基于K8S的大数据服务供给之道

北京亚信智慧数据科技有限公司
大数据云平台部
叶鹏

- 当前大数据服务供给的难点
- 如何更好的供给大数据服务
- 基于K8s的大数据服务供给探索
- 效果及存在问题

大数据技术在不断快速更新

少量共性　　**大数据应用**　　大量差异

GBASE　APACHE STORM　mongoDB　PostgreSQL　H2O.ai

HBASE　Spark　Apache Solr　neo4j　TensorFlow

MySQL　redis　kafka

RabbitMQ　GREENPLUM DATABASE　cassandra　elasticsearch.

但这些年都是如何开始应用大数据工具

大数据服务供给的问题

手工操作多

- 基础环境准备
- 大数据服务配置

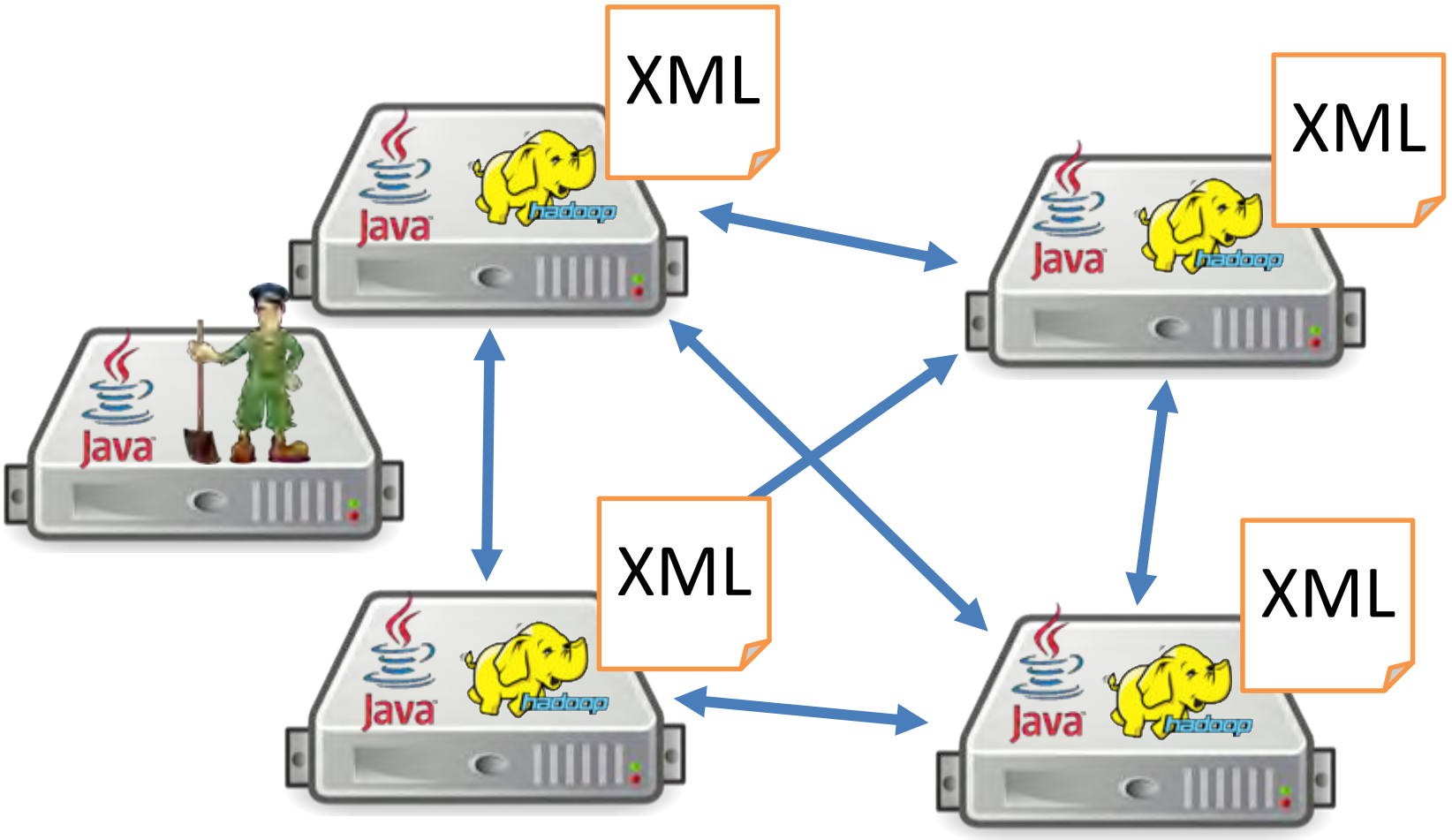
部署时间长

- 从零开始
- 环境差异
- 成功经验不能很好复制

不确定因素多

- 底层服务与上层应用冲突
- 资源预计不足或者过多

大数据服务供给的问题



更大的问题

逻辑复杂
交付效率低

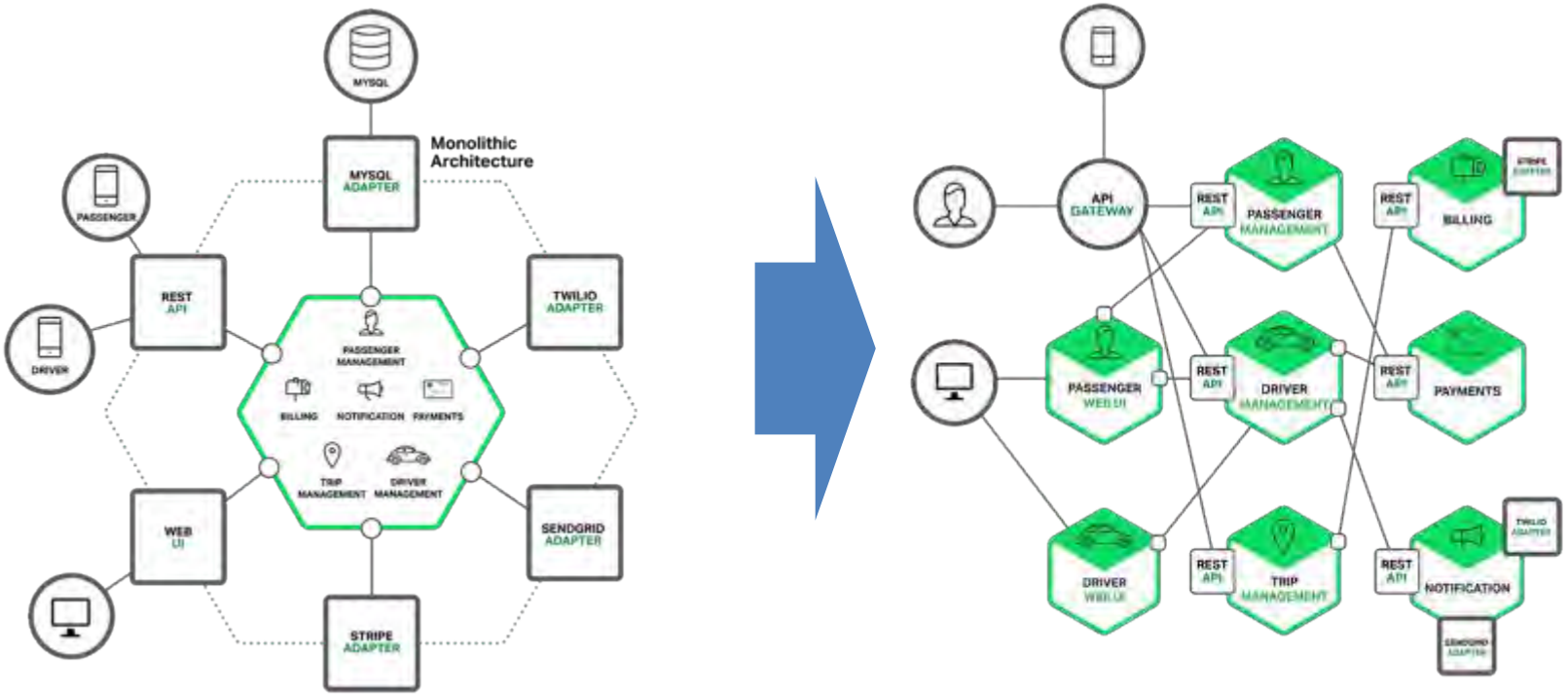
功能层层叠加
烟囱式应用

- 当前大数据服务供给的难点
- 如何更好的供给大数据服务
- 基于K8s的大数据服务供给探索
- 效果及存在问题

微服务时代大数据服务还要单体?

