F5 NGINX

Lessons Learned: Adding OpenTelemetry to a Modern App

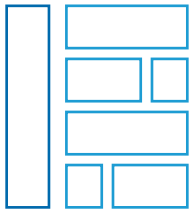
Dave McAllister

NGINX
Part of F5

Every company is on a cloud journey

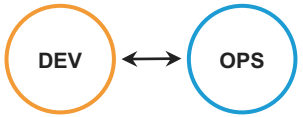
To increase velocity, agility and responsiveness

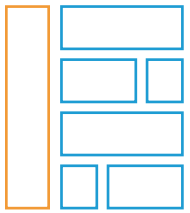
Retain & Optimize



Tightly Coupled Apps,
Slow Deployment Cycles

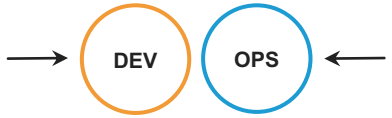
Lift & Shift



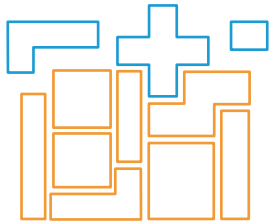


Primarily using
Cloud IaaS

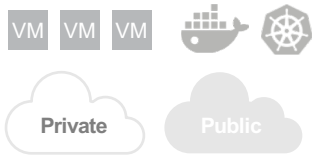
Re-Factor



Cloud Managed e.g. RDS,
DynamoDB, SaaS

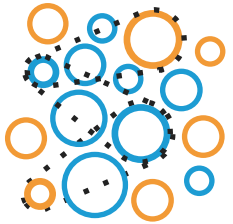


More Modular, but
Dependent App Components



Re-Architect / Cloud-Native

Cloud First Architecture



Loosely Coupled Microservices,
and Serverless Functions

"By 2025, 85% of organizations will run containers in production, up from less than 30% in 2020" – *Gartner, Dec 14, 2020*

Modern Apps

Modern Apps
Walks beside me
Modern Apps
Walks on by
Modern Apps

Are defined by their capabilities not their implementation.

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

No One Cares if it is all in COBOL - They Care if it Works

When your app can do this, it is modern

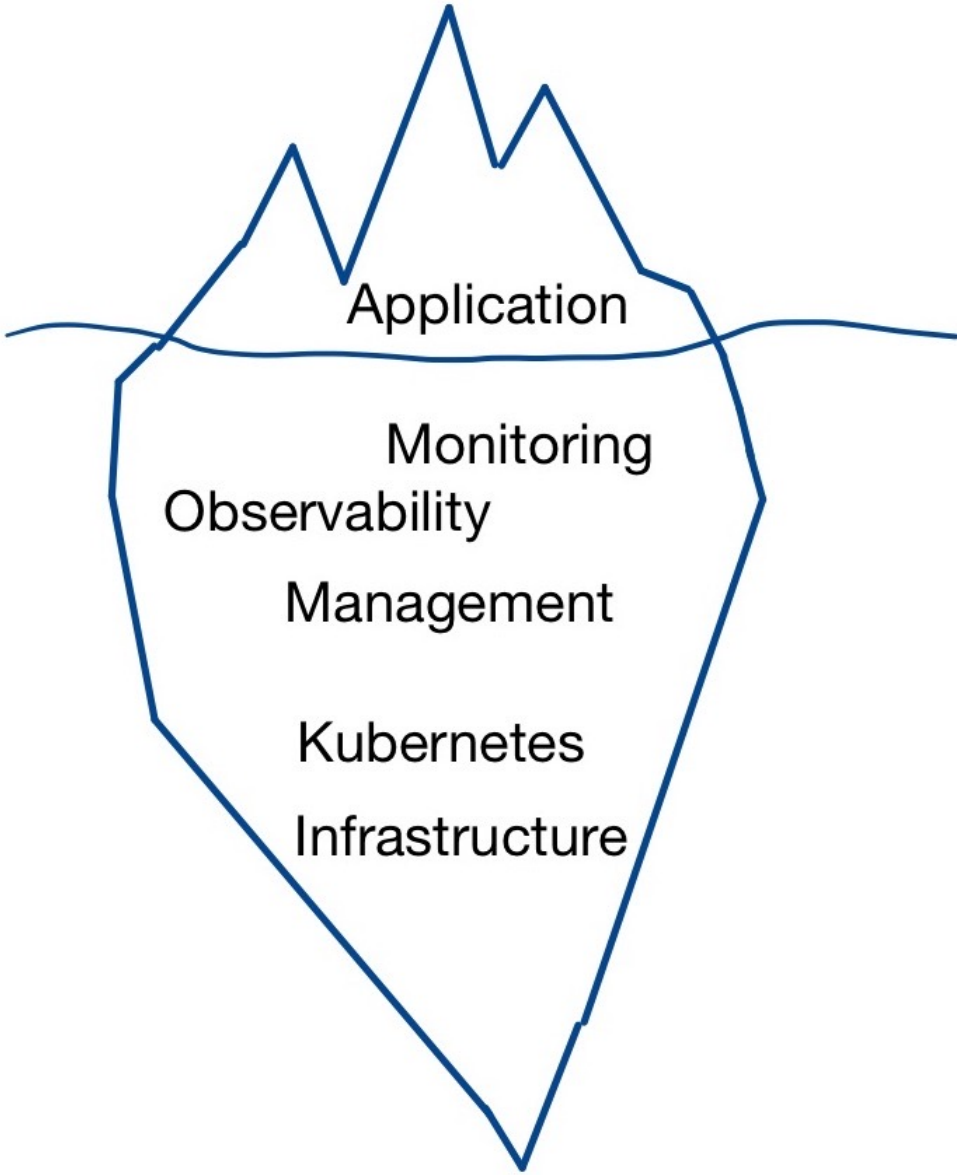
- Deploy in minutes
- **Quickly find performance bottlenecks**
- Easily change hosting provider
- Scale up/down quickly
- Gracefully degrade upon failure
- **Provide answers to platform engineers' questions**
- Protect itself from attacks
- Manage state in a knowable way
- **Provide context on errors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

