Policy-based Deployment in a Hybrid and Multicloud Environment

Giuseppe Di Modica[1], Orazio Tomarchio[1], Hao Wei[2] and Joaquin Salvachua Rodriguez[2]
[1]*Department of Electrical, Electronic and Computer Engineering, University of Catania, Catania, Italy*
[2]*Departamento de Ingeniera de Sistemas Telemáticos, Universidad Politécnica de Madrid, Madrid, Spain*

Keywords: Policy based Management, Multi Cloud, Cloud Provisioning, Cloud Orchestration, TOSCA, BPMN.

Abstract: Hybrid and multi-cloud become prominent infrastructure strategy of enterprise. However, the complexity of such infrastructure is a considerable challenge. It is common to see that multi-cloud infrastructure is divided into several smaller units to facilitate the management. The division criteria are geolocation, cost, security, etc. Therefore, how to manage application deployment in such partitioned environment is an intriguing topic of multi-cloud management. We propose a policy-based deployment in multi-cloud infrastructure, which contains policy evaluation and TOSCA standard based orchestration. The system architecture is introduced and a case study with two empirical scenarios is discussed. The results indicate that the proposed policy-based deployment is useful in finding suitable resource and improving deployment efficiency.

1 INTRODUCTION

Based on deployment models, cloud computing can be classified into private cloud, public cloud, and hybrid cloud (Mell and Grance, 2010). The hybrid cloud model combines the public cloud and private cloud, and it delivers the best of both types of clouds. A deep analysis of the economics of cloud computing suggested that for enterprise hybrid cloud can offer the best economic benefits (Weinman, 2016). In addition to this, the compliance issue, legacy IT system, etc. make the hybrid cloud model the optimal choice for enterprise. A RightScale survey in 2017 (RightScale, 2017) confirms that 67% of enterprises worldwide currently adopt hybrid cloud.

Despite the fast growth of cloud computing, the barriers of adopting a cloud solution still exist. Cloud Administrators' main concerns regard which technology best suites the enterprise needs, which third party cloud provider is to be preferred, how to avoid being locked-in, how to correctly and smoothly integrate geographically distributed private resources and public resources, how to enforce the same security level across all resource domains (Di Modica and Tomarchio, 2016). When in particular a hybrid approach has been adopted, the complexity of the management gets higher. In this case a good strategy is to split the multi-cloud environment into multiple infrastructure domains in a way that simplifies the management and, at the same time, preserves the flexibility. This paper proposes a policy based application deployment method, which aims to relieve the administrative burden of the cloud service provisioning and accelerate the deployment of applications in hybrid cloud environments. The proposed approach separates the non-functional requirements from the functional ones of cloud applications. The former are addressed by means of policies, while a TOSCA-based cloud provisioning framework caters for the latter. To further study the insights of the proposal, realistic experiments were conducted on a concrete case study. The experimental results elucidate the viability of the proposed approach.

The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 sketches the proposed system's architecture. Section 4 presents an overview of proposed policy based application deployment model. A case study experiment is discussed in Section 5. Section 6 concludes the paper and anticipates the future work.

2 RELATED WORK

Many cloud industry players have developed cloud management platforms for automating the provisioning of cloud services (e.g. Amazon

Di Modica, G., Tomarchio, O., Wei, H. and Rodriguez, J.
Policy-based Deployment in a Hybrid and Multicloud Environment.
DOI: 10.5220/0007726503880395
In *Proceedings of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, pages 388-395
ISBN: 978-989-758-365-0

CloudFormation(Amazon, 2016), Rightscale Cloud Management Platform(Rightscale, 2016), RedHat CloudForms(RedHat, 2016), IBM Cloud Orchestrator(IBM, 2016)). The most advanced platforms also offer services and tools for the management of cloud applications' lifecycle. None of these commercial products are open to the community, nor the solutions they offer are portable across third-party providers.

