法律声明

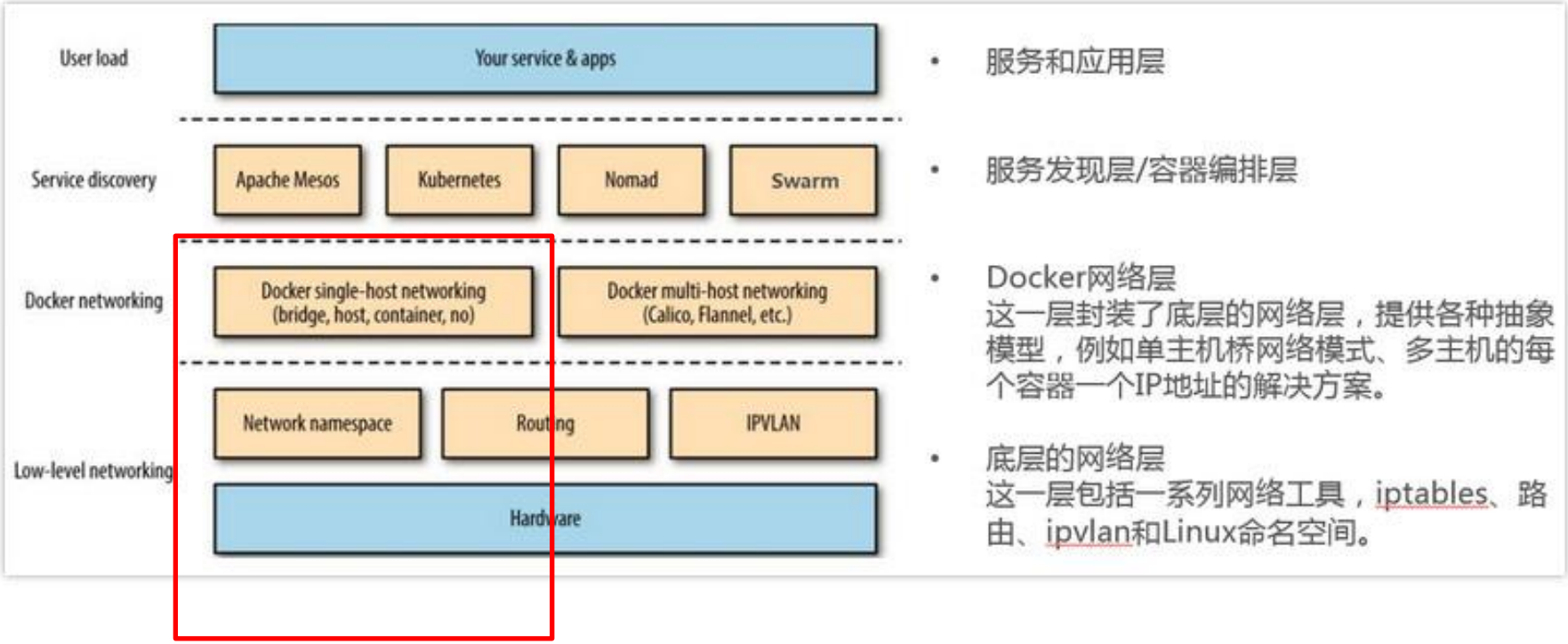
- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，讲师及小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：小象
  - 新浪微博：ChinaHadoop

Kubernetes网络详解

目录

1. 网络技术栈


User load
Your service & apps
Service discovery
Apache Mesos
Kubernetes
Nomad
Swarm
Docker networking
Docker single-host networking (bridge, host, container, no)
Docker multi-host networking (Calico, Flannel, etc.)
Network namespace
Routing
IPVLAN
Low-level networking
Hardware
• 服务和应用层
• 服务发现层/容器编排层
• Docker网络层
这一层封装了底层的网络层，提供各种抽象模型，例如单主机桥网络模式、多主机的每个容器一个IP地址的解决方案。
• 底层的网络层
这一层包括一系列网络工具，iptables、路由、ipvlan和Linux命名空间。

1. Docker网络模型及Bridge

- 随机端口：-P

```
$ docker run -d -P training/webapp python app.py

$ docker container ls -l
CONTAINER ID  IMAGE                     COMMAND       CREATED        STATUS        PORTS
bc533791f3f5  training/webapp:latest  python app.py 5 seconds ago  Up 2 seconds  0.0.0.0:49155->5000/tcp
```

- 映射所有接口地址，固定端口

```
$ docker run -d -p 5000:5000 training/webapp python app.py
```

- 映射到指定地址的指定端口

```
$ docker run -d -p 127.0.0.1:5000:5000 training/webapp python app.py
$ docker run -d -p 127.0.0.1:5000:5000/udp training/webapp python app.p  #udp
```

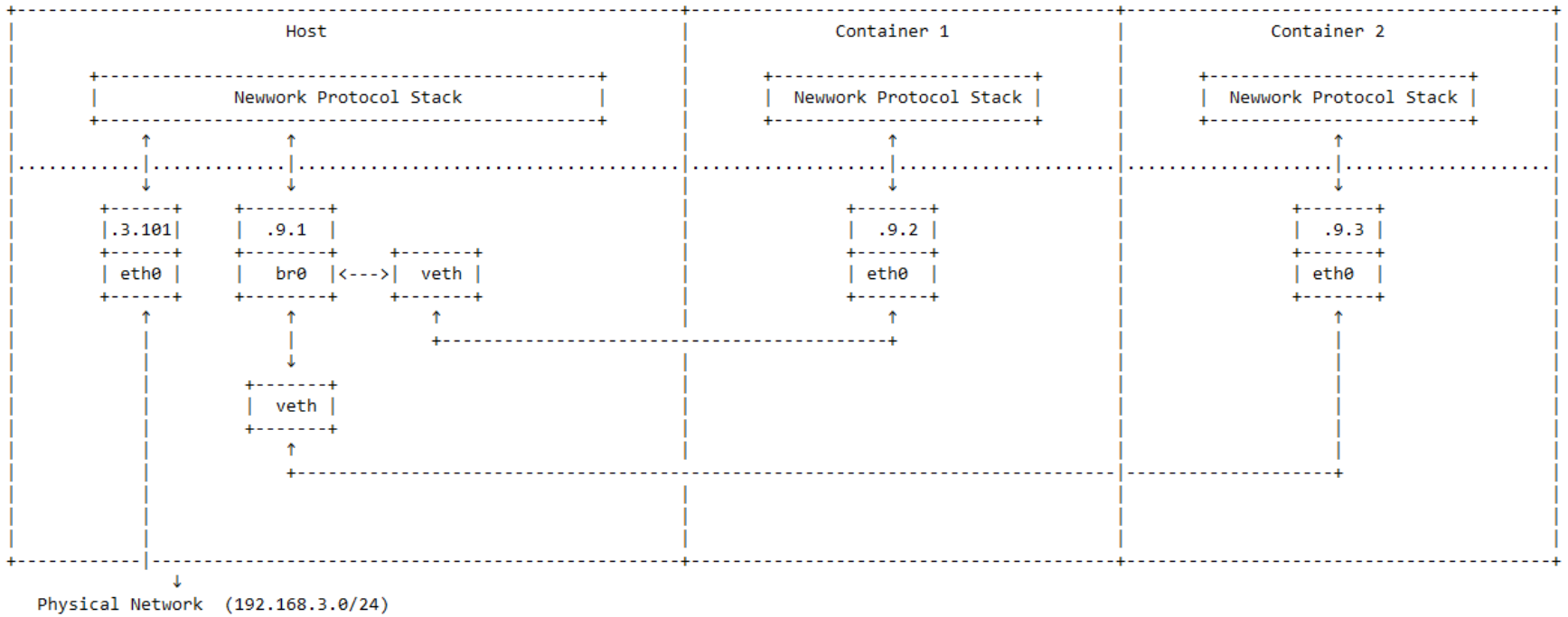
- 多端口

```
$ docker run -d  -p 5000:5000  -p 3000:80  training/webapp  python app.py
```

1. Docker网络模型及Bridge

格式	用途
ip:hostport:containerport	指定ip、指定主机port、指定容器port
ip::containerport	指定ip、未指定主机port、指定容器port
hostport:container	未指定ip port、指定主机port、指定容器port

1. Docker网络模型及Bridge

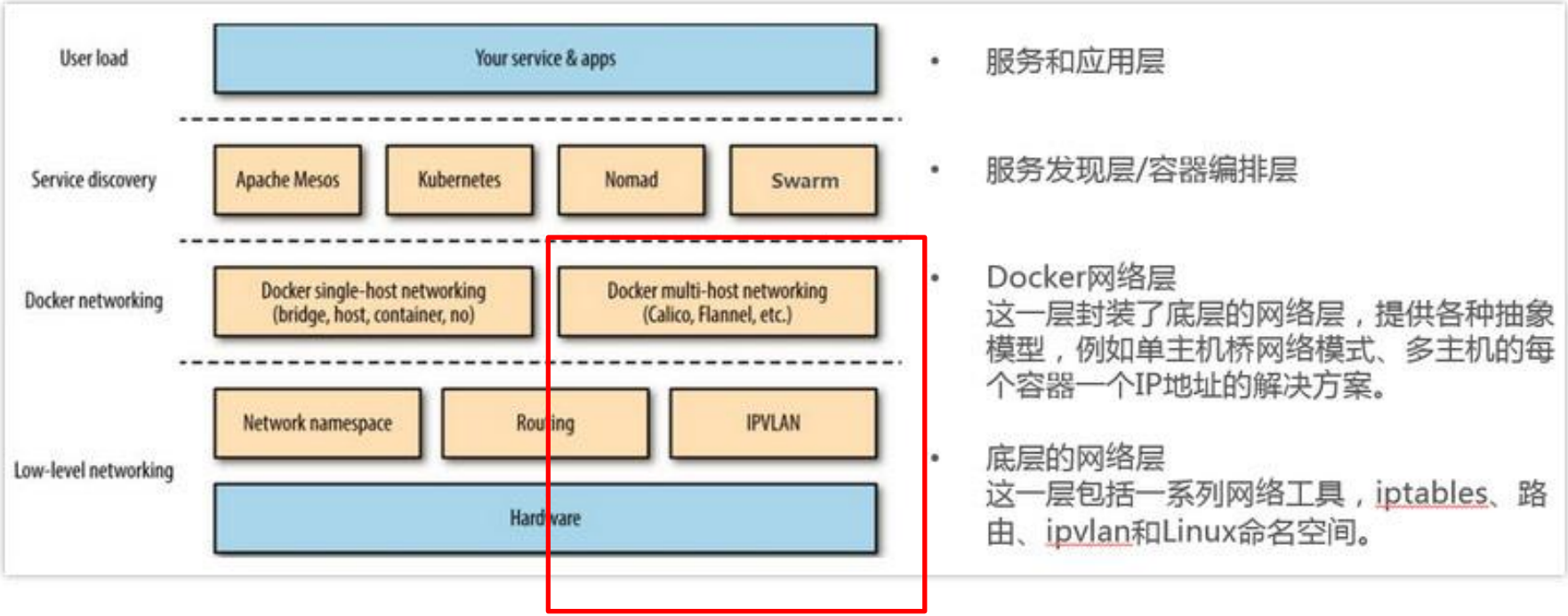


- 容器—虚拟网卡—tup设备—网桥—宿主机协议栈—宿主机网卡
- 容器访问宿主机网络通过SNAT
- 容器绑定宿主机端口，被外部访问，通过DNAT
- 《SNAT/DNAT》

