Simulating Live Cloud Adaptations Prior to a Production Deployment using a Models at Runtime Approach

Johannes Erbel, Alexander Trautsch and Jens Grabowski
University of Goettingen, Institute of Computer Science, Goldschmidtstraße 7, Goettingen, Germany

Keywords: Cloud, Simulation, Runtime Model, Adaptation, DevOps.

Abstract: The utilization of distributed resources, especially in the cloud, has become a best practice in research and industry. However, orchestrating and adapting running cloud infrastructures and applications is still a tedious and error-prone task. Especially live adaptation changes need to be well tested before they can be applied to production systems. Meanwhile, a multitude of approaches exist that support the development of cloud applications, granting developers a lot of insight on possible issues. Nonetheless, not all issues can be discovered without performing an actual deployment. In this paper, we propose a model-driven concept that allows developers to assemble, test, and simulate the deployment and adaptation of cloud compositions without affecting the production system. In our concept, we reflect the production system in a runtime model and simulate all adaptive changes on a locally deployed duplicate of the model. We show the feasibility of the approach by performing a case study which simulates a reconfiguration of a computation cluster deployment. Using the presented approach, developers can easily assess how the planned adaptive steps and the execution of configuration management scripts affect the running system resulting in an early detection of deployment issues.

1 INTRODUCTION

Due to the rise of scalable, elastic, and on demand resources many different tools emerged allowing the description infrastructure and application deployments using software artifacts (Brikman, 2019). As a result, the once individual teams, responsible for the development of the software (*Dev*) and the management of hardware (*Ops*), merged into one known as *DevOps* (Humble and Molesky, 2011). Even though many tools got developed to support DevOps in provisioning and configuring infrastructure, an increased knowledge is required as systems get more complex due to the huge amount of different resources that need to be managed. Therefore, developing, deploying, and orchestrating large and dynamic cloud infrastructures remains a difficult and error-prone task. As a result changes made to the production system need to be well tested and inspected to assess the impact of the change. Hereby, not all issues can be discovered without performing an actual deployment requiring a test environment, which can be difficult to set up and maintain (Guerriero et al., 2019).

In this paper, we present a model-driven concept that allows developers to assemble, test, and assess the impact of cloud adaptations and deployments without affecting the production system using a *runtime model*. This runtime model provides an abstract representation of the system and maintains a direct connection to it. The use of runtime models can be found in many approaches as they allow to easily design, validate, and manage cloud applications (Korte et al., 2018; Achilleos et al., 2019; Ferry et al., 2018). Similar to the concept presented in (Ferry et al., 2015), we reflect the production system in a runtime model and duplicate it for testing purposes. In our approach, we further annotate each individual resource in the duplicated runtime model with different simulation details. Additionally, we reduce the required infrastructure to a minimum so that a local and offline environment is sufficient to develop adaptive steps for distributed systems. Furthermore, this enables the user to investigate the extent to which the developed changes cope with scaling rules attached to the runtime model and thus the production cloud deployment. Our approach focuses on the simulation and reflection of *Infrastructure as a Service* (IaaS) clouds providing full access to infrastructure comprising compute, storage, and network resources. We demonstrate the feasibility of the approach by per-

Erbel, J., Trautsch, A. and Grabowski, J.
Simulating Live Cloud Adaptations Prior to a Production Deployment using a Models at Runtime Approach.
DOI: 10.5220/0010552003350343
In *Proceedings of the 11th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2021)*, pages 335-343
ISBN: 978-989-758-528-9

forming a case study in which we duplicate the production runtime model of a deployed computation cluster and extend its capabilities with additional features offline. We show, that the approach helps developers to assess how the planned adaptive steps and the execution of configuration management scripts affect relevant parts of the cloud deployment. This results in an early detection of deployment and performance issues using an abstract model of the system.

The remainder of this paper is structured as follows. Section 2 provides the papers foundations. Section 3 discusses related work. Section 4 presents our simulation approach. In Section 5, we describe the execution of our case study and discuss it in Section 6. Finally, in Section 7 an overall conclusion is given as well as an outlook into future work.

