Origin, Architecture and Usage

Miek Gieben (miek@coredns.io; miek@miek.nl)

22nd November 2021

Looking at CoreDNS, how it works and a peek at writing plugins.

Introduction

CoreDNS — a flexible DNS server, written in Go. Runs queries through a chain of plugins. Open Source – APLv2 licensed. Started in 2016. Graduated project in the CNCF.

- **Make running a DNS server easy**. Single binary, sane defaults, autotuning of options (not always possible);
- **Allow plugins to focus on functionality**. CoreDNS deals with DNS details.

In this presentation: why CoreDNS, how does it work, how do you use it and some closing thoughts. (Time permitting: look at source code)

Enough DNS Servers Already?

Yes, but maybe not? Wasn't planning to write one. *But*:

- Wanted Prometheus metrics out of BIND9; was quite a kludge;
- Already written a nice DNS library (miekg/dns);
- A DNS server in Go didn't exist.

Creation documented at `https://miek.nl/2016/march/14/first-light/`.

Popular in service discovery space (Kubernetes), also suited for plain old DNS stuff.

How Does it Work: Architecture

```
     +-----+    +-----+   +-----+   +------+
     | DNS |    | DoH |   | DoT |   | gRPC |   - protocols
+----+-----+----+-----+---+-----+---+------+-------+
|                                                  |
|            +-----+  +-----+  +-----+  +------+   |
| CoreDNS    |plug1|  |plug2|  | ... |  | ...  |   |
|            +-----+  +-----+  +-----+  +------+   |
|                                                  |
+-------+-------------+------+----------------+----+
        |  miekg/dns  |      |  mholt/caddy   | - libraries
        +-------------+      +----------------+
          DNS protocol         plugins + conf
```

Corefile

Main configuration for CoreDNS.

```
ZONE[:PORT] {
  PLUG1  # comment
  PLUG2
}
ANOTHERZONE[:PORT] {
  PLUG3
}
```

- `https://example.net` → DoH
- `tls://example.net` → DoT
- `grpc://example.net` → gRPC
- `dns://example.net` → DNS (default)
- `quic://example.net` → QUIC (once implemented)

DoH, DoT, QUIC and gRPC need TLS certs (*tls* plugin).

Corefile II

```
coredns.io {
  file db.coredns.io.signed
  transfer {
   to * 185.49.140.62
  }
  sign coredns.io {
   key file Kcoredns.io.+013+16376
  }
}
example.net:1053 {
   file example.net.db
   log
```

Creates 2 (internal servers)

1. runs on port 53 and is authoritative for coredns.io.
2. runs on port 1053 and answers example.net queries.

Both chain a few plugins, *file*, *transfer*, *sign* and *log*

CoreDNS

Corefile III

file - load RFC 1035 zone data from disk.

transfer - outgoing zone transfers.

sign - auto-sign zone file (i.e. DNSSEC).

log - query log (to standard output).

plugin/sign: Signing ”coredns.io.” because inception ”2020-09-30T17:20:46.000Z” was more than: 144h0m0s ago

plugin/sign: Successfully signed zone ”coredns.io.” in ”db.coredns.io.signed” with key tags ”16376” and 1602010680 SOA serial

plugin/file: Successfully reloaded zone ”coredns.io.” in ”db.coredns.io.signed” with 1602010680 SOA serial

plugin/transfer: Sent notifies for zone ”coredns.io.” to [* 185.49.140.62:53]

Type of Plugins

No official type system, but plugins can:

- generate response from code/data backend: *whoami*, *forward*, *kubernetes*, ...
- inspect query and perform action: *log*, *cache*, ...
- inspect and change query: *rewrite*, *template*, *bufsize*, ...
- affect CoreDNS process: *health*, *ready*, *reload*, *on*, *prometheus*, ...
- Corefile helpers: *import*.

The order in which plugins are traversed is set at compile time, ordering in Corefile is purely aesthetic.

Plugins for Debugging

debug

- have extra logging.
- stop recovering from `panics`.

pprof - add profile endpoint.
prometheus - add prometheus metrics endpoint.
trace - opentracing of the request (what functions are called internally). *dnstap* - DNS tap support (DNS standard).

