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:

D5.4 First Set of Templates and Services of Use Cases

Due date of deliverable: 31/12/2017 Actual submission date: 25/12/2017

WPL: Gabriele Pierantoni

Dissemination Level: PU

Version: Final

1. Table of Contents

2. List of Figures and Tables

Figures

Tables

3. Status, Change History and Glossary

Status:	Name:	Date:	Signature:
Draft:	Gabriele Pierantoni	17/12/17	*Gabriele Pierantoni*
Reviewed:	Anastasia Anagnostou	20/12/17	*Anastasia Agnoustou*
Approved:	Tamas Kiss	25/12/17	Tamas Kiss

Table 1, Status Change History

Version	Date	Pages	Author	Modification
V0.1	07/12	ALL	G. Pierantoni	Skeleton
V1.0	16/12	ALL	G. Pierantoni	First Draft of the Deliverable
V1.1	17/12	ALL	G. Pierantoni	Second Draft to be reviewed.
V1.3	20/12	ALL	G. Pierantoni	Addressed Tamas remarks, added code description
V1.4	22/12	ALL	G. Pierantoni	Addressed Anastasia (Reviewer) remarks, added some corrections and small text additions

Table 2, Deliverable Change History

4. Glossary

ADT	Application Description Template
API	Application Programming Interface
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
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

Table 3, Glossary

5. Introduction

COLA Description of Work (DoW) specifies **Deliverable D5.4 "*First set of templates and services of Use Cases*"** as follows: **"*This deliverable will publish the first set of templates and the description of the services to be used in the SME and public sector use-cases and will write a report about these templates and services*"**

This deliverable describes the Application Description Templates (ADTs) implementation of the three Use Cases proposed by the COLA Industrial and Academic partners plus one additional Use Case, that of Data Avenue[1][2]. This deliverable explains the design changes that have been adopted for this release and describes the generic structure of the implementation of the Application Description Templates.

The deliverable also describes the structure of the github[3] repository that has been created to contain the TOSCA code (https://github.com/COLAProject/COLARepo) and provides the commented code of the ADTs to offer both a clear view on their structure and the related implementation details.

This Deliverable is the fourth 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 that are involved in either developing new applications or porting existing ones to the COLA infrastructure.
Deliverable D5.4 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** - Defines the generic structure of the Application Description Templates and how it relates to the various components of the MiCADO infrastructure.
4. **Section 8** - Describes the generic steps needed to create an Application Description Template.
5. **Section 9** – Describes the implementation of the four Use Cases:
    a. **Use Case 1: Scalable hosting, testing and automation for Small to Medium-sized Enterprises (SMEs) and public sector organisations**, developed by Outlandish and The Audience Agency,
    b. **Use Case 2: Bursting onto the Cloud from SakerGrid** – developed by Brunel University and Saker Solutions,
    c. **Use Case 3: Social media data Analytics for Public Sector Organisations** – developed by Inycom and SARGA.
    d. **Use Case 4: Data Avenue**, **a file commander tool for data transfer, enabling easy data moving between various storage services** – developed by CloudSME and SZTAKI
6. **Section 10** – Describes the structure of the Github repository that contains the ADTs, it further describes each of its directories and lists the declared types.
7. **Section 11** – Describes the structure of the main component (the Topology Template) of a meaningful ADT example, that of Use Case 3.
8. **Section 12** – Concludes the Deliverable offering some remarks on the status of WP5 and future work.
9. **Section 13** – Contains the References.
10. **Sections 14 to 20 –** Contain the listing of all the code that compose the ADTs.

6. Relationship with other Work Packages and Deliverables

As introduced in Section 5, the **Application Description Templates (ADTs)** are closely related to various fundamental aspects of the COLA project:
Firstly, the ADTs describe the applications that are to be ported to the COLA infrastructure; because of this, ADTs are closely related to the COLA architecture (more specifically to the MiCADO[4][5] infrastructure which COLA is based upon) itself. Finally, the ADTs described in this Deliverable are the result of the joint process of the definition of an abstract approach to describe the applications and the implementation choices of the MiCADO infrastructure.
As a result, this deliverable is closely related to Work Packages and Deliverables that describes the concepts highlighted in the previous paragraph.

Deliverable 5.4 is published by **Work Package 5 – Application Definition Templates.** This work package has previously published three Deliverables: **D5.1 – "*Analysis of existing Application Description Approaches*"**, **D5.2 – "*Specification of the Application Description Concept",*** and **D5.3 – "Integration of the Templates with the Selected Application Description Approach".**

- D5.1 offers a state of the art overview of the application description and execution. D5.1 details the reason behind the choice of adopting TOSCA[6][7] as the language specifications for COLA ADTs.
- D5.2 describes the proposed COLA approach to the problem: a three-layered Application Description Template based on the language specifications which also defines policies at each of its layers.
- D5.3 describes how the concept highlighted in D5.2 are applied to define the three Use Cases defined in D8.1.

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

The ADTs 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 D5.4 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 ensures that the Use Cases of Work Package 8 can be supported by the Application Descriptions Templates.

7. The Application Description Templates in COLA

The COLA project uses TOSCA-based Application Description Templates to describe the application architecture and the policies that govern their lifecycle. While previous deliverables (D5.2 and D5.3) describe a generic structure for the description of applications and policies that referred to the abstract COLA architecture and concepts, this Deliverable focuses on the current MiCADO implementation. The Application Description Templates described in this Deliverable reflect the MiCADO current and planned implementation for the foreseeable six months and it will represent the design of the ADTs that will be parsed and used by the TOSCA submitter component that will be implemented starting in January 2018.

As introduced in Deliverable D5.2 and D5.3, the COLA Application Description Templates (ADTs) describe two main aspects of each application: its topology (e.g. the set of components that compose it alongside the relationships) and its policies (the set of rules that govern the lifecycle of its components). The ADTs represent an information conduit between the Application Developers and various components of the MiCADO infrastructure as illustrated in Figure 1.

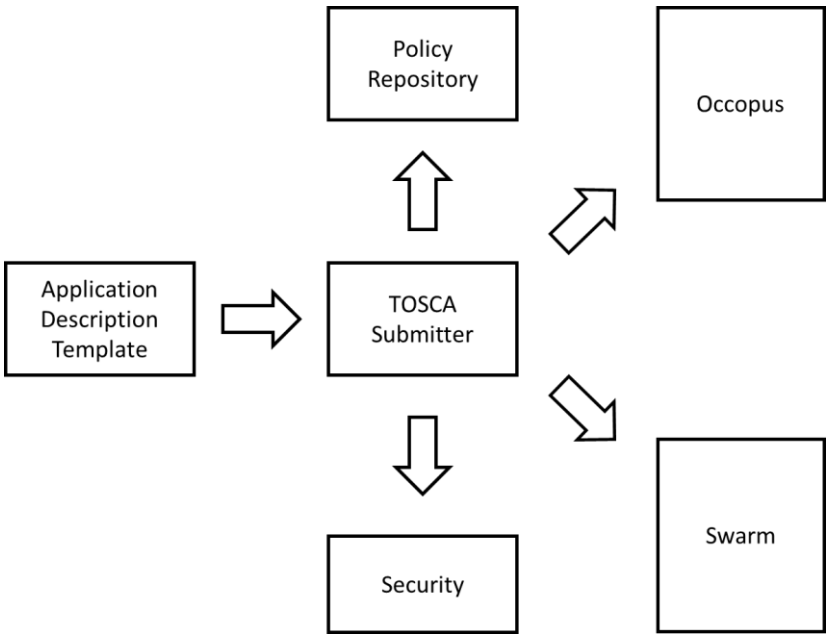

Figure 1, Connections among the Application Description Template and related components of the COLA architecture.

The TOSCA submitter parses the ADTs and dispatches the relevant information to the different components of the MiCADO infrastructure. These connections can be detailed as:

- **Policies** are stored in the Policy Repository and used throughout the lifecycle of the application to define the right action.
- **Occopus**[8] receives and processes the information that describes the selection and execution of the virtual machines.

- **Swarm**[9] receives and processes the information that describes the selection and execution of the Docker Images.
- **The Security Infrastructure** receives and processes the information that is specific to security concerns.

The Application Description Templates reflect this architecture and have been modified from those described in D5.2 and D5.3 to more closely adapt to the current and planned MiCADO implementations. The main change between the current implementation of the ADTs and the general architecture proposed in D5.2 are two fold and are described in Figure 2.

First: the current ADT have compressed the Application, Service and Resource Layer into a Docker Image. The general approach proposed in the COLA Description of Work proposed three layered ADTs that detailed the application, service and resource layer to foster re-usability of the ADTs. Although this design principle had not been abandoned, the current implementation of MiCADO requires that the Application Developers provide a Docker Image (Uploaded to Docker Hub) that contains both the service and the specific application code and data. This renders unnecessary to describe the different application and service components (and their lifecycle steps) as they are deployed as one or more Docker Images, the entire ADT has been simplified accordingly.

Second: the container layer has been separated into two sub-levels, one that indicates which Docker Image has to be used (along the command line details to execute it) and one level that details the virtual machine inside which the container has to be deployed.

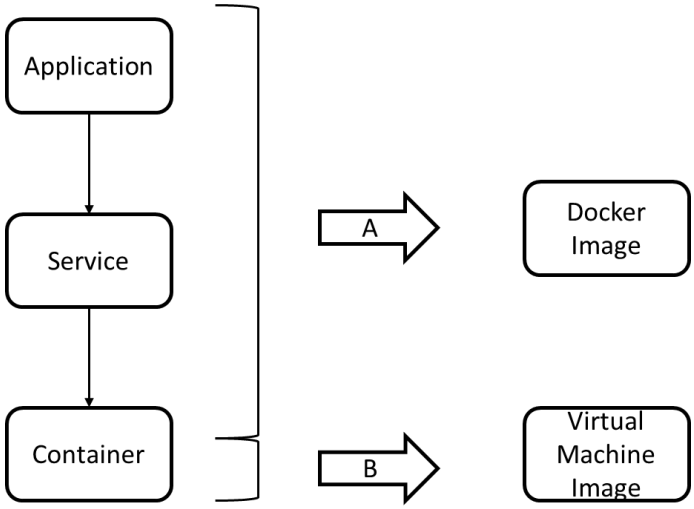

Figure 2, Implementation of ADTs with respect of the three-layered design.

The mapping of the ADT to the type of resources of MiCADO is achieved by the two layers of the ADT by specifying information at each of the levels as illustrated in Figure 43. The Docker Image node specifies the Docker Name and the command line which will be used to define select the image and how to run it. The Docker node does not need any mapping as it defines a specific Docker Image.

