Logs Told Us It Was DNS
It Felt Like DNS
It Had To Be DNS
It Wasn't DNS

Elijah Andrews, Hemanth Malla

2023-03-21, SREcon23 Americas, Santa Clara

DATADOG

Who are we?

Elijah Andrews

Software Engineer

Datadog

🐦 **@elijahca**

Hemanth Malla

Software Engineer

Datadog

🐦 **@hemanthmalla**

Datadog

Over 600 integrations
Over 5,000 employees
Over 23,000 customers
Runs on millions of hosts
Tens of trillions of events per day

Tens of thousands of nodes
Hundreds of thousands of pods
100s of k8s clusters with 100-4000 nodes
Multi-cloud
Very fast growth

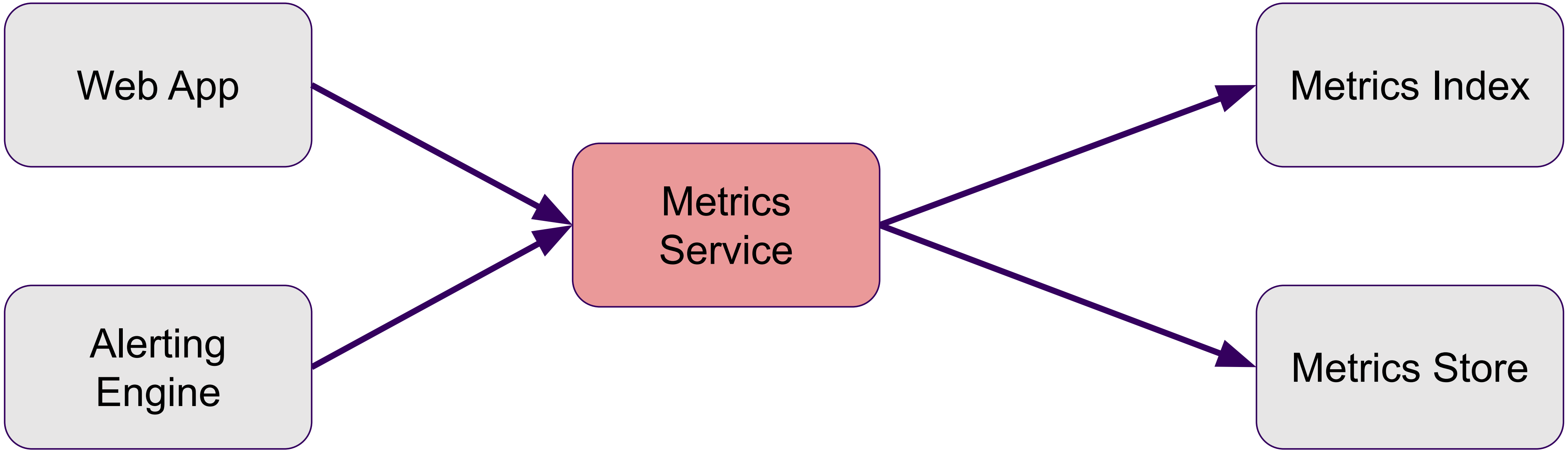
DATADOG

Kubernetes primer

- *pods* are scheduled on *nodes*
- applications run in pods
- each pod has a unique IP address
- *Cilium* is our Container Network Interface (CNI)
- Cilium glues node, pod, and AWS networking together

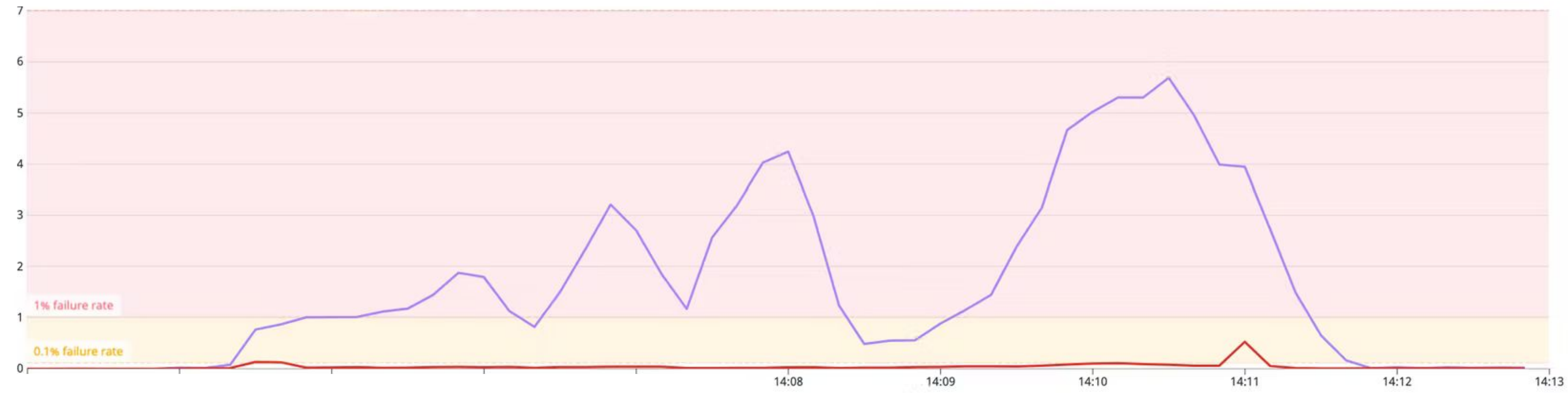
How it all started

Applications involved

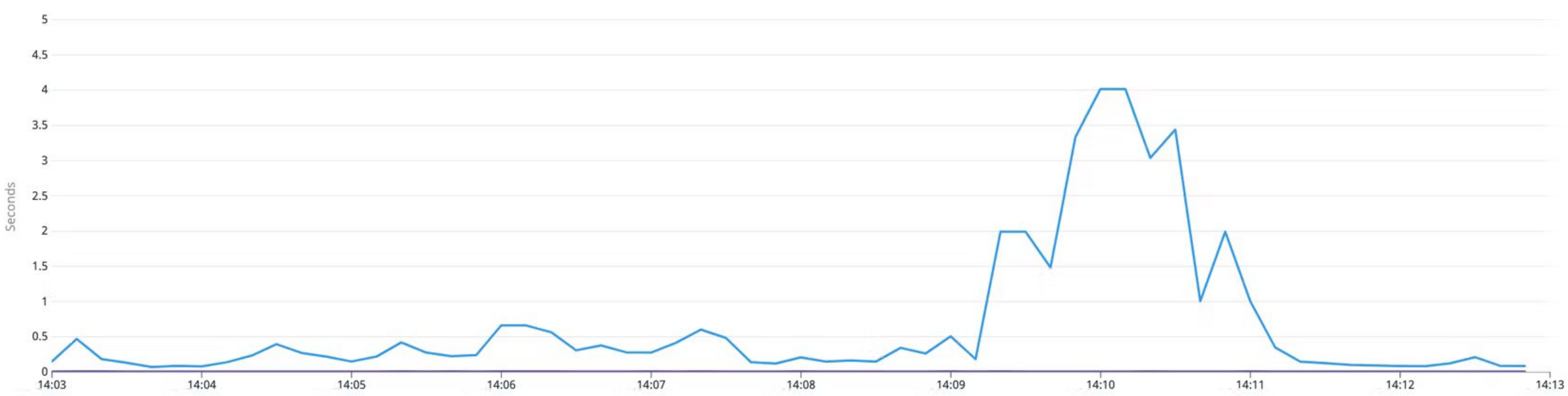


Metrics service errors during rollouts

Error rate (**server** / **client**)



Latency (p90 and p99)



It's always DNS

Logs told us it was DNS

It's always DNS

Logs told us it was DNS

It looked like DNS

It's always DNS

Logs told us it was DNS

It looked like DNS

It had to be DNS

It's always DNS

Logs told us it was DNS

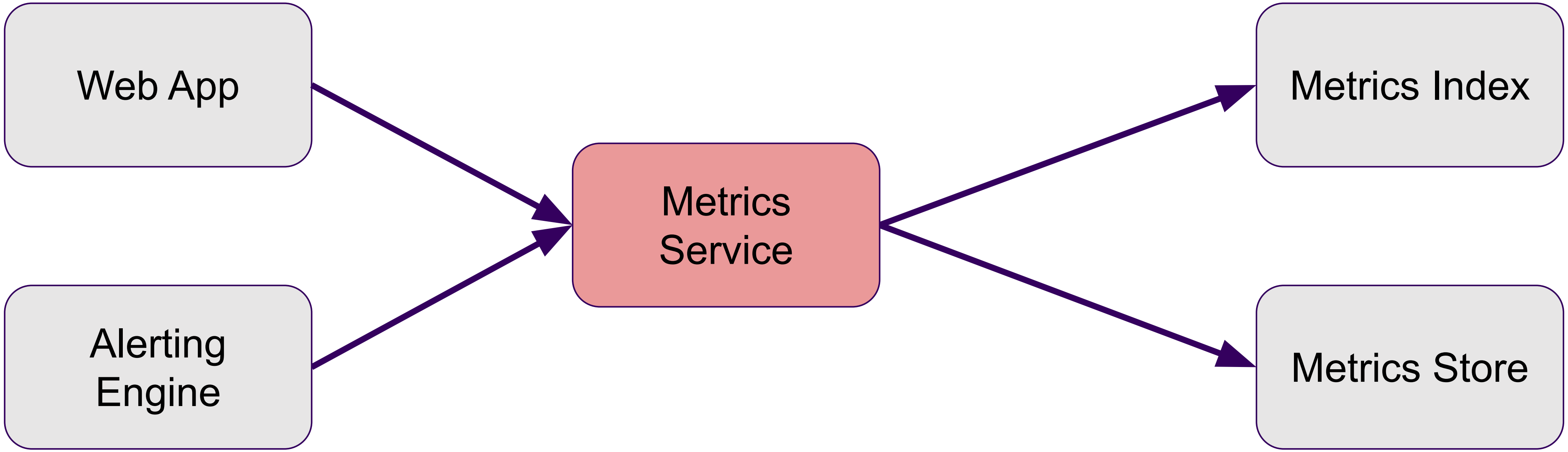
It looked like DNS

It had to be DNS

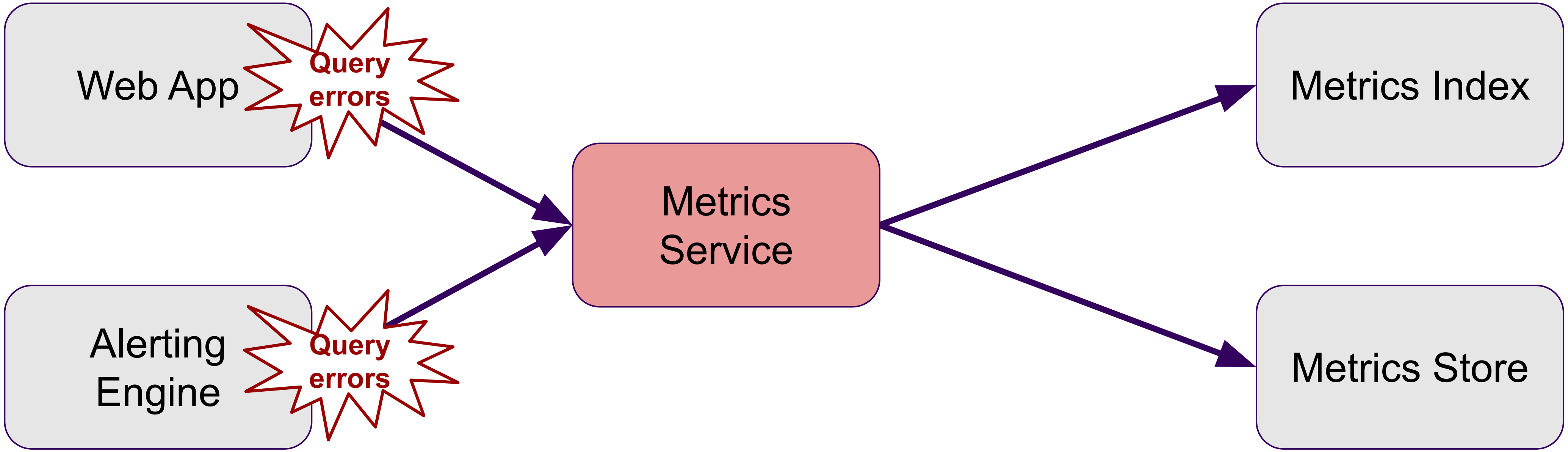
Right?

