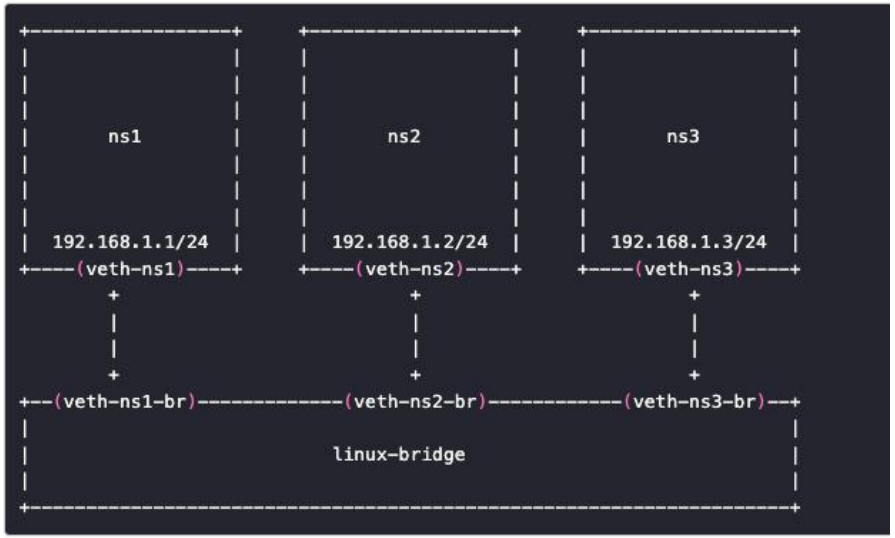
k8s 网络入门

xiangui.wang 202405

目录

CONTENTS

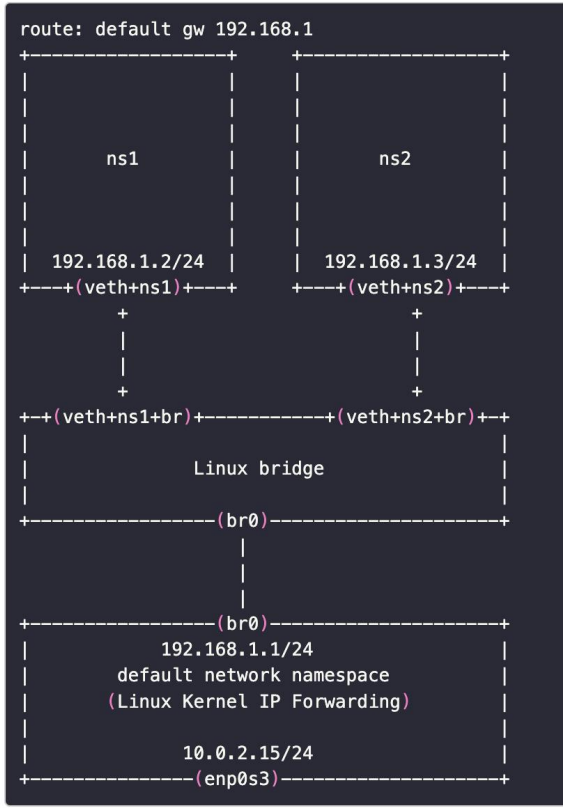
Linux 网络虚拟化基础 - veth pair和bridge



- Network namespace 实现网络隔离
- Veth pair提供了一种连接两个network namespace的方法
- Bridge 实现同一网络中多个namespace的连接

```
$ sudo ip netns exec ns1 ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2): 56 data bytes
64 bytes from 192.168.1.2: seq=0 ttl=64 time=0.068 ms
--- 192.168.1.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.060/0.064/0.068 ms
$ sudo ip netns exec ns1 ping 192.168.1.3
PING 192.168.1.3 (192.168.1.3): 56 data bytes
64 bytes from 192.168.1.3: seq=0 ttl=64 time=0.055 ms
--- 192.168.1.3 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.055/0.378/1.016 ms
```

