The CMS monitoring applications for LHC Run 3

Brij Kishor Jashal[1], *Valentin* Kuznetsov[2], *Federica* Legger[3,*], and *Ceyhun* Uzunoglu[4] on behalf of the CMS Collaboration

[1]Cornell University, Ithaca (NY), USA
[2]Tata Institute of Fundamental Research, Mumbai, India
[3]Istituto Nazionale di Fisica Nucleare, Torino, Italy
[4]CERN, Geneva, Switzerland

Abstract. Data taking at the Large Hadron Collider (LHC) at CERN restarted in 2022. The CMS experiment relies on a distributed computing infrastructure based on WLCG (Worldwide LHC Computing Grid) to support the LHC Run 3 physics program. The CMS computing infrastructure is highly heterogeneous and relies on a set of centrally provided services, such as distributed workload management and data management, and computing resources hosted at almost 150 sites worldwide. Smooth data taking and processing requires all computing subsystems to be fully operational, and available computing and storage resources need to be continuously monitored. During the long shutdown between LHC Run 2 and Run 3, the CMS monitoring infrastructure has undergone major changes to increase the coverage of monitored applications and services, while becoming more sustainable and easier to operate and maintain. The used technologies are based on open-source solutions, either provided by the CERN IT department through the MONIT infrastructure, or managed by the CMS monitoring team. Monitoring applications for distributed workload management, submission infrastructure based on HTCondor, distributed data management, facilities have been ported from mostly custom-built applications to use common data flow and visualization services. Data are mostly stored in non-SQL databases and storage technologies such as ElasticSearch, VictoriaMetrics, Prometheus, InfluxDB and HDFS, and accessed either via programmatic APIs, Apache Spark or Sqoop jobs, or visualized preferentially using Grafana. Most CMS monitoring applications are deployed on Kubernetes clusters to minimize maintenance operations. In this contribution we present the full stack of CMS monitoring services and show how we leveraged the use of common technologies to cover a variety of monitoring applications and cope with the computing challenges of LHC Run 3.

1 Introduction

After over three years of shutdown due to maintenance and upgrades, the LHC recorded the first beams of its third physics data taking period, also known as Run 3, on July 5^{th}, 2022. The higher collision energy and luminosity will allow the CMS experiment [1] to almost triple the dataset available for physics analysis by the end of Run 3. To store and

*e-mail: federica.legger@cern.ch

process data collected and produced by the LHC, CMS exploits a tiered distributed computing infrastructure comprising more than one hundred computing centers around the world, the Worldwide LHC Computing Grid (WLCG) [2]. The main components of the distributed computing infrastructure are workload management and data management, which are handled centrally together with access and authentication.

CMS compute nodes are provided as execution slots in a Vanilla Universe HTCondor pool [3] and provisioned through GlideinWMS [4]. Job submission to the HTCondor pool is done through specific tools. WMAgent is used for central data processing and Monte Carlo production jobs, and CRAB for user jobs [5]. The data management system includes several components. The data transfer and location services are handled by Rucio [6]. DBS [7] is the Data Bookkeeping Service, a metadata catalog. DAS [8], the Data Aggregation Service, is designed to aggregate views and provide them to users and services. Data from these services are available to CMS collaborators through a web suite of applications known as CMSWEB.

To keep watch on such a complex collection of computing services and tools, CMS has developed over the three years of LHC shutdown a comprehensive suite of monitoring applications. Predefined views allow operators to check the status of the computing systems in real time. Automatic alerts are setup to help operation teams to spot possible issues. Detailed information are available to debug problems. In addition we provide historical views that allow to monitor the system performance over time, for example accounting of storage and data access patterns, usage of computing resources (walltime, memory, CPU).

2 Design of the CMS monitoring services

The CMS monitoring infrastructure acts as a data sink of metrics produced by the various CMS computing systems. The CMS monitoring group is responsible for storing, aggregating and visualising the injected data. Monitoring services have been designed with the idea that they should be easy to use for CMS colleagues, quick to implement and extend for developers, and should need low effort to operate and maintain. To achieve this, we strive to streamline the technologies we use, favouring the adoption of standards and well-supported open-source products. The number of custom applications is kept to a minimum and used to satisfy requirements of specific use cases, which can not be accommodated with the standard infrastructure.

