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

Robert Poenaru
DFCTI
IFIN-HH
Magurele, Romania
robert.poenaru@protonmail.ch

Dragos Ciobanu-Zabet
DFCTI
IFIN-HH
Magurele, Romania
zdragos@nipne.ro

***Abstract*—The full stack logging service provided by Elastic™ 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 inside our department, which will be used for ingesting, parsing, and 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.

I. 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 (DFCTI) 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 the 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 a 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. To minimize these kinds of problems, constant real-time analysis, and monitorization of the system metrics must 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 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 because 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 computing 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 any issues require an 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 that 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. 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 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 (a 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). Section IV is dedicated to the process of manipulating and parsing the incoming logs to structured data which will be stored on the Elasticsearch server. Next, in section V one can see how the log visualization was possible through Kibana. Conclusions and future work are given in section VI.

II. 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 of the workflow involved in parsing and analyzing the SLURM logs will be made in a forthcoming paper, as it is out of the scope of the current work.

A. 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]). 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. Its architecture can be seen in Appendix I.

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 that 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 the user with the Nova service is done through a REST API interface, but internally, Nova components communicate via an RPC message passing mechanism. 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 can request and consume resources without the proper knowledge of where their storage is deployed (Cinder API is also 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.

B. 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).

C. System metrics from Virtual Machines

DigitalOcean (or DO for short) [14] is a platform that allows the 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 indicates that our logging implementation is working properly.

III. ELK Stack – Development of the Logging System

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

A. 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 an 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.

B. 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 offers) are open source.

a) Elasticsearch – Based on Apache's Lucene search engine, it provides a RESTful JSON-based search engine. It also allows efficient data storage, through a so-called Elasticsearch index: the analog of a classical (SQL-type) database.

b) Logstash - Tool that allows a dynamical ingesting of data from multiple sources, transformation and parsing of the data through filters, and finally sending the filtered 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 depending 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).

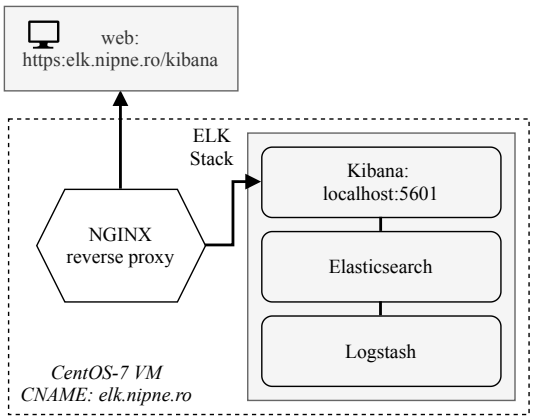

Figure 2: Log visualization with Kibana through the browser.

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

C. 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 servers 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 the 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 an event) and then sends that event (as aggregated data) to the configured output: in our case Logstash.

Using Filebeat, we can 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.

D. 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 choose 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. The 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 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 perform any log transformations, and it finally outputs each event to Elasticsearch can be seen in Figure 4.

```
input {
    beats {
        port => "5044"
    }
}
 filter {
}
output {
    elasticsearch {
        hosts => [ "localhost:9200" ]
    }
}
```

Figure 4: A simple Logstash configuration pipeline.

