LHCbDIRAC as Apache Mesos microservices

LHCbDIRAC as Apache Mesos microservices

Christophe Haen, Benjamin Couturier
CERN, 385 route de Meyrin, 1217 Geneva, Switzerland
E-mail: `christophe.haen@cern.ch`

Abstract. The LHCb experiment relies on LHCbDIRAC, an extension of DIRAC, to drive its offline computing. This middleware provides a development framework and a complete set of components for building distributed computing systems. These components are currently installed and run on virtual machines (VM) or bare metal hardware. Due to the increased workload, high availability is becoming more and more important for the LHCbDIRAC services, and the current installation model is showing its limitations.
Apache Mesos is a cluster manager which aims at abstracting heterogeneous physical resources on which various tasks can be distributed thanks to so called "frameworks" The Marathon framework is suitable for long running tasks such as the DIRAC services, while the Chronos framework meets the needs of cron-like tasks like the DIRAC agents. A combination of the service discovery tool Consul together with HAProxy allows to expose the running containers to the outside world while hiding their dynamic placements.
Such an architecture brings a greater flexibility in the deployment of LHCbDirac services, allowing for easier deployment maintenance and scaling of services on demand (e..g LHCbDirac relies on 138 services and 116 agents). Higher reliability is also easier, as clustering is part of the toolset, which allows constraints on the location of the services.
This paper describes the investigations carried out to package the LHCbDIRAC and DIRAC components into Docker containers and orchestrate them using the previously described set of tools.

1. Introduction

The distributed computing of LHCb[1] relies solely on the DIRAC [2] middleware and its extension LHCbDIRAC [3]. It is composed of a client part used by the worker nodes or interactively by the users; and of a core server part exposing all the needed functionality and running the activities. It is by nature a distributed system.

The distributed activity activity grows with the increased amount of data collected by LHCb: more files, more data management operations, more analysis, more jobs, etc. Thus not only does the central infrastructure have to sustain a higher load, but any downtime period has a greater impact.

There are two main types of components composing the core infrastructure. *"Agents"* are periodically executed tasks, which often cannot be duplicated. The LHCb installation counts at the time of writing 146 of them. *"Services"* are stateless frontend servers, relying on databases, accepting and answering requests from clients. They can be duplicated to improve stability and scalability. However, this requires manual interventions to install, start, and declare a new instance to the central configuration. Moreover, this new instance has to be installed on a carefully chosen host: it should not be already overloaded, it has to be - for redundancy purposes - a different host than the other instances, etc.

In this paper, we will see how an Apache Mesos [4] cluster can be used to orchestrate the 134 services of the LHCb installation. We will focus on the various components of this cluster, and give feedback on our first experiences.

2. Orchestrating LHCbDIRAC services

Services are servers queried by clients and agents. They are normally simple frontends to databases. As they are stateless, they can easily be replicated on multiple hosts. In order to make a service instance visible to clients, it needs to be started on a host where LHCbDIRAC has been installed, and its url needs to be declared to the *"Central Configuration System"*(CS).

2.1. Packaging LHCbDIRAC services

The current LHCb installation relies on a set of virtual machines configured in a specific way to install and run the LHCbDIRAC components. While the machine is managed by Puppet [5], the installation, updates and administration of LHCbDIRAC remain a manual operations, and great efforts are put into keeping the different hosts homogeneous. Services and agents are run and supervised by runit [6].

Container technologies like Docker [7] provide a way to package together the application and all the environment it needs, but also to run the application on a platforms different than the host.

Packaging an installation of LHCbDIRAC and all its dependencies into a Docker container appears to be trivial. We are left with one image that can be run on any machine, expose any LHCbDIRAC service, and is managed by the Docker daemon. This image is hosted on the registry integrated to the gitlab [8] service offered at CERN.

2.2. Orchestrating the services

Because we have one Docker image that can act as any LHCbDIRAC service, the next step is to have a list of all the services we want to run, and have them running *somewhere* by distributing this image and running it with dedicated parameters.

