Describing and Processing Topology and Quality of Service Parameters of Applications in the Cloud

Gabriele Pierantoni, Tamas Kiss, Gabor Terstyanszky, James DesLauriers, Gregoire Gesmier, Hai-Van Dang

Centre for Parallel Computing, University of Westminster,
115 New Cavemdish Street, London W1W 6UW, United Kingdom

Abstract

Typical cloud applications require high-level policy driven orchestration to achieve efficient resource utilisation and robust security to support different types of users and user scenarios. However, the efficient and secure utilisation of cloud resources to run applications is not trivial. Although there have been several efforts to support the coordinated deployment, and to a smaller extent the run-time orchestration of applications in the Cloud, no comprehensive solution has emerged until now that successfully leverages applications in an efficient, secure and seamless way. One of the major challenges is how to specify and manage Quality of Service (QoS) properties governing cloud applications. The solution to address these challenges could be a generic and pluggable framework that supports the optimal and secure deployment and run-time orchestration of applications in the Cloud. A specific aspect of such a cloud orchestration framework is the need to describe complex applications incorporating several services. These application descriptions must specify both the structure of the application and its QoS parameters, such as desired performance, economic viability and security. This paper proposes a cloud technology agnostic approach to application descriptions based on existing standards and describes how these application descriptions can be processed to manage applications in the Cloud.

Keywords: Application Description Template; Application-level Cloud Orchestration; Quality of Service, Automated Scalability, TOSCA

1. Introduction and Problem Statement

Cloud computing has successfully addressed issues how to run applications on complex distributed computing infrastructures. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [1] solutions are widely used in academia and business to manage applications in the Cloud. However, there are application and infrastructure specific challenges, such as deployment, scalability and security requirements that must be addressed. At the one hand, on-demand access to cloud resources and services in a flexible and elastic way could result in significant cost savings due to more efficient and convenient resource utilisation. Additionally, it can also replace large investment costs and decrease long-term operational costs. On the other hand, the efficient and dynamic utilisation of cloud resources and services is not trivial. The take up of cloud computing is still relatively low due to limited application-level flexibility and shortages in cloud specific skills.

Porting and running applications in the Cloud has also been slowed down by the intrinsic complexity required to describe the services that compose the applications considering their deployment, migration, scalability and security requirements. As a result, the move to the Cloud has been somehow slower and more cautious in some application areas due to both application- and infrastructure-level complexity. For example public sector organisations [2] and Small- and Medium-sized Enterprises (SME) [3] are increasingly considering using the Cloud in their everyday activities but they still face difficulties of both economic and technical nature. Applications in these areas might run simulations, collect and process public service and social media data, etc. They have to process large volume of data and might have restrictions on execution, such as costs, deadlines, security, etc. To meet these requirements, efficient resource utilisation including resource scalability, such as CPU, disk and memory scalability has to be achieved. When faced with such complexity, application developers may decide not to take up or to abandon the Cloud if they are not properly supported. Although there have been several efforts to support the deployment, and to a smaller extent run-time orchestration of cloud applications, no comprehensive solution has emerged that could be applied in both academia and business to address the above challenges.

To enable the execution of a large variety of applications in the Cloud in a cost effective, flexible, seamless and secure way, applications must be deployed, launched, executed and removed through a framework that hides cloud specific details from users. These phases need information about the applications, such as their architecture, resources and services they need, and Quality of Service (QoS) parameters they must meet. Application descriptions should define the application architecture, specify where to deploy and run applications, and formulate requirements towards their cost-effective execution and desired security policies. Application description is one of the major challenges in cloud computing considering complexity of applications and the Cloud itself. This description should help application developers to define applications in a simple, flexible, reusable and seamless way. It allows them specifying services and QoS properties of applications to enable their deployment and execution in the Cloud.

Although there are several approaches that describe application architectures or even specify some policies, such descriptions are typically limited in their reach and specific to particular cloud infrastructures. Existing application description approaches (see in Section 3) allow specification of application architecture and definition of some policies that regulate deployment and execution of applications, but these approaches are not as efficient and flexible as required. Moreover, there is also a lack of a cloud agnostic framework that processes and enforces such descriptions in various cloud infrastructures. To support application developers, we elaborated a technology agnostic application description solution, called Application Description Template (ADT) that is presented in this paper together with a prototype framework that processes and acts upon such descriptions.

The paper is structured as follows. **Section 2** introduces an abstract view of application description. It identifies three challenges we addressed and design guidelines that the challenges are mapped to. **Section 3** describes the state of the art in application description approaches used in the Cloud and explains why we selected TOSCA (Topology and Orchestration Specification for Cloud Applications, an OASIS standard) [4] to implement ADT. **Section 4** outlines the design of the ADT and the extended TOSCA policy architecture and explains how this design realises our design guidelines. **Section 5** demonstrates the feasibility of the concept by presenting how a commercial application, called Magician, can be described with the ADT and managed in the Cloud with the MiCADO (Microservices-based Cloud Application-level Dynamic Orchestrator) [11] generic cloud orchestration framework. **Section 6** contains conclusions and future work.

2. Abstract View of Application Description

To deploy and execute applications in the Cloud, first we have to describe them in a way that can be understood by all components involved. Such application description acts as conduits of information which connects various stakeholders and components. Furthermore, to foster cloud adoption we must strive to lower the learning barrier required to write the application descriptions and reduce as much as possible technology-specific constraints.

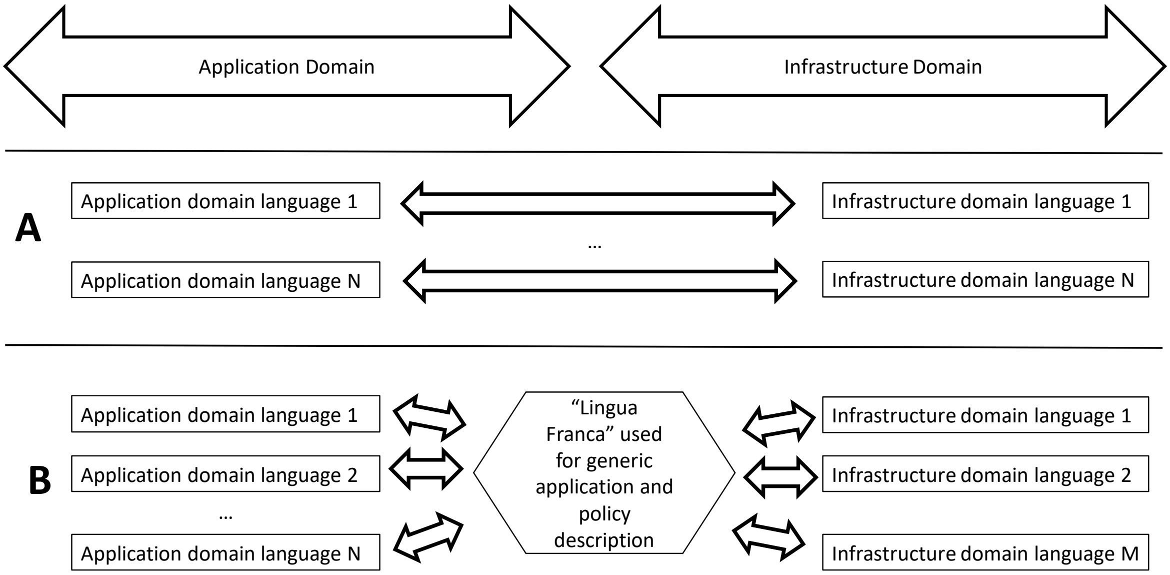

Application Domain
Infrastructure Domain
A
Application domain language 1
Infrastructure domain language 1
...
Application domain language N
Infrastructure domain language N
B
Application domain language 1
Application domain language 2
...
Application domain language N
"Lingua Franca" used for generic application and policy description
Infrastructure domain language 1
Infrastructure domain language 2
Infrastructure domain language M

Figure 1 - Abstract View of the Context of the Application Description Template

We defined two main domains of this context. On the left of Figure 1 lies the technology agnostic Application Domain in which various application stakeholders engage in creating applications, describing them in a way which supports their deployment on cloud infrastructures, and finally, defining and selecting appropriate policies that govern their lifecycle. The right-hand side of Figure 1, Infrastructure Domain, contains elements that are specific to the deployment and execution services used by cloud providers (e.g. monitoring services, orchestration tools, security frameworks, etc.). These two domains might contain different solutions that offer similar or overlapping functionalities, and this can raise interoperability issues among different technologies.

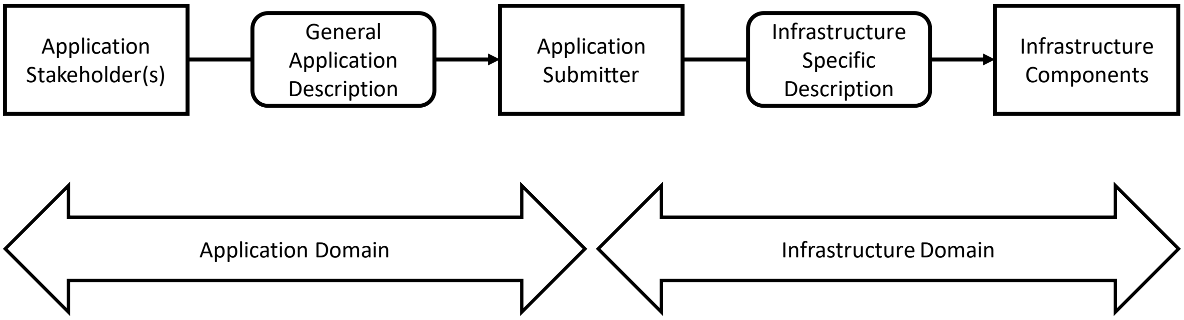

Application Stakeholder(s)
General Application Description
Application Submitter
Infrastructure Specific Description
Infrastructure Components
Application Domain
Infrastructure Domain

Figure 2 - The Application Submitter as a decoupling element.

