.msg .consulting .solutions .partnership

Ops for Developers – Monitor your Java application with Prometheus

Alexander Schwartz, Principal IT Consultant

CloudNativeCon + KubeCon Europe 2017 – 30 March 2017

Ops for Developers – Monitor your Java application with Prometheus

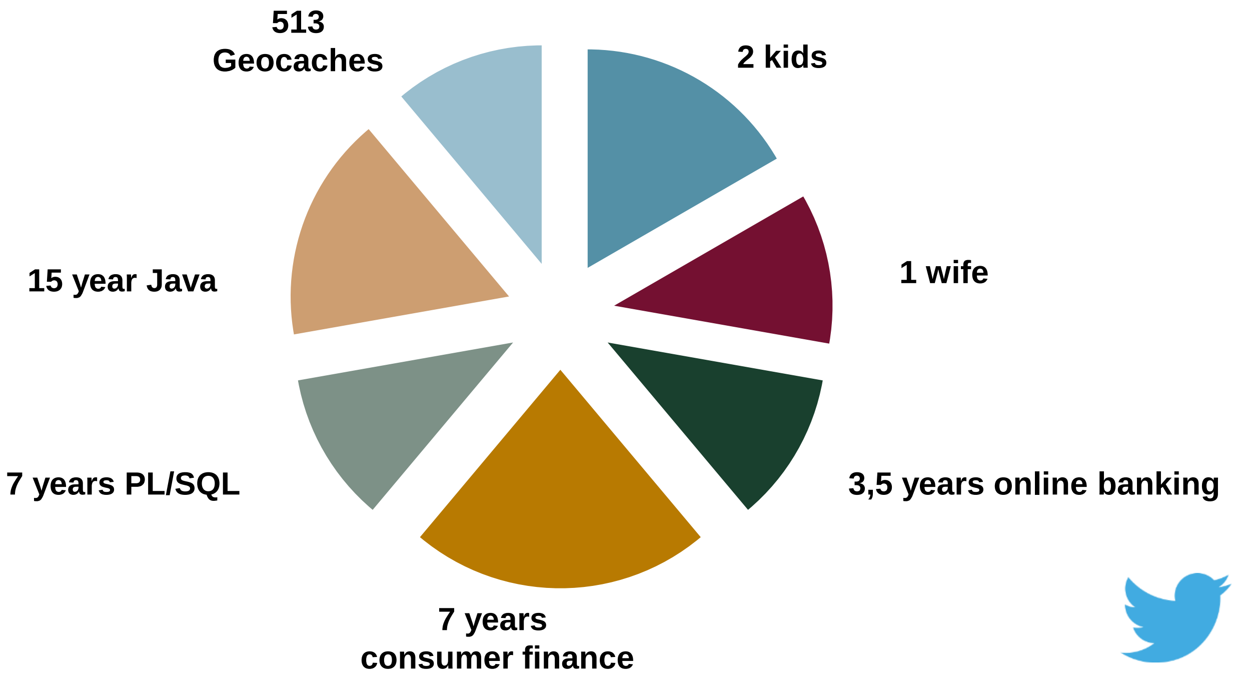
1 About Prometheus

2 Setup

3 How to...

4 Prometheus works for Developers (and Ops)

About me – Principal IT Consultant @ msg Travel & Logistics



@ahus1de

Prometheus Monitoring Retreat

What to expect:

- Experiment and setup Prometheus monitoring on your own laptop or in the cloud
- Exchange experiences and try out new exporters
- Share tips and tricks on creating dashboard with Grafana

Location:

- Frankfurt/Main (DE) Area

Date:

