VISoR
Virtual Machine Images Management Service for Cloud Infrastructures

João Pereira and Paula Prata
Instituto de Telecomunicações (IT)
Department of Informatics, University of Beira Interior, Covilhã, Portugal

Keywords: Cloud Computing, Virtual Images, Storage Systems.

Abstract: Virtual machine images represent one of the most valuable components in providing a cloud infrastructure, so managing them becomes a key concern in such systems. Each one of the Infrastructure-as-a-Service (IaaS) offers provides its own version of a local image repository. This fact raises problems when managing multiple environments with different IaaS, or when migrating from one to another, because such images management services are mainly designed to interact with its corresponding IaaS and its own storage system. This article presents VISoR, a work-in-progress project, whose main goal is to achieve an agnostic service for managing virtual machine images among different cloud offers.

1 INTRODUCTION

Cloud computing brought the capability to provide access to a wide range of machines as virtual machine (VM) instances, whose number varies depending on the amount of required resources. Cloud computing platforms are no longer confined to local clusters of machines, they are growing, and today they vary from machines within a single data center to networks of data centers spread all around the world. These facts led to the expansion of VMs uses, as they are one of the key features of the emerging Cloud Computing paradigm (Buyya et al., 2009). Managing large amounts of virtual images being deployed over multiple distributed machines could become an exponential bottleneck, so the way to manage them is very important (Begnum, 2006). Providing the capability to manage virtual images through an organized, reliable and scalable service is very important, as the number of such systems continues to grow, it is necessary to produce tools capable of simplifying the work of those dealing with them.

Current cloud Infrastructure-as-a-Service (IaaS) frameworks provide their own version of an image repository, although they present limitations and incompatibilities between them, as they do not allow storing images and the needed information about them (metadata) in a compatible way between different IaaS. Their implementation suffers from some natural limitations, such as the storage systems compatibility and its design that is tied to their own needs and requirements. Facing the cloud computing paradigm, sometimes there is the need to manage a couple of different IaaS in the same infrastructure, bringing hard integration problems when managing available virtual images. Users of such frameworks should not be constrained in choosing the right tools for the job and switch back and forth between them as they need, maintaining a centralized agnostic image management service repository.

Having said that, our work differs from such approaches as we present VISoR (which stands for Virtual Images Service Repository), a service providing a generic purpose and multi-cloud repository, completely open-source and designed from bottom to top not to fit in a specific platform or purpose, but designed to overreach sharing and interoperability limitations among different IaaS and storage systems. This provides the ability to enable the storing and managing of virtual images among different clouds, in the same generic repository. Also, placing the cloud as first target audience is not a limitation for other use cases, given the agnostic design concerns, the service can be successfully applied to a wide range of image management environments.

The remainder of this paper is organized as follows. In Section 2 we discuss the background of IaaS offers, its image management solutions and storage systems. In section 3 we present the service features. In section 4 we outline the security risks and the ap-

Pereira J. and Prata P..
VISoR - Virtual Machine Images Management Service for Cloud Infrastructures.
DOI: 10.5220/0003903104010406
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 401-406
ISBN: 978-989-8565-05-1

proach to contain them. The architecture of the system, its components and its description is detailed in Section 5. The status of the system implementation and future work are described in Section 6 and finally, in Section 7 we present the paper's conclusions.

2 BACKGROUND

Currently there are many IaaS offers, among which we highlight the Amazon Web Services (AWS), Eucalyptus, OpenStack, OpenNebula and Nimbus, as the primary IaaS targets for VISoR. Following we conduct an overview about these IaaS and their image services and storage systems.

Amazon AWS (Amazon, 2006) is a commercial platform that provides the capability to use the cloud infrastructure through their web services. It provides a wide set of public images shared by users and also let them store their owns, in its storage system called Simple Storage Service (S3) and in its Elastic Block Storage (EBS).

Eucalyptus (Nurmi et al., 2009) (Eucalyptus, 2009) is a platform for building private and hybrid clouds, with one open source version and another enterprise edition. Eucalyptus provides it own distributed storage system, called Walrus, which is used to store the images. It also implements an Amazon S3 compatible interface.