Such an orchestration can be organized by Apache Mesos. The role of Mesos is to administrate so-called *"resources"* - which are hosts providing CPU, memory or disk space - and to offer them to *"frameworks"*. The role of a framework is to schedule tasks on the resources offered by Mesos. An example of framework is Marathon [9], which behaves like a distributed systemd: it schedules tasks and ensures that they are running. An other example is Chronos [10], which behaves like a distributed crond, and periodically executes tasks.

This architecture fits perfectly with our goal: we can declare all our voboxes as resources available to Mesos, list in Marathon the list of LHCbDIRAC services we want to run, and these tasks will be executed as docker containers. The current work focuses only on LHCbDIRAC services, and the agents will come in a second step.

Shall a host go down or a Docker container fail, Marathon will take care of restarting the tasks somewhere else.

The Mesos cluster relies on an agent running on every machine either in “slave” mode - this is for the hosts offering their resources - or in “master” mode - this is for the hosts acting as the brain of the cluster and dispatching the resources to the frameworks. For redundancy purposes, we have setup a cluster of three Mesos Masters, allowing for one master machine to go down. Two machines are necessary to ensure a quorum. This cluster uses Apache Zookeeper [11] for synchronization. It is a centralized service for maintaining configuration information and distributed synchronization, allowing the masters to agree on a leader and survive to its failures.

The marathon framework is run on the same machines as the Mesos master, and is setup in a similar cluster of three nodes.

2.3. Discovering and exposing

At this stage, we have all the LHCbDIRAC services that we want running on some hosts. However, in order to be useable, we still need to have all these services exposed to the outside world: we need a list of URLs at which the services are reachable.

Establishing such a list of URLs can be done using Marathon specific service discovery tools like marathon-lb [12]. However, it was felt better to have a tool independent of the Mesos framework used. It allows us to use several frameworks, or to change a framework, without changing all the service discovery layer. This role is filled by Consul [13].
Consul is a service discovery, key/value storage and failure detection tool. It provides a complete list of the resources, the running tasks, their URLs, as well as their health status (see below). All this information can be retrieved using an HTTP API, or used in configuration files using the template rendering mechanism of consul-template [14].
Because we did not want to modify the LHCbDIRAC services to advertise themselves to Consul, the registration of the services is done by an external process - Mesos-consul [15]- which registers all the Mesos tasks inside Consul.

Consul is also performing regular health checks on the resources (the hosts) and on the tasks. By default, when querying Consul, only healthy tasks and resources are returned. In our setup, the health of the Mesos slaves is evaluated with basic Nagios [16] plugins. The health of the LHCbDIRAC services is infered by a ping like functionality implemented in the DIRAC specific protocol DISET. Since Consul can only perform HTTP queries, we needed to implement a simple HTTP server converting HTTP to DISET.

Consul-template is used to generate two configuration files.
The first one is a resource whitelist for Mesos: this file contains a list of slave hosts that are considered healthy. Only whitelisted hosts will be offered to the frameworks to start tasks. Already running tasks on the unhealthy hosts will not be interrupted.
The second is the configuration of HAProxy [17] servers. HAProxy is a load-balancing proxy server. It can be used to translate a single URL into a list of hidden URLs, and switch from one to the other based on various policy (round-robin, load balancing, etc). We can thus declare in the CS only the address of the HAProxy server, and this server will redirect to one of the healthy Docker containers running the LHCbDIRAC service we want. This allows the clients to always talk to the same address without knowing how the infrastructure is setup. Obviously, for redundancy purposes, we have several independent HAProxy servers.

2.4. Operating the system

The previously described setup allows to run all the LHCbDIRAC services we want, and expose them to the clients. However, the administration of such a system is very difficult: because of the dynamic placement of the containers, the monitoring becomes extremely complex, and it is nearly impossible to find the logs of a service. Several additional tools are needed in order to tackle these problems.

