MUSA Deployer: Deployment of Multi-cloud Applications

Valentina Casola*, Alessandra De Benedictis*, Massimiliano Rak†, Umberto Villano‡,
Erkuden Rios§, Angel Rego§, Giancarlo Capone¶
* *Università di Napoli Federico II, DIETI, Napoli, Italy*
alessandra.debenedictis@unina.it, casolav@unina.it
† *Università della Campania Luigi Vanvitelli, DIII, Aversa, Italy*
massimiliano.rak@unina2.it
‡ *Università del Sannio, DING, Benevento, Italy*
villano@unisannio.it
‡ *Technalia, Spain*
erkuden.rios, angel.rego@technalia.es
‡ *CeRICT, Benevento, Italy*
gcapone@gmail.com

***Abstract*—The development of applications based on services offered by different, not conscious, providers, is expected to be growing in the next years. In order to offer effectively multi-cloud applications, many challenges still need to be faced. At this aim, the MUSA framework provides a DevOps approach to develop multi-cloud applications with desired Security Service Level Agreements (SLAs). This paper describes the MUSA Deployer models, which help developers to express their security requirements, and a Deployer tool that automatically provides cloud security services to offer Security SLAs.**

***Keywords*-Cloud Security, Per-service SLA, Security Service Level Agreement**

I. Introduction

Cloud computing is one of the most successful computing paradigms of the last decades. As pointed out by a recent study of the statistical office of the European Union *Eurostat* [1], 21% of EU enterprises used cloud computing in 2016. The wide diffusion of cloud services, offering functionalities related to different application domains and addressing different computing and storage needs, opens up to the possibility of building complex cloud-based applications that rely upon heterogeneous services, possibly offered by different cloud service providers (CSPs), to deliver value-added services to end-users.

Multi-cloud applications are an example of such applications. According to the definition provided by [2], multi-cloud applications combine resources from multiple independent cloud providers without the need for an explicit cooperation among involved CSPs. A different model is represented by the cloud federations, where providers voluntarily interconnect their infrastructures to allow sharing of resources. In both models, the software components of the application may be deployed anywhere (i.e., on any cloud provider), provided that functional and non-functional requirements are satisfied. This flexibility not only enables an efficient usage of existing resources, but also allows, in some cases, to cope with specific requirements in terms of security and performance. Common challenges related to the deployment of multi-cloud applications include (i) the identification of the cloud services (for the deployment of the application components) that allow to meet the application functional and non-functional requirements; (ii) the definition of guarantees on the fulfillment of existing regulations and security policies upon setting-up the application; and (iii) the support to the application consistent behavior (e.g., access to data) regardless of the components locations and of the topology of their interconnection in order to support potentially the migration of application components across clouds, if required.

These challenges, among others, have been recently tackled by the MUSA project [3], [4], which aims at offering a framework to support the developer of multi-cloud applications throughout their complete life-cycle. MUSA, in particular, takes care of the security aspects of the multi-cloud applications, identifying and trying to address the risks that multi-cloud deployments might open, through a security-by-design approach and by applying risk analysis from the very early development stages.

In this paper, we focus on the deployment phase of a multi-cloud application in MUSA. We describe a tool that, starting from the high-level model defined according to the MUSA flow, produces a concrete deployment plan (that summarizes the software components to be installed and the services to be acquired in order to satisfy given security requirements) and automatically executes it, acquiring all the needed cloud resources and deploying and suitably configuring all the software components on such resources.

The remainder of the paper is structured as follows: Section II provides a brief overview of the MUSA project and of the proposed multi-cloud security-by-design development process. Section III introduces our proposal for the management of the deployment phase, which relies upon

a two-layer model for the representation of multi-cloud applications. Sections III-A and III-B discuss this model in detail, while Section IV illustrates the architecture of the MUSA Deployer tool. Finally, Section V presents some related work on deployment configuration and automatic execution, and Section VI summarizes our conclusions.

II. The MUSA Framework

The MUSA framework, which is the main result of the MUSA project, supports agile and DevOps-oriented engineering of secure multi-cloud applications. The DevOps approach promotes the continuous collaboration of the development team with the testing team and the IT operations specialists in order to improve the frequency and quality of software release. The core idea of the MUSA solution is to address the full application life-cycle, from design and development up to runtime management and application maintenance, with security as the key driver of the whole process. A multi-cloud application life-cycle is managed by MUSA through the flow shown in Figure 1.

