参考资料

1.技术分享：Kubernetes Networking Model

2.万字长文，带你搞懂 Kubernetes 网络模型

3.使用kubeadm搭建k8s集群

4.k8s的kubectl源码解析

5.Kubernetes--Service负载均衡机制

6.Kube Controller Manager 源码分析

7.k8s源码-scheduler流程深度剖析

8.kubectl 创建 Pod 背后到底发生了什么?

9.源码解析：一文读懂 Kubelet

10.kube-proxy源码分析：深入浅出理解k8s的services工作原理

本文基于k8s v1.28.2

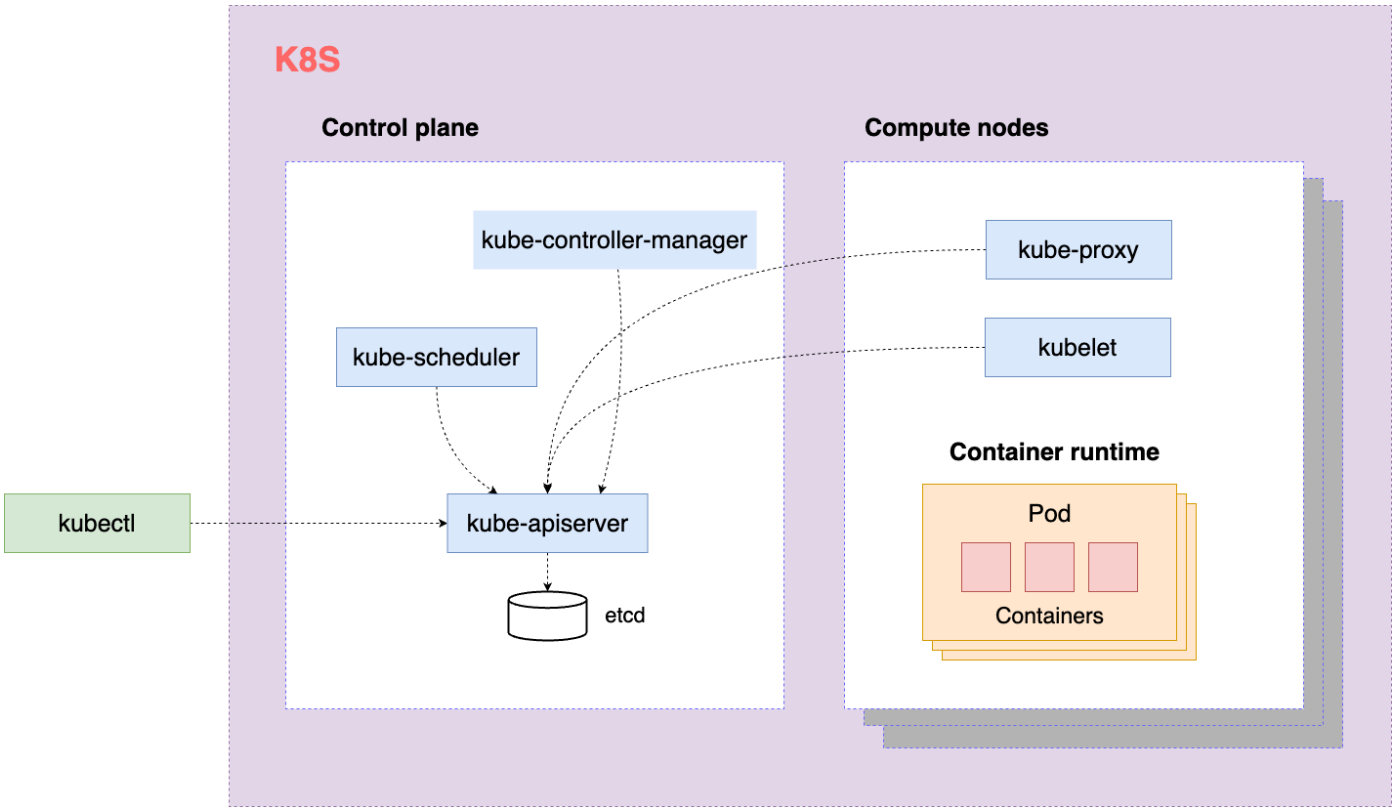
大纲

4. k8s使用小贴士

---

1. k8s架构

1.1 架构图



HTTP/GRPC接口一般配有SSL的双向认证(使用自签名的根证书)

- kube-apiserver
- etcd

kube-apiserver处于核心地位，其它组件通过它实现了解耦，其作用相当于看板

kube-apiserver也可以做高可用，比如使用负载均衡器（Load Balancer）来分发流量到多个kube-apiserver实例

扩展点

- CNI　网络
- CSI　存储
- CRI　container runtime
- Operator（CRD + controller）

k8s可以独立于container runtime进行升级

1.2 组件

1.2.1 控制平面

1) kube-apiserver

组件负责公开 Kubernetes API，负责处理接受请求的工作。

- HTTP 服务
- API Group
- 标准的RESTful API
  - GET 查询
  - POST 增加
  - PUT 修改
  - DELETE 删除

