Using Fogbow to federate private clouds

Abmar Barros[1], Francisco Brasileiro[1], Giovanni Farias[1], Francisco Germano[1], Marcos Nóbrega[1], Ana C. Ribeiro[1], Igor Silva[1], Leticia Teixeira[1]

[1]Laboratório de Sistemas Distribuídos – Universidade Federal de Campina Grande
Av. Aprígio Veloso, 882 - Bloco CO, Bodocongó, CEP 58429-900
Campina Grande - PB, Brasil,

abmar@lsd.ufcg.edu.br, fubica@computacao.ufcg.edu.br, {giovanni, chicog, marcosancj, anaribeiro, igorvcs, leticia}@lsd.ufcg.edu.br

Abstract. *Infrastructure-as-a-service (IaaS) has become the major choice for the provisioning of computing infrastructures. One of the advantages explored by providers is the possibility of reducing the overall capacity needed to support the aggregated demand of all customers. The federation of private IaaS that intend to work together for their mutual benefit has been advocated as a way to increase the infrastructure utilization, and, as a consequence, to increase the efficiency of the federation members, especially for small and medium-sized IaaS providers. This paper presents the Fogbow middleware, an approach based on a peer-to-peer architecture that aims at creating a cloud federation from totally distributed and independently IaaS deployments. Fogbow allows local cloud users to use surplus resources coming from other members of the federation, while being minimally intrusive regarding existing IaaS deployments.*

1. Introduction

Nowadays, infrastructure-as-a-service (IaaS) is increasingly becoming the preferred choice for the provisioning of computing infrastructures. From the customer's perspective, computing resources encapsulated in Virtual Machines (VMs) can be rapidly provisioned, and later released, offering a theoretically unlimited elasticity in the way service consumption increases and decreases with time. Moreover, customers incur in costs only during the time that the VMs are being used. After they are released, it is up to the provider to deal with the costs of the physical resources, installations, operational team, etc. required to support the service. On the other hand, from the provider's perspective, IaaS brings a number of advantages that allow for more efficient management of the installed capacity, and consequently a reduced total cost of ownership.

According to the NIST Definition of Cloud Computing [Mell and Grance 2011], IaaS provision may follow different deployment models. In particular, there is a clear distinction between the private deployment model, and the public one. A private cloud is provisioned for exclusive use by a single organization, offering service to multiple users that access them typically using the organization’s intranet. On the other hand, public cloud is provisioned for open use by the general public, with the service being accessed through the Internet.

One of the main advantages that are explored by both public and private IaaS deployments is the possibility of reducing the overall capacity needed to support the aggregated demand of all customers through the consolidation of servers. This is because it is unlikely that all applications will experience peak demand at the same time. In addition, most public IaaS, as well as large private IaaS, also benefit from the economies of scale that arise when the number of customers served is big, and consequently a sizable infrastructure is needed.

Small and medium private IaaS deployments can still take some advantage of the benefits of server consolidation, but they normally experience a smaller utilization of their physical resources, which represents costs that are not amortized by the utility associated with the execution of applications. Federation of private IaaS, by which a number of IaaS providers work together for their mutual benefit, has been advocated as a way to increase the infrastructure utilization, and as a consequence increase the efficiency of the federation members. In a federation the utilization can be increased mainly because the larger, and potentially more diverse, aggregated workload is likely to require less resources than the aggregated capacity of the IaaS providers, were they to work isolated. Peak loads experienced by a member can be served by other members that are currently experiencing a low utilization of their resources, thus allowing the local capacity of each member to be reduced. A smaller capacity leads to a higher utilization.

However, implementing a federation of IaaS uncovers a set of challenges previously inexistent [Celesti et al. 2012]. The first of these challenges is the management of the federation membership, since the whole federation might not be known a priori, and it can be rather heterogeneous and dynamic. The second challenge is match-making: once the members of the federation are discovered, both the cloud that requests resources and the other available clouds must agree on a policy to request and offer resources, such as CPU, storage, networking, images, etc. The last challenge is distributed authentication and authorization of both federation and local users, and amongst federation members, i.e. administrative domains running private clouds.

The Fogbow middleware addresses those challenges by providing an additional layer for federation atop each local private IaaS that wants to join a fogbow federation. It uses designed-to-federate, internet-friendly technologies like XMPP and it is flexible enough to deal with a wide range of cloud technologies, since it provides a plugin framework for authentication (for users and members), authorization, image storage management and compute services.

The rest of the paper is structured as follows. In Section II, we enumerate the requirements for cloud federation. In Section III, we describe the architecture and the implementation details of the Fogbow middleware. In Section IV, we describe some deployment cases of Fogbow, before concluding in Section V.

2. Federation requirements

A multi-domain and federated IaaS deployment comes with a set of requirements of its own. We enumerate the most outstanding and challenging ones.

