Security-by-design in Multi-Cloud Applications: An Optimization Approach

Valentina Casola[a], Alessandra De Benedictis[a,*], Massimiliano Rak[b], Umberto Villano[c]

[a] *Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Naples, Italy*
[b] *Department of Computer Engineering, University of Campania Luigi Vanvitelli, Aversa, Italy*
[c] *Department of Engineering, University of Sannio, Benevento, Italy*

Abstract

Currently an increasing number of customers require cloud services with guaranteed security levels. At this aim, the adoption of multi-cloud strategies is spreading in a large number of interesting application domains, since they may potentially improve security and reduce development costs. However, the problem of identifying the optimal distribution of the components of a cloud application on resources belonging to multiple and heterogeneous providers is very challenging, especially in the presence of different security and performance constraints.

This paper presents a novel security-driven approach for the design, development and deployment of multi-cloud applications. It is based on a fully-automatable process that supports the developer from the elicitation of the application requirements up to the identification of the optimal deployment configuration, allowing to find the best compromise between overall cost and achieved level of security. The proposed optimization process takes explicitly into account two critical aspects that are often overlooked in similar approaches, namely the cloud on-demand leasing model for the allocation of resources and the impact that the deployment has on the security policies actually imple-

*Alessandra De Benedictis
Email address: alessandra.debenedictis@unina.it (Alessandra De Benedictis)

mented by a complex application.

Keywords: Security Service Level Agreement SLA, Analytic Hierarchy Process AHP, Security-driven Cloud Deployment Optimization, Secure Multi-cloud Application

1. Introduction

Over the last few years, new methods and techniques are emerging to support the development and automatic deployment of innovative ICT solutions. In this context, the adoption of *multi-cloud* strategies is recently becoming more and more interesting, to improve overall enterprise performance, to reduce costs and also to improve security. The term *multi-cloud* refers to the idea of accessing resources from different Cloud Service Providers (CSPs) that have not explicitly agreed to join their resources and to form a federation [21]. The use of multiple infrastructures can help meet the needs of diverse customers and, potentially, also minimize the risk of data loss or downtime due to a localized component failure. Moreover, it could be profitably exploited to distribute the components of an application on the providers that offer the best security guarantees with respect to their specific requirements.

Despite the high potentiality of multi-clouds, they pose many problems that require further research [40]. Security is a challenging issue even in single-provider scenarios, due to the different security policies and regulations adopted by the CSPs. The problem is significantly harder when there is concurrence of several cloud providers in the delivery of a single application. It is worth pointing out that the overall level of security that can be ascribed to a multi-cloud application depends not only on the level of security declared in the Service Level Agreements (SLAs) by the providers offering the resources used for the application deployment, but also on the way the application components interact with one another, and on the resulting impact on the implemented security policies. This dramatically complicates the choice of a suitable deployment configuration: A developer has to take into account too many parameters to

decide which is the optimal allocation of the application components on multiple providers offering different SLAs and different delivery models.

Moreover, the on-demand leasing model that is a distinctive feature of cloud environments makes the deployment problem in clouds substantially different from traditional resource allocation and scheduling problems, encountered in grids and other domains, for which well-known and effective solutions exist. These are usually modelled in terms of the generalized assignment problem [27], where the number of available resources is essentially pre-determined and fixed, and the goal is to minimize cost and/or maximize performance. In the cloud and multi-cloud environments, instead, the number of available offerings is potentially unlimited, and appropriate deployment optimization strategies must be devised.

To be able to take advantage of the benefits of the multi-cloud approach and to optimize the adoption of cloud resources from multiple providers, while taking into account the needed security guarantees, new security-driven design and deployment approaches must be introduced. To this end, this paper presents multiple contributions:

- We define a *security-by-design development process* aimed to support a developer in building and deploying a secure multi-cloud application while taking into account the potential security issues from the beginning of the development process, in order to reduce the risks associated with existing vulnerabilities and threats;

- We provide a *formulation of the cloud deployment problem* that explicitly takes into account the on-demand leasing model for the allocation of cloud resources, and that allows to express security-related constraints which take into consideration the impact of the deployment configuration on the overall level of security;

- We discuss a novel *security-driven optimization process*, aimed to identify the deployment configuration that fulfills the existing performance and

security constraints while ensuring the best compromise between the deployment cost and the overall level of security achieved, based on a scale of preferences expressed by the developer.

The remainder of the paper is structured as follows. In Section 2 we present the security-by-design multi-cloud application development process. In Section 3 we summarize the main issues behind the functional and security design of a multi-cloud application and present a graphical formalism used to model applications in both a deployment-independent and deployment-specific way. In Section 4 we formalize the cloud deployment problem, while in Section 5 we describe in the detail the optimization process. In Section 6, in order to demonstrate the feasibility and efficacy of our proposal, we present a simple case study and discuss the main phases of the development process focusing on the deployment optimization. Finally, in Section 7 we discuss related work on scheduling and allocation problems, and in Section 8 we draw our conclusions and sketch future research directions.

2. The *Security-by-design* Approach to Multi-cloud Application Development

A multi-cloud application can be defined as a collection of cooperating software components (simply *components*, in what follows) that rely for their execution upon cloud resources leased by multiple providers. The cloud resources can either be software components offered by CSPs in *SaaS* mode, or infrastructural resources. Most typically, the latter are virtual machines (VMs) onto which custom software artifacts are deployed. Besides code design and development, setting up a multi-cloud application also involves identifying how components are mapped to cloud resources, i.e., finding the application *deployment configuration*. This configuration has to be identified by projecting the application architecture, in terms of its components and of their interconnections, onto the multitude of CSP offerings, by taking into account constraints related to factors such as performance, cost and security. In particular, in this paper we consider

performance constraints related to the fulfillment of the resource demand of each application component in terms of memory and CPU usage. We do not consider the perceived performance of the whole application, typically specified in terms of response time and throughput. As regards security, we are interested in determining the deployment configuration that allows to fulfill existing security requirements related to each individual component and to the application as a whole. Moreover, we assume that security properties of both the available cloud services and the developed components and applications are expressed by means of Service Level Agreements (SLAs). Finally, we do not consider cost as an absolute constraint, but rather as an aspect to be taken into account in a trade-off with security, as will be clarified in the next sections.

In the remainder of this paper, we will focus on the problem of mapping custom software components to infrastructure resources (VMs) offered by multiple providers, deferring the use of SaaS components to future work. Currently providers offer several types of VMs, each characterized by different features in terms of performance, cost and security. These differences should be carefully taken into account for the identification of the "best" deployment configuration to be adopted for a given application. At the state of the art, the reasonable management of the inevitable security issues due to the de-localization of computations is an enabling factor for the further diffusion of the cloud computing paradigm. In fact, many users still prefer to host data and software services on premises, due to a perceived lack of security. Things are worse if multiple providers come into play, as the security of a multi-cloud application is likely to be the one of the weakest link in the chain. Our proposal is the use of a *security-by-design multi-cloud development approach* that aims to support the application developer during all the development phases of a multi-cloud application, from the specification of its high-level architecture, up to the identification of the deployment configuration that best fits its requirements. The proposed process consists of four main phases (see Figure 1).

During the *Functional Design* phase, the developer (i) describes the multi-cloud application in terms of its high-level logic components and of their in-

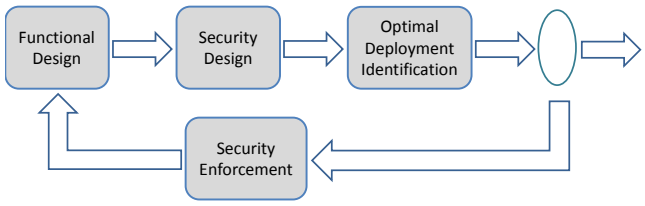

Figure 1: The security-by-design multi-cloud development process

terconnections and (ii) specifies the requirements of each component in terms of its resource demand. This phase is discussed in detail in Section 3.1, where a graph-based modeling technique suited for multi-cloud applications is also introduced.

In the *Security Design* phase, the developer (i) characterizes the application in terms of the security controls and security policies implemented internally by its components, and (ii) specifies the security requirements of each component, based on the specific needs and on existing risks. This phase, as will be discussed in Section 3.2, relies upon a new cloud security model based on the use of Security Service Level Agreements (Security SLAs) for the formalization of security policies, and the adoption of standards (as NIST SP800-53 or ISO 27000) for the selection of the security controls. Given the Security SLAs, security assessment and risk analysis procedures are used to identify the security features implemented by the application components and to elicit their requirements, respectively.

The outputs of the functional and security design phases are used in the *Optimal Deployment Identification* phase to identify the optimal deployment configurations for the components of the multi-cloud application. In particular, the adopted optimization process, presented in detail in Section 5, is aimed at identifying the deployment configuration that allows to fulfill the requirements identified during design phases, while ensuring the best trade-off between the deployment cost and the overall level of security achieved.

Depending on the specific application and on its requirements, the optimiza-

tion process might not be able to produce a valid result, due to the impossibility of covering security requirements (because of the lack of security controls in the component implementation). In these cases, an additional phase can be carried out, namely the *Security Enforcement* phase, devoted to identifying a set of *Security Libraries* (i.e., software modules) suitable to provide the enforcement of the security policies needed to cover the missing requirements. As the introduction of these security libraries implies the integration of additional software components within the application architecture, the whole process may undergo several iterations, until all the requirements are satisfied.

