一文搞懂 Kubernetes 如何实现 DNS 解析

2021-02-22 16:45:56

最近在处理 Kuberntes 中的 DNS 解析问题，正好借这个机会学习下 Kubernetes 中的 DNS 服务器工作原理，处理的 DNS 服务器问题会稍后再水一篇博客介绍.

我对解析过程的了解也比较粗浅，仅介绍下配置中的内容.

Pod 中的 DNS 概览

众所周知，DNS 服务器用于将域名转换为 IP (具体为啥要转换建议复习下 7 层网络模型). Linux 服务器中 DNS 解析配置位于 `/etc/resolv.conf`，在 Pod 中也不例外，下面是某个 Pod 中的配置：

```
nameserver 10.96.0.10
search kube-system.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

假如我们平时想要修改自己本机上的 DNS 服务器，比如想要修改为 `8.8.8.8`，就会这么去修改：

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

如果想要调试 DNS 服务器，测试返回结果，可以使用 dig 工具：

```
> dig baidu.com @8.8.8.8

; <<>> DiG 9.16.10 <<>> baidu.com @8.8.8.8
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5114
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;baidu.com.   IN A

;; ANSWER SECTION:
baidu.com.  159 IN A 39.156.69.79
baidu.com.  159 IN A 220.181.38.148

;; Query time: 10 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Tue Jan 12 09:26:13 HKT 2021
;; MSG SIZE  rcvd: 70
```

DNS 服务器 – nameserver

我们先从 `nameserver 10.96.0.10` 来看，为什么请求这个地址可以进行 DNS 解析. 这个答案就是 `iptables`，我仅截取 UDP 的 53 端口，以下内容可以通过 `iptables-save` 获得.

```
-A KUBE-SERVICES -d 10.96..10/32 -p udp -m comment --comment "kube-system/kube-dns:dns cluster IP" -m udp --
dport 53 -j KUBE-SVC-TCOU7JCQXEZGVUNU
# 简单解释下，这条规则表示，如果目标地址是 10.96.0.10的udp53端口，那么就会跳转到这条链上`KUBE-SVC-TCOU7JCQXEZGVUNU`
```

我们再看下这条链 `KUBE-SVC-TCOU7JCQXEZGVUNU`：

```
-A KUBE-SVC-TCOU7JCQXEZGVUNU -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-
Q3HNNZPXUAYYDXW2
-A KUBE-SVC-TCOU7JCQXEZGVUNU -j KUBE-SEP-BBR3Z5NWFGXGVHEZ

-A KUBE-SEP-Q3HNNZPXUAYYDXW2 -p udp -m udp -j DNAT --to-destination 172.32.3.219:53
-A KUBE-SEP-BBR3Z5NWFGXGVHEZ -p udp -m udp -j DNAT --to-destination 172.32.6.239:53

# 联系之前的规则，这几条规则完整的意思是：

# 本机中，发给10.96.0.10:53的流量，一半转发到172.32.3.219:53，另一半转发到172.32.6.239:53
```

Kubernetes 的 Deployment

再看下我们的 Kubernetes 中 Pod 的 IP 地址，也就是说，DNS 请求实际上会到我们的 Coredns 容器中被处理.

```
> kubectl -n kube-system get pods -o wide | grep dns
coredns-646bc69b8d-
jd22w                              1/1     Running            57d    172.32.6.239    m1  <none>
  <none>
coredns-646bc69b8d-
p8pqq                              1/1     Running   8        315d   172.32.3.219    m2  <none>
  <none>
```

Kubernetes 中 Service 的具体实现

再查看下对应的 Service，可以看到，上述机器中的 Iptables 其实就是 Service 的具体实现方式.