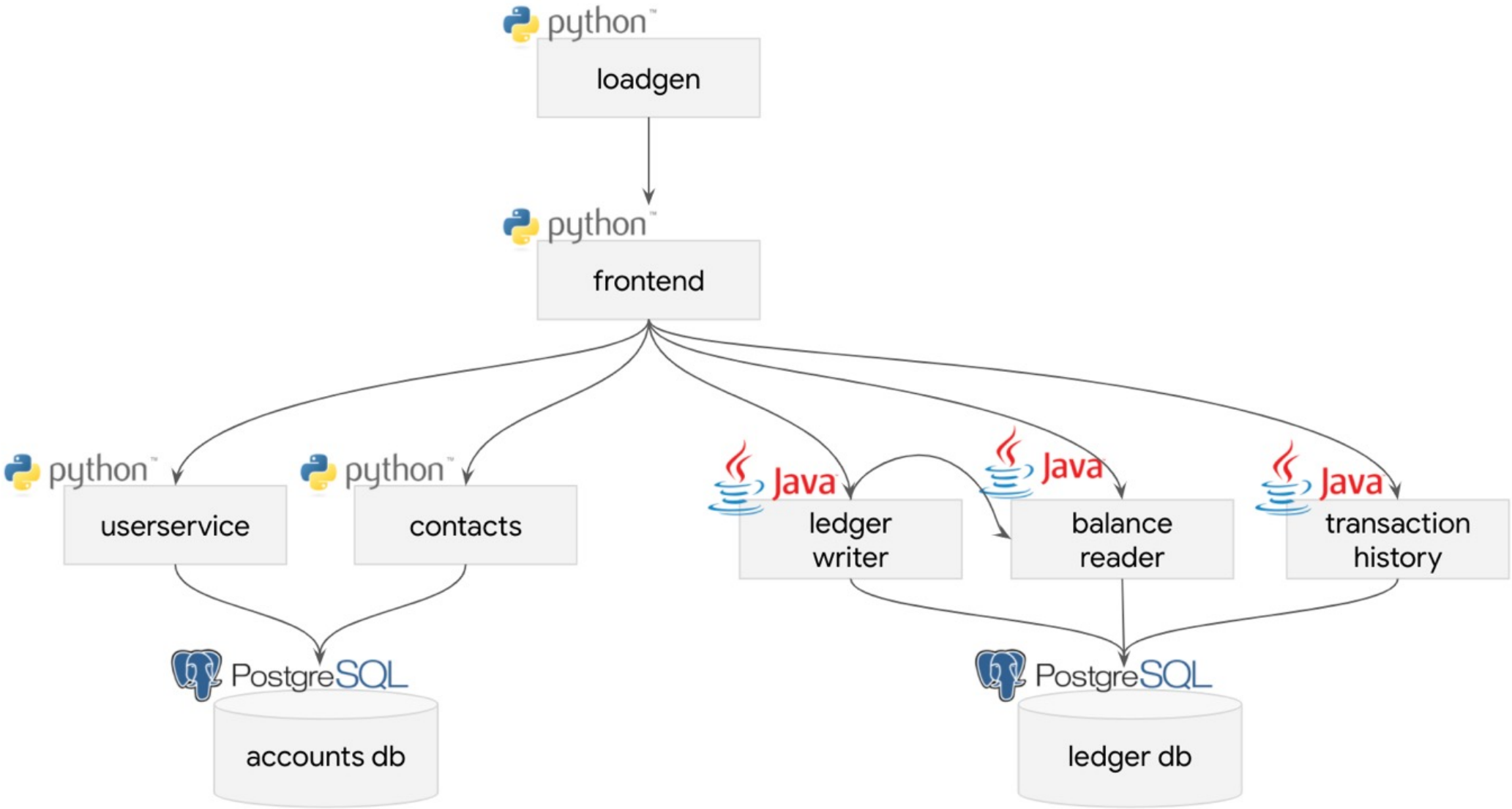
What is MARA?

An microservices architecture using Kubernetes that aims to be as production ready as possible.

