Grafana Labs

Do you know what your systems are doing?

Kyle Eckhart
Senior Software Engineer
Grafana Labs

About me

- Live outside of Cleveland
- Dad of two boys (5 and 8)
- Play board games and video games
- I love being outside

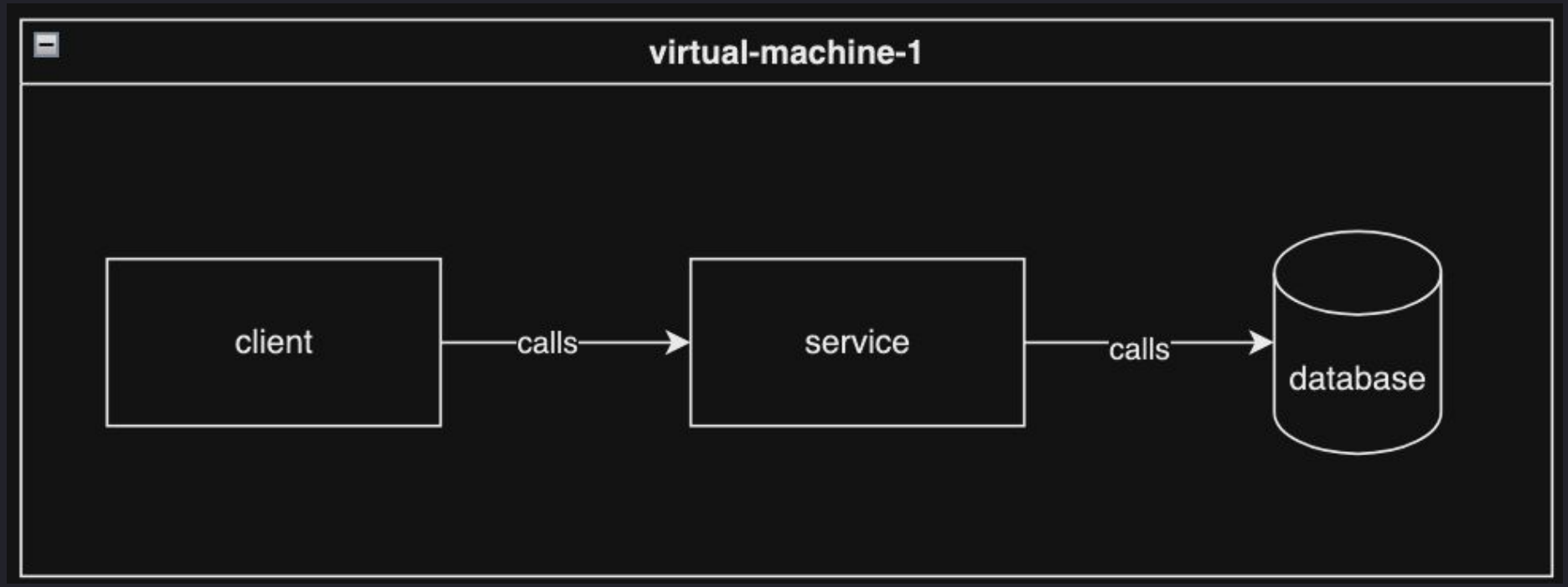
- Backend developer for 10+ years: **golang,** dotnet, Kotlin/JVM
- Tech lead for Cloud Provider Observability
- First conference talk

What is observability?

What are the base signals of observability?

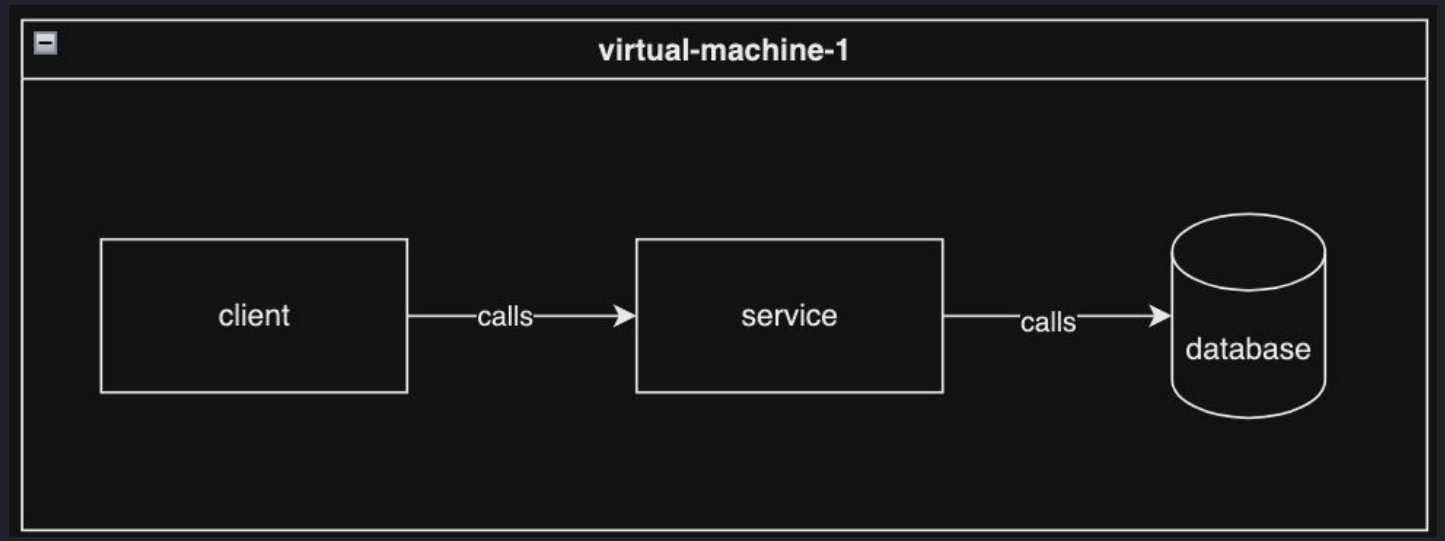
How do we get and use these signals?

