Received: 11 May 2020 | Revised: 15 September 2020 | Accepted: 5 October 2020

DOI: 10.1002/cpe.6093

SPECIAL ISSUE PAPER

WILEY

Cloud apps to-go: Cloud portability with TOSCA and MiCADO

James DesLauriers[1] | Tamas Kiss[1] | Resmi C. Ariyattu[1] | Hai-Van Dang[1] | Amjad Ullah[1] | James Bowden[2] | Dagmar Krefting[2,3] | Gabriele Pierantoni[1] | Gabor Terstyanszky[1]

[1]Research Centre for Parallel Computing, University of Westminster, London, UK

[2]Hochschule für Technik und Wirtschaft, Berlin, Germany

[3]University Medical Center, Göttingen, Germany

Correspondence
James DesLauriers, Research Centre for Parallel Computing, University of Westminster, London, UK.
Email: j.deslauriers@westminster.ac.uk

Summary

As cloud adoption increases, so do the number of available cloud service providers. Moving complex applications between clouds can be beneficial—or other times necessary—but achieving this so-called cloud portability is rarely straightforward. This article presents the adoption of OASIS TOSCA, a standard in the declarative description of cloud applications, to encourage and facilitate cloud portability in MiCADO, an application-level multi-cloud orchestration and auto-scaling framework. The interface to MiCADO is an Application Description Template, which draws from the TOSCA specification to describe an application in MiCADO. The generic design of these templates is presented and their applicability for achieving portability between different container and cloud environments is analysed and evaluated. A proof-of-concept where MiCADO serves as the deployment and execution engine for a Science Gateway in Sleep Healthcare is then described. In this proof-of-concept, MiCADO facilitates the deployment of a complex healthcare application, which is then moved from one cloud service provider to another with only minimal changes to the template which originally described it. This TOSCA-based approach to templates in MiCADO encourages movement between clouds by making cloud portability more approachable.

KEYWORDS

cloud, MiCADO, multi-cloud, orchestration, portability, TOSCA

1 | INTRODUCTION

The cloud computing model is attractive for many research, public sector, and enterprise organizations. Having flexible, on-demand access to computing resources and services can result in significant cost and time savings. Moreover, large, upfront capital investments can be replaced by day-to-day operational costs over a longer period of time. With cloud adoption on the rise, the number of cloud service providers is increasing and is providing more choice and flexibility for running workloads off-premise. There are clear advantages in bursting or migrating to different cloud offerings, whether it be to take advantage of specific resources or services in the short-term, or to relocate or handover a project on a permanent basis.

However, achieving portability between clouds is not trivial. Here, vendor lock-in is a real threat: after investing in the necessary training to become comfortable with the services of one cloud provider, learning those of several others can seem an onerous, costly task. Any manual approach to portability requires familiarity with different cloud provider platforms, services, and APIs, and, as with most manual tasks, it can be a repetitive and time-consuming exercise. One way to facilitate portability from cloud to cloud is to focus on automating the steps involved in provisioning and configuring the necessary cloud resources as well as deploying the desired application or services. Most widely used cloud providers offer

mechanisms for exactly this purpose (AWS CloudFormation,[1] OpenStack HEAT[2]) but these proprietary languages are again encouraging of vendor lock-in and do nothing for solving the issue of portability.

Fortunately, cloud-agnostic approaches to automation do exist and are provided by a wide range of tools that generally fall under the monikers of DevOps, Infrastructure-as-Code or Configuration Management (Terraform,[3] Chef,[4] Ansible,[5] Docker, Docker Swarm,[6] and Kubernetes,[7] to list a few). These tools act on declarative or imperative templates to provision, configure and deploy an application and its environment in the cloud. However, the configuration languages, syntactic approaches, complexity, and compatibility of these templates can vary immensely and the integration between tools is not always straightforward. The savings made by not having to learn the proprietary APIs of several cloud platforms are quickly offset by the perhaps more challenging task of learning the various languages, parameters, and structures of this tool set.

Both the manual and automated approaches can present their own challenges for research groups, independent developers, smaller institutions, or other organizations that lack the cloud-specific skills or training investment to become familiar with multiple cloud interfaces, or efficiently use the available set of automation tools. Additionally, once basic cloud migration is realized, these groups may struggle with getting the same benefits of the newly adopted cloud platform due to a lack of availability or familiarity with services providing features such as scalability, flexibility, and security.

As early as 2013, the Organization for the Advancement of Structured Information Standards (OASIS)[8] identified the challenges presented to cloud portability and template re-usability and began work on advancing a new standard in describing cloud applications called Topology and Orchestration Specification for Cloud Applications (TOSCA).[9] Since 2013, the TOSCA standard has evolved into a detailed reference specification, and has been adopted by a number of open-source projects which have developed tools to read and orchestrate TOSCA-based templates. OASIS provides their own rendering of this specification in YAML (YAML Ain't Markup Language) called TOSCA Simple Profile,[10] which defines *normative TOSCA types* for describing components of a cloud deployment, from servers, software, networks, and volumes to the various policies which will govern the application life-cycle.

A typical template in TOSCA Simple Profile will describe a cloud and application topology and its components as *normative types* which a conforming TOSCA Orchestrator can then provision, configure, and deploy through communications with a cloud provider interface and with scripts that manage the application state at runtime. However, this approach limits the user to the cloud service providers and script interfaces supported by a given TOSCA Orchestrator.

To put forth a problem statement: Current solutions utilizing TOSCA do well to introduce a generic interface and encourage re-usability and portability in their templates but do not take advantage of the full set of cloud agnostic approaches to automation that can benefit users transitioning from cloud to cloud. As it stands, seekers of cloud portability must choose between DevOps automation and its multitude of tools, or a specific implementation of TOSCA, tied to a fixed set of cloud service providers and orchestration tools.

This article, significantly extending the concepts described in Reference 11, presents the unique approach to TOSCA taken by the Microservices-based Cloud Application-level Dynamic Orchestrator (MiCADO) framework.[12] MiCADO uses a TOSCA-based Application Description Template (ADT)[13] as an abstraction layer over a modular, changeable set of automation tools for cloud provisioning, configuration management, and application deployment and execution. Here, the user is limited only by the available set of tools on the market, rather than by MiCADO itself. The MiCADO framework enables cloud portability through its ability to automate all aspects of deploying, executing, and managing an application on a selected cloud. Highly re-usable templates mean keeping the configuration and overarching policies of a complex application intact, while being able to swap out the underlying cloud resources for a different cloud provider. Being a modular framework, the automation tools which drive orchestration in MiCADO can also be swapped in and out to provide access to other cloud service providers, or even different deployment environments.

In a proof of concept, we demonstrate how MiCADO is being used to facilitate cloud portability in the Horizon2020 EU Project ASCLEPIOS (Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare),[14] providing solutions for securing healthcare data in a multi-cloud environment. Three healthcare application demonstrators from different healthcare providers and a cloud test-bed featuring a mix of four private or public clouds make up the project use-cases. When implementing the ASCLEPIOS demonstrators, we author ADT templates for three different complex healthcare demonstrator applications, to be deployed alongside a bespoke set of security components being developed in the ASCLEPIOS project. These templates are then re-used with minor changes to achieve portability between the various clouds in the testbed.

The rest of this article is structured as follows. Section 2 provides a short introduction to MiCADO and its modular design, while Section 3 offers an introduction to TOSCA and the MiCADO ADT. In Section 4, we build the base TOSCA types for cloud resources and containerized applications and in Section 5 demonstrate how those types facilitate a change in orchestrator or cloud provider. Section 6 offers the proof-of-concept where one ADT is re-used with only minor changes to deploy a healthcare demonstrator and security components to a variety of different private and public clouds. Finally we conclude with a look at related work in cloud portability and the adoption of TOSCA by industry and academia.

2 | MICADO

MiCADO (also marketed as MiCADOscale[15]) is an application-level multi-cloud orchestration and auto-scaling framework. Developed in the European COLA Project (Cloud Orchestration at the Level of Application),[16] it set out to address the issues of vendor lock-in, security and

scalability. A variety of cloud middleware is supported by MiCADO, including that of both large and small commercial cloud providers such as Microsoft Azure and AWS EC2, as well as private clouds such as OpenNebula and OpenStack. MiCADO is entirely open source (hosted at `github.com/micado-scale`) and implements a microservices architecture in a modular way. The modular design supports varied implementations where any of the components can easily be replaced with a different realization of the same functionality. The concept of MiCADO is described in detail in Reference 12. In this section, a high-level overview of the framework is provided to explain its architecture, building blocks, and modular implementation.

