Monitoring with Prometheus and Grafana
ISOTDAQ 2024
14th International School of Trigger and Data AcQuisition
19-28 June 2024, Hefei China

Camilo Carrillo

CERN/Imperial College UK

26/06/24

Motivation:

Why today's presentation? What do you monitor in your daily life?

- Time tables: Buses, Trains, Planes, your calendar... etc.
- Weather: Temperatures, Pressure, Humidity.
- Your phone/laptop: Charge-status. Internet-connection status, type.
- Stock market. FOREX, shares, etc.
- Your experiment / production system: Is it recording data? Is it working?

Time series databases (TSDB) are everywhere... Why do you monitor them?

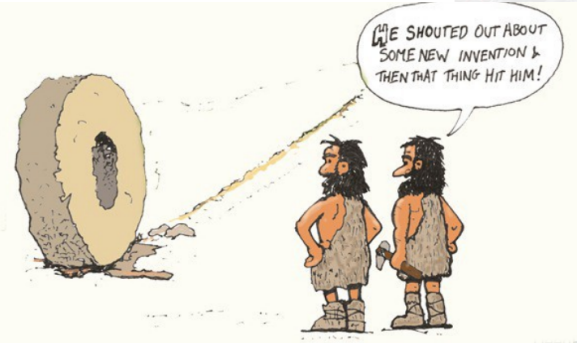
- If you know the **past**, you can: **understand problems** and predict the future.
- If you know the **present**, you can: **make problems last as short as possible**.
- Nobody knows the **future**, but good **alerts** will give you good hints.

Monitoring is just everywhere now days. **Home made solutions for monitoring must be well justified**.

DO NOT REINVENT THE WHEEL FOR YOUR PRODUCTION/DAQ SYSTEM!

Today's presentation

1. Prometheus
2. Grafana
3. use case: Temperature, Humidity, Pressure, Energy
4. use case: Energy
5. use case: The CMS experiment
6. Conclusions



Why Grafana and Prometheus?

- Two different technologies developed and supported by two different communities (Grafana Labs, Independent/SoundCloud).
- Grafana and Prometheus are the most prominent tools in the application monitoring and analytics space.
- Prometheus is an **open source** monitoring and alerting platform which collects and stores metrics as **time-series**[1] data.
- Grafana is an **open source** analytics and interactive visualization web application. It allows you to ingest data sources, query this data, and display it on beautiful customized charts for easy analysis.
- This forms the basis of the perfect duo in application monitoring, the **Prometheus - Grafana** relationship, which we intend to dive deeper into over the course of this talk.
- Prometheus can be seen as a back-end, Grafana can be seen as the front-end (with some approximations)
- Both technologies are becoming popular!

+

[1] [time value labels]

Who uses Grafana and Prometheus today?

Prometheus

Prometheus is an open source systems and service monitoring solution designed for developers. It enables users to collect metrics from configured targets at given intervals, evaluate rules expressions, display the results, and trigger alerts. Features include the Prometheus server which scrapes and stores time series data, client libraries for instrumenting ... show more

COMPANIES WE TRACK USING PROMETHEUS

26,690 +0.48% 12 MONTHS CHANGE

Create a Target Segment

PRODUCTS RELATED TO PROMETHEUS

Clarity SC | NetIQ AppManager | Apache Chainsaw | Azure Web App for Containers | Radia Client Automation

Companies Currently Using Prometheus

Download CSV Sample (25 companies)

COMPANY NAME	WEBSITE	HQ ADDRESS	CITY	STATE	ZIP	COUNTRY	TOP LEVEL INDUSTRY	SUB LEVEL INDUSTRY
Walmart	walmart.com	702 SW 8TH St	Bentonville	AR	72716-6...	US	Retail	Department Stores & Superstores
NVIDIA	nvidia.com	2701 San Tomas Ex...	Santa Clara	CA	95050	US	Technical	Computer Hardware Manufacturers
Ford Motor Company	ford.com	1 American Rd	Dearborn	MI	48126-2...	US	Manufacturing	Automobiles & Auto Parts
Bristol Myers Squibb	bms.com	430 E. 29th St., 14th...	New York	NY	10016	US	Manufacturing	Health & Nutrition Products
McDonald's	mcdonalds.com	110 N. Carpenter St	Chicago	IL	60607	US	Hospitality	Restaurants
o9 Solutions Inc.	o9solutions.com	1501 Lyndon B. Joh...	Dallas	TX	75234	US	Technical	Software Development & Technical Cor

`https://discovery.hgdata.com/` 19101 companies a year ago.

Who uses Grafana and Prometheus today? (31751 t-1y)

Grafana

Grafana is a time series analytics and monitoring platform designed for database administrators. It enables users to query, visualize, alert on, and understand metrics of the stored data and create reusable dashboards with template variables. Features include different data sources mixing, client-side graphs visualization, authentication, alert rules definin... show more

COMPANIES WE TRACK USING GRAFANA

35,915 +2.05% 12 MONTHS CHANGE

Create a Target Segment

PRODUCTS RELATED TO GRAFANA

SentryOne SQL Sentry | Apache Phoenix | SolarWinds Database Performance Analyzer | Adabas Online System | Idera DB Change Manager

Companies Currently Using Grafana

Download CSV Sample (25 companies)

COMPANY NAME	WEBSITE	HQ ADDRESS	CITY	STATE	ZIP	COUNTRY	TOP LEVEL INDUSTRY	SUB LEVEL INDUSTRY
Fiserv	fiserv.com	255 Fiserv Dr	Brookfield	WI	53045	US	Finance	Banking
Walmart	walmart.com	702 SW 8TH St	Bentonville	AR	72716-6...	US	Retail	Department Stores & Superstores
Banco Santander	santander.com	PASEO PEREDA, 9 -...	Madrid	Madrid	39004	ES	Finance	Banking
Experian	experian.com	2 Cumberland Pl. Fe...	Dublin	Leinster	D02 HY...	IE	Technical	Software Manufacturers
Maersk Line	maersk.com	Esplanaden 50	Copenhagen	Capital ...	1263	DK	Transportation	Marine Shipping & Transportation
Pluralsight	pluralsight.com	42 Future Way	Draper	UT	84020	US	Technical	Software Manufacturers

https://discovery.hgdata.com/ https://grafana.com/success/

Prometheus

prometheus.io

Prometheus: (...the back-end, the TSDB)

When to use prometheus: It works well for **recording any purely numeric time series**. It fits both machine-centric monitoring as well as monitoring of highly dynamic service-oriented architectures.

- Prometheus: An **open-source** system's monitoring and alerting toolkit.
- Maintained **independently of any company**.
- In few words: It **collects and stores** its metrics as time series data, i.e. metrics information is stored with the **timestamp** at which it was recorded AND optional **key-value** pairs called **labels**.
- 3 metric types: **Counter, Gauge, Histogram**
  `https://prometheus.io/docs/concepts/metric_types/`

Main features:

- Multi-dimensional data model with time series data identified by metric name and key/value pairs
- PromQL, a flexible query language to leverage this dimensionality
- Targets are discovered via service discovery or static configuration.

