APACHE: BIG_DATA EUROPE

14-16.11.16

SEVILLE, SPAIN

Performance Monitoring for the Cloud

Werner Keil

Director, Creative Arts & Technologies

@wernerkeil | @UnitAPI | wkeil@apache.org

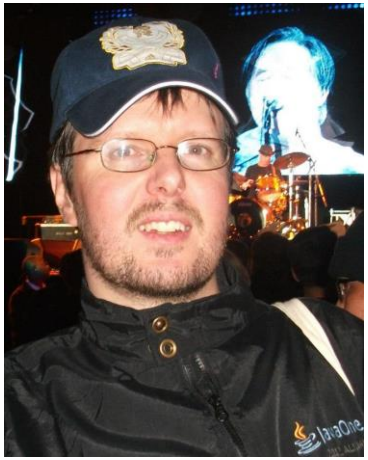
THE LINUX FOUNDATION

Agenda

- Introduction
- Performance Co-Pilot
- Dropwizard Metrics
- Apache Sirona
- StatsD
- Demo
- Conclusion

© 2016 Creative Arts & Technologies

Who am I?



- Consultant – Coach
- Creative Cosmopolitan
- Open Source Evangelist
- Software Architect
- Apache Committer
- JCP EC Member
- JSR 363 Co Spec Lead
- Java EE Guardian | DevOps Guy …

Twitter @wernerkeil | Email wkeil@apache.org

© 2016 Creative Arts & Technologies

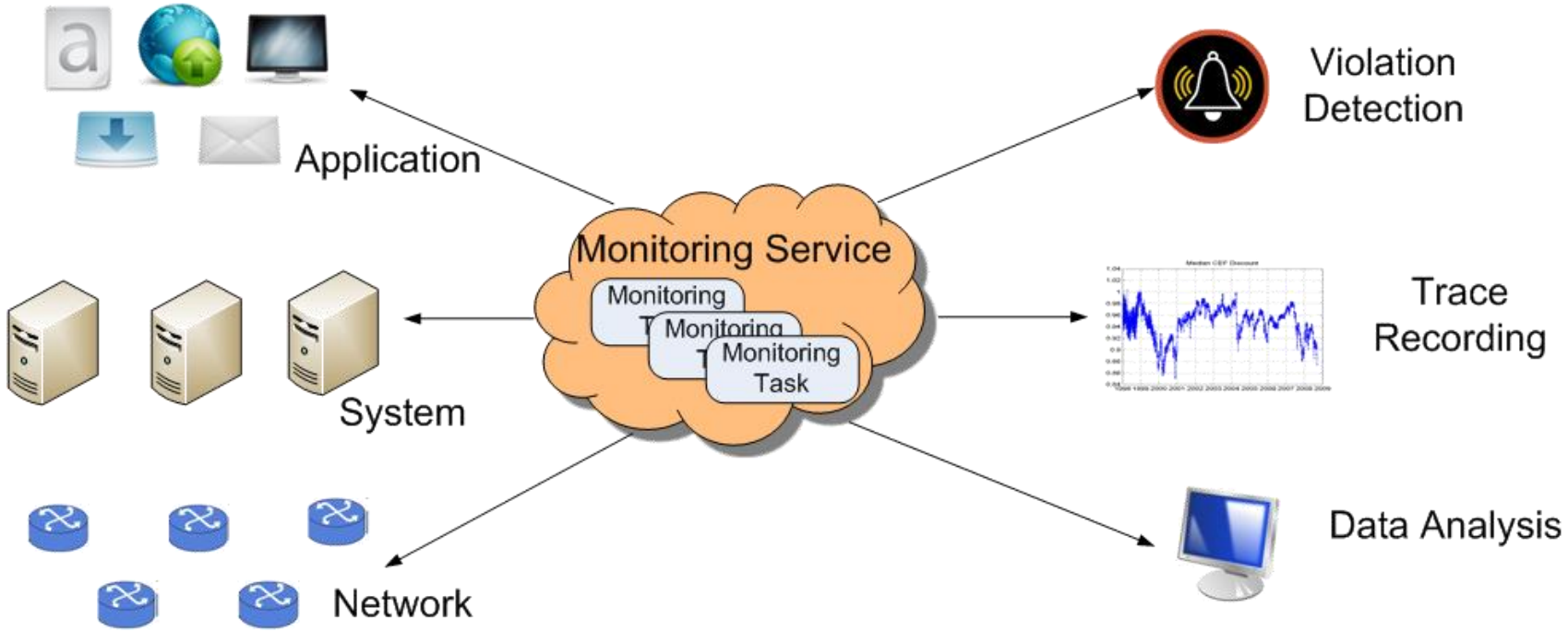
What is Monitoring?

Monitoring applications is observing, analyzing and manipulating the execution of these applications, which gives information about threads, CPU usage, memory usage, as well as other information like methods and classes being used.

A particular case is the monitoring of distributed applications, aka the Cloud where an the performance analysis of nodes and communication between them pose additional challenges.

© 2016 Creative Arts & Technologies

A high-level view of Cloud Monitoring



© 2016 Creative Arts & Technologies

Challenges at System Level

- Efficient Scalability
  - Supporting tens of thousands of monitoring tasks
  - Cost effective: minimize resource usage
- Monitoring QoS
  - Multi-tenancy environment
  - Minimize resource contention between monitoring tasks
- Implication of Multi-Tenancy
  - Monitoring tasks: adding, removing
  - Resource contention between monitoring tasks

© 2016 Creative Arts & Technologies

Performance vs Number of Hosts

60 items per host, update frequency once per minute

Number of hosts	Performance (values per second)
100	100
1000	1000
10000	10000

600 items per host, update frequency once per minute

Number of hosts	Performance (values per second)
100	1000
1000	10000
10000	100000

Monitoring Tips

- Regularly apply “Little’s Law” to all data... generic (queueing theory) form:

$$Q = \lambda R$$

- Length = Arrival Rate x Response Time
  - e.g. 10 MB = 2 MB/sec x 5 sec
- Utilization = Arrival Rate x Service Time
  - e.g. 20% = 0.2 = 100 msec/sec x 2 sec

© 2016 Creative Arts & Technologies

Types of Monitoring

Monitoring Logs

- Logstash
- Redis
- Elasticsearch
- Kibana Dashboard

Monitoring Performance

- Collectd
- Statsd
- PCP
- Graphite
- Database (eg: PSQL)
- Grafana Dashboard

© 2016 Creative Arts & Technologies

Monitoring Logs – Kibana Dashboard

© 2016 Creative Arts & Technologies

Monitoring Performance

How is this traditionally done?

