阿里云 × CLOUD NATIVE COMPUTING FOUNDATION

云原生技术公开课

第 25 讲

Kubernetes网络模型进阶

叶磊 阿里巴巴高级技术专家

关注“阿里巴巴云原生”公众号
获取第一手技术资料

1

Kubernetes网络模型
来龙去脉

2

Pod究竟
如何上网？

3

Service究竟
怎么工作？

4

啥！负载均衡
还分内部外部？

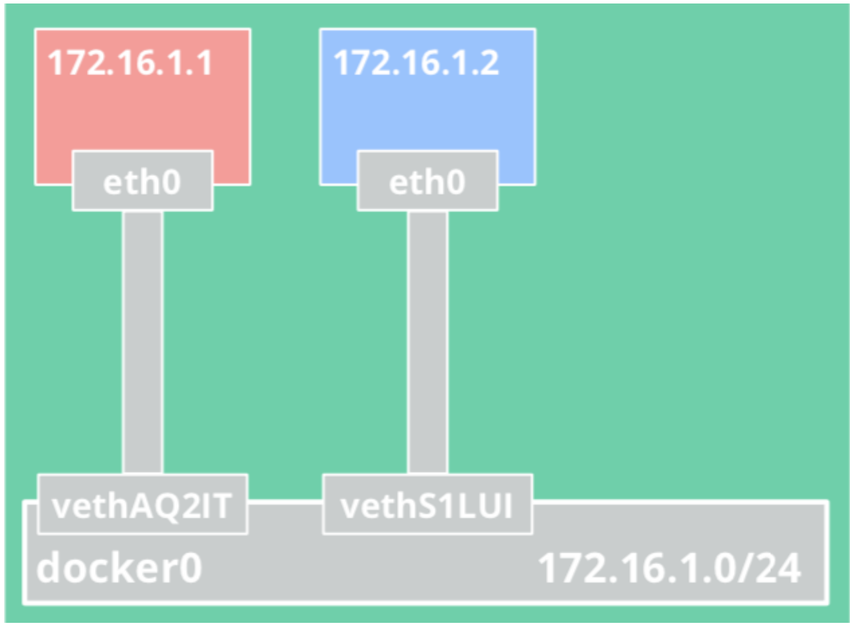
5

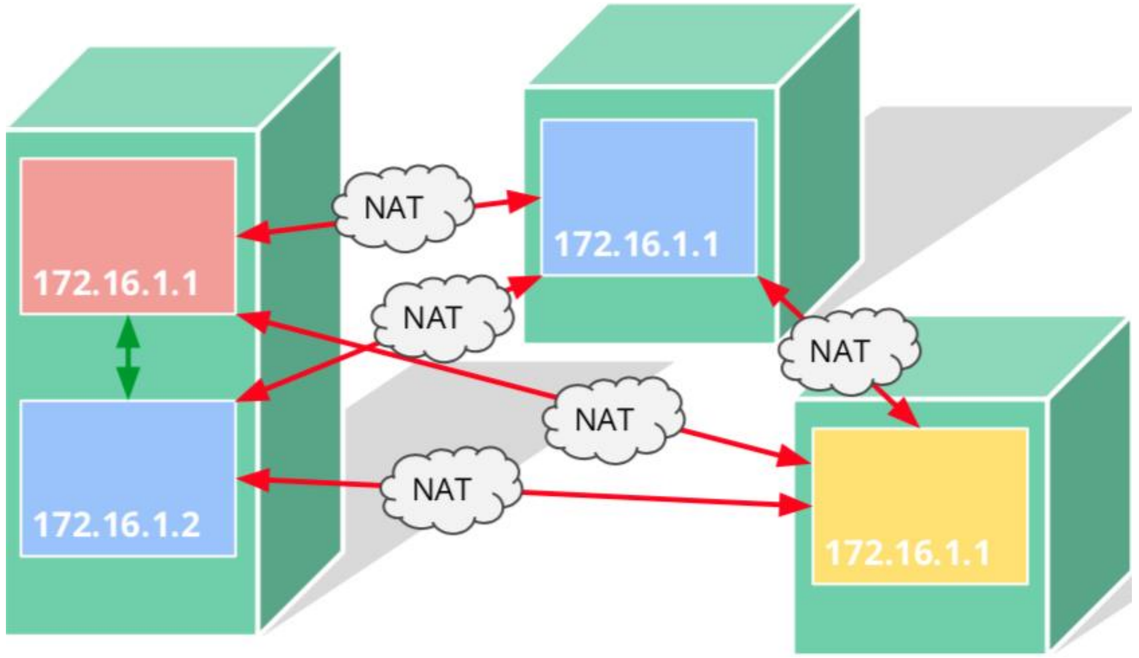
思考时间

前人挖坑： 早期Docker网络的由来及弊端

凡是用过Docker的，都见过Docker0 Bridge 和 172.XX：

- 最好的便利设计是与外部世界解耦，使用私网地址 + 内部 Bridge
- 出宿主机，采用SNAT借IP，进宿主机用DNAT借端口
- 问题就是一堆NAT包在跑，谁也不认识谁了



