Journal of Grid Computing (2021) 19: 29
https://doi.org/10.1007/s10723-021-09572-0

SODALITE@RT: Orchestrating Applications on Cloud-Edge Infrastructures

Indika Kumara · Paul Mundt · Kamil Tokmakov · Dragan Radolović · Alexander Maslennikov · Román Sosa González · Jorge Fernández Fabeiro · Giovanni Quattrocchi · Kalman Meth · Elisabetta Di Nitto · Damian A. Tamburri · Willem-Jan Van Den Heuvel · Georgios Meditskos

Received: 9 November 2020 / Accepted: 21 June 2021 / Published online: 10 July 2021

Abstract IoT-based applications need to be dynamically orchestrated on cloud-edge infrastructures for reasons such as performance, regulations, or cost. In this context, a crucial problem is facilitating the work of DevOps teams in deploying, monitoring, and managing such applications by providing necessary tools and platforms. The *SODALITE@RT* open-source framework aims at addressing this scenario. In this paper, we present the main features of the *SODALITE@RT*: modeling of cloud-edge resources and applications using open standards and infrastructural code, and automated deployment, monitoring, and management of the applications in the target infrastructures based on such models. The capabilities of the *SODALITE@RT* are demonstrated through a relevant case study.

Keywords Orchestration · Cloud · Edge · Heterogeneous infrastructures · TOSCA · Containers

European Commission grant no. 825480 (H2020), SODALITE.

I. Kumara (✉) · D. A. Tamburri · W.-J. Van Den Heuvel
Jheronimus Academy of Data Science, Eindhoven University of Technology, Sint Janssingel 92, 5211 DA 's-Hertogenbosch, Netherlands
e-mail: i.p.k.weerasingha.dewage@tue.nl

P. Mundt
Adaptant Solutions AG, Munich, Germany

K. Tokmakov
University of Stuttgart, Stuttgart, Germany

D. Radolović · A. Maslennikov
XLAB Research, Ljubljana, Slovenia

R. González · J. F. Fabeiro
ATOS, Madrid, Spain

G. Quattrocchi · E. Di Nitto
Politecnico di Milano, Milano, Italy

K. Meth
Haifa Research Lab, Haifa, Israel

G. Meditskos
Information Technologies Institute, Centre for Research and Technology Hellas, Hellas, Greece

1 Introduction

Over the last few years, cloud computing technologies have become mature, and organizations are increasingly using the cloud as their IT infrastructure [1]. On the other hand, the era of the Internet of Things (IoT) is rapidly coming of age, with a large number of IoT devices already deployed in network edges [2]. Organizations typically have complex applications consisting of multiple components that need to be deployed on multiple infrastructure types to utilize characteristics of a particular type to achieve the best performance, for example, usage of cloud resources for compute-intensive tasks and edge resources for

latency-sensitive services. However, manually deploying complex applications with heterogeneous deployment models is a highly complex, time-consuming, error-prone, and costly task [3].

In the last decade, the automated deployment and management of applications have been considered vitally crucial by both academia and industry [3–9]. Most current works focus on deploying applications on clouds [10–13], including multi-clouds [5, 14–16] and hybrid clouds [17, 18]. Recently, several studies have employed the container technology for deploying applications on edge infrastructures [19, 20]. However, the containerization-based solutions fail to deal with complex applications that span across multiple heterogeneous container clusters or hybrid VM and container clusters [21, 22].

In this paper, we present the *SODALITE* (SOftware Defined AppLication Infrastructures managemenT and Engineering) platform (namely *SODALITE@RT/runtime*), which aims to support the deployment, execution, monitoring, and management of applications on heterogeneous cloud-edge infrastructures. To deal with the heterogeneity of resources and applications, we use the open standard TOSCA (Topology and Orchestration Specification for Cloud Applications) [23] to describe heterogeneous cloud and edge resources and applications in a portable and standardized manner. The TOSCA-based models are implemented by using the industrial IaC (Infrastructure-as-Code) technologies [24]. IaC enables the automated management and provisioning of infrastructures using machine-readable definition files rather than manual setup and configuration. The *SODALITE@RT* platform includes a meta-orchestrator that employs IaC to deploy and manage the applications by utilizing and coordinating the low-level resource orchestrators offered by different execution platforms (e.g., OpenStack, AWS, and Kubernetes at Edge). The *SODALITE@RT* also supports the monitoring and policy-based runtime adaptation of the application deployments.

The rest of the paper is organized as follows. Section 2 motivates the needs for orchestrating applications on cloud-edge environments and highlights the key challenges. Section 3 provides an overview of TOSCA and IaC, and summarizes the related studies. Section 4 presents the *SODALITE@RT* in detail, including high-level architecture, modeling, deployment, monitoring, and deployment adaptation. Sections 5 and 6 present the implementations of the *SODALITE@RT* and the motivating case study. Section 7 discusses the key usage scenarios for the *SODALITE@RT*, and Section 8 concludes the paper.

2 Motivation: Vehicle IoT Case Study

In this section, using an industrial case study from our SODALITE H2020 project,[1] we illustrate the challenges in orchestrating dynamic applications over cloud-edge infrastructures.

The SODALITE Vehicle IoT use case involves the provisioning and delivery of data-driven services from the cloud to a connected vehicle (or across a fleet of vehicles), leveraging a combination of data both from the vehicle itself (e.g., GPS-based telemetry data, gyroscope and accelerometer readings, biometric data from driver monitoring) and from external sources that can enrich the vehicle data and provide additional context to the service (e.g., weather and road condition data based on the location and heading of the vehicle). Figure 1 shows the simplified high-level architecture, highlighting the services and other components deployed at the cloud and the edge. The services include deep/machine learning (DL/ML) based applications such as drowsiness detection, license plate detection, and intrusion and theft detection. As computational capabilities at the edge are often limited, the corresponding DL/ML model training services are hosted at the cloud.

The vehicle IoT application highlights the following two key challenges pertaining to orchestrating cloud-edge applications:

1. **Supporting Portability of Cloud-Edge Application Deployments.** The application needs to be deployed over multiple cloud and edge infrastructures with little or no modification. Moreover, some components of the application may be deployed on either cloud or edge nodes. Within a given cloud or edge infrastructure, there may exist heterogeneous resources, for example, different VM types, edge gateways, and hardware accelerators. Thus, portashould be supported at each phase of the application deployment workflow, including packaging application components, modeling

[1] https://www.sodalite.eu/

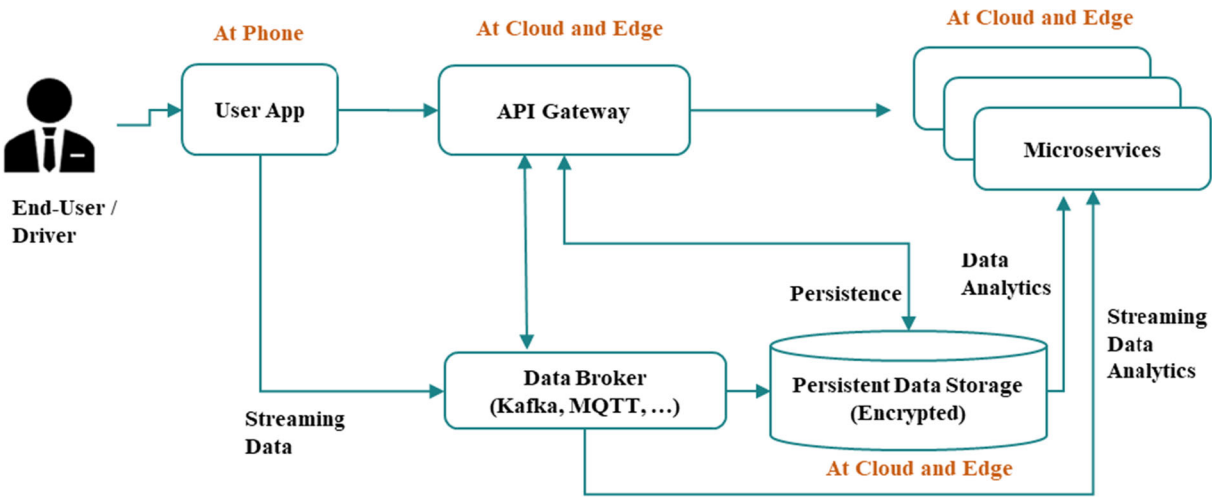

Fig. 1 A simplified high-level architecture of the vehicle IoT application

the application's deployment topology, and provisioning and configuring resources.

2. **Supporting Runtime Management of Cloud-Edge Application Deployments.** Cloud-Edge infrastructures and users exhibit considerable dynamism, which can make the deployed application sub-optimal, defective, and vulnerable as the usage context changes. For example, the vehicle is not a stationary object and may, at any time, crosses over into another country, - subjecting the data processing activities carried out by the services to the regulatory compliance requirements of not only the country where it started its journey, but also every country it enters along the way. As the workload changes, the utilization of cloud-edge resources also changes. Overutilization of resources can lead to violations of the application's performance objectives, while underutilization can incur an undue cost. Different edge accelerators have different performance modes and thermal operating ranges. Stepping outside of these ranges can lead to (machine learning) inference failures or other types of hard-to-detect undefined behaviors. In order to cope with the dynamism of the cloud-edge applications successfully, their deployments need to be monitored and managed at runtime. For example, the thermal states of the edge nodes should be monitored, and the redeployment using more thermally-conservative configurations should be triggered when a predefined threshold is crossed. In response to the location-changed events originated from the vehicle or user app, the application should be partially redeployed to prevent the violation of regulatory compliance requirements.

3 Background and Related Work

In this section, we first introduce the technologies that the *SODALITE@RT* uses to model and implement deployment models of complex heterogeneous applications. A deployment model is a specification of the components belonging to the application and their connectors, as well as their dependencies on a specific technological stack [3]. Next, we present an overview of the existing studies on orchestrating applications on cloud-edge infrastructures.

3.1 TOSCA

TOSCA [23, 25, 26] is an OASIS standard for describing deployment and management of distributed applications declaratively in a portable way. The key TOSCA concepts for describing a deployment model are : *Topology Template*, *Node Template*, *Node Type*, *Relationship Template*, and *Relationship Type*. *Topology Template* specifies the structure of the application in terms of *Node Templates* and *Relationship Templates*. Node Templates model application components (e.g., virtual machines, databases, and web services), whose semantics (e.g., properties, attributes,

requirements, capabilities and interfaces) are defined by *Node Types*. *Relationship templates* capture relations between the nodes, for example, a node hosting another node or network connection between nodes. *Relationship types* specify the semantics (e.g., properties and interfaces) of these relationships. The properties and attributes represent the desired and actual states of nodes or relationships, e.g., IP address or VM image type. Interfaces define the management operations that can be invoked on nodes or relationships, e.g., creating or deleting a node.

