Micrometer

Spring Boot 3 Workshop - Devnexus 2023 - Atlanta

Jonatan Ivanov & Phil Webb

What is Observability?

- 3 pillars: Logging, Metrics, Distributed Tracing
- 4 pillars: + Events/Lineage(?)/Context/Metadata
- 6 pillars: + Profiles + Exceptions Arbitrary Wide Events, Signals
- But what about:
  - /health, /info, etc.
  - Service Registry/Discoverability
  - API Discoverability

What is Observability?

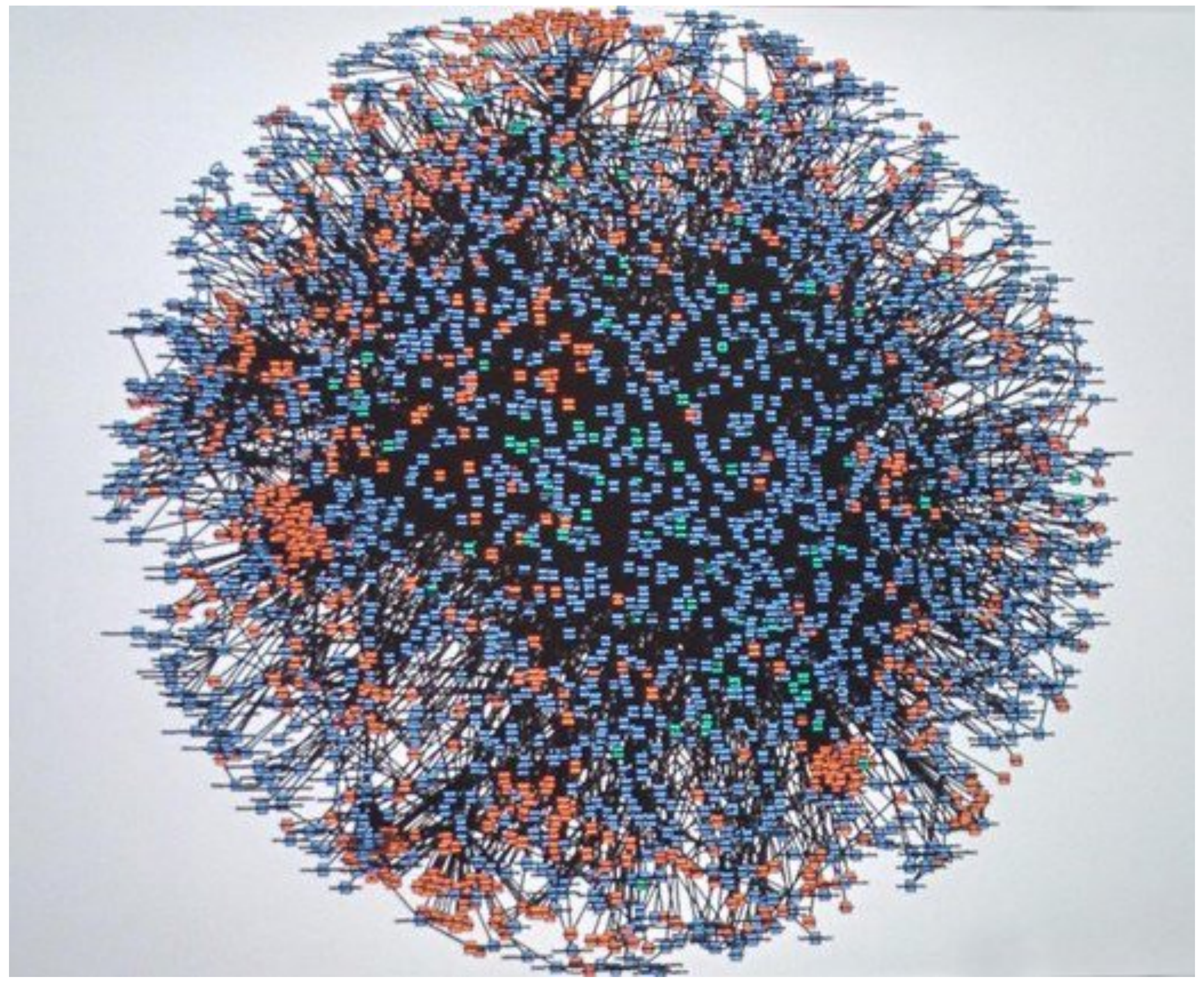
How well we can understand the internals of a system based on its outputs

