GOBIERNO DE ESPAÑA

MINISTERIO DE CIENCIA, INNOVACIÓN Y UNIVERSIDADES

Ciemat
Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas

EXCELENCIA MARÍA DE MAEZTU

cfp CIEMAT física de partículas

CERN ALICE ATLAS EXPERIMENT CMS LHCb

Design and Implementation of a Monitoring Framework

ISOTDAQ
International School of Trigger and Data Acquisition

Serguei Kolos,

University of California, Irvine

Introduction

- For a complex HW&SW system there is no off the shelf monitoring framework solution
  - Such a Framework can be constructed from a number of tools, which are available on the market
- The aim of the talk is to give conceptual overview of a Monitoring Framework and explain how existing tools can be combined to build it:
  - Technical details are intentionally omitted
  - The SW tools are just named without any detailed explanation
- The presented solution is general enough to be used for any system:
  - DAQ specifics are explicitly highlighted

How Higgs boson discovery would have looked like in the ideal world

In real life problems always happen

- If a problem occurs we need:
  - Someone to spot this problem
  - A witness to tell us what happened
  - An investigator to analyze this information and uncover the root of the issue
- These duties are done by a Monitoring Framework

"I swear to tell the truth, the whole truth, and nothing but the truth, from my perspective."

The Simplest Monitoring Example Ever

The monitoring API function that is used for reporting events

An event

```
print("Hello, World")
```

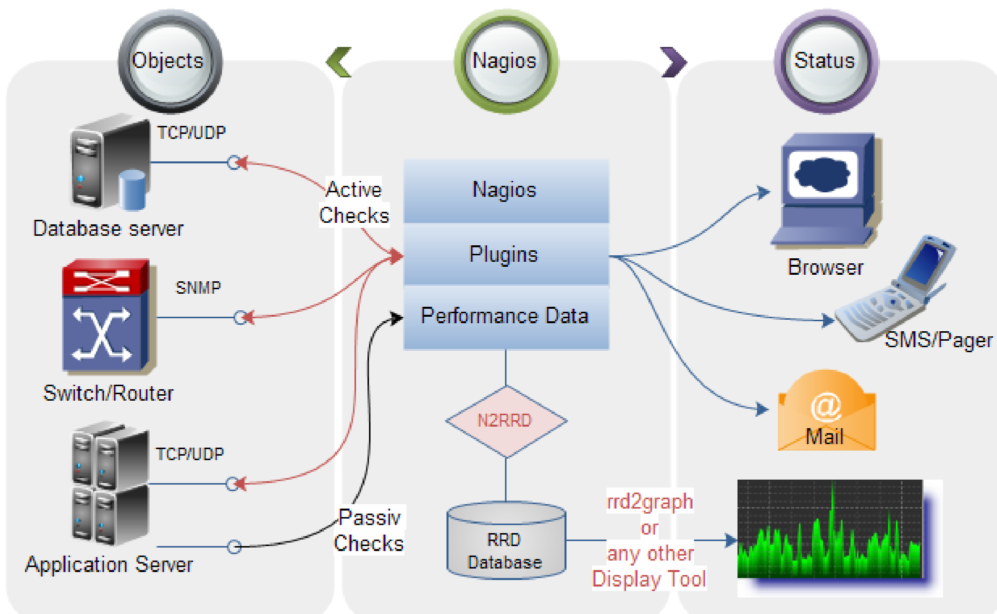
When you execute this program "Hello, World" appears on the screen:

- Informs us that the program fulfills its objective

Two Monitoring Framework Architectures

- Example in the previous slide uses one of the two fundamental approaches to design a monitoring system:
  - It uses ***Push***, ***White Box*** or ***Object Centered*** approach
  - As opposite to the ***Pull***, ***Black Box*** or ***Framework Centered*** one

- Nagios is a classical example of the ***Black Box*** approach:
  - The states of monitored objects can be tested by external checks
  - They remain black boxes otherwise with respect to the Monitoring Framework



Black Box approach is not suitable for DAQ system

Quoting Andrea's talk of the last Monday

- **Data AcQuisition** is an heterogeneous field
  - Boundaries not well defined
  - An **alchemy** of physics, electronics, networking, computer science, …
  - Hacking and experience
  - …, money and manpower matter as well

- DAQ system has many custom HW and SW components:
  - Difficult to extract monitoring information by means of external probing
- DAQ components operate at high frequencies:
  - Polling for monitoring information is inefficient
- Requires immediate reaction to issues:
  - Operation time is expensive

The White Box approach

- The Framework that will be described in this talk sticks to the ***White Box*** architecture
- The objects of the system been monitored actively advertise their states to the Monitoring Framework
  - SW applications do self-monitoring
  - HW objects states are exposed via their SW counterparts (drivers, controllers, etc.)
- This implies that the Monitoring is built in to every application by design:
  - This looks like a limitation but in fact this is a gift because it enforces the use of the Monitoring Framework from the very beginning

Theoretical Digression

- Using Monitoring Framework from the start gives a number of advantages:
  - Saves time for the DAQ system development by giving powerful means for testing and debugging
  - The Framework evolves together with the DAQ system taking into account live feedback:
    - Will be ready from the day one
- Monitoring is often considered as an ad hoc facility:
  - This is simply wrong!
  - You won't be able to control your DAQ system without a good Monitoring Framework

"It looks like you have everything under control."

“White Box” Monitoring: Information Types

- **Events** – anything of importance that happens in the system
- **Logs** – auxiliary information that is used for debugging and testing the system
- **Metrics** – numbers showing how the system perform
- Information types are different with respect to how they are produced and being handled

Handling Events* and Logs

* ***Event*** in the Monitoring domain means an incident.
Physics Event will be used instead when talking about data from a physics detector.