1. Docker网络模型及Bridge

- 《网络分层概述》
- 《Linux网络 - 数据包的接收过程》
- 《Linux网络 - 数据包的发送过程》
- 《Linux虚拟网络设备之tun/tap》
- 《Linux虚拟网络设备之veth》
- 《Linux虚拟网络设备之bridge(桥)》

2. Kubernetes网络模型

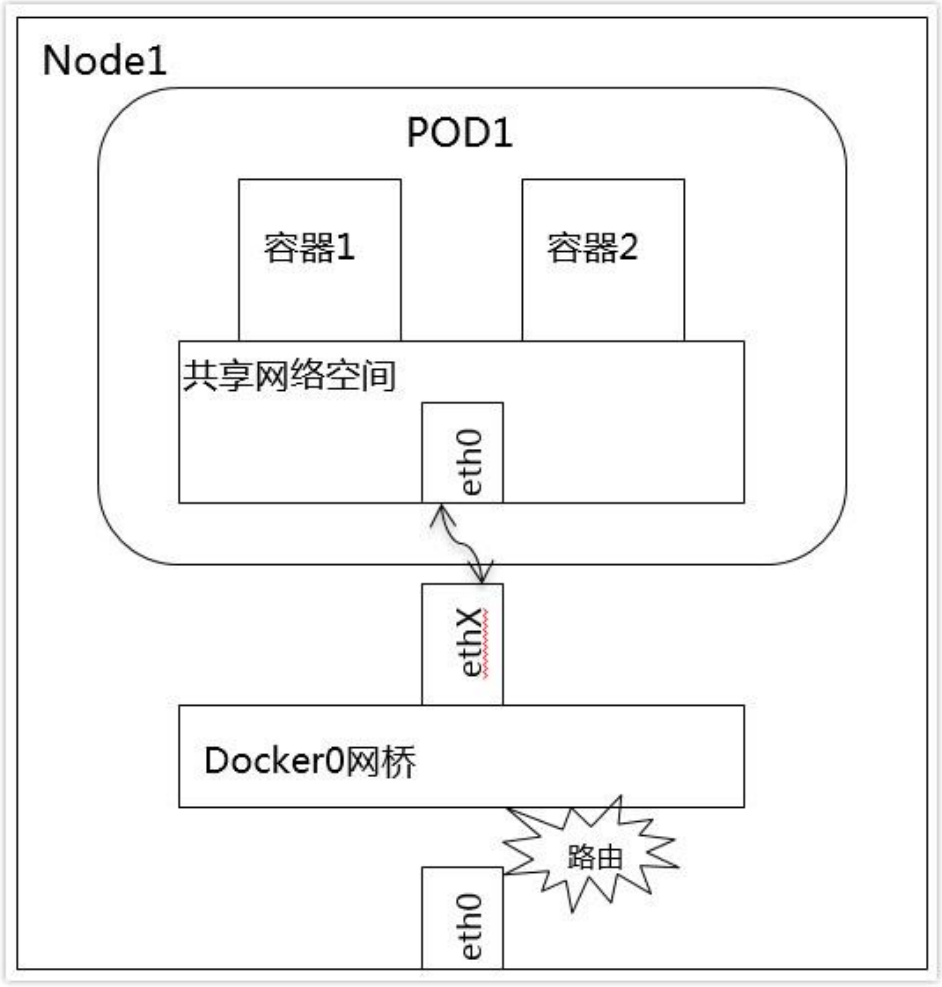


- 服务和应用层
- 服务发现层/容器编排层
- Docker网络层
  这一层封装了底层的网络层，提供各种抽象模型，例如单主机桥网络模式、多主机的每个容器一个IP地址的解决方案。
- 底层的网络层
  这一层包括一系列网络工具，iptables、路由、ipvlan和Linux命名空间。

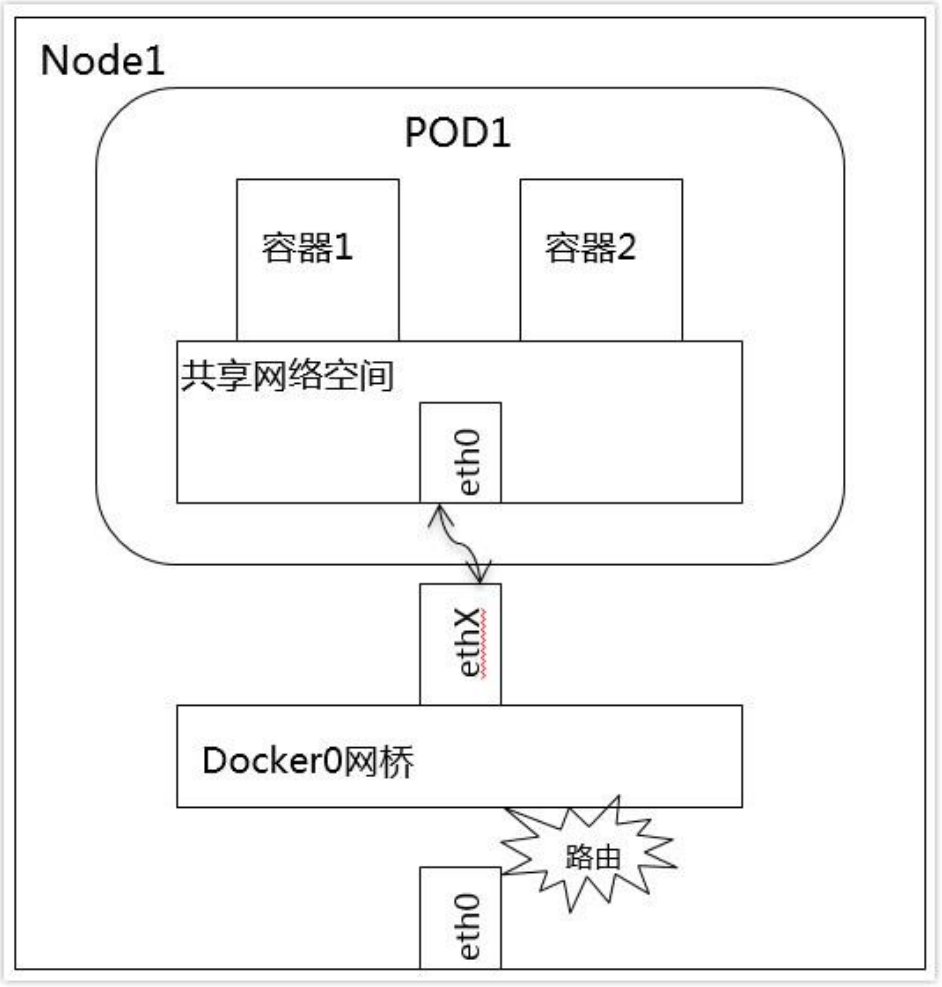
2. Kuberntes网络模型设计理念

- 所有容器不使用 NAT 就可以互相通信（这跟 Docker 的默认实现是不同的）；
- 所有节点跟容器之间不使用 NAT 就可以互相通信；
- 容器自己看到的地址，跟其他人访问自己使用的地址应该是一样的（其实还是在说不要有 NAT）。