The TOSCA standard originally was developed for defining deployment models for automating the orchestration of cloud applications in a vendor-agnostic fashion. The TOSCA language is highly extensible as new types (e.g., node types, capability types, and policy types) can be defined without extending the language it self. The deployment models specified in TOSCA are generally enacted by the middleware systems called *orchestrators*. The management operations of a deployment model can be realized using different languages including classical shell scripts. Overall, the TOSCA standard enables achieving the portability and reusability of the deployment model definitions.

3.2 IaC and Ansible

Infrastructure-as-Code (IaC) [24] is a model for provisioning and managing a computing environment using the explicit definition of the desired state of the environment in source code via a Domain Specific Language (DSL), and applying software engineering principles, methodologies, and tools. The interest in IaC is growing steadily in both academia and industry [7, 27]. Instead of low-level shell scripting languages, the IaC process uses high-level DSLs that can be used to design, build, and test the computing environment as if it is a software application/project. The conventional management tools such as interactive shells and UI consoles are replaced by the tools that can generate an entire environment based on a descriptive model of the environment. A key property of the management tasks performed through IaC is idempotence [28]. The idempotence of a task makes the multiple executions of it yielding the same result. The repeatable tasks make the overall automation process robust and iterative, i.e., the environment can be converted to the desired state in multiple iterations. IaC languages and tools typically support the provision and management of a wide range of infrastructures including public clouds, private clouds, HPC clusters, and containers. Thus, the IaC approach also enables achieving greater application portability as the applications can be moved across different infrastructures with little or no modification to IaC programs.

SODALITE@RT prototype uses the Ansible IaC language[2] to operationalize the TOSCA based deployment models. Ansible is one of the most popular languages amongst practitioners, according to our previous survey with practitioners [7]. In Ansible, a *playbook* defines an IT infrastructure automation workflow as a set of ordered *tasks* over one or more *inventories* consisting of managed infrastructure nodes. A *module* represents a unit of code that a task invokes. A module serves a specific purpose, for example, creating a MySQL database and installing an Apache webserver. A *role* can be used to group a cohesive set of tasks and resources that together accomplish a specific goal, for example, installing and configuring MySQL.

3.3 Related Work

In this section, we discuss the existing studies on modeling and orchestrating Cloud and Edge application deployments, with respect to the two key challenges mentioned in the previous section. As a basis of our analysis, as appropriate, we refer to the recent relevant literature reviews, for example, [3–5, 12, 13].

There exist many approaches that enable specifying the deployment model of an application, for example, Ansible,[3] Chef,[4] Puppet,[5] OpenStack Heat,[6] and TOSCA [23]. Wurster et al. [3] compared these technologies with respect to their ability to model the essential aspects of a declarative deployment model. Among these approaches, TOSCA comprehensively supports the declarative deployment models in a technology agnostic way. As TOSCA is an open standard, the adoption of TOSCA enables more interoperable, distributed and open infrastructures [4, 17, 29].

When a deployment model is available, then an orchestrator can execute it and deploy the corresponding

[2] https://www.ansible.com/

[3] https://www.ansible.com/

[4] https://www.chef.io/

[5] https://puppet.com/

[6] https://docs.openstack.org/heat/latest/

components on the available resources. The recent surveys from Tomarchio et al. [5] and Luzar et al. [13] compared the existing orchestrators for the Cloud (including multi-clouds). The analysis covers both commercial products (e.g., Cloudify[7] and CloudFormation[8] and academic projects (e.g., SWITCH [11], MODAClouds [30], SeaClouds [31], MiCADO [32, 33], Occopus [15], and INDIGO-DataCloud [17, 29]) in terms of criteria such as portability, containerization, resource provisioning, monitoring, and runtime adaptation. The portability is typically supported by adopting open standards such as TOSCA [11, 17, 18, 34, 35] and OCCI (Open Cloud Computing Interface) [10]. As regards to resource provisioning, there exist a limited support for dynamic selection of resources, as well as for deployment and management of resources through IaC (or configuration management tools). As regards to monitoring, the collection of both system/infrastructure metrics and application metrics are supported for heterogeneous cloud environments. The key focus of the runtime adaptation support in the existing tools is threshold-based horizontal scaling. There are needs for policy-based adaptation as well as proactive data-driven adaptation of application deployments.

Kubernetes[9] and Docker Compose[10] are well-known container-based orchestration mechanisms. Both of them, though, have not been conceived to deal with complex applications that span across multiple heterogeneous container clusters and, to overcome this limitation, have been integrated with TOSCA-based approaches [21, 22, 35].

The containerization has been employed to deploy microservice-based applications on the Edge and hybrid Cloud-Edge infrastructures [19, 20]. There are also studies using OpenStack Heat [36] and TOSCA [37]. The key focus of these works is on the deployment of the applications while satisfying deployment constraints such as geographical constraints and inbound network communication restrictions.

Table 1 compares the existing projects and our proposed framework. There exist many studies on orchestrating applications on multi-clouds. However, a little research has been done on orchestrating applications on heterogeneous cloud-edge infrastructures, especially on portability and runtime management of application deployments. Multi-cloud orchestrators such as SWITCH, MiCADO, INDIGO-DataCloud, and Occopus leverage the TOSCA standard and containerization (mostly Docker) to support portability. Among these projects, INDIGO-DataCloud employs IaC (Ansible) for specific tasks such as deploying a Mesos cluster. SWITCH and MiCADO offer runtime adaptation capabilities in terms of vertical and horizontal resource scalability. In comparison to the existing studies, our focus is on supporting portability and runtime management for cloud-edge application deployments. To achieve portability, we rely on the TOSCA standard, containerization, and IaC. Regarding runtime adaptation, we aim to support the common structural changes to the deployment topology of an application, for example, adding, removing, and updating nodes or a fragment of the topology.

4 SODALITE@RT: A Runtime Environment for Orchestrating Applications on Cloud-Edge Infrastructures

The SODALITE runtime environment (*SODALITE@RT*) attempts to support the automated deployment and management of applications across cloud and edge infrastructures in a portable manner. To this end, to reduce the complexity introduced by the infrastructure and application heterogeneity, and to support the deployment portability, we adopt and extend the TOSCA standard to describe the deployment model of a managed heterogeneous distributed application. The *SODALITE@RT* also offers the capabilities of the enactment, monitoring, and adaptation of such TOSCA-based application deployments.

Figure 2 shows the high-level architecture of the *SODALITE@RT* platform, which consists of *TOSCA Repository*, *IaC Repository*, *Orchestrator*, *Monitoring System*, and *Deployment Refactorer*. *TOSCA Repository* includes TOSCA node types and templates, which represent both application and cloud-edge infrastructure components (types and instances). To implement the management lifecycle operations (e.g., create, install, and delete) of a defined component

[7] https://cloudify.co/

[8] https://aws.amazon.com/cloudformation/

[9] https://kubernetes.io/

[10] https://docs.docker.com/compose/

Table 1 Comparison of existing studies and our proposed framework

Study	Portability	Runtime management			Infrastructure	
		Deployment	Monitoring	Adaptation	Multi-Cloud	Cloud-Edge
INDIGO-DataCloud [17, 29]	+	+	+	∼	+	-
SWITCH [11]	+	+	+	∼	+	-
MiCADO [32, 33]	∼	+	+	∼	+	-
Occopus [15]	+	+	∼	-	+	-
Buzachis et al. [19]	-	+	∼	∼	-	∼
Kepes et al. [37]	+	+	-	-	-	∼
SODALITE@RT	+	+	+	+	+	+

+ : Sufficiently Support ∼: Partially Support - : Limited or No Support

type, we use the Ansible IaC language, which is one of the most popular languages amongst practitioners [7]. *IaC Repository* stores the reusable Ansible scripts corresponding to the realization and management of the component types (i.e., the TOSCA node types) in *TOSCA Repository*. *IaC Repository* offers RESTFul APIs for adding, removing, updating, retrieving Ansible artifacts. *TOSCA Repository* provides RESTFul APIs for adding, removing, updating, retrieving the definitions of TOSCA node types and node templates, and for finding node templates that satisfy a predicate over node properties. *Orchestrator* is responsible for (re)deploying a given application on the cloud-edge infrastructures by executing Ansible IaC scripts as necessary. It receives the initial deployment model (from a developer) or a new alternative deployment model (from *Deployment Refactorer*) as a TOSCA model instance. *Monitoring System* collects different metrics and events from both the application and cloud-edge infrastructure. It can also emit alerts, which are complex events over metrics or simple events. In response to the events from *Monitoring System*, *Deployment Refactorer* may decide to modify and reconfigure the current deployment model instance of the application. A developer can codify the desired deployment adaptation decisions as ECA (Event-condition-action) rules. *Deployment Refactorer* applies the codified decisions to derive a new alternative deployment model instance, and enact it through *Orchestrator*.

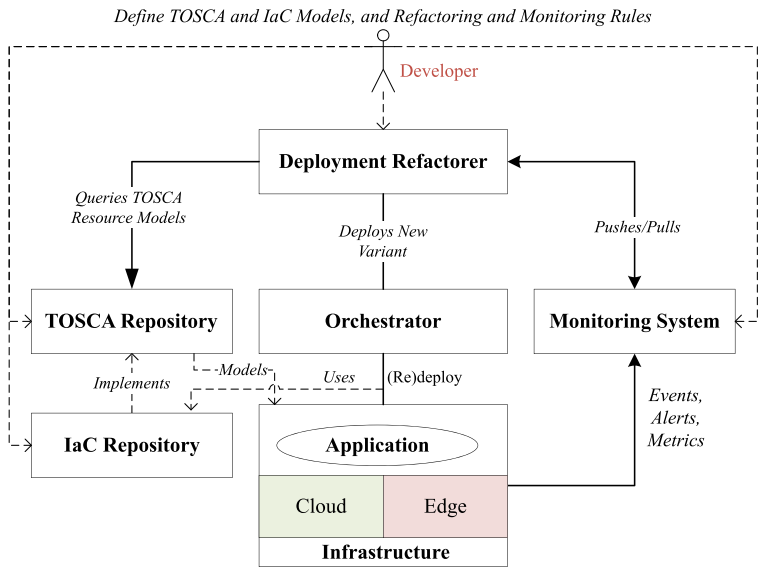

Fig. 2 Architecture of the SODALITE@RT environment

In the rest of this section, we discuss the *SODALITE@RT* environment in detail. We first present TOSCA and IaC based modeling of deployment models of cloud-edge applications, highlighting the mappings between cloud and edge resources and application components to TOSCA and IaC concepts. Next, we focus on deployment and monitoring of cloud-edge applications with our *Orchestrator* and *Monitoring System*. Finally, our support for the policy-based adaptation of the deployment models at runtime is discussed.