As to open-source Cloud orchestration frameworks, the OpenStack platform [1] includes an Orchestration service which provides a template-based way to describe a cloud application based on TOSCA. The templates, written according to the HOT (*Heat Orchestration Template*) template language[2], enable to create most OpenStack resource types, such as instances, storage volumes, networking information, security policies, and all the other required cloud infrastructure to run an application. Another notable example of orchestration platform in the open source domain is Cloudify(GigaSpaces, 2016): it allows to model applications and automate their entire lifecycle through a set of built-in workflows.

Some minor research initiatives have come up with open source solutions as well. Roboconf (Pham et al., 2015) is an open sourced hybrid cloud orchestrator for application deployment. It implements the basic administration mechanisms which are called Autonomic Computing Systems (ACS). Roboconf has simple and extensible design, many of its components are reusable. A Domain Specific Language (DSL) is also presented for fine-grain definition of applications and execution environments. Their research has tended to focus on installation and configuration of applications rather than addressing the management issues of hybrid and multi-clouds.

With respect to standardizing initiatives, OASIS is the most active on the cloud provisioning topic. TOSCA, which stands for *Topology and Orchestration Specification for Cloud Applications*, is a standard designed by OASIS to enable the portability of cloud applications and the related IT services (OASIS, 2013). This specification permits to describe the structure of a cloud application as a *service template*, that is in turn composed of a *topology_template* and the types needed to build such a template. The TOSCA Simple Profile (OASIS, 2017) is an isomorphic rendering of a subset of the TOSCA v1.0 XML specification in the YAML language. It provides a more accessible syntax as well as a more concise and incremental expressiveness of the TOSCA language in order to speed up the adoption of TOSCA to describe portable cloud applications.

A couple of initiatives regarding the use of policies to support the deployment of cloud application are worth to be cited. A scheduling method based on SLA metrics for deploying application in cloud (Emeakaroha et al., 2011) is discussed. This method takes into account the amount of required CPU, network bandwidth and storage. The resource with constrained SLA in IaaS, Platform as a Service (PaaS), Software as a Service (SaaS) is provided by the scheduling method. This research outlines a novel aspect of application deployment in cloud. In the paper, the resources are differentiated only by the metrics of SLA. However, in enterprise hybrid cloud there are far more parameters which need to be considered to classify a cloud resource. Similarly, a cost based datacenter resource decision model (Strebel and Stage, 2010) is proposed for application deployment in hybrid cloud. It builds optimization model to compare virtual machines' tariffs to achieve economic efficiency of application operation. A major limitation of this research is that it takes only one factor into account the cloud resource selection in hybrid cloud.

3 SYSTEM ARCHITECTURE

For the enterprise that owns or uses a large scale IT infrastructure with multi-clouds, the common practice is to divide the infrastructure into several independent infrastructure domains based on business and IT requirements. For example, a global multi-clouds infrastructure can be divided into two management domains, because it owns two data centers in two different locations (e.g., one in Frankfurt and one in Paris). Should private resources be insufficient to sustain the enterprise's business growth, publicly provided resources come to the aid (Amazon AWS[3], Microsoft Azure[4], etc.). The depicted scenario can become very problematic to manage, as it includes the presence of hybrid and geographically distributed resources. Issues can arise from both the technical point of view (different APIs to access resources provided by third parties) and the administrative point of view (different law restrictions on data enforced by countries, heterogeneous security and SLA guarantees offered by third parties).

The complexity of multi-domain infrastructures can be faced with the help of *policies*. A policy applies to a specific context and defines a set of possible actions that can be triggered according to the activation of given rules. So, for instance, a policy

[1] www.openstack.org

[2] https://wiki.openstack.org/wiki/Heat

[3] aws.amazon.com

[4] azure.microsoft.com

could define the security level that is to be granted in a given domain upon specific conditions. Policies could even span multiple, geographically distant domains (e.g., the same policy may apply in both Frankfurt and Paris domains). Figure 1 depicts the data model on which the design of the hereby proposed framework is grounded. Basically, an enterprise's cloud infrastructure is composed of multiple *Cloud pools*, whereas by pool we mean a cluster of resources residing in a specific place and managed in a cloud fashion by way of any *IaaS software*. A *Policy context* defines the logical boundaries of a set of resources. Resources taken from different pools can be part of the same context. Finally, a *Policy* can apply to one or multiple contexts, and more policies can be grouped to form a *Policy Group*.

