Integrating multiple scientific computing needs via a Private Cloud infrastructure

S Bagnasco[1], **D Berzano**[1,2,3], **R Brunetti**[1,4], **S Lusso**[1] **and S Vallero**[1,2]

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

E-mail: `svallero@to.infn.it`

Abstract. In a typical scientific computing centre, diverse applications coexist and share a single physical infrastructure. An underlying Private Cloud facility eases the management and maintenance of heterogeneous use cases such as multipurpose or application-specific batch farms, Grid sites catering to different communities, parallel interactive data analysis facilities and others. It allows to dynamically and efficiently allocate resources to any application and to tailor the virtual machines according to the applications' requirements. Furthermore, the maintenance of large deployments of complex and rapidly evolving middleware and application software is eased by the use of virtual images and contextualization techniques; for example, rolling updates can be performed easily and minimizing the downtime. In this contribution we describe the Private Cloud infrastructure at the INFN-Torino Computer Centre, that hosts a full-fledged WLCG Tier-2 site and a dynamically expandable PROOF-based Interactive Analysis Facility for the ALICE experiment at the CERN LHC and several smaller scientific computing applications. The Private Cloud building blocks include the OpenNebula software stack, the GlusterFS filesystem (used in two different configurations for worker- and service-class hypervisors) and the OpenWRT Linux distribution (used for network virtualization). A future integration into a federated higher-level infrastructure is made possible by exposing commonly used APIs like EC2 and by using mainstream contextualization tools like CloudInit.

1. Introduction

The Computer Centre at INFN-Torino is a medium-size infrastructure counting about $1PB$ of disk storage and approximately 1000 job slots. The facility caters to the needs of different applications, the largest ones being a WLCG Tier-2 site for the ALICE experiment at the CERN LHC and a PROOF-based Analysis Facility for the same experiment [1,2]. Moreover, it provides computing resources to the PANDA (FAIR) and Belle-2 (KEK) collaborations and to some local research groups. The centre is planned to grow in the near future by integrating more use cases: a Grid Tier-2 for the BES-III Experiment (IHEP) and a small (Tier-3 size) WLCG site for CMS (LHC).

Nowadays, the diversity of resources and the heterogeneity of applications is steadily increasing with time, however manpower is not. Therefore, it is necessary to consolidate the available resources in order to attain scalability and economies of scale. This is achieved both by centralizing provisioning and by separating application management, that can be delegated to experienced users, from infrastructure management that can not. The Cloud Computing model is a way to raise the level of abstraction and to implement such separation. By converting our computing farm into a Cloud infrastructure, according to the Infrastructure-as-a-Service (IaaS) paradigm, we were successful in reducing the amount of manpower needed by management tasks such as reallocation of resources across the applications and maintenance of large deployments of complex and rapidly evolving middleware and application software. At the same time, the flexibility of the infrastructure was increased, allowing us to provide on-demand computing resources to several smaller applications and short-term requirements. Furthermore, a number of larger-scale Cloud Computing projects are starting in Italy and Europe. A local Cloud designed for interoperability and which follows existing standards and exposes widely-used interfaces, will allow to immediately take part in such activities.

In section 2 we describe the INFN-Torino computing infrastructure. In the following sections we present our recent developments: an EC2 interface to the private Cloud (section 3), a new contextualization strategy based on CloudInit (section 4) and the deployment of virtual farms on-demand (section 5).

2. The infrastructure

Given the aforementioned considerations, and taken into account the need not only to maintain but also to develop the infrastructure with limited manpower and effort, the system was designed according to three guidelines [3]:

- provide a production service to users in a timely fashion
- favor manageability and flexibility over performance
- ensure interoperability with existing and upcoming infrastructures

This translated into a number of design and implementation choices that will be described briefly below. We choose only stable and widely used tools and components, such as OpenNebula [4] and the GlusterFS [5] distributed filesystem, and tried to develop in-house as few pieces as possible. Furthermore, an agile development cycle was adopted, with resources given to the users as soon as possible and features and functionalities introduced only when needed by the applications.

