Kingdee

星辰微服务架构

陈兵

2021.4.7

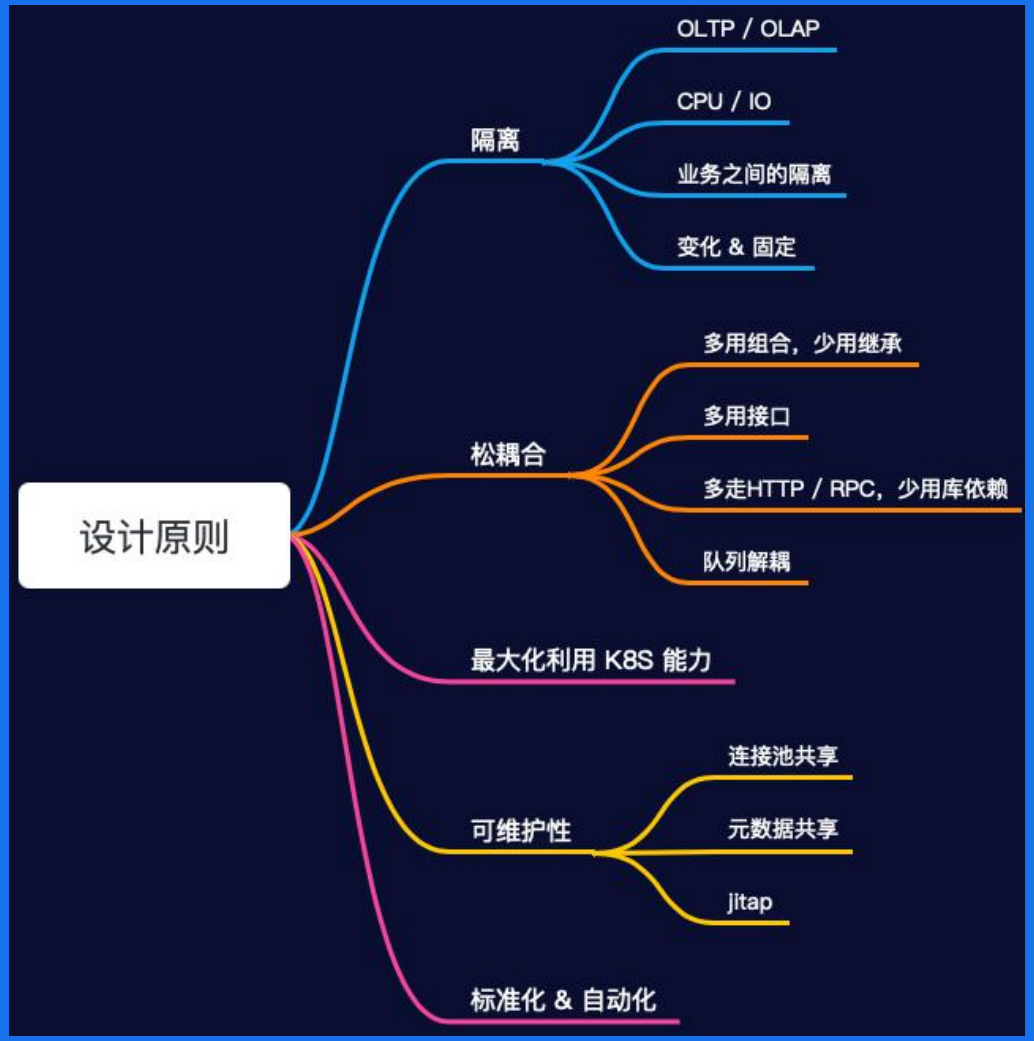
提纲

- 架构设计目标
- 容器 & K8S 知识回顾
- 数据架构
- 部署架构

设计目标

1. 高可用
2. 高性能
3. 可水平扩展

设计原则


设计原则
隔离
OLTP / OLAP
CPU / IO
业务之间的隔离
变化 & 固定
松耦合
多用组合，少用继承
多用接口
多走HTTP / RPC，少用库依赖
队列解耦
最大化利用 K8S 能力
可维护性
连接池共享
元数据共享
jitap
标准化 & 自动化

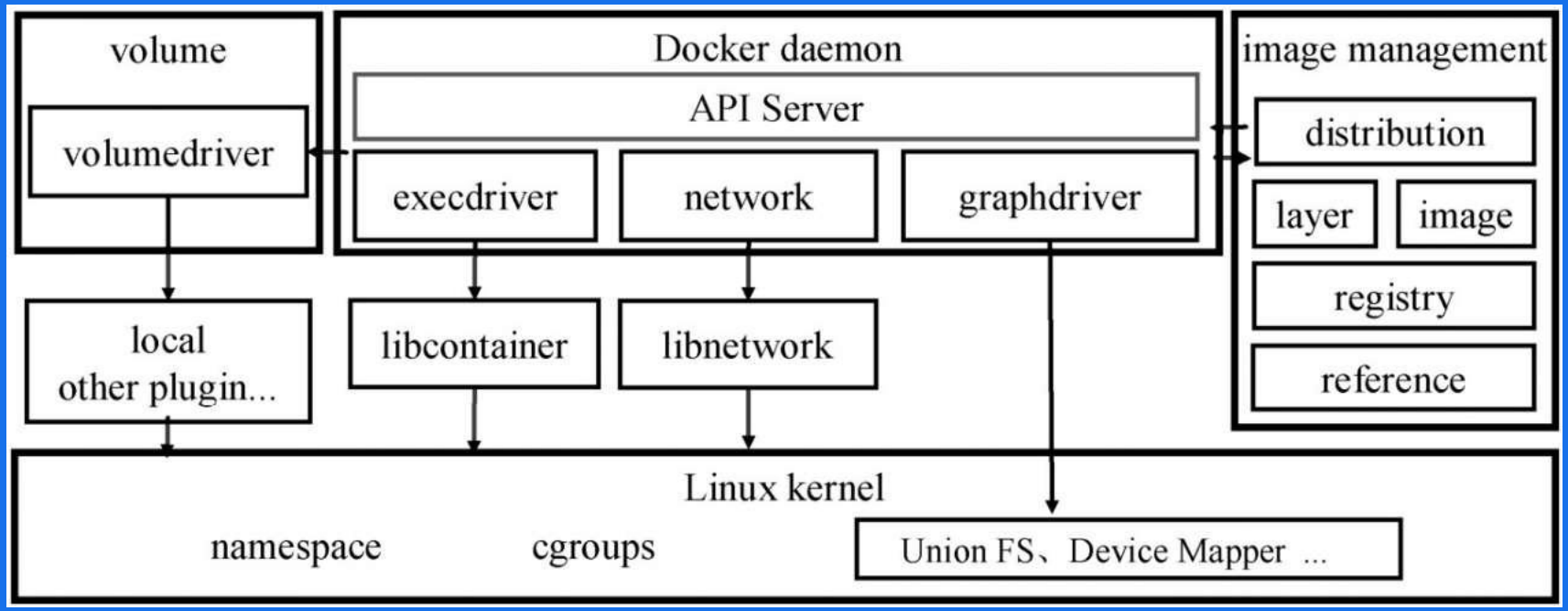
提纲

- 架构设计目标
- 容器 & K8S 知识回顾
- 数据架构
- 部署架构

docker——why

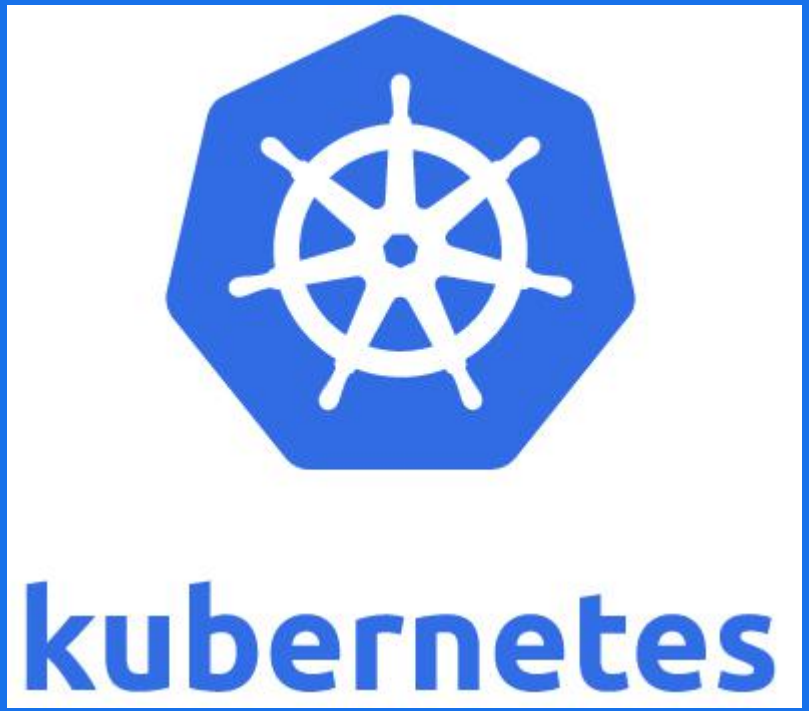
1. 标准化：打包、部署、依赖
2. 装更多的箱子：没有缝隙
3. 自动化：作业