Chapter 1: DNS

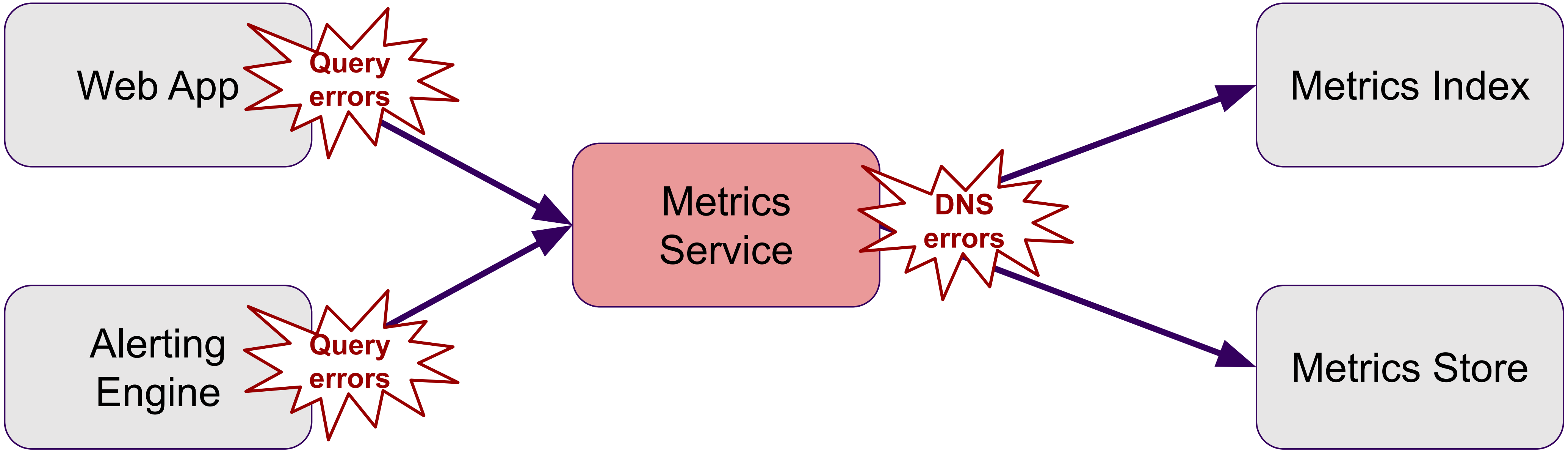
Applications involved



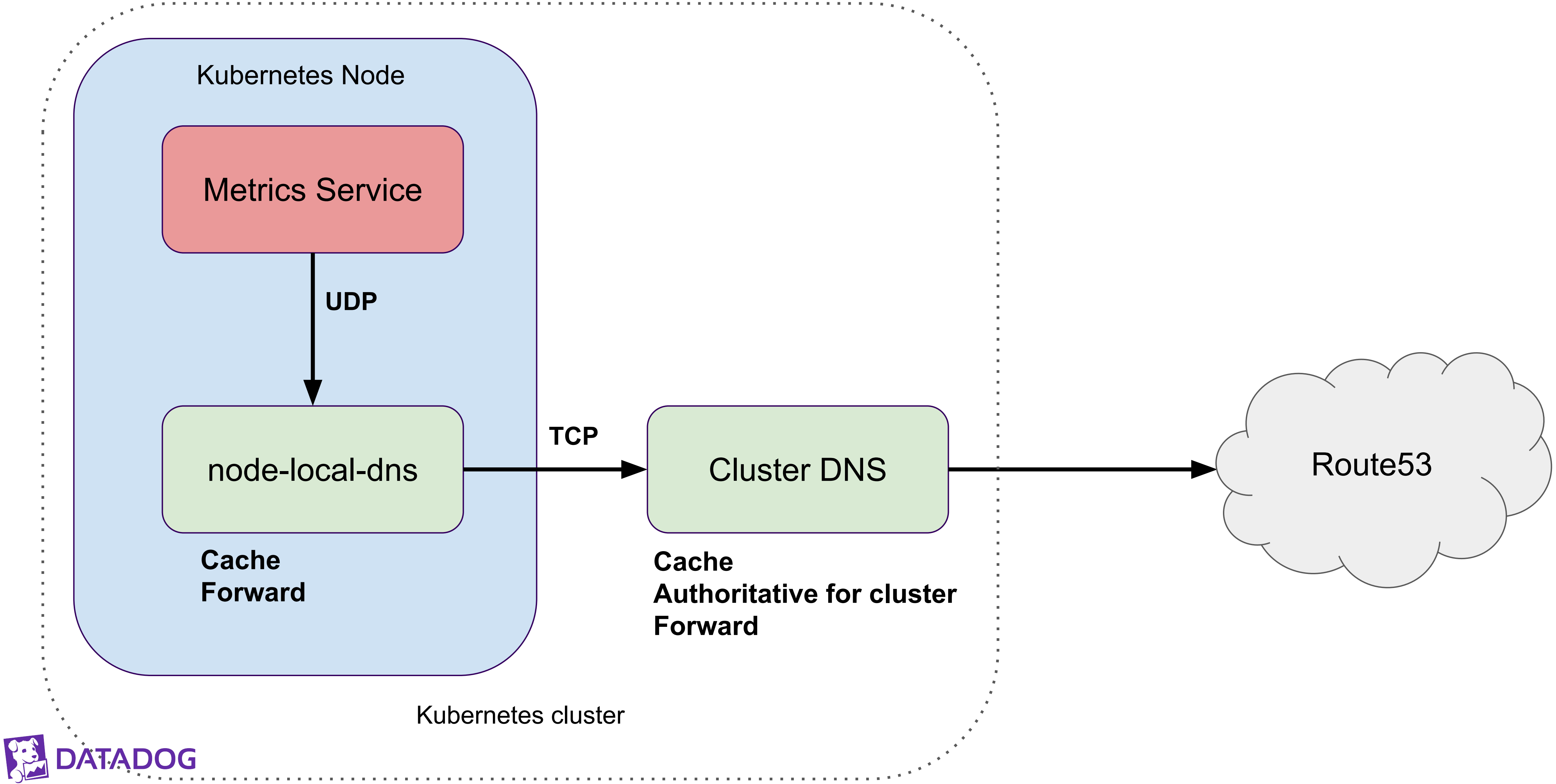
Applications involved



Applications involved



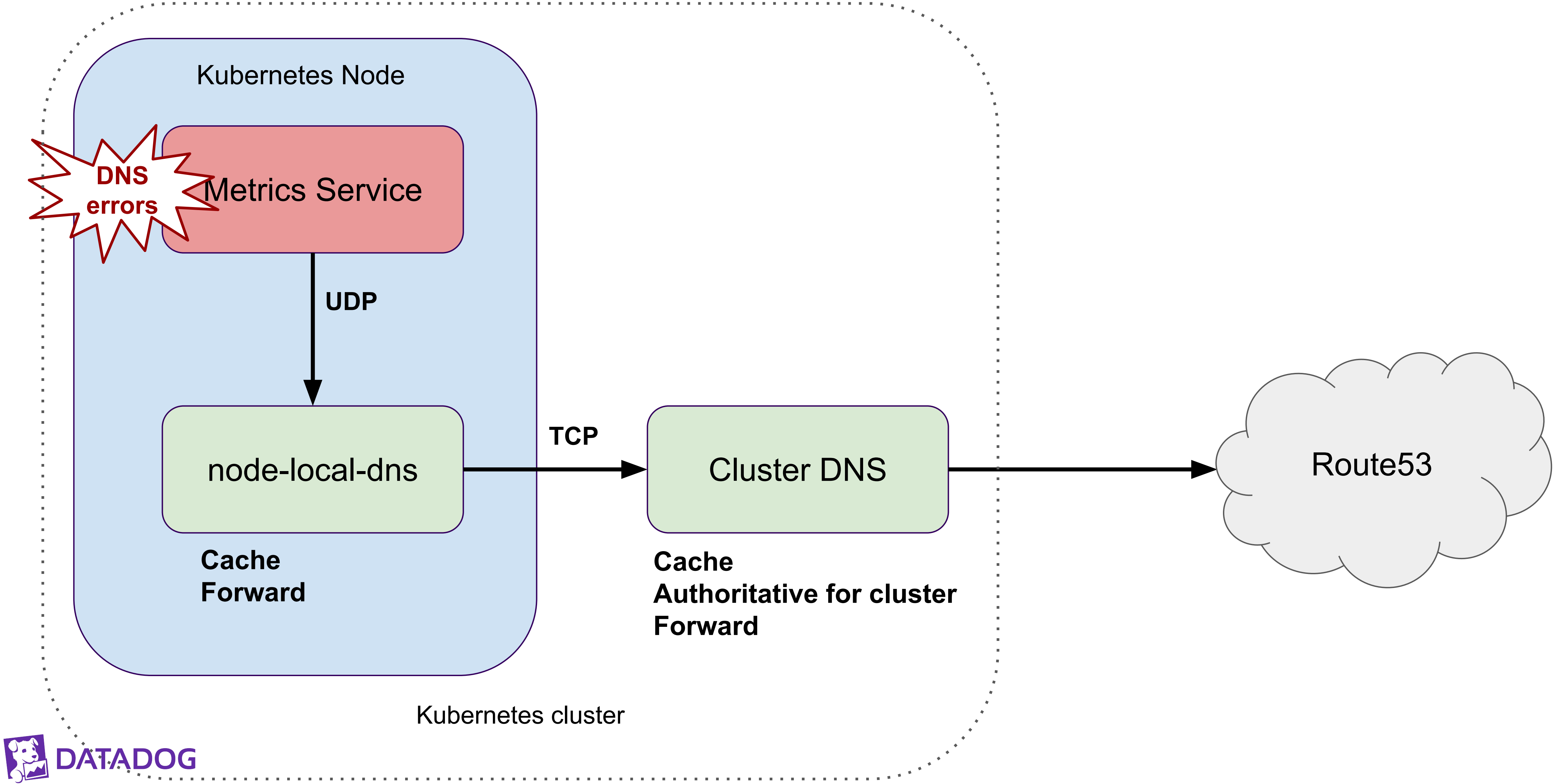
DNS setup



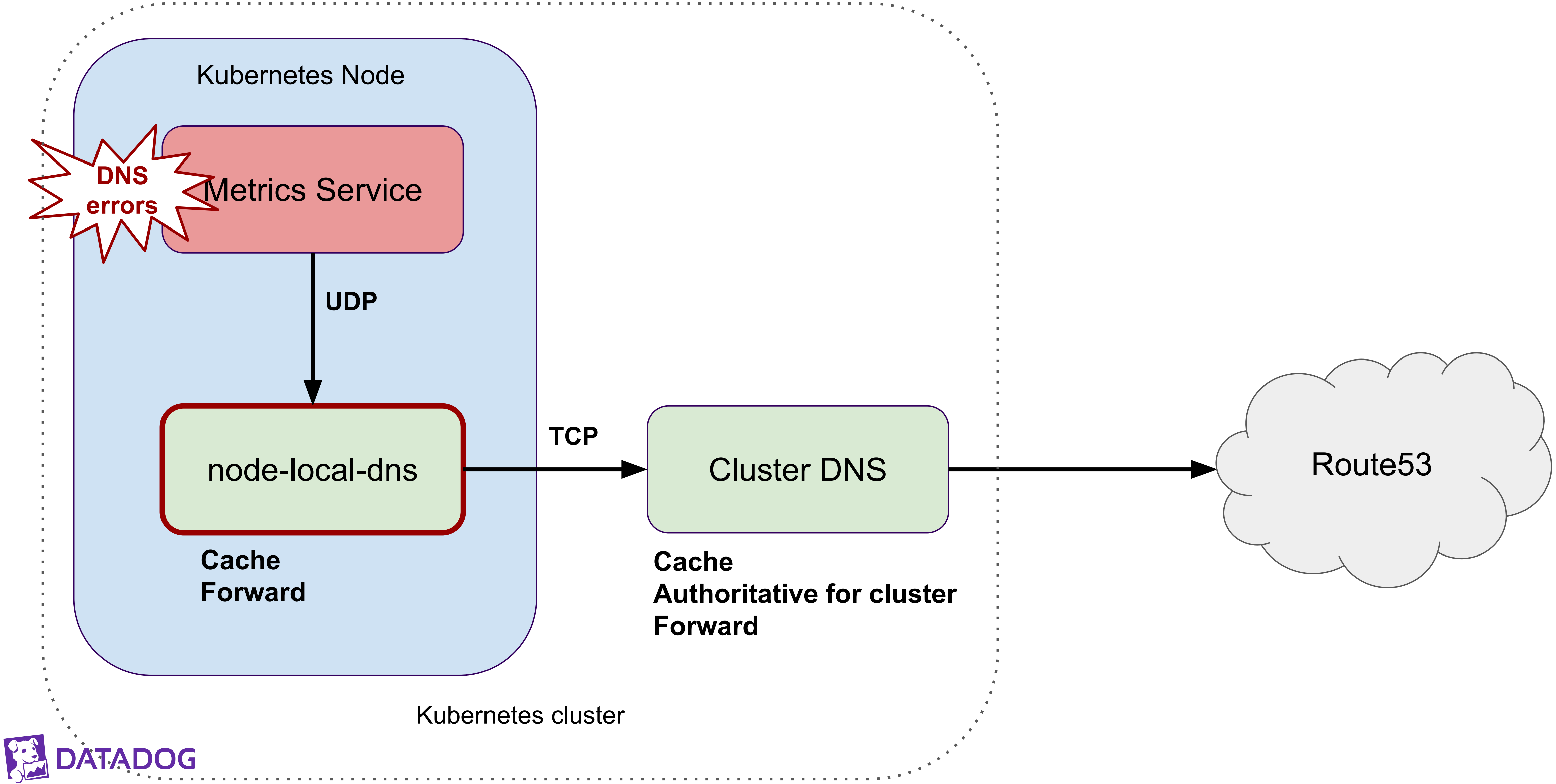
Cache
Forward

Cache
Authoritative for cluster
Forward

DNS setup


Kubernetes Node
DNS errors
Metrics Service
UDP
node-local-dns
Cache
Forward
TCP
Cluster DNS
Cache
Authoritative for cluster
Forward
Route53
Kubernetes cluster

DNS setup



Kubernetes Node

DNS errors

Metrics Service

UDP

node-local-dns

Cache
Forward

TCP

Cluster DNS

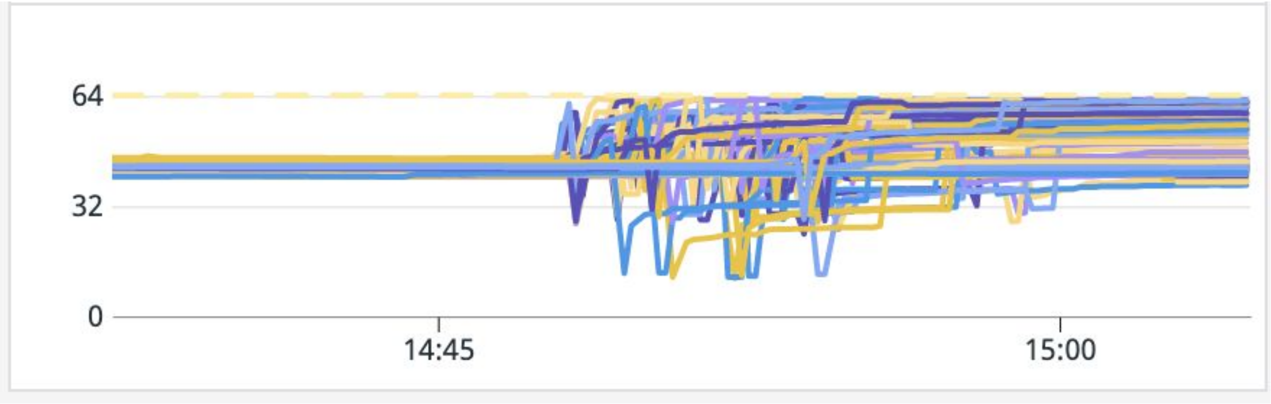
Cache
Authoritative for cluster
Forward

Route53

Kubernetes cluster

Node-Local-DNS (NLD)

NLD Memory per pod on Metrics Service hosts (and limit)



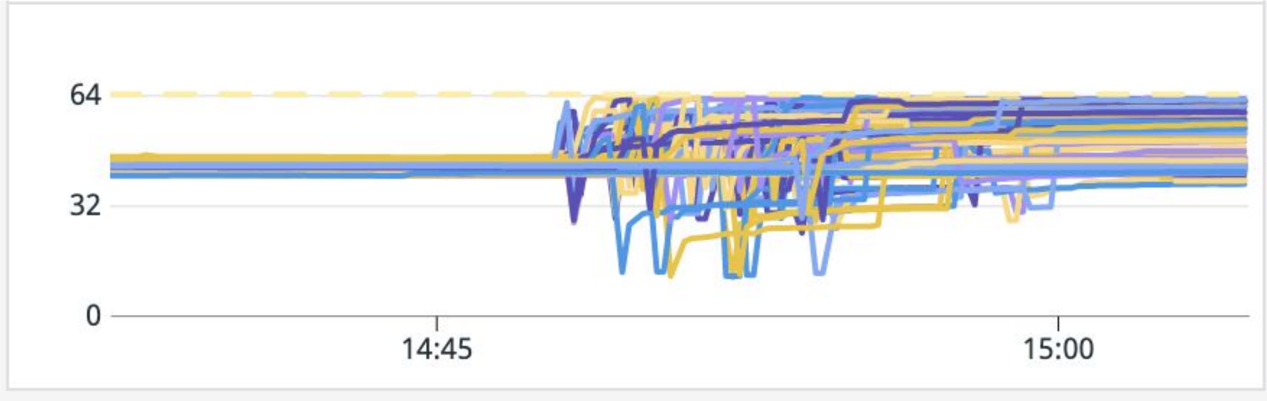
ran out of memory (OOM killed)

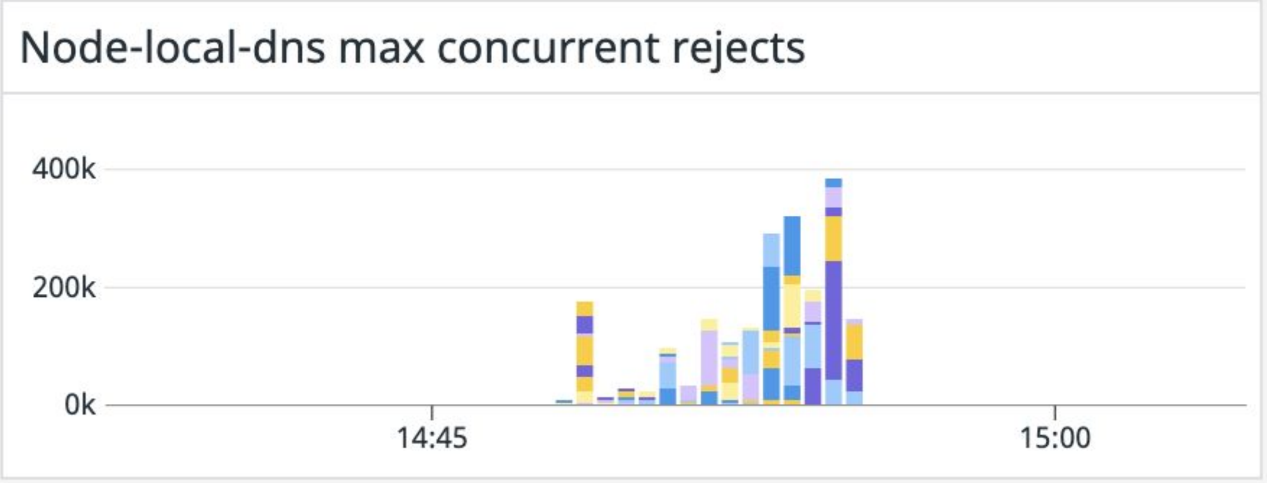
during rollouts

Should *never* happen

Node-Local-DNS (NLD)

NLD Memory per pod on Metrics Service hosts (and limit)

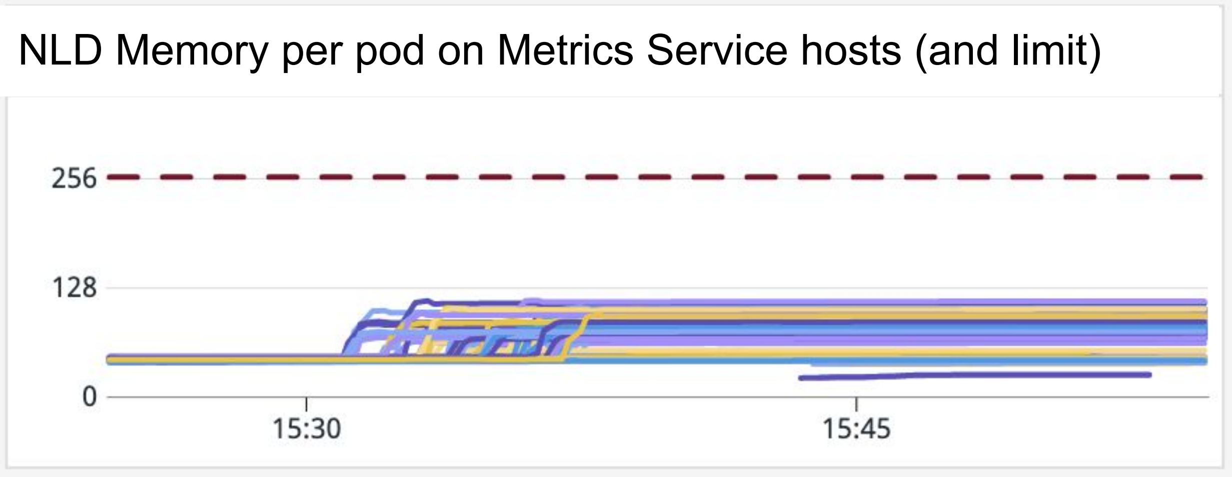



Node-local-dns max concurrent rejects

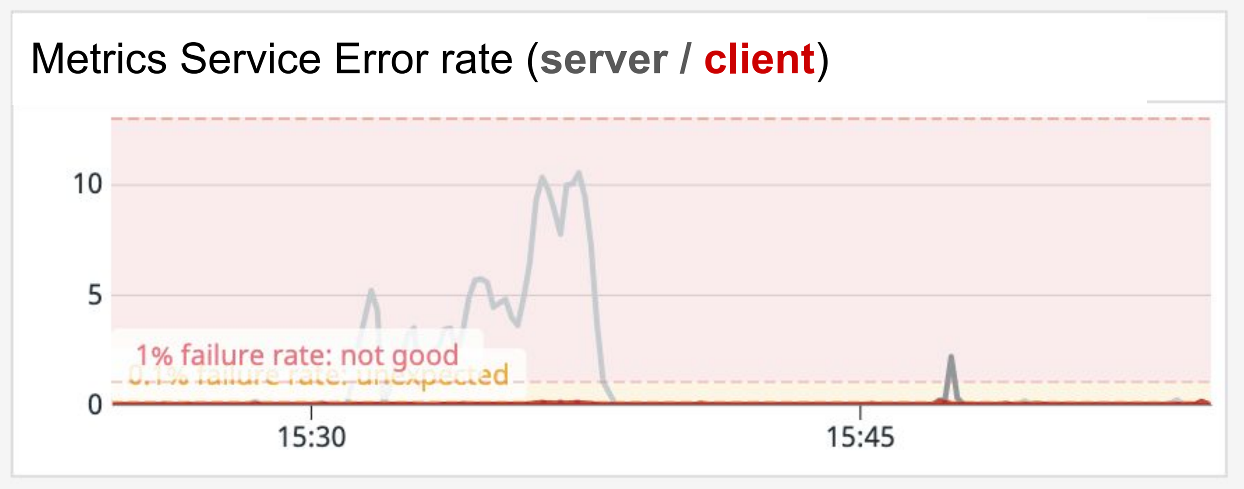
max_concurrent is working

Sizing is wrong

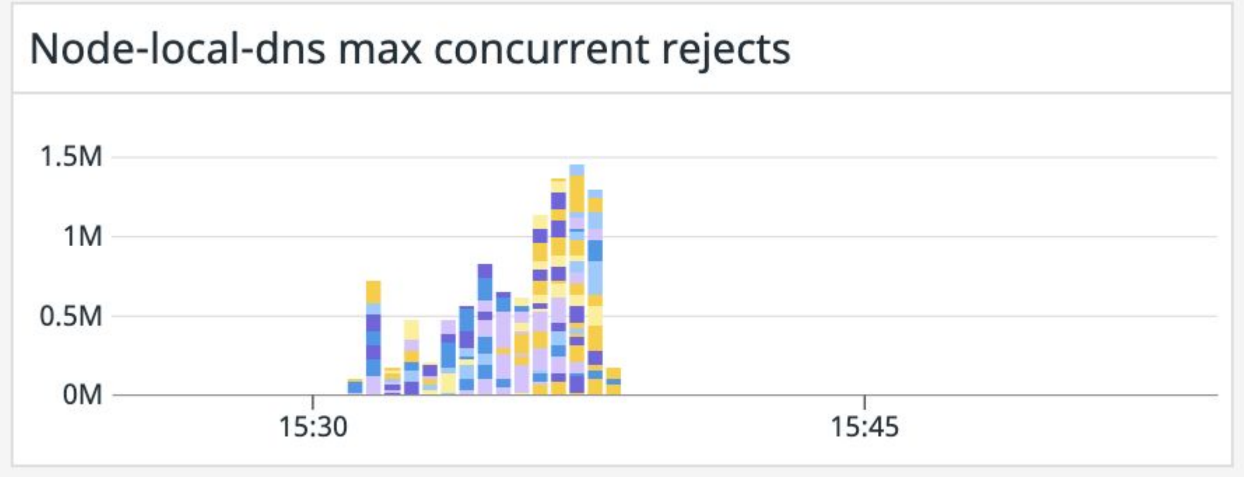
Node-local-dns, 64MB => 256MB



NLD Memory per pod on Metrics Service hosts (and limit)



Metrics Service Error rate (**server** / **client**)

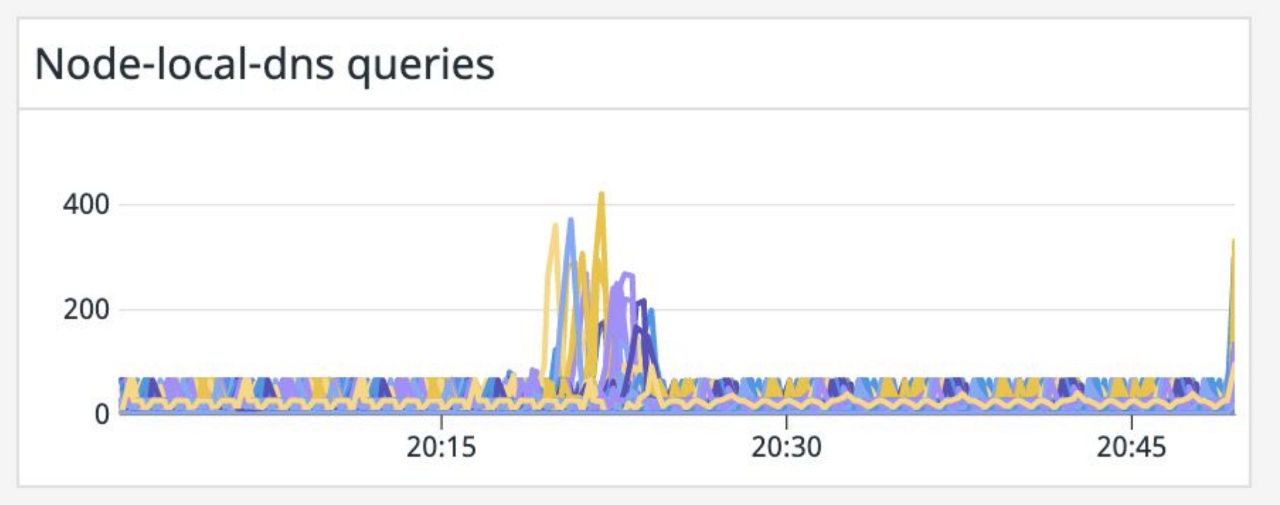


Node-local-dns max concurrent rejects

No more OOM-kills

But not any better for Metrics Service

Too many queries at startup?



Max_concurrent: **1000**

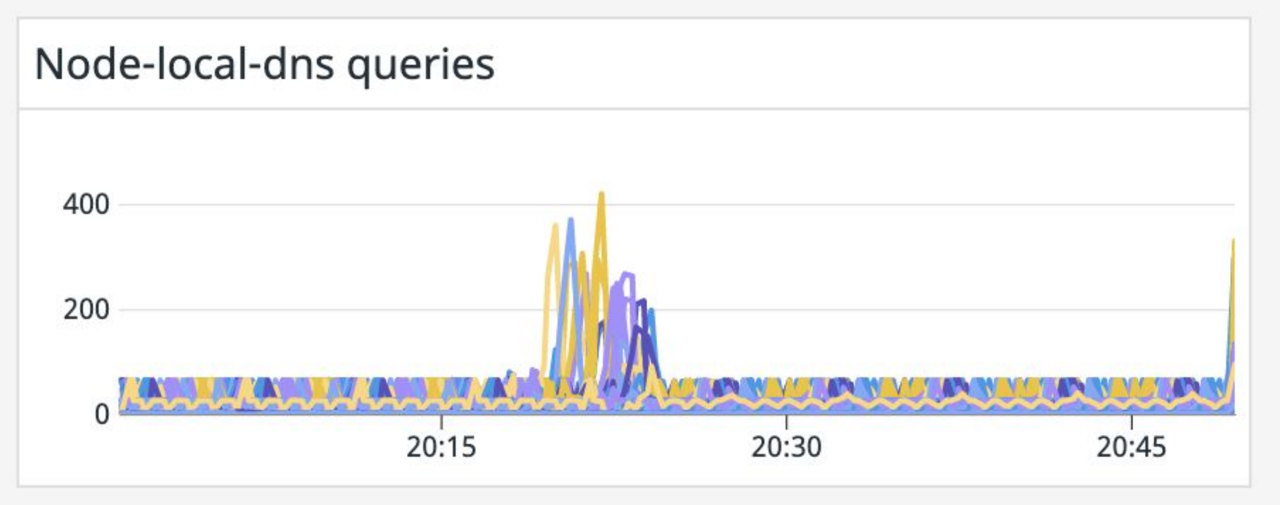
Upstream queries: ~5ms

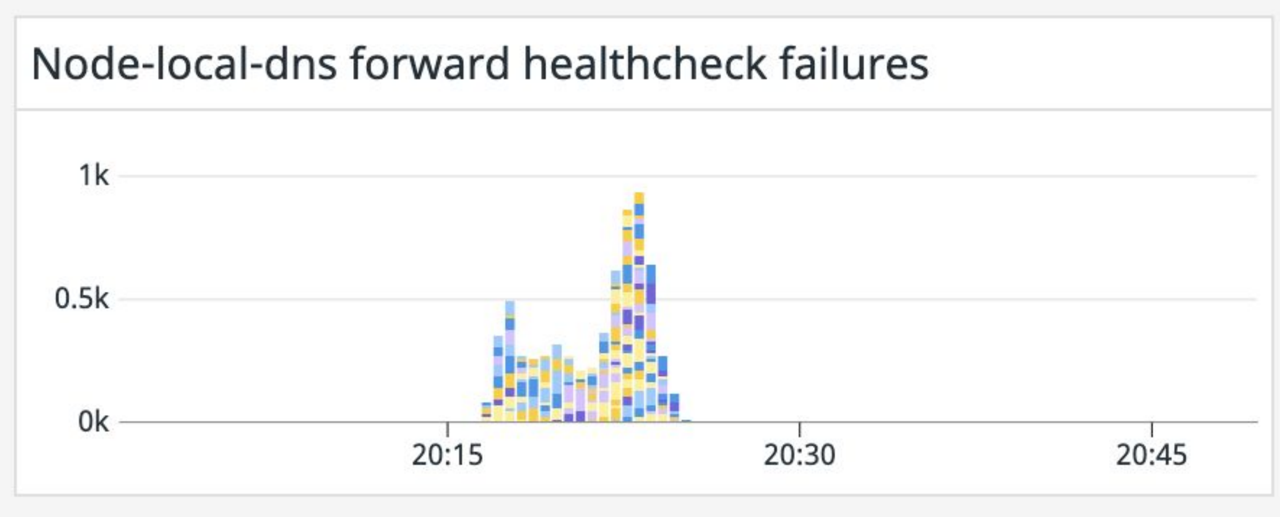
=> NLD should do **> 200k rps**

=> with **<400 rps** we hit max_concurrent

What's happening?

Too many queries at startup?