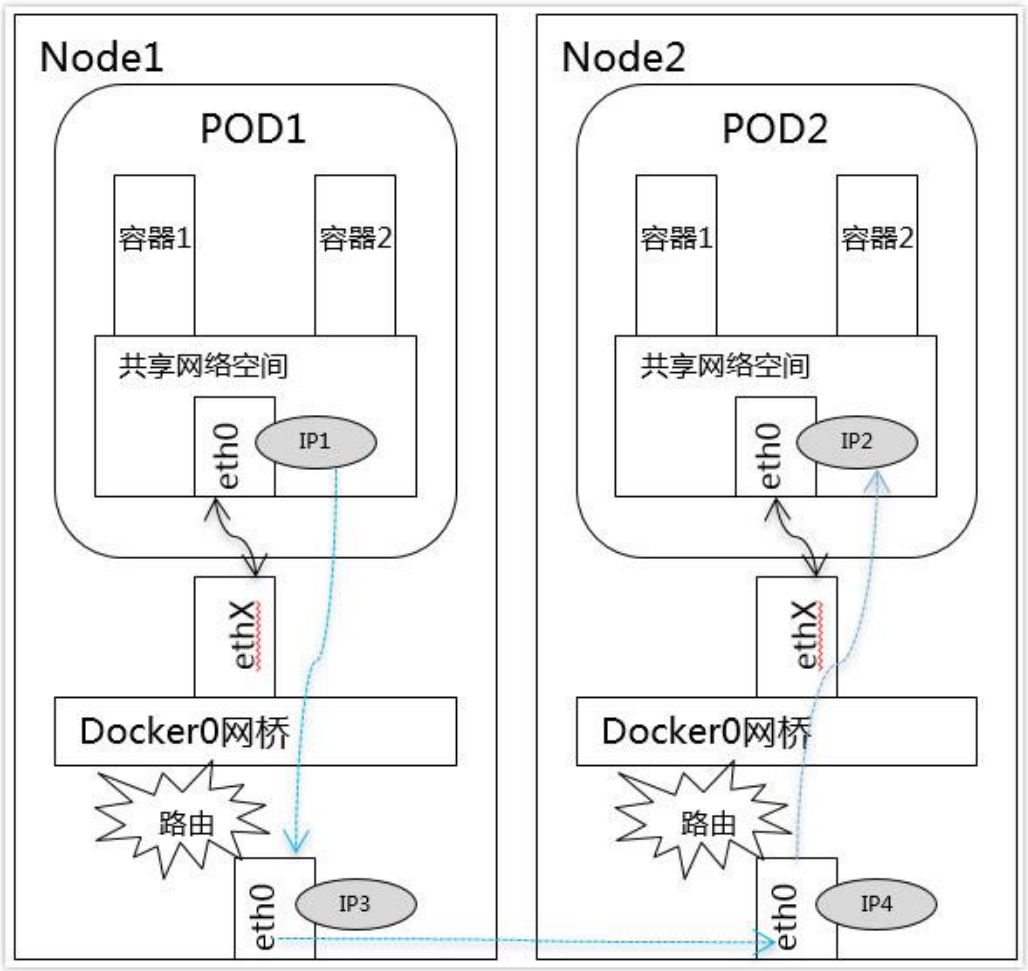
2. Kuberntes网络模型—Pod内



2. Kuberntes网络模型—同节点Pod



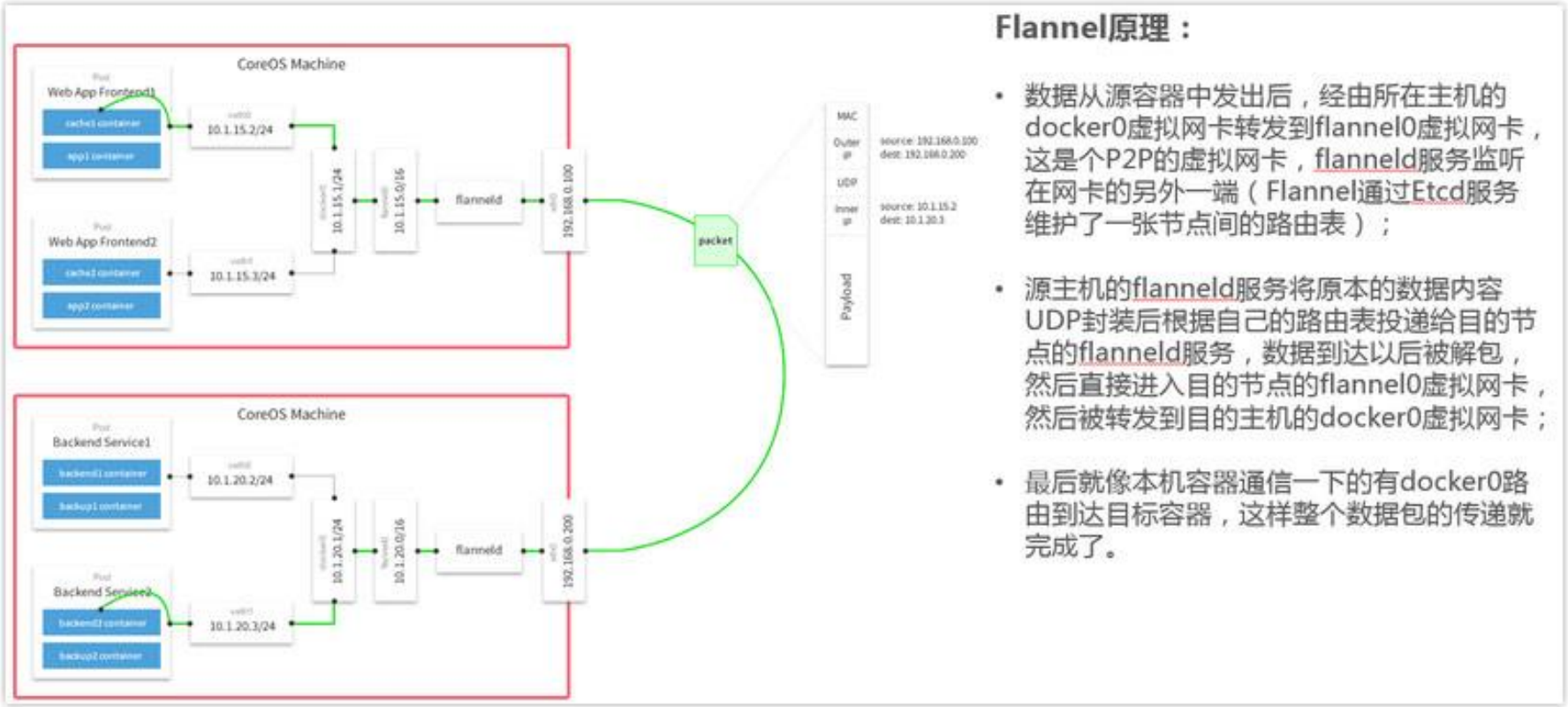
2. Kuberntes网络模型—不同节点Pod



2. 容器网络方案

方案	典型实现
隧道方案（Overlay Networking）	**Flannel**：UDP广播，VxLan
	Weave：UDP广播，本机建立新的BR，通过PCAP互通
	Open vSwitch（OVS）：基于VxLan和GRE协议，但是性能方面损失比较严重
	Racher：IPsec
路由方案（Underlay Networking)	**Calico**：基于BGP协议的路由方案，支持很细致的ACL控制，对混合云亲和度比较高。
	Macvlan：从逻辑和Kernel层来看隔离性和性能最优的方案，基于二层隔离，所以需要二层路由器支持，大多数云服务商不支持，所以混合云上比较难以实现。

2. Flannel



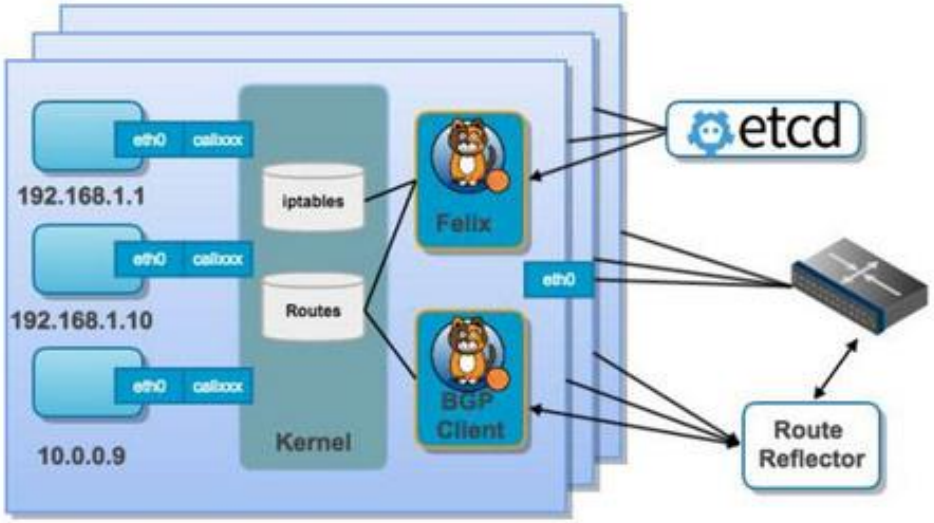
网桥类似于docker，但加了一个桥用于跨机

安装文档：https://feisky.gitbooks.io/sdn/container/flannel/

2. Flannel

- Flannel是CoreOS团队针对Kubernetes设计的一个网络规划服务，简单来说，它的功能是让集群中的不同节点主机创建的Docker容器都具有全集群唯一的虚拟IP地址。
- 在默认的Docker配置中，每个节点上的Docker服务会分别负责所在节点容器的IP分配。这样导致的一个问题是，不同节点上容器可能获得相同的内外IP地址。并使这些容器之间能够之间通过IP地址相互找到，也就是相互ping通。
- Flannel的设计目的就是为集群中的所有节点重新规划IP地址的使用规则，从而使得不同节点上的容器能够获得“同属一个内网”且”不重复的”IP地址，并让属于不同节点上的容器能够直接通过内网IP通信。
- Flannel实质上是一种“覆盖网络(overlaynetwork)”，也就是将TCP数据包装在另一种网络包里面进行路由转发和通信，目前已经支持udp、vxlan、host-gw、aws-vpc、gce和alloc路由等数据转发方式，默认的节点间数据通信方式是UDP转发。

