A Big Data Platform for heterogeneous data collection and analysis in large-scale data centres

Simone Rossi Tisbeni,[*] Daniele Cesini, Barbara Martelli, Arianna Carbone, Claudia Cavallaro, Doina Cristina Duma, Antonio Falabella, Matteo Galletti, Jacopo Gasparetto, Elisabetta Furlan, Diego Michelotto, Francesco Minarini, Lucia Morganti, Elisabetta Ronchieri and Giusy Sergi

INFN-CNAF,
Bologna, Italy

E-mail: simone.rossitisbeni@cnaf.infn.it

The INFN-CNAF data centre hosts the Italian Tier 1 site for the Worldwide LHC Computing Grid (WLCG), while also serving several other research and technological transfer programs. The challenges posed by the upcoming runs of LHC, together with the opportunity of moving the data centre itself to a bigger site, require a thorough redesign of its monitoring system. The large but heterogeneous amount of logging data and metrics produced daily are fundamental for monitoring activities and, once harmonised, can also be used to build Predictive Maintenance models based on Big Data techniques. In this work we describe the Big Data Platform, the new monitoring infrastructure under development at CNAF. The Big Data Platform relies on a modular, highly scalable architecture based on open source technologies and able to exploit modern frameworks such as containerisation and cloud support. It is capable of collecting data from heterogeneous data sources, clean and harmonise them, and store them as JSON files on different solutions, based on the needs of the end user. Data can then be visualised using Kibana, or analysed through a platform based on Jupyter Notebooks.

International Symposium on Grids & Clouds 2021, ISGC2021 22-26 March 2021
Academia Sinica, Taipei, Taiwan (online)

[*]Speaker

1. Introduction

Collection, storage and analysis of large amounts of data are crucial to both industry and research. Just to give an example in the field of fundamental physics, in one year alone (2018) the LHC experiment at CERN generated 88 PB of data worth recording and analysing - and this is just a glimpse on what is expected to happen in the next runs. In the high luminosity phase, or Run 4 (2026-2029), the experiment is estimated to produce up to 500 PB of data per year. Other facilities worldwide face similar challenges. The Open Science Grid (OSG) [1], a consortium of over a hundred computing and storage sites throughout the United States, handles daily the requests of hundreds of users, executing tens of thousands of jobs and transferring several TB of data every day. In these infrastructures it is therefore crucial to ensure the reliability of the service provided, and in-home monitoring and accounting services have been designed to analyse usage and anticipate potential failures. Recently, CERN moved to a new unified monitoring infrastructure, MONIT [2], for its IT Data Centre and the Worldwide LHC Computing Grid. Also the Open Science Grid relies on a new generation GRid ACCounting system, GRACC [3].

CNAF, the INFN facility dedicated to the research and development of information and communication technologies, is gearing up for the upcoming challenges as well. The centre has ongoing technology transfer activities with industrial and medical partners, which will greatly benefit from more powerful, reliable computing infrastructures. Most importantly, as a Tier 1 data centre of CERN Worldwide LHC Computing Grid (WLCG), it will soon need to handle the huge amount of data expected from the next runs of LHC, especially during the high-luminosity phase. It is estimated that about 2000 m^2 of data halls (a factor 2.5 larger than what currently available), with up to 10 MW of power and cooling capacity, will be necessary. A rough preliminary estimate of the resources needed by the end of 2021 is 550 HS06 [4] of HTC computing power, 50 PB of disk and 100 PB of tape storage, with a forecast 20% yearly increase of installed resources. In addition, half of the computing resources are currently located outside CNAF and need to be managed remotely. CNAF has therefore taken the opportunity offered by the new, large area available in Bologna Tecnopolo, a nearby infrastructure dedicated to research and innovation, to move, expand and upgrade its Tier 1 data centre. The transition requires a complete modernisation of the computing infrastructure and, in turn, a redesign of the way in which system administrator operations are currently done, seeking possible uniformity in procedures and tools adopted for all infrastructures (HTC, HPC, Cloud, high security zone, network, storage). The aim is to use the same procedures and tools on all components of the data centre, thus evolving the synergic activities currently in place for provisioning, monitoring and alerting systems.

Over the years the Tier 1 data centre hosted at CNAF has used various monitoring tools [5], all replaced, a few years ago, by a system common to all CNAF functional units [6, 7]. This system, based on Sensu [8], InfluxDB [9], and Grafana [10], is used to acquire, store, and visualise facility metrics (i.e. CPU load, memory usage, IO requests). The new data centre operation will require the inclusion of logging data into the monitoring system and an automation of metrics and logs analysis. This step currently is in an early exploratory phase [11, 12].

In addition, INFN-CNAF coordinates the distributed INFN Cloud infrastructure [13], developing software and distributed systems to be provided as integrated services on the Cloud. Monitoring and accounting of Cloud resources is provided through the use of an infrastructure based on Zabbix [14], Elastic [15] and Grafana, which also monitor the applications and Virtual Machines (VMs) running on the various OpenStack [16] tenants of the Cloud.

