Monitoring for Developers with Prometheus and Grafana

Isaac Rubio Torres / July 2018

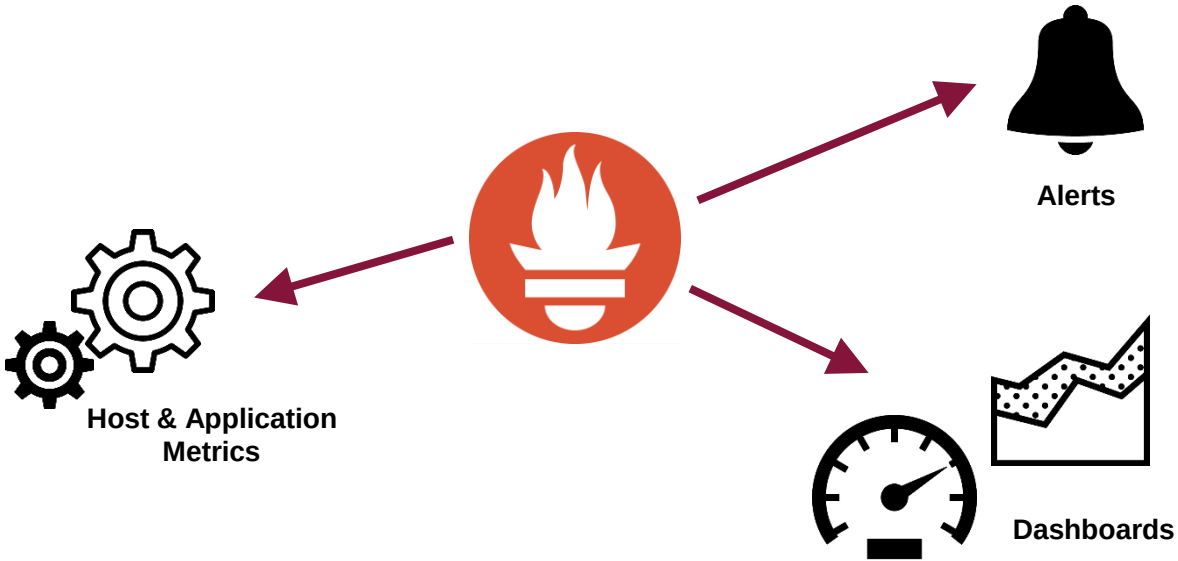
Monitoring for Developers with Prometheus and Grafana

1. Prometheus Manifesto
2. Setup
3. How to...
4. Prometheus works for Developers (and Ops)

Monitoring for Developers with Prometheus and Grafana

1. **Prometheus Manifesto**
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

