Cloud Federations in Contrail

Emanuele Carlini[1,2], Massimo Coppola[1], Patrizio Dazzi[1], Laura Ricci[3], and Giacomo Righetti[1,3]

[1] ISTI-CNR, Pisa, Italy
[2] IMT Lucca, Lucca, Italy
[3] Dept. Computer Science, Univ. of Pisa, Italy

Abstract. Cloud computing infrastructures support dynamical and flexible access to computational, network and storage resources. To date, several disjoint industrial and academic technologies provide infrastructure level access to Clouds. Especially for industrial platforms, the evolution of de-facto standards goes together with worries about user lock-in to a platform. The Contrail project [6] proposes a federated and integrated approach to Clouds. In this work we present and motivate the architecture of Contrail federations. Contrail's goal is to minimize the burden on the user and increase the efficiency in using Cloud platforms by performing both a vertical and a horizontal integration. To this end, Contrail federations play a key role, allowing users to exploit resources belonging to different cloud providers, regardless of the kind of technology of the providers and with a homogeneous, secure interface. Vertical integration is achieved by developing both the Infrastructure- and the Platform-as-a-Service levels within the project. A third key point is the adoption of a fully open-source approach toward technology and standards. Beside supporting user authentication and applications deployment, Contrail federations aim at providing extended SLA management functionalities, by integrating the SLA management approach of SLA@SOI project in the federation architecture.

1 Introduction

Cloud computing is a computing model aimed at providing resources as services according to a pay-per-use paradigm. The provided resources differ in type and level of abstraction. While the basic candidates are computational power and storage, resources can be provided to users from the infrastructure level (IaaS, e.g. virtual machines, virtual storage), the platform level (PaaS, e.g. programming libraries, application templates and components) and the software level (SaaS, e.g. complete applications like Google Documents). Almost all current IT behemoths offer their own cloud computing solution: Amazon [1], Google [7], Microsoft [2] and many others. Unfortunately, this leads to some issues. Each cloud provider forces its users to operate according to specific models, e.g. communication protocols and virtualization formats. This is known as *vendor-lock in* and from users perspective it leads to a major disadvantage. Namely, even

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 159–168, 2012.

if users running their applications in a certain cloud find better providers to exploit, the burden of cloud switching may lead to cut this solution off.

As the resources offered by each provider belong to different levels of abstraction, standardization decreases. Typically providers offer somewhat interchangeable services only at a low level of abstraction, while higher level services provide a low or no degree of personalization. Any user willing to exploit resources belonging to different levels while keeping the option open to move across provider boundaries, needs to cope with the burden of adapting the applications. In order to address these issues, in recent times several standards like Open Cloud Computing Interface [8] and Open Virtualization Format [9] have been proposed to provide cooperation among different cloud solutions. Each of these standards covers, however, only specific aspects and portions of the Cloud management stack, and no standard is universally adopted by cloud providers, so far.

The Contrail approach to cloud federation tackles the issue of integration among heterogeneous clouds. Important factors are the open-source choice and the collaborations with other projects involved in Cloud research. On top of this, as a main line of research, Contrail aims at a two-way integration: (i) a *vertical* integration, which provides a unified platform for the different kind of resources and (ii) a *horizontal* integration that abstracts the interaction models of different cloud providers. In this work we describe the general architecture of Contrail cloud federations, thereby focusing on horizontal integration and on the federation services which are essential in allowing vertical integration.

Section 2 explores the motivations and commitments of a cloud federations, with a particular stress on users ID managements, Service Level Agreement (SLA) integration and application execution. Section 3 gives an overview of the Contrail federation architecture and how appliances are deployed. Section 4 presents a selection of current research work on cloud federations. Finally Section 5 concludes the paper.

2 Contrail Federations: Motivations and Commitments

From a practical point of view, a federation can be considered as a bridge linking cloud users and cloud providers. As the role of the federation goes beyond mere interface adaptation, federation services act as mediators between users and providers. From a user's perspective, the components and the services supporting the federation (we will refer to them as *federation-support* in the rest of the paper) act as a broker for the resources owned by providers participating to the federation. As each provider has its own, potentially specific, mechanisms and policies for managing resources, the goal of the federation-support is to provide translation and mapping mechanisms for matching user needs and exploiting federated cloud providers.