During the first phase (*Modelling*), the DevOps team models the application using a Cloud Provider Independent Model (CPIM) of the multicloud application. Such phase, carried out by the MUSA Modeler tool, relies on a specific application modeling language (based on CAMEL [5]), which allows to describe, at a high-level of abstraction, both the application architecture and the deployment requirements, independently of the specific providers that will be actually used by the components. Then, in the *Risk Assessment* phase, the DevOps team invokes the MUSA Risk Assessment tool to carry out an early risk assessment process in order to identify the security controls and the security Service Level Objectives (SLOs) required by the multi-cloud application components [6]. The subsequent phase (*Cloud Service Selection*) entails the selection of the cloud services to be used to implement and deploy the application components. It is supported by the MUSA Decision Support Tool (DST), which is responsible for identifying the combinations of cloud services that best match the requirements specified in the *Modelling* phase.

After the definition of the security requirements of the application components and the selection of the list of cloud services they will use, the DevOps team can generate the Security Service Level Agreement (SLA) templates for the components by means of the MUSA SLA Generator tool (*SLA Template Generation* phase). The Security SLAs are contracts that formally state the level of security offered by a given provider or service [7], [8], and are modeled here as a set of declarations about the security controls implemented by the service [9] and the related guaranteed SLOs (Service Level Objectives) . SLAs generated in this phase do not contain yet formal guarantees about the security offered by selected services, since the actual level of security may depend on the mutual inter-dependencies among components and services. For this reason, we talk about SLA *templates*. SLA Templates generated in this phase are stored in the SLA Repository. The *Deployment phase* is split into two sub-phases, namely *Deployment planning* and *Deployment and Execution*. In the *Deployment planning* sub-phase, the *deployment plan* is generated, which includes the low-level information needed to acquire the cloud services selected in the *Cloud Service Selection* phase, and to configure them in order to run the application components. Once the deployment plan has been generated, the MUSA Deployer shares it with the MUSA SLA Generator, so that the latter can start the *Composite SLA Generation* phase, which is responsible for the automatic creation of the *composite SLA* for the whole multi-cloud application. The composite SLA will include the security guarantees associated with the multi-cloud application. It is built by properly combining the SLA templates of the application components (generated in the *SLA Template Generation* phase) and the SLAs published by involved providers, by taking into account the relationships among the components and the underlying cloud services and the impact that they may have on involved security control families.

Finally, in the *Deployment and Execution* sub-phase, the MUSA Deployer is used by the DevOps team to automatically deploy the multi-cloud application components by following the deployment plan. This step involves the acquisition and configuration of the cloud services as well as the installation of the application components software artifacts to guarantee security SLAs. When the application is up-and-running, the MUSA Security Assurance Platform starts monitoring it, based on the final composite SLA and on the deployment plan. If any violation of the SLA is detected, a notification is sent to the DevOps team and a re-design or re-deployment iteration of the multi-cloud application is launched.

One of the key advantages of the MUSA framework is that each of the tools used in the different phases of the application life-cycle can be used independently of the others and addresses specific needs. This paper, as anticipated, will focus on the multi-cloud application modeling and deployment, which involves the first, fifth and seventh steps in the process described above.

III. Modeling a multi-cloud application

As already mentioned, a multi-cloud application consists of one or more software components that interact with one another and need cloud resources for their implementation and/or execution. In particular, application components use SaaS (Software-as-a-Service) cloud services, are hosted by IaaS (Infrastructure-as-a-Service) cloud services and/or use PaaS (Platform-as-a-Service) cloud services offered by multiple CSPs.

In the flow presented in the previous section, the multi-cloud application is described through incremental models.

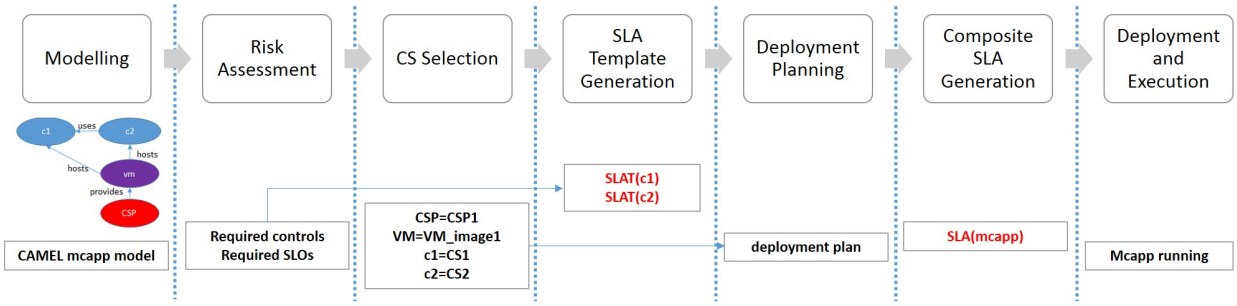

Figure 1. The MUSA flow for the development and execution of secure multi-cloud applications

