Computing and Software for Big Science (2021) 5:5
https://doi.org/10.1007/s41781-020-00051-x

ORIGINAL ARTICLE

The CMS monitoring infrastructure and applications

Christian Ariza-Porras[1] · **Valentin Kuznetsov**[2] · **Federica Legger**[3]

Received: 13 July 2020 / Accepted: 23 December 2020 / Published online: 24 January 2021

Abstract
The globally distributed computing infrastructure required to cope with the multi-petabyte datasets produced by the Compact Muon Solenoid (CMS) experiment at the Large Hadron Collider (LHC) at CERN comprises several subsystems, such as workload management, data management, data transfers, and submission of users' and centrally managed production requests. To guarantee the efficient operation of the whole infrastructure, CMS monitors all subsystems according to their performance and status. Moreover, we track key metrics to evaluate and study the system performance over time. The CMS monitoring architecture allows both real-time and historical monitoring of a variety of data sources. It relies on scalable and open source solutions tailored to satisfy the experiment's monitoring needs. We present the monitoring data flow and software architecture for the CMS distributed computing applications. We discuss the challenges, components, current achievements, and future developments of the CMS monitoring infrastructure.

Keywords Monitoring · Distributed computing · Real-time · Variety

Introduction

Data from the CMS experiment [1] at the LHC are stored and processed in a tiered distributed computing infrastructure. More than one hundred computing centers worldwide, connected through a set of grid services responsible for storage, computing, and connectivity. They are used to process LHC data and produce Monte Carlo simulated events of relevant physics processes. A recent overview of the computing model of the LHC experiments, based on the Worldwide LHC Computing Grid (WLCG), can be found in [2].

The CMS distributed computing infrastructure includes central services and middleware that handle authentication, workload management, and data management. The workload management system executes payloads in compute nodes provisioned through GlideinWMS [3] and made available as execution slots in a Vanilla Universe HTCondor pool [4]. HTCondor jobs are submitted via specific workload management tools: WMAgent for central data processing and Monte Carlo production jobs, and CRAB for user jobs [5]. The data management system is modular and includes several components: PhEDEx, the data transfer and location system; DBS, the Data Bookkeeping Service, a metadata catalog; and DAS, the Data Aggregation Service designed to aggregate views and provide them to users and services [6]. Data from these services are available to CMS collaborators through a web suite of services known as CMSWEB.

Until recently, we monitored most services through custom tools and web applications. The logging information was scattered over several sources and mostly accessible only by experts. The maintenance and operation of the monitoring applications were becoming increasingly time-consuming and complicated due to the high turnover of experts in the computing community.

Nowadays, several solutions are available to gather, store, and process large amounts of data (for example, the data produced by monitoring and logging services of computing applications). Many technologies are available under an open-source license and are developed and supported by large software companies and communities.

✉ Federica Legger
federica.legger@cern.ch

Christian Ariza-Porras
christian.ariza.porras@cern.ch

Valentin Kuznetsov
vkuznet@protonmail.com

1 CERN, Geneva, Switzerland

2 Cornell University, Ithaca, NY, USA

3 Istituto Nazionale di Fisica Nucleare, Via Pietro Giuria 1, Torino, Italy

During the last two years, the CMS computing community has been gradually abandoning in-house solutions in favor of widely used scalable, and non-SQL tools, such as Hadoop [7], InfluxDB [8], and ElasticSearch [9]. These services are available through MONIT [18], a central monitoring infrastructure provided by the CERN IT department, and allow for the easy deployment of monitoring and accounting applications using visualization tools such as Kibana [10], and Grafana [11]. Grafana allows setting alarms and notifications upon certain conditions. We build complex monitoring workflows using different subsystems and perform various predictive analytics studies. To complement the monitoring infrastructure provided by the CERN IT department, the CMS monitoring group set up and runs additional monitoring applications based on technologies such as VictoriaMetrics [12], NATS [13], and Prometheus [14].

In the following sections, we summarise the past and current developments of monitoring solutions for CMS and show why such a technology change was needed and beneficial to the experiment needs. We describe MONIT, the monitoring infrastructure at CERN, the organization of CMS monitoring applications based on MONIT and CMS's infrastructure, and future developments.

Previous work and chosen solutions

Till recently, the monitoring infrastructure of CMS, as well as that of ATLAS [15], another major experiment at the LHC, was based on a combination of several custom solutions. These were independently developed by the various teams providing relevant computing services, such as workflow management, data management, site operations. For instance, the CMSWEB dashboard relied on a custom Python script regularly querying */proc* file systems to provide a snapshot of service activities of the CMSWEB cluster. Metrics showing the status and operations of the CMS Tier-0 [16] were collected by an in-house web application serving through an Apache frontend. ATLAS and CMS used a custom application provided by the WLCG group of the CERN IT department to monitor the time evolution of a set of metrics relative to the jobs executed by their distributed computing systems, and the status of the computing sites (also known as SSB - Site Status Board) [17].

Even though the different monitoring applications satisfied the basic needs of the experiment, the disadvantages of this approach are obvious. There was no common data format (different systems used XML, JSON, CSV) or visualization framework for monitoring metrics. Monitoring data was stored on different virtual machines accessible only to a few developers, and therefore we were unable to aggregate data across various dimensions. The burden of maintaining and operating each monitoring application fell on teams with several other duties. Development of additional features of the monitoring applications, or even their maintenance and operation became increasingly difficult without allocation of additional FTEs (Full-Time Equivalent) for that. For instance, the code base of the data popularity service was abandoned for years and ran on an outdated Linux distribution (SLC6). The end-of-lifetime support of SLC6 at CERN forced us to port it to a newer Linux distribution (CentOS 7) with significant effort. Besides, the performances of the custom Python-based solution scaled poorly with increasing amounts of data, resulting in significant latencies in data visualization and aggregation over extended time ranges.

Several solutions are nowadays available on the open-source market to store, handle, and visualize large amounts of data as those collected by the CMS monitoring applications. A common monitoring infrastructure has several benefits:

- consolidation of the resources needed to operate, maintain, and develop the infrastructure itself and the monitoring applications. We can share knowledge and development efforts among several groups;
- portability of monitoring solutions when using common data formats for metrics, and common visualization tools;
- common storage and access to different metrics, easing the development of in-depth analysis and monitoring applications of the whole computing system;
- better scalability and reduction in operational costs.

