欢迎参加IBM开源技术微讲堂

第五讲 网络管理

“Kubenetes”系列公开课

- 每周四晚8点档

1. Kubernetes 初探
2. 上手 Kubernetes
3. Kubernetes 的资源调度
4. Kubernetes 的运行时
5. **Kubernetes 的网络管理**
6. Kubernetes 的存储管理
7. Kubernetes 的日志与监控
8. Kubernetes 的应用部署
9. 扩展 Kubernetes 生态
10. Kubernetes 的企业实践

课程Wiki：http://ibm.biz/opentech-ma

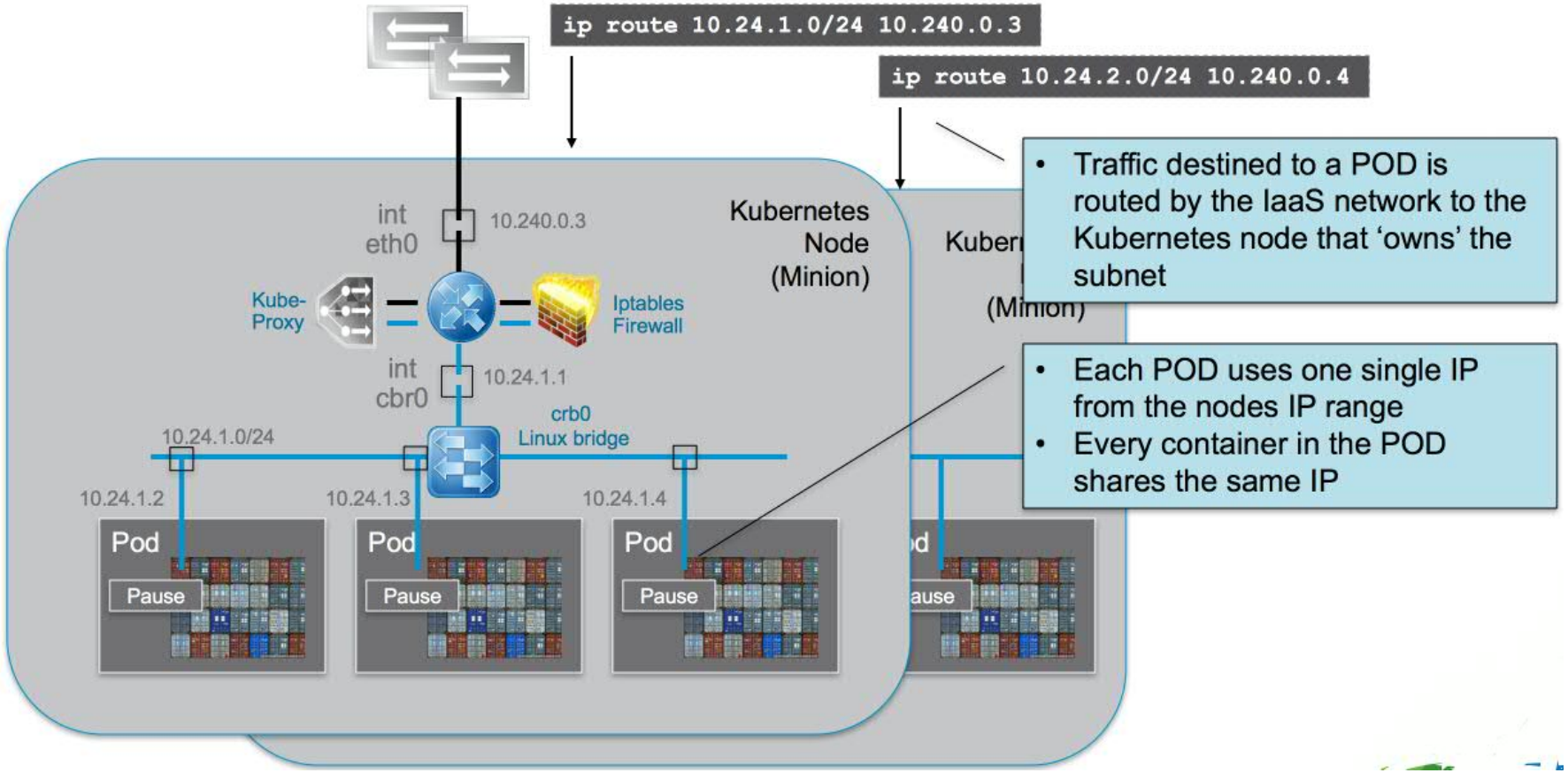
Kubernetes网络管理

邱见

Kubernetes 网络

- 容器网络
  - 每个POD分配单独的Ip地址
  - 使用NetworkPolicy实现访问控制
- 负载均衡（Service IP）
  - 内部的虚拟Ip映射到后端多个容器
- 外部访问
  - 外部负载均衡
  - Ingress的反向代理