Upstream marked unhealthy

Upstream is TCP

Connections are reused

but expire=10s

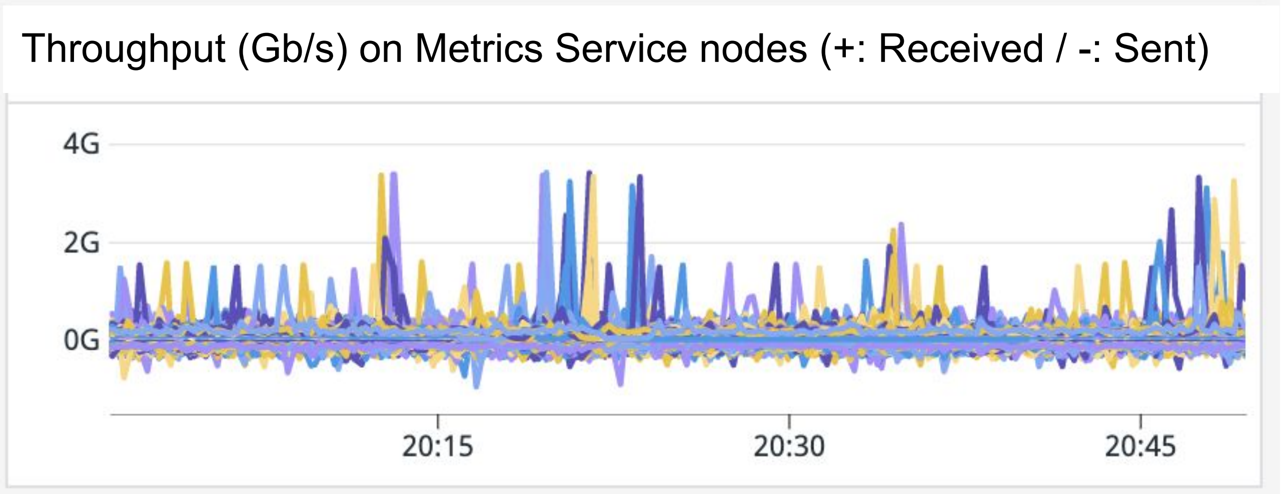
NLD can't create connections?

Why we hit max_concurrent

- NLD can't establish connections to upstreams
- The Forward plugin has a 5s timeout by default
- Incoming queries occupy a query slot for 5s

=> We hit max_concurrent=1000 with only 200rps

Networking issues?



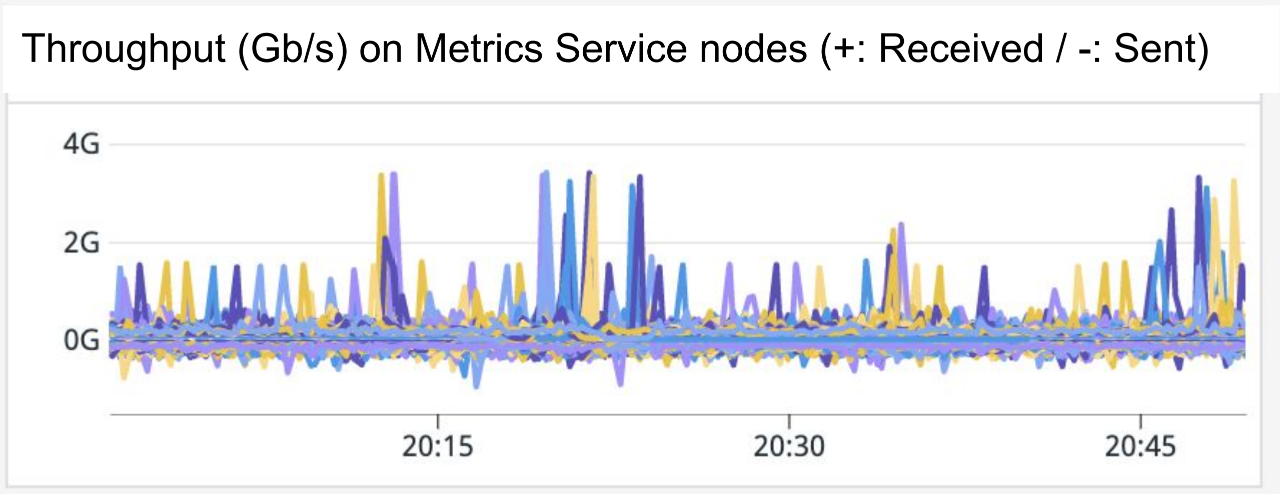
m5.4xlarge

Max: 10Gb/s

Sustained: 5Gb/s

=> looks ok

Networking issues?

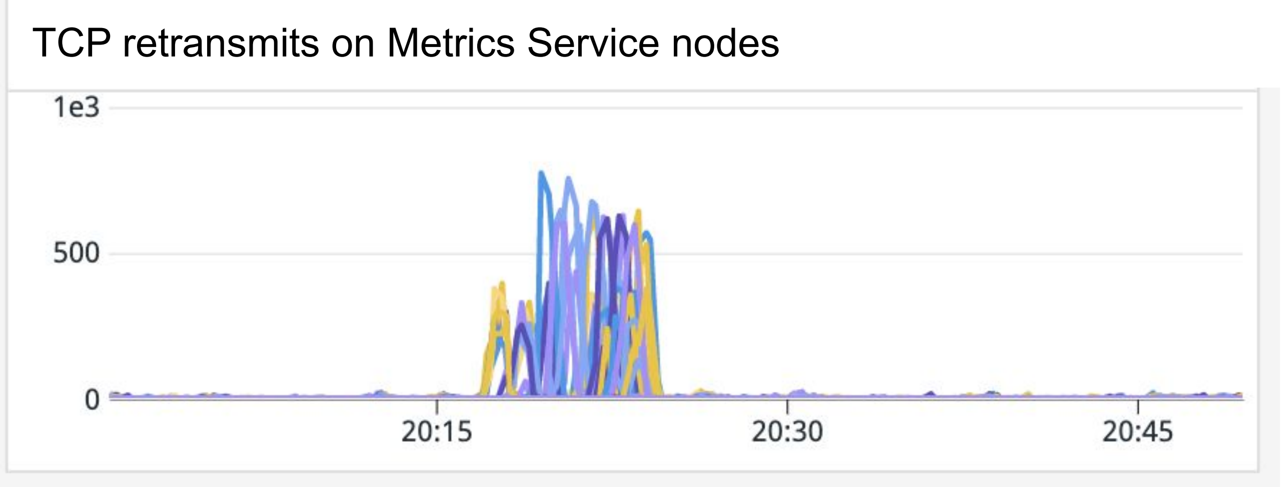

Throughput (Gb/s) on Metrics Service nodes (+: Received / -: Sent)

m5.4xlarge

Max: 10Gb/s

Sustained: 5Gb/s

=> looks ok


TCP retransmits on Metrics Service nodes

But we are dropping packets

Microbursts?

=> Elastic Network Adapter (ENA) metrics

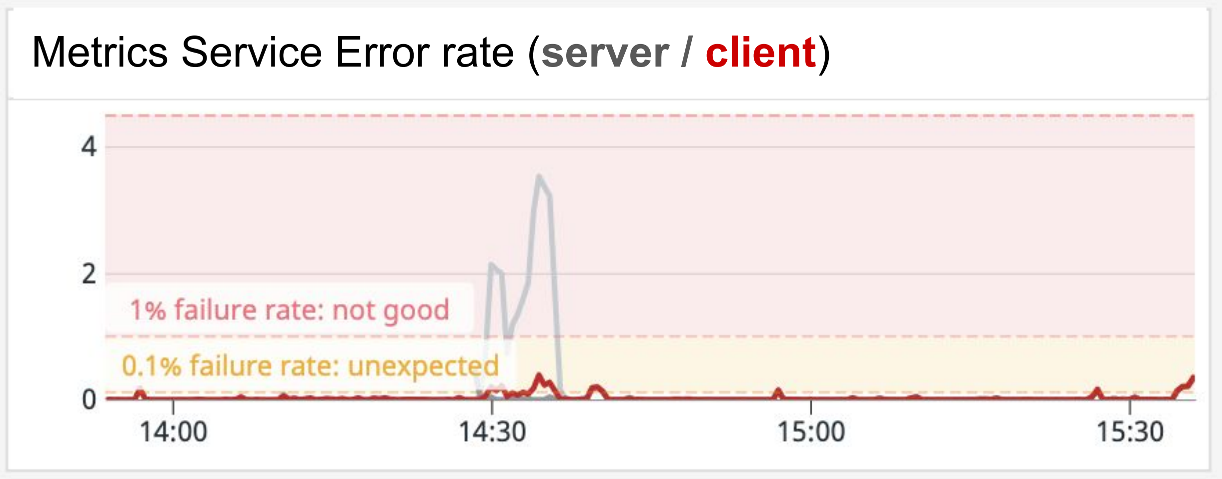
Status

- DNS errors in Metrics Service on rollouts
- Node-local-DNS can't establish connections

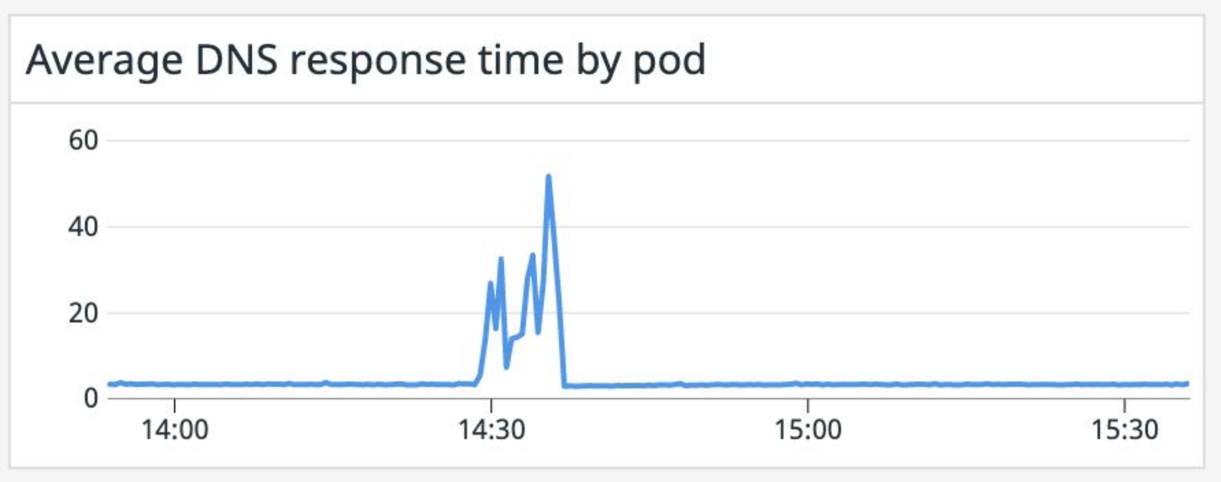
=> *Network issue?*

Chapter 2: AWS Networking

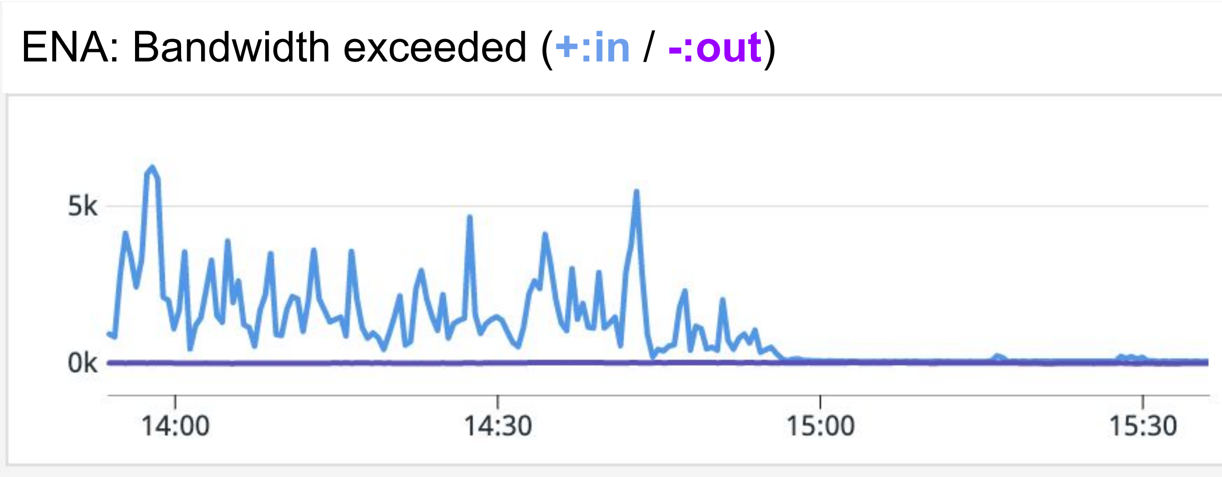
Are we bursting over the instance limits?



Metrics Service Error rate (**server** / **client**)



Average DNS response time by pod



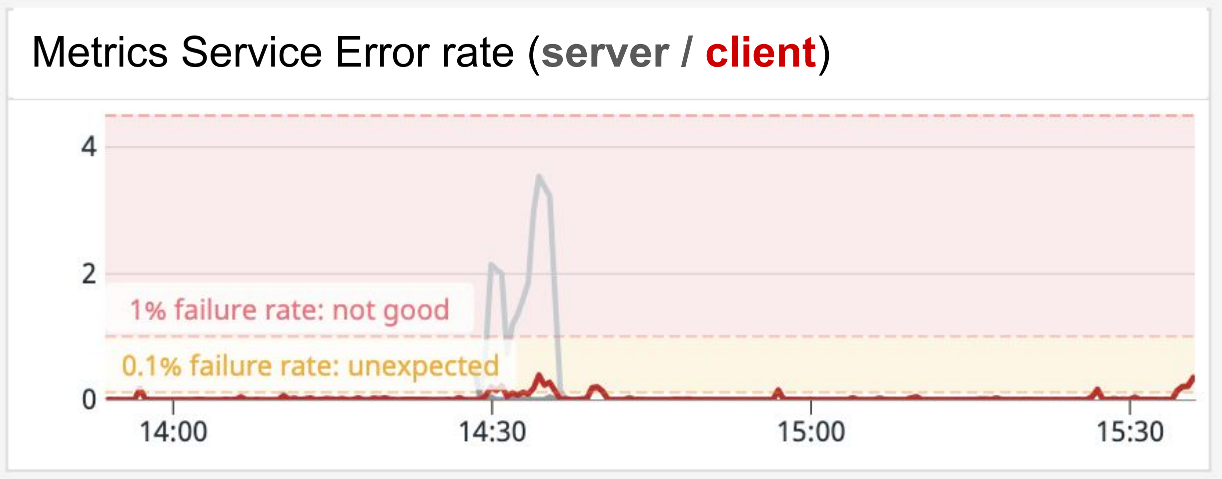
ENA: Bandwidth exceeded (**+:in** / **-:out**)

We are saturating the interface

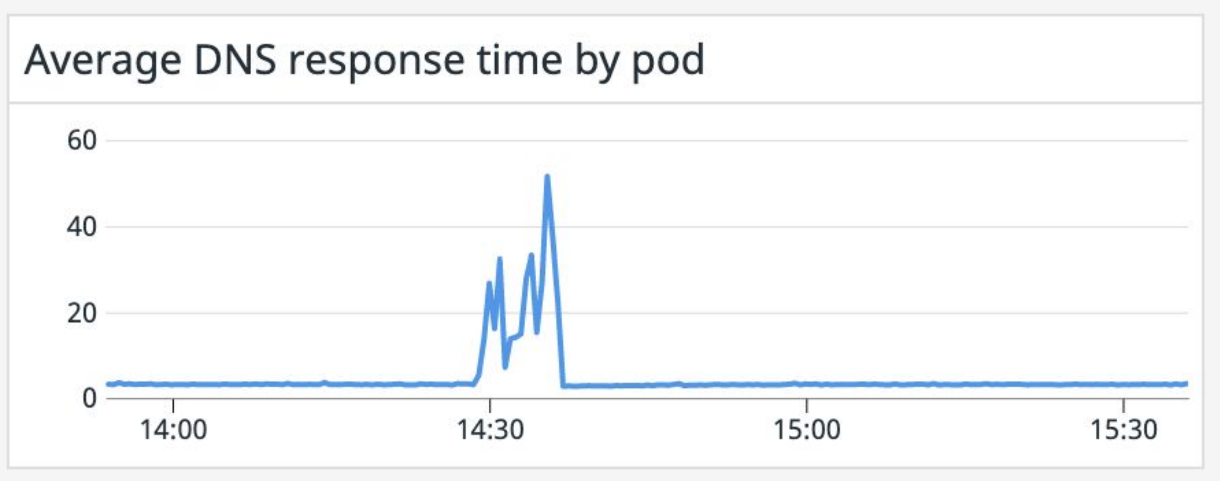
But no correlation with errors

Are we bursting over the instance limits?

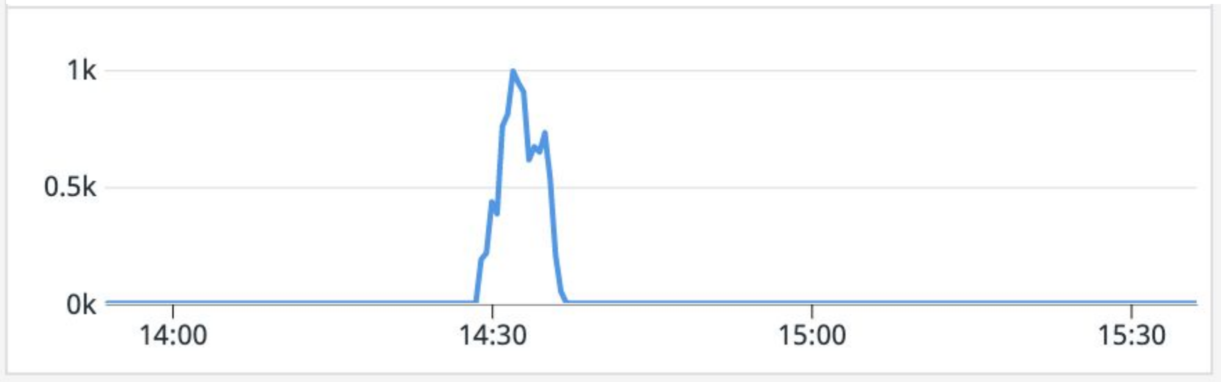
Metrics Service Error rate (**server** / **client**)



Average DNS response time by pod



ENA: Conntrack exceeded



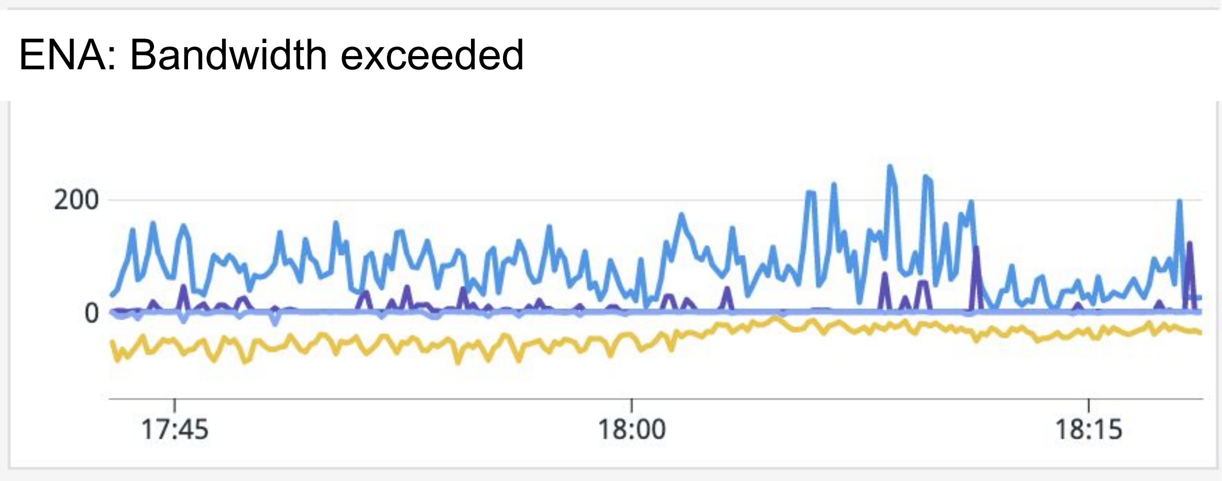
conntrack allowance exceeded?

aws.ec2.conntrack_allowance_exceeded

The number of packets dropped because connection tracking exceeded the maximum for the instance and new connections could not be established. This can result in packet loss for traffic to or from the instance

Connection tracking is required for security groups (stateful)

Let's test with network optimized instances

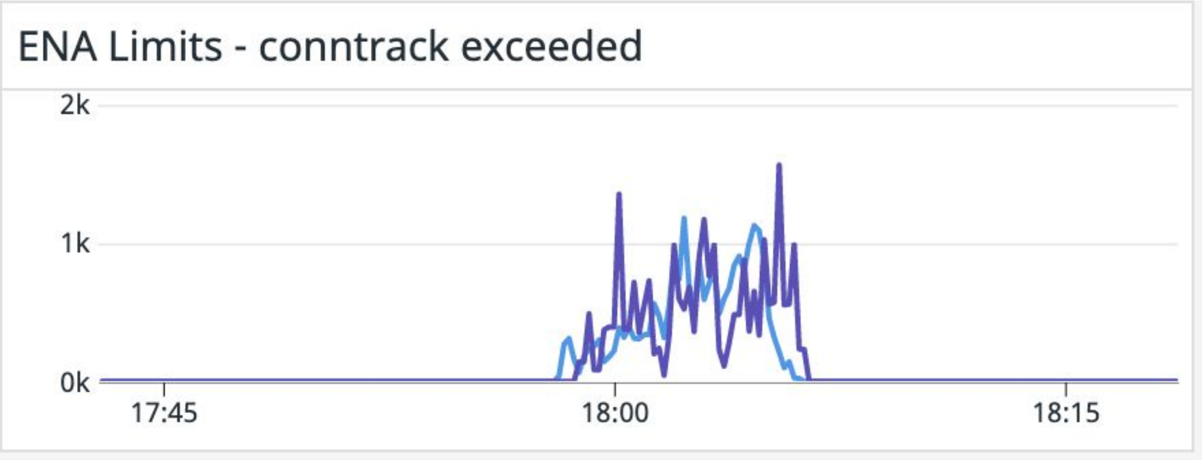


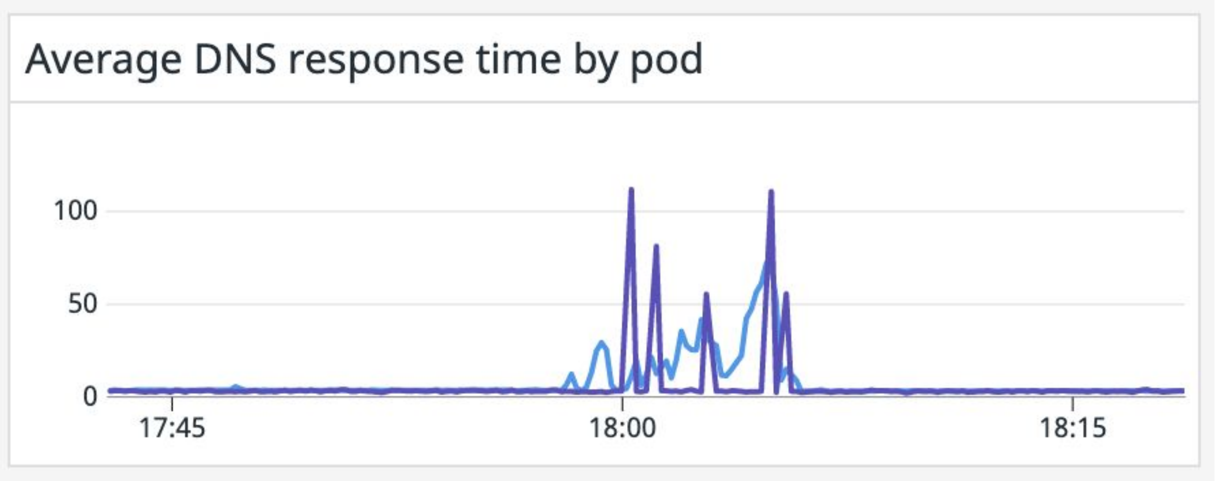
blue/yellow => m5.4xlarge

purple/grey (~0) => m5n.4xlarge

Promising!