OpenStack (Rackspace, 2010) is an open source project jointly launched by Rackspace and NASA, providing software for building private and public clouds. It integrates a distributed storage system, the OpenStack Swift. It also provides its own image repository service, called Glance, which integrates with Swift and optionally with Amazon S3.

OpenNebula (NASA, 2008) (Sotomayor et al., 2009) is an open source project for building private, public and hybrid clouds. It also provides an image repository. Its storage system relies on the POSIX standard and supports the Network File System (NFS) and Logical Volume Manager (LVM) too.

Nimbus (Chicago, 2009) is a platform that combines a set of tools to provide clouds for scientific use cases. It also uses its own storage system for the cloud, called Cumulus (Bresnahan et al., 2011). Images are stored in Cumulus and Nimbus also provides a compatible Amazon S3 interface.

Finally we would also like to mention the FutureGrid platform (von Laszewski et al., 2011), an interesting testbed for scientific projects installed across multiple High-Performance Computing (HPC) resources distributed across several USA states, which also integrates an image service in its architecture for the FutureGrid platform, towards its inside grids and clouds management.

Unlike the solutions presented here, VISoR targets different requirements. Users can expect VISoR to be a free, completely open-source tool. They can download, customize and use it inside their own environment (unlike solutions like FutureGrid tools, which are integrated into the project and are intended to operate inside it, through the on-line platform portal). VISoR aims to tackle the heterogeneity of existing IaaS clouds and other use cases which need to rely on an image repository service, making it possible to sit it in the middle of these frameworks. Also, VISoR is not intended to replace a VM instantiation tool or a cloud management utility, like FutureGrid RAIN (von Laszewski et al., 2011) and commercial services like RightScale (RightScale, 2006). It is intended to be a generic centralized image catalogue and repository, that can be used to manage images and expose them to the endpoint cloud hosts.

3 VISOR FEATURES

The VISoR service should be designed considering a set of features, which we have defined based on its aims and purposes, leading to the following presented set in which we rely for designing the system.

- *Open Source.* If anyone wants to contribute or just learn how it works, it can be done freely. The development process should be community-driven.
- *Multi-Interface.* The system should provide access through more than one interface, being them a Representational State Transfer (REST) interface, an API and a command-line interface (CLI).
- *Modular.* It should be designed and implemented in a modular way, so all subsystems are isolated and can be easily customized and extended.
- *Extensible.* The service should provide a way to add new extensions to it, or rely on it to build new tools through the defined set of interfaces.
- *Flexible.* It should be possible to install such a service by requiring minimal setup operations and strategically close to the needed resources.
- *Scalable.* The service should be designed with scalability in mind, it must adapt to high load requirements.
- *Multi-Format.* The service should provide compatibility with multiple virtual machine images disk and container formats.

- *Cross-Infrastructure.* The system must provide an unified multi-infrastructure service repository, sitting in the middle of different IaaS, including Eucalyptus, OpenStack, OpenNebula and Nimbus.
- *Multi-Storage.* It should provide compatibility with popular cloud storage systems, by relying on a seamless abstraction API layer.

4 SECURITY

Given the growth of cloud computing implementations, there are some risks and security issues that arise (Heiser and Nicolett, 2008). These security issues and concerns basically address the need to guarantee the initial image integrity and the need to provide a secure way to let users share their images and retrieve them safely. Sharing virtual images implies safety concerns, because many of these are made by third party providers and users share them with many other users. Existing approaches to cloud security fail in assessing virtual images security (Wei et al., 2009), as most of this techniques are not applicable to virtual images in a dormant state.

It is necessary to take into account all the risks involved in a VM image sharing activity, mainly addressing the needs of administrators, publishers and retrievers of the virtual images present in the image repository (Wei et al., 2009). For the administrator it is a risk to have an infected image hosted in the repository, because when it gets executed it can infect other files. The biggest concern for the publisher is to not publish images with personal data, like user accounts and passwords that should not be made public. This is common when users launch the virtual image for some initialization configurations without taking care to do not expose any sensitive information. The biggest risk for the retriever is that he can retrieve some maliciously crafted image, committed to the repository by some publisher.

