
第七届
数据技术嘉年华
Data Technology Carnival
云·数据·智能 - 数聚价值智胜未来

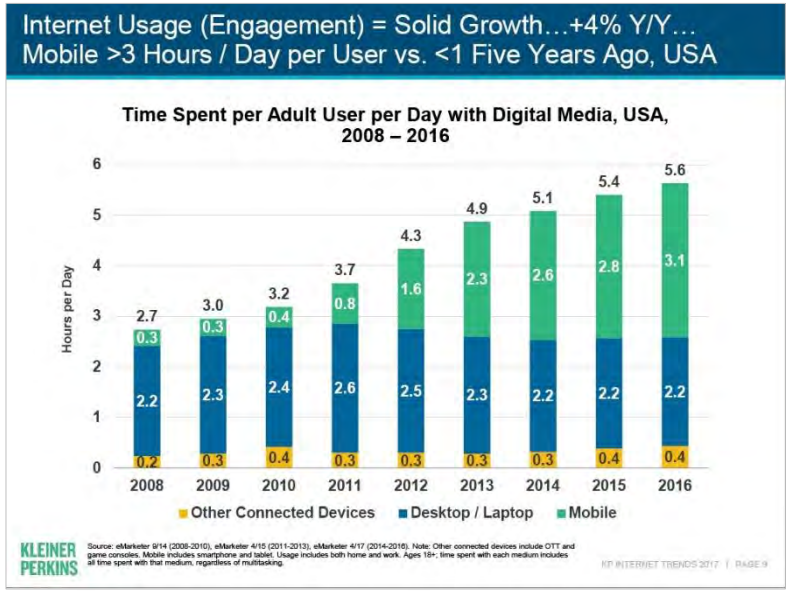
KUBERNETES 与 OPENSTACK 融合
支撑企业级微服务架构

周崇毅 @EasyStack

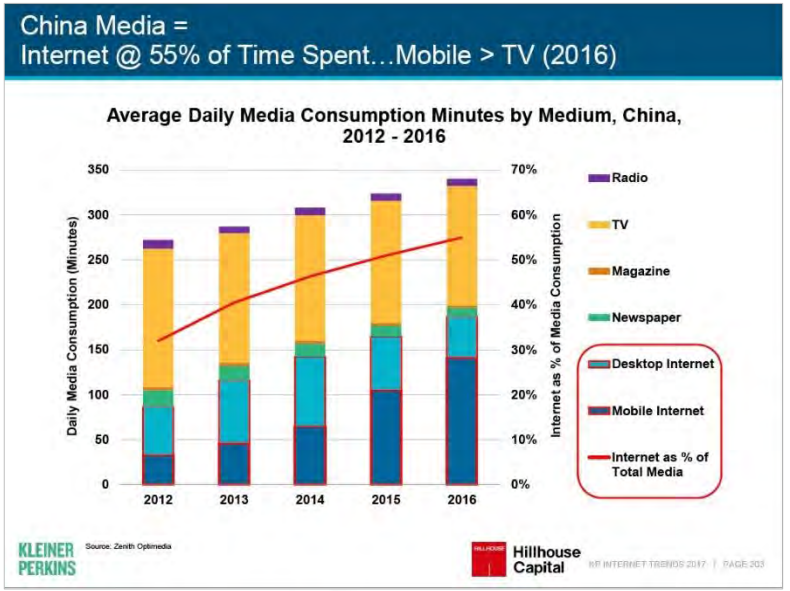
2017.11.

微服务架构基本概念

世界正在改变，历史不可逆转



2016年，全球平均每个用户
花在移动互联网上的时间为3.1小时



2016年，中国人看手机的时间平均超过2小时

云计算拉开”互联网+”架构变革序幕

OpenStack is not the end , it is the beginning or a journey towards ”Internet+” Architecture

互联网+
（”Internet+” Architecture）

移动互联
（Mobile）

社交
（Social）

大数据
（Information）

微服务和分布式架构
（Micro-service）

云计算平台
（Cloud & OpenStack）

软件定义网络
（SDN）

软件定义存储
（SDS）

虚拟化
（virtualization）

IT基础设施
（硬件和操作系统）

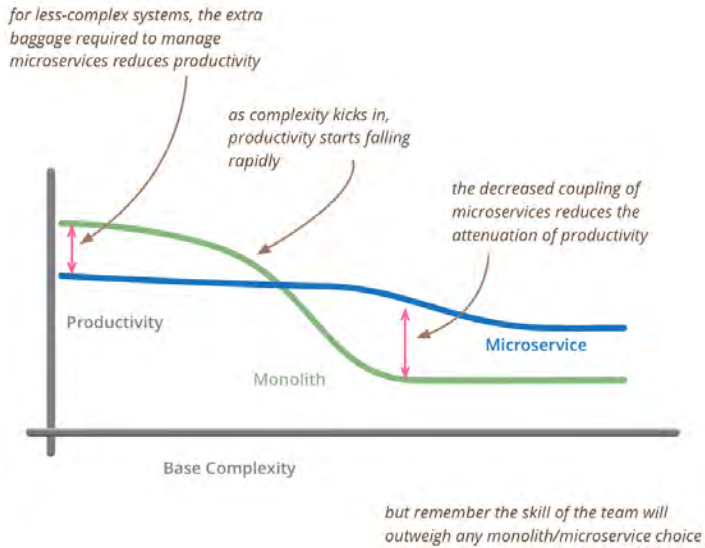
EasyStack
open cloud computing

微服务架构：WHEN？

伴随业务的快速成长，当你的应用面临以下瓶颈：

- 业务复杂度日益增高
- 团队规模日益庞大
- 原有单体应用的维护、升级和部署难度变大
- 应用的迭代效率越来越慢
- 应对高并发业务存在性能瓶颈
- 缺乏足够的容错性

是时候考虑微服务架构了！



单体架构 VS 微服务架构
https://www.martinfowler.com/bliki/MicroservicePremium.html

微服务架构：WHAT?