On the other hand, the Virtual Machine Nodes do not specify any specific instance of the Image to be used but rather give information on its characteristic and let Occopus find the optimal Virtual Image to match-make its request. The Application Developer can use a generic

Virtual Machine Node Image to let Occopus decide which Cloud Provider to use or she/he can select a sub type of the Virtual Machine Node to directly instruct Occopus on which provider she/he wants to use.

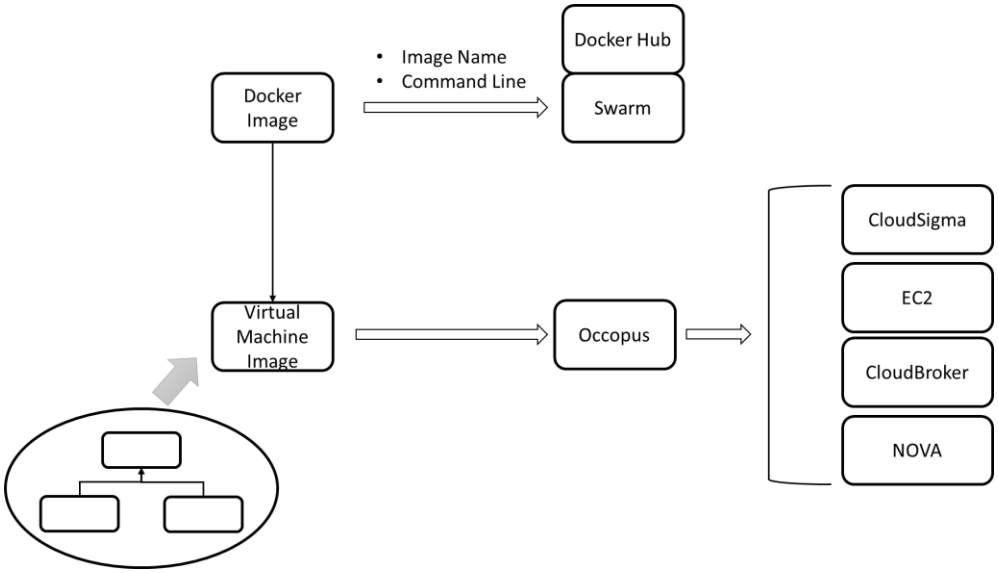

Figure 3, Mapping of Docker and Virtual Machine Images with ADTs

All applications that are ported into COLA for execution in MiCADO, are defined as an ADT which contains a Topology Templates and various custom defined types. In turn, each Topology template contains Inputs, Policies, Nodes and Relationships.

The Application Developer decides in how many Docker Images she/he wants to decompose his application and the relative Topology Template contains two nodes for each Docker Image as illustrated in Figure 4. The top node describes the Docker Images while the bottom node describes the Virtual Machine in which the Docker Image will be instantiated.

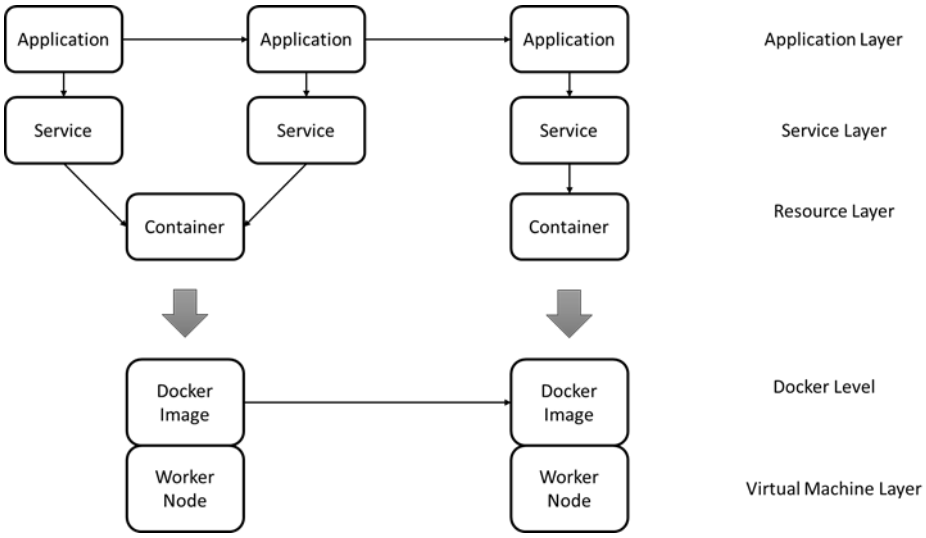

Figure 4, Mapping of Application Description Templates into Docker Images and Worker Nodes

Each ADT follows the same structure which is described in Figure 5. Each ADT consists of one Topology Template that comprehends the following components:

- **Input** section groups together fields that Application Developers are likely to override their default value.
- **Policies** section defines the policies that are applied within the Application. Each policy can be applied to one or more nodes.
- **Docker Images** section define a set of nodes that specify the Docker Images that contain the applications. Application Developers will upload the Docker Image to Docker Hub prior to defining the ADT and will specify the name of the Docker Image and the details of its command line in the Docker Image Type.
- **Worker Nodes** section defines the characteristics of the Virtual Machines that will host the Docker Images defined in the section above. As MiCADO is a multi-cloud platform that supports various Cloud Providers, Worker Nodes are defined with either a generic node type leaving to Occopus to select the Cloud Provider that is most appropriate or directly defined using a sub-type that specifies the Cloud Provider that has to be used.
- **Output** section groups together fields whose values will be set by the TOSCA submitter and returned to the Application Developer

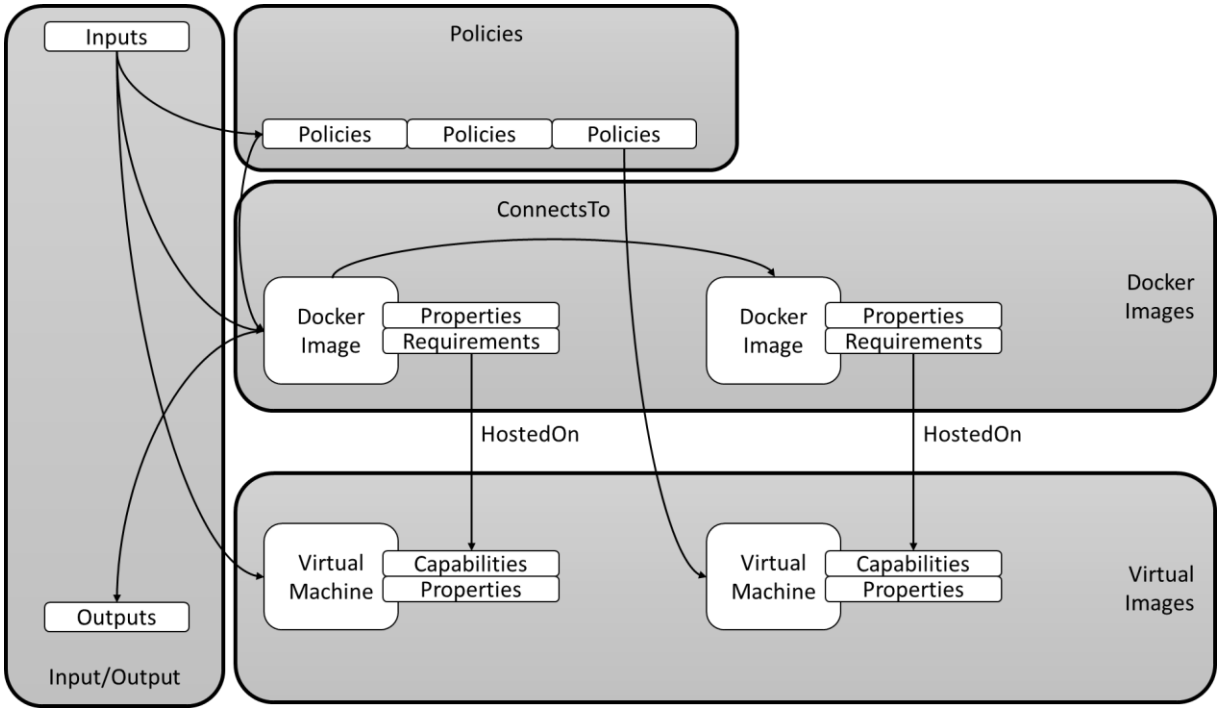

Figure 5, Structure of the Implementation of the Application Description Templates

For the implementation of these Use Cases only, a subset of the policies that were originally defined in Deliverable D5.3 have been used. Other policies have been modified to adapt to the evolving design of the infrastructure while some others have been changed to more closely reflect the TOSCA design guidelines.

8. Proposed steps to define the Application Description Templates in COLA

The creation of the ADTs can be summarized with the following steps which can be followed by Application Developers.

Analysis of the architecture of the application. The application is divided into its main components. The goal of this step is to determine which components will be internal (e.g. deployed in MiCADO) and which components will be external (e.g. not deployed in MiCADO). The ADT will only describe the internal components.

Definition of the Docker structure of the application. The application is divided into sections each of which represents a Docker Image and a Virtual Machine where the image will be deployed. The definition of the "vertical slices" that compose the ADT should reflect not only the architectural structure of the application but also the different policies that need to be applied. As an example, an application that contains components that requires scalability should separate them from components that do not require scalability to optimize the execution of the application.

- All Docker Images are of the same type. The user specifies the name of the Docker Image that is uploaded into Docker Hub and the command line that will execute it.
- Select the Type of Virtual Machine to reflect the choice of Cloud Provider. Generic Type to leave the mapping to Occopus, or specific types to directly choose a particular Cloud Provider.

Selection of policies and their association to the relative nodes in the Application Topology.

Selection of input values that users are likely to be overridden by user and list them in the Input section, specify the default value for each of the inputs.

Selection of output values that will be returned by the TOSCA submitter.

9. Examples of Application Description Templates in COLA

9.1 Introduction

The implementations described in this Deliverable represent the ADTs of the three main Use Cases described in D8.1. These Use Cases cover a significant range of the MiCADO functionalities and their ADTs constitute a reasonable base for further extension to implement the other scenarios that will be covered during the project.

The three use cases covered in D8.1 did not require the definition of “horizontal” relationships between the nodes that describe the Docker Images (Use Case 1 contains two Docker Images but their relationship is implicit through the definition of a cron-job and not explicit). As this is an important feature that will be key to describe complex topologies, we have added the ADT for a further Use Case: Data Avenue in which we describe a Template Topology of six nodes with horizontal “ConnectsTo” and vertical “HostedOn” relationships.

9.2 Use Case 1 - Scalable hosting, testing and automation for SMEs and public sector organisations

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. 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[1][2] in Linux).