docker——整体架构



docker 三板斧：namespace、cgroups、UFS/DeviceMapper。

kubernetes——是什么



Kubernetes = Kubernetes = K8S

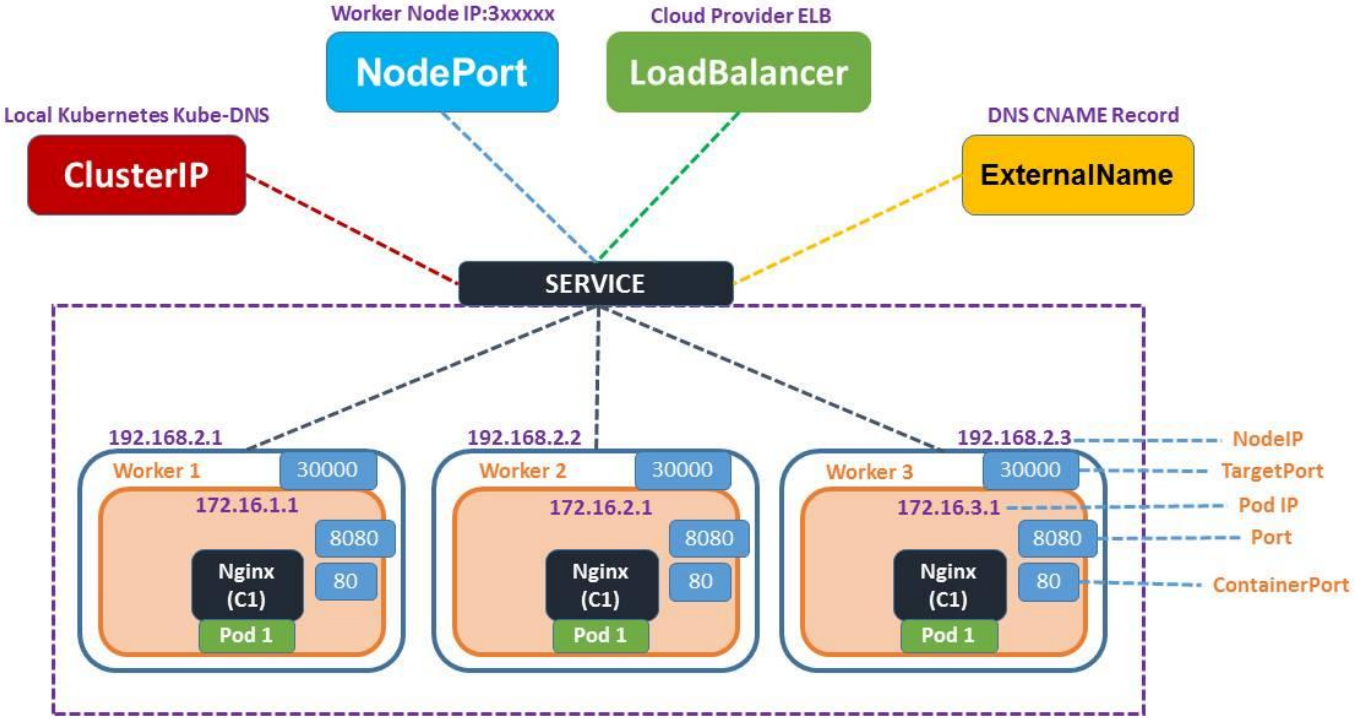
kubernetes——是什么

- 首先，它是一个全新的基于容器技术的分布式架构领先框架。K8S是谷歌严格保密十几年的秘密武器——Borg的一个开源版本。它基于容器技术，目的是实习资源管理的自动化，以及跨多个数据中心的资源利用率最大化。
- 其次，如果我们的系统设计遵循了K8S的设计思想，那么传统系统架构中的那些和业务没有多大关系的底层代码或功能模块，都可以立刻从我们的视线中消失。我们不必再费心于负载均衡、部署实施、服务治理、服务监控和故障处理这些琐事了。
- 然后，K8S是一个开放的开发平台。由于K8S对现有的编程语言、编程框架、中间件没有任何侵入性，因此现有的系统也很容易改造升级并迁移到K8S平台上。
- 最后，K8S是一个完备的分布式系统支撑平台。

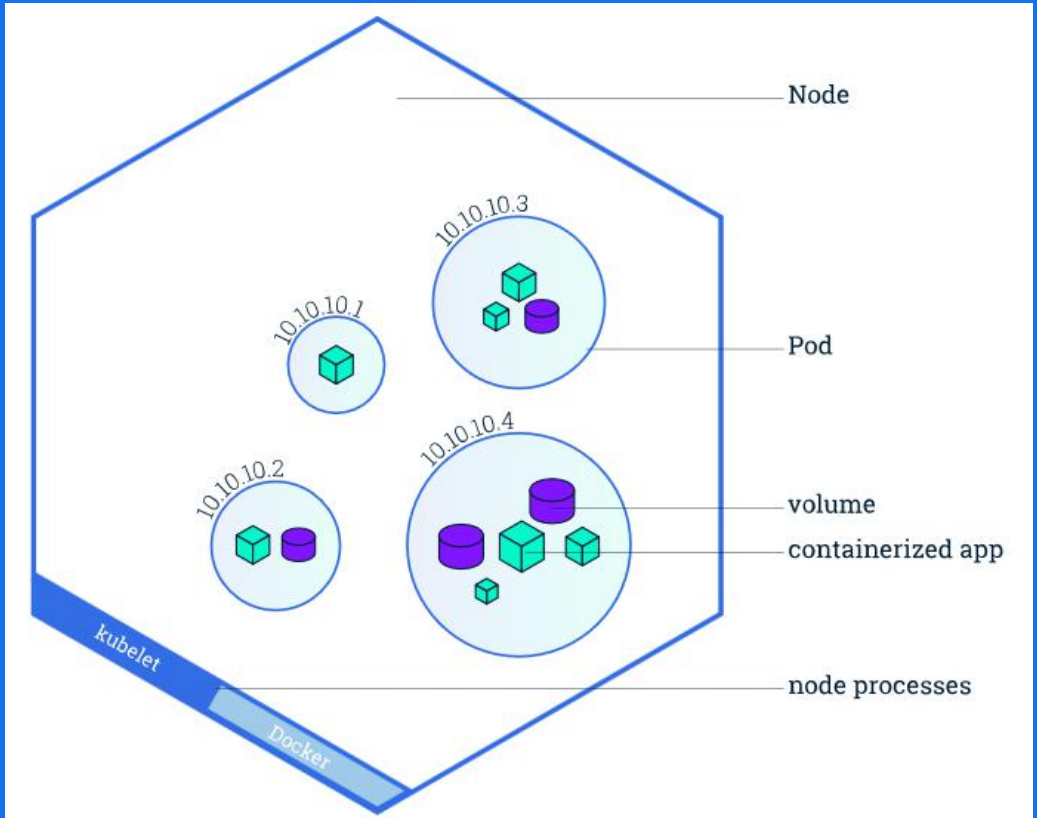
kubernetes——为什么要用

- 首先，我们可以“轻装上阵”地开发复杂系统了。
- 其次，使用K8S就是在全面拥抱微服务架构。微服务架构的核心是将一个巨大的单体应用分解为很多小的互相连接的微服务，一个微服务背后可能有多个实例副本在支撑，副本的数量可能会随着系统的负荷变化而进行调整，内嵌的负载均衡器在这里发挥了重要作用。每个微服务独立开发、升级、扩展，因此系统具备很高的稳定性和快速迭代进化能力。
- 然后，K8S可以兼顾公有云和私有云。在K8S的架构方案中，底层网络的细节完全被屏蔽，基于服务的Cluster IP甚至都无须我们改变运行时的配置文件，就能将系统从物理机环境无缝迁移到公有云中，或者在服务高峰期将部分服务对应的Pod副本放入公有云以提升系统的吞吐量，不仅节省了公司的硬件投入，还大大改善了客户体验。
- 最好，K8S系统架构具备了超强的横向扩容能力。不用改代码，一个K8S集群即可从只包含几个Node的小集群平滑扩展到拥有上千个Node的大规模集群。