Get back to our example

```
print("Hello, World")
```

- A print-like function can be used for reporting events
- But it has a number of flaws:
  - It provides both the API and the implementation – no customization is possible
  - It does not support event levels
  - It does not add time stamp and other useful attributes to an event
- Do better solutions exist?

Logging API to the rescue

```
import logging

logging.basicConfig(format='%(asctime)s %(levelname)s [%(filename)s:%(lineno)s - %(funcName)s()] %(message)s', level=logging.INFO)

logging.info("Hello, World")
```

Use the standard well-designed API

The output format can be easily changed

```
2017-02-02 16:21:13,583 INFO [hello.py:4 - <module>()] Hello, World
```

Extra properties are added automatically

Events destination can be changed without touching the application code

Logging APIs

Python

```
import logging

critical(msg, *args, **kwargs)
debug(msg, *args, **kwargs)
error(msg, *args, **kwargs)
info(msg, *args, **kwargs)
warning(msg, *args, **kwargs)
```

Java

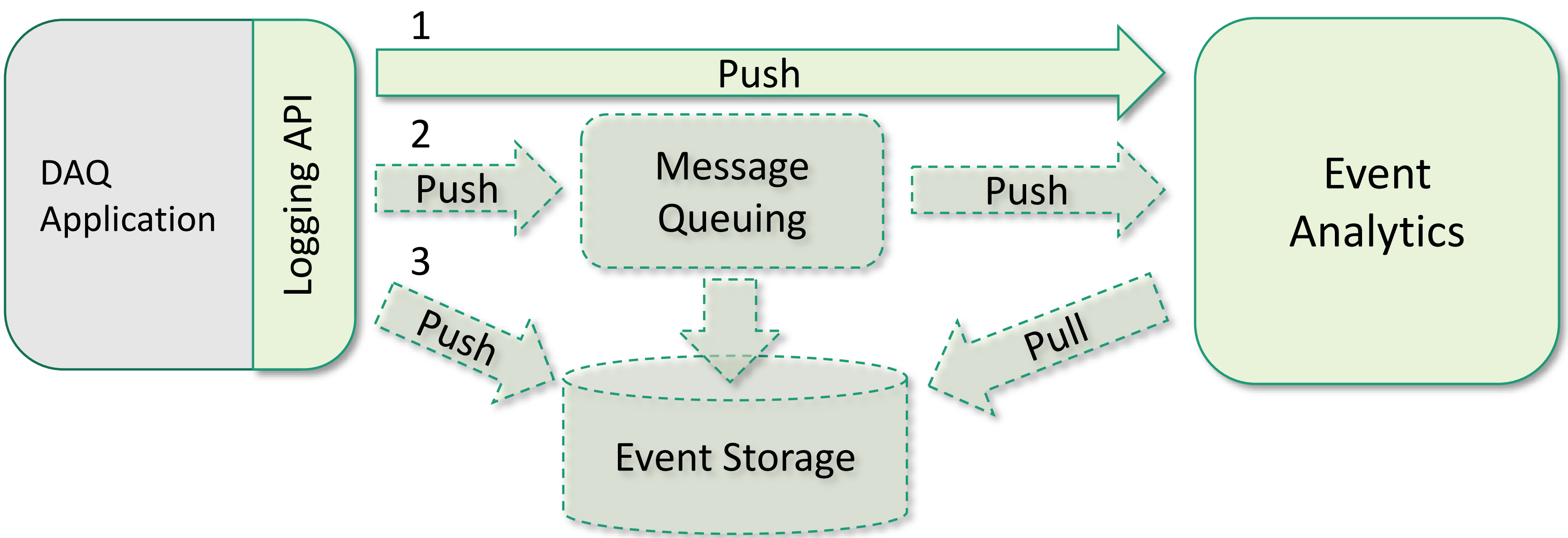
```
import java.util.logging.Logger

severe(String msg);
fine(String msg);
error(String msg);
info(String msg);
warning(String msg);
```

- Events destination is fully customizable by configuring a set of so called Appenders

Java Logger: Existing Appenders

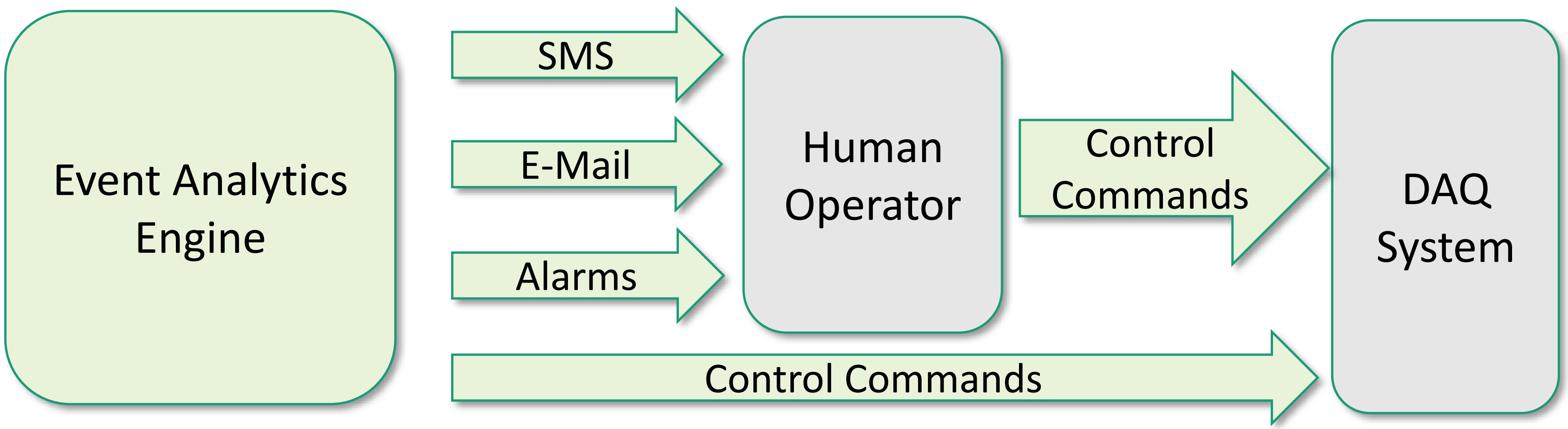
- **CassandraAppender** - writes its output to an Apache Cassandra database
- **FileAppender** – writes events to an arbitrary file.
- **FlumeAppender** - Apache Flume is a distributed, reliable and highly available system for efficiently collecting, aggregating, and moving large amounts of log data
- **JDBCAppender** - writes log events to a relational database table using standard JDBC
- **NoSQLAppender** - writes log events to a NoSQL database
- **SMTPAppender -** sends an e-mail when a specific logging event occurs, typically on errors or fatal errors
- **ZeroMQAppender -** uses the JeroMQ library to send log events to one or more ZeroMQ endpoints