INFN-CNAF is also one of the main contributor to the Horizon 2020 IoTwins project [17]. The project provides a reference architecture, for medium and small enterprises, for the development of a platform for the collection of manufacturing and maintenance data and their analysis through the use of machine learning and digital twins simulations [18]. In this project, INFN-CNAF does not only cover the role of cloud provider, but it has also taken a major role in defining the software and technologies solution to be used in the sample architecture.

Given the complexity of the inter-dependencies of the several services running at the data centre and the foreseen large increase of resource usage, CNAF is promoting effort to introduce a more powerful and versatile monitoring system. This platform should be able to harmonise the different infrastructures currently in use at the centre, with the purpose of exploring possible solutions for the development of Predictive Maintenance models to detect and anticipate failures. This new monitoring system should provide a framework able to correlate log files and metrics coming from heterogeneous sources and devices using Big Data analysis tools.

In this paper, we discuss the design principles of this new Big Data Platform (BDP). In particular, in Section 2, we discuss the details of the components and the functionalities provided by the architecture. In Section 3 we present a sample use case that make use of the platform functionalities. Section 4 closes the work touching on the next steps of development.

2. Architecture

The functional diagram of the Big Data Platform is shown in Figure 1. The platform is based on a layered architecture with dedicated components for the various tasks in the data flow:

- **ingestion layer** for the acquisition of the monitored data from different data sources, their validation and formatting in a structure readable by the higher levels of the platform;
- **transfer layer** for the collection and aggregation of data from the producers, their buffering on a reliable and resilient cluster, and their forward to the different storage solutions;
- **storage layer** to store the data on suitable storage solutions according to the different retention policies, security, and technology requirements;
- **access layer** to expose the data collected to the final user for discovery, visualisation, reporting and analysis.

The modularity of the architecture allows to decouple the production and consumer side of the platform. This approach allows for different distribution systems for the different layers. The production side, being system-specific, can for example be installed on bare metal with resources scaled to the requirements of the producers; the consumer side can be provided on-demand with scalable resources on a cloud-based environment.

The components chosen for the Big Data Platform are briefly described in the following sections. They have been selected to satisfy the requirements of using only mainstream solutions, with open source technologies and frequent updates, and able to exploit modern frameworks such as containerisation and cloud support.

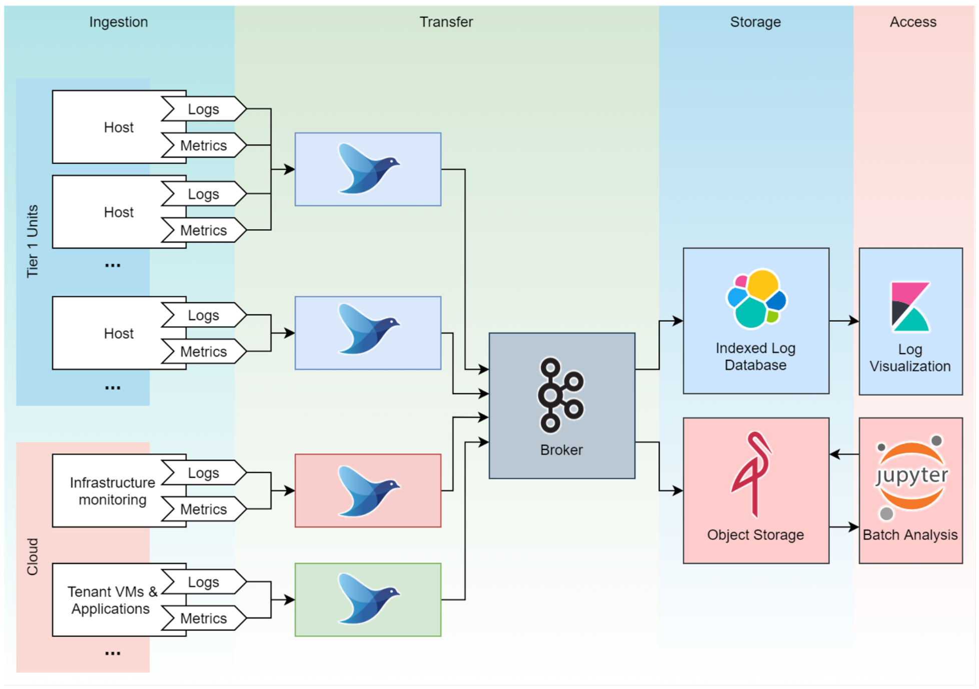

Figure 1: Diagram of the Big Data Platform architecture.

2.1 Ingestion

The bottom layer of the infrastructure consists of data ingestion tools that ingest data produced by hosted services and forward them to the higher levels of the platform. Services produce data of various types and transfer them using several protocols. For this reason dedicated tools are needed to extract the data.