It is worth pointing out that part of the concepts behind the *Functional Design* and the *Security Design* phases of the development process have been introduced in the authors' previous papers produced in the context of the European projects SPECS [47] and MUSA [35]. However, in order to make this paper self-consistent, before going into the details of the deployment optimization process, we will briefly deal with the design steps in Section 3.

3. Design of a Multi-cloud Application

In this section, we summarize the main issues behind the functional and security design of a multi-cloud application. We provide the needed background on the techniques and methodologies used to (i) model a multi-cloud application from the point of view of its high-level architecture and of its deployment configuration, and (ii) enrich its description with functional and non-functional features and requirements.

3.1. Functional Design

The goal of the *Functional Design* phase consists in identifying the main components of a multi-cloud application, along with their mutual relationships.

There exist several languages to model cloud applications, such as CAMEL [38] and CloudML [46]. In this paper, we adopt a graph-based model named *Multi-cloud Application Composition Model (MACM)* [42], which can be directly obtained from descriptions in the above languages. Being a graph-based

representation, MACM allows to describe in a simple and immediate way the logic building blocks of an application, their mutual relationships and the information related to their deployment. Moreover, manipulating the representation and obtaining the information needed for deployment and security evaluation can be achieved easily by means of common graph-based query tools. However, it is worth pointing out that an MACM can be used to describe the static deployment of (multi-)cloud applications and to put in evidence the relationships among nodes that affect security properties, but it is not meant to model the run-time behavior of the application components.

An MACM model is a directed graph composed of several different types of nodes. As mentioned before, in this paper we only consider application components represented by custom software artifacts that have to be deployed on VMs in order to offer the specific functionalities invoked *as-a-service* by the application. This concept is modeled by means of the three fundamental MACM node types, represented by the *SaaS services*, corresponding to the application components, the *IaaS services*, corresponding to the VMs, and the *CSPs*, which model the providers offering the VMs. The relationships among these entities are represented as directed edges. In particular, three main relationships have been devised:

- A CSP *provides* compute IaaS services (i.e., VMs). This relationship is represented by an edge directed from a CSP node to an IaaS service node and labeled as "provides";
- An IaaS service *hosts* an SaaS service. This relationship is represented by an edge directed from an IaaS service node to an SaaS service node and labeled as "hosts";
- Any service, either IaaS or SaaS, *uses* an SaaS service. This relationship is represented by an edge directed from an IaaS/SaaS service node to an SaaS service node and labeled as "uses".

The high-level functional characteristics of each component have to be spec-

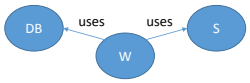

Figure 2: An example of deployment-independent MACM

ified by identifying its *type* (e.g., the component may be a custom web application accessible via a network, a generic software-as-a-service provided by a third party, or a storage service) and its performance requirements in terms of resource demand (i.e., required amount of RAM and CPU). Similarly, it is necessary to specify the VM characteristics in terms of the available storage and computation power. The component types, their performance requirements and the VM features are modeled as *properties*, used to annotate the respective nodes in the graph representation. Moreover, it is also possible to annotate links with suitable *properties* that give additional information on the existing relationships (e.g., a property may specify the protocol used on a communication link).

Figure 2 shows the deployment-independent MACM model of a very simple application. The model does not include IaaS service nodes or CSP nodes, but only the service nodes representing the application components. The application consists of a web application component W (i.e., a software that offers an HTTP interface) that uses a mySQL database component DB, and a component S that offers generic services in SaaS mode (e.g., a Map service).

3.2. Security Design

The goal of the *Security Design* phase is two-fold: On the one hand, (i) the security policy implemented by each component of the application is assessed; on the other, (ii) the security requirements of the application and of its components are identified.

As mentioned before, Security SLAs are adopted to model security policies [4]. These are SLAs that explicitly include security-related terms and guarantees. In particular, in Security SLAs the implemented security policies can be specified in terms of the set of enforced *security controls* prescribed by one of the

existing standard frameworks (such as the NIST Security Control Framework [36] or the Cloud Security Alliance's Cloud Control Matrix [14]). According to NIST, a security controls is a *safeguard or countermeasure prescribed for an information system or an organization designed to protect the confidentiality, integrity, and availability of its information and to meet a set of defined security requirements*. In this paper, we will adopt the NIST Security Control Framework, which lists more than 900 security controls, organized in different security domains ranging from access control (AC) to identification and authentication (IA), to media (MP) and physical environment (PE) protection, and includes also organizational aspects, as personnel training and planning.

The identification of security requirements is carried out by means of a risk analysis process that identifies for each component the main threats it is subject to, based on its nature and on its interactions with other components, and suggests the right countermeasures in terms of security controls to be applied to mitigate such threats. Presenting the risk analysis process is out of the scope of this paper; the interested reader is referred to the paper [7] and to a MUSA project demonstrator video available at [10] for further details.

The security assessment of a component, instead, aims at assessing the security controls it implements internally, and is carried out by means of *ad-hoc* questionnaires. It is worth noting that, at this stage, such process does not take into account the possible impact that the deployment of components on concrete cloud resources and the interactions among components and services might have on the overall implemented security policies. Therefore, the resulting security controls just represent the security policies that are *virtually* implemented by components, and are used to build the components' *Security SLA Templates* (SLATs). As will be widely discussed in the following, focusing on the deployment optimization process, a further step will be needed to obtain the actual components' SLAs based on the specific selected deployment configuration, namely the *SLA Composition process* [42]. These SLAs will be then used for the verification of the coverage of security requirements.

An MACM model can be enriched so as to include security-related informa-

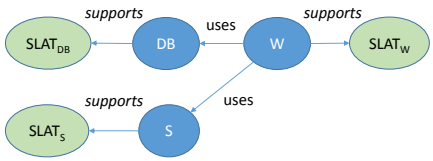

Figure 3: An example of security-annotated deployment-independent MACM

tion. In particular, the list of security controls resulting from the requirement elicitation process can be added to every component in the MACM model as a *property* of the corresponding node, in a way similar to the addition of node performance requirements. Offered security policies, either virtually implemented (if deployment-related issues have not yet been taken into account) or actually guaranteed, can be instead represented by means of two additional model node types, namely the *SLAT* and *SLA* nodes, respectively. The relationships among the MACM main node types (i.e., CSP, SaaS and IaaS) and the above two nodes are the following:

- Any service, either SaaS or IaaS, *supports* an SLAT. This relationship is represented by an edge directed from an IaaS/SaaS service node to an SLAT node and labeled as "supports";
- Any service, either SaaS or IaaS, *grants* an SLA. This relationship is represented by an edge directed from an IaaS/SaaS service node to an SLA node and labeled as "grants".

After the *Security Design* phase, the MACM of the application under development will include information on the security requirements of the application components with their SLATs. As an example, Figure 3 shows the security-annotated version of the deployment-independent MACM in Figure 2. Clearly, since it is a deployment-independent representation, no actual services and providers are involved yet, and no "grants" relationships are shown.

Later on, during the *Optimal Deployment Identification* phase, the model of the application will be updated by adding deployment-specific information,

including the actual security policies implemented by each node involved in the deployment. This information will be used to assess the level of security associated with a given deployment configuration and to verify the coverage of existing security requirements.

Before going into the details of the optimization process, in the following section we discuss the secure multi-cloud deployment problem, introducing a formalization that takes into account all involved inputs and constraints.

4. The Secure Multi-cloud Deployment Problem

The on-demand leasing model, which is a distinctive characteristic of cloud environments, makes the deployment problem in clouds substantially different from traditional resource allocation and scheduling problems, encountered in grids and other domains. These are usually modeled in terms of the generalized assignment problem [27], where the number of available resources is essentially pre-determined and fixed, and the goal is to minimize cost and/or maximize performance or to find a trade-off between the two criteria. Current cloud infrastructure providers, instead, typically offer several different types of VMs (often referred to as *instance types*), which are thought to fit specific cost/performance requirements of customers. In the following, we will use the terms "VM" and "instance" interchangeably, to indicate a virtual machine of some type hosted in a cloud infrastructure.

Given an application made up of several interconnected software components, each component has to be deployed on one and only one instance. On the other hand, an instance may in general *host* one or more components. The deployment of a cloud application may involve the acquisition of multiple instances of the same type, or even of a heterogeneous set of instances (i.e., VMs of different types, possibly leased from multiple providers). In both cases, given the possibility to host multiple components on the same VM, the number of instances (whether homogeneous or heterogeneous) is not a constraint of the problem, and alternative deployment solutions characterized by different num-

bers of leased VM may be considered. Of course, traditional resource scheduling models might still be adopted to describe the cloud deployment problem, but the resulting complexity would be intolerably high. As shown in the following, a model tailored to the particular cloud features turns out to be more manageable.

4.1. Problem input parameters

Let us refer to the available instance types (possibly offered by multiple providers) with the term *offerings*, and let us use the term *instance* to identify the generic VM, belonging to one of these offerings, which is actually leased for the application deployment.

The main inputs of the deployment problem are the application to be deployed, whose architecture has been specified during the *Functional Design* phase, and the cloud providers' offerings, which we assume to be available from a public service catalogue. Let $C = \{c_1, \cdots, c_n\}$ be the set of the n software components making up a cloud application and let $O = \{o_1, \cdots, o_m\}$ be the set of the m available offerings, each available in multiple instances (we will assume for simplicity's sake that the number of instances of any particular offering is unlimited).

