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:

D5.5 Second Set of Templates and Services of Use Cases

Due date of deliverable: 31/07/2019 Actual submission date: 30/09/2019

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	01/08/19	Dr. Gabriele Pierantoni
Reviewed:	Kovács Bálint	07/08/19	Kovács Bálint
Approved:	Tamas Kiss	30/09/19	Tamas Kiss

Table 1, Status Change History

Version	Date	Pages	Author	Modification
V0.1	08/07/19	ALL	G. Pierantoni	Skeleton
V0.2	29/07/19	ALL	G. Pierantoni, James DesLauriers	Introduction and additional ADT details
V0.3	30/07/19	ALL	G. Pierantoni, James DesLauriers	Most sections completed
V0.4	1/08/19	ALL	James DesLauriers	Final additions before sending for review
V0.5	11/9/19	ALL	G. Pierantoni	Addressing first batch of Tamas' feedback
V0.91	26/09/19	All	G. Pierantoni	Addressing second batch of Tamas' feedback
V0.92	29/09/19	All	G. Pierantoni	Addressing second batch of Tamas' feedback

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
SOA	Service Oriented Architecture
TOSCA	Topology Orchestration Specification for Cloud Application
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

Table 3, Glossary

5. Introduction

COLA Description of Action (DoA) specifies **Deliverable D5.5** as follows: **"*This deliverable will upload the upgraded first set of descriptions and templates. It will publish further templates and the service description to be used in use-cases and will write a report about these templates and services*.**"

This deliverable describes how the Application Description Templates (ADTs) implementation for the three Use Cases proposed by the COLA Industrial and Academic partners have evolved to cope with the requirements of the users. Furthermore, the document offers a "semi-final" version of the ADTs (as they stood at the time of the compilation of the Deliverable as some minor changes are still envisaged and designed to improve flexibility and ease of use by the users). This deliverable explains the design changes that have been adopted throughout the project and describes the generic structure of the implementation of the Application Description Templates.

The deliverable also describes the structure of the GitHub repository[1] that has been created to contain the TOSCA code and provides the commented code of one of the ADTs to offer both a clear view on its structure and the related implementation details.

This Deliverable is the fifth 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 D5.5 is structured as follows:

- **Section 5 – Introduction** – Offers an introduction to the Deliverable introducing general concepts and its relevance within the COLA project.
- **Section 6 – Relationship with other packages and deliverables** – Further details the topics introduced in **Section 5** by describing the mutual dependencies among this Deliverable with the other most relevant Project Deliverables.
- **Section 7 – The Application Description Template in COLA** – Defines the generic structure of the Application Description Templates and how it relates to the various components of the MiCADO infrastructure.
- **Section 8 – Examples of Application Description Templates used for real applications in COLA** – Describes the implementation of three Use Cases described in D8.4:
    - **Use Case 1: Social media data Analytics for Public Sector Organisations** developed by Inycom and SARGA
    - **Use Case 2: Bursting onto the Cloud from SakerGrid** – developed by Brunel University and Saker Solutions,
    - **Use Case 3: Scalable hosting, testing and automation for Small to Medium-sized Enterprises (SMEs) and public sector organisations**, developed by Outlandish and The Audience Agency,
    - **Section 8.4 – Test & Demonstrator Applications** describes applications that have been developed to test particular aspects of the system to support the development of the three main use cases.

[1] https://github.com/micado-scale/tosca/

  - **Section 8.5 – Proof-of-concept Feasibility Study Prototypes** describes the ADT prototypes developed to describe the proof of concept prototypes proposed by WP8 in D8.4. This section links to the location of prototype ADTs in the GitHub repository and where relevant illustrates how certain prototypes can be built using existing ADTs as a base.
- **Section 9** – Describes the structure of the GitHub repository that contains the ADTs, it further describes each of its directories and lists the declared types.
- **Section 10** – Describes the strengths and weaknesses of the proposed approach.
- **Section 11** – Describes the conceptual and practical improvements of the latest version of the ADT structures.
- **Section 12 –** Concludes the Deliverable offering some remarks on the status of WP5 and future work.
- **Sections 13 –** Appendices containing full ADTs for the three primary use cases.

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 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 describe the concepts highlighted in the previous paragraph.

Deliverable 5.5 is published by Work Package 5 – Application Definition Templates. This work package has previously published four Deliverables:

- **D5.1** "Analysis of existing Application Description Approaches" offers a state-of-the-art overview of the application description and execution. D5.1 details the reasons behind the choice of adopting TOSCA as the language specification for COLA ADTs.
- **D5.2** "Specification of the Application Description Concept" describes the COLA's proposed approach to the problem: a three-layered Application Description Template based on the language specification which also defines policies at each of its layers.
- **D5.3** "Integration of the Templates with the Selected Application Description Approach" describes how the concept highlighted in D5.2 are applied to define the three Use Cases defined in D8.1.
- **D5.4** "First Set of Templates and Services of Use Cases" presents the first version of the ADTs that has been developed for the three large scale demonstrators and also implements and describes the repository where the ADT profiles are stored. At the moment being this is a git-based repository and we use metadata embedded in the code to do queries. It is currently under discussion if a different approach is needed to be of greater help to the users.

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". From a more abstract architectural point of view, we can see as WP5 and the ADT Submitter developed in WP6 as both abstraction and interoperability layers that allow users to write Application Description and Policies in a generic language that is then adapted to the various technologies employed by WP6 and WP4.

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 D5.5 is also closely linked to Work Package 8 – "SME and public sector use-case pilots and demonstrators", particularly with:

- **D8.1** – "Business and Technical Requirements of COLA Use Cases";
- **D8.2** – "Customisation and further development of software applications" that provided further details on how to ameliorate the ADT that describes the application;
- **D8.3** – "COLA near to operational level pilots and demonstrators"
- **D8.4** - "Proof of Concept Feasibility Studies" described as "This deliverable will report on the 20 proof-of concept feasibility studies. The report will describe users behind these use cases and the way they could use MiCADO services, with illustrative examples as appropriate. The report will also contain implementation and commercialisation roadmap for these use-cases" that allowed WP5 to test and demonstrate how the prototypical ADTs and their components can be re-used for a variety of different applications.

WP5's interactions with WP8 ensure that the Use Cases of Work Package 8 can be supported by the Application Description Templates.

7. The Evolution ADTs in COLA

The user facing interface of MiCADO is an Application Description Template (ADT). The structure of an ADT has evolved significantly throughout the duration of the project and such changes are described in the various deliverables produced by WP5:

- **D5.2** describes in various sections the abstract structure of an ADT.
- **D5.3 (Section 7 - Application Descriptions within COLA and the three Use Cases)** describes how the design of the ADT meets the requirements of the three use cases described in D8.1
- **D5.3 (Section 8 - Use Case Overview)** describes the main concepts (Application Topologies and Policies) for each of the ADTs.
- **D5.3 (Annexes)** contains all the code described in Section 7 and Section 8 of that document.

- **D5.4 (Section 7 - The Application Description Templates in COLA)** details the first releases of the ADTs developed for each of the three Use Cases initially described in D8.1.

In this current deliverable (D5.5), we have dedicated a specific section (Section 11) to describe the theoretical and structural improvement we have introduced in the ADTs. We describe how abstraction and inheritance can be used to improve an ADT.

At this point in the evolution of an ADT, it describes the application in full, including the cloud infrastructure which supports it, the software containers which make up the application proper, and the policies which regulate aspects of the application's orchestration and lifecycle such as scalability and security. The ADT is written to the TOSCA[2] specification - the Topology and Orchestration Specification for Cloud Applications - an OASIS Standard for describing applications in the cloud. TOSCA aims to avoid vendor lock-in and ensure simplified portability and management of applications and infrastructure across different public, private and hybrid clouds. TOSCA Simple Profile in YAML expresses the original XML-based TOSCA in the more readable data serialisation standard, YAML[3] (YAML Ain't Markup Language). TOSCA works strongly with the concept of nodes. A node is any component in the cloud topology of an application which includes virtual machines, containers, volumes, software and networks. The ADT in MiCADO was originally designed based on TOSCA Simple Profile in YAML v1.0[4], but also takes cues from the more recent TOSCA Simple Profile in YAML v1.1 & v1.2.

TOSCA, and by extension, MiCADO ADTs, broadly divide templates into two general sections - topology and policies. For an ADT, the cloud topology can be further categorised into two sections: one covering the cloud infrastructure which supports the application, and another for the application components themselves. The policies section can also be subdivided into policies and security-specific policies. Each of these subsections describes the desired state or orchestration output of a specific core component of the MiCADO framework. The cloud infrastructure section describes the virtual machines to be provisioned by the cloud orchestrator. The section on application components describes the container (or containers, in a service-oriented architecture (SOA) or microservices architecture) of the application to be managed by a container orchestrator. On the policy side, the policy descriptions of scalability, alerting, monitoring and the like are enforced by a general policy keeper component, while security policy descriptions are enforced by a security-specific component. These sections of an ADT are described in further details below.

7.1 Cloud Infrastructure

The cloud infrastructure of an application in MiCADO refers primarily to the virtual machines provisioned in the cloud. These virtual machines provide the compute resources for the application itself. The preferred method for automating the provisioning of virtual machines is through interfacing with the API of a cloud service provider (CSP) where parameters in the API request define the desired virtual machine. In an ADT, these parameters have been mapped to the properties of a TOSCA Compute node representing a virtual machine in MiCADO. There is no well-established standard for describing virtual machines across

[2] https://www.oasis-open.org/committees/tosca/
[3] https://yaml.org/
[4] http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html

different CSPs and since MiCADO is a cloud-agnostic solution, these compute node properties are specific to a given CSP. This means a different node for each supported cloud, but abstraction and inheritance within TOSCA greatly facilitate this. A custom parent type for MiCADO compute nodes is created for each CSP which defines those parameters as TOSCA properties. This parent node type specifies defaults, constraints and requirements for these properties and then enforces them when the ADT is submitted.