One of the major applications of MiCADO is as an embedded application deployment and executor service in Science Gateways, as it is described in References 17 and 18. Its current role in the Horizon 2020 ASCLEPIOS Project is to support the deployment and execution of a set of healthcare application demonstrators alongside a set of security components developed in the project. In this way, MiCADO becomes the deployment tool behind the web or desktop interfaces of the healthcare applications, ensuring that they are correctly and efficiently deployed to and managed on a suitable set of cloud resources.

The design of MiCADO for cloud portability was based on two major principles. First, there was the need for a generic orchestration framework providing support for launching and managing a variety of applications in the cloud. The framework supports a mix of public, private, and community clouds and provides flexibility at the application level, regardless of the underlying cloud. This includes automated deployment and optimized run-time orchestration with features such as automated scaling[19] and enhanced security.

Second, a single generic interface to the framework was required. This interface acts as an abstraction layer over the various underlying components of the framework and describes the application, its cloud resources and any policies which govern performance, cost, security, or other non-functional application requirements. This generic interface applies the concept of Infrastructure-as-Code (IaC), the name given to the programmatic way in which IT infrastructure can be written either as the steps which will realize a desired state (imperative) or simply as the description of its desired state (declarative). Here, the declarative approach is taken—a template describes the complete and final state of the application that should be deployed and the orchestration and execution engines that process the template determine the necessary steps.

Many of the tools that make up MiCADO fall under the heading of DevOps and are traditionally used in industry to improve the software development lifecycle. They offer cloud resource provisioning, environment configuration, application deployment, and monitoring—unique features which MiCADO can piece together and leverage to provide deployment, scalability, and runtime management in a highly automated way. It is the pairing of these various tools with a generic interface that establishes the footing for cloud portability in MiCADO.

The high-level architecture of MiCADO is presented in Figure 1. MiCADO consists of two main logical components: Master node and Worker node. The MiCADO Master node is deployed using an Ansible Playbook, itself a DevOps tool for configuring and deploying environments on a remote host. Once the MiCADO Master is running, the submitter component can take an ADT (explained in detail in Section 3) describing the application's topology and the required scaling and security policies as input. Based on this input, the Cloud Orchestrator creates the necessary virtual machines in the cloud as MiCADO Worker nodes and the Container Orchestrator deploys the application's microservices in Docker containers on these nodes. After deployment, the MiCADO Monitoring System monitors the execution of the application and the Policy Keeper performs scaling decisions based on the monitoring data and the user-defined scaling policies. Optimizer is a background microservice performing long-running calculations on demand for finding the optimized setup of both cloud resources and container infrastructures.

Currently there are various implementations of MiCADO based on its modular architecture which enables changing and replacing its components with different tools and services. As a Cloud Orchestrator, the latest implementation of MiCADO can utilize either Occopus[20] or Terraform, which are both capable of launching virtual machines on various private or public cloud infrastructures. However, as the clouds supported by these two orchestrators differ, supporting both allows MiCADO to deploy a wider variety of targeted cloud resources. For Container Orchestration, earlier versions of MiCADO applied Docker Swarm, which was later replaced by Kubernetes. The monitoring component is based on Prometheus,[21]

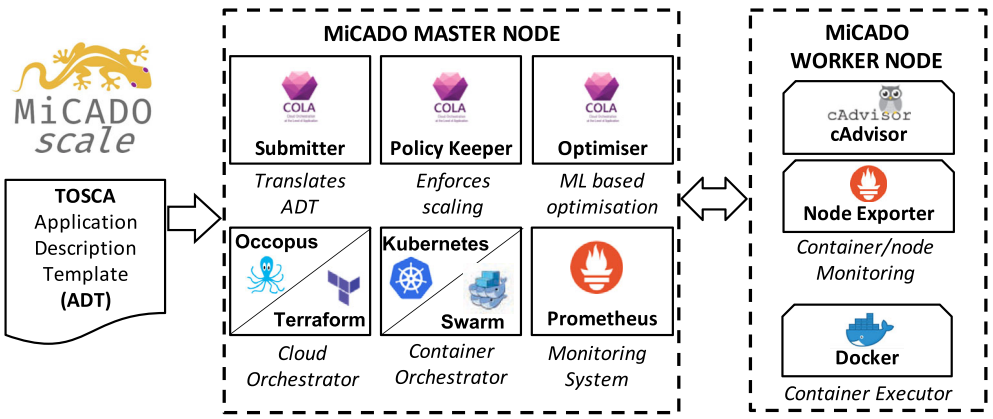

FIGURE 1 High-level architecture of MiCADO

a lightweight, low resource consuming, but powerful monitoring tool. The MiCADO Submitter,[13] Policy Keeper[22] and Optimizer components were custom implemented for MiCADO during the COLA project.

3 | TOSCA AND APPLICATION DESCRIPTION TEMPLATES

TOSCA is a reference specification for describing the full topology and operational behavior of an application running in the cloud. It can be described as declarative IaC, being that it describes simply the desired state of applications in the cloud, rather than the steps that realize that state. Topology in TOSCA is defined as a set of connected building-blocks called *nodes*, which represent components such as the software, virtual machines, storage volumes, and networks that make up the application. The operational behavior is managed by defined *relationships* between the above components and through lifecycle management *interfaces* in the form of scripts, configurations, or API invocations. *Policies* for scaling, monitoring, or placement can be defined to manage the application behaviour at runtime.

The various nodes, relationships, interfaces, and policies for use within a system are pre-defined as *types* with default properties, requirements, capabilities, and input constraints. These types can be further extended into child types, or they can be referenced in the *topology template* which declaratively describes the desired state of the application components in a final, ready to submit *TOSCA Service Template*. TOSCA types can be defined directly in a Service Template, or they can be prepared in an external TOSCA Definitions file and later imported into a Service Template for use. There are many good resources for TOSCA, and a good starting point is the current standard itself—TOSCA Simple Profile in YAML Version 1.3.[10]

MiCADO applies TOSCA-based ADTs for writing IaC to define the cloud topology (containers and virtual machines) and policies for a given application. This template was designed for MiCADO based on a TOSCA Service Template and is derived from version 1.0 of the TOSCA Simple Profile. A stripped-back example of a basic ADT describing a compute node, web server and a simplified scaling policy can be seen in Figure 2. All TOSCA Service Templates, and by association all ADTs, must begin with the TOSCA version. This ADT additionally defines a list of *imports*—external TOSCA Definitions files that contain custom pre-defined types, a map of *repositories* that can be referenced throughout the template, and a *description* of the template. The Topology Template then defines *node templates*, which represent the components of the application, and *policies*, which will govern the application at runtime.

In the example ADT in Figure 2, the first node template describes a virtual machine of the OpenStack compute type with the name *my-virtualmachine* and defines a relevant property. An application container named *my-app* of the Docker type defines a relationship with *my-virtualmachine*, identifying it as the required host for this Docker container. Finally a scalability policy is defined using the scaling type. It targets the previously defined Docker container node by name and sets the value of a required property.

There are two sections to an ADT—one to describe the cloud infrastructure and the application itself, and a second to describe the policies which will govern the application at runtime. These policies may include scalability, monitoring, or other non-functional requirements as were discussed previously in the proceedings of the 10th International Workshop on Science Gateways.[13] The rest of this article focuses only on the first of the two ADT sections: that related to the description of the application and the cloud resources which will support it—with a special focus on how that description encourages reuse and facilitates portability from cloud to cloud.

Several considerations were taken in the design of the ADT interface. It needed to support re-use and portability, as well as compatibility with MiCADO's modular framework. This presented challenges for designing an appropriate template since a single application might need to be re-used not only on a different cloud but also by a different cloud or container orchestrator entirely. The updated approach to authoring and applying declarative IaC using TOSCA-based ADTs is presented in this article. The approach facilitates the portability of an application between orchestrators and cloud providers alike, and encourages reuse of previous application and cloud resource descriptions. The ADT format provides more flexibility and control for those template authors who understand the underlying technologies of the respective components they are describing. At the same time,

```
1  tosca_definitions_version: tosca_simple_yaml_1_0
2
3  imports: [ micado_custom_types.yaml ]
4  repositories:
5    docker_hub: https://hub.docker.com/
6
7  description: Compute node, web server and scaling
8
9  topology_template:
10   node_templates:
11     my-virtualmachine:
12       type: tosca.nodes.MiCADO.Compute.Openstack
13       properties:
14         flavor_name: m1.medium
15     my-app:
16       type: tosca.nodes.MiCADO.Container.Application.Docker
17       properties:
18         image: uow-cpc/nginx:latest
19         environment: [ NGINX_HOST=foobar.com ]
20       requirements:
21       - host: my-virtualmachine
22
23 policies:
24 - scalability:
25     type: tosca.policies.MiCADO.Scaling
26     targets: [ my-app ]
27     properties:
28       max_instances: 4
```

FIGURE 2 Sample ADT describing a compute instance, web server, and simple scaling policy

the ADT structure still features variable levels of abstraction, which make it possible for users without component-specific knowledge to author templates and deploy applications in MiCADO.