kubernetes——Service

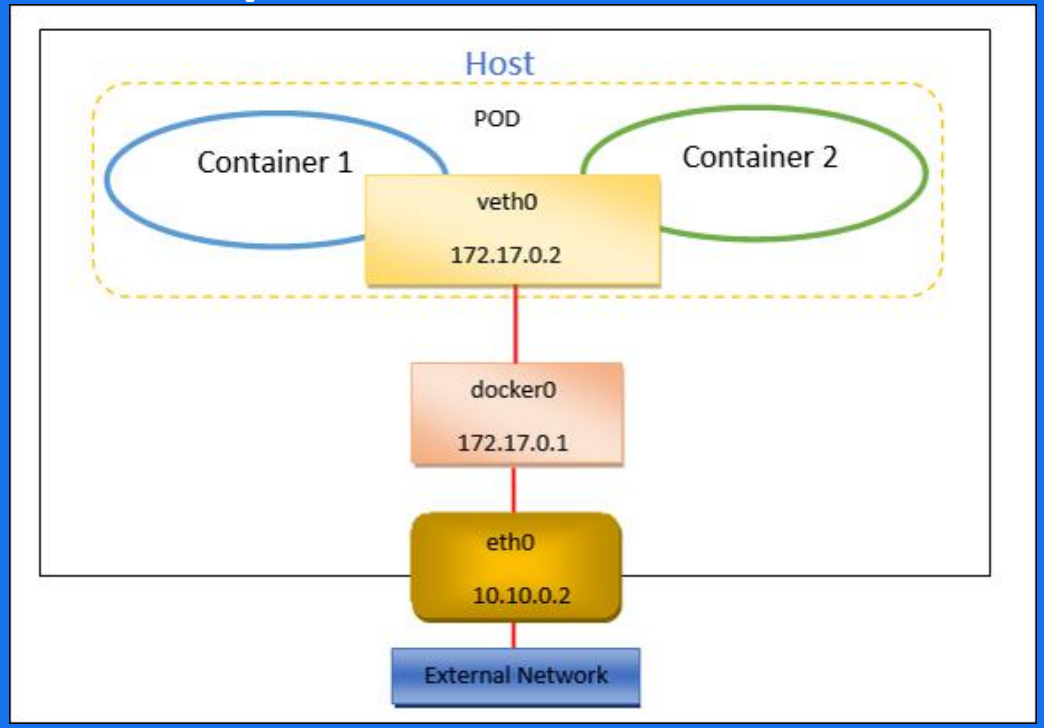


kubernetes——node



Node就是我们常说的虚拟机或物理机。

kubernetes——pod



Pod是K8S的最重要也最基本的概念。每个Pod都有一个特殊的被称为“根容器”的Pause容器，其他容器则为业务容器，这些业务容器共享Pause容器的网络栈和Volume挂载卷，因此它们之间的通信和数据交换更为高效，在设计时我们可以充分利用这一特性将一组密切相关的服务进程放入同一个Pod中。

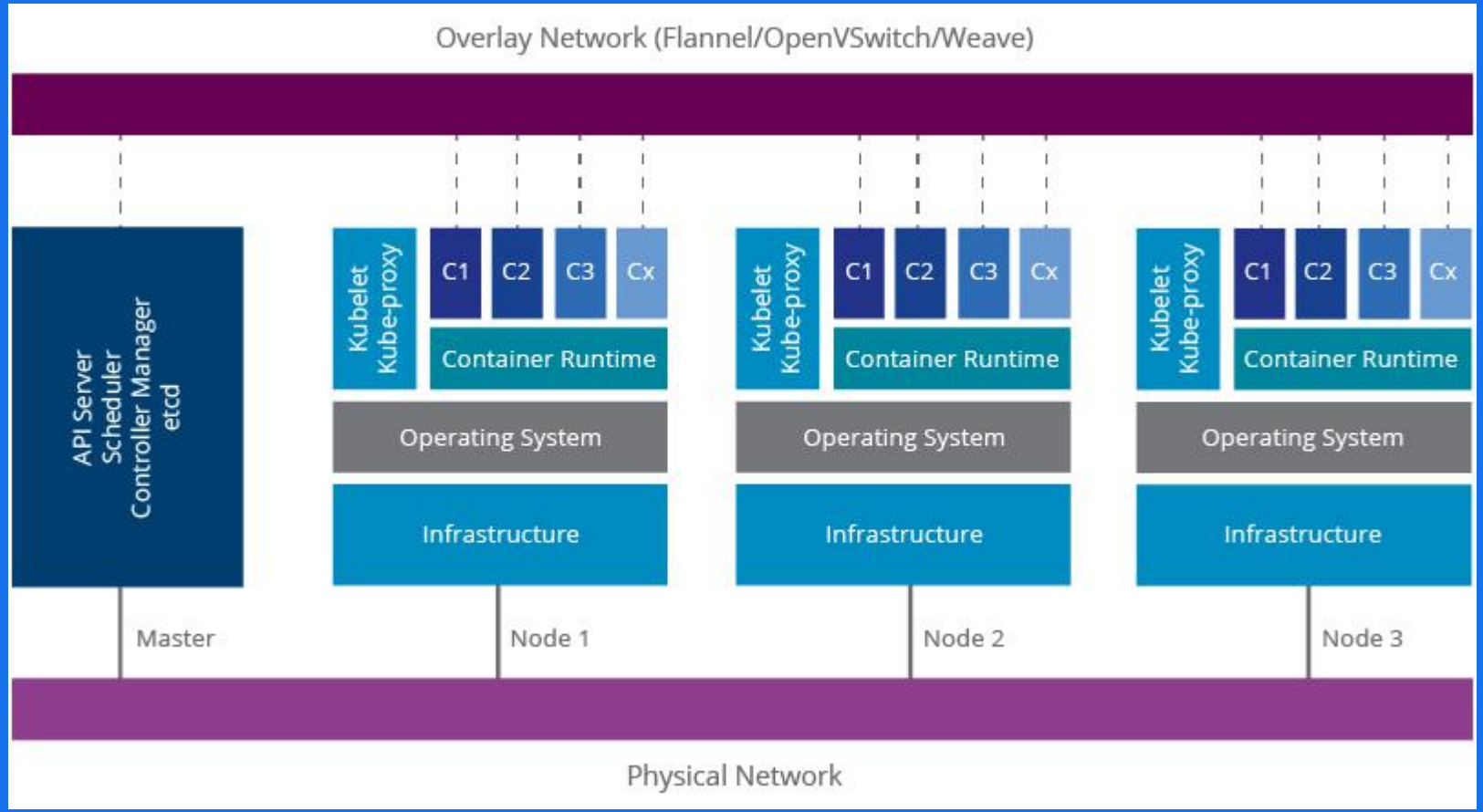
kubernetes——为什么要有Pod

原因之一：在一组容器作为一个单元的情况下，我们难以对“整体”简单地进行判断及有效地进行行动。比如，一个容器死亡了，此时算是整体死亡吗？是N/M的死亡率吗？引入业务无关且不易死亡的Pause容器作用Pod的根容器，以它的状态代表整个容器组的状态，就简单、巧妙地解决了这个问题。

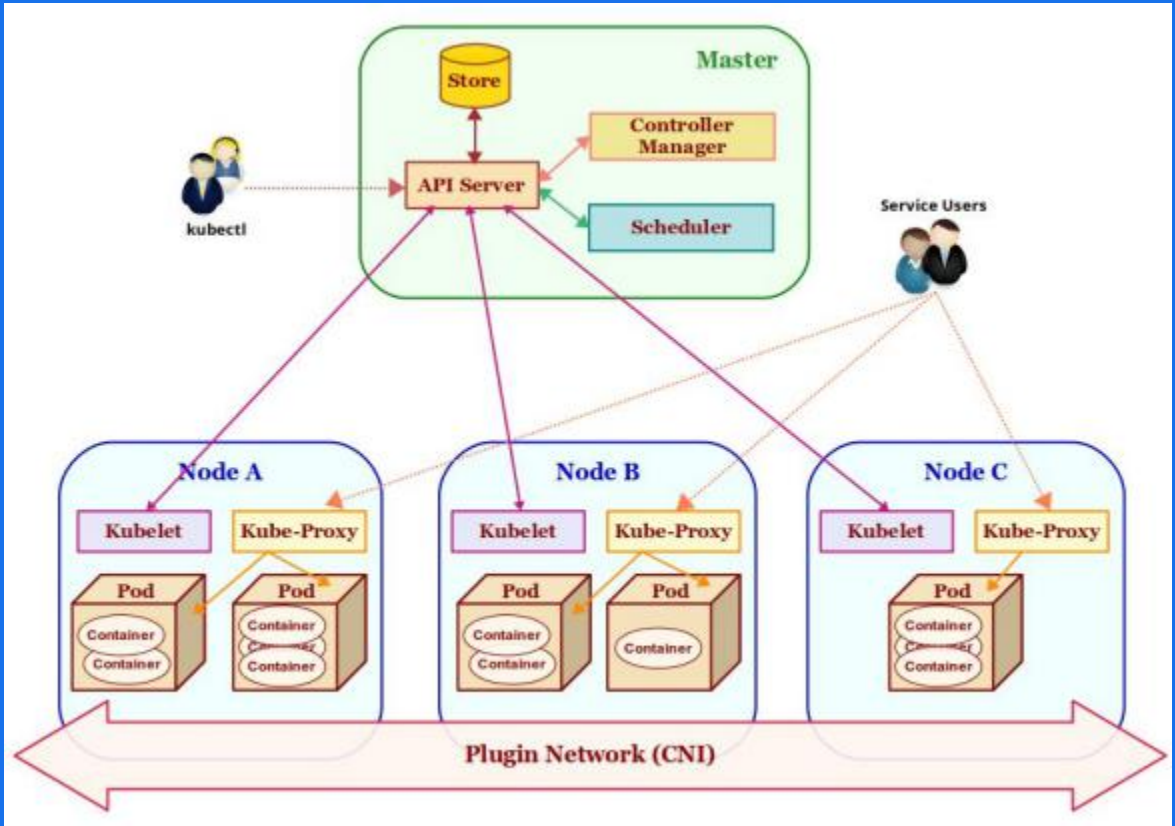
原因之二：Pod里多个业务容器共享Pause容器的IP，共享Pause容器挂接的Volume，这样既简化了密切联系的业务容器之间的通信问题，也很好地解决了它们之间的文件共享问题。