In particular, we adopt a *two-layer modeling approach*, which entails the adoption of (i) a high-level model of the application, based on the CAMEL language, which is refined through the first four phases of the process to describe both provider-independent and provider-specific aspects of the application, and of (ii) a low-level description of the application (deployment plan), which can be directly used by the MUSA Deployer to carry out the deployment and configuration tasks. These two models are discussed in the following subsections before dealing with the MUSA Deployer tool.

A. The high-level multi-cloud application model

The high-level model of a multi-cloud application used by the MUSA flow is based on an extension of the CAMEL language. CAMEL (Cloud Application Modelling and Execution Language) allows to specify through a human-readable and structured text format the deployment of a cross-cloud application (i.e., of an application that can be defined once, and deployed at different providers seamlessly) along with its requirements and objectives. In particular, with CAMEL it is possible to specify the architecture of the application (in terms of its components), the high-level deployment model (in terms of the types of virtual machines to be used for each component) and the installation and configuration parameters for each component. Moreover, CAMEL allows to manage the information about the credentials to be used to access cloud services.

The first extension to the CAMEL language introduced by MUSA involves the refinement of the security aspects related to the credential management. In particular, we extended the set of supported credential types, added expiration dates for credentials, and introduced additional parameters to describe user accounts as, for example, its Role and Security Level. The second extension to CAMEL involves the introduction of the explicit modeling of the communication protocols used between two components. In fact, in CAMEL users can only specify that two components are interconnected and provide limited information (e.g., they can specify the IP addresses and ports used by components in case of IP communications), but they cannot specify the adoption of a specific protocol. Moreover, we introduced the definition of a set of dynamic features over communications, such as context paths (instead of IP addresses) and dynamic port ranges.

The third extension to CAMEL involves the improvement of the management of the components' life-cycle. In CAMEL, users assign life-cycle handlers to components that are responsible for installing, configuring, starting, and stopping the components on virtual machines. This deployment process is restricted to scripted commands, while there is no support for more advanced frameworks such as Cloudify [10], Puppet [11], or Chef [12]. In MUSA, we filled the gap between multi-cloud application models and existing configuration management tools by introducing the Configuration entity, embedding the concepts needed to interface with these tools (e.g., cookbooks and recipes, in the case of Chef).

Finally, we added explicit support for the coverage of security aspects. Indeed, as illustrated in the previous section, during the *Risk Assessment* phase the DevOps team identifies the security requirements (in terms of controls and security SLOs) of each application component. MUSA offers a set of enforcement agents that are able to fulfill the multi-cloud application security policies at runtime, by enforcing the needed security control in an as-a-service fashion. Such agents are collected in a Catalogue and can be included in the application model as any other component.

The MUSA framework offers a web-based user interface to write CAMEL-based high-level multi-cloud application models. It relies on the Xtext [13] technology and allows end-users to edit and update remotely the multi-cloud applications models without any program installation and by using any web browser.

B. The low-level multi-cloud application model

The *deployment plan* provides the application low-level description, and can be directly translated into deployment and configuration operations by the MUSA Deployer. It is represented in JSON, and its schema is available in the MUSA source repository [14]. It has a dedicated section for each CSP involved in the multi-cloud application deploy-

ment and, for each CSP, it contains two sections, including general infrastructure-related information and specific information on the virtual machines to acquire, respectively. In particular, the `infrastructure` section defines (i) the Provider, i.e., the specific CSP, (ii) the Zone, i.e., the region in which the resources can be acquired, (iii) the User needed to access the resource in order to install and configure the software components, and (if supported by the provider) (iv) the Network in which the resources have to be configured.

The `VM` section includes the list of VMs that have to be acquired with the information on whether they must be assigned a public IP or not, and reports the list of software components that have to be installed and configured on each machine. VMs are identified by (i) a Sequence Number, (ii) the installed Appliance (i.e., the identifier of the image of a default operating system, or a custom image that can be uploaded on the provider itself), and (iii) the underlying Hardware (i.e., an identifier of a specific combination of virtual CPU, RAM size and hard-disk size).

