Application Behaviour: Exposed

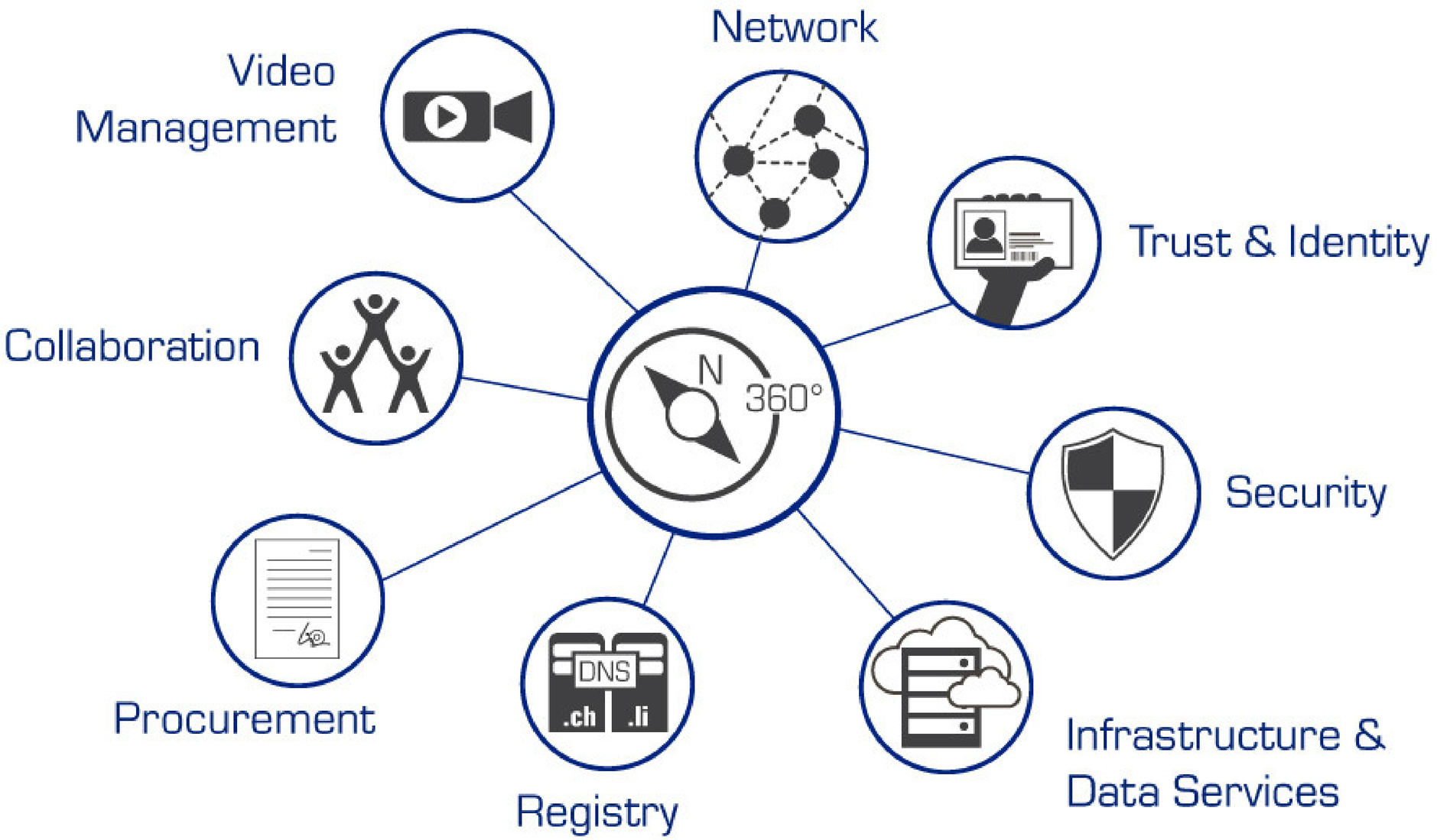
Delight everyone with insightful dashboards

SWITCH

Etienne Dysli Metref
software engineer
etienne.dysli-metref@switch.ch

Java User Group Switzerland 2021-08-24

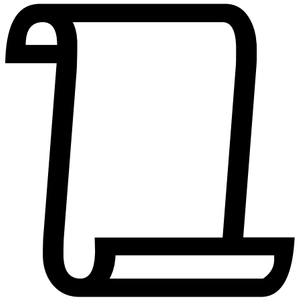
SWITCH


Network
Video Management
Trust & Identity
Collaboration
N
360°
Security
Procurement
DNS
.ch
.li
Registry
Infrastructure & Data Services

What's your application doing when you're not looking?

The application observability trinity

Instrument application code to surface information

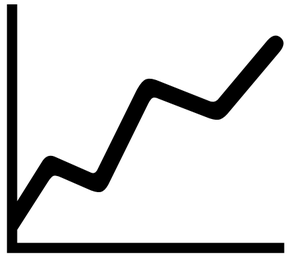


logging

journal of events

tracing

correlate requests
between components



metrics

statistics over time

Which metrics are interesting?

Technical

request Rate, Error and Duration (RED)

Utilisation, Saturation and Error (USE)

cache hit ratio

processor usage

memory usage

thread pool size

queue size

Business

Monthly Active Users (MAU)

number of new customers

customer churn rate

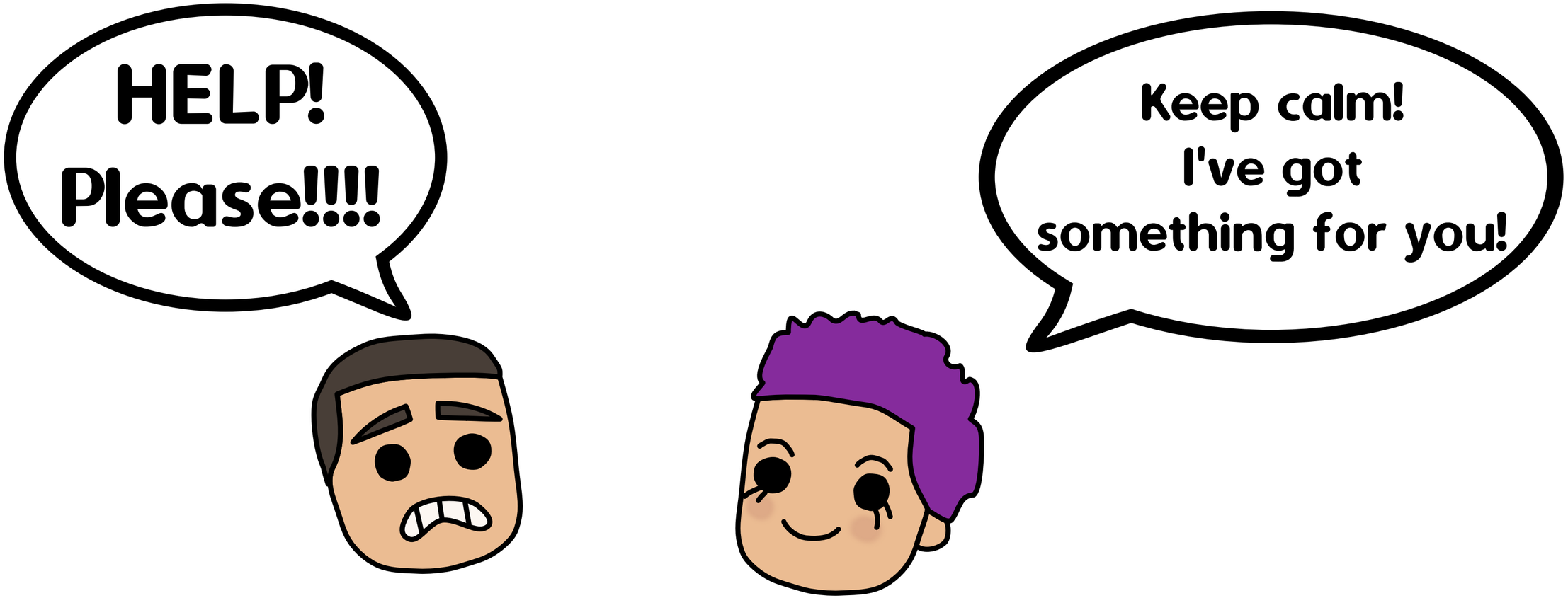
most used functionality

total value of orders

number of items in shopping cart

time spent on product page

authentication requests rate


HELP!
Please!!!!
Keep calm!
I've got
something for you!