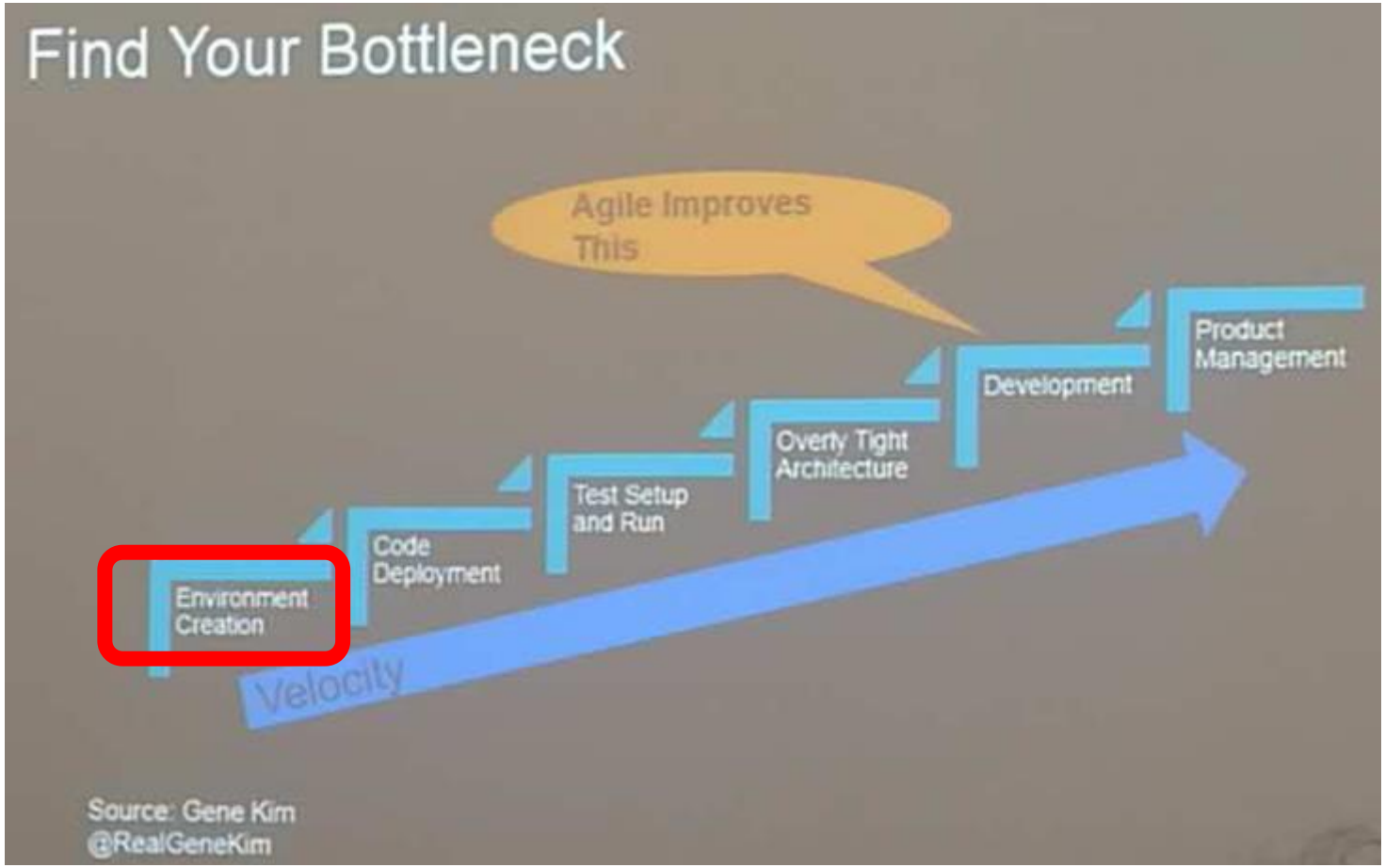
BLOG TECH Chris Richardson · May 19, 2015

Introduction to Microservices



https://www.nginx.com/blog/introduction-to-microservices/

大数据服务的供给如何满足DevOps要求



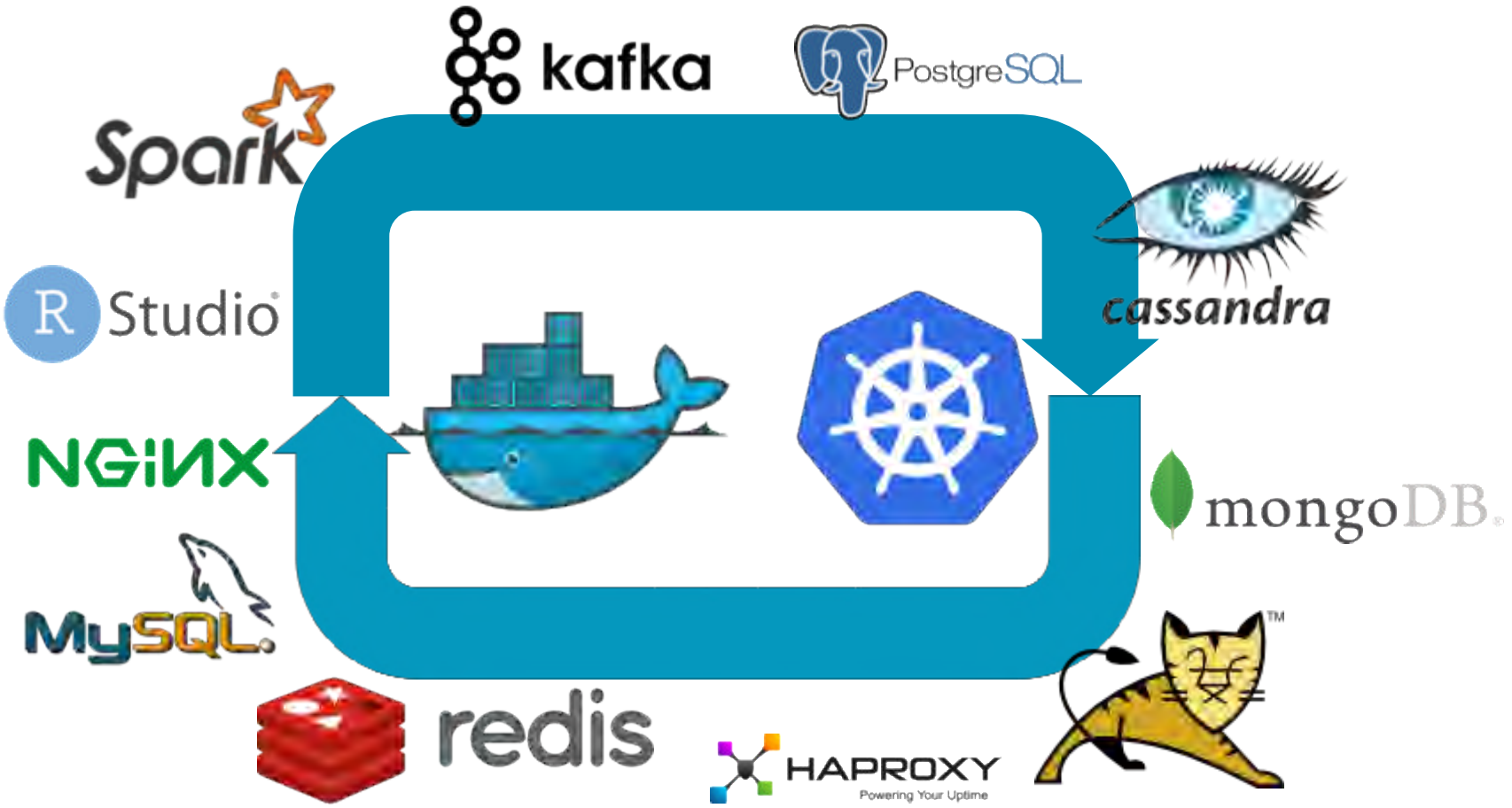
Atlassian 《数字转型的成功：敏捷和DevOps在企业的应用》
2016.11.19

大数据服务供给的基本需求

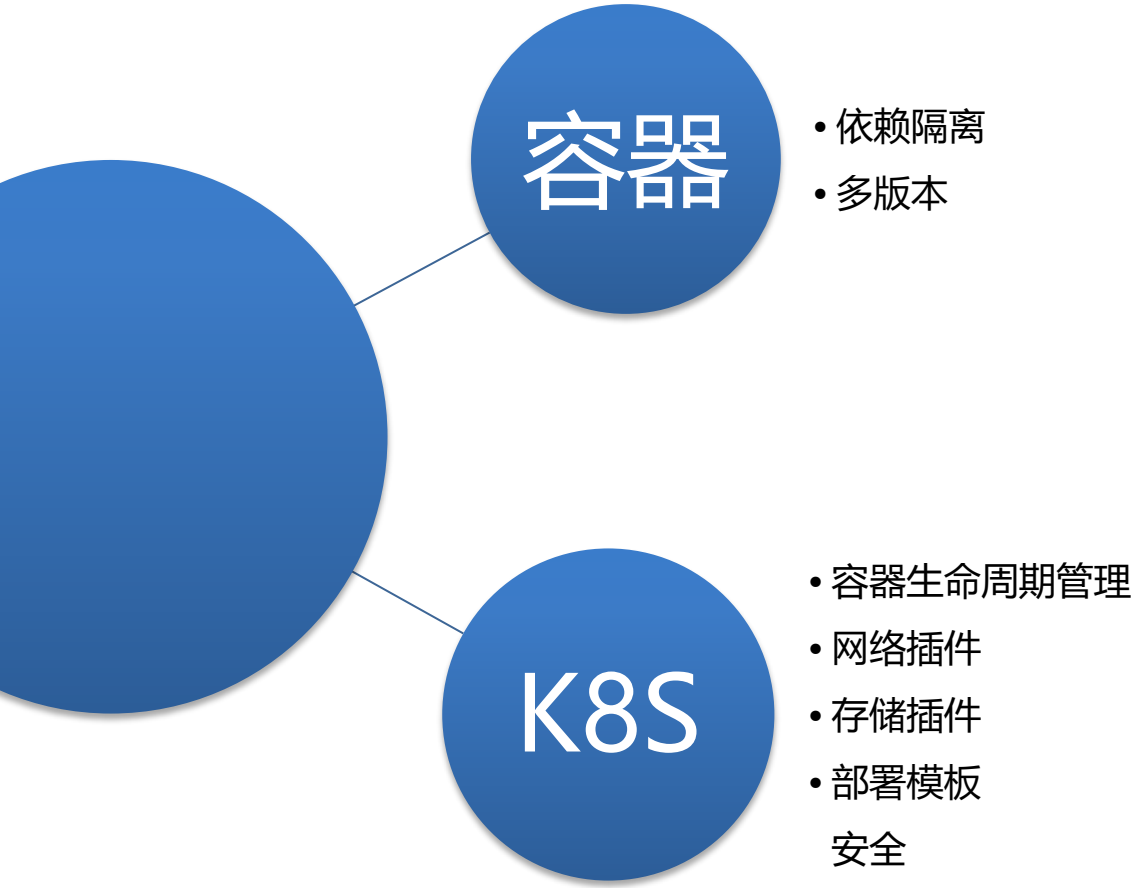
编排	• 代码定义的大数据服务 • 可重复性
自动供给	• 响应式按需供给 • 快速发布至应用系统
资源隔离	• 大数据服务间的资源隔离 • 大数据服务内的资源隔离
依赖隔离	• 对交付环境的隔离
自服务	• 服务申请 • 应用绑定

