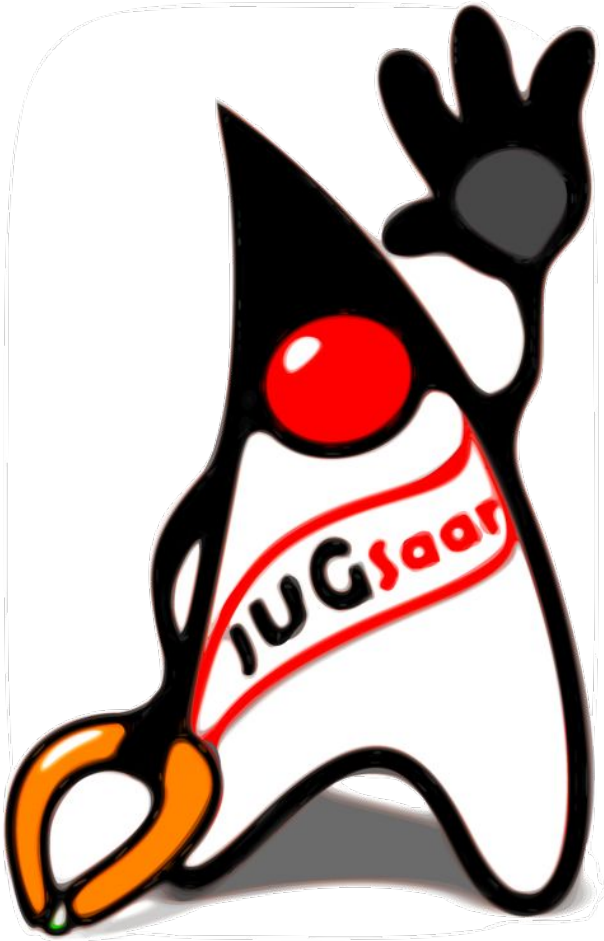
Java User Group Saarland



Monitoring with Prometheus

Thomas Darimont eurodata AG

32. Meeting

17. October 2017

Sponsored by

eurodata

INFOSERVE
>eurodata-Gruppe

Monitoring

Why monitor?