The components to be installed and executed on each machine are specified by means of the associated Chef *cookbook* and Chef *recipes*. Chef recipes allow to install and configure automatically one or many software components on a VM. Since the full multi-cloud deployment may need a specific order of installation and configuration of the software components, for each VM, the deployment plan defines the ordering of installation by means of the *implementation step* parameter.

Finally, for each VM, the plan specifies the inbound and outbound access rules in terms of the network ports that have to be opened, in order to allow or deny the communications among the VMs and of the VMs with the Internet. In particular, the deployment plan defines, for each VM, the protocol (i.e. UDP, TCP, ICMP) and the ports that have to be opened both as ingress rule and egress rule.

IV. MUSA Deployer architecture

In this section, we illustrate the architecture of the MUSA Deployer tool, responsible for the automation of the deployment of multi-cloud applications developed according to the MUSA flow. In particular, the MUSA Deployer:

1) *Prepares the deployment plan* based on the application's high-level description;
2) *Acquires the resources* needed to run the application according to what is reported in the `infrastructure` and `VM` sections of the deployment plan;
3) *Deploys and configures the application's components* based on the recipes specified in the deployment plan for each virtual machine;
4) *Starts the cloud services* and launches the application. The deployment plan is *implemented* at this point.

The above operations are managed by three main architectural components, namely the **Deployer Core**, the **Planner** and the **Broker** (see Figure 2).

The **Deployer Core** orchestrates the **Planner** and the **Broker** components that are responsible for the deployment plan preparation and implementation, respectively, and manages the persistence of deployment plans. This component offers the MUSA Deployer API, invoked by the MUSA front-end, and in turn it invokes the Planner API and the Broker API exposed by the other components.

The **Planner** is in charge of building the deployment plan based on the high-level model generated by the MUSA Modeller and on the SLA templates stored in the SLA Repository and associated with the selected services. SLA templates contain not only the information needed to invoke such services but also the references to the enforcement agents to be activated in order to cover specific security requirements (if any). The deployment plan can be reviewed and updated by the DevOps team through the MUSA front-end. As shown in the figure, the **Planner** is connected also to the SLA Generator tool: as discussed in Section II, the deployment plan is shared by the Deployer with the SLA Generator to launch the *Composite SLA Generation* phase.

The **Broker** component is invoked when the deployment plan is ready. It automatically acquires the services from external CSPs and deploys and configures the different cloud components of the plan by relying upon the well-known Chef technology. The **Broker** is composed of a **Chef Server**, which stores the cookbooks for components installation and configuration, and of a **Resource Broker**, which performs the actual provisioning of the required virtual machines. In particular, when the **Resource Broker** acquires a virtual machine, it installs a *Chef Client* on it. Chef Clients are responsible for the actual execution of the recipes taken from the Chef Server on acquired resources, which enable the deployment of the application components according to the plan.

The MUSA brokering process may take from few minutes up to an hour to start and setup completely a multi-cloud application, depending on the software to be installed and configured. When the deployment is completed, the MUSA Security Assurance Platform is informed by the **Deployer Core** about the components of the multi-cloud application that have been deployed, and starts monitoring them.

V. Related work

According to [15], an ideal deployment service should enable the automatic deployment of distributed applications, handle the complex dependencies between the application components, allow for the dynamic provisioning of the required cloud resources, support multiple cloud providers and enable the monitoring of the state of the deployment. Although some interfaces supporting the provisioning of IaaS cloud resources do exist, current CSPs still do not provide customers with automated and standard provisioning and deployment tools. Among the main initiatives for cloud

