OpenTelemetry Tracing

Not just for webapps

Pieter Lexis

cfgmgmtcamp 2026 - February 3, 2026

`whoami`

Pieter Lexis

Accidental breaker of ankles (unofficial)

- Senior PowerDNS Developer
- Sometimes software developer
- Sometimes system admin
- DNS Nerd
- cfgmgmtcamp visitor since forever

POWERDNS

Agenda

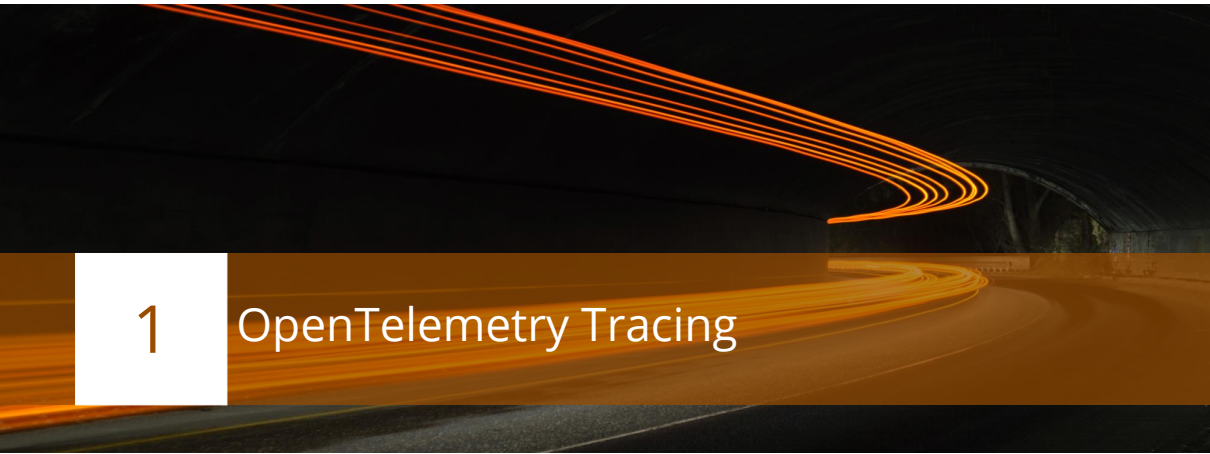
1. OpenTelemetry Tracing
2. PowerDNS Software
3. Tracing implementaion in PowerDNS
4. Using Tracing
5. Distributed Tracing in DNS Systems

Agenda

1. OpenTelemetry Tracing
2. PowerDNS Software
3. Tracing implementaion in PowerDNS
4. Using Tracing
5. Distributed Tracing in DNS Systems

Content warning

- C++ Code snippets
- IETF draft document snippets
- YAML



1 OpenTelemetry Tracing

OpenTelemetry (OTel)

- CNCF observability framework
- Open Source, Vendor Agnostic
- Telemetry data *signals*:
  - Metrics
  - Logs
  - Traces
- Defines transport mechanisms used by Collectors
- Has a large, existing software ecosystem

OpenTelemetry Traces

A *Trace* consists of one or more *Span*s.

A Span has

- A Name
- Optional Parent Span ID
- Start Timestamp
- End Timestamp
- Zero or More Attributes

Span Relationships

```
Causal relationships between Spans in a single Trace

        [Span A]  ←←←(the root span)
            |
     +------+------+
     |             |
 [Span B]      [Span C] ←←←(Span C is a `child` of Span A)
     |             |
 [Span D]      +---+-------+
               |           |
           [Span E]    [Span F]
```

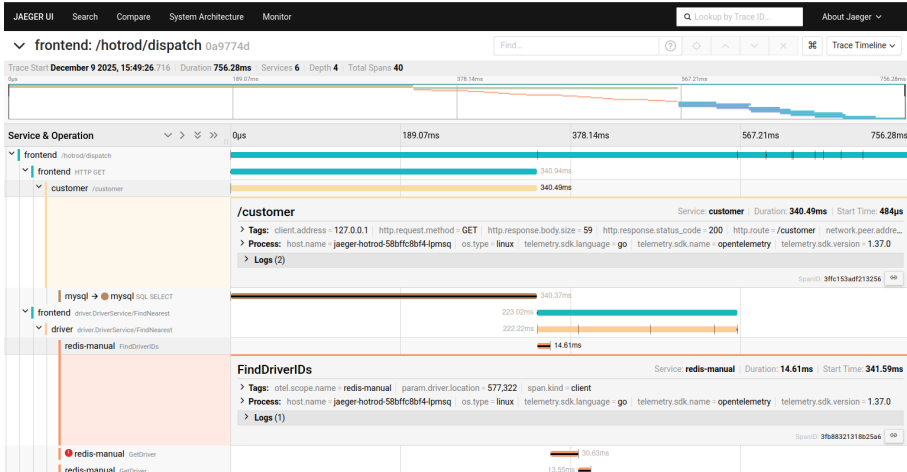
Sometimes it's easier to visualize **Traces** with a time axis as in the diagram below:

```
Temporal relationships between Spans in a single Trace

--|-------|-------|-------|-------|-------|-------|-------|-> time

 [Span A·····································]
   [Span B·····························]
      [Span D·····················]
    [Span C·····································]
         [Span E·····]        [Span F··]
```