Observability standardisation initiatives

OpenTelemetry

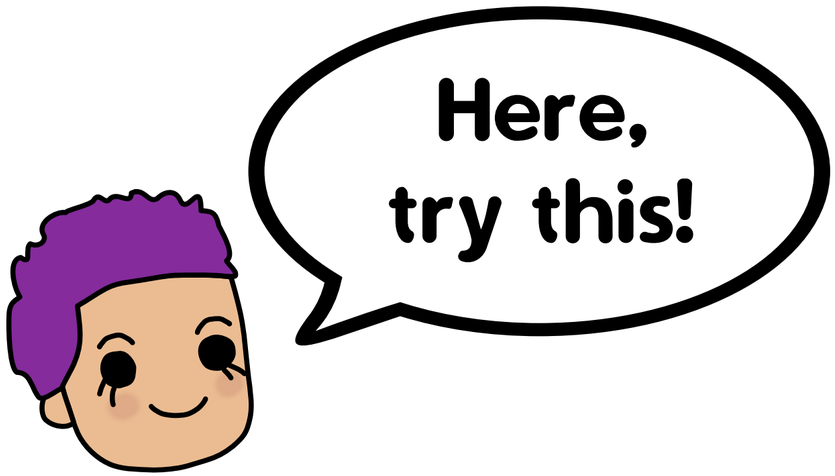
vendor-neutral interfaces and libraries to capture application metrics and distributed traces

Micrometer

vendor-neutral application metrics facade (Java)
“Think SLF4J, but for metrics”

⇒ flexibility of backend or export format without changing code

Quick win example



Spring Boot metrics: it's (almost) free!

Adding metrics to your Spring Boot application in three easy steps

1. Add dependencies on `spring-boot-starter-actuator` and `micrometer-registry-prometheus` to your application
2. Expose the actuator endpoint for Prometheus
3. Export HTTP request latency as histogram with custom boundaries

Spring Boot metrics: it's (almost) free!

