

Fermilab Summer Internship Report

Evaluation of visualization software for CMS space monitoring

Intern: Alessandro Lazzari

alazzari@fnal.gov

Supervisor: Natalia Ratnikova

natasha@fnal.gov

September 23, 2016

Table of contents

- 1 Introduction
- 2 CMS space monitoring
- 3 Visualisation for CMS space monitoring
 - 3.1 Goals of visualisation
 - 3.2 Aim of this work
- 4 Software tools to evaluate
 - 4.1 Elasticsearch
 - 4.2 Kibana
 - 4.3 Grafana
- 5 Collecting monitoring data
 - 5.1 Storage records retrieving
 - 5.2 LFN to PFN conversion
 - 5.3 Quality check on storage records
 - 5.4 Storage record processing
 - 5.5 Additional APIs
- 6 Evaluating software
 - 6.1 Populating ES with data
 - 6.2 Creating dashboards in Kibana and Grafana
 - 6.3 Feature comparison between Kibana and Grafana
- 7 Conclusion
- 8 References
- Appendix A: Twiki

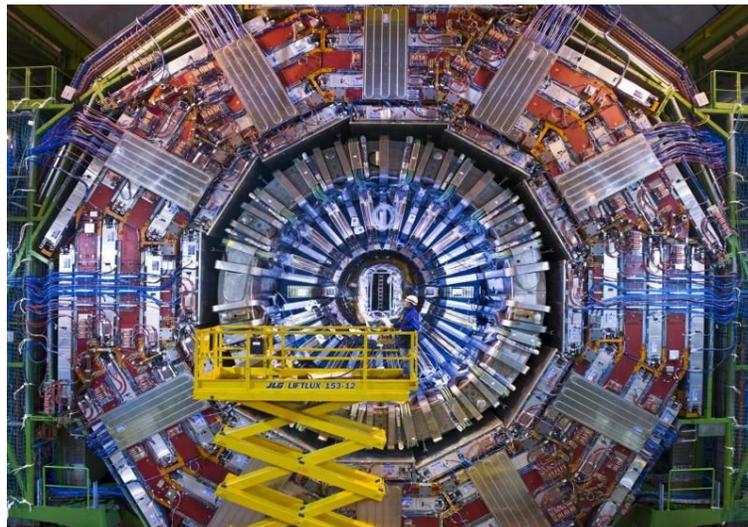
1 Introduction

The Compact Muon Solenoid (CMS) is a particle detector, operating at the Large Hadron Collider (LHC) at CERN, that was conceived to observe particles and phenomena produced in high-energy collisions. The range of physics investigated by the CMS experiment include extra dimensions, the search for the Higgs boson and particles that could make up dark matter.

The large amounts of data produced by CMS are entrusted to the CMS grid computing system [1] [2], which takes care of handling, distributing and storing those data across multiple geographically distributed sites, whose storage capacity ranges from hundreds of terabytes to a few petabytes.

To help CMS data management cope with such a vast storage infrastructure, a CMS space monitoring system, called SpaceMon [3], was designed and developed, in order to provide an overall view of the distributed storage based on sites local storage information. That information is aggregated locally at the sites, then uploaded into a central database at CERN.

As of today, there exists no consolidated form of visualization for CMS space monitoring data [4]. Purpose of this work is to assess the aptness of a group of software tools, namely Elasticsearch, Kibana and Grafana, to displaying such information, retrieved from the central database.



The CMS detector. From: <https://home.cern/about/experiments/cms>

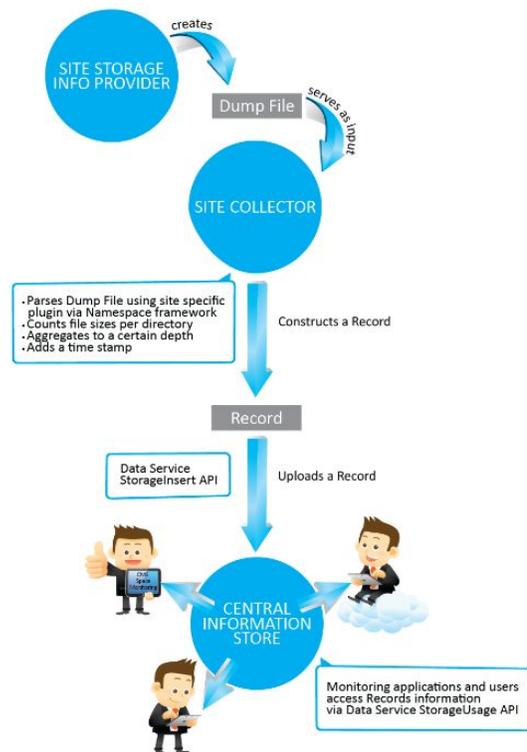
2 CMS space monitoring

SpaceMon, the CMS space monitoring system, collects information on actual site storage usage in the form of ‘storage records’. A storage record is a file containing metadata information on a particular file, including its path and size. Storage records are produced at the computing sites and then uploaded to the central database.

It is worth noting that those storage records are not limited to the official data: in fact they contain information regarding all the data present on the site storage—so as to get a realistic view of the site storage usage. Such storage records can then be retrieved from the central database by means of web data service APIs [5].

