

Automated Generation of Application Management Workflows using TOSCA Policies

Domenico Calcaterra and Orazio Tomarchio^a

Department of Electrical Electronic and Computer Engineering, University of Catania, Catania, Italy

Keywords: Automated Management, Cloud Application Management, Policies, TOSCA, BPMN.

Abstract: Cloud orchestration frameworks are recognised as a key driver to tackle the complexity of deploying and managing cloud resources and services. Although automated orchestration approaches have been widely adopted in industry and academia, most of them only provide limited management capabilities, such as monitoring or scaling individual components. Holistic management features concerning multiple application components are mostly not covered by existing solutions. Supporting such features typically involves a custom implementation of management logic, which can be error-prone, time-consuming and frequently outdated. In this work, we present an approach for the automated generation of holistic management workflows based on TOSCA application models, reusable node types and policy types. A prototype implementation and a case study are also discussed to prove the feasibility of the proposed idea.

1 INTRODUCTION

As a result of several benefits in terms of availability, scalability, costs and time savings (Marston et al., 2011), cloud computing adoption has expanded rapidly over the past decade, leading to a larger distribution of resources and services across the internet (Dikaiakos et al., 2009). COVID-19 has further accelerated the migration to cloud computing, causing cloud adoption to increase at an even greater rate. According to Flexera 2021 State of the Cloud Report (Flexera, 2021), many organisations have successfully adopted cloud computing worldwide.


A key point to optimise resource usage and fully exploit the potential of cloud computing is cloud orchestration (Ranjan et al., 2015; Weerasiri et al., 2017). Due to the proliferation of heterogeneous providers, cloud orchestration is considered to be challenging, error-prone and time-consuming (Caballer et al., 2018). Manual solutions to orchestration are discouraged because they require a familiarity with different cloud provider platforms, services and low-level APIs. Most commercial cloud providers offer automated orchestration platforms to end-users (Bousselmi et al., 2014): however, these products are proprietary and not portable. Cloud-agnostic approaches to automation are also provided by tools

falling under the realm of DevOps (Leite et al., 2019), Infrastructure as Code (IaC) (Morris, 2016) and Configuration Management (CM) (Delaet et al., 2010).

By contrast, these approaches only have limited automated management, such as monitoring or scaling individual components (Toffetti et al., 2017), whereas holistic application management involving components distributed over multiple environments is not automatically supported for the most part. Holistic application management includes features such as performing backups of stateful components, testing the availability of application components, etc. However, the support for such features requires custom implementations which can be error-prone, time-consuming and frequently outdated.

In this paper we propose an approach to automatically generate holistic management workflows from TOSCA application models, which are transparently enriched with component-specific management features by means of an appropriate combination of reusable TOSCA node types and policy types. As a consequence of such enrichment process, application architects are not required to provide manual implementation of management functionalities, leading up to reduced efforts and more error resilience.

The remainder of the paper is structured as follows. Section 2 provides a background of the concepts used in this work and motivates the need for holistic management automation. Section 3 delves

^a  <https://orcid.org/0000-0003-4653-0480>

into the proposed approach to automatically generate management workflows from TOSCA-based application models. Section 4 discusses a prototype implementation and a case-study showing the potential of the proposed idea. Related work is discussed in Section 5. Concluding remarks and future directions are debated in Section 6.

2 BACKGROUND AND MOTIVATION

In this section, we first outline basic principles for deployment and management automation and then provide a brief overview on the TOSCA specification. Finally, a motivation scenario for holistic management automation is introduced.

2.1 Deployment and Management Automation

Over the past few years, quite a few orchestration and management frameworks have emerged from both industry and academia (Tomarchio et al., 2020). Most cloud industry players have developed Cloud Management Platforms (CMP) to automate cloud service provisioning. The most advanced platforms also offer lifecycle management of cloud applications. These commercial products are typically neither open to the community nor portable across third-party providers.

There are a number of tool categories which share similarities with cloud orchestrators. Configuration management tools, such as Ansible, Chef, Puppet and Salt, mostly automate the development, delivery, testing and maintenance throughout the software lifecycle. These tools have recently moved towards orchestration, which leverage Infrastructure as Code (IaC) to change, configure and automate infrastructure. Terraform is one of the most notable IaC open-source solutions. These technologies use either *declarative* or *imperative* models to automate application deployment. While declarative models describe the desired application state from which the deployment tasks are automatically derived, imperative models specify the deployment tasks and their execution order.

Although declarative models are highly intuitive and reusable, they suffer from a few limitations. On the one hand, deployment systems can directly infer all tasks to be executed from the models, but on the other hand they can neither customise tasks nor alter their execution order. Imperative models are then necessary when it comes to modelling complex application deployments with customised tasks. Work-

flows languages, such as BPMN (OMG, 2011) and BPEL (OASIS, 2007), are typical examples of imperative technologies. However, imperative models require technical expertise and are frequently outdated as compared to declarative models.

While the available technologies support automated deployments over multiple environments, they only provide limited support for automated management (Toffetti et al., 2017). As far as cloud providers are concerned, they usually only offer management features for the hosted components, with the result that single management features need to be orchestrated when multiple providers are involved. Holistic management of multiple components located in different environments is mostly unsupported as well. Typical holistic management features include but are not limited to component backup, testing, update, etc. Even in that case, automating holistic management requires single management features to be orchestrated (by a workflow, for instance).

2.2 The TOSCA Specification

High-level specification languages to describe the topology of cloud services facilitate the orchestration process and foster interoperability across different providers. TOSCA (OASIS, 2013) represents a notable contribution to the development of cloud standards, since it allows to describe multi-tier applications and their lifecycle management in a modular and portable fashion (Bellendorf and Mann, 2018).