1. Add dependencies on `spring-boot-starter-actuator` and `micrometer-registry-prometheus` to your application

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
  <dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
    <scope>runtime</scope>
  </dependency>
</dependencies>
```

2. Expose the actuator endpoint for Prometheus
3. Export HTTP request latency as histogram with custom boundaries

Spring Boot metrics: it's (almost) free!

1. Add dependencies on `spring-boot-starter-actuator` and `micrometer-registry-prometheus` to your application

2. Expose the actuator endpoint for Prometheus

   Endpoint `/actuator/prometheus` enabled by default, but must be exposed in `application.properties` with

   ```
   management.endpoints.web.exposure.include=info,health,prometheus
   ```

3. Export HTTP request latency as histogram with custom boundaries

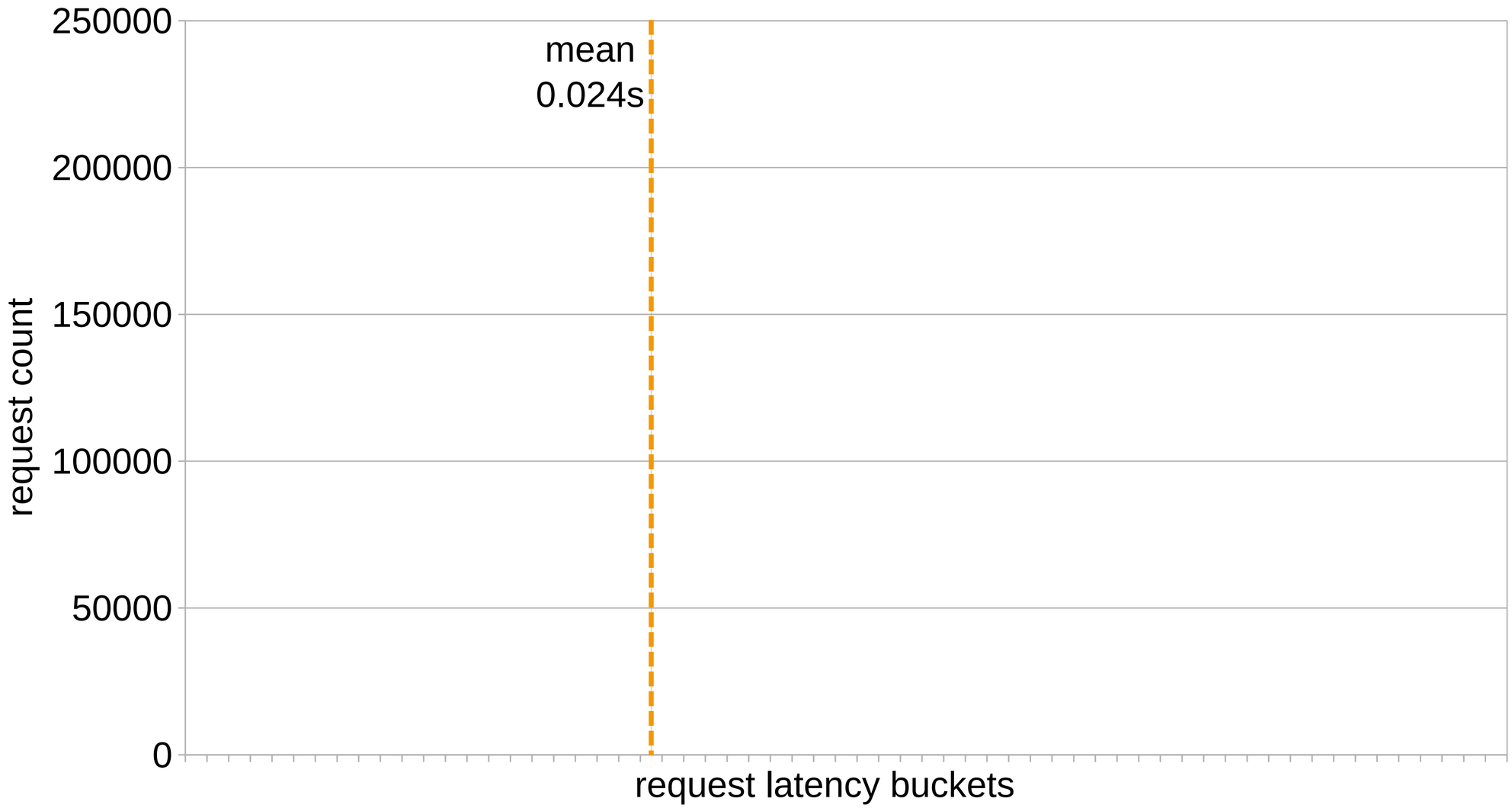
Spring Boot metrics: it's (almost) free!

1. Add dependencies on `spring-boot-starter-actuator` and `micrometer-registry-prometheus` to your application
2. Expose the actuator endpoint for Prometheus
3. Export HTTP request latency as histogram with custom boundaries

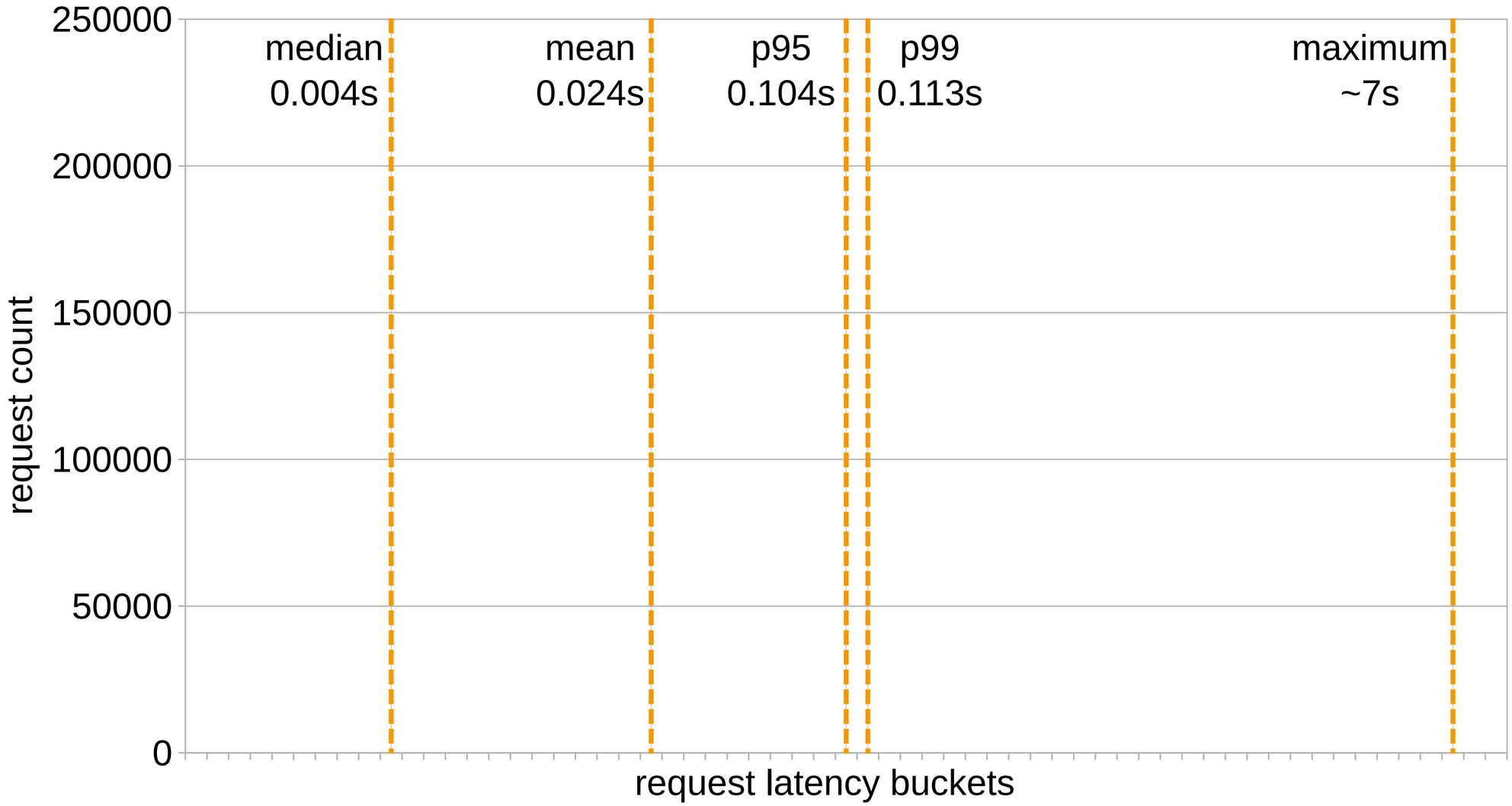
```
management.metrics:
  distribution:
    percentiles-histogram.http.server.requests: true
    minimum-expected-value.http.server.requests: 1ms
    maximum-expected-value.http.server.requests: 10s
    slo.http.server.requests: 100ms
  web.server.request:
    autotime.enabled: true
    metric-name: http.server.requests
```

⚠ *pseudo* `application.yaml`

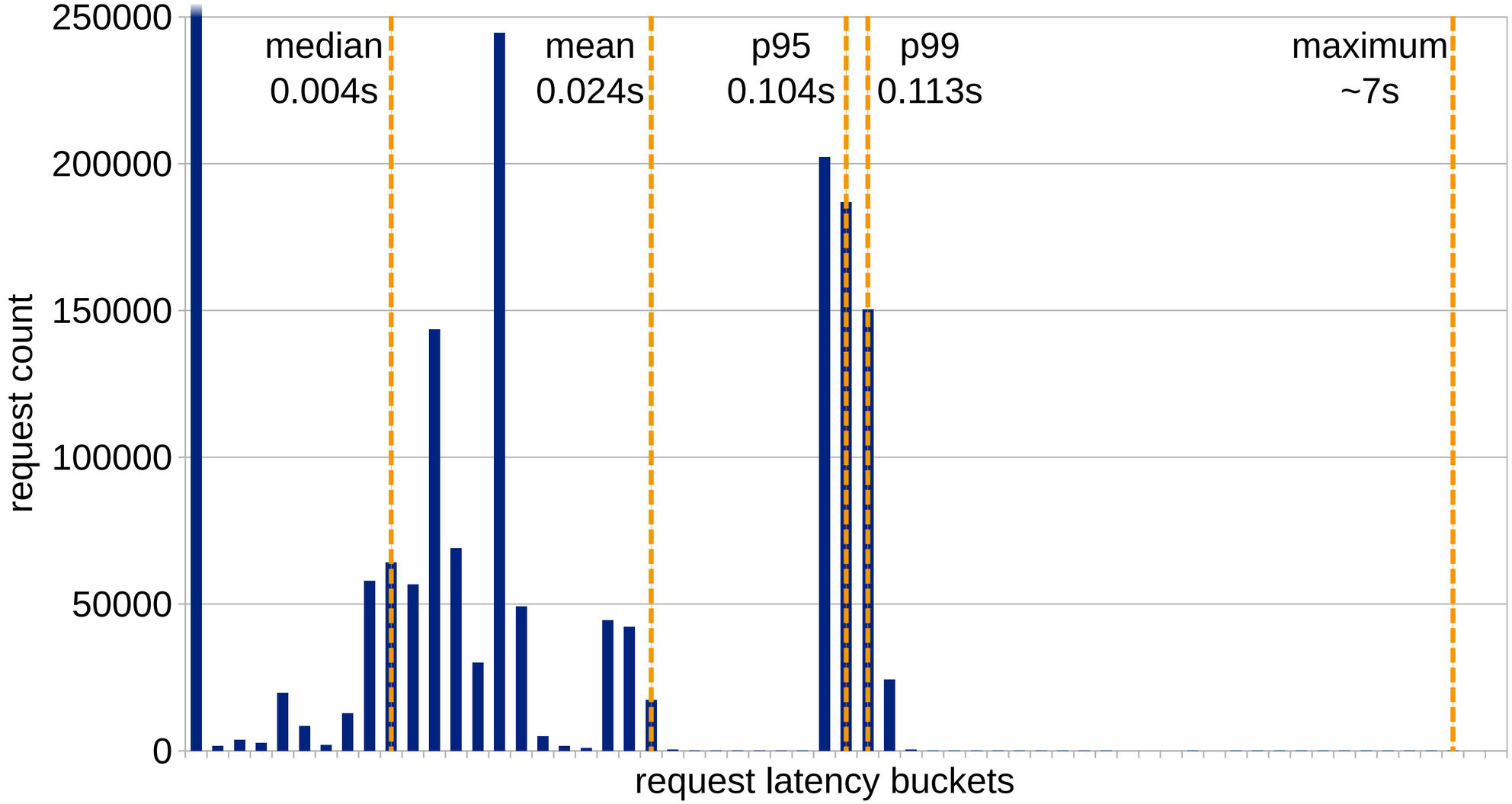
Histograms for request latency



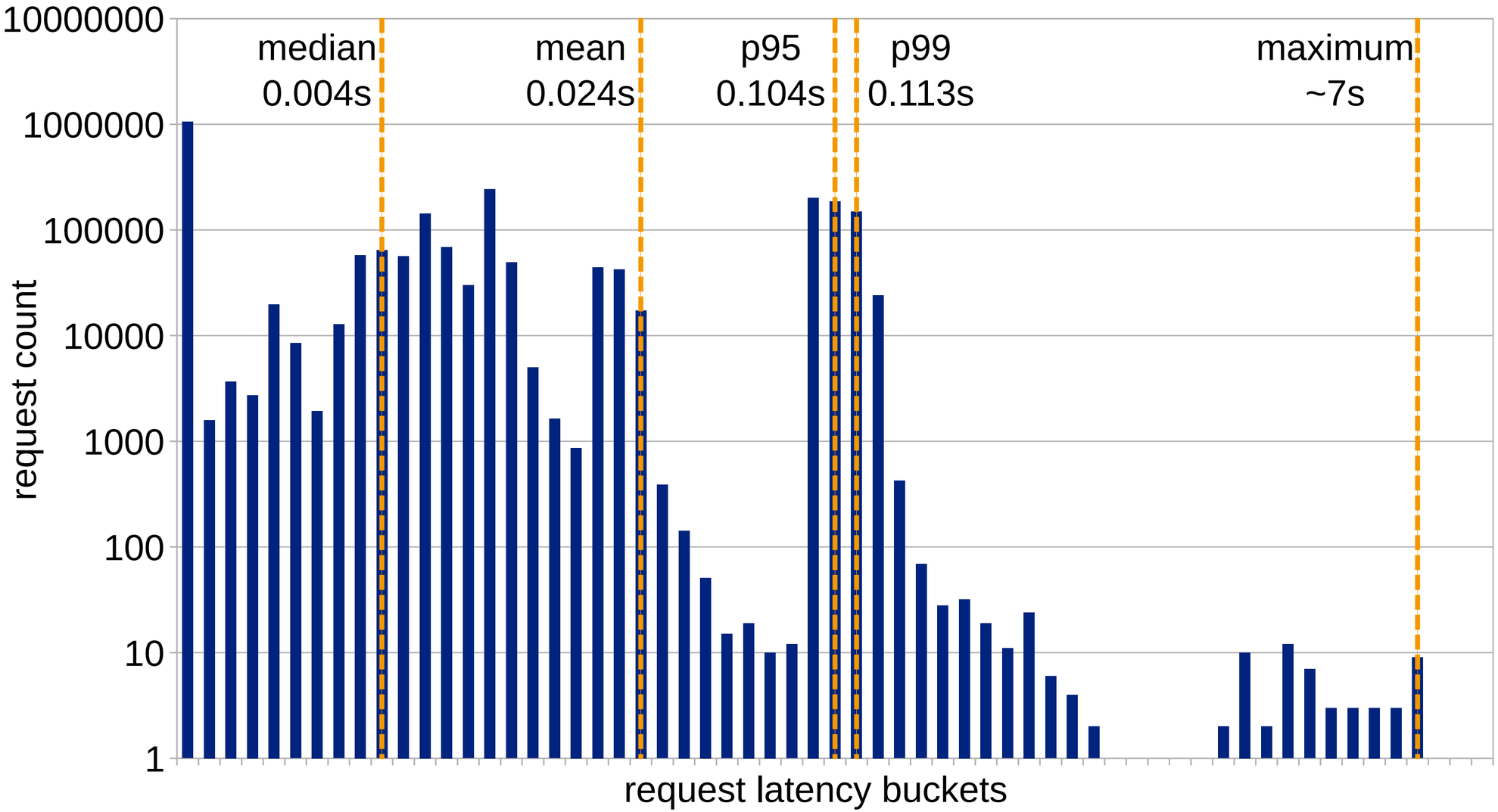
Histograms for request latency



Histograms for request latency



Histograms for request latency



Spring Boot metrics: what you get

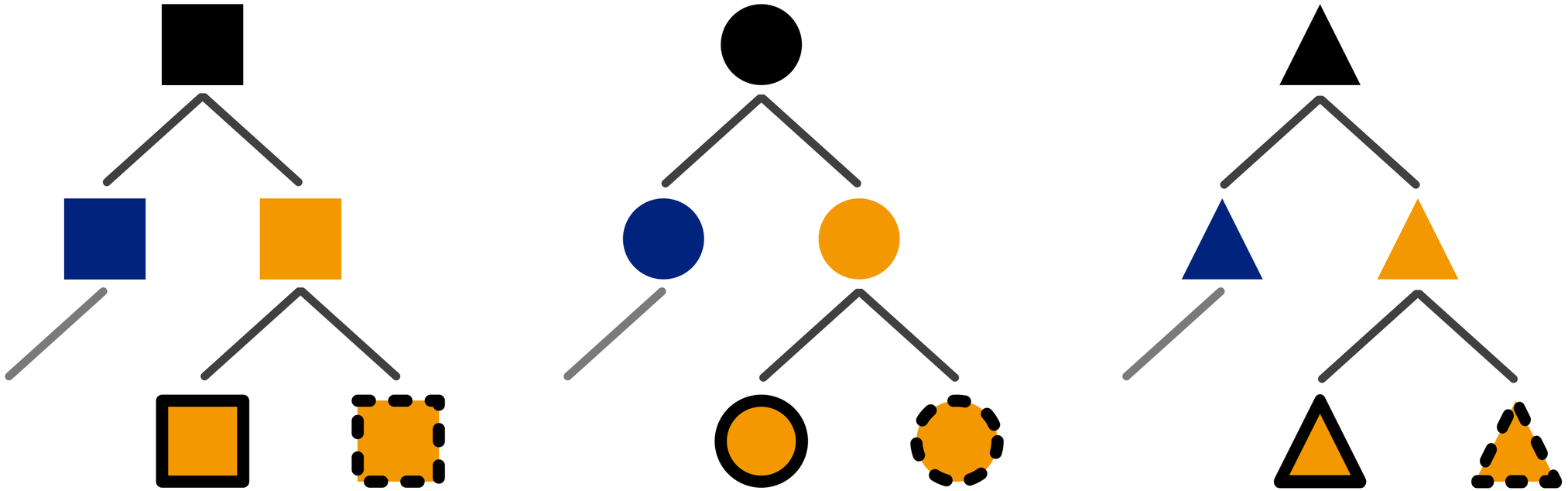
- JVM (memory, GC, threads, classloader)
- process (CPU, file descriptors, uptime)
- logging (number of log events)
- HTTP server and client requests timing
- DataSource, Hibernate, caches

Adding your own business metrics with Micrometer