For log files, among the available forwarding tools, we select Filebeat [19], a lightweight log shipper from the Beats family of software in the Elastic Stack suite. Filebeat allows to ingest data coming from log files with variable structure, supporting multiline logs. For each input location specified in the configuration, Filebeat starts a harvester that reads each file, line by line, and sends the content to the configured output. The state of each file is frequently flushed to disk in a registry file. This state is used to restore the last offset a harvester was reading from, to ensure high reliability in case the output is not reachable or Filebeat is restarted. In addition to reference to the harvester offset, Filebeat stores in the registry the delivery state of each log line, ensuring at-least-once delivery to the configured output.

The use of a dedicated shipper to send the log data to the Big Data Platform allows to reserve Rsyslog [20] for critical services already in production at the data centre, without altering preexisting configurations.

Data are sent to the transfer layer as JSON objects. This format allows for flexible data formats, maintaining the original data structure and content. The only requirement is that the ingestion tools provide each event with a predefined subset of fields that contains the information required to perform the correct shipping of the data (i.e. the producer name, an identifier for the data, the topic for the broker).

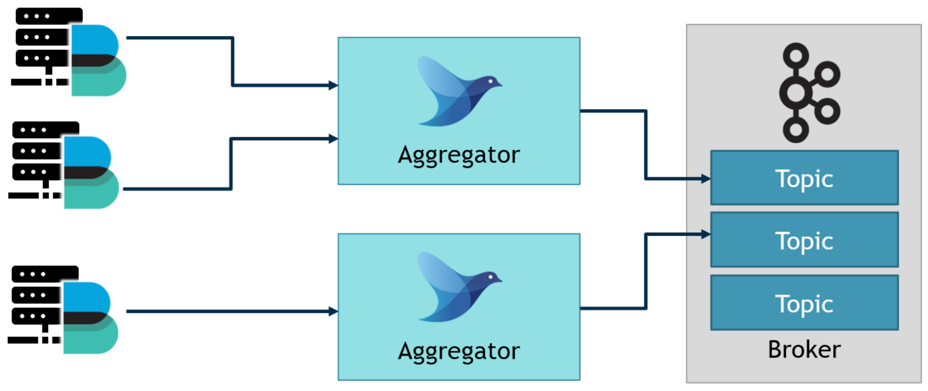

Figure 2: Diagram of the ingestion and transfer layers of the platform.

2.2 Transfer

The ingestion tools forward the data collected towards an aggregator. One or multiple aggregators can be setup for different services or projects, allowing for independent management of the data by the owner. Different tools can be used based on the different requirements of the data owner, granted that the tool selected supports the data format and is able to forward data to the chosen data distribution system. Figure 2 shows a schematic representation of the data transfer pipeline.

In our implementation, the selected tool is Fluentd [21], an open source data collector that allows to unify and filter the data and outputs them to multiple destinations. Fluentd is JSON native and can forward the data without altering their original structure, providing at the same time high flexibility for refactoring and filtering the incoming data. It has a flexible plugin system that allows it to support multiple sources and output destinations. Many of these plugins are included in its stable distribution package. Fluentd supports memory- and file-based buffering to prevent data loss and ensure at-least-once delivery. It can also be set up for high availability when used both for ingestion and aggregation.

The data collected are then distributed to the processing nodes at the higher level of the platform by the topic-based publish-subscribe engine Kafka [22]. The aggregator can publish data coming from different producers to different topics in the Kafka engine. This allows for the separation of data, protecting the users from being affected by each other activities, while ensuring the reliability provided by a common, reliable infrastructure.

The role of Apache Kafka is to decouple the data ingestion tool from the consumers, allowing to broadcast data to multiple subscribers simultaneously. Since Kafka represents a fundamental building block of the platform, it is important to ensure its fault tolerance and high performance. Both have been achieved through a set of three brokers with two replicas for each of the three partitions of a topic. The data in Kafka are distributed to the storage component and buffered for a retention period of three days in order to prevent a data loss in case of network loss with the consumer. The trade-off of this approach is the need of larger storage space.

The deployment of the tools for data ingestion and transfer can be automated via Puppet [23, 24], the provisioning system available on all the equipment of the CNAF data centre.

