OpenTelemetry - The Universal Observability Umbrella

About me

Abhinav Kapoor

- **Principal Engineer, Nagarro**
- Work as a Technical Architect for .NET Solutions.
- 15+ years of technical experience in architecting & developing solutions for Banking, Supply chain & Embedded domains.
- I enjoy writing about software architecture, communication patterns, modernization & more on Medium.com https://medium.com/@kapoorabhinav.

I'm available on Linked at https://www.linkedin.com/in/abhinav-kapoor-0394456/

Accreditations

What we'll cover

1. Observability Concepts
    1. What is observability & Why do we need It ?
    2. How does it fit in a cloud native landscape?
    3. Application Instrumentation – The pillars for getting insights
        1. Logs
        2. Metrics
        3. Traces
2. Future of Observability - OpenTelemetry
    1. What is it & why is it needed?
    2. Its components.
        1. Traces
        2. Metrics
        3. Logs
3. Demo

What is observability ?

Sir Lewis Carl Davidson Hamilton
- 7-time F1 champion

Steering Wheel with controls and dashboard
- Image credit Mercedes X

What is observability ?

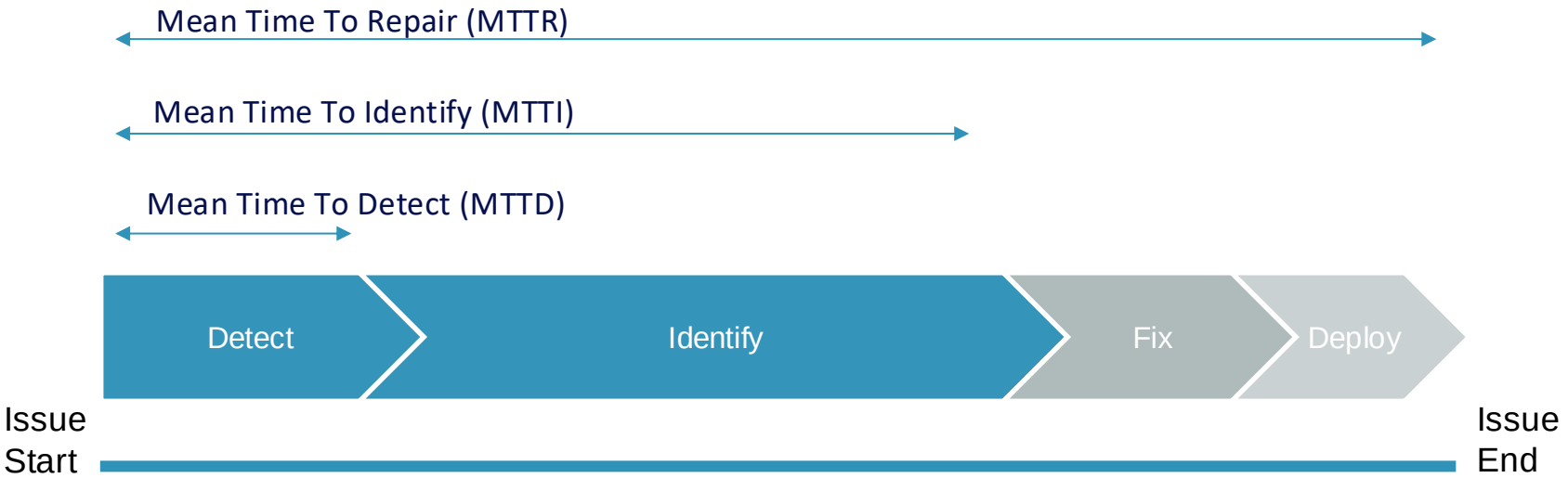
- Moving from Formula 1 to Modern Software applications
  - Complex systems – Lot of moving parts
  - High stakes
  - Cutting edge engineering
  - Fast Reaction
  - Several teams with different specializations at work
- Observability is the ability to observe what is going on inside the system.
- It can be considered as an NFR which helps measures other NFRs

Why do we need Observability?

- We want to avoid these reactions

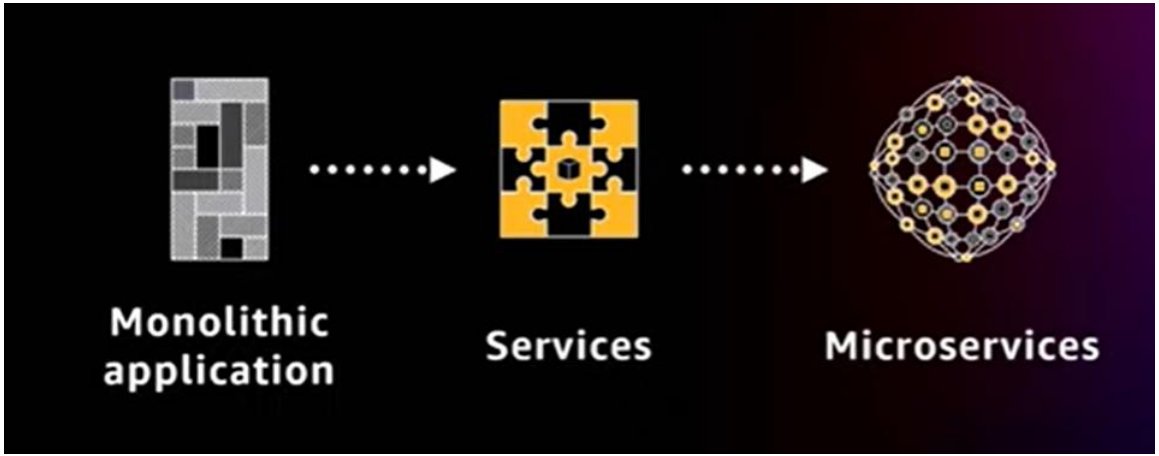
Why do we need Observability?

- Issue Timeline in Operation Support



Why is Observability more critical now?

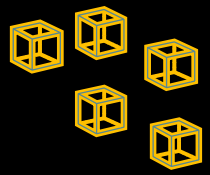
- Evolution to smaller but more services - high cohesion and low coupling.



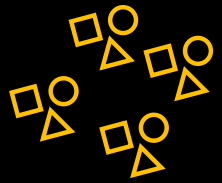
“Complexities arises when the dependencies among the elements become important”
- We have reduced code complexity but introduced much more moving parts

Why is Observability more critical now?