Though the format and syntax of parameters passed to cloud APIs may vary, MiCADO compute nodes generally express the following properties which serve to describe the desired virtual machine:

- Virtual Machine Image
- Instance Size (CPU & Memory)
- Port configuration
- Public key (for SSH access)

In fact, MiCADO does not interface directly with a cloud service provider's API. A cloud orchestration tool (sometimes infrastructure-as-code) is expected to provide communication with a variety of cloud service providers. Because of the modular design of MiCADO, one cloud orchestration tool can be swapped with another to support a new set of cloud providers. Regardless of the tool, a cloud orchestrator generally requires its own parameters to be passed in to configure or manage its operation. These parameters, however, are not specific to the CSP or virtual machine, and so do not belong within the properties section of our custom MiCADO compute node types. Previously, to express these cloud orchestrator parameters, individual custom node types were created for each VM as orchestrated by each cloud orchestrator. This led to a very repetitive structure of TOSCA types, where we were not benefitting from the inheritance or abstraction offered by TOSCA. Take the following as an example:

- tosca.nodes.MiCADO.Occopus.CloudSigma.Compute
- tosca.nodes.MiCADO.Terraform.CloudSigma.Compute
- tosca.nodes.MiCADO.Occopus.EC2.Compute
- tosca.nodes.MiCADO.Terraform.EC2.Compute

To counteract this repetitive structure, the TOSCA interface structure is used. The TOSCA Standard Interface is used to set input parameters or environment variables for custom scripts or code snippets which then manage lifecycle operations of the node. In the case of cloud orchestration these operations might be *create virtual machine* or *stop virtual machine*. Since MiCADO uses a cloud orchestrator which handles these operations natively, custom scripts are not required. The ability to set specific parameters and pass them to the cloud orchestrator, however, is very useful. MiCADO defines a custom TOSCA interface for each supported cloud orchestration tool and passes parameters to the orchestrator though this structure. Using the example above, we can re-use the same CloudSigma Compute type, and simply change the orchestrator attached to it:

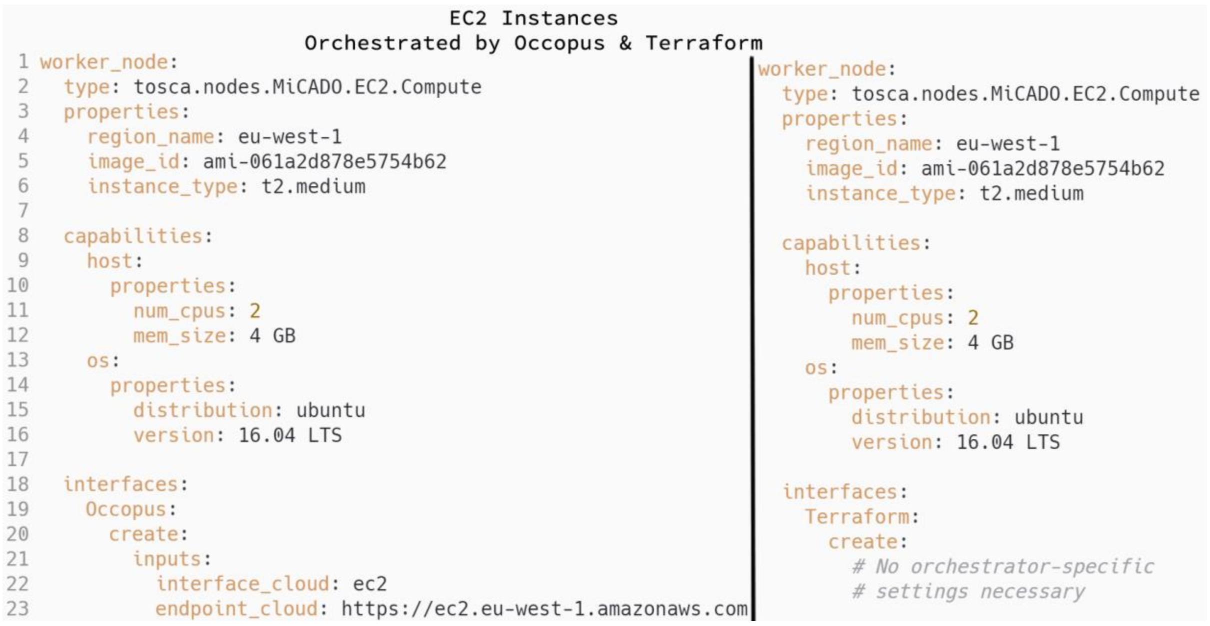
- tosca.nodes.MiCADO.CloudSigma.Compute
  - interfaces:
    - Occopus

  tosca.nodes.MiCADO.CloudSigma.Compute
  - interfaces:
    - Terraform

At the time of writing, MiCADO (v0.7.3) implements Occopus as cloud orchestrator, which supports the following clouds:

- CloudSigma
- Openstack Nova
- CloudBroker
- AWS EC2

For each supported cloud, a custom MiCADO compute node type has been created. Each of these node types supports a custom created TOSCA interface which defines the available parameters which can be passed to Occopus on submission of the ADT. Existing custom compute node types can be modified and new custom compute nodes added, at any time and can even be included inline in an ADT. If the cloud orchestrator is changed, a new custom interface can be created and attached to the MiCADO compute nodes it supports. The potential for supporting more CSPs is limited only by the cloud orchestration component, and ADTs make it simple to support changes in this component. Figure 1 shows the same EC2 instance orchestrated by two different cloud orchestrators.



EC2 Instances
Orchestrated by Occopus & Terraform

```
worker_node:
  type: tosca.nodes.MiCADO.EC2.Compute
  properties:
    region_name: eu-west-1
    image_id: ami-061a2d878e5754b62
    instance_type: t2.medium

  capabilities:
    host:
      properties:
        num_cpus: 2
        mem_size: 4 GB
    os:
      properties:
        distribution: ubuntu
        version: 16.04 LTS

  interfaces:
    Occopus:
      create:
        inputs:
          interface_cloud: ec2
          endpoint_cloud: https://ec2.eu-west-1.amazonaws.com
```

```
worker_node:
  type: tosca.nodes.MiCADO.EC2.Compute
  properties:
    region_name: eu-west-1
    image_id: ami-061a2d878e5754b62
    instance_type: t2.medium

  capabilities:
    host:
      properties:
        num_cpus: 2
        mem_size: 4 GB
    os:
      properties:
        distribution: ubuntu
        version: 16.04 LTS

  interfaces:
    Terraform:
      create:
        # No orchestrator-specific
        # settings necessary
```

Figure 1, Example of EC2 Virtual Machines, provisioned by different cloud orchestrators

7.2 Application Infrastructure

Applications in MiCADO are generally assumed to be in containers. While so-called “virtual machine only” proof of concepts has been realised in MiCADO, many of the scaling and scheduling orchestration benefits conferred by MiCADO are only available when applications are in containers. Due to the popularity and widespread adoption of Docker as a container runtime, there exists a mostly standard set of options which are used to describe, build and run a container. Containers must be built before being used in MiCADO, though MiCADO does support managing the options of a container at runtime. To do so, these options have been mapped to the properties of a custom MiCADO node type which extends a non-normative TOSCA node type for Docker containers. Many properties are supported, but some of the most common are:

- Container image
- Container name
- Container command
- Container arguments

- Port configuration
- Resource limitations
- Environment variables
- Labels

As with cloud orchestration, another layer sits between MiCADO and Docker. MiCADO does not interact directly with Docker, but rather uses an orchestrator to schedule and manage the desired containers on top of the cloud infrastructure that is also described in the ADT. At the time of writing (MiCADO v0.7.3), Kubernetes provides the container orchestration capabilities of MiCADO. Again, because of the modularity of MiCADO, the container orchestrator can be replaced if the need arises. In fact, an earlier implementation of MiCADO saw Docker Swarm as the container orchestrator, and the move to Kubernetes was made simpler by taking an approach similar to what was described under the cloud infrastructure section. Custom TOSCA interfaces were created for each supported container orchestrator - one for Swarm, and one for Kubernetes. Just as in cloud orchestration, these custom interfaces were used to describe specific parameters of the orchestrator. These were orchestration-related parameters which did not fit with the general options that we had expressed as properties of the custom node type we created for Docker containers. When it came time to replace Swarm with Kubernetes, our existing ADTs did not require any changes to the containers described using our custom node type. The only change required was replacing the Swarm interfaces attached to these nodes with Kubernetes interfaces.

Other application components such as networks, volumes and configurations should be defined alongside application containers and their interoperability should be defined with a TOSCA relationship. The most common additional component which needs defining in an ADT is the volume. These can be defined using the type *tosca.nodes.MiCADO.Container.Volume*. Since the container orchestrator is responsible for orchestrating volumes as well as containers, additional orchestrator specific parameters can again be passed for managing lifecycle stages of volumes. These volumes can then be attached to containers using the TOSCA relationship *tosca.relationships.AttachesTo*. Networks (Docker networks) were previously supported in ADTs in a past implementation of MiCADO, and support for defining configuration files (Kubernetes ConfigMaps) in the ADT is currently being investigated and will be present in a future release.

7.3 Policies

Policies (excluding security) in MiCADO have so far generally centred on scalability. Deployment policies such as the location policy and connection policy are expressed directly as properties on the relevant virtual machine. Execution policies are expressed directly as the runtime properties of the relevant container. Currently work is ongoing to create new policies which can generate alerts based on certain metrics. Other work is investigating the possibility of specifying in an ADT which metrics should be collected from worker nodes - these would be written as policies too.

Policies in MiCADO are enforced by the PolicyKeeper component, which takes a number of parameters to generate different scaling rules. PolicyKeeper first accepts user-defined constants, such as deadlines, minimum or maximum instances, and upper and lower thresholds. These constants will be used in the scaling logic to make decisions. Next, PolicyKeeper accepts a customisable set of Prometheus queries which get resolved on each PolicyKeeper evaluation. These queries are stored as variables and compared against constants in the scaling logic to make scaling decisions. PolicyKeeper also accepts a custom set of AlertManager alerts, generated through Prometheus queries. These alerts are stored as variables and can trigger different events in the scaling logic. Lastly, PolicyKeeper accepts the custom scaling logic itself, as a Python script.

A very simple performance policy which scales containers based on a queried latency metric might be defined as:

```
policies:
  type: tosca.policies.MiCADO.scaling
  properties:
    constants:
      HIGH_LATENCY: 70
    queries:
      LATENCY: avg(latency{container=”my-wp”})
    logic: |
      if LATENCY > HIGH_LATENCY:
        container_count = container_count + 1
```

An equally simple consumption policy scaling virtual machines based on a CPU alert can be defined as:

```
policies:
  type: tosca.policies.MiCADO.scaling
  properties:
    constants:
      HIGH_CPU: 85
    alerts:
    - alert: CPU_OVERLOAD
      expression: avg(node_cpu) > HIGH_CPU
      for: 90s
    logic: |
      if CPU_OVERLOAD:
        node_count = node_count + 1
```

7.4 Security Policies

Security policies facilitate the firewall configuration for a specific application with a set of predefined security policies that correspond to application-level filtering rules. The policy objects are organized in a hierarchical manner and attributes are added to the specific types through inheritance, where each object type will automatically carry forward the attributes of any of their ancestors. The object that exposes the ports also has a “security” parameter that accepts an object derived from the AbstractNetworkSecurityPolicy type. This fits well with the overall design approach of the Application Description Templates whereby Policies can be arranged in a hierarchy akin to that of classes in an Object-Oriented language.