The decision to start using a common set of tools and technologies for monitoring was driven by the development of the MONIT infrastructure at CERN. As a result, the custom applications for a job and site monitoring offered by the WLCG group needed to be migrated to MONIT. Within this environment, the experiments, ATLAS and CMS, are responsible for injecting the monitoring metrics into MONIT and set up the necessary visualizations (dashboards). The technologies supported and implemented in MONIT were carefully evaluated by CMS. Clear advantages were recognized in having a common infrastructure and tools that could be used by several computing teams. However not all CMS use cases would be covered. For example, quasi-real-time monitoring of CMS computing services and distributed workflows need a lightweight implementation that does not fit well with the data pipeline implemented in MONIT.

A small team of CMS experts (consisting of less than two FTEs) became responsible for the migration of the WLCG monitoring applications for CMS, for providing support to migrate CMS monitoring applications to MONIT, and for developing and operating the additional monitoring services needed by CMS.

A full description of MONIT is given in Sect. 3, and the integration with CMS monitoring services is presented in Sect. 4. In Sect. 4.2 we discuss CMS specific needs and the implementation of independent monitoring data flows, and their integration with some specific services of the MONIT infrastructure.

The CERN MONIT infrastructure

The CERN IT department offers a variety of monitoring services through the MONIT infrastructure. A typical monitoring workflow consists of the following steps (see also Fig. 1):

- **Data injection:** data is pushed into the MONIT infrastructure through ActiveMQ [19] messaging. As an alternative, data providers inside the CERN network boundaries may use an HTTP endpoint. Log files may be directly injected using Logstash [20].
- **Data transport and processing:** Data is streamed into MONIT using Apache Kafka [21]. Apache Spark [22] and Apache Flume [23] are used for further data processing. Data may be enriched with additional information or data from several sources can be aggregated as needed.
- **Data storage:** Three storage technologies are available, depending on the needs of the particular use case: data volume, schema, and retention policy. ElasticSearch (ES) is used to store semi-structured data for a short period of time (up to one month). InfluxDB is used to store time-series structured, aggregated data for a longer time range (up to five years, depending on the granularity). The Hadoop Distributed File System (HDFS) is used for long-term (currently unlimited in time) storage of data in a variety of formats (Apache Avro [24], Apache Parquet [25], JSON [26], plain text).
- **Data access:** For data access we rely on the stack ES/Kibana, Grafana, and the SWAN [27] service at CERN. Kibana is used for data exploration and visualization of ES data sources. Grafana is a visualization tool that can read data from several sources such as ES, InfluxDB, Prometheus, Graphite [28], Open TSDB [29], and MySQL [30]. Data sources can also be accessed through the Grafana proxy. The Apache Spark framework is used to process data on HDFS. The SWAN service provides access to the CERN Hadoop clusters through a JupyterHub interface [31].

The ActiveMQ servers and the SWAN service are not strictly part of the MONIT infrastructure but are offered and managed by the CERN IT department. The MONIT infrastructure is currently used to monitor the CERN computing center and several WLCG services as data transfers and network [32]. ATLAS exploits the MONIT infrastructure for job accounting and distributed data management monitoring.

The CMS monitoring infrastructure

In this section, we provide an overview of various components of the CMS monitoring infrastructure. Comprehensive monitoring of a complex distributed infrastructure has several requirements. The status of all systems must be monitored in real time using predefined views, and detailed information to debug issues must be provided to the operation teams. Understanding the root cause of a certain problem often means correlating information from different subsystems. An alert system to efficiently collect and sort metrics from several sources, combine information to raise alerts, and route them to the right team must be set up. Relevant metrics should be stored to monitor the evolution of the performances of the systems over time, and be sufficient for in-depth analyses of the main system parameters (data access patterns, wall-time consumption, memory usage).

Having recognized the advantages of using well established open-source products to build a common monitoring

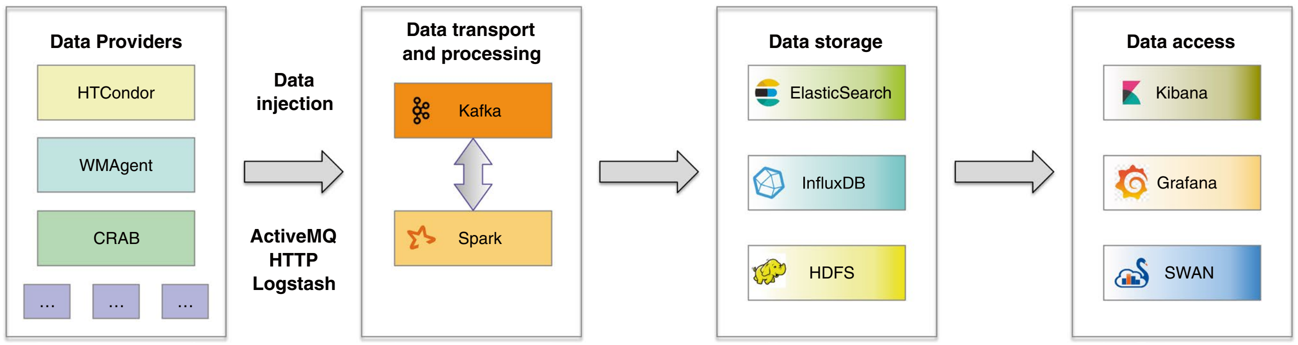


Fig. 1 The data flow in the CERN MONIT infrastructure. The various steps of a typical monitoring workflow (data injection, transport, processing, storage, and access) are shown from left to right

infrastructure to be shared by several monitoring applications in CMS, we aim to leverage as much as possible the services offered by MONIT, and complement them when necessary. We provide a complete set of monitoring tools and applications to the CMS community. In Sect. 4.1 we describe how CMS is building monitoring applications based on the MONIT infrastructure. In Sect. 4.2 we provide details of the CMS specific monitoring infrastructure, and we discuss the CMS monitoring Kubernetes [33] clusters in Sect. 4.3.

CMS usage of the MONIT infrastructure

A variety of CMS data producers regularly injects data into the MONIT infrastructure. There are currently more than twenty-five active CMS monitoring workflows, covering a large variety of systems: HTCondor job parameters, GlideinWMS submission infrastructure, data transfers, and access patterns, CRAB and WMAgent tasks, CMSWEB services. Most of these services used to be monitored through custom web applications and the monitoring metrics were stored using a variety of different technologies. Regarding the data flow outlined in Sect. 3, we describe in the following paragraphs how CMS is using the MONIT infrastructure to deploy its monitoring applications.

Data injection

Most CMS monitoring data producers use the ActiveMQ endpoints—with a few exceptions for producers inside the CERN internal network which use an HTTP REST endpoint—to inject data in JSON format. The choice of ActiveMQ is driven by the fact that it is supported by MONIT for producers inside and outside the CERN network.

