CloudCAMP: Automating the Deployment and Management of Cloud Services

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*—Users of cloud platforms often must expend significant manual efforts in the deployment and orchestration of their services on cloud platforms due primarily to having to deal with the high variabilities in the configuration options for virtualized environment setup and meeting the software dependencies for each service. Despite the emergence of many DevOps cloud automation and orchestration tools, users must still rely on specifying low-level scripting details for service deployment and management. Using these tools required domain expertise along with a steep learning curve. To address these challenges in a tool-and-technology agnostic manner, which helps promote interoperability and portability of services hosted across cloud platforms, we present initial ideas on a GUI based cloud automation and orchestration framework called CloudCAMP. CloudCAMP uses model-driven engineering techniques to provide users with intuitive and higher-level modeling abstractions that preclude the need to specify all the low-level details. CloudCAMP's generative capabilities leverage a built-in knowledge-base to automate the synthesis of Infrastructure-as-Code (IAC) solution that subsequently can be used to deploy and orchestrate services in the cloud. Preliminary results from a small user study are presented in the paper.**

***Keywords*—cloud services, automation, orchestration, domain-specific modeling, knowledge base**

I. Introduction

A. Emerging Trends and Challenges

Service deployment and management in cloud platforms demand significant manual efforts. Users of cloud platforms have to deal with the high variabilities in the configuration options for virtualized environment setup and meeting the software dependencies for each service. Deployment, management and continuous delivery of composite applications pose inherent challenges because of their complexity. Despite the emergence of many DevOps cloud automation and orchestration tools, users must still rely on specifying low-level scripting details to implement their service deployment and management strategies. Using these tools incurs a steep learning curve, which is further exacerbated due to the required domain expertise to use these tools. Thus, self-service application deployment, orchestration, and management platform are desired for enterprises to speed up time-to-market for their services while ensuring their reliable deployment and management. The self-service platform needs to abstract all the low-level details of deployment and management, especially in the cloud environment.

B. Solution Approach: CloudCAMP

To address these challenges, we present preliminary ideas on a model-driven and scalable, rapid provisioning framework called CloudCAMP that significantly reduces the burden on the users. CloudCAMP complies with the OASIS standard *Topology and Orchestration Specification for Cloud Applications (TOSCA)* specification [1].

CloudCAMP incorporates domain-specific modeling so that the specifications and dependencies of clouds and applications architecture are specified at an intuitive, higher level of abstraction without the need for specifying all the low-level domain details as shown in Figure.1. The extensible and reusable knowledge base maintained by CloudCAMP helps to complete the partial specifications and generate deployable Infrastructure-as-Code (IAC), which can be handled by the existing tools to provision the services components during deployment and management phase. We validate our approach by a prototypical application model and present results from a preliminary user study to evaluate its relevance.

C. Comparison with Related Research

We have studied the state-of-art of deployment and management abstraction in the literature of cloud automation and orchestration. Cloud orchestration tools like Apache Scalr (https://scalr-wiki.atlassian.net/wiki/display/docs/Apache), Cloudify (http://getcloudify.org/) provide sophisticated techniques to deploy, manage and monitor applications on cloud providers. However, the users need to define the complete and correct deployable model with all the functionalities, features and order to deploy services using these tools. The correctness of deployable script cannot be verified in pre-deployment phase.

Several efforts come close to the CloudCAMP idea. Several pattern-based approaches [2], [3] are proposed to alleviate the complexity of designing the deployment of the application, however unlike our approach, they lack verification and extensibility, and they did not consider distributed micro-service architecture based applications. In other approaches, the requirements are predefined as constraints on the configuration by the developers. Engage [4] deploys and manages the application from a partial specification using a constraint-based algorithm. Aeolus Blender [5] also use the configuration optimizer Zephyrus [6], the ad-hoc planner Metis [7], and deployment engine Arnomic to deploy a service from partial