Components are heterogeneous, in that they have different requirements in terms of performance and security. Similarly, every element of the set O is a distinct instance type, characterized by unique cost, performance and security features. Both the components' requirements and the offerings' features are an input for the deployment problem. In particular, it is possible to identify performance-related inputs, cost-related inputs and security-related inputs, discussed in the following.

The *performance-related inputs* are represented by the amount of computation, storage, or communication resources provided by available offerings and required by the application components, respectively. For simplicity's sake, we will consider a generic type of resource in our formalization, represented by a scalar quantity. Under this assumption, we can define the ***capacity*** $cap(o_j)$ of an offering $o_j \in O$ as the maximum amount of resource that is available on the

instances of type o_j. We assume that this information is specified in the service catalogue, along with all the other relevant characteristics of available offerings. Similarly, we define the ***resource demand*** $creq(c_i)$ of a component $c_i \in C$ as the minimum amount of resource that the component needs for its execution. Clearly, the resource demand of a component refers to the same resource type considered for the definition of offerings' capacity.

For what regards *cost-related inputs*, our model takes into account the ***cost*** associated with each offering $o_j \in O$, referred to as $cost(o_j)$, and representing the (hourly or monthly) unitary leasing cost of the instances of type o_j (available in a catalogue), specified by the developer at the beginning of the development process.

The *security-related inputs* are the most critical ones. They include (i) the security requirements of each component, elicited during the *Security Design* phase, and (ii) the security policies actually enforced by each component, which depend on their deployment and can be identified only with the SLA Composition process. As it will be discussed in Section 5.2, this process relies upon the components SLATs (obtained from the security assessment carried out in the *Security Design* phase), the SLAs associated with involved providers and related to the specific offerings selected for deployment (obtained from a catalogue), and the deployment configuration itself.

As said in Section 3.2, security attributes are modeled in terms of standard security controls. Let $SEC = \{sec_1, \cdots, sec_s\}$ be the set of the s reference security controls belonging to the NIST framework [33]. The ***security demand*** of a component $c_i \in C$ is represented by the subset $RSEC(c_i) \subseteq SEC$ of security controls that have been identified during the security design phase as the countermeasures to be adopted in order to mitigate existing security risks for the component c_i. Similarly, the ***security provision*** of a component $c_i \in C$ is represented by the subset $PSEC(c_i) \subseteq SEC$ of security controls that have been identified during the SLA Composition process as the actual security policies implemented by component c_i.

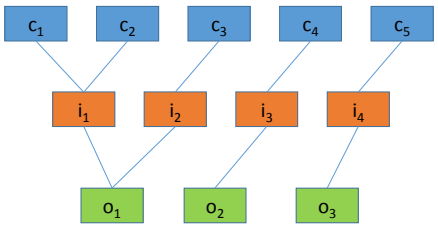

Figure 4: Example of deployment configuration

4.2. Problem output parameters

The output of the secure multi-cloud deployment problem is a physical cloud deployment configuration, which identifies how the components of an application are distributed (i.e., mapped) onto the instances belonging to available offerings. As previously pointed out, this is not a traditional assignment problem, since the number of assignment targets (i.e., instances) is not fixed, and can be freely increased/decreased, as discussed above. As shown intuitively in the example depicted in Figure 4, in general a cloud application deployment involves one or more instances (i_1,...,i_4 in the figure), belonging to one or more available offerings (o_1,...,o_3 in the figure), where each instance hosts one or more application components (c_1,...,c_5 in the figure).

The deployment problem can be modeled as a double assignment problem, which links components to instances and instances to offerings.

Let us refer to the subset of components deployed on the same instance with the term *VM-set*. Any partition of C into a set of non-empty and disjoint VM-sets is a potential deployment configuration, which describes how components may be distributed over different instances without taking into account the existing requirements and constraints. The ordering of components in a VM-set and of VM-sets in a partition is unessential: for instance, the partition depicted in the example in Figure 4, $\{\{c_1, c_2\}, \{c_3\}, \{c_4\}, \{c_5\}\}$, is identical to $\{\{c_3\}, \{c_2, c_1\}, \{c_4\}, \{c_5\}\}$.

Let $P = \{p_1, p_2, \ldots, p_K\}$ be a partition of C with cardinality K, referred to

as a *logical deployment*, with the elements $p_k \in P$ representing the associated VM-sets. An alternative representation of P, which explicitly identifies the mapping between each of the n components in C and the VM-set in P onto which the component has to be deployed, is the ***COMPONENT-VMset mapping vector*** $L = [l_1, l_2, \ldots, l_n]$. The i-th element of L is the index in P of the VM-set onto which the component c_i is mapped. In formal terms, $c_i \in p_h \Rightarrow l_i = h$, with $i \in [1..n]$, $h \in [1..K]$ and $p_h \in P$.

Logical deployments only show how components are distributed over different VM-sets (i.e., over different instances). The mapping between the K VM-sets involved in the logical deployment P and the corresponding concrete offerings belonging to the set of m offerings in O can be expressed through the ***VMset-OFFERING mapping vector*** $Q = [q_1, \ldots, q_K]$. The h-th element of Q identifies the index in O of the offering to which the h-th VM-set of P belongs. More formally, $p_h \in o_w \Rightarrow q_h = w$, with $h \in [1..K]$. $w \in [1..m]$ and $o_w \in O$.

In light of the above, a ***cloud physical deployment configuration*** D, which is the output of the deployment problem, can be represented through the pair $< L, Q >$, which describes how components are distributed over VM-sets, and how VM-sets are assigned to offerings.

4.3. Problem constraints

A deployment configuration is said to be *valid* if it fulfills both the functional and non-functional requirements elicited during the design phases, which represent the constraints of the deployment problem, as discussed in the following.

- **RC1**. **Coverage of resource demand requirements**.

  Let $D =< L, Q >$ be a physical deployment configuration, with $Q = \{q_1, q_2, \ldots, q_K\}$, and let $P = \{p_1, p_2, \ldots, p_K\}$ be the logical deployment associated with D. Let O be the set of the m available offerings. For each

$p_h \in P$, it must be:

$$VMReq(p_h) \leq cap(o_k) \tag{RC1}$$

where $k = q_h$ and $o_k \in O$.

The function $VMReq(p_h)$ returns the overall resource demand of the components allocated on the VM-set $p_h \in P$ according to L: it must be lower or equal to the capacity of the offering associated with p_h according to Q, namely o_k. Said $\overline{C_h}$ the set of components of C that have been assigned to p_h, and assuming that the amount of resource requested by components is additive, $VMReq(p_h)$ can be computed as:

$$VMReq(p_h) = \sum_{c \in \overline{C_h}} creq(c) \tag{1}$$

- **RC2**. **Coverage of security requirements**.

  Let D be a physical deployment configuration and let $PSEC(c_i)$ be the set of security controls actually implemented by component $c_i \in C$ according to the SLA Composition process carried out for D. Let $RSEC(c_i)$ be the set of security controls that have been identified as security requirements for c_i. For each $c_i \in C$, it must be:

  $$PSEC(c_i) \supseteq RSEC(c_i) \tag{RC2}$$

  The expression states that the security controls actually implemented by component c_i, according to the SLA Composition process, must include those that have been identified as its security requirements.

5. Multi-cloud Security-aware Deployment Optimization

The *Optimal Deployment Identification* phase aims at identifying a deployment configuration that is able to fulfill existing requirements in an optimal

way. It is worth pointing out that two different deployment configurations, both able to satisfy the resource demand of the application components and to cover the needed security controls, may have a very different impact, under several aspects, on the overall *quality* of the resulting deployment. For example, their cost may considerably differ, or they may involve offerings of providers with a highly different security level (based on their SLAs). In these cases, the optimization process should be able to identify the solution that best suits the requirements of the developer with respect to such criteria, possibly finding an optimal compromise based on an individual scale of preference.

The strategy implemented by the optimization process basically consists in (i) identifying possible deployment solutions, (ii) computing, for each solution, a *score* representative of its quality with respect to cost and security criteria, and (iii) trying to optimize such score. More detailed, the optimization process carried out during the *Optimal Deployment Identification* phase is the one sketched in Figure 5. It consists of four main steps:

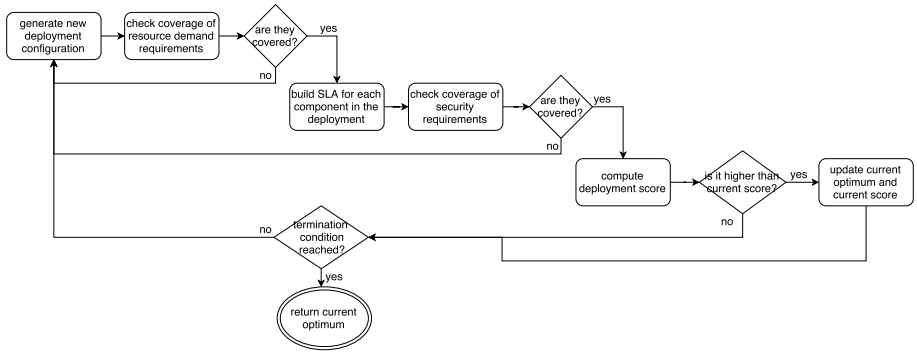

Figure 5: The optimization process

1. *Generation of a new deployment configuration.* The first step of the process entails the generation of a candidate deployment solution, expressed in terms of the couple $< L, Q >$ and specifying how components are mapped to offerings.
2. *Verification of the validity of the generated configuration.* The second

step of the process is devoted to verifying the validity of the generated deployment configuration. As said in Section 4.3, a deployment is valid when fulfills both resource demand and security requirements. While the maximum capacity of involved offerings is known *a-priori*, and thus the verification of the performance requirements can be done immediately, checking the coverage of security requirements of the candidate solution requires the execution of the SLA Composition process. Therefore, for each component of the application, the actual SLA is built based on the component's SLAT, which is available as output of the *Security Design* phase, on the information on the specific deployment configuration and on the SLAs associated with involved providers. The resulting components' SLAs are then compared with the security requirements specified for the components during design, to check the security requirements coverage.

