Kubernetes核心原理
—Kubelet工作机制

王志彬

资料整理
火龙果软件
荐

	KVM	Machine Container	App Container
对业务的侵入性	无	低，内核参数配置会引起全局变化	高，需要写Dockerfile，业务启动的方式不同了
性能	中，太重。 IO性能损失较大	好	好
运维成本	低	中，top / sar等常用工具数据不正确，需要改造。	高，监控 / 日志需要对接，运维需要重新熟悉

一 Kubernetes是什么？

二 Kubernetes主要概念

三 Kubernetes总体架构

四 Kubernetes核心原理

五 业界应用案例介绍

六 在思源落地

资料整理
火龙果软件
荐

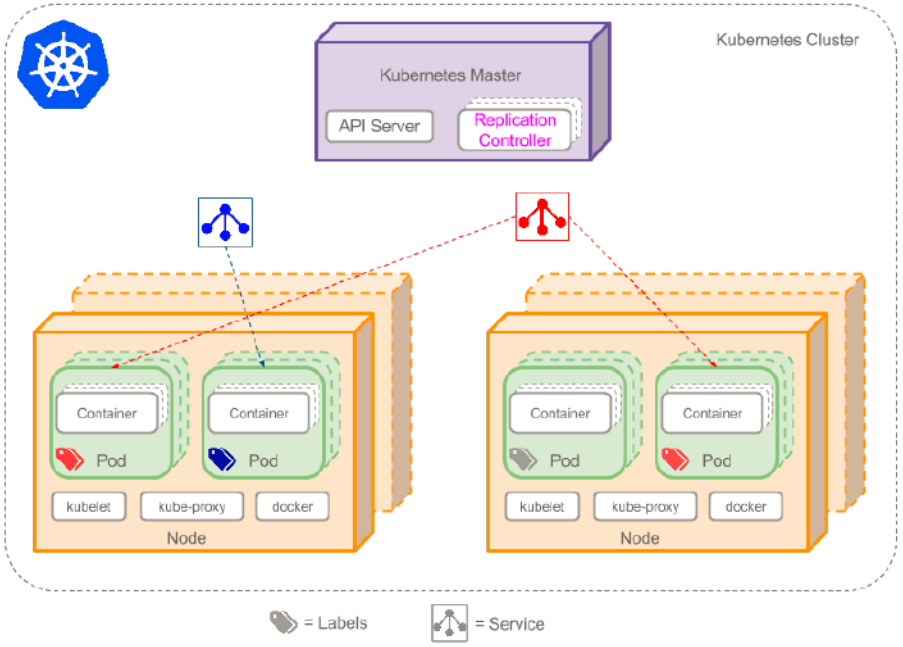
Kubernetes是Google开源的**容器集群管理系统**，其提供应用部署、维护、 扩展机制等功能，利用Kubernetes能方便地管理跨机器运行容器化的应用，其主要功能如下：

1) 使用Docker对应用程序包装(package)、实例化(instantiate)、运行(run)。
2) 以集群的方式运行、管理跨机器的容器。
3) 解决Docker跨机器容器之间的通讯问题。
4) Kubernetes的自我修复机制使得容器集群总是运行在用户期望的状态。

http://kubernetes.io/

资料整理
火龙果软件
荐

1. “轻装上阵”地开发复杂系统；
   - 方便地管理跨机器运行容器化的应用；
2. 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，谷歌将微服务架构的基础设施直接打包k8s解决方案中。
3. Kubernetes系统架构具备了超强的横向扩容能力。
   - 不修改代码，一个Kubernetes集群即可从只包含几个Node的小集群平滑扩展到拥有上百个Node的大规模集群。