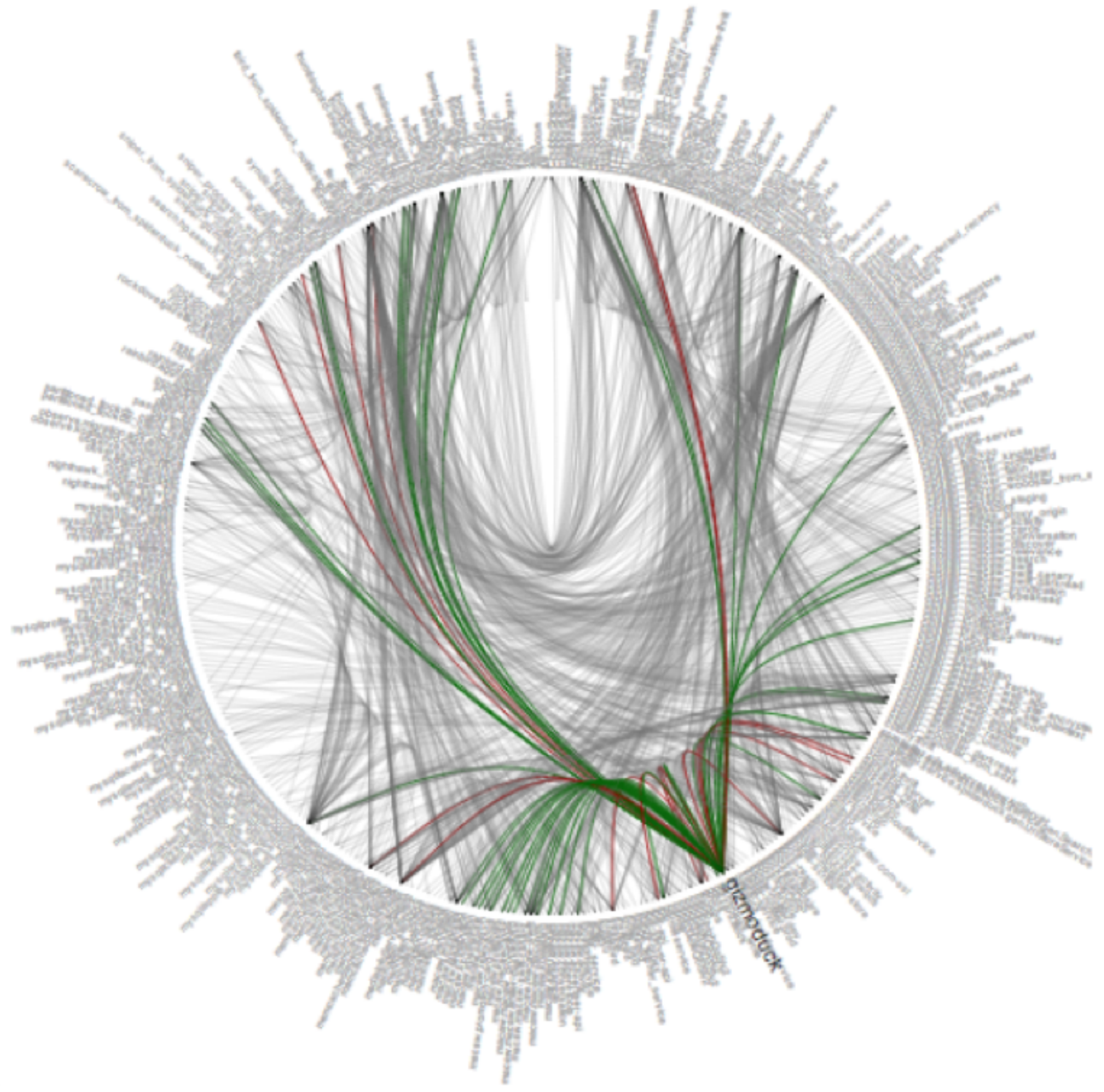
*(Providing **meaningful** information about what happens inside)*

Why do we need Observability?

Today's systems are insanely complex (cloud)

(Death Star Architecture, Big Ball of Mud)





Why do we need Observability?

