KubeCon | CloudNativeCon

China 2021

*Virtual*

# Best Practice:
# DNS Failure Observability and Diagnosis in Kubernetes

Yuning Xie
Software Engineer
Alibaba Cloud

KubeCon | CloudNativeCon
China 2021
Virtual

# Agenda



DNS in Kubernetes

Why DNS is critical?

Built-in observabilities in CoreDNS

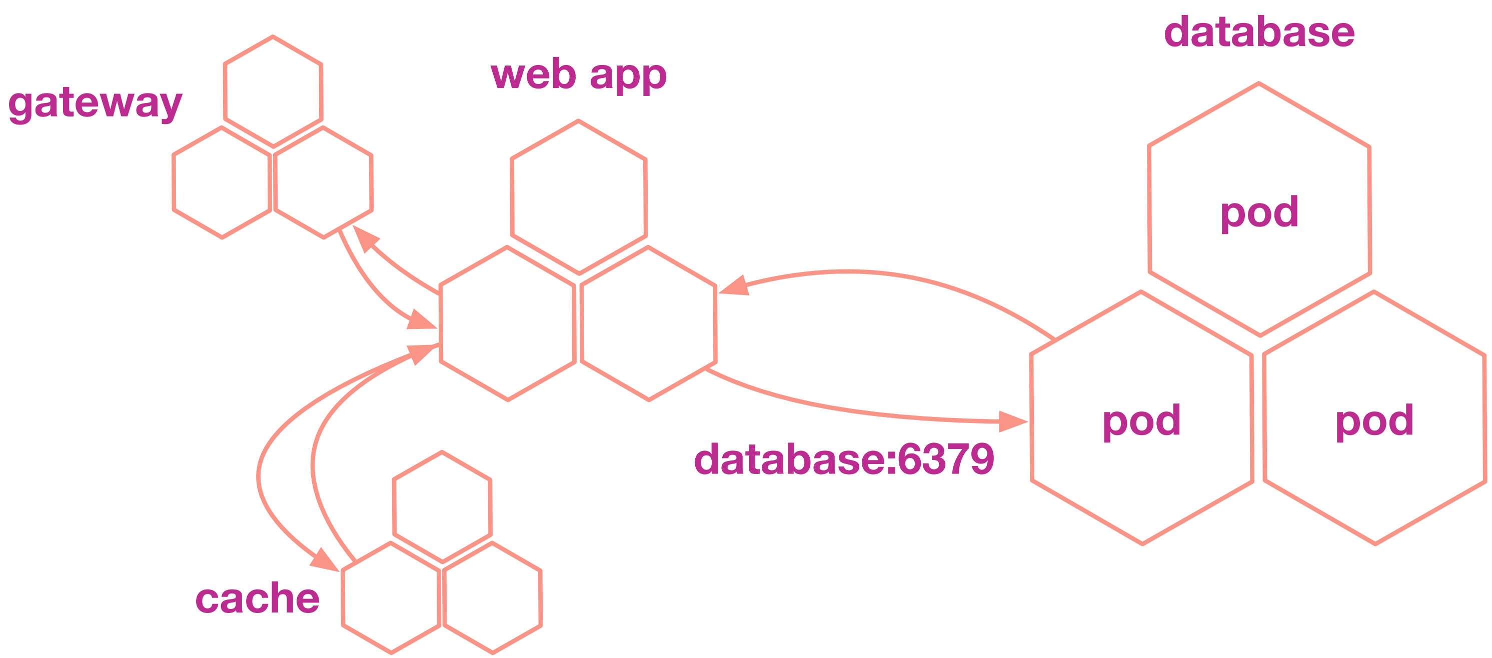
BPF-based Tools for diagnosis

An example of diagnostic procedure

# DNS in Kubernetes

**We need DNS when:**

- One microservice app talks to another
- Peer-to-peer comms in database cluster
- Monitoring, logging agents talks to API Server



