ALTIMETRIK™

SIMPLIFY TECHNOLOGY | AMPLIFY POSSIBILITY

Spring Boot, Micrometer, Prometheus and Grafana

How to add custom metrics to your application

Agenda

1. **Technology stack**

   Prometheus, Micrometer, Grafana

2. **Demo**

   Custom metrics and Grafana walkthrough

3. **Best practices**

Technology stack

Prometheus

- https://prometheus.io
- Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud
- 100% open source and community-driven
- data collection – collects various kinds of data from different sources
- storage – stores data over (configurable) time, so it allows to see trends
- querying – PromQL, built-in functions
- alerting - address issues proactively
- visualization – some built-in charts (but it’s better to use Grafana)

Prometheus

- stores all data as time series: streams of timestamped values belonging to the same metric and the same set of labeled dimensions
- time series is uniquely identified by
    - metric name
    - optional key-value pairs called labels
- notation
    - <metric name>{<label name>=<label value>, …}
    - api_http_requests_total{method="POST", handler="/messages"}

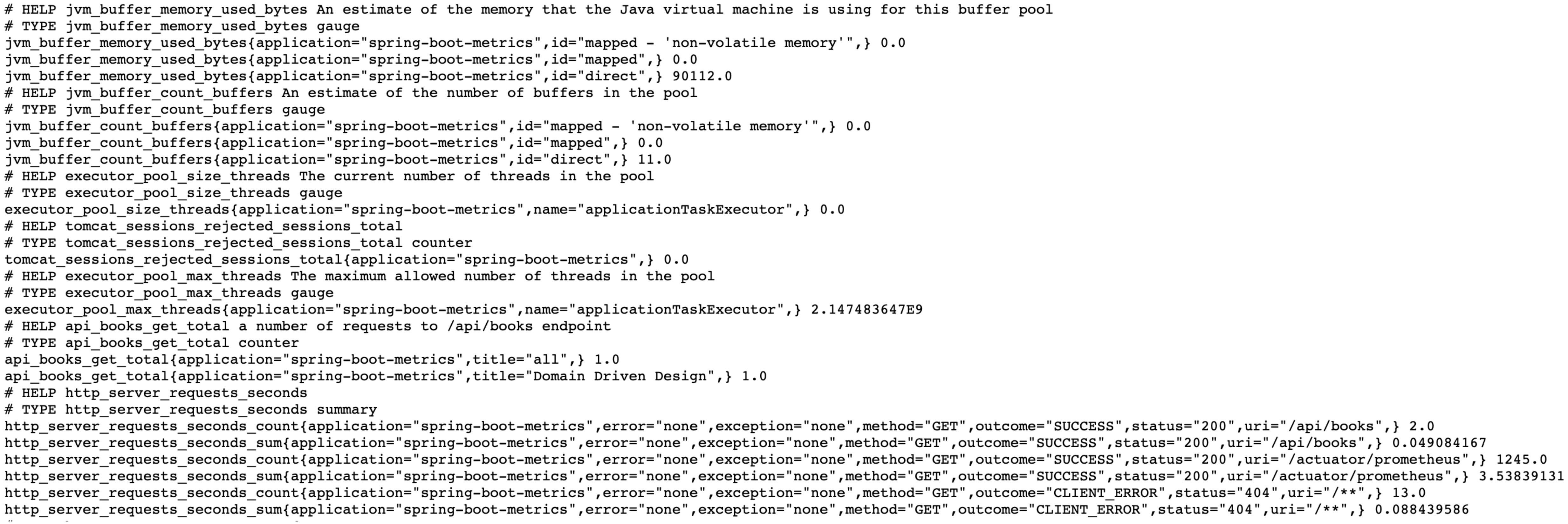
Prometheus

- metric types
  - **counter** – a single counter whose value can only increase or be reset to zero on restart
    - e.g., number of completed tasks, number of errors
    - **do not use a counter to expose a value that can decrease**
    - usually used with rate() function for per-second rate
  - **gauge** - a single numerical value that can arbitrarily go up and down
    - e.g., current memory usage
    - **if the value can go down, it is a gauge**
  - **histogram** – samples observations and counts them in a configurable buckets
    - calculate quantiles on the **server side** and **expose them using histogram_quantile() function**
  - **summary**
    - calculate quantiles on the **client side** and **expose them directly**
    - https://prometheus.io/docs/practices/histograms/#histograms-and-summaries

