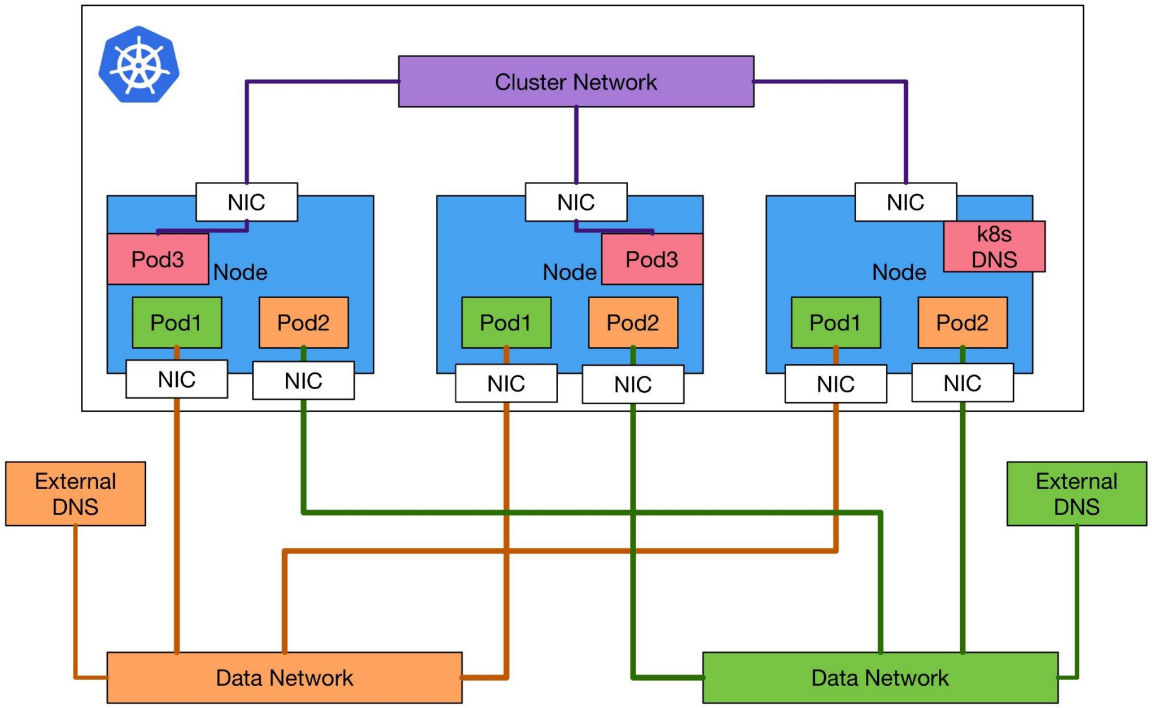
k8s External DNS 云原生的方式与容器平台对接

DNS 是 Kubernetes 的核心功能之一，Kubernetes 通过 kube-dns 或 CoreDNS 作为集群的必备扩展来提供命名服务，通过 DNS 扩展，每一个 Service 都会产生一个独一无二的 FQDN（Fully Qualified Domain Name）名称。



一、自定义 DNS

一般的使用场景下，我们的 Kubernetes 集群的使用方式就像图中紫色/粉红色（Pod3）区域一样，所有的 Pod 如果有任何要存取 DNS 的需求，都会透过集群内的的 k8s DNS 来处理对应的请求与回复。

然而在 NFV 的使用场景下，网络变成一个很重要的区域，整体的性能都取决于该应用的设计与集群的网络架构设计。这部分应用通常都会追求高输出或是低延迟，为了得到更好的性能，需要避免这些流量跟其他无关的流量使用相同的网络线路进行传输。

在这种情况下，通常就会把整个集群的网络设计成两种架构，分别是 Control Network 和 Data Network 这两个不同用途的网络架构。在 Kubernetes 中，Control Network 就类似于图中的 Cluster Network，负责整个集群之间的沟通。图中绿色/橘色（Pod1，Pod2）这两个区域就是所谓的 Data Network，其网卡本身也被独立出来，不会与本来的 Kubernetes 集群发生冲突，它们之间的流量通过独立的网络进行传输。