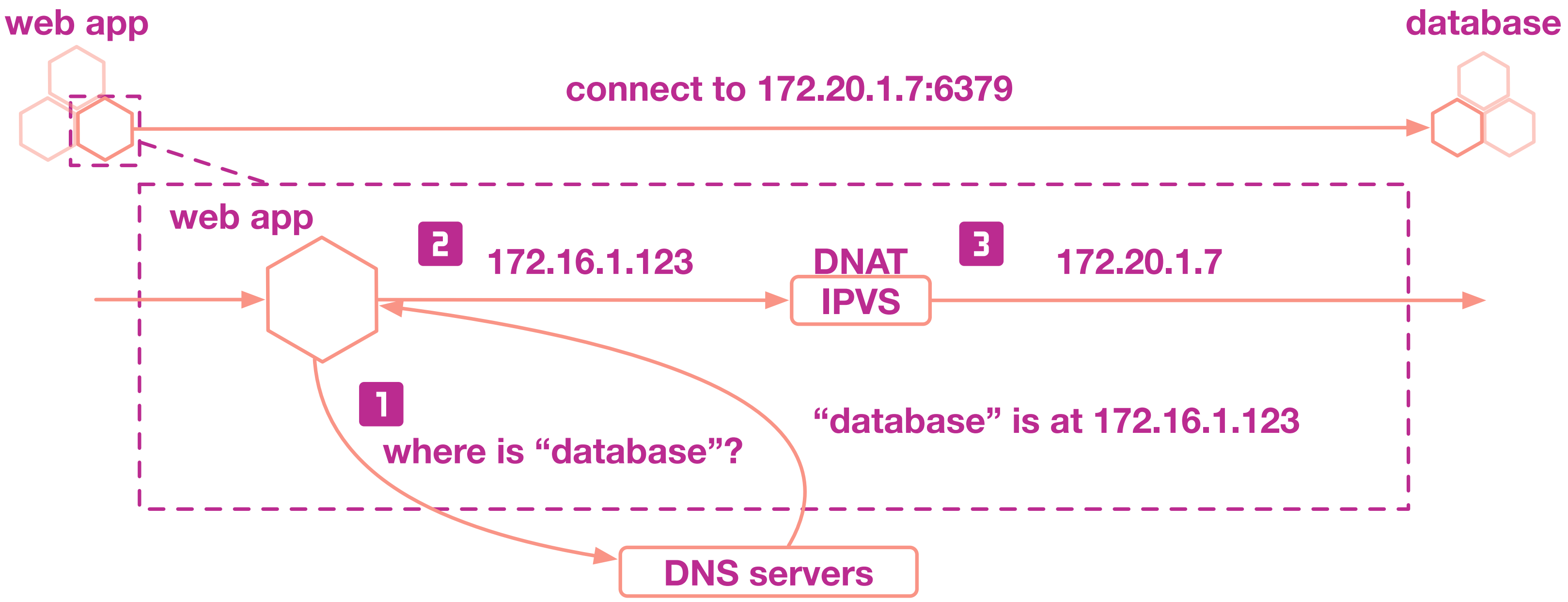
# Under the hood



**What it appears**
- A TCP connection established.

```
[root@web-app-0 /]# telnet database 6379
Trying 172.21.7.76...
Connected to database.
Escape character is '^]'.
```

**Under the hood**
- App only knows "database" :6379
- App must know IP address first



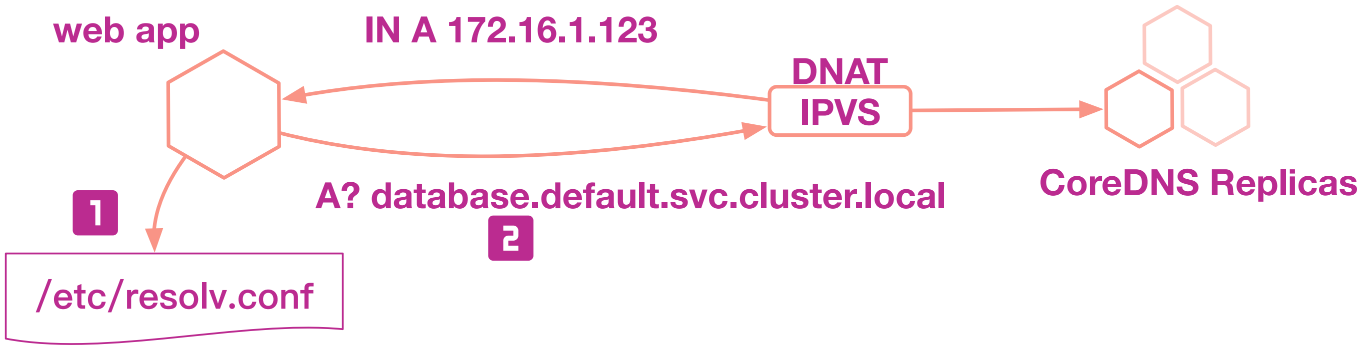
# Surprises



**How it works**

- Pods are configured to use in-cluster DNS servers
- The in-cluster DNS itself is a Kubernetes Service
- redundant search paths and ndots bring Service Discovery

```
[root@web-app-0 /]# cat /etc/resolv.conf
nameserver 172.21.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```




**Lots of DNS queries**

- half of them are wasted AAAA queries
- more queries if domain is not local, e.g. querying A of www.aliyun.com will cost 8 queries.