Configured and integrated components deployed via Infrastructure as Code

The application is only one part of the modern application deployment

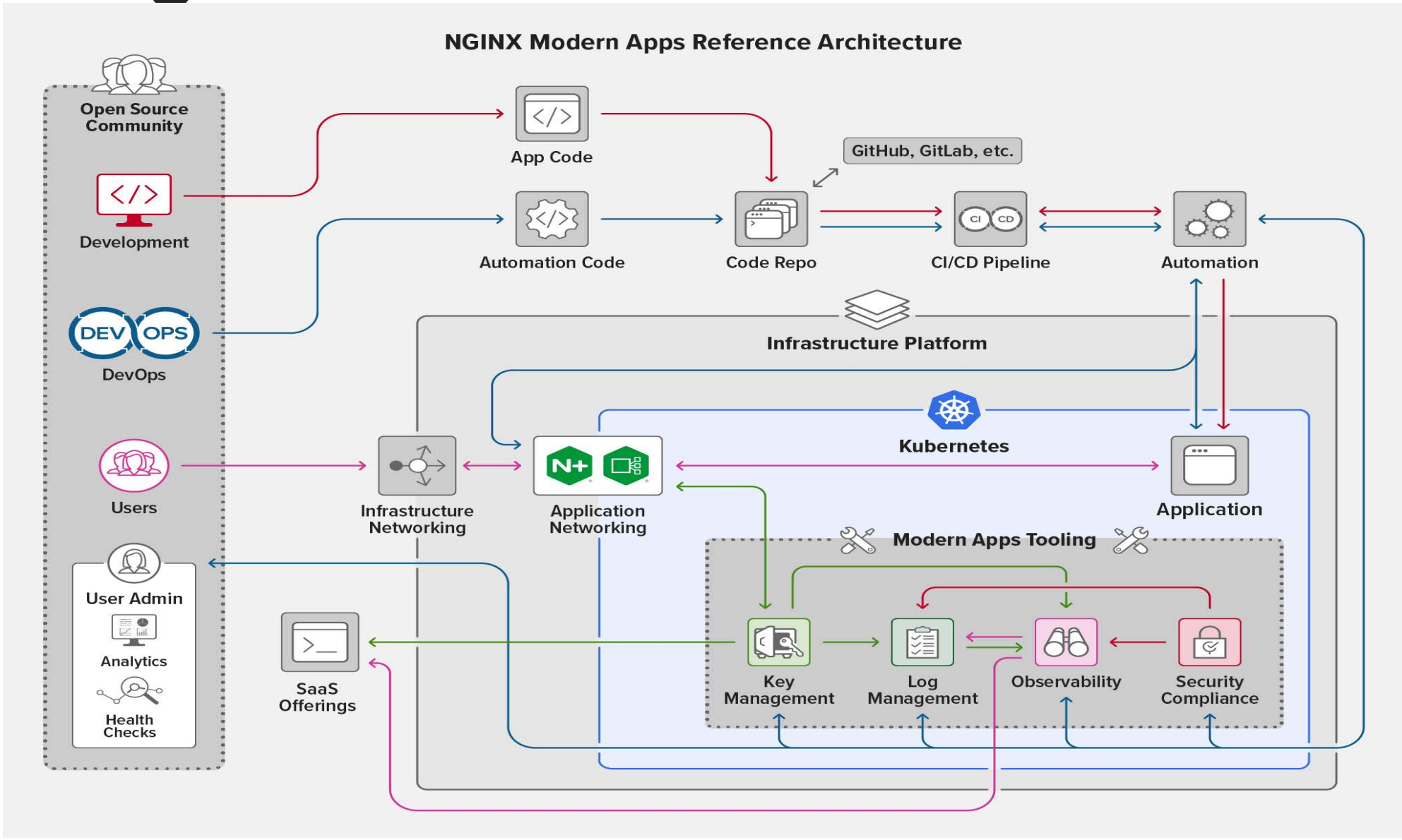



python
loadgen
python
frontend
python
userservice
python
contacts
Java
ledger
writer
Java
balance
reader
Java
transaction
history
PostgreSQL
accounts db
PostgreSQL
ledger db

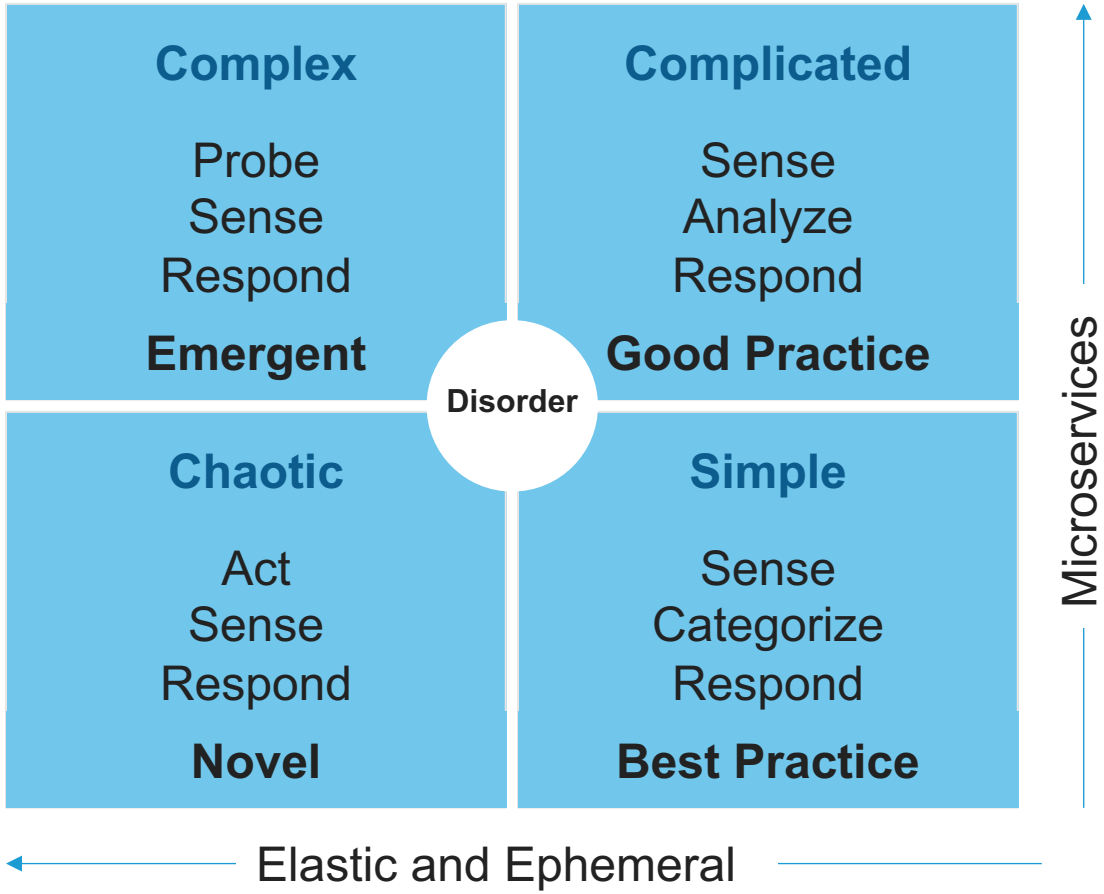
What's in the box?