Technology transformation trends



Microservices

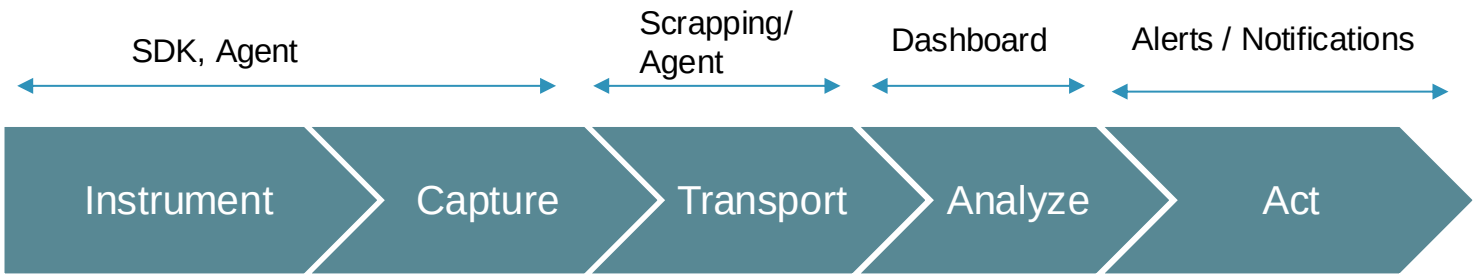


Polyglot Applications

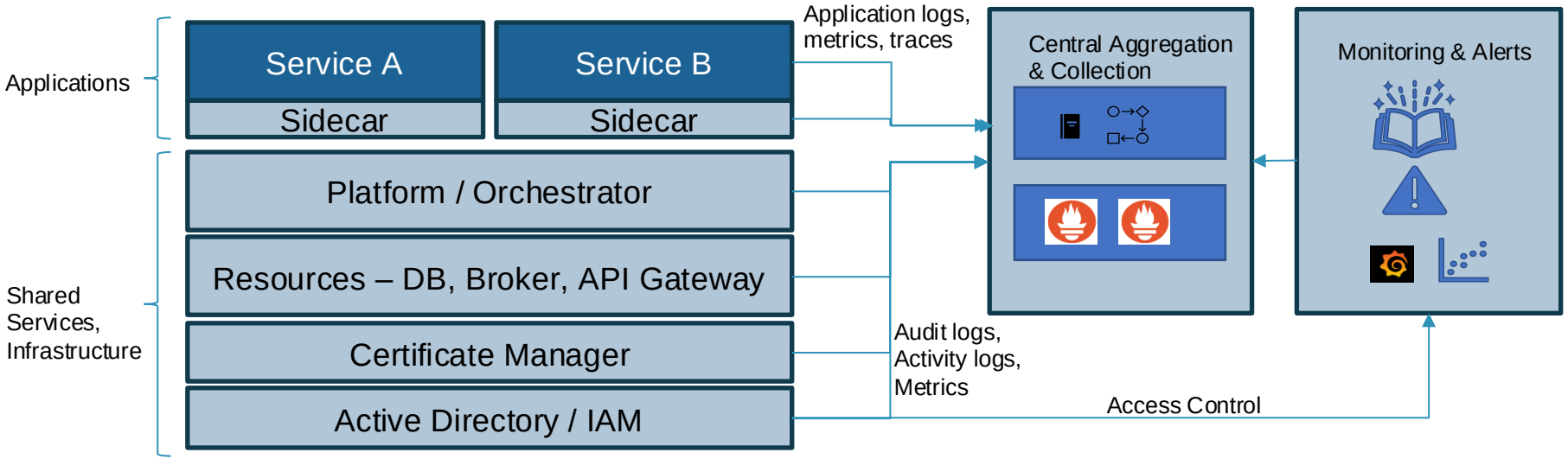
Ephemeral & cloud native environments

How does it fit in a cloud native landscape?

How does it fit in a cloud native landscape?



How does it fit in a cloud native landscape?



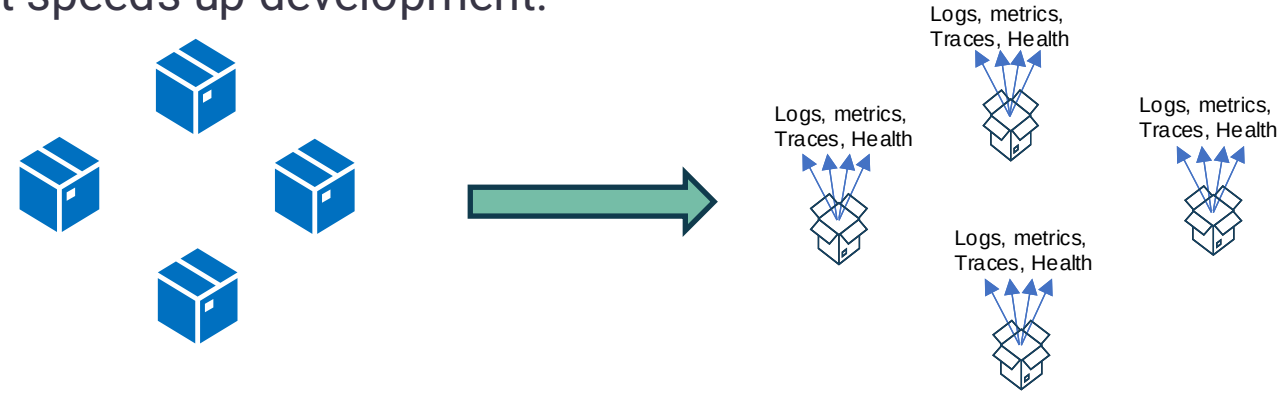
Deep Dive - Application Instrumentation

Application Instrumentation

- Logs – What happened & who did it ?
  - Important events. State changes, who did.
- Traces – How did it happen ?
  - Flows within the program or outside the program.
- Metrics – How much happened ?
  - Numbers which can be aggregated to have formulas – SLOs, KPI & alerts
- Health – How is the system doing ? Current state of health