CoreDNS

prometheus

```
example.net {
 log
 whoami
 prometheus
}
```

```
% # reads Corefile in .
% coredns -p 1053
example.net.:1053
CoreDNS-1.8.6
linux/amd64, go1.17.3,
```

```
% dig @localhost whoami.example.net -p 1053 +noall +add
whoami.example.net. 0 IN A 127.0.0.1
_udp.whoami.example.net. 0 IN SRV 0 0 44962 .
```

prometheus II

```
[INFO] 127.0.0.1:46297 - 15653 "A IN whoami.example.net. udp 59 false
    4096" NOERROR qr,aa,rd 112 0.000146456s
```

And metrics:

```
% curl -s http://localhost:9153/metrics | grep '^coredns_'
...
coredns_dns_responses_total{rcode="NOERROR",
    server="dns://:1053",zone="example.net."} 4
...
```

CoreDNS

dnstap

taps into a server and shows queries performed/received.

```
.:1053 {
    forward . 8.8.8.8
    log
    dnstap /tmp/dnstap.sock full
}
```

Run `dnstap` on the same socket and see what's happening.

Assumes queries end up in CoreDNS, if not use last report: `tcpdump`.

Project

- Github: `https://github.com/coredns/coredns`. Discussions mostly via issues.
- Each plugin owned by person (via `CODEOWNERS`).
- Release every few months,
- The ”core” is relative stable, mostly bug fixes in plugins or new plugins.
- Because Go: compiles to x64, ARM, MIPS, ..., 9 architectures/OSes in current release.

CoreDNS

Personal Reflections on Developing CoreDNS

- UDP is hard to deal with. TCP saw many optimizations (in Linux at least), UDP not a lot (QUIC will change this)
- Switching to kernel mode (system calls) is expensive, *forward* plugin is hampered by that.
- Caching and full recursive resolver is harder than it looks.

Might start writing full recursive DNS resolver, which should solve some of these issues, mostly seen in *forward* and (to some extend) *cache* plugin.

Closer Look at Plugins

Plugin Building Blocks

Each plugin has (basically) four parts:

1. A `README.md` - man page like structured document detailing all you need to know about this plugin. Includes: health information, metrics, etc..
2. A `setup()` function that parses the configuration out of the Corefile.
3. The `ServeDNS()` function that does the DNS serving.
4. Tests, both e2e and unit.

Look at Fun Little Plugin: *any*

Very small plugin: *any* - gives a minimal response to ANY queries.

```
;; ANSWER SECTION:
whoami.example.net. 8482 IN HINFO "ANY obsoleted" "See RFC 8482"
```

Let look at its source:

`https://github.com/coredns/coredns/blob/master/plugin/any/...`

CoreDNS

Source of *any*, `README.md`

```
# any

## Name
*any* - gives a minimal response to ANY queries.

## Description
*any* basically blocks ANY queries by responding ...

## Syntax
~~~ txt
any
~~~
```

Source of *any*, types

```
 // Any is a plugin that returns a HINFO reply to ANY queries.
type Any struct {
  Next plugin.Handler
}
// Name implements the Handler interface.
func (a Any) Name() string { return "any" }
```

Source of *any*, `setup()`

```
func init() { plugin.Register("any", setup) }

func setup(c *caddy.Controller) error {
  a := Any{}

  dnsserver.GetConfig(c).AddPlugin(func(next plugin.Handler)
            plugin.Handler {
    a.Next = next
    return a
  })
  return nil
}
```

Source of *any*, `ServeDNS()`

```
// ServeDNS implements the plugin.Handler interface.
func (a Any) ServeDNS(ctx context.Context, w dns.ResponseWriter, r *dns.Msg)
    (int, error) {
  if r.Question[0].Qtype != dns.TypeANY {
    return plugin.NextOrFailure(a.Name(), a.Next, ctx, w, r)
  }
  m := new(dns.Msg)
  m.SetReply(r)
  hdr := dns.RR_Header{Name: r.Question[0].Name, Ttl: 8482, Class: dns.ClassINET,
    Rrtype: dns.TypeHINFO}
  m.Answer = []dns.RR{ &dns.HINFO{Hdr: hdr, Cpu: "ANY obsoleted", Os: "See RFC 84
  w.WriteMsg(m)
  return 0, nil
}
```

CoreDNS

CoreDNS

- A capable DNS server, and popular (150M docker pulls)
- Extensive tests; cold run takes 90s.
- Easy to extend. No DNS knowledge needed (YMMV).
- Exiting new developments on the horizon.

Thank you!