F5 NGINX

# Adding OpenTelemetry to Modern Adaptive Apps

Dave McAllister

NGINX

NGINX
Part of F5

# Modern Adaptive Apps

*Are defined by their capabilities not their implementation.*

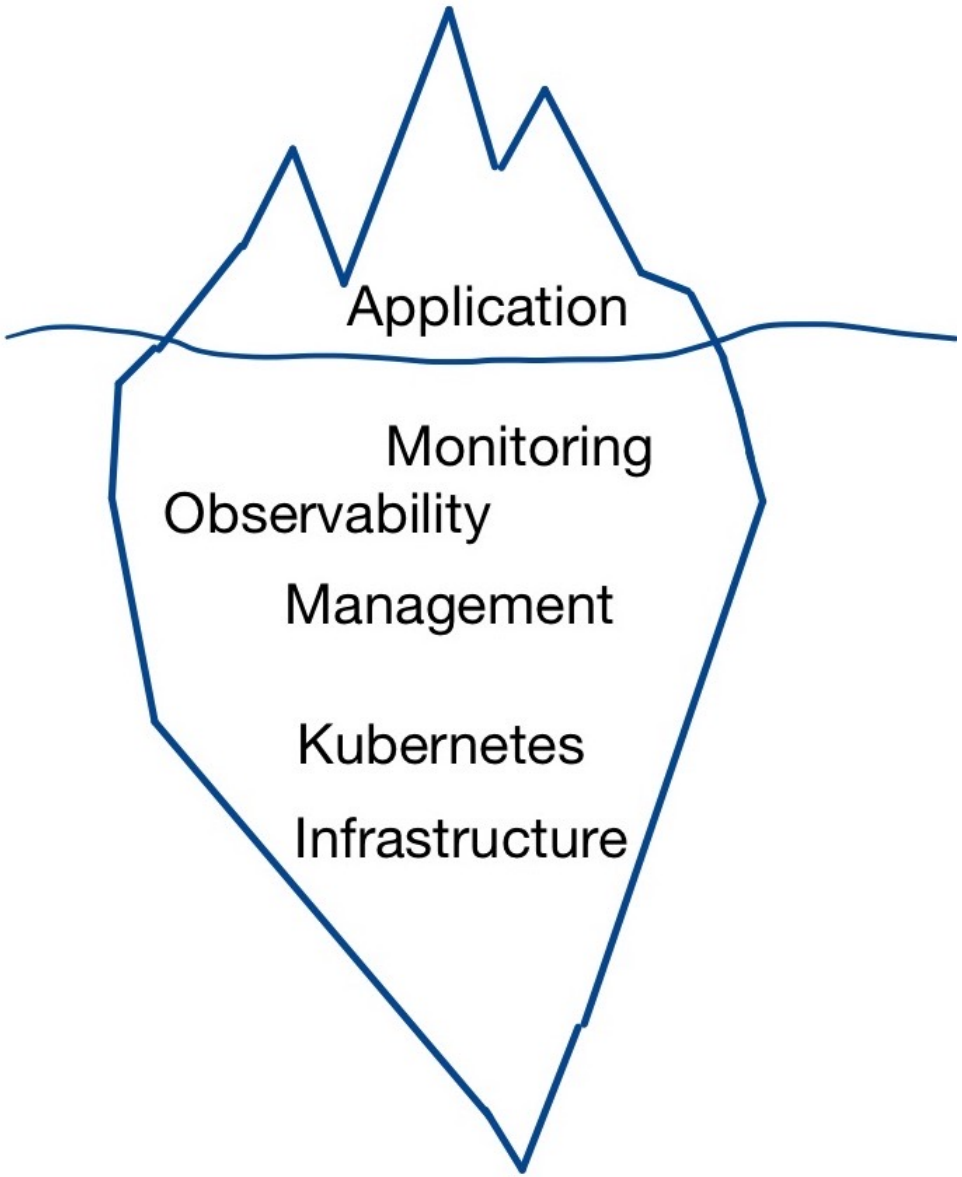
*We speak about them using terms like portable, scalable, observable, reproducible, and debuggable.*

- Deploy in minutes
- **Quickly find performance bottlenecks**
- Easily change hosting provider
- Scale up/down quickly
- Gracefully degrade upon failure
- Serve the customers' needs
- **Provide answers to platform engineers' questions**
- Protect itself from attacks
- Manage state in a knowable way
- **Provide context on errors and crashes**
- Costs scale reasonable with consumption

# What is our target?

A microservices architected app using Kubernetes that aims to be as production ready as possible.