Linux 网络虚拟化基础 - bridge Route 路由

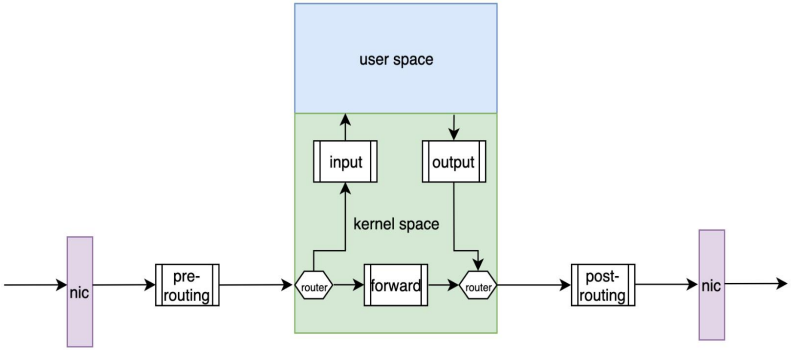


添加缺省网关

```
ip netns exec ns1 ip route add default via 192.168.1.1
ip netns exec ns2 ip route add default via 192.168.1.1
```

```
ip netns exec ns1 ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.052 ms
^C
--- 10.0.2.15 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.052/0.052/0.052/0.000 ms
ip netns exec ns2 ping 10.0.2.15
PING 10.0.2.15 (10.0.2.15) 56(84) bytes of data.
64 bytes from 10.0.2.15: icmp_seq=1 ttl=64 time=0.083 ms
64 bytes from 10.0.2.15: icmp_seq=2 ttl=64 time=0.115 ms
```

Linux 网络虚拟化基础 - iptabels 和NAT



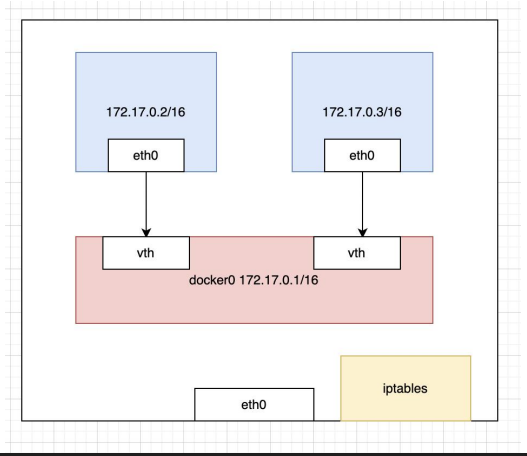
资料来源

```
$ sudo ip netns exec ns1 ping www.baidu.com
PING www.baidu.com (157.148.69.80): 56 data bytes
^C
--- www.baidu.com ping statistics ---
5 packets transmitted, 0 packets received, 100% packet loss
$
$
$ sudo iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -j MASQUERADE
$
$
$ sudo ip netns exec ns1 ping www.baidu.com
PING www.baidu.com (157.148.69.80): 56 data bytes
64 bytes from 157.148.69.80: seq=0 ttl=61 time=15.459 ms
64 bytes from 157.148.69.80: seq=1 ttl=61 time=12.488 ms
^C
--- www.baidu.com ping statistics ---
7 packets transmitted, 6 packets received, 14% packet loss
round-trip min/avg/max = 11.267/29.112/107.044 ms
```

```
3
4   $ sysctl net.ipv4.ip_forward
5   net.ipv4.ip_forward = 1
6
```

怎么实现外网访问虚拟网络ns1?