Let's test with network optimized instances


ENA Limits - conntrack exceeded


Average DNS response time by pod

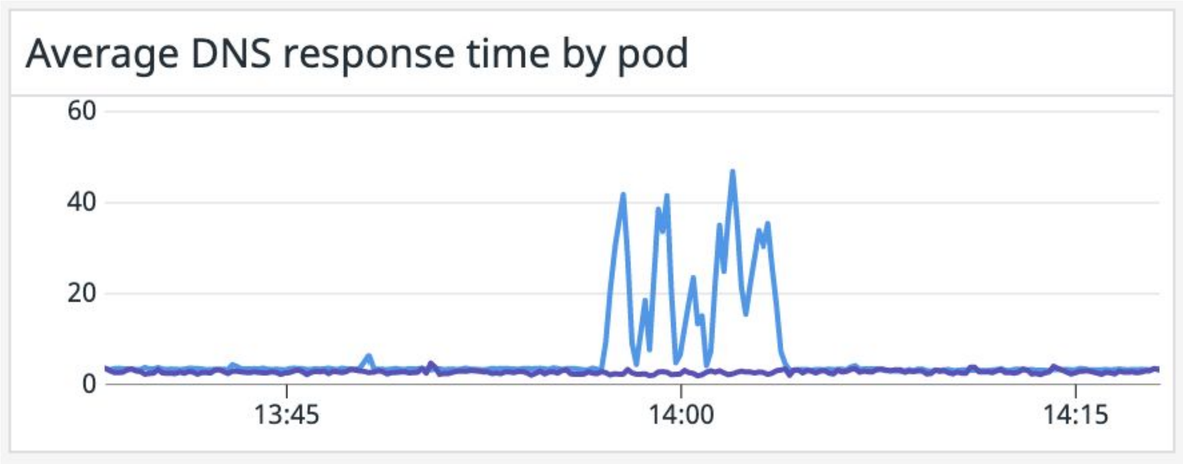
■ m5.4xlarge

■ m5n.4xlarge

No impact on

- Conntrack
- Metrics Service errors / latency

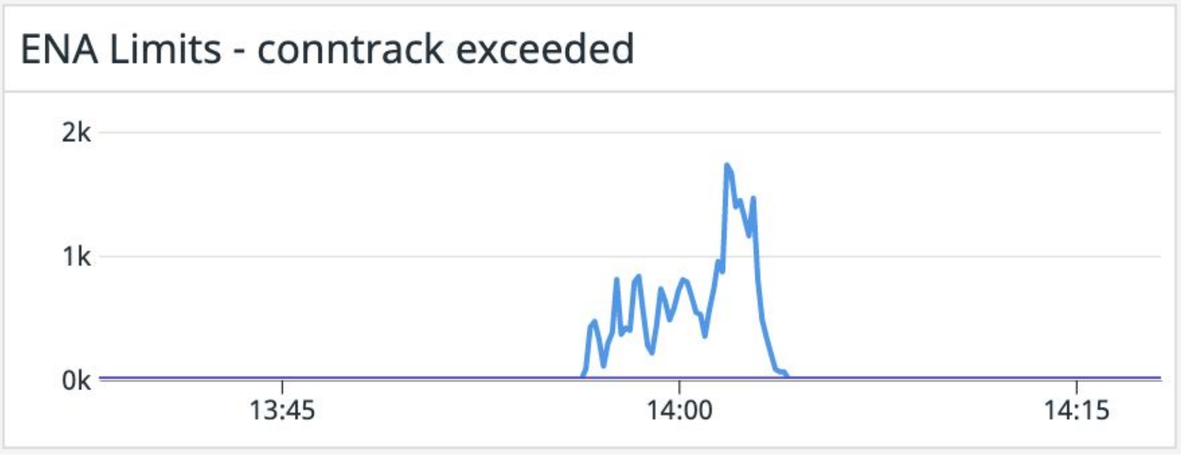
What about bigger instances?

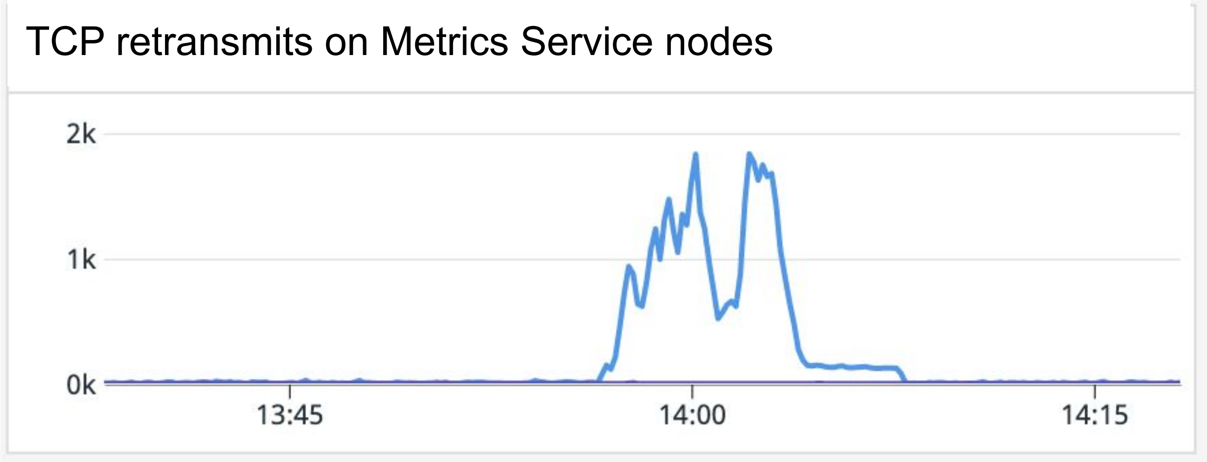

Average DNS response time by pod

🟦 m5.4xlarge

🟪 m5.8xlarge

Much better!


ENA Limits - conntrack exceeded


TCP retransmits on Metrics Service nodes

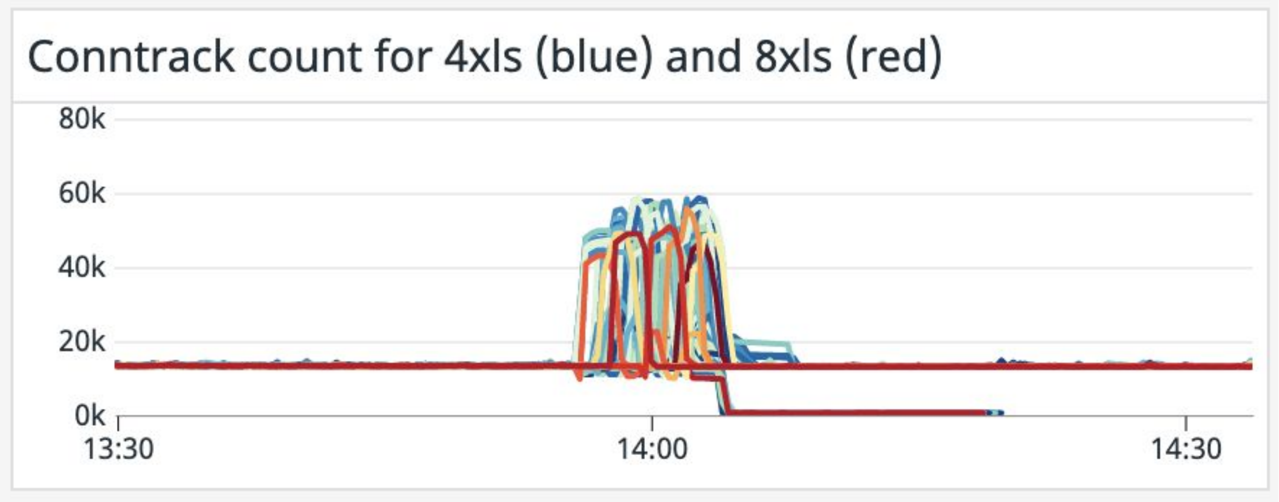
Conntrack limits?

From AWS

- Hypervisor conntrack can track hundreds of thousands of flows
- m5.8xlarge : can track 2x the flows compared to m5.4xlarge
- m5n.4xlarge : same as m5.4xlarge

=> Makes sense based on our tests

How can we saturate this conntrack?



Stable state: **~13k** connections

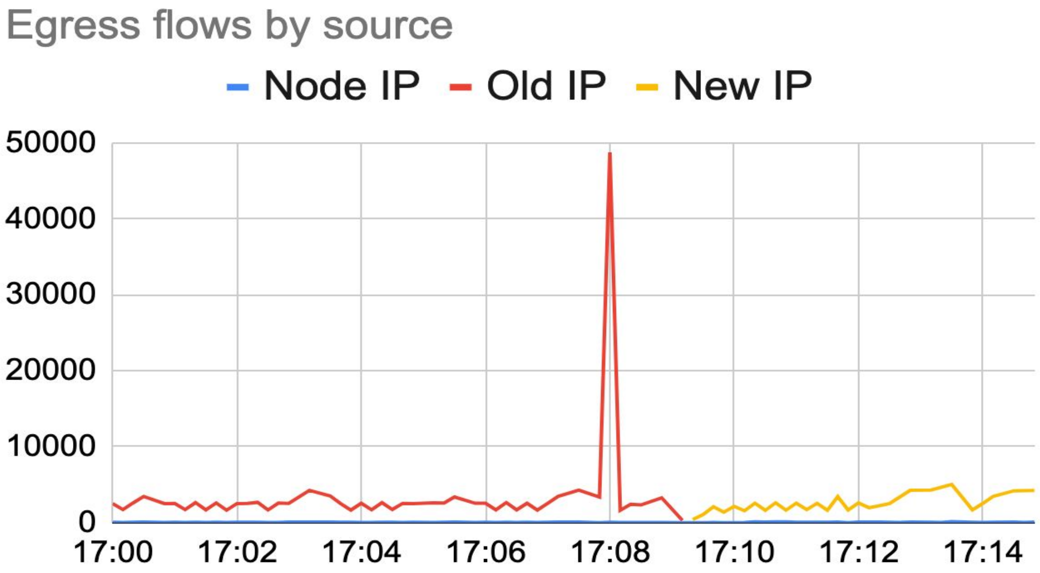
Rollouts: **~60k**

Pretty high but **60k vs X00k ????**

VPC Flow Logs

- Capture IP flow information on Elastic Network Interfaces (ENI)
- **Flow level: 5-tuple, 2 flows per TCP connection**
- Flow record: 5 tuple, bytes, packets, TCP flags...
- Aggregated every 1mn and delivered to S3
- **Not always complete**
- Huge amount for large VPCs (we filtered with Athena)

Flows from a Metrics Service node

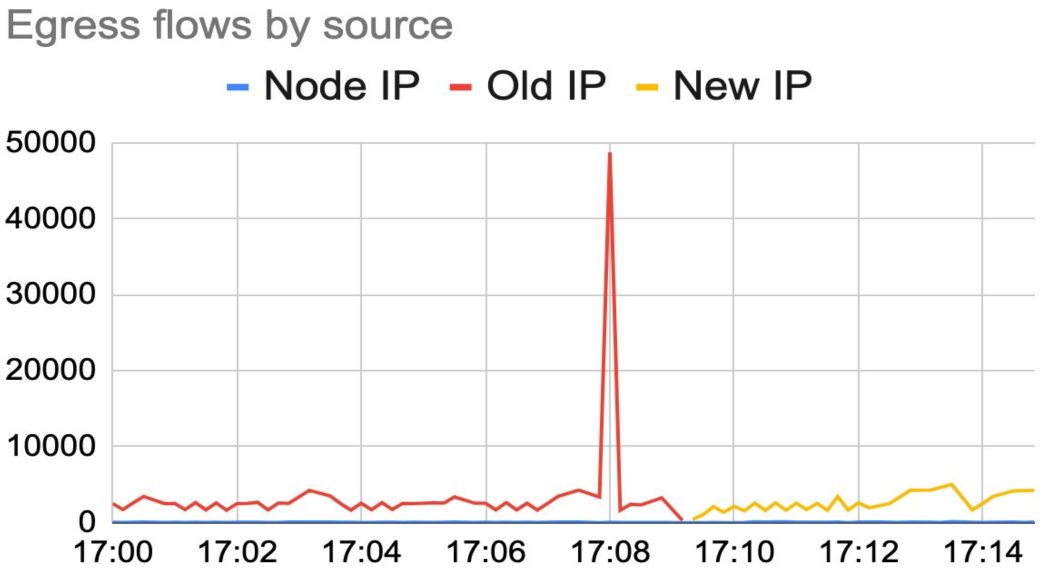


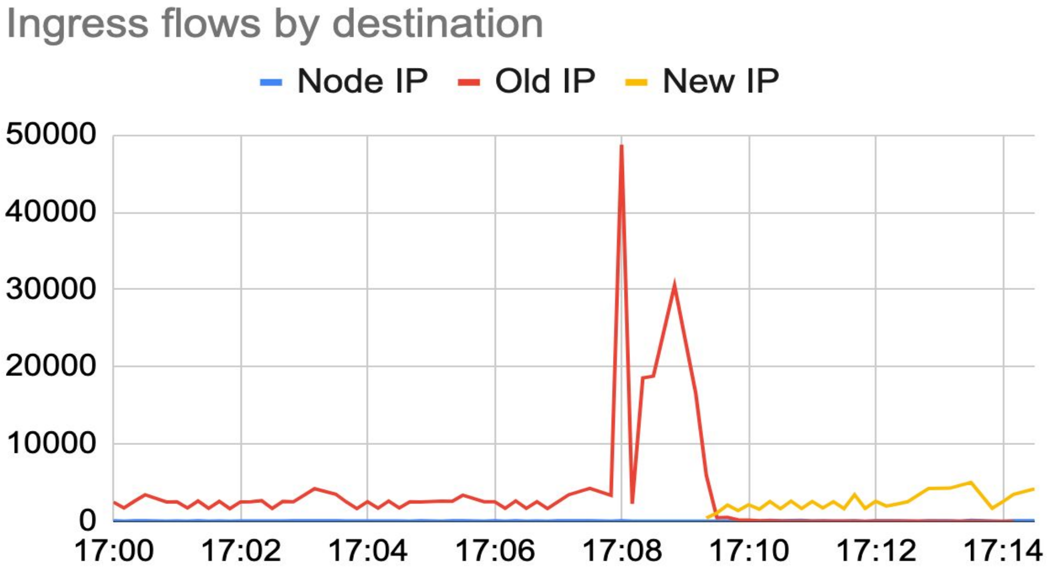
Old pod IP disappears after ~60s

Spike in flows at pod deletion

50k flows in 1mn feels very high

What about ingress flows?



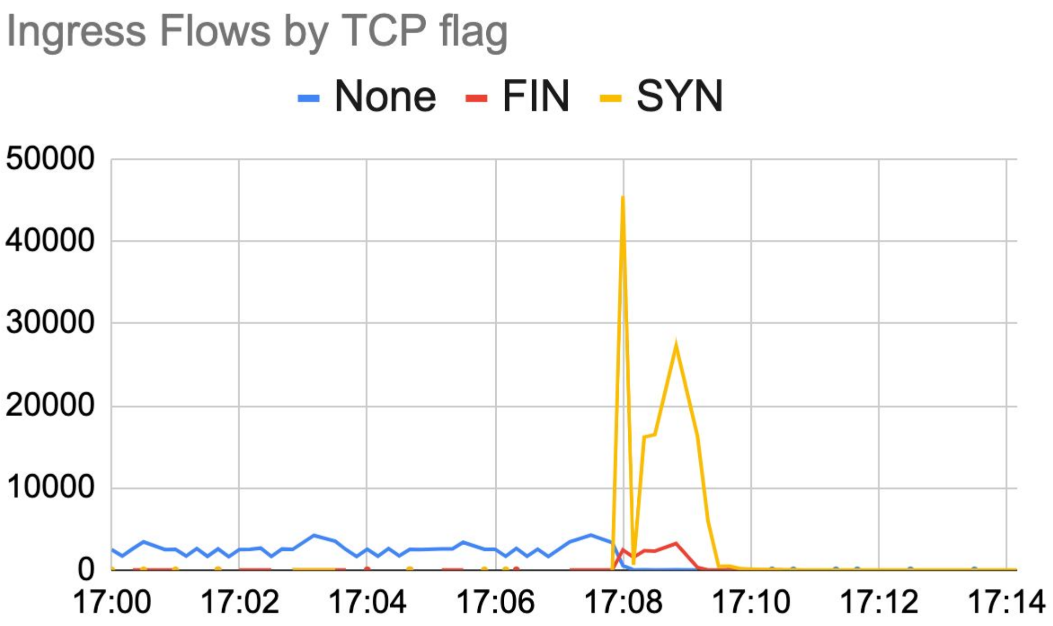


Ingress flows should ~match Egress

Very weird second spike

What are these flows?

Zoom on ingress flows to old IP

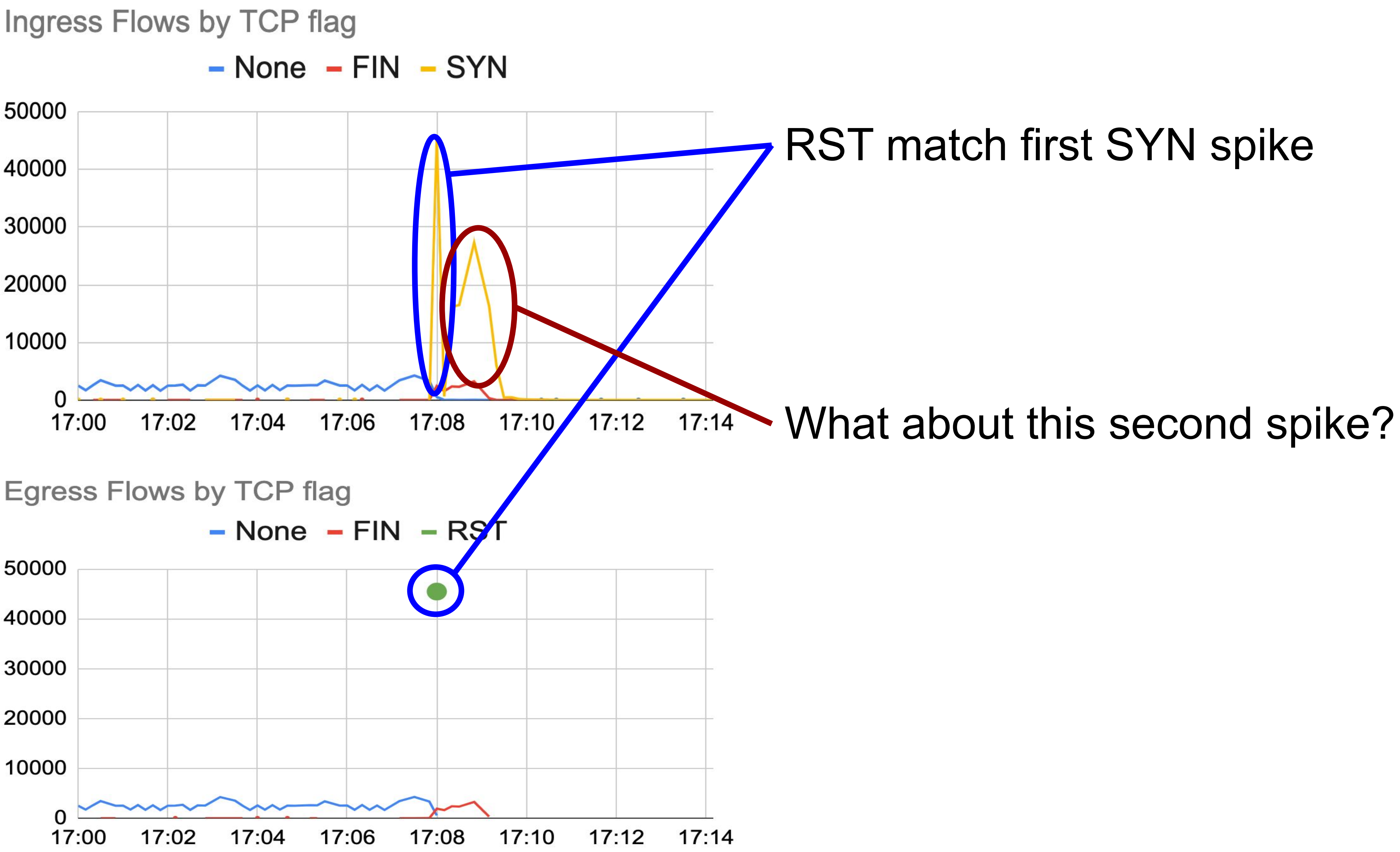


None: already established

FIN: terminating

SYN: reconnect attempts: **130k over 90s!**

What about egress?



RST match first SYN spike

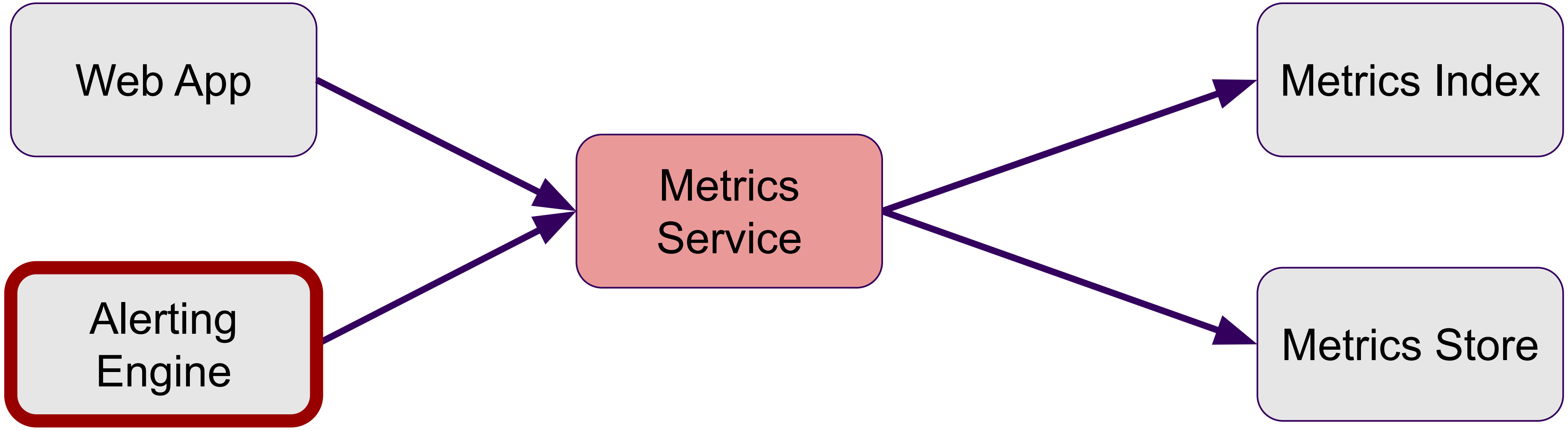
What about this second spike?