Environments can be chaotic
(You turn a knob here a little and services are going down there)

We need to deal with unknown unknowns
(We can’t know everything)

Things can be perceived differently by observers
(Everything is broken for the users but seems ok to you)

Logging - Metrics - Distributed Tracing

Logging

What happened (why)?
Emitting events

Metrics

What is the context?
Aggregating data

Distributed Tracing

Why happened?
Recording causal ordering of events

Examples

Latency

Logging

Processing took 140ms

Metrics

P99.999: 140ms
Max: 150 ms

Distributed Tracing

DB was slow (lot of data was requested)

Examples

Error

Logging
Processing failed (stacktrace?)

Metrics
The error rate is 0.001/sec
2 errors in the last 30 minutes

Distributed Tracing
DB call failed (invalid input)

Checkpoint

Everyone knows that Observability != "Three Pillars" 🙂

Logging with Spring

SLF4J + Logback

- SLF4J with Logback comes pre-configured
- SLF4J: Simple Logging Façade for Java
  - Simple API for logging libraries
- Logback Natively implements the SLF4J API
- If you want Log4j2 instead of Logback:

```
- spring-boot-starter-logging
+ spring-boot-starter-log4j2
```

Setup Logging

Add org property

- We will need something that we can use to query:
  - All of our apps (**`spring.application.org`**)
  - Only one app (**`spring.application.name`**)
  - Only one instance (we only have one instance per app)

```
spring:
  application:
    name: dog-service
    org: petclinic
```

Setup Logging

Add Loki4J

- Copy
  From: **`dog-client/src/main/resources/logback-spring.xml`**
  To: **`dog-service/src/main/resources/logback-spring.xml`**

- Add dependency to **`pom.xml`**

```
<dependency>
  <groupId>com.github.loki4j</groupId>
  <artifactId>loki-logback-appender</artifactId>
  <version>1.4.0</version>
<dependency>
```

Setup Logging

Do we have logs?

- Got to Grafana: `http://localhost:3000`
- Choose Explore, then Loki from the drop down
- Search for `application = dog-service`
- Search for `org = petclinic`
- We will get back to our logs later

Checkpoint

Everyone has logs in Loki for both services

Metrics with Spring

Micrometer

- Popular Metrics library on the JVM
- Like SLF4J, but for metrics
- Simple API
- Supports the most popular metric backends
- Comes with **`spring-boot-actuator`**
- Spring projects are instrumented using Micrometer
- A lot of third-party libraries use Micrometer

Metrics with Spring

Like SLF4J, but for metrics …

AppOptics

Atlas

Azure Monitor

CloudWatch (AWS)

Datadog

Dynatrace

Elastic

Ganglia

Graphite

Humio

InfluxDB

JMX

KairosDB

New Relic

(/actuator/metrics)

OpenTSDB

OTLP

Prometheus

SignalFx

Stackdriver (GCP)

StatsD

Wavefront (VMware)

Sidetrack: Observation API

You want to instrument your application…

- Add logs (application logs)
- Add metrics
  - Start/Stop Timers, Increment Counters
- Add Distributed Tracing
  - Start/Stop Spans
- Log Correlation
- Context Propagation

Observation API (Micrometer 1.10)

```java
Observation observation = Observation.start("talk",registry);

try { // TODO: scope
 Thread.sleep(1000);
}
catch (Exception exception) {
 observation.error(exception);
 throw exception;
}
finally { // TODO: attach tags (key-value)
 observation.stop();
}
```

Observation API (Micrometer 1.10)

```
ObservationRegistry registry = ObservationRegistry.create();

registry.observationConfig()
     .observationHandler(new MeterHandler(...))
     .observationHandler(new TracingHandler(...))
     .observationHandler(new LoggingHandler(...))
     .observationHandler(new AuditEventHandler(...));


Observation observation = Observation.start("talk",registry)
     // let the fun begin…
observation.stop();
```

Observation API (Micrometer 1.10)

```
Observation.createNotStarted("talk",registry)
  .lowCardinalityKeyValue("conference", "DN")
  .highCardinalityKeyValue("uid", userId)
 .observe(this::talk);


@Observed
```

Setup Metrics

Add the org to Observations

```
…actuator.ActuatorConfiguration.java

@Bean
ObservationFilter orgFilter(
    @Value("${spring.application.org}") String org) {
    return context -> context.addLowCardinalityKeyValue(
        KeyValue.of("org", org)
    );
}
```

