Introduction to Vector & Lessons Learned

dnstap ingestion, enrichment, and aggregation

John Todd - 26/27 October 2024 DNS-OARC Prague CZ v1

quad9

Non-goals/assumptions for sake of time

- Not a review of all possible dnstap logging/parsing systems - fluent bit, fluentd, logstash, dtap, etc. Honorable mention: Denis Machard's **go-dnscollector** is amazingly useful and dnstap-focused.
- Not a review of all backend storage (logstash, redis, clickhouse, etc.)
- Not an introduction to dnstap
- You know what a TSDB is (even better, you run Prometheus)
- You have a lot of free time to get pulled into a deep rabbit hole

The Problem:

How to interpret network and application current conditions & long-term behaviors with too much data to consume in a limited-resource environment?

Partial answer: dnstap!

Sub-problems:

- **Too much of it**
- **Each event is an island**
- **Privacy concerns**
- **Not complete enough (layer 3 context missing)**

Producers of dnstap

- dnsdist, PDNS-recursor, Unbound, BIND
- front-end (client) traffic
- back-end (recursive-to-authoritative) traffic
- block traffic (we treat as separate dnstap output)
- at the core: need a kafka reader

Consumers of dnstap (or summaries)

- kafka
- clickhouse
- redis summary engine(s)
- files (for blocking summaries)
- prometheus scraping

Requirements

So we need:

- aggregation of dnstap streams

- sampling & aggregation of DNS events

- transformation of dnstap data into “something else” (kafka, protobuf, files, etc.)

- ability to extensively apply logic to dnstap data to exclude, modify, enrich events

Constraints:

- run at the edge of the network - small memory/CPU footprint

- simple to configure

- scales well horizontally & vertically for larger sites

VECTOR
BY DATADOG

https://vector.dev/

Vector

Observability pipeline construction

- It is a protocol adapter - convert A to B
- It performs transformations - re-write, edit, delete, split events
- It is an enrichment tool - MMDB, CSV, DNS lookups
- It is an aggregation tool - summarize many events into one metric
- Sampling and “circuit breaker” throttles to prevent overloads
- It does more than it should, probably

Vector

Observability pipeline construction - transforms

From most-to-least used in our configurations:

- Remap
- Aggregate
- Log-to-metric
- Metric-to-log
- Sample
- Throttle
- Filter
- Lua
- Reduce
- Route
- Tag cardinality limit

Formats for sources / sinks <deep breath>

AMQP, Apache Metrics, AppSignal, AWS ECS Metrics/SQS/SNS/S3/ Cloudwatch/Kinesis Firehose, Axiom, Azure Blob Storage, Azure Monitor logs, **ClickHouse**, Console, Datadog agent/logs/metrics, Databend, **dnstap**, Docker logs, Elasticsearch, EventStore DB, Exec(stdout), File, File descriptor, Fluent, GCP PubSub, GCP Chronicle Unstructured, GCP Cloud Monitoring, Greptime DB logs, GreptimeDB metrics, Heroku Logplex, Honeycomb, Host metrics, HTTP Client, HTTP Server, Humio Logs, Humio Metrics, InfluxDB metrics, InfluxDB logs, Internal logs, Internal metrics, JournalD, Kafka, Kubernetes logs, Logstash, Loki, Mezmo/LogDNA, MongoDB metrics, MQTT, NATS, NGINX metrics, New Relic, OpenTelemetry, PostgreSQL metrics, Prometheus Pushgateway, Prometheus remote write, Prometheus scrape, Pulsar, Redis, Sematext logs, Sematext metrics, **Socket client (protobuf)**, Splunk HEC, Static metrics, StatsD, stdin, Syslog, WebHDFS, Websocket, Vector

Vector

Actually two different large codebases - all in Rust

Vector:

- handles source/sink/transforms

VRL:

- **V**ector **R**emap **L**anguage - forms the core of the "remap" function, with ability to perform transformations on "event" data (aka: json objects.) This is where the heavy lifting is done.

They are closely tied together but are distinct.

Maslow's hammer?

"it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail."

So it's a log parser - is it really useful for dnstap at scale?

Yes.

But.

But.

It needed some extensions.
It was not very DNS-aware.

DNS-specific abilities / functions

Work funded by Quad9

- punycode encoder/decoder/validator, optional 0x20 un-uppercasing syntax, sieve to replace invalid chars
- lowercasing support in dnstap natively
- Extended MMDB support for geoIP (and other) lookup files with arbitrary fields
- dnstap over TCP and PowerDNS-style protobuf ingestion
- Extended QTYPE/EDNS expansions (still not complete)
- Much wider range of aggregations (sum, max, mean, last, etc.)