原因之三：Pod对接的是CFI（container foundation interface），摆脱对具体容器的依赖。

kubernetes——架构



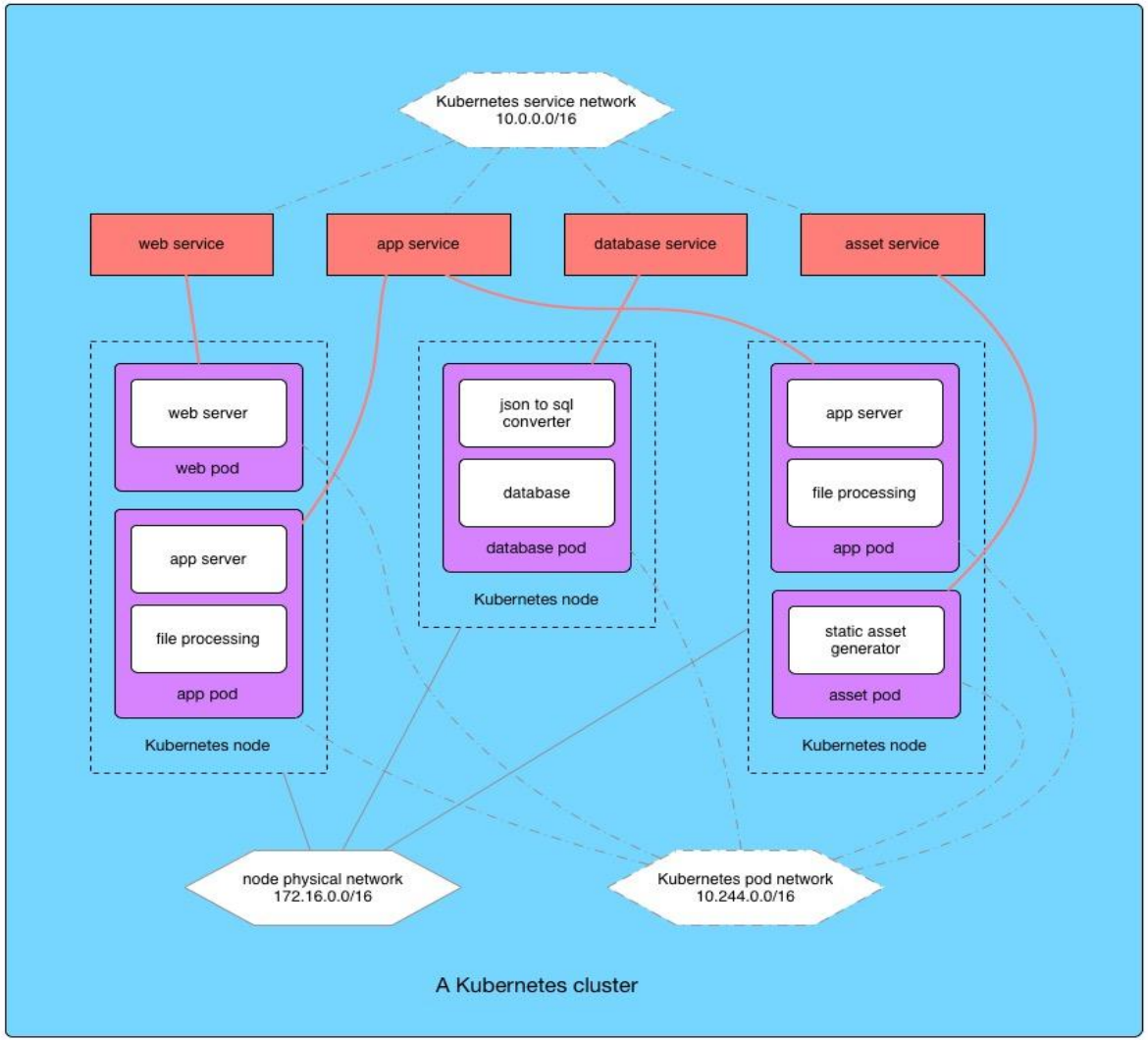
kubernetes——架构



kubernetes——架构

- kube-apiserver：提供了K8S各类资源对象（Pod、RC、Service等）的增删改查及Watch等HTTP Rest接口，成为集群各个功能模块之间数据交互和通信的中心枢纽，是整个系统的数据总线和数据中心。
- ReplicationController：作为集群内部的管理控制中心，负责集群内的Node、Pod副本、Endpoint、命名空间、服务账号、资源配额等的管理，当某个Node意外宕机时，它会及时发现此故障并执行自动化修复流程，确保集群始终处于预期的工作状态。
- scheduler：在整个系统中承担了“承上启下”的重要功能，“乘上”是指它负责接收ReplicationController创建的新Pod，为其安排一个落脚的“家”——目标Node；“启下”是指安置工作完成后，目标Node上的kubelet服务进程接管后继工作。
- kube-proxy：运行在每个Node上的，其实就是一个智能的负载均衡器，它负责把对Service的请求转发到后端的某个Pod实例上，并在内部实现服务的负载均衡与会话保持机制。
- kubelet：用于处理Master节点下发到本节点的任务，管理Pod及Pod中的容器。