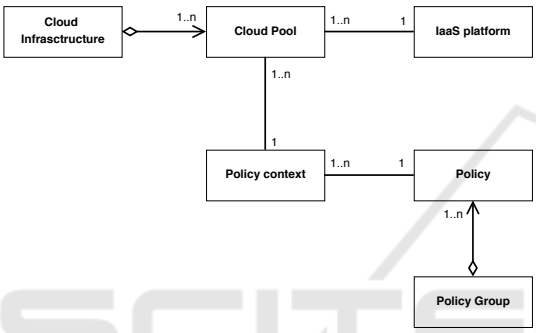

Figure 1: Data Model.

In Figure 2 the framework's **System Architecture** is depicted on top of a typical hybrid Cloud Infrastructure consisting of both private and public resources. In the figure, the Cloud infrastructure is represented by a number of silos of resources (Cloud Pools). A Pool can either be privately managed (e.g., by way of the Openstack software) or be constituted by aggregating resources rented from a public Cloud Provider (Amazon in the example). Policy contexts have also been highlighted in the figure. Some contexts refer to a single domain (PolicyContext1, PolicyContext2, PolicyContext4, PolicyContext6, PolicyContext7), others span two domains (PolicyContext3, PolicyContext5).

The System Architecture layer hosts the software components responsible for the policy-based deployment of cloud services. A deployment process is started upon a customer's submission of a *Cloud Service Request*, which is a combination of two inputs: a Cloud Service Policy and a Cloud Service Blueprint. The role of the System Architecture's components are explained below:

- **Manager.** The Manager is the component in charge of handling the customer's Cloud Service Request. Upon the reception of a Request, the Manager triggers and coordinates cloud resource allocation and cloud service deployment activities.
- **Policy Engine.** The policy engine loads the policy and evaluates the request against it, and then the engine gives back the response. The policy engine's request evaluation has two steps. Firstly, it matches each deploy condition to the policy and gets the corresponding policy contexts. Secondly, it finds the intersected domains from the result of the previous step.
- **Cloud Infrastructure Broker.** The cloud infrastructure broker plays a key role as facade of multiple Cloud Resource Pools. It has the responsibility of selecting the resources that match the target policy context and the resource requirement.
- **Cloud Orchestrator.** The Cloud Orchestrator receives in input application's deployment requirements, turns them into a pipeline of activities of deployment/configuration of the cloud application components and finally run the activities' execution. Details on the cloud orchestration process can be found in Section 4.2.

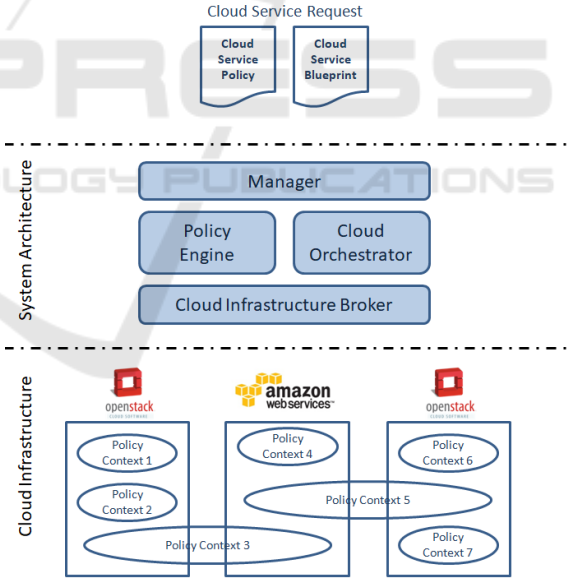

Figure 2: System Architecture.

In order to give the reader a sense of how the whole system actually works, we provide a dynamic view highlighting the components' interaction upon the arrival of a Cloud Service Request.