The pay-per-use concept relies on the existence of a formally agreed SLA between the user and the provider(s), and the ability to monitor, enforce and account service use and QoS. Besides the resource selection functionalities across multiple providers and the consistent management of resources, a Contrail federation coordinates the SLA support of cloud providers. Indeed, as cloud providers

have their own SLA management mechanisms which is useful to exploit, the role of the federation is to setup, coordinate and enforce a global SLA, eventually negotiating SLAs with providers on behalf of the users. This leads to the possibility of a *vertical SLA management*, allowing to define PaaS services which are provider-invariant and also supporting interactions between public and private cloud providers. The federation-support has to monitor the application in order to verify that SLA is fulfilled by providers, and to react to SLA violations.

Federation-Level User ID Management. A federation has to provide users with mechanisms to specify their preferences about cloud providers and resources. Federation-support manages the user identities for accessing to cloud providers, and users have also to be informed about their actual and historical usage of resources (accounting, billing).

The task of protecting personal user-related information, like user identities, is only the first stone when building security into a federated cloud approach. One of the problems related with federations is to save the users from the burden of authenticating with resources belonging to different cloud providers, especially as many of the actions on the resources have to be automated and performed 24/7. The federation should exploit proper mechanisms and credentials, in accordance with both user preferences and the authentication support of the providers.

Appliance Deployment. An application is composed by: (i) a set of appliances and (ii) a SLA description that provides the user requirements on a per appliance basis. With the term appliance we identify a set of VM images strictly cooperating to realize an application fundamental block (e.g. a pool of web servers, or a firewall and back-end database combination). The federation has to map those appliances in the federation resources according to both user requirements and preferences, as specified in the application SLA and possibly as constrained by user identity and related resource policies.

In order to effectively set up and enact the application-resources mapping the federation needs static (geographic location, cost-models, installed software) and dynamic information regarding cloud providers and their resources. It is also relevant to record past history of providers with respect to SLA violations, in order to evaluate their reliability.

SLA Coordination. SLAs negotiated by the federation-support and users define a set of functional and non-functional requirements that have to be addressed and enforced in order to properly execute user's appliances. In the Contrail project we assume that every cloud provider belonging to the federation has proper mechanisms able to deal with the SLA descriptions regarding the appliances it has to execute.

Most of those mechanisms and the underlying formalism are inherited from the SLA@SOI [12] [13] project. In particular, the SLA management yielded by Contrail cloud providers is based on three main entities: (i) SLA, (ii) SLA Template and (iii) SLA Manager. The SLA is a structured description of user and appliance requirements, which is derived by a SLA Template. A SLA template

provides a customizable base that can be exploited in order to derive specific SLAs. A SLA@SOI SLA Manager monitors a running appliance and reacts in case the appliance misbehaves with respect to its associated SLA. The actions enacted by a SLA Manager include intra-cloud appliance migration, appliance reconfiguration and network set-up. The federation-support should intervene to coordinate the involved SLA Managers, in case one or more SLA Managers were unable to enforce the SLA of one or more appliances.

Non-functional Requirements. In addition to the functional commitments, the federation-support has also to address specific non-functional requirements. They are mainly related with platform scalability, flexibility and security. Scalability and flexibility are key performance aspects for a federation, dealing with a relevant amount of resources and users, and can be regarded as a non-functional requirement of the federation design. These considerations influence the design of the federation-support, presented in Section 3.

Other classical non-functional goals of application execution, once the application gets deployed on a Cloud in accordance with an SLA, become functional requirements for the federation-support. Besides performance, one of the major concerns of the federation-support is security. Security plays an important role in the federation as well as in the whole Contrail project, since it directly affects the acceptance with respect to possible customers. The federation-support must offer a secure environment in which users execute applications, and store their data and personal information. In this context protection is two-fold: first, both the users data and their applications should be protected from unauthorized accesses and modifications. For instance, the federation should protect users from affecting each other, from snooping on each other one's jobs, or data. Second, the federation shall protect itself from malicious or erratic applications.

3 Federation Architecture

The federation acts as a bridge between users and cloud providers. The *federation-support* offers to users, in an uniform fashion, resources belonging to different cloud providers. A Contrail federation can exploit two kind providers, those based on the Contrail cloud infrastructure and the ones based on other public and commercial infrastructures. As shown in Figure 1 the federation architecture is composed of three layers. Every layer is in turn composed by modules, where each module addresses a well defined commitment.

