Kibana Dashboards for Grid Services

Emiliano Di Marino
e.dimarino5@gmail.com

Information **T**echnology Department
Platform & **E**ngineering **S**ervices Group
Platform **S**ervices Section

August 26, 2015

The **P**latform **S**ervice (PS) section of the CERN runs a number of grid services used by thousands of users worldwide. So as to manage these services even more successfully, plan their future, be able to rapidly address problems and easily provide information to the users, it would be useful to have dashboards reporting technical status data. The team of the PS section has put together a monitoring infrastructure based on Kibana/ElasticSearch and found that it can be possible to design dashboards and write collectors rather easily. The main goal of this summer student project is to create more of such dashboards to help PS section to monitor CERN's grid services.

1 Introduction

1.1 Cloud Computing

Cloud computing refers to the delivery of on-demand computing resources (both hardware and software) over the internet [11]. A cloud computing provider can offer its services according three different models [5]:

- **SaaS** (**S**oftware **a**s **a** **S**ervice): users can run software packages installed on remote machines supplied by the service provider (often accessible via web server);
- **PaaS** (**P**latform **a**s **a** **S**ervice); users can develop, run, improve and debug applications on the platform supplied by the service provider;
- **IaaS** (**I**nfrastructure **a**s **a** **S**ervice); users can use hardware and/or software resources supplied by the service provider.

In figure 1 are represented the three cloud computing models.

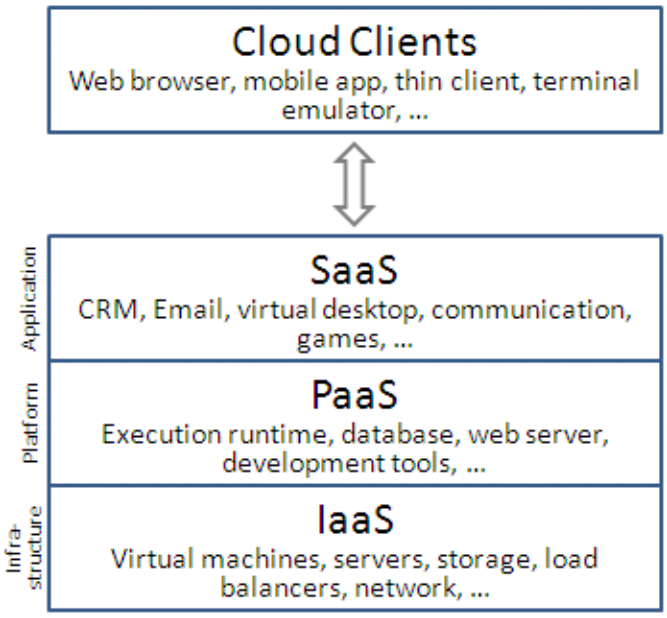

Figure 1: Cloud Computing models

Each of the cloud computing model above can be developed according to three different types of cloud computing infrastructure:

- **Public Cloud**: it is based on the standard cloud computing model, in which a service provider makes resources available to the general public over the Internet [15];

- **Private Cloud**: it is a cloud infrastructure built only for a single organization. It may be owned and managed by the same organization, a trusted third part company or a combination of them [3];
- **Community Cloud**: it is developed for a specific group of users (called community) who belong to different organizations which share the same interests. It may be owned and managed by one or more of the organizations in the community, a trusted third part, or some combination of them [3];
- **Hybrid Cloud**: it is a composition of two or more distinct cloud infrastructure listed above (Public, Private, Community).

CERN **private** cloud computing infrastructure was developed according to the **IaaS** model and it is based on OpenStack [1]. Thanks to the CERN Data Center, scientists can manage all the data coming from the LHC's experiments and it is possible to monitor the entire scientific, administrative and computing infrastructure. An extension of the main Data Center is hosted in Hungary, at the Wigner Research Centre for Physics. The 110000 processor cores and 10000 servers run 24/7 to process one petabyte of data each day. Furthermore, thanks to 35000 km of optical fibre it was possible to build a network infrastructure that is essential for running the computing facilities and services of the laboratory as well as for transferring the colossal amount of LHC data to and from the Data Center. For more information visit `http://information-technology.web.cern.ch/about/computer-centre`

Figure 2: CERN Main Data Center.

2 Brief Overview of the involved technologies

The aim of this section is to provide an high-level overview of the main technologies used in this project.

2.1 OpenStack

OpenStack is a cloud operating system that, according to the official definition in [12], is able to control large pools of computer, storage, and networking resources throughout a data center. All these resources are managed through a dashboard (or via OpenStack API) that gives administrators control while empowering their users to provision resources through a web interface. In figure 3 are illustrated the main elements of the OpenStack architecture and their relationships.