More DNS-specific abilities / functions

Work funded by Quad9

- Mozilla PSL enrichment (with custom PSL file support)
- dns_lookup to perform queries and include results
- additional GraphViz inclusions for self-documentation support
- *Coming soon!* Local memory-based caching across threads.

Bonus nice features we use frequently

- Graphviz - self-documenting, self-drawing configurations
- VRL sandbox for testing json parsing logic
- Extremely descriptive error messages
- Terminal-based “vector top” view shows realtime in/out events/bytes per component

Some fixed-width text examples

```
root@vector-test:/home/jtodd# cat /etc/vector/dnstap-demo.yaml
# simplest possible Vector dnstap configuration example:
sources:
  dnstap-from-dnsdist:
    type: "dnstap"
    mode: tcp
    address: "192.168.70.176:6000"
    permit_origin: ["192.168.70.0/24"]
    multithreaded: true

sinks:
  send-to-local-file:
    type: file
    inputs: ["dnstap-from-dnsdist"]
    path: /tmp/dnstap-data-%Y-%m-%d.log
    encoding:
      codec: "json"
```

```
root@vector-test:/etc/vector# vector -c dnstap-demo.yaml
2024-10-10T02:39:03.835902Z  INFO vector::app: Log level is enabled. level="info"
2024-10-10T02:39:03.836927Z  INFO vector::app: Loading configs. paths=["dnstap-demo.yaml"]
2024-10-10T02:39:03.843157Z  INFO vector::topology::running: Running healthchecks.
2024-10-10T02:39:03.843264Z  INFO vector: Vector has started. debug="false" version="0.41.1"
arch="x86_64" revision="745babd 2024-09-11 14:55:36.802851761"
2024-10-10T02:39:03.843315Z  INFO vector::app: API is disabled, enable by setting `api.enabled`
to `true` and use commands like `vector top`.
2024-10-10T02:39:03.843315Z  INFO vector::topology::builder: Healthcheck passed.
2024-10-10T02:39:03.843337Z  INFO source{component_kind="source" component_id=dnstap-from-dnsdist
component_type=dnstap}: vector::sources::util::framestream: Listening.addr=192.168.70.176:6000
```

```
root@vector-test:/etc/vector# tail -1 /tmp/dnstap-data-2024-10-10.log |jq -r
{
  "dataType": "Message",
  "dataTypeId": 1,
  "host": "192.168.70.176:34718",
  "messageType": "ClientQuery",
  "messageTypeId": 5,
  "requestData": {
    "fullRcode": 0,
    "header": {
      "aa": false,
      "ad": true,
      "anCount": 0,
      "arCount": 1,
      "cd": false,
      "id": 11098,
      "nsCount": 0,
      "opcode": 0,
      "qdCount": 1,
      "qr": 0,
      "ra": false,
      "rcode": 0,
      "rd": true,
      "tc": false
    },
    "opt": {
      "do": false,
      "ednsVersion": 0,
      "extendedRcode": 0,
      "options": [
        {
          "optCode": 10,
          "optName": "Cookie",
          "optValue": "EFj6rns2C3g="
        }
      ],
      "udpPayloadSize": 1232
    },
    "question": [
      {
        "class": "IN",
        "domainName": "www.quad9.net.",
        "questionType": "A",
        "questionTypeId": 1
```

Enrichment

This line in a “remap”:

```
.asn = get_enrichment_table_record("asn", {"ip":  ."sourceAddress"})
```