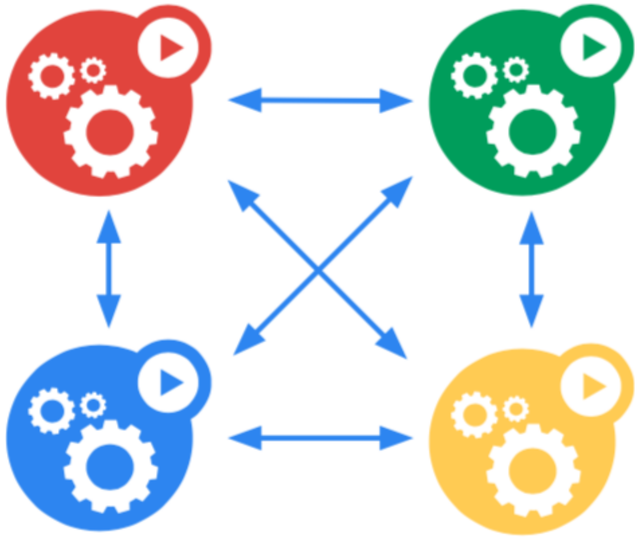


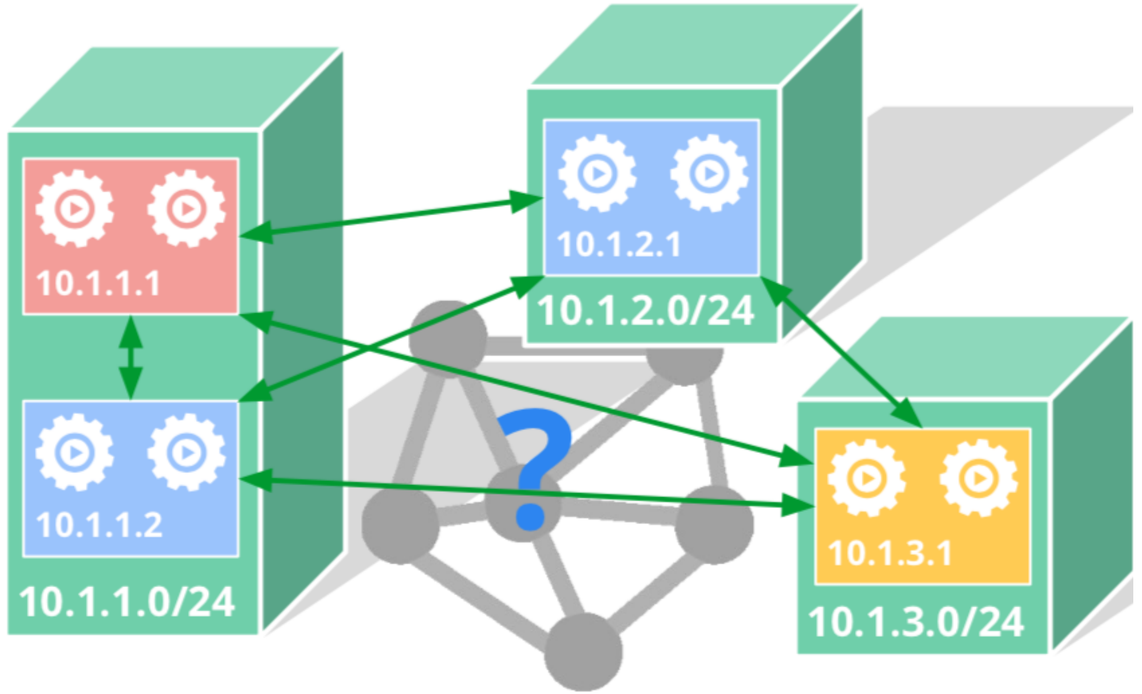
后人填坑： Kubernetes新人新气象

一句话，让一个功能聚集小团伙（Pod）正大光明的拥有自己的身份证 —— IP：

- Pod的IP是真身份证，通行全球就这一个号，拒绝任何变造（NAT）
- Pod内的容器共享这个身份证
- 实现手段不限，你能说通外部路由器帮你加条目？OK！

  你自己架了个Overlay网络穿越？ 也OK！





1 Kubernetes网络模型
来龙去脉

2 Pod究竟
如何上网？

3 Service究竟
怎么工作？

4 啥！负载均衡
还分内部外部？

5 思考时间

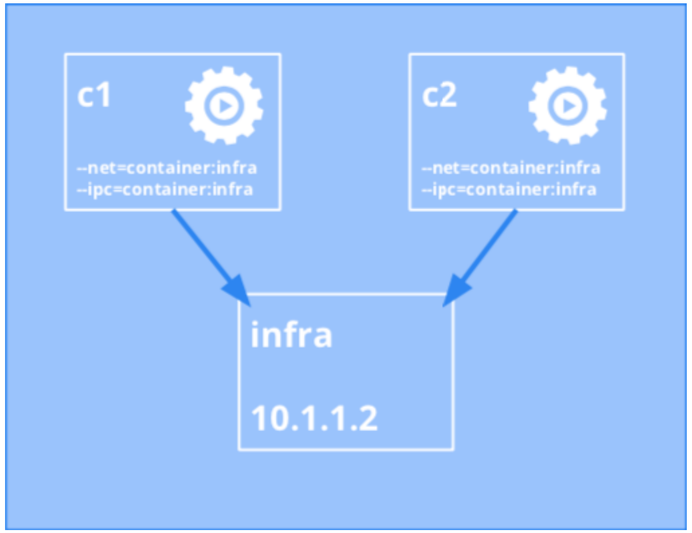