Why do we get RST for a few seconds only?

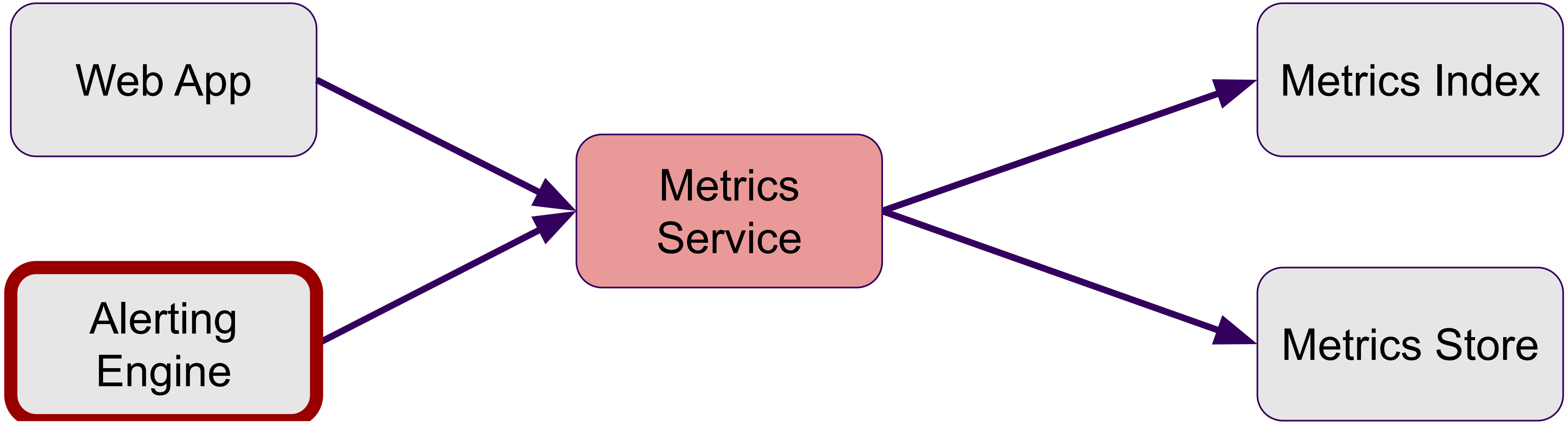
- Metrics Service performs a grpc.GracefulStop with 10s timeout
    - Server stops accepting new connections
    - Server waits for existing RPC to finish
    - Server tells clients to disconnect (HTTP2 GoAway)
- During these 10s, incoming connection attempts get an RST
- After these 10s, the pod is deleted and its IP is not bound by anything

Where are these attempts coming from?

Only a few IPs => Alerting Engine



Where are these attempts coming from?



Web App

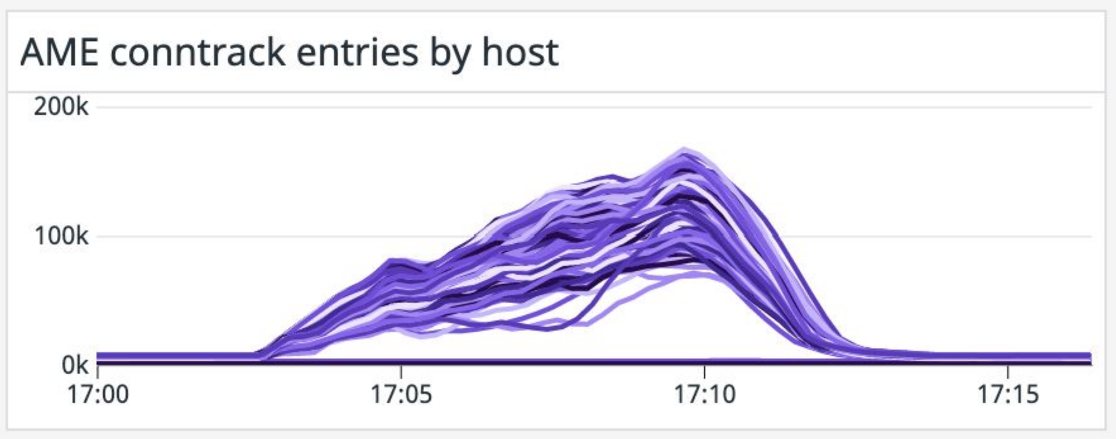
Alerting Engine

Metrics Service

Metrics Index

Metrics Store

#conntrack entries on Alerting Engine nodes



AME conntrack entries by host

Seems to confirm!

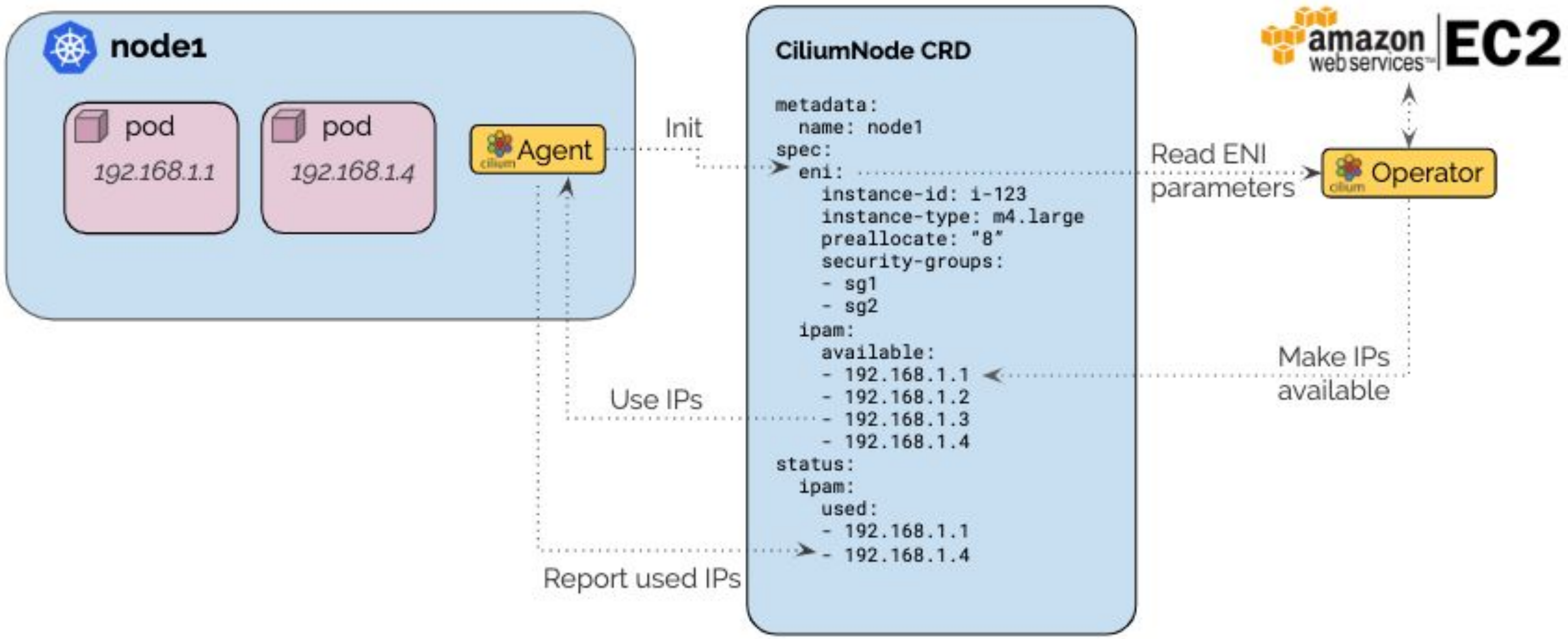
Status

- DNS errors in Metrics Service on rollouts
- Node-local-DNS can't establish connections
- AWS conntrack for instance is saturated
- Alerting Engine is SYN-Flooding Metrics Service on rollouts

=> *Why don't we see these connections on Metric Service Nodes?*

Chapter 3: Node Networking

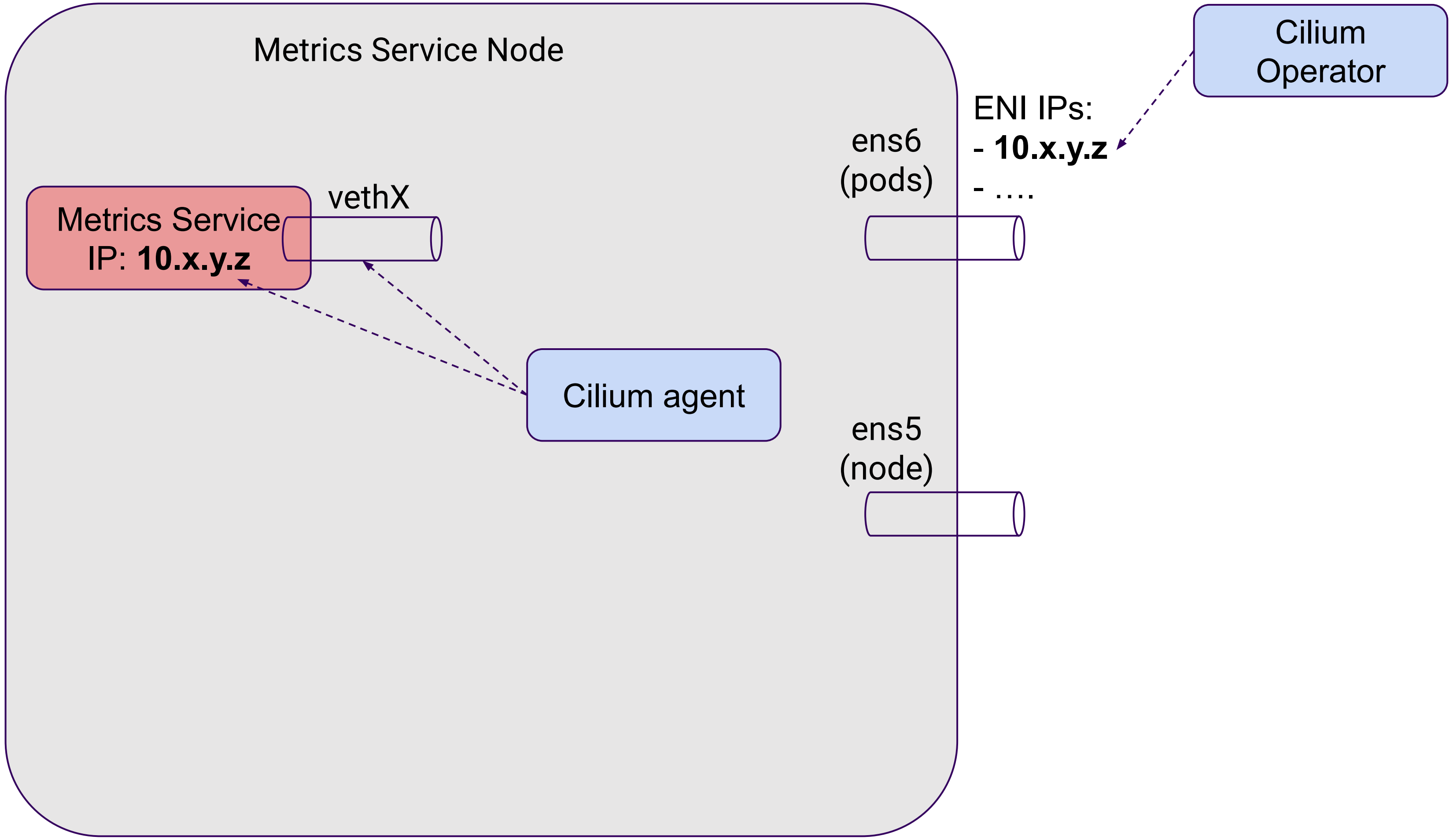
Cilium Architecture



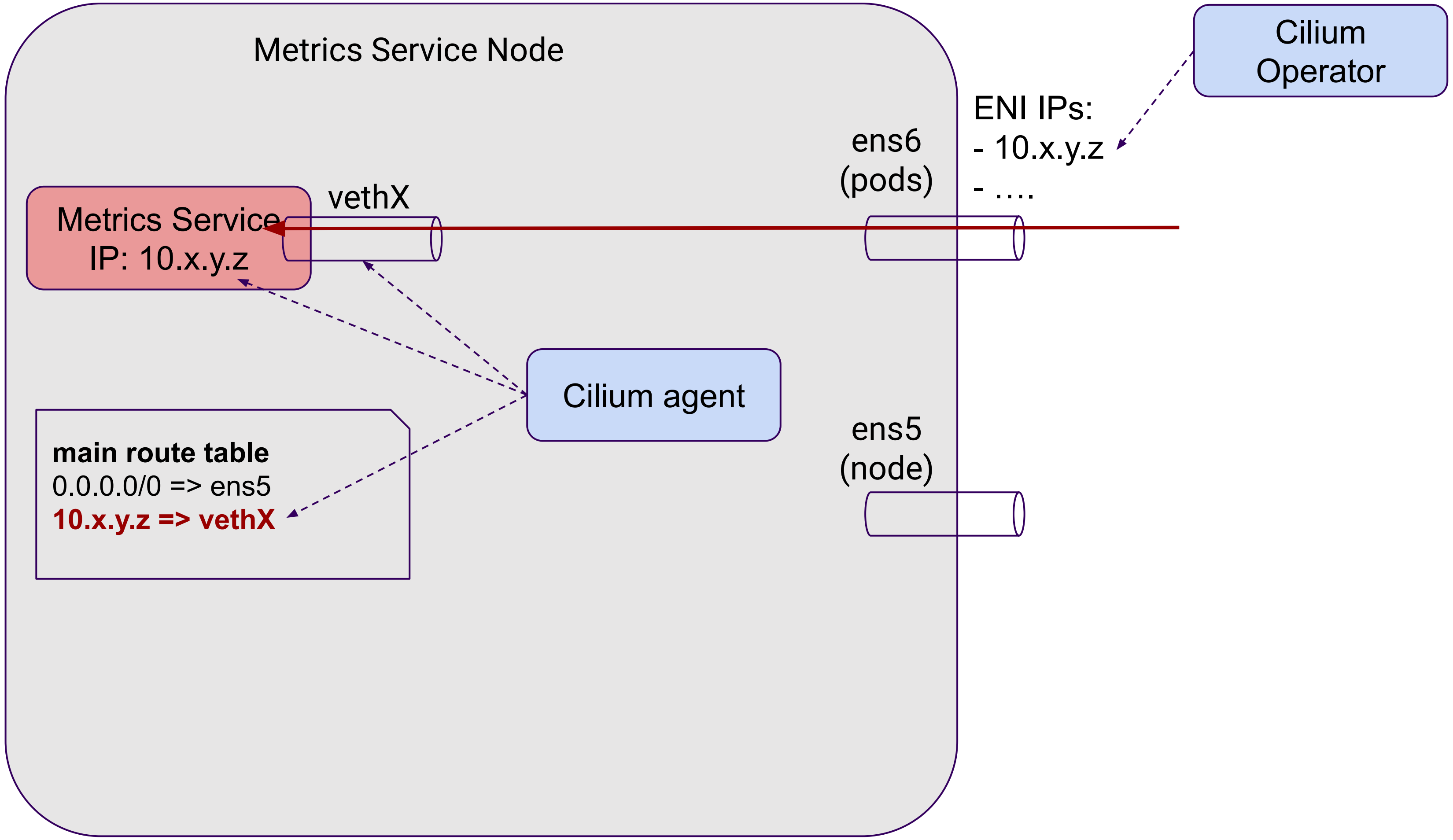
```
metadata:
  name: node1
spec:
  eni:
    instance-id: i-123
    instance-type: m4.large
    preallocate: "8"
    security-groups:
    - sg1
    - sg2
  ipam:
    available:
    - 192.168.1.1
    - 192.168.1.2
    - 192.168.1.3
    - 192.168.1.4
status:
  ipam:
    used:
    - 192.168.1.1
    - 192.168.1.4
```

Source : Cilium documentation

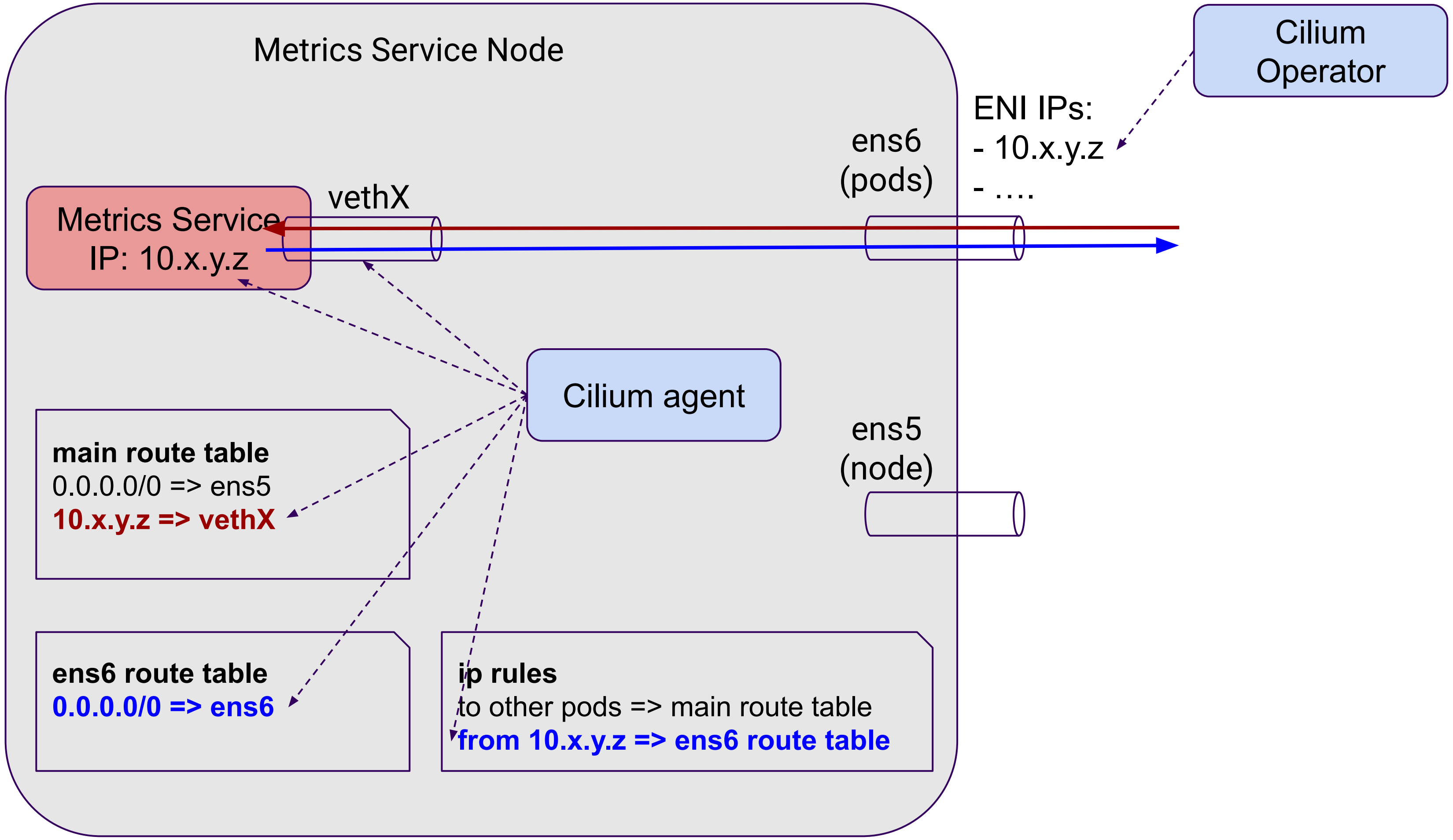
Routing on nodes



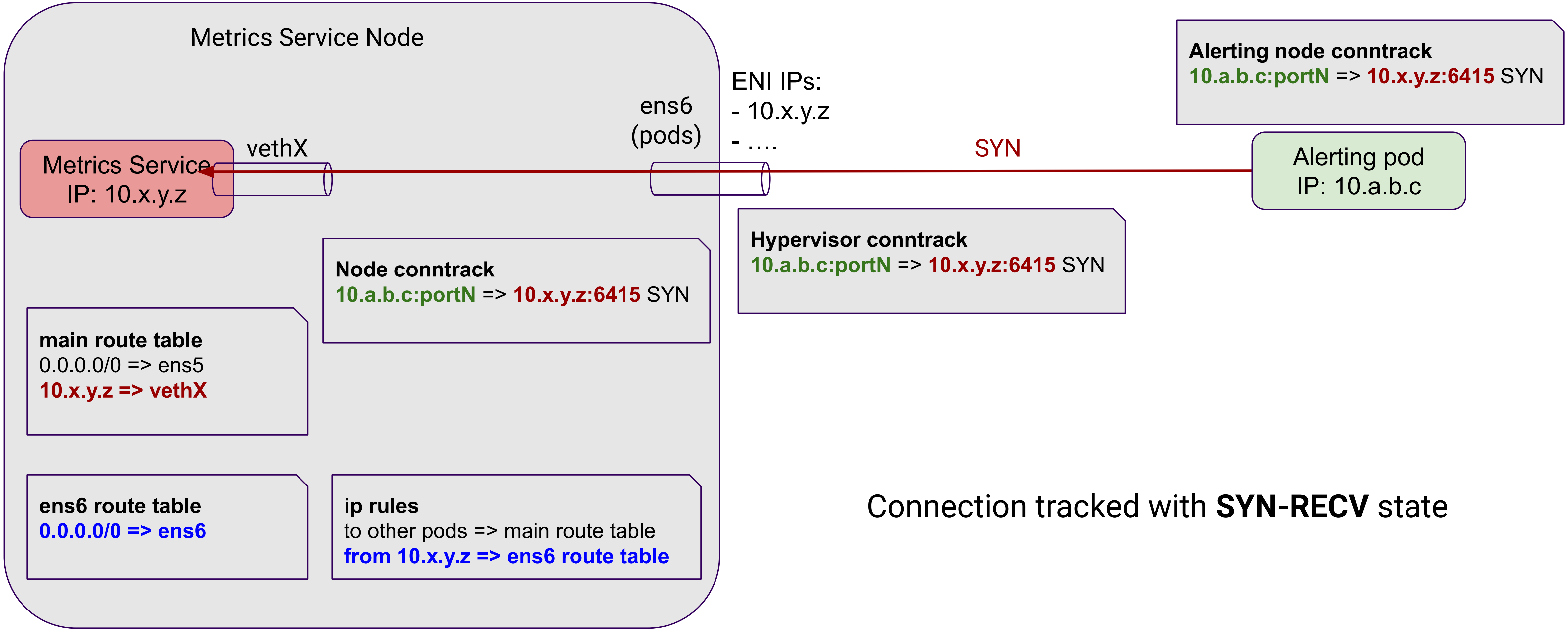
Routing on nodes



Routing on nodes



Stable state



Metrics Service Node

Metrics Service
IP: 10.x.y.z

vethX

ens6
(pods)

ENI IPs:
- 10.x.y.z
- ....

