Automating the Deployment of Multi-Cloud Applications in Federated Cloud Environments

Alfonso Panarello
DICIEAMA, University of Messina
Messina, Italy
apanarello@unime.it

Uwe Breitenbücher
IAAS, University of Stuttgart
Stuttgart, Germany
breitenbuecher@iaas.uni-stuttgart.de

Frank Leymann
IAAS, University of Stuttgart
Stuttgart, Germany
leymann@iaas.uni-stuttgart.de

Antonio Puliafito
DICIEAMA, University of Messina
Messina, Italy
apuliafito@unime.it

Michael Zimmermann
IAAS, University of Stuttgart
Stuttgart, Germany
zimmermann@iaas.uni-stuttgart.de

ABSTRACT

Cloud federation allows cloud providers to dynamically use resources of other federated providers in order to fulfil the requirements of customer requests. This concept enables the federated cloud providers to use external resources for increasing their profit as they do not have to reject customers in case their own resources are occupied. However, (i) comparing the offers of the federated providers in order to decide which provider to use as well as (ii) adapting the installation scripts of the components to be deployed for the different providers is complex, error-prone, and time consuming. In this paper, we present an approach that enables customers to describe their desired application deployments in the form of a topology model that is independent of any concrete provider. We show how this model can be automatically adapted by a provider participating in a cloud federation to deploy components on different other participants. To ensure the practical feasibility of the approach, we employ the TOSCA standard for describing these models and present a technical system architecture based on existing technologies.

Keywords

Cloud Federation; Federated Multi-Cloud Deployment; Deployment Automation; TOSCA; XMPP

1. INTRODUCTION

The cloud federation concept finds its own roots in the political field and refers to a political organization where several self-governing participants are coordinated by a central government but, at the same time, keeping their independence from it. Adopting this organizational model in the field of IT, we define a cloud federation as a scenario where different clouds, belonging to independent administrative domains, interact with each other in order to share their available resources. Thus, providers participating in such a cloud federation play both roles the *user* and the *Cloud Resource Provider (CRP)* at the same time. In such scenarios, a CRP makes its own unused resources available to other CRPs which need external assets in order to be able to fulfil their users' requests. Cloud federation offers two main benefits to CRPs. First, it allows providers to earn revenue from computing resources that would otherwise be underutilized or unused. Second, cloud federation enables CRPs to virtually expand their offerings making them able to face sudden spikes in demand without having to invest money in new hardware resources. However, the interaction and cooperation among the participating providers is a complex challenge due to (i) different cloud management systems employed by the providers, (ii) proprietary metamodels used for describing application deployments, and (iii) the missing combination of standards to coordinate the available cloud resources and the actual federated application deployment.

In this paper, we propose an automated approach for deploying federated cloud applications across multiple providers based on standards. We introduce an architecture for a *Coordinated Application Deployment System* and show how the deployment of an application consisting of multiple components can be coordinated across multiple providers belonging to the same federation by automatically adapting the deployment model. We validate the practical feasibility of this system based on three technologies: (i) an XMPP server [30], which manages the communication between CRPs, (ii) a pluggable Message Oriented Middleware (MOM), which is XMPP-compliant and able to manage the CRPs' cloud technologies [28][9] and (iii) the TOSCA standard [20], which enables describing deployments in a portable way.

The paper is organized as follows: Section II describes a motivating scenario used throughout this paper and explains fundamentals. Our automated deployment approach and system architecture is presented in the section III. Section IV and V validate the approach by a technical system architecture while Section VI describes related work. Section VII concludes the paper and discusses planned future work.

VALUETOOLS 2016, October 25-28, Taormina, Italy

DOI 10.4108/eai.25-10-2016.2266363

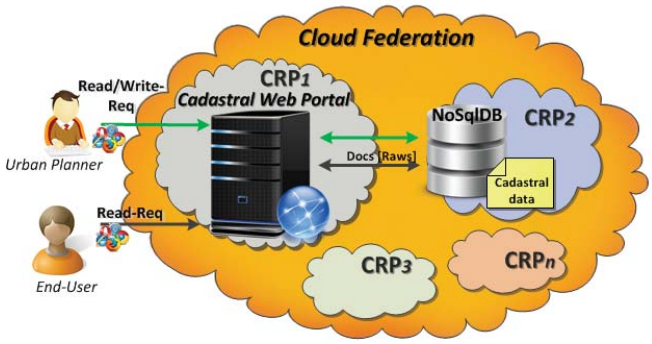


Figure 1: Federated Multi-Cloud Cadastral Web Application

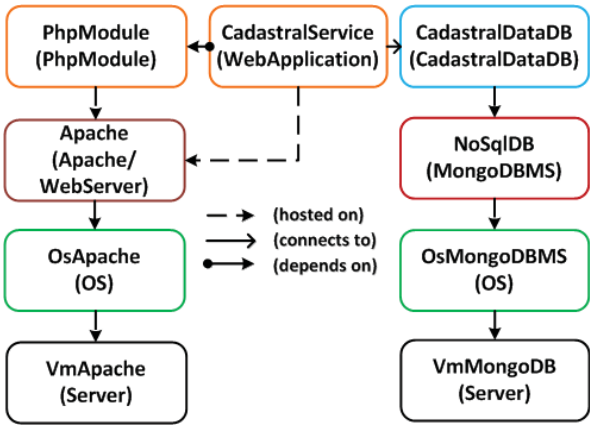


Figure 2: Cadastral Web Application Topology Template

2. MOTIVATION & FUNDAMENTALS

In this section, we introduce a motivating scenario that is used to explain the presented approach. Afterwards, we introduce the TOSCA standard to provide the background required for understanding the concepts of this paper.