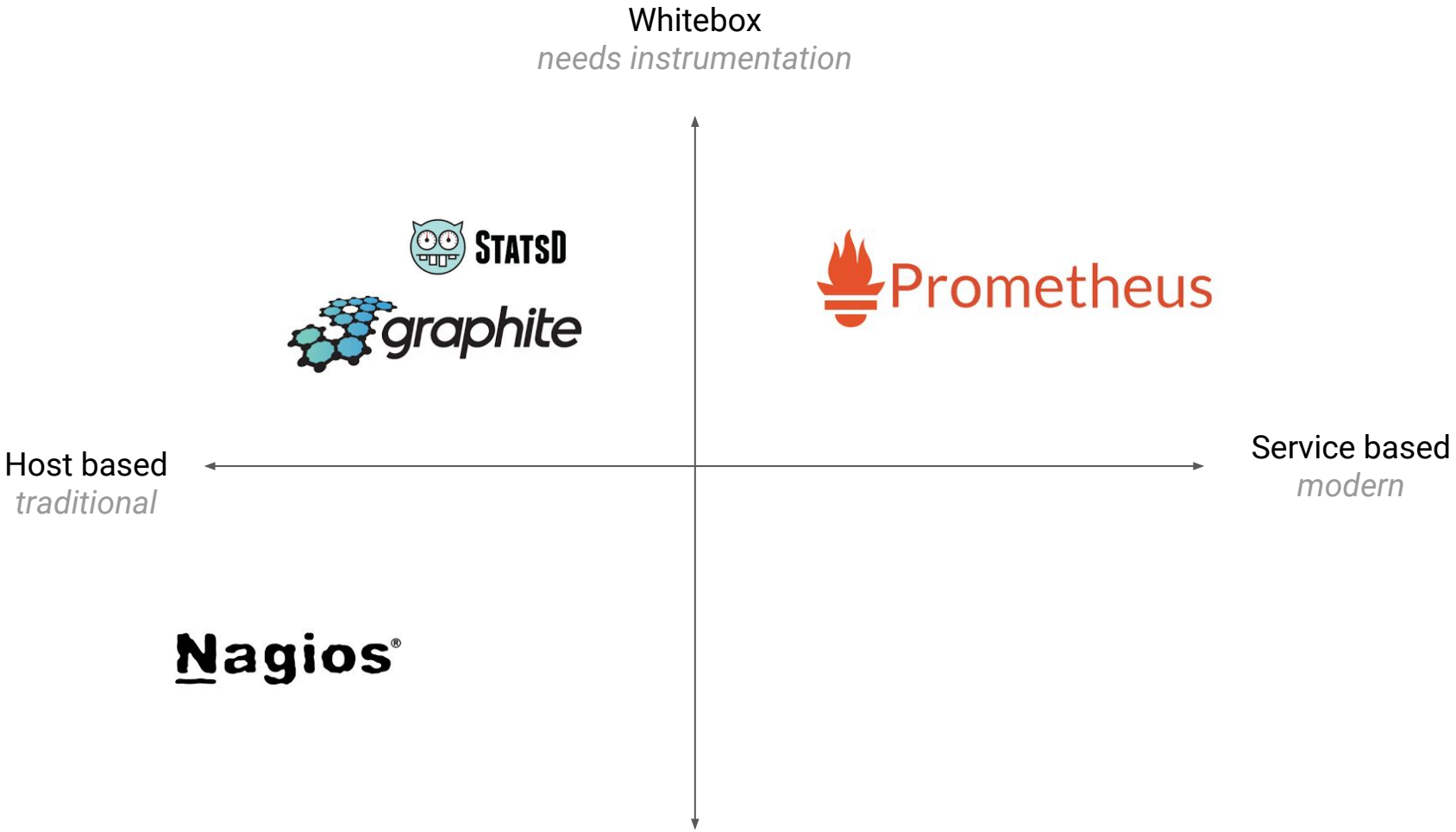
Why Monitor

- Know when things go wrong
- Be able to debug and gain insight
  - See changes over time
  - Notice trends
- Keep eye on KPIs
  - Service Level Indicators (SLI) — Measurement
  - Service Level Objectives (SLO) — Goal
  - Service Level Agreements (SLA) — Hard limit → €€€
- Build impressive dashboards ;-)

Monitor everything

Level	What to monitor
Host	Hardware failure, Provisioning, Resources
Container	Resource Usage, Performance characteristics
JVM	GC, Threads, ClassLoading
Application	Latencies, Errors, APIs, Internal State
Orchestration	Cluster Resources, Scheduling

Dimensions of Monitoring



not a complete list of the monitoring landscape

Logfiles vs. Metrics

- **Logfiles** have *information about* an event
- **Metrics** *aggregate across* events
- Metrics help to show where the problem is
  - … increased latency of image-service API
  - … increased error rate on host 0xidspispopd
- … from there Logfiles can help to pinpoint the Problem
  - via drill down analysis