The Flow of Monitoring Events



- All components shown in this diagram can be found on the market:
  - No custom programming is needed
- Message Queuing can be part of the Event Analytics engine
- Event Storage is optional but very convenient:
  - Provides Events archive that can be used for post mortem analysis
  - Distributed storage can be used in place of an Message Queuing

The Final Destination

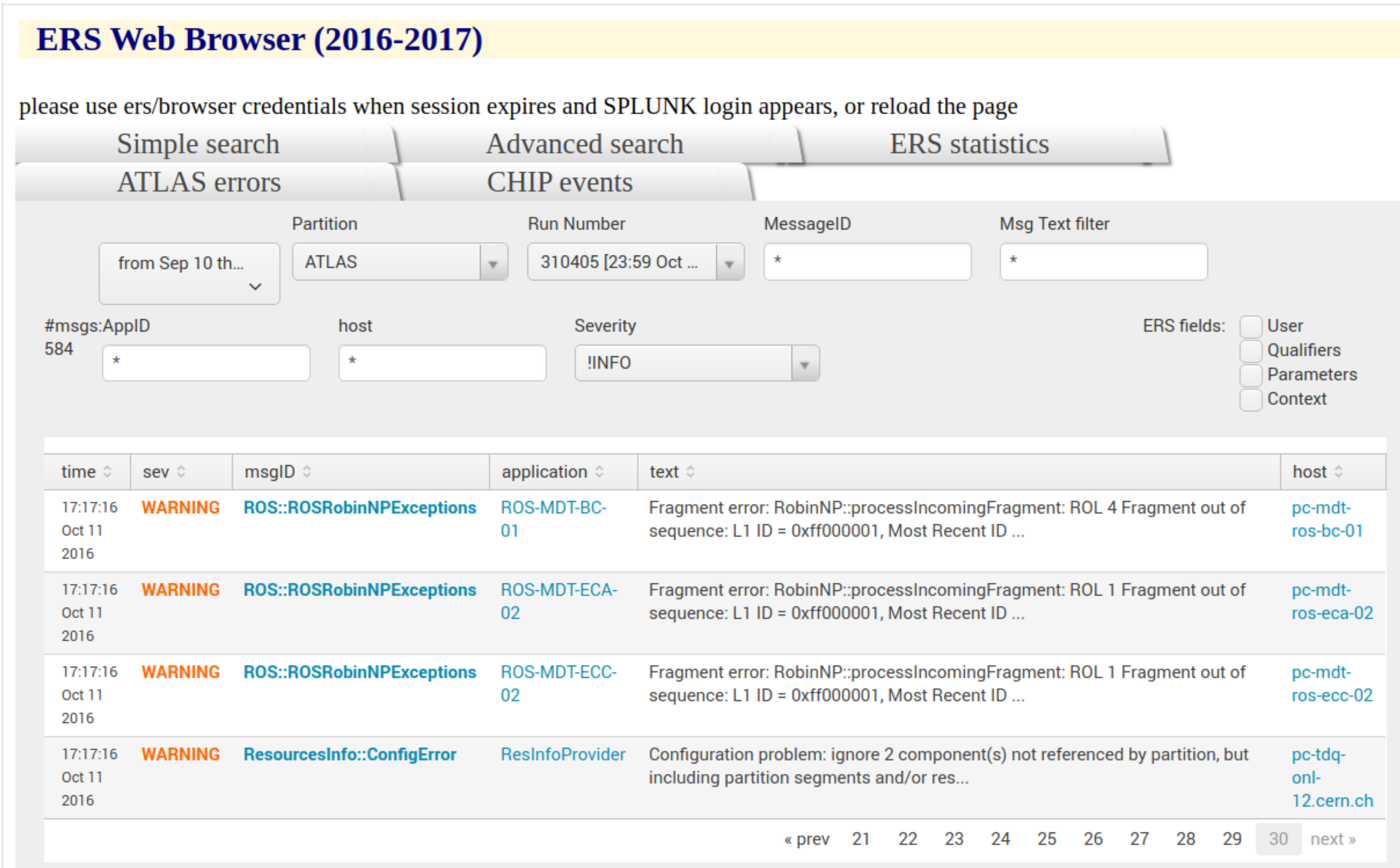


- Any Event Analytics engine allows to specify how to react to a combination of events:
  - It’s just a matter of a configuration