2.1. Membership management

Physically distributed private clouds must be discoverable, and the discovery mechanism should be flexible enough to deal with components coming up or down unexpectedly. It should also be internet-friendly, in terms of coping with firewalls and NATs. It is also desirable for this membership management mechanism to be replicated in order to avoid being a single point of failure.

2.2. Resource matching

Federating cloud resources - e.g. images, volumes, flavours, etc - implies a great challenge in cross-cloud requests. Even if one is able to tell that, for instance, an image in Cloud A is exactly the same as another image in Cloud B, it is not possible to rely on a global identifier for a single resource.

Thus, there is a need for a federation-wide mechanism, capable of matching global resource identifiers to the ones valid in each cloud. Such a mechanism would allow for federated requests that can be understood in various cloud deployments, even when different middleware are adopted.

2.3. Authentication and authorization

In a private cloud, Authentication and Authorization take place in a single level - user database is usually centralized, and the local cloud will rely on that, not only to tell if users are really themselves, but also to tell which operations these users are able to perform on cloud resources.

In the context of cloud federation, the whole user database is considered to be spread over federation members, i.e. each administrative domain, therefore, Authentication and Authorization must take place in three distinct levels:

a) *In the federation layer*: to tell if the user is able to use the cloud federation itself, which means s/he can access cloud resources in the requesting private cloud or in other clouds in the federation; b) *In the local cloud layer*: to tell if the user can access resources in the requesting private cloud as a local user that exists in its user database; and c) *Among clouds*: to tell if a cloud is able to accept requests from, or to create requests targeting other clouds in the same federation.

Besides implementing the aforementioned levels of Authentication and Authorization, a middleware for IaaS federation must be aware of the fact that a wide range of Identity Services are actively used in different cloud stack deployments, e.g. LDAP [Howes et al. 2003], Virtual Organization Management Service [Alfieri et al. 2004], Keystone [Openstack Keystone 2014], etc. Therefore, such a middleware must be compatible with the most used Identity technologies in order to be widely adopted - rationale that goes along with the discussion on intrusiveness that follows.

2.4. Intrusiveness

The inclusion of a private IaaS in a cloud federation must be performed with as little intrusiveness as possible. That means it should preserve all security policies, users, network topologies, etc. of the existing cloud. This requirement also applies to the stack technology used by the underlying cloud, therefore the solution must be compatible with wide-spread middleware such as OpenStack [Sefraoui et al. 2012], OpenNebula [Moreno-Vozmediano et al. 2012], CloudStack [CloudStack 2015], etc.

2.5. External Accessibility

In contrast with public clouds, private clouds may provide their services with limited connectivity to the external world. When these clouds are federated, this is no more the case, since it is unfeasible to provide Virtual Private Networks connecting all members of the federation. Thus, an additional requirement for federated clouds is their ability to provide cross-site connectivity, so that instances created in a target cloud are accessible from the requesting cloud.

3. Architecture and implementation

In order to meet the requirements outlined in the previous section, we have designed and implemented the Fogbow middleware, available at http://www.fogbowcloud.org/.

The Fogbow approach is based on a peer-to-peer architecture that aims to create a federation from totally distributed and independent IaaS deployments, whose main objective is to allow local cloud users to use surplus resources coming from other members of the federation.

As depicted in Figure 1, a deployment of a Fogbow federation comprises at least two components: the Fogbow Manager (FM), and the Fogbow Rendezvous (FR). The FM, deployed on top of an existing private cloud, works as a proxy that receives requests issued to the local cloud and redirects them to the local underlying cloud, or passes them on to other federation members, when the local cloud is exhausted. The FR acts as a discovery service to FMs. In the following we discuss how Fogbow addresses each of the previously mentioned requirements.

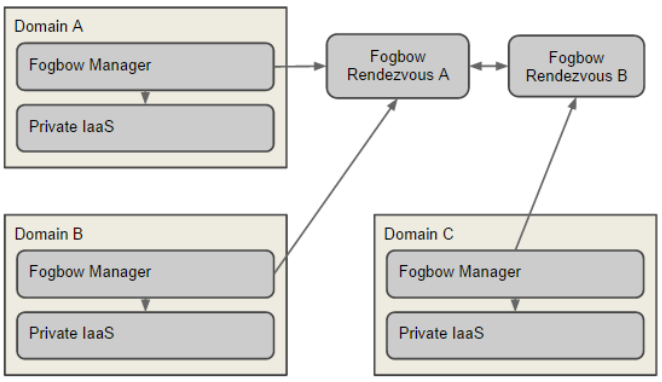

Figure 1. Deployment case of a Fogbow federation

3.1. Membership management

The FR is responsible for implementing membership management. The FR keeps track of all active FMs that report to it. We use a push-style failure detector, which means that FMs will send periodic heartbeats to the FR, which, in turn, will consider an FM to be inactive if it does not hear from it within a given timeout.

