

# Application Behaviour: Exposed

Delight everyone with insightful dashboards

SWITCH

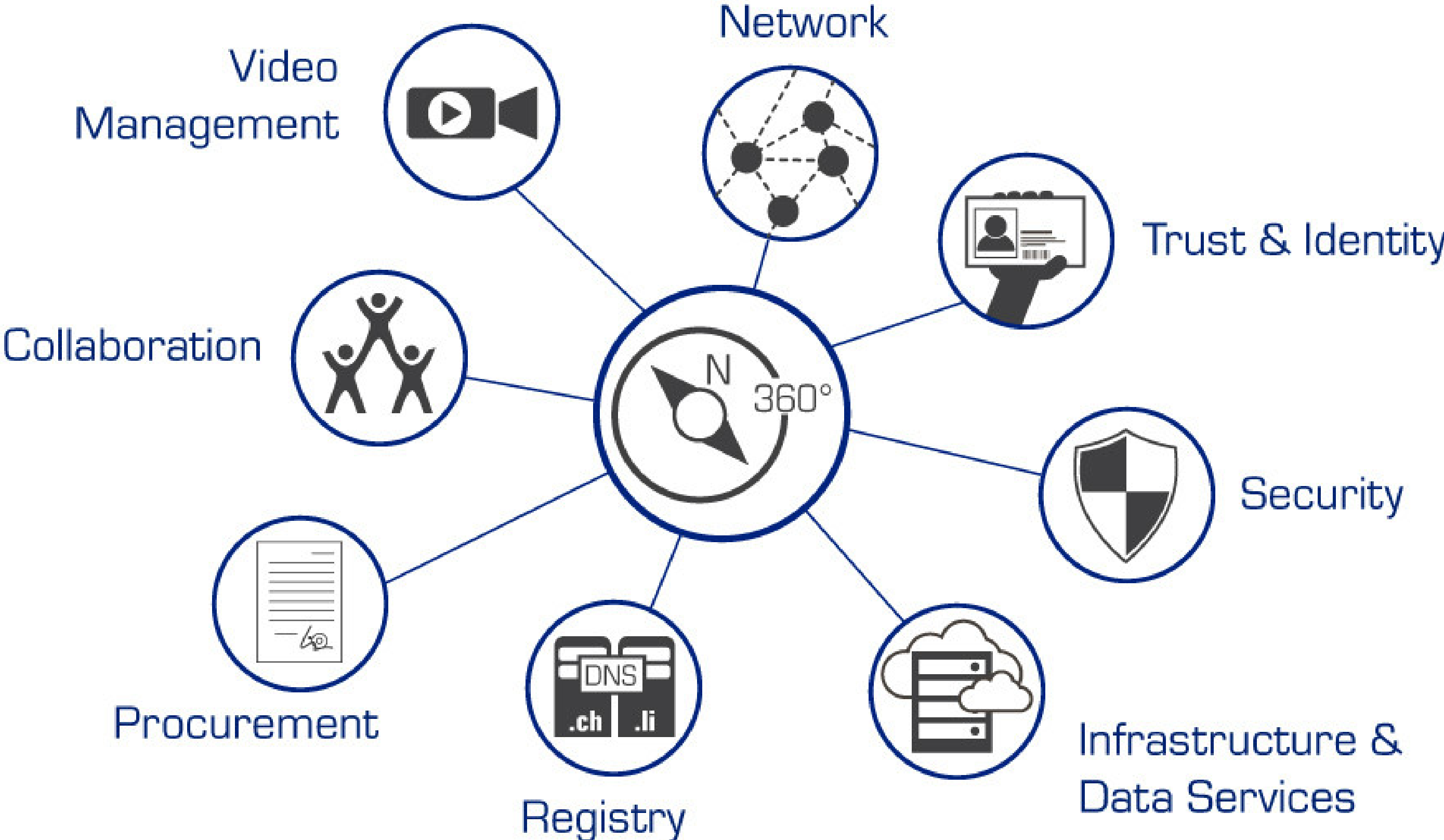
Etienne Dysli Metref

software engineer

[etienne.dysli-metref@switch.ch](mailto:etienne.dysli-metref@switch.ch)

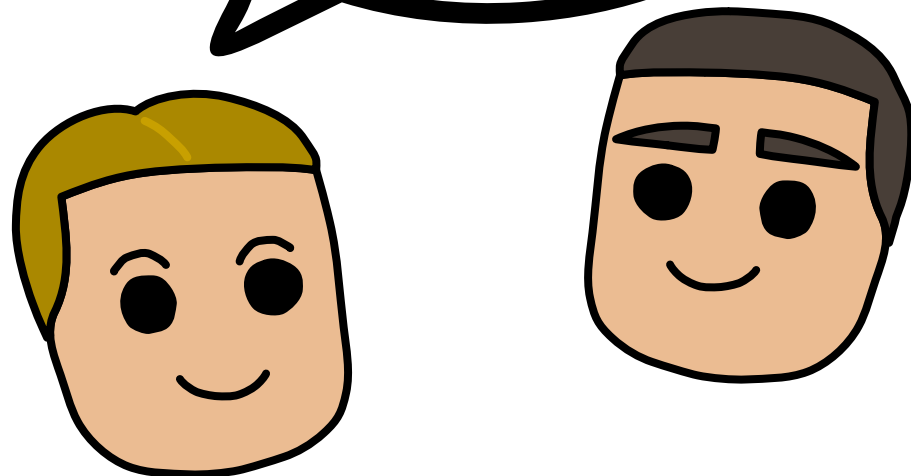
Java User Group Switzerland 2021-08-24

# SWITCH



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

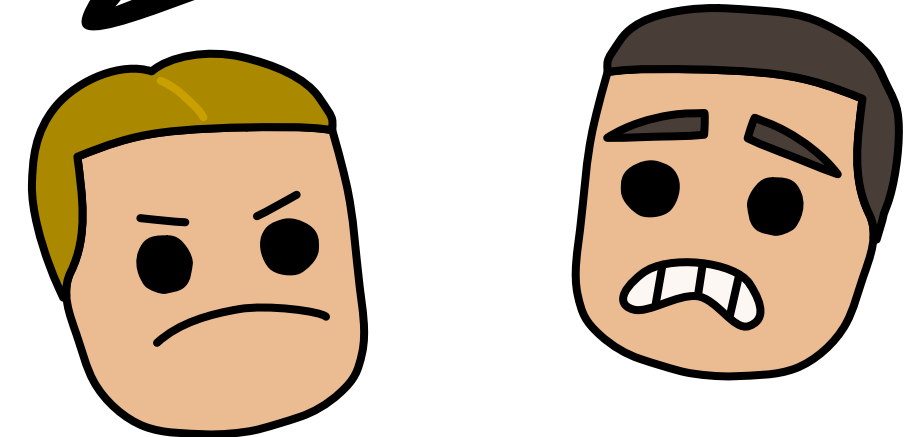
Hey Buddy!  
How's our app  
doing today?



Fine!  
Monitoring  
is all green!

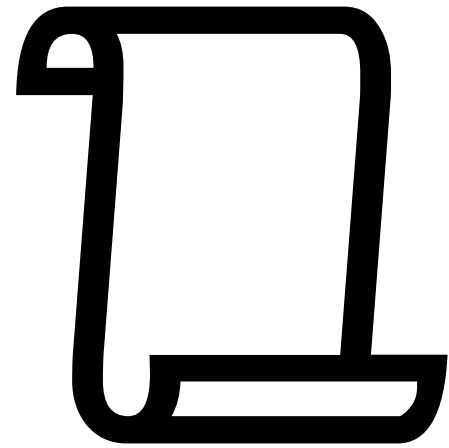


Then, why do  
we have 36 tickets  
for slow connections  
?!?!?



# The application observability trinity

Instrument application code to surface information



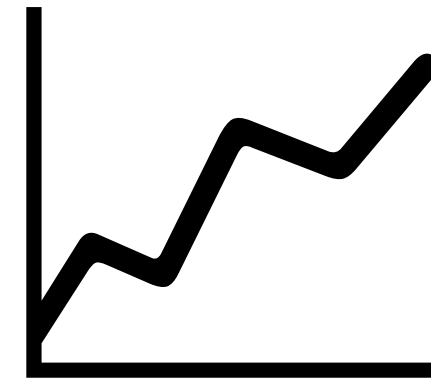
**logging**

journal of events



**tracing**

correlate requests  
between components



**metrics**

statistics over time

# Which metrics are interesting?

request Rate, Error and Duration (RED)

Utilisation, Saturation and Error (USE)

cache hit ratio

## Technical

processor usage

memory usage

thread pool size

queue size

Monthly Active Users (MAU)

number of new customers

customer churn rate

most used functionality

## Business

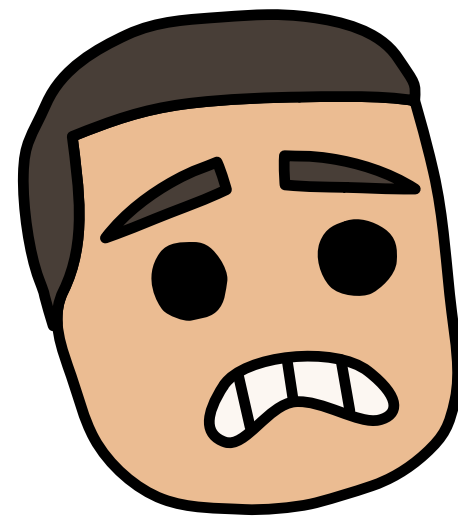
total value of orders

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”



