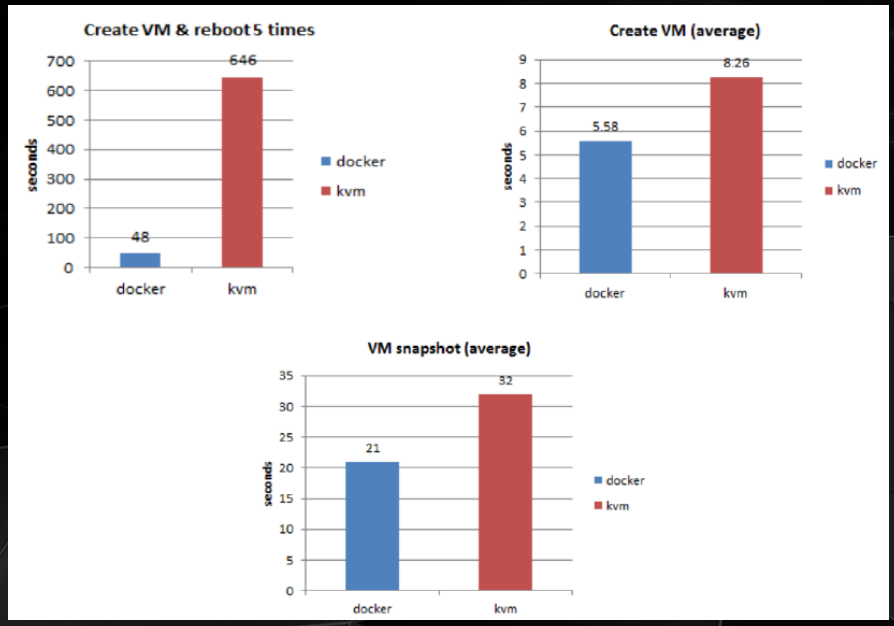
发展最快的开源项目

kubernetes

最具颠覆性的开源项目

容器只是轻量的虚拟机吗？



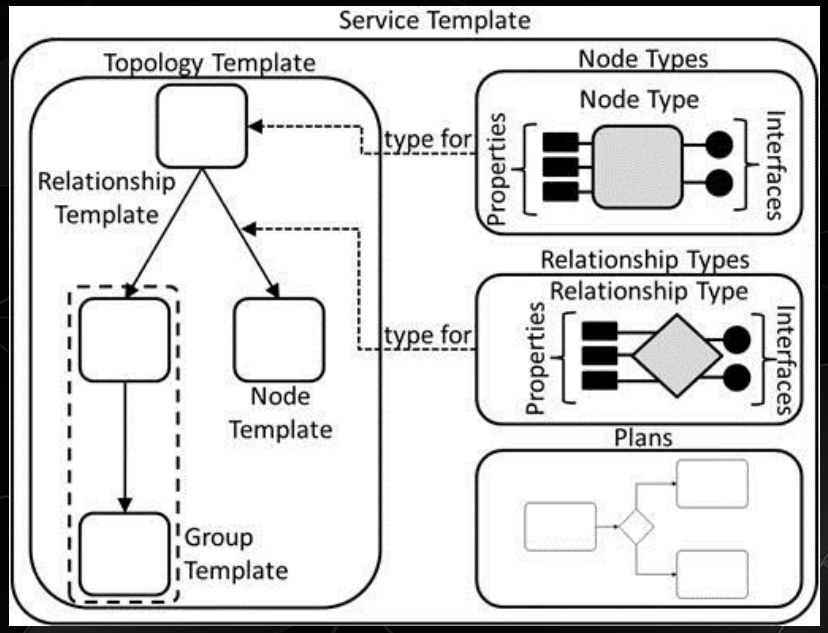


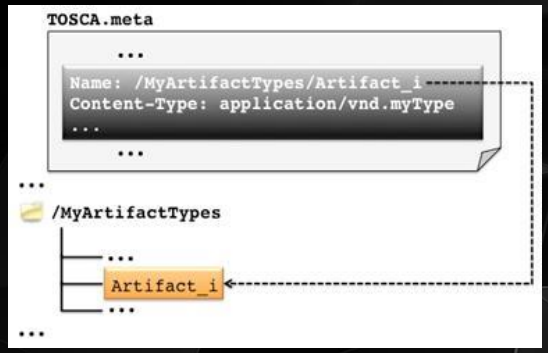
Source: Container and Lightweight virtualizatio

