
# Kubernetes DNS Horror Stories (and how to avoid them)

*Laurent Bernaille*
*Staff Engineer, Datadog*

# Datadog



Over 350 integrations
Over 1,200 employees
Over 8,000 customers
Runs on millions of hosts
Trillions of data points per day

10000s hosts in our infra
10s of k8s clusters with 100-4000 nodes
Multi-cloud
Very fast growth

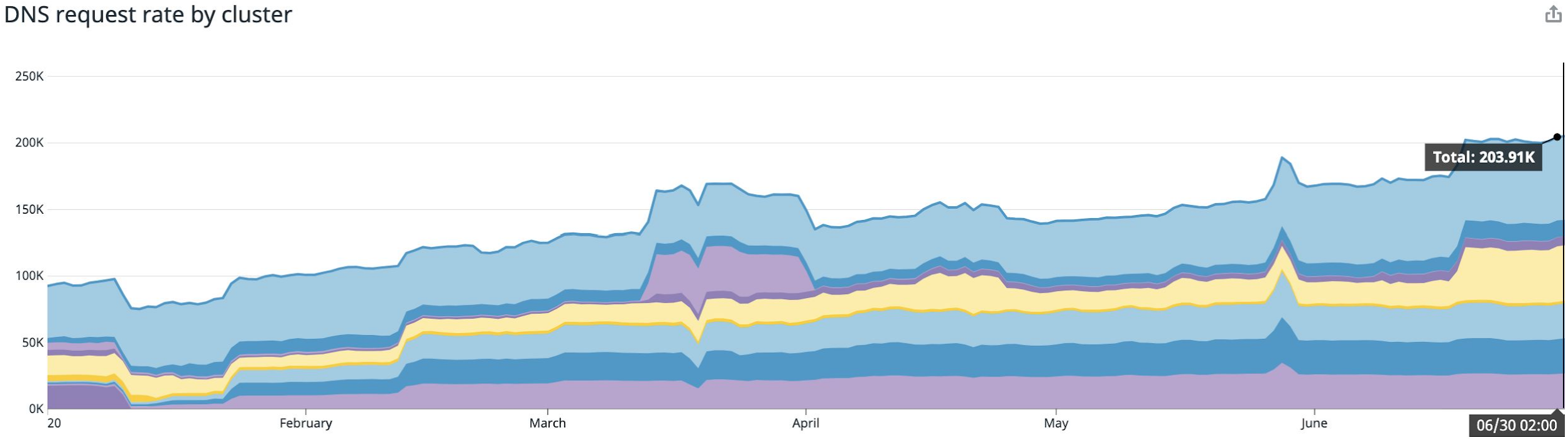
# Challenges



- Scalability
- Containerization
- Consistency across cloud providers
- Networking
- Ecosystem youth
- Edge-cases

# What we did not expect






Our Kubernetes DNS infrastructure currently serves **~200 000 DNS queries per second**

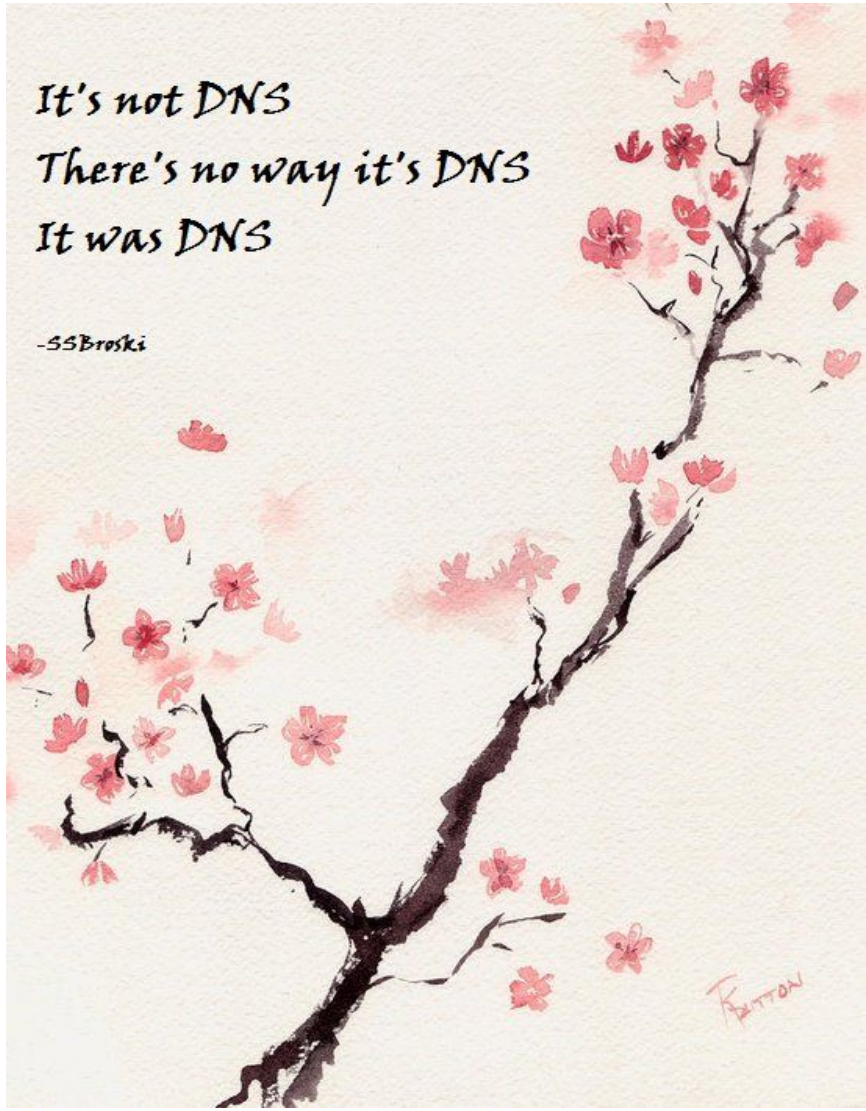
Largest cluster is at ~60 000 DNS queries per second

**DNS is one of your more critical services when running Kubernetes**

# Outline



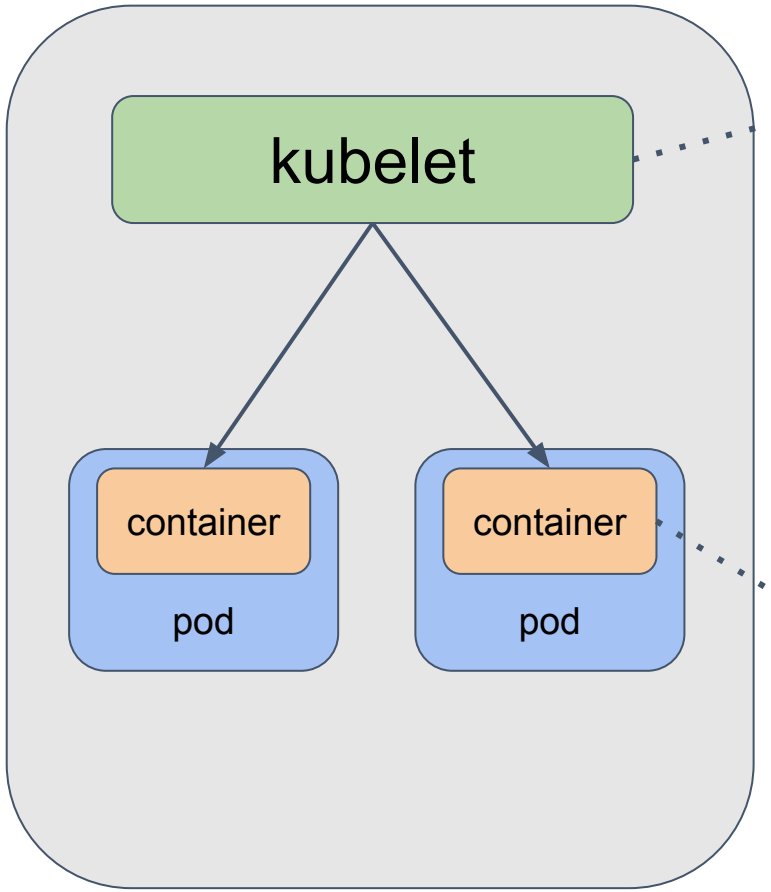
- DNS in Kubernetes
- Challenges
- "Fun" stories
- What we do now




# DNS in Kubernetes

# How it works (by default)






**kubelet configuration**

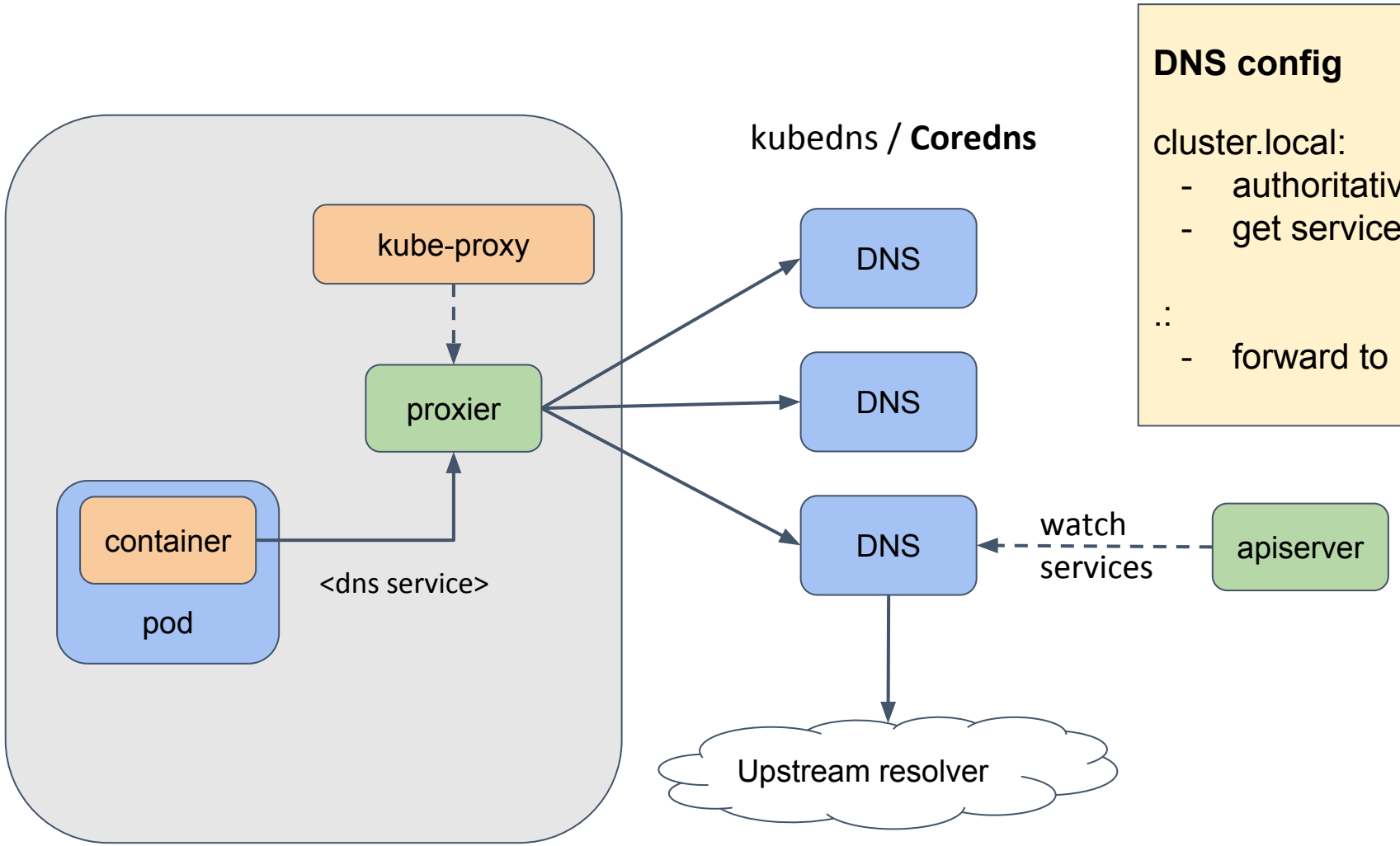
ClusterDNS: <dns service>

**/etc/resolv.conf**

```
search      <namespace>.svc.cluster.local
            svc.cluster.local
            cluster.local
            ec2.internal

nameserver <dns service>
options ndots:5
```

# Accessing DNS






**DNS config**

cluster.local:
- authoritative
- get services from apiserver

.:
- forward to upstream resolver

# Theory: Scenario 1



Pod in namespace "metrics", requesting service "points" in namespace "metrics"

**getent ahosts points**

1. points has less than 5 dots
2. With first search domain: points.metrics.svc.cluster.local
3. Answer

# Theory: Scenario 2



Pod in namespace "logs", requesting service "points" in namespace "metrics"

**getent ahosts points.metrics**

1. points.metrics has less than 5 dots
2. With first search domain: points.metrics.logs.svc.cluster.local
3. Answer: NXDOMAIN
4. With second domain points.metrics.svc.cluster.local
5. Answer