**MICROMETER**

⇒ flexibility of backend or export format without changing code

# Quick win example



**Here,  
try this!**





# 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

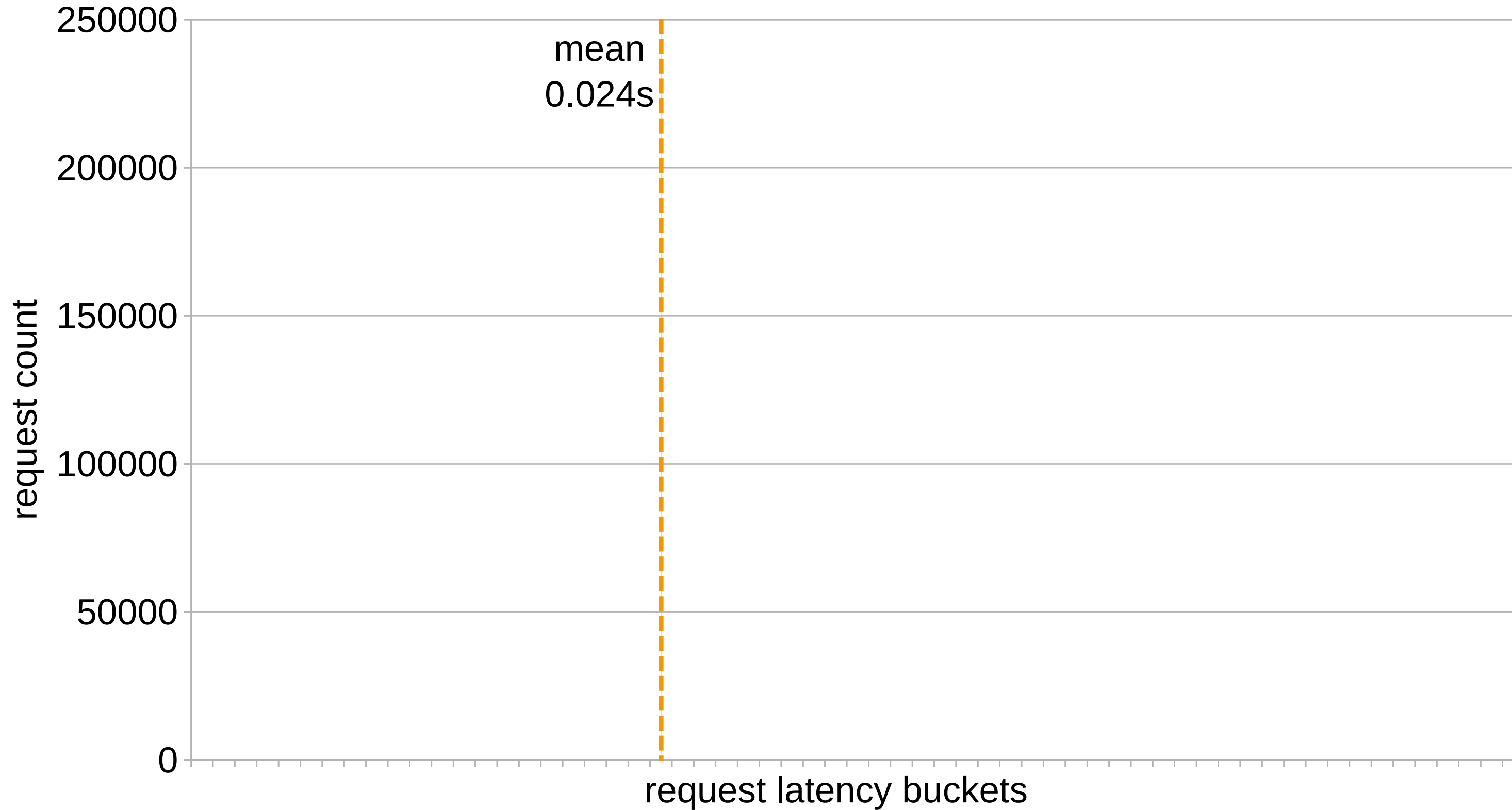
# Spring Boot metrics: it's (almost) free!

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

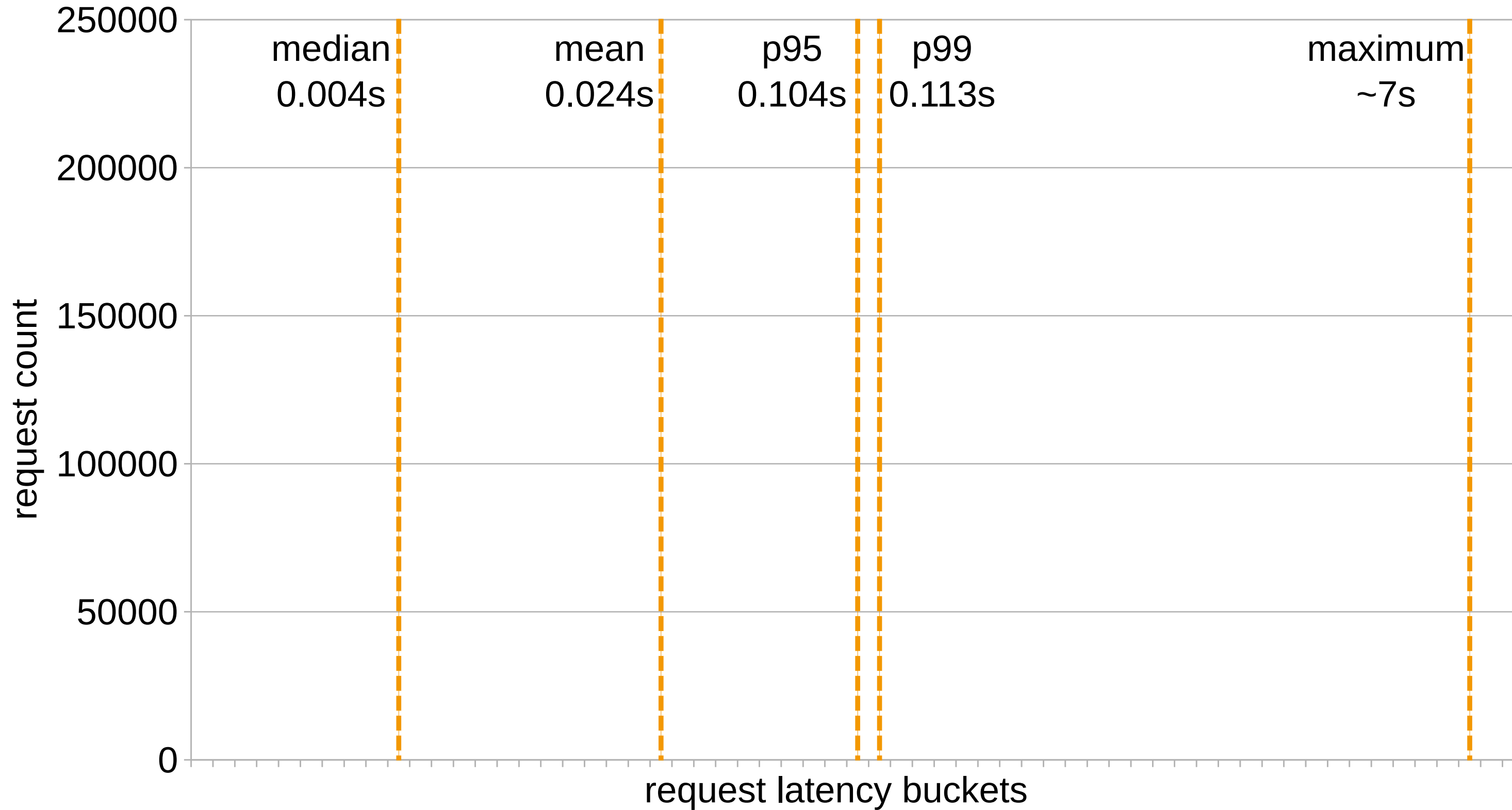
```
management.metrics:  
  distribution:  
    percentiles-histogram.http.server.requests: true  
    minimum-expected-value.http.server.requests: 1ms  
    maximum-expected-value.http.server.requests: 10s  
    slo.http.server.requests: 100ms  
web.server.request:  
  autotime.enabled: true  
  metric-name: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

⚠ *pseudo* application.yaml

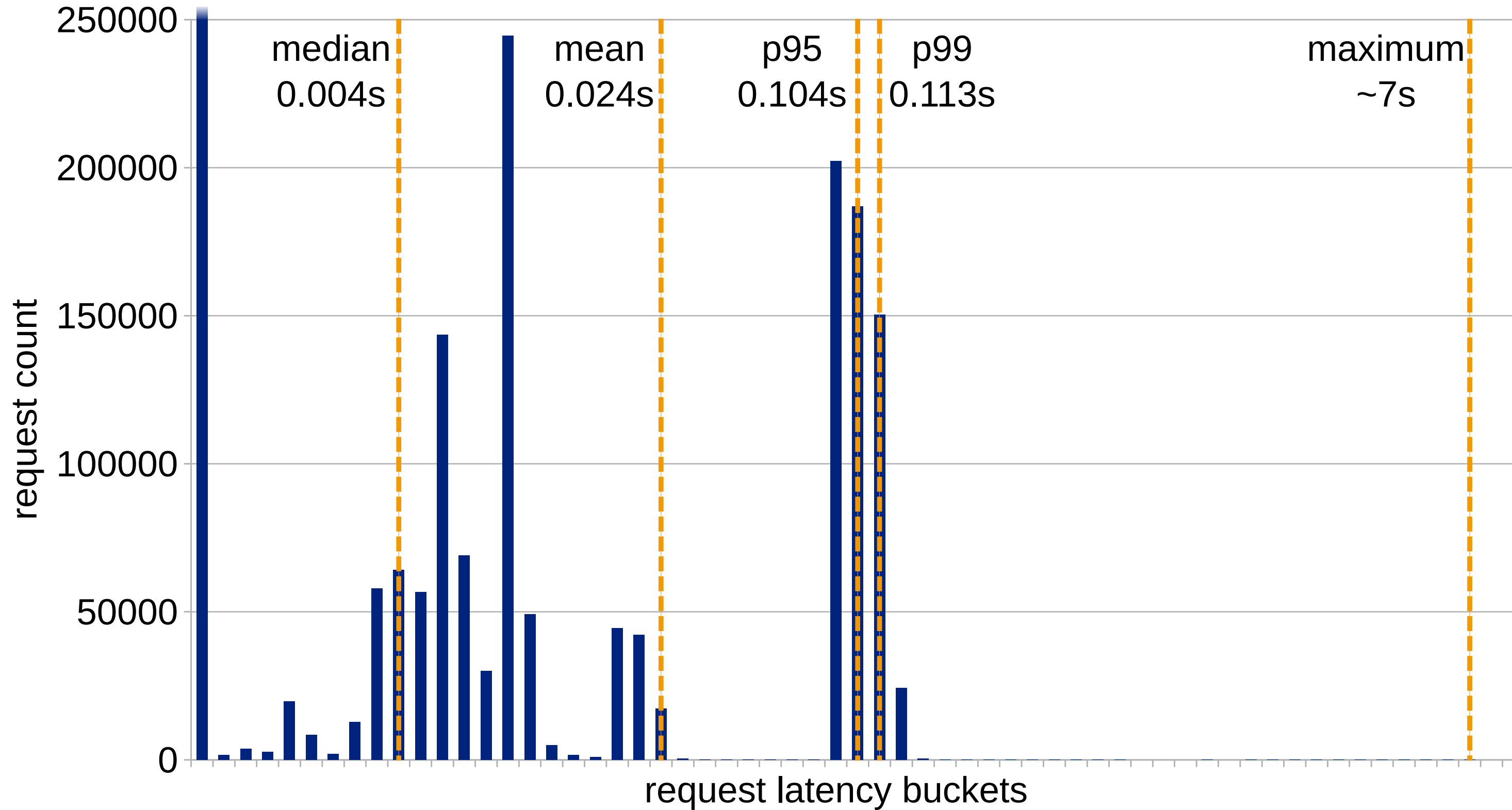
# Histograms for request latency



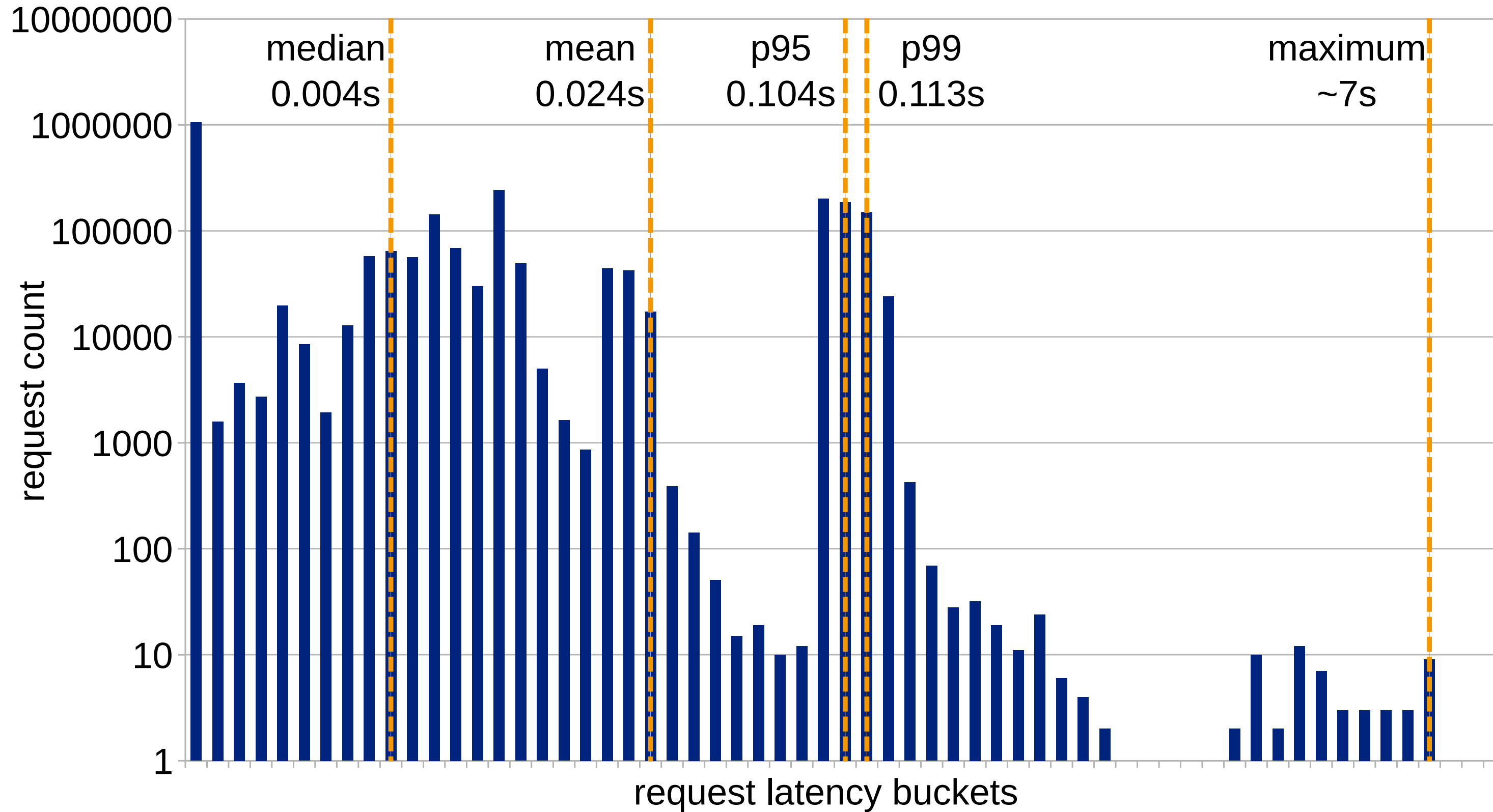
# Histograms for request latency



# 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

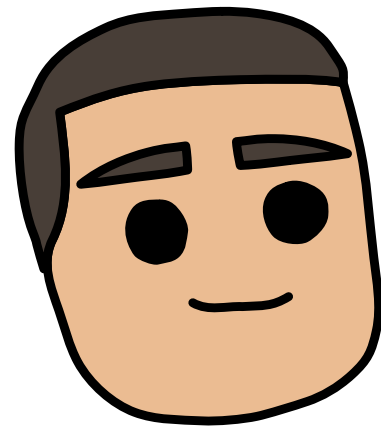
```
MeterRegistry registry;
Counter counter = registry.counter( "interestingOperation" );