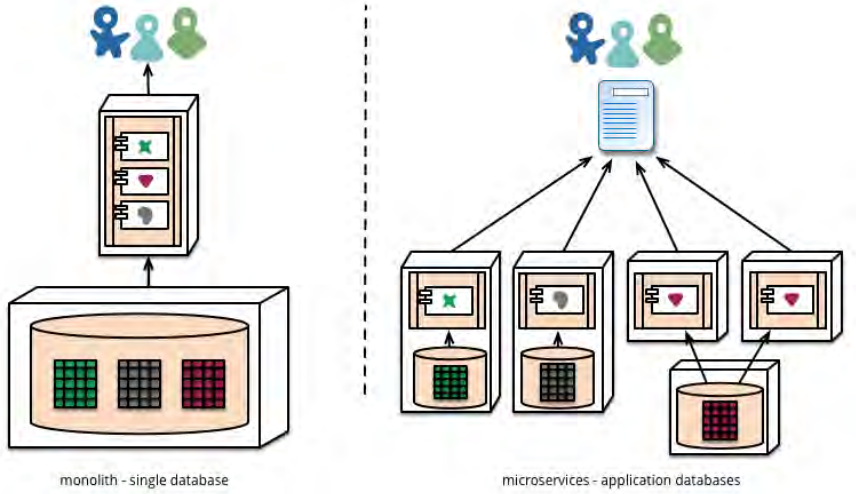
微服务架构是随着云计算的发展应运而生的一种软件设计风格，与之对立的是传统的单体架构

技术视角

- 采用一组服务的方式来构建一个应用系统
- 微服务具备自包含性，可作为独立的进程存在
- 不同微服务之间通过一些轻量级交互机制来通信，通常是HTTP型的API

业务视角

- 单一职责原则 (Single Responsibility Principle)
- 按照业务(而非技术)的边界来确定服务的边界



单体架构 VS 微服务架构
https://www.martinfowler.com/articles/microservices.html

微服务架构：WHY?

1 提升迭代效率

- 与传统单体架构应用不同，每个微服务是一个能**独立发布**的应用服务，可作为独立组件进行编译、部署
- **升级代价小**，大大提升效率，需求变更响应速度快

2 弹性扩展

- 单体架构紧耦合，而微服务强调模块化的结构，微服务之间是**松耦合**的
- 在应用扩展时，仅需扩展有瓶颈的微服务，有利于应用的**模块化扩展**

3 容错能力

- 单体架构应用所有功能在同一进程内实现，一个功能bug可能导致整体崩溃，而微服务可实现**进程级别隔离**
- 每个微服务是**自治**的，单个服务的异常通常不会导致整个系统的故障

4 丰富的技术栈

- 根据业务的需求，不同的服务可以根据业务特性实现**灵活的技术选型**
- 不同的微服务可以依赖不同的编程语言、开发框架以及数据存储技术。针对不同的微服务，可以选择**最合适的技术栈**

5 提高组织效率

- 每个微服务可以由独立的Team开发和维护，依赖方按照统一的接口规范定义好输入和输出即可，**沟通成本低、效率高**
- “2 pizza”原则，团队小而精，**释放生产力**

研发团队小型化已经成为趋势

Enterprise Software (2000 → 2017) = Users Expect Products to be as… Well Designed / Easy-to-Use / Reliable as Consumer Apps

Perpetual, On-Premise Software → Cloud-Based SaaS Apps → Mobile-First Smart Apps

	2000	2017
Delivery Method	On-Prem	Cloud-based
Pricing	Perpetual License	Subscription
UX	Generic	Personalized
Intelligence	Constrained	Unlimited (AI / ML)
Growth Engine	Sales	Product
Purchase Decision	Top-Down	Bottoms-Up
Measure of Engagement & Customer Satisfaction	N/A	DAUs / MAUs / NPS

KLEINER PERKINS

2017，基于云环境的创新模式

Design = Increasingly Core to Enterprise R&D… End-Users Demanding Consumer-Quality Product Experiences

Change in Designer : Developer Ratio, Selected Enterprises, 2010-2017

	2010 - 2012	2017
Atlassian	1 designer : 25 developers	1 designer : 9 developers
Dropbox	N/A	1 designer : 6 developers
IBM	1 designer : 72 developers	1 designer : 8 developers On Mobile – 1 designer : 3 developers
INTERCOM	N/A	1 designer : 5 developers
LinkedIn	1 designer: 11 developers	1 designer : 8 developers

KLEINER PERKINS

Source: Company data, Figma
Note: Ratios for entire orgs, unless noted otherwise. Atlassian historical ratio from 2012; Dropbox data for product org only; IBM historical ratio from 2012, data for product org only; Intercom data for product org only; LinkedIn historical ratio from 2010.