The CERN MONIT infrastructure consumes the data via a Kafka pipeline and redirects them to the ES, InfluxDB, and HDFS data sinks. CMS data flows may use any combination of these three storage technologies. We do not impose a specific schema on injected documents, but the schema should be consistent over time. The document schema and daily data volume are defined a priori for each CMS data producer with the CERN MONIT team, which then allocates and controls the resources to consume the data in their infrastructure.

Data storage

The CMS data producers exploit all the provided storage options depending on their particular use case:

- Data is stored in ES for the last 30–40 days depending on the data source. This data is used to monitor the detailed status of the system and provide debugging information in the short term. In addition to the raw data indexes, we take advantage of the document indexing features of ES to create purpose-specific indexes that incorporate some additional logic. For example, we compute and aggregate over HTCondor jobs that have been retried several times, or over temporary job statuses.
- Aggregated structured data as time series are stored in InfluxDB for a limited set of tags and fields. This is used to build historical views of key performance metrics.
- Raw data containing detailed information is stored on HDFS for long term analyses.

Data access

Access to the data stored in MONIT is provided through visualization tools such as Grafana and Kibana for data in InfluxDB and ES, and Apache Spark jobs (either through the SWAN service or standalone scripts) for data in HDFS. Kibana is mainly used for interactive data exploration of the ES data sources. The search capabilities of ES enable our users to quickly create ad-hoc queries and visualizations. Most CMS official dashboards are implemented in Grafana. Additionally, users can create personalized views.

Data stored in HDFS are typically accessed using Spark-based workflows and are used to create dedicated views over long periods (for example, to regularly produce yearly data popularity plots), or for complex queries and visualizations that can not be implemented in Grafana or Kibana.

Experiences with the MONIT infrastructure

In the past two years, several custom CMS monitoring applications have been ported to the MONIT infrastructure. The first tests with the new infrastructure started in 2017, and the migration process formally began in October 2018. Since then the number of data sources, stored data volume, and usage have steadily grown (see Fig. 2).

HTCondor job monitoring is one of the most important views, since it provides an overview of the performances of the CMS distributed computing infrastructure, and was one of the first to be migrated. Data from the HTCondor pool are collected, processed, and injected into MONIT at regular intervals (every twelve minutes) by a service, implemented in Python, called the spider. It is an in-house pre-processing tool that converts the job metrics from the custom CMS specific data formats (HTCondor ClassAds) to JSON documents that can be fed in MONIT. The spider sends data to MONIT through the ActiveMQ endpoint. The HTCondor data is finally stored in ES, InfluxDB, and HDFS.

The complete migration took about one year through various steps: first, an evaluation phase, where all use cases were gathered and requirements for the new application drafted, then the actual development phase, where

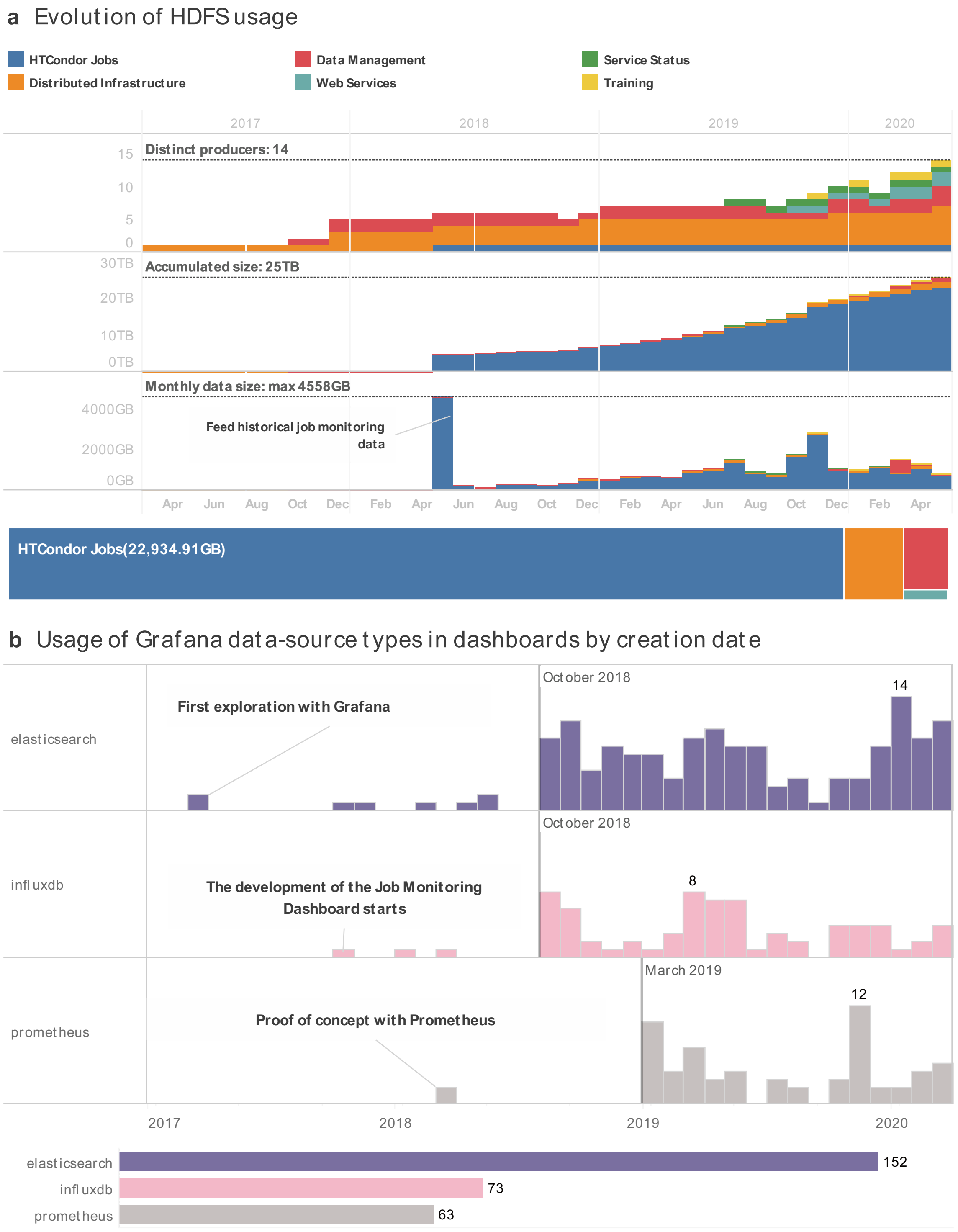


Fig. 2 Time evolution distributions showing the development and adoption of the CMS Monitoring infrastructure. **a** From top to bottom: the evolution of the total number of CMS data producers in HDFS, the amount of stored data, the monthly data volume, and the snapshot (taken in June 2020) of the fraction of storage used by various data sources. **b** The number of Grafana dashboards using ES, InfluxDB, and Prometheus as data sources, ordered by date of creation

the complete data flow was implemented, then a final evaluation phase, where the new data flow was validated against the old one, first from CMS monitoring experts, and then from the wider CMS community. The migration required the coordination and interplay of several teams: the MONIT team, the developers of the CMS monitoring group, and the CMS stakeholders (the data providers and consumers). In particular, it was crucial to define the schema of the data in a way that all user requirements can be satisfied using the visualization options of Grafana or Kibana.

