

[简书](#)[发现](#)[关注](#)[消息](#)[搜索](#)

Aa

beta



k8s网络原理-ipvs



左小星

关注

0.155

2020.07.05 10:02:03 字数 3,944 阅读 1,331



一、背景知识

本文主要介绍k8s网络中service的两种模式(clusterip、nodeport)，数据是如何通过ipvs&iptables流转的。在学习上述知识的同时，还需要了解一下ipset、conntrack的相关知识。

往期回顾文章

- [iptables还能这么玩，厉害了](#)
- [一文理解docker单机网络](#)
- [docker集群网络玩法](#)

1.1、ipset

ipset是什么？ipset其实是iptables的扩展，可以定义一些列地址的集合。拿黑名单来举例，我想让黑名单里面的ip拒绝访问网站(黑名单有很多个)，按照传统iptables做法，需要在filter表添加很多规则匹配时一条一条匹配效率很低(严重影响性能)，而有了ipset，则只用添加一条规则即可，使用hash结构效率很高。

```
1 // 传统iptables方式不让这两个ip访问特定端口
2 iptables -I INPUT -p tcp -s 192.168.113.101 --dport 8410 -j DROP
3 iptables -I INPUT -p tcp -s 192.168.113.99 --dport 8410 -j DROP
```

而使用ipset命令如下

```
1 // 创建一个ipset规则，名字叫black，以hash类型存储，ip + port作为hash的key
2 ipset create black hash:ip,port
3 // 向black中添加具体 ip + port
4 ipset add black 192.168.113.100,8410
5 // 删除black中的某条规则
6 ipset del black 192.168.113.100,8410
7 // 添加到iptables规则上
8 iptables -t filter -I INPUT -m set --match-set black src,dst -j DROP
```

当然，ipset还支持 hash:ip, hash:ip,port,ip等多种hash key的组成，具体可以通过 ipset -h 查看。接下来说明一下 -m set 后面 src 和 dst 两个的含义。src 指来源，dst 指目标，此规则的意思是来自192.178.113.100 ip 访问本机8410端口的流量给DROP掉。

ipset使用hash结构，比iptables的链表遍历效率要高很多。ipset还有很多更加高级的玩法，本文就不在阐述了。

1.2、ipvs

lvs是什么？全称是Linux Virtual Server，是由章文嵩博士主导的开源负载均衡项目，目前已经集成到linux内核中。lvs提供了丰富的负载均衡能力，接收到用户请求后根据具体的负载均衡算法在内核态把请求转发到后端的某个server上，也就是说lvs不需要监听具体的端口。接下来我们看一下lvs的一些基本概念。

- DS : director server 指负载均衡节点。

写下你的评论...

评论0

赞1

...

[简书](#)[发现](#)[关注](#)[消息](#)[搜索](#)[beta](#)

- DIP : director ip 指负载均衡节点ip。
- RIP : real server ip 指后端节点ip。
- CIP : client ip 指client端ip。