Prometheus

Prometheus is an open-source systems monitoring and alerting toolkit

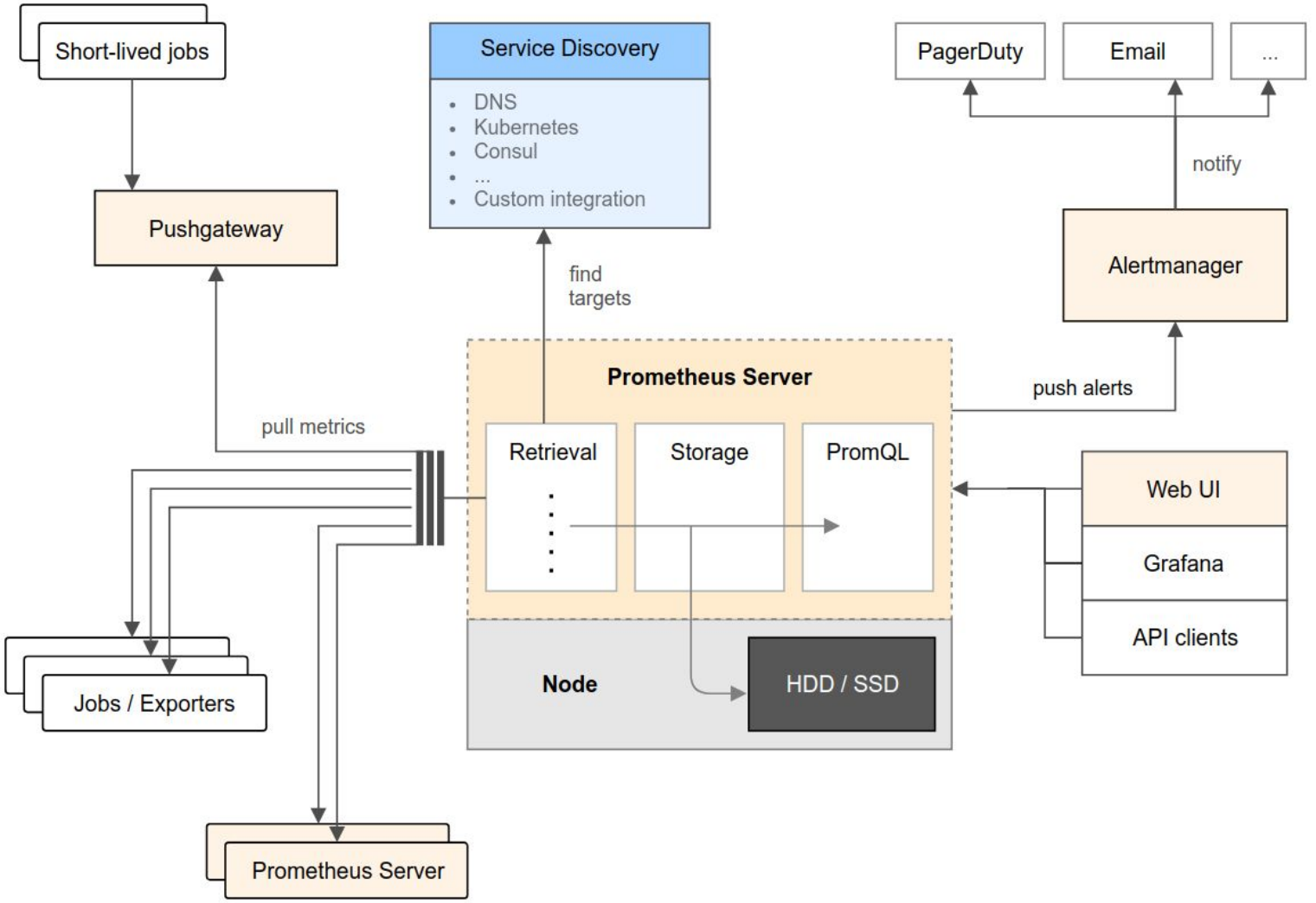
Overview

- Open Source Monitoring System
  - Provides Metrics Collection
  - Storage via Time Series Database (TSDB)
  - Querying, Alerting, Dashboarding
- Opinionated approach
  - Favours whitebox monitoring via instrumentation
  - Favours metrics ingestion via Pull vs. Push
- Built with dynamic cloud environments in mind
  - Service discovery
- Written in Go
  - Cross Platform
  - Robust & flexible
  - Standalone (no dependencies)

Main Features

- **Multi-dimensional** data model
- **Flexible query language** PromQL
- **Pull model** for time series collection over HTTP
- **Pushing** time series **is supported** via push gateway
- **Target** definition via **service discovery** or **static config**

