ELK Stack – Improving the Computing Clusters at DFCTI Through Log Analysis

Robert Poenaru  
DFCTI  
NIPNE-HHMagurele, Romania  
robert.poenaru@protonmail.chDragos Ciobanu-Zabet  
DFCTI  
NIPNE-HHMagurele, Romania  
zdragos@nipne.ro

*Abstract*—The full stack logging service provided by ElasticTM has become a powerful tool within the high-performance computing community due to its ease of use, lightweight impact on the machines, performance speeds, and scalability. In the current work, we attempt to deploy such a stack on a server within our department, which will be used for ingesting, parsing, and finally analyzing logs coming from multiple clusters. By analyzing the overall performance of each machine that is under continuous monitoring, we can provide immediate support in case of any issues that might occur, and more importantly, we can improve the computing power of our clusters through optimizations in terms of system management, networking, and other specific features.

Keywords— Elasticsearch, Kibana, Logstash, pipelines, logs, metrics, clusters, compute notes, Kubernetes.

# Introduction

Extending the computational resources in terms of raw compute power, storage space, memory bandwidth, and network capabilities is a top priority for any research institute nowadays. Recently, the Department of Computational Physics and Information Technology at the National Institute of Nuclear Physics and Engineering – Horia Hulubei (situated in Magurele, Romania) is planning a big upgrade in terms of the computing equipment that will be used for system administration, to improve the network infrastructure within the institute center, and also to provide the computational resources required for the research teams and their projects.

With the increase in size of the computing resources comes a challenge in having a proper system administration and making sure that every component is working properly, well within its performance margins. Having a large number of CPUs does not necessarily scale up the performance of a compute cluster without proper optimizations; accessing large amount of data from a storage unit could also result in potential issues if the configuration is not done properly. Problems like these could appear during the early development stages of the infrastructure itself, or even after its final deployment. In order to minimize this kind of problems, it is crucial that a constant real-time analysis and monitorization of the system metrics should be performed by the system administration team. Thus, having a great log management tool, that is capable of ingesting logs from a multitude of sources, convert them to in a human-readable format, analyze them in real-time, and also store them for later use (if necessary) is essential within our department. Fortunately, there is such a toolset available to us: Elasticsearch [1]. The team at Elastic [2] developed a very useful instrument which became very popular over the years, mainly due to the fact that it provides the entire stack for monitoring a computing resource: log shipping, log ingesting, log parsing, log storing and finally analysis – ELK stack [3].

We attempt to build and configure a full Elasticsearch logging service (ELK stack) that will be used within the department for analyzing logs from a multitude of compute clusters. In this way, the team will be able to check the status of the machines that run simulations (or, in other words, compute jobs) and see if there are any issues that require immediate fix. Not only that, but by having an insight on the resource performances with time, one could pinpoint certain patterns preventing issues or even deploy optimizations throughout the system. Analyzing the overall performance for the compute nodes which run large simulations could also be useful in helping the research teams that develop the actual simulations: through allocating the available resources in such a way that great scalability is achieved with the increase in numerical algorithm’s complexity. All these aspects act as a motivation for adopting a service which will be able to provide us a tool capable of improving the computing architecture of the department (DFCTI) and later on for the entire research institute, that is NIPNE-HH.

In the present work, an ELK stack is developed as a testing phase for what will eventually be a proper log analysis service inside DFCTI (and further, to the entire institute). The stack is deployed on a virtual machine from one of the available servers inside the department. In order to see how the log ingesting performs in terms of dropped connections, network lag (which induces a delay between each logging event), or even issues related to the local VM, we sent metrics from different sources, each requiring individual parsing. One of the sources was a Kubernetes (k8s for short) cluster [4] managed by the OpenStack service [5]. This has a major importance, as the cluster will be part of an incoming project for extending the cloud infrastructure that is available at DFCTI [6]. Within the testing phase of the stack, we noticed an overall great performance of the ingesting pipeline and more importantly, the data parsing. The data parsing that is made through Logstash (component of the ELK stack) was properly configured for all the sources (e.g. system metrics from a personal computer, logs from a SLURM workload manager [7] and the k8s compute cluster), and we were able to extract the necessary information from parsed logs, which is planning for fixing and improving the infrastructure. Elasticsearch has been proven to be a great tool for storing the incoming logs, which can be accessed at a later time through the available REST API. Visualization of logs was also successful by using the Kibana UI. The stored logs (in Elasticsearch indices) and the Kibana interface were deployed on the web for remote access via browser.