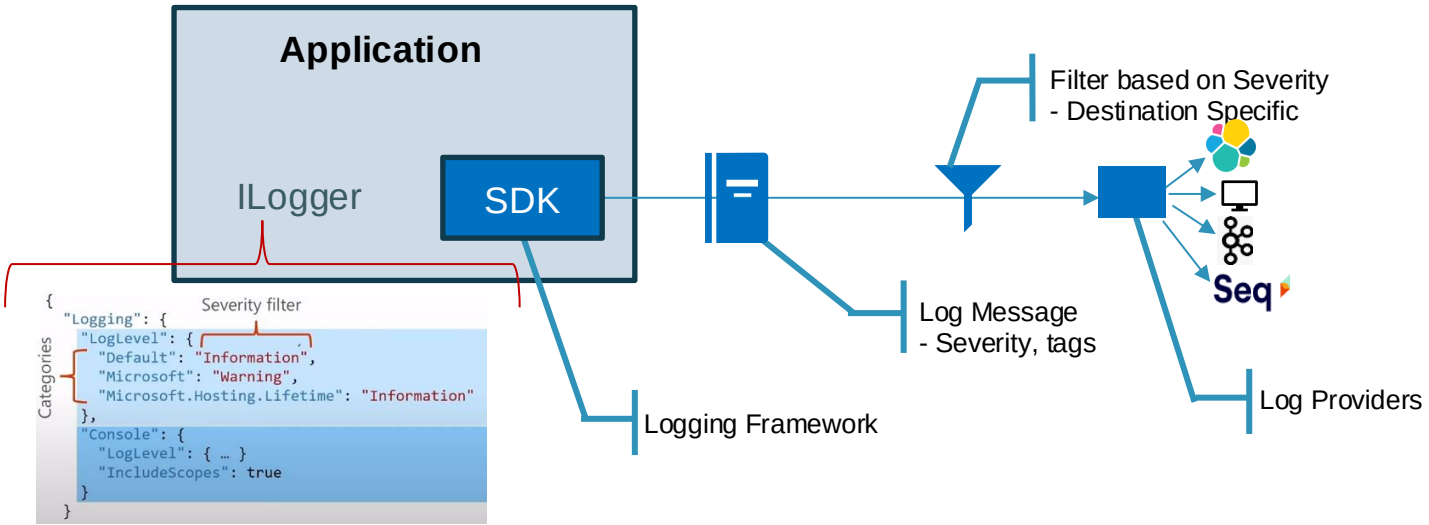
Big picture

- We don’t just want to have connected black boxes.
- While infrastructure & sidecars can give a good overview, there are always blind spots.
- Do an **intentional** instrumentation.
- It speeds up development.



Logs – What Happened ?

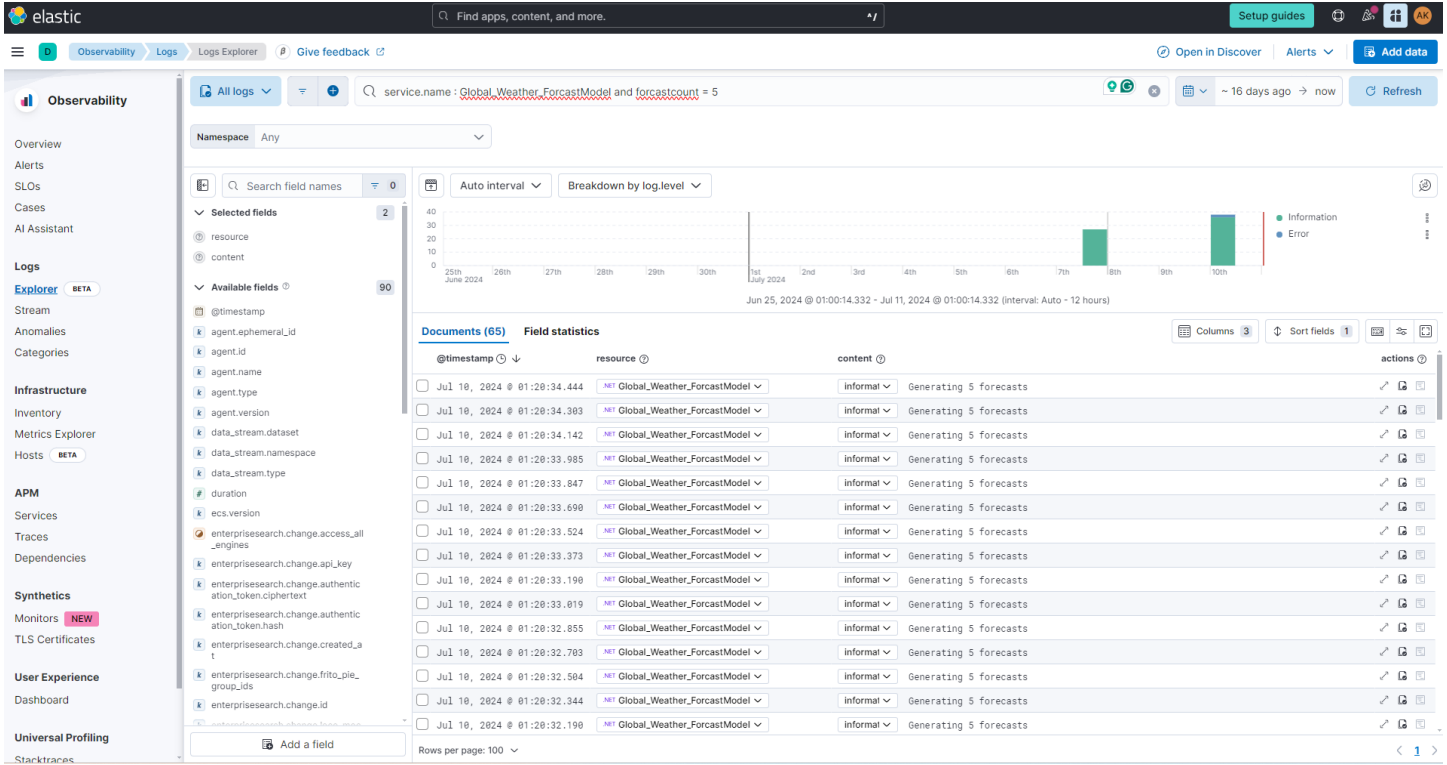
Logs – Components of Logging



Types of Logs

- Based on Information they contain – Governs where they are stored, how long they are stored, who can access them
    - Technical – Startup configurations, system event that the application receives
    - Business - Business and User Information
- Based on the schema
    - Unstructured – Simple statements with severity.
    - Structured / Schematic logs - Logs with Metadata to provide context. Pod, Service, IP address & more.