Setup Metrics

Let’s check Metrics

- Go to **`http://localhost:8080/actuator/prometheus`**
- 401 🧐
- So Prometheus is broken too? **`http://localhost:9090/targets`**
- Spring Security! 👀
- Let’s disable it, what could go wrong!? 😈
- Everything, please don’t do this in prod!
  Except you want everyone know about it. 😈

Setup Metrics

Disable auth for certain endpoints

```
SecurityConfiguration.java
```

```
requests
  .requestMatchers("/dogs", "/actuator/**").permitAll();
```

Setup Metrics

Add org and application tags to every meter

- Why only a few Prometheus time series has the org tag?
- Not everything is created through the Observation API e.g.: heap utilization
- Let’s add tags to everything!

```
management:
  metrics:
    tags:
      application: ${spring.application.name}
      org: ${spring.application.org}
```

Setup Metrics

Remove Observation name customization

- We are going to depend on default behavior
- So let's remove the custom http observation renaming
- Remove/comment out

```
#  observations:
#    http:
#      server:
#        requests:
#          name: "http.server.in"
```

Setup Metrics

Add histogram support for http metrics

- We want to see the latency distributions on our dashboards
- We want to calculate percentiles (P99?)

```
management:
  metrics:
    distribution:
      percentiles-histogram:
        # all: true
        http.server.requests: true
```

Setup Metrics

Let’s check the HTTP and JVM metrics

- Let’s check **`/actuator/metrics`**
  `/actuator/metrics/{metricName}`
  `/actuator/metrics/{metricName}?tag=key:value`
- Let’s write a Prometheus query (**`HELP.md`**)

  ```
  sum by (application)
  (rate(http_server_requests_seconds_count[5m]))
  ```
- Let’s check the dashboards: go to Grafana, then Browse
  - Spring Boot Statistics
  - Dogs

Checkpoint

Everyone has metrics on the dashboards

Distributed Tracing with Spring

Micrometer Tracing and Spring Cloud Sleuth

- **Boot 2.x** - Spring Cloud Sleuth
- **Boot 3.x** - Micrometer Tracing (Sleuth w/o Spring dependencies)
- Provide an abstraction layer on top of tracing libraries
  - Brave (OpenZipkin), default
  - OpenTelemetry (CNCF), experimental
- Instrumentation for Spring Projects, third-party libraries, your app
- Support for various backends

Setup Distributed Tracing

Add Micrometer Tracing Dependencies

```
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-tracing-bridge-brave</a
</dependency>
<dependency>
  <groupId>io.zipkin.reporter2</groupId>
  <artifactId>zipkin-reporter-brave</artifactId>
</dependency>
```

Setup Distributed Tracing

Set sampling probability

```
management:
  tracing:
    sampling:
      probability: 1.0
```

Setup Distributed Tracing

Setup log correlation

Copy this from dog-client’s **`application.yml`**

```
logging:
  pattern:
    level: "%5p [${spring.application.name:},%X{traceId:-},%X{spanId:-}]"
  level: org.springframework.web.servlet.DispatcherServlet: DEBUG
```

Setup Distributed Tracing

Let's look at correlated logs

```
[
  dog-service,
  641e50ff9c6911eb4d96f9d4a19ebc82,
  c3e2ea2a9b4a3f60
]
```

```
[app-name,traceId,spanId]
```