A major problem we encountered was the lack of schema validation in the MONIT infrastructure that may lead to some documents being rejected by ES if the same field has different types in different documents. To avoid this problem, we introduced the schema validation of the injected documents through the CMS client tools, and we are gradually enforcing it for all CMS data providers. We also noticed that some attributes are reserved by ES (such as version, timestamp, UUID), and those should be avoided.

In InfluxDB a limited set of aggregated tags are stored. The aggregations are computed every twelve minutes and stored for one week. Additional aggregations, e.g. 1-h, 1-day, 7-days and 30-days bins are stored for five years. We currently store eighteen tags, with a series cardinality (unique combination of tags values) of almost nine million. The performance of InfluxDB is affected by the cardinality of the stored time series, and that imposes strong limits on the number and type of tags that we can store. The MONIT team is currently evaluating a future retirement of the InfluxDB workflow for this use case, and its replacement with a dedicated ES index to store the time aggregated data.

With the experience gained in the migration of the HTCondor job monitoring application, more custom workflows have been moved to MONIT in the past months: data popularity and data access, and the Site Status Board, a monitoring application that gathers metrics about performances and status of all CMS computing sites. Several analytics workflows have been added to exploit the wealth of data stored in HDFS, which allows in-depth study of several performance metrics, such as the evolution in time of CPU efficiencies for HTCondor jobs, or resource requests for memory, CPU's, storage.

CMS additional components

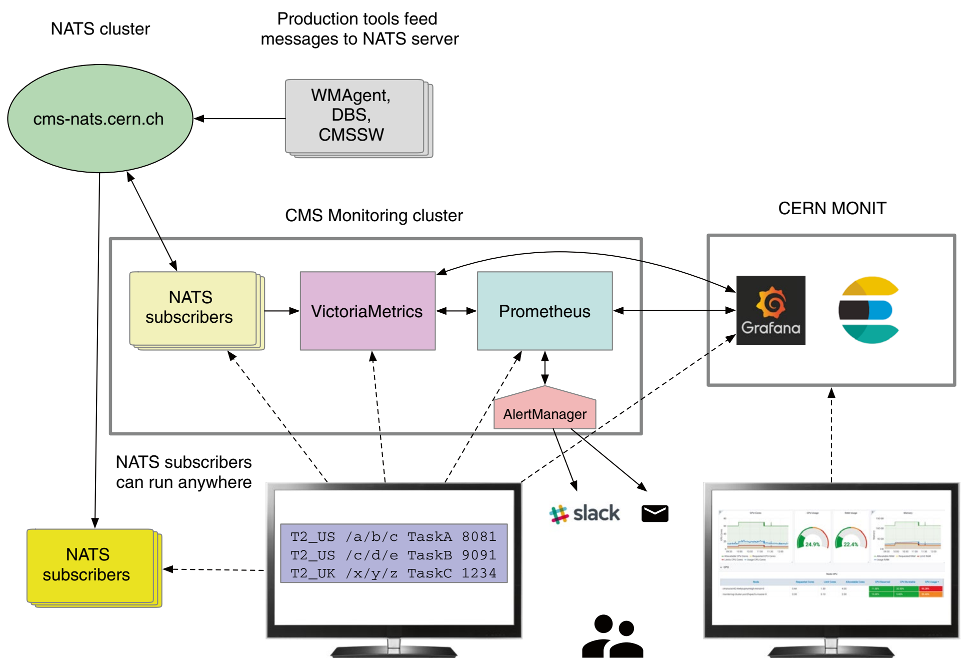
The MONIT infrastructure does not cover all CMS monitoring requirements, for example, the need for quasi-real-time monitoring of CMS computing nodes, complex HTCondor job workflows, applications, and services. To fill this gap, we evaluated several open-source solutions and deployed additional monitoring services based on tools such as Prometheus, VictoriaMetrics and AlertManager [34]. The choice of these tools was based on their wide adoption, performances, and solid reputation in the IT world. They are all written in Go and can be easily integrated in Kubernetes. In comparison to the technologies offered by MONIT, we value the following advantages:

- Prometheus does not require a metrics schema. The service is used to collect a dynamic set of metrics (load, I/O, network usage) from hundreds of CMS nodes and services. Due to the flexibility of the Prometheus ecosystem, we can follow a decentralized approach to fit the constraints of our system (such as authentication boundaries). We run different Prometheus instances on different CMS clusters, but store all the metrics using a common back end (VictoriaMetrics).
- Victoria Metrics outperforms InfluxDB and other time-series databases for data with large cardinality [35].
- The built-in alert mechanism in Grafana is based on raising alerts when certain values are outside of a preset range. The alerts can be configured using a graphical interface. While this approach is intuitive, it is limited to use cases where only a few and simple alert rules are needed. AlertManager allows us to collect alerts from several services: Prometheus, VictoriaMetrics, Grafana, and covers alert unit tests, templates, various routes, regular expressions. Alerts can be handled programmatically in AlertManager, and applying changes to AlertManager alerts can be accomplished with a few Unix shell commands rather than hundreds of clicks on a web interface. Complex alert rules, based on the combination of several conditions, can be easily programmed in AlertManager while achieving the same effect can be cumbersome in Grafana. The details of the alert system are described in Sect. 5.5.

The CMS monitoring infrastructure includes the following components (see also Fig. 3):

- the Prometheus service used to monitor various CMS services and nodes;
- the Logstash service used to parse and send the CMSWEB logs to the MONIT timber service;
- the NATS (Neural Autonomic Transport System) service to provide real-time messaging to monitor the status of CMS production workflows and campaigns;
- Pushgateway, a service provided by Prometheus that allows pushing metrics from jobs which cannot be scraped, such as ephemeral and batch jobs;
- VictoriaMetrics, a fast and efficient time series DB, used as long-term storage for Prometheus and NATS messages;
- the AlertManager to handle alerts for Prometheus metrics.