What is observability?


virtual-machine-1
client
calls
service
calls
database

Monitoring Our System

- Tell me when CPU > 80%
- Tell me when Memory > 80%
- Tell me when average response times are > 1 sec



Monitoring vs Observability

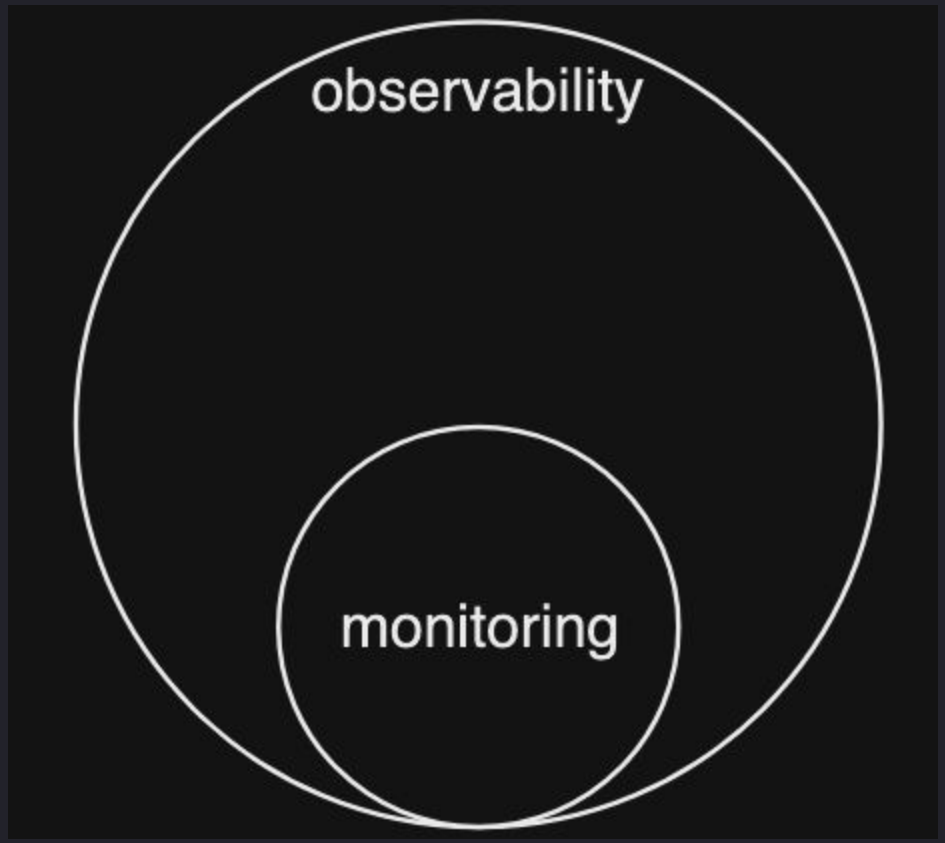
Monitoring

- CPU > 80%
- Memory > 80%
- Average response times are > 1 sec

vs

Observability

- Traffic doubled so CPU > 80%
- The APIs being called are resource intensive so Memory > 80%
- The machine is starved for resources so average response times are > 1 sec

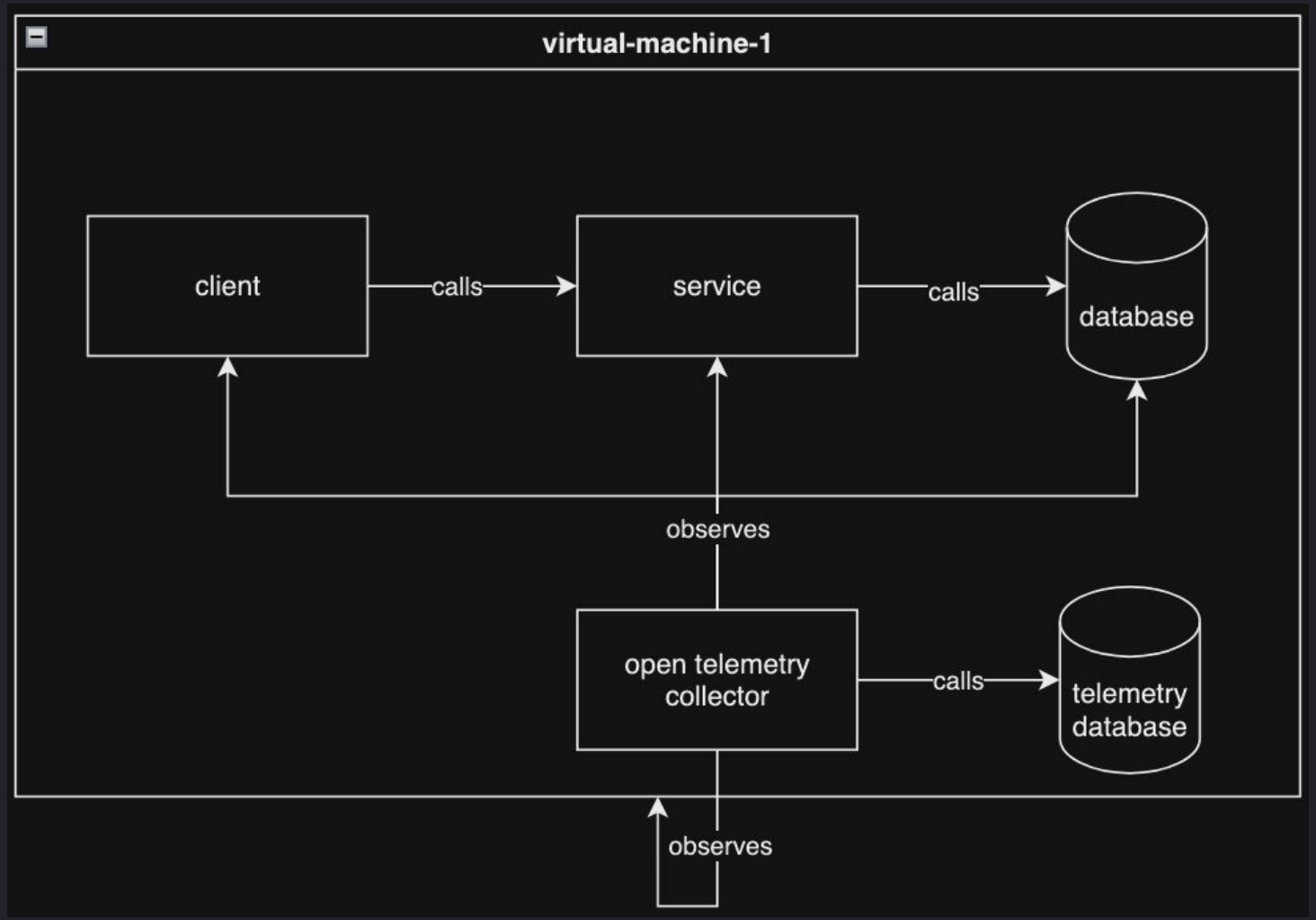

observability
monitoring

Grafana Labs

Open Telemetry

Also known as OTel, is a **vendor-neutral open source Observability framework** for instrumenting, generating, collecting, and exporting telemetry data