There are two main approaches to solve this problem and connect these two domains. In the first case (Approach A in Figure 1) a single description language specific to an infrastructure domain is propagated throughout all the elements and must be used to describe application and policies. This approach has the advantage to be simple and naturally arises when one single technology becomes dominant imposing its own language. On the other hand, it constrains the freedom of choice to the solution decided by the adopted technology. In comparison, Approach B allows for both domains to use different languages and employs a "lingua franca" which acts as a decoupling element in the middle. In our work we have followed Approach B whereby the two domains are connected by a conceptual component, named Application Submitter, as depicted in the middle of Figure 2. The Application Submitter is a service that is be capable of converting the technology agnostic General Application Description (created and provided by the Application Stakeholder(s), typically the users or owners of the system) to an Infrastructure Specific Description used by different Infrastructure Components.

3. Related Works: Application Description in Cloud Computing

3.1 Overview of application description approaches

There are currently three major application description approaches to target challenges described previously: cloud platform (e.g. Amazon, Microsoft Azure, Oracle, or OpenStack) dependent, cloud orchestration tool (e.g. Chef, Ansible or Juju) dependent, and platform and tool independent approaches (e.g. Camp and TOSCA).

Cloud platform dependent approaches. Most leading cloud providers offer ways of describing applications and their properties. **Amazon** uses Amazon Machine Image (AMI) Template to describe all information required to launch an Amazon EC2 instance, and AWS Cloud Formation Template [5] to support development, deployment and running of applications on the Amazon cloud. **Microsoft** Azure Resource Manager Template [6] combines compute, storage and network resources into a single entity to manage applications in the Cloud. **ORACLE** uses Oracle VM Template [7] to enable quick configuration and provisioning of multi-tier application topologies onto virtualised and cloud environments by capturing the configuration and packaging of software components as self-contained building blocks called appliances that can be easily connected to form application blueprints, called as assemblies. **OpenStack** Heat Orchestration Template (HOT) [10] provides a template-based orchestration for describing a cloud application by executing OpenStack API calls. The template allows creating most OpenStack resource types as well as more advanced functionalities, such as high availability instances, auto-scaling and nested stacks instances.

Cloud orchestration tool dependent approaches. Cloud orchestration tools typically offer higher level automation of application deployment compared to the native solutions of cloud providers. Chef and Juju are both open-source cloud orchestration tools. **Chef** [8] uses cookbooks and recipes to support integration with cloud-based platforms. Cookbooks and recipes describe system configuration and explicitly specify how to deploy and connect cloud application components. **Ansible** [40] takes an agentless approach and deploys and configures services and systems on cloud-based hosts from a remote server. Playbooks declare the desired state of a system, and the Ansible server executes commands via SSH to realise that state on a host. **Juju** [9] uses charms to enable deploying, managing, and scaling services on a wide variety of clouds. Charms encapsulate application configuration, define how services must be deployed, how they connect to other services, and how they can be scaled. They also define how services can be integrated, and how services react to events in the distributed environment.

Platform and tool independent approaches. These approaches provide a high-level description of applications that is not coupled with any specific cloud platform/middleware and tool. Topology and Orchestration Specification for Cloud Applications (**TOSCA**) [4] is an open source language specification that enables the description of portable cloud applications and the automation of their deployment and management. It allows the description of topologies, including nodes with their relationships and their policies. Cloud Application Management Platform (**CAMP**) [12] is a simple API specification to standardise the API of PaaS systems. The CAMP API has been designed for lifecycle management of applications. This management supports performing, uploading, configuring, customising, deploying/un-deploying, starting/stopping, snapshotting, suspending/restarting and deleting operations on an application/service, as well as monitoring the operation of the application.

3.2 Comparison of application description approaches and justification of the selected approach

All three investigated approaches properly describe the application architecture (or topology) specifying services they are composed of, how these services are connected, and artefacts and resources needed to run applications. Although cloud platform dependent approaches work well in the given cloud environment, these do not fulfil our need for a "Lingua Franca" due to their specific dependencies on a particular cloud technology. Using these approaches would mean adopting Approach A and developing different application descriptions based on the targeted cloud platform. Although some of the cloud orchestration tools (e.g. Chef and Juju) support multiple cloud technologies providing certain independence from the cloud middleware, application developers are still restricted to use the given cloud orchestrator tool only. Therefore, such solutions still do not fulfil the requirements of a "Lingua Franca". Further, these approaches are not based on standards and the majority of these approaches do not provide flexible QoS properties management.

On the other end, TOSCA, as an application description solution, offers all features that cloud platform and orchestration dependent approaches do plus provides some additional ones. First, it defines the application architecture describing it as a combination of services specifying their topology and relationships. It also supports publishing, sharing and storing application descriptions. Second, TOSCA specifies how to manage applications defining implementation characteristics and constraints such as the packaging of installation artefacts and large variety of installation methodologies that vary from simple scripts to complex workflows. TOSCA does explicitly provide container or runtime support, for example it can specify how to run applications in Docker containers and virtual machines. Third, TOSCA allows specification of applications' QoS properties, such as deployment, scalability, security, etc. This approach is flexible and generic enough to allow for the development of a comprehensive policy structure for the definition of various aspects and various stages of the applications' lifecycle. Fourth, TOSCA is an open standard application description language supported by a growing number of communities and by OASIS as standardisation body. As a result, all major cloud orchestration tools and several cloud platforms either created plug-ins to process TOSCA-based application descriptions or developed translators to convert TOSCA descriptions into their native descriptions. For example, IBM developed a TOSCA plug-in [13] to process TOSCA application descriptions in Chef to be used in IBM Smart Cloud Orchestrator. OpenStack Heat leverages TOSCA as a standard based approach for modelling cloud stacks and applications using TOSCA Parser and Heat Translator [14]. Juju was extended to parse TOSCA based application descriptions transforming Juju topology model components into TOSCA compliant topology model components [15]. Finally, TOSCA is being actively used in both academic and non-academic communities. Therefore, there is large variety of implementations that offers a vast experience from which application developers can greatly benefit. Among these, the most promising are OpenTOSCA [16] and TOSCAMart [17]. Considering the features of TOSCA listed above, we selected it as the basis of our Application Description Template. However, TOSCA also has some limitations as it is highlighted in the related literature.

Based on the overview of related works we identified three challenges designing and implementing ADT:

- **Challenge 1:** to describe and manage containerised/virtualised applications in the Cloud,
- **Challenge 2**: to define extensible and flexible policies for the management of a wide range of QoS properties and to provide parametrised support for these policies, and
- **Challenge 3**: to support the deployment and management of the applications in a platform agnostic way.

There were several efforts using TOSCA to address challenge 1 (description and management of containerised/virtualised applications in the Cloud) and challenge 3 (deploying and managing applications in a platform agnostic way). **Cloudify** [18] is an orchestration framework that has been extended with plugins to provide support for different cloud service providers, container platforms, as well as a variety of automation tools. It has its own Domain Specific Language (DSL) that uses TOSCA Simple Profile in YAML v1 as a base specification. DSL has strict typing, for example, there is one type defined for creating a non-orchestrated Docker container, another type for a Docker container orchestrated by Docker Swarm, and a third type for a Docker container orchestrated by Kubernetes. **ARIA** [19], built on the Cloudify code base, keeps strict adherence to the normative DSL. It offers a set of TOSCA-based tools to support the orchestration of TOSCA normative templates. **Puccini** [20] extended ARIA with a frontend that translates an extended

TOSCA v1.1/v1.2 template into a middle language called Clout, then again into an orchestrator specific language before being fed to that orchestrator. **Alien4Cloud** [21] is an application management platform, which leverages the portability of TOSCA to encourage uptake of the cloud by enterprise organisations. It offers a custom DSL with strict, but not total adherence to TOSCA Simple Profile in YAML v1.0. Plugins and a graphical interface offer support for orchestrating and designing these TOSCA templates using various tools, including Cloudify, Mesos, Kubernetes and Puccini.