ipvs的原理如下。ipvs工作在iptables 的 input链上，VIP一般定义在DS节点上的一个虚拟ip，拿nat模式举例如下。

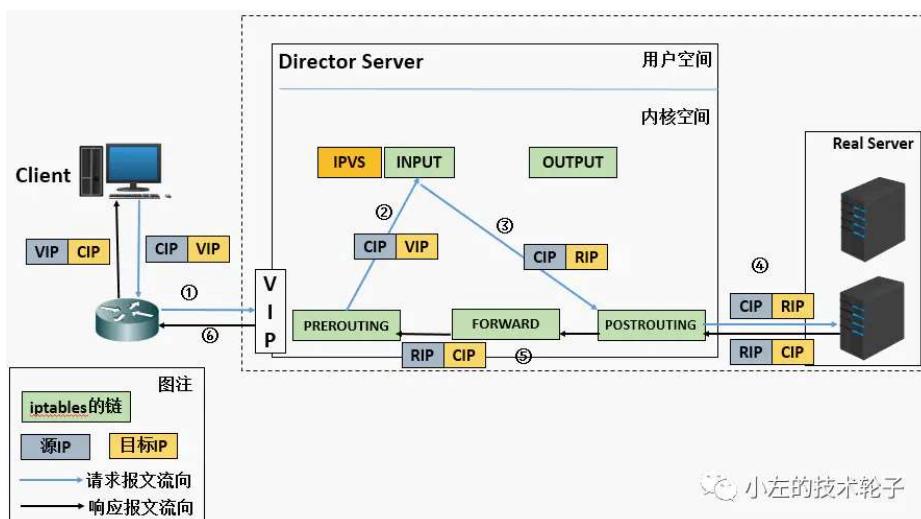


- ①: 当请求数据包到DS上最先经过iptables 的PREROUTING链，判断目标ip (VIP) 是本机的ip，于是把请求转发到INPUT链上。
- ②: 因为lvs工作在INPUT链上，数据到达INPUT链上后lvs会将用户请求和定义的后端服务做对比，如果是请求的后端服务，则使用某种负载均衡算法找到一个后端RIP，修改数据包的目的ip和端口为某个RIP的(DNAT转换)。
- ③: 此时数据到达POSTROUTING链(不会做SNAT)，数据包的源ip 为CIP，目的ip为RIP，数据包发往RIP上。

lvs提供了三种包转发模式，如下所示

- NAT 模式，做DNAT 转换修改目的ip地址，需要DIP和RIP在一个网段内&RIP所在节点的网关指向DIP。(做了NAT转换，考验DS节点的性能)
- DR 模式，DS修改数据包的mac地址，将源mac 地址修改为DIP的mac地址，目的mac地址为RIP的mac地址。同时RS的lo接口ip修改为VIP。
- TUN模式，在原有的ip数据包上在封装一层ip包，最外层ip包 (源ip 为 DIP，目的ip为RIP)，最内层ip包(源ip CIP，目的ip为VIP)，同时RS上的lo接口的ip设置为VIP。

由于k8s使用的是NAT模式，接下来看下NAT模式下的数据包流向。如下图所示



image

- ①: 请求数据包到达DS，数据包经过PREROUTING链，此时ip 包 src ip为CIP, dst ip 为VIP
- ②: 由于请求的VIP是DS上的虚拟ip，数据包发往INPUT链。
- ③: 数据包到INPUT链上后，ipvs发现数据包请求是定义的集群服务，于是使用定义好的负载均衡算法找到一个具体的RS节点，做DNAT，修改数据包dst ip为RIP，数据包到达POSTROUTING链，发送给RS。
- ④: RS收到数据包后对比dst ip 发现是自己，接收数据包做处理，处理完成后ip 数据包 src ip 为DIP, dst ip 为CIP, 把数据包发回给DS。

写下你的评论...

评论0

赞1

[简书](#)[发现](#)[关注](#)[消息](#)[搜索](#)[beta](#)

- NAT模式下，进出流量都需要经过director server，并且需要做NAT转换，在高流量时对director server的性能有一定影响。并且RS的网关要指向DIP。
- DR模式下，只有入口流量要经过director server，由于只是修改了mac地址这就要求RS和DS在一个二层网络内，效率会非常高。RS上的lo接口要配置vip的ip地址，并且RS的网关不能指向DIP。
- TUN模式下，也是只有入口流量要经过director server，此种模式建立了ip隧道，RS的lo接口ip地址要设置为vip的ip地址，并且RS的网关不能设置为DIP。



接下来在简单聊一下ipvs的负载均衡策略，简单介绍下面四种。

- rr 最简单的一种调度算法，轮训的把请求转发到后端server上。
- wrr 加权轮训，可以为RS设置相应的权重，权重越高表示该RS的性能越高，接收到的请求也就越多。
- lc 最少连接，根据RS请求的连接数，比如RS1的连接数比RS2的少，则把请求发给RS2
- wlc 在最少连接的基础上添加了权重的概念。

上面介绍完了ipvs内核态的基本原理，接下来介绍一下如何使用 ipvsadm 用户态命令来操作ipvs。说明：此次试验是在四个虚拟机上，ipvs的模式使用的nat模式，RS的网关没有指向DS的ip(没办法做到)在DS节点上手动创建SNAT命令，下文有详细介绍。创建一个vip，在ip为192.168.113.101上

```

1 // -A 表示添加一个VIP, -t 表示tcp 协议, -s 负载均衡算法
2 ipvsadm -A -t 10.10.0.1:8410 -s rr
3 // -D 删除一个VIP
4 ipvsadm -D -t 10.10.0.1:8410
5 // 将vip地址添加到本地ens33网卡上
6 ip addr add 10.10.0.1/24 dev ens33

```

为vip添加RS

```

1 // -t 表示tcp, -m 表示使用nat
2 ipvsadm -a -t 10.10.0.1:8410 -r 192.168.113.99:8080 -m
3 ipvsadm -a -t 10.10.0.1:8410 -r 192.168.113.100:8080 -m

```

添加完成RS后，查看ipvs规则，如下图所示

```

[root@docker1 ~]# ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  10.10.0.1:8410 rr
  -> 192.168.113.99:8080          Masq    1      0          2
  -> 192.168.113.100:8080         Masq    1      0          0

```

image

client端的ip地址为192.168.113.102，client端要想直接访问vip的话，需要在client端添加静态路由，添加命令如下

```
1 route add -net 10.10.0.0/16 gw 192.168.113.101
```

写下你的评论...