Prometheus and VictoriaMetrics are the main data sources. We store metrics in Prometheus for fifteen days, and up to one month in VictoriaMetrics. We plan to increase further the time retention policy for data in VictoriaMetrics.



```
T2_US /a/b/c TaskA 8081
T2_US /c/d/e TaskB 9091
T2_UK /x/y/z TaskC 1234
```

Fig. 3 The architectural diagram of the CMS monitoring infrastructure. The MONIT infrastructure (described in Sect. 1) is shown in the right box, and the components maintained and provided by CMS (described in Sect. 4.2) are displayed in the left box

Also we developed and maintain a set of scripts and services for several purposes:

- the spider service [36] described in Section 4.1.4 to collect the HTCondor job parameters and push them into MONIT;
- the CMSSpark framework [37] that provides common access to a variety of CMS data sources on HDFS. CMSSpark is used by several workflows, for example, to collect and aggregate the data used for the data popularity views;
- various CMS databases, such as DBS, are regularly dumped in HDFS by a set of Apache Sqoop [38] jobs.

The maintenance of such a complex infrastructure represents certain challenges, such as service deployment, version control, and resource utilization. We gradually migrated the monitoring services described above to a Kubernetes infrastructure to fully leverage its ease of deployment, dynamic scalability, and minimal maintenance costs.

The CMS monitoring Kubernetes clusters

As shown in Fig. 3, the CMS monitoring infrastructure is based on two distinct Kubernetes clusters: the NATS cluster that contains the NATS service, and the CMS monitoring cluster that hosts various services dedicated to CMS monitoring needs (as described in Sect. 4.2). The Kubernetes clusters are deployed in the CERN Openstack cloud infrastructure [39]. The reasons to set up two different clusters are the following:

- ease of maintenance: the clusters have different capacities and require different node and storage allocations. The NATS cluster runs on nodes with two cores and 4 GB RAM, whereas the Monitoring cluster is made of larger nodes, with four cores per node and 16 GB RAM;
- upgrades and maintenance can be scheduled independently;
- security and networking, we use different authentication schemas in each cluster as well as expose different ports, e.g. the NATS cluster is exposed to the outside world on a dedicated (non-standard) port, while the Monitoring cluster and its services are accessible via standard HTTPs port.

NATS Kubernetes cluster

NATS is a simple, secure, and high-performance open source messaging system for cloud-native applications. Due to its excellent performance benchmarks, lightweight nature, and easily maintainable infrastructure, it fits well our requirements for real-time monitoring applications. It provides a basic publisher subscription model for clients written in a variety of programming languages. The NATS cluster is available outside of the CERN firewall, and accessible to all CMS collaborators and services through token-based authentication. The NATS cluster provides the NATS service, which works as a proxy between CMS data providers, such as CMSSW (the CMS software framework), DBS, and WMAgent, and data subscribers located either on the client infrastructure or within the CMS monitoring cluster. In the latter case, we run a series of dedicated NATS subscribers which consume data from the NATS server and feed them into the VictoriaMetrics back-end.

The NATS Kubernetes cluster consists of two nodes with two CPU cores and 4 GB RAM each, and hosts only the NATS operator and NATS cluster pods. They handle all NATS requests coming from distributed clients. The cluster is currently occupied at 50% of its capacity and can be easily scaled horizontally by adding more nodes in the future.

CMS monitoring Kubernetes cluster

The CMS monitoring Kubernetes cluster is used to monitor our computing nodes, services, and applications. It is deployed in the internal CERN network and is available on a private network for CMS nodes and services via dedicated firewall rules. The cluster hosts both the Prometheus service that consumes metrics from various exporters running on CMS nodes and services and the VictoriaMetrics service as long-term storage back-end for the Prometheus server. Since both Prometheus and VictoriaMetrics data sources are supported by the MONIT infrastructure, we designed Grafana dashboards to monitor our Kubernetes clusters, the CMSWEB Kubernetes cluster, and various CMS nodes, services, and applications.

A dedicated AlertManager service runs within the CMS monitoring cluster. It is configured to set up various alerts based on key metrics of the monitored applications. The AlertManager service is tightly integrated into Prometheus and provides a monitoring service to handle all alerts in our monitoring infrastructures. We discuss the details of the alert notification system in Sect. 5.5.

The CMS Kubernetes monitoring cluster consists of two nodes with 16 CPU cores and 30 GB RAM each and hosts 68 individual applications running within the Kubernetes pods. In total only 15% of the cluster resources are currently used, leaving us room to horizontally scale the provided monitoring services.

CMS monitoring applications

The architectural diagram of the complete monitoring infrastructure, including all components described in Sect. 4, is shown in Fig. 4. In this Section, we discuss various monitoring use cases and applications for CMS based on this architecture. We start with an overview of HTCondor job monitoring in Sect. 5.1. In Sect. 5.2 we provide details of the monitoring applications for CMS services. CMSWEB and Kubernetes monitoring are described in Sect. 5.3. Real-time monitoring is discussed in Sect. 5.4. Alert handling is presented in Sect. 5.5. We conclude with a discussion of various command-line tools experts use for monitoring in Sect. 5.6.

HTCondor job monitoring

Monitoring central and user activities on the CMS distributed computing resources play an important role in a variety of stakeholders (computing experts, management, developer, and operation teams). This monitoring workflow is based on data collected by the spider service described in Sect. 4.1.4. Using the InfluxDB data source we provide a full set of dashboards for CMS collaborators containing several views: the time evolution of jobs in different statuses (completed, running, and pending), utilization of CPU cores, average CPU efficiency, queue and wall-clock times per core and per CPU. All visualization can be further broken down into various categories such as job types, job input types, computing site where the job was executed. A crucial view of the job monitoring application based on the ES data source allows CMS users to monitor the status of their HTCondor