There were further research efforts to address challenge 2 (extensible and flexible policies to manage QoS properties). The TOSCA specification defines some abstract non-normative policy types (access control, placement, etc.), and offers some design guidelines (such as the declarative approach), but it does not offer a detailed description on how to specify such policies. **Breitenbücher** et al. [22] elaborated an approach to assign policies to node templates and extended the TOSCA access policy with public access, no public access, secure password and only modelled port sub-policy. They also introduced a policy-aware deployment approach that generates an imperative executable Policy-Aware Provisioning Plan. It translates the topology template into an executable provisioning plan to enforce provisioning policies using the Policy-Aware Provisioning Plan Generator. They developed the OpenTOSCA platform to create, process and execute TOSCA specifications using Winery [23] and Vinothek [24]. **Waizenegger** et al. [25] proposed a taxonomy to describe policies. It contains four entities: stage, layer, effect and property. Stage defines the lifecycle stage in which the policy must be applied. Layer specifies the topology layer where the policy needs to be applied (similarly to TOSCA Targets). Effect defines how the policy effects the application. Property specifies parameters of the application. They combine the topology template and policies. They also defined two new policies: database encryption policy (sub-policy of the access control policy) and region policy (sub-policy of the placement policy). **Kepes** et al. [26] further extended policy taxonomies defining the policy signature and gave detailed overview of their policy taxonomy. They also introduced further sub-policies, such as response time and SQL injection firewall sub-policy. The authors elaborated a policy framework that transforms the abstract TOSCA entities into specific ones considering their functional requirements and policies. Their Plan Engine deploys and runs the application cooperating with the Runtime Monitor and the Policy Enforcing Manager.

Our research, that was first outlined and initiated in [39], extends the above described related work on TOSCA regarding all three identified challenges. As a result, the presented ADT enables the description of applications at two different levels (virtual machines and containers), supports the definition of an extendable set of policies to manage QoS requirements, and enables the definition of a generic framework to process and act upon such descriptions for application-level cloud orchestration. Detailed analysis of these contributions is provided in Section 4.

4. Extending TOSCA to Support Application-level Orchestration in the Cloud

This section defines the Application Description Template (ADT), describes its structure and its elements, outlines the extended TOSCA policy hierarchy and explains the MiCADO reference architecture used to process and execute ADTs.

To design the ADT, we have mapped the three challenges listed in Section 3 into six Design Guidelines. To target **Challenge 1** (description of containerised/virtualised applications) we defined:

- ***DG1 - Topology-based Description***: We assume that the application architecture is described as topologies which represent the application services, their relations and how they are deployed into the infrastructure.
- ***DG2 - Two-Level Topologies***. We have restricted the deployment and execution of the applications into either containers or virtual machines, where containers could be further embedded in virtual machines. Such assumption does not dictate that each application needs to be deployed in containers (as some applications may be deployed directly into virtual machines) but sets a limit on the number of layers within the topologies.

To address **Challenge 2** (extensible and flexible policies) and **Challenge 3** (technology agnosticism) we defined

- ***DG3 - Policy-based Behaviour***: The behaviour of applications can be described by policies which govern the various aspects of the application lifecycle.
- ***DG4 - Extensible Description***. At the time of design not all requirements may have been known. Therefore, application descriptions should be extensible to cope with additional requirements.
- ***DG5 - Infrastructure and Technology Agnosticism***. The implementation of cloud infrastructure services must not affect the application description in any way.

To help application developers in describing and managing applications we specified one more design guideline:

- ***DG6 – Stakeholder vs Application Description.*** There are several stakeholders in the Cloud, such as cloud service developers, application developers, users, etc. They may need either the whole or a sub-set of the application description based on their role. The application description should provide information about the application considering stakeholders' role.

Considering the limitations of the TOSCA based solutions, discussed in Section 3, we elaborated three major contributions addressing challenge 1, 2 and 3:

- **Contribution 1:** developing the concept of the Application Description Template to describe applications deployed and executed in two levels, i.e. in containers and/or virtual machines (challenge 1),
- **Contribution 2**: introducing an extendable set of TOSCA policies to manage deployment, performance, scalability and security requirements of applications (challenge 2), and
- **Contribution 3**: elaborating a generic framework that can automatically process Application Description Templates to deploy and manage applications in the Cloud in a platform agnostic way (challenge 3).

4.1 Entities of the Application Description Template

The ADT should manage three major structural entities: container images, virtual machines images and their policies, depicted in Figure 3. These entities are derived from the TOSCA Node element. They allow ADTs to satisfy three of the Design Guidelines: DG1 (Topology-based Description), DG2 (Two-Level Topologies), and DG3 (Policy-based Behaviour). In Figure 3 there are TOSCA nodes representing container images (Cont. 1, Cont. 2 and Cont. M) and a correlated set of nodes representing virtual machine images (VM 1 and VM N). Container images are connected by TOSCA Relationships (continuous arrow) that define their mutual dependencies. Container images can be assigned to virtual machines using TOSCA Relationships (dotted arrow). Virtual machines can host one or more container images (e.g. VM 1 hosts two container images). Finally, policies (Policy 1 to Policy L) can target different nodes (containers or virtual machines). While relations between container images and container images and virtual machines are directly implemented with the TOSCA Relationship type, the connection between policies and nodes is implemented indirectly by defining one or more target nodes within the policies (dashed arrow).

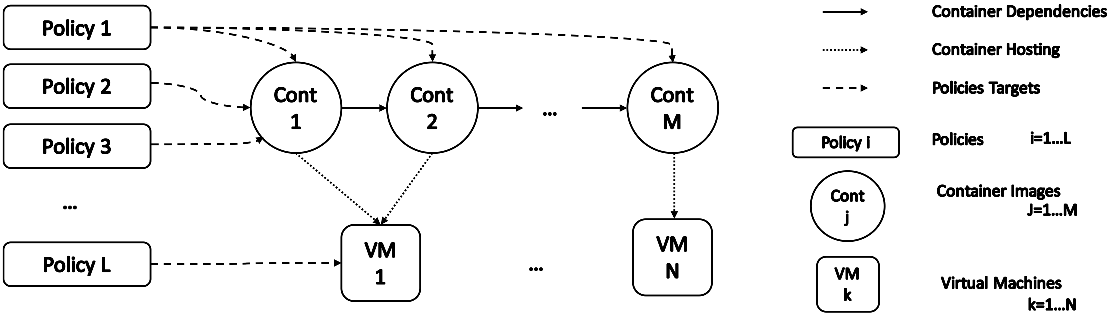


Figure 3 – Structural Entities of the Application Description Template

4.2 Structure of the Application Description Template

TOSCA describes applications in Service Templates. These templates incorporate the Topology Template and the Management Plan. The Topology Template defines the structure of the application using `NodeTemplates` to specify nodes, and `RelationTemplates` to define how `NodeTemplates` are connected. `NodeTemplates`, derived from `NodeTypes`, define attributes, capabilities, interfaces, properties and requirements of applications' nodes. TOSCA also specifies abstract `PolicyTypes` and `PolicyTemplates` that can be used to define certain aspects of the lifetime behaviour of the application. The Management Plan describes how to deploy and run the application in the Cloud.

We have defined the Application Description Template (ADT), presented in Figure 4, based on many (although not all at the moment) of the features offered by TOSCA. We have used hierarchies of `NodeTypes` to define a simplified application topology spanning Containers and Virtual Machines, and TOSCA Policy constructs to define extensible and composable policies. An ADT defines the container and virtual machine levels in the Topology Template. At the bottom level are the virtual machines that host one or more containers. ADT specifies the number of containers and virtual machines and how containers are allocated to virtual machines. This template enables assigning policies to applications, containers and virtual machines to govern how these entities are deployed and executed in the Cloud. It allows the definition of complex topologies and a rich and extendable set of policies that specify properties of the applications, for example deployment, security, scalability, etc. The ADT-based descriptions can be processed by different deployment and run-time orchestrators. ADT minimises the application developers' efforts required to specify applications. This can be achieved by decomposing the application's topology and policies into components that can be reused by application developers.

The structure of the ADT is elaborated considering design guidelines DG4 (Extensible Description) and DG6 (Stakeholders vs Application Description). Each ADT contains four sections as illustrated on Figure 4. The **Input/Output Section** consists of the input and output variables of the application. The input sub-section enables application developers, who have written the ADT, to create a list of parameters of those values that either they or the applications' users can define before submitting the application without any further knowledge of TOSCA or the ADT. This capability addresses DG6 (Stakeholders vs Application Description) distinguishing application developers, who have a deeper understanding of ADT, and application users, who only want to override a few selected variables. The output sub-section contains variables which are set during execution and should be returned to the user, for example the public IP address of a virtual machine provisioned during application deployment. These variables can refer to structural entities included in the Containers, Policies and Virtual Machines sections. The **Container** and **Virtual Machine Sections** describe containers, virtual machines and their relations. The containers are connected through TOSCA *ConnectsTo* Relationships, while containers are linked to a virtual machine on which they should be deployed using the TOSCA *HostedOn* Relationship. The **Policies Section** defines the QoS properties as Policy Elements which, in addition to specific information, also define a set of target nodes and the lifecycle phase to which these apply.