2.1 Data injection

We extensively leverage MONIT [9], the SaaS (Software as a Service) infrastructure provided by the CERN IT department, for data injection, storage, and access. Data is injected in JSON format [10] into MONIT either through ActiveMQ [11] messages or an HTTP endpoint for data providers inside the CERN network boundaries. We provide a Python library for message broker communication to help CMS data providers and to standardise data injection. Logs may be sent also through Logstash [12]. Data are stored in MONIT in OpenSearch [13] and HDFS [14]. OpenSearch is used for short term storage of metrics that need to be accessed for real time monitoring of the computing systems. Detailed information for debugging purposes is also available with a lifetime between one and three months according to the data volume. A reduced set of aggregated metrics is also stored in OpenSearch to provide historical views of system performances. HDFS is used to store raw metrics for longer time periods (one to several years). They can be used for accounting purposes, in-depth studies, and data mining.

2.2 Data access

Data in OpenSearch can be visualised in either Kibana [15] or Grafana [16]. Grafana is the tool of choice for official CMS dashboards, while Kibana is mostly used for quick data exploration. Data in HDFS are accessed through Apache Spark [17] and the SWAN [18] service at CERN. Overall, we support metrics that can come as unstructured, structured, time-series and static data. Access to Grafana data sources can also happen through a Grafana proxy. We developed a set of Command Line Interface (CLI) tools to ease and standardise access for CMS users through the proxy, to manage alerts through queries in either JSON or InfluxQL [19] formats, and to handle annotations on any CMS Grafana dashboard based on its tags. All tools are implemented in Go and are publicly available on CVMFS [20] in the */cvmfs/cms.cern.ch/cmsmonit* area which is available on all grid sites.

The MONIT ecosystem is used by several CMS computing subsystems, such as HTCondor job monitoring data, CMSWEB user activities, metrics about analysis and Monte Carlo (MC) production workflows coming from the WMAgent and CRAB job submission tools. On average, CMS producers send more than 4.5 million messages per hour to the ActiveMQ brokers, with rates close to 6.5 KHz. The total size of CMS data stored in OpenSearch is more than 30 TB, with a daily index average of around 30 GB. In HDFS we collected 45 TB of compressed data over the last five years, mostly stored in JSON format which allows to achieve a 90% compression level. Grafana is the official visualization tool, and we currently manage more than one hundred production dashboards, and a total of more than five hundred.

2.3 The CMS monitoring services

To complement services provided by MONIT, CMS developed its own infrastructure to cover additional monitoring needs. Individual CMS services and nodes are monitored via Prometheus [21] with VictoriaMetrics [22] as a backend. AlertManager [23] is a powerful tool to handle alerts based on Prometheus metrics. All alerts from AlertManager are visualised in karma dashboards [24]. Prometheus, VictoriaMetrics and AlertManager are not provided by MONIT, therefore we manage and operate them in Kubernetes (see Section 3). A detailed description of CMS monitoring applications and services not covered by MONIT can be found in [25].

Heavily populated CMS database, such as those holding DBS or Rucio data, can not be accessed directly by monitoring applications for performance reasons. Daily snapshots are created by a set of Apache Sqoop [26] jobs and stored in HDFS. Several ETL (Extract, Transform, Load) pipelines have been developed to produce aggregated information which require processing metrics over a large period of time (spanning several months or years). Examples of such workflows include the production of popularity datasets by combining data from several sources (DBS, Rucio, EOS [27] and XRootD [28] data access logs) and a comprehensive set of Rucio dataset monitoring applications (see Section 4). We developed and maintain CMSSpark [29], a common framework to transparently access CMS datasets on HDFS using Spark clusters.