In addition to that, there exist central data catalogs containing the experiment knowledge of the storage content at each site in terms of Logical File Names, LFNs. LFN namespace conventions group data together by type, i.e. raw, MC simulated and more. On the other hand, real data on site storage devices are accessed via Physical File Names, PFNs. A Trivial File Catalog, TFC, maintained by each site, provides rules to convert LFNs to PFNs. Such conversion relies on the the CMS data placement and location system PhEDEx [6] [7].

Unlike central data catalogs, SpaceMon does not store any information on single files, but only aggregated space usage for all the files located under a certain folder, which is calculated at various levels of depth in the directory structure.



SpaceMon flow. From [3]

3 Visualisation for CMS space monitoring

3.1 Goals of visualisation

The main goal of visualisation, as far as the SpaceMon infrastructure is concerned, is to represent space monitoring information in a convenient and meaningful form according to the needs of CMS computing management as well as site administrators and individual storage users (use cases).

3.2 Aim of this work

The purpose of this work is to evaluate a predefined set of software tools, namely Elasticsearch, Kibana and Grafana, as regards their capabilities and suitability to provide graphical representation to CMS space monitoring data, in accordance with the computing system model that has so far been presented. In a sense, evaluation implies exploration and that is the reason why several dead ends were reached over the course of that process. Hereafter are reported only a few of them, alongside the proceedings and steps that proved to be successful, encompassing from collection of monitoring data to their visualisation.

4 Software tools to evaluate

4.1 Elasticsearch

Elasticsearch [11], say 'ES', is a highly scalable open-source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time. It is generally used as the underlying engine/technology that powers applications that have complex search features and requirements. In our case, it was used as the underlying engine to store monitoring data, retrieved from the data service, and serve as the data source for both Kibana and Grafana, which in turn visualize those data. Version used: 2.3.4



4.2 Kibana

Kibana [12] is an open source analytics and visualization platform designed to work with ES. Kibana is used to search, view, and interact with data stored in ES. You can easily perform advanced data analysis and visualize your data in a variety of charts, tables, and maps. Kibana makes it easy to understand large volumes of data. Its simple, browser-based interface enables you to quickly create and share dynamic dashboards that display changes to Elasticsearch queries in real time. Version used: 4.5.3



4.3 Grafana

Grafana [13] provides a powerful and elegant way to create, explore, and share dashboards and data with your team and the world. It is most commonly used for visualizing time series data for Internet infrastructure and application analytics but many use it in other domains including industrial sensors, home automation, weather, and process control. Version used: 3.1.1



Note: this report gives for granted that the tools above have already been installed on the machine.

5 Collecting monitoring data

5.1 Storage records retrieving

Storage records are obtained through a web data service API named *dumpspacequery*. A few options, related to site name (required) and time range (optional), must/may be specified. For more information, please refer to the official documentation [5].

In our case the following query was used:

```
.../datasvc/json/dumpspacequery?node=T*&level=10&time_since=1451606400
```

For the sake of convenience, its output is then rearranged differently, so as to have each ‘record’ on one line. At that point, it looks like this:

```
{"timestamp":1468369801,"name":"T1_SITE1","space":112025084852908,"dir":"/aaa"}
{"timestamp":1466369801,"name":"T1_SITE2","space":284472656106,"dir":"/bbb/cc"}
{"timestamp":1468369801,"name":"T2_SITE3","space":414513438,"dir":"/qqq/kllk"}
{"timestamp":1468367801,"name":"T1_SITE1","space":86058297,"dir":"/aaa/qqq"}
...
```

Note: actual data were replaced with fake data not to disclose any sensitive information. The same applies to the rest of this report.

Let us look at the meaning of the fields in each record:

- ‘timestamp’: unix time when the storage record was created
- ‘name’: name of the site the storage record belongs to
- ‘dir’: directory on the site storage (PFN)
- ‘space’: aggregated space, taken up by all the files under the directory ‘dir’

5.2 LFN to PFN conversion

It could be needed throughout the use cases to work out the LFN of the folder from a PFN, contained in the ‘dir’ field. However, it would not be advisable to identify a PFN with an LFN exclusively on the basis of their possible similarity in name. Therefore, a solid conversion between PFNs and LFNs is needed.

Such conversion can be obtained from the PhEDEx web APIs [7]. It was chosen to take as reference the folder at the top of the directory tree in the CMS namespace [9], which is '/store'. As a result, the following query was used:

```
.../datasvc/json/prod/lfn2pfn?node=T*&lfn=/store/&protocol=direct
```

The output, say 'LFN-to-PFN mapping', looks like:

```
{"phedex":{"mapping":[{"protocol":"direct","custodial":null,"destination":null,"space_token":null,"node":"T1_SITE1","lfn":"/store/","pfn":"/aaa/qqq/store/"},{"protocol":"direct","custodial":null,"destination":null,"space_token":null,"node":"T1_SITE2","lfn":"/store/","pfn":"/bb/ccc/d/store/"}],...}}
```

For example, let us consider the following record R:

```
{"timestamp":1468369801,"name":"T1_SITE1","space":112025084852908,"dir":"/aaa/qqq/store/mc"}
```

From the LFN-to-PFN mapping, we work out that the PFN corresponding to '/store', say '/store' PFN, for the site that R belongs to, is "/aaa/qqq/store/". Comparing it with the 'dir' in R, we deduce that "/aaa/qqq/store/mc" on that site corresponds to the '/store/mc' folder in the CMS namespace, given that under "/aaa/qqq/store/" there are only directories belonging to CMS namespace.