Kubernetes 容器网络

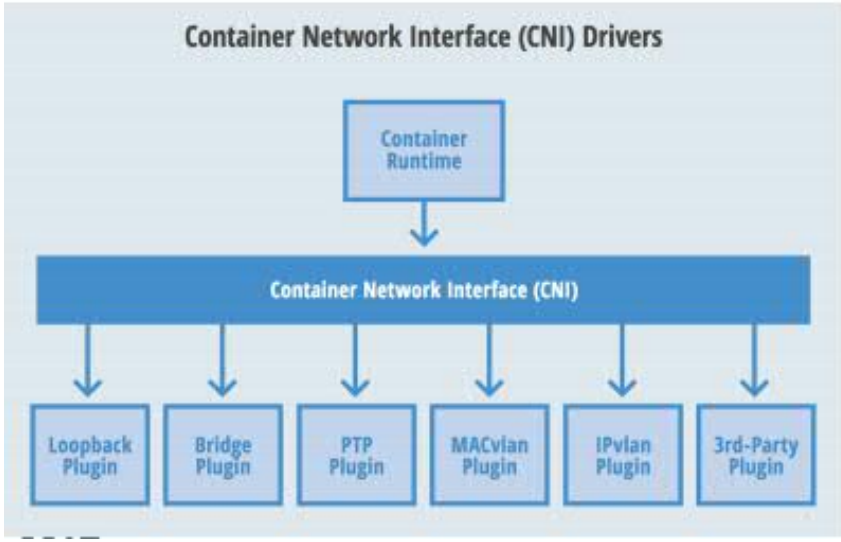


Container Network Interface (CNI) 容器网络

- Kubernetes 使用CNI插件来组建容器网络
- 当POD的创建和销毁时，kubernetes将调用CNI插件的接口来生成网络配置
- CNI插件将生成虚拟NIC，将其挂载在主机网络之上，并和POD的namespace关联

```
{
    "name": "net",
    "type": "bridge",
    "bridge": "br-int",
    "isGateway": true,
    "ipMasq": false,
    "ipam": {
        "type": "host-local",
        "subnet": "10.96.0.64/26"
    }
}
```

/etc/cni/net.d/10-bridge.conf



Kubernetes CNI 网络实现

- 在Kubernetes创建Pod后 CNI提供网络的过程主要分三个步骤：
  - Kubelet runtime创建network namespace
  - Kubelete调用CNI插件，制定网络类型。网络类型决定哪一个CNI plugin将会被使用。
  - CNI调用相应网络类型的插件
  - CNI插件将创建veth pair, 检查IPAM类型和数据，触发IPAM插件，获取空闲的Ip地址并将地址分配给容器的网络接口

CNI创建网络步骤1

- Kubelet runtime创建network namespace

```
$ cd /var/lib/cni
$ touch myns
$ unshare -n mount --bind /proc/self/ns/net myns
```

CNI创建网络步骤2

- Kubelete触发CNI插件

```
$ export CNI_COMMAND=ADD
$ export CNI_NETNS=/var/lib/cni/myns
$ export CNI_CONTAINERID=5248e9f8-3c91-11e5-...
$ export CNI_IFNAME=eth0

$ $CNI_PATH/bridge </etc/cni/net.d/10-mynet.conf
```

CNI创建网络步骤3

- CNI插件内部创建veth pair

```
$ brctl addbr mynet
$ ip link add veth123 type veth peer name $CNI_IFNAME
$ brctl addif mynet veth123
$ ip link set $CNI_IFNAME netns $CNI_IFNAME
$ ip link set veth123 up
```