Events Analytics Tools: The Market Overview

- There are a few Event Analytics tools available on the market
- They can be split into two main groups:
  - Standalone Tools:
    - **Splunk** – standalone log monitor with real-time alerts and events visualization:
      - Needs an external Message Queuing software
      - Used in ATLAS for events display in Web Browsers
    - **Esper** – complex event analysys tool implemented in Java.
  - Distributed tools:
    - **Apache Kafka** – a distributed real-time publish-subscribe messaging system
    - **Apache Heron** - a distributed stream processing engine developed at Twitter

ATLAS Web Based Events browser implemented with Splunk



Event Properties

- An Event shall contain:
  - Importance Level (or Severity)
  - A human readable (and understandable) explanation of the corresponding incident:
    - A shifter calling you in the middle of a night can just read it
  - A source ID that identifies the origin of the incident (Application or a HW module ID)
  - The time stamp of the incident

Event Importance Levels

- **CRITICAL**
  - A fatal failure has occurred. This indicates that the component can not do its work any more without external intervention
- **ERROR**
  - A recoverable error has happened
- **WARNING**
  - Nothing is bad so far but the system is close to a certain limit
  - Do not neglect warnings as they tend to become errors
- **INFO**
  - An expected incident has occurred
- **DEBUG**
  - Detailed information used for testing and debugging
  - This level corresponds to what we call **Logs**

Event Source ID Schema

- It's important to know who reported a specific event
- A common naming schema has to be developed in advance and used by the Logging API implementation:
  - Allows easy grouping and filtering of events
  - Is very helpful for scaling up the Monitoring Framework implementation
- It may look like:
  - **Location.Application.Environment**

Time Stamps

- Time Stamp is a vital property of an Event and of any monitoring information in general:
  - Used for correlating events from different sources
  - Used for correlating monitoring events with real-life incidents
- The time stamp guidelines:
  - Use NTP service on all computers
  - Use the best possible precision (nanoseconds) when generating time stamps
  - Use UTC time when reporting
  - Conversion to the human readable local time shall be done by a event displaying applications with respect to its location

Finally, what to do with Logs?

- Logs are a sub-type of Events:
  - Provides additional information for debugging
  - Normally is used by human experts only:
    - Not suitable for automated analysis by an event analytics engine
  - Amount of Logs usually exceeds the total amount of all other types of Events
- The best destination for logs would probably be a file system:
  - Foresee enough space on the local disks
  - Old logs can be archived or removed after some configurable time

System Metrics

- Metrics are measures of properties in the software or hardware system components:
  - Have to be watched over time
  - Crucial information for the monitoring
- Metrics should be kept as close as possible to the properties being measured:
  - In the White Box architecture Metrics are produced by the SW Applications being monitored

Metric Types

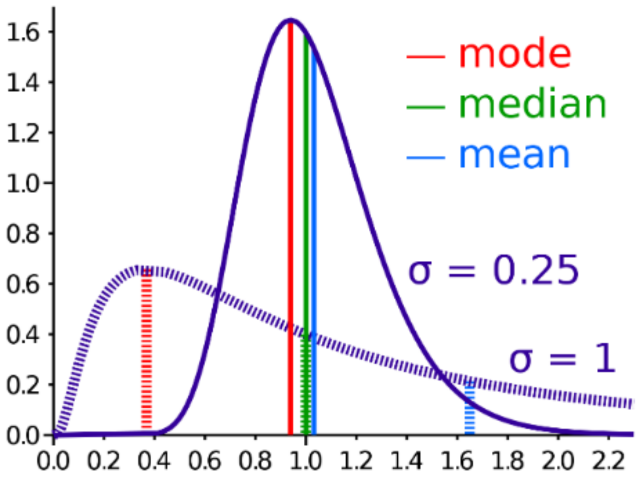
- Gauge:
  - A snapshot of a specific measurements
    - CPU, memory, disk usage; voltages, currents, pressure, etc.
- Timer:
  - Time interval that is taken by a certain operation
  - A kind of Gauge measured by a chronometer
- Counter:
  - Monotonically increasing integer number
    - Number of triggers, number of bytes sent/received, etc.

Metrics Naming

- Each Metric shall have a unique IDs reflecting its origin and the nature:
  - Event Naming Schema + Metric Name
  - **Location.Application.Environment.Metric**
  - *ATLAS.Dataflow.RecordedEvents.Counter*
- Obeying to a common naming schema can greatly simplify metrics handling:
  - Easy selection and filtering using regular expressions
  - Common operations like for example aggregation can be split into multiple stages for scalability

Derivative Metrics

- Basic metrics are not very useful for the Monitoring
- Mathematical transformations are applied to the original values to produce more useful numbers:
  - Time based:
    - Average, Min/Max and Median for Gauges and Timers
    - Rates for Counters
  - Frequency distribution:
    - Histograms for Gauges and Timers



Derivative Metrics Can be Produced By…

- Metric Providers:
  - Define and publish extra Metrics
  - Example:
    - Original Metric: ATLAS.Dataflow.RecordedEvents.Counter
    - Derivative Metric: ATLAS.DataFlow.RecordedEvents.Rate
- Metrics Visualization Engine:
  - Reads time series for the Counter Metric
  - Calculates and display the Rate Metric on the fly
  - Pro:
    - Reduces the amount of data being stored and passed around
    - Reduces Metrics production overhead
  - Cons:
    - Places more load on the visualization software

An API for Metric Providers?