3 The Kubernetes infrastructure

Kubernetes [30] allows us to deploy and scale services with minimal operational and maintenance effort. Over the years we gradually migrated most of our services to the Kubernetes infrastructure. The current architecture includes eight clusters (also shown in Fig. 1):

- MAIN: the main CMS monitoring cluster;

- HA-1 and HA-2: two High-Availability (HA) clusters for critical services;
- NATS: a cluster for NATS (Neural Autonomic Transport System) services [31]. NATS is a messaging-oriented middleware service that provides an alternative solution for for message broker and may be used for real-time monitoring of services;
- CRONS: a cluster for ETL batch pipelines (Spark and Sqoop based jobs). Each pipeline runs on a schedule that may be hourly, daily or monthly. To ensure a fair resource usage among all pipelines which may be CPU and memory hungry, jobs need to be properly orchestrated;
- VM-AGG: two instances of VictoriaMetrics for long-term time-series metrics storage for aggregated data;
- DM-MONIT: hosts services dedicated to the data management monitoring described in Section 4;
- TEST: a cluster for testing purposes.

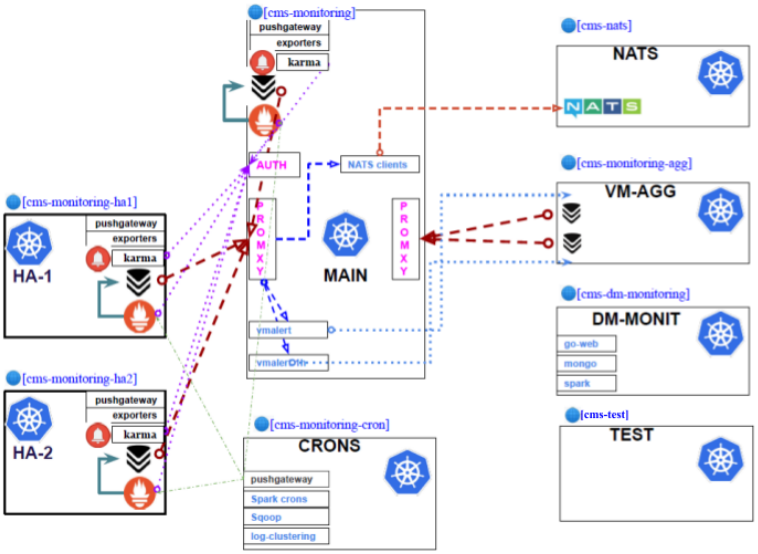

Figure 1. The Kubernetes cluster architecture.

To minimise downtimes, the most critical services (Prometheus, VictoriaMetrics, and AlertManager) are deployed in HA mode, exploiting three clusters (MAIN, HA-1, and HA-2) allocated in different zones in the CERN network. The MAIN cluster runs non-critical services such as the NATS subscribers which listen to various real-time messaging channels from the NATS server, the HTTP exporters used to monitor the status of our services, and a Prometheus Promxy proxy server [32]. The Promxy server is used to access services on both HA-1 and HA-2. Each HA cluster runs independent instances of Prometheus, AlertManager and VictoriaMetrics. The Prometheus server scrapes metrics from a set of CMS services and stores them in the VictoriaMetrics backend. Prometheus is configured to access both AlertManager instances, which can exchange information through a gossip-based mechanism if necessary. The HTTP exporters are also deployed in both HA clusters. In case of outages in HA-1 or HA-2, the Prometheus Promxy server is able to repopulate metrics from the other

one. VictoriaMetrics is used as long-term metric and remote-write storage for our Prometheus instances, as well as for more than ten external Prometheus instances hosted by other CMS groups.

The Prometheus service currently covers more than one hundred computing nodes and more than twenty Kubernetes clusters running hundreds of services overall. We support more than one hundred exporters (mostly written in GoLang and Python), scraping more than three thousand metrics. Since these metrics come from different services in various domains and infrastructures, we access them in pull mode only. AlertManager handles more than 150 alert records and rules. In VictoriaMetrics we store around 500 billion data points with a data retention policy of thirty days. The MAIN cluster fits into two nodes with 16 CPU cores and 30 GB RAM, each HA cluster has one node with 16 CPU cores and 30 GB of RAM and one node with 8 CPU cores and 16 GB of RAM. This architecture is fault-tolerant (no downtimes in three years of operation) and easy to maintain via the Kubernetes deployment procedure. Thousands of metrics scraped by the HTTP exporters are exposed as a single Grafana data source through the Prometheus Promxy server.