Duality of Logs - Readable messages & Quarriable metrics



Traces - How did it happen ?

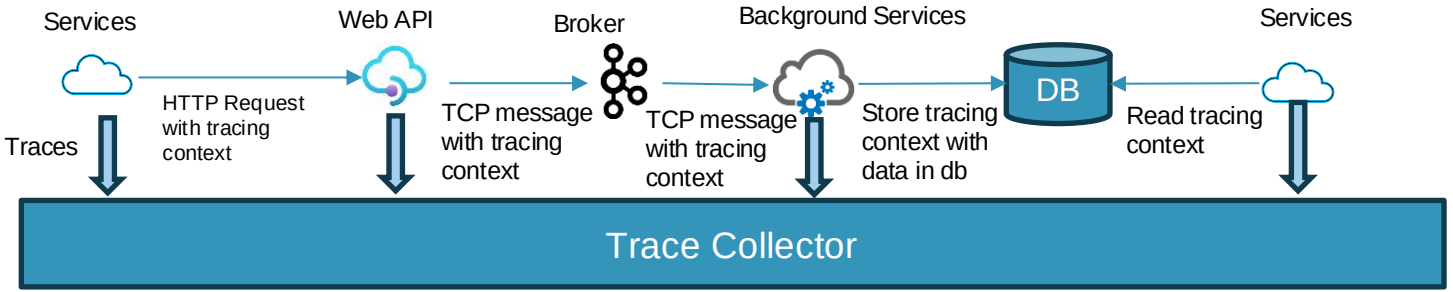
Traces – Types of traces

- Tracing – This is similar to logging. But noisier and collects more information from deeper parts of the application.

- Distributed Trancing – It's the method to observe requests as they propagate through different services.

    - Its structured log which comes from different processes, nodes and can be stitched together to give an end to end view.

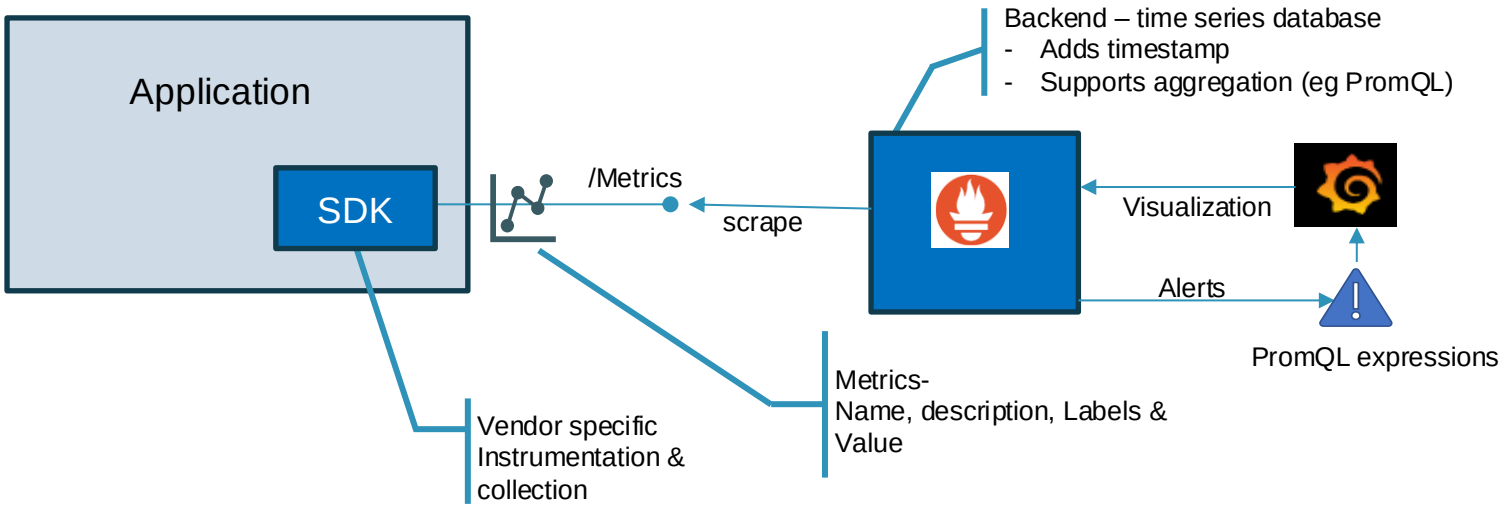
Distributed Tracing



- Correlation IDs are passed in the remote calls/messages.
- The receiver regenerates context proceeds with its unit of work.
- Depending upon business transaction, traces can be of few seconds to hours or days.

Metrics–How much is happening?

Metrics – Components



Types of Prometheus Metrics - Counters

Track the occurrence of an event in application.

- The absolute value may not be helpful.
- The delta between two timestamps or the rate of change over time is helpful. Functions like Increase(), Rate().
- Consider the case when applications restart.
- **Examples - Orders processed counter, unsuccessful counters, business error, technical errors.**
- Sample output of meta-endpoint (/metrics)

```
# HELP http_requests_total Total number of http api requests
# TYPE http_requests_total
counter http_requests_total{api="add_Product", company="ITC"} 3433
counter http_requests_total{api="add_Product", company ="TATA"} 2000

PromQL
increase(http_requests_total{api="add_product", company="ITC"}[5m])
```

Types of Prometheus Metrics - Gauges

Snapshots of a metric at a single point in time. Not an event.