# Challenges

# In practice



Pod in namespace default, requesting www.google.com (GKE)

```
getent ahosts www.google.com

10.220.1.4.58137 > 10.128.32.10.53:  A? www.google.com.default.svc.cluster.local. (58)
10.220.1.4.58137 > 10.128.32.10.53:  AAAA? www.google.com.default.svc.cluster.local. (58)
10.128.32.10.53 > 10.220.1.4.58137: NXDomain 0/1/0 (151)
10.128.32.10.53 > 10.220.1.4.58137: NXDomain 0/1/0 (151)
10.220.1.4.54960 > 10.128.32.10.53:  A? www.google.com.svc.cluster.local. (50)
10.220.1.4.54960 > 10.128.32.10.53:  AAAA? www.google.com.svc.cluster.local. (50)
10.128.32.10.53 > 10.220.1.4.54960: NXDomain 0/1/0 (143)
10.128.32.10.53 > 10.220.1.4.54960: NXDomain 0/1/0 (143)
10.220.1.4.51754 > 10.128.32.10.53:  A? www.google.com.cluster.local. (46)
10.220.1.4.51754 > 10.128.32.10.53:  AAAA? www.google.com.cluster.local. (46)
10.128.32.10.53 > 10.220.1.4.51754: NXDomain 0/1/0 (139)
10.128.32.10.53 > 10.220.1.4.51754: NXDomain 0/1/0 (139)
10.220.1.4.42457 > 10.128.32.10.53:  A? www.google.com.c.sandbox.internal. (59)
10.220.1.4.42457 > 10.128.32.10.53:  AAAA? www.google.com.c.sandbox.internal. (59)
10.128.32.10.53 > 10.220.1.4.42457: NXDomain 0/1/0 (148)
10.128.32.10.53 > 10.220.1.4.42457: NXDomain 0/1/0 (148)
10.220.1.4.45475 > 10.128.32.10.53:  A? www.google.com.google.internal. (48)
10.220.1.4.45475 > 10.128.32.10.53:  AAAA? www.google.com.google.internal. (48)
10.128.32.10.53 > 10.220.1.4.45475: NXDomain 0/1/0 (137)
10.128.32.10.53 > 10.220.1.4.45475: NXDomain 0/1/0 (137)
10.220.1.4.40634 > 10.128.32.10.53:  A? www.google.com. (32)
10.220.1.4.40634 > 10.128.32.10.53:  AAAA? www.google.com. (32)
10.128.32.10.53 > 10.220.1.4.40634: 3/0/0 AAAA 2a00:1450:400c:c0b::67
10.128.32.10.53 > 10.220.1.4.40634: 6/0/0 A 173.194.76.103
```

**12 queries!**

**Reasons:**
- 5 search domains
- IPv6

**Problems:**
- latency
- packet loss => 5s delay
- load on DNS infra


# Challenges: Resolver behaviors

# IPv6?



getaddrinfo will do IPv4 and IPv6 queries by "default"

**The Good:** We're ready for IPv6!

**The Bad**: Not great, because it means twice the amount of traffic

**The Ugly**: IPv6 resolution triggers packet losses in the Kubernetes context

- Accessing the DNS service requires NAT
- Race condition in netfilter when 2 packets are sent within microseconds
- Patched in the kernel (4.19+)
- Detailed issue: https://github.com/kubernetes/kubernetes/issues/56903

***If this happens for any of the 10+ queries, resolution takes 5s (at least)***
**Impact is far lower with IPVS (no DNAT)**

# Let's disable IPv6!



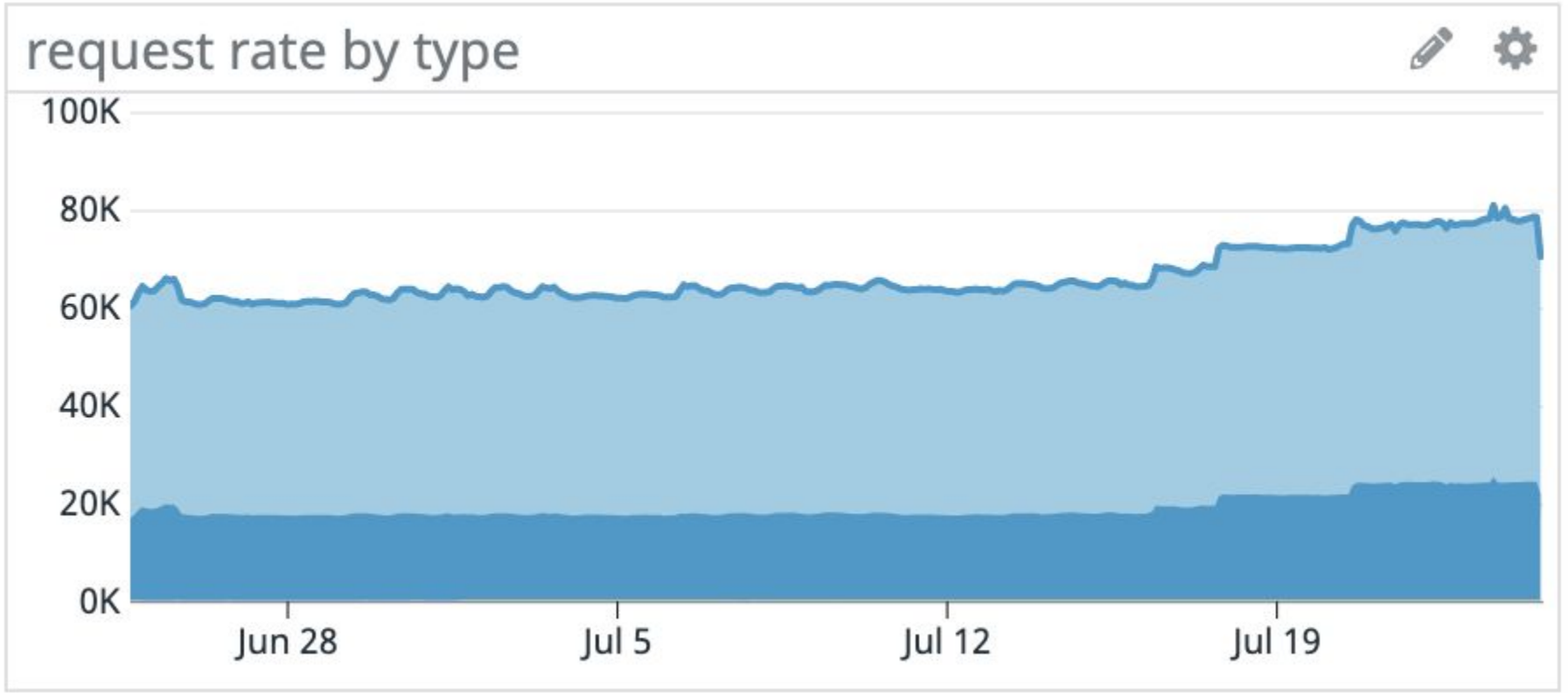
```
GRUB_CMDLINE_LINUX_DEFAULT="$GRUB_CMDLINE_LINUX_DEFAULT ipv6.disable=1"

getent ahosts www.google.com

IP 10.140.216.13.53705 > 10.129.192.2.53:  A? www.google.com.datadog.svc.cluster.local. (63)
IP 10.129.192.2.53 > 10.140.216.13.53705: NXDomain*- 0/1/0 (171)
IP 10.140.216.13.34772 > 10.129.192.2.53:  A? www.google.com.svc.cluster.local. (55)
IP 10.129.192.2.53 > 10.140.216.13.34772: NXDomain*- 0/1/0 (163)
IP 10.140.216.13.54617 > 10.129.192.2.53:  A? www.google.com.cluster.local. (51)
IP 10.129.192.2.53 > 10.140.216.13.54617: NXDomain*- 0/1/0 (159)
IP 10.140.216.13.55732 > 10.129.192.2.53:  A? www.google.com.ec2.internal. (45)
IP 10.129.192.2.53 > 10.140.216.13.55732: NXDomain 0/0/0 (45)
IP 10.140.216.13.36991 > 10.129.192.2.53:  A? www.google.com. (32)
IP 10.129.192.2.53 > 10.140.216.13.36991: 1/0/0 A 172.217.2.100 (62)
```

Much better !

No IPv6: no risk of drops, 50% less traffic

# We wanted to celebrate, but...






A

AAAA

Still a lot of AAAA queries
Where are they coming from?

# What triggers IPv6?



According to POSIX, getaddrinfo should do IPv4 and IPv6 by default

**BUT glibc includes hint AI_ADDRCONFIG by default**

> According to POSIX.1, specifying *hints* as NULL should cause *ai_flags*
> to be assumed as 0. The GNU C library instead assumes a value of
> **(AI_V4MAPPED | AI_ADDRCONFIG)** for this case, since this value is
> considered an improvement on the specification.
>
> **man getaddrinfo, glibc**

**AND AI_ADDRCONFIG only makes IPv6 queries if it finds an IPv6 address**

> If *hints*.ai_flags includes the **AI_ADDRCONFIG** flag, then IPv4
> addresses are returned in the list pointed to by *res* only if the
> local system has at least one IPv4 address configured

*So wait, disabling IPv6 should just work, right?*

# Alpine and Musl



Turns out Musl implements the POSIX spec, and sure enough:

**Service in the same namespace** **No hints (use defaults)**

```
getaddrinfo("echo", NULL , NULL , &servinfo)

Ubuntu base image
10.140.216.13.52563 > 10.129.192.2.53: A? echo.datadog.svc.fury.cluster.local. (53)
10.129.192.2.53 > 10.140.216.13.52563:  1/0/0 A 10.129.204.147 (104)

Alpine base image
10.141.90.160.46748 > 10.129.192.2.53: A? echo.datadog.svc.cluster.local. (53)
10.141.90.160.46748 > 10.129.192.2.53: AAAA? echo.datadog.svc.cluster.local. (53)
10.129.192.2.53 > 10.141.90.160.46748:  0/1/0 (161)
10.129.192.2.53 > 10.141.90.160.46748:  1/0/0 A 10.129.204.147 (104)
```

But, we don't use alpine that much. So what is happening?

# We use Go a lot



net.ResolveTCPAddr("tcp", "www.google.com:80"), **on Ubuntu**

```
10.140.216.13.55929 > 10.129.192.2.53: AAAA? www.google.com.datadog.svc.cluster.local. (63)
10.140.216.13.46751 > 10.129.192.2.53: A? www.google.com.datadog.svc.cluster.local. (63)
10.129.192.2.53 > 10.140.216.13.46751: NXDomain*- 0/1/0 (171)
10.129.192.2.53 > 10.140.216.13.55929: XDomain*- 0/1/0 (171)
10.140.216.13.57414 > 10.129.192.2.53: AAAA? www.google.com.svc.cluster.local. (55)
10.140.216.13.54192 > 10.129.192.2.53: A? www.google.com.svc.cluster.local. (55)
10.129.192.2.53 > 10.140.216.13.57414: NXDomain*- 0/1/0 (163)
10.129.192.2.53 > 10.140.216.13.54192: NXDomain*- 0/1/0 (163)
```

IPv6 is back...

# What about CGO?



Native Go

```
10.140.216.13.55929 > 10.129.192.2.53: AAAA? www.google.com.datadog.svc.cluster.local. (63)
10.140.216.13.46751 > 10.129.192.2.53: A? www.google.com.datadog.svc.cluster.local. (63)
10.129.192.2.53 > 10.140.216.13.46751: NXDomain*- 0/1/0 (171)
10.129.192.2.53 > 10.140.216.13.55929: XDomain*- 0/1/0 (171)
...
```

CGO: export GODEBUG=netdns=cgo

```
10.140.216.13.49382 > 10.129.192.2.53:  A? www.google.com.datadog.svc.cluster.local. (63)
10.140.216.13.49382 > 10.129.192.2.53:  AAAA? www.google.com.datadog.svc.cluster.local. (63)
10.129.192.2.53 > 10.140.216.13.49382: NXDomain*- 0/1/0 (171)
10.129.192.2.53 > 10.140.216.13.49382: NXDomain*- 0/1/0 (171)
...
```

Was GODEBUG ignored?

# Subtle difference



Native Go

```
10.140.216.13 55929 > 10.129.192.2.53: AAAA? www.google.com.datadog.svc.cluster.local. (63)
10.140.216.13 46751 > 10.129.192.2.53: A? www.google.com.datadog.svc.cluster.local. (63)
```

CGO: export GODEBUG=netdns=cgo

```
10.140.216.13 49382 > 10.129.192.2.53:  A? www.google.com.datadog.svc.cluster.local. (63)
10.140.216.13 49382 > 10.129.192.2.53:  AAAA? www.google.com.datadog.svc.cluster.local. (63)
```

Native Go uses a different source port for A and AAAA

# CGO implementation



https://github.com/golang/go/blob/master/src/net/cgo_linux.go

```
14  // NOTE(rsc): In theory there are approximately balanced
15  // arguments for and against including AI_ADDRCONFIG
16  // in the flags (it includes IPv4 results only on IPv4 systems,
17  // and similarly for IPv6), but in practice setting it causes
18  // getaddrinfo to return the wrong canonical name on Linux.
19  // So definitely leave it out.
20  const cgoAddrInfoFlags = C.AI_CANONNAME | C.AI_V4MAPPED | C.AI_ALL
```

No AI_ADDRCONFIG

So we can't really avoid IPv6 queries in Go unless we change the app

net.ResolveTCPAddr("**tcp4**", "www.google.com")
*But only with CGO...*


# Challenges: Query volume reduction

# Coredns Autopath



With this option, coredns knows the search domains and finds the right record

```
10.140.216.13.37164 > 10.129.192.23.53:  A? google.com.datadog.svc.cluster.local. (58)
10.140.216.13.37164 > 10.129.192.23.53:  AAAA? google.com.datadog.svc.cluster.local. (58)
10.129.192.23.53 > 10.140.216.13.37164:  2/0/0 CNAME google.com., A 216.58.218.238 (98)
10.129.192.23.53 > 10.140.216.13.37164:  2/0/0 CNAME google.com., AAAA 2607:f8b0:4004:808::200e (110)
```