- 当前大数据服务供给的难点
- 如何更好的供给大数据服务
- 基于K8s的大数据服务供给探索
- 效果及存在问题

基于K8S的大数据服务供给



通过容器和K8S部署大数据服务

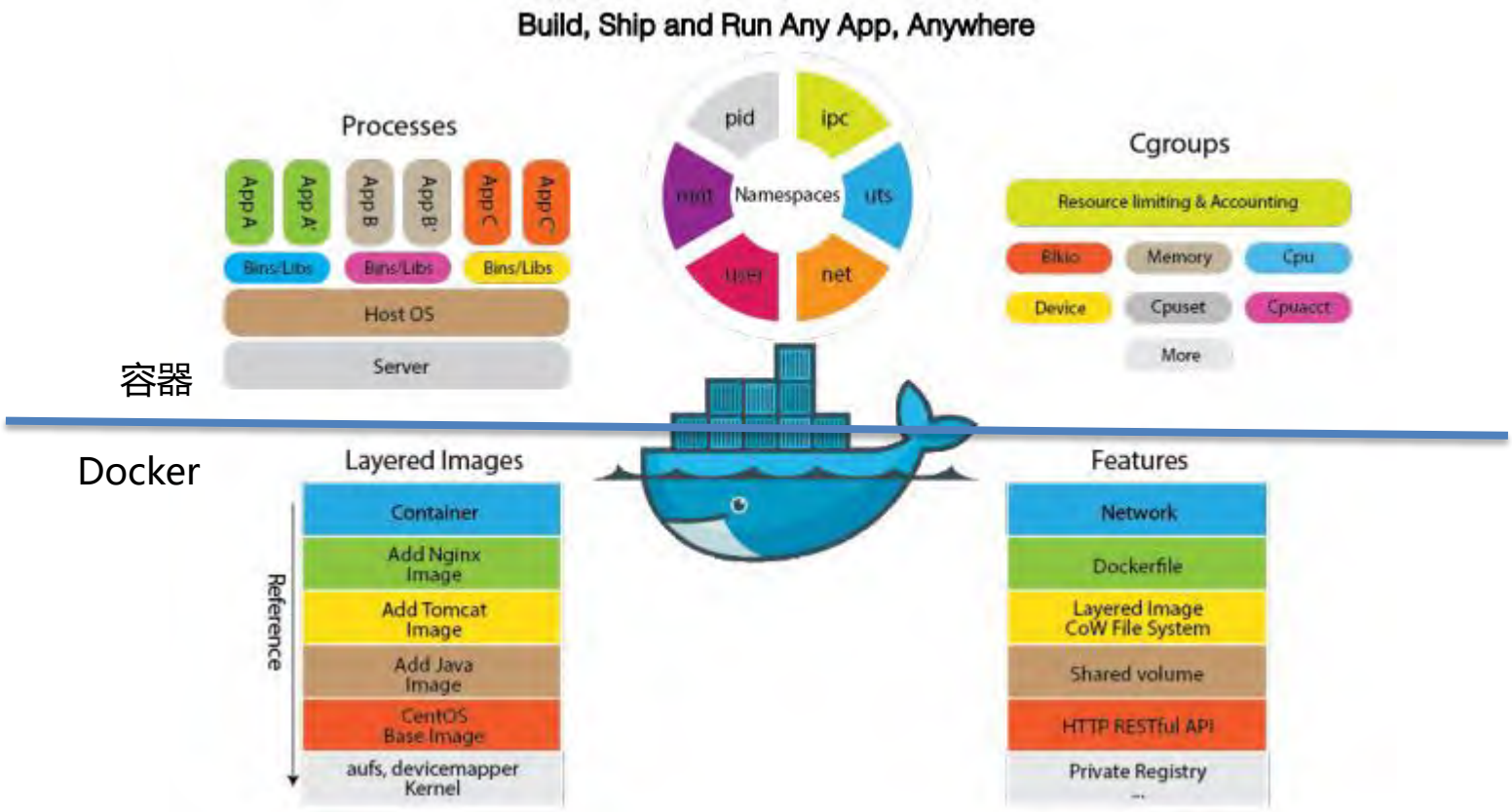


- 依赖隔离
- 多版本

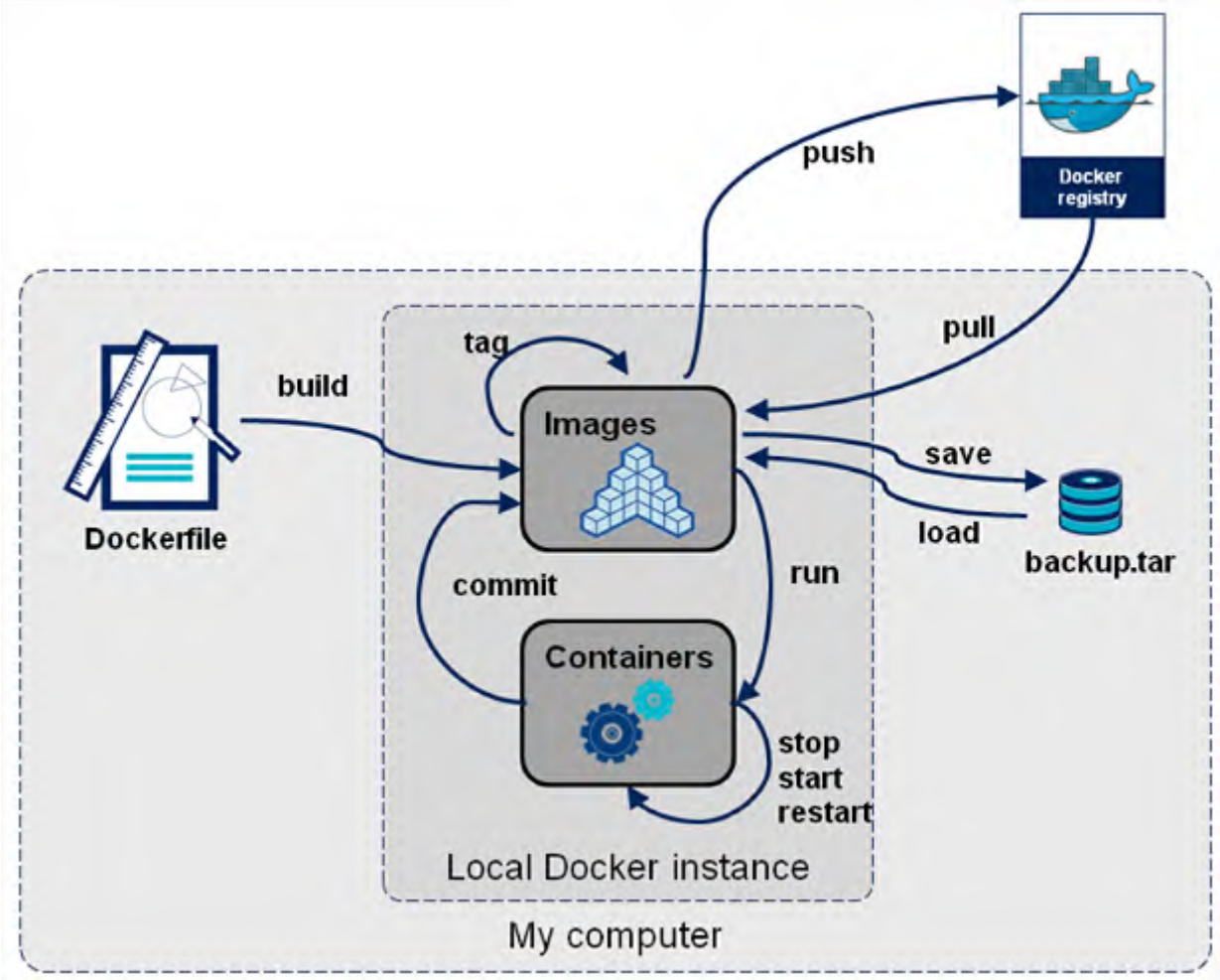
- 容器生命周期管理
- 网络插件
- 存储插件
- 部署模板

安全

Docker与容器

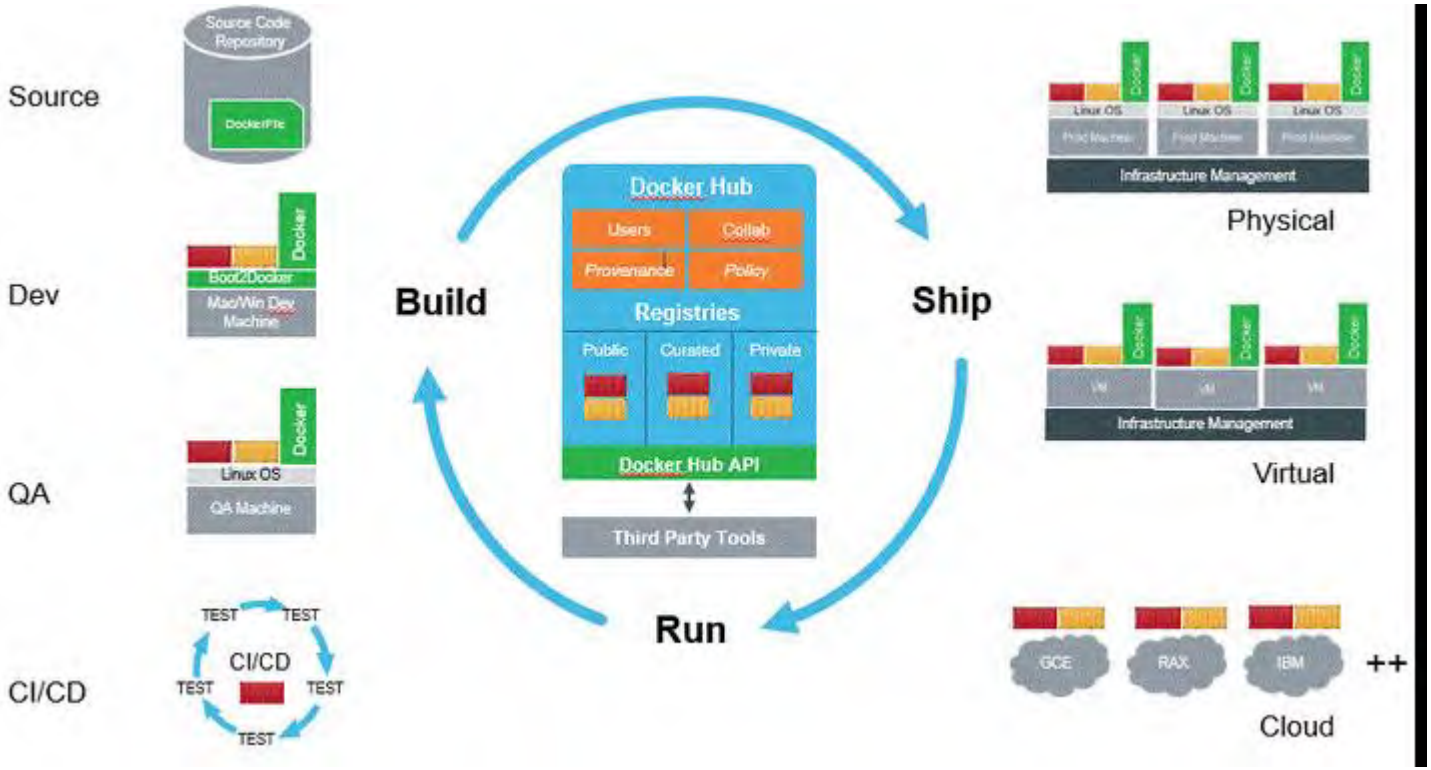

Build, Ship and Run Any App, Anywhere
Processes
App A
App A'
App B
App B'
App C
App C'
Bins/Libs
Bins/Libs
Bins/Libs
Host OS
Server
pid
ipc
mnt
Namespaces
uts
net
Cgroups
Resource limiting & Accounting
Blkio
Memory
Cpu
Device
Cpuset
Cpuacct
More
容器
Docker
Layered Images
Container
Add Nginx Image
Add Tomcat Image
Add Java Image
CentOS Base Image
aufs, devicemapper Kernel
Reference
Features
Network
Dockerfile
Layered Image CoW File System
Shared volume
HTTP RESTful API
Private Registry