The components fall into two separate categories as described in Figure 6: Internal Components that will be deployed in MiCADO (depicted in red colour) and External Components that will not be deployed in MiCADO (depicted in blue colour).

[1] http://man7.org/linux/man-pages/man8/cron.8.html
[2] https://linux.die.net/man/1/crontab

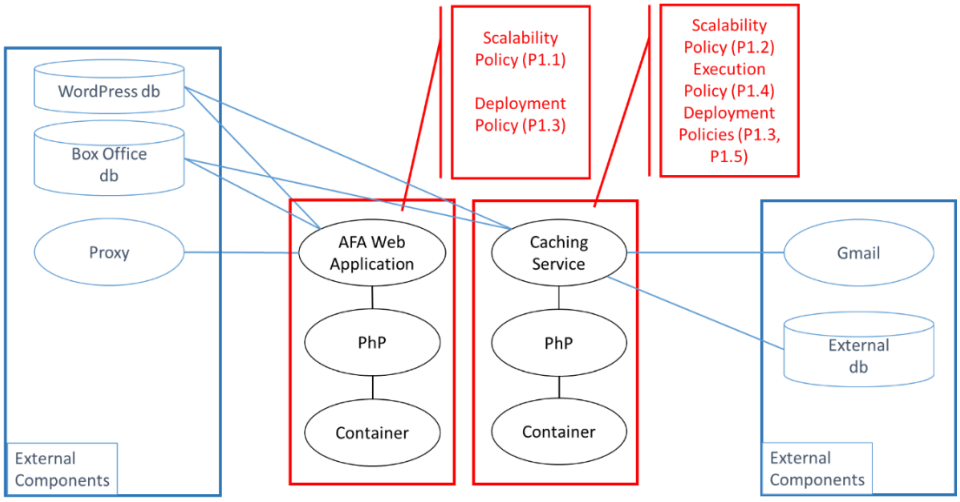

Figure 6, Components and Policies of Use Case 1

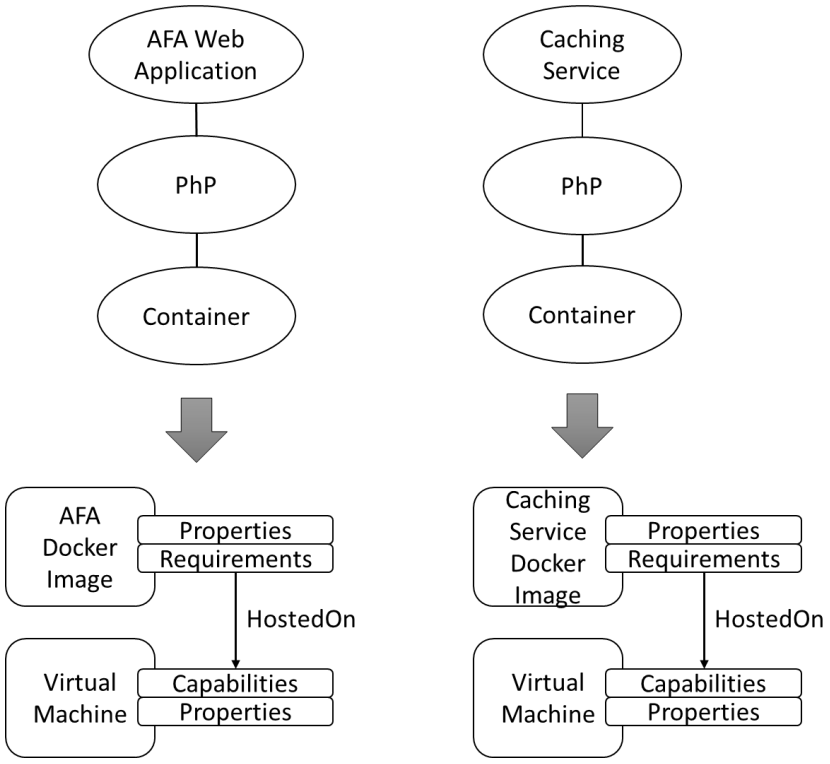

Figure 7, Implementation of Use Case 1

Figure 7 describes the architecture of the implementation of Use Case 1. The application is divided into two Docker images and Virtual Machine nodes that are connected with a HostedOn type relationship. The Docker Image nodes are called AFA and CachingService both of type tosca.nodes.MiCADO.Container.Application.Docker. The Virtual Machine nodes are of type tosca.nodes.MiCADO.Occopus.CloudSigma.Compute and define a specific Cloud Provider (CloudSigma) for the Virtual Machine execution.

The policies that apply to the implementation of the nodes are recapitulated in the Table 4 below.

	Policy	Notes
P1.1	Consumption Based Scalability	It has been modified to Consumption Based Scalability as at the moment the MiCADO infrastructure does not support monitoring of web connections. Consumption Based Scalability defines a threshold above which a new instance will be deployed and a threshold below which the instance will be un-deployed.
P1.2	Deadline Based Scalability	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.
P1.3	Connection Deployment Policy	Defines the set of inbound connections. This policy has been modified to take into account that only inbound connections are to be specified to the security infrastructure.
P1.4	Execution Policy	It defines that the component has to be executed at fixed times.
P1.5	Resource Deployment Policy	Defines the requirements of the Virtual Machine in terms of CPU, Memory Size and Disk size. It is directly defined as Capabilities of the Virtual Machine.

Table 4, Policies applied to the implementation of Use Case 1

9.3 Use Case 2 - Bursting onto the Cloud from SakerGrid

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 simulation experimentation. The solution is to extend to the Cloud the Distributed Computing Environment which is now based on a desktop grid (SakerGrid) to achieve a near linear improvement in response performance. The application follows a standard job-submission system architecture and consists of several components only one of which has to be ported into the MiCADO infrastructure. The implemented Use Case only covers the REPAST open source simulation while the Evacuation Simulation implementation is currently being finalized.

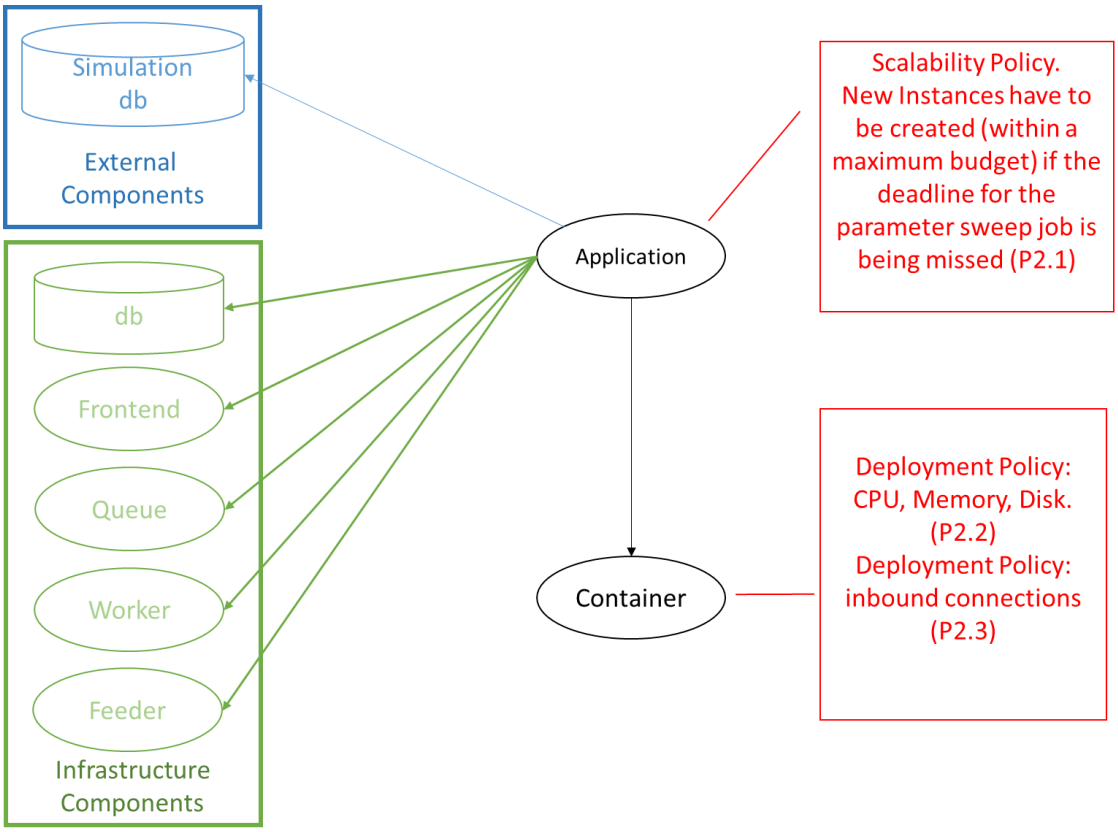

Figure 8, Components and Policies of Use Case 2

The components fall into two separate categories as described in Figure 8: Internal (depicted in black colour) and External (depicted in blue colour). External Components are further divided into External and Infrastructure Components (depicted in green colour) which represent elements of the simulation platform that are not described in the ADT. The components that will be ported into the MiCADO framework is the Application, which is supported by several Infrastructure-Level components: Frontend, Queue, Feeder, Worker, Application and db.

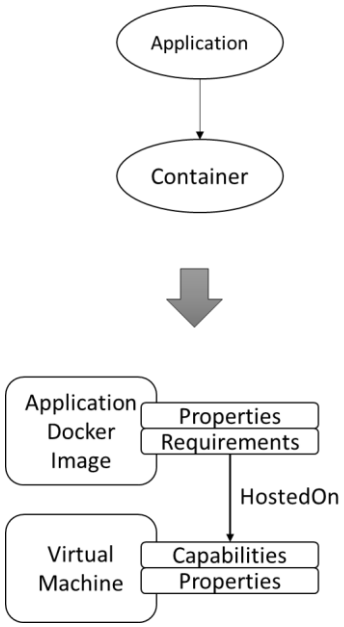

Figure 9, Implementation of Use Case 2

Figure 9 describes the architecture of the implementation of Use Case 2. The application consists in one Docker image and one Virtual machine node that are connected with a HostedOn type relationship. The Application Docker Image node is of type tosca.nodes.MiCADO.Container.Application.Docker, the Virtual Machine node is of type tosca.nodes.MiCADO.Occopus.CloudSigma.Compute and defines a specific Cloud Provider (CloudSigma) for the Virtual Machine execution. The policies that apply to the implementation of the nodes are recapitulated in Table 5 below.

