Basler Versicherungen

bluesky CONSIDER IT DONE

KARAKUN

sympany

magnolia®

JAVA USER GROUP CH

ERNI

BLKB

twilio

ELCA We make it work.

Gradle

OPTRAVIS Transfer Pricing Solutions

Runtime Metriken 101

BaselOne, 19.10.22

Markus Schlichting

Stephan Classen

KARAKUN

KARAKUN

Wir entwickeln Software.

Agil. Nutzerzentriert. Erfolgreich.

Karakun im Detail

Nachhaltige Individuallösungen

Kunden aus unterschiedlichen Bereichen, u.a. Versicherung, Finanz, Life Science, Logistik

Kompetenzen

State-of-the-Art Tech-Stack (Java, Web)
Text Analytics / KI / Big Data
Fokus auf Open-Source-Software

Community Engagement

Autoren, Referenten, Java Champions, Universitätsdozenten, Kontributoren in Open-Source-Projekten

Dienstleistungen

Software Engineering, UX-Design, Consulting, Training, Wartung & Support

Plattformen & Produkte

Effizienzsteigernde Software-Plattformen, fertige Produkte für ausgewählte Bereiche

Erfahrenes, eingespieltes Team

60+ Mitarbeitende an 4 Standorten in CH (Hauptsitz), D und IN

Markus Schlichting

- Softwareengineer & -architect
- Co-Founder of Karakun
- Hackergarten Organizer
- OpenSource Enthusiast
- Family man
- Speaker

Markus.Schlichting@karakun.com
@madmas

Stephan Classen

- Softwaredeveloper for 14 years
- Co-Founder of Karakun
- SoCraTes CH Community Member
- OpenWebStart Committer
- Family man
- Speaker

Metrics 101

Poll

Show of hands?

Who is using logging in their application ??

Poll

Show of hands?

Who is using logging in their application ??

Almost everyone

Poll

Show of hands?

Who is using metrics in their application ??

Poll

Show of hands?

Who is using metrics in their application ??

Not as many

Two sides of the same medal

And both are shiny

Both logs and metrics are messages from within the application to inform an external observer about its run time behavior.

karakun

Two sides of the same medal

With a small difference

	Logs	Metrics
Meta Data	structured • time • level • logger	structured • time • name • tags
Data	unstructured • message	structured • value (numeric) • unit

Why do we need both?

Is there something we missed?

If metrics are almost the same as logs

why do we need both ??

Why do we need both?

Is there something we missed?

If metrics are almost the same as logs

why do we need both ??

$\longrightarrow$ Mathematics

Are metrics supperior to logs?

So we would not need both?

Metrics are primarily used to create statistics

Less suited for tracking a single request or event and correlate with other entries.

Who is interested in Metrics?

karakun

Interested parties

The short list

Operations

- Resource usage (cpu, memory, network, disc)
- Alerting in extreme cases
- Prediction about scaling