- Infrastructure: Kubeconfig, AWS, Linode, Digital Ocean
- Observability:
  - Tracing: OpenTelemetry Operator
  - Metrics: Prometheus
  - Logging: Elasticsearch / Filebeat / Kibana
  - Visualization: Grafana
- Management:
  - Certificates: Cert-Manager
  - Secrets Management: Pulumi / Kubernetes
  - Ingress: NGINX Ingress Controller / NGINX Plus Ingress Controller
- Load Generation: Locust
- Application: Bank of Sirius
- Infrastructure as Code: Pulumi in Python
- Continuous Integration: Jenkins

Drawing it all out



Observability answers Challenges



Cynefin Framework

- Microservices create complex interactions.
- Failures don't exactly repeat.
- Debugging multi-tenancy is painful.
- Scale of data is massive
- Faster release cycles
- You own the code

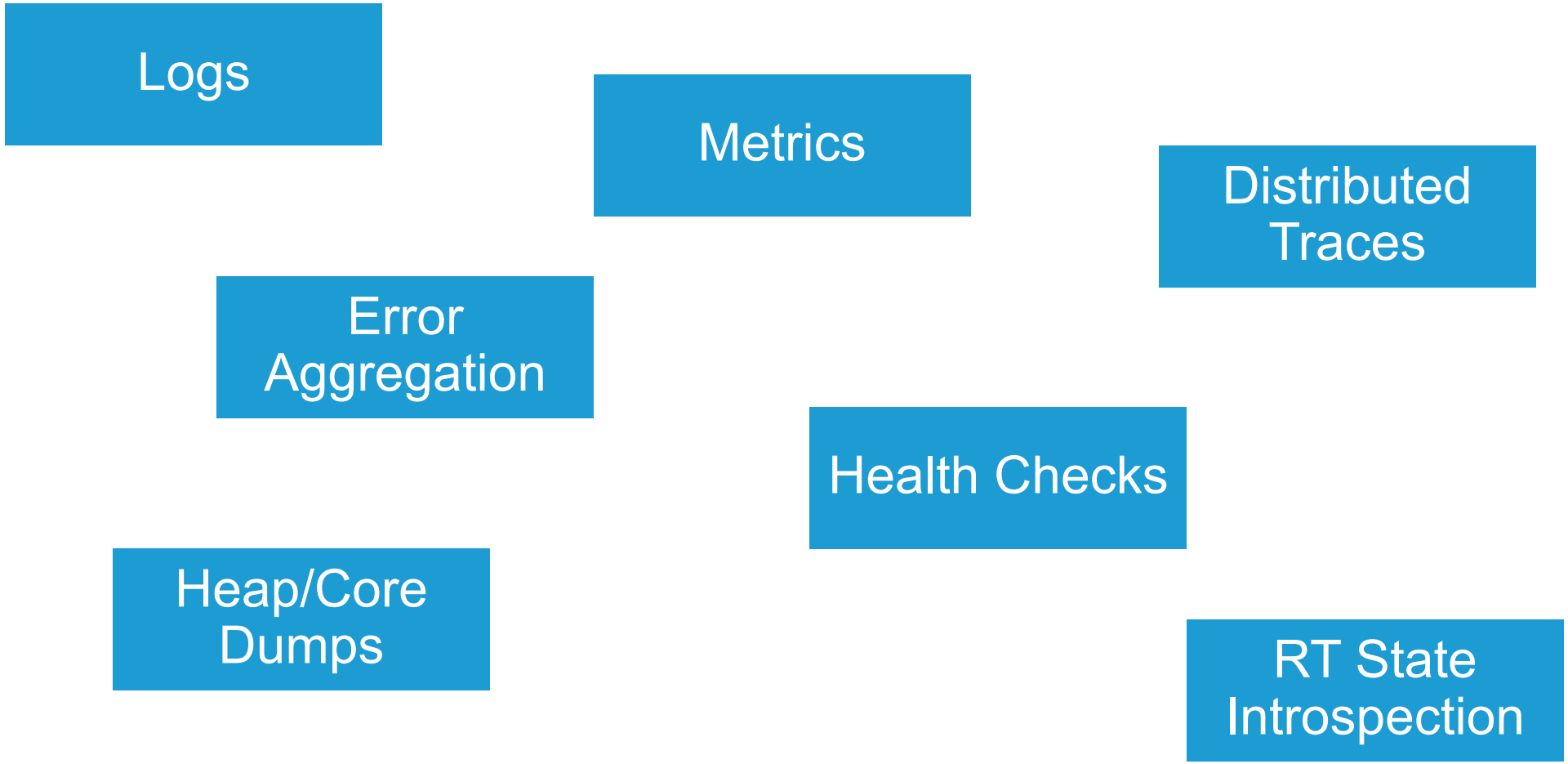
OBSERVABILITY IS A DATA PROBLEM

THE MORE OBSERVABLE A SYSTEM, THE QUICKER WE CAN UNDERSTAND WHY IT'S ACTING UP AND FIX IT

Full-Stack Visibility & Context-Rich Insights

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

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

OpenCensus and OpenTracing have merged into OpenTelemetry!

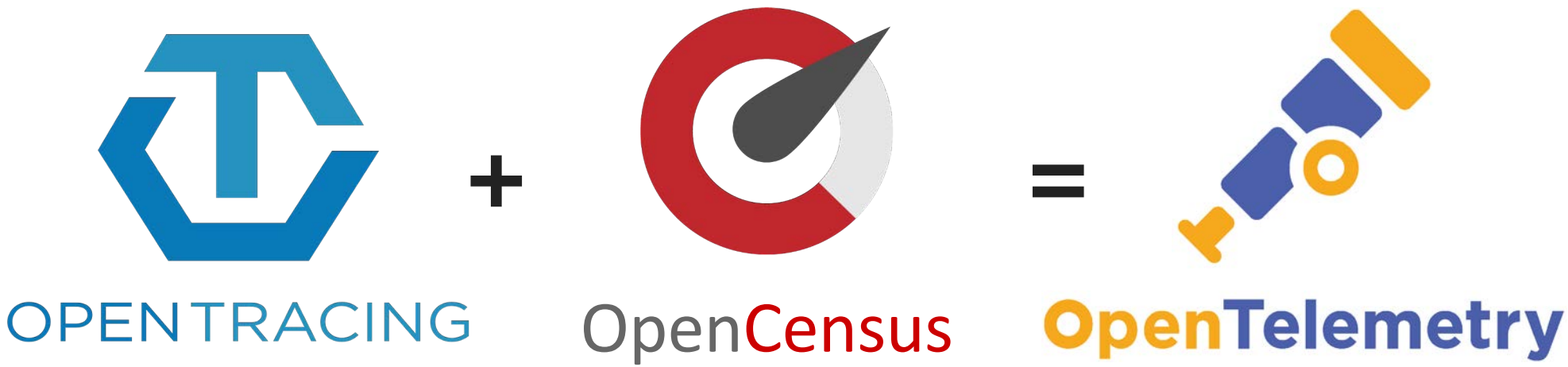
OpenCensus

Easily collect telemetry like metrics and distributed traces from your services

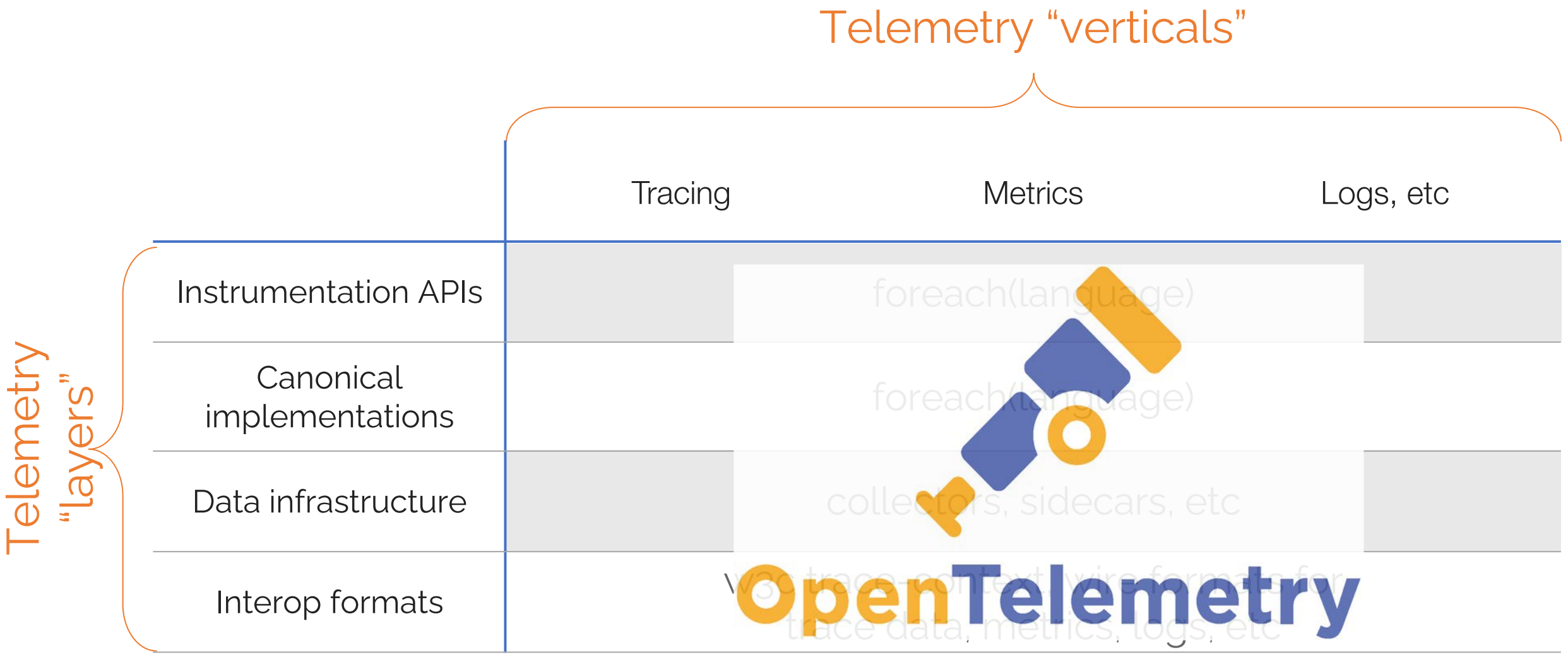
The OpenTracing project is *archived*. Learn more.
Migrate to OpenTelemetry today!