Also, it is extremely important to consider security threats related with the VM images life cycle, which can compromise the overall cloud security. Vaquero et al. exposes a detailed analysis of such threats (Vaquero et al., 2011). From there, we want to address the stages that VISoR should pay attention to, being them the images transportation and storage. By taking into account these issues and the proposed security approach by Wei in (Wei et al., 2009), we address VISoR risk contention approach with three security components, being them the Access Control, Tracking and Maintenance. We will describe each one of these components in Section 5.

5 ARCHITECTURE

Based on the design features and security concerns, we have defined the VISoR architecture (Figure 1).

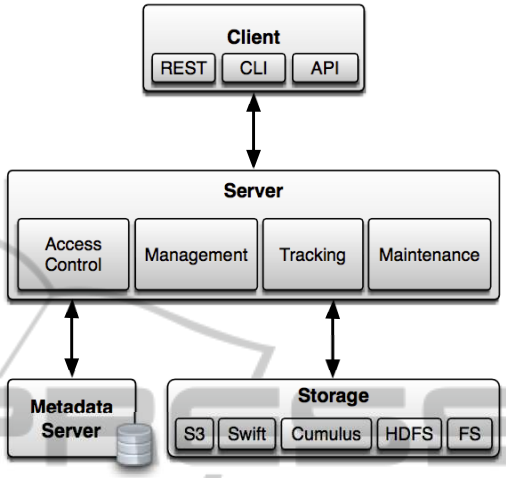

Figure 1: The VISoR system architecture.

All the system stack is implemented in a *modular* way, so given this isolation and separation of components, the system becomes *flexible* and builds a solid foundation to provide a *scalable* service. The system relies on a client-server architecture, with four main components, being them the clients, the main server, the metadata server and the storage layer.

5.1 Client

This component is responsible for providing access to the service. We address the *multi-interface* feature by implementing three different interfaces to the service, exposing it to a variety of target communities, such as administrators, developers and end users.

REST. Implementing the REST architecture (Fielding, 2000), the RESTful web services are the main interface to the system. By relying in the HTTP protocol, we manage to expose the service in a way that makes it possible to easily integrate it with other web-based services.

CLI. We also provide a command-line interface, which provides users with the capability to administrate and use the full system stack, being the most important tool to interact with the system.

API. Finally, we provide the API exposed by the clients class, which provides the capability to interact with the system through programming.

By exposing these interfaces we are making the service *extensible*, as new tools, clients and extensions to the system are easy to develop and integrate.

5.2 Server

The Server is the core of the system, managing all the processes and providing access to the resources, being them the images files and its metadata. It contains four major components, being them the Access Control, Tracking, Management and Maintenance.

5.2.1 Access Control

This component is responsible for ensuring that publisher's security risks are minimized, by providing a framework capable of controlling the sharing of virtual images, by leveraging in access permissions. At storing time, virtual images may be set as *public* or *private*, defining whether they must be seen by everyone or only by its owner or those with granted access permissions to that image. The owner of the image is responsible for granting and revoking access permissions to other users and groups, with two different types of permissions, the *checkout* and *commit* permissions. A checkout permission means that users or groups of users granted with this permission are allowed to retrieve and use some image. A commit permission implies the existence of a checkout permission and allows users or groups of users to commit some modified existing virtual image.

5.2.2 Management

The management component integrates two main management concerns, being them the users and images management. This is the main control actor over users actions and images in the VISoR repository.

User Management. When interacting with the system, users are managed by roles and group memberships, so the first step from users is to authenticate themselves providing valid credentials. The system maintains basic information about each user. The administration of user accounts is delegated to the service administrator, using the provided CLI tools.

Image Management. This element consists in managing associated metadata about a given image and the physical image itself. When managing images metadata, the server handles incoming requests from clients and redirect them to the metadata server, handling the response back again to clients. When handling the images files upload and download, the server is responsible for managing the images storage, making all the process as transparent as possible for users.

5.2.3 Tracking

This component is responsible for minimizing administrator's and retriever's security risks. It keeps tracking and recording operations through the service API, creating a full history of virtual images life cycle since they were published in the repository. Also, this component will provide statistical data, useful for tracking the service usage, which can help automating common tasks, such as reports about images that are never used and can be deleted. Having statistical data about the repository usage and images life cycle can greatly improve administrator's capabilities.