存在于独立出来的网络架构中的这些特殊的 Pod 基本上没法跟 Kubernetes 集群内的 DNS 互连，而且这些应用还有可能在外部有自己的 DNS Server，所以在这种场景下，我们希望这些应用（Pod1/Pod2）能够使用自定义的 DNS Server。

为了让用户更容易控制 Pod 中的 DNS 设置，Kubernetes v1.9 引入了一项新的 Alpha 特性（在 v1.10 中处于 Beta 阶段）。该特性在 v1.10 中被默认启用，在 v1.9 中如果想要启用此功能，集群管理员需要在 apiserver 和

kubelet 上启用 CustomPodDNS 特性，例如：“--feature-gates=CustomPodDNS=true,...”。启用了该特性之后，用户可以将 Pod 的 dnsPolicy 字段设置为 "None"，并且可以在 Pod.Spec 中添加新的字段 dnsConfig。

其中 dnsConfig 用来自定义 DNS 参数，而 dnsPolicy 用来给 Pod 选取预设的 DNS。接下来就看看可以通过哪些手段自定义 DNS。

二、dnsConfig

dnsConfig 可以让操作者延伸到 Pod 内部关于 DNS 的配置，这边需要特别注意的是，我使用的字眼是 延伸 而不是 配置，这是因为通过下一节的 dnsPolicy，每个 Pod 都会有一组预设的 DNS 配置。通过 dnsConfig 我们可以继续往上叠加相关的 DNS 参数到 Pod 之中。目前总共支持三个参数，分别是：

1.nameservers

2.searches

3.options

这三个参数对应的就是大家熟悉的 /etc/resolv.conf 里面的三个参数，这里就不针对 DNS 进行详细解释了，不熟悉的朋友可以自行去 Google 学一下这些参数的意思。

在 Kubernetes 里面，这三个参数都包含在 dnsConfig 配置项中，而 dnsConfig 包含在 PodSpec 配置项中，因为 Pod 内所有的容器都共享相同的 Network Namespace，所以网络方面的配置都会共享。

这边提供一个简单的 yaml 示例：

```
apiVersion: v1
kind: Pod
metadata:
name: ubuntu-setting
namespace: default
spec:
containers:
- image: hwchiu/netutils
command:
- sleep
- "360000"
imagePullPolicy: IfNotPresent
name: ubuntu
restartPolicy: Always
dnsConfig:
nameservers:
- 1.2.3.4
searches:
- ns1.svc.cluster.local
- my.dns.search.suffix
options:
- name: ndots
value: "2"
```

```
- name: edns0
```

通过上述 yaml 创建 Pod 之后，通过下面的命令可以观察到容器中 DNS 配置文件中会出现额外的配置。

```
$ kubectl exec ubuntu-setting cat /etc/resolv.conf
nameserver 10.254.0.2
nameserver 1.2.3.4
search default.svc.cluster.local svc.cluster.local cluster.local ns1.svc.cluster.local my.dns.search.suffix
options ndots:2 edns0
```

可以看到 nameserver 多了一个 1.2.3.4，而 search 则多了 ns1.svc.cluster.local my.dns.search.suffix 这两个自定义的值，最后 options 则增加了我们示例中指定的 ndots:2 edns0。

dnsConfig 非常简单直观，如果你需要自定义 DNS 参数，就可以通过这个字段来指定。

三、dnsPolicy

前面提过，dnsConfig 提供的是延伸 Pod 内预设的 DNS 配置，而 dnsPolicy 就是决定 Pod 内预设的 DNS 配置有哪些。

目前总共有四个类型可以选择：

1.None
2.Default
3.ClusterFirst
4.ClusterFirstHostNet

接下来针对这四个类型分别介绍。

None 表示会清除 Pod 预设的 DNS 配置，当 dnsPolicy 设置成这个值之后，Kubernetes 不会为 Pod 预先载入任何自身逻辑判断得到的 DNS 配置。因此若要将 dnsPolicy 的值设为 None，为了避免 Pod 里面没有配置任何 DNS，最好再添加 dnsConfig 来描述自定义的 DNS 参数。