- rsyslog/syslog-ng/journald
- top/iostat/vmstat/ps
- Mixture of scripting languages (bash/perl/python)
- Specific tools vary per platform
- Proper analysis requires more context

© 2016 Creative Arts & Technologies

Performance Co-Pilot

PCP

http://www.pcp.io

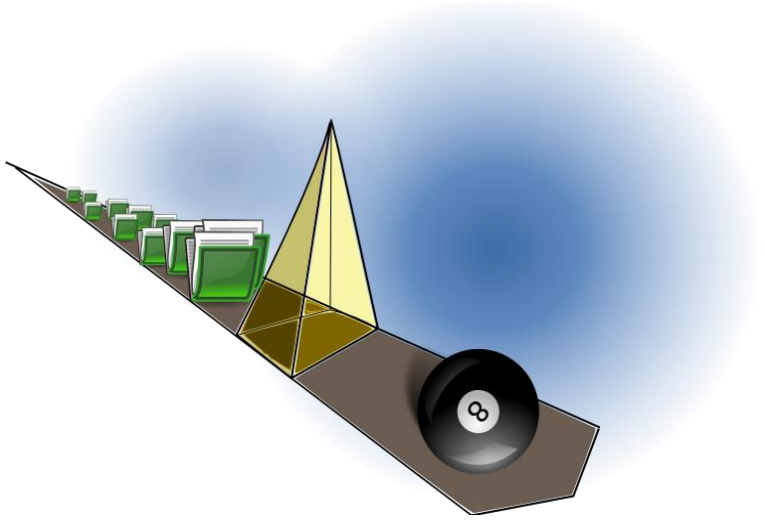
GitHub

https://github.com/performancecopilot

© 2016 Creative Arts & Technologies

What is PCP?

- Open source toolkit
- System-level analysis
- Live and historical
- Extensible (monitors, collectors)
- Distributed
- Unix-like component design
- Cross platform
- Ubiquitous units of measurement



© 2016 Creative Arts & Technologies

PCP Basics

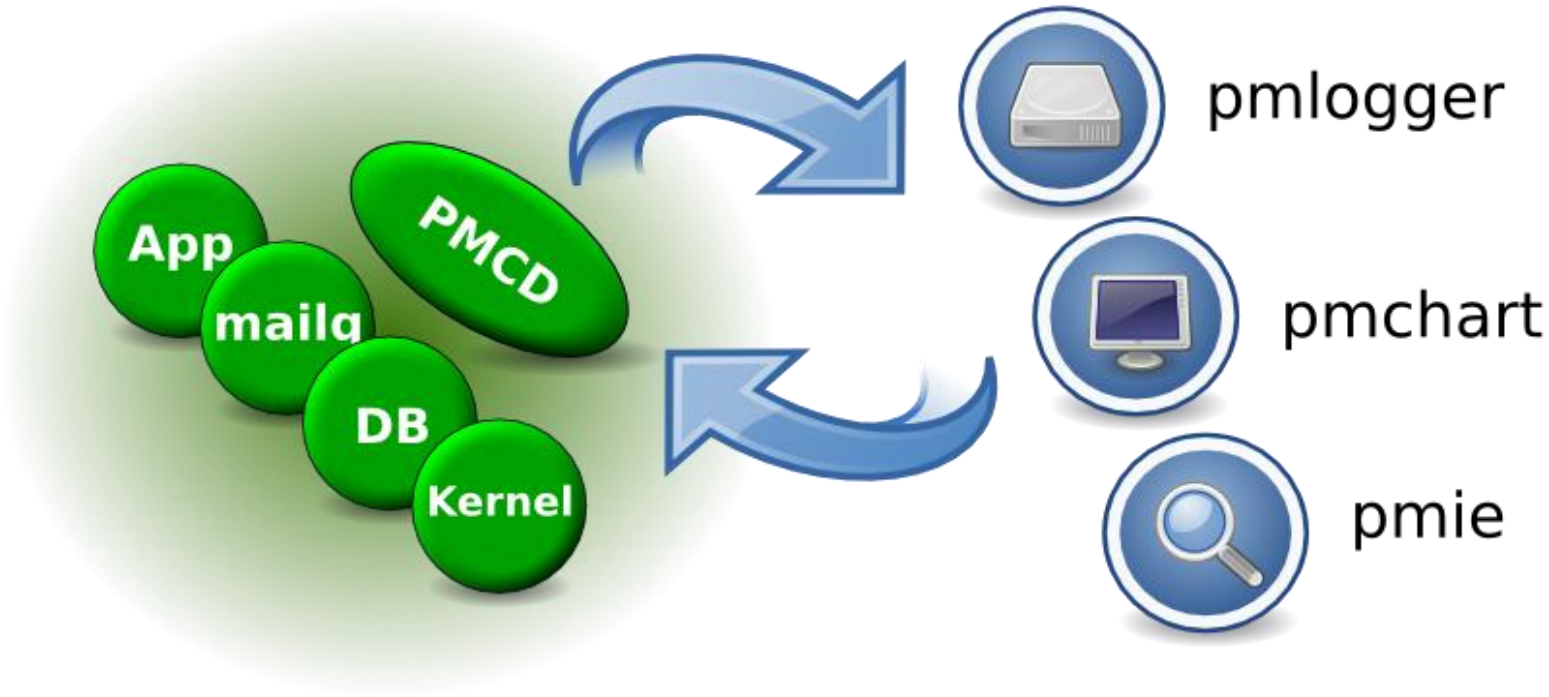
Agents and Daemons

At the core we have two basic components:

1. Performance Metric Domain Agents
   - Agents
2. Performance Metric Collection Daemon
   - PMCD

© 2016 Creative Arts & Technologies

PCP Architecture

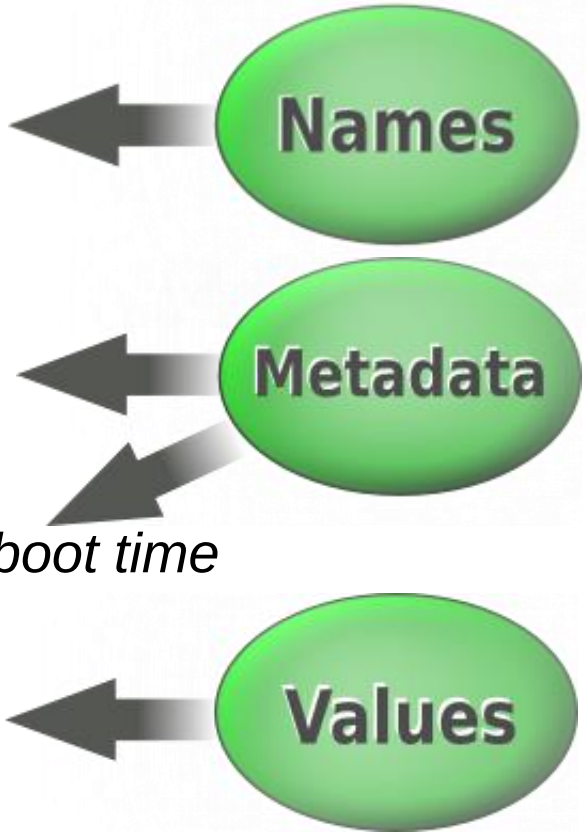


© 2016 Creative Arts & Technologies

PCP Metrics

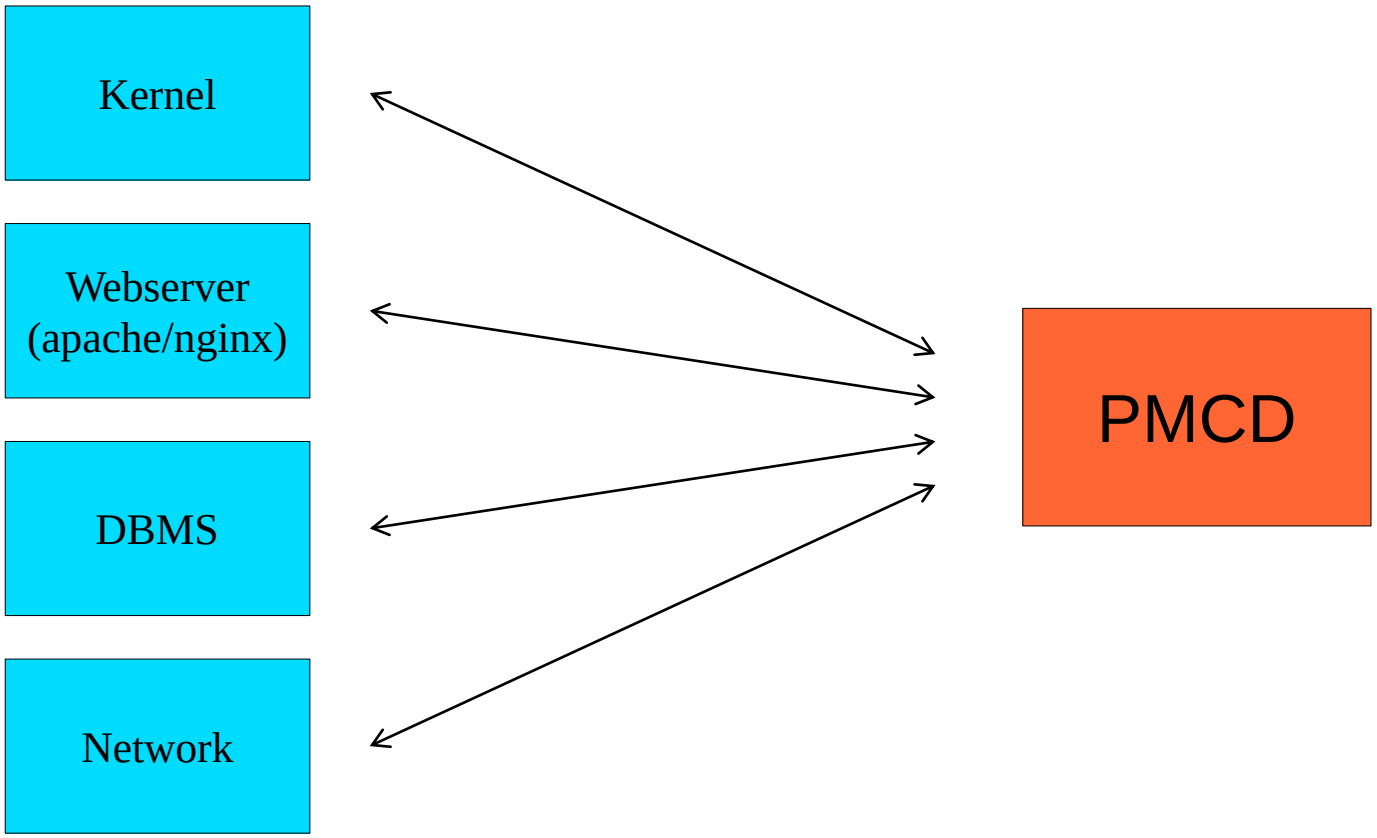
- **pminfo** --desc -tT --fetch ***disk.dev.read***

disk.dev.read [*per-disk read operations*]
Data Type: *32-bit unsigned int* InDom: *60.1*
Semantics: *counter* Units: *count*
Help: *Cumulative count of disk reads since boot time*
Values:
inst [0 or "***sda***"] value ***3382299***
inst [1 or "***sdb***"] value ***178421***