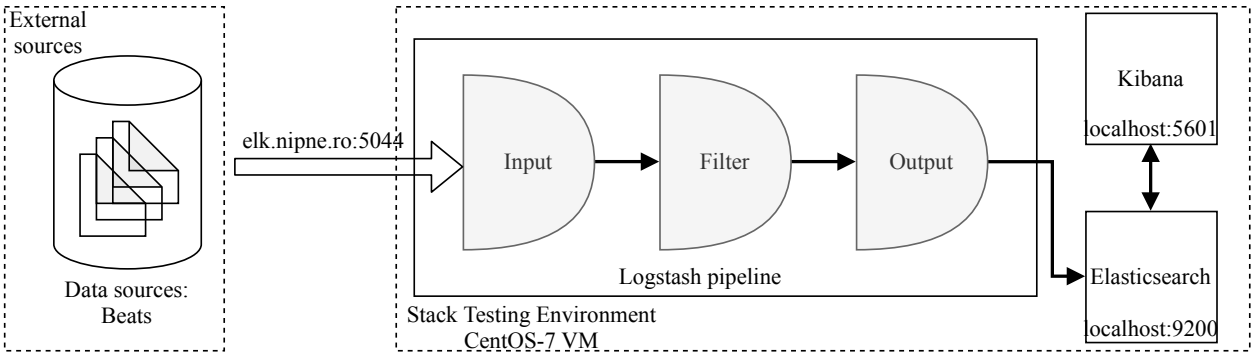

Figure 3: The Logstash pipeline.

It is easy to see why the configuration pipeline does not transform incoming events: due to the empty filter field. We used such a pipeline as a starting point for the development process. Regarding the multiple log sources what we try to analyze, one has to take into account that only one instance of Logstash can run on a machine. Because each source has different data to be shipped and analyzed, a total of three different filters need to be applied to the incoming logs. This means that instead of running three Logstash processes for each log source, a different approach must be considered. Fortunately, Logstash provides a way to have multiple configuration pipelines running at the same time [22]. As a result, we can have three configuration files constructed in the same manner as the one described in Figure 4. The scheme for such a setup that we developed can be seen in Figure 5: for each of the three sources we have a Logstash configuration pipeline that properly ingests events, transforms then accordingly, and send/store processed data to Elasticsearch.

Because the three sources have different data in terms of the provided information, it is necessary to organize each log source in a different Elasticsearch database. We already mentioned that Elasticsearch stores its data into an index. As a result, we have three different indices, namely one for the system metrics coming from the personal computers, one from the virtual machine deployed on DO, and one for storing the logs coming from the nova-compute cluster. We can perform queries on each index separately, in order to view the information inside. In Kibana, the user can select

```
- pipeline.id: pc-logs-pipeline
  path.config: "/etc/path/to/config1.conf"
  pipeline.workers: 1
- pipeline.id: droplet-logs-pipeline
  path.config: "/etc/path/to/config2.conf"
  pipeline.workers: 1
- pipeline.id: nova-logs-pipeline
  path.config: "/etc/path/to/config3.conf"
  pipeline.workers: 1
  queue.type: persisted
```

Figure 5: Multiple pipelines simultaneously running on a Logstash instance.

the desired index for data visualization and perform analysis. In terms of performance, Logstash will use 1 core per pipeline, so for the current testing environment, we can scale up to about 8 simultaneous log sources (since the CentOS-7 machine has only an 8-core CPU). However, once the project will be ready for final deployment, we can use larger machines (in terms of the number of CPU cores) that could scale up easily for the required demand in log processing power.

In Figure 6 a representation of the current setup is shown, where the personal computer and the DO droplet are sending system metrics via Metricbeat service to the ELK stack installed on the CentOS-7 VM. The compute cluster which was configured with OpenStack is sending the logs via Filebeat.

IV. Parsing of The Logs

In this section, a brief description of the parsing process for the incoming logs will be made. Fortunately, the system metrics that are ingested from the droplet and personal

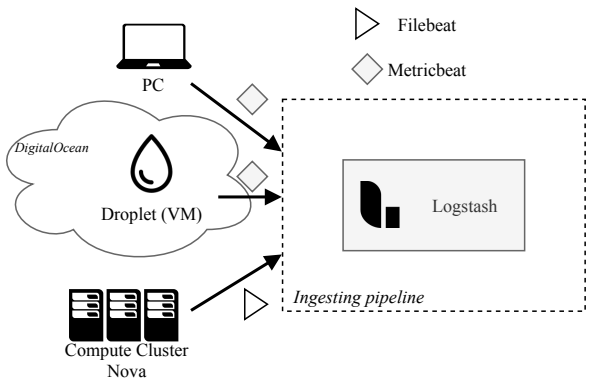

Figure 6: The three different log sources that were tested on the ELK stack

computer do not require any additional log transformation or restructuring, since the incoming events are predefined by the Metricbeat service itself (the available fields that are aggregated into a logline or event can be seen in Figure 7). However, this is not valid for the case of log data coming from the Nova compute cluster.