What is OpenTelemetry?

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



Cloud Native Telemetry



Let’s start with logs

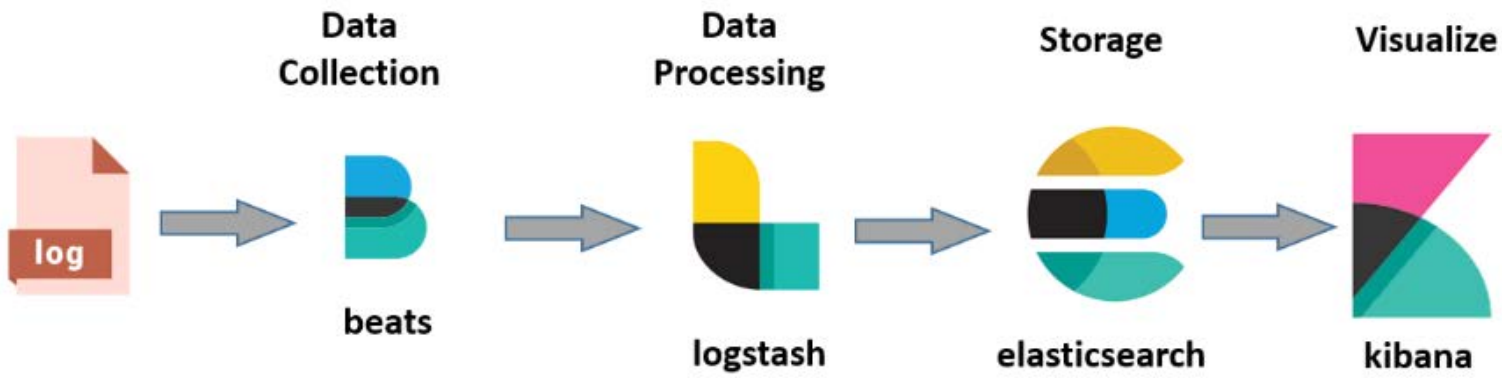
We all know what logs are but their simplicity rapidly leads to some complex decisions

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

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

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

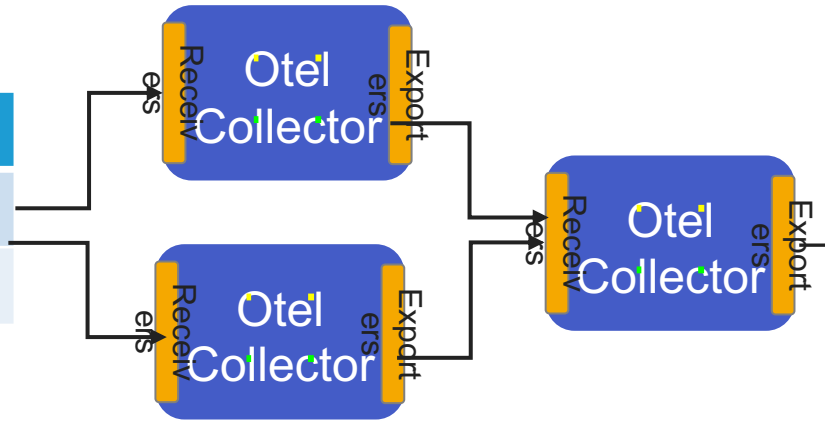
- Extremely resource hungry
- Query variance is okay

