TECHNICAL REPORT

CloudCAMP: A Model-driven Generative Approach for Automating Cloud Application Deployment and Management

Anirban Bhattacharjee, Yogesh Barve, Aniruddha Gokhale
Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, Tennessee, USA
Email: {anirban.bhattacharjee; yogesh.d.barve; a.gokhale}@vanderbilt.edu

Takayuki Kuroda
NEC Corporation
Kawasaki, Kanagawa, Japan
Email: t-kuroda@ax.jp.nec.com

Abstract

Businesses today are moving their infrastructure to the cloud environment to reduce their IT budgets and improve compliance to regulatory control. By using the cloud, industries also aim to deploy and deliver new applications and services rapidly with the ability to scale their applications horizontally and vertically to meet customer demands. Despite these trends, reliance on old school IT management and administration has left a legacy of poor manageability and inflexible infrastructure. In this realm, the DevOps community has made available various tools for deployment, management, and orchestration of complex, distributed cloud applications. Despite these modern trends, the continued reliance on old school IT management and administration methods have left a majority of developers lacking with the domain expertise to create, provision and manage complex IT environments using abstracted, script-centric approaches provided by DevOps tools. To address these challenges while emphasizing vendor-agnostic approaches for broader applicability, we present a model-driven generative approach that is compliant with the TOSCA specification where the users can model their business-relevant models, requiring only little domain expertise. In this context, the paper describes the metamodel of the domain-specific language that abstracts the business-relevant application requirements to generate the corresponding fully deployable infrastructure-as-code solutions, which can deploy, run, and manage the applications in the cloud environment. Our work is focused on realizing a high-performance deployment and management platform for cloud applications with an emphasis on extensibility, (re)usability, and scalability. We validate our approach by a prototypical application model and present a user study to evaluate its relevance.

Contents

1 INTRODUCTION

Cloud Computing is revolutionizing the IT industries by accruing the benefits of agility, scalability, flexibility and ubiquitous computing via a pay-as-you-go utility pricing model. More and more organizations, as well as scientific research and academic communities, are moving towards virtualized cloud environments to host their infrastructure at a low cost with self-service capabilities. Platform-as-a-Service(PaaS) allows application developers to deploy their applications or services on top of Infrastructure-as-a-Service(IaaS) provided by cloud infrastructure providers [1]. PaaS, the cloud utility model, enables deployment and provisioning for the applications in the homogeneous or heterogeneous cloud environment, and enables the applications to elastically scale horizontally and vertically as business requirements change with a pay-per-use cost structure as per the Service-level-Agreement(SLA). It reduces the management complexities from the application developers by abstracting the intriguing details of infrastructure management specifications [2].

Self-service application deployment and management platform is critical for modern cloud-based applications to aid their developers in improved time-to-market. Nevertheless, outages and delays caused by manual changes to configurations, release integration and handoff issues pose significant challenges in this regard. Moreover, modern composite applications depend on component-based modular architectures. The features of the business application's components is combined and orchestrated into a composite application to provide a higher level of functionality. Components typically are connected to other components, e.g., the Web server component runs on an operating system or an application connects to a database and external services such as Apache Spark [3]. Each component of such composite applications can be hosted on different cloud providers depending on the services provided by that cloud provider and their cost. In such scenarios, the different application components need to deployed and configured in a cloud platform-agnostic way to provide elasticity, scalability, interoperability, and portability. Platform-as-a-Service(PaaS), the cloud utility model, needs to guarantee that the applications will elastically scale horizontally and vertically as business requirements change as per the Service-level-Agreement(SLA) [4, 5, 6, 7].

Designing a full-blown deployment model for these composite applications is a time-consuming and error-prone process, and requires the verification of the management plan to justify its validity and reachability. Thus, self-service application provisioning requires extensive planning to run smoothly. Presently, the DevOps community provides tools, such as Puppet[1], Ansible[2], Chef[3], etc. to seamlessly automate the deployment and management tasks for complex, distributed cloud applications. Despite the benefits accrued from these technologies [8, 9], there remain two key unresolved challenges described below, and addressing these problems is the focus of this paper.

Challenge 1: Complexities in Automated Deployment of Composite Applications in Cloud environment: Deployment and continuous delivery of multi-tier, multi-service, and multi-node composite applications pose inherent challenges because of their complexity. Although DevOps tools preclude the need to manually install the individual software components on each machine, configure them and accelerate the application delivery process, developers must possess domain expertise to use the tools. Cloud application deployment and migration workflow includes setting up orchestration tools and creating provisioning environments in a script-centric way to perform a series of automation tasks. Existing provisioning and integration tools need complete topologies by specifying all requirements and functionalities of the components, the order in which the management operations of the component must be invoked, and the relationship between the components. These tools can be prone to vendor lock-in if not configured correctly by acquiring the details of services, functionalities and management features offered by providers to be captured. Further, the pre-deployment validation to satisfy all the end-user requirements and software dependencies is out of scope for these cloud automation approaches. Exploiting pre-configured virtual machines images make the deployment faster, but this method sacrifices the flexibility and agility of the application.

Challenge 2: Complexities in Automated Migration of Application Components in Cloud environment: Application components should be able to migrate between cloud providers to derive the benefit of best services along with optimal pricing [10, 11]. Moreover, for continuous delivery, application components need to the added or migrated between virtual machines or hosts. However, issues such as the application drivers' compatibility with the operating system and their versions, data formatting issues, and inadequately addressed standardization of service definitions among multiple cloud providers, make it tedious and error-prone for component migration, and can even lead to application unavailability. Moreover, for the migration of components, the administrator has to rewrite different plans for each different cloud providers, and for each deployment of new applications, the plans must be written from scratch [12] because the components are not reusable. Migration or addition of components in the DevOps-centric way requires

[1]https://puppet.com/
[2]https://www.ansible.com/
[3]https://www.chef.io/chef/

domain expertise to define the configurations and to install, and uninstall specific software packages. Finally, the DevOps-centric approach does not offer pre-migration validation.

Challenge 3: Need to Leverage Vendor-agnostic Standards: The OASIS standardization body has released the Topology and Orchestration Specification for Cloud Applications (TOSCA) [13] to enable the creation of portable and interoperable cloud applications. TOSCA defines a XML-based modeling specification [14] that formalizes the topology and management tasks of an application in the form of a plans-as-a-service template and it separates the application from the cloud provider-specific API [7, 15]. Despite the existence of standards, such as TOSCA, there is a general lack of capabilities to define the plan for generating TOSCA-compliant full-blown deployable models from the business-relevant model, and launch the application automatically while addressing the challenges mentioned above.

Solution Approach: To address these challenges, this paper describes a rapid provisioning and scalable platform to deploy and manage cloud applications. We propose a deployment and management framework that abstracts the application component specifications and cloud providers specifications into intuitive representations using the principles of Model-Driven Engineering (MDE) [16]. Our CloudCAMP platform facilitates business application developers to select application components from a wide range of choices across hybrid cloud environments. We capture the application component specifications and cloud specifications in a meta-model and develop a domain-specific modeling language (DSML) that can transform the business-related design into infrastructure-as-code (in our case it is Ansible specific). The main contributions of the paper are threefold:

1. **Abstraction Layer for Cloud Provider's specifications and Applications specifications:** We present an extensible meta-model that captures the commonalities and variabilities in the application type specifications, as well as operating systems, hardware and cloud providers endpoint specifications. We also capture the dependencies and intrinsic relationships among application components. Finally, the metamodel also captures the scaling and replication capabilities.

2. **Model-to-Infrastructure-as-code Transformation:** We describe how this metamodel is realized in a DSML to automate the synthesis of a full-blown deployment model based on DevOps Ansible specifications. It also ensures the order in which application components should be executed by checking the relationship among the application elements of the business relevant model. It is integrated with known constraint checking capabilities to verify the correctness of the model. It can also

parallelize the deployment and management process to reduce the maintenance time.

3. **Concrete implementation and demonstration in WebGME:** We implement our approach in a cloud-based MDE environment called WebGME [17]. The generative capabilities of our approach are applied as a WebGME plugin which generates infrastructure code based on TOSCA specifications. These TOSCA-compliant IAC solutions are executed by our plugin to deploy, (re)start, stop the application components and manage the application in the cloud environment. The deployment and management platform is extensible, reusable, and scalable. We validate our approach in the context of a prototypical implementation and user study.

Organisation of the paper: This paper is focused primarily on system configuration design automation for deployment and management of composite applications in the cloud environment. It assesses the state-of-the-art system configuration design automation challenges and proposes a novel architecture to solve the configuration automation problem. The rest of the paper is organized as follows: Section 2 provides an overview of the TOSCA specification and the DevOps tools and their limitations; Section 3 presents a survey of existing solutions in literature and compares to our solution; Section 4 describes the problem formulation; Section 5 presents our approach to design and implement CloudCAMP for automating TOSCA-compliant deployment plan generation using MDE environment WebGME and IAC (Ansible) solution; Section 6 evaluates our metamodel for a prototypical case study and present a user survey; and finally, Section 7 concludes the paper alluding to future directions.

2 BACKGROUND

A collaborative study by Emerson Network Power and Ponemon Institute showed that the average expense of data center outage across industries was $\$8,851$ per minute, which is a significant 57-percent rise from the $\$5,617$ in 2010 [18]. To minimize the maintenance cost speeding up deployment and management for cloud application is necessary. While TOSCA leverage interoperability issues among cloud providers, DevOps is an attempt to make cloud operations run as efficiently the. We now describe DevOps tools and TOSCA briefly and will explain why the combination of DevOps and TOSCA helps to make high-performance deployment and management automation tool.

2.1 DevOps Tools: Significance and Limitations

Automation is the key to tighten the integration and efficient collaboration between development and operations, and DevOps community is continuously delivering new approaches, tools, and open-source artifacts to implement such automated processes. To simplify provisioning, deployment, and management of applications in the cloud environment, the script-centric DevOps (development & operations) community provides proprietary toolchains, such as Chef [19], Cfengine [20], Puppet [21], Ansible[4], Salt[5] and Juju [6]. However, the infrastructure-as-a-code approach needs technical domain expertise and manual construction of different configuration scripts, recipes (Chef), manifests (Puppet), charms (Juju), playbooks (Ansible) or modules (Salt) is time-consuming and not reusable. The burden of describing all the component specification and dependencies among them is on the developers. Also, pre-deployment validation is not integrated to check whether deployment will succeed or not and these bottlenecks can cause prolonged maintenance and deployment issue.

To address these problems, we propose declarative, high-level and intuitive abstractions on top of the script-centric approaches to save the developer from the complexity and needed domain expertise. There are a few more criteria such as compatibility with different clouds and different operating systems and their versions, support for both network and IT resources, satisfy constraints, and the dependency of the application on software packages need to be considered.