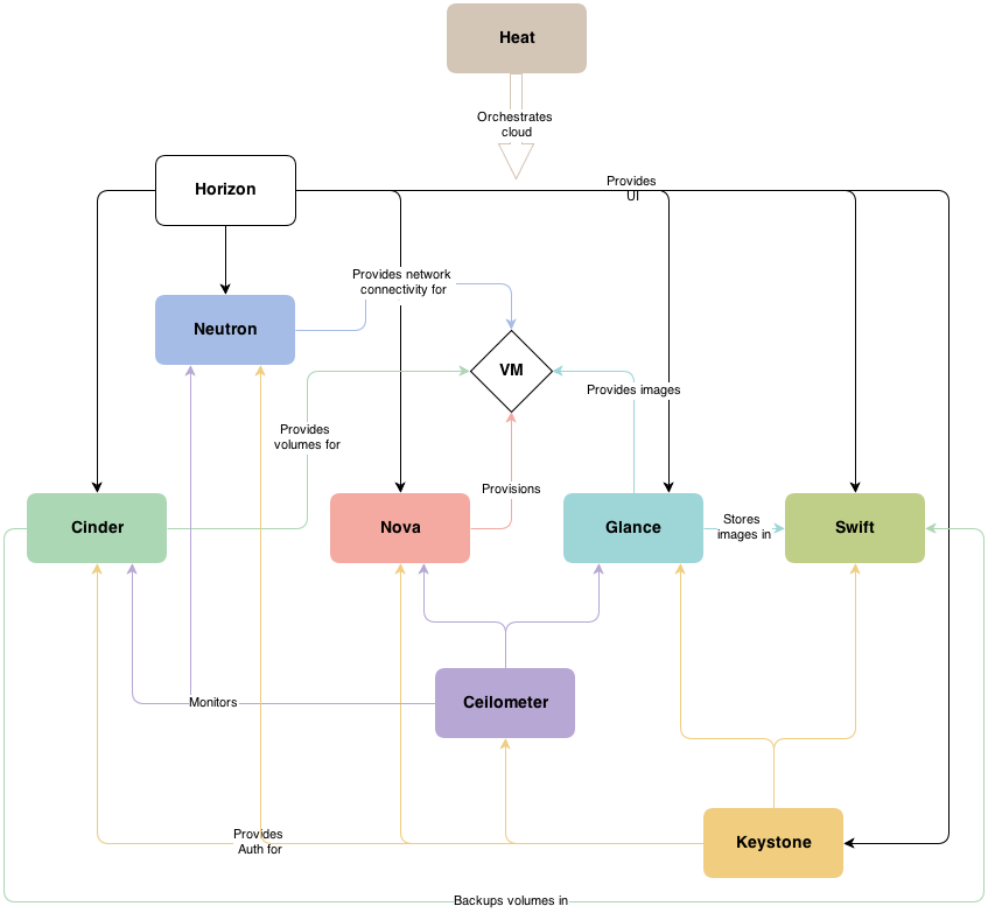

Figure 3: OpenStack Architecture

Below there is a brief explanation of the components of the OpenStack architecture:

- **Nova (Compute)**: this element provides a cloud computing fabric controller[1], supporting a wide variety of virtualization technologies [10];
- **Neutron or Quantum (Networking)**: this component enables the final user to create his own network and interfaces as well as manage IPs [16];
- **Swift and Cinder (Storage)**:
  - **Swift (Object Storage)**: it manages static data such as virtual machine (VM) images, photo storage, email storage, backups and archives [13];
  - **Cinder (Block Storage)**: it manages the creation, attaching and detaching of the block devices to servers [13].
- **Glance (Image Service)**: this component defines services for discovering, registering, retrieving and storing virtual machine images [8];
- **Keystone (Identity Service)**: it provides authentication, authorization and service discovery mechanisms via HTTP [9];
- **Horizon (Dashboard)**: this component provides the main interface to interact with Openstack modules;
- **Ceilometer (Telemetry Service)** this element monitors the usage of the Cloud services and decides the billing accordingly. Furthermore it is also used to decide the scalability and obtain the statistics regarding the usage.

2.2 Configuration Management System

To facilitate the management of the CERN's private cloud, a configuration management system (CMS) was introduced. Thanks to this software, users can manage easily the configuration of any machine they have access. The CMS is based on Puppet. It allows users to define the state of their IT infrastructure, and then automatically enforces the desired state. Puppet automates every step of the software delivery process, from early-stage code development through testing, production release and updates [14]. It is based on a declarative language for expressing system configuration, a client and server for distributing it and a library for realizing the configuration [4].

The CMS is based on the following components [2]:

[1] A Fabric Controller is a component that knows the what, where, when, why, and how of the resources in cloud.

- **Puppet**: it is the daemon which runs on the nodes controlled by an user and it is used to configure and manage them;
- **Facter**: this component provides information about the nodes (e.g. hardware details, network settings, OS type and version, IP addresses, MAC addresses, SSH keys...) to Puppet;
- **PuppetDB**: this component collects data generated by Puppet. If a user wants to know some information about the nodes, they can query it;
- **Puppet manifest** (code): it is the code written in DSL language which is stored in git and describes the machine configurations;
- **Hiera** (data): it is a key/value lookup tool for configuration data. The data is also stored in git;
- **Foreman**: it is a management tool for physical and virtual servers. It is used to assign (and create) individual hosts to clusters (called hostgroups) and to visualise the reports returned from Puppet runs.
- **Mcollective**: this tool allows users to run a variety of commands remotely on the node and retrieve the results.

