38 | 从外界连通Service与Service调试“三板斧”

38 | 从外界连通Service与Service调试“三板斧”

张磊 2018-11-19

11:03

讲述：张磊 大小：5.07M

你好，我是张磊。今天我和你分享的主题是：从外界连通 Service 与 Service 调试“三板斧”。

在上一篇文章中，我为你介绍了 Service 机制的工作原理。通过这些讲解，你应该能够明白这样一个事实：Service 的访问信息在 Kubernetes 集群之外，其实是无效的。

这其实也容易理解：所谓 Service 的访问入口，其实就是每台宿主机上由 kube-proxy 生成的 iptables 规则，以及 kube-dns 生成的 DNS 记录。而一旦离开了这个集群，这些信息对用户来说，也就自然没有作用了。

所以，在使用 Kubernetes 的 Service 时，一个必须要面对和解决的问题就是：**如何从外部（Kubernetes 集群之外），访问到 Kubernetes 里创建的 Service?**

这里最常用的一种方式就是：NodePort。我来为你举个例子。

```
apiVersion: v1
kind: Service
metadata:
name: my-nginx
labels:
run: my-nginx
spec:
type: NodePort
ports:
- nodePort: 8080
targetPort: 80
protocol: TCP
name: http
- nodePort: 443
protocol: TCP
name: https
selector:
run: my-nginx
```

□复制代码

在这个 Service 的定义里，我们声明它的类型是，type=NodePort。然后，我在 ports 字段里声明了 Service 的 8080 端口代理 Pod 的 80 端口，Service 的 443 端口代理 Pod 的 443 端口。

当然，如果你不显式地声明 nodePort 字段，Kubernetes 就会为你分配随机的可用端口来设置代理。这个端口的范围默认是 30000-32767，你可以通过 kube-apiserver 的–service-node-port-range 参数来修改它。

那么这时候，要访问这个 Service，你只需要访问：

```
< 任何一台宿主机的 IP 地址 >:8080
```

□复制代码

就可以访问到某一个被代理的 Pod 的 80 端口了。

而在理解了我在上一篇文章中讲解的 Service 的工作原理之后，NodePort 模式也就非常容易理解了。显然，kube-proxy 要做的，就是在每台宿主机上生成这样一条 iptables 规则：

```
-A KUBE-NODEPORTS -p tcp -m comment --comment "default/my-nginx: nodePort" -m tcp --dport 8080 -j KUBE-SVC-67RL4FN6JRUPOJYM
```

复制代码

而我在上一篇文章中已经讲到，KUBE-SVC-67RL4FN6JRUPOJYM 其实就是一组随机模式的 iptables 规则。所以接下来的流程，就跟 ClusterIP 模式完全一样了。

需要注意的是，在 NodePort 方式下，Kubernetes 会在 IP 包离开宿主机发往目的 Pod 时，对这个 IP 包做一次 SNAT 操作，如下所示：

```
-A KUBE-POSTROUTING -m comment --comment "kubernetes service traffic requiring SNAT" -m mark --mark 0x4000/0x4000 -j MASQUERADE
```

复制代码

可以看到，这条规则设置在 POSTROUTING 检查点，也就是说，它给即将离开这台主机的 IP 包，进行了一次 SNAT 操作，将这个 IP 包的源地址替换成了这台宿主机上的 CNI 网桥地址，或者宿主机本身的 IP 地址（如果 CNI 网桥不存在的话）。

当然，这个 SNAT 操作只需要对 Service 转发出来的 IP 包进行（否则普通的 IP 包就被影响了）。而 iptables 做这个判断的依据，就是查看该 IP 包是否有一个“0x4000”的“标志”。你应该还记得，这个标志正是在 IP 包被执行 DNAT 操作之前被打上去的。

可是，**为什么一定要对流出的包做 SNAT操作呢?**

这里的原理其实很简单，如下所示：

```
client
\ ^
\ \
v \
node 1 <--- node 2
| ^ SNAT
| | --->
```

```
v |
endpoint
```

□复制代码

当一个外部的 client 通过 node 2 的地址访问一个 Service 的时候，node 2 上的负载均衡规则，就可能把这个 IP 包转发给一个在 node 1 上的 Pod。这里没有任何问题。

而当 node 1 上的这个 Pod 处理完请求之后，它就会按照这个 IP 包的源地址发出回复。

可是，如果没有做 SNAT 操作的话，这时候，被转发来的 IP 包的源地址就是 client 的 IP 地址。**所以此时，Pod 就会直接将回复发给client。**对于 client 来说，它的请求明明发给了 node 2，收到的回复却来自 node 1，这个 client 很可能会报错。

所以，在上图中，当 IP 包离开 node 2 之后，它的源 IP 地址就会被 SNAT 改成 node 2 的 CNI 网桥地址或者 node 2 自己的地址。这样，Pod 在处理完成之后就会先回复给 node 2（而不是 client），然后再由 node 2 发送给 client。