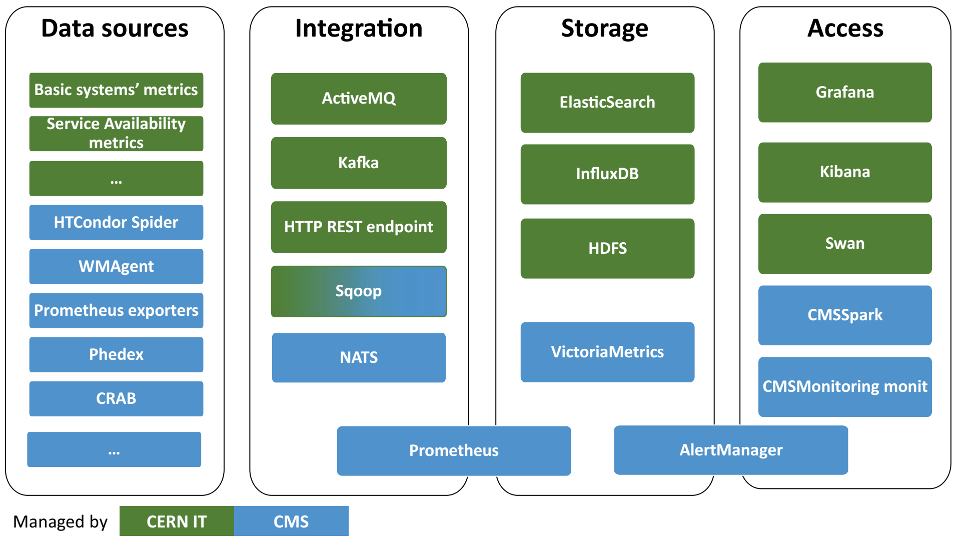


Fig. 4 Architectural diagram depicting the various components used by the CMS monitoring applications. Components managed by the CERN MONIT team are shown in green, while components managed by CMS are shown in blue. The Sqoop module belongs to both infrastructures since it executes CMS workflows on the HDFS storage system provided by the MONIT infrastructure

jobs submitted through CRAB. This is an example where additional support indexes needed to be created to satisfy all user requirements for visualization.

Services and nodes monitoring

Various CMS applications, services, and hardware nodes are monitored using Prometheus, VictoriaMetrics, and AlertManager. Prometheus is a very useful tool to monitor individual services, from a detailed overview of various metrics of a single Linux node to more complex metrics representing service behavior. We use both standard and custom made exporters to scrape service metrics and expose them to our Prometheus servers. The Prometheus service provides a functional query language called PromQL (Prometheus Query Language), that allows users to explore service behaviors. It is integrated with Grafana and supported in the CERN MONIT infrastructure as a data source. VictoriaMetrics provides several enhancements to PromQL and can be used as a fast storage backend for Prometheus. We leverage this functionality to extend the time retention policy for our metrics. Alert Manager provides a useful way to write individual alerts based on nodes and service metrics.

All CMS production systems, including CMSWEB services, individual virtual machines (VMs), and k8 clusters, are monitored through these tools. We provide both service-specific dashboards and overview dashboards representing the status of nodes, services, databases associated with certain activities in CMS.

CMSWEB and Kubernetes monitoring

User activities on the CMSWEB and CMS monitoring Kubernetes clusters are monitored by collecting metrics scraped by Prometheus. System and application logs are collected using the Logstash, Filebeat [40], and ES stack, as shown in Fig. 5. All logs are streamed into the CERN MONIT and CERN security infrastructures. Selected log metrics are injected into ES and then visualized in Grafana. For instance, hourly and daily statistics for services on the CMS distributed computing infrastructure are collected. This information is extremely valuable to track the on-going users' activities and the load on our systems, services, databases and is successfully used to debug various issues with CMS production systems.

In addition, we run independent log scraping and send relevant information, including system and Apache server logs on all production nodes, via Apache Flume [23] streams to the CERN security infrastructure. This data is securely stored within the CERN security infrastructure, in compliance with the CERN privacy policies, and may be used by the CERN security team to monitor potential hacker activities.

Recently many CMS services were migrated to Kubernetes clusters, including our own CMS monitoring infrastructure. Resources on Kubernetes are monitored through metrics scraped by Prometheus and visualized in Grafana by the Kube eagle [41] middleware. The collected metrics show activities on the Kubernetes cluster, and they are integrated

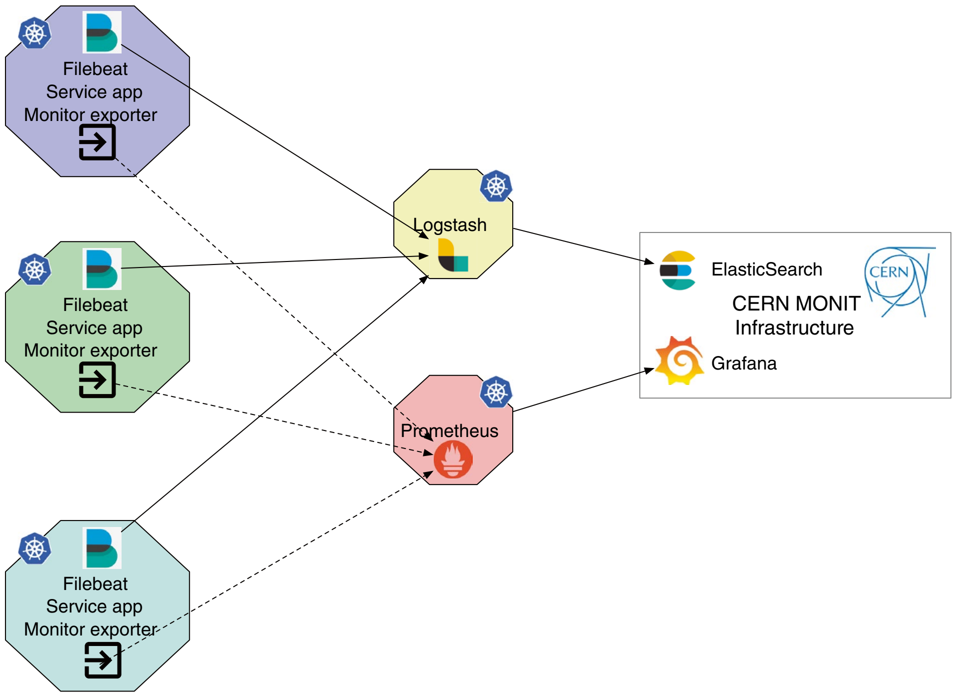


Fig. 5 The data flow for CMSWEB and Kubernetes logs. Logs are processed and streamed by the Filebeat daemon on each individual Kubernetes pod, and sent to the Logstash service that redirects them to ES. The Prometheus service scrapes service metrics via various exporters and is available as a data source in the CERN MONIT infrastructure. In both cases, the information can be visualized with Kibana or Grafana

with the AlertManager workflow so that notifications are issued in case of problems with the monitored cluster.