The sequence diagram depicted in the Figure 3 shows the actions taken by the System Architecture components in reaction to the submission of a request. The service request specifies two types of requirements: policy requirements, embedded in the *Cloud*

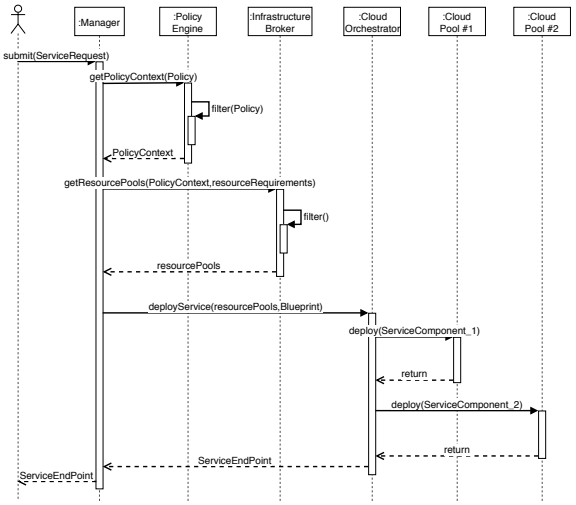

Figure 3: Sequence diagram.

Service Policy file, and application deployment requirements, contained in the *Cloud Service Blueprint* file. The Policy and the Blueprint files carries information on the cloud service's non-functional and functional requirements respectively. In Sections 4.1 and 4.2 insights on the two documents' structure can be found.

The *Manager* receives the service request and coordinates the provisioning operations. The first component to be invoked is *Policy Engine*. It is in charge of matching the policy requirements specified in the Cloud Service Policy file to the enterprise's policies that are pre-loaded form a private policy storage. The output of the task will be the *Policy Context* that best accommodates the request. In the subsequent step the *Cloud Infrastructure Broker* is engaged in the process. The Broker is the component responsible for selecting those Cloud resources belonging to the just identified Policy Context which also match the request's functional requirements (e.g., number and type of VMs, RAM size, etc.). Output of this selection will be a list of Cloud Pools where resources will be taken from. This information, along with the Cloud Service Blueprint, is handed over to the *Cloud Orchestrator* in order to run the deployment routine. In the specific case depicted in the figure, we have assumed that the application to be deployed consists of two components, which are eventually deployed in *CloudPool*#1 and *CloudPool*#2 respectively.

4 POLICY-BASED DEPLOYMENT

The cloud service deployment process is jointly carried out by the Policy Engine and the Cloud Orchestrator, which are the two core components of the System Architecture. The former identifies which of the multiple enterprise contexts are eligible for hosting the service, the latter interacts with the selected contexts and runs the service deployment procedures. The following subsections provide insights on the policy management and the service deployment respectively.

```
{
        "policyName": "NameOfThePolicy",

        "rules": {

                "rule_1": ["PolicyContext_1", "PolicyContext_2",
                                        ".........", "PolicyContext_m"],

                "rule_2": ["PolicyContext_1", "PolicyContext_2",
                                        ".........", "PolicyContext_m"],

                "........." : [""],

                "rule_n": ["PolicyContext_1", "PolicyContext_2",
                                        ".........", "PolicyContext_m"]
        }
}
```

Listing 1: Policy structure.

4.1 Policy Management

Basically, a Policy file is the smallest unit in policy management (Wei and Rodriguez, 2018). It may contain one or multiple rules. Each rule is a key-value pair. The purpose of rules is organizing and sorting Policy contexts with their features or standards. The structure of a Policy file is reproduced in Listing 1. For example, a rule can describe all policy contexts that are bound to the Frankfurt data center. When a cloud service request requires the Frankfurt premise, the policy containing that rule will match and return all policy contexts associated to that premise.