5.3 Quality check on storage records

Given that storage records and LFN-to-PFN mapping come from two separate databases, the latter could be at times inconsistent with the former, in the sense that the PFN from the mapping does not correspond to a real folder in the actual storage. That was in fact the case for a few sites, which were singularly pointed out—please look at the following table for more details. Moreover, it was observed that the 'dir' field string can end with or without a final '/', depending on the client used by the site to create storage records.

Such findings were understood by the experts and will eventually help improve the robustness of the system. In any case, those kinds of inconveniences can be temporarily sorted out including exceptions during the storage records processing.

Inconsistencies found in monitoring data from May, 2016 to July, 2016.

Site	Expected PFN for «/store/»	Actual PFN for «/store/»
T2_US_Florida	/cms/data/store/	/store/
T2_KR_KNU	/pnfs/knu.ac.kr/data/cms/store/	/store/
T1_DE_KIT_Disk	/grid/fzk.de/mounts/pnfs/cms2/disk-only/store/	/pnfs/gridka.de/cms/disk-only/store/
T2_HU_Budapest	/store/	/dpm/kfki.hu/home/cms/phedex/store/
T2_FR_IPHC	/dpm/in2p3.fr/home/cms/phedex/store/	/dpm/in2p3.fr/home/cmsphedex/store/

5.4 Storage record processing

If operations on record fields need to be performed and/or new fields need to be added to them, possibly on the basis of preexistent fields, this is the right step to allow for that. In fact, it will not be possible later—neither in Elasticsearch nor in Kibana nor in Grafana—due to inherent software constraints, apart from very simple operations, mostly numerical expressions.

As an example, a new field—dependent also on the LFN-to-PFN mapping, whose discrepancies, already discussed in section 4.3, were taken into account—was added to each record. The new field, named ‘rlvl’, is a signed integer number representing how many level the ‘dir’ folder is deeper in the directory-tree structure in comparison with the ‘/store’ PFN on the same site.

For instance, in site X, the ‘store’ PFN is `"/aaa/qqq/store/"`, thus its ‘level’ is 3—considering that the level of the root folder ‘/’ is 0 and assuming levels incremental. Considering a record R1 with ‘dir’ equal to `"/aaa/qqq/"`, the ‘rlvl’ field for R1 will be $2-3=-1$. Considering a record R2 with ‘dir’ equal to `"/aaa/qqq/ccc/aaa"`, the ‘rlvl’ field for R2 will be $4-3=1$. The ‘rlvl’ field can turn out to be of considerable convenience when performing queries on the visualisation platforms that will be described later.

A python script was used for such purposes, complete with comments and examples. It can be found online in the DMWMMON GitHub repository [10], under `MonitoringScripts/SpaceMonViz`.

Ultimately, any other useful field that one can think of can be calculated and added to the records just the way it was done in the case above.

5.5 Additional APIs

An alternative to formatting and the processing dumps, which were explained in the previous sections, are additional data service APIs, which would include in themselves the work done in order to format and process properly storage records. It has to be noted though that these APIs depend heavily upon the use case they are intended for and, as use cases are not static entities but evolve dynamically over time, this solution is not as flexible as desired.

In fact, a use case, or a set of use cases, has to be specified prior to writing the correspondent APIs. Moreover, the format and the information retrieved from a certain API, intended for one or a given set of use cases, could not be appropriate to other use cases come up at a later point.

As an example, hereafter is reported a use case that was dealt with and for which a tailored, additional API was written out.

Use case: weekly total site storage.

Issues in the existing APIs: monitoring data is sampled asynchronously whereas ES/Kibana/Grafana do not provide flexible time aggregation; different format needed for ES.

Solution: Provide ES-formatted weekly-synchronized records.

API description:

```

NAME
DMWMMON::Web::API::totstor - Total site storage.
DESCRIPTION
Weekly-sync total site storage. Output formatted for ES.
OPTIONS
node          node name, could be multiple, all(T*), required
---options from DumpSpaceQuery
time_since    former time range, since this time, if not specified, time_since=0
time_until    later time range, until this time
              if not specified, time_until=10000000000
              if both time_since and time_until are not specified,
              the latest record will be selected

OUTPUT
For every week, a series of 'records' (one per site) reporting their total storage with same
timestamps.
EXAMPLE
Input:
...?node=T1_*
Output:
{"index":{"_type":"sync-
weekly","_id":"4d8245c97c93890b0f04118dcbc1c6f0","_index":"test-sync"}}
{"timestamp":1472204564,"name":"T1_AAA","space":56418073}
{"index":{"_type":"sync-
weekly","_id":"4d8245c97c93890b0f04118dcbc1c6f1","_index":"test-sync"}}
{"timestamp":1472204564,"name":"T1_BBB","space":26418073}
{"index":{"_type":"sync-
```

```
weekly", "_id": "4d8245c97c93890b0f04118dcbc1c6f2", "_index": "test-sync" } }  
{ "timestamp": 1472204564, "name": "T1_CCC", "space": 16418073 }  
...
```

As far as the ES format is concerned, as you can see in the example, before the record containing actual data we need a json block that specifies index, type and id of the record. Those are sort of references to access the record once it is loaded in ES in order to update or delete it. Please refer to the ES documentation of the *bulk* [11] function for further information on this topic.