2.1 Motivating Scenario

In this subsection, we first describe a motivating scenario that is used to explain the presented concepts. Figure 1 depicts a web application, made available by the public administration, providing "Cadastral Servicies" to citizens. This application enables users to access the cadastral archives containing documents like maps, legal documents, data of real estates' owners, economic rents to calculate the taxes, and so on. The typical users of the system are basically (i) the *end user* and (ii) the *urban planner*. The former assesses the system only in read-mode, while the urban planner has got both read and write permissions. The application consists of two main application blocks: a web portal acting as user interface between the client and the application logic, and a NoSQL-based DB for collecting all the cadastral and user data. The choice to use a NoSQL database is motivated by the kind of data has to be collected: maps, documents, geographic coordinates, and other kind of related meta-data. Our goal is to realize the automated deployment of the application's components across several cloud providers participating in the same federation at runtime, i. e., when a *Public Administration (PA)* asks one CRP to deploy and to set up this an application. The resulting scenario is shown in Fig. 1, where the two mentioned blocks are deployed on two different Cloud Resource Providers, namely CRP_1 and CRP_2 .

In order to be able to automate this kind of deployment, in this paper, we will take advantages of the application modelling and orchestration features provided by the OASIS standard TOSCA. To provide all required background information about TOSCA, we explain the standard in the next Subsection 2.2. However, since TOSCA does not provide cloud federation management capabilities natively, we will exploit two other technologies to manage the federated communications between the involved CRPs: (i) the *XMPP* protocol for message exchanging and (ii) an additional abstraction layer made on top of a Message Oriented Middleware (MOM), which is able to maintain the federation and to drive the underlying piece of middleware of each CRP. These technologies will be deeply discussed in Section 4.

2.2 The TOSCA Standard

The Topology and Orchestration Specification for Cloud Applications (TOSCA) [20, 19, 5] is an official OASIS standard. The main goals of the standard are to describe cloud-applications in an interoperable and portable manner in order to enable automating their deployment and management. TOSCA enables cloud users to model the structure of an application to be deployed using a *topology model*. Applications can be deployed based on these models by a *TOSCA Runtime Environment*, e. g., OpenTOSCA [6]. To automate management functionalities, for example, to scale out an application component, TOSCA employs the concept of *management plans*, which are executable process models, e. g., workflows, implementing a certain management functionality.

Based on these two concepts, TOSCA enables the automated provisioning and management of cloud-applications. TOSCA topology models are modelled in the form of a *topology template* (see Figure 2), which is a directed graph consisting of *node templates* representing the components (e. g., an Apache HTTP Server) and infrastructure resources (e.g. a virtual machine). These nodes are connected through directed edges, called *relationship templates*, which represent the dependencies between the nodes. For example, a relationship template may model that a PHP Web Application is *hosted on* an Apache HTTP Server and needs to be *connected to* a MongoDB database. TOSCA also enables modelling *node types* and *relationship types*, which define the semantics of the node and relationship templates. These type definitions are reusable building blocks specifying properties (e.g. the IP address) and management operations (e.g. "install the component"). These management operations are implemented using so called *implementation artifacts*, which are executable programs such as scripts. The business logic of the nodes themselves are implemented using *deployment artifacts*, which are implementations of the component—for example, a deployment artifact could be an archive containing the PHP files of the application to be deployed.

The standard also specifies the portable and self-contained *Cloud Service ARchive* (CSAR), which enables packaging all the aforementioned management plans, topology models, type definitions, and artifacts. As a result, CSARs can be automatically processed by any standard-compliant TOSCA Runtime Environment to deploy the described application.

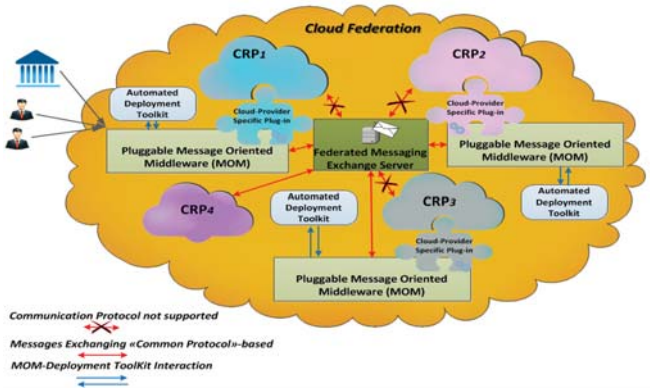


Figure 3: Conceptual Architecture of CADS

3. A CONCEPTUAL ARCHITECTURE FOR FEDERATED CLOUD DEPLOYMENT

In the perspective of deploying multiple application components in a federated cloud environment, this section presents the first contribution of this paper in the form of an approach for coordinating and automating the deployment of components across multiple cloud providers participating in a cloud federation based on a single deployment model. We (i) first describe the conceptual architecture of a *Coordinated Application Deployment System (CADS)* in Section 3.1, followed by concepts for (ii) decision making, and (iii) deployment model completion in Section 3.2 and Section 3.3.