```
MeterRegistry registry;
Counter counter = registry.counter("interestingOperation");

void interestingOperation() {
  counter.increment();
  //...
}

List<String> customers = registry.gaugeCollectionSize("nbCustomers",
  Tags.empty(), new ArrayList<>());

// ⚠ requires Spring AOP configuration to work
@Timed("interestingQuery")
R interestingQuery() {
  //...
}
```


Not bad,
but not very
useful in there...
Let's
collect!!

Organising metrics: hierarchical system



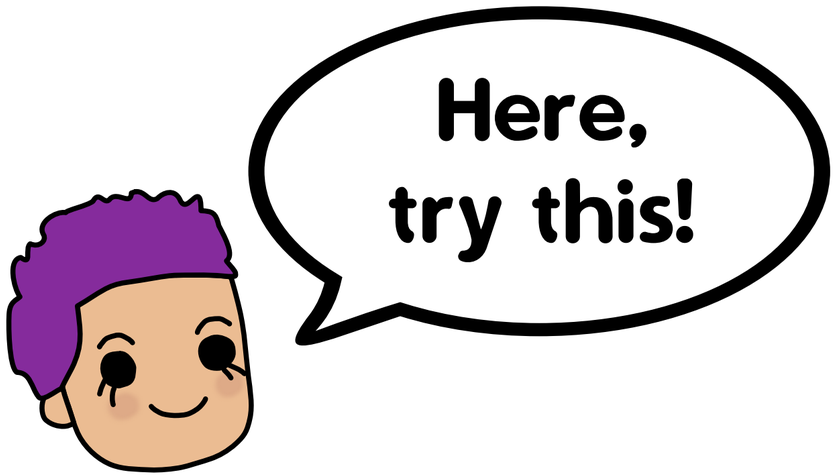
```
things_count.square.blue.plainborder    2
things_count.square.orange.plainborder  4
things_count.square.orange.dottedborder 1
things_count.disc.blue.plainborder      0
things_count.disc.blue.dottedborder     3
```

Organising metrics: dimensional system


```
things_count{shape=square, colour=blue, border=plain}    2
things_count{shape=square, colour=orange, border=plain}  4
things_count{shape=square, colour=orange, border=dotted} 1
things_count{shape=disc, colour=blue, border=plain}      0
things_count{shape=disc, colour=blue, border=dotted}     3
```

```
sum(things_count) by (colour)
```

Quick win example



Collecting data with Prometheus

- Use the official Docker image `prom/prometheus`
- Configure it to scrape your application in `prometheus.yml`

```
global:
  scrape_interval: '10s'
scrape_configs:
  - job_name: 'example-app'
    metrics_path: '/actuator/prometheus'
    scheme: 'https'
    basic_auth:
      username: 'prometheus'
      password: $(pwgen --secure --symbols 64)
    static_configs:
      - targets: ['app.test.example.com', 'app.example.com']
```

Trap: cardinality explosion

- *Cardinality:* number of possible values
- Prometheus stores one time series **per combination** of labels and values

```
http.server.requests{method="get",status="200",app="example-app"}
http.server.requests{method="get",status="404",app="example-app"}
http.server.requests{method="post",status="201",app="example-app"}
```

Trap: cardinality explosion

- *Cardinality:* number of possible values
- Prometheus stores one time series **per combination** of labels and values

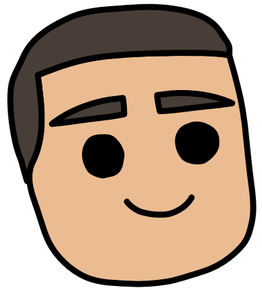
```
http.server.requests{method="get",status="200",app="example-app"}
http.server.requests{method="get",status="404",app="example-app"}
http.server.requests{method="post",status="201",app="example-app"}
```

💥 username, 💥 HTTP request path, 💥 random identifier