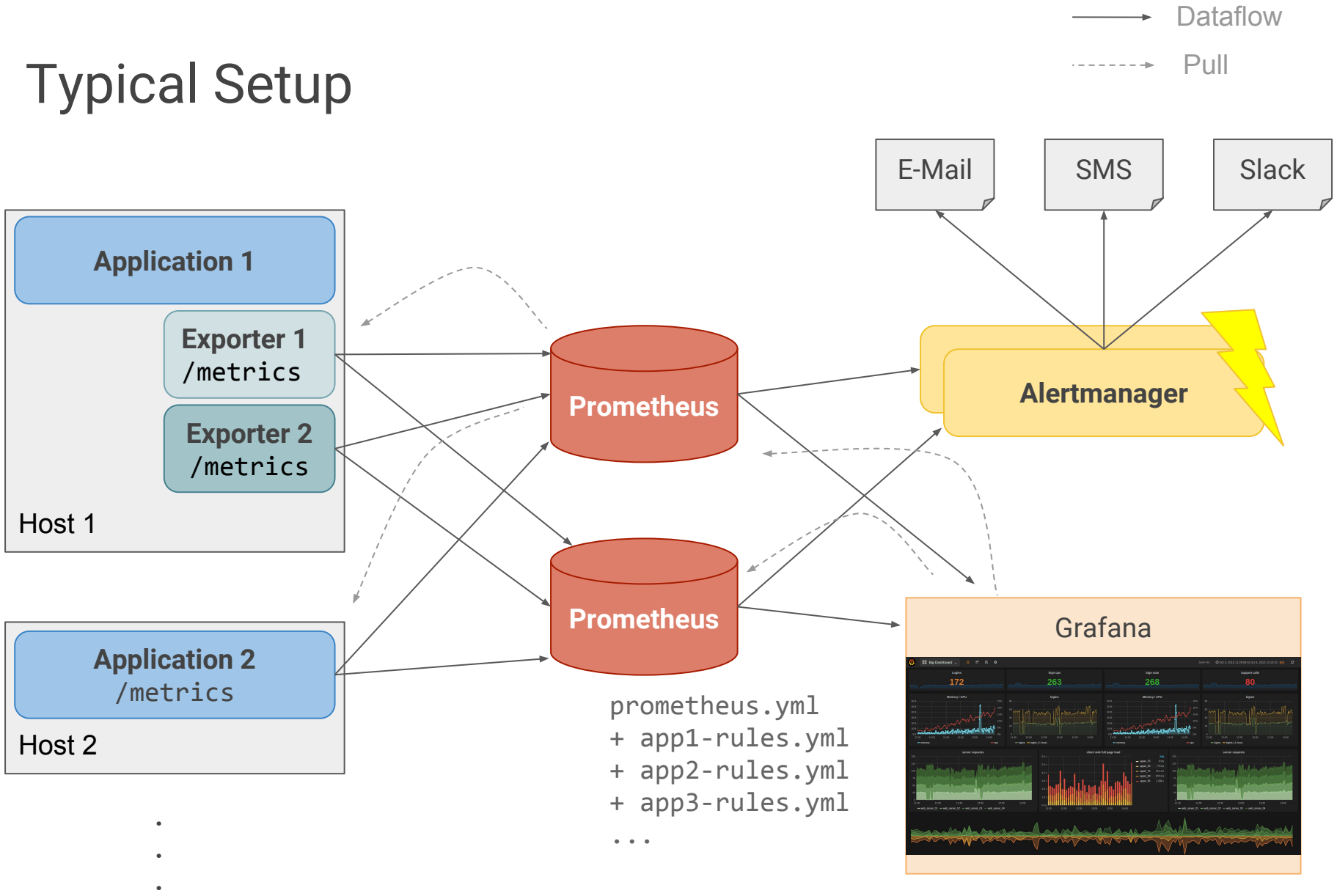
Architecture



Components

- Prometheus server scrapes and stores time series data
- Client libraries to instruments application code
- Push gateway supports short-lived (batch) jobs
- Exporters exposes metrics for ingestion over HTTP
- Alertmanager conditional alerts via multiple channels
- Support tools

Typical Setup



Concepts

- Timeseries & Samples
- Metric names & labels
- Metric types
- Queries & rules
- Job & instances

Timeseries & Samples

- **Timeseries** streams of timestamped values
  - belonging to the same metric
  - the same set of labeled dimensions
- **Sample** tuple of the actual time series data
  - float64 value
  - millisecond-precision timestamp

Metrics & Labels

- ***Metric = Metric name*** and a set of *key-value pairs* aka ***labels***
- Metric name
  - Specifies the feature that is measured
  - e.g. `http_requests_total`
- Labels
  - Enables Prometheus's dimensional data model
  - Allows many sub metrics from a base metric
  - New time series for each label combination → Memory!
- Notation
  - `<metric name>{<labelname>=<labelvalue>,...} value [timestamp]`
- Example metric
  - `http_requests_total{method="post",code="200"} 1027 15081...`

Metric types

- Counter an monotonic incrementing value
  - e.g. # processed requests total
- Gauge measures a value
  - e.g. # currently connected clients
- Histogram measures the distribution of values
  - e.g. # requests below 1 seconds / 5 seconds / 10 seconds
- Summary similar to a histogram
  - provides a total count of observations and a their sum
  - provides quantiles over sliding time window

Queries

- PromQL query language
- Vectorized functions of time series values over ranges
  - +, -, *, /, %, ^, aggregates (avg, sum, stddev…) , functions(rate, join, predict,...)
- Answers questions like
  - What is the 95th percentile latency over the past month?
  - How full will the disks be in 4 days?
  - Which servers are the Top5 consumers of CPU?
- Example
  - Average number of HTTP Post requests per second in 1 min time window
  - `rate(http_requests_total{method=”post”}[1m])`

Function
rate of change

Variable Reference

Range expression

Rules

- Rule types
  - Recording Rules for precalculating metrics
  - Alerting Rules alert conditions and handling
- Configuration
  - Included via `rule_files` in `prometheus.yml`
  - Rules & Alerts can be mixed

Jobs & Instances

- Configured via `prometheus.yml`
- Job
  - Logical target be scraped
  - Application, Service, System
  - Contains generic scraping configuration
  - Defines additional labels
  - Static or dynamic configuration
- Instance
  - Concrete target
  - Host, Container Instance, Process

Exporters

- Expose metrics via HTTP endpoint `/metrics`
  - Simple text format
  - Metric + Float64
- Many third-party exporters available
- Useful examples
  - node_exporter disk, cpu, mem, io, network stats on Linux
  - WMI_exporter *node_exporter* for Windows
  - blackbox_exporter pulls data from HTTP, TCP endpoints
  - grok_exporter extracts metrics from logfiles
  - cadvisor analyzes resource usage of containers
  - postgres exporter information about database usage