2.2 Significance and Basics of TOSCA Standardization

The OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) [13] is a standardization to enable a portable and automated deployment and management platform of composite and distributed applications in the cloud environment. TOSCA modularizes the components of composite applications and their management as topology templates and dependencies among the application components as relationships. Thus, TOSCA provides abstract standardization of automated provisioning.

The Node Templates and Relationship Templates collectively define the Topology Template of a distributed composite application in TOSCA. Node template represents the application components, and the relationship template defines the relationship between two nodes. Both nodes and relationships are typed and have a set of type specifications

[4]https://www.ansible.com/
[5]https://saltstack.com/
[6]https://jujucharms.com/

to provide significance and variability to generic TOSCA elements. The Node type is the building block of a service, and it defines the implementation artifacts and deployment artifacts for different software components. Artifacts represent the content needed to deploy and manage the defined application component [22]. Relationships comprise of various semantics, such as node A is hosted on node B, or node A connects to node B, and node A has to be started before node B, etc., that are defined as part of the relationship type specification.

TOSCA aids application developers and operators to explicitly model management best practices and recurring tasks into plans to reduce the manual effort. Plans automate the creation, migration, and termination of the instance of a service. These plans include several strategies for scaling, deployment, migration, and completion of an application. TOSCA also enables interoperability and portability of management plans so that a cloud service can quickly deploy and migrate between different cloud environments. The plan/workflow combines the sequence of the management operations involving various nodes and relationships between them. However, deploying and managing services require extensive – mostly manual – efforts by the end-users. The attributes and parameters of multiple software node stacks, which is contained with different node templates for middleware and application layer components, hosted on different servers, need to be defined and the dependency of the software stacks and the relationship among each other need to be predefined in the TOSCA topology template.

In contrast to the presented approaches, we have captured the TOSCA specifications for node and relationship types for the cloud applications in our WebGME based CloudCAMP metamodel. The templates to model and the specific elements associated with the cloud applications are handled by the DSL to automatically derive Ansible infrastructure-as-code solution based on TOSCA standardization from the high-level business compatible models. Our WebGME plugin then executes this Ansible code for deployment and management of cloud applications. Our approach allows users to focus primarily on the business-relevant application components, by simplifying the error-prone and time-consuming creation process immensely and by minimizing the requisite expertise.

3 RELATED WORK

The problem of deployment and management abstraction has been explored in the area of cloud automation and orchestration. In this section, we compare the existing efforts in the literature with our work.

The script-centric DevOps community provides toolchains for eliminating the disconnect between developers and operations providers [23]. However, we need declarative, high-level modeling abstractions on top of the script-centric tools to provide a self-service provisioning platform. Cloudify and Apache Brooklyn[7] enable cloud application orchestration of topology templates according to the TOSCA specification. In this context, Alien4Cloud [24] proposed a visual way to generate TOSCA topology model, which can be orchestrated by Apache Brooklyn. However, building the proper topology even using such an MDE approach combined with the TOSCA specification needs domain expertise. In our approach, we abstracted all the application, and cloud-specific details in our metamodel and our DSML converts the business model to TOSCA-compliant Ansible-specific code using an extensible knowledge base of the application related dependencies.

Cloud orchestration tools like Apache Scalr[8], CloudFoundry[9], Cloudify[10] are excellent toolchains to deploy and manage applications on any cloud providers. They provide sophisticated techniques to monitor the health of the application and to migrate between the cloud providers using standardized approaches. However, they all suffer from the same limitations of defining the complete deployable model with all the functionalities and features. The use of these toolchains adds the burden of configuring the application components and integrating pre-deployment verification on application developers.

To that end, several pattern-based approaches are proposed to reduce the complexity of designing the deployment of application service [25, 26, 27]. They differentiate between business logic and the deployment on a platform in service-oriented architectures. Each pattern offers a set of capabilities, and characteristics. Model-based patterns of proven solutions in the functional and non-functional properties of application service deployment in cloud infrastructures [28] is also proposed. MODAClouds (MOdel-Driven Approach for the design and execution of applications on multiple Clouds) [29, 30] allows users to design, develop and re-design application components to operate and manage in multi-cloud environments using a Decision Support System (DSS). In Computation Independent Model (CIM) design artifacts are semi-automatically translated to Cloud-Provider Independent Model (CPIM) level, where full deployable abstract cloud model is generated by matching the application patterns. The abstract deployment model is concretized to Cloud-Provider Specific Model (CPSM) by domain-specific language. Similar to our approach, they also support reuse and role-based iterative refinement in a component-based approach. However,

[7]https://brooklyn.apache.org/

[8]https://scalr-wiki.atlassian.net/wiki/display/docs/Apache

[9]https://www.cloudfoundry.org/

[10]http://getcloudify.org/

their deployment plan generation lacks verification and extensibility. Unlike our approach, they also did not consider distributing application components in a heterogeneous cloud environment.

Several efforts come close to our vision of CloudCAMP. For instance, ConfigAssure [31] is a requirement solver to synthesize infrastructure configuration in a declarative fashion. All the requirements are expressed as constraints on the configuration by the developer and a configuration database containing variables is predefined as a deployment model by the provider. Kodkod [32] is a relational model finder which takes these arguments as a first-order logic constraint in the finite domains. Similarly, Fischer et al. [33] proposed a system called Engage to deploy and manage the complex application from a partial specification using a constraint-based algorithm. Aeolus Blender [34] is a similar toolchain for automatic deployment and configuration of applications in the cloud. The toolchain comprises the configuration optimizer Zephyrus [35], the ad-hoc planner Metis [36] and the deployment engine Arnomic [11]. Zephyrus automatically generates an abstract configuration of the desired system based on the partial description. It translates the model to MiniZinc, which is a CSP solver to verify the generated model. Metis concretizes the deployable model produced by Zephyrus. Arnomic takes the optimized deployable model and deploys it on the cloud platform. They guarantee to satisfy all the end-user requirements software dependencies and also provide the optimal solution for the number of active virtual machines. Unlike our use of knowledge base, these efforts use a CSP solver to transform the business model. CSP solvers can take significant time to execute, and defining constraints on the configurations requires expert knowledge of the modeling system.

Hirmer et al. [37] focused on producing complete TOSCA-compliant topology from users' partial business relevant topology. The end-users have to specify the requirements defined directly by the definitions of the corresponding node types or added manually by the developer for refinement. Their completion engine compares this specification with target models and adds the missing components to make it a fully deployable model. Finally, the deployable model can be deployed in the cloud platform, and the service components can be executed in the right order using OpenTOSCA toolchain [38]. CELAR [39] exercises the combination of MDE and TOSCA specification to automate deployment cloud applications. However, their approach to topology completion is fulfilled by requirement and capability analysis on the node template, whereas our model transformation is based on querying the knowledge base and idempotent infrastructure code generation.

[11]http://armonic.readthedocs.org/en/latest/index.html

4 PROBLEM FORMULATION

In this section we use a real-world use case to motivate the key requirements of our solution and the challenges incurred in meeting these needs that address the issues highlighted in Section 1.

4.1 A Motivating Scenario

Our use case example describes a simple business application that is to be deployed on a cloud platform. It illustrates a PHP-based website application that stores data in a relational database. The application is hosted on two cloud provider platforms. Figure 1 shows the application topology consisting of a Web front-end and a MySQL database at the backend. The application consists of two connected software stacks. The left stack of the desired model hosts the business logic of the frontend. The PHP module is the front-end, which needs to be hosted on Apache web server. The web server needs to be deployed on Ubuntu 16.04 server[12], which runs as a virtual machine(VM) server. This Ubuntu VM needs to be managed using an OpenStack [40] cloud platform. The product data are stored in a database, and the architecture is shown in the right stack of Figure 1. The backend database runs on MySQL DBMS [41], which needs to be deployed on Ubuntu 14.04 server, which itself will run as a virtual machine(VM) server. This database VM needs to be managed by AmazonEC2 [42] cloud platform [43].

4.2 Requirement 1: Automating the Completion of Infrastructure Design Provisioning

As depicted in Figure 1, to start PHP and MySQL-based website, first the virtual machine should be spawned with Ubuntu operating Systems in the OpenStack and Amazon AWS cloud platforms, respectively. The PHP module requires Apache httpd server as a dependency, so Apache needs to be installed and configured along with PHP. On the backend server, MySQL needs to be installed and configured also. In addition to that, the web application needs to have PHP Database Connectivity to access the database, so the required drivers need to be installed. In addition to this complexity, the database service should start before the PHP application service to run the WebApp properly. Launching this connection is application-specific because there is no standardized way to describing how to set such a database endpoint information. Thus, domain knowledge is required to

[12] http://www.ubuntu.com/

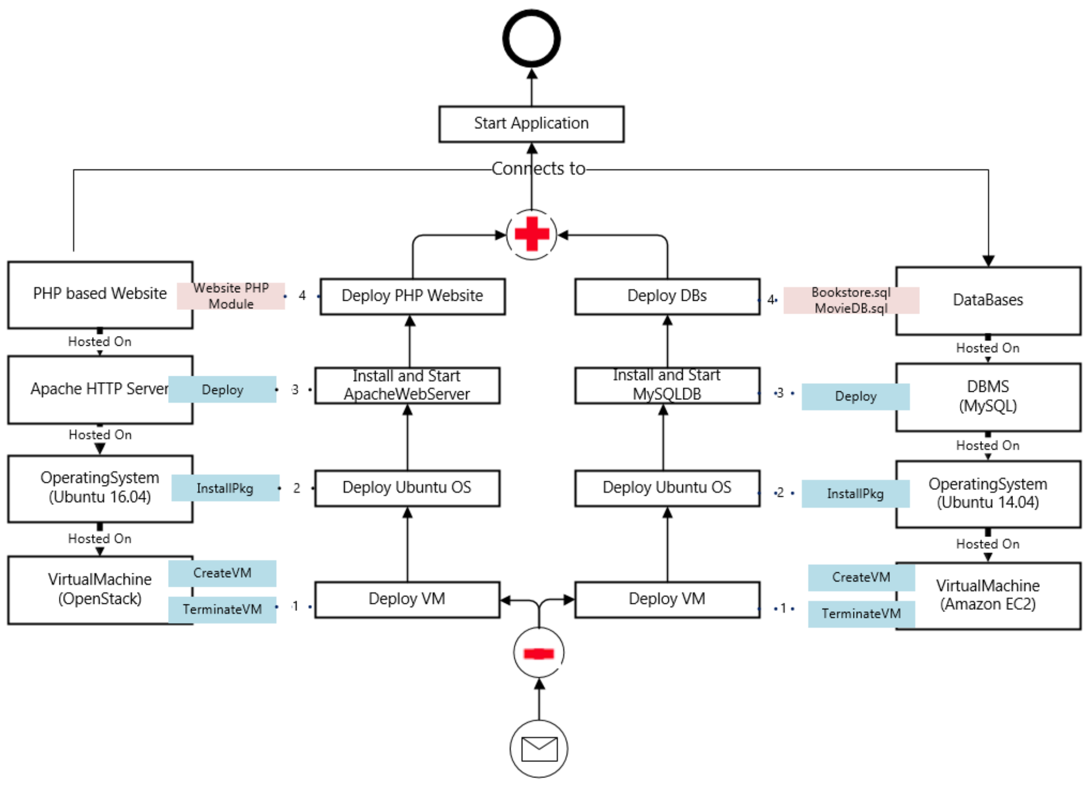
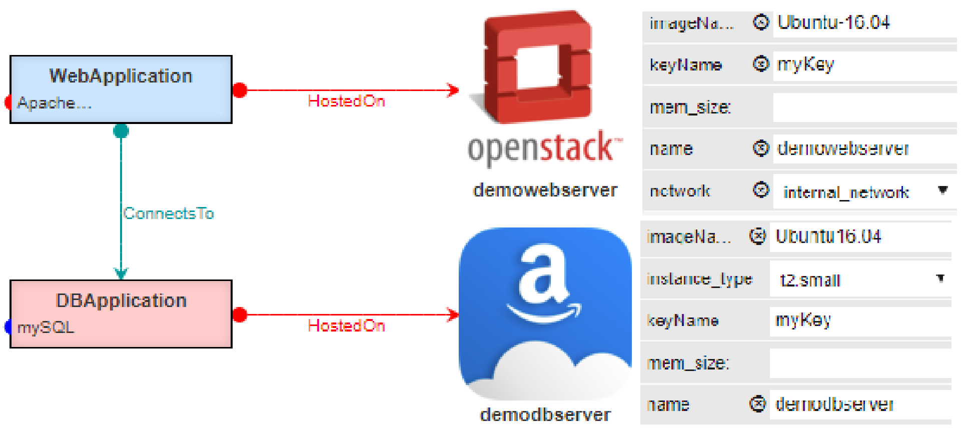