3.1 Overall CADS Architecture

Figure 3 shows a conceptual, high-level architecture of the proposed system. The depicted system aims for managing the multi-component application deployment over multiple federated clouds by splitting the application in different components and placing them on (possibly) different CRPs. The main actors of the system are explained in the following. (i) The *end-user* sends an application deployment request to one CRP participating in the federation where the user has a valid account. This request (a) either refers to an application offered by the provider using an unique application ID or (b) contains a complete deployment model describing the application that shall be deployed, for example, a TOSCA CSAR (see Section 4). In this paper, we focus on the latter scenario. Moreover, (ii) several *CRPs* taking part in the same federation and offering their available resources to end-users and other CRPs represent the second class of actors. (iii) Another actor is introduced by *Message Oriented Middlewares (MOM)* enabling the federation establishment, maintenance, and the communication among the CRPs, for example, to ask for available resources. Each CRP may be connected directly or through such a MOM to a (iv) *Federated Messaging Exchange Server*, which is the central core of the communication mechanism, through which all the messages exchanged by CRPs to coordinate the federated application deployment flow. Each CRP hosts an (vii) *application deployment toolkit*, which is responsible for receiving the deployment requests from the end-user. The toolkit then first decides which components of the received deployment model can be hosted directly by the respective CRP itself and which components shall be hosted on other CRPs of the federation by using the communication mechanism described above. Therefore, the toolkit sends out broadcast requests via the MOM and the exchange server to the federated CRPs that respond with their capabilities, i. e., the resources they are able to offer. After receiving these offers, the toolkit selects the CRPs that fit best for hosting the components and executes the deployment on them.

As shown in Figure 3, we can count four CRPs in the depicted scenario namely CRP_1, CRP_2, CRP_3 and CRP_4. The first three providers do not directly have support for the central communication mechanism, so in order to take part in the federation, they *must* set up and configure the MOM, whereas CRP_4 is natively compliant with that communication mechanism, so it can join the federation without an own MOM. Anyway, in order to handle the runtime application deployment, CRP_4 has to implement its own business intelligence controlling local component deployments requested by a federated provider. In the figure, missing support of the communication mechanism is highlighted by the red-crossed-arrows. The deployment toolkit is the entity in charge to perform the application component deployment.

In a dynamic environment like a cloud federation where CRPs can join and leave whenever they want, each CRP needs to know how many and which CRPs are present within the federation at a given time. For this reason, the communication mechanism is based on the presence concept. In other words, in order to be easily identified and to identify, each available CRP has to provide a software agent that can access to a specific shared location. By exploiting this kind of communication mechanism, CRPs can retrieve (in addition to the presence information) the available resources of the other CRPs participating in the federation.

3.2 Discovery Phase and Decision-Making

According to the previous sections, our goal is to set up an environment which is able to accomplish the automated multi-component application deployment over multiple federated cloud providers. To achieve this goal, we need a software entity that takes as input the desired application deployment model (e. g., a topology model) as well as information about the participating CRPs' resource offerings in order to give as output the deployment choices, namely *where* (on which CRP) the application's components have to be deployed on. However, we are not discussing selection algorithms because this is out of the scope of this paper. Anyway, there are many works in the literature facing this topic in different ways. For example, [10] and [32] propose two similar match-making-based approaches, whereas [17], [24], and [31] propose different mathematical solutions. In this paper, we instead focus on the information retrieval concerning the availability of the CRPs' resources. Because our system adopts a decentralized solution, a CRP, in order to find the other CRPs' resources, needs to broadcast to each of them discovery messages. We call this message broadcasting process and the related response messages *discovery phase*. These information exchanges are performed by the MOMs and the central exchange server that, by means of the presence-based communication mechanism, will query all the other present CRPs retrieving their detailed resource information including the amount of available resources. For the sake of simplicity, we focus on virtual machine resources (VMs), VM types (amount of CPUs, memory, storage), and their price in this paper. However, the presented approach

and the following details can be easily applied to other *as a Service* offerings, too, e. g., PaaS offerings. These information about the CRPs will be used as input for the decision-making process. Then, after the system decided on which CRPs the application components shall be deployed, these results are sent to the automated deployment toolkit for the final *deployment phase*. In this phase, the original deployment request or the deployment model sent by the end-user (cf. Section 3.1) gets adapted for the chosen deployment decisions. For example, in case an end-user requested the deployment of an application consisting of several virtual machines described by a topology model, based on the selected CRPs, this model gets automatically adapted by the toolkit in terms of inserting the selected CRP nodes below the virtual machines. In particular, if the topology model depicted in Figure 1 was initially requested by the end-user, the system selects appropriate CRPs for hosting the two servers as virtual machines and inserts the respective nodes that describe these CRPs. Then, the resulting model gets automatically deployed on these CRPs. Based on this adaptation, the application's components get distributed over different cloud providers participating in the federation. This step consists of exploiting the selected CRPs' information to complete the final deployment model and gets explained in the next subsection.