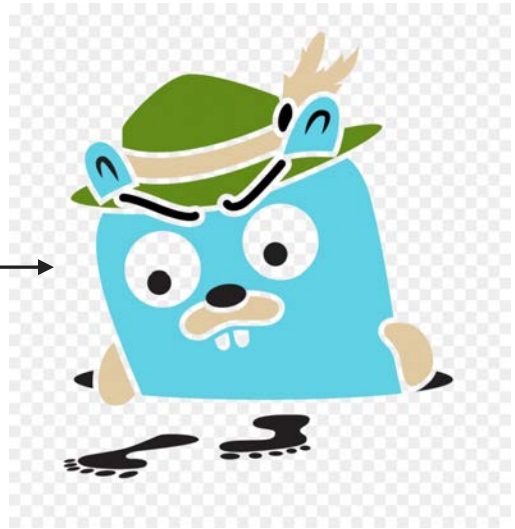
Distributed Tracing

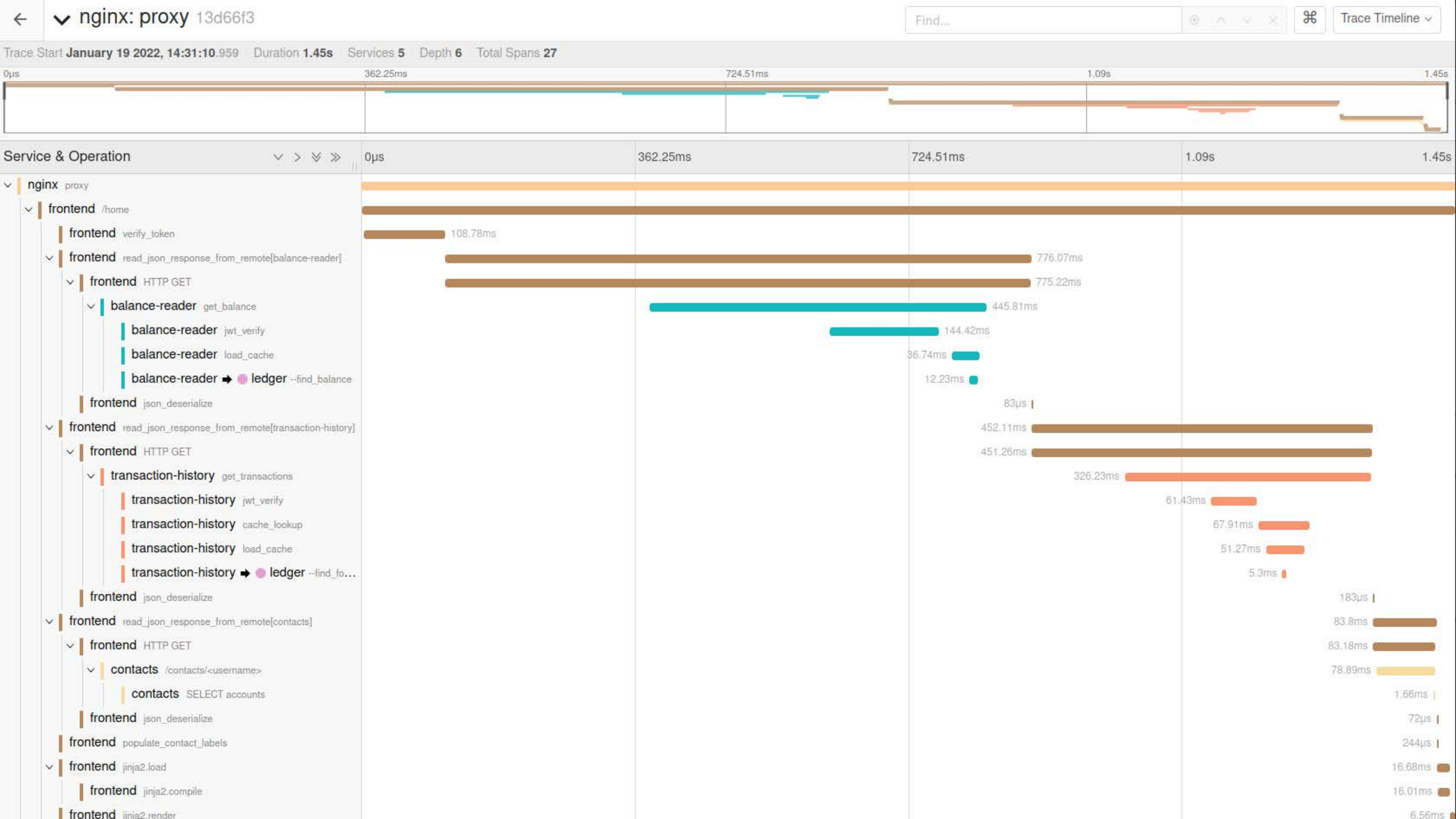
Complex and semi-chaotic

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

Language	Framework	Services
Java	Sprint Boot	3
Python	Flask	3



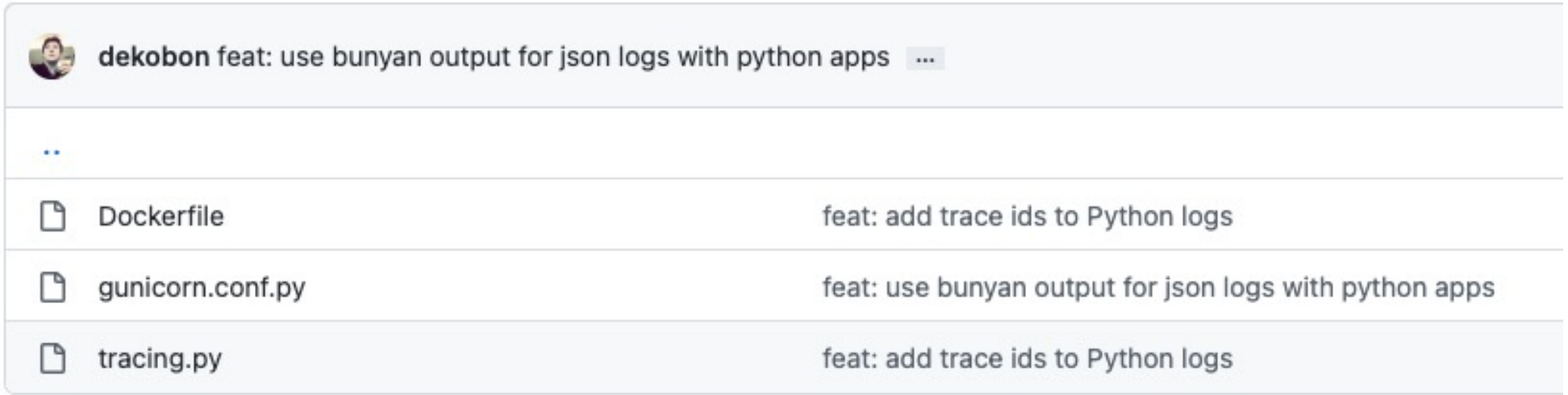



nginx: proxy 13d66f3
Find...
Trace Timeline
Trace Start January 19 2022, 14:31:10.959 Duration 1.45s Services 5 Depth 6 Total Spans 27
0µs
362.25ms
724.51ms
1.09s
1.45s
Service & Operation
nginx proxy
frontend /home
frontend verify_token
108.78ms
frontend read_json_response_from_remote[balance-reader]
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
frontend jinja2.render
6.56ms

But it's all about the language

Python

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



And some were easier 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

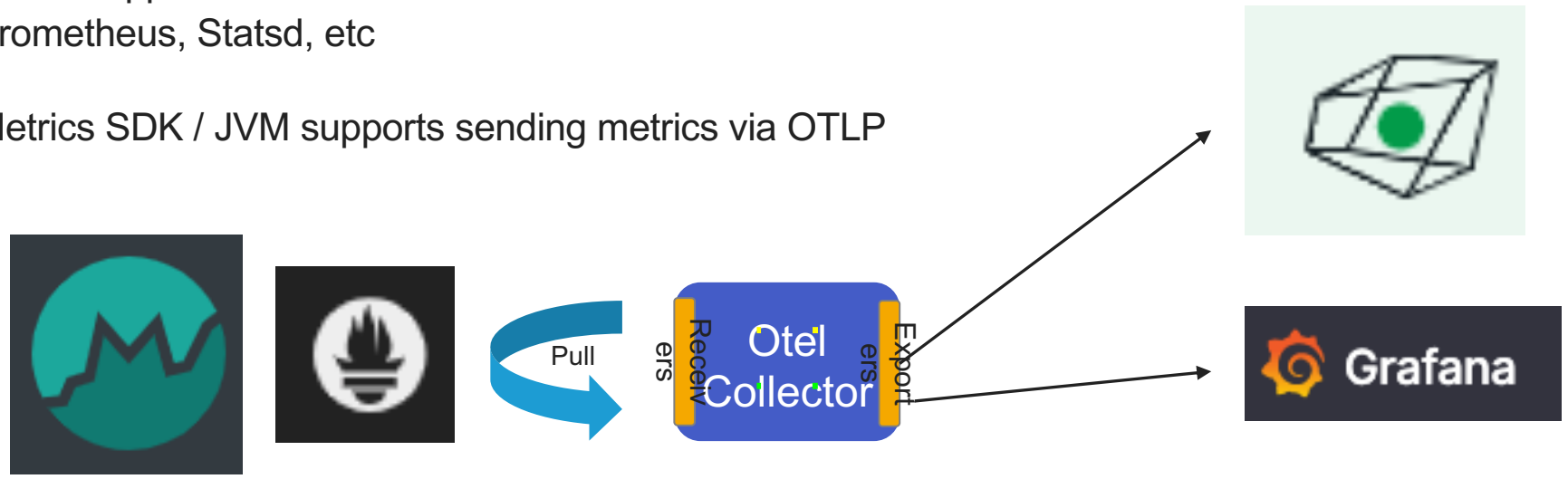
All this was integrated using the OpenTelemetry NGINX module (beta)
nginxinc/nginx-unsupported-modules: Container builds of unsupported NGINX modules (github.com)

Counting on Metrics

- We skipped Python
- Java required some study and decisions.
  - Original code (Bank of Anthos) 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



