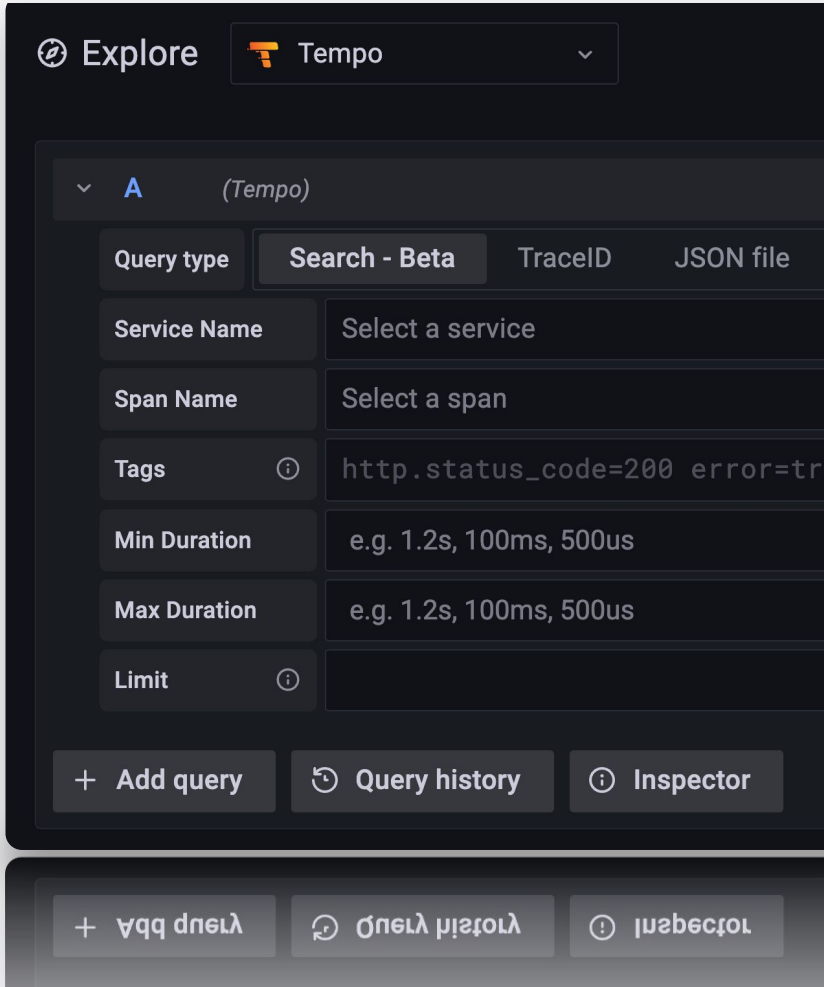
Distributed Tracing with Tempo and TraceQL

Suraj Sidh
Senior Software Engineer, Tempo



Suraj Sidh
Senior Software Engineer
@electron0zero

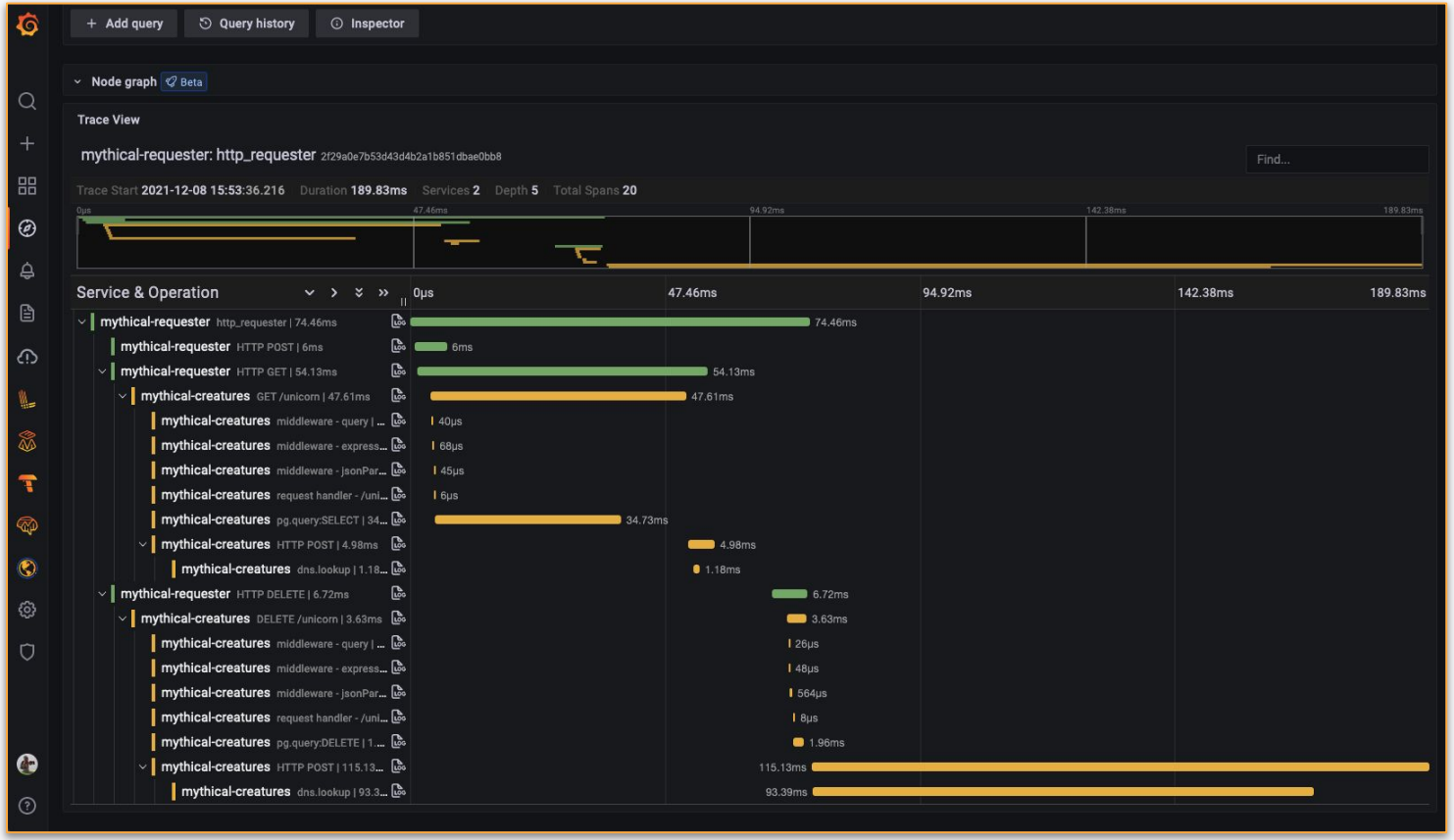
Agenda

- What is Distributed Tracing
- Distributed Tracing with Tempo
- Grafana Tempo & TraceQL
- Storage layouts & Parquet
- TraceQL Demo
- Q&A

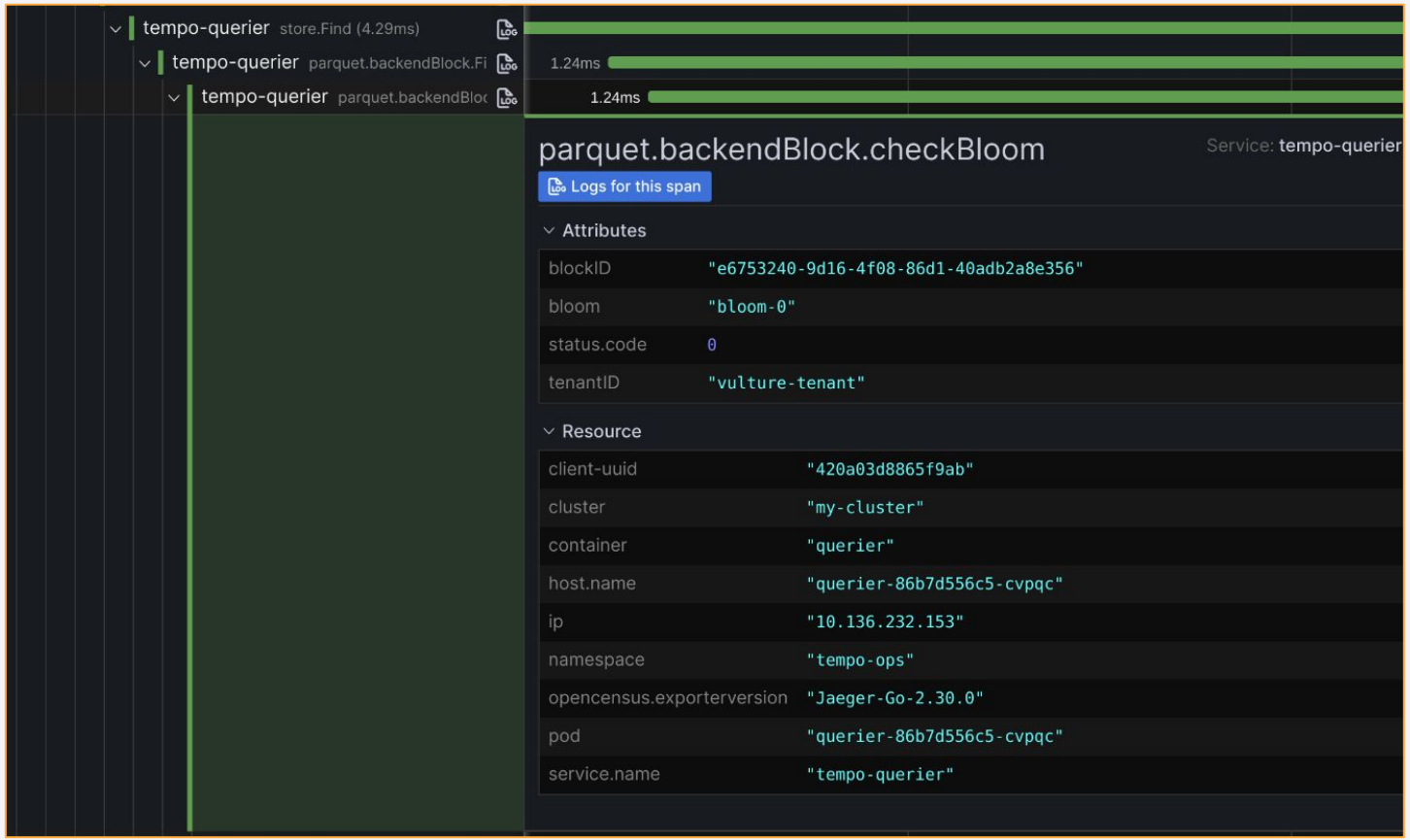
Poll: Are you familiar with distributed tracing?