The diagram in figure 4 shows what happens when a node tries to configure itself (in this case it is example.cern.ch). As mentioned above, all the hosts in the configuration system are registered in the Foreman service.

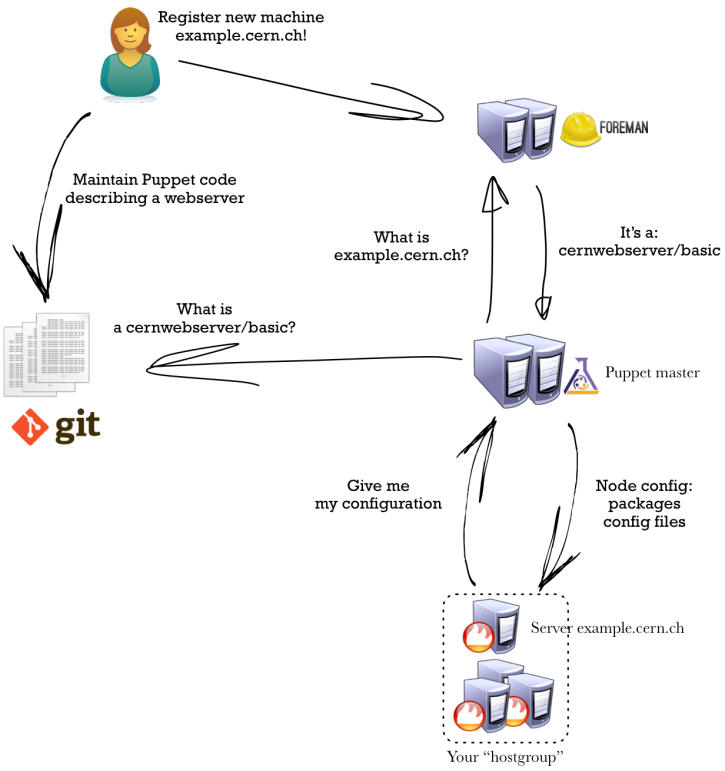

Figure 4: How Puppet works

1. The puppet daemon which is in execution on the node will request its configuration from the puppet masters;
2. The puppet master:
   a) asks the Foreman service which hostgroup it belongs to;
   b) find out what are the packages and configuration needed for the hostgroup (in this case webserver/basic). It can do this by reading the Puppet manifest (code) and Hiera (data) stored in a git repository;
   c) compiles the configuration for that node and hands it back to the nodes puppet daemon for it to apply.
3. If the node (webserver/basic) has any configuration that is out of step with the desired configuration:
   a) the appropriate changes are made by the local puppet daemon. In the other case, it does nothing.

Steps from 2 to 3 are referred to as a puppet run, and typically happens several times a day on all nodes in the system.

Let us focus, now, on:

- Hostgroups in Foreman and their structure.

An hostgroup in Foreman represents the concept of a "cluster". It has a hierarchical structure, can be sub-divided into sub-hostrugrups and it is easy to understand that a set of nodes that belong the same hostgroup have the following proprieties:

1. are all part of the same "service";
2. share some common configuration;
3. are under the control of the same group of people.

The Puppet code is stored inside the version control system git (as seen in figure 4), and arranged so that each top-level hostgroup has its own git repository. In this way, each hostgroup can be managed by different group op people and the configuration of a specified houstgrup can be different from the configuration of the another one.

2.3 ElasticSearch

Elasticsearch is a distributed RESTful[2] search engine built for the cloud [6]. It can be used for a near real time search of all kinds of documents, to discover and analyze data in a manner that users can improve their business. It provides tools to detect new and/or failed nodes and it is able to reorganize and rebalance data to ensure the accessibility to the information automatically. Other features are provided too. Elasticsearch is based on Apache Lucene that is a high performance, full-featured Information Retrieval library, written in Java.

To interact with elasticsearch Kibana was developed. Basically it is a web frontend to analyze data held in the cluster [7]. This means that it helps users to understand data with tools created for searching and visualizations. Then, it is possible to combine those visualizations to build dashboards.

[2]Representational State Transfer (REST) is a software architecture style for building scalable web services

2.4 Apache Flume

Apache Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It is written primarily in Java and has been tested on unix-like systems.

The Architecture illustrate in figure 5 is made up of three fundamental components:

- **Source**: that consumes events delivered to it by an external source like a web server (the last one sends events to Flume in a format that is recognized by the target Flume source):
- **Channel**: is a passive store that keeps the event until it's consumed by a Flume sink;
- **Sink**: The sink removes the event from the channel and puts it into an external repository like HDFS or forwards it to the Flume source of the next Flume agent in the flow. The source and sink within the given agent run asynchronously with the events staged in the channel.

