CoreDNS 使用与架构分析

2018-04-22 · 云原生

概述

CoreDNS 是新晋的 CNCF 孵化项目（社区也计划将其作为 Kubernetes 默认的 DNS 方案）。CoreDNS 的目标是成为 cloud-native 环境下的 DNS 服务器和服务发现解决方案，即：

> Our goal is to make CoreDNS the cloud-native DNS server and service discovery solution.

它有以下几个特性：

- **插件化（Plugins）**

  基于 Caddy 服务器框架，CoreDNS 实现了一个**插件链**的架构，将大量应用端的逻辑抽象成 plugin（下文将混用 plugin 和 插件 这两个词汇）的形式（如 Kubernetes 的 DNS 服务发现，Prometheus 监控等）暴露给使用者。CoreDNS 以预配置的方式将不同的 plugin 串成一条链，按序执行 plugin 的逻辑。**从编译层面，用户选择所需的 plugin 编译到最终的可执行文件中，使得运行效率更高**。CoreDNS 采用 Go 编写，所以从具体代码层面来看，每个 plugin 其实都是实现了其定义的 interface 的组件而已。第三方只要按照 CoreDNS Plugin API 去编写自定义插件，就可以很方便地集成于 CoreDNS；

- **配置简单化**

  引入表达力更强的 DSL，即 Corefile 形式的配置文件（也是基于 Caddy 框架开发）；

CONTENTS

- 概述
- 编译与运行
- Corefile 介绍
- 插件的工作模式
- CoreDNS 如何处理 l 请求
- 在 Kuberntes 中使用 CoreDNS
- 参考文档

- **一体化的解决方案**

  区别于 kube-dns，CoreDNS 编译出来就是一个单独的二进制可执行文件，内置了 cache，backend storage ，health check 等功能，无需第三方组件来辅助实现其他功能，从而使得部署更方便，内存管理更为安全；

其实从功能角度来看，CoreDNS 更像是一个通用 DNS 方案（类似于 BIND），然后通过插件模式来极大地扩展自身功能，从而可以适用于不同的场景（比如 Kubernetes）。正如官方博客所说：

> CoreDNS is powered by plugins.

编译与运行

推荐使用 Docker 方式来编译代码（Go > 1.9）：

Code

```
$ docker run --rm -i -t -v $PWD:/go/src/github.com/coredns/coredns \
      -w /go/src/github.com/coredns/coredns golang:1.10 make
```

编译最后将生成一个可执行文件 `coredns`，因为 CoreDNS 是一个通用 DNS 服务器，所以无需依赖特定的场景，可以直接执行：

Code

```
$ ./coredns -dns.port=1053
```

用 `dig` 工具进行测试（查找 A 记录）：

Code

```
$ dig @localhost -p 1053 A whoami.example.org
```

CoreDNS 的 code base 并未对所有的依赖做 vendor 化管理，所以编译时期仍需要联网下载某些依赖组件。之所以采用这种方式，更多是因为其外部插件总是会引入不同的依赖组件，为了能让插件也能顺利编译，所以需要采用 `go get` 的方式获取其他依赖组件。关于这个问题，可参考讨论。

Corefile 介绍

Corefile 是 CoreDNS 的配置文件（源于 Caddy 框架的配置文件 Caddyfile），它定义了：

- **server 以什么协议监听在哪个端口（可以同时定义多个 server 监听不同端口）**
- **server 负责哪个 zone 的权威（authoritative）DNS 解析**
- **server 将加载哪些插件**

常见地，一个典型的 Corefile 格式如下所示：

Code

```
1  ZONE:[PORT] {
2          [PLUGIN] ...
3  }
```

- `ZONE`：定义 server 负责的 zone，`PORT` 是可选项，默认为 53；
- `PLUGIN`：定义 server 所要加载的 plugin。每个 plugin 可以有多个参数；

比如：

Code

```
1  . {
2      chaos CoreDNS-001
3  }
```

上述配置文件表达的是：server 负责根域 `.` 的解析，其中 plugin 是 `chaos` 且没有参数。

- **定义 server**

  一个最简单的配置文件可以为：

Code

```
.{}
```

即 server 监听 53 端口并不使用插件。**如果此时在定义其他 server，要保证监听端口不冲突；如果是在原来 server 增加 zone，则要保证 zone 之间不冲突**，如：