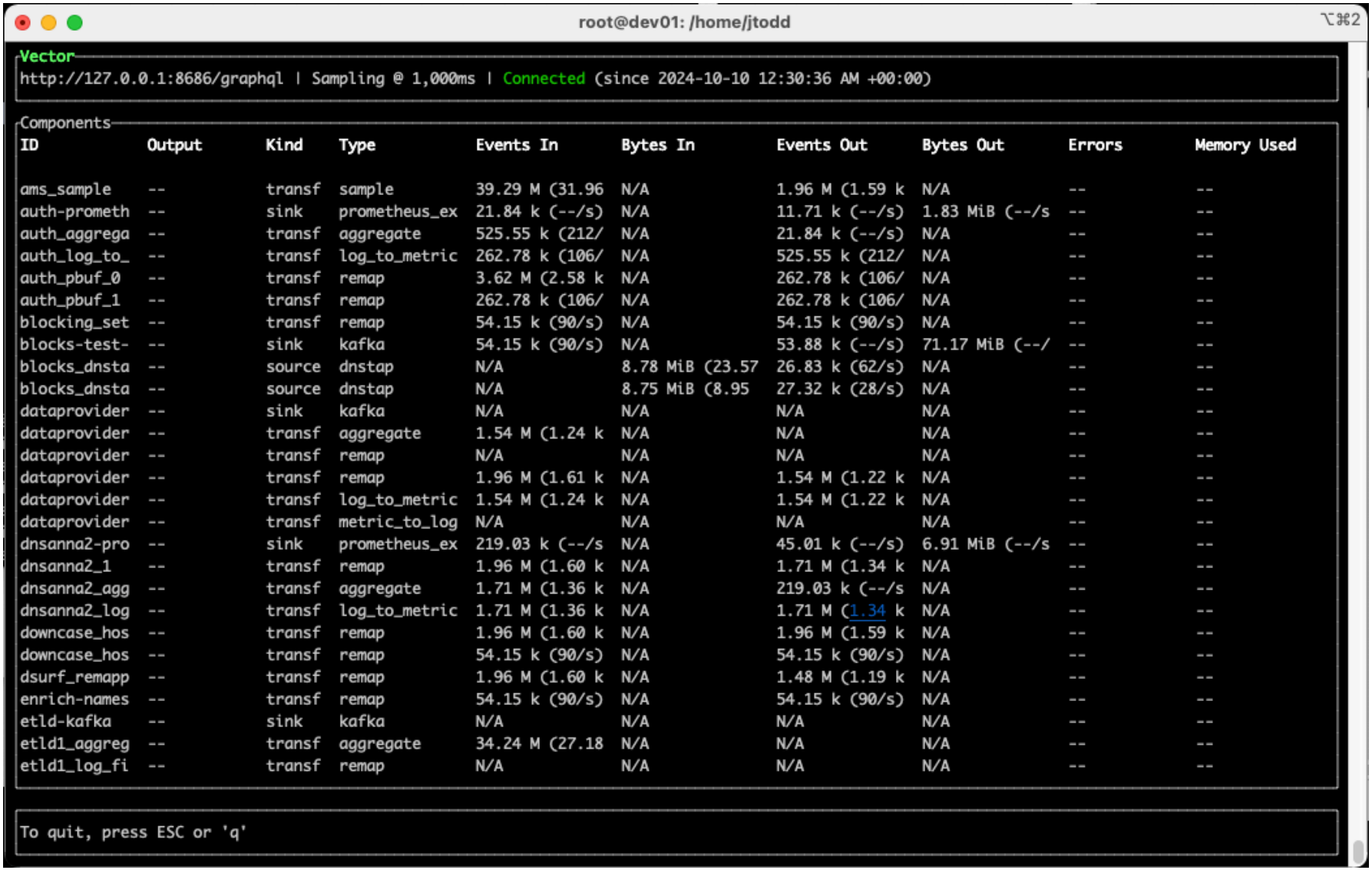
...results this addition to the dnstap object:

```
⋮
"asn": {
 "autonomous_system_number": 202125,
 "autonomous_system_organization": "Acacio Servicios Telematicos S.l.",
 "path": "42 202125",
 "prefix": "39.44.0.0/16"
 },
⋮
```

Enrichment -> Aggregation

Then perform a log_to_metric, then an aggregation, and finally send to a Prometheus sink.

```
root@vector-test:/etc/vector# curl http://127.0.0.1:8332/metrics
 ⋮
dnsanna2_asn{asn="33915",asn_name="Vodafone Libertel B.V.",
   country="NL",path="42 6830 33915",
   pop_code="ams",rcode="noerror"} 798720163 1728871787005
 ⋮
```