设计者和开发者的比例直线上升

微服务架构：HOW?

业务域&技术域 | 技术域 | 组织结构域&技术域

Step 1

服务建模

- 微服务拆分：领域驱动设计(DDD)

Step 2

架构设计与实现

- 微服务开发框架：Spring Cloud / Dubbo / ...
- 支撑平台：
  - ✓ 容器（**Kubernetes** / Docker / ...）
  - ✓ IaaS平台（**OpenStack** / 公有云 / ...）
  - ✓ PaaS平台（OpenShift / CloudFoundry / ...）

Step 3

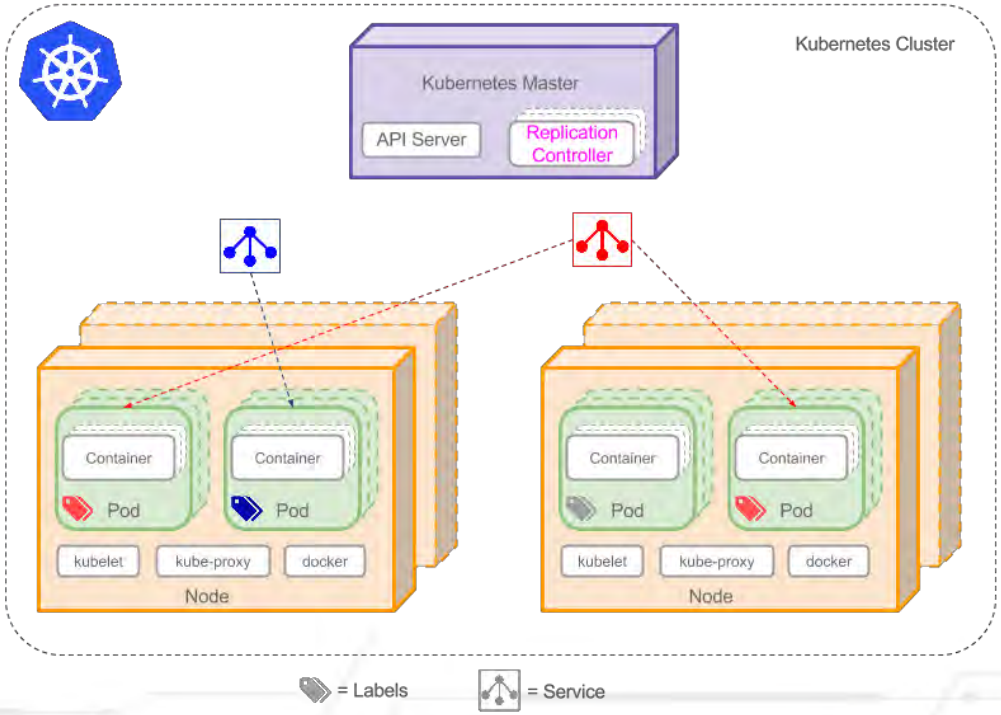
DevOps

- 持续集成: Jenkins / ...
- 代码管理: GitLab / GitHub / ...
- 自动部署：Ansible / Puppet / ...

KUBERNETES与微服务架构

KUBERNETES – 容器编排管理领域的标杆

- Kubernetes
  - 设计理念源于Google过去15年在生产环境中运行容器的管理经验（前身为Google数据中心资源管理系统Borg）
  - 2014年开源，目前已经迭代到V1.5版本
  - 为管理容器而生，核心是面向应用
  - 集合了社区中先进的理念和实战技术
  - 由CNCF(Cloud Native Computing Foundation) 管理
- 主要功能
  - 完备的应用生命周期管理
  - 弹性伸缩
  - 负载均衡
  - 服务编排
  - 服务发现
  - 资源调度
  - 服务自愈机制