4 | A NOVEL APPROACH TO TOSCA

The rendering of TOSCA used in MiCADO ADTs is a further simplification of the so-called normative TOSCA prescribed by the OASIS TOSCA working group in TOSCA Simple Profile in YAML. This is in large part due to the environment in which MiCADO orchestrates applications and cloud resources. For a MiCADO deployment, the assumption is that the application or its microservices have already been packed into one or more container images which are all in a ready-state. These containers can be customized at deployment time by passing in various parameters and arguments or mounting the necessary configurations, as supported by the container runtime. When container orchestration is not possible, for example, with some Windows applications, MiCADO also supports a so-called VM-only deployment, which again makes the assumption that the attached virtual machine image contains the necessary libraries and binaries and that the application is in a ready-state. Both types of deployments can see their virtual machines further customized through cloud contextualization by executing commands at start-up through custom user data scripts such as cloud-init.[23] This contextualization support exists across all supported clouds in MiCADO, which ensures a consistent compute environment when moving between cloud service providers.

Normative TOSCA, for example, would define up to four different nodes in order to describe the deployment of a logical database in a database management system (DBMS) which is itself running on the virtual machine image (software component) of a compute instance. An ADT for MiCADO could accomplish the same using only two node types. Since the virtual machine image and compute instance in MiCADO are always considered together, and a logical database and its DBMS are considered as a single container, each respective pair can be defined by a single non-normative node type. The defined types, structures, and syntax in an ADT still follow the TOSCA specification, but because of the already-configured nature of applications and virtual machines in MiCADO, an ADT can describe the same application with fewer overall nodes. The node types used for this example can be seen in Table 1.

As well as being different from normative TOSCA, the approach taken to adopting TOSCA for the MiCADO ADT is also inherently different from the approach taken by other frameworks and research activities described in the related works in Section 7. In following with other projects that had also implemented TOSCA-based languages, the early ADTs of MiCADO defined TOSCA types for applications (in containers) and cloud resources (virtual machines) which were all strongly related to their respective orchestrators (Occopus and Docker Swarm, for example). This did not cater well to re-usability, since a change of orchestrator in the implementation of MiCADO meant a new set of TOSCA types had to be defined, even though the basic unit the orchestrator was acting on might not have changed. As an example, both of the Terraform and Occopus cloud orchestration tools are able to provision an EC2 instance. To avoid this issue and better encourage re-usability, ADT types decouple the orchestrator from the node, leaving the node type to describe the compute or container resource at a generic level and ignoring the orchestration tool entirely.

This approach meant that an ADT could simply define two broad types of nodes to cover the two main orchestrated components in MiCADO—one for virtual machines (compute), and one for containers (applications). This gave us a base node type for each, which could be extended to support a variety of cloud resources from different providers, or different container runtimes, as can be seen in Table 2.

The next step was to define the orchestration tool within the MiCADO framework that would act on these resources in order to start and configure them and further manage them at runtime. To this end, we leveraged TOSCA interface types. In the TOSCA specification, an interface

TABLE 1 TOSCA normative types compared with MiCADO ADT types for sample deployment of a database

TOSCA Simple Profile, normative	MiCADO ADT, non-normative
tosca.nodes.Compute	tosca.nodes.MiCADO.Compute.EC2
tosca.nodes.SoftwareComponent	tosca.nodes.MiCADO.Container.Application.Docker.MySQL
tosca.nodes.DBMS	
tosca.nodes.Database	

TABLE 2 Cloud and container resources represented as specific types in a MiCADO ADT

Virtual machine types	Container types
tosca.nodes.MiCADO.Compute **(base)**	tosca.nodes.MiCADO.Container.Application **(base)**
tosca.nodes.MiCADO.Compute.OpenStack	tosca.nodes.MiCADO.Container.Application.Docker
tosca.nodes.MiCADO.Compute.EC2	tosca.nodes.MiCADO.Container.Application.rkt
tosca.nodes.MiCADO.Compute.Azure	tosca.nodes.MiCADO.Container.Application.crio

Cloud interfaces	Container interfaces
tosca.interfaces.MiCADO.Occopus	tosca.interfaces.MiCADO.Kubernetes
tosca.interfaces.MiCADO.Terraform	tosca.interfaces.MiCADO.Swarm

TABLE 3 Orchestration tools represented as interface types in a MiCADO ADT

```
tosca_definitions_version: tosca_simple_yaml_1_0

node_types:
  tosca.nodes.MiCADO.Container.Application.Docker:
    derived_from: tosca.nodes.MiCADO.Container.Application
      properties:
        ...
      interfaces:
        Swarm:
          type: tosca.interfaces.MiCADO.Swarm
        Kubernetes:
          type: tosca.interfaces.MiCADO.Kubernetes
```

FIGURE 3 A custom type definition file describing potential orchestrators for Docker containers using TOSCA interfaces

should be defined for each node and takes the responsibility for overseeing the lifecycle of that node. The so-called default Standard interface of TOSCA uses implementation scripts (which can include Python or bash scripts or Chef or Puppet configurations) to manage that lifecycle through four main stages: create, configure, start, and stop. TOSCA also provides an input mechanism to feed additional parameters into these scripts at deployment time, which can be defined directly in the inputs field of an interface in the TOSCA template.

In MiCADO, these lifecycle stages are handled by whichever respective orchestrator is responsible for that node so there is no requirement to associate those stages with a script or piece of automation code as is done in normative TOSCA. However, it is still necessary to pass information from the ADT to the relevant orchestrator so it knows which nodes it is responsible for. The interface determines the responsible orchestration tool and allows for additional custom parameters to be passed to that orchestrator via the inputs field. The possible interface types inherit from the **tosca.interfaces.MiCADO** base type and are shown in Table 3.

To attach these interfaces to a node type, the node definition can specify which orchestrators it supports, as seen in the truncated definition of a Docker container node in Figure 3. In this figure, a new node type is defined for the Docker container type, which extends, or *derives from* the base container type. Required properties could be set here but have been omitted from this example. Possible cloud or container orchestrators for this node are defined under the *interfaces* key, and are linked to their own type definitions. The approach to identifying which options or parameters are set under the *properties* key, and which are set in the orchestrator type is discussed in Section 5. This node type definition for a Docker container would be found inside a TOSCA Definitions file that could later be imported at the top of an ADT (see Figure 2).

The node and interface types described above are the base for describing cloud resources in all MiCADO ADTs. When an ADT is sent to the MICADO Submitter, the template is parsed and the information for each node is translated to the native format of the indicated orchestration tool running in MiCADO. The newly translated templates, one for each activated orchestration tool, are used to deploy, update, or delete the application in MiCADO. The translate, execute, update, and delete functionality are provided by a modular set of adaptors acting as plugins to the Submitter, one for each of the underlying orchestration tools in MiCADO. New adaptors can be written to support new orchestration tools as they are added to MiCADO, taking input from the TOSCA-based ADT and generating template files in native formats.

5 | TOSCA SUPPORTING PORTABILITY IN MICADO

The adoption of a TOSCA-based language as the interface to MiCADO facilitates application portability on several levels, which are discussed in this section. First is portability at the level of the container orchestration environment - an application can be ported from running in one environment to another (for example, from Docker Swarm to Kubernetes). Second is at the level of cloud orchestration—in the same way an application can be moved between container orchestration environments, so too can it be managed by an entirely different cloud orchestration tool (for example, Occopus to Terraform). Third, and also the focus of the proof of concept which follows in Section 6, is portability between cloud service providers.