Prometheus values …

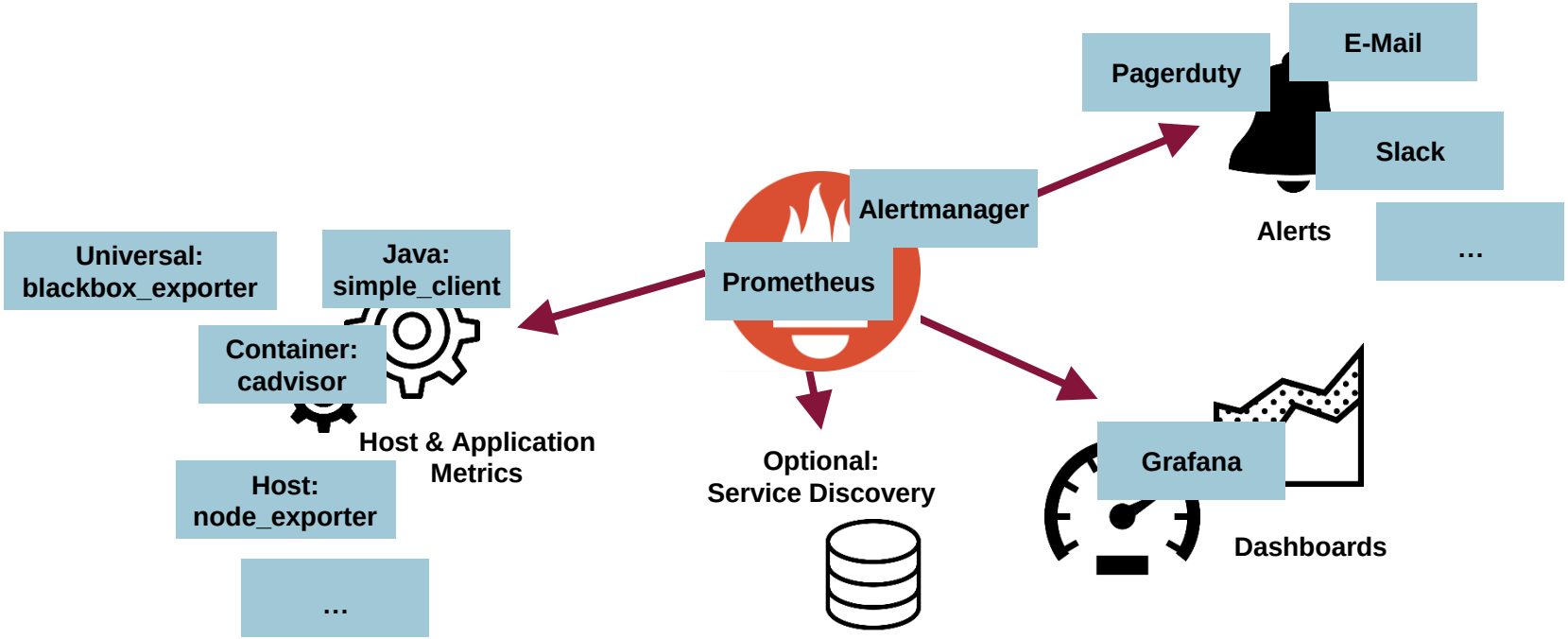
operational systems monitoring (not only) for the cloud	over	raw logs and events, tracing of requests, magic anomaly detection, accounting, SLA reporting
simple single node w/ local storage for a few weeks	over	horizontal scaling, clustering, multitenancy
configuration files	over	Web UI, user management
pulling data from single processes	over	pushing data from processes, aggregation on nodes
NoSQL query & data massaging multidimensional data everything as float64	over	point-and-click configurations, data silos, complex data types

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

Monitoring for Developers with Prometheus and Grafana

1. Prometheus Manifesto
2. **Setup**
3. How to...
4. Prometheus works for Developers (and Ops)

Setup


Pagerduty
E-Mail
Slack
Alerts
…
Alertmanager
Prometheus
Universal:
blackbox_exporter
Java:
simple_client
Container:
cadvisor
Host & Application
Metrics
Host:
node_exporter
…
Optional:
Service Discovery
Grafana
Dashboards