Much better: only 2 queries instead of 10+

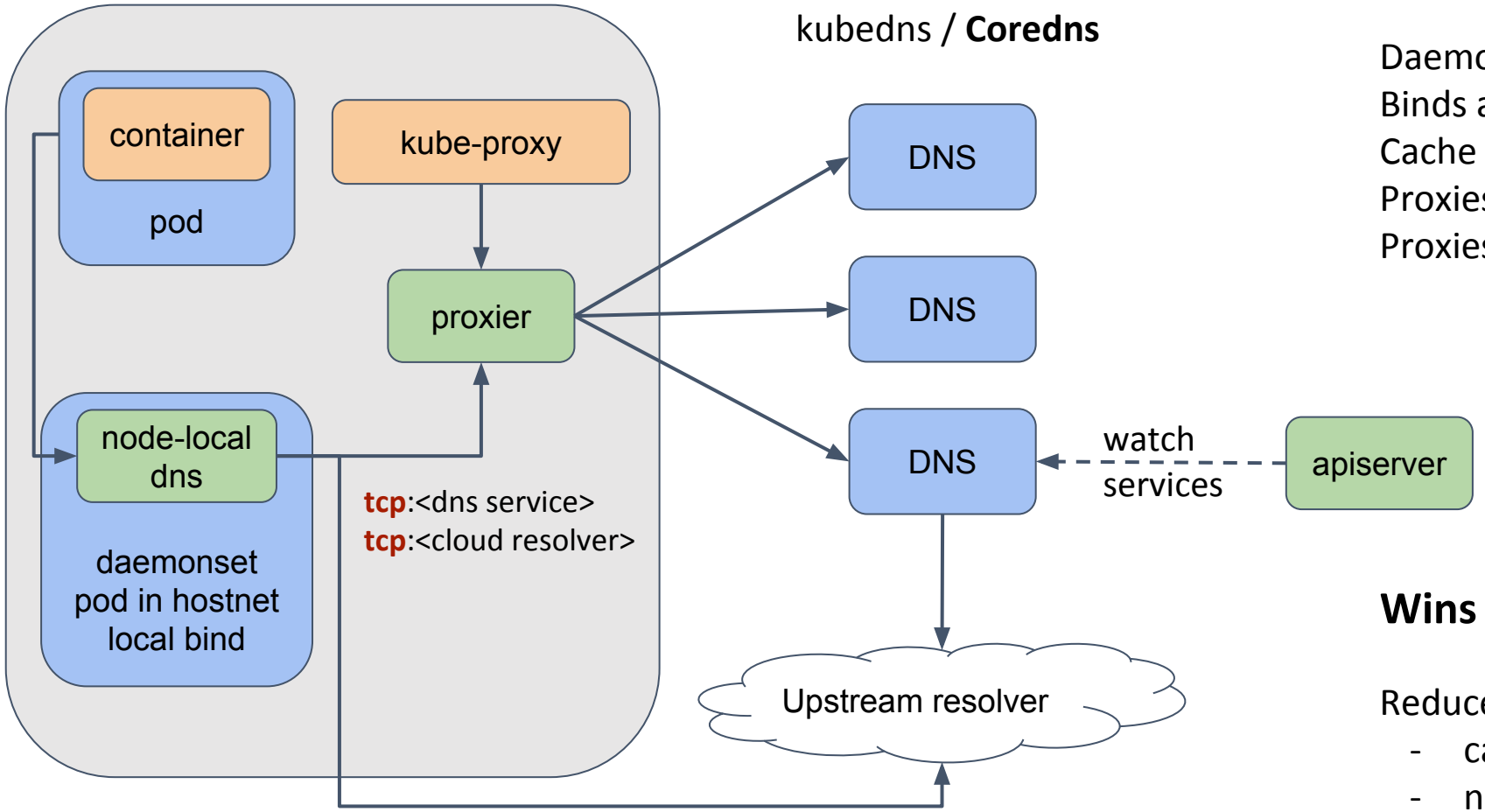
**But**

Requires to run the Coredns Kubernetes plugin with "pods: verified"

- To infer the full search domain (pod namespace)
- Memory requirements becomes proportional to number of pods
- Several OOM-killer incidents for us

# Node-local-dns






## node-local-dns

Daemonset
Binds a non-routable IP (169.x.y.z)
Cache
Proxies queries to DNS service in TCP
Proxies non-kubernetes queries directly

## Wins

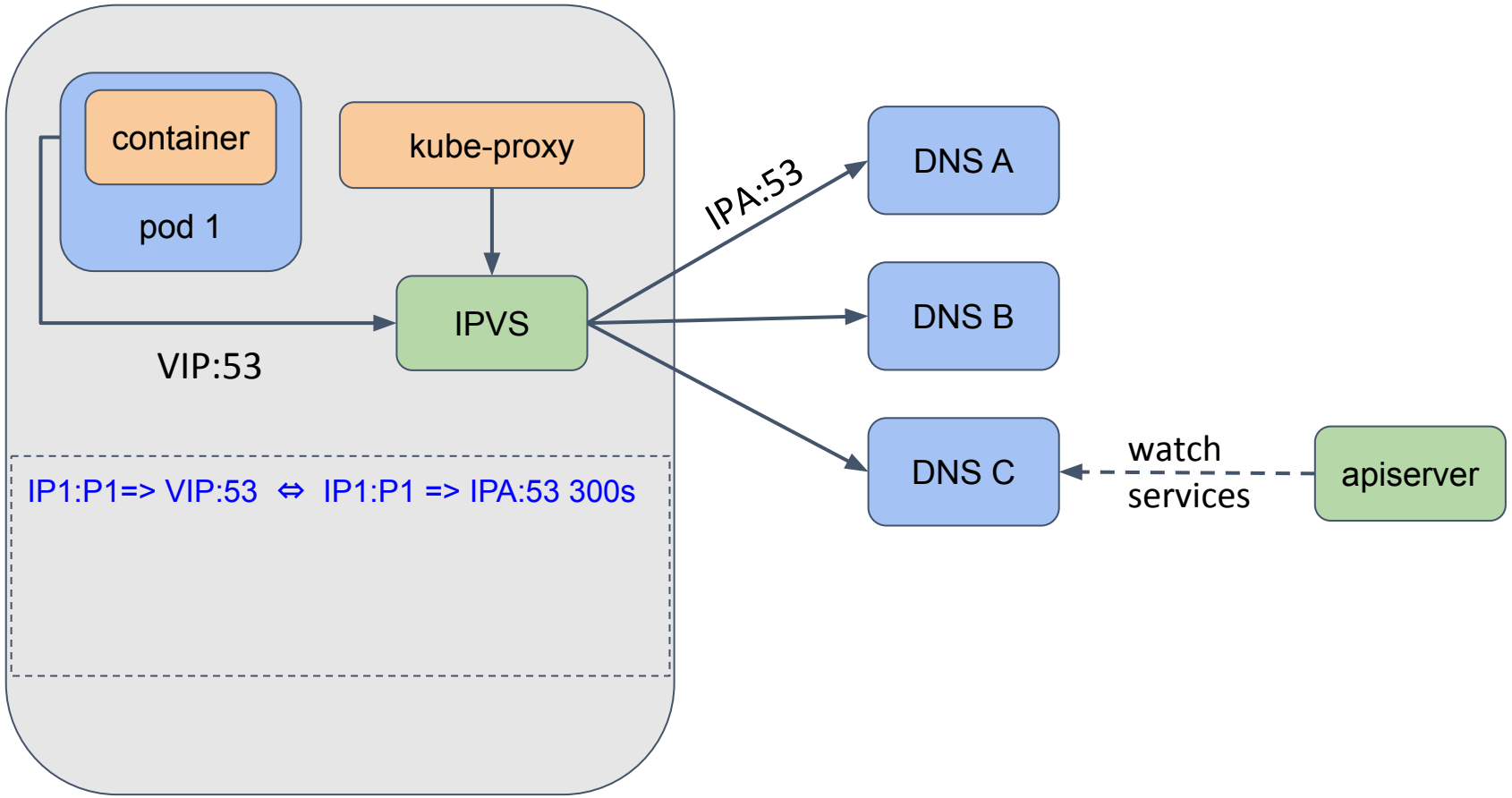
Reduces load
- cache
- non Kubernetes queries bypass service