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

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

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

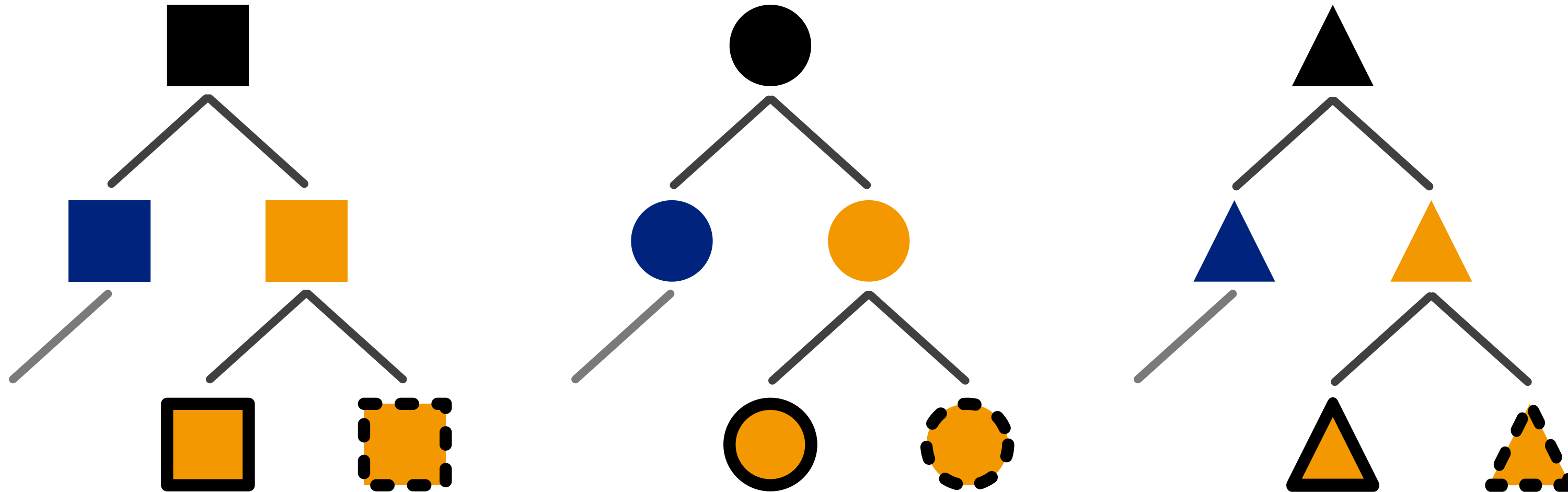
**Not bad,  
but not very  
useful in there...**