6 Evaluating software

6.1 Populating ES with data

As a result of the previous steps, we produced a file containing records now ready to be loaded in ES. I described in great detail all the steps needed to load that data into ES in the Twiki document [14] I drew up for this project, under the section “Elasticsearch”.

Of all the content of that section, it is important to mention here the ES mapping. The mapping specifies the format of records, including the format of their field, since several data types are available in ES, including: core types, i.e. string, numeric, date and more; geo data types, i.e. geo point and geo shapes, used to locate points and figures on world charts. Here is the mapping used in our case:

```
{  
  "mappings" : {  
    "_default_" : {  
      "properties" : {  
        "timestamp": {"type": "date", "format": "epoch_second"},  
        "name": {"type": "string", "index" : "not_analyzed" },  
        "space": {"type": "long"},  
        "rlvl": {"type": "integer"},  
        "dir": {"type": "string", "index" : "not_analyzed" }  
      }  
    }  
  }  
}
```

The number of records loaded was 2'225'403 records, which belong to the period May 2016 to July 2016.

Actually, more than one method to load data in ES was attempted, although only one—the one that turned out to be the more efficient in terms of time—is described in the Twiki, i.e. the ‘curl’

method. The other that was tried out consists in the use of the function *index* under the class *elasticsearch.Elasticsearch* in the ES python library [15]. A comparison on the basis of the time spent by the process (sys+user), using the ‘time’ command, revealed that the *index* function is far less efficient in terms of time than the ‘curl’ method when loading in ES the same set of data (to load 100’000 records, 1min50s vs 1.3s).

6.2 Creating dashboards in Kibana and Grafana

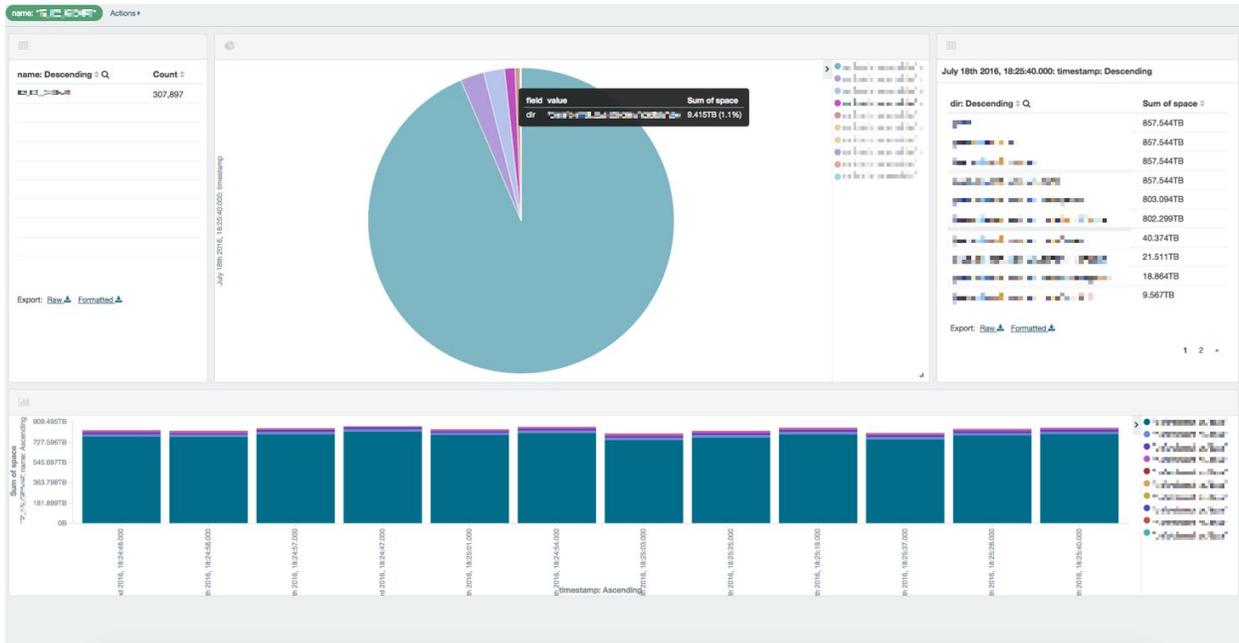
A dashboard is a screen containing multiple visualisations of interest. The procedure to create a dashboard in Kibana or Grafana is slightly different, but the results are similar: in Kibana you first create visualisations one after another and eventually assemble them in a dashboard, whereas in Grafana you first create an empty dashboard and then begin creating visualisations in it.