Docker应用生命周期

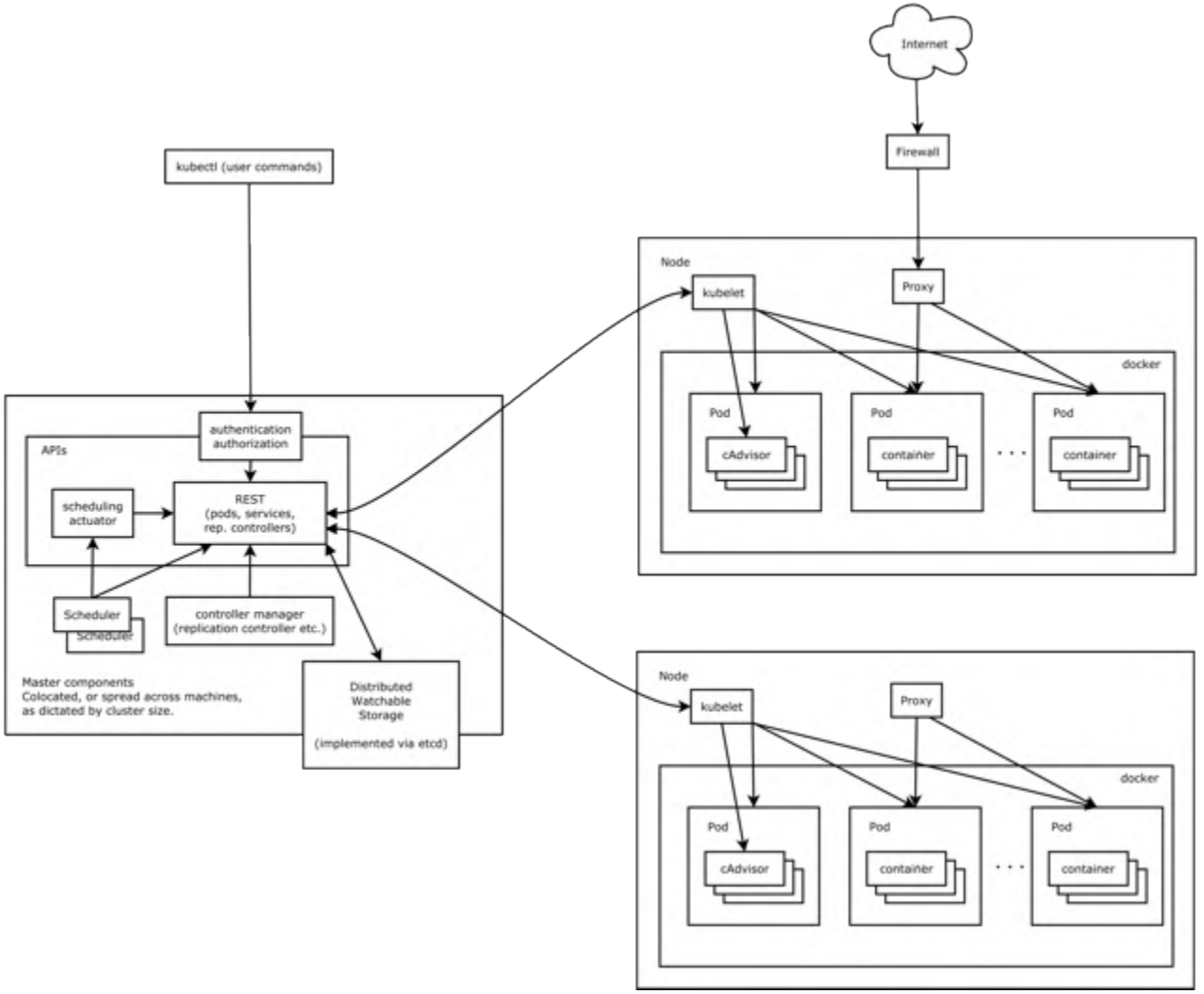


Docker给开发、交付带来的变化

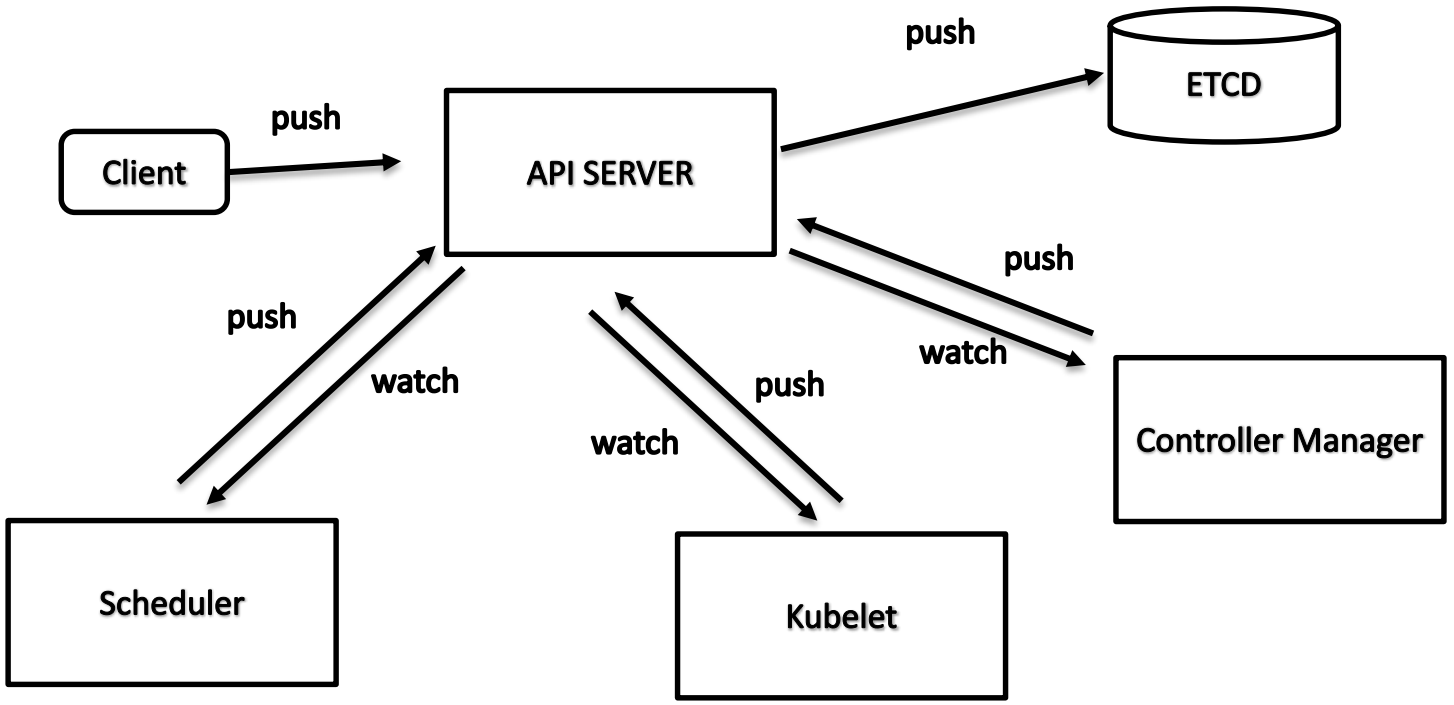
不变应万变



Kubernetes技术架构

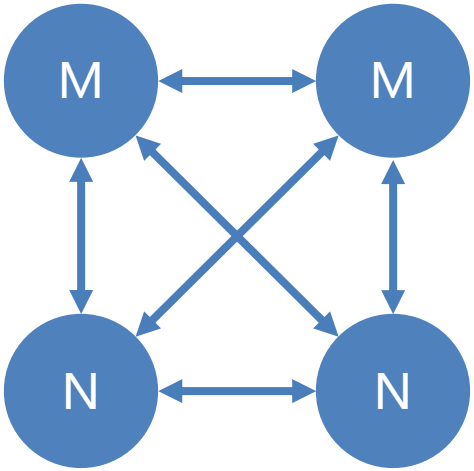


kubernetes微服务架构设计

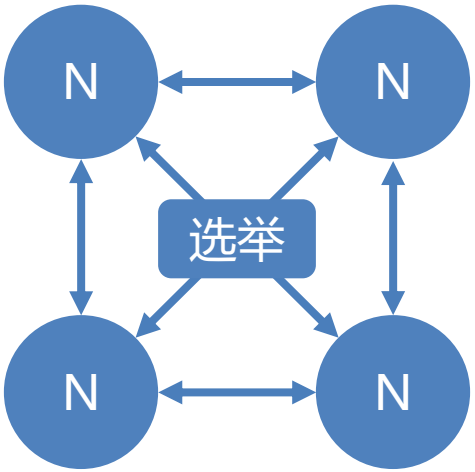


常见大数据服务部署模型

- 大数据服务多以主从结构方式部署



指定主节点



选举主节点

大数据服务编排的基础条件

服务发现

- 预先分配的节点名
- 节点名相对固定
- 特定端口的互通

数据存储

- 应用数据
- 配置数据

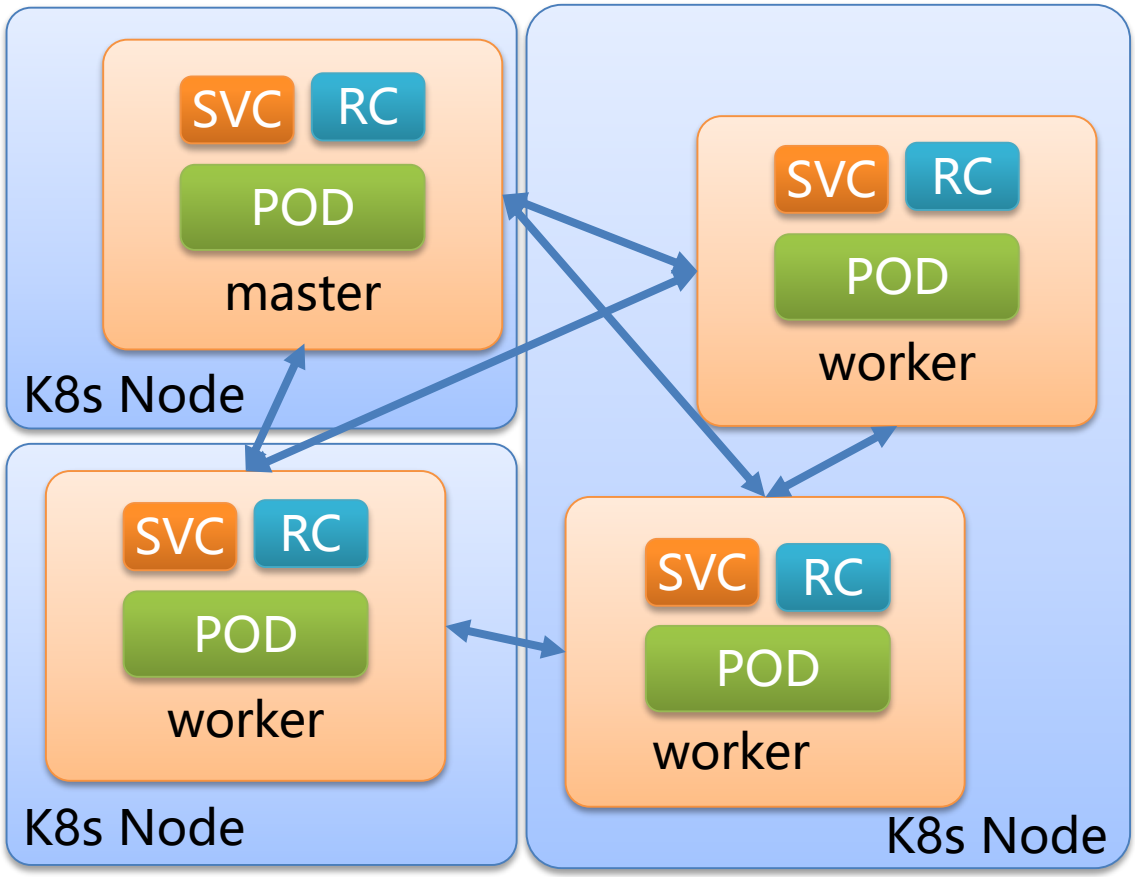
运维管理

- 生命周期管理
- 可用性探测
- 性能、日志数据采集
- 弹性扩缩容

大数据服务编排中常用的K8S组件

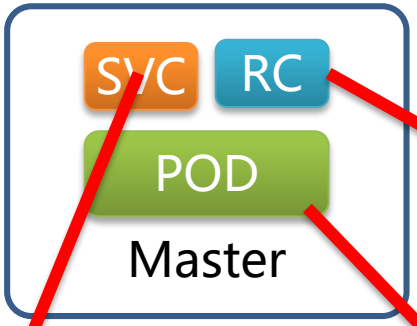
组件	功用
POD	kubernetes的最小调度单元，运行容器，在大数据服务部署过程中代表一个“物理节点”
Service	提供相对固定的服务名字，在大数据服务部署过程中代表一个逻辑节点，供大数据服务节点间相互发现，应用与大数据服务间相互发现
RC	负责POD的期望副本数与实际数的一致性，提供最基本的服务连续性
Heathcheck	可用性探测
Lifecycle	生命周期管理
PetSet	kubernetes的新特性，一定程度上具备RC+Service的特性

基于K8S的大数据服务编排模型



- Service
  - 服务发现，DNS
- RC
  - 生命周期管理
- POD
  - 大数据服务节点
  - 生命周期管理
  - 存储
  - 网络

一个简单的spark服务-master



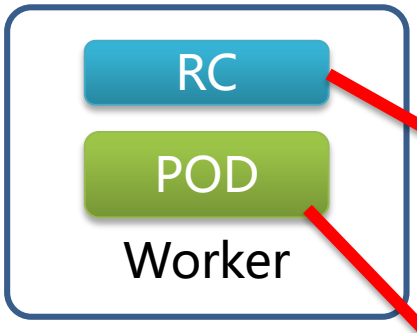
service.yaml

```yaml
kind: Service
apiVersion: v1
metadata:
  name: sb-instanceid-sm
spec:
  ports:
    - port: 7077
      targetPort: 7077
  selector:
    servicebroker: sb-instanceid-spark-master
```

rc.yaml

```yaml
kind: ReplicationController
apiVersion: v1
metadata:
  name: sb-instanceid-spark-master
spec:
  replicas: 1