In this section we gave the motivation behind constructing an ELK stack. In section II, a detailed workflow of the resources which will be monitored is given. Furthermore, in the third section we aim at discussing each service that was used for the project (with a few arguments on the choices we made). **Finish introductory part…**

# Compute Resources & Log Sources

In this section we enumerate all the platforms and systems that were as sources for logs within our testing phase. Methods for the actual shipping of logs and then further ingesting process will be described in a separate section for each source. Description with the workflow involved in parsing and analyzing of SLURM logs will be made in a forthcoming paper, as it is out of the scope of the current work.

## Nova Compute Nodes

As we already mentioned, the ELK stack which we developed was tested for ingesting logs from different sources. The most important one (mainly because of the complexity within the deployment process) was a Kubernetes (k8s) cluster that is deployed on a bare metal managed with OpenStack (through services like Nova [8], Keystone [9-10] and Cinder [11-12]). The cluster has an architecture which can be seen in Figure 1. Although the development and deployment process of the Nova compute cluster is far beyond the scope of the present work, we will mention a few aspects with regards to the services involved, just to get a grasp on the content of the logs themselves.

The Nova project is used to create compute instances (or virtual servers) on top of bare metal servers. It runs as a set of daemons on top of the existing Linux-based servers. Within the OpenStack, Nova is the tool which has to be used when implementing new services on compute servers that require great scalability, on demand access to compute resources (bare metal, virtual machine and also containers [13]). Interaction of a user with the Nova service is done through a REST API interface, but internally, Nova components communicate via an RPC message passing mechanism. In order to create and manage a storage service within OpenStack, the Cinder project is used. It virtualizes the management of block storage devices and also provides an API to the user, which is able to request and consume resources without the proper knowledge of where their storage is actually deployed (Cinder API is also be device-type agnostic). Our OpenStack infrastructure has also implemented client authentication, service discovery and multi-tenant authorization through Keystone service [10] that is part of the OpenStack project.

## System metrics from a personal computer

Another source of incoming logs that we tested was a batch of system metrics from a personal computer (laptop). The motivation behind this is that one could gather all the system metrics from the computers within our department, and perform a constant analysis of their overall performance with regards to the processing power, memory and network latency, and it is possible to detect unwanted attacks on a specific machine via the network. The computer which we tested was a MacBook Pro running MacOS X Catalina (10.15).

## System metrics from Virtual Machines

DigitalOcean [14] is a platform that allows deployment of virtual machines (VMs) that are hosted on their servers (technical specifications for the VM can be chosen during the configuration phase). These virtual machines are called droplets [15] – we used one droplet as a source for system metrics. Testing such a setup is also useful for the project, as the computing resource that we analyzed was hosted on servers outside of the research institute network. Log shipping from external networks could show issues if the ingesting system does not have a proper configuration. Successful log analysis of such resources indicate that our logging implementation is working properly.

# ELK Stack – Development of the Logging System

In the following section, we focus on the implementation of the logging system itself, namely on how we configured the ELK stack, what services we used in order to create a logging pipeline compatible with all the sources mentioned in the previous section.

## Hardware

As an environment for testing the stack, we have a VM running CentOS-7 deployed on a server located in the department. It is configured to accept incoming connections from several ports that will be used for ingesting logs. The VM is also configured with a NGINX [16] web server that allows us to set up reverse proxies (the reason for setting up the reverse proxies will be discussed later on). In terms of specifications, the VM was configured with 8 CPU cores (no hyper-threading), 16GB of RAM and 50GB of storage space. Since this is only for testing procedures, having large storage space does not help improve the performance of our logging pipeline. However, storage space will be a crucial aspect when deploying a final system for monitoring logs.