**Let's  
collect!!**

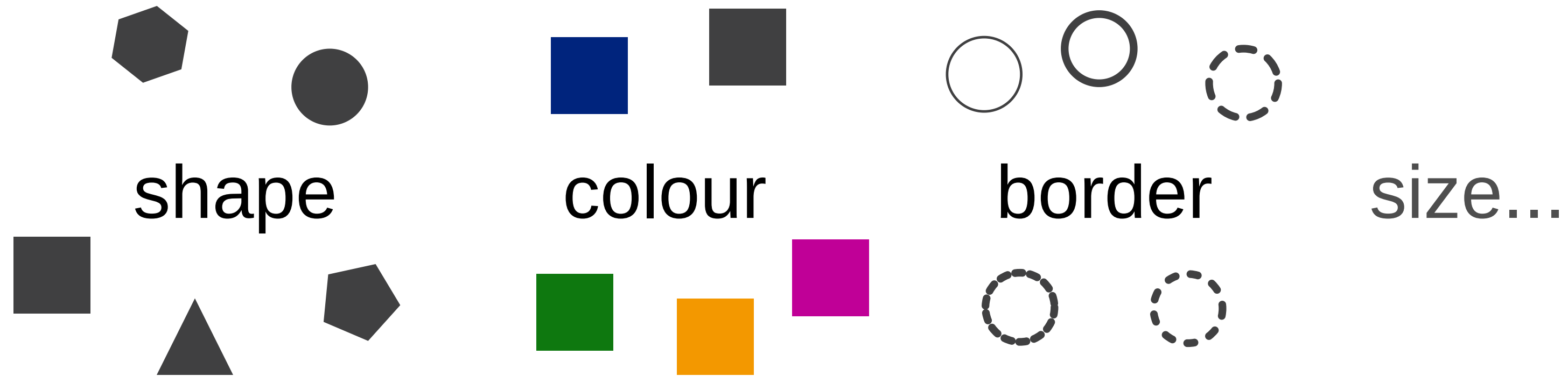


# Organising metrics: hierarchical system



```
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



**Here,  
try this!**



# 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

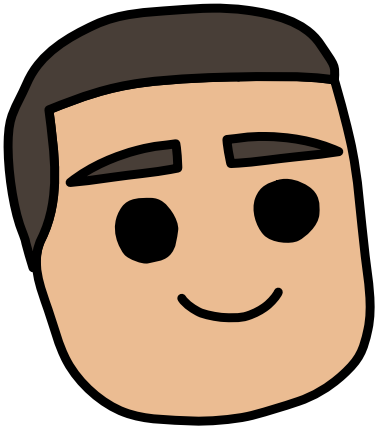
```
http.server.requests{method="get",status="200",app="example-app"}  
http.server.requests{method="get",status="404",app="example-app"}  
http.server.requests{method="post",status="201",app="example-app"}
```

✦ 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?**

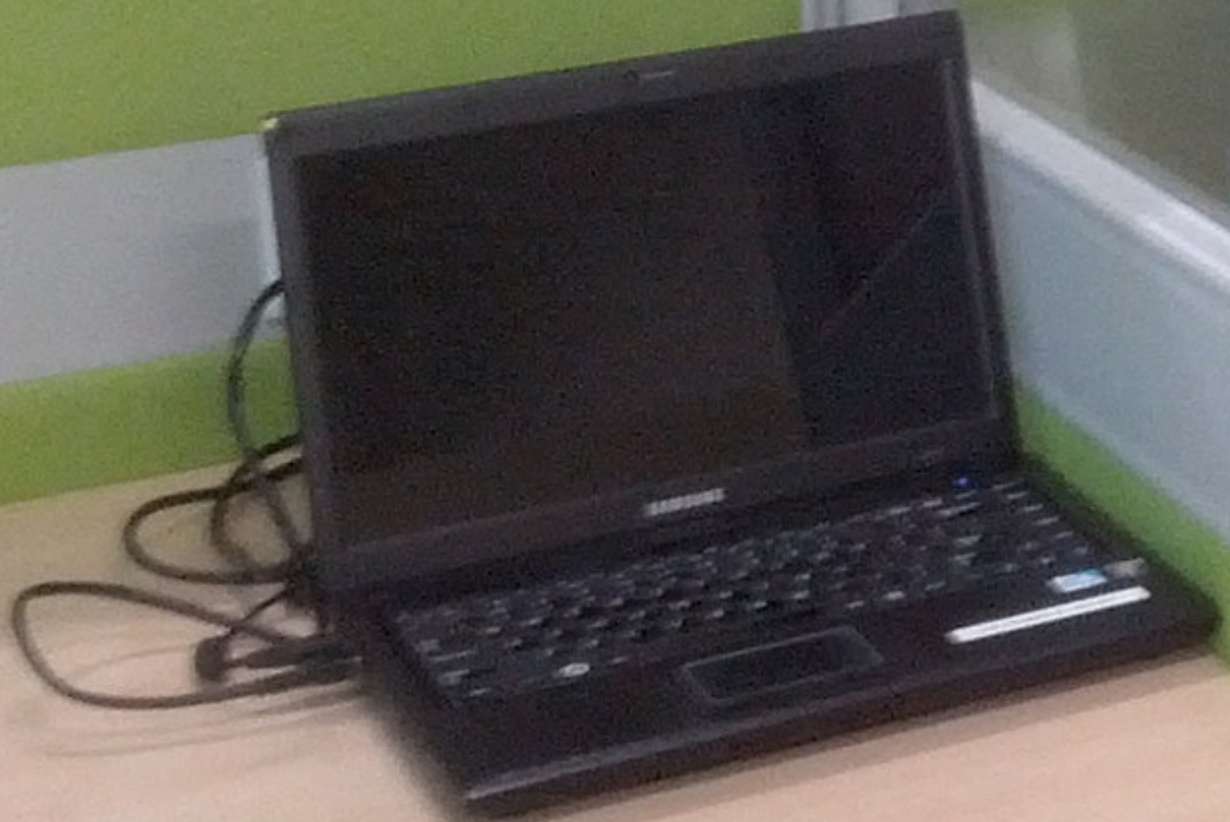
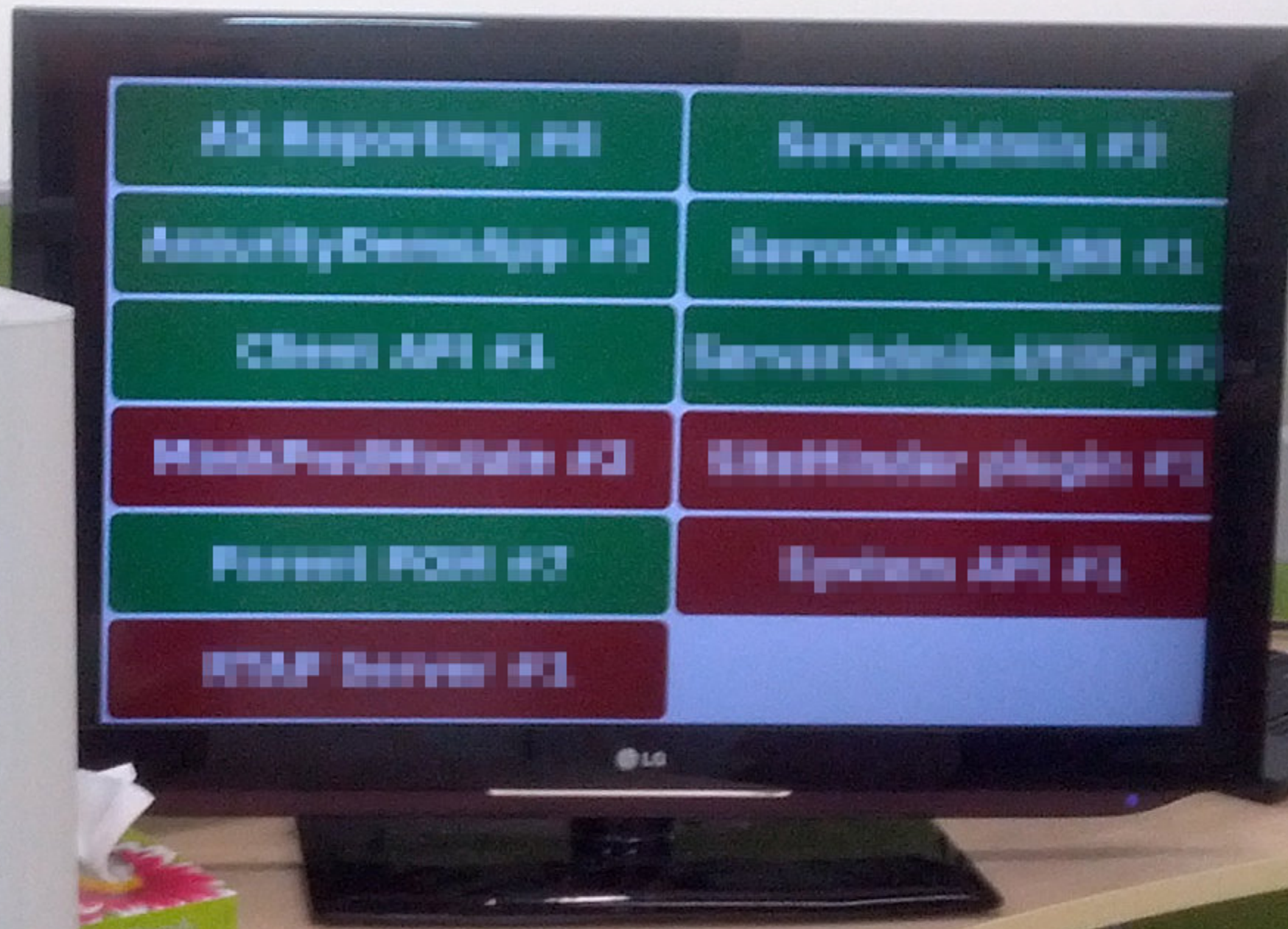


**Showtime!**

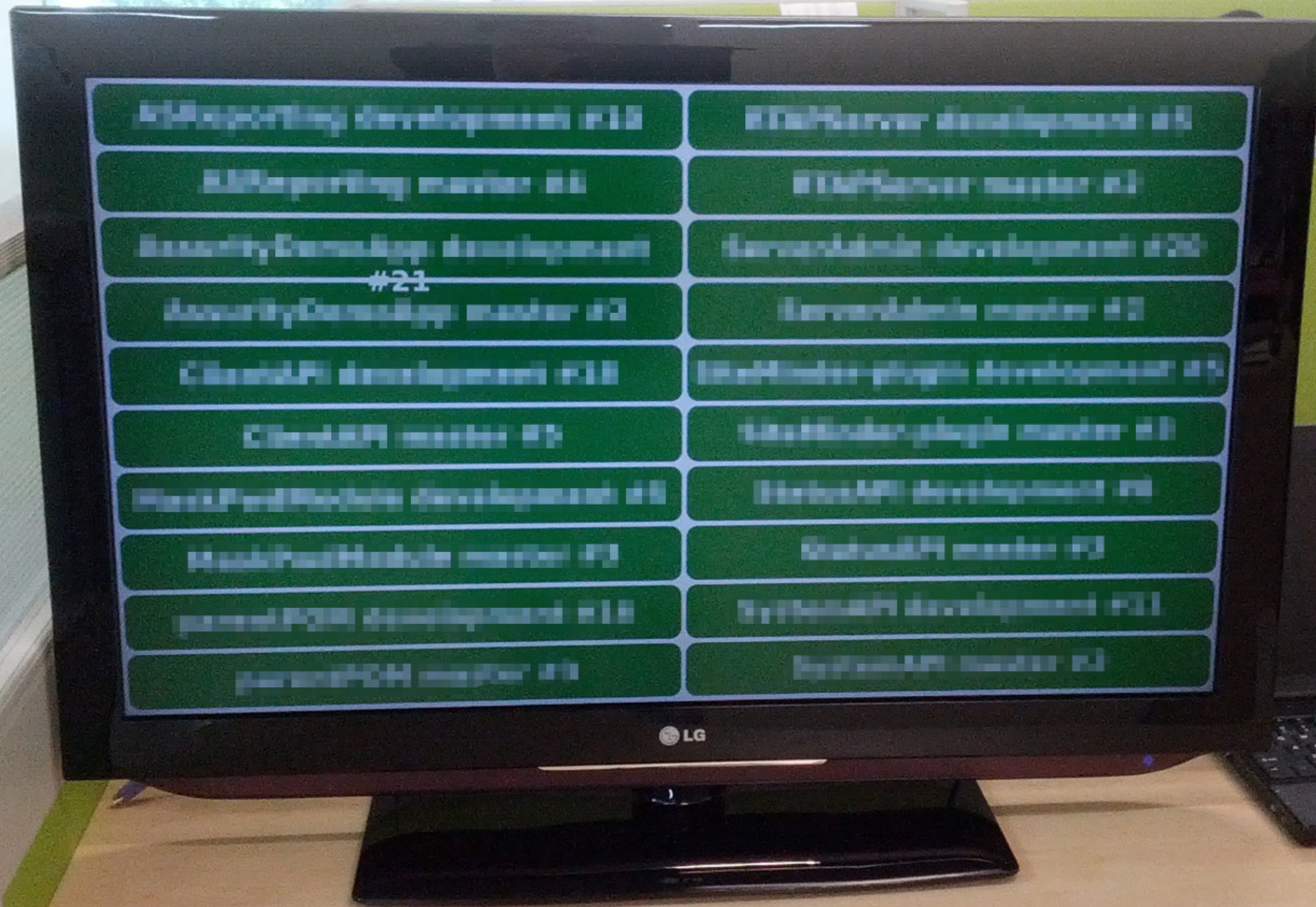
# Sharing data efficiently

Fernverkehr			S-Bahn			S-Bahn		
Nach	Gleis	Hinweis	Nach	Gleis	Hinweis	Nach	Gleis	Hinweis
<b>IR36</b> 08.52 Oerlikon Flughafen →	34		<b>S7</b> 08.49 Hardbrücke Oerlikon Winterthur	42		<b>S11</b> 09.01 Stadelhofen Winterthur Seuzach	43/44	
<b>IR17</b> 08.53 Altstetten Olten Burgdorf Bern	18		<b>S14</b> 08.49 Altstetten Urdorf Affoltern a/A	31		<b>S3</b> 09.04 Stadelhofen Stettbach Wetzikon	43/44	
<b>IC3</b> 09.00 Basel SBB	15		<b>S19</b> 08.49 Oerlikon Wallisellen Dietlikon Effretikon	33		<b>S10</b> 09.05 Selnau Binz Triemli Uetliberg	22	
<b>IC8</b> 09.02 Bern Thun Spiez Visp Brig	32		<b>S24</b> 08.51 Wiedikon Enge Wollishofen Zug	4		<b>S8</b> 09.07 Wiedikon Enge Pfäffikon SZ	31	