  selector:
    servicebroker: sb-instanceid-spark-master
  template:
    metadata:
      labels:
        servicebroker: sb-instanceid-spark-master
    spec:
      containers:
        - name: sb-instanceid-spark-master
          env:
            - name: SPARK_MASTER
              value: sb-instanceid-sm
            - name: SPARK_SECRET
              value: test1234
          image: registry.dataos.io/guestbook/spark:v1.6.2
          command: ["/start-master"]
          ports:
            - containerPort: 7077
            - containerPort: 8080
```

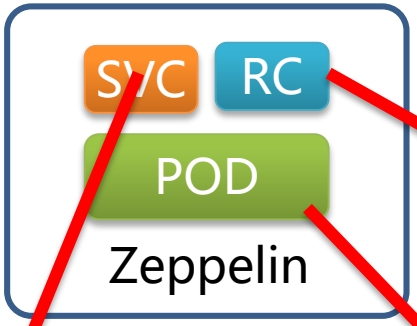
一个简单的spark服务-worker



rc.yaml

```yaml
kind: ReplicationController
apiVersion: v1
metadata:
  name: sdcc2016-spark-worker
spec:
  replicas: 3
  selector:
    servicebroker: sdcc2016-spark-worker
  template:
    metadata:
      labels:
        servicebroker: sdcc2016-spark-worker
    spec:
      containers:
        - name: sdcc2016-spark-worker
          env:
            - name: SPARK_MASTER
              value: sdcc2016-sm
            - name: SPARK_SECRET
              value: test1234
          image: registry.dataos.io/guestbook/spark:v1.6.2
          command: ["/start-worker"]
          ports:
            - containerPort: 8081
```