5.2.4 Maintenance

The maintenance component is responsible for assuring the safety and integrity of virtual images without harming the system performance. It aims to provide the capability to scan virtual images looking for malicious software and similar threats, delegating this task to a scanner through the repository service. This may be invoked as the administrator prefers, automatically running the scan based on some event or manually.

5.3 Storage

The storage component represents a *multi-storage* abstraction layer that provides seamless integration with multiple storage systems. This will include integration with Amazon S3, OpenStack Swift, Cumulus, as well as the local file system and the Apache Hadoop Distributed File System (HDFS) (Foundation, 2007) (Shvachko et al., 2010). Also, given the system *modularity* it is possible to easily extend it with other storage systems compatibility.

The server is responsible for handling clients upload and download requests and then interacts with the storage layer to accomplish that requests, acting as a bridge between storage systems and endpoint machines. Users should describe through configuration options, were can VISoR find a specific cloud storage, this is, its address and listening port, or user credentials if its an on-line service like Amazon S3. With an unified interface to multiple storage systems it is simpler to achieve a *cross-infrastructure* storage service, abstracting the complexity of such heterogeneous systems.

5.4 Metadata Server

The metadata server handles all the information about virtual images stored in the VISoR repository. The metadata is stored in the database and then can be retrieved from there, with support to all the CRUD (Create, Read, Update and Delete) operations through the REST interface defined in Table 1. When issuing GET requests, it is possible to provide query parameters, filtering and ordering returned results, which are

Table 1: The REST API exposed by the metadata server.

Method	Path	Operation
GET	/images	Return brief metadata of all public images.
GET	/images/detail	Return detailed metadata of all public images.
GET	/images/(id)	Return metadata of the given image.
PUT	/images/(id)	Update metadata of the given image.
POST	/images	Add metadata of a new image.
DELETE	/images/(id)	Remove metadata of the given image.

Table 2: Image's metadata fields, data types, predefined values and access permissions.

Field	Type	Predefined Values	Permission
id	Integer		Read-Only
uri	String		Read-Only
name	String		Read-Write
architecture	String	i386, x86_64	Read-Write
status	String	locked, uploading, error, available	Read-Only
size	String		Read-Only
access	String	public, private	Read-Write
uploaded_at	Date		Read-Only
updated_at	Date		Read-Only
accessed_at	Date		Read-Only
access_count	Long		Read-Only
owner	String		Read-Only
checksum	String		Read-Only
type	String	none, kernel, ramdisk, amazon, eucalyptus, openstack, opennebula, nimbus	Read-Write
kernel	Integer		Read-Write
ramdisk	Integer		Read-Write
disk	String	none, iso, vhd, vdi, vmdk, ami, aki, ari	Read-Write
store	String	s3, swift, cumulus, hdfs, fs	Read-Write
others	Key/Value		Read-Write

passed through the response's body. Also, when issuing PUT and POST requests, the metadata should be provided through the body of the request in JSON (Crockford, 2006) format key/value pairs. The metadata server describes images based on the recommended fields listed in Table 2. To achieve *multi-format* support, so we provide to users the ability to use other image properties not listed in Table 2, by providing additional key/value pairs attributes or by ignoring some of the optional attributes if they are not useful. When a new image is being registered, the service defines a unique identifier, the *id* and the uniform resource identifier, the *uri*, for it. Users should provide the *name*, the *architecture* of the operating system and the *access* permission. Optionally, it is possible to provide the *store*, which indicates in which service they want to store the image. The *status* of the image is defined by the system, where the 'locked' state means that the image was already registered without an upload, the 'uploading' status informs that the image is being uploaded, the 'error' status reports that there was an error on the image upload and finally, the 'available' status defines that the image is already available. Also defined by the system are the *owner*, which identifies the image publisher, the image *size*, its *checksum*, as well as the *uploaded_at* and *updated_at* timestamps. The system also maintains other two tracking fields, being them the *accessed_at* timestamp and the *access_count* which counts the number of accesses. Optionally, users can define other fields, which are the image *disk* format, *type*, as well as if the image has some associated *kernel* or *ramdisk* image already stored in the repository, providing its *id* for these fields.