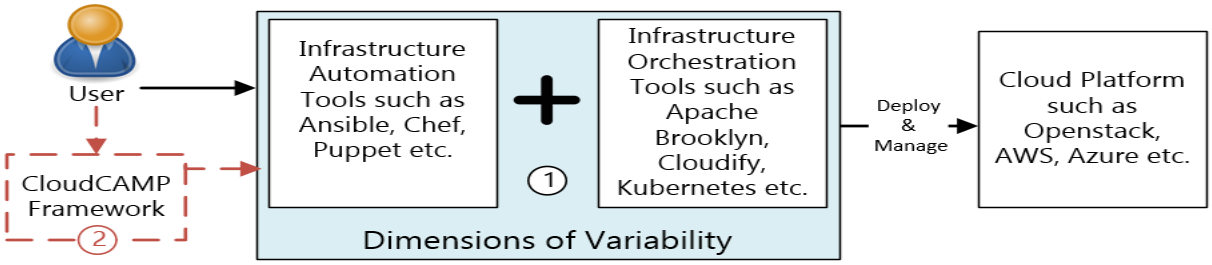

Fig. 1: Box 1 (blue color) depicts the responsibilities of service management team for defining the low-level details in the existing automation tools as scripts to automate the service management and orchestration tools can provision the infrastructure for service components and execute them on heterogeneous cloud environments. Box 2 (red color) depicts the contributions of this paper which introduces a self-service framework to automate the generation of whole IAC design solutions.

description. In contrast to the use of knowledge base in CloudCAMP, these efforts require lots of domain expertise to define the model specifications as constraints for the CSP solver. CSP solvers, however, can take significant time to transform the partial model to concrete deployable model.

D. Paper Organization

The rest of the paper is organized as follows: Section II presents the design of CloudCAMP alluding to the challenges faced and solution approach; Section III presents preliminary results from a user study; and finally, Section IV concludes the paper alluding to future directions.

II. CloudCAMP Approach

This section delves into the design of CloudCAMP alluding to the key challenges and how its design helps resolve these.

A. Challenges Faced in Cloud-based Service Provisioning

Although IAC helps mask the heterogeneity stemming from the differences in the cloud platforms and their resource types, two key challenges remain unresolved as described below.

Challenge 1: Variability and Complexities in Automated Deployment of Services: Although IAC decouples the user from directly accessing the cloud platform APIs, service deployment and continuous delivery workflow include manually creating the IAC logic and configuring the provisioning environments to perform a series of automation tasks. Since each of the orchestration and automation frameworks use their language with its syntax, semantics and formatting rules, this requires users to write IAC code that is very specific to the orchestration and automation tool being used. Consequently, in addition to dealing with the variability in the framework space and learning curve incurred, users also experience a *framework lock-in*. Existing frameworks also require elaborate specification of service topologies comprising requirements, functionalities, dependencies and relationships of the components. The variability in application package dependencies makes the script writing further challenging. For instance, depending on the technology used such as MySQL versus PostgreSQL or PHP versus Node.js, the script must include the appropriate drivers. Moreover, additional dimensions of variability (i.e., addressing application's compatibility and cloud providers' incompatible APIs) as depicted in Box 1 of Figure 1 further amplifies the manual effort which is daunting, tedious and error-prone. Finally, existing approaches do not account for pre-deployment validation to check if the end-user requirements and software dependencies are met.

Challenge 2: Complexities in Automated Migration of Service Components across Providers: Service components may need to migrate between cloud providers to derive the benefit of best quality along with optimal pricing [8]. Migrating application components in the heterogeneous cloud environment requires rewriting different plans for different cloud providers, and for each deployment of new application components [9]. Defining the configurations and the scripts to install/uninstall specific software packages for the components is time-consuming and can even lead to service unavailability or erroneous deployment. Finally, the script-centric approach does not offer pre-migration validation [10].

B. CloudCAMP Approach

We address these challenges in CloudCAMP as follows:

1) *Domain-specific Modeling:* The complexity of infrastructure design and deployment is abstracted and restricted to just the most business-relevant model.
2) *Generative techniques:* We improve user productivity by reducing both the manual effort and need for substantial domain expertise.
3) *Portability:* Our approach is TOSCA-compliant so that all the models are portable, vendor-neutral and interoperable.

We have designed CloudCAMP's cloud-based service provisioning approach in the form of a workflow as depicted in Figure 2. The different actors of CloudCAMP architecture are described below.