<b>IC5</b> 09.03 Aarau Olten Genève-Aéroport →	12		<b>S15</b> 08.52 Hardbrücke Oerlikon Niederweningen	41/42		<b>S9</b> 09.07 Hardbrücke Oerlikon Schaffhausen	41/42	
<b>IR70</b> 09.04 Thalwil Zug Luzern	6		<b>S5</b> 08.54 Stadelhofen Uster Pfäffikon SZ	43/44		<b>S5</b> 09.09 Hardbrücke Altstetten Urdorf Zug	41/42	
<b>IC8</b> 09.05 Flughafen → Winterthur Romanshorn	34		<b>S10</b> 08.55 Selnau Binz Friesenberg Triemli	22		<b>S15</b> 09.09 Stadelhofen Uster Rapperswil	43/44	
<b>RE</b> 09.05 Oerlikon Bülach Schaffhausen	5		<b>S8</b> 08.55 Oerlikon Wallisellen Winterthur	33		<b>S7</b> 09.11 Stadelhofen Meilen Rapperswil	43/44	
<b>IR16</b> 09.06 Baden Brugg Aarau Olten Bern	16		<b>S3</b> 08.56 Hardbrücke	41/42		<b>S19</b> 09.11 Altstetten Dietlikon	31	
<b>IR37</b> 09.08 Lenzburg Aarau Liestal Basel SBB	14		<b>S4</b> 08.58 Selnau Adliswil Langnau-G.	21		<b>S14</b> 09.12 Oerlikon Wallisellen Dübendorf Hinwil	33	
<b>IR13</b> 09.09 Oerlikon Flughafen → St. Gallen	13		<b>S9</b> 08.58 Stadelhofen Stettbach Uster	43/44		<b>S2</b> 09.14 Oerlikon Flughafen →	34	
<b>EC</b> 09.10 Zug Bellinzona Milano C. Venezia S.L.	7		<b>S11</b> 08.59 Hardbrücke Altstetten Schlieren Aarau	41/42		<b>S12</b> 09.14 Hardbrücke Altstetten Schlieren Brugg	41/42	
<b>IR36</b> 09.10 Altstetten Dietlikon Baden Basel SBB	32		<b>S6</b> 09.00 Stadelhofen Tiefenbrunnen Uetikon	43/44		<b>S24</b> 09.14 Wipkingen Flughafen → Weinfelden	3	