Technical Building Blocks

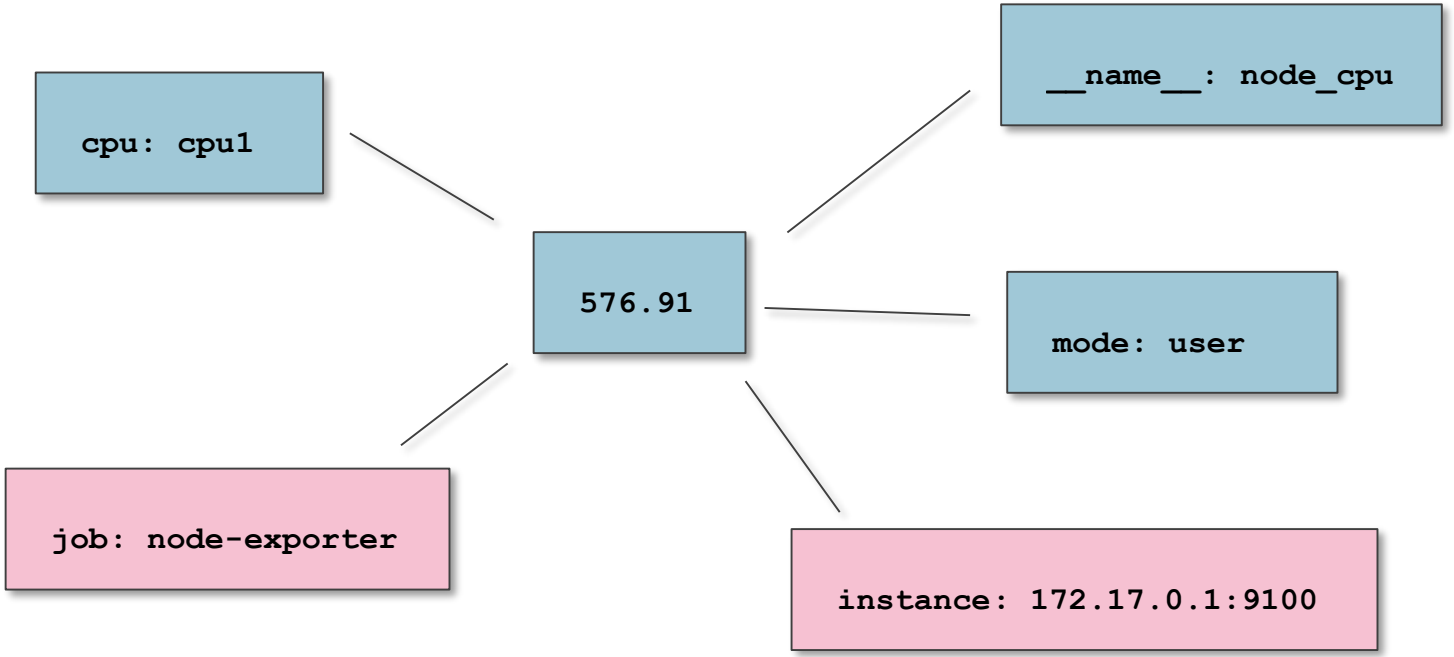

Disk Total
84.4 GiB
Total Memory
15.7 GiB
Load
0.5000%
Uptime
26.1 week
Scrape CPU use
0.04%
CPU Usage
13:30
14:00
14:30
15:00
15:30
user
iowait
softirq
system
user
iowait
softirq
system
user
user
Memory usage
17 GiB
15 GiB
13 GiB
11 GiB
9 GiB
13:00
13:30
14:00
14:30
15:00
15:30
Used
Buffers
Cached
Free
Available memory
30%
Swap Activity
13:30
14:00
14:30
15:00
15:30
Network Usage
500 kBps
0 Bps
-500 kBps
13:00
13:30
14:00
14:30
15:00
15:30
docker0 outbound
eno16777984 outbound
docker0 inbound
eno16777984 inbound
Free Filesystem Space (Lowest)
78.4%
5 most used filesystems
13:30
14:00
14:30
15:00
15:30
Uso de Filesystem
Metric
Current
Max
/boot
21.60%
21.60%
/
17.56%
17.56%
Tamaño de Filesystem
Metric
Current
/
84 GiB
/boot
497 MiB
IOPs
10 ms
8 ms
6 ms
4 ms
2 ms
0 ms
13:10
13:20
13:30
13:40
13:50
14:00
14:10
14:20
14:30
14:40
14:50
15:00
15:10
15:20
15:30
IOPS
io time
Tiempo de IO por device (ms/s)
0.6 ms
0.4 ms
0.2 ms
0 ms
12:40
12:50
13:00
13:10
13:20
13:30
13:40
13:50
14:00
14:10
14:20
14:30
14:40
14:50
15:00
15:10
15:20
15:30
dm-0
dm-1
dm-2
dm-4
dm-5
dm-6
dm-7
dm-8
sda
sdb

```
scrape_configs:
 - job_name: 'node-exporter'
    scrape_interval: 5s
    static_configs:
      - targets: ['172.17.0.1:9100']
```