What’s distributed tracing?

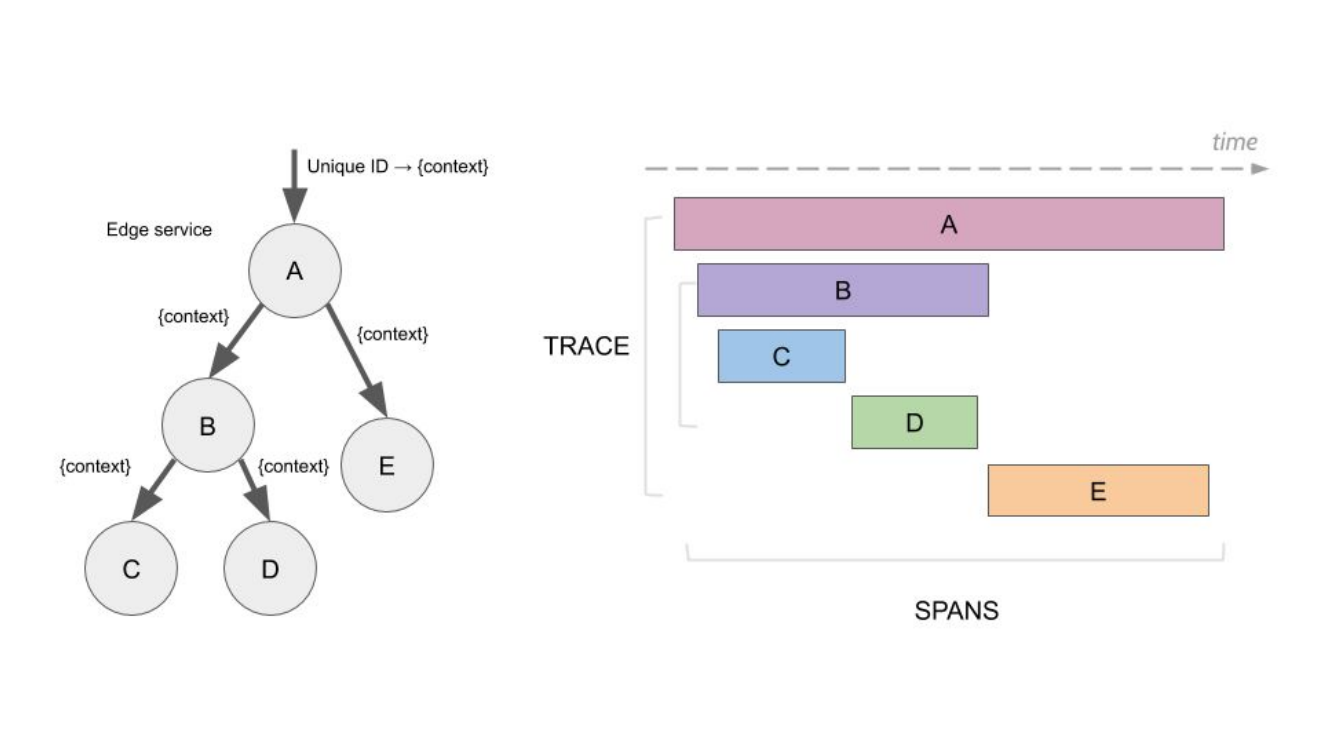
Trace


Add query
Query history
Inspector
Node graph
Beta
Trace View
mythical-requester: http_requester 2f29a0e7b53d43d4b2a1b851dbae0bb8
Find...
Trace Start 2021-12-08 15:53:36.216 Duration 189.83ms Services 2 Depth 5 Total Spans 20
0µs
47.46ms
94.92ms
142.38ms
189.83ms
Service & Operation
mythical-requester http_requester | 74.46ms
mythical-requester HTTP POST | 6ms
mythical-requester HTTP GET | 54.13ms
mythical-creatures GET /unicorn | 47.61ms
mythical-creatures middleware - query | ...
mythical-creatures middleware - express...
mythical-creatures middleware - jsonPar...
mythical-creatures request handler - /uni...
mythical-creatures pg.query:SELECT | 34...
mythical-creatures HTTP POST | 4.98ms
mythical-creatures dns.lookup | 1.18...
mythical-requester HTTP DELETE | 6.72ms
mythical-creatures DELETE /unicorn | 3.63ms
mythical-creatures middleware - query | ...
mythical-creatures middleware - express...
mythical-creatures middleware - jsonPar...
mythical-creatures request handler - /uni...
mythical-creatures pg.query:DELETE | 1 ...
mythical-creatures HTTP POST | 115.13...
mythical-creatures dns.lookup | 93.3...
74.46ms
6ms
54.13ms
47.61ms
40µs
68µs
45µs
6µs
34.73ms
4.98ms
1.18ms
6.72ms
3.63ms
26µs
48µs
564µs
8µs
1.96ms
115.13ms
93.39ms

Span


tempo-querier store.Find (4.29ms)
tempo-querier parquet.backendBlock.Fi
tempo-querier parquet.backendBloc
1.24ms
1.24ms
parquet.backendBlock.checkBloom
Service: tempo-querier
Logs for this span
Attributes
blockID "e6753240-9d16-4f08-86d1-40adb2a8e356"
bloom "bloom-0"
status.code 0
tenantID "vulture-tenant"
Resource
client-uuid "420a03d8865f9ab"
cluster "my-cluster"
container "querier"
host.name "querier-86b7d556c5-cvpqc"
ip "10.136.232.153"
namespace "tempo-ops"
opencensus.exporterversion "Jaeger-Go-2.30.0"
pod "querier-86b7d556c5-cvpqc"
service.name "tempo-querier"

Context Propagation



Why do we need tracing?

Modern services are complex

Logs and metrics are half of the story

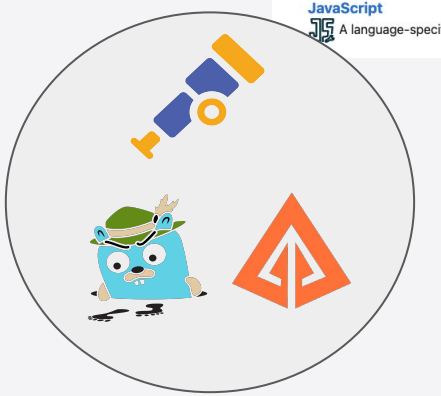
More detailed view into data transactions

So - how do I get started?

Step 1 - Instrument your application

- Many ways to instrument
  - Open Telemetry
  - Jaeger
  - Zipkin

- Many Languages have auto Instrumentation in client SDK



Instrumentation

OpenTelemetry code instrumentation is supported for the languages listed below. Depending on the language, topics covered will include some or all of the following:

- Automatic instrumentation
- Manual instrumentation
- Exporting data

C++
A language-specific implementation of OpenTelemetry in C++.

.NET
A language-specific implementation of OpenTelemetry in .NET.

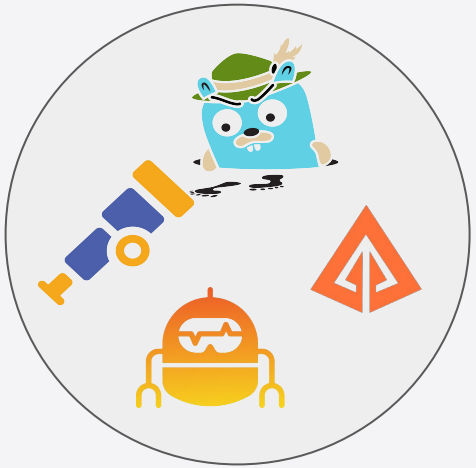
Erlang/Elixir
A language-specific implementation of OpenTelemetry in Erlang/Elixir.

Go
A language-specific implementation of OpenTelemetry in Go.

Java
A language-specific implementation of OpenTelemetry in Java.

JavaScript
A language-specific implementation of OpenTelemetry in JavaScript (for Node.JS & the browser).

Step 2 - Set up your tracing pipeline



A pipeline (also collector) provides:

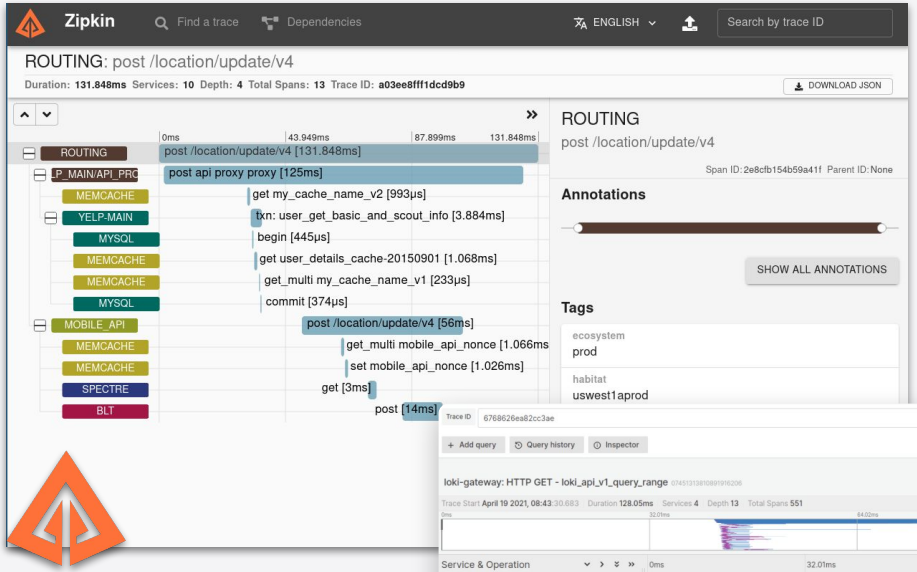
- Buffering of spans for efficient data transmission
- Ability to sample and control the flow of traces
- Enriching & processing of trace data