评论0

赞1

**简书**

发现

关注

消息

搜索



```
tcpping: verbose output= suppressed, use -v or -vv for full protocol decode
listening on interface en1, link layer EN1000 (Ethernet), capture size 262144 bytes
00:0c:29:b5:b7:6b > 00:0c:29:18:cb:f6, ethernet type IPv4 (0x8800), length 74: 192.168.113.182.35518 > 192.168.113.99.webcache: Flags [S], seq 2819148967, win 29200, options [mss 1460,sackOK,TS val 19598817 e
c 0,nop,wscale 7], length 0
00:0c:29:b5:b7:6b > 00:0c:29:b5:b7:6b, ethernet type IPv4 (0x8800), length 74: 192.168.113.99.webcache > 192.168.113.182.35518: Flags [S.], seq 2038849820, ack 19598817, win 29200, options [mss 1460,sackOK,
T5 val 57489578 ecr 19598817,nop,wscale 7], length 0
00:0c:29:b5:b7:6b > 00:0c:29:18:cb:f6, ethernet type IPv4 (0x8800), length 60: 192.168.113.182.35518 > 192.168.113.99.webcache: Flags [R], seq 2819148968, win 0, length 0
```

image

上图抓包显示，client 直接访问的vip，而数据包的目的ip 变为了rs的ip，因此可以看出ipvs 做了DNAT转换。因为做了DNAT，RS发送响应数据直接发给client，client收到RS的数据包。client给vip发的包却收到了RS的响应包(client 想我从来没有给RS发过数据)，因此client端会把此数据包丢弃。

因为ipvs没有做SNAT，接下来在DS上添加iptables规则自己实现SNAT的功能，添加完SNAT后，RS就看不到真实的CIP了。

```
1 | iptables -t nat -A POSTROUTING -m ipvs --vaddr 10.10.0.1 --vport 8410 -j MASQUERADE
```

此时还是不通，查找资料后发现ipvs 的conntrack 没有开，手动打开，后续文章介绍conntrack是什么，设置完成后可以愉快的访问了。

```
1 | sysctl net.ipv4.vs.conntrack=1
```

总结:通过ipvs提供的DNAT功能和负载均衡功能，很容易实现外部用户访问内网的需求。但是还要考虑高可用层面，比如主DS宕机VIP要漂移到备DS上，后端RS重启或宕机，ipvs负载均衡列表中要及时把有问题的RS剔除，这样才能真正的实现高可用。

1.3、conntrack

大家在家上网时用到的都是192.168.x.x的ip地址，这是私网ip地址。那么大家是如何能够成功的访问外网的呢？答案是路由器帮我们做了SNAT的功能，使我们发出的数据包的src ip变为路由器的公网ip，这样数据包就能在互联网上愉快的转发了。从而实现了对内网的保护。

那么问题来了，既然做了SNAT转换，那响应数据包回来以后路由器怎么知道转到哪台PC上呢？路由器可能链接了很多PC，不可能都给每一个PC转发吧。。。答案就是conntrack实现的。

接下来我拿上面ipvs的例子举例，我们手动实现了在DS上SNAT转换，在client上curl vip:8410，这时候查看DS上和client上的conntrack表如下

```
1 | cat /proc/net/nf_conntrack
2 | 或者
3 | conntrack -E
```

```
1 | // DS 上的记录
2 | ipv4    2  tcp      6 44 TIME_WAIT src=192.168.113.102 dst=10.10.0.1 sport=35562 dport=8410 sr
3 | // client 上的记录
4 | ipv4    2  tcp      6 38 TIME_WAIT src=192.168.113.102 dst=10.10.0.1 sport=35562 dport=8410 sr
```

先从client上的连接跟踪分析起:主要看 src、dst、sport、dport这几个字段。

client发送数据包

client端发出数据包的src ip 为192.168.113.102，dst ip 为10.10.0.1 (VIP)，sport 为35562这个端口，dport为8410(VIP 定义端口)。

写下你的评论...