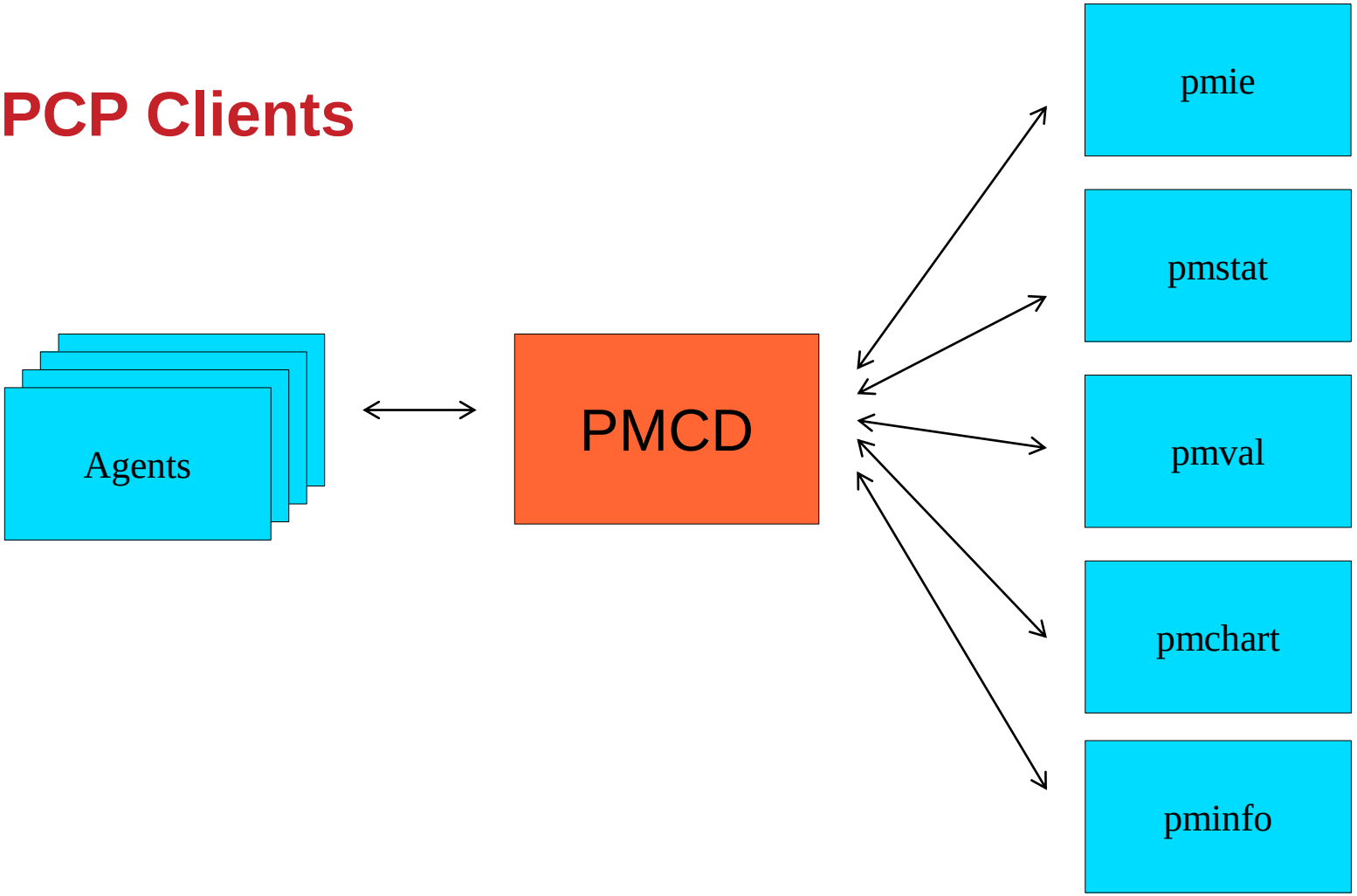
© 2016 Creative Arts & Technologies

PCP Agents



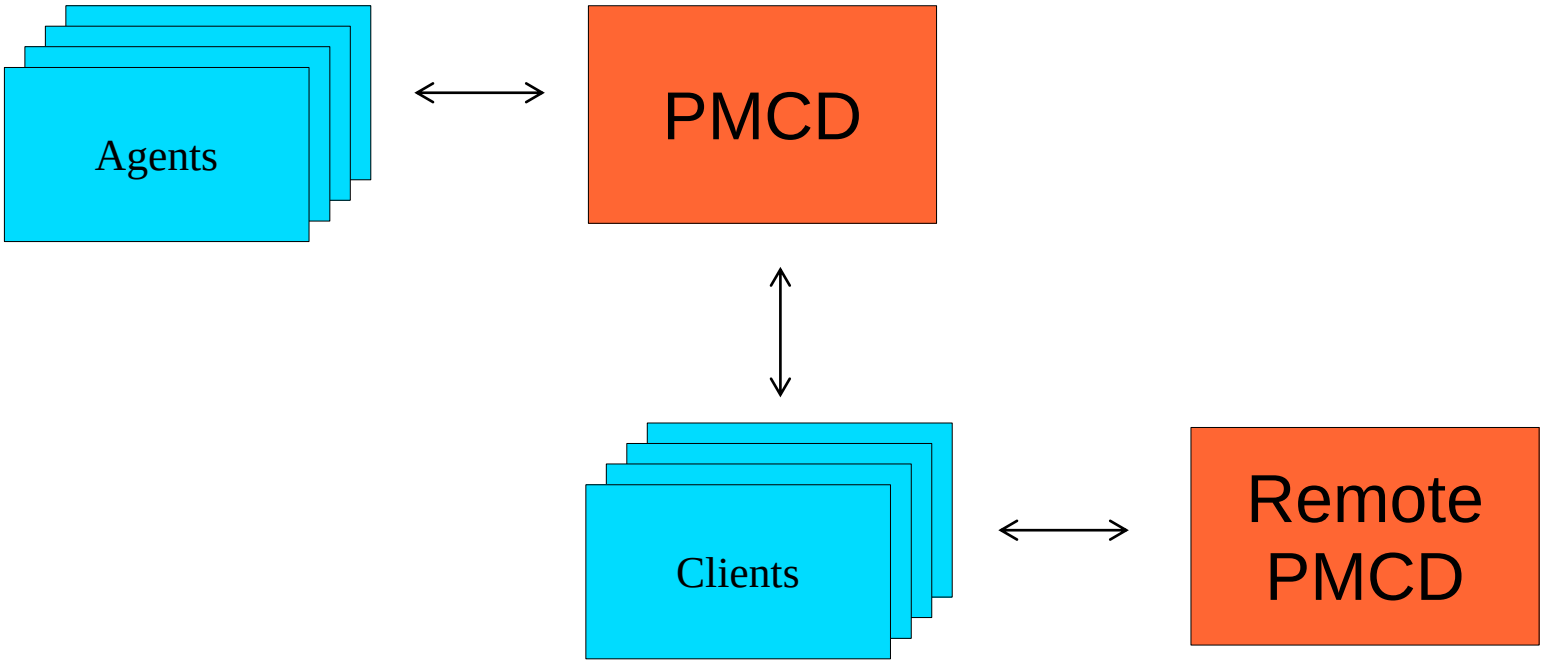
© 2016 Creative Arts & Technologies

PCP Clients



© 2016 Creative Arts & Technologies

PCP Remote Clients



© 2016 Creative Arts & Technologies

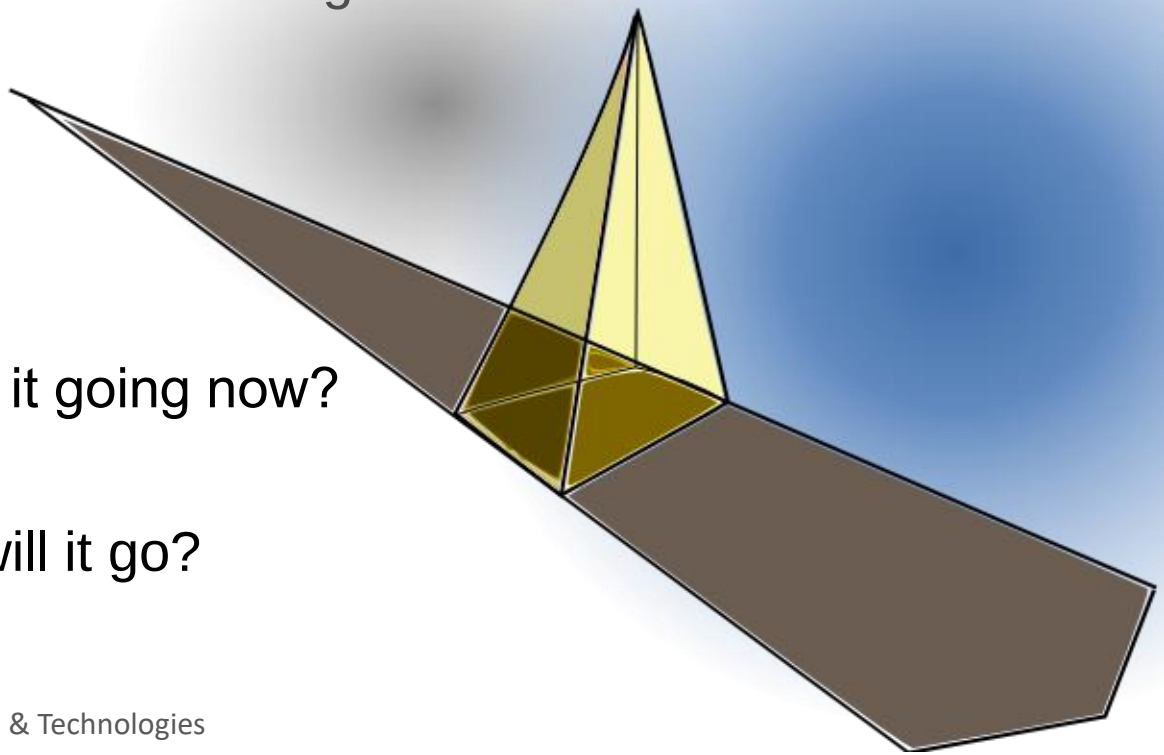
PCP Data Model

- Metrics come from one source (host / archive)
- Source can be queried at any interval by any monitor tool