```
> kubectl -n kube-system get svc | grep dns
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   398d
```

可能有人会有疑问，现在是 2 个 Pod 可以均分流量，如果是 3 个，4 个 Pod，Iptables 是如何做转发的呢，正好我有这个疑问，因此我就再加了 2 个 Pod，看看 `iptables` 是怎么实现对于 4 个 Pod 均分流量的.

这是最后的实现方式：

```
-A KUBE-SVC-TCOU7JCQXEZGVUNU -m statistic --mode random --probability 0.25000000000 -j KUBE-SEP-
HTZHQHQPOHVVNWZS
-A KUBE-SVC-TCOU7JCQXEZGVUNU -m statistic --mode random --probability 0.33333333349 -j KUBE-SEP-
3VNFB2SPYQ3RRPK6
-A KUBE-SVC-TCOU7JCQXEZGVUNU -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-
Q3HNNZPXUAYYDXW2
-A KUBE-SVC-TCOU7JCQXEZGVUNU -j KUBE-SEP-B8R3Z5NWFGXGVHEZ
```

这些语句的意思应该是：

1. 前 1/4 的流量到一条链中，剩 3/4
2. 剩下 3/4 的流量，1/3到一条链，剩 2/4
3. 剩下 2/4 的浏览，1/2到一条链，剩 1/4
4. 最后 1/4 到一条链

通过这样的方式对流量进行了均分，还是挺巧妙的，这样，5个，10个也是可以依次去分的.

resolv.conf 中其他参数解析

```
search kube-system.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

详细的介绍可以看这里：resolv.conf 手册，我简单的说下我的理解.

search 参数

假如没有这个 `search` 参数，我们查找时：

```
> ping kube-dns
ping: kube-dns: Name or service not known
```

如果增加了 `search` 参数后，再去查找：

```
> ping kube-dns
PING kube-dns.kube-system.svc.pslgor-dev.nease.net (10.96.0.10) 56(84) bytes of data.
```

可以看到，解析域名时，如果给定的域名无法查找，会添加 `search` 后面的后缀进行查找(假如以 `.` 结尾，类似 `kube-dns.` ，这样的域名不会再去尝试，FQDN域名).

`search` 的工作就是帮我们去尝试，用在 Kubenetes 中，配置 `kube-system.svc.cluster.local svc.cluster.local cluster.local` 就会帮我们尝试，我们 `ping abc` ，就会这样进行查询

```
[INFO] 10.202.37.232:50940 - 51439 "A IN abc.kube-
system.svc.cluster.local. udp 51 false 512" NXDOMAIN qr,aa,rd 144 0.000114128s
[INFO] 10.202.37.232:51823 - 54524 "A IN abc.svc.cluster.local. udp 39 false 512" NXDOMAIN qr,aa,rd 132 0.00012
4048s
[INFO] 10.202.37.232:41894 - 15434 "A IN abc.cluster.local. udp 35 false 512" NXDOMAIN qr,aa,rd 128 0.000092304
s
[INFO] 10.202.37.232:40357 - 43160 "A IN abc. udp 21 false 512" NOERROR qr,aa,rd,ra 94 0.000163406s
```

ndots 以及其优化问题

`search` 配置需要与 `ndots` 一起使用，默认的 `ndots` 是 1，它的作用是：如果检查到被查询的域名中 `dot` 的数量小于该值时，就会优先尝试添加 `search` 域中的后缀.

```
Resolver queries having fewer than
ndots dots (default is 1) in them will be attempted using
each component of the search path in turn until a match is
found.
```

实际举例

假如我们的 DNS 配置如下：

```
search kube-system.svc.cluster.local svc.cluster.local cluster.local
options ndots:2
```

当我们 `ping abc.123` (此域名只有一个 dot)，DNS 服务器的日志如下，可以注意到日志中最先尝试的是 `abc.123.kube-system.svc.cluster.local.` ，最后才会尝试我们的域名.

```
[INFO] 10.202.37.232:33386 - 36445 "A IN abc.123.kube-
system.svc.cluster.local. udp 55 false 512" NXDOMAIN qr,aa,rd 148 0.001700129s
[INFO] 10.202.37.232:51389 - 58489 "A IN abc.123.svc.cluster.local. udp 43 false 512" NXDOMAIN qr,aa,rd 136 0.0
01117693s
[INFO] 10.202.37.232:32785 - 4976 "A IN abc.123.cluster.local. udp 39 false 512" NXDOMAIN qr,aa,rd 132 0.001047
215s
[INFO] 10.202.37.232:57827 - 56555 "A IN abc.123. udp 25 false 512" NXDOMAIN qr,rd,ra 100 0.001763186s
```

