Cloud Standby Deployment: A Model-Driven Deployment Method for Disaster Recovery in the Cloud

Alexander Lenk
FZI Forschungszentrum Informatik
Friedrichstr. 60, Berlin, Germany
lenk@fzi.de

***Abstract*—Disaster recovery planning and securing business processes against outtakes have been essential parts of running a company for decades. As IT systems have become more important, and especially since more and more revenue is generated over the Internet, securing the IT systems that support the business processes against outages is essential. For cost reasons, we proposed Cloud Computing for disaster recovery in prior work. In order to update and deploy a standby system, however, a deployment method that allows a provider independent, automated deployment of distributed systems is necessary. In this work we present such a deployment method. We show that by using Cloud Standby Deployment, the recovery time can be reduced significantly, and the distributed system can be deployed provider independently.**

Keywords— Disaster Recovery, Warm Standby, Cloud Computing, Cloud Standby

I. INTRODUCTION

Protecting [1] critical business processes against potential risks like earthquakes, fire, power outages, theft, illness, floods, and similar events has been a major concern of companies for a long time. To be prepared for these worst-case scenarios, disaster recovery plans are made in order to resume operations as soon as possible. These measures to keep up critical business processes in case of an emergency are often referred to as Business Continuity Management (BCM) [2] or, in the context of IT, as IT Service Continuity Management (ITSCM) [3]. To reduce the downtime of a system after an outage, it is common to replicate the whole system to another standby site [4] or even better, to another provider [5]. This is an established but expensive approach. Recent studies show that especially small and medium enterprises already use data backup solutions, but do not have systems in place that ensure a continuous operation in the case of a disaster – even though the downtime costs for these companies easily sum up to $12,500-23,000 per day [6]. In order to allow small and medium companies to prepare a standby system but have it switched off most of the time (warm standby), we developed Cloud Standby Disaster Recovery, a novel approach for warm standby in the Cloud (see Fig. 1). We showed that with Cloud Standby it is possible to provide a warm standby with small additional cost [7].

Cloud Standby uses a state of the art data backup method to sync the business data between the primary system (PS) and standby system (RS). So if a disaster strikes and the primary Cloud (C1) is no longer available, the standby system can be deployed in the Standby Cloud (C2) and operates as a production system during the outtake. Cloud Standby thereby takes advantage of the principles of Cloud Computing [8], [9], particularly that the user only pays for resources he actually needs. In Cloud Standby Disaster Recovery the cost savings are achieved by having the standby system shut down most of the time. From time to time, however, the standby system has to be started and updated with new business data, which is realized through an automated deployment method for distributed systems in the Cloud. In order to be used in the field of disaster recovery, this method also needs to be capable of deploying a distributed system to different Cloud providers. Even though in the related work there are several approaches that can be used to describe and deploy a system in the Cloud [10]–[16], none of these approaches allow a provider-independent description and deployment of a distributed system in the Cloud [17]. The emerging standard TOSCA [18] generally allows the needed description but is, however, too generic to be easily used in particular. It lacks Cloud specific elements that can be used in the topology and process description of a distributed system.

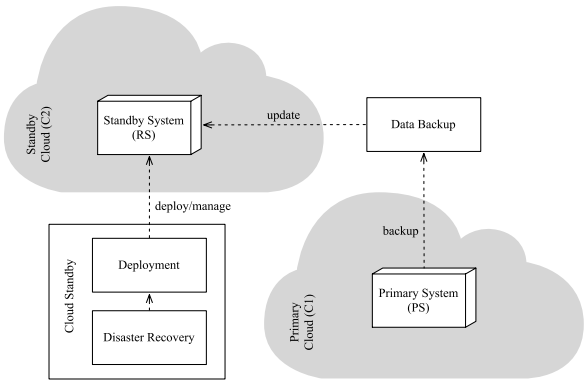

Fig. 1. Cloud Standby components and methods [7]

Therefore, the goal of this work is to provide a new deployment method for disaster recovery. This deployment method consists of an appropriate description language and a deployment process based on the same description language. The deployment method can be used standalone or as a basis to define abstract TOSCA topology elements and therefore solves the problem of TOSCA being too generic.

This paper is structured as follows: In section 2 we describe the related work in the field of deployment languages for the Cloud; in section 3 we present our new deployment method. Afterwards, we present a use case based evaluation in section 4 and finally in section 5 a conclusion and the future work.