2. Calico



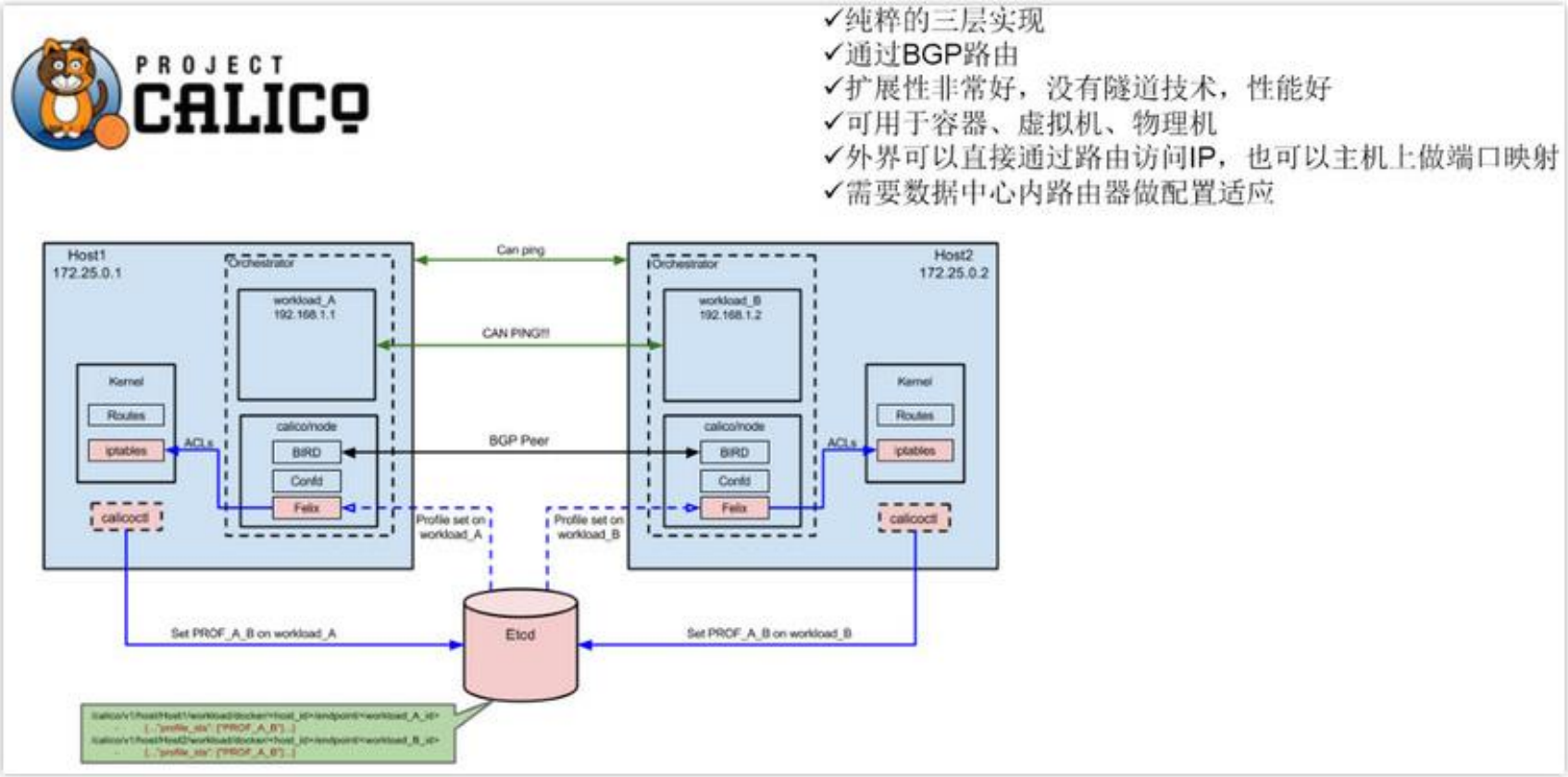
Felix，Calico Agent，跑在每台需要运行Workload的节点上，主要负责配置路由及ACLs等信息来确保Endpoint的连通状态；

etcd，分布式键值存储，主要负责网络元数据一致性，确保Calico网络状态的准确性；

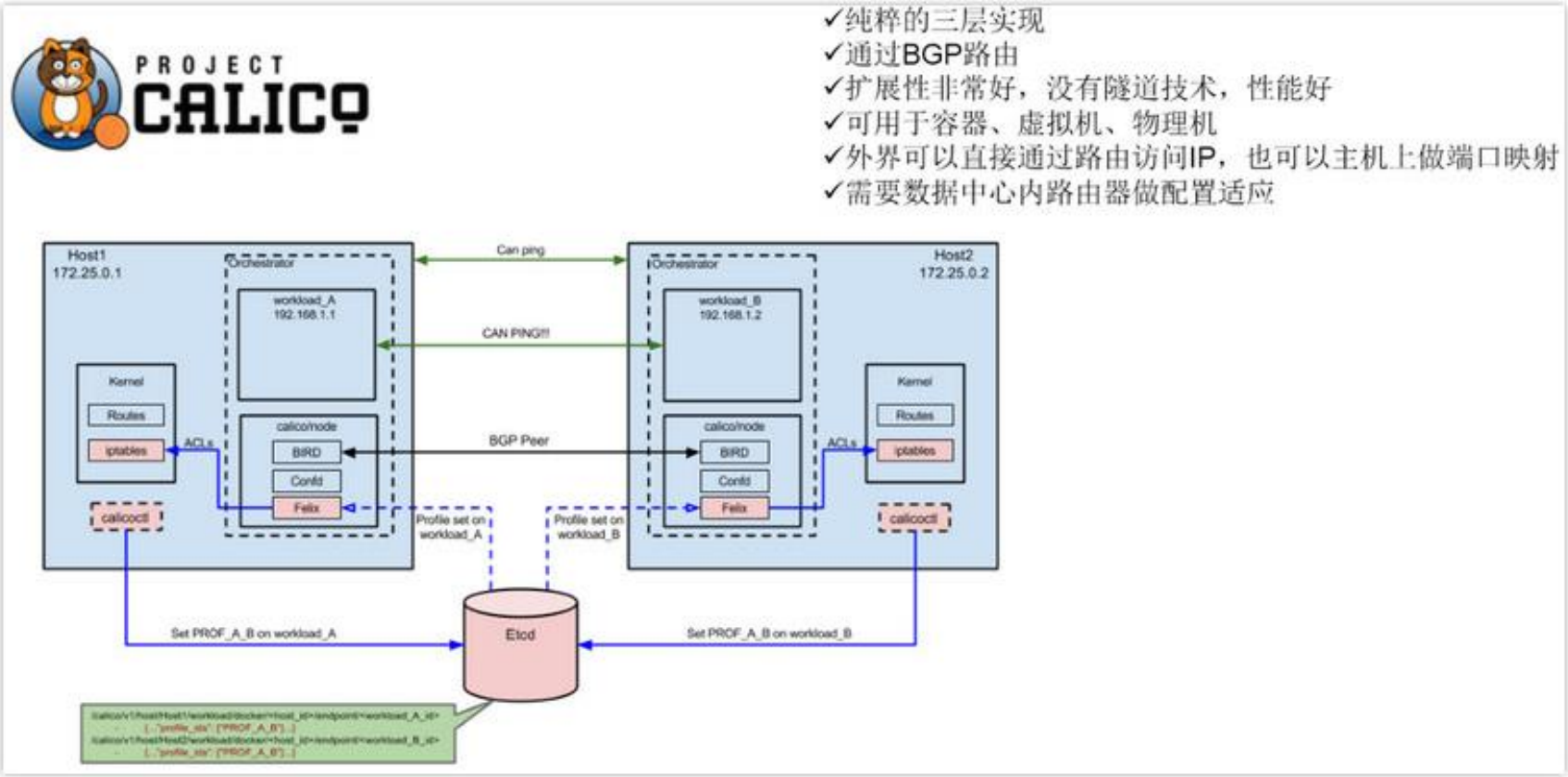
BGP Client（BIRD），主要负责把Felix写入Kernel的路由信息分发到当前Calico网络，确保Workload间的通信的有效性；

BGP Route Reflector（BIRD），大规模部署时使用，摒弃所有节点互联的 mesh 模式，通过一个或者多个BGP Route Reflector来完成集中式的路由分发。

2. Calico


PROJECT
CALICO
✓纯粹的三层实现
✓通过BGP路由
✓扩展性非常好，没有隧道技术，性能好
✓可用于容器、虚拟机、物理机
✓外界可以直接通过路由访问IP，也可以主机上做端口映射
✓需要数据中心内路由器做配置适应
Host1
172.25.0.1
Orchestrator
workload_A
192.168.1.1
Kernel
Routes
iptables
ACLs
calico/node
BIRD
Confd
Felix
calicoctl
Can ping
CAN PING!!!
BGP Peer
Profile set on workload_A
Profile set on workload_B
Host2
172.25.0.2
Orchestrator
workload_B
192.168.1.2
Kernel
Routes
iptables
ACLs
calico/node
BIRD
Confd
Felix
calicoctl
Set PROF_A_B on workload_A
Etcd
Set PROF_A_B on workload_B

2. Calico


PROJECT
CALICO
✓纯粹的三层实现
✓通过BGP路由
✓扩展性非常好，没有隧道技术，性能好
✓可用于容器、虚拟机、物理机
✓外界可以直接通过路由访问IP，也可以主机上做端口映射
✓需要数据中心内路由器做配置适应
Host1
172.25.0.1
Orchestrator
workload_A
192.168.1.1
Kernel
Routes
iptables
ACLs
calico/node
BIRD
Confd
Felix
calicoctl
Can ping
CAN PING!!!
BGP Peer
Profile set on workload_A
Profile set on workload_B
Host2
172.25.0.2
Orchestrator
workload_B
192.168.1.2
Kernel
Routes
iptables
ACLs
calico/node
BIRD
Confd
Felix
calicoctl
Set PROF_A_B on workload_A
Etcd
Set PROF_A_B on workload_B

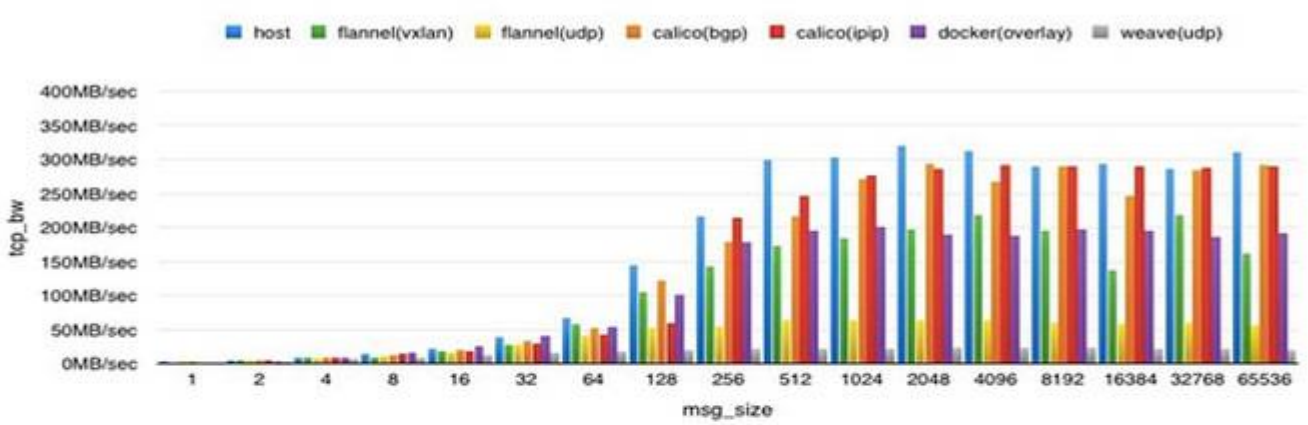
2. Calico

- Calico是一个纯3层的数据中心网络方案，而且无缝集成像OpenStack这种IaaS云架构，能够提供可控的VM、容器、裸机之间的IP通信。Calico不使用重叠网络比如flannel和libnetwork重叠网络驱动，它是一个纯三层的方法，使用虚拟路由代替虚拟交换，每一台虚拟路由通过BGP协议传播可达信息（路由）到剩余数据中心。
- Calico在每一个计算节点利用Linux Kernel实现了一个高效的vRouter来负责数据转发，而每个vRouter通过BGP协议负责把自己上运行的workload的路由信息像整个Calico网络内传播——小规模部署可以直接互联，大规模下可通过指定的BGP route reflector来完成。
- Calico节点组网可以直接利用数据中心的网络结构（无论是L2或者L3），不需要额外的NAT，隧道或者Overlay Network。
- Calico基于iptables还提供了丰富而灵活的网络Policy，保证通过各个节点上的ACLs来提供Workload的多租户隔离、安全组以及其他可达性限制等功能。