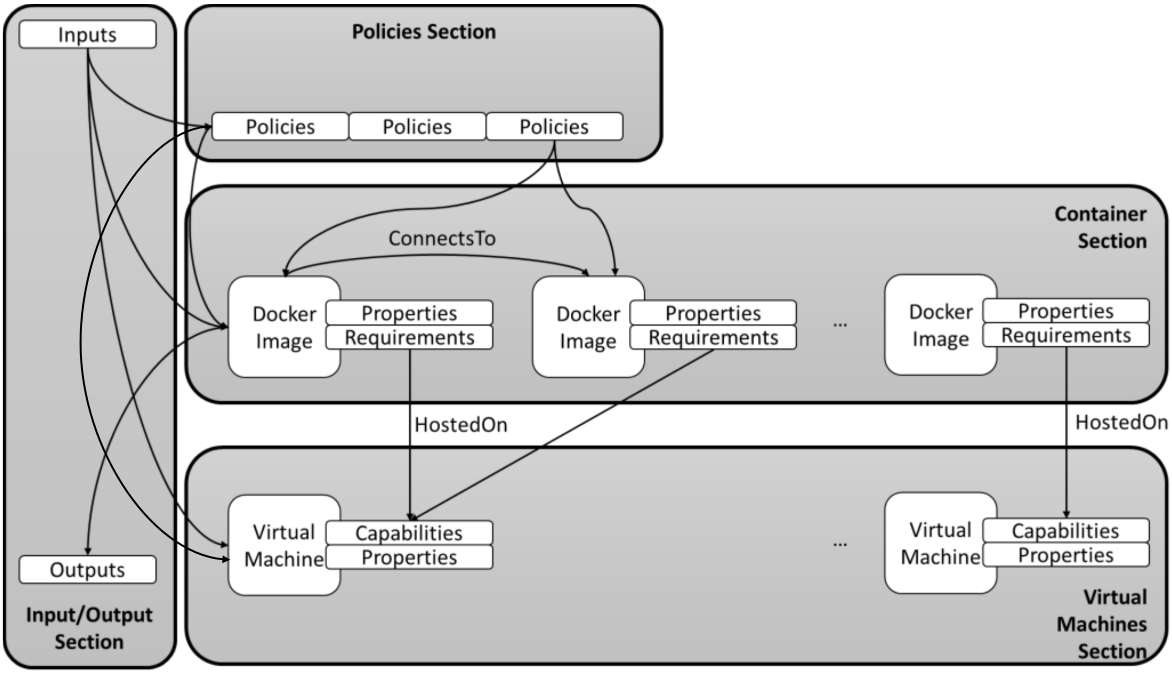


Figure 4 – Structure of the Application Description Template (ADT)

4.3 Extended Policy Hierarchy

In order to comply with DG4 (Extensible Description) we use TOSCA types arranged in hierarchies to define the various structural entities of the ADT to allow extension of elements to match with a modification in one of the elements of the ADT. As a result, application developers can define a new sub-type in the hierarchy whilst the topology and overall structure of the ADT remains unchanged. This approach is particularly relevant for the definition of the extended policy hierarchy which we have designed considering TOSCA recommendations. First, the extended policy hierarchy follows the Declarative Model, e.g. it describes the parameters that govern the policy, but it does not specify how to implement the policy. Such Declarative Model supports developing various different application level orchestrators that act upon the defined policies, i.e. the policy definition does not define or restrict the implementation of the orchestrator. Second, we support the aggregation of policies in two different ways. First, policies can target one, more or all nodes, i.e. it is possible to define one policy for the entire application and a second one for a sub-set of nodes or for a single node. Second, policies cover distinct aspects of QoS, for example scalability, security, etc., and can be composed for each node. Such composition could lead to conflicts among the policies. As an example, a budget-constraining policy applied to the entire application may be in conflict with a deadline policy applied to either the entire application or one of its components that requires the usage of expensive resources. Another example could be that of a privacy constraint that requires the placement of a database in a certain geographical area with a policy that defines an incompatible budget limit. It must be emphasised that we do not address the conflicts of policies, but we added a priority field to the policy template which expresses conflict resolution criteria that can be used by the relevant element of the ADT.

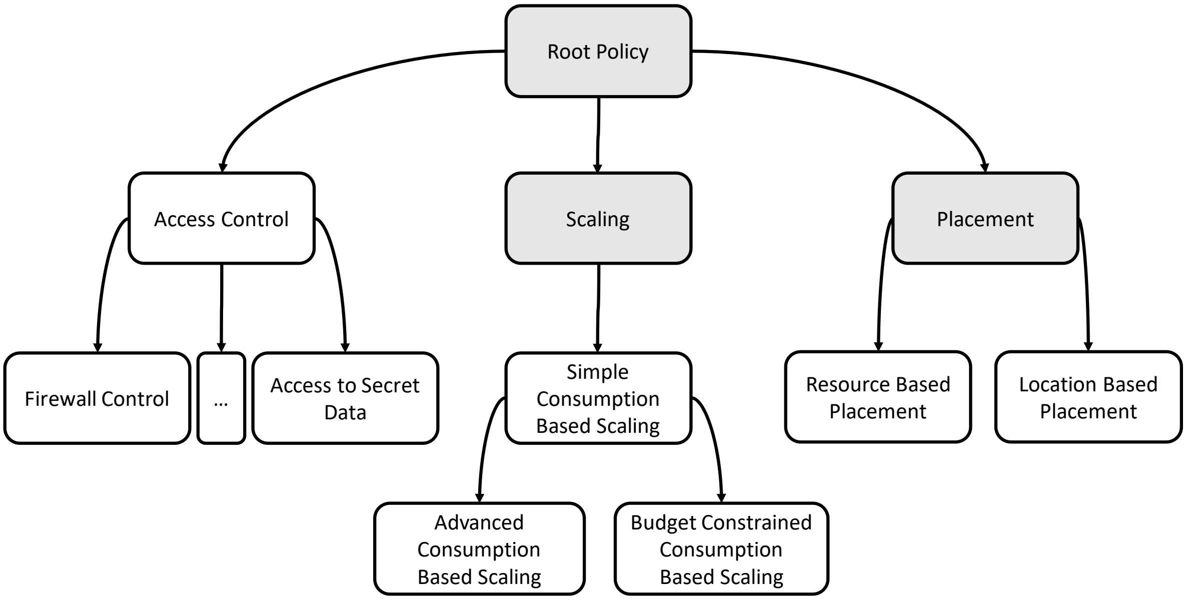
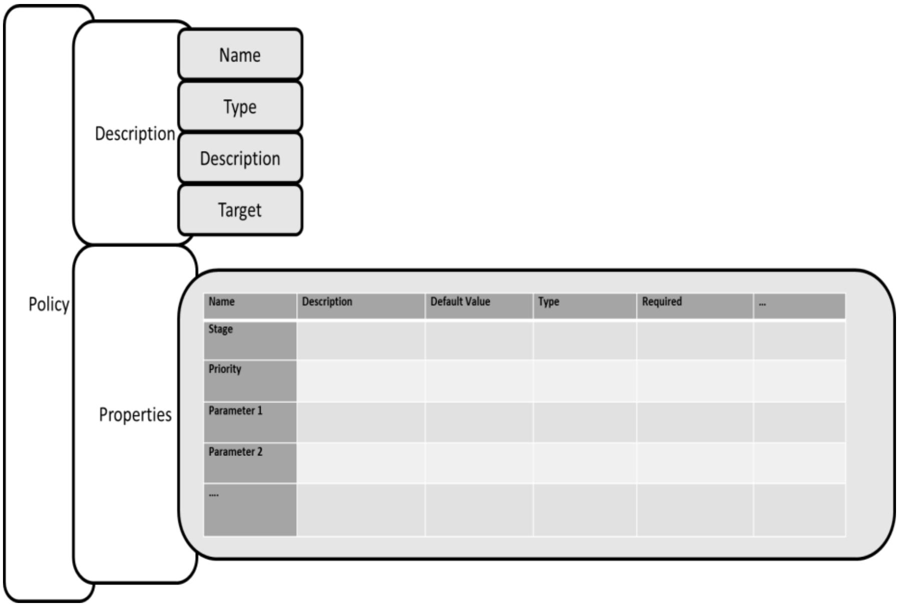


Figure 5 - ADT Extended Policy Hierarchy

We extended the Scaling and the Placement sub-policies of the TOSCA PolicyType and specified a new sub-policy called Access Control sub-policy to support deployment and execution of applications in the Cloud. The original TOSCA policies are highlighted in grey and new ones are presented in white in Fig. 5. The Placement and Scaling sub-policies contain several sub-sub-policies to support placement and scaling of different types of applications, for example Advanced Consumption Based Scaling and Budget Constrained Consumption Based Scaling. The Access Control branch encompasses security-related policies to describe functionalities such as Firewall Control and Secret Data Management which can be handled by security-specific services of the targeted orchestrator (e.g. the MiCADO framework). By extending each branch, we have created a multi-layer hierarchy of TOSCA. Each sub-policy contains the information summarised in Figure 6.

The first part, the Description Section, comprises of meta-data which defines the name, type and description of the policy, as well as a target (defined as a set of nodes in the topology) to which the policy should apply. The second part, the Properties Section, contains data that is either common to all policy types or specific to a particular policy type. Common Properties are Stage that defines at which stage of the lifecycle of the element the policy is applied, and Priority that is an arbitrary integer ranging from 0 to 100 used to define the priority with which the policy will be implemented. Specific Properties vary depending on the nature of the policy itself. For example, a scalability policy based on CPU consumption defines various parameters that specify scalability thresholds, while a deployment policy defines minimum number of CPUs and minimum memory size for deployment. To allow for a uniform representation and to support the automatic parsing of the policy parameters, the specific properties are arranged in a table whereby each property is defined with name, value, range, default value and other meta-data fields.