```
http_server_requests_seconds_count{...,method="GET",
  path="/idp/profile/SAML2/Redirect/SSO;jsessionid=82E1FDE795578..."}
```

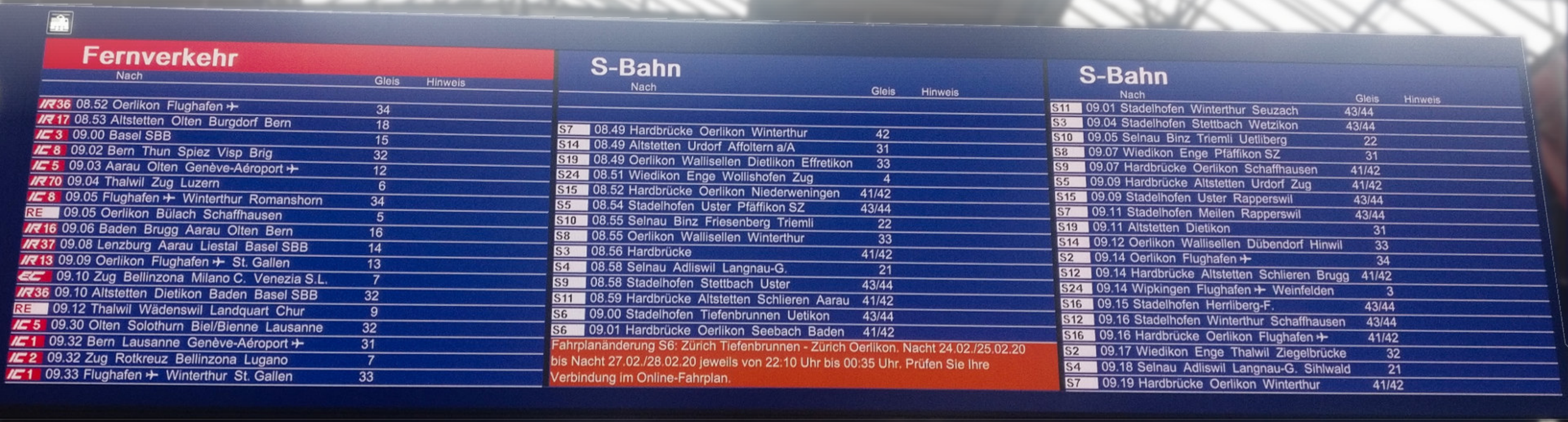
⇒ Logs are better for high-cardinality dimensions

Ok!
What do we do with it now?

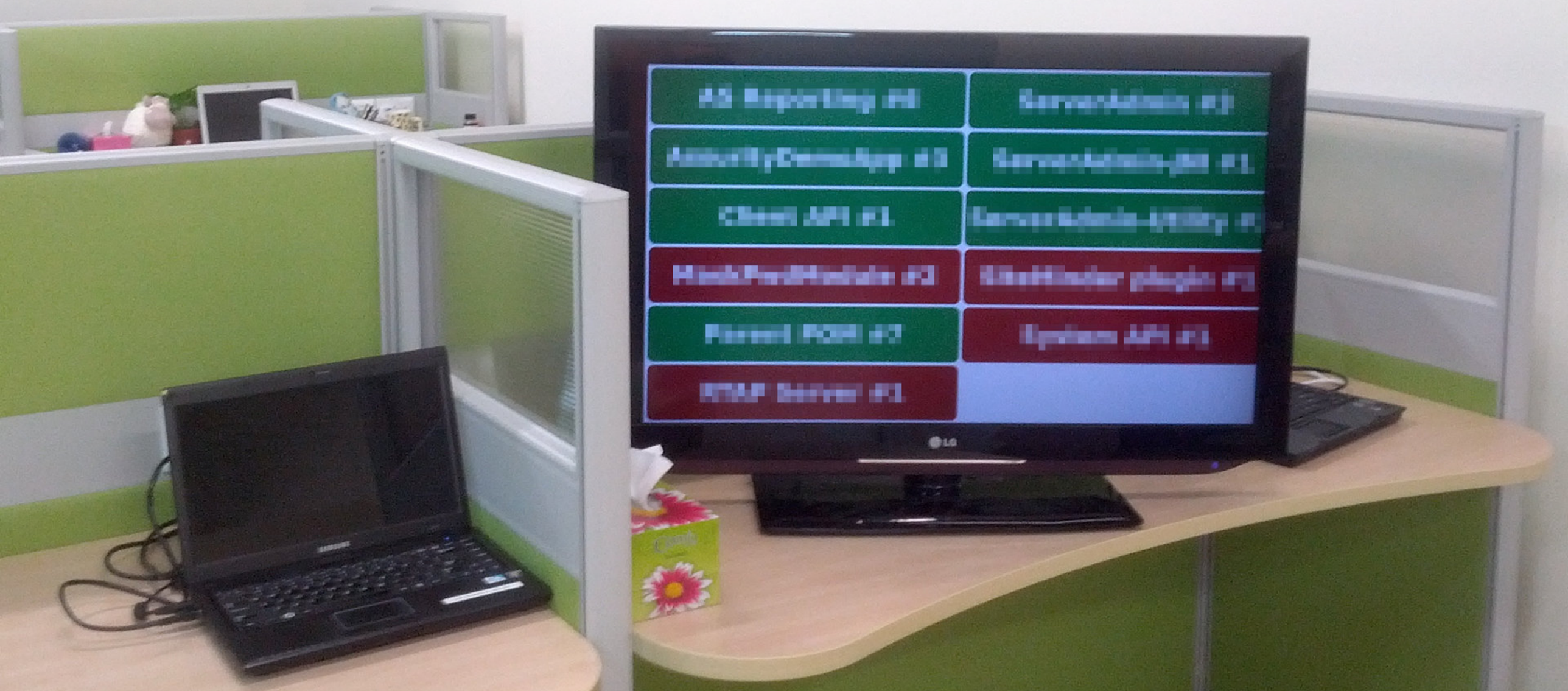


Showtime!

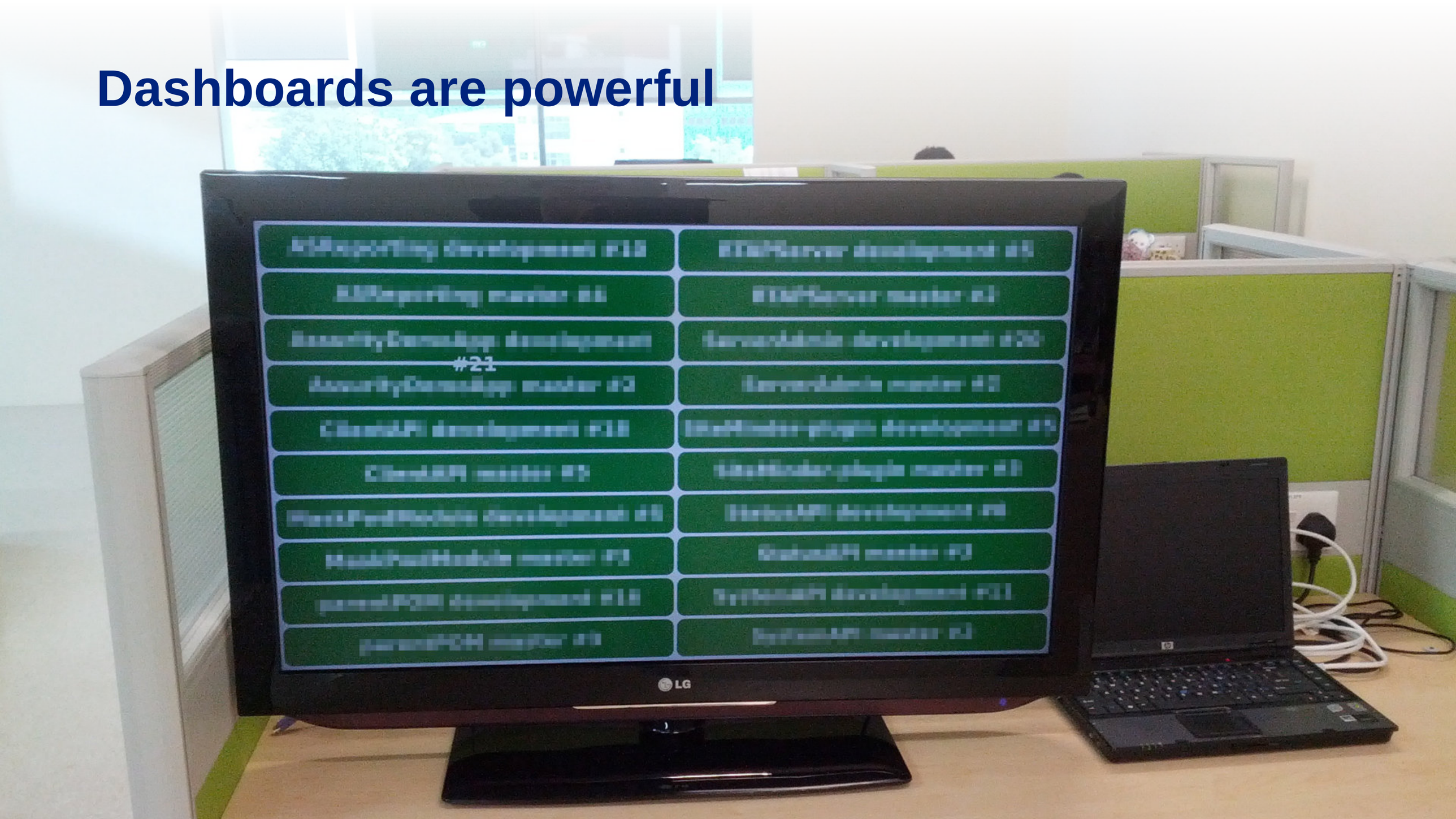
Sharing data efficiently



Dashboards are powerful

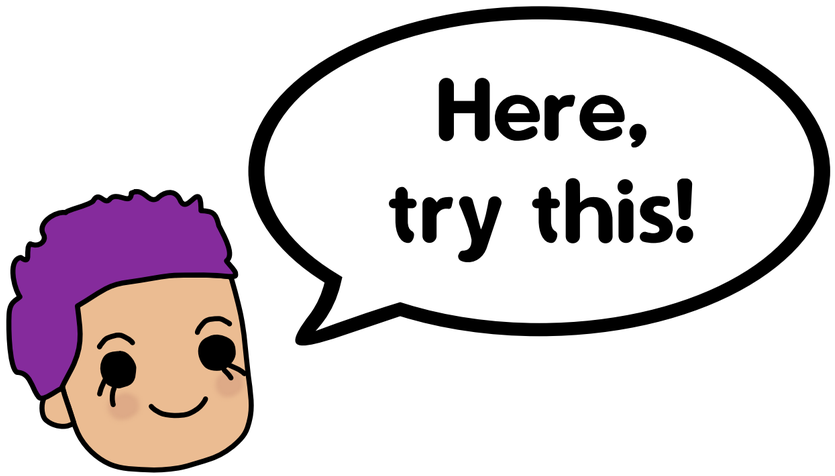


Dashboards are powerful



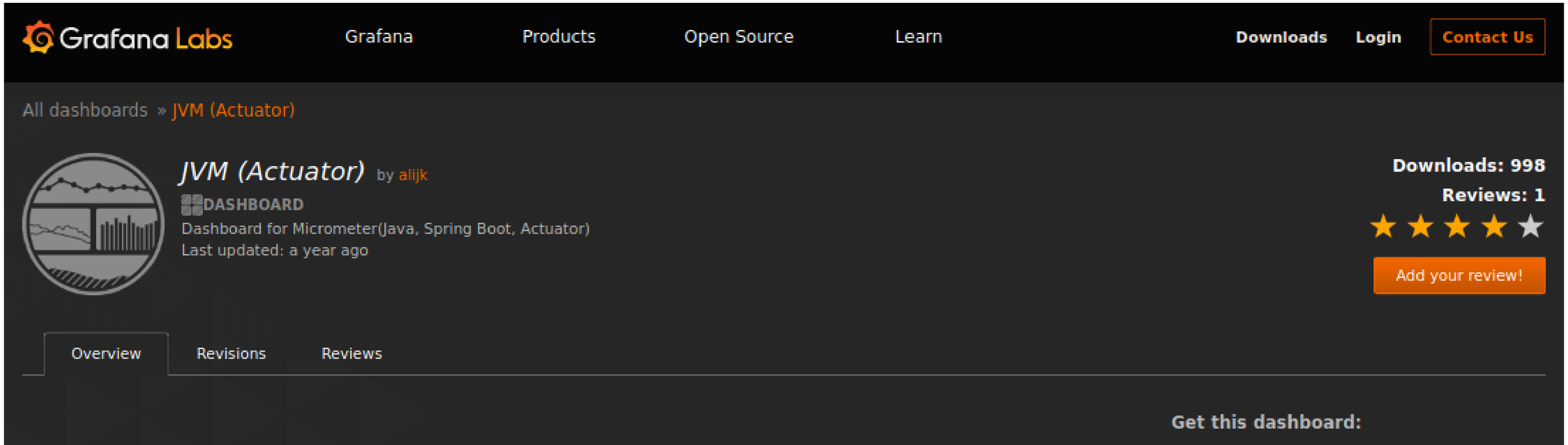
Sharing data efficiently

Quick win example



Nice-looking dashboards with Grafana

- Use the official Docker image `grafana/grafana`, default theme
- Configure Grafana to read from Prometheus
- Community dashboard for Spring Boot applications
  https://grafana.com/grafana/dashboards/9568
  https://grafana.com/grafana/dashboards/4701

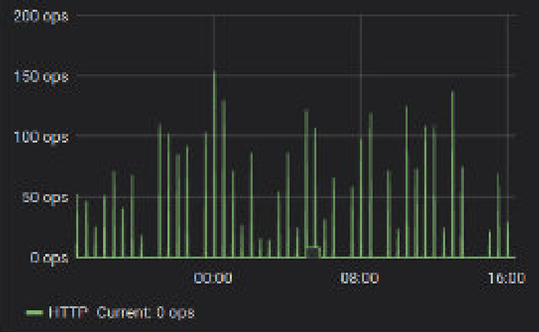