- either Friday June 23rd or Saturday June 24th

Pre-Registration: http://eepurl.com/cIjNr9

@ahus1de

Ops for Developers – Monitor your Java application with Prometheus

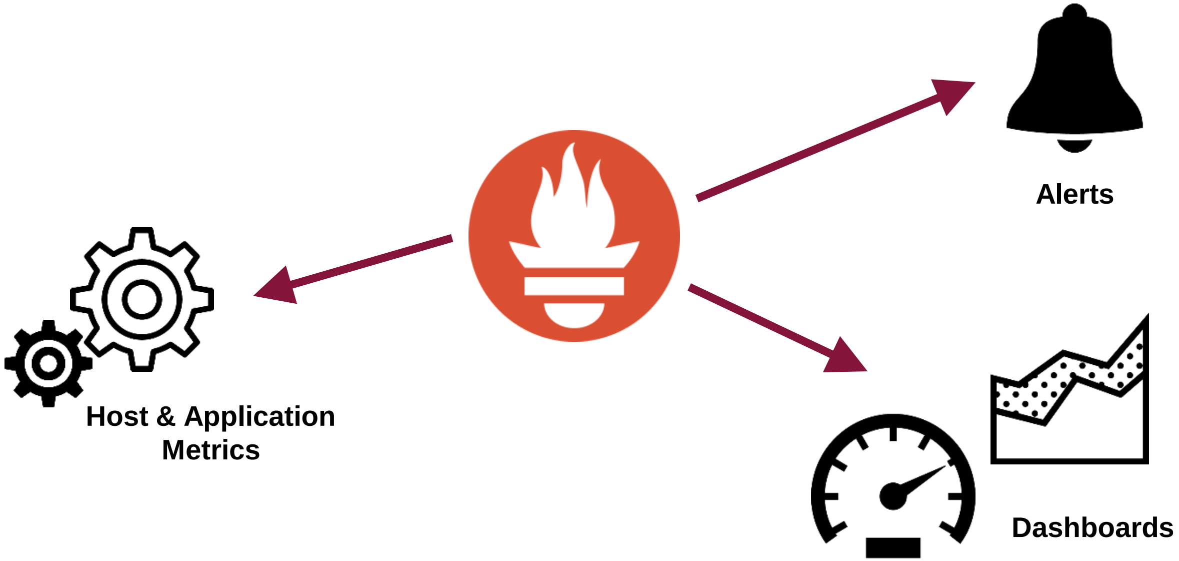
1 **About Prometheus**

2 Setup

3 How to...

4 Prometheus works for Developers (and Ops)

Monitoring



Prometheus is a Monitoring System and Time Series Database

Prometheus is an opinionated solution

for

instrumentation, collection, storage
querying, alerting, dashboards, trending

1. PromCon 2016: Prometheus Design and Philosophy - Why It Is the Way It Is - Julius Volz
   https://youtu.be/4DzoajMs4DM / https://goo.gl/1oNaZV

Ops for Developers – Monitor your Java application with Prometheus

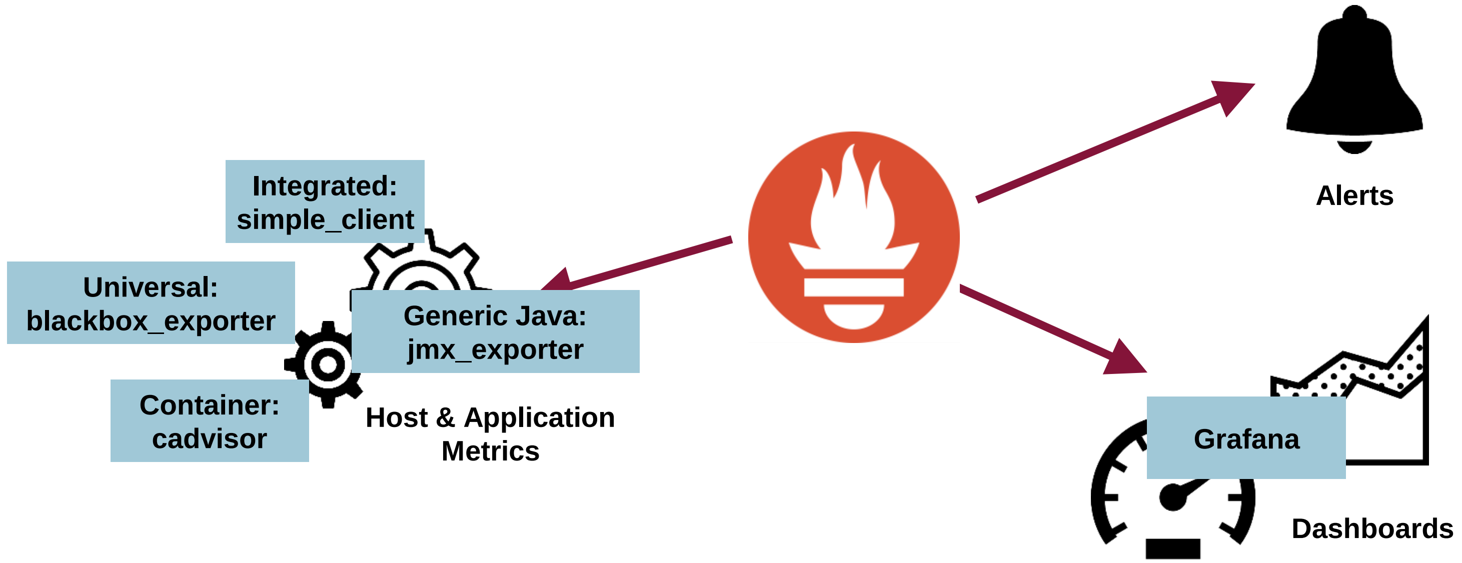
1 About Prometheus

2 **Setup**

3 How to...

4 Prometheus works for Developers (and Ops)

Technical Building Blocks



Ops for Developers – Monitor your Java application with Prometheus

1 About Prometheus

2 Setup

3 How to...

4 Prometheus works for Developers (and Ops)

Information about your containers

Presented by: cadvisor

RAM Usage per container:

Variable:	container_memory_usage_bytes
Expression:	container_memory_usage_bytes{name=~'.+',id=~'/docker/.*'}

CPU Usage per container:

Variable:	container_cpu_usage_seconds_total

Expression:	rate(container_cpu_usage_seconds_total [30s])
	irate(container_cpu_usage_seconds_total [30s])
	sum by (instance, name) (irate(container_cpu_usage_seconds_total{name=~'.+'} [15s]))

Information about your JVM

Presented by: Java simple_client

RAM Usage of Java VM:

Variable:	jvm_memory_bytes_used
Expressions:	sum by (instance, job) (jvm_memory_bytes_used) sum by (instance, job) (jvm_memory_bytes_committed)

CPU seconds used by Garbage Collection:

Variable:	jvm_gc_collection_seconds_sum
Expression:	sum by (job, instance) (irate(jvm_gc_collection_seconds_sum [10s]))
Test:	ab -n 100000 -c 10 http://192.168.23.1:8080/manage/metrics

Information about your JVM

Add a Configuration to Spring Boot to serve standard JVM metrics using a custom URL.

```
@Configuration
public class MetricsApplicationConfig {

    @Bean
    public synchronized ServletRegistrationBean metrics() {
        DefaultExports.initialize();
        return new ServletRegistrationBean(new MetricsServlet(),
                "/manage/metrics");
    }
}
```

Information about your Spring Application

Presented by: Java simple_client, Dropwizard Metrics/Spring Metrics

Timings of a method call:

Java Annotation:	@Timed
Variables:	countedCallExample_snapshot_mean countedCallExample_snapshot_75thPercentile countedCallExample_snapshot_98thPercentile
Test:	ab -n 10000 -c 10 http://192.168.23.1:8080/api/countedCall

Information about your JVM

Add a Configuration to Spring Boot to serve standard JVM metrics using a custom URL.

```
@Configuration
@EnableMetrics(proxyTargetClass = true)
public class MetricsApplicationConfig extends MetricsConfigurerAdapter {

      /* ... */
}
```

Information about your Spring Application

Add @Timed annotations to any method of any Bean to collect metrics

```
@Component
public class RestEndpoint {

    @Path("countedCall")
    @GET
    @Timed(absolute = true, name = "countedCallExample")
    public Response countedCall() throws InterruptedException {
        /* ... */
        return Response.ok("ok").build();
    }

}
```

Information about your External Interfaces – Hystrics Metrics

Presented by: Java simple_client, Hystrix/Spring, Soundcloud's HystrixMetricsCollector

Hystrix Metrics:

Java Annotation:	@HystrixCommand
Beware:	No Prometheus-Style Histograms supported
Test:	ab -n 10000 -c 10 http://192.168.23.1:8080/api/externalCall
Variables:	hystrix_command_count_success, hystrix_command_count_exceptions_thrown hystrix_command_latency_total_*
Expressions:	irate(hystrix_command_count_success [15s]) irate(hystrix_command_count_exceptions_thrown [15s]) hystrix_command_latency_total_mean hystrix_command_latency_total_percentile_90 hystrix_command_latency_total_percentile_99

Information about your External Interfaces – Hystrics Metrics

Register the Hystrix Publisher and add @HystrixCommand for resilience and timing of external calls.

```
HystrixPrometheusMetricsPublisher.register();
```

```
@Component
public class ExternalInterfaceAdapter {

    @HystrixCommand(commandKey = "externalCall", groupKey = "interfaceOne")
    public String call() {
        /* ... */
    }
}
```

Information about your External Interfaces – Hystrix Histograms

Presented by: Java simple_client, Hystrix, Your own Collector

Hystrix Metrics:

Java Annotation:	@HystrixCommand + your own collector
Test:	ab -n 10000 -c 10 http://192.168.23.1:8080/api/externalCall
Variables:	hystrix_command_latency_execute.*
Expressions:	{ __name__ =~ "hystrix_command_latency_execute_(bucket|sum|count)" }

```
histogram_quantile(0.95,
          sum(rate(hystrix_command_latency_execute_bucket[5m]))
              by (le, command_name, command_group))
```