Figure 1: A full-blown PHP and MySQL based Application deployment workflow based on TOSCA-compliance

provision a simple web application correctly. The script-centric deployment plan creation and managing all the services in proper order for complex distributed application is tedious and error-prone and leads to delaying application deployment time.

This motivating scenario illustrates many different platform and technology dependencies as well as the ordering in which application components must be connected and started. An application developer is unlikely to possess the in-depth technical expertise needed in deploying such application topologies across potentially a range of variable platform choices. Thus, there is a need for application developers to be able to model only the most important parts of the application structure and a tooling mechanism then transforms an incomplete and abstract business-relevant model to a complete topology as an infrastructure-as-a-code solution. A full, resulting topology is needed because otherwise it cannot be fed into an automatic provisioning and orchestration engine because of the missing components.

As an example of our idea, consider how a developer can only design the Web App as shown in Figure 2 by specifying business needs and the underlying DSML will transform it into Figure 1. Thus, the high-level model will simplify the modeling of business-relevant logic and automatically take care of non-business centric deployment and management



imageNa...	Ubuntu-16.04
keyName	myKey
mem_size:	
name	demowebserver
network	internal_network

imageNa...	Ubuntu16.04
instance_type	t2.small
keyName	myKey
mem_size:	
name	demodbserver

Figure 2: A WebApp business model

artifacts and relationship types. The key challenges in meeting this requirement are listed below:

4.2.1 Capturing the Application and Cloud Specifications in the Metamodel

To transform the half-baked model to the full-blown model, we need to capture all the application and cloud-specific endpoints in our metamodel. The deployment modeling automation metamodel needs to be developed by harnessing a combination of (1) reverse engineering, (2) dependency mapping across heterogeneous clouds, (3) dependency mapping across different operating systems and their versions, (4) semantic mapping, (5) business policy, and (6) prototyping. Capturing this variability enriches the expressive power, multi-cloud tool support and interoperability of the platform. Prototyping and reverse Engineering can identify the different application, cloud and operating system specific endpoints. The dependent software packages, their relationship mapping and configuration templates can be recognized and realized in the metamodel. In this way, the abstraction necessary to support the application deployment and management can be determined and clarified. The set of available building blocks, requirements, policies, SLAs, cost, and other information concerning the implementation of the services and all other known constraints can be pre-defined in the high-level application metamodel.

4.2.2 Defining a Language for Model Transformation

The domain-specific modeling language must then transform the business model to an infrastructure-as-a-code solution based on the metamodel and the pre-defined knowledge base. The transformation should guarantee a semantically and syntactically correct model based on TOSCA specifications. It must be able to parallelize the processes to speed up the entire deployment process. The resulting full-blown models should be usable as input to an existing provisioning engine, which will execute the model and deploy the desired application in the cloud environment, by installing all the required packages and configuration. Finally, all the application services must be started in the right order, based on the runtime dependencies of each service.

4.3 Requirement 2: Support for Continuous Integration, Migration, and Delivery

Now suppose that for the use case in Figure 1, the customer wants to execute a management task to migrate the web front-end to Amazon's IaaS offering Elastic Compute Cloud (EC2) with the purpose of shrinking the number of cloud providers. This migration strategy gives rise to a few issues, such as missing database drivers and missing configurations of the database service. These problems compromise the application's functionality and need to be considered by experts possessing the knowledge to recognize the following challenges in advance. To migrate the frontend, we have to perform the following steps: (i) shut down the old virtual machine on OpenStack, (ii) create the new virtual machine on Amazon EC2, (iii) then install the Apache HTTP server and the other dependencies, (iv) deploy the PHP based frontend, and (v) set the database's IP-address, username, and password in the frontend's configuration. Moreover, migration can be stateful also, so the current state of the application needs to be actively replicated in a new VM and then the old VM will be detached.

If the administrators have to accomplish this migration manually for a complex system, sheer technical expertise about the different APIs and employed technologies is required. Moreover, the existing application might be extended, such as adding one database server or adding analytics tool with the application. Our platform should provide seamless support for up-gradation and addition of the composite application. Thus, it motivates the need for a fully automated standardized mechanism to generate perfect plans. So, the management cost is always a player for IT, and that drives the scenario for model-driven migration tool for cloud applications [44].

The extended scenario indicates that due to constant changes in business strategies, the business model is always changing, and hence the platform should be integrated with smart migration techniques to cope with the speed of business up-gradation and speed of service delivery. Thus, for providing scaling, load-balancing, fault-tolerance, etc. developers might need to move the applications and related VMs from one server to another server, and might have to add, upgrade resources based on requirements. The migration path of the application stack from one cloud provider to another is also very steep because of heterogeneity, proprietary APIs, non-standardized data formats and different security mechanisms of various cloud providers. Nevertheless, the challenge here also lies in capturing the application and cloud specifications in the metamodel and the DSML. However, apart from that, there are few more problems, which are listed below:

4.3.1 Extensibility and Reusability of the Application Components

New application components need to be added at runtime to the existing application by leveraging the platform. The specifications captured in the metamodel should be modularized and loosely coupled with a particular application. DSMLs should do all the binding after querying for the specification for particular application type in the knowledge base, and then the DSML will generate concrete cloud-specific, operating-system specific infrastructure-as-a-code solution. The IAC is idempotent, so it will not change the existing deployment if configured correctly. The correctness of the added application components can be validated using constraint checker at the model level.

4.3.2 Extensibility of the Platform

The platform can transform the business-relevant model to actionable infrastructure-as-a-code, which produce application deployments in the cloud. However, the challenge is to make the platform loosely coupled with any DevOps or orchestrating tool, so that later different tools can be added if required. Moreover, adding new application requires reverse engineering the application components, and capturing the application specifications in the metamodel of our platform, and adding new cloud providers also requires a similar approach. Defining commonality and variability points is critical to building a modularized platform so that extensibility of platform will be relatively easy.

To summarize, the goals of our approach can be described along three different dimensions as follows:

1. The complexity of infrastructure designing is abstracted to a business-relevant model.
2. The overall required manual effort and expertise for modeling is reduced.
3. The approach is TOSCA-complaint, all models are portable, vendor-neutral and interoperable.

5 DESIGN & IMPLEMENTATION of CloudCAMP

The CloudCAMP platform is intended to aid the application providers to deploy and manage their applications with speed and agility using Model-driven Engineering (MDE) approach. This section delves into the design and implementation of the CloudCAMP platform showing how it addresses the challenges from Section 4. The complete workflow of our approach is depicted in Figure. 3.

5.1 Modeling and Deployment Workflow

To better understand the design details of the our visual MDE framework called Cloud-CAMP, we first briefly describe the user's perspective and the steps they must take as shown in the Figure. 3. In this regard, the business application developer models the application in CloudCAMP. A business application is a compendium of different application components, which the developer has to select. Moreover, they need to specify on which type of hosts they want to deploy the application components. These modeling features in CloudCAMP are dictated by CloudCAMP's domain-specific modeling language (DSML), which in turn has an underlying metamodel (which is akin to the grammar of a programming language) that captures the abstract and concrete syntax of all the artifacts of interest: cloud infrastructure specifications, application type specifications and functionalities. A few user-defined specifications, which represent the variability points for our model, need to be specified by the application developer. Thus, given an abstract description of a cloud application model as an input, the MDE-based workflow is responsible for transforming the business model to a deployable infrastructure-as-code (IAC) solution. Our model transformation code realizes the operational mapping between the application components, which are selected by the developer and the attributes they specify. Then the IAC code generation algorithm, which we describe later in this section, queries the knowledge base and finds all the required dependencies that are needed based on the selected host type. Finally, CloudCAMP generates the workflow by abiding to business rules to execute the IAC solutions in the proper order to deploy and run the business application correctly across cloud providers. For our solution, we will elaborate the approach to building our CloudCAMP platform on top of WebGME and Ansible.