3.3 Deployment Phase and Model Completion

After the CRPs to be employed are selected and which component stack of the application should be deployed to which CRP, the original deployment model needs to be adapted accordingly. There are works, for example the approaches presented by Hirmer et al. [12], for automatically completing incomplete deployment models. However, the available approaches lack support for choosing a specific CRP of a cloud federation based on requirements such as available computing resources or memory — moreover, many available approaches are not based on standards. Therefore, we introduce a software component called *Deployment Model Completion Manager*, which is able to adapt the stacks modelled in a deployment model to specific CRPs. As input, this component gets (i) the deployment model to be adapted as well as (ii) the decisions made by the decision making component described in Section 3.2. Afterwards, for every stack to be deployed, the Deployment Model Completion Manager adds the corresponding infrastructure node defined for the specific selected CRP. This requires a *CRP Registry* containing a list of CRPs and the required information to deploy software on this CRP, for example, information about the employed infrastructure for instantiating new virtual machines, e. g., OpenStack, and the required credentials. Therefore, each CRP maintains such a registry that is based on contracts between the CRPs. Thus, if a new CRP enters the cloud federation, its specific properties need to be added to the registries of participating CRPs, which is typically a manual process based on contracts. Using this information, properties of the infrastructure nodes, for example, credentials and desired ports, which are CRP-specific, are added automatically by the Deployment Model Completion Manager. After this step, the resulting model gets automatically deployed on the modelled CRPs using an appropriate runtime, which accesses the APIs of the involved CRPs to deploy the respective components remotely. In the following section, we explain these steps in detail based on standards.

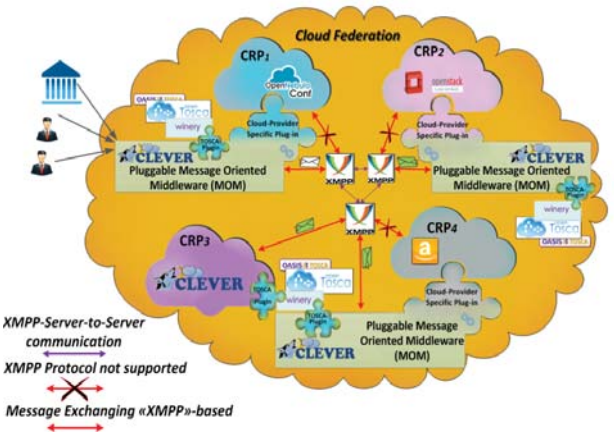


Figure 4: Technical Realization of the CADS Architecture

4. VALIDATION OF THE CADS APPROACH

In this section, we refine the conceptually described CADS approach presented in the previous section based on concrete standards and technologies. In Section 4.1, we first refine the presented CADS architecture and describe in Section 4.2 how the TOSCA standard can be employed to automate deploying application components on different federated providers.

4.1 Standards-based CADS Architecture

Section 3.1 showed an overview of the conceptual CADS architecture. Based on that abstract system presentation, this section presents a validation based on standards and existing technologies. Figure 4 shows a detailed version of the architecture previously seen in Figure 3. Substantially, we replaced the conceptual entities with concrete technologies we identified as best solutions to achieve our goal.

The first refined entity is the end-user asking for the deployment of an application. In this refinement, we focus on the scenario in which the end-user sends a deployment model to be provisioned to a CRP. As we have several CRPs participating in the federation, each of them has its own hardware asset and provides virtualization middleware such as OpenStack. As mentioned earlier, we focus on VMs but the architecture can be easily adapted for other aaS-offerings, too. Another important system's entity is an open source communication protocol exploited by the federated CRPs to exchange all the communication needed to manage the federation. Such a protocol should be based on the presence concept (cf. Section 3.1). In this domain, the *Instant Messaging and Presence Service (IMPS)*[1], the *Session Initiation Protocol (SIP)* and its derivative *SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE)*[2], as well as the *Extensible Messaging and Presence Protocol (XMPP)* [30] are the most important protocols. We chose XMPP due to its extensible paradigm, its decentralized architecture, its flexibility, and high-level of re-activeness. To exploit the XMPP protocol, there is the need to set up a Jabber/XMPP server (or more) providing basic messaging and XML routing features within the federation. It is possible to have more than one XMPP server, as it can be seen in Figure 4.

[1]http://www.ietf.org/rfc/rfc2778.txt/
[2]https://datatracker.ietf.org/wg/simple/charter/

However, typically not all CRPs are natively XMPP-compliant. Therefore, in order to join the federation, each of them needs to exploit the features of an additional abstraction layer namely a Message Oriented Middleware (MOM) that supports XMPP. Our choice is the *CLoud-Enabled-Virtual-EnviRonment (CLEVER)* [9], which is a secure Message-Oriented Middleware for cloud federation. Our choice is validated by three main reasons: (i) the CLEVER's inherent XMPP-based communication, (ii) its pluggable and (iii) federation management features. These features, therefore, make the CLEVER middleware suitable to manage both the federated communication and the underlying cloud platform. Moreover, in our federated scenario, we can directly have a CLEVER-based CRP (CRP_3 in Figure 4). Once the CLEVER-MOM of a specific CRP receives a deployment application request (including a deployment model), it first analyses the request to figure out how many resources that application needs, and afterwards triggers the "discovery phase". In particular, it sends discovery messages to the other CRPs to retrieve the federated resources' availability and the related information (via response messages). This will take place merely querying the *shared location* on which it is published. More specifically, in an XMPP-based environment, the shared location consists of a specific *XMPP chat-room* where the cloud providers taking part in the federation have to hold a software agent's subscription. The above mentioned messages are XML-based messages. Figure 5 shows a possible structure of these query messages:

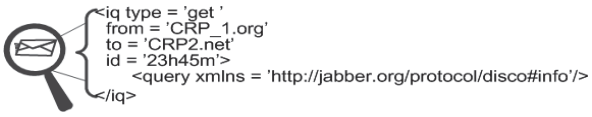


Figure 5: Simplified XMPP Query Message

Figure 6 shows a possible response of a CRP.

Figure 6: Simplified XMPP Result Message

In order to retrieve information about the CRPs' resource state, the requesting cloud provider sends and XMPP *IQ* stanza of type `get` containing an empty <query/> element qualified by the `http://jabber.org/protocol/disco#info` name-space to all other providers present in the federation room. Each CRP receiving the query message will respond sending an XMPP *IQ* stanza of type `result`, which contains one or more <identity/> elements including static information and a potentially unbounded number of <feature/> elements describing resource availability information (e. g., number of available VMs at a given moment for a specific type, VMs' properties, VM's price, etc.). This information will be elaborated by the decision-making software component to find out the best deployment solution.

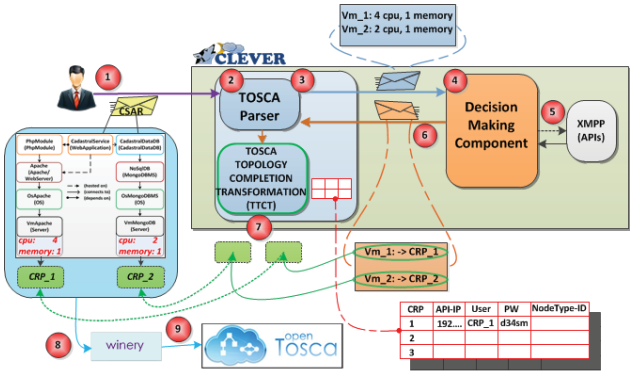


Figure 7: Federated Topology Completion Process

To realize the automated deployment of federated cloud applications, the TOSCA standard is employed. Therefore, a Java-based plugin component is integrated into CLEVER providing a *TOSCA Topology Template Parser* and a *TOSCA Topology Completion Manager (TTCM)*, which implements the concept of the Deployment Model Completion Manager for TOSCA (cf. Section 3.3). The parser is responsible for analysing TOSCA Topology Models requested for deployment by the end-user. Afterwards, CLEVER uses the discovered information to utilize the TTCM for inserting corresponding node and relationship templates for injecting the selected CRPs into the topology template. For automating the actual deployment of the resulting model, a TOSCA Runtime Environment is employed. In our realization, we employ the *OpenTOSCA Runtime Environment* [6][29] as a deployment toolkit, which is a standard-compliant, open-source implementation of a TOSCA Runtime. These two entities work together to accomplish the final topology completion and application deployment. The topology completion process will be discussed in detail in the next subsection.

4.2 TOSCA Topology Completion

In this section, we describe how the Deployment Model Completion Manager can be realized using TOSCA. Figure 7 shows CLEVER's internal components as well as the necessary external tools to achieve the federated application deployment based on TOSCA. The Java-based TOSCA plugin is composed by (i) the TOSCA Topology Parser, (ii) the TOSCA Topology Completion Manager and (iii) the *Decision-Making component*. The process starts (step 1) with the user's deployment request submission. It consists in sending a CSAR containing the incomplete topology template (VMs are modelled but without CRPs). The submitted topology model is the same as the one shown in Figure 2, but now decorated with additional resource requirements. At step 2, the parser gets the CSAR and analyses the topology to figure out how many resources (VMs, CPUs, memory) the application to be deployed needs. At step 3, the parser composes an XML-based message with the above informa-

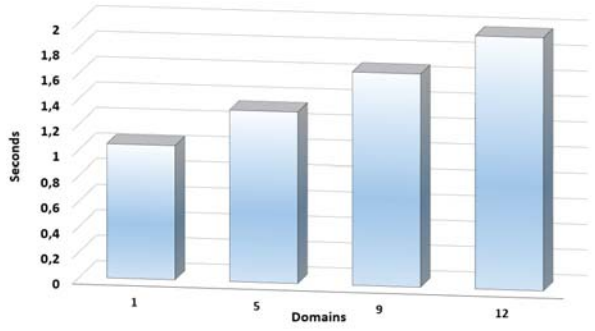
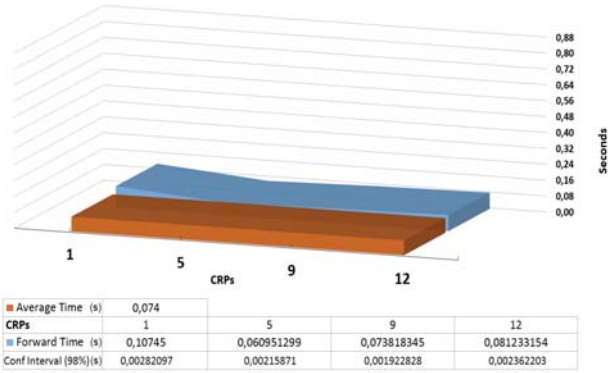