As mentioned earlier, the Policy Engine's task is to receive the policy request, load the available enterprise policies and evaluate the request against them. We designed an ad-hoc algorithm, called "match-and-select", to evaluate the request. The match-and-select is a counting sort based algorithm. It features two different steps. Firstly, it matches each deploy condition to the policy and gets the corresponding contexts. Secondly, it finds the intersected contexts from the result of last step. The pseudo code for this algorithm is shown in Algorithm 1. To evaluate the request, the algorithm iterates each condition of the request. For each condition, the algorithm finds the corresponding policy and gets the contexts which are defined in the policy. The results from all conditions are sorted by count sort. Then the algorithm iterates through the sorted set and returns an array of intersected contexts. If no intersected domains are found, null is returned.

Two or more policies can be grouped into a Policy group. Other than policies, a policy group contains

Algorithm 1: Policy engine *match-and-select* algorithm.

Input: Set of deploy conditions $C = (c_1, c_2, \ldots, c_n)$, n is the number of conditions

Output: Set of infrastructure contexts $C = (C_1, C_2, \ldots, C_m)$, m is the number of contexts

```
for c_i ∈ C do
    p = findPolicy(c_i)
    PC_i = getContexts(p, c_i)
    PC = addToSet(PC_i)
PC = countSort(PC, 1, N_contexts);
/* N_contexts = numberOfAllContexts */
D ← null
for d ← 1 to N_contexts do
    if PC_d = n then
        D = addToArray(d)
return D;
```

configuration options, which could be useful to handle complicated evaluation results. The configuration options are the select algorithm and the evaluation strategy respectively. The select algorithm is useful when more than one data center is returned from policy engine. Two options are available in this case: random and all. The random option selects one data center randomly. The all option keeps all the returned data centers, later the resources will be equally distributed to the selected data centers. The evaluation strategy is useful when the request has multiple subrequests. This algorithm has two options, strict and easy. The strict option only returns the success response when all subrequests are successful. The easy option returns success response from any successful subrequests.

4.2 Service Deployment

The Cloud Orchestrator component depicted in Figure 2 is responsible for carrying out the cloud provisioning process. In Figure 4 the orchestrator's reference scenario is depicted. The **Cloud Orchestrator** takes a TOSCA service template (in the YAML format) as input, translates it into *BPMN schemes* (OMG, 2011) and feeds such schemes to a *BPMN Engine* which eventually orchestrates the provisioning tasks. Provisioning tasks, in turn, invoke provisioning services in a SOA (Service Oriented Architecture) fashion. This enables a scenario of a market of services in which many Providers advertise their services and Customers can get the best combination of services that meet their own requirements. We envision two categories of provisioning services: *Cloud Services* and *Packet-based Services*. The first category consists of resources offered by way of any Cloud delivery model (IaaS, PaaS, SaaS), be them virtual machines, platforms or even software application instances hosted on a third-party provider's premises; by contrast, the second category includes all the downloadable software "packets" which require a pre-configured runtime environment to run (be it a database management system or a web server executable, to name a few).

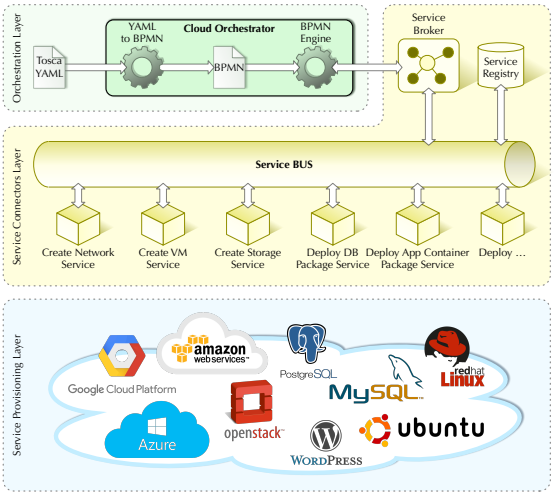

Figure 4: Cloud Orchestration scenario.