评论0

赞1

[简书](#)[发现](#)[关注](#)[消息](#)[搜索](#)[beta](#)

DS接收数据包

DS接收到src ip 为CIP(192.168.113.102), dst ip 为vip(10.10.0.1), sport为35562, dport为8410的数据包

DS接收响应数据包

由于在DS侧做了DNAT转换，根据负载均衡策略找到了一个RS(RIP 192.168.113.99)，同时也做了SNAT转换(判断是否是VIP和端口)，转换为DS的DIP。所以当DS收到src ip 为192.168.113.99(RIP)，dst ip 为192.168.113.101(DIP)，sport为8080，dport为35562，会根据连接跟踪表找到这个包是192.168.113.102这个client发过来的，因此把数据包在转发给192.168.113.102:35562 上。

conntrack各个字段的含义

1. 第一列为网络协议名字，比如ipv4
2. 第二列为网络协议号
3. 第三列为传输协议名字，比如TCP
4. 第四列为传输层协议号，TCP为6
5. 第五列为该条记录剩余生存时间，当有新的数据包到来时更新此记录
6. 第六列为连接状态，此处说明client端主动关闭连接
7. [ASSURED]说明:表示在请求和响应上都看到了流量。[UNREPLIED]说明未在响应上见到流量。

总结:

本文只是简单的说明了一下conntrack，并没有具体说明数据流经netfilter时何时创建记录，数据存储的数据结构啥样，底层比较复杂，感兴趣的大佬可以自行研究~

二、k8s网络通信

介绍完了ipset、ipvs、conntrack，接下来进入正题，看一下ipvs模式下k8s的网络通信。kube-proxy的主要作用是watch apiserver，当监听到pod或service变化时，修改本地的iptables规则或ipvs规则。

2.1、clusterip模式

clusterip模式为一个集群内部可访问的ip，集群外部没办法访问这个ip，试验环境如下：

```

1 // 首先创建一个deployment, kubectl apply -f nginx.yaml
2 apiVersion: apps/v1
3 kind: Deployment
4 metadata:
5   name: nginx-deployment
6   labels:
7     app: nginx
8 spec:
9   replicas: 1
10  selector:
11    matchLabels:
12      app: nginx
13  template:
14    metadata:

```



写下你的评论...

评论0

赞1



[简书](#)[发现](#)[关注](#)[消息](#)[搜索](#)[beta](#)

```

21 |     ports:
22 |       - containerPort: 80
23 |
24 |
25 |
26 |
27 |
28 |
29 |
30 |
31 |
32 |
33 |
34 |
35 |
36 |
37 |
38 |
39 |
40 |
41 |
42 |
43 |
44 |
45 |
46 |
47 |
48 |
49 |
50 |
51 |
52 |
53 |
54 |
55 |
56 |
57 |
58 |
59 |
60 |
61 |
62 |
63 |
64 |
65 |
66 |
67 |
68 |
69 |
70 |
71 |
72 |
73 |
74 |
75 |
76 |
77 |
78 |
79 |
80 |
81 |
82 |
83 |
84 |
85 |
86 |
87 |
88 |
89 |
90 |
91 |
92 |
93 |
94 |
95 |
96 |
97 |
98 |
99 |
100 |
101 |
102 |
103 |
104 |
105 |
106 |
107 |
108 |
109 |
110 |
111 |
112 |
113 |
114 |
115 |
116 |
117 |
118 |
119 |
120 |
121 |
122 |
123 |
124 |
125 |
126 |
127 |
128 |
129 |
130 |
131 |
132 |
133 |
134 |
135 |
136 |
137 |
138 |
139 |
140 |
141 |
142 |
143 |
144 |
145 |
146 |
147 |
148 |
149 |
150 |
151 |
152 |
153 |
154 |
155 |
156 |
157 |
158 |
159 |
160 |
161 |
162 |
163 |
164 |
165 |
166 |
167 |
168 |
169 |
170 |
171 |
172 |
173 |
174 |
175 |
176 |
177 |
178 |
179 |
180 |
181 |
182 |
183 |
184 |
185 |
186 |
187 |
188 |
189 |
190 |
191 |
192 |
193 |
194 |
195 |
196 |
197 |
198 |
199 |
200 |
201 |
202 |
203 |
204 |
205 |
206 |
207 |
208 |
209 |
210 |
211 |
212 |
213 |
214 |
215 |
216 |
217 |
218 |
219 |
220 |
221 |
222 |
223 |
224 |
225 |
226 |
227 |
228 |
229 |
230 |
231 |
232 |
233 |
234 |
235 |
236 |
237 |
238 |
239 |
240 |
241 |
242 |
243 |
244 |
245 |
246 |
247 |
248 |
249 |
250 |
251 |
252 |
253 |
254 |
255 |
256 |
257 |
258 |
259 |
260 |
261 |
262 |
263 |
264 |
265 |
266 |
267 |
268 |
269 |
270 |

