Cloud Orchestration at the Level of Application

Project Acronym: **COLA**

Project Number: **731574**

Programme: **Information and Communication Technologies Advanced Computing and Cloud Computing**

Topic: **ICT-06-2016 Cloud Computing**

Call Identifier: **H2020-ICT-2016-1**
Funding Scheme: **Innovation Action**

Start date of project: 01/01/2017 Duration: 30 months

Deliverable:

D5.3 Integration of the Templates with the Selected Application Description Approach

Due date of deliverable: 31/10/2017 Actual submission date: 03/11/2017

WPL: Gabriele Pierantoni

Dissemination Level: PU

Version: Final

1. Table of Contents

2. List of Figures and Tables

Figures

3. Status, Change History and Glossary

Status:	Name:	Date:	Signature:
Draft:	Gabriele Pierantoni	22/10/17	Gabriele Pierantoni
Reviewed:	Alex Worrad Andrews	29/10/17	Alex Worrad Andrews
Approved:	Tamas Kiss	03/11/17	Tamas Kiss

Table 1, Status Change History

Version	Date	Pages	Author	Modification
V0.1	13/10	ALL	G. Pierantoni	Skeleton
V0.2	19/10	ALL	G. Pierantoni	Architecture and Policies diagrams and tables
V0.3	20/10	ALL	G. Pierantoni	Formatting and list of todos
V1.0	21/10	ALL	G. Pierantoni	First Complete Draft
V1.1	22/10	ALL	G. Pierantoni	Addressed some of Gabor's Corrections and feedback
V1.2	29/10	ALL	G. Pierantoni	Addressed Gabor and Tamas Corrections and feedback for Sections 5,6 and part of 7
V1.3	30/10	ALL	G. Pierantoni	Addressed Gabor and Tamas Corrections and feedback for Sections 8 and 9, updated Outlandish Use case based on the feedback of 30.10.17 Teleconference
V1.4	31/10	ALL	G. Pierantoni	Addressed Gabor and Tamas Corrections and feedback for Sections 8 and 9, updated Outlandish Use case based on the feedback of 30.10.17 Teleconference
V1.5	01/11	ALL	G. Pierantoni	Addressed Gabor and Jose Comments
V1.6	02/11	ALL	G. Pierantoni	Some final corrections
V1.7	03/11	ALL	G. Terstyanszky G. Pierantoni	Final Corrections

Table 2, Deliverable Change History

4. Glossary

API	Application Programming Interface
CAMP	Cloud Application Management for Platforms
COLA	Cloud Orchestration at the level of Application
CLI	Command Line Interface
DoW	Description of Work
GUI	Graphical User Interface
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
TOSCA	Topology Orchestration Specification for Cloud Application

Table 3, Glossary

5. Introduction

COLA DoW specifies **Deliverable D5.3 "*Integration of the Templates with the selected application description approach*"** as follows: "This deliverable will report on the implementation of the application template and its integration with the selected application description approach."

It describes how the abstract Application Descriptions Templates developed by Work Package 5 are integrated to implement three Use Cases proposed by the COLA Industrial and Academic partners. Albeit limited in their scope, these initial three use cases are representative both of the applications that will be ported to the COLA infrastructure and of the capabilities of the selected approach to the Application Description Templates.

Whenever an existing application is ported to the COLA infrastructure (or an entirely new one is developed), all the different facets of the application have to be expressed and catered for by this infrastructure. These entail the description of two main characteristics of an application:

- the **Topology** that represents its components and how they are connected, and,
- the **Policies** that represent how the modalities of the various steps of the application lifecycle will be implemented for example how will be deployed, executed and un-deployed.

The Application Description Templates developed in COLA, capture these two characteristics of applications. The management of the applications by the infrastructure will, in turn, require certain infrastructure-level functionalities (E.g. Security Infrastructure, Container Management, Monitoring, etc.) that are developed by other work packages.

As the COLA infrastructure is currently being developed and the Cloud Orchestrator (MiCADO [1][2]) component does not yet parse a fully TOSCA [3] compliant description, it is impossible at the moment to directly test the deployment and execution of the selected applications with the Application Description Templates described in this document. Instead of this direct test (which will be implemented with the future releases of MiCADO), the validity of the implemented Application Description Templates has been tested with two different approaches:

- **Syntactical Consistency** has been tested by running the developed templates with the OpenStack TOSCA parser [4], and,
- **Semantic Consistency** has been tested by manually checking that the information expressed by the Application Description Templates matches the behaviour of the current implementation of the MiCADO framework and is not contradictory with the specifications of the Security Infrastructure and future MiCADO releases.

This deliverable is the third deliverable of Work Package 5 "Application Description Templates" and is an open document which visibility is allowed to both internal and external readers. The intended audience of this deliverable is application developers those are involved in either developing new applications or porting existing ones to the COLA infrastructure.

Deliverable D5.3 is structured as follows:

1. **Section 5** – Offers an introduction to the Deliverable introducing general concepts and its relevance within the COLA project.

2. **Section 6** – Further details the topics introduced in **Section 5** by describing the mutual dependencies among this Deliverable with the other most relevant Project Deliverables.
3. **Section 7** – Recapitulates the COLA approach to Application Description Templates describing how the proposed Templates span three conceptual levels for each applications: Application Level, Service Level and Resource Level and how the behaviour of each of the application components can be specified by different policies.
4. **Section 8** – Introduces the three Use Cases.
5. **Section 8.1** – Describes the Application Description Template for Use Case 1 - Scalable hosting, testing and automation for SMEs and public sector organisations - Outlandish and The Audience Agency
6. **Section 8.2 –** Describes the Application Description Template for Use Case 2 - Bursting onto the Cloud from SakerGrid – Brunel University and Saker Solutions.
7. **Section 8.3** – Describes the Application Description Template for Use Case 3 - Social media data analytics for public sector organisations Inycom and SARGA
8. **Section 9 –** Offers information on the implementation details of the components that process the Application Description Templates**.**
9. **Section 10** – Concludes the Deliverable by drawing some final remarks. The relevance of the implementation of these three Use Cases is analysed in relation to the generic description of applications and the various components of the COLA architecture.

6. Relationship with other Work Packages and Deliverables

As introduced in Section 5, the **Application Description Templates** are closely related to various fundamental aspects of the COLA project:

- the applications that are to be ported to the COLA infrastructure,
- the COLA infrastructure itself, and,
- the abstract approach to describe the applications.

As a result, this deliverable is closely related to Work Packages and Deliverables that describe the concepts highlighted in the previous paragraph.

Deliverable 5.3 has been published by **Work Package 5 – Application Definition Templates.** This work package has published two Deliverables: **D5.1 – "*Analysis of existing Application Description Approaches*"**, and **D5.2 – "*Specification of the Application Description Concept"***. D5.1 offers a state of the art overview of the application description and execution while D5.2 describes the COLA proposed approach to the problem: a three-layered Application Description Template based on the TOSCA [5][6] language specifications which also defines policies at each of its layers.

The Application Description Templates are interpreted by the MiCADO framework, hence the dependencies with **Work Package 6 – "MicroServices deployment and execution layer"**, particularly with Deliverables **D6.1 – "*Prototype and Documentation of the Cloud Deployment Orchestrator Service*"** and **D6.2 - "*Prototype and Documentation of the Monitoring Service*"**).

The Application Description Templates are also closely related to security issues and concerns, hence the dependencies with **Work Package 7 – "Security, privacy and trust at the level of cloud applications",** particularly with, Deliverables **D7.1 – "*Security Requirements*"** and **D7.2 – "*Security Architecture Specifications*"**.

Mutual dependencies among WP5, WP6 and WP7 are of a technical nature and it is important that the information defined in the Application Description Templates can be understood, acted upon and enforced by the MiCADO framework, particularly by the Cloud Orchestrator and the Security Infrastructure.

Deliverable D5.3 is also closely linked to **Work Package 8 – "SME and public sector use-case pilots and demonstrators**, particularly with Deliverable **D8.1 – "Business and Technical Requirements of COLA Use Cases"**. WP5 interactions with WP8 will ensure that the Use Cases of Work Package 8 can be supported by the Application Descriptions Templates.

7. Application Descriptions within COLA and the three Use Cases

This section briefly recapitulates the salient aspects of the Application Description Templates approach detailed in **Deliverable D5.2 – *"Specification of the Application Description Concept"***. It also contextualizes its implementation with reference of the COLA architecture and its current implementation status.

As mentioned in Section 5 Introduction, describing an application in COLA covers two main and equally important requirements:

- the **Topology** that represents the graph of the various components that constitute each application, and,
- the **Policies** that govern how the application is managed by the infrastructure during its lifecycle.

In addition to these two fundamental requirements, there are further requirements that helped in the definition of the following design guidelines:

- The Application Description Templates must minimize the application developers' efforts required to specify applications. This can be achieved by decomposing the application's topology and policies into components that are as re-usable as possible by application developers. See Fig. 1.
- The Application Description Templates must support the definition of policies that regulate the deployment, execution and un-deployment of the application.
- The Application Description Templates must support an incremental development of the COLA architecture.
- The Application Description Templates should reflect the conceptual layers of an application (Application , Service and Resource layer) and the related concepts in Cloud Computing: Software as a Service (PAAS), Platform as a Service (PAAS) and Infrastructure as a Service (IAAS) [7].

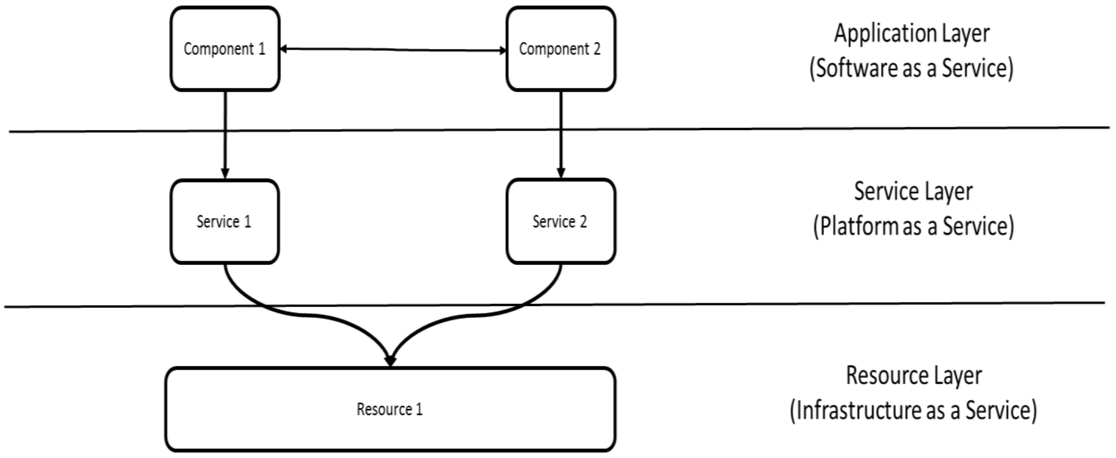


Figure 1, Example of the COLA three layers

The **Topology** (alongside with the policies that will be described later in this Section) are described with Application Description Templates that are used by various components of the COLA architecture as illustrated in Figure 2.

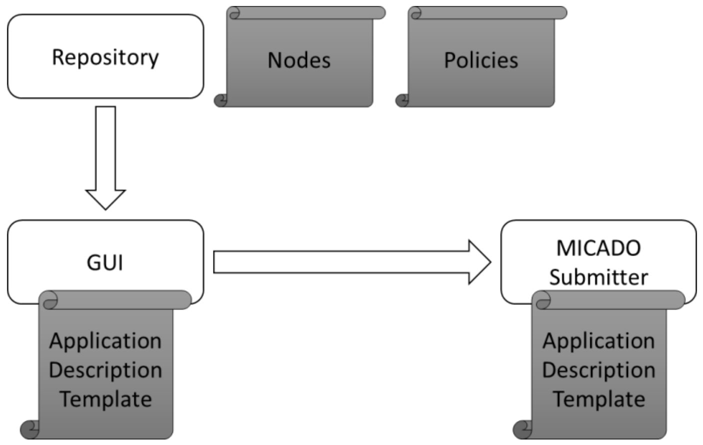

Repository
Nodes
Policies
GUI
Application Description Template
MICADO Submitter
Application Description Template

Figure 2, Interaction between Application Description Templates and the relevant components of the COLA Platform.

Application Developers can build Application Description Templates (Application Topologies) by assembling existing Nodes Types and setting their values (or overriding default values).

Policies are selected and composed in a similar fashion and attached to the relevant nodes of the topology. This approach maximizes re-usability as Application Developers will be able to re-use entire topologies (when they are available), nodes and policies.
Once an Application Developer has defined the Application Description Template, it is passed to the MiCADO framework that will care of the management of application throughout its entire lifecycle in compliance to the defined policies. The interactions of the Application Description Template with the MiCADO platform are detailed in Figure 3.

a) Application Topologies

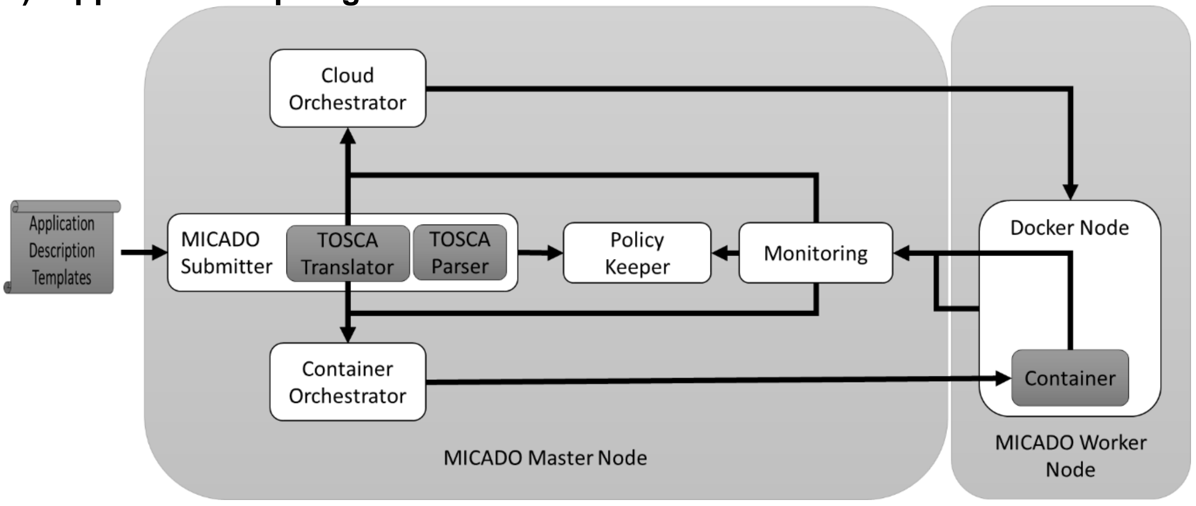


Figure 3, Application Description Templates and the COLA Architecture

The MiCADO Submitter [8] will contain a TOSCA Parser and TOSCA Translator. The **TOSCA Parser** will check the validity of the YAML [9] code of the Application Description Templates. It will pass the TOSCA descriptions to the **TOSCA Translator** that will instruct three components: The Cloud Orchestrator, the Container Orchestrator and the Policy Keeper.
For the moment being, the Cloud Orchestrator is implemented as the latest release of Occopus [10], and the Container Orchestrator as an instance of Docker Swarm [11]. The

current implementation of the **Cloud Orchestrator** provides automatic features for configuring and orchestrating distributed applications on single or multi cloud systems. Occopus can be used by application developers and application controllers to manage virtual infrastructures at deployment time and at runtime to create and provide complex virtual infrastructures. Occopus uses a descriptor that defines the services to be deployed in the Cloud and the order of their deployment. The **Container Orchestrator** is a clustering and scheduling tool for Docker containers that allows administrators and developers to manage a cluster of Docker nodes as a single virtual system.

The **Policy Keeper** will enforce the policies by matching its values to those provided by the components that monitor the infrastructure and will instruct the Cloud and Container Orchestrator accordingly

As introduced in the previous sections, the Application Description Templates cover two main aspects: the application topology and the policies which will be detailed in the following sections.

b) Application Topologies

COLA applications are composed of various elements (service instances) that may run on a single or multiple servers considering user demands. COLA describes these applications using the TOSCA meta-model which defines generalized and base node types such as application server, web server, database, compute node, storage node etc. These elements are arranged in three layers: application, service, and resource layer (an example of a standard Proxy-Service-Database application is described in Figure 4).