```
[root@web-app-0 ~]# tcpdump -nnn port 53
14:47:16.363701 IP 172.20.2.7.54526 > 172.21.0.10.53: 28204+ A? database.default.svc.cluster.local. (52)
14:47:16.363787 IP 172.20.2.7.54526 > 172.21.0.10.53: 2863+ AAAA? database.default.svc.cluster.local. (52)
14:47:16.364175 IP 172.21.0.10.53 > 172.20.2.7.54526: 28204*- 1/0/0 A 172.21.7.76 (102)
14:47:16.364184 IP 172.21.0.10.53 > 172.20.2.7.54526: 2863*- 0/1/0 (145)
```

# DNS becomes fragile



**DNS itself is simple and reliable, but**

- Almost every component needs DNS – Huge Blast Radius
- Each DNS query undergo too many components – Vulnerable
- For a single result of DNS query, app may trigger up to 8 queries – High QPS

DNS failures are becoming the top common issues in daily operations, but these failures' root causes vary.

# Resource limits



DNS is probably the busiest protocol inside K8s.

## CPU Starvation

**Phenomenon**: DNS delays under heavy load

**Reason**:
- DNS QPS is proportional to CPU limit

**Mitigation**: increase CPU limit or scale out CoreDNS replicas

## Conntrack Table Entries Starvation

**Phenomenon**: DNS failure under heavy load

**Reason**:
- Each Service access will consume a conntrack entry
- If short-lived connection is widely used, Conntrack Table will soon be full
- DNS cannot be queried if Conntrack Table is full

**Mitigation**: switch to long-lived connections or increase Conntrack Table size

# Conflicts



UDP is unreliable, from the day it was built.

## Conntrack drops UDP packets when source port conflicts

**Phenomenon**: intermittent DNS delays

**Reason**:
- DNS itself is a Service in K8s
- Parallel requests to a Service reusing a same source port will leads to race condition
- Many resolvers lookup A and AAAA in parallel, e.g., musl in alpine

**Mitigation**: upgrade kernel to 4.19+, kubernetes issue#56903

## IPVS drops UDP packets when source port conflicts

**Phenomenon**: intermittent DNS delays during CoreDNS rolling update or scaling in

**Reason**:
- IPVS does not expire connections after real server deleted
- When source port of an existing connection reused, UDP packets will be dropped

**Mitigation**: upgrade kernel to 5.9+, kubernetes issue#71514

# Bugs

Some legacy versions of CoreDNS have bugs.

## CoreDNS plugins stop working

**Phenomenon**: intermittent lookup failure for Headless Service and external domains

**Reason**:
- Mishandle of Kubernetes Update events causing watch panics, and stop updating caches, coredns issue#3860
- Autopath stop working and return NXDOMAINs for external domains, coredns issue#4366

**Mitigation**: upgrade CoreDNS to latest

DNS in Kubernetes

Built-in observabilities in CoreDNS

How do we observe these failure?

BPF-based Tools for diagnosis

An example of diagnostic procedure

# Built-In Observabilities



Luckily, CoreDNS is a DNS server that chains a lot plugins, including many observability tools.

| Observability | Plugin Name |
| --- | --- |
| Logging | log |
| | dump |
| | debug |
| | dnstap |
| Tracing | trace |
| Metrics | prometheus |

# Loggings - log



**Usage**: log enables query logging to standard output.

**Example**:

```
[root@linux ~]# kubectl -n kube-system logs coredns-8fdd9787b-2bjvz --timestamps | grep database
2021-10-26T07:43:24.507437718Z [INFO] 172.20.2.7:48376 - 51559 "AAAA IN
database.default.svc.cluster.local. udp 52 false 512" NOERROR qr,aa,rd 145 0.000135929s
2021-10-26T07:43:24.507477682Z [INFO] 172.20.2.7:48376 - 30565 "A IN database.default.svc.cluster.local.
udp 52 false 512" NOERROR qr,aa,rd 102 0.000174301s
```

**Note**:

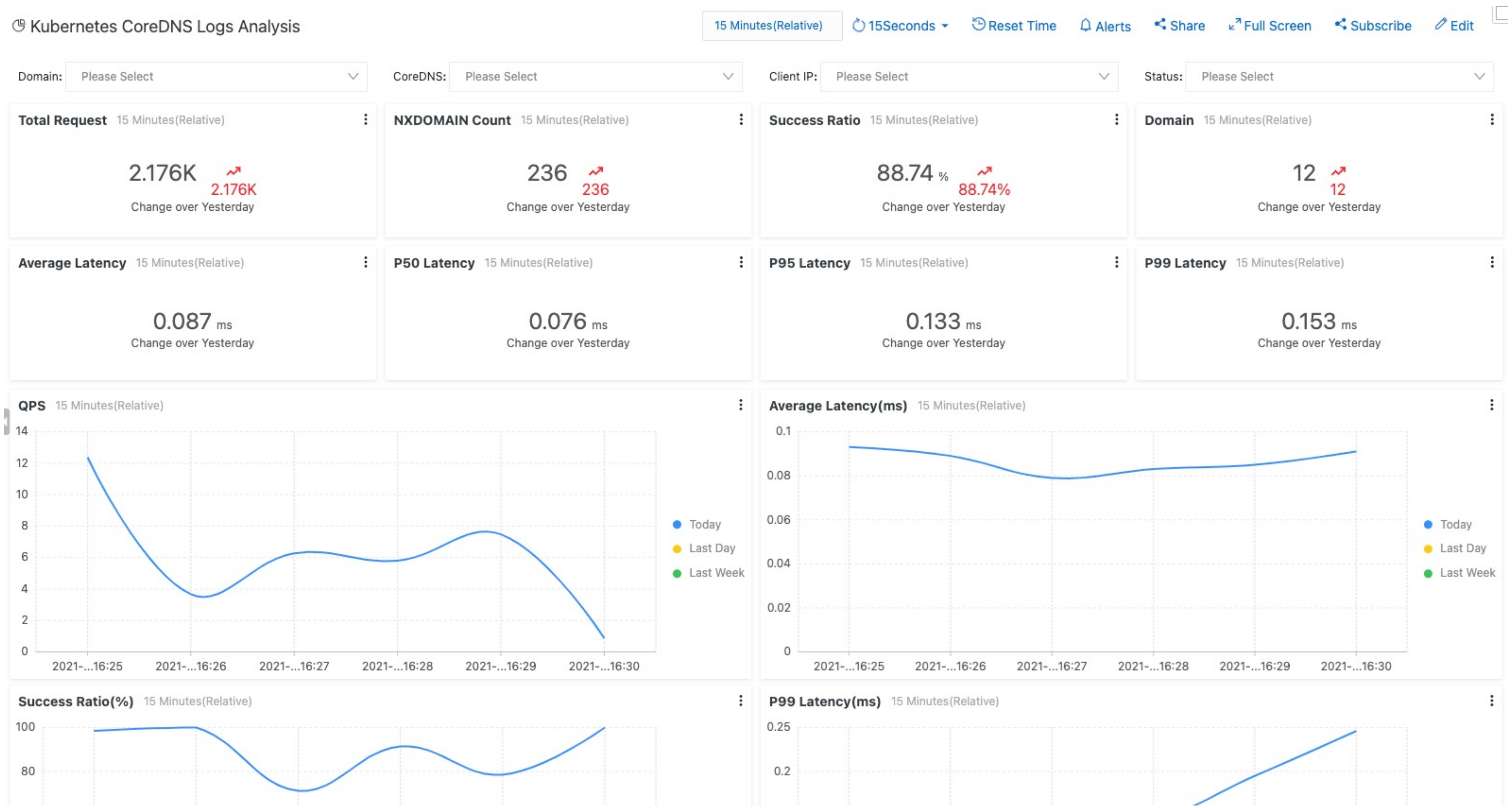
- Logs are printed on responding instead on receiving
- Timestamps can be fetched from "kubectl –timestamps"

**Use Cases**:

- Inspect a query's result at server-side
- Auditing cluster's outgoing domain queries, alert if any risky domains accessed

# Loggings - log

**Example**: Auditing and analysis using Logging Dashboard



# Loggings - dump



**Usage**: dump dumps all incoming queries on standard output.

**Example**:

```
[root@linux ~]# kubectl -n kube-system logs coredns-8fdd9787b-2bjvz --timestamps | grep database
192.168.6.110 - 9858 A IN database.default.svc.cluster.local. udp 41973
```

**Note**:
- Dump is an external plugin of CoreDNS, must recompile CoreDNS to use
- Dump print queries on receiving

**Use Cases**:
- If you believe DNS queries arrived at CoreDNS but not logs shown, you should try this

# Loggings - debug



**Usage**: debug disables the automatic recovery upon a crash so that you'll get a nice stack trace.

**Example**:

```
debug: 000000 00 0a 01 00 00 01 00 00 00 00 00 01 07 65 78 61
debug: 000010 6d 70 6c 65 05 6c 6f 63 61 6c 00 00 01 00 01 00
debug: 000020 00 29 10 00 00 00 80 00 00 00
debug: 00002a
```

**Note**:

- Probably not suitable for production environment

**Use Cases**:

- When upstream is sending strange replies, debug can output raw hex dump of DNS response

# Loggings - dnstap

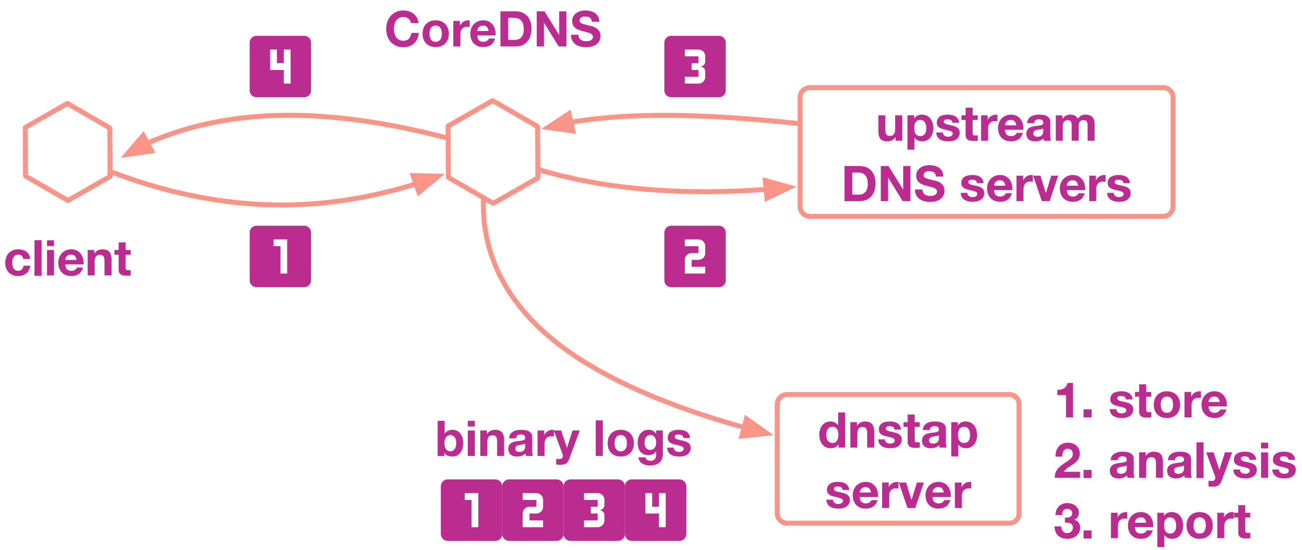
**Usage**: dnstap enables logging to dnstap.

**What is dnstap**: dnstap is a flexible, structured binary log format for DNS software. It uses Protocol Buffers to encode events that occur inside DNS software in an implementation-neutral format.

**How**:
1. CoreDNS send every DNS packets to dnstap server
2. Dnstap server store and analyze the packets
3. Dnstap can alert on any abnormal queries

**Advantages**:
- No performance degradation on DNS servers
  - asynchronous I/O
  - buffered circular queue
  - no disk writes
- No text parsing
- Extensible DNS message analysis

**Use Cases**:
- Locating the issues, CoreDNS or upstream
- Alternative for tcpdump and stdout logs



# Loggings - dnstap - how to analyze



**Response code(RCODE) is a signal**

- when the receiving RCODE is bad, we should alert

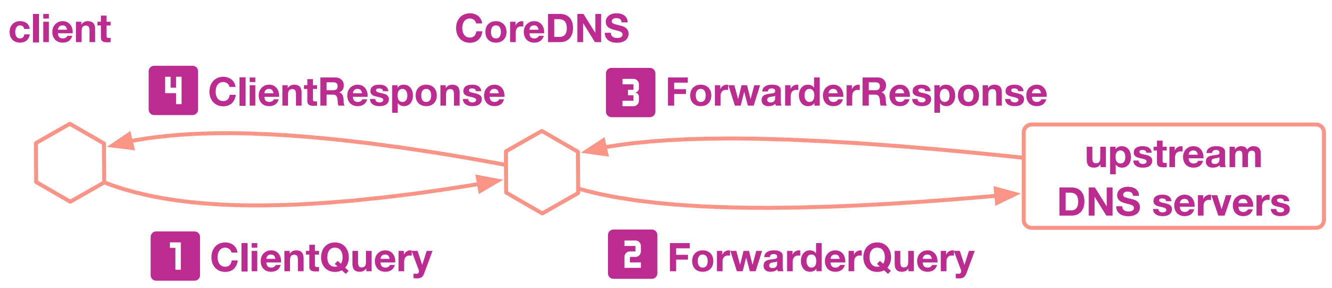
```
              DNS header format rfc1035

