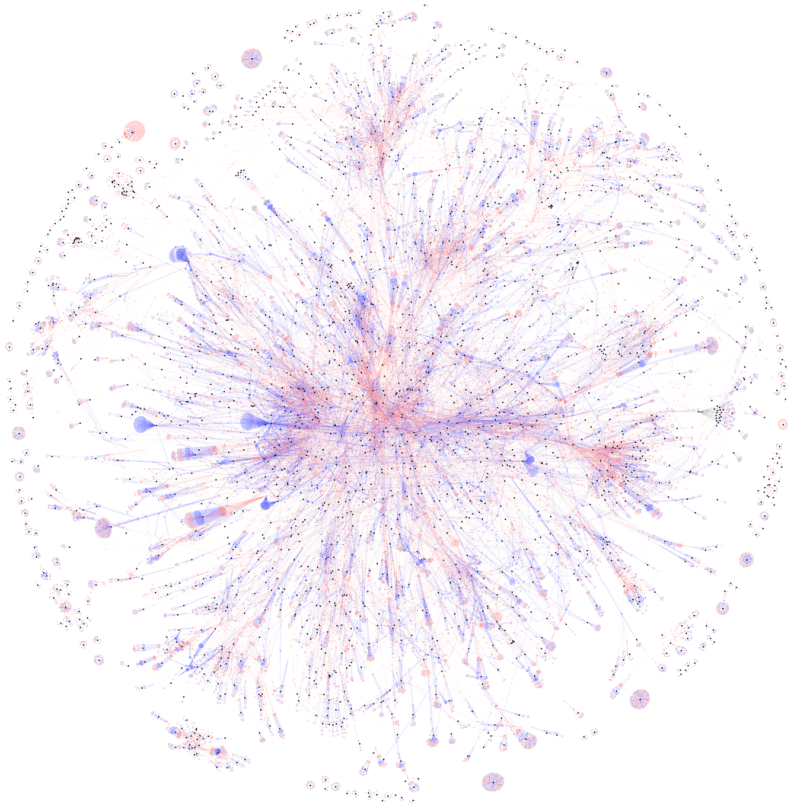
Spring Boot Observability

Metrics, Tracing, Logging

Why do we need Observability?

- Today's systems are insanely complex (cloud)
  (Death Star Architecture, Big Ball of Mud)



Metrics, Tracing, Logging

- **Logging**
  What happened (why)?
  Emitting events
- **Metrics**
  What is the context?
  Aggregating data
- **Distributed Tracing**
  Why happened?
  Recording causal ordering of events
- **… and more**
  Configuration, Health, Auditing, …

Spring Boot Actuator Setup

Production-Ready features

- Observability is the ability to measure the internal state of a system only by its external outputs.
- The actuator module lets you monitor and interact with your running application. Logging, metrics, tracing, health, and auditing are just a few things.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Endpoints

Endpoints

- Monitoring the application and interacting with it as well e.g. Loggers
- Many endpoints available depending on starters, custom endpoints possible
- Most endpoints are sensitive by default and limited accessible via HTTP
- `/actuator/health` – Shows application health information
- `/actuator/info` – Displays arbitrary application info
- `/actuator/metrics` – Shows ‘metrics’ information
- `/actuator/loggers` – Shows logger details

/actuator/health Endpoint

- Check the health and status of a running application
- Can be used by monitoring software to alert
- Information collected from beans implementing `HealthIndicator`
- *Sensitive by default*, can be changed


```
{ "status": "UP",
  "components": {
    "diskSpace": {
      "status": "UP",
      "details": {
        "total": 499963174912,
        "free": 334100381696,
        "threshold": 10485760 }
    },
    "ping": {
      "status": "UP"
    } } }
```

Health Indicator Example

```
@Component
public class FlakyCustomHealthIndicator extends AbstractHealthIndicator {
    private Random random = new Random();
    @Override
    protected void doHealthCheck(Health.Builder builder) throws Exception {
        switch (random.nextInt(3)) {
            case 1:
                builder.up();
                break;
            case 2:
                builder.down();
                break;
            case 3:
                builder.outOfService();
                break;
            default:
                builder.unknown();
                break;
        }
        builder.withDetail("message", "Hello from FlakyCustomHealthIndicator");
    }
}
```

/actuator/info Endpoint

Customizing app information

```
info.app.name=Spring Demo Application
info.app.description=My Spring Boot Demo application
info.app.version=1.0.0
```

```
{
  "app": {
    "version":"1.0.0",
    "description":"My Spring Boot Demo application",
    "name":"Spring Demo Application"
  }
}
```

/actuator/metrics Endpoint

- Publishes information about OS, JVM and Application-level metrics
  - memory, heap, processors, threads, classes loaded, classes unloaded, thread, HTTP metrics ...

```
{
  "names": [
    "jvm.buffer.memory.used",
    "jvm.memory.used",
    "jvm.buffer.count",
    "http.server.requests",
    "jvm.memory.committed",
    "jvm.buffer.total.capacity",
    "jvm.memory.max",
    "cpu",
    …
  ]
}
```