	Policy	Notes
P2.1	Cost Constrained Deadline Based Scalability	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. Scalability is constrained by a maximum budged and a cut-off threshold.
P2.2	Connection Deployment Policy	Defines the set of inbound connections. This policy has been modified to take into account that only inbound connections are to be specified to the security infrastructure.
P2.3	Resource Deployment Policy	Defines the requirements of the Virtual Machine in terms of CPU, Memory Size and Disk size. It is directly defined as Capabilities of the Virtual Machine.

Table 5, Policies applied to the implementation of Use Case 2

9.4 Use Case 3 - Social media data analytics for public sector organisations

Use Case 3 deals with a modified version of the Eccobuzz[10] platform with enriched functionalities called Magician[11]. 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[12]). The deployment in MiCADO aims at achieving greater scalability. The components fall into two separate categories as described in Figure 10: Internal Components that will be deployed in MiCADO (depicted in red colour) and External Components that will not be deployed in MiCADO (depicted in blue colour).

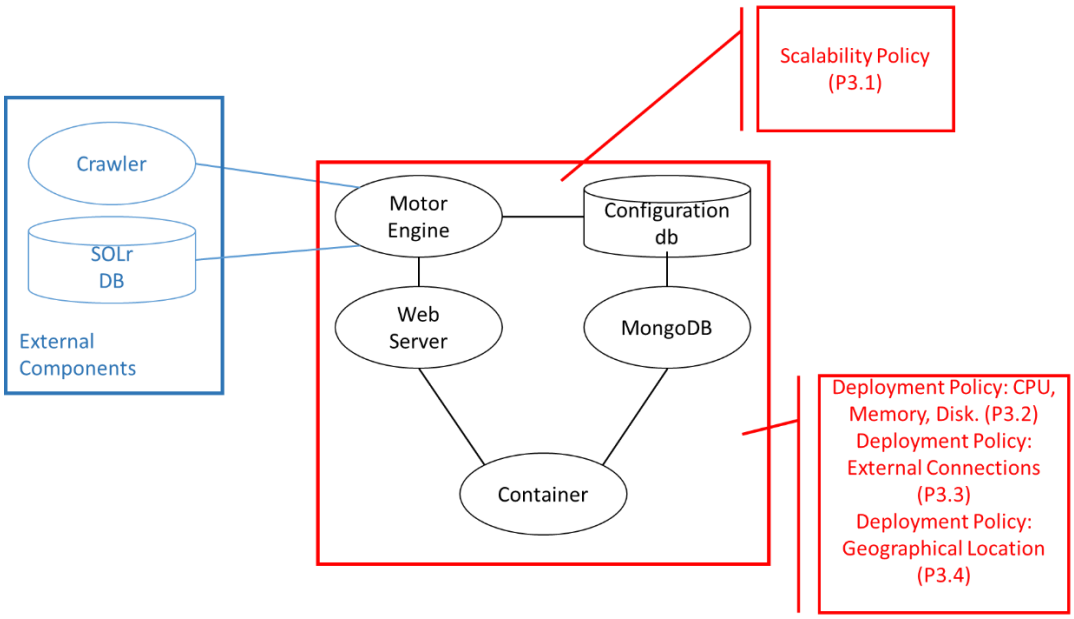

Figure 10, Components and Policies of Use Case 3

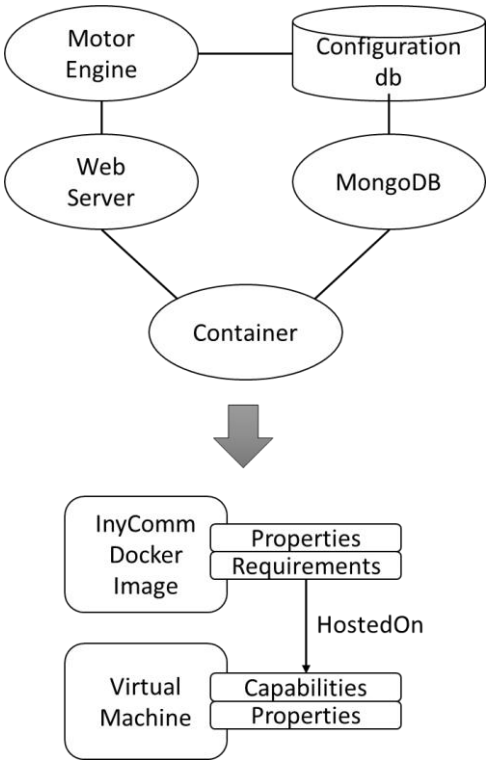

Figure 11, Implementation of Use Case 3

Figure 11 describes the architecture of the implementation of Use Case 3. The application consists of one Docker image and one Virtual Machine nodes that are connected with a HostedOn type relationship. The InyComm Docker Image node is named inycom and it is of type tosca.nodes.MiCADO.Container.Application.Docker, the Virtual Machine node is of type tosca.nodes.MiCADO.Occopus.CloudSigma.Compute and defines a specific Cloud Provider (CloudSigma) for the Virtual Machine execution. The policies that apply to the implementation of the nodes are recapitulated in Table 6 below.

	Policy	Notes
P3.1	Consumption Based Scalability	Consumption Based Scalability defines a threshold above which a new instance will be deployed and a threshold below which the instance will be un-deployed.
P3.2	Resource Deployment Policy	Defines the requirements of the Virtual Machine in terms of CPU, Memory Size and Disk size. It is directly defined as Capabilities of the Virtual Machine.
P3.3	Connection Deployment Policy	Defines the set of inbound connections. This policy has been modified to take into account that only inbound connections are to be specified to the security infrastructure.
P3.4	Location Deployment Policy	It dictates that the container must be physically located in the European Union.

Table 6, Policies applied to the implementation of Use Case 3

9.5 Use Case 4 – Data Avenue

Use Case 4 – Data Avenue covers the definition of a Topology Template for the deployment of an instance of the Data Avenue service into the MiCADO infrastructure. DataAvenue [2] is a multi-platform file transfer and file staging facility developed by SZTAKI within the SCI-BUS and CloudSME projects, its porting into MiCADO is one of the official Use Cases of Cola. The deployment in MiCADO aims at achieving greater scalability of its main component: a web application deployed in a Tomcat container. Figure 12 describes the three internal components of the DataAvenue application: the Proxy, the DataAvenue Web Application and the DataAvenue database.

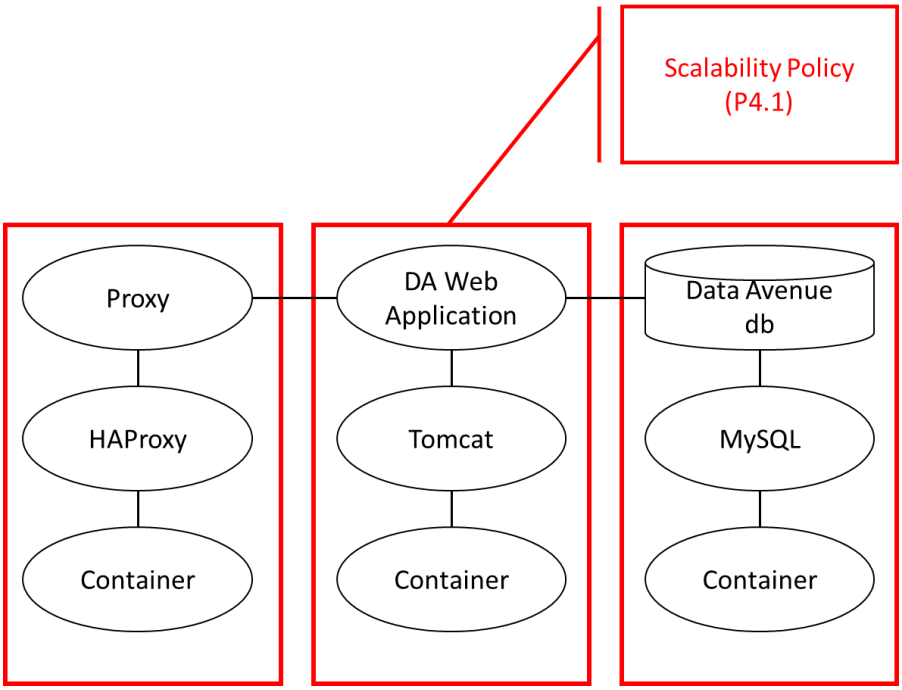

Figure 12, Components and Policies of Use Case 4

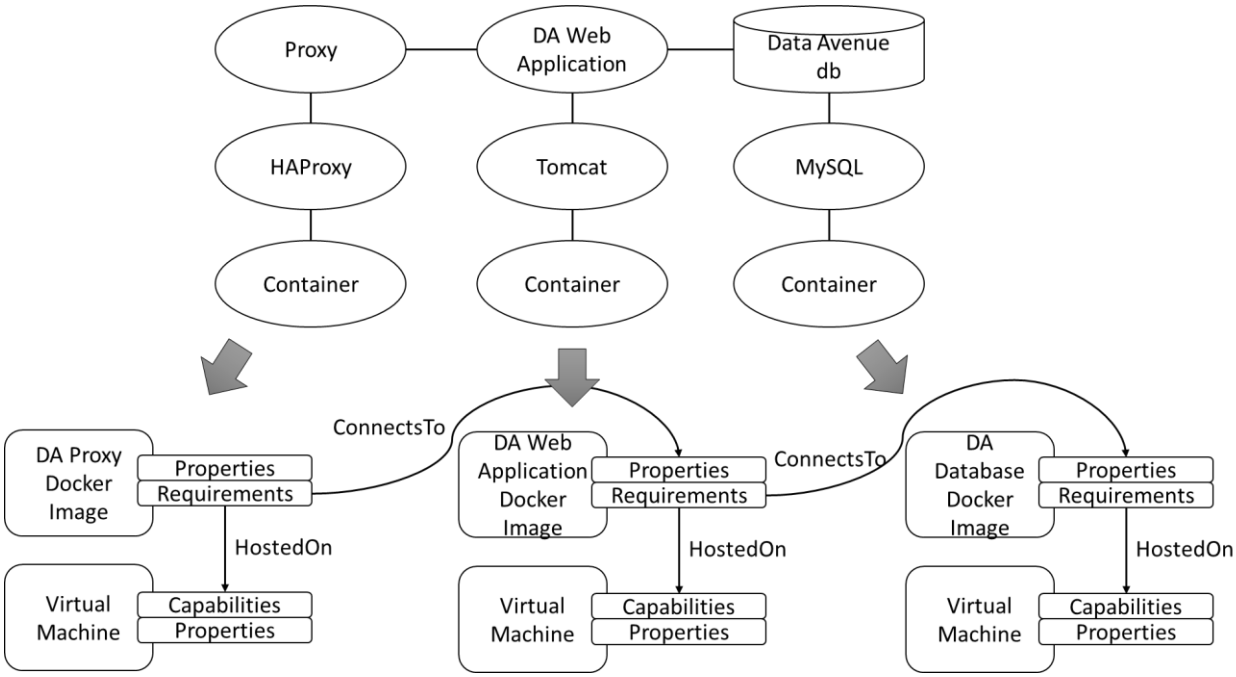

Figure 13, Implementation of Use Case 4

Figure 13 describes the architecture of the implementation of Use Case 4. The application consists in three Docker images connected through a HostedOn relationship to three Virtual Machine nodes. The policies that apply to the implementation of the nodes are recapitulated in the table below.