使用下面的示例来进行测试：

```
apiVersion: v1
kind: Pod
metadata:
name: ubuntu-none
namespace: default
spec:
containers:
- image: hwchiu/netutils
command:
- sleep
- "360000"
imagePullPolicy: IfNotPresent
name: ubuntu
restartPolicy: Always
dnsPolicy: None
dnsConfig:
```

```
nameservers:
- 1.2.3.4
searches:
- ns1.svc.cluster.local
- my.dns.search.suffix
options:
- name: ndots
value: "2"
- name: edns0
```

通过上述 yaml 创建 Pod 之后，通过下面的命令可以观察容器中的 DNS 配置文件，可以观察到跟之前的 dnsConfig 的结果有一点差异，这里只有我们在 yaml 里配置的那些参数，而没有加入集群预设的 DNS 配置。

```
$ kubectl exec ubuntu-none cat /etc/resolv.conf
nameserver 1.2.3.4
search ns1.svc.cluster.local my.dns.search.suffix
options ndots:2 edns0
```

Default 表示 Pod 里面的 DNS 配置继承了宿主机上的 DNS 配置。简单来说，就是该 Pod 的 DNS 配置会跟宿主机完全一致。

使用下面的示例来进行测试：

```
apiVersion: v1
kind: Pod
metadata:
name: ubuntu-default
namespace: default
spec:
containers:
- image: hwchiu/netutils
command:
- sleep
- "360000"
imagePullPolicy: IfNotPresent
name: ubuntu
restartPolicy: Always
dnsPolicy: Default
```

首先，我们先观察 Node 上面的 DNS 配置：

```
$ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 10.0.2.3
```

可以观察到，Node 上面的 DNS 配置得很简单，只有单纯的 10.0.2.3。

接下来我们观察该 Pod 内的 DNS 配置：

```
$ kubectl exec ubuntu-default cat /etc/resolv.conf
nameserver 10.0.2.3
```

可以看到这两个的 DNS 配置完全一致，该 Pod 内的 DNS 配置已经直接继承 Node 上面的配置了。

ClusterFirst

相对于上述的 Default，ClusterFirst 是完全相反的操作，它会预先把 kube-dns（或 CoreDNS）的信息当作预设参数写入到该 Pod 内的 DNS 配置。

```
apiVersion: v1
kind: Pod
metadata:
name: ubuntu-clusterfirst
namespace: default
spec:
containers:
- image: hwchiu/netutils
command:
- sleep
- "360000"
imagePullPolicy: IfNotPresent
name: ubuntu
restartPolicy: Always
dnsPolicy: ClusterFirst
```

通过上述 yaml 创建 Pod 之后，通过下面的命令观察容器中的 DNS 配置文件：

```
$ kubectl exec ubuntu-clusterfirst cat /etc/resolv.conf
nameserver 10.254.0.2
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

可以看到这里使用的是 k8s DNS 的设置。

此外，ClusterFirst 还有一个冲突，如果你的 Pod 设置了 HostNetwork=true，则 ClusterFirst 就会被强制转换成 Default。

HostNetwork

使用下面的示例来进行测试：

```
apiVersion: v1
kind: Pod
metadata:
name: ubuntu-hostnetwork-policy-default
namespace: default
spec:
containers:
- image: hwchiu/netutils
command:
- sleep
- "360000"
imagePullPolicy: IfNotPresent
name: ubuntu
```

```
hostNetwork: true
restartPolicy: Always
dnsPolicy: ClusterFirst
```

通过上述 yaml 创建 Pod 之后，通过下面的命令观察容器中的 DNS 配置文件：