docker网络 - docker0/route/iptables



docker0网桥和缺省路由

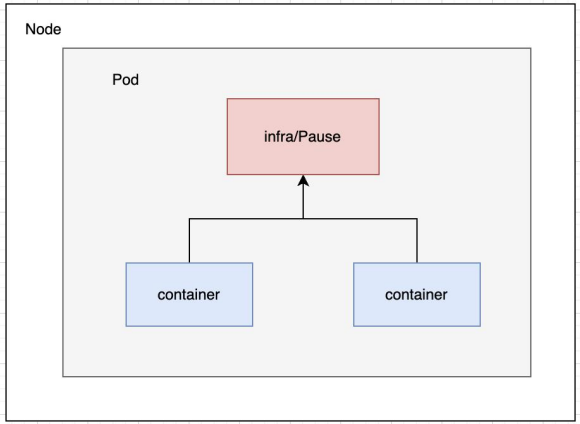
```
$ brctl show
bridge name bridge id       STP enabled interfaces
docker0     8000.024269d1a3af   no      veth7b96586
                                        vethab1075c

$  docker exec busybox1 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.17.0.1      0.0.0.0         UG    0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth0
```

```
$ docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id": "257b9bfbcc4e9299cc48d672bd120491811b02fe81fb16e3a9aedbcddf380c63",
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Containers": {
            "3fc0e929cdea019128738757e9a9e778ba0285f3a4c90b823e4b69663dc7bbe4": {
                "Name": "busybox1",
                "EndpointID": "4b8f18694479f30ae2cd40f720fe419478927c0700f2077518678240a2182f4e",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },
            "5ef1c4280127ddfa62d1e477a71ecebf368fd03ea0077fff25b7b45bc71a92ec": {
                "Name": "busybox2",
                "EndpointID": "f9b09e6593f37fa50762bb185f93cffb8d71406a9086bb8cc78f1e241bff5e3c",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            }
```

```
$ sudo iptables -t nat -S | grep docker
# （容器访问外部网络）所有出口不为 docker0 的流量，都做下 SNAT,把 src ip 换成出口接口的 ip 地址
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
-A DOCKER -i docker0 -j RETURN
```

Pod 网络 - infra/pause 容器

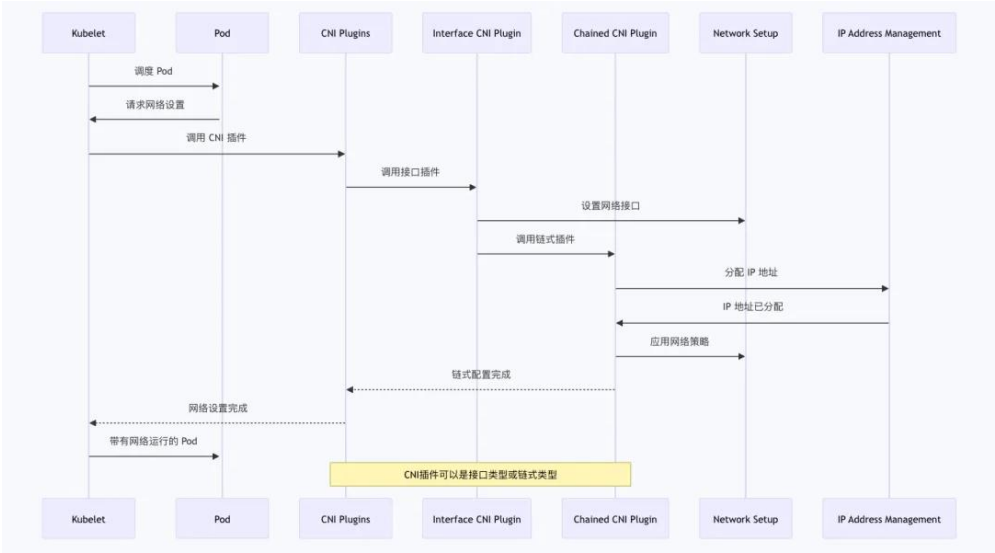


- 1个pod是1个network namespace，但是1个pod会包含多个container，**怎么实现多个container共享网络**？
- Pause 用于实现容器之间共享网络，如果其中部分容器挂掉，其余容器网路正常工作
- 启动pid命名空间，开启init进程，pod生命周期
- https://github.com/kubernetes/kubernetes/blob/master/build/pause/linux/pause.c

```
$ docker ps | grep etcd
dcdbcc2e55c2   73deb9a3f702              "etcd --advertise-cl…"   2 weeks ago   Up 2 weeks    k8s_etcd_e
33c06d925f2f   registry.k8s.io/pause:3.9  "/pause"                 2 weeks ago   Up 2 weeks    k8s_POD_et


$ docker inspect dcdbcc2e55c2 | grep -i networkMode
        "NetworkMode": "container:33c06d925f2f606587240ed8c0e502afcdcfa976cbd3e3e6d65539105cae05fc",
$
```