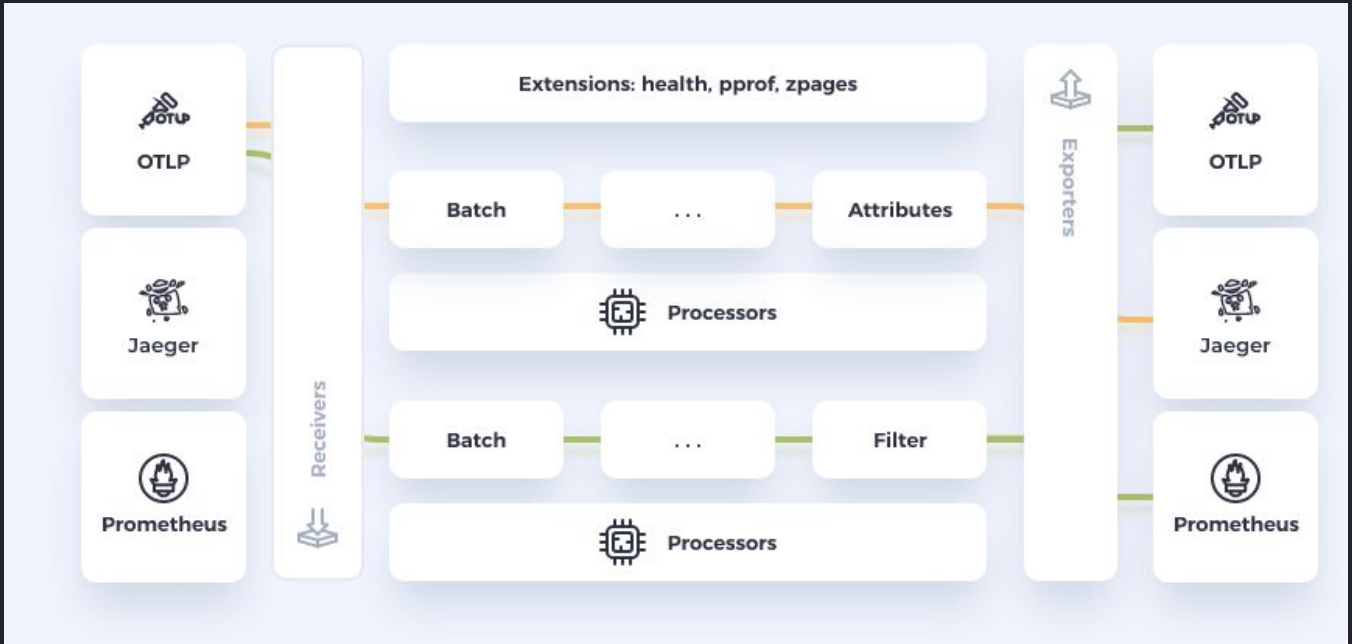
https://opentelemetry.io/docs/


virtual-machine-1
client
calls
service
calls
database
observes
open telemetry collector
calls
telemetry database
observes

And many many more

Open Telemetry Collector

- The “collecting, and exporting telemetry data” side of OTEL



What are the base signals of observability?

Metrics

- A measurement captured at runtime
- They have,
  - A Name: http.client.request.duration
  - Optional Unit: seconds
  - Optional Description
  - **A Kind**
  - **Attributes**

Metric Kind: Counter

- A value that accumulates over time only going up

Metric Kind: Counter

Metric Kind: Gauge

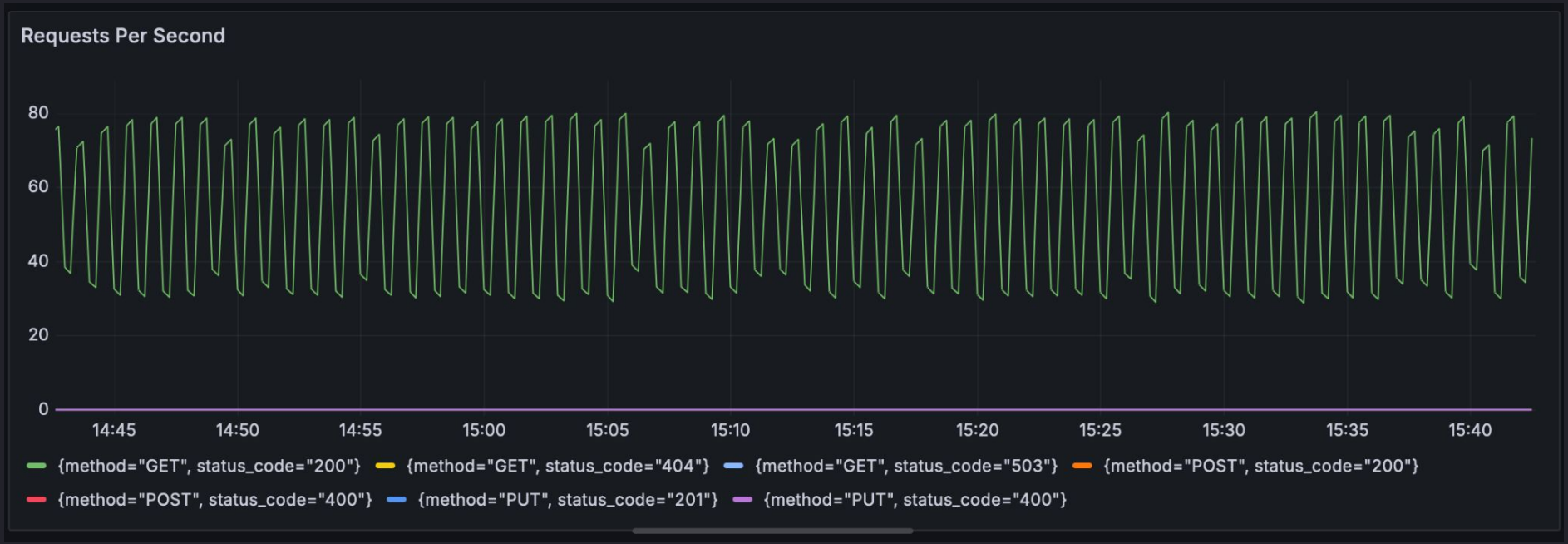
- Tracks the current value at the time it is read