1) *Abstraction of Business Model*: An application model is comprised of different application component types. The user has to select appropriate application component types from the CloudCAMP application pane and needs to specify the attributes to bind it with the cloud provider and the operating systems, on which they want to deploy the application components. The domain-specific modeling language handles all low-level details of deployment and management and abstracts it from the business users.
2) *Configurator*: This actor is responsible for transforming each user specified abstract application component to a

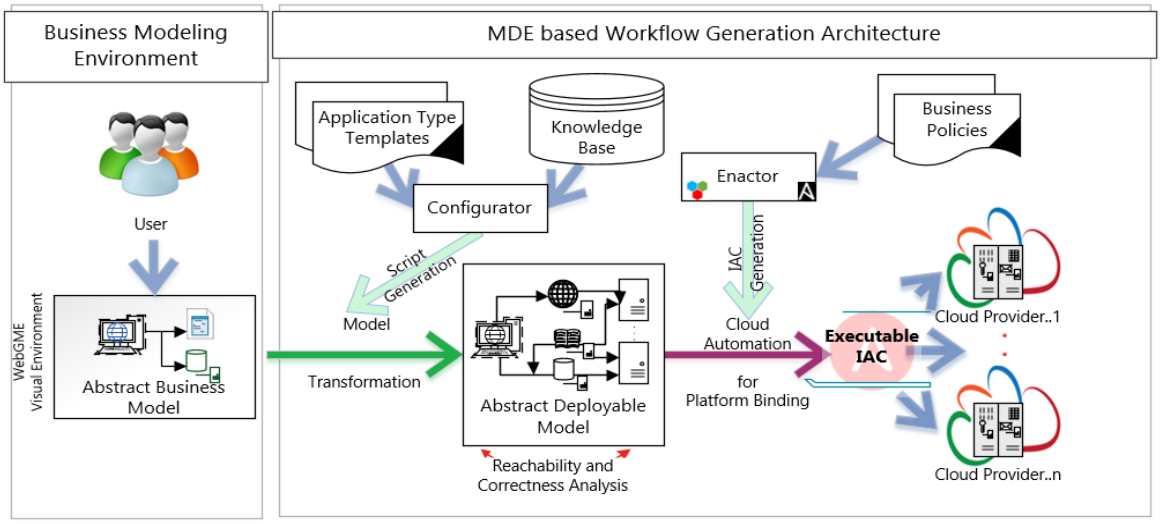

Fig. 2: The CloudCAMP Workflow

cloud automation task (e.g., Ansible-specific). Configurator realizes a user-defined abstract description of a cloud application model, and then it maps the application components with the operating system, and query the knowledge base to find the software dependency tree. Using the software dependency tree, it generates full 'correct-by-construction' Ansible-specific code from the application type template.

3) *Enactor*: It generates the infrastructure design workflow of IAC solutions by integrating the generated automation code with the business rules and cloud infrastructure specifications. The connection types between the application components are defined by the users. There are four types of connections: 'hostedOn', 'connectsTo', 'deleteFrom', 'migrateTo'. The details of the connection types and their role is described in [11]. The orchestration tool executes all the automation tasks based on the connection types to deploy and run the business application components in proper order.
4) *Knowledge Base:* A knowledge base aids to generate full automation code from the partially specified deployment models. We predefine the software dependencies for application type in a relational table with a key-value pair. All the software packages needed for a particular application component are defined in the tables along with their dependency on the operating system and its version. The application developer needs to populate the tables with all software dependencies for including the new application component type in the CloudCAMP.

C. CloudCAMP Design Implementation

We have used the WebGME MDE framework (www.webgme.org) to define the metamodel for the CloudCAMP Domain-specific Modeling Language (DSML).[1] WebGME is a cloud-based framework that offers an environment for DSML developers to define their language and create model parsers that can serve as generators of code artifacts. The CloudCAMP platform comprises of three services: (a) the WebGME modeling UI, and orchestration and automation frameworks, as one service, (b) the WebGME modeling details are stored in a MongoDB database, and (c) the knowledge base in a MySQL database.

[1]Due to space constraints, we do not describe the DSML design. The interested reader can find these details in [11].