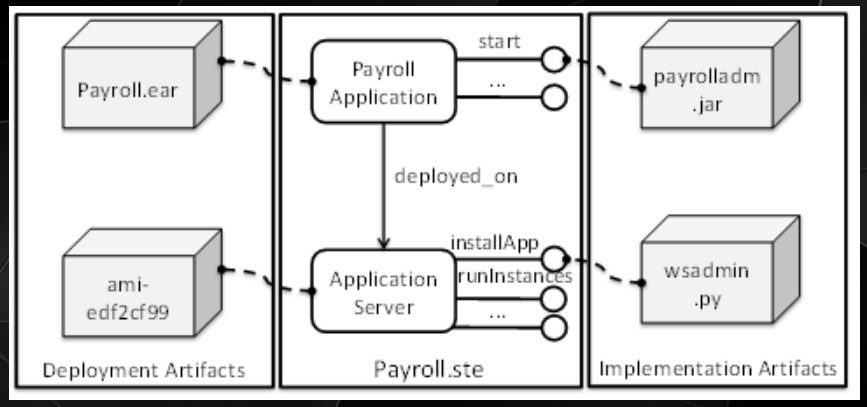
基础设施提升抽象层次的挣扎

TOSCA: Topology and Orchestration Specification for Cloud Applications

OASIS TOSCA





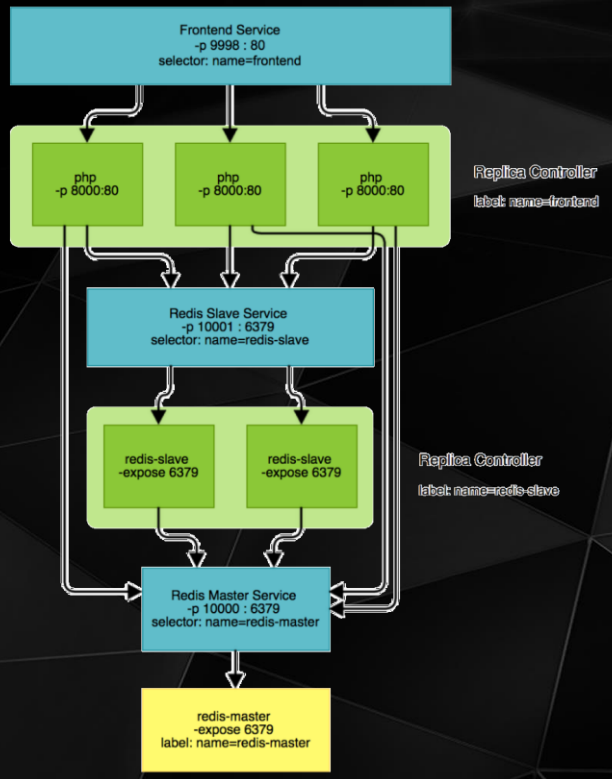


Source: Container and Lightweight virtualizatio

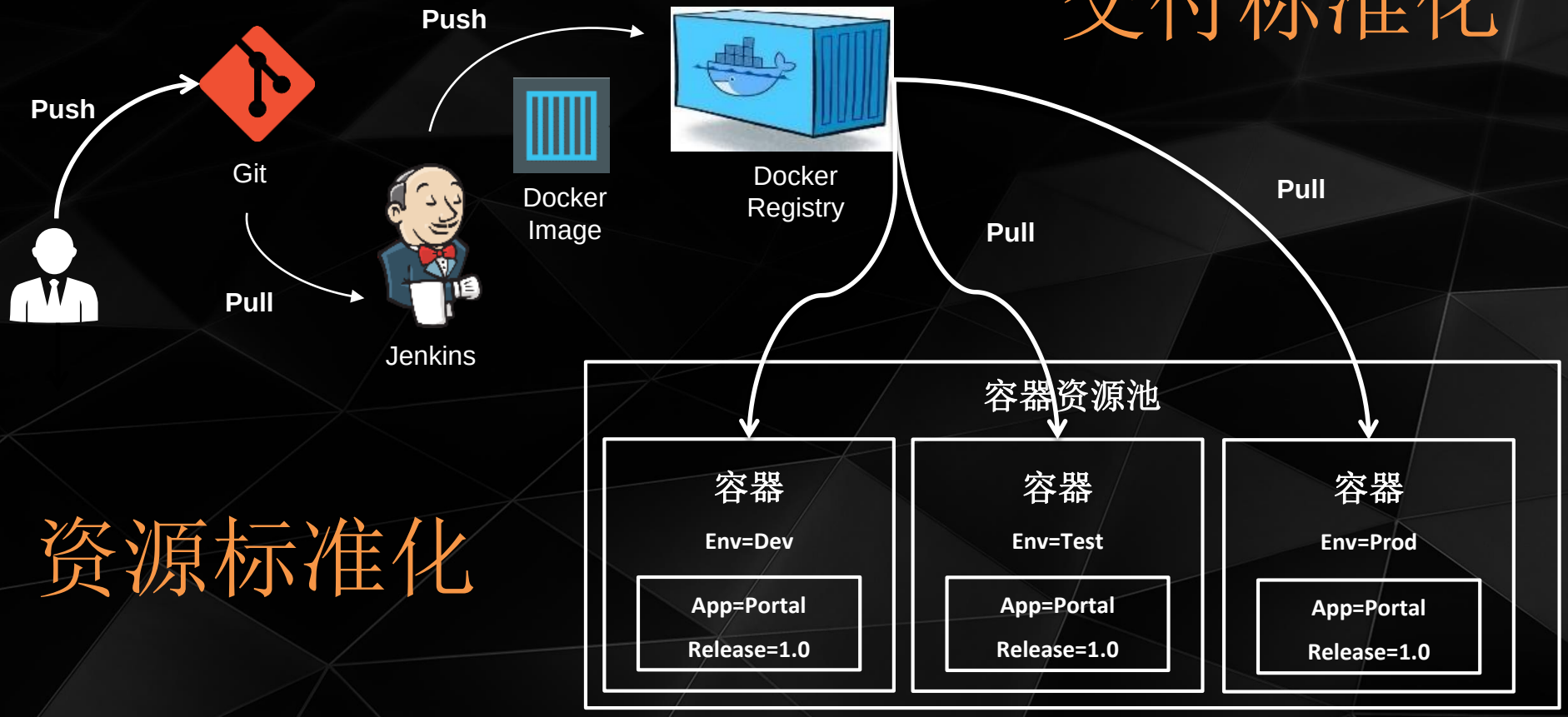
容器的一体两面？

是“轻量的IT基础设施”，还是“应用的发布和管理工具”？

VS

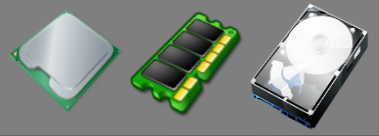


容器带来了应用新时代


交付标准化
Push
Push
Git
Docker Image
Docker Registry
Pull
Pull
Pull
Jenkins
资源标准化
容器资源池
容器
Env=Dev
App=Portal
Release=1.0
容器
Env=Test
App=Portal
Release=1.0
容器
Env=Prod
App=Portal
Release=1.0

容器最大的价值提升应用抽象层级

缺少：数据、配置、监控、状态、日志、安全、流控……

Layer 7	应用	
Layer 6	容器	Rkt, Docker
Layer 5	编排	Kubernetes, Mesos, fleet
Layer 4	集群资源	flannel, weave, locksmith
Layer 3	集群协调	etcd, Zookeeper, Consul
Layer 2	OS	Kernel, Systemd, cgroups, jails, zones
Layer 1	设备/虚拟设备	

容器技术栈
应用的带内管理

虚拟化技术栈
设备的带外管理

企业在拥抱容器中思考的问题

100% 已有历史系统的迁移与演进

70% 实施需要多少成本、多少时间

90% 容器能给企业带来什么样的提升

50% 安全、自主掌控、有前瞻性

与互联网异构的企业现状

N套
环境

开发环境
测试环境
实验室
预发环境
生产环境
……

异构
环境

容器
虚拟机
物理机
……

权限与
隔离

应用组
数据组
系统组
监控组
……

个性化
流程

服务申请
发布管理
变更管理
配置管理
监控管理
……

不要期望所有应用运行在容器

无状态应用

Heroku 12 factor

网络高吞吐

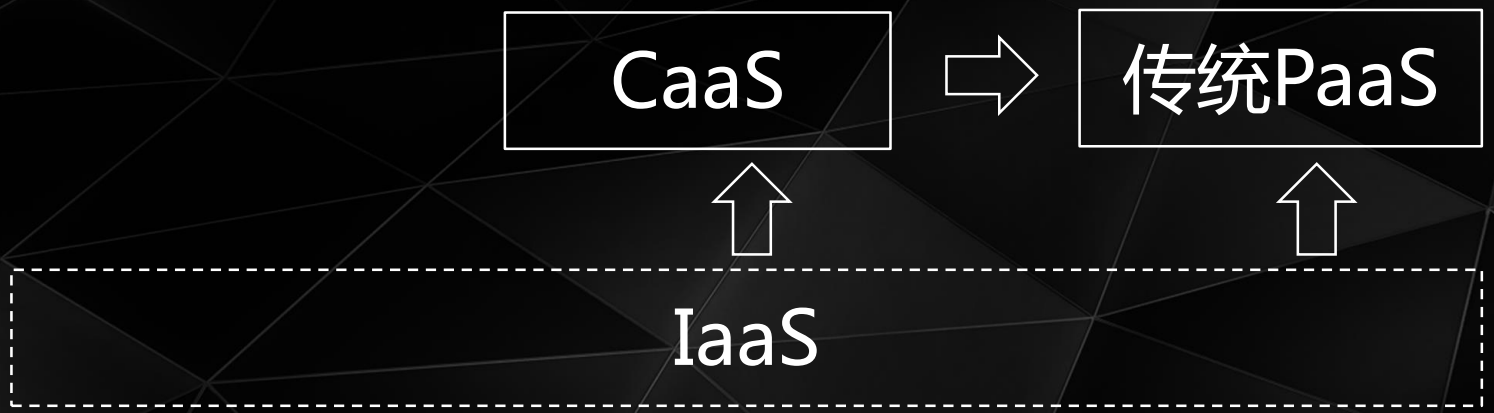
频繁动态调整CPU、内存

重型基础设施：数据库 ...