Logstash [18] is a server data processing pipeline which aggregates and possibly parses logs from various sources. Hence we can solve our log problems by sending the logs of all the Docker containers running on our Mesos slaves to a Logstash server. This can be done using Filebeat [19]. However, since containers are started and stopped dynamically, the configuration of Filebeat also needs to change dynamically: docker-gen [20] is in charge of updating this configuration.
On the Logstash side, the logs are, at the moment, simply aggregated together in files grouped by LHCbDIRAC service.

It can be very interesting to keep track of various metrics regarding our infrastructure, both for the hosts and the containers: consumed CPU, average load, memory consumption, response time, etc. A simple use case is simply to monitor the system, check that it behaves properly, eventually decide to add an extra Mesos-slave or change the number of instances of a given service. A more advanced use case could be to automate this decision of increasing/decreasing the number of services in order to absorb an unusually heavy load, based on these metrics.
Again due to the distributed and dynamic nature of this system, performing such a metric gathering requires quite a number of tools. The first one is cAdvisor [21], which runs on every Mesos slaves. This software collects by default basic metrics on the host and all the containers running on this host. The issue is that the metrics are collected in real-time and are not stored for later use. Thus we need to collect the metrics from all the hosts, aggregate and store them. Already existing software - like Heapster [22] or Prometheus [23] - propose such functionality but were deemed too complex and unpractical for our use. We decided to write a simple python script running on every host, collecting all the metrics from cAdvisor using its REST interface and sending them to a centralized InfluxDB [24] database. Grafana [25] is then used to plot the data.

Very few other tools were developed in order to ease and automate frequent operations, and hide the technology used for this cluster. For example, a single script is enough to build a new Docker image of an LHCbDIRAC release, publish it and deploy it on the cluster.

The final infrastructure is visible on Fig. 1.

3. First test case: LHCbDIRAC certification

All the setup described previously is being deployed and persisted with Puppet [5]. It currently is in a production ready state. However, in order to test its behavior and the behavior of LHCbDIRAC without perturbating the LHCbDIRAC production system, it was decided to migrate first the LHCbDIRAC certification system to the cluster.
This certification system is an almost independent installation from the production system of LHCbDIRAC on which new versions are being tested before being released. It does not run all the LHCbDIRAC components, but provides sufficient test cases to experiment our Mesos based infrastructure. It will however require separate scaling tests.

The general feedback is very positive. The high availability criteria has been evaluated by perturbating the normal behavior of services or hosts. The recovery time is very short, in the order of a couple of minutes, and the time before hiding a misbehaving resource is in the order of seconds. The recovery time is directly related to the type of failures and to the Marathon's health check frequency. If the container exits, Marathon restarts it immediately and the recov-

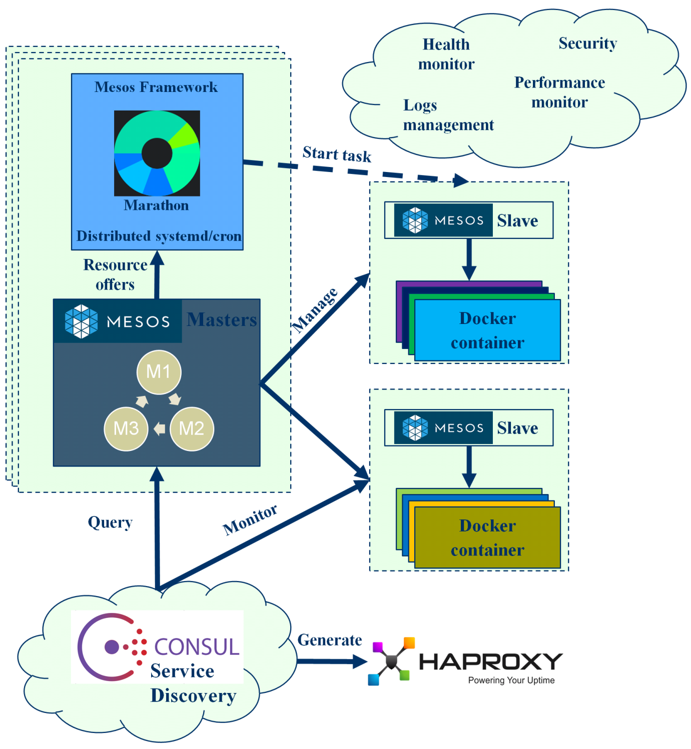

Figure 1. The overview of our Mesos infrastructure.