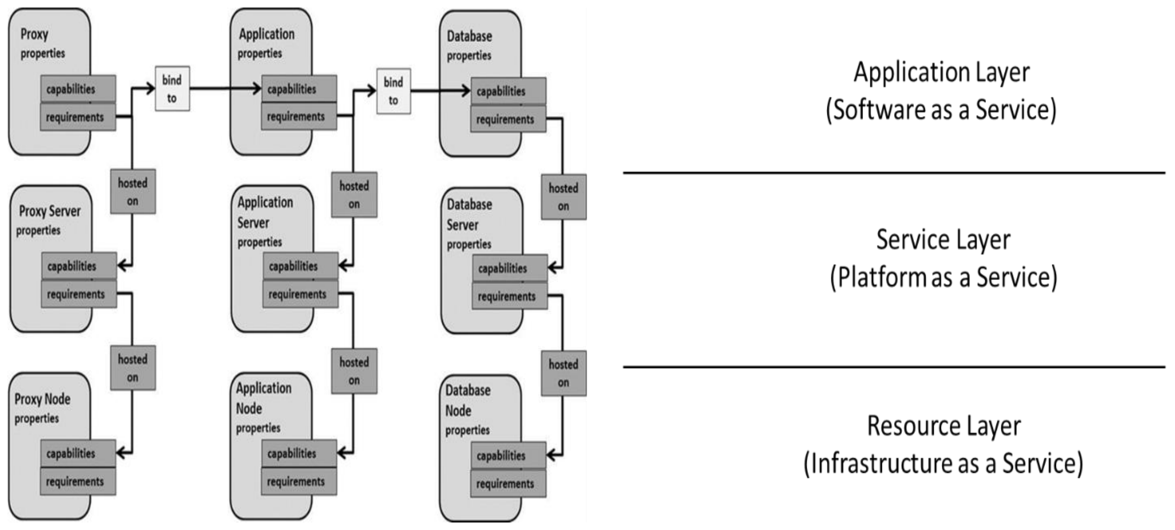

Proxy properties
capabilities
requirements
bind to
Application properties
Database properties
hosted on
Proxy Server properties
Application Server properties
Database Server properties
Proxy Node properties
Application Node properties
Database Node properties
Application Layer (Software as a Service)
Service Layer (Platform as a Service)
Resource Layer (Infrastructure as a Service)

Figure 4, TOSCA specification of the COLA application architecture

As COLA applications can run either on Linux or on Windows operating system, support for virtualization must be provided. Some Windows applications could be too heavy for Docker containers, and in this case, the MiCADO framework has to support running Linux based applications in containers and Windows-based application in virtual machines. But as container technology evolves, it is possible (if not likely) that it would be feasible to run all applications (both Linux and Windows) inside Docker Containers.

c) Policies

The description and enforcement of policies in TOSCA is an additional dimension to the description of the application topologies. Policies define and enforce the modalities that regulate the application lifecycle.

The design proposed in this Deliverable is based on the following assumptions and restrictions

- **Declarative Model**. The policies description follows the Declarative Model. They describe a policy but they do not imperatively describe how to implement the policy, which is left to the proper components of the COLA architecture
- **Policies can be selected and combined**. Users can select and combine policies from a basket of existing policies but cannot create new policies
- **Modular Policies**. Users can select and specify the parameters of atomic (that cannot be decomposed further) policies that cover the various features of the service. These policies are combined for each service to define overall policies of that service and the overall policies of each service are then merged in the overall policies that govern the complete service topology (the application).
- **User-defined Policy Parameters**. Users can define parameters of the selected policies
- **No Consistency is ensured**. Policies may be contradictive, at this stage of the policies description, only a priority level will be offered to the user to define which policy has the highest priority.

The concepts of modularity and consistency are further explained in Figure 5 below. Policies that define certain aspects of the entire application topology (and hence each of the services that compose the application) may raise conflicts with policies that define aspects of a specific service. Furthermore, policies that define the various aspects of the policies of individual services may also raise conflicts.

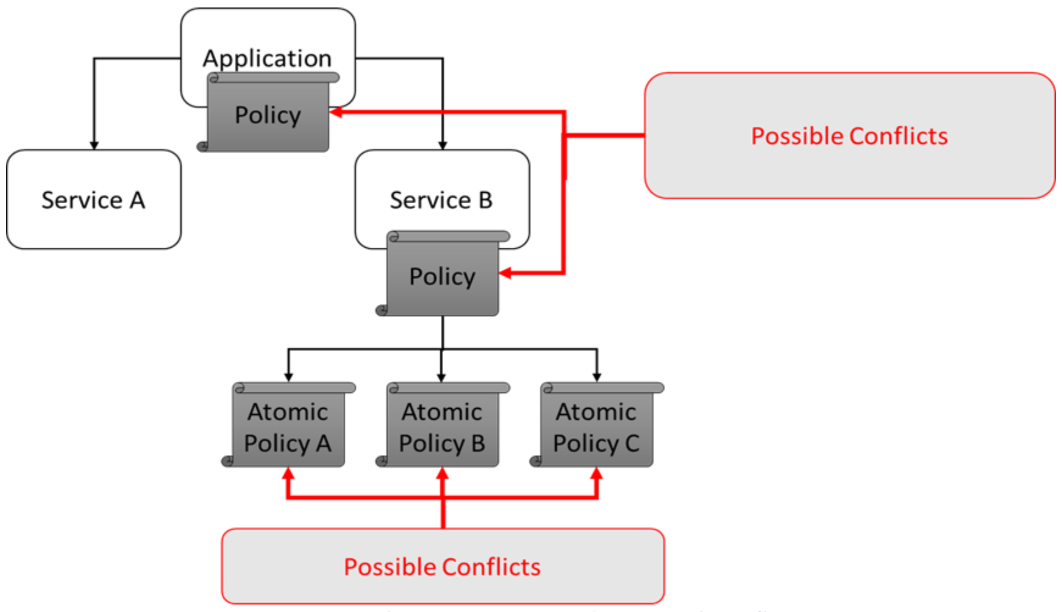


Figure 5, Policies Structure and Potential Conflicts

TOSCA allows the definition of type hierarchies of arbitrary complexity. COLA defines a three-layered hierarchy of policies that all derive from the TOSCA Root Policy. Each Policy Type will then be further detailed in its sub-types. A partial set of the Abstract Hierarchy of Policy Types is depicted in Figure 6.

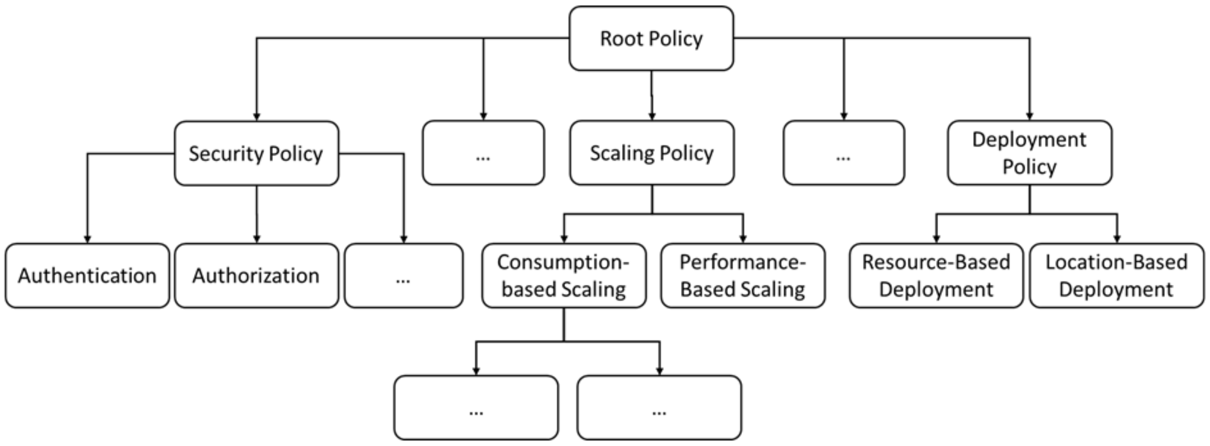

Root Policy
Security Policy
...
Scaling Policy
...
Deployment Policy
Authentication
Authorization
...
Consumption-based Scaling
Performance-Based Scaling
Resource-Based Deployment
Location-Based Deployment
...
...

Figure 6, Abstract Policy Hierarchy

After they have been selected from the policy hierarchy, policies have to be placed at the right level of the topology to define which elements of the topology will be regulated by it. This is obtained by linking the policy to the related node.

It is important to highlight how the policies placed at application level, may well be applied to lower levels of the topology. As an example, a scalability policy for an application, will affect all the services it is based upon and, ultimately, it will be enforced by deploying/undeploying containers at the resource level. To facilitate the application of policies at the various levels, each policy defines the nodes it has to be applied to. For each node, the overall policy is composed of sub-policies that describe the various features such as security, scalability, etc. These sub-policies are selected from the Abstract Policy Hierarchy of Figure 6. The structure and relationship of policies with the other component of the topology is illustrated in Figure 7.

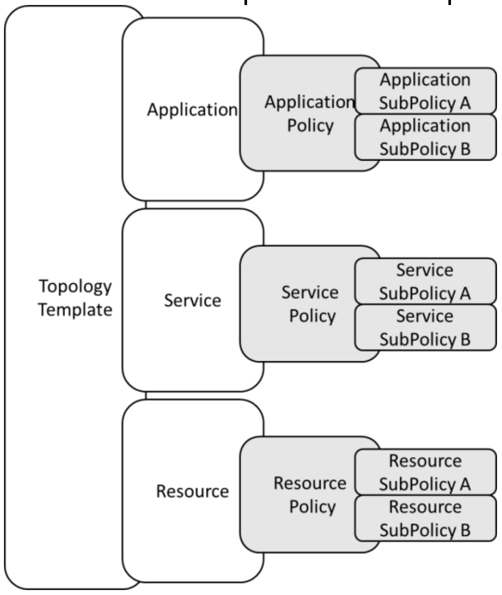


Figure 7, Policies and sub-policies at Application, Service and Resource Level

Finally, each policy value and element has to be described. WP5 has proposed the generic structure for policy description as drafted in Figure 8.

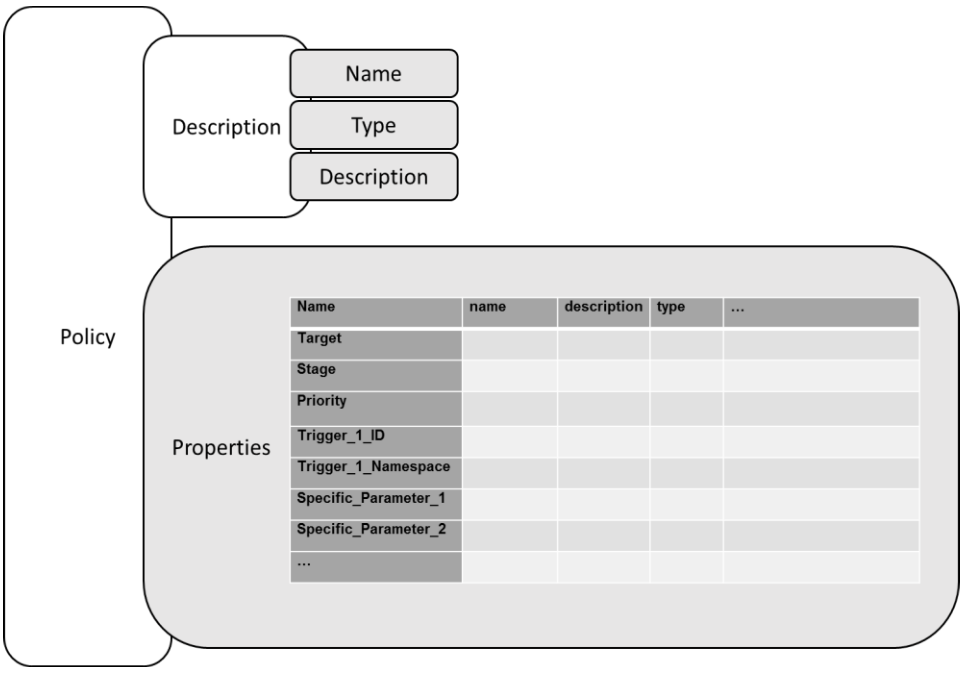


Figure 8, Generic Policy Structure

Each policy type is divided into two main sections:

- **A Description Section**: This section of the Policy is composed of different fields that give its overall description. It comprises meta-data composed of the following fields:
  - **Name**: A string that represents the name of the Policy.
  - **Type**: The type defined in the policy hierarchy. This field defines the link with the first level of policy structure (Abstract Policy Hierarchy Type).
  - **Description**: A textual description of the policy.

- **A Properties Section**: This section contains the actual data of the policy. Such data fall under two kinds of parameters: those that are common to all COLA policies types, and those that are specific to each policy type.
  - **Common Properties** that are present in all COLA Policies are:
    - **Target**: defines the Topology Layers to which the policy has to be applied.
    - **Stage**: defines at which stage of the lifecycle of the element the policy is applied.
    - **Priority**: This is an arbitrary integer of 0 to 100 used to define the priority with which the policy will be implemented. It is used to resolve possible conflicts with other policies.
  - **Specific Properties** are specific to each Policy. These parameters vary depending on the nature of the policy itself, as an example: a scalability policy based on a deadline will define the maximum amount of time in which the process will have to complete; a deployment policy will define the minimum number of CPUs required, etc.
    Properties are defined as a list of TOSCA properties definitions as specified in section 3.5.12 of the TOSCA Simple Profile[12] each contain various parameters definition including type, constraints and other useful fields.

8. Use Case Overview

Section 7 has described the language that COLA proposes to adopt to describe the applications. Still, this approach has to be implemented and integrated within COLA and this is an ongoing process.

Throughout the entire duration of the project, multiple applications will be ported into COLA. Three Use Cases must reach near production stage and up to twenty other scenarios will have to reach proof of concept stage; it is possible that the present approach to the Application Description Templates may have to be further modified to cater for all future applications and scenarios.

At the moment, the three Use Cases described in this Deliverable have been selected for their relevance to the SMEs and public sector organisations that participate in COLA. At the same time, the implementation of these Use Cases also represents the first proof of concept validation test for the proposed Application Description Templates. For this reason, the three Use Cases are related to a variety of aspects (complexity of the topology, type of policies and virtualization levels) that cover many of the project relevant aspects.

A brief overview of the three use cases is provided at the beginning of each Use Case Section while full details of each Use Case are available in Deliverable D8.1. The information contained in D8.1 has been further integrated and updated with “vis-à-vis” meetings during the COLA annual project meeting, teleconferences, and exchange of emails and draft notes. In each section, the Use Cases are briefly described, decomposed in their various components and a three-layered architecture with the related policies is described. For each of the components, implementation details of the policies are described.

8.1 Use Case1: Scalable hosting, testing and automation for SMEs and public sector organisations - Outlandish and The Audience Agency

Use Case 1 - Scalable hosting, testing and automation for SMEs and public sector organisations (in this case specifically for The Audience Agency's Audience Finder application) is an application that performs data-mining analysis regarding the audience of different sources, such as Theatres, Museums, etc.

This use case describes a three-tier application with a Web Interface, a Controller Module and a Database backend. Scalability of the application is based on two main aspects. First, the web interface has to respond to fluctuations in the number of connections, and second, the computational requirements needed to fulfil the queries that can vary significantly from case to case. In order to meet the computational requirements of the queries a Caching Service has been implemented to pre-calculate a set of queries to shorten the computational times. The Caching Service is executed at regular intervals (it is implemented as a Cron Job in Linux). The system is connected to two Databases. The Box Office Database holds the ticketing information and is used to share data between the AFA Web Application and the Caching Service. The second database is the WordPress database.

Use Case 1 is introduced in pages 5 to 17 of deliverable D8.1. This initial description was updated during interactions between WP5 leader, Outlandish and the Audience Agency. The overall architecture of Use Case 1 is drafted in Figure 9.

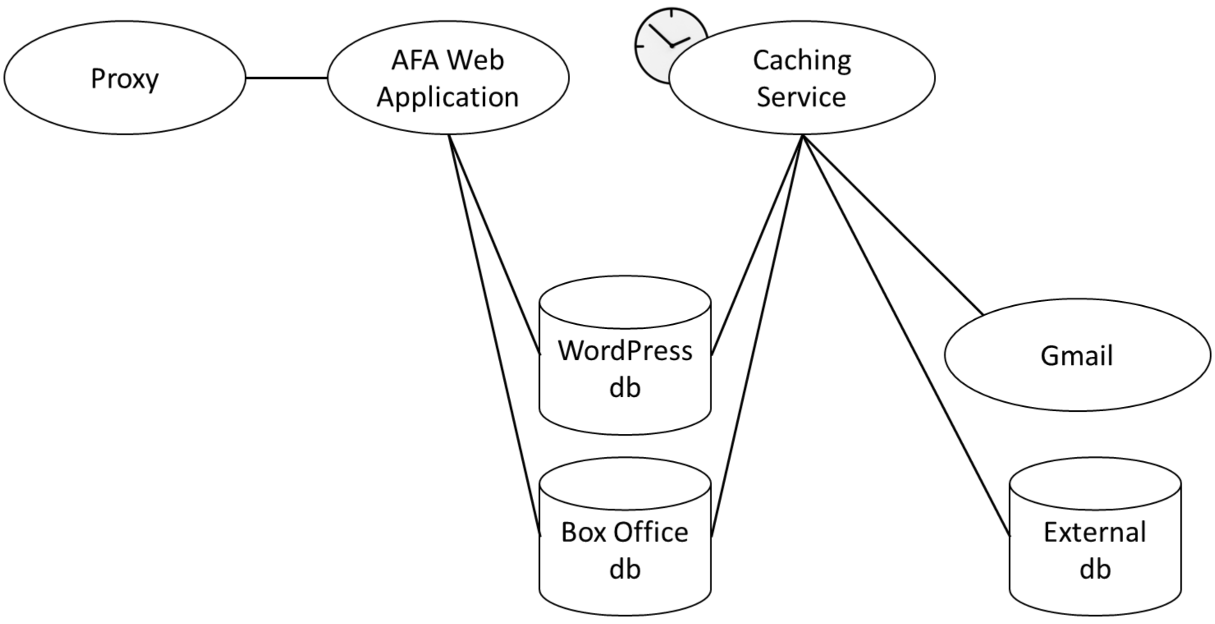


Figure 9, Architecture Overview of Use Case 1

The components are:

- **Proxy**: The proxy will offer a single point of contact to potentially multiple copies of the web interface (for scalability reasons) by redirecting the incoming connections to the proper instance of the AFA Web Application.
- **AFA Web Application**: The GUI with which the client interacts. The AFA Web Application is written in PHP and is developed on the WordPress framework.
- The **Caching Service** is based on the same technology of the AFA Web Application, it executes a set of queries that are pre-calculated from aggregate metrics and stored

in the Box Office Database. The Caching Service also connects to the WordPress External Database and to Gmail as it uses the OAuth Authorization System

- The **Box Office database** contains the source and target data of the Caching Service that is queried by the web application. It is a MySQL Database.
- The **WordPress database** contains all the configuration information for the AFA Web Application and the Caching Service.