Real-time monitoring

Real-time monitoring is useful for several CMS use cases, for example, to track failures of workflows executed on the CMS distributed computing resources. CMS users or operators may need to monitor failures at a certain computing center, or for access to certain data. The size of the data produced by specific workflows may need to be monitored. Such requests can be made to specific CMS production databases, such as DAS or DBS. These queries create additional loads to CMS production services, which may have undesirable outcomes such as DoS (Denial of Service). Therefore, we proposed and implemented a (quasi) real-time monitoring service for typical use cases as those above described through NATS (introduced in Sect. 4.2).

We integrated NATS publishers into the CMS production services and collected the required information in the NATS server. The benefit of using the NATS server is that it works as a proxy between data publishers and subscribers, i.e. it does not store the injected metrics, but guarantees their redirection to any number of subscribers. In other words, it works like a one-way chat messaging system where publishers push the message and any subscriber can receive it right away. Since the data volume for message exchange can be high, and quite often happening in a burst, we kept the NATS server separated from the CMS Monitoring cluster, and deployed it into a dedicated Kubernetes cluster, as shown in Fig. 3. This allows us to separate the load between the distributed agents (the publishers) and the CMS subscribers. Several subscribers within the CMS monitoring cluster capture the flow of information and redirect it to the VictoriaMetrics backend so that it can be later used to build Grafana dashboards for a variety of use cases.

Alerts

Alerts play an important role in a constantly growing monitoring infrastructure. Grafana provides simple threshold-based alerts that can be set up for any supported data sources. Metrics scraped by Prometheus and VictoriaMetrics are used to set up more complex rule-based alerts in AlertManager. Alerts can be configured to be sent out to various channels: email notifications, ticketing systems, or communication platforms such as Slack [42].

However, these systems cannot intelligently group alerts, and correlate them with already known system outages. We often found that, while it is desirable to get proper notifications about system misbehavior, human operators cannot cope with a sustained flow of alerts. This typically happens in cases where the same event raises multiple failures and therefore multiple alerts from several systems.

We are currently developing an intelligent alert system that takes into account system outages notifications from several data sources, such as the Site Status Board and several ticketing systems (ServiceNow [43], GGUS [44]), and overlays them with alerts coming from our metrics. For this purpose, we exploit several features of AlertManager: alerts chaining, grouping, and silencing.

Command line tools

While the majority of CMS monitoring use cases are covered by the broad spectrum of visualization capabilities of Grafana and Kibana, for some cases Command Line Interface (CLI) tools are required. For instance, some services need to programmatically access the data sources in the MONIT infrastructure, or some services running behind a firewall need to provide terminal-based monitoring. In this section, we review several CLI tools that we developed in the past months. The degree of adoption in the CMS computing community of such tools strongly depends on their ease of use. We opted to write our tools in the Go programming language since it can provide static executables for all computer architectures supported by CMS.

The following set of tools are distributed through CVMFS [46]:

- **monit**: a tool to query ES and InfluxDB as well as to inject data to ES through a Grafana proxy;
- **NATS**: subscriber and publisher CLI tools;
- **dbs_vm**: a tool to query VictoriaMetrics;
- **grafterm** [45]: a tool to visualize metrics stored in Prometheus or VictoriaMetrics in a terminal-based user interface.

All tools are also available on GitHub [47] and released under the MIT license [48].

Results

We present a summary of various measurements (data volumes, data sizes, usage statistics) related to the various CMS monitoring components described in the previous sections.

Rates of received and sent messages on the ActiveMQ brokers range between 4 KHz and 7.5 KHz. On average, CMS producers send more than 3.5 million messages per hour. Most of the messages are sent by the spider.

We maintain forty data sources in ES, twenty-seven in InfluxDB, eighteen in Prometheus, and collect data in more than twenty HDFS locations. The data in ES are organized in daily indexes corresponding to more than twenty different document types specific to CMS. The total size amounts to more than 20 TB. The data volume on ES is mostly driven by the HTCondor job data, with a daily index size of about 30 GB, while the other data sources have a daily index sizes of 1 GB or less.

On HDFS we currently store around 300 GB per day before compaction. The compaction process takes care of deleting duplicate records and compresses the JSON documents reducing the storage needs for historical data of a factor up to 90%. After compaction, the CMS data sources account for around 32 GB per day. The largest data source is the HTCondor job monitoring metrics that currently has a size of 22.5 TB.

The total data volume in Prometheus and VictoriaMetrics is smaller than that in MONIT but is gradually growing. Currently, we store the 30000 active time series for a total of 15.7 billion data points at an ingestion rate of 4.2 kHz. The total number of entries in the inverted index is 11 million, and the daily time series churn rate is 30000. The total size on disk amounts to 8 GB, with a total index size of 170 MB. The average query range duration is 30 ms. The Prometheus service currently covers more than one hundred nodes with 125 exporters, more than three thousand measurements, and provides almost one hundred different alert records and alert rules.

The CMS computing community built more than three hundred Grafana dashboards using all available data sources. On average CMS users are daily accessing more than thirty different dashboards. More than fifty Grafana alert rules are set up to send alerts to about twenty different channels, while in AlertManager we configured fourteen different receivers. Several Spark-based workflows periodically generate a variety of views to monitor specific aspects, for example, CMS data access and popularity, and HTCondor job metrics such as CPU efficiency and memory consumption.

Due to the unmanaged structure of the previous monitoring systems in CMS, it is not possible to make a direct comparison with the presented metrics. An obvious advantage of the new system is, therefore, the possibility to track all monitoring applications and data sources available to CMS, and being able to measure its performances. This can be taken as a baseline for future improvements.

While the custom applications needed a consistent manpower effort to be developed and maintained, with the common monitoring infrastructure we can leverage components that can be re-used by several groups, for instance, the tools to inject messages into MONIT, or the CLI tools. The experience gained in developing particular features or services can be easily shared. The CMS monitoring infrastructure is currently operated and maintained by less than two FTEs, and that includes also work on future developments and support and training for the CMS user communities.

The versatility of Grafana dashboards allows us to combine several data sources in a single view, and additional customized views can be effortlessly created. In the past, only the standard views were available, and creating new ones or making modifications was a long process that required expert work. On the other hand, custom applications can be more easily tailored to specific needs. For example, Grafana views are designed to give global overviews but are not particularly suited for digging into specific details. This disadvantage can be overcome by a careful design of the data schema, to have a structure that can be easily visualized in Grafana. Nevertheless, it was not always possible to recreate the same ‘look and feel’ of the previous custom applications.