当然，这也就意味着这个 Pod 只知道该 IP 包来自于 node 2，而不是外部的 client。对于 Pod 需要明确知道所有请求来源的场景来说，这是不可以的。

所以这时候，你就可以将 Service 的 spec.externalTrafficPolicy 字段设置为 local，这就保证了所有 Pod 通过 Service 收到请求之后，一定可以看到真正的、外部 client 的源地址。

而这个机制的实现原理也非常简单：**这时候，一台宿主机上的 iptables 规则，会设置为只将 IP 包转发给运行在这台宿主机上的 Pod**。所以这时候，Pod 就可以直接使用源地址将回复包发出，不需要事先进行 SNAT 了。这个流程，如下所示：

```
client
^ / \
/ / \
/ v X
node 1 node 2
^ |
| |
| v
endpoint
```

□复制代码

当然，这也就意味着如果在一台宿主机上，没有任何一个被代理的 Pod 存在，比如上图中的 node 2，那么你使用 node 2 的 IP 地址访问这个 Service，就是无效的。此时，你的请求会直接被 DROP 掉。

从外部访问 Service 的第二种方式，适用于公有云上的 Kubernetes 服务。这时候，你可以指定一个 LoadBalancer 类型的 Service，如下所示：

```
---
kind: Service
apiVersion: v1
metadata:
name: example-service
spec:
ports:
- port: 8765
targetPort: 9376
selector:
app: example
type: LoadBalancer
```

□复制代码

在公有云提供的 Kubernetes 服务里，都使用了一个叫作 CloudProvider 的转接层，来跟公有云本身的 API 进行对接。所以，在上述 LoadBalancer 类型的 Service 被提交后，Kubernetes 就会调用 CloudProvider 在公有云上为你创建一个负载均衡服务，并且把被代理的 Pod 的 IP 地址配置给负载均衡服务做后端。

而第三种方式，是 Kubernetes 在 1.7 之后支持的一个新特性，叫作 ExternalName。举个例子：

```
kind: Service
apiVersion: v1
metadata:
name: my-service
spec:
type: ExternalName
externalName: my.database.example.com
```

□复制代码

在上述 Service 的 YAML 文件中，我指定了一个 externalName=my.database.example.com 的字段。而且你应该会注意到，这个 YAML 文件里不需要指定 selector。

这时候，当你通过 Service 的 DNS 名字访问它的时候，比如访问：my-service.default.svc.cluster.local。那么，Kubernetes 为你返回的就是 my.database.example.com。所以说，ExternalName 类型的 Service，其实是在 kube-dns 里为你添加了一条 CNAME 记录。这时，访问 my-service.default.svc.cluster.local 就和访问 my.database.example.com 这个域名是一个效果了。

此外，Kubernetes 的 Service 还允许你为 Service 分配公有 IP 地址，比如下面这个例子：

```
kind: Service
apiVersion: v1
metadata:
name: my-service
spec:
selector:
app: MyApp
ports:
- name: http
protocol: TCP
port: 80
targetPort: 9376
externalIPs:
- 80.11.12.10
```

复制代码

在上述 Service 中，我为它指定的 externalIPs=80.11.12.10，那么此时，你就可以通过访问 80.11.12.10:80 访问到被代理的 Pod 了。不过，在这里 Kubernetes 要求 externalIPs 必须是至少能够路由到一个 Kubernetes 的节点。你可以想一想这是为什么。

实际上，在理解了 Kubernetes Service 机制的工作原理之后，很多与 Service 相关的问题，其实都可以通过分析 Service 在宿主机上对应的 iptables 规则（或者 IPVS 配置）得到解决。

比如，当你的 Service 没办法通过 DNS 访问到的时候。你就需要区分到底是 Service 本身的配置问题，还是集群的 DNS 出了问题。一个行之有效的方法，就是检查 Kubernetes 自己的 Master 节点的 Service DNS 是否正常：

```
# 在一个 Pod 里执行
$ nslookup kubernetes.default
Server: 10.0.0.10
Address 1: 10.0.0.10 kube-dns.kube-system.svc.cluster.local
Name: kubernetes.default
Address 1: 10.0.0.1 kubernetes.default.svc.cluster.local
```

复制代码

如果上面访问 kubernetes.default 返回的值都有问题，那你就需要检查 kube-dns 的运行状态和日志了。否则的话，你应该去检查自己的 Service 定义是不是有问题。

而如果你的 Service 没办法通过 ClusterIP 访问到的时候，你首先应该检查的是这个 Service 是否有 Endpoints：

```
$ kubectl get endpoints hostnames
NAME ENDPOINTS
hostnames 10.244.0.5:9376,10.244.0.6:9376,10.244.0.7:9376
```