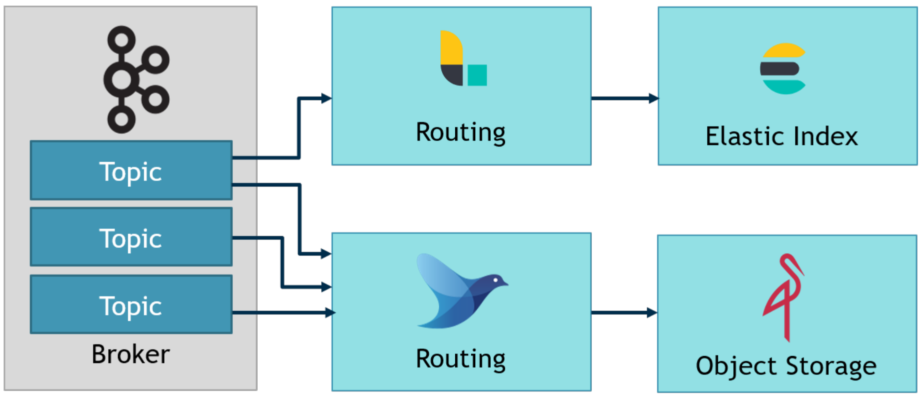

Figure 3: Diagram of the consumer side of the platform.

2.3 Storage

The data received by the Kafka broker are then routed to the storage endpoints. Their format and granularity depend on the producer use case. In this first setup of the Big Data Platform, two storage solutions have been provided: the first is based on the Elastic Stack and provides an indexed database for short term storage of log data; the second is based on the MinIO Object Storage system [25] and provides long term storage for the data. Figure 3 shows a diagram of the consumer side of the architecture.

The Elastic Stack consists of three services: Logstash, Elasticsearch and Kibana. Logstash is a log-ingestion tool capable of aggregating large quantities of logs, filtering them to extract significant information and patterns in dedicated fields, and converge them into Elasticsearch indexes. It is an highly scalable tool that can be configured to handle different data sources with a single configuration file. In our architecture Logstash aggregates logs from Kafka topics and filters them extracting relevant information from each log line. This parsing process transforms logs into structured data, preparing them for more powerful analysis.

The data processed by Logstash are forwarded to Elasticsearch, the central component of the Elastic Stack, which features a distributed search and analytics engine. Elasticsearch stores the data as JSON documents and indexes them in a way that supports fast searches. The software is a distributed, multi-tenant capable, full-text search engine built in Java on the Apache Lucene library [26] and its functionalities are natively exposed as CRUD (Create, Read, Update, Delete) operations via a simple RESTful API over HTTP. The data processed by Logstash can also be shipped back to Kafka to expose the parsed logs to the rest of the Platform.

In our setup, Elasticsearch stores the log data for two weeks and exposes them for discovery and visualisation to the third software component of the Stack, Kibana, which will be discussed in the next section.

The data buffered by Kafka are also saved for long-term storage through MinIO. MinIO is a highly scalable, cloud-native platform that supports modern orchestration and containerisation. The data are stored as JSON objects to maintain the original structure and data fields. Fluentd is used to redirect the messages from the different Kafka topics to dedicated storage buckets, ensuring the data separation required by the data producers. The data stored on MinIO nodes can be accessed in

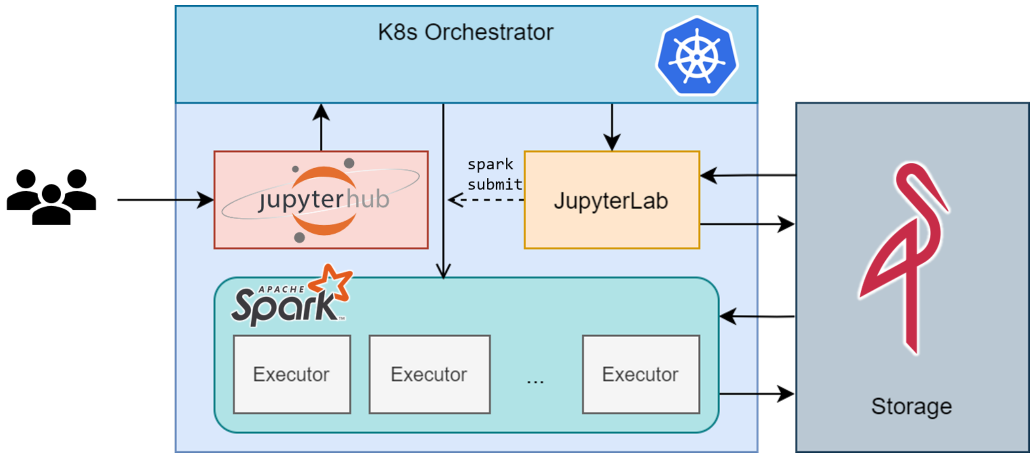

Figure 4: Diagram of the batch analysis pipeline.

multiple ways: web-based UI, SQL-like query languages, or Amazon S3 API, the de facto standard in the object storage world. MinIO can be integrated with a vast number of services, from identity providers to modern data processing service, which will be discussed in the following sections.

2.4 Access

The data made available on the different storage backends can then be accessed using well-known visualisation and data analysis technologies. These solutions provide the end user with easy-to-use interfaces to extract useful insights form the data without working directly with source systems, different transfer protocols, or data format.