Prometheus

- an endpoint you can scrape is called an instance
- a collection of instances with the same purpose is called a job
- example:
    - job: my-microservice
    - instances:
        - host1:8080
        - host2:8080
- pull model over HTTP
- pushing model is also possible (via gateway)

Prometheus



```
# HELP jvm_buffer_memory_used_bytes An estimate of the memory that the Java virtual machine is using for this buffer pool
# TYPE jvm_buffer_memory_used_bytes gauge
jvm_buffer_memory_used_bytes{application="spring-boot-metrics",id="mapped - 'non-volatile memory'",} 0.0
jvm_buffer_memory_used_bytes{application="spring-boot-metrics",id="mapped",} 0.0
jvm_buffer_memory_used_bytes{application="spring-boot-metrics",id="direct",} 90112.0
# HELP jvm_buffer_count_buffers An estimate of the number of buffers in the pool
# TYPE jvm_buffer_count_buffers gauge
jvm_buffer_count_buffers{application="spring-boot-metrics",id="mapped - 'non-volatile memory'",} 0.0
jvm_buffer_count_buffers{application="spring-boot-metrics",id="mapped",} 0.0
jvm_buffer_count_buffers{application="spring-boot-metrics",id="direct",} 11.0
# HELP executor_pool_size_threads The current number of threads in the pool
# TYPE executor_pool_size_threads gauge
executor_pool_size_threads{application="spring-boot-metrics",name="applicationTaskExecutor",} 0.0
# HELP tomcat_sessions_rejected_sessions_total
# TYPE tomcat_sessions_rejected_sessions_total counter
tomcat_sessions_rejected_sessions_total{application="spring-boot-metrics",} 0.0
# HELP executor_pool_max_threads The maximum allowed number of threads in the pool
# TYPE executor_pool_max_threads gauge
executor_pool_max_threads{application="spring-boot-metrics",name="applicationTaskExecutor",} 2.147483647E9
# HELP api_books_get_total a number of requests to /api/books endpoint
# TYPE api_books_get_total counter
api_books_get_total{application="spring-boot-metrics",title="all",} 1.0
api_books_get_total{application="spring-boot-metrics",title="Domain Driven Design",} 1.0
# HELP http_server_requests_seconds
# TYPE http_server_requests_seconds summary
http_server_requests_seconds_count{application="spring-boot-metrics",error="none",exception="none",method="GET",outcome="SUCCESS",status="200",uri="/api/books",} 2.0
http_server_requests_seconds_sum{application="spring-boot-metrics",error="none",exception="none",method="GET",outcome="SUCCESS",status="200",uri="/api/books",} 0.049084167
http_server_requests_seconds_count{application="spring-boot-metrics",error="none",exception="none",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 1245.0
http_server_requests_seconds_sum{application="spring-boot-metrics",error="none",exception="none",method="GET",outcome="SUCCESS",status="200",uri="/actuator/prometheus",} 3.53839131
http_server_requests_seconds_count{application="spring-boot-metrics",error="none",exception="none",method="GET",outcome="CLIENT_ERROR",status="404",uri="/**",} 13.0
http_server_requests_seconds_sum{application="spring-boot-metrics",error="none",exception="none",method="GET",outcome="CLIENT_ERROR",status="404",uri="/**",} 0.088439586
```