- Unlike the Logging API for Events&Logs there is no commonly accepted API for Metrics:
  - SW tools for metrics collection and analysis use their proprietary APIs
- This may not be a problem for a small short-living project:
  - Directly using a specific SW API is a viable option
  - Be careful to choose a SW with the live-time going beyond your project time-scale
- HEP experiments have a life-time of ~30 years:
  - It's difficult to find a SW system that is likely to survive that long

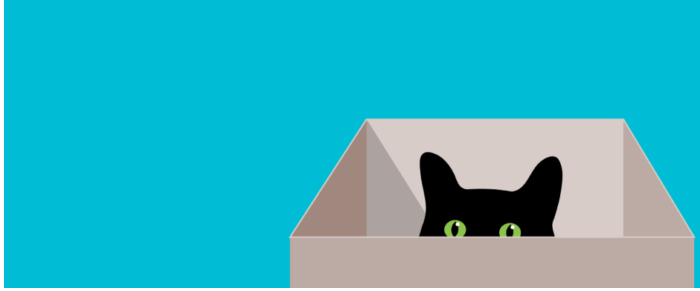
A Simple API for Metric Providers

```
package Monitoring;
interface Gauge {
	void setValue(double v);
}
interface Counter {
	void increment();
}
interface Metric {
	Gauge createGauge(String name);
	Counter createCounter(String name);
}
```

Metrics Providers API implementation can give some extra benefits

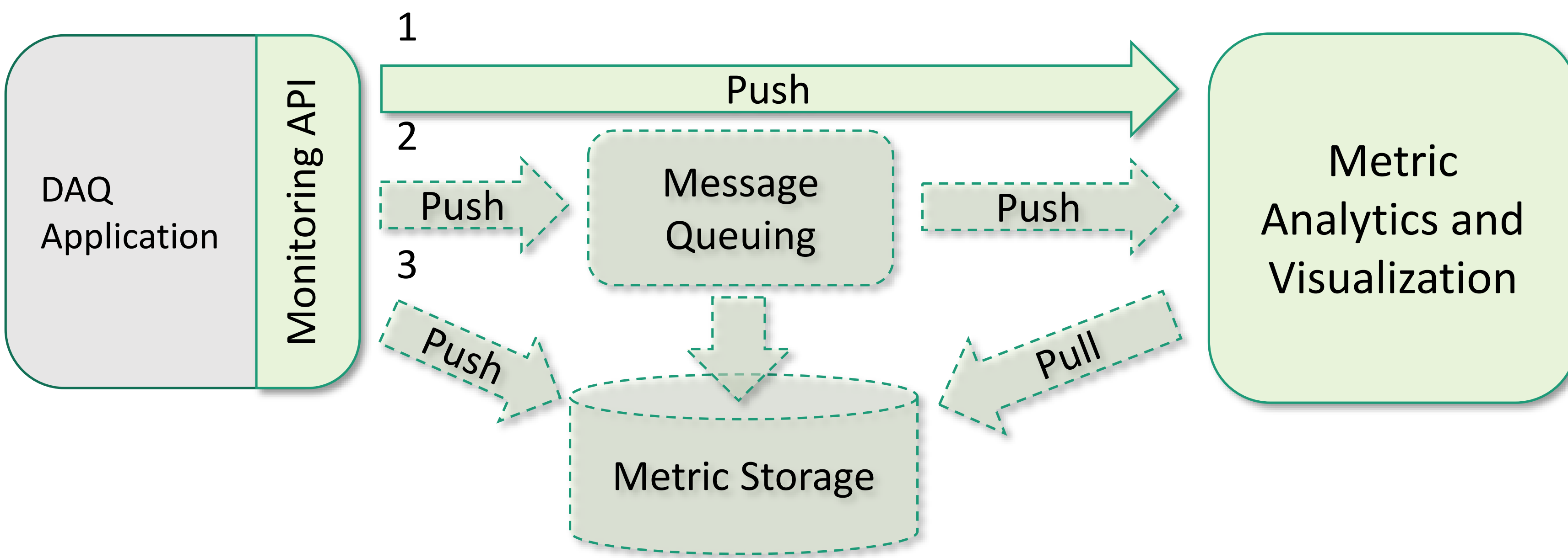
- Isolates dependency on the underlying SW tools:
  - Switching from one SW to the other is transparent for the system applications
- Enforces the Naming Scheme:
  - The Application just provides a local Metric name, the prefix is added automatically
- Ensures uniqueness of the Metrics IDs
- Calculates derivative Metrics automatically with respect to a given configuration:
  - Applications don’t need to warry about that
- Keeps “Observation Effect” under control

“Observation Effect”



- An observation has an overhead:
  - It consumes some resources (CPU, memory, network bandwidth)
  - It may have an influence on the monitoring data providing application:
    - Depends on the monitoring framework architecture
- To minimize this overhead any communication with external tools should be done asynchronously

The Monitoring Framework Architecture



- The underlying architecture is transparent for DAQ applications:
  - Central Storage implementation can be changed to a Distributed Access one and vice versa
  - Using both of them at the same time may also be a feasible option

The Market Overview

What tools can be used to implement a Monitoring Framework

Metrics Checking Tools

- The same tools as for Events processing may be used:
  - **Riemann** - aggregates events from servers and applications with a powerful stream processing language
  - **Esper** - complex events processing streaming analytics
    - Used in ATLAS for Shifter Assistant and Expert Systems implementation
- This allows to define advanced rules based on both Events and Metric
- Reduces time for development and simplify maintenance of the Monitoring Framework

Storage Implementations

- Traditional relational databases may work well for a small-scale project
- For a large DAQ system one should consider NoSQL distributed alternatives:
  - **Whisper** – a lightweight, flat-file database format for storing time-series data
  - **InfluxDB** – a time-series database that's written in Go
  - **Cassandra** – scalable, high availability storage platform
  - **MongoDB** - a general purpose, document-based, distributed database

Metric Visualization Tools

- There are many Open Source tools:
  - **Grafana** - the open observability platform
  - **D3** - a Javascript library for data visualization
  - **Rickshaw** – a Javascript library for data visualization
  - **Facette** – a multi-data source dashboard written in Go
  - There are a few others as well…
- In general It is a good idea to choose a tool that supports RESTful interface for data access