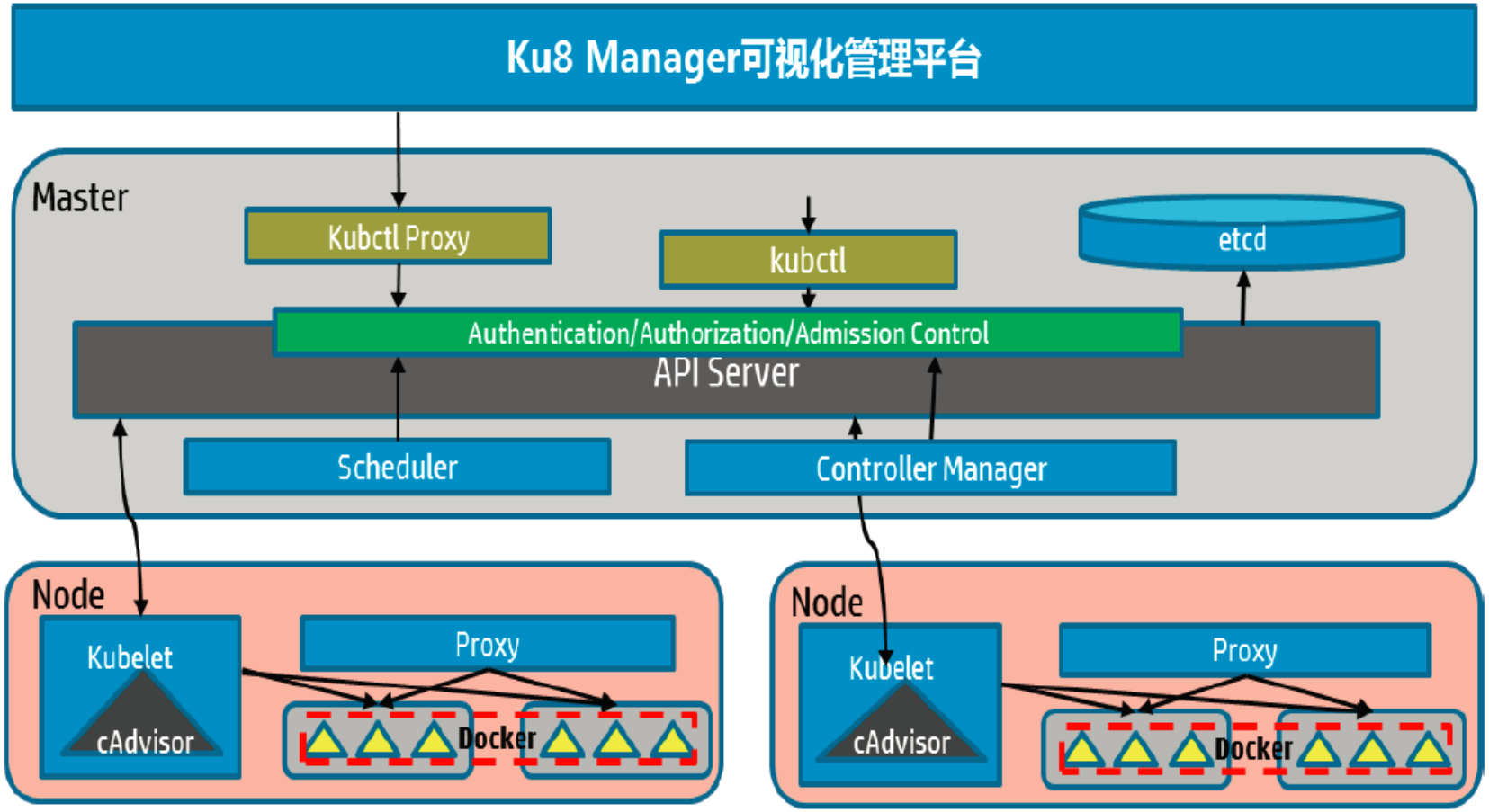


鸟语	人话
Container	容器
Pod	容器组
Replication Controller	复制控制器
Service	服务
Label	标签
Node	节点
Kubernetes Master	主节点

资料整理
火龙果软件

- **Pods** 是连接在一起的容器组合并共享文件卷。它们是最小的部署单元，由 Kubernetes 统一创建、调度、管理。Pods是可以直接创建的，但推荐的做法是使用 Replication Controller，即使是创建一个 Pod。
- **Replication controllers** 管理 Pods 的生命周期。它们确保指定数量的 Pods 会一直运行，通过创建和杀掉 Pods 可以保证到这个效果。
- **Labels** 它被用来管理和选取基于键值对为基础的对象组。
- **Services** 提供独立、可靠名称和地址的 Pods 集合。它就像一个基础版本的负载均衡器。
- **Node** 相对master是工作主机，较早版本也叫Minon，物理机、虚拟机均可；kubelet、kube-proxy、docker daemon；相当于openstack中的计算节点。
- **Namespace** 不同的namespace形成逻辑上不同的项目或用户组。
- **Volume** 是Pod中被多个容器访问的共享目录。支持EmptyDir、hostPath、gcePersistenDisk、awsElasticBlockStore、nfs、iscsi等等。