Installing Prometheus

- Available in all **popular Linux package-manager**, (apt-get, yum, pacman,etc) as simply "prometheus".
- Other OS (Windows/Mac) Binaries also available.
- A main configuration file is written in YAML format.
- Written in GO. Suitable for docker images!
- Prometheus is portable: A binary + a configuration file

Generic place holders for configuration file:

- `<boolean>` : a boolean that can take the values `true` or `false`
- `<duration>` : a duration matching the regular expression `(([0-9]+)y)?(([0-9]+)w)?(([0-9]+)d)?(([0-9]+)h)?(([0-9]+)m)?(([0-9]+)s)?(([0-9]+)ms)?|0)`, e.g. `1d`, `1h30m`, `5m`, `10s`
- `<filename>` : a valid path in the current working directory
- `<float>` : a floating-point number
- `<host>` : a valid string consisting of a hostname or IP followed by an optional port number
- `<int>` : an integer value
- `<labelname>` : a string matching the regular expression `[a-zA-Z_][a-zA-Z0-9_]*`
- `<labelvalue>` : a string of unicode characters
- `<path>` : a valid URL path
- `<scheme>` : a string that can take the values `http` or `https`
- `<secret>` : a regular string that is a secret, such as a password
- `<string>` : a regular string
- `<size>` : a size in bytes, e.g. `512MB`. A unit is required. Supported units: B, KB, MB, GB, TB, PB, EB.

Prometheus Configuration

In the configuration file you find sections. The most general one is called the **global section**:

```
global:
  # How frequently to scrape targets by default.
  [ scrape_interval: <duration> | default = 1m ]

  # How long until a scrape request times out.
  [ scrape_timeout: <duration> | default = 10s ]

  # How frequently to evaluate rules.
  [ evaluation_interval: <duration> | default = 1m ]

  # The labels to add to any time series or alerts when communicating with
  # external systems (federation, remote storage, Alertmanager).
  external_labels:
    [ <labelname>: <labelvalue> ... ]

  # File to which PromQL queries are logged.
  # Reloading the configuration will reopen the file.
  [ query_log_file: <string> ]
```

Prometheus Configuration

The scrape_configs section:

```
# The job name assigned to scraped metrics by default.
job_name: <job_name>

# How frequently to scrape targets from this job.
[ scrape_interval: <duration> | default = <global_config.scrape_interval> ]

# Per-scrape timeout when scraping this job.
[ scrape_timeout: <duration> | default = <global_config.scrape_timeout> ]

# The HTTP resource path on which to fetch metrics from targets.
[ metrics_path: <path> | default = /metrics ]

# honor_labels controls how Prometheus handles conflicts between labels that are
# already present in scraped data and labels that Prometheus would attach
# server-side ("job" and "instance" labels, manually configured target
# labels, and labels generated by service discovery implementations).
#
# If honor_labels is set to "true", label conflicts are resolved by keeping label
# values from the scraped data and ignoring the conflicting server-side labels.
#
# If honor_labels is set to "false", label conflicts are resolved by renaming
# conflicting labels in the scraped data to "exported_<original-label>" (for
# example "exported_instance", "exported_job") and then attaching server-side
# labels.
#
# Setting honor_labels to "true" is useful for use cases such as federation and
# scraping the Pushgateway, where all labels specified in the target should be
# preserved.
```

Prometheus Configuration

The scrape_configs section:

```
- job_name: node-exporter
  static_configs:
  - targets:
    - l1ts-grafana:9100
    - l1ts-prometheus:9100
    - l1page:9100
    - l1page-test:9100
    - l1ce:9100
    - l1ce-test:9100
    - l1ts-central:9100
    - l1ts-ugt:9100
    - l1ts-ugt-test:9100
    - l1ts-calol1:9100
    - l1ts-calol2:9100
    - l1ts-ugmt:9100
    - l1ts-bmtf:9100
    - l1ts-omtf:9100
```

- The targets can be seen as the list of nodes (PCs) running a given job and exposing metrics in a given port (9100 in this case).
- The list of nodes can be set as a static list, as in the example on the left. Or Dynamically by automatically discovering the jobs running in the same network/VPN **puppet DB**.
- A target can be running in the same prometheus-server, just add it as `http://localhost:9100`
- **prometheus-node-exporter** (also available in main repositories) is a popular exporter

Prometheus Configuration

- Prometheus adapts to a huge list of scenarios.
- Common Authentication Methods and encrypted metric exposure also available.
- Central prometheus server/node managing all metrics exposed by targets.
- Prometheus central node redundancy easily achieved.
- Later in the slides we will see in more details two use cases.

- Configuration file
  - <scrape_config>
  - <tls_config>
  - <oauth2>
  - <azure_sd_config>
  - <consul_sd_config>
  - <digitalocean_sd_config>
  - <docker_sd_config>
  - <dockerswarm_sd_config>
  - <dns_sd_config>
  - <ec2_sd_config>
  - <openstack_sd_config>
  - <ovhcloud_sd_config>
  - <puppetdb_sd_config>
  - <file_sd_config>
  - <gce_sd_config>
  - <hetzner_sd_config>
  - <http_sd_config>
  - <ionos_sd_config>
  - <kubernetes_sd_config>
  - <kuma_sd_config>
  - <lightsail_sd_config>
  - <linode_sd_config>
  - <marathon_sd_config>
  - <nerve_sd_config>
  - <nomad_sd_config>
  - <serverset_sd_config>
  - <triton_sd_config>
  - <eureka_sd_config>
  - <scaleway_sd_config>
  - <uyuni_sd_config>
  - <vultr_sd_config>
  - <static_config>
  - <relabel_config>
  - <metric_relabel_configs>
  - <alert_relabel_configs>
  - <alertmanager_config>
  - <remote_write>
  - <remote_read>
  - <tsdb>
  - <exemplars>
  - <tracing_config>

Prometheus metrics exposure, node-exporter

After installing the package. Run it / Start the service.

```
miloc@lenovo:~> $systemctl status prometheus-node-exporter
● prometheus-node-exporter.service - Prometheus exporter for machine metrics
     Loaded: loaded (/usr/lib/systemd/system/prometheus-node-exporter.service; >
     Active: active (running) since Mon 2023-06-19 08:47:22 CEST; 20h ago
   Main PID: 745 (prometheus-node)
      Tasks: 6 (limit: 18808)
     Memory: 27.6M
        CPU: 33ms
     CGroup: /system.slice/prometheus-node-exporter.service
             └─745 /usr/bin/prometheus-node-exporter

Warning: some journal files were not opened due to insufficient permissions.
miloc@lenovo:~> $
```

visit: prometheus-node-exporter `http://localhost:9100` (in your laptop)

Node Exporter

Prometheus Node Exporter

Version: (version=1.6.0, branch=tarball, revision=1.6.0)

- Metrics

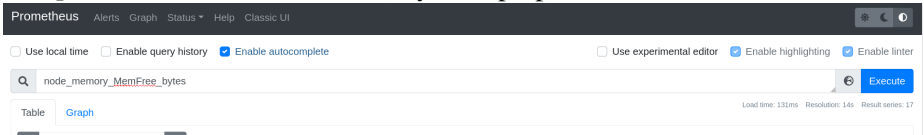
Prometheus metrics exposure, node-exporter "API”