2 FOUNDATIONS

Cloud Computing is an umbrella term, usually describing a service in which consumers can dynamically rent virtualized resources from a provider pooling large amounts of resources together (Mell and Grance, 2011). This in turn creates an illusion of infinite resources available to the consumer that can be scaled up and down on demand (Armbrust et al., 2009). Cloud orchestration refers to the creation, manipulation and management of rented cloud resources. This includes not only the infrastructure, but also applications deployed on top of them (Liu et al., 2011). To reach this goal the concept of *Infrastructure as Code* (IAC) was introduced, which allows to describe infrastructure via software artifacts. IAC tools are utilized to provision infrastructural resources and manage the configuration and deployment of applications on top of it. The latter is usually done either via *configuration management* performing a sequence of configuration steps or *containerization* providing access to templated resource configurations (Wurster et al., 2020). Configuration management directly affects the state of the system by interpreting management scripts. When executed, these scripts perform the required actions to bring the resource to configure in to the described state. Often these tools provide *idempotence* mechanisms which ensure that even when an operation is executed multiple times the result stays the same. Containerization tools, like docker, on the other hand compress all the information required within an isolated part of the compute resource. A generic composition of a cloud deployment and its accompanying artifacts is depicted in Figure 1. A Compute node represents, e.g., a *Virtual Machine* (VM) or container node which resides on the infrastructure layer. On this node a Component node is deployed. This node describes, e.g., a running service. Finally, an Application node groups up related component instances. Each of these elements, possesses different attributes. For example the Component is linked to a Configuration Management script. The Compute node on the other hand is of a set Size and represents a defined Image, e.g., an ubuntu image for a VM or a docker container image hosting already deployed components.

Models at Runtime (M@R) is a subcategory of *Model Driven Engineering* (MDE) that promotes the utilization of models as a central artifact of software or its development. In M@R, however, these models are not only used as a static representation of a system, but rather as a live representation of it. To understand the concept of M@R, the term *model* and its relationship to a *metamodel* is fundamental. In the following, we refer to the term model as defined by (Kühne, 2006): “A model is an abstraction of a (real or language based) system allowing predictions or inferences to be made”. Each “model is an instance of a metamodel” (OMG, 2011) whereby the metamodel specifies elements, e.g., entities, resources, links and attributes, that can be used to create a model. In general, a metamodel is a model of models (OMG, 2003) forming a language of models (Favre, 2004). The formality introduced by a metamodel allows to validate its instances to check whether its “language” is correctly used. To allow for such a validation developers can infuse their metamodel with constraints that need to hold using declarative languages for rules and expressions like the *Object Constraint Language* (OCL) (OMG, 2016). This in turn supports an early detection of errors in a model. Designing cloud deployments within a model is a common approach as it grants an abstracted view on the overall system. Compared to typical models, a *runtime model* additional possess a causal connection to the abstracted system (Bencomo et al., 2013; Szvetits and Zdun, 2016). Hereby, *effector* components propagate changes in the runtime model directly to the system and vice versa. Using this strong connection to the system, a runtime model forms a monitorable runtime state of the system that allows to control and reason about it (Blair et al., 2009).

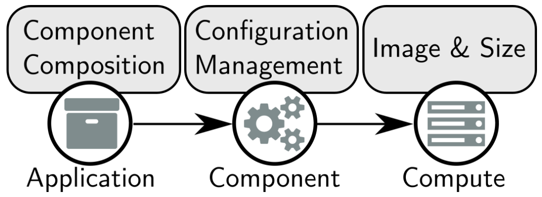

Figure 1: Generic cloud deployment stack.

3 RELATED WORK

Creating a domain specific language is a common approach to reduce the complexity of designing cloud deployments. Some of these languages support the utilization of a runtime model like CAMEL (Achilleos et al., 2019) and CloudML (Ferry et al., 2018). Also, approaches around the *Open Cloud Computing Interface* (OCCI) cloud standard exist which cover with runtime model functionalities (Zalila et al., 2017; Korte et al., 2018). Furthermore, an extension for OCCI is described by (Ahmed-Nacer et al., 2020) which is build around CloudSim (Calheiros et al., 2011). Among others this simulation allows to model data centers and simulate resource utilization and pricing strategies for cloud deployments.

Still, testing IAC and configuration management scripts is challenging because of missing information in form of a real world scenario, e.g., the current state of the system to be configured and possible state dependencies. A recent mapping study (Rahman et al., 2019) found that most research concerning infrastructure as code is concerned with frameworks and tools to support developers. In (Hanappi et al., 2016) an automated model-based test framework is proposed to determine whether a system configuration specification converges to a desired state. In contrast to their work our concept not only encompasses testing but also allows practitioners a high level overview and direct interactions for debugging purposes via a local deployment.

