Towards Monitoring-as-a-service for Scientific Computing Cloud applications using the ElasticSearch ecosystem

Journal of Physics: Conference Series **664** (2015) 022040 doi:10.1088/1742-6596/664/2/022040

Towards Monitoring-as-a-service for Scientific Computing Cloud applications using the ElasticSearch ecosystem

S Bagnasco[1], D Berzano[2], A Guarise[1], S Lusso[1], M Masera[1,3], S Vallero[1,3]

[1] Istituto Nazionale di Fisica Nucleare, Via Pietro Giuria 1, 10125 Torino, IT
[2] CERN - European Organization for Nuclear Research, CH-1211 Geneva 23, CH
[3] Dipartimento di Fisica, Università degli Studi di Torino, Via Pietro Giuria 1, 10125 Torino, IT

E-mail: svallero@to.infn.it

Abstract. The INFN computing centre in Torino hosts a private Cloud, which is managed with the OpenNebula cloud controller. The infrastructure offers Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) services to different scientific computing applications. The main stakeholders of the facility are a grid Tier-2 site for the ALICE collaboration at LHC, an interactive analysis facility for the same experiment and a grid Tier-2 site for the BESIII collaboration, plus an increasing number of other small tenants. The dynamic allocation of resources to tenants is partially automated. This feature requires detailed monitoring and accounting of the resource usage. We set up a monitoring framework to inspect the site activities both in terms of IaaS and applications running on the hosted virtual instances. For this purpose we used the ElasticSearch, Logstash and Kibana (ELK) stack. The infrastructure relies on a MySQL database back-end for data preservation and to ensure flexibility to choose a different monitoring solution if needed. The heterogeneous accounting information is transferred from the database to the ElasticSearch engine via a custom Logstash plugin. Each use-case is indexed separately in ElasticSearch and we setup a set of Kibana dashboards with pre-defined queries in order to monitor the relevant information in each case. For the IaaS metering, we developed sensors for the OpenNebula API. The IaaS level information gathered through the API is sent to the MySQL database through an ad-hoc developed RESTful web service. Moreover, we have developed a billing system for our private Cloud, which relies on the RabbitMQ message queue for asynchronous communication to the database and on the ELK stack for its graphical interface. The Italian Grid accounting framework is also migrating to a similar set-up. Concerning the application level, we used the Root plugin TProofMonSenderSQL to collect accounting data from the interactive analysis facility. The BESIII virtual instances used to be monitored with Zabbix, as a proof of concept we also retrieve the information contained in the Zabbix database. In this way we have achieved a uniform monitoring interface for both the IaaS and the scientific applications, mostly leveraging off-the-shelf tools. At present, we are working to define a model for monitoring-as-a-service, based on the tools described above, which the Cloud tenants can easily configure to suit their specific needs.

1. Introduction

1.1. The INFN Torino Private Cloud

The INFN-Torino computing centre hosts a private Cloud infrastructure, which provides Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) services to different scientific computing applications. The infrastructure counts more than 70 hypervisors (providing about 1300 computing cores shared among 250-300 virtual machines) and 1600 TB of gross storage. Local connectivity within services and hypervisors is handled with 1 Gbps links. The WAN link is 10 Gbps via the Italian NREN GARR-X. Storage servers are connected with 10 Gbps links. The deployment of virtual machines is managed with the OpenNebula cloud controller [1]. For more details on the Torino Cloud infrastructure see [2].

The main stakeholder of the facility is a WLCG Tier-2 site, which serves mainly the ALICE collaboration. About 60% of the virtual instances running at the Torino Cloud are grid worker-nodes. The second largest application is a separate DIRAC grid site for the BESIII collaboration, which counts about 200 dedicated cores. Resources are allocated statically to these applications, although we are working to achieve dynamic cluster resizing according to the load [3].