<b>RE</b> 09.12 Thalwil Wädenswil Landquart Chur	9		<b>S6</b> 09.01 Hardbrücke Oerlikon Seebach Baden	41/42		<b>S16</b> 09.15 Stadelhofen Herrliberg-F.	43/44	
<b>IC5</b> 09.30 Olten Solothurn Biel/Bienne Lausanne	32		Fahrplanänderung S6: Zürich Tiefenbrunnen - Zürich Oerlikon. Nacht 24.02./25.02.20 bis Nacht 27.02./28.02.20 jeweils von 22:10 Uhr bis 00:35 Uhr. Prüfen Sie Ihre Verbindung im Online-Fahrplan.			<b>S12</b> 09.16 Stadelhofen Winterthur Schaffhausen	43/44	
<b>IC1</b> 09.32 Bern Lausanne Genève-Aéroport →	31					<b>S16</b> 09.16 Hardbrücke Oerlikon Flughafen →	41/42	
<b>IC2</b> 09.32 Zug Rotkreuz Bellinzona Lugano	7					<b>S2</b> 09.17 Wiedikon Enge Thalwil Ziegelbrücke	32	
<b>IC1</b> 09.33 Flughafen → Winterthur St. Gallen	33					<b>S4</b> 09.18 Selnau Adliswil Langnau-G. Sihlwald	21	
						<b>S7</b> 09.19 Hardbrücke Oerlikon Winterthur	41/42	

# Dashboards are powerful



# Dashboards are powerful



# Sharing data efficiently



# Quick win example

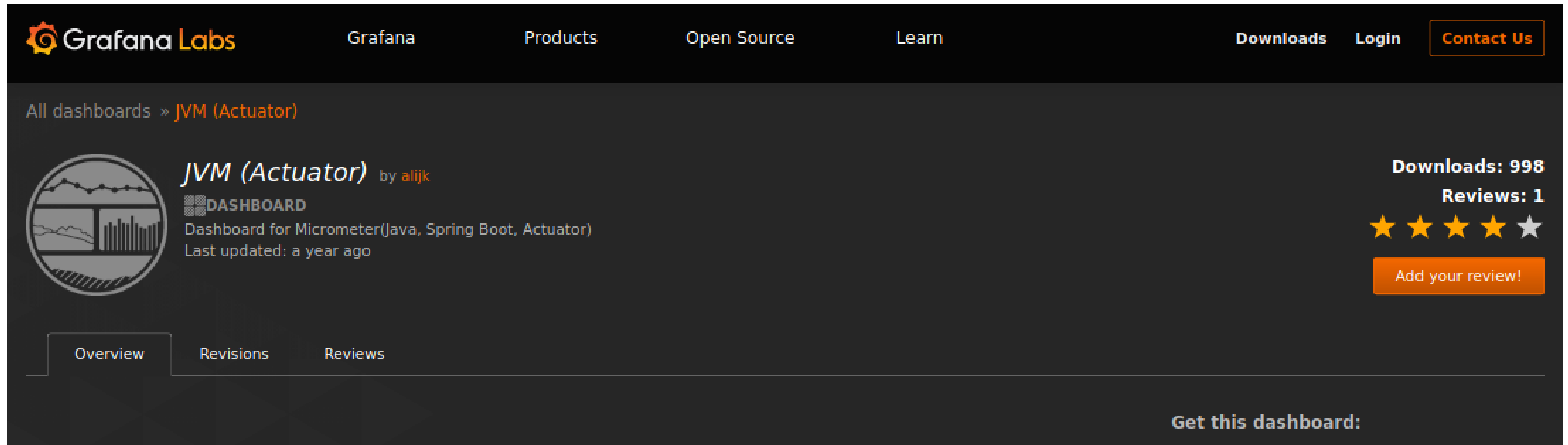
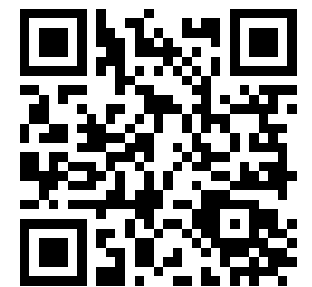


Here,  
try this!



# Nice-looking dashboards with Grafana

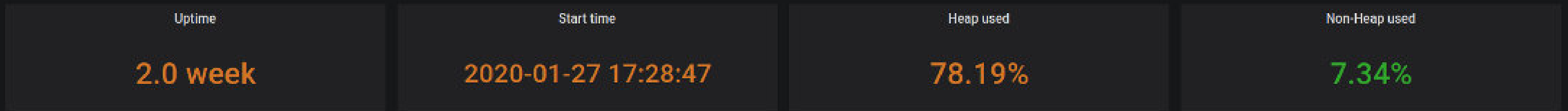
- Use the official Docker image grafana/grafana, default theme
- Configure Grafana to read from Prometheus
- Community dashboard for Spring Boot applications  
<https://grafana.com/grafana/dashboards/9568>  
<https://grafana.com/grafana/dashboards/4701>



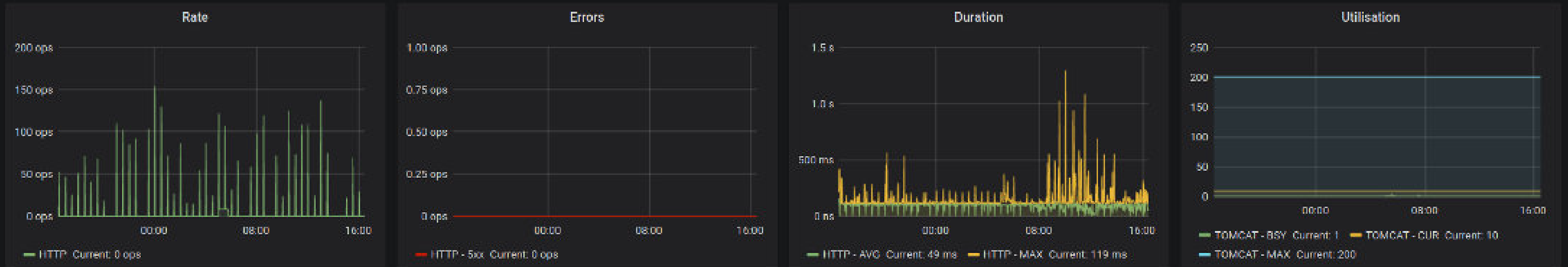