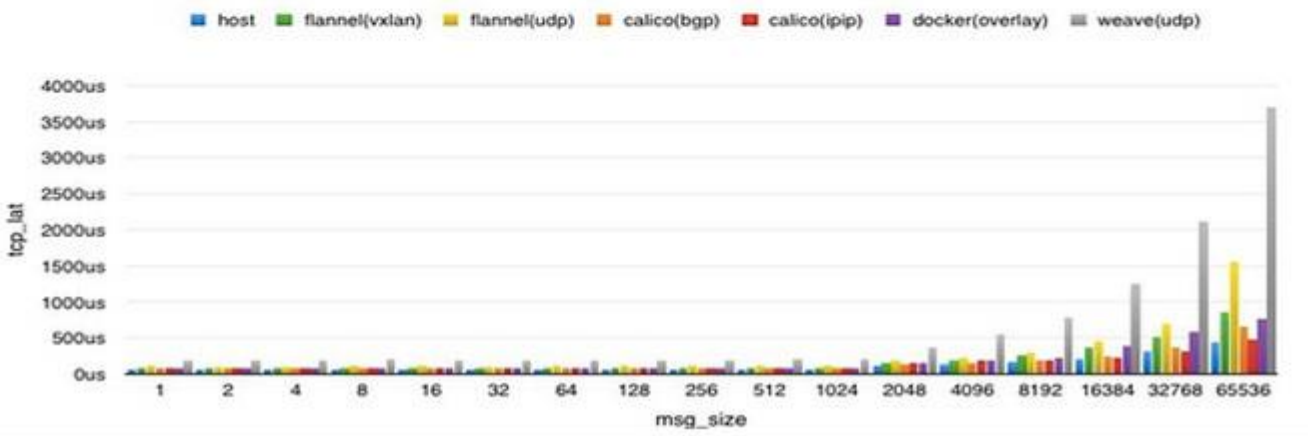
2. 网络对比

CPU压力： host < calico(bgp) < calico(ipip) = flannel(vxlan) = docker(vxlan) < flannel(udp) < weave(udp)

带宽：



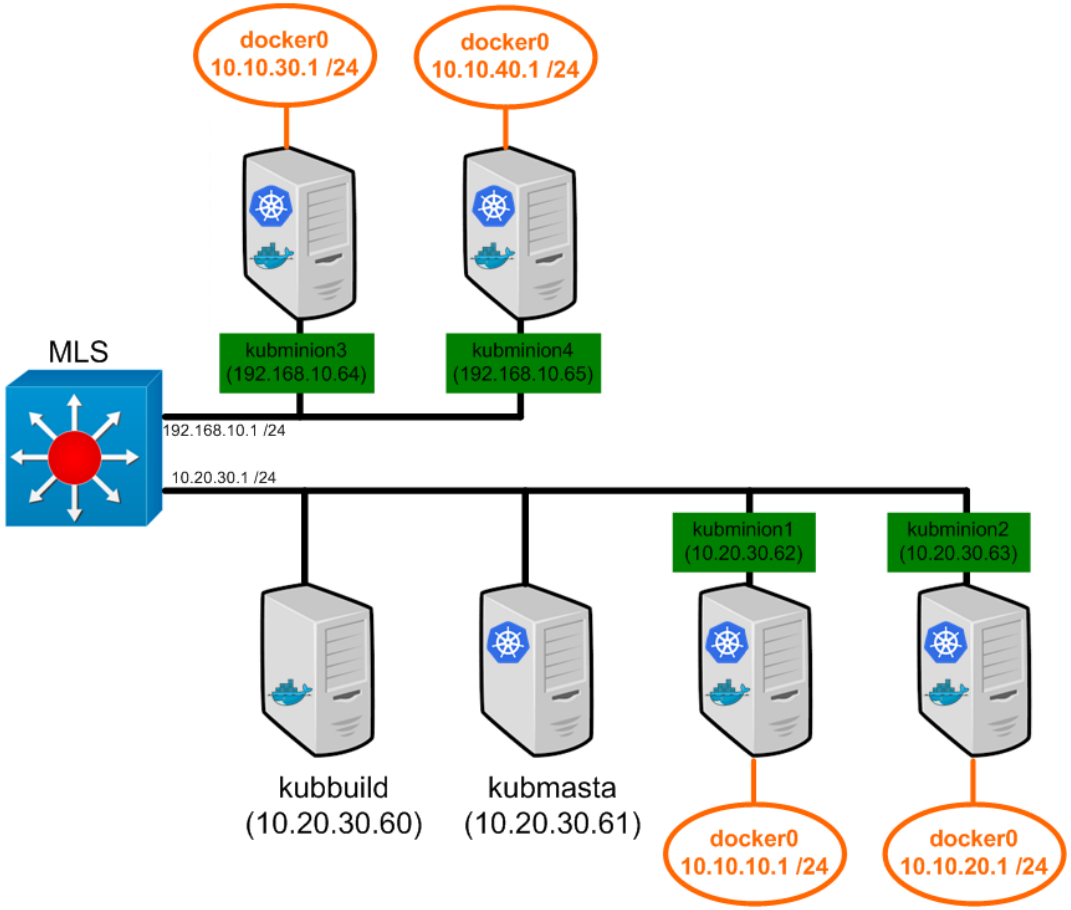
延迟：



2. 网络对比

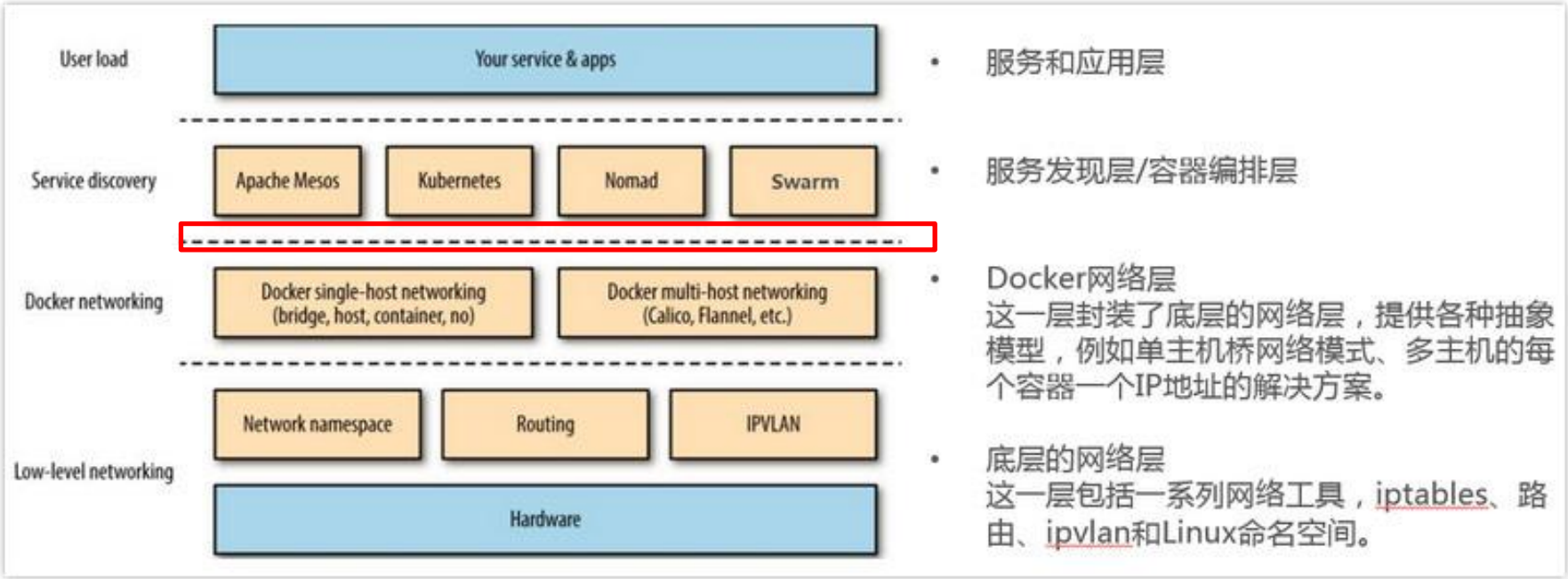
方案	结论	优势	劣势
Calico	calico 的 2 个方案都有不错的表现，其中 ipip 的方案在 big msg size 上表现更好，但骚骁是在 128 字节的时候表现异常，多次测试如此。bgp 方案比较稳定，CPU 消耗并没有 ipip 的大，当然带宽表现也稍微差点。不过整体上来说，无论是 bgp 还是 ipip tunnel，calico 这套 overlay sdn 的解决方案成熟度和可用度都相当不错，为云上第一选择。	性能好，可控性高，隔离性好	操作起来还是比较复杂，比如对 iptables 有依赖
Flannel	flannel 的 2 个方案表现也凑合，其中 vxlan 方案是因为没法开 udp offload 导致性能偏低，其他的测试报告来看，一旦让网卡自行解 udp 包拿到 mac 地址什么的，性能基本上可以达到无损，同时 cpu 占用率相当好。udp 方案受限于 user space 的解包，仅仅比 weave(udp) 要好一点点。	部署简单，性能还行	没法实现固定 IP 的容器漂移，没法多子网隔离，对上层设计依赖度高，没有 IPAM，IP地址浪费，对 docker 启动方法有绑定
docker 原生 Overlay	其实也是基于 vxlan 实现的。受限于 cloud 上不一定会开的网卡 udp offload，vxlan 方案的性能上限就是裸机的 55% 左右了。大体表现上与 flannel vxlan 方案几乎一致。	docker 原生，性能凑合	对内核要求高（>3.16），对 docker daemon 有依赖需求（consul / etcd），本身驱动实现还是略差点，可以看到对 cpu 利用率和带宽比同样基于 vxlan 的 flannel 要差一些，虽然有 api 但对 network 以及多子网隔离局部交叉这种需求还是比较麻烦，IPAM 很差