网络包不会自己飞：只能一步一步爬

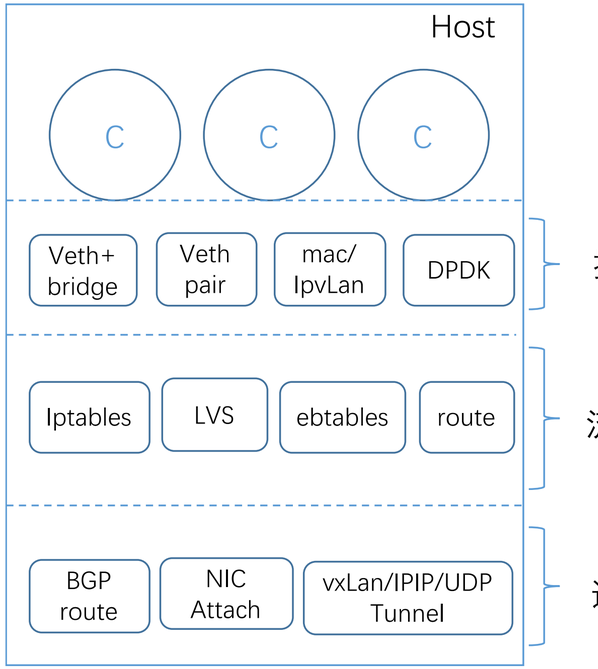
我们从两个维度来看：

1） 协议层次，需要从L2层（mac寻址） To L3层（IP寻址） To L4+（4层协议+端口）

2） 网络拓扑，需要从容器空间 To 宿主机空间 To 远端

Container Network Solution = 接入 + 流控 + 通道





接入:桥接、ipvlan派生等不同方式，负责容器数据包进出容器，
作为单独的CNI功能插件

流控：支持network policy功能落地，可采用Felix-Iptables、ebtables、
eBPF-XDP方式或Hook在数据路径上（Data Path），
可作为单独add-on或CNI插件方式部署