kubernetes

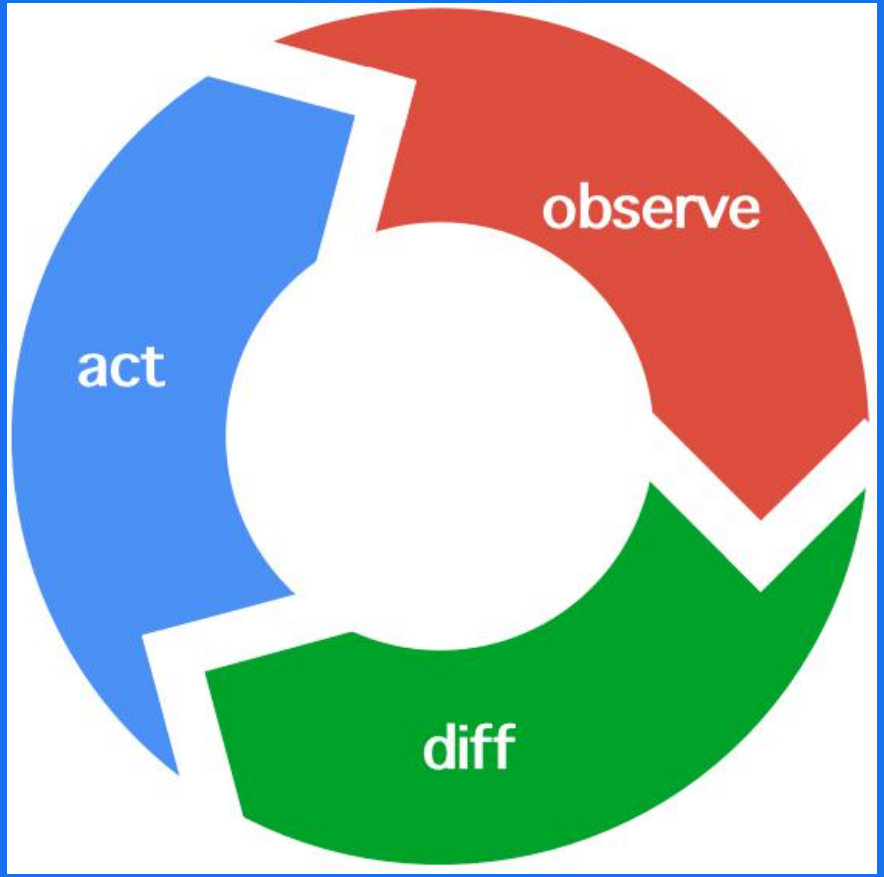


Kingdee

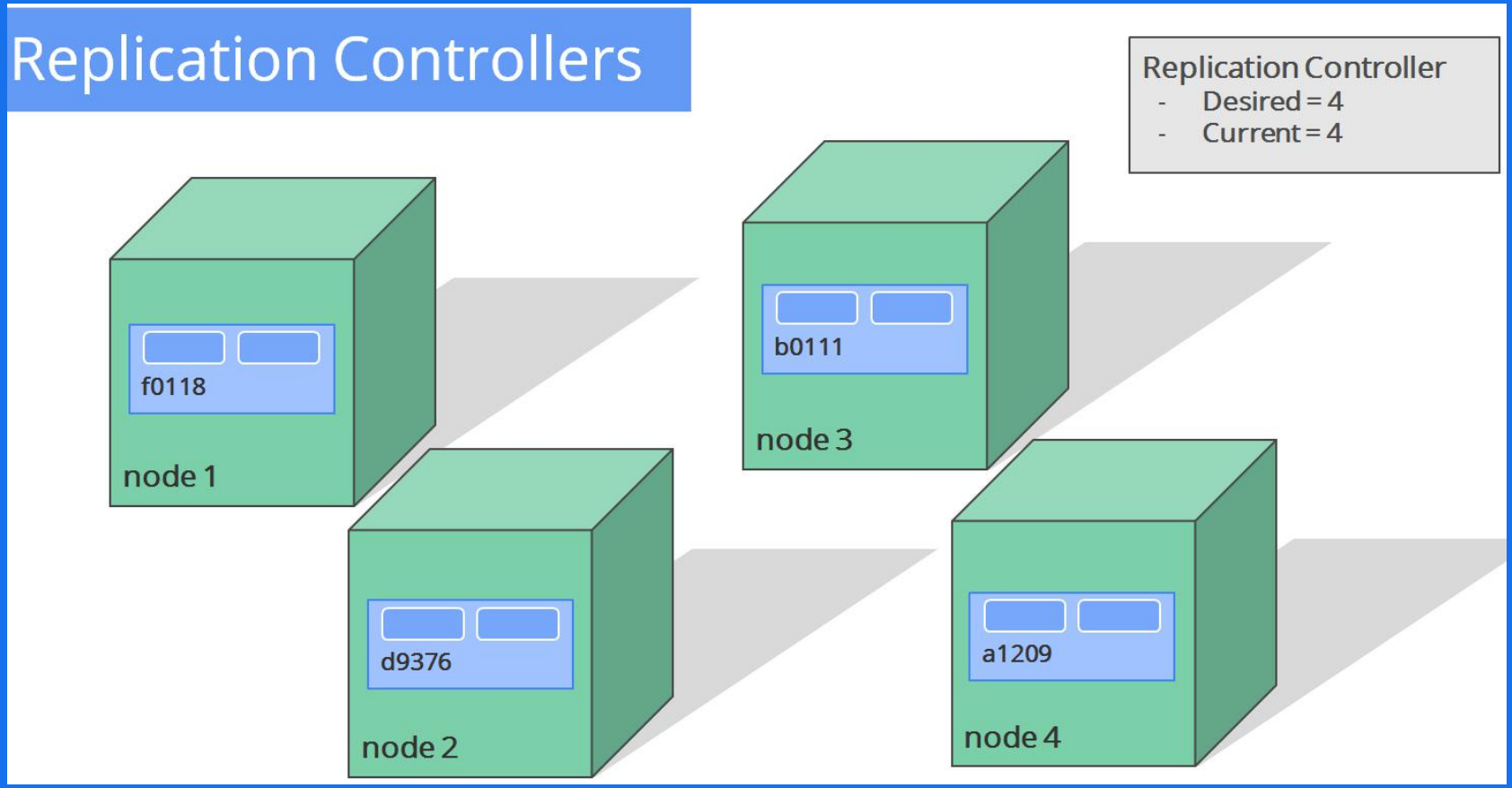
kubernetes——容器编排

控制流程：

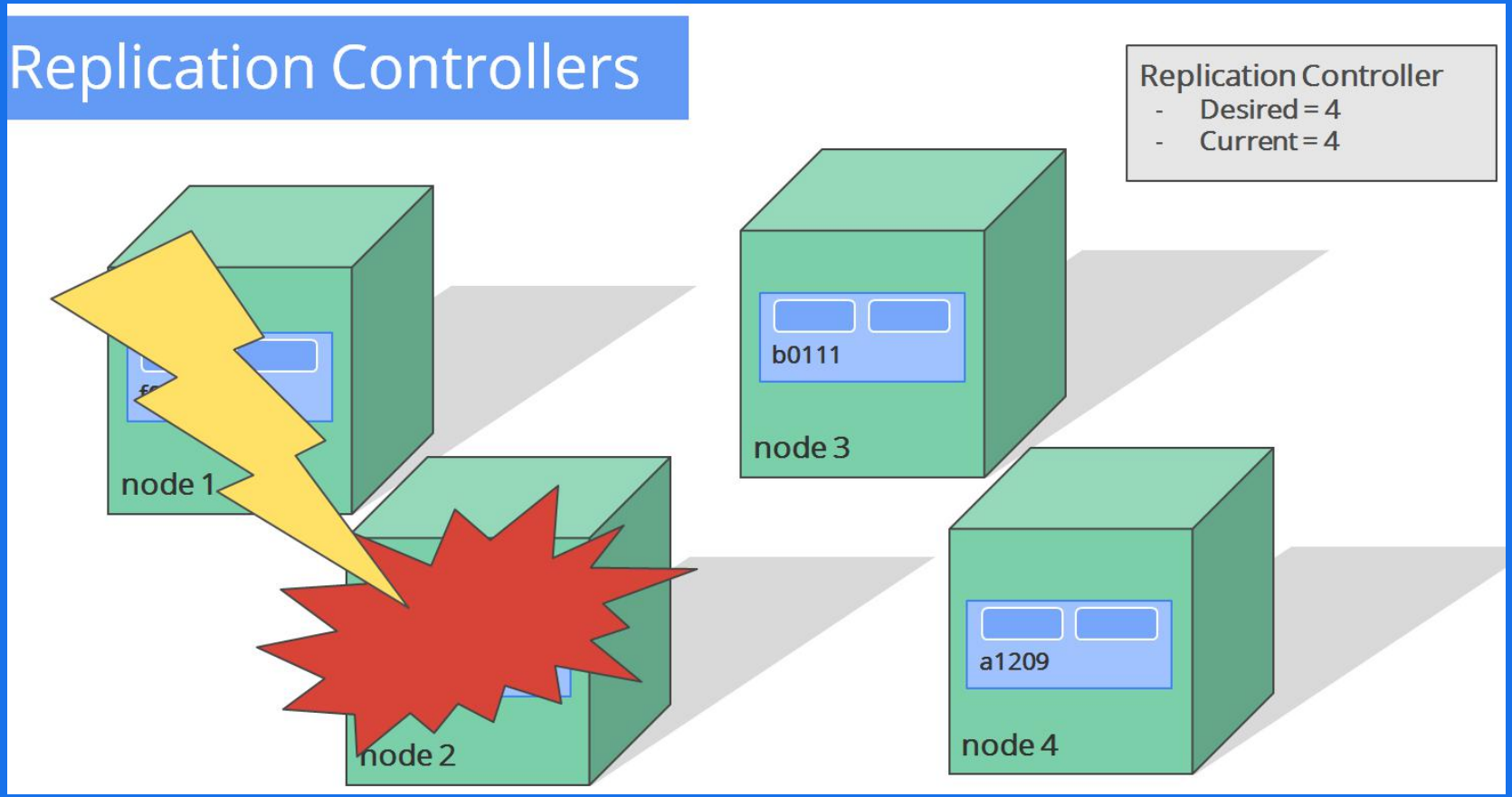
- 驱动当前状态 -> 目标状态
- Act independently
- Observed state is truth
- Recurring pattern in the system