Figure 8: Retrieving Availability Resource Information



Average Time (s)	0,074			
CRPs	1	5	9	12
Forward Time (s)	0,10745	0,060951299	0,073818345	0,081233154
Conf Interval (98%)(s)	0,00282097	0,00215871	0,001922828	0,002362203

Figure 9: CRP-to-CRP message delivery time

tion and then sends it to the Decision-Making component (step 4). In order to retrieve the federated CRPs' resource offerings, it invokes the XMPP APIs (step 5), thus, triggering the Discovery-Phase. After receiving the federated offerings, the Decision-Making component selects the CRPs which better meet the application components' requirements. Then at step 6, it sends a response XML-message to the parser containing information about the selected CRPs to host the requested VMs. This information is forwarded to the TOSCA Topology Completion Manager, that (step 7) adds the node templates corresponding to the made decision as well as required properties to the incomplete topology model. Therefore, the manager also contains the registry described in Section 3.3, which contains all CRPs involved in the federation and the associated properties as well as the id of the corresponding node template of each CRP. The completed topology template is then sent (step 8) to Winery [16]. Primarily, Winery is a modelling tool supporting the modelling of topologies based on the TOSCA standard. But besides that, Winery also has a repository containing TOSCA related artifacts like implementation artifacts. Therefore, Winery is able to fetch further required artifacts such as the implementation artifacts belonging to the previously added node templates and packages them into a final CSAR. This CSAR now contains a complete topology model and all artifacts required for executing the deployment. Therefore, it can be deployed fully automatically (step 9) using an appropriate standard-compliant TOSCA Runtime Environment.

5. EVALUATION

This Section focuses on several experiments that we carried out on a real test-bed taking in consideration thirteen different CLEVER-based CRPs. With this section, we want to demonstrate that the addition of domains to the federation does not negatively affect the whole application deployment process. We arranged the test-bed considering federation composed of 13 CRPs. Twelve of them act as lender CRPs and only one instead acts as applicant. We assume that the applicant does not have enough resources to fulfil the user's request. These experiments have been conducted with the following hardware configuration: CPU: AMD Opteron 2218 HE Santa Rosa Dual-Core 2.6GHz ; 8GB RAM, running Linux Ubuntu 12.04 x86 64 OS and VirtualBox. Each trial was repeated for 30 times and finally the mean value and confidence intervals were evaluated. Our interest falls on the XMPP-Communication times, specifically we analysed the time to obtain the resource availability from the federated CRPs and the effective time needed to a XMPP-message to go from a CRP to another one. In particular, Figure 8 shows the delay experienced by a CRP when asking for external resources to obtain availability information from all the federated CRPs. We can see how the delay (due to the discovery phase) increases almost linearly when the number of CRPs grows. This behaviour is due to the fact that the discovery algorithm adopts multi-threaded programming. Thus, the software creates a new thread per each CRP that has to be queried. However, such a delay is of only 1 second per each twelve additional CRPs. In fact, Figure 9 shows that the point-to-point message delivery time remains constant, very small, and *negligible* (roughly $74ms$), regardless of the number of CRPs present within the federation. The differences among the obtained values are very small and have an order of magnitude of ms, therefore, they are comparable with network delays. For this reason, it is hard to figure out and argue the trend of the $ForwardTime$. Surely it does not depend on the number of CRPs within the federation.

6. RELATED WORK

This section highlights the main published works concerning the cloud federation, multi-cloud and application deployment topics and approaches similar to ours.

Jung et al. in [14] present an OpenStack-based scheduler able to deploy an application topology over a cluster optimizing the resources utilization of the data center so called *Ostro*. *Ostro*, taking as input the application topology to be deployed and considering the data center's available resources information, gives as output the best VMs placement that better meets the application requirements. This paper anyway does not face the placement across multi cloud providers. In [22], the authors introduce *STRATOS* as a brokering service able to take deployment decisions of cloud application topologies among multiple cloud providers. The STRATOS "Cloud Manager" node receives the application topology description and contacts the *Broker* asking for an instantiation of that topology. Loulloudes et al. [18] and Papaioannou et al. [21] presented two novel frameworks supporting both the TOSCA-based application orchestration. The first one makes easy the cloud application life-cycle management through Simple-Yet-Beautiful Language (SYBL) "*SYBL*" which is a language to specify in detail elasticity monitoring, constraints, and