The FR is implemented as an XMPP component [Saint-Andre 2005], therefore it relies on a message bus provided by an XMPP server that binds on a single port for all server-to-server communication, making it easy to overcome NATs and firewalls.

It can also be deployed in a replicated and synchronized manner, which allows for higher availability, server offload, and can improve resource utilization when the wide area network gets partitioned.

3.2. Resource matching

The FM provides a resource mapping mechanism that associates global to local resource identifiers that should be configured in a per FM basis. Therefore, global identifiers are loosely coupled among federation members.

Images, though, are handled in a particular manner. In order to check if an image exists in the local cloud, the FM tries to match the global identifier with a "friendly name", whose meaning depends on the middleware being used - in most cases this is the image name reported by the cloud dashboard.

In addition to that, on every request considered for scheduling, the FM will call an ImageStoragePlugin instance, that may take an action, e.g. download the requested image from an appliance database, when the image is not found locally.

3.3. Authentication and authorization

As stated in the previous section, Authentication and Authorization should happen in three different levels and should support a wide range of Identity Providers' technologies.

To comply with that, the FM implements a Plugin framework for the Federation and Local levels. For every Identity Service API - e.g. VOMS, OpenStack's Keystone - there should be an Authentication plugin. This plugin takes a set of credentials and implements the interaction with the corresponding API, in order to tell if these credentials are valid or not. In that sense, a request for instance at the FM will hold two access tokens, one for each level - although only the federation one is mandatory.

After the user is authenticated, an Authorization plugin is consulted to tell what that authenticated user is able to perform within the federation. Notice that the Authorization plugin only makes sense in the Federation Layer, since the cloud stack itself will authorize the local user based on its own policies.

The third level of Authentication and Authorization takes places between two federation members during remote requests. FMs will exchange X.509 certificates plus the publishable snippet of the requesting user's access token – e.g. the VOMS proxy certificate without the private key. We also use the Plugin framework here to tell if/what remote members can do (request or donate) based on the certificate exchanged and on the requester credentials. It is also important to highlight that the channel itself is encrypted using the members' X.509 certificates in all member-to-member communication through XMPP's TLS encryption [Saint-Andre 2011].

3.4. Intrusiveness

To make the deployment of the FM not intrusive in relation to the underlying private cloud, we cannot rely on changes in local security policies, privacy settings, specific resources' deployment or on any technology replacement.

To comply with such a requirement, the FM extends the aforementioned Plugin framework to cope with any interaction with the underlying private cloud. In other words, the business logic of the Manager is agnostic to the cloud stack on top of which it runs.

Fogbow currently provides plugins for OpenStack (OCCI and Nova API) and OpenNebula (rOCCI server and ONE), but new plugins can be easily implemented and integrated into the FM.

The Manager has to serve as a proxy for the underlying cloud. If we were using a Federated Identity mechanism we could simply use that to authenticate any federation user in the local cloud, however that would imply in changes on how the underlying cloud authenticate local users.

In order to circumvent that, the FM requires the creation of a special user in the local cloud, which we call the federation user. This user will act as a proxy for any remote request, resulting in a minimal change to the current deployment of the private cloud.

3.5. External accessibility

In order to provide connectivity to isolated VMs in the federation, we have created the Fogbow Reverse Tunneling Service (FRT).

The FRT is an SSH Server fork that only provides the reverse tunneling feature plus an HTTP layer for token creation and validation. When an instance is about to be requested by the FM in the local cloud, it generates a random token that will be injected into the instance via cloud-init [Cloud Init 2014]. Together with the token, the FM will also pass along a script that asks the FRT for a new port, providing the token it was given, and, ultimately, creates the reverse tunnel.

Given that the tunnel is created, every time the FM needs to provide the SSH address of that instance, it will reach the FRT for the port associated to the aforementioned token.

The FRT is supposed to be deployed in a machine with a public IP and with a range of ports allowed through the domain's firewall.

4. Deployment cases

Fogbow has been deployed in various scenarios using different Identity and Compute service technologies, which, so far, has validated the Plugin mechanism being adopted. The steps to install Fogbow in such environments are described in the following.

4.1. Pure Openstack

For an FM to run atop of a regular OpenStack deployment, which commonly uses Nova and Keystone, we have developed the OpenStack NovaV2 Compute Plugin and the

Keystone Identity Plugin. In this case, both Federation and Local Identity layers will use the same plugin, therefore only local users will be able to use the federation resources.

4.2. Openstack's OCCI API plus VOMS

Although this deployment is quite similar to the one just described, it uses different Identity plugins in the Federation and Local layers, as depicted in Figure 3. This deployment allows for users authenticated via a VOMS to use the federation, even if they are not local. Thus, we have created both the OpenStack OCCI Compute Plugin and the VOMS Identity Plugin.