The top-most layer, called *interface*, gives a view on the federation and provides proper ways to interact with the federation. The interface gathers requests from users as well as from other Contrail components that rely on the federation functionality and facilities. The interface layer includes a CLI and HTTP interface, from which is possible to access to REST services. The mid layer, called *core*, contains modules that fulfill the functional (e.g. application life-cycle management) and non-functional (e.g. security) requirements of the federation. The bottom layer, called *adapters*, contains the modules that retrieve information

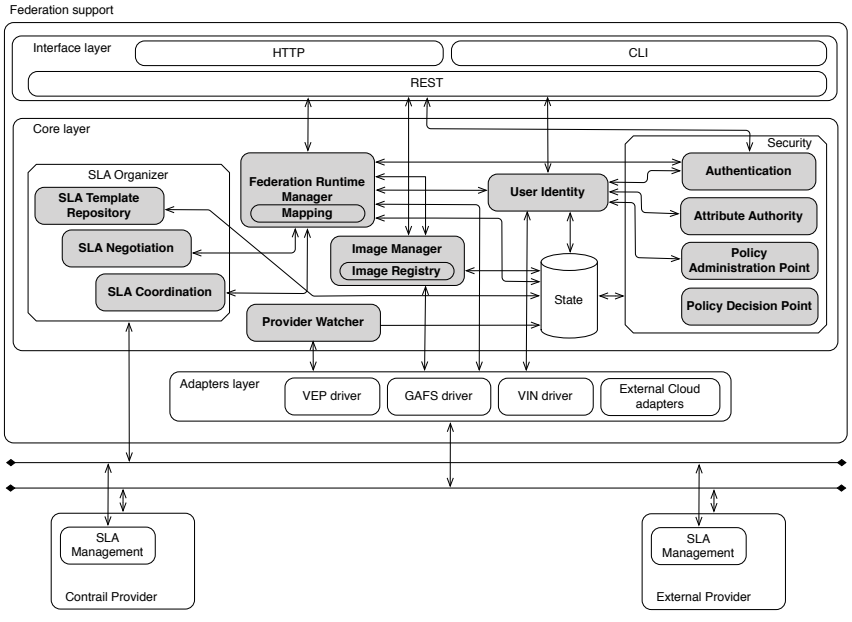

Fig. 1. Federation-support architecture

and operate on different cloud providers. This layer provides also a unified interface that possibly copes with heterogeneity of providers. A detailed description of mechanisms provided by the interface layer are beyond the scope of this paper, therefore they are not presented. In the next sections we present a detailed description of the modules belonging to the core and adapters layer.

3.1 Core Layer

The core layer contains the modules that implement the business logic of the federation. These modules solve the three main commitments demanded to the federation-support, namely identity management, application deployment and SLA coordination. These modules are in turn supported in their activities by additional auxiliary modules. In the following of this chapter we present in detail the modules that implement the business logic of the federation as well as the state module, which is in charge of the federation state management. We refer to the auxiliary modules whenever it is necessary.

User Identity. The federation-support provides to each user a federation-level account. By using this account the user can have access to all the resources owned by the federated cloud providers. In order to interact with different providers, the federation-level user account is bound with different local providers identities. The *user identity* module is in charge of realizing the aforementioned bind. The actual connection between the module and the providers is done through the Adapter layer (discussed later).

The access to resources is managed in a seamless way, i.e., once authenticated to a Contrail federation, users should not be prompted again to access federated Cloud providers (e.g. single sign-on). The local cloud identities are stored in the state module. In order to guarantee isolation and data integrity, of the user-related data, the federation-support takes advantages of the mechanisms and policies provided by the authentication and authorization modules.

Federation Runtime Manager. One of the core task of the federation is application deployment. This is not a trivial task, since the user will expect the federation-support to find proper mappings between submitted appliances and clouds belonging to the federation.

In order to devise a good mapping onto the compute, storage and network services of the federation, the *federation runtime manager* (FRM) uses a set of heuristics that consider different aspects, such as to minimize economical cost and to maximize performance levels. This actual task and the heuristics are implemented by the *mapping* component, while the FRM is in charge of the orchestration between the mapping component, the SLA management system and the drivers layer. In particular, the FRM is responsible of the application life cycle management. The FRM gathers information to cover these aspects from the State module. The information is both static and dynamic. *Static* information is mainly related with general properties about cloud providers; it includes, for instance, their geographic location, their resource- and cost-models as well as the installed software. *Dynamic* information is related to the status of cloud provider resources, as well as to cloud providers as autonomous entities forming the federation. It is the kind of information obtained by monitoring resource availability either on a per cloud-provider basis or by recording and analysing the past history of each provider with respect to violated SLA. This information can be exploited to evaluate their reliability.

Image Manager. From the user's perspective the images can be managed in two ways: they can be packed inside an OVF archive or referenced within the OVF files by using URI. The task of deciding what is the best storage solution is carried out by the *Image Manager*. It associates metadata to the images and decides when is necessary to copy an image or when indirection can be exploited. The actual metadata are kept inside the State module; however an *Image Registry* is introduced to decouple federation code from being modified whenever State module is modified moving from the centralized scenario to the distributed one.

