K8s-DNS关键问题

liangdong@smzdm.com

[背景 1](#_Toc533670185)

[实战 2](#_Toc533670186)

[创建测试service 2](#_Toc533670187)

[Service域名解析 3](#_Toc533670188)

[Service的headless模式 5](#_Toc533670189)

[Resolve.conf解释 5](#_Toc533670190)

[配置out-cluster的domain 6](#_Toc533670191)

[Corefile配置简介 7](#_Toc533670192)

[配置in-cluster的domain 9](#_Toc533670193)

[其他 11](#_Toc533670194)

[参考资料 11](#_Toc533670195)

# 背景

当前服务间调用是按域名实现的，每个节点基于/etc/hosts解析到云厂商的load balanace，进而转发到目标服务的具体节点，即没有内网DNS存在。

上k8s之后，肯定要面临部分服务在k8s集群内in-clsuter，部分服务在集群外out-cluster，面临几种情况：

1，out-cluster访问in-cluter：保持/etc/hosts走load balance代理到k8s ingress的nodePort，经过ingress转发到具体backend POD。

2，in-cluster访问in-cluster：显然k8s的service domain和原本服务的domain是不一样的，前者长这样my-nginx.default.svc.cluster.local，后者长这样nginx.company.com，所以如何保持原有domain方式访问是一个问题。

3，in-cluster访问out-cluster：需要让in-cluster的container拥有和out-cluster一样的/etc/hosts配置，而且还不能覆盖POD自带的hosts配置，才能保持按domain走到云上的load balance进入Out-cluster服务。

可见，问题就出在in-cluster上。

一旦业务搬到k8s上，

1，in-cluster到out-cluter的情况：如果继续采用传统配置/etc/hosts的方式将很难用，虽然可以挂ConfigMap到container的/etc/hosts，configmap更新可以实时反馈到container中，但这种做法也把container自身的hosts文件覆盖掉了，显然挂载hosts不是可行方案。

2，in-cluster到in-cluster的情况：目标service domain是k8s风格的，让业务改调用域名显然不合适，所以得让原本nginx.company.com这样的域名可以解析到目标service上。因为目标service以及POD都在k8s集群内动态分配，所以coredns就最适合做这个解析工作，它动态维护了service的IP地址，我们如果可以让coredns把nginx.company.com做cname到my-nginx.default.svc.cluster.local，那么coredn就可以最终返回my-nginx.default.svc.cluster.local这个service的IP，就可以用nginx.company.com域名访问到目标service了。

既然决定用Coredns作in-cluster的domain解析，那么也可以把out-cluster的domain直接配到coredns里，这样无论是in-cluster->in-cluster，还是in-cluster->out-cluster，都可以找coredns解析到IP，这个IP可能是外部的loadbalance，也可能是内部的service ip，做到了业务透明。

# 实战

一共要做2个事情。

1. 把out-cluster的domain，配置到coredns直接解析为load balance IP，这样POD可以直接找coredns解析到外部IP。
2. 把in-cluster的domain，配置到coredns做cname到service domain，这样POD可以直接找coredns解析到service IP。

## 创建测试service

先来创建一个in-cluster的service：

apiVersion: apps/v1

kind: Deployment

metadata:

 name: my-nginx

spec:

 selector:

 matchLabels:

 app: my-nginx

 replicas: 2

 template:

 metadata:

 labels:

 app: my-nginx

 spec:

 dnsPolicy: ClusterFirst

 containers:

 - name: my-nginx

 image: nginx

 ports:

 - containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

 name: my-nginx

 labels:

 app: my-nginx

spec:

 type: ClusterIP

 clusterIP: None

 ports:

 - port: 80

 protocol: TCP

 selector:

app: my-nginx

注意，podTemplate里有一个dnsPolicy配置，其实POD默认就是clusterFirst，意思就是让container里的/etc/resolv.conf配置指向Coredns，即所有的dns请求发给coredns，这是k8s的默认行为。