*The application is only one part of the modern application deployment*



# Observability is a data problem

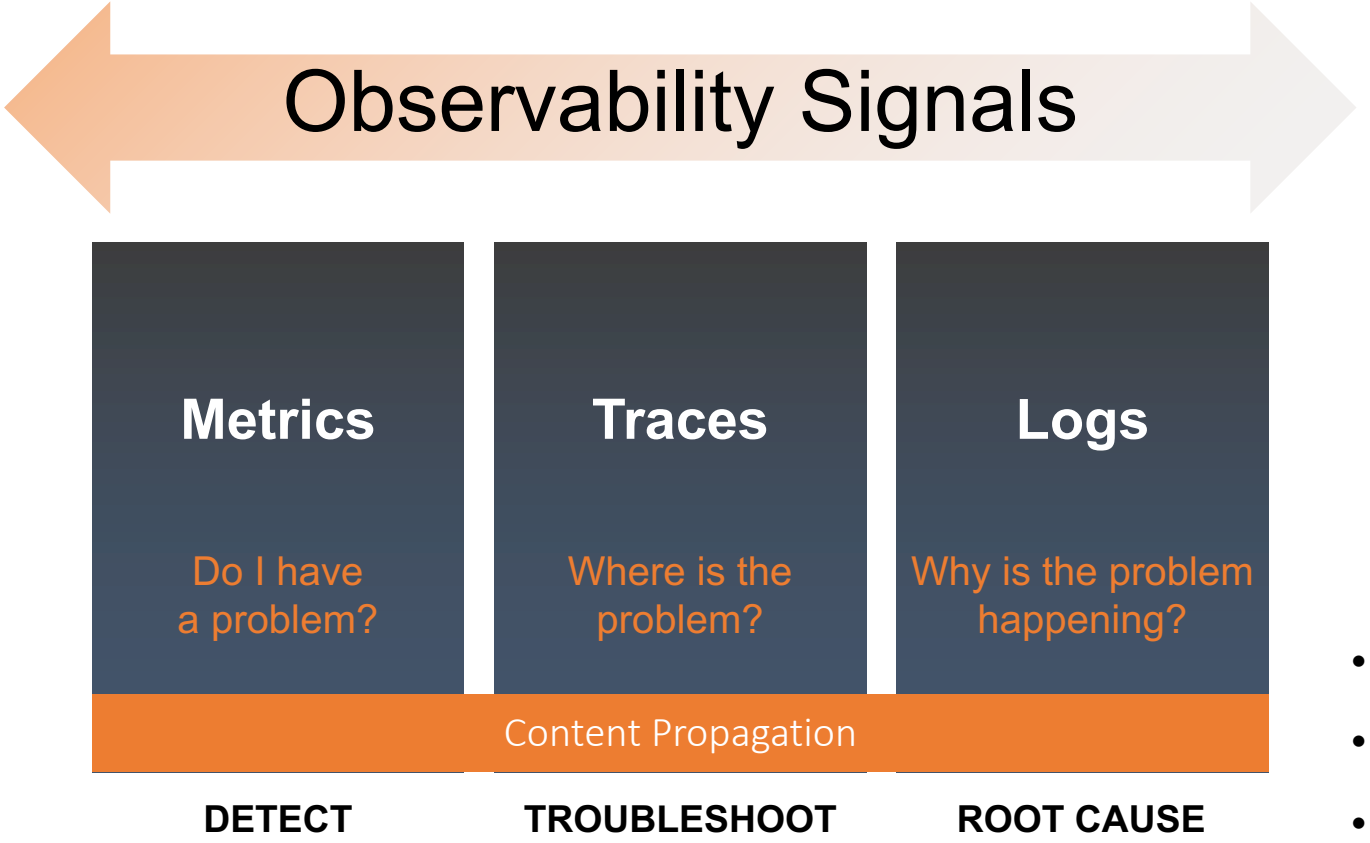
# Observability is a data problem

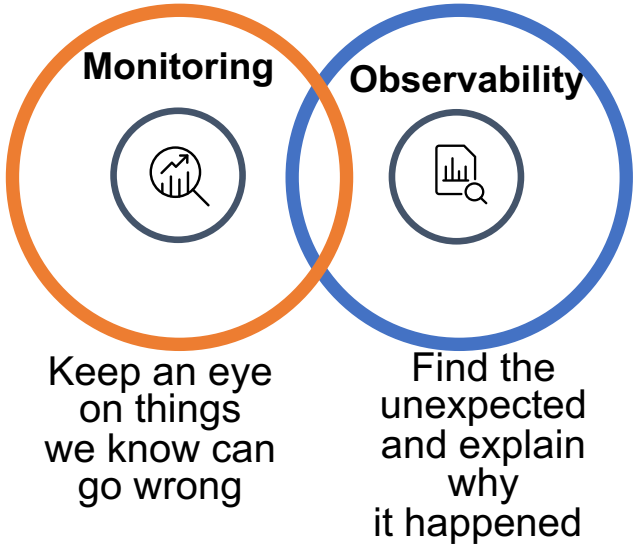
*The more observable a system, the quicker we can understand why it's acting up and fix it*

# Observability

Observability helps detect, investigate and resolve the *unknown unknowns* – FAST



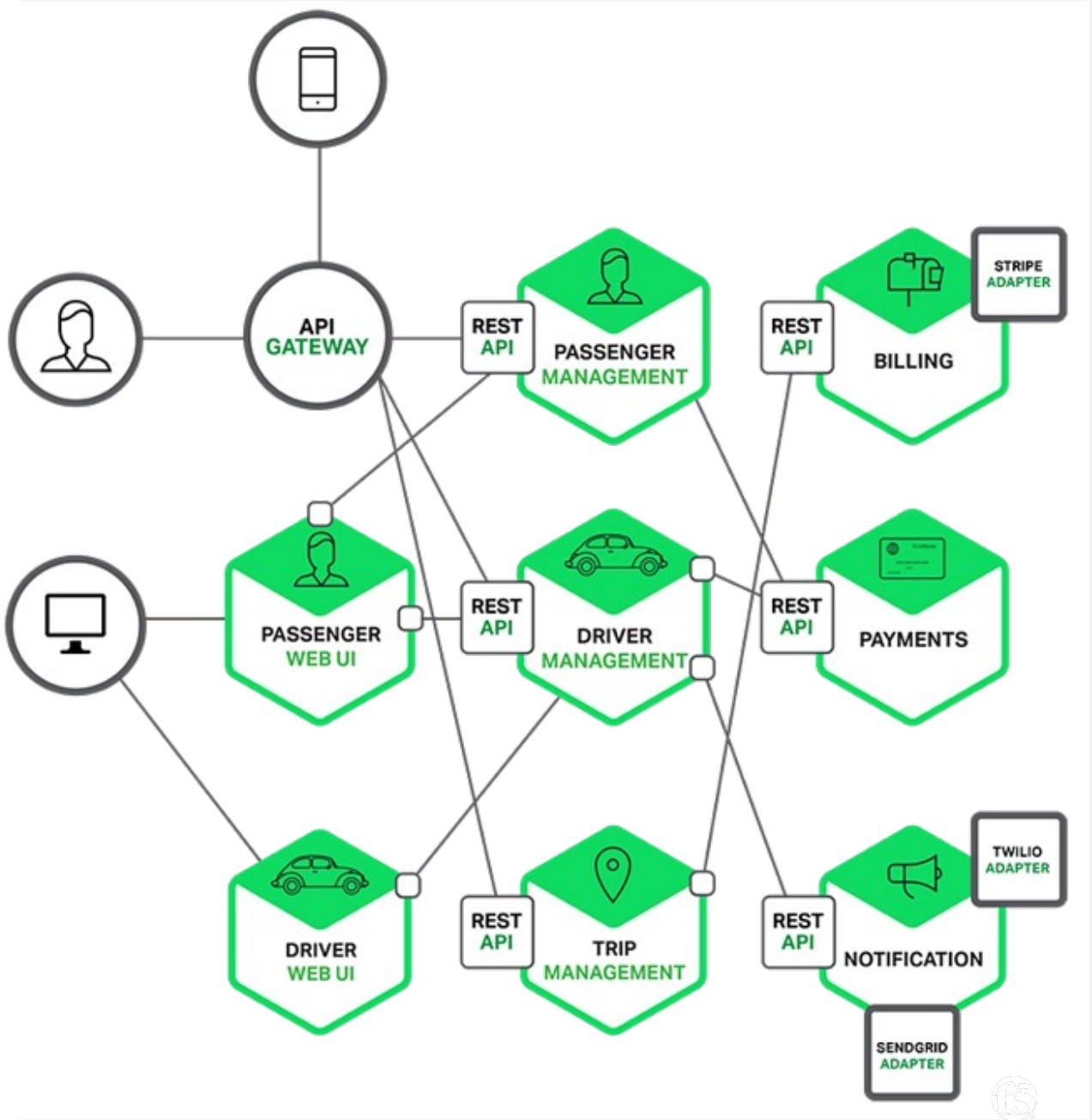

- Better visibility to the state of the system
- Precise and predictive alerting
- Reduces Mean Time to Clue (MTTC) and Mean Time to Resolution (MTTR)

