Application Observability

like you've never heard before

2025-02-27

Jonatan Ivanov

About Me

- Spring Team
  - Micrometer
  - Spring Cloud, Spring Boot
  - Spring Observability Team
- Java Champion
- Seattle Java User Group
- develotters.com
- **@jonatan_ivanov**

Observability basics recap

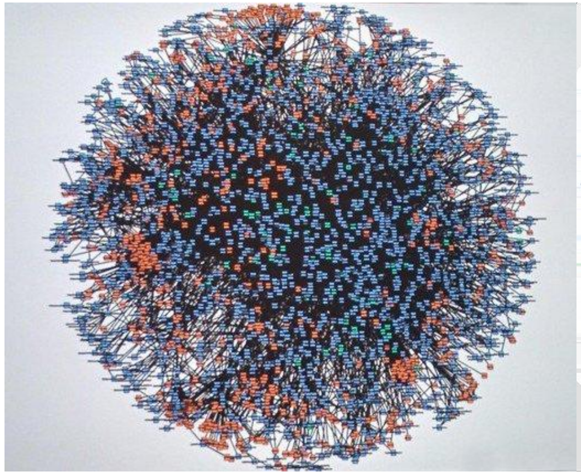
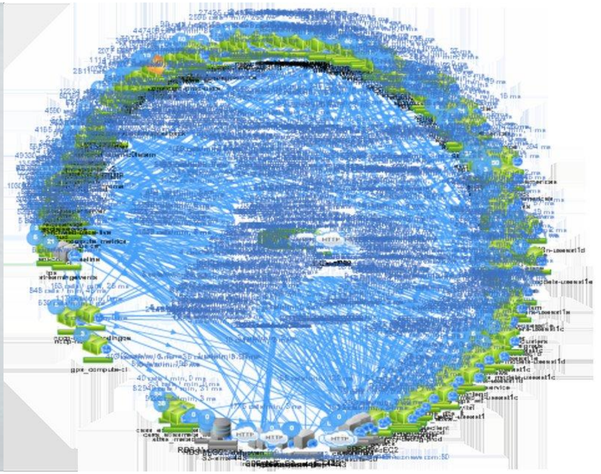
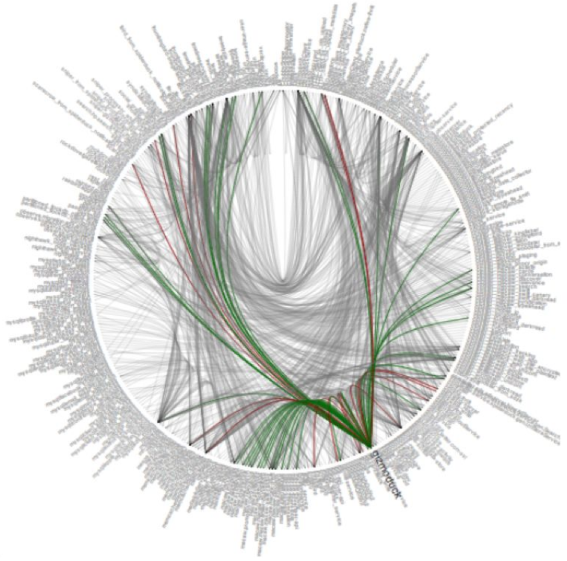
What is Observability?

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

(Providing ***meaningful*** information about what happens inside)
(Data about your app)

Why do we need Observability?