一个简单的spark服务-zeppelin



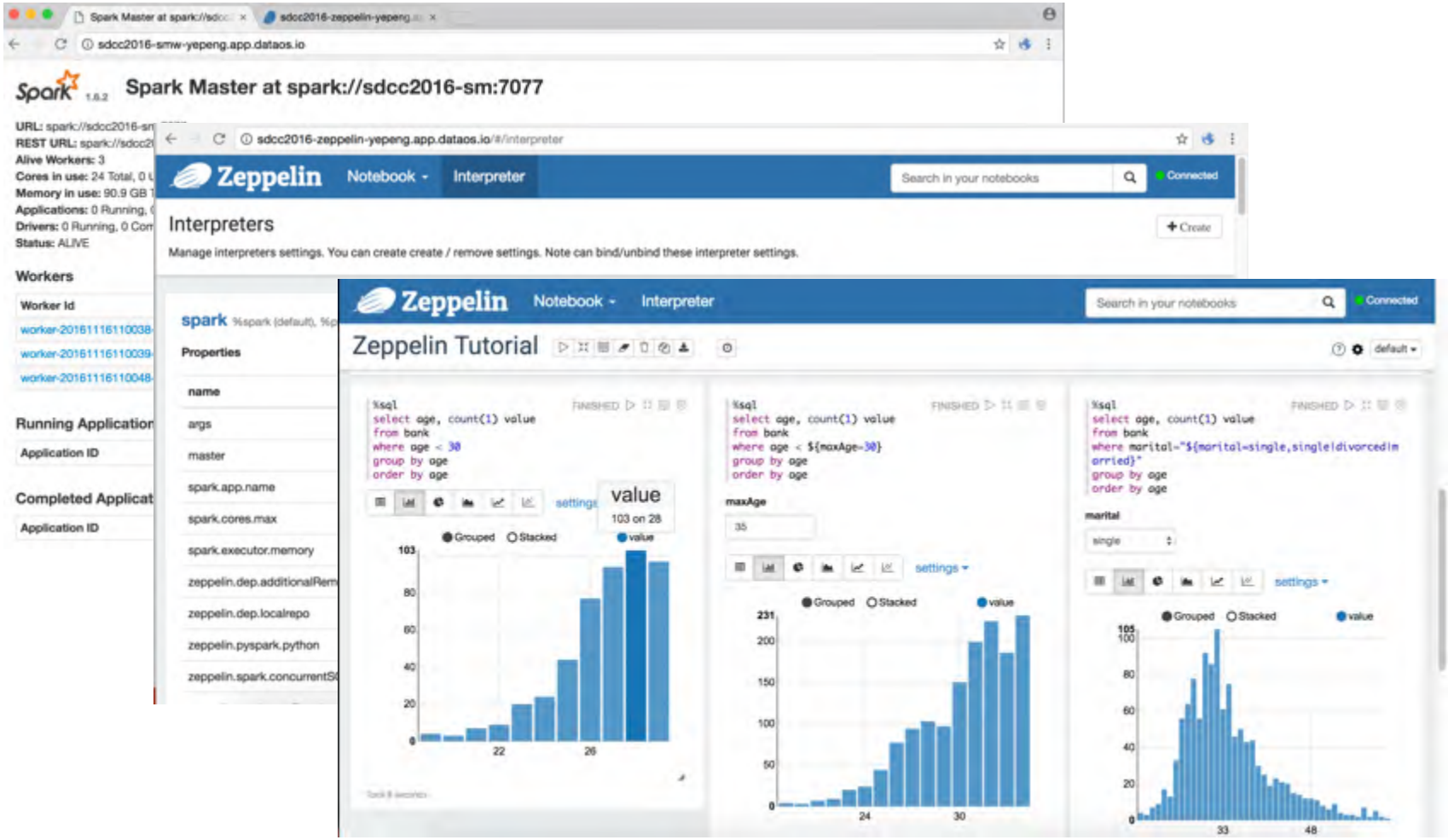
service.yaml

```yaml
kind: Service
apiVersion: v1
metadata:
  name: sdcc2016-zeppelin
spec:
  ports:
    - port: 8080
      targetPort: 8080
  selector:
    servicebroker: sdcc2016-zeppelin
```

rc.yaml

```yaml
kind: ReplicationController
apiVersion: v1
metadata:
  name: sdcc2016-zeppelin
spec:
  replicas: 1
  selector:
    servicebroker: sdcc2016-zeppelin
  template:
    metadata:
      labels:
        servicebroker: sdcc2016-zeppelin
    spec:
      containers:
        - name: sdcc2016-zeppelin
          env:
            - name: SPARK_MASTER
              value: sdcc2016-sm
            - name: SPARK_SECRET
              value: test1234
          image: registry.dataos.io/guestbook/zeppelin:v0.5.6-incubating
          ports:
            - containerPort: 8080
```

一个简单的spark服务

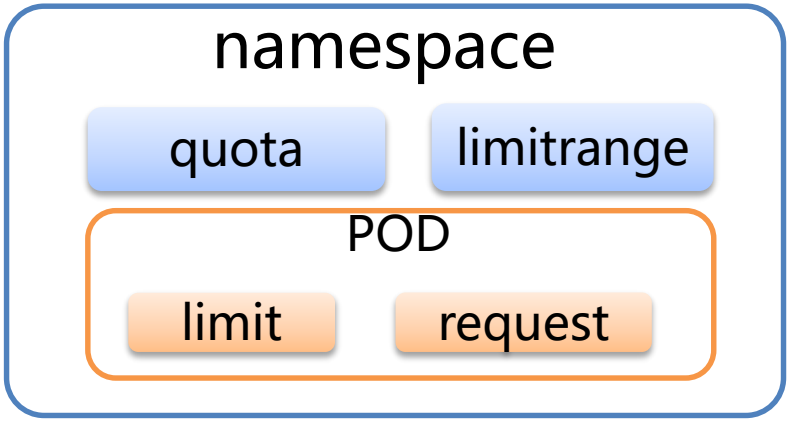


给大数据服务分配独立的资源

- quota/limits/request

rc.yaml

```
spec:
  containers:
  - command:
    - /start-worker
    env:
    - name: SPARK_MASTER
      value: sdcc2016-sm
    - name: SPARK_SECRET
      value: test1234
    - name: SPARK_WORKER_MEMORY
      value: 1000m
    - name: SPARK_WORKER_CORES
      value: "1"
    image: registry.dataos.io/guestbook/spark:v1.6.2
    imagePullPolicy: IfNotPresent
    name: sdcc2016-spark-worker
    ports:
    - containerPort: 8081
      protocol: TCP
    resources:
      limits:
        cpu: 500m
        memory: 1000Mi
    terminationMessagePath: /dev/termination-log
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  securityContext: {}
  terminationGracePeriodSeconds: 30
```




```
- name: SPARK_WORKER_MEMORY
  value: 1000m
- name: SPARK_WORKER_CORES
  value: "1"
```

```
resources:
  limits:
    cpu: 500m
    memory: 1000Mi
```

提高大数据服务可用性

- RC提供了基本的可用性
- 使用kubernetes应用生命周期管理功能
  - 周期性探测
    - LivenessProbe，如果探测失败重启POD
    - ReadinessProbe，如果探测失败将POD从Service EndPoint列表中移除
  - 容器启动、退出任务
    - postStart，容器创建后执行相关任务
    - preStop，容器退出前执行相关任务
  - 执行方式
    - exec执行命令（0）
    - httpGet请求（200 or 399）
    - tcpsocket连接（建立连接）

大数据服务的基础安全强化

- 用户名密码
- 证书
- HTTPS
- ……

SSL Configuration

Configuration for SSL is organized hierarchically. The user can configure the default SSL settings which will be used for all the supported communication protocols unless they are overwritten by protocol-specific settings. This way the user can easily provide the common settings for all the protocols without disabling the ability to configure each one individually. The common SSL settings are at `spark.ssl` namespace in Spark configuration, while Akka SSL configuration is at `spark.ssl.akka` and HTTP for broadcast and file server SSL configuration is at `spark.ssl.fs`. The full breakdown can be found on the configuration page.

Web UI

The Spark UI can also be secured by using javax servlet filters via the `spark.ui.filters` setting. A user may want to secure the UI if it has data that other users should not be allowed to see. The javax servlet filter specified by the user can authenticate the user and then once the user is logged in, Spark can compare that user versus the view ACLs to make sure they are authorized to view the UI. The configs `spark.acls.enable` and `spark.ui.view.acls` control the behavior of the ACLs. Note that the user who started the application always has view access to the UI. On YARN, the Spark UI uses the standard YARN web application proxy mechanism and will authenticate via any installed Hadoop filters.

rc.yaml

```
env:
  - name: SPARK_MASTER
    value: sb-instanceid-sm
  - name: SPARK_SECRET
    value: test1234
```

spark-defaults.conf

```
spark.master spark://SPARK_MASTER:7077
spark.driver.extraLibraryPath /opt/hadoop/lib/native
spark.app.id KubernetesSpark
spark.authenticate true
spark.authenticate.secret SPARK_SECRET
```

如何让大数据服务满足个性化要求

- 环境变量
- secret/configmap

cassandra.yaml

```
num_tokens: 256


hinted_handoff_enabled: true
hinted_handoff_throttle_in_kb: 1024
max_hints_delivery_threads: 2

batchlog_replay_throttle_in_kb: 1024

authenticator: PasswordAuthenticator

authorizer: CassandraAuthorizer

permissions_validity_in_ms: 2000

partitioner: org.apache.cassandra.dht.Murmur3Partitioner

data_file_directories:
    - /cassandra_data/data

commitlog_directory: /cassandra_data/commitlog

disk_failure_policy: stop

commit_failure_policy: stop

key_cache_size_in_mb:

key_cache_save_period: 14400


row_cache_size_in_mb: 0

row_cache_save_period: 0
```