Metric Kind: Histogram

- Tracks the statistical distribution of an event in a system

Metric Attributes

Metric Attributes



Logs

- A timestamped record of an event with metadata
- Structured (preferred)

```
{
    "timestamp": "2022-12-23T12:34:56Z",
    "level": "error",
    "message": "There was an error processing the request",
    "request_id": "1234567890",
    "user_id": "abcdefghij"
}
```

- Unstructured

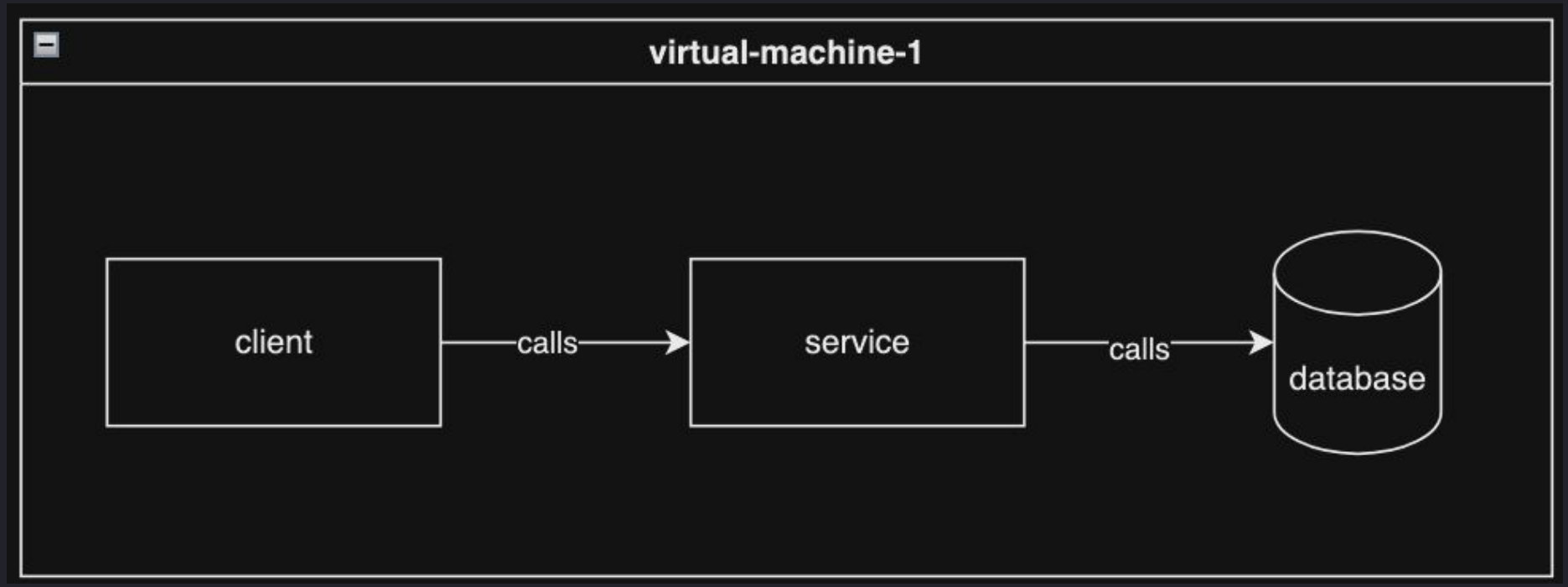
```
TLSv1.2 AES128-SHA 1.1.1.1 "Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0"
TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256 3.3.3.3 "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:58.0) Gecko/20100101 Firefox/58.0"
TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256 4.4.4.4 "Mozilla/5.0 (Android 4.4.2; Tablet; rv:65.0) Gecko/65.0 Firefox/65.0"
TLSv1 AES128-SHA 5.5.5.5 "Mozilla/5.0 (Android 4.4.2; Tablet; rv:65.0) Gecko/65.0 Firefox/65.0"
```

Logs by OTEL

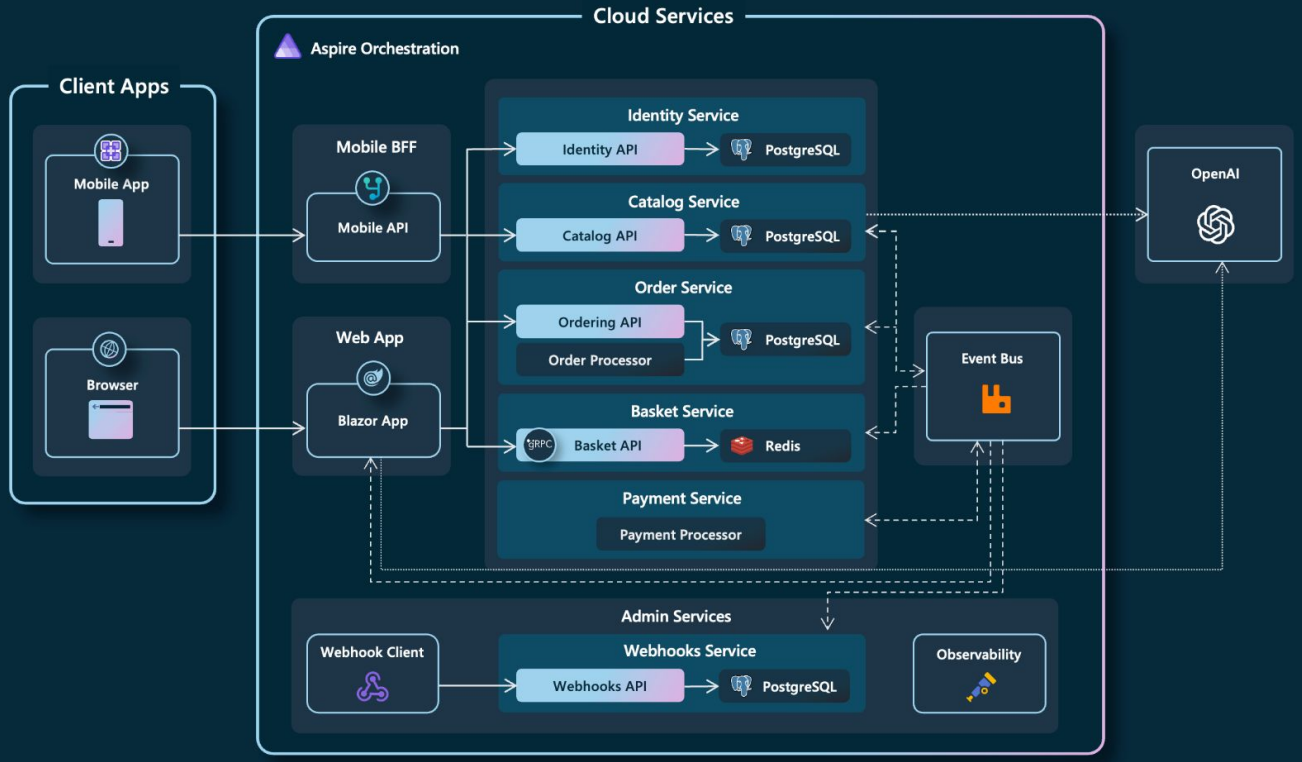
- Timestamp: Time when the event occurred.
- ObservedTimestamp: Time when the event was observed.
- **TraceId: Request trace ID.**
- SpanId: Request span ID.
- TraceFlags: W3C trace flag.
- SeverityText: The severity text (also known as log level).
- SeverityNumber: Numerical value of the severity.
- Body: The body of the log record.
- InstrumentationScope: Describes the scope that emitted the log.
- Resource: Describes the source of the log.
- Attributes: Additional information about the event.