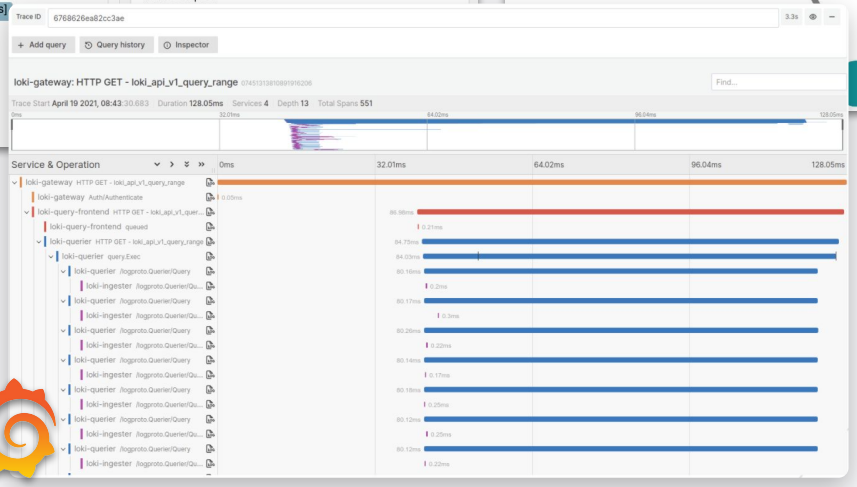
Step 3 - Set up your tracing backend

Somewhere to store all the traces that you generate…

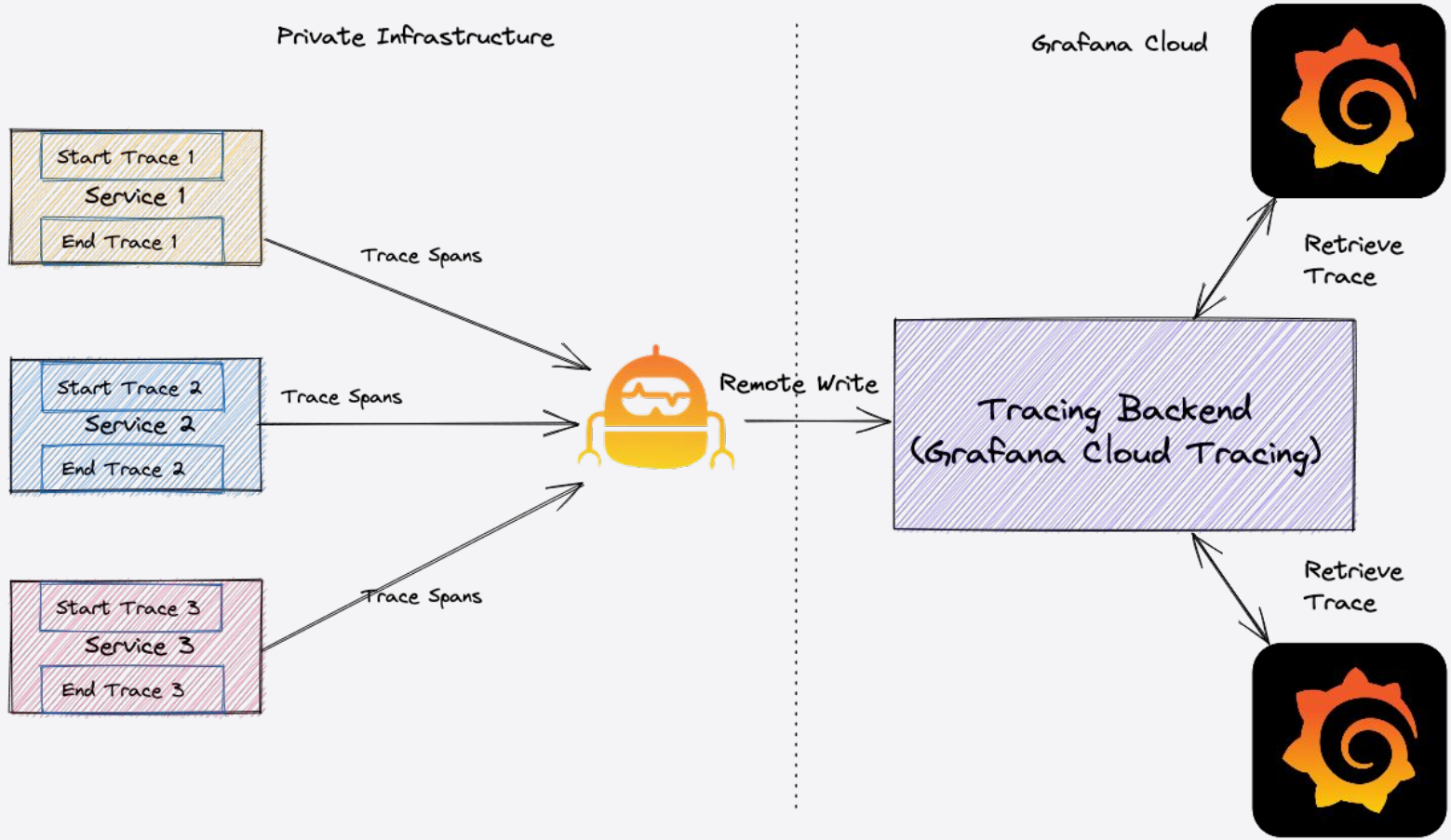
…and a way to query those traces back!

Step 4 - Visualize





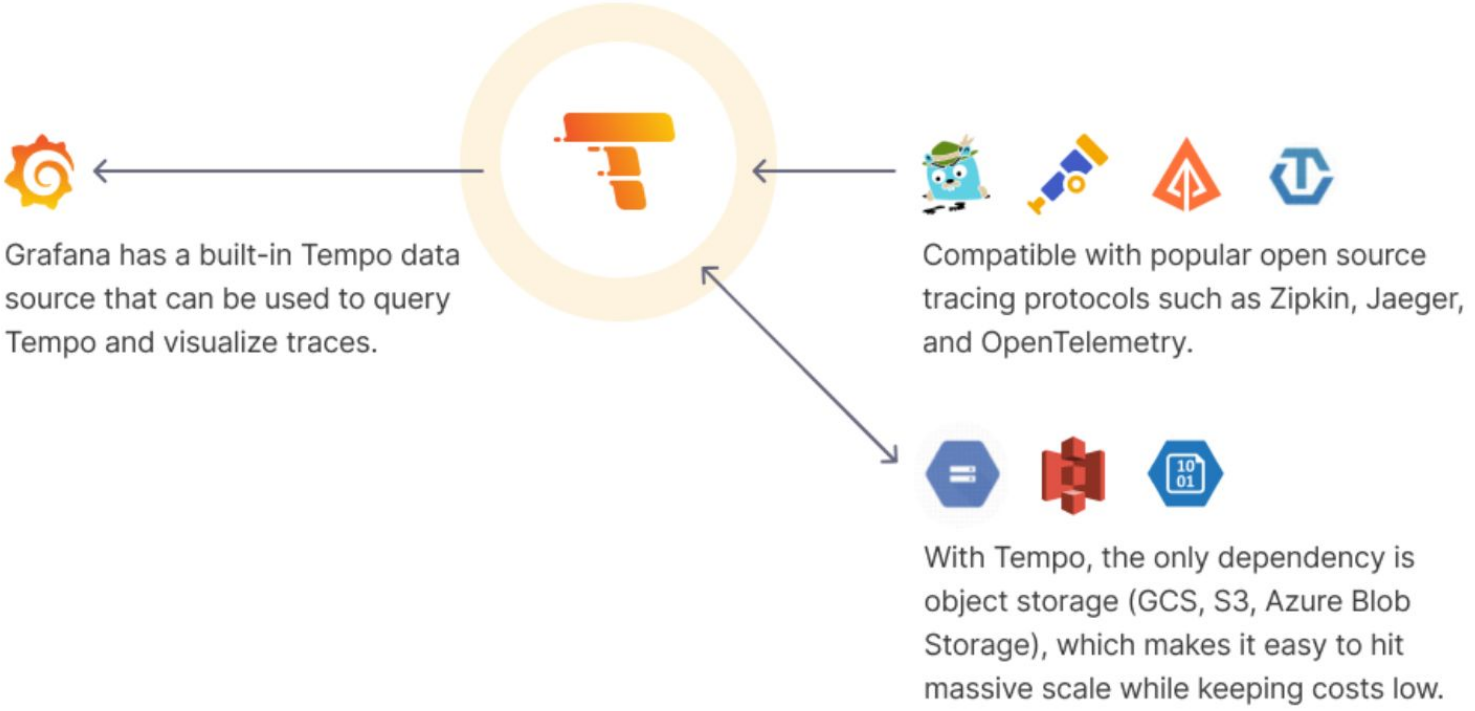
How to do Distributed Tracing (with Tempo and Grafana Agent!)