Example: prometheus-node-exporter `http://localhost:9100/metrics` (in your laptop)

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 0
go_gc_duration_seconds{quantile="0.25"} 0
go_gc_duration_seconds{quantile="0.5"} 0
go_gc_duration_seconds{quantile="0.75"} 0
go_gc_duration_seconds{quantile="1"} 0
go_gc_duration_seconds_sum 0
go_gc_duration_seconds_count 0
# HELP go_goroutines Number of gorouting that currently exist.
# TYPE go_goroutines gauge
go_goroutines 7
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.20.4"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 1.006888e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 1.006888e+06
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.445232e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 521
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 7.171432e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 1.006888e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 1.196032e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 2.637824e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 8458
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 1.196032e+06
# HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system.
# TYPE go_memstats_heap_sys_bytes gauge
```

PromQL (Prometheus Query Language)

- Prometheus provides a functional query language called PromQL.
- It lets the user select and aggregate time series data in real time. The result of an expression can either be shown as a graph, viewed as tabular data in Prometheus's expression browser, or consumed by external systems, like Grafana, via the HTTP API.
- Queries can evaluate to:
  - **Instant vector** :a set of time series containing a single sample for each time series, all sharing the same timestamp
  - **Range vector** - a set of time series containing a range of data points over time for each time series
  - **Scalar** - a simple numeric floating point value
- PromQL queries can be explored in the prometheus server. Once the package is installed and running (service started) you can see it in port 9090 (if you are running in your own laptop `http://localhost:9090` (in your laptop):



PromQL (Prometheus Query Language)

Prometheus Alerts Graph Status Help Classic UI

node_memory_MemFree_bytes

Execute

Table Graph

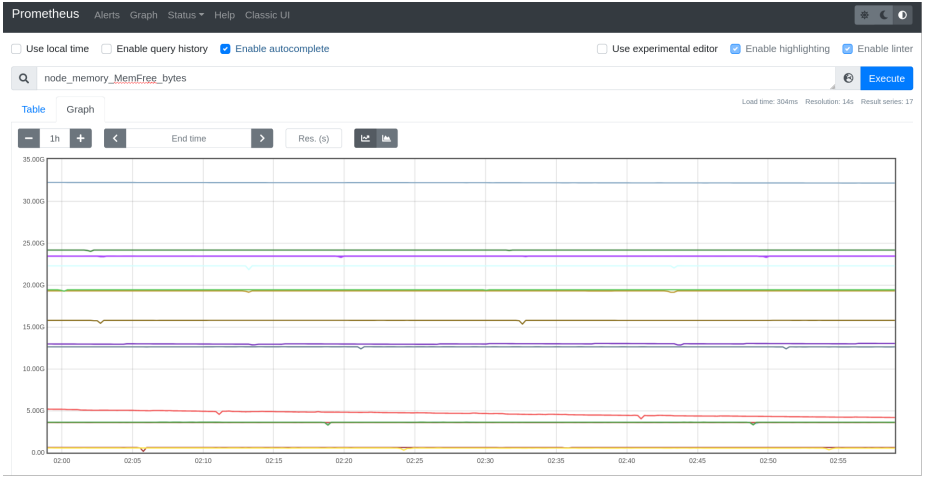
Load time: 148ms Resolution: 14s Result series: 17

Evaluation time

node_memory_MemFree_bytes{**instance**="11ce-test:9100", **job**="node-exporter"}	3582156800
node_memory_MemFree_bytes{**instance**="11ce:9100", **job**="node-exporter"}	3610353664
node_memory_MemFree_bytes{**instance**="11page-test:9100", **job**="node-exporter"}	3582332928
node_memory_MemFree_bytes{**instance**="11page:9100", **job**="node-exporter"}	3611369472
node_memory_MemFree_bytes{**instance**="11ts-bmtf:9100", **job**="node-exporter"}	23415132160
node_memory_MemFree_bytes{**instance**="11ts-calol1:9100", **job**="node-exporter"}	19278241792
node_memory_MemFree_bytes{**instance**="11ts-calol2:9100", **job**="node-exporter"}	32144568320
node_memory_MemFree_bytes{**instance**="11ts-central:9100", **job**="node-exporter"}	610902016
node_memory_MemFree_bytes{**instance**="11ts-cppf:9100", **job**="node-exporter"}	24152424448
node_memory_MemFree_bytes{**instance**="11ts-emtf:9100", **job**="node-exporter"}	13013467136
node_memory_MemFree_bytes{**instance**="11ts-grafana:9100", **job**="node-exporter"}	532193280
node_memory_MemFree_bytes{**instance**="11ts-omtf:9100", **job**="node-exporter"}	22264958976
node_memory_MemFree_bytes{**instance**="11ts-prometheus:9100", **job**="node-exporter"}	4194197504
node_memory_MemFree_bytes{**instance**="11ts-twinmux:9100", **job**="node-exporter"}	19415490560