Interested parties

The short list

Developers

- Performance
- Bottlenecks
- Limitations

Interested parties

The short list

UX

- User behavior
- A/B Testing
- Drop-Off Points

Interested parties

The short list

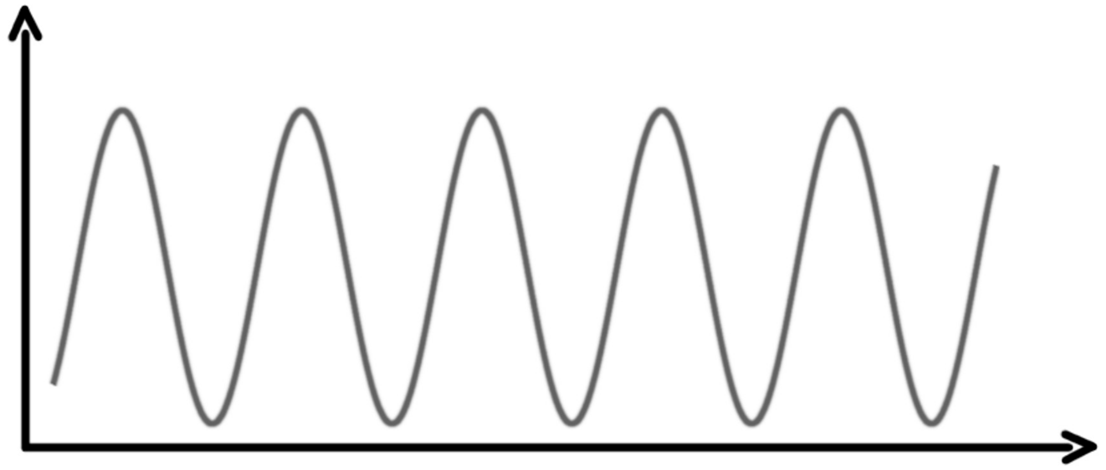
Management

- User count
- Conversion rate
- Retention

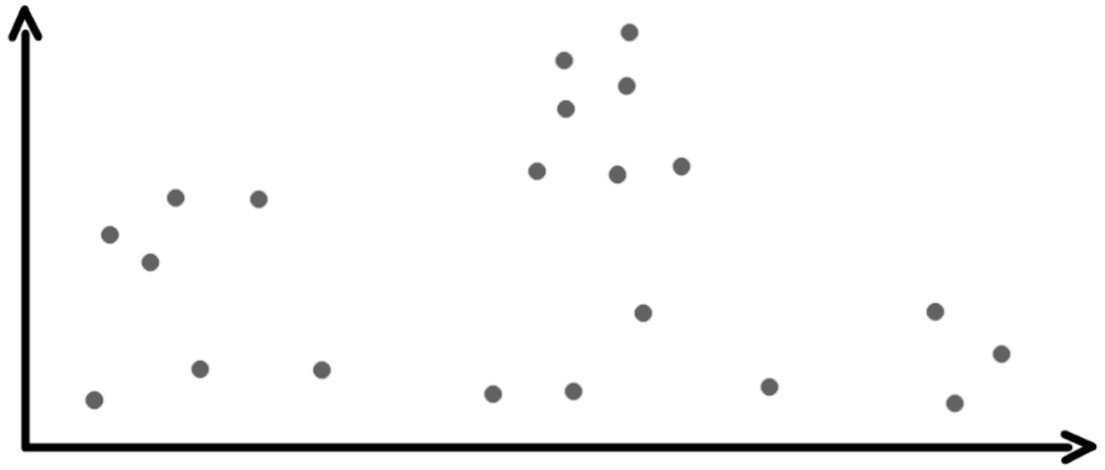
Metrics **basics**

Two types of sources

Both are very common



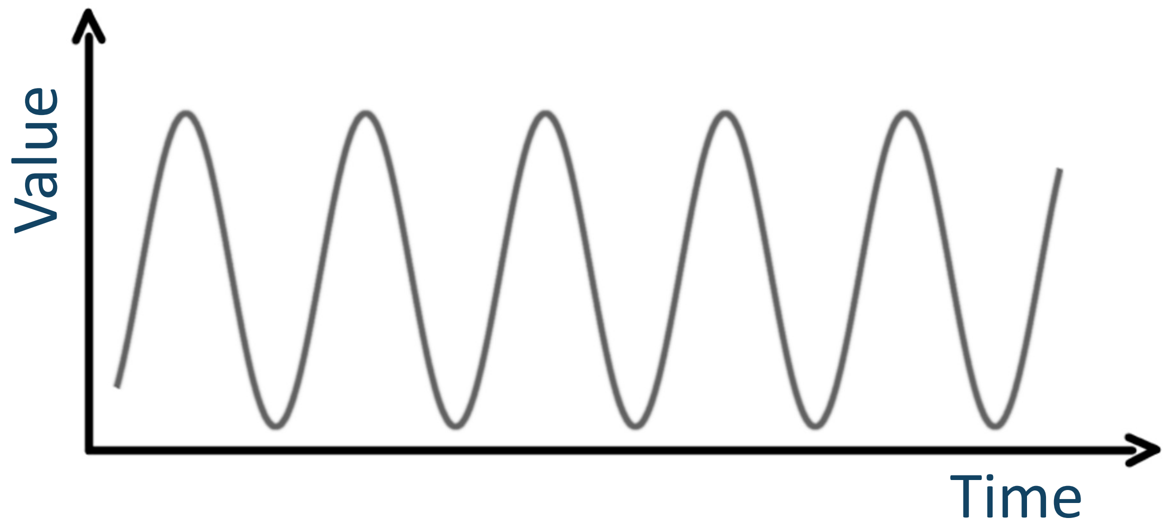
Continuous value



Events with value

Continous value

This is just like a gauge

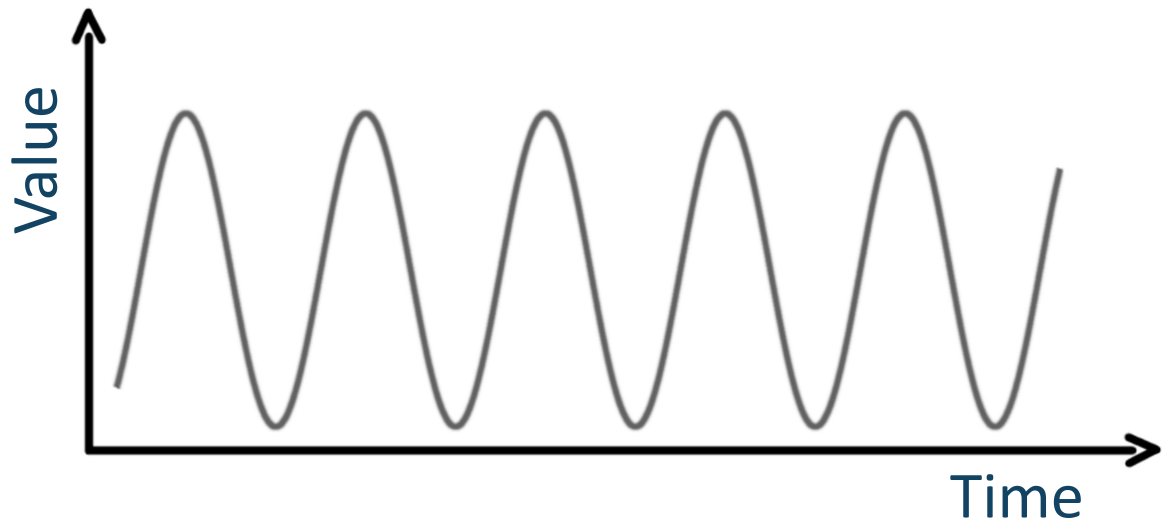


Sample the source in a regular interval

Gauge

Continous value

Sampling

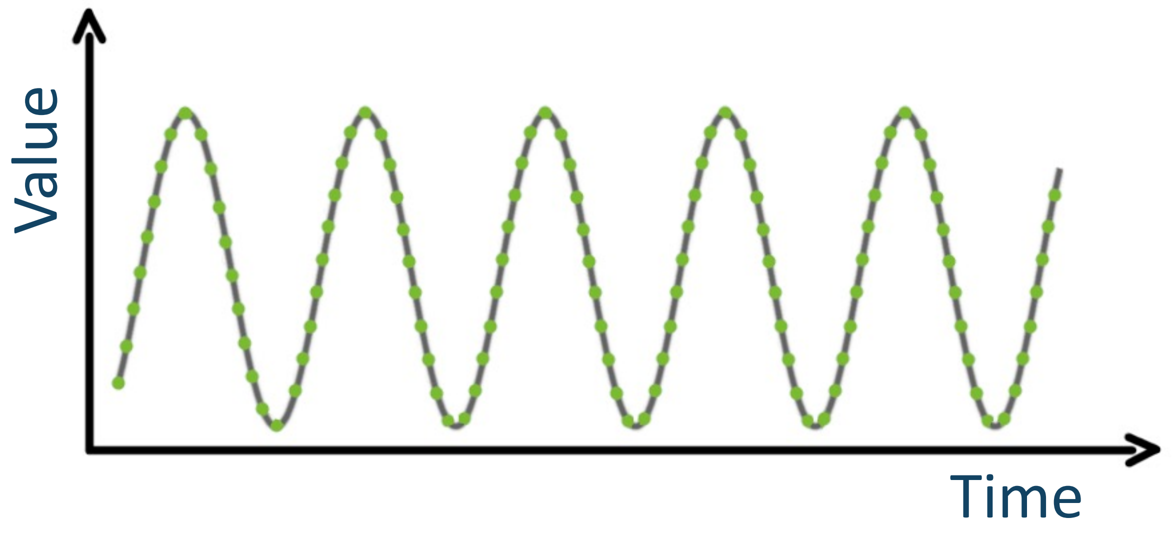


Sample the source in a regular interval

Chose sampling rate according to expected source feature

Continous value

Sampling

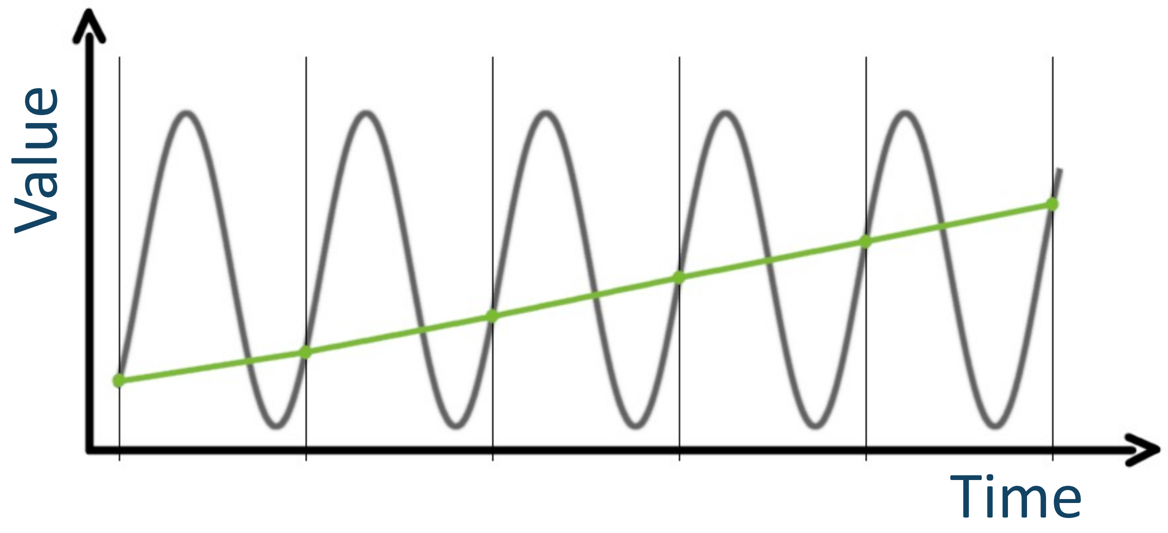


High sample rate

+ High accuracy
− Large amount of data
− High overhead for sampling and handling data