复制代码

需要注意的是，如果你的 Pod 的 readniessProbe 没通过，它也不会出现在 Endpoints 列表里。

而如果 Endpoints 正常，那么你就需要确认 kube-proxy 是否在正确运行。在我们通过 kubeadm 部署的集群里，你应该看到 kube-proxy 输出的日志如下所示：

```
I1027 22:14:53.995134 5063 server.go:200] Running in resource-only container "/kube-proxy"
I1027 22:14:53.998163 5063 server.go:247] Using iptables Proxier.
I1027 22:14:53.999055 5063 server.go:255] Tearing down userspace rules. Errors here are acceptable.
I1027 22:14:54.038140 5063 proxier.go:352] Setting endpoints for "kube-system/kube-dns:dns-tcp" to [10.244.1.3:53]
I1027 22:14:54.038164 5063 proxier.go:352] Setting endpoints for "kube-system/kube-dns:dns" to [10.244.1.3:53]
```

```
I1027 22:14:54.038209 5063 proxier.go:352] Setting endpoints for "default/kubernetes:https" to [10.240.0.2:443]
I1027 22:14:54.038238 5063 proxier.go:429] Not syncing iptables until Services and Endpoints have been received from master
I1027 22:14:54.040048 5063 proxier.go:294] Adding new service "default/kubernetes:https" at 10.0.0.1:443/TCP
I1027 22:14:54.040154 5063 proxier.go:294] Adding new service "kube-system/kube-dns:dns" at 10.0.0.10:53/UDP
I1027 22:14:54.040223 5063 proxier.go:294] Adding new service "kube-system/kube-dns:dns-tcp" at 10.0.0.10:53/TCP
```

复制代码

如果 kube-proxy 一切正常，你就应该仔细查看宿主机上的 iptables 了。而**一个 iptables 模式的 Service 对应的规则，我在上一篇以及这一篇文章里已经全部介绍到了，它们包括：**

1. KUBE-SERVICES 或者 KUBE-NODEPORTS 规则对应的 Service 的入口链，这个规则应该与 VIP 和 Service 端口一一对应；
2. KUBE-SEP-(hash) 规则对应的 DNAT 链，这些规则应该与 Endpoints 一一对应；
3. KUBE-SVC-(hash) 规则对应的负载均衡链，这些规则的数目应该与 Endpoints 数目一致；
4. 如果是 NodePort 模式的话，还有 POSTROUTING 处的 SNAT 链。

通过查看这些链的数量、转发目的地址、端口、过滤条件等信息，你就能很容易发现一些异常的蛛丝马迹。

当然，**还有一种典型问题，就是 Pod 没办法通过 Service 访问到自己。**这往往就是因为 kubelet 的 hairpin-mode 没有被正确设置。关于 Hairpin 的原理我在前面已经介绍过，这里就不再赘述了。你只需要确保将 kubelet 的 hairpin-mode 设置为 hairpin-veth 或者 promiscuous-bridge 即可。

> *这里，你可以再回顾下第 34 篇文章《Kubernetes 网络模型与 CNI 网络插件》中的相关内容。*

其中，在 hairpin-veth 模式下，你应该能看到 CNI 网桥对应的各个 VETH 设备，都将 Hairpin 模式设置为了 1，如下所示：

```
$ for d in /sys/devices/virtual/net/cni0/brif/veth*/hairpin_mode; do echo "$d = $(cat $d)"; done
/sys/devices/virtual/net/cni0/brif/veth4bfbfe74/hairpin_mode = 1
/sys/devices/virtual/net/cni0/brif/vethfc2a18c5/hairpin_mode = 1
```

□复制代码

而如果是 promiscuous-bridge 模式的话，你应该看到 CNI 网桥的混杂模式（PROMISC）被开启，如下所示：

```
$ ifconfig cni0 |grep PROMISC
UP BROADCAST RUNNING PROMISC MULTICAST MTU:1460 Metric:1
```

□复制代码

总结

在本篇文章中，我为你详细讲解了从外部访问 Service 的三种方式（NodePort、LoadBalancer 和 External Name）和具体的工作原理。然后，我还为你讲述了当 Service 出现故障的时候，如何根据它的工作原理，按照一定的思路去定位问题的可行之道。

通过上述讲解不难看出，所谓 Service，其实就是 Kubernetes 为 Pod 分配的、固定的、基于 iptables（或者 IPVS）的访问入口。而这些访问入口代理的 Pod 信息，则来自于 Etcd，由 kube-proxy 通过控制循环来维护。

并且，你可以看到，Kubernetes 里面的 Service 和 DNS 机制，也都不具备强多租户能力。比如，在多租户情况下，每个租户应该拥有一套独立的 Service 规则（Service 只应该看到和代理同一个租户下的 Pod）。再比如 DNS，在多租户情况下，每个租户应该拥有自己的 kube-dns（kube-dns 只应该为同一个租户下的 Service 和 Pod 创建 DNS Entry）。

当然，在 Kubernetes 中，kube-proxy 和 kube-dns 其实也是普通的插件而已。你完全可以根据自己的需求，实现符合自己预期的 Service。

思考题

为什么 Kubernetes 要求 externalIPs 必须是至少能够路由到一个 Kubernetes 的节点?