- 通过IPAM插件获取空闲的Ip地址

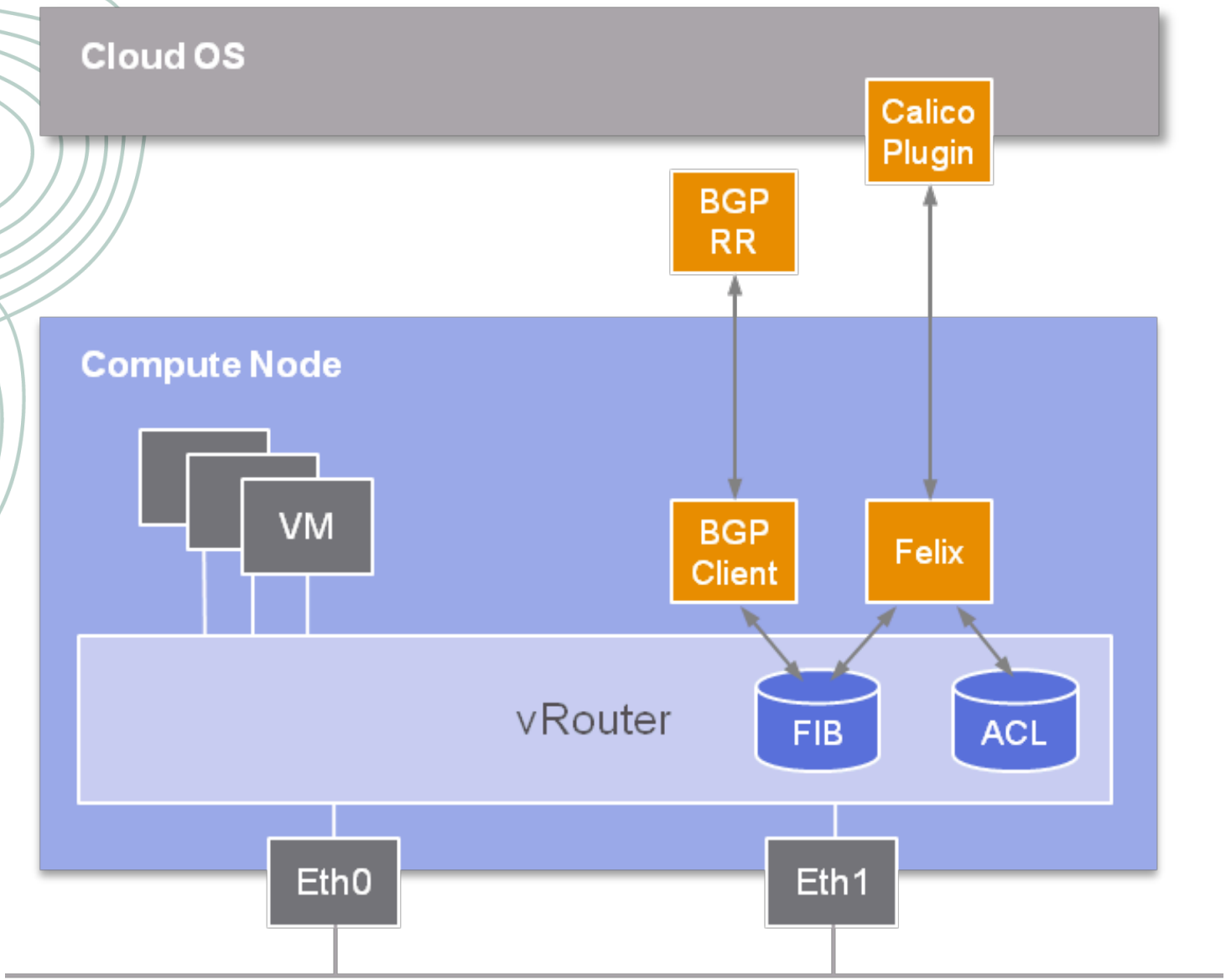

```
$ brctl addbr mynet
$ ip link add veth123 type veth peer name $CNI_IFNAME
$ brctl addif mynet veth123
$ ip link set $CNI_IFNAME netns $CNI_IFNAME
$ ip link set veth123 up
```