[1] Parts of this paper have been published in German as part of a PhD thesis [1]

II. Related Work

Over the years many approaches for automated deployment in the Cloud were proposed. Many of them are related to this work. Chieu et al [19] have introduced the concept of "Composite Appliance" in their work. A composite appliance is a collection of virtual machine images that have to be started together in order to run a distributed system. The architecture proposed in this work requires the language to be implemented in the vendor's Cloud computing platform and thereby does not allow provider independence as necessary for disaster recovery. Konstantinou et al. [14] describe in their work a model-driven deployment approach. This approach is based on different parties that participate in the deployment process. It does however not allow having different Cloud vendors in a single deployment and the deployment process itself is not described. Based on this work and the work of Eilam et al. [20], [21] a new approach was presented by Kalantar et al. [13]. This approach is rather generic and lacks details that would be necessary to use it in the context of disaster recovery. If a approach is too generic almost everything can be described with it. The checking and validating of the correctness, however is no longer possible. The approach of Maximilien et al. [15] introduces a meta model that allows deploying middleware systems on a Cloud infrastructure. This approach however is focused on the deployment of middleware technologies for the disaster recovery. A more holistic approach is necessary to allow deploying a whole distributed system. This approach does not support modeling the distributed system provider independently. Mietzner et al. [22] orchestrate in their approach services by using workflows and web services. They focus on high level composition of services, rather than proposing an actual language for deployments. Cosmo et al. [23] and Catan et al. [16] introduce an approach for the description of software stacks and the dependencies between the software packages within a virtual machine. They provide a mathematical model that allows the optimization of the deployment process. They, however, focus on the software and do not take the cloud infrastructure into account, which is necessary for the usage in the given context of this work. Ferry et al. [12], [24] focus in their work on a description language that can be used to describe the deployment of distributed systems in the cloud. They focus on the structure of the application rather than on the software stack, resulting in a description language that can be used to deploy a distributed system with different cloud providers, but the language cannot ensure that the same software stacks are installed on every cloud. Brandtzæg et al. [11] describe an early idea for a deployment description language based on the approach of Flissi et al. [25], which was developed for Grid computing. The deployment description language however is rather generic and during the implementation it has to be decided how a deployment artifact is treated. In a disaster recovery context this generic nature of the language is a downside. Another very generic approach for the description of distributed systems in the Cloud is TOSCA [18]. The first standard for interoperable deployment of cloud applications on an infrastructure level is very promising, but in its current state very generic. Without cloud specific elements, not only cloud deployments but all kinds of things can be described and so it cannot be ensured that the modeled application is executable [26]. In the industry, Amazon Web Services [10] is a Cloud provider that added additional services over the time. For deployment, Amazon created the service Cloud Formation deployments. Even if the description language was ported to other platforms like Openstack, [27] only a single cloud provider can be described within a single model and no provider independence is possible. Approaches like IBM UrbanCode Deploy [28], IBM Smart Cloud [29] focus as well on the proprietary clouds rather than having an open, interoperable approach which is necessary for the field of disaster recovery with several different cloud providers.

For the use case of disaster recovery, a provider independent deployment method for the cloud is needed and none of the existing approaches allow the deployment of a distributed system on different cloud providers or have assumptions that make it hard to adapt them for this purpose. Therefore, we present in the next section a novel deployment method, designed for the use in the field of disaster recovery. Many deployment methods of the related work use meta modelling as a basis for the description language, so we also decided to use a model driven approach in our work. This also enables the incorporation of our results with an existing method or standard like TOSCA.

III. Deployment Method

In the related work, some approaches are based on meta-modeling in order to make it easier for the user to create models [30]. Also emerging standards like TOSCA [18] are based on meta-models, allowing the description of the topology and process of a distributed system. As described in the related work, the approaches are, however, not capable of describing a distributed system provider independently, or are too generic so that the Cloud specific parts are missing [26]. In this section, we therefore describe a meta-model based description language and deployment process that can either be used as a standalone deployment method or as a basis to define abstract TOSCA topology elements. We therefore solve the problem of TOSCA as being too generic. First we describe the design decisions and assumptions we made when we created the elements of the meta-model followed by a description of the meta-model itself and the process based on the meta-model.

A. Language Design and Assumptions

The goal of the method described in the paper is to provide a deployment method for the description of a distributed system on different Cloud providers in the context of disaster recovery. Therefore, the new deployment method has to be able to describe a *1) Distributed System 2) Provider Independent*ly and allow an *3) Automated Deployment*. Furthermore, these requirements have to be incorporated in the deployment method. Each of these requirements will be addressed in the following subsections.