Setup Distributed Tracing

Let’s look at some traces

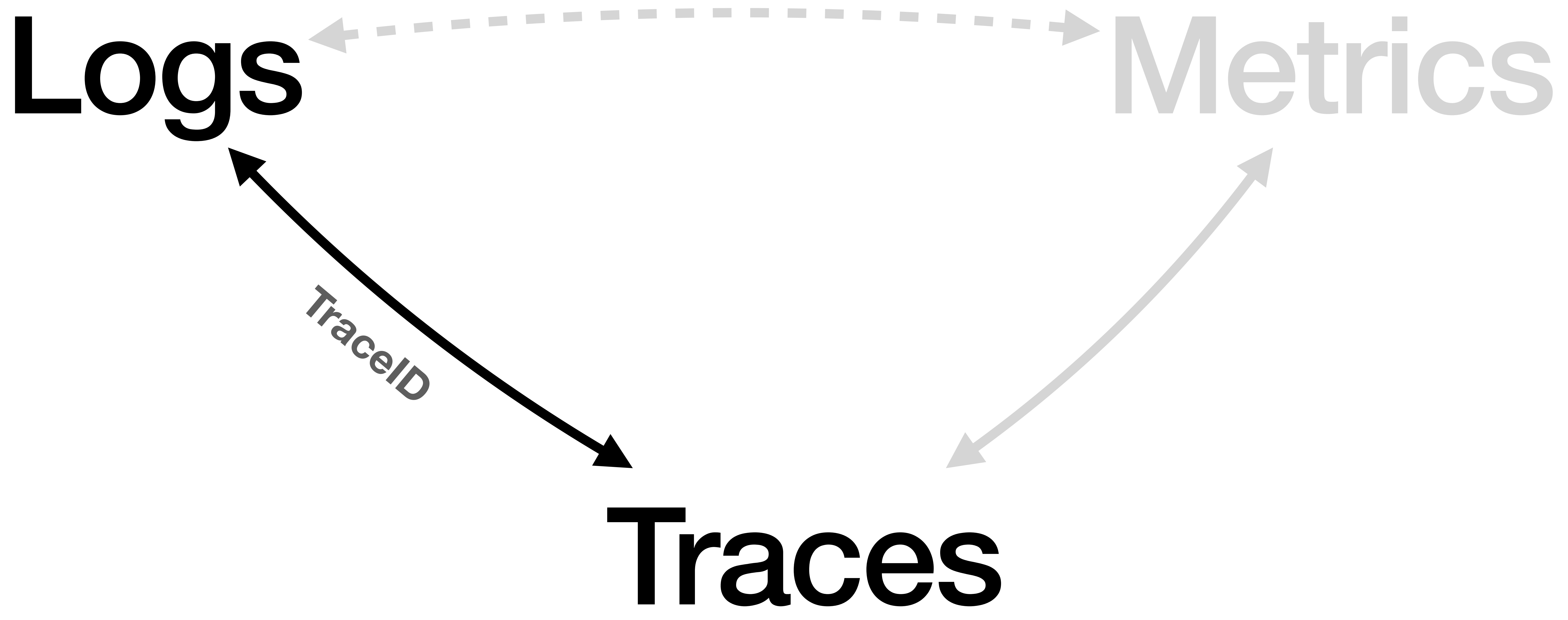
- Go to Grafana, then Explore and choose Tempo
- Terminology
  - Span
  - Trace
  - Tags
  - Annotations

Checkpoint

Everyone has log correlation and traces in Tempo

Interoperability

Logs -> Trace / Trace -> Log



Setup Observations

Disable Spring Security Observations

```
ActuatorConfiguration.java

@Bean
ObservationPredicate noSpringSecurityObservations() {
 return (name,ctx)-> !
name.startsWith("spring.security.");
}
```

Setup Observations

Disable Actuator Observations

`ActuatorConfiguration.java`

```
if (name.equals("http.server.requests") &&
  ctx instanceof ServerRequestObservationContext sc) {
    return !sc.getCarrier()
      .getRequestURI()
      .startsWith("/actuator");
}
```

Setup Observations

Enable JDBC Observations

- Tadaya Tsuyukubo 😎
- `net.ttddyy.observation:datasource-micrometer-spring-boot` (1.0.1)

```
jdbc:
  datasource-proxy:
    include-parameter-values: true
    query:
      enable-logging: true
      log-level: INFO
```