通道：有BGP路由（Felix BGP agent），直接挂接外网卡，
各种隧道，纯Gateway路由托方式，负责节点间通讯，
作为单独功能CNI插件

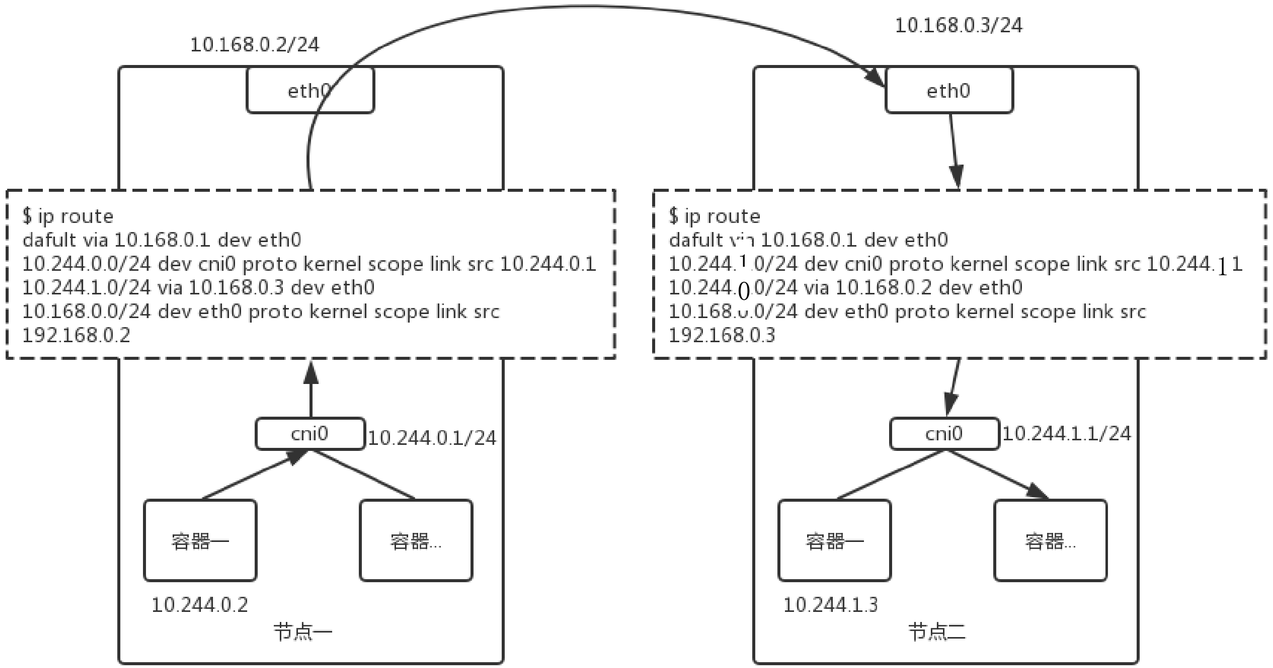
一个最简单的路由方案： Flannel-host-gw

IPAM 方案：

每个Node独占网段，Gateway放在本地，好处是管理简单，坏处是无法跨Node迁移Pod；

网络包勇往直前的一生：

1. **Pod-netns内出生：**容器应用产生高层数据 比如从左边10.244.0.2发送到10.244.1.3，根据路由决定目的Mac，如属于同一subnet内直接发给本机另一容器，如另一网段，则填写Gw-mac（cni0桥），通过pod-netns内 veth pair发送到 cni0桥；
2. **mac-桥转发：**Bridge的默认行为是按照mac转发数据，如目的mac为本桥上某port(另一容器)就直接转发；如目的为cni0 mac，则数据上送到Host 协议栈处理；
3. **ip-主机路由转发**：此时包剥离mac层，进入IP寻址层，如目的IP为本机Gw（cni0）则上送到本地用户进程，如目的为其他网段，则查询本地路由表（见图ip route 块），找到该网段对应的远端Gw-ip（其实是远端主机IP），通过neigh系统查出该远端Gw-ip的mac地址，作为数据的目的mac，通过主机eth0送出去；
4. **IP-远端路由转发**： 由于mac地址的指引，数据包准确到达远端节点，通过eth0进入主机协议栈，再查询一次路由表，正常的话该目的网段的对应设备为自己的cni0 IP，数据发给了cni0；
5. **mac-远端桥转发**：cni0作为host internal port on Bridge，收到数据包后，继续推送，IP层的处理末端填写目的ip（10.244.1.3）的mac，此时通过bridge fdb表查出mac（否则发arp广播），经过bridge mac转发，包终于到了目的容器netns。