A. Development of the Nova compute cluster

Within our department, two computing technologies are under continuous development and improvement: i) DFCTI – Grid and ii) DFCTI Cloud Computing.

i) The Grid – contains a High-Performance Computing (HPC) stack built on top of the actual grid framework. The HPC service is configured with special tools and applications used for advanced calculus in research (e.g. Biology, Nanotechnology, Quantum Optics). On the HPC cluster, the compute nodes are configured with 56GBps InfiniBand network interfaces that allow very high bandwidth between the nodes, making sure that communication between nodes has little to no latency (the running simulations require parallelization and great scalability of the available compute resources).

ii) The Cloud Compute - The cloud compute nodes developed inside our department are services that allow hosting of virtualized computing instances (e.g. virtual machines) where up to 6 VMs can be part of the same

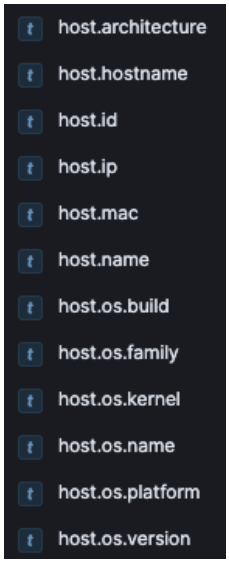

Figure 7: Fields that Metricbeat sends as an event through the logging pipeline.

```
   2020-10-01 17:54:17.631 4831 WARNING nova.pci.utils [req-fef1267e-f155-417a-b5aa-73e1fa7eb7f1 - - -
- -] No net device was found for VF 0000:3b:01.5: PciDeviceNotFoundById: PCI device 0000:3b:01.5 not
found
   2020-10-01 18:36:01.835 4831 INFO nova.compute.resource_tracker [req-fef1267e-f155-417a-b5aa-
73e1fa7eb7f1 - - - - -] Final resource view: name=dual-c phys_ram=128453MB used_ram=41472MB
phys_disk=66967GB used_disk=200GB total_vcpus=64 used_vcpus=20 pci_stats=[]
   2020-10-02 10:28:05.952 4831 WARNING nova.pci.utils [req-fef1267e-f155-417a-b5aa-73e1fa7eb7f1 - - -
- -] No net device was found for VF 0000:3b:01.6: PciDeviceNotFoundById: PCI device 0000:3b:01.6 not
found
   2020-10-02 10:28:07.238 4831 INFO nova.compute.resource_tracker [req-fef1267e-f155-417a-b5aa-
73e1fa7eb7f1 - - - - -] Final resource view: name=dual-c phys_ram=128453MB used_ram=41472MB
phys_disk=66967GB used_disk=200GB total_vcpus=64 used_vcpus=20 pci_stats=[]
```

Figure 8: Example with a few log lines sent by the Nova compute (main) node.

compute node. However, depending on the requirements, the cluster can be configured to have a node with a single VM but with a large number of CPU cores. The initialized resources that run on the cloud are available for computational tasks (e.g. simulations, numerical analysis, etc.). However, the cloud computing paradigm works very well on jobs allocated to singular resources (that is a singular VMs), also called serial jobs. Due to the lack of scalability of multiple cloud-based VM instances, running advanced simulations that require a high degree of parallelism presents a real challenge. Creating an HPC stack on top of the cloud-based VMs, which is capable of allowing communication between each VM just as fast as a physical HPC cluster was a big objective for the cloud team inside the department. The main advantage of the cloud is that VMs can have any operating system and run any kind of applications, services, and libraries, achieving in this way a large degree of flexibility in terms of the computing needs of the research teams. Initializing a virtualized compute cluster can be done very easily once the configuration process has been established.