# So Why Observability?

# Microservices!

Single application composed of many loosely coupled and independently deployable smaller services

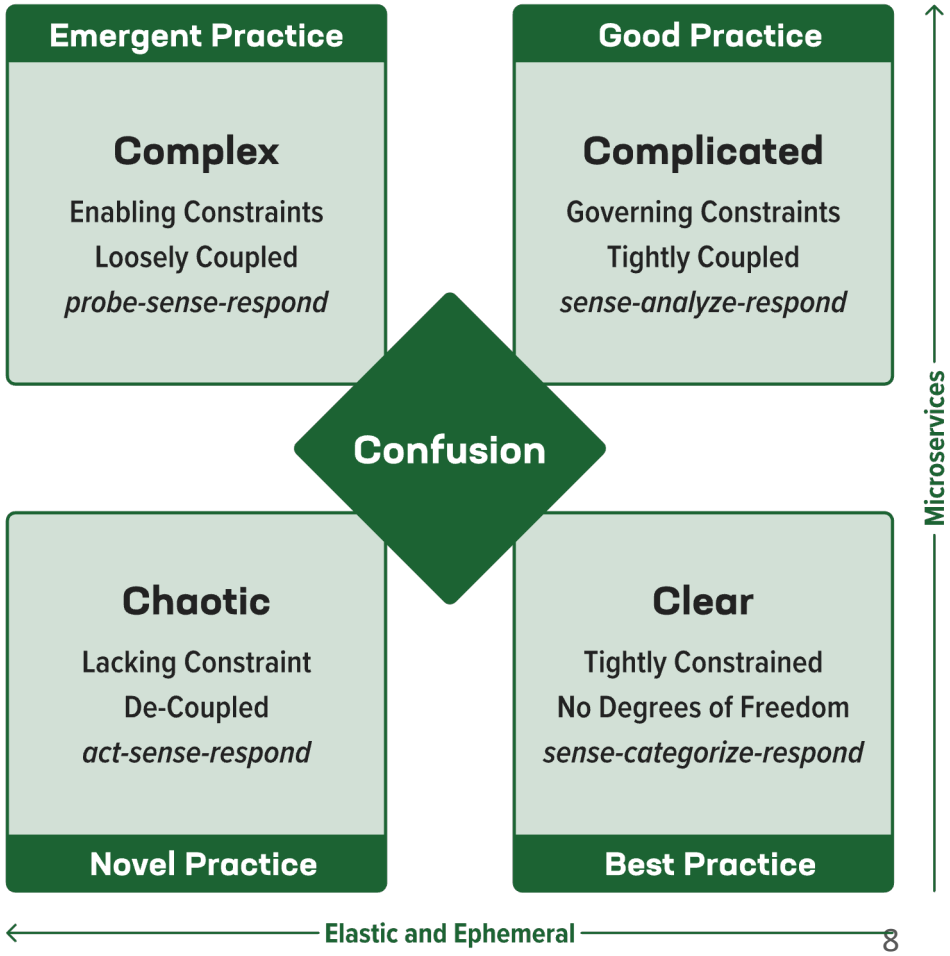
- Often polyglot in nature
- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Often in Cloud environments
- Organized around business capabilities
- Each potentially owned by a small team



# But They Add Challenges

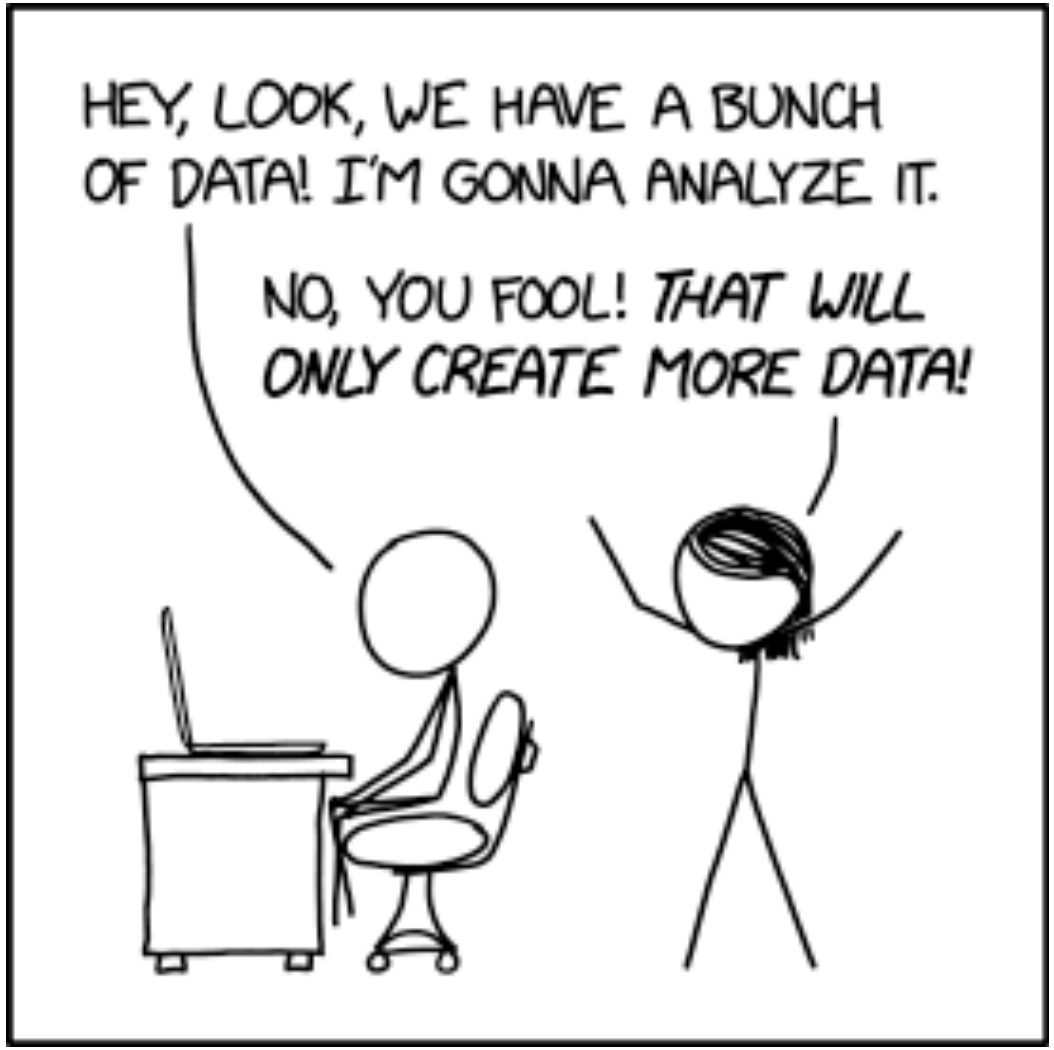
Especially when we consider this in a cloud