The components fall into two separate categories: Internal (depicted in red colour) and External (depicted in blue colour) Components. Internal Components will be modelled with the COLA extension of the TOSCA language, while external components will be not be covered, and the COLA description and related deployment only have to ensure that such external services can be reached. The two components that will be ported into the MiCADO framework and have to be described in the Application Description Template are the AFA Web Application and the Caching Service. The Application Description Template will define the topology and policies drafted in Figure 10.

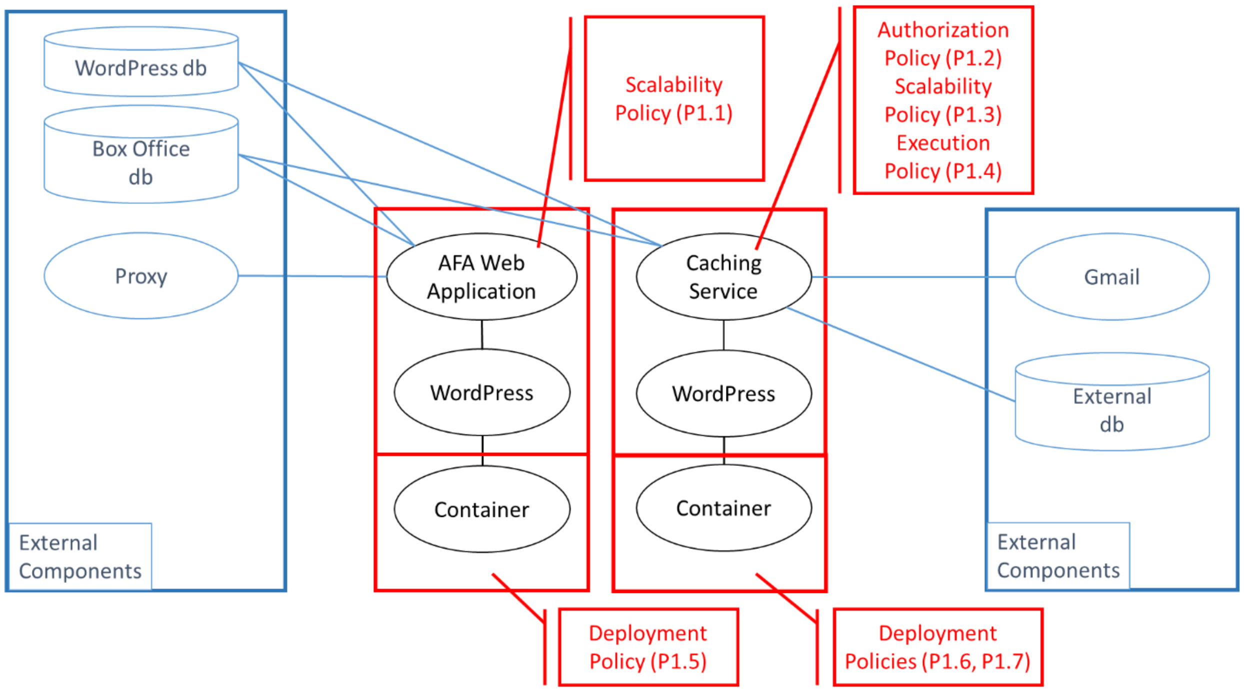


Figure 10, Architectural Overview of the Implementation Step A of Use Case 1

The policies that govern the deployment and scaling during execution of the various components are detailed in the table below.

Architectural Level	**AFA Web Application**	**Caching Service**
Application	**Scalability Policy (P1.1)** It defines the deployment of a new instance under condition that the number of inbound connections is greater that a defined threshold	**Authorization Policy (P1.2)** It defines that the component uses an external Authorization Service (This service is based on OAUTH and requires a connection to a GMAIL account) **Scalability Policy (P1.3)** It defines the deployment of a new instance under condition that the

		expected completion time of pre-computed datasets to be calculated is greater that the given threshold. **Execution Policy (P1.4)** It defines that the component has to be executed at fixed interval times
Service	**No specific Policy at this level**	**No specific Policy at this level**
Container	**Deployment Policy (P1.5)** It dictates that the container must be capable of reaching the set of addresses (with the related ports and protocols) needed by all the external components	**Deployment Policy (P1.6)** It dictates that the container must have at least the specified characteristics (CPU, RAM, DISK) **Deployment Policy (P1.7)** It dictates that the container must be capable of reaching the set of addresses (with the related ports and protocols) needed by all the external components

Table 4, Policies of Use Case 1

It is worth noticing here an apparent contradiction on the level to which policies are linked. The **Scalability Policies are defined at Application Level** (it is the application that is being scaled) but **they are then applied at Container Level** (it is the container that includes the application that is being deployed or un-deployed. It will be the duty of the TOSCA parser within the MiCADO framework to apply the policy at the right level. To simplify this process, policies themselves contain a list of the levels to which they will be applied.

Another interesting aspect of this use case is the presence of a Cron Job that is automatically executed at pre-defined times. At the moment being, although such scenario has not been explicitly analysed in COLA, there are two possible solutions.

- One, proposed in this Deliverable, is to define execution policies that specify the timing of the execution of a component of an application. The information of the execution policy will then be acted upon by an appropriate component of the MiCADO framework.
- A second solution is to modify the Application Topology Template adding one node with a timer application that will execute the Cron Job at fixed intervals.

We suggest the first solution as it does not require modification of the topology of the Application Description Template and it is a more generic solution to the problem.

Application Description Skeleton

The AFA Web Application is governed by an application-level scalability policy (P1.1) that will create and launch a new container with a web-application and a word-press instance when a certain number of connection requests are reached. The service that provides the information on the number of connections and its namespace is defined in the specific parameters of the policy. This horizontal scalability policy is defined at Application Level but it is enforced at Resource Level by deploying or un-deploying Docker [13] containers with the required software stack. At Resource Level, there is a Deployment Policy (P1.5) that will select the Resources on which the Containers are executed so that the list of required external services can be reached.

The Caching Service is governed by slighter more complex set of policies, at application level, there are three main policies. The Authorization Policy (P1.2) defines that an external service will be used for Authorization. This policy indirectly relates to the Deployment Policy at Resource Level (P1.6) as the Authorization Service must be reachable. The Caching Service is the component that is most likely to reap the maximum benefit from the MiCADO framework. Its scalability is a fundamental concern which is now tackled by deploying the component on a multi-core auto-scale AWS instance [14]. In COLA, the Caching Service will be governed by an Application-Level Horizontal Scalability Policy (P1.3) that will determine the amount of instances being deployed depending on the expected completion time of the query executions. This policy is similar to the Scaling Policy of Use Case 2 discussed in Section 0. At Application-Level, we also define an Execution Policy (P1.4) that defines the time at which the application must start in a similar fashion to that of a Cron Job. The Resource-Level of the Caching Service also defines two additional policies: Deployment Policy (P1.6) that will determine the minimum resource requirements for the deployment of containers and Deployment Policy (P1.7) that would be similar to Deployment Policy (P.1.2) and will define that a certain list of services must be reachable from the Resource. The overall topology of the Application Description Template is depicted in Figure 11.

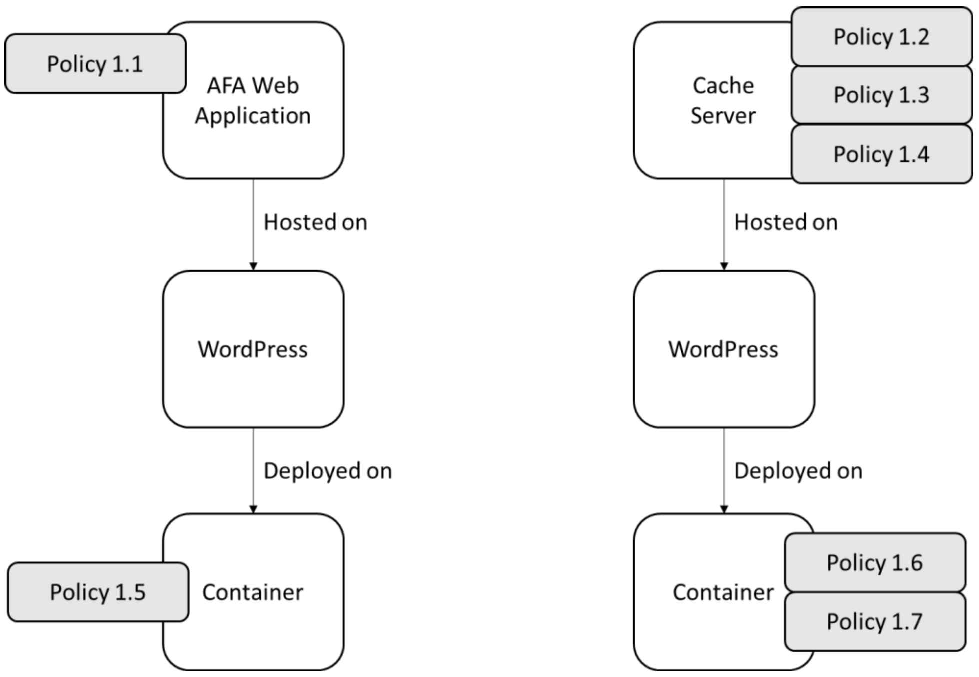
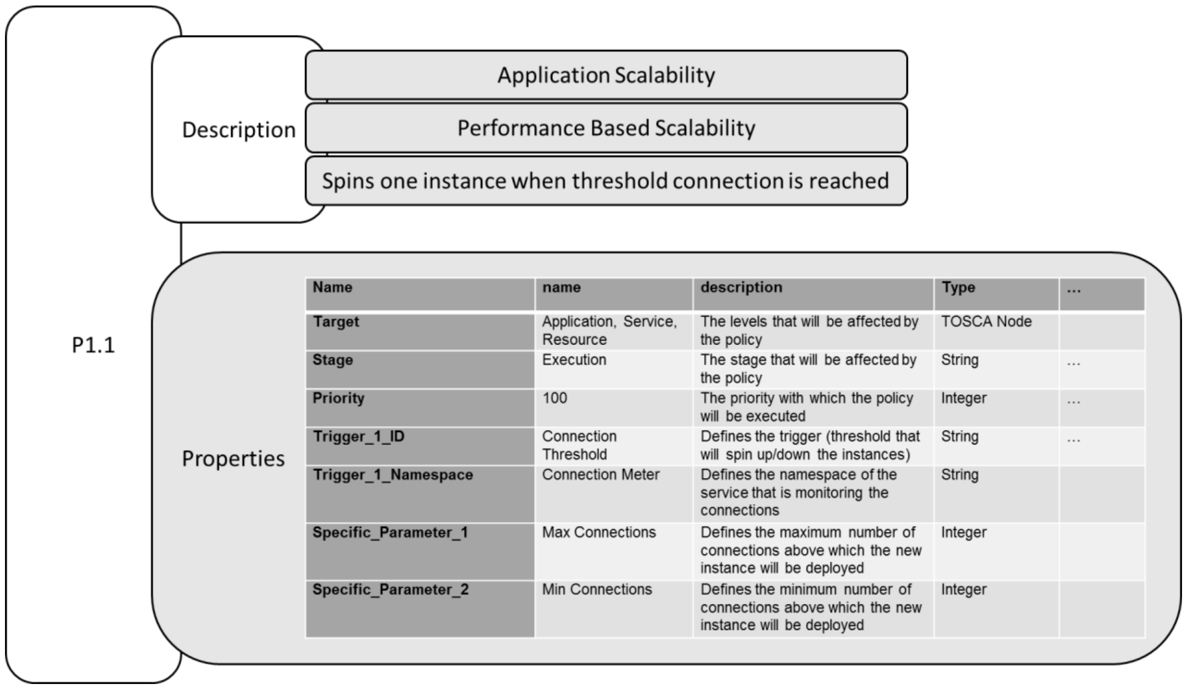


Figure 11, Use Case 1, Application Description Template

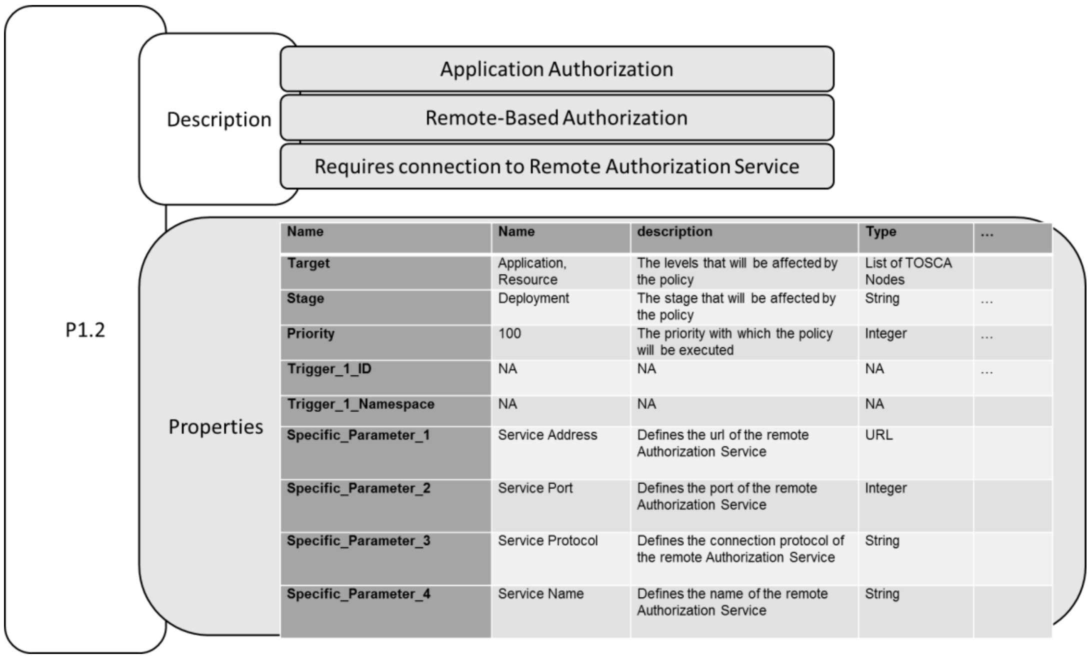
The policy that governs the horizontal scalability of the AFA Web Application (P1.1) is defined in the COLA Abstract Policy Hierarchy as a “Performance Based Scalability” policy and it is detailed by the parameters of Figure 12.