2.1. OpenNebula as cloud controller

The middleware stack of choice is OpenNebula [4], an open-source software suite aimed at deploying, managing and monitoring Infrastructure-as-a-Service clusters widely used both in industry and in the research community. Even though most of the more recent Cloud projects use OpenStack [6], at the time we made the original choice the latter was judged not to be mature enough for a production environment. OpenNebula has many attractive features, for instance it has a modular and easily customizable architecture, mainly based on shell and ruby scripts. It provides most of the needed features and functionalities, with the notable exception (up to version 3.8) of an integrated tool for network-on-demand, which prompted the developments described in section 5.1. Furthermore, OpenNebula exposes, even if with different levels of maturity, industry-standard interfaces such as OCCI and EC2, along with the native OCA. Finally, it includes SunStone: a practical web application for most of the management tasks. The version of OpenNebula used for the work described in this paper is 3.8.

2.2. Two classes of virtual machines

Servers that provide high-level services and workers that provide computational power have different requirements. In order to efficiently accommodate the two classes of VMs, our infrastructure comprises two categories of physical hosts (*Clusters* in OpenNebula terminology):

the *Services Cluster* and the *Worker Clusters.* Server-class virtual machines are instantiated on the Services Cluster, and worker-class ones on different Worker Clusters (each use case can be assigned a separate Worker Cluster).

The services are often critical for the functionality of the whole system, so they need resilient hardware and some form of High Availability may be desirable. Some services require inbound connectivity (at a bare minimum, for granting access to the system), whereas the local disk performance requirements may not be very tight. The hosts in this cluster are built with redundant server-class hardware. They are dual-homed with interfaces in both the public network and the workers private network (private and public IP) with both in- and out-bound connectivity.

Conversely, workers that do data analysis need a high-throughput access to data storage and usually don't need a public IP. The workers are typically dual-twin servers that share a redundant power supply among four motherboards, but have good network connection with the storage servers. The infrastructure currently includes about 7 such hosts.

High-level services can run either on the Workers Cluster head node (e.g. a batch queue manager) or, if they are more critical or need outbound connectivity, on a separate VM on the Services Cluster. Furthermore, the two clusters have been provided with different types of back-end storage in order to optimize the performances and satisfy the above requirements, as described in section 2.3.

2.3. Storage back-end architecture using GlusterFS

The back-end storage system is used by the Cloud infrastructure to distribute the virtual machine images (in our case in raw or qcow2 format) across the physical hosts. It is well distinguished from storage services for data, which are beyond the scope of this paper. In OpenNebula terminology a filesystem for VM images is called a *Datastore.*

GlusterFS [5] was chosen to implement the back-end storage because of its robustness and scalability, which have been proven by several very large deployments in the scientific and high-performance computing. GlusterFS is an open-source filesystem for network-attached storage. It provides horizontal scalability by aggregating storage servers over the network into one large parallel file system with no master host: all synchronizations are peer-to-peer and clients access data directly from the node hosting it. GlusterFS can mimic some RAID functionalities at filesystem level, like striping or mirroring, so that a filesystem can be optimized for performance or reliability.

As mentioned above, the Services and the Workers Clusters have different requirements, most of which are fulfilled by GlusterFS. The filesystems are served by two file-servers equipped with redundant hardware (RAID disks and power supplies) and $10Gbps$ Ethernet interfaces. The virtual images repository resides on a simple Gluster filesystem that currently comprises a single brick, but that can be scaled out by simply adding more bricks should more capacity be needed. All hosts mount this filesystem, from which the images need to be copied by the relevant OpenNebula Transfer Manager to the Datastore on which the running machine images will reside before it boots. Both the Datastores and the copy mechanisms are different for the Services and Workers Clusters (figure 1).

Services should tolerate with little or no interruption the failure of a hardware or software component. To this aim, all hosts in the Services Cluster share a Gluster filesystem on which the image is copied just before the virtual machine boots. The filesystem is replicated across the two file-servers to ensure availability in case of the failure of one, with the Gluster's self-healing feature enabled. Moreover, it is shared across all physical hosts to allow for live migration of virtual machines from one host to another. The latency in the startup of the virtual machine while the image is copied to the shared Datastore is generally deemed tolerable since the redeployment of a service is a rare event. Also the cost in terms of performance of having