6 STATUS AND FUTURE WORK

Currently we have already implemented the metadata server with a MongoDB and an MySQL database backend. We decided to adopt a NoSQL system, beyond the classic SQL approach with MySQL, as these

modern tools target the use case where our service fits, providing good performance and scheme free capability (Cattell, 2011). Additionally we have implemented the main server, following an event-driven approach, using a framework based on the Reactor Pattern (Schmidt, 1995). We are currently working on the storage options expansion, which already includes the S3 and FS stores, and in the API security approach implementation.

We expect to expose more insights on API internals and detail our main server implementation with the described security approach in future work. Outside of the article scope are the technologies choices and the analysis of the overall system structure and performance, which we expect to conduct soon in future work too. We also want to proceed with a more in-depth analysis of the integration of VISoR with the described target IaaS and custom modification that can surge from such analysis.

7 CONCLUSIONS

In this paper we have presented our work towards VISoR, a virtual image service that is being developed to address the need to keep a centralized generic repository, while managing multiple environments with different IaaS. VISoR will support multiple IaaS and storage systems and for its design we focused on security and performance towards an efficient and scalable virtual images management service. Given this, we expect it to be the agnostic service that will act as a holder between multiple heterogeneous systems that rely on virtual images management capabilities. The source can be accessed, as documentation and further information, through the project web page at http://www.cvisor.org.

REFERENCES

Amazon. (2006), Amazon web services (aws) [online]. Available: http://aws.amazon.com/ [Accessed on January 2012].

Begnum, K. (2006). Managing large networks of virtual machines. In *Proceedings of the 20th Large Installation System Administration Conf.*, pages 205–214.

Bresnahan, J., LaBissoniere, D., Freeman, T., and Keahey, K. (2011). Cumulus: an open source storage cloud for science. In *Proceedings of the 2nd int'l workshop on Scientific cloud computing*, pages 25–32. ACM.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25:599–616.

Cattell, R. (2011). Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27.

Chicago, U. (2009), Nimbus, cloud computing for science [online]. Available: http://nimbusproject.org [Accessed on January 2012].

Crockford, D. (2006), Rfc 4627, the application/json media type for javascript object notation (json) [online]. Available: http://www.ietf.org/rfc/rfc4627 [Accessed on January 2012].

Eucalyptus. (2009), Eucalyptus infrastructure as a service [online]. Available: http://eucalyptus.com [Accessed on January 2012].

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis.

Foundation, A. S. (2007), Hadoop distributed file system [online]. Available: http://hadoop.apache.org/hdfs/ [Accessed on January 2012].

Heiser, J. and Nicolett, M. (2008). Assessing the security risks of cloud computing. *Gartner Report*.

NASA. (2008), Opennebula, the open source toolkit for cloud computing [online]. Available: http://opennebula.org [Accessed on January 2012].

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, Washington, DC, USA. IEEE.

Rackspace, N. (2010), Openstack, open source software for private and public clouds [online]. Available: http://openstack.org [Accessed on January 2012].

RightScale, I. (2006), Cloud computing management platform [online]. Available: http://www.rightscale.com [Accessed on January 2012].

Schmidt, D. C. (1995). *Reactor: an object behavioral pattern for concurrent event demultiplexing and event handler dispatching*, pages 529–545. ACM, USA.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE.

Sotomayor, B., Montero, R., Llorente, I., and Foster, I. (2009). Virtual infrastructure management in private and hybrid clouds. *Internet Computing, IEEE*, 13(5):14–22.

Vaquero, L. M., Rodero-Merino, L., and Morán, D. (2011). Locking the sky: a survey on iaas cloud security. *Computing*, 91:93–118.

von Laszewski, G., Diaz, J., Wang, F., Younge, A. J., Kulshrestha, A., and Fox, G. (2011). Towards generic futuregrid image management. In *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, pages 15:1–15:2, NY, USA. ACM.

Wei, J., Zhang, X., Ammons, G., Bala, V., and Ning, P. (2009). Managing security of virtual machine images in a cloud environment. In *Proceedings of the ACM workshop on Cloud computing security*, CCSW '09, pages 91–96, NY, USA. ACM.