- Microservices create complex interactions.
- Failures don't exactly repeat.
- Debugging multitenancy is painful.
- So much data!




Cynefin Framework

# The problem data - there’s so much of it



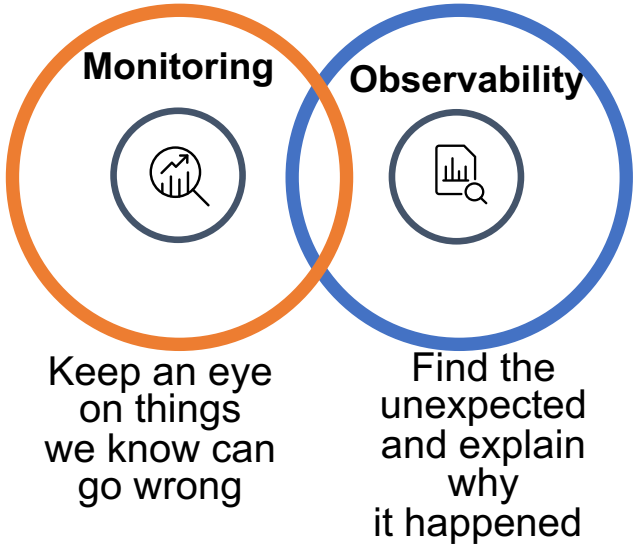



[xkcd: Data Trap]

# Observability -

Observability helps detect, investigate and resolve the *unknown unknowns* – FAST


| Metrics | Traces | Logs |
|---|---|---|
| Do I have a problem? | Where is the problem? | Why is the problem happening? |
| **DETECT** | **TROUBLESHOOT** | **ROOT CAUSE** |




- Better visibility to the state of the system
- Precise and predictive alerting
- Reduces Mean Time to Clue (MTTC) and Mean Time to Resolution (MTTR)

# Identifying the needs : Wishlist

| Technology | Logging | Tracing | Metrics | Error Agg | Health Checks | Runtime Intro | Heap/Core Dumps |
|---|---|---|---|---|---|---|---|
| Elastic APM | Yes | Yes | Yes | Yes | Yes | No | No |
| Grafana | Yes | Yes | Yes | Yes | Yes | No | No |
| Graylog | Yes | No | No | No | No | No | No |
| Jaeger | No | Yes | No | Yes | No | No | No |
| OpenCensus | No | Yes | Yes | No | No | No | No |
| OTel | Beta | Yes | Yes | Yes | Yes | No | No |
| Prometheus | No | No | Yes | No | No | No | No |
| Statsd | No | No | Yes | No | No | No | No |
| Zipkin | No | Yes | No | Yes | No | No | No |

# Time to get Qualitative

The OpenTracing project is *archived*. Learn more.
Migrate to OpenTelemetry today!

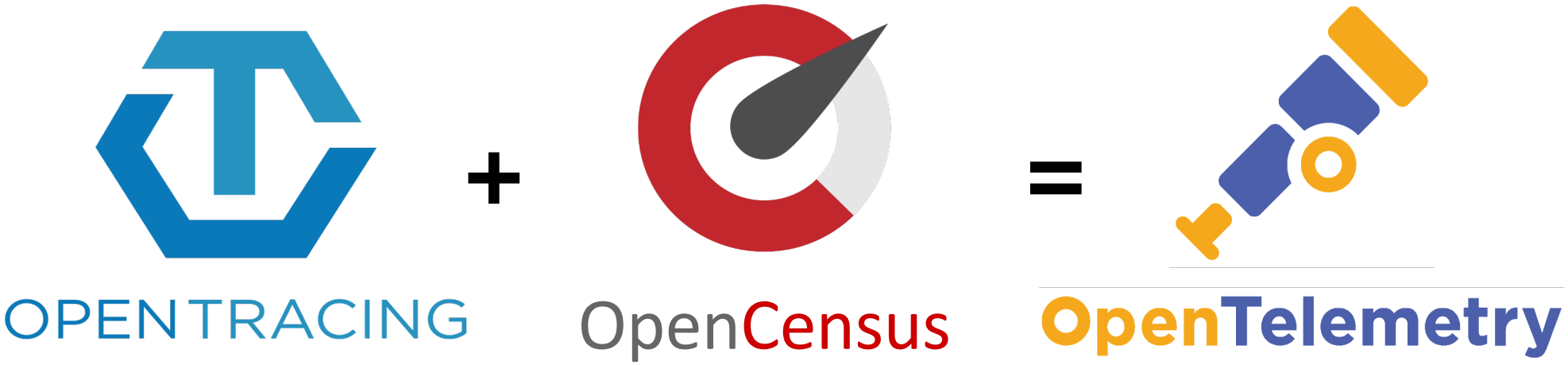
OpenCensus and OpenTracing have merged into OpenTelemetry!