- **Example is message queue size, CPU utilization at a given time.**
- Useful functions - avg_over_time, max_over_time, min_over_time, and quantile_over_time

Types of Prometheus Metrics - Histograms

- Histograms divide the entire range of measurements into a set of intervals—named buckets—and count how many measurements fall into each bucket.
- These buckets are defined at compile time, they are upper inclusive (le).
- Histogram looks like the following

```
http_request_duration_seconds_sum{api="add_product" instance=" host1 "} 8953.332
http_request_duration_seconds_count{api="add_product" instance=" host1"} 27892
http_request_duration_seconds_bucket{api="add_product", instance=" host1", le="0.05"} 1672
http_request_duration_seconds_bucket{api="add_product", instance=" host1", le="0.1"} 8954
http_request_duration_seconds_bucket{api="add_product", instance=" host1", le="0.25"} 14251

sum by (le) (rate(http_request_duration_seconds_bucket[5m]))
```

Types of Prometheus Metrics - Summaries

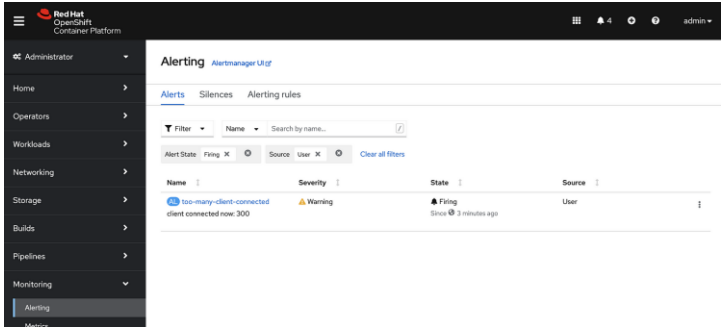
- Similar to Histogram but they add quantile label to bound the summary.
- OpenTelemetry has been marked it legacy.
- Summary looks like the following

```
http_request_duration_seconds_sum{api="add_product" instance="host1.domain.com"} 8953.332
http_request_duration_seconds_count{api="add_product" instance="host1.domain.com"} 27892
http_request_duration_seconds{api="add_product" instance=" host1 " quantile="0"}
http_request_duration_seconds{api="add_product" instance="host1" quantile="0.5"} 0.232227334
```

- Considerations due to quantile.

Metrics - What happens after instrumentation?

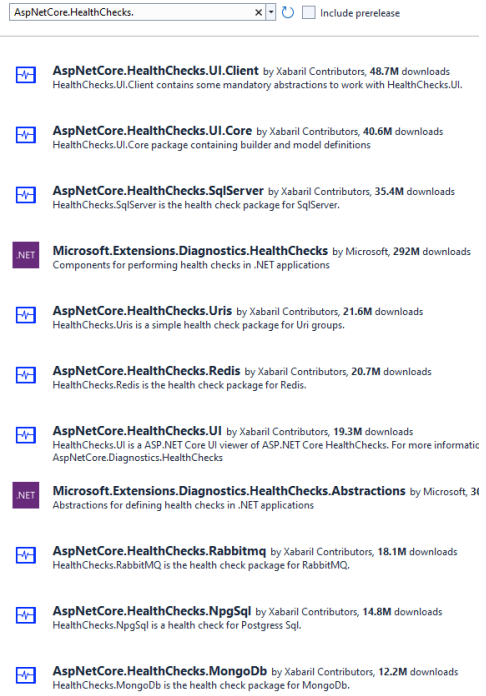
- The orchestrator can enrich the application instrumentation with infrastructure information such as Service Name, Pod, IP address.
- The backend saves it with timestamp
- Add Alerts & dashboards.



Health Check– How is the system doing ?

Health check

- Basic use is liveness probe.
- More useful when we add dependencies.
- Prebuilt libraries depending upon the language
- Considerations
    - Orchestrator may restart the service while the issue is in a dependency.
    - Checking all downstream services could be time consuming.
    - Meta-end points (/health) may not be exposed. Instead of writing to the response metrics may be used to indicate issue.

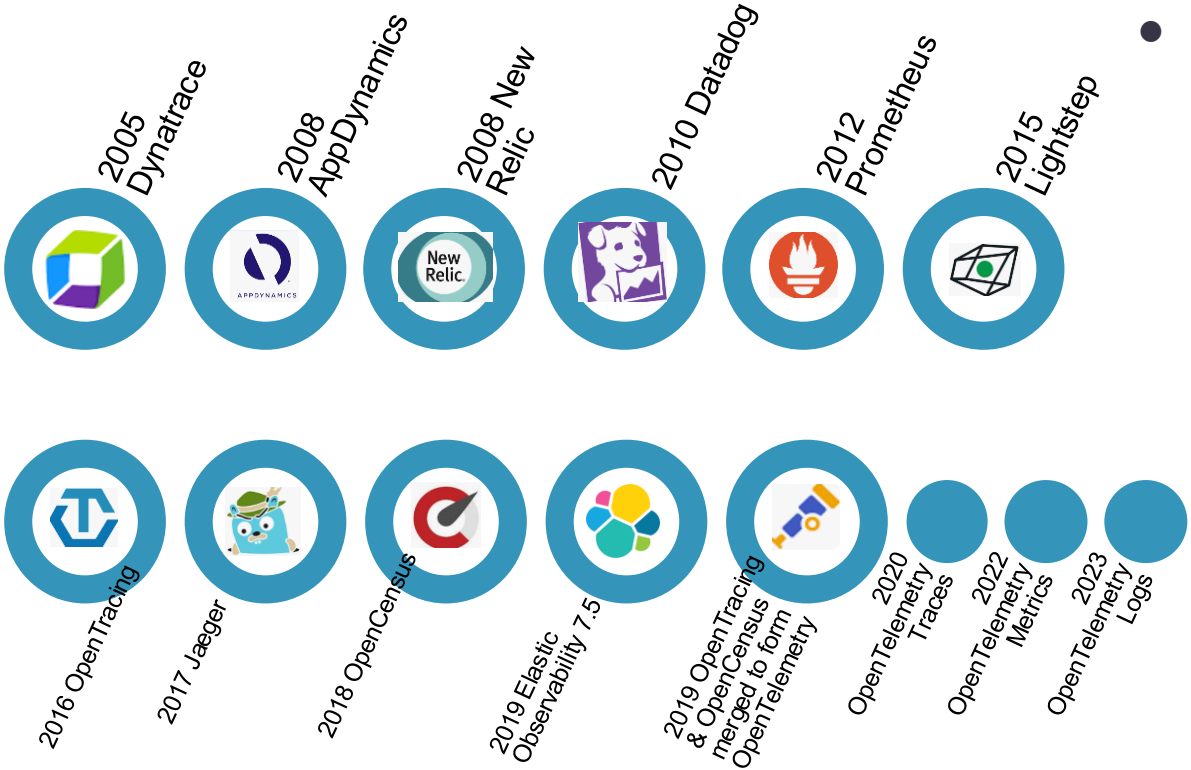


OpenTelemetry

What is OpenTelemetry ?