RESTful Protocol

- REST – **Re**presentational **S**tate **T**ransfer
- Client-server HTTP-based stateless communication protocol
- Supported by most of the modern information storage as well as Web-based Visualisation systems:
  - Supports seamless interoperations
- Makes it easy to switch from one Storage or Visualisation platform to another

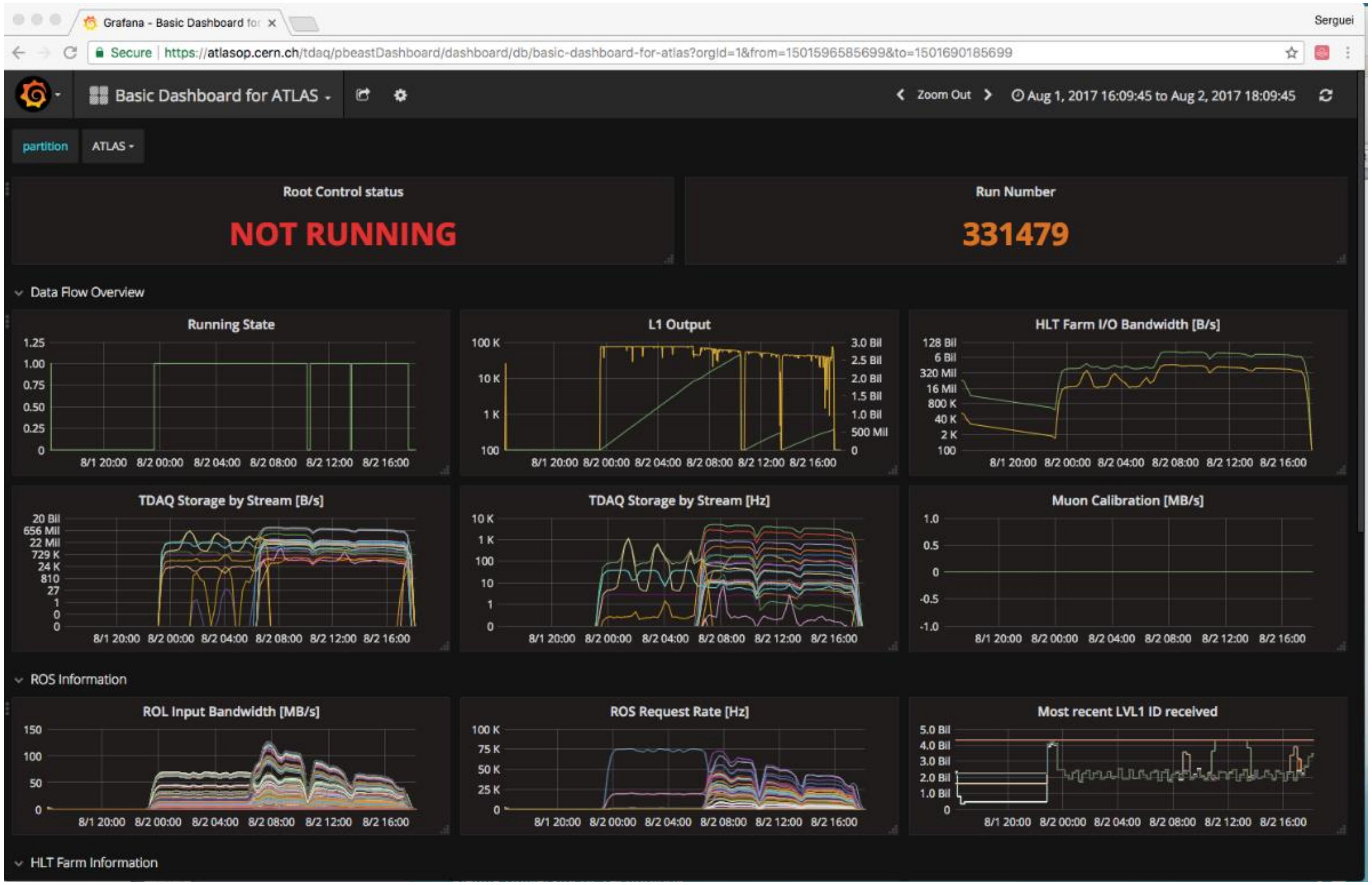
REST Protocol Example

- Request:
```
https://atlasop.cern.ch/monitoring/
    ? id=ATLAS.Dataflow.RecordedEvents.Rate
    & from=now-30d
    & to=now
```
- Response:

  Json Time Series, e.g.:


```
[
    {t:1579104640,v:12345},
    {t:1579104645,v:12354},
    {t:1579104650,v:12354},
    {t:1579104655,v:12352}
]
```

Web-based ATLAS Online Monitoring Customizable Dashboard implemented using Grafana