资料整理
火龙果软件
荐

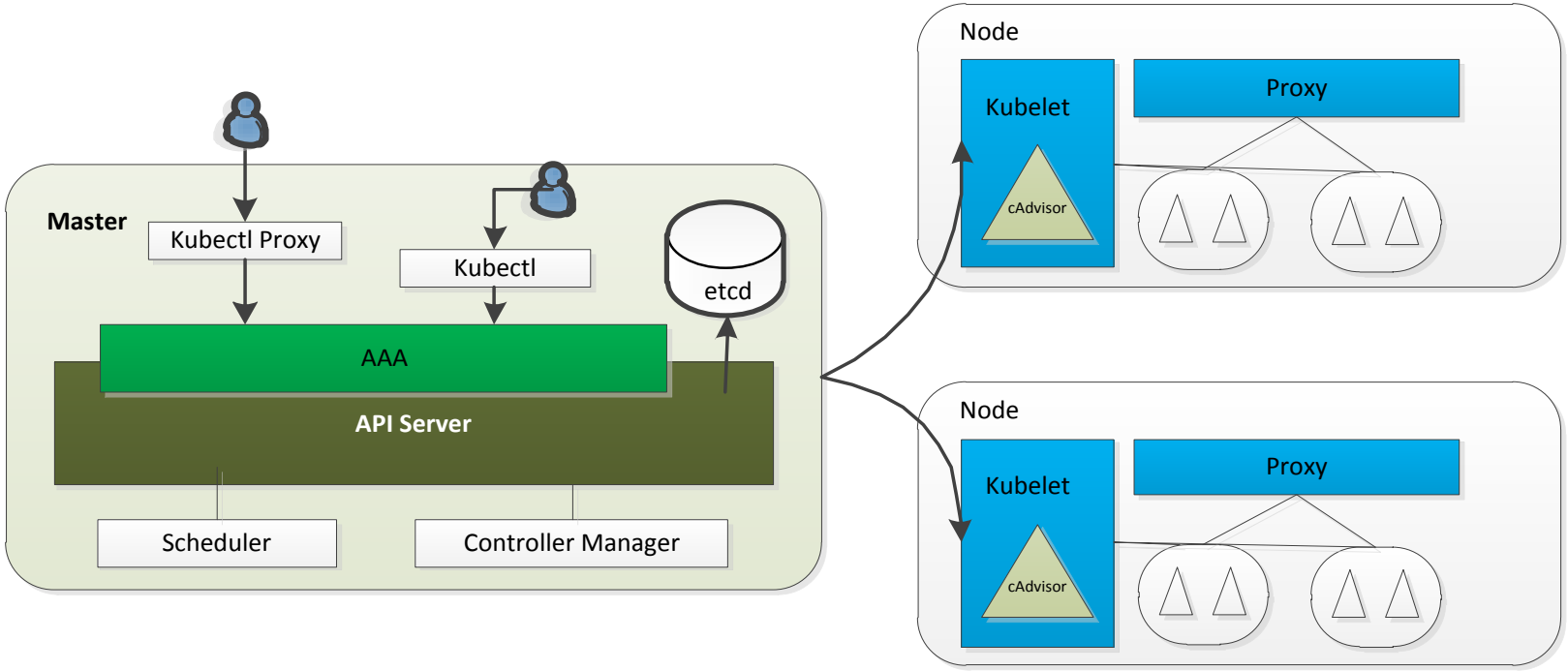


图例： 组件 用户 可靠存储 节点或Pod 容器 Docker

资料整理
火龙果软件
荐

API Server

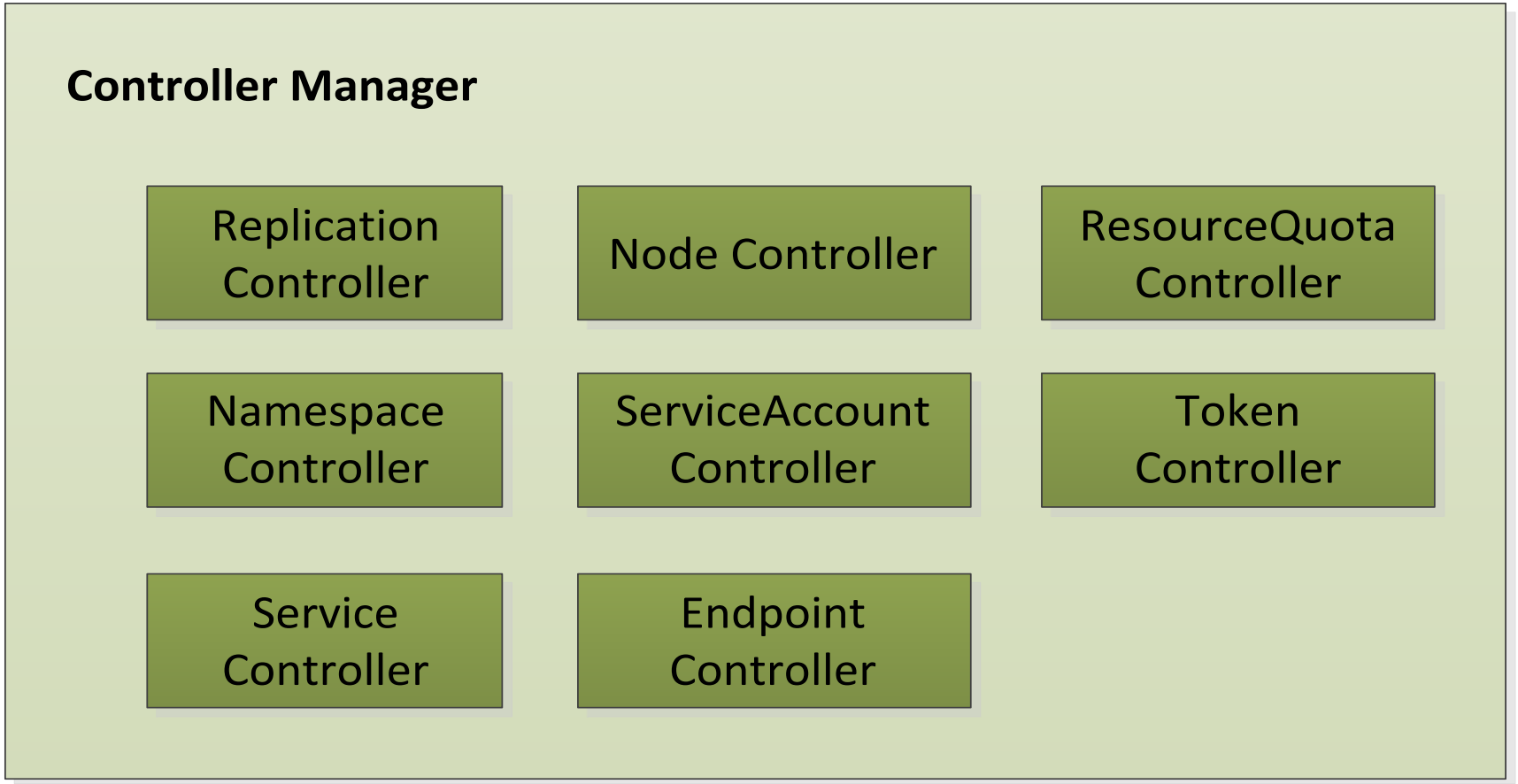
- 提供集群管理的API接口；
  - api-server进程提供http(8080)/https(6443)两个端口供访问；
- 成为集群内各个功能模块之间数据交互和通信的中心枢纽；