4.1 Modeling of Cloud and Edge Deployments with TOSCA and IaC

A deployment model describes the structure of an application to be deployed including all elements, their configurations, and relationships [3]. An element can be an application component (e.g., a microservice), a hosting platform or software system (e.g., MySQL database or Apache Web server), and an infrastructure resource (e.g., VM or network router). We apply the containerization to model software systems, and application components that are standalone or hosted on a hosting platform. As the containerization technology, we use Docker. As mentioned above, we use the TOSCA standard (Simple Profile in YAML 1.3) to represent edge resources, cloud resources, and containerized application components. To create and manage the instances of resources and components, we use Ansible IaC scripts. Table 2 shows the mappings between cloud-edge resources and components to TOSCA and Ansible concepts. In the rest of this section, we discuss the key mappings, and provide examples.

4.1.1 Modeling Cloud Resources

The common types of computing infrastructure resources are compute resources such as VMs and containers, and network resources such virtual communication networks and network devices. There exist different providers of such resources, for example, AWS and Openstack. The creation and management of resources is provider-specific, for example, AWS VM and Openstack VM. Thus, we use the TOSCA node types to model different types of compute resources, and employ Ansible scripts to implement the relevant management operations. The parameters or labels of resources are represented as the properties of TOSCA Node types (*OpenStack.VM* and *AWS.VM*), and the instances of resources are modeled as TOSCA node templates (Table 2).

A container runtime pulls containerized application components (e.g., container images) from the container registry and hosts them. To model the semantics of container runtime and containerized components, we introduce two TOSCA node types *DockerHost* and *DockerizedComponent*. Ansible playbooks are used to create the Docker engine in a host node, and to run Docker images. To specify a given containerized application component, a corresponding TOSCA template with the appropriate properties such as image names and environment variables should be created.

Figure 3 shows the snippets of the TOSCA node type and a node template for OpenStack VMs, and the Ansible playbook that implements the *create* management operation of the node type. The node type defines configuration properties, e.g., image and flavor, and specifies the requirements for protecting the VM with the security policies. The node template *vehicle-demo-vm* is an instance of this node type, and specifies the values for the properties of the node type, e.g., image as centos7 and flavor as m1.small. The task *Create VM* in the playbook uses the Ansible module *os_server* to create compute instances from OpenStack.

Figure 4 shows an example (snippets) for the TOSCA node type *DockerHost* and its instance, and the Ansible playbook that can instantiate the node type. The node type *DockerHost* defines a Docker container runtime. The property *registry-ip* specifics the Docker image repository. The capabilities of the node type indicate that it can host Docker containers (*DockerizedComponent*). The node type also defines the management operation for installing the Docker runtime in a host as a reference to the relevant Ansible playbook, which uses some Ansible roles to installs Docker, and some tasks to configure and start the Docker daemon.

4.1.2 Modeling Edge Resources

We use the container clusters as edge infrastructures, in particular, Kubernetes. The application components that target edge resources can be modelled as Kubernetes objects, such as a Kubernetes Deployment,

Table 2 Mapping from cloud-edge resources and components to TOSCA and Ansible

Edge/Cloud resource/Component	TOSCA	IaC (Ansible) and other files
VM Type	TOSCA Node Type (AWS, OpenStack, etc.)	Ansible playbooks that configure, create, and delete VMs using respective Ansible modules or collections (e.g., AWS EC2 module, OpenStack module)
VM Parameter/Label	TOSCA Node Properties (flavor, image, network, volume, etc.)	
VM Instance	TOSCA Node Template with values for node properties	Any possible configurations for a particular instance (e.g., userdata, cloud-configs, and additional ssh-keys)
Edge Cluster	TOSCA Node Type (Kubernetes Cluster)	
Edge Cluster Parameter/Label	TOSCA Node Properties (cluster access information, kubeconfig)	
Edge Node	TOSCA Node Type (Kubernetes Node)	
Edge Node Parameter/Label	TOSCA Node Properties (hardware architecture, accelerators, devicetree properties, etc.)	-
Edge Node with Accelerators (GPU, EdgeTPU, etc.)	TOSCA Node Template	-
Container Runtime Parameters	TOSCA Node Properties (image_registry_ip, etc.)	
Application Component on Cloud	TOSCA Node Type (DockerizedComponent)	Ansible playbook to configure and run the container image using Docker container module. Docker Image and possible configuration or source artifacts specific to the component need to be provided.
Application Component on Edge	TOSCA Node Type (Kubernetes Object or Helm chart)	Ansible playbook to configure and run an application component specified either as a Kubernetes Object or Helm chart. Possible configuration or source artifacts specific to the component can be provided.
Application Component Parameters	TOSCA Node properties (container image, ports, environment variables, etc.)	-
Application Component Instance	TOSCA Node Template	Any possible configuration or source artifacts for a particular component instance
Node Exporters Skydive Exporters IPMI Exporters	Ansible playbook that creates the VMs installs the exporter, launches the daemon and registers as a Consul service on creation; and deregisters from Consul on destruction	
Application-level Edge Exporters Accelerator-level Edge Exporters MQTT-level Edge Exporters *MQTT stands for The Standard for IoT Messaging.	-	A Helm chart installs the exporter into the Kubernetes cluster with the appropriate scrape annotation, which is picked up automatically by the Prometheus server. An Ansible playbook may install the Helm chart directly, using the Helm module. Consul service sync may be enabled to automatically synchronize Kubernetes Pods with the Consul service catalog.

```
sodalite.nodes.OpenStack.VM:                                    (a)
  derived_from: tosca.nodes.Compute
  properties:
    name: ...
    image: ...
    flavor: ...
    ...
  interfaces:
    Standard:
      type: tosca.interfaces.node.lifecycle.Standard
      create:
        ...
        implementation: openstack/vm/create.yml
      delete:
        ...
        implementation: openstack/vm/delete.yml
  requirements:
    - protected_by:
        capability: tosca.capabilities.Node
        node: sodalite.nodes.OpenStack.SecurityRules
        relationship: tosca.relationships.DependsOn
```

```
vehicle-demo-vm:                                                (b)
  type: sodalite.nodes.OpenStack.VM
  properties:
    name: vehicle-demo-vm
    image: centos7
    flavor: m1.small
    network: orchestrator-net
    security_groups: default,vehicle-iot
  requirements:
    - protected_by: security-rules-vehicle-iot
```

```
tasks:                                                          (c)
  - name: Create VM
    os_server:
      state: present
      name: "{{ vm_name }}"
      image: "{{ image }}"
      key_name: "{{ key_name }}"
      flavor: "{{ flavor }}"
      network: "{{ network }}"
      security_groups: "{{ security_groups }}"
```

Fig. 3 Snippets of, **a** the TOSCA node type for OpenStack VM, **b** a node template example for the node type, **c** the Ansible playbook for creating OpenStack VM (create.yml)

or can be encapsulated in Helm[11] charts. Helm is an application package manager for Kubernetes, which coordinates the download, installation, and deployment of Kubernetes applications. We developed TOSCA node types that handle the Kubernetes/Helm deployment onto edge clusters or edge nodes with specific accelerator types.

As shown in Fig. 5, the node type *sodalite.nodes.Kubernetes.Cluster* provides properties that define cluster access information (such as *kubeconfig*) and contains host capability for cluster-wide deployment via Kubernetes definitions or Helm charts. The node type *sodalite.nodes.Kubernetes.Node* defines the properties of an edge node, such as accelerators and CPU architecture, as well as the accelerators selectors (*gpu_selector* and *edgetpu_selector*). These selectors are represented as a mapping between accelerator type and the Kubernetes node labels it represents: for instance, an edge node that contains an NVIDIA GPU can be labeled with a node label - *nvidia.com/gpu*. The reason for such mapping is to specify a node affinity, such that application pods will be scheduled to a node with the specific accelerator, where a node affinity is set by patching values of Helm charts using Ansible. Figure 6 presents an example of a node template for a MySQL Helm chart deployment on the GPU edge node. It also shows the fragments of the corresponding TOSCA node type and the Ansible playbook that realizes the create management operation using the Ansible Helm module.

[11] https://helm.sh/

4.2 Deployment and Monitoring of Applications

In this section, we present the capabilities of the *SODALITE@RT* environment for deploying and monitoring applications over the cloud-edge infrastructures.

4.2.1 Deployment

There exist different infrastructure providers, and they generally offer the REST APIs to create and management the resources in their infrastructures. These REST APIs hide the underling low-level resource orchestrators, and aid achieving interoperability of heterogeneous infrastructures. Thus, we design and

(a)

```
sodalite.nodes.DockerHost:
  derived_from: tosca.nodes.SoftwareComponent
  properties:
    registry_ip: ...
    …
  interfaces:
    Standard:
      type: tosca.interfaces.node.lifecycle.Standard
      create:
        implementation:
          primary: docker/create_docker_host.yml
      delete:
        implementation:
          primary: docker/destroy_docker_host.yml
  capabilities:
    host:
      type: tosca.capabilities.Node
      valid_source_types: [ sodalite.nodes.DockerizedComponent ]
```

(b)

```
vehicle-demo-docker-host:
  type: sodalite.nodes.DockerHost
  properties:
    registry_ip: { get_input: docker-registry-ip }
  requirements:
    - host: vehicle-demo-vm
```

(c)

```
vars:
  pip_install_packages:
    - name: docker
tasks:
  - ...
  - name: Configure the docker for OpenStack
    service:
        name: docker
        state: restarted
roles:
  - geerlingguy.repo-epel
  - geerlingguy.docker
```

Fig. 4 Snippets of, **a** the TOSCA node type for Docker runtime, **b** a node template example for the node type, **c** the Ansible playbook for creating the node type (create_docker_host.yml)

implement our orchestrator as a meta-orchestrator that coordinates multiple low-level resource orchestrators. Figure 7 shows the main components in the architecture of the orchestrator.