SYN

Alerting pod
IP: 10.a.b.c

Alerting node conntrack
10.a.b.c:portN => 10.x.y.z:6415 SYN

Hypervisor conntrack
10.a.b.c:portN => 10.x.y.z:6415 SYN

Node conntrack
10.a.b.c:portN => 10.x.y.z:6415 SYN

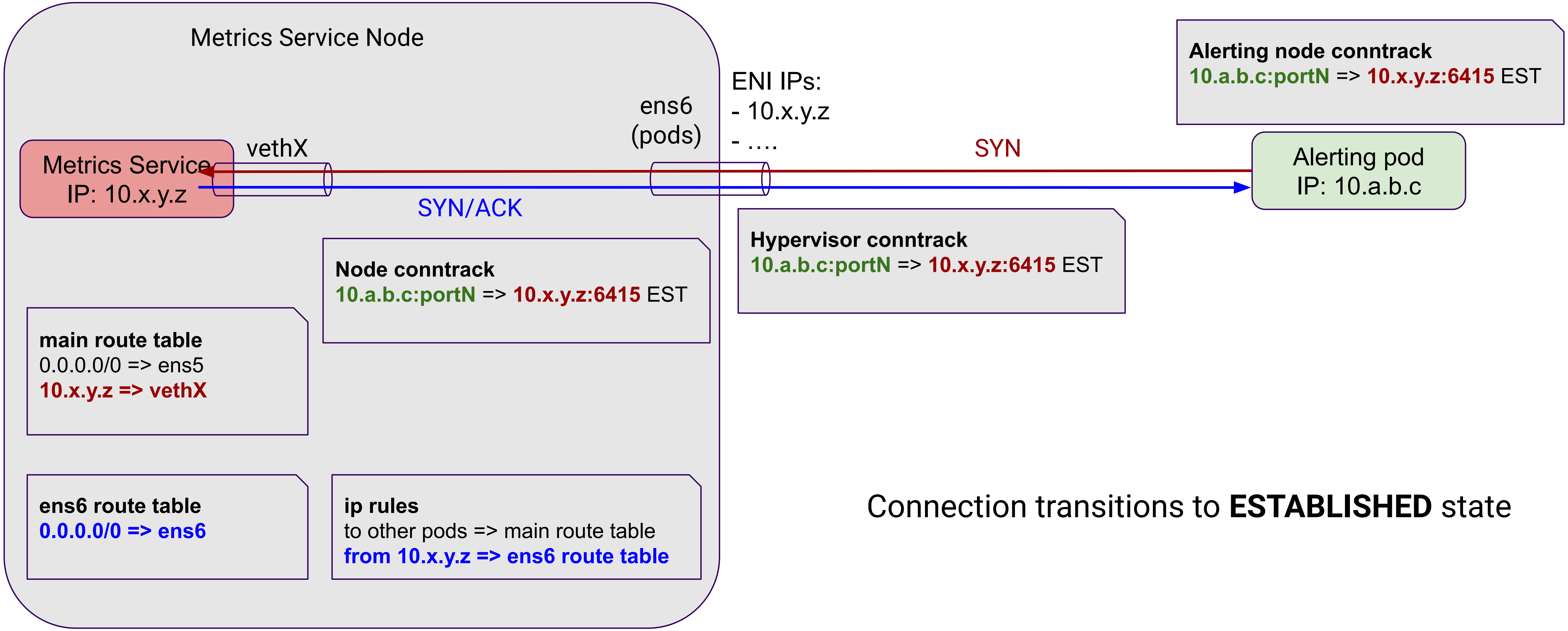
main route table
0.0.0.0/0 => ens5
10.x.y.z => vethX

ens6 route table
0.0.0.0/0 => ens6

ip rules
to other pods => main route table
from 10.x.y.z => ens6 route table

Connection tracked with **SYN-RECV** state

Stable state



Metrics Service Node

Metrics Service
IP: 10.x.y.z

vethX

ens6
(pods)

ENI IPs:
- 10.x.y.z
- ….

SYN

SYN/ACK

Alerting pod
IP: 10.a.b.c

Alerting node conntrack
10.a.b.c:portN => 10.x.y.z:6415 EST

Hypervisor conntrack
10.a.b.c:portN => 10.x.y.z:6415 EST

Node conntrack
10.a.b.c:portN => 10.x.y.z:6415 EST

main route table
0.0.0.0/0 => ens5
10.x.y.z => vethX

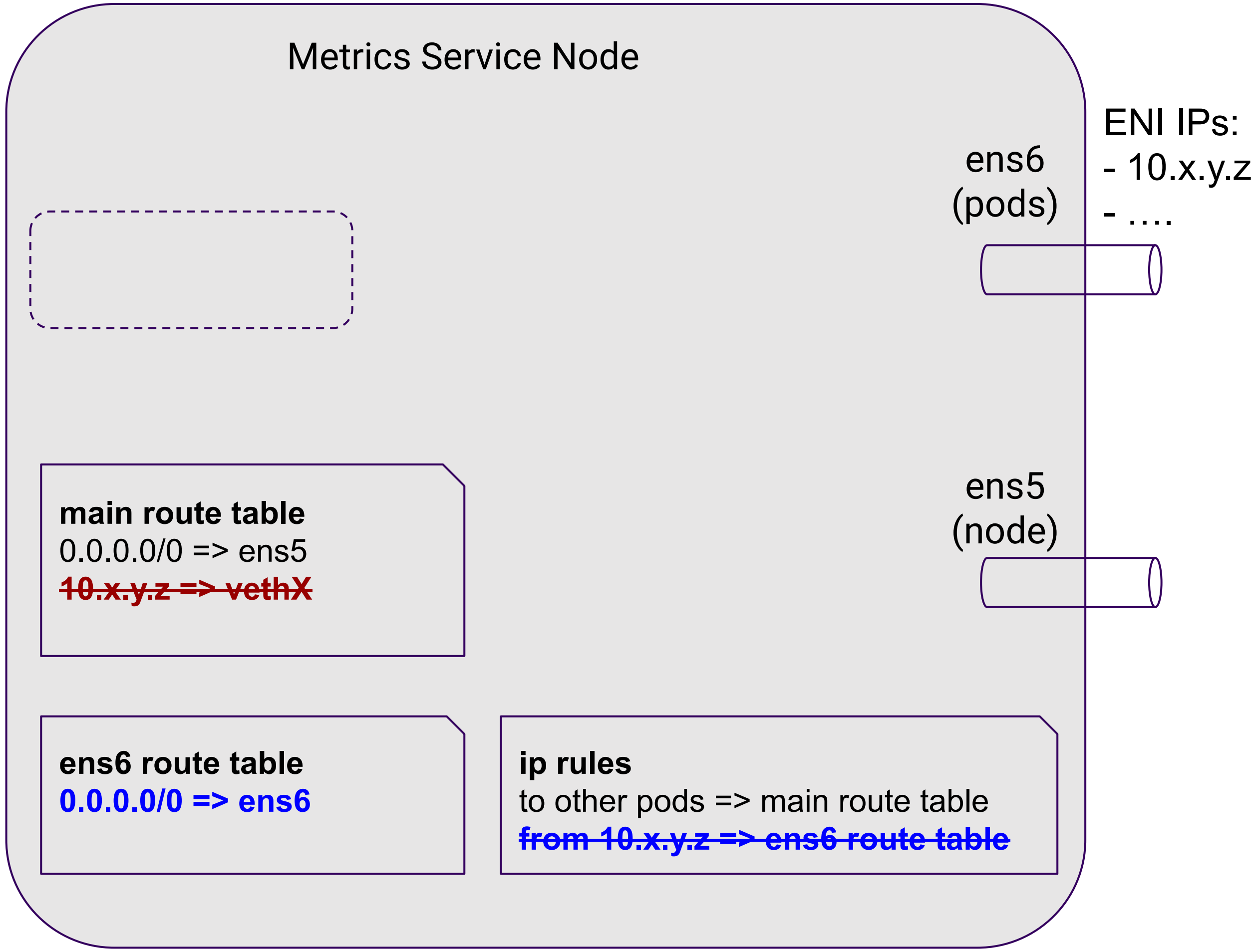
ens6 route table
0.0.0.0/0 => ens6

ip rules
to other pods => main route table
from 10.x.y.z => ens6 route table

Connection transitions to **ESTABLISHED** state

What happens on pod deletion?

SRE CON AMERICAS



Metrics Service Node

ENI IPs:
- 10.x.y.z
- ….

ens6 (pods)

ens5 (node)

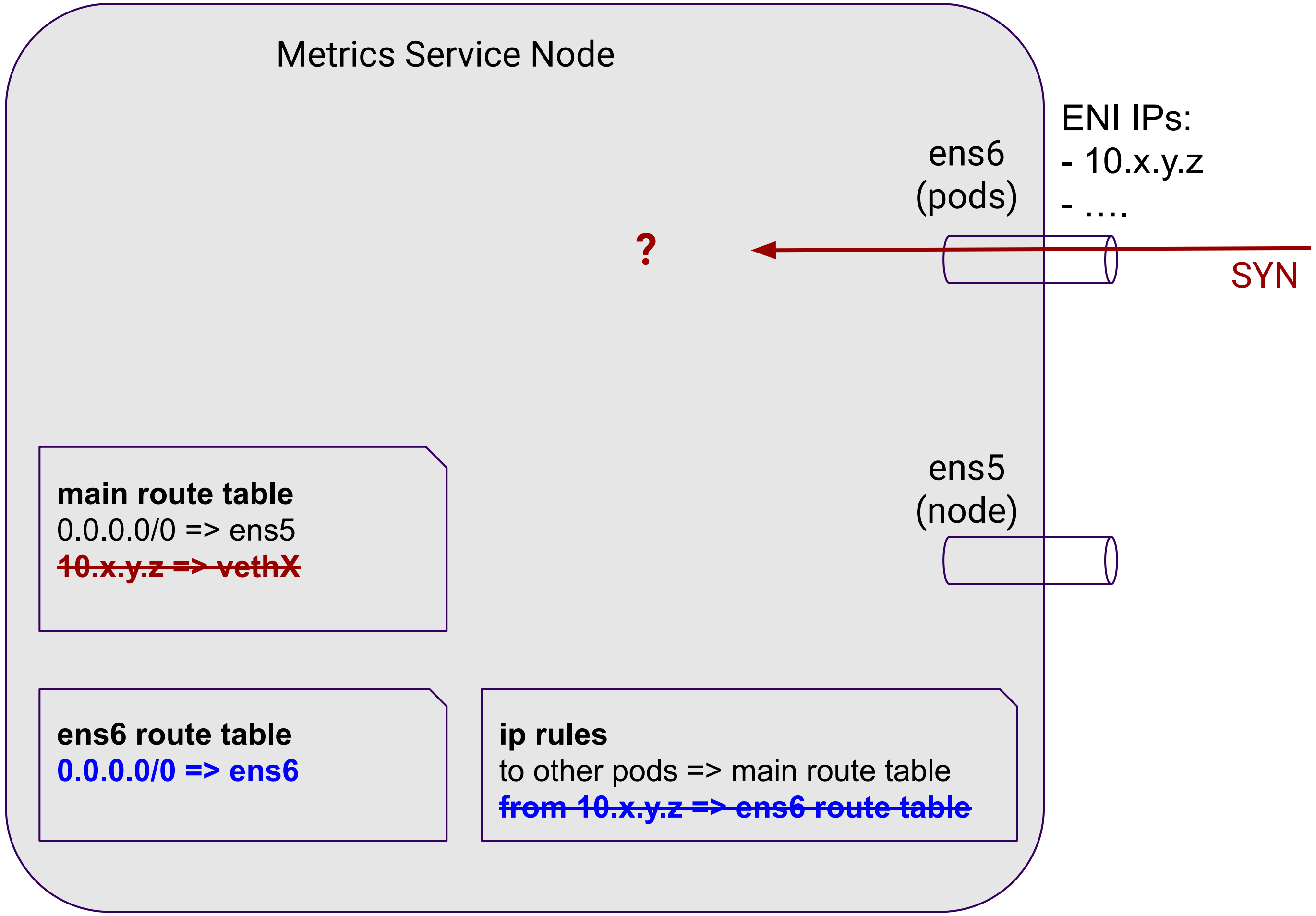
main route table
0.0.0.0/0 => ens5
~~**10.x.y.z => vethX**~~

ens6 route table
0.0.0.0/0 => ens6

ip rules
to other pods => main route table
~~**from 10.x.y.z => ens6 route table**~~

DATADOG

What about traffic to old IP?



Metrics Service Node

ENI IPs:
- 10.x.y.z
- ....

main route table
0.0.0.0/0 => ens5
~~10.x.y.z => vethX~~

ens6 route table
0.0.0.0/0 => ens6

ip rules
to other pods => main route table
~~from 10.x.y.z => ens6 route table~~

Let's simulate

Delete pod with IP 10.x.y.z on nodeB and attempt to connect from nodeA

Connection attempt

```
nodeA:~$ nc -vz 10.x.y.z 12345
```

Let's simulate

Delete pod with IP 10.x.y.z on nodeB and attempt to connect from nodeA

Connection attempt

```
nodeA:~$ nc -vz 10.x.y.z 12345
```

On nodeB => SYN without an answer

```
nodeB:~$ sudo tcpdump -pni ens6 "port 12345"
listening on ens6, link-type EN10MB (Ethernet), capture size 262144 bytes
08:28:52.086251 IP 10.a.b.c.51718 > 10.x.y.z.12345: Flags [S], seq 4126537246, win 26883, options [mss
8961,sackOK,TS val 2002199904 ecr 0,nop,wscale 9], length 0
```

Let's simulate

Delete pod with IP 10.x.y.z on nodeB and attempt to connect from nodeA

Connection attempt

```
nodeA:~$ nc -vz 10.x.y.z 12345
```

On nodeB => SYN without an answer

```
nodeB:~$ sudo tcpdump -pni ens6 "port 12345"
listening on ens6, link-type EN10MB (Ethernet), capture size 262144 bytes
08:28:52.086251 IP 10.a.b.c.51718 > 10.x.y.z.12345: Flags [S], seq 4126537246, win 26883, options [mss
8961,sackOK,TS val 2002199904 ecr 0,nop,wscale 9], length 0
```

Where would the SYN be routed to? => Reverse Path filter!

```
$ ip route get 10.x.y.z from 10.a.b.c iif ens6
RTNETLINK answers: Invalid cross-device link
```

Let's simulate

Delete pod with IP 10.x.y.z on nodeB and attempt to connect from nodeA

Connection attempt

```
nodeA:~$ nc -vz 10.x.y.z 12345
```

On nodeB => SYN without an answer

```
nodeB:~$ sudo tcpdump -pni ens6 "port 12345"
listening on ens6, link-type EN10MB (Ethernet), capture size 262144 bytes
08:28:52.086251 IP 10.a.b.c.51718 > 10.x.y.z.12345: Flags [S], seq 4126537246, win 26883, options [mss
8961,sackOK,TS val 2002199904 ecr 0,nop,wscale 9], length 0
```

Where would the SYN be routed to? => Reverse Path filter!

```
$ ip route get 10.x.y.z from 10.a.b.c iif ens6
RTNETLINK answers: Invalid cross-device link
```

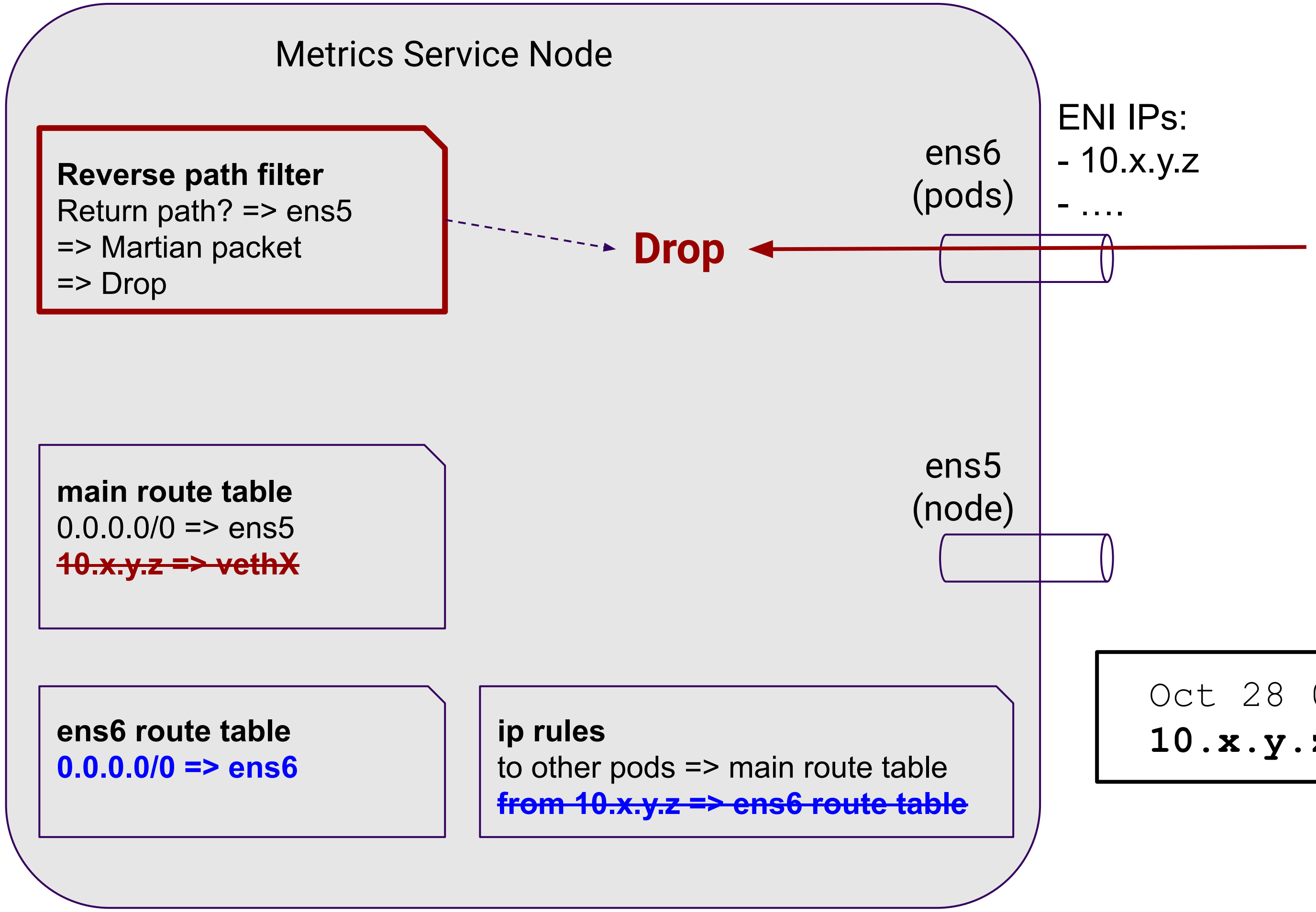
Sure enough, martian packet warning in kernel logs

```
Oct 28 08:25:54 nodeB kernel: IPv4: martian source 10.x.y.z from 10.a.b.c, on dev ens6
```

Reverse Path filtering

- Security feature from the kernel to prevent IP spoofing
  - If return path uses incoming interface accept the packet
  - Otherwise drop it
- Log these events : "Martian Packets"
- Loose mode: only drop if there is no return route

Back to our node



Metrics Service Node

Reverse path filter
Return path? => ens5
=> Martian packet
=> Drop

Drop

ens6
(pods)

ENI IPs:
- 10.x.y.z
- ….

ens5
(node)

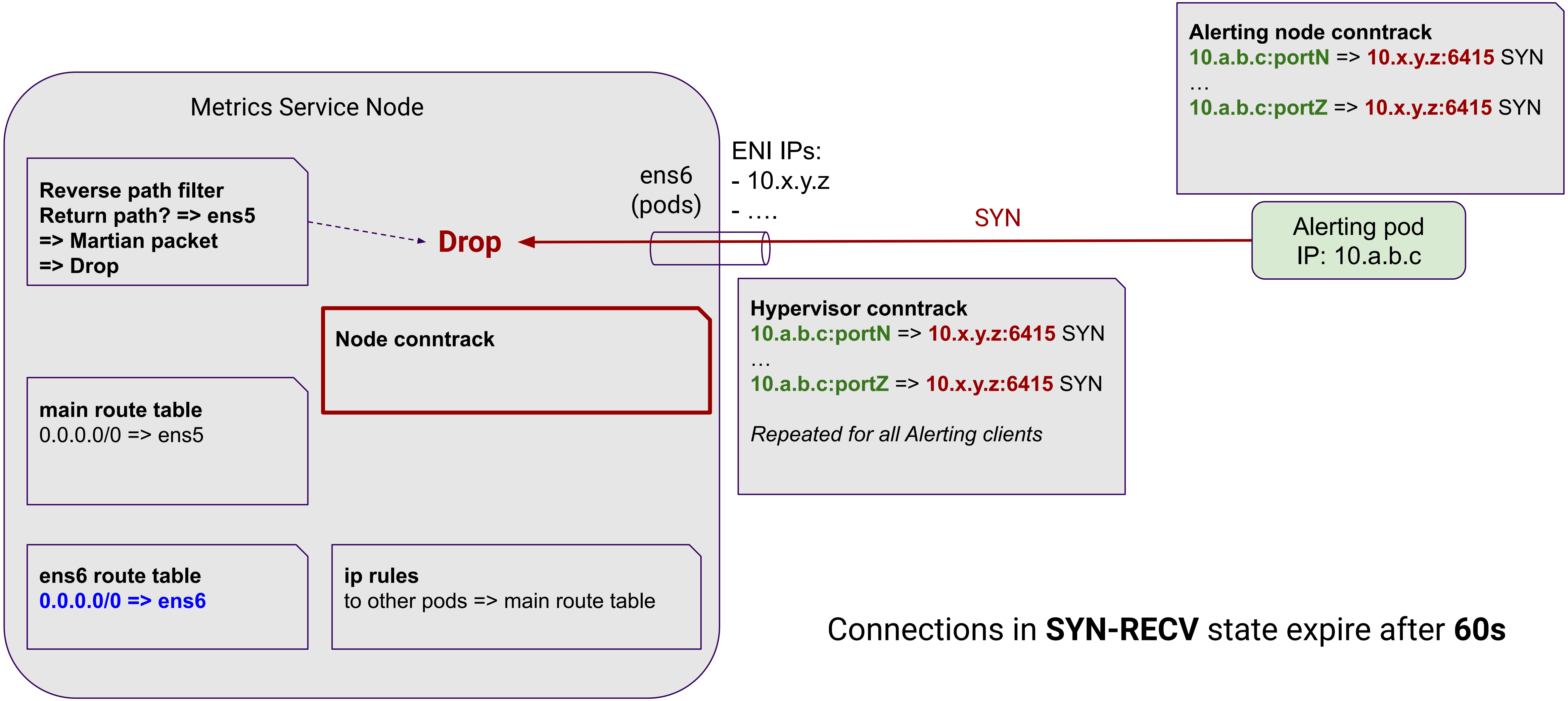
main route table
0.0.0.0/0 => ens5
~~**10.x.y.z => vethX**~~

ens6 route table
0.0.0.0/0 => ens6

ip rules
to other pods => main route table
~~**from 10.x.y.z => ens6 route table**~~

```
Oct 28 08:25:54 kernel: IPv4: martian source
10.x.y.z from 10.a.b.c, on dev ens6
```

What about conntracks?