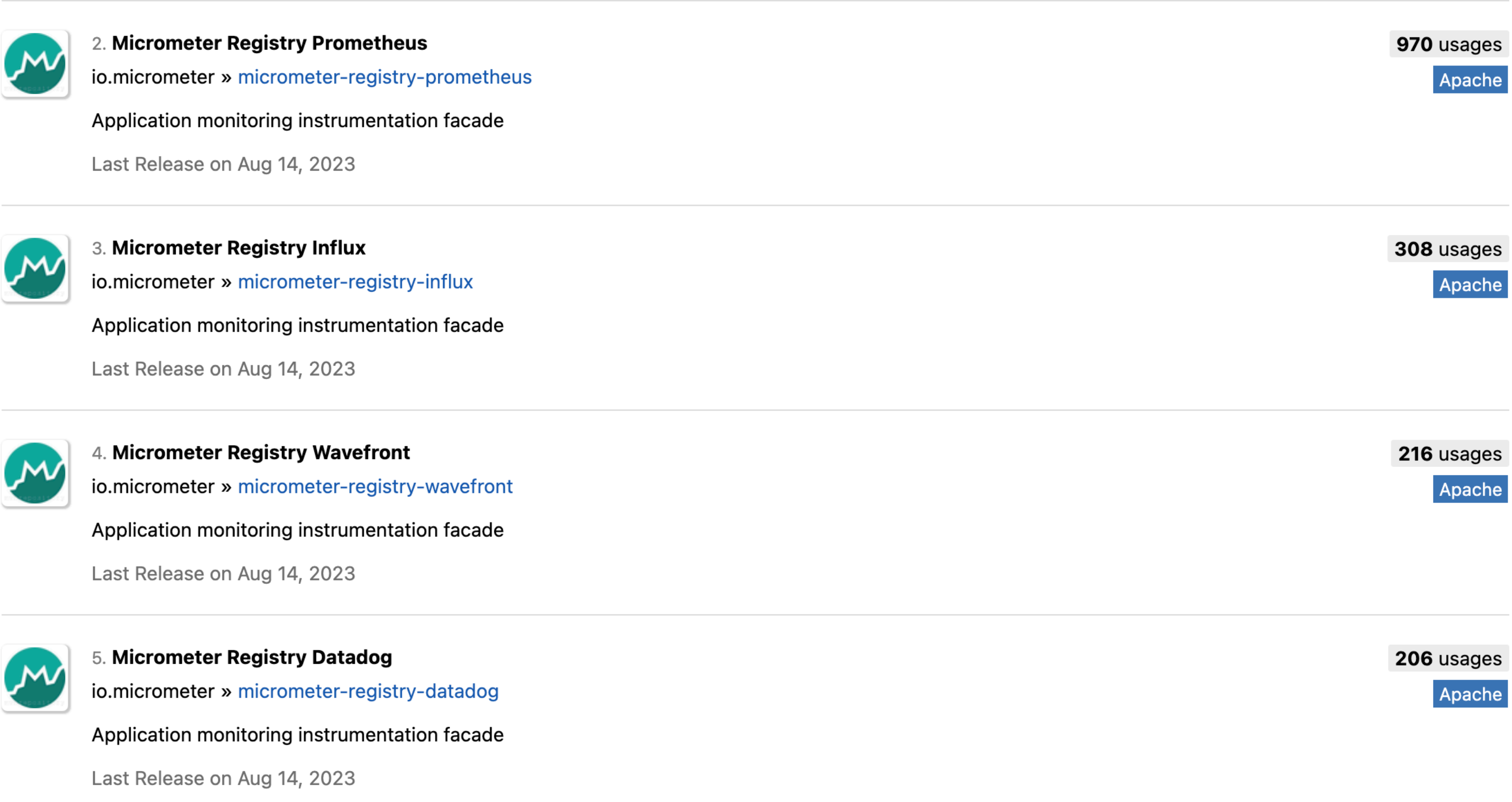
Micrometer

- https://micrometer.io
- vendor-neutral application observability facade
- SLF4J, but for observability
- … but why?
  - alternatives
    - Graphite, InfluxDB, DataDog
  - different metrics format
  - usage of more than one monitoring system
- its primary focus is on exposing metrics from within the application itself