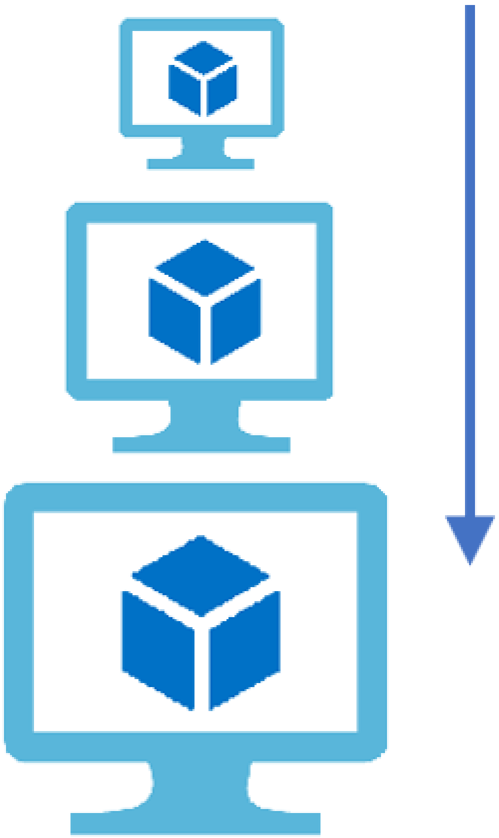
Scaling up the Monitoring Framework

The HEP Experimental Realm

- A DAQ system of a modern HEP experiment includes:
  - O(1K) computers and network devices
  - O(10K) SW applications
  - O(100K) HW sensors
- When choosing SW technologies for the Monitoring Framework one has to consider:
  - Number of monitoring data providers
  - Number of metrics and their update rates
  - The total amount of monitoring data produced for the life-time of the experiment
- There are two approaches for scaling:
  - Horizontal and Vertical

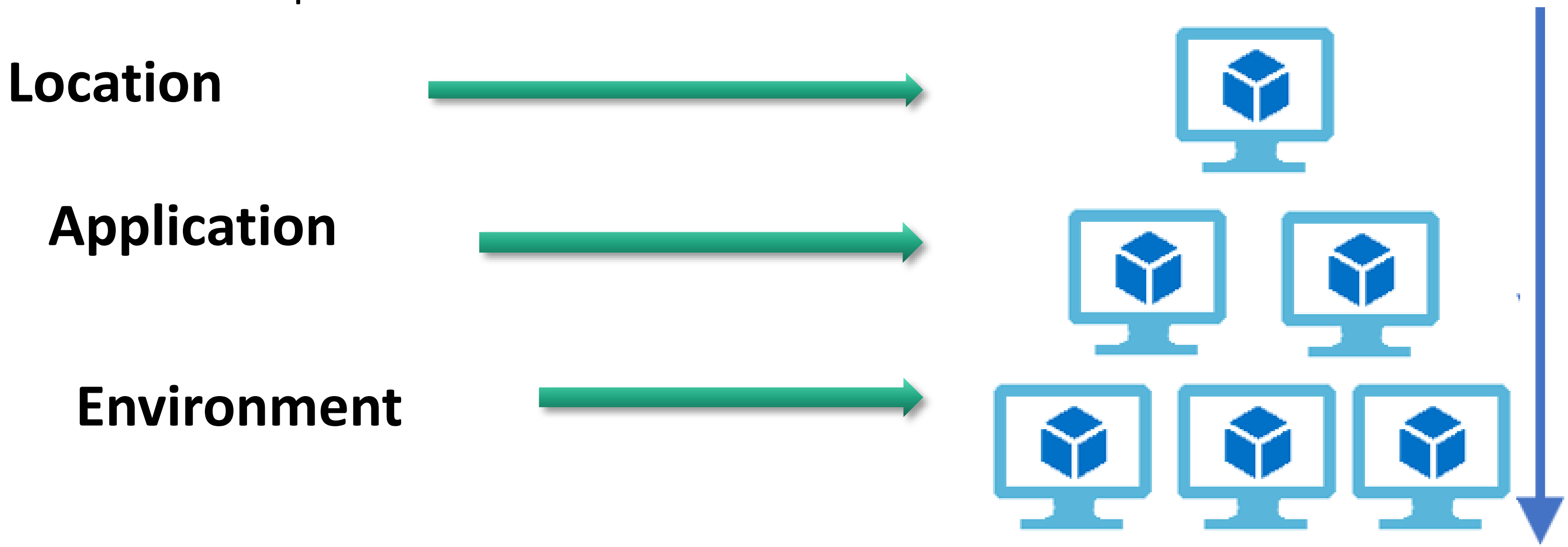
Vertical Scaling

- Vertical Scaling is achieved by increasing the power of individual computers
- Vertical scalability heavily depends on the tools used for monitoring data processing:
  - Some tools may scale better than the others
  - Should be taken into account when choosing the right tools for your project



Horizontal Scaling Works Better!

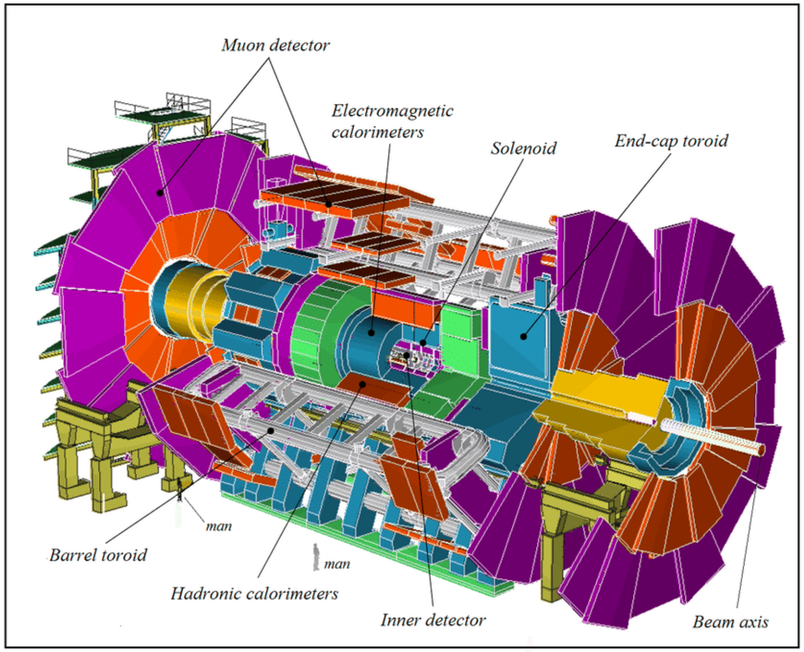
- Horizontal scaling works by adding more computers to the system
- Horizontal scaling naturally maps to the Naming Schemas of Events Sources and Metrics Names:
  - Extra server(s) can be at different levels of the Naming Schema to handle a respective sub-set of Events and Metrics