Traces

- The path of a request through your application
- A trace is represented as a collection of “spans” where each span is a unit of work or operation
- Context Propagation helps govern how the necessary information flows through our systems to ensure we can properly associate all spans in to a single trace


virtual-machine-1
client
calls
service
calls
database

eShop reference application



https://github.com/dotnet/eShop

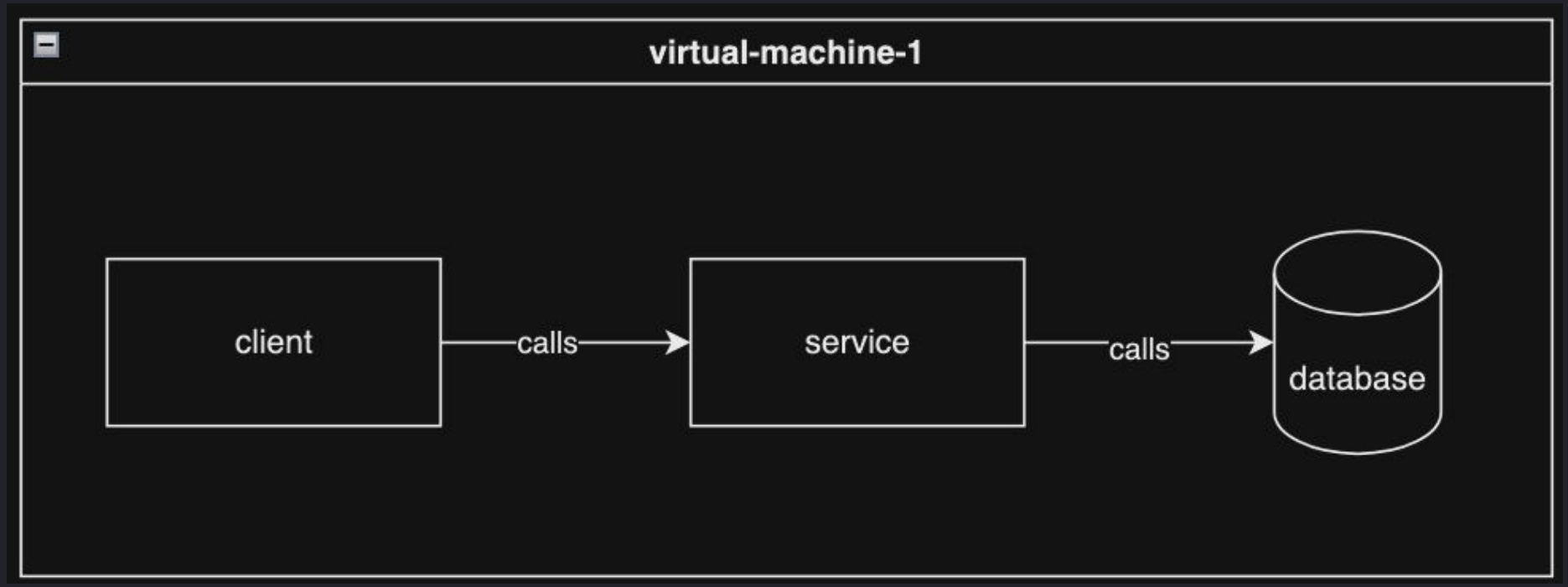
Grafana Labs

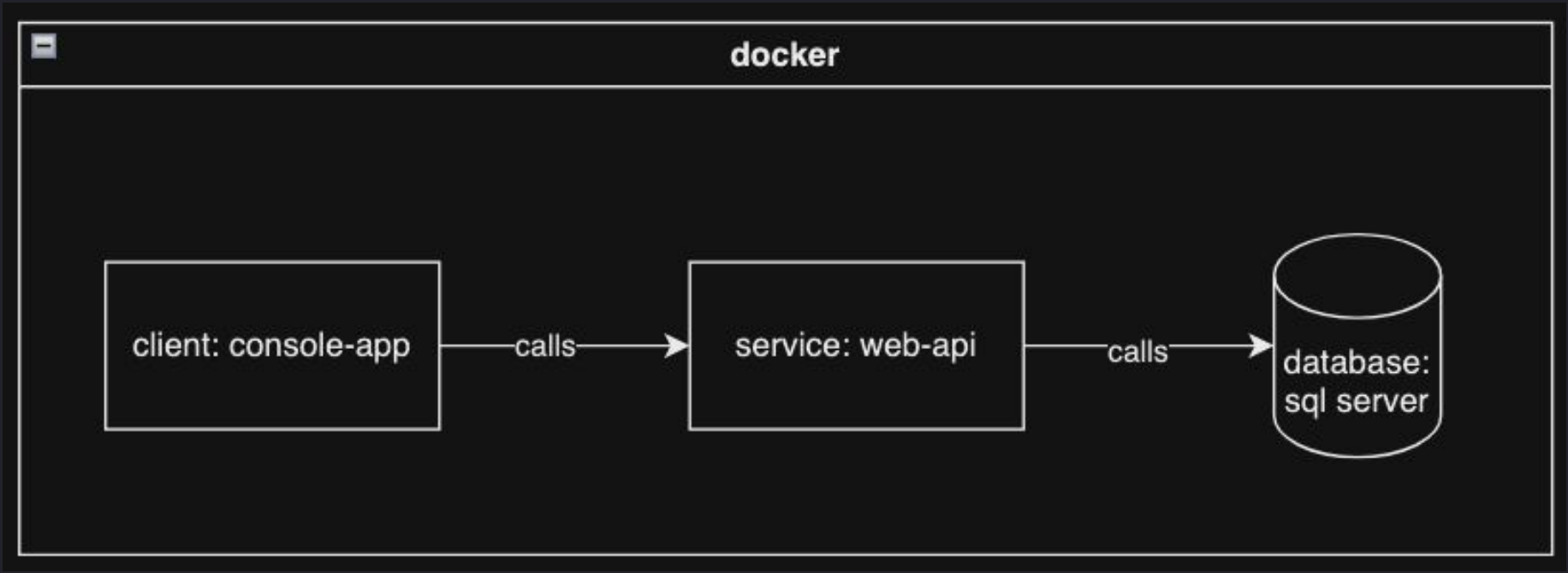
Picture removed for publication

How I use various signals

- Metrics: “big picture”
  - Monitoring
  - System troubleshooting
  - Usage tracking
- Logs: “fine grained”
  - Bug troubleshooting
  - Audit + compliance
- Traces: “what is going on here?”
  - Bug troubleshooting in a distributed system
  - Performance Analysis

How do we get these signals?


virtual-machine-1
client
calls
service
calls
database


docker
client: console-app
calls
service: web-api
calls
database:
sql server

Applications

Language	Traces	Metrics	Logs
C++	Stable	Stable	Stable
C#/.NET	Stable	Stable	Stable
Erlang/Elixir	Stable	Development	Development
Go	Stable	Stable	Beta
Java	Stable	Stable	Stable
JavaScript	Stable	Stable	Development
PHP	Stable	Stable	Stable
Python	Stable	Stable	Development
Ruby	Stable	Development	Development
Rust	Beta	Alpha	Alpha
Swift	Stable	Development	Development

dotnet: Metric

```
using System.Diagnostics.Metrics;

using var meter = new Meter("Examples.Service", "1.0");
var successCounter = meter.CreateCounter<long>("srv.successes.count", description: "Number of successful responses");

async Task<string> Handler()
{
    // .NET Diagnostics: update the metric
    successCounter.Add(1);

    return "Hello there";
}
```

dotnet: Logs

```
async Task<string> Handler(ILogger<Program> logger)
{
    // .NET ILogger: create a log
    logger.LogInformation("Success! Today is: {Date:MMMM dd, yyyy}", DateTimeOffset.UtcNow);

    return "Hello there";
}
```

dotnet: Trace

```
using System.Diagnostics;

// .NET Diagnostics: create the span factory
using var activitySource = new ActivitySource("Examples.Service");

async Task<string> Handler(ILogger<Program> logger)
{
    // .NET Diagnostics: create a manual span
    using (var activity = activitySource.StartActivity("SayHello"))
    {
        activity?.SetTag("foo", 1);
        activity?.SetTag("bar", "Hello, World!");
        activity?.SetTag("baz", new int[] { 1, 2, 3 });
        activity?.SetStatus(ActivityStatusCode.Ok);
    }
    return "Hello there";
}
```