1) Distributed System

According to Tanenbaum et al. [31] a distributed system consists of different tiers that work together in order to provide an application. A distributed system can be static or adapt to the current needs by scaling and is thereby elastic. The topic of elasticity became particularly prominent due to the emergence of Cloud computing. Before, scaling could only be implemented at great time intervals (new hardware had to be purchased and installed). With Cloud computing, resources can be added in a short time and can also be released again, which thus facilitates scaling downwards. In this work, elasticity is understood as the possibility of a distributed system or parts to be scaled upwards or downwards within a short time. Even if this work is focusing on disaster recovery and not scalable or elastic applications, we are not ignoring this fact and thereby limit our method to static applications. To keep a distributed system flexible, it is necessary to add and to remove virtual machines as desired without adversely affecting the overall system.

In this work, the concept of the "component" is used for horizontal scaling. A component (also called tier) is the entity known from the field of distributed systems [31]. It is a part of an application that can represent itself. Within a component, the virtual machines are homogeneous and can be added and removed. However, the scaling and especially the scaling decisions are not a focus of this work.

2) Provider Independence

One problem when dealing with standby systems hosted with different Cloud providers are lock-in effects [32]. Lock-in effects are caused by proprietary APIs, data formats, etc. A solution to avoid these lock-in effects is Cloud federation [33]. By the use of Cloud federation, various Cloud services are connected so that this creates a resource pool with the same or different functionality. These resource pools are not limited to a single Cloud provider but can combine different providers. Cloud federation distinguishes thereby two dimensions*: horizontal federation* and *vertical federation*. The horizontal federation establishes the resource pool with the same functionality, whereas vertical federation connects different complementary resources. In this work we use the concept of horizontal federation for grouping the resources of different Cloud providers.

In this work the federation is performed on the operating system level. While modeling, it will be ensured that functionally identical images which represent an operating system and virtual machines with the same non-functional properties are provided on all participating Clouds. For this reason, there are federated elements that allow to group functional (images) and non-functional (virtual machines) elements provider independently. In the following, these federated elements are presented in detail. If a running virtual machine (instance) has to fulfill the same function as part of a distributed system on multiple providers, the functional and non-functional properties of both instances have to be comparable. Thus, regardless of the provider, the same functionality with similar performance, costs, etc. must be provided. Therefore, the federated instance is referring to the elements of *federated image* and *federated virtual machine*, in which elements of the various Clouds are grouped so that in the end a functionally identical and non-functionally similar entity can be deployed. Fig. 2 illustrates this.

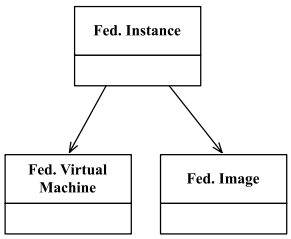

Fig. 2. Modeling of a horizontal federation

As mentioned before a federation on operating system level requires the ability to group elements with the same operating system configuration. These elements are therefore subject to the same basic operating system, the same installed software which is configured identically and thus have the same functional properties. These functionally similar operating system packages are persisted as images and can be instantiated several times. In order to group functionally identical images provider independently to *federated images*, this may be done on the basis of standardized software packages. Linux operating systems offer this standardization in form of distributions. A distribution is maintained by a single institution and always offers the possibility to manage the software packages via standardized distribution mechanisms (e.g. yum[2] or apt[3]). By introducing federated images and the use of standardized package managers, the modeler is given the possibility to describe a distributed system independent from the concrete provider and thus guarantee functional provider independence. However, it must also be ensured that the quality, with which these functional properties are made available, is comparable across providers. For the grouping of virtual machines the grouping on the basis of non-functional properties is not trivial: There is no standardization of the providers in terms of time, performance, safety, costs, etc. In many cases, however, it is not necessary that exactly the same properties are available for both providers. While it is already a problem if a function of the operating system is not provided, the basic functions of the virtual machines are standardized, but the quality of the properties that they are provided with, are not standardized. In this case it is up to the modeler to conduct the suitable grouping of the virtual machines for the concrete application. To this end, there are also suggestions in the literature [34] or additional measures, such as decision support.

3) Automation

Regarding the choice of the modeling approach in this work, the benefit and the applicability for implementation in a concrete system, represent important decision criteria. While BNF-based methods are particularly popular in compiler construction and programming languages, many web service technologies and emerging standards like TOSCA are based on XML. For this reason, in this work meta-modeling and the standard MOF [35] are used for the description and serialization of deployments. Furthermore, meta-modeling has the advantage that there is a good tool support for MOF-based metamodels by the Eclipse Foundation [36]. In the following subsection, the description language using the MOF formalization is presented.