4 Dataset monitoring

Monitoring of CMS distributed storage is paramount to ensure efficient use of disk and tape resources allocated to the experiment. Information about stored datasets and their usage can be obtained by combining two data sources: Rucio and DBS. Rucio is the distributed data management of choice for official CMS data, and contains information about dataset names, list of files belonging to each dataset, size, creation date, last access date, policy on the number of copies to keep on different types of storage (tape or disk). Dataset metadata, such as data tier, production campaign, and acquisition era, are stored in DBS. To summarise, DBS knows about each dataset content, whereas Rucio knows its location. A complete monitoring application must integrate information from both data sources. As additional challenge, such application can not directly query the production Rucio and DBS databases, for performance reasons. Custom ETL pipelines were developed to cover this use case. Relevant database tables are extracted on a daily basis, metrics of interest are aggregated, and displayed in a variety of views. A detailed description of the data flows and used technologies can be found in [33].

We currently support three data pipelines (also shown in Fig. 2) to cover various needs:

- *not accessed datasets*: views of least popular datasets based on the last access date metrics;
- *tabular dataset monitoring*: tables summarising for each dataset the content, size, creation and last access dates, and location on the CMS Rucio Storage Elements (RSEs);
- *time-serie dataset monitoring*: time evolution of usage of all CMS RSEs.

The first part of the data flow, namely the extraction and aggregation from the Rucio and DBS databases, is common to the three pipelines. Sqoop jobs are used to create the daily DB snapshots that are stored in HDFS. The Sqoop jobs are optimised and executed in parallel, query about ten DB tables, and typically last less than thirty minutes. Snapshots in HDFS are processed by Spark jobs that aggregate metrics of interest for each specific use case. Each Spark job consists of more than twenty join operations between Spark datasets and tens of aggregations to reach the final data schema.

Storage and access of the final results vary according to the size of the data and visualisation type. Data for the *not accessed datasets* are stored in EOS and CERNBox [34], and visualised with Grafana and DataTables [35] respectively. Data for the *tabular dataset monitoring* are injected in a stand-alone MongoDB [36] instance that serves as a storage to a web service backed by Go and JQuery DataTables stack. The DataTables web page provides advanced

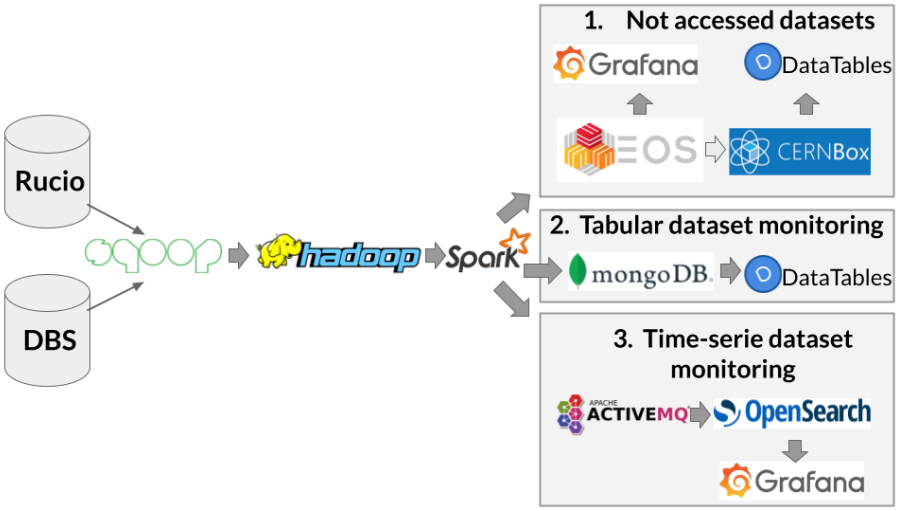

Figure 2. The three ETL pipelines for dataset monitoring: not accessed datasets (1), tabular dataset monitoring (2), and time-serie dataset monitoring (3).

search capabilities over more than 600 thousand dataset entries and close to 6 million RSE replica entries. Time-serie data are injected in OpenSearch through the standard AMQ broker, and visualised with Grafana.

5 Summary

We presented an overview of the current CMS monitoring infrastructure and recent developments, the Kubernetes cluster architecture and the dataset monitoring applications. Services provided by MONIT are complemented with additional applications deployed on Kubernetes clusters which provide better scalability and reduction in operational costs. Such architecture satisfies CMS requirements for monitoring in the challenging environment of LHC Run 3.