OpenCensus

*Easily collect telemetry like metrics and distributed traces from your services*

# What is OpenTelemetry?

- Standards-based agents, cloud-integration
- Automated code instrumentation
- Support for developer frameworks
- Any code, any time



# Cloud Native Telemetry

Telemetry "verticals"

Telemetry "layers"

| | Tracing | Metrics | Logs, etc |
|---|---|---|---|
| Instrumentation APIs | foreach(language) | | |
| Canonical implementations | foreach(language) | | |
| Data infrastructure | collectors, sidecars, etc | | |
| Interop formats | w3c trace-context, wire formats for trace data, metrics, logs, etc | | |

# Why Open Telemetry Matters

OpenTelemetry users build and own their collection strategies, without vendor lock-in.

OpenTelemetry puts the focus on **analytics** not **collection.**


OpenTelemetry

# MARA and Open Telemetry: The Vision

1

Integration of OTEL tracing into MARA

2

Convert Metrics to use OTEL

3

Convert Log Management to use OTEL

4

Operationalize OTEL

# Our Considerations

- Avoid Lock In
  - Ability to switch between observability technologies
- Ease of Use
  - Reduction in friction for implementation
  - Automated instrumentation when possible
- Visualization Tooling
  - Ability to use and correlate data to make decisions
- Resource Use

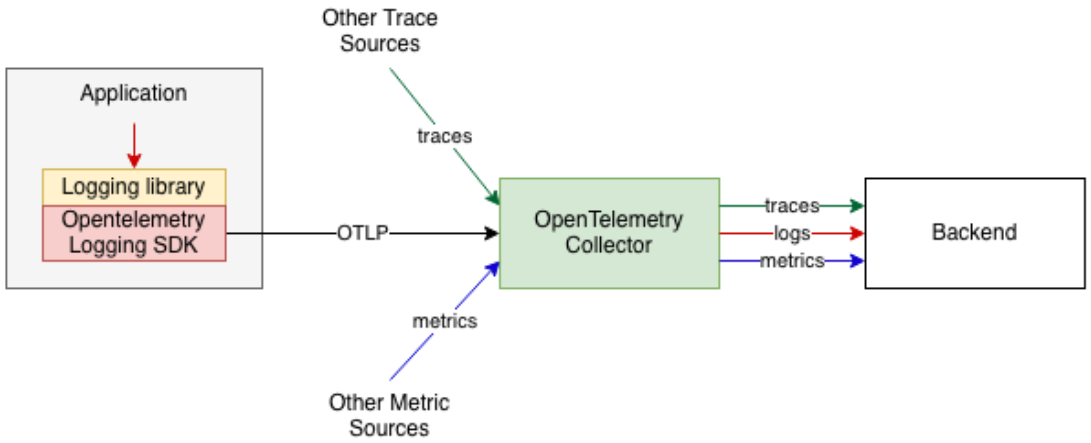
# Let’s start with logs

We all know what logs are but their simplicity rapidly leads to some complex decisions

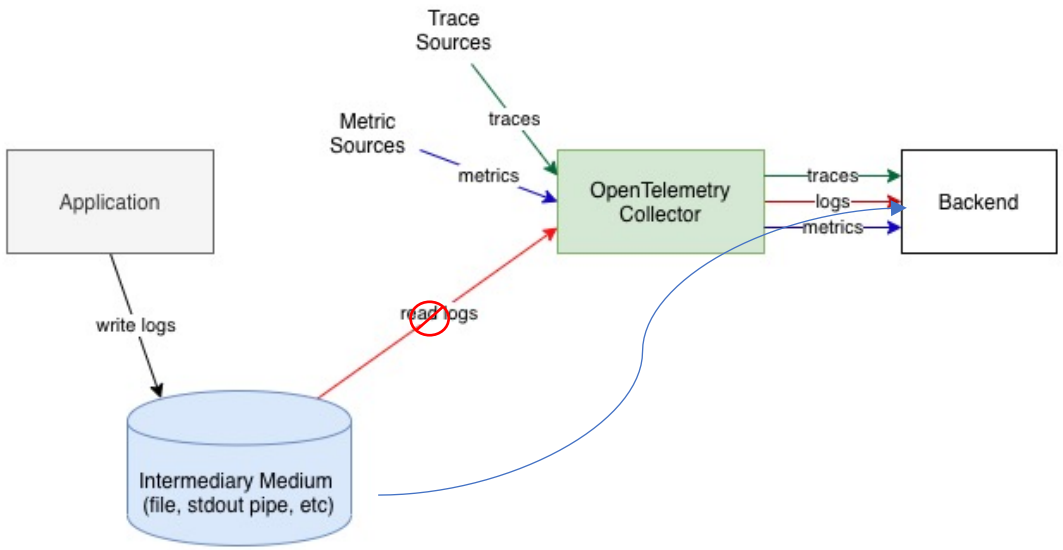
- Simple - Harvest the log output
- Complicated – Transport, Storage, indexing (?), lifetime

To be useful, our log files must be easily searchable based on varying criteria

# What we wanted

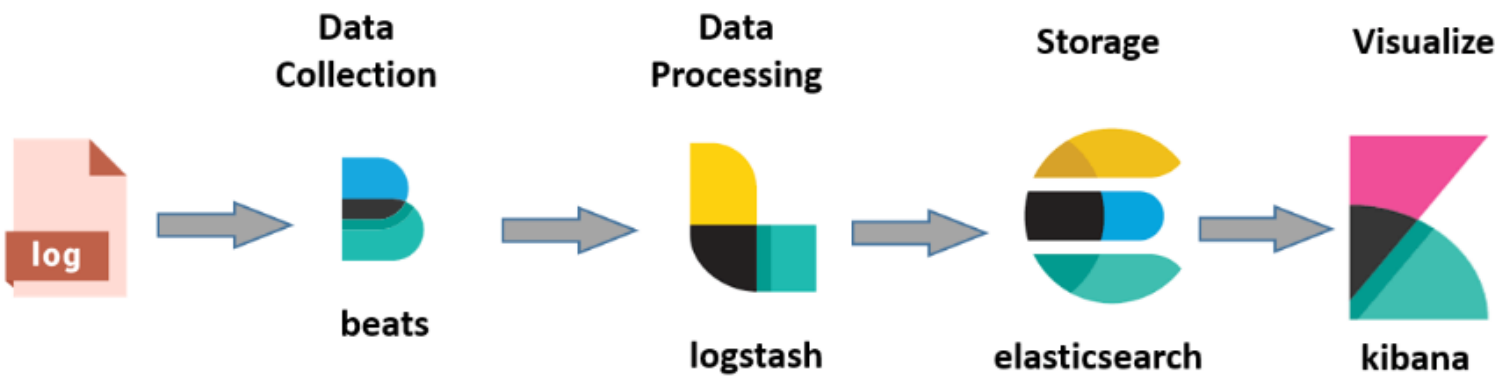



# What we got



# Elastic Stack, for now

- Filebeat – data transport in Kubernetes DaemonSet
- Bitnami – chart to split the deployment into ingest, co-ordinating, master and data nodes
- Kibana – search + pre-loaded stuff