strategies at different cloud application levels. The second architecture system instead aims to capture and store into its internal meta-data repository important application information, such as application descriptions and requirements, cloud providers' characteristics in order to achieve the multi-cloud deployment of complex multi-tier applications. The approach of Antoniades et al. [2] is closely linked to Loulloudes et al. [18] being based on "*CAMF*", too. The main contribution of the paper is to enhance the *TOSCA*-based application description by adding extra user requirements through a novel language called *CARL* (Cloud Application Requirement Language). In [11] Ferry et al. presented their own solution so called *CloudML* (Cloud Modelling Language) which supports matchmaking between deployment requirements and infrastructure descriptions. This solution wants to facilitate the application provisioning, deployment, monitoring, and adaptation in a multi clouds system. CloudML does not support Multi-Cloud provisioning: a single application cannot be spread across multiple providers. In Carrasco et al. [13], Brogi et al. [8], and [4] the authors presented a TOSCA-CAMP-based approach for managing and deploying applications over different and independent cloud providers. The works couple TOSCA's and CAMP's features and their respective approaches to exploit the benefits of both standards. Specifically, in Brogi et al. [8], the authors present *SeaClouds* and implemented an extension of TOSCA by defining a new element "*location*" into the node template enriching it with information about the target providers where each component has to be deployed. SeaClouds works on top technologies such as *DeltaCLoud*, *CAMP* and TOSCA adding for the latter the support to the applications over multiple clouds. Anyway, they do not focus specifically on the federation. In fact the authors do not give information regarding the maintenance of the federated environment and the communication technologies between the involved CRPs. Also Pham et al. [23] introduced a prototype of a novel framework for the multi-tier application deployment over multiple clouds. The different approach against Brogi et al.[8] is the use of a DSL-language for describing the application topology. Each component description also provides the dependencies between that component and the others. Trummer et al. [27] tackled the challenge of cost optimization. Their idea is very similar to that of Brogi et al. [8]. The input of the presented mathematical algorithm is an application template described as a graph where each node is a component of the application. Also in this scenario, the cloud providers are independent and they do not stipulate any federated agreement. In *Uni4Cloud* [26], the authors present a *OVF*- and *OCCI*-based framework enabling the automated multi-tier application deployment on multiple clouds. It is able to translate a High-Level Application Model in an OVF package (.ova extension). The framework plays only the role of a broker, which able to talk with several providers thanks to the OCCI APIs. Ardagna et al. [3] presented *MODAClouds*, which is a framework enabling the design and execution of applications over multiple clouds. MODAClouds performs a gradual transformation of a high-level application model into a final cloud provider-specific model enabling the installation and orchestration of the application on the selected clouds. Juve et al. [15] faces the topic of automating application deployment over a single or multi cloud infrastructure. Especially, the authors present their system called *Wrangler*, which is able to deploy and orchestrate VMs in an automated way. The system's central component is the "Coordinator" which receives the users' application deployment requests (a XML application descriptor) and makes decisions regarding the application deployment across multiple clouds. In Wrangler, a prior step to prepare the VMs is installing the wrangler agent and configuring it to connect to the coordinator node. In Rochwerger et al. [1], the authors present the *Reservoir* architecture ([25]). This architecture creates a dedicated virtual environment (VEE) where each application component can be executed. These VEEs can be placed in the VEE hosts within a single Reservoir site or on different sites. One of the main *Reservoir* elements is the *Service Manifest*, which contains information about the structure of the application components such as master image references, resource requirements of a single instance (CPUs, memory size, etc.) and more. The Service Manifest extends the OVF standard.The Reservoir project is considered the ancestor of our presented work. The paper Bresnahan et al. [7] presents an interesting tool called "*Cloudinit.d*" for automating the virtual machine (VM) creation and maintaining processes (launching,configuring, monitoring, and repairing operations), the contextualization, and the coordination of service deployment. Cloudinit.d tool takes as input an *.ini* configuration file. This file contains the service description: the (IaaS provider) location where it will be run, the VM information, and the file system path (within the VM) of the *scripts* that need to be executed in order to install and configure the required software to start-up the service. However, enabling the VMs contextualization, using simple scripts could not be a good way in complex scenarios with multiple VMs and different kinds of operating systems.

7. CONCLUSION

In this paper, we tackled the challenge of a *multi-component application deployment over multiple federated cloud providers* by presenting a deployment system architecture. We described and designed the architectural solution highlighting the peculiarities of the selected technologies to achieve the desired goal. Moreover, the paper analyses the XMPP communication delay to demonstrate that it does not negatively affect the overall application deployment process. In future work, we are going to test the performances of such a cross-cloud federation deployments to evaluate not only the XMPP communication time but all its components' behaviour. In this scope, we will consider, for example, realistic and more complex multi-component applications and a federated environments characterized by hundreds of clouds that can dynamically take part and leave the federation in order to test the system's scalability.

8. REFERENCES

[1] B. R. andothers. The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, July 2009.

[2] D. Antoniades et al. Enabling Cloud Application Portability. In *Proceedings of the 8th International Conference on Utility and Cloud Computing (UCC)*, pages 354–360. IEEE, Dec. 2015.

[3] D. Ardagna et al. MODAClouds: A model-driven approach for the design and execution of applications on multiple clouds. In *Proceedings of the 4th International Workshop on Modeling in Software Engineering (MISE)*, pages 50–56. IEEE, June 2012.

[4] G. Baryannis et al. Lifecycle management of service-based applications on multi-clouds. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds - MultiCloud 13*, pages 13–20. ACM), 2013.

[5] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann. *TOSCA: Portable Automated Deployment and Management of Cloud Applications*, pages 527–549. Advanced Web Services. Springer, Jan. 2014.