The screenshot shows the Grafana Labs website interface. At the top, there is a navigation bar with the Grafana Labs logo and links for Grafana, Products, Open Source, Learn, Downloads, Login, and a Contact Us button. Below the navigation bar, the breadcrumb trail reads "All dashboards » JVM (Actuator)". The main content area features a dashboard card for "JVM (Actuator)" by "alijk". The card includes a circular icon with a line and bar chart, the title "JVM (Actuator)", the author "alijk", and a "DASHBOARD" label. Below the title, it says "Dashboard for Micrometer (Java, Spring Boot, Actuator)" and "Last updated: a year ago". To the right of the card, the statistics are "Downloads: 998" and "Reviews: 1", accompanied by a five-star rating system where the first four stars are filled and the fifth is empty. Below the stars is an orange button that says "Add your review!". At the bottom of the card, there are three tabs: "Overview" (which is active), "Revisions", and "Reviews". At the very bottom of the page, there is a section titled "Get this dashboard:".



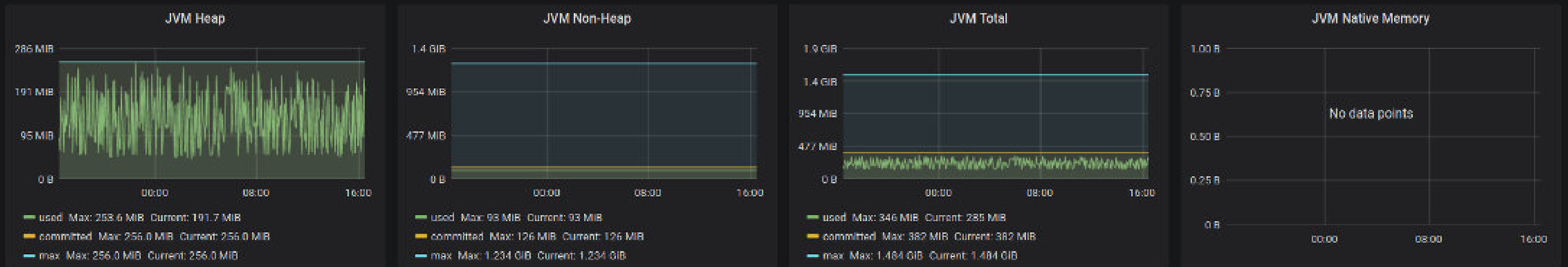
Quick Facts



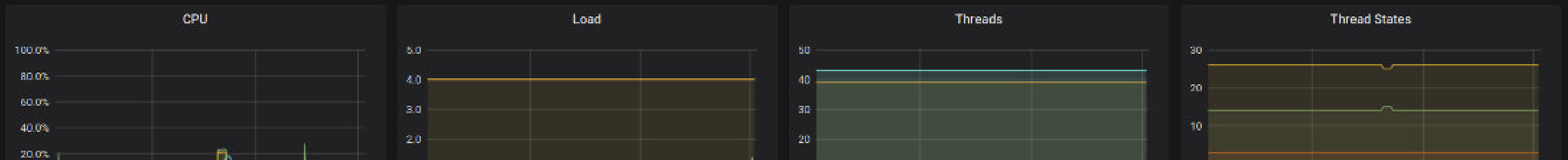
I/O Overview



JVM Memory

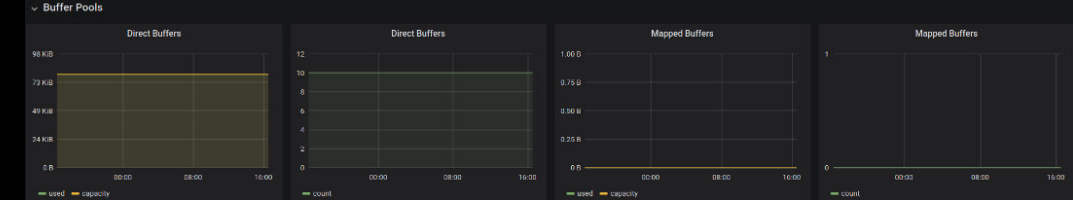
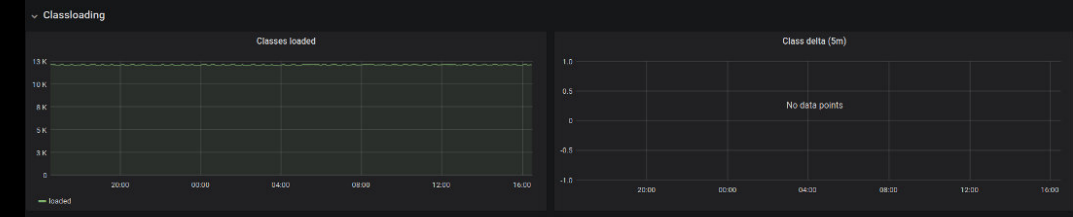
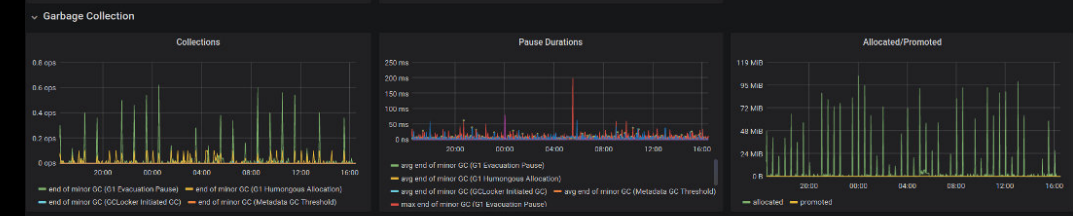
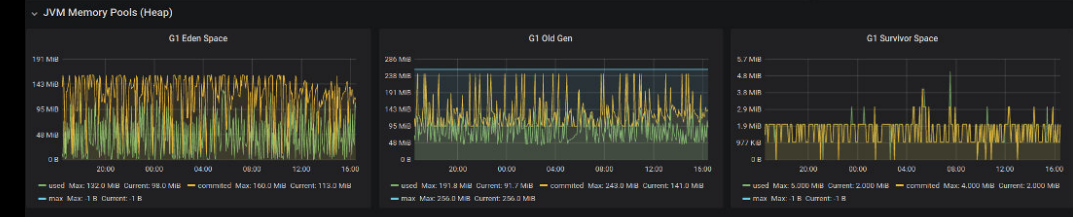
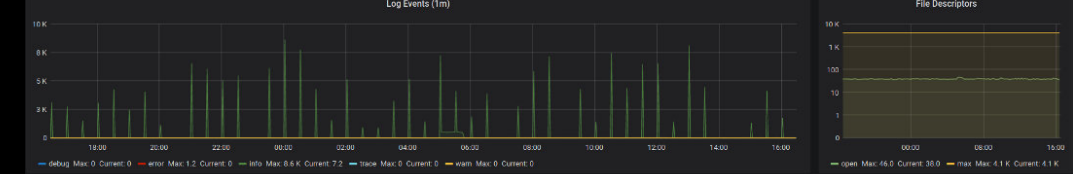