避免以资源为中心建设企业云平台

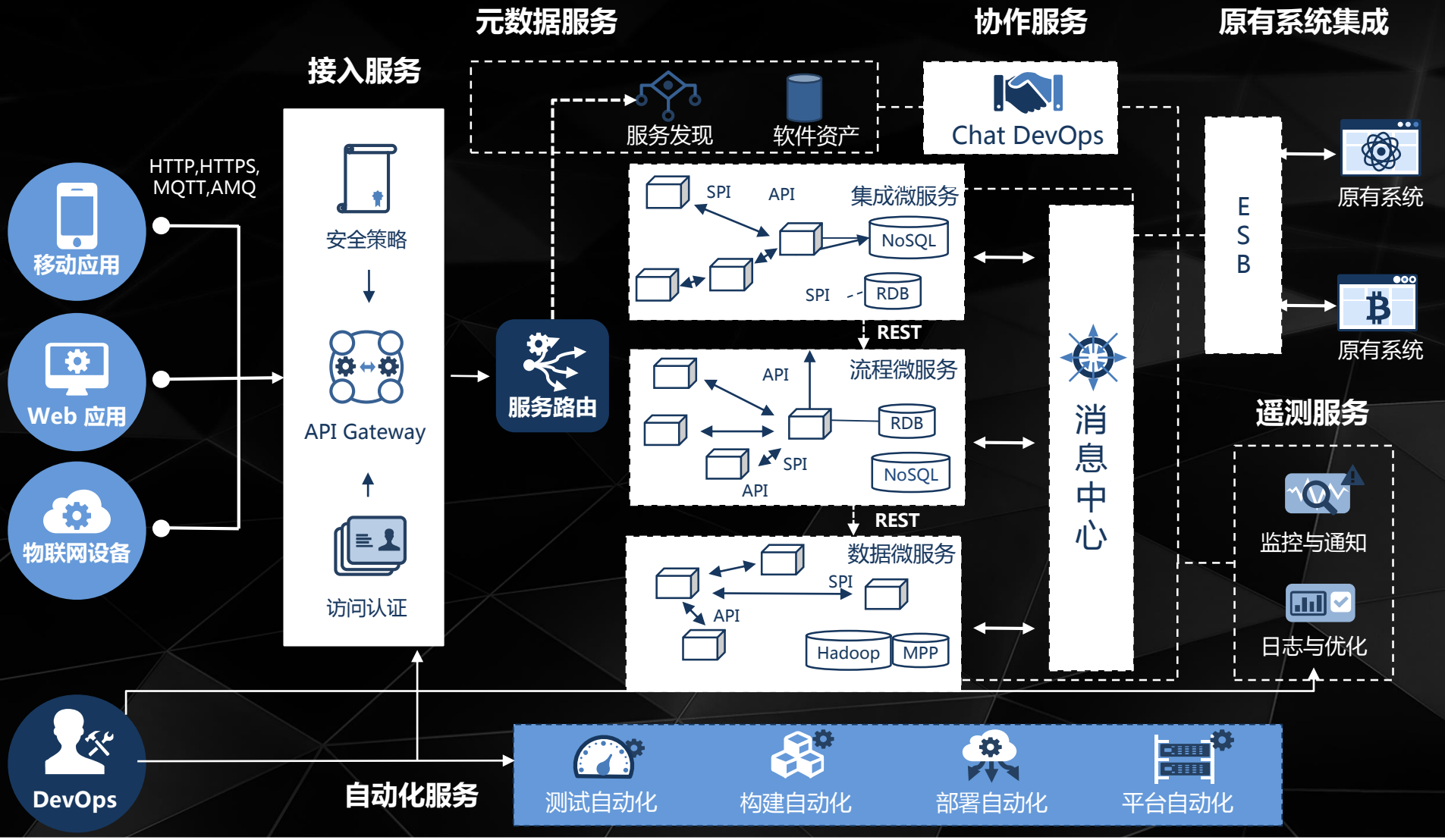
- 主机管理
- 代码构建
- 镜像仓库
- 应用部署（软件部署）
- 安全、权限
- …