/metrics

```
# HELP api_requests_latency_seconds API Request latency in seconds.
# TYPE api_requests_latency_seconds histogram
api_requests_latency_seconds_bucket{le="0.005",} 512968.0
api_requests_latency_seconds_bucket{le="0.01",} 516819.0
api_requests_latency_seconds_bucket{le="0.025",} 519532.0
api_requests_latency_seconds_bucket{le="0.05",} 520041.0
api_requests_latency_seconds_bucket{le="0.075",} 520083.0
api_requests_latency_seconds_bucket{le="0.1",} 520085.0
api_requests_latency_seconds_bucket{le="0.25",} 520767.0
api_requests_latency_seconds_bucket{le="0.5",} 520767.0
api_requests_latency_seconds_bucket{le="0.75",} 520767.0
api_requests_latency_seconds_bucket{le="1.0",} 520767.0
api_requests_latency_seconds_bucket{le="2.5",} 520866.0
api_requests_latency_seconds_bucket{le="5.0",} 520918.0
api_requests_latency_seconds_bucket{le="7.5",} 520918.0
api_requests_latency_seconds_bucket{le="10.0",} 520918.0
api_requests_latency_seconds_bucket{le="+Inf",} 520918.0
api_requests_latency_seconds_count 520918.0
api_requests_latency_seconds_sum 810.3394127730076
# HELP httpsessions_max httpsessions_max
# TYPE httpsessions_max gauge
httpsessions_max -1.0
# HELP httpsessions_active httpsessions_active
# TYPE httpsessions_active gauge
httpsessions_active 0.0
# HELP mem mem
# TYPE mem gauge
mem 564160.0
# HELP mem_free mem_free
# TYPE mem_free gauge
mem_free 198571.0
```

- Feature rich metrics dashboard and graph editor
- Many free dashboards and plugins available
  - Look amazing out of the box!
- Support for Alerting
- Support for many Metrics Providers
  - Graphite, Elasticsearch, OpenTSDB
  - InfluxDB, and Prometheus … any more
- Open Source, Written in Go + AngularJS (1.x)

Grafana Dashboards

Java Integration

- Simple Java Client
- Supports all metrics types
- JVM & Hotspot Metrics
  - ClassLoading
  - Garbage Collector
  - Threads
  - Application Info
- Pushgateway Support for ephemeral and batch jobs
- Generic JMX Exporter Java Agent
- Custom Metrics via embedded DSL
- Exposes `/metrics` endpoint via HTTP Servlet
  - Requires Servlet container...

Spring Boot Integration

- Add Dependency

```
<dependency>
    <groupId>io.prometheus</groupId>
    <artifactId>simpleclient_spring_boot</artifactId>
    <version>${prometheus.simpleclient.version}</version>
</dependency>
```

- Add Prometheus Config

```
@Configuration
// Registers /prometheus endpoint
@EnablePrometheusEndpoint
// Exposes spring boot metrics via the prometheus endpoint
@EnableSpringBootMetricsCollector
class PrometheusConfig { … }
```

- Define Counter

```
private static final Counter GREETINGS_TOTAL = Counter.build()
 .name("api_greeting_requests_total")
 .help("Total number of greeting requests.")
 .register();
```

- Increment Counter

```
@GetMapping("/greet")
Object greet(@RequestParam(defaultValue = "World") String name) {
 // shows up in the /prometheus endpoint
 GREETINGS_TOTAL.inc();
 …
}
```

Promagent

- Open Source https://github.com/fstab/promagent
- extensible Java Agent using Byte Buddy & client_java
- “Monitoring for Java Applications *without* Modifying their Source.”
- Default Metrics
  - HTTP: Number and duration of web requests
  - SQL: Number and duration of database queries

```
java \
    -javaagent:promagent/promagent-dist/target/promagent.jar=port=9300 \
    -jar gs-accessing-data-rest/complete/target/gs-accessing-data-rest-0.1.0.jar
```

Summary

- Prometheus + Exporters + Grafana works great!
- Easy to setup & use
- Good documentation
- Active Community
- Many libraries, exporters and integrations
- Plays well with others
  - Linux, Windows, Java, Docker, Kubernetes and other Platforms

Links

- Code & Slides https://github.com/jugsaar/jugsaar-meeting-32
- Prometheus https://prometheus.io/
- Videos Promcon 2016
- Videos Promcon 2017
- Promagent
- My Philosophy on Alerting
- Site Reliability Engineering Book