When making visualisations, it is possible to specify a few options, that vary depending on whether you are using Kibana or Grafana. Please see correspondent section for feature comparison. In principle, those options include:

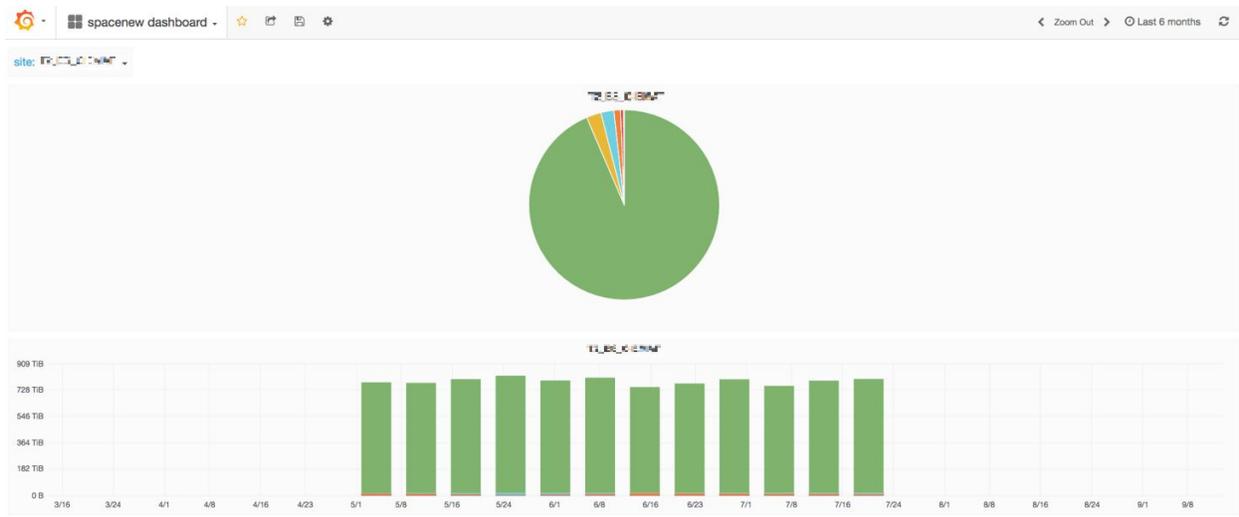
- type of diagram (histogram, time graph, pie chart, table, map and more)
- data filtering and aggregation
- display options: graph style and title, axis measure units, variable labels, etc

In the Twiki, the sections ‘Kibana’ and ‘Grafana’ describe the procedure to create a dashboard, intended as a proof of concept, similar in both the visualisation platforms, making use of the monitoring data loaded in ES.

Let us say we are interested in the folders that take up space on the site storage, but do not belong to the CMS namespace, which means they can be temporary files or belong to local users or etc. Let us assume that there are no such folders at a previous level than the ‘/store’ PFN folder on that site. Then the pie chart in the dashboards show exactly the amount of data taken up by such folders alongside the ‘store’ folder, and their paths come up when hovering over them with the mouse. The time graph in turn shows the same situation over time. Besides, all the folders up to the ‘store’ folder can be shown in a table to make sure the initial hypothesis holds true—as shown in the Kibana dashboard in the following page.



Kibana dashboard



Grafana dashboard

Note: sensitive data have been removed from the dashboards.

6.3 Feature comparison between Kibana and Grafana

Drawing on the knowledge acquired from tutorials, documentation and first hand experience, a feature comparison between the two visualisation platforms was written out.

Feature	Kibana	Grafana
<i>License</i>	Open source	Open source
<i>Documentation</i>	Well documented	Well documented
<i>Data sources</i>	Elasticsearch	Elasticsearch, Graphite, CloudWatch, ...
<i>Access via:</i>	Browser	Browser, http api
<i>Role-Based Access</i>	No. Available for purchase (Shield)	Yes. Many auth options: usr/psw, github oauth, ...
<i>Data discovery</i>	Yes	No
<i>Data search</i>	Lucene query, Query DSL based on Json	Lucene query (for ES data sources)
<i>Charts</i>	Several, including pie charts, histograms, tables	Several, including time graphs and tables
<i>Chart plugins</i>	Yes	Yes
<i>Sharing</i>	(embedded) dashboards	(embedded) dashboards
<i>Import/Export</i>	Yes, Json format	Yes, Json format
<i>More remarks:</i>	Rather intuitive	Less intuitive at first
	General purpose charts	Charts focused on time series
	Possibility to save and reuse intermediate objects—searches, charts,...	Intermediate objects cannot be saved or reused across different dashboards

Let us take a closer look to the major features among the ones listed above.

Data sources. This is major difference since Kibana in meant to work exclusively with ES whereas Grafana can work with several data sources at the same time.