互联网上的一些常见能力是这样的

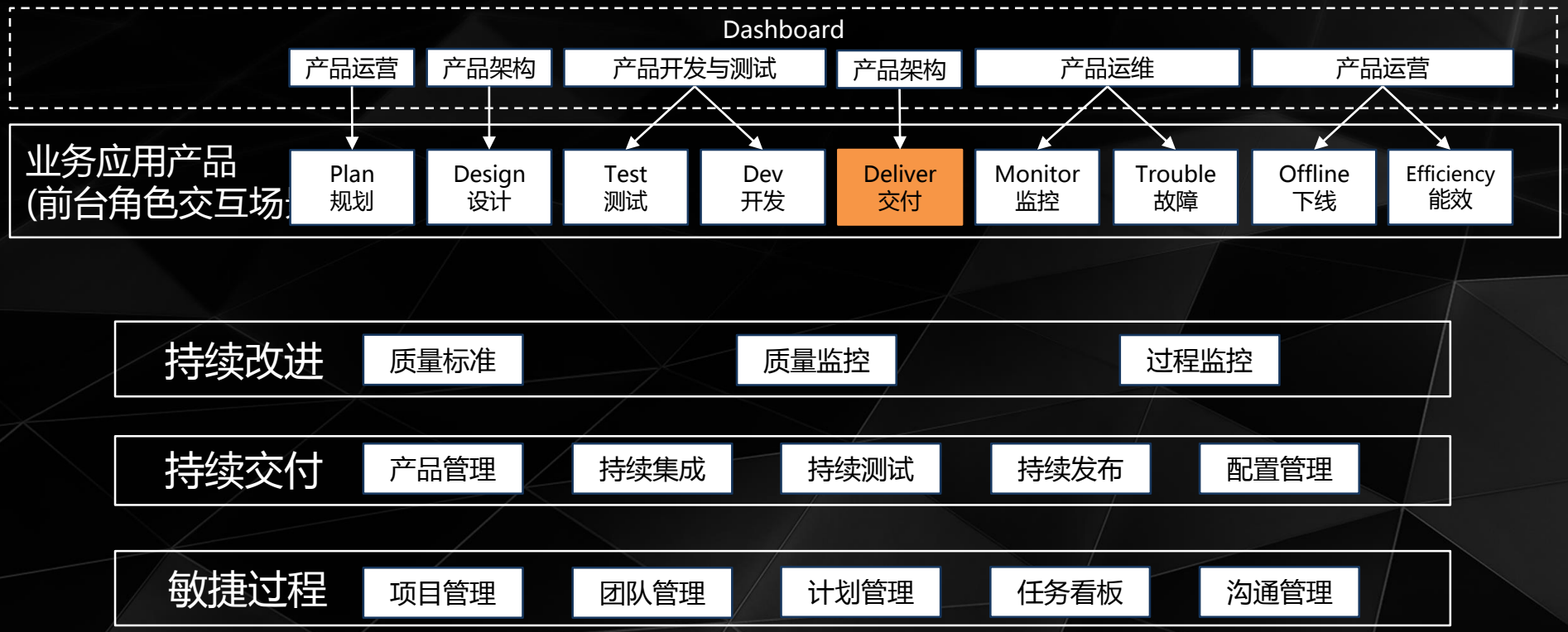


以应用为中心构建企业容器云平台

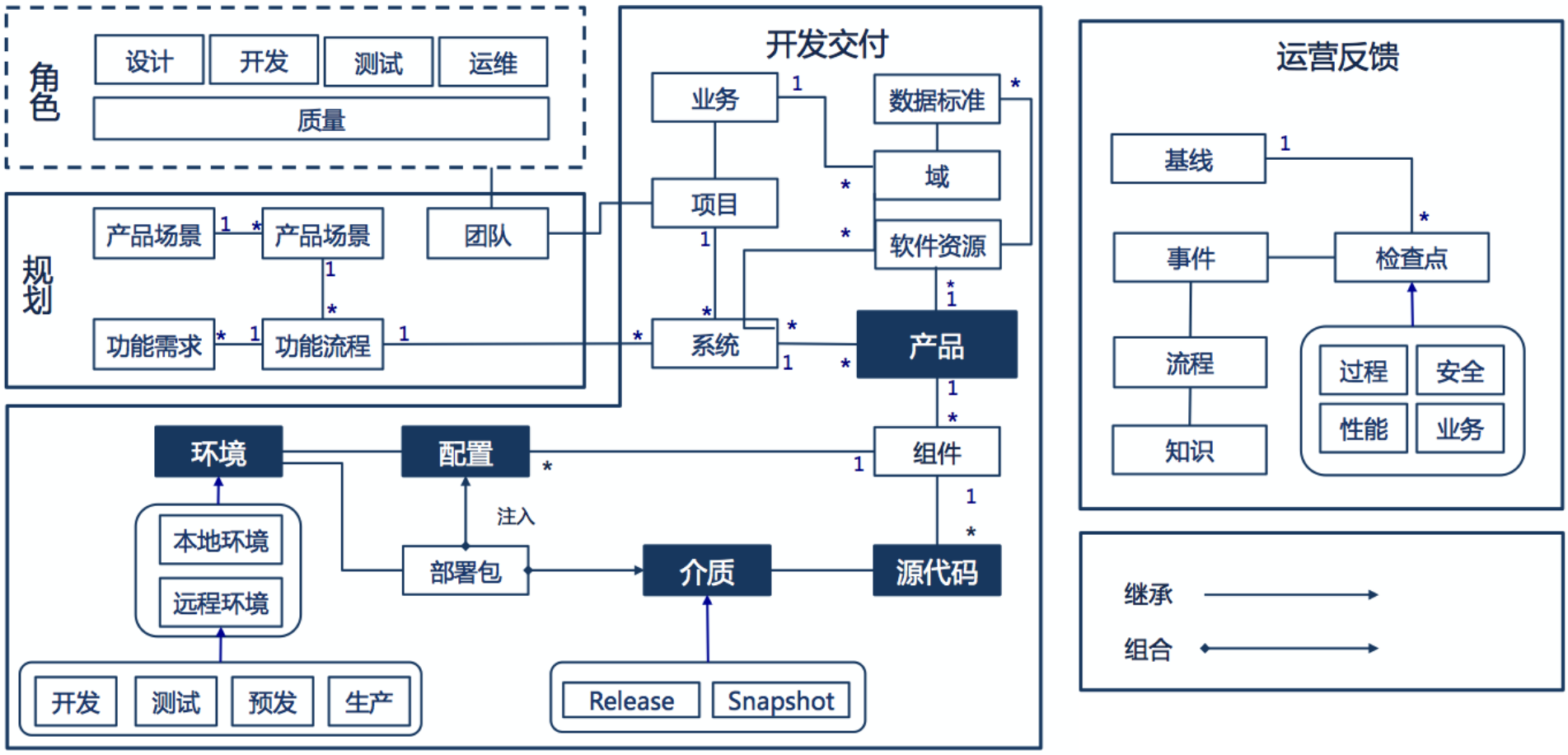
企业级容器云平台架构图



容器支撑的DevOps搭建IT生产线



DevOps概念模型，实现代码、配置、环境分离



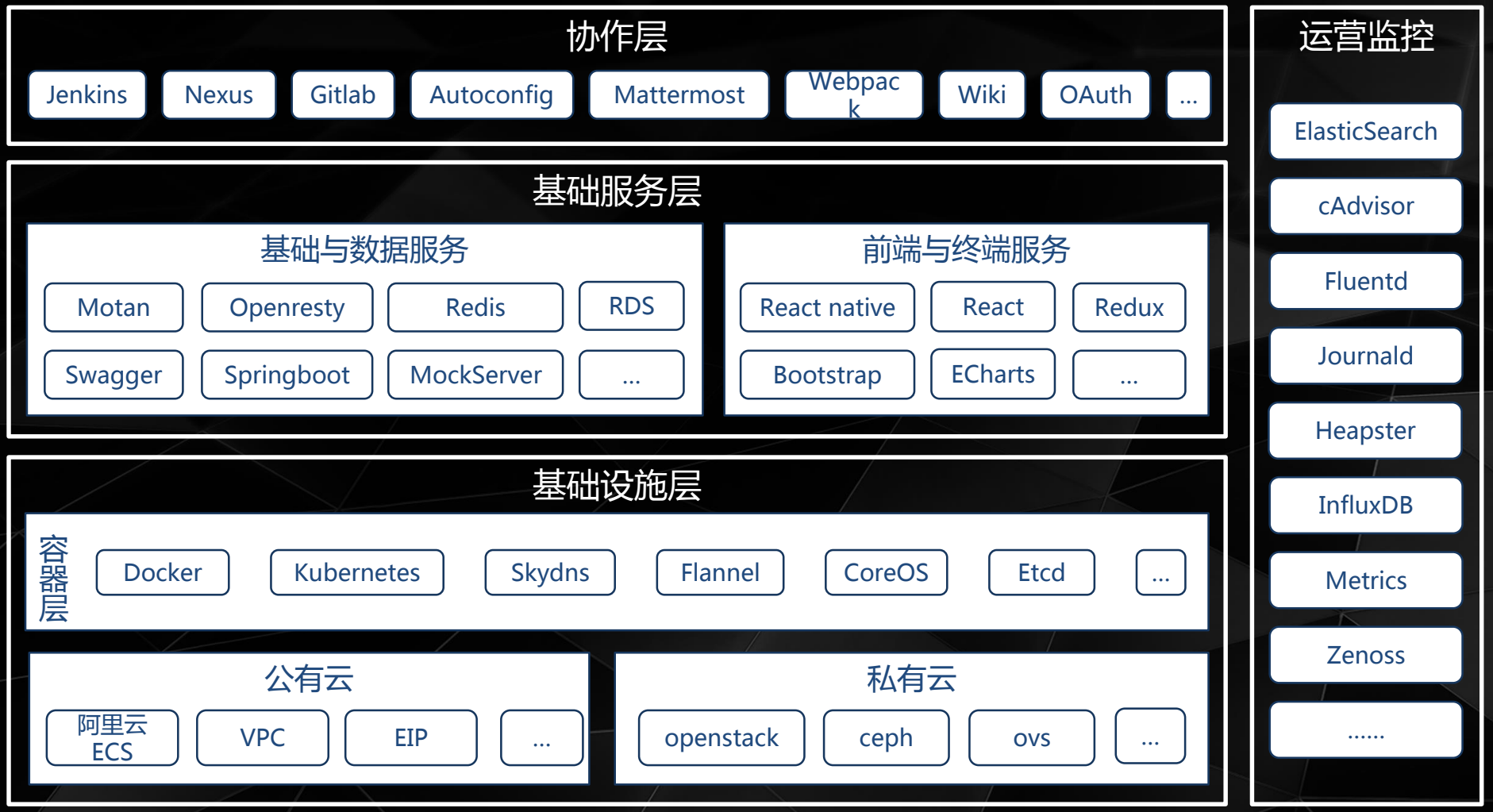
全面降低集成与交付的协作成本

SEM 环境 | IAM 身份 | SPM 产品 | SRM 资源 | PM 项目 | CI 编译 | BPR 介仓 | DPR 部仓

支撑微服务的应用平台


开发&集成
微服务
开发设计器
代码仓库
介质仓库
部署包仓库
持续集成交付
服务挡板
自动测试平台
上下文状态保持
数据引用 / 同步
服务编排
微服务容器
服务
组件
认证鉴权
分布式
事务
元数据驱
动
监控 | 日志 · 性
能 · 业务
非阻塞异步通信
支撑微服务的平台服务
微服务监控治理
容器状态管理
服务注册中心
运行时
配置管理
统一日志
收集分析展现
资源监控
容器/传统环境