那我们 `ping abc.123.def` (此域名有两个 dot)，DNS 服务器的日志像下面这样，注意到日志中最优先尝试的是 `abc.123.def.`

```
[INFO] 10.202.37.232:39314 - 794 "A IN abc.123.def. udp 29 false 512" NXDOMAIN qr,rd,ra 104 0.025049846s
[INFO] 10.202.37.232:51736 - 61456 "A IN abc.123.def.kube-
system.svc.cluster.local. udp 59 false 512" NXDOMAIN qr,aa,rd 152 0.001213934s
[INFO] 10.202.37.232:53145 - 26709 "A IN abc.123.def.svc.cluster.local. udp 47 false 512" NXDOMAIN qr,aa,rd 140
```

```
 0.001418143s
[INFO] 10.202.37.232:54444 - 1145 "A IN abc.123.def.cluster.local. udp 43 false 512" NXDOMAIN qr,aa,rd 136 0.00
1009799s
```

希望借这个例子让大家明白两点：

1. 无论 ndots 是多少，search 参数中的后缀都会被以此查找(我们测试时使用了一个不存在的域名，解析工具尝试了全部的可能)
2. ndots 的不妥当设置，可能会给 DNS 服务器造成压力(假如域名是存在的，dns查询会尽快返回，不会继续查找了，会减少服务器压力)

优化讨论

假如现在 ndots 是 2，我们想要查询 `baidu.com`，由于 dot 数目为 1 小于配置中的 2，会首先添加后缀进行查找：

```
[INFO] 10.202.37.232:42911 - 55931 "A IN baidu.com.kube-
system.svc.cluster.local. udp 57 false 512" NXDOMAIN qr,aa,rd 150 0.000116042s
[INFO] 10.202.37.232:53722 - 33218 "A IN baidu.com.svc.cluster.local. udp 45 false 512" NXDOMAIN qr,aa,rd 138 0
.000075077s
[INFO] 10.202.37.232:46487 - 50053 "A IN baidu.com.cluster.local. udp 41 false 512" NXDOMAIN qr,aa,rd 134 0.000
067313s
[INFO] 10.202.37.232:48360 - 51853 "A IN baidu.com. udp 27 false 512" NOERROR qr,aa,rd,ra 77 0.000127309s
```

那么，我们会产生 3 个无用的 DNS 查询记录. 对于DNS服务器来说，仅仅是 `baidu.com` 这个域名，流量就变成了4倍. 假如 n继续增大呢，就像是 `Kubernetes` 中默认给定的5，那我们会产生更多的无效请求，因为不只是 `baidu.com`，就连 `map.baidu.com`， `m.map.baidu.com`，这些域名也要从search域中开始尝试，会对 DNS 服务器造成比较大的压力.

> 我个人建议：
>
> 1. 如果内部服务之间请求十分频繁，也就是我们需要经常访问 `xxx.svc.cluster.local` 这样的域名，那么可以保持 ndots 较大
> 2. 但是内部服务之间请求比较少时，强烈建议调小 ndots，以减少无用流量的产生，减轻 dns 服务器的压力 我个人用的话，改成 2 就好

总结

很抱歉，这篇文章的大部分篇幅都是在说 `nameserver` 是如何解析的， `resolv.conf` 中的内容比较少，主要原因是我前些天一直在看 `iptables`，这次正好有，所以花时间看下，可能有种想要炫技的心理吧.

解决问题的时候，理解后面的参数是比较重要的，我也贴了一些自己的实验，希望能对大家有所帮助吧，至少了解了ndots之后再考虑调优.

The end

上一篇：再见 RPM/DEB/TAR，是时候拥抱下一代全平台安装程序 AppImage 了!
下一篇：5分钟快速了解Docker的底层原理