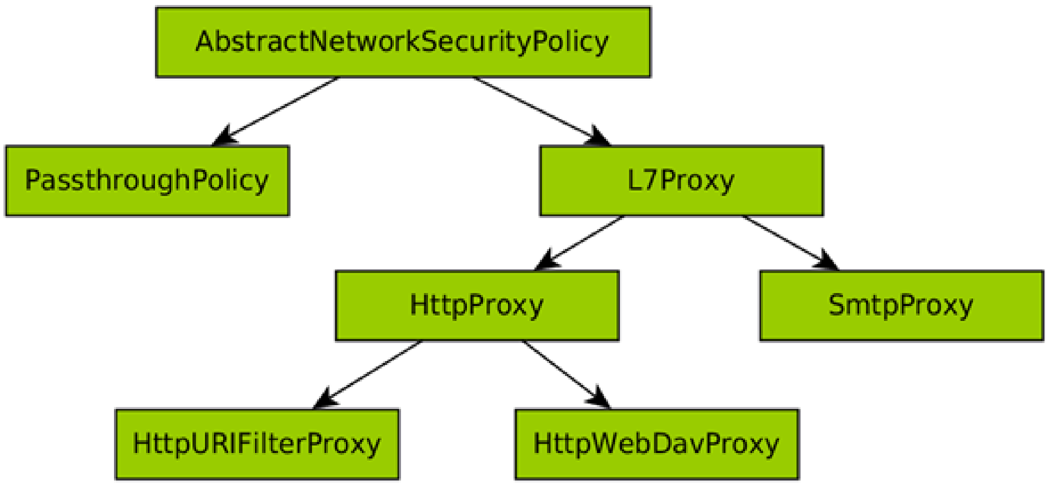


Figure 2, Hierarchy of network security policies

Each Policy in the hierarchy of network security policies (see Figure 2) is further described with two sets of parameters: generic parameters that describe the metadata of the policy and parameters that are specific. These specific sets of parameters are listed in deliverable 7.4 in detail. Based on such information, corresponding security policies are defined in the ADT to fulfil the requirement of describing both open ports and its relevant network policies. Here follows a basic description of the available network security policies in MiCADO ADTs.

AbstractNetworkSecurityPolicy

- **Name**: *tosca.policies.MiCADO.Security.Network*
- **Type**: Abstract container of all security policies
- **Description**: Requires to set specific configuration for firewalls in worker nodes

Derived from: *tosca.policies.Root*

PassthroughPolicy

- **Name**: *tosca.policies.MiCADO.Security.Network.Passthrough*
- **Type**: Policy that specifies no filtering
- **Description**: Policy that specifies that no additional filtering should be done and no application-level firewall should be applied on the traffic

Derived from: *tosca.policies.MiCADO.Security.Network*

L7Proxy

- **Name**: *tosca.policies.MiCADO.Security.Network.L7Proxy*
- **Type**: Policy that specifies application level relaying and TLS control
- **Description**: Policy that specifies no additional protocol enforcement, but states that and application-level firewall should be applied to the traffic and also can provide TLS
- **Target**: Worker nodes

Derived from: *tosca.policies.MiCADO.Security.Network*

SmtpProxy

- **Name**: *tosca.nodes.MiCADO.SecurityPolicy.Network.SmtpProxy*
- **Type**: Policy that specifies that the SMTP protocol should be enforced
- **Description**: Policy that specifies SMTP protocol enforcement, specifies that an application-level firewall should be applied to the traffic and also can provide TLS control
- **Target**: Worker node

Derived from: *tosca.nodes.MiCADO.SecurityPolicy.L7Proxy*

HttpProxy

- **Name**: *tosca.nodes.MiCADO.SecurityPolicy.Network.HttpProxy*
- **Type**: Policy that specifies application level relaying and TLS control
- **Description**: Policy that specifies HTTP protocol enforcement and states that and application-level firewall should be applied on the traffic and also can provide TLS control
- **Target**: Worker node

Derived from: *tosca.nodes.MiCADO.SecurityPolicy.L7Proxy*

HttpURIFilterProxy

- **Name**: *tosca.nodes.MiCADO.SecurityPolicy.Network.HttpURIFilterProxy*
- **Type**: Policy that specifies that the HTTP protocol should be enforced and provides URL filtering
- **Description**: Policy that specifies HTTP protocol enforcement with regex-based URL filtering capabilities, specifies that an application-level firewall should be applied to the traffic and also can provide TLS control
- **Target**: Worker node

Derived from: *tosca.nodes.MiCADO.SecurityPolicy.HttpProxy*

HttpWebdavProxy

- **Name**: *tosca.nodes.MiCADO.SecurityPolicy.Network.HttpWebdavProxy*
- **Type**: Policy that specifies that the HTTP protocol should be enforced and allows request methods required for WebDAV
- **Description**: Policy that specifies HTTP protocol enforcement with extended set of request methods, but states that and application-level firewall should be applied to the traffic and also can provide TLS control

Derived from: *tosca.nodes.MiCADO.SecurityPolicy.HttpProxy*

8. Examples of Application Description Templates used for real applications in COLA

This section of the deliverable describes the ADTs that have been developed for the three near operational pilots, the over 20 proof of concept feasibility studies, and further applications that were used mainly for testing and demonstration purposes. As many of these templates have been introduced and described in detail in D5.4, here we only highlight the evolution of theseb ADTs and the differences when compared to the earlier versions.

8.1 Use Case 1 – Social Media Data Analytics by Inycom & SARGA

This use case deals with social media data analytics for public sector organisations. Since the publication of D5.4, a second container has been added to the application architecture. This second container is responsible for classifying data in the SOLr database to improve its display on the website. This new classifier container has no scaling requirements - only a single instance is necessary. Figure 3 shows the new architecture of the application, with the original Magician container remaining unchanged on the left side of the figure, and the newly added classifier container represented on the right side of the figure.

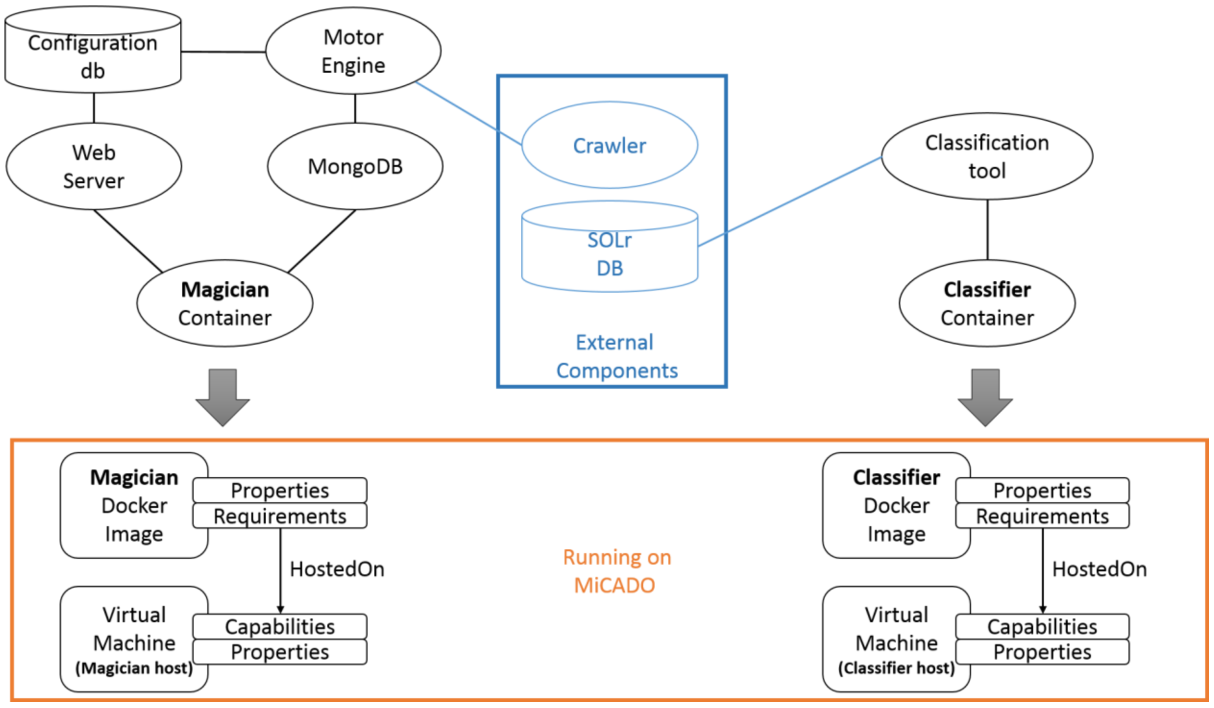


Figure 3, New architecture diagram for the Inycom Use Case

The logical separation of scalable components from non-scaling components is ensured with the TOSCA relationship HostedOn. The Classifier container is restricted to running on a specific Virtual Machine, and no scaling policy is attached to either of these components. The Magician container is restricted to running on another Virtual Machine, and matching scaling policies are attached to both of these components. During the initial phases of deploying the application to MiCADO, consumption-based scaling policies for both CPU load and Memory utilisation were tested. Policies based on CPU load better matched the actual scaling requirements of the application. As such, the current version of the ADT for this use case scales both the Magician container and the Virtual Machine hosting it using CPU load as a metric.

The policy description remains largely unchanged since D5.4, though the implementation of some policies has changed. Table 4 below shows how each policy is implemented in the ADT (added in bold). The full ADT of this use-case can be found in the COLA Github ADT repository[5] and also in Appendix A.

[5] https://github.com/micado-scale/tosca/blob/D5.5/ADT/inycom.yaml

	Policy	Notes
P1.1	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. **These are defined in the policies section of the ADT and target the Magician container and the virtual machine hosting Magician.**
P1.2	Resource Deployment Policy	Defines the requirements of the Virtual Machine in terms of CPU, Memory Size and Disk size. **It is directly defined as properties of the Virtual Machine and stored as metadata in the capabilities of the Virtual Machine.**
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. **It is directly defined in the properties section at the Virtual Machine level, and in the properties section at the Container level.**
P1.4	Location Deployment Policy	It dictates that the container must be physically located in the European Union. **This is specified in the interface section of the Virtual Machine.**

Table 4, Modified policy descriptions for the Inycom Use Case

8.2 Use Case 2 – Evacuation Simulation by Saker Solutions & Brunel University

This second use case deals with extending a desktop grid to the cloud to improve simulation experiment execution times. The components which make up the application have not been changed, though the deployment strategy for this use case has changed significantly. The application was found to be incompatible with containerisation in its current state and so the application was simply built into a virtual machine image. This use case is now the primary proof-of-concept for so-called VM-only deployments in MiCADO and serves as an example for other VM-only deployments. The updated application architecture is depicted in Figure 4.