技术栈，难得产生共鸣


协作层
Jenkins
Nexus
Gitlab
Autoconfig
Mattermost
Webpack
Wiki
OAuth
...
基础服务层
基础与数据服务
Motan
Openresty
Redis
RDS
Swagger
Springboot
MockServer
...
前端与终端服务
React native
React
Redux
Bootstrap
ECharts
...
基础设施层
容器层
Docker
Kubernetes
Skydns
Flannel
CoreOS
Etcd
...
公有云
阿里云ECS
VPC
EIP
...
私有云
openstack
ceph
ovs
...
运营监控
ElasticSearch
cAdvisor
Fluentd
Journald
Heapster
InfluxDB
Metrics
Zenoss
.....

选择开源的四项基本原则

- 功能特性
- 技术架构
- 开发语言
- Roadmap

- License
- 开发模式
- 测试模式
- 决策模式

- 产业经验
- 自己有没有规模化
- 需求来源

- 提供者的产业链位置
- 与友商的关系
- 众望所归/单打独斗？

不要只看单一维度的功能

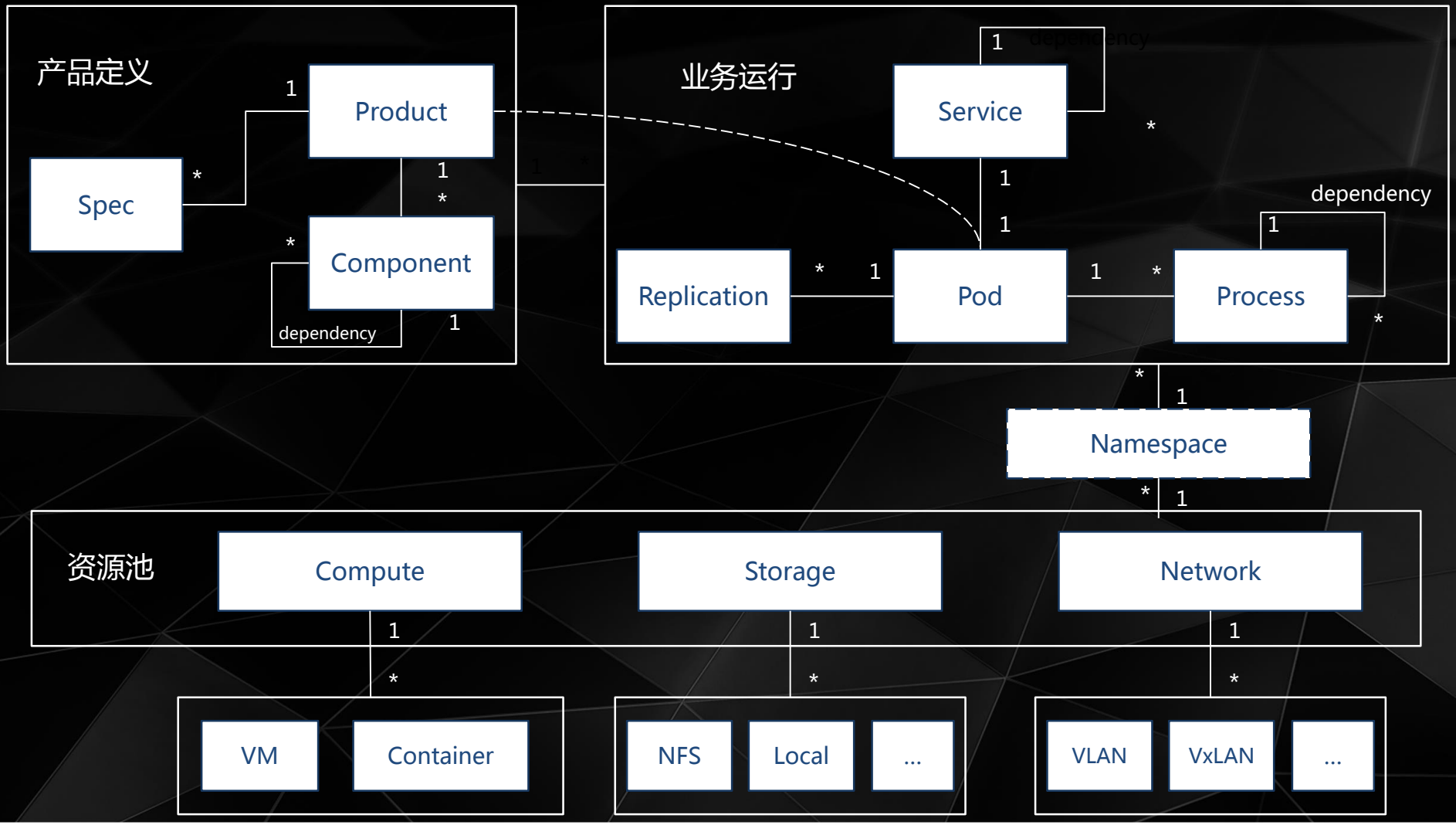
确保你的团队能cover住

生产者是否是使用者

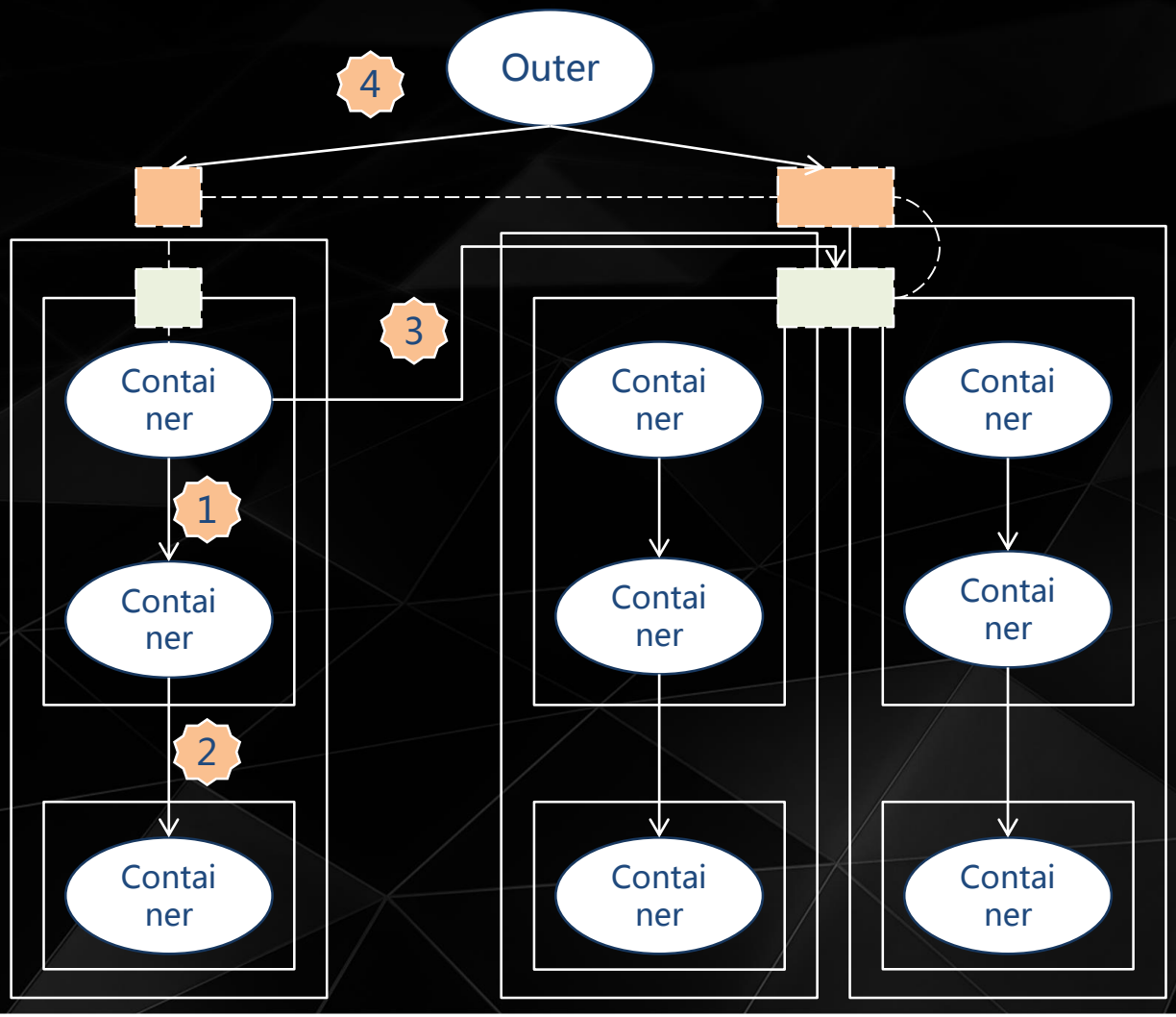
建立开源基线平台

……

关键设计：异构资源一体化模型



关键设计：容器资源的隔离与互通



1. 同一Pod里的所有容器拥有同样的网络空间，可当本地（127.0.0.1）访问
2. Pod之间的网络采用内部IP的方式调用，每个Pod拥有自己的IP