node_memory_MemFree_bytes{**instance**="11ts-ugmt:9100", **job**="node-exporter"}	23434600448
node_memory_MemFree_bytes{**instance**="11ts-ugt-test:9100", **job**="node-exporter"}	15766491136
node_memory_MemFree_bytes{**instance**="11ts-ugt:9100", **job**="node-exporter"}	12606533632

Remove Panel

PromQL (Prometheus Query Language)

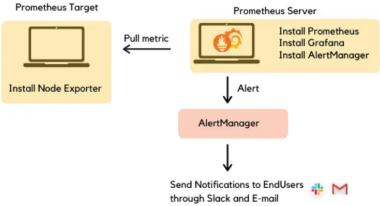


PromQL (Prometheus Query Language)

Prometheus Alertmanager

The Alertmanager handles alerts sent by client applications such as the Prometheus server. It takes care of deduplicating, grouping, and routing them to the correct receiver integration such as email, PagerDuty, or OpsGenie. It also takes care of silencing and inhibition of alerts.

- Alertmanager runs as an independent process/service.
- Rules for alerting can be defined in YAML files pointed in the prometheus configuration file.
- Rules definitions follow PromQL
- Alerts can be routed in many different ways: Mattermost, Telegram, SMS, email, etc.

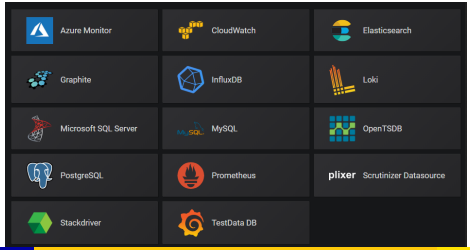


Grafana

grafana.com

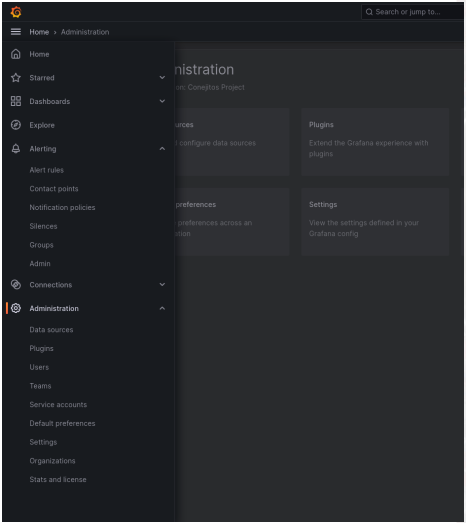
Grafana

- Grafana is a **time series analytics** and monitoring platform designed for database administrators.
- It enables users to query, visualize, alert on, and understand metrics of the stored data and create **reusable dashboards** with template variables.
- Features include different data sources mixing, client-side graphs visualization, authentication, alert rules defining, email notifications, and graphical annotations.
- Go powers Grafanas backend + Javascript for last layer frontend.
- Prometheus **is not** the only data-source that can be used with grafana:

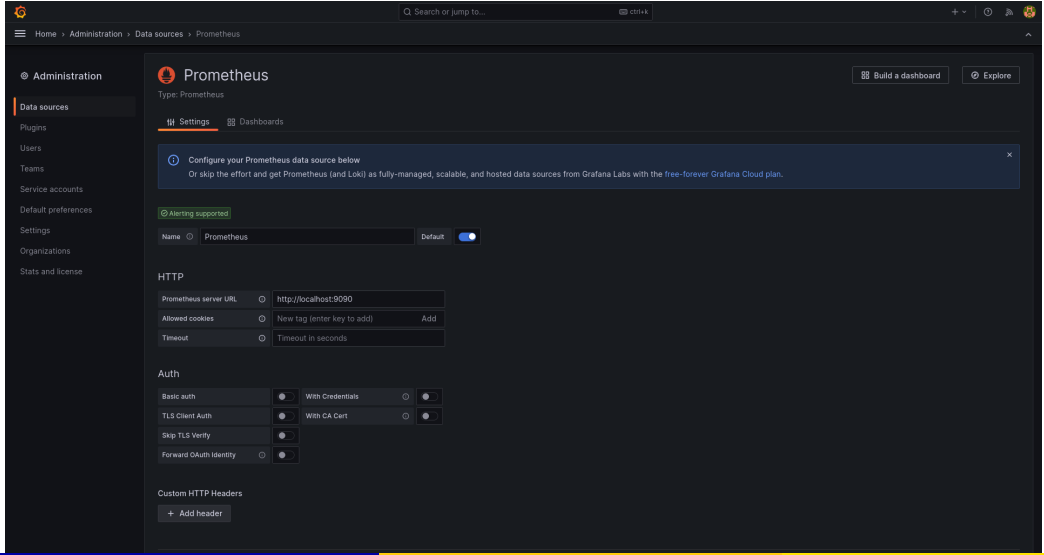


Installing Grafana

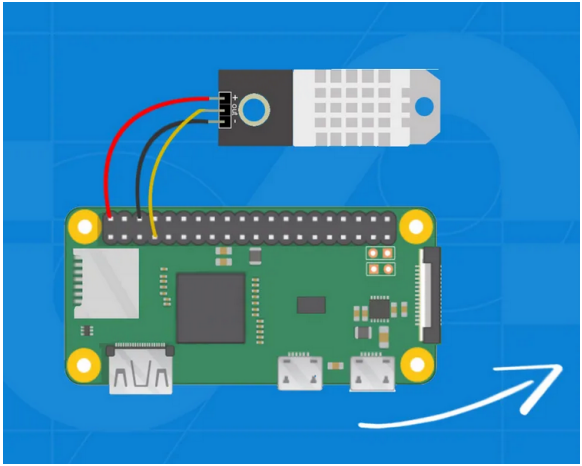
- Grafana: **Available in mainstream repositories**. Todays version v10.0.0 (apt-get, yum, pacman etc)
- You can install it and run it in your own laptop (together with prometheus).
- Default **GUI port is 3000**, you can start playing by visiting: `http://localhost:3000` (your laptop)
- This **GUI is also your grafana-development tool**. Be sure to log-in (admin/admin)
- First step is to set a **data-source**.



Prometheus data-source, and you are set!



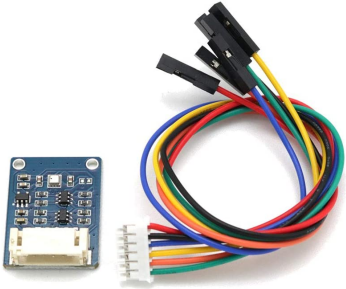
BME280 use case. (budget 32 EUR)

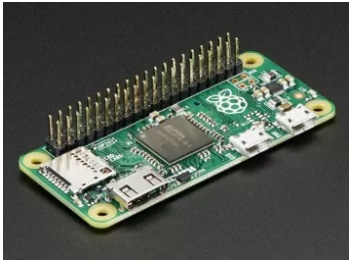


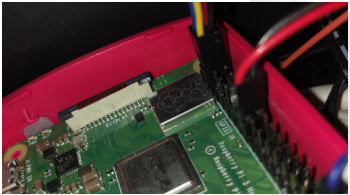
- Raspberrypi Zero (GPIO pins presoldered): **17 EUR**
- BME280 (temperature,humidity,pressure sensor): **14 EUR**

Try it at home!

use case: Temperature, Humidity, Pressure

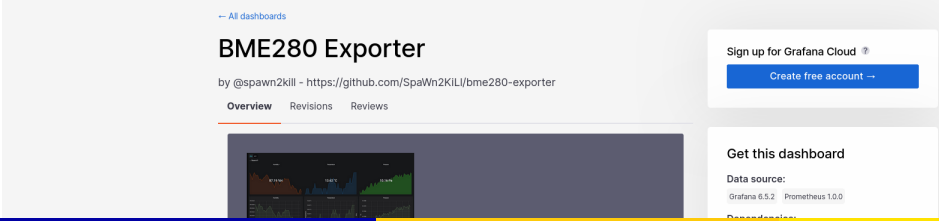




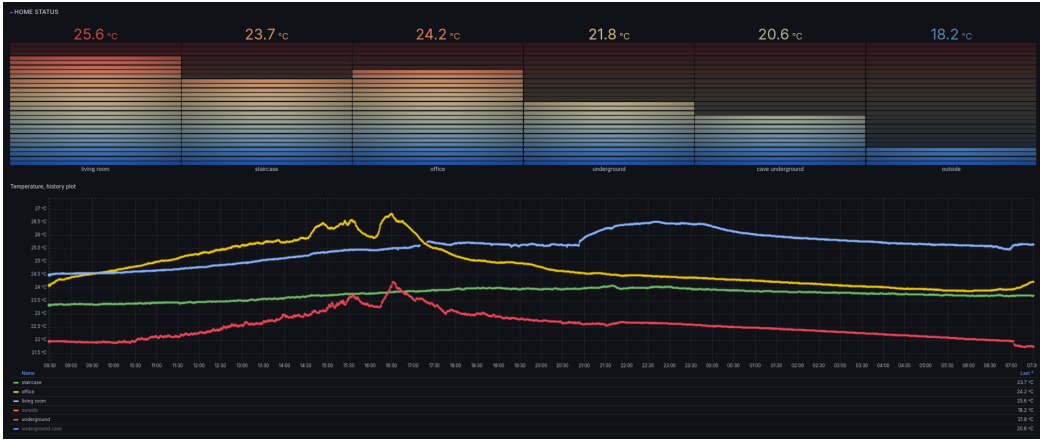


use case: Temperature, Humidity, Pressure (Instant and History Plots)

- Prometheus exporters already available for several sensors, here the bme280 one: `https://pypi.org/project/bme280_exporter/` (python based) install it in the rasbperrypi.
- Run it and check the exporter is actually working by visiting the metrics endpoint `http://rasbperrypizero:9500` (port 9500 by default)
- Add the raspberrypi target to your prometheus configuration (static_configs) (see Prometheus configuration file slides).
- Import or create a new dashboard to visualize the data. **Dashboards are defined as json files**, open and check them.

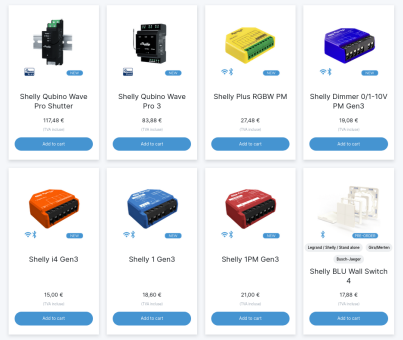


use case: Temperature, Humidity, Pressure



Energy use case. budget: 23 EUR

Measure energy consumption (a large variety of hardware in the market):
`https://www.shelly.com`



The prometheus exporter runs on top of MQTT.[2]
`https://fr.wikipedia.org/wiki/MQTT`

[2] In your very same Raspberry pi

Energy use case. MQTT

- Connect your hardware to your local network.
- Once your hardware is connected it serves a wifi SSID for configuration

- Install and configure MQTT.