3. *Computation of the deployment score and update of the current optimal solution.* If the candidate solution turns out to be valid, its *score* is computed as a combination of its cost and level of security, taking into account the preferences expressed by the developer with respect to the relative importance of such criteria. If the score of the new deployment is higher than the score of the best solution found so far, the current optimal solution and the current best score are updated accordingly.

4. *Verification of the termination condition.* The optimization process ends when a termination condition is encountered. This may be, for example, one of the following: the maximum time predetermined for the optimization process has expired; all possible deployment solutions have been checked; the current best score has exceeded a given threshold. Depending on the termination condition used, the process will produce the optimal or simply a sub-optimal solution.

In Section 5.1 and 5.2, we will provide an overview of the strategies adopted to generate new deployment solutions and of the outcome of the deployment security assessment by means of the SLA Composition process, respectively.

In Section 5.3, instead, we will discuss the core of the optimization process, namely the choice of the utility function and the computation of the score for each deployment solution.

5.1. Generation of deployment solutions

Several techniques can be adopted for the generation of candidate deployment solutions, including partial/total enumeration approaches and random selection approaches. It must be noted that the number of possible deployment configurations is very high even for small-size problems involving few components and offerings, as the deployment optimization problem is *NP-hard*. Clearly, the problem complexity can be reduced by adopting heuristic algorithms able to find sub-optimal deployments in a pretty efficient way; our implementation, available at [9], also supports heuristic solutions based on genetic algorithms. However, discussing the efficiency of the chosen algorithm is out of the scope of this paper, which is focused on illustrating the security-driven approach used in the optimization process.

For simplicity's sake, in the following we will illustrate an algorithm that derives directly from our formulation of the deployment problem, and that is able to generate **all** the distinct deployment configurations $< L, Q >$ of a multi-cloud application based on a combinatorial approach. The proposed *deployment configuration generation algorithm* takes in input the deployment-independent security-annotated MACM model of a multi-cloud application, and produces in output an enriched model including the mapping of components onto offerings, i.e., the deployment-specific security-annotated MACM[1].

The deployment configuration generation algorithm is a composition of two combinatorial algorithms, namely the (i) *Component-VMset mapping generation algorithm* and the (ii) *VMset-Offering mapping generation algorithm*.

[1] Considering the notation introduced in the previous section, this model allows to represent in a graphical way a logical deployment, namely the mapping between components and the instances that host such components.

The former algorithm takes in input the set of components C [2] and produces in output all the possible mappings of components onto distinct instances, which can be represented in terms of the L vector, i.e., the *COMPONENT-VMset mapping vectors* introduced in Section 4.2. In our implementation, the different vectors L are generated one-by-one by the function *nextL()*, reported in Listing 2, which is invoked iteratively by the deployment generation algorithm. The generation follows a specific order, starting from the configuration that maps all the components to a single VM-set, and ending with the configuration that assigns each component to a different VM-set. The first COMPONENT-VMset mapping vector is generated by the initialization function *initL()*, reported in Listing 1, which initializes to 1 all the elements of L and of another vector, max, which keeps status information and is used to avoid possible repetitions in the generation of the mappings.

The function *nextL()* takes in input the number of components to deploy `n`, the current vector L and the current vector max and returns true if a new vector has been successfully generated, or false if there are no other possible partitions of C.

[2] The set of components of a multi-cloud application can be obtained from the application MACM model by retrieving all the nodes of SaaS type. Our implementation adopts a graph database to maintain graph-based models and allows to get C in an automated way by means of an SQL-like query.

Listing 1: Initialization of L

```
void initL(int L[], int max[],
     int n) {
 int L[n];
 int max[n];
 for(int i=0; i<n; ++i){
  L[i]=1;
  max[i]=1;
 }
}
```

Listing 2: The Component-VMset generation Algorithm

```
bool nextL(int L[], int max[],
     int n) {
 int i=0;
 ++L[i];
 while((i<n-1)&&(L[i]>max[i
     ]+1)){
  L[i]=1;
  ++i;
  ++L[i];
 }
 if (i==n-1)
  return false;
 int t=L[i];
 for(i=i-1; i>=0; --i)
  max[i]=t;
return true;
}
```

The *VMset-Offering mapping generation algorithm* takes in input a logical deployment P and the set of offerings O and produces in output all the possible mappings of VM-sets in P onto available offerings, which can be represented in terms of the VMset-OFFERING mapping vectors Q, introduced in Section 4.2. It is worth noting that, given K VM-sets and m offerings, the algorithm simply generates all the possible K-element combinations (also referred to as the K-combinations) of m values.

In our implementation, the different Q vectors are generated one-by-one by the function *nextQ()* reported in Listing 4. The first VMset-OFFERING mapping vector Q, corresponding to assigning the same offering o_1 to all the VM-sets, is generated by the function initQ(), reported in Listing 3.

The function *nextQ()* takes in input K, m, the current Q and the current counter value, and generates a new vector Q, returning true if a new combination has been obtained, false otherwise.

Listing 3: Initialization of Q

```
void initQ(int K, int Q[]) {
 int Q[K];
 for(int i=0;i<K;i++)
   Q[i]=0;
}
```

Listing 4: The VMset-Offering Generation Algorithm

```
bool nextQ(int K, int m, int*
     counter, int Q[]) {
 *counter=*counter+1;
 int val = *counter;
 int zeros = 0;
 for(int i = 0; i < K; i++) {
   Q[i] = val\%m;
   zeros += Q[i];
   val = (int) val/m;
 }
 if (zeros!=0)
   return true;
 else
   return false;
}
```

The above functions are invoked by the deployment configuration generation algorithm in order to generate all possible deployment configurations $< L, Q >$, as illustrated in Listing 6. In particular, the function *nextL()* is invoked to generate a COMPONENT-VMset mapping vector L, which is successively used to generate all possible VMset-OFFERING mappings by means of function *nextQ()*. The procedure is iterated until no more vectors L are generated. The function *initDeployment()*, reported in Listing 5, must be invoked prior to the deployment generation in order to build the first vector L and start the whole process.

Listing 5: Initialization of deployment generation

```
int initDeployment(int L[],
     int Q[], int n, int max[],
      int* counter) {
  initL(L,max,n);
  initQ(1,Q);
  *counter=0;
}
```

Listing 6: The VMset-Offering Generation Algorithm

```
bool nextDeployment(int L[],
     int max[], int Q[], int n,
      int m, int* counter) {
 bool ok=true;
 if(!nextQ(getMaxValue(L), m,
      &counter, Q)){
  if(!nextL(L,max,n))
    return false;
  else{
    initQ(getMaxValue(L),Q);
    return true;
  }
 }
 return true;
}
```

5.2. Security assessment of deployment solutions

The security assessment carried out during the *Security Design* phase does not take into account deployment-related issues, which may have effect, in general, on the actual security guarantees offered by each component.

Let us consider the two deployment-specific MACM models in Figure 6, which represent two different deployment configurations for the simple application reported in Figure 2, composed of a web app component W, a database component DB and a component S offering generic services in SaaS mode. The model on the left represents a deployment where a single powerful VM offered by a CSP hosts all the components, while in the second one a VM is used to host S, and another VM offered by a different CSP hosts W and DB. It is worth noting that, although the components' SLATs do not vary, the final security policies implemented by the two deployments may be very different, depending on the security controls granted by involved CSPs and on the way these controls are affected by the deployment over multiple VMs.

The SLA Composition process combines the SLATs associated with each component of the application with the SLAs of the CSPs that provide the host-

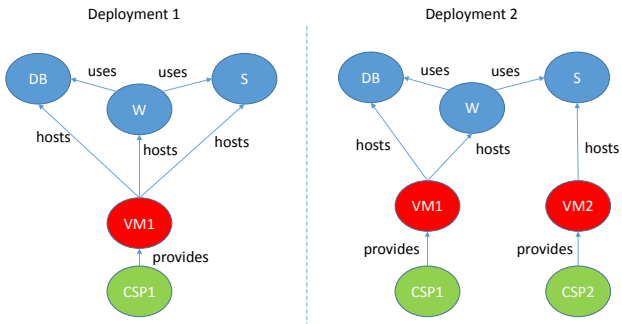

Figure 6: Alternative deployment configurations for a simple application

ing services, by applying composition rules to involved controls and control families, according to the relationships existing among components and selected providers.