Alerting node conntrack
10.a.b.c:portN => 10.x.y.z:6415 SYN
…
10.a.b.c:portZ => 10.x.y.z:6415 SYN

Hypervisor conntrack
10.a.b.c:portN => 10.x.y.z:6415 SYN
…
10.a.b.c:portZ => 10.x.y.z:6415 SYN

Repeated for all Alerting clients

Connections in **SYN-RECV** state expire after **60s**

But, we use "loose" mode

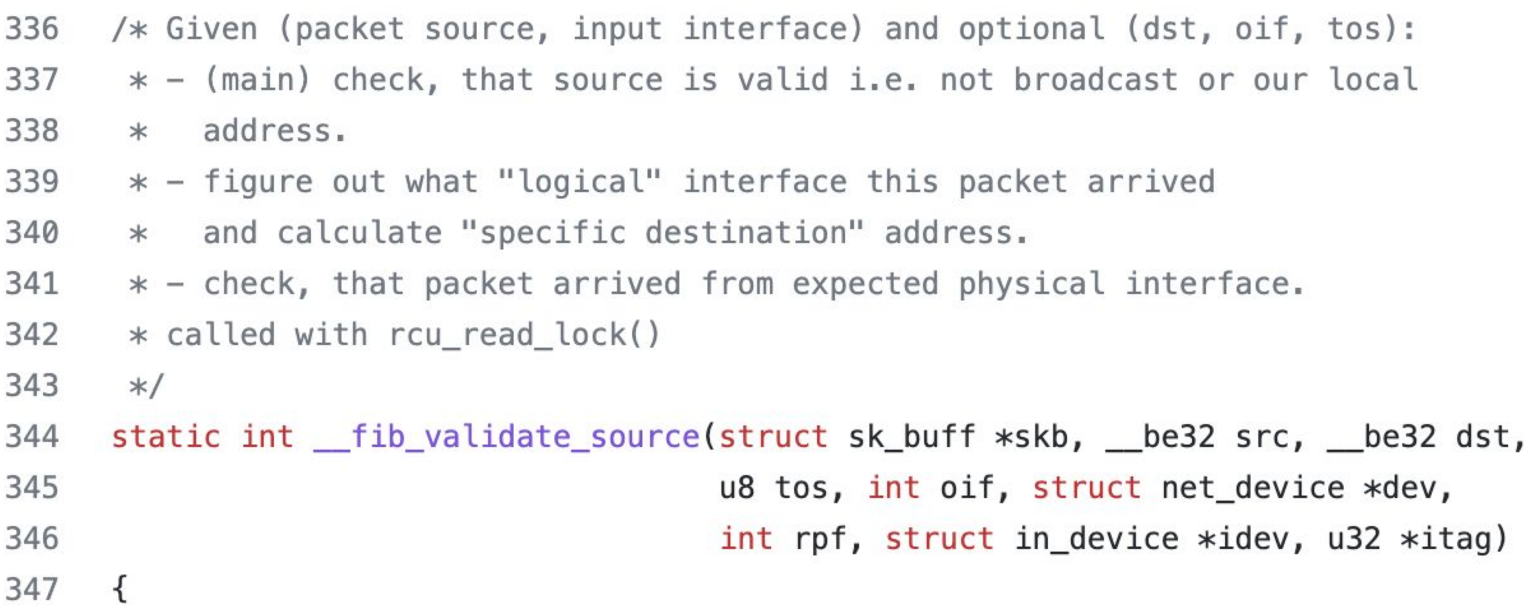
```
$ ip route get 10.x.y.z from 10.a.b.c iif ens6
RTNETLINK answers: Invalid cross-device link

$ sysctl net.ipv4.conf.ens6.rp_filter
net.ipv4.conf.ens6.rp_filter = 2
```

- rp_filter = 2 => loose mode
- Loose + default route (ens5) => we should not drop
- What's happening?

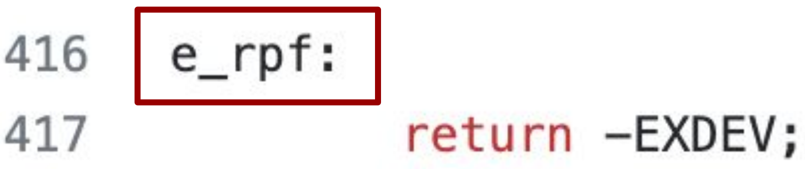
Let's have a look

https://github.com/torvalds/linux/blob/master/net/ipv4/fib_frontend.c#L344

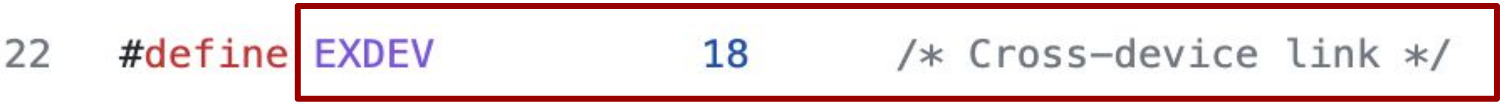


```
336  /* Given (packet source, input interface) and optional (dst, oif, tos):
337   * - (main) check, that source is valid i.e. not broadcast or our local
338   *   address.
339   * - figure out what "logical" interface this packet arrived
340   *   and calculate "specific destination" address.
341   * - check, that packet arrived from expected physical interface.
342   * called with rcu_read_lock()
343   */
344  static int __fib_validate_source(struct sk_buff *skb, __be32 src, __be32 dst,
345                                   u8 tos, int oif, struct net_device *dev,
346                                   int rpf, struct in_device *idev, u32 *itag)
347  {
```

Let's have a look

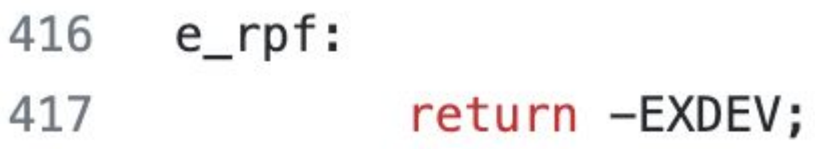


```
e_rpf:
                return -EXDEV;
```

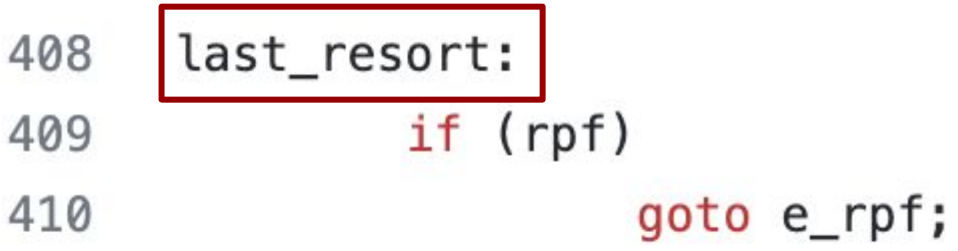


```
#define EXDEV           18      /* Cross-device link */
```

Let's have a look

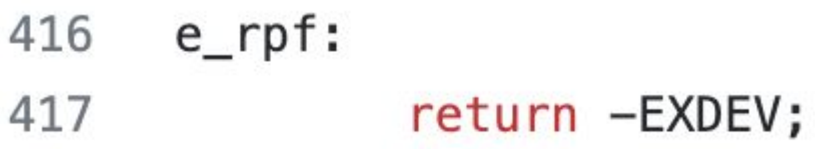


```
e_rpf:
        return -EXDEV;
```

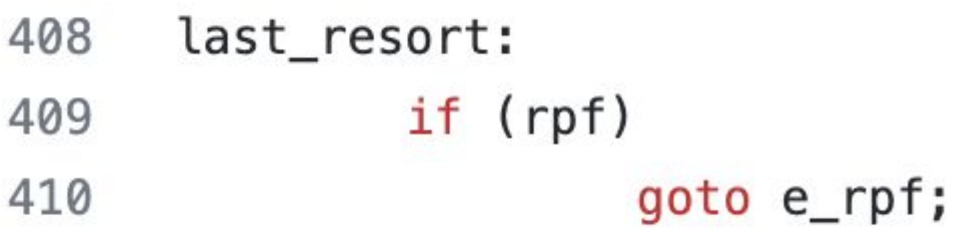


```
last_resort:
        if (rpf)
                goto e_rpf;
```

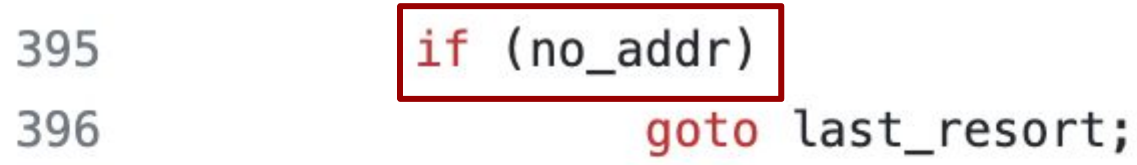
Let's have a look



```
e_rpf:
        return -EXDEV;
```

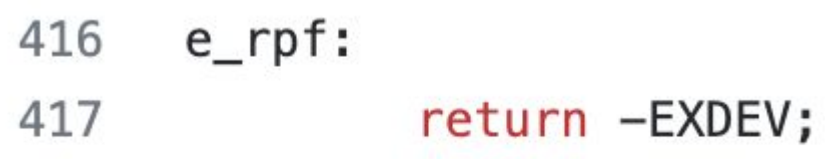


```
last_resort:
        if (rpf)
                goto e_rpf;
```

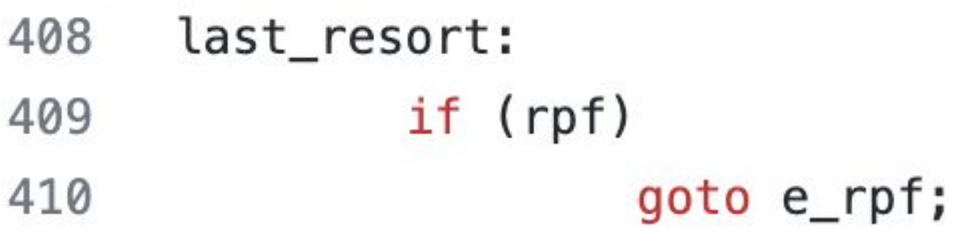


```
        if (no_addr)
                goto last_resort;
```

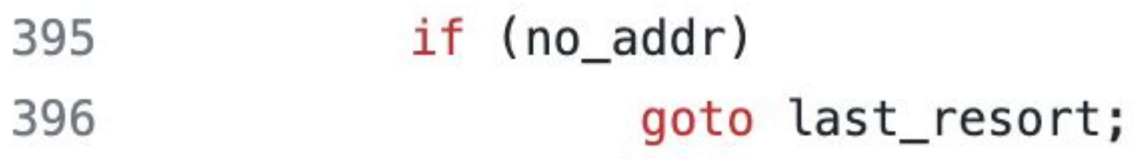
Let's have a look



```
e_rpf:
        return -EXDEV;
```

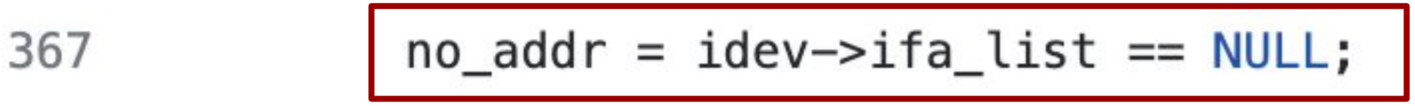


```
last_resort:
        if (rpf)
                goto e_rpf;
```



```
        if (no_addr)
                goto last_resort;
```

Interface IP check is made after evaluating loose mode



```
        no_addr = idev->ifa_list == NULL;
```

ifa_list => List of IPs associated with device

But pod interfaces don't have IPs assigned

Let's test

```
$ ip route get 10.x.y.z from 10.a.b.c iif ens6
RTNETLINK answers: Invalid cross-device link
```

Expected, Let's now give ens6 a random IP unrelated to our network

```
$ ip addr add 192.168.1.1/32 dev ens6

$ ip route get 10.x.y.z from 10.a.b.c iif ens6
10.x.y.z from 10.a.b.c via 10.m.n.1 dev ens5
    cache iif ens6
```

We are hitting reverse path filtering because the pod interface has no IP…

- Recent versions of Cilium give it an IP
- If it has an IP, SYN are still dropped but conntrack sizes are consistent (and no martian packet warnings)
- We contributed a PR to make old IPs unreachable and send ICMP errors to clients
  https://github.com/cilium/cilium/pull/18505

Status

- DNS errors in Metrics Service on rollouts
- Node-local-DNS can't establish connections
- AWS conntrack for instance is saturated
- Alerting Engine is SYN-Flooding Metrics Service on rollouts
- Conntracks are not consistent because Reverse Path Filtering drops SYNs
- We hit Reverse Path filtering because of an edge case in the kernel

=> *Why do we have so many SYNs?*

Chapter 4: gRPC client configuration

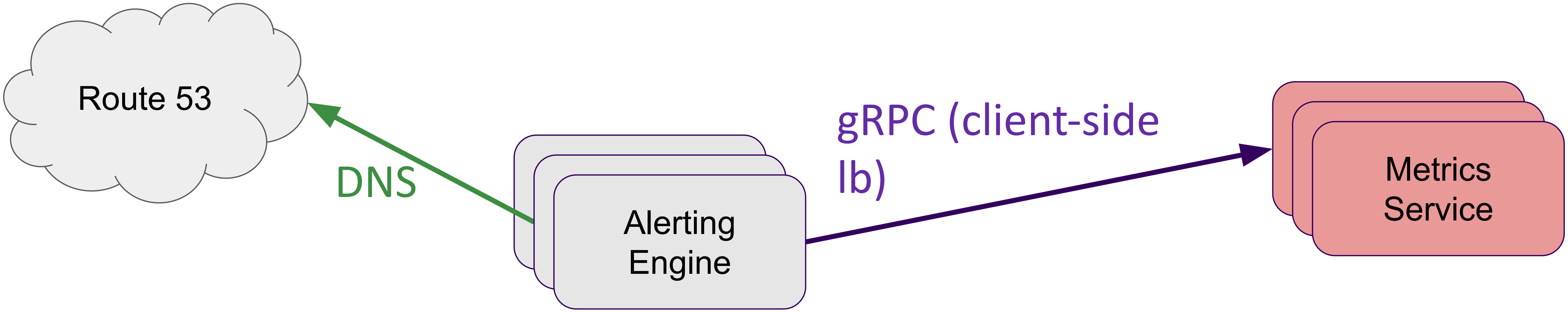
2 questions

1. Why were clients sending SYN requests for so long?
2. Why were clients sending SYN requests so frequently?

RPC setup

1. Service Discovery

2. Query



Route 53

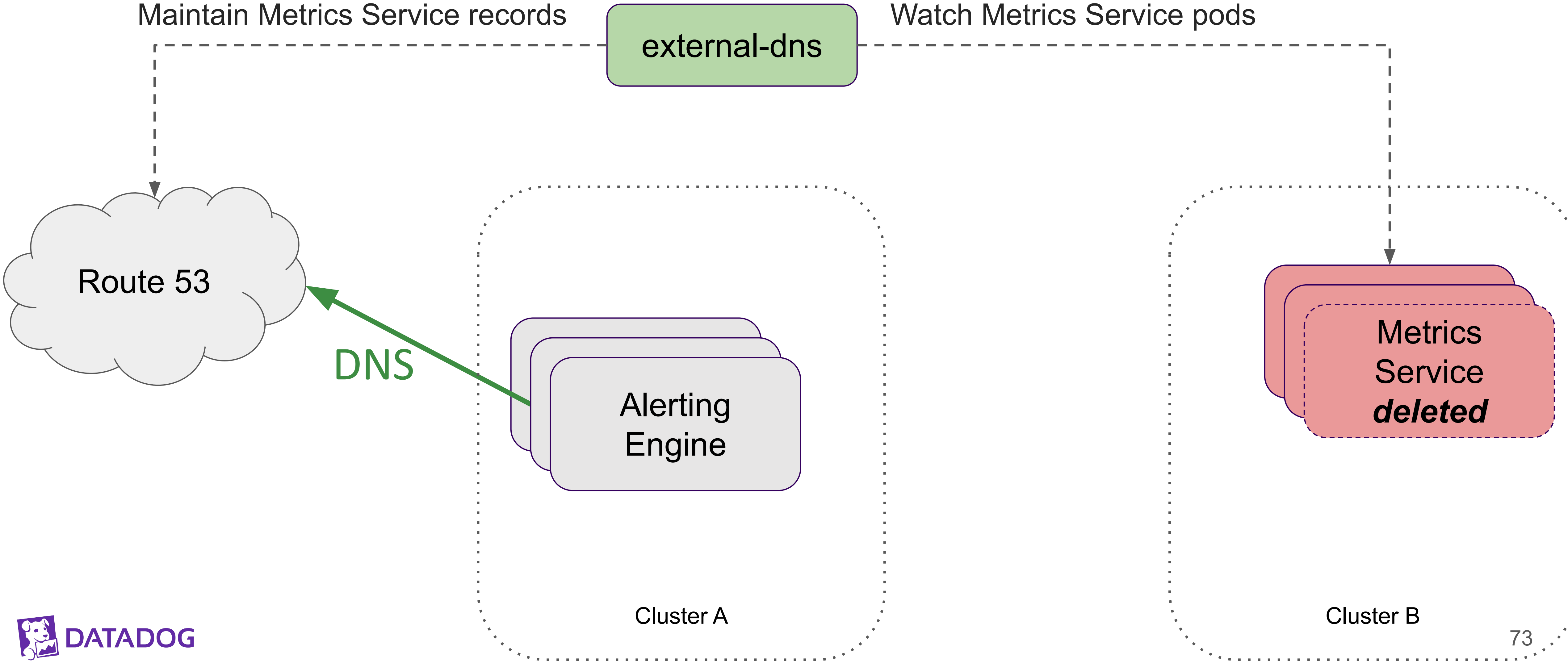
DNS

Alerting Engine

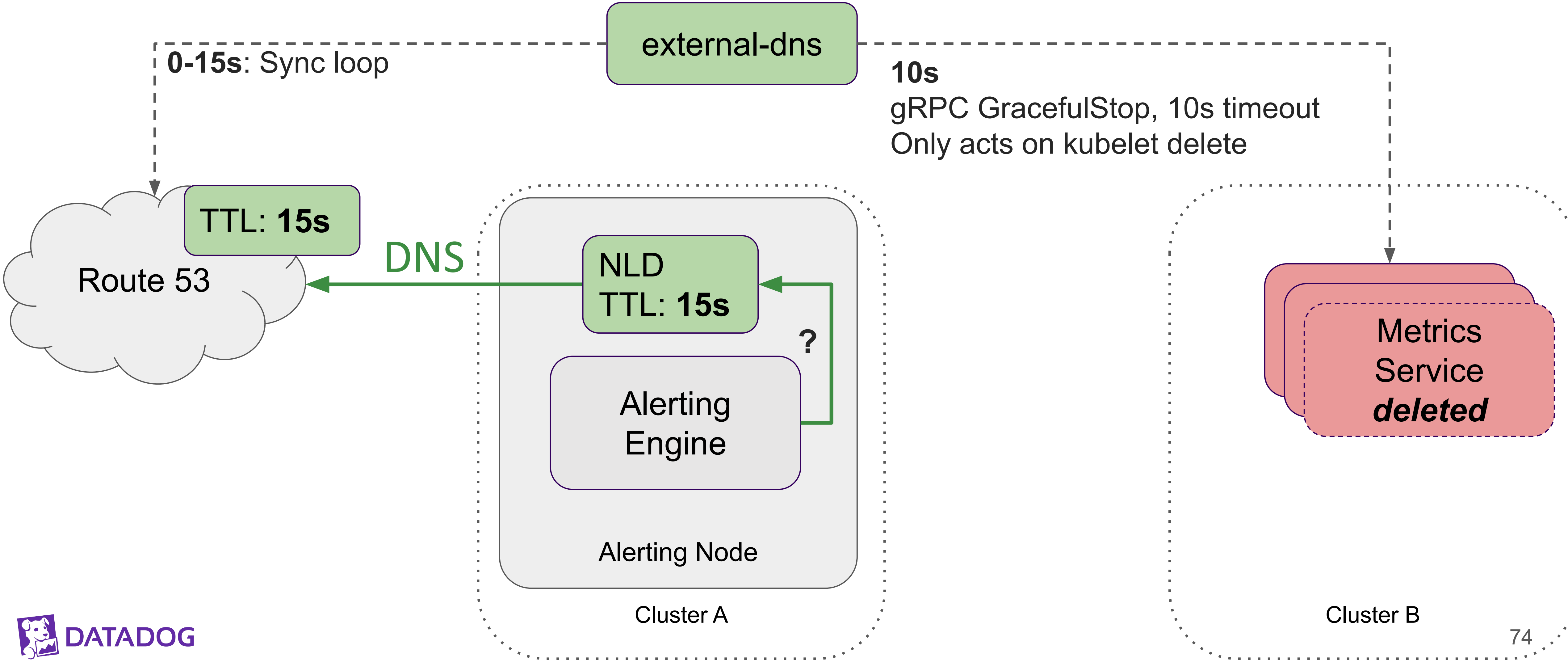
gRPC (client-side lb)