                                    1  1  1  1  1  1
      0  1  2  3  4  5  6  7  8  9  0  1  2  3  4  5
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                      ID                       |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |QR|   Opcode  |AA|TC|RD|RA|   Z    |   RCODE   |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    QDCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    ANCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    NSCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    ARCOUNT                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

| RCODE | Description |
|---|---|
| 0 | No error condition |
| 1 | Format error - The name server was unable to interpret the query. |
| 2 | Server failure - The name server was unable to process this query due to a problem with the name server. |
| 3 | Non-Existent Domain, or search path is incorrect |
| 5 | Query Refused |

# Loggings - dnstap - how to analyze



## Dnstap - Message Types




**Successful queries always come in pairs**

- the dnstap receiver stores dnstap in LRU cache
- we can know which part is causing the errors by the contexts

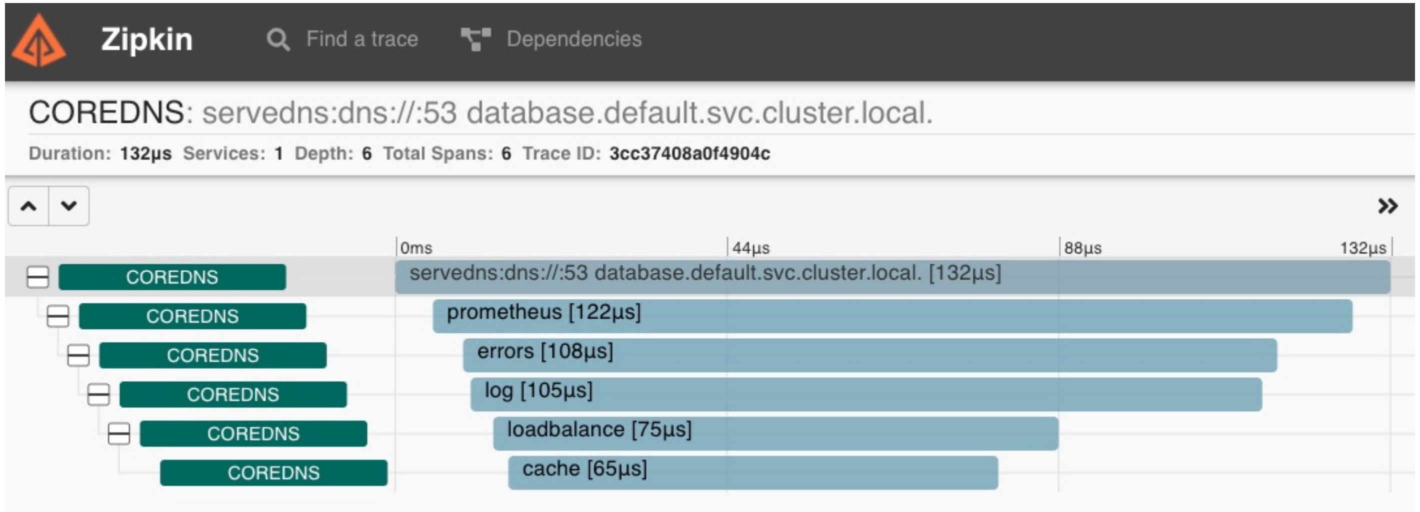
| Scenario | CLIENT_Q. | CLIENT_R. | FORWARDER_Q. | FORWARDER_R. |
|---|---|---|---|---|
| non-cached out of cluster domain query | yes | yes | yes | yes |
| cached out of cluster domain query | yes | yes | no | no |
| in-cluster domain query | yes | yes | no | no |
| cannot connect to upstream | yes | yes | yes | no |