This combination of the hierarchical structure of types and sub-types, combined with the standard tabular representation of the parameters, support the extension of different policies with different levels of sophistication. As an example, the Consumption Based Budget Constrained policy (see Figure 5) extends the data set that defines Simple Consumption Based scalability policy. The possibility to define sub-policies similarly to the creation of sub-classes in object-oriented design also allows definition of the level of details which are exposed to ADT developers thus improving separation of concerns between application developers with different interests and competences in the policy details. For example, the Advanced Consumption Based policy defines additional parameters governing the policy which are not defined in the Simple Consumption Based scalability policy (see Figure 5).



Name	Description	Default Value	Type	Required	...
Stage					
Priority					
Parameter 1					
Parameter 2					
....					

Figure 6 - Structure of the ADT Policy Template

4.4 Reference Architecture to Manage Application Description Templates

Although DG5 (Technology Agnosticism) predicates to keep the dependencies on the technologies of the technology-specific domain (See Figure 1) to a minimum, we must define a minimal set of components and functionalities which we assume will be the recipients of the information contained in the ADT. To such aim, we have defined a generic reference architecture that is presented in Figure 7 and that we call MiCADO (Microservices-based Cloud Application-level Dynamic Orchestrator) Reference Architecture. Detailed description of MiCADO can be found in [11]. In this paper we only identify and describe the high-level building blocks of MiCADO that are required to process ADTs.

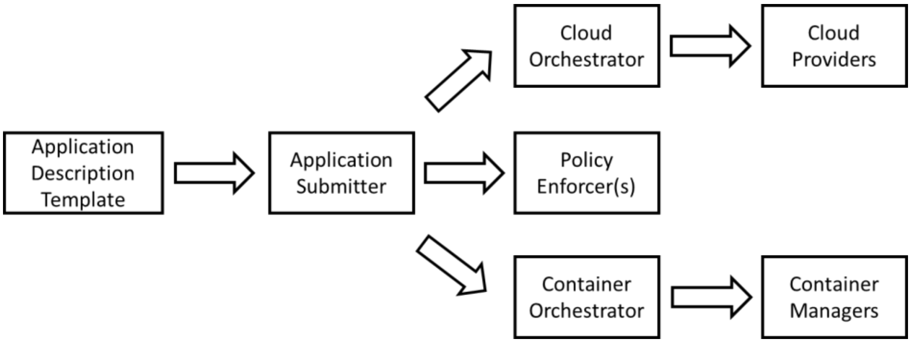


Figure 7 - MiCADO Reference Architecture

In the MiCADO Reference Architecture, ADTs are submitted to the Application Submitter which parses the description and creates three datasets: Virtual Machine (VM), container and policy related datasets. The Cloud Orchestrator creates and runs VMs in the Cloud using the VM data set. This orchestrator can either be cloud specific that is tightly coupled to particular cloud middleware (e.g. Amazon or OpenStack), or it can also be more generic that supports the deployment of VMs in multiple heterogeneous clouds (e.g. Occopus [27]). The Container Orchestrator is working with a container manager (e.g. Docker Swarm or Kubernetes) to deploy and run containers in previously created VMs using the container data set. Finally, one or more Policy Enforcers receive the policy dataset and enforce various types of policies, for example scaling policies, security policies, etc. Due to the very different nature and behaviour of these enforcers, the reference architecture allows multiple independent Policy Enforcer components. The presence of the Cloud Orchestrator and the Container Orchestrator supports the two levels of DG2 (Two-Level Topologies), whilst one or more Policy Enforcer(s) allow to follow DG4 (Policy-based Behaviour).

5. Deploying and Orchestrating Applications Using ADT – Case Study

Although ADTs have been designed without the constraints of specific implementation technology, they have been used and tested for the description and execution of applications in the COLA (Cloud Orchestration at the Level of Application) project [28]. COLA is elaborating a generic pluggable framework called MiCADO (Microservices-based Cloud Application-level Dynamic Orchestrator) [4], to support optimal and secure deployment and run-time orchestration of applications in the Cloud, following the idea of the generic reference architecture presented in Figure 7 . MiCADO is a generic framework whose services are not restricted to particular technologies and can be implemented using different existing technologies. This framework provides the missing link between existing non-cloud aware applications and the dynamic capabilities of IaaS clouds by allowing connection to multiple technology implementations on demand. For example MiCADO can be connected to multiple cloud middleware (e.g. EC2 [29], CloudSigma [30], OpenStack [31], OpenNebula [32], etc.) and generic cloud access layers (e.g. CloudBroker Platform [33]) via well-defined interfaces to avoid dependence on one particular cloud technology. The current implementation of the framework is based on existing container management technologies (e.g. Docker Swarm [34]), cloud management and orchestration solutions (e.g. Occopus [27]), and monitoring tools (Prometheus [35]). For detailed architectural description of MiCADO please refer to [11].

5.1 Magician –Data Mining Application

In order to demonstrate the feasibility of the ADT concept, particularly how applications can be described, deployed and executed in a secure and scalable way, a social media data analytics application called Magician has been utilised as an example.

The Aragon Regional Government in Spain decided to develop new communication channels with citizens to collect their feedback about the government’s services in order to further improve them. The authorities also want to provide information to companies in the region to improve existing businesses and develop new ones. The Regional Government utilises Magician, developed by Inycom [36], a Spanish ICT company. Magician offers social media data mining, competitor analysis and brand management needed for the Aragon Regional Government. It collects information from interactions with citizens and companies, and from Twitter tweets. Magician runs crawlers every two hours to collect Twitter tweets and data from authorities’ websites. The crawlers produce at least 1 TB data every year.

The high-level architecture of Magician is presented in Figure 8. There are two major Magician services, highlighted in the figure in red boxes. The Semantic Processing service receives data from the social media crawlers and from local/regional authorities’ websites. It semantically processes and stores data in the Feed DB. The Classification service runs periodically to assort the semantically processed information based on categories defined by particular users. The Semantic Processing service can be deployed into multiple containers while the Classification service should be hosted in a single container to avoid database read/write inconsistency issues. Magician has to process and classify the collected data in less than two hours. The bottleneck is the Semantic Processing service because of the large data volume it has to handle. To process data and meet the time constraint the processing service should be scaled up and down automatically based on its load. Otherwise the service crashes when the load exceeds a certain level.

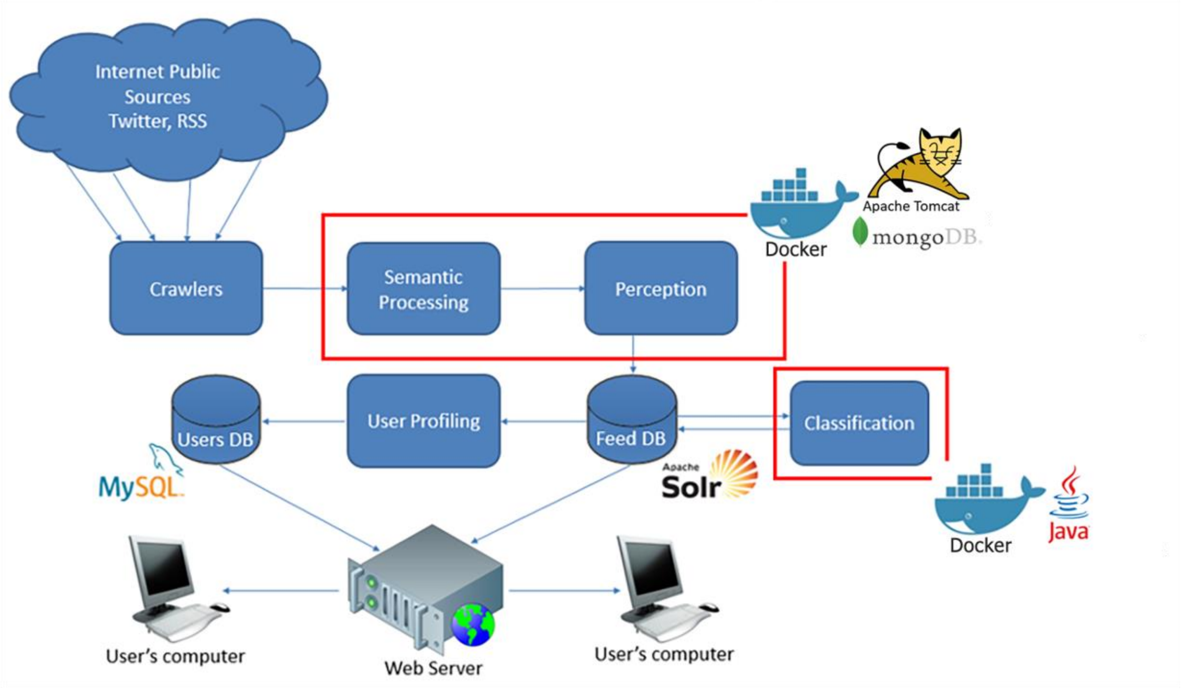
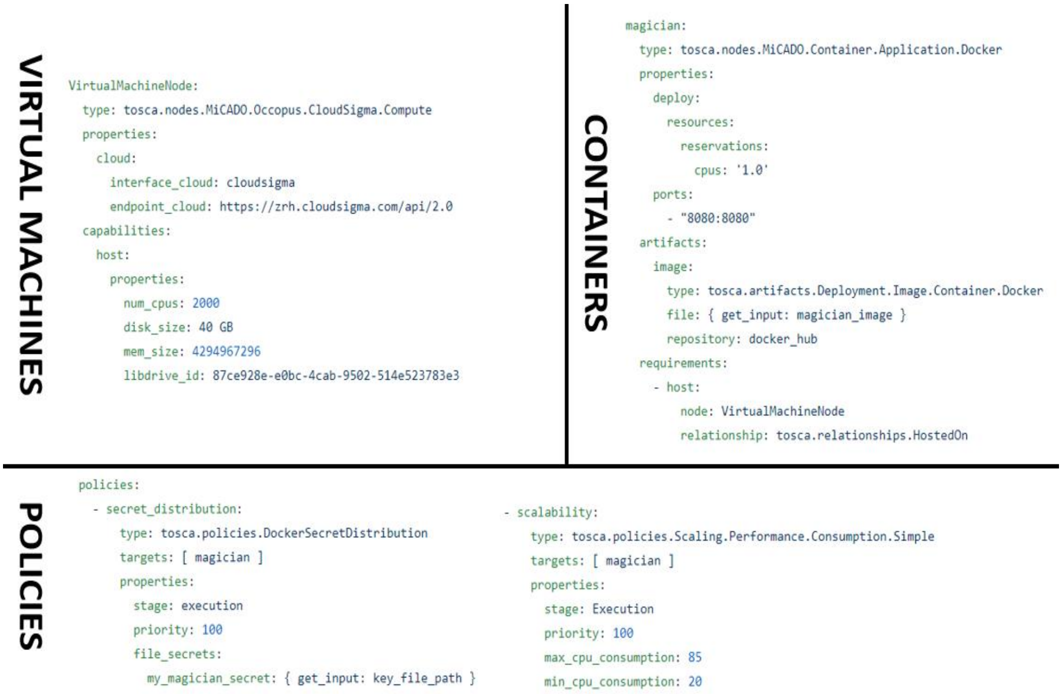