In particular, the SLA Composition process assumes that security controls are independent of one another, and that their coverage can be verified separately. Hence, the application graph model is translated into a set of logical rules (first-order logic) that depend on how security controls are implemented on each node and that are suitably combined based on the graph structure. A security control is considered as correctly implemented, and it is thus included in the multi-cloud application SLA, only if it is implemented in all the services composing the application. Explaining the details of the SLA Composition process is out of the scope of this paper; the interested reader is referred to [42] for an extensive discussion about the composition techniques and rules. Figure 7 shows graphically the output expected from the SLA Composition process for the deployment configuration in Figure 6. From the deployment-specific security-annotated model, including the main relationships among nodes and the static security policies implemented by each component (i.e., the SLATs), the composition process builds the final application MACM. This specifies which are the actual security policies (i.e., the SLAs) that are enforced by each component in the selected physical deployment configuration. These policies can then be compared with the security requirements in order to verify their coverage (as

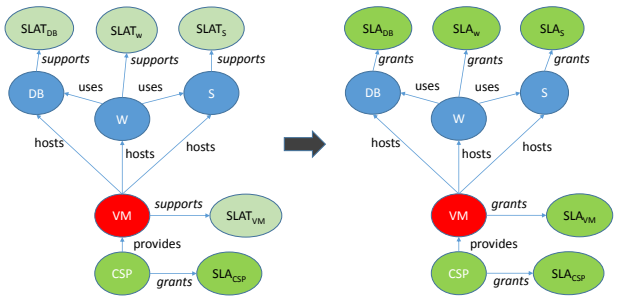

Figure 7: The SLA Composition Process

illustrated in Section 4.3).

5.3. Computation of the deployment score

In order to identify the best compromise between cost and security, we devised an optimization strategy able to take into account the preferences of the developer with respect to these criteria. It is clear that, depending on the specific application under development, the absolute or relative importance of the above parameters may substantially change. Due to regulatory compliance, for example, it may be essential to cover a certain set of security controls and to rely on providers with specific security certifications, regardless of the price to pay. In other cases, cost may be the primary aspect to consider when comparing different solutions.

Multiple criteria decision making is based on the formulation of mathematical optimization problems involving multiple objective functions to be optimized simultaneously. It is very useful in real-world problems (belonging to engineering, economics and logistics fields, for example), where optimal decisions have to be made in the presence of trade-offs between two or more conflicting objectives. In the literature there exist several optimization strategies for solving multi-objective problems [25, 41, 26]. Given the nature of the deployment problem and considering that the optimization has to be controlled by the developer,

we chose to adopt a multi-criteria decision strategy based on the flexible and powerful Analytic Hierarchy Process (AHP) methodology [45].

Indeed, the AHP offers a simple way to automate the process of decision making by taking into account different aspects of the possible solutions. The AHP makes it possible to compare different alternatives according to quantitative or qualitative criteria, which are assigned different (relative) levels of importance by the decision maker. Furthermore, the advantage of expressing the relative quality of a solution in terms of judgements (Low, Medium, High) makes the AHP suitable for the security context, being very close to the way in which security experts commonly describe and evaluate security levels.

The methodology entails the identification of a hierarchy, which in the simplest case is composed of three levels: the root represents the goal, the intermediate level includes the optimization criteria and the leaves represent the available alternatives, among which a choice must be done. In our case, the goal can be summarized as *"choose the best deployment configuration for the application's components"*, the criteria are represented by the overall security coverage and the cost, and the leaves correspond to the possible deployment alternatives. The algorithm used by the AHP methodology assigns a *weight* or *priority* in [0,1] to each node of the hierarchy. Priorities are assigned by performing pairwise comparisons among the nodes belonging to same level of the hierarchy with respect to their impact on each of the parent nodes. Comparisons are then performed (i) among the criteria with respect to the goal and (ii) among the alternatives with respect to each of the criteria. The priorities computed in this way are properly combined by means of addition and multiplication operations to obtain the final scores of the alternatives with respect to the final goal, which allows to rank them from the best to the worst solution.

It should be noted that determining all possible deployment alternatives in order to directly apply the AHP methodology to the simple 3-level hierarchy discussed above is not a viable solution, due to the complexity of deployment generation. Hence, we considered a different hierarchy that includes deployment *clusters* in its leaves instead of single deployments, as shown in Figure 8.

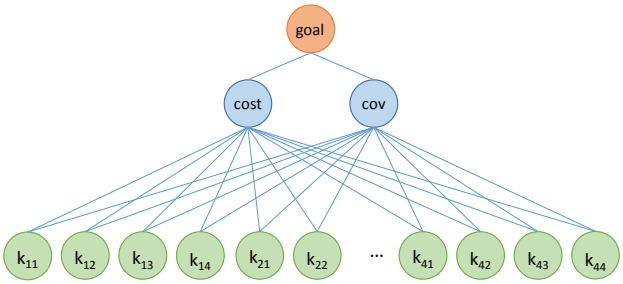

Figure 8: The AHP hierarchy used for deployment optimization

Although more complex strategies may be devised to build clusters, we considered a simple strategy based on the use of four different percentage classes, namely: $CL_1 = [0\%; 25\%)$, $CL_2 = [25\%; 50\%)$, $CL_3 = [50\%; 75\%)$ and $CL_4 = [75\%; 100\%]$ representing the possible intervals in which both the coverage and the cost values may fall. In particular, here coverage is measured in terms of the number of security controls that a given deployment configuration is able to enforce over the total number of controls of interest (e.g., the controls belonging to specific families). Alternatively, controls may be assigned different weights, and coverage may take into account how controls are covered based on their importance. In both cases, the information on the controls that are actually enforced at the application level is provided as the result of the SLA Composition process. The cost, instead, is compared to the maximum possible cost to be payed. This maximum cost is obtained by selecting the most expensive deployment that satisfies the performance requirements defined in the functional design phase. Cost classes are built by considering cost in positive terms, namely in terms of the percentage of money saved as compared to the maximum cost.

By combining the four classes, it is possible to obtain 16 clusters $k_{ij} =< COV_i, COST_j >$, with $i, j = 1, \ldots, 4$, where COV_i is the coverage class and $COST_j$ is the cost class.

Our optimization strategy works as follows:

1. The above hierarchy is pre-evaluated in order to assign priorities to the clusters with respect to their impact on the criteria, and to the criteria with respect to their impact on the goal, based on the preference levels expressed by the developer during the design phase. Let $w(k_{ij}, COV)$ and $w(k_{ij}, COST)$ be the priorities computed for the cluster k_{ij} with respect to coverage and cost, respectively, and let $w(COV, goal)$ and $w(COST, goal)$ be the priorities of coverage and cost with respect to the goal.

2. For each deployment solution D_n identified in the first step of the optimization process, its (percentage) values of coverage and cost are computed. As stated before, the coverage percentage is evaluated with respect to the controls of interest, while the cost percentage is computed with respect to the maximum possible cost. Let them be $COV(D_n)$ and $COST(D_n)$, respectively.

3. A weight vector $W_n = [w_n(COV), w_n(COST)]$ is assigned to deployment D_n, where $w_n(COV) = COV(D_n)$ and $w_n(COST) = 1 - COST(D_n)$. The value of $w_n(COST)$ represents the the amount of money that can be saved with respect to the maximum cost when choosing the deployment D_n.

4. The cluster to which D_n belongs is identified based on its coverage and cost values. Let it be k_{ij}.

$$w_n(COV) \in CL_i \wedge w_n(COST) \in CL_j \Rightarrow D_n \in k_{ij} \tag{2}$$

5. The score of the deployment $score(D_n)$ is computed as:

$$\begin{aligned} score(D_n) = w_n(COV) \cdot w(k_{ij}, COV) \cdot w(COV, goal) + \\ w_n(COST) \cdot w(k_{ij}, COST) \cdot w(COST, goal) \end{aligned} \tag{3}$$

The pairwise comparisons needed to compute the priorities at step 1 of the above process can be made in any sequence, but we considered first the evaluation of the criteria with respect to the goal and then the evaluation of the alternatives (clusters) with respect to the criteria. The scale used to carry out the com-

parisons is the AHP fundamental scale reported in Table 1, which considers 5 different levels of relative importance.

Intensity of importance	**Definition**	**Meaning**
1	Equal importance	Equal contribution to the objective
3	Moderate importance	Experience and judgement moderately favor one element over the other
5	Strong importance	Experience and judgement strongly favor one element over the other
7	Very strong importance	One element is very strongly favored over the other, its dominance is demonstrated in practice
9	Extreme importance	One element is extremely favored over the other, with the highest possible evidence

Table 1: The scale for pairwise comparisons

The preference levels, set according to the values reported in Table 1, are specified by the developer and used to fill a square symmetric diagonal matrix (the *Importance Matrix*). Mathematically speaking, priorities are computed as the values in the matrix's principal right eigenvector.

6. A Case Study

In the following, we will illustrate the main steps of the development process discussed in Section 2 by means of a simple case study application. In particular, we will discuss the requirements elicited during the *Functional* and *Security Design* phases, and show the main steps behind the *Optimal Deployment Identification* phase.

6.1. Case study application design

Let us consider a simple cloud application that allows users to upload and share personal documents through a web interface and to talk with one another thanks to a live chat service. The application is made of three components:

- *Application Core*. It is the web application that offers the upload and visualization functionalities to the users.

- *Application Database*. It is the database used by the application core to store the users' data.

- *Application Chat Service*. It is a live chat service, offered in SaaS mode, used by the application core to provide its users with instant messaging functionalities.

During the *Functional Design* phase, the developer first defines the high-level architecture of the application by means of the deployment-independent MACM (the model of the case study application matches the one of the example application in Figure 2, where the generic component `S` is substituted with the Chat Service component), and then specifies the type of each component and its performance requirements. The type of each component allows to identify the security threats the component may be subject to, as resulting from available vulnerability databases and assessment tools, and is used in the subsequent *Security Design* phase. In this case study, web applications, storage services and SaaS services represent three distinct component types, each characterized by their own threat model and by a specific set of security guidelines to be applied for risk mitigation.