P1.1
Description
Application Scalability
Performance Based Scalability
Spins one instance when threshold connection is reached
Properties

Name	name	description	Type	...
Target	Application, Service, Resource	The levels that will be affected by the policy	TOSCA Node	
Stage	Execution	The stage that will be affected by the policy	String	...
Priority	100	The priority with which the policy will be executed	Integer	...
Trigger_1_ID	Connection Threshold	Defines the trigger (threshold that will spin up/down the instances)	String	...
Trigger_1_Namespace	Connection Meter	Defines the namespace of the service that is monitoring the connections	String	
Specific_Parameter_1	Max Connections	Defines the maximum number of connections above which the new instance will be deployed	Integer	
Specific_Parameter_2	Min Connections	Defines the minimum number of connections above which the new instance will be deployed	Integer	

Figure 12, Parameters of Policy 1.1 of Use Case 1

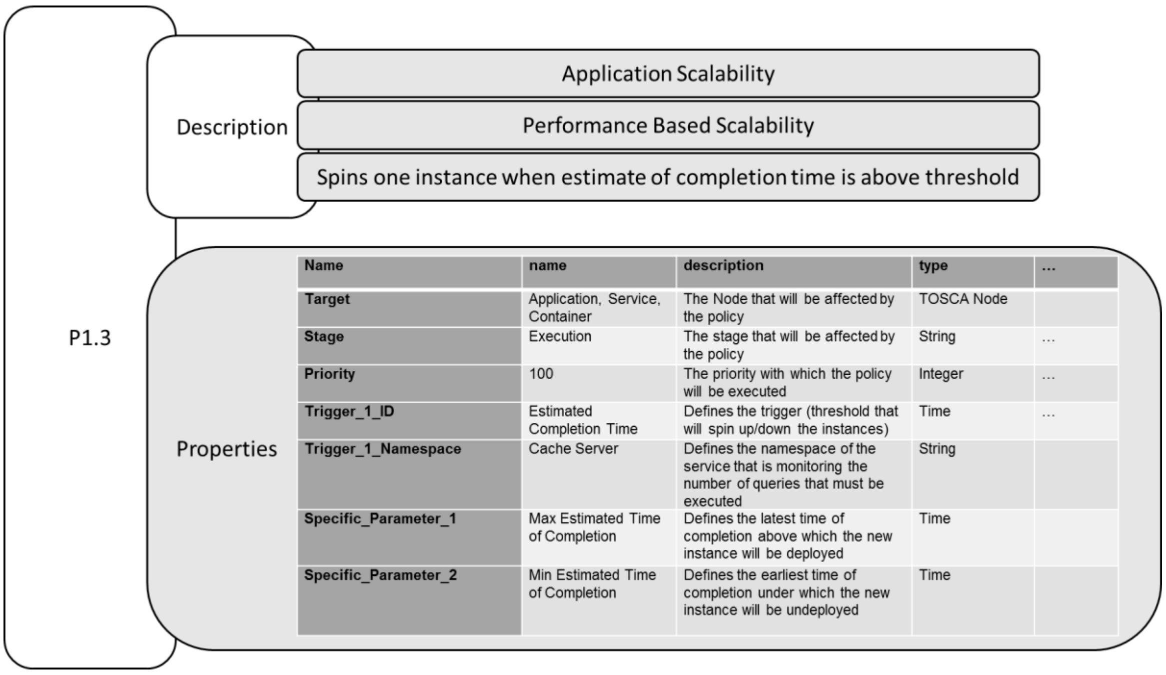
The policy that describes the Authorization of the Cache Server (P1.2) is defined in the COLA Abstract Policy Hierarchy as an “Authorization” policy and it is detailed by the parameters of Figure 13.


P1.2
Description
Application Authorization
Remote-Based Authorization
Requires connection to Remote Authorization Service
Properties

Name	Name	description	Type	...
Target	Application, Resource	The levels that will be affected by the policy	List of TOSCA Nodes	
Stage	Deployment	The stage that will be affected by the policy	String	...
Priority	100	The priority with which the policy will be executed	Integer	...
Trigger_1_ID	NA	NA	NA	...
Trigger_1_Namespace	NA	NA	NA	
Specific_Parameter_1	Service Address	Defines the url of the remote Authorization Service	URL	
Specific_Parameter_2	Service Port	Defines the port of the remote Authorization Service	Integer	
Specific_Parameter_3	Service Protocol	Defines the connection protocol of the remote Authorization Service	String	
Specific_Parameter_4	Service Name	Defines the name of the remote Authorization Service	String	

Figure 13, Parameters of Policy 1.2 of Use Case 1

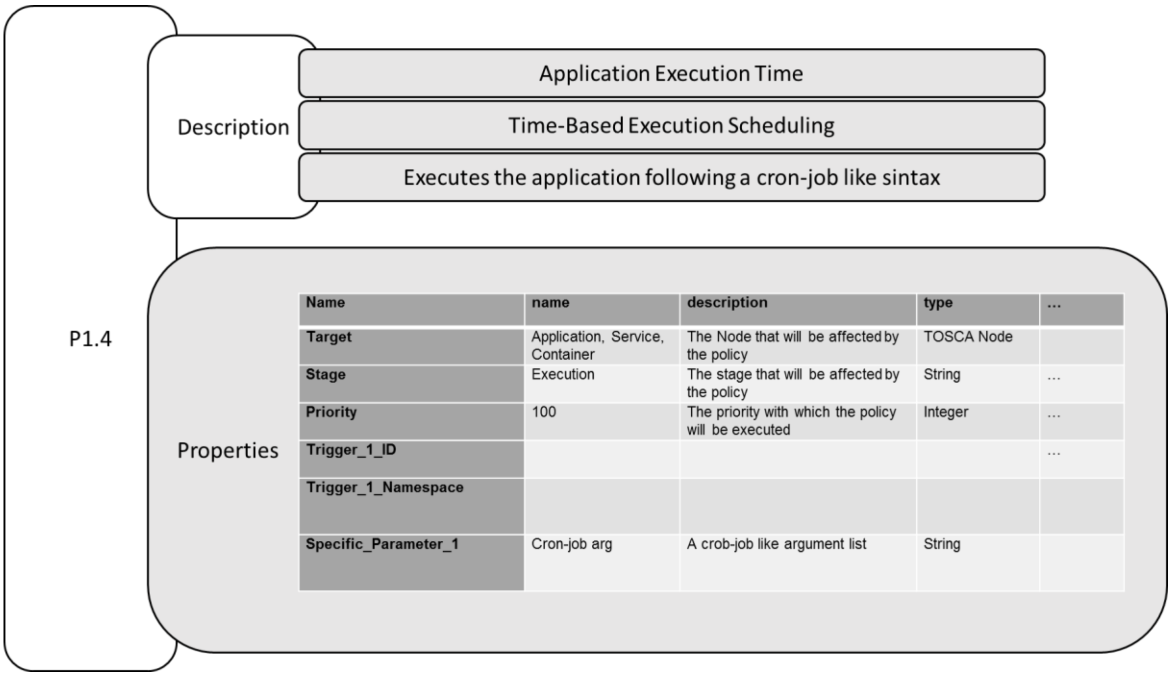
The policy that governs the horizontal scalability of the Caching Service (P1.3) is defined in the Abstract Policy Hierarchy as a “Performance Based Scalability” policy and it is detailed by the parameters of Figure 14.


P1.3
Description
Application Scalability
Performance Based Scalability
Spins one instance when estimate of completion time is above threshold
Properties

Name	name	description	type	...
Target	Application, Service, Container	The Node that will be affected by the policy	TOSCA Node	
Stage	Execution	The stage that will be affected by the policy	String	...
Priority	100	The priority with which the policy will be executed	Integer	...
Trigger_1_ID	Estimated Completion Time	Defines the trigger (threshold that will spin up/down the instances)	Time	...
Trigger_1_Namespace	Cache Server	Defines the namespace of the service that is monitoring the number of queries that must be executed	String	
Specific_Parameter_1	Max Estimated Time of Completion	Defines the latest time of completion above which the new instance will be deployed	Time	
Specific_Parameter_2	Min Estimated Time of Completion	Defines the earliest time of completion under which the new instance will be undeployed	Time	

Figure 14, Parameters of Policy 1.3 of Use Case 1

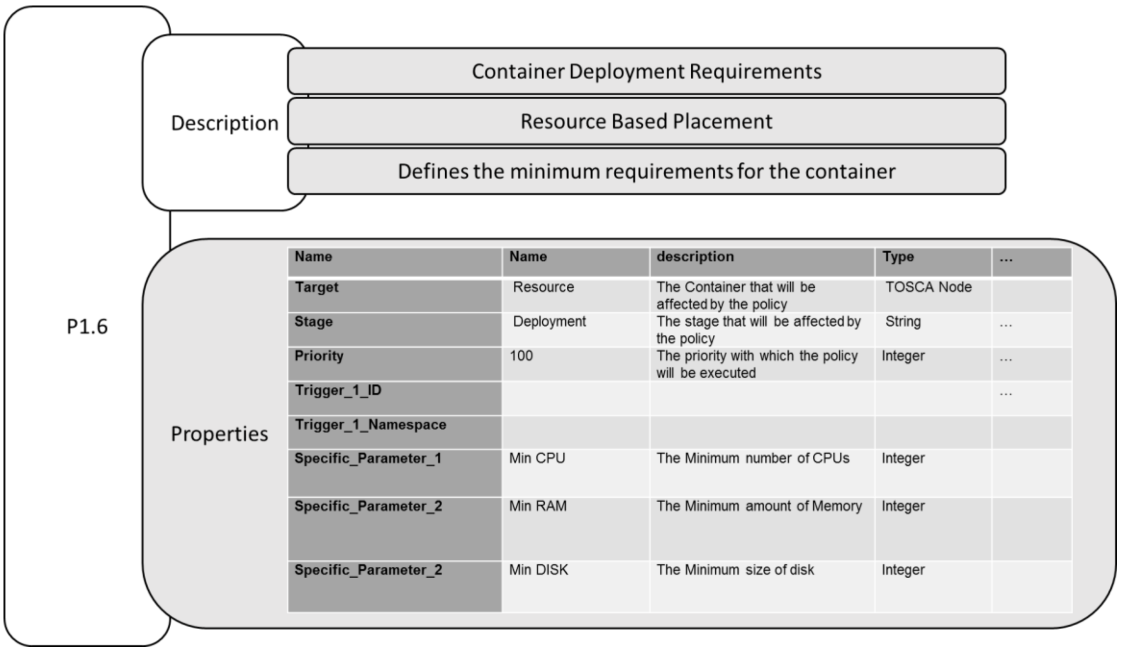
The policy, that governs the execution time, has not been added yet to the Abstract Policy Hierarchy, a proposal of its parameters are described in Figure 15.


P1.4
Description
Application Execution Time
Time-Based Execution Scheduling
Executes the application following a cron-job like sintax
Properties

Name	name	description	type	...
Target	Application, Service, Container	The Node that will be affected by the policy	TOSCA Node	
Stage	Execution	The stage that will be affected by the policy	String	...
Priority	100	The priority with which the policy will be executed	Integer	...
Trigger_1_ID				...
Trigger_1_Namespace				
Specific_Parameter_1	Cron-job arg	A crob-job like argument list	String	

Figure 15, Parameters of Policy 1.4 of Use Case 1

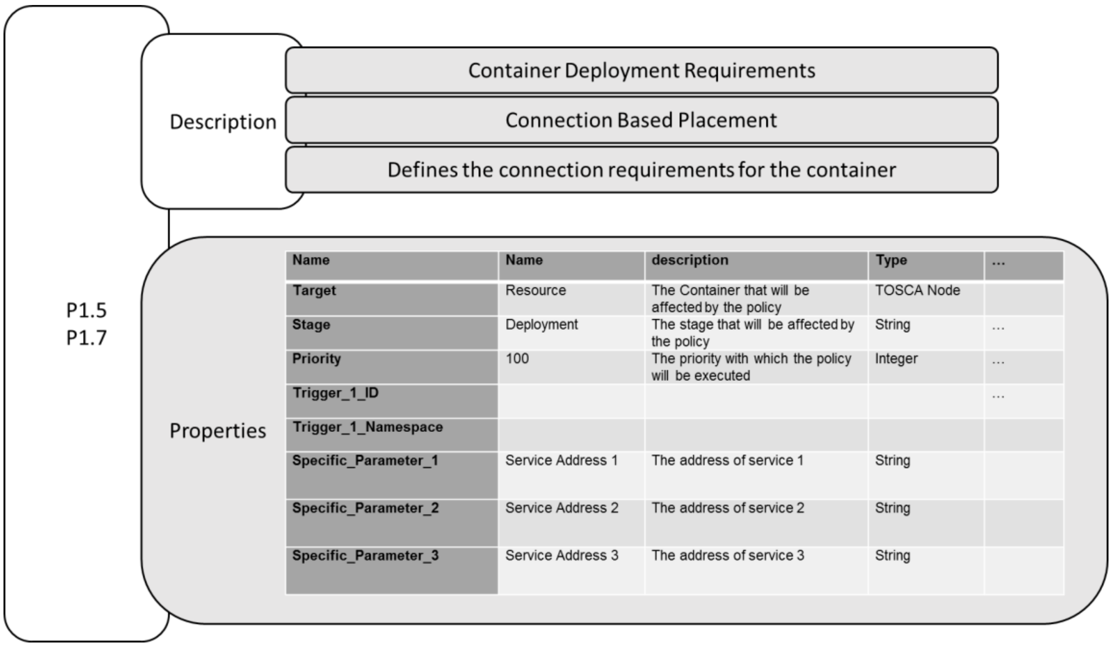
The policy (P1.6) that governs the deployment requirements of the container hosting the Caching Service is defined in the Abstract Policy Hierarchy as a “Resource Based Placement” policy and it is detailed by the parameters of Figure 16.


P1.6
Description
Container Deployment Requirements
Resource Based Placement
Defines the minimum requirements for the container
Properties

Name	Name	description	Type	...
Target	Resource	The Container that will be affected by the policy	TOSCA Node	
Stage	Deployment	The stage that will be affected by the policy	String	...
Priority	100	The priority with which the policy will be executed	Integer	...
Trigger_1_ID				...
Trigger_1_Namespace				
Specific_Parameter_1	Min CPU	The Minimum number of CPUs	Integer	
Specific_Parameter_2	Min RAM	The Minimum amount of Memory	Integer	
Specific_Parameter_2	Min DISK	The Minimum size of disk	Integer	

Figure 16, Parameters of Policy 1.6 of Use Case 1

The policies (P1.5 and P1.7) that govern the deployment requirements of the Containers hosting the AFA Web Application and the Cache Server are defined in the Abstract Policy Hierarchy as a “Resource Based Placement” policy and they are detailed by the parameters of Figure 17.


P1.5
P1.7
Description
Container Deployment Requirements
Connection Based Placement
Defines the connection requirements for the container
Properties

Name	Name	description	Type	...
Target	Resource	The Container that will be affected by the policy	TOSCA Node	
Stage	Deployment	The stage that will be affected by the policy	String	...
Priority	100	The priority with which the policy will be executed	Integer	...
Trigger_1_ID				...
Trigger_1_Namespace				
Specific_Parameter_1	Service Address 1	The address of service 1	String	
Specific_Parameter_2	Service Address 2	The address of service 2	String	
Specific_Parameter_3	Service Address 3	The address of service 3	String	

Figure 17, Parameters of Policy 1.5 and 1.7 of Use Case 1

8.2 Use Case 2: Bursting onto the Cloud from SakerGrid – Brunel University and Saker Solutions

Use Case 2 deals with a simulation platform that has been developed by Saker Solutions and Brunel University to reduce the amount of time required executing simulations. The solution is to extend to the Cloud the Distributed Computing Environment, now based on a desktop grid (SakerGrid) which achieves a near linear improvement in response performance.

The application follows a standard job-submission system architecture and consists of three main components:

- the User Interface
- the Worker Node,
- the Job Manager, and,
- the Simulation Database

Porting Use Case 2 into the MiCADO framework requires deployment of the Job Manager and the Worker Nodes to support two main types of simulations: the REPAST open source agent-based simulator[15], and the Evacuation Simulation scenario that is developed by Saker Solutions using the proprietary Flexsim discrete event simulation package [16].
Use Case 2 is introduced in pages 18 to 26 of deliverable D8.1. This initial description was updated during interactions between WP5 leader and Brunel University. The overall architecture of Use Case 2 is presented in Figure 18.

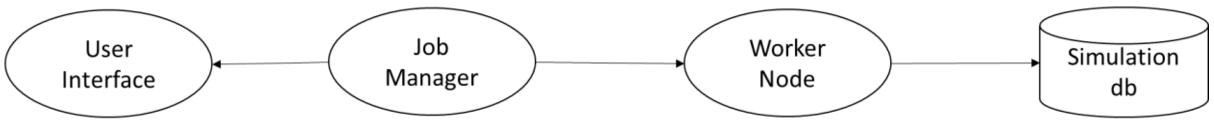


Figure 18, Architecture Overview of Use Case 2

This use case can be decomposed into two separate cases depending on the type of simulation being executed. They both share some common traits however, at the moment, the evacuation simulation jobs will require to be executed in a Windows environment (Windows Virtual Machine), while the REPAST jobs can be executed on a Linux machine. The architectural overviews of these two sub-cases are depicted in Figure 19 and Figure 20.