资料整理
火龙果软件
荐

- 并拥有完备的集群安全机制；
  - Authentication认证
  - Authorization授权
  - Admission Control准入控制
  - Secret私密凭据
  - Service Account

资料整理
火龙果软件
荐

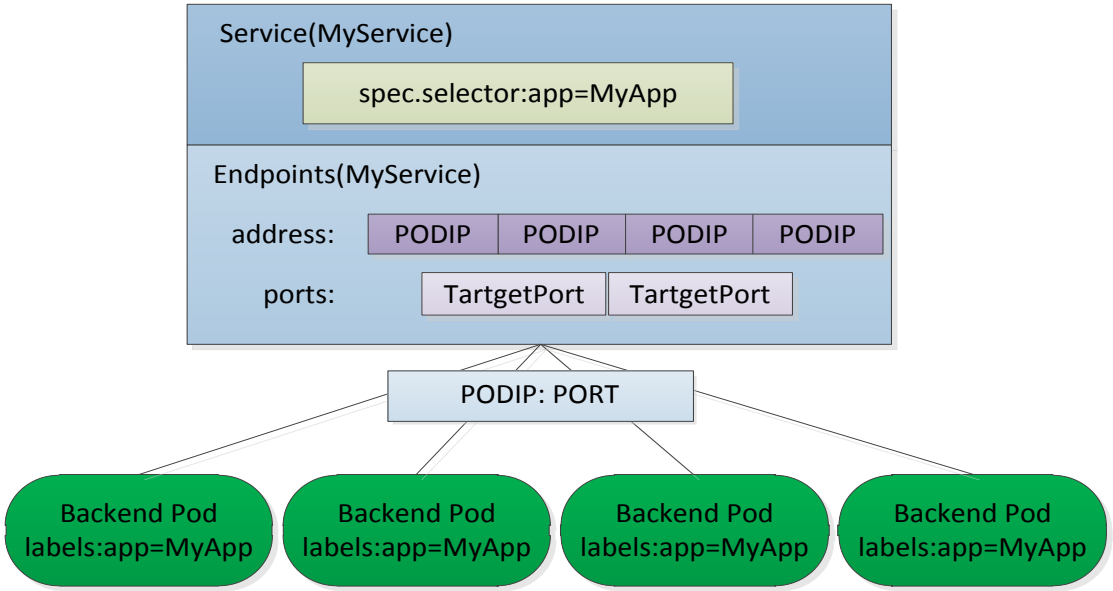

Controller Manager
Replication Controller
Node Controller
ResourceQuota Controller
Namespace Controller
ServiceAccount Controller
Token Controller
Service Controller
Endpoint Controller

资料整理
火龙果软件
荐

- Replication Controller
  - 确保在任何时候集群中一个rc下的pod都保持一定数量的pod副本处于正常运行状态。
  - 重新调度、弹性伸缩。
  - 滚动更新（Rolling Updates）
- Node Controller
  - 负责发现、管理和监控集群中的各个Node节点。
- ResourceQuota Controller
  - 资源配额管理确保制定对象在任何时候都不会超量占用系统资源。
  - 三个维度配额：容器（C/M）、Pod、namespace（pod、rc、service、resourceQuota、Secret、PV）