```



创建完deployment和service后，查看一下service的ip如下。

```
[root@k8s-master01 svc]# kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP,70<sup>小左的技术轮子</sup>,80/TCP   73d
my-service   ClusterIP   10.108.113.237    <none>        80/TCP      11s
image
```

接下来看下宿主机网卡、ipvs规则、ipset规则有什么变化

```

1 // 可以看到，kube-ipvs0 上多了创建的service的clusterIp。
2 // 可以看到，kube-ipvs0 上多了创建的service的clusterIp。
3 kube-ipvs0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default
4 ...
5     inet 10.108.113.237/32 brd 10.108.113.237 scope global kube-ipvs0
6         valid_lft forever preferred_lft forever
7 // 查看ipset规则，多了刚创建的clusterIp，k8s内部的serviceIp省略
8 Name: KUBE-CLUSTER-IP
9 Type: hash:ip,port
10 Revision: 5
11 Header: family inet hashsize 1024 maxelem 65536
12 Size in memory: 632
13 References: 2
14 Number of entries: 8
15 Members:
16 ...
17 10.108.113.237,tcp:80
18 // 查看ipvs规则
19 TCP 10.108.113.237:80 rr
20     -> 10.244.1.143:80          Masq    1      0      0

```

查看iptables 的nat表和filter表，看一下k8s创建了哪些规则以及经过哪些链

写下你的评论...

评论0

赞1

简书

发现

关注

消息

搜索



beta



image

```

Chain INPUT (policy ACCEPT 2510 packets, 413K bytes)
pkts bytes target prot opt in out source destination
292K 57M KUBE-FIREWALL all -- * * 0.0.0.0/0 0.0.0.0/0
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target prot opt in out source destination
1594 131K DOCKER-USER all -- * * 0.0.0.0/0 0.0.0.0/0 /* kubernetes forwarding rules */
1594 131K DOCKER-ISOLATION-STAGE-1 all -- * * 0.0.0.0/0 0.0.0.0/0 ctstate RELATED,ESTABLISHED
0 0 ACCEPT all -- * docker0 0.0.0.0/0 0.0.0.0/0
0 0 ACCEPT all -- * docker0 docker0 0.0.0.0/0 0.0.0.0/0
0 0 ACCEPT all -- docker0 docker0 0.0.0.0/0 0.0.0.0/0
0 0 ACCEPT all -- * 10.244.0.0/16 0.0.0.0/0
0 0 ACCEPT all -- * 0.0.0.0/0 10.244.0.0/16
Chain OUTPUT (policy ACCEPT 2503 packets, 471K bytes)
pkts bytes target prot opt in out source destination
292K 63M KUBE-FIREWALL all -- * * 0.0.0.0/0 0.0.0.0/0
Chain DOCKER (1 references)
pkts bytes target prot opt in out source destination
Chain DOCKER-ISOLATION-STAGE-1 (1 references)
pkts bytes target prot opt in out source destination
0 0 DOCKER-ISOLATION-STAGE-1 all -- docker0 docker0 0.0.0.0/0 0.0.0.0/0
1594 131K RETURN all -- * 0.0.0.0/0 0.0.0.0/0
Chain DOCKER-ISOLATION-STAGE-2 (1 references)
pkts bytes target prot opt in out source destination
0 0 DROP all -- * docker0 0.0.0.0/0 0.0.0.0/0
0 0 RETURN all -- * 0.0.0.0/0 0.0.0.0/0
Chain DOCKER-USER (1 references)
pkts bytes target prot opt in out source destination
1594 131K RETURN all -- * 0.0.0.0/0 0.0.0.0/0
Chain KUBE-FIREWALL (2 references)
pkts bytes target prot opt in out source destination
0 0 DROP all -- * * 0.0.0.0/0 0.0.0.0/0 /* kubernetes firewall for dropping marked packets */
mark match 0x8000/0x8000
Chain KUBE-FORWARD (1 references)
pkts bytes target prot opt in out source destination
0 0 ACCEPT all -- * * 0.0.0.0/0 0.0.0.0/0 /* kubernetes forwarding rules */
mark match 0x4000/0x4000
64 720K ACCEPT all -- * * 10.244.0.0/16 0.0.0.0/0 /* kubernetes forwarding conntrack pod source rule */
8 530 ACCEPT all -- * 0.0.0.0/0 10.244.0.0/16 /* kubernetes forwarding conntrack pod destination rule */
ctstate RELATED,ESTABLISHED