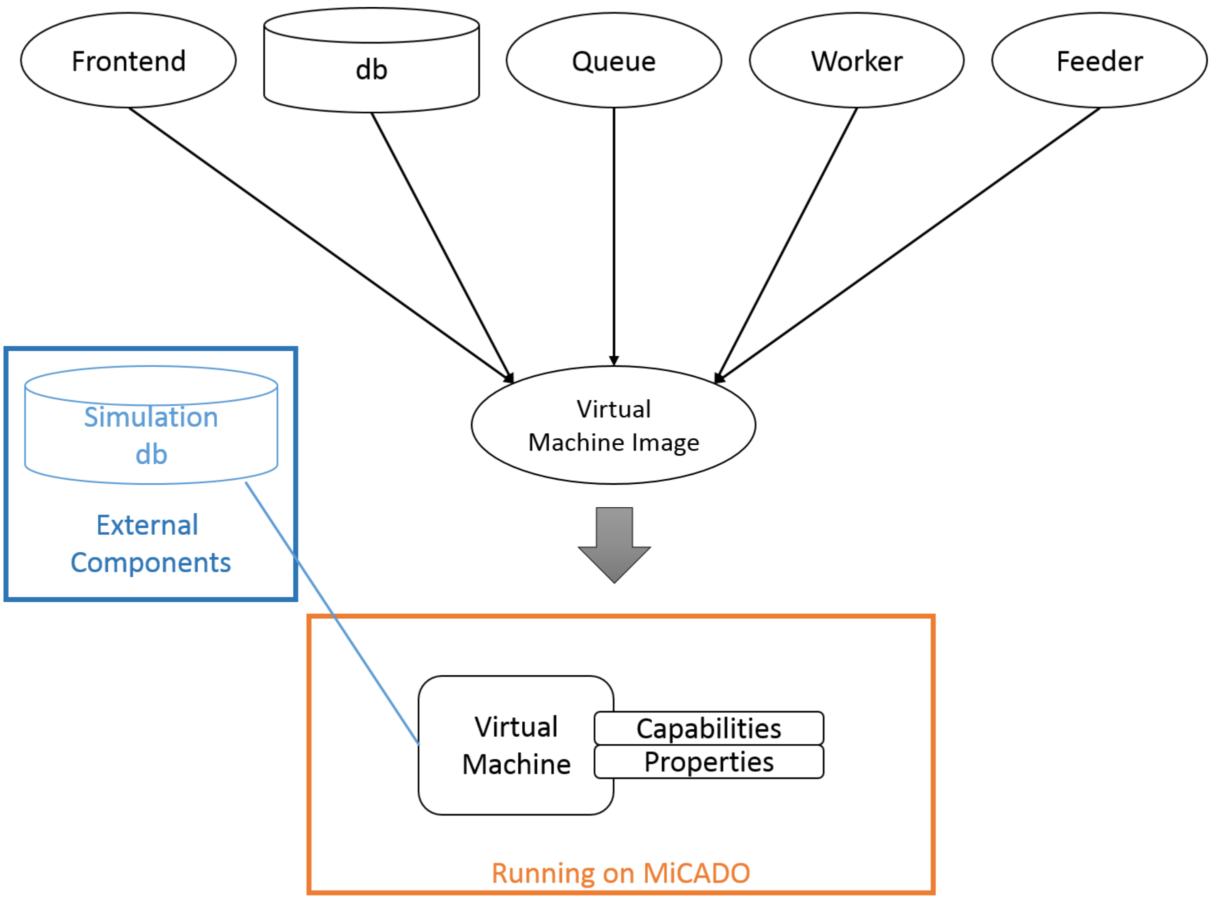


Figure 4, New architecture diagram for the SAKER/Brunel Use Case

While the main scalability policy for this use case still strives to reduce cost, no explicit maximum budget or monetary cut-off threshold are specified. The policy aims to reduce costs by using the least number of virtual machine compute nodes to complete the given jobs by a specified deadline. Hard cut-offs are specified for the maximum number of compute nodes to use for an experiment. Otherwise, the policy descriptions for this use case have not changed, but their implementation differs slightly. The updated Table 5 shows these changes in bold. The full ADT of this use-case can be found in the COLA Github ADT repository[6] and also in Appendix B.

[6] https://github.com/micado-scale/tosca/blob/D5.5/ADT/saker.yaml

	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 setting the maximum number of Workers possible for a session, and through the logic of the auto-scaler. This policy is set in the policies section of the ADT.**
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. **It is directly defined as properties of the virtual machine.**
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 properties of the Virtual Machine and stored as metadata in the capabilities of the Virtual Machine.**

Table 5, Modified policy descriptions for the Saker/Brunel Use Case

8.3 Use Case 3 – Scalable Hosting, Testing and Automation of Applications by Outlandish and The Audience Agency

This use case involves scalable hosting, testing and automation of applications and application development stages for SMEs and the public sector. The primary application being tested in this use case is The Audience Agency's Audience Finder - a data-mining application for analysis of audiences of theatres, museums, and other types of entertainment and venues. The architecture of the application has changed slightly, and now features the web server and PHP implementation in separate containers, communicating with an instance of Memcached for caching needs. Communication with the external components is unchanged. The separation of these two components into their respective containers is visualised in Figure 5.

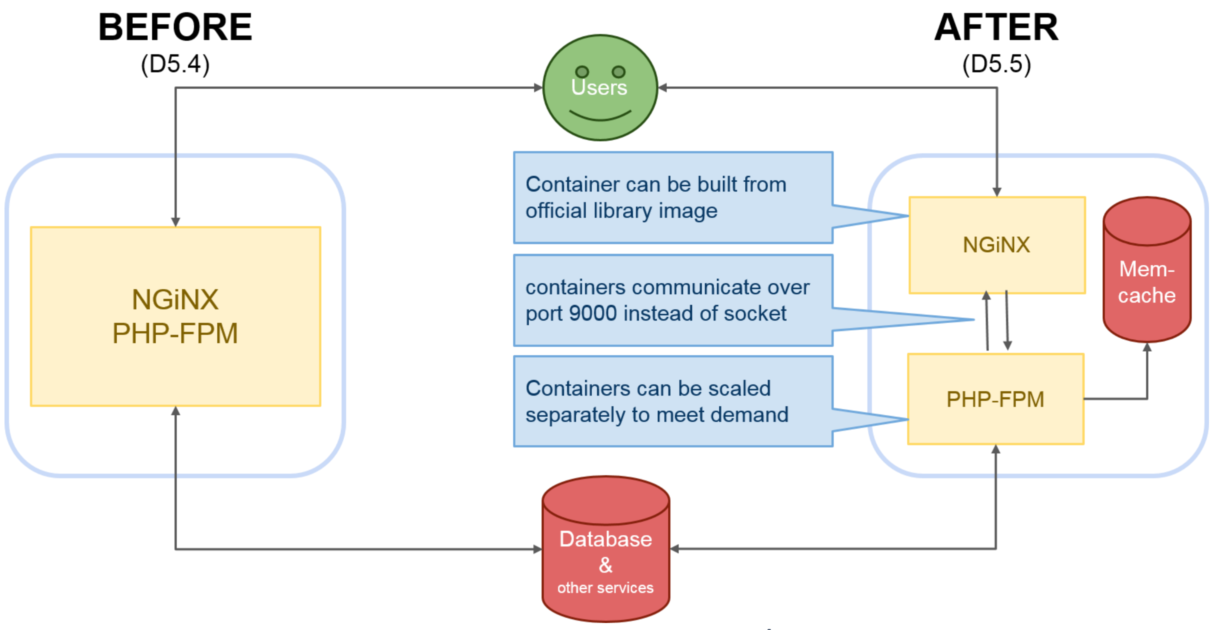

BEFORE
(D5.4)
AFTER
(D5.5)
Users
NGiNX
PHP-FPM
Container can be built from official library image
containers communicate over port 9000 instead of socket
Containers can be scaled separately to meet demand
NGiNX
PHP-FPM
Mem-cache
Database & other services

Figure 5, New container architecture for the Outlandish/Audience Agency Use Case

Within the policy description, the consumption-based scalability policy has been split, since MiCADO now supports monitoring of web connections. The original CPU consumption-based policy remains, and the deadline-based scalability policy has been replaced with a performance based scalability policy. This performance-based policy scales based on metrics pulled directly from the web server such as number of accepted requests, and the latency of certain connections. The logic expressed by the execution policy is encapsulated into the application logic. Otherwise, policies have remained unchanged, though their implementation in the ADT may have changed. Table 6 below shows the changes to the use case policy descriptions in bold. The full ADT of this use-case can be found in the COLA Github ADT repository[7] and also in Appendix C.

	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. **This policy scales based on CPU and is set in the policies section of the ADT.**
P3.2	**Performance Based Scalability**	**It describes the deployment of a new instance on the condition that the number of served web requests, or the measured latency is greater than the given threshold.**

[7] https://github.com/micado-scale/tosca/blob/D5.5/ADT/outlandish.yaml

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. **It is directly defined as properties of the virtual machine.**
P3.4	Resource Deployment Policy	Defines the requirements of the Virtual Machine in terms of CPU, Memory Size and Disk size. **It is directly defined as properties of the Virtual Machine and stored as metadata in the capabilities of the Virtual Machine.**

Table 6, Modified policy descriptions for the Outlandish/Audience Agency Use Case

8.4 Test and Demonstrator Applications

During the development of MiCADO, a series of test applications were necessary to perform end to end testing of MiCADO releases. Early test applications were small and very basic, only testing one or two features of MiCADO. More recently, test applications are fuller distributed applications in a SOA or microservices architecture, which test many aspects of MiCADO and the ADT. The ADTs for these test applications are included with each new MiCADO release, and tutorials are available to help users get started with running them. Short descriptions of these applications follow below, and their benefits are further discussed in the next section. The development of these ADTs was also driven by the need to develop reusable components that might be used in the future in the implementation of the 8.4 prototypes.

stressng

The tool stressng was the first test application for which a MiCADO ADT was composed. It is an enhanced release of stress - a standard linux binary for generating deliberate load of certain hardware resources, namely CPU and memory. The architecture for this application is a single container (stressng) running on a single, scalable virtual machine. A basic consumption scalability policy is attached to both container and VM and given some simple CPU thresholds on which to scale. After the application is deployed to MiCADO, operators can perform a runtime update of the ADT to change the amount of load the stressng container generates. When the container generates CPU over the set high threshold, MiCADO will scale containers and virtual machines to the set maximum. If the CPU load is adjusted back downwards below the low threshold, MiCADO will scale the infrastructure back down to the minimum. This is a good example of a basic single container application running in MiCADO and can be used to verify that MiCADO has been configured correctly. The full ADT for this test application can be found in the COLA GitHub repository[8].