Figure 8 – Magician Application

5.2 Description of the Magician Application with an ADT

Magician has several requirements that need to be described in the ADT to support its deployment and execution. First, the application requires a two-layer topology made up of Docker containers running inside Virtual Machines. Second, a scaling policy is needed that allocates and releases resources and containers at certain CPU thresholds. Third, a security policy is required that controls access to sensitive data at runtime.

In order to meet these requirements, standard TOSCA normative types were used alongside custom defined types which would support the specific technologies used in the MiCADO implementation that performs the required deployment, monitoring and scaling functionalities. As depicted in Figure 9, first, virtual machines are described in the *VirtualMachineNode* section using TOSCA normative *compute* nodes which have been extended to support a specific cloud orchestrator (Occopus [27]) and a specific IaaS cloud (CloudSigma [30]). This description features capabilities that match the CPU, memory and storage resources that the virtual machine must provide. Next, the Docker containers and their specific deployment properties are specified as custom types, which derive from the normative TOSCA type for *container* nodes, in the *Magician* section. At least two containers are needed to run Magician: one for the Semantic Processing service and another one for the Classifier service. These container nodes also describe the specific requirement of *host*, which links containers to virtual machines using the TOSCA normative *HostedOn* relationship type. The processing container should be scaled up and down considering the workload. The scaling policy is defined as a custom type which derives from the normative TOSCA type for scaling policies in the *Scalability* section. The scaling policy specifies the minimum and maximum CPU utilisation as 20% and 85% respectively, to guide the scaling operation. This means that when average CPU utilisation is above 85% then a new container is launched, and the infrastructure is scaled up. Similarly, when CPU usage falls below 20% then a container is released and the infrastructure scales down. The security policy is specified as a custom type in the *Secret_Distribution* section. It is derived from the *TOSCA Security* policy. All these policies are related to the Semantic Processing service containers, and there is no specific policy assigned to the Classifier container.



```
VirtualMachineNode:
  type: tosca.nodes.MiCADO.Occopus.CloudSigma.Compute
  properties:
    cloud:
      interface_cloud: cloudsigma
      endpoint_cloud: https://zrh.cloudsigma.com/api/2.0
  capabilities:
    host:
      properties:
        num_cpus: 2000
        disk_size: 40 GB
        mem_size: 4294967296
        libdrive_id: 87ce928e-e0bc-4cab-9502-514e523783e3
```

```
magician:
  type: tosca.nodes.MiCADO.Container.Application.Docker
  properties:
    deploy:
      resources:
        reservations:
          cpus: '1.0'
    ports:
      - "8080:8080"
  artifacts:
    image:
      type: tosca.artifacts.Deployment.Image.Container.Docker
      file: { get_input: magician_image }
      repository: docker_hub
  requirements:
    - host:
        node: VirtualMachineNode
        relationship: tosca.relationships.HostedOn
```

```
policies:
  - secret_distribution:
      type: tosca.policies.DockerSecretDistribution
      targets: [ magician ]
      properties:
        stage: execution
        priority: 100
        file_secrets:
          my_magician_secret: { get_input: key_file_path }
```

```
- scalability:
    type: tosca.policies.Scaling.Performance.Consumption.Simple
    targets: [ magician ]
    properties:
      stage: Execution
      priority: 100
      max_cpu_consumption: 85
      min_cpu_consumption: 20
```

Figure 9 – Application Description Template Instance for Magician.

5.3 Processing the Application Description Template

In order to process the generated ADT, we implemented a first prototype of the Application Submitter (see the MiCADO Reference Architecture on Figure 7). While the final version of the submitter will be richer in functionality, this version is sufficient to provide evidence regarding the applicability of the ADT concept. The generic architecture of the submitter is shown in Figure10. It is derived from the MiCADO Reference Architecture illustrated in Figure 7. In the current submitter implementation, the Cloud Orchestrator is hard-coded (i.e. no VM scalability is supported, only container scalability), which permits testing of container orchestration and policy enforcement in isolation. This is the reason why in Figure 10 there is no VM Adaptor, only a Docker Adaptor for container orchestration. The Application Submitter contains two Policy Enforcers: the Scaling Policy Enforcer and the Security Enforcer.

After submission, the ADT passes through the series of steps visualised in Figure 10. In the first step, the ADT is parsed and validated by the OpenStack TOSCA Parser [37], which checks whether the ADT follows the YAML syntax, and whether it adheres to the syntactic rules laid out in the TOSCA specification. Successful validation returns a complex Python object whose attributes and methods facilitate future processing of the template. This object is passed to a proprietary MiCADO Validator, which performs further validation to ensure that the ADT is compliant with the custom types that were described earlier. The Mapper component resolves any relative links and references in the Python object and separates the security-relevant sections of the template from the whole. Next, adaptors developed for each of the three end-components (Docker, Scaling Policy Enforcer and Security Enforcer) receive the newly parsed ADT and begin the translation step.

The container-level portion of the ADT is translated into the format of a Docker Compose file so that it may be processed by Docker Swarm. The scaling policy and security descriptions from the ADT are translated into configuration files which will be interpreted by their respective proprietary components, by the Scaling Policy Enforcer and the Security Enforcer.