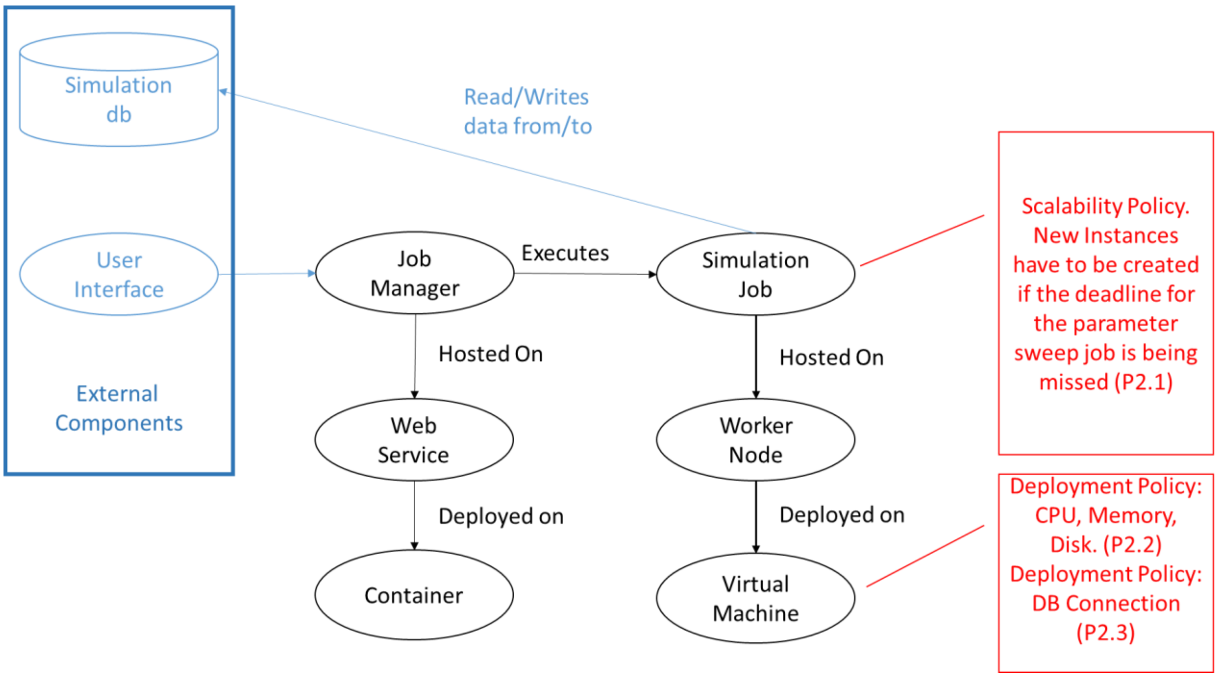

Simulation db
User Interface
External Components
Read/Writes data from/to
Job Manager
Executes
Simulation Job
Hosted On
Web Service
Deployed on
Container
Hosted On
Worker Node
Deployed on
Virtual Machine
Scalability Policy. New Instances have to be created if the deadline for the parameter sweep job is being missed (P2.1)
Deployment Policy: CPU, Memory, Disk. (P2.2)
Deployment Policy: DB Connection (P2.3)

Figure 19, Architectural Overview of the Implementation of Use Case 2 - Evacuation Simulation

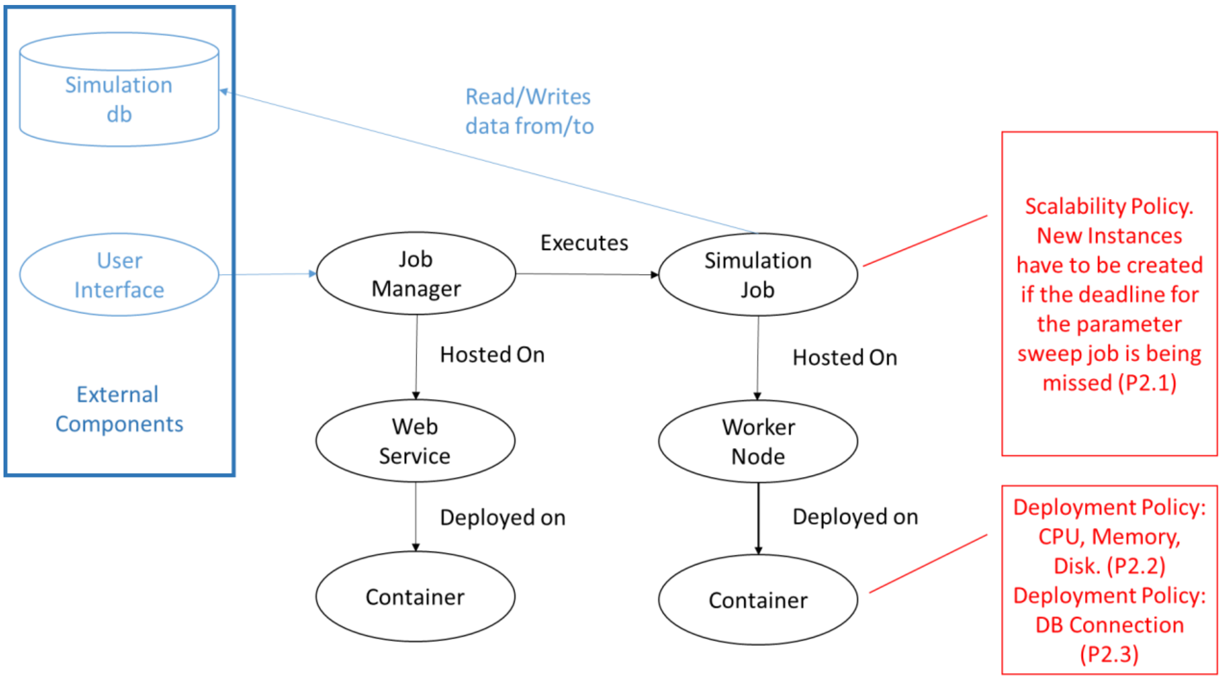

Simulation db
User Interface
External Components
Read/Writes data from/to
Job Manager
Executes
Simulation Job
Hosted On
Web Service
Deployed on
Container
Hosted On
Worker Node
Deployed on
Container
Scalability Policy. New Instances have to be created if the deadline for the parameter sweep job is being missed (P2.1)
Deployment Policy: CPU, Memory, Disk. (P2.2)
Deployment Policy: DB Connection (P2.3)

Figure 20, Architectural Overview of the Implementation of Use Case 2– Repast Simulation

The Job Manager does not require any specific policy either at application, service or deployment level but the worker node will be governed by a performance-based scalability policy (new instances will be deployed if the deadline for the completion of the simulation is likely to be missed), and a resource-based placement policy for the container will govern the placement of the worker nodes. The policies that govern the deployment and scaling during execution of the various components are summarized in Table 5, Policies of Use Case 2.

Architectural Level	Job Manager	Simulation Job
Application	**No specific Policy at this level**	**Scalability Policy (P2.1)** The Entire application is constrained by a high priority scaling policy whereby the user specifies an overall deadline and an estimate of the duration of each job and MiCADO will deploy new instances of the Workers to meet the deadline
Service	**No specific Policy at this level**	**No specific Policy at this level**
Resource	**No specific Policy at this level**	**Deployment Policy (P2.2)** It dictates that the container must have at least the following characteristics (ex CPU, RAM, DISK) **Deployment Policy (P2.3)** It dictates that the container must be capable of reaching the set of addresses (with the related ports and protocols) needed by all the external components: mainly the external database

Table 5, Policies of Use Case 2

The evacuation simulation requires to be deployed on a completely self-contained and isolated infrastructure for security reasons but this requirement cannot be expressed as an application level policy as it is an infrastructure-level requirement.

Application Description Skeleton

The topology branch that models the Job Manager is not governed by any specific policy while the branch that models the worker nodes is governed by application and container-level policies. The overall topology of the Application Description Template for Use Case two are described in Figure 21 and Figure 22.

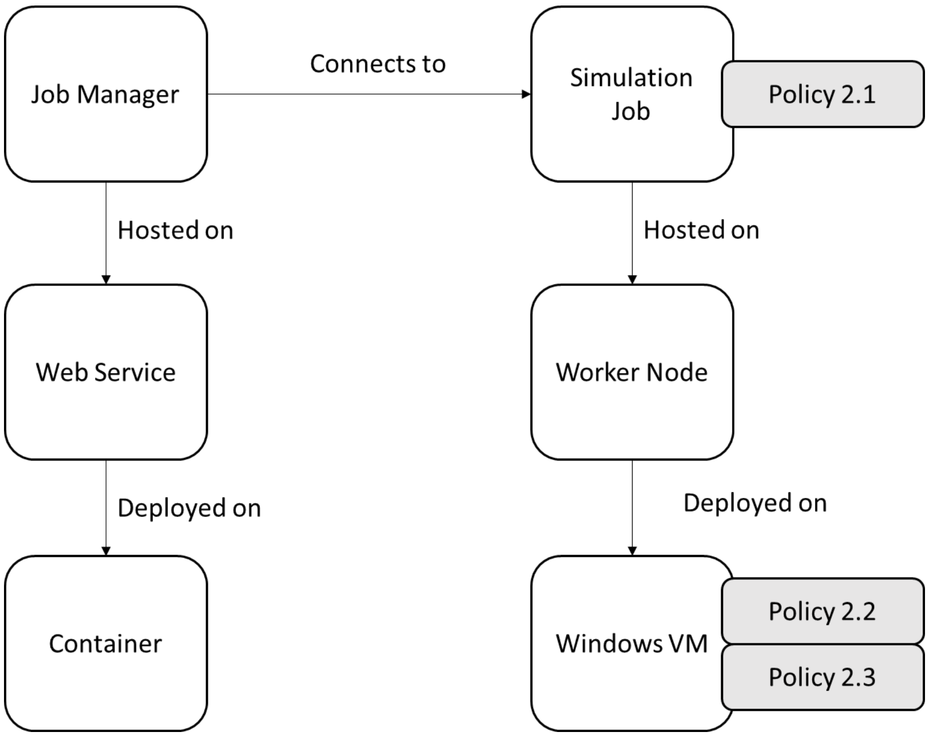


Figure 21, Use Case 2, Application Description Template for the Evacuation Simulation

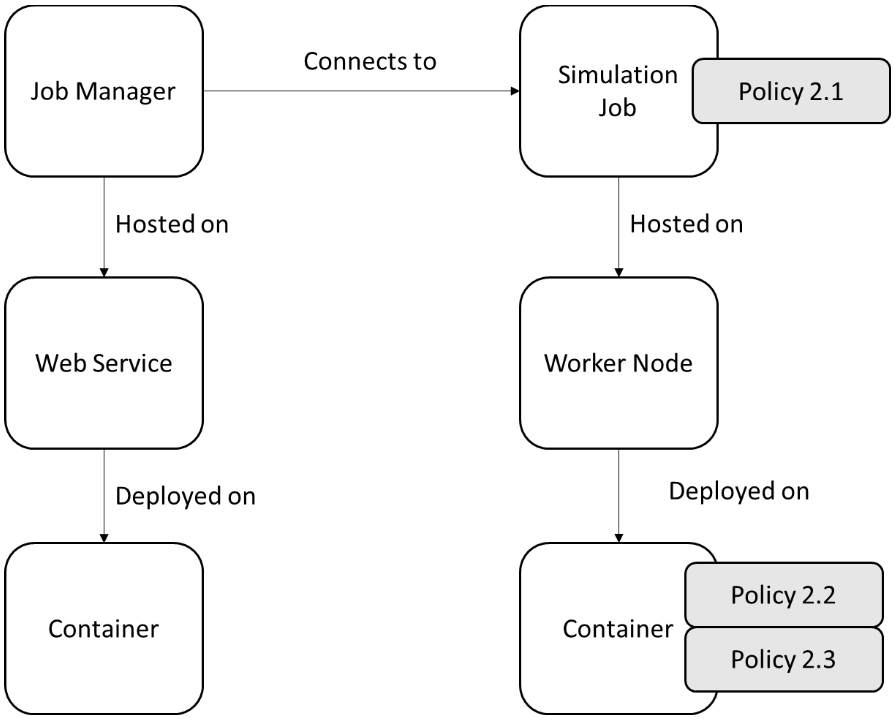
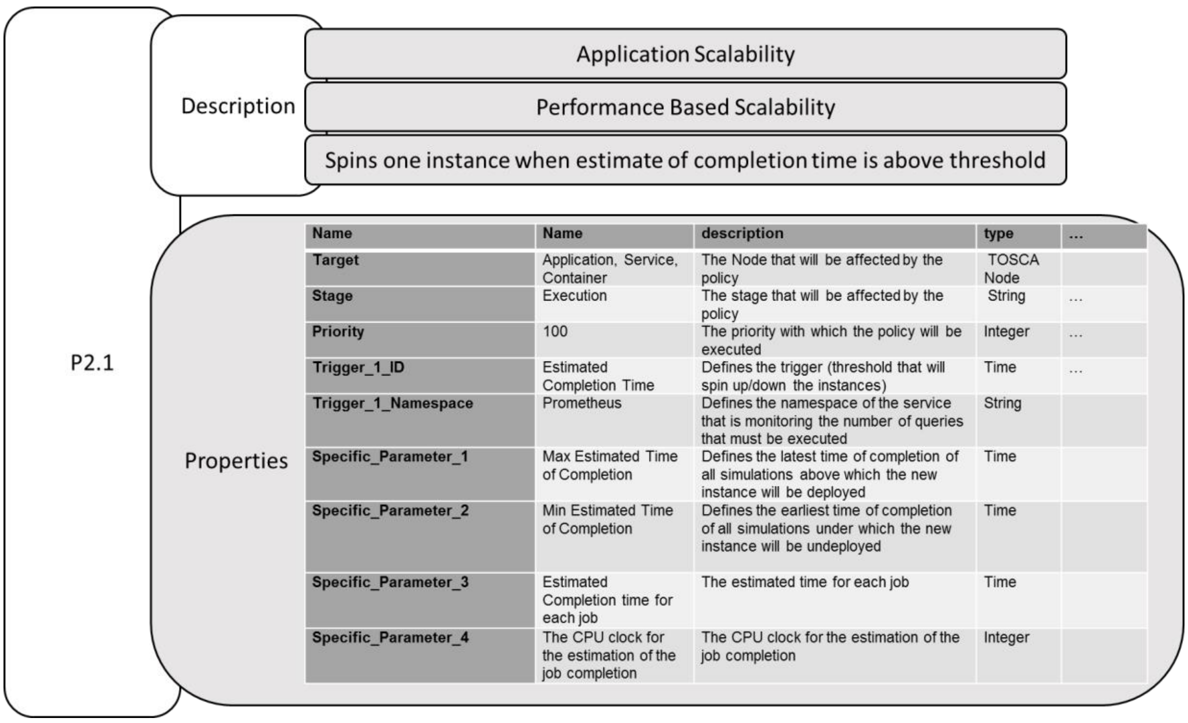


Figure 22, Use Case 2, Application Description Template for the REPAST simulation

The policy that governs the horizontal scalability of the Simulation Job is defined in the abstract policy hierarchy as a “Performance Based Scalability” policy and it is detailed by the parameters of Figure 23. Please note that horizontal Scalability Policies are the same for both the Repast and Evacuation Simulation scenarios. The Deployment Policy dictating the minimum requirements and the connection requirements are the same as of Use Case 1:

Policies P1.5, P1.6 and P1.7


P2.1
Description
Application Scalability
Performance Based Scalability
Spins one instance when estimate of completion time is above threshold
Properties

Name	Name	description	type	...
Target	Application, Service, Container	The Node that will be affected by the policy	TOSCA Node	
Stage	Execution	The stage that will be affected by the policy	String	...
Priority	100	The priority with which the policy will be executed	Integer	...
Trigger_1_ID	Estimated Completion Time	Defines the trigger (threshold that will spin up/down the instances)	Time	...
Trigger_1_Namespace	Prometheus	Defines the namespace of the service that is monitoring the number of queries that must be executed	String	
Specific_Parameter_1	Max Estimated Time of Completion	Defines the latest time of completion of all simulations above which the new instance will be deployed	Time	
Specific_Parameter_2	Min Estimated Time of Completion	Defines the earliest time of completion of all simulations under which the new instance will be undeployed	Time	
Specific_Parameter_3	Estimated Completion time for each job	The estimated time for each job	Time	
Specific_Parameter_4	The CPU clock for the estimation of the job completion	The CPU clock for the estimation of the job completion	Integer	

Figure 23, Parameters of Policy 2.1 of Use Case 2

8.3 Use Case 3 – Social media data analytics for public sector organisations Inycom and SARGA

Use Case 3 deals with a modified version of the Eccobuzz [17] platform with enriched functionalities called Magician [18] (see D8.1 for details). Eccobuzz allows its users to monitor internet resources for specified information and it provides structured results by the means of reports that are received automatically by email. Use Case 3 focuses on the deployment of the Motor Engine of Eccobuzz and its Configuration Database (based on MongoDB [19]). The deployment in MiCADO aims at achieving greater scalability.

Use Case 3 is introduced in pages 26 to 38 of deliverable D8.1. This initial description was updated during interactions between WP5 leader and Inycom. The overall architecture of Use Case 3 is drafted in Figure 24.

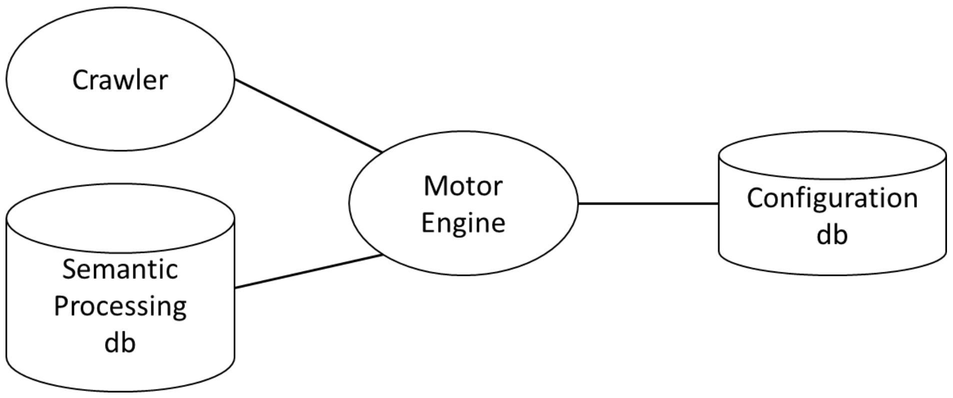


Figure 24, Architecture Overview of Use Case 3

The main functionalities are provided by the **Motor Engine** which uses a **Configuration Database** based on MongoDB. The Motor Engine uses two external services: a **Crawler** and a **Semantic Processing Database** based on a SOLr [20] database where the Motor stores its results. One single container can be used to contain both the Motor Engine and the Configuration Database on MiCADO. The architectural overview of the implementation of Use Case 3 is described in Figure 25.