```
apt-get install mosquitto
install mosquitto-clients
systemctl enable mosquitto
nano /etc/mosquitto/mosquitto.conf
systemctl daemon-reload
systemctl restart mosquitto
[root@rbp4] </home/miloc> #
```

```
[root@rbp4] </home/miloc> # cat  /etc/mosquitto/mosquitto.conf
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

listener 1883
allow_anonymous true

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log
log_type error
log_type warning
log_type notice
log_type information

include_dir /etc/mosquitto/conf.d
```

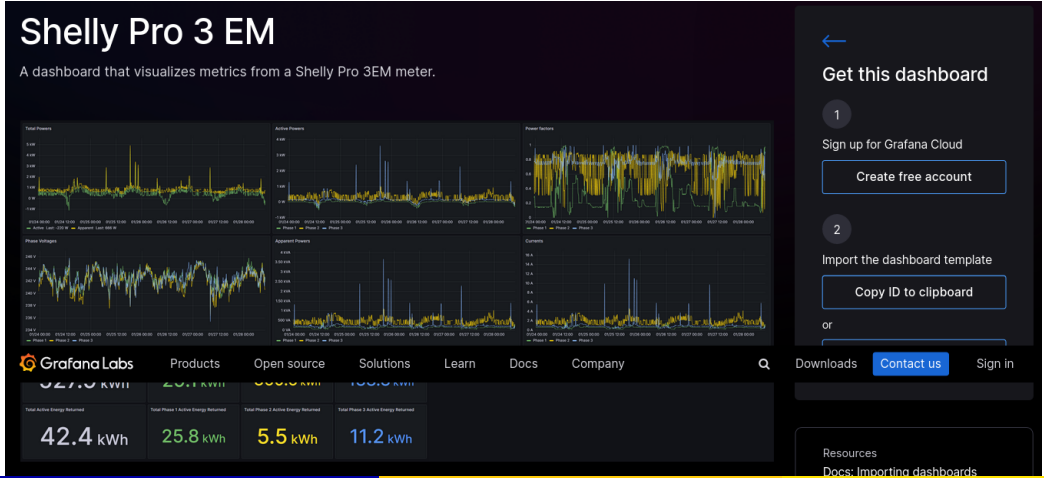
Energy use case. Prometheus exporter (python)

```
File Edit Options Buffers Tools Python Help
import json
import time
import paho.mqtt.client as mqtt
from prometheus_client import start_http_server, Gauge

# Configuration
MQTT_BROKER = '192.168.1.188'
MQTT_PORT = 1883
MQTT_TOPIC = 'shellypro3em-08f9e0e7a46c/events/rpc'
PROMETHEUS_PORT = 8000
metric_names = ['a_act_power', 'a_aprt_power', 'a_current', 'a_freq', 'a_pf', 'a_voltage', 'b_act_power','b_aprt_power', 'b_current', 'b_freq', 'b_pf', 'b_voltage', 'c_act_power', 'c_aprt_po\
wer', 'c_current', 'c_freq', 'c_pf', 'c_voltage', 'total_act_power', 'total_aprt_power', 'total_current'] #n_current excluded


# Prometheus metrics
pairs_gauges = []
for metric in metric_names:
    pairs_gauges.append((metric,Gauge(metric,metric)))
print(pairs_gauges)

# MQTT on_message callback
def on_message(client, userdata, msg):
    try:
        data = json.loads(msg.payload)
        #print(data)
        metrics = data['params']['em:0']
#        print(metrics)

        for metric,gauge in pairs_gauges:
            print(metric,' ---> ',metrics[metric])
            gauge.set(metrics[metric])
#        time.sleep(15)

    except Exception as e:
        print(f"Error processing message: {e}")

