COREDNS INTRO

KUBECON, COPENHAGEN

MIEK GIEBEN

MAY 2-4, 2018

In this presentation

- CoreDNS status and roadmap-ish
- CoreDNS in k8s
- Debugging features
- Plugins
    - Write your own: NXDOMAIN domains

STATUS

CoreDNS: flexible DNS server powered by plugins (support DNS, DNS over TLS, DNS over gRPC)

- Picking new developers (thanks!)
- Moved to OWNERS system for plugins
- Security review from Cure53 (sponsored by the CNCF)
  - one big hole plugged with 1.1.1 (cache spoofing)
  - 2 other minor bugs
- Release cadence: every 3 to 4 weeks

ROADMAP(-ISH)

- Core (non-plugin side) very stable; occasional feature enhancements
- Plugins
  - *kubernetes*: small tweaks here and there
  - *etcd3* in the works
  - *sql* serve from database (in the works)
    - note: explugins/pdsql
- Watch functionality:
  - gRPC based, register interested in *name*
  - get ping when name changes

COREDNS IN K8S

GA in 1.11 (I hope) some issues need to be worked out.

- pushing gcr.io - CoreDNS being the guinea pig for a new process

> *"This is progressing"*

COREDNS CONFIGURATION IN K8S

Kubernetes setup comes with this Corefile:

```
.:53 {
    errors
    health
    kubernetes cluster.local 10/8 {
      pods insecure
      upstream
      fallthrough in-addr.arpa
    }
    prometheus :9153
    proxy . /etc/resolv.conf
    cache 30
}
```

Each line a plugin.

`.:53` a server for *everything* on port 53.

ERRORS

Log errors from plugins to standard output.

i.e. `unreachable backend: <error>` from the *proxy* plugin.

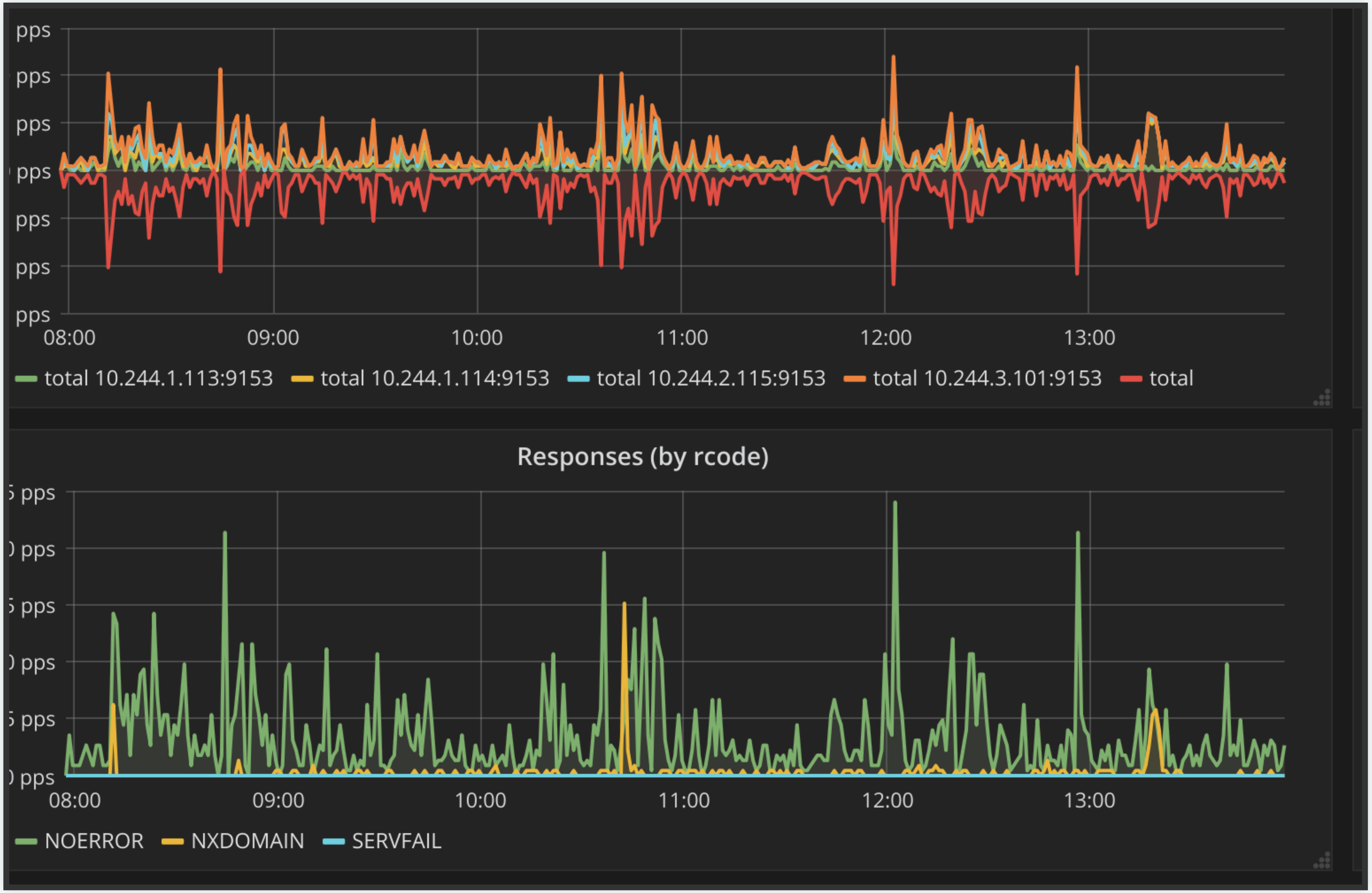
Note: all plugins have documention at: https://coredns.io/plugins/<NAME>