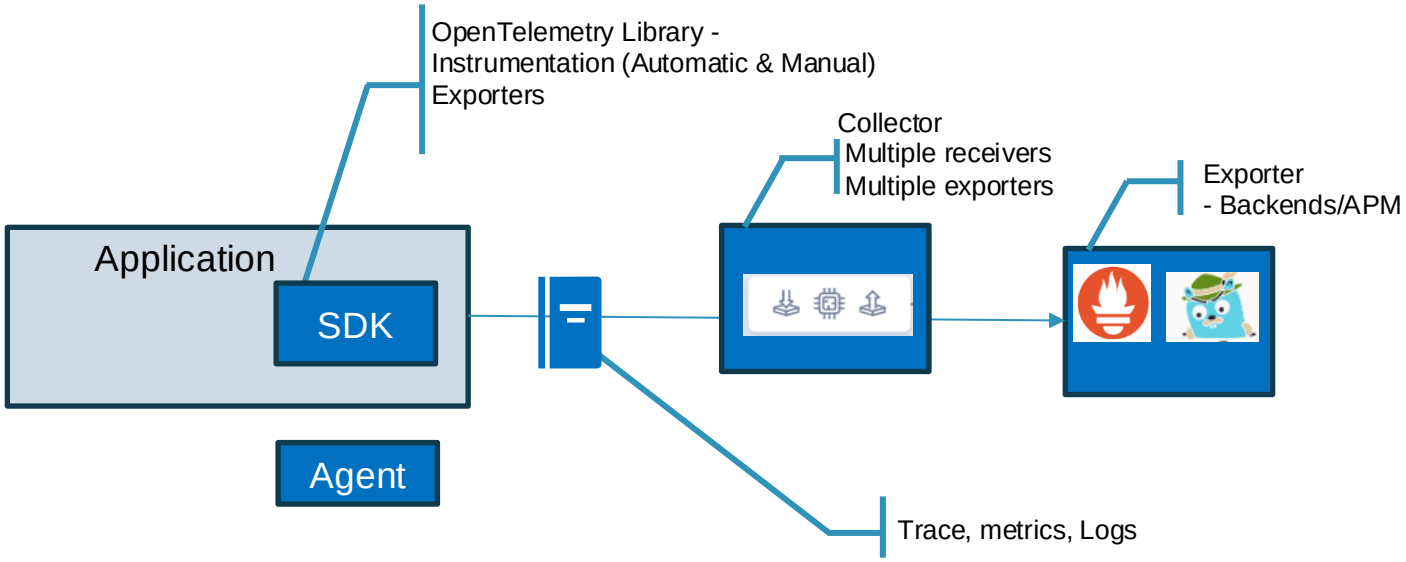
- It is an open-source observability framework for **infrastructure** & **application instrumentation.**
- It's a Protocol at heart which aims to be **vendor agnostic. It** produces SDK in multiple languages.
- As Open Telemetry doesn't provide a backend implementation (its concern is **creating, collecting, and sending signals**), the data will flow to another system or systems for stora and querying.
- 3 Signals -
    - Traces - 2020 – Capture distributed traces & propagate context.
    - Metrics 2022 – Capture Metrices from application & infrastructure.
    - Logs 2023 – Performant Logs with unified semantics.

Why OpenTelemetry?

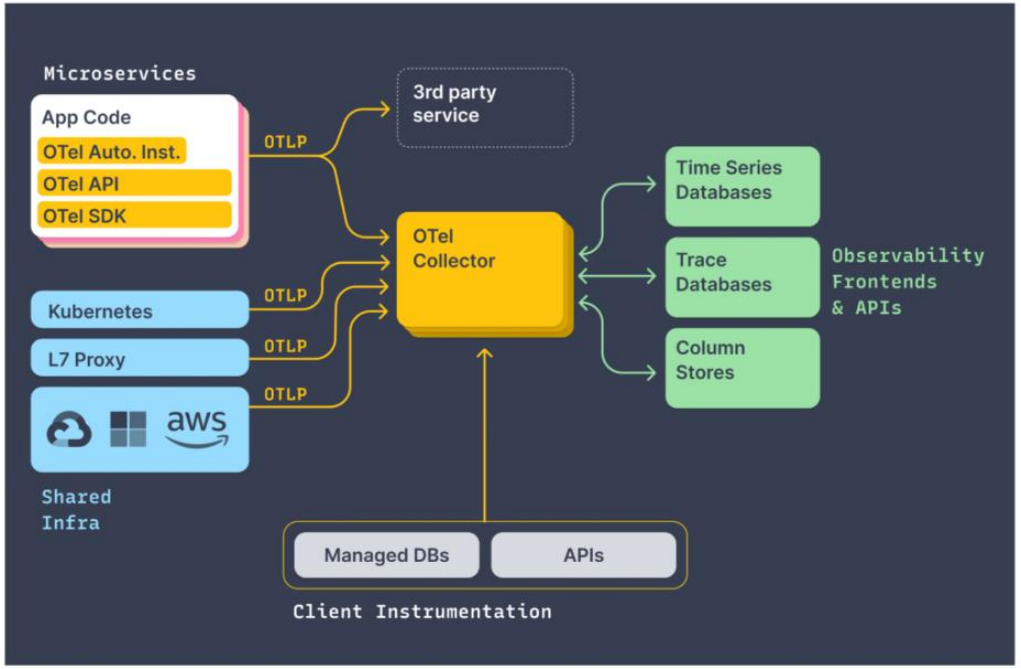


- Before Open Telemetry
  - APM vendors had their own things – SDKs, Agents, Grammer
  - Vendor lock-in.
  - Application developers, built wrappers, event hooks to keep application isolated.
  - Competing Open standards – OpenTracing (Traces), OpenCensus (Metrics, Traces)
  - Different Standards for Logs