According to TOSCA, the structure of a cloud application is described as a *service template*, which consists of a topology template and the types needed to build such a template. The topology template is a typed directed graph, whose nodes (called *node templates*) model the application components, and edges (called *relationship templates*) model the relations occurring among such components. Each topology node can also be associated with the corresponding *capabilities* and *requirements*, the *interfaces* to manage it, the *attributes* and *properties* it features, the software *artifacts* it uses and the *policies* applied to it.

TOSCA supports the deployment and management of applications in two different flavours: *imperative processing* and *declarative processing*. The imperative processing requires that management logic is contained in the Cloud Service Archive (CSAR), which stores all software artifacts needed to provision and manage the application. Management plans imperatively orchestrate low-level management operations that are provided either by the application components themselves or by publicly accessible services. Management plans are typically implemented

using workflow languages (e.g., BPMN, BPEL). The declarative processing shifts management logic from plans to runtime. TOSCA runtimes automatically infer the corresponding logic by interpreting the application topology template. Management functionalities depend on the corresponding runtime, which is not standardised by the TOSCA specification.

TOSCA Simple Profile (OASIS, 2020) is an isomorphic rendering of a subset of the TOSCA specification in the YAML language. It provides a more concise and accessible syntax to describe portable cloud applications. TOSCA Simple Profile defines a few normative workflows that are used to operate a topology and specifies how they are declaratively generated. Imperative workflows can be used for complex use cases that cannot be solved in declarative workflows. However, they provide less reusability as they are defined for a specific topology rather than being dynamically generated based on the topology content. The work described in this paper heavily grounds on the TOSCA Simple Profile.

2.3 Motivation

As mentioned in Section 2.1, while the available approaches provide automated deployment over multiple environments, they only support limited automated management which requires orchestration when multiple cloud providers, services and components are involved. Besides, automated holistic management is mostly unsupported and, when it is supported, custom implementations are often required.

Figure 1 illustrates a declarative application model describing a typical cloud application scenario based on TOSCA. In order to have a proper application deployment, it is crucial to check if it is successful from both a technical and business perspective. This is where testing becomes relevant. In case of multi-cloud applications, it might be necessary for application components to communicate with each other. For instance, depending on the testing, SSH connections (e.g. *Compute* nodes), HTTP connections (e.g. *WordPress* and *Apache* nodes) or even SQL connections (e.g. *DBMS* and *Database* nodes) might need to be established. Since huge expertise is required to implement testing, the automated generation of tests is important to ensure that all components and communication among them work as expected. In case of a web application, it is also important to copy current data on a regular basis. In order to back up the Database in our scenario, it is required to either use the backup feature of the underlying DBMS or establish a direct connection to the database and execute a query to retrieve all data. However, additional tech-

nology and domain-specific logic need to be orchestrated and executed in the correct order.

In general terms, even automating the management of simple applications can be a major challenge when management features must be implemented manually, since manual implementations require massive domain-specific expertise and are error-prone, time consuming and frequently outdated. In this work we propose an approach for the automatic generation of holistic management workflows based on TOSCA application models and policies.

3 POLICY-BASED APPLICATION MANAGEMENT

In this section, we delve into the proposed methodology to automatically generate management workflows from TOSCA-based application models. First, we provide a bird's-eye view of the approach; then, we analyse the main steps of the entire process.

3.1 High-level Approach

In order to automatically generate executable management workflows from TOSCA application models, we propose the approach depicted in Figure 2.

The overall process is composed of three distinct sequential phases. The process begins with the *Application Specification* phase, where the application architect is in charge of modelling and submitting a TOSCA application model which is specified based on *interface types*, *node types* and *policy types* included in a TOSCA Simple Profile extension. All these types define to varying degrees management interfaces and policies for different management features, using a general-to-specific pattern. In a nutshell, given a general management feature (e.g. testing), general interface types and policy types are defined for such a feature which get to be extended for specific node types (e.g. Database). *Managed node types* are also defined by extending basic TOSCA node types with all available management interfaces.

The submitted input triggers the *Workflow Generation* phase, that in turn consists of the transformation of a TOSCA application model into different workflow plans, namely provisioning and management plans, based on input data. Specifically, the provisioning plan is generated based on node dependencies, whereas management plans are generated based on policies and management interface operations defined on nodes. The TOSCA Processor component is responsible for parsing a TOSCA application model, generating its provisioning and manage-

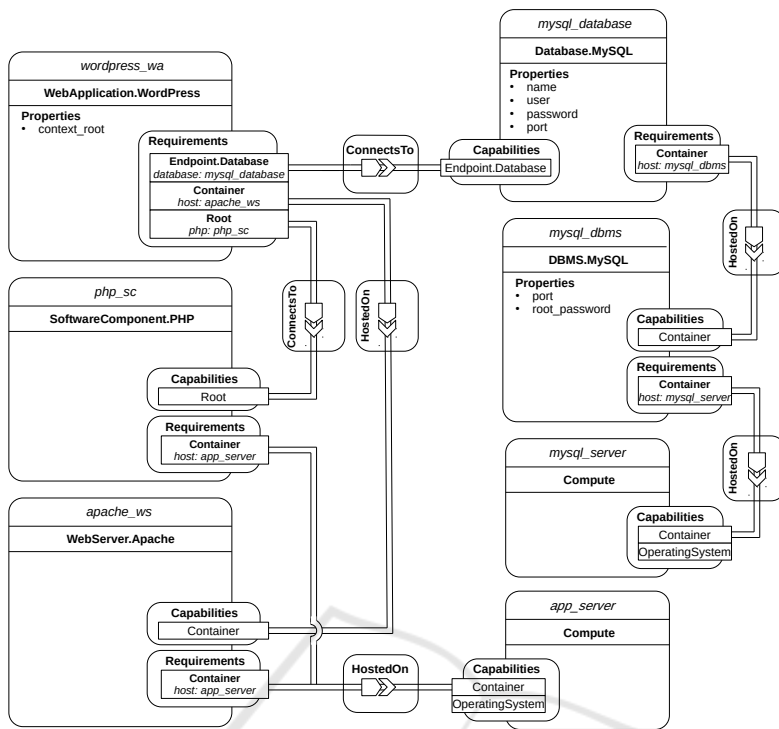


Figure 1: TOSCA application model of a WordPress scenario.