configmap.yaml

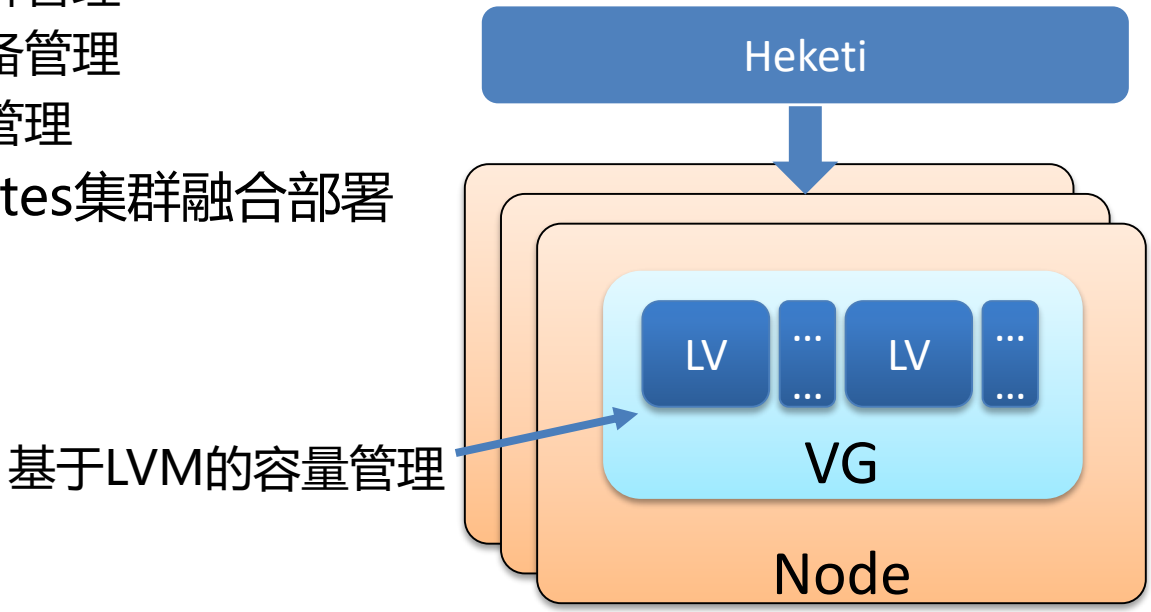
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cassandra-config
  namespace: cassandra
data:
  cassandra.yaml: "……"
```

rc.yaml

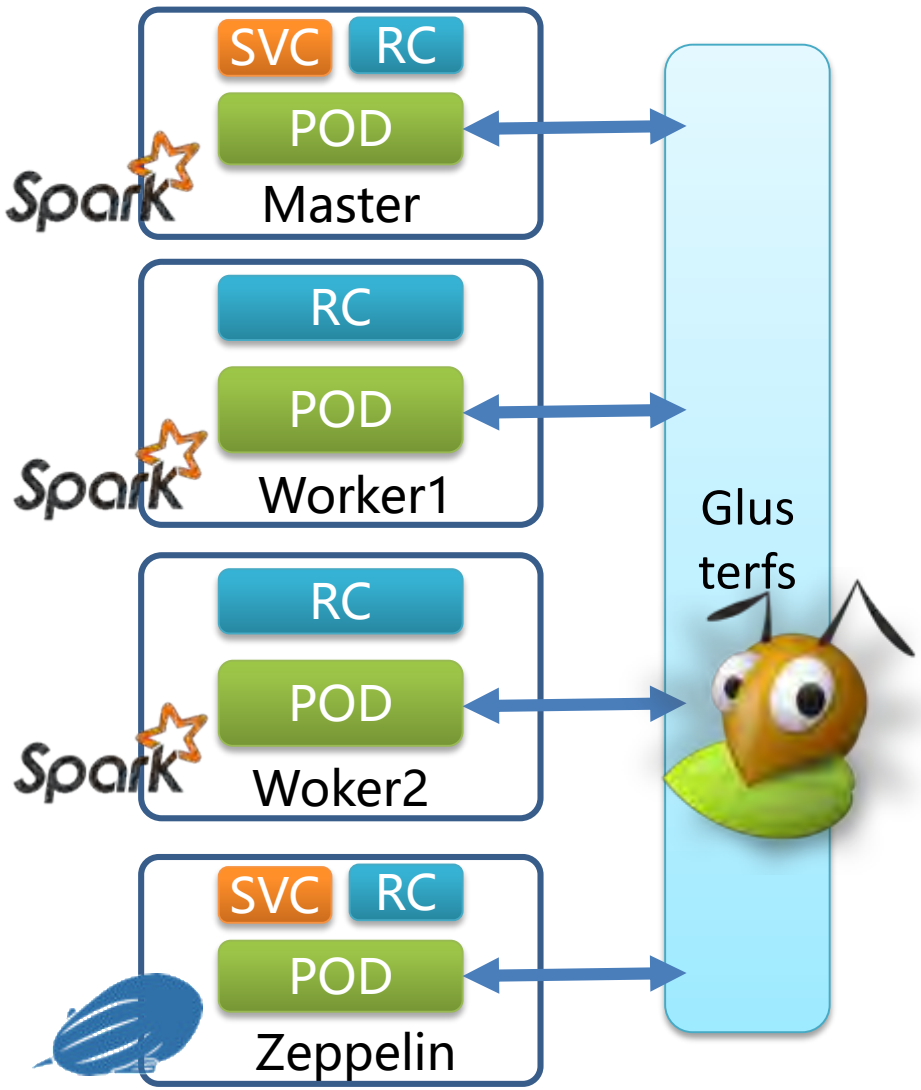
```
        name: sb-#cluster_uid#-cassandra
        ports:
          - containerPort: 9042
            name: cql
          - containerPort: 9160
            name: thrift
        volumeMounts:
          - mountPath: /cassandra_data
            name: data
          - mountPath: /cassandra_config
            name: cassandra-config
    volumes:
      - name: data
        emptyDir: {}
      - name: cassandra-config
        configMap:
          name: cassandra-config
```

大数据服务的数据持久化

- pv
  - NFS, GlusterFS, Ceph RBD, OpenStack Cinder, AWS EBS,GCE Persistent Disk, iSCSI, and Fibre Channel
- heketi
  - https://github.com/heketi/heketi
  - 基于Go语言和Restful接口的GlusterFS卷管理框架
    - GlusterFS集群管理
    - GlusterFS设备管理
    - GlusterFS卷管理
  - 支持与kubernetes集群融合部署



大数据服务的数据持久化



pv.yaml

```
spec:
  accessModes:
  - ReadWriteMany
  capacity:
    storage: 10Gi
  glusterfs:
    endpoints: glusterfs-cluster
    path: vol_949bd3002319b2abeae540b73e09f054
  persistentVolumeReclaimPolicy: Retain
```

pvc.yaml

```
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

rc.yaml

```
  volumeMounts:
  - mountPath: /spark_data
    name: data
dnsPolicy: ClusterFirst
restartPolicy: Always
securityContext: {}
terminationGracePeriodSeconds: 30
volumes:
- name: data
  persistentVolumeClaim:
    claimName: vol-10gi-1
```

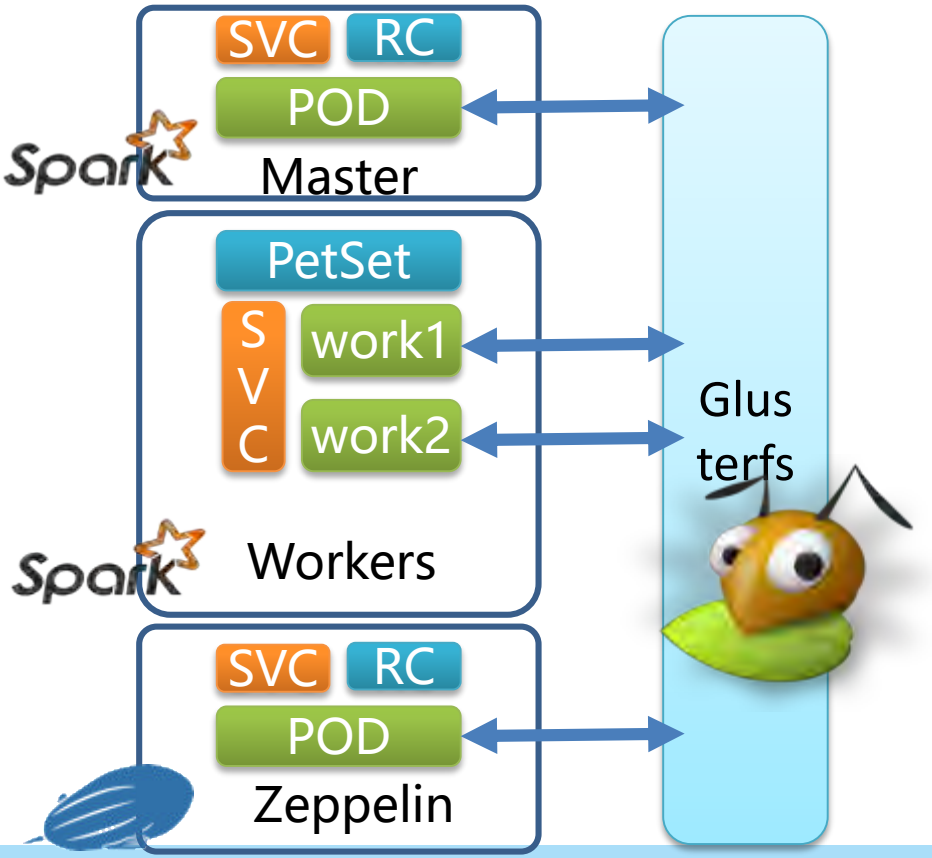
大数据服务的数据持久化

heketi目前的问题

- 元数据单点
  - 基于文件同步的双活互备
- 管理效率略低，5-10s
- 异常处理能力略显不足

使用PetSet来简化部署模型

- PetSet/StatefulSet特点
  - 固定的POD名称（主机名）
  - 按序编号
  - 映射存储
  - 只能更新副本数
  - 手工更新镜像
  - 删除PetSet后需手工删除pvc