ery is instantaneous. However, if the service inside the container has a problem without exiting, then Marathon will only restart it after several health checks indicating the problem. Hiding a faulty service is much faster because it is sufficient to restart the HAProxy services without the unhealthy URL, and because the Consul health check frequency is higher than the Marathon one.
The heterogeneity of the machines running the services is not a problem anymore, and Mesos accommodates very well with hosts of very different sizes. The tools developed to deploy new releases was felt handy, and Marathon makes the deployment very smooth by supporting rolling deployments: a new release is deployed in the order of a minute with no service interruption.
It is also very appreciable that besides a few cases, the whole infrastructure is independent of LHCbDIRAC, which means that it could be used to run other unrelated services. Moreover, no changes were needed on the LHCbDIRAC side.

From the pure LHCbDIRAC point of view, only a couple of points are to be noted on the negative side. The first one is that because of the HAProxy setup and the way the network is setup at CERN, the LHCbDIRAC services logs do not contain anymore the original client's IP address. The second aspect that might be problematic is that hotfixes on services are not possible anymore: each change in the code requires a new release of LHCbDIRAC.

From the infrastructure point of view, the difficulties come from the fact that this technology is new to everyone in the team, and will require practice and education because of the multitude of tools that sustain its. Moreover, a misconfiguration or a failure of the master cluster means that the whole LHCbDIRAC infrastructure will be brought down, which was not really a risk before.

4. Conclusion

A production ready Mesos cluster and all the necessary ecosystem around it has been deployed. Its current usage is limited to the certification of new versions of LHCbDIRAC. The first experience gathered is very positive: high availability, scalability and ease of administration are achieved. However, a lot of training to maintain the underlying infrastructure will be needed, some methodologies might need to be adapted, and client traceability is partially lost at the level of the application.

The next step will be to run some components of the LHCbDIRAC production system as Mesos tasks. A second step will be to run LHCbDIRAC agents in the same way as the service. Finally, we will investigate possibilities to isolate and run in parallel different activities, like LHCbDIRAC production and certification, or use the resources in an opportunistic way to perform for example nightly builds or Jenkins tests.

References

[1] LHCb technical proposal, LHCb, *CERN/LHCC, volume 4, 1998, https://cds.cern.ch/record/622031*
[2] DIRAC: a community grid solution, A.Tsaregorodtsev and others, *Journal of Physics Conference Series, volume 119, number 6, 2008*
[3] LHCbDirac: distributed computing in LHCb, F.Stagni and others, *Journal of Physics Conference Series, volume 396, number 3, 2012*
[4] http://mesos.apache.org/
[5] https://puppet.com/
[6] http://smarden.org/runit/
[7] https://www.docker.com/
[8] https://about.gitlab.com/
[9] https://mesosphere.github.io/marathon/
[10] https://mesos.github.io/chronos/
[11] https://zookeeper.apache.org/
[12] https://github.com/mesosphere/marathon-lb
[13] https://www.consul.io/
[14] https://github.com/hashicorp/consul-template
[15] https://github.com/CiscoCloud/mesos-consul
[16] https://www.nagios.org/
[17] http://www.haproxy.org/
[18] https://www.elastic.co/products/logstash
[19] https://www.elastic.co/products/beats/filebeat
[20] https://github.com/jwilder/docker-gen
[21] https://github.com/google/cadvisor
[22] https://github.com/kubernetes/heapster
[23] https://prometheus.io
[24] https://www.influxdata.com/
[25] http://grafana.org/