Kubernetes 服务发现之 coreDNS

原力注入 7月14日

文章转载自公众号 CS实验室，作者 海的澜色

服务发现是 K8s 的一项很重要的功能。K8s 的服务发现有两种方式，一种是将 svc 的 ClusterIP 以环境变量的方式注入到 pod 中；一种就是 DNS，从 1.13 版本开始，coreDNS 就取代了 kube dns 成为了内置的 DNS 服务器。这篇文章就来简单分析一下 coreDNS。

K8s DNS 策略

Kubernetes 中 Pod 的 DNS 策略有四种类型。

1. Default：Pod 继承所在主机上的 DNS 配置；
2. ClusterFirst：K8s 的默认设置；先在 K8s 集群配置的 coreDNS 中查询，查不到的再去继承自主机的上游 nameserver 中查询；
3. ClusterFirstWithHostNet：对于网络配置为 hostNetwork 的 Pod 而言，其 DNS 配置规则与 ClusterFirst 一致；
4. None：忽略 K8s 环境的 DNS 配置，只认 Pod 的 dnsConfig 设置。

下面主要来了解一下 coreDNS 解析域名的过程。

resolv.conf 文件分析

在部署 pod 的时候，如果用的是 K8s 集群的 DNS，那么 kubelet 在起 pause 容器的时候，会将其 DNS 解析配置初始化成集群内的配置。

比如我创建了一个叫 `my-nginx` 的 deployment，其 pod 中的 `resolv.conf` 文件如下：

```
[root@localhost ~]# kubectl exec -it my-nginx-b67c7f44-hsnpv cat /etc/resolv.conf
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

在集群中 pod 之间互相用 svc name 访问的时候，会根据 `resolv.conf` 文件的 DNS 配置来解析域名，下面来分析具体的过程。

域名解析的过程

pod 的 resolv.conf 文件主要有三个部分，分别为 nameserver、search 和 option。而这三个部分可以由 K8s 指定，也可以通过 pod.spec.dnsConfig 字段自定义。

nameserver

`resolv.conf` 文件的第一行 nameserver 指定的是 DNS 服务的 IP，这里就是 coreDNS 的 clusterIP：

```
[root@localhost ~]# kubectl -n kube-system get svc |grep dns
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   32d
```

也就是说所有域名的解析，都要经过 coreDNS 的虚拟 IP `10.96.0.10` 进行解析，不论是 Kubernetes 内部域名还是外部的域名。

search 域

`resolv.conf` 文件的第二行指定的是 DNS search 域。解析域名的时候，将要访问的域名依次带入 search 域，进行 DNS 查询。

比如我要在刚才那个 pod 中访问一个域名为 `your-nginx` 的服务，其进行的 DNS 域名查询的顺序是：

```
your-nginx.default.svc.cluster.local. -> your-nginx.svc.cluster.local. -> your-ng
```

直到查到为止。

options

`resolv.conf` 文件的第三行指定的是其他项，最常见的是 `dnots`。`dnots` 指的是如果查询的域名包含的点“.”小于 5，则先走 search 域，再用绝对域名；如果查询的域名包含点数大于或等于 5，则先用绝对域名，再走 search 域。K8s 中默认的配置是 5。

也就是说，如果我访问的是 `a.b.c.e.f.g`，那么域名查找的顺序如下：

```
a.b.c.e.f.g. -> a.b.c.e.f.g.default.svc.cluster.local. -> a.b.c.e.f.g.svc.cluster
```

如果我访问的是 `a.b.c.e`，那么域名查找的顺序如下：

```
a.b.c.e.default.svc.cluster.local. -> a.b.c.e.svc.cluster.local. -> a.b.c.e.clust
```

pod 之间的通信

在了解完了域名解析的过程后，再来了解一下 pod 之间的通信。

通过 svc 访问

众所周知，在 K8s 中，Pod 之间通过 svc 访问的时候，会经过 DNS 域名解析，再拿到 ip 通信。而 K8s 的域名全称为 "<service-name>.<namespace>.svc.cluster.local"，而我们通常只需将 svc name 当成域名就能访问到 pod，这一点通过上面的域名解析过程并不难理解。

我们来看个例子。有两个 deployment，一个叫 busybox，在 default 这个 namespace 下；一个叫 your-nginx，在 hdls 这个 namespace 下，svc 同名。我们在 busybox 中尝试访问 your-nginx。

```
[root@localhost ~]# kubectl get po
NAME                          READY   STATUS    RESTARTS   AGE
busybox-5bbb5d7ff7-dh68j      1/1     Running   0          8m35s
[root@localhost ~]#
[root@localhost ~]# kubectl exec -it busybox-5bbb5d7ff7-dh68j sh
/ # wget your-nginx
wget: bad address 'your-nginx'
/ #
/ # wget your-nginx.hdls
Connecting to your-nginx.hdls (10.100.3.148:80)
saving to 'index.html'
index.html           100% |********************************************************|
'index.html' saved
/ #
[root@localhost ~]# kubectl -n hdls get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
your-nginx   ClusterIP   10.100.3.148   <none>        80/TCP    14m
```

可以看到，当直接用 your-nginx 去访问的时候，提示 bad address，说明域名错了，因为在不同的 namespace 下，所有的 search 域都找过了还是找不到；当用 your-nginx.hdls 去访问的时候，会解析到 10.100.3.148 这个 IP，而这个 IP 正是 your-nginx 的 ClusterIP。

所以，在不同的 namespace 下的 pod 通过 svc 访问的时候，需要在 svc name 后面加上 .<namespace>。

pod 的 hostname 与 subdomain

在 K8s 中，如果不指定 pod 的 hostname，其默认为 pod.metadata.name，通过 spec.hostname 字段可以自定义；另外还可以给 pod 设置 subdomain，通过

spec.subdomain 字段。比如下面这个例子：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    name: nginx
spec:
  hostname: domain-test
  subdomain: subdomain-test
  containers:
  - image: nginx
    name: nginx
---
apiVersion: v1
kind: Service
metadata:
  name: subdomain-test
spec:
  selector:
    name: nginx
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
```