Grafana Labs

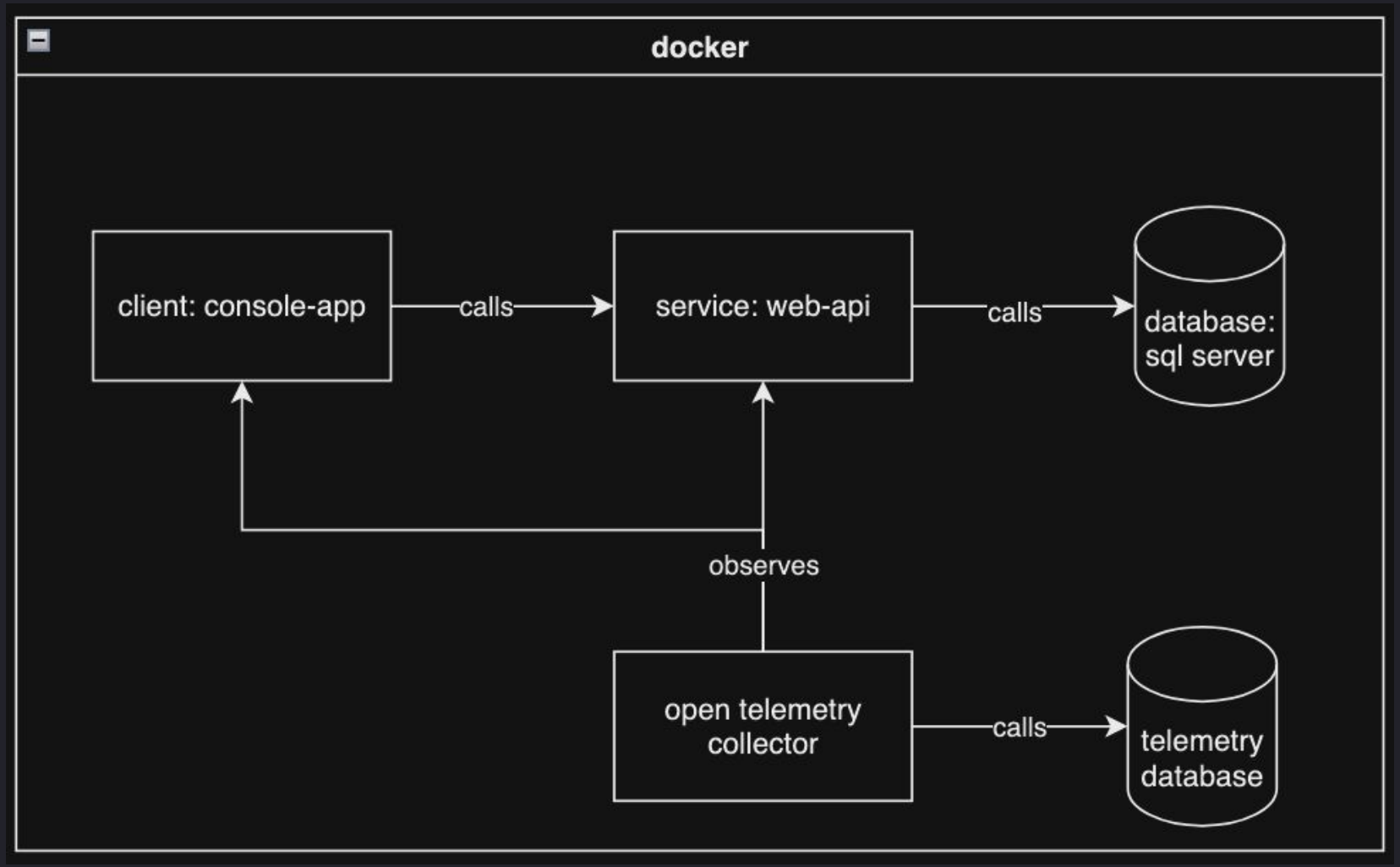
dotnet zero-code instrumentation

dotnet: OOTB Metrics

ID	Instrumented library	Documentation	Supported versions
`ASPNET`	ASP.NET Framework [1] **Not supported on .NET**	ASP.NET metrics	*
`ASPNETCORE`	ASP.NET Core [2] **Not supported on .NET Framework**	ASP.NET Core metrics	*
`HTTPCLIENT`	System.Net.Http.HttpClient and System.Net.HttpWebRequest	HttpClient metrics	*
`NETRUNTIME`	OpenTelemetry.Instrumentation.Runtime	Runtime metrics	*
`PROCESS`	OpenTelemetry.Instrumentation.Process	Process metrics	*
`NSERVICEBUS`	NServiceBus	NServiceBus metrics	≥8.0.0 & < 10.0.0

dotnet: OOTB Trace Support

ID	Instrumented library	Supported versions
ASPNET	ASP.NET (.NET Framework) MVC / WebApi [1] **Not supported on .NET**	* [2]
ASPNETCORE	ASP.NET Core **Not supported on .NET Framework**	*
AZURE	Azure SDK	[3]
ELASTICSEARCH	Elastic.Clients.Elasticsearch	* [4]
ELASTICTRANSPORT	Elastic.Transport	≥0.4.16
ENTITYFRAMEWORKCORE	Microsoft.EntityFrameworkCore **Not supported on .NET Framework**	≥6.0.12
GRAPHQL	GraphQL **Not supported on .NET Framework**	≥7.5.0