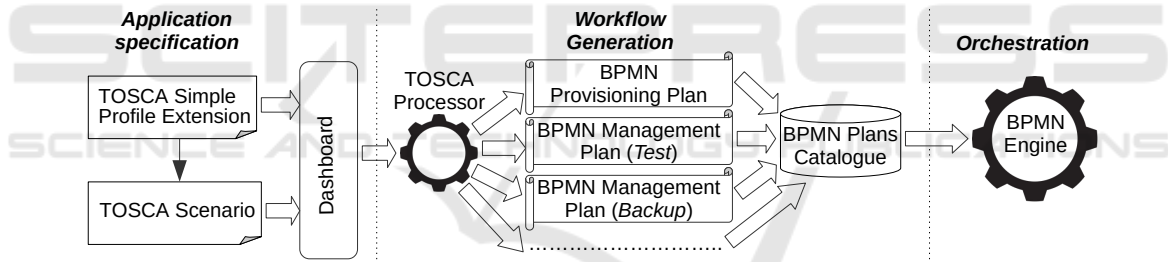


Figure 2: Overview of the approach for the automated generation of application management workflows.

ment plans and validating them. Finally, in the *Orchestration* phase, a workflow engine performs the enactment of such plans.

3.2 Application Specification

One of the main strengths of this work is the development of a standards-based approach for the description of application topology and management, which leverages the TOSCA standard.

Although the TOSCA Simple Profile specification includes two normative interface types, i.e. *Standard* and *Configure*, for component lifecycle and configuration management respectively, there is a lack of support for management features. As a result, we extended the standard specification by defining management-oriented interface types. By way of illustration, Figure 3 shows a few exemplifying inter-

face type definitions included in our TOSCA Simple Profile extension. As mentioned in Section 3.1, given a specific management feature, interface types are defined via a general-to-specific pattern. In fact, as we can see in Figure 3a, two interface types are present: a general interface type (*Test*), which is defined for testing purposes, and a specific interface type (*TestDB*), which is an extension of the former for testing database components. Similarly, Figure 3b shows two interface type definitions: a general interface type for backup purposes and a specific interface type for backing up database components.

All these additional interface types must be included in node type definitions, which is why we also extended the normative node type hierarchy in order to support management features. For the sake of clarity, Figure 4 depicts a node type definition and a few policy type definitions included in our TOSCA Sim-

```

tosca.interfaces.node.Test:
  version: 1
  description: >
    TOSCA management interface type for testing
    generic components

tosca.interfaces.node.TestDB:
  derived_from: tosca.interfaces.node.Test
  version: 1
  description: >
    TOSCA management interface type for testing
    database components
  operations:
    test_db_connection:
      description: |
        Management operation to test the DB
        connection
  notifications:
    db_connection_testing:
      description: |
        Notification to trigger connection testing

```

```

tosca.interfaces.node.State:
  version: 1
  description: >
    TOSCA management interface type for saving/
    restoring the state of generic components

tosca.interfaces.node.StateDB:
  derived_from: tosca.interfaces.node.State
  version: 1
  description: >
    TOSCA management interface type for saving/
    restoring the state of database components
  operations:
    db_freeze:
      description: |
        Management operation to freeze DB state
    db_thaw:
      description: |
        Management operation to thaw DB state
  notifications:
    db_state_freezing:
      description: |
        Notification to trigger DB state freezing
    db_state_thawing:
      description: |
        Notification to trigger DB state thawing

```

(a) *Test* and *TestDB* interface types(b) *State* and *StateDB* interface types

Figure 3: Illustrative interface type definitions for Testing and Backup features.

ple Profile extension. In particular, as we can see in Figure 4a, the *Database.Managed* node type extends the normative *Database* node type by adding *TestDB* and *StateDB* interfaces (see Figure 3) for testing and backup purposes. Policy types are also defined in order to specify the targeted node types and the actions to perform in relation to management features, when specific events are triggered. In particular, policy triggers are linked to notification events from the targeted node types' management interfaces. Similar to interface types, policy types are defined via a general-to-specific pattern as well. As we can see in Figure 4a, two general policy types are present: *Testable* and *Freezable* for testing and backup purposes, respectively. By contrast, Figure 4b shows two specific policy types: *TestableDB* and *FreezableDB* for testing and backing up database components, respectively.

In summary, while normative interface types and node types provide sufficient deployment capabilities, the current version of the TOSCA standard is not well equipped with management features. In order to fill this gap, we propose to define: *management interface types*, which get to be extended depending on a combination of management features and node type categories, *managed node types*, which extend normative node types and include these management interfaces, and *management policy types*, which define the oper-

ations to execute on the targeted node types based on management features. The complete set of extended types is fully TOSCA-compliant, since it is valid according to the grammar and rules defined in the standard, and can be further enriched with additional management features.

3.3 Management Workflow Generation

As mentioned in Section 3.1, the application architect models a TOSCA application model according to interface types, node types and policy types included in a TOSCA Simple Profile extension. By applying this approach, the application model is automatically enriched with management capabilities, which the TOSCA Processor (see Figure 2) leverages to generate management workflows based on policies and interface operations defined on nodes. Policies are the focal point of the entire process, as they specify the targeted nodes and the actions triggered depending on management features.