5.1 | Swapping container orchestration environments

Before MiCADO supported Kubernetes, Docker Swarm was featured as the primary container orchestrator. With the decision to support Kubernetes came an opportunity to test the benefits of the TOSCA-based approach to the ADT. Both Docker Swarm and Kubernetes support orchestrating

```
nginx-server:
  type: tosca.nodes.MiCADO.Container.Application.Docker
  properties:
    container_name: nginx
    environment: [ NGINX_PORT=80 ]
    entrypoint: /code/entrypoint.sh
    ...
  interfaces:
    Swarm:
      create: {}
```

```
nginx-server:
  type: tosca.nodes.MiCADO.Container.Application.Docker
  properties:
    container_name: nginx
    environment: [ NGINX_PORT=80 ]
    entrypoint: /code/entrypoint.sh
    ...
  interfaces:
    Kubernetes:
      create:
        inputs:
          kind: Deployment
```

FIGURE 4 A Docker container defined with compose specification property names, orchestrated by Swarm (left) and Kubernetes (right)

Docker containers, meaning that the basic component to orchestrate would remain unchanged when switching between them. Since the ADT was designed to be an abstraction layer over the underlying components, even though a fairly major change of component was underway, the interface to the user—especially the section describing the container itself—could remain unchanged.

When the ADT separated the logic of the orchestrated component from the orchestrator (as discussed in Section 4), it was necessary to determine which properties or options belong to the application container and which belong to the container orchestrator. To define the generic set of options that a user can set in the properties section of a Docker container node type, we made a review of the options available and inputs required when orchestrating a Docker container with each of Docker Swarm and Kubernetes. Any options which were clearly related to orchestration, such as scheduling or update strategies, would become the inputs for the TOSCA interface related to that specific orchestrator. The remaining options, more closely related to the properties of the container itself, became the TOSCA node properties of the Docker container type. The naming and grammar of these properties varied slightly between container orchestrators, so to support portability, MiCADO understands the nomenclature of both major container orchestration platforms. In contrast, the options tightly related to orchestration, now the inputs in the TOSCA interfaces section of the definition, are only supported in the native format and naming convention of the selected orchestrator.

Figure 4 provides an example of the portability and extended support offered by this approach. In this example, a single simple NGINX container is defined in an ADT and then orchestrated by each of Swarm and Kubernetes. The generic container properties (under properties in the definitions) are flexible in that they can be expressed using any supported orchestrator's nomenclature, and then scheduled by any supported orchestrator. Here, the generic container definition is the same on both sides of the figure, and uses Compose specification property names.[24] When selecting the orchestrator (under interfaces), other orchestrator-specific options can be specified as inputs, so long as they match the naming and grammar of that specific orchestrator. In the example, Swarm is orchestrating on the left and requires no additional parameters. On the right, Kubernetes is orchestrating, and an additional parameter specifying the workload kind is passed in. Because this parameter is specific to the Kubernetes API,[25] orchestrator-specific grammar is required here.

On the implementation side, leveraging the modularity of MiCADO was relatively straightforward. The configuration of Docker Swarm and its visualizer component were removed from the Ansible playbook responsible for building the MiCADO Master, and the installations of the Kubernetes core components and dashboard were added in their place. MiCADO worker nodes were instructed to join a Kubernetes cluster instead of a Swarm cluster as they had done previously. The internal security components of MiCADO, such as the application-level firewall, presented special challenges because of their tight integration with the container environment, so the security enablers were rewritten to support the Kubernetes environment and its networking approach. Finally, a MiCADO Submitter adaptor was introduced for translating to Kubernetes manifests and managing them via the kubectl command, and a new Policy Keeper handler was added for scaling those Kubernetes workloads.

5.2 | Adding a cloud orchestration tool

During most of its development, provisioning of cloud resources in MiCADO was handled by the Occopus cloud orchestration engine. Occopus supports provisioning compute instances with a number of different cloud providers, such as the public Amazon Web Services (AWS), some commercial European cloud offerings such as CloudSigma[26] and CloudBroker,[27] and private infrastructures based on OpenStack or OpenNebula. Resources other than compute instances, including AWS S3 object storage or Lambda serverless functions, as well as compute on other large public clouds such as Microsoft Azure, Google Cloud and Oracle Cloud are not supported by Occopus. To extend the capabilities of MiCADO with a wider range of cloud resources on a more complete range of cloud service providers, Terraform was introduced as second, alternative cloud orchestration tool. Terraform is one of the most widely used tools for the programmatic provisioning of cloud resources, and supports AWS, Azure, Google and Oracle clouds, among many others. Plugins called providers—developed and maintained by the Terraform community—are continually adding support for more cloud service platforms, and more resources within those clouds.

```
compute-instance:
  type: tosca.nodes.MiCADO.Compute.EC2
  properties:
    ...
    region_name: eu-west-2
    image_id: ami-006a0174c6c25ac06
    instance_type: t2.small
  interfaces:
    Occopus:
      create:
        inputs:
          endpoint: https://ec2.eu-west-2.amazonaws.com
```

```
compute-instance:
  type: tosca.nodes.MiCADO.Compute.EC2
  properties:
    ...
    region_name: eu-west-2
    image_id: ami-006a0174c6c25ac06
    instance_type: t2.small
  interfaces:
    Terraform:
      create: {}
```

FIGURE 5 An EC2 compute instance defined with Occopus property names, orchestrated by Occopus (left) and Terraform (right)

The option for running Terraform, Occopus, or both together was added as a user option at deployment time of the MiCADO Master. Since MiCADO could support running both cloud orchestrators in parallel, the choice between cloud orchestrator had to be supported at the level of the ADT, so that either Occopus or Terraform could be set for different cloud resources at deployment time. Just as it did with container orchestrators, the same TOSCA-based approach to MiCADO ADTs would benefit portability between cloud orchestrators, and encourage re-use of TOSCA definitions featuring the same unit of cloud resource—in this case, basic compute instances.

Given that both Occopus and Terraform support the provisioning of compute instances with the AWS Elastic Compute Cloud (EC2) service, EC2 compute nodes in MiCADO serve as a good example of the flexibility that comes with the previously described approach to the ADT. In the same way we determined properties for a basic Docker container in the previous subsection, the required TOSCA properties for a simple EC2 compute node were determined to be those most strongly related to the compute instance itself, such as instance type, virtual machine image, and security groups. On the other hand, TOSCA interface inputs for each of Occopus and Terraform were chosen as those options or parameters that were more closely related to the orchestration tool, for example, the EC2 endpoint, which in the case of Terraform is discovered automatically, but in the case of Occopus must be defined explicitly. Again, the adaptors within the MiCADO Submitter would support the naming conventions of either Occopus or Terraform in the TOSCA node properties, but would support only the native orchestrator format and naming in the TOSCA interface inputs.

Once the properties related to the EC2 instance were separated from those related to the cloud orchestrator, a TOSCA node defining an EC2 compute instance could be orchestrated by either Occopus or Terraform simply by modifiying the interface attached to that node. This is shown in Figure 5, where the same EC2 compute instance is defined and then orchestrated by each of Occopus and Terraform. On both sides of the figure, the same compute instance is defined, using the same property names. The only change is to the interfaces section where, on the left, Occopus orchestrates and requires an additional parameter, and on the right, Terraform orchestrates, which requires no additional parameters.

With regards to the implementation, the work in adding support for Terraform was straightforward—more so than switching container orchestrators, since the security enablers were not so tightly coupled to cloud orchestration. New tasks were added to the Ansible playbook for the installation and configuration of Terraform, with the setup of both Terraform and Occopus being optionally set when deploying the MiCADO Master node. A new adaptor was implemented in the MiCADO Submitter for translating, executing, updating, and deleting Terraform plans, and a new handler was added to the Policy Keeper for scaling Terraform resources up and down.

5.3 | Moving to a different cloud service provider

As well as supporting portability between orchestration environments and tools, taking a TOSCA-based approach to ADTs in MiCADO also enables and encourages portability between cloud service providers. Portability is one of the primary aims of the TOSCA specification and so by simply adopting TOSCA, the ADT became immediately more supportive of portability. Aspects of MiCADO itself, as well as the specific design of the ADT, further facilitate application portability between clouds.

The ways a TOSCA-based approach to the ADT encourages portability between cloud providers are best seen through a wide lens. An ADT describes a complex application, often implemented in a microservices architecture made up of many different containers, spread across multiple different compute instances, each with their own individual configurations and relationships with the other components in the infrastructure. When moving to a new cloud service provider, the only necessary modifications within that ADT are to the definitions of resources which need provisioning by the cloud orchestrators. Definitions of application components such as containers, relationships, and additional configurations can remain unchanged.