Mitigates the netfilter race condition


# Challenges: Rolling updates

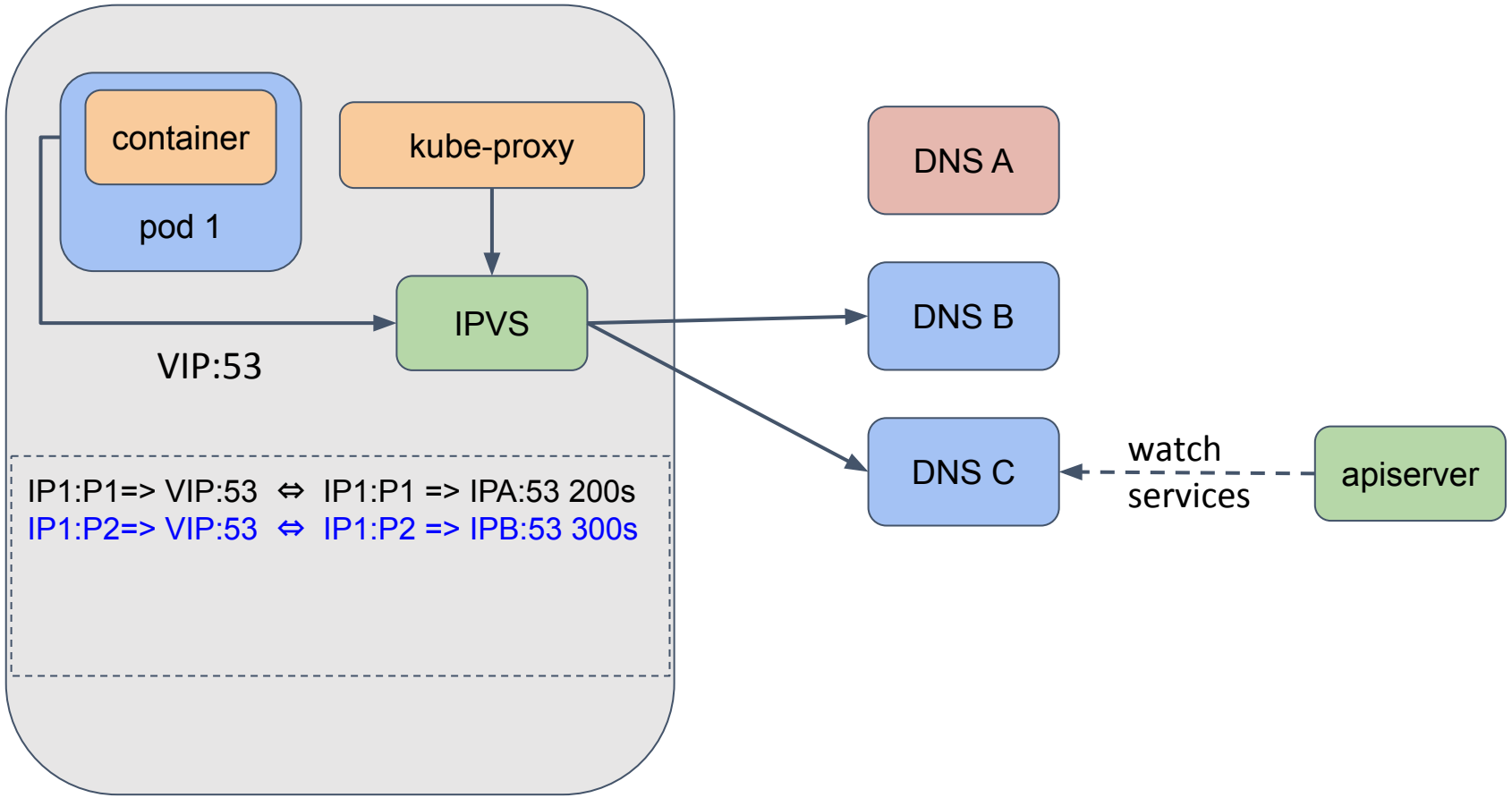
# Initial state





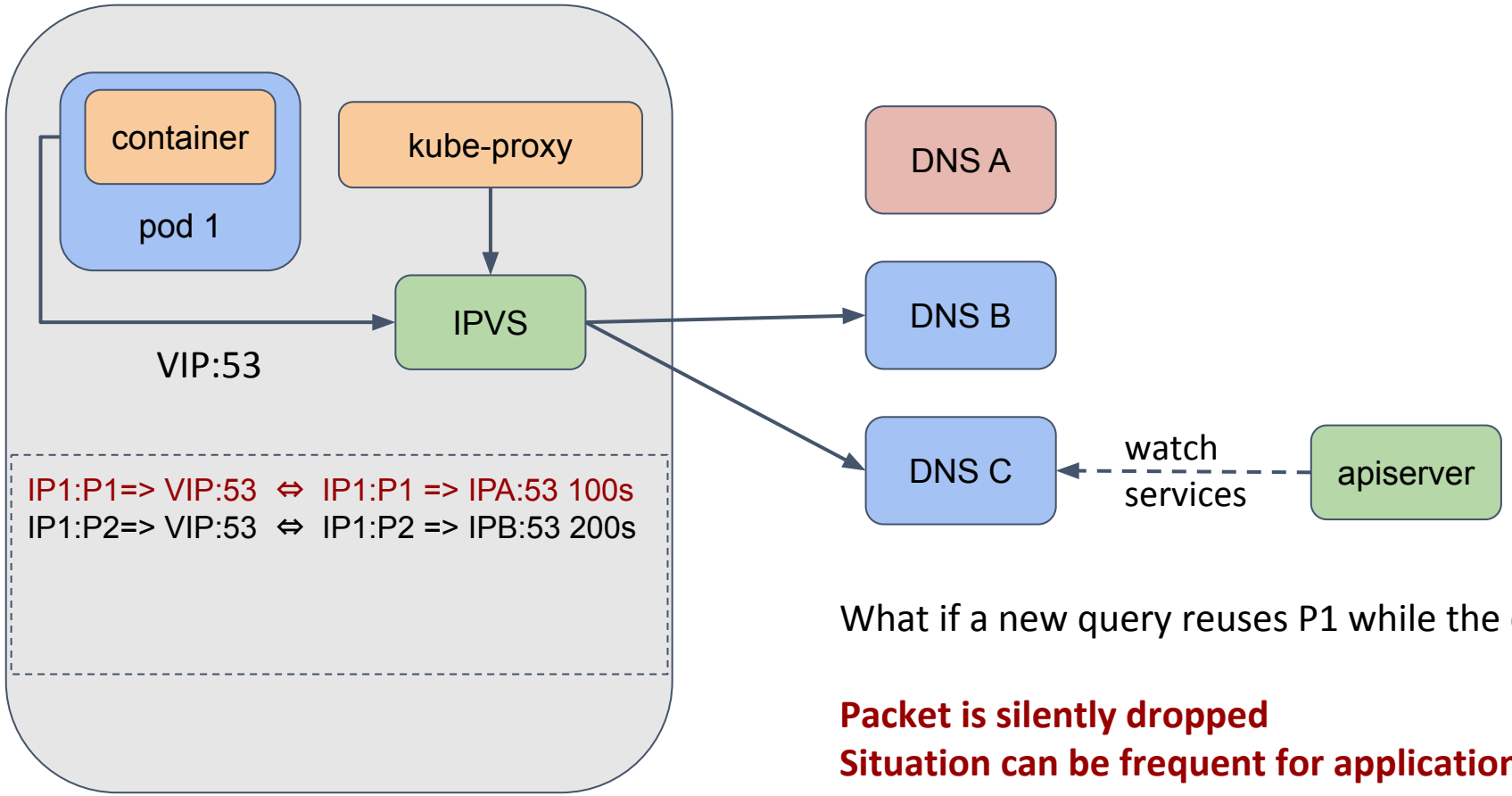
# Pod A deleted





# Source port reuse



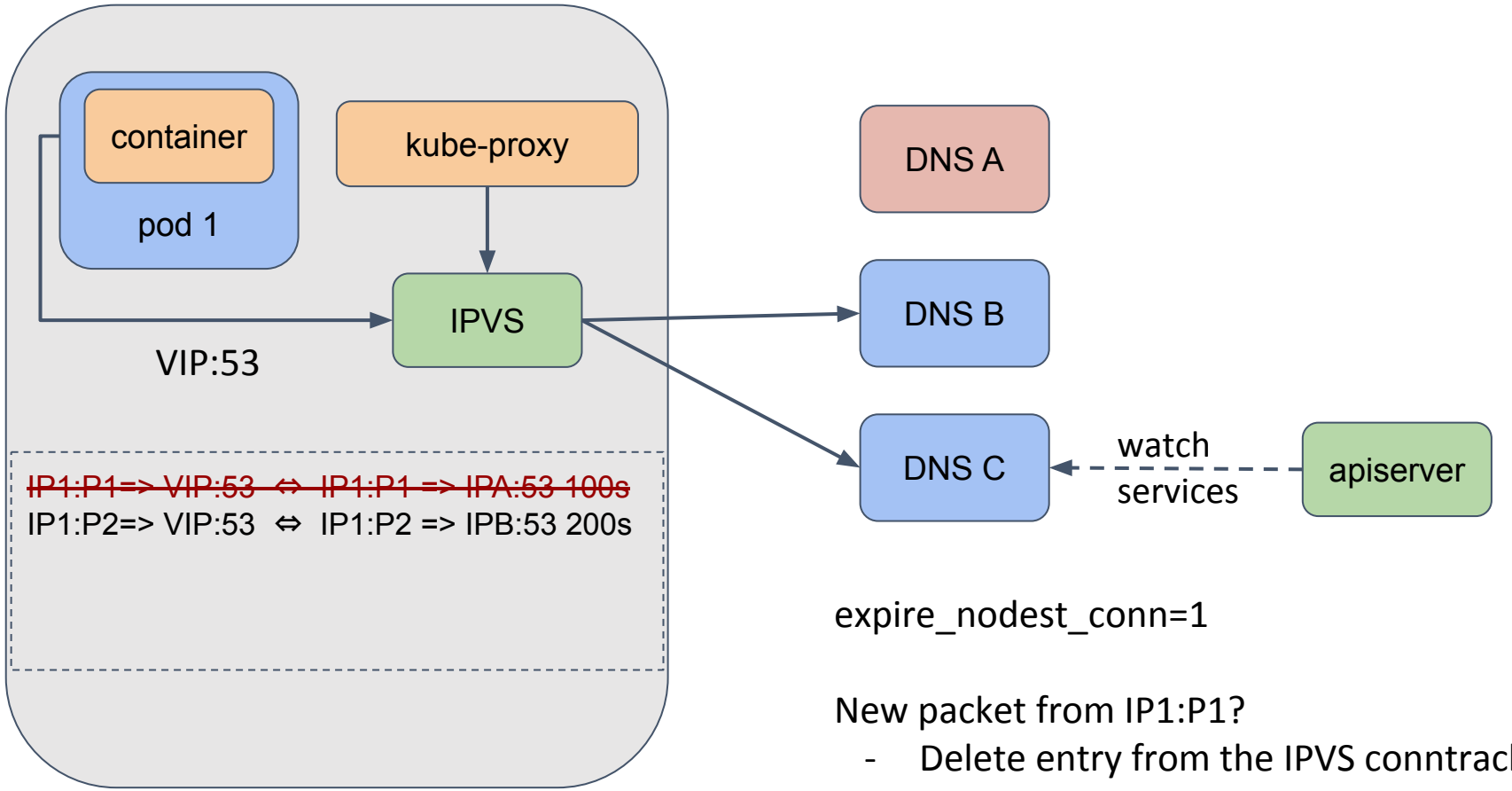



What if a new query reuses P1 while the entry is still in the IPVS conntrack?

**Packet is silently dropped**
**Situation can be frequent for applications sending a lot of DNS queries**

# Mitigation #1






expire_nodest_conn=1

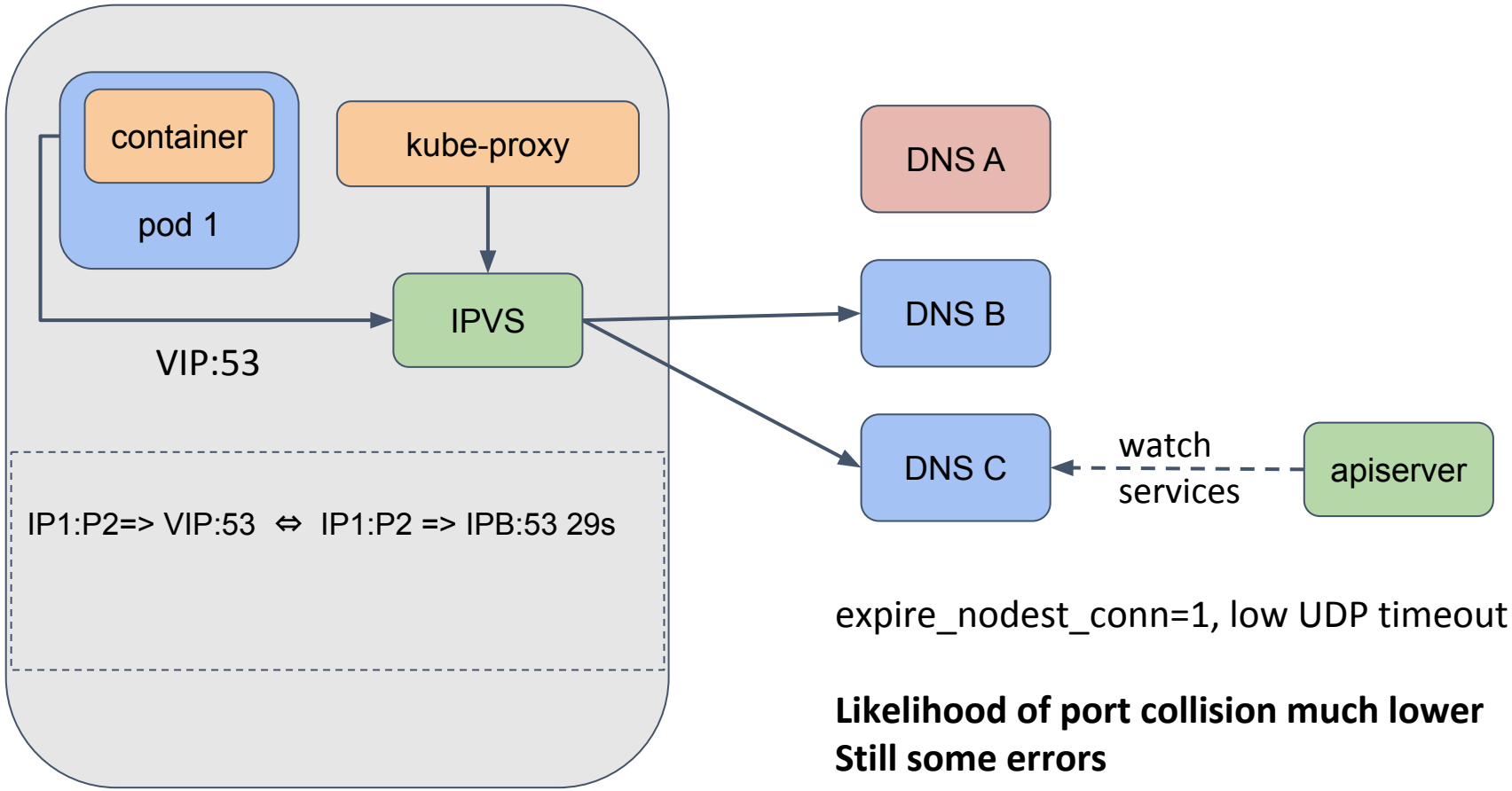
New packet from IP1:P1?

- Delete entry from the IPVS conntrack
- Send ICMP Port Unreachable

**Better, but under load, this can still trigger many errors**

# Mitigation #2






expire_nodest_conn=1, low UDP timeout (30s) [kube-proxy 1.18+]

**Likelihood of port collision much lower**
**Still some errors**

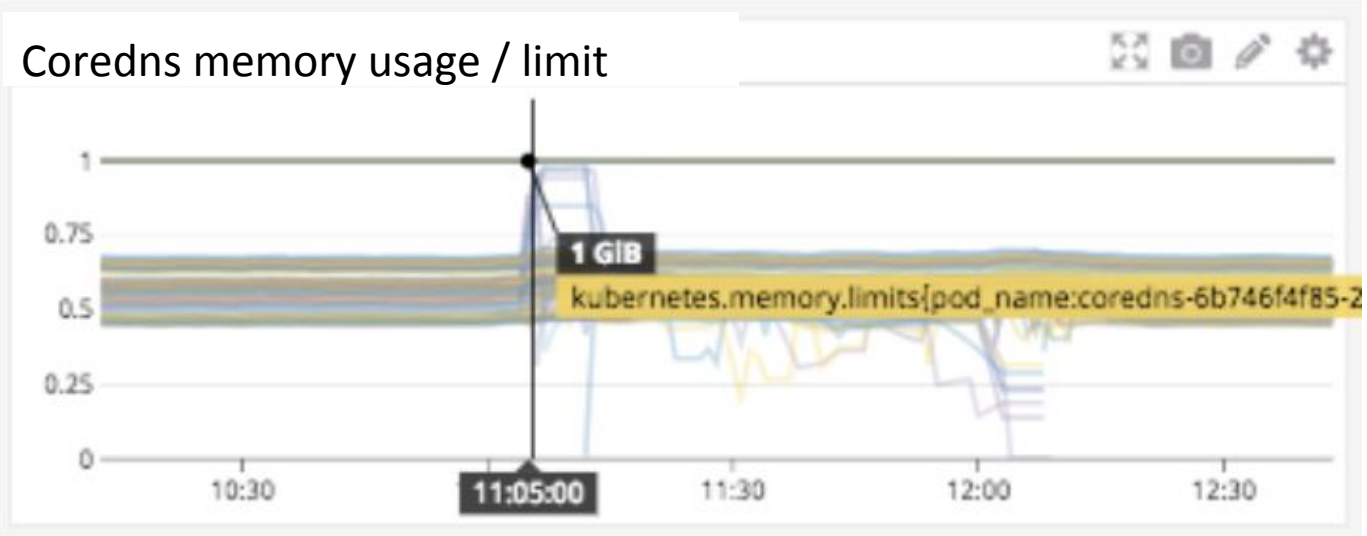
Kernel patch to expire entries on backend deletion (5.9+) by @andrewsykim


# "Fun" stories


# Sometimes your DNS is unstable

# Coredns getting OOM-killed

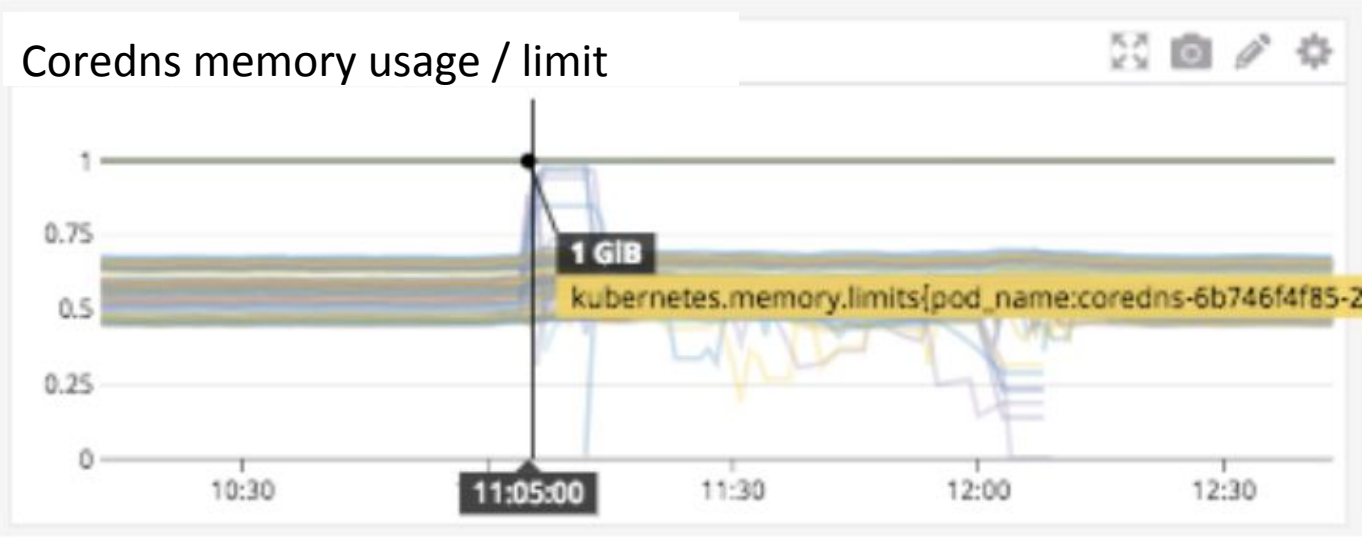





Some coredns pods getting OOM-killed
Not great for apps...