	Policy	Notes
P4.1	Consumption Based Scalability	Consumption Based Scalability defines a threshold above which a new instance will be deployed and a threshold below which the instance will be un-deployed.

Table 7, Policies applied to the implementation of Use Case 4

10. Structure of the COLA Application Description Repository

The ADTs are stored in a github repository available at: https://github.com/COLAProject. To help with re-usability of the code, we have structured the code in three main areas described in the screenshot of Figure **14**. The status of the code has been saved for the submission of this Deliverable with a Release (0.1) called Release for Deliverable D5.4. The full code of this Release is listed in Appendices A to L.

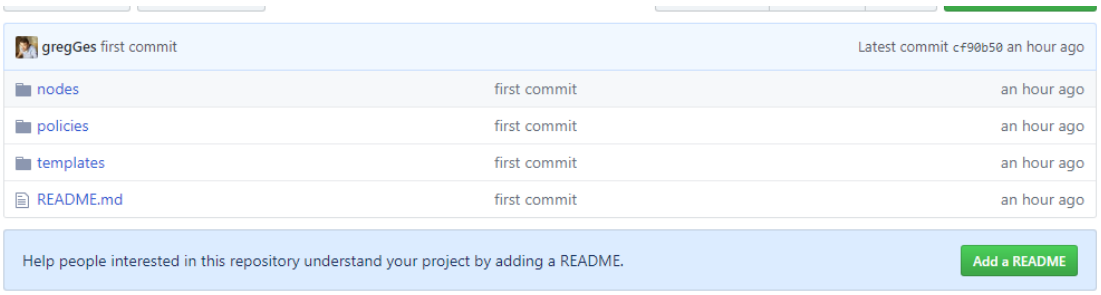

Figure 14, Structure of the COLA repository in github

The directory **nodes** contain one file (**custom_types.yaml**) where the all the custom node types are declared alongside related custom types (capabilities). Table 8 recapitulates all the custom node types declared in file **custom_types.yaml**.

node_types	Derived From	Description
tosca.node.MiCADO.Compute	Tosca.nodes.Compute (Normative TOSCA type)	Abstraction for the different types of virtual machines
tosca.node.MiCADO.Occopus.CloudSigma.Compute	tosca.nodes.MiCADO.Compute	Describes the VirtualMachine provided by the CloudSigma Cloud Provider
tosca.node.MiCADO.Occopus.EC2.Compute	tosca.nodes.MiCADO.Compute	Describes the VirtualMachine provided by the EC2 Cloud Provider
tosca.node.MiCADO.Occopus.CloudBroker.Compute	tosca.nodes.MiCADO.Compute	Describes the VirtualMachine provided by the CloudBroker Cloud Provider
tosca.node.MiCADO.Occopus.Nova.Compute	tosca.nodes.MiCADO.Compute	Describes the VirtualMachine provided by the Nova Cloud Provider

Table 8, Custom Defined Node Types

Table 9 recapitulates all the custom capabilities types declared in file custom_types.yaml.

capability_types	Derived From	Description
tosca.capabilities.MiCADO.Container.Docker	tosca.capabilities.Container.Docker (Non Normative TOSCA type but specified in TOSCA)	Defines the generic capabilities of an host
tosca.capabilities.MiCADO.Occopus.EC2.Cloud:	tosca.capabilities.Container	Defines the specific capabilities of the EC2 Cloud
tosca.capabilities.MiCADO.Occopus.CloudSigma.Cloud:	tosca.capabilities.Container	Defines the specific capabilities of the CloudSigma Cloud
tosca.capabilities.MiCADO.Occopus.Nova.Cloud:	tosca.capabilities.Container	Defines the specific capabilities of the Nova Cloud
tosca.capabilities.MiCADO.Occopus.CloudBroker.Cloud:	tosca.capabilities.Container	Defines the specific capabilities of the CloudBroker Cloud

Table 9, Custom Defined Capabilities Types

Table 10 recapitulates all the custom capabilities types declared in file custom_types.yaml.

data_types	Derived From	Description
tosca.datatypes.MiCADO.Occopus.Cloud	tosca.datatypes.Root (Normative TOSCA type)	Defines the generic datatype for the Occopus Cloud Node Type

Table 10, Custom Defined Data Types

The directory policies contains several sub-directories that contain the policy types hierarchy, the policies are recapitulated in Table 11.

Policies_types	Derived From	
tosca.policies.Execution	Tosca.policies.root	Generic execution policy. Added to the main TOSCA policies types.
tosca.policies.Execution.Schedule	tosca.policies.Execution	Executes the application following a cron job like syntax
tosca.policies.Placement.Requirement.Connection:	tosca.policies.Placement	Defines the inbound connection that must be allowed
tosca.policies.Placement.Requirement.Location:	tosca.policies.Placement	Defines the geographical location of the Virtual Machines
tosca.policies.Scaling.Performance.Consumption:	tosca.policies.Scaling	Scales up or down based on CPU Consumption
tosca.policies.Scaling.Performance.Completion:	tosca.policies.Scaling	Scales up or down depending on the expected completion time of the jobs
tosca.policies.Scaling.Performance.Completion.Job:	tosca.policies.Scaling.Performance.Completion	Scales up or down depending on the expected job completion time within a defined cut-off budget.

Table 11, Custom Defined Policy Types

Each policy is defined in different files under the policy directory as listed in

Policies_types	file
tosca.policies.Execution	/policies/execution/tosca_policy_execution
tosca.policies.Execution.Schedule	policies/execution/tosca_policy_execution_schedule
tosca.policies.Placement.Requirement.Connection:	policies/placement/reuirement/connection/**tosca_policy_deployment_placement_requirement_connection.yaml**
tosca.policies.Placement.Requirement.Location:	policies/placement/requirement/location/**tosca_policy_Placement_Requirement_location.yaml**
tosca.policies.Scaling.Performance.Consumption:	policies/scalability/consumption/**tosca_policy_scalability_comsumption.yaml**
tosca.policies.Scaling.Performance.Completion:	policies/scalability/performance/completion/**tosca_policy_scalability_performance_completion.yaml**
tosca.policies.Scaling.Performance.Completion.Job:	policies/scalability/performance/completion/**tosca_policy_scalability_performance_completion_job.yaml**

Table 12, Files in which the custom policies are declared

The last directory, templates, contains the Topology Templates of all the Use Cases. Table 13 recapitulates where the Topology Templates are saved. The directory contains an extra file (blank_topology.yaml) that can be used as a starting point by Application Developers to follow the steps

Topology Template	**Use Case**	**File**
Outlandish	Use Case 1	outlandish.yaml
Repast	Use Case 2	repast.yaml
Inycom	Use Case 3	inycom.yaml
Data Avenue	Extra Use Case	dataavenue.yaml
Empty Topology	NA	blank_topology.yaml

Table 13, Files in which the topologies templates are defined

11. Structure of the Topology Template

Each Topology Template follows a common structure which we illustrate in this section using the example of Use Case 3 – Inycom.

11.1 Declaration of the TOSCA Version

Each Topology Template starts with the declaration of the TOSCA Version.

```
## declaration of the tosca version
tosca_definitions_version: tosca_simple_yaml_1_0
```

11.2 Import Section

The import section defines which TOSCA files are needed for the Topology Template.

```
imports:
  - ../nodes/custom_types.yaml
  - ../policies/execution/tosca_policy_execution_schedule.yaml
-   ../policies/placement/requirement/connection/tosca_policy_deployment_placement_requirement_connection.yaml
-   ../policies/scalability/performance/completion/tosca_policy_scalability_performance_completion.yaml
  - ../policies/scalability/consumption/tosca_policy_scalability_comsumption.yaml
```

11.3 Location of the Docker Images

This sections defines the URL of the main repository of the Docker Images. User can add private repositories after the main hub.

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

11.4 Start of the main topology section

This line marks the start of the main section of the topology.

```
topology_template:
```

11.5 Input Section

The input section starts here and declares all the values that may be set in the topology template. They are grouped together to make it easier to edit the file. The input section defines the values through the default field. The full capability of the input section may be used in a future GUI to let different user profiles override the default values.

```
## The input section starts here. This is where all the values
## that may be set in the topology template are grouped together.
inputs:
  docker_image:
    type: string
    description: Docker image to run
    required: yes
    default: "magician"
  port_exposed:
    type: integer
    description: port_exposed
    required: yes
    default: 8080
  host_cpu:
    type: integer
    description: cpu of the host
    required: yes
    default: 2
  host_version:
    type: version
    description: version of host
    required: yes
    default: 16.0
  url_list:
    type: list
    description: list of url
    default: ["http://url_for_solr", "http://url_for_tomecat", "http://url_for_webpage"]
    entry_schema:
      type: string
  host_mem:
    type: scalar-unit.size
    description: host mem capacity
    required: yes
    default: 4 GB
  host_disk:
    type: scalar-unit.size
    description: host disk capacity
    required: yes
    default: 50 GB
  libdrive_id:
    type: string
    description: id of the instance image to launch
    required: yes
    default: "some id"
```

```
  max_cpu_consumption:
    type: float
    description: max cpu consumption
    required: yes
    default: 0.8
  min_cpu_consumption:
    type: float
    description: min cpu consumption
    required: yes
    default: 0.2
  scale_up_max_time:
    type: scalar-unit.time
    description: max time of scale scale up
    required: yes
    default: 5 m
  scale_down_max_time:
    type: scalar-unit.time
    description: max time of scale scale down
    required: yes
    default: 5 m
  location:
    type: string
    description: location to launch the instance
    required: yes
    default: Europe
```