– **Meta-Orchestrator** receives the TOSCA blueprint file describing the deployment model of the application through its REST API, validates the received model via *TOSCA Parser*, and uses

(a)

```
sodalite.nodes.Kubernetes.Cluster:
  derived_from: tosca.nodes.Compute
  capabilities:
    host:
      type: tosca.capabilities.Compute
      valid_source_types: [ sodalite.nodes.Kubernetes.Kind,
                            sodalite.nodes.Kubernetes.Definition ]
  properties:
    kubeconfig: ...
    username: ...
sodalite.nodes.Kubernetes.Node:
  derived_from: tosca.nodes.Compute
  capabilities: ...
  properties:
    name: ..
    gpu_selector:
      type: map
      default: { "key": "accelerators/gpu", "value": "true" }
    edgetpu_selector: ...
    cpus:
      type: integer
      default: 0
    gpus: ..
    ...
```

(b)

```
edge-cluster:
  type: sodalite.nodes.Kubernetes.Cluster
  properties:
    kubeconfig: ~/.kube/config
    username: { get_input: frontend_user }
  attributes:
    public_address: { get_input: frontend_address }
edge-gpu-node:
  type: sodalite.nodes.Kubernetes.Node
  properties:
    name: gpu-node
    gpus: 1
    gpu_selector: { "key": "nvidia.com/gpu",
                    "value": "1" }
    cpus: 1
```

(c)

```
tasks:
  - name: "Create from
              Kubernetes definition file"
    community.kubernetes.k8s:
      state: present
      kubeconfig: "{{ kubeconfig }}"
      src: "{{ path }}"
```

Fig. 5 Snippets of, **a** TOSCA node types for Kubernetes edge clusters and nodes with accelerators, **b** node template examples of the node types, and **c** an Ansible playbook for creating Kubernetes environment from their definitions

```
sodalite.nodes.Kubernetes.Definition.Helm.Node:
  derived_from: sodalite.nodes.Kubernetes.Definition.Helm
  requirements:
    - host: ...
    - kube_node:
        node: sodalite.nodes.Kubernetes.Node
  interfaces:      ...
      operations:
       create:
         inputs: ...
         implementation:
           primary: playbooks/create_from_helm.yaml
```

(a)

```
- name: Install chart
  community.kubernetes.helm:
    name: "{{ helm_name }}"
    namespace: "{{ helm_namespace }}"
    chart_ref: "{{ actual_helm_chart }}"
    values_files: "{{ helm_values_files }}"
    values: "{{ actual_helm_values }}"
```

(b)

```
mysql-deployment-via-helm-on-edgetpu:
  type: sodalite.nodes.Kubernetes.Definition.Helm.Node
  properties:
    name: mysql-release-1-from-helm-on-edgetpu
    namespace: default
    chart: stable/mysql
    chart_version: latest
    repo_name: stable
    repo_url: "https://charts.helm.sh/stable"
    keep_repo: false
    values:
      replicas: 1
      persistence:
        enabled: False
  requirements:
    - host: edge-cluster
    - kube_node: edge-gpu-node
```

(c)

Fig. 6 Snippets of, **a** the TOSCA node type for Helm, **b** an Ansible playbook for Helm charts deployment, and **c** the node template for a MySQL Helm chart deployment on a GPU edge node

IaC-based Orchestration Layer to provision the resources and deploy components in the deployment model. The deployment states of TOSCA nodes and relationships and their design time (node properties) and runtime (node attributes) parameters are stored in a database so that the

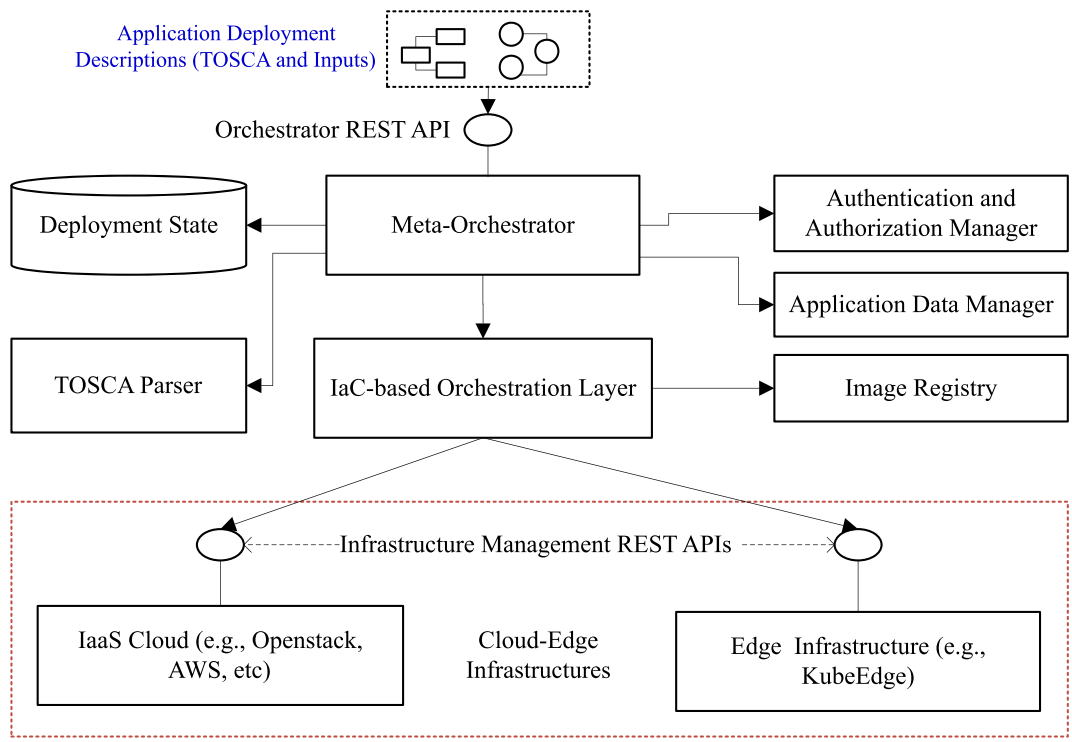

Fig. 7 Architecture of the *SODALITE@RT* orchestrator

deployment status can be monitored via the REST API and the next deployment iterations can be compared against current version. The deployment version comparison helps to efficiently update and reconfigure running application topology, handling only those nodes/relationships that need modifications.

- **TOSCA Parser** parses and validates syntax of TOSCA models based on the version v1.3 of TOSCA Simple YAML Profile specification. It is a general purpose tool that can be used by any other orchestrators to validate TOSCA models.
- **Authentication and Authorization Manager** handles the user and secrets management across the whole SODALITE stack and Orchestrator REST API in particular. Each TOSCA blueprint and deployment is associated to a project domain, an access to which requires an access token with specific JWT (JSON Web Token) claims. Given that the access token is valid, one can perform orchestration actions such as deployment or undeployment, getting information about a deployment state as well as performing deployment updates. In order to hide sensitive data being passed as inputs for the TOSCA blueprints, special directives are implemented on the inputs that allow to retrieve the secrets during the deployment operation. The secrets are registered by the users with the Authentication and Authorization Manager and the IDs of the secrets are passed to the inputs. Moreover, Orchestrator implements an encrypted storage, where the deployment state is securely stored and can be retrieved with the storage key.
- **Application Data Manager** incorporates various transfer protocols and endpoints to achieve transparent data management across multiple infrastructure providers. The data transfer pipelines can be implemented by using the ETL (Extract, transform, load) tools as well as IaC. Apart from using Ansible's built-in modules for data transfer, for example, files modules (*copy* and *fetch*) or URI modules (*get_url* and *uri*), other advanced data transfer protocols such as SFTP, GridFTP can be incorporated on the supported targets, which will provide advanced features such as security, performance and third-party data transfer.
- **Image Registry** is an internal or external repository of container images. The internal repositories should provide APIs to pull the images through IaC. We use Docker Hub for storing images of the application components in the case studies.
- **IaC-based Orchestration Layer** interfaces with various APIs and endpoints, for example, IaaS management APIs and platforms APIs in order to request the resources needed for the deployment, configure and deploy the application components. It pulls and executes Ansible playbooks that implement the lifecycle operations for nodes/relationships defined in a TOSCA model from the *IaC Repository*. Ansible provides several convenient modules, which enable interaction with a particular platform. For OpenStack, there are several modules available, allowing the creation of various components of the virtual infrastructure: virtual machines (*os_server*), networks (*os_networks*), block storage (*os_volume*), and so on. It also has modules (*cloud_modules*) for creating and managing resources in Azure, AWS, and GCP public Clouds. The dedicated module *k8s* for management of Kubernetes objects allows creation of deployment pods and services.

Due to the fact that the application development and deployment are nowadays continuous, for example, shipping new releases frequently, there will be updates in the previously deployed application topology. Alternatively, the updates can be triggered on the infrastructure level in order to satisfy QoS parameters, for example, increase of responsiveness of the application by provisioning greater resources. Therefore, it is a task of the Orchestrator to handle these updates and implement redeployment actions on deployed application topology. Application redeployment is requested by submitting the new version of the application deployment topology via the Orchestrator REST API. Current implementation of redeployment is to have the new version and the old version coexist (with a HA proxy forwarding requests to the correct version) and tearing down the old version once the new version is deployed and can be used by end-users.

4.2.2 Monitoring

The deployed application is continuously monitored to collect the metrics that can be used by the components such as *Deployment Refactorer*. As shown in Fig. 8, the monitoring system is composed of the following elements: a number of *Exporters* that collect and

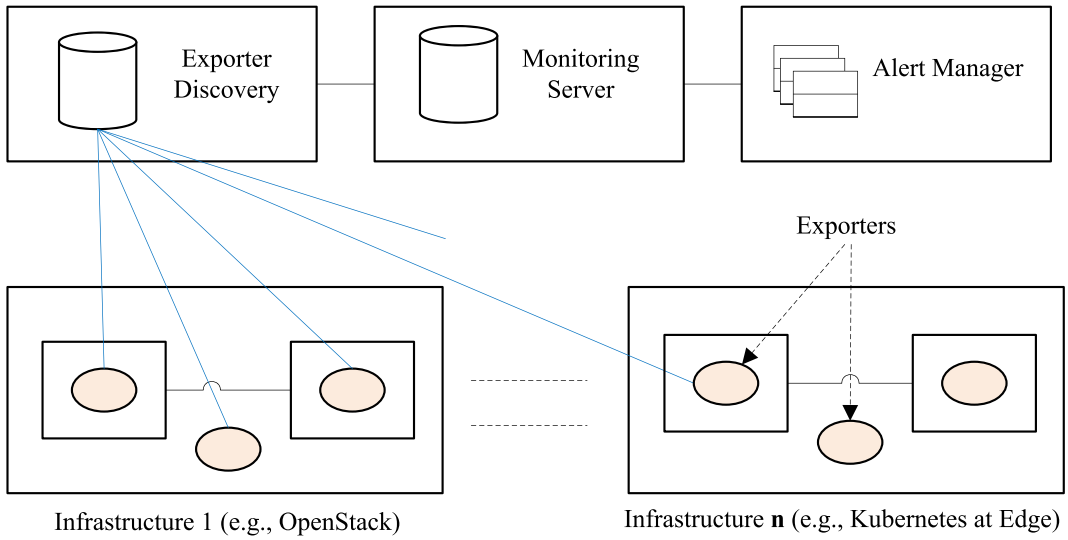

Fig. 8 Architecture of the *SODALITE@RT* monitoring system

publish relevant information about the resources on which they are installed, *Exporter Discovery service* that discovers and allows registering exporters, *Monitoring Server)* that gathers all the information via exporters or directly scraping nodes, and *Alert Manager* that receives data from *Monitoring Server* and emit alerts by evaluating a set of *rules* over the received data.