The strategy of generation of management workflows depends on management features. In case of the *backup* feature a parallel strategy is adopted, since every stateful component can be backed up independently; in case of the *testing* feature two strategies are viable: a) parallel strategy and b) sequential strat-

```

tosca.nodes.Database.Managed:
  derived_from: toska.nodes.Database
  description: >
    Extension of the toska.nodes.Database node type with
    specific management interfaces
  interfaces:
    TestDB:
      type: toska.interfaces.node.TestDB
    operations:
      test_db_connection:
        description: |
          Management operation to test DB connection
    notifications:
      db_connection_testing:
        description: |
          Notification to trigger connection testing
    StateDB:
      type: toska.interfaces.node.StateDB
    operations:
      db_freeze:
        description: |
          Management operation to freeze DB state
      db_thaw:
        description: |
          Management operation to defrost DB state
    notifications:
      db_state_freezing:
        description: |
          Notification to trigger DB state freezing
      db_state_thawing:
        description: |
          Notification to trigger DB state thawing

tosca.policies.Testable:
  derived_from: toska.policies.Root
  description: >
    The TOSCA Policy Type for testing TOSCA nodes or groups
    of nodes.

tosca.policies.Freezable:
  derived_from: toska.policies.Root
  description: >
    The TOSCA Policy Type for freezing/thawing TOSCA nodes
    or groups of nodes.
  
```

```

tosca.policies.TestableDB:
  derived_from: toska.policies.Testable
  description: >
    The TOSCA Policy Type for testing TOSCA database nodes.
  targets:
    - toska.nodes.Database.Managed
  triggers:
    db_connection_testing_trigger:
      description: |
        The trigger kicks in when connection testing gets
        notified
      event: db_connection_testing
      action:
        - call_operation: toska.interfaces.node.TestDB.
          test_db_connection

tosca.policies.FreezableDB:
  derived_from: toska.policies.Freezable
  description: >
    The TOSCA Policy Type for freezing/thawing TOSCA
    database nodes.
  targets:
    - toska.nodes.Database.Managed
  triggers:
    db_state_freezing_trigger:
      description: |
        The trigger kicks in when DB state freezing gets
        notified
      event: db_state_freezing
      action:
        - call_operation: toska.interfaces.node.StateDB.
          db_freeze
    db_state_thawing_trigger:
      description: |
        The trigger kicks in when DB state thawing gets
        notified
      event: db_state_thawing
      action:
        - call_operation: toska.interfaces.node.StateDB.
          db_thaw
  
```

(a) Database node type, Testable and Freezable policy types

(b) TestableDB and FreezableDB policy types

Figure 4: Illustrative node type and policy type definitions for Testing and Backup features.

egy. Components can be either tested in isolation with the parallel strategy or tested consecutively following node dependencies with the sequential strategy.

The parallel generation of Backup workflows works as shown in Figure 5. In general, considering more than one Freezable policies (see Figure 4a), a new workflow is generated where more than one paths can trigger the workflow instantiation by means of an event-based gateway. Each path consists of a message intermediate event, which is linked to a notification event from the targeted nodes' State interface (see Figure 3b), and a parallel multi-instance subprocess, whose number of instances depends on the number of target nodes the policy is applied to. A service task in the subprocess executes the action of the policy trigger, which enacts the actual backup.

Figure 6 shows how Testing workflows are generated. Firstly, a workflow is generated for each node type. Secondly, a distinction is made between nodes without dependencies and nodes with dependencies. The former are tested in isolation, while the latter can be either tested in isolation or tested sequentially ac-

ording to node dependencies. In the first case (see Figure 6a), considering more than one testing operations, a new workflow is generated where more than one paths can trigger the workflow instantiation by means of an event-based gateway. Each path consists of a message intermediate event, which is linked to a notification event from the targeted nodes' Test interface (see Figure 3a), and a parallel multi-instance subprocess, whose number of instances depends on the number of target nodes the policy is applied to. A service task in the subprocess executes the action of the policy trigger, which enacts the actual testing. Finally, a message end event marks the end of the workflow. In the second case (see Figure 6b), a new workflow is generated where either a message start event, coming from the targeted nodes' Test interface notifications, or a parallel multi-instance receive task, waiting for notifications from node requirements under test, triggers the workflow instantiation. A parallel multi-instance subprocess is then activated depending on the number of target nodes the policy is applied to. A service task in the subprocess executes the action of the

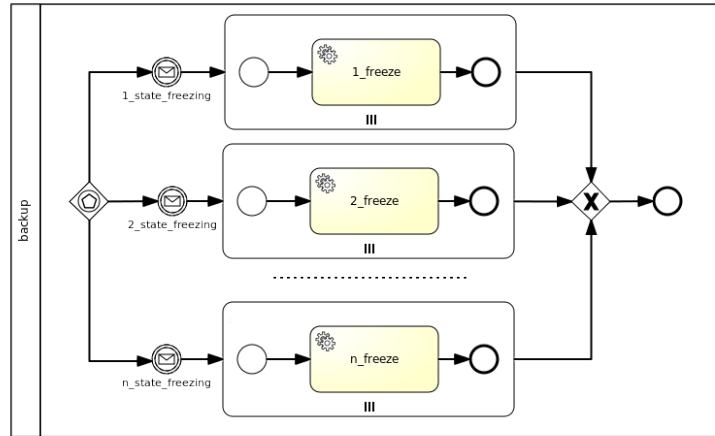


Figure 5: Parallel strategy for Backup workflows.

policy trigger enacting the actual testing. Finally, a message end event marks the end of the workflow.

4 PROTOTYPE VALIDATION

In this section, we first discuss the implementation of a prototype and then corroborate the working behaviour of our approach by applying both *testing* and *backup* management features on our motivation scenario (see Figure 1).

4.1 System Overview

To validate our approach, we implemented a prototype based on the open-source TORCH (Tomarchio et al., 2021). TORCH is a TOSCA-based framework which deploys and orchestrates VM-based and container-based applications on top of different cloud providers. In order to do so, TORCH transforms a TOSCA application model into an equivalent BPMN workflow and dataflow model, which a BPMN engine leverages to enforce the operations specified in the model.