The approach most similar to our concept is presented in (Ferry et al., 2015), which describes the utilization of a models at runtime test environment next to a production one to continuously deploy a multi cloud system. In contrast to their work we consider the local replication of a production cloud deployment. Furthermore, our approach enables the developer to replay workloads on the local duplicate to assess the impact of planned adaptations to the production environment.

4 APPROACH

To ease the replication of production environments and to easily assess the impact of adaptation changes, we utilize a local runtime model duplicate as shown in Figure 2. We separate the Production Environment ① from the Local Environment ② and use the Simulation Configurator ③ to create a duplicate version of the production runtime model to locally assess the impact of adaptation changes.

The Production Environment ① consists of a Runtime Model (Prod) that is causally connected over a dedicated effector to a Cloud system. Within this model all provisioned infrastructure resources and deployed applications are present. This runtime model is used by the DevOps to operate the system which is an approach that can be often found in related work (Achilleos et al., 2019; Korte et al., 2018). One of the benefits of this model is that the DevOps have access to an abstracted representation of the system that can be easily adjusted to adapt to the needs of changing requirements.

The Local Environment ② is used to adjust, develop, and simulate desired adaptive actions, instead of applying these live adaptation changes directly. Within this environment a Runtime Model (Loc) is maintained that is causally connected to a local environment, e.g., on a local Workstation or a server instance. Hereby, the effector accompanying the runtime model implements the simulation behaviour providing a *simulation environment* in which, e.g., configuration management scripts can be applied to locally spawned compute resources. Depending on the used environment, the developer has access to different amounts of computing resources and therefore can simulate the deployment on different levels of detail.

The Simulation Configurator ③ configures the runtime model for two successive simulation appliances. A *deployment simulation* and an *orchestration simulation*. For the deployment simulation, we du-

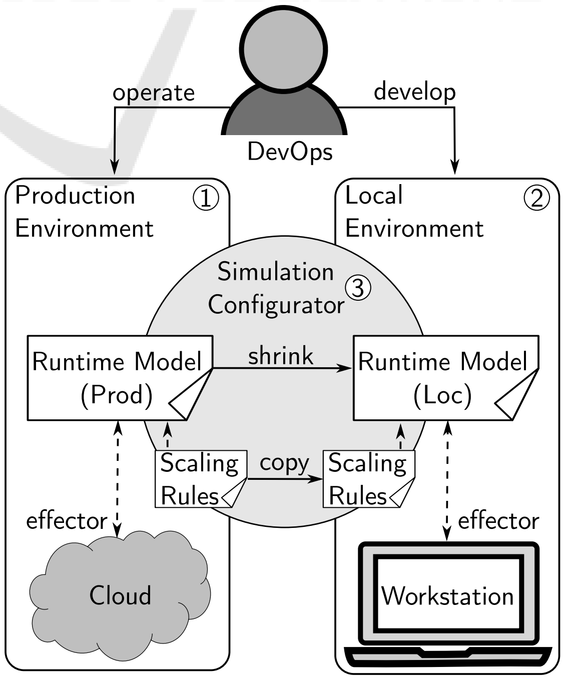

Figure 2: Overview of utilized environments.

plicate and shrink the Runtime Model (Prod) from the Production Environment into the Runtime Model (Loc) so that it fits into the Local Environment. Hereby, the transformation prunes the runtime model in terms of assigned resources so that parts of the deployment are actually deployed in a simulation environment. The remaining parts of the model are simulated on a *model-based level* only performing state transitions using a stub implementation of the elements state machine. To simulate runtime behaviour, we perform an orchestration simulation. For this simulation, we create a copy of the Scaling Rules applied to the Production Environment to apply them on our local Runtime Model (LOC). This allows to investigate the extent to which developed changes cope with the utilized orchestration processes. Hereby, the runtime model supports the investigation of runtime states being changed by the orchestration process while the simulation environment, e.g., allows to define automated tests. Later on, the DevOps can propagate the offline developed cloud application components, states, and scaling rules to the Production Environment.

4.1 Deployment Simulation

To locally work on cloud deployments, our approach combines an actual local provisioning of resources of interest while simulating the rest of the deployment. Therefore, we reduce the size of the production environments resources for our local replication. We aim to mimic as many resources as possible without breaking the limit of resources available for the development of new cloud deployment states. Additionally, we allow the developer to select different simulation levels for individual elements of the current cloud deployment. Depending on the chosen simulation level, the developer can work on new configuration management scripts and test actual deployments. To ease this process, we perform a transformation on the runtime model to investigate, such as the one of the production environment. We use this transformation to shrink the size of selected resources to fit to the size of the new environment and add annotations to specify simulation behaviour. This process is depicted in Figure 3 and described in the following.