MiCADO ensures these application components can deploy, execute, and perform consistently because of the containerized environment it supports. It provides further assurance of a consistent environment by supporting cloud contextualization of virtual machines directly in the ADT.

```
my-app:
  type: tosca.nodes.MiCADO.Container.Application.Docker
  ...
  requirements:
  - host: compute-instance

compute-instance:
  type: tosca.nodes.MiCADO.Compute.EC2
  properties:
    region_name: eu-west-2
    ...
      cloud_config: |
        runcmd:
        - echo never > /sys/kernel/hugepage/enabled
  interfaces:
    Terraform:
      create: {}
```

```
my-app:
  type: tosca.nodes.MiCADO.Container.Application.Docker
  ...
  requirements:
  - host: compute-instance

compute-instance:
  type: tosca.nodes.MiCADO.Compute.Azure
  properties:
    resource_group_name: micado-scale
    ...
      cloud_config: |
        runcmd:
        - echo never > /sys/kernel/hugepage/enabled
  interfaces:
    Terraform:
      create: {}
```

FIGURE 6 Truncated descriptions of an application and compute instances in Amazon EC2 (left) and Microsoft Azure (right), both orchestrated by Terraform. Application description, relationships, cloud contextualization, and interfaces remain unchanged

Re-deployment of an ADT will build the same application environment time and time again, even across different cloud service provider compute instances.

Swapping between cloud or container orchestrators had a low impact on the ADT since only the TOSCA interface within a given TOSCA node definition needed modifying. The node definition itself needs no modification because the basic component it describes remains unchanged (a Docker container and an EC2 compute instance in the examples in Sections 5.1 and 5.2). Changing between cloud providers requires a more substantial change to an ADT—a new TOSCA node definition for each cloud service provider is required, even for cloud resources of the same type. A basic compute instance for example—which every major cloud has support for—can require vastly different inputs for provisioning depending on the cloud platform. Certain cloud providers require project names or identifiers, or explicitly defined networks or subnets, while other clouds are able to rely on default values for such parameters. The TOSCA node definition properties for that compute instance will be different, and unique to the cloud provider. When moving an application to a new cloud service provider, a new TOSCA node definition for that cloud provider's basic compute instance will need authoring.

However, because of the benefits conferred by TOSCA and our adoption of it in the ADT, many parts of the definition of that compute instance can enjoy re-use. The TOSCA interface of the node definition should need little to no changes if the described cloud orchestrator supports the new cloud service provider. Cloud contextualization, where a virtual machine is further configured at runtime by scripts such as cloud-init, is supported by default across all supported clouds in MiCADO, and requires no changes when moving to a new cloud. Relationships, such as the requirement of an application container to be hosted by a specific compute node, can also stay unchanged. These minor changes are best visualized in Figure 6 where only the TOSCA node type, as well as the properties specific to the cloud service provider need modifying. In these examples, the sample application is represented with a single Docker container node to be hosted on a virtual machine named *compute-instance* and remains unchanged. The definition of the compute instance inherits from different types, with Amazon EC2 compute on the left, and Azure compute on the right, and therefore the properties required are different. The contextualization section, seen here as an additional *cloud config* command to execute at runtime, and the interfaces section are the same on both sides of the figure.

6 | PROOF OF CONCEPT: ASCLEPIOS

Consider the use of cloud computing in domains and use cases with complex requirements, such as those in the field of healthcare. Over the course of a project that develops cloud solutions in such a domain, an array of public and private clouds may be appropriate—or even required—at different stages of the project. A given cloud might confer a particular security benefit, or a certain healthcare provider may be bound to using a specific private cloud infrastructure. In cases of this sort, accessible application portability is of the utmost importance. One such project is ASCLEPIOS.

The vision of the ASCLEPIOS project is to maximize and fortify the trust of users on cloud-based healthcare services by exploiting modern cryptographic approaches to build a cloud-based eHealth framework that protects users' privacy and prevents both internal and external attacks. ASCLEPIOS demonstrates the applicability of the developed framework on healthcare applications provided by three European hospitals, with the intention of deploying these applications alongside the ASCLEPIOS framework with several different cloud service providers. The ASCLEPIOS cloud test-bed features a mix of private (at the University of Westminster and at the Norwegian Centre for E-Health Research) and public (Amazon Web Services and Microsoft Azure) clouds and facilitating portability between them plays an important role in development of both the ASCLEPIOS framework and the healthcare applications it is designed to support.

6.1 | Sleep healthcare

One of the use cases within the ASCLEPIOS project involves bringing data sharing and analysis on inpatient and outpatient sleep medicine to the cloud. Using the cloud for storing and processing the different sleep measurements can be highly beneficial. Giving different actors in the healthcare domain access to parts of patient data could reduce measurement failures by identifying them more quickly and would help to achieve real-time monitoring even when such measures are collected outside of a state-of-the-art sleep lab.

Sleep is an important factor in human health and is for example crucial for a powerful immune system.[28] Sleep depends on and affects the very complex interactions of different physiological processes. Sleep disturbance—for example due to cultural habits of the "24h-society"—might cause or worsen health issues such as cardiovascular diseases or mental disorders. On the other hand, many disorders can affect recreational sleep. This complex two-way interaction makes diagnosis in sleep medicine and sleep research a complex task itself. The de-facto standard in sleep diagnosis is overnight recording of several biosignals, including among others electroencephalography, electrocardiography, and breathing effort. This is called polysomnography (PSG) and must be performed in a sleep laboratory.

The most prevalent sleep disorder is sleep apnea, a repeated cessation of breathing during sleep. Sleep apnea is associated with an overall higher risk of morbity and mortality, as it causes stress to the cardiovascular system and leads to fragmented sleep. If sleep apnea is suspected, in many countries the typical diagnosis is performed based on home sleep testing (HST), where airflow, breathing efforts, oxygen saturation and heart rate are measured at the patient's home. This method is much less expensive than a PSG and waiting times are much shorter.

While not yet established, wearable sensors such as smartwatches are considered to help diagnosing sleep disorders, as they would allow largely undisturbed sleep in the home environment. Although it is common to store personal health data from such lifestyle products in the manufacturer's cloud, this would not be legal for official medical data taken in the context of medical treatment. Here, higher levels of data protection are required. Sharing and remote visualization of sleep data is already available, based on the popular open source biomedical data repository xnat and WebRTC.[29,30] However, only transport layer encryption is currently enabled, making it inappropriate for cloud deployment in the current state.

One measure to enable the use of the cloud in healthcare is the encrypted storage of medical data. However, studies have shown[31] that metadata used to query such encrypted data, such as birth dates, zip code, and race, are enough for a malicious actor to identify individuals and reveal suspected health issues. Therefore, encryption of this metadata is crucial for health data protection. Searchable Encryption (SE) is a promising new technology to allow queries on encrypted data in a way that the cloud provider cannot reveal the metadata search term nor the query result. In the context of a Science Gateway in Sleep Healthcare which is currently under development, SE technologies would allow medical professionals to manage biosignal recordings from the different inpatient and outpatient settings to enhance both the process and the precision of sleep diagnosis and therapy control, while preserving patient data privacy.

6.2 | Symmetric searchable encryption for sleep healthcare

In the scope of ASCLEPIOS, a novel Symmetric Searchable Encryption (SSE) scheme has been proposed.[32] The SSE encryption technique enables a search on outsourced encrypted data while preserving the privacy of both the data and any search queries. Figure 7 presents the high level architecture of the SSE scheme in integration with the Sleep Healthcare application using MiCADO as its deployment and orchestration system.

The SSE system model consists of two core components—a Trusted Authority (TA) and an SSE Server. The SSE Server represents the cloud service provider that is responsible for data storage (in its database, *sse-db*), whereas the TA stores the metadata (in its database, *ta-db*) required for the facilitation of data searches. Using the SSE scheme, a client application (the Sleep Healthcare demonstrator in this case) encrypts data and creates a dictionary which maps extracted keywords to data files at the end-user side before sending them to the SSE Server for storage. The data sent for storage are fully encrypted and therefore, the SSE Server has no capability to understand and/or decrypt the stored data. The Sleep Healthcare application also sends metadata about the encrypted data to the TA. This metadata will be used to assist the Sleep Healthcare application to search over the encrypted data. In the search process, the application receives keywords from end-users. Using the keywords and with the help of the TA, it creates search tokens and sends them to the SSE Server, which relies on the stored dictionary and the received token to retrieve the specific encrypted data.

For the sake of security, when in production, the TA and SSE must be deployed in a Trusted Execution Environment (TEE),[33] for example with Intel Software Guard Extensions (SGX)[34] capabilities. However, such a restriction is not mandatory for client applications (i.e., Sleep Healthcare), nor is it a restriction in development environments. Being that Microsoft Azure is currently the only cloud in the ASCLEPIOS testbed with support for SGX, much of the development work for the SSE Scheme has taken place there. The development and end-to-end testing of the Sleep Healthcare demonstrator, on the other hand, has mainly utilized AWS as the cloud of choice. The current versions of SSE Server and TA have not yet enabled SGX related functions so the preliminary integration of the SSE Scheme into the Sleep Healthcare application took place on AWS. As a second step, to demonstrate the feasibility of the framework across multiple clouds, further testing of the Sleep Healthcare demonstrator using SSE is taking place on the University of Westminster OpenStack cloud. Finally, to test the full solution with the SSE Scheme in an SGX environment, and to prepare for an eventual move to a production environment, deployment of the Sleep Healthcare demonstrator to Microsoft Azure is performed.