In this section, a brief description of the parsing process for the incoming logs will be made. Fortunately, the system metrics that are ingested from the droplet and personal computer do not require any additional log transformation or restructuring, since the incoming events are predefined by the Metricbeat service itself (the available fields that are aggregated into a logline or event can be seen in Figure 7). However, this is not valid for the case of log data coming from the Nova compute cluster.

With this context in mind, one can now understand the importance of performance analysis for the Nova compute clusters deployed inside DFCTI using the ELK stack. We can centralize all the stats from each compute node into an Elasticsearch index, then perform queries or visualize aggregated data with Kibana.

B. Nova compute logs

The corresponding file of each compute service is stored in a predefined path (i.e. `/var/log/nova/`) inside of the host on which each service runs. The final resource view contains information about the hardware resources of the physical machine (e.g. RAM, used memory, physical disk space, the total number of CPU cores, and used CPU cores). An example with a few events from this log file can be seen in Figure 8. The entire log file contains more information than we actually need (e.g. timestamps, warnings, and some UUIDs). Using the Logstash filter, we were able to extract only the fields of interest, and the next step was to deliver parsed data as JSON format (key-value pairs) to Elasticsearch (inside the index dedicated for the Nova logs).

Using the grok plugin [23] it is possible to extract unstructured data into fields with corresponding values. According to the documentation, Grok works by combining text patterns into something that matches our customized logs. The syntax for a grok pattern is "`%{SYNTAX:SEMANTIC}`". An example of the grok filter inside Logstash can be seen in Figure 9, or also in a public GitHub repository, where all the work involved with the development process of the entire ELK stack is available [24].

```
filter {
  grok {
    match => { "message" =>
"%{GREEDYDATA:content}" }
  }
}
```

Figure 9: Example of a grok filter inside the Logstash configuration file.

Other than the grok filter, we also used Ruby scripts within the filter, since Logstash can execute Ruby code within the pipeline [25]. The plugin helped us in adding extra keys into the JSON data before sending the logs to Elasticsearch (i.e. fields that the log file itself did not provide). An example where we used Ruby together with grok was another log file format that we receive from the Nova compute nodes (see Figure 10), and it is in fact the central node that sends this particular data. From two different timestamps within the same event (logline) we were able to calculate a new field called `VM_duration`: see Figure 11. We were also able to obtain fields that contained aggregated data (e.g. total lifetime of VMs within a certain node, or the number of inactive VMs). This was possible using another Logstash plugin: aggregate [26].

```
2020-06-17 05:25:56,2020-06-17
05:27:14,b2793dec81d24496bbad171c74dfc965,0dc56f246ec14c92ba85
a0239a5862d9,provider-instance,fba88b4f-66ba-4fdb-ba1d-
a5e17f576e12,bchs65,1,deleted,64
2020-06-17 06:33:35,2020-06-17
06:45:30,b2793dec81d24496bbad171c74dfc965,0dc56f246ec14c92ba85
a0239a5862d9,k8s-cluster-qtiowr3ddyny-master-0,5cec469f-fb65-4cb8-
aefd-f4234e926ab0,bchs65,2,deleted,2048
2020-06-17 06:47:47,2020-06-17
07:29:19,b2793dec81d24496bbad171c74dfc965,0dc56f246ec14c92ba85
a0239a5862d9,k8s-cluster-vlwovohpq7fl-master-0,e3cb26e4-91dd-
49c6-b1a5-c0a7c9d6e6c6,bchs65,2,deleted,2048
2020-06-17 06:48:30,2020-06-17
07:28:58,b2793dec81d24496bbad171c74dfc965,0dc56f246ec14c92ba85
a0239a5862d9,k8s-cluster-vlwovohpq7fl-minion-0,eb20b53c-af89-
4094-b400-058f043cb138,bchs65,2,deleted,2048
```

Figure 10: Log lines given by a central node within the Nova compute cluster.

```
ruby {
      init => "require 'time'"
      code => '
              t1=event.get("deleted_at")
              t2=event.get("created_at")
              duration = ""
              unless (t2.empty? || t2.nil?) &&
       (t1.empty? || t1.nil?)
              duration=(Time.parse(t1).to_i-
       Time.parse(t2).to_i).abs
              end
              event.set("VM_duration",duration)
              '
}
```

Figure 11: Ruby code in the filter stage within the Logstash pipeline that is used for computing a new event field from two existing fields.