2. 一个典型的跨网段网络方案

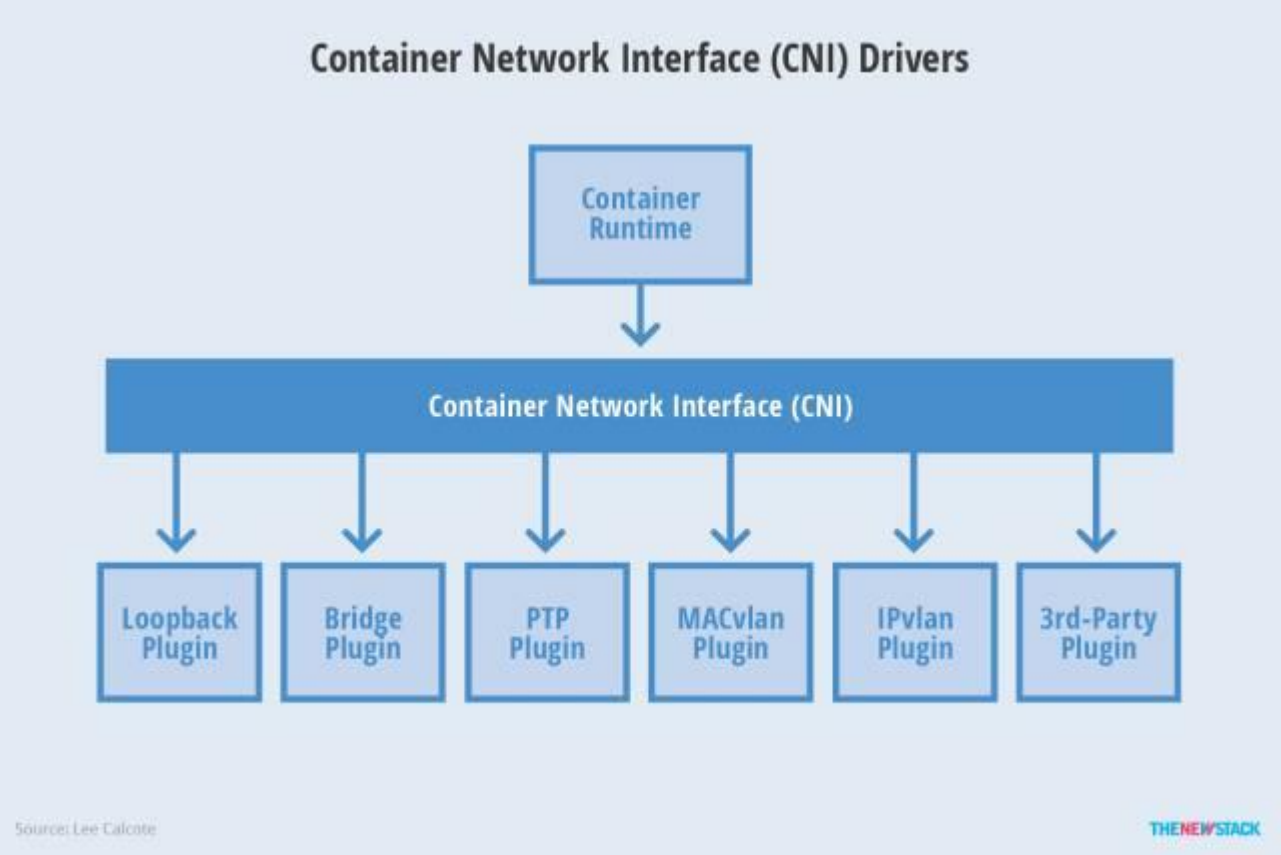


http://www.open-open.com/lib/view/open1437616369584.html

3. CNI vs CNM


User load
Your service & apps
Service discovery
Apache Mesos
Kubernetes
Nomad
Swarm
Docker networking
Docker single-host networking (bridge, host, container, no)
Docker multi-host networking (Calico, Flannel, etc.)
Low-level networking
Network namespace
Routing
IPVLAN
Hardware
服务和应用层
服务发现层/容器编排层
Docker网络层
这一层封装了底层的网络层，提供各种抽象模型，例如单主机桥网络模式、多主机的每个容器一个IP地址的解决方案。
底层的网络层
这一层包括一系列网络工具，iptables、路由、ipvlan和Linux命名空间。

3. CNI



[docker的网络-Container network interface(CNI)与Container network model(CNM)]

3. CNI ADD/DELETE

CNI的接口设计的非常简洁，只有两个接口ADD/DELETE。

以 ADD接口为例

Add container to network
参数主要包括:

- Version. CNI版本号
- Container ID. 这是一个可选的参数，提供容器的id
- Network namespace path. 容器的命名空间的路径，比如/proc/[pid]/ns/net。
- Network configuration. 这是一个json的文档，具体可以参看network-configuration
- Extra arguments. 其他参数
- Name of the interface inside the container. 容器内的网卡名

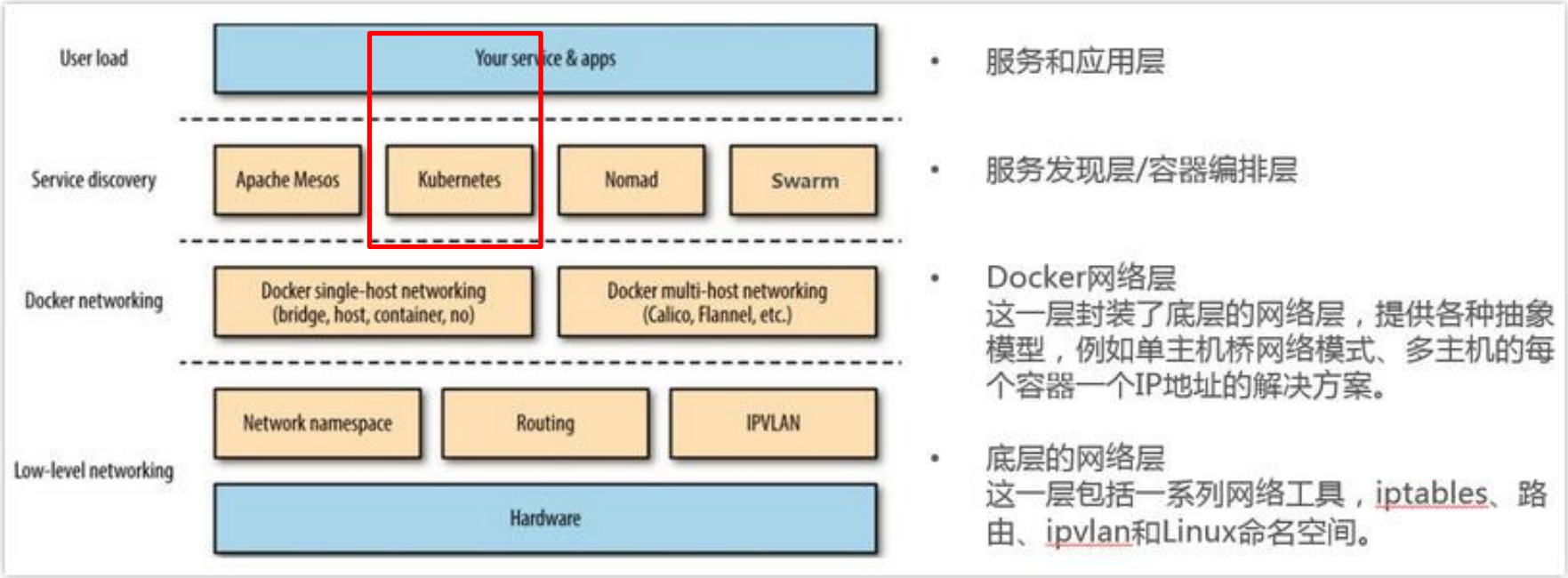
返回值:

- IPs assigned to the interface. ipv4或者ipv6地址
- DNS information. DNS相关信息

3. CNI

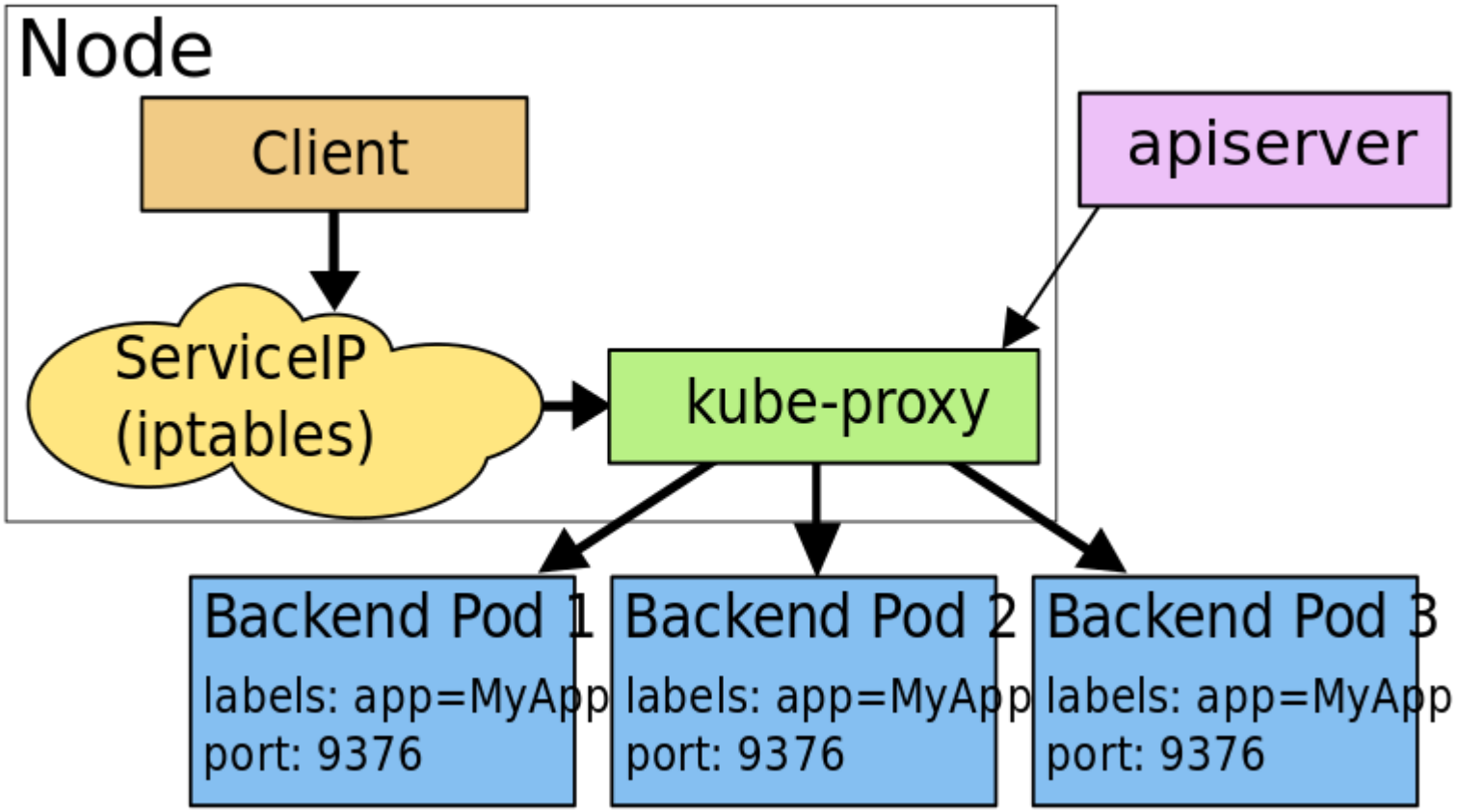
```
mkdir -p /etc/cni/net.d
$ cat >/etc/cni/net.d/10-calico.conf <<EOF
{
    "name": "calico-k8s-network",
    "type": "calico",
    "etcd_authority": "<ETCD_IP>:<ETCD_PORT>",
    "log_level": "info",
    "ipam": {
        "type": "calico-ipam"
    },
    "policy": {
        "type": "k8s"
    }
}
EOF
```