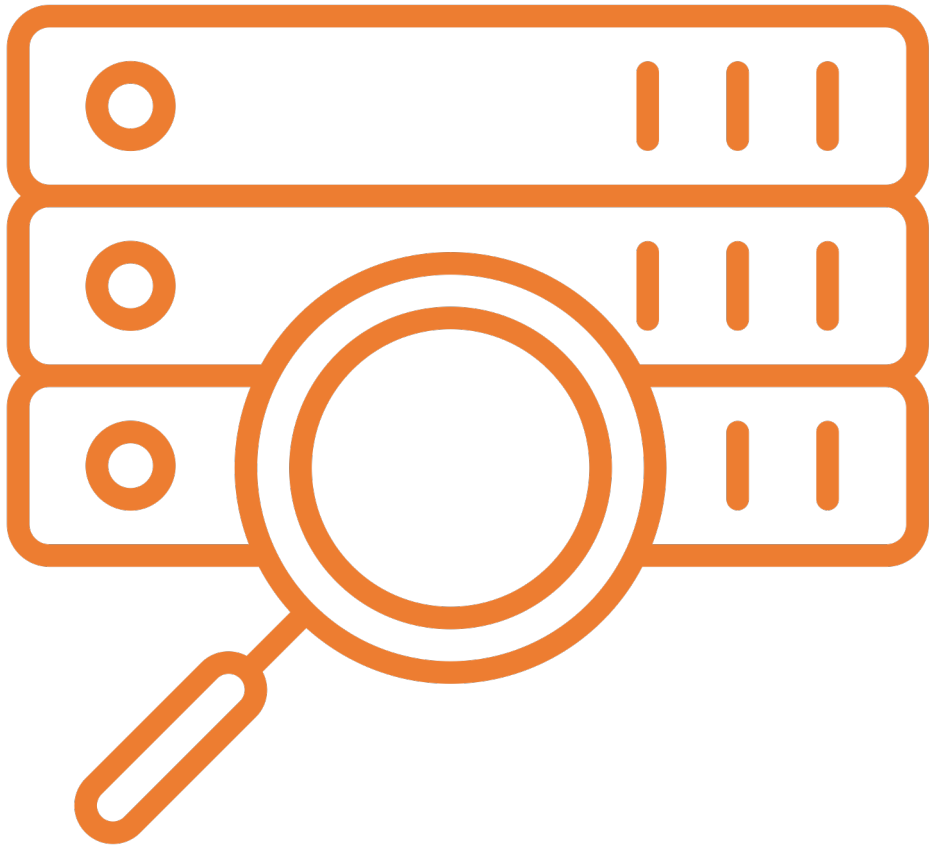



It works, but

- Extremely resource hungry
- Query variance is okay

# Metrics: Prometheus and Friends

- Deployed via helm
  - kube-prometheus-stack
- Chart Deploys
  - Prometheus Operator
  - Prometheus Rules
  - Grafana deployment and dashboards
  - Does not deploy IC dashboards
- Uses service monitors for dynamic scraping
- Configured to work with statsd
  - Deployed to monitor Prometheus
- Available to all MARA components