kubernetes——容器编排

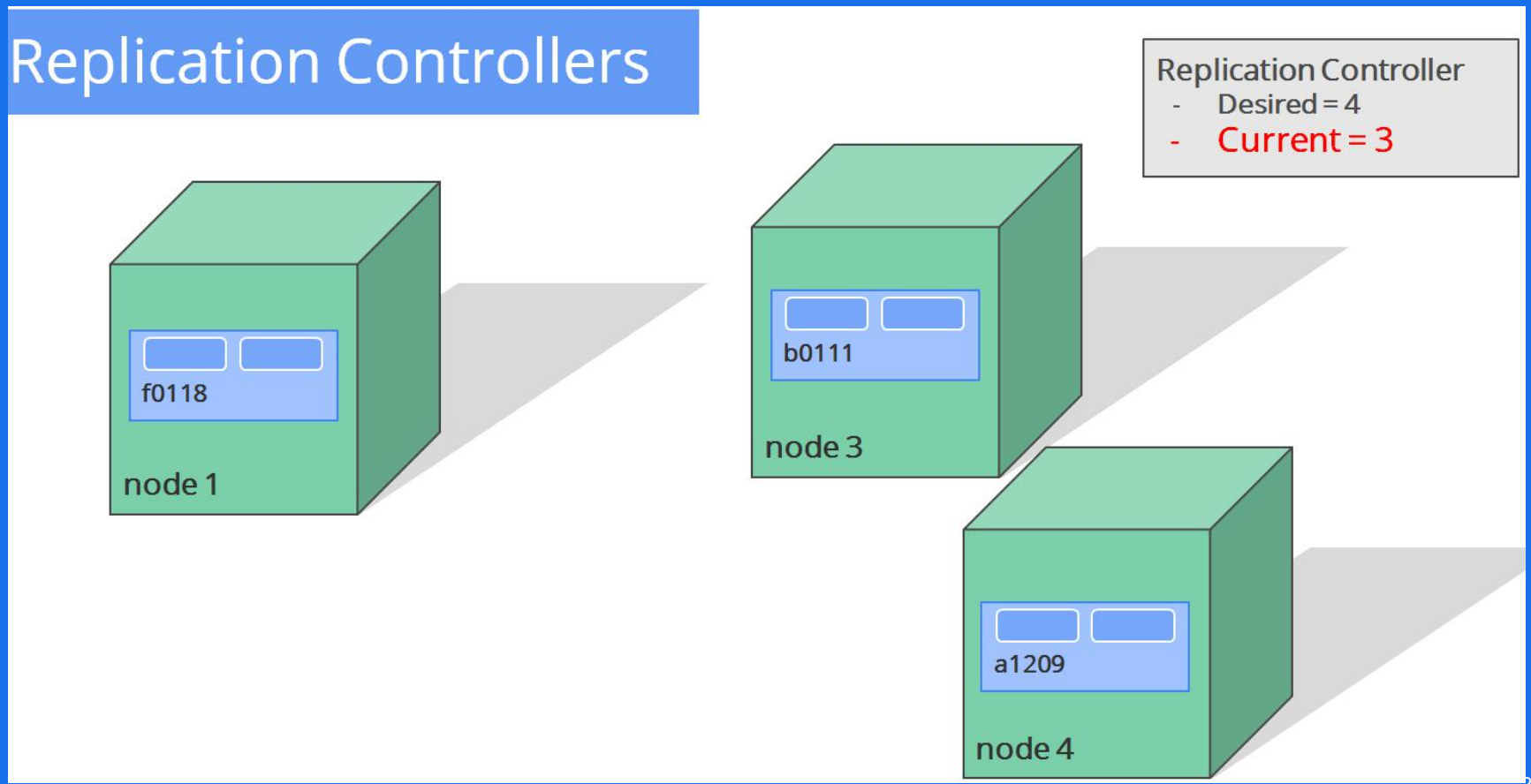


kubernetes——容器编排

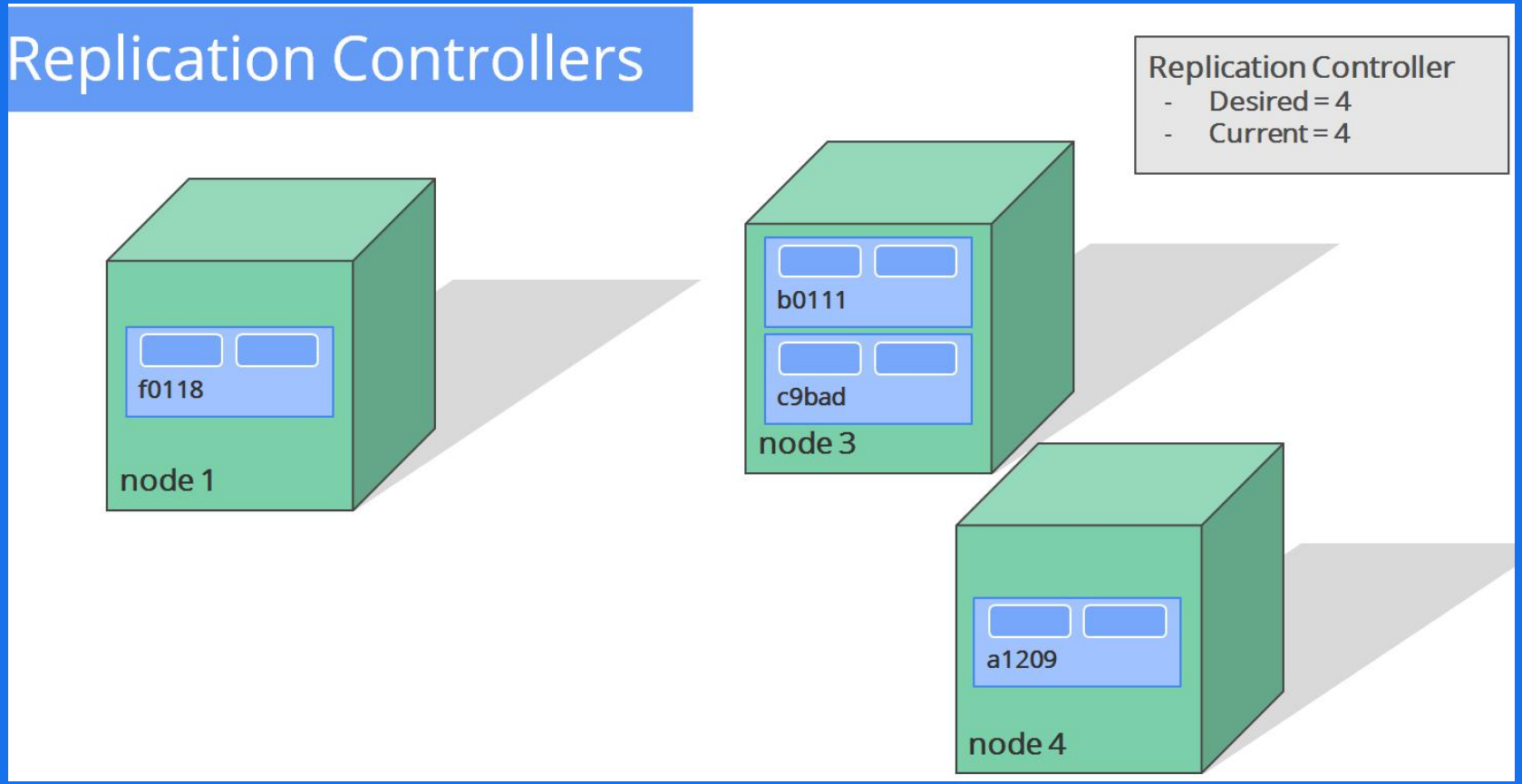


kubernetes——容器编排

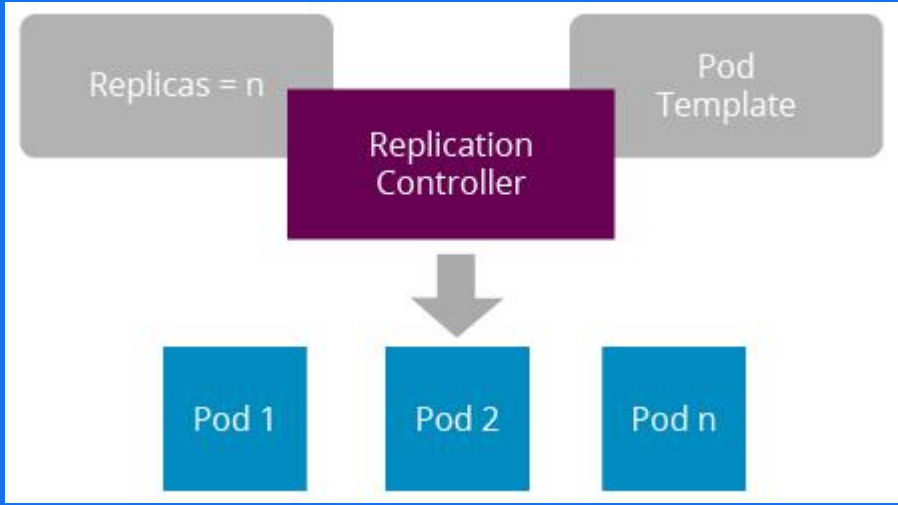
Replication Controllers



kubernetes——容器编排



kubernetes——容器编排



容器编排的工作主要由 Replication Controller完成，以下简称 RC。

RC 定义了一个期望的场景，即声明某种Pod的副本数量在任意时刻都符合某个预期值，所以RC的定义包含如下几个部分：

1. Pod期待的副本数；
2. 创建Pod的模板(包含了docker镜像、资源需求、环境变量等)；

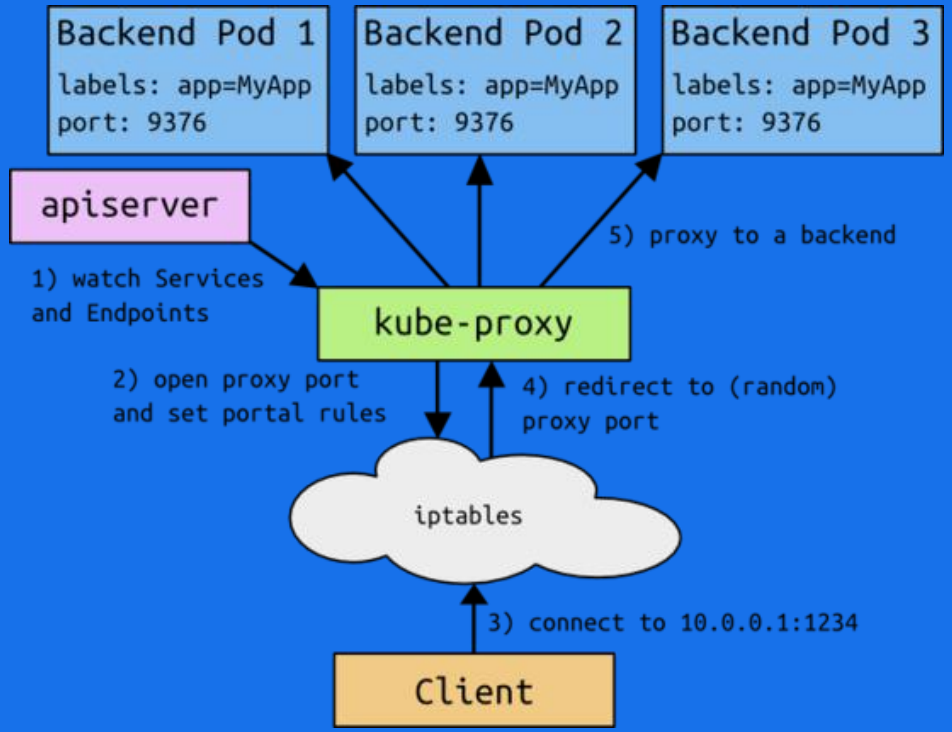
kubernetes——容器编排

```yaml
spec:
  replicas: 2
  selector:
    k8s-app: elasticsearch-logging
    version: v1
  template:
    metadata:
      labels:
        k8s-app: elasticsearch-logging
        version: v1
        kubernetes.io/cluster-service: "true"
    spec:
      serviceAccountName: efk
      containers:
      - name: elasticsearch-logging
        image: registry.kingdee.com/elasticsearch:v2.4.1-2
        resources:
          # need more cpu upon initialization, therefore burstable class
          limits:
            cpu: 1000m
          requests:
            cpu: 100m
        ports:
        - containerPort: 9200
          name: db
          protocol: TCP
        - containerPort: 9300
          name: transport
          protocol: TCP
        volumeMounts:
        - name: es-persistent-storage
          mountPath: /data
```