As a first step in the deployment simulation process, we transform and duplicate the Runtime Model (Prod) representing the production environment into a Runtime Model (Loc) used for the local deployment. Within this transformation both models are instances of the same metamodel and therefore refer to the same language to describe the cloud deployment. During the transformation a *Simulation Tag* is added to each resource. This simulation tag serves as an indicator to be recognized by the effector of the runtime model. This effector ensures the causal connection to the system. While the effector of the production runtime model forwards requests to an actual cloud, the effector of the local runtime model handles the simulation behaviour. The transformation automatically adds the simulation tag with a default simulation level of zero which describes that only management operations should be simulated on a model-based level. Furthermore, this simulation level includes the simulation of lifecycle actions. Among others these actions trigger state changes on a resource. Such actions, e.g., start or stop a VM describing a transition from an active to inactive state of the compute node representing it.

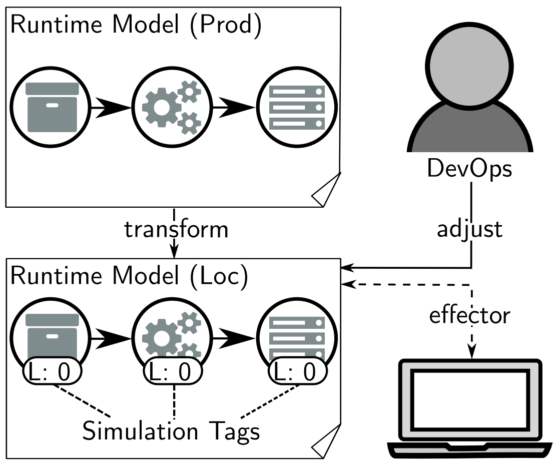

Figure 3: Runtime model simulation annotation.

After the transformation, the DevOps can adjust the transformed Runtime Model (Loc) with desired simulation levels (L) and specializations. Therefore, the automatically added tag can be infused with additional information that can be used by the effector handling the simulation logic. A subset of used SimulationTag specializations is shown in Figure 4. In this figure, the simulation.level attribute is shown with a default value of zero that can be increased by the developer to set the desired simulation level of each resource in the model. As different simulation levels for different entity types may require different attributes, we specify a specialization to be used for individual resource types.

The ComputeSim is used for compute resources like VM or container instances. Here, a simulation.level greater than zero is used to actually provision the compute resource on the local environment. To choose the kind of virtualization to be used for the provisioning, the tag introduces multiple attributes. Among others the virtualization.type and compute.image can be set, allowing the developer to define the desired configuration of the compute instance. Depending on the effort made by the developer, exact images may be extracted from the production environment and assigned to the simulation tag. Herewith, all information of the environment is replicated granting the most information of how the adaptive changes affect the system. The simulation tag is important for the local replication of the cloud deployment for development purposes, as virtualized resources are provisioned taking space from the local workstation. To cope with the space limitations, further attributes in the simulation tag may be used to control the assigned sizes of the resource in relation to its production environment counterpart.

The ComponentSim shares the same behaviour for the lowest simulation level, i.e., a simulation.level of zero results in a simulation of state changes. For components, this comprises, e.g., changing the state of it from undeployed to deployed once a deploy action is triggered. For the simulation of components multiple simulation levels can be defined. For example, a simulation.level of one can additionally simulate the time required to perform lifecycle actions without actually executing them. Among others, this simulation is useful as it allows to more easily observe shifting runtime states enforced by orchestration processes. To define the timing of the actions the attributes action.timing.min and action.timing.max can be set describing the timings upper and lower boundaries. This tag can even be further specialized to handle higher levels of simulations. For example, a simulation.level of two may add artificial workload to be enforced on the compute node once the component's lifecycle actions are triggered. To generate the workload we directly coupled the ComponentSim tag with a configuration management script that allows to stress the compute resource. Finally, when setting the simulation.level to three an actual deployment is performed on the attached compute host. This simulation level helps to develop new configuration management scripts using the context provided by the runtime

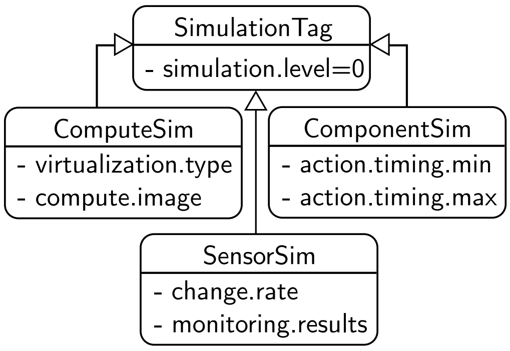

Figure 4: A subset of utilized simulation tags.

model or adjust existing ones. It should be noted, that for an actual deployment the attached compute host must have a simulation level greater than zero which can be checked by model validation.