/actuator/loggers Endpoint

- Application logging levels at runtime
- `/actuator/loggers` or `/actuator/loggers/{logger}`

```
{
  "levels": ["OFF", "ERROR", "WARN", "INFO", "DEBUG", "TRACE"],
  "loggers": {
    "ROOT": {
      "configuredLevel": null,
      "effectiveLevel": "TRACE"
    },
    …
  }
}
```

/actuator/loggers Endpoint Update

- Changing application logging levels at runtime
- `/actuator/loggers/{logger}` partial update

```
$ curl -i -X POST -H 'Content-Type: application/json' \
 -d '{"configuredLevel": "DEBUG"}' \
 http://localhost:8080/actuator/loggers/ROOT
```

Application (Development) Info

/actuator/info with Information

```
info.app.name=Spring Actuator Demo Application
info.app.description=My Spring Boot Actuator Demo Application
info.app.version=1.0.0
info.app.something=Additional information with a random key.

management.info.env.enabled=true
```

```
{
    "app": {
        "description": "My Spring Boot Actuator Demo Application",
        "name": "Spring Actuator Demo Application",
        "something": "Additional information with a random key.",
        "version": "1.0.0"
    }
}
```

/actuator/info with Build Information

```
<plugin>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-maven-plugin</artifactId>
  <executions>
    <execution>
      <goals>
        <goal>build-info</goal>
      </goals>
      <configuration>
        <additionalProperties>
          <encoding.source>UTF-8</encoding.source>
          <encoding.reporting>UTF-8</encoding.reporting>
          <java.source>${maven.compiler.source}</java.source>
          <java.target>${maven.compiler.target}</java.target>
        </additionalProperties>
      </configuration>
    </execution>
  </executions>
</plugin>
```

/actuator/info with Build Information


```
{
    "build": {
        "artifact": "tutorial-observability-demo",
        "encoding": {
            "reporting": "UTF-8",
            "source": "UTF-8"
        },
        "group": "com.fortytwotalents",
        "java": {
            "source": "17",
            "target": "17"
        },
        "name": "tutorial-observability-demo",
        "time": "2023-03-06T10:11:20.343Z",
        "version": "0.0.1-SNAPSHOT"
    }
}
```

/actuator/info with GIT Information

- Adding Git plugin to extract git information.

```
<plugin>
  <groupId>io.github.git-commit-id</groupId>
  <artifactId>git-commit-id-maven-plugin</artifactId>
</plugin>
```

```
management.info.git.enabled=true
management.info.git.mode=full
```

/actuator/info with GIT Information


```
{
    "git": {
        "branch": "main",
        "build": {
            "host": "myXPS13Plus",
            "time": "2023-03-06T09:58:30Z",
            "user": {
                "email": "patrick.baumgartner@42talents.com",
                "name": "Patrick Baumgartner"
            },
            "version": "0.0.1-SNAPSHOT"
        },
        ...
    }
}
```

Further Customization (1)

- Actuator endpoints over a non-standard port
- Restrict access to endpoints over the network

```
# application.properties
management.server.port=8081
management.server.address=127.0.0.1
```

- Most endpoints sensitive by default

```
# application.properties
management.endpoints.web.exposure.include=*
management.endpoints.web.exposure.exclude=env
management.endpoint.health.show-details=always
```

- When using Spring Security configure URLs

Metrics

Adding dependencies

```
<!-- Spring boot actuator to expose metrics endpoint -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<!-- Micrometer core dependecy -->
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-core</artifactId>
</dependency>

<!-- Micrometer Prometheus registry -->
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

Custom Metrics with Aspects

```
@RestController
public class CustomMetricsController {

    @GetMapping("/countsCallsAspect")
    @Counted("my.custom.aspect.counter")
    public void countCallsAspect() throws Exception {
      // Do something
    }

    @GetMapping("/takesTimeAspect")
    @Timed("my.custom.aspect.timer")
    public void takesTimeAspect() throws Exception {
        Thread.sleep(random.nextInt(5));
    }

}
```

Custom Metrics Using Micrometer API

```
@RestController
public class BeerCustomMetricsController {

    // Global variables omitted for brevity

    BeerCustomMetricsController(MeterRegistry meterRegistry) {
        timer = meterRegistry.timer("beer.orderTimings");

        gauge = Gauge.builder("beer.ordersInQueue", orders, Collection::size)
                .description("Number of unserved orders")
                .register(meterRegistry);

        distributionSummary = DistributionSummary
                .builder("beer.prices")
                .description("Oder price summary distribution")
                .baseUnit("EUR")
                .register(meterRegistry);

        lightOrderCounter = meterRegistry.counter("beer.orders", "type", "light");

        aleOrderCounter = Counter.builder("beer.orders")
                .tag("type", "ale")
                .description("The number of orders ever placed for ale beers")
                .register(meterRegistry);
    }
}
```

Custom Metrics Using Micrometer API (2)