At the PaaS level, the centre hosts the ALICE Virtual Analysis Facility (VAF) [4], namely a Proof on Demand (PoD) cluster based on μ-cernvm [5] and HTCondor [6]. The cluster is able to automatically adjust its size according to the user requests. The same is true for two elastic batch farms, based on identical tools, dedicated to the local ALICE users and to a team involved in medical imaging studies. Moreover, the Torino Cloud hosts customised single virtual machines tailored to the needs of specific use-cases (i.e nuclear plant and ultra fast silicon detector simulations).

At the IaaS level, the facility provides a number of sandboxed environments where tenants can instantiate private batch farms. This is the case for a number of small local tenants such as a theory group, the Compass and JLab groups and the team who runs the Italian Grid accounting services. For the time being, resources are allocated statically to these tenants.

1.2. The problem to be solved

Strictly speaking, a monitoring system should provide near real-time information on resource utilization and infrastructure health. Moreover, it should be able to raise alarms and trigger countermeasures. There are several tools on the market well suited for these tasks and the monitoring framework described in this paper does not try to be yet another option. Instead, the framework was designed to collect and store information on resource usage for individual tenants. Therefore, one could think of it more like an accounting and billing system. The challenges posed by the computing facility described above are:

- multi-tenancy
- application autoscaling
- heterogeneous data sources

Concerning the latter point, we need to gather information at the IaaS level (from the cloud controller) and at the application level (e.g. Root [7]) and possibly exploit the data coming from other tools already in use to monitor some application on our Cloud (i.e. Zabbix [8]). Our goal is to achieve a uniform and user-friendly monitoring interface as a single entry point to these miscellaneous data. This paper illustrates how we set up a monitoring framework based on the ElasticSearch [9], Logstash [10] and Kibana [11] stack. In addition, we are working to define a model for monitoring-as-a-service based on the same tools. The idea is to generalize the service in place, providing a set of default sensors that each tenant can extend according to their need. Within this scenario, the monitoring framework should be flexible enough to accommodate even more unpredictable data sources. This is one of the reasons for choosing the ElasticSearch ecosystem described in the next section.

2. The ElasticSearch ecosystem

The ElasticSearch ecosystem is composed by ElasticSearch (ES), Logstash and Kibana and it is generally referred to as the 'ELK stack'.

ES is a search and analytics engine built on top of the Apache Lucene information retrieval library. It is document-driven: entries are stored as JSON documents and all fields can be indexed and used in a single query. In our set-up, each application is indexed separately. ES also allows for full-text search on unstructured data, though in our specific case this feature is not fully exploited. Moreover, ES is API driven and can be interfaced with any RESTful API using JSON over http.

Logstash is an open source tool used to collect and parse events and logs to a central service. It can be easily customised via plugins for input, output and data-filters. In order to fit Logstash into our set-up we have developed only a simple plugin to retrieve input data from a MySQL database and a set of configuration files to customise data indexing (one for each application we wished to monitor).

Kibana is the officially supported GUI to display and search ES data. The software could be exposed with any web server, we chose Apache, and no installation is required. Since Kibana does not provide any authentication/authorisation mechanism, these should be implemented at the web server level. Kibana has an easy search syntax to query ES. The creation of custom interactive dashboards can be achieved in few mouse clicks, without any prior GUI programming knowledge. This feature makes it a good candidate for provisioning monitoring-as-a-service to the Cloud tenants. Moreover, Kibana provides a set of useful pre-defined plot types like pies, histograms or trends.

The ELK stack was our first choice for implementing the monitoring infrastructure because it is open source, it provides most of the needed features out of the box and it is relatively easy to configure and use. So far, this solution has proven to be suited for our purposes.

3. Implementation

In the current implementation, the heterogeneous accounting data is fed to different MySQL databases and sent to ElasticSearch via Logstash. We configured a set of Kibana dashboards with pre-defined queries in order to display the relevant information in each case. The set-up is shown in Figure 1 and explained in more detail in the following sub-sections.