Quick Facts

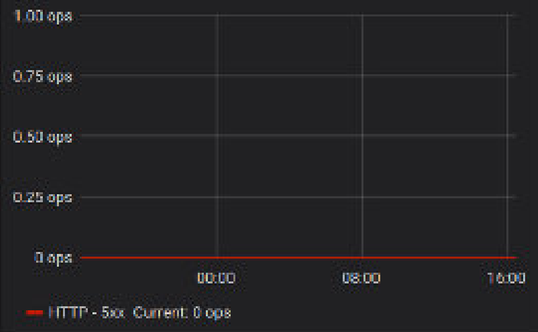
Uptime	Start time	Heap used	Non-Heap used
2.0 week	2020-01-27 17:28:47	78.19%	7.34%

I/O Overview

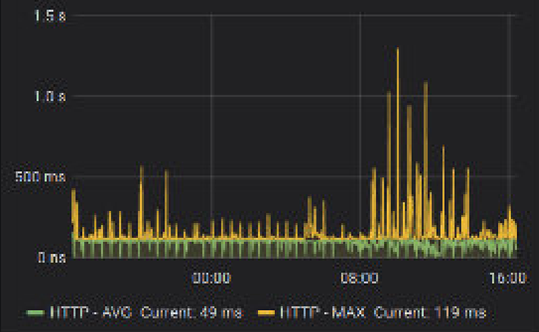
Rate



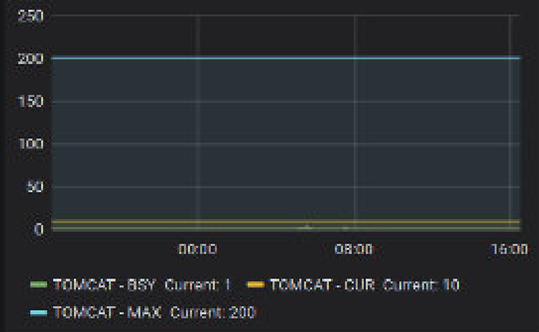
Errors



Duration

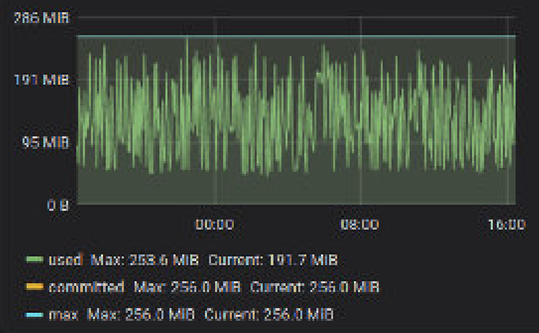


Utilisation

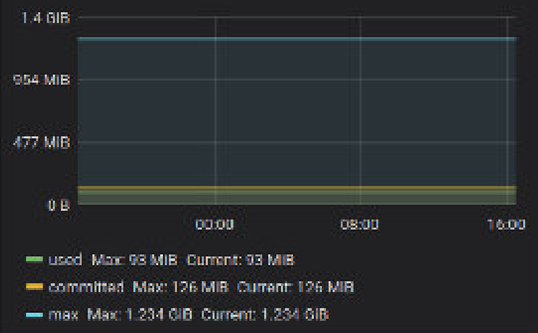


JVM Memory

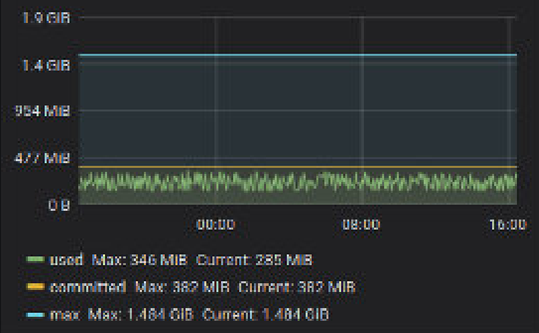
JVM Heap



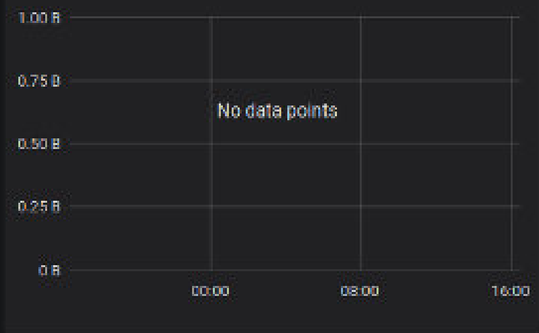
JVM Non-Heap



JVM Total



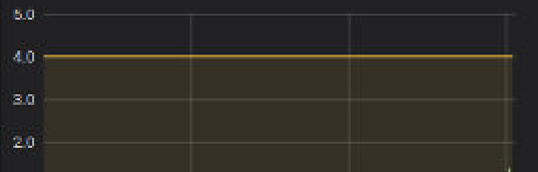
JVM Native Memory



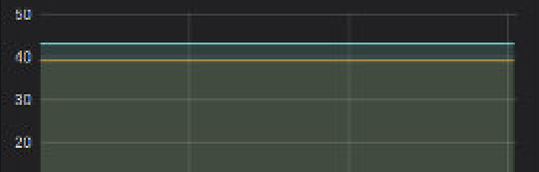
JVM Misc

CPU

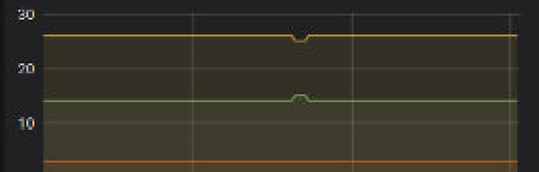
Load



Threads



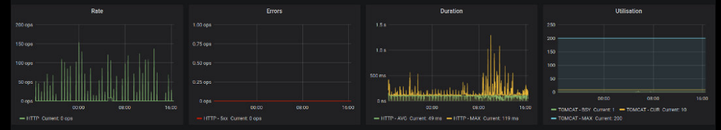
Thread States



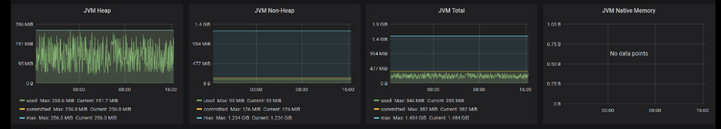
Quick Facts

Uptime	Start time	Heap used	Non-Heap used
2.0 week	2020-01-27 17:28:47	78.19%	7.34%