Continous value

Sampling

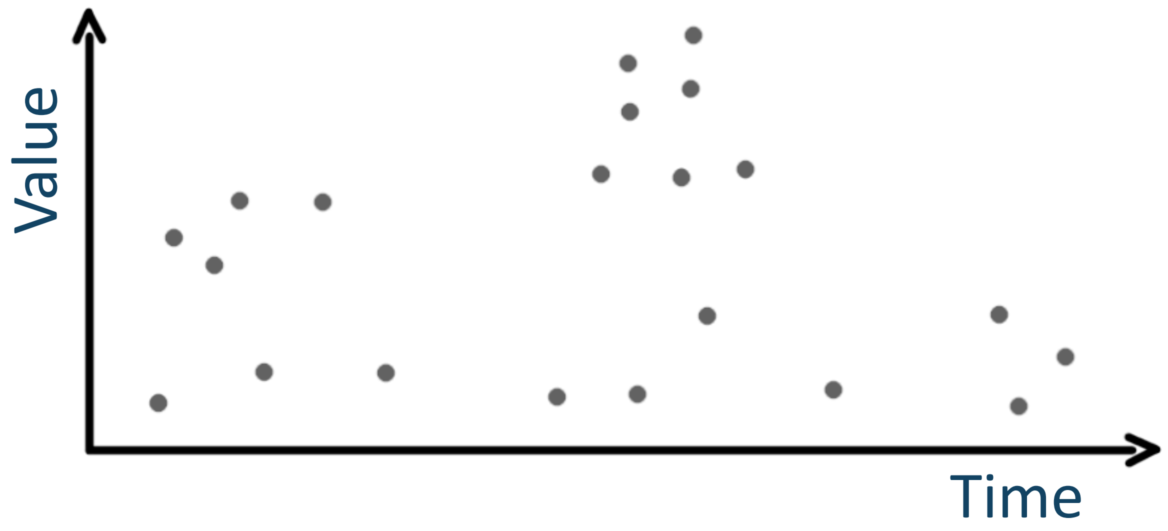


Low sample rate

- + Low overhead
- − Low accuracy
- − Important information may be lost

Events with value

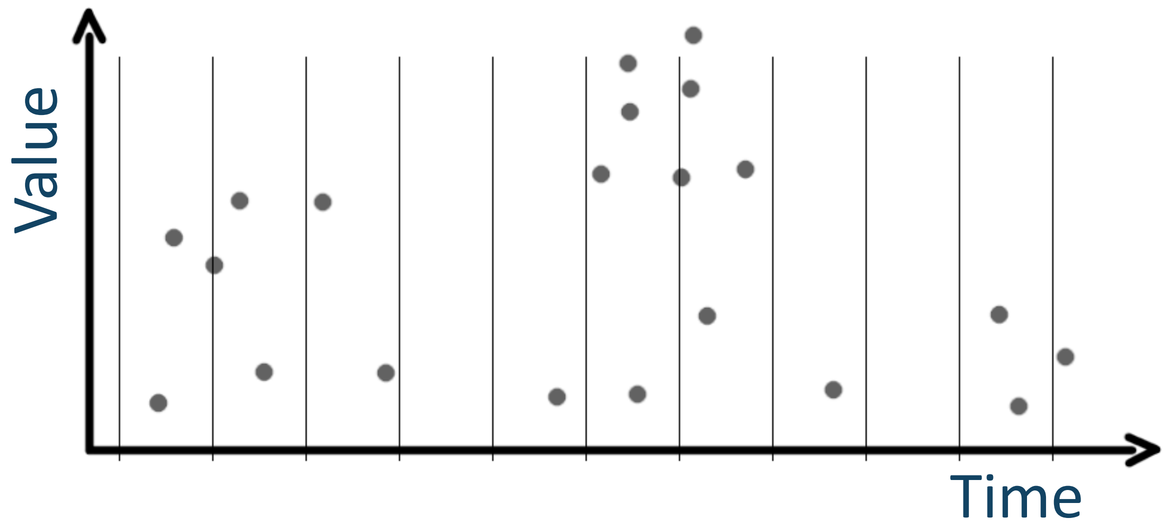
Happen sporadically



Most of the time the value is not defined

Events with value

Happen sporadically

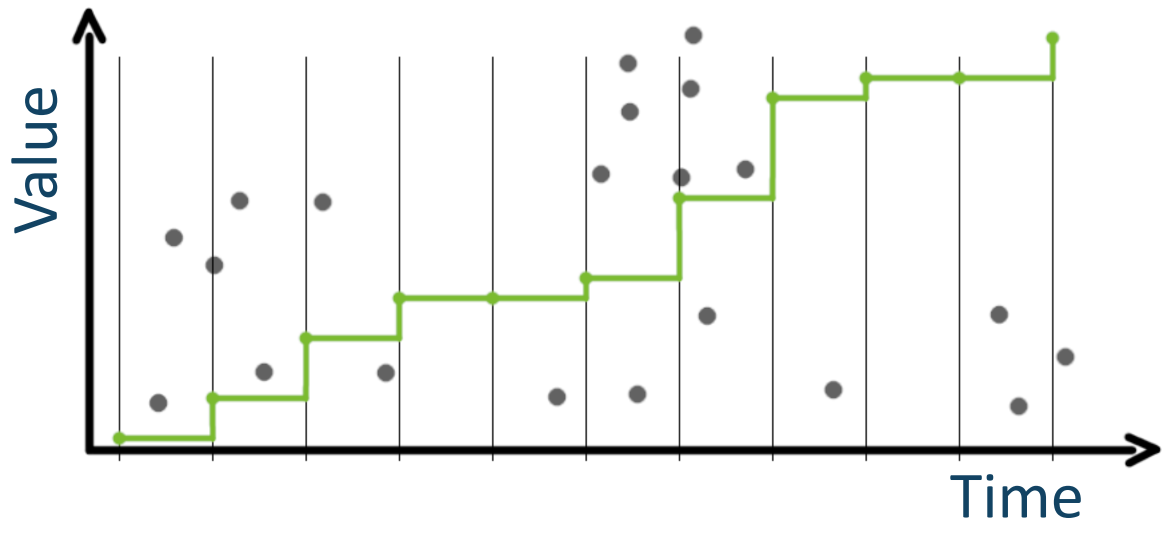


Most of the time the value is not defined

Aggregate events between two samples

Events with value

Sampling

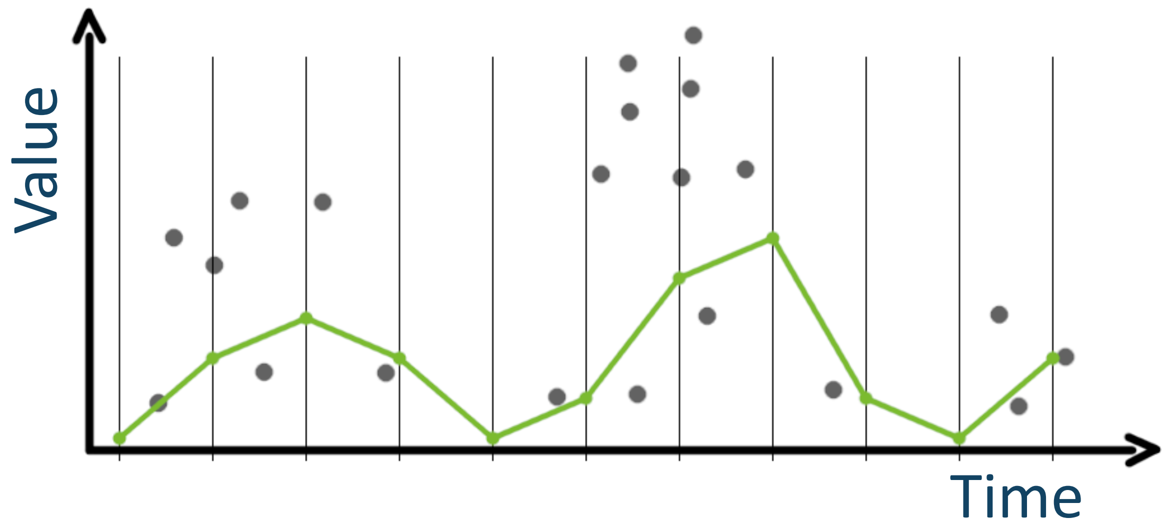


Count the total events

+ Very easy to do
− Not very meaningful
− Values are ignored

Events with value

Sampling

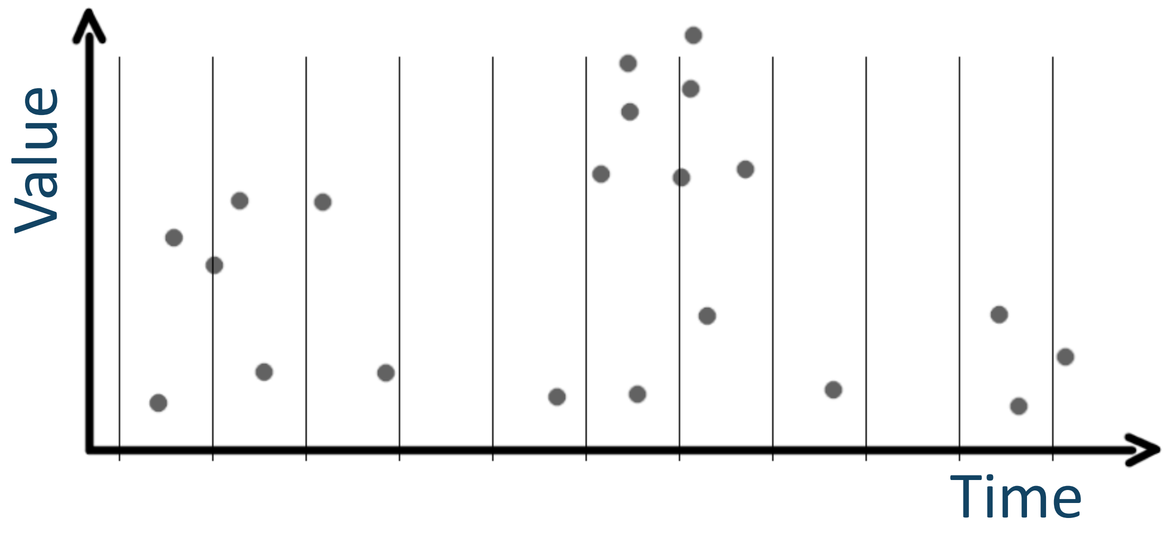


Count the delta

- + Very easy to do
- + More insight
- − Values are still ignored

Events with value

Sampling

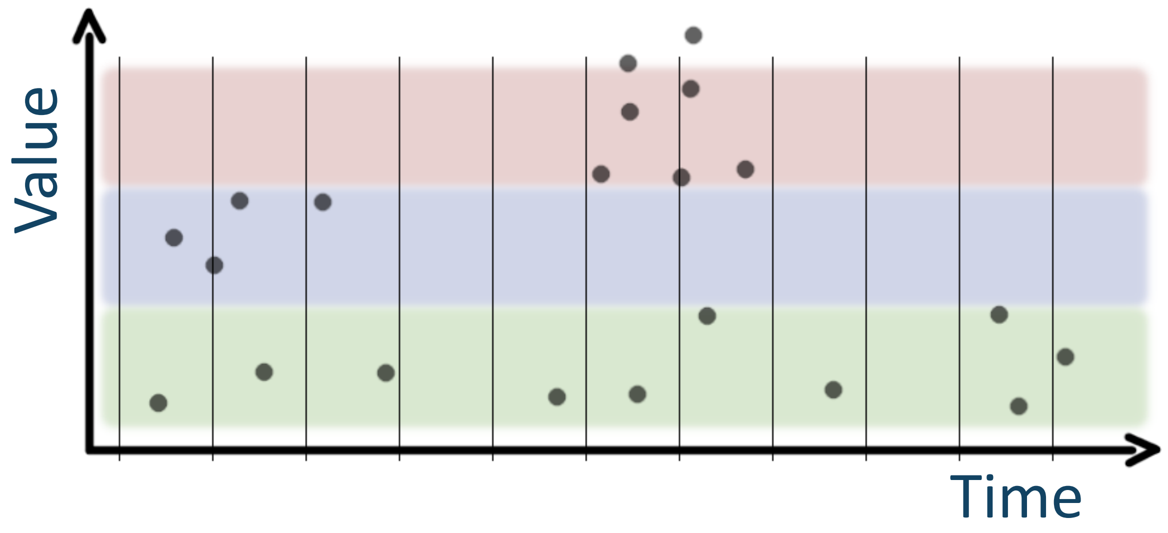


Incorporate the values

- ? Sum of all values
- ? Sum of values between two samples
- ? Average and variance