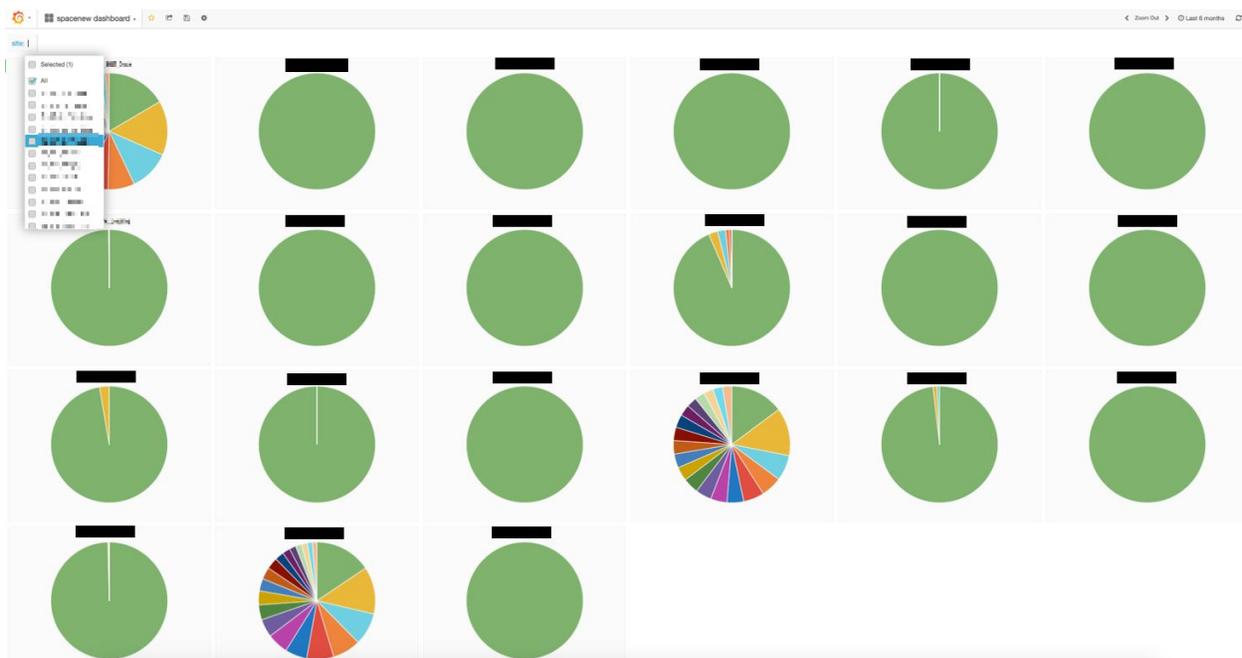
Role-Based Access and sharing. Sharing is done through a link to the dashboard or embedding the dashboard in a web page. That is why authentication plays a key role as regards these tools. In fact, while RBA access is part of the Grafana architecture—you can select who can look at such dashboard, who can edit it and who can manage permissions—, it

is not provided in Kibana, in which everyone who has the permission to access via browser the machine can visualise, edit or even delete any visualisations on it. There is though a purchasable RBA system by the same software company, called Shield [16], that protects Elasticsearch as well as Kibana.

Data discovery and data search. In Kibana you can search through your data without the need to visualize it. In Grafana that is not possible: in order to look at your data you have to create a specific visualization. Moreover, in Kibana you have one more query system at your disposal to search through data, Query DSL based on JSON, which is far more powerful than Lucene Query, the only option in Grafana.

Intermediate objects. As a result of their different workflow described earlier, in Kibana you can save intermediate objects, such as searches and visualisations, and use them as you please in whatever dashboard you want, whereas in Grafana that is not allowed—every visualisation belongs to one and only one dashboard.

Chart repetition. This is another feature that, although not listed in the table, turned out to be of our interest. In Grafana but not in Kibana, you can repeat the same chart for every value that is taken by a certain field throughout the data. In our case, that field was the site field, which specifies what site the storage record belongs to, so as to have the same visualisation, created once, showing the same kind of information for all the sites



Grafana: chart repetition. Sensitive data, including site names, have been removed.

7 Conclusion

As the natural result of the evaluation carried out, here are some final considerations:

- Data processing is limited once the data is loaded in ES, therefore it should be performed prior to feed it the data, if needed.
- The range of use cases that were possible to draw was largely limited by the data service APIs currently available—as an example, in one case it was due to the fact that monitoring data is sampled asynchronously whereas ES/Kibana/Grafana do not provide flexible enough time aggregation. To generally solve that sort of inconvenience, an intermediate step, in which the monitoring data is aggregated and arranged according to the use cases of interest, is needed. That step can be represented by a different data service API or a software tool other than ES/Kibana/Grafana.
- Provided that it is given data in convenient format and aggregation with respect to the visualisation to draw, ES is a powerful engine that stores monitoring data and makes it easily accessible and searchable by Kibana or Grafana. Both the visualisation tools in turn offer a good set of charts to represent monitoring data, along with a few options to customise those charts.