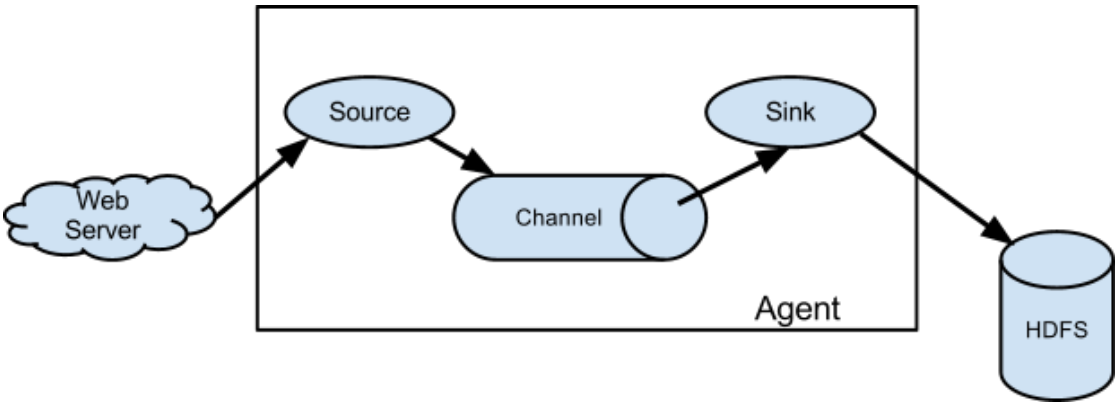

Figure 5: Apache Flume Architecture

Flume has the capability to modify/drop events in-flight. This is done with the help of interceptors. An interceptor can modify or even drop events based on any criteria chosen by the developer of the interceptor. In this project the **Morphline** interceptor was used.

3 Project

The main purpose of this project is to build several Kibana dashboards to analyse huge amounts of information which are very useful to improve the reliability of CERN's grid services. Furthermore, these information will be used to manage grid services as better as possible, plan their future, provide information to the CERN's users and get feedbacks from them.

Specifically, CERN IT PES Group is interested in analysing information from:

- logs stored in 3 squid[3] proxy servers called "**front20.cern.ch**", "**front21.cern.ch**" and "**front22.cern.ch**". In each of these server are stored $\sim$ 150000000 logs ($\sim$ 35 GB) per week;
- logs stored in **lcgvoms21.cern.ch**. This server acts as VOMS[4] server and it is used as a central repository for user authorization information.

This project focuses the attention on "front20.cern.ch" and "lcgvoms21.cern.ch" servers. In order to manage and analyze these data the following steps have been performed:

1. study the format of the log files and improve it if necessary;
2. create an elasticsearch cluster with enough nodes to manage the data;
3. configure Apache Flume agent and the Morphline interceptor to send the files to the cluster;
4. build a Kibana dashboard to get the information required.

The information required for the two types of logs are different. For logs stored in the 3 squid servers it is useful to know:

- Number and type of hits and non-hits;
- Number of clients ordered by the number of requests and by size;
- Number of domains ordered by the number of requests and by size;
- Geographic plot of requests.

Instead, for the logs stored in lcgvoms21.cern.ch, the information to obtain are:

- Number of requests by V.O.;
- Number of requests by users;
- Geographic plot of requests.

[3]Squid is a caching and forwarding web proxy. It has a wide variety of uses, from speeding up a web server by caching repeated requests; to caching web, DNS and other computer network lookups for a group of people sharing network resources; to aiding security by filtering traffic.

[4]VOMS is an acronym used for Virtual Organization Membership Service in grid computing.

3.0.1 Overview and improvement of Squid Log Files

The default format of the squid log file used in the "front20.cern.ch", "front21.cern.ch", "front22.cern.ch" is the following:

time elapsed remotehost resultcodes bytes method URL user hierarchycode type (1)

where:

- **time**: it is the Unix timestamp as UTC[5] seconds with a millisecond resolution. It represents the time when Squid started to log the transaction;
- **elapsed**: it represents how many milliseconds the transaction busied the cache;
- **remotehost**: it represents the the client IP address;
- **resultcodes**: this field is made up of two entries separated by a slash called code and status where:
  - **code** represents information about the kind of request, how it was satisfied, or in what way it failed;
  - **status** contains the HTTP result codes with some Squid specific extensions. Check the following link to consult Squid result codes `http://wiki.squid-cache.org/SquidFaq/SquidLogs#Squid`;
- **bytes**: it represent the amount of data delivered to the client;
- **requestmethod**: it represent the request method to obtain an object. Check the following link to consult the available Squid request methods `http://wiki.squid-cache.org/SquidFaq/SquidLogs#Request_methods`;
- **URL**: it is the URL requested.
- **user**: it represents the user identity for the requesting client;
- **hierarchycode**: this information basically consists of two items:
  - A code that explains how the request was handled (see the following link for more information `http://wiki.squid-cache.org/SquidFaq/SquidLogs#Hierarchy_Codes`);
  - The IP address or hostname where the request (if a miss) was forwarded.
- **type**: it represents the content type of the object as seen in the HTTP reply header.

At the following link `http://wiki.squid-cache.org/Features/LogFormat` there other details about the parameter described above.

[5]UTC or Coordinated Universal Time is a time standard.