Grafana Tempo

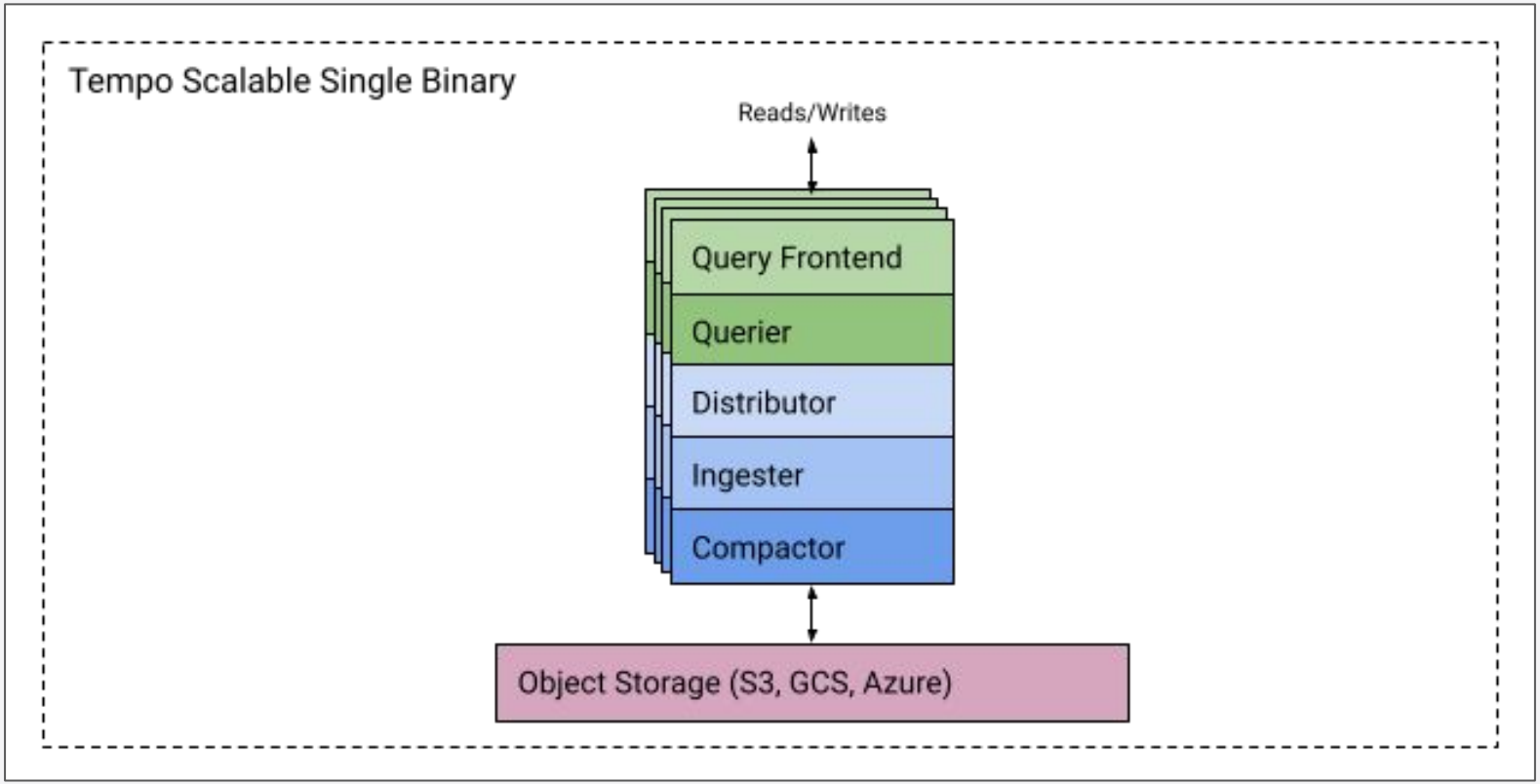



How does Grafana Tempo work?