- Hierarchical metric names
  e.g. disk.dev.read and aconex.response_time.avg
- Metrics are singular or set-valued (“instance domain”)
- Metadata associated with every metric
  - Data type (int32, uint64, double, ...)
  - Data semantics (units, scale, ...)
  - Instance domain

© 2016 Creative Arts & Technologies

Performance Timeline

- Where does the time go?

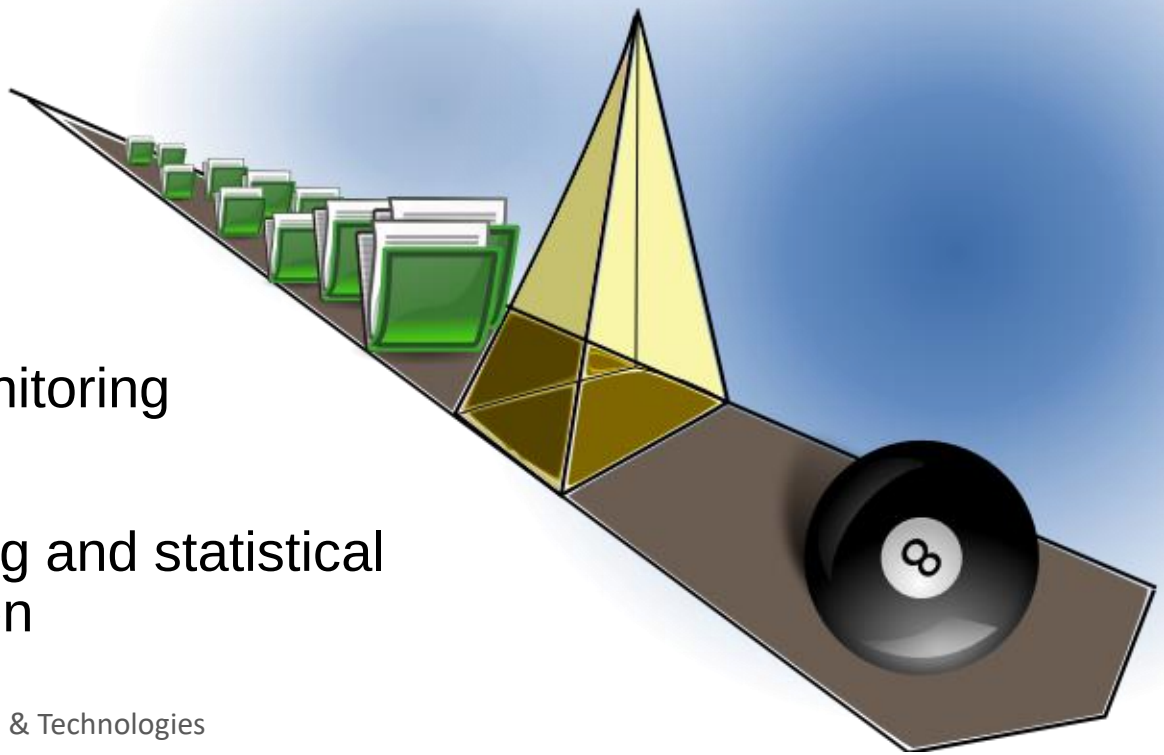


- Where's it going now?
- Where will it go?