Setup Observations

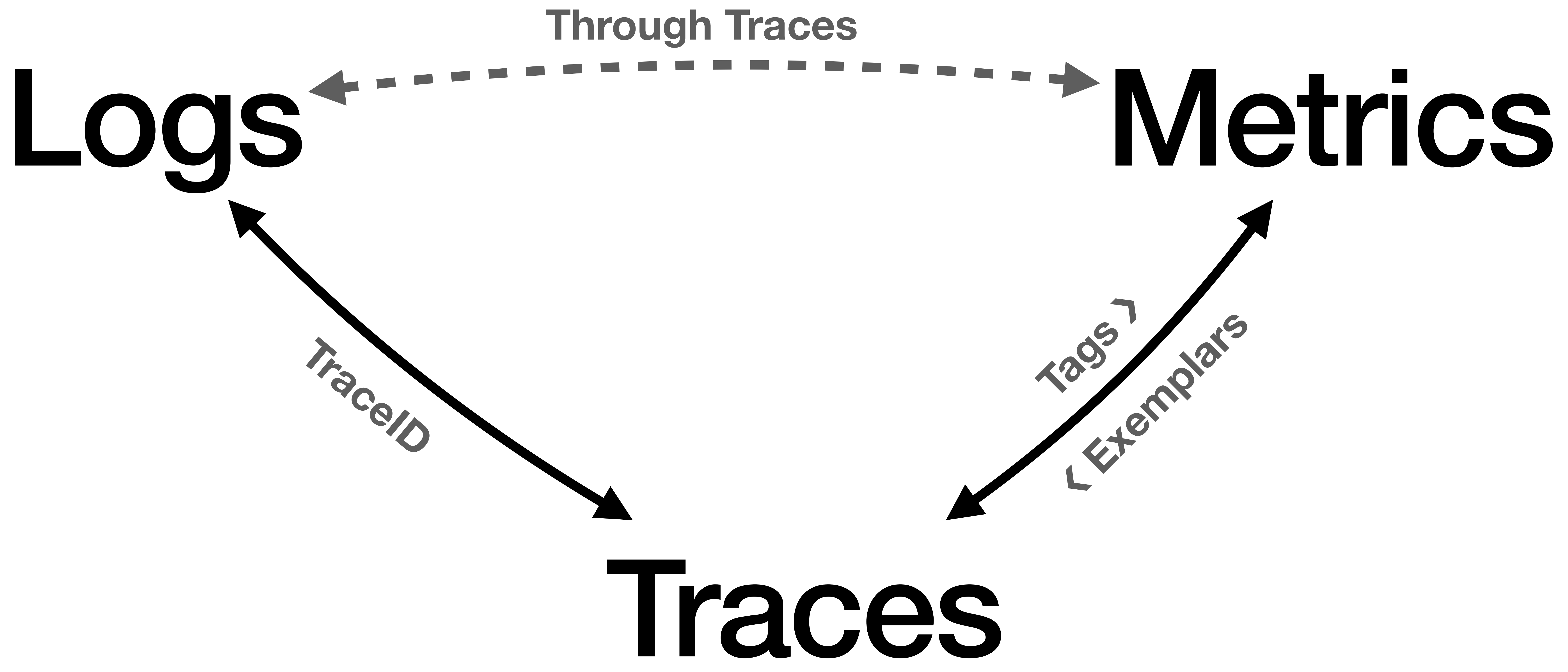
Add custom Observation

```
OwnerService.java
```

```
Observation
   .createNotStarted("getDogs", registry)
   .contextualName("gettingOwnedDogs")
   .highCardinalityKeyValue("owner", owner)
   .observe(() -> {
     //…
   });
```

Interoperability

Logs ⇔ Traces ⇔ Metrics



Interoperability

How to check Exemplars

- Exemplars are only available if you request the OpenMetrics format
- Your browser does not do this

```
http :8081/actuator/prometheus
'Accept: application/openmetrics-text;version=1.0.0'


(| grep trace_id)
```

Checkpoint

Logs <-> Metrics <-> Traces

Setup Observations

Log error and signal it

```
OwnerController.java

ProblemDetail onNoSuchDogOwner(
  HttpServletRequest request,
  NoSuchDogOwnerException ex) {

  logger.error("Ooops!", ex);
  ServerHttpObservationFilter
    .findObservationContext(request)
    .ifPresent(context -> context.setError(ex));
```

Setup Observations

Hack error reporting for Tempo

```
ObservationFilter tempoErrorFilter() {
  return context -> {
    if (context.getError() != null) {
      context.addHighCardinalityKeyValue(
        KeyValue.of("error", "true")
      );
      context.addHighCardinalityKeyValue(
        KeyValue.of(
          "errorMessage", context.getError().getMessage())
      );
    }
    return context;
  };
}
```

Setup Observations

Hack DB tags for Tempo ServiceGraph

```
if(ctx instanceof DataSourceBaseContext dsCtx){
  ctx.addHighCardinalityKeyValue(
    KeyValue.of(
      "db.name", dsCtx.getRemoteServiceName()
    )
  );
}
```

Actuator!