Today's systems are increasingly 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 apps 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 with JVM/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
```

Metrics with JVM/Spring: Micrometer

Dimensional Metrics library on the JVM

Like SLF4J, but for metrics

API is independent of the configured metrics backend

Supports many backends

Comes with `spring-boot-actuator`

Spring projects are instrumented using Micrometer

Many third-party libraries use Micrometer

Distributed Tracing with JVM/Spring

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”

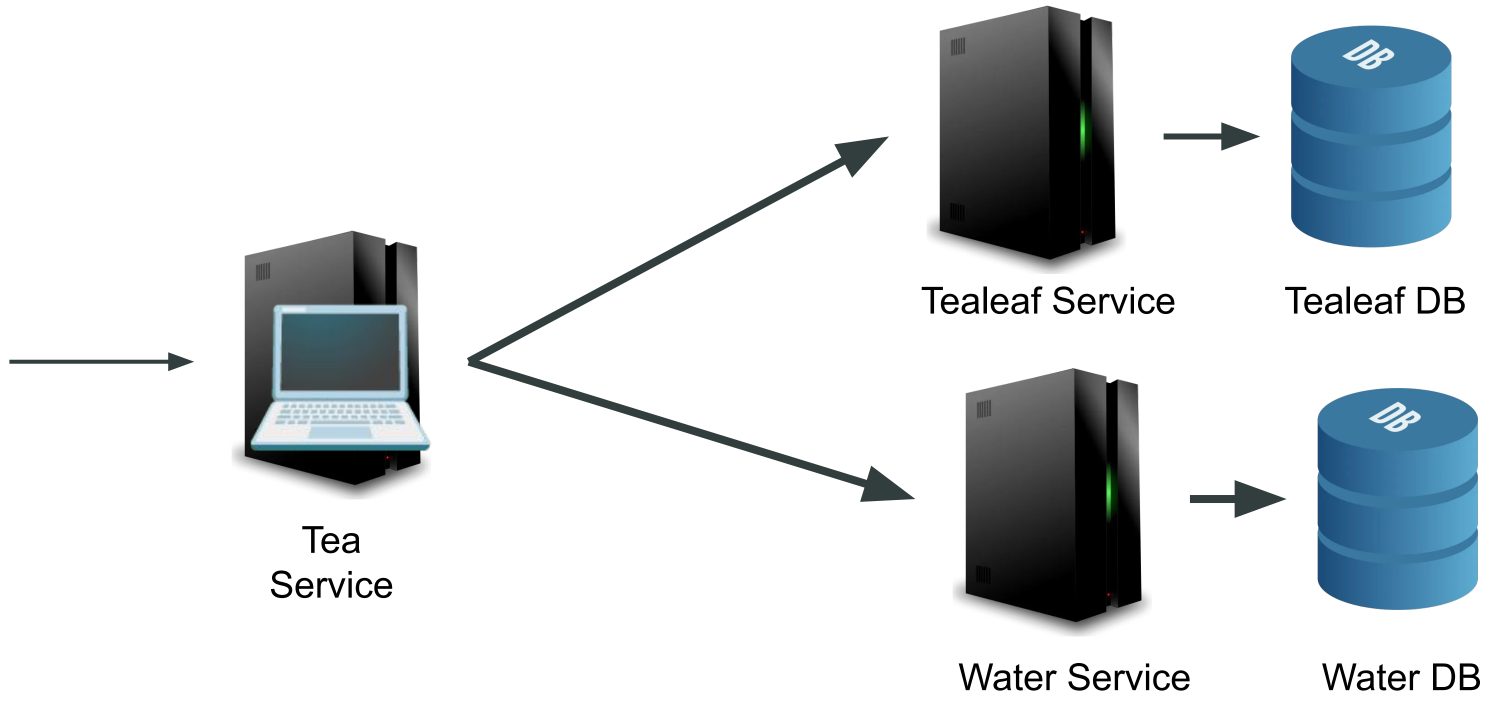
Instrumentation for Spring Projects, 3rd party libraries, your app

Support for various backends

DEMO 🍵

github.com/jonatan-ivanov/teahouse

Architecture



Observation API

You want to instrument your application...

- Add Logs
  (application logs)
- Add Metrics
- Add Distributed Tracing

Observation API basic usage example

```java
Observation observation = Observation.start("talk",registry);
try { // TODO: scope
    doSomething(); // ← This is what we’re observing
}
catch (Exception exception) {
    observation.error(exception);
    throw exception;
}
finally { // TODO: attach tags (key-value)
    observation.stop();
}
```

Configuring an ObservationHandler (without Boot)

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

registry.observationConfig()
     .observationHandler(new MetricsHandler(...))
     .observationHandler(new TracingHandler(...))
     .observationHandler(new LoggingHandler(...))
     .observationHandler(new AuditEventHandler(...));
```

Observation API shortcuts

```java
Observation.createNotStarted("talk",registry)
    .lowCardinalityKeyValue("event", "ConFoo")
    .highCardinalityKeyValue("uid", userId)
    .observe(this::talk);
@Observed
```

Observation.Context

- Holds the state/data of an Observation
  - e.g. request/response object
- **`ObservationHandler`** / **ObservationConvention** will receive it
- Mutable, you can add data to it
  - Instrumentation time
  - Pass data between handler methods

Let’s look at some code

Observation Predicate and Filter

`ObservationPredicate`

- **`BiPredicate`**: (name, context) → **`Boolean`**
- Whether an Observation is ignored (noop)

`ObservationFilter`

- Modify the **`Observation.Context`**
- Right before **`ObservationHandler#onStop`**
- e.g. common tags (KeyValues)

Conventions for instrumentation

- Instrumentation by default provides a convention
  - Naming, tags (KeyValues)
- You may want to customize the convention for an instrumentation without rewriting the instrumentation
- Control the timing of changing conventions
  - Convention changes are breaking changes

Introducing ObservationConvention

- Instrumentation can use a default **`ObservationConvention`** while allowing users to provide a custom implementation
- Extend a default implementation or implement your own
- See, for example, Spring Framework’s docs

Let’s look at some code (again)

Micrometer Docs Generator

Documenting instrumentation

- Keeping documentation in sync with the implementation is difficult and error-prone.
- Introducing Micrometer Docs Generator
- Define an **`ObservationDocumentation`** enum for your Observation-based instrumentation and generate documentation based on it as part of the build
- Integrate it with **`ObservationConvention`**

Let’s look at some code
(again)
(again)

DEMO 👂 🎶

github.com/jonatan-ivanov/teahouse

Tomorrow @ 10:00

A Million Ways to Fail in Production

Thank you!

@jonatan_ivanov

develotters.com

slack.micrometer.io

GH: jonatan-ivanov/teahouse