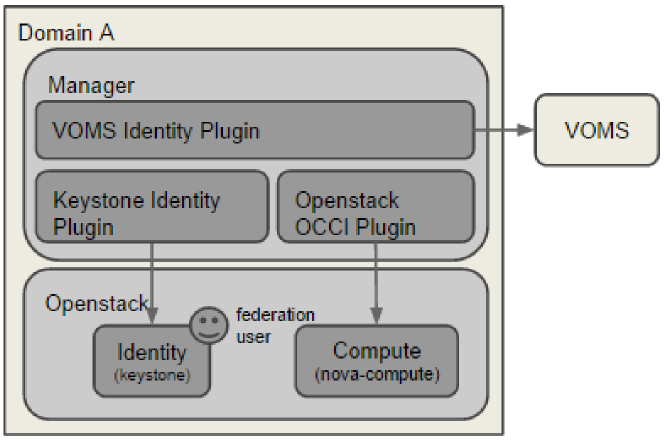

Figure 3. Fogbow Manager's deployment using OpenStack and VOMS

5. Conclusion

Fogbow, through its policies, allows for the quick identification of new private clouds, resource listing of each cluster, resource matching and resolution of identity and authorization conflicts, enabling the pooling of resources from various clouds in a secure and rather non-intrusive manner. The design decisions regarding the use of a Plugin framework allowed greater flexibility in the technologies with which the FM is able to interact, e.g.: different cloud platforms, which could be validated in our Deployment Cases. In the context of the EUBrazilCC project (http://eubrazilcloudconnect.eu/), we currently run a federated cloud that comprises private clouds managed by both OpenStack and OpenNebula. Fogbow is also being considered to federate private clouds at SERPRO, as part of the CICN project.

6. Pointers and demonstration

All source code related to Fogbow is hosted at https://github.com/fogbow. Each component is hosted in a project of its own, under that URL. Documentation on installation, configuration, architecture, design and more can be found at http://www.fogbowcloud.org/.

The demonstration we intend to make at the SBRC will comprise most of the features of Fogbow. We will use the Fogbow Dashboard hosted at http://ufcg-servers.dashboard.fogbowcloud.org/ in order to create requests at the Fogbow Manager that runs atop the Openstack cloud at the LSD-UFCG. This FM is configured to perform authentication at the federation layer against the EUBrazilCC VO (eubrazilcc-voms.i3m.upv.es), thus we will need to use a VOMS proxy certificate as the federation token, and a Keystone token at the local level. We will issue requests for a large number of instances so that we can exhaust the local quota, and start creating virtual machines at all members of the federation. Once requests are fulfilled, we will connect to the instances via SSH using a private key paired with the public key used at requesting time, and the IPs and ports provided by the Fogbow Reverse Tunneling component. After connecting to a few instances provided by different members, we will terminate both instances and requests we just created, so that we depict the whole life-cycle of a Fogbow request.

Acknowledgement

Fogbow is partially funded by CNPq/Brazil and the European Commission through the EUBrazilCC project, and by FINEP/Brazil through the CICN project.

References

Mell, P. and Grance, T. (2011) “The NIST definition of cloud computing.”.

Celesti, A., Tusa, F. and Villari, M. (2012) “Toward Cloud Federation: Concepts and Challenges.”

Howes, T. A., Smith, M. C. and Good, S. G. (2003) “Understanding and deploying LDAP directory services”. Addison-Wesley Longman Publishing Co., Inc.

Alfieri, R., Cecchini, R., Ciaschini, V., dell’Agnelo, L., Frohner, A., Gianoli, A., Lõrentey, K. and Sparato, F. (2004) "VOMS, an authorization system for virtual organizations." Grid computing. Springer Berlin Heidelberg.

(2014) “OpenStack Keystone”. Available at http://docs.openstack.org/developer/keystone/. Accessed at September, 24th.

Sefraoui, O., Aissaoui, M., and Eleuldj. M. (2012) “OpenStack: toward an open-source solution for cloud computing.” International Journal of Computer Applications 55.3, pp 38-42.

Moreno-Vozmediano, R., Montero, R.S. and Llorente, I.M. (2012) “IaaS Cloud Architecture: From Virtualized Datacenters to Federated Cloud Infrastructures”, IEEE Computer, vol. 45, pp. 65-72.

CloudStack, A. (2015) “Understanding CloudStack”. Available at <http://cloudstack.apache.org/software.html>. Accessed at March, 13th.

Saint-Andre, P. (2005) "XEP-0114: Jabber Component Protocol." XMPP Standards Foundation.

Saint-Andre. P. (2011) "Extensible Messaging and Presence Protocol (XMPP): Address Format".

(2014) “Cloud Init”. Available at <http://cloudinit.readthedocs.org/>. Accessed at September, 24th.