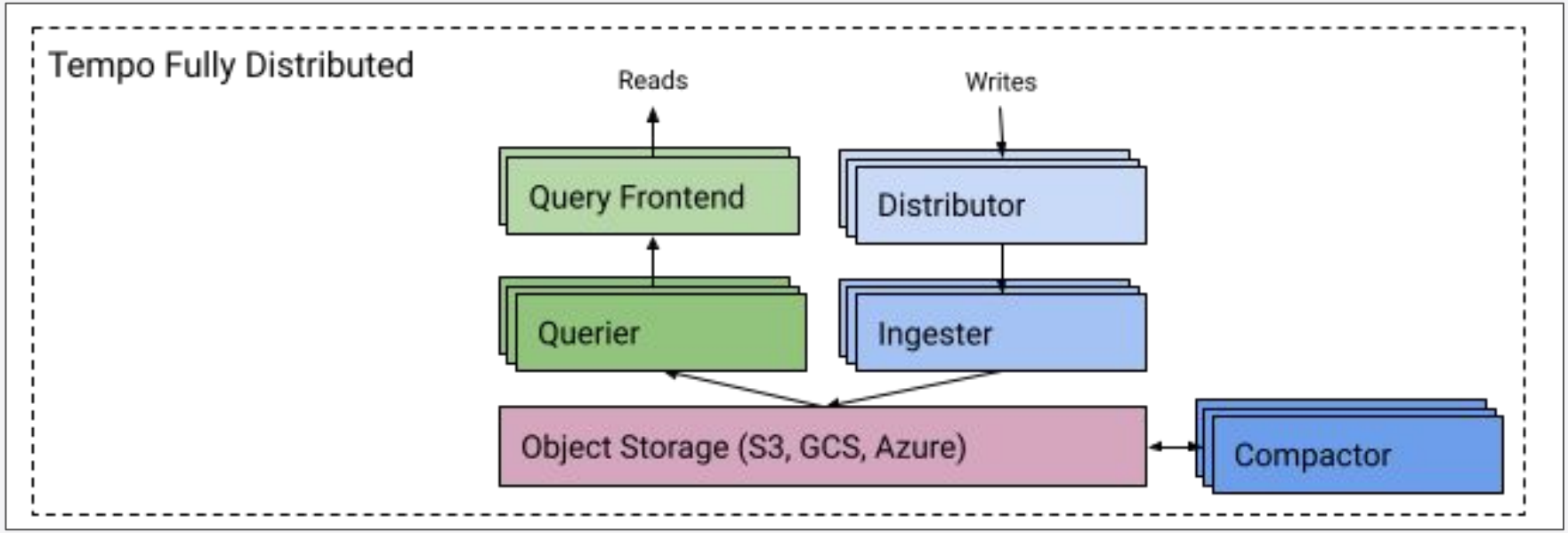



Running Tempo

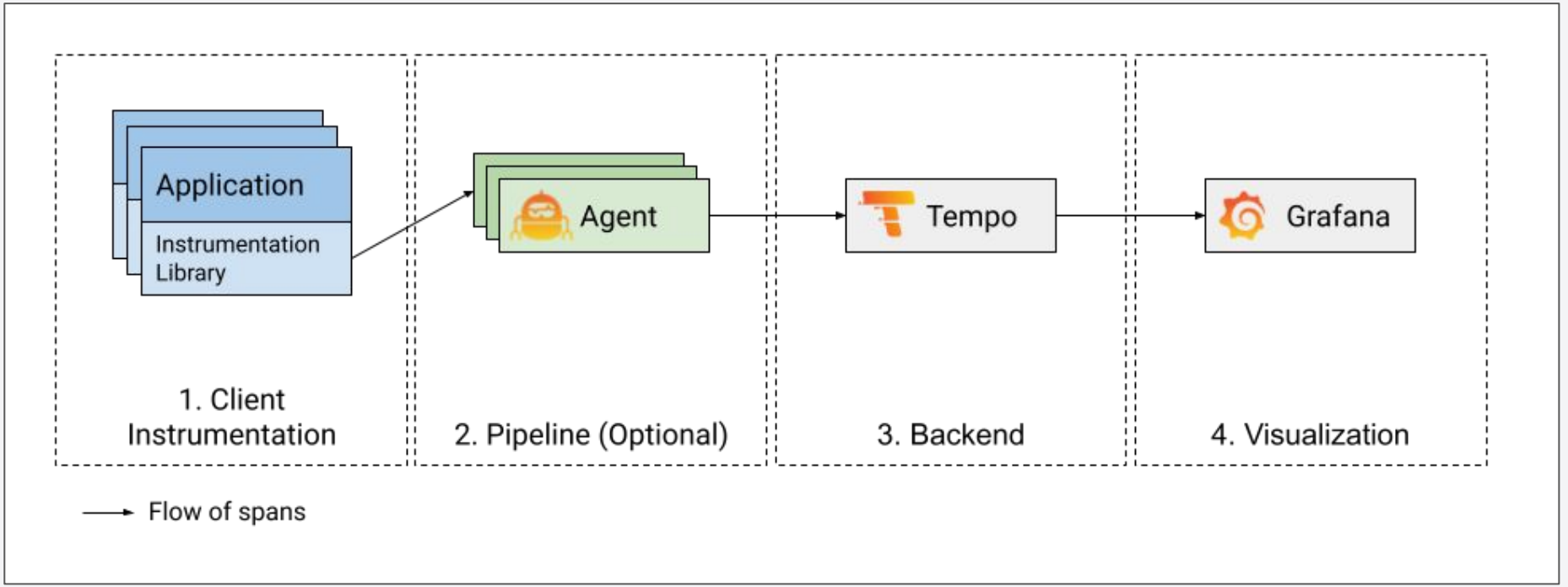
Run it as a single process



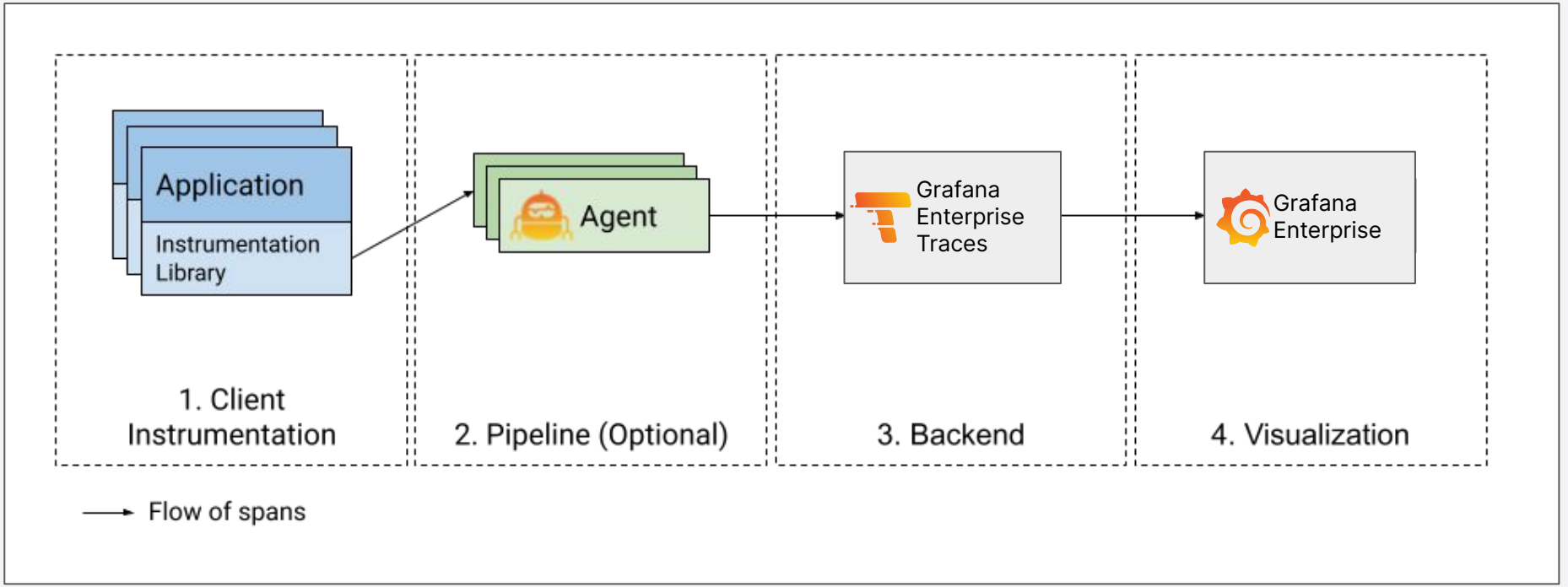
Run it as microservices



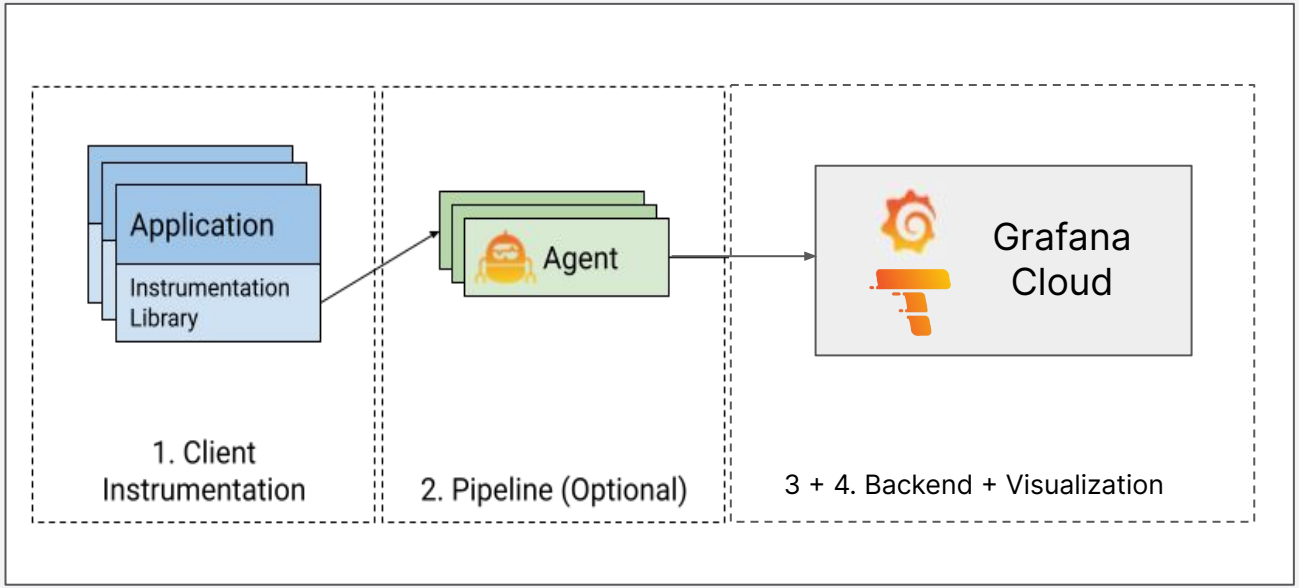
Run it yourself with open source



Run it yourself with Enterprise



Grafana Cloud Traces, fully managed



You focus on instrumentation, we handle the rest.

Up to 50GB/month free, with 30 day retention

TraceQL

```
{ .service.name = "foo" &&
.http.status_code = 500 }
```

I don't want to learn your garbage query language

2018-08-30

This is a bit of a rant but I really don't like software that invents its own query language. There's a trillion different ORMs out there. Another trillion databases with their own query language. Another trillion SaaS products where the only way to query is to learn some random query DSL they made up.

- Splunk has SPL
- Mixpanel has JQL
- Rollbar has RQL
- New Relic has NRQL
- Adwords has AWQL

Source: I don't want to learn your garbage query language

TraceQL - Why?

Traces are the flow of events throughout your components, in a tree structure.

We believe you should be able ask more questions from this data, instead of just looking at single trace

TraceQL - Why?

Example:

- *How many connections to the database are your slowest requests doing?*

- *Show me requests that took more than 10 seconds **AND** did more than 3 SQL queries?*

Type awareness

```
{ .http.status_code >= 200 &&
  .http.status_code < 300 }
```

```
{ duration > 1s && duration < 2s }
```

Regex

```
{ .http.url =~ "/api/v1/.*" }
```

```
{ .environment =~ "prod-.*" }
```

Aggregates

```
{ .db.name = "postgres" } | count() > 1
```

```
{ .db.statement =~ "INSERT.*" } |
avg(duration) > 1s
```

```
{
 .http.status_code >= 500 &&
 .http.url =~ "/api/.*" &&
  ( .http.method = "POST" || .http.method = "PUT" )
} | avg(duration) > 1s
```

Show me all the spans with 5xx or higher, where path contains /api/ and request is POST or PUT and avg duration of span is more then 1 second

TraceQL - Pipelining

The first version of TraceQL includes basic pipelining that can be used to filter span sets using the following **aggregations**:

- **Average**: {name=”service”} | **avg**(duration) > 1 ms
  - Selects all the traces with spans with name “service” that have an average duration over 1 ms.
- **Count**: {.service.name=”db”} | **count**() > 10
  - Selects all the traces that contain more than 10 spans with service.name “db”.

Advanced TraceQL

Aggregates

```
{ .cluster = "prod" } | by(.namespace)
```

```
{ .cluster = "dev" } | max(end) - min(start) > 10s
```

Pipelining

```
{ .cluster = "prod" } | by(.namespace) | count() > 5
```

```
{ .cluster = "dev" } | avg(duration) > { .cluster = "prod" } | avg(duration)
```

Advanced TraceQL

Structural operators

```
{ .namespace = "test" } >> { .namespace = "ops" }
```

```
{ .namespace = "test" } ~ { .namespace = "ops" }
```

TraceQL metrics (experimental)

```
{ .cluster = "prod" } | rate(5m)
```

```
{ .cluster = "prod" } | select(span.http.status_code, span.http.url)
```

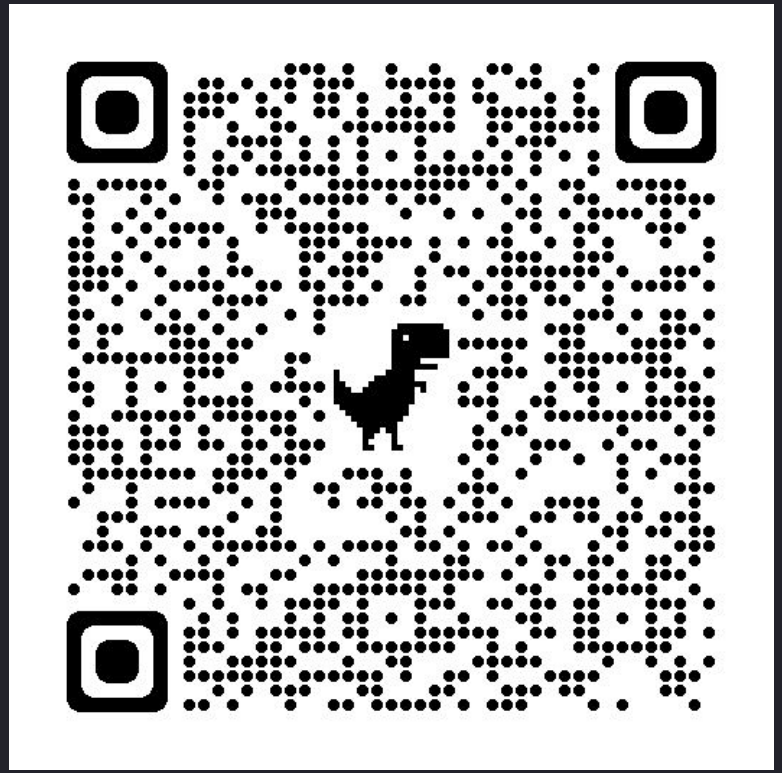
TraceQL - Grafana Support

Connects with the Trace view in order to expand any of the traces returned.

Grafana Labs

Want to learn more TraceQL??

Scan QR code to watch an hour long session on TraceQL :)



How is this possible with object storage?

i.e. Why and how parquet helps??

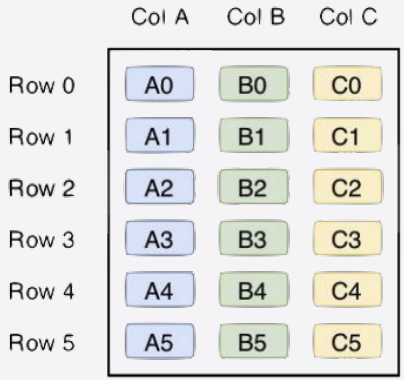
Requirements for TraceQL

- Data with very high cardinality
- Large operations involving subsets of columns
- Operations involving rows only when retrieving traces
- Only object storage

+ Object Storage

Physical storage layout models

Logical

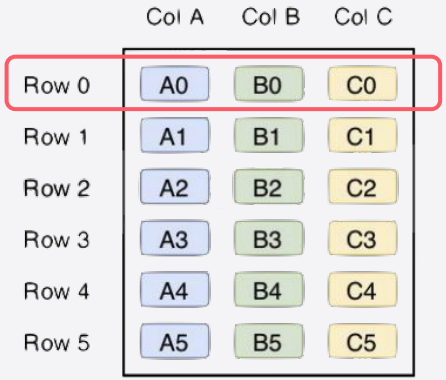


Physical

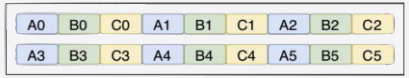
https://www.youtube.com/watch?v=1j8SdS7s_NY