POWERDNS

Visualizing Traces

Plethora of software available: Jaeger, `otel-tui`, Grafana's Tempo



2 PowerDNS Software

PowerDNS

- Three Open Source (GPLv2) DNS Applications:
  - Authoritative Server – database (and more)-backed DNS server
  - Recursor – DNS resolver with Lua hooks
  - DNSDist – high-performance, scriptable (Lua) DNS Swiss army-knife
    - Query routing
    - Query and response mangling
    - DDoS prevention (dropping queries with eBPF)
- **OLD!** – Initial GPL-licensed code check-in on November 27th 2002
- Written in C++, using (mostly) modern C++17 and some Rust

All kinds of special I

- All 3 applications live in the same git repository
- We have many specialized data structures
  - DNSName – Represents a name in DNS
  - ComboAddress – An IP address (v4 and v6)
  - Netmask – Represents a netmask
  - NetmaskTree – Binary tree of netmasks
- Wrappers to/from Lua

All kinds of special II

`ComboAddress` and `Netmask`

- Union of IPv4 and IPv6 address with optional port
- Easy manipulation

```
ComboAddress ca{"192.0.2.1:4500"};
foo.setPort(53);
auto bar = foo.toStringWithPort(); // "192.0.2.1:53"
auto quux = foo.isIPv4MappedIPv6(); // false
Netmask nm("192.168.2.0/24");
auto match = nm.match(ca); // false
```

All kinds of special III

NetmaskTree

Fast lookup binary tree of IP addresses

```
NetmaskTree<std::string> nmt;
nmt.insert(Netmask("192.0.2.0/27")).second = "hello";
auto node = nmt.lookup(ComboAddress("192.168.1.0")); // nullptr
node = nmt.lookup(ComboAddress("192.0.2.2")); // Returns the node at 192.0.2.0/27
auto nm = node->first; // nm is now a Netmask("192.0.2.0/27");
```

All kinds of special IV

DNSName

- Stores DNS names in wireformat
- Can check against other names

```
DNSName dn{"www.powerdns.com"}; // \x03www\x08powerdns\x03com\x00
auto isPartOf = dn.isPartOf(DNSName("com")); // true
dn.chopOff(); // dn is now \x08powerdns\x03com\x00
```

3 Tracing implementaion in PowerDNS

ProtoBuf in PowerDNS

- PowerDNS already uses ProtoBuf
  - Query logging
  - Some event logging
- Custom streaming protocol to a collector
- `protozero` for ProtoBuf (de-)serialization
- Can decode/encode all our special data types
- Low overhead

OpenTelemetry SDK[1]

A very complete implementation, but overkill for us.

- Depends on `libabsl` (abseil)
- HTTP submission via cURL
- Includes GRPC engine
- Uses Google's C++ ProtoBuf library
- Requires using data structures generated from the `.proto` file

[1] `https://github.com/open-telemetry/opentelemetry-cpp`

Implementing a Tracer and a `defer` I

- Go's `defer` is great!
- C++ has "Resource acquisition is initialization" (raii)
- We created an object (`Tracer`) that represents a trace
- When creating a Span, you get a `Closer` object
- The `Closer` destructor will close the Span

Implementing a Tracer and a `defer` II

```
class Closer
{
public:
  Closer(std::shared_ptr<Tracer> tracer, const SpanID& spanid) :
    d_tracer(std::move(tracer)), d_spanID(spanid) {};

  ~Closer() {
    if (d_tracer != nullptr) {
      d_tracer->closeSpan(d_spanID);
    }
  }
};
```

Implementing a Tracer and a `defer` III

```
Tracer::Closer Tracer::getCloser(const SpanID& spanid) {
  return {shared_from_this(), spanid};
}
```

Implementing a Tracer and a `defer` IV

```
void someFunc(/* removed */) {
  auto closer = tracer.getCloser(__func__);
  /* simplified loop */
  for (const auto& rule : rules) {
    // Automatically parents
    auto ruleCloser = tracer.getCloser("Rule: " + rule.d_name);
  }
}
```