Code

```
.     {}
.:54 {}
```

另一个 server 运行于 54 端口并负责根域 `.` 的解析。

又如：

Code

```
example.org {
    whoami
}
org {
    whoami
}
```

同一个 server 但是负责不同 zone 的解析，有不同插件链。

- **定义 Reverse Zone**

  跟其他 DNS 服务器类似，Corefile 也可以定义 Reverse Zone：

Code

```
0.0.10.in-addr.arpa {
    whoami
}
```

或者简化版本：

Code

```
10.0.0.0/24 {
    whoami
}
```

- **使用不同的通信协议**

  CoreDNS 除了支持 DNS 协议，也支持 TLS 和 gRPC，即 DNS-over-TLS 和 DNS-over-gRPC 模式：

  Code

  ```
  tls://example.org:1443 {
      #...
  }
  ```

插件的工作模式

当 CoreDNS 启动后，它将根据配置文件启动不同 server ，每台 server 都拥有自己的插件链。当有 DNS 请求时，它将依次经历如下 3 步逻辑：

1. 如果有当前请求的 server 有多个 zone，将采用贪心原则选择最匹配的 zone；
2. 一旦找到匹配的 server，按照 `plugin.cfg` 定义的顺序执行插件链上的插件；
3. 每个插件将判断当前请求是否应该处理，将有以下几种**可能**：
   - **请求被当前插件处理**

     插件将生成对应的响应并回给客户端，此时请求结束，下一个插件将不会被调用，如 `whoami` 插件；
   - **请求不被当前插件处理**

     直接调用下一个插件。如果最后一个插件执行错误，服务器返回 `SERVFAIL` 响应；
   - **请求被当前插件以 Fallthrough 形式处理**

如果请求在该插件处理过程中有可能将跳转至下一个插件，该过程称为 fallthrough，并以关键字 `fallthrough` 来决定是否允许此项操作，例如 `host` 插件，当查询域名未位于 `/etc/hosts`，则调用下一个插件；