The specialized simulation tags provided in Figure 4 are by no means exhaustive. For example, a tag for network simulations may be added that provides actions that inject, e.g., http faults. In the following, we discuss how locally deployed cloud applications can be simulated in a more dynamic environment by paying special attention to the SensorSim tag.

4.2 Orchestration Simulation

While the deployment simulation represents a rather static simulation, cloud applications need to be tested in dynamic scenarios to investigate their behavior under different load scenarios. As already mentioned, many runtime models provide scalability features allowing to define rules when to add or remove compute resources. Moreover, often orchestration engines are used which manage modeled resources and directly affect the runtime state. The partial deployment of the runtime model coupled with the simulated part allows to investigate how developed adaptive changes and new components cope with the activities performed by utilized orchestration engines. While we affect the effectors of the runtime model to simulate behaviour, the interface to the runtime model itself remains the same. Therefore, orchestration engines need next to no adjustments allowing to test the same scaling rules applied to the production environment in the local simulation environment.

To allow for a simulation of the orchestration process, we artificially generate workload by making use of monitoring information reflected within the runtime model (Erbel et al., 2019). Therefore, we introduce the SensorSim tag, see Figure 4, which allows to generate monitoring information over different means. Either the simulation mechanisms added by the tag can be used to generate monitoring information or the modeled sensor can be actually deployed and artificial workload can be enforced on the system. To generate monitoring information the montoring.results and change.rate can be specified by the developer to define the values the sensor can observe and how often it changes. To enforce the workload, the load simulation configuration management script coupled with the ComponentTag may be used. As a result, the developer can not only simulate how orchestration processes behave under generating monitoring results, but also under artificial workload. This in turn allows to locally test the scalability and robustness of designed cloud applications.

5 CASE STUDY

In our case study the production environment is a private Openstack cloud that currently hosts an Apache Hadoop cluster which needs to be enhanced with the functionalities provided by Apache Spark. To not compromise the functionality offered by the production environment, the required deployment and adaptation scripts need to be developed apart from it, which using our approach can be done offline and without the need of a dedicated staging environment. In this study, we completely develop the new desired component and its accompanying script offline. Furthermore, we investigate how the newly developed management scripts cope with scaling rules applied to the deployed cluster. In the following, the setup, execution and results of the case study are presented.

5.1 Case Study Setup

We implemented our approach using the OCCI a cloud standard, as it provides an extensible datamodel accompanied by a uniform interface supporting the runtime model management. The OCCIWare ecosystem (Zalila et al., 2017) implements an *Eclipse Modeling Framework* (EMF) metamodel for this standard and provides editors and generators, as well as a server maintaining an OCCI runtime model. Within our case study, OCCI extensions and effectors to support sensor (Erbel et al., 2019), container (Paraiso et al., 2016) and configuration management (Korte et al., 2018) are used. We enhanced these effectors with simulation behaviour for use in the local environment. For the deployment of the runtime model, a model-driven adaptation process is used that is already presented in previous work (Erbel et al., 2018). While a complete introduction of the standard is out of scope of this paper, its general structure fits to the one shown in Figure 1.

Figure 5 shows the runtime model describing the deployed hadoop cluster. While the production environment consists of multiple VM spanning the cluster, we only depict three machines for clarity. These three VMs consist of the fundamental components to deploy a Hadoop Cluster application. In this case one VM hosting a hMaster component which describes the clusters NameNode and ResourceManager. Furthermore, the application consists of two VMs hosting a hWorker component, representing the clusters DataNodes. Additionally, each VM in the model is connected over a network present in the runtime model. We omitted this network from the visualisation due to clarity.