In order to get more useful information the default squid log format was modified. The hostname and the user agent of the client were added at the end of string described in 1. The final format of the log string is:

time elapsed remotehost resultcodes bytes method URL user hierarchycode type clienthostname useragent

If you want to know how to modify the squid log format you can visit `http://www.squid-cache.org/Doc/config/logformat/`.

3.0.2 Overview of VOMS Log Files

There is not an official documentation of the format of the log files stored in "lcgvoms21.cern.ch" server. For this reason, few examples of log string have been analyzed and some of them are reported in this section. Furthermore, PS Section is interested in gathering information about log files stored in the folders /var/log/voms and /var/log/voms-admin. The first examples of log strings coming from "alice.log" file stored in /var/log/voms:

- Wed Aug 12 13:53:13 2015:lcgvoms21.cern.ch:vomsd[5330]: msg="LOG_INFO:REQUEST:makeAC (vomsd.cc:1168):Issued FQAN: /alice/lcg1/Role=NULL/Capability=NULL"
- Wed Aug 12 13:53:24 2015:lcgvoms21.cern.ch:vomsd[17199]: msg="LOG_INFO:REQUEST:logconnection (ipv6sock.cc:64):Received connection from: 147.231.25.183:59720."
- Wed Aug 12 13:53:24 2015:lcgvoms21.cern.ch:vomsd[17199]: msg="LOG_INFO:REQUEST:Run (vomsd.cc:773):Started child executor with pid = 5621. Active requests = 1"

Basically, these strings are made up of three parts:

- a first part that represents the timestamp:
- a second part before the real message;
- a third part that starts with "msg".

Instead, the following lines coming from "voms-admin-vo.sixt.cern.ch.log " files stored in /var/log/voms-admin

- 2015-08-12 11:51:25.387Z - INFO [InitSecurityContext] - Connection from "188.184.135.229" by /DC=ch/DC=cern/OU=computers/CN=lcg-voms2.cern.ch (issued by "/DC=ch/DC=cern/CN=CERN Grid Certification Authority", serial 237121519819302478350696)
- 2015-08-12 11:51:25.393Z - INFO [VomsAdminService] - getVOName();

Also these strings can be divided into three parts:

- a first part that represents the timestamp;
- a second part that starts with "-" and ends with "-";
- a third part that in the fist example starts with "-" and ends with ")" and in the second example starts with '-"' and ends with ";".

3.0.3 Create the ElasticSearch cluster

Thanks to the OpenStack infrastructure it is very easy to create and manage VMs. For this project a cluster composed by the following machines was created:

- 1 VM with medium flavour as master node called elasticsearch-master-emdimari.cern.ch
- 9 VMs with medium flavour for the data nodes (the first five machines have a 20 GB Ceph volume attached) called elasticsearch-dat$i$-emdimari.cern.ch with $i \in 1 \ldots, 9$.

The name of the cluster is emdimari-cluster. In the picture 6 is represented the structure of the cluster created.

Figure 6: ElasticSearch cluster

The master machine was hosted in the hostgroup playground/emdimari/master. The other one in the playground/emdimari/data.

In order to better manage files on each machine Puppet was used. So, the manifests "master.pp" and "data.pp" were defined in playground/code/manifests/emdimari.

Mainly, these files are used to install ElasticSearch and Kibana on the cluster, to set its name, to decide what machine is used as master node and what machines are the datanodes. Furthermore another important step was to create the template to apply to the data in the cluster.

The template for the the squid logs is described in "squid_template.json" file; the template for the logs stored in the "lcgvoms21.cern.ch" server is described in "lcgvoms21_template.json" and both are located in playground/code/files/emdimari. These templates are used to define the mapping[6] of the data. To know how to apply them to the data in in elasticsearch visit `https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-templates.html`.

[6]Mapping is the process of defining how a document should be mapped to the Search Engine

3.0.4 Configure Apache Flume and the morphline Interceptor

A very important step was to configure the flume agent on the "front20.cern.ch" and "lcgvoms21.cern.ch" in is such way that it can tail a list of files and send the data directly to the elasticsearch cluster.

To achieve this task the puppet manifests "live.pp" of the first server and "admin.pp" for the last one were modified.

Basically with this edit, these machines are able to find the flume configuration file (`flume_cvmfs.erb, flume_voms.erb, flume_voms_admin.erb`) and the morphline configuration file (`morphline_cvmfs.erb, morphline_voms.erb , morphline_voms_admin.erb`). On an high level point of view, in the flume configuration file are define how many sources, channel and sinks are used, the size of these components, where is/are the file/s to be read, the type of Interceptor to be used and the name of the cluster where send the data. In the morphline configuration file is described in which way the log string is splitted and the name of each created field.