Summary

Otel FTW

Distributed Tracing

- Java: Spring Cloud Sleuth → Spring Cloud Sleuth OTEL exporter → OTEL Collector → Pluggable Store
- Python: OTEL Python libraries → OTEL Collector → Pluggable Store
- NGINX (Ingress Controller is not supported yet): NGINX OTEL module → OTEL Collector → Pluggable Store

Metrics

- Java: Micrometer via Spring → Prometheus exporter → OTEL Collector
- Python WSGI: GUnicorn statsd → Prometheus (via statsd / ServiceMonitor)
- Python: No implementation yet
- NGINX: Prometheus Endpoint → Prometheus (via ServiceMonitor)

Logs

- All Container Logs: Filebeat → Elasticsearch / Kibana

Summary

Error Aggregation

- OTEL distributed traces → Pluggable Store search

Health Checks

- Java: Spring Boot Actuator → Kubernetes
- Python: Flask Management Endpoints Python module → Kubernetes

Runtime Introspection

- Java: Spring Boot Actuator
- Python: Flask Management Endpoints Python module

Heap/Core Dumps

- Java: Spring Boot Actuator support for thread dumps
- Python: no support yet

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-configuration 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)

F5 NGINX

Questions

Join our Open Source Community slack!

NGINX
Part of F5