3.1. Data preservation: the SQL backend

Monitoring data are stored on a high-availability MySQL server prior to be inserted in the ES engine. This might be seen as a redundant step since data are partially duplicated between the database and ES. Indeed, ES itself can be configured as a fully-fledged (NoSQL) database cluster using its built-in high-availability features. However, in order to ensure flexibility and modularity to our monitoring system, we chose to use ES as a pure search engine with a simple single-node configuration and to delegate data preservation to the MySQL database. With the use of a standard and widely used back-end solution we would be able to move seamlessly away from the ELK stack without losing historical data, should it prove itself not to be the optimal tool. Moreover, the main application hosted at our Cloud besides the Tier-2 sites (i.e. the VAF) relies on Proof [7], which comes with a built-in plugin to send monitoring data to a SQL database at the end of each query. We preferred to avoid writing a custom plugin fitting explicitly our specific monitoring solution.

3.2. Sending data to the database

For the IaaS metering, we developed sensors for the OpenNebula xml-rpc API. The IaaS level information gathered through the API is sent to the MySQL database through an ad-hoc developed RESTful web service. This service is currently also used to store accounting

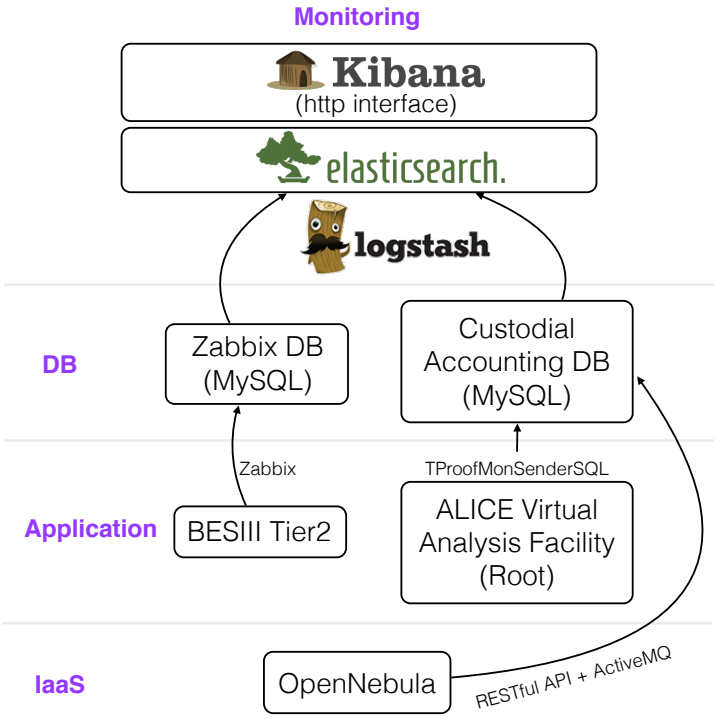

Figure 1. Set-up of the monitoring framework for IaaS and applications at the INFN-Torino private Cloud.

information for the Italian Grid sites. The service relies on the ActiveMQ messaging server to asynchronously and efficiently feed metering records to the database. A similar application has also been developed for billing (see section 4.2). The relevant quantities that we feed to the database are the resource usage per tenant/group in terms of number of virtual machines, number of cores, memory and ephemeral disk space.

Concerning the application level, we use the Proof plugin *TProofMonSenderSQL* to collect accounting data from the interactive analysis facility. At the end of each user query, data are gathered from the worker-nodes to the Proof master and sent to the database with a standard MySQL client/server protocol. In this case, we also monitor some additional observables such as e.g. the number of workers, the datasets analysed or the number of events processed. Moreover, since the BESIII Tier-2 site is already monitored with Zabbix, as a proof of concept we also use the data from the Zabbix database. In this case we can get even more refined information such as resource consumption for specific processes, e.g. the BESIII Offline Software System (BOSS).

4. Use cases

At present, the monitoring framework described in this paper is used to display data coming from the following sources:

- the cloud controller (both from the accounting and billing services)
- the ALICE Virtual Analysis Facility (VAF)
- the BESIII Tier-2

Each use-case has its own Kibana dashboard. Moreover, the Italian Grid accounting infrastructure is also migrating to this model.