## Elasticsearch + Logstash + Kibana

The ELK Stack consists of three products developed by the Elastic company, namely Elasticsearch, Logstash, and Kibana. All of these services (together with the other standalone products that Elastic offer) are open source.

#### Elasticsearch – Based on the Apache’s Lucene search engine, it provides a RESTful JSON-based search engine. It also allows storing the data in an efficient way, through a so-called Elasticsearch index: the analogue of a classical (SQL-type) database.

#### Logstash - Tool that allows a dynamical ingesting of data from multiple sources, transformation and parsing of the data through filters, and finally sending the fitlered information to Elasticsearch (Input -> Filter -> Output). It is worth mentioning that Logstash is very flexible in each of the three stages of the processing pipeline; meaning that it can input data from many types of sources, it can process the data and change its structure depeding on the user needs, and also it can output the parsed results to multiple sources (e.g. the output plug-in mechanism can send data to files located on a disk, to a specified e-mail address, generic HTTPS endpoint and so on).

#### Kibana – Acting as the data visualization within the stack, it is a a web-based user interface that allows navigation through the Elasticsearch data. Large volumes of data can be visualized as charsts, diagrams, histograms, tables and so on. The Kibana user interface is accessible through the browser, available on the localhost (with the default port 5601). In order to access it from the web, a reverse proxy with NGINX was used on the VM (a workflow is graphically represented in Fig. 2).

## Filebeat

Filebeat [17], also developed by Elastic, is an open source shipper for log files. It forwards any log file that the user chose within the configuration file to the desired output pipeline (e.g. Logstash). It needs to be installed as an agent (daemon) on the severs which need to be analyzed, then chose the paths to the log files, and then select Logstash as the desired output method. As a mode of operation, when Filebeat agent is started, it searches for inputs in the specified paths, and for each file it finds, Filebeat starts a so-called harvester [18]. The harvester reads a single log line (also called event) and then sends that event (as aggregated data) to the configured output: in our case Logstash.

Using Filebeat, we are able to send logs from the Nova compute nodes to the ELK stack. Being a lightweight service with minimal impact on the machine itself, it can be installed on every compute node, container or even cluster.

## Metricbeat

A service [19] used collecting metrics from systems and services (e.g. CPU stats, memory information, NGINX stats and so on), basically offering a system-level monitorization. We used Metricbeat for the shipping of system stats from the personal computer and the droplet. The configuration process for Metricbeat is straightforward, similar to Filebeat: one can simply chose the output to Logstash.

It is worth mentioning that except the case when the two Beats services [20] (i.e. Filebeat and Metricbeat) are sending logs to a local Logstash instance (running on the same machine), then the output must be configured with the IP address of the machine on which Logstash is hosted and the port through which Logstash ingests events (the default one is 5044).

Now that the services needed for developing a logging pipeline were discussed, it is necessary to describe the configuration steps that we adopted, with each step through a complete workflow. Firstly, once the ELK stack was installed on the CentOS-7 VM, we had to configure Elasticsearch’s default network to the localhost. Second step was to set up Logstash for accepting incoming beats via port 5044 and also send them to the Elasticsearch instance. For Kibana, the only configuration step was to select the Elasticsearch service running on the localhost as the default instance to be used for all the queries.

In Figure 3 we can see how the Logstash service actually works, where the pipeline is made up of three stages, namely the input (where Logstash is listening to Beats) that gathers each new incoming even, the filter stage (where one can apply different filters/plug-ins in order to transform, adjust, and restructure the event content), and finally the output stage (where the standard Elasticsearch service is selected). The configuration file is a YAML file [21] which one can change accordingly. A simple ingesting pipeline that listens to any incoming Beats, does not performs any log transformations, then finally outputs each event to Elasticsearch can be seen in Figure 4.