Add Java and OS InfoContributors

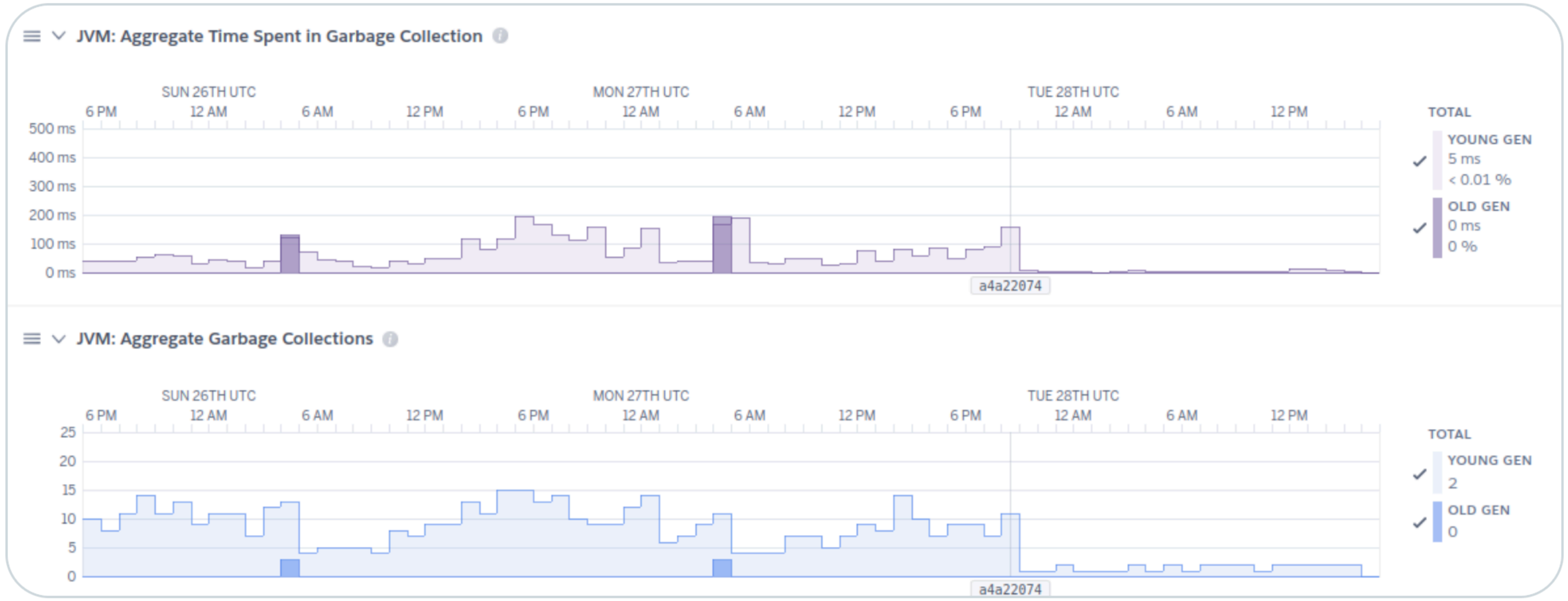
```
management:
  info:
    java:
      enabled: true
    os:
      enabled: true
```

Checkpoint

Everything works! 😎

James Ward @_JamesWard

Upgraded another service to JDK 17 and holy crap these out-of-the-box improvements in GC are 🤯