Parsing the data from Figure 10 is done with the help of a pattern developed by us, using grok. This pattern is shown in Figure 12 (see Ref. [24] for the full configuration file).

```
grok{
        match=>{
                "message"=>
                ["%{TIMESTAMP_ISO8601:created_
        at},%{TIMESTAMP_ISO8601:deleted_at},%{
        DATA:user_id},%{DATA:project_id},%{DAT
        A:display_name},%{DATA:uuid},%{DATA:la
        unched_on},%{INT:v_cpus},%{DATA:VM_sta
        te},%{INT:VM_memory_MB}"]}
    }
```

Figure 12: Grok pattern for parsing logs given by a central compute node.

The events coming from a central compute node provides information with regards to the time of creation and destruction for every VM that was deployed on that particular node, the user ID which solicited the deployment of the mentioned VM and the project ID for the allocated jobs, information about the users' group, the unique ID of the VM, and the name of the node itself. Parsing this data and using aggregated filters proven to be very useful in getting insight about the total number of cloud-based virtual machines running jobs on the compute node, the aggregated time of running jobs on each VM, but also information about the users (e.g. number of VMs per user).

V. User Interaction - Kibana

In the last stage of the ELK stack, the data visualization using Kibana is possible. We already discussed the process of setting up a reverse proxy through the NGINX service, which will allow us to access the VM's localhost (on port 5601) from the web. Because we have a registered CNAME (elk.nipne.ro) and a static IP of the VM itself, we tied `localhost:5601` to `elk.nipne.ro/kibana`. In Figure 13 one can see a screenshot of the Kibana UI together with some events inside the index where Nova logs are stored (logs which are initially ingested and parsed through Logstash as described in the previous section). Within the screenshot, section 1 contains each event as it is coming, section 2 contains the available fields within the Elasticsearch index, and finally, in section 3 we can follow a real-time evolution with the incoming events (counts per unit time).

VI. Conclusions & Future Work

Within the present work, we showed the workflow involved in the development process of an ELK stack that ingests logs coming from three different sources. The first two sources consisted of system metrics from a personal computer and a virtual machine hosted on a 3rd party service (DigitalOcean). The two machines sent logs via the Metricbeat service and storing the two logs in real-time on

Figure 13: Kibana UI. Access through the browser at https://elk.nipne.ro/kibana.

the Elasticsearch server within our department was successful. The third log source was a cloud compute cluster developed using Nova and OpenStack. The custom log files were parsed using multiple plugins within the filter stage of Logstash. Output into a dedicated Elasticsearch index was also successful. Every index could be visualized with Kibana, using the web browser. Reverse proxy with NGINX allowed remote access to the Kibana instance which runs on the testing environment (the CentOS-7 VM) as localhost.

The developed ELK stack proved to be ready for a final deployment inside the department, where logs coming from a larger compute cluster and larger compute grid network will need constant monitorization and analysis. If the current computing resources of the VM that hosts the stack are not enough, we will consider the transition to a more powerful machine and setting up the same ELK stack.

We believe that the currently developed logging pipelines are capable of ingesting a large volume of events, and scalability should not be an issue and we are confident that the ELK stack can meet the needs of the entire department (considering the fact that the compute power available inside DFCTI will increase even more in the near future).

During the development phases, we always worked with the latest available versions of each service within the stack: we had the latest available versions of Elasticsearch, Logstash, Kibana, Filebeat, Metricbeat, and NGINX.