We adopted common data formats for metrics and visualisation tools, while keeping the flexibility of using different solutions for specific use cases. A common monitoring infrastructure for CMS applications and the choice of open-source technologies have several advantages: increased portability, lower maintenance costs, and the possibility to share knowledge and developments among several CMS groups. We continuosly develop and add services to the current infrastructure, to improve the CMS user experience of monitoring solutions, and to cope with future challenges at the LHC due to higher data rates.

References

[1] CMS Collaboration, JINST 3 (2008) S08004
[2] I. Bird et al., CERN-LHCC-2014-014 (2014), LCG-TDR-002
[3] D. Thain, T. Tannenbaum, and M. Livny, Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, (2005) 323-356
[4] I. Sfiligoi et al., proceedings of the WRI World Congress on Computer Science and Information Engineering, Vol. 2, (2009) 2428-432
[5] T. Ivanov et al, EPJ Web Conf, (2019) 03006

[6] Barisits, M., Beermann, T., Berghaus, F. et al. Comput Softw Big Sci (2019) 3: 11
[7] V. Kuznetsov et al. J. Phys.: Conf. Ser. 219 (2010) 042043
[8] M. Giffels, Y. Guo, V. Kuznetsov, N. Magini and T. Wildish, J. Phys.: Conf. Ser., Vol. 513, Issue 4 (2014)
[9] A. Aimar, et al., J. Phys.: Conf. Ser. 898 (2017) 092033
[10] *JSON (JavaScript Object Notation)*, https://www.json.org (2023), accessed: 2023-08-18
[11] *Apache ActiveMQ*, http://activemq.apache.org (2023), accessed: 2023-08-18
[12] *Logstash*, https://www.elastic.co/logstash (2023), accessed: 2023-08-18
[13] *OpenSearch*, https://opensearch.org/ (2023), accessed: 2023-08-18
[14] *Apache Hadoop*, http://hadoop.apache.org (2023), accessed: 2023-08-18
[15] *Kibana*, https://www.elastic.co/products/kibana (2023), accessed: 2023-08-18
[16] *Grafana*, http://grafana.org (2023), accessed: 2023-08-18
[17] *Apache Spark*, http://spark.apache.org (2023), accessed: 2023-08-18
[18] D. Piparo, et al., Future Gener. Comput. Syst. 78 (2018) 1071-1078
[19] *Influx Query Language*, https://docs.influxdata.com/influxdb/v1.8/query_language/ (2023), accessed: 2023-08-18
[20] P. Buncic, et al., J. Phys. Conf. Ser., 219 (2010) 042003
[21] *Prometheus*, https://prometheus.io/ (2023), accessed: 2023-08-18
[22] *VictoriaMetrics*, https://victoriametrics.com/ (2023), accessed: 2023-08-18
[23] *Prometheus AlertManager* https://prometheus.io/docs/alerting/alertmanager/ (2023), accessed: 2023-08-18
[24] *karma, Alert dashboard for Prometheus Alertmanager* https://karma-dashboard.io/ (2023), accessed: 2023-11-18
[25] C. Ariza-Porras, V. Kuznetsov, F. Legger, Comput Softw Big Sci (2021) 5:5
[26] *Apache Sqoop*, https://sqoop.apache.org/ (2023), accessed: 2023-08-18
[27] A. J. Peters, et al., J. Phys.: Conf. Ser. (2015) 664 042042
[28] *XRootD project page* http://www.xrootd.org/ (2023), accessed: 2023-08-18
[29] *CMSSpark framework*, https://github.com/dmwm/CMSSpark (2023), accessed: 2023-08-18
[30] *Kubernetes*, https://kubernetes.io/ (2023), accessed: 2023-08-18
[31] *NATS* https://nats.io/ (2023), accessed: 2023-08-18
[32] *Prometheus proxy*, https://github.com/jacksontj/promxy (2023), accessed: 2023-08-18
[33] C. Uzunoglu, CERN note CMS-CR-2023-034 (2023)
[34] H. González Labradoret et al., EPJ Web Conf. 214 (2019) 04038
[35] *DataTables*, https://datatables.net (2023), accessed: 2023-08-18
[36] R. Guo, Commun. ACM 60, 5 (2017) 43–47