# Coredns getting OOM-killed






apiserver restart

Apiserver restarted
Coredns pod reconnected
Too much memory => OOM

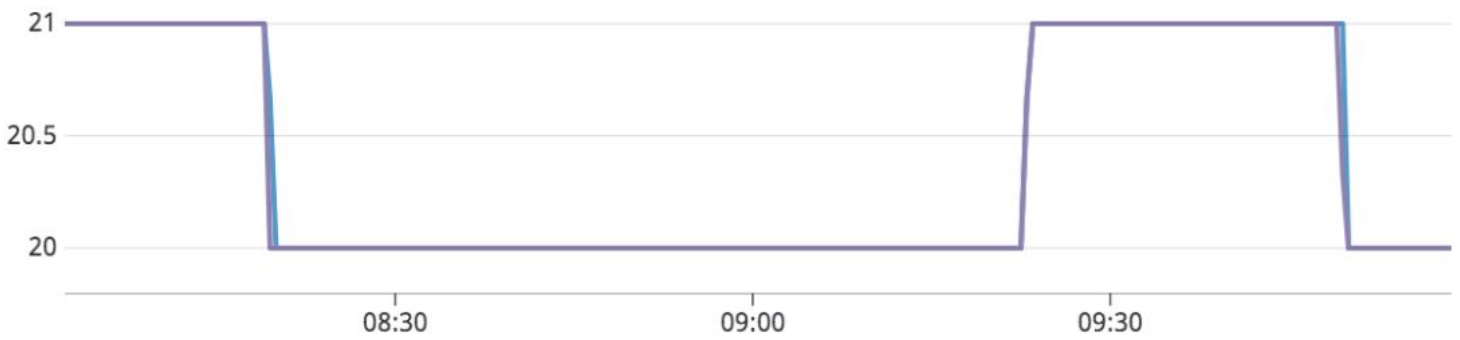
Startup requires more memory
Autopath "Pods:verified" makes sizing hard


# Sometimes Autoscaling works too well

# Proportional autoscaler



Number of coredns replicas

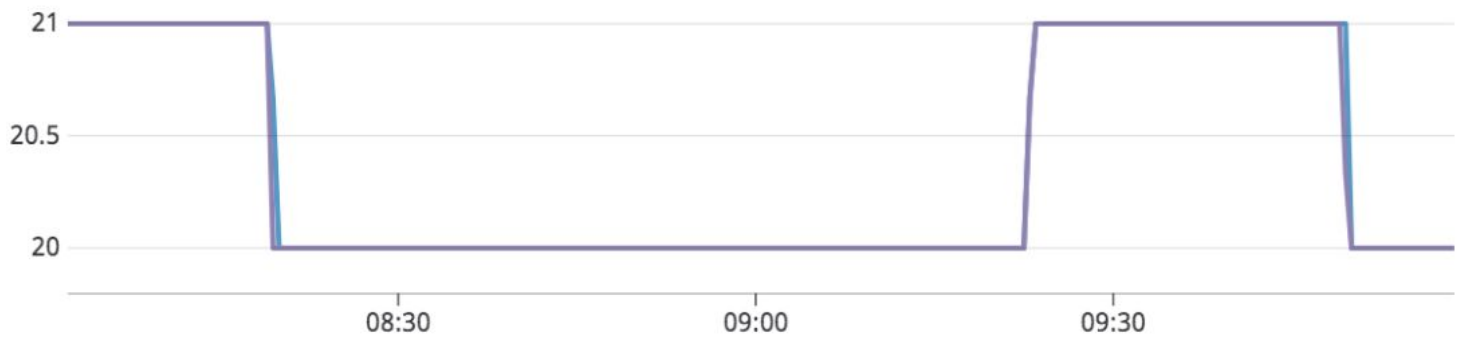



Proportional-autoscaler for coredns:
Less nodes => Less coredns pods

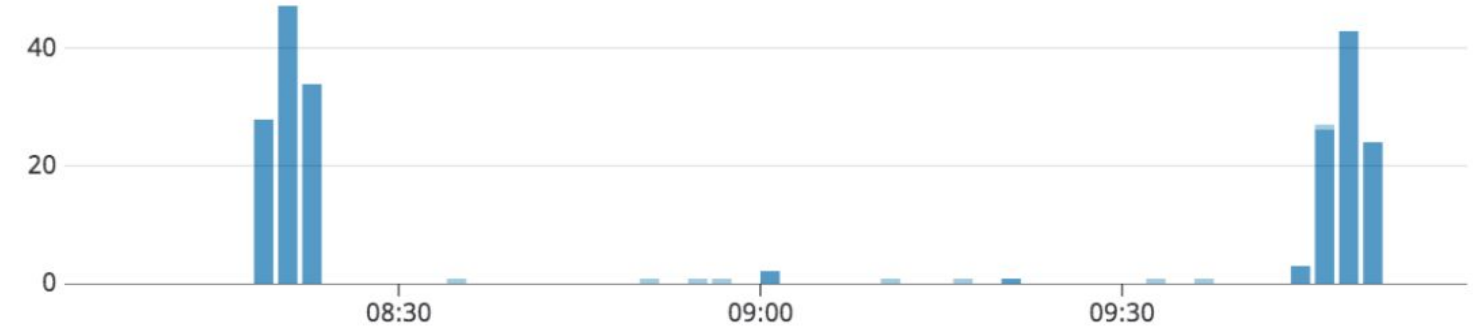
# Proportional autoscaler



Number of coredns replicas




Exceptions due to DNS failure




Triggered port reuse issue
Some applications don't like this


# Sometimes it's not your fault

# Staging fright on AWS

KubeCon CloudNativeCon Europe 2020 Virtual

```
cluster.local:53 {
    kubernetes cluster.local
}

.:53 {
     proxy . /etc/resolv.conf
     cache
}
```

```
.:53 {
    kubernetes cluster.local {
        pods verified
    }
    autopath @kubernetes

    proxy . /etc/resolv.conf
}
```

Enable autopath
Simple change right?

Can you spot what broke the staging cluster?

# Staging fright on AWS

KubeCon CloudNativeCon Europe 2020 Virtual

```
cluster.local:53 {
    kubernetes cluster.local
}

.:53 {
    proxy . /etc/resolv.conf
    cache
}
```

```
.:53 {
    kubernetes cluster.local {
        pods verified
    }
    autopath @kubernetes

    proxy . /etc/resolv.conf
}
```

With this change we disabled caching for proxied queries
AWS resolver has a strict limit: 1024 packets/second to resolver per ENI
A large proportion of forwarded queries got dropped

# Staging fright on AWS #2

KubeCon CloudNativeCon Europe 2020 Virtual

```
cluster.local:53 {
    kubernetes cluster.local
}

.:53 {
    proxy . /etc/resolv.conf
    cache
}
```

```
cluster.local:53 {
    kubernetes cluster.local
}

.:53 {
    proxy . /etc/resolv.conf
    cache
    force_tcp
}
```

Let's avoid UDP for upstream queries: avoid truncation, less errors
Sounds like a good idea?

# Staging fright on AWS #2

KubeCon CloudNativeCon Europe 2020 Virtual

```
cluster.local:53 {
    kubernetes cluster.local
}

.:53 {
    proxy . /etc/resolv.conf
    cache
}
```

```
cluster.local:53 {
    kubernetes cluster.local
}

.:53 {
    proxy . /etc/resolv.conf
    cache
    force_tcp
}
```

AWS resolver has a strict limit: 1024 packets/second to resolver per ENI

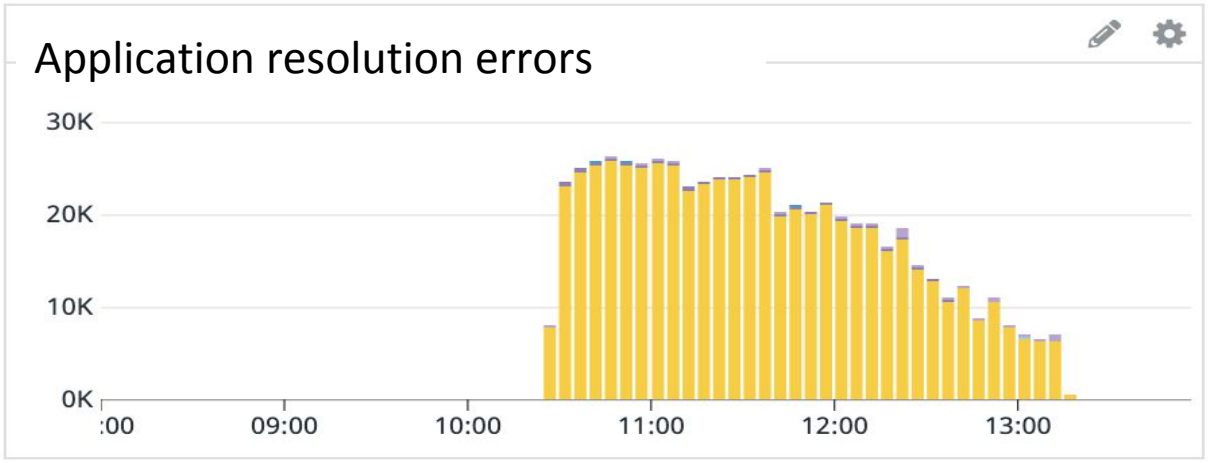
DNS queries over TCP use at least 5x more packets

**Don't query the AWS resolvers using TCP**


# Sometimes it's *really* not your fault

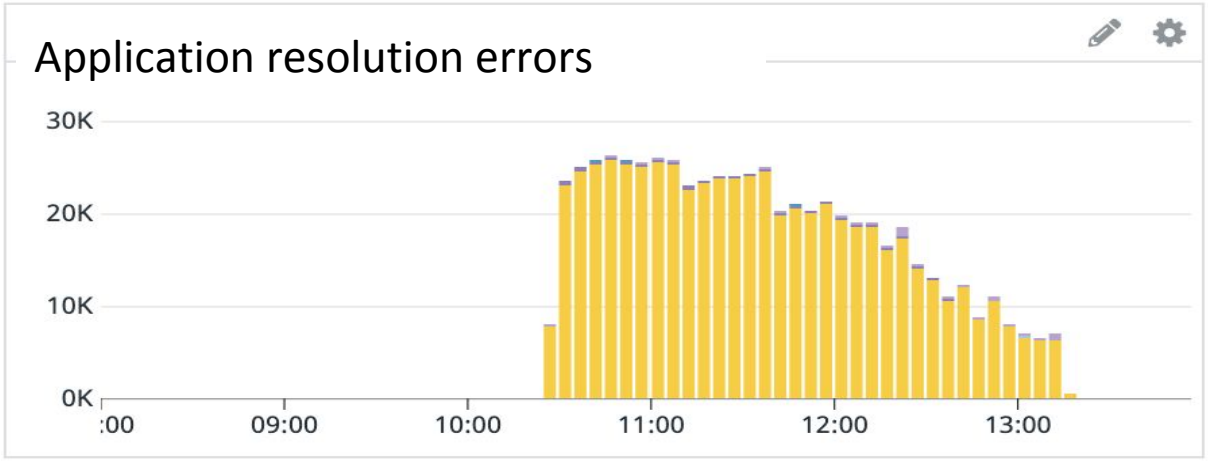
# Upstream DNS issue

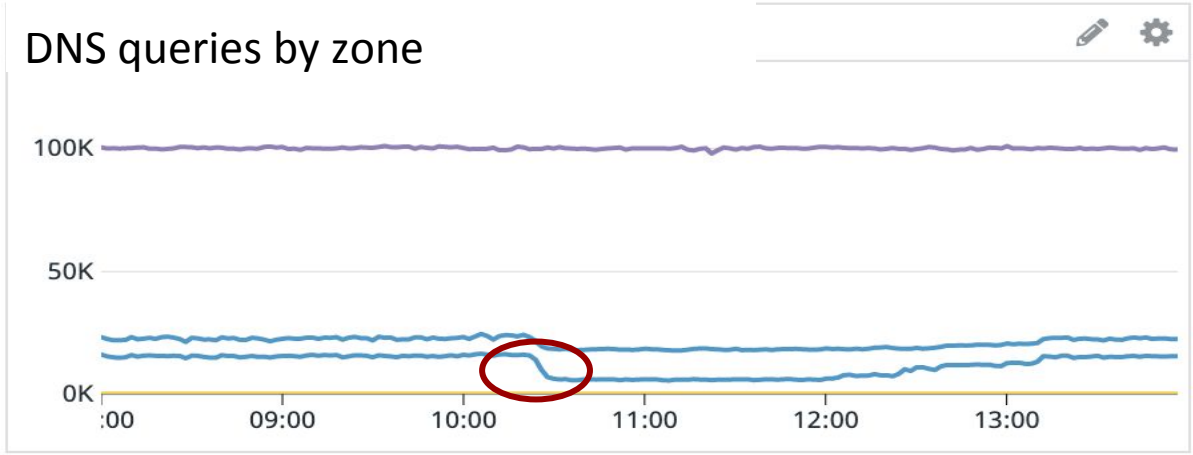




# Upstream DNS issue





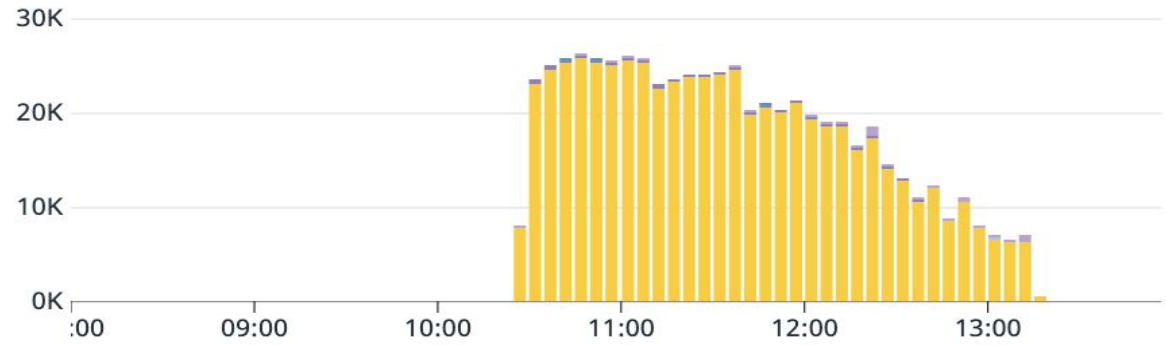




Sharp drop in number of queries for zone "."