Pod 网络 - CNI 标准和插件

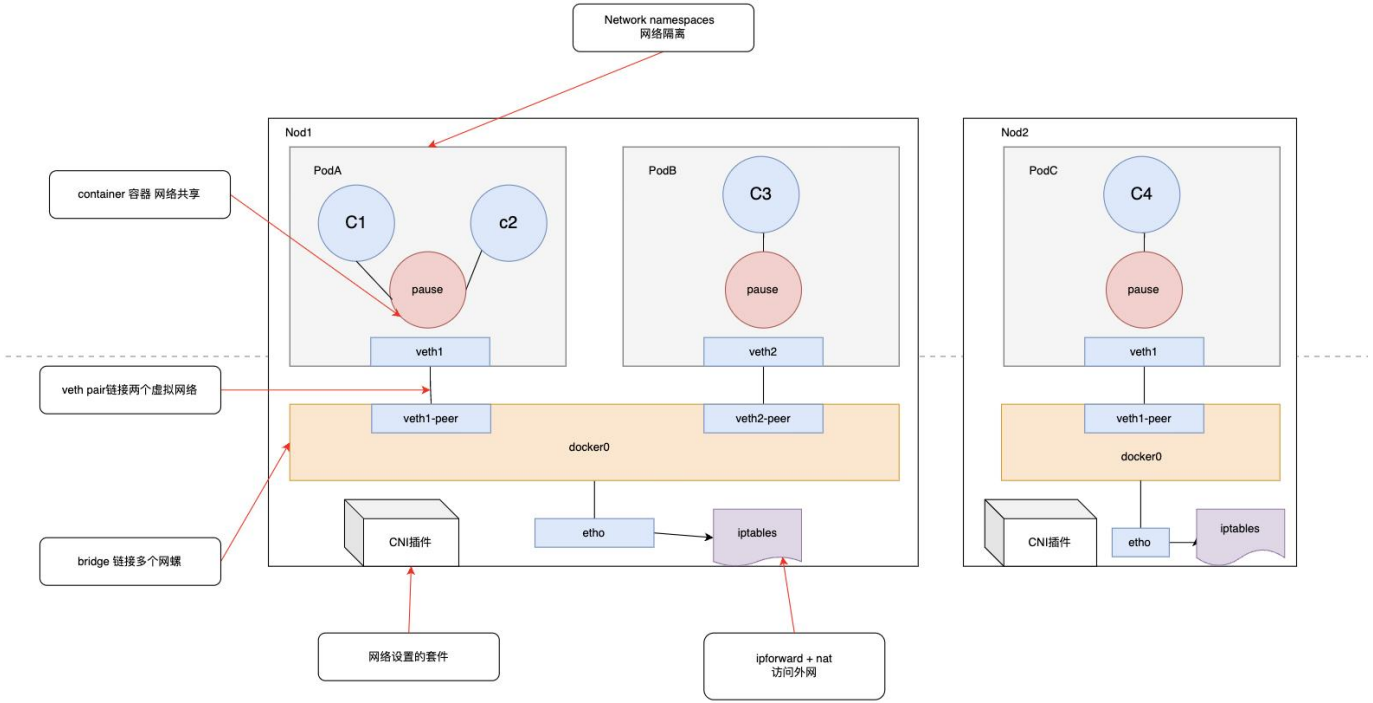



```
$ cat /etc/cni/net.d/1-k8s.conflist
{
    "cniVersion": "0.3.1",
    "name": "bridge",
    "plugins": [
        {
            "type": "bridge",
            "bridge": "bridge",
            "addIf": "true",
            "isDefaultGateway": true,
            "forceAddress": false,
            "ipMasq": true,
            "hairpinMode": true,
            "ipam": {
                "type": "host-local",
                "subnet": "10.244.0.0/16"
            }
        },
        {
            "type": "portmap",
            "capabilities": {
                "portMappings": true
            }
        }
    ]
}
```

```
$ ls /opt/cni/bin/
bandwidth  bridge  cnitool  dhcp  firewall  host-local  ipvlan  loopback  macvlan  portmap  ptp  tuning  vlan
$
```