A novelty introduced by this approach is the separation between the orchestration of the provisioning tasks and the invocation of the provisioning services themselves. Since there is a multitude of providers offering a huge variety of services (each with different features and QoS), an *intermediate* Service Connectors Layer has been introduced. This layer is populated with REST services called *Service Connectors*. Connectors provide a unified interface model for the invocation of the provisioning services. The unified model allows to achieve service location transparency and loose coupling between the BPMN provisioning tasks and the actual provisioning services.

In Listing 2 we provide an example of a two-tiers application topology file expressed in TOSCA Simple Profile. The template file describes a web application stack hosted on the web server compute resource ("web_server"), and a database software hosted on the database server compute resource ("db_server"). The application topology file specifies provisioning constraints by means of the *requirements* tag. If a TOSCA node contains a requirement tag, the provisioning of that node must be delayed until the node pointed by the tag is successfully provisioned. In this case we say that the pointing node "depends" upon

```
topology_template:

  inputs:
    # omitted here for brevity

  node_templates:

    wordpress:
      type: tosca.nodes.WebApplication.WordPress
      properties:
        requirements:
        - host: apache
        - database_endpoint: wordpress_db
      interfaces:
        Standard:
          inputs:

    apache:
      type: tosca.nodes.WebServer.Apache
      properties:
      requirements:
        - host: web_server

    web_server:
      type: tosca.nodes.Compute
      capabilities:
        host:
         properties:
           num_cpus: 1
           disk_size: 50 GB
           mem_size: 8192 MB
      os:

    wordpress_db:
      type: tosca.nodes.Database.MySQL
      properties:
      requirements:
        - host: mysql

    mysql:
      type: tosca.nodes.DBMS.MySQL
      properties:
      requirements:
        - host: db_server

    db_server:
      type: tosca.nodes.Compute
      capabilities:
        host:
         properties:
           num_cpus: 2
           disk_size: 50 TB
           mem_size: 16384 MB
        os:
```

Listing 2: A simple TOSCA topology template.

the pointed node. The Cloud Orchestrator is responsible of parsing the template file, building the graph of node dependencies and translating the graph into a BPMN workflow of provisioning tasks. So in the case of the topology depicted in Listing 2, the wordpress-apache-web_server dependency will translate into a workflow in which the web_server provisioning task precedes the apache provisioning task, which in turn precedes the wordpress.

In the framework, the provisioning of any Cloud application is modeled through a variable number of two kinds of provisioning tasks: cloud service provisioning tasks and packet-based provisioning tasks. The BPMN scheme governing the cloud application provisioning is depicted in Figure 5.

The BPMN scheme is composed of a parallel multi-instance sub-process, i.e., a set of sub-processes (called "Instantiate Node") each processing a TOSCA node in a parallel fashion. Since a TOSCA node can

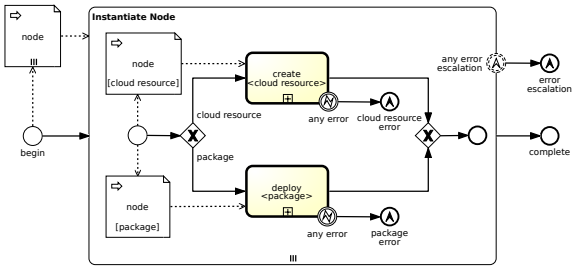

Figure 5: BPMN scheme of the cloud application provision workflow.

be either a cloud resource or a software package, a sub-process will proceed to either a "create ‹cloud resource›" or a "deploy ‹package›" sub-process.

The reader may refer to (Di Modica et al., 2017) to discover more technical details of the provisioning process. A concrete use case example of how the provisioning step was combined with the policy filtering presented in Section 4.1 is discussed in Section 5.

5 CASE STUDY

In this section the viability of the proposal is assessed through the discussion of a simple case study of two empirical deployment scenarios.

Let us assume that an enterprise owns a data center and that the Openstack software is used to virtualize and manage all internal resources. Following the business needs, the enterprise intends to deploy on the Cloud a new web application developed according to the classic *two-tiers* logic. The application is then composed of a Database for data persistence and a Business logic. The enterprise's adopted policy file is shown in Listing 3. The policy group defines 3 policies, ownership, location and security level. The select algorithm is random and the evaluate strategy is set to easy, which means that in the case of several matched policy contexts the policy engine will randomly select one and will return partially succeed result even though some requests fail.