# MQTT connect callback
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT Broker!")
        client.subscribe(MQTT_TOPIC)
    else:
        print(f"Failed to connect, return code {rc}")
```

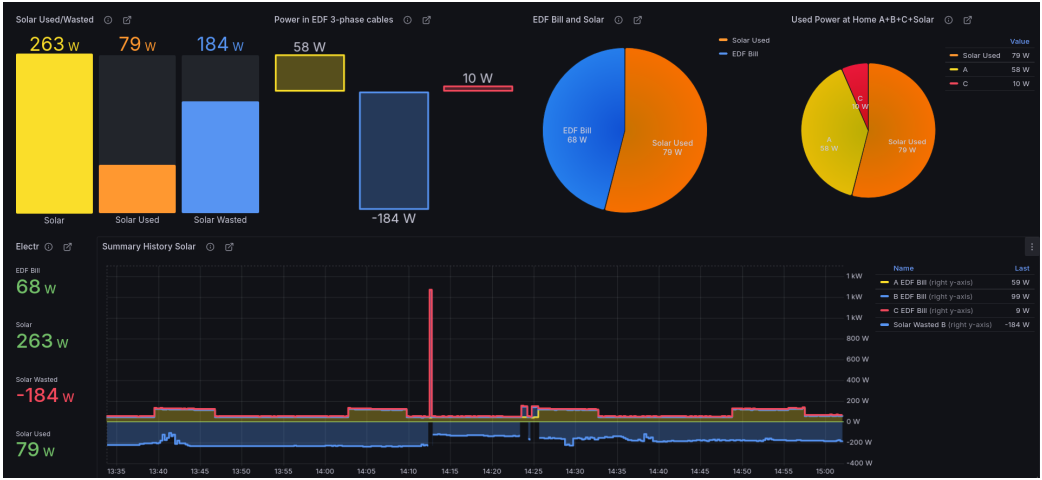
Energy use case. Grafana

Many dashboards already available.

https://grafana.com/grafana/dashboards/20444-shelly-pro-3-em/

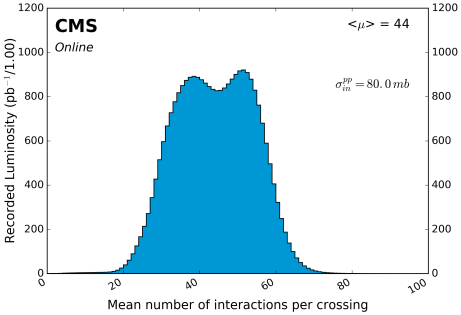


Energy use case. Grafana



CMS use case. (budget: 0.5 MCHF)

- Take a "photo" of each LHC pp/HI collision, **every 25 ns**
- Then, try to understand fundamental laws of physics



https://twiki.cern.ch/twiki/bin/view/CMSPublic/LumiPublicResults

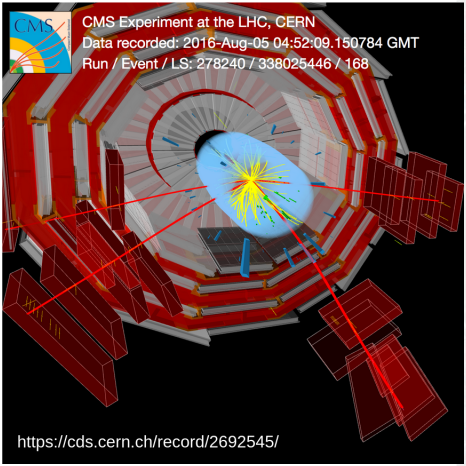


Fig: 4μ final state topology or..

A Higgs boson with a given probability

How data flows in CMS

40 MHz

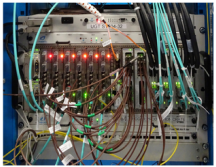
40-80 TB/s

Readout

40 MHz

~ 1 TB/s

Accept/reject



Level-1 trigger

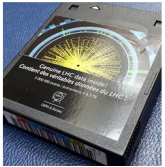
100 kHz

100 GB/s

Event builder &
High-level trigger

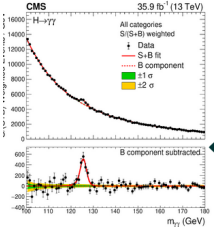
1 kHz

~ 1-2 GB/s



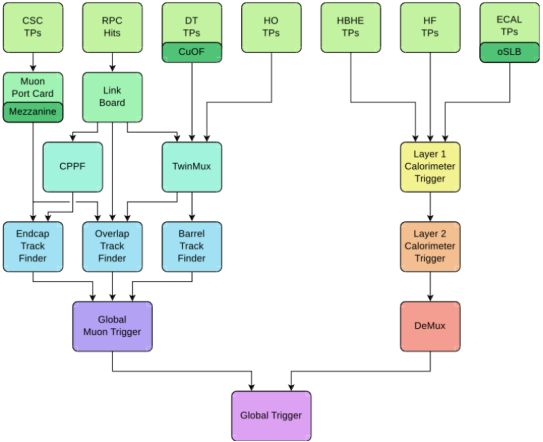
3 μHz

100 papers/y

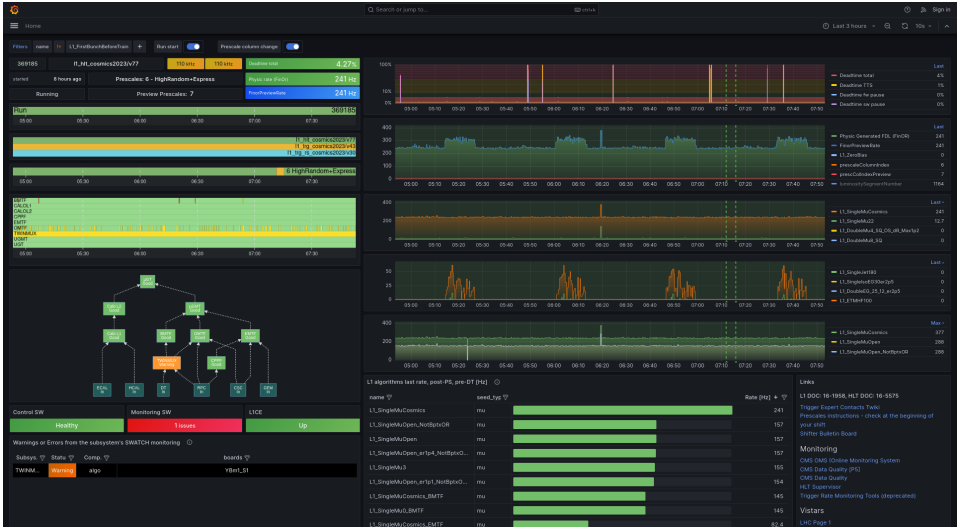


The L1 Trigger and Online Software

- ~ 150 boards
- ~ 3000 optical links
  - Up to 10Gbps
- Complex system
  - Input from 5 detector systems
    - 9 subsystems → different data-processing algorithms (calo clusters, local μ reco ...)
  - **5 different board designs**



L1 Monitoring Software, overview dashboard



- **prometheus-backend**

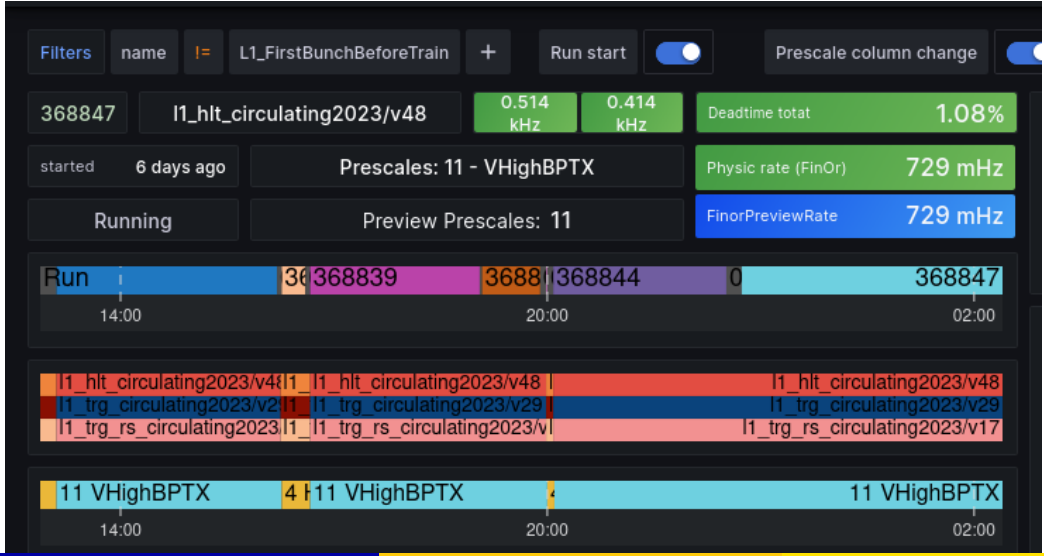
prometheus.io

- **grafana-frontend**

Grafana

grafana.com

CMS overview configuration

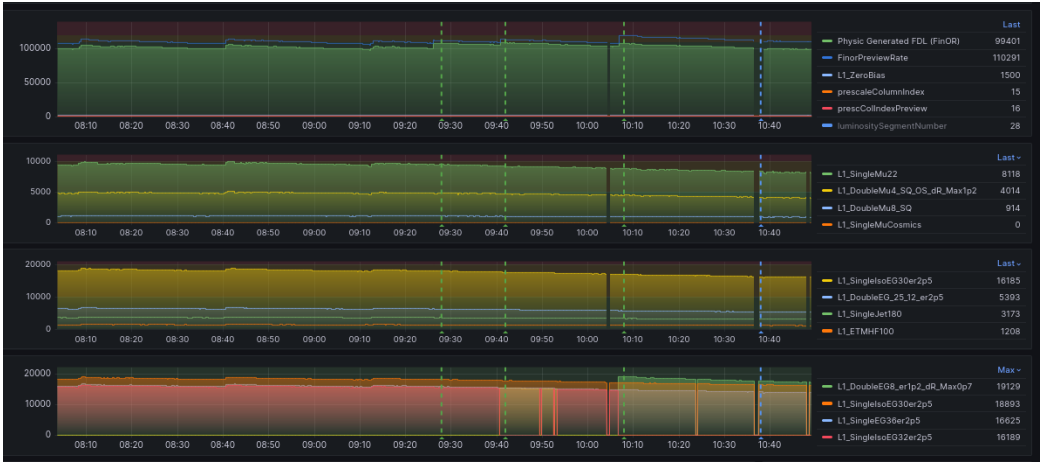


Bar charts

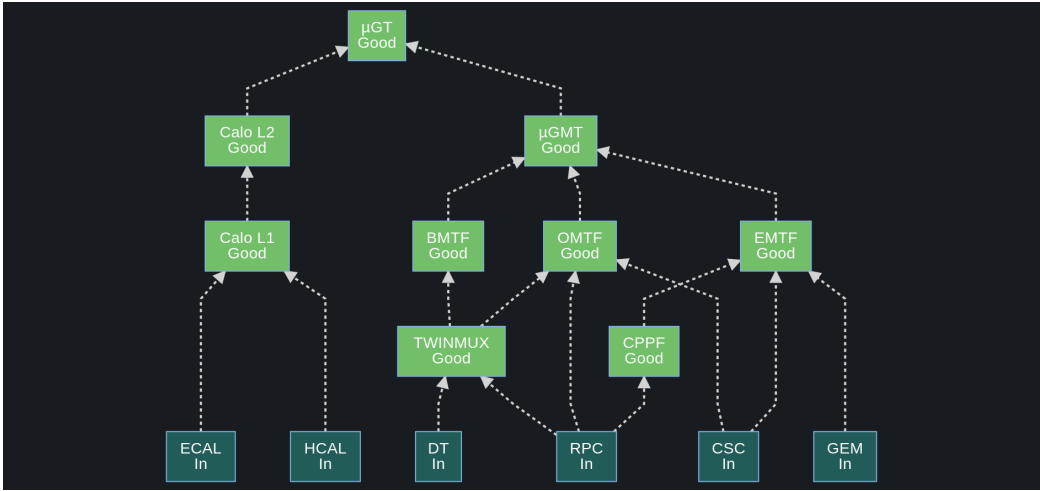
L1 algorithms last rate, post-PS, pre-DT [Hz]

name	seed_typ	Rate [Hz]