# Tracing - trace



**Usage**: trace enables OpenTracing-based tracing of DNS requests as they go through the plugin chain.

**Example**:




**Note**:
- Similar to dnstap, trace requires additional OpenTracing servers

**Use Cases**:
- To aid diagnosis on which plugin is slowing system down

# Metrics - prometheus



**Usage**: prometheus enables Prometheus metrics.

**Example**:

```
[root@linux ~]# curl http://127.0.0.1:9153/metrics
# HELP coredns_cache_entries The number of elements in the cache.
# TYPE coredns_cache_entries gauge
coredns_cache_entries{server="dns://:53",type="denial"} 68
coredns_cache_entries{server="dns://:53",type="success"} 19
# HELP coredns_cache_hits_total The count of cache hits.
# TYPE coredns_cache_hits_total counter
coredns_cache_hits_total{server="dns://:53",type="denial"} 101302
coredns_cache_hits_total{server="dns://:53",type="success"} 96638
# HELP coredns_dns_request_duration_seconds Histogram of the time (in seconds) each request took.
# TYPE coredns_dns_request_duration_seconds histogram
coredns_dns_request_duration_seconds_bucket{server="dns://:53",type="A",zone=".",le="0.00025"} 99124
coredns_dns_request_duration_seconds_bucket{server="dns://:53",type="A",zone=".",le="0.0005"} 99312
coredns_dns_request_duration_seconds_bucket{server="dns://:53",type="A",zone=".",le="0.001"} 99382
```

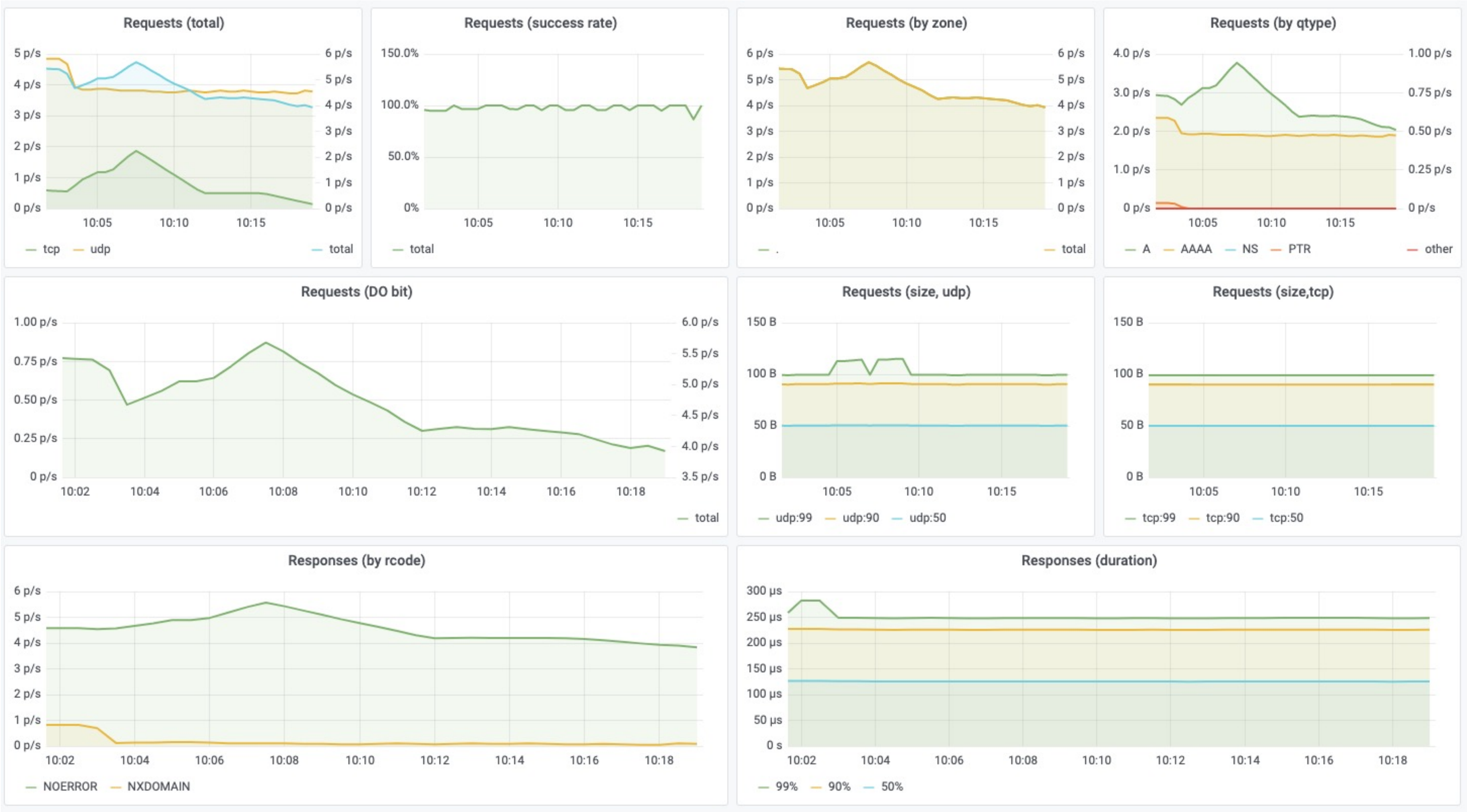
**Note**:
- You will need Prometheus to scape CoreDNS service at intervals