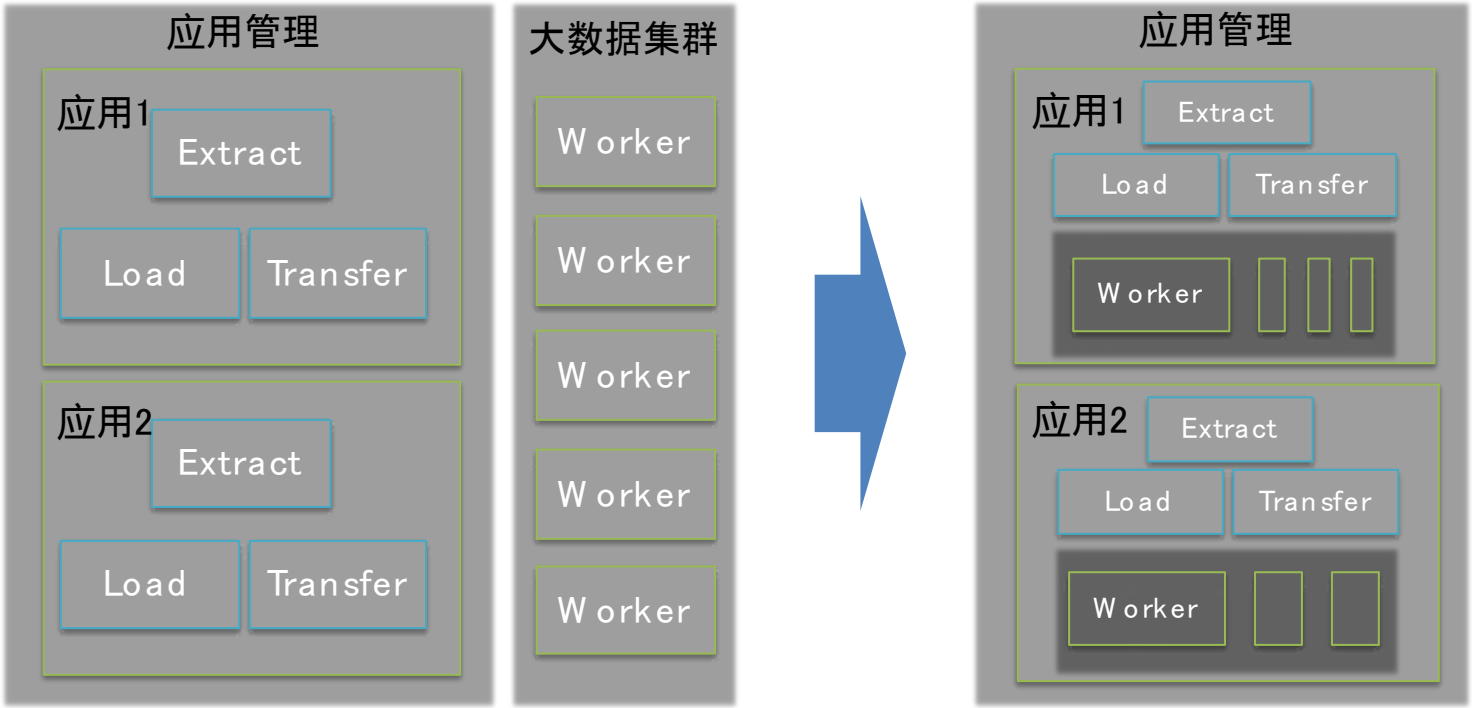
petset.yaml

```yaml
apiVersion: "apps/v1beta1"
kind: StatefulSet
metadata:
  name: spark-worker
spec:
  serviceName: spark-worker
  replicas: 3
  template:
    metadata:
      annotations:
        pod.alpha.kubernetes.io/initialized: "true"
      labels:
        app: spark-worker
    spec:
      containers:
      ........
  volumeClaimTemplates:
  - metadata:
      name: spark-data
      annotations:
        volume.alpha.kubernetes.io/storage-class: anything
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

- 当前大数据服务供给的难点
- 如何更好的供给大数据服务
- 基于K8s的大数据服务供给探索
- 效果及存在问题

新的大数据服务供给模式

- 为每个大数据应用分配一个大数据服务
  - 高效
  - 可控



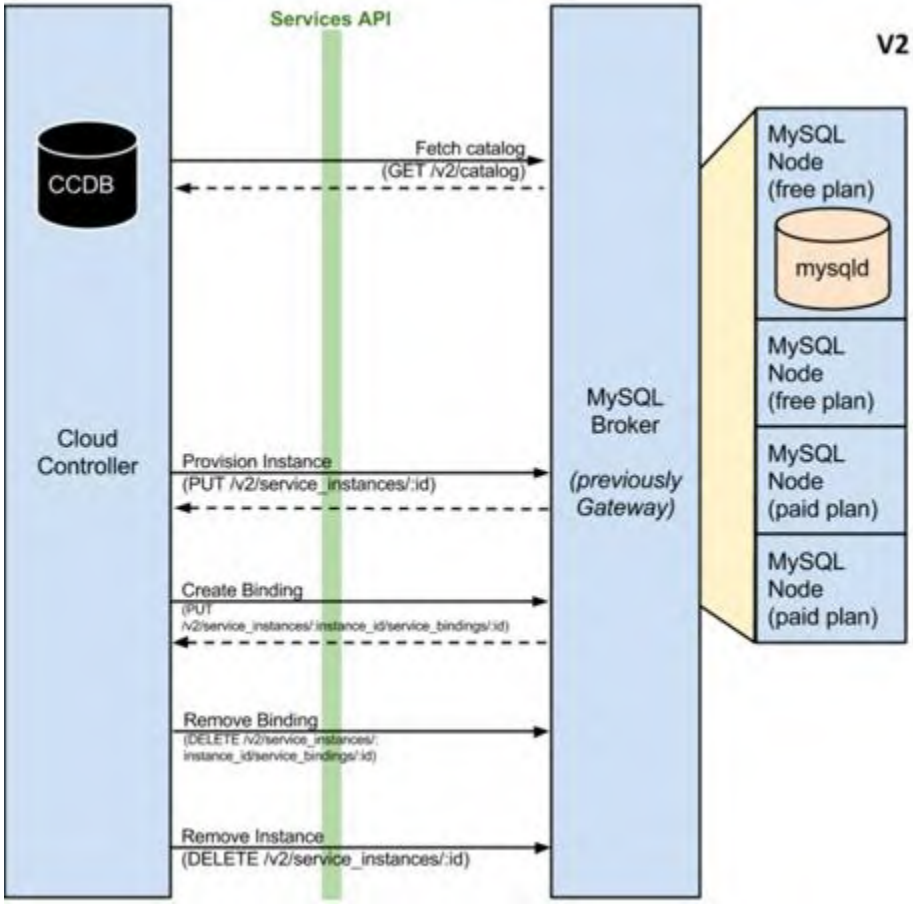
新的大数据服务供给模式

- 与Spark on Yarn比较

	Spark on K8s	Spark on Yarn
集群资源	通用	专属
融合编排	是	否
资源隔离	cpu、内存	cpu、内存
多租户	简易	一般
本地计算	不支持	支持
适宜场景	无状态计算、流处理	存储型批处理
运维技术环境	统一	分裂

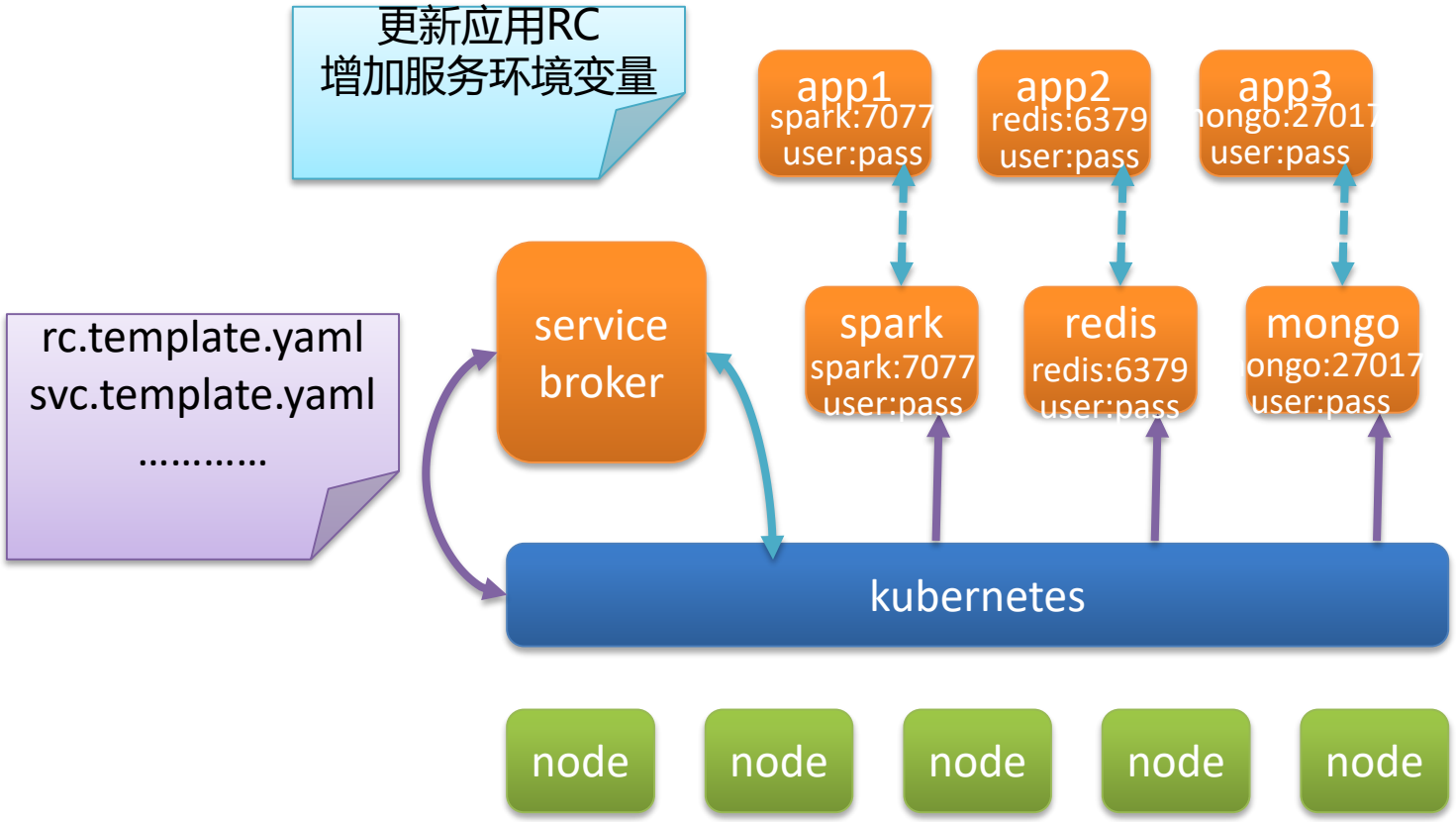
用ServiceBroker来快速交付大数据服务

- ServiceBroker
  - Cloudfoundry中后端服务与应用集成协议
  - http://docs.cloudfoundry.org/services/api.html



- ✓ 业界事实标准，降低集成难度，复用开源实现，降低迁移难度。
- ✓ 以清晰的边界实现服务自治，实现了专业化地提供服务和便利地使用服务的分工。
- ✓ 以优雅的方式实现了大数据能力的虚拟化，提高资源使用率。
- ✓ 控制流和数据流分离，服务和服务可以打通，更加适合多样化大数据能力的集成。

用ServiceBroker来快速交付大数据服务



基于容器的大数据服务供给还存在的问题

- 问题
  - 个别大数据服务编排困难
    - redis3.0 sharding
    - mongo sharding+replica
  - overlay网络性能损耗较大
  - kubelet串行完成镜像拉取
  - 容器用户与glusterfs卷属主不一致
  - 容器日志处理
- 测试、PoC环境使用

谢谢

DataFoundry

lab.dataos.io

极速铸造云端应用 释放大数据之美

扫描二维码　关注更多精彩