3. Pod可暴露成service，每个service有自己的IP（虚IP），下层是Pod集群，可通过service进行集群调用
4. 通过宿主机的端口映射，解决外部访问内部的问题

关键设计：提供应用最佳实践模式

后端服务组件(standalone jar)

```
/
|--- app
|    |--- conf
|    |    |--- autoconfig.properties
|    |    |--- application.yaml
|    |--- bin
|    |    |--- entrypoint.sh
|    |    |--- autoconfig.sh
|    |--- lib
|    |    |--- application.jar
|    |--- data
|--- jdk
```

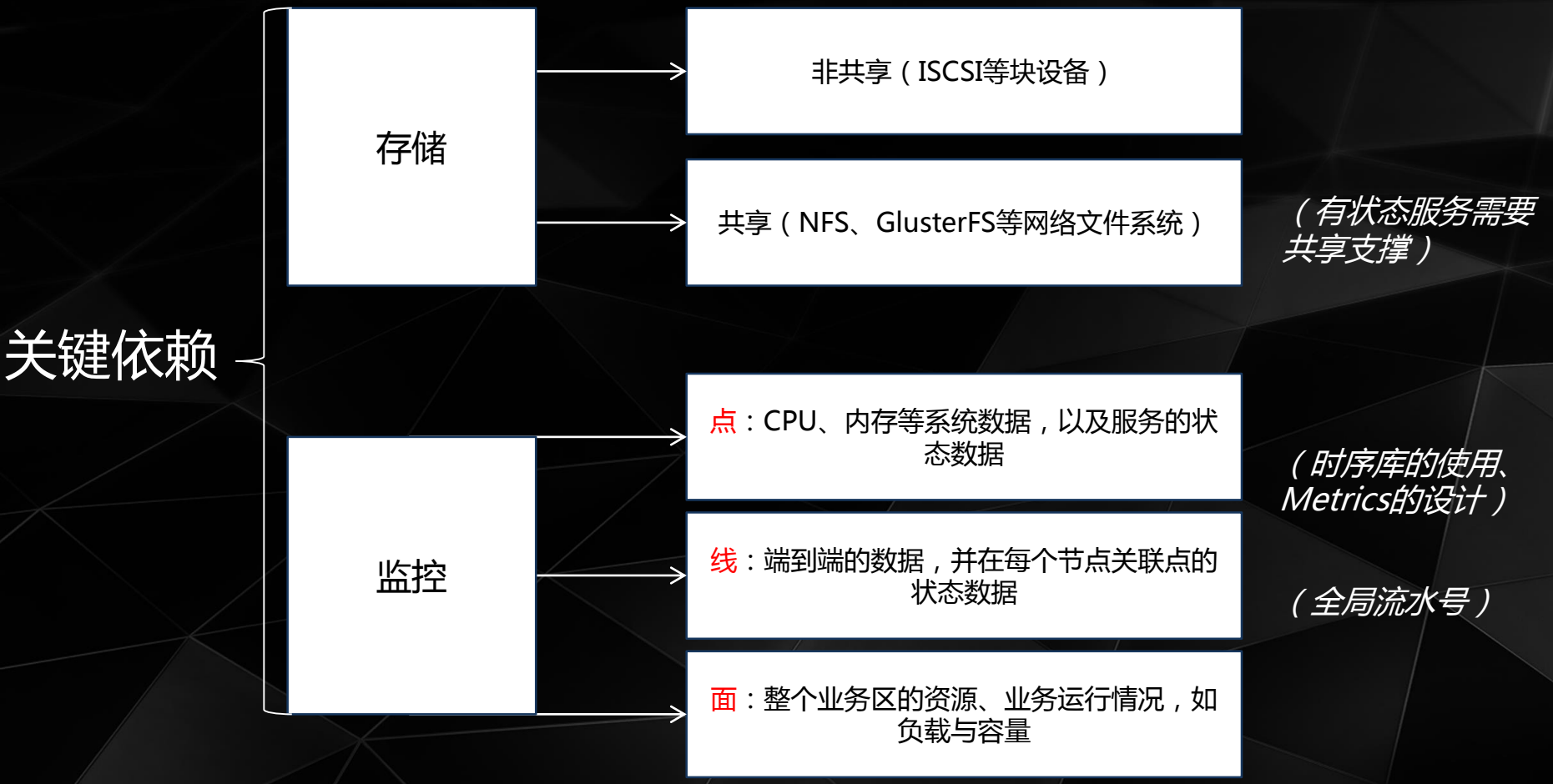
前端服务组件(html)

```
/
|--- app
|    |--- conf
|    |    |--- autoconfig.properties
|    |    |--- nginx.conf
|    |    |--- conf.d
|    |         |--- *.conf
|    |--- bin
|    |    |--- entrypoint.sh
|    |    |--- autoconfig.sh
|    |--- html
|    |    |--- *.html
|    |    |--- images
|    |         |--- *.png
|    |--- data
|--- usr
     |--- lib
          |--- nginx
```