The log data stored in the Elasticsearch indexed database can be explored using Kibana, the third component of the Elastic Stack. Kibana is a GUI-based data visualisation service that accesses Elasticsearch indexes. It performs advanced filtering and leverages stored data to build dashboards with use-case driven visualisations. These dashboards allow users to monitor the status of services by looking at selected metrics among the huge amount of data produced in the centre. Kibana also provides several additional features, such as developer tools for advanced interactions with the Elastic Stack and quick discovery tools that make use of Kibana SQL-like syntax for filtering Elasticsearch data and perform advanced queries on log data fields.

In order to perform batch analysis, predictive maintenance and anomaly detection on the data stored in MinIO as JSON files, we build a platform based on Jupyter Notebooks. This setup allows to manipulate the large amount of stored data using common data analysis frameworks, mainly written in Python, such as NumPy, Pandas, Scikit-Learn and Natural Language Processing (NLP) toolkits. In addition, we use Apache Spark [27] to perform the analysis in parallel on several computing cores. We take advantage of the user-friendly paradigm provided by Jupyter Notebooks [28], because it allows for block code execution, in-text graphic visualisations and text cells. These features are directly available on a web application that the user accesses with no need of further setup. We allow for multiple users and for each one we create on-demand a sandboxed environment managed by JupyterHub, deployed on a Kubernetes [29] cluster. The architecture is shown in Figure 4.

Kubernetes is a state-of-the-art, open source container orchestrator system. It provides high scalability of applications and resilience against failure of cluster nodes. Applications are deployed by the Kubernetes master node through pods which consist in virtual hosts, each running one or more Docker [30] containers. Pods communicate through a virtual network managed by the master node. The deployment of the applications is realised through Helm [31] charts. Helm provides a handy package management based on YAML [32] files, allowing for reproducibility and portability. Within this framework we deploy two applications: a NGINX [33] Ingress Controller (not shown in figure) and JupyterHub. NGINX provides an entry point to the cluster and handles the incoming network traffic. JupyterHub is a multi-user server which spawns and manages multiple single-user Jupyter Notebook servers. It can be easily integrated with Kubernetes through the KubeSpawner [34] plugin.

As shown in Figure 4, the user connects to JupyterHub, which sends a request to Kubernetes; Kubernetes in turn creates a pod containing a single-user Jupyter Notebook server. Details of the Notebook server (number of CPUs, RAM, environment image) can be specified by the user at login. The user is forwarded to a JupyterLab[1] environment and a dedicated MinIO bucket is mounted as a POSIX directory. This directory provides long-term storage for the user files and data. The single-user Jupyter Notebook server is embedded with a fully configured Apache Spark setup. Spark is a data processing framework that provides an interface for handling resilient distributed dataset (RDD) over a cluster. It can interface with various distributed file systems and object storage architectures, among which Amazon S3/MinIO, providing a handy SQL-like syntax to access the stored log files. Spark also uses a MapReduce [35] paradigm to efficiently handle heavily demanding tasks on a very large number of CPU cores. In the Spark environment evoked by JupyterHub, the user can submit Spark jobs to the Kubernetes orchestrator, opening a persistent SparkContext via the PySpark [36] library. When the user creates the SparkContext, it takes the role of Spark Driver and requests Kubernetes to instantiate an on-demand Spark cluster. The number of Spark Executors (worker nodes) and their resources are allocated accordingly to the configuration provided at login.

2.5 Authentication and Authorisation

For batch data analysis, JupyterHub and subsequently MinIO storage are accessed with user-based control policies. Authentication and authorisation are performed using the INDIGO IAM service developed by INFN [37, 38]. The IAM service uses the full OpenIDConnect/OAuth2.0 protocols to identify and authorise users, with OpenID Connect (OIDC) [39] providing an identity layer on top of the authorisation layer defined with the OAuth2.0 protocol [40]. When a registered IAM user successfully logs in to JupyterHub, an access token, specifically a JsonWebToken (JWT), is exchanged between IAM (the authorisation server) and JupyterHub (the client) with the usual OAuth2.0 authorisation code flow. The token includes specific scopes with a number of claims which identify the user. These claims are verified against specific policies set up in the MinIO storage, which define authorisation rights for the user, e.g. the ability to create/update/read/write/list/delete content at specified paths and folders.

As described in the previous section, MinIO is mounted as a POSIX directory on the container filesystem. The mount is provided by the INFN STS-WIRE [41] tool, which is a wrapper of

[1]JupyterLab is the modern version of the Jupyter Notebook interface, already embedded in the single-user server installation, and is set as the default within our setup.

the famous mounting tool Rclone [42]. STS-WIRE handles the Rclone authentication to MinIO forwarding the JWT exposed by JupyterHub at the spawn time of the single user container.