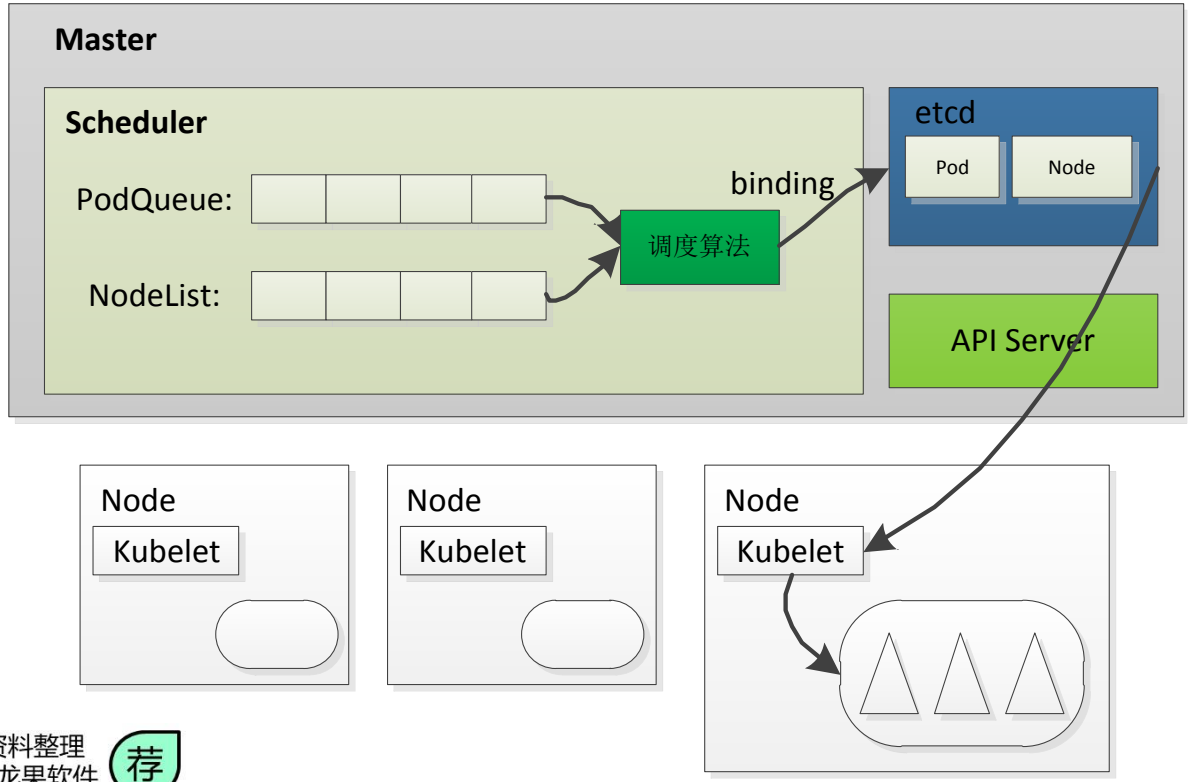
资料整理
火龙果软件
荐

- Namespace Controller
  - 负责namespace的全局管理
- ServiceAccount\Token Controller
  - 安全相关的控制器
- Service\Endpoint Controller



资料整理
火龙果软件
荐

- Scheduler
  - 负责Pod调度
  - 通过调度算法为待调度Pod列表的每个Pod从Node列表中选择一个最合适的Node。



预选策略：

- NoDiskConflict
- PodFitsResources
- PodSelectorMatches
- PodFitsHost
- CheckNodeLabel…
- …..