Exporters are in charge of measuring their targeted metrics across the heterogeneous infrastructure. There exist four types of exporters: node exporter, Skydive exporter, IPMI (Intelligent Platform Management Interface) exporter, and edge exporter. Node exporter is used to gather information such as CPU, input/output, and memory usage from virtual machines. Skydive exporter enables collecting various network metrics such as network flow and traffic metrics using the Skydive tool[12]. IPMI exporter gathers low-level information (e.g., power consumption) from IPMI-compatible sensors installed on the physical nodes in the infrastructure.

Edge nodes are expected to run a node exporter and accelerator-specific metric exporters for any attached heterogeneous accelerators (e.g., Edge TPU and GPU). As with the cloud VMs, the node exporter is responsible for gathering and exposing general information about the node, whereas the accelerator-specific exporters provide specific insight into the attached accelerators. This may include aspects such as the number of devices available, the load average, or thermal properties.

The Ansible playbooks that are responsible for setting up nodes also deploy the exporters. The configuration parameters for exporters can be provided using TOSCA node properties. Figure 9 shows a snippet of an Ansible playbook that installs the EdgeTPU exporters into the edge nodes in a Kubernetes cluster. It uses the Ansible modules for executing the relevant Helm charts.

Monitoring Server gathers data from all of the different exporters running all over the computing infrastructure. It queries *Exporter Discovery* to find information about exporters. The exporters publish the collected data through their HTTP endpoints. The collected real-time metrics are recorded in a time series database.*Alert Manager* receives the collected real time metrics from the monitoring server, and triggers different types of alerts based on a set of rules. Figure 10 defines an alert rule to generate the alert *HostHighCPULoad* when the CPU load in the node is greater than 80%.

4.3 Adaptation of Application Deployments

In response to the data collected and events received from *Monitoring System*, *Deployment Refactorer* decides and carries out the desired changes to the current deployment of a given application. To allow a

[12] http://skydive.network/

```
tasks:
 - name: Add Prometheus Community repository
   community.kubernetes.helm_repository:
     name: prometheus-community
     repo_url: https://prometheus-community.github.io/helm-charts
 - name: Add Adaptant repository
   community.kubernetes.helm_repository:
     name: adaptant
     repo_url: https://adaptant-labs.github.io/charts/adaptant
 - name: Install EdgeTPU exporter
     community.kubernetes.helm:
     name: edgetpu-exporter
     chart_ref: adaptant/edgetpu-exporter
     release_namespace: "{{namespace}}"
```

Fig. 9 Snippet of an Ansible playbook for installing the EdgeTPU exporter

software engineer to define the deployment adaptation decisions, we provide an ECA (event-condition-action) based policy language. Figure 11 the key concepts of the policy language. A policy consists of a set of ECA rules.

- **Events and Conditions**. A *condition* of a rule is a logical expression of events. We consider two common types of events pertaining to the deployment model instance of an application: deployment state changes and application and resource metrics. The former event type captures the state of a node or relation in a deployment model instance, which are fourfold: *Added*, *Removed*, *Updated*, and *ConstraintsViolated*. The *Updated* event type comprises the changes to the properties, requirements, capabilities of a node and the properties of a relation. The *ConstraintsViolated* event type indicates the violation of the constraints on deployment states, for example, removal or failure of a CPU (in a node representing a VM) can violate the constraint that the number of CPUs should be greater than a given threshold. The application and resource metric events include (raw or aggregated) primitive metrics collected from the running deployment, for example, average CPU load, as well as alerts or complex events that represent predicates over primitive metrics, for example, the above-mentioned *HostHighCPULoad* alert. The application components may also generate custom events, for example, a component (the user app) in the Vehicle IoT application periodically does a reverse geocoding of the GPS coordinates and when there is a country change it triggers a notification. Moreover, time of the day or other context conditions can also be the conditions of deployment adaptation rules.
- **Actions**. The actions primarily include the common change operations (*Add*, *Remove*, and *Update*) and the common search operations (*Find* and *EvalPredicate*) on nodes, relations, and their properties. Additionally, the custom actions can be implemented and then used in the deployment adaptation rules, for example, actions for predicting performance of a particular deployment model instance or predicting workload. To ensure the safe and consistent changes to the

```
- alert: HostHighCPULoad
  expr: 100 - (avg by(instance,os_id) (irate(node_cpu_seconds_total{mode="idle"}[5m])) * 100) > 80
  for: 5m
  labels:
  severity: warning
  annotations:
  summary: "Host high CPU load (instance {{ $labels.instance }})"
  description: "CPU load is > 80%\n VALUE = {{ $value }}\n LABELS: {{ $labels }}"
```

Fig. 10 An alerting rule for indicating high CPU usage in a node

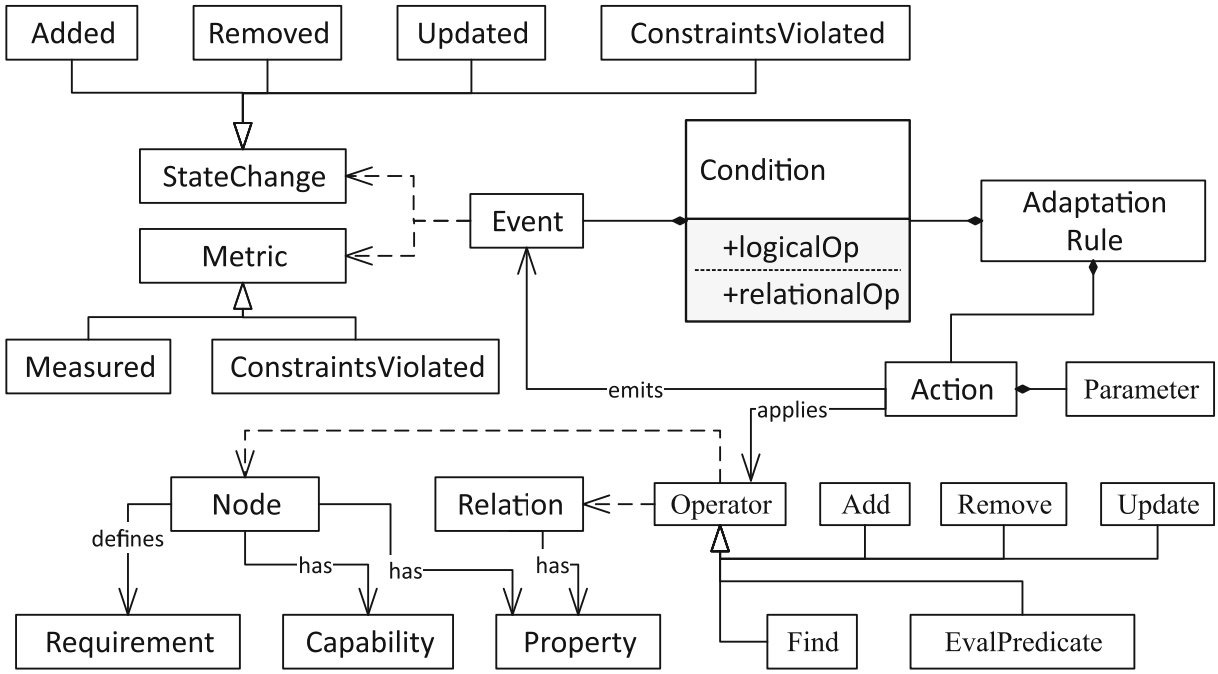

Fig. 11 Meta-model of the deployment adaptation policy language

deployment model instance, *Deployment Refactorer* makes the change operations to a local representation (a Java Object model) of the deployment model (represented using the concept of models@runtime [38]). Once the adaptation rules in a rule session are executed, *Deployment Refactorer* translates the current local object model to a TOSCA file, and calls the update API operation of the Orchestrator with the generated file. To implement search actions, *Deployment Refactorer* uses the corresponding API operations provided by *TOSCA Repository*.

There exist dependencies between adaptation decisions. An enactment of a given adaptation decision may require the enactment or prevention or revocation of some other adaptation decisions. To capture these dependencies, we introduce an action to generate custom events. A rule can emit an event indicating the state (e.g., completion) of the enactment of an adaptation decision. The dependent rules can use that event in their conditions.

– **Execution**. The correct ordering of the rules as well as that of the actions within each rule are required to achieve a desired outcome. The rules are independent and are activated based on their conditions. When multiple rules are activated at the same time, the priorities of the rules can be used to resolve any conflicts. Within a rule, if-then-else conditional constructs can be used to order the actions.

The *Deployment Refactorer* uses a policy engine to enact the deployment adaptation policies. It supports addition, removal, and update of policies. It can parse given policies, process events and execute the policies. The policy rules are triggered as their conditions are satisfied, and the desired changes are propagated to the deployment model instance.

Figure 12 show an example of a deployment adaptation rule that reacts to the event *LocationChangedEvent* by un-deploying a data processing service deployed in a VM located in a data center at the previous location (de-Germany), and deploying the same service in a VM from a data center at the new location (it-Italy). A predicate over the TOSCA node properties *location* and *service name* is used to find the correct TOSCA node template.

```
import eu.sodalite.TOSCARepositoryAPI repoAPI;

rule "location_change_from_de_to_it"
  when
    $f1 : LocationChangedEvent(preLoc == "de", currentLoc == "it") and
    $dm : DeploymentModel()
  then
    $dm.removeNode("(?location = \"" + $f1.getPreLoc() + "\" )
                    && (?service-name = \"" + $f1.getServiceName() + "\" )");
    $dm.addNode(repoAPI.find("(?location = \"" + $f1.getCurrentLoc() + "\" )
                    && (?service-name = \"" + $f1.getServiceName() + "\" )"));
    emit NodeReplaced($f1.getServiceName());
  end
```

Fig. 12 A snippet of a deployment adaptation rule

5 SODALITE@RT Prototype Implementation

We implemented the *SODALITE@RT* environment using a set of open source projects/tools. Figure 13 shows the key components of the prototype implementation and the open source projects/tools used. The implementation of the *SODALITE* platform is maintained at GitHub.[13]

We implemented the meta-orchestrator with xOpera,[14] which supports TOSCA YAML v1.3. The current features of xOpera includes: 1) registering, removing, and validating TOSCA blueprints, 2) deploying and undeploying the applications based on the registered blueprints, and 3) monitoring the progress of deployment and undeployment operations. xOpera executes the blueprints through Ansible playbooks, which implement the necessary infrastructure management operations. xOpera uses PostgreSQL to store the TOSCA blueprints and the states of the application deployments. The token-based authentication and role-based authorization were implemented using Keycloak[15] identity and access management solution. We use Docker[16] as the container technology. We employ Ansible and Apache NiFi[17] to implement data pipelines that can transfer application data across various platforms and storage systems such as Amazon S3, Google Storage, Hadoop file system (HDFS), and Apache Kafka message broker.

We implemented the policy engine using the Drools business rule management system.[18] Drools supports both production business rules and complex event processing. It also offers a web UI and an Eclipse IDE for authoring policies, and fully supports the DMN (Decision Model and Notation) standard for modeling and executing decisions. We implemented the SODALITE monitoring system using Prometheus[19] and Consul.[20] Prometheus implements exporters, the monitoring server, and the alert manager, while Consul implements the exporter discovery.