4 Using Tracing

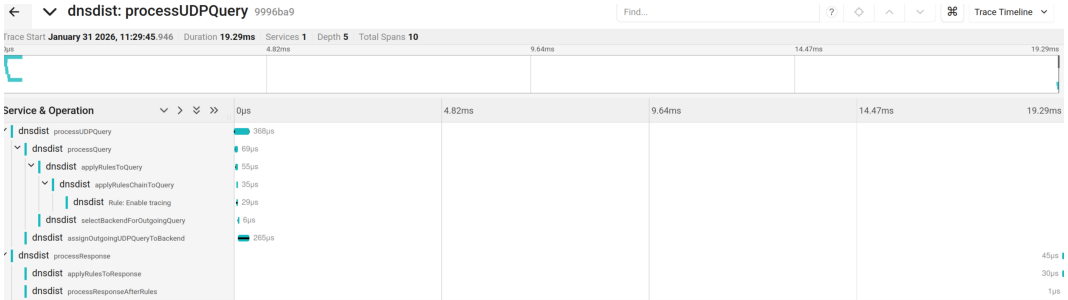
Configuration

```
# dnsdist
remote_logging:
  protobuf_loggers:
    - name: pblog
      address: 127.0.0.1:5301

query_rules:
  - name: Enable tracing
    selector:
      type: All
    action:
      type: SetTrace
      value: true
      remote_loggers:
        - pblog
```

```
# PowerDNS Recursor
recursor:
  event_trace_enabled: 4

logging:
  opentelemetry_trace_conditions:
    - acls:
        - 127.0.0.0/8
        - "::1"
  protobuf_servers:
    - servers:
        - "127.0.0.1:5301"
```

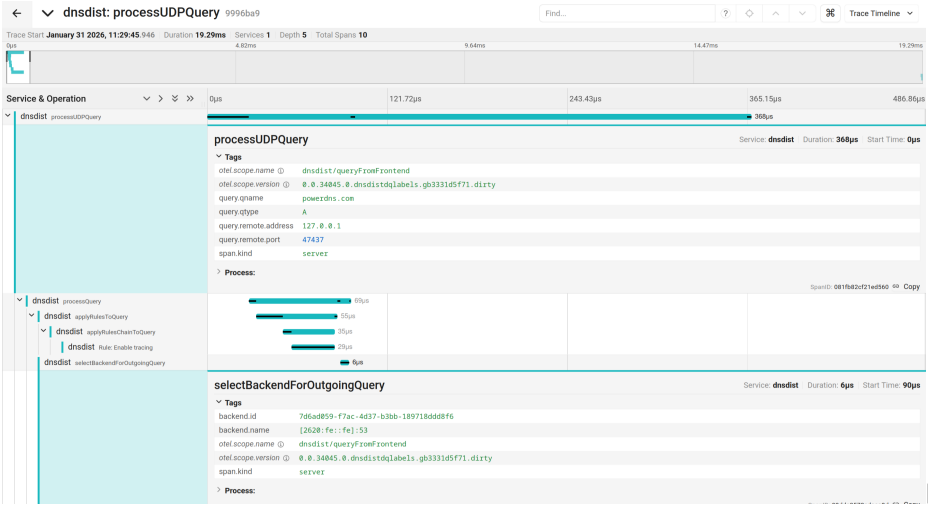
POWERDNS

What does it look like? I



POWERDNS

What does it look like? II



ERDNS

5 Distributed Tracing in DNS Systems

Passing trace information

- W3C has standardized HTTP Header (Traceparent[2])

[2] https://www.w3.org/TR/trace-context/#traceparent-header

Passing trace information

- W3C has standardized HTTP Header (Traceparent[2])
- DNS does not do do headers

[2] https://www.w3.org/TR/trace-context/#traceparent-header

Passing trace information

- W3C has standardized HTTP Header (Traceparent[2])
- DNS does not do do headers
- But we have EDNS Options

[2] https://www.w3.org/TR/trace-context/#traceparent-header

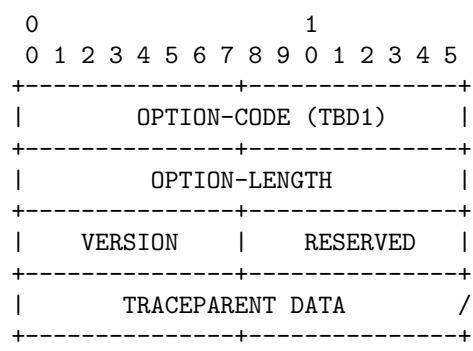
Passing trace information

- W3C has standardized HTTP Header (Traceparent[2])
- DNS does not do do headers
- But we have EDNS Options
- So let's just create a way to pass this info

[2] https://www.w3.org/TR/trace-context/#traceparent-header

The TRACEPARENT EDNS(0) Option[3]

The TRACEPARENT option has the following wire format:

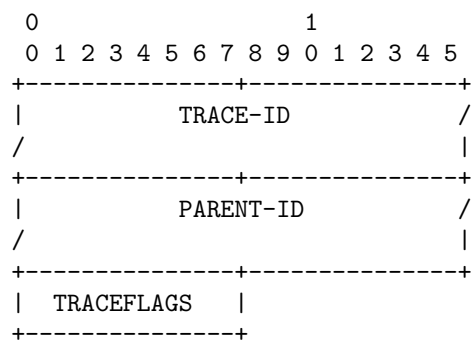


[3] https://github.com/PowerDNS/draft-edns-otel-trace-ids

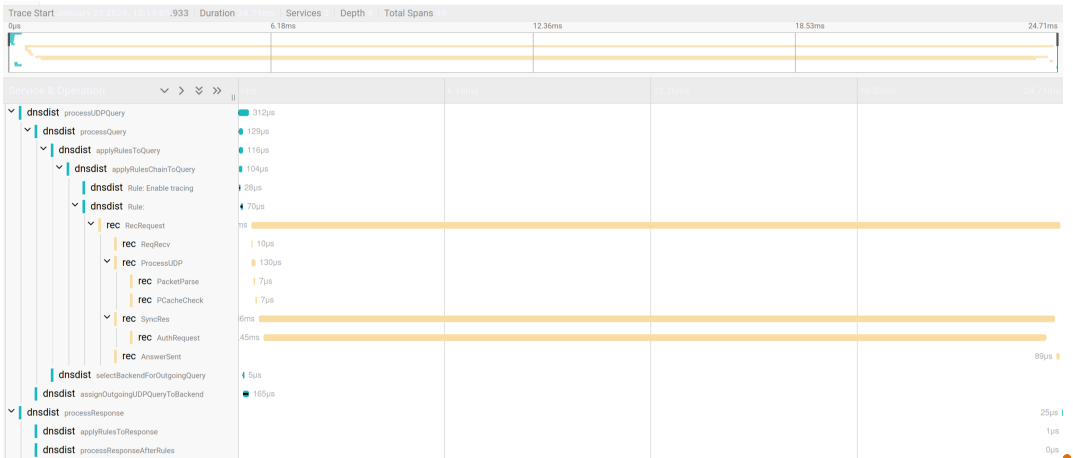
POWERDNS

The TRACEPARENT EDNS(0) Option

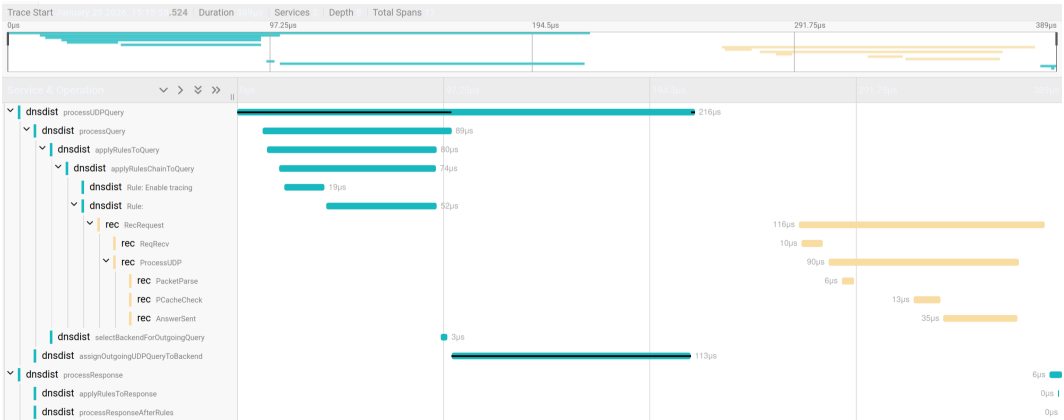
The TRACEPARENT DATA field for version 0 contains 3 fields: a 16 byte trace-id, an 8 byte parent-id, and a 1 byte trace-flags field. All these fields are MANDATORY.



Show, don’t tell



Show, don’t tell (answer in cache)



POWERDNS

Current Status

- Experimental Support in upcoming DNSDist 2.1.0
  - Generates traces
  - Can accept TRACEPARENT
  - Can pass TRACEPARENT to backends
- Experimental Support in upcoming Recursor 5.4.0
  - Generates traces
  - Can accept TRACEPARENT

Future Plans

- Tracing support in the Authoritative Server
- More functions being traced
- Finish TRACEPARENT “standard”
- Write DNS OpenTelemetry dictionary
- Feature to send Traces via HTTP
- Get other DNS vendors to implement this
- Better parenting and integration with OTel tooling

Thank you

Questions?

@lieter@mastodon.lieter.nl
@PowerDNS@fosstodon.org

@pieterlexis
@PowerDNS

POWERDNS