副本数

模板

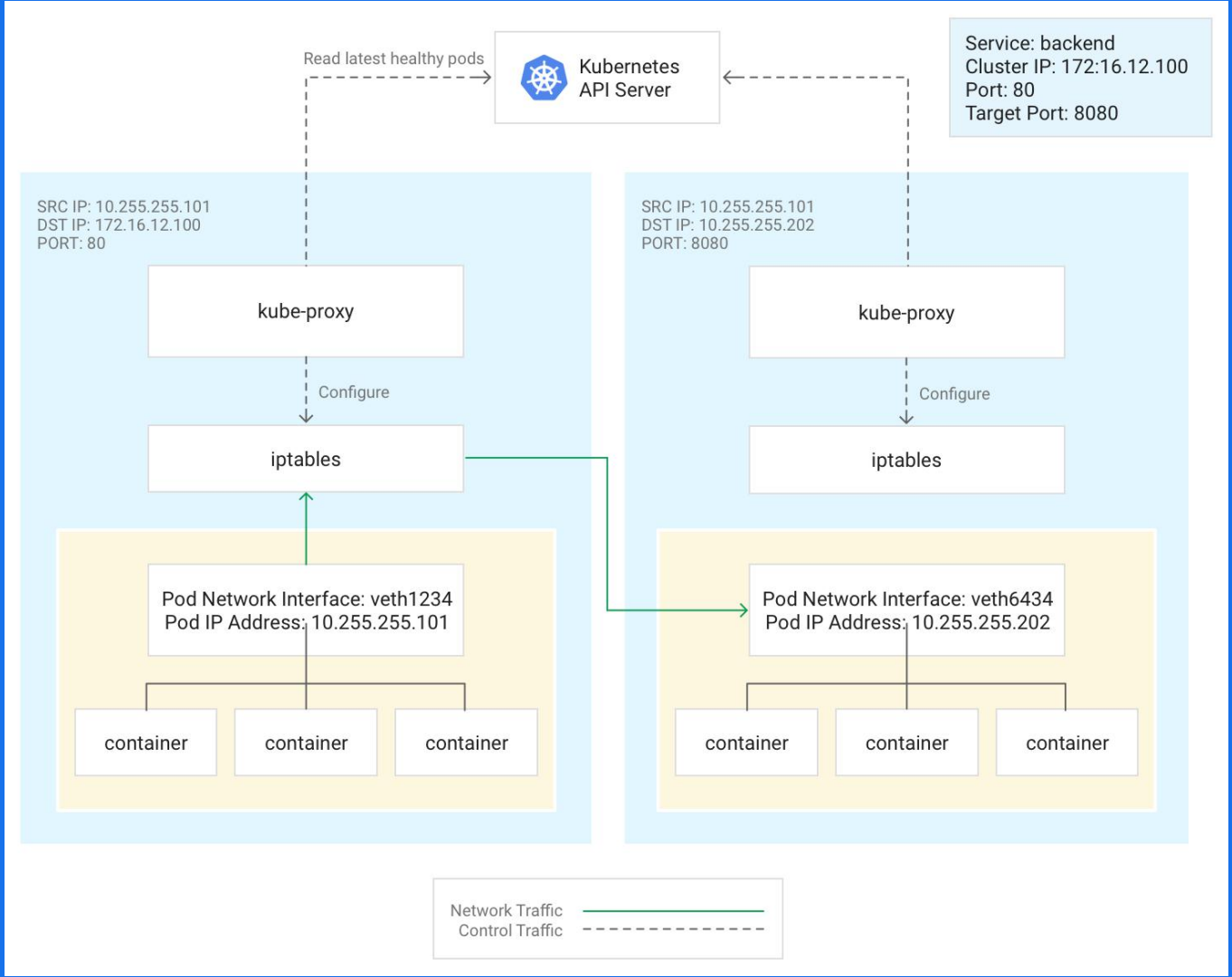
kubernetes——服务发现与负载均衡



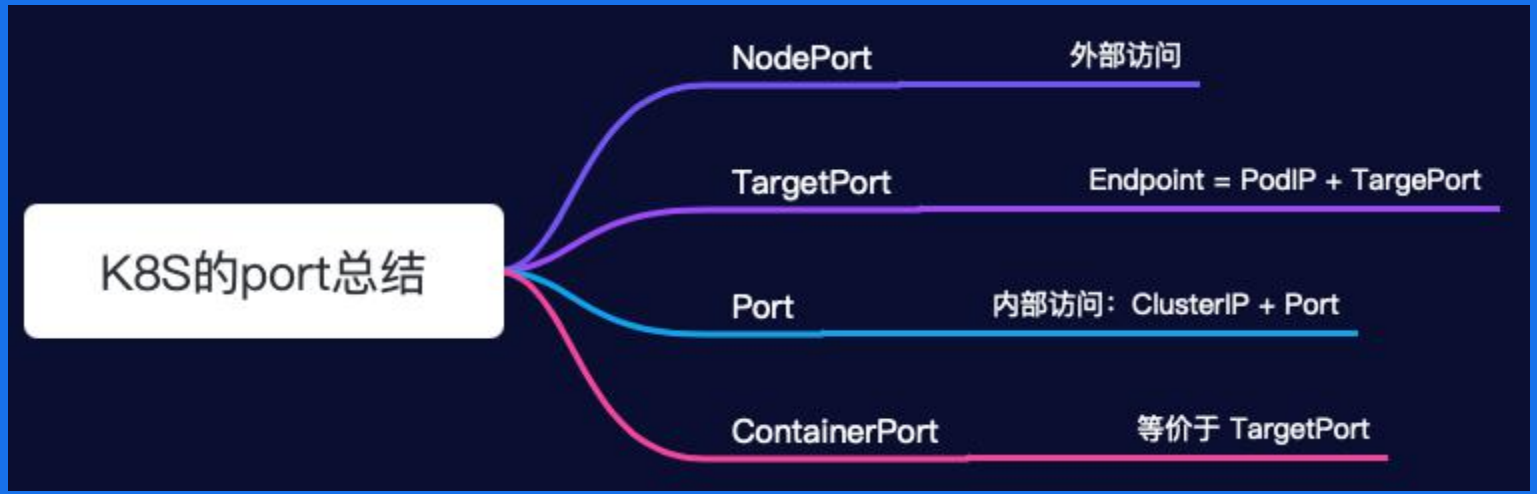
可以把Kube-proxy看作Service的透明代理兼负载均衡器，其核心功能是将到某个Service的访问请求转发到后端的多个Pod实例上。