1 Kubernetes网络模型来龙去脉

2 Pod究竟如何上网？

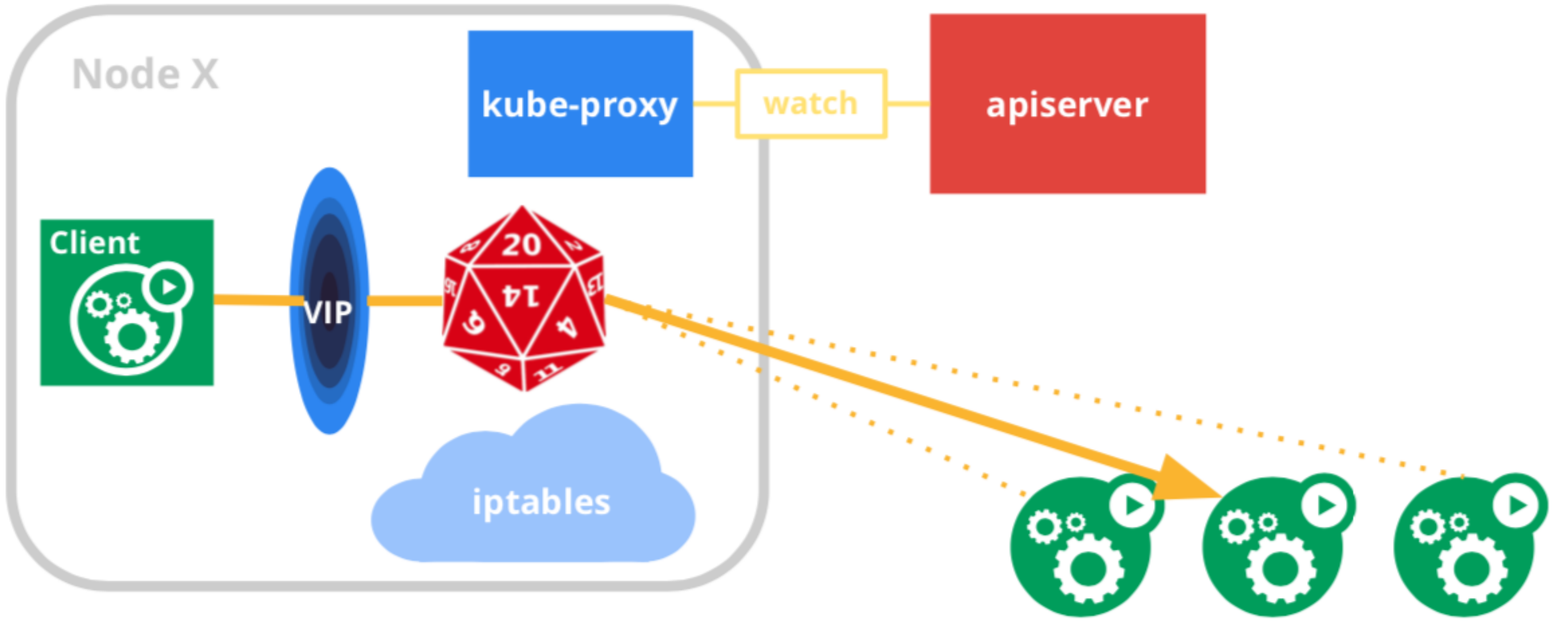
3 Service究竟怎么工作？

4 啥！负载均衡还分内部外部？

5 思考时间

Service = Internal Load Balance @Client Side

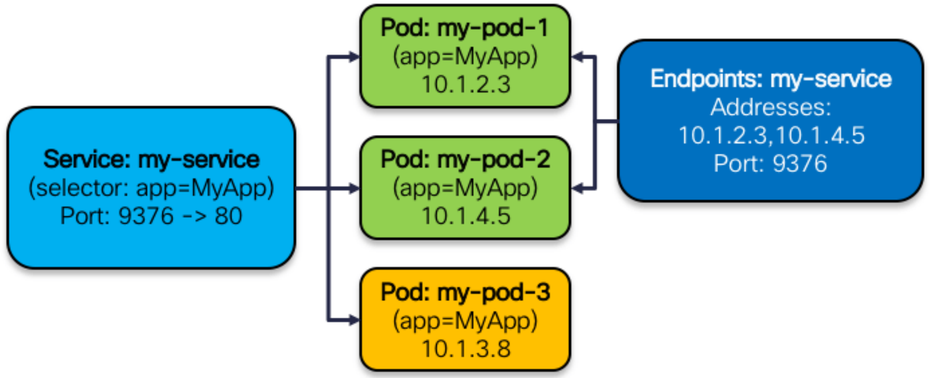
1) 一群Pod组成一组功能后端；
2) 定义一个稳定的虚IP作为访问前端，一般还附赠一个DNS域名，Client无需感知Pod的细节；
3) Kube-proxy是实现核心，隐藏了大量复杂性，通过apiserver监控Pod/Service的变化，反馈到LB配置中；
4) LB的实现机制与目标解耦，可以是个用户态进程，也可以是一堆精心设计的Rules（iptables/ipvs）；