- 将ip配置到容器net namespace的网络设备

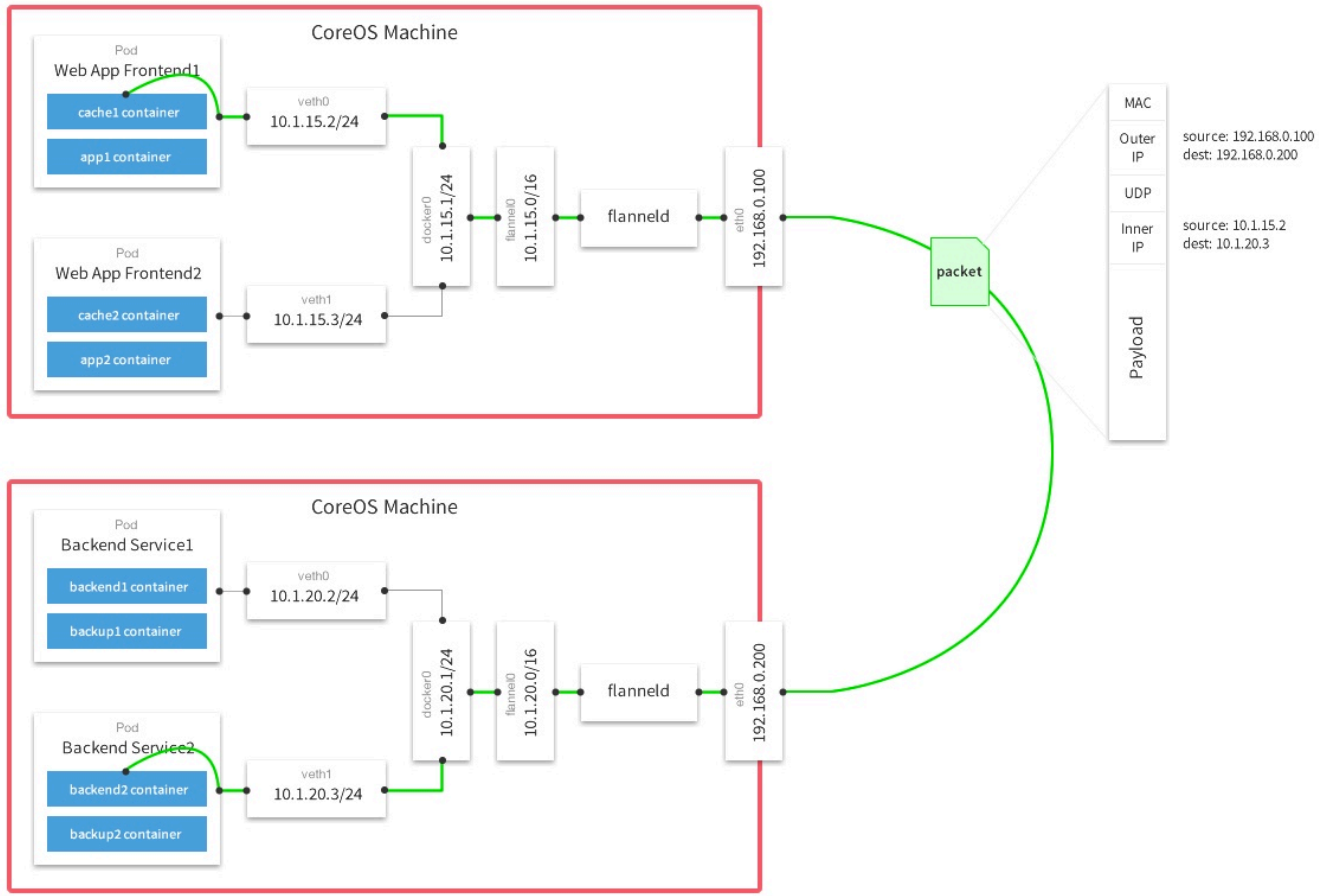
```
# switch to container namespace

$ ip addr add 10.0.5.9/16 dev $CNI_IFNAME

# Finally, print IPAM result JSON to stdout
```