© 2016 Creative Arts & Technologies

Performance Timeline – PCP Toolkit

- Archives



- Live Monitoring
- Modelling and statistical prediction

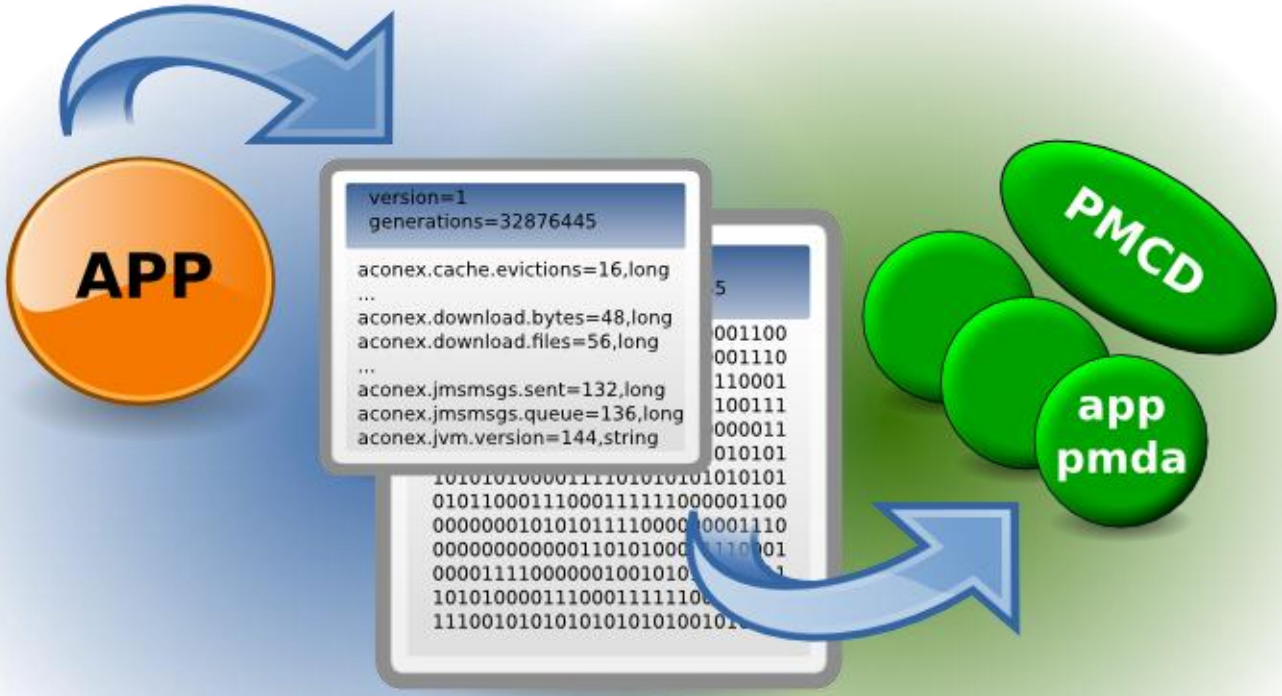
© 2016 Creative Arts & Technologies

Performance Timeline – PCP Toolkit

- Yesterday, last week, last month, ...
- All starts with pmlogger
  - Arbitrary metrics, intervals
  - One instance produces one PCP archive for one host
  - An archive consists of 3 files
    - Metadata, temporal index, data volume(s)
- pmlogger_daily, pmlogger_check
  - Ensure the data keeps flowing
- pmlogsummary, pmwtf, pmdumptext
- pmlogextract, pmlogreduce

© 2016 Creative Arts & Technologies

Custom Instrumentation (Applications)



© 2016 Creative Arts & Technologies

PCP – Parfait

Parfait has 4 main parts (for now)

- Monitoring
- DXM
- Timing
- Requests

© 2016 Creative Arts & Technologies

Parfait – Monitoring

- This is the ‘original’ PCP bridge metrics (heavily modified)
- Simple Java objects (MonitoredValues) which wrap a value (e.g. AtomicLong, String)
- MonitoredValues register themselves with a registry (container)
- When values changes, observers notice and output accordingly
  - PCP
  - JMX
  - Other (Custom/Extended)
- Very simple to use
- ‘Default registry’ (legacy concept)

© 2016 Creative Arts & Technologies

Parfait – DXM

- This is the PCP output side of aconex-pcp-bridge
- Rewritten to use the new non-custom MMV PMDA
- Advantages:
  - Flexible, standardized, less maintenance work
- Disadvantages
  - Have to assign ID to each metric
- Map metrics names to ‘pseudo-PCP’ names, e.g.:
  - aconex.controllers.time.blah → aconex.controllers[mel/blah].time
- Placement of brackets is significant (determines PCP domains)

© 2016 Creative Arts & Technologies

Parfait – Timing

- Logs the resources consumed by a request (an individual user action)
- Relies on a single request being thread-bound (and threads being used exclusively)
- Basically needs a Map<Thread, Value>
- Take the value for a Thread at the start, and at the end
- Delta is the ‘cost’ of that request

© 2016 Creative Arts & Technologies

Parfait – Timing Example

```
[2010-09-22 15:02:13,466 INFO ][ait.timing.Log4jSink][http-8080-Processor3
gedq93kl][192.168.7.132][20][] Top taskssummaryfeatures:tasks
       taskssummaryfeatures:tasks        Elapsed time: own 380.146316 ms, total
380.14688 ms  Total CPU: own 150.0 ms, total 150.0 ms   User CPU: own 140.0 ms,
total 140.0 ms       System CPU: own 10.0 ms, total 10.0 ms    Blocked count: own
40, total 40  Blocked time: own 22 ms, total 22 ms      Wait count: own 2, total
2      Wait time: own 8 ms, total 8 ms    Database execution time: own 57 ms,
total 57 ms   Database execution count: own 11, total 11       Database logical
read count: own 0, total 0  Database physical read count: own 0, total 0
       Database CPU time: own 0 ms, total 0 ms   Database received bytes: own
26188 By, total 26188 By    Database sent bytes: own 24868 By, total 24868 By
       Error Pages: own 0, total 0 Bobo execution time: own 40.742124 ms, total
40.742124 ms  Bobo execution count: own 2, total 2      Bytes transferred via
bobo search: own 0 By, total 0 By  Super search entity count: own 0, total 0
       Super search count: own 0, total 0 Bytes transferred via super search: own
0 By, total 0 By     Elapsed time during super search: own 0 ms, total 0 ms
```