Metrics Service

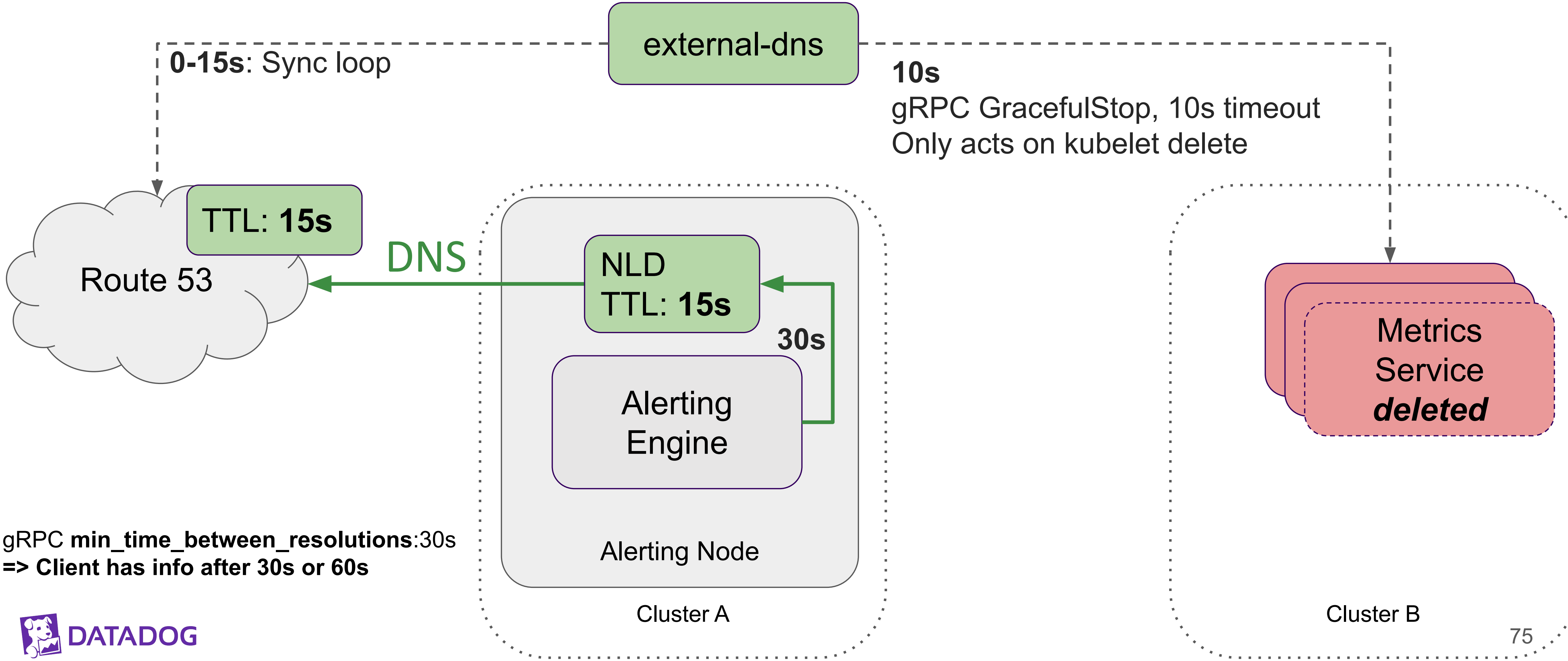
DNS propagation time during Rollouts


Maintain Metrics Service records
external-dns
Watch Metrics Service pods
Route 53
DNS
Alerting Engine
Cluster A
Metrics Service *deleted*
Cluster B

DNS propagation time during Rollouts

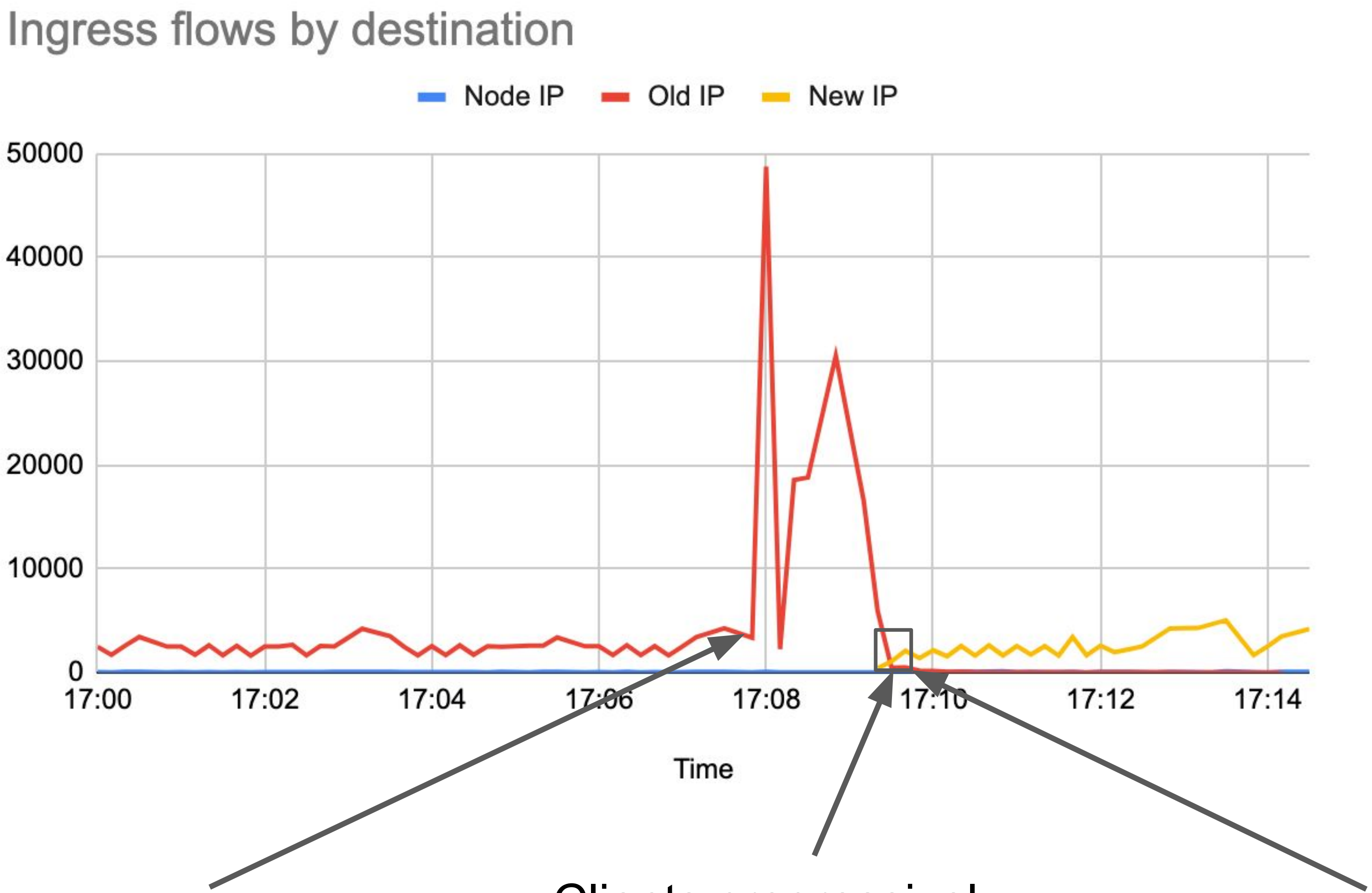


DNS propagation time during Rollouts



gRPC **min_time_between_resolutions**:30s
=> Client has info after 30s or 60s

DNS propagation time during Rollouts



Deletion starts

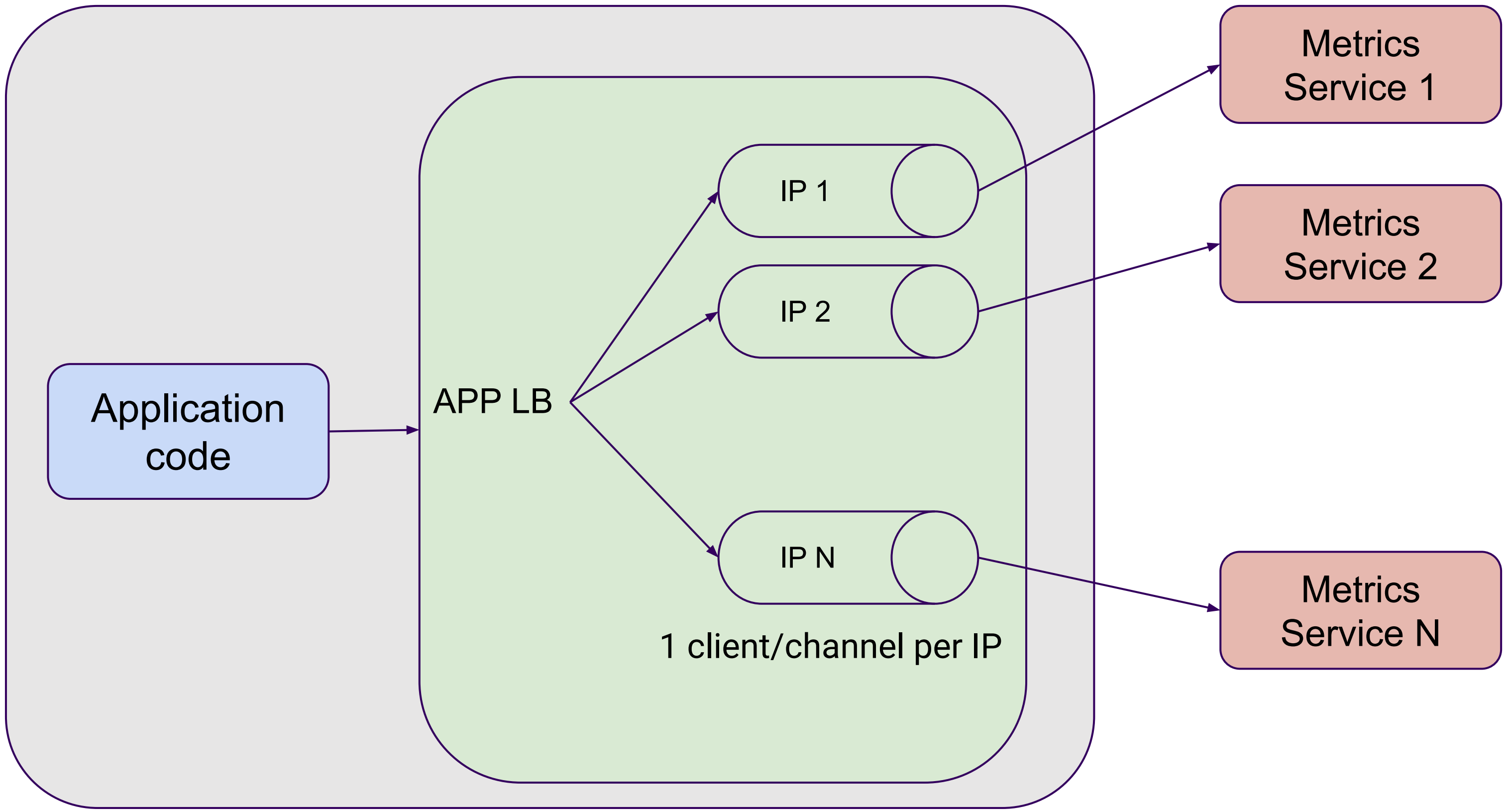
Clients progressively start using new IP

No client is using the Old IP

gRPC history at Datadog

- Originally, clients optimized for complex logic
  - DNS resolution in application code
  - One channel per backend IP
  - **pick_first** gRPC load balancing
- We changed the default to gRPC "standards"
  - Channels get a domain name and gRPC resolves
  - **round_robin** load balancing policy
  - This is when the issue started!

Alerting still had one channel per backend



Alerting Engine Pod

Reconnection differences

- pick_first and round_robin **have very different policies on connection failures**
    - pick_first: do not attempt to reconnect until the application asks for it
    - round_robin: automatically attempt to reconnect using reconnect options
- when using pick_first, we used **max_reconnect_backoff_ms=300 ms**
- ~reasonable for on-demand reconnects

Does it add up?

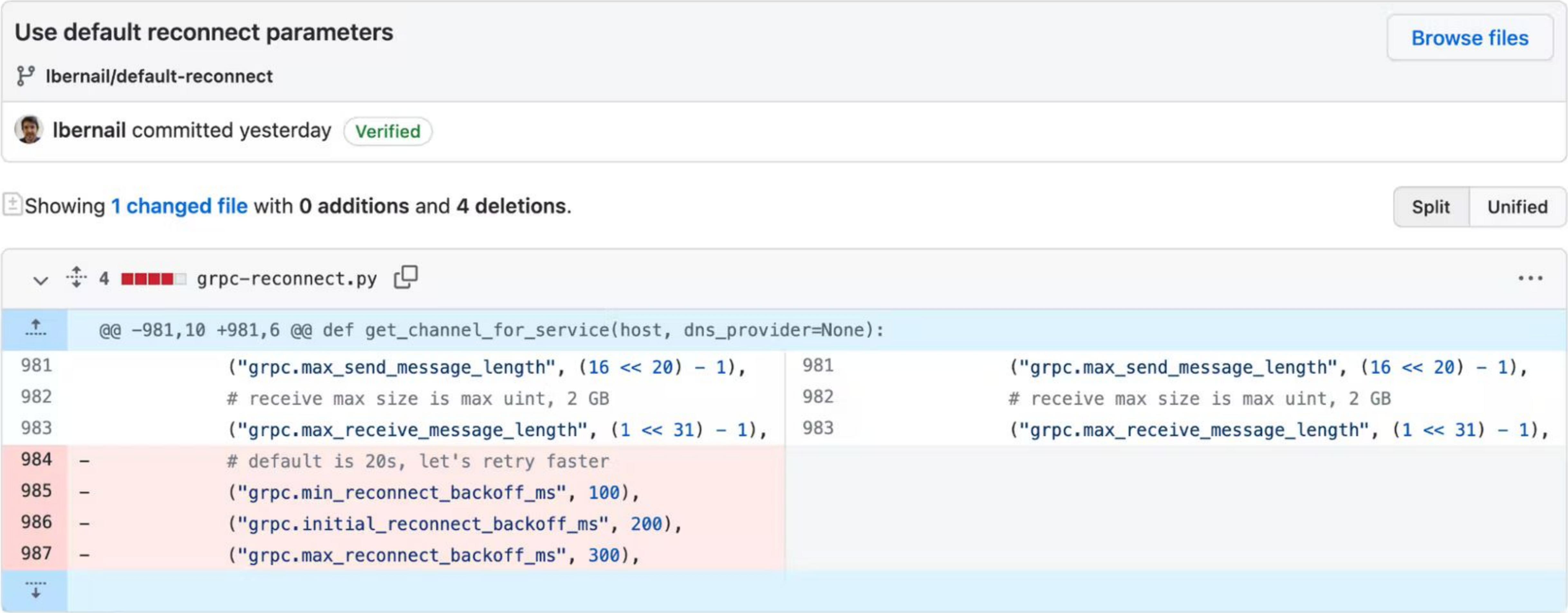
Alerting Engine * X000

reconnect every 0.3 s

= X0,000 SYN / sec

to each Metric Service Pod!

The fix



Use default reconnect parameters

lbernail/default-reconnect

lbernail committed yesterday Verified

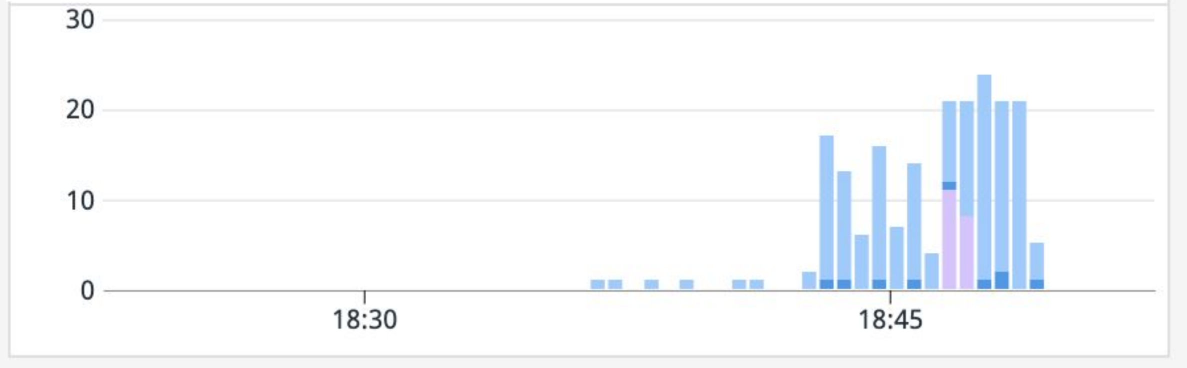
Showing 1 changed file with 0 additions and 4 deletions.

4 grpc-reconnect.py

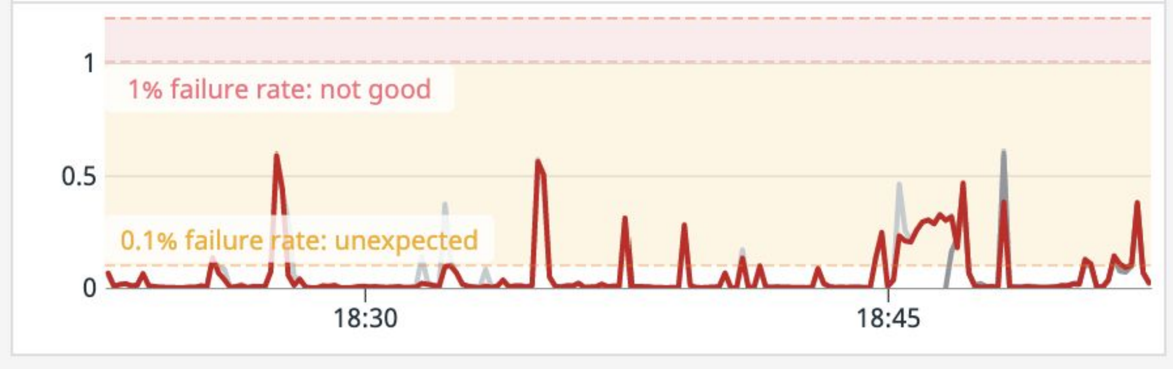
```
@@ -981,10 +981,6 @@ def get_channel_for_service(host, dns_provider=None):
              ("grpc.max_send_message_length", (16 << 20) - 1),
              # receive max size is max uint, 2 GB
              ("grpc.max_receive_message_length", (1 << 31) - 1),
-             # default is 20s, let's retry faster
-             ("grpc.min_reconnect_backoff_ms", 100),
-             ("grpc.initial_reconnect_backoff_ms", 200),
-             ("grpc.max_reconnect_backoff_ms", 300),
```

Finally

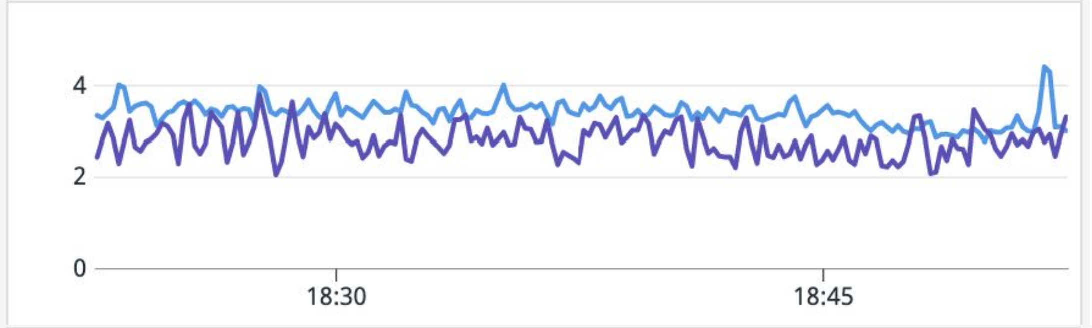
Metrics Service pod replacements (rollout)



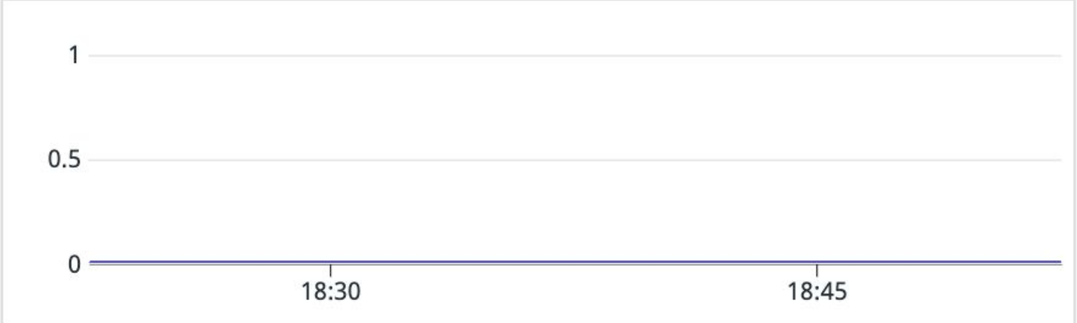
Metrics Service Error rate (**server** / **client**)



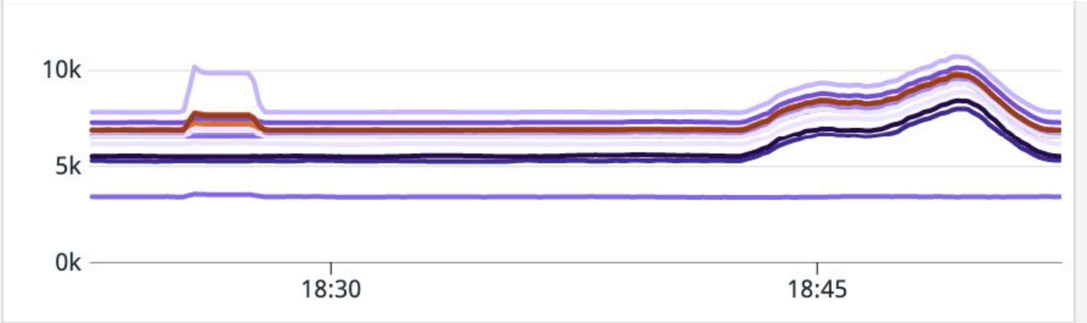
average DNS response time by Metrics Service pod (ms)



ENA Limits - conntrack exceeded



conntrack count for Alerting Engine



Lessons Learned

Lessons Learned

- Debugging this incident was long and painful but we learned a lot
- Sometimes it's not DNS
- Powerful abstractions leak in complex ways
- gRPC setup can be complex, making changes dangerous
- ENA metrics and VPC flow logs are extremely useful
- Required complex team efforts (thanks Laurent, Wendell, Matt, Nayef!)

Thank you

See the blog post for more details: dtdg.co/not-always-dns

We’re hiring: datadoghq.com/careers/

elijah@datadoghq.com
hemanth.malla@datadoghq.com

@elijahca
@hemanthmalla