Scenario One: Private/Public Cloud. Because of the strict data security policy, the company decides to internally host all data collected from the application. However, for a better user experience the application should be deployed close to user. Unfortunately, due to the limitation of company's internal resource and the cost of the resource, internal resource could not reach to all the users.

Therefore, in order to satisfy both constraints, a good deployment strategy would be to have the business logic installed off premise, i.e., on computing re-

```
{
  "policyGroup": [
    {
      "policyName": "ownership",
      "rules": {
        "on-premises": ["PolicyContext1", "PolicyContext2",
          "PolicyContext6", "PolicyContext7"],
        "supplier": ["PolicyContext4"]
      }
    },
    {
      "policyName": "location",
      "rules": {
        "muc": ["PolicyContext1"],
        "fra": ["PolicyContext2", "PolicyContext4"],
        "sgp": ["PolicyContext6", "PolicyContext7"]
      }
    },
    {
      "policyName": "securityLevel",
      "rules": {
        "high": ["PolicyContext3","PolicyContext5"],
        "middle": ["PolicyContext2", "PolicyContext4",
                "PolicyContext7"],
        "low": ["PolicyContext1", "PolicyContext6"]
      }
    }
  ],
  "selectAlgorithm": "random",
  "evaluateStrategy": "easy"
}
```

Listing 3: Policy file. The policy file contains 3 policies. Each policy has several rules which is mapping to policy contexts.

sources leased from a public cloud provider, while keeping the database running on internal resources (on premise). The technical burden introduced by this solution is, of course, the setup of a network overlay between the public and the private domain.

So, in this scenario we prepared the cloud service request file to require on premise cloud resources for the database and external cloud resources for the business logic.

The request file is shown in Listing 4. The request file has two subrequests, each of them has only one deploy condition and one topology group, which links to topology template and later will transfer obtained resource to the groups via the Manager. The deploy conditions require the available resource from "on-premises" and "supplier". From the policy file we can see that both the requests can be satisfied, and corresponding results will be sent to the Infrastructure Broker to further require the resource information, which later will be passed to the Cloud Orchestrator.

The Cloud Orchestrator will then receive two inputs: the resource information and the Blueprint (TOSCA topology file). The latter is reported in the Listing 5. The reader may notice that in the bottom of the Blueprint file two groups have been defined: $off\_premise$, that collects the business logic nodes, and $on_premise$, that includes the database nodes. By crossing this information with the policy's, the Cloud Orchestrator will activate the deployment of the business logic in the Policy Context #4 (Amazon Supplier), while the database will be deployed in any of

```
[
  {
    "name": "subrequest1",
    "deployConditions": {
      "ownership": "on-premises"
    },
    "resourceRequirement": {
      "topologyGroup": "on_primes_group"
    }
  },
  {
    "name": "subrequest2",
    "deployConditions": {
      "ownership": "supplier"
    },
    "resourceRequirement": {
      "topolgyGroup": "off_primes_group"
    }
  }
]
```

Listing 4: Request file for scenario one. The request file has two requests which require private cloud resource and public cloud resource.

```
topology_template:

  inputs:
    # omitted here for brevity

  node_templates:

    wordpress:
    apache:
    web_server:
    wordpress_db:
    mysql:
    db_server:
        # omitted here for brevity

    groups:
      off_premise_group:
        type: tosca.groups.Root
        members: [wordpress,apache,webserver]
      on_premise_group:
        type: tosca.groups.Root
        members: [wordpress_db,mysql,db_server]
```

Listing 5: Nodes grouping for scenario one.

the policy contexts bound to the private premises.

Scenario Two: Geo-location. To handle the increasing user demand in the German market, the company decided to scale out the application's database layer in Frankfurt data center. In this scenario we added one extra request to only require the private cloud resource which is located in the Frankfurt data center. The request file is shown as in Listing 6. The request has two deploy conditions "on-premises" and "fra". The requested resource will be assigned to "all_group" nodes in the TOSCA topology template. As expected, the policy engine gave "Policy context2" as result.