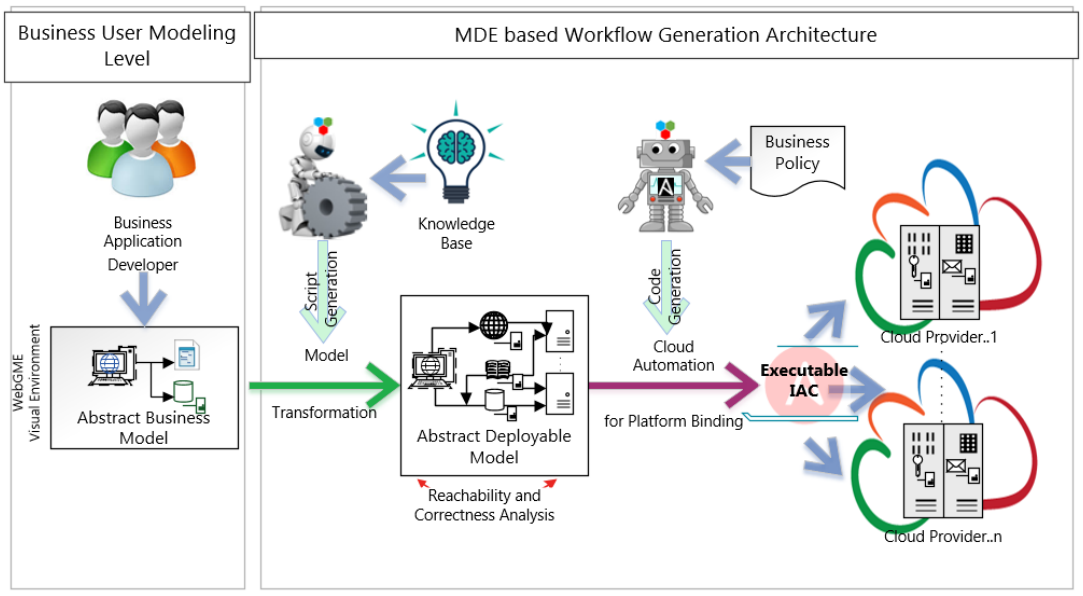

Business User Modeling Level
MDE based Workflow Generation Architecture
Business Application Developer
WebGME Visual Environment
Abstract Business Model
Knowledge Base
Script Generation
Model
Transformation
Abstract Deployable Model
Reachability and Correctness Analysis
Business Policy
Code Generation
Cloud Automation
for Platform Binding
Executable IAC
Cloud Provider..1
Cloud Provider..n

Figure 3: The complete workflow of the CloudCAMP approach

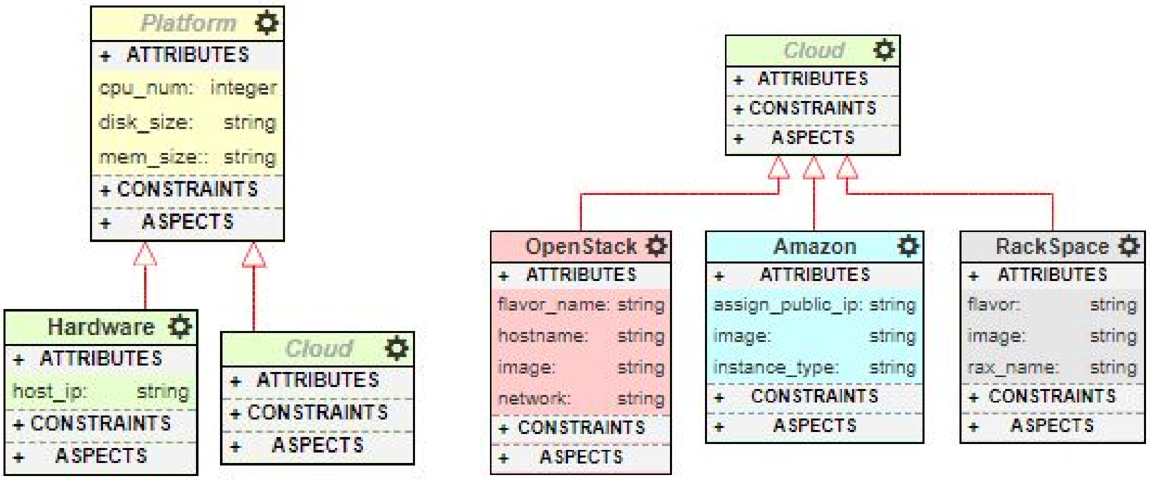
5.2 Metamodeling: Capturing the Application and Cloud Specifications

The CloudCAMP platform aims to ease the design by abstracting the design complexities by isolating the application from deployment and infrastructure technologies according to TOSCA specification as described in Requirement 4.2.1. In much the same way that programming language grammar is specified in the Backus-Naur form, meta-rules for modeling languages need to be defined. To that end, as the CloudCAMP language designers, we decided to define the key elements in our language and their relationships. We decided to offer different node types, which are application components such as WebApplication, DatabaseApplication, DataAnalyticsApplication, etc., and different cloud providers such as OpenStack, Amazon, etc. The goal then is to concretize the abstract node type by matching the application developers' desired specification with the pre-defined functionalities captured in the CloudCAMP metamodel. Concretized node templates will be bound to specific cloud provider types and their VMs and operating system to create a dependency graph that has to be executed to deploy the application on the desired target machine.

We used the WebGME (`www.webgme.org`) MDE framework to define the metamodels for the CloudCAMP DSML. WebGME is a cloud-based framework that offers an environment for DSML developers to define their language and create model parsers that among many other things can serve as generators of code artifacts, in our case Ansible scripts.

The model parsers can also trigger additional actions, such as executing ansible to deploy the application according to the model.

Metamodel for the Cloud Platforms: In creating the metamodel for cloud platforms, we observed (i.e., reverse engineered) the process of hosting applications across different cloud environments, and captured all the commonalities and variabilities in the CloudCAMP metamodel. In our metamodel, the specifications for various cloud platforms (OpenStack, Amazon EC2, and RackSpace) for provisioning virtual machines(VMs) with different operating systems are captured as the variability. The developer can select the pre-defined VM flavor, available networks, available images, which are obtained by querying the specific cloud platform to populate our metamodel. They will be able to choose their desired operating system images to spawn the VM. They have to specify their environment file, key for the selected cloud host type, which are the endpoints to bind to a particular cloud provider. They can also choose a pre-deployed machine by providing the IP address and operating system and versions. The platform dependent specifications are captured as shown in Figure 4.



(a) The predefined platform types and their attributes

(b) The predefined cloud providers with captured platform specific attributes

Figure 4: The predefined specifications for Deployment Platform

All these specifications is captured in the metamodel as shown in the M1 and M2 level of Figure 5. The figure is defined based on Meta-Object Facility (MOF) standard provided by Object Management Group (OMG)[13].This implies that with this approach, the application developer can configure the node in defined cloud platform or particular

[13]http://www.omg.org/

target system without providing any deployment or implementation artifacts that contain code or logic.

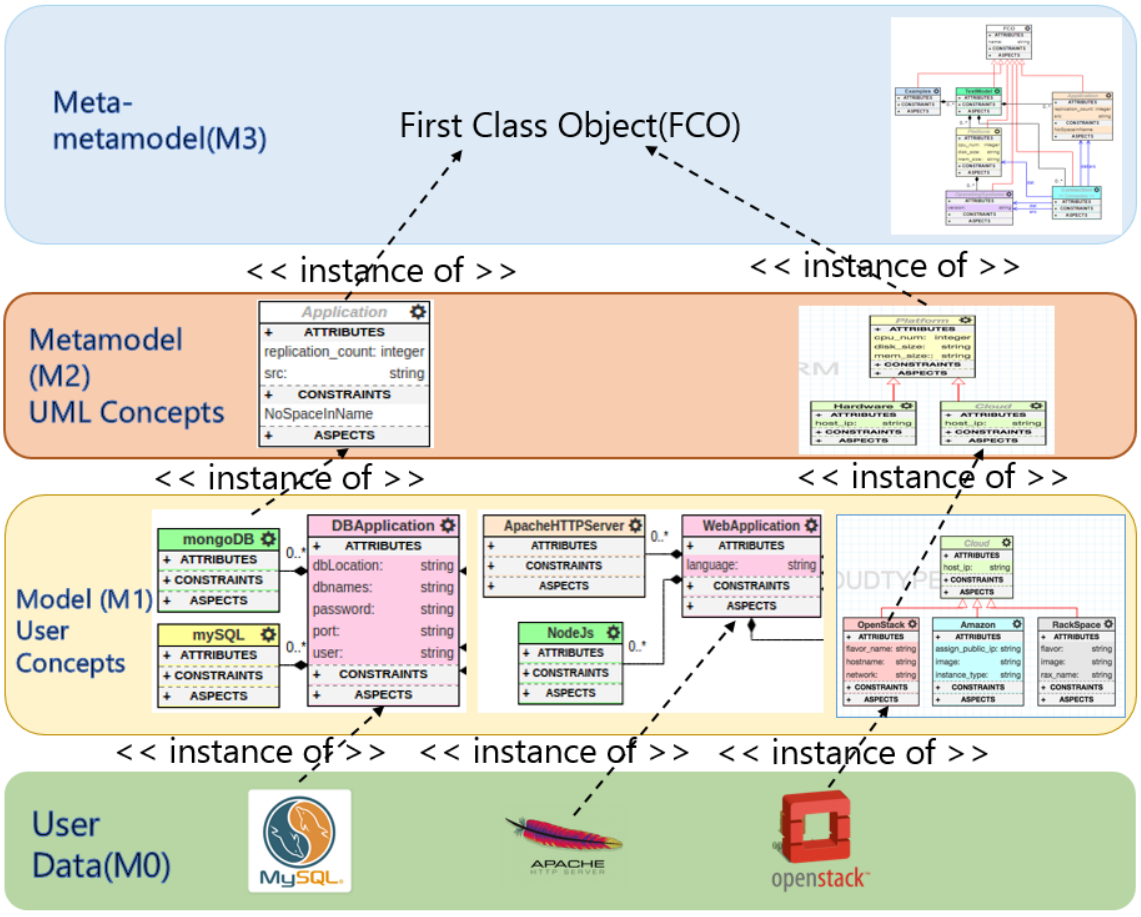


Figure 5: A part of Meta-Object Facility model of CloudCAMP Platform

Metamodel for Application Component Specifications: The metamodel of CloudCAMP also captures each application type components. For instance, the 3-tier application from Section 4, shows different application node types, e.g., web server, database server. It also shows their attributes, e.g., an Apache web server or MySQL database. The programming language is also specified, e.g., PHP. Once again, the different application node types captured in our metamodel are the result of reverse engineering typical applications that are hosted on the cloud these days. The metamodel has been designed for extensibility so that in future we can add more application node types, e.g., stream processing operators that execute on systems such as Apache Spark or Edgent.

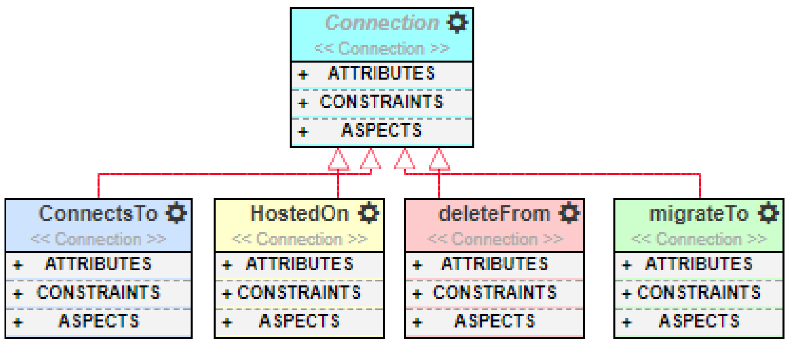
As an example, for deploying LAMP (Linux operating system, the Apache HTTP Server, the MySQL RDBMS, and the PHP programming language) archetypal model of application stacks as shown in the M0 level of FigureFig:mof. We will walk through the specifications needed to be captured for WebApplication and DBApplication component types.