During the design phase, besides specifying the components' types, the developer is also asked to provide a high-level characterization of the components in terms of their resource demand (i.e., in terms of CPU cores and RAM capacity). Let us assume that the developer or an expert, after an analysis of the expected workload, gives the following requirements for the given application:

Component	**Requirements**
Core	CPU: 0.7 core; RAM: 512MB
Database	CPU: 0.5 core; RAM: 2048MB
Chat Service	CPU: 0.3 core; RAM: 1024MB

Table 2: Components' resource demand

A value of '0.7 core' in the table for the CPU means that the component needs the 70% of the computation power of a single CPU core. The required amount of CPU and RAM constitute the resource demand of each component, generically identified as $creq(c_i)$ (cfr Section 4).

After the *Functional Design* phase, the *Security Design* phase takes place. Here, the developer has to (i) identify the application components' security requirements by carrying out a risk analysis process, and (ii) determine the security policies virtually enforced by the components, based on their internal implementation.

The elicitation of security requirements by means of the risk analysis process (named SLA Generation process) has been illustrated in [7], where we also discussed its implementation through a suitable tool (the SLA Generator application). As for the pre-deployment security assessment of components, we prepared ad-hoc questionnaires based on public security guidelines, aimed at identifying which security controls are actually taken into account by the components' implementation. For conciseness' sake, let us consider only the controls belonging to the NIST families *AC-Access Control* and *IA-Identity Management*, and let us assume that the security requirements identified for the application components are those reported in Table 3, expressed in terms of Required Controls, and that the Enforced Controls are those reported in Table 4. These sets of controls respectively represent the $RSEC(c_i)$ and $PSEC(c_i)$ inputs, as explained in Section 4.

Component	**Required Controls**
Core	AC-1, AC-6, AC-7, AC-9, AC-10, AC-11, AC-12, IA-1, IA-9
Database	AC-1, AC-16, AC-17, AC-23
Chat Service	AC-1, AC-2, AC-3, IA-1, IA-4, IA-5, IA-7

Table 3: Components' security requirements

In the above described steps, part of the inputs needed by the *Optimal Deployment Identification* phase have been identified. The remaining inputs are those related to the available offerings, in terms of their capacity, cost and

Component	Enforced Controls
Core	AC-1, AC-2, AC-3, AC-4, AC-5, AC-6, AC-7, AC-8, AC-9, AC-10, AC-11, AC-12, AC-13, AC-14, AC-15, AC-19, AC-20, AC-21, AC-24, AC-25, IA-1, IA-2, IA-4, IA-5, IA-6, IA-8, IA-9, IA-10, IA-11
Database	AC-1, AC-2, AC-3, AC-4, AC-5, AC-6, AC-7, AC-9, AC-10, AC-11, AC-12, AC-14, AC-15, AC-16, AC-17, AC-21, AC-22, AC-23, AC-24, IA-1, IA-2, IA-4, IA-5, IA-6, IA-8, IA-9, IA-10
Chat Service	AC-1, AC-2, AC-3, AC-4, AC-5, AC-6, AC-7, AC-9, AC-10, AC-11, AC-12, AC-16, AC-17, AC-24, IA-1, IA-2, IA-3, IA-4, IA-5, IA-6, IA-7, IA-9

Table 4: Components' implemented security controls

security features.

Offering	CSP	Capacity	Cost	Security controls
o_1	CSP1	1core, 1024MB	0,1	AC-1, AC-2, AC-5, AC-6, AC-7, AC-9, AC-10, AC-11, AC-12, AC-13, AC-14, AC-21, AC-22, AC-24, IA-1, IA-2, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9, IA-10, IA-11
o_2	CSP1	2core, 2048MB	0,2	AC-1, AC-2, AC-3, AC-5, AC-6, AC-7, AC-9, AC-10, AC-11, AC-12, AC-16, AC-17, AC-23, IA-1, IA-2, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9
o_3	CSP2	2core, 4096MB	0,35	AC-1, AC-2, AC-6, AC-7, AC-9, AC-10, AC-11, AC-13, AC-14, IA-1, IA-2, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9, IA-10, IA-11
o_4	CSP3	1core, 1024MB	0,2	AC-1, AC-2, AC-5, AC-6, AC-7, AC-9, AC-10, AC-11, AC-12, AC-13, AC-14, AC-21, AC-22, AC-24, IA-1, IA-2, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9, IA-10, IA-11
o_5	CSP3	2core, 4096MB	0,5	AC-1, AC-2, AC-3, AC-4, AC-5, AC-6, AC-7, AC-8, AC-9, AC-10, AC-11, AC-12, AC-13, AC-14, AC-15, AC-16, AC-17, AC-19, AC-20, AC-21, AC-22, AC-23, AC-24, AC-25, IA-1, IA-2, IA-3, IA-4, IA-5, IA-6, IA-7, IA-8, IA-9, IA-10, IA-11

Table 5: Offerings

Let us consider the offerings reported in Table 5. For each offering, the table specifies the provider, the capacity (in terms of CPU cores and MB of RAM), the hourly cost (in cents) and the enforced security controls.

6.2. Case study application deployment optimization

The procedure shown above makes it possible to obtain all the inputs for the *Optimal Deployment Identification* phase. As mentioned in Section 5.3, this phase requires the pre-evaluation of the AHP hierarchy shown in Figure 8, so as to compute all the relative priorities of the hierarchy nodes. These are needed to compute the score of the candidate deployment solutions in the optimization process. To compute the node priorities, we first carry out pairwise comparisons among criteria with respect to their impact on the goal, and then among clusters with respect to their impact on the criteria, by using the scale reported in Table 1. In this case study, let us assume that the developer considers security essential, and so security coverage is *very strongly favoured over* cost. According to the AHP scale and recalling how the importance matrix is built, the *Intensity of Importance of coverage* with respect to cost is 7, while the *Intensity of Importance of cost* with respect to coverage is 1/7. With these values, it is easy to verify that the coverage and cost priorities are $w(COV, goal) = 0.875$ and $w(COST, goal) = 0.125$, respectively.

For what regards the priorities of clusters with respect to their impact on each of the criteria, we have built *the importance matrix* related to the pairwise comparisons among the clusters with respect to coverage, by assigning the intensity of importance based on the coverage class each cluster belongs to. This way, clusters belonging to the same coverage class are considered equally important, while those belonging to different classes are assigned a relative importance from moderate (level 3) to very strong (level 7). The importance matrix related to the comparisons with respect to cost has been built in the same way. The matrices are not reported here due to space limitations.

With the importance matrices set up, it is possible to compute the *priorities of each cluster* with respect to their impact on the considered criteria. These

are reported in Table 6.

$\mathbf{k}_{ij}$	$\mathbf{w(k}_{ij}\mathbf{,COV)}$	$\mathbf{w(k}_{ij}\mathbf{,COST)}$
k11	0.014222	0.139473
k12	0.030468	0.139473
k13	0.065836	0.139473
k14	0.139473	0.139473
k21	0.014222	0.065836
k22	0.030468	0.065836
k23	0.065836	0.065836
k24	0.139473	0.065836
k31	0.014222	0.030468
k32	0.030468	0.030468
k33	0.065836	0.030468
k34	0.139473	0.030468
k41	0.014222	0.014222
k42	0.030468	0.014222
k43	0.065836	0.014222
k44	0.139473	0.014222

Table 6: Clusters' priorities with respect to criteria

The above priorities are used to compute the score of each candidate deployment. It should be noted that, with three components to deploy and five available offerings, there are 205 possible deployment solutions. Although this number may seem relatively low from the point of view of the complexity of the problem, it must be taken into account that, for each candidate deployment, a validity check must be done before computing the score. The validity verification includes a check of resource demand constraints, which is quite straightforward, and a check of security constraints, which instead requires the execution of the SLA Composition process that is quite complex and time consuming. Indeed, this operation would require too much effort for a developer, if performed manually, and the adoption of the proposed approach enables to completely automate the security check verification.

Returning to the case study, six solutions can be immediately discarded since

they do not fulfill resource demand requirements. These include the solutions that allocate all the components on the same instance, regardless of the involved offering, plus the solution $< [1,2,2],[1,1] >$, which uses two instances of the same offering (o_1) and deploys the Core component (c_1) on the first one, and the Database (c_2) and the Chat Service (c_3) together, on the second one, since this solution does not meet requirements on resource demand.

There are many solutions that fulfill resource demand requirements but do not fulfill security requirements. Among them, as an example, let us consider the candidate deployment $< [1,2,2],[1,3] >$, which allocates c_1 on an instance belonging to o_1, and c_2 and c_3 on an instance belonging to o_3. When launching the SLA Composition process, it turns out that controls AC-16, AC-17 and AC-23, required by the Chat Service component and implemented internally according to the assessment reported in Table 4, would no longer be covered with this deployment. In fact, these controls are not covered by o_3, while the adopted composition rules require their implementation both at component and VM level. Moreover, according to composition rules, a control of the `AC` family can be considered as correctly implemented on a component c if it is correctly implemented not only by c and by the offering on which it is deployed, but also by all the components `used` by c and on the respective offerings. This does not happen for control AC-12, which is required by c_1 but is not covered by o_3, selected to deploy c_2 and c_3, which are used by c_1.