JEE组件

```
/
|--- app (tomcat)
|    |--- conf
|    |    |--- autoconfig.properties
|    |    |--- *.xml
|    |--- bin
|    |    |--- entrypoint.sh
|    |    |--- autoconfig.sh
|    |--- webapps
|    |    |--- ROOT
|    |         |--- WEB-INF
|    |--- data
|--- jdk
```

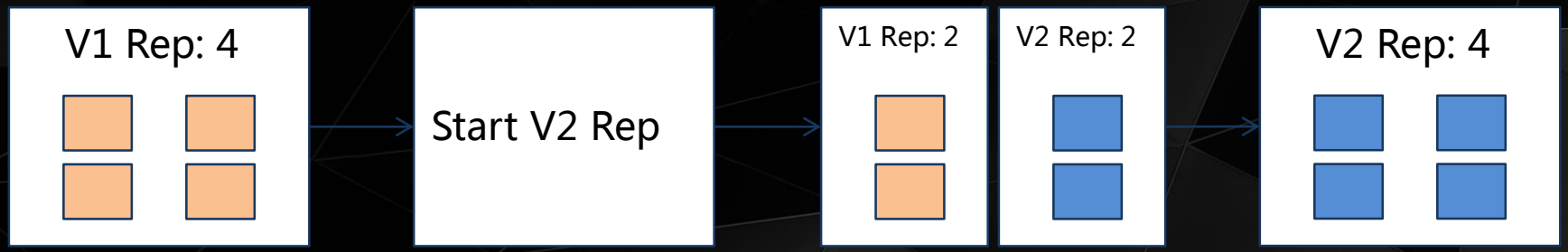
关键设计：伸缩与漂移


关键依赖
存储
非共享（ISCSI等块设备）
共享（NFS、GlusterFS等网络文件系统）
（有状态服务需要共享支撑）
监控
点：CPU、内存等系统数据，以及服务的状态数据
（时序库的使用、Metrics的设计）
线：端到端的数据，并在每个节点关联点的状态数据
（全局流水号）
面：整个业务区的资源、业务运行情况，如负载与容量