- CNI标准: https://github.com/containernetworking/cni
- CNI 插件:https://github.com/containernetworking/plugins

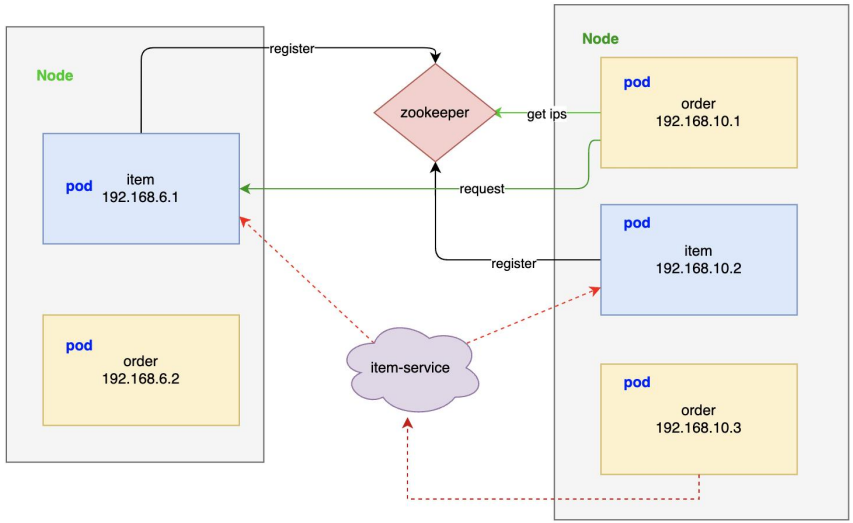
Pod 网络 - 小结


Network namespaces
网络隔离
Nod1
PodA
C1
c2
pause
veth1
PodB
C3
pause
veth2
Nod2
PodC
C4
pause
veth1
container 容器 网络共享
veth pair链接两个虚拟网络
bridge 链接多个网螺
veth1-peer
veth2-peer
docker0
CNI插件
etho
iptables
网络设置的套件
ipforward + nat
访问外网