Future Work

- We are currently looking into new ways of extracting more information related to the state of each compute node, virtual machine, and service that runs on a VM. Being able of adding extra fields with aggregated data or on-the-fly event calculation: just like we did with the ruby plug-in when the VM lifetime was computed. It is essential that we can provide stats for every component within the computational workflow (e.g. nodes, VMs, services, apps, network, users, projects).
- Introducing Machine Learning into the stack. ML has been proven a powerful tool within the ELK stack: anomaly detection can categorize events and then find anomalous event counts within user-defined time buckets. It can also find anomalous event ingest rates. A user can build and tune machine learning jobs to visualize these anomalies [27].

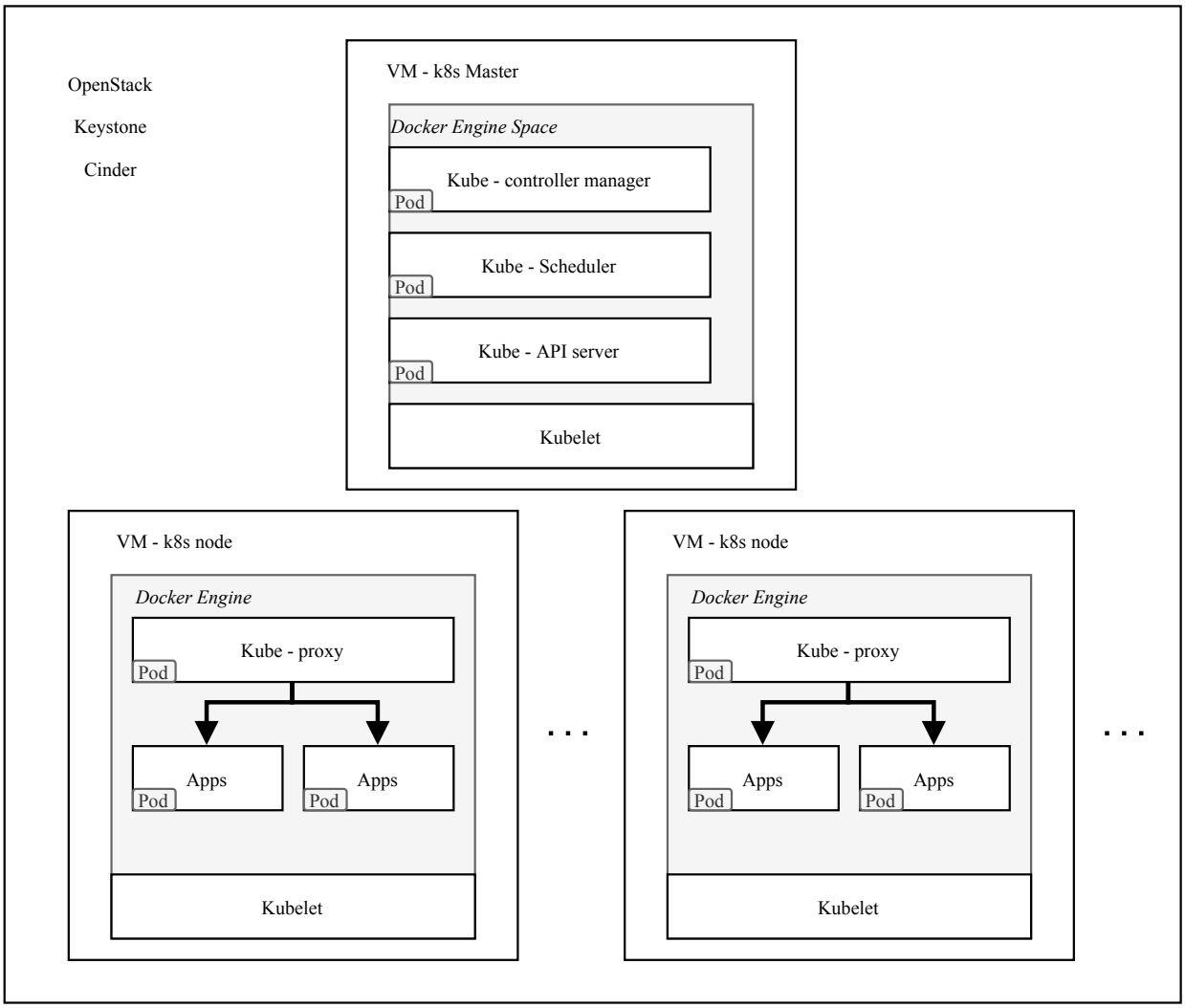

Figure A1: Nova compute architecture deployed using k8s clusters.