关键设计：应用的升级与回退

企业最常见的三种：

用Replication结合rollingupdate的方式来做灰度，难点在于流量的控制和数据兼容



针对于数据灰度的两种方式：

- 类似现在很多传统行业的方案，只可以增加字段，不可以删除资源
- 从数据访问层中间件着手，往两个库同时差异写（这个有些互联网公司是这样的）

流量控制：

- 业务方解决
- 可通过APIGateway实现

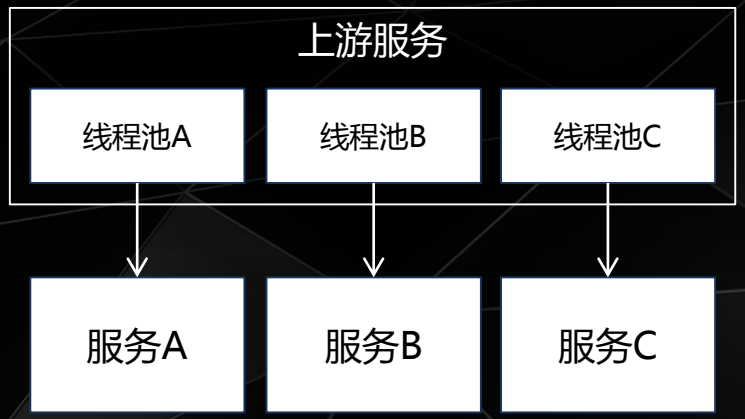
关键设计：简单有效的容错机制

概念

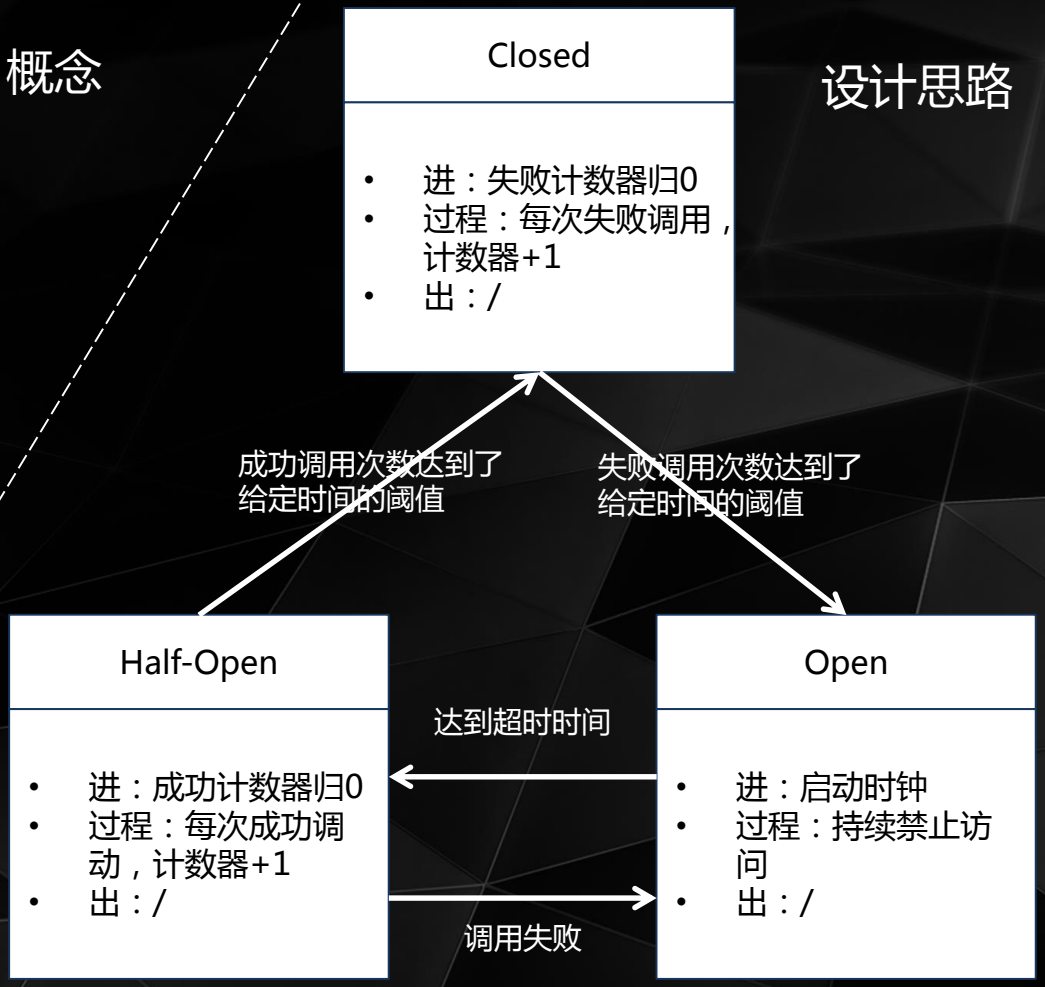
熔断：与股市熔断概念一致，当调用下游服务出现问题时，需要自行进行智能处理

降级：由于整体资源出现瓶颈时，为保证核心服务的可用性，将低等级服务暂停

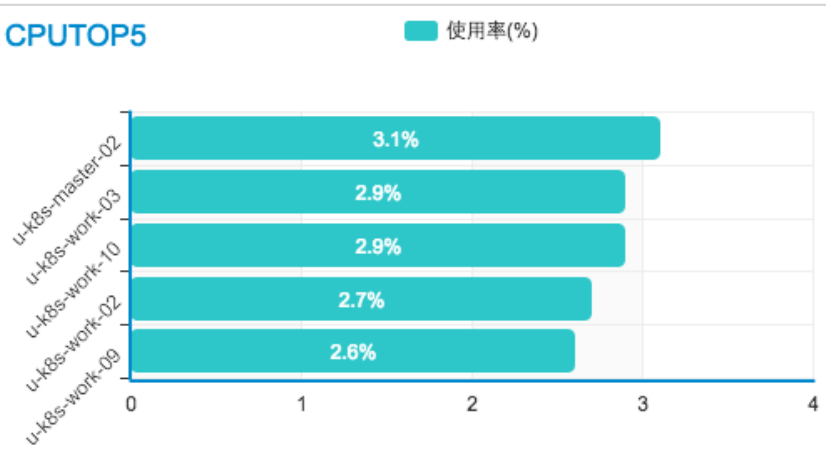
关键点

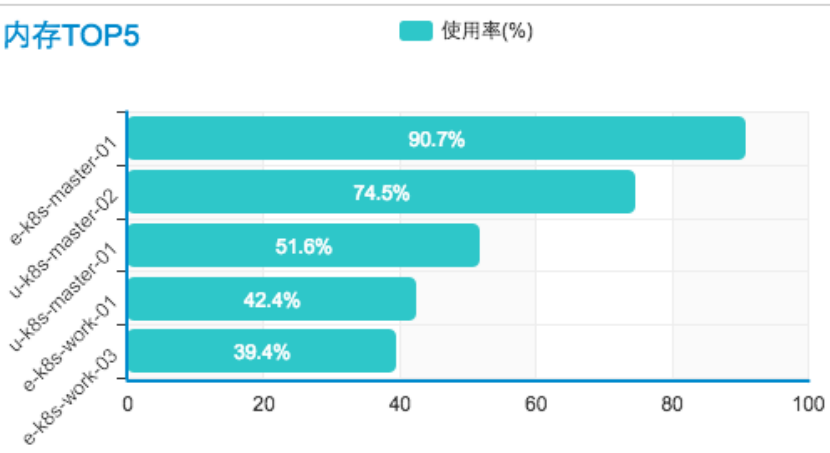


设计思路



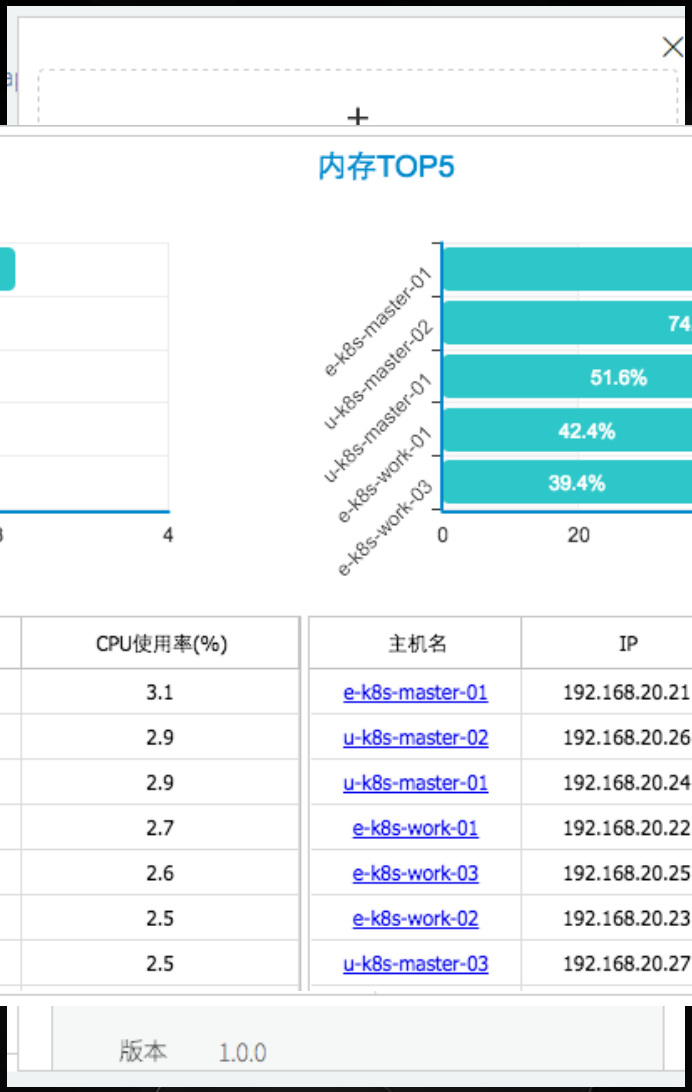
平台截图概览





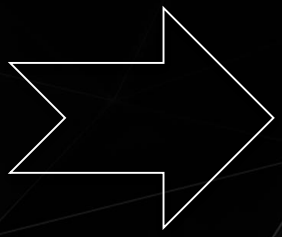
主机名	IP	CPU使用率(%)
u-k8s-master-02	192.168.20.26	3.1
u-k8s-work-03	192.168.20.30	2.9
u-k8s-work-10	192.168.20.37	2.9
u-k8s-work-02	192.168.20.29	2.7
u-k8s-work-09	192.168.20.36	2.6
u-k8s-work-01	192.168.20.28	2.5
u-k8s-work-04	192.168.20.31	2.5

主机名	IP	内存使用量(G)	内存总量(G)	内存使用率(%)
e-k8s-master-01	192.168.20.21	7.07	7.8	90.7
u-k8s-master-02	192.168.20.26	5.81	7.8	74.5
u-k8s-master-01	192.168.20.24	4.02	7.8	51.6
e-k8s-work-01	192.168.20.22	26.71	62.92	42.4
e-k8s-work-03	192.168.20.25	24.79	62.92	39.4
e-k8s-work-02	192.168.20.23	16.85	62.92	26.8
u-k8s-master-03	192.168.20.27	2.05	7.8	26.3



总结与展望

应用服务器已死



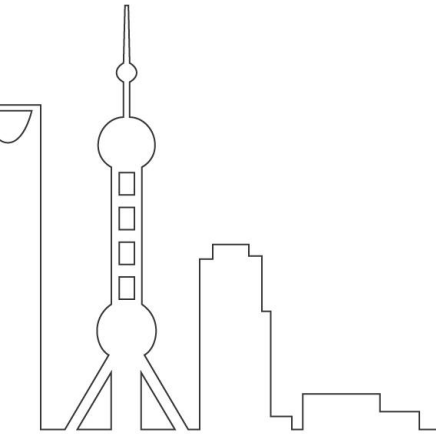
- Docker
- Kubernetes
- 分布式计算
- DevOps
- 微服务
- …

新的应用服务器？

欢迎关注二维码

讲师二维码

扫描二维码关注



Thanks!

International Software Development Conference

主办方 Geekbang. 极客邦科技 InfoQ