可以查看这个 pod 的 hostname 和 hosts 文件：

```
[root@localhost ~]# kubectl get po -owide
NAME                          READY   STATUS    RESTARTS   AGE     IP
busybox-5bbb5d7ff7-dh68j      1/1     Running   0          112m    10.244.1.246
nginx                         1/1     Running   0          2m      10.244.1.253
[root@localhost ~]# kubectl exec -it nginx bash
root@domain-test:/# cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1    localhost
::1    localhost ip6-localhost ip6-loopback
fe00::0    ip6-localnet
fe00::0    ip6-mcastprefix
fe00::1    ip6-allnodes
fe00::2    ip6-allrouters
10.244.1.253    domain-test.subdomain-test.default.svc.cluster.local    domain-te
root@domain-test:/#
```

在 busybox 容器中访问这个 pod：

```
[root@localhost ~]# kubectl exec -it busybox-5bbb5d7ff7-dh68j sh
/ # wget domain-test.subdomain-test
Connecting to domain-test.subdomain-test (10.244.1.253:80)
```

```
saving to 'index.html'
index.html           100% |*************************************************|
'index.html' saved
/ #
/ # wget subdomain-test
Connecting to subdomain-test (10.108.213.70:80)
wget: can't open 'index.html': File exists
/ #
```

可以看到，当访问 `domain-test.subdomain-test` 解析出来的是 `10.244.1.253`，这个是 nginx 的 pod ip，而不是 clusterIP；而访问 `subdomain-test` 时，解析出来的是 `10.108.213.70`，这是 clusterIP，属于正常的 svc name 途径。

coreDNS Corefile 文件

CoreDNS 实现了应用的插件化，用户可以选择所需的插件编译到可执行文件中；CoreDNS 的配置文件是 Corefile 形式的，下面是个 coreDNS 的 configMap 例子。

```
[root@localhost ~]# kubectl -n kube-system get cm coredns -oyaml
apiVersion: v1
data:
  Corefile: |
    .:53 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           upstream
           fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf
        cache 30
        loop
        reload
        loadbalance
    }

kind: ConfigMap
metadata:
  creationTimestamp: "2019-06-10T03:19:01Z"
  name: coredns
  namespace: kube-system
  resourceVersion: "3380134"
  selfLink: /api/v1/namespaces/kube-system/configmaps/coredns
  uid: 7e845ca2-8b2e-11e9-b4eb-005056b40224
```

Corefile 文件分析

第一部分：

```
kubernetes cluster.local in-addr.arpa ip6.arpa {
   pods insecure
   upstream
   fallthrough in-addr.arpa ip6.arpa
}
```

指明 cluster.local 后缀的域名，都是 kubernetes 内部域名，coredns 会监听 service 的变化来维护域名关系，所以cluster.local 相关域名都在这里解析。

第二部分：

```
proxy . /etc/resolv.conf
```

proxy 指 coredns 中没有找到记录，则去 /etc/resolv.conf 中的 nameserver 请求解析，而 coredns 容器中的 /etc/resolv.conf 是继承自宿主机的。实际效果是如果不是 k8s 内部域名，就会去默认的 dns 服务器请求解析，并返回给 coredns 的请求者。

第三部分：

`prometheus`： CoreDNS 的监控地址为：http://localhost:9153/metrics ，满足 Prometheus 的格式。

`cache`： 允许缓存

`loop`： 如果找到循环，则检测简单的转发循环并停止 CoreDNS 进程。

`reload`： 允许 Corefile 的配置自动更新。在更改 ConfigMap 后两分钟，修改生效

`loadbalance`： 这是一个循环 DNS 负载均衡器，可以在答案中随机化 A，AAAA 和 MX 记录的顺序。

指定 hosts

有的时候，某个域名的服务在集群外，而我希望在集群内访问到，我们可以在 corefile 中指定 hosts 的方法实现。具体方式是将域名及对应的 ip 以 hosts 插件的方式加入到 corefile 中，如下：

```
hosts {
    10.244.1.245 other-company.com
    fallthrough
}
```

其中， `10.244.1.245` 是 `your-nginx` 的 pod ip。然后再在上面的 busybox pod 中访问 `other-company.com` 这个服务，情况如下：

```
[root@localhost ~]# kubectl exec -it busybox-5bbb5d7ff7-dh68j sh
/ # wget other-company.com
Connecting to other-company.com (10.244.1.245:80)
saving to 'index.html'
index.html           100% |********************************************************|
'index.html' saved
/ #
```

阅读原文