11.6 Node Template Section

The node template section declares all the nodes and relationships that compose the topology.

```
## The is the node template section. Where the node that compose
  ## the topology and their relationships are specified
  node_templates:

    ## This specifies the Docker Image node type and its parameters
    Inycom:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      properties:
        privileged: true
        exposed_port: { get_input: port_exposed }

      ## This is the artefact of the Docker Image node type which represents
      ## the Docker Image in either the common or personal docker hub (specified repository)
      artifacts:
        image:
          type: tosca.artifacts.Deployment.Image.Container.Docker
          file: { get_input: docker_image }
          repository: docker_hub

      ## This specifies the requirements to select the Virtual Machine on which the
```

```
    ## Docker Image will be executed.
    requirements:
      - host:
          node: VM
          relationship: tosca.relationships.HostedOn


  ## This specifies the Virtual Machine Image node type and its parameters
  VM:
    type: tosca.nodes.MiCADO.Occopus.CloudSigma.Compute

    ## This specifies the properties of the Virtual Machine Image node type
    properties:
      cloud:
        interface_cloud: cloudsigma
        endpoint_cloud: https://zrh.cloudsigma.com/api/2.0

    ## This specifies the capabilities of the Virtual Machine Image node type which
    ## are matched against the requirements of the Docker Image type
    capabilities:
      host:
        properties:
          num_cpus: { get_input: host_cpu }
          disk_size: { get_input: host_disk }
          mem_size: { get_input: host_mem }
          libdrive_id: { get_input: libdrive_id }
```

11.7 Output Section

The output section declares the value that will be returned by the TOSCA submitter upon successful deployment of the topology.

```
## This specifies the output that are defined by the attribute of the Output
  outputs:
    ip_address:
      value: { get_attribute: [ Inycom, ip_address ]}
    port:
      value: { get_attribute: [ Inycom, port ]}
```

11.8 Policies Section

The policies section declares the policies that are applied to the different nodes of this topology template.

```
## This specifies the policies that are applied to the different nodes of this topology template
  policies:
    - scalability:
```

```
      type: tosca.policies.Scaling.Performance.Consumption
      targets: [ Inycom ]
      properties:
        stage: Execution
        priority: 100
        trigger_1_ID: estimate_completion_time
        trigger_1_Namespace: prometheus
        max_cpu_consumption: { get_input: max_cpu_consumption }
        scale_up_max_time: { get_input: scale_up_max_time }
        min_cpu_consumption: { get_input: min_cpu_consumption }
        scale_down_max_time: { get_input: scale_down_max_time }

    - deployment_connection:
      type: tosca.policies.Placement.Requirement.Connection
      targets: [ Inycom ]
      properties:
        stage: Deployment
        properties: 100
        url: { get_input: url_list }

    - deployment_location:
      type: tosca.policies.Placement.Requirement.Location
      targets: [ VM ]
      properties:
        stage: Deployment
        priority: 100
        trigger_1_Namespace: prometheus
        accepted_domaine: { get_input: location }
```

12. Conclusions

This deliverable details the implementation of the ADTs for four Use Cases that have been selected as the first applications to be deployed in the MiCADO infrastructure. The development of the code has highlighted some needed changes in the abstract design that was first proposed at the start of the project.

The ADT structure has been adapted to contain two nodes: one that specifies the Docker Image that contains the application and another that specifies the criteria with which the Virtual Machine will be selected.

To help Application Developers in the process of entering the values of already created templates, an input section contains all the values that are more likely to be over-ridden. The policies originally designed have also been adapted; one (the Resource-Based Placement Policy) has been implemented directly into the node type. Other policies (such as the Performance-Based Scalability Policy) have been substituted as they are not currently supported by the MiCADO infrastructure.
The hierarchy of the policy types that have been used for the implementation of these use cases is a sub-set of the policies that were part of the original design. In one case (the definition of the hardware characteristics of the resource), the Capabilities/Requirements structure of TOSCA proved to be more apt that the definition of a specific policy. We envisage that further implementations of these Use Cases and the introduction of the new scenarios introduced in D8.1 will require the introduction of some of the policies initially envisaged in D5.2 and D5.3.

Finally, to help in creating ADTs for new applications, the code has been structured in sub-components (nodes, policies and templates) which are stored in separate sections of a github repository. A volunteer-based effort to create a Graphical User Interface inspired by Winery[13] will start in January. Should this interface be completed, it will allow user to override the default values of the input sections without having to directly edit the code. Such interface, will also offer a Drag&Drop workbench for a User-Friendly creation of Template Topologies and the association to their relative policies.

13. References

[1] "DataAvenue Project." [Online]. Available: https://data-avenue.eu/.

[2] A. Hajnal, Z. Farkas, and P. Kacsuk, "Data Avenue: Remote Storage Resource Management in WS-PGRADE/gUSE," in *2014 6th International Workshop on Science Gateways*, 2014, pp. 1–5.

[3] "Git Hub Repository Web Page." .

[4] H. Visti, T. Kiss, G. Terstyanszky, G. Gesmier, and S. Winter, "MiCADO – Towards a microservice-based cloud application-level dynamic orchestrator," Jan. 2016.

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

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

[7] "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].

[8] "Welcome - Occopus." [Online]. Available: http://occopus.lpds.sztaki.hu/en_GB/. [Accessed: 29-Mar-2017].

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

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

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

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

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

14. Appendix A – Outlandish Topology Template

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - ../nodes/custom_types.yaml
  - ../policies/execution/tosca_policy_execution_schedule.yaml
  -
../policies/placement/requirement/connection/tosca_policy_deployment_placement_requirem
ent_connection.yaml
  -
../policies/scalability/performance/completion/tosca_policy_scalability_performance_completi
on.yaml
  - ../policies/scalability/consumption/tosca_policy_scalability_comsumption.yaml

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

topology_template:
  inputs:
    docker_image_afa:
      type: string
      description: Docker image to run
      required: yes
      default: "image_to_use"

    docker_image_cache_service:
      type: string
      description: Docker image to run
      required: yes
      default: "image_to_use"

    ENVIR_A:
      type: string
      description: ENVIR_A string
      default: "some environment_variables"

    ENVIR_B:
      type: string
      description: ENVIR_B string
      default: "some environment_variables"

    ENVIR_C:
      type: string
      description: ENVIR_C string
      default: "some environment_variables"

    url_list_caching:
      type: list
      description: list of url
      default:          ["http://external_db:port/",          "          http://wordpress_db:port/",
```

```
"http://box_office_db:port/"]
      entry_schema:
        type: string
    url_list_afa:
      type: list
      description: list of url for afa component to connect to
      default: ["http://wordpress_db:port/","http://box_office_db:port/", "http://proxy_url:port/"]
      entry_schema:
        type: string

    port_exposed:
      type: integer
      description: port_exposed
      required: yes
      default: 8080
    host_mem:
      type: scalar-unit.size
      description: host mem capacity
      required: yes
      default: 4 GB
    host_disk:
      type: scalar-unit.size
      description: host disk capacity
      required: yes
      default: 50 GB
    libdrive_id:
      type: string
      description: id of the instance image to launch
      required: yes
      default: "some id"
    host_cpu:
      type: integer
      description: cpu of the host
      required: yes
      default: 2
    max_completion_time:
      type: integer
      description: estmated max completion time
      required: yes
      default: 10
    cmd_cron:
      type: string
      description: command to run the cron job
      required: yes
      default: "some command"

    max_cpu_consumption:
      type: float
      description: max cpu consumption
      required: yes
      default: 0.8
```

```
  min_cpu_consumption:
    type: float
    description: min cpu consumption
    required: yes
    default: 0.2

  scale_up_max_time:
    type: scalar-unit.time
    description: max time to scale up
    required: yes
    default: 5 m

  scale_down_max_time:
    type: scalar-unit.time
    description: max time scale down
    required: yes
    default: 5 m



node_templates:
  AFA:
    type: tosca.nodes.MiCADO.Container.Application.Docker
    properties:
      exposed_port: { get_input: port_exposed }
      env:
        ENVIR_A : { get_input: input_a }
        ENVIR_B : { get_input: input_b }
        ENVIR_N : { get_input: input_n }

    artifacts:
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
        file: { get_input: docker_image_afa  }
        repository: docker_hub
    requirements:
      - host:
          node: VM
          relationship: tosca.relationships.HostedOn

  Caching_service:
    type: tosca.nodes.MiCADO.Container.Application.Docker
    properties:
      exposed_port: { get_input: port_exposed }
      env:
        ENVIR_A : { get_input: input_a }
        ENVIR_B : { get_input: input_b }
        ENVIR_N : { get_input: input_n }

    artifacts:
```

```
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
        file: { get_input: docker_image_cache_service }
        repository: docker_hub
    requirements:
      - host:
          node: VM
          relationship: tosca.relationships.HostedOn


  VM:
    type: tosca.nodes.MiCADO.Occopus.CloudSigma.Compute

    properties:
      cloud:
        interface_cloud: cloudsigma
        endpoint_cloud: https://zrh.cloudsigma.com/api/2.0

    capabilities:
      host:
        properties:
          num_cpus: { get_input: host_cpu }
          disk_size: { get_input: host_disk }
          mem_size: { get_input: host_mem }
          libdrive_id: { get_input: libdrive_id }

outputs:
  ip_address:
    value: { get_attribute: [ AFA, ip_address ]}
  port:
    value: { get_attribute: [ AFA, port ]}

policies:
  - consumption:
      type: tosca.policies.Scaling.Performance.Consumption
      targets: [ AFA ]
      properties:
        stage: started
        priority: 100
        trigger_1_ID: estimate_completion_time
        trigger_1_Namespace: prometheus
        max_cpu_consumption: { get_input: max_cpu_consumption }
        scale_up_max_time: { get_input: scale_up_max_time }
        min_cpu_consumption: { get_input: min_cpu_consumption }
        scale_down_max_time: { get_input: scale_down_max_time }

  - execution:
      type: tosca.policies.Execution.Schedule
      targets: [ Caching_service ]
      properties:
        stage: started
```

```
      priority: 100
      cron_cmd: { get_input: cmd_cron }

  - scalability:
      type: tosca.policies.Scaling.Performance.Completion
      targets: [ Caching_service ]
      properties:
        stage: started
        priority: 100
        trigger_1_ID: estimate_completion_time
        trigger_1_Namespace: cache_server
        max_estimation_time: { get_input: max_completion_time }


  - deployment_connection_AFA:
      type: tosca.policies.Placement.Requirement.Connection
      targets: [ Caching_service ]
      properties:
        stage: created
        properties: 100
        url: { get_input: url_list_afa }