资料整理
火龙果软件
荐

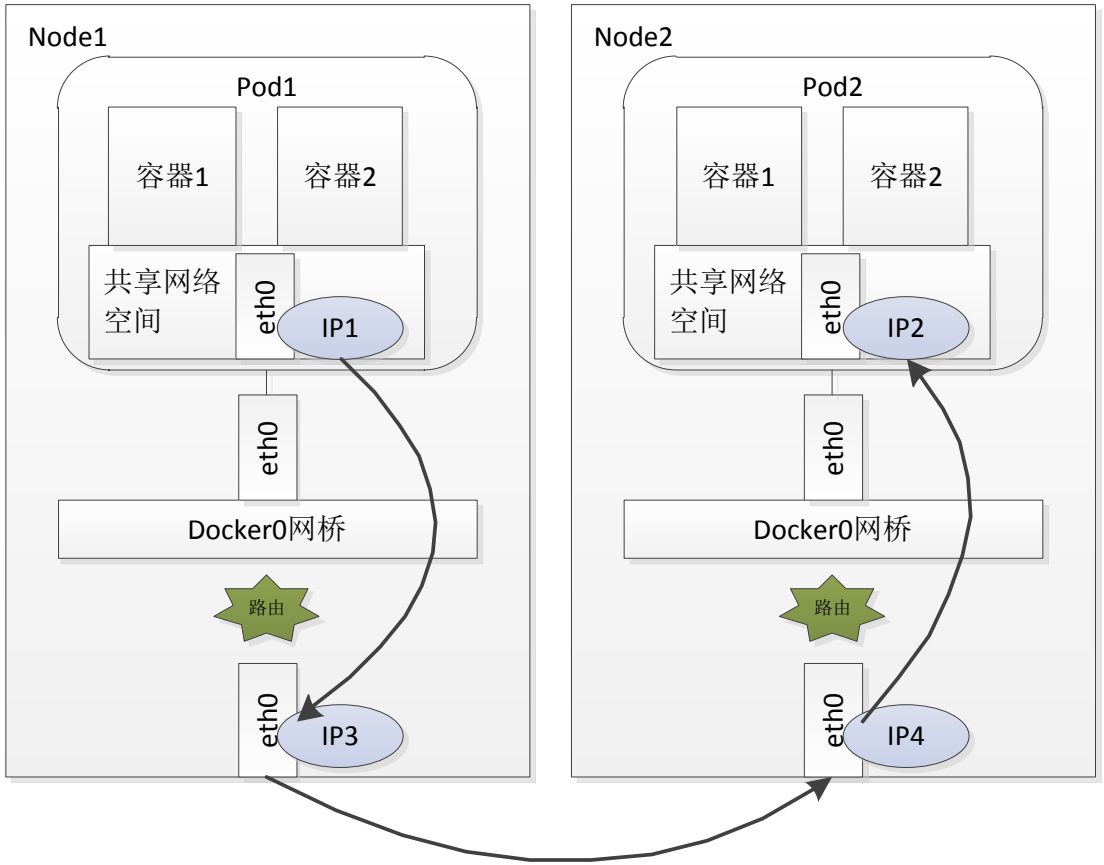
Kubelet机制

- 节点管理，向API Server注册自己
- Pod管理，通过API Server监听etcd目录，同步pod清单，增加、删除、修改pod；
- 通过LivenessProbe和ReadinessProbe的方式对容器做健康检查；
- cAdvisor资源监控，默认节点4194端口；