NGINX

NGINX is a popular and widely used web server. The architecture is very similar to stressng with a single container on a single scalable virtual machine. The added value is that certain

[8] https://github.com/micado-scale/tosca/blob/D5.5/ADT/tests-demos/stressng.yaml

metrics can be exported using a Prometheus exporter, and can then be used in MiCADO to define scaling rules. MiCADO pulls the number of accepted connections from the NGINX exporter and makes scaling decisions based on the average rate of accepted connections over a set time. Using an HTTP load testing tool such as wrk, it is possible to generate a large number of requests and force MiCADO to respond by scaling the infrastructure up. When the load test completes and the rate of connections falls, MiCADO scales the infrastructure back down. The NGINX example is a good example of allowing ingress into a container, and for testing external Prometheus exporters and metrics in MiCADO. The full ADT for this demonstrator application can be found in the COLA GitHub repository[9].

WordPress

WordPress is a popular content hosting platform for blogs and websites. The architecture is significantly more complex than the previous two testing applications. This application features three different containers hosted on two different virtual machines. One virtual machine hosts the backend components - an NFS server for shared storage and a database server for storing and organising data. A second virtual machine hosts the WordPress frontend running on an Apache web server. Again, a load testing tool such as wrk can be used to generate network traffic. MiCADO will scale the frontend container and frontend virtual machine up and down in response to alerts generated based on the network traffic measured across the infrastructure. This sample application is ideal for testing volumes, containers on specific hosts, connections between containers. The complete ADT for the WordPress demonstrator can be found in the COLA GitHub repository[10].

cQueue / jQueuer

The final two test applications for MiCADO are both open source asynchronous task queues for deadline-based scenarios. The architecture for these applications is similar to WordPress, where one virtual machine plays host to the queue master application, which is made up of containers running the database, message broker, queue managers, queue frontend and custom Prometheus exporters. A second scalable virtual machine hosts the queue worker containers, which collect jobs from the queue master and execute them in containers. The scaling logic for these applications pull metrics such as length of queue, completed jobs, and average execution time from the queue master exporters. Using these metrics, MiCADO calculates the minimum required amount of virtual machines and queue worker containers to complete a given number of jobs by a supplied deadline. Both of these ADTs can be found in the COLA GitHub repository – one for cQueue[11], and one for jQueuer[12].

8.5 Proof of concept Feasibility Study Prototypes

A large number of further applications have been investigated by the project, as described in D8.4, for future deployment with MiCADO, and resources are already in place in order to facilitate authoring ADTs to describe these applications. The existing use cases form a good base of varied applications - the consumption-based scalability of the Inycom use case; the virtual machine-only deployments of the Saker use case; the performance based scalability

[9] https://github.com/micado-scale/tosca/blob/D5.5/ADT/tests-demos/nginx.yaml
[10] https://github.com/micado-scale/tosca/blob/D5.5/ADT/tests-demos/wordpress.yaml
[11] https://github.com/micado-scale/tosca/blob/D5.5/ADT/tests-demos/cqueue.yaml
[12] https://github.com/micado-scale/tosca/blob/D5.5/ADT/tests-demos/jqueuer.yaml

and multiple containers in the Outlandish use case. The test applications for MiCADO also provide a range of different architectures and scalability policies which can be reused when authoring new ADTs - web server deployments can be based on the NGINX or WordPress examples; queue and job-based experimentation can be based on the cQueue or jQueuer examples.

Index	Feasibility Study	Area	Draft ADT
1	Evacuation Service	High Performance Modelling & Simulation	✓
2	High Performance Simulation Analytics		✓
3	MAGOS		✓
4	JaamSim Portal		✓
5	Repast Portal		✓
6	PALMS		✓
7	FLEE		✓
8	D-SIMLAB		
9	CAROL	Artificial Intelligence	
10	Feature Branch	Web Applications	
11	School Cuts		✓
12	Shared Hosting		✓
13	Social Monitor	Social Media	
14	Competitors Alerts	Web Monitoring and Alerts	✓
15	Attendance Analysis Service	Data Analytics	✓
16	New sectors		✓
17	New territories		✓
18	MainRail	Internet of Things / Smart Cities	✓
19	DataAvenue	Remote Storage	✓
20	MiCADOscale on cloudSME cloudbroker platform	Autoscale as a Service	✓
21	WordPress HKN	Cloud Hosting	✓
22	MiCADOscale HKN		✓
23	Integrated Testing Solution	Software Testing	✓
24	Application Status		

Table 7, List of MiCADO feasibility studies

Table 7 shows the list of proof-of-concept feasibility studies for MiCADO, defined in D8.4. Where indicated in the table, a draft ADT for the given prototype exists in the MiCADO GitHub repository[13]. Those studies for which a draft ADT does not already exist can be crafted from existing ADTs as suggested below.

The central requirement of many studies in the areas "High Performance Modelling & Simulation" and "Software Testing", as well as case 13 and 18 are deadline-based and job-queue scaling, for which ADT authors can refer to the ADT topology and policy descriptions

[13] https://github.com/micado-scale/tosca/tree/D5.5/ADT/prototypes

in either the jQueuer[14] or cQueue[15] test applications or in the SAKER use case (Appendix B).

The area of “Data Analytics” as well as case 19 are consumption driven applications which will require load-based scaling similar to that seen in the stressng[16] demonstrator included with MiCADO, or the Inycom use case (Appendix A). Study 9 looks to benefit from both consumption and deadline policies, so will use both types of ADT mentioned above as a reference point.

Feasibility studies in the area “Web Applications” as well as case 21 are WordPress based applications which can closely follow the WordPress demonstrator ADT[17] included with MiCADO or the Outlandish use case (Appendix C). Studies 2 and 18 require Windows as a base OS, and so can re-use the VM-only descriptions found in the Saker ADT (Appendix B).

The studies 20 and 22 plan to interface with MiCADO and will not require any one specific Application Description Template, but rather a wide range to support the applications their users intend to run. In preparing for this, Outlandish has already authored an ADT which matches the architecture of basic web applications. It currently supports the deployment of WordPress with different architecture and scalability requirements than those used in the test applications of MiCADO itself. This ADT has the potential to support the other similar prototypes built on similar infrastructure with similar scaling requirements.

Creating “cut & paste” ADT templates has been another focus of this work package. So far a small number of these are stored in the ADT repository on GitHub[18], which is further described in the next section. The idea of these templates is to provide future ADT authors with a selection of TOSCA descriptions for containers, virtual machines and policies which can be reused in a variety of potential applications. These sample descriptions will feature the properties of containers or virtual machines which should run or behave in a certain way. Descriptions of policies will include examples of consumption based, performance based, and deadline-based policies, each using different alerts, queries and exporters. Future ADT authors will be able to mix and match, taking one container, matching it with some virtual machine, and applying some policy to them.

9. Structure of the COLA Application Description Repository

One of the deliverables of WP5 was to provide a repository where to store the ADTs. Such repository has been developed and made available to the consortium with Deliverable D5.4. In particular in D5.4 – Section 10, the structure of the repository is explained in detail.

This first repository was previously hosted in GitHub at https://github.com/COLAProject/COLARepo. When the development of MiCADO and its components moved to its own organisation in GitHub, so too moved the ADT repository. The most recent publicly shareable ADT resources can all be found at https://github.com/micado-

[14] https://github.com/micado-scale/tosca/blob/D5.5/ADT/tests-demos/jqueuer.yaml
[15] https://github.com/micado-scale/tosca/blob/D5.5/ADT/tests-demos/cqueue.yaml
[16] https://github.com/micado-scale/tosca/blob/D5.5/ADT/tests-demos/stressng.yaml
[17] https://github.com/micado-scale/tosca/blob/D5.5/ADT/tests-demos/wordpress.yaml
[18] https://github.com/micado-scale/tosca/tree/D5.5/node_example

scale/tosca. GitHub tags are used to create a version history of ADT resources at specific releases. The most important resource available in this repository is the *micado_types.yaml* file, where all the rich parent TOSCA types are defined for later use in specific ADTs. Also, within this repository are the base ADTs of each use case, as well as any previously investigated policy descriptions.

Newly included in this repository are helpful “cut & paste” template sections which hope to help ADT authors build new templates from existing pieces. Different samples of virtual machine types, container types and policy types are available in individual files, which can then be re-used to create a complete template with varying components. This section is still a work-in-progress, but by the project end aims to be a complete resource for simplifying ADT creation for future authors and operators.

Also, under current investigation is the idea of moving this collection of ADT resources away from GitHub and into a more customisable repository which would support filtering and searches based on more detailed metadata and metatypes. There are a number of open-source options currently being evaluated, but GitHub itself is also being investigated as a potential candidate if this enhanced metadata functionality could be implemented on top of the existing repository. If another repository is found to be a more suitable option than GitHub, ADT resources will be moved across before the end of the project.

10. Strengths and weaknesses of the adopted approach

10.1 Application Developers

The ADT’s approach in MiCADO is easy to understand for a developer already familiar with Kubernetes and its syntax, although the configuration of MiCADO is more complex than the Kubernetes or Docker compose ones. The good news is that for unskilled developers there are already several sample templates of MiCADO demo applications (cqueue, stressng, WordPress…) that can be used as a starting point to modify them based on our application requirements.

Inycom has mostly used stressng to define the ADT of for the Magician application. In this aspect, it is useful to link MiCADO documentation to the ADTs and use code samples, as it is easier for newbies to understand everything. A new project depends on friendly tutorials to become used. On the other hand, as MiCADO is already under development and there are quite a few changes when a new version is released, Inycom has had to redefine several times their ADTs and it hasn’t been straight forward for developers. In this sense, stability as the end of the project approaches is very helpful.

Another particularly interesting opportunity is the possibility for skilled developers of programming dynamic scaling policies using Prometheus formulas and Python coding. Using these dynamic scaling policies to create valid ADTs is complex, but results in a variably scalable application.