MICROMETER

Micrometer

https://mvnrepository.com/artifact/io.micrometer



2. **Micrometer Registry Prometheus**

io.micrometer » micrometer-registry-prometheus

Application monitoring instrumentation facade

Last Release on Aug 14, 2023

970 usages

Apache

3. **Micrometer Registry Influx**

io.micrometer » micrometer-registry-influx

Application monitoring instrumentation facade

Last Release on Aug 14, 2023

308 usages

Apache

4. **Micrometer Registry Wavefront**

io.micrometer » micrometer-registry-wavefront

Application monitoring instrumentation facade

Last Release on Aug 14, 2023

216 usages

Apache

5. **Micrometer Registry Datadog**

io.micrometer » micrometer-registry-datadog

Application monitoring instrumentation facade

Last Release on Aug 14, 2023

206 usages

Apache

Prometheus vs Micrometer

	Prometheus	Micrometer
Purpose	Standalone monitoring and alerting system	Library to instrument your code with metrics, with a vendor-neutral interface
Data collection	Scraps data from various endpoints	Defines and records data in a vendor-neutral interface. Does not scrap data
Metric types	Defined in Prometheus documentation, stored in a specific format	Vendor-neutral interfaces for timers, gauges, counters, distribution summaries, and long task timers
Alerting	Built-in support	Not an alerting system
Ecosystem	Mature ecosystem which supports integration with other software	Focues on providing standard API for metrics in Java applications

Grafana

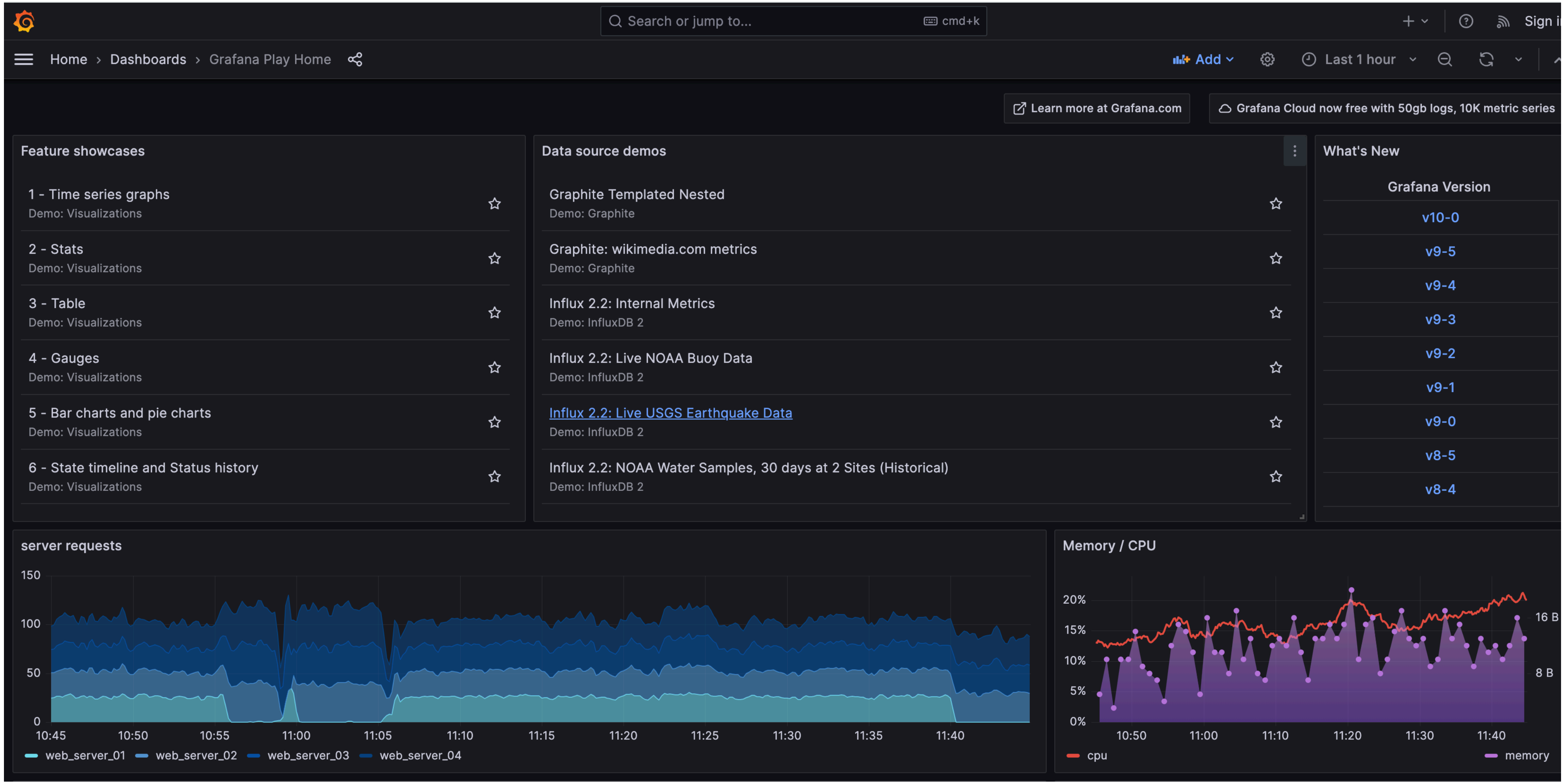
- https://grafana.com
- analytics, data-visualization and monitoring solution tool
- open source
- consolidate data from different systems into a single dashboard
- plugins – https://grafana.com/grafana/plugins
- dashboards - https://grafana.com/grafana/dashboards
- you can monitor…
  - JVM, databases, Kubernetes, RabbitMQ and many others
  - logs & traces
  - custom, “business” metrics
  - and many others
- alerts