API Group

API路径的构成通常遵循以下格式：

```
/apis/<API_Group>/<API_Version>/<Resource_Type>/<Resource_Name>
```

Core API Group:

简写: ""

API路径: /api/v1

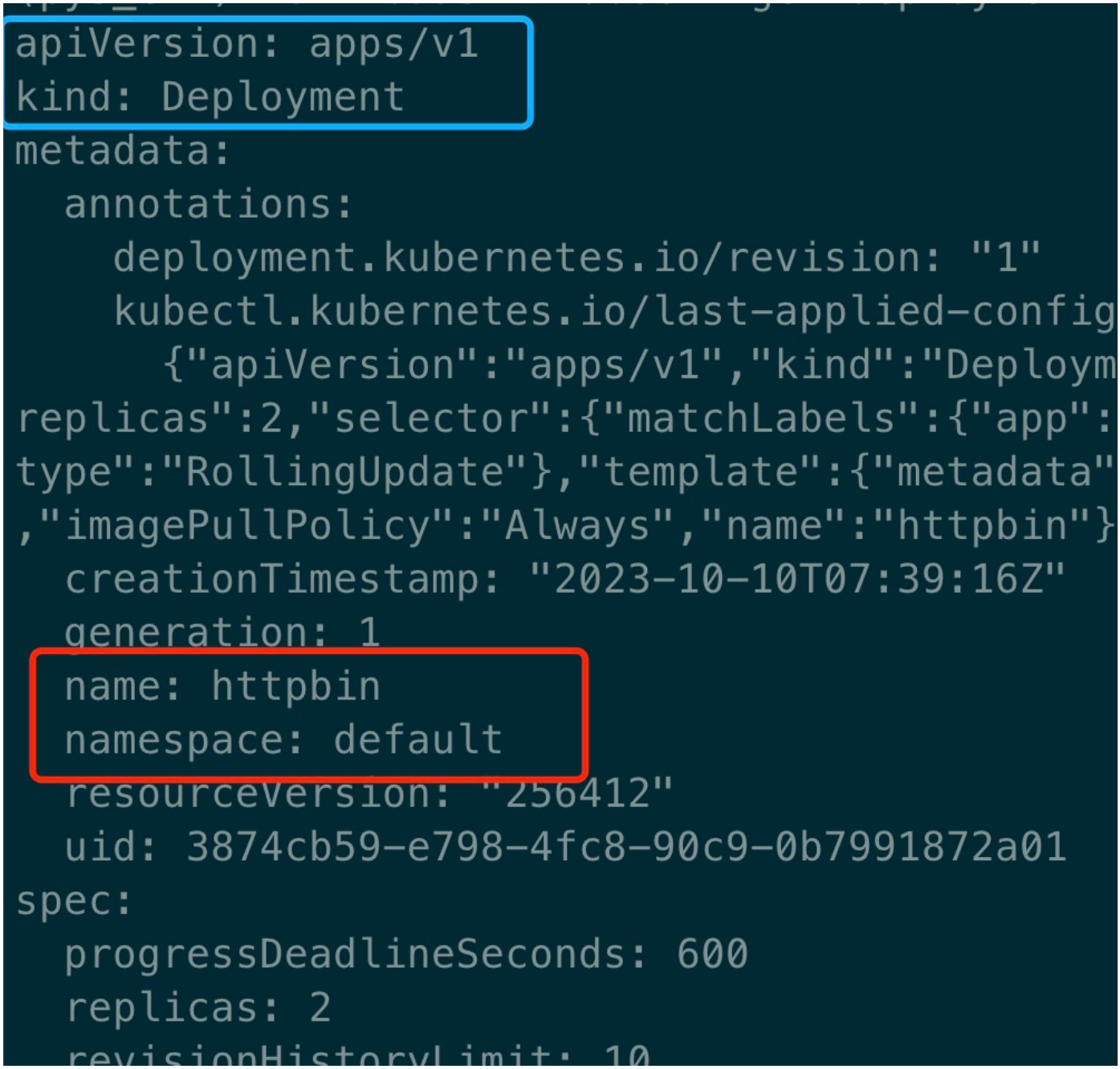
包含核心的 Kubernetes 资源，如 Pod、Service、Node、Namespace 等。

Apps API Group:

简写: "apps"

API路径: /apis/apps

包含应用相关的资源，如 Deployment、StatefulSet、ReplicaSet 等。



2）etcd

一致且高可用的键值存储，配置数据以及有关集群状态的都保存在其中。

其外也被用于选主 leader election

```
/registry/<Resource_Type>/<namespace>/<name>
```

示例

```
/registry/pods/default/httpbin-66c877d7d-pk7fl
/registry/pods/default/httpbin-66c877d7d-xx4j9
```

注意： {namespace} + {name} 必须是全局唯一的

---

3) kube-controller-manager

负责运行控制器(controller)进程。

从逻辑上讲， 每个controller都是一个单独的进程， 但是为了降低复杂性，它们都被编译到同一个可执行文件，并在同一个进程中运行。

- NamespaceController
- DeploymentController
- ReplicaSetController
- DaemonSetController
- StatefulSetController
- HorizontalPodAutoscalerController
- GarbageCollectorController