- **请求在处理过程被携带 Hint**

  请求被插件处理，并在其响应中添加了某些信息（hint）后继续交由下一个插件处理。这些额外的信息将组成对客户端的最终响应，如 `metric` 插件；

CoreDNS 如何处理 DNS 请求

如果 Corefile 为：

Code

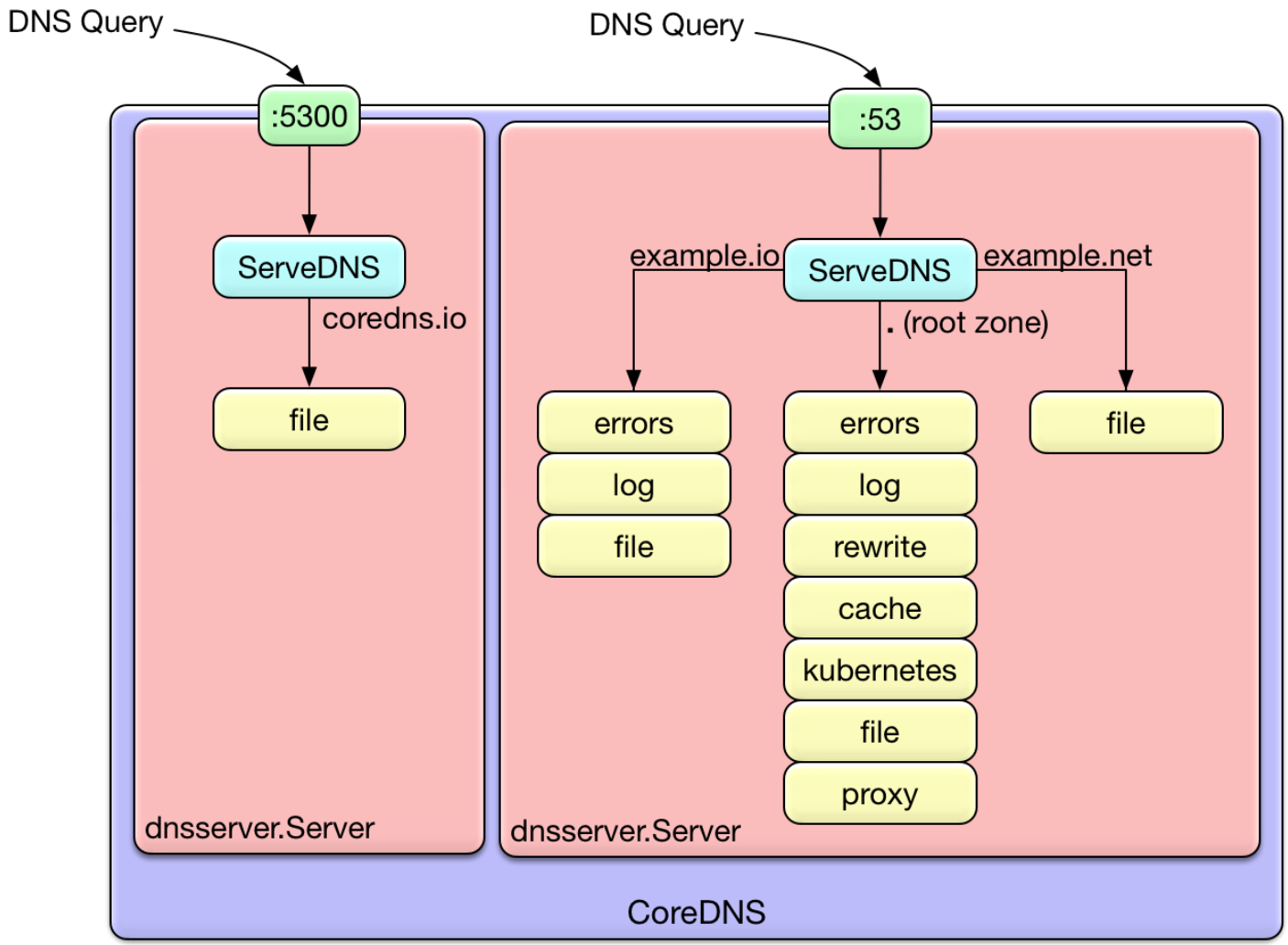
```
coredns.io:5300 {
    file /etc/coredns/zones/coredns.io.db
}

example.io:53 {
    errors
    log
    file /etc/coredns/zones/example.io.db
}

example.net:53 {
    file /etc/coredns/zones/example.net.db
}

.:53 {
    errors
    log
    health
    rewrite name foo.example.com foo.default.svc.cluster.local
}
```

从配置文件来看，我们定义了两个 server（尽管有 4 个区块），分别监听在 5300 和 53 端口。其逻辑图可如下所示：



每个进入到某个 server 的请求将按照 `plugin.cfg` 定义顺序执行其已经加载的插件。

从上图，我们需要注意以下几点：

- 尽管在 `.:53` 配置了 `health` 插件，但是它并为在上面的逻辑图中出现，原因是：该插件并未参与请求相关的逻辑（即并没有在插件链上），只是修改了 server 配置。更一般地，我们可以将插件分为两

种：

- **Normal 插件**：参与请求相关的逻辑，且插入到插件链中；
- **其他插件**：不参与请求相关的逻辑，也不出现在插件链中，只是用于修改 server 的配置，如 `health`， `tls` 等插件；

在 Kuberntes 中使用 CoreDNS

CoreDNS 其实和 kubedns 可以等价替换，我们在实际把 kubedns 更新为 CoreDNS 的时候，可以部署 CoreDNS 的 Pod 并设置新的 Service，然后通过重新配置 kubelet 的 DNS 选项让新启动的 Pod 使用新的 DNS 服务。

参考文档

- CoreDNS Manual
- Corefile Explained
- How Queries Are Processed in CoreDNS

#Kubernetes #CoreDNS #network

‹ Linux Namespace 特性简要介绍 chroot 小记 ›