Components of OpenTelemetry

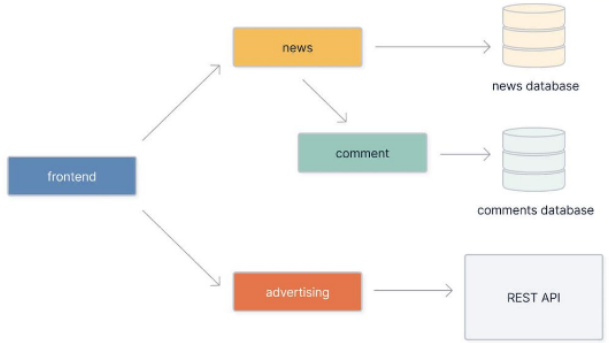


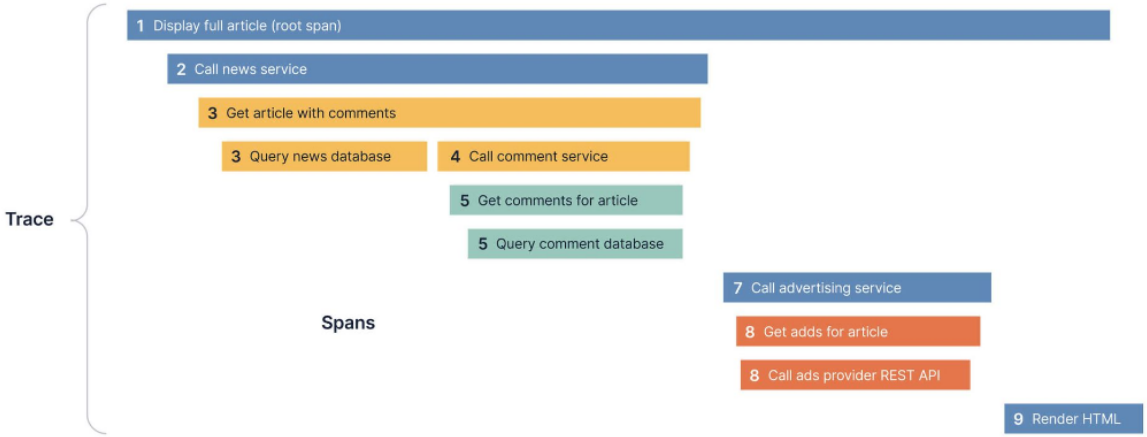
Components of OpenTelemetry



OpenTelemetry – Traces & Spans

Traces & Spans





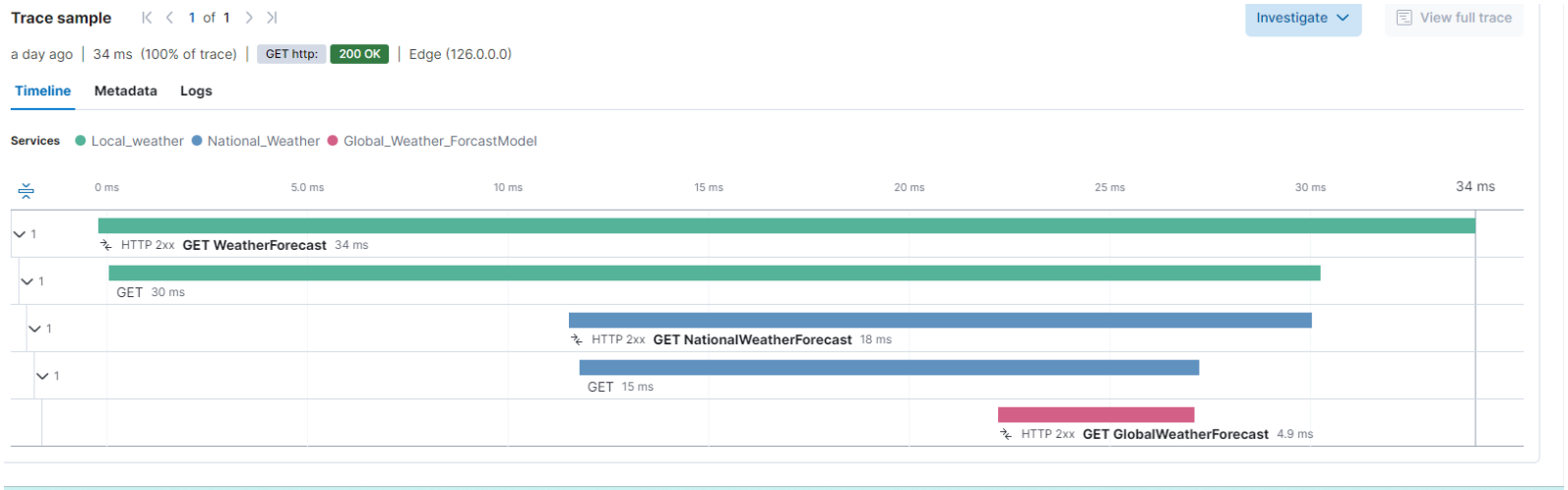
- Trace is the “Thing” being done – Usecase
- The “Thing” is composed of smaller things either locally or remotely (called Spans)
- It’s a container for Spans

Traces & Spans

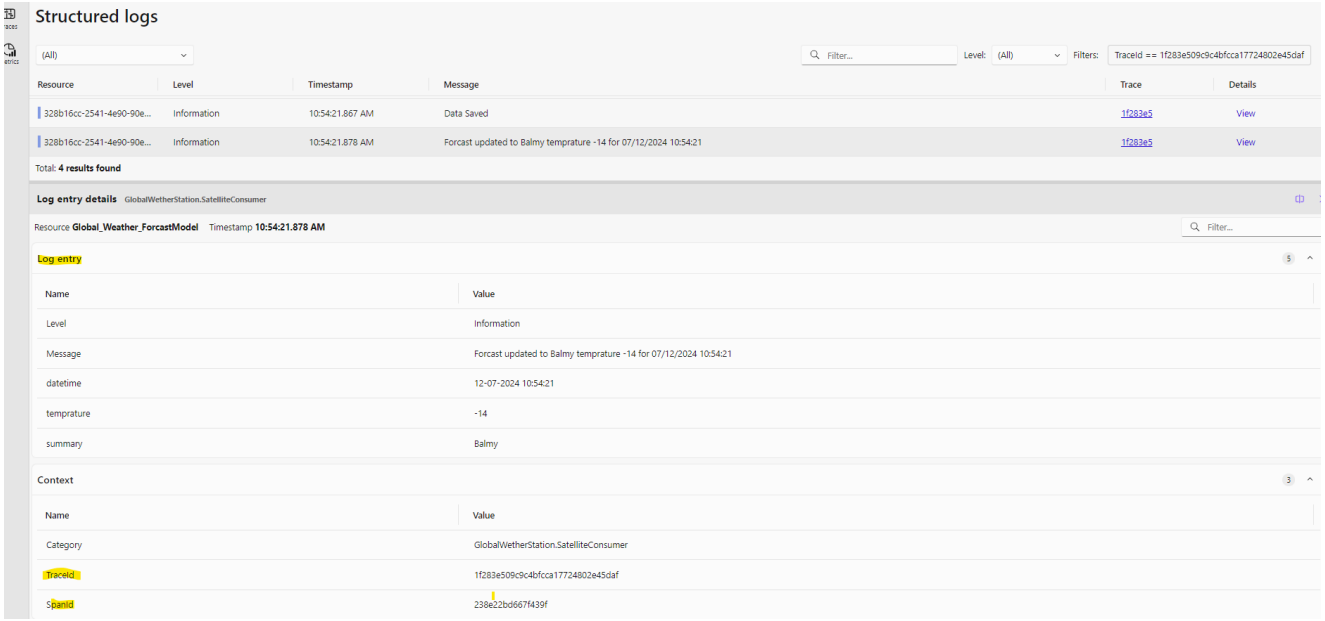
- Span is time bound analysis
- Scoped piece of work
- It is a structured blob of data
  - UniqueId (SpanID)
  - CorrelationId (TraceId)
  - CausalityID (parentSpanID)
  - Point in time action (Events)
  - Related Spans (Links – Causal vs Casual)
  - Enrich with Tags Attributes