Prometheus vs Grafana

- often used together
  - Prometheus for data collection, aggregations and storage
  - Grafana is primarily focused on data visualization and dashboard creation
- Grafana
  - can display data from different sources on a single dashboard
  - has more user-friendly interface
  - has better alerting solutions
  - available plugins, dashboards – rich extensibility
  - active community
  - centralized monitoring hub

Grafana

- https://play.grafana.org



Search or jump to...
cmd+k
Sign i
Home › Dashboards › Grafana Play Home
Add
Last 1 hour
Learn more at Grafana.com
Grafana Cloud now free with 50gb logs, 10K metric series

Feature showcases

- 1 - Time series graphs
  Demo: Visualizations
- 2 - Stats
  Demo: Visualizations
- 3 - Table
  Demo: Visualizations
- 4 - Gauges
  Demo: Visualizations
- 5 - Bar charts and pie charts
  Demo: Visualizations
- 6 - State timeline and Status history
  Demo: Visualizations

Data source demos

- Graphite Templated Nested
  Demo: Graphite
- Graphite: wikimedia.com metrics
  Demo: Graphite
- Influx 2.2: Internal Metrics
  Demo: InfluxDB 2
- Influx 2.2: Live NOAA Buoy Data
  Demo: InfluxDB 2
- Influx 2.2: Live USGS Earthquake Data
  Demo: InfluxDB 2
- Influx 2.2: NOAA Water Samples, 30 days at 2 Sites (Historical)
  Demo: InfluxDB 2

What's New

Grafana Version
v10-0
v9-5
v9-4
v9-3
v9-2
v9-1
v9-0
v8-5
v8-4

server requests

Memory / CPU

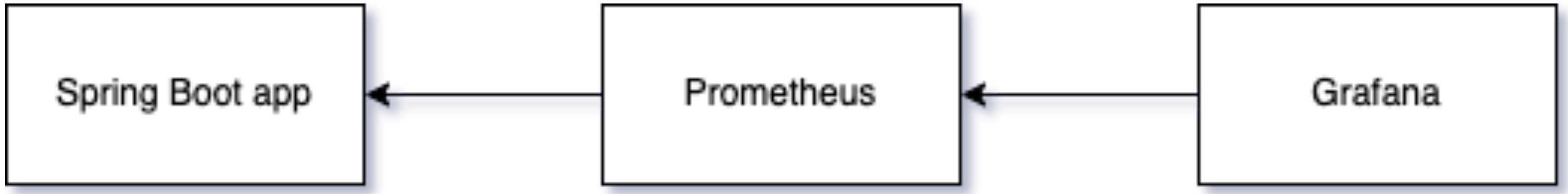
Why software monitoring is important

- early issues detection
- improved reliability
- performance optimization
- capacity planning
- troubleshooting
- user experience
- alerting automation
- … and many others