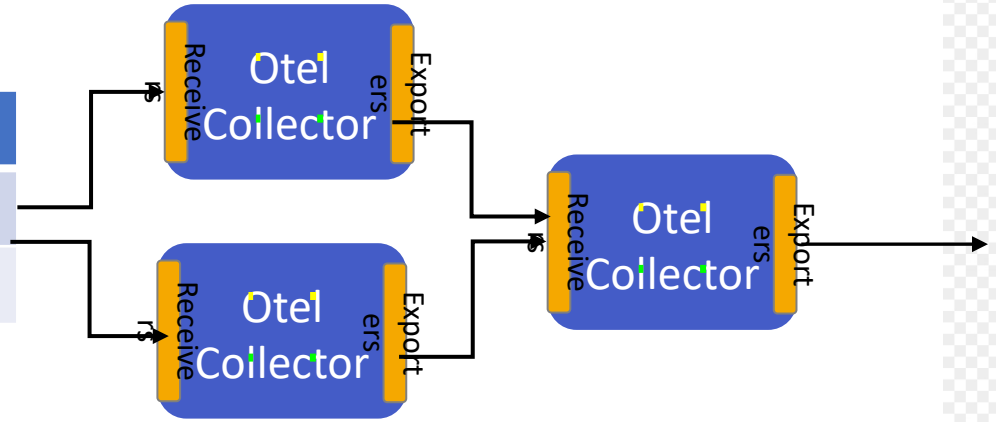
# Distributed Tracing

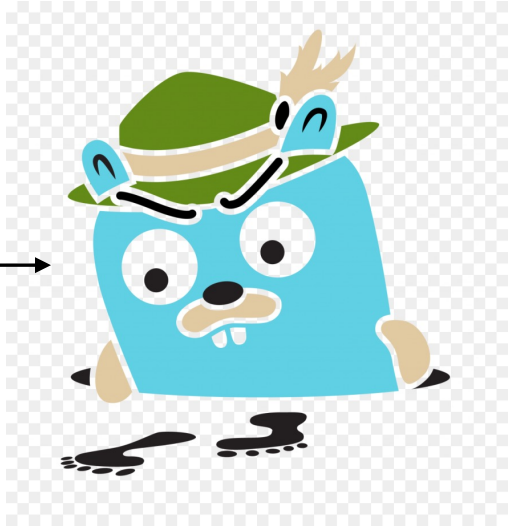
Complex and semi-chaotic

- Must not impact QoS of the application
- Must support all desired languages

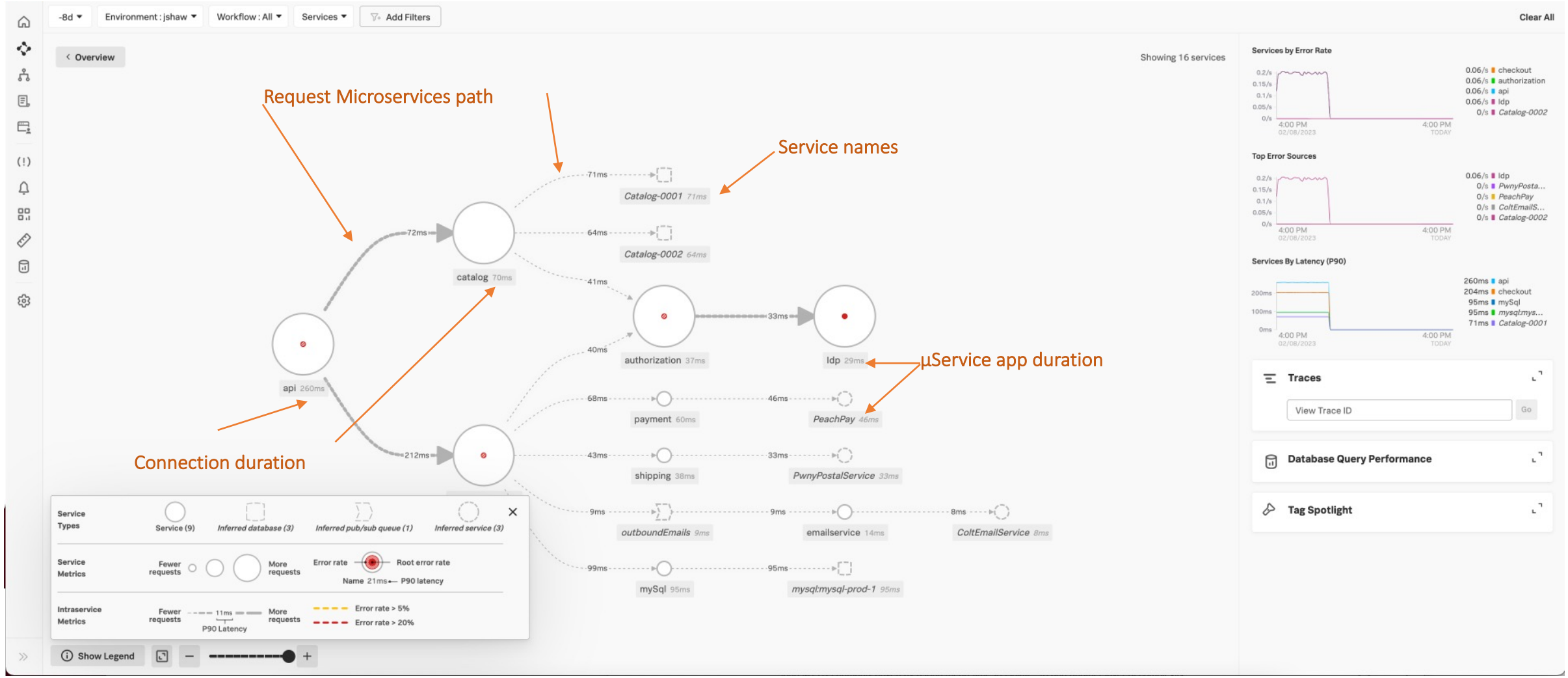
- OTEL operator is deployed
  - Several options for collector deployment
  - Includes log to console, Lightstep, Sumo Logic
  - All POC
- App (s) instrumented for tracing
- OTEL is still developing

| Language | Framework | Services |
|---|---|---|
| Java | Sprint Boot | 3 |
| Python | Flask | 3 |



# Let’s look at a trace



# A different way of looking at a trace


Request Microservices path

Service names

µService app duration

µService total performance

Note the 2 spans makes up the trace duration (almost)

nginx: proxy 13d66f3

Find... | Trace Timeline

rt January 19 2022, 14:31:10.959 | Duration 1.45s | Services 5 | Depth 6 | Total Spans 27

362.25ms | 724.51ms | 1.09s | 1.45s

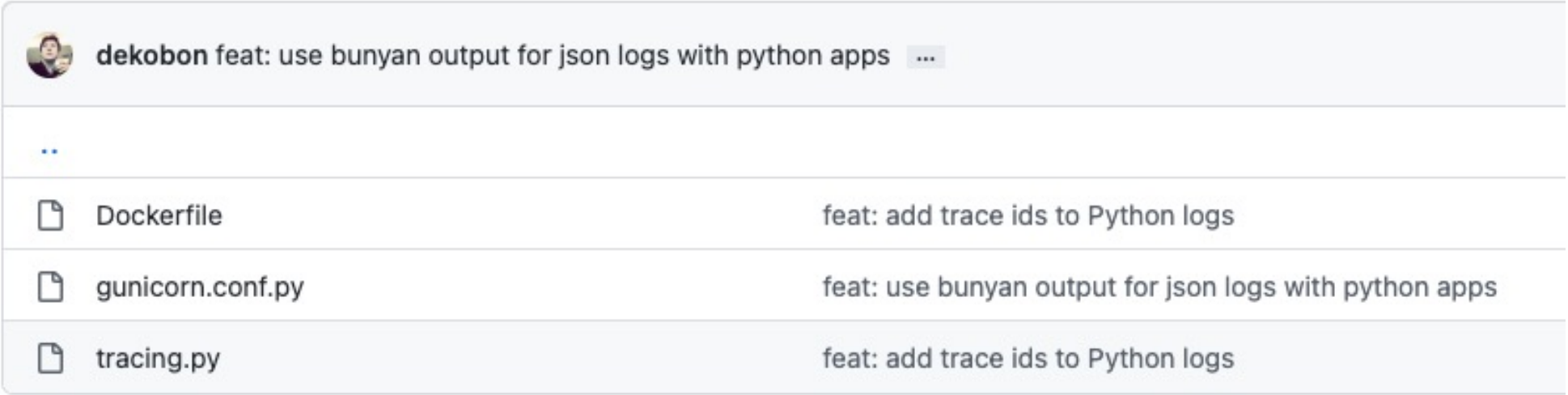
& Operation | 0µs | 362.25ms | 724.51ms | 1.09s | 1.45s

| Operation | Duration |
|---|---|
| proxy | |
| ontend /home | |
| frontend verify_token | 108.78ms |
| frontend read_json_response_from_remote[balance-reader] | 776.07ms |
| frontend HTTP GET | 775.22ms |
| balance-reader get_balance | 445.81ms |
| balance-reader jwt_verify | 144.42ms |
| balance-reader load_cache | 36.74ms |
| balance-reader → ledger --find_balance | 12.23ms |
| frontend json_deserialize | 83µs |
| frontend read_json_response_from_remote[transaction-history] | 452.11ms |
| frontend HTTP GET | 451.26ms |
| transaction-history get_transactions | 326.23ms |
| transaction-history jwt_verify | 61.43ms |
| transaction-history cache_lookup | 67.91ms |
| transaction-history load_cache | 51.27ms |
| transaction-history → ledger --find_fo… | 5.3ms |
| frontend json_deserialize | 183µs |
| frontend read_json_response_from_remote[contacts] | 83.8ms |
| frontend HTTP GET | 83.18ms |
| contacts /contacts/<username> | 78.89ms |
| contacts SELECT accounts | 1.66ms |
| frontend json_deserialize | 72µs |
| frontend populate_contact_labels | 244µs |
| frontend jinja2.load | 16.68ms |
| frontend jinja2.compile | 16.01ms |