Proxy为每个Service在本地打开一个随机选择的端口，Proxy决定哪个后台Pod被选定。任何访问该端口的连接将被代理到相应的某个后端Pod。核心技术：动态iptables NAT规则链。

kubernetes


Read latest healthy pods
Kubernetes API Server
Service: backend
Cluster IP: 172:16.12.100
Port: 80
Target Port: 8080
SRC IP: 10.255.255.101
DST IP: 172.16.12.100
PORT: 80
SRC IP: 10.255.255.101
DST IP: 10.255.255.202
PORT: 8080
kube-proxy
kube-proxy
Configure
Configure
iptables
iptables
Pod Network Interface: veth1234
Pod IP Address: 10.255.255.101
Pod Network Interface: veth6434
Pod IP Address: 10.255.255.202
container
container
container
container
container
container
Network Traffic
Control Traffic

kubernetes——网络术语总结



kubernetes——关于K8S的感悟

专注长期主义：

1. Linux
2. K8S
3. Java / JVM
4. 数据库
5. 数据结构 & 算法
6. 领域业务知识

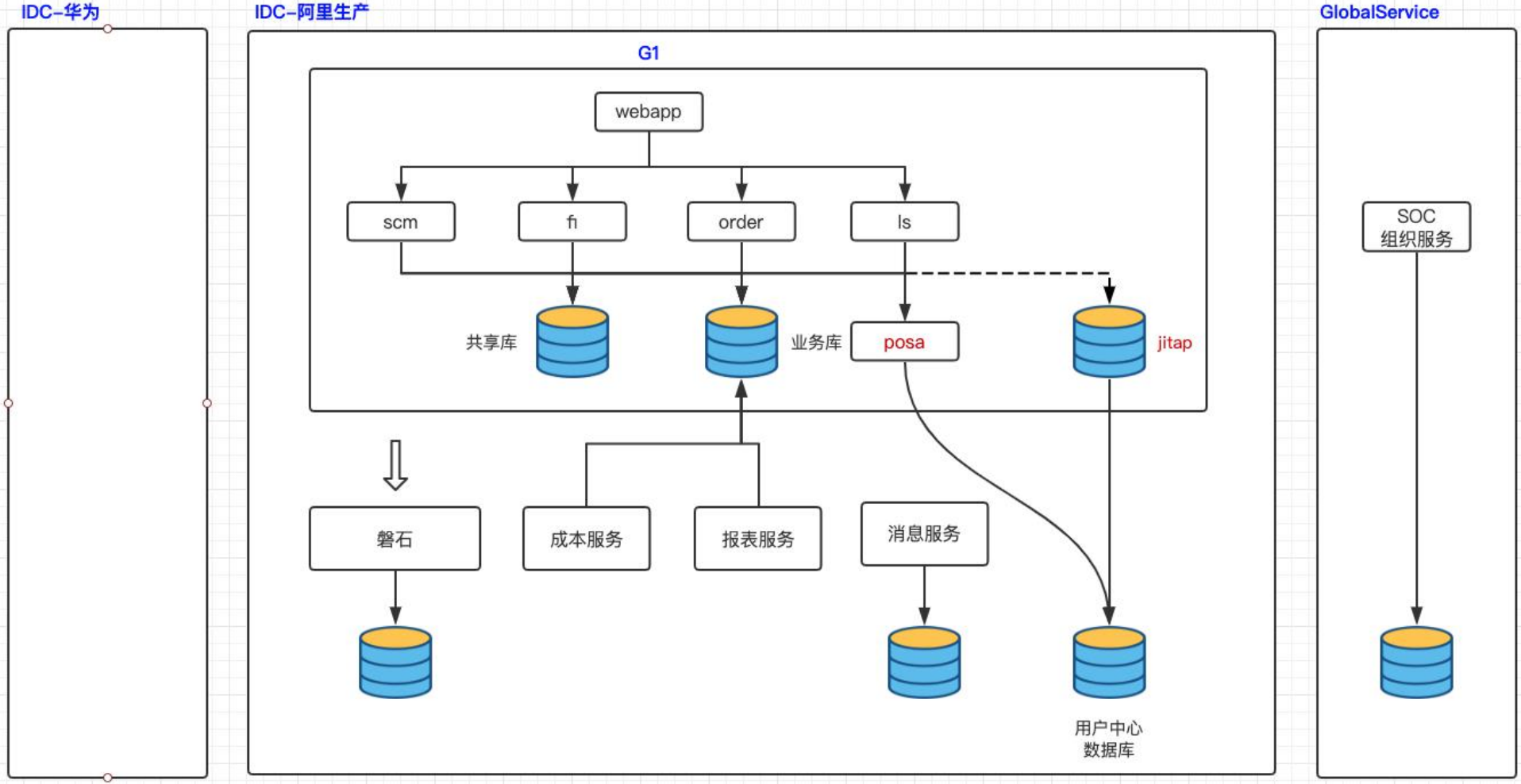
提纲

- 架构设计目标
- 容器 & K8S 知识回顾
- 数据架构
- 部署架构

数据架构

scope：

1. Global：组织
2. IDC：磐石
3. Group：标准元数据
4. ISV：二开元数据
5. 账套：私有元数据、业务数据

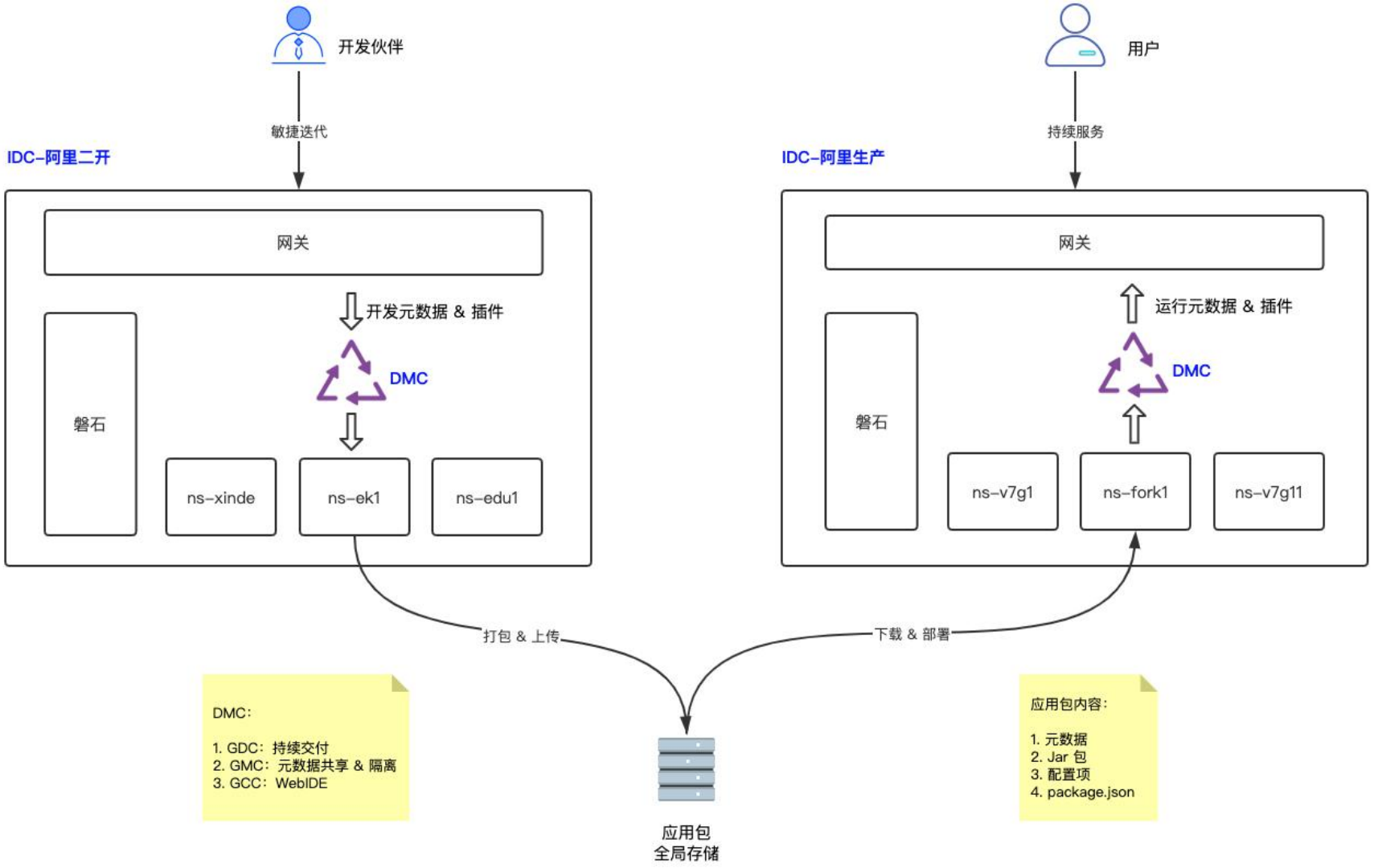

IDC-华为
IDC-阿里生产
GlobalService
G1
webapp
scm
fi
order
ls
共享库
业务库
posa
jitap
磐石
成本服务
报表服务
消息服务
用户中心
数据库
SOC
组织服务

提纲

- 架构设计目标
- 容器 & K8S 知识回顾
- 数据架构
- 部署架构

生产环境部署架构

星辰生产环境部署架构


开发伙伴
敏捷迭代
IDC-阿里二开
网关
开发元数据 & 插件
DMC
磐石
ns-xinde
ns-ek1
ns-edu1
用户
持续服务
IDC-阿里生产
网关
运行元数据 & 插件
DMC
磐石
ns-v7g1
ns-fork1
ns-v7g11
打包 & 上传
下载 & 部署
DMC:
1. GDC：持续交付
2. GMC：元数据共享 & 隔离
3. GCC：WebIDE
应用包
全局存储
应用包内容:
1. 元数据
2. Jar 包
3. 配置项
4. package.json

Salesforce-架构设计原则

1. Stateless Appservers
2. Database system of record
3. No Database definition Language(DDL) at Runtime
4. All tables partitioned by OrgId
5. Smart Primary Keys, Polymorphic Foreign Keys
6. Creative de-normalization and pivoting
7. Use every RDBMS feature & optimization

推荐书籍

《kubernetes权威指南》
《DDIA》
《Fundamentals of Software Architecture》

Kingdee

感謝 ありがとう Thanks

ขอบคุณ

terima kasih 谢谢