目录

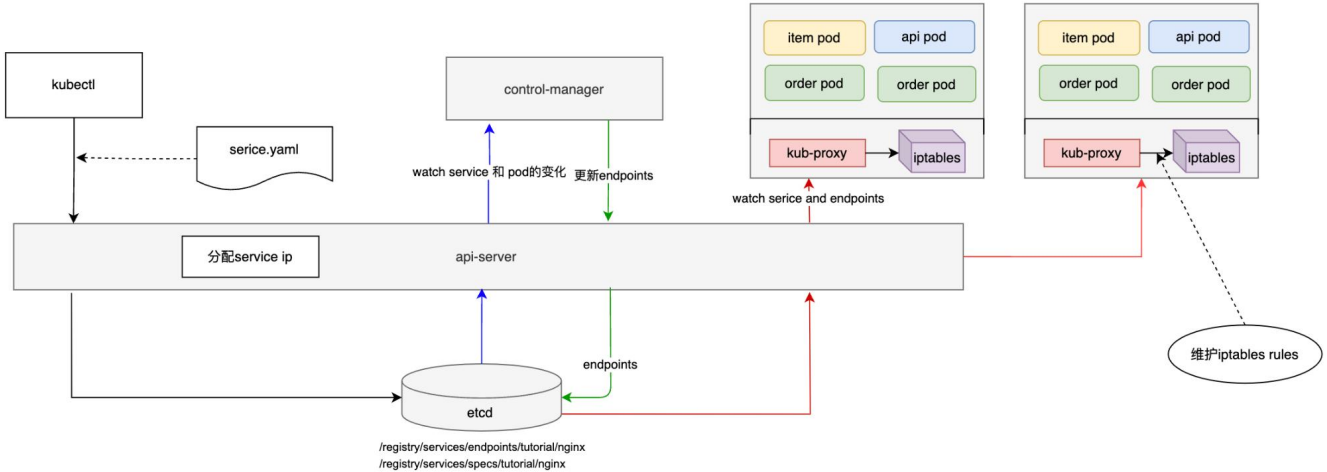
CONTENTS

Service 网络 - 背景和用途,与Pod的练习



- Zookeeper提供名字服务，pod自身实现负载均衡，RPC框架实现负载均衡
- Service 为 Pods 提供的固定 IP，其他服务可以通过 Service IP 找到提供服务的 Endpoints。
- Service提供负载均衡。Service 由多个 Endpoints 组成，kubernetes 对组成 Service 的 Pods 提供的负载均衡方案，例如随机访问、robin 轮询等。
- 暂时将Pod等同于Endpoint

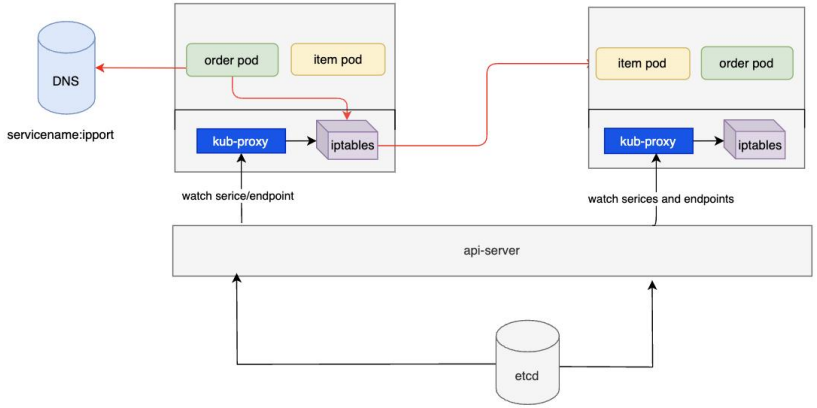
Service 网络 - 整体流程原理



- Service IP IP 由API server分配，写入etcd
- Etcd 中存储service和endpoints
- Controllermanager watch etcd的变换生成endpoints

```
C02FV5XLMD6M:.minikube xianguiwang$ kubectl describe svc nginx -n tutorial
Name:              nginx
Namespace:         tutorial
Labels:            app=nginx
Annotations:       <none>
Selector:          app=nginx
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.107.35.208
IPs:               10.107.35.208
Port:              <unset>  8080/TCP
TargetPort:        80/TCP
Endpoints:         10.244.0.6:80,10.244.0.7:80
Session Affinity:  None
Events:            <none>
```