**Use Cases**:
- Provides statistical summaries about CoreDNS instances and DNS queries
- Alerts if any unexpected DNS RCODEs

# Metrics - prometheus

KubeCon CloudNativeCon China 2021 Virtual

**Example**: Monitoring and alerting using Prometheus and Grafana



DNS in Kubernetes

Built-in observabilities in CoreDNS

BPF-based Tools for diagnosis

What if failures happen at client-side?

An example of diagnostic procedure

# BPF-based Tools for diagnosis



**Homemade BPF-based DNS Observer and Problem Diagnoser?**

- UDP packets are not reliable by design
- BPF is great to locate the root cause

An example of locating DNS drops caused by IPVS drops UDP packets when source port conflicts and misconfiguration of iptables.

```
[root@linux ~]# bpftrace trace_dns_drop.bt
Attaching 8 probes...
Tracing DNS drops. Hit Ctrl-C to end.
TIME     PID      SKB              FUNC                                LADDR:LPORT          RADDR:RPORT
15:06:24 203751   ffff8b8fe531a300 ip_vs_remote_request4          172.20.6.28:38888     172.21.0.10:53
15:06:28 203884   ffff8b8fe4b9ee00 ip_vs_local_request4         192.168.0.198:38888     172.21.0.10:53
15:07:04 204343   ffff8b8f85603d00 ipt_do_table                 192.168.0.198:38888     172.20.6.51:53
15:07:05 204347   ffff8b8fa068fc00 ipt_do_table                   172.20.6.28:38888     172.20.6.51:53
15:08:09 205378   ffff8b8f85603400 ip_vs_local_request4         192.168.0.198:38888     172.21.0.10:53
15:08:10 205491   ffff8b8f85708e00 ip_vs_remote_request4          172.20.6.28:38888     172.21.0.10:53
```

Code at https://gist.github.com/xh4n3/61d8081b834d7e21bff723614e07777c

**Great tools**:

- BCC gethostlatency.py https://github.com/iovisor/bcc/blob/master/tools/gethostlatency.py
- Packet, where are you? https://github.com/cilium/pwru

DNS in Kubernetes

Built-in observabilities in CoreDNS

BPF-based Tools for diagnosis

An example of diagnostic procedure

How to deal with a real DNS incident?

# Diagnostic Procedure



An example of diagnostic procedure for DNS failures:

**Confirm**

- do not rush into conclusion too quickly
- config is desired and pods are running
- no change recently

**Collect**

- collect all possible info from coredns server, client, client's kernel
- make dig tests from different places to confirm the failure scope

**Verify**

- turn on CoreDNS's log, metrics, trace plugins to help diagnosis
- use BPF tools or tcpdump to verify where the packets dropped

**Fix**

- perform the fix gradually
- confirm result from metrics dashboard

# Diagnostic Procedure



A couple of questions to help:

| How often are failures? | Possible Cause | Next |
|---|---|---|
| Always | • config error<br>• network issue | • check coredns log |
| Only on peak hours | • resource limit on cpu, conntrack | • check resource usage on nodes<br>• check CoreDNS's metrics |
| Intermittently | • conflicts issue caused by port reuse | • use log plugin to locate cause first<br>• check coredns pod status<br>• check kernel version |

| How many nodes impacted? | Possible Cause | Next |
|---|---|---|
| Whole cluster | • misconfig at server-side<br>• kubernetes apiserver issue | • check coredns config |
| Only a number of nodes | • resource limit on cpu, conntrack | • check network connectivity<br>• check resource usage on nodes |

# Conclusion



- DNS in Kubernetes is critical but fragile.
- CoreDNS has many observability plugins.
- BPF is helpful for locating in-kernel DNS failures.
- Follow best practice when set up CoreDNS in kubernetes, use NodeLocal DNSCache.

Thank You!