Calico

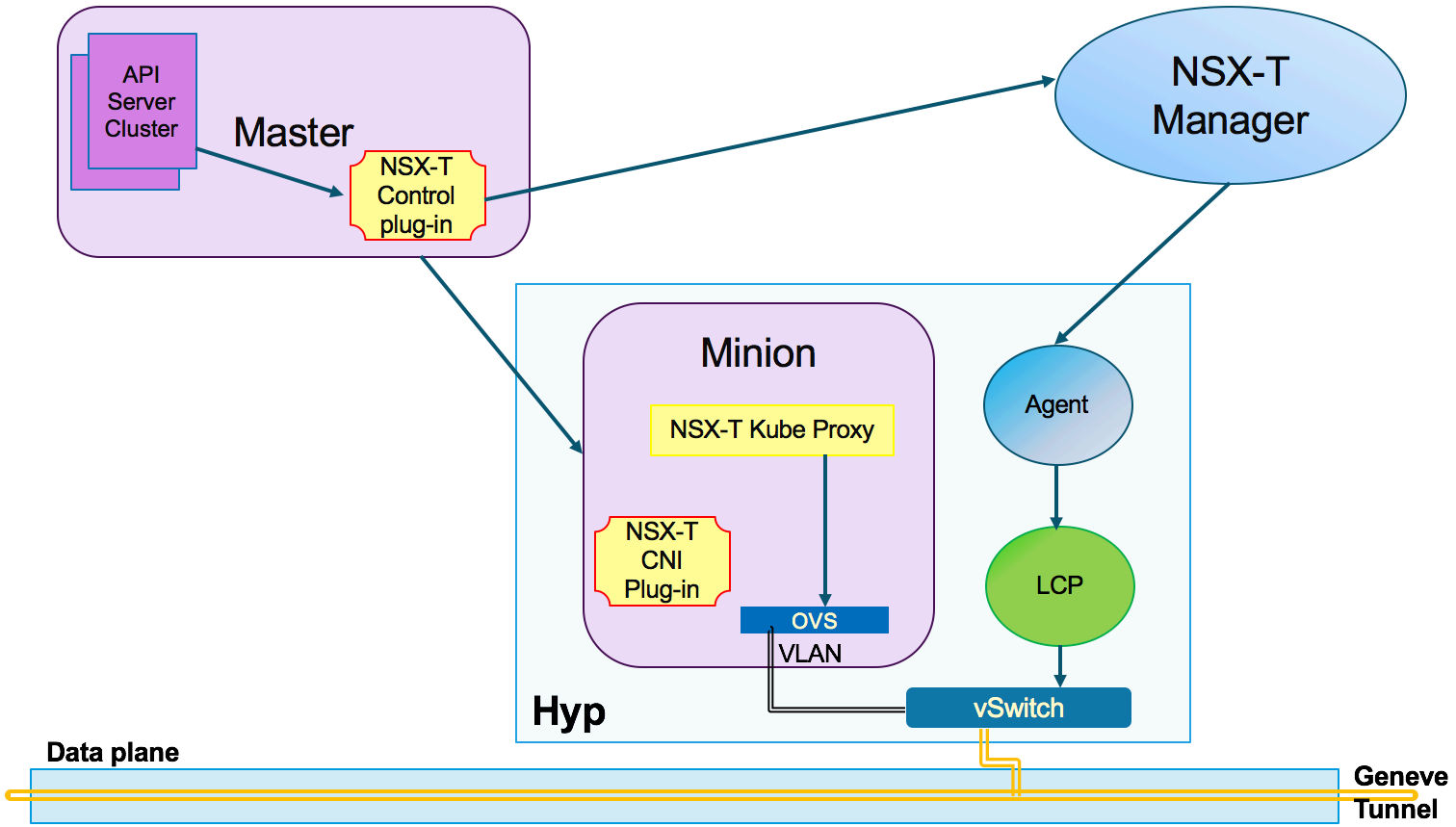
- 利用linux内核ip转发机制实现高效网络传输
- 基于三层路由，不依赖二层软件交换机等
- 通过BGP协议分发路由表



Flannel


CoreOS Machine
Pod
Web App Frontend1
cache1 container
app1 container
veth0
10.1.15.2/24
Pod
Web App Frontend2
cache2 container
app2 container
veth1
10.1.15.3/24
docker0
10.1.15.1/24
flannel0
10.1.15.0/16
flanneld
eth0
192.168.0.100
CoreOS Machine
Pod
Backend Service1
backend1 container
backup1 container
veth0
10.1.20.2/24
Pod
Backend Service2
backend2 container
backup2 container
veth1
10.1.20.3/24
docker0
10.1.20.1/24
flannel0
10.1.20.0/16
flanneld
eth0
192.168.0.200
packet
MAC
Outer IP
UDP
Inner IP
Payload
source: 192.168.0.100
dest: 192.168.0.200
source: 10.1.15.2
dest: 10.1.20.3

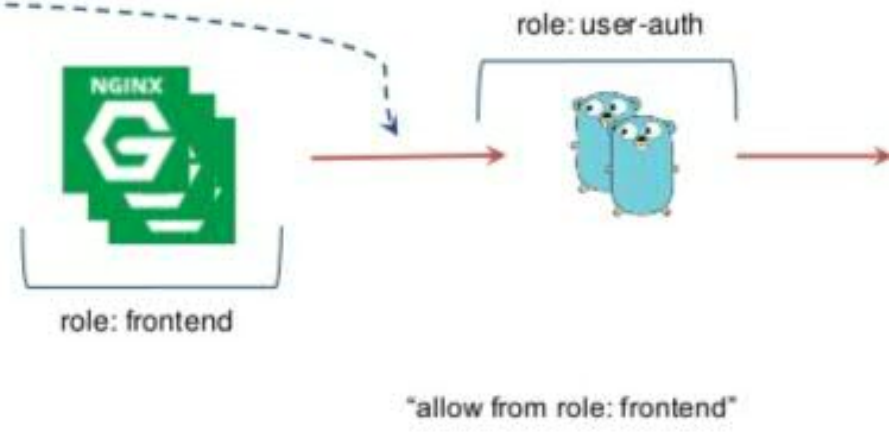
NSX-t


API
Server
Cluster
Master
NSX-T
Control
plug-in
NSX-T
Manager
Minion
NSX-T Kube Proxy
NSX-T
CNI
Plug-in
OVS
VLAN
Agent
LCP
vSwitch
Hyp
Data plane
Geneve
Tunnel