```

第四步

filter表



image

接下来分析一下curl 10.108.113.237 数据是如何走的，只讨论在nat表和filter表的流向，因为在mangle和raw都没有规则。

1、nat表PREROUTING链

- ①:数据首先进入PREROUTING链，所有请求都会进入KUBE-SERVICES链。
- ②:进入KUBE-SERVICES后，查看对应在此链上的规则，发现请求的目的ip和port在KUBE-CLUSTER-IP 对应的ipset里面(上面已有展示)，匹配上了则跳往KUBE-MARK-MASQ链。

```
1 | -A KUBE-SERVICES ! -s 10.244.0.0/16 -m comment --comment "Kubernetes service cluster ip + port"
```

③:数据流向KUBE-MARK-MASQ链，主要做了mark 打标记的功能，iptables命令如下

```
1 | -A KUBE-MARK-MASQ -j MARK --set-xmark 0x4000/0x4000
```

- ④:之后走向KUBE-NODE-PORT链，因为没有定义nodepode类型的service，此处先略过。2、filter表的INPUT链

- ⑤:首先进入INPUT链，所有数据转向KUBE-FIREWALL链。

- ⑥:进入KUBE-FIREWALL链，如果发现数据包打了0x8000/0x8000，DROP掉。因为ipvs工作在INPUT链上，做完DNAT之后直接转发到POSTROUTING链上。

3、nat表POSTROUTING链

- ⑦:进入POSTROUTING链，所有数据转向KUBE-POSTROUTING链

- ⑧:进入KUBE-POSTROUTING链，对有0x4000/0x4000标记的数据包做SNAT转换，因为ipvs只有DNAT功能。

```
1 | -A KUBE-POSTROUTING -m comment --comment "kubernetes service traffic requiring SNAT" -m mark -
```

4、数据转发给flannel网卡，进行转发

- ⑨:flannel 根据具体的backend模式，对数据做封包等操作，然后发出去。flannel的网络模式比较复杂，之后会专门文章进行说明。

2.2. nodeport模式

写下你的评论...

评论0

赞1

[简书](#)[发现](#)[关注](#)[消息](#)[搜索](#)[beta](#)

```

1 // 此处在每个node 上开了一个30080端口
2 apiVersion: v1
3 kind: Service
4 metadata:
5   name: my-service
6 spec:
7   selector:
8     app: nginx
9   type: NodePort
10  ports:
11    - protocol: TCP
12      nodePort: 30080
13      port: 80

```



查看创建service的信息，发现也创建了集群内部的一个ip。

```

[root@k8s-master01 svc]# kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kubernetes  ClusterIP  10.96.0.1      <none>        443/TCP      20m
my-service  NodePort  10.111.174.22  <none>        80:30080/TCP  20m

```

[image](#)