The philosophy developed in COLA, can be replicated by other projects. As an example Outlandish has created a general ADT for basic web applications similar in architecture to Audience Agency’s use case application. They are currently hosting this ADT in a GitHub repository under their organisation[19].

10.2 Support for Security

Developers can define security policies in Application Description Template (ADT). More specifically, they can configure:

- firewall to open defined ports for applications
    - with the property "ports" of type "tosca.nodes.MiCADO.Container.Application.Docker"[20]
- firewall settings defined at Cloud Providers level
    - with the "firewall_policy" in the Virtual Machine Image node type description (e.g "tosca.nodes.MiCADO.CloudSigma.Compute")[21]
- network security policies
    - through "tosca.policies.MiCADO.Security.Network.L7Proxy" or other similar types in "policy_types"[22]
- application sensitive information to be stored as well as configure access to it
    - through "tosca.policies.Security.MiCADO.Secret.KubernetesSecretDistribution"

Specific documentation for applying network security policies and application sensitive information policies in an ADT can be found in the main documentation of MiCADO[23].

The capabilities to define security policies through ADT is convenient and centralized, which facilitates the maintenance of security features. Any updates to the security configuration can be managed simply by editing ADT file. Furthermore, the ADT description for security features can be reused with customization for other applications or deployment easily. Finally, in addition to the current supported security policies, ADT can be extended to further support others demanded security policies by defining further TOSCA types.

On the other hand, the current lack of specific examples for security policies definition in ADT can lead to difficulties for the very first time when developers try to compose security policies in ADT. In addition to that, without a detailed manual, the separation in defining security features (i.e. network security policies and application sensitive information are defined in "policy_types" while open ports are defined in "node_templates") can also cause some difficulties to manage such policies.

10.3 Support for Policies Description

Support for flexible and expressive policies has been one of the design focus of the ADT developed in COLA. In order to achieve an extensible and flexible description of policies, we use TOSCA types arranged in hierarchies to define the various structural entities of the ADT to allow extension of elements to match with a modification in one of the elements of the ADT. As a result, application developers can define a new sub-type in the hierarchy whilst

[19] https://github.com/outlandishideas/bedrock-micado/
[20] https://github.com/micado-scale/tosca/blob/D5.5/node_example/applications/application_example_c.yaml
[21] https://github.com/micado-scale/tosca/blob/D5.5/node_example/Occopus/cloudsigma.yaml
[22] https://github.com/micado-scale/tosca/blob/D5.5/micado_types.yaml
[23] https://micado-scale.readthedocs.io/en/latest/application_description.html

the topology and overall structure of the ADT remains unchanged. This approach is particularly relevant for the definition of the extended policy hierarchy which we have designed considering TOSCA recommendations. First, the extended policy hierarchy follows the Declarative Model, e.g. it describes the parameters that govern the policy, but it does not specify how to implement the policy. Such Declarative Model supports developing various different application level orchestrators that act upon the defined policies i. e. the policy definition does not define or restrict the implementation of the orchestrator. Second, we support the aggregation of policies in two different ways. First, policies can target one, more or all nodes, i.e. it is possible to define one policy for the entire application and a second one for a subset of nodes or for a single node. Second, policies cover distinct aspects of QoS, for example scalability, security, etc. and can be composed for each node. Such composition could lead to conflicts among the policies. As an example, a budget-constraining policy applied to the entire application may be in conflict with a deadline policy applied with either to the entire application or one of its components that requires the usage of expensive resources. Another example could be that of a privacy constraint that requires the placement of a database in a certain geographical area with a policy that fixes an incompatible budget limit.

We do not address the conflicts of policies, but we added a priority field to the policy template which expresses conflict resolution criteria that can be used by the relevant element of the ADT. Further technical details of the structure of Policies are described in D5.4

In conclusion, the development of the ADT has proven to be rather successful with a good balance between strengths and weaknesses. The strongest aspect of the ADT has been in its flexibility and expressiveness capable of describing applications defined by complex topologies and sophisticated policies, it has proven capable to be highly adaptable to different technologies. As an example, substituting allowing to substitute the Container technology with only minor changes of the code. On the user side, ADT have achieved both successes and weaknesses.

On the positive side, ADT are highly modular, and they support two kind of developers: Advanced Developers can develop ADTs buy using the elements present in the repository and can at the same time, select a set of values that are accessible through the input section.
Less knowledgeable developers can submit the applications developed just be setting the set of values flagged as input at the point before.

Nevertheless, the learning curve necessary to develop an ADT is still quite steep and work is still brought on in WP5 and WP4 to ameliorate this aspect. In WP5 we are creating a metadata section that will be used to queries in the GitHub repository while WP4 is working on a GUI interface to the ADT repository.

11. Improving ADTs through abstraction and inheritance (an example)

The most notable general improvement to ADTs since D5.4 is a reduction in the overall complexity of node template descriptions by benefitting from concepts of abstraction and inheritance in TOSCA. Our collection of node type definitions has been extended to support a wider range of common settings for both cloud infrastructure and application container descriptions. Similarly, policy descriptions have been simplified by making better use of policy type definitions. These type definitions can be stored in separate TOSCA files in GitHub and can be imported into an ADT as needed. Abstraction provides a means for removing some complexity from the final ADT, resulting in better readability and shorter length. These ADTs are supported from MiCADO v0.8.0.

As an example, here follows a break-down of the ADT for the WordPress application demonstrator currently included in the MiCADO test suite and described in Section 8.5 of this document. The new v0.8.0 ADT and the relevant type definitions are included, as well as the matching v0.7.x ADT, for comparison. As the WordPress demonstrator has quite a large topology, this example only includes the description of three frontend descriptions: the application container for the WordPress frontend, the compute node hosting that container, and the policy which enforces the scaling of that compute node.

11.1 TOSCA Version, Imports, Repository & Input Definitions

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - https://raw.githubusercontent.com/micado-scale/tosca/v0.8.0/micado_types.yaml

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

The version, imports, repository & input section remains unchanged from the previous description found in D5.4. Furthermore, it will not vary greatly across different application descriptions but for two possible adjustments: updating the import URL to pull the extended node and policy type definitions (*micado_types.yaml)* file, and modifying the repository URL in case container images are hosted outside of DockerHub.

11.2 Node Templates - Cloud Infrastructure Definitions

```
topology template:
  node templates:
    scaling-worker:
      type: tosca.nodes.MiCADO.CloudSigma.Compute
      properties:
        num_cpus: 2000
        mem_size: 2147483648
        vnc_password: xx
        libdrive_id: xx-xx-xx
        public_key_id: xx-xx-xx
        context:
          append: yes
          cloud_config: |
            runcmd:
            - apt-get install -y nfs-kernel-server nfs-common
        nics:
        - firewall_policy: xx-xx-xx
          ip_v4_conf:
            conf: dhcp
      interfaces:
        Occopus:
          create:
            inputs:
              interface_cloud: cloudsigma
              endpoint_cloud: https://xx-xx-xx.com/api/v1
      capabilities:
        host:
          properties:
            num_cpus: 2
            mem_size: 2 GB
```

TOSCA Snippet 1. Cloud infrastructure description in the WordPress ADT, MiCADO v0.7.x

```
topology template:
  node templates:
    scaling-worker:
      type: tosca.nodes.MiCADO.CloudSigma.Occo.small.NFS
      properties:
        endpoint: https://xx-xx-xx.com/api/v1
        vnc_password: xx
        libdrive_id: xx-xx-xx
        public_key_id: xx-xx-xx
        nics:
        - firewall_policy: xx-xx-xx
          ip_v4_conf:
            conf: dhcp
```

TOSCA Snippet 2. Cloud infrastructure description in the WordPress ADT, MiCADO v0.8.0

```
  tosca.nodes.MiCADO.CloudSigma.Occo.small.NFS:
    description: CloudSigma VM (2GHz/2GB) with NFS dependencies, by Occopus
    derived_from: tosca.nodes.MiCADO.CloudSigma.Compute.Occo.small
    properties:
      num_cpus:
        type: integer
        default: 4000
        required: true
      mem_size:
        type: integer
        default: 4294967296
        required: true
      context:
        type: map
        default:
          append: yes
          cloud_config: |
            runcmd:
            - apt-get install -y nfs-kernel-server nfs-common
        required: true

    interfaces:
      Occopus:
        type: tosca.interfaces.MiCADO.Occopus
        create:
          inputs:
            interface_cloud: cloudsigma
            endpoint_cloud: { get_property: [ SELF, endpoint ] }
```

TOSCA Snippet 3. Cloud infrastructure type definition in *micado_types.yaml*, MiCADO v0.8.0

This is the definition of a CloudSigma compute node, which will be interpreted by Occopus to provision virtual machines in the Cloud. This node will host the containers which will be defined in the next section. Snippet 1 represents the state of this section of the ADT before richer type definitions were implemented. Snippet 2 achieves the same configuration of a cloud instance, but reads with much more clarity.

Snippet 3 defines defaults for the size of the instance, pre-defines contextualisation through cloud-init, and shifts the complexity of the user-provided endpoint from the interfaces section to the properties section. This results in a much cleaner description, where the complexity has been abstracted out of the ADT, now saved in a separate type definitions file – *micado_types.yaml*.