Service网络 - kube-proxy 服务发现和负载均衡



- Order -> item 的流程
- 服务发现：环境变量和DNS
- servicename.namespace.svc.cluster.local
- kub-proxy 通过watch etcd中service和endpoint的变更，维护本地的iptables/ipvs
- kub-proxy 通过转发规则实现service ip 到pod ip的转发，通过规则实现负载均衡

```
-A KUBE-SEP-BMYZ7L5WPNEKEEIX -s 10.244.0.7/32 -m comment --comment "tutorial/nginx" -j KUBE-MARK-MASQ
-A KUBE-SEP-BMYZ7L5WPNEKEEIX -p tcp -m comment --comment "tutorial/nginx" -m tcp -j DNAT --to-destination 10.244.0.7:80
-A KUBE-SEP-YDVU3MZJROB7LW5N -s 10.244.0.6/32 -m comment --comment "tutorial/nginx" -j KUBE-MARK-MASQ
-A KUBE-SEP-YDVU3MZJROB7LW5N -p tcp -m comment --comment "tutorial/nginx" -m tcp -j DNAT --to-destination 10.244.0.6:80
-A KUBE-SERVICES -d 10.107.35.208/32 -p tcp -m comment --comment "tutorial/nginx cluster IP" -m tcp --dport 8080 -j KUBE-SVC-LDCPNUJYSP4HZM76
-A KUBE-SVC-LDCPNUJYSP4HZM76 ! -s 10.244.0.0/16 -d 10.107.35.208/32 -p tcp -m comment --comment "tutorial/nginx cluster IP" -m tcp --dport 8080 -j KUBE-MARK-MASQ
-A KUBE-SVC-LDCPNUJYSP4HZM76 -m comment --comment "tutorial/nginx -> 10.244.0.6:80" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-YDVU3MZJROB7LW5N
-A KUBE-SVC-LDCPNUJYSP4HZM76 -m comment --comment "tutorial/nginx -> 10.244.0.7:80" -j KUBE-SEP-BMYZ7L5WPNEKEEIX
```

Service网络 - 网络类型

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
  name: nginx
  namespace: tutorial
spec:
  clusterIP: 10.97.137.138
  clusterIPs:
  - 10.97.137.138
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

k8s集群内部访问

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
  name: nginx
  namespace: tutorial
spec:
  clusterIP: 10.109.13.110
  clusterIPs:
  - 10.109.13.110
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - nodePort: 32322
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

支持外部节点访问

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app.kubernetes.io/name: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  clusterIP: 10.0.171.239
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.0.2.127
```

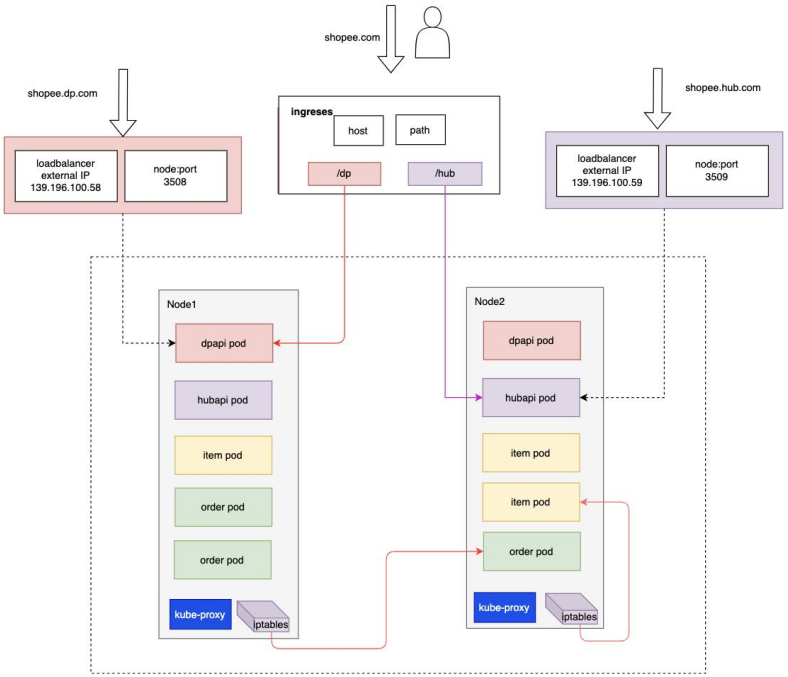
支持外部节点访问

官方资料来源

目录

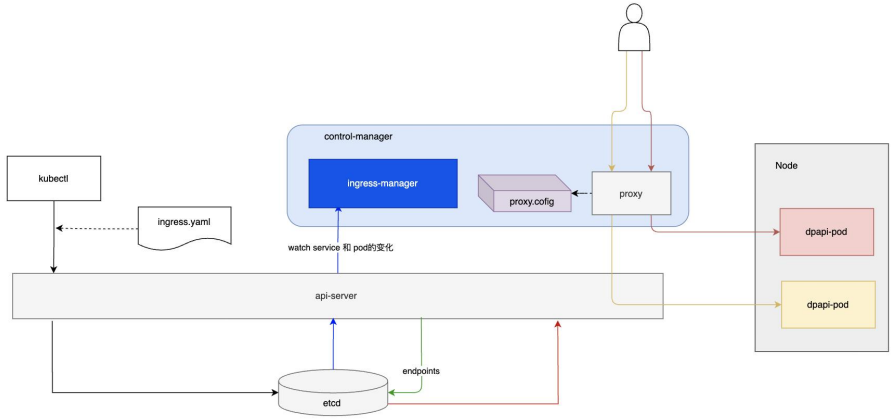
CONTENTS

Ingress网络 - 背景和用途



- 集群外部访问集群内部资源？nodeport,loadbalancer。一个服务一个port或者一个外网IP，一个域名
- Ingress 是 Kubernetes 中的一种 API 对象，用于管理入站网络流量，基于域名和URL路径把用户的请求转发到对应的service
- ingress相当于七层负载均衡器，是k8s对反向代理的抽象
- ingress负载均衡，将请求自动负载到后端的pod

Ingress 网络 - 整体流程(controller+proxy)



- ingress 资源对象用于编写资源配置规则
- Ingress-controller 监听apiserver感知集群中service和pod的变化动态更新配置规则，并重载proxy反向代理的配置
- proxy反向代理负载均衡器，例如ngnix，接收并按照ingress定义的规则进行转发，常用的是ingress-nginx等，直接转发到pod中