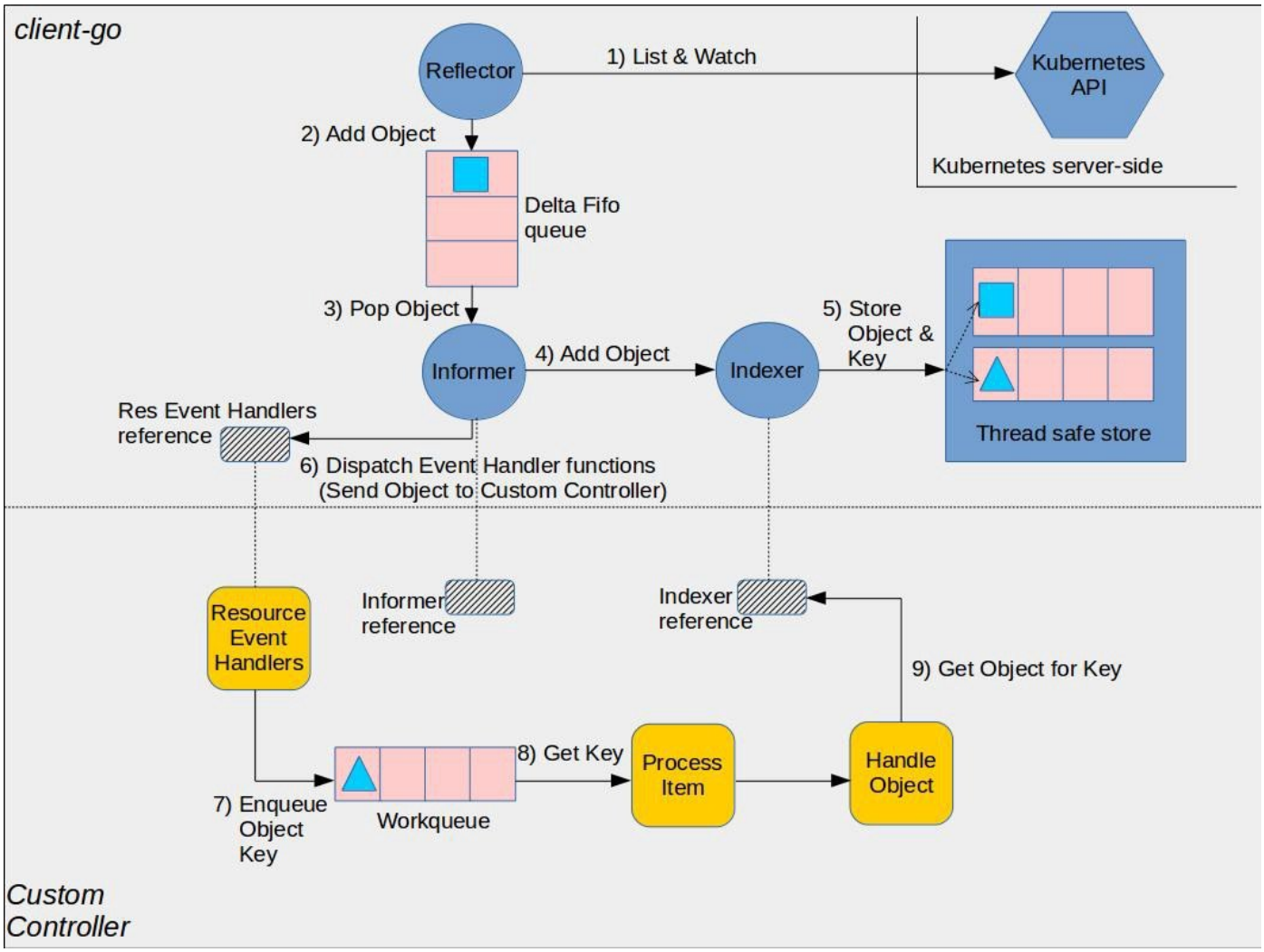
每种controller一般只监听一种或几种特定的资源

---

controller执行逻辑的伪代码

```
while(true) {
  expectState := GetResourceExpectState(rs)
  actualState := GetResourceActualState(rs)
  if(actualState == expectState){
    // do nothing
  } else {
    Reconcile(rs) // 编排逻辑，调谐的最终结果一般是对被控制对象的某种写操作，比如增/删/改 Pod
  }
}
```