11.3 Node Templates - Application Definitions

```
    wordpress:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      properties:
         name: wordpress
         env:
         - name: WORDPRESS_DB_HOST
           value: wordpress-mysql
         - name: WORDPRESS_DB_PASSWORD
           value: admin
         resources:
           requests:
             cpu: "900m"
         ports:
         - target: 80
           nodePort: 30010
           type: NodePort
         - containerPort: 80
           name: wordpress
      artifacts:
       image:
         type: tosca.artifacts.Deployment.Image.Container.Docker
         file: wordpress:5.0.3-apache
         repository: docker_hub
      requirements:
        - volume:
            node: nfs-volume
            relationship:
              type: tosca.relationships.AttachesTo
              properties:
                location: /var/www/html
        - host: scaling-worker
      interfaces:
        Kubernetes:
          create:
            inputs:
              strategy:
                type: Recreate
```

TOSCA Snippet 4. Application container description in the WordPress ADT, MiCADO v0.7.x

```
    wordpress:
      type: tosca.nodes.MiCADO.Container.Application.Docker.Deployment
      properties:
        image: wordpress:5.0.3-apache
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql
        - name: WORDPRESS_DB_PASSWORD
          value: admin
        resources:
          requests:
            cpu: "900m"
        ports:
        - port: 80
          nodePort: 30010
        - containerPort: 80
        labels:
          tier: frontend
      requirements:
        - host: scaling-worker
        - volume:
            node: nfs-volume
            relationship:
              type: tosca.relationships.AttachesTo
              properties:
                location: /var/www/html
```

TOSCA Snippet 5. Application container description in the WordPress ADT, MiCADO v0.8.0

```
  tosca.nodes.MiCADO.Container.Application.Docker.Deployment:
    description: Abstraction of Docker container node. A Kubernetes Deployment
    derived_from: tosca.nodes.MiCADO.Container.Application.Docker
    artifacts:
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
        file: { get_property: [ SELF, image ] }
        repository: docker_hub
    interfaces:
      Kubernetes:
        type: tosca.interfaces.MiCADO.Kubernetes
        create:
          inputs:
            kind: Deployment
            spec:
              strategy:
                type: Recreate
```

TOSCA Snippet 6. Application container definition in *micado_types.yaml*, MiCADO v0.8.0

The above snippet shows the definition of an application container for the WordPress Apache server that makes up the WordPress frontend. This description is interpreted by Kubernetes to orchestrate this container across the virtual machine nodes which have been described in the above section. Snippet 4 shows the description from previous MiCADO versions, and Snippet 5 shows the new description which achieves the same orchestration configuration, only with less complexity. The type definition which enables this is seen in Snippet 6, where the complexity of pointing to a Docker image in the artifacts section has been shifted to the properties section, and a default interface with Kubernetes settings has been pre-defined.

This demonstrator requires the attachment of volumes to containers, as seen in the requirements section in the container description above. These volume descriptions have also been simplified using the same methods of abstraction, as seen below.

```
    nfs-volume:
      type: tosca.nodes.MiCADO.Container.Volume
      properties:
        name: nfs-volume
      interfaces:
        Kubernetes:
          create:
            inputs:
              nfs:
                server: 10.96.0.240
                path: /
```

TOSCA Snippet 7. Volume description in the WordPress ADT, MiCADO v0.7.x

```
    nfs-volume:
      type: tosca.nodes.MiCADO.Container.Volume.NFS
      properties:
        server: 10.96.0.240
        path: /
```

TOSCA Snippet 8. Volume description in the WordPress ADT, MiCADO v0.8.0

```
  tosca.nodes.MiCADO.Container.Volume.NFS:
    description: An abstraction of the volume node for Kubernetes NFS volumes
    derived_from: tosca.nodes.MiCADO.Container.Volume
    properties:
      path:
        type: string
        description: path on host
        required: true
      server:
        type: string
        description: NFS server IP
        required: true
    interfaces:
      Kubernetes:
        type: tosca.interfaces.MiCADO.Kubernetes
        create:
          inputs:
            spec:
              nfs:
                path: { get_property: [ SELF, path ] }
                server: { get_property: [ SELF, server ] }
```

TOSCA Snippet 9. Volume type definition in *micado_types.yaml*, MiCADO v0.8.0

Snippet 7 shows the description of a volume, also to be orchestrated by Kubernetes, as it was in MiCADO v0.7.x. The simpler, updated description for v0.8.0 can be seen in Snippet 8. The type definition which helps to realise this simplification is in Snippet 9, where the complexity of the interfaces section is moved to the properties section.

11.4 Policies

Policies have benefitted from abstraction since before MiCADO v0.8.0, but their implementation has changed since D5.4, so they are mentioned briefly here below.

```
policies:
  - scalability:
      type: tosca.policies.Scaling.MiCADO.VirtualMachine.Net.wordpress
      targets: [ scaling-worker ]
      properties:
        constants:
          NODE_NAME: 'scaling-worker'
        min_instances: 1
        max_instances: 3
```

TOSCA Snippet 10. Policy description in MiCADO

```
tosca.policies.Scaling.MiCADO.VirtualMachine.Net.wordpress:
  derived_from: tosca.policies.Scaling.MiCADO
  description: base WordPress policy defining alerts and rules
  properties:
    alerts:
      type: list
      description: pre-define alerts for WordPress virtual machines
      default:
      - alert: node_overloaded
        expr: 'avg(rate(container_network_receive_bytes_total)) > 60'
        for: 1m
      - alert: node_underloaded
        expr: 'avg(rate(container_network_receive_bytes_total)) < 30'
        for: 1m
      required: true
    scaling_rule:
      type: string
      description: pre-define scaling rule for WordPress VMs
      default: |
        if len(m_nodes) <= m_node_count:
          if node_overloaded:
            m_node_count+=1
          elif node_underloaded:
            m_node_count-=1
        else:
          print('Transient phase, skipping update of nodes...')
      required: true
```

TOSCA Snippet 11. Policy type definition in MiCADO

Snippet 10 shows the abstracted description of the policy which will enforce scaling of the compute nodes which are hosting the WordPress frontend. This policy can be applied with its defaults (seen in the type definition in Snippet 11) and certain parameters important to the operator, such as minimum and maximum instances can easily be modified. The type definition describes the alerts to be generated using Prometheus queries as well as the scaling logic to apply using Python script. These are set as the defaults which will be applied whenever this policy type is reused.

12. Conclusions

After almost three years a few conclusions could be drawn on the strengths and weaknesses of the design and implementations of the ADTs.

1) The overall approach proved to be efficient. The decision to use TOSCA (A wide spread standard as a “lingua franca” as an abstraction layer that separates and connects the description of the applications to the underlying cloud technologies has proven so successful that switching from one technology - Docker - to another technology - Kubernetes only required a few days of work.

2) The decision to adopt a two-layer topology (Virtual Machines and Containers) allowed to simplify the ADT structure whilst covering a significant number of use cases.

3) The overall design of the policies proved to be quite effective and the possibility to extend policies with specific python code made them particularly flexible. On the other end, many of the policies that were first envisaged ended up not being used (e.g. Authentication and Authorization) as those functionalities have been implemented within the code of the application and not in their deployment and execution description.

4) Unfortunately, the TOSCA language proved to be rather difficult to learn and the effort we have spent so far to ease the learning curve (Tutorials, structures repositories with examples, etc.) have not achieved the success we hoped for. Efforts will continue in this direction until the very end of the project.

13. Appendices

13.1 Appendix A: Inycom Use Case ADT

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - https://raw.githubusercontent.com/micado-scale/tosca/v0.7.3/micado_types.yaml

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

topology_template:
  node_templates:
    magician:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      properties:
        resources:
          requests:
            cpu: "800m"
        ports:
        - target: 8080
          published: 8081
          nodePort: 30808
          type: NodePort
      requirements:
      - host:
          node: worker-node
      artifacts:
       image:
         type: tosca.artifacts.Deployment.Image.Container.Docker
         file: magician
         repository: docker_hub
      interfaces:
        Kubernetes:
          create:
            implementation: image
            inputs:
              strategy:
                type: Recreate

    magicianclassifier:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      requirements:
      - host:
          node: classifier-server
      artifacts:
       image:
         type: tosca.artifacts.Deployment.Image.Container.Docker
         file: magician-classifier
         repository: docker_hub
      interfaces:
        Kubernetes:
          create:
            implementation: image
            inputs:
              strategy:
                type: Recreate

    classifier-server:
      type: tosca.nodes.MiCADO.CloudSigma.Compute
      properties:
        num_cpus: 2000
        mem_size: 2147483648
        vnc_password: xx
        libdrive_id: xx
        public_key_id: xx
        nics:
        - firewall_policy:  xx
          ip_v4_conf:
            conf: dhcp
      interfaces:
        Occopus:
          create:
            inputs:
              interface_cloud: cloudsigma
              endpoint_cloud: https://zrh.cloudsigma.com/api/2.0
      capabilities:
        host:
          properties:
            num_cpus: 2
            mem_size: 2 GB

    worker-node:
      type: tosca.nodes.MiCADO.CloudSigma.Compute
      properties:
        num_cpus: 2600
        mem_size: 4294967296
        vnc_password: xx
        libdrive_id: xx
        public_key_id: xx
        nics:
        - firewall_policy: xx
          ip_v4_conf:
            conf: dhcp
      interfaces:
        Occopus:
          create:
            inputs:
              interface_cloud: cloudsigma
              endpoint_cloud: https://xx/api/
      capabilities:
        host:
          properties:
            num_cpus: 2
            mem_size: 4 GB

  policies:
    - scalability:
        type: tosca.policies.Scaling.MiCADO.VirtualMachine.CPU.magician
        targets: [ worker-node ]
        properties:
          constants:
            NODE_NAME: 'worker-node'
            NODE_TH_MAX: '70'
            NODE_TH_MIN: '50'
          min_instances: 1
          max_instances: 4
    - scalability:
        type: tosca.policies.Scaling.MiCADO.Container.CPU.magician
        targets: [ magician ]
        properties:
          constants:
            SERVICE_NAME: 'magician'
            SERVICE_FULL_NAME: 'magician'
            SERVICE_TH_MAX: '70'
            SERVICE_TH_MIN: '50'
          min_instances: 1
          max_instances: 4

policy_types:
  tosca.policies.Scaling.MiCADO.Container.CPU.magician:
    derived_from: tosca.policies.Scaling.MiCADO
    description: base MiCADO policy defining data sources, constants, queries, alerts,
limits and rules
    properties:
      alerts:
        type: list
        description: pre-define alerts for container CPU
        default:
        - alert: service_overloaded
          expr:
'avg(rate(container_cpu_usage_seconds_total{container_label_io_kubernetes_container_name="{
{SERVICE_FULL_NAME}}"}[60s]))*100 > {{SERVICE_TH_MAX}}'
          for: 30s
        - alert: service_underloaded
          expr:
'avg(rate(container_cpu_usage_seconds_total{container_label_io_kubernetes_container_name="{
{SERVICE_FULL_NAME}}"}[60s]))*100 < {{SERVICE_TH_MIN}}'
          for: 2m
        required: true
      scaling_rule:
        type: string
        description: pre-define scaling rule for container CPU
        default: |
          if len(m_nodes) == m_node_count:
            if service_overloaded and m_node_count > m_container_count:
              m_container_count+=1
            if service_underloaded:
              m_container_count-=1
          else:
            print('Transient phase, skipping update of containers...')
        required: true