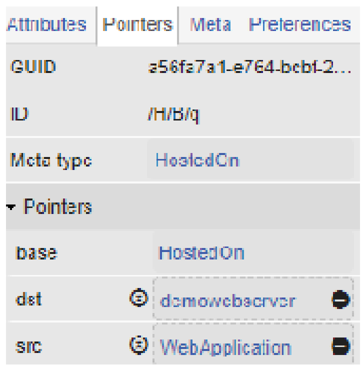
As shown in the M0 level of FigureFig:mof, the HTTP servers for the webEngines are captured in WebApplication component type, and that is related to the node template for a WebApplication. The development languages and frameworks (Node.js, PHP, Django, etc.) of the webApplication is taken as attributes in the software property as depicted in M1 level of Figure 5. Similarly, as shown the M0 level of FigureFig:mof, the software for the database types are captured in DBApplication component type, and that is related to the node template for the Database Application. The related features such as the user id, password, specific binding port number of the Database application are stated as attributes, which is captured in M1 level of the MOF.

With some existing application component node types defined in our metamodel, the user can just select their desired node types for deploying the service template. Our metamodel is extensible and reusable, so the component types can be added as required.

Defining the relationship among Components: There are four relationship types that bind the node types in the CloudCAMP metamodel. A 'hostedOn' relationship type means the source node type is required to be deployed after the destination node type, e.g., Webserver is hosted on Ubuntu 16.04.

The 'ConnectsTo' relationship type is used to relate the source node type's endpoint to the required target node type endpoint if they are dependent on each other, e.g., web server connects to database server. The stack of node types linked by 'connectsTo' can be configured in parallel, but the service at the source node needs to deploy after starting the target node. The 'deleteFrom' connection type defines the source node type is required to be removed from the end node type. Finally, the 'migrateTo' connection type defines the





GUID	a56fa7a1-e764-bcbf-2...
ID	/H/B/q
Meta type	HostedOn
Pointers	
base	HostedOn
dst	demowebserver
src	WebApplication

Figure 6: Captured relationship specifications

source node type that is to be migrated to the end node type. The 'migrateTo' relation type cannot be defined without a 'deleteFrom' connection type to assure correctness of the business model. The Figure 6 depicts types relationship specifications.

As shown in Figure 2, the DBApplication service should start before PHP based WebApplication. We have captured these relationship types in our metamodel.

5.3 Creating the Domain Specific Modeling Language (DSML)

As per Requirement 4.2.2, CloudCAMP generates the full blown infrastructure-as-code (IAC) solution from the abstract business model by incorporating the deployment and infrastructure specification captured by reverse engineering. According to the TOSCA specification, node types have associated operations that deliver the automation (e.g. in the form of an IAC solution) for the application lifecycle operations and deployment of a node.

For example, the node type has to be hosted on Ubuntu, and the implementation artifacts for an Apache would associate scripts to TOSCA node operations like configure, start, or stop to manage the state of Apache at runtime as shown in Figure 1. For automating the deployment and management workflow, we transform our WebGME metamodel to deployable Ansible specific IAC solutions.

Generation of Ansible specific Infrastructure-as-code Solution for Deployment: The CloudCAMP DSML in WebGME is built using JavaScript, NodeJS, and a MySQL database. The developer specified the desired cloud specification, Operating system and version specification, etc. for the host cloud platform type. The source path of their application code (such as git repository path) and the other specifications based on the application type needs to be defined.

Once the developer has modeled the application, they choose to execute the model parser and generator, which is available in WebGME as a plugin. This plugin executes code that implements Algorithm 1. The algorithm will spawn the VMs in the specified cloud platforms because it is the destination of 'HostedOn' connection. Wherever possible, CloudCAMP will ensure that scripts specific to VM spawning run in parallel to provide faster deployment. Once the VMs are spawned, CloudCAMP queries a knowledge base we have created, which is a MySQL RDBMS, where we created different tables to store the software dependencies. Based on the operating system type, the packaging tool (apt for Ubuntu, yum for RHEL) is needed to be selected to install the software.

We predefine the software dependencies in a relational table with a key-value pair, and we maintained normalized form with indexing to make the database easier to use and maintain. The software packages needed for particular application type is also dependent on operating system, and its version. We maintain primary key and the foreign key relationship among tables to build robust maintenance structure. Further, based on the user's business model specifications, we fire the query in the database table, join the related tables and fetch the result. Then, we fill the application specific predefined templates and generate the whole Ansible specific infrastructure code.

Algorithm 1 Ansible Deployment Script Generating algorithm

```
1: procedure GenerateExecuteAnsible
2:     cloudModel ← js Objects to store cloud specs
3:     appModel ← js Objects to store app specs
4:     top:
5:     if ConectionType 'HostedOn' then
6:         cloudType ← the destination node of connection
7:         appType ← the source node of connection
8:         if cloudType 'Desired Cloud Platform' then
9:             loop:
10:            Traverse the cloudModel
11:            Fill 'cloudType' specific API Template
12:            Generate 'cloudType' specific ansible script
13:            Execute 'cloudType' specific ansible script
14:        end if
15:        IPAddress(es) ← IP Address of target machine
16:        Create empty Ansible Tree Structure
17:        Fill 'hosts' with IPAddress(es) in context of Application Component Location
18:        if appType 'Desired Application Type' then
19:            loop:
20:            Traverse the appModel
21:            Query the dataBase 'WHERE appType = '..."
22:            Fill 'appType' specific API Templates to deploy
23:            Create complete Ansible Tree Structure
24:        end if
25:        Check Connection Type among app components
26:        Wait_for SSH to be enabled in target machine
27:        loop:
28:        if ConectionType 'ConnectsTo' then
29:            Find the source and destination application type
30:            Execute the script of destination first
31:            Execute the script of source after
32:        elseRun all scripts in parallel
33:        end if
34:    end if
35: end procedure
```

Determining order of deployment and execution: The code execution plugin, which is a NodeJS script builds the dependency tree for the application types defined in the metamodel. We generate different Ansible roles for different component types, and Ansible can dispatch tasks to multiple hosts in parallel. If there is a 'connectsTo' relationship in the model, we add a ‘wait_for’ condition to let the dependent script be completed first. All the ‘HostedOn’ dependent building blocks run in a linear fashion. Thus, the Ansible script remotely connects to the deployment hosts and deploys the application in proper order and runs it successfully. The business user can deploy a predefined application using our tool without writing a single line of code and without needing any significant domain expertise. Moreover, the components of the model are reusable, and the business model can be extended with great ease, which is beneficial for continuous delivery.

Algorithm 2 Ansible Migration Script Generating algorithm

```
1: procedure GenerateMigrateAnsible
2:     cloudModel ← js Objects to store cloud specs
3:     appModel ← js Objects to store app specs
4:     top:
5:     if $ConectionType$ ‘deleteFrom’ then
6:         cloudType ← the destination node of connection
7:         appType ← the source node of connection
8:         loop:
9:         Find IP address of the destination node
10:        Traverse the appModel
11:        Query the dataBase ‘WHERE appType = ‘...”
12:        Fill ‘appType’ specific API Templates for deletion
13:        Generate ‘apptype’ specific script for deletion
14:
15:    end if
16:    if $ConectionType$ ‘migrateTo’ then
17:        ...//same as GenerateExecuteAnsible of Algo. 1 steps [6-26]
18:    end if
19:    if $migrationType$ ‘stateless’ then
20:        Execute deletion and migration scripts in parallel
21:    else if $migrationType$ ‘stateful’ then
22:        Instantiate a manager node
23:        Generate ansible script to deploy HAProxy on the node
24:        Generate ansible script to deploy HAProxy on the node
25:        Checkpoint the current state of the old machine
26:        Redirect all the new connections to the migrated node
27:        Restore checkpoint on the current machine
28:        Execute deletion script on old machine
29:        Remove HAProxy Loadbalancer
30:    end if
31: end procedure
```

Generation of Ansible-specific Infrastructure-as-code Solution for Migration: The algorithm for generating the Ansible specific code is briefly portrayed in Algorithm 2. The ‘deleteTo’ connection type specifies from where the user wants to move the application components and attach a ‘migrateTo’ connection type to indicate the destination. The migrationType (stateless or stateful) is also needed to be selected. Although actions are taken for live migration, an application component from one VM to another depends on the application component type, which is a hard problem. For example, live migration of DBApplication needs two-phase commit protocol, and consensus algorithm to make it reliable. For the sake of simplicity, in the Algorithm 2 we generalize our approach. Our future work will consider more complex scenarios of live migration and application consistency and availability issues.

According to Algorithm 2, it will spawn a new VM with the new operating system for the 'migrateTo' destination node. For Stateful migration, our platform creates a manager node with a load balancer, and deploy the application on the current node. From that point of time, load balancer redirects all the new request to the current node, and it checkpoints the current state of the old node and restores it in the current node. Finally, it detaches the load balancer node. Thus, it produces the full infrastructure-as-code solution along with the related configuration files. All of these complete the Ansible layout tree structure helps to migrate application components from one node to another node.

Constraints checking of Business Model: We also validate the business model by checking the constraints of the model to ensure that the user specified models are "correct-by-construction." We determine the endpoints for application component types, the relationship types, cloud-specific types, etc., and verify the business model as a whole before generating any infrastructure code. The example of few constraints are shown below:

- $\forall$ Applications $\in$ WebApplication $\exists!$ WebEngine
- $\forall$ Applications $\in$ DBApplication $\exists!$ DBEngine
- $\forall$ Platform $\in$ Openstack $\exists!$ imageName
- $\forall$ Applications $\in$ DBApplication $\exists!$ (user $\wedge$ password $\wedge$ port)
- $\forall$ Applications $\in$ DataAnalyticsApp $\exists$ processEngine etc.

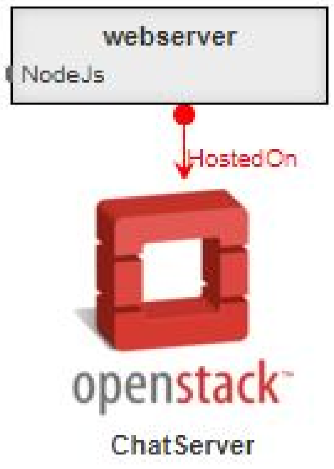
Thus, we validate the business model by satisfying the constraints and alert the user if there are any discrepancies in their business model.