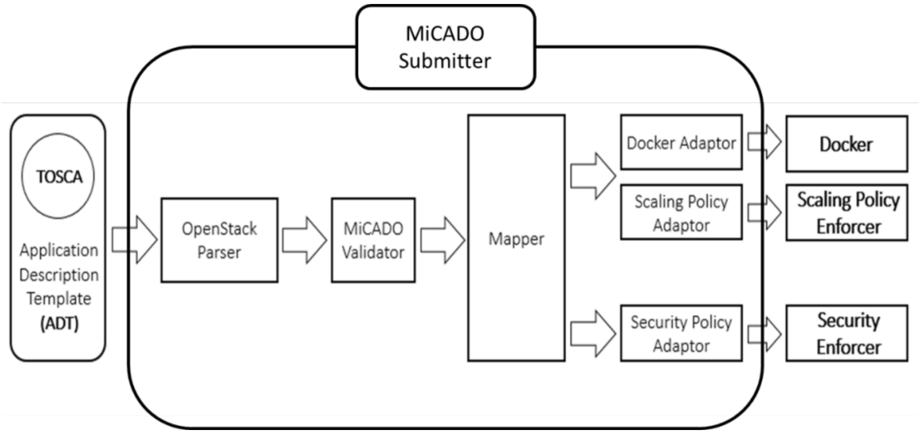


Figure 10 - Flow of ADT during processing by the Application Submitter

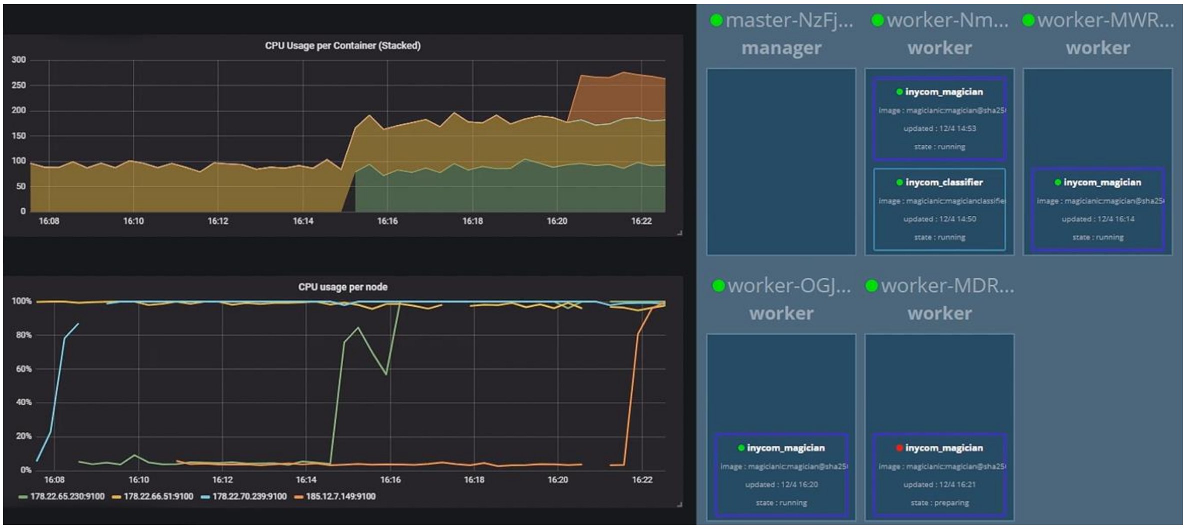
The translation of the container-level portion considers three basic sets of information within data provided by the Mapper. The first are the TOSCA properties defined within the description of containers. These properties should align with the runtime arguments that can be passed to Docker via the *docker run* command or via a Docker Compose file and should follow the naming conventions of the Docker Compose format. The second set of information is contained within TOSCA artefacts, which define external data which must be retrieved during orchestration. The image from which to build the container is described as an artefact with properties that define the image name, as well as the repository where it can be found. The final set of information to translate comes in the form of TOSCA relationships. Relationships, such as the `HostedOn` relationship used in the Magician ADT, describe how the various nodes defined within the ADT should interact with each other. Translation of the `HostedOn` relationship first defines a constraint inside the Docker Compose file, but also requires cooperation from the cloud orchestrator to ensure an appropriate reference is made for that constraint. TOSCA standards also define a `ConnectsTo` relationship between two containers, and an `AttachesTo` relationship for connecting a container to a block storage volume. Translating an `AttachesTo` relationship requires defining a new volume and providing an appropriate reference to that volume, both inside the Docker Compose file. On the other hand, translating a `ConnectsTo` relationship involves defining a new network and referencing that network under both of the connecting components, again inside the Docker Compose file. Translating the scaling policies into an appropriate format is straightforward as the scaling component configuration file is also in YAML. Translation involves creating a new key in the scaling configuration with the name of the container to be deployed and the minimum and maximum CPU thresholds for scaling. The translation step for the Security Enforcer involves the creation of the Docker secret through API calls to Docker’s SDK.

Once the translation is complete, the submitter instructs the related components to deploy and manage the application as specified in the policies. In the current prototype Docker Swarm [34] is responsible for the deployment of containers. We implemented a simple Scaling Policy Enforcer that uses alerts generated by a Prometheus-based monitoring system [35] to monitor the applications’ behaviour. The proof of concept Security Enforcer leverages the secrets feature built into Docker.

To launch the application and enforce the necessary policies, the Application Submitter begins the execution step. The three components are executed in sequence and point to the configuration files which were created during the translation step. First, the Docker Compose file is passed to Docker Swarm and executed with the *docker deploy* command. Next, the Scaling Policy Enforcer generates an alert based on the new information in the configuration file by reloading the Prometheus monitoring to check whether any scaling operation is needed. Finally, the Security Enforcer is executed and makes API calls to pass the sensitive data along a secure channel to the deployed container.

5.4 Orchestrating Magician in the Cloud

In order to test that deployment has been successful, and that the intended policies are indeed being enforced, several tools are used to monitor the application at runtime. Grafana [38] is an open-source graphing tool which offers out-of-the-box support for the graphical display of metrics from the Prometheus monitoring system. Grafana is used to show CPU usage as it surpasses the scaling thresholds set out in the ADT policies. Prometheus itself is utilised to ensure that accurate alerts have been generated by the scaling policy adaptor. Lastly, to provide a real-time visualisation of containers as they scale, Docker Swarm Visualiser, which connects directly to the Docker socket, is applied. Please note that as it was explained earlier, in this implementation only containers are scaling as the Cloud Orchestrator component is hard-coded.



master-NzFj...
manager

worker-Nm...
worker

inycom_magician
image : magicianic:magician@sha25
updated : 12/4 14:53
state : running

inycom_classifier
image : magicianic:magicianclassifie
updated : 12/4 14:50
state : running

worker-MWR...
worker

inycom_magician
image : magicianic:magician@sha25
updated : 12/4 16:14
state : running

worker-OGJ...
worker

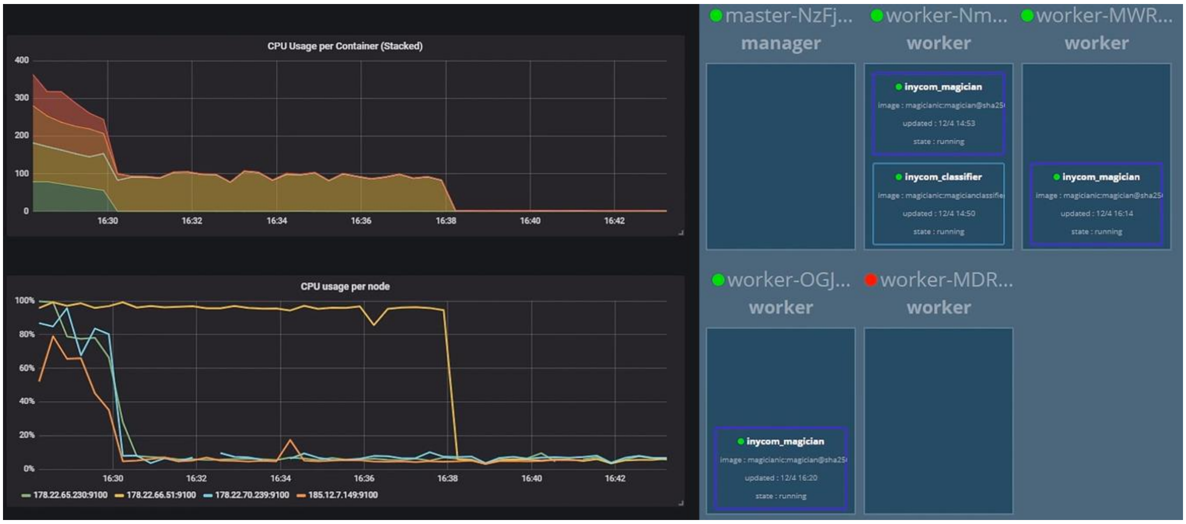
inycom_magician
image : magicianic:magician@sha25
updated : 12/4 16:20
state : running

worker-MDR...
worker

inycom_magician
image : magicianic:magician@sha25
updated : 12/4 16:21
state : preparing

Figure 11 - Magician after launch and during the scale-up phase

After deploying Magician on the CloudSigma cloud [30] it must be connected to an external database to begin data mining. Once connected, data mining starts, and CPU resources will be consumed. After a period of mining, Magician enters a sleep phase, waits for more data to be consolidated in the database and then begins mining again. Figure 11 shows the scaling-up stage of the Magician lifecycle after the application has been deployed and after the connection to the database has been established. The scaling policies are taking effect and the framework is scaling-up in response to the container CPU usage (in the upper-left graph) being well above the set threshold of 85%. At 16:08 a second (manually provisioned) virtual machine begins to pull the Magician image from an external repository as the container continues to scale-up. At 16:15 the pull completes, and the second Magician container begins mining data as well. Other virtual machines, which have also been started manually, begin pulling the Magician image as the containers scale, at 16:14 and 16:22. The first of these can been seen completing at 16:20 when the new container begins to mine and the CPU usage increases. The Docker-Swarm-Visualiser shows all four *worker* virtual machines and the status of the containers within them – either *running* or *preparing*. The Magician image on the fourth worker machine can still be seen in a *preparing* state since the pull has not yet finished. The Classifier container can be seen running on the first virtual machine and it is not scaling with the rest of the deployment.



master-NzFj... manager

worker-Nm... worker
- inycom_magician
  image : magicianic:magician@sha25
  updated : 12/4 14:53
  state : running
- inycom_classifier
  image : magicianic:magicianclassifie
  updated : 12/4 14:50
  state : running

worker-MWR... worker
- inycom_magician
  image : magicianic:magician@sha25
  updated : 12/4 16:14
  state : running

worker-OGJ... worker
- inycom_magician
  image : magicianic:magician@sha25
  updated : 12/4 16:20
  state : running

worker-MDR... worker

Figure 12 - Magician during the end of its execution, during a scale-down phase.

Figure 12 shows the scaling down stage of the lifecycle, where Magician enters sleep mode and the Scaling Policy Enforcer begins a scale down response. CPU usage by containers can be seen dropping as there are fewer database operations to carry out. Finally, with all mining tasks completed, all CPU usage drops, and containers are removed one by one from the infrastructure. Virtual machines which have already been manually shut down are shown with a red light in the Docker Swarm Visualiser.