HEALTH

Expose health on port 8080. OK once we've synced to k8s api. (SERVFAIL until that happens)

PROMETHEUS

Metrics exposed on 9153; queries, latency, etc.



Grafana dashboard

PROXY

Forward queries *not* for `cluster.local` to nameserver(s) defined in `/etc/resolv.conf`

CACHE

Cache every (sane) response up to 30 seconds. Can do other stuff: *prefetching*, seperate *positive/negative* cache.

KUBERNETES

Kubernetes provides K8S service discovery.

```
kubernetes cluster.local 10/8 {
  pods insecure
  upstream
  fallthrough in-addr.arpa
}
```

- handle `cluster.local`
- 10/8: `10.in-addr.arpa` (10.x.x.x reverse lookups)

DIRECTIVES

- `pods insecure`: always return A record for IP requests `1-2-3-4.default.pod.cluster.local.` → 1.2.3.4 .
- `upstream`: resolve external names.
- `fallthrough in-addr.arpa` - reverse (PTR) lookups, fallthrough to proxy when NXDOMAIN.

NEED MORE?

`log` and maybe `debug`?

- `log`: log all processed queries to standard output
- `debug`: enable debug logging
  - disables `recover` after `panic`

DEBUGGING

- with *debug* `log.Debug` enabled output shown.
- zone transfers in *kubernetes*: Give me **all** records. enable with "`tranfer to *`", check with `dig AXFR cluster.local`.
- *health* exports metric: `coredns_health_request_duration_seconds`
- (external) plugin *dump*: log everything *before* any processing. (with `tcpdump`).

feedback/ideas/help welcome!

PROBLEM

dnsmasq has this feature: NXDOMAIN domains

```
server=/example.net/
```

Shortcuts the resolving of this domain.

I want this in CoreDNS and don't want to deepdive into *rewrite* or *template*.

Let's write a plugin that does the same thing. (github.com/miekg/nxdomain)

PLUGIN

For a plugin a work, we need:

1. (Go) code the implements the `plugin.Handler` interface:
2. A setup function that parses the Corefile.
3. To register it into CoreDNS.

HANDLER INTERFACE

```
Name() string
ServeDNS(context.Context, dns.ResponseWriter, *dns.Msg) \
  (int, error)
```

- **Name** return name of the plugin (nxdomain)
- **ServerDNS** guts of the plugin: query handling

```
// N implements the plugin interface.
type N struct {
    Next  plugin.Handler // needed for chaining the plugins
    names []string       // domains we shortcut
}
```

SETUP

```go
func setup(c *caddy.Controller) error {
    // Parse: nxdomain <list of domains>
    names := []string{}
    for c.Next() {
        args := c.RemainingArgs()
        if len(args) == 0 {
            return plugin.Error("nxdomain", c.ArgErr())
        }
        // I'll bet these are not fully qualified.
        for _, a := range args {
            names = append(names, dns.Fqdn(a))
        }
    }
    // ...
```

REGISTER IN COREDNS

```
    // ...
    dnsserver.GetConfig(c).AddPlugin(func(
      next plugin.Handler) plugin.Handler {
        return N{Next: next, names: names}
    })

    return nil
}
```

SERVEDNS

```go
func (n N) ServeDNS(ctx context.Context,
      w dns.ResponseWriter, r *dns.Msg) (int, error) {
    for _, n := range n.names {
        if dns.IsSubDomain(n, r.Question[0].Name) {
            m := new(dns.Msg)
            m.SetRcode(r, dns.RcodeNameError)
            m.Ns = []dns.RR{soa(n)}
            w.WriteMsg(m)
            return 0, nil
        }
    }
    return plugin.NextOrFailure(n.Name(), n.Next, ctx, w, r)
}
```

COMPILING COREDNS

In `plugin.cfg`

```
...
debug:debug
trace:trace
health:health
nxdomain:github.com/miekg/nxdomain
pprof:pprof
prometheus:metrics
errors:errors
log:log
...
```

```
$ go generate && go build
```

Check if it's there: `$ coredns -plugins` (should list `dns.nxdomain`)

TRYING OUT

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

```
$ coredns -dns.port=1043
$ dig @localhost -p 1043 example.net
;; ADDITIONAL SECTION:
example.net.        0   IN  AAAA    ::1
_udp.example.net.   0   IN  SRV 0 0 38755 .

$ dig @localhost -p 1053 example.org
->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 46392
```

That's all! Questions?