L1_SingleMuCosmics	mu	242
L1_SingleMuOpen_NotBptxOR	mu	156
L1_SingleMuOpen	mu	156
L1_SingleMuOpen_er1p4_NotBptxO...	mu	156
L1_SingleMu3	mu	154
L1_SingleMuOpen_er1p1_NotBptxO...	mu	152
L1_SingleMuCosmics_BMTF	mu	144
L1_SingleMu0_BMTF	mu	144
L1_SingleMuCosmics_EMTF	mu	84.9
L1_SingleMu5	mu	66.0
L1_SingleMu7	mu	44.1
L1_SingleMuCosmics_OMTF	mu	33.0
L1_SingleMu0_OMTF	mu	19.9

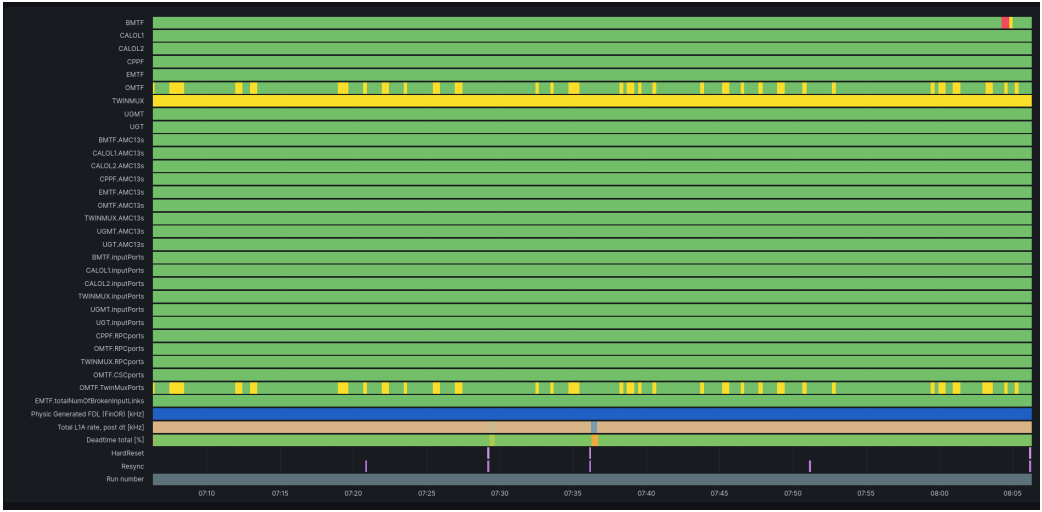
History Plots



Live status


µGT
Good
Calo L2
Good
µGMT
Good
Calo L1
Good
BMTF
Good
OMTF
Good
EMTF
Good
TWINMUX
Good
CPPF
Good
ECAL
In
HCAL
In
DT
In
RPC
In
CSC
In
GEM
In

Timeline status



Tables, color code status

BMTF input last rates (DT and RPC) - select the stations in the "bmtf_input_rate" on the top of the page

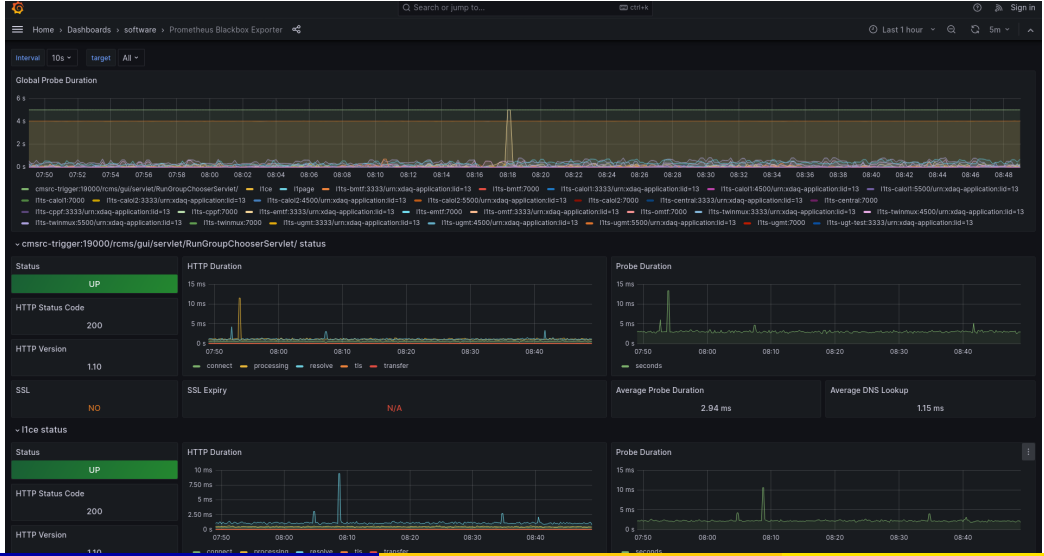
dtTotalRate

board\wheel	W-2 St1	W-1 St1	W0 St1	W+1 St1	W+2 St1	W-2 St2	W-1 St2	W0 St2	W+1 St2	W+2 St2	W-2 St3	W-1 St3	W0 St3	W+1 St3	W+2 St3	W-2 St4	W-1 St4	W0 St4	W+1 St4	W+2 St4
wedge01	0 Hz	28 kHz	19 kHz	28 kHz	0 Hz	18 kHz	13 kHz	10 kHz	11 kHz	17 kHz	4 kHz	5 kHz	4 kHz	4 kHz	4 kHz	5 kHz	4 kHz	4 kHz	4 kHz	5 kHz
wedge02	0 Hz	31 kHz	22 kHz	29 kHz	0 Hz	17 kHz	11 kHz	10 kHz	12 kHz	16 kHz	4 kHz	5 kHz	4 kHz	4 kHz	4 kHz	5 kHz	4 kHz	4 kHz	4 kHz	5 kHz
wedge03	0 Hz	27 kHz	7 kHz	29 kHz	0 Hz	18 kHz	10 kHz	10 kHz	12 kHz	17 kHz	4 kHz	4 kHz	4 kHz	4 kHz	4 kHz	6 kHz	6 kHz	6 kHz	6 kHz	7 kHz
wedge04	0 Hz	31 kHz	21 kHz	26 kHz	0 Hz	16 kHz	12 kHz	10 kHz	10 kHz	17 kHz	4 kHz	5 kHz	4 kHz	4 kHz	4 kHz	9 kHz	9 kHz	14 kHz	8 kHz	10 kHz
wedge05	0 Hz	30 kHz	22 kHz	31 kHz	0 Hz	17 kHz	13 kHz	10 kHz	11 kHz	17 kHz	4 kHz	5 kHz	4 kHz	4 kHz	4 kHz	6 kHz	5 kHz	7 kHz	5 kHz	7 kHz
wedge06	0 Hz	29 kHz	21 kHz	29 kHz	0 Hz	17 kHz	12 kHz	10 kHz	11 kHz	16 kHz	4 kHz	4 kHz	4 kHz	4 kHz	4 kHz	5 kHz	4 kHz	4 kHz	5 kHz	5 kHz