6.3 | Deployment using MiCADO

To realize the multiple deployments of the Sleep Healthcare demonstrator featuring SSE, the MiCADO framework was employed as the deployment and execution engine behind it. It can be seen from Figure 7 that all components are deployed on separate MiCADO worker nodes (shown as dotted line rectangles). The description of the required resources and application topology are provided in a MiCADO ADT (versioned for each of the three different cloud service providers) that is available at `github.com/micado-scale/tosca/tree/asclepios/ADT/sleep`. The definitions of required virtual machines and Docker containers are handled using the custom TOSCA types listed in Table 4 and extend the sample deployment scenarios presented in Figures 4 and 5.

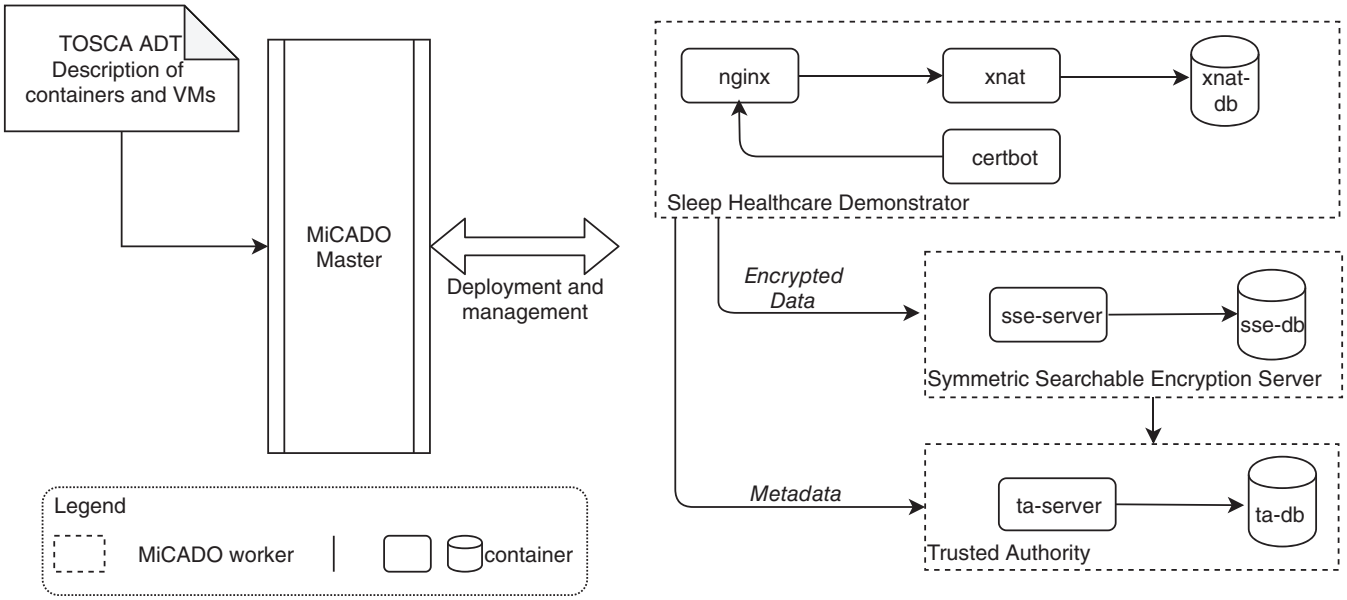

FIGURE 7 Integration of Symmetric Searchable Encryption scheme, Sleep Healthcare demonstrator, and MiCADO

TABLE 4 Summary of the comparative analysis of the technologies and solutions presented in related works based on a 12-point comparison reflecting application portability

	OpenTOSCA	SeaClouds	Cloudify	Puccini	Opera	Alien4Cloud	TosKer	Trans-Cloud	INDIGO-DataCloud	MiCADO
Automated Deployment and Configuration	X	X	X		X	X	X	X	X	X
Run-time Orchestration	X	X	X		X	X		X	X	X
Auto-Scaling via User Defined Policies		X	X			X				X
Modular Design	X		X	X		X	X			X
Open-Source	X	X	X	X	X	X	X	X	X	X
Virtual Machine Support			X	X	X	X		X	X	X
Container Support	X	X	X	X	X	X	X	X	X	X
Cloud Agnostic	X	X	X	X	X	X	X	X	X	X
Extendable Cloud Support			X			X				X
Cloud Model	I	P	I	I	I	I	I	I/P	I	I
Language(s) of Abstraction Layer	T	T/C	T	T/Clo	T/An	T	T	T/C	T/An	T
Resource Independent from Orchestrator		X	X		X	X	X	X	X	X

Legend | *Cloud Model:* **(I)**aaS / **(P)**aaS | *Languages:* **(T)**OSCA / **(C)**AMP / **(Clo)**ut / **(An)**sible.

Once the ADT is finalized and submitted to MiCADO, it deploys the Sleep Healthcare demonstrator as well as the necessary SSE components and starts managing the application based on operator-defined policies (e.g., the operator can define certain thresholds for the utilization of CPUs; if CPU usage goes beyond the threshold then MiCADO scales up the utilized resources). As a first step of testing, an ADT that deploys the Sleep Healthcare demonstrator to AWS was applied. As a second test, the ADT was re-used, by only changing the TOSCA nodes describing compute instances, to deploy the application on the University of Westminster private OpenStack cloud. The sections of the ADT describing the application containers, cloud and container orchestrators, and monitoring and scaling policies remained unchanged. Such migration is explained in detail in Section 5.3 and the example of the necessary modifications, using a simple example, are presented in Figure 6. For the final step, to mimic a production environment where SGX was enabled, MiCADO deployed the Sleep Healthcare demonstrator to Microsoft Azure—again re-using the previous ADT and only making changes to the section describing the cloud compute resources. MiCADO facilitated portability of the Sleep Healthcare demonstrator featuring the ASCLEPIOS-developed SSE Scheme to three different cloud service providers as it moved from the early stages of development to a pre-production environment.

7 | RELATED WORKS

With the increasing use of cloud, application portability across different cloud providers has been given much importance. Hence, over the last a few years, the topic of portability has gained a lot of attention from the research community as well as from industry. The focus is mostly on the use of cloud descriptive languages, among which TOSCA is one of the most widely employed. Most TOSCA related existing works claim to be a viable solution toward the well-known vendor lock-in problem while offering flexibility and portability. However, not all such approaches consider modularity, where applications and cloud resources can be described in a way that is easily portable across a modular framework. This section provides an overview of the most relevant existing works that focused on the use of TOSCA and similar approaches to achieve portability.

OpenTOSCA[35] is one of the earliest run-times for TOSCA-based cloud applications. It orchestrates TOSCA XML-based templates (an earlier version of the TOSCA specification), which can also be created with the integrated graph-based modeling tool called Winery.[36] OpenTOSCA supports the imperative processing of TOSCA applications, indicating that the deployment and management of logic plans are implemented as workflows. Both OpenTOSCA and Winery adhere to the TOSCA v1.0 normative XML specification. Analogous to OpenTOSCA, Seaclouds[37] also fully supports TOSCA and was one of the initial solutions for deploying and managing multi-component applications on heterogeneous clouds.

Cloudify[38] facilitates the modeling of applications and services to automate their entire life cycle including deployment, monitoring, failure detection, and maintenance tasks. Cloudify uses its own Domain Specific Language (DSL) that relies on the base specification of TOSCA. The Cloudify DSL uses strict types. For example, there are different types of containers (non-orchestrated and orchestrated) defined for each orchestrator (Docker Swarm and Kubernetes). This means that each of these different types requires key/value pairs specific to a different orchestrator. Such level of complexity makes it very unlikely or impossible to reuse the container definition for a different orchestrator. Though it supports multiple clouds, it is complex to achieve via the command-line interface (CLI). The user has to download a plugin for the required cloud and individually configure the node and network details associated with that cloud.