CPU Metric as exported by the Node Exporter

```
# HELP node_cpu Seconds the cpus spent in each mode.
# TYPE node_cpu counter
node_cpu{cpu="cpu0",mode="guest"} 0
node_cpu{cpu="cpu0",mode="idle"} 4533.86
node_cpu{cpu="cpu0",mode="iowait"} 7.36
...
node_cpu{cpu="cpu0",mode="user"} 445.51
node_cpu{cpu="cpu1",mode="guest"} 0
node_cpu{cpu="cpu1",mode="idle"} 4734.47
...
node_cpu{cpu="cpu1",mode="iowait"} 7.41
node_cpu{cpu="cpu1",mode="user"} 576.91
...
```

Multidimensional Metric as stored by Prometheus



Calculations based on metrics

Metric:

node_cpu: Seconds the CPUs spent in each mode (Type: Counter).

What percentage of a CPU is used per core?

```
1 - rate(node_cpu{mode='idle'} [5m])
```

What percentage of a CPU is used per instance?

```
avg by (instance) (1 - rate(node_cpu{mode='idle'} [5m]))
```

1. Prometheus Manifesto
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: irate(container_cpu_usage_seconds_total [30s]) 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]))

Information about your JVM

Add a Configuration to Spring Boot to serve standard JVM metrics using /prometheus actuator endpoint.

```
@Configuration
@EnablePrometheusEndpoint
public class ApplicationConfig {

    @PostConstruct
    public void metrics() {
        DefaultExports.initialize();
        /* ... */
    }
}
```

Information about your Application Metrics

Presented by: Java simple_client and Spring

Timings of a method call:

Java Annotation: @PrometheusTimeMethod(name = "example", help = "...") Variables:

example_count

example_sum

Information about your Application Metrics

Add a Configuration to collect Prometheus timings from Annotations.

```
@Configuration
@EnablePrometheusTiming
public class MetricsApplicationConfig {

      /* ... */
}
```

Information

about your Application Metrics

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

```
@Component
public class RestEndpoint {

    @Path("countedCall")
    @GET
    @PrometheusTimeMethod(name = "example", help = "...")
    public Response countedCall() throws InterruptedException {
        /* ... */
        return Response.ok("ok").build();
    }

}
```

Information about your External Interfaces

Presented by: Java simple_client, Hystrix/Spring

Hystrix Metrics:

Java Annotation: @HystrixCommand

Variables: hystrix_command_total {command_name="externalCall", …}
hystrix_command_error_total {command_name="externalCall", …}

Expressions: histogram_quantile(0.99,
rate(hystrix_command_latency_execute_seconds_bucket[1m]))

External Interfaces – Hystrix Metrics

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

Information about your

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

```
@Component
public class ExternalInterfaceAdapter {

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

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

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

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());
```

Information about your Federation of Prometheus

Any Metric can be exported to other Prometheus instances

http://localhost/prometheus/federate?match[]={job=%22prometheus%22}

Alerting with Prometheus

Any expression can be used for alerting

```
alert: HDD_Alert_warning
        (1 - node_filesystem_free{mountpoint=~".*"} / node_filesystem_size{mountpoint=~".*"}) * 100 > 70
expr: for: 5m labels:
  severity: warning
annotations: summary: High disk usage on {{ $labels.instance }}: filesystem {{$labels.mountpoint}} more
  than 70 % full.
```