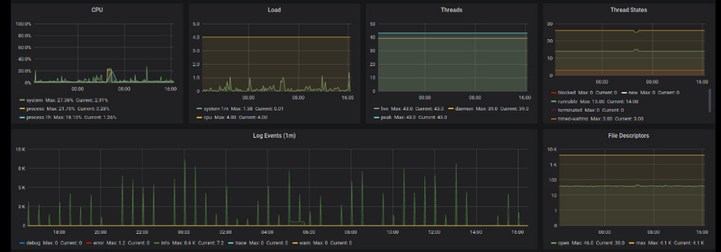
I/O Overview



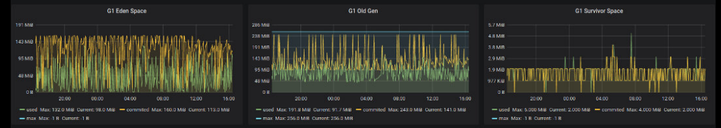
JVM Memory



JVM Misc

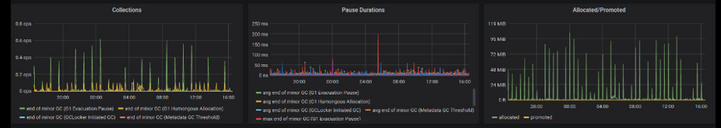


JVM Memory Pools (Heap)



JVM Memory Pools (Non-Heap)

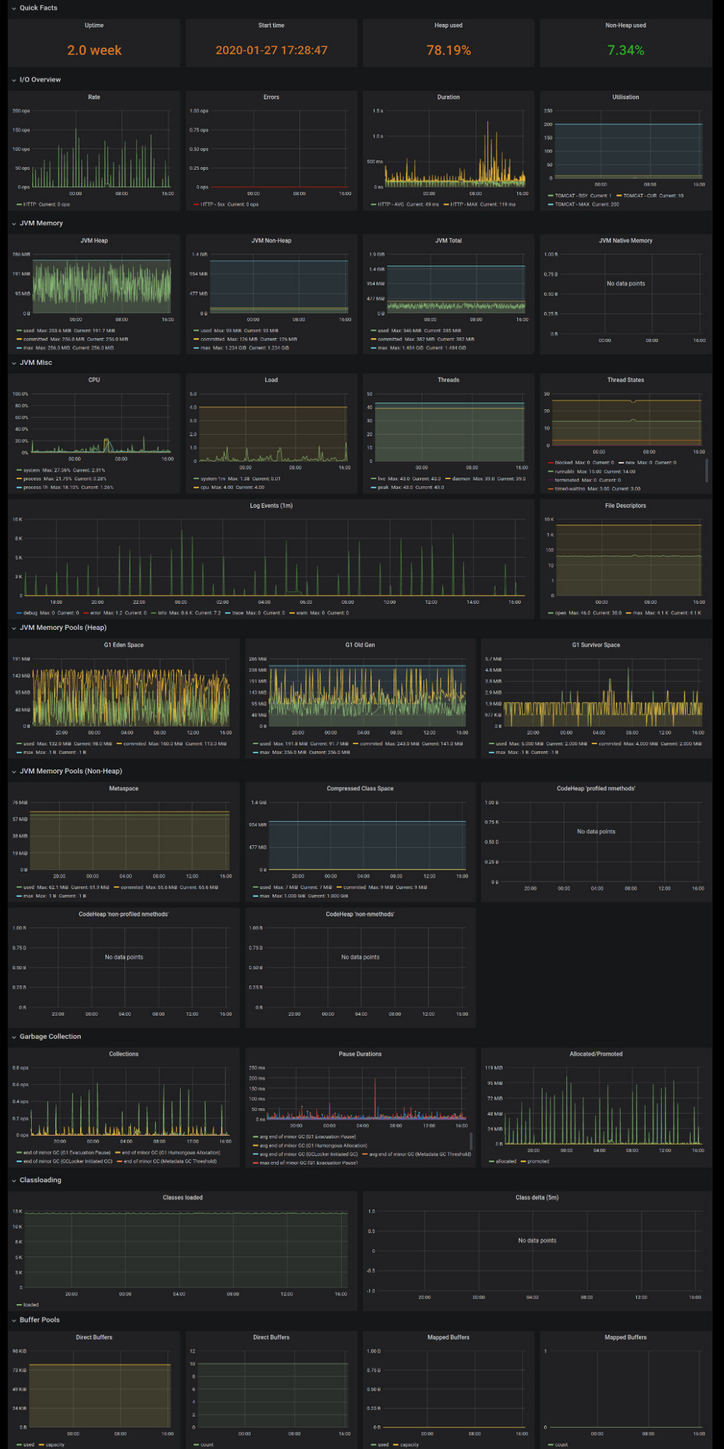
Garbage Collection



Classloading

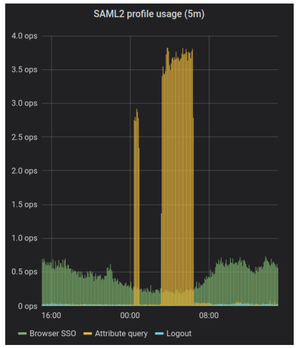
Buffer Pools

Photo by Shandell Venegas on Unsplash

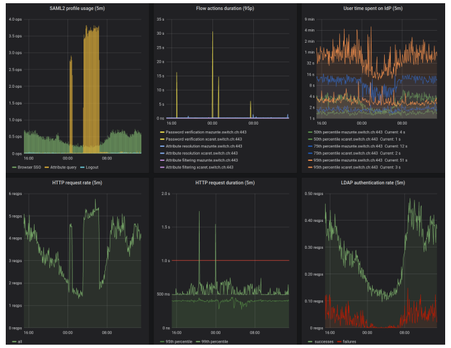


Trap: going overboard with dashboards

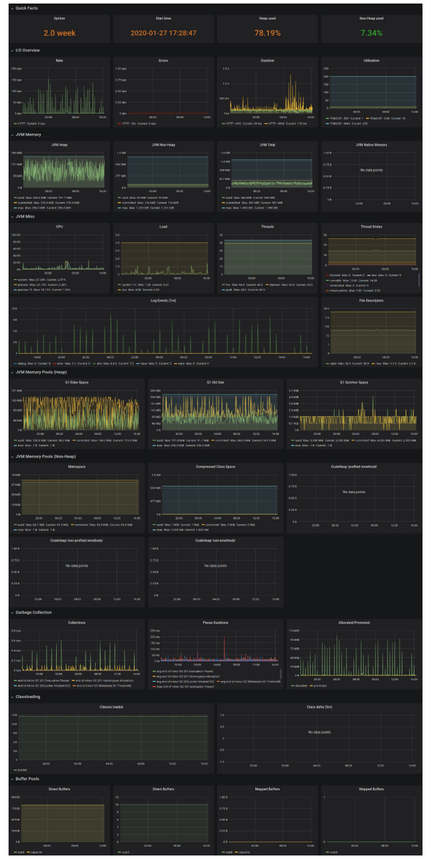
business



technical



troubleshooting

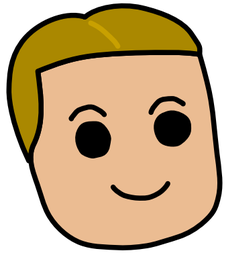


Trap: going overboard with dashboards

- start with simple business or technical metric
- display them → conversation
- then expand into more business metrics

Collect application usage data, everybody wins!

Everyone involved with the product benefits from the collected usage data.