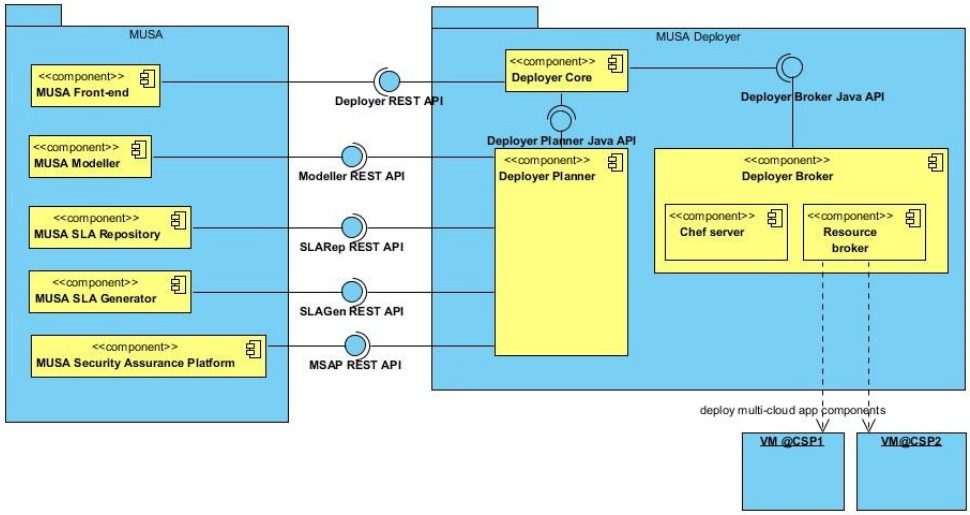

Figure 2. The MUSA Deployer UML component diagram

interoperability standardization, the Cloud Infrastructure Management Interface (CIMI) by the Distributed Management Task Force [16], [17] and the Open Cloud Computing Interface (OCCI) by the Open Grid Forum [18] stand out. The CIMI specification aims at standardizing the interactions between cloud environments to achieve interoperable cloud infrastructure management between service providers and their consumers and developers. The IaaS resources are modelled and made accessible via a RESTful HTTP-based protocol, although the model can be mapped to other protocols as well. CIMI supports resource representation in JSON and XML formats, via standard HTTP content-type negotiation. OCCI is a protocol and API for all kinds of management tasks, and it was originally born to create a remote management API for IaaS resources. The current version allows managing also other cloud model resources, e.g., PaaS and SaaS. OCCI allows the development of interoperable tools for common tasks including deployment, autonomic scaling and monitoring. The provided API is intended to be used via HTTP, in a RESTful fashion.

Despite these standardization efforts, the actual de facto standard for IaaS cloud resource acquisition is the Amazon Machine Image (AMI) [19], i.e., the API offered by Amazon AWS EC2, which is the API most adopted by cloud products and/or providers. For what regards PaaS providers, some of them provide target-dependent SDKs, which can work integrated in an IDE (Integrated Development Environment) and allow the application developers to deploy automatically the application designed inside the IDE into the managed cloud resources. However, this kind of deployment tools are proprietary, and therefore limited in terms of interoperability and of the number of cloud resources that they can handle. They only can manage the resources provided by the CSP, which is the owner of the automated deployment tool. Examples of such tools are Google App Engine SDK and Microsoft Azure SDK.

The SPECS broker [20], developed in the context of the SPECS project [21], [22], [23], offers a flexible solution to acquire resources from different CSPs, since it enables to manage pools of VMs, execute scripts on clusters of VMs and automatically register and configure software components with an automated configuration management toolkit. This tool, as well as the other mentioned solutions, does not explicitly address the goal of dynamically configuring resources acquired concurrently on multiple CSPs, as required in MUSA. The SPECS broker, however, enables functionalities similar to the ones offered in MUSA, even if it handles one CSP at a time. As a consequence, we developed the MUSA Deployer on top of the SPECS broker, adding the multi-cloud capabilities. It is worth noting that the SPECS broker is open-source, and offers a public API, built on top of a multi-cloud toolkit library, which enables full control over all resources acquired, thus simplifying the process of synchronizing multi-cloud resources. In the context of MUSA, the deployment execution is manually initiated by the DevOps Team that is using the framework to develop and deploy all components of the application. Therefore, it should allow working with single and multi-cloud component deployments and re-deployments that may affect only individual components. Besides, the separation of the planning and the broker (provision) services allows to have full control over the integration of the Deployer with other MUSA tools.

VI. Conclusions

The development of multi-cloud applications is very challenging and currently many open issues still need to be faced. The MUSA framework aims at providing a

DevOps environment to easily develop new secure multi-cloud applications, and provides support for their automated deployment, made more complex by inevitable security requirements. In this paper we described the MUSA Deployer tool, which is responsible for preparing a deployment plan and automatically configuring all the components of a multi-cloud application on different CSPs.