```
PID   USER     TIME  COMMAND
    7 www-data  0:15 /nginx-ingress-controller --election-id=ingress-nginx-leader --controller-class=k8s.io/ingress-nginx
    --watch-ingress-without-class=true --co
   20 www-data  0:00 nginx: master process /usr/bin/nginx -c /etc/nginx/nginx.conf
  237 www-data  0:03 nginx: worker process
  238 www-data  0:03 nginx: worker process
```

Ingress网络 - Ingress 规则和路由、示例

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - host: hello-world.info
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: web
                port:
                  number: 8080
          - path: /v2
            pathType: Prefix
            backend:
              service:
                name: web2
                port:
                  number: 8080
```

- 通过使用路径规则。例如： /app1 路径映射到一个服务，将 /app2 路径映射到另一个服务。路径匹配支持精确匹配和前缀匹配两种方式。
- 基于主机的路由匹配。例如，可以将 app1.example.com 主机名映射到一个服务，将 app2.example.com 主机名映射到另一个服务。主机匹配也可以与路径匹配结合使用，实现更细粒度的路由控制。
- 其他条件的路由匹配：：请求方法（如 GET、POST）、请求头（如 Content-Type）、查询参数等。

```
C02FV5XLMD6M:lib xianguiwang$ kubectl get ingress
NAME              CLASS   HOSTS              ADDRESS          PORTS   AGE
example-ingress   nginx   hello-world.info   192.168.59.101   80      5d21h
C02FV5XLMD6M:lib xianguiwang$ curl --resolve "hello-world.info:80:$( minikube ip )" http://hello-world.info
Hello, world!
Version: 1.0.0
Hostname: web-57f46db77f-pc272
C02FV5XLMD6M:lib xianguiwang$ curl --resolve "hello-world.info:80:$( minikube ip )" http://hello-world.info/v2
Hello, world!
Version: 2.0.0
Hostname: web2-866dc4bcc8-cgbgq
```

学习资料

- Minikube 环境安装
- Kubectl 命令和集群体验
- Linux 虚拟网络
  - Linux network namespace,veth,bridge 和 路由
  - 从0到1搭建linux虚拟网络
- Docker 网络：模拟docker网络
- Docker 网络：从docker0开始
- Pod网络和pause容器
- 认识CNI插件
- 深度解读CNI：容器网络接口
- 官方文档：服务service
- 创建service之后，k8s会发生什么
- 探究k8s service iptables 路由规则
- 官方文档：在minikube中使用nginx ingress 控制配置ingress
- 官方文档：ingress

Thank You.