For the squid logs a single flume agent with one source, one channel and one sink was used. In figure 7 is represented the architecture of the flume agent created. The flume configuration file and the morphline configuration file are hosted in cvmfs/code/templates

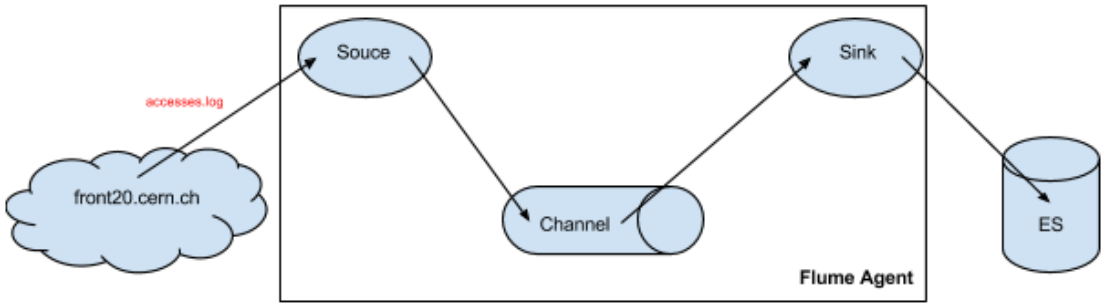

Figure 7: Flume Architecture for squid log files

For log files in /var/log/voms and /var/log/voms-admin was used a single flume agent in which the number of sources, channels and sinks was equal to the number of log files to be read. In figure 8 is reported the flume architecture for files stored in "lcgvoms21.cern.ch" server.

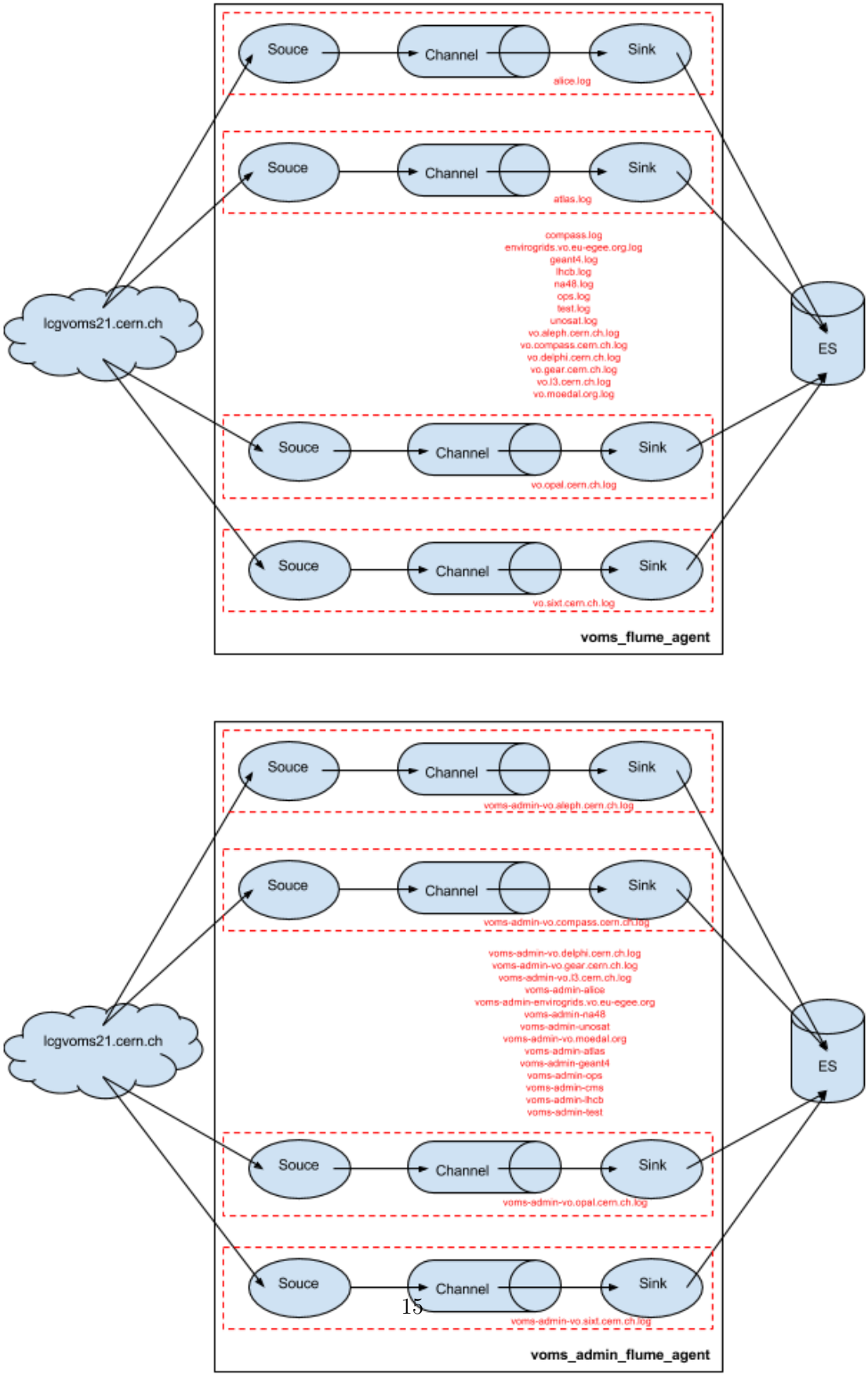

Figure 8: Flume architecture for log files stored in lcgvoms21 server

3.0.5 Create a Kibana dashboard

Once the elasticsearch cluster was created (using OpenStack) and populated with data (sent with Apache Flume and splitted with Morphline interceptor), two Kibana dashboards were built.