http://XXX.XXX.XXX.XXX:4194/containers/

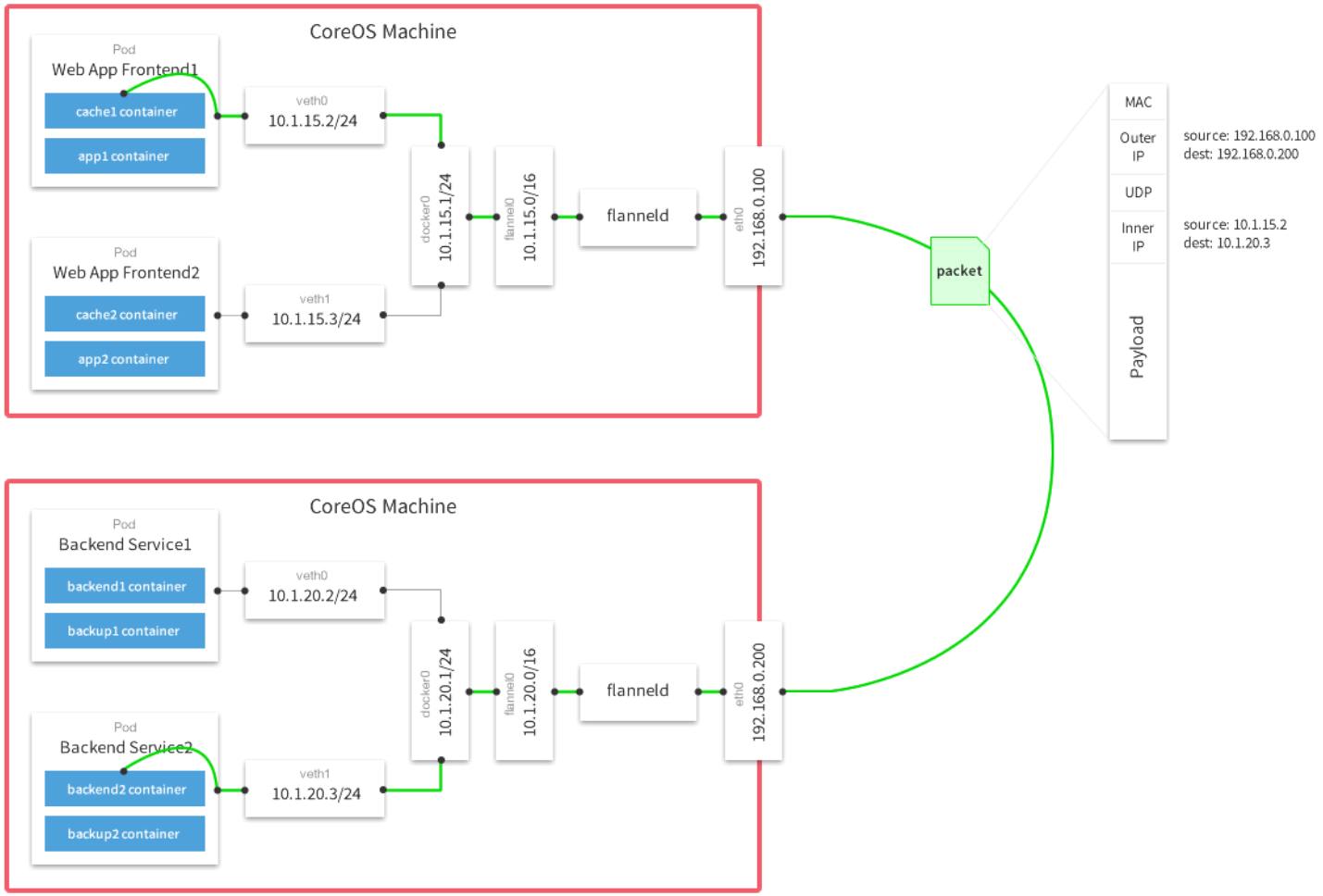
资料整理
火龙果软件
荐

- 容器到容器通信，共享网络空间；
- 同Node上pod间通信，通过docker0网桥；
- 不同Node上pod间通信？



资料整理
火龙果软件
荐

从IP1到IP2我们选Flannel解决方案



资料整理
火龙果软件
荐

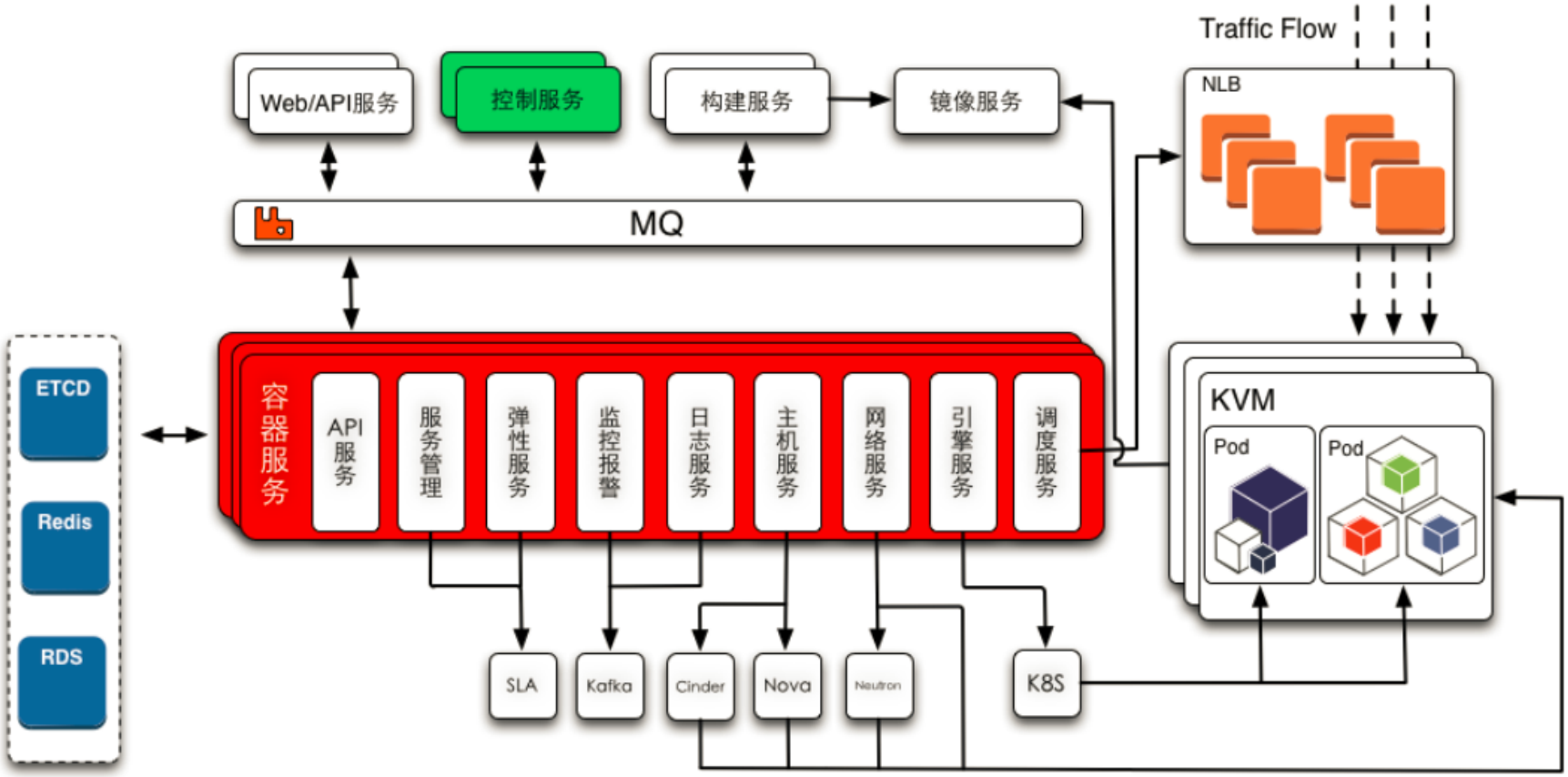
网络模式的性能对比

Results

Name	TCP BW (MB/s)	TCP Lat (µs)	BW %	Lat %
Native	116	91.8	100.00	100.00
Weave	6.91	372	5.96	405.23
Flannel UDP	23	164	19.83	178.65
Flannel VXLan	112	129	96.55	140.52