获取某个资源的期望状态或者实际状态的公共逻辑，所以k8s提供了informer组件



4) kube-scheduler

负责监视新创建的、未指定运行节点（node）的 Pods， 并选择节点来让 Pod 在上面运行。

- 亲和性(affinity)规则
- 反亲和性(anti-affinity)规则

举例：

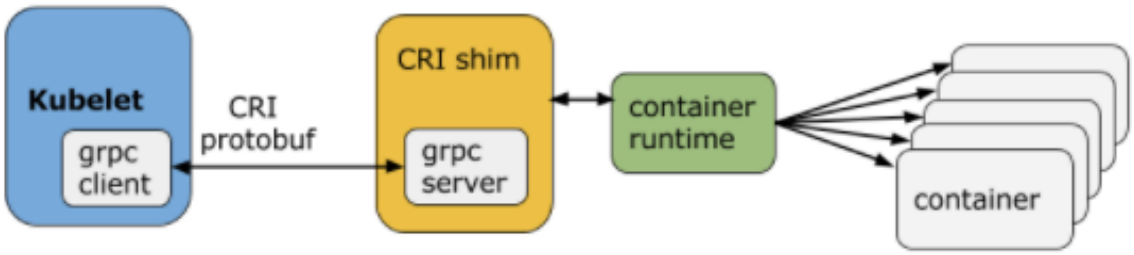
- 某视频处理程序所在Pod，需要被调度到拥有GPU的node上
- 某个thanos实例，需要被被调度到拥有固态硬盘的node上

1.2.2 计算节点

1) kubelet

kubelet是Kubernetes集群中运行在每个节点上的核心组件，它负责管理节点上的容器和与Master节点的通信。 kubelet的主要作用包括**容器生命周期管理、资源管理和调度、容器网络配置、存储卷管理以及与Master节点的通信**

kubelet本身也相当于一个控制器



CRI 一般监听在某个unix套接字上，形如： unix:///var/run/cri-dockerd.sock

2) kube-proxy

它的主要作用是实现服务的网络代理和负载均衡。

通过iptables或者ipvs组件，设置service -> pod的网络转发规则

2. k8s的基本工作流程

Step1 `kubectl`

请求 `kube-apiserver` 创建在ETCD中创建一个 `Deployment`

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: httpbin
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: httpbin
  template:
    metadata:
      labels:
        app: httpbin
    spec:
      containers:
        - image: kennethreitz/httpbin
          imagePullPolicy: Always
          name: httpbin
```

Step2 `DeploymentController`

watch `kube-apiserver` 发现 `Deployment` default/httpbin，请求 `kube-apiserver` 在ETCD中创建一个 `ReplicaSet`

Step3 `ReplicaSetController`

watch `kube-apiserver` 发现 `ReplicaSet` default/httpbin-66c877d7d，请求 `kube-apiserver` 在ETCD中创建2个 `Pod`

- default/httpbin-66c877d7d-pk7fl
- default/httpbin-66c877d7d-xx4j9

Step4 `kube-scheduler`

watch `kube-apiserver` 发现有 `Pod` 的nodeName属性为空，根据各个Node的CPU、内存容量以及各种亲和性、反亲和性规则为 `Pod` 更新nodeName属性。

```
kind: Pod
metadata:
  creationTimestamp: "2023-10-10T07:39:16Z"
  generateName: httpbin-66c877d7d-
  labels:
    app: httpbin
    pod-template-hash: 66c877d7d
  name: httpbin-66c877d7d-pk7fl
  namespace: default
  resourceVersion: "256410"
  uid: 5edbd129-c3c2-4ee7-ad4f-52a159789427
spec:
  containers:
  - image: kennethreitz/httpbin
    imagePullPolicy: Always
    name: httpbin
  dnsPolicy: ClusterFirst
  nodeName: ""        # 初始时nodeName为空，调度完成后，此字段被赋值   myk8s-
1a1cf5d13-4
  preemptionPolicy: PreemptLowerPriority
```

Step5 `kubelet`

myk8s-1a1cf5d13-4 上的 `kubelet` watch `kube-apiserver` 发现有 `Pod` 的nodeName属性为"myk8s-1a1cf5d13-4", 然而本机的docker runtime上，并没有对应的 `Pod` 被运行。因此需要执行操作。

- 拉取对应image
- 创建Pod等

上面的Step1 ~ Step4都发生在控制平面，只有Step5涉及计算节点

3. k8s网络模型

3.1 2个重要参数service-cidr和pod-network-cidr

```
kubeadm init \
--apiserver-advertise-address=10.128.246.197 \
--kubernetes-version=v1.28.2 \
--service-cidr=192.18.0.0/16 \
--pod-network-cidr=192.20.0.0/16
```

"service-cidr"（Service Cluster IP Range）是一个网络配置参数，它定义了用于Kubernetes服务的虚拟IP地址范围。

"pod-network-cidr" 是一个Kubernetes集群的网络配置参数，它用于定义分配给Pod的IP地址范围。

每个计算节点加入k8s之后，它的podCIDR就已经确定了，下图是拥有4个计算节点的某个k8s集群的podCIDR分配情况