6 EXPERIMENTATION WITH CASE STUDIES

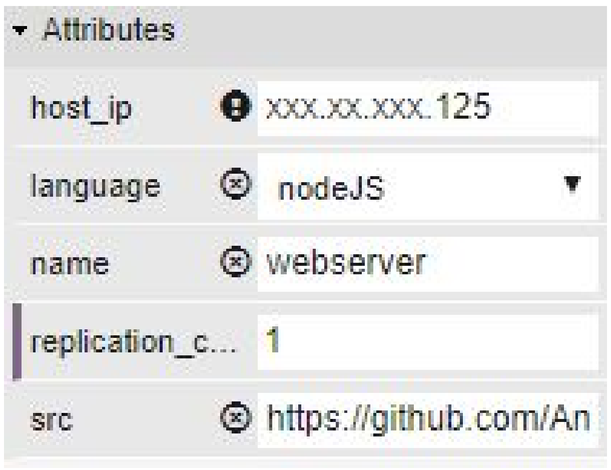
To synthesize the full blown architecture from the abstract business model, an application topology by using the WebGME modeler needs to be designed. WebGME offers all available node types in the left pane, and the user can define the properties as necessary in the right pane. Drag and Drop the desired node types in the editor pane, is all the business developer have to do to design the infrastructure for their application. The application components need to be linked to specify the order of execution. In this section, we will discuss three case studies as follow: (1) A single-tier website deployment, (2) A two-tier application deployment and (3) A data analytics application model deployment. We will compare the time and effort incurred in deploying two application use-cases using (a) manual efforts, where the operator must log into each machine and type the commands to install packages and deploy the applications, (b) manually creating Ansible scripts to deploy these applications, and (c) using CloudCAMP modeling.

6.1 Case Study 1: A single-tier website deployment

Figure 7 shows an application topology of simple website deployment use case. In the model scenario, the web application will be deployed on OpenStack VM with the Ubuntu16.04 operating system, and these have a 'hostedOn' relationship. The user defined model, which is a half-baked, i.e., incomplete model is shown in Fig. 7a. The source location of the website source code (e.g. *https://github.com/Anirban2404/simpleChatServer.git*) needs to be mentioned in the business model. The language of the of the source code(e.g. NodeJS) has to be defined along with the web server software(e.g., NodeJS, Apache Tomcat, etc.) as shown in Fig. 7b.



(a) The Sample website deployment model



host_ip	xxx.xx.xxx.125
language	nodeJS
name	webserver
replication_c...	1
src	https://github.com/An

(b) Attributes, which needs to be defined

Figure 7: The Sample Website Deployment model Design

We have captured the WebApplication and OpenStack dependency in the CloudCAMP.

Given, all the specifications in the business model, our CloudCAMP DSML first spawns a VM in the OpenStack because it is the end node of the 'hostedOn' connection. Then, the DSML generates the SQL script, and it fetches all the defined software dependencies from the knowledge base for the web server application type and the language, based on its os and version. A sample of the SQL script is shown below:

```
SELECT pkg.pkg_name FROM packages pkg,
swdependency dep
WHERE pkg.app_id = dep.id
AND pkg.apptype = <<LANGUAGE>>
AND pkg.sw_id IN (SELECT app_sw_id
FROM os_dependency WHERE os_id IN
(SELECT id FROM os_pkg_mgr
WHERE Concat(os_type, os_version) =
<<OS>>,<<VERSION>>))
```

Then, the WebGME plugin generates the Ansible-playbook with all the provisioned tasks by filling the templates based on the application component type. Moreover, in the manual effort, we need to configure the files, create the handlers to specify the deployment order in the desired host, which is also handled by CloudCAMP DSL. Our, NodeJS based DSL also execute the playbook and deploy the code in proper order for perfect deployment. Thus, we generate Ansible script, for deployment and configuration of software lifecycle operations on the target node(s) from the half-baked user model.

6.2 Case Study 2: A two-tier application deployment

Our use case involves a prototypical three-tier Linux, Apache, MySQL, and PHP (LAMP)-based application deployment. Figure 8 shows an application topology illustrating the modeling effort in CloudCAMP,[14] where the PHP-based web application needs to be 'HostedOn' on an OpenStack platform on a Ubuntu 16.04 VM, and the database application needs to be deployed on another OpenStack platform on a Ubuntu 14.04 VM, and these two tiers must have a 'ConnectsTo' relationship between them.

As described in section 5.2, we have captured the all the endpoints of applications in our metamodel and all endpoints of cloud providers in the CloudCAMP metamodel. All the attributes of define the functionalities of the node type.

[14]The relationships between the model elements are from the metamodel and are shown on the arcs joining the elements.

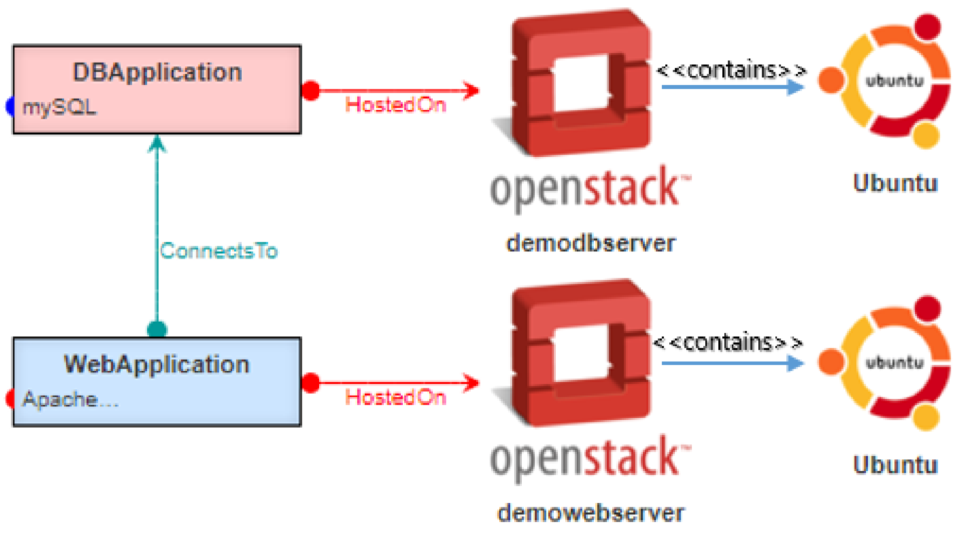


Figure 8: Sample LAMP Application Model

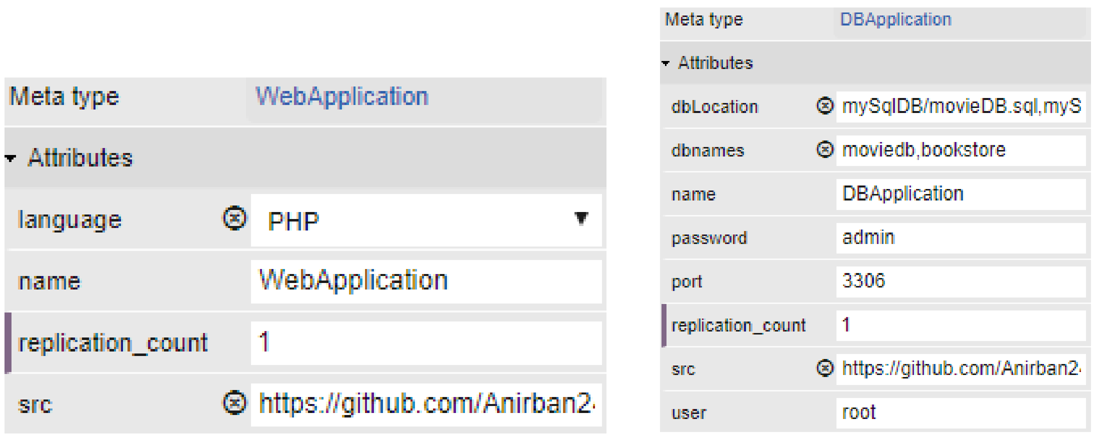
In addition to the structural model as shown, a use must also supply appropriate parameters to the different model elements. For instance, for the WebApplication node type, the language of the source code (e.g., PHP) has to be specified as shown in Figure 9a along with the web server software to be used (e.g., Apache). Likewise, for the DBApplication node type, attributes such as database name, location, port, user, password, etc. need to be specified as shown in Figure 9b along with the database system used (e.g., MySQL). The WebApplication component type connects to DBApplication component type, which is a MySQL database as a backend server.

Since we reverse engineer the applications, all endpoints (e.g., IP addresses) and all the constraints are predefined and specified in the model.

6.2.1 Qualitative Evaluation

The “correct-by-construction” and automation benefits of CloudCAMP are achieved as follows. Modeling errors are resolved at modeling time via constraint checking. Once the model is completely specified in CloudCAMP, it then generates Ansible scripts to deploy and trigger the execution of the application according to the steps outlined in Algorithm 1. For this case study, first, the VMs will be spawned in the specified OpenStack platforms based on the user-defined environment variables. The installation and execution ordering is dictated by the ’hostedOn’ connection.

Once the VMs are spawned, CloudCAMP queries the knowledge base used by the model transformation logic. The SQL script fetches all the defined software dependencies from the



Meta type	WebApplication
▾ Attributes	
language	PHP
name	WebApplication
replication_count	1
src	https://github.com/Anirban2

(a)

Meta type	DBApplication
▾ Attributes	
dbLocation	mySqlDB/movieDB.sql,myS
dbnames	moviedb,bookstore
name	DBApplication
password	admin
port	3306
replication_count	1
src	https://github.com/Anirban2
user	root

(b)

Figure 9: (a) shows the specifications related to WebApplication type and (b) shows the specifications related to DBApplication type

knowledge base for the web server application type and the specified language based on its OS and version. Similarly, the SQL query fetches all the software dependencies for database application type specific to MySQL RDBMS. Then, the model transformation algorithm reads the template files specific to the functionalities of the application component types and generates the complete TOSCA-compliant deployable IAC solutions based on Ansible specification, called Ansible-playbooks. Then, the IP address of the newly spawned VM is acquired from the destination node type OpenStack platforms and binds it to Ansible 'hosts' files for the 'HostedOn' source application type.

The WebApplication component type connects to the DBApplication component type based on the 'connectsTo' relationship in the business model. Moreover, CloudCAMP automatically infers from this relationship that the webserver must wait for the database server to start first. For those cases that are not constrained by the 'ConnectsTo' relationship, CloudCAMP ensures that deployment can be executed with maximal parallelism by leveraging the underlying generated Ansible scripts.