## Service域名解析

可以kubectl exec -it my-nginx-7448cc6b89-kqxx2 bash登录到my-nginx的一个POD里，看一下resolv.conf：



nameserver就是coredns的service Cluter IP，可以查看一下：



上述service叫做my-nginx，coredns会创建对应的k8s domain，叫什么呢？

my-nginx.default.svc.cluster.local，就是 服务名.命名空间.svc.cluster.local，其中svc就是资源类型是service的意思。

解析域名可以得到什么呢？先apt-get update && apt-get install dnsutils安装一下网络工具。



竟然可以解析service的名字，其实这就与resolv.conf有关了。

如果你请求的domain中.少于5个（my-nginx一个.也没有），那么就会逐个把search中的后缀拼到my-nginx上，然后去nameserver的地址请求解析IP。所以最终其实请求解析的域名是my-nginx.default.svc.cluster.local，如果拼接后都请求不到A记录，最后还会直接请求解析my-nginx。

如果domain包含的.大于等于5个，那么会先请求原始my-nginx这个domain，如果没有得到IP则会拼接server后缀再发起请求。

你也可以直接请求完整域名：my-nginx.default.svc.cluster.local：



同时，我们要注意到，k8s宿主机的resolv.conf是物理网络分配的，与container内是两码事：



所以，在k8s宿主机上，虽然可以与容器IP互通，但是解析k8s内的域名是不可能的，因为宿主机指定的nameserver是物理网络的：



当然可以指定去coredns的clusterip进行解析：



此时，因为宿主机的resolv.conf没有server后缀配置，所以需要指定完整的k8s domain，coredns那边保存只有完整域名的IP解析。

简称my-nginx只有在resolv.conf中有server后缀配置的情况下由请求方补全，再请求coredns才有效。

## Service的headless模式

上述service指定了clusterIp为None，这叫headless service。

coredns解析这种service的时候，返回的是POD的IP列表，而不是service的cluster ip。

如果我们删掉service的clusterIP: None配置，那么coredns就只会返回cluster ip了，这里就不演示了（需要先删掉现在的service，再重新建）。

Headless service的好处，就是可以coredns可以直接返回pod的ip列表，不需要经过cluster ip的中转，性能更好，另外coredns每次返回的ip顺序不同，即支持round robin，后续会看到coredns的相关配置。

## Resolve.conf解释

/etc/resolv.conf大家可以学习一下，这是linux基础：<https://www.ichenfu.com/2018/10/09/resolv-conf-desc/>

实际上，当我们解析少于5个.的domain时候，会经历多次server拼接，每次拼接后会去nameserver问一下：



所以请求量还是挺大的，网上有关于如何解决option ndots性能浪费问题的方法，思路是把ndots改成1，这样就总是首先按原始查询发起，之后再追加后缀：

<https://pracucci.com/kubernetes-dns-resolution-ndots-options-and-why-it-may-affect-application-performances.html>

## 配置out-cluster的domain

如果我们有一个some-api.company.com的服务在k8s集群外，然后in-cluster想按域名访问它，该如何实现呢？

公司目前没有内网dns，都是/etc/hosts解析，容器的hosts不方便配置。

解决思路就是在coredns里配置hosts解析，作为一个dns server，它支持这个功能。

首先，把coredns的configmap拖下来，coredns的配置文件是通过configmap配置的：

kubectl get configmap coredns -n kube-system -o yaml > coredns-configmap.yaml

保存到文件后，我们增加这么一段：



然后apply提交configmap：

kubectl apply -f coredns-configmap.yaml

k8s会重新mount configmap到各个容器中去，因为coredns的配置中有一个reload选项，意思就是coredns会定时重新加载corefile配置文件，所以稍微等一会解析就生效了：



解析得到了IP地址，这个IP是out-cluster的云load balance地址。

## Corefile配置简介

kubernetes cluster.local in-addr.arpa ip6.arpa {

 pods insecure

 upstream

 fallthrough in-addr.arpa ip6.arpa

 }

这一段配置，就是说cluster.local后缀的domain，都是kubernetes内部域名，coredns会监听service的变化维护域名关系，所以cluster.local相关域名都在这里解析。

proxy . /etc/resolv.conf

这个proxy意思是如果在coredns中没有找到A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，所以实际效果就是如果不是k8s内部域名，就会去宿主机的dns服务器请求解析，并返回给coredns的请求者。