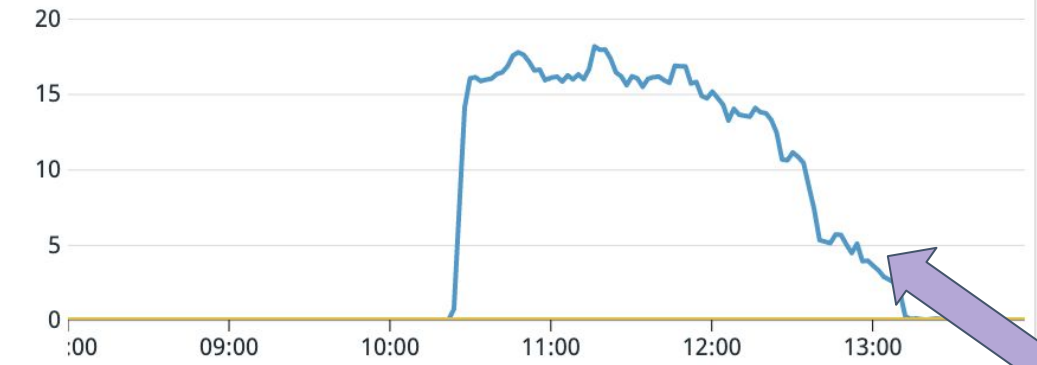
# Upstream DNS issue



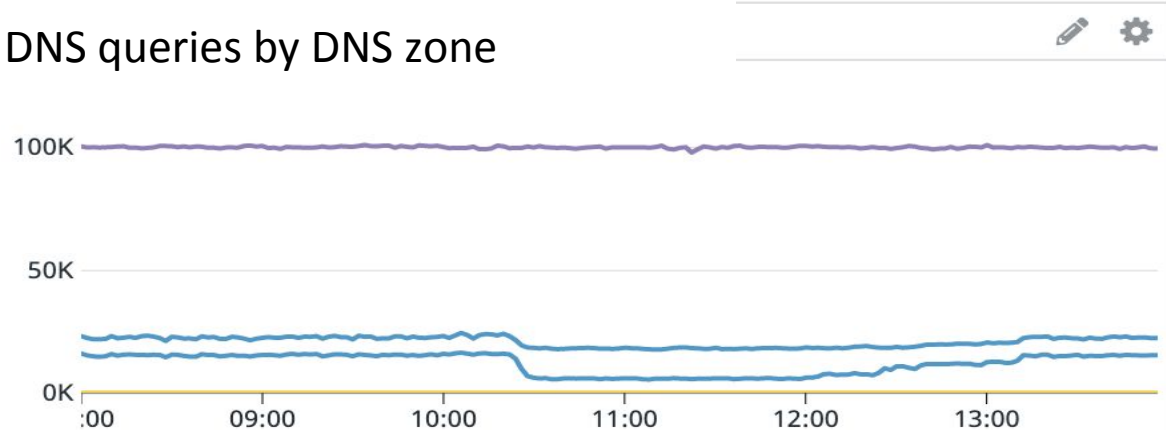
Application resolution errors



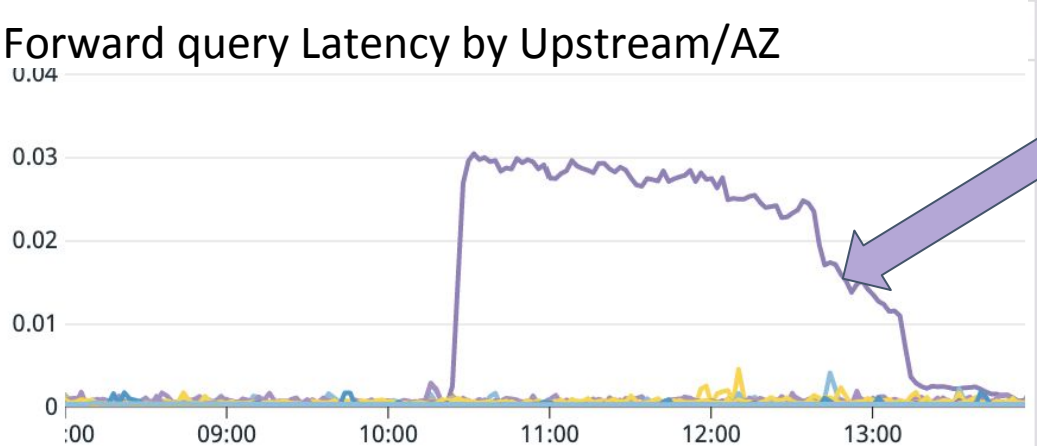

Forward Health-check failures by Upstream/AZ




DNS queries by DNS zone




Forward query Latency by Upstream/AZ




Provider DNS Zone "A"

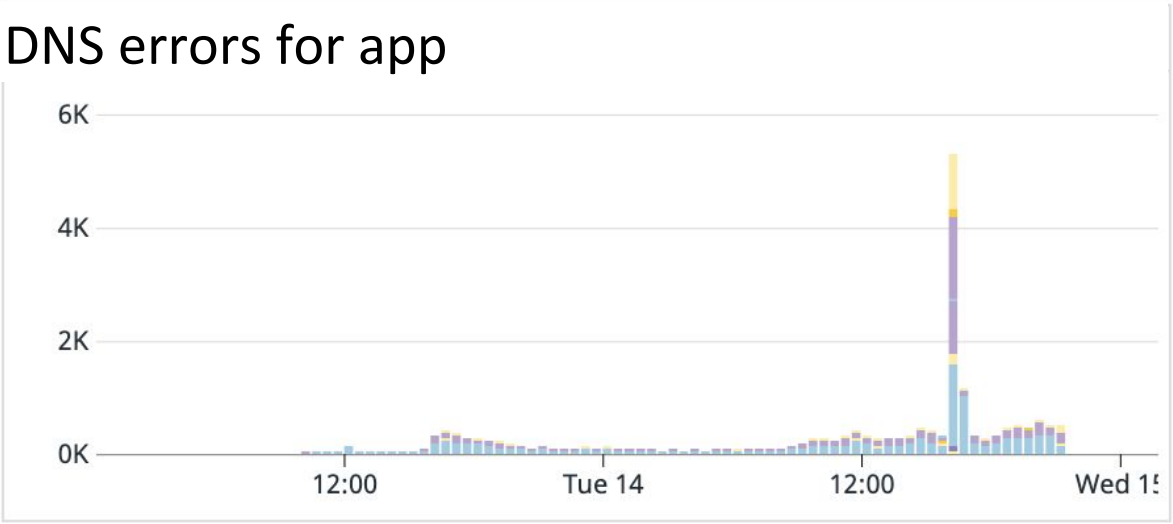
**Upstream resolver issue in a single AZ**


# Sometimes you have to remember pods run on nodes

# Node issue



Familiar symptom: some applications have issues due to DNS errors

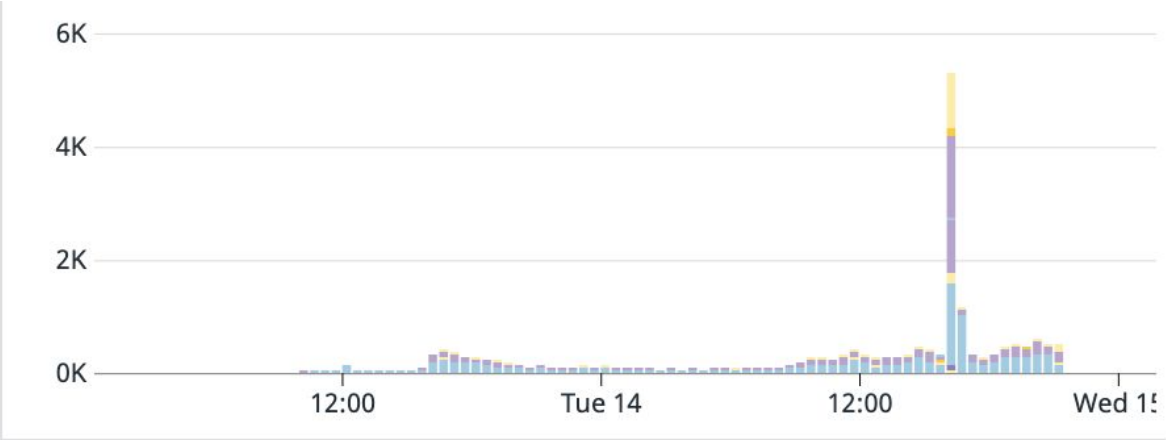


# Node issue

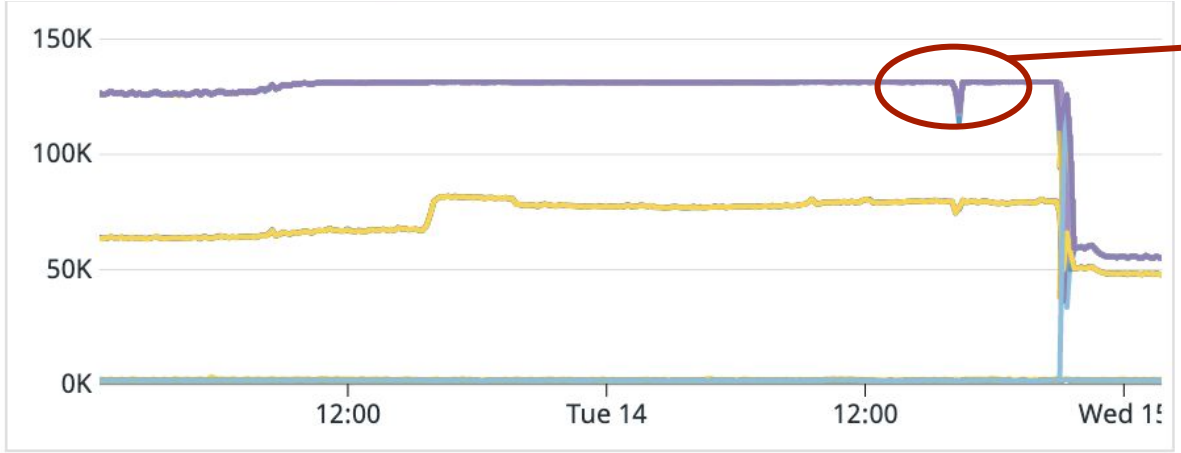


Familiar symptom: some applications have issues due to DNS errors

DNS errors for app




Conntrack entries (hosts running coredns pods)



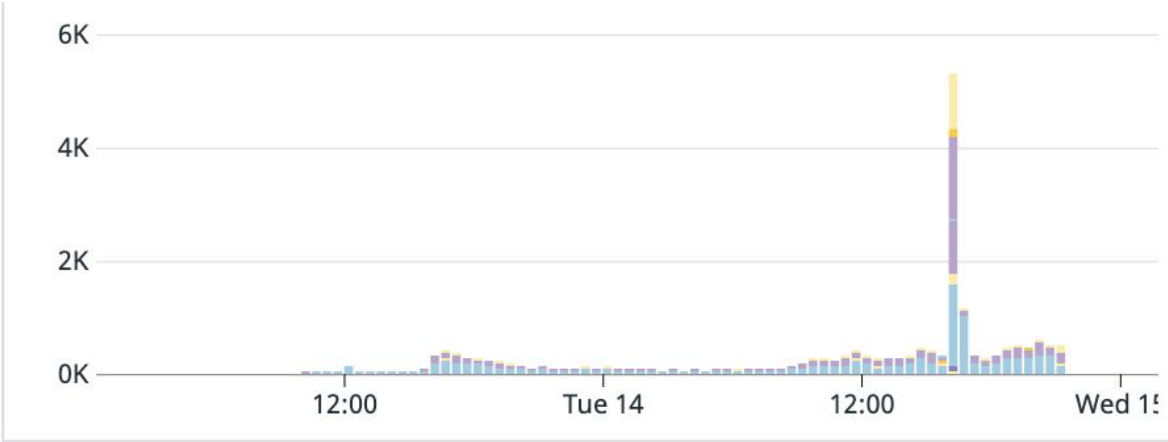

~130k entries

# Node issue

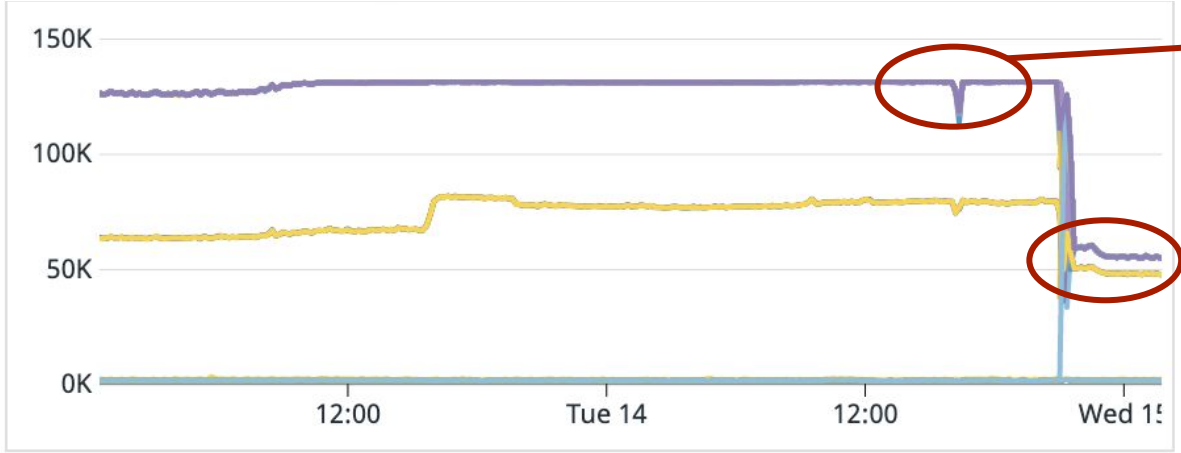


Familiar symptom: some applications have issues due to DNS errors

DNS errors for app




Conntrack entries (hosts running coredns pods)