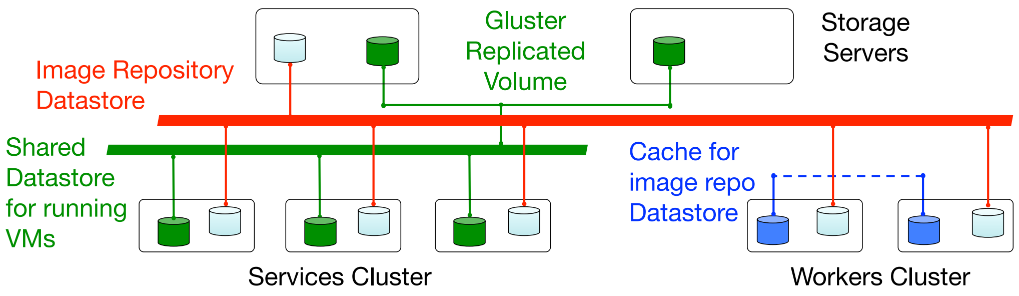
Image Repository Datastore
Gluster Replicated Volume
Storage Servers
Shared Datastore for running VMs
Cache for image repo Datastore
Services Cluster
Workers Cluster

Figure 1. Architecture of the back-end storage. Data storage is not included. The Datastore for running VM instances is shared among the Service Cluster hosts to allow for live-migration and cached for Worker Cluster hosts to reduce the VM start-up time.

a host accessing its running image over the network is justified by a higher stability of the system.

On the opposite, dynamic allocation of resources entails the need for the fast start-up of large numbers of virtual machines, which would be impossible if the images needed to be copied every time. Furthermore, the computing capacity of the site depends heavily on the performance of worker nodes, so network latency towards the running images is to be avoided. Thus, hosts in the Workers Cluster share no filesystem; their running images are cached asynchronously on the local disk of the host by a custom torrent-like tool (built upon scpWave [7] and rsync [8]) so that they are available on the host when a virtual machine needs to boot.

3. An EC2 interface to the private Cloud

OpenNebula provides a web service, called the *OpenNebula EC2 Query* service [9], which allows for launching and managing virtual machines through the *Amazon EC2 Query Interface*. The service is implemented upon two main components:

- the OpenNebula Cloud API (OCA) layer, which exposes the full capabilities of an OpenNebula private Cloud
- the light web framework Sinatra [10]

The OpenNebula distribution includes some basic tools needed to use the EC2 Query service, namely for image upload and registration and for the VM *run*, *describe* and *terminate* operations. At present these tools do not allow the user to send to the VM instance a file containing custom configuration data (*user-data*), which can only be passed in the form of an argument string. This limitation is unfortunate in the case of a complex contextualization, as in our case (see section 4). Therefore, we decided to use the Eucalyptus native implementation of the Amazon clients: the so-called *euca2ools* [11]. These are a set of open-source command-line tools which better suit our needs.

4. Contextualization with CloudInit

To simplify the VM management, we rely on a small number of very generic and simple Operating System images, which are configured (contextualized) at boot time using the standard OpenNebula tools. So far, we have maintained a number of shell scripts which perform the required contextualization actions for both service-class and worker-class VMs: configuring the filesystem, installing and configuring the LRMS, the Grid middleware and so on.

At present, we are developing a contextualization strategy relying on CloudInit [12]: a more advanced tool recently adopted by many Cloud Computing projects. CloudInit is a native

Ubuntu package to handle early initialization of a Cloud instance and it comes as a default on Ubuntu images. It is available for multiple Linux distributions, for instance we use it with CentOS 6 base images. The source code is written in Python and it relies on different known modules, for example the *boto* [13] application framework is used to retrieve the configuration data. The latter can be provided either by the cloud stack (meta-data) or by the user (user-data). Among the features that make CloudInit an appealing tool for our purposes, we mention here customizability and modularity. Different configuration formats are foreseen, such as cloud-config (YAML syntax), shell-scripts, include URLs and MIME archives, among others. Moreover, the user can implement additional features either by adding new modules to the CloudInit source code or via *part-handlers*. Part-handlers are custom modules that can be passed to the VM instance as user-data, without modifying the CloudInit installation on the base image. They can be viewed as plug-ins that allow the user to perform custom contextualization tasks not foreseen by the core package. Moreover, part-handlers allow to handle user-defined configuration formats.

Starting from version 0.73 CloudInit supports the OpenNebula contextualization disk as source of configuration data. Only a minor modification to the original package was needed to allow CloudInit to handle user-data files. We plan to submit this modification for inclusion in the next release.