root@dev01: /home/jtodd
⌥⌘2
Vector
http://127.0.0.1:8686/graphql | Sampling @ 1,000ms | Connected (since 2024-10-10 12:30:36 AM +00:00)
Components
ID Output Kind Type Events In Bytes In Events Out Bytes Out Errors Memory Used
ams_sample -- transf sample 39.29 M (31.96 N/A 1.96 M (1.59 k N/A -- --
auth-prometh -- sink prometheus_ex 21.84 k (--/s) N/A 11.71 k (--/s) 1.83 MiB (--/s -- --
auth_aggrega -- transf aggregate 525.55 k (212/ N/A 21.84 k (--/s) N/A -- --
auth_log_to_ -- transf log_to_metric 262.78 k (106/ N/A 525.55 k (212/ N/A -- --
auth_pbuf_0 -- transf remap 3.62 M (2.58 k N/A 262.78 k (106/ N/A -- --
auth_pbuf_1 -- transf remap 262.78 k (106/ N/A 262.78 k (106/ N/A -- --
blocking_set -- transf remap 54.15 k (90/s) N/A 54.15 k (90/s) N/A -- --
blocks-test- -- sink kafka 54.15 k (90/s) N/A 53.88 k (--/s) 71.17 MiB (--/ -- --
blocks_dnsta -- source dnstap N/A 8.78 MiB (23.57 26.83 k (62/s) N/A -- --
blocks_dnsta -- source dnstap N/A 8.75 MiB (8.95 27.32 k (28/s) N/A -- --
dataprovider -- sink kafka N/A N/A N/A N/A -- --
dataprovider -- transf aggregate 1.54 M (1.24 k N/A N/A N/A -- --
dataprovider -- transf remap N/A N/A N/A N/A -- --
dataprovider -- transf remap 1.96 M (1.61 k N/A 1.54 M (1.22 k N/A -- --
dataprovider -- transf log_to_metric 1.54 M (1.24 k N/A 1.54 M (1.22 k N/A -- --
dataprovider -- transf metric_to_log N/A N/A N/A N/A -- --
dnsanna2-pro -- sink prometheus_ex 219.03 k (--/s N/A 45.01 k (--/s) 6.91 MiB (--/s -- --
dnsanna2_1 -- transf remap 1.96 M (1.60 k N/A 1.71 M (1.34 k N/A -- --
dnsanna2_agg -- transf aggregate 1.71 M (1.36 k N/A 219.03 k (--/s N/A -- --
dnsanna2_log -- transf log_to_metric 1.71 M (1.36 k N/A 1.71 M (1.34 k N/A -- --
downcase_hos -- transf remap 1.96 M (1.60 k N/A 1.96 M (1.59 k N/A -- --
downcase_hos -- transf remap 54.15 k (90/s) N/A 54.15 k (90/s) N/A -- --
dsurf_remapp -- transf remap 1.96 M (1.60 k N/A 1.48 M (1.19 k N/A -- --
enrich-names -- transf remap 54.15 k (90/s) N/A 54.15 k (90/s) N/A -- --
etld-kafka -- sink kafka N/A N/A N/A N/A -- --
etld1_aggreg -- transf aggregate 34.24 M (27.18 N/A N/A N/A -- --
etld1_log_fi -- transf remap N/A N/A N/A N/A -- --
To quit, press ESC or 'q'

Performance

Example lab system

- 16 cores @ 2.2Ghz (E5-2660) w/16G 1333 Mhz RAM (circa 2012 - old!)
- 60k ingest events per second
- 60% CPU utilization avg.
- 6 aggregations, 23 remaps, 9 log-to-metrics - VERY heavy transforms
- RAM usage: depends on aggregations, but this one used 12G steady-state (could be as little as 1g)

So what did we learn?

- Sample inside of dnstap (use “ProbaRule” function)
- Convert JSON into simpler protobuf for long-haul transport (CBOR? maybe in the future!)
- dns_lookup is fatally slow; sample or cache (new feature pending)
- time aggregates for as long as possible (hours?)
- we had some delay to get more DNS-specific features added
- as always, error management takes up more time than you think it will

How did it help?

- Replaced years of ossified Go-based code - four different repositories
- All configuration-based - no compile needed for logic changes
- Sped up ingestion pipeline 5x or more
- Very simple binary distribution and installation - no dependencies
- One person, ~2 months part-time - retrofit of all “moving parts”
- Connected to existing Kafka stream-based model - smooth migration by emulating older code outputs

...but it isn't perfect.

- Still crashes (locks up, but does not halt) with high volumes every few days - working on this bug (only 1 of our systems has this issue)
- DNSTAP ingestion takes a lot of CPU (bug? or just very non-optimized?)
- a few missing functions - ring buffers, notably
- still not "1.0" yet, despite ~6 years of production use by community
- user community is robust, but mostly asks questions. Fewer devs.
- timber.io was bought by Datadog but no commercial support

Things we built to help

- mrt2mmdb - convert full BGP table to MMDB-formatted database
- filter.py - strip MMDB files of non-essential data to take less memory

end

jtodd@quad9.net