Network Policy

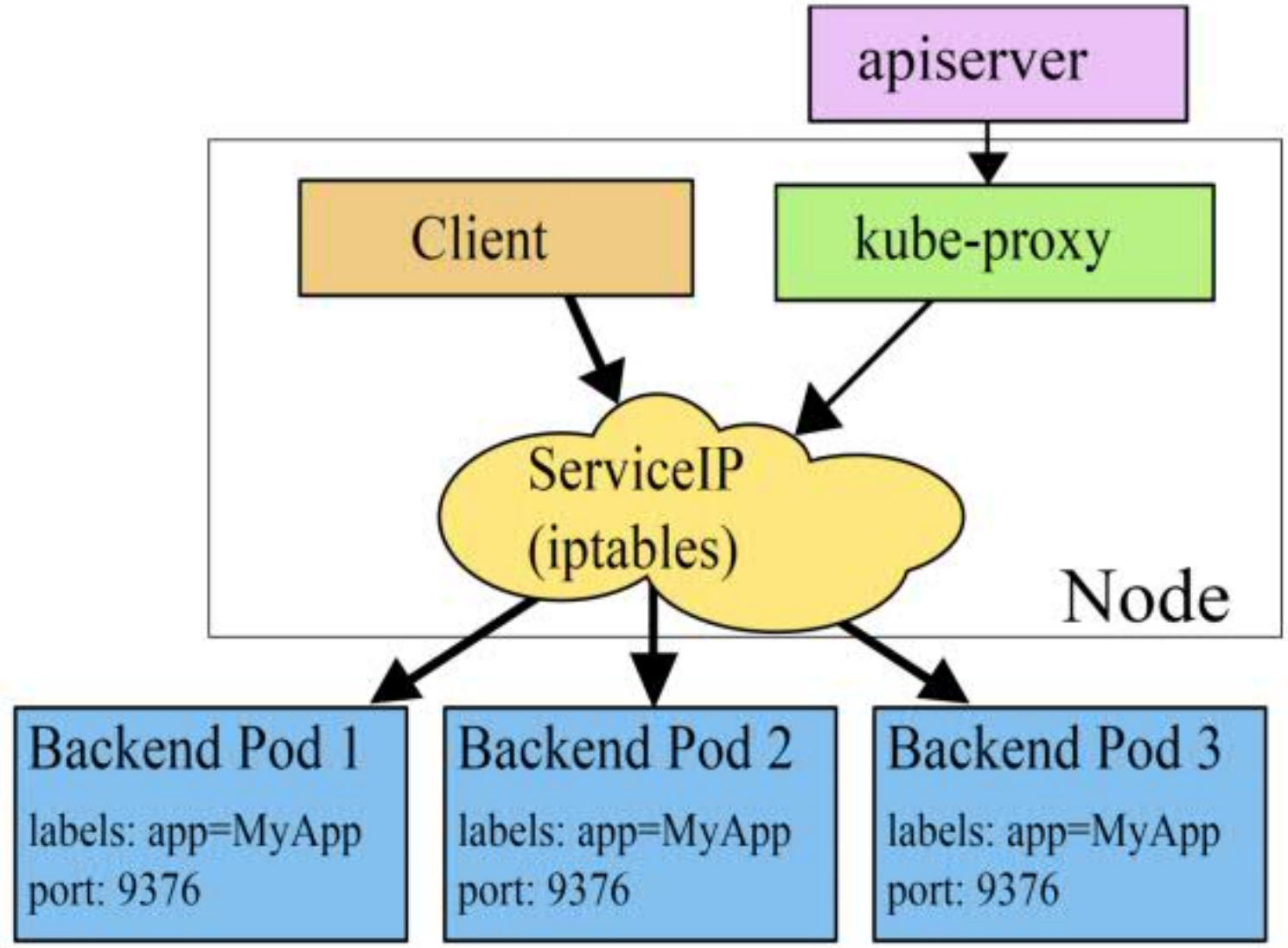
- 基于容器标签实现网络端口的访问控制

```
podSelector:
  role: user-auth
ingress:
  - from:
      - podSelector:
          role: frontend
    ports:
      - port: 8001
        protocol: tcp
```