Figure 2. Partial view of the Kibana dashboard for the ALICE Analysis Facility at the INFN-Torino computing centre.

4.1. An example: the VAF dashboard

As an example, in Figure 2 we show a snapshot of part of the VAF dashboard. The top-most plot is a *terms* panel, which displays the results of an ES *facet* as a bar chart. By mouse-clicking on a user's bar, only entries for that user are displayed in all other panels.

The second top-most plots are *histogram* panels, showing the number of workers requested by each user and the average CPU efficiency as a function of the query time (similar plots for the memory usage on the master/workers are hidden in the figure). The plots' time-range can be easily selected by dragging with the mouse on the corresponding axis. This operation also sets the desired time-range on all other plots in the dashboard. In order to draw a separate line for

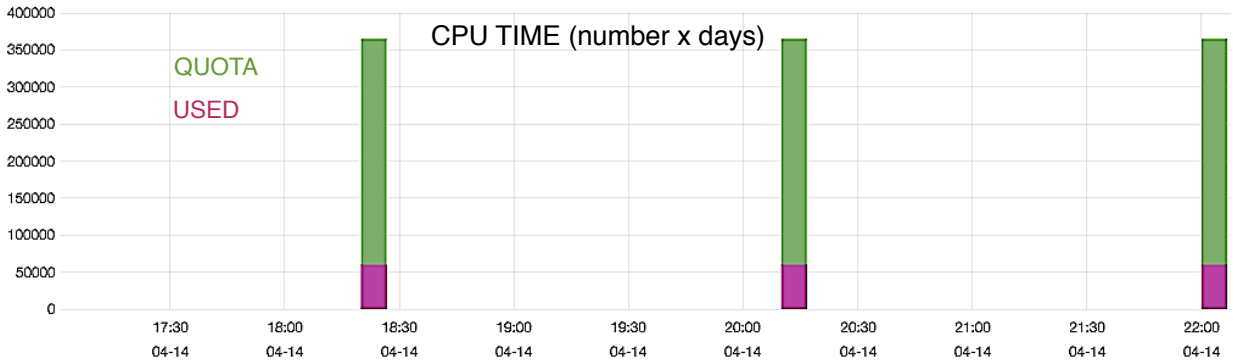

Figure 3. Example plot from the billing system dashboard. The CPU-time (number of CPUs per days) for a specific tenant is plotted as a function of time. The green bars represent the quota, the magenta bars represent the actual consumption.

each user in the plots, we have saved a set of pre-defined ES queries, one for each user, and configured Kibana in order to show them all in the same panel.
In the bottom-most row one can see some information concerning the datasets analysed: the LHC data-taking period flag, the run number and the type of file analysed. These quantities are the result of an ES analysis on a complex string. This is an example of information that would be difficult to obtain with a traditional monitoring system. By selecting e.g. one data-taking period in the bottom-left panel, one automatically gets a list of users who analysed it in the top-most panel.

Similar dynamic dashboards can be set-up in few minutes without any programming, only a good knowledge of the back-end database schema is required.

4.2. A custom billing service

In our specific case, the goal of a billing system is not to present the users with actual 'bills', but rather a way to convince them to free unused resources (when auto-scaling is not implemented). We want to be able to set quotas on quantities like i.e. CPU-time instead of simply the maximum number of CPUs a tenant can allocate. In this way, if the user releases resources whenever they become idle, she will be able to obtain higher computing power in times of peak requests. OpenNebula implements its own accounting system, but for our purposes a finer tuning is needed. Moreover, we need some additional functionality like the possibility to trigger alarms or actions in case quotas are exceeded. Therefore, we developed a custom service with an asynchronous two-daemons logic. The service is written in Python and is called *Mnemos*. The first daemon (publisher) gathers resource usage metrics from the OpenNebula xml-rpc API and publishes them to a message queue (RabbitMQ). The second daemon (receiver) processes the requests and inserts data into the database or sends an e-mail in case quotas are exceeded (but other custom actions can be implemented).