```
[root@myk8s-1a1cf5d13-1 ~]# kubectl get nodes -o json | jq -r '.items[].spec'
{
  "podCIDR": "192.20.0.0/24",
  "podCIDRs": [
    "192.20.0.0/24"
  ],
  "taints": [
    {
      "effect": "NoSchedule",
      "key": "node-role.kubernetes.io/control-plane"
    }
  ]
}
{
  "podCIDR": "192.20.1.0/24",
  "podCIDRs": [
    "192.20.1.0/24"
  ]
}
{
  "podCIDR": "192.20.2.0/24",
  "podCIDRs": [
    "192.20.2.0/24"
  ]
}
{
  "podCIDR": "192.20.3.0/24",
  "podCIDRs": [
    "192.20.3.0/24"
  ]
}
```

Kubernetes 对任何网络实现都规定了以下要求：

- 所有 Pod 都可以在不使用网络地址转换 (NAT) 的情况下与所有其他 Pod 通信。
- 所有节点都可以在没有 NAT 的情况下与所有 Pod 通信。
- Pod 认为自己的 IP 与其他人认为的 IP 相同。

鉴于这些限制，我们需要解决四种不同的网络问题：

- 容器和容器
- Pod 和 Pod
- Pod 和 Service

- Internet 和 Service

3.2 Container 和 Container 的交互

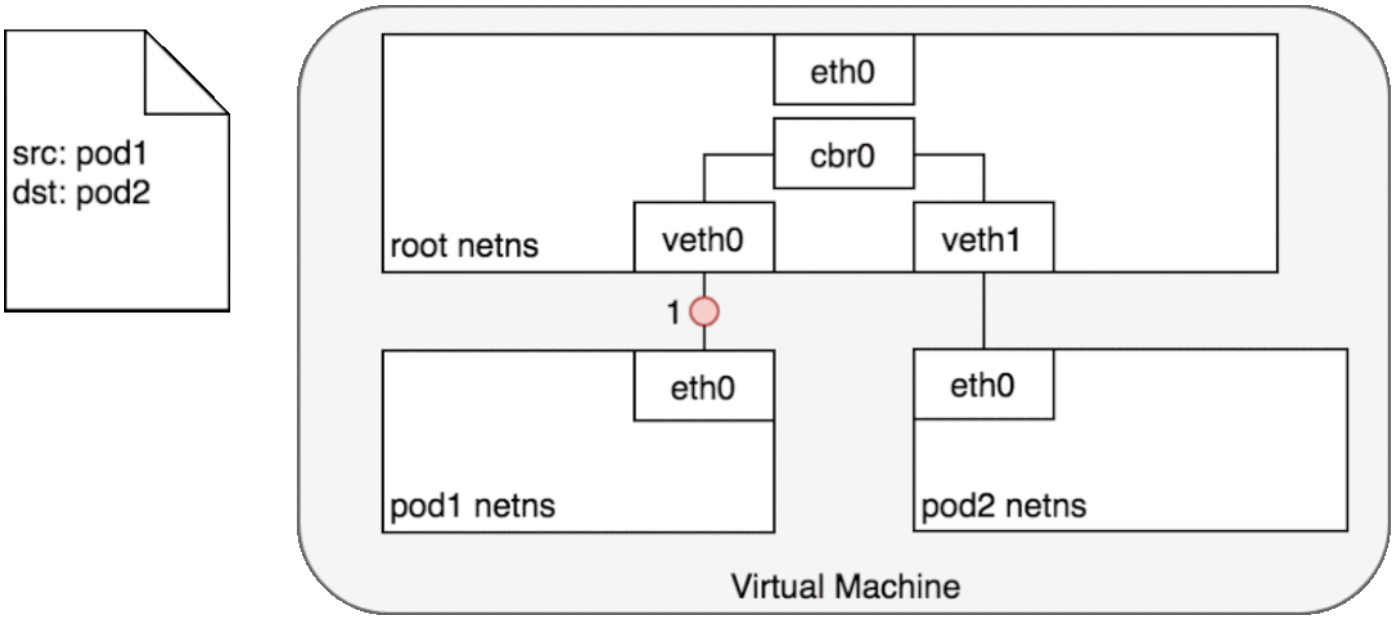
同一个Pod中的多个Container使用相同的网络命名空间，使用同一个虚拟网卡

3.3.Pod 和 Pod 的交互

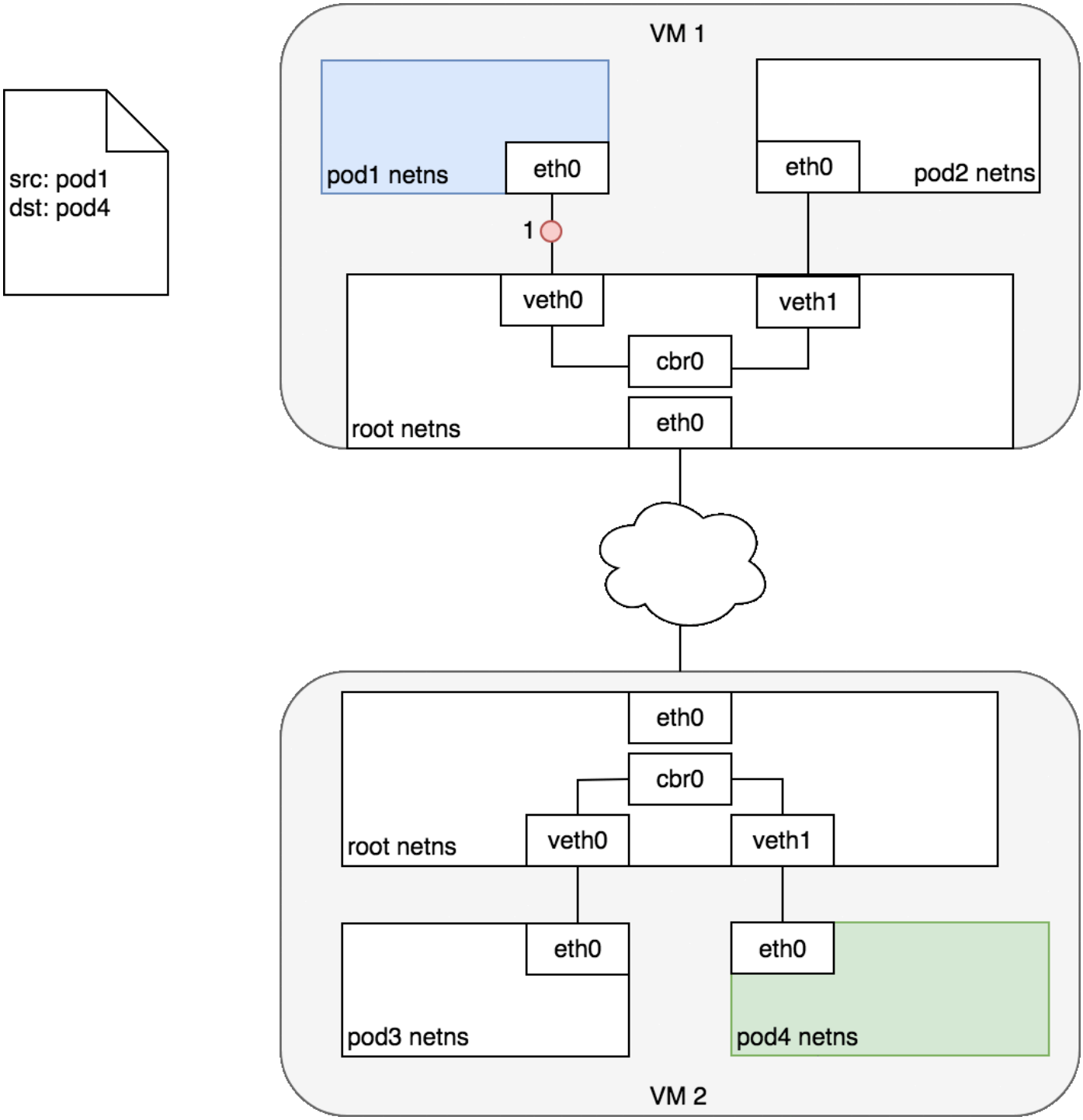
3.3.1 同一个Node上的Pod

使用以太网桥进行通讯相互通讯，多个网络命名空间使用虚拟网卡对打通

注: 网桥工作在数据链路层



3.3.2 不同Node上的Pod



注: 网络插件使用Flannel