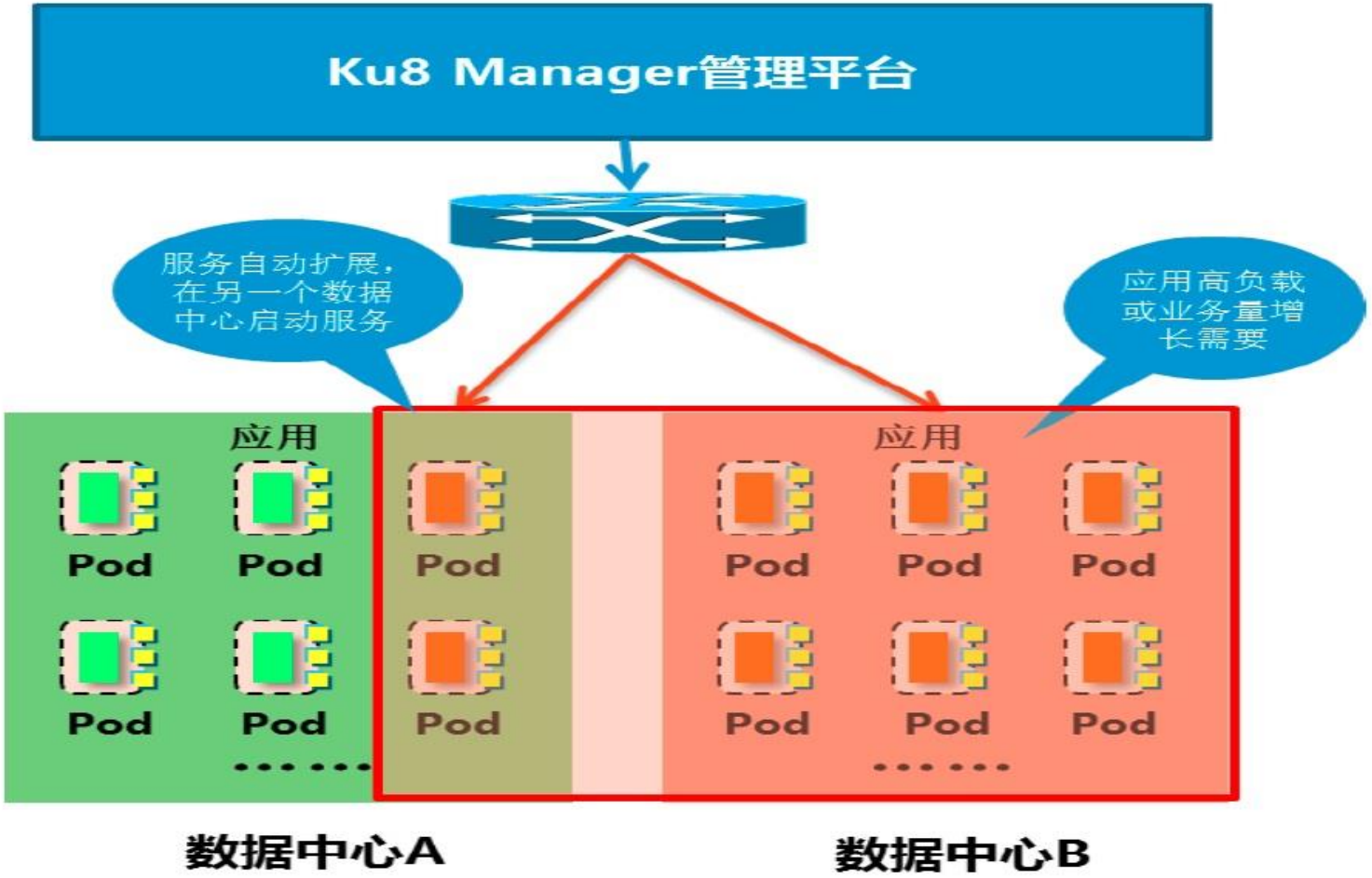
资料整理
火龙果软件
荐

–	Swarm	Mesos	k8s
架构特点	针对Docker体系	借鉴Borg理念	源自Borg，原生Docker
与Mesos结合	社区完善中	成熟	社区积极推荐
隔离机制	Docker	Mesos/Docker/其它	Docker
资源类型	内存 CPU 端口	内存 CPU 端口 Ulimit	内存 CPU 端口 Ulimit
调度非Docker	No	Yes	No
主机分组	Docker进程标签	Slave配置	Slave配置
打分策略	资源使用情况	资源使用情况	资源使用情况+应用节点均衡
端口编排	No	Yes	Yes
网络模式	Docker原生	支持自建	支持自建
主高可用	双主切换	zk	etcd
扩缩容	Roam二次开发	API修改实例数	API修改实例数
应用节点健康检测	No	Yes	Yes
应用节点自动故障转移	No	Yes	Yes
服务发现	Consul	zk	etcd
负载均衡	No	Haproxy	Kube-proxy
DNS	No	No	skydns
使用复杂度	低	中	中
集群规模	小	中	中，在提升
生产使用	尚无大规模使用	业界大规模使用	业界大规模使用


Traffic Flow
Web/API服务
控制服务
构建服务
镜像服务
NLB
MQ
ETCD
Redis
RDS
容器服务
API服务
服务管理
弹性服务
监控报警
日志服务
主机服务
网络服务
引擎服务
调度服务
KVM
Pod
Pod
SLA
Kafka
Cinder
Nova
Neutron
K8S

资料整理
火龙果软件
荐

PAAS平台功能 – 集群间业务动态调整



Q&A

资料整理
火龙果软件
荐