KUBERNETES如何支撑微服务架构

服务发现

- Kube-DNS
- Service

配置中心

- ConfigMap
- Secret

负载均衡

- Kube-Proxy
- Service

弹性伸缩

- Auto-Scaling

容错性

- Self-Healing
- Health Check

服务编排

- Deployment
- Helm

升级

- Rolling-Update

任务管理

- Job
- CronJob

日志管理

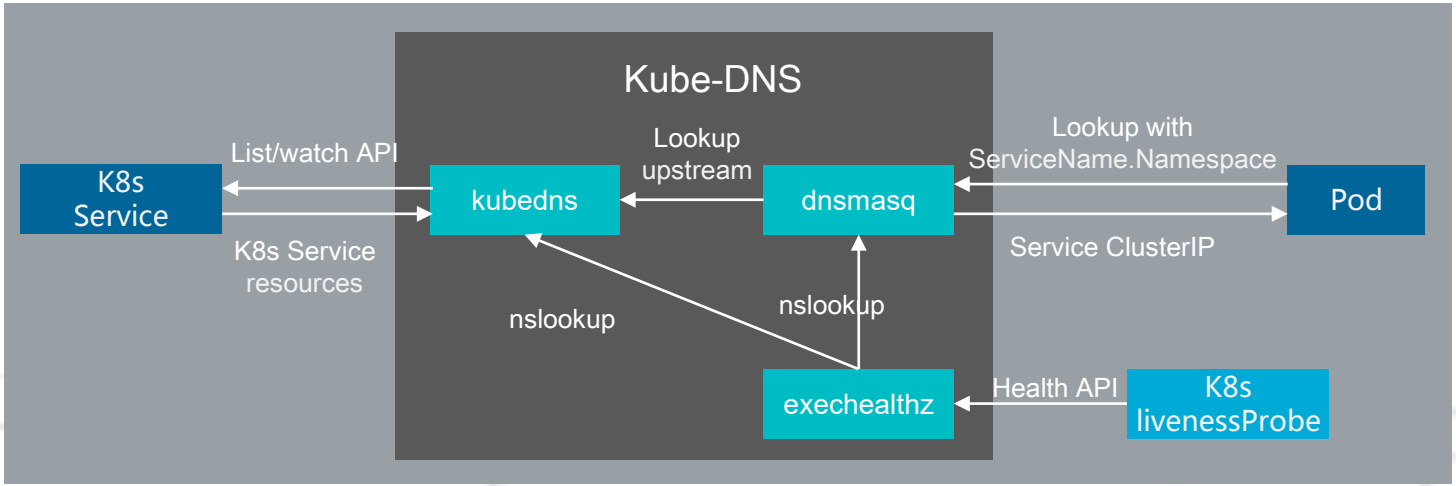
- EFK

监控

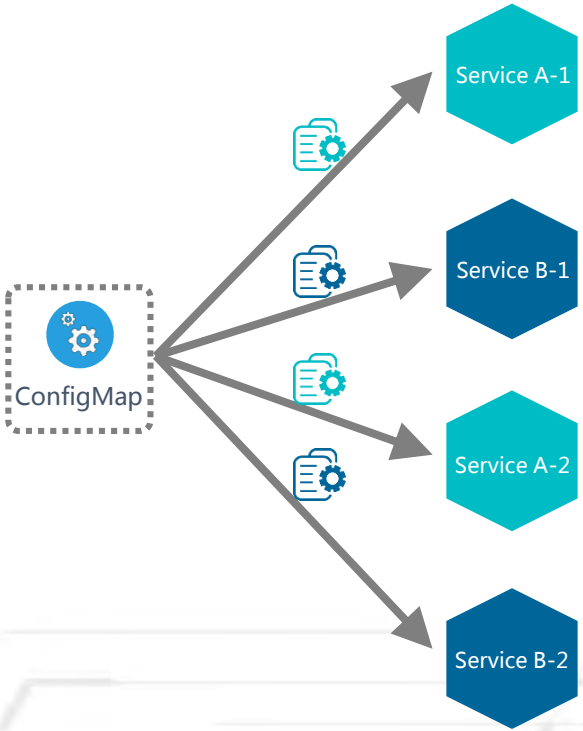
- Prometheus
- Heapster

服务发现

- 基于云平台构建的微服务架构的应用，由于频繁的扩缩容和更新升级，微服务的网络标识经常会动态变化，因此需要引入服务发现机制
- 服务发现的实现方式：云平台内置的服务发现组件、自建服务发现系统（如Etcd、ZooKeeper、Consul）、微服务框架（如SpringCloud Eureka）
- Kubernetes通过集群内部的DNS服务(Kube-DNS)，配合Service，实现服务发现
- 实现原理： Kubernetes将Service的名称当做域名注册到Kube-DNS中，通过DNS服务完成Service名称到Service的ClusterIP的解析，因此通过Service的名称就可以访问其提供的服务



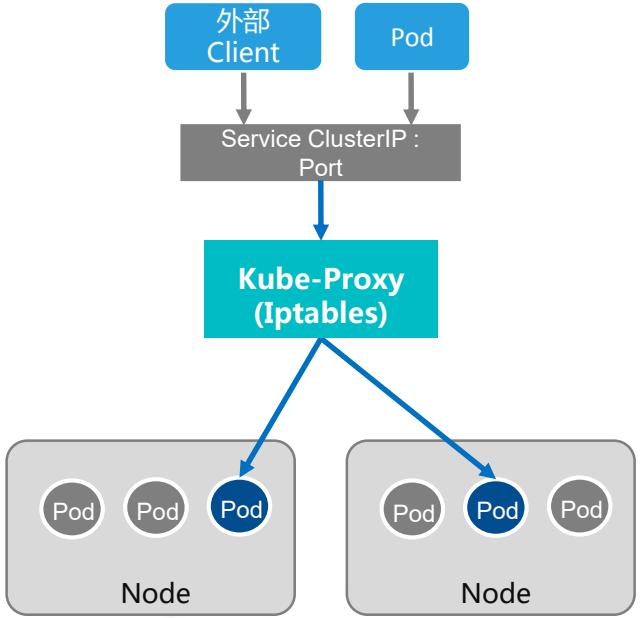
配置中心



- 企业级应用的配置管理复杂：
  - ✓ 配置信息的种类繁多
  - ✓ 多套环境并存：开发、测试、预发布、生产...
  - ✓ 采用微服务设计，服务数量增加导致配置管理复杂度增加
- 配置更新繁琐：
  - ✓ 更新配置经常需要重新构建、部署应用
- 构建微服务架构的**统一配置中心**：Kubernetes ConfigMap
  - ✓ 配置与容器应用分离
  - ✓ 集中管理应用配置
  - ✓ 批量动态更新应用配置
- Secret：处理敏感信息

负载均衡

- Kubernetes Service到后端一组Pod实例的负载均衡和请求转发，由Kubernetes组件Kube-Proxy实现
- Kube-Proxy借助Iptables工具，设置转发规则，将Service的流量进行重定向
- 默认采用轮询方式（Round Robin）进行分配，支持会话保持
- 外部负载均衡实现：
  - ✓ 自建软件/硬件负载均衡器(HAProxy/Nginx/F5）
  - ✓ Kubernetes Ingress
  - ✓ 使用IaaS平台的负载均衡器



弹性伸缩

应对高并发高流量业务场景

定时性/周期性

- 秒杀活动
- 促销活动
- 定时抢票
- ...