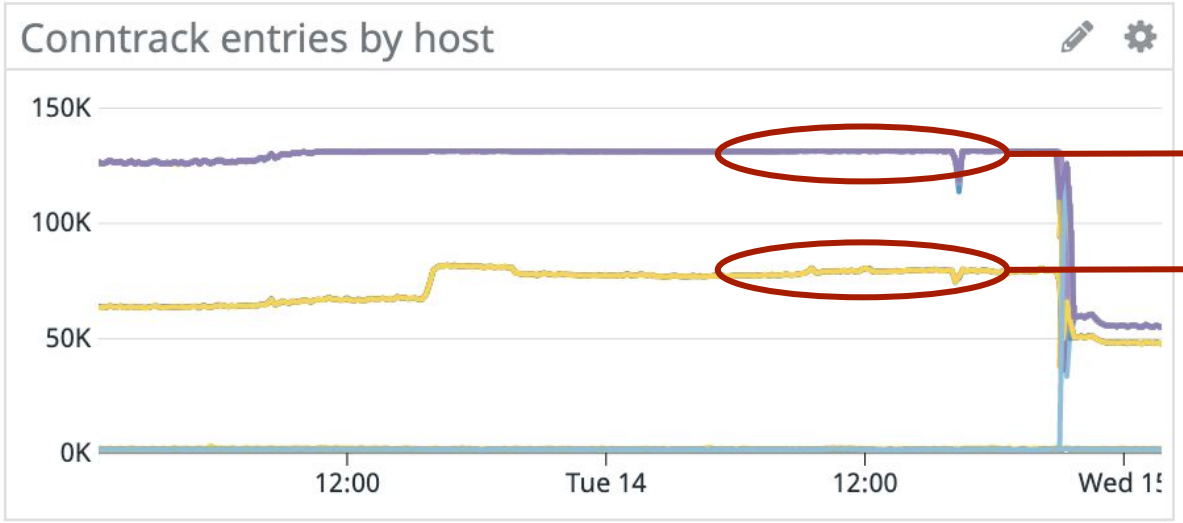

~130k entries

*--conntrack-min => Default: 131072*

Increase number of pods and nodes

# Something weird



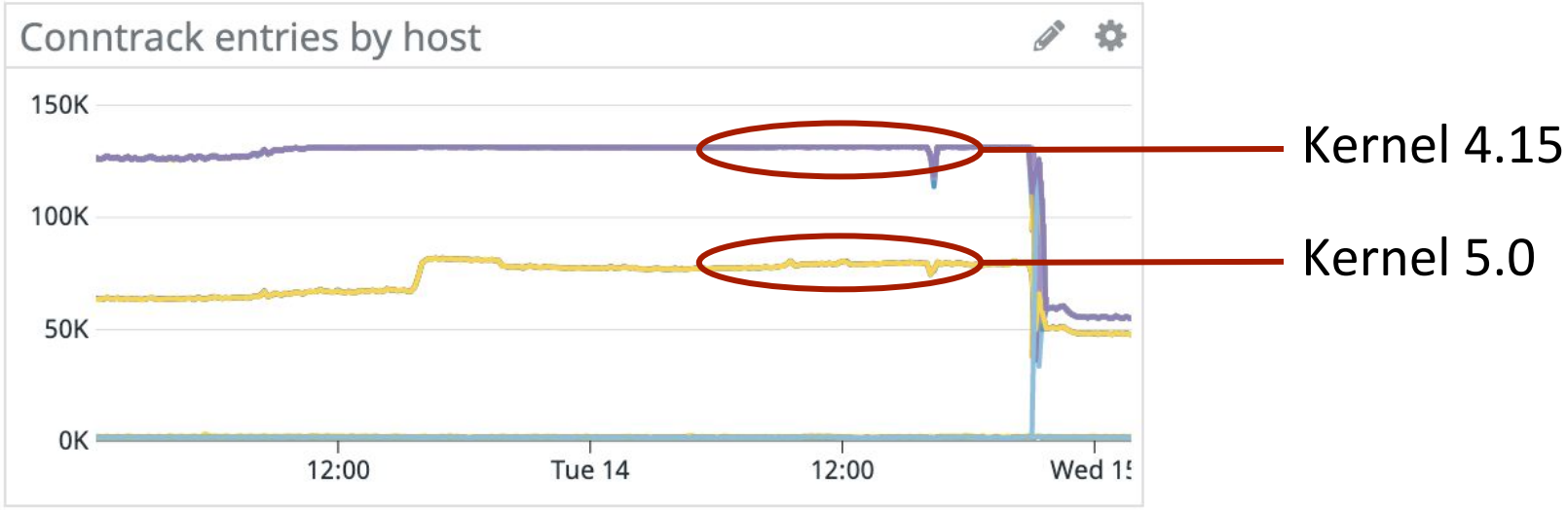



Group of nodes with ~130k entries

Group of nodes with ~60k entries

# Something weird






Kernel patches in 5.0+ to improve conntrack behavior with UDP

**=> conntrack entries for DNS get 30s TTL instead of 180s**

Details

- netfilter: conntrack: udp: only extend timeout to stream mode after 2s
- netfilter: conntrack: udp: set stream timeout to 2 minutes


# Sometimes it's just weird

# DNS is broken for a single app



**Symptom**

pgbouncer can't connect to  postgres after coredns update

But, **everything else works completely fine**

# DNS is broken for a single app

KubeCon CloudNativeCon Europe 2020 Virtual

Let's capture and analyze DNS queries

```
10.143.217.162  41005   1          0      BAcKEnd.DAtAdOG.serViCe.CONsul.DATAdog.svC.clusTEr.LOCaL
10.143.217.162  41005   28         0      BaCKEND.dATaDOg.sErvicE.cONSuL.dataDog.svc.ClUsTER.loCaL
10.129.224.2    53      1    3     0      BAcKEnd.DAtAdOG.serViCe.CONsul.DATAdog.svC.clusTEr.LOCaL
10.129.224.2    53      28   3     0      BaCKEND.dATaDOg.sErvicE.cONSuL.dataDog.svc.ClUsTER.loCaL

[...]


10.143.217.162  41005   1          0      bACKEND.dataDog.SeRvICe.CONsUl
10.143.217.162  41005   28         0      BaCkenD.DatADOg.ServiCE.COnsUL
10.129.224.2    53      1    0     1      bACKEND.dataDog.SeRvICe.CONsUl
```

1: A
28: AAA

3: NXDOMAIN
0: NOERROR

Queried domain. **Random case??**
IETF Draft to increase DNS Security
"Use of Bit 0x20 in DNS Labels to Improve Transaction Identity"

Source IP/Port
Same source port across all queries??

**This DNS client is clearly not one we know about**

@lbernail

# DNS is broken for a single app



Let's capture and analyze DNS queries

```
10.143.217.162  41005   1           0     BAcKEnd.DAtAdOG.serViCe.CONsul.DATAdog.svC.clusTEr.LOCaL
10.143.217.162  41005   28          0     BaCKEND.dATaDOg.sErvicE.cONSuL.dataDog.svc.ClUsTER.loCaL
10.129.224.2    53      1       3   0     BAcKEnd.DAtAdOG.serViCe.CONsul.DATAdog.svC.clusTEr.LOCaL
10.129.224.2    53      28      3   0     BaCKEND.dATaDOg.sErvicE.cONSuL.dataDog.svc.ClUsTER.loCaL

[...]


10.143.217.162  41005   1           0     bACKEND.dataDog.SeRvICe.CONsUl
10.143.217.162  41005   28          0     BaCkenD.DatADOg.ServiCE.COnsUL
10.129.224.2    53      1       0   1     bACKEND.dataDog.SeRvICe.CONsUl
```

Truncate Bit (TC)

pgbouncer compiled with evdns, which doesn't support TCP upgrade (and just ignores the answer if TC=1)
Previous coredns version was not setting the TC bit when upstream TCP answer was too large (bug was fixed)

Recompiling pgbouncer with c-ares fixed the problem


# Sometimes it's not DNS


# Sometimes it's not DNS
# Rarely

# Sometimes it's not DNS



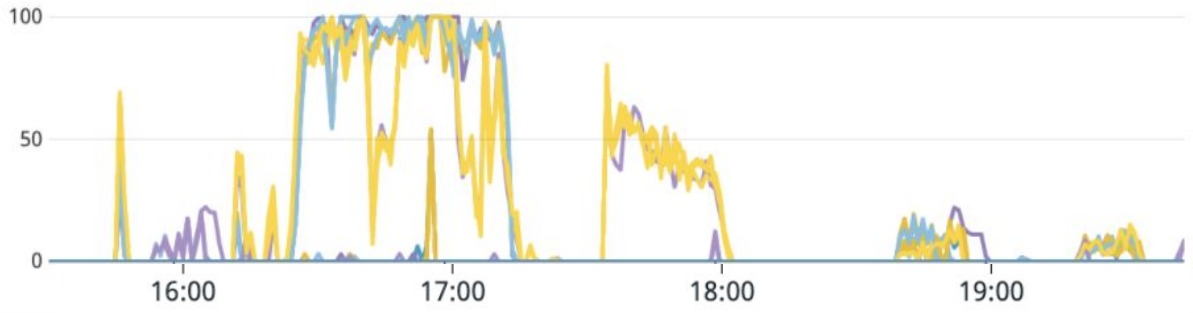
% of Errors in app


Logs are full of DNS errors

# Sometimes it's not DNS

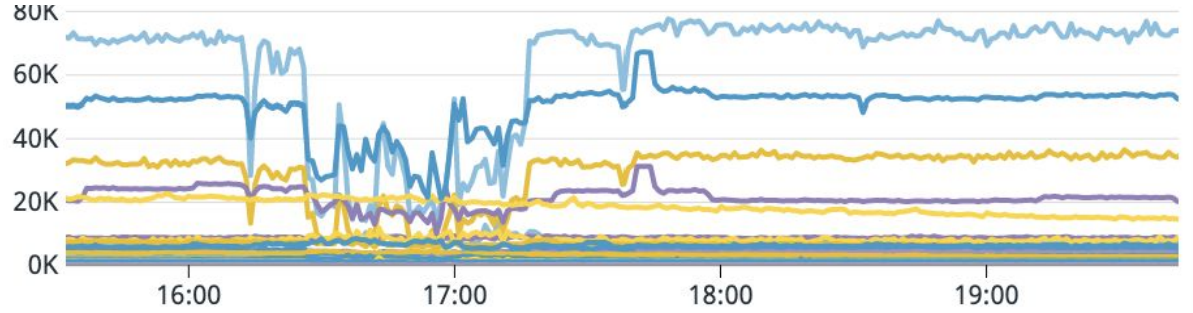


% of Errors in app




Logs are full of DNS errors

Average #Packets received by nodegroups




Sharp drop in traffic received by nodes
??

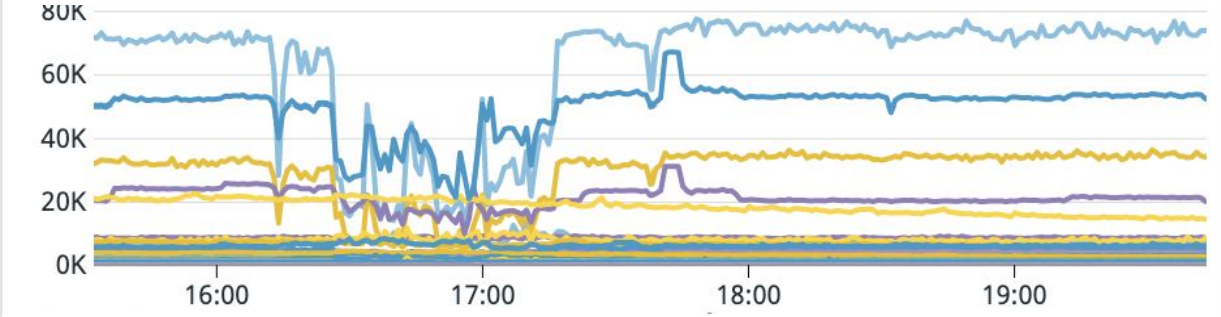
# Sometimes it's not DNS



% of Errors in app

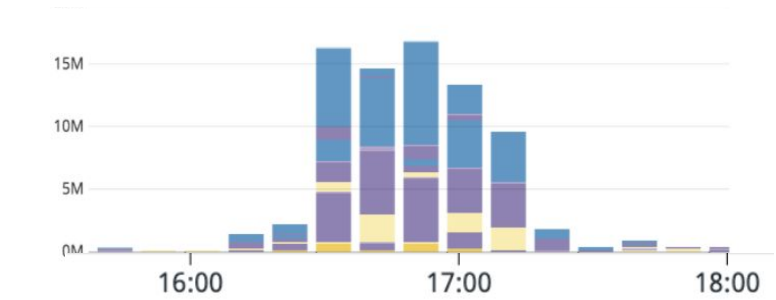

Logs are full of DNS errors

Average #Packets received by nodegroups




Sharp drop in traffic received by nodes
??