[6] T. Binz et al. OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*. Springer, Dec. 2013.

[7] J. Bresnahan et al. Managing appliance launches in infrastructure clouds. In *Proceedings of the 2011 TeraGrid Conference on Extreme Digital Discovery*. ACM, 2011.

[8] A. Brogi, A. Ibrahim, J. Soldani, J. Carrasco, J. Cubo, E. Pimentel, and F. D'Andria. SeaClouds. *SIGSOFT Softw. Eng. Notes*, pages 1–4, Feb. 2014.

[9] A. Celesti, M. Fazio, and M. Villari. SE CLEVER: A secure message oriented Middleware for Cloud federation. In *Symposium on Computers and Communications (ISCC)*. IEEE, July 2013.

[10] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. How to Enhance Cloud Architectures to Enable Cross-Federation. In *Proceedings of the 3rd International Conference on Cloud Computing*, pages 337–345. IEEE, July 2010.

[11] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg. Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems. In *Proceedings of the 6th International Conference on Cloud Computing*, pages 887–894. IEEE, June 2013.

[12] P. Hirmer, U. Breitenbücher, T. Binz, and F. Leymann. Automatic Topology Completion of TOSCA-based Cloud Applications. In *Proceedings des CloudCycle14 Workshops auf der 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI)*, pages 247–258. GI, 2014.

[13] C. Jose, C. Javier, and P. Ernesto. Towards a Flexible Deployment of Multi-cloud Applications Based on TOSCA and CAMP. In *Communications in Computer and Information Science*. Springer, 2015.

[14] G. Jung, M. Hiltunen, K. Joshi, R. Panta, and R. Schlichting. Ostro: Scalable Placement Optimization of Complex Application Topologies in Large-Scale Data Centers. In *Proceedings of the 35th International Conference on Distributed Computing Systems*, pages 143–152. IEEE, June 2015.

[15] G. Juve and E. Deelman. Automating Application Deployment in Infrastructure Clouds. In *Proceedings of the 3rd International Conference on Cloud Computing Technology and Science*, pages 658–665. IEEE, 2011.

[16] O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann. Winery – A Modeling Tool for TOSCA-based Cloud Applications. In *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*, pages 700–704. Springer, Dec. 2013.

[17] C.-Y. Liu, K.-C. Huang, Y.-H. Lee, and K.-C. Lai. Efficient Resource Allocation Mechanism for Federated Clouds. *International Journal of Grid and High Performance Computing*, pages 74–87, Oct. 2015.

[18] N. Loulloudes et al. Enabling Interoperable Cloud Application Management through an Open Source Ecosystem. *IEEE Internet Computing*, pages 54–59, May 2015.

[19] OASIS. *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0*, 2013.

[20] OASIS. Topology and Orchestration Specification for Cloud Applications Version 1.0, 2013.

[21] A. Papaioannou and K. Magoutis. An Architecture for Evaluating Distributed Application Deployments in Multi-clouds. In *Proceedings of the 5th International Conference on Cloud Computing Technology and Science*, pages 547–554. IEEE, Dec. 2013.

[22] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski. Introducing STRATOS: A Cloud Broker Service. In *Proceedings of the 5th International Conference on Cloud Computing*, pages 891–898. IEEE, June 2012.

[23] L. M. Pham, A. Tchana, D. Donsez, V. Zurczak, P.-Y. Gibello, and N. de Palma. An adaptable framework to deploy complex applications onto multi-cloud platforms. In *Proceedings of the International Conference on Computing & Communication Technologies*, pages 169–174. IEEE, Jan. 2015.

[24] S. Rebai, M. Hadji, and D. Zeghlache. Improving profit through cloud federation. In *Proceedings of the 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE, Jan. 2015.

[25] B. Rochwerger et al. Reservoir - When One Cloud Is Not Enough. *Computer*, pages 44–51, Mar. 2011.

[26] A. Sampaio and N. Mendonça. Uni4Cloud. In *Proceeding of the 2nd international workshop on Software engineering for cloud computing - SECLOUD '11*, pages 15–21. Association for Computing Machinery (ACM), 2011.

[27] I. Trummer, F. Leymann, R. Mietzner, and W. Binder. Cost-Optimal Outsourcing of Applications into the Clouds. In *Proceedings of the Second International Conference on Cloud Computing Technology and Science*, pages 135–142. IEEE, Nov. 2010.

[28] F. Tusa, M. Paone, and M. Villari. CLEVER. In *Theory and Practice*, pages 219–241. IGI Global, 2012.

[29] University of Stuttgart – Institute of Architecture of Application Systems. OpenTOSCA – Open Source TOSCA Ecosystem, 2016. http://www.iaas.uni-stuttgart.de/OpenTOSCA/indexE.php.

[30] XSF. Extensible Messaging and Presence Protocol (XMPP): Core. Internet RFC 3920, October 2004.

[31] W. Yao and L. Lu. A Selection Algorithm of Service Providers for Optimized Data Placement in Multi-Cloud Storage Environment. In *Communications in Computer and Information Science*. Springer, 2015.

[32] K.-H. Yeh. An Efficient Resource Allocation Framework for Cloud Federations. *Information Technology And Control*, pages 64–76, Mar. 2015.