Within MinIO, policies are enforced using the Open Policy Agent [43], which is an open source, general-purpose policy engine used to enforce policies across the stack. Policies are defined in a high-level declarative language called Rego and are populated with claims retrieved from the JWT access token. When the agent receives the policy, it generates policy decisions verifying the rights of the specific user against the claims presented in the policy. Based on such policies, the user usually is allowed all operations within a specific MinIO bucket identified by the user name/surname, while a restricted series of operations are allowed in a scratch bucket where the stored log files are accessed via the Spark data processing service.

The usefulness of such an approach lies in the automatic compartmentalisation of working directories based on the user identity, which is originally defined at login via the Indigo IAM service and then processed through the full stack via the Open Policy Agent.

3. Sample use case

Using the Big Data tools described earlier (Filebeat, Kafka, and the Elastic Stack), we devised a pipeline which aims at integrating the current monitoring system at INFN Tier 1 data centre [6], based on Sensu, InfluxDB and Grafana, with information coming from log files. Such integration stems from the idea of enriching the numerical information one can extract from time series with highly specific metadata that can only be obtained through log parsing and processing. This should lead to quick detection of problematic patterns or irregularities, thorough debugging, and potentially lay the basis for data-driven decision making in a predictive maintenance infrastructure.

At the moment of writing, we only focused on log files from Data Management and Data Transfer Services. More specifically, we concentrated on the StoRM service. StoRM provides a SRM (Storage Resource Manager) solution designed to take advantage of high performing cluster file systems, such as GPFS, as well as leveraging standard POSIX file systems. Moreover, StoRM supports the HTTP verb extension for Web Distributed Authoring and Versioning (WebDAV). StoRM-related log files come, at time of writing, from 8 StoRM WebDAV servers and 7 StoRM endpoints.

After being collected, log lines from such services are sent to the BDP, where they are parsed with specific grok filters in Logstash and then forwarded to the Elasticsearch instance for indexing. Kibana works in strict collaboration with the Elasticsearch instance to provide interactive data discovery and visualisation tools.

Figure 5 provides an example of data visualisation which can be obtained from the BDP, namely the HTTP response status codes of StoRM WebDAV servers dedicated to a specific experiment (ATLAS) over time, with prefix 2 corresponding to successful responses and prefix 4 to client errors. Figure 6 shows instead how requests are shared among the three StoRM WebDAV endpoints dedicated to the experiment (left side) and which methods are actually requested (right).

In general, any valuable information can be easily obtained from the log files and plotted - for example, the number and type of requests, their distribution on the servers, the geographical origin of the requests, the storage areas on which the requests insist and the rate of failures. Thus, the first result of this activity is the possibility of interactive data discovery in support of the everyday

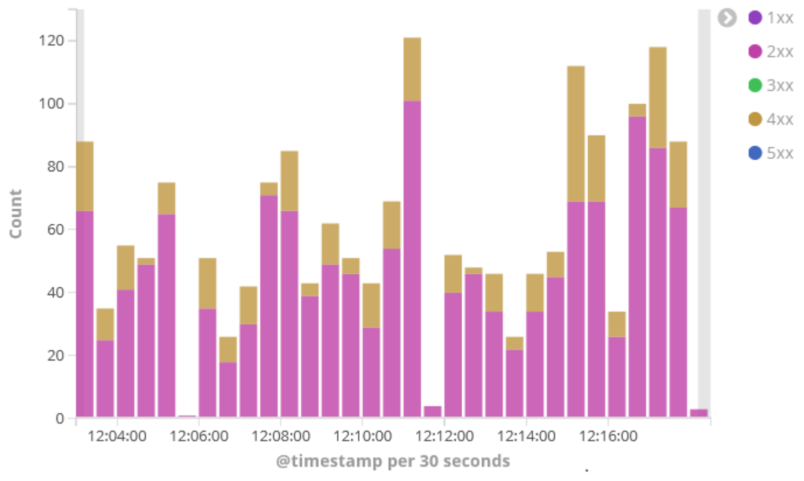

Figure 5: HTTP response codes for the StoRM WebDAV endpoints dedicated to ATLAS in a given time range.

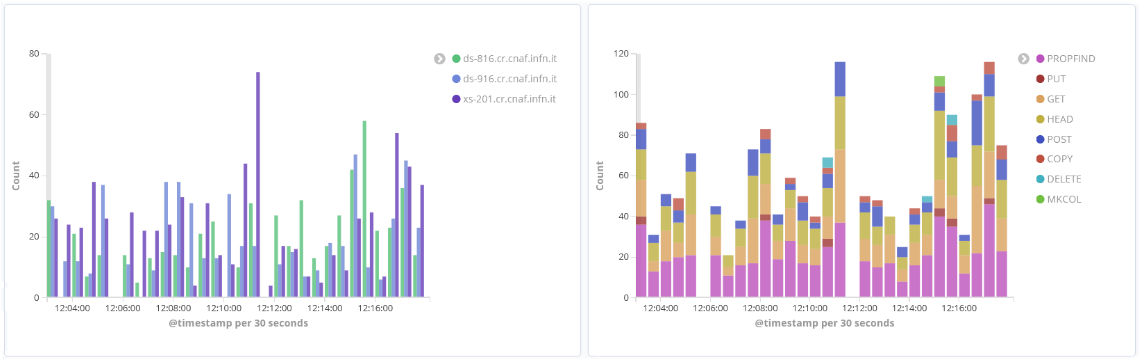

Figure 6: Number of requests per StoRM WebDAV endpoints (left) and methods requested (right).