The variety of storage technologies and associated visualization tools allows us to build different kinds of monitoring applications, such as real-time monitoring, accounting, and historical analysis, for the same data source. In the previous system, typically only one use case was implemented for each data source. However, we remark that Grafana and Kibana dashboards are by far the most preferred way to access the data. Metrics stored on HDFS, requiring programmatic access through Spark workflows, are used only by a handful of experts. The possibility to browse HDFS data using a graphical interface is currently part of our R&D efforts.

In general, to fully exploit the power of open-source technologies, we recognize the value of providing specific training to our collaborators, in addition to the publicly available online resources that can be found for open-source products. We, therefore, organize periodic training events with hands-on sessions focused on topics such as data injection and visualization, alert setup, and management. We observe a growing community of CMS users that master access to the various metrics and can build monitoring applications.

Summary

In the era of distributed computing, the monitoring infrastructure plays an important role to ensure the efficient operation of computing nodes, services such as data management and workflow management, computing clusters, and distributed facilities. We presented a global overview of the CMS monitoring infrastructure, which leverages both the CERN MONIT infrastructure and a variety of additional monitoring services and applications deployed on in-house Kubernetes clusters. We discussed architectural choices, lessons, and presented measurements of various performance metrics and statistics.

Based on our experience, the choice of open-source tools is the key to build scalable and maintainable applications in such a complex heterogeneous environment. The various tools discussed in a paper address specific functionalities, from data injection to data visualization, and their choice is driven by the need for ease of maintenance and sustainable evolution of the infrastructure. In particular, we show that the CERN MONIT infrastructure is suitable for pushing monitoring data from distributed data providers, while the stack composed by Prometheus, VictoriaMetrics, and AlertManager covers specific needs of CMS internal data services. The adoption of Kubernetes significantly simplifies the deployment of all the tools and allows to build a scalable infrastructure based on dynamic resource allocation. The chosen visualization solutions, Kibana, and Grafana have been proven to be easy-to-use tools for interactive data exploration and production quality dashboards respectively. In conclusion, we are confident that the current monitoring strategy will fulfill CMS expectations and challenges in the up-coming HL-LHC (High Luminosity LHC) era, where experiments will be required to cope with at least a factor ten higher data rates.

Acknowledgements We thank the CERN IT department, and in particular the MONIT team, for the successful collaboration throughout these years. We also thank our CMS colleagues who contributed to the migration effort and helped us build a solid monitoring infrastructure. This work has received funding from the European Union Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement LHCBIGDATA No. 799062.

References

1. Collaboration CMS (2008) The CMS Experiment at the CERN LHC. JINST 3:S08004
2. Bird I et al (2014) Update of the Computing Models of the WLCG and the LHC Experiments, CERN-LHCC-2014-014, LCG-TDR-002
3. Sfiligoi I et al (2009) The pilot way to grid resources using Glidein WMS. Proc WRI World Congress Comput Sci Inf Eng 2:2428–432
4. Thain D, Tannenbaum T, Livny M (2005) Distributed computing in practice: the condor experience. Concur Comput Pract Exp 17(2–4):323–356
5. Ivanov T et al (2019) Improving efficiency of analysis jobs in CMS. EPJ Web Conf 03006
6. Giffels M, Guo Y, Kuznetsov V, Magini N, Wildish T (2014) The CMS data management system. J Phys Conf Ser Vol 513, Issue 4
7. Apache Hadoop, http://hadoop.apache.org
8. InfluxDB, https://www.influxdata.com/time-series-platform/influxdb/
9. Elasticsearch, http://elastic.co
10. Kibana, https://www.elastic.co/products/kibana
11. Grafana, http://grafana.org
12. VictoriaMetrics, https://victoriametrics.com/
13. NATS https://nats.io/
14. Prometheus, https://prometheus.io/
15. ATLAS Collaboration, The ATLAS Experiment at the CERN Large Hadron Collider, JINST 3 S08003 (2008)
16. Hufnagel D et al (2011) The architecture and operation of the CMS Tier-0. J Phys Conf Ser 331:032017
17. Andreeva J et al (2010) Experiment dashboard for monitoring computing activities of the LHC virtual organizations. J Grid Comp 8:323–339
18. Aimar A et al (2017) Unified monitoring architecture for IT and grid services. J Phys Conf Ser 898:092033. https://doi.org/10.1088/1742-6596/898/9/092033
19. Apache ActiveMQ, http://activemq.apache.org
20. Logstash, https://www.elastic.co/products/logstash
21. Apache Kafka, http://kafka.apache.org
22. Apache Spark, http://spark.apache.org
23. Apache Flume, https://flume.apache.org/
24. Apache Avro, https://avro.apache.org/
25. Apache Parquet, https://parquet.apache.org/
26. JSON (JavaScript Object Notation), https://www.json.org
27. Piparo D et al (2018) SWAN: a service for interactive analysis in the cloud. Fut Gen Comput Syst 78:1071–1078. https://doi.org/10.1016/j.future.2016.11.035
28. Graphite, https://graphiteapp.org/
29. Open TSDB, http://opentsdb.net/
30. MySQL, https://www.mysql.com/
31. Jupyter, https://jupyter.org
32. Aimar A et al (2019) MONIT: monitoring the CERN data centres and the WLCG infrastructure. EPJ Web Conf 214:08031
33. Kubernetes, https://kubernetes.io/
34. Prometheus AlertManager https://prometheus.io/docs/alerting/alertmanager/
35. VictoriaMetrics benchmarks, https://victoriametrics.github.io/Articles.html
36. The spider repository, https://github.com/dmwm/cms-htcondor-es
37. CMSSpark framework, https://github.com/dmwm/CMSSpark
38. Apache Sqoop, https://sqoop.apache.org/
39. Castro León J (2019) Advanced features of the CERN OpenStack Cloud, EPJ Web Conf. 214 07026
40. Filebeat, https://www.elastic.co/beats/filebeat
41. Kube-eagle Prometheus exporter for Kubernetes clusters, https://github.com/cloudworkz/kube-eagle
42. Slack, https://slack.com
43. ServiceNow, https://www.servicenow.com/
44. Antoni T, Buhler W, Dres H, Grein G, Roth M (2008) Global grid user support: building a worldwide distributed user support infrastructure. J Phys Conf Ser 119:052002
45. Grafterm, https://github.com/slok/grafterm/
46. Buncic P, Aguado Sanchez C, Blomer J, Franco L, Harutyunian A, Mato P, Yao Y (2010) CernVM: a virtual software appliance for LHC applications. J Phys Conf Ser 219:042003
47. CMSMonitoring framework, https://github.com/dmwm/CMSMonitoring
48. MIT license, https://opensource.org/licenses/MIT