Physical storage layout models

Logical



Physical

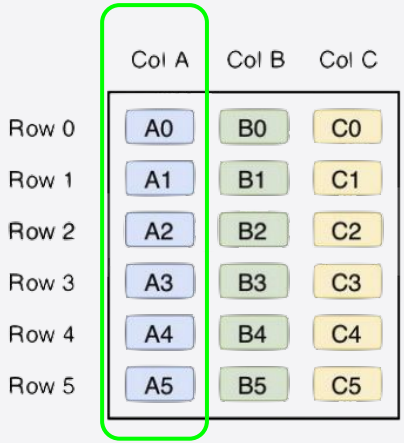


Row-wise

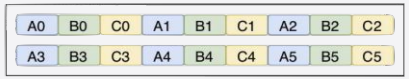
https://www.youtube.com/watch?v=1j8SdS7s_NY

Physical storage layout models

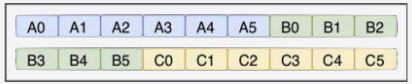
Logical



Physical



Row-wise

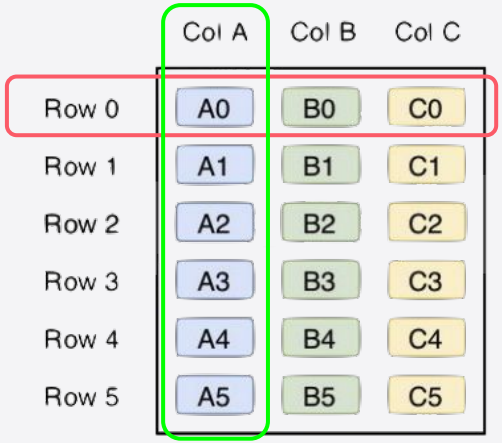


Columnar

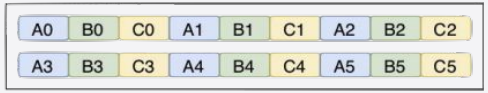
https://www.youtube.com/watch?v=1j8SdS7s_NY

Physical storage layout models

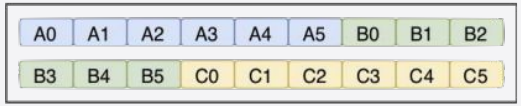
Logical



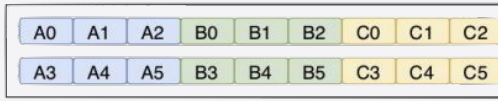
Physical



Row-wise



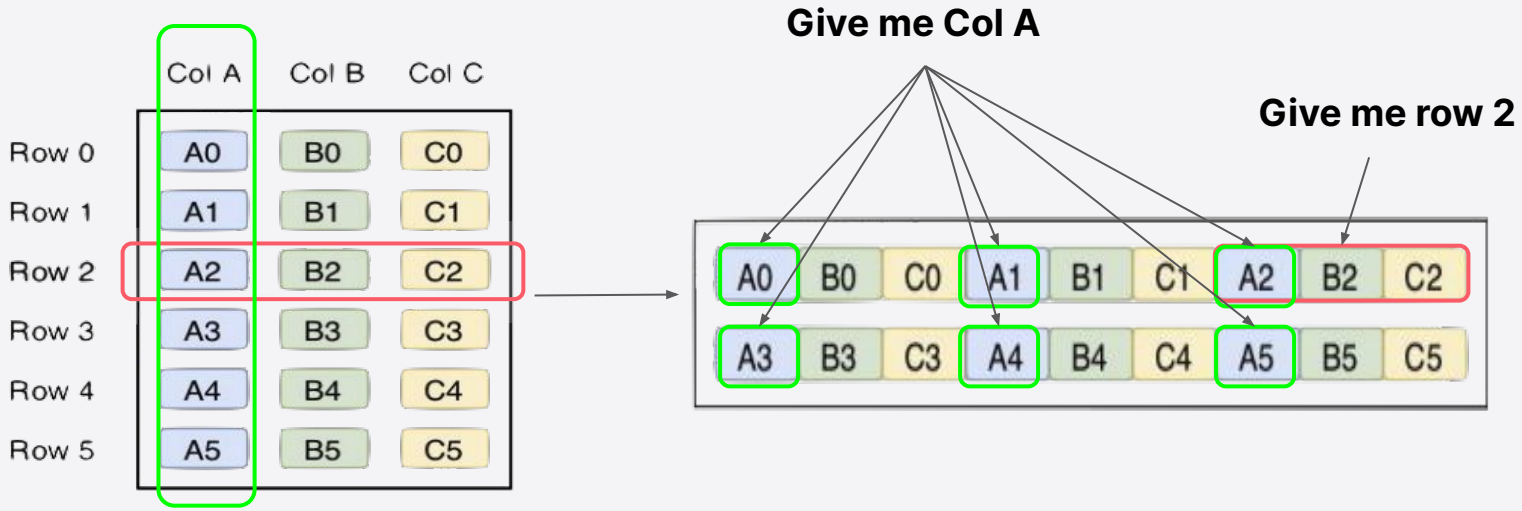
Columnar



Hybrid

https://www.youtube.com/watch?v=1j8SdS7s_NY

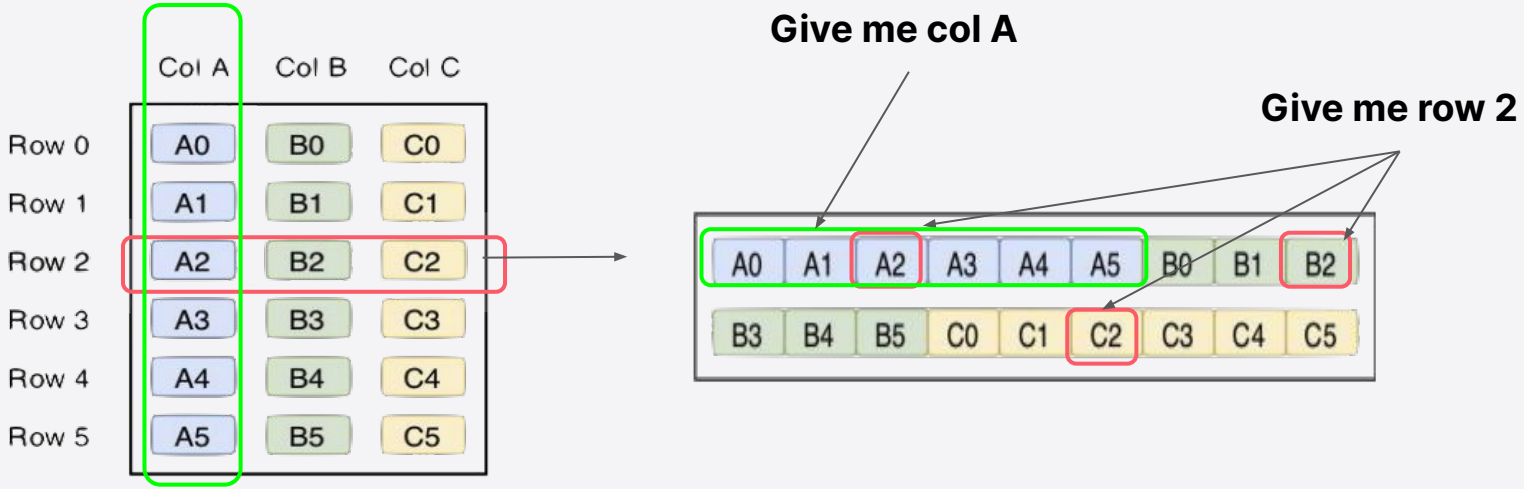
Row-wise



- Horizontal partitioning
- Great for lots of small operations involving ***whole*** *rows*

https://www.youtube.com/watch?v=1j8SdS7s_NY

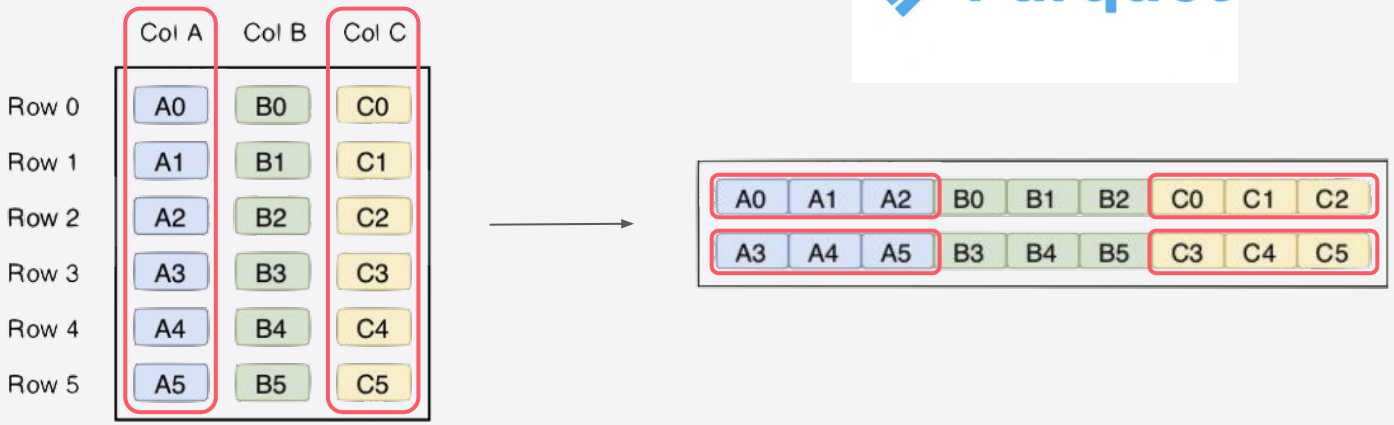
Columnar



- Vertical partitioning
- Great for few large operations involving ***subset*** *of all columns*