GRPCNETCLIENT	Grpc.Net.Client	≥2.52.0 & < 3.0.0
HTTPCLIENT	System.Net.Http.HttpClient and System.Net.HttpWebRequest	*
KAFKA	Confluent.Kafka	≥1.4.0 & < 3.0.0 [5]
MASSTRANSIT	MassTransit **Not supported on .NET Framework**	≥8.0.0
MONGODB	MongoDB.Driver.Core	≥2.13.3 & < 3.0.0
MYSQLCONNECTOR	MySqlConnector	≥2.0.0
MYSQLDATA	MySql.Data **Not supported on .NET Framework**	≥8.1.0
NPGSQL	Npgsql	≥6.0.0
NSERVICEBUS	NServiceBus	≥8.0.0 & < 10.0.0
ORACLEMDA	Oracle.ManagedDataAccess.Core and Oracle.ManagedDataAccess **Not supported on ARM64**	≥23.4.0
QUARTZ	Quartz **Not supported on .NET Framework 4.7.1 and older**	≥3.4.0
SQLCLIENT	Microsoft.Data.SqlClient, System.Data.SqlClient and `System.Data` (shipped with .NET Framework)	* [6]
STACKEXCHANGEREDIS	StackExchange.Redis **Not supported on .NET Framework**	≥2.0.405 & < 3.0.0
WCFCLIENT	WCF	*
WCFSERVICE	WCF **Not supported on .NET.**	*

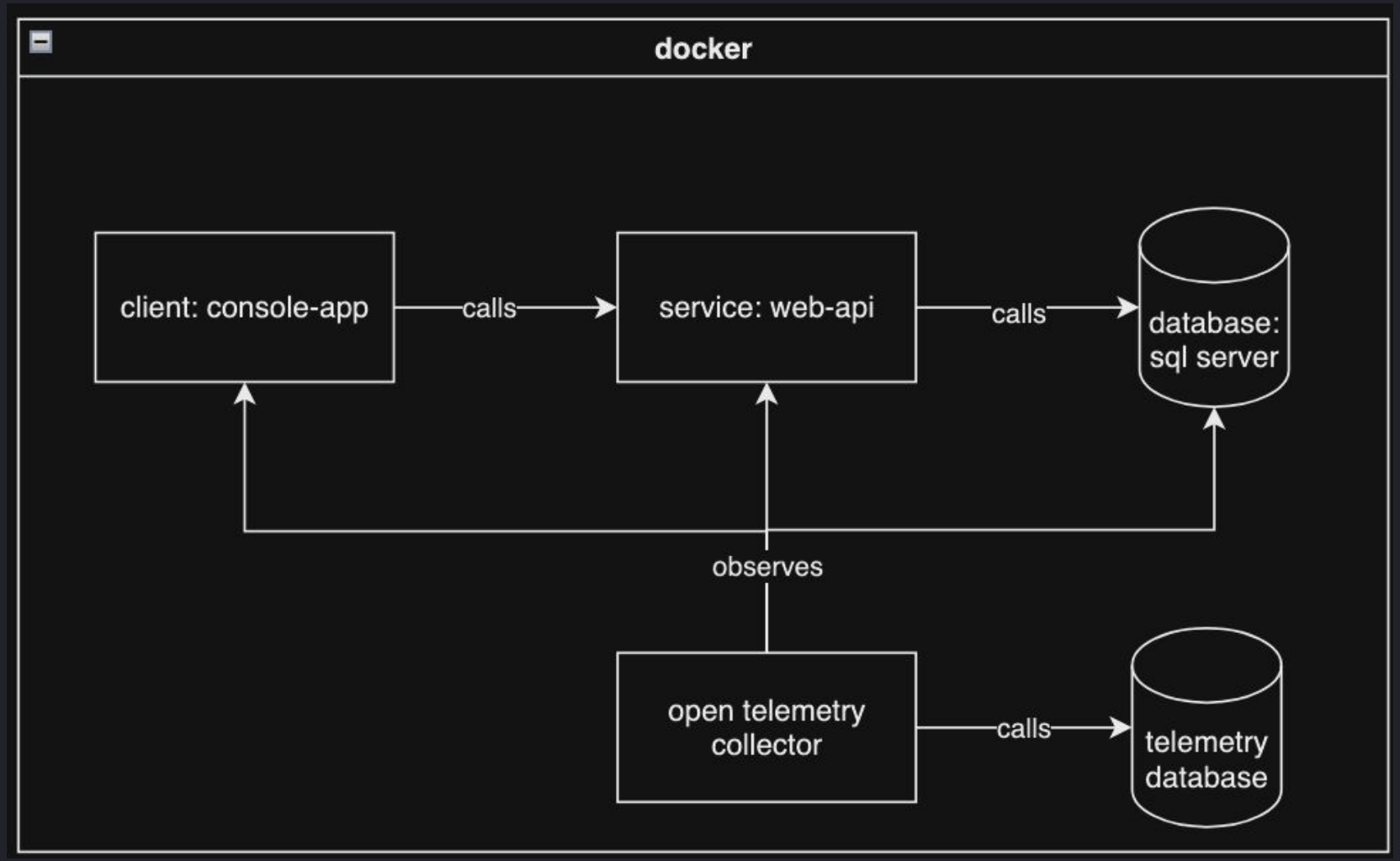

docker
client: console-app
calls
service: web-api
calls
database:
sql server
observes
open telemetry
collector
calls
telemetry
database