In contrast to CloudCAMP, in a fully manual effort, the user will need to configure the files, create the handlers to specify the deployment order in the desired host, log into each host where the application components are deployed and manually install the packages, configure the software packages and finally start the different components in the correct order. In the manually created Ansible script case, the user will first incur a significant learning curve for Ansible and must manually create the different playbooks. Thereafter we expect that despite improving automation via Ansible, the user will still incur trial-and-error, which is likely to be amplified for complex deployment scenarios.

In addition to the advantages of CloudCAMP highlighted above, assume the case of continuous delivery where a website application provider later wants to attach more Database servers. For CloudCAMP, it is just a matter of extending the existing business model with additional DBApplication node type and 'connectsTo' relationships from webserver to the database server. CloudCAMP will generate Ansible-specific IAC for the newly added component and executes it to deploy newly added component without hampering availability of the existing business application. Thus, CloudCAMP provides seamless integration of new application component types, which only needs to be linked to the present application component type. Since Ansible is idempotent, it always sets the same configuration in the target environment regardless of their current state.

6.2.2 Quantitative Evaluation based on a User Study

We also conducted a user study for case study 1 involving sixteen teams of three students each for an ongoing Cloud Computing course we are teaching. At the time of writing this paper, we were only able to measure both the time taken and efforts for a fully manual effort in deploying the scenario. At the time of the survey, students were being introduced to Ansible and they realize its strengths, and moreover, the working of CloudCAMP was also demonstrated in class although they have not yet used the tool.[15] The following questionnaire was created to conduct the survey. For each question, the students were asked to evaluate on a scale of 1–10 where 1 is easiest and 10 is hardest:

Table 1: Survey Questionnaire: For Q1–Q3, rate on a scale of (1-10), where 1 is easiest, 10 is hardest.

Num	Question
Q1	How easy is it to deploy PHPMySQL application manually?
Q2	How easy is it to deploy PHPMySQL using DevOps tool like Ansible?
Q3	How easy is it to deploy PHPMySQL using CloudCAMP?
Q4	How much time and effort did you require to deploy the application manually (in minutes)?
Q5	How much time and effort is required in deploying the application using DevOps tool like Ansible (in minutes)?
Q6	How much time and effort is required deploying the application using CloudCAMP (in minutes)?
Q7	How likely are you to use the CloudCAMP platform to deploy applications in future?

Responses to Q1 and Q3: Ease of use: As seen from Figure 10, the "ease of use" rating for the CloudCAMP platform is much higher compared to manual effort. Although the students have not yet deployed the application via CloudCAMP, they have seen the in-class demonstration of the tool. The median difficulty in manual effort is rated as 70%, while the median difficulty rating for CloudCAMP use is 30%. The visual drag and drop

[15] We expect to complete this user study during the semester and the full set of results will be available before the camera ready deadline.

environment helps the users to quickly design and deploy various scenarios of the business application topology in distributed systems.

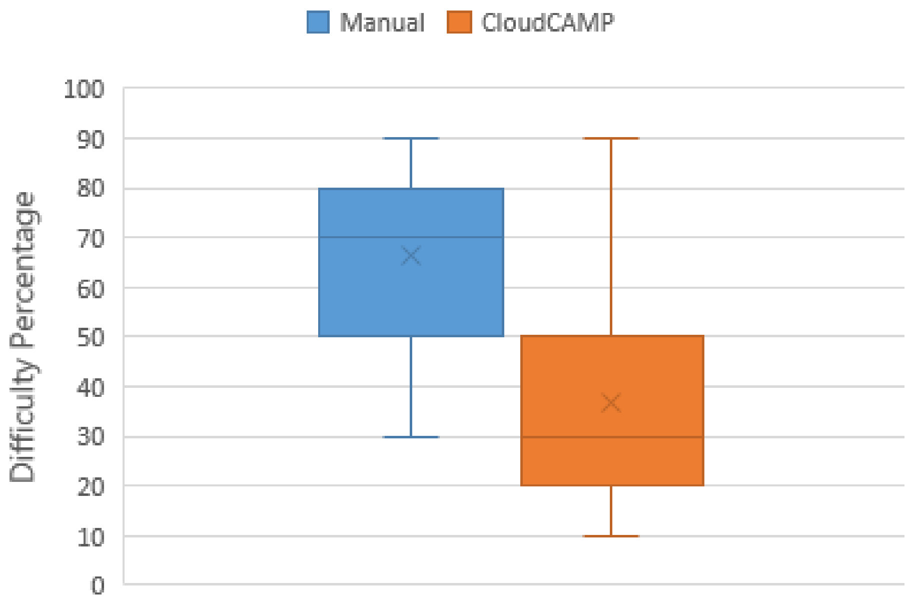


Figure 10: Comparing the difficulties to deploy the application in different approach in percentage scale

Responses to Q4 and Q6: Time to complete the whole deployment: The LAMP stack webservice deployment with the provided source code comprises installing and configuring PHP, Apache HTTP server, and MySQL RDBMS. The average time the students took to manually complete the entire deployment process is 171 minutes, whereas our rough estimates for students using the CloudCAMP-based topology creation and deployment will be only 15-20 minutes for the first time users.

Response to Q7: As shown in the Figure 11, 57.1% of the respondents agreed to use CloudCAMP tool to deploy cloud applications in future, whereas 35.7% are still unsure about using it. We expect the numbers for CloudCAMP to improve once the students actually use it in their class assignment.

Responses to Q2 and Q5: For these questions, we are still gathering data about the usability of the DevOps Ansible usability, which is the experiment the students are currently conducting as part of their second assignment.

Initial results from our user study strengthens our belief that the CloudCAMP platform that is designed to develop a business model without domain expertise will be a very resourceful tool for business application developers. It helps the application developer to build and deploy their application rapidly across multiple cloud providers. In future, more studies on more complicated scenarios are necessary to substantiate our claims.

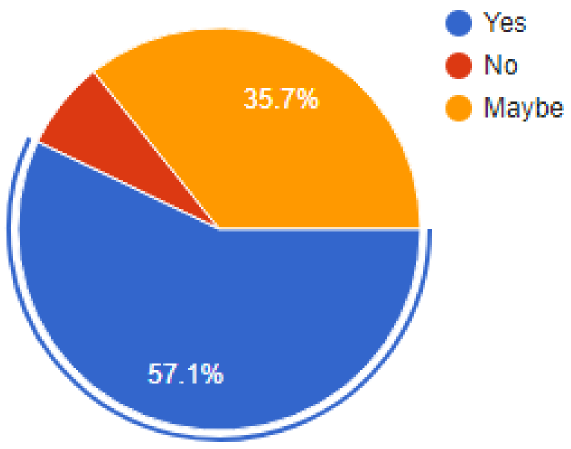


Figure 11: Likeliness of using CloudCAMP for future cloud application deployment

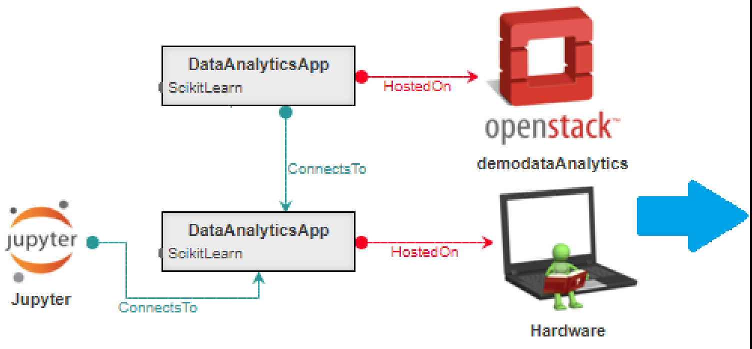
6.2.3 Extended Case Study 2: Application Components Migration for LAMP-based Web Service

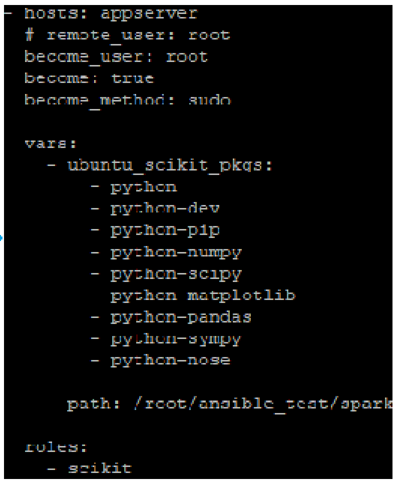
CloudCAMP platform also supports application component migration with ease for which we have two connection types 'deleteFrom' and 'migrateTo'. As described in scenario 4.3, suppose the developer wants to migrate the database application component from one machine to another machine which resides on different OpenStack platform. Using the 'deleteTo' connection type, the business application developer can specify from where (s)he wants to move the application components and attach a 'migrateTo' connection type to indicate the destination. The migrationType (stateless or stateful) is also needed to be selected. CloudCAMP generates a new Ansible tree structure based on the changed user specifications as described in the Algorithm 2. Accordingly, it will spawn a new VM with the new operating system and deploy the application with the required software dependencies, which are fetched from our knowledge base. Then, depending on the 'migrationType' specified it will set the migration strategy and thus move the DBApplication component.

Although we have not conducted a user study for this use case, the benefits of the automation accrued using CloudCAMP can easily be understood for this use case too since it is similar to adding a new component, however, in this case we use different relationships and constraints.

6.3 Case Study 3: An DataAnalytics Application model deployment

Figure 12 shows application topology of a simple DataAnalytics application based on big data processing engine. Here, for the case study scenario, we select scikit-learn machine learning library[16] as our engine. The DataAnalytics App contains scikit-learn in the nodes, and the users have to mention the source code of the application. The components of DataAnalyticsApp can communicate between each other as application demands. For example, one node can run all the data fetching and preprocessing steps and other node can run all the machine learning algorithms to make sense of the data. The user only has to drag and drop the predefined element to deploy their topology on a target system without writing a single line of code.





```
- hosts: appserver
  # remote_user: root
  become_user: root
  become: true
  become_method: sudo

  vars:
    - ubuntu_scikit_pkgs:
        - python
        - python-dev
        - python-pip
        - python-numpy
        - python-scipy
          python matplotlib
        - python-pandas
        - python-sympy
        - python-nose

      path: /root/ansible_test/spark

  roles:
    - scikit
```

Figure 12: Sample DataAnalytics Topology Model and Generated sample Ansible Code

As described before, the DSML generates the Ansible playbook which has all the defined software dependencies. The sample topology is then deployed on the source location as mentioned by the user, after installing all the dependencies. Now, for visualization of the analysis result, if the user wants to add Jupyter Notebook later, (s)he can do it just by connecting the Jupyter node with DataAnalyticsApp. The DSML will then install all the dependencies for Jupyter[17] Notebook and configure the node. Then, the user can access the Jupyter web application, which allows users to manipulate the code and visualizations for better analysis of data. Moreover, our tool is extensible, so adding new libraries with current metamodel is a feasible option, because of our modularized approach.