wedge07	0 Hz	28 kHz	19 kHz	30 kHz	0 Hz	17 kHz	11 kHz	10 kHz	11 kHz	17 kHz	4 kHz	4 kHz	4 kHz	5 kHz	4 kHz	5 kHz	4 kHz	4 kHz	4 kHz	5 kHz
wedge08	0 Hz	33 kHz	20 kHz	30 kHz	0 Hz	17 kHz	12 kHz	7 kHz	11 kHz	17 kHz	4 kHz	4 kHz	4 kHz	4 kHz	4 kHz	4 kHz	4 kHz	3 kHz	4 kHz	5 kHz
wedge09	0 Hz	30 kHz	21 kHz	32 kHz	0 Hz	17 kHz	12 kHz	10 kHz	12 kHz	16 kHz	4 kHz	3 kHz	4 kHz	4 kHz	4 kHz	1 kHz	1 kHz	984 Hz	1 kHz	1 kHz
wedge10	0 Hz	31 kHz	22 kHz	29 kHz	0 Hz	16 kHz	12 kHz	10 kHz	11 kHz	16 kHz	4 kHz	5 kHz	4 kHz	4 kHz	4 kHz	3 kHz	3 kHz	2 kHz	3 kHz	3 kHz
wedge11	0 Hz	32 kHz	20 kHz	27 kHz	0 Hz	17 kHz	11 kHz	10 kHz	12 kHz	17 kHz	4 kHz	4 kHz	4 kHz	4 kHz	4 kHz	1 kHz	1 kHz	980 Hz	1 kHz	1 kHz
wedge12	0 Hz	27 kHz	20 kHz	29 kHz	0 Hz	17 kHz	12 kHz	9 kHz	12 kHz	16 kHz	4 kHz	5 kHz	4 kHz	4 kHz	4 kHz	5 kHz	4 kHz	3 kHz	4 kHz	5 kHz

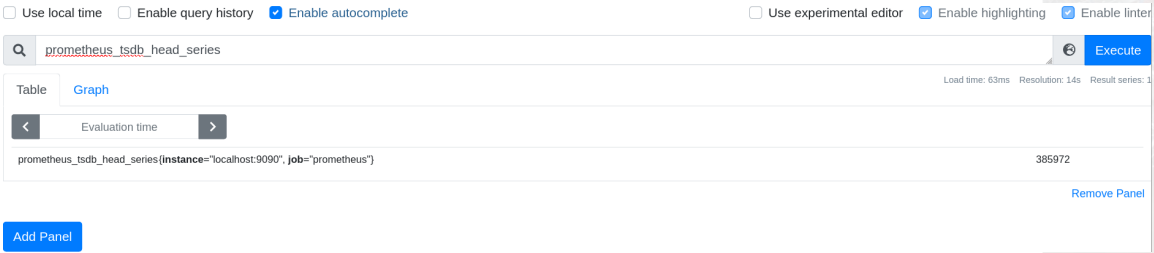
dtCorrRateSt2

Node exporter dashboards in CMS

Blackbox approach also used in CMS



L1 Monitoring Software, metrics



- **386k metrics**
- L1 trigger rates.
- All node-exporters metrics ($\approx$ 20 nodes)
- Status metrics
- Lumi metrics
- CMS Configuration key
- SQL exporters and other monitoring tools exporters.

Conclusions

- **prometheus & grafana**: a great combo for monitoring your DAQ / production system.
- Several use cases have been shown: Node-performance, Sensors, CMS.
- All software is **free and open-source**
- CMS Grafana/Prometheus dashboards and configurations are open:
  - `https://gitlab.cern.ch/cms-cactus/ops/monitoring/grafana`
  - `https://gitlab.cern.ch/cms-cactus/ops/monitoring/prometheus`
  - `https://gitlab.cern.ch/cms-cactus/ops/monitoring/alertmanager`
- CI/CD pipelines implemented for checks, RPM build, docker images.
- **Contributors to CMS online software** are very **welcome**
- Contact me if you have specific questions!

Backup

Backup

CMS Quadrant