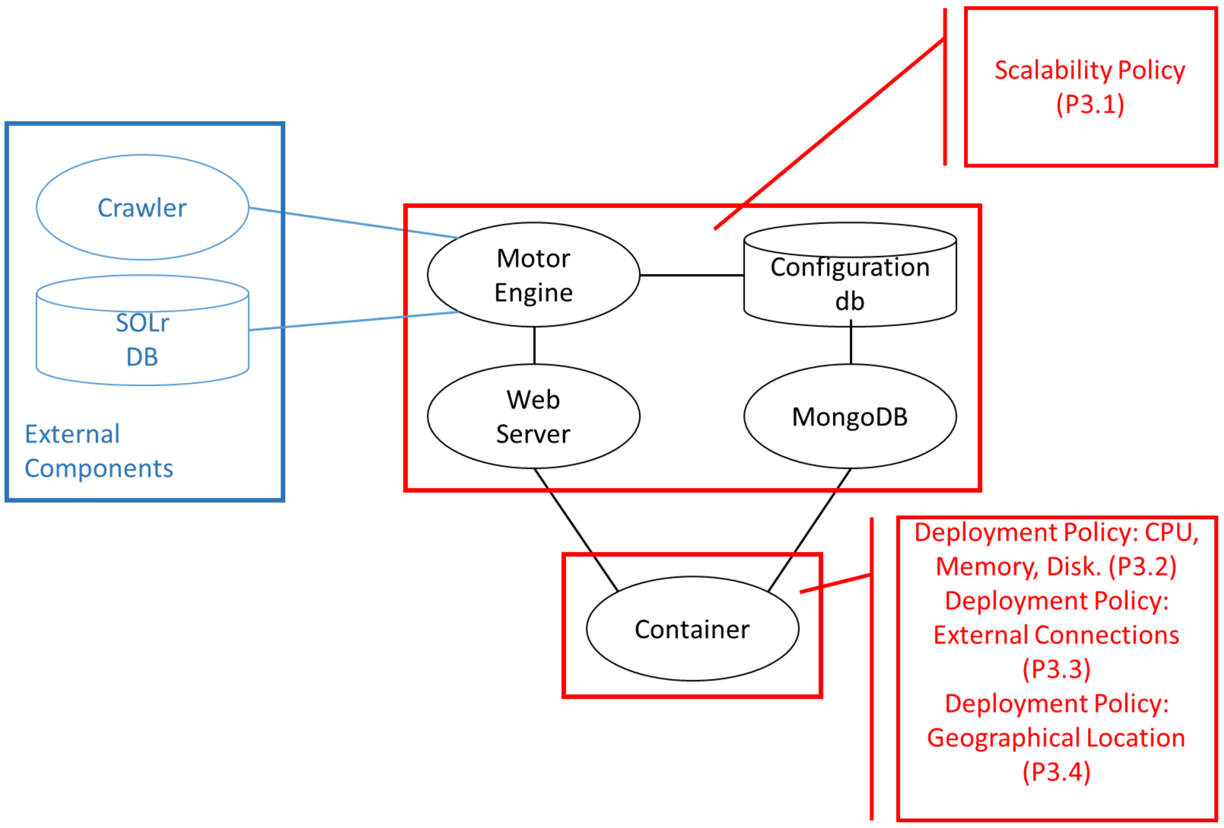

Scalability Policy (P3.1)
Crawler
SOLr DB
External Components
Motor Engine
Configuration db
Web Server
MongoDB
Container
Deployment Policy: CPU, Memory, Disk. (P3.2)
Deployment Policy: External Connections (P3.3)
Deployment Policy: Geographical Location (P3.4)

Figure 25, Architectural Overview of the Implementation of Use Case 3

The policies that govern the deployment and scaling of the various components during execution are in Table 6.

Architectural Level	**Motor Engine**	**Configuration Database**
Application	***Scalability Policy (P3.1)*** It defines the deployment of a new instance under a consumption requirement based on CPU usage.	
Service	**No specific Policy at this level**	
Resource	**Deployment Policy (P3.2)** It dictates that the container must have at least the following characteristics (ex CPU, RAM, DISK) **Deployment Policy (P3.3)** It dictates that the container must be capable of reaching the set of addresses (with the related ports and protocols) needed by all the external components. **Deployment Policy (P3.4)** It dictates that the container must be physically located in the European Union	

Table 6, Policies of Use Case 3

Application Description Skeleton

As detailed in Table 6, Policies of Use Case 3, both applications and the services deployed on the platform are modelled within the same topology which is governed by four separate policies.

An Application-Level Scalability Policy (P3.1) will define the conditions under which new instances will be deployed. At Resource level, there are three policies. A Deployment Policy (P3.4) related to privacy concerns will restrict the geographical location where the container can be deployed. Two additional Deployment Policies (P3.2 and P3.3) will dictate the capabilities (CPU, RAM, and DISK) of the resources and the services that can be reached. Policy 3.2 is the same of Policy P2.2 of Use Case 2 and Policy P3.4 is the same of Policy P2.3 of Use Case 2. The overall topology of the Application Description Template is described in Figure 26.

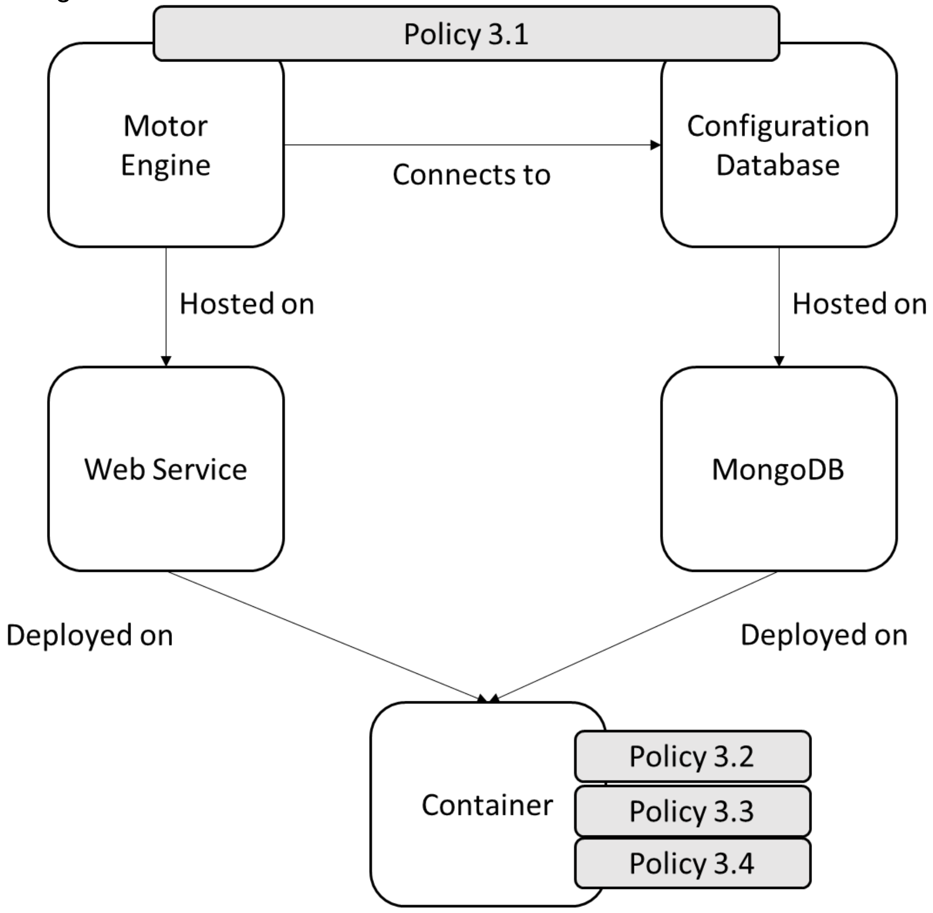
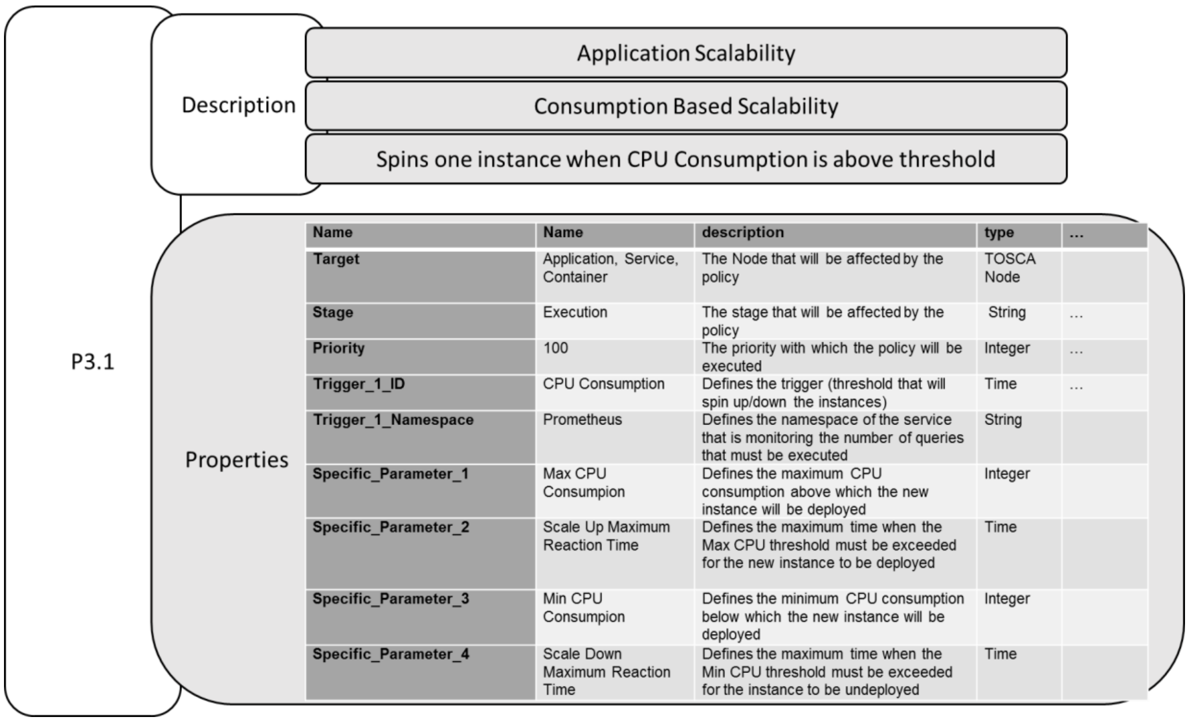


Figure 26, Use Case 3, Application Description Template

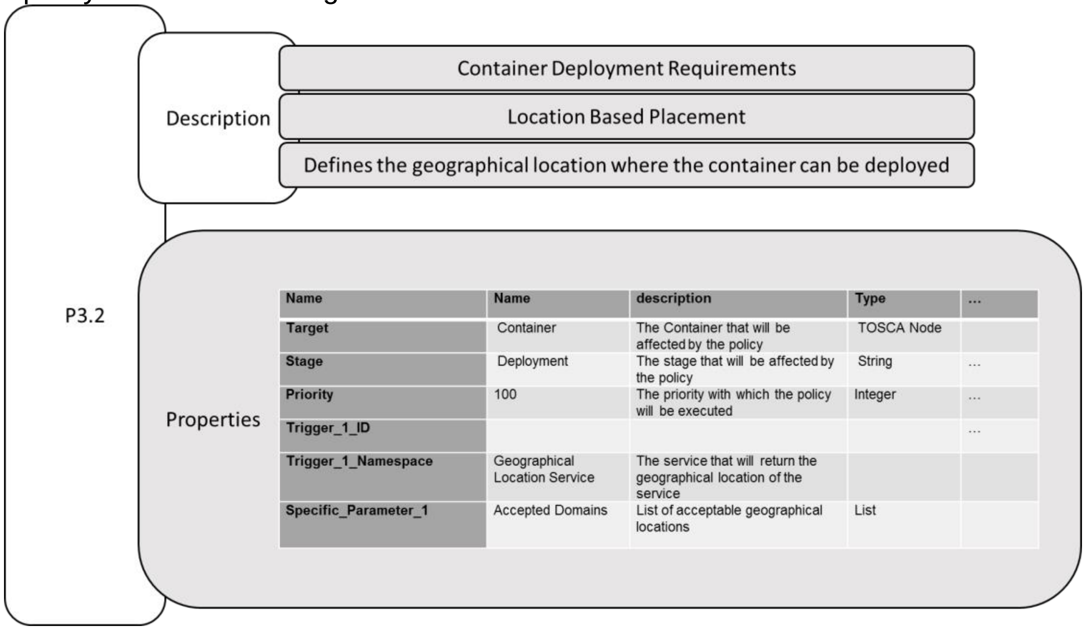
The policy that governs the horizontal scalability of the Motor Engine and Configuration Database is defined in the abstract policy hierarchy as a “Consumption Based Scalability” policy and it is detailed in Figure 27.


P3.1
Description
Application Scalability
Consumption Based Scalability
Spins one instance when CPU Consumption is above threshold
Properties

Name	Name	description	type	...
Target	Application, Service, Container	The Node that will be affected by the policy	TOSCA Node	
Stage	Execution	The stage that will be affected by the policy	String	...
Priority	100	The priority with which the policy will be executed	Integer	...
Trigger_1_ID	CPU Consumption	Defines the trigger (threshold that will spin up/down the instances)	Time	...
Trigger_1_Namespace	Prometheus	Defines the namespace of the service that is monitoring the number of queries that must be executed	String	
Specific_Parameter_1	Max CPU Consumption	Defines the maximum CPU consumption above which the new instance will be deployed	Integer	
Specific_Parameter_2	Scale Up Maximum Reaction Time	Defines the maximum time when the Max CPU threshold must be exceeded for the new instance to be deployed	Time	
Specific_Parameter_3	Min CPU Consumption	Defines the minimum CPU consumption below which the new instance will be deployed	Integer	
Specific_Parameter_4	Scale Down Maximum Reaction Time	Defines the maximum time when the Min CPU threshold must be exceeded for the instance to be undeployed	Time	

Figure 27, Parameters of Policy 3.1 of Use Case 3

The policy that governs the acceptable geographical locations requires the presence of a service (and his related namespace) capable of returning the geographical coordinates of the resources that will be used. COLA components that can implement such functionality can be the **Services R2 – Workload Node Verifier** and **Service R4 – Workload Node Selector** defined in page 29 of the Deliverable D7.1 – COLA Security Requirements. The policy defines a series of acceptable locations that will be matched against. The parameters of the policy are detailed in Figure 28


P3.2
Description
Container Deployment Requirements
Location Based Placement
Defines the geographical location where the container can be deployed
Properties

Name	Name	description	Type	...
Target	Container	The Container that will be affected by the policy	TOSCA Node	
Stage	Deployment	The stage that will be affected by the policy	String	...
Priority	100	The priority with which the policy will be executed	Integer	...
Trigger_1_ID				...
Trigger_1_Namespace	Geographical Location Service	The service that will return the geographical location of the service		
Specific_Parameter_1	Accepted Domains	List of acceptable geographical locations	List	

Figure 28, Parameters of Policy 3.3 of Use Case 3

9. Implementation of the Application Description Templates.

As introduced in Sections 5 and Section 7, the Application Description Templates are composed, edited and validated using a variety of tools. This section gives some details on which tools are currently used in each of these steps.

Application Description Template Repository: At the moment being, no Application Description Repository has been implemented and GitHub is being used as a temporary solution: https://github.com/COLAProject/ColaPolicies. It is also possible, if not likely, that GitHub will be adopted as a final solution for the Application Description Template Repositories.

Application Description Template Editors: Two main tools have been investigated to edit the Application Description Templates: Winery [21] and Alien4Cloud [22]. As Alien4Cloud defines its own description language that is not fully TOSCA-Compliant, its adoption in COLA has been abandoned. Until version 2.0.0, Winery (https://github.com/OpenTOSCA/winery) produced XML-based TOSCA code which became incompatible with the new TOSCA specification (based on YAML1.0); the Docker Version of Winery (https://github.com/OpenTOSCA/opentosca-docker), produces YAML1.1-based code but it is currently still under development. Figure 31 shows the Three Layered Application Description Template edited using Winery.