8 References

- [1] CMS Collaboration, The CMS Computing Project TDR, CERN-LHCC-2005-023
- [2] Bonacorsi D *et al.* 2007 The CMS computing model *Nucl. Phys. B (Proc. Suppl.)* **172** 53-56
- [3] Ratnikova N *et al.* 2014 CMS Space Monitoring *J. Phys.: Conf. Ser.* **513** 042036
- [4] Ratnikova N *et al.* 2015 Comprehensive Monitoring for Heterogeneous Geographically Distributed Storage *Journal of Physics: Conference Series* **664** 042055
- [5] DMWMMON web APIs *DMWMMON::Web::Core - fetch, format, and return DMWMMON data*, url omitted due to lab internal policies
- [6] Egeland R, Metson S and Wildish T 2008 "Data transfer infrastructure for CMS data taking" *XII Advanced Computing and Analysis Techniques in Physics Research* (Erice, Italy: Proceedings of Science)
- [7] PhEDEx web APIs, *phedex::Web::Core - fetch, format, and return phedex data*, url omitted due to lab internal policies
- [8] Andreeva J *et al.* 2012 Experiment Dashboard - a generic, scalable solution for monitoring of the LHC computing activities, distributed sites and services *J. Phys. Conf. Ser.* **396** 032093
- [9] CMS LFN Namespace, internal documentation, https://twiki.cern.ch/twiki/bin/view/CMS/DMWMPG_Namespace
- [10] DMWMMON GitHub repository, <https://github.com/dmwm/DMWMMON>
- [11] Elasticsearch website, <https://www.elastic.co/products/elasticsearch>
- [12] Kibana website, <https://www.elastic.co/products/kibana>
- [13] Grafana website, <http://grafana.org/>
- [14] Twiki: SpaceMon visualization, <https://cdcv.sfnal.gov/redmine/projects/cms-spacemon-visualization/wiki>
- [15] Elasticsearch Python Client, <https://elasticsearch-py.readthedocs.io/en/master/>
- [16] Shield website, <https://www.elastic.co/products/shield>

Appendix A: Twiki

The text hereafter is taken from project Twiki page [14]. It is an representative *example* on how to create a dashboard from scratch with CMS monitoring data both in Grafana and Kibana. This guide just describes its technical steps, while the meaning of the visualisations is reported in section 6.2.

Elasticsearch

Elasticsearch version: 2.3.4

1. Retrieve information from the CMS monitoring system

First of all, from the data service we need to get:

- lfn2pfn mapping

```
$ wget --no-check-certificate -O lfn2pfn.js
  "https://cmsweb.cern.ch/phedex/datasvc/json/prod/lfn2pfn?node=T*&lfn=/store/&protocol=direct"
```

- storage records

```
$ wget --no-check-certificate -O dump1
  "https://cmsweb.cern.ch/dmwmmon/datasvc/json/dumpspacequery?node=T*&level=5&time_sinc
  e=1468276170"
```

Note: you can change query options in the urls with regard to your needs.

2. Formatting

We need to give the dump records a proper format.

```
$ cat dump1 | sed "s|{|?{|g" | tr "?" "\n" | sed "s|,|,?{|g" | tr "?" "\n" > dump2

$ egrep '"timestamp"\|"name"\|"space"\|"dir"' dump2 | perl -p -e
  '{s/("timestamp:")"(\d*)",\n/$1$2,/};s/("name":)(.*)\n/$1$2,/};s/("space:")"(\d*)",\n/$1$2,/};s/},/};s/^\\n//;' | sed -e "s/,,$//g" > dump3
```

3. ES mapping

Insert a mapping in ES for the storage records:

```
$ curl -XPUT http://localhost:9200/spacenew -d '
{
  "mappings" : {
    "_default_" : {
      "properties" : {
        "timestamp": {"type": "date", "format": "epoch_second"},
```

```

    "name": {"type": "string", "index" : "not_analyzed" },
    "space": {"type": "long"},
    "r1v1": {"type": "integer"},
    "dir": {"type": "string", "index" : "not_analyzed" }
  }
}
}
}
';

```

4. Loading data into ES

To load the data in ES, we first run the script "proc.py".

```
$ python proc.py lfn2pfn.js dump3 dump4
```

Link: <https://github.com/alesslazzari/DMWMMON/tree/master/MonitoringScripts/SpaceMonViz>

Now load the records in ES.

If "dump4" size is small enough, run:

```
$ curl -XPOST 'localhost:9200/_bulk?pretty' --data-binary @dump4
```

Otherwise, the command above will result in an error. In that case, you have to split "dump4" and load it piece by piece. Example:

```
$ split -d -l 100000 dump4
$ for i in {00..89}; do curl -XPOST 'localhost:9200/_bulk?pretty' --data-binary @x$i; done
```

Check if it was successful:

```
$ curl 'localhost:9200/_cat/indices?v'
```

Kibana

Kibana version: 4.5.3

1. Set up our index pattern in Kibana.

Create a new index "spacenev" in Settings->Indices, selecting "index contains time based events" option and "timestamp" as time-field name.

Still in Settings->Indices, click on the index name on the left, click on the edit button of "space" field and change its visualisation format choosing "Bytes". Click update field.

In Discover section, select your index on the left, then you can use either the Lucene queries or Query DSL based on JSON to discover data, for example respectively:

```

name:T2_GR_Ioannina AND r1v1:0
{"query":{"filtered":{"filter":{"and":[{"regexp":{"dir":"/[^/]*}{5}/store/"}, {"regexp":{"name":"T1_.*"}}]}}, "query":{"match_all":{}}}}}

```

Be always sure to select an appropriate time range in the right top corner.

2. Create "In-out store" pie chart

Click on Visualize->Pie chart->from a new search->spacenew.

Type in the search bar "rlvl:0".

Under Metrics, select Sum as aggregation, space as Field. That states that the amount of space is our metric.