Figure 7 shows the multi-layered framework architecture of TORCH. The *Application Specification Layer* consists of two components: the Dashboard, which is the front-end component, and the TOSCA Modeller, which guides the user to sketch application requirements. The *Orchestration Layer* comprises the TOSCA Processor component, which is in charge of validating, parsing and converting TOSCA application templates into BPMN plans, and the BPMN Engine component, which is responsible for instantiating and orchestrating such BPMN plans. The *Service Binding Layer* manages the orchestration of application resources and services.

The proposed approach for the automated generation of management workflows (see Section 3) is based on the extension of the *TOSCA Processor* (see Figure 7), which consists of three components: *TOSCA-Parser*, *BPMN-Generator* and *BPMN-Validator*. The *TOSCA-Parser* provides means to load, parse and validate a TOSCA service template and create the corresponding dependency graph. The *BPMN-Generator* creates a BPMN plan depending on a parsed service template and its dependency graph. The *BPMN-Validator* validates an automatically generated plan against the BPMN specification. Further details about these components can be found in (Calcaterra et al., 2017; Calcaterra et al., 2018).

As mentioned above, the TOSCA Processor was extended in order to support the automatic generation of holistic management workflows. Specifically, several novelties were introduced in the *TOSCA-Parser* and the *BPMN-Generator*. As regards the *TOSCA-Parser*, it was extended in order to provide an ad-hoc parsing of node properties and artifacts, custom interfaces, interface operations and notifications, policy triggers. As for the *BPMN-Generator*, since it was originally conceived to generate provisioning plans based on the TOSCA dependency graph, it was extended in order to provide support for the generation of management plans based on TOSCA policies. In particular, while the original *BPMN-Generator* was only capable of generating workflows depending on node dependencies, the extended *BPMN-Generator* is capable of generating workflows even when there are no explicit node dependencies. For instance, it can support both sequential and parallel strategies for workflow generation.

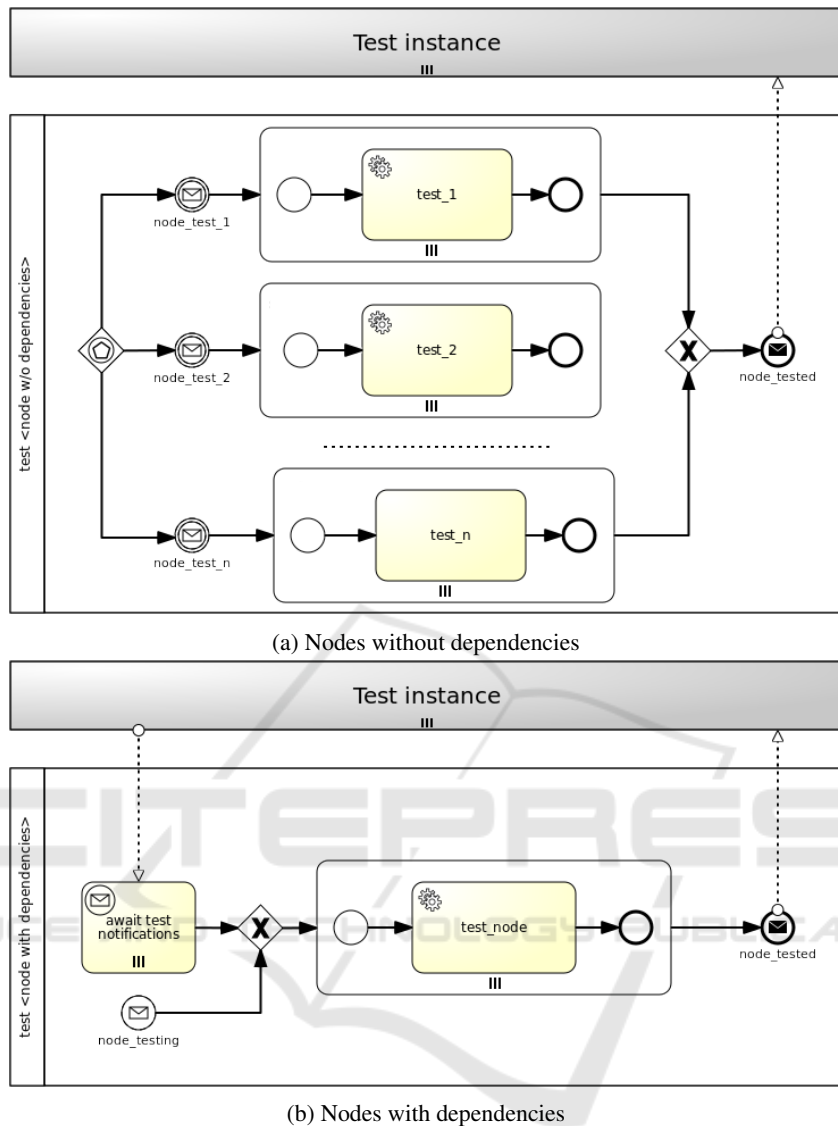


Figure 6: Parallel and sequential strategies for Testing workflows.

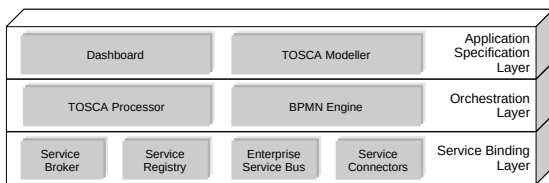


Figure 7: TORCH - Framework architecture.

4.2 Case Study

Based on the motivation scenario introduced in Section 2.3, we discuss two different management features, namely *Testing* and *Backup*. By applying our approach, the motivating scenario is automatically enriched with test and backup operations by means of

feature-specific node types and policy types. The added operations are shown in Figure 8. With reference to the backup feature, we can see that the *mysql_database* node includes the *StateDB* interface. With regard to the testing feature, every node contains a specific interface for testing purposes. By way of illustration, we can observe that the *mysql_server* and *mysql_dbms* nodes include the *TestCompute* and *TestDBMS* interfaces, respectively.