Events with value

Histograms

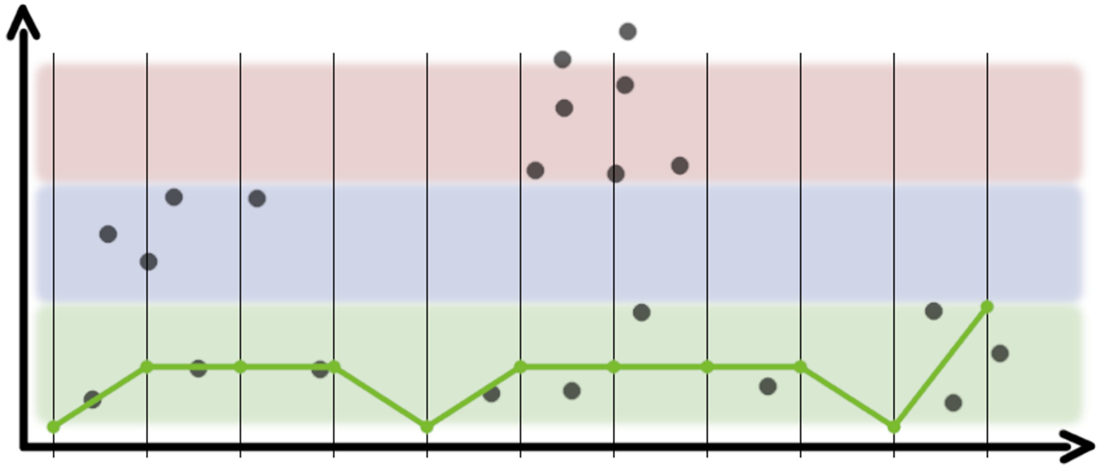


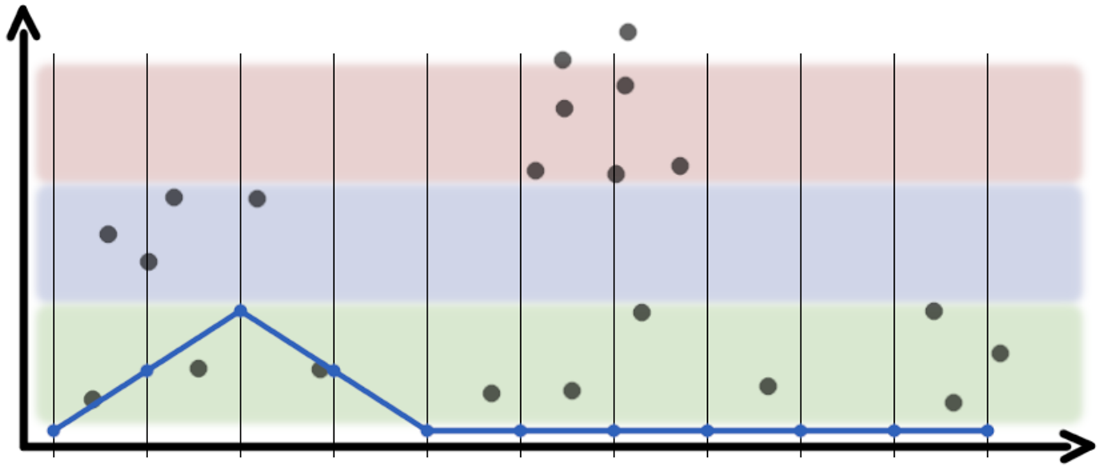
Group events in buckets

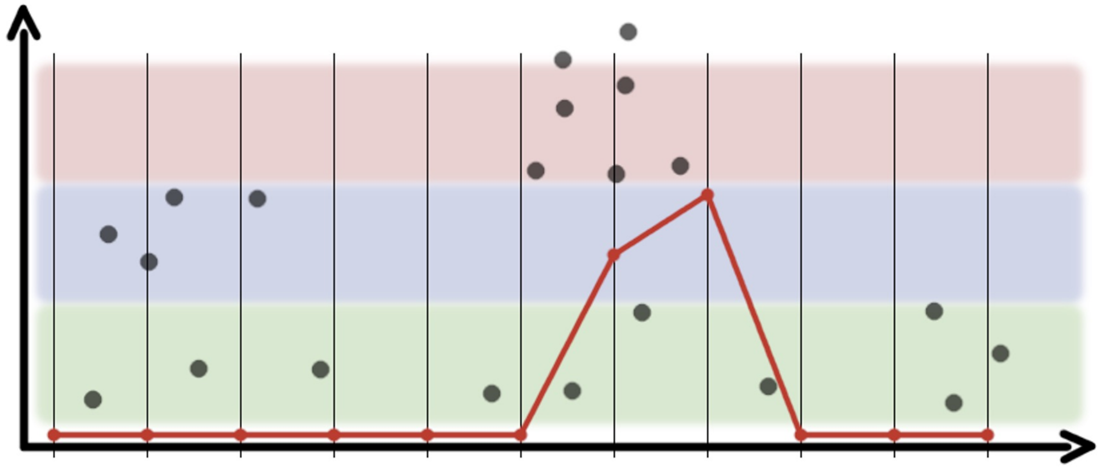
+ Easy to do
+ Allows to qualify by value

Events with value

Histograms



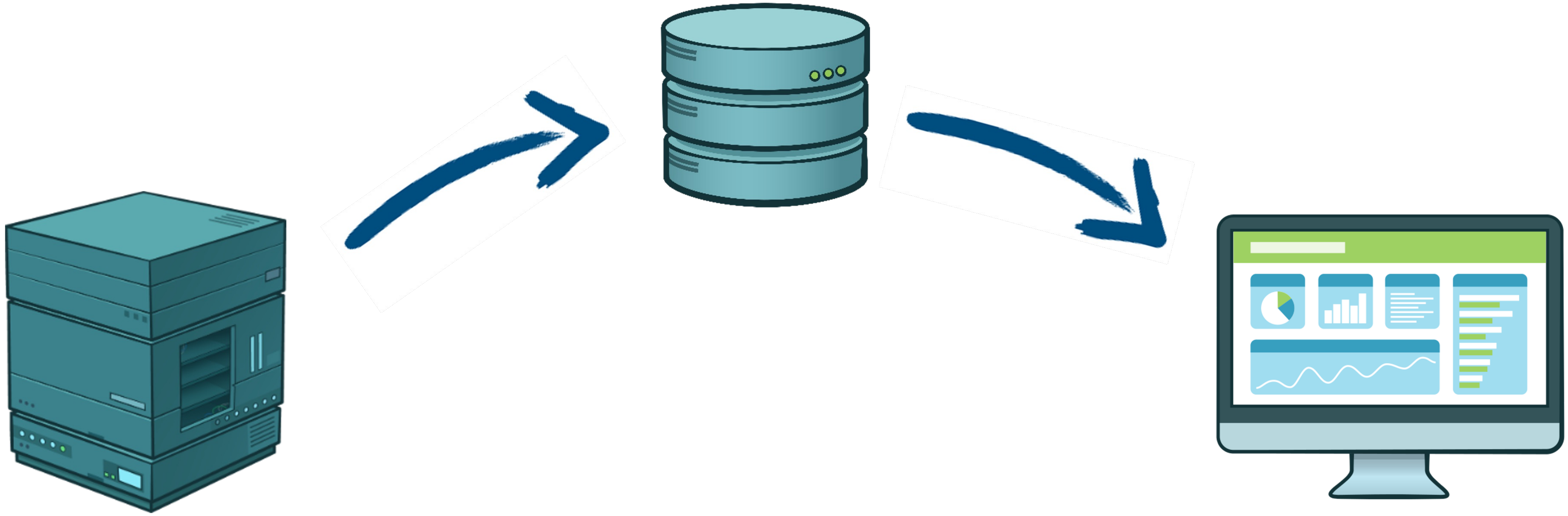




Metrics handling

Metrics handling

From source to insight



Aquire metrics

What to emit?

Emit raw events

+ Small overhead
+ All information is persisted
− Large amount of data must be handled

Aquire metrics

What to emit?

Emit aggregations

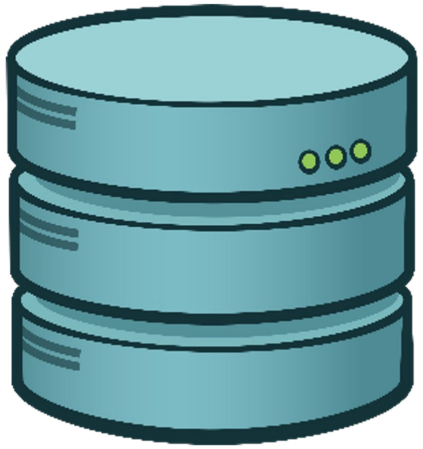
+ Data volume is reduced

− Some information is lost

− CPU/memory usage for creating the aggregations

Store metrics

Where to store?



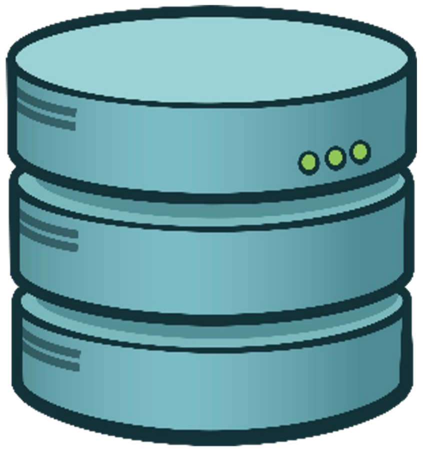
Store on local file system

+ Fast access and low latency

− Hard to collect data for evaluation

Store metrics

Where to store?



Store on central server

+ Simple to collect data for evaluation

− Slow access and high latency

− Limitation in bandwidth

Java Flight Recorder

Monitoring tools in your JDK

There are tons of metrics available out of the box

- Java VisualVM
- JConsole
- Dignostic Command Tool
- Java Flight Recorder and Mission Control

Not shipped anymore with Java 9+*

*Can be download separately: https://visualvm.github.io

Java Flight Recorder …

… and the Mission Control Center

- Java Flight Recorder (JFR) is part of OpenJDK based Java builds since version 11
- JFR is integrated directly in the JVM
- JFR affects the performance of a running application as little as possible

JFR for Oracle JDK 8

A complicated story

- Before Java 8 update 262 JFR was only available as part of the **Oracle JDK**
- It was only allowed to be used by support customers of Oracle and was hidden behind command line flags.