  - deployment_connection_Caching:
      type: tosca.policies.Placement.Requirement.Connection
      targets: [ Caching_service ]
      properties:
        stage: created
        properties: 100
        url: { get_input: url_list_caching }
```

15. Appendix B – Repast Topology Template

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - ./nodes/custom_types.yaml

-  ./policies/placement/requirement/connection/tosca_policy_deployment_placement_requirement_connection.yaml

- ./policies/scalability/performance/completion/tosca_policy_scalability_performance_completion_job.yaml

-   ./policies/scalability/performance/completion/tosca_policy_scalability_performance_completion.yaml

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

topology_template:
  inputs:
    ## Input for Container
    docker_image:
      type: string
      description: Docker image to run
      required: yes
      default: "image_to_use"

    url_simultation_db:
      type: string
      description: list of url for afa component to connect to
      default: "http://url_simulation_db:port/"

    ## Input to define the VM
    host_mem:
      type: scalar-unit.size
      description: host mem capacity
      required: yes
      default: 4 GB
    host_disk:
      type: scalar-unit.size
      description: host disk capacity
      required: yes
      default: 50 GB
    libdrive_id:
      type: string
      description: id of the instance image to launch
      required: yes
      default: "some id"
    host_cpu:
```

```
    type: integer
    description: cpu of the host
    required: yes
    default: 2
  ## Input for policies
  estimated_completion_time_one_job:
    type: scalar-unit.time
    description: time for one job to be executed
    required: true
    default: 1 h

  max_estimation_time:
    type: scalar-unit.time
    description: time execution experiment
    required: true
    default: 1 d 2 h
  total_budget:
    type: float
    description: total budget for the experiment
    required: true
    default: 50.0
  cut_off:
    type: float
    description: allowance to go behind total budget
    required: true
    default: 100.0

node_templates:

  Repast:
    type: tosca.nodes.MiCADO.Container.Application.Docker
    properties:
      cmd: "URL_MODEL [ BATCH_PARAMS ]"
    artifacts:
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
        file: { get_input: docker_image }
        repository: docker_hub
    requirements:
      - host:
          node: VM
          relationship: tosca.relationships.HostedOn


  VM:
    type: tosca.nodes.MiCADO.Occopus.CloudSigma.Compute

    properties:
      cloud:
        interface_cloud: cloudsigma
        endpoint_cloud: https://zrh.cloudsigma.com/api/2.0
```

```
    capabilities:
      host:
        properties:
          num_cpus: { get_input: host_cpu }
          disk_size: { get_input: host_disk }
          mem_size: { get_input: host_mem }
          libdrive_id: { get_input: libdrive_id }

policies:
  - scalability:
      type: tosca.policies.Scaling.Performance.Completion.Job
      targets: [ Repast ]
      properties:
        stage: started
        priority: 100
        trigger_1_ID: estimate_completion_time
        trigger_1_Namespace: prometheus
        max_estimation_time: { get_input: max_estimation_time }
        estimated_completion_time_one_job: { get_input: estimated_completion_time_one_job }
        total_budget: { get_input: total_budget }
        cut_off_percentage: { get_input: cut_off }
        params:
          url: { get_input: url_simulation_db }


  - connect:
      type: tosca.policies.Placement.Requirement.Connection
      targets: [ Repast ]
      properties:
        stage: created
        properties: 100
        url: { get_input: url_simultation_db }
```

16. Appendix C – Inycom Topology Template

```
## declaration of the tosca version
tosca_definitions_version: tosca_simple_yaml_1_0


## imports section defines the tosca files that are needed to parsed
imports:
  - ../nodes/custom_types.yaml
  - ../policies/scalability/consumption/tosca_policy_scalability_comsumption.yaml

-   ../policies/placement/requirement/connection/tosca_policy_deployment_placement_requirement_co
nnection.yaml
  - ../policies/placement/requirement/location/tosca_policy_Placement_Requirement_Location.yaml


## Url of the main repository fir the location of the docker Images. The user can specify her/his own
## repository by adding another line with a name and url eg: my_repo: http://my_repo/
repositories:
  docker_hub: https://hub.docker.com/


## the topology template start here
topology_template:

  ## The input section starts here. This is where all the values
  ## that may be set in the topology template are grouped together.
  inputs:
    docker_image:
      type: string
      description: Docker image to run
      required: yes
      default: "magician"
    port_exposed:
      type: integer
      description: port_exposed
      required: yes
      default: 8080
    host_cpu:
      type: integer
      description: cpu of the host
      required: yes
      default: 2
    host_version:
      type: version
      description: version of host
      required: yes
      default: 16.0
    url_list:
      type: list
      description: list of url
```

```
    default: ["http://url_for_solr", "http://url_for_tomecat", "http://url_for_webpage"]
    entry_schema:
      type: string
  host_mem:
    type: scalar-unit.size
    description: host mem capacity
    required: yes
    default: 4 GB
  host_disk:
    type: scalar-unit.size
    description: host disk capacity
    required: yes
    default: 50 GB
  libdrive_id:
    type: string
    description: id of the instance image to launch
    required: yes
    default: "some id"
  max_cpu_consumption:
    type: float
    description: max cpu consumption
    required: yes
    default: 0.8
  min_cpu_consumption:
    type: float
    description: min cpu consumption
    required: yes
    default: 0.2
  scale_up_max_time:
    type: scalar-unit.time
    description: max time of scale scale up
    required: yes
    default: 5 m
  scale_down_max_time:
    type: scalar-unit.time
    description: max time of scale scale down
    required: yes
    default: 5 m
  location:
    type: string
    description: location to launch the instance
    required: yes
    default: Europe


## The is the node template section. Where the node that compose
## the topology and their relationships are specified
node_templates:

  ## This specifies the Docker Image node type and its parameters
  Inycom:
```

```
    type: tosca.nodes.MiCADO.Container.Application.Docker
    properties:
      privileged: true
      exposed_port: { get_input: port_exposed }

    ## This is the artefact of the Docker Image node type which represents
    ## the Docker Image in either the common or personal docker hub (specified repository)
    artifacts:
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
        file: { get_input: docker_image }
        repository: docker_hub

    ## This specifies the requirements to select the Virtual Machine on which the
    ## Docker Image will be executed.
    requirements:
      - host:
          node: VM
          relationship: tosca.relationships.HostedOn


  ## This specifies the Virtual Machine Image node type and its parameters
  VM:
    type: tosca.nodes.MiCADO.Occopus.CloudSigma.Compute

    ## This specifies the properties of the Virtual Machine Image node type
    properties:
      cloud:
        interface_cloud: cloudsigma
        endpoint_cloud: https://zrh.cloudsigma.com/api/2.0

    ## This specifies the capabilities of the Virtual Machine Image node type which
    ## are matched against the requirements of the Docker Image type
    capabilities:
      host:
        properties:
          num_cpus: { get_input: host_cpu }
          disk_size: { get_input: host_disk }
          mem_size: { get_input: host_mem }
          libdrive_id: { get_input: libdrive_id }


## This specifies the output that are defined by the attribute of the Output
outputs:
  ip_address:
    value: { get_attribute: [ Inycom, ip_address ]}
  port:
    value: { get_attribute: [ Inycom, port ]}


## This specifies the policies that are applied to the different nodes of this topology template
```

```
policies:
  - scalability:
      type: tosca.policies.Scaling.Performance.Consumption
      targets: [ Inycom ]
      properties:
        stage: Execution
        priority: 100
        trigger_1_ID: estimate_completion_time
        trigger_1_Namespace: prometheus
        max_cpu_consumption: { get_input: max_cpu_consumption }
        scale_up_max_time: { get_input: scale_up_max_time }
        min_cpu_consumption: { get_input: min_cpu_consumption }
        scale_down_max_time: { get_input: scale_down_max_time }

  - deployment_connection:
      type: tosca.policies.Placement.Requirement.Connection
      targets: [ Inycom ]
      properties:
        stage: Deployment
        properties: 100
        url: { get_input: url_list }

  - deployment_location:
      type: tosca.policies.Placement.Requirement.Location
      targets: [ VM ]
      properties:
        stage: Deployment
        priority: 100
        trigger_1_Namespace: prometheus
        accepted_domaine: { get_input: location }
```

17. Appendix D – DataAvenue Topology Template

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - ../nodes/custom_types.yaml
  - ../policies/scalability/consumption/tosca_policy_scalability_comsumption.yaml


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

topology_template:
  inputs:

    host_mem:
      type: scalar-unit.size
      description: host mem capacity
      required: yes
      default: 4 GB
    host_disk:
      type: scalar-unit.size
      description: host disk capacity
      required: yes
      default: 50 GB
    libdrive_id:
      type: string
      description: id of the instance image to launch
      required: yes
      default: "some id"
    host_cpu:
      type: integer
      description: cpu of the host
      required: yes
      default: 2


    max_cpu_consumption:
      type: float
      description: max cpu consumption
      required: yes
      default: 0.8

    min_cpu_consumption:
      type: float
      description: min cpu consumption
      required: yes
      default: 0.2

    scale_up_max_time:
      type: scalar-unit.time
```

```
    description: max time to scale up
    required: yes
    default: 5 m

  scale_down_max_time:
    type: scalar-unit.time
    description: max time scale down
    required: yes
    default: 5 m

node_templates:
  proxy:
    type: tosca.nodes.MiCADO.Container.Application.Docker
    properties:

    artifacts:
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
        file: { get_input: docker_image }
        repository: docker_hub
    requirements:
      - service:
          node: data_avenue
          relationship: tosca.relationships.ConnectsTo
      - host:
          node: VM
          relationship: tosca.relationships.HostedOn

  data_avenue:
    type: tosca.nodes.MiCADO.Container.Application.Docker
    properties:
    artifacts:
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
        file: { get_input: docker_image }
        repository: docker_hub
    requirements:
      - service:
          node: mongo
          relationship: tosca.relationship.ConnectsTo
      - host:
          node: VM
          relationship: tosca.relationships.HostedOn

  mongo:
    type: tosca.nodes.MiCADO.Container.Application.Docker
    properties:

    artifacts:
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
```

```
        file: { get_input: docker_image }
        repository: docker_hub
    requirements:
      - host:
          node: VM
          relationship: tosca.relationships.HostedOn

  VM:
    type: tosca.nodes.MiCADO.Occopus.CloudSigma.Compute
    properties:
      cloud:
        interface_cloud: cloudsigma
        endpoint_cloud: https://zrh.cloudsigma.com/api/2.0

    capabilities:
      host:
        properties:
          num_cpus: { get_input: host_cpu }
          disk_size: { get_input: host_disk }
          mem_size: { get_input: host_mem }
          libdrive_id: { get_input: libdrive_id }

outputs:
  ip_address:
    value: { get_attribute: [ data_avenue, ip_address ]}
  port:
    value: { get_attribute: [ data_avenue, port ]}

policies:
  - consumption:
      type: tosca.policies.Scaling.Performance.Consumption
      targets: [ data_avenue ]
      properties:
        priority: 100
        trigger_1_ID: estimate_completion_time
        trigger_1_Namespace: prometheus
        max_cpu_consumption: { get_input: max_cpu_consumption }
        scale_up_max_time: { get_input: scale_up_max_time }
        min_cpu_consumption: { get_input: min_cpu_consumption }
        scale_down_max_time: { get_input: scale_down_max_time }
```

18. Appendix E– Custom Types