Under buckets, select Split chart (rows), Terms as aggregation, timestamp as Field, order by term, descending order, size 1.

Still under buckets, select add sub-buckets, Split slices, Terms as sub aggregation, dir as Field, order by metric: sum of space, descending order, size 100.

Click on the green "play" icon, then on the "saving" icon and save as "IN_OUT_piechart".

3. Create "In-out store" table

Click on Visualize->Data table->from a new search->spacenew.

Type in the search bar:

```
{"query":{"filtered":{"filter":{"range":{"rlvl":{"lte":0}}},"query":{"match_all":{}}}}
```

Under Metrics, select Sum as aggregation, space as Field.

Under Buckets, select Split table (rows), Terms as aggregation, timestamp as Field, Order by Term, descending order, size 1.

Still under buckets, select add sub-buckets, split rows, terms as sub aggregation, dir as field, order by metric: sum of space, descending order, size 100000.

Save it as "IN_OUT_table".

4. Create "In-out store" history.

Click on Visualize->Vertical bar chart->from a new search->spacenew.

Type in the search bar "rlvl:0", which only selects the folders of interest for our use case.

Under metrics, click select sum as aggregation, space as field.

Under buckets, select split chart (rows), terms as aggregation, name as field, order by term, ascending order, size 1.

Still under buckets, select add sub-buckets, X-Axis, terms as sub aggregation, timestamp as field, order by term, ascending order, size 100000.

Still under buckets, select add sub-buckets, split bars, terms as sub aggregation, dir as field, order by metric: sum of space, descending order, size 100.

Save it.

5. Create site name list.

Click on Visualize->Data table->from a new search->spacenew.

Under metrics, select count as aggregation.

Under buckets, select split rows, terms as aggregation, name as field, order by metric: count, descending order, size 100000.

Click on "play" button. Save it as "site_list".

6. Create instruction widget.

You can also add an instruction widget to ease the use of the final dashboard.

For that, please see the reference here:

<https://www.elastic.co/guide/en/kibana/current/markdown-widget.html>

7. Assemble a dashboard.

Click on Dashboard and add to the dashboard all the visualisation we have created, adjusting them within the screen as you like. Save the dashboard.

You can then share the link to the dashboard.

8. Usage of the dashboard.

First, click on a site from the site list, which will apply a filter to the whole dashboard: you can see a green label at the top left corner.

After that, look at the pie chart: it should represent the space used in percentage by all the folders that are on the same directory tree level of the "store" folder, for the site you chose shortly before, for its last timestamp of storage records.

If you look instead at the "in-out store" table, you can see all the directories up to the "store" level, and verify there is no inappropriate directory that far.

Later if you look at the "in-out store" history graph, you can see the same situation as in the pie chart but for several timestamps, that is the evolution over time.

To check another site, click the remove icon on the green label above the charts and start again from the first step.

Grafana

Grafana version: 3.1.1

1. Add a data source.

You need admin permission to perform the following.

In Data Sources, add data source, with *spacenew* as name for example, Elasticsearch as type, version 2.x, index name *spacenew*, default url, time field name *timestamp*.

Save and test.

2. Dashboard creation.

In dashboards, create a new dashboard called "spaceneu dashboard".

You can edit the name in dashboard settings.

Always remember to choose the desired time range in the dashboard and save the dashboard before leaving it.

3. Templating.

In the dashboard settings, select Templating, add new.

Select "site" as variable, Query as type, spaceneu as datasource, "refresh" "on dashboard load", {"find": "terms", "field": "name"} as query, tick multi-value and "include all option". Press add.

That creates a box which enables you to choose your sites of interest on the dashboard.

4. Create "In-out store" pie chart.

You have to install the pie chart plugin first.

```
$ grafana-cli plugins install grafana-piechart-panel
```

On the dashboard, click Add row, Add panel inside the green button on the right, Pie chart.

Click on the panel title and choose edit. The pie chart options will be:

- in general tab, \$site as title, site as repeat panel, 2 as min span.
- in metrics tab, remove the predefined query, choose spaceneu data source and add query, query "rlvl:0 AND name:\$site", metric: Sum / space, group by: Terms / timestamp / Top / 1 / Term value, then by: Terms / dir / top / 20 / sum space, then by: date histogram / timestamp / auto.
- in options tab, bites as unit, untick show legend.

5. Create "In-out store" history.

In the dashboard, add row, add panel, graph. Edit it. Its options will be:

- in general tab, \$site as title.
- in metrics tab, remove predefined query, add new query on spaceneu data source, group by time interval 1w otherwise 1d, query "rlvl:0 AND name:\$site", metric: sum / space, group by: terms / dir / top / 20 / sum space, then by: date histogram / timestamp / auto.
- in axes tab, bytes unit for leftY.
- in legend tab, untick show legend. You can hover the mouse directly on the graph the see the related information.



- in display section, tick only bars among draw models.

You can also repeat this panel as we did with the previous one.

6. Using the dashboard.

You can choose the sites whose diagrams you want to look at in the box at the right top corner. The pie diagram meaning and the time graph meaning are respectively the same as in the Kibana dashboard.

Note: in the twiki [14] you can also find snapshots of intermediate steps.