Provider Watcher. This component is responsible for the State update, upon receiving monitoring information from the Adapter layer. It decouples the State from doing this task leading to a more cohesive architectural design.

SLA Organizer. The *SLA Organizer* is a collection of modules related to SLA management at the Federation level, which is achieved by leveraging and coordinating SLA agreements stipulated with the federated resource providers. These modules are:

- **SLA Coordination.** The *SLA Coordination* module checks that running appliances comply with the user provided and agreed SLA, and plans corrective actions as needed. Upon being notified a violation, the SLA Coordination module logs the event, evaluates the current status of all related appliances and providers, and tries to define a reconfiguration plan for the application which compensates the violation. The SLA coordination module undertakes actions that may involve either a single cloud provider, or, in more complex scenarios, multiple providers and the federation-support.
- **SLA Negotiation.** The *SLA Negotiation* is responsible of the negotiation protocols with providers. Its main purpose is to decouple the protocols for SLA (re)negotiation from the core Business logic of the Federation Runtime manager.
- **SLA Template Repository.** This module gathers and stores SLA templates published by the providers. The Federation SLA Template Repository acts primarily as a cache of the Provider's SLA Template Registries, supporting scalable SLA-based queries and user interface template selection within the federation. The repository can as well holds federation-specific SLA templates not bound to any provider.

The State Module. The *state* module collects, aggregates and provides information exploited by the federation-support. Information is subject to diverse constraints in terms of frequency, atomicity and consistency of updates, thus different architectural solutions may be needed to fulfil scalability and reliability.

The involved issues become relevant when deploying the federation in a highly distributed scenario, with many federation access points. The specific purpose of the State module is to keep the core business logic of the federation unaware of the distribution aspects, only exposing the choice among different classes of data services. Each kind of information and the related constraints can be addressed by specific design patterns, whose use we will investigate further during the project. The federation modules require the state to manage different kinds of information.

- The User Identity module, security modules and the Federation Runtime Manager need read or write capability to access/manage user identity information and system-wide preferences;
- The Provider Watcher needs write capability to keep an up-to-date view of available resources belonging to federated and external cloud providers;
- The Provider Watcher module and the SLA Organizer gather a characterization of cloud providers, such as their geographic location, SLA templates, cost models and peculiar features;
- The Federation Runtime Manager accesses meta-data about providers (reputation, availability) and running appliances (including associated SLA).

Clearly, such an approach requires a proper distributed communication mechanism to support the flow of information among the state modules. To this end, we plan to integrate different distributed communication patterns. The decoupling of distributed communication within the State module is allowed since

most tasks requiring atomicity are performed at the provider level (e.g. resource pre-reservation and commitment), thus simplifying the implementation of the federation state.

3.2 Adapters Layer

This layer contains the modules that enable the access to infrastructural services for both Contrail cloud and External clouds. They are referred respectively as *internal* and *external* adapters. These components enrich the typical cloud infrastructural services with additional features targeting federations.

Internal Adapters. The components of the internal adapters module are: (i) the Virtual Infrastructure Network (VIN) which provides network, (ii) the Global Autonomous File System (GAFS) which provides storage and (iii) the Virtual Execution Platform (VEP) which provides computing power.

The VIN provides APIs to define a virtual network among multiple virtual machines, both intra- and inter-provider. Also the VIN provides API to know the QoS level of an inter-provider link, and if it is possible, the proper mechanisms to enforce a given QoS. The GAFS provides shared data space, with the possibility for an application spanning in multiple providers to access a virtual volume. Finally, the VEP provides the proper OCCI interfaces to enable access to provider resources. Its APIs include mechanisms for reservation and configuration of resources, and starting and monitoring of machines.

External Adapters. In order to extend Contrail's functionality onto external clouds and at the same time to maintain modularity, the federation has been designed in a provider-agnostic fashion. This means that each module of the federation-support do not have any knowledge if it is issuing command to a Contrail cloud or to an external cloud. Commands toward external cloud are issued via a type-specific adapter, which translates requests from the federation support into requests that are understood by the provider. This task is assigned to the *External Provider* module of the federation Model. This module does not contain any driver supporting the VIN, GAFS, or VEP. Instead, an External-Provider exploits the interface exposed by the public cloud.