三步！写一个高端大气的LVS版Service

背景知识

一定要让Kernel认为VIP是本地地址，这样4层的LVS才能开始干活！



第1步，绑定VIP到本地（欺骗内核）

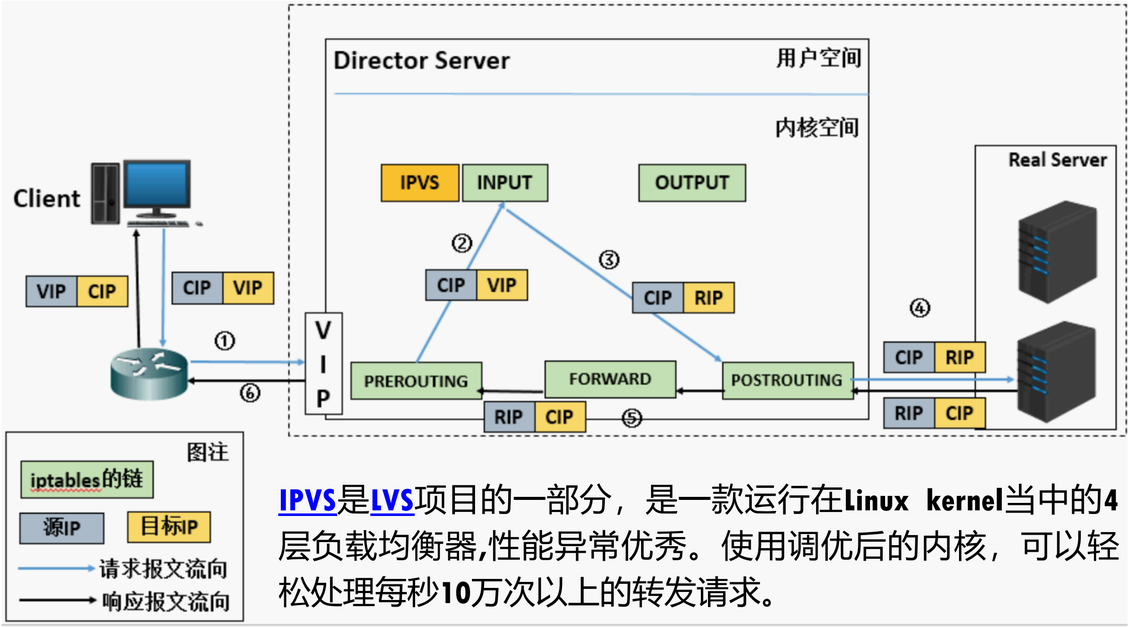
```
# ip route add to local 192.168.60.200/32 dev eth0proto kernel
```

第2步，为这个虚 IP 创建一个 IPVS 的 virtual server

```
# ipvsadm -A -t 192.168.60.200:9376 -s rr -p 600
```

第3步，为这个 IPVS service 创建相应的 real server

```
# ipvsadm -a -t 192.168.60.200:9376 -r 10.1.2.3:80 –m
# ipvsadm -a -t 192.168.60.200:9376 -r 10.1.4.5:80 –m
# ipvsadm -a -t 192.168.60.200:9376 -r 10.1.3.8:80 –m
```