VII. Acknowledgments

The authors thank the Department of Computational Physics and Information Technology (DFCTI) at IFIN-HH, for providing access to the CLOUDIFIN site as testing environment of the ELK stack. This work would not have been possible without the support from the Ministry of Education and Research (MER) under contract no. 7 / 2020 (PN3-5.2-CERN-RO). The second author was partially supported by the MER under project PN19060205 and the European Regional Development Fund through project CECBID-EOSC (POC/397/1/1-124405).

Appendix I

The architecture of a Kubernetes cluster developed using OpenStack and Nova services can be seen in Figure A1.

References

[1] Elasticsearch service, (site available November 2020): https://www.elastic.co/elasticsearch/

[2] Elastic – the company (site available November 2020): https://www.elastic.co/about/

[3] The ELK stack, (site available November 2020): https://www.elastic.co/what-is/elk-stack

[4] Kubernetes, main webpage (site available November 2020): https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[5] The OpenStack project: official website (site available November 2020): https://www.openstack.org/

[6] CeCBiD-EOSC: project within NIPNE-HH, sustained by European Regional Development Fund

[7] SLURM workload manager, main webpage (available November 2020): https://slurm.schedmd.com/overview.html

[8] Nova – OpenStack, official documentation (available November 2020): https://docs.openstack.org/nova/latest/

[9] Keystone – OpenStack, main webpage (available November 2020): https://www.openstack.org/software/releases/victoria/components/keystone

[10] Keystone – OpenStack, official documentation (available November 2020): https://docs.openstack.org/keystone/latest/

[11] Cinder – OpenStack, main webpage (available November 2020): https://www.openstack.org/software/releases/victoria/components/cinder

[12] Cinder – OpenStack, official documentation (available November 2020): https://docs.openstack.org/cinder/latest/

[13] Docker Containers, official documentation – Orientation and setup (available November 2020): https://docs.docker.com/get-started/

[14] DigitalOcean, Main webpage (available November 2020): https://www.digitalocean.com/

[15] DigitalOcean, Main webpage (available November 2020): https://www.digitalocean.com/products/droplets/

[16] NGINX, Official documentation (available November 2020): https://docs.nginx.com/nginx/admin-guide/installing-nginx/installing-nginx-open-source/

[17] Filebeat, Official documentation (available November 2020): https://www.elastic.co/guide/en/beats/filebeat/current/how-filebeat-works.html

[18] Filebeat – Harvester, (available November 2020): https://www.elastic.co/guide/en/beats/filebeat/7.9/filebeat-overview.html

[19] Metricbeat, Official documentation (available November 2020): https://www.elastic.co/beats/metricbeat

[20] Beats – Elastic, Official webpage (available November 2020): https://www.elastic.co/beats/

[21] YAML, Official webpage (available November 2020): https://yaml.org/

[22] Logstash – Multiple Pipelines (available November 2020): https://www.elastic.co/guide/en/logstash/current/multiple-pipelines.html

[23] Grok filter plugin, Elastic documentation (available November 2020): https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html#plugins-filters-grok

[24] ELK Stack - Public Repository on GitHub (available November 2020): https://github.com/basavyr/ELK-Stack-macOS

[25] Ruby filter plugin, Elastic documentation (available November 2020): https://www.elastic.co/guide/en/logstash/current/plugins-filters-ruby.html

[26] Aggregate filter plugin, Elastic documentation (available November 2020): https://www.elastic.co/guide/en/logstash/current/plugins-filters-aggregate.html

[27] ZELK vs ELK: Zebrium vs Elastic Machine Learning, webpage available November 2020: https://www.zebrium.com/blog/zelk-vs-elk-zebrium-ml-vs-elastic-machine-learning-zebrium