III. Validation via User Study

This section describes a LAMP based application scenario and initial results from a simple user study we conducted as part of a cloud computing course taught by us.

Use Case: A LAMP stack deployment: We conducted a small user study in a Cloud Computing course for case study involving sixteen teams of three students each. We goal of the user study was to measure both the time taken and efforts (a) for a fully manual effort, (b) for writing scripts in Ansible and executing these manually and (c) use CloudCAMP framework to deploy the scenario.

Our use case handed out to students of our cloud computing class involved a prototypical three-tier Linux, Apache, MySQL, and PHP (LAMP)-based microservice architecture deployment. Figure 3 shows the application topology illustrating the modeling effort in CloudCAMP, where the PHP-based web application needs to be 'HostedOn' on OpenStack platform on Ubuntu 16.04 VM, and the database service will be deployed on another OpenStack platform on Ubuntu 14.04 VM, and these two tiers must have 'ConnectsTo' relationship between them.

To test the usefulness of the CloudCAMP for deployment of cloud application, the students were first asked to manually install all the software and dependencies to deploy the LAMP web services by spawning OpenStack VMs. As a second step, the students were asked to use the CloudCAMP platform to create the same deployment topology and specify the

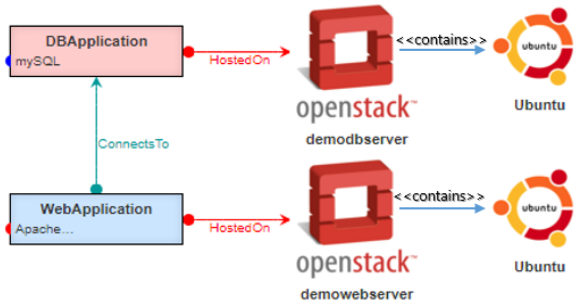

Fig. 3: Sample LAMP Application Model

attributes, and then run the code generation plugin to deploy the intended application on the desired cloud platform.

Hypothesis: The LAMP stack web service deployment with the provided source code requires installing and configuring PHP, Apache HTTP server, and MySQL RDBMS. We surmise that in a fully manual effort, the users 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 manual scripting case, the user will first incur a significant learning curve for Automation and Orchestration tools. Thereafter we expect that despite improving automation via these tools, the user will still incur trial-and-error, which is likely to be amplified for complex deployment scenarios and decrease the productivity.

Qualitative Benefits using CloudCAMP: 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 the CloudCAMP framework, then CloudCAMP generates IAC solution for the application components and deploys it via backend DevOps tools. 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 web service must wait for the database service to start first. For those cases that are not constrained by 'ConnectsTo' relationship, CloudCAMP ensures that deployment can be executed with maximal parallelism by leveraging the underlying generated scripts.

Quantitative Evaluation based on a User Study: The average time the students took to complete the entire deployment process manually is 171 minutes, and the average time is 516 minutes to script and debug Ansible code correctly, 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. The average line of code written for the deployment process is 315 lines.

IV. Conclusions

This paper presented preliminary ideas on a model-driven engineering and generative programming approach to automated deployment and management of cloud applications. CloudCAMP's use of domain-specific modeling is meant to aid the application deployer in modeling the service provisioning at a higher level of abstraction, and deploy its code without requiring significant domain expertise, minimal modeling effort and no low-level scripting. The generative capabilities help in the automated synthesis of infrastructure-as-code scripts that then can be used by existing DevOps tools.

This paper presented only preliminary ideas and ongoing work. A substantial additional work is needed to make this technology mature and adopted by practitioners. Our ongoing work is addressing many of these unresolved challenges. The technology's capabilities are being made available in open source at https://doc-vu.github.io/DeploymentAutomation/. Additionally, substantial additional user studies are necessary and our future efforts will focus on addressing this unresolved issue.

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 views of NEC or NSF.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] A. Bhattacharjee, "Mde-based automated provisioning and management of cloud applications."

[11] A. Bhattacharjee, Y. Barve, A. Gokhale, and T. Kuroda, "CloudCAMP: A Model-driven Generative Approach for Automating Cloud Application Deployment and Management," Vanderbilt University, Nashville, TN, USA, Tech. Rep. ISIS-17-105, Sep. 2017.