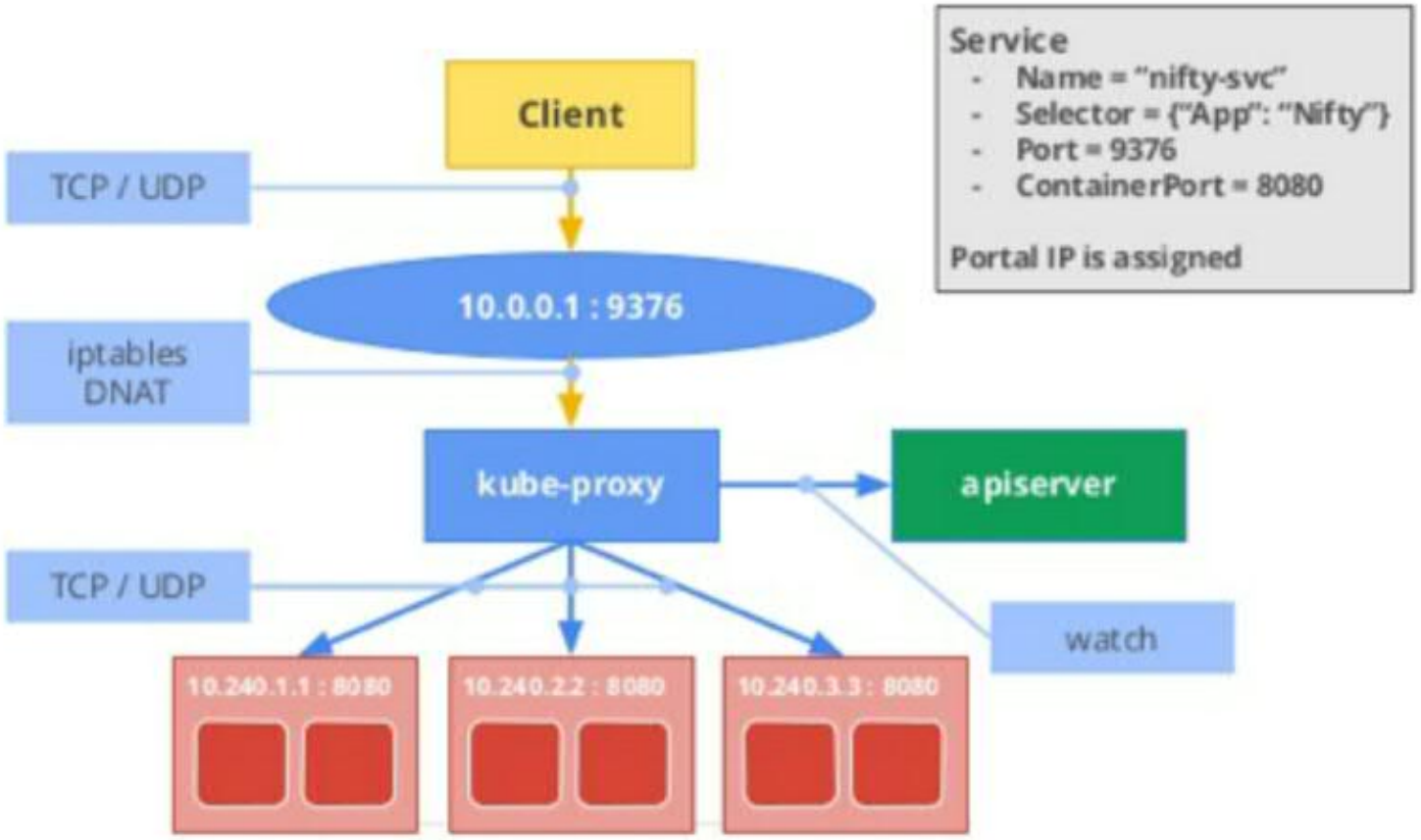


负载均衡

- 服务名称映射到虚拟IP（ServiceIP）并可通过DNS解析服务名
- Kube-proxy检测service的改变并更新Iptables，实现虚拟IP，端口到容器IP，端口的映射