https://www.youtube.com/watch?v=1j8SdS7s_NY

Hybrid

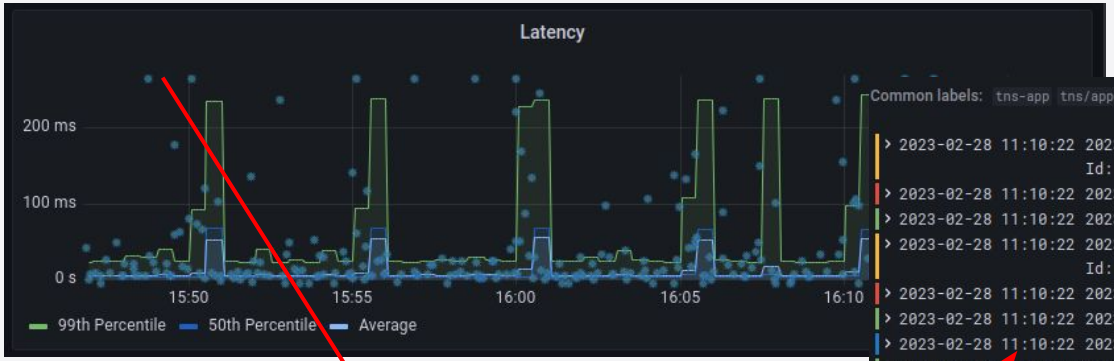


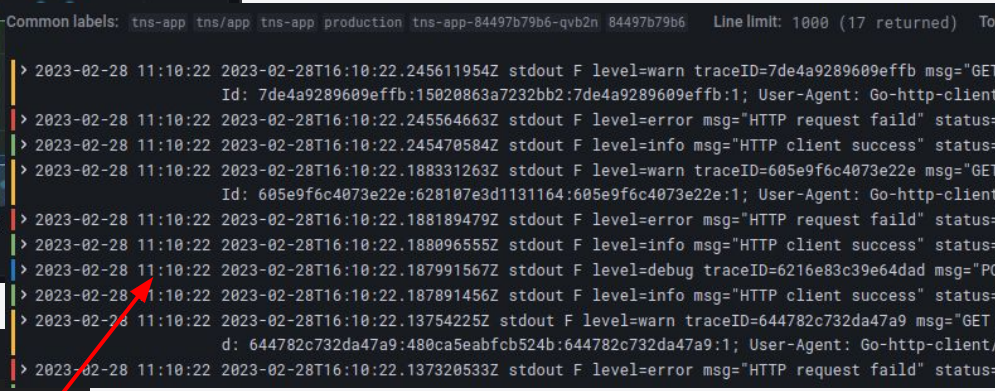
- Data with very high cardinality
- Large operations involving subsets of columns
- Operations involving rows only when retrieving traces

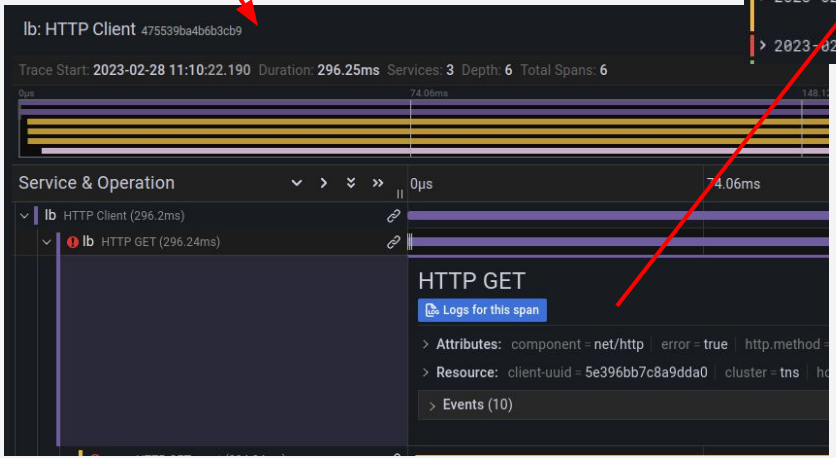
https://www.youtube.com/watch?v=1j8SdS7s_NY

How to get more from Tracing?