We are revisiting our contextualization strategy in order to achieve high portability of the instances to any EC2-compatible Cloud, provided that CloudInit is installed on the base images, by embedding all the configuration data in a single self-contained user-data file. Our contextualization procedure is quite complex and involves multiple configuration *tasks*. For example for a Grid worker-node we need to configure the local filesystem and to install and configure CVMFS, the Grid software and the Torque client. Each task might involve different actions like installing packages, writing configuration files or running shell commands. Tasks should be executed in a specific order and we need to be able to exclude/add contextualization blocks according to needs in a simple way. Moreover, task-specific logging would be advisable. For this purposes we pinpointed the use of part-handlers as the most efficient solution. The new contextualization strategy we have developed is the following:

- one part-handler is available for each customization task (to be modified only if new features are needed)
- write the configuration file in simple *cloud-config-like* format, eventually the only thing that the user or the system administrator should modify
- run a script to embed all the information (part-handlers, configuration, SSH keys...) in a single MIME archive
- instance the VM with EC2-compatible query tools (euca2ools) passing the MIME archive as user-data file
- user-data are securely transmitted to the instance via the OpenNebula context disk

During the procedure, each part-handler produces a detailed log-file also including the shell standard output and error streams. A script is run as last contextualization step, which parses these log-files for errors and produces a summary log-file reporting the list of failed handlers. As future development, the information contained in the summary log-file will be used to detach the VM from the LRMS queue in case of fatal errors. At present, this new contextualization strategy is in testing phase.

5. Virtual Farms on-demand

Any generic application (a *virtual farm* in the following) on our infrastructure can be viewed as a number of virtual machines in the Workers Cluster managed by a number of services that run either in a head-node in the Workers Cluster or in VMs in the Services Cluster.

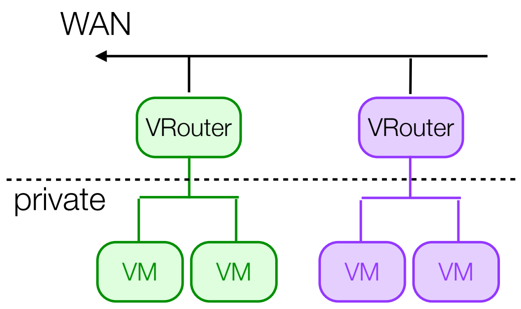

Figure 2. Example deployment of two Virtual Farms. VMs in the Workers Cluster (called *private* in the figure) are isolated from other VMs running on the infrastructure and connected to the external network through the Virtual Router.

5.1. Networking and Virtual Routers

In our infrastructure, all physical hosts share a common private network, while the ones in the Services Cluster also have a second interface to a public network. At the bare minimum, the only service needed by virtual farms is a router to provide network services and access from the outside to a user's virtual Worker Nodes.

We currently deploy Virtual Routers (VRouters) as small virtual machines (one physical CPU, 150 MB memory) based on OpenWRT [14], a lightweight Linux distribution designed for embedded systems. VRouters provide internal and external connectivity to the virtual machines of a given application, alongside with tools for network configuration and management. OpenNebula implements tools for the definition and management of virtual networks at the cloud controller and hypervisor level. Nevertheless, at the time our infrastructure was set-up it lacked a tool to provide virtual farms with an isolated, flexible and secure network environment with access from the outside (now available since version 3.8 under the name of Open vSwitch).

In our implementation each application has its own private fully-featured class-C network, built on top of the physical network (Figure 2). At level 2, these networks are isolated from each other using appropriate *ebtables* [15] rules defined on the hypervisor network bridge, to which the users have no access. At level 3, each private network is managed by a VRouter.

VRouters provide NAT, DHCP, firewalling and port forwarding functionalities, along with a web interface for their configuration and monitoring. They run on the Services Cluster, both for robustness and to allow them to be accessed from the outside. In the standard configuration, the tcp port 22 on the public network of the VRouter is forwarded to tcp port 22 on the private network of one of the nodes in the virtual farm, thus providing SSH access to manage the nodes. Communication between nodes in the same virtual farm is open, whereas they have no way to access the private networks of other clusters.

Moreover, we have implemented on the VRouter the *elastic IP* functionality. It is therefore possible to use an EC2-compatible APIs to bind dinamically a public IP (called *elastic IP* in EC2 terminology) to one of the private VM instances.