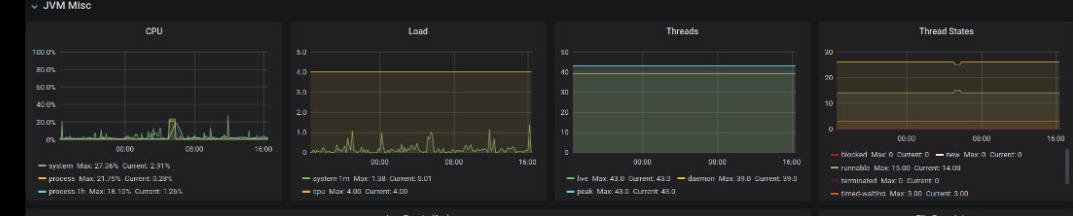
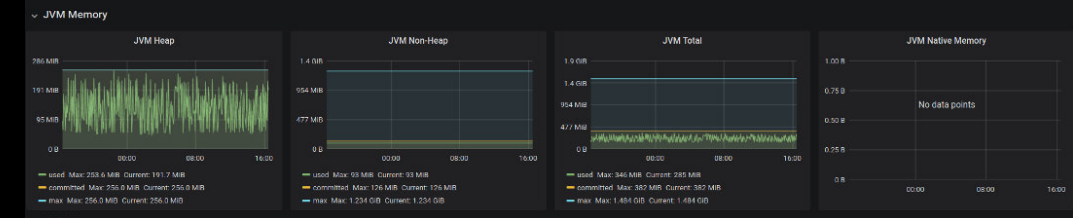
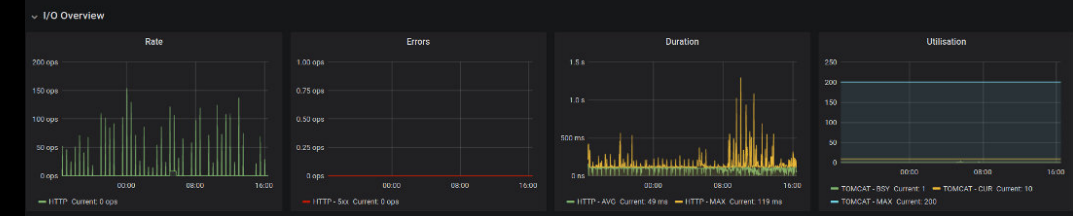


JVM Misc



**Quick Facts**

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



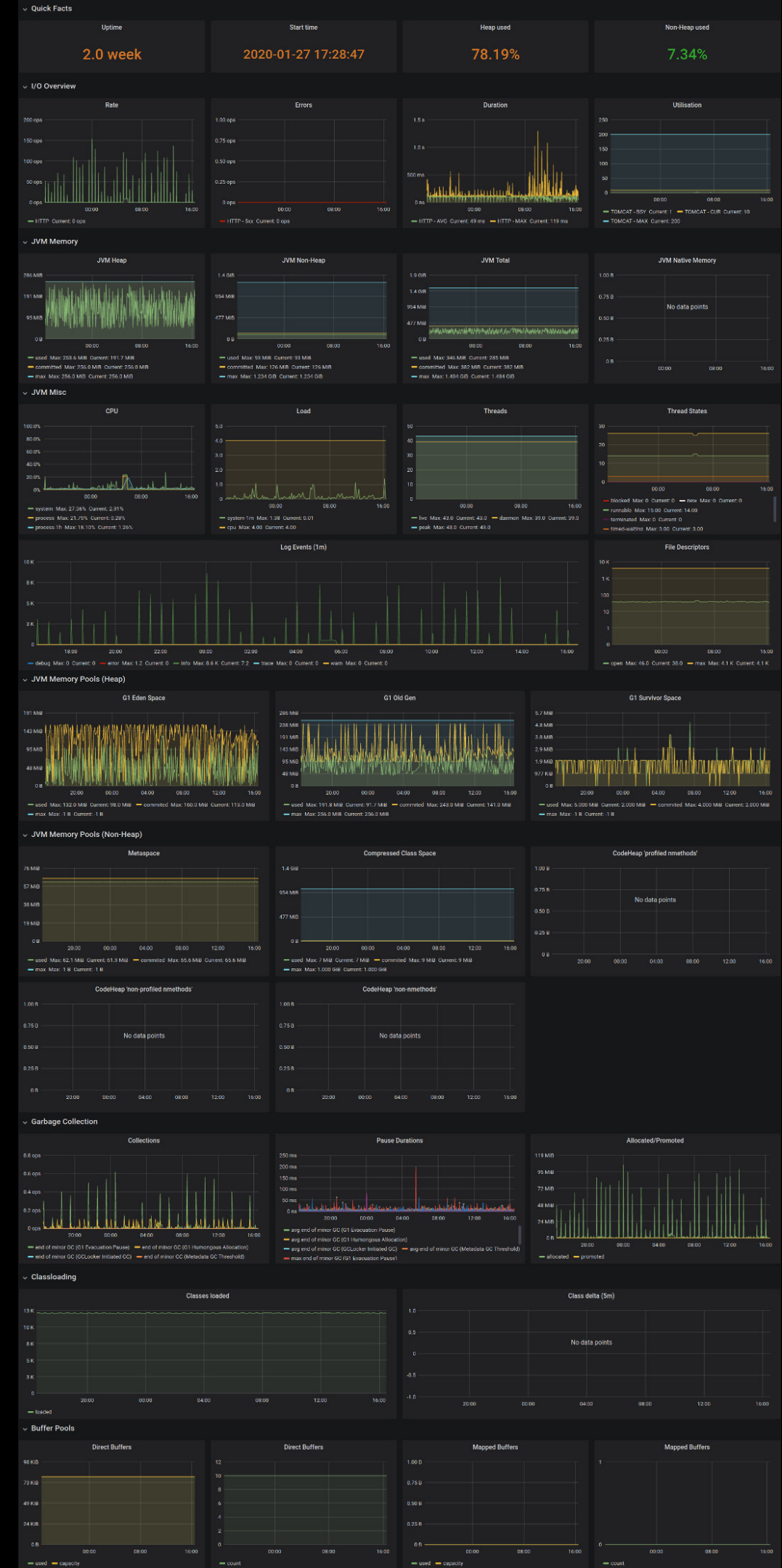
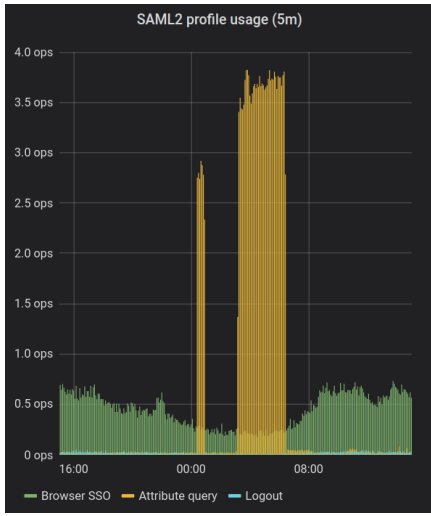


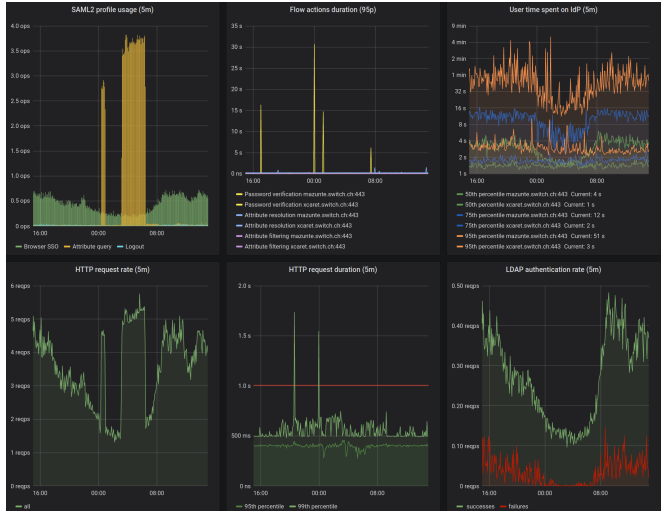
Photo by Shandell Venegas on Unsplash

# Trap: going overboard with dashboards

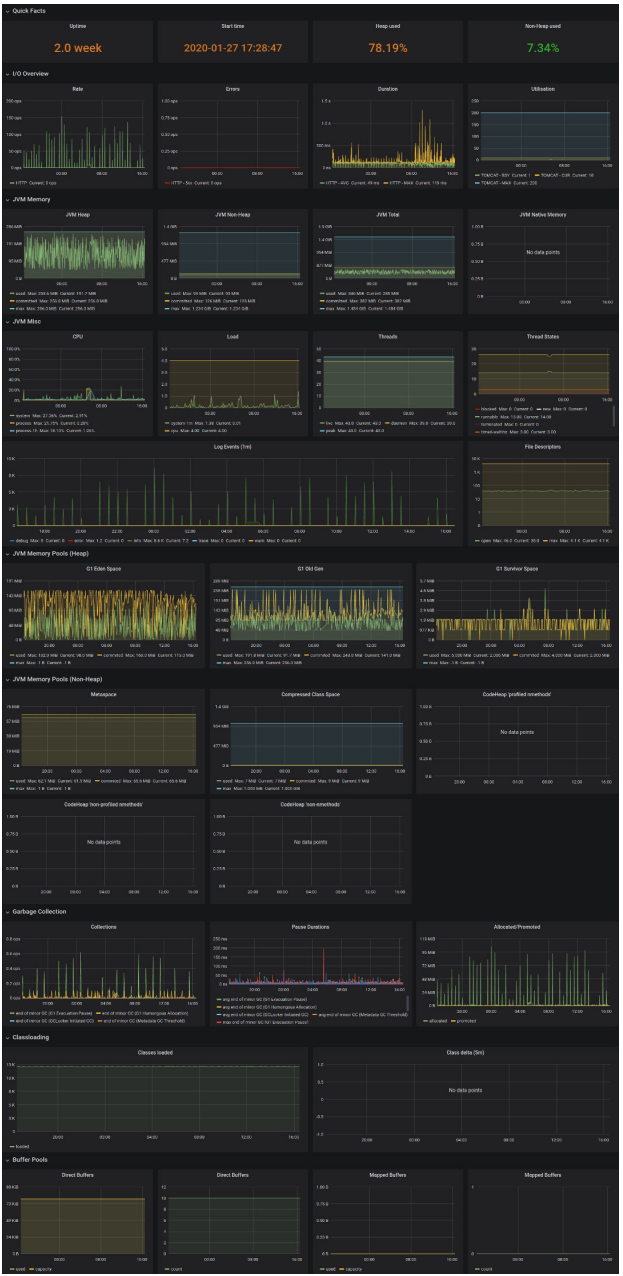
## business



## technical



## troubleshooting

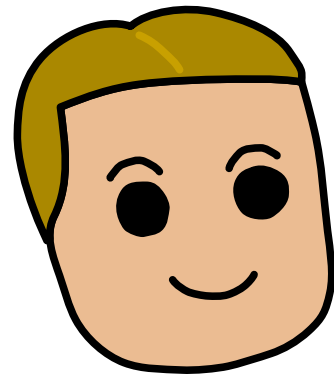


# Trap: going overboard with dashboards

- 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.



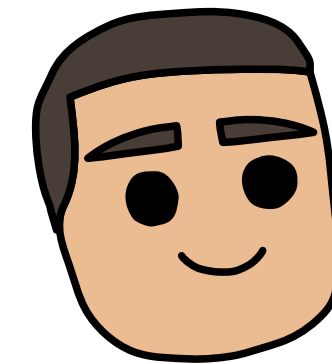
**business**

How do we prioritise new features?



**operations**

Where are the bottlenecks?

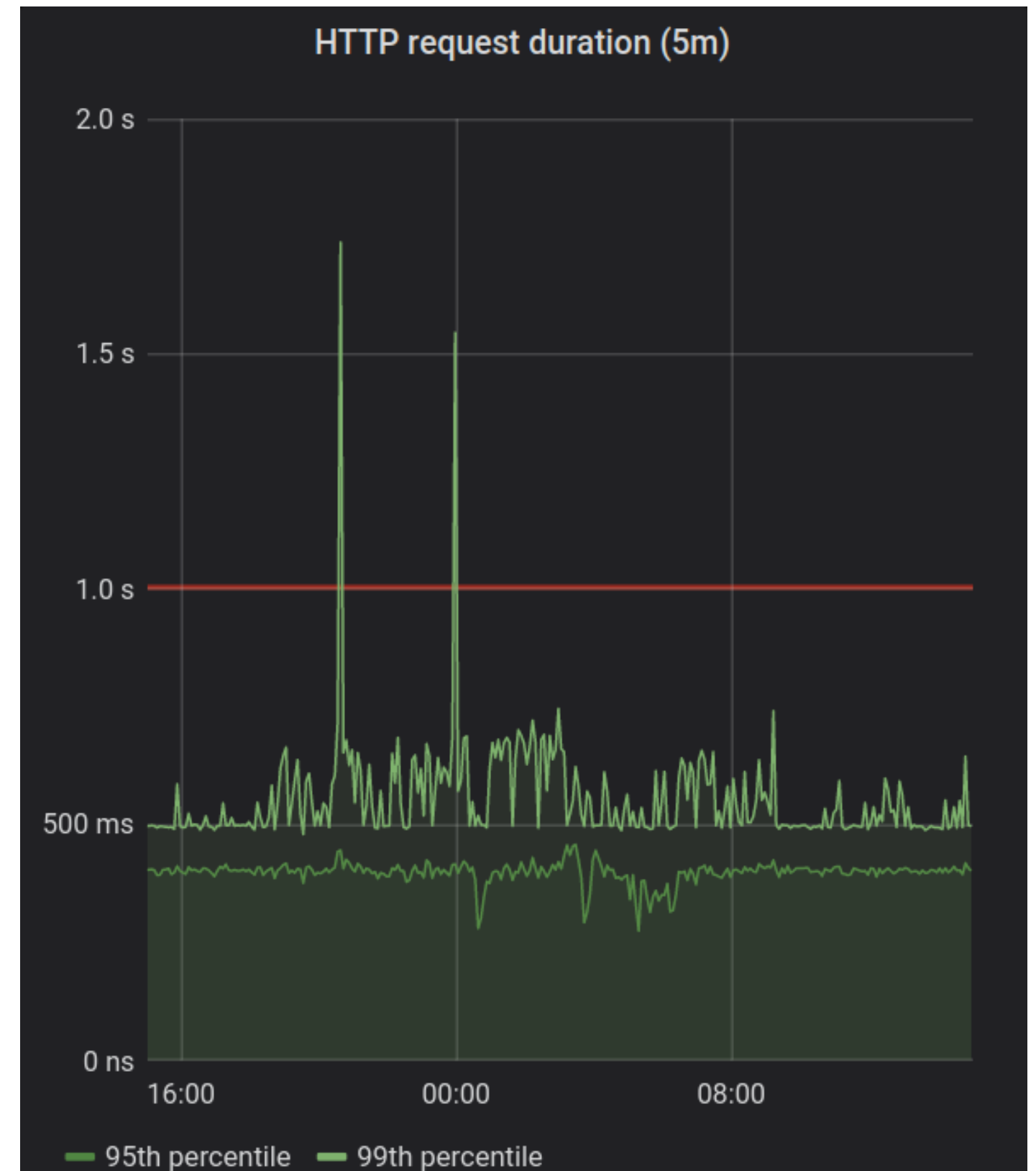


**developers**

Are technical specifications met?

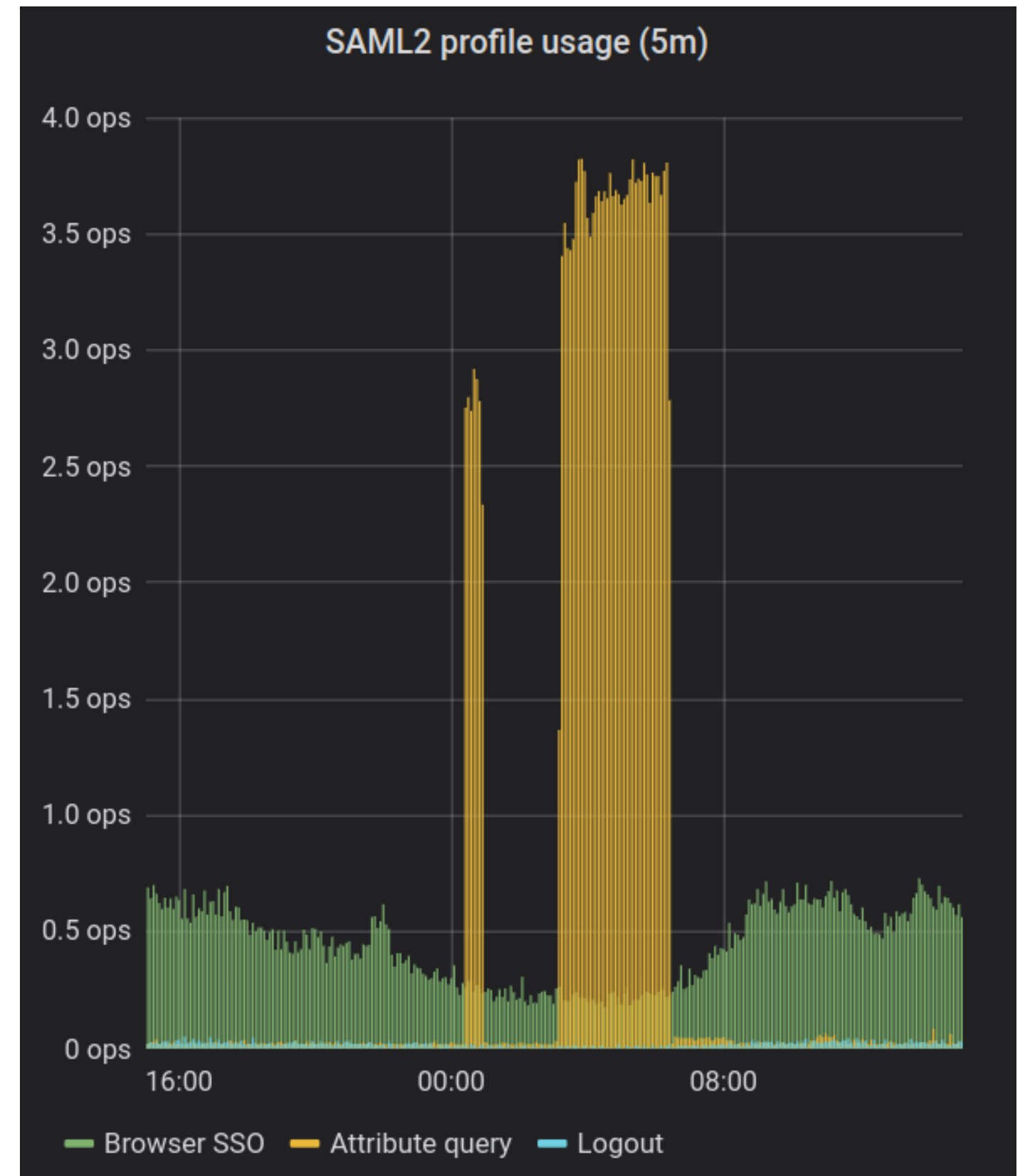
# Real-life example: developers / operations

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

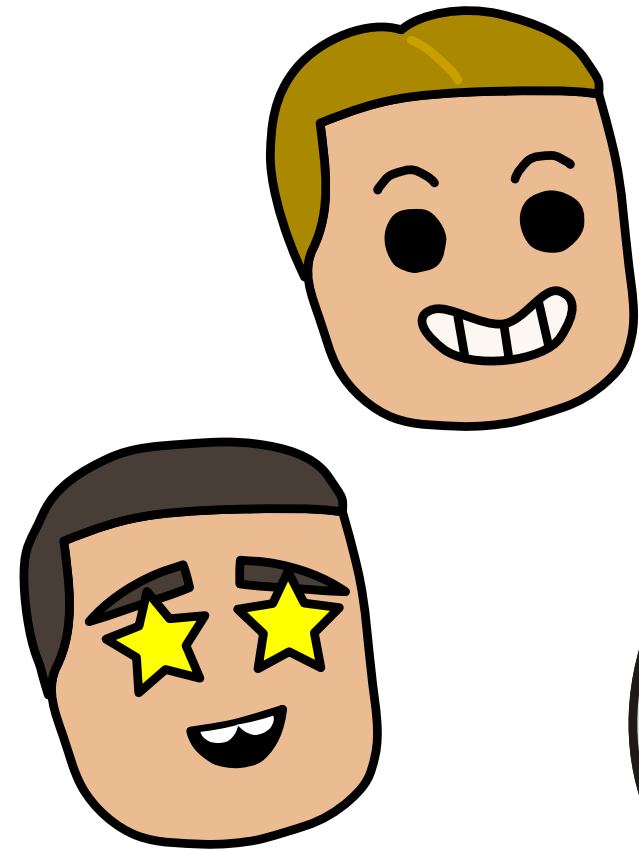


# Real-life example: business

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

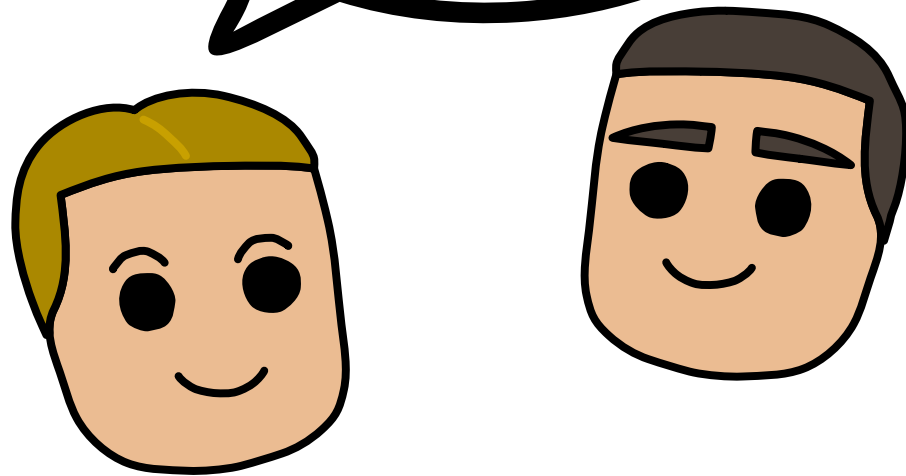






**Voilà!**

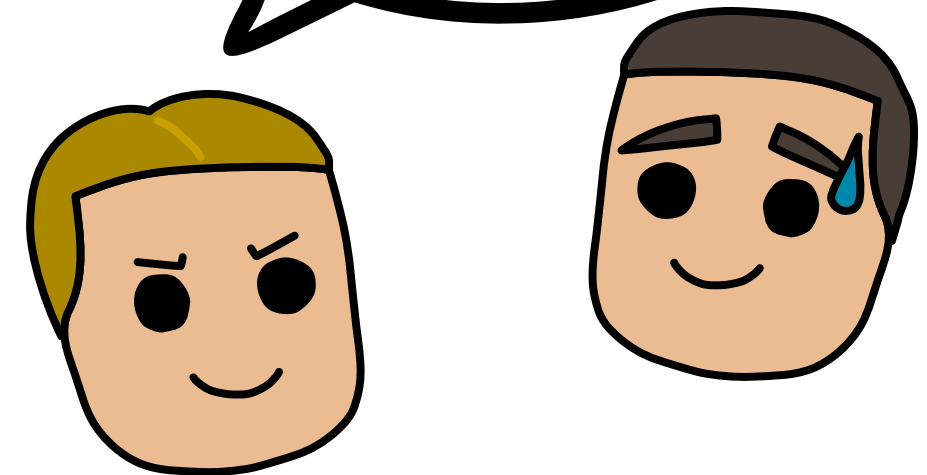
Hey Buddy!  
How's our app  
doing today?



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](mailto:etienne.dysli-metref@switch.ch)

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