6 CONCLUSION

The paper proposes a policy based application deployment in hybrid and multi-cloud environment. The requirements come from a growing adoption of cloud in large enterprises and increasingly complex hetero-

```
[
  {
    "name": "demoSceneTwo",
    "deployConditions": {
      "ownership": "on-premises",
      "location": "fra"
    },
    "resourceRequirement": {
      "topologyGroup": "all_group"
    }
  }
]
```

Listing 6: Request file for scenario two. The request requires private cloud resource which also locates in Frankfurt.

geneous infrastructure background. In this paper, the overall system architecture is presented. The Policy Engine and the Cloud Orchestrator are discussed in details. A case study of the proposed system is conducted with a practical real world application, and a set of experiments further show the insights of the system. The results suggest that the policy based deployment can well manage and simplify the deployment in multi-cloud environment.

In future work, we will continue to investigate in the following directions. We will first extend the support of cloud broker to Azure cloud and further develop the proposed system into SaaS service. Then we will investigate to add user friendly methods for editing policies, requests and topology templates. Additionally, the current design will be integrated with a container system and extended to support the automatic management the application lifecycle.

REFERENCES

Amazon (2016). Amazon CloudFormation. https://aws.amazon.com/cloudformation/. Last accessed on 04-01-2019.

Di Modica, G. and Tomarchio, O. (2016). Matchmaking semantic security policies in heterogeneous clouds. *Future Generation Computer Systems*, 55:176 – 185.

Di Modica, G., Tomarchio, O., Calcaterra, D., and Cartelli, V. (2017). Combining TOSCA and BPMN to Enable Automated Cloud Service Provisioning. In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 187–196, Porto (Portugal).

Emeakaroha, V. C., Brandic, I., Maurer, M., and Breskovic, I. (2011). Sla-aware application deployment and resource allocation in clouds. In *Proc. IEEE 35th Annual Computer Software and Applications Conf. Workshops*, pages 298–303.

GigaSpaces (2016). Cloudify. http://getcloudify.org/. Last accessed on 04-01-2019.

IBM (2016). IBM Cloud Orchestrator. https://www.ibm.com/us-en/marketplace/deployment-automation. Last accessed on 04-01-2019.

Mell, P. and Grance, T. (2010). The nist definition of cloud computing. *Communications of the ACM*, 53(6):50.

OASIS (2013). Topology and Orchestration Specification for Cloud Applications Version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html. Last accessed on 04-01-2019.

OASIS (2017). TOSCA Simple Profile in YAML Version 1.2. http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.2. Last accessed on 04-01-2019.

OMG (2011). Business Process Model and Notation (BPMN 2.0). http://www.omg.org/spec/BPMN/2.0/. Last accessed on 04-01-2019.

Pham, L. M., Tchana, A., Donsez, D., de Palma, N., Zurczak, V., and Gibello, P. Y. (2015). Roboconf: A hybrid cloud orchestrator to deploy complex applications. In *Proc. IEEE 8th Int. Conf. Cloud Computing*, pages 365–372.

RedHat (2016). RedHat CloudForms. https://www.redhat.com/it/technologies/management/cloudforms. Last accessed on 04-01-2019.

Rightscale (2016). Rightscale Cloud Management Platform. http://www.rightscale.com/why-cloud-management-platform/benefits. Last accessed on 04-01-2019.

RightScale (2017). Rightscale 2017 state of the cloud report. Technical report, RightScale.

Strebel, J. and Stage, A. (2010). An economic decision model for business software application deployment on hybrid cloud environments. *Multikonferenz Wirtschaftsinformatik 2010*, page 47.

Wei, H. and Rodriguez, J. S. (2018). A policy based application deployment method in hybrid cloud environment. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 93–99.

Weinman, J. (2016). Hybrid cloud economics. *IEEE Cloud Computing*, 3(1):18–22.