The optimal deployment found is $D =< [1,2,2],[1,2] >$. This implies the deployment of c_1 on an instance belonging to the offering o_1, and c_2 and c_3 on an another instance belonging to o_2. The solution is valid and has a cost equal to 0.3, which corresponds to the 20% of the maximum possible cost (i.e., 1.5), obtained by deploying each component on a separate instance of the most expensive offering, namely o_5. For what regards coverage, we considered it respect to the `AC` control family, and computed the percentage of controls of this family that result covered by each deployment solution after composition. For the optimal solution D, the SLA Composition process outlined that 9 controls are covered out of the available 25

of the AC family. The coverage, therefore, results to be equal to 36%.

In conclusion, the just evaluated cost and coverage percentages are used to assign the weight vector to the deployment D, represented by $W_D = [0.36, 0.8]$ (cost is represented in positive terms as discussed in Section 5.3, i.e., 1-0.20). Such vector, in turn, allows to identify the *coverage class* (CL_2) and the *cost class* (CL_4) to which the deployment belongs, and to determine the reference cluster, which in this case is $k_{2,4}$. Finally, by applying the formula (3) in Section 5.3, it is straightforward to compute the final score of D, which is equal to 0.0505.

This simple case study has illustrated the main advantages of the proposed approach. First, in the design phase a completely automated way was proposed to gather security requirements and match them with needed standard security controls to apply and configure in order to mitigate potential security risks. Second, the validity and feasibility of a possible deployment is automatically performed with the adoption of the SLA composition process. Finally, an optimal and feasible deployment can be identified and evaluated from a security and cost perspective.

7. Related Work

One of the main contributions of this paper is the formalization of the deployment problem. This problem has been widely addressed in the literature with approaches often borrowed from other contexts (e.g., task scheduling and resource allocation in parallel and distributed systems). These approaches have been previously adopted in cluster and grid computing, and later adapted to clouds.

Applications running in the cloud are subject to many non-functional requirements, which typically are stated in an informal way. For example, the choice among the configuration and deployment options can be influenced by factors such as cost, performance, scalability, security, etc. These factors affect each other, making the optimization process very challenging. In this paper, the focus is on a single non-functional requirement, security, which currently is

still considered to be the biggest flaw of cloud environments. The paper [22] is the literature contribution closest to our work. It presents a multi-criteria optimization method (based on AHP, as ours) that relies on non-functional requirements to select the best architectural options for deploying applications in the cloud. However, the paper takes into account only efficiency, cost and scalability as non-functional requirements, but security is neglected.

Other non-functional parameters linked to deployment choices are considered in [48], whose focus is on the availability of applications deployed on clouds that are characterized by relatively weak availability guarantees. The paper [28], instead, deals with energy efficiency. The authors propose to use power consumption as a metric to drive cross-cloud deployment in heterogeneous multi-clouds.

As a matter of fact, most of the deployment/placement optimization studies and algorithms in the literature consider a stable infrastructure, and so they are not immediately exploitable in the cloud and multi-cloud context. Among the recent contributions that study the deployment problem in clouds, the paper [32] tackles the problem of the optimum deployment taking into account the data protection problems linked to the cloud inherent multi-tenancy. Reference [24] presents a vision for an autonomic deployment system. In particular, it details the architecture of a learning automata based reasoning component envisioned to be able to provide feasible allocations of the artefacts on the available infrastructures.

CloudOpt [29] is a comprehensive approach to find optimal deployments for large service centres and clouds. It uses a combination of bin-packing, mixed integer programming and performance models to make decisions affecting diverse and interacting goals, including satisfaction of different service level agreements for many different applications (e.g., response time, workload, throughput), memory requirements and availability, costs and power consumption. It is scalable and extendible to new objectives. In [18], moreover, the authors initiate the study of customer-controlled placement gaming, including strategies by which customers exploit performance heterogeneity to lower their costs. In particular,

they start from the assumption that not all VM instances are created equal: Distinct underlying hardware differences, contention, and other phenomena can result in vastly differing performance across supposedly equivalent instances. They developed a formal model for placement strategies and evaluate potential strategies via simulation. In [43], the authors investigate a resource management framework that consists of both architecture and scheduling policies with regards to multi-tenancy issues, especially scalability and shared resources for scheduling compute-intensive workflow applications. An interesting resource management framework is developed to separate resource management policies from the control mechanisms required to implement them, introducing the concept of Workflow-as-a-Service that can be used for any software components, including security controls. The paper [39] deals with the prediction of the cost of the cloud deployment of applications subject to performance-oriented service-level agreements. Reference [50] introduces ClouDiA, a deployment advisor that minimizes either the latency or the longest critical path between application nodes.

Papers more focused on the algorithms for optimization of the deployment onto clouds include [16], which describes a toolchain that automates the assembly and deployment of complex applications in the cloud. Given the set of available components together and their requirements, and the maximal amount of virtual resources to be committed, it places components in an optimal manner using the minimal number of available machines and automatically deploys the complete system. Reference [15] presents a nature-inspired distributed heuristic algorithm for deployment optimization. The algorithm is able to minimize execution or communication costs between the application software components. The paper [30] deals with the optimization of the placement of VMs on physical compute nodes. This is a problem different from the one discussed in this paper, where the objective is to optimize the deployment of software on VMs, whose allocation of physical nodes is neglected. The use of genetic optimizations for the deployment and the possible reconfiguration of software in the cloud is discussed in [20].

New work is recently appearing that takes into account the heterogeneity and dynamism of offerings in a multi-cloud environment, as well as the specific cloud customers needs (costs, security, quality, etc.). The exploitation of the inherent infrastructure elasticity (i.e, the dynamical scaling up/down of the number of deployed software components in order to adapt to the actual workload) is out of the scope of this paper. The approach proposed here is suitable only applications with a fixed number of software components to be deployed. If this number elastically changes, the optimization procedure has to be performed again. Among the papers tackling the elastic" deployment problem, it is worth mentioning reference [17], which studies the deployment in light of the highly-dynamical nature of the cloud. It presents a solution to leverage multi-cloud environments, rebalancing the workload by replacing instances in lower-preferred clouds with instances in higher-preferred clouds. The objective of the paper [23] is instead the deployment of three-tier applications across multiple clouds. The proposed resource provisioning and load distribution algorithms can heuristically optimize overall cost and response delays. The paper [49] introduces an algorithm to minimize costs and latency of application deployment by exploiting multi-cloud environments. Finally, in [31], the authors introduce a technique for application deployment in multi-cloud. It uses deployment configuration tool (Chef [11]) to automate the deployment and a DSL language to model the application and drive the deployment of the application. No optimization is addressed in this paper, but specific constraints from the multi-cloud environment are analysed. Game-theoretic approaches for the optimization of the dynamic management of cloud/multi-cloud resources are presented in references [19] and [1]. The search of the most trusted node for providing new resources and the maximization of the perceived QoS in dynamical cloud infrastructures are dealt with in [34] and [12], respectively.

Overall, none of the aforementioned studies takes into account the security level associated to the possible deployments. The approach focused on security proposed in this paper tries to put together the most innovative results from recent European projects and research activities, introducing a security-

driven process for the optimization of the deployment of software components in a multi-cloud environment. As previously discussed, the whole process relies upon the adoption of Security SLAs as a means to express the security guarantees offered by services and applications, with these SLAs representing the glue among the different process phases.

Providing and granting security features in cloud through Security SLAs is a challenging task, as firstly outlined in [2, 3], where the authors discussed how cloud SLAs could be extended to cover security aspects, allowing composition of cloud services from several service providers with a defined security level. Recently, attempts to automatically manage Security SLAs have been devised in a few European projects, such as Contrail [13], OPTIMIS [37], SPECS [8, 6] and MUSA [44, 7], as described in [5].

SPECS, in particular, proposed an easy-to-use Security SLA model that allows to declare the security-related attributes of a service in terms of security controls and to specify how to monitor them through proper security metrics. The MUSA project, whose activities have inspired this paper, extended the SPECS SLA model with the introduction of the SLA Composition process discussed in [42], and the adoption of multi-cloud specific security controls and metrics.

8. Conclusions and Future Work

In this paper, we have shown the high potential of multi-cloud strategies for the development of secure and cost-effective applications, and proposed a comprehensive approach able to support a developer in the design, development and deployment of (multi-)cloud applications with specific performance and security requirements. Our main contribution is the definition of a *security-by-design development process*, in which security is taken into account from the early stages of development and the deployment of applications is identified through an optimization process aimed at fulfilling the existing performance and security constraints, while ensuring the best compromise between deployment

cost and overall level of security achieved.

The proposed process is very complex and some work is still to be done to make it completely compliant with the current cloud landscape. First of all, we plan to improve the optimal deployment problem formalization in order to take into account additional aspects of actual cloud environments. These include the adoption of containers, and the mapping of components to SaaS services. Moreover, more sophisticated performance and cost models and further deployment models have to be considered.

As mentioned several times in this paper, the proposed optimization strategy is focused on small-size problems; however, the approach adopted remains valid when different optimization algorithms are used. In the next future, we plan to test new heuristic optimization algorithms in order to be able to manage more complex problems. Finally, we are in the process of building a comprehensive tool for the implementation of the whole development process, in order to provide developers with a fully-automated solution.

References

[1] Ardagna, D., Ciavotta, M., & Passacantando, M. (2017). Generalized nash equilibria for the service provisioning problem in multi-cloud systems. *IEEE Transactions on Services Computing*, *10*, 381–395.