Puccini,[39] an open source front-end, translates TOSCA (v1.0-v1.3) to a middle-language called Clout and then Clout to an orchestrator specific language (e.g., Kubernetes manifests), before being piped into the specific orchestration engine (e.g., Kubernetes CLI). The Clout involves strict typing where applications are first fully defined, along with all the properties and requirements, using TOSCA types. These types are then imported and referenced in the TOSCA template to be used at deployment time. Such an approach adds additional complexity by introducing another layer during template creation for application deployment, in contrast to extending a generic type with specific properties and requirements for an application. Similar to Puccini, Opera[40] is also compliant with OASIS TOSCA v1.3. Opera, developed within the scope of the RADON[41] and Sodalite[42] projects, provides a DevOps framework to create and manage microservices-based applications. However, unlike Puccini, it has focused on the optimal exploitation of Function as a Service (FaaS) technology to avoid FaaS provider lock-in.

Alien4Cloud[43] is an application management platform that leverages TOSCA portability to encourage enterprise organizations to deploy their applications over a cloud. Alien4Cloud provides a custom DSL with strict but not full adherence to TOSCA Simple Profile in YAML v1.0. Furthermore, it provides different plugins and GUI support for orchestrating and designing the required TOSCA templates using various tools, including Cloudify, Kubernetes, and Puccini. The Alien4Cloud DSL follows a more complex layered approach. A typical example is the following three-layered scenario, where the Docker container is defined as a generic type, irrespective of the orchestrator, followed by the definition of container runtime, and finally, the container deployment unit that instructs the framework as to which container orchestrator should be used. Such a layered approach facilitates the high level of flexibility and ease of portability across different orchestration tools. However, this also complicates the initial authoring of a TOSCA template.

In contrast to the above-mentioned container focused approaches, TosKer[44] separates the definition of application from the container. It defines one type for Docker and other type for the software that may (or may not) run inside the container. With TosKer, the user can easily specify the generic dependencies and connections of the software component with other software components and containers. As the software components and Docker containers can have their own requirements and capabilities, the way in which they are interconnected influences the order in which they

have to be orchestrated. This approach allows more flexibility to define and manage systems that combine both the containers and the traditionally run applications. However, at the same time, this adds another layer of complexity.

Carrasco et al.,[45] in comparison to the above-mentioned approaches, proposed an orchestration solution entitled Trans-cloud, to address the portability at two different levels, that is, IaaS and PaaS levels. It unifies components' deployment using Iaas and PaaS of multiple providers and builds upon TOSCA and, additionally, Cloud Application Management for Platforms (CAMP)[46]—another standard with a specific focus on application deployment and management. Trans-cloud extends Apache Brooklyn,[47] a tool to describe application components and their deployment using a CAMP-based interface. Trans-cloud accepts a TOSCA YAML-based description of an application topology and transforms it to the corresponding Brooklyn compliant template. The unification of IaaS and PaaS interfaces into one makes Trans-cloud very distinct from all other proposals. However, in Trans-cloud certain aspects related to applications, for example, configurations related to security features, must be handled separately at the Brooklyn level. A MiCADO ADT, in contrast, has the support to define network and security related policies directly with the application topology definition.

Challita et al.[48] combined TOSCA with the Open Cloud Computing Interface (OCCI).[49] The OCCI is a standardization approach toward a common API for the IaaS providers. Using TOSCA and OCCI, the authors proposed a model driven cloud orchestration framework that maps a TOSCA-based description, using Ecore meta-modeling,[50] into deployable OCCI meta-model configurations. OCCI, similarly to TOSCA, is a widely known standard. However, mapping new TOSCA custom types also requires deeper knowledge of the corresponding OCCI meta-model configurations and thereby increases the complexity in adaptation of such an approach.

The research work in Caballer et al.,[51] within the scope of INDIGO-DataCloud[52] project, proposed a TOSCA based system for the deployment and management of scientific applications over heterogeneous cloud infrastructure. The proposed orchestration, however, only supports OpenStack and OpenNebula based cloud systems. The focus of INDIGO-DataCloud are specifically for scientific applications. Furthermore, the applications are required to have an Ansible role, an entry in Ansible Galaxy and a new node type for each application.

The above-mentioned related works as well as MiCADO are further evaluated against the following key characteristics. These characteristics are important when considering a solution for cloud orchestration and application portability. The summarized results from the evaluation can be seen in Table 4.

1. Automated deployment and configuration of cloud applications;
2. Run-time orchestration of cloud applications;
3. Automated scaling based on dynamic and user-defined policies;
4. Modular design;
5. Open-source;
6. Support for virtual machines;
7. Support for containers;
8. Cloud agnostic;
9. Extendable cloud provider and cloud middleware support;
10. Cloud model supported;
11. Language(s) used for abstraction layer;
12. Resource definition layer independent from underlying orchestration component.

It is evident from the comparative analysis of Table 4 that the majority of the listed approaches (except Cloudify, Alien4Cloud, and MiCADO) lack either one or more aspects. Though Cloudify fulfils all the important criteria considered for comparison, the use of strict types (e.g., different container types for different orchestrators) makes it unlikely or impossible to reuse the container definition for a different orchestrator. Similarly, in the case of Alien4Cloud, the use of complex layered approach helps in achieving higher flexibility, however, it also complicates the initial authoring of TOSCA templates. To summarize the comparative analysis from the table and from the analytical discussion of individual approaches, it is evident that the wide range of currently available approaches to TOSCA are either too complexly layered, or specifically tailored to handle certain domains or scenarios or lack of certain important aspects, for example, generality, inability to orchestrate at runtime, no extensibility for adoption of further cloud providers, lack of modularity, and no support for automated scaling based on user-defined policies. In contrast, TOSCA adoption in the form of the MiCADO ADT offers a more flexible and modular approach to describe all aspects of cloud applications ranging from basic application and cloud resource definitions to the description of a variety of policies such as security and scalability.

8 | CONCLUSION AND FURTHER WORK

MiCADO and its ADT work together to facilitate and encourage portability in the cloud, whether it be between container or cloud orchestration environments or cloud service providers. Its implementation behind a Science Gateway such as Sleep Healthcare can simplify the deployment and

execution of that Gateway across different cloud service providers, which could otherwise prove difficult for research groups or application developers. TOSCA is the basis for the interface in MiCADO, and by extending it to suit the specific environment of MiCADO, the ADT acts as the ideal bridge between the user and the modular set of components which drive application orchestration. This novel approach to TOSCA has already seen MiCADO through a transition of container orchestration environments and more recently, it has made simple the addition of a second cloud orchestration tool.

As the development of MiCADO continues, new orchestration tools will emerge, and by exploiting the modularity of MiCADO and utilizing the flexibility of the ADT, integrating them into future versions will be an approachable task. Thus far, MiCADO has only included support for provisioning compute instances. However, Terraform adds potential support for additional cloud resources such as object storage or serverless functions. A future release of MiCADO will see these resources given their own TOSCA types, added to ADTs and orchestrated by MiCADO as part of an even more powerful application infrastructure.

In addition to supporting new tools and resources in MiCADO, the ADT interface will also be extended to support new or existing constructs in TOSCA, especially as it moves into Simple Profile v2.0. Simple policies exist within the ADT, but the approach to them can be improved and made to support portability across different policy engines. TOSCA Workflows can enhance the flexibility of MiCADO ADTs by giving template authors finer-grained control over the orchestration flow, and to enable such control over the modular set of components in MiCADO will be highly beneficial.

By supporting modularity in both its implementation and interface, the MiCADO framework can be extended to support virtually any combination of different cloud providers, environments, and resources. Hiding some of the complexity that is inherent in such a system is crucial, and is accomplished with the TOSCA-based approach to the ADT interface. Making cloud portability more approachable is a step toward making vendor lock-in less common, and MiCADO, the ADT and TOSCA all work together toward this end.

ACKNOWLEDGEMENT

This research was supported by European Commission H2020 Programme, ASCLEPIOS Advanced Secure Cloud Encrypted Platform for Internationally Orchestrated Solutions in Healthcare Project No. 826093.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available. MiCADO is fully open-source and its code is hosted on GitHub under the organization page `github.com/micado-scale`, with TOSCA-specific material at `github.com/micado-scale/tosca`. The Sleep Healthcare application can be found at `github.com/somnonetz/snet-docker-compose` and the SSE scheme components can be found at `gitlab.com/asclepios-project`. The proof of concept described in the article can be reproduced by deploying an instance of MiCADO using the documentation at `readthedocs.org/projects/micado-scale/` and submitting the Sleep Healthcare ADT found on GitHub at `github.com/micado-scale/tosca/tree/asclepios/ADT/sleep`.