Based on the approach for the management workflow generation in Section 3.3, Figure 9 shows the Backup workflow for the *mysql_database* node in the motivation scenario. Given a *FreezableDB* policy targeting a *Database.Managed* node (see Figure 4b), a new workflow is generated where the

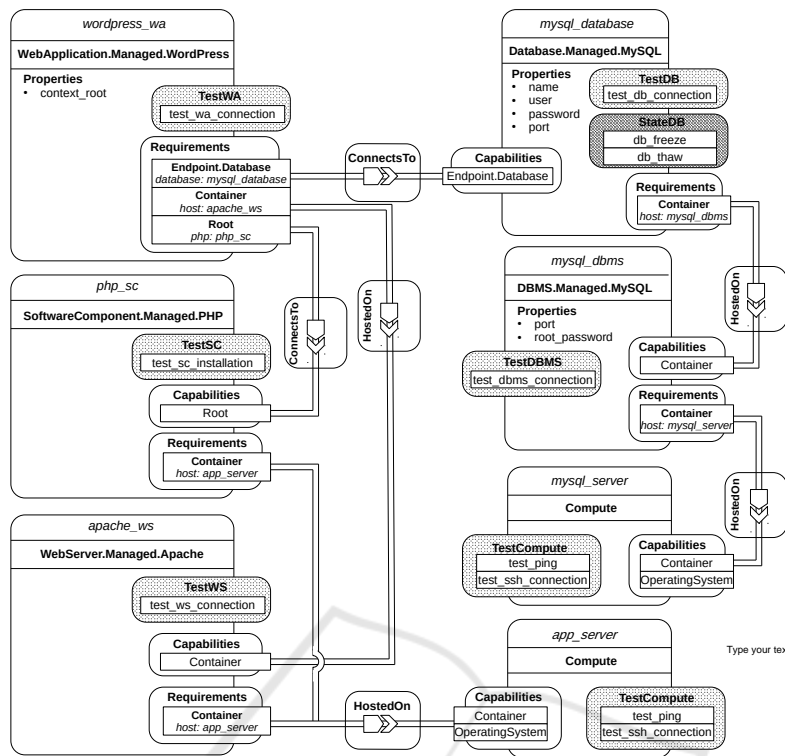


Figure 8: Management-oriented TOSCA application model of a WordPress scenario.

db_state_freezing message start event, coming from the StateDB interface notifications, triggers the workflow instantiation. A parallel multi-instance subprocess is then activated depending on the number of target nodes the policy is applied to. In this case there is only one target node (i.e. *mysql_database*). The *db_freeze* service task in the subprocess executes the action of the policy trigger, which enacts the actual database backup.

Figure 10 shows exemplifying Testing workflows for the *mysql_server* and *app_server* nodes (Figure 10a) and the *mysql_dbms* node (Figure 10b) in the motivation scenario. In contrast with the strategy of generation of Backup workflows, the strategy of generation of Testing workflows can take node dependencies into account. In particular, a distinction is made between nodes having no dependencies, such as *Compute* nodes, and nodes having dependencies, such as *DBMS* nodes.

In the first case (see Figure 10a), given a *Testable-Compute* policy targeting a *Compute* node, two paths are generated where either the *compute_test_ping* message intermediate event or the *compute_test_ssh* message intermediate event, coming from the TestCompute interface notifications, triggers the workflow instantiation by means of an event-based gateway. Regardless of the path being activated, a paral-

lel multi-instance subprocess is then instantiated depending on the number of target nodes the policy is applied to. In this case there are two target nodes (i.e. *mysql_server* and *app_server*). Either the *test_ping* service task or the *test_ssh_connection* service task executes the action of the policy trigger, which enacts the actual *Compute* testing. Finally, the *compute_tested* message end event marks the end of the workflow.

In the second case (see Figure 10b), given a *TestableDBMS* policy targeting a *DBMS* node, a new workflow is generated where either the *dbms_connection_testing* message start event, coming from the TestDBMS interface notifications, or the *await_test_notifications* parallel multi-instance receive task, waiting for notifications from node requirements under test (i.e. *mysql_server*), triggers the workflow instantiation. A parallel multi-instance subprocess is then activated depending on the number of target nodes the policy is applied to. In this case there is only one target node (i.e. *mysql_dbms*). The *test_dbms_connection* service task in the subprocess executes the action of the policy trigger, which enacts the actual *DBMS* testing. Finally, the *dbms_tested* message end event marks the end of the workflow.

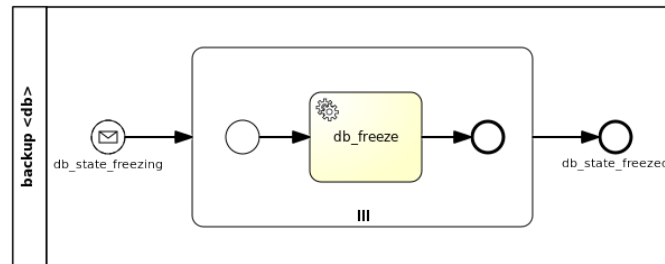


Figure 9: Database Backup workflow for a WordPress scenario.

5 RELATED WORK

The topics of policy-based automation and application management have gained popularity in business-oriented and research projects (Tomarchio et al., 2020). This section provides a brief overview on a few works that exploited policies, TOSCA or similar approaches for application management.

In (Alexander et al., 2017) the authors presented TOSCamp, an end-to-end cloud orchestration solution based on TOSCA and CAMP, which introduces extensions to CAMP to allow for multi-cloud application orchestration through declarative policies. Even though this work leveraged policies for application orchestration, no holistic management was considered. In (Pierantoni et al., 2020) the authors introduced MiCADO, a cloud orchestration framework to deploy and manage TOSCA-based applications in the cloud. An extensible set of TOSCA policies was also elaborated to manage deployment, performance, scalability and security requirements of applications.