Open Telemetry Receivers

- A receiver is how data gets into the Open Telemetry Collector
- A receiver accepts data in a specified format, translate it to an OTEL compatible format, and pass the data through the collector
- A receiver can get telemetry via,
  - Push: data is sent to an endpoint opened by the receiver
  - Pull: the receiver is responsible for pulling the telemetry from configured locations

Telemetry from “Infrastructure”

- There’s a receiver for that https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/receiver
- No? Okay there’s probably a prometheus exporter for that https://prometheus.io/docs/instrumenting/exporters/
- SQL Server
  - https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/receiver/sqlserverreceiver (pull)
  - https://github.com/awaragi/prometheus-mssql-exporter (pull)


docker
client: console-app
calls
service: web-api
calls
database:
sql server
observes
open telemetry
collector
calls
telemetry
database

Grafana Labs

Demo!

How do we use these signals?

Know how much load our systems are currently under?

Ensure a recent release is not causing instability?

Proactively be alerted to bottlenecks and unexpected errors?

Map the impact of a bottleneck to **customer impact**?


SAY THE LINE, SENIOR DEVELOPER
"IT DEPENDS"
imgflip.com

Grafana Labs

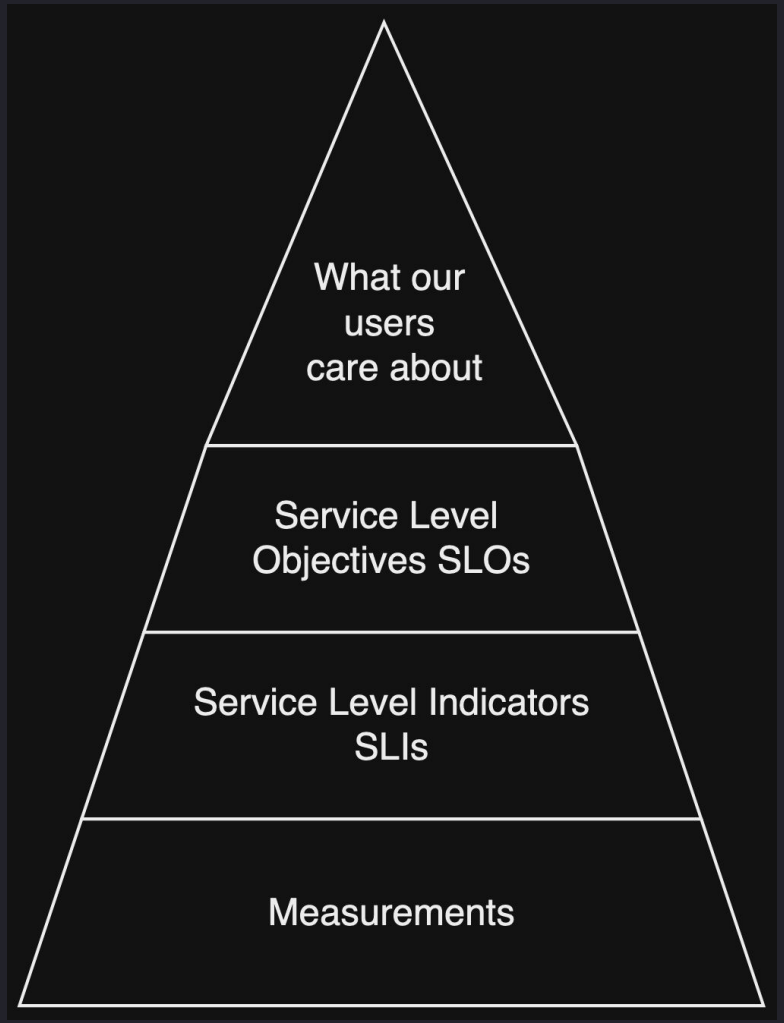
Open Telemetry

Also known as OTel, is a vendor-neutral open source Observability framework for **instrumenting, generating, collecting, and exporting telemetry data**

It’s impossible to manage a service correctly, let alone well, without understanding which behaviors really matter for that service

Chapter 4 - Service Level Objectives

Site Reliability Engineering
How Google Runs Production Systems


What our users care about
Service Level Objectives SLOs
Service Level Indicators SLIs
Measurements

The Four Golden Measurements

- Latency: How long does it take to process a request
- Traffic: How much demand is being placed on your system
  - HTTP services - number of HTTP requests per second by route
  - Database - operations per second by Query/Manipulation
- Errors: The rate of requests that fail
  - Unsuccessful response codes
  - Breaches of an agreed upon response time
- Saturation: How “full” your service is
  - CPU or Memory utilization being high
  - Disk drive filling up

Chapter 6 - Monitoring Distributed Systems

The Four Golden Measurements

- Latency: How long does it take to process a request → Histogram
- Traffic: How much demand is being placed on your system → Counter
- Errors: The rate of requests that fail → Counter
- Saturation: How “full” your service is - Gauge

Know your system

Measurement

- High latency
- Unexpected flood of traffic
- High error rate
- Disk nearing saturation

vs

User Impact

- High page load times
- DDOS? Bad actor looking for flaws?
- Pages failing to be rendered
- Potential data loss imminent

Example

“We will always accept new orders”

- SLO: The order creation API will succeed 99.999% of the time in under 15s
- SLIs
  - Latency
  - Errors
- Measurements: Http Server

Example

“Changes to available inventory will be visible to customers ASAP”

- SLO: The inventory update pipeline will succeed in less than 1 minute
- SLIs
  - End-to-end Latency
  - Errors
- Measurements
  - Time of receipt
  - Time available to application
  - Component errors

What is observability?

What are the base signals of observability?

How do we get and use these signals?

Grafana Labs

Questions