iptables规则如下

```

Chain PREROUTING (policy ACCEPT 24 packets, 2164 bytes)
pkts bytes target  prot opt in   out  source               destination
1728 167K KUBE-SERVICES all  -- *   *   0.0.0.0/0          0.0.0.0/0          /* kubernetes service portals */
710 86269 DOCKER  all  -- *   *   0.0.0.0/0          0.0.0.0/0          ADDRTYPE match dst-type LOCAL
Chain INPUT (policy ACCEPT 6 packets, 839 bytes)
pkts bytes target  prot opt in   out  source               destination
Chain OUTPUT (policy ACCEPT 29 packets, 2764 bytes)
pkts bytes target  prot opt in   out  source               destination
2316 282K KUBE-SERVICES all  -- *   *   0.0.0.0/0          0.0.0.0/0          /* kubernetes service portals */
285 17100 DOCKER  all  -- *   *   0.0.0.0/0          0.0.0.0/0          ADDRTYPE match dst-type LOCAL
Chain POSTROUTING (policy ACCEPT 35 packets, 3294 bytes)
pkts bytes target  prot opt in   out  source               destination
556 326K KUBE-POSTROUTING all  -- *   *   0.0.0.0/0          0.0.0.0/0          /* kubernetes postrouting rules */
0 0 MASQUERADE  all  -- *   *   10.244.0.0/16       0.0.0.0/0
1247 89987 RETURN   all  -- *   *   10.244.0.0/16       10.244.0.0/16
583 39117 MASQUERADE all  -- *   *   10.244.0.0/16       10.244.0.0/24
0 0 MASQUERADE  all  -- *   *   10.244.0.0/16       10.244.0.0/16
Chain DOCKER (2 references)
pkts bytes target  prot opt in   out  source               destination
0 0 RETURN   all  -- docker0 *   0.0.0.0/0          0.0.0.0/0
Chain KUBE-POORT (0 references)
pkts bytes target  prot opt in   out  source               destination
0 0 KUBE-MARK-DROP all  -- *   *   0.0.0.0/0          0.0.0.0/0
Chain KUBE-LOAD-BALANCER (0 references)
pkts bytes target  prot opt in   out  source               destination
0 0 KUBE-MARK-DROP all  -- *   *   0.0.0.0/0          0.0.0.0/0
Chain KUBE-MARK-MASQ (1 references)
pkts bytes target  prot opt in   out  source               destination
0 0 MARK    all  -- *   *   0.0.0.0/0          0.0.0.0/0          MARK or 0x8000
Chain KUBE-MARK-MASQ (3 references)
pkts bytes target  prot opt in   out  source               destination
0 0 MARK    all  -- *   *   0.0.0.0/0          0.0.0.0/0          MARK or 0x4000
Chain KUBE-NODE-PORT (1 references)
pkts bytes target  prot opt in   out  source               destination
0 0 KUBE-MARK-MASQ tcp  -- *   *   0.0.0.0/0          0.0.0.0/0          → ② Kubeernetes nodeport TCP port for masquerade purpose */
      KUBE-NODE-PORT-TCP dst
Chain KUBE-POSTROUTING (1 references)
pkts bytes target  prot opt in   out  source               destination
0 0 KUBE-MARK-MASQ all  -- *   *   0.0.0.0/0          0.0.0.0/0          /* kubernetes service traffic requiring SNAT */
0 0 MASQUERADE  all  -- *   *   0.0.0.0/0          0.0.0.0/0          /* Kubeernetes endpoints dst ip:port, source ip for solving hairpin purpose */
      match-set KUBE-LOOPBACK dst,dst,src
Chain KUBE-SERVICES (2 references)
pkts bytes target  prot opt in   out  source               destination
0 0 KUBE-MARK-MASQ all  -- *   *   !10.244.0.0/16    0.0.0.0/0          /* Kubeernetes service cluster ip + port for masquerade purpose */
      match-set KUBE-CLUSTER-IP dst,dst
16 1430 KUBE-NODE-PORT all  -- *   *   0.0.0.0/0          0.0.0.0/0          ADDRTYPE match dst-type LOCAL
0 0 ACCEPT   all  -- *   *   0.0.0.0/0          0.0.0.0/0

```

[小左的技术轮子](#)[image](#)

写下你的评论...

评论0

赞1

简书

发现

关注

消息

搜索



beta



接下来看下ipset规则有什么变化，发现KUBE-NODE-PORT-TCP下的一个成员是刚才我们指定的那个nodePort的值。

```

1 | Name: KUBE-NODE-PORT-TCP
2 | Name: KUBE-NODE-PORT-TCP
3 | Type: bitmap:port
4 | Revision: 3
5 | Header: range 0-65535
6 | Size in memory: 8300
7 | References: 1
8 | Number of entries: 1
9 | Members:
10 | 30080

```



接下来看一下iptables规则，nat表和filter表

1、nat表PREROUTING链

- ①:数据首先进入PREROUTING链，所有请求都会进入KUBE-SERVICES链。
- ②:ip和port匹配不上KUBE-CLUSTER-IP 的ipset，判断是访问的本地地址，进入KUBE-NODE-PORT链。

```
1 | -A KUBE-SERVICES -m addrtype --dst-type LOCAL -j KUBE-NODE-PORT
```

- ③:进入KUBE-NODE-PORT链后，判断访问端口在 KUBE-NODE-PORT-TCP ipset规则中，因此进入KUBE-MARK-MASQ链。

```
1 | -A KUBE-NODE-PORT -p tcp -m comment --comment "Kubernetes nodeport TCP port for masquerade pur
```

- ④:进入KUBE-MARK-MASQ链，对数据做mark标记

```
1 | -A KUBE-MARK-MASQ -j MARK --set-xmark 0x4000/0x4000
```