```
@RestController
public class BeerCustomMetricsController {

    // Global variables and constructor omitted for brevity

    @GetMapping("/orderBeer")
    public void orderBeer() {
        Order order = toOrder(random.nextLong());

        // Measuring how long it takes to order
        timer.record(() -> {
            try {
                orders.add(order);
                // Record the prices
                distributionSummary.record(order.getPrice());
                Thread.sleep(random.nextInt(5));
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        });

        // Count the light and ale beers
        if ("light".equals(order.getType())) {
            lightOrderCounter.increment(1.0);
        } else if ("ale".equals(order.getType())) {
            aleOrderCounter.increment();
        }
    }
}
```

Prometheus & Grafana

Tracing

Tracing

- Adding tracing to your application. Currently supported OpenZipkin Brave and OpenTelemetry.

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing</artifactId>
</dependency>`
```

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing-bridge-brave</artifactId>
</dependency>
```

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing-bridge-otel</artifactId>
</dependency>
```

Enriched Logs

- Enriched logs with trace and span ids needs to be enabled.

```
# application.properties
logging.pattern.level=%5p [${spring.application.name:},%X{traceId:-},%X{spanId:-}]
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.0.4)

2023-03-06T13:09:41.270+01:00  INFO [tutorial-observability-demo,,] 13466 --- [           main] t.o.TutorialObservabilityDemoApplication : Starting TutorialObservabilityDemoApplication using Java 19.0.2 with PID 13466 (...)
2023-03-06T13:09:41.277+01:00 DEBUG [tutorial-observability-demo,,] 13466 --- [           main] t.o.TutorialObservabilityDemoApplication : Running with Spring Boot v3.0.4, Spring v6.0.6
2023-03-06T13:09:41.279+01:00  INFO [tutorial-observability-demo,,] 13466 --- [           main] t.o.TutorialObservabilityDemoApplication : No active profile set, falling back to 1 default profile: "default"
2023-03-06T13:09:43.368+01:00  INFO [tutorial-observability-demo,,] 13466 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2023-03-06T13:09:43.381+01:00  INFO [tutorial-observability-demo,,] 13466 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-03-06T13:09:43.382+01:00  INFO [tutorial-observability-demo,,] 13466 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.5]
2023-03-06T13:09:43.574+01:00  INFO [tutorial-observability-demo,,] 13466 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-03-06T13:09:43.577+01:00  INFO [tutorial-observability-demo,,] 13466 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2176 ms
2023-03-06T13:09:44.592+01:00  INFO [tutorial-observability-demo,,] 13466 --- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 13 endpoint(s) beneath base path '/actuator'
2023-03-06T13:09:44.731+01:00  INFO [tutorial-observability-demo,,] 13466 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2023-03-06T13:09:44.766+01:00  INFO [tutorial-observability-demo,,] 13466 --- [           main] t.o.TutorialObservabilityDemoApplication : Started TutorialObservabilityDemoApplication in 4.201 seconds (process running for 5.04)
2023-03-06T13:09:47.455+01:00  INFO [tutorial-observability-demo,,] 13466 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-03-06T13:09:47.455+01:00  INFO [tutorial-observability-demo,,] 13466 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2023-03-06T13:09:47.457+01:00  INFO [tutorial-observability-demo,,] 13466 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
2023-03-06T13:09:47.516+01:00 DEBUG [tutorial-observability-demo,6405d80b5f5f72276ee98ca20d77d54f,6ee98ca20d77d54f] 13466 --- [nio-8080-exec-1] c.f.t.observability.LoggingController    : Controller method invoked by HTTPie/2.6.0
2023-03-06T13:09:49.441+01:00 DEBUG [tutorial-observability-demo,6405d80de27bf04341cc5d6c6c4bcf05,41cc5d6c6c4bcf05] 13466 --- [nio-8080-exec-3] c.f.t.observability.LoggingController    : Controller method invoked by HTTPie/2.6.0
2023-03-06T13:09:51.032+01:00 DEBUG [tutorial-observability-demo,6405d80f611d11a6caa9a2c81fc990c6,caa9a2c81fc990c6] 13466 --- [nio-8080-exec-5] c.f.t.observability.LoggingController    : Controller method invoked by HTTPie/2.6.0
2023-03-06T13:09:52.640+01:00 DEBUG [tutorial-observability-demo,6405d810858ab048b628c00d95ccfdf9,b628c00d95ccfdf9] 13466 --- [nio-8080-exec-7] c.f.t.observability.LoggingController    : Controller method invoked by HTTPie/2.6.0
```

Visualizing Latency