负载均衡（iptables）



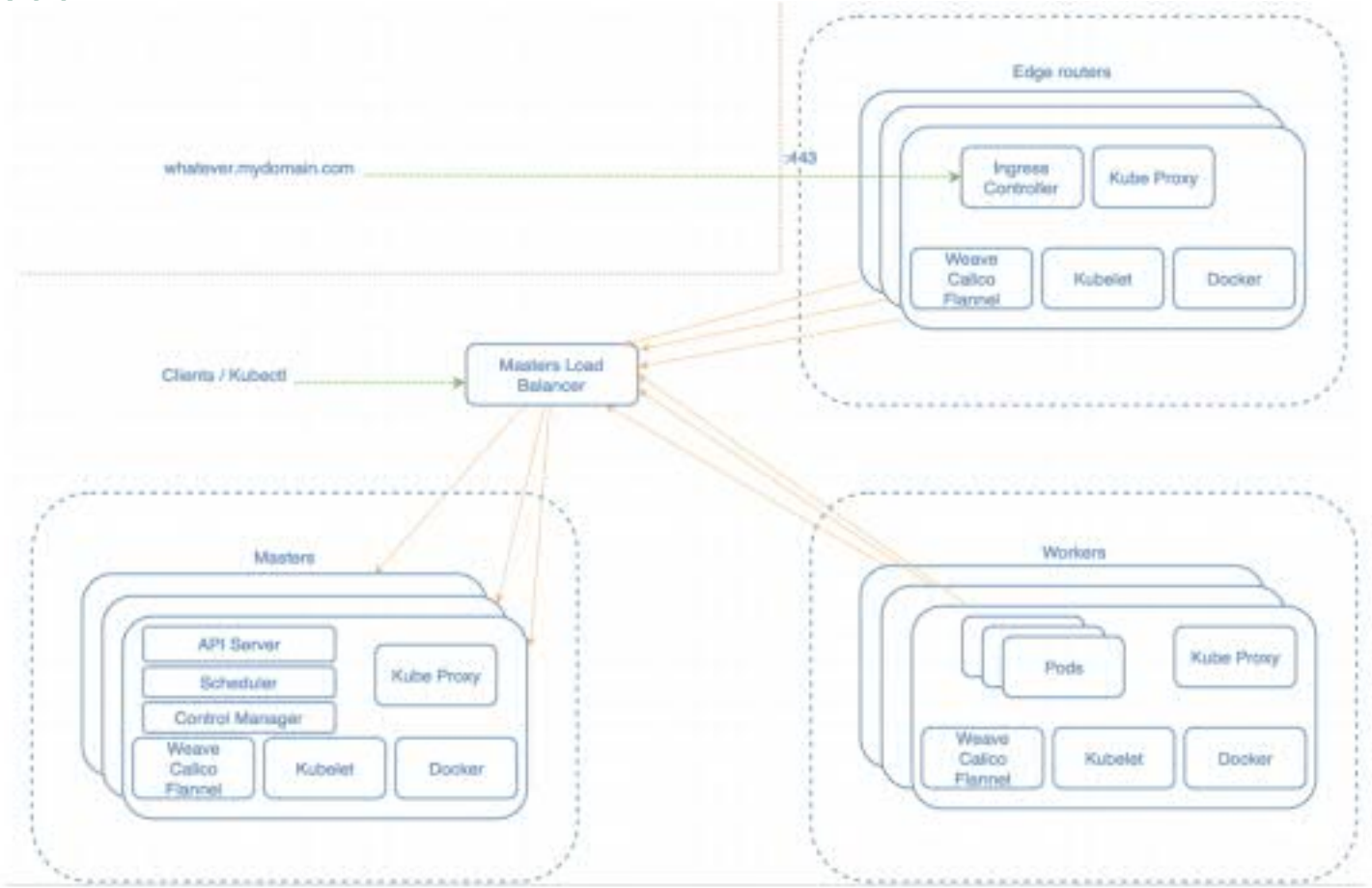
负载均衡（lvs）

- Kubernetes 1.8引入lvs作为实验性的kube-proxy backend
  - 解决iptables性能问题
  - 解决iptables无法保留源Ip地址头的问题
  - 可提供多种不同负载均衡算法，如最少连接和加权路由等

外部访问

- Kubernetes 提供三种外部访问方式:
  - NodePort: Kubernetes 将服务暴露在节点Ip地址的特定端口范围内(30000-32767)
  - Loadbalancer: Kubernetes 使用外部IaaS cloud provider 动态创建负载均衡将外部请求重定向到PODs
  - Ingress Controller : 使用Kubernetes ingress提供七层反向代理并支持tls等功能

Ingress


Edge routers
whatever.mydomain.com
:443
Ingress Controller
Kube Proxy
Weave Calico Flannel
Kubelet
Docker
Clients / Kubectl
Masters Load Balancer
Masters
API Server
Scheduler
Control Manager
Kube Proxy
Weave Calico Flannel
Kubelet
Docker
Workers
Pods
Kube Proxy
Weave Calico Flannel
Kubelet
Docker

Thanks!