Traces & Spans

- Spans can be in process or out of process
- Keep Spans wide than deep



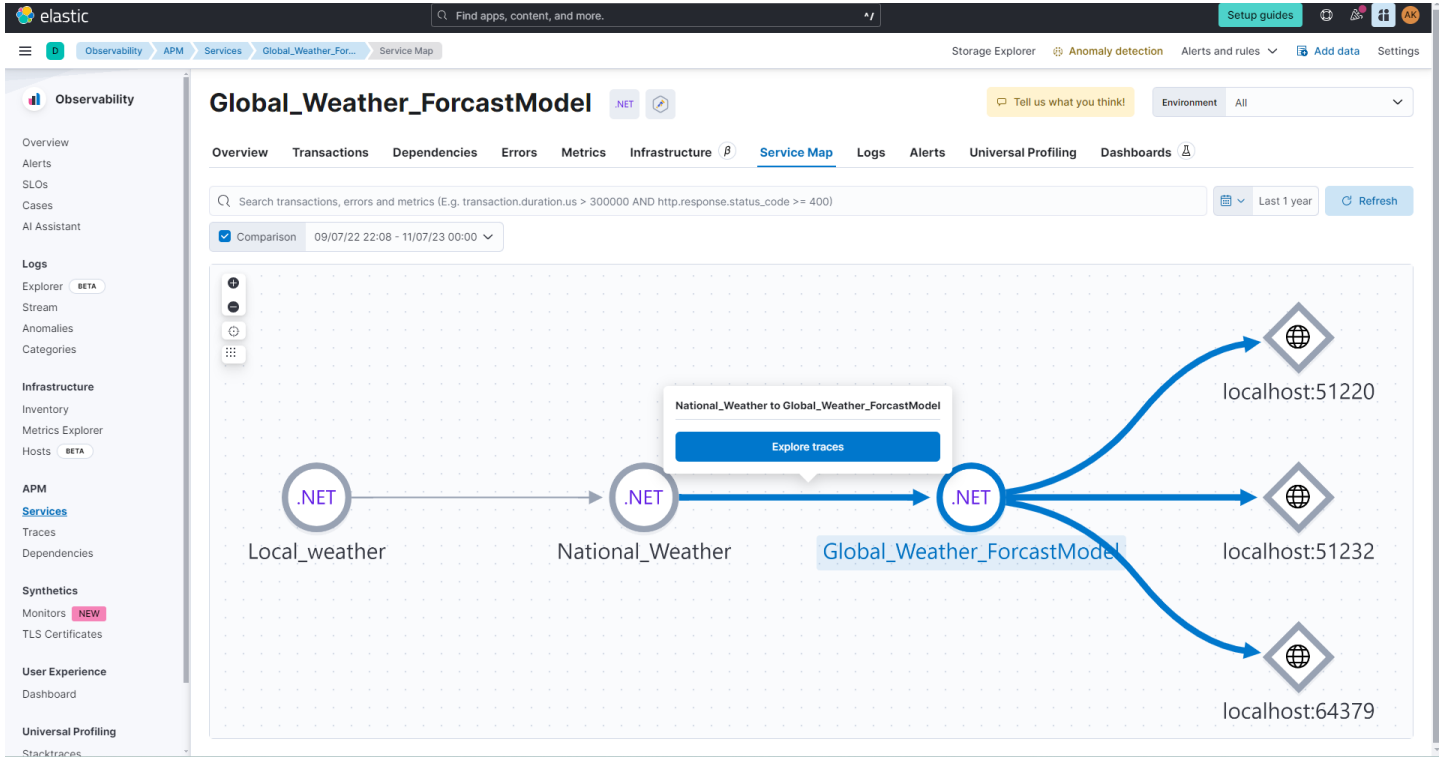
Signals of OpenTelemetry –Logs & Metrics



- Related to Spans
- Logs may be kept for several years due to legal reasons (unlike traces)
- Summary is marked deprecated

Demo

Demo


elastic
Find apps, content, and more.
Setup guides
Observability
APM
Services
Global_Weather_For...
Service Map
Storage Explorer
Anomaly detection
Alerts and rules
Add data
Settings
Observability
Overview
Alerts
SLOs
Cases
AI Assistant
Logs
Explorer
Stream
Anomalies
Categories
Infrastructure
Inventory
Metrics Explorer
Hosts
APM
Services
Traces
Dependencies
Synthetics
Monitors
TLS Certificates
User Experience
Dashboard
Universal Profiling
Global_Weather_ForcastModel
Tell us what you think!
Environment
All
Overview
Transactions
Dependencies
Errors
Metrics
Infrastructure
Service Map
Logs
Alerts
Universal Profiling
Dashboards
Search transactions, errors and metrics (E.g. transaction.duration.us > 300000 AND http.response.status_code >= 400)
Last 1 year
Refresh
Comparison
09/07/22 22:08 - 11/07/23 00:00
National_Weather to Global_Weather_ForcastModel
Explore traces
.NET
Local_weather
.NET
National_Weather
.NET
Global_Weather_ForcastModel
localhost:51220
localhost:51232
localhost:64379

Questions ?

Thank you!