DAQ Special:
Data Quality Monitoring

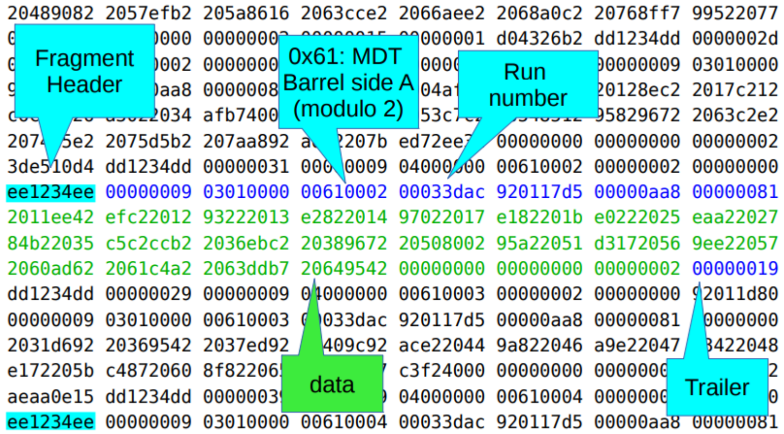
How to Monitor the Detector?

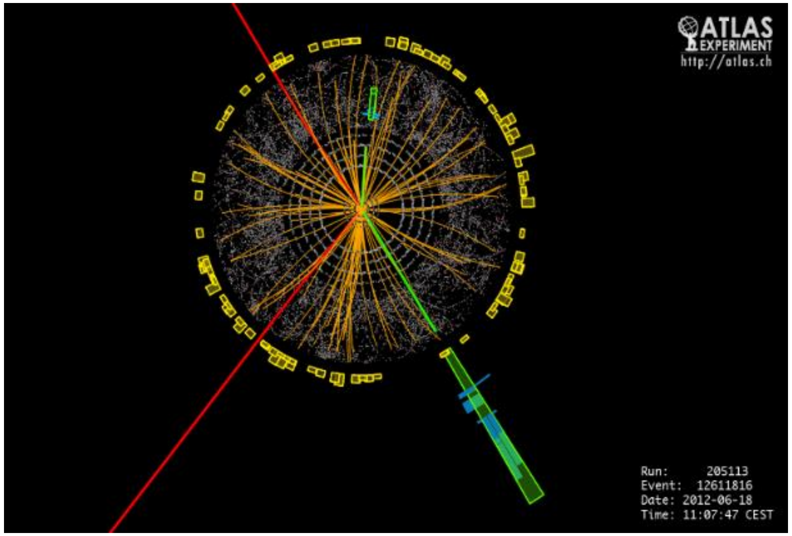
- Detectors of LHC experiments are incredibly complex devices:
  - Up to 10^8 output data channels
  - Mostly custom electronics
  - 40 MHz operational frequency
- Traditional monitoring would yield in O(1) PHz (petahertz) of metrics update rate:
  - These metrics are not even attempted to be produced explicitly
- However DAQ system has a handle on these metrics…



Detector Metrics

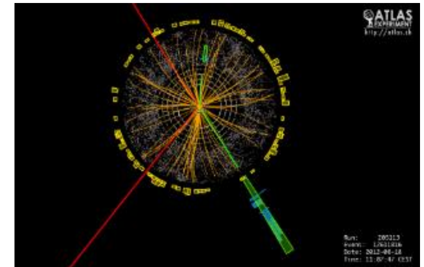
- Each **Physics Event** taken from the detector by the DAQ system contains metrics for a sub-set of detector channels:
  - An expert can spot problems by looking into a graphical event representation
  - This is of course very difficult and unreliable





Automated Data Quality Analysis

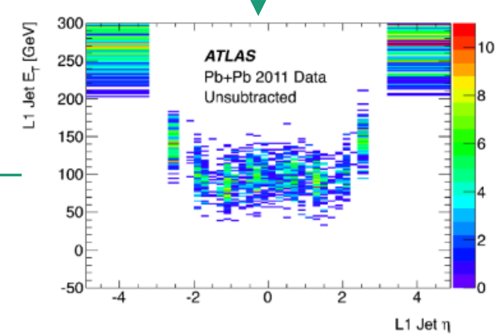
- Dedicated DAQ applications apply standard physics analysis algorithms to a statistical sub-set of the Physics Events:
  - Extract Detector Metrics and build their statistical distributions(histograms)
  - Analyze histograms and produce a new set of Metrics – Data Quality statuses



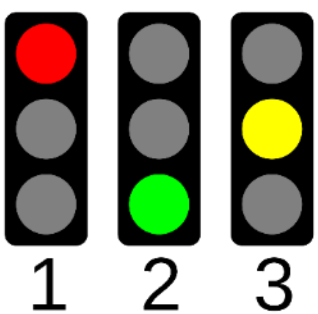
Samples of Physics Events

Physics Event Analysis Algorithms

Statistical Distribution



Statistical Analysis Algorithms



Wrap-up

Key Points to Keep in Mind when designing A Monitoring Framework

- Use standard Monitoring APIs if possible:
  - e.g. Logging API
- Think carefully when designing a custom API:
  - It should not depend on a particular technology
- Use off the shelf solutions for the Monitoring Framework components, e.g. Analytics and Visualization:
  - A custom made implementation should be well justified
- It is acceptable that only Monitoring APIs are available at the very beginning:
  - Monitoring Framework implementation will evolve in the course of DAQ system development for the mutual benefit