JFR for OpenJDK 8

A complicated story

- Since Java 8 update 262 the JFR is part of any OpenJDK build

Java Mission Control

A way to browse the JFR data

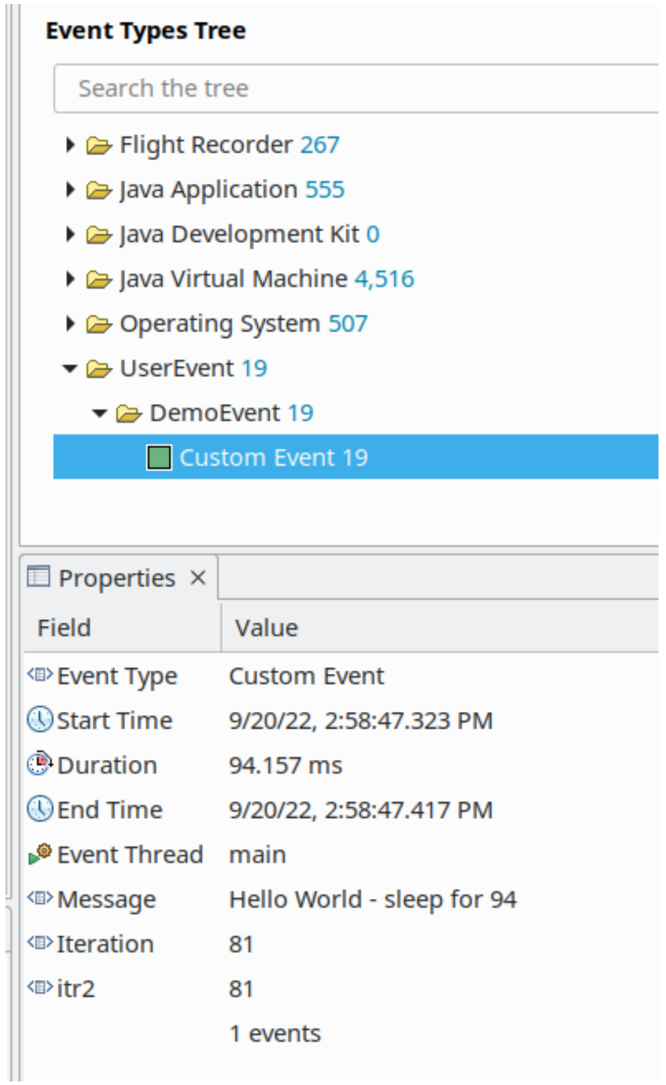
- The Java Mission Control release can be downloaded at Eclipse Adotpium https://adoptium.net/jmc.html

Demo

Custom JFR events

Add your own stuff

```java
@Category({"UserEvent", "DemoEvent"})
@Label("Custom Event")
static class CustomJfrEvent extends Event {
    @Label("Message") String message;
    @Label("Iteration") int itr1;
    int itr2;

    CustomJfrEvent(int iteration) {
        this.itr1 = iteration;
        this.itr2 = iteration;
    }
}
```



Event Types Tree

Search the tree

- Flight Recorder 267
- Java Application 555
- Java Development Kit 0
- Java Virtual Machine 4,516
- Operating System 507
- UserEvent 19
  - DemoEvent 19
    - Custom Event 19

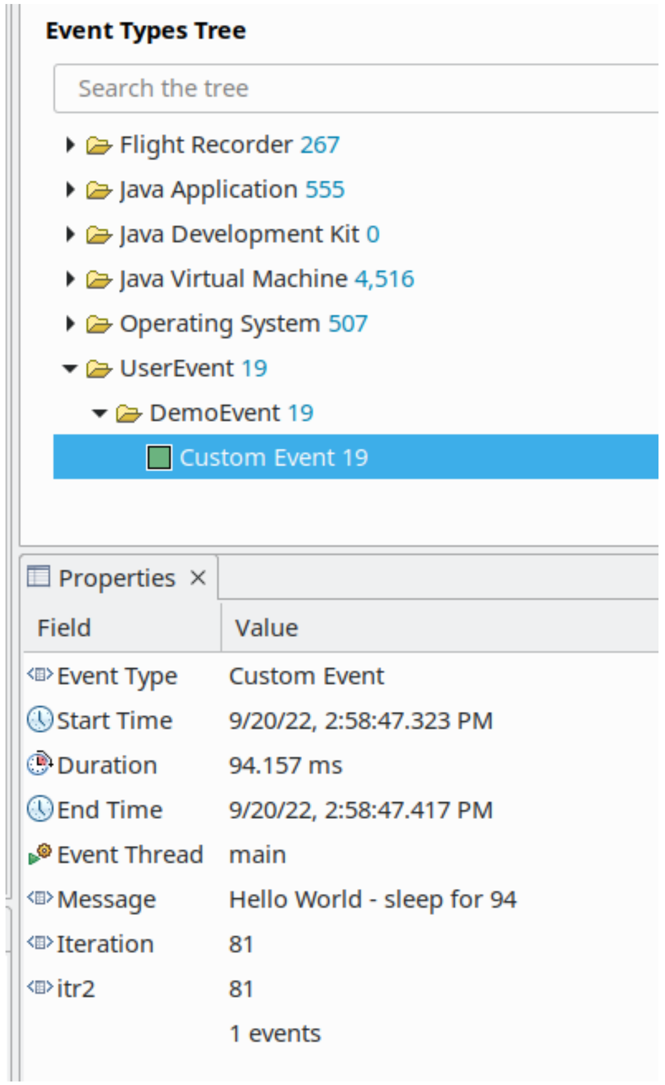
Properties

Field	Value
Event Type	Custom Event
Start Time	9/20/22, 2:58:47.323 PM
Duration	94.157 ms
End Time	9/20/22, 2:58:47.417 PM
Event Thread	main
Message	Hello World - sleep for 94
Iteration	81
itr2	81
	1 events

Custom JFR events

Add your own stuff

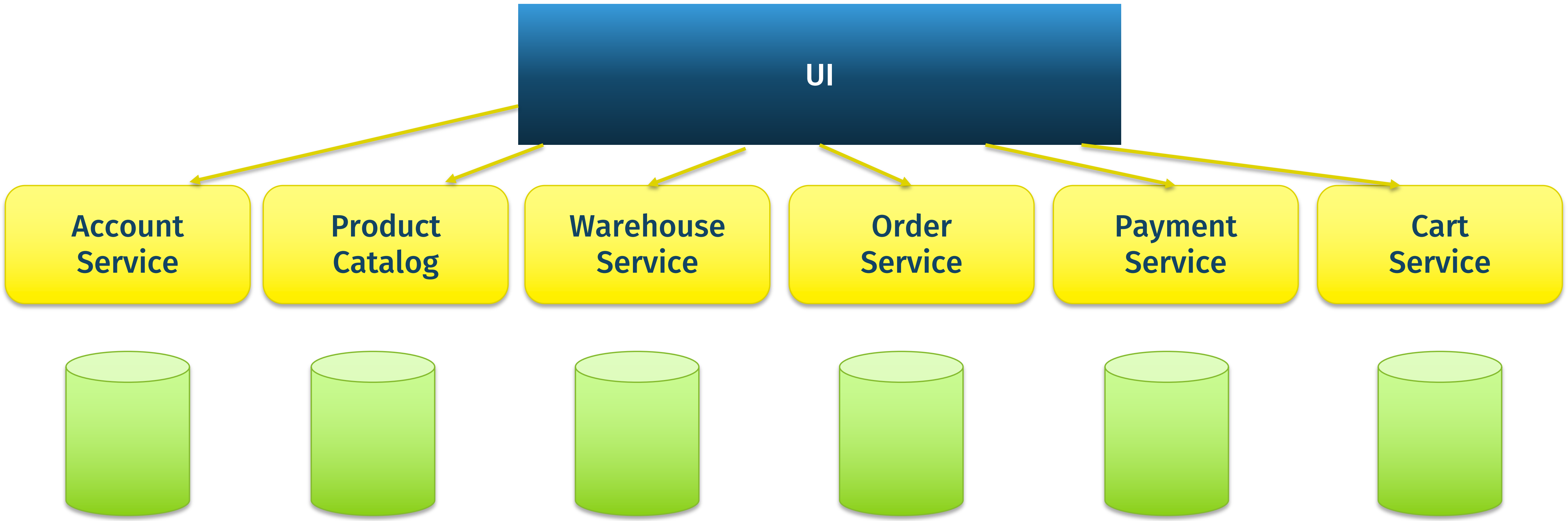
```
for (int i = 0; i < 1000; i++) {
   final int random = r.nextInt(500);

   final CustomJfrEvent event = new CustomJfrEvent(i);
   event.begin();
   event.message = "Hello World - sleep for " + random;
   Thread.sleep(random);
   event.commit();
}
```



Field	Value
Event Type	Custom Event
Start Time	9/20/22, 2:58:47.323 PM
Duration	94.157 ms
End Time	9/20/22, 2:58:47.417 PM
Event Thread	main
Message	Hello World - sleep for 94
Iteration	81
itr2	81
	1 events

Metrics in context

of a state-of-the-art system


UI
Account Service
Product Catalog
Warehouse Service
Order Service
Payment Service
Cart Service

New Relic.

Prometheus

STATSD

Grafana

DATADOG

influxdb

graphite

Stackdriver

dynatrace

MICROMETER

https://www.micrometer.io

MICROMETER

Facade?