  tosca.policies.Scaling.MiCADO.VirtualMachine.CPU.magician:
    derived_from: tosca.policies.Scaling.MiCADO
    description: base MiCADO policy defining data sources, constants, queries, alerts,
limits and rules
    properties:
      alerts:
        type: list
        description: pre-define alerts for VM CPU
        default:
        - alert: service_working
          expr:
'avg(rate(container_cpu_usage_seconds_total{container_label_io_kubernetes_container_name="{
{SERVICE_FULL_NAME}}"}[60s]))*100 > {{SERVICE_TH_MAX}}'
          for: 30s
        - alert: node_overloaded
          expr: '(100-(avg(rate(node_cpu{node="{{ NODE_NAME }}", mode="idle"}[60s]))*100))
> {{NODE_TH_MAX}}'
          for: 1m
        - alert: node_underloaded
          expr: '(100-(avg(rate(node_cpu{node="{{ NODE_NAME }}", mode="idle"}[60s]))*100))
< {{NODE_TH_MIN}}'
          for: 2m
        required: true
      scaling_rule:
        type: string
        description: pre-define scaling rule for VM CPU
        default: |
          if len(m_nodes) <= m_node_count:
            if node_overloaded and service_working:
              m_node_count+=1
            if node_underloaded:
              m_node_count-=1
          else:
            print('Transient phase, skipping update of nodes...')
        required: true
```

13.2 Appendix B: SAKER Use Case ADT

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - https://raw.githubusercontent.com/micado-scale/tosca/v0.7.3/micado_types.yaml

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

topology_template:
  node_templates:
    worker-node:
      type: tosca.nodes.MiCADO.CloudSigma.Compute
      properties:
        num_cpus: 2000
        mem_size: 4294967296
        vnc_password: xx
        libdrive_id: xx
        public_key_id: xx
        hv_relaxed: true
        hv_tsc: true
        nics:
        - vlan: xx
        context:
          append: no
          cloud_config: |
            write_files:
            - path: 'C:\xx\yy\zz.yaml'
              permissions: '0644'
              content: |
                ManagerHostName: xx.xx.xx.xx
                ManagerPort: yyyy
                WorkerPort: zzzz
                WorkingDirectory: C:\xx\yy\
                PriorityThreshold: 0
                MaximumNumberOfCores: 1
                UseBlobStore: True
                StorageHost: 'http://xx.xx.xx.xx'
                Tags: cloud
      interfaces:
        Occopus:
          create:
            inputs:
              interface_cloud: cloudsigma
              endpoint_cloud: https://xx/api/
      capabilities:
        host:
          properties:
            num_cpus: 2
            mem_size: 4 GB

  policies:
    - scalability:
        type: tosca.policies.Scaling.MiCADO
        targets: [ worker-node ]
        properties:
          sources:
          - 'x.x.x.x:xxxx'
          constants:
            MINNODES: 1
            MAXNODES: 100
          queries:
            REMAININGTIME: 'closest_deadline-time()'
            ERD: 'longest_ERD*60'
            ITEMS_W: 'waiting_replications'
            ITEMS_R: 'running_replications'
          min_instances: 1
          max_instances: '{{MAXNODES}}'
          scaling_rule: |
            print "ITEMS_W:",ITEMS_W
            print "ITEMS_R:",ITEMS_R
            if ITEMS_W>0 and REMAININGTIME>0:
              reqnodes = ceil(ERD/(REMAININGTIME/ITEMS_W))
              print "REQNODES (1): ",reqnodes
              reqnodes = min([reqnodes, ITEMS_W+ITEMS_R])
              print "REQNODES (2): ",reqnodes
              if reqnodes >= m_node_count:
                m_node_count = reqnodes
                print "NEW number of requested nodes:",m_node_count
            elif ITEMS_R==0:
                m_node_count = MINNODES
                print "NEW number of requested nodes:",m_node_count
```

13.3 Appendix C: Outlandish Use Case ADT

(This is a general ADT prepared by Outlandish, on which Audience Agency is based)

```
tosca_definitions_version: tosca_simple_yaml_1_0

imports:
  - https://raw.githubusercontent.com/micado-scale/tosca/v0.x.2/micado_types.yaml

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

topology_template:
  node_templates:
    nginxapp:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      properties:
        resources:
          requests:
            cpu: "200m"
        env:
          - name: NGINX_HOST
            value: "xx"
          - name: NGINX_PORT
            value: "8080"
          - name: FPM_HOST
            value: "xx"
          - name: FPM_PORT
            value: "9000"
        ports:
          - target: 8080
            type: NodePort
            nodePort: 32349
          - target: 9432
      artifacts:
        image:
          type: tosca.artifacts.Deployment.Image.Container.Docker
          file: outlandish/bedrock-web
          repository: docker_hub
      interfaces:
        Kubernetes:
          create:
            implementation: image

    wordpress:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      properties:
        resources:
          requests:
            cpu: "200m"
        env:
          - name: DB_HOST
            value: "db"
          - name: DB_NAME
            value: "xx"
          - name: DB_USER
            value: "xx"
          - name: DB_PASSWORD
            value: "xx"
          - name: WP_HOME
            value: "xx"
        ports:
          - target: 9000
      artifacts:
        image:
          type: tosca.artifacts.Deployment.Image.Container.Docker
          file: outlandish/bedrock
          repository: docker_hub
      interfaces:
        Kubernetes:
          create:
            implementation: image

    db:
      type: tosca.nodes.MiCADO.Container.Application.Docker
      properties:
        resources:
          requests:
            cpu: "200m"
        env:
          - name: MYSQL_ROOT_PASSWORD
            value: "xx"
          - name: MYSQL_USER
            value: "xx"
          - name: MYSQL_PASSWORD
            value: "xx"
          - name: MYSQL_DATABASE
            value: "xx"
        ports:
          - target: 3306
      artifacts:
        image:
          type: tosca.artifacts.Deployment.Image.Container.Docker
          file: mysql:5.7
          repository: docker_hub
      interfaces:
        Kubernetes:
          create:
            implementation: image

    worker_node:
      type: tosca.nodes.MiCADO.EC2.Compute
      properties:
        region_name: eu-west-2
        image_id: xx
        instance_type: t2.medium
        security_group_ids:
          - "xx"
      interfaces:
        Occopus:
          create:
            inputs:
              interface_cloud: ec2
              endpoint_cloud: https://ec2.eu-west-2.amazonaws.com
      capabilities:
        host:
          properties:
            num_cpus: 2
            mem_size: 2 GB
```

```
  outputs:
    ports:
      value: { get_attribute: [ nginxapp, port ]}

  policies:
    - scalability:
        type: tosca.policies.Scaling.MiCADO.VirtualMachine.CPU.node
        targets: [ worker_node ]
        properties:
          constants:
            NODE_TH_MAX: '66'
            NODE_TH_MIN: '33'
          min_instances: 1
          max_instances: 10

    - scalability:
        type: tosca.policies.Scaling.MiCADO.Container.connections.nginx
        targets: [ nginxapp ]
        properties:
          min_instances: 1
          max_instances: 10

    - scalability:
        type: tosca.policies.Scaling.MiCADO.Container.CPU.wordpress
        targets: [ wordpress ]
        properties:
          constants:
            SERVICE_NAME: 'wordpress'
            SERVICE_FULL_NAME: 'wordpress'
            SERVICE_TH_MAX: '60'
            SERVICE_TH_MIN: '25'
          min_instances: 1
          max_instances: 10

policy_types:

  tosca.policies.Scaling.MiCADO.Container.CPU.wordpress:
    derived_from: tosca.policies.Scaling.MiCADO
    description: base MiCADO policy defining data sources, constants, queries, alerts,
limits and rules
    properties:
      alerts:
        type: list
        description: pre-define alerts for container CPU
        default:
          - alert: wordpress_overloaded
            expr:
'avg(rate(container_cpu_usage_seconds_total{container_label_io_kubernetes_container_name="{
{SERVICE_FULL_NAME}}"}[30s]))*100 > {{SERVICE_TH_MAX}}'
            for: 30s
          - alert: wordpress_underloaded
            expr:
'avg(rate(container_cpu_usage_seconds_total{container_label_io_kubernetes_container_name="{
{SERVICE_FULL_NAME}}"}[30s]))*100 < {{SERVICE_TH_MIN}}'
            for: 30s
        required: true
      scaling_rule:
        type: string
        description: pre-define scaling rule for container CPU for WordPress/PHP service
        default: |
          if len(m_nodes) == m_node_count:
            if wordpress_overloaded and m_node_count > m_container_count:
              m_container_count+=1
            if wordpress_underloaded:
              m_container_count-=1
          else:
            print('Transient phase, skipping update of containers...')
        required: true

  tosca.policies.Scaling.MiCADO.Container.connections.nginx:
    derived_from: tosca.policies.Scaling.MiCADO
    description: base MiCADO policy defining data sources, constants, queries, alerts,
limits and rules
    properties:
      alerts:
        type: list
        description: pre-define alerts for container writing vs waiting connections
        default:
          - alert: nginx_overloaded
            expr: '(avg(rate(nginx_vts_main_connections{status=~"writing"}[2m])) -
avg(rate(nginx_vts_main_connections{status=~"waiting"}[2m]))) < 0'
            for: 30s
          - alert: nginx_underloaded
            expr: '(avg(rate(nginx_vts_main_connections{status=~"writing"}[2m])) -
avg(rate(nginx_vts_main_connections{status=~"waiting"}[2m]))) > 0'
            for: 2m
        required: true
      scaling_rule:
        type: string
        description: pre-define scaling rule for connections to NGINX
        default: |
          if len(m_nodes) == m_node_count:
            if nginx_overloaded and m_node_count > m_container_count:
              m_container_count+=1
            if nginx_underloaded:
              m_container_count-=1
          else:
            print('Transient phase, skipping update of containers...')
        required: true

  tosca.policies.Scaling.MiCADO.VirtualMachine.CPU.node:
    derived_from: tosca.policies.Scaling.MiCADO
    description: base MiCADO policy defining data sources, constants, queries, alerts,
limits and rules
    properties:
      alerts:
        type: list
        description: pre-define alerts for VM CPU
        default:
          - alert: node_overloaded
            expr: '(100-(avg(rate(node_cpu{group="worker_cluster",mode="idle"}[60s]))*100))
> {{NODE_TH_MAX}}'
            for: 1m
          - alert: node_underloaded
            expr: '(100-(avg(rate(node_cpu{group="worker_cluster",mode="idle"}[60s]))*100))
< {{NODE_TH_MIN}}'
            for: 1m
        required: true
      scaling_rule:
        type: string
        description: pre-define scaling rule for VM CPU
        default: |
          if len(m_nodes) <= m_node_count and m_time_since_node_count_changed > 60:
            if node_overloaded:
              m_node_count+=1
            if node_underloaded:
              m_node_count-=1
          else:
            print('Transient phase, skipping update of nodes...')
        required: true
```