© 2016 Creative Arts & Technologies

Parfait – Requests

- As well as snapshotting requests after completion, for many metrics we can see meaningful ‘in-progress’ values
- Simple JMX bean which ‘walks’ in-progress requests
- Tie in with ThreadContext (MDC abstraction)
- Include UserID
- ThreadID

© 2016 Creative Arts & Technologies

PCP – Speed

Golang implementation of the PCP instrumentation API

There are 3 main components in the library

- Client
- Registry
- Metric

© 2016 Creative Arts & Technologies

PCP – Speed Metric

- SingletonMetric
  - This type defines a metric with no instance domain and only one value. It requires type, semantics and unit for construction, and optionally takes a couple of description strings. A simple construction

```
metric, err := speed.NewPCPSingletonMetric(
    42,                                                    // initial value
    "simple.counter",                                      // name
    speed.Int32Type,                                       // type
    speed.CounterSemantics,                               // semantics
    speed.OneUnit,                                        // unit
    "A Simple Metric",                                    // short description
    "This is a simple counter metric to demonstrate the speed API", // long descr
)
```

© 2016 Creative Arts & Technologies

APACHE:
BIG_DATA
EUROPE

PCP for Containers

Image © HitchhikersHandbook.com

PCP for Containers – Cgroup Accounting

- [subsys].stat files below /sys/fs/cgroup
- individual cgroup or summed over children
- **blkio**
- IOPs/bytes, service/wait time – aggregate/per-dev
- Split up by read/write, sync/async
- **cpuacct**
- Processor use per-cgroup - aggregate/per-CPU
- **memory**
- mapped anon pages, page cache, writeback, swap, active/inactive LRU state

© 2016 Creative Arts & Technologies

PCP for Containers – Namespaces

- Example: **cat /proc/net/dev**
- Contents differ inside vs outside a container

- Processes (e.g. **cat**) in containers run in different network, ipc, process, uts, mount namespaces

- Namespaces are inherited across fork/clone
- Processes within a container share common view

© 2016 Creative Arts & Technologies

PCP Container Analysis – Goals

- Allow targeting of individual containers
- e.g. ***/proc/net/dev***
- pminfo --fetch network
- vs
- pminfo –fetch –container=crank network
- Zero installation inside containers required
- Simplify your life (dev_t auto-mapping)
- Data reduction (proc.*, cgroup.*)

© 2016 Creative Arts & Technologies

PCP Container Analysis – Mechanisms

- pminfo -f –host=**acme.com** –container=**crank** network
- Wire protocol extension
- Inform interested PCP collector agents
- Resolving container names, mapping names to cgroups, PIDs, etc.
- setns(2)
- Runs on the board, plenty of work remains
- New monitor tools with container awareness

© 2016 Creative Arts & Technologies

What is Metrics?

- Code instrumentation
- Meters
- Gauges
- Counters
- Histograms
- Web app instrumentation
- Web app health check

© 2016 Creative Arts & Technologies

Metrics Reporters

- Reporters
  - Console
  - CSV
  - Slf4j
  - JMX
- Advanced reporters
  - Graphite
  - Ganglia

© 2016 Creative Arts & Technologies

Metrics 3rd Party Libraries

- AspectJ
- InfluxDB
- StatsD
- Cassandra
- Spring

© 2016 Creative Arts & Technologies

Metrics Basics

- MetricsRegistry
  - A collection of all the metrics for your application
  - Usually one instance per JVM
  - Use more in multi WAR deployment
- Names
  - Each metric has a unique name
  - Registry has helper methods for creating names

```
MetricRegistry.name(Queue.class, "items", "total")
//com.example.queue.items.total
MetricRegistry.name(Queue.class, "size", "byte")
//com.example.queue.size.byte
```

© 2016 Creative Arts & Technologies

Metrics Elements

- Gauges
  - The simplest metric type: it just returns a *value*

```
final Map<String, String> keys = new HashMap<>();
  registry.register(MetricRegistry.name("gauge", "keys"),
new          Gauge<Integer>() {

@Override
public Integer getValue() {
  return keys.keySet().size();
}
});
```

© 2016 Creative Arts & Technologies

Metrics Elements (2)

- Counters
  - Incrementing and decrementing 64.bit integer

```
final Counter counter= registry.counter(MetricRegistry.name("counter",
        "inserted"));
counter.inc();
```

© 2016 Creative Arts & Technologies

Metrics Elements (3)

- Histograms
  - Measures the distribution of values in a stream of data

```
final Histogram resultCounts = registry.histogram(name(ProductDAO.class,
"result-counts");
resultCounts.update(results.size());
```

- Meters
  - Measures the rate at which a set of events occur

```
final Meter meter = registry.meter(MetricRegistry.name("meter", "inserted"));
meter.mark();
```

© 2016 Creative Arts & Technologies

Metrics Elements (4)

- Timers
  - A histogram of the duration of a type of event and a meter of the rate of its occurrence

```
Timer timer = registry.timer(MetricRegistry.name("timer", "inserted"));
Context context = timer.time();
//timed ops
context.stop();
```