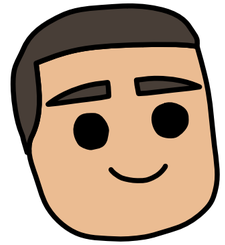


business

How do we prioritise new features?

operations

Where are the bottlenecks?



developers

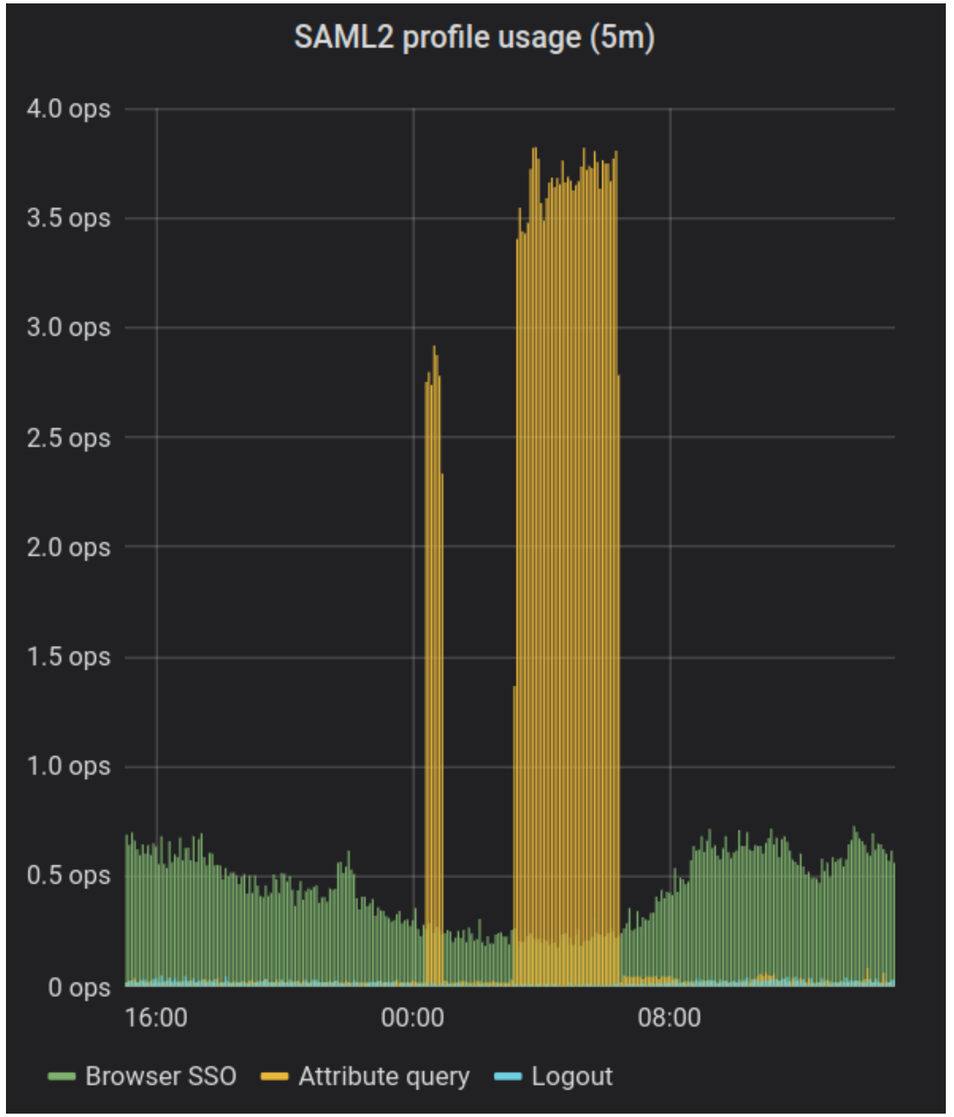
Are technical specifications met?

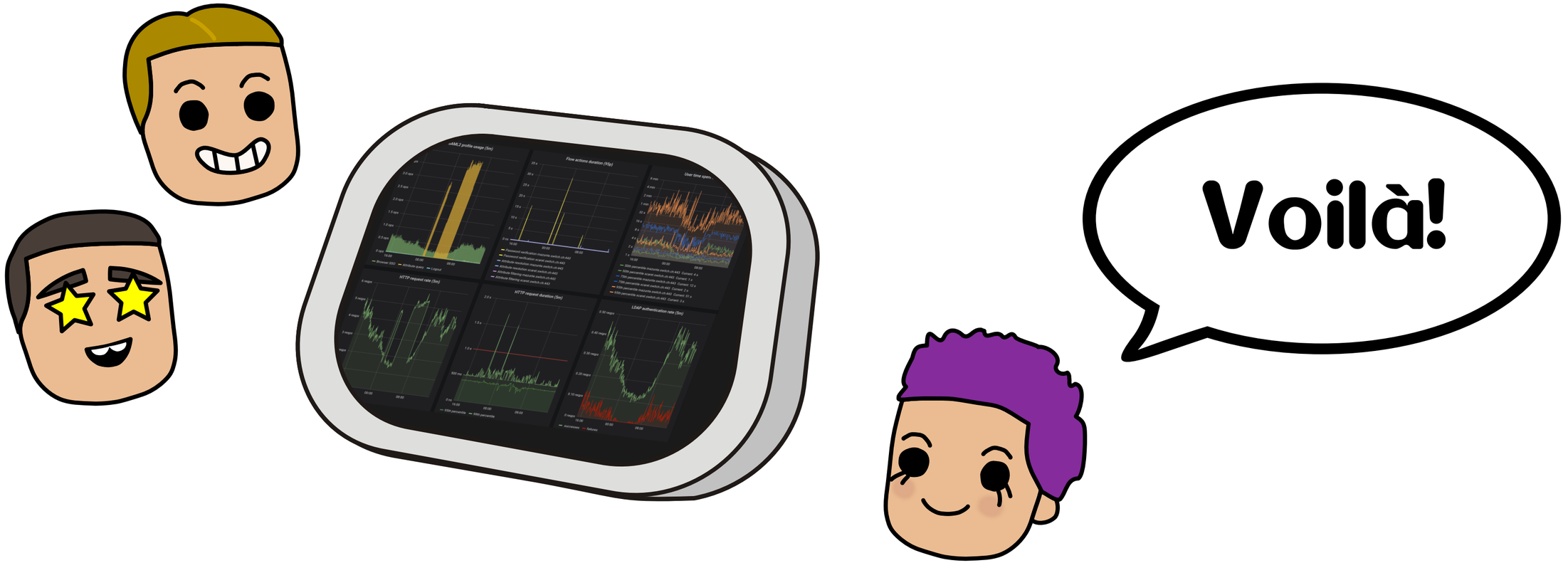
Real-life example: developers / operations

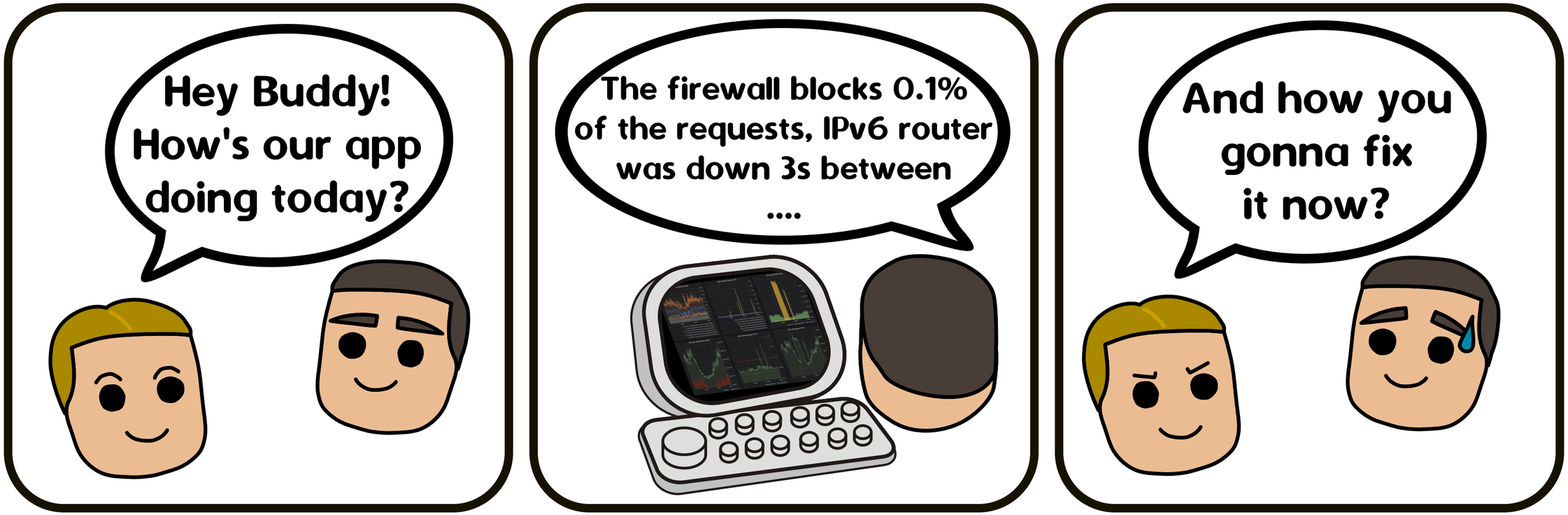
- HTTP response time of the Shibboleth Identity Provider
- Does it fulfill our requirements?

Real-life example: business

- SAML profile usage on the Shibboleth Identity Provider
- Logout function seldom used
  ⇒ no time invested in improving it




Voilà!


Hey Buddy! How's our app doing today?
The firewall blocks 0.1% of the requests, IPv6 router was down 3s between ....
And how you gonna fix it now?

SWITCH

Working for a better digital world

Etienne Dysli Metref
software engineer
etienne.dysli-metref@switch.ch

https://www.switch.ch/edu-id/