4 Related Work

In this section we briefly describe state-of-the-art solutions dealing with federations of clouds. InterCloud [3] is a federated cloud computing environment that addresses the issue of provisioning application services in a scalable computing environment, achieving QoS under variable workload, resource and network conditions. InterCloud performs application scheduling, resource allocation and migration of workloads. The authors implemented it on top of CloudSim [4], a framework to model and simulate cloud computing infrastructures and services. Their solution is built on three concepts: Cloud coordinators, Cloud Brokers and

Cloud Exchange. A *Cloud Coordinator* (CC) exports the services provided by a cloud to the federation by implementing basic functionalities for resource management such as scheduling, allocation, workload and performance models. This actor also supports virtualization, dynamic sensing/monitoring, discovery, and application composition. CCs periodically update the *Cloud Exchange* (CEx) with their availability, pricing, and SLAs policies. This information repository aggregates information supplied by CCs in order to support the Cloud Brokers activity. The *Cloud Broker* identifies suitable cloud service providers published on the CEx, negotiating with CCs for an allocation of resources that meets QoS needs of users. Since Contrail's brokers interact with each others, instead of having single-user context brokering, our federation-support can exploit information of what other users are requesting. This means a more reactive scenario in which better reservation strategies can be adopted. In addition, to the best of our knowledge, Contrail adopts a more cloud independent approach. Indeed, in our solution the federation plays a more central role, incorporating most of the functionalities described into InterCloud's CC.

The authors of [11] describe the architecture of an open federated cloud computing platform in the context of the Reservoir [10] project. In the Reservoir model, each resource provider is an autonomous entity with its own business goals. A provider can choose the providers with which to federate. There is a clear separation between the functional roles of *service providers* and *resource providers*. Service providers are the entities that matches the user needs by finding resources that their application need. However, service providers do not own the resources. They lease such resources from resource providers. Reservoir succeeds in defining a reference architecture capable of dealing with common IaaS requirements and even new ones, such as service-orientation and separation between infrastructure and services. Nevertheless, Contrail tries to built upon its results, adding vertical integration of IaaS and PaaS service models.

In [5] the authors propose *Dynamic Cloud Collaboration* (DCC), an approach for setting up highly dynamic cloud federations. The cloud provider (CP) that wants to setup a federation assumes the role of the *primary cloud provider* (pCP), whereas the federated cloud providers are called *collaborating CPs*. To federate new collaborating CPs, adding their resource/services to a DCC platform, an approval of other providers based on their own policies is needed. Users request services published on the service catalogue of the pCP. Then the pCP finds suitable partners based on the business objectives, and stipulate a contract with specific SLAs requirements for each partner involved. If after a distributed negotiation an agreement among all partners is reached a new dynamic cloud became operational.

5 Conclusions

This position paper presents the cloud federations of the Contrail project. A Contrail cloud federation supports the horizontal integration of different cloud providers by easing the task of distributing applications among different cloud

providers as well as managing them in order to fulfil negotiated SLAs. In order to achieve this goal, a Contrail federation manages users identities, coordinates application deployment and the SLA management conducted by single cloud providers. In this paper we presented these commitments in detail. Then, we described the architecture of the cloud federation-support by showing the main software modules it is composed of, and describing the relationships among those modules. This description is an outline for the future work that has to be conducted in order to realize Contrail cloud federations.

Acknowledgment. The authors acknowledge the support of Project FP7-257438, Contrail: Open Computing Infrastructures for Elastic Services (2010-2013).

References

1. Amazon Elastic Compute Cloud, `http://aws.amazon.com/ec2/`
2. Windows Azure, `http://www.microsoft.com/windowsazure/`
3. Buyya, R., Ranjan, R., Calheiros, R.N.: InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In: Hsu, C.-H., Yang, L.T., Park, J.H., Yeo, S.-S. (eds.) ICA3PP 2010. LNCS, vol. 6081, pp. 13–31. Springer, Heidelberg (2010)
4. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 41(1), 23–50 (2011)
5. Celesti, A., Tusa, F., Villari, M., Puliafito, A.: How to enhance cloud architectures to enable cross-federation. In: 3rd International Conference on Cloud Computing, pp. 337–345. IEEE (2010)
6. Contrail project, `http://www.contrail-project.eu`
7. Google App Engine, `http://code.google.com/appengine/`
8. Open Cloud Computing Interface, `http://occi-wg.org/`
9. The Open OVF project, `http://www.dmtf.org/standards/ovf`
10. Reservoir project, `http://www.reservoir-fp7.eu`
11. Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I.M., Montero, R., Wolfsthal, Y., Elmroth, E., Caceres, J.: The reservoir model and architecture for open federated cloud computing. IBM Journal of Research and Development 53(4), 4 (2010)
12. The SLA@SOI project, `http://sla-at-soi.eu/`
13. Theilmann, W., Yahyapour, R., Butler, J.: Multi-level sla management for service-oriented infrastructures. In: Towards a Service-Based Internet, pp. 324–335 (2008)