→ 提前规划，手动实施弹性伸缩 → 调整RC/RS控制的Pod实例副本数量

突发性

- 突发应急性事件
- 热点事件
- ...

→ 基于监控指标，设置自动弹性伸缩 → 设置HPA，根据Pod负载动态调整实例数量

典型应用行业

- 金融
- 电商
- 运营商
- 公共服务
- 应急管理

容错性

故障自愈（Self-Healing）：

- ✓ 容器异常时，自动重启
- ✓ Node节点宕机时，重新调度并启动容器

应用健康检查（Health-Check）：

- ✓ Liveness探针：检查容器是否处于运行状态，如检测到容器运行状态不正常，则Kubelet会杀掉容器，并根据设置的Restart Policy进行相应操作（如本地重启）
- ✓ Readness探针：判断容器内服务是否已正常工作，如未启动完成或工作异常，则将该服务所在Pod的IP从Service的Endpoint中移除，不接受请求
- ✓ 三种类型的检测：
  - TCP检查：通过容器的IP和端口号进行TCP检查
  - HTTP检查：通过容器的IP、端口号及路径，进行HTTP检查
  - 在容器内部执行一条命令，以判断容器健康状态

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
  name: nginx
spec:
    containers:
    - image: nginx
      imagePullPolicy: Always
      name: http
      livenessProbe:
        httpGet:
        path: /
        port: 80
        initialDelaySeconds: 15
        timeoutSeconds: 1
      readinessProbe:
        httpGet:
        path: /ping
        port: 80
        initialDelaySeconds: 5
        timeoutSeconds: 1
```

服务编排

- **Deployment**
  - ✓ 对Pod和Replica Set的定义和描述模板，目的是更好的解决容器应用的编排问题
  - ✓ 典型应用场景：
    - 通过创建Deployment对象来生成Pod和RS
    - Deployment更新/回滚（均采用灰度方式）
    - 检查Deployment状态来查看部署进度
- **Helm项目**

```
houming@rmbp ~/helm_charts/mysql$ ll templates
total 48
-rwxr-xr-x  1 houming  staff   704B Jan  1  1970 NOTES.txt
-rwxr-xr-x  1 houming  staff   516B Jan  1  1970 _helpers.tpl
-rwxr-xr-x  1 houming  staff   2.5K Jan  1  1970 deployment.yaml
-rwxr-xr-x  1 houming  staff   697B Jan  1  1970 pvc.yaml
-rwxr-xr-x  1 houming  staff   642B Jan  1  1970 secrets.yaml
-rwxr-xr-x  1 houming  staff   365B Jan  1  1970 svc.yaml
```

Deployment编排文件示例