The *SODALITE@RT* currently supports five key types of infrastructures: edge (Kubernetes[21]), private cloud (OpenStack[22] and Kubernetes), public cloud (AWS), federated cloud (EGI OpenStack[23]), and HPC (TORQUE[24] and SLURM[25]). The HPC support was partially presented in a previous publication [39]. The examples for orchestrating applications on each type of these infrastructures can be found in our GitHub repository.

In addition to the runtime environment, the *SODALITE* project also includes a development environment, implemented as an Eclipse plugin to support authoring defect-free TOSCA blueprints and Ansible scripts. We have presented our development environment and its capabilities in our previous publications [39–43].

6 Case Study: Realization of Vehicle IoT with SODALITE@RT

This section illustrates three different scenarios in the Vehicle IoT case study that have been implemented with the *SODALITE@RT* platform. The selected scenarios

[13] https://github.com/SODALITE-EU

[14] https://github.com/xlab-si/xopera-opera

[15] https://www.keycloak.org/

[16] https://www.docker.com/

[17] https://nifi.apache.org/

[18] https://www.drools.org/

[19] https://prometheus.io/

[20] https://www.consul.io/

[21] https://kubernetes.io/

[22] https://www.openstack.org/

[23] https://www.egi.eu/

[24] https://adaptivecomputing.com/cherry-services/torque-resource-manager/

[25] https://slurm.schedmd.com/

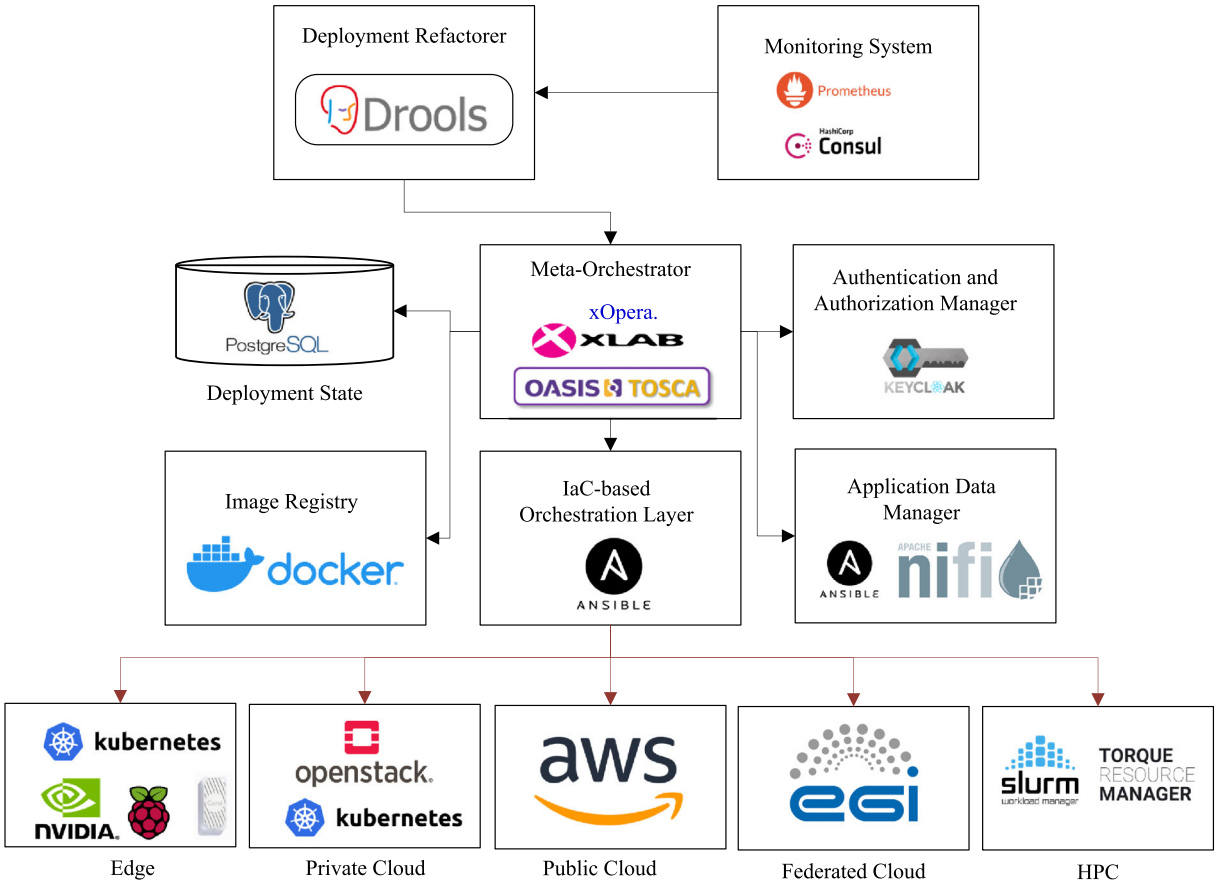

Fig. 13 Prototype implementation of the *SODALITE@RT* environment

demonstrate deployment, monitoring, location-aware redeployment, and alert-driven redeployment. Each scenario covers deployment modeling, actual deployment, monitoring, and deployment adaptation. The case study implementation can be found in the SODALITE project's GitHub repository[26][27] and the industrial partner's GitHub repository.[28] The recorded demonstration videos of the three scenarios are also available in the GitHub.[29] In this section, we first provide an overview of the deployment of the vehicle IoT application with the *SODALITE@RT*. Then, we present three scenarios and a performance evaluation of the *SODALITE@RT* with respect to the use cases.

[26] https://github.com/SODALITE-EU/iac-management

[27] https://github.com/SODALITE-EU

[28] https://github.com/adaptant-labs

[29] https://github.com/IndikaKuma/SODALITEDEMOS

6.1 Deployment of the Case Study

Figure 14 shows the deployment of the vehicle IoT case study in the SODALITE testbeds. It includes the key components used by the three scenarios. The edge testbed consists of 3 nodes managed by Kubernetes. Three edge devices are Raspberry Pi 4, Google Coral AI Dev Board, and NVIDIA Jetson Xavier NX. Their accelerators are NCS2 (Neural Compute Stick 2), EdgeTPU, and NVDLA x2. The cloud testbed provisions virtualized resources (e.g., virtual machines and containers) managed by OpenStack and Kubernetes. Furthermore, the cloud testbed hosts the development environment, which contains the SODALITE CI/CD server and deployed SODALITE components. It offers Ubuntu 18.04 VMs in flavors small(1 vCPUs and 2GB RAM), medium (2 vCPUs and 4GB RAM), large (4 vCPUs and 8 GB RAM), and xlarge (8 vCPUs and 16 GB RAM).

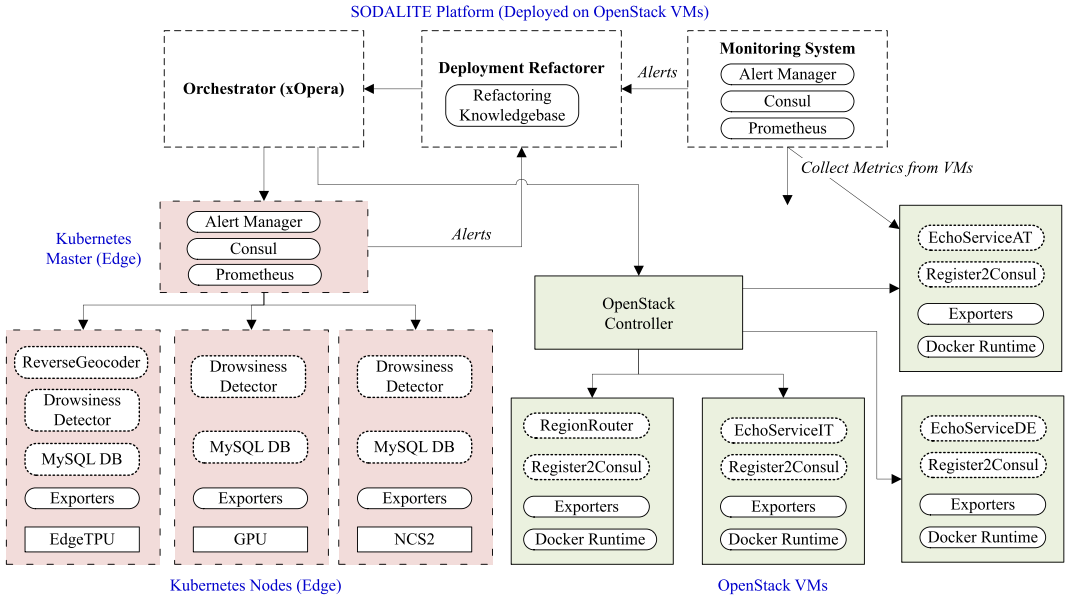

Fig. 14 Vehicle IoT case study deployment with the SODALITE@RT environment

Each SODALITE@RT component (i.e., the orchestrator, the deployment refactorer, and the monitoring system) is deployed on medium VMs. The inference service drowsiness detector, the MySQL storage, and the reverse geocoder service are deployed on edge nodes. The region router and three echo services are deployed on cloud VMs. The echo services are used to simulate the services deployed in the data centers at three different countries.

6.2 Location-aware Redeployment

This case demonstrates the capability of the *SODALITE@RT* to redeploy an application in response to changes in legal jurisdiction, helping deployed applications maintain both service continuity and meet their compliance requirements as vehicles travel between countries. An in-vehicle driver monitoring service making use of biometric data (classified as special category data by GDPR Art. 9) for drowsiness detection and alerting requires physical locality of processing for both latency and regulatory compliance reasons, limiting the ability to carry out cross-border data transfers. In vehicles with sufficient resources, this is ideally carried out directly in the vehicle itself, while in others, it may be necessary to stream data to the cloud and carry out the analysis in-cloud.

A region router handles region-specific routing for in-bound REST API requests originating from the frontend application (the user app). In the case where a suitable region is available, in-bound requests are passed through directly. Where no matching region is provisioned, a notification is sent to the deployment refactorer in the form of a JSON payload that designates the affected service, the country being left, and the country being entered. The deployment adaptation rule described in Section 4.3 is related to the implementation of this scenario.