In addition to illustrating the Deployer architecture, we discussed the modelling approach adopted to enable the whole MUSA flow, which entails the adoption of a high-level model, obtained as an enhancement of the CAMEL language and enabling to specify, at a high level of abstraction, the application architecture and its requirements, and a low-level model (deployment plan), containing the information needed to automatically acquire, install and configure involved software components on cloud resources.

ACKNOWLEDGMENT

This research is partially supported by the grant FP7-ICT-2013-11-610795 (SPECS) and H2020-ICT-07-2014-644429 (MUSA).

REFERENCES

[1] "Cloud computing - statistics on the use by enterprises," http://ec.europa.eu/eurostat/statistics-explained/index.php/Cloud_computing_-_statistics_on_the_use_by_enterprises#Use_of_cloud_computing:_highlights.

[2] Global Inter-Cloud Technology Forum, "Use Cases and Functional Requirements for Inter-Cloud Computing. GICTF White Paper," http://www.gictf.jp/doc/GICTF_Whitepaper_20100809.pdf, 2010.

[3] E. Rios, E. Iturbe, L. Orue-Echevarria, M. Rak, and V. Casola, "Towards self-protective multi-cloud applications - MUSA - a holistic framework to support the security-intelligent lifecycle management of multi-cloud applications," in *CLOSER 2015 - Proceedings of the 5th International Conference on Cloud Computing and Services Science, Lisbon, Portugal, 20-22 May, 2015.*, 2015, pp. 551–558.

[4] MUSA Consortium, "The musa project web site," http://musa-project.eu/, 2015.

[5] "Cloud Application Modelling and Execution Language - CAMEL," http://camel-dsl.org/, 2017.

[6] V. Casola, A. De Benedictis, M. Rak, and E. Rios, "Security-by-design in clouds: A security-sla driven methodology to build secure cloud applications," *Procedia Computer Science*, vol. 97, pp. 53 – 62, 2016, 2nd International Conference on Cloud Forward: From Distributed to Complete Computing.

[7] V. Casola, A. De Benedictis, J. Modic, M. Rak, and U. Villano, "Per-service security sla: a new model for security management in clouds," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2016 IEEE 25th International Conference on*. IEEE, 2016, pp. 83–88.

[8] A. De Benedictis, V. Casola, M. Rak, and U. Villano, "Cloud security: From per-provider to per-service security slas," in *2016 International Conference on Intelligent Networking and Collaborative Systems (INCoS)*, Sept 2016, pp. 469–474.

[9] V. Casola, A. De Benedictis, M.Rak, J. Modic, and M. Erascu, "Automatically enforcing security slas in the cloud," *IEEE Transactions on Services Computing (PrePrints)*, 2016.

[10] "Cloudify," http://getcloudify.org/.

[11] "Puppet Labs," http://puppetlabs.com/.

[12] "Chef," https://www.chef.io/chef/.

[13] "X-text," http://www.eclipse.org/Xtext/.

[14] "The MUSA Project Bitbucket repository," https://bitbucket.org/account/user/cerict/projects/MUSA.

[15] G. Juve and E. Deelman, "Automating application deployment in infrastructure clouds," in *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, Nov 2011, pp. 658–665.

[16] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *2013 IEEE Sixth International Conference on Cloud Computing*, June 2013, pp. 887–894.

[17] "Cloud Infrastructure Management Interface. CIMI Standards," http://dmtf.org/sites/default/files/standards/documents/DSP0263_1.1.0.pdf.

[18] "Open Cloud Computing Interface. OCCI Standards," http://occi-wg.org/.

[19] "Amazon Machine Image Documentation," http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html.

[20] "SPECS Project. Secure Provisioning of Cloud Services based on SLA management. FP7- ICT-2013.1.5, 2013-2016," http://specs-project.eu/.

[21] M. Rak, N. Suri, J. Luna, D. Petcu, V. Casola, and U. Villano, "Security as a service using an SLA-based approach via SPECS," in *Proc. of CloudCom, 2013 IEEE 5th Int. Conf. on*, vol. 2, Dec 2013, pp. 1–6.

[22] SPECS Consortium, "SPECS project web site." [Online]. Available: http://www.specs-project.eu

[23] V. Casola, A. De Benedictis, M. Rak, and U. Villano, "Preliminary design of a platform-as-a-service to provide security in cloud," in *CLOSER 2014 - Proc. of the 4th Int. Conf. on Cloud Computing and Services Science, Barcelona, Spain, April 3-5, 2014.*, 2014, pp. 752–757.