The billing system dashboard displays the integrated CPU-time, memory-time or disk-time in user-defined time ranges. Moreover, it shows the number of disks and the type of OS images used and the evolution of the resource usage compared to the quota. As an example, in Figure 3 we show the CPU-time consumption for a specific user (magenta bars) and the quota (green bars) in units of number of CPUs per days. This plot helps the site administrator and the specific tenant to monitor the resource usage evolution.

At present the billing service is in a pre-production phase and is available to the site administrators only.

4.3. Italian Grid accounting

The Torino private cloud facility also runs the service responsible to collect and visualize the accounting information for the Italian Grid facilities. In the Italian Grid accounting framework, resource usage metrics are collected by APEL [12] sensors at the various sites and delivered to a message system based on the Apache ActiveMQ JMS and Apache Camel for message internal routing. In the current implementation, the accounting data are stored in a MySQL database, processed by Graphite [13] and displayed with Grafana [14] (a similar tool to Kibana that relies on Graphite or InfluxDB as back-end). At the time this paper is being written, the Italian Grid accounting team is designing a new set-up in which the intermediate step of the relational database is dismissed in order to leverage the better horizontal scaling capabilities of NoSQL databases when compared to traditional SQL ones. In this new design accounting data are stored directly in ES and displayed with Kibana. As for what was described for the Torino private Cloud monitoring system, an SQL backend could still be retained for long term archival purposes and data preservation.

5. Summary and outlook

At the INFN-Torino computing centre we have set-up a uniform monitoring interface for both the IaaS and the scientific applications, mostly leveraging off-the-shelf tools. Our choice fell on the ELK stack because it easily allows gathering and digesting heterogeneous data from different sources. The framework relies on a SQL database back-end for data preservation and to ensure flexibility to choose a different monitoring solution if needed. Nevertheless, in our experience at the Torino INFN computing centre, the ELK stack solution proved to be well suited for the task at hand, so that also the Italian Grid accounting service is migrating to this framework.
Our monitoring solutions is used at present to monitor: the cloud controller (both from the accounting service and the billing service), the ALICE Virtual Analysis Facility (VAF) and the BESIII Tier-2. The next step will be to define a model for monitoring-as-a-service, based on the same tools, which the Cloud tenants could easily configure to suit their needs.

Acknowledgements

The present work is partially funded under contract 20108T4XTM of Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale (PRIN), Italy.

References

[1] Moreno-Vozmediano R, Montero R S and Llorente I M 2012 *IEEE Computer* **45** 65-72
[2] Bagnsco S, Berzano D, Brunetti R, Lusso S and Vallero S 2014 *Journal of Physics: Conference Series* **513** 032100
[3] Bagnasco S, Berzano D, Lusso S, Masera M, Vallero S "Managing competing elastic Grid and Cloud scientific computing applications using OpenNebula" *Journal of Physics: Conference Series (in this volume)*
[4] Berzano D, Blomer J, Buncic P, Charalampidis I, Ganis G, Lestaris G and Meusel R 2014 *Journal of Physics: Conference Series* **513** 032007
[5] Berzano D, Blomer J, Buncic P, Charalampidis I, Ganis G, Lestaris G and Meusel R, Nicolaou V 2014 *Journal of Physics: Conference Series* **513** 032009
[6] Thain D, Tannenbaum T, and Livny M *Concurrency and Computation: Practice and Experience* 2005 , **17 No. 2-4** 323-356
[7] Brun R and Rademakers F 1997 *Nuclear Instruments and Methods in Physics Research* A **389** 81-86
[8] Zabbix [`http://www.zabbix.com/`]
[9] ElasticSearch [`http://www.elasticsearch.org`]
[10] Logstash [`http://logstash.net`]
[11] Kibana [`http://www.elasticsearch.org/overview/kibana`]
[12] APEL [`https://wiki.egi.eu/wiki/APEL`] (authentication required)
[13] Graphite [`http://graphite.wikidot.com/`]
[14] Grafana [`http://grafana.org/`]