The above experiment has successfully demonstrated that a suitable Application Submitter can be written that reads and processes the designed ADT, and that is capable of deploying the described infrastructure and enforcing the defined policies. Further experimentation is currently ongoing and long-term pre-production runs are planned to further investigate how such solution eliminates system crashes and reduces costs.

6. Conclusion and future work

This paper described how a technology agnostic application description allows the definition of complex topologies and specification of an extendable set of policies based on the TOSCA language specification. We elaborated the Application Description Template (ADT) that can be processed by various deployment and run-time orchestrators. ADT describes applications to be deployed and executed in two levels: in containers and/or in virtual machines. Additionally, ADT enables defining scale up/down rules allowing adding and removing containers to/from virtual machines during scaling up/down operations. The TOSCA policy hierarchy was extended with several scalability and security policies, such as advanced consumption-based scalability, consumption-based budget constrained scalability, firewall setting, and secret management policy to handle deployment, performance, scalability and security requirements of applications. We defined a generic reference architecture, the MiCADO Reference Architecture, that processes Application Description Templates to deploy and manage applications in the Cloud in a platform agnostic way. The implemented proof of concept prototype has demonstrated the viability of the technology-agnostic approach and the way a TOSCA-based ADT can be used to describe application topologies and delegate to a component (Application Submitter) the translation to formats that are technology-specific. The use case presented in this paper has demonstrated that the designed ADT was capable of describing a multi-node topology of a fairly complex commercial application with specific policies.

As ongoing and future work, the MiCADO application-level orchestration framework is currently being extended to implement further functionalities such as flexible submission lifecycle management and support for more detailed policies through an advanced Policy Keeper component. The Application Submitter is being embedded into the MiCADO architecture and it will connect the ADTs with the application-level orchestration features of MiCADO. Additionally, over 20 applications (both commercial and scientific applications) are being described with ADTs and prototyped with MiCADO within the COLA project.

Acknowledgements

This work was funded by the European Commission's H2020 COLA (Cloud Orchestration at the level of Applications) Project No. 731574 and ASCLEPIOS (Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare) Project No. 826093 projects.

References

[1] S. Khurana and A. G. Verma, "Comparison of Cloud Computing Service Models: SaaS, PaaS, IaaS," IJECT Vol. 4, Issue Spl-3, April-June 2013.

[2] Outsourcery and Cloud Industry Forum, "Cloud UK White Paper 16: Cloud Adoption Trends in the UK Public Sector - 2015," [Online]. Available: https://www.cloudindustryforum.org/content/cloud-adoption-trends-uk-public-sector-2015, [Accessed 28 April 2020].

[3] F. Leymann, "Cloud computing" it-Information Technol. Methoden und Innov. Anwendungen der Inform. und Informationstechnik, vol. 53, no. 4, pp. 163–164, 2011.

[4] "TOSCA Simple Profile in YAML Version 1.0." [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csd03/TOSCA-Simple-Profile-YAML-v1.0-csd03.html. [Accessed: 20-Jun-2018].

[5] "Learn Template Basics - AWS CloudFormation." [Online]. Available: http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide [Accessed 28 April 2020].

[6] "ARM Template Documentation" [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/, [Accessed 28 April 2020].

[7] Y. Gao, K.Yu, "Design and Implement a Self-Enabled Private Cloud: Oracle Enterprise Manager 12" Oracle Open World 2015.

[8] "Chef - Automate IT Infrastructure | Chef." [Online]. Available: https://www.chef.io/chef/. [Accessed: 20-Jun-2018].

[9] C. Butler, "Automating Orchestration in the Cloud with Ubuntu Juju" USENIX Configuration Management Summit 2014 (UCMS '14), June 19, 2014, Philadelphia, USA.

[10] "Heat - OpenStack." [Online]. Available: https://wiki.openstack.org/wiki/Heat. [Accessed: 29-Mar-2017].

[11] T. Kiss, P. Kacsuk, J. Kovacs, B. Rakoczi, A. Hajnal, A. Farkas, G. Gesmier, and G. Terstyanszky, "MiCADO –Microservice-based Cloud Application-level Dynamic Orchestrator," Future Generation Computer Systems, Vol 95, pp 937-946, May 2019.

[12] "Cloud Application Management for Platforms Version 1.1," [Online]. Available : http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html, 9 November, 2014 [Accessed 28 April 2020].

[13] G. Breiter, M. Behrendt, M. Gupta, S. D. Moser, R. Schulze, I. Sippli, and T. Spatzier: Software defined environments based on TOSCA in IBM cloud implementations, IBM Journal of Research and Development, Volume: 58 , Issue: 2/3 , March-May 2014.

[14] M. Caballer, S. Zala,·A. L. Garcia,·G. Molto, P. O. Fernandez·and M. Velten: Orchestrating Complex Application Architectures Inheterogeneous Clouds, Journal of Grid Computing, March 2018, Volume 16, Issue 1, pp 3–18.

[15] J. Wettinger, U. Breitenb ̈ucher and F. Leymann: Standards-based DevOps Automationand Integration Using TOSCA, in Proceedings of the 7th International Conference on Utility and Cloud Computing (UCC 2014), p 59-68.

[16] T. Binz et al., "OpenTOSCA – A Runtime for TOSCA-based Cloud Applications." 11th International Conference on Service-Oriented Computing, December 2013.

[17] J. Soldani, T. Binz, U. Breitenbücher, F. Leymann, and A. Brogi, "ToscaMart: A method for adapting and reusing cloud applications," J. Syst. Softw., vol. 113, pp. 395–406, Mar. 2016.

[18] Cloudify, "Cutting Edge Orchestration." [Online]. Available: https://cloudify.co/. [Accessed: 5-Mar-2019].

[19] Apache, "About ARIA TOSCA." [Online]. Available: http://ariatosca.incubator.apache.org/. [Accessed: 5-Mar-2019].

[20] Puccini - Deliberately stateless cloud topology management and deployment tools based on TOSCA." [Online]. Available: https://github.com/tliron/puccini. [Accessed: 5-Mar-2019].

[21]` "ALIEN 4 Cloud." [Online]. Available: http://alien4cloud.github.io/. [Accessed: 5-Mar-2019].

[22] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann, and J. Wettinger, "Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA." 2014 IEEE International Conference on Cloud Engineering, *Boston, MA, 2014, pp. 87-96*

[23] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann, and U. Breitenb, "Winery – A Modeling Tool for TOSCA-based Cloud Applications." 8274. 10.1007/978-3-642-45005-1_64.

[24] U. Breitenbücher, T. Binz, O. Kopp, and F. Leymann, "Vinothek - A self-service portal for TOSCA," ZEUS 2014.

[25] T. Waizenegger et al., "Policy4TOSCA: A Policy-Aware Cloud Service Provisioning Approach to Enable Secure Cloud Computing," Springer, Berlin, Heidelberg, 2013, pp. 360–376.

[26] K. Képes, U. Breitenbücher, M. Philipp Fischer, F. Leymann and M. Zimmermann: Policy-Aware Provisioning Plan Generation for TOSCA-based Applications, In Proc of 11th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE 2017)

[27] J. Kovács and P. Kacsuk, "Occopus: a Multi-Cloud Orchestrator to Deploy and Manage Complex Scientific Infrastructures," Journal of Grid Computing, vol. 16, no. 1, pp. 19–37, Mar. 2018.

[28] "About – COLA Project – Cloud Orchestration at the Level of Application." [Online]. Available: http://www.project-cola.eu/cola-project/. [Accessed: 27-Mar-2017].

[29] Amazon EC2, Amazon, [Online], available https://aws.amazon.com/ec2, [Accessed: 28-April-2020]

[30] "Cloud Hosting Pricing | CloudSigma." [Online]. Available: https://www.cloudsigma.com/pricing/. [Accessed: 27-Jun-2018]

[31] OpenStack - Built the Future of Open Infrastructure, [Online], available www.openstack.org, [Accessed: 28-April-2020]

[32] OpenNebula, [Online], available https://opennebula.org/, [Accessed: 28-April-2020]

[33] "CloudBroker GmbH | Compute-intensive applications in the cloud." [Online]. Available: http://cloudbroker.com/. [Accessed: 27-Jun-2018]

[34] "Docker Swarm overview | Docker Documentation." [Online]. Available: https://docs.docker.com/swarm/overview/. [Accessed: 20-Jun-2018].

[35] Prometheus web page [Online] Available: https://prometheus.io/ [Accessed: 5 May 2018]

[36] "Inycom | Tecnología e Innovación para tu Negocio." [Online]. Available: https://www.inycom.es/. [Accessed: 18-Mar-2018].

[37] "TOSCA-Parser - OpenStack." [Online]. Available: https://wiki.openstack.org/wiki/TOSCA-Parser. [Accessed: 29-Oct-2017].

[38] "Grafana - The open platform for analytics and monitoring." [Online]. Available: https://grafana.com/. [Accessed: 28-Jun-2018].

[39] G. Pierantoni, T. Kiss, G. Terstyanszky: Towards Cloud Application Description Templates Supporting Quality of Service, in proceedings of IWSG 2017, 9th International Workshop on Science Gateways, 19-21 June, 2017, Poznan, CEUR Workshop Proceedings, Vol 2363, ISSN1613-0073

[40] "Ansible is Simple IT Automation" [Online]. Available: https://www.ansible.com/. [Accessed: 29-Apr-2020].