# But it’s all about the language

## Python

- Pretty straightforward
  - Added 2 files
  - Updated requirements.txt to include dependencies



# And some were harder than others

## Java

- Straight Java / Greenfield = Not too bad
  - Import the libraries and use the APIs
- With Spring, life looked easy
  - Use Spring Cloud Sleuth
    - Adds trace/span IDs to Slf4j
    - Instruments common ingress and egress points
    - Adds traces to scheduled tasks
    - Can generate Zipkin traces
- But at that time:
  - Autoconfig was a milestone release, and supported old OTel versions
  - We needed to pull from Spring Snapshot due to coded dependency references

# Answer: a common telemetry module

- Extended tracing functionality
  - Provided Spring enabled autoconfiguration classes, adding more trace resource attributes
  - Built a NoOp implementation to let us disable tracing
  - Added a trace name interceptor to standardize our trace names
  - Added an error handler to output errors both to logs and traces
  - Enhanced the implementation of tracing attributes (service name, instance id, machine id, etc)
  - Built a tracing statement inspector to put trace ids into comments that precede SQL statements

We also extended the reach to Apache by creating a Spring compatible HTTP client

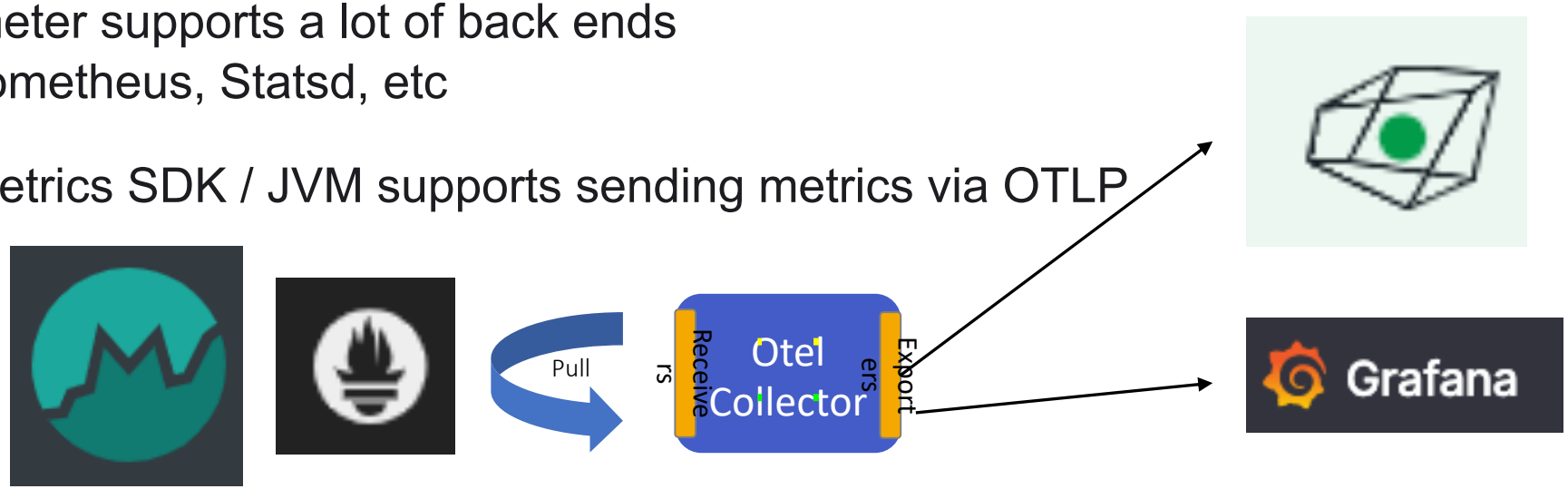
All this was integrated using the OpenTelemetry NGINX module (beta)
[nginxinc/nginx-unsupported-modules: Container builds of unsupported NGINX modules (github.com)]

# Counting Metrics

- We skipped Python
- Java required some study and decisions.
  - Original code used Micrometer/Stackdriver
- We found there were some significant limits to OTel for metrics in JVM
- Micrometer is a mature metrics layer for JVM
  - It is the default metrics API in Spring

# OpenTelemetry and Micrometer

- OTel Collector can use metrics from *anything*
  - Prometheus, Statsd, OTLP, etc
- Micrometer supports a lot of back ends
  - Prometheus, Statsd, etc
- OTel Metrics SDK / JVM supports sending metrics via OTLP



# Known Challenges

- Maturity of components
  - Some components still in beta
- Early in life cycle of other components
  - Instrumentation type
  - Manual versus Auto-instrumentation
- Operational concerns
  - Resource concerns (cpu / mem)
  - Performance impact
  - Long term data storage concerns
- Standards / Best Practices

# What’s Next

- Load and Performance testing
- Track and adopt OpenTelemetry enhancements (LOGS!)
  - Log data model is pretty settled
  - Stanza work within the Collector means can handle logs.
- Extend OpenTelemetry to other open source NGINX projects
- Test OpenTelemetry solution versus various analyzers

# TL;DR

Metrics, Traces and Logs

- All took different approaches to get what we wanted
- The Collector was our friend

Metrics and Traces had some interesting gotchas

This is a snapshot in time; things have changed

Auto-instrumentation is great

- When it works
- And gives you what you want

https://github.com/nginxinc/kic-reference-architectures

Try it for yourself

nginxinc/bank-of-sirius: Bank of Sirius (github.com)

# Questions?

# Thanks for attending

https://www.linkedin.com/in/davemc