4. Service和kube-dns

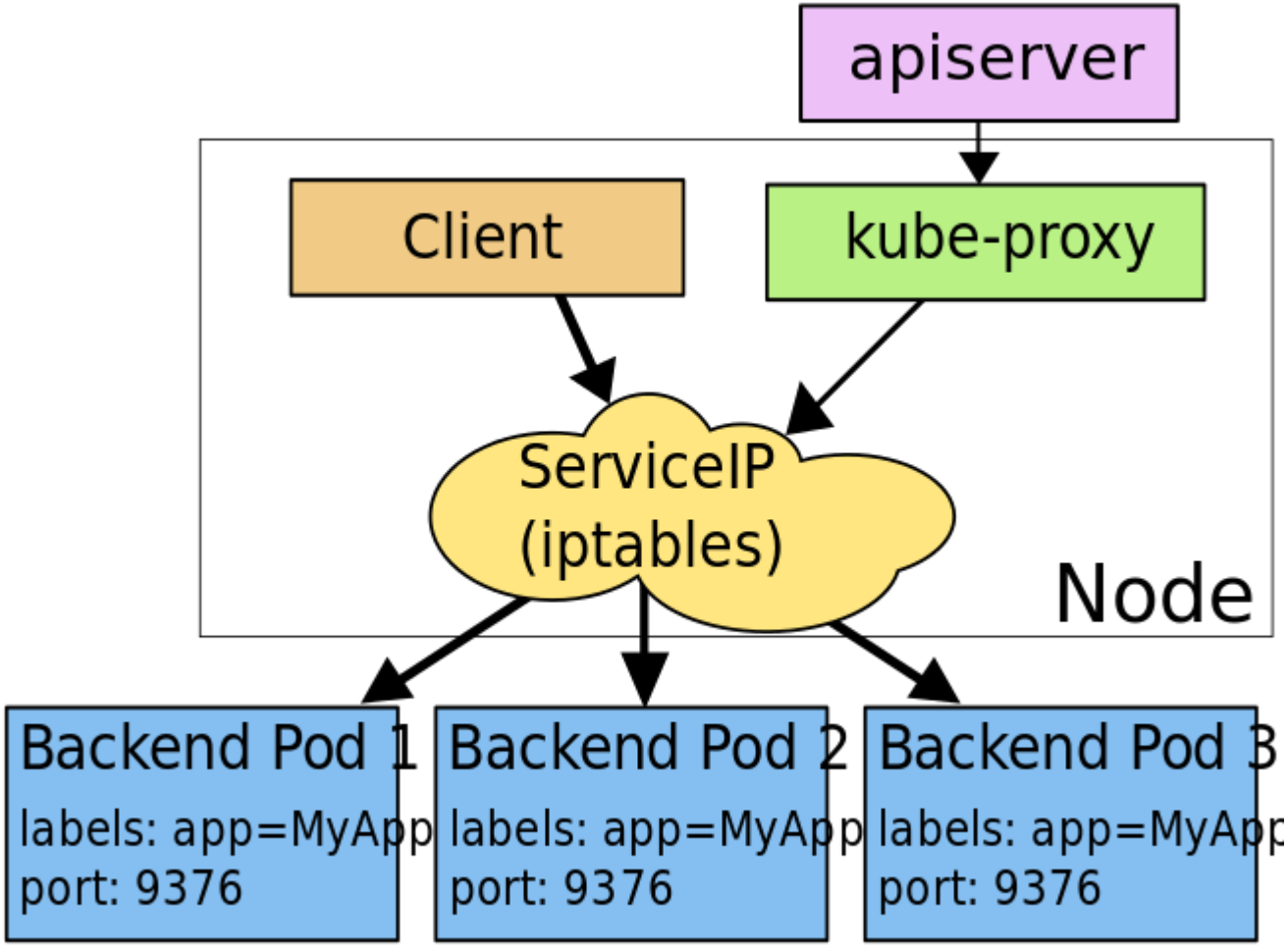


- 服务和应用层
- 服务发现层/容器编排层
- Docker网络层
  这一层封装了底层的网络层，提供各种抽象模型，例如单主机桥网络模式、多主机的每个容器一个IP地址的解决方案。
- 底层的网络层
  这一层包括一系列网络工具，iptables、路由、ipvlan和Linux命名空间。

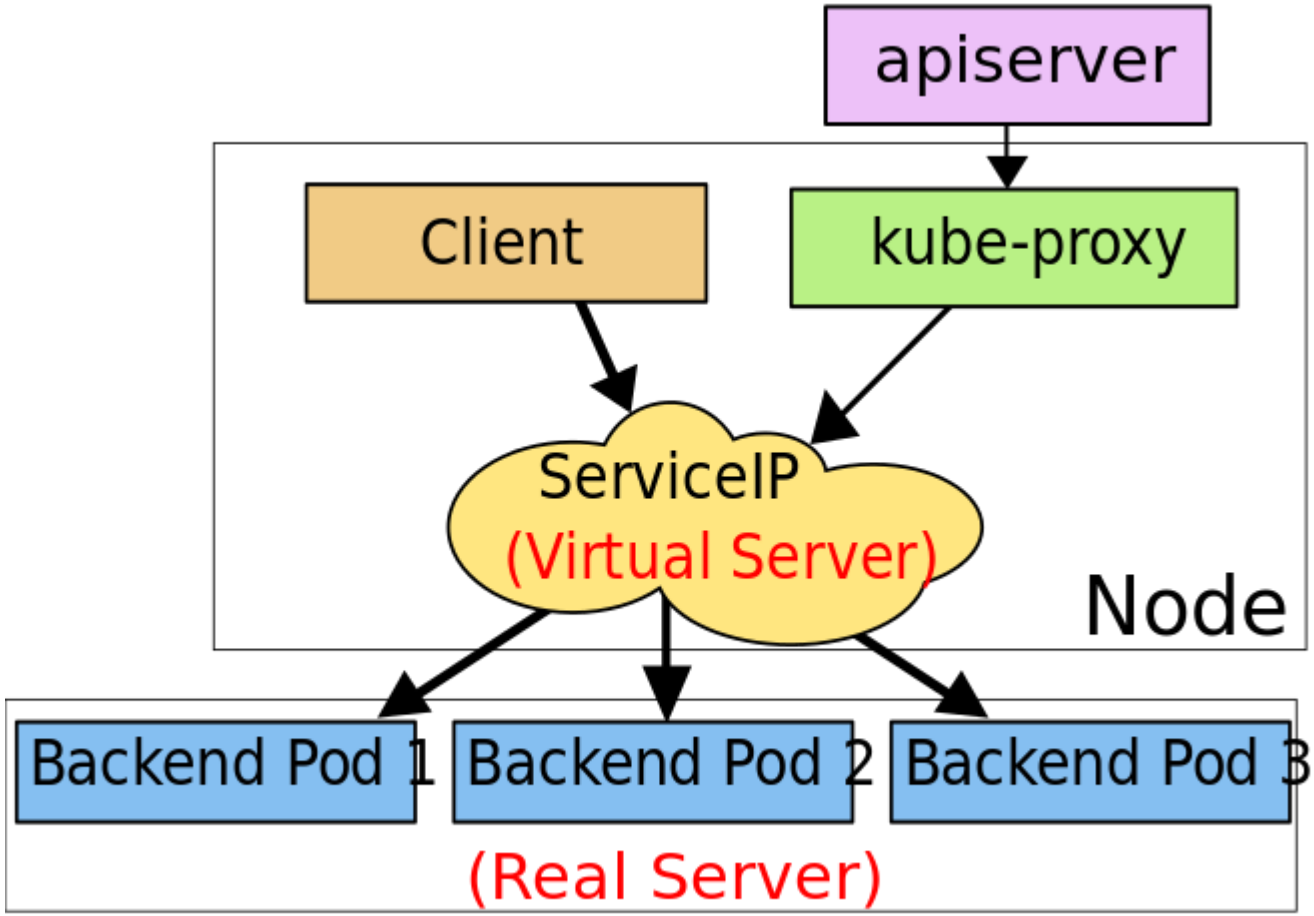
4.Service： kube-proxy： Proxy-mode: userspace