In (Caballer et al., 2018) the authors presented INDIGO-DataCloud, a cloud orchestration platform to orchestrate TOSCA-based applications with complex topologies and operational requirements (e.g. auto-scaling resources) across heterogeneous cloud infrastructures. In (Kumara et al., 2021) the authors proposed the SODALITE platform to support the deployment, execution, monitoring and policy-based runtime adaptation of TOSCA-based applications on heterogeneous cloud-edge infrastructures. In (Cankar et al., 2020) the authors presented xOpera, an orchestrator capable of enacting application deployment and addressing autoscaling in a policy-based fashion by using TOSCA standard templates. All the aforementioned works dealt with application deployment and policy-based management. Nevertheless, no support for holistic management was provided.

In (Wurster et al., 2018) the authors presented a modelling concept to annotate a deployment model with deployment tests. Tests are automatically run af-

ter deployment to verify that the application is working properly. Limitations of the proposal include the fact that the deployment system must provide a basic set of plugins and be extensible with custom plugins from arbitrary sources. In addition, the approach features no other holistic management except testing. In (Harzenetter et al., 2020) the authors introduced an approach to terminate an application in its current state and restart it in the same state again. Freeze plans and defrost plans are automatically generated in order to save and restore the state of stateful components. Limitations of the proposal include the assumption that a stateful component type must be annotated to indicate that it will hold a state between requests. Besides, the approach features no other holistic management except state backup and recovery. In (Harzenetter et al., 2019) the authors proposed a concept to automatically generate executable management workflows based on the application deployment model. The modelled components are enriched with component-specific management operations and a management workflow gets generated to orchestrate such operations. However, the approach suffers from some limitations. To provide new management features, on the one hand, the workflow generation must be extended with a corresponding plugin and, on the other one, domain experts have to implement management operations in new component types.

6 CONCLUSION

The ever-growing interest in cloud computing has fostered a business landscape, where application provisioning and management have become a key factor for the competitiveness of cloud providers. Although a number of automated orchestration frameworks have appeared, holistic application management regarding diverse components distributed across multiple heterogeneous environments is still mostly unsupported.

In this paper, we presented an approach for the

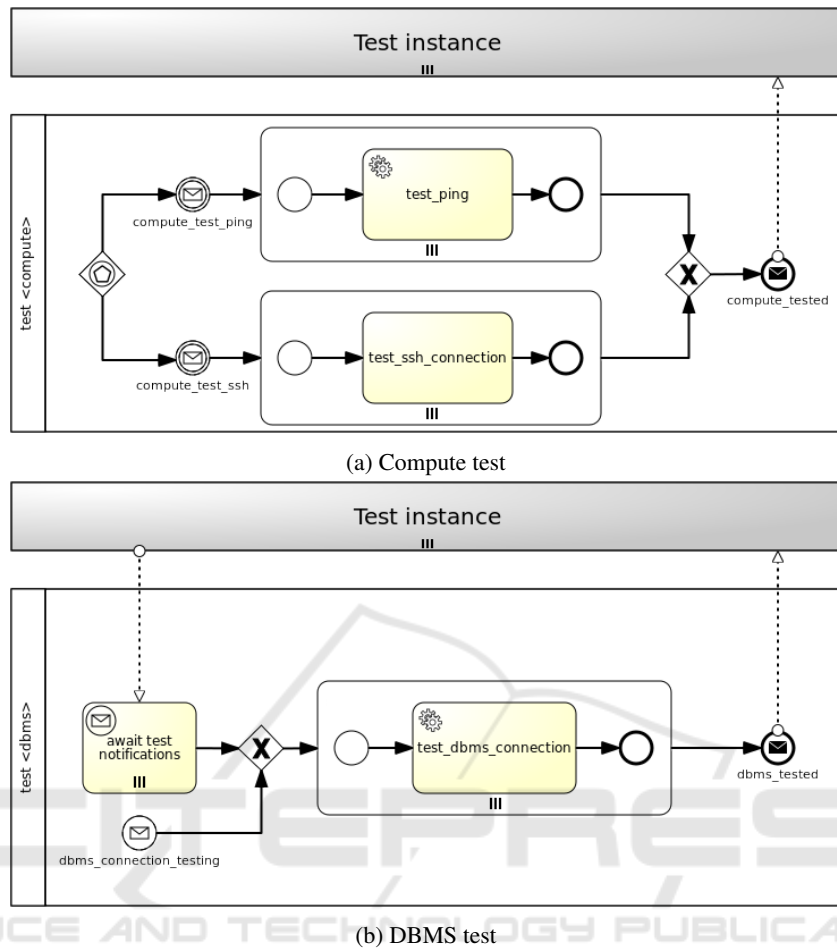


Figure 10: Illustrative Testing workflows for a WordPress scenario.

automated generation of holistic management workflows by enriching TOSCA application models with reusable TOSCA node types and policy types. The main effort was devoted to the application model description based on a TOSCA extension and the following management workflow generation. A prototype implementation and a case study were also discussed to showcase the viability of the proposal.

Future work will focus on two research directions. On the one hand, more and more management operations will be under examination to be included in our TOSCA extension. On the other hand, supplementary strategies besides sequential and parallel workflow generation will be investigated.