这是coredns deployment的截图，可见coredns的dnsPolicy是Default而不是clusterFirst，此时coredns container的resolv.conf将直接继承自宿主机。

Loop没具体看。

Reload是定期重新加载coredns配置的意思，我们更新configmap后k8s会立即重新挂载，配合coredns定期check配置文件的变化，就可以实现热加载了。

Loadbalance是随机打乱A记录的顺序，因为headlesss service会返回POD列表的IP，如果每次都同样的顺序，客户端可能就都访问第一个IP了，这就是DNS负载均衡的功能。

 hosts {

 123.59.59.33 some-api.company.com

 fallthrough

 }

这段就是指定hosts文件的作用，如果请求的是some-api.company.com的域名，那么直接返回IP。如果没有hosts命中的话，则继续执行其他的解析插件，所以hosts其实是一种插件： <https://coredns.io/plugins/hosts/> 。

## 配置in-cluster的domain

之前说过，如果希望保持按原有域名nginx.company.com访问my-nginx这个service，显然就要提供nginx.company.com的解析能力。

有一种思路就是，请求coredns完成解析，这是POD的默认dnspolicy。

可以给coredns配置从nginx.company.com CNAME 到my-nginx.default.svc.cluster.local，这样就相当于请求的my-nginx的A记录。

所以，我们可以再编辑configmap：



新增一个配置文件company.db，相当于配置了company.com的权威DNS，目前提供了一个nginx.company.com到my-nginx.default.svc.cluster.local的CNAME。

File插件如果遇到company.com的域名后缀，就会用company.db作解析。

经过解析后得到了cname my-nginx.default.svc.cluster.local，并不是最终A记录，此时需要让coredns做递归解析my-nginx.default.svc.cluster.local，此时会命中kubernetes插件得到POD IP列表：

。

操作系统的DNS客户端，是不会自己递归解析cname的，必须由递归DNS sever去完成，我们生活中都是用运营商DNS，它们都是递归DNS，你问他domain，它直接给你A记录。

而自上而下的DNS拓扑，通过NS记录逐级向下查找：



最终走到权威DNS，得到A记录。

local dns就是递归DNS，它和权威DNS的关系：



我们这里coredns是个k8s内部的local dns，既能基于宿主机的ns做递归解析外部域名，也能本地解析内部domain或者hosts或者权威记录。

DNS配置的一些说明：

<https://klionsec.github.io/2017/12/11/Dns-tips/>

<https://coredns.io/2017/05/08/custom-dns-entries-for-kubernetes/>

## 其他

既然coredns承载了集群所有的dns解析工作，那么一定要给它足够多的实例和资源，可以修改coredns的deployment的replicas，resource limit/request。

Coredns默认是配置定时加载的，如果配置有错误是不影响服务的。 但是一旦操作deployment触发重新部署，就需要确保配置文件没有问题，否则coredns拉不起来，集群通讯就挂了。这和传统dns操作风险是一样的，例如百度前一阵的1小时故障。

另外，上述in-cluter直接将nginx.company.com解析到my-nginx甚至headless的my-nginx在性能上存在优势，避免了经过ingress，避免了流量先离开集群走到云load balance再走nodePort回到k8s集群的浪费，但也导致无法方便的监控nginx日志，incluter -> incluter成为了点对点通讯，这是需要考虑的问题。 也许可以让nginx.company.com CNAME到ingress上，或者干脆hosts到集群外的loadbalance，对运维方式改变更小，以牺牲性能为代价。

# 参考资料

Corefile配置：<https://coredns.io/2017/07/23/corefile-explained/>

DNS配置概念：<https://ephen.me/2016/dns-rr/>

Coredns file插件：<https://github.com/coredns/coredns/tree/master/plugin/file>

Coredns host插件：

<https://coredns.io/plugins/hosts/>

coredns loadbalance插件：

<https://github.com/coredns/coredns/tree/master/plugin/loadbalance>

coredns 定制dns entry：

<https://coredns.io/2017/05/08/custom-dns-entries-for-kubernetes/>