The first dashboard is used to analyse data coming from "front20@cern.ch" server and the second one to analyse logs from "lcgvoms21.cern.ch".

In the following figures (9, 10, 11, 12, 13) are represented some parts of the dashboards developed.

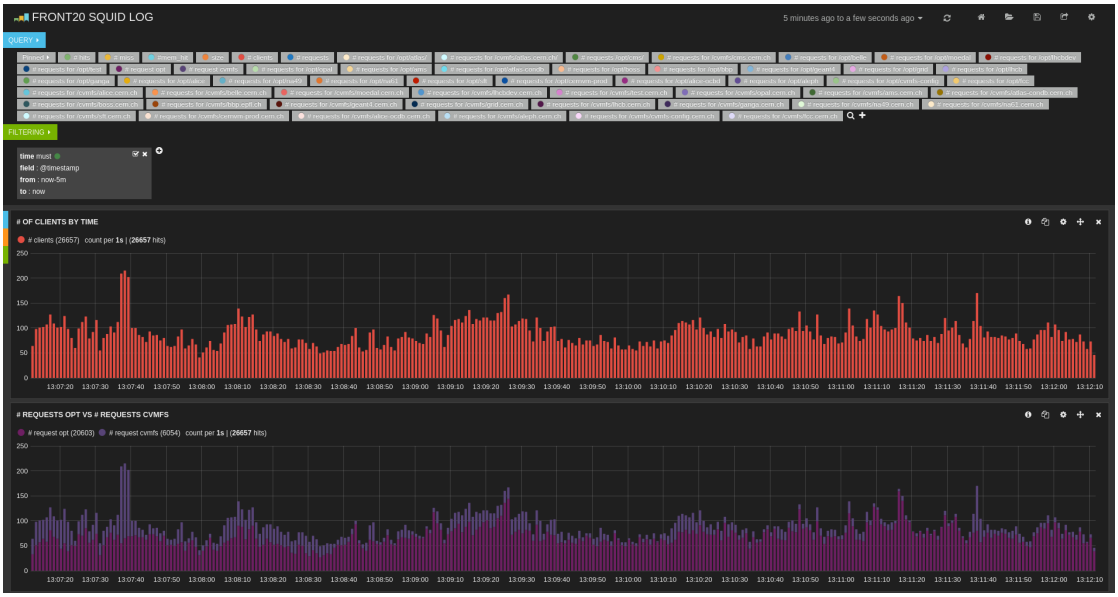

Figure 9: Part of the Kibana dashboard for front20.cern.ch

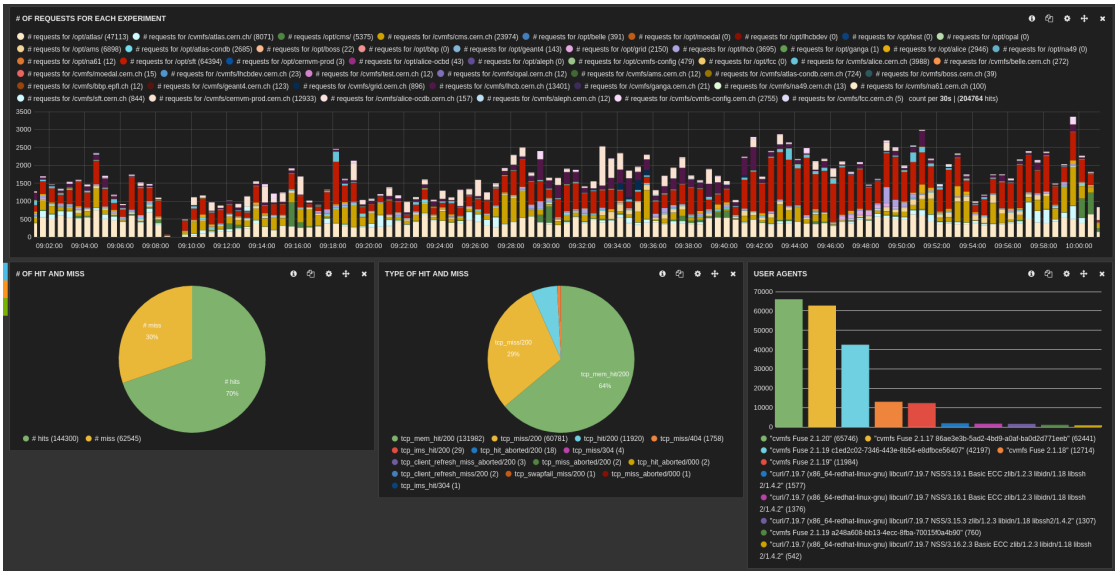

Figure 10: Part of the Kibana dashboard for front20.cern.ch

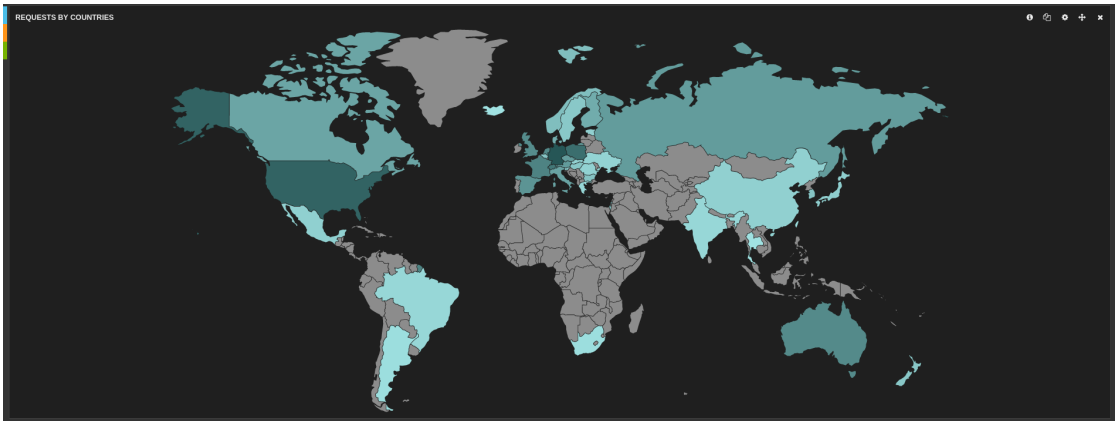

Figure 11: Part of the Kibana dashboard for front20.cern.ch

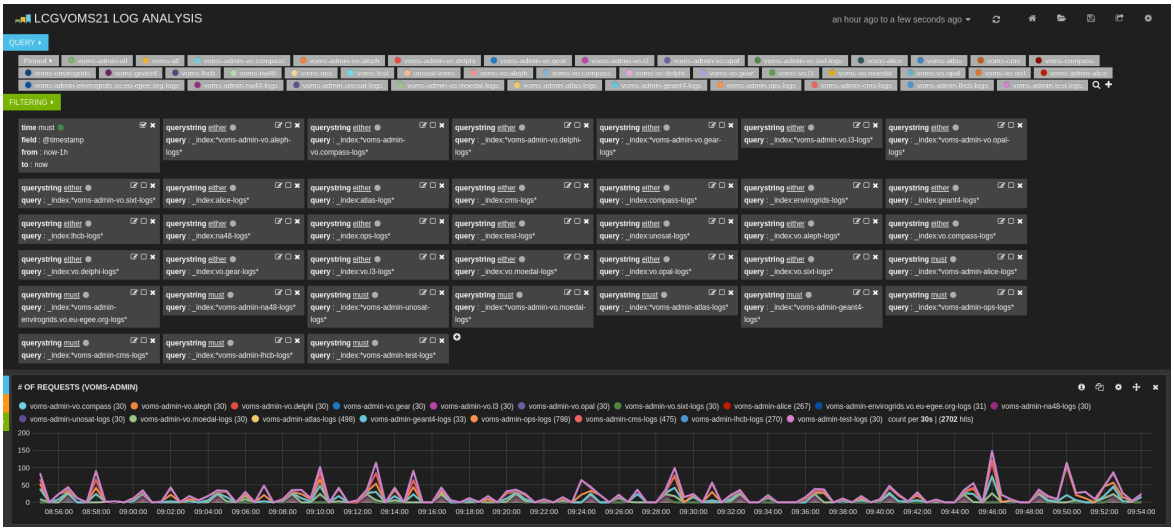

Figure 12: Part of the Kibana dashboard for lcgvoms21.cern.ch

Figure 13: Part of the Kibana dashboard for lcgvoms21.cern.ch

References

[1] Cern openstack private cloud guide. http://clouddocs.web.cern.ch/clouddocs/index.html.

[2] Puppet architecture overview. http://configdocs.web.cern.ch/configdocs/overview/system.html.

[3] The nist definition of cloud computing. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf.

[4] Overview of puppet (guide). http://docs.puppetlabs.com/guides/faq.html#what-is-puppet.

[5] Cloud computing models. https://en.wikipedia.org/wiki/Cloud_computing.

[6] Elasticsearch definition. https://github.com/elastic/elasticsearch.

[7] Kibana definition. https://github.com/elastic/kibana.

[8] Openstack image service overview. https://github.com/openstack/glance.

[9] Openstack identity overview. https://github.com/openstack/keystone.

[10] Openstack nova overview. https://github.com/openstack/nova.

[11] Cloud computing definition. http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html.

[12] Openstack: The open source cloud operating system. http://www.openstack.org/software/.

[13] Openstack storage overview. https://www.openstack.org/software/openstack-storage/.

[14] Overview of puppet (definition). https://puppetlabs.com/puppet/what-is-puppet.

[15] Public cloud definition. http://searchcloudcomputing.techtarget.com/definition/public-cloud.

[16] Openstack neutron overview. https://wiki.openstack.org/wiki/Neutron.