```yaml
kind: Deployment
apiVersion: v1
metadata:
  name: webserver
  labels:
    name: webserver
spec:
  replicas: 3
  selector:
    matchLabels:
      name: webserver
  template:
    metadata:
      labels:
        name: webserver
    spec:
      containers:
        - name: webserver
          image: tomcat
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8080
              protocol: TCP
          env:
            - name: HOME
              value: /
```

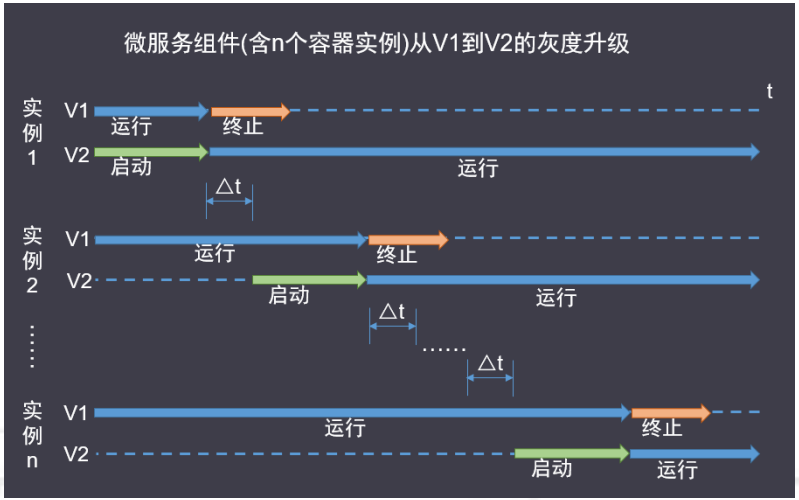
副本数量

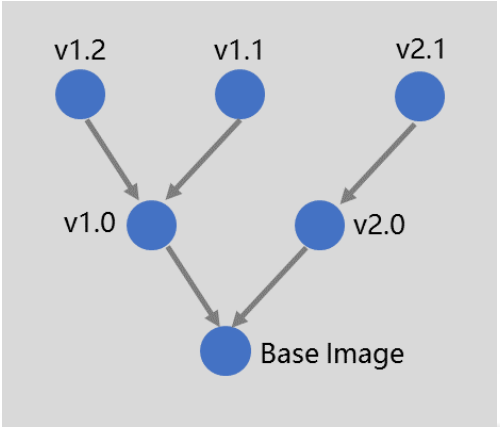
Pod定义模板

灰度升级

降低微服务升级的风险

- **灰度升级**：采用Kubernetes原生的灰度升级方式，保证业务系统在升级过程中不中断，升级过程中新/旧版本并存