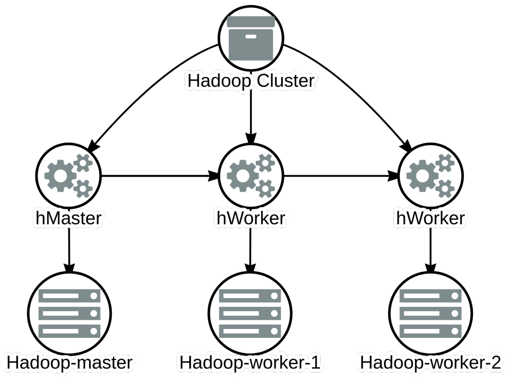

Figure 5: Production environment runtime model.

5.2 Case Study Execution

To perform our deployment simulation, we first retrieve the runtime model from the production environment. Thereafter, we performed the simulation transformation on it annotating each resource with a simulation tag. As shown in Figure 6, we then adjusted the simulation level of one worker and one master node in such a manner that an actual provisioning and deployment is performed. To enhance the cluster with the capabilities of Spark, we developed a new component spark and attached it to the Hadoop Cluster application. To not strain our local resources, we tagged the Hadoop-master and Hadoop-worker-1 so that they are provisioned in form of containers on which their components can be deployed. Furthermore, we adjusted the tags so that the Hadoop-worker-2 and its hosted component are only simulated on a model-based level, i.e., its state changes. We tag the runtime model in such a manner, as we assume that when the configuration management script to be developed operates correctly on one worker and master node, that it also performs correctly on each other node in the cluster. It should be noted, that the simulation levels can be also be adjusted at runtime. Finally, we deployed the tagged runtime model in our local simulation environment. This process resulted in the provisioning of two local container instances representing the Hadoop-master and Hadoop-worker-1 compute node, while the Hadoop-worker-2 is simulated on a model-based level and thus is only present in the runtime model in state active.

After checking the functionality of the deployed cluster, we enhanced the management scripts to additionally provision Spark capabilities. During this development process, we used the functionalities provided by the runtime model to manually trigger individual lifecycle actions of the components which allowed to develop individual parts of the configuration management scripts step by step. At first, we adjusted the deploy step responsible to simply download the Spark binaries and triggered the deploy lifecycle action. Thereafter, we adapted the configuration step of the component which among others utilizes information stored in connected components. In case of the hMaster component, e.g., the hostnames of connected worker nodes are registered. Especially, this step is of interest as it utilizes the information provided by the runtime model and thus also the simulated Hadoop-worker-2 compute node. Even though, the compute node is not started, we could observe that the configuration registered the node. It should be noted, that this configuration setup could lead to failing startups for non-dynamic and non fault tolerant cloud applications. Finally, for the start lifecycle action no adjustments had to be performed, as Spark requires the same services as the Hadoop cluster. To check whether the designed adaptations can be applied to the production environment, we store our performed changes. Thereafter, we freshly deployed the first version of the runtime model, i.e., the original duplicate of the production environment, and infused it with our newly developed components.

After we ensured that the adaptation of the hadoop cluster is applicable. We tested how our orchestration processes handle the deployment under specific workload. Therefore, we enhanced our runtime model with sensor elements and generated monitoring information for the CPU load of each worker and master node. To scale the deployment, we reutilize an engine previously developed, which monitors the gathered monitoring results in the runtime model which allows to adjust the deployment accordingly (Erbel et al., 2019). To quickly get an overview of the engines scaling mechanisms, we use the capabilities of the runtime model to directly set up and affect the results to be observed by the sensor. For example, to investigate the upscale behaviour we set the CPU sensor to only monitor critical load.

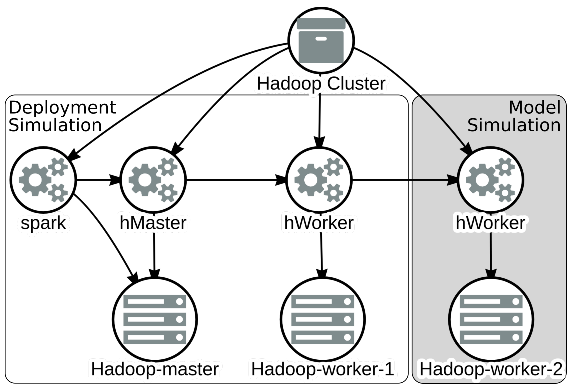

Figure 6: Local deployment simulation runtime model.