6.3 Alert-driven Redeployment: Cloud Alerts

This scenario demonstrates the capability of reacting to the events from cloud resource monitoring. To prevent over/under utilization of resources, the vehicle IoT application needs to be redeployed based on the CPU usage of the cloud VMs that host the application. We first modelled and deployed the initial application in a medium flavor VM, and created two alerting rules: one for the alert *HostHighCPULoad* (CPU load $>$ 80%) and other for the alert *CPUUnderUtilized* (40% $>$ CPU load $<$ 50%). The deployment adaptation rules for reacting to these two alerts are also defined: redeploy the application in a Medium VM for the alert *CPUUnderUtilized*, and redeploy the

```
- alert: TPUTempNormal
  expr: edgetpu_temperature_celsius < 70.0
  labels:
  severity: info
  annotations:
  summary: "Normal EdgeTPU device temperature"
- alert: TPUTempCritical
  expr: edgetpu_temperature_celsius > 95.0
  labels:
  severity: critical
  annotations:
  summary: "Critical EdgeTPU device temperature"
```

Fig. 15 Alerting rules for EdgeTPU temperature monitoring

application in a large VM for the alert *HostHighCPULoad*. Next, we stressed the VM to change the CPU load, and observed alert generation, receiving events and triggering of adaptation rules, and finally successful redeployment.

6.4 Alert-driven Redeployment: Edge Alerts

This scenario demonstrates the capability of the edge-based monitoring and alerting to throttle an application deployment that has exceeded thermal tolerances. In this case, we consider an AI inference workload running on an edge-attached EdgeTPU accelerator. The EdgeTPU itself has a narrowly defined operating temperature range, where exceeding certain levels can produce erratic behavior, ranging from silent (and difficult to debug) inference failure, to physical damage to the package itself. While thermal trip points can be configured to physically power off the device where a critical temperature being exceeded could damage the hardware itself, the *SODALITE@RT* platform is leveraged to mitigate the risks of rising temperature inducing inference failure.

The EdgeTPU run-time libraries[30] are provided in *-max* and *-std* versions, the former providing the highest clock rate (500MHz) and performance, while producing the highest operating temperature. The latter divides the input clock in half, running at a reduced clock rate (250MHz), providing reduced performance and producing a lower operating temperature. We created two different variants of the inference application containers, each linked against one version of the run-time library, using an appropriate accelerator-specific base container.[31] The *EdgeTPU exporter*[32] provides EdgeTPU-specific metrics, including the number of devices and per-device temperature, which are scraped by the monitoring server. Based on these metrics, alerting rules that allow for different actions to be taken at different thermal trip points are also defined (see Figure 15).

Figure 16 illustrates the switching between the *-max* variant and the *-std* variant of the inference service depending on the measured temperature of the EdgeTPU device. First, the default *-max* variant of the inference application is deployed to the edge node by the orchestrator. As other workloads are deployed onto the node, the ambient temperature within the enclosure rises, slowly increasing the EdgeTPU device temperature. The monitoring server, using the defined alerting rules, identifies that a thermal limit has been passed, and fires the alert *TPUTempCritical*. The alert manager receives the alert and notifies the deployment refactorer, which identifies a throttling measure as a possible mitigating solution (by selecting the *-std* variant of the inference service), and informs the orchestrator by providing the revised TOSCA blueprint. The orchestrator updates the deployment on the edge node. When the EdgeTPU device temperature drops below 70, and the alert *TPUTempNormal* is generated, which initiates the switching back to the *-max* variant.

[30] https://coral.ai/software/

[31] https://github.com/adaptant-labs/accelerator-base-containers

[32] https://github.com/adaptant-labs/edgetpu-exporter

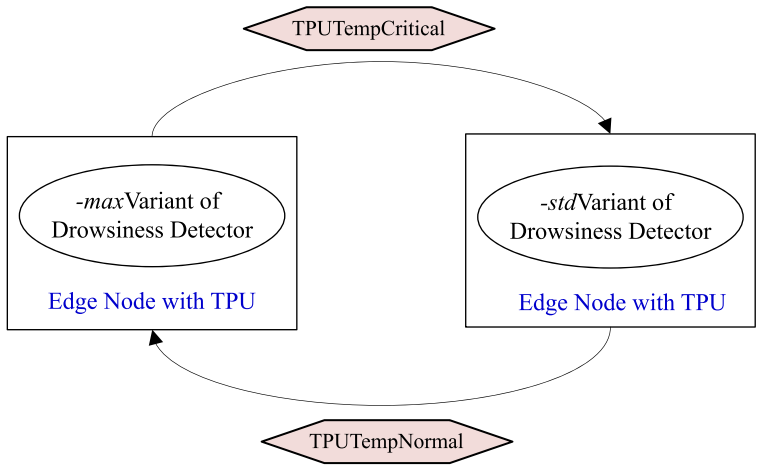

Fig. 16 Switching between deployment variants as the edge device temperature changes

6.5 Performance Evaluation

To get an insight of performance overhead of the orchestrating capabilities of the *SODALITE@RT*, we measured the average time to deploy and undeploy the use cases. In addition to the vehicle IoT application, we also consider the cloud-based use case of the *SODALITE* project, namely the snow use case, which implements a deep learning pipeline for assessing the availability of water on mountains based on snow images. The snow use case consists of 10 components (containerized microservices) and a MySQL database, and is deployed on two medium VMs.

Table 3 shows the results of the performance evaluation. It reports the average values over 10 runs of deployment and undeployment operations. The deployment overhead is between 134.72-424.7 seconds, and the undeployment overhead is between 43.2-114.6 seconds. Since the *SODALITE@RT* uses a meta-orchestrator that employs IaC for orchestrating applications, the performance of the low-level orchestrators and IaC tools (e.g., Ansible) can potentially determine the overhead incurred by the *SODALITE@RT*. Thus, we consider this overhead acceptable since the *SODLITE@RT* can benefit from the performance improvements made at the low-level orchestrators and IaC tools, which are generally industrial tools, and have active developer and user communities.

7 Supported Scenarios

In the previous section, we provided several scenarios within the vehicle IoT use case that were supported using the *SODALITE@RT* framework. In this section, we provide a general discussion on the potential scenarios, which can be implemented using the framework.

– **Machine/deep learning pipelines.** A ML/DL pipeline consists of a set of steps such as data preprocessing, feature engineering, training and tuning models, evaluating models, and deploying and monitoring models. Typically, the training process can be computationally intensive, and offloaded to more compute-capable cloud or HPC clusters. However, the models can be deployed at the edge as microservices to provide the fast inferences to the end-users. The inference performance needs to be continuously monitored. When new training data becomes available or the inference performance drops below a given threshold, the models

Table 3 Average deployment and undeployment times for use cases

	Vehicle IoT use case			Snow use case
	Scenario 1	Scenario 2	Scenario 3	
Deployment Time	245s	121.86s	134.72s	424.7s
Undeployment Time	60s	43.2s	48s	114.6s

need to be retrained at the cloud and redeployed on the edge. This heterogeneity and dynamism of ML/DL pipelines makes the *SODALITE@RT* framework a suitable candidate to orchestrate them. For example, the orchestrator can deploy the inference service to the edge, transfer training data to the HPC/cloud cluster, submit the job for training and monitors the job execution. After the job is executed, the inference model can then be transferred by the orchestrator via data management utilities and integrated into the business logic of the service at runtime. The monitoring system can be used to monitor the model performance, and the deployment refactor can be used to trigger necessary resource reconfigurations.

– **Deployment switching.** The increasing heterogeneity of computing resources gives rise to a very large number of deployment options for constructing distributed multi-component applications. For example, the individual components of an application can be deployed in different ways using different resources (e.g., a small VM, a large VM, and an edge GPU node) and deployment patterns (e.g., a single node, a cluster with load balancer, with or without cache, and with or without firewall). A valid selection of deployment options results in a valid deployment model variant for the application. Different deployment variants can exhibit different performance under different contexts/workloads. Hence, the ability to switch between deployment variants as the context changes can offer performance and cost benefits. The deployment refactorer was designed to support deployment switching use cases. To enable deployment model switching, we are currently developing a learning based efficient approach that can accurately predict the performance of all possible deployment variants using the performance measurements for one or few subsets (samples) of the variants.
– **Orchestrating and managing applications on dynamic environments.** As a deployment environment evolves overtime, the new resources will be added and the existing resources will be removed or updated. Moreover, as discussed within the vehicle IoT use case, the precise requirements of the workloads are also subject to change based on factors such as the regulatory environment, the privacy preferences of the driver, resource availability, requisite processing power, and connectivity state. A key usage scenario for the *SODALITE@RT* is to enable deploying and managing applications on dynamic heterogeneous environments. The monitoring system can collect metrics from different environments and trigger alerts. In response to these alerts, the refactorer can make necessary changes to the deployment instances at runtime. In addition to the rule-based decision making, we are also extending the refactorer with a learning-based decision support for performance prediction, deployment switching, and performance anomaly detection. The orchestrator is also being extended to support more infrastructure options, and the graceful and efficient update of running deployment instances.

8 Conclusion and Future Work

The *SODALITE@RT* platform enables the deployment of complex applications on heterogeneous cloud and edge infrastructures. It supports the modeling of heterogeneous application deployments using the TOSCA open standard, deploying such applications based on created models, and monitoring and adapting application deployments. It also utilizes the containerization technology (Docker and Kubernetes) to encapsulate applications and execution platforms, and IaC (Infrastructure as Code) to provision heterogeneous resources and deploy applications based on the TOSCA-based deployment models. We validated the capabilities of our platform with an industrial case study across a range of real-world scenarios. The TOSCA standard, the containerization, and the IaC approach enabled developing portable deployment models for heterogeneous cloud-edge applications. They also enabled managing such applications at runtime since moving applications' components from one deployment environment to another becomes more manageable.

We will be conducting future work in two key directions. On the one hand, we will further develop the *SODALITE@RT* by incorporating new infrastructures such as Open FaaS and Google Cloud, and by completing the integration of the runtime layer within the overall *SODALITE* stack. On the other hand, the monitoring and deployment adaptation support will be extended with the federated monitoring, and

the machine learning-based approaches to switching between different deployment variants and detecting performance anomalies. Moreover, we are also developing the distributed control-theoretical planners that can support vertical resource elasticity for containerized application components that use both CPU and GPU resources [44]. The integration of such capabilities with the deployment refactorer will also be investigated.

Acknowledgements This work is supported by the European Commission grant no. 825480 (H2020), SODALITE. We thank all members of the SODALITE consortium for their inputs and feedback to the development of this paper.