debugging activity of our system administrators. A second outcome is the visualisation, through monitoring dashboards, of the instantaneous and historical status of the data management and data transfer services that run at CNAF Tier 1.

4. Future developments

In addition to log files, the monitoring metrics collected from the system nodes are also available at the CNAF data centre, and can in principle be used for predictive maintenance and anomaly detection. The current monitoring service is based on Sensu, to manage the gathering of metrics and alerting systems, and InfluxDB, as a timeseries database. To satisfy the requirements of moving towards a unified platform for monitoring all of the data produced at the centre, this critical infrastructure should be merged into the Big Data Platform. InfluxDB will probably remain as the storage solution of choice for metrics and numerical data, with Grafana for dashboard visualisation. The current dataflow from Sensu should be redirected through Kafka to provide a consistent pipeline for the data coming from these sources, allowing for simultaneous routing to different output, including InfluxDB.

In addition, INFN-CNAF is promoting effort to develop an IoT-as-a-service platform to be possibly provided in the cloud services of INFN Cloud. This platform, based on ThingsBoard [44], would allow to collect data from various IoT datasources using different communication protocols (i.e. MQTT, HTTP, CoAP). In addition to storing data into NoSQL Database, Thingsboard is also able to forward messages towards Kafka topics using built-in plugins. With this functionality, Thingsboard is easily embeddable in the current architecture, extending the ingestion functionalities of the platform.

A very challenging use case is related to anomaly detection at the CNAF data centre. Using the log information of the various services running at the data centre, an initial effort will be dedicated to integrate into the BDP the parsing activity of the various entries in the log files. On top of this, feature selection, clustering and classification activities will be performed in order to identify which services and resources (may) cause problems at the data centre. Various approaches have been explored for the analysis. One is natural language processing, that we use to identify message patterns and to discover anomalous system behaviour. This technique can be used to determine e.g. key-anomaly terms to be considered in the clustering or classification phases. Another approach is based on autoencoder techniques for feature extraction. The large amount of log messages makes it difficult to label data for supervised methods. Therefore we will first build an unsupervised model, thus reducing the manual error in defining anomaly categories and labelling entries. The initial analysis will take into account a reference period for both the training and the validation parts; then the whole procedure will be extended injecting new data. An iterative process will be setup to detect anomaly messages and classify them with the help of experts (e.g. system administrators). This will allow to build a semi-supervised model where it will be possible to combine what we achieve by the unsupervised model and expert information.

This type of analysis could benefit greatly from the BDP infrastructure by uniforming the data access, and providing software to do Big Data analysis with high performance. In addition to batch analysis, we are investigating the development of a streaming analysis layer, to elaborate and enrich the data moving through the Kafka broker. A possible solution could make use of the Spark Streaming extension to the core Spark APIs. Data can be ingested from many sources, including Kafka, and can be processed using complex algorithms. Finally, processed data can be pushed out to multiple outputs, including filesystems, dashboards and Kafka itself.

5. Conclusion

In this paper we presented the Big Data Platform, a new monitoring infrastructure at INFN-CNAF. It is a modular and highly scalable architecture built on de facto standard open-source technologies and designed to be able to sustain the INFN-CNAF growth in the next years. It is structured in such a way to decouple all key functionalities. This allows to have multiple, heterogeneous data sources and different storage solutions according to the granularity required by the producer and the usage of the data. All the data are accessible in a seamless way from a common easy-to-use user interface.

Currently this new infrastructure is running parallel to the already existing monitoring systems used in the different projects and services of the data centre, which are critical for production. The main difficulty in the development of the BDP is the adoption of technologies likely to fit all the requirements from these previous systems. To ensure its ubiquitous adoption in the data centre, the BDP must be reliable and should be able to seamlessly integrate in the preexisting infrastructure. For the passage to this new monitoring infrastructure not to be disruptive, its integration will be gradual over the next months.

References

[1] R. Pordes et al., *The Open Science Grid*, *J. Phys. Conf. Ser.* **78** (2007) 012057.

[2] A. Aimar, A. Aguado Corman, P. Andrade, J. Delgado Fernandez, B. Garrido Bear, E. Karavakis et al., *MONIT: Monitoring the CERN Data Centres and the WLCG Infrastructure*, *EPJ Web Conf.* **214** (2019) 08031.