REFERENCES

- Alexander, K., Lee, C., Kim, E., and Helal, S. (2017). Enabling end-to-end orchestration of multi-cloud applications. *IEEE Access*, 5:18862–18875.
- Bellendorf, J. and Mann, Z. Á. (2018). Cloud Topology and Orchestration Using TOSCA: A Systematic Literature Review. In Kritikos, K., Plebani, P., and de Paoli, F., editors, *Service-Oriented and Cloud Computing*, pages 207–215. Springer International Publishing.
- Bousselmi, K., Brahmi, Z., and Gammoudi, M. M. (2014). Cloud services orchestration: A comparative study of existing approaches. In *IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, (WAINA 2014)*, pages 410–416.
- Caballer, M., Zala, S., García, Á. L., Moltó, G., Fernández, P. O., and Velten, M. (2018). Orchestrating Complex Application Architectures in Heterogeneous Clouds. *Journal of Grid Computing*, 16(1):3–18.
- Calcaterra, D., Cartelli, V., Di Modica, G., and Tomarchio, O. (2017). Combining TOSCA and BPMN to Enable Automated Cloud Service Provisioning. In *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER 2017)*, pages 159–168, Porto (Portugal).
- Calcaterra, D., Cartelli, V., Di Modica, G., and Tomarchio, O. (2018). A framework for the orchestration and provision of cloud services based on toasca and bpmn.

- In Ferguson, D., Muñoz, V. M., Cardoso, J., Helfert, M., and Pahl, C., editors, *Cloud Computing and Service Science*, pages 262–285, Cham. Springer International Publishing.
- Cankar, M., Luzar, A., and Tamburri, D. A. (2020). Auto-scaling using toasca infrastructure as code. In Muccini, H., Avgeriou, P., Buhnova, B., Camara, J., Caporuscio, M., Franzago, M., Koziolok, A., Scandurra, P., Trubiani, C., Weyns, D., and Zdun, U., editors, *Software Architecture*, pages 260–268, Cham. Springer International Publishing.
- Delaet, T., Joosen, W., and Vanbrabant, B. (2010). A survey of system configuration tools. In *Proceedings of the 24th International Conference on Large Installation System Administration, LISA'10*, page 1–8, USA. USENIX Association.
- Dikaiiakos, M. D., Katsaros, D., Mehra, P., Pallis, G., and Vakali, A. (2009). Cloud Computing: Distributed Internet Computing for IT and Scientific Research. *IEEE Internet Computing*, 13(5):10–13.
- Flexera (2021). Flexera 2021 State of the Cloud Report. <https://info.flexera.com/CM-REPORT-State-of-the-Cloud>. Last accessed on 30-11-2021.
- Harzenetter, L., Breitenbücher, U., Képes, K., and Leymann, F. (2020). Freezing and defrosting cloud applications: automated saving and restoring of running applications. *SICS Software-Intensive Cyber-Physical Systems*, 35(1):101–114.
- Harzenetter, L., Breitenbücher, U., Leymann, F., Saatkamp, K., Weder, B., and Wurster, M. (2019). Automated Generation of Management Workflows for Applications Based on Deployment Models. In *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 216–225. IEEE.
- Kumara, I., Mundt, P., Tokmakov, K., Radolović, D., Maslennikov, A., González, R. S., Fabeiro, J. F., Quattrocchi, G., Meth, K., Di Nitto, E., Tamburri, D. A., Van Den Heuvel, W.-J., and Meditskos, G. (2021). Sodalite@rt: Orchestrating applications on cloud-edge infrastructures. *Journal of Grid Computing*, 19(3):29.
- Leite, L., Rocha, C., Kon, F., Milojevic, D., and Meirelles, P. (2019). A survey of devops concepts and challenges. *ACM Comput. Surv.*, 52(6).
- Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., and Ghalsasi, A. (2011). Cloud computing — the business perspective. *Decision Support Systems*, 51(1):176 – 189.
- Morris, K. (2016). *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, Inc., 1st edition.
- OASIS (2007). Web Services Business Process Execution Language Version 2.0. <https://www.oasis-open.org/committees/wsbpel/>. Last accessed on 30-11-2021.
- OASIS (2013). Topology and Orchestration Specification for Cloud Applications Version 1.0. <http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html>. Last accessed: 30-11-2021.
- OASIS (2020). TOSCA Simple Profile in YAML Version 1.3. <http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3>. Last accessed on 30-11-2021.
- OMG (2011). Business Process Model and Notation (BPMN 2.0). <http://www.omg.org/spec/BPMN/2.0/>. Last accessed on 30-11-2021.
- Pierantoni, G., Kiss, T., Terstyanszky, G., DesLauriers, J., Gesmier, G., and Dang, H.-V. (2020). Describing and processing topology and quality of service parameters of applications in the cloud. *Journal of Grid Computing*, 18(4):761–778.
- Ranjan, R., Benatallah, B., Dustdar, S., and Papazoglou, M. P. (2015). Cloud Resource Orchestration Programming: Overview, Issues, and Directions. *IEEE Internet Computing*, 19:46–56.
- Toffetti, G., Brunner, S., Blöchlinger, M., Spillner, J., and Bohnert, T. M. (2017). Self-managing cloud-native applications: Design, implementation, and experience. *Future Generation Computer Systems*, 72:165–179.
- Tomarchio, O., Calcaterra, D., Di Modica, G., and Mazzaglia, P. (2021). Torch: a toasca-based orchestrator of multi-cloud containerised applications. *Journal of Grid Computing*, 19(5).
- Tomarchio, O., Calcaterra, D., and Modica, G. D. (2020). Cloud resource orchestration in the multi-cloud landscape: a systematic review of existing frameworks. *Journal of Cloud Computing*, 9(49).
- Weerasiri, D., Barukh, M. C., Benatallah, B., Sheng, Q. Z., and Ranjan, R. (2017). A Taxonomy and Survey of Cloud Resource Orchestration Techniques. *ACM Comput. Surv.*, 50(2):26:1–26:41.
- Wurster, M., Breitenbücher, U., Kopp, O., and Leymann, F. (2018). Modeling and Automated Execution of Application Deployment Tests. In *Proceedings of the IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 171–180. IEEE Computer Society.