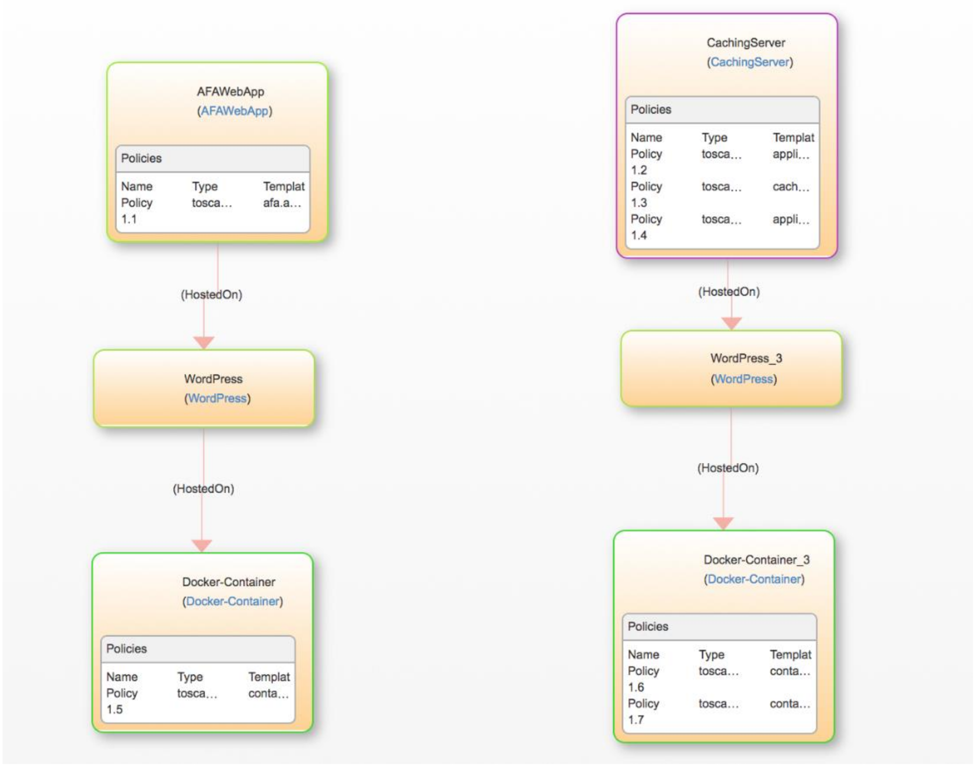
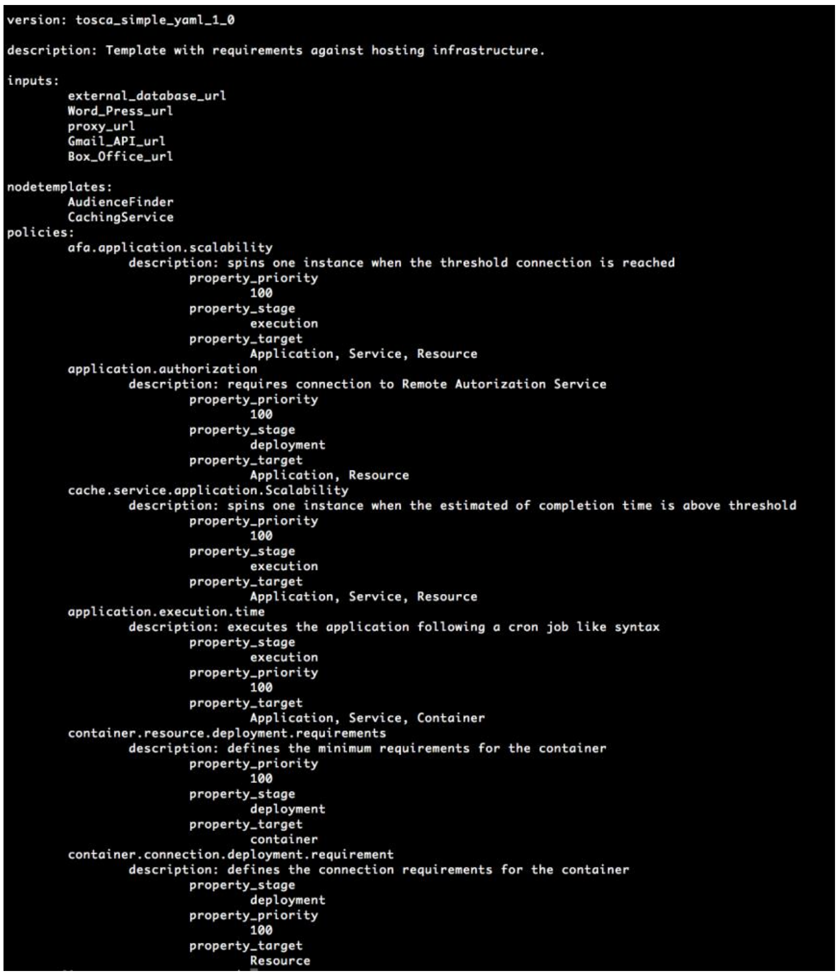


Figure 29, Three Layered Topology of Use Case 1 edited on Winery

Application Description Template Parses: The syntactical validity of the Application Description Templates is checked by parsing them with the OpenStack TOSCA Parser [4], that currently processes only YAML1.0-based code; this YAML-version difference requires manual code corrections in the Application Description Templates.

Implementation of the Application Description Templates: the full three-layered Application Description Templates offer flexibility and composability to the Application Developers allowing also for the definition of expressive policies at each of its layers. However its structure poses conceptual challenges when it is used at the Infrastructure Level. The MiCADO components of the COLA infrastructure, will use the Application Description Templates to select the best Container or Virtual machine for its deployment and will take appropriate actions to enact its policies, this is complicated when the policies are spread over the three layers of the Application Description Templates. To facilitate the selection of the optimal Container/Virtual machine and the enforcement of the related policies, the full topology of the Three Layered Application Description Template is reduced to a “Compressed Topology”, containing a single node per Container/Virtual machine detailing the list of needed services and the policies (Annexe A and B contain the listing of the Three Layered and Compressed Topology Application Description Template for Use Case 1). Figure 30 shows the parser output of the compressed Topology Application Description Template for Use Case 1.



```
version: tosca_simple_yaml_1_0

description: Template with requirements against hosting infrastructure.

inputs:
        external_database_url
        Word_Press_url
        proxy_url
        Gmail_API_url
        Box_Office_url

nodetemplates:
        AudienceFinder
        CachingService
policies:
        afa.application.scalability
                description: spins one instance when the threshold connection is reached
                        property_priority
                                100
                        property_stage
                                execution
                        property_target
                                Application, Service, Resource
        application.authorization
                description: requires connection to Remote Autorization Service
                        property_priority
                                100
                        property_stage
                                deployment
                        property_target
                                Application, Resource
        cache.service.application.Scalability
                description: spins one instance when the estimated of completion time is above threshold
                        property_priority
                                100
                        property_stage
                                execution
                        property_target
                                Application, Service, Resource
        application.execution.time
                description: executes the application following a cron job like syntax
                        property_stage
                                execution
                        property_priority
                                100
                        property_target
                                Application, Service, Container
        container.resource.deployment.requirements
                description: defines the minimum requirements for the container
                        property_priority
                                100
                        property_stage
                                deployment
                        property_target
                                container
        container.connection.deployment.requirement
                description: defines the connection requirements for the container
                        property_stage
                                deployment
                        property_priority
                                100
                        property_target
                                Resource
```

Figure 30, Short Parser Log of the Compressed Application Description Template for Use Case 1

Mapping of Application to Existing Containers: The mapping of existing containers to the requirements expressed in the Application Description Templates poses different challenges and trade-offs that are currently investigated. On one end, a minimalistic approach suggests to have a minimal set of Containers and Virtual Machines (possibly with just an Operating System installed) to minimize the complexity of their management and updates. This approach requires re-creating all the Service Layer every time and this has significant drawbacks both on the complexity of the Application Description Template - that will have to define installation scripts and data (Artefacts) for various Services - and on the execution time overhead (especially for rapidly-changing scalability reactions). On the other end, we can have all possible combinations of Services already available as images. This approach reduces the installation complexity to a minimum but there exists a risk of combinatorial explosions of available Container/Virtual machines images and the related complexity of maintaining and updating them.

The Implementations of the Three Layered and Compressed Topology Application Description Templates are also available in the GitHub repository under:
https://github.com/COLAProject/ColaPolicies/tree/master/use_case_1

10. Conclusions

The three Use Cases analysed in this document cover several relevant aspects of the policy extensions of the TOSCA Application Description Template defined and outlined in D5.2.

The three layered approach for the Application Description Template have highlighted several node components that can be re-used in further use cases, more specifically the nodes that describe generic services such as WordPress and MongoDB.

The description of the three COLA use cases with the Application Description Templates constitutes a proof of concept of its applicability and feasibility, particularly how to manage policies of these use cases. Figure 31 gives an overview of these policies. The policies initially proposed in D5.2 were expressive enough to cover all the use cases except Use Case 1.
It requires a new Execution Policy to schedule the execution of applications in the same fashion as a Cron Job. Such a policy was not envisaged in the D5.2. For the moment being, we can envisage a set of Execution Policies of which Scheduling could be a sub-type.

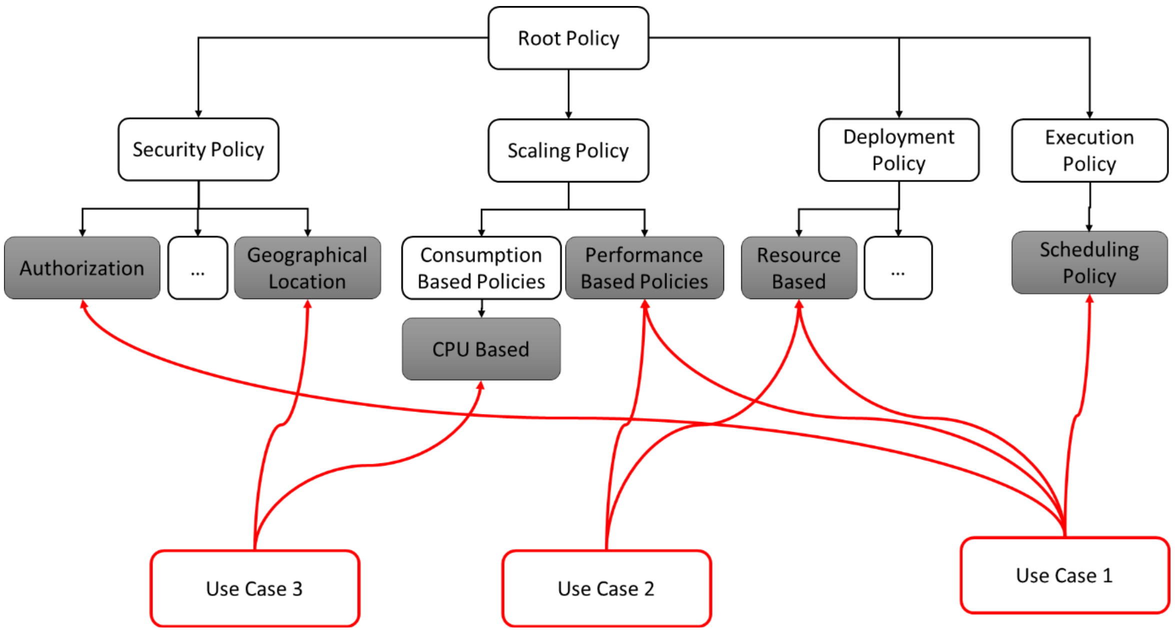


Figure 31, Various Policies Types used in the three Use Cases

11. References

[1] Tamas Kiss, Peter Kacsuk, Jozsef Kovacs, Botond Rakoczi, Akos Hajnal, Attila Farkas, Gregoire Gesmier, Gabor Terstyanszky: MiCADO –Microservice-based Cloud Application-level Dynamic Orchestrator, in *Future Generation Computing Systems,* https://doi.org/10.1016/j.future.2017.09.050.

[2] “MiCADO Developer Tutorials Online – Cloud Orchestration at the Level of Application.” [Online]. Available: http://project-cola.eu/micado-tutorials-online/. [Accessed: 28-Oct-2017].

[3] “TOSCA_overview.”

[4] “TOSCA-Parser - OpenStack.” [Online]. Available: https://wiki.openstack.org/wiki/TOSCA-Parser. [Accessed: 29-Oct-2017].

[5] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, “TOSCA: Portable Automated Deployment and Management of Cloud Applications.”

[6] “Topology and Orchestration Specification for Cloud Applications Version 1.0.” [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html. [Accessed: 30-Mar-2017].

[7] S. Khurana and A. G. Verma, “Comparison of Cloud Computing Service Models: SaaS, PaaS, IaaS,” vol. 4.

[8] “tosca-primer-v1.0.”

[9] “The Official YAML Web Site.” .

[10] “Welcome - Occopus.” [Online]. Available: http://occopus.lpds.sztaki.hu/. [Accessed: 30-Mar-2017].

[11] “Docker Swarm overview - Docker Documentation.” [Online]. Available: https://docs.docker.com/swarm/overview/. [Accessed: 30-Mar-2017].

[12] “TOSCA-spec-v1.0.”

[13] “Docker - Build, Ship, and Run Any App, Anywhere.” [Online]. Available: https://www.docker.com/. [Accessed: 30-Mar-2017].

[14] “AWS | Auto Scaling.” [Online]. Available: https://aws.amazon.com/autoscaling/. [Accessed: 02-Nov-2017].

[15] “Repast Suite Documentation.” [Online]. Available: https://repast.github.io/. [Accessed: 02-Nov-2017].

[16] “Simulation software for manufacturing, material handling, healthcare, etc. - FlexSim Simulation Software.” [Online]. Available: https://www.flexsim.com/. [Accessed: 02-Nov-2017].

[17] “EccoBuzz | Manage your buzz around.” [Online]. Available: http://www.eccobuzz.com/. [Accessed: 02-Nov-2017].

[18] “Soluciones Magician - Business Analytics | INYCOM.” [Online]. Available: http://www.inycom.es/soluciones-y-servicios-informatica/business-analytics/soluciones-magician. [Accessed: 02-Nov-2017].

[19] “MongoDB for GIANT Ideas | MongoDB.” [Online]. Available: https://www.mongodb.com/. [Accessed: 30-Apr-2017].

[20] “Apache Solr -.” [Online]. Available: http://lucene.apache.org/solr/. [Accessed: 01-Nov-2017].

[21] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann, and U. Breitenb, “Winery – A Modeling Tool for TOSCA-based Cloud Applications.”

[22] “ALIEN 4 Cloud.” [Online]. Available: https://alien4cloud.github.io/#alien-for-cloud-high-level-concept. [Accessed: 31-Oct-2017].

12. Annex A

This Annex contains the Three Layered Application Description Template (**audience-finder.yaml** and **topology.yaml**) of Use Case 1.

audience-finder.yaml defines the node types that are used in the Three-Layered Application Description Template of use case 1

```
tosca_definitions_version: tosca_simple_yaml_1_0

node_types:
  tosca.nodes.Container.Application.AFA:
    derived_from: tosca.nodes.Container.Application
    requirements:
      - proxy: tosca.capabilities.Endpoint
      - boxoffice-database: tosca.capabilities.Endpoint
      - wordpress-database: tosca.capabilities.Endpoint
      - wordpress:
        node: tosca.nodes.Container.Application.WordPress
        relationships: HostedOn

  tosca.nodes.Container.Application.CachingService:
    derived_from: tosca.nodes.Container.Application
    requirements:
      - wordpress-database: tosca.capabilities.Endpoint
      - boxoffice-database: tosca.capabilities.Endpoint
      - external-database: tosca.capabilities.Endpoint
      - gmail-api-url: tosca.capabilities.Endpoint
      - wordpress:
        node: tosca.nodes.Container.Application.WordPress
        relationships: HostedOn

  tosca.nodes.Container.Application.WordPress:
    derived_from: tosca.nodes.Container.Application
    requirements:
      - container:
        capability: tosca.capabilities.Endpoint
        node: tosca.nodes.Container.Application.Docker.Container
        relationships: HostedOn
    capabilities:
        wordpress:
            type: tosca.capabilities.Endpoint

  tosca.nodes.Container.Application.Docker.Container:
    derived_from: tosca.nodes.Container.Application.Docker
    capabilities:
      container:
          type: tosca.capabilities.Endpoint

policy_types:
  tosca.policies.Security.Authorization:
```

```
  derived_from: tosca.policies.Security
  description: authorization policy derived from security

tosca.policies.Scaling.Performance:
  derived_from: tosca.policies.Scaling
  description: Performance policy derived from scaling

tosca.policies.Deployment.Ressource:
  derived_from: tosca.policies.Deployment
  description: Ressource policy derived from Deployment

tosca.policies.Execution.Schedule:
  derived_from: tosca.policies.Execution
  description: Schedule policy derived from Execution
```

topology.yaml defines the Three-Layered Application Description Template of use case 1

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - audience_finder.yaml

repositories:
  docker_hub: https://registry.hub.docker.com/

description: Template with requirements against hosting infrastructure.

topology_template:
  inputs:
    proxy_url:
      type: string
      description: endpoint for the proxy
    external_database_url:
      type: string
      description: enpoint for the external database connected to the caching service
    Word_Press_url:
      type: string
      description: endpoint for the WordPress database
    Box_Office_url:
      type: string
      description: endpoint for the box office database
    Gmail_API_url:
      type: string
      description: endpoint to connect to the gmail API url

  node_templates:
    Container_for_Caching:
      type: tosca.nodes.Container.Application.Docker.Container
      capabilities:
        container:
          properties:
            protocol: tcp
            secure: false
            network_name: PRIVATE
            initiator: source
            url_path: { get_attributes: [ SELF, private_ip ]}

    WordPress_for_Caching:
      type: tosca.nodes.Container.Application.WordPress
      requirements:
        - container:
            node: Container_for_Caching
            relationship: my_connection
      capabilities:
        wordpress:
            properties:
            protocol: tcp
            secure: false
            network_name: PRIVATE
            initiator: source
            url_path: { get_attributes: [ SELF, private_ip ]}