=

Provide data to

AppOptics, Azure Monitor, Netflix **Atlas,** AWS **CloudWatch, Datadog, Dynatrace, Elastic, Ganglia, Graphite, Humio, Influx/Telegraf, JMX, KairosDB, New Relic, Prometheus, SignalFx,** Google **Stackdriver, StatsD,** and **Wavefront.**

without pain

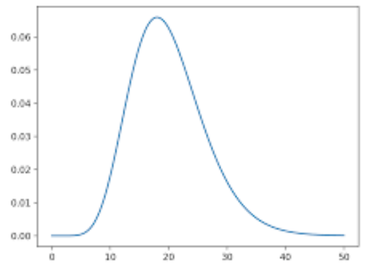
MICROMETER
Meter types

Timer

Long task timers

Counter

Gauge



Distribution summary

MICROMETER
Integrations

Ktor

MICRONAUT

spring boot

VERT.X™

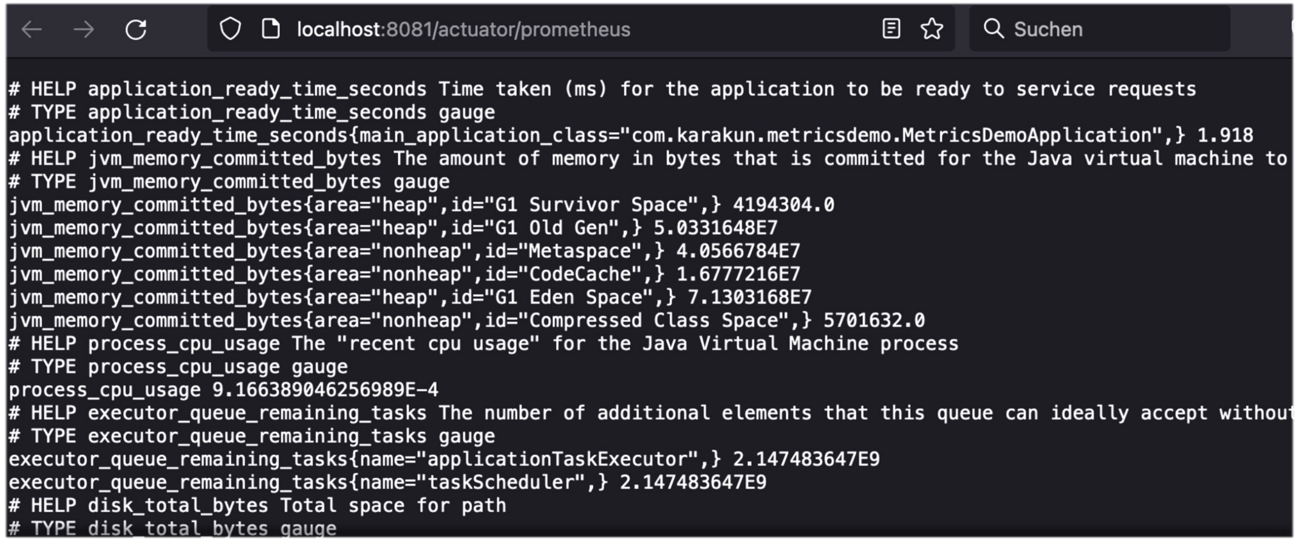
QUARKUS

MICROMETER

Using it (Spring Boot)

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

<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-core</artifactId>
</dependency>

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

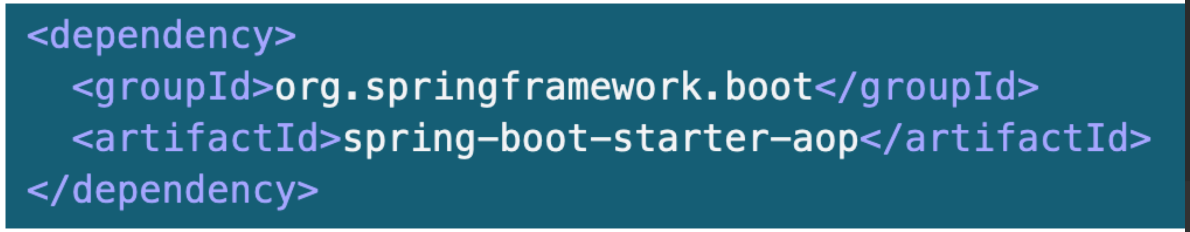
```
management.endpoints.web.exposure.include=prometheus
```



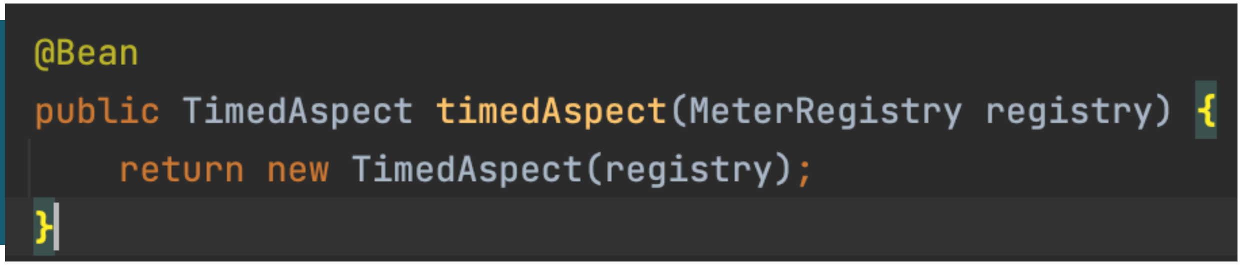
localhost:8081/actuator/prometheus

```
# HELP application_ready_time_seconds Time taken (ms) for the application to be ready to service requests
# TYPE application_ready_time_seconds gauge
application_ready_time_seconds{main_application_class="com.karakun.metricsdemo.MetricsDemoApplication",} 1.918
# HELP jvm_memory_committed_bytes The amount of memory in bytes that is committed for the Java virtual machine to
# TYPE jvm_memory_committed_bytes gauge
jvm_memory_committed_bytes{area="heap",id="G1 Survivor Space",} 4194304.0
jvm_memory_committed_bytes{area="heap",id="G1 Old Gen",} 5.0331648E7
jvm_memory_committed_bytes{area="nonheap",id="Metaspace",} 4.0566784E7
jvm_memory_committed_bytes{area="nonheap",id="CodeCache",} 1.6777216E7
jvm_memory_committed_bytes{area="heap",id="G1 Eden Space",} 7.1303168E7
jvm_memory_committed_bytes{area="nonheap",id="Compressed Class Space",} 5701632.0
# HELP process_cpu_usage The "recent cpu usage" for the Java Virtual Machine process
# TYPE process_cpu_usage gauge
process_cpu_usage 9.166389046256989E-4
# HELP executor_queue_remaining_tasks The number of additional elements that this queue can ideally accept withou
# TYPE executor_queue_remaining_tasks gauge
executor_queue_remaining_tasks{name="applicationTaskExecutor",} 2.147483647E9
executor_queue_remaining_tasks{name="taskScheduler",} 2.147483647E9
# HELP disk_total_bytes Total space for path
# TYPE disk_total_bytes gauge
```

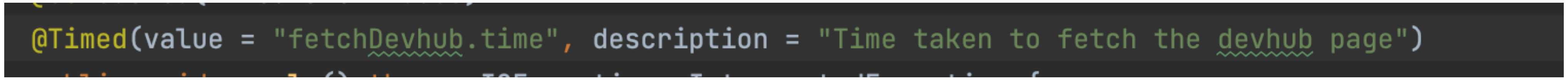
MICROMETER
Using annotations (Spring Boot)



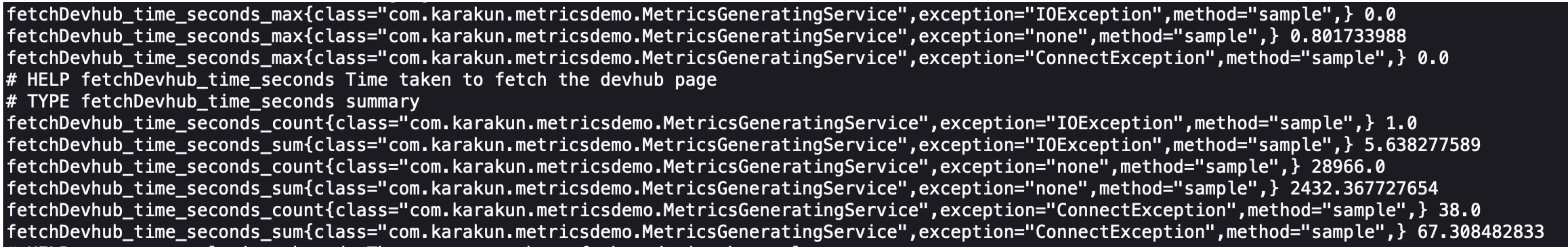
```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
```



```java
@Bean
public TimedAspect timedAspect(MeterRegistry registry) {
    return new TimedAspect(registry);
}
```



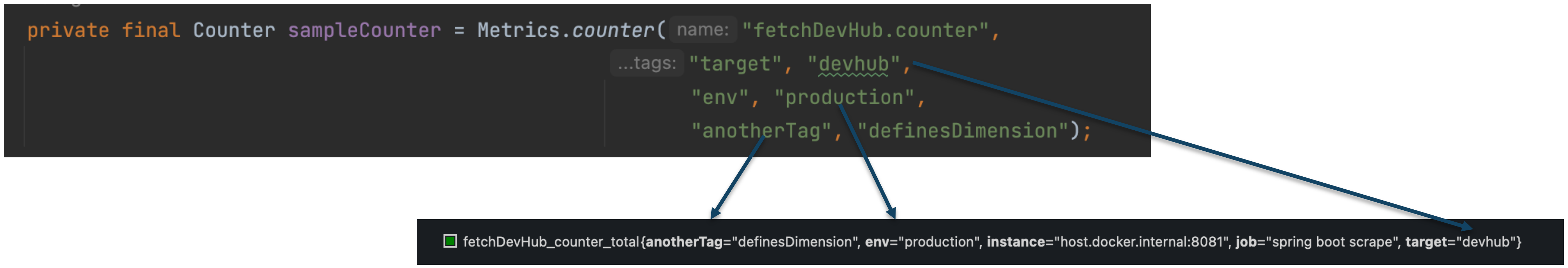
```java
@Timed(value = "fetchDevhub.time", description = "Time taken to fetch the devhub page")
```



```
fetchDevhub_time_seconds_max{class="com.karakun.metricsdemo.MetricsGeneratingService",exception="IOException",method="sample",} 0.0
fetchDevhub_time_seconds_max{class="com.karakun.metricsdemo.MetricsGeneratingService",exception="none",method="sample",} 0.801733988
fetchDevhub_time_seconds_max{class="com.karakun.metricsdemo.MetricsGeneratingService",exception="ConnectException",method="sample",} 0.0
# HELP fetchDevhub_time_seconds Time taken to fetch the devhub page
# TYPE fetchDevhub_time_seconds summary
fetchDevhub_time_seconds_count{class="com.karakun.metricsdemo.MetricsGeneratingService",exception="IOException",method="sample",} 1.0
fetchDevhub_time_seconds_sum{class="com.karakun.metricsdemo.MetricsGeneratingService",exception="IOException",method="sample",} 5.638277589
fetchDevhub_time_seconds_count{class="com.karakun.metricsdemo.MetricsGeneratingService",exception="none",method="sample",} 28966.0
fetchDevhub_time_seconds_sum{class="com.karakun.metricsdemo.MetricsGeneratingService",exception="none",method="sample",} 2432.367727654
fetchDevhub_time_seconds_count{class="com.karakun.metricsdemo.MetricsGeneratingService",exception="ConnectException",method="sample",} 38.0
fetchDevhub_time_seconds_sum{class="com.karakun.metricsdemo.MetricsGeneratingService",exception="ConnectException",method="sample",} 67.308482833
```