- **快速回滚**：Docker镜像版本管理机制，在升级发生问题时，保障镜像版本可随时进行回滚操作





KUBERNETES与OPENSTACK融合支撑微服务架构

容器云平台落地微服务面临的困难

新的问题

- **状态问题**：有状态（集群）服务并不完全适合部署在容器上，如何处理？
- **管理问题**：容器需要与虚拟机、物理机协同工作，如何管理和编排？
- **安全问题**：容器化微服务应用，应用之间如何实现安全隔离？
- **网络问题**：异构支撑平台的SDN网络解决方案，如何选择？
- **存储问题**：异构支撑平台的软件定义存储解决方案，如何选择？

通过Kubernetes与OpenStack的融合，构建一体化支撑平台

KUBERNETES 与 OPENSTACK 融合：1+1>2

OpenStack

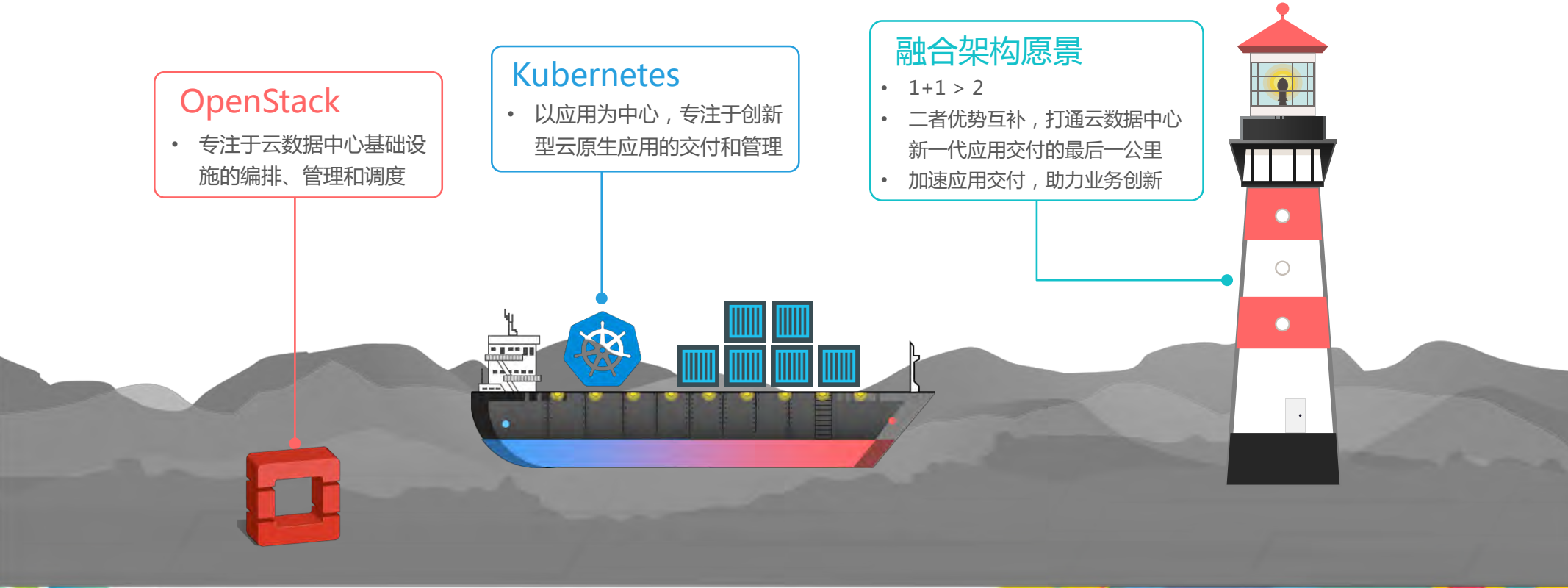
- 专注于云数据中心基础设施的编排、管理和调度

Kubernetes

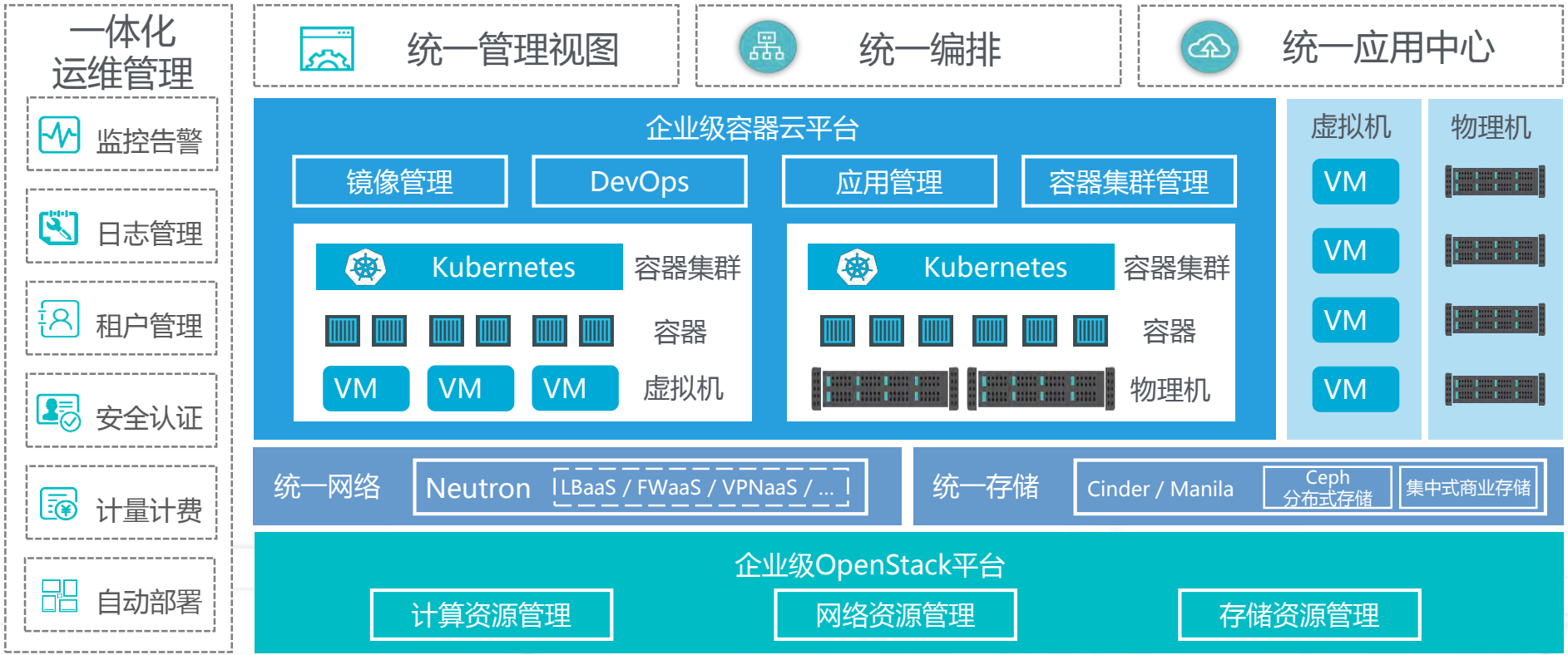
- 以应用为中心，专注于创新型云原生应用的交付和管理

融合架构愿景

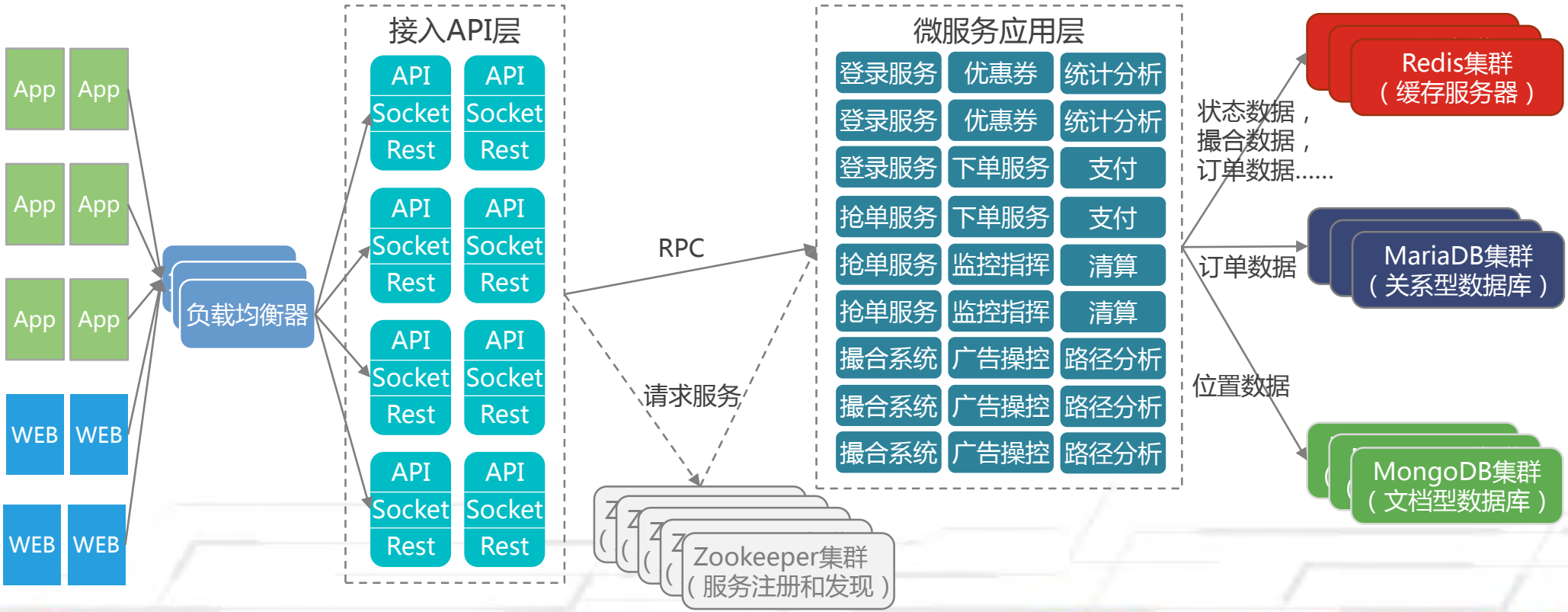
- 1+1 > 2
- 二者优势互补，打通云数据中心新一代应用交付的最后一公里