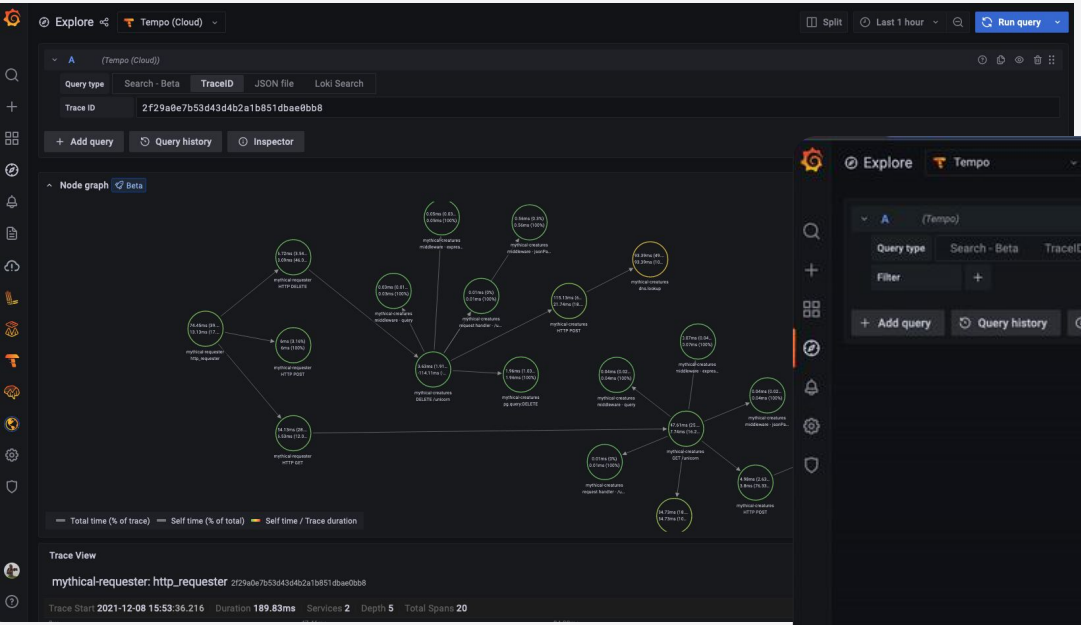
Correlations - Triage

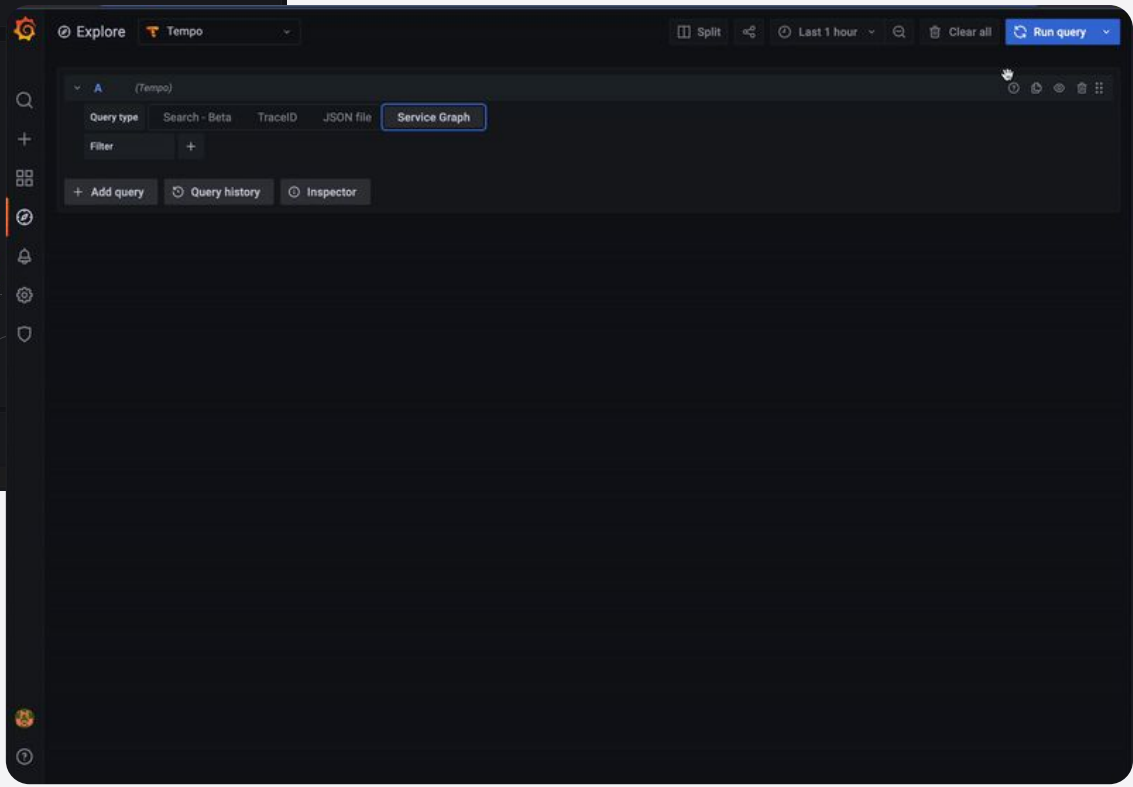






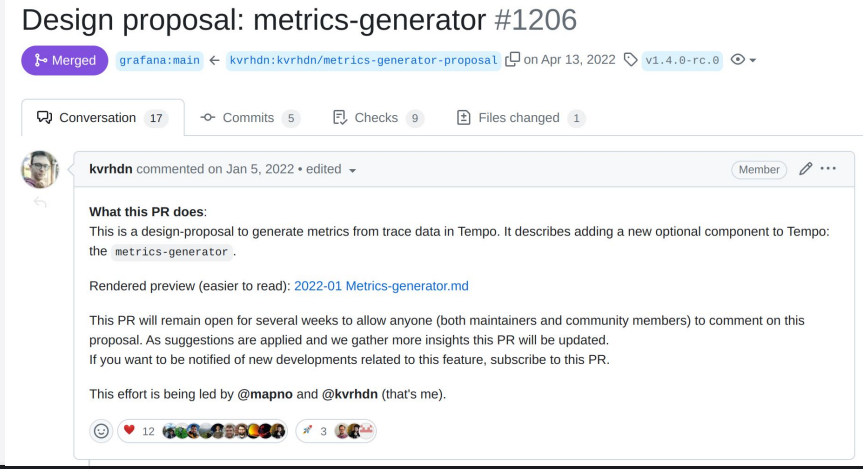
node (or service) graphs

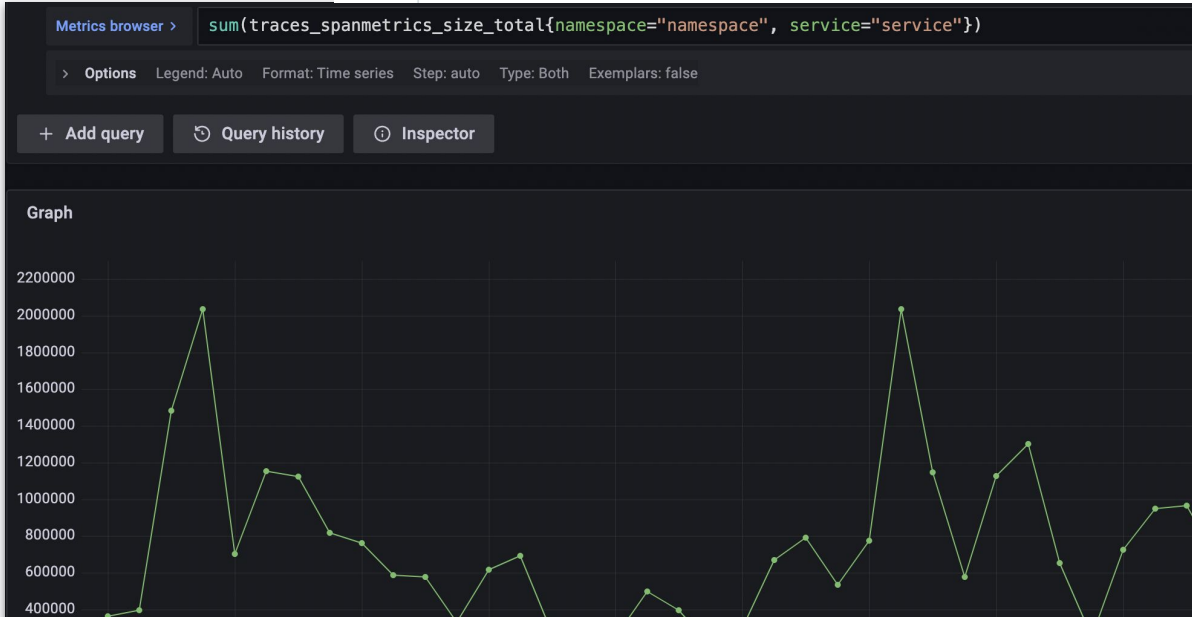




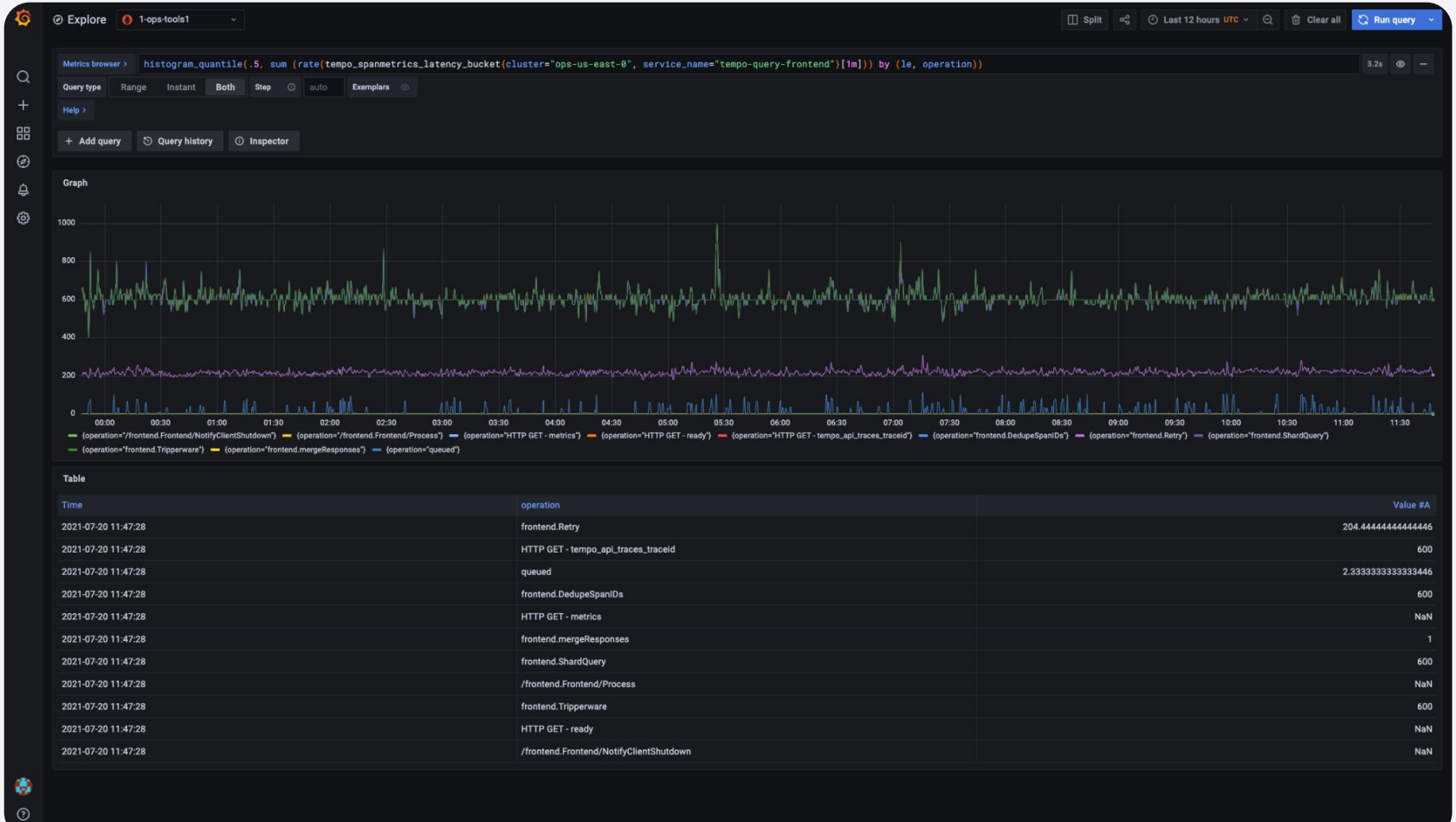
Metrics generator

Tempo’s metrics generator remote writes span metrics to a Prometheus compatible backend.





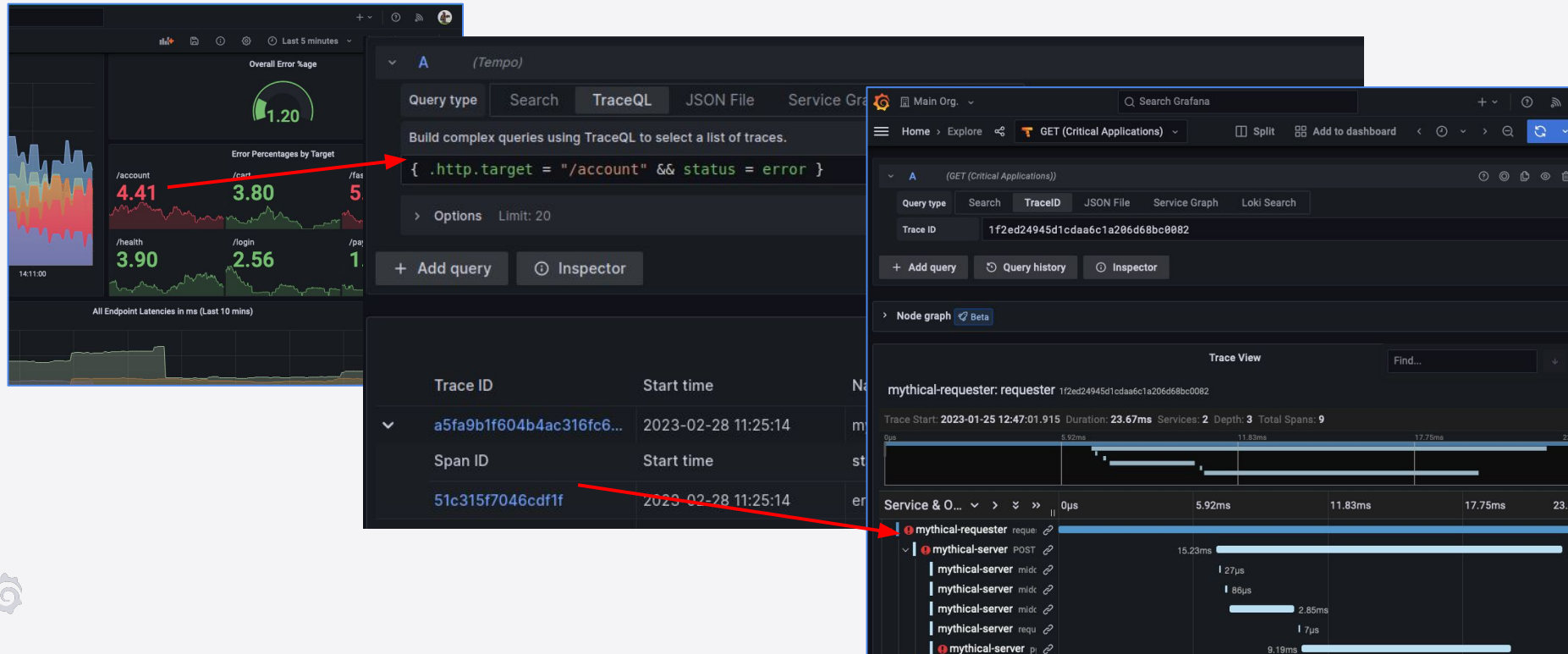
span metrics



Time	operation	Value #A
2021-07-20 11:47:28	frontend.Retry	204.44444444444446
2021-07-20 11:47:28	HTTP GET - tempo_api_traces_traceid	600
2021-07-20 11:47:28	queued	2.3333333333333446
2021-07-20 11:47:28	frontend.DedupeSpanIDs	600
2021-07-20 11:47:28	HTTP GET - metrics	NaN
2021-07-20 11:47:28	frontend.mergeResponses	1
2021-07-20 11:47:28	frontend.ShardQuery	600
2021-07-20 11:47:28	/frontend.Frontend/Process	NaN
2021-07-20 11:47:28	frontend.Tripperware	600
2021-07-20 11:47:28	HTTP GET - ready	NaN
2021-07-20 11:47:28	/frontend.Frontend/NotifyClientShutdown	NaN

Metrics Generation - Application Insights

Use data links to move easily from dashboard panels to trace exploration



Demo!

https://github.com/grafana/intro-to-mlt

Q&A

Thank you

Slides: suraj.dev/talks

Twitter: @electron0zero