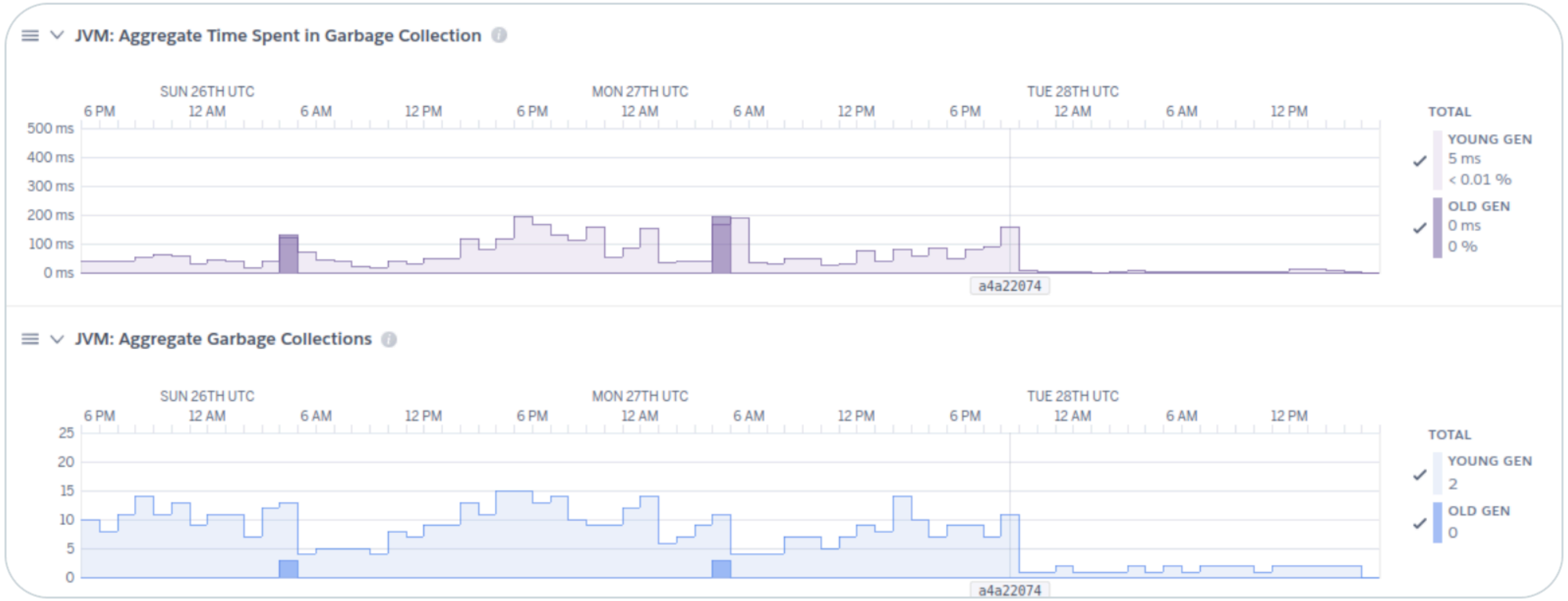


9:29 AM · Mar 28, 2023 · **82.3K** Views

89 Retweets **7** Quotes **612** Likes **52** Bookmarks

James Ward @_JamesWard

Upgraded another service to JDK 17 and holy crap these out-of-the-box improvements in GC are 🤯



9:29 AM · Mar 28, 2023 · **82.3K** Views

89 Retweets **7** Quotes **612** Likes **52** Bookmarks

Jonatan Ivanov @jonatan_ivanov · Mar 28

Replying to @_JamesWard and @spencerbgibb

Is it G1-G1? Or what is the GC before/after?

Extra

Inject Latency

```
docker exec toxiproxy
  /toxiproxy-cli toxic add
  --toxicName base-latency
  --type latency
  --downstream
  --toxicity 1.0
  --attribute latency=50
  --attribute jitter=0
  dog-db
```

Extra

- `avg` and `p99`
- The memory leak that only existed in prod
- The memory leak that was not a memory leak
- The zombie bot
- The hidden service from the past
- Auth failure (clock skew)
- Trade offs of Java Agents (for instrumentation)
- High Cardinality - Friend or Foe?
- How Not to Measure Elapsed Time

Extra

JVM ergonomics vs. Containers

- cgroups + namespaces
- Memory rq/limit: heap size
- CPU rq
    - Number of GC threads and GC algorithm
    - Number of runtime compiler threads (JIT)
    - Common Pool size (ForkJoinPool, Parallel Streams)
    - 3rd party thread Pools: `Runtime#availableProcessors` (Little's law)

Extra

TeaHouse

- `actuator/health` and `actuator/info` (TeaHouse)
- Service Registry/Discoverability (Eureka, Spring Boot Admin)
- API Discoverability (Swagger, HATEOAS)
- Access logs, Logbook, GC logs
- FlyWay

Extra

`actuator/health` and `actuator/info` (TeaHouse)

- Wrong version deployed
- Right version is deployed but it was not build against the commit you thought
- Wrong environment
- Wrong timezone or local
- Wrong certificate/cert-chain
- Unpatched OS
- The memory leak that only existed in prod

Checkpoint

Observability done!