IPVS是LVS项目的一部分，是一款运行在Linux kernel当中的4层负载均衡器,性能异常优秀。使用调优后的内核，可以轻松处理每秒10万次以上的转发请求。

1 Kubernetes网络模型来龙去脉

2 Pod究竟如何上网？

3 Service究竟怎么工作？

4 啥！负载均衡还分内部外部？

5 思考时间

Kubernetes的Service丰富多样

ClusterIP：Node内部使用，将Service承载在一个内部ClusterIP上，注意改服务只能保证集群内可以触达，这也是默认的服务类型。

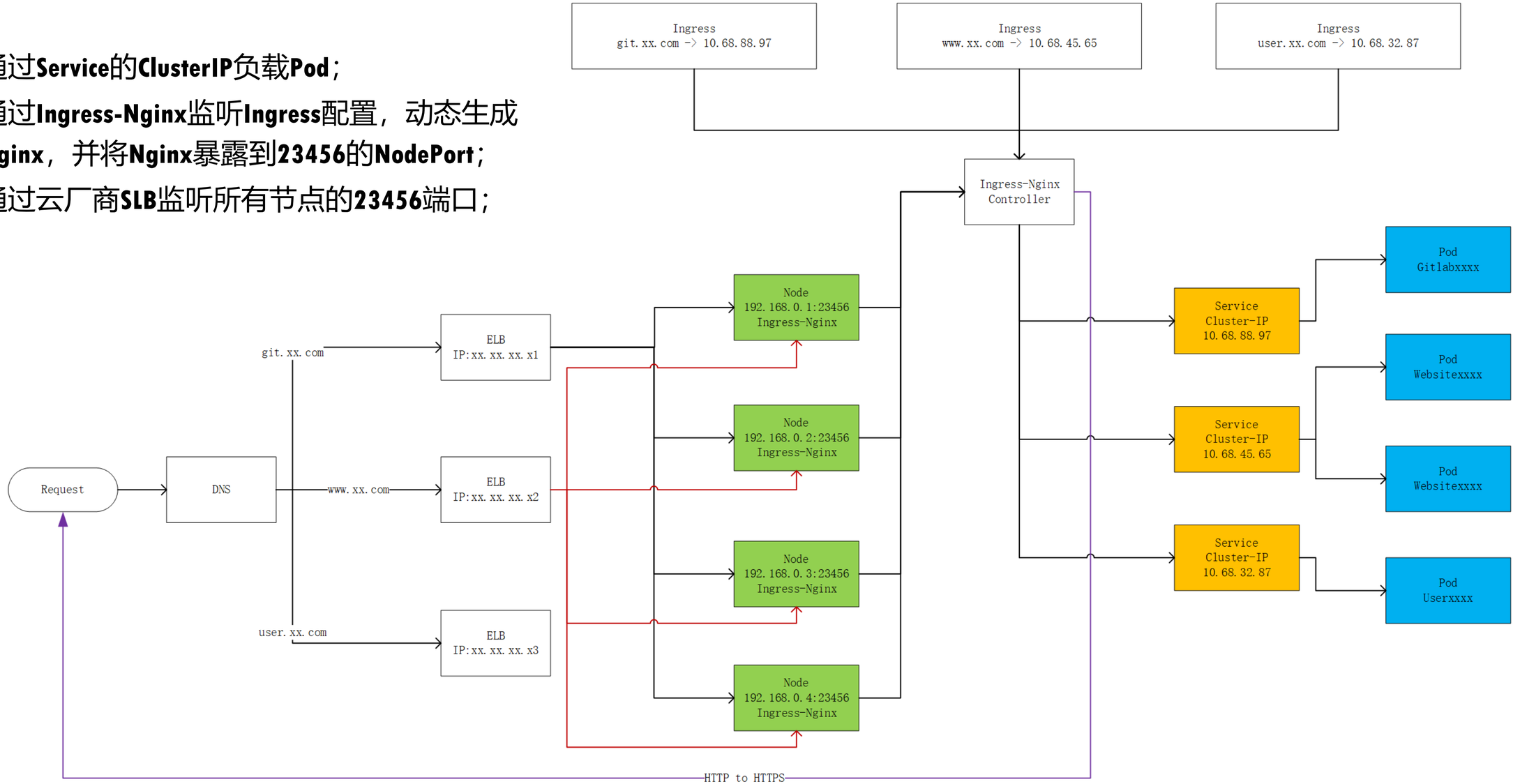
NodePort：供集群外部调用，将Service承载在Node的静态端口上，其实服务创建额时候也会自动创建一个ClusterIP，这样一来，服务就暴露在Node的端口上，集群外的用户可通过<NodeIP>:<NodePort>的形式调用到Service。

LoadBalancer：给云厂商留的扩展接口，将Service通过外部云厂商的负载均衡接口承载，其实为方便云厂商的插件编写，NodePort和ClusterIP两种机制也会自动创建，云厂商可已有选择将外部LB挂载到这两种机制上去。

ExternalName：去外面自由的飞翔，将Service的服务完全映射到外部的名称（如某域名），这种机制完全依赖外部实现，Service与CNAME挂钩，内部不会自动创建任何机制。

一个真正能工作云上的、从0搭建的负载均衡系统

- 通过Service的ClusterIP负载Pod；
- 通过Ingress-Nginx监听Ingress配置，动态生成Nginx，并将Nginx暴露到23456的NodePort；
- 通过云厂商SLB监听所有节点的23456端口；



1 Kubernetes网络模型来龙去脉

2 Pod究竟如何上网？

3 Service究竟怎么工作？

4 啥！负载均衡还分内部外部？

5 思考时间

思考一下

- 容器层的网络究竟如何与宿主机网络共存，合一？叠加？爱咋地咋地？
- **Service**还可以有怎样的实现？
- 为什么一个容器编排系统要大力搞服务发现和负载均衡？

谢谢观看

THANK YOU

关注“阿里巴巴云原生”公众号
获取第一手技术资料