[3] K. Retzke, D. Weitzel, S. Bhat, T. Levshina, B. Bockelman, B. Jayatilaka et al., *GRACC: New Generation of the OSG Accounting*, *J. Phys. Conf. Ser.* **898** (2017) 092044.

[4] HEP-SPEC 06: `https://w3.hepix.org/benchmarking.html`.

[5] S. Antonelli, D. De Girolamo, L. dell'Agnello, D. Gregori, G. Guizzunti, P.P. Ricci et al., *INFN-CNAF monitor and control system*, *Journal of Physics: Conference Series* **331** (2011) 042032.

[6] S. Bovina and D. Michelotto, *The evolution of monitoring system: the INFN-CNAF case study*, *Journal of Physics: Conference Series* **898** (2017) 092029.

[7] S. Dal Pra, A. Falabella, E. Fattibene, G. Cincinelli, M. Magnani, T. De Cristofaro et al., *Evolution of monitoring, accounting and alerting services at INFN-CNAF tier-1*, *EPJ Web of Conferences* **214** (2019) 08033.

[8] Sensu: `https://sensu.io`.

[9] InfluxDB: `https://www.influxdata.com`.

[10] Grafana: `https://grafana.com`.

[11] T. Diotalevi, D. Bonacorsi, A. Falabella, L. Giommi, B. Martelli, D. Michelotto et al., *Collection and harmonization of system logs and prototypal analytics services with the elastic (ELK) suite at the INFN-CNAF computing centre*, in *Proceedings of International Symposium on Grids & Clouds 2019* — `PoS(ISGC2019)027`, Sissa Medialab, Nov., 2019, DOI.

[12] L. Giommi, D. Bonacorsi, T. Diotalevi, L. Rinaldi, L. Morganti, A. Falabella et al., *Towards predictive maintenance with machine learning at the INFN-CNAF computing centre*, in *Proceedings of International Symposium on Grids & Clouds 2019* — `PoS(ISGC2019)003`, Sissa Medialab, Nov., 2019, DOI.

[13] INFN Cloud: https://www-cloud.infn.it.

[14] Zabbix: https://www.zabbix.com.

[15] The Elastic Stack (Elasticsearch, Kibana, Beats, and Logstash): https://www.elastic.co/elastic-stack.

[16] OpenStack: https://www.openstack.org.

[17] IoTwins: https://www.iotwins.eu.

[18] A. Borghesi, G.D. Modica, P. Bellavista, V. Gowtham, A. Willner, D. Nehls et al., *Iotwins: Design and implementation of a platform for the management of digital twins in industrial scenarios*, in *Cloud2Things(2021)*, In Press.

[19] Filebeat: https://www.elastic.co/beats/filebeat.

[20] Rsyslog: https://www.rsyslog.com.

[21] Fluentd: https://www.fluentd.org.

[22] Kafka: https://kafka.apache.org.

[23] S. Bovina, A. Chierici, E. Fattibene, D. Michelotto, G. Misurelli and S. Virgilio, *Cnaf provisioning system*, *INFN-CNAF Annual Report 2015* (2016) 107–110.

[24] S. Bovina, D. Michelotto, S. Virgilio, E. Fattibene, A. Falabella and A. Chierici, *Cnaf provisioning system: on the way to puppet 5*, *INFN-CNAF Annual Report 2017* (2018) 152–153.

[25] MinIO: https://min.io.

[26] Apache Lucene: https://lucene.apache.org.

[27] Apache Spark: https://spark.apache.org.

[28] Jupyter: https://jupyter.org.

[29] Kubernetes: https://kubernetes.io.

[30] Docker: https://www.docker.com.

[31] Helm: https://helm.sh.

[32] YAML: https://yaml.org.

[33] NGINX: https://www.nginx.com.

[34] KubeSpawner: https://github.com/jupyterhub/kubespawner.

[35] J. Dean and S. Ghemawat, *Mapreduce: Simplified data processing on large clusters*, *Commun. ACM* **51** (2008) 107–113.

[36] PySpark: https://pypi.org/project/pyspark.

[37] INDIGO IAM service documentation:
https://indigo-iam.github.io/docs/v/current.

[38] D. Salomoni, I. Campos, L. Gaido, J.M. de Lucas, P. Solagna, J. Gomes et al., *INDIGO-DataCloud: a platform to facilitate seamless access to e-infrastructures*, *Journal of Grid Computing* **16** (2018) 381.

[39] OpenID Connect: https://openid.net/connect.

[40] OAuth 2.0: https://oauth.net/2.

[41] INFN STS-WIRE: https://github.com/dodas-ts/sts-wire.

[42] Rclone: https://rclone.org.

[43] Open Policy Agent: https://www.openpolicyagent.org/docs/latest/.

[44] ThingsBoard: https://thingsboard.io/.