5.3 Results and Observations

In the deployment simulation, we chose to provision the production compute nodes locally as containers rather than as VMs. This decision resulted in faster local deployment times and lesser strain on our local system compared to the utilization of hardware virtualization. However, to spawn container nodes we needed to create a specialized image which mimics the VM of the production environment. Depending on the scenario, the change of virtualization techniques can apply some constraints such as denying access to specific configurations. After the creation of the image, the actual execution of the component on a local basis allowed for short cycles to create, improve and test the respective configuration management script. By simulating parts of the deployment model, we could utilize the information stored within the runtime model without an actual provisioning of the compute nodes resulting in less resources required. Among others, this simulation supported the development of queries used to retrieve the address and port of associated worker and master nodes.

While the orchestration simulation did not support the development of the cloud deployment directly, it allowed to test the robustness of developed adjustments. Furthermore, by applying already existing scaling rules, we observed that the some adjustments were required. The down scaling rule worked as intended, as only resources needed to be removed form the model. The upscaling rule however needed to be adjusted to incorporate the newly developed component. In the following, we provide a discussion about the extent to which the described simulation processes support DevOps to develop and integrate adaptive changes to running deployments.

6 DISCUSSION

For practitioners, the access to a local environment for cloud applications can support the development process, as it allows multiple developers to work simultaneously using their own hardware resources. Furthermore, it allows to develop and test adaptations offline, without requiring access to a staging environment. However, to create such a local environment, the size of modeled compute resources had to be reduced while focusing on areas of interest. Therefore, to create a meaningful environment, the person or team holding the DevOps role need to know what is required to observe and adjust the simulation levels of the system accordingly. Furthermore, attention must be paid to the available resources including their restrictions. To cope with this issue, defining a range of resources of interest could be beneficial to automatically adjust the partial deployment to local resources. Moreover, the described simulation capabilities helped to develop and test orchestration engines built around the runtime model. Especially, as a pure model-based simulation allows to execute automated tests. Additionally, when trying to identify bugs the timing of the lifecycle actions can be slowed providing a clearer picture about transitioning runtime states. Hereby, test cases can be built around individual runtime model states that can be easily extracted. Also, the research community may greatly benefit from a runtime model-based simulation environment. Not only because it allows to quickly develop and test cloud deployments, but also because it allows to provide an environment that may serve as a replication kit for performed studies. Hereby, the simulation environment allows an easier replication of the study, as no cloud access is required.

Overall, having a local simulation environment for distributed systems proved to be useful for the development of cloud applications while assessing the impact of changes prior to an actual deployment in a production environment.

6.1 Threats to Validity

Even though we only provided one case study to show the feasibility of our concept, the study revealed the possibilities of individual simulation levels of our runtime model-based approach. Furthermore, the study allowed to demonstrate how the environment supports the development process for cloud deployments and connected orchestration tools. Our approach depends on the utilization of runtime models for which many approaches can be found in the literature. We chose to evaluate the simulation approach using OCCI, as it is standardized and has an existing ecosystem built around it. Even though only one example cloud runtime modelling language was investigated, the concept of the approach is applicable to a wide variety of runtime models as the tag can be implemented by different means, e.g., as a simple annotation. To perform our study, only a single production environment cloud was used. However, due to the abstract nature of the model-driven approach the resulting model and thus our process is applicable on different providers as long as the implementation supports it. Furthermore, in terms of virtualization techniques our case study focused on the utilization of locally spawned container instances. While only the utilization of one kind of compute resources was discussed, there is conceptually no difference to provision locally VMs.

7 CONCLUSION

In this paper, we presented a simulation concept utilizing a runtime model to assess the impact of live cloud adaptations prior to a production deployment. We demonstrate how a production environment can be locally recreated by annotating individual model elements with desired simulation behaviour. Based on an example case study, we discussed how the partial simulation supports the development process of adaptive changes on a local workstation. Furthermore, we presented how the runtime model supports to assess the impact of developed changes in a dynamic environment. Therefore, orchestration engines adapting the cloud deployment were used which utilize simulated monitoring results. Overall, the approach supports the development process for cloud application by providing access to a local deployment and simulation environment which allows to early assess the impact of developed adaptations.

In future work, we aim at extending the simulation capabilities of the individual cloud components to exhaustively investigate the possibilities for a local replication. Additionally, we plan to extend the amount of cloud application scenarios the simulation environment is tested. Furthermore, we will investigate to what extent the simulation environment can be linked to the production environment to directly stream successful adaptive actions.