```
[root@myk8s-1a1cf5d13-4 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.128.255.254  0.0.0.0         UG    100    0        0 eth0
10.0.0.0        10.128.255.254  255.0.0.0       UG    100    0        0 eth0
10.128.240.0    0.0.0.0         255.255.240.0   U     100    0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
192.20.0.0      192.20.0.0      255.255.255.0   UG    0      0        0 flannel.1
192.20.1.0      192.20.1.0      255.255.255.0   UG    0      0        0 flannel.1
192.20.2.0      192.20.2.0      255.255.255.0   UG    0      0        0 flannel.1
192.20.3.0      0.0.0.0         255.255.255.0   U     0      0        0 cni0
```

```
[root@myk8s-1a1cf5d13-4 ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether fa:16:3e:bc:3e:cc brd ff:ff:ff:ff:ff:ff
    inet 10.128.242.42/20 brd 10.128.255.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:27:37:d0:88 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
4: flannel.1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default
    link/ether 2a:e7:db:e4:29:1d brd ff:ff:ff:ff:ff:ff
    inet 192.20.3.0/32 scope global flannel.1
       valid_lft forever preferred_lft forever
5: cni0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP group default qlen 1000
    link/ether 72:21:13:53:15:56 brd ff:ff:ff:ff:ff:ff
    inet 192.20.3.1/24 brd 192.20.3.255 scope global cni0
       valid_lft forever preferred_lft forever
```

3.4 Pod 和 Service的交互

Kubernetes Service也有域名，其形式为 `<service-name>.<namespace>.svc.cluster.local` 。这使得其他Pod可以通过Service域名来访问Service，而不必关心Service后面的Pod IP地址。

注: DNS解析需要安装CoreDNS

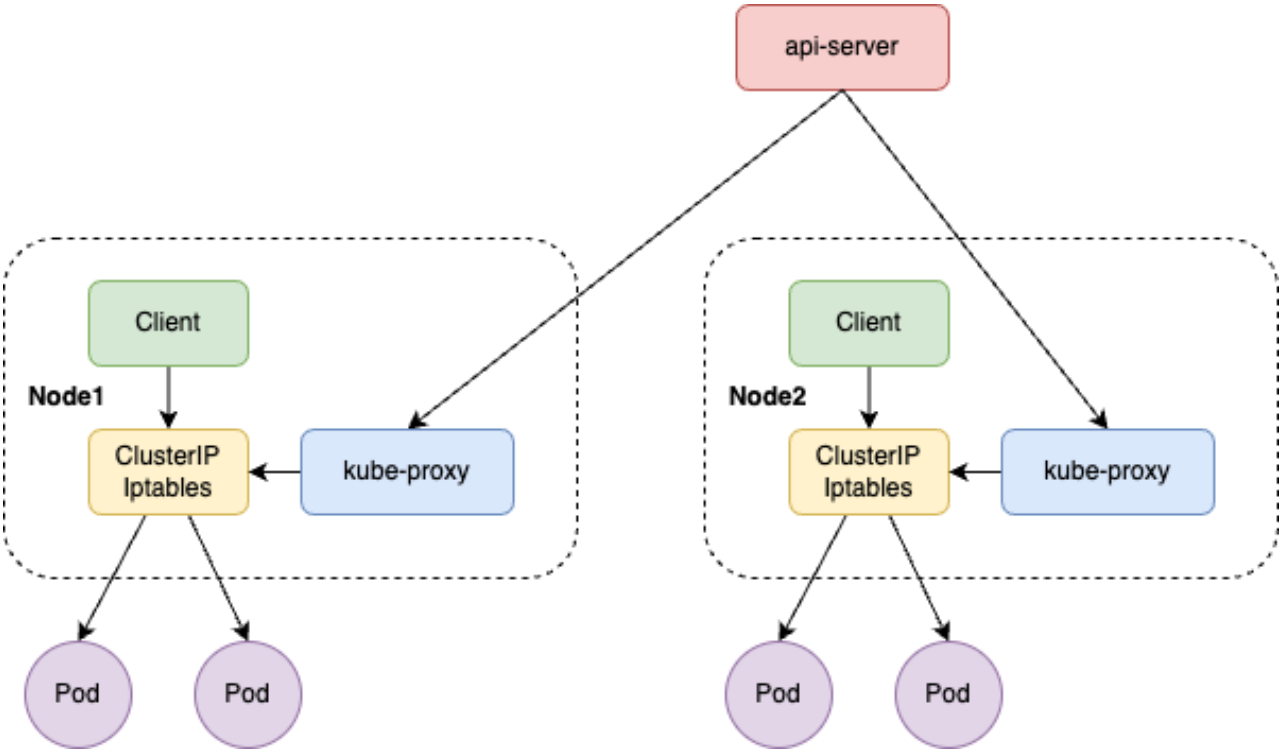
```
root@simple-pod:/# dig httpbin-svc.default.svc.cluster.local

; <<>> DiG 9.16.44-Debian <<>> httpbin-svc.default.svc.cluster.local
;; global options: +cmd
;; Got answer:
;; WARNING: .local is reserved for Multicast DNS
;; You are currently testing what happens when an mDNS query is leaked to DNS
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 20218
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: c26aec6f469e147c (echoed)
;; QUESTION SECTION:
;httpbin-svc.default.svc.cluster.local. IN A

;; ANSWER SECTION:
httpbin-svc.default.svc.cluster.local. 30 IN A  192.18.86.37

;; Query time: 0 msec
;; SERVER: 192.18.0.10#53(192.18.0.10)
;; WHEN: Fri Oct 27 16:40:29 CST 2023
;; MSG SIZE  rcvd: 131
```