    CachingService:
      type: tosca.nodes.Container.Application.CachingService
      requirements:
        - host:
            node_filter:
              capabilities:
                - host:
                   properties:
                    - num_cpus: { in_range: [ min, max ] }
                    - mem_size: { in_range: [ min, max ] }
                    - disk_size: { in_range: [ min, max ] }
        - wordpress-database:
            node_filter:
              capabilities:
                - host:
                    properties:
                       - url_path: { get_property: [ cacheservice.connection.deployment.requirement,
specific_parameter_1,  property_value ] }

        - boxoffice-database:
            node_filter:
              capabilities:
                - host:
                    properties:
                       - url_path: { get_property: [ cacheservice.connection.deployment.requirement,
specific_parameter_2,  property_value ] }
        - external-database:
             node_filter:
               capabilities:
                 - host:
                     properties:
                        -url_path: { get_property: [ cacheservice.connection.deployment.requirement,
specific_parameter_3,  property_value ] }
        - gmail-api-url:
             node_filter:
               capabilities:
                 - host:
                     properties:
                        -url_path: { get_property: [ cacheservice.connection.deployment.requirement,
specific_parameter_4,  property_value ] }

        - wordpress:
            node: WordPress_for_Caching
            relationship: my_connection
```

```
  Container_for_AFA:
    type: tosca.nodes.Container.Application.Docker.Container
    capabilities:
      container:
        properties:
          protocol: tcp
          secure: false
          network_name: PRIVATE
          initiator: source
          url_path: { get_attributes: [ SELF, private_ip ]}

  WordPress_for_AFA:
    type: tosca.nodes.Container.Application.WordPress
    requirements:
      - container:
          node: Container_for_Caching
          relationship: my_connection
    capabilities:
      wordpress:
          properties:
          protocol: tcp
          secure: false
          network_name: PRIVATE
          initiator: source
          url_path: { get_attributes: [ SELF, private_ip ]}

  AudienceFinder:
    type: tosca.nodes.Container.Application.AFA
    requirements:
      - proxy:
          node_filter:
            capabilities:
              - host:
                  properties:
                    -     url_path:     {get_property:     [     afa.connection.deployment.requirement,
specific_parameter_1,  property_value ] }
      - boxoffice-database:
          node_filter:
            capabilities:
              - host:
                  properties:
                    -     url_path:     {get_property:     [     afa.connection.deployment.requirement,
specific_parameter_2,  property_value ] }
      - wordpress-database:
          node_filter:
            capabilities:
              - host:
                  properties:
                    -     url_path:     {get_property:     [     afa.connection.deployment.requirement,
specific_parameter_3,  property_value ] }
```

```
        - wordpress:
            node: WordPress_for_Caching
            relationship: my_connection


  relationship_templates:
    my_connection:
      type: ConnectsTo
      interfaces:
        Configure:
          inputs:
            targ_notify_port: "some port"

  policies:
    - afa.application.scalability:
        type: tosca.policies.Scaling.Performance
        description: spins one instance when the threshold connection is reached
        properties:
          property_target:
            property_name: Application, Service, Resource
            property_description: The levels that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: execution
            property_description: The stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          property_trigger_1_ID:
            property_name: connection threshold
            property_description: defines the trigger (threshold that will spin up/down the
instances)
            property_type: string
          property_trigger_1_Namespace:
            property_name: connection meter
            property_description: defines the namespace of the service that is monitoring the
connections
            property_type: string
          specific_parameter_1:
            property_name: max connections
            property_description: defines the maximum defines the maximum number of
connections above which the new instance will be deployed
            property_type: integer
          specific_parameter_2:
            property_name: min connections
            property_description: defines the minimum number of connections above wich the
new instance will be deployed
            property_type: integer
    - application.authorization:
        type: tosca.policies.Security.Authorization
        description: requires connection to Remote Autorization Service
        properties:
          property_target:
            property_name: Application, Resource
            property_description: The levels that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: deployment
            property_description: the stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          specific_parameter_1:
            property_name: service address
            property_description: define the url of the remote authorization service
            property_type: url
          specific_parameter_2:
            property_name: service port
            property_description: defines the port of the remote aythorization Service
            property_type: integer
          specific_parameter_3:
            property_name: service protocol
            property_description: defines the connection protocol of the remote Authorization
Service
            property_type: string
          specific_parameter_4:
            property_name: service name
            property_description: defines the name of the remote Autorization Service
            property_type: string

    - cache.service.application.Scalability:
        type: tosca.policies.Scaling.Performance
        description: spins one instance when the estimated of completion time is above
threshold
        properties:
          property_target:
            property_name: Application, Service, Resource
            property_description: The levels that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: execution
            property_description: The stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          property_trigger_1_ID:
            property_name: estimated completion time
            property_description: defines the trigger (threshold that will spin up/down the instances)
            property_type: time
          property_trigger_1_Namespace:
            property_name: cache server
            property_description: defines the namespace of the service that is monitoring the number of queries that must be executed
            property_type: string
          specific_parameter_1:
            property_name: max estimated time of completion
            property_description: defines the latest time of completion above which the new instance will be deployed
            property_type: time
          specific_parameter_2:
            property_name: min estimated time of completion
            property_description: defines the earliest time of completion under which the new instance will be undeployed
            property_type: time

    - application.execution.time:
        type: tosca.policies.Execution.Schedule
        description: executes the application following a cron job like syntax
        properties:
          property_target:
            property_name: Application, Service, Container
            property_description: the node that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: execution
            property_description: the stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          specific_parameter_1:
            property_name: cron-job arg
            property_description: a cron-job like argument list
            property_type: string

    - container.resource.deployment.requirements:
        type: tosca.policies.Deployment.Ressource
        description: defines the minimum requirements for the container
        properties:
          property_target:
            property_name: container
            property_description: the container that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: deployment
            property_description: the stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          specific_parameter_1:
            property_name: min cpu
            property_description: the minimum number of cpus
            property_type: integer
          specific_parameter_2:
            property_name: min ram
            property_description: the minimum amount of Memory
            property_type: integer
          specific_parameter_3:
            property_name: min disk
            property_description: the minimum size of disk
            property_type: integer

    - cacheservice.connection.deployment.requirement:
        type: tosca.policies.Deployment.Ressource
        description: defines the connection requirements for the container
        properties:
          property_target:
            property_name: Resource
            property_description: the container that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: deployment
            property_description: the stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          specific_parameter_1:
            property_name: Word_Press_url
            property_description: the url of the Word_Press database that must be reachable
            property_type: url
            property_value: http://some_word_press_database_url
          specific_parameter_2:
            property_name: Box_Office_url
            property_description: the url of the Box_Office database that must be reachable
            property_type: url
            property_value: http://some_external_box_office_database_url
          specific_parameter_3:
            property_name: external_database_url
            property_description: the url of the external_database that must be reachable
            property_type: url
          property_value: http://some_external_database_url
        specific_parameter_4:
          property_name: Gmail_API_url
          property_description: the url of the Gmail_API that must be reachable
          property_type: url
          property_value: http://some_gmail_api_url

- afa.connection.deployment.requirement:
    type: tosca.policies.Deployment.Ressource
    description: defines the connection requirements for the container
    properties:
      property_target:
        property_name: Resource
        property_description: the container that will be affected by the policy
        property_type: tosca.nodes
      property_stage:
        property_name: deployment
        property_description: the stage that will be affected by the policy
        property_type: string
      property_priority:
        property_name: 100
        property_description: the priority with which the policy will be executed
        property_type: integer
      specific_parameter_1:
        property_name: Word_Press_url
        property_description: the url of the Word_Press database that must be reachable
        property_type: url
        property_value: http://some_word_press_database_url
      specific_parameter_2:
        property_name: Box_Office_url
        property_description: the url of the Box_Office database that must be reachable
        property_type: url
        property_value: http://some_box_office_database_url
      specific_parameter_3:
        property_name: proxy_url
        property_description: the url of the poxy that must be reachable
        property_type: url
        property_value: http://some_proxy/
```

13. Annex B

This Annex contains the Compressed Topology Application Description Template (**audience-finder.yaml** and **topology.yaml**) of Use Case 1.

audience-finder.yaml defines the node types that are used in the Compressed Topology Application Description Template of Use Case 1

```
tosca_definitions_version: tosca_simple_yaml_1_0

node_types:
  tosca.nodes.Container.Application.Docker.AFA:
    derived_from: tosca.nodes.Container.Application.Docker

    requirements:

      - proxy: tosca.capabilities.Endpoint
      - boxoffice-database: tosca.capabilities.Endpoint
      - wordpress-database: tosca.capabilities.Endpoint

  tosca.nodes.Container.Application.Docker.CachingService:
    derived_from: tosca.nodes.Container.Application.Docker
    requirements:
      - wordpress-database: tosca.capabilities.Endpoint
      - boxoffice-database: tosca.capabilities.Endpoint
      - external-database: tosca.capabilities.Endpoint
      - gmail-api-url: tosca.capabilities.Endpoint

policy_types:
  tosca.policies.Security.Authorization:
    derived_from: tosca.policies.Security
    description: authorization policy derived from security

  tosca.policies.Scaling.Performance:
    derived_from: tosca.policies.Scaling
    description: Performance policy derived from scaling

  tosca.policies.Deployment.Ressource:
    derived_from: tosca.policies.Deployment
    description: Ressource policy derived from Deployment

  tosca.policies.Execution.Schedule:
    derived_from: tosca.policies.Execution
    description: Schedule policy derived from Execution
```

topology.yaml defines the Compressed Topology Application Description Template of Use Case 1

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - audience_finder.yaml

repositories:
  docker_hub: https://registry.hub.docker.com/

description: Template with requirements against hosting infrastructure.

topology_template:

  node_templates:
    CachingService:
      type: tosca.nodes.Container.Application.Docker.CachingService
      requirements:
        - host:
            node_filter:
              capabilities:
                - host:
                   properties:
                    - num_cpus: { in_range: [ min, max ] }
                    - mem_size: { in_range: [ min, max ] }
                    - disk_size: { in_range: [ min, max ] }
        - wordpress-database:
            node_filter:
              capabilities:
                - host:
                    properties:
                       - url_path: { get_property: [ cacheservice.connection.deployment.requirement,
specific_parameter_1,  property_value ] }
        - boxoffice-database:
            node_filter:
              capabilities:
                - host:
                    properties:
                       - url_path: { get_property: [ cacheservice.connection.deployment.requirement,
specific_parameter_2,  property_value ] }
        - external-database:
             node_filter:
               capabilities:
                 - host:
                     properties:
                        - url_path: {get_property: [ cacheservice.connection.deployment.requirement,
specific_parameter_3,  property_value ] }
        - gmail-api-url:
             node_filter:
               capabilities:
                - host:
                    properties:
                      - url_path: {get_property: [ cacheservice.connection.deployment.requirement,
specific_parameter_4,  property_value ] }

      artifacts:
        file: CachingService
        type: tosca.artifacts.Deployment.Image.Container.Docker
        repository: docker_hub

    AudienceFinder:
      type: tosca.nodes.Container.Application.Docker.AFA
      requirements:
        - proxy:
            node_filter:
              capabilities:
                - host:
                    properties:
                      -    url_path:    {    get_property:    [    afa.connection.deployment.requirement,
specific_parameter_1,  property_value ] }
        - boxoffice-database:
            node_filter:
              capabilities:
                - host:
                    properties:
                      -    url_path:    {    get_property:    [    afa.connection.deployment.requirement,
specific_parameter_2,  property_value ] }
        - wordpress-database:
            node_filter:
              capabilities:
                - host:
                    properties:
                      -    url_path:    {    get_property:    [    afa.connection.deployment.requirement,
specific_parameter_3,  property_value ] }

      artifacts:
        file: AFA
        type: tosca.artifacts.Deployment.Image.Container.Docker
        repository: docker_hub

  policies:
    - afa.application.scalability:
        type: tosca.policies.Scaling.Performance
        description: spins one instance when the threshold connection is reached
        properties:
          property_target:
            property_name: Application, Service, Resource
            property_description: The levels that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: execution
            property_description: The stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          property_trigger_1_ID:
            property_name: connection threshold
            property_description: defines the trigger (threshold that will spin up/down the instances)
            property_type: string
          property_trigger_1_Namespace:
            property_name: connection meter
            property_description: defines the namespace of the service that is monitoring the connections
            property_type: string
          specific_parameter_1:
            property_name: max connections
            property_description: defines the maximum defines the maximum number of connections above which the new instance will be deployed
            property_type: integer
          specific_parameter_2:
            property_name: min connections
            property_description: defines the minimum number of connections above wich the new instance will be deployed
            property_type: integer

    - application.authorization:
        type: tosca.policies.Security.Authorization
        description: requires connection to Remote Autorization Service
        properties:
          property_target:
            property_name: Application, Resource
            property_description: The levels that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: deployment
            property_description: the stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          specific_parameter_1:
            property_name: service address
            property_description: define the url of the remote authorization service
            property_type: url
          specific_parameter_2:
            property_name: service port
            property_description: defines the port of the remote aythorization Service
            property_type: integer
          specific_parameter_3:
            property_name: service protocol
            property_description: defines the connection protocol of the remote Authorization
Service
            property_type: string
          specific_parameter_4:
            property_name: service name
            property_description: defines the name of the remote Autorization Service
            property_type: string

    - cache.service.application.Scalability:
        type: tosca.policies.Scaling.Performance
        description: spins one instance when the estimated of completion time is above
threshold
        properties:
          property_target:
            property_name: Application, Service, Resource
            property_description: The levels that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: execution
            property_description: The stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          property_trigger_1_ID:
            property_name: estimated completion time
            property_description: defines the trigger (threshold that will spin up/down the
instances)
            property_type: time
          property_trigger_1_Namespace:
            property_name: cache server
            property_description: defines the namespace of the service that is monitoring the
number of queries that must be executed
            property_type: string
          specific_parameter_1:
            property_name: max estimated time of completion
            property_description: defines the latest time of completion above which the new
instance will be deployed
            property_type: time
          specific_parameter_2:
            property_name: min estimated time of completion
            property_description: defines the earliest time of completion under which the new
instance will be undeployed
            property_type: time

    - application.execution.time:
        type: tosca.policies.Execution.Schedule
        description: executes the application following a cron job like syntax
        properties:
          property_target:
            property_name: Application, Service, Container
            property_description: the node that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: execution
            property_description: the stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          specific_parameter_1:
            property_name: cron-job arg
            property_description: a cron-job like argument list
            property_type: string

    - container.resource.deployment.requirements:
        type: tosca.policies.Deployment.Ressource
        description: defines the minimum requirements for the container
        properties:
          property_target:
            property_name: container
            property_description: the container that will be affected by the policy
            property_type: tosca.nodes
          property_stage:
            property_name: deployment
            property_description: the stage that will be affected by the policy
            property_type: string
          property_priority:
            property_name: 100
            property_description: the priority with which the policy will be executed
            property_type: integer
          specific_parameter_1:
            property_name: min cpu
            property_description: the minimum number of cpus
            property_type: integer
          specific_parameter_2:
            property_name: min ram
            property_description: the minimum amount of Memory
            property_type: integer
          specific_parameter_3:
            property_name: min disk
            property_description: the minimum size of disk
            property_type: integer

    - cacheservice.connection.deployment.requirement:
        type: tosca.policies.Deployment.Ressource
        description: defines the connection requirements for the container
        properties:
```

```
    property_target:
      property_name: Resource
      property_description: the container that will be affected by the policy
      property_type: tosca.nodes
    property_stage:
      property_name: deployment
      property_description: the stage that will be affected by the policy
      property_type: string
    property_priority:
      property_name: 100
      property_description: the priority with which the policy will be executed
      property_type: integer
    specific_parameter_1:
      property_name: Word_Press_url
      property_description: the url of the Word_Press database that must be reachable
      property_type: url
      property_value: http://some_word_press_database_url
    specific_parameter_2:
      property_name: Box_Office_url
      property_description: the url of the Box_Office database that must be reachable
      property_type: url
      property_value: http://some_external_box_office_database_url
    specific_parameter_3:
      property_name: external_database_url
      property_description: the url of the external_database that must be reachable
      property_type: url
      property_value: http://some_external_database_url
    specific_parameter_4:
      property_name: Gmail_API_url
      property_description: the url of the Gmail_API that must be reachable
      property_type: url
      property_value: http://some_gmail_api_url

- afa.connection.deployment.requirement:
    type: tosca.policies.Deployment.Ressource
    description: defines the connection requirements for the container
    properties:
      property_target:
        property_name: Resource
        property_description: the container that will be affected by the policy
        property_type: tosca.nodes
      property_stage:
        property_name: deployment
        property_description: the stage that will be affected by the policy
        property_type: string
      property_priority:
        property_name: 100
        property_description: the priority with which the policy will be executed
        property_type: integer
      specific_parameter_1:
        property_name: Word_Press_url
      property_description: the url of the Word_Press database that must be reachable
      property_type: url
      property_value: http://some_word_press_database_url
    specific_parameter_2:
      property_name: Box_Office_url
      property_description: the url of the Box_Office database that must be reachable
      property_type: url
      property_value: http://some_box_office_database_url
    specific_parameter_3:
      property_name: proxy_url
      property_description: the url of the poxy that must be reachable
      property_type: url
      property_value: http://some_proxy/
```