后续流程跟clusterip一样，此处就不在阐述。

2.3、dns相关

k8s中的dns默认使用的是coredns，通过以下命令查看。k8s中定义的service是有域名的，访问域名要通过dns解析，此时coredns就发挥它的作用了。

```
[root@k8s-master01 ~]# kubectl get pod -n kube-system
NAME                               READY   STATUS    RESTARTS   AGE
coredns-5c98db65d4-csfks          1/1    Running   204        75d
coredns-5c98db65d4-j759s          1/1    Running   2          2d21h
etcd-k8s-master01                 1/1    Running   17         79d
kube-apiserver-k8s-master01       1/1    Running   27         79d
kube-controller-manager-k8s-master01 1/1    Running   37         79d
kube-flannel-ds-amd64-rhw9s       1/1    Running   18         79d
kube-flannel-ds-amd64-vjxxn       1/1    Running   17         79d
kube-proxy-fhg1b                  1/1    Running   17         79d
kube-proxy-l4c19                  1/1    Running   18         79d
kube-scheduler-k8s-master01       1/1    Running   35         79d
```

image

```

1 | // 查看某个pod内dns server 的ip
2 | kubectl exec nginx-deployment-7fd6966748-qxsrf -it -- cat /etc/resolv.conf
3 | nameserver 10.96.0.10
4 | search default.svc.cluster.local svc.cluster.local cluster.local

```

写下你的评论...

评论0

赞1

简书

发现

关注

消息

搜索



beta



ipvs，如何实现，端口映射的实现，入出口匹配。

```
1 | nslookup my-service.default.svc.cluster.local 10.96.0.10
```

```
[root@k8s-master01 ~]# nslookup my-service.default.svc.cluster.local 10.96.0.10
Server:      10.96.0.10
Address:     10.96.0.10#53
```

```
Name:   my-service.default.svc.cluster.local
Address: 10.111.174.22
```

```
[root@k8s-master01 ~]# kubectl get svc
NAME        TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
kubernetes  ClusterIP  10.96.0.1    <none>        443/TCP       79d
my-service  NodePort   10.111.174.22 <none>        80:30080/TCP  74m
[root@k8s-master01 ~]#
```



image

参考:

- <https://www.cnblogs.com/CasonChan/p/5319364.html>
- <https://zhuanlan.zhihu.com/p/94418251>

以上相关内容介绍了k8s service ipvs的相关实现，如有错误欢迎指出~



日记本

...

"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



左小星

总资产2(约0.11元) 共写了1.9W字 获得34个赞 共21个粉丝

关注



智慧井盖



网络工程师培训机构



加工中心cnc



mt4平台下载



邮件服务器搭建



封闭母线



写下你的评论...

全部评论 0 只看作者

按时间倒序 按时间正序



被以下专题收入 发现更多相似内容

写下你的评论...

评论0

赞1

简书

发现

关注

消息

搜索

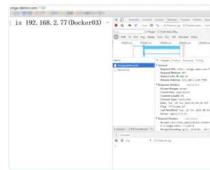


beta

**推荐阅读****史上最全的Nginx配置参数中文说明**来自：SegmentFault，作者：Ably链接：<https://segmentfault.com/a/119...>

夜空_2cd3 阅读 2,237 评论 1 赞 82

更多精彩内容 >

**这样轻松两步，我在SpringBoot 服务上实现了接口限流**

Sentinel是阿里巴巴开源的限流器熔断器，并且带有可视化操作界面。在日常开发中，限流功能时常被使用，用于对某...

马士兵老师 阅读 466 评论 0 赞 6

**就是要让你搞懂Nginx，这篇就够了！**来自：CSDN，作者：渐暖链接：<https://blog.csdn.net/yujing1314/article...>

码农小光 阅读 2,798 评论 1 赞 60

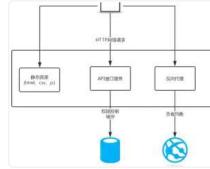
**k8s源码部署01**k8s源码集群搭建 集群搭建： 1、生产环境k8s平台规划 老男孩老师文档：
<https://www.cnblogs.com/>

MingxD_d 阅读 93 评论 0 赞 0

1. 生产环境K8S平台规划
2. 服务端硬件配置推荐
3. 官方提供三种部署方式
4. 为Etcd和APIServer自签SSL证书
5. Etcd数据集群部署
6. 部署Master组件
7. 部署Node组件
8. 部署K8S集群网络
9. 部署Web UI (Dashboard)
- 10.部署集群内部DNS解析服务 (CoreDNS)

NginxNginx应用场景 静态资源服务器 反向代理服务 API接口服务(Lua&Javascript)1.jpg
Ngi...

强某某 阅读 607 评论 2 赞 19



写下你的评论...

评论0

赞1