```
[root@myk8s-1a1cf5d13-4 ~]# iptables -S -t nat | grep default/httpbin-svc
-A KUBE-SERVICES -d 192.18.86.37/32 -p tcp -m comment --comment "default/httpbin-svc cluster IP" -m tcp --dport 80 -j KUBE-SVC-OHIL3USJ2RV6WHZD
-A KUBE-NODEPORTS -p tcp -m comment --comment "default/httpbin-svc" -m tcp --dport 32694 -j KUBE-EXT-OHIL3USJ2RV6WHZD
-A KUBE-EXT-OHIL3USJ2RV6WHZD -m comment --comment "masquerade traffic for default/httpbin-svc external destinations" -j KUBE-MARK-MASQ
-A KUBE-SVC-OHIL3USJ2RV6WHZD ! -s 192.20.0.0/32 -d 192.18.86.37/32 -p tcp -m comment --comment "default/httpbin-svc cluster IP" -m tcp --dport 80 -j KUBE-MAR
K-MASQ
-A KUBE-SVC-OHIL3USJ2RV6WHZD -m comment --comment "default/httpbin-svc -> 192.20.1.5:80" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-T
JEKGOQ2VXNS7DIF
-A KUBE-SVC-OHIL3USJ2RV6WHZD -m comment --comment "default/httpbin-svc -> 192.20.3.7:80" -j KUBE-SEP-G6KQE5HYRPUZF7BK
-A KUBE-SEP-TJEKGOQ2VXNS7DIF -s 192.20.1.5/32 -m comment --comment "default/httpbin-svc" -j KUBE-MARK-MASQ
-A KUBE-SEP-TJEKGOQ2VXNS7DIF -p tcp -m comment --comment "default/httpbin-svc" -m tcp -j DNAT --to-destination 192.20.1.5:80
-A KUBE-SEP-G6KQE5HYRPUZF7BK -s 192.20.3.7/32 -m comment --comment "default/httpbin-svc" -j KUBE-MARK-MASQ
-A KUBE-SEP-G6KQE5HYRPUZF7BK -p tcp -m comment --comment "default/httpbin-svc" -m tcp -j DNAT --to-destination 192.20.3.7:80
[root@myk8s-1a1cf5d13-4 ~]#
```

上图中iptables把访问192.18.86.37:80地址的请求转发给

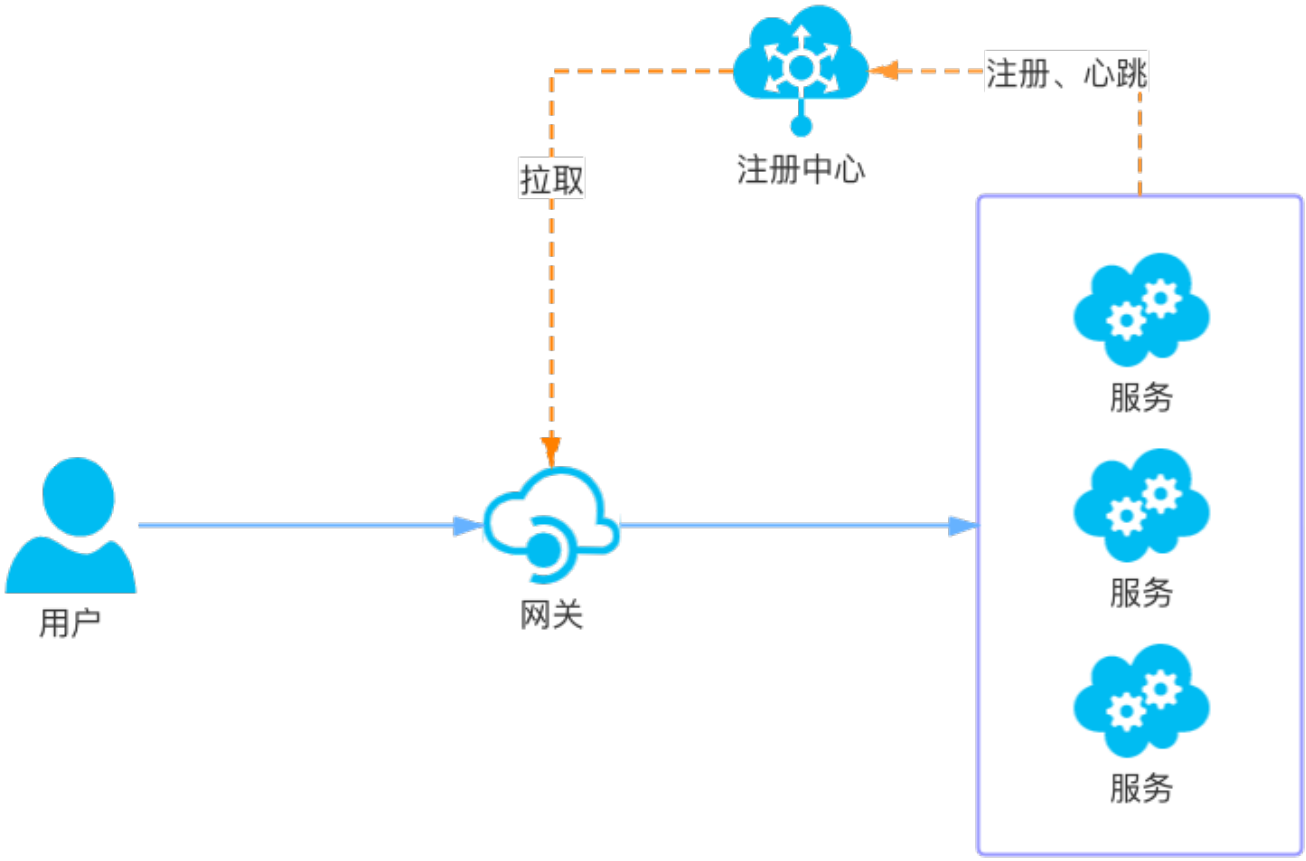
- (1) 192.20.1.5:80
- (2) 192.20.3.7:80

注意：

- k8s的负载均衡在内核中运行(iptables或ipvs)
- 以 `连接` 为负载均衡的维度，所以对支持多路复用的长连接服务无效。
- 每个Node都会包括全量的service到pod的路由信息(比如service1、service2 ... service100)

4. k8s使用小贴士

4.1 通过API网关访问服务



4.2 DeployMent

- 如果一个Deployment对象的名称不变，修改它的属性通常会触发滚动发布（Rolling Update）
- 修改一个Deployment对象的名称

  相当于删除一个旧的Deployment对象，并创建一个新的Deployment对象， **会导致之前Deployment关联的Pod被全部删除无法达到平滑升级的目的** 。