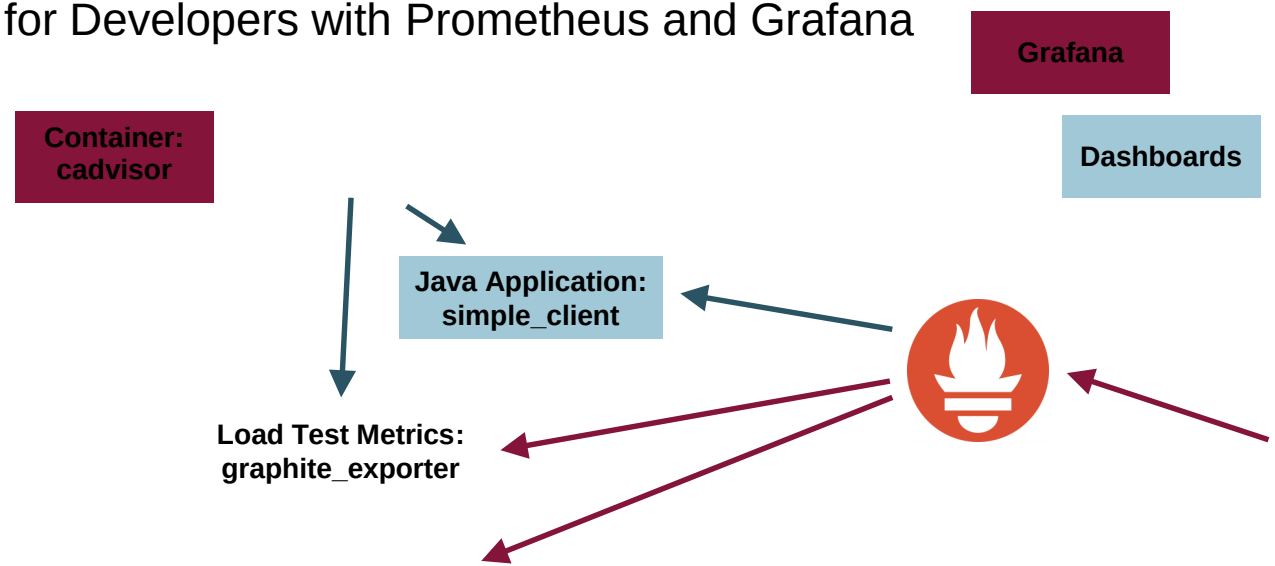
Setup of the Environment

Technical Building Blocks for Load Testing

Load Test:

Purple (including Prometheus): Provided as infrastructure in a testing environment
Blue: Setup and maintained by product team (developers/testers)

Monitoring for Developers with Prometheus and Grafana



1. Prometheus Manifesto
2. Setup
3. Howto...
4. **Prometheus works for Developers (and Ops)**

What to expect

Lessons learned

The approach worked well for us to pass the load tests:

- Load Tool metrics correlated with application and infrastructure metrics
- Inter-application communication captured by Hystrix
- Self-service functionality for product teams to add applications and metrics

… but to use the instrumentation also in production create awareness:

- Exported metrics should following Prometheus naming conventions
- Collector for Dropwizard Metrics can’t fill HELP text of metrics
- Counters and averages vs. histograms, summaries and percentiles
- Consistent use of USE Method (utilization – saturation – errors) or RED Method (rate – errors – duration) for metrics

1. http://www.brendangregg.com/usemethod.html
2. https://www.weave.works/blog/prometheus-and-kubernetes-monitoring-your-applications/

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
- Changes can be tested locally and easily staged to the next environment

Simple Setup

- Go binaries for *prometheus* and *alertmanager* available for major operating systems
- Client libraries for several languages available (also adapters to existing metrics libraries)
- Several existing exporters for various needs

Links

Prometheus:
https://prometheus.io

Java Simple Client
https://github.com/prometheus/client_java

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

Prometheus Hystrix Metrics Publisher
https://github.com/ahus1/prometheus-hystrix

Dropwizard Metrics
http://metrics.dropwizard.io

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

CAdvisor
https://github.com/google/cadvisor