[2] Bernsmed, K., Jaatun, M. G., Meland, P. H., & Undheim, A. (2011). Security SLAs for Federated Cloud Services. In *Proc. of the 6th International Conference on Availability, Reliability and Security (ARES)* (pp. 202–209).

[3] Bernsmed, K., Jaatun, M. G., & Undheim, A. (2011). Security in Service Level Agreements for Cloud Computing. In *Proc. of the 1st International Conference on Cloud Computing and Services Science (CLOSER)* (pp. 636–642).

[4] Casola, V., De Benedictis, A., Erascu, M., Modic, J., & Rak, M. (2017).

Automatically enforcing security SLAs in the cloud. *IEEE Transactions on Services Computing*, *10*, 741–755.

[5] Casola, V., De Benedictis, A., & Rak, M. (2015). On the adoption of security SLAs in the cloud. *Lecture Notes in Computer Science*, *8937*, 45–62.

[6] Casola, V., De Benedictis, A., & Rak, M. (2015). Security monitoring in the cloud: An SLA-based approach. In *Proceedings - 10th International Conference on Availability, Reliability and Security, ARES 2015* (pp. 749–755).

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

[8] Casola, V., De Benedictis, A., Rak, M., & Villano, U. (2016). SLA-Based Secure Cloud Application Development: The SPECS Framework. In *Proceedings - 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2015* (pp. 337–344).

[9] CeRICT (2017). The deployment optimization tool - Bitbucket Repository. URL: `https://bitbucket.org/cerict/securitycontrolgraphs`.

[10] CeRICT (2017). The SLA Generation application - Bitbucket Repository. URL: `https://bitbucket.org/cerict/sla-generator-v2`.

[11] Chef Software Inc. (2018). Chef Web Site. `https://www.chef.io/chef/`.

[12] Comi, A., Fotia, L., Messina, F., Rosaci, D., & Sarné, G. M. (2016). A partnership-based approach to improve qos on federated computing infrastructures. *Information Sciences*, *367*, 246–258.

[13] Contrail Consortium (2017). The Contrail project web site. URL: `http://www.contrail-project.eu`.

[14] CSA (2015). Cloud controls matrix v3.0. https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3/.

[15] Csorba, M. J., & Heegaard, P. E. (2010). Swarm intelligence heuristics for component deployment. *Lecture Notes in Computer Science*, *6164*, 51–64.

[16] Di Cosmo, R., Lienhardt, M., Treinen, R., Zacchiroli, S., Zwolakowski, J., Eiche, A., & Agahi, A. (2014). Automated synthesis and deployment of cloud applications. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering - ASE '14* (pp. 211–222).

[17] Duplyakin, D., Marshall, P., Keahey, K., Tufo, H., & Alzabarah, A. (2013). Rebalancing in a multi-cloud environment. In *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing* Science Cloud '13 (pp. 21–28). ACM.

[18] Farley, B., Juels, A., Varadarajan, V., Ristenpart, T., Bowers, K. D., & Swift, M. M. (2012). More for your money: Exploiting performance heterogeneity in public clouds. In *Proceedings of the Third ACM Symposium on Cloud Computing* SoCC '12 (pp. 20:1–20:14).

[19] Ficco, M., Esposito, C., Palmieri, F., & Castiglione, A. (2018). A coral-reefs and Game Theory-based approach for optimizing elastic cloud resource allocation. *Future Generation Computer Systems*, *78*, 343–352.

[20] Frey, S., Fittkau, F., & Hasselbring, W. (2013). Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *Proceedings of the 2013 International Conference on Software Engineering* ICSE '13 (pp. 512–521). IEEE Press.

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

[22] Goncalves, R., Rolim, T., Sampaio, A., & Mendonca, N. C. (2015). A multi-criteria approach for assessing cloud deployment options based on

non-functional requirements. In *Proceedings of the ACM Symposium on Applied Computing* (pp. 1383–1389). volume 13-17-Apri.

[23] Grozev, N., & Buyya, R. (2014). Multi-Cloud Provisioning and Load Distribution for Three-Tier Applications. *ACM Transactions on Autonomous and Adaptive Systems*, *9*, 1–21.

[24] Horn, G. (2013). A vision for a stochastic reasoner for autonomic cloud deployment. In *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies - NordiCloud '13* (pp. 46–53). ACM Press.

[25] Islam, M. R., & Habiba, M. (2012). Dynamic scheduling approach for data-intensive cloud environment. In *2012 International Conference on Cloud Computing Technologies, Applications and Management (ICCCTAM)* (pp. 179–185).

[26] Jiao, L., Li, L., Shang, R., Liu, F., & Stolkin, R. (2013). A novel selection evolutionary strategy for constrained optimization. *Information Sciences*, *239*, 122 – 141. doi:`https://doi.org/10.1016/j.ins.2013.03.002`.

[27] Kundakcioglu, O. E., & Alizamir, S. (2009). Generalized assignment problem. In C. A. Floudas, & P. M. Pardalos (Eds.), *Encyclopedia of Optimization* (pp. 1153–1162). Boston, MA: Springer US.

[28] Kurpicz, M., Sobe, A., & Felber, P. (2014). Using power measurements as a basis for workload placement in heterogeneous multi-cloud environments. In *Proceedings of the 2Nd International Workshop on CrossCloud Systems* CCB '14 (pp. 6:1–6:6). ACM.

[29] Li, J. Z. W., Woodside, M., Chinneck, J., & Litoiu, M. (2011). CloudOpt: Multi-goal optimization of application deployments across a cloud. In *Network and Service Management (CNSM), 2011 7th International Conference on* (pp. 1–9).

[30] López-Pires, F., Barán, B., Amarilla, A., Benítez, L., Ferreira, R., & Zalimben, S. (2016). An experimental comparison of algorithms for virtual machine placement considering many objectives. In *Proceedings of the 9th Latin America Networking Conference* LANC '16 (pp. 1–8). ACM.

[31] Lu, H., Shtern, M., Simmons, B., Smit, M., & Litoiu, M. (2013). Pattern-based deployment service for next generation clouds. In *2013 IEEE Ninth World Congress on Services* (pp. 464–471).

[32] Mann, Z. A., & Metzger, A. (2017). Optimized cloud deployment of multi-tenant software considering data protection concerns. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* CCGrid '17 (pp. 609–618). IEEE Press.

[33] Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. *NIST Special Publication*, *800*, 145.

[34] Messina, F., Pappalardo, G., Rosaci, D., Santoro, C., & Sarné, G. M. (2016). A trust-aware, self-organizing system for large-scale federations of utility computing infrastructures. *Future Generation Computer Systems*, *56*, 77–94.

[35] MUSA Consortium (2015). The MUSA project web site. `http://musa-project.eu/`.

[36] National Institute of Standards and Technology (2013). *SP 800-53 Rev 4: Recommended Security and Privacy Controls for Federal Information Systems and Organizations*. Technical Report. URL: `http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf`.

[37] Optimis Consortium (2017). The Optimis project web site. URL: `http://www.optimis-project.eu`.

[38] PaaSage Consortium (2016). The CAMEL web site. http://camel-dsl.org/.

[39] Perez-Palacin, D., Calinescu, R., & Merseguer, J. (2013). Log2cloud: Log-based prediction of cost-performance trade-offs for cloud deployments. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* SAC '13 (pp. 397–404). New York, NY, USA: ACM.

[40] Petcu, D. (2013). Multi-cloud: Expectations and current approaches. In *Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds* MultiCloud '13 (pp. 1–6). ACM.

[41] Qi, R., & Yen, G. G. (2017). Hybrid bi-objective portfolio optimization with pre-selection strategy. *Information Sciences*, *417*, 401 – 419. doi:`https://doi.org/10.1016/j.ins.2017.07.018`.

[42] Rak, M. (2017). Security assurance of (multi-)cloud application with security SLA composition. *Lecture Notes in Computer Science*, *10232*, 786–799.

[43] Rimal, B. P., & Maier, M. (2016). Workflow Scheduling in Multi-Tenant Cloud Computing Environments. *IEEE Transactions on Parallel and Distributed Systems*, *9219*, 1–1.

[44] Rios, E., Iturbe, E., Orue-Echevarria, L., Rak, M., & Casola, V. (2015). Towards self-protective multi-cloud applications: MUSA-a holistic framework to support the security-intelligent lifecycle management of multi-cloud applications. In *CLOSER 2015 - 5th International Conference on Cloud Computing and Services Science, Proceedings* (pp. 551–558).

[45] Saaty, T. L. (1990). How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, *48*, 9 – 26.

[46] SINTEF (2016). The CloudML Wiki web site. https://github.com/SINTEF-9012/cloudml/wiki.

[47] SPECS Consortium (2013). The SPECS project web site. `http://specs-project.eu/`.

[48] Xu, X., Lu, Q., Zhu, L., Li, Z., Sakr, S., Wada, H., & Webber, I. (2013). Availability analysis for deployment of in-cloud applications. In *Proceedings of the 4th International ACM Sigsoft Symposium on Architecting Critical Systems* ISARCS '13 (pp. 11–16). ACM.

[49] Yaw, S., Howard, E., Mumey, B., & Wittie, M. P. (2015). Cooperative group provisioning with latency guarantees in multi-cloud deployments. *ACM SIGCOMM Computer Communication Review*, *45*, 4–11.

[50] Zou, T., Le Bras, R., Salles, M. V., Demers, A., & Gehrke, J. (2015). ClouDiA: a deployment advisor for public clouds. *VLDB Journal*, *24*, 633–653.