Why software monitoring is important

- *business* questions
  - how many orders did we receive today?
  - our clients complain about an error during checkout – why we don’t know about it?
  - a number of active sessions at the moment
  - …
- *technical* aspects
  - a number of HTTP 500s responses
  - a number of available DB connections
  - a number of messages on RabbitMQ DLQ/ParkingLot
  - memory usage
  - …

Architecture



Spring Boot app

Prometheus

Grafana

Demo

Not covered by demo

- @Timed annotation
- LongTimer – for long tasks, reports progress in the meantime
- DistributionSummary – similar to timer structurally, but records values that do not represent a unit of time
  - e.g., response size in bytes

Best practices

Best practices

- measure key metrics
- standardize naming conventions
- set meaningful alerts
- use percentiles
- automate where possible
- continuously improve
- think about a new metrics when building a new feature

USE method

- USE
  - Utilization - percent time the resource is busy (CPU usage)
  - Saturation - amount of work a resource has to do (queue length)
  - Errors - count of error events
- **for every resource, check utilization, saturation, and errors**
- good for hardware resources in infrastructure (CPU, memory, …)
- it is a methodology for analyzing the performance of any system

RED method

- RED
  - Rate - requests per second
  - Errors - number of requests that are failing
  - Duration - amount of time these requests take, distribution of latency measurements
- good for services, alerting, SLAs
- it is a good proxy to how happy your customers will be

The Four Golden Signals

- The four golden signals of monitoring according to Google Site Reliability Engineering
  - latency - time taken to serve a request
    - per successful/failed request, not only “in general”
  - traffic - how much demand is placed on your system
    - HTTP request per second (by static/dynamic content), network I/O, concurrent sessions
  - errors - rate of requests that are failing
    - 500s, 200s but with the “wrong” content
  - saturation - how “full” your system is
    - constrained resources – e.g. memory
    - can your system handle 10% more traffic?
    - predictions – your hard drive is likely to be filled in X hours
- for user-facing systems

Cardinality

- the number of series stored in Prometheus in a timeframe
  - e.g., a label containing HTTP methods
- more data = more resource usage, more latency, slower queries, …
- example histogram displaying the duration of HTTP requests
  - 6 instances, 10 histogram's buckets = 60
  - 20 endpoints = 1200
  - 10 HTTP response codes = 12 000
  - 4 HTTP methods = 48 000
- label per title – might not be the best idea
- but for some series it might be ok

Cardinality

- operational data (must be fast and reliable)
  - is the service running correctly?
  - does the service meet SLA?
- telemetry data (less sensitive in terms of latency)
  - for further, more detailed investigation
- **Prometheus is designed for operational data**
- metrics based monitoring gives you a **real time information** about your system
  - e.g., store data from the last 14 days
  - do not use metrics to describe trends from the last 5 years
- more granular data → go the the logs/ DB warehouse

Aggregation - rate(metric[time-range])

- rate(http_requests_total{job="api-server"}[5m])
  - returns the per-second rate of HTTP requests as measured over the last 5 minutes
- small time-range is useful to see a system’s state at the moment
- big time-range is useful to see a trend
- always use the same time-range to find some correlation or when comparing values

Metrics

- use the single unit (do not mix seconds with milliseconds)
- use the base units (seconds, bytes, meters) - https://prometheus.io/docs/practices/naming/#base-units
- use consistent naming convention
  - **prometheus**_notifications_**total**
  - prefix related to the domain that metrics belongs to
  - suffix describing the unit - _total, _seconds, _bytes, _info
- represent the same logical thing between all label dimensions
- labels should differentiate the characteristics of the thing that is being measured
  - **remember about cardinality**
- have a default, common value – do not set null to a label value
  - NA/other/all/0

ALTIMETRIK™

SIMPLIFY TECHNOLOGY | AMPLIFY POSSIBILITY

Thank you

Aleksander Kołata