5.2. A model for Virtual Farm provisioning

We have also developed a tool for the automated creation of a sandboxed environment within our Private Cloud. The provisioning of an on-demand Virtual Farm consists in the creation of a new OpenNebula user, of an isolated Virtual Network with dedicated VRouter and a single assigned public and elastic IP. The tool takes care of template creation and instantiation for the Virtual Network and the VRouter. The new user has a restrictive quota on the amount of resources at her disposal and can choose among a subset of base images with public permissions. VM configuration is simplified through the definition of Amazon-like *flavours*, which are mapped to OpenNebula templates. The service is now experimentally offered to users, who can access and control their Virtual Farm from within the INFN-Torino network. New VMs can be instantiated and manged with a custom command-line interface based on the euca2ools.

This model for Virtual Farm provisioning allows a more flexible use of the Cloud infrastructure. For example it favors a computing model in which the user can request dinamically new resources to the Cloud through a Workload Management System (WMS). See for instance the recent implementation of the *Elastiq* daemon [16], which was partially tested in our facility.

6. Conclusions and outlook

Currently the Torino Cloud hosts two main applications: a full-fledged WLCG Tier-2 site for the ALICE experiment and a PROOF-based Virtual Analysis Facility for the same experiment. Alongside those, a number of smaller use cases have been successfully tested and more are planned to join in the next future, including a Tier-2 centre for the BES-III experiment. The infrastructure has been in production for nearly two years: the tools of choice proved themselves satisfyingly stable and robust. This, together with the flexibility and manageability of the IaaS model, helped to reduce the management load on the Data Centre staff.

We have enabled the Amazon EC2 interface of OpenNebula to ensure interoperability with other similar infrastructures and we plan to participate in upcoming projects aimed at developing higher-level federated Cloud facilities. In this perspective, we are as well restructuring our contextualization strategy to achieve higher portability (for instance with the use of the main stream tool CloudInit). Moreover, we are now able to provide self-service systems for advanced users to deploy their virtual application farm in a simple way.

There is still room for improvement and we are planning several updates to the system, both to improve the stability of the infrastructure and to provide new functionalities. For example, we are exploring the opportunities given by the CernVM [17] *ecosystem*, to provide higher-level Platform-as-a-Service tools to experiments.

Acknowledgments

The authors want to acknowledge the support by the ALICE and BES-III communities, and in particular by their Italian components. This work is partially funded under contract 20108T4XTM of *Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale* (Italy).

References

[1] Bagnasco S, Berzano D, Lusso S and Masera M 2008 *PoS ACAT* **08** 050
[2] Bagnasco S, Berzano D, Lusso S and Masera M 2010 *J. Phys.: Conf. Ser.* **219** 062033
[3] Bagnasco S, Berzano D, Brunetti R, Lusso S and Vallero S 2013 *Managing a Tier-2 Computer Centre with a Private Cloud Infrastructure* to be published in *J. Phys.: Conf. Ser.*
[4] Moreno-Vozmediano R, Montero R S and Llorente I M 2012 *IEEE Computer* **45** 65-72 .
[5] Gluster [`http://www.gluster.org`]
[6] OpenStack [`http://www.openstack.org`]
[7] scpWave [`https://code.google.com/p/scp-wave`]
[8] rsync [`http://rsync.samba.org`]
[9] EC2 Service Configuration 3.8 [`http://opennebula.org/documentation:archives:rel3.8:ec2qcg`]
[10] Sinatra Web Framework [`http://www.sinatrarb.com`]
[11] Eucalyptus and AWS-compatible Tools [`http://www.eucalyptus.com/eucalyptus-cloud/tools`]
[12] CloudInit [`https://help.ubuntu.com/community/CloudInit`]
[13] boto: A Python interface to Amazon Web Services [`http://docs.pythonboto.org/en/latest/`]
[14] openWRT [`https://openwrt.org`]
[15] ebtables [`http://ebtables.sourceforge.net`]
[16] Berzano D, Blomer J, Buncic P, Charalampidis I, Ganis G, Lestaris G and Meusel R 2013 *PROOF as a Service on the Cloud: a Virtual Analysis Facility based on the CernVM ecosystem* to be published in *J. Phys.: Conf. Ser.*
[17] Buncic P, Aguado Snchez C, Blomer J, Harutyunyan A and Mudrinic M 2011 *The European Physical Journal Plus* **126** 13