[16]http://scikit-learn.org/stable/
[17]https://jupyter.org/

As described with the case-studies, the users can create the business models using our metamodel, and the defined DSML can convert the half-baked user model to a full-blown Ansible model and deploy it on the desired machine without requiring detailed domain knowledge. The automation makes the deployment faster and error-free. It will not only save the time to write hundreds of lines of code, but it will also select proper software packages, proper package manager and proper configuration files for the application components. Since WebGME has based a web-based design environment, our CloudCAMP platform also supports collaboration and model versioning.

7 CONCLUSION

In this paper, we presented a model-based approach for an automated, deployment and management platform for cloud applications. It aids the business user to model their application at a higher level of abstraction closer to their domain, and deploy their code without requiring specific domain expertise and without writing a single line of code. It helps to save an enormous amount of time in configuring the system and running the actual business model. CloudCAMP can easily be integrated with existing cloud provider APIs.

Based on our literature survey, we find some gaps that need to be filled for automatic transformation of the business-relevant model to IAC solution. First, we need to define a metamodel for the business user, who can make the model without the need to be a domain expert. Then, the DSML will then transform the user model to TOSCA-compliant deployable model in an efficient and will also guarantee the correctness of the model based on defined constraints found by reverse engineering. The MODACLOUD and Aeolus project approach is a good solution to reach this goal, but in contrast to our approach of querying knowledge base, they use CSP solver to transform the business model. On the other hand, OpenTOSCA approach is TOSCA-compliant, and it fulfilled the node template by requirement and capability analysis, but it lacks validation of the model and robustness. We built the metamodel for the CloudCAMP using WebGME, which is a cloud-based toolkit for creating DSMLs. Through metamodels, we specified the modeling language for the cloud application domain. In defining the modeling language, we included all the syntactic and semantic information regarding the application and cloud domain, which needs to be realized to construct instruction-as-a-code deployable model.

By virtue of using WebGME to develop the CloudCAMP framework, its metamodel(s) and knowledge base is decoupled from the generative aspects. Thus, although we have demonstrated only Ansible-specific code generation from the business model, code generation for other tools such as Chef or Puppet requires plugging in a tool-specific model parser and generator. Likewise, both the metamodel(s) and the knowledge base are extensible. Our future work will involve improving the soundness and robustness of the models using CSP solvers. We will also add reflection features to our framework so that the dynamic changes happening at the system level will be reflected back into the design view level and incremental deployment artifacts can be generated and system changes effected.

CloudCAMP source code is available at *https://doc-vu.github.io/DeploymentAutomation*.

Acknowledgment

This work was supported in part by NEC Corporation, Kanagawa, Japan and NSF US Ignite CNS 1531079. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NEC or NSF.

Bibliography

[1] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[2] C. Vecchiola, S. Pandey, and R. Buyya, "High-performance cloud computing: A view of scientific applications," in *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on.* IEEE, 2009, pp. 4–16.

[3] A. Spark, "Apache spark: Lightning-fast cluster computing," 2016.

[4] J. Varia, "Cloud Architectures," http://jineshvaria.s3.amazonaws.com/public/cloudarchitectures-varia.pdf, 2008, amazon Web Services.

[5] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[6] Y. Barve, P. Patil, A. Bhattacharjee, and A. Gokhale, "Pads: Design and implementation of a cloud-based, immersive learning environment for distributed systems algorithms," *IEEE Transactions on Emerging Topics in Computing*, 2017.

[7] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "Tosca: portable automated deployment and management of cloud applications," in *Advanced Web Services.* Springer, 2014, pp. 527–549.

[8] S. Hoare, "A study of the state-of-the-art of paas interoperability," in *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering.* ACM, 2016, p. 7.

[9] K. Képes, U. Breitenbücher, and F. Leymann, "The sepade system: Packaging entire xaas layers for automatically deploying and managing applications," *month*, 2017.

[10] X. Li, Y. Li, T. Liu, J. Qiu, and F. Wang, "The method and tool of cost analysis for cloud computing," in *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on.* IEEE, 2009, pp. 93–100.

[11] B. Dougherty, J. White, and D. C. Schmidt, "Model-driven auto-scaling of green cloud computing infrastructure," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 371–378, 2012.

[12] K. El Maghraoui, A. Meghranjani, T. Eilam, M. Kalantar, and A. V. Konstantinou, "Model driven provisioning: Bridging the gap between declarative object models and procedural provisioning tools," in *Middleware 2006.* Springer, 2006, pp. 404–423.

[13] OASIS, "Topology and orchestration specification for cloud applications," http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf, 2013, oASIS Standard.

[14] ——, "TOSCA XML schema definition," http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/schemas/TOSCA-v1.0.xsd, 2013, oASIS.

[15] ——, "Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0," http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.html, 2013, oASIS.

[16] D. C. Schmidt, "Model-driven engineering," *COMPUTER-IEEE COMPUTER SOCIETY-*, vol. 39, no. 2, p. 25, 2006.

[17] M. Maroti, T. Kecskes, R. Kereskenyi, B. Broll, L. J. Peter Volgyesi, T. Levendoszky, and A. Ledeczi, "Next Generation (Meta)Modeling: Web- and Cloud-based Collaborative Tool Infrastructure," https://webgme.org/WebGMEWhitePaper.pdf.

[18] E. E. Co., "Emerson Network Power Study on DataCenter outage cost," http://www.emersonnetworkpower.com/en-US/About/NewsRoom/NewsReleases//Pages/Emerson-Network-Power-Study/-Says-Unplanned-Data-Center-Outages-Cost-/Companies-Nearly-9000-Per-Minute-.aspx, 2016.

[19] S. Nelson-Smith, *Test-Driven Infrastructure with Chef: Bring Behavior-Driven Development to Infrastructure as Code.* " O'Reilly Media, Inc.", 2013.

[20] M. Burgess *et al.*, "Cfengine: a site configuration engine," in *in USENIX Computing systems, Vol.* Citeseer, 1995.

[21] J. Loope, *Managing Infrastructure with Puppet.* " O'Reilly Media, Inc.", 2011.

[22] A. Brogi, J. Soldani, and P. Wang, "Tosca in a nutshell: Promises and perspectives," in *Service-Oriented and Cloud Computing.* Springer, 2014, pp. 171–186.

[23] J. Humble and J. Molesky, "Why enterprises must adopt devops to enable continuous delivery," *Cutter IT Journal*, vol. 24, no. 8, p. 6, 2011.

[24] J. Carrasco, J. Cubo, F. Durán, and E. Pimentel, "Bidimensional cross-cloud management with tosca and brooklyn," in *Cloud Computing (CLOUD), 2016 IEEE 9th International Conference on.* IEEE, 2016, pp. 951–955.

[25] C. Fehling, F. Leymann, R. Retter, D. Schumm, and W. Schupeck, "An architectural pattern language of cloud-based applications," in *Proceedings of the 18th Conference on Pattern Languages of Programs.* ACM, 2011, p. 2.

[26] T. Eilam, M. Elder, A. V. Konstantinou, and E. Snible, "Pattern-based composite application deployment," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on.* IEEE, 2011, pp. 217–224.

[27] H. Lu, M. Shtern, B. Simmons, M. Smit, and M. Litoiu, "Pattern-based deployment service for next generation clouds," in *Services (SERVICES), 2013 IEEE Ninth World Congress on.* IEEE, 2013, pp. 464–471.

[28] A. Homer, J. Sharp, L. Brader, M. Narumoto, and T. Swanson, "Cloud design patterns: Prescriptive architecture guidance for cloud applications," 2014.

[29] D. Ardagna, E. Di Nitto, G. Casale, D. Petcu, P. Mohagheghi, S. Mosser, P. Matthews, A. Gericke, C. Ballagny, F. D'Andria *et al.*, "Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds," in *Proceedings of the 4th International Workshop on Modeling in Software Engineering.* IEEE Press, 2012, pp. 50–56.

[30] E. Di Nitto, M. A. A. da Silva, D. Ardagna, G. Casale, C. D. Craciun, N. Ferry, V. Muntes, and A. Solberg, "Supporting the development and operation of multi-cloud applications: The modaclouds approach," in *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on.* IEEE, 2013, pp. 417–423.

[31] S. Narain, G. Levin, S. Malik, and V. Kaul, "Declarative infrastructure configuration synthesis and debugging," *Journal of Network and Systems Management*, vol. 16, no. 3, pp. 235–258, 2008.

[32] E. Torlak and D. Jackson, "Kodkod: A relational model finder," in *Tools and Algorithms for the Construction and Analysis of Systems.* Springer, 2007, pp. 632–647.

[33] J. Fischer, R. Majumdar, and S. Esmaeilsabzali, "Engage: a deployment management system," in *ACM SIGPLAN Notices*, vol. 47, no. 6. ACM, 2012, pp. 263–274.

[34] R. Di Cosmo, A. Eiche, J. Mauro, S. Zacchiroli, G. Zavattaro, and J. Zwolakowski, "Automatic deployment of services in the cloud with aeolus blender," in *Service-Oriented Computing.* Springer, 2015, pp. 397–411.

[35] R. Di Cosmo, M. Lienhardt, R. Treinen, S. Zacchiroli, J. Zwolakowski, A. Eiche, and A. Agahi, "Automated synthesis and deployment of cloud applications," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering.* ACM, 2014, pp. 211–222.

[36] T. A. Lascu, J. Mauro, and G. Zavattaro, "A planning tool supporting the deployment of cloud applications," in *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on.* IEEE, 2013, pp. 213–220.

[37] P. Hirmer, U. Breitenbücher, T. Binz, F. Leymann *et al.*, "Automatic topology completion of tosca-based cloud applications." in *GI-Jahrestagung*, 2014, pp. 247–258.

[38] U. Breitenbucher, T. Binz, K. Képes, O. Kopp, F. Leymann, and J. Wettinger, "Combining declarative and imperative cloud application provisioning based on tosca," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on.* IEEE, 2014, pp. 87–96.

[39] I. Giannakopoulos, N. Papailiou, C. Mantas, I. Konstantinou, D. Tsoumakos, and N. Koziris, "Celar: automated application elasticity platform," in *Big Data (Big Data), 2014 IEEE International Conference on.* IEEE, 2014, pp. 23–25.

[40] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[41] A. MySQL, “Mysql: the world’s most popular open source database,” 2005.

[42] E. Amazon, “Amazon elastic compute cloud (amazon ec2),” *Amazon Elastic Compute Cloud (Amazon EC2)*, 2010.

[43] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and J. Wettinger, “Integrated cloud application provisioning: interconnecting service-centric and script-centric management technologies,” in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2013, pp. 130–148.

[44] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and M. Wieland, “Context-aware cloud application management.” in *CLOSER*, 2014, pp. 499–509.