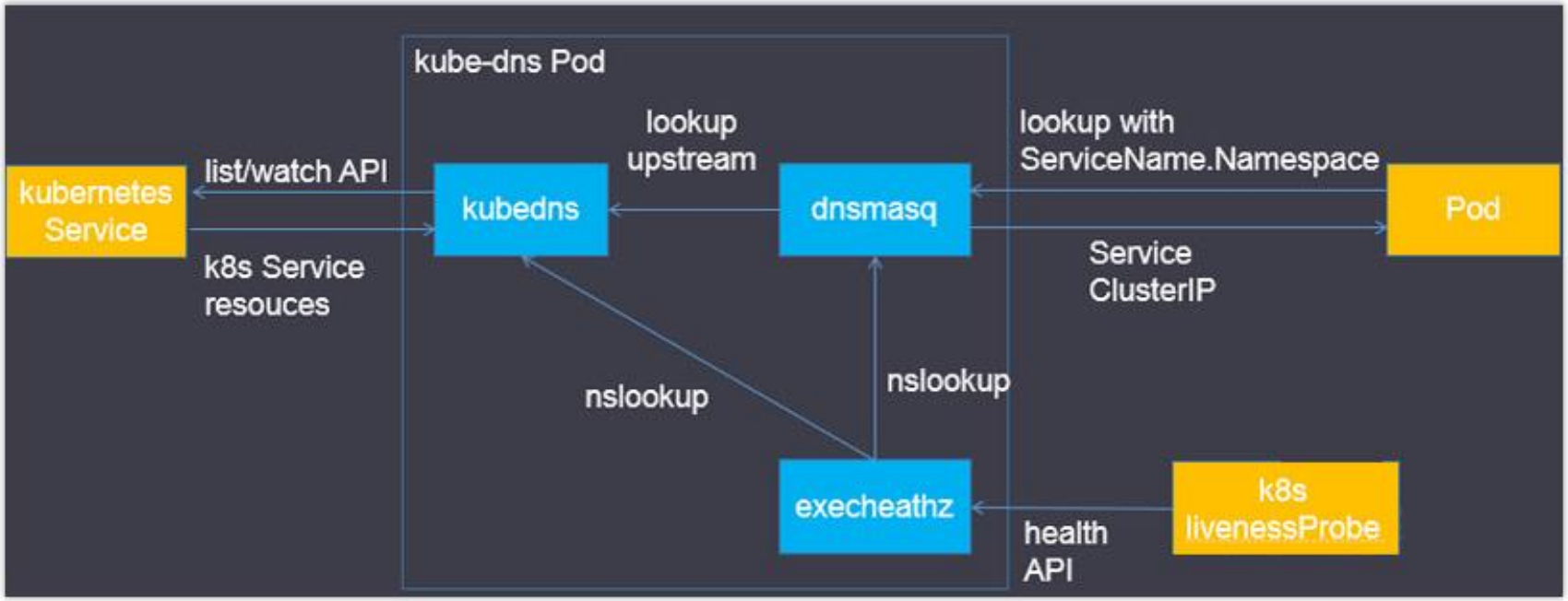
4. Service： kube-proxy： Proxy-mode: iptables



4. Service： kube-proxy： Proxy-mode: ipvs(1.9 beta)

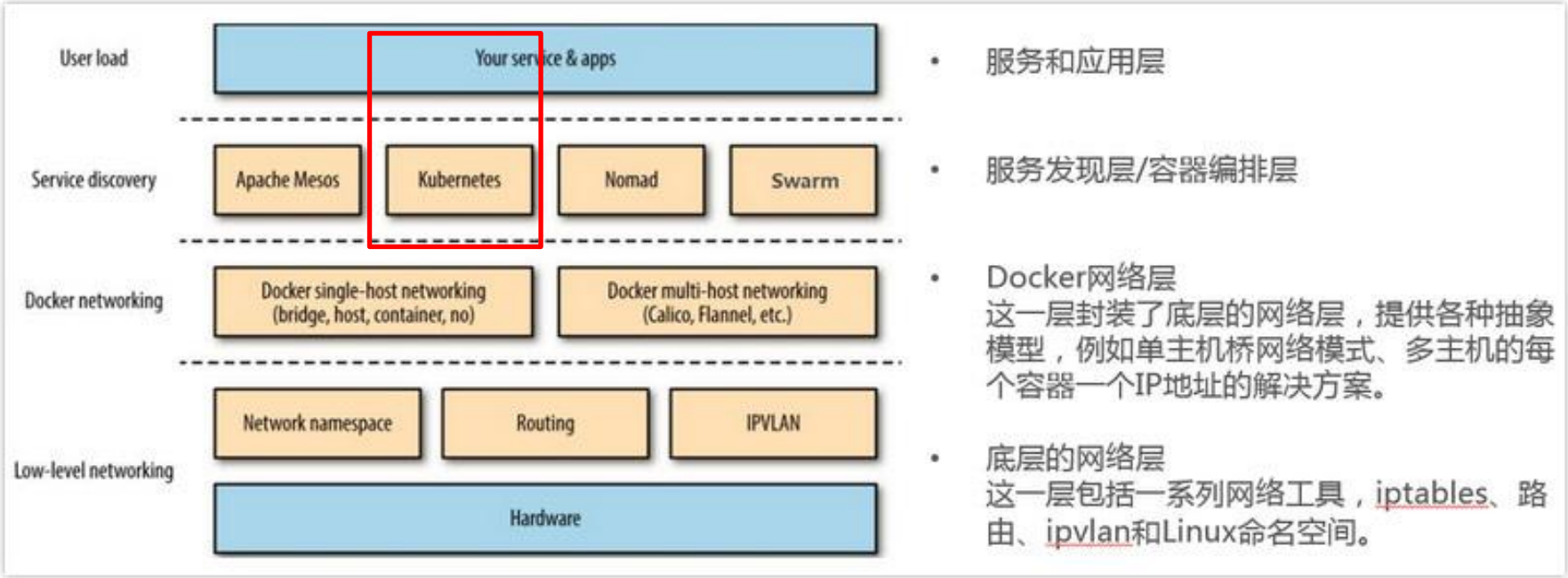


4. kube-dns



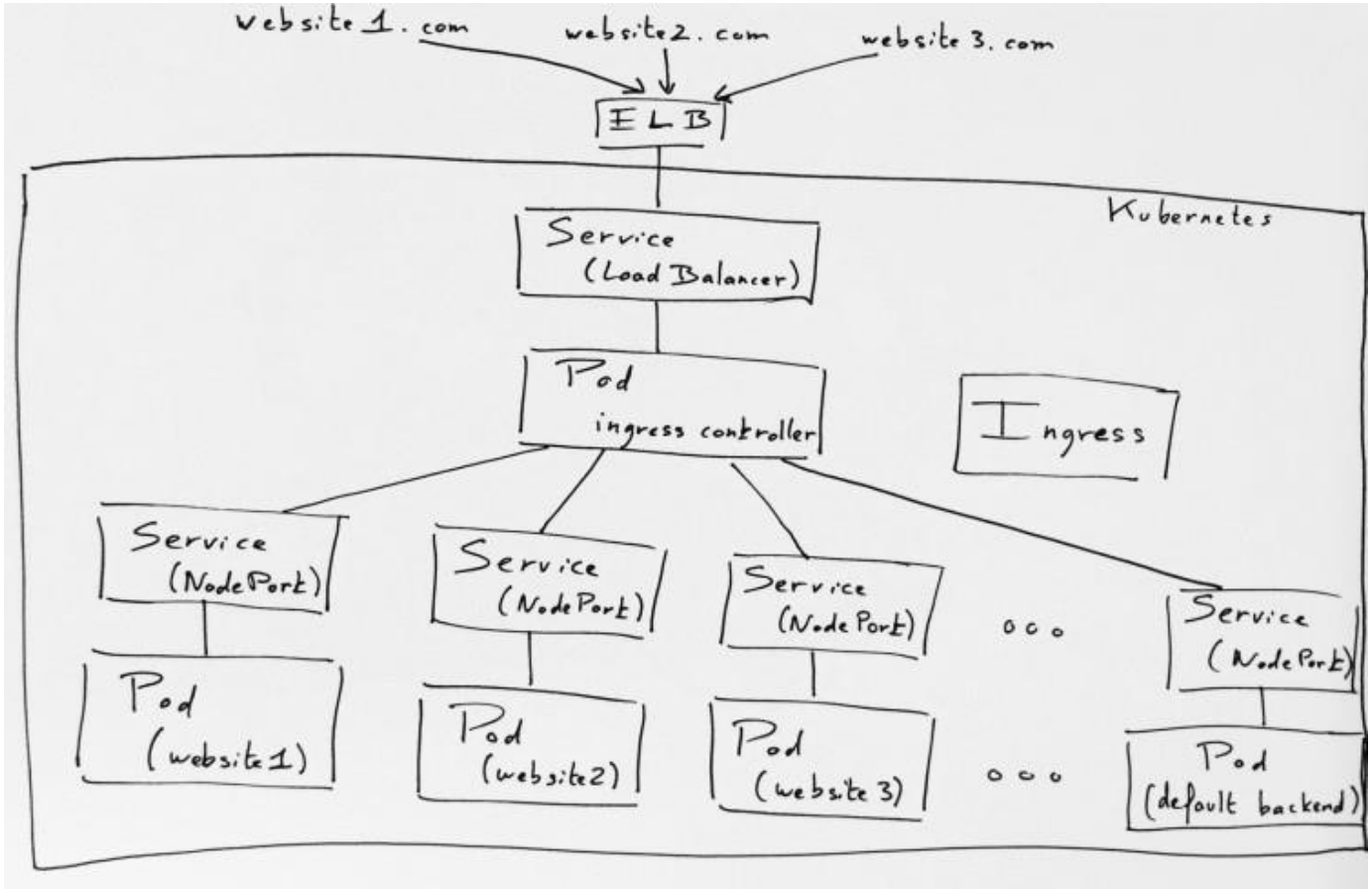
《kube-dns的前世今生》

5. Ingress



- 服务和应用层
- 服务发现层/容器编排层
- Docker网络层
  这一层封装了底层的网络层，提供各种抽象模型，例如单主机桥网络模式、多主机的每个容器一个IP地址的解决方案。
- 底层的网络层
  这一层包括一系列网络工具，iptables、路由、ipvlan和Linux命名空间。

5. Ingress



《Ingress解析》

作业

- 使用kubectl把上面的各种应用类型都玩一遍

联系我们

小象学院：互联网新技术在线教育领航者

- 微信公众号：大数据分析挖掘
- 新浪微博：ChinaHadoop