- 加速应用交付，助力业务创新



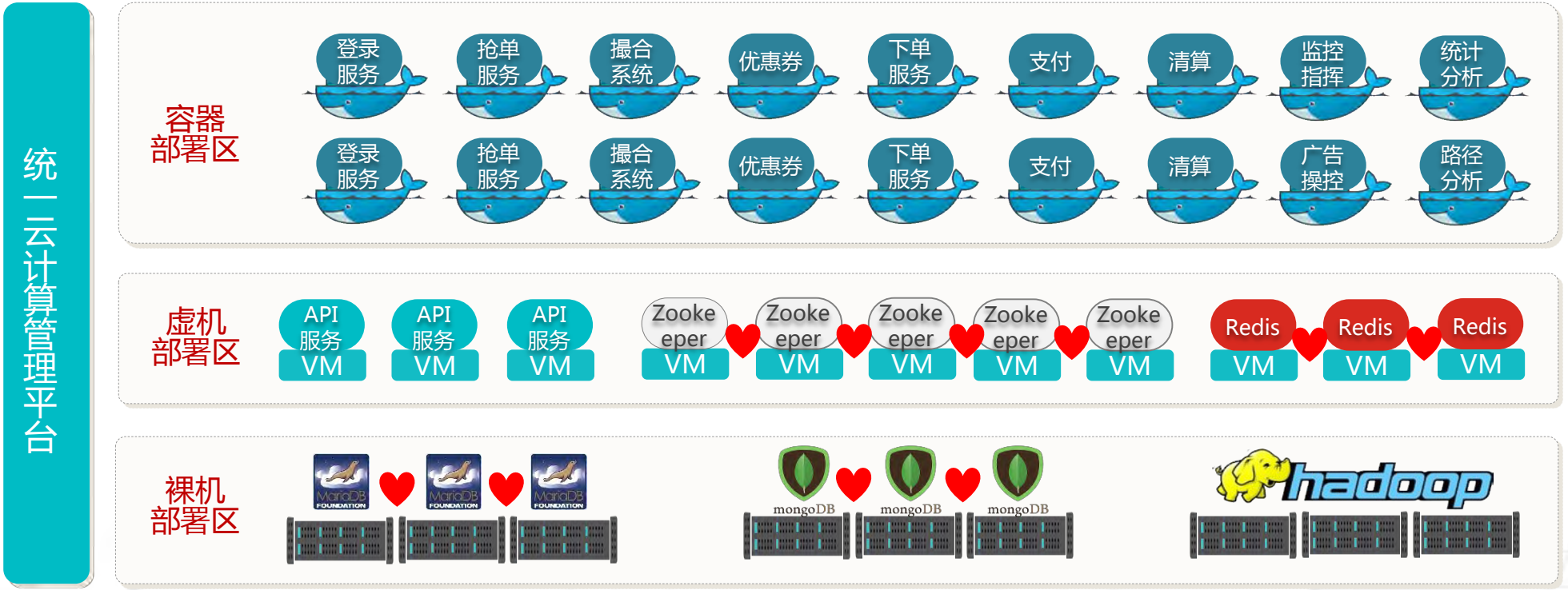
KUBERNETES 与 OPENSTACK 融合架构



基于微服务的应用架构设计案例



统一云平台管理下的应用部署



KUBERNETES 和 OPENSTACK 深度融合带来的收益

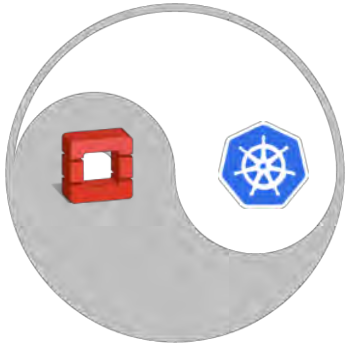
企业IT系统视角

应用架构

- 纷繁复杂的企业级应用架构：传统应用、云原生应用
- 无缝支撑全类型的应用

基础资源

- 异构的私有云基础资源并存：容器、虚拟机、物理机
- 统一的编排、调度和管理



IT组织架构视角

开发者

- 提升应用系统架构设计和开发的灵活度和敏捷度

运维者

- 降低异构的私有云基础设施的运维管理复杂度
- 统一的云平台运维管理

EasyStack——中国开源云计算的领导者

愿景：做以开源技术为核心的下一代基础软件世界级公司

公司成立于2014年2月
中国最大的开源云计算中立厂商

- 3年完成3轮融资，总融资额数亿人民币
- OpenStack核心代码贡献全球前十
- Ceph核心代码贡献全球前十
- Kubernetes核心代码贡献全球前十
- 在全球5个国家和地区有分支机构
- 在全国16个城市有分公司或者办事处
- 200+位员工，专注于开源基础软件