MICROMETER
Dimensions

With Dimensions metrics can be **sliced**, ***diced***, **aggregated** and ***compared***.
Dimensions are defined as **tags**



```
private final Counter sampleCounter = Metrics.counter( name: "fetchDevHub.counter",
                                                  ...tags: "target", "devhub",
                                                           "env", "production",
                                                           "anotherTag", "definesDimension");
```

```
fetchDevHub_counter_total{anotherTag="definesDimension", env="production", instance="host.docker.internal:8081", job="spring boot scrape", target="devhub"}
```

MICROMETER Registry

Destination for all measurements

Can be **instanciated and configured** as required

```
final MeterRegistry registry = new PrometheusMeterRegistry(PrometheusConfig.DEFAULT);
```

Meters belong to a registry

```
registry.counter( name: "registrySpecificMeter", Tags.empty());
```

Multiple registries can be addressed!

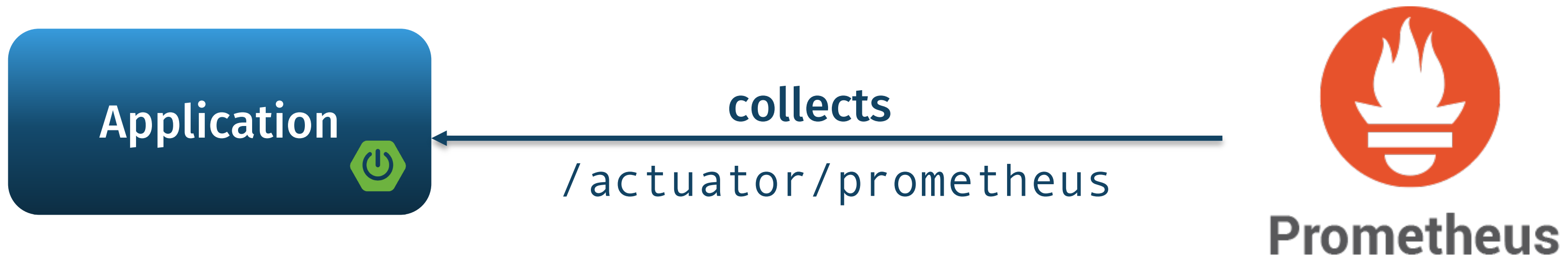
Utilizing metrics: centralized storage and analysis

- Most systems consist of **distributed components**
- Some **components** are even instanciated **several times**
- Compontents get restarted or replaced
- But we need the **big picture** of the system over **longer period of time**
- Thus we need to **store metrics** in a **seperate system**

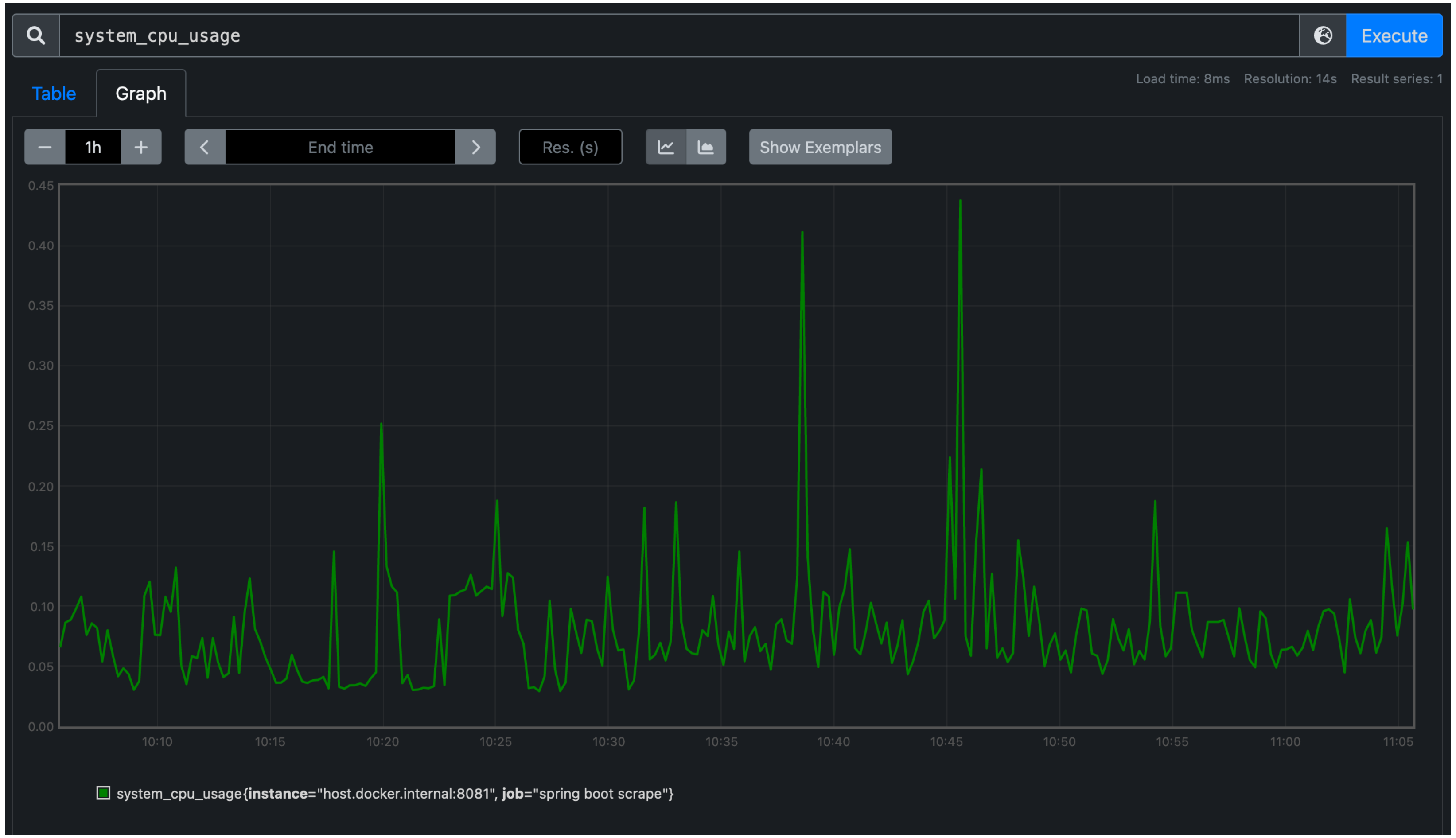
Utilizing metrics: centralized storage and analysis

- **Several established tools available**
- **Cloud platforms offer integrated solutions**
- **Example with Grafana and Prometheus**
  based on **OpenSource** components
  very **established combination**
  good integration in tools and APIs