AVAILABILITY

The implementation and videos demonstrating the case study are available at: https://gitlab.gwdg.de/rwm/de.ugoe.cs.rwm.domain.workload.

REFERENCES

Achilleos, A. P., Kritikos, K., Rossini, A., Kapitsaki, G. M., Domaschka, J., Orzechowski, M., Seybold, D., Griesinger, F., Nikolov, N., Romero, D., and Papadopoulos, G. A. (2019). The cloud application modelling and execution language. *Journal of Cloud Computing*, 8(1):20.

Ahmed-Nacer, M., Kallel, S., Zalila, F., Merle, P., and Gaaloul, W. (2020). Model-Driven Simulation of Elastic OCCI Cloud Resources. *The Computer Journal*.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the Clouds: A Berkeley View of Cloud Computing.

Electrical Engineering and Computer Sciences, University of California at Berkeley.

Bencomo, N., Blair, G., Götz, S., Morin, B., and Rumpe, B. (2013). Report on the 7th International Workshop on Models@run.time. *ACM SIGSOFT Software Engineering Notes*, 38(1):27–30.

Blair, G., Bencomo, N., and France, R. B. (2009). Models@ run.time. *Computer*.

Brikman, Y. (2019). *Terraform: Up & Running: Writing Infrastructure as Code*. O'Reilly Media.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., and Buyya, R. (2011). Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50.

Erbel, J., Brand, T., Giese, H., and Grabowski, J. (2019). OCCI-compliant, fully causal-connected architecture runtime models supporting sensor management. In *Proceedings of the 14th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*.

Erbel, J., Korte, F., and Grabowski, J. (2018). Comparison and runtime adaptation of cloud application topologies based on occi. In *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER)*.

Favre, J.-M. (2004). Towards a Basic Theory to Model Model Driven Engineering. In *Proceedings of the 3rd UML Workshop in Software Model Engineering (WiSME)*.

Ferry, N., Chauvel, F., Song, H., Rossini, A., Lushpenko, M., and Solberg, A. (2018). Cloudmf: Model-driven management of multi-cloud applications. *ACM Transactions on Internet Technology (TOIT)*, 18(2):1–24.

Ferry, N., Chauvel, F., Song, H., and Solberg, A. (2015). Continous deployment of multi-cloud systems. In *Proceedings of the 1st International Workshop on Quality-Aware DevOps (QUDOS)*.

Guerriero, M., Garriga, M., Tamburri, D. A., and Palomba, F. (2019). Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*.

Hanappi, O., Hummer, W., and Dustdar, S. (2016). Asserting reliable convergence for configuration management scripts. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*.

Humble, J. and Molesky, J. (2011). Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, 24(8):6.

Korte, F., Challita, S., Zalila, F., Merle, P., and Grabowski, J. (2018). Model-driven configuration management of cloud applications with occi. In *Proceedings of the 8th International Conference on Cloud Computing and Services Science (CLOSER)*.

Kühne, T. (2006). Matters of (meta-) modeling. *Software & Systems Modeling*, 5(4):369–385.

Liu, C., Mao, Y., Van der Merwe, J., and Fernandez, M. (2011). Cloud resource orchestration: A data-centric approach. In *Proceedings of the biennial Conference on Innovative Data Systems Research (CIDR)*.

Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. Available online: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf, last retrieved: 04/22/2021.

OMG (2003). MDA Guide Version 1.0.1. Available online: http://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf, last retrieved: 04/22/2021.

OMG (2011). Unified Modeling Language Infrastructure Specification. Available online: http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF, last retrieved: 04/22/2021.

OMG (2016). Object Constraint Language. Available online: http://www.omg.org/spec/OCL/2.4/PDF/, last retrieved: 04/22/2021.

Paraiso, F., Challita, S., Al-Dhuraibi, Y., and Merle, P. (2016). Model-Driven Management of Docker Containers. In *Proceedings of the 9th IEEE International Conference on Cloud Computing (CLOUD)*.

Rahman, A., Mahdavi-Hezaveh, R., and Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108:65 – 77.

Szvetits, M. and Zdun, U. (2016). Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. *Software & Systems Modeling*.

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., and Soldani, J. (2020). The essential deployment metamodel: a systematic review of deployment automation technologies. *SICS Software-Intensive Cyber-Physical Systems*, 35(1):63–75.

Zalila, F., Challita, S., and Merle, P. (2017). A model-driven tool chain for OCCI. In *Proceedings of the 25th International Conference on Cooperative Information Systems (CoopIS)*.