References

1. Inc, G.: Gartner forecasts worldwide public cloud revenue to grow 17.5 percent in 2019. Gartner, Stamford (2018)
2. Ren, J., Zhang, D., He, S., Zhang, Y., Li, T.: A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. ACM Comput. Surv. **52**(6). https://doi.org/10.1145/3362031 (2019)
3. Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., Soldani, J.: The essential deployment metamodel: a systematic review of deployment automation technologies. SICS Softw.-Intens. Cyber-Phys. Syst. **35**(1), 63–75 (2020). https://doi.org/10.1007/s00450-019-00412-x
4. Bergmayr, A., Breitenbücher, U., Ferry, N., Rossini, A., Solberg, A., Wimmer, M., Kappel, G., Leymann, F.: A systematic review of cloud modeling languages. ACM Comput. Surv. **51**(1). https://doi.org/10.1145/3150227 (2018)
5. Tomarchio, O., Calcaterra, D., Modica, G.D.: Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks. J. Cloud Comput. **9**(1), 49 (2020). https://doi.org/10.1186/s13677-020-00194-7
6. Weerasiri, D., Barukh, M.C., Benatallah, B., Sheng, Q.Z., Ranjan, R.: A taxonomy and survey of cloud resource orchestration techniques. ACM Comput. Surv. **50**(2). https://doi.org/10.1145/3054177 (2017)
7. Guerriero, M., Garriga, M., Tamburri, D.A., Palomba, F.: Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In: 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 580–589. IEEE (2019)
8. Kumara, I., Han, J., Colman, A., van den Heuvel, W.-J., Tamburri, D.A., Kapuruge, M.: Sdsn@rt: A middleware environment for single-instance multitenant cloud applications. Softw. Pract. Exper. **49**(5), 813–839 (2019). https://doi.org/https://doi.org/10.1002/spe.2686
9. Kumara, I., Han, J., Colman, A., Kapuruge, M.: Runtime evolution of service-based multi-tenant saas applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) Service-Oriented Computing, pp. 192–206. Springer, Berlin (2013)
10. Zalila, F., Challita, S., Merle, P.: Model-driven cloud resource management with occiware. Futur. Gener. Comput. Syst. **99**, 260–277 (2019). https://doi.org/10.1016/j.future.2019.04.015, http://www.sciencedirect.com/science/article/pii/S0167739X18306071
11. Štefanič, P., Cigale, M., Jones, A.C., Knight, L., Taylor, I., Istrate, C., Suciu, G., Ulisses, A., Stankovski, V., Taherizadeh, S., Salado, G.F., Koulouzis, S., Martin, P., Zhao, Z.: Switch workbench: A novel approach for the development and deployment of time-critical microservice-based cloud-native applications. Futur. Gener. Comput. Syst. **99**, 197–212 (2019). https://doi.org/10.1016/j.future.2019.04.008, http://www.sciencedirect.com/science/article/pii/S0167739X1831094X
12. Bellendorf, J., Mann, Z.A.: Specification of cloud topologies and orchestration using tosca: a survey. Computing, 1–23 (2019)
13. Luzar, A., Stanovnik, S., Cankar, M.: Examination and comparison of tosca orchestration tools. In: Muccini, H., Avgeriou, P., Buhnova, B., Camara, J., Caporuscio, M., Franzago, M., Koziolek, A., Scandurra, P., Trubiani, C., Weyns, D., Zdun, U. (eds.) Software Architecture, pp. 247–259. Springer International Publishing, Cham (2020)
14. Kritikos, K., Skrzypek, P., Zahid, F.: Are cloud platforms ready for multi-cloud?. In: Brogi, A., Zimmermann, W., Kritikos, K. (eds.) Service-Oriented and Cloud Computing, pp. 56–73. Springer International Publishing, Cham (2020)
15. Kovács, J., Kacsuk, P.: Occopus: a multi-cloud orchestrator to deploy and manage complex scientific infrastructures. J. Grid Comput. **16**(1), 19–37 (2018). https://doi.org/10.1007/s10723-017-9421-3
16. Wei, H., Rodriguez, J.S., Garcia, O.N.-T.: Deployment management and topology discovery of microservice applications in the multicloud environment. J. Grid Comput. **19**(1), 1 (2021). https://doi.org/10.1007/s10723-021-09539-1
17. Salomoni, D., Campos, I., Gaido, L., de Lucas, J.M., Solagna, P., Gomes, J., Matyska, L., Fuhrman, P., Hardt, M., Donvito, G., et al.: Indigo-datacloud: A platform to facilitate seamless access to e-infrastructures. J. Grid Comput. **16**(3), 381–408 (2018)
18. Di Modica, G., Tomarchio, O., Wei, H., Rodriguez, J.S.: Policy-based deployment in a hybrid and multicloud environment. In: CLOSER, pp. 388–395 (2019)
19. Buzachis, A., Galletta, A., Celesti, A., Carnevale, L., Villari, M.: Towards osmotic computing: a blue-green strategy

for the fast re-deployment of microservices. In: 2019 IEEE Symposium on Computers and Communications (ISCC), pp. 1–6 (2019)
20. Pahl, C., Helmer, S., Miori, L., Sanin, J., Lee, B.: A container-based edge cloud paas architecture based on raspberry pi clusters. In: 2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW), pp. 117–124 (2016)
21. Kim, D., Muhammad, H., Kim, E., Helal, S., Lee, C.: Tosca-based and federation-aware cloud orchestration for kubernetes container platform. Appl. Sci. **9**, 191 (2019). https://doi.org/10.3390/app9010191
22. Brogi, A., Rinaldi, L., Soldani, J.: TosKer: A synergy between TOSCA and Docker for orchestrating multi-component applications. Softw.-Pract. Exper., 2061–2079. https://doi.org/10.1002/spe.2625 (2018)
23. Lipton, P., Lauwers, C., Rutkowski, M., Lauwers, C., Noshpitz, C., Curescu, C.: Tosca simple profile in yaml version 1.3. OASIS Committ. Specif. **1** (2020)
24. Morris, K.: Infrastructure as code: managing servers in the cloud. O'Reilly Media, Inc. (2016)
25. Binz, T., Breiter, G., Leyman, F., Spatzier, T.: Portable cloud services using tosca. IEEE Internet Comput. **16**(3), 80–85 (2012). https://doi.org/10.1109/MIC.2012.43
26. Lipton, P., Palma, D., Rutkowski, M., Tamburri, D.A.: Tosca solves big problems in the cloud and beyond! IEEE Cloud Comput., 1–1. https://doi.org/10.1109/MCC.2018.111121612 (2018)
27. Rahman, A., Mahdavi-Hezaveh, R., Williams, L.: A systematic mapping study of infrastructure as code research. Inf. Softw. Technol. **108**, 65–77 (2019)
28. Hummer, W., Rosenberg, F., Oliveira, F., Eilam, T.: Testing idempotence for infrastructure as code. In: Eyers, D., Schwan, K. (eds.) Middleware 2013, pp. 368–388. Springer, Berlin (2013)
29. Caballer, M., Zala, S., García, A.L., Moltó, G., Fernández, P.O., Velten, M.: Orchestrating complex application architectures in heterogeneous clouds. J. Grid Comput. **16**(1), 3–18 (2018). https://doi.org/10.1007/s10723-017-9418-y
30. Ferry, N., Almeida, M., Solberg, A.: The modaclouds model-driven development. In: Di Nitto, E., Matthews, P., Petcu, D., Solberg, A. (eds.) Model-Driven Development and Operation of Multi-Cloud Applications: The MODAClouds Approach, pp. 23–33. Springer International Publishing, Cham (2017). https://doi.org/10.1007/978-3-319-46031-4_3
31. Brogi, A., Ibrahim, A., Soldani, J., Carrasco, J., Cubo, J., Pimentel, E., D'Andria, F.: Seaclouds: A european project on seamless management of multi-cloud applications. SIGSOFT Softw. Eng. Notes **39**(1), 1–4 (February 2014). https://doi.org/10.1145/2557833.2557844
32. Kiss, T., Kacsuk, P., Kovacs, J., Rakoczi, B., Hajnal, A., Farkas, A., Gesmier, G., Terstyanszky, G.: Micado-microservice-based cloud application-level dynamic orchestrator. Futur. Gener. Comput. Syst. **94**, 937–946 (2019). https://doi.org/10.1016/j.future.2017.09.050, http://www.sciencedirect.com/science/article/pii/S0167739X17310506
33. Kovács, J.: Supporting programmable autoscaling rules for containers and virtual machines on clouds. J. Grid Comput. **17**(4), 813–829 (2019). https://doi.org/10.1007/s10723-019-09488-w
34. Caballer, M., Antonacci, M., Šustr, Z., Perniola, M., Moltó, G.: Deployment of elastic virtual hybrid clusters across cloud sites. J. Grid Comput. **19**(1), 4 (2021). https://doi.org/10.1007/s10723-021-09543-5
35. Tomarchio, O., Calcaterra, D., Di Modica, G., Mazzaglia, P.: Torch: a tosca-based orchestrator of multi-cloud containerised applications. J. Grid Comput. **19**(1), 5 (2021). https://doi.org/10.1007/s10723-021-09549-z
36. Villari, M., Celesti, A., Tricomi, G., Galletta, A., Fazio, M.: Deployment orchestration of microservices with geographical constraints for edge computing. In: 2017 IEEE Symposium on Computers and Communications (ISCC), pp. 633–638 (2017)
37. Képes, K., Breitenbücher, U., Leymann, F., Saatkamp, K., Weder, B.: Deployment of distributed applications across public and private networks. In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC), pp. 236–242 (2019)
38. Blair, G., Bencomo, N., France, R.B.: Models@ run. time. Computer **42**(10), 22–27 (2009)
39. Di Nitto, E., Gorroñogoitia, J., Kumara, I., Meditskos, G., Radolović, D., Sivalingam, K., González, R.S.: An approach to support automated deployment of applications on heterogeneous cloud-hpc infrastructures. In: 2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 133–140 (2020)
40. Kumara, I., Quattrocchi, G., Tamburri, D., Van Den Heuvel, W.-J.: Quality assurance of heterogeneous applications: The sodalite approach. In: Zirpins, C., Paraskakis, I., Andrikopoulos, V., Kratzke, N., Pahl, C., El Ioini, N., Andreou, A.S., Feuerlicht, G., Lamersdorf, W., Ortiz, G., Van den Heuvel, W.-J., Soldani, J., Villari, M., Casale, G., Plebani, P. (eds.) Advances in Service-Oriented and Cloud Computing, pp. 173–178. Springer International Publishing, Cham (2021)
41. Kumara, I., Vasileiou, Z., Meditskos, G., Tamburri, D.A., Van Den Heuvel, W.-J., Karakostas, A., Vrochidis, S., Kompatsiaris, I.: Towards semantic detection of smells in cloud infrastructure code. In: Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics, WIMS 2020, pp. 63–67. Association for Computing Machinery (2020)
42. Borovits, N., Kumara, I., Krishnan, P., Palma, S.D., Di Nucci, D., Palomba, F., Tamburri, D.A., van den Heuvel, W.-J.: Deepiac: Deep learning-based linguistic anti-pattern detection in iac. In: Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation, MaLTeSQuE 2020, pp. 7–12. Association for Computing Machinery (2020)
43. Mujkanovic, N., Sivalingam, K., Lazzaro, A.: Optimising ai training deployments using graph compilers and containers. In: 2020 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–8 (2020)
44. Baresi, L., Leva, A., Quattrocchi, G.: Fine-grained dynamic resource allocation for big-data applications. IEEE Trans. Softw. Eng., 1–1. https://doi.org/10.1109/TSE.2019.2931537 (2019)