TCP Retransmits by AZ-pairs

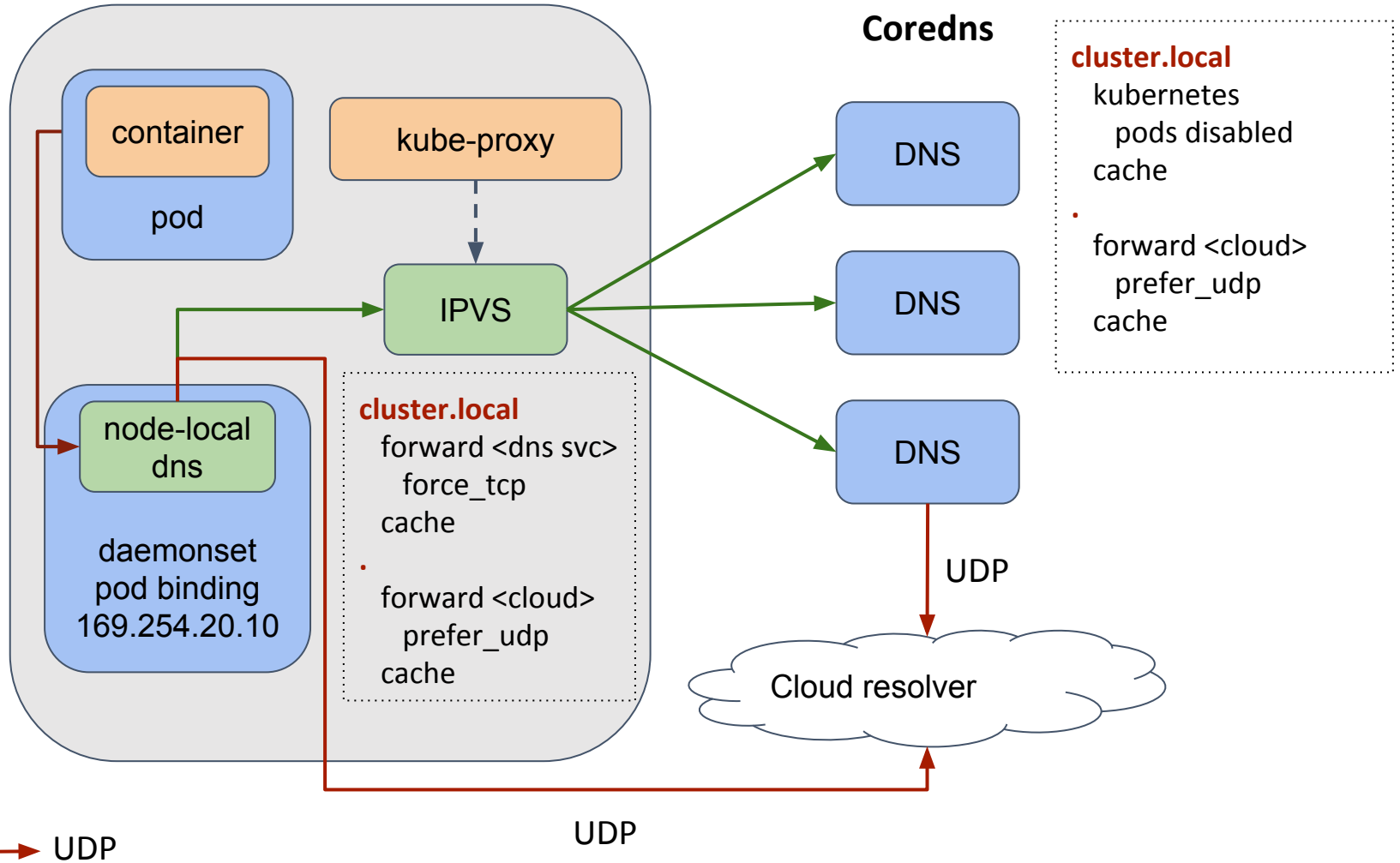



High proportion of drops for any traffic involving zone "b"
Confirmed transient issue from provider

**Not really DNS that time, but this was the first impact**

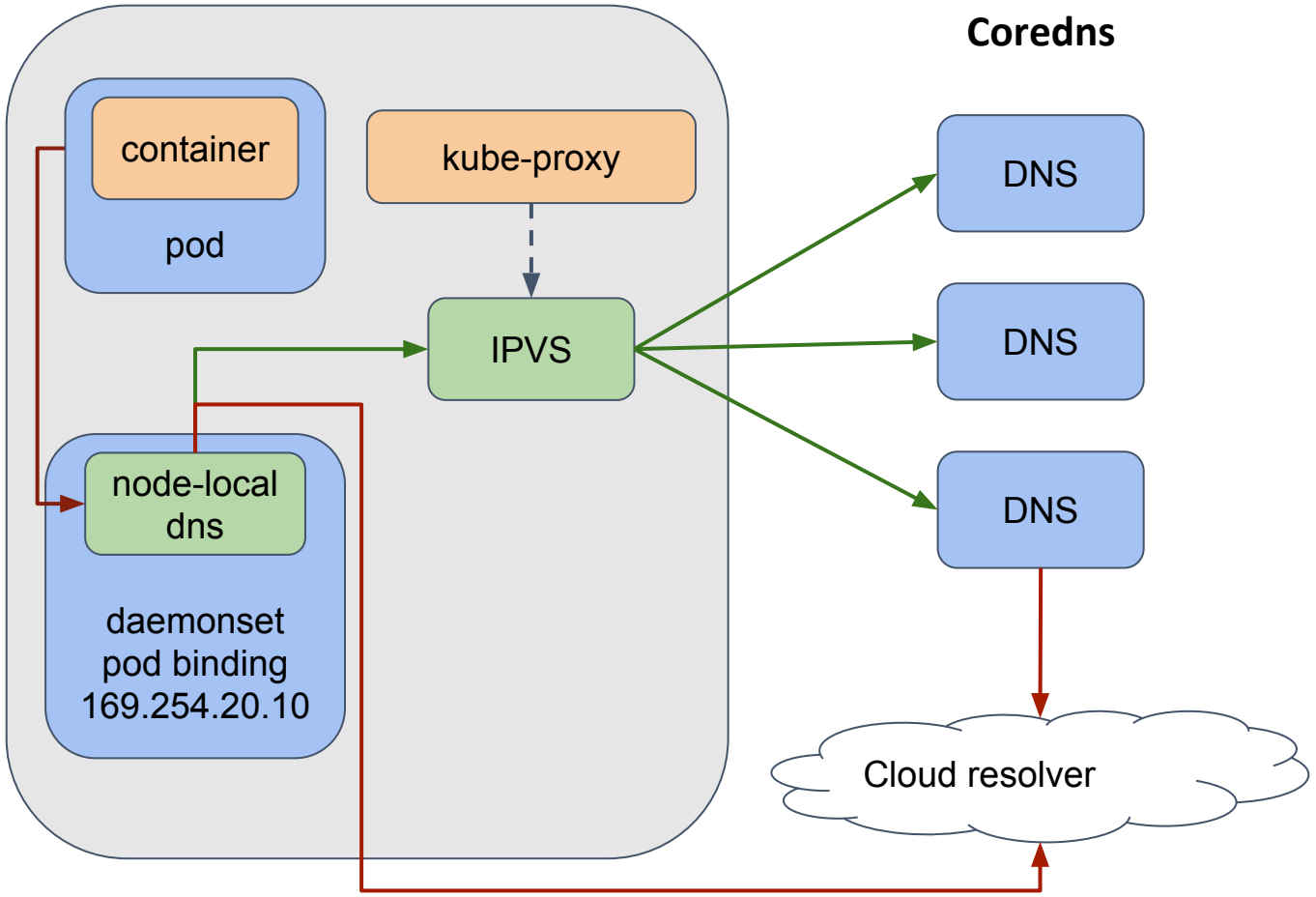

# What we run now

# Our DNS setup



# Our DNS setup






**Container /etc/resolv.conf**

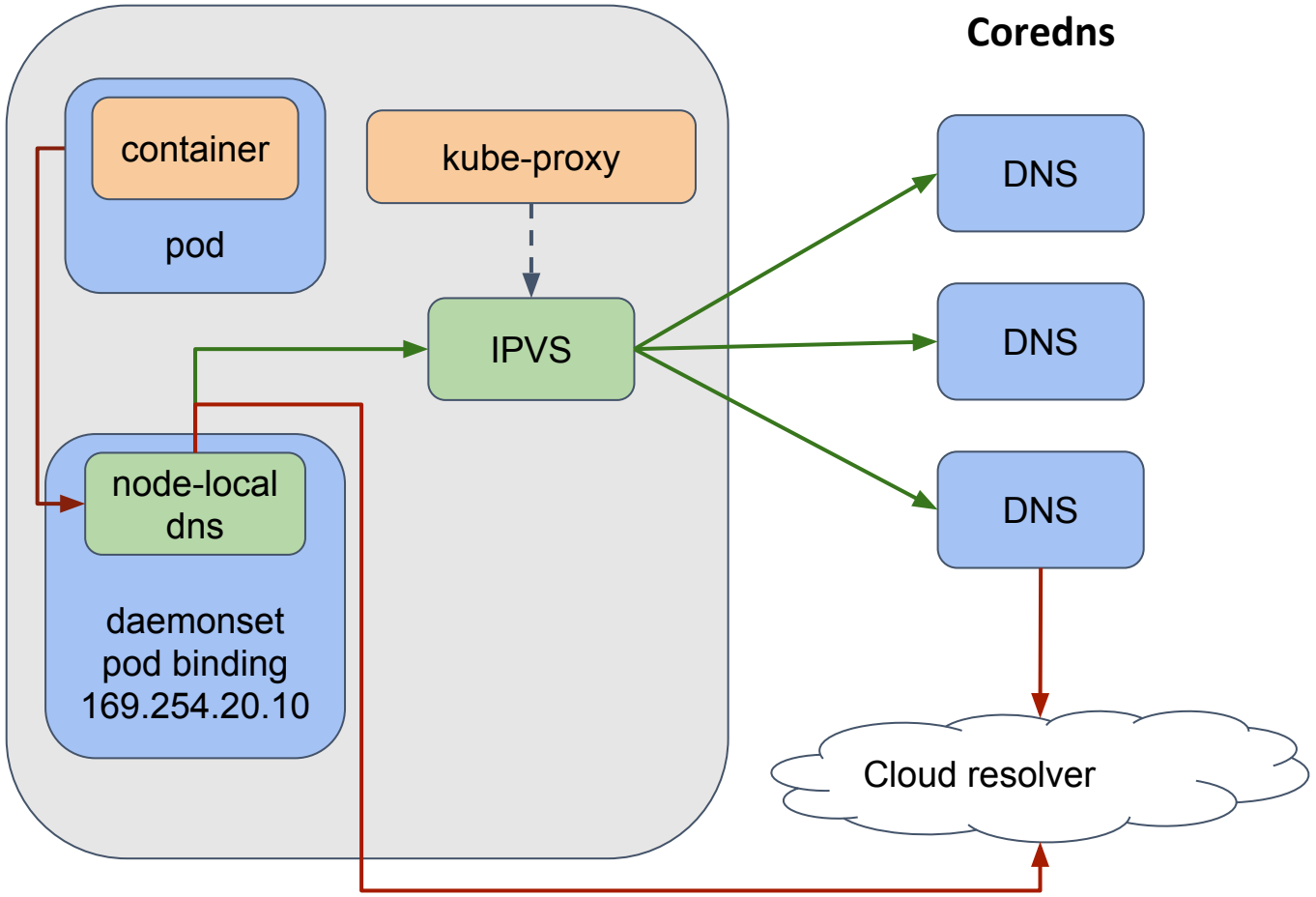
search <namespace>.svc.cluster.local
svc.cluster.local
cluster.local
ec2.internal

nameserver **169.254.20.10**
**<dns svc>**

options ndots:5, **timeout: 1**

# Our DNS setup






**Container /etc/resolv.conf**

search <namespace>.svc.cluster.local
svc.cluster.local
cluster.local
ec2.internal

nameserver **169.254.20.10**
**<dns svc>**

options ndots:5, **timeout: 1**

**Alternate /etc/resolv.conf**
Opt-in using **annotations** (mutating webhook)

search **svc.cluster.local**

nameserver 169.254.20.10
<dns svc>

options **ndots:2**, timeout: 1


# Conclusion

# Conclusion



- Running Kubernetes means running DNS
- DNS is hard, especially at scale

- Recommendations
  - Use node-local-dns and cache everywhere you can
  - Test your DNS infrastructure (load-tests, rolling updates)
  - Understand the upstream DNS services you depend on

- For your applications
  - Try to standardize on a few resolvers
  - Use async resolution/long-lived connections whenever possible
  - **Include DNS failures in your (Chaos) tests**


# Thank you

We're hiring!
https://www.datadoghq.com/careers/

laurent@datadoghq.com
@lbernail


KubeCon
CloudNativeCon
Europe 2020
Virtual
HELM
KEEP CLOUD NATIVE
CONNECTED