```
tosca_definitions_version: tosca_simple_yaml_1_0

capability_types:
  tosca.capabilities.MiCADO.Container.Docker:
    derived_from: tosca.capabilities.Container.Docker
    properties:
      num_cpus:
        type: float
        required: false
        constraints:
          - greater_or_equal: 0.0

  tosca.capabilities.MiCADO.Occopus.EC2.Cloud:
    derived_from: tosca.capabilities.Container
    properties:
      region_name:
        type: string
        required: true
      image_id:
        type: string
        required: true
      instance_type:
        type: string
        required: true
      key_name:
        type: string
        required: false
      security_groups_ids:
        type: string
        required: false
      subnet_id:
        type: string
        required: false

  tosca.capabilities.MiCADO.Occopus.CloudSigma.Cloud:
    derived_from: tosca.capabilities.Container
    properties:
      libdrive_id:
        type: string
        required: true
      vnc_password:
        type: string
        required: false
      host_name:
        type: string
        required: false
      public_key_id:
        type: string
        required: false
```

```
    firewall_policy:
      type: string
      required: false
    description:
      type: string
      required: false
      description: overrides the capability type

tosca.capabilities.MiCADO.Occopus.Nova.Cloud:
  derived_from: tosca.capabilities.Container
  properties:
    image_id:
      type: string
      required: true
    flavour_name:
      type: string
      required: true
    tenant_name:
      type: string
      required: false
    project_id:
      type: string
      required: false
    user_domain_name:
      type: string
      required: false
    network_id:
      type: string
      required: false
    server_name:
      type: string
      required: false
    key_name:
      type: string
      required: false
    security_groups:
      type: string
      required: false
    floating_ip:
      type: string
      required: false
    floating_ip_pool:
      type: string
      required: false

tosca.capabilities.MiCADO.Occopus.CloudBroker.Cloud:
  derived_from: tosca.capabilities.Container
  properties:
    deployment_id:
      type: string
      required: true
```

```
      instance_type_id:
        type: string
        required: true
      key_pair_id:
        type: string
        required: false
      opened_port:
        type: string
        required: false



data_types:
  tosca.datatypes.MiCADO.Occopus.Cloud:
    derived_from: tosca.datatypes.Root
    properties:
      interface_cloud:
        type: string
        required: yes
        constraints:
          - valid_values: [ 'ec2', 'nova', 'cloudsigma', 'cloudbroker' ]
      endpoint_cloud:
        type: string
        required: yes
      credentials:
        type: tosca.datatypes.credentials
        required: false



node_types:
  tosca.nodes.MiCADO.Compute:
    derived_from: tosca.nodes.Compute
    properties:
      cloud:
        type: tosca.datatypes.MiCADO.Occopus.Cloud

    capabilities:
      host:
        type: tosca.capabilities.MiCADO.Occopus.Cloud

  tosca.nodes.MiCADO.Occopus.CloudSigma.Compute:
    derived_from: tosca.nodes.MiCADO.Compute
    properties:
      cloud:
        type: tosca.datatypes.MiCADO.Occopus.Cloud
    capabilities:
      host:
        type: tosca.capabilities.MiCADO.Occopus.CloudSigma.Cloud
```

```
tosca.nodes.MiCADO.Occopus.EC2.Compute:
  derived_from: tosca.nodes.MiCADO.Compute
  properties:
    cloud:
      type: tosca.datatypes.MiCADO.Occopus.Cloud

  capabilities:
    host:
      type: tosca.capabilities.MiCADO.Occopus.EC2.Cloud


tosca.nodes.MiCADO.Occopus.CloudBroker.Compute:
  derived_from: tosca.nodes.MiCADO.Compute
  properties:
    cloud:
      type: tosca.datatypes.MiCADO.Occopus.Cloud

  capabilities:
    host:
      type: tosca.capabilities.MiCADO.Occopus.CloudBroker.Cloud

tosca.nodes.MiCADO.Occopus.Nova.Compute:
  derived_from: tosca.nodes.MiCADO.Compute
  properties:
    cloud:
      type: tosca.datatypes.MiCADO.Occopus.Cloud

  capabilities:
    host:
      type: tosca.capabilities.MiCADO.Occopus.Nova.Cloud


## Node Type to describe the container node (it will contains the command
## line to pass to the docker)
tosca.nodes.MiCADO.Container.Application.Docker:
  derived_from: tosca.nodes.Container.Application.Docker
  description: description of main container
  properties:
    cmd:
      type: string
      description: command line to be executed by the container.
      required: false
    exposed_port:
      type: integer
      description: port exposed inside container
      range: [ 32768, 61000 ]
      required: false
    env:
      type: map
      description: map of all the environment variable required.
      required: false
```

```
      entry_schema:
        type: string
    constraints:
      required: false
      type: list
      entry_schema:
        type: list
        entry_schema:
          type: string
    labels:
      required: false
      type: map
      entry_schema:
        type: string
    privileged:
      type: boolean
      required: false
      default: false
    force_pull_image:
      type: boolean
      required: false
      default: false
    other_options:
      type: map
      required: false
      entry_schema:
        type: string
  attributes:
    ip_address:
      type: string
    port:
      type: integer
  capabilities:
    service:
      type: tosca.capabilities.Endpoint

  requirements:
    - service:
        capability: tosca.capabilities.Endpoint
        node: tosca.nodes.MiCADO.Compute
        relationship: tosca.relationships.ConnectsTo
    - host:
        capability: tosca.capability.Container
        node: tosca.nodes.MiCADO.Compute
        relationship: tosca.relationships.HostedOn
```

19. Appendix F – TOSCA Policy Execution

```
tosca_definitions_version: tosca_simple_yaml_1_0

policy_types:
  tosca.policies.Execution:
    derived_from: tosca.policies.Root
```

20. Appendix G – TOSCA Policy Execution Schedule

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - ./tosca_policy_execution.yaml
policy_types:
  tosca.policies.Execution.Schedule:
    derived_from: tosca.policies.Execution
    description: executes the application following a cron job like syntax
    properties:
      stage:
        type: string
        description: the stage that will be affected by the policy
        default: started
      priority:
        type: integer
        description: the priority with which the policy will be executed
        default: 100
      cron_cmd:
        type: string
        description: a cron-job like cmd line
```

21. Appendix H – TOSCA Deployment Connection

```
tosca_definitions_version: tosca_simple_yaml_1_0

policy_types:
  tosca.policies.Placement.Requirement.Connection:
    derived_from: tosca.policies.Placement
    description: defines the connection requirements for the container
    properties:
      stage:
        type: string
        description: the stage that will be affected by the policy
        default: created
      priority:
        type: integer
        description: the priority with which the policy will be executed
        default: 100
      url:
        type: list
        description: address or list of addresses that needs to be reachable by the container
        entry_schema:
          type: string
```

22. Appendix I – TOSCA Deployment Location

```
tosca_definitions_version: tosca_simple_yaml_1_0

policy_types:
  tosca.policies.Placement.Requirement.Location:
    derived_from: tosca.policies.Placement
    description: defines the geographical location where the container can be deployed
    properties:
      stage:
        type: string
        description: the stage that will be affected by the policy
        default: Created
      priority:
        type: integer
        description: the priority with which the policy will be executed
        default: 100
      Trigger_1_Namespace:
        type: string
        description: the service that will return the geographical location of the service
      accepted_domaine:
        type: list
        description: list of the acceptable geographical locations
        schema_entry:
          type: string
```

23. Appendix J – TOSCA Scalability Consumption

```
tosca_definitions_version: tosca_simple_yaml_1_0


policy_types:
  tosca.policies.Scaling.Performance.Consumption:
    derived_from: tosca.policies.Scaling
    description: spins one instance when CPU Consumption is above the threshold
    properties:
      stage:
        type: string
        description: the stage that will be affected by the policy
        default: started
      priority:
        type: integer
        description: the priority with which the policy will be exectued
        default: 100
      trigger_1_ID:
        type: string
        description: defines the trigger (thesholdt that will spin up/down the instances)
        default: estimate_completion_time
      trigger_1_Namespace:
        type: string
        description: defines the namespace of the service that is monitoring th number of queries
that must be executed
      max_cpu_consumption:
        type: integer
        description: defines the maximum time when the max cpu threshold must be exceeded
for the new instance to be deployed
      scale_up_max_time:
        type: scalar-unit.time
        description: defines the maximum time when the max cpu threshold must be exceeded
for the new instance to be deployed
      min_cpu_consumption:
        type: integer
        description: defines the minimum cpu consumption below which the new instance will be
deployed
      scale_down_max_time:
        type: scalar-unit.time
        description: defines the maximum time whe nthe min cpu threshold must be exceeded
for the instance to be undeployed.
```

24. Appendix K – TOSCA Scalability Performance Completion

```
tosca_definitions_version: tosca_simple_yaml_1_0


policy_types:
  tosca.policies.Scaling.Performance.Completion:
    derived_from: tosca.policies.Scaling
    description: scale up or down depending on the expected completion time
    properties:
      stage:
        type: string
        description: the stage that will be affected by the policy
        default: started
        required: false
      priority:
        type: integer
        description: the priority with which the policy will be exectued
        default: 100
        required: false
      trigger_1_ID:
        type: string
        description: defines the trigger (theshold that will spin up/down the instances)
        default: estimate_completion_time
        required: false
      trigger_1_Namespace:
        type: string
        description: defines the namespace of the service that is monitoring the connections
        required: false
      max_estimation_time:
        type: scalar-unit.time
        required: false
        description: defines the latest time of completion above which the new instance will be deployed
      min_estimation_time:
        type: scalar-unit.time
        required: false
        description: defines the earliest time of completion under which the new instance will be
undeployed
```

25. Appendix L – TOSCA Scalability Performance Completion Job

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:

-  ./policies/scalability/performance/completion/tosca_policy_scalability_performance_completion.yaml
policy_types:
  tosca.policies.Scaling.Performance.Completion.Job:
    derived_from: tosca.policies.Scaling.Performance.Completion
    description: scales up and down the instances of the jobs to meet the defined deadline within a cost maximum defines as budget times cut-off percentage
    properties:
      estimated_completion_time_one_job:
        type: scalar-unit.time
        description: the estimated time for one job
        required: false
      total_budget:
        type: float
        description: the total_budget for the experiment
        required: false
      cut_off_percentage:
        type: float
        description: the percentage allowance to go over the total_budget
        required: false
      params:
        type: list
        description: list of params for feeder
        required: false
        entry_schema:
          type: string
```