Utilizing metrics: centralized storage and analysis

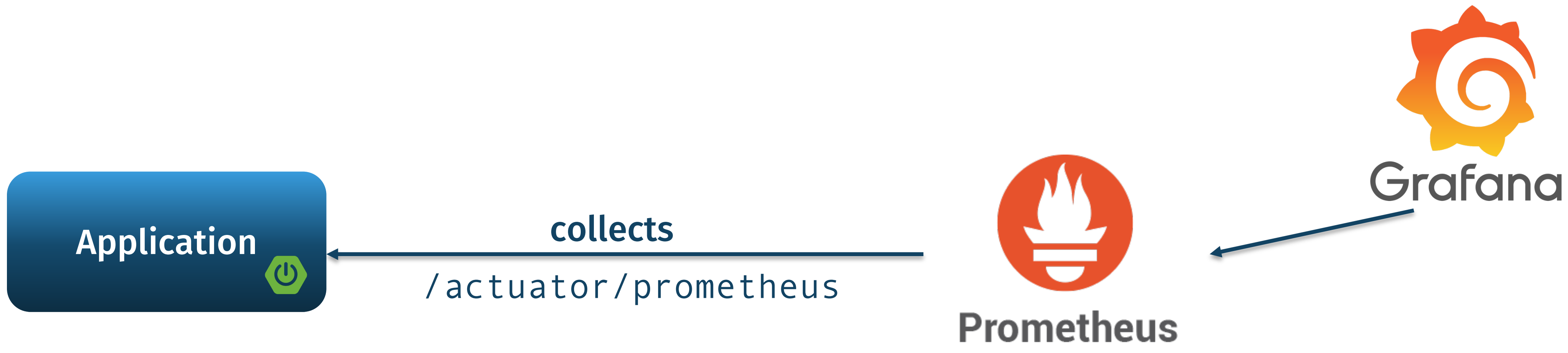


Utilizing metrics: centralized storage and analysis

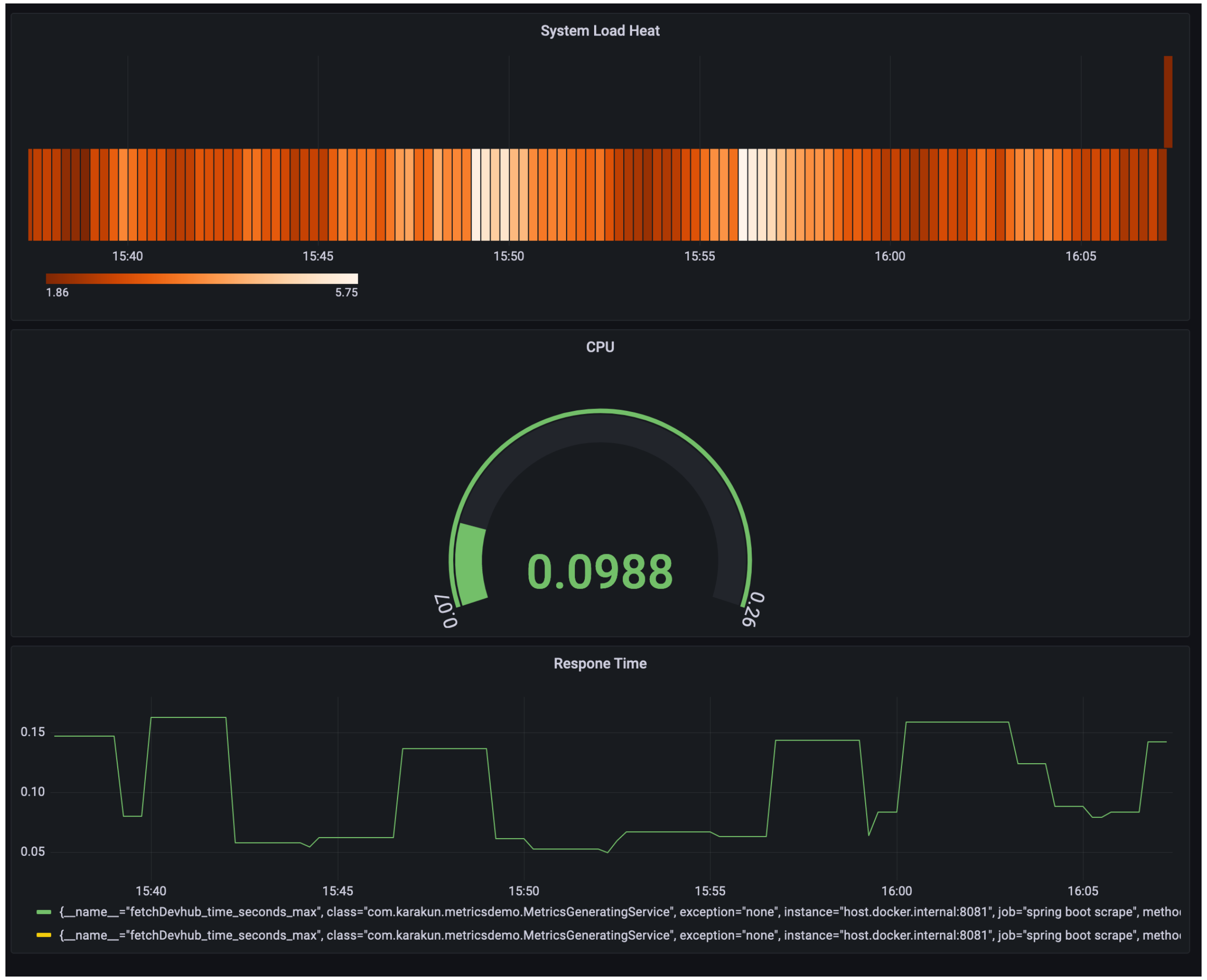


🟩 system_cpu_usage{**instance**="host.docker.internal:8081", **job**="spring boot scrape"}

Utilizing metrics: centralized storage and analysis



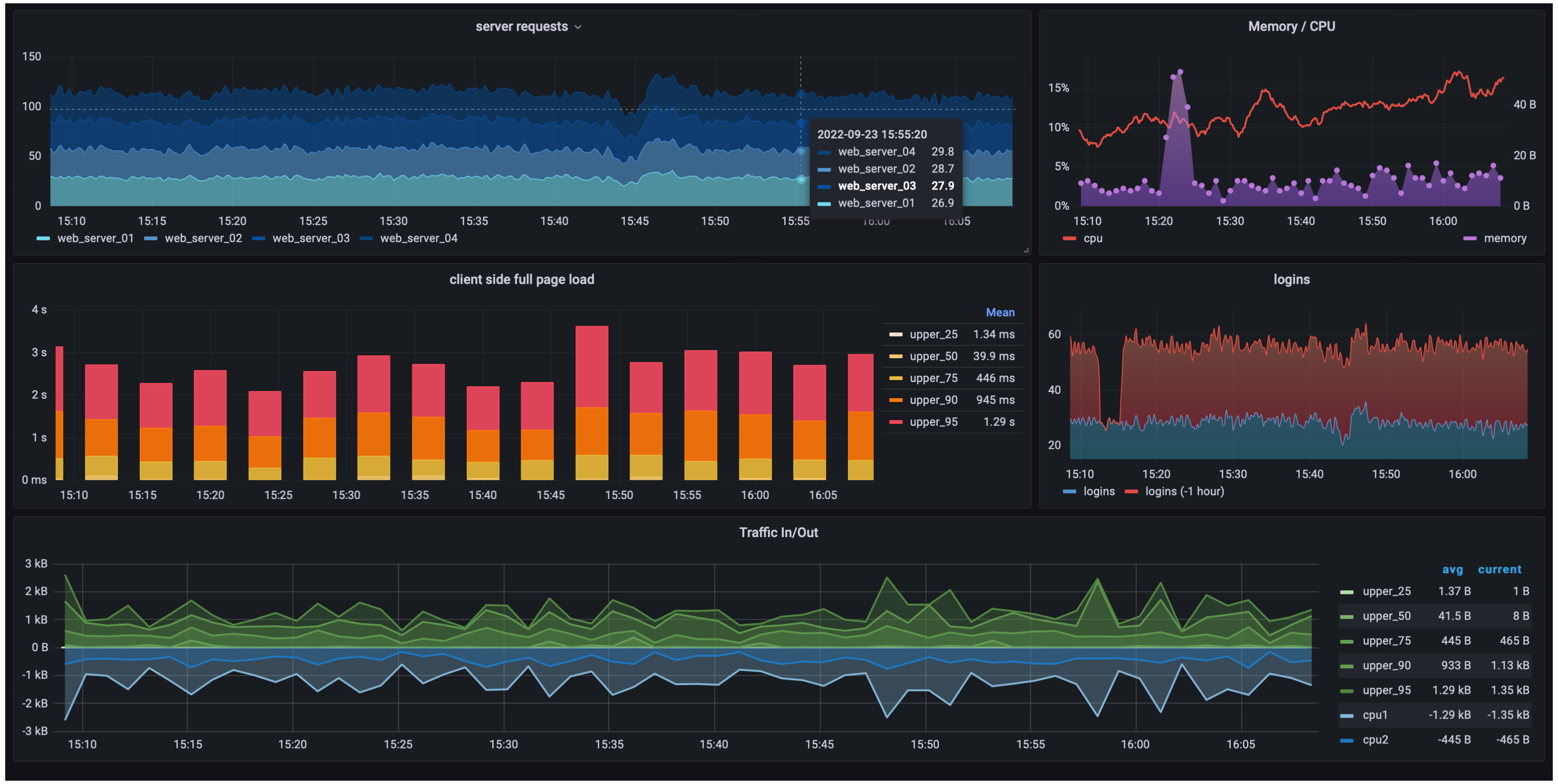
Utilizing metrics: centralized storage and analysis



🟩 {__name__="fetchDevhub_time_seconds_max", class="com.karakun.metricsdemo.MetricsGeneratingService", exception="none", instance="host.docker.internal:8081", job="spring boot scrape", metho

🟨 {__name__="fetchDevhub_time_seconds_max", class="com.karakun.metricsdemo.MetricsGeneratingService", exception="none", instance="host.docker.internal:8081", job="spring boot scrape", metho

Utilizing metrics: centralized storage and analysis



2022-09-23 15:55:20

web_server_04	29.8
web_server_02	28.7
web_server_03	27.9
web_server_01	26.9

	Mean
upper_25	1.34 ms
upper_50	39.9 ms
upper_75	446 ms
upper_90	945 ms
upper_95	1.29 s

	avg	current
upper_25	1.37 B	1 B
upper_50	41.5 B	8 B
upper_75	445 B	465 B
upper_90	933 B	1.13 kB
upper_95	1.29 kB	1.35 kB
cpu1	-1.29 kB	-1.35 kB
cpu2	-445 B	-465 B

https://play.grafana.org

Gerne beraten und schulen wir auch Sie dabei, welche Technologien Sie am besten einsetzen und wie Sie Ihre Software-Entwicklung verbessern können.

Wir freuen uns auf Ihre Kontaktaufnahme!

karakun

Stephan Classen

T. +49 231 226 157 00
E. stephan.classen@karakun.com

Markus Schlichting

T. +49 172 3062009
E. markus.schlichting@karakun.com

Karakun GmbH

Selkamp 12
44287 Dortmund
Deutschland

T. +49 231 226 157 00
E. info@karakun.com
W. www.karakun.com

Karakun AG

Elisabethenanlage 25
4051 Basel
Schweiz

T. +41 61 551 36 00
E. info@karakun.com
W. www.karakun.com