© 2016 Creative Arts & Technologies

Metrics – Graphite Reporter

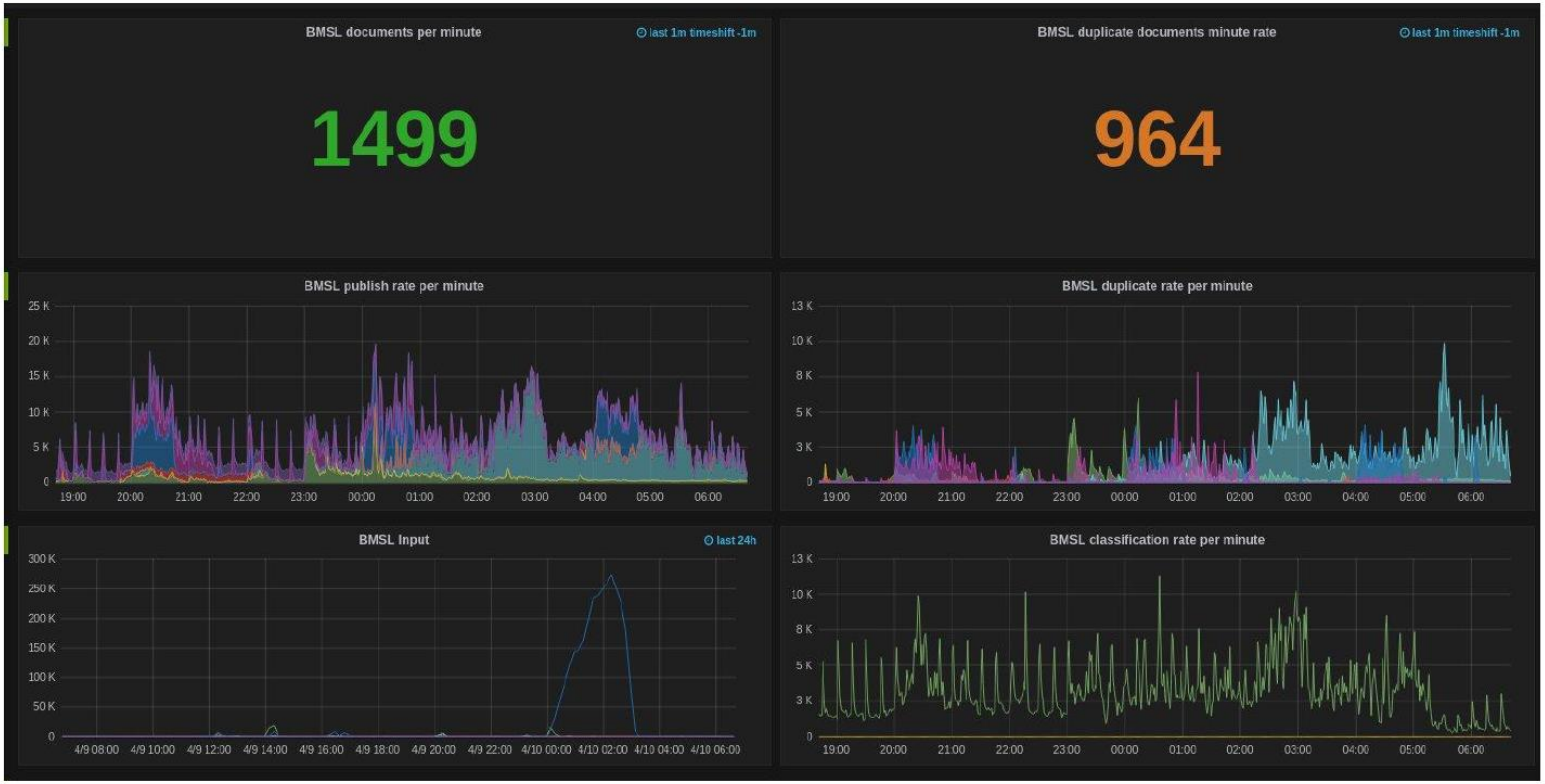
```
final Graphite graphite = new Graphite(new
InetSocketAddress("graphite.example.com", 2003));
final GraphiteReporter reporter = GraphiteReporter.forRegistry(registry)
.prefixedWith("web1.example.com")
.convertRatesTo(TimeUnit.SECONDS)
.convertDurationsTo(TimeUnit.MILLISECONDS)
.filter(MetricFilter.ALL)
.build(graphite);
reporter.start(1, TimeUnit.MINUTES);
```

Metrics can be prefixed

Useful to divide environment metrics: prod, test

© 2016 Creative Arts & Technologies

Metrics – Grafana Application Overview



© 2016 Creative Arts & Technologies

Apache Sirona – Inspired by JaMon

© 2016 Creative Arts & Technologies

Sirona Basics

- Repository
  - The repository is a singleton for the JVM. It is the entry point to get access to counters and gauges.

```
public interface Repository extends Iterable<Counter> {
    Counter getCounter(Counter.Key key);
    void clear();
    StopWatch start(Counter counter);

    Map<Long, Double> getGaugeValues(long start, long end, Role
role);
    void stopGauge(Role role);
}
```

© 2016 Creative Arts & Technologies

Sirona Elements

- Counter
  - A counter is a statistic and concurrency holder. It aggregates the information provided computing the average, min, max, sum of logs, ….

```
public interface Counter {
    Key getKey();
    void reset();
    void add(double delta);
    AtomicInteger currentConcurrency();
    int getMaxConcurrency();
    double getMax();
    double getMin();
    long getHits();
    double getSum();
    double getStandardDeviation();
    double getVariance();
    double getMean();
    double getSecondMoment();
```

© 2016 Creative Arts & Technologies

Sirona Elements (2)

- Gauge
  - A gauge is a way to get a measure. It is intended to get a history of a metric.

```
public interface Gauge {
    Role role();
    double value();
}
```

- StopWatch
  - A StopWatch is just a handler for a measure with a counter.

```
public interface StopWatch {
    long getElapsedTime();
    StopWatch stop();
}
```

© 2016 Creative Arts & Technologies

What is StatsD?

A network daemon that runs on the **Node.js** platform and listens for statistics, like counters and timers, sent over UDP or TCP and sends aggregates to one or more pluggable backend services (e.g., Graphite).

StatsD was inspired (heavily) by the project (of the same name) at Flickr.

© 2016 Creative Arts & Technologies

APACHE:
BIG_DATA
EUROPE

Demos

Image © HitchhikersHandbook.com

Links

- Performance Co-Pilot
  http://www.pcp.io
- Dropwizard Metrics
  http://metrics.dropwizard.io
- Apache Sirona
  http://sirona.apache.org/
- StatsD
  https://github.com/etsy/statsd/wiki
- Java Community Process
  https://jcp.org/

© 2016 Creative Arts & Technologies

APACHE:
BIG_DATA
EUROPE

Thank You

Image © HitchhikersHandbook.com