Information about your External Interfaces – Hystrix Histograms

Access the stream of completed commands to calculate Histograms “Prometheus style” that can be aggregated over several instances.

```
Histogram.Child histogramLatencyTotal
                        = addHistogram("latency_total", latencyTotalDoc);

HystrixCommandCompletionStream.getInstance(commandKey)
        .observe()
        .subscribe(hystrixCommandCompletion -> {
             histogramLatencyTotal.observe
                     (hystrixCommandCompletion.getTotalLatency() / 1000.0);
             });
```

Information about your Spring Servlet container

Presented by: your own Java metric provider

Tomcat Connector:

Java Class:	Write your own: TomcatStatisticsCollector
Variables:	tomcat_thread_pool_current_thread_count tomcat_thread_pool_current_threads_busy

Tomcat DB Connection Pool:

Java Class:	Write your own: DatasourceStatisticsCollector
Variables:	tomcat_datasource_active tomcat_datasource_idle tomcat_datasource_max_idle

Information about your Spring Servlet Container

```
public class DatasourceStatisticsCollector extends Collector {

    /* ... */

    @Override
    public List<MetricFamilySamples> collect() {
        /* ... */
        result.add(buildGauge("active", "number of connections in use",
            labelNames, labelValues, tomcatDS.getActive()));
        return result;
    }

}
```

```
new DatasourceStatisticsCollector(dataSource).register();
```

Information about your Vert.x application

Presented by: Java Simple Client for Vert.x

Internal Event Bus:

Variables:	vertx_eventbus_messages_sent_total vertx_eventbus_messages_pending vertx_eventbus_messages_delivered_total vertx_eventbus_messages_reply_failures_total

HTTP Server metrics:

Variables:	vertx_http_servers_requests_count vertx_http_servers_open_netsockets
Test:	ab -n 100000 -c 100 http://192.168.23.1:8081/manage/metrics

Information about your Vert.x application

```
// During Setup
vertx = Vertx.vertx(new VertxOptions().setMetricsOptions(
            new DropwizardMetricsOptions()
                .setRegistryName("vertx")
                .addMonitoredHttpClientEndpoint(
                    new Match().setValue(".*").setType(MatchType.REGEX))
                .setEnabled(true)
        ));

DefaultExports.initialize();
new DropwizardExports(SharedMetricRegistries.getOrCreate("vertx")).register();

// When starting up Routes and a HTTP Server
final Router router = Router.router(vertx);
router.route("/metrics").handler(new MetricsHandler());
```

Alerting with Prometheus

Any expression can be used for alerting

```
ALERT gc_cpu_warning
  IF (sum by (job, instance) (irate(jvm_gc_collection_seconds_sum [10s]))) * 100 > 70
  FOR 5m
  LABELS {severity="warning"}
  ANNOTATIONS  {summary=“High CPU GC usage on {{ $labels.job }}: instance {{$labels.instance}}
                              more than 70 % on one CPU."}
```

Ops for Developers – Monitor your Java application with Prometheus

1 About Prometheus

2 Setup

3 How to...

4 **Prometheus works for Developers (and Ops)**

Prometheus is “friendly tech” in your environment

Team friendly

- Every team can run its own Prometheus instance to monitor their own and neighboring systems
- Flexible to collect and aggregate the information that is needed

Coder and Continuous Delivery friendly

- All configurations (except dashboard) are kept as code and are guarded by version control
- Client libraries available to provide metrics directly or via adapters to existing metrics collectors
- Changes can be tested locally and easily staged to the next environment

Simple Setup

- Go binaries for *prometheus* and *alertmanager* available for all major operating systems
- Several existing exporters for various needs

Links

Prometheus:
https://prometheus.io

Prometheus Simple (Java) Client:
https://github.com/prometheus/client_java

Hystrix
https://github.com/Netflix/Hystrix

Dropwizard Metrics
http://metrics.dropwizard.io

Spring Metrics
http://metrics.ryantenney.com

Julius Volz @ PromCon 2016
Prometheus Design and Philosophy - Why It Is the Way It Is
https://youtu.be/4DzoajMs4DM
https://goo.gl/1oNaZV

@ahus1de

Alexander Schwartz
Principal IT Consultant

+49 171 5625767
alexander.schwartz@msg-systems.com

@ahus1de

msg systems ag (Headquarters)
Robert-Buerkle-Str. 1, 85737 Ismaning
Germany

www.msg-systems.com

.consulting .solutions .partnership