ORCID


James DesLauriers https://orcid.org/0000-0003-0336-3213

REFERENCES

1. AWS CloudFormation Documentation. http://docs.aws.amazon.com/cloudformation/index.html. Accessed April 24, 2020.
2. HEAT Documentation. http://docs.openstack.org/heat/latest. Accessed April 24, 2020.
3. Terraform http://www.terraform.io. Accessed April 12, 2020.
4. Chef. http://www.chef.io. Accessed April 24, 2020.
5. Ansible. http://www.ansible.com. Accessed April 24, 2020.
6. Docker. http://www.docker.com. Accessed April 24, 2020.
7. Kubernetes. http://www.kubernetes.io. Accessed April 24, 2020.
8. OASIS. http://www.oasis-open.org. Accessed April 24, 2020.
9. OASIS topology and orchestration specification for cloud applications. http://www.oasis-open.org/committees/tosca. Accessed April 24, 2020.
10. OASIS topology and orchestration specification for cloud applications (TOSCA) TC. TOSCA Simple Profile in YAML Version 1.3. OASIS Open; 2020. https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/TOSCA-Simple-Profile-YAML-v1.3.pdf.
11. Deslauriers J, Kiss T, Pierantoni G, Gesmier G, Terstyanszky G. Enabling modular design of an application-level auto-scaling and orchestration framework using TOSCA-based application description templates. Paper presented at: Proceedings of the 11th International Workshop on Science Gateways, IWSG 2019; 2019; Ljubljana, Slovenia.
12. Kiss T, Kacsuk P, Kovacs J, et al. MiCADO—Microservice-based Cloud Application-level Dynamic Orchestrator. *Future Generation Computer Systems*. 2019;94 937–946. https://dx.doi.org/10.1016/j.future.2017.09.050.
13. Pierantoni G, Kiss T, Gesmier G, Deslauriers J, Terstyanszky G, Rapun M. Flexible deployment of social media analysis tools. Paper presented at: Proceedings of the 10th International Workshop on Science Gateways, IWSG 2018; 2018; Edinburgh, UK.
14. ASCLEPIOS Projet. http://www.asclepios-project.eu. Accessed April 24, 2020.
15. MiCADO-scale. http://www.micado-scale.eu. Accessed April 12, 2020.
16. COLA Project. http://www.cola-project.eu. Accessed April 24, 2020.

17. Kiss T, Bolotov A, Pierantoni G, et al. Science gateways with embedded ontology-based E-learning support. Paper presented at: Proceedings of the 12th International Workshop on Science Gateways, IWSG, June 11, 2020, Online Event (under review).
18. Kovacs J, Kiss T, Taylor SJE, et al. Industry simulation gateway on a scalable cloud. Paper presented at: Proceedings of the 12th International Workshop on Science Gateways, IWSG, June 11, 2020, Online Event (under review).
19. Kiss T, DesLauriers J, Gesmier G, et al. A cloud-agnostic queuing system to support the implementation of deadline-based application execution policies. *Future Generat Comput Syst*. 2019;101:99-111. https://doi.org/10.1016/j.future.2019.05.062.
20. Kovacs J, Kacsuk P. Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures. *J Grid Comput*. 2018;16(1):19-37.
21. Prometheus. http://www.prometheus.io. Accessed April 12, 2020.
22. Kovacs J. Supporting programmable autoscaling rules for containers and virtual machines on clouds. *J Grid Comput*. 2019;17:813-829. https://doi.org/10.1007/s10723-019-09488-w.
23. cloud-init Documentation. http://cloudinit.readthedocs.io. Accessed May 11, 2020.
24. Compose Specification. https://compose-spec.io. Accessed April 20, 2020.
25. Kubernetes Reference. https://kubernetes.io/docs/reference. Accessed April 20, 2020.
26. CloudSigma Cloud servers & hosting. http://www.cloudsigma.com. Accessed May 4, 2020.
27. Taylor SJ, Kiss T, Anagnostou A, et al. The CloudSME simulation platform and its applications: a generic multi-cloud platform for developing and executing commercial cloud-based simulations. *Future Generat Comput Syst*. 2018;88):524-539. https://doi.org/10.1016/j.future.2018.06.006.
28. Besedovsky L, Lange T, Born J. Sleep and immune function. *Pflugers Archiv*. 2012;463(1):121-137. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3256323/.
29. Beier M, Jansen C, Mayer G, et al. Multicenter data sharing for collaboration in sleep medicine. *Future Generat Comput Syst*. 2017;67:466-480. https://doi.org/10.1016/j.future.2016.03.025.
30. Beier M, Penzel T, Krefting D. A performant web-based visualization, assessment, and collaboration tool for multidimensional biosignals. *Front Neuroinform*. 2019;13. https://doi.org/10.3389/fninf.2019.00065.1–11.
31. Sweeney L. k-anonymity: a model for protecting privacy. *Int J Uncert Fuzz Knowl-Based Syst*. 2002;10(05):557-570.
32. Bakas A, Michalas A. Power range: forward private multi-client symmetric searchable encryption with range queries. Paper presented at: Proceedings of the 25th IEEE International Conference on Communications (ISCC 2020); 2020.
33. Sabt M, Achemlal M, Bouabdallah A. Trusted execution environment: what it is, and what it is not. Paper presented at: Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA. Helsinki, Finland; vol. 1, 2015:57-64; IEEE.
34. Costan V, Devadas S. Intel SGX explained. *IACR Cryptol ePrint Arch*. 2016;2016(086):1-118.
35. Binz T, Breitenbücher U, Haupt F, et al. OpenTOSCA – a runtime for TOSCA-based cloud applications. *Service-Oriented Computing*. Berlin, Heidelberg/Germany: Springer; 2013:692-695.
36. Kopp O, Binz T, Breitenbücher U, Leymann F. Winery – a modeling tool for TOSCA-based cloud applications. *Service-Oriented Computing*. Berlin, Heidelberg/Germany: Springer; 2013:700-704.
37. Brogi A, Carrasco J, Cubo J, et al. EU project SeaClouds - adaptive management of service-based applications across multiple clouds. Paper presented at: Proceedings of the 4th International Conference on Cloud Computing and Services Science - Volume 1: MultiCloud, (CLOSER 2014). INSTICC. Barcelona, Spain; 2014:758-763; SciTePress.
38. Cloudify "Cutting Edge Orchestration." https://cloudify.co. Accessed April 14, 2020.
39. "Puccini - deliberately stateless cloud topology management and deployment tools based on TOSCA". https://github.com/tliron/puccini. Accessed April 14, 2020.
40. Opera Orchestrator. https://github.com/xlab-si/xopera-opera. Accessed April 14, 2020.
41. RADON H2020 PROJECT. http://radon-h2020.eu/. Accessed April 14, 2020.
42. Software defined application infrastructures management and engineering | Sodalite. https://www.sodalite.eu/. Accessed September 14, 2020.
43. ALIEN 4 Cloud. http://alien4cloud.github.io. Accessed April 14, 2020.
44. Brogi A, Rinaldi L, Soldani J. TosKer: a synergy between TOSCA and Docker for orchestrating multicomponent applications. *Softw Pract Exp*. 2018;48(11):2061-2079. https://doi.org/10.1002/spe.2625.
45. Carrasco J, Durán F, Pimentel E. Trans-cloud: CAMP/TOSCA-based bidimensional cross-cloud. *Comput Stand Inter*. 2018;58:167-179.
46. OASIS cloud application management for platforms (CAMP) TC. cloud application management for platforms Version 1.1. OASIS Open; 2014. http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html.
47. Apache brooklyn. https://brooklyn.apache.org/. Accessed April 25, 2020.
48. Challita S, Korte F, Erbel J, Zalila F, Grabowski J, Merle P. Model-based cloud resource provisioning with TOSCA and OCCI; 2020. arXiv preprint arXiv:200107900.
49. Open cloud computing interface. https://occi-wg.org. Accessed April 14, 2020.
50. Zalila F, Challita S, Merle P. A model-driven tool chain for OCCI. Paper presented at: Proceedings of the OTM Confederated International Conferencesε On the Move to Meaningful Internet Systemsε; 2017:389-409; Springer, New York, NY.
51. Caballer M, Zala S, García ÁL, Moltó G, Fernández PO, Velten M. Orchestrating complex application architectures in heterogeneous clouds. *J Grid Comput*. 2018;16(1):3-18.
52. European Horizon2020 Indigo-DataCloud project. https://www.indigo-datacloud.eu/. Accessed April 25, 2020.

How to cite this article: DesLauriers J, Kiss T, Ariyattu RC, et al. Cloud apps to-go: Cloud portability with TOSCA and MiCADO. *Concurrency Computat Pract Exper*. 2021;33:e6093. https://doi.org/10.1002/cpe.6093