```
$ kubectl exec ubuntu-hostnetwork-policy-default cat /etc/resolv.conf
nameserver 10.0.2.3
```

通过上述 yaml 创建 Pod 之后，通过下面的命令观察容器中的 DNS 配置文件：

可以观察到，Pod 里面的 DNS 配置直接继承了宿主机上的 DNS 配置。

这边稍微来解释一下这个设计上的原理以及流程：

因为设置了 HostNetwork=true，会让该 Pod 与该节点共用相同的网路空间(网卡/路由等功能)。

预设的 k8s DNS 是使用 ClusterIP 的 kubernetes serivce. 这种情况下，只有属于 Cluster 内的 Pod 可以获取该 ClusterIP。

所以设置了 HostNetwork=true 的 Pod 就没有办法获取该 ClusterIP。

于是预设就会将对应的 DNS 配置改回 Default 的形式，从节点继承其 DNS 配置信息。

这种情况下，就会有人想要问，如果我刻意想要这样设置不行吗？

原先的设计中，是没有办法刻意处理的，原因是当 Pod yaml 配置文件被发送出去后，在发现没有设定 dnsPolicy 的情况下，会自动帮你把该 dnsPolicy 补上 ClusterFirst 的数值。

然后最后面的程序处理逻辑中，其实并没有办法分别下列两种情况：

HostNetwork：我希望走 Host DNS

HostNetwork & dnsPolicy=ClusterFirst：我希望走 ClusterIP DNS

上述两种情况对于后端的程序来看都长得一样，完全没有办法分辨，我们可以直接从 Kubernetes 源码 来阅读一下其运作流程

```
func getPodDNSType(pod *v1.Pod) (podDNSType, error) {
dnsPolicy := pod.Spec.DNSPolicy
switch dnsPolicy {
case v1.DNSNone:
if utilfeature.DefaultFeatureGate.Enabled(features.CustomPodDNS) {
return podDNSNone, nil
}
// This should not happen as kube-apiserver should have rejected
// setting dnsPolicy to DNSNone when feature gate is disabled.
return podDNSCluster, fmt.Errorf(fmt.Sprintf("invalid DNSPolicy=%v: custom pod DNS is disabled", dnsPolicy))
case v1.DNSClusterFirstWithHostNet:
return podDNSCluster, nil
case v1.DNSClusterFirst:
if !kubecontainer.IsHostNetworkPod(pod) {
return podDNSCluster, nil
```

```
}
// Fallback to DNSDefault for pod on hostnetowrk.
fallthrough
case v1.DNSDefault:
return podDNSHost, nil
}
// This should not happen as kube-apiserver should have rejected
// invalid dnsPolicy.
return podDNSCluster, fmt.Errorf(fmt.Sprintf("invalid
DNSPolicy=%v", dnsPolicy))
}
```

使用下面的示例来进行测试：

```
apiVersion: v1
kind: Pod
metadata:
name: ubuntu-hostnetwork-policy
namespace: default
spec:
containers:
- image: hwchiu/netutils
command:
- sleep
- "360000"
imagePullPolicy: IfNotPresent
name: ubuntu
hostNetwork: true
restartPolicy: Always
dnsPolicy: ClusterFirstWithHostNet
```

通过上述 yaml 创建 Pod 之后，通过下面的命令观察该 Pod 的状态：

```
$ kubectl exec ubuntu-hostnetwork-policy cat /etc/resolv.conf
nameserver 10.254.0.2
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

进入 k8s 管理平台，将下载的 yk-ingress.tar 镜像文件文件上传至 k8s 管理平台，并加载 yk-ingress 镜像。

启动并进入镜像

配置 yk-ingress.yaml 文件

```
- name: ingress-controller yk-adc
  hosts: yk-adc
  connection: ssh
     - name: config pool
       yk-adc_pool:
           state: present
           name: pool_k8s-test_80
           lb_method: round-robin
           monitors:
             - monitor_http
           provider:
             server: "{{ inventory_hostname }}"
             server_port: 8443
             user: yk-ingress
             password: yk-ingress
           validate_certs: no
       delegate_to: localhost

     - name: config pool_member
       yk-adc_pool_member:
          state: present
          pool: pool_k8s-test_80
          aggregate:
            - pod: k8s-test_01::21
              port: 8090
              address:1.1.1.1
            - pod: k8s-test_01::22
              port: 8090
              address:1.1.1.1
          provider:
            server: "{{ inventory_hostname }}"
            server_port: 8443
            user: yk-ingress
            password: yk-ingress
          validate_certs: no
       delegate_to: localhost

     - name: config virtual_server
       yk-adc_virtual_server:
           state: present
           name: vs_k8s-test_80
```

通过以上方式可以看到设备已实现与 k8s External DNS 对接，实现容器平台与外部 DNS 的自动化配置。