B. Description Language

For a better understanding and presentation, the grammar of the description language, illustrated in the MOF graphical presentation, is divided into different elements or packages, individually described as: *Software*, *Infrastructure*, *Federation*, *Distributed System*, and *Data Restore*. Here, *Infrastructure* refers to the elements that are provider specific (e.g. Cloud datacenter, virtual machines and images). Since especially in the field of disaster recovery the provider independence and the reduction of lock-in effects are important, not only a simple but a federated infrastructure is used. In the *Federation* package these provider independent elements are described. The *Distributed Systems* package contains all the elements necessary to create a distributed system capable of running on the federated infrastructure. The *Data Restore* package incorporates all the elements needed to restore a data backup and thereby replicating the data of the primary system to the standby system[4].

1) Software

According to Tanenbaum et al. [31] a distributed system consists of different computing units such as virtual machines with installed software. There are different methods to distribute software on virtual machines. Each operating system has, for example, built-in package managers in order to maintain

[2] http://yum.baseurl.org/

[3] https://wiki.debian.org/Apt

[4] a rudimentary first version of the meta model was published in [37]

software. Furthermore, in addition to these built-in package managers, such as Apt, there are a number of configuration managers which can manage software packages across different operating systems [38], [39]. A goal in the design of the software package introduced here is to integrate existing solutions as flexible as possible. While this step is still relatively easy for package managers, because they can be selected from the operating system using simple batch files (batch scripts), it is often more difficult for configuration managers. These are provided by external components and manage the installation scripts, variables and any other data in its own format that are involved in the configuration. The translation of the meta-model presented here is in the implementation. A representation of the meta-model is shown in Fig. 3.

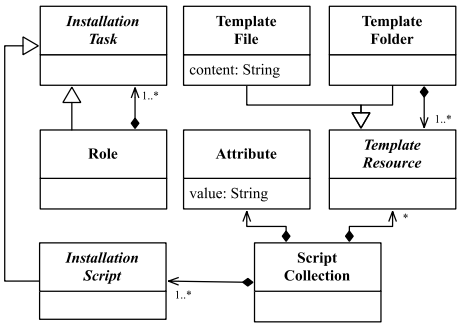

Fig. 3. MOF class diagram of the software package

In this meta-model the abstract element *Installation Task* describes the task that is executed on a running instance in order to do a configuration or installation. The Installation Task is an abstract meta class, so in the final Model it has to be specified by an *Installation Script* or *Role.* Together these three elements build the composite design pattern [40], forming a tree data structure with Installation Scripts as leaves. During runtime a depth-first search can be used to serialize the tree to an execution plan. The abstract element *Installation Script* is a program executed on the running instance. The concrete element has to be defined in the implementation and could, for example, be a bash script. For installation scripts it is important that they are idempotent [41], since they might be executed several times during the update cycles of the disaster recovery process. Several installation scripts can be combined in *Script Collections.* All the scripts in a script collection can have the same variables or *Attributes* and the same template files. Similar to the tasks, the template files are stored in a tree data structure defined by the *Template Files* as leaves and the *Template Folders* as nodes, connected by the abstract element *Template Resource*.

2) Infrastructure

The infrastructure is the foundation of the distributed system. It combines the software with the hardware. Since the goal of this method is the provider independent deployment, all of the elements in this package that are provided directly or indirectly by a single Cloud provider are combined. If a new Cloud provider is to be included in the Cloud Standby Disaster Recovery, it has to be added using the infrastructure part shown in Fig. 4.

The element *Cloud* describes the data center where the Cloud resources are provided. Within a data center all the services and costs models are the same. If a single provider has different data centers in different regions (e.g. US and Europe), different Cloud elements have to be created.

In order to startup a virtual server, virtualized hardware has to be provided by the hypervisor. In this work this virtualized hardware is called *Virtual Machine*. The software running within a virtual machine is packaged as an *Image*. This means an image is a collection of executable software artifacts, configuration files and so on, executed in the virtual machine. The image always consists of the operating system and all necessary software packages. In this meta-model, the abstract element image is an interface for the *Standard Image* and the *Configured Image*. The standard image is provided by the Cloud provider as a whole; while on the configured image during deployment different installation tasks (see Software) are executed. The modeler decides if in a certain use case a standard or configured image should be used. If the software packages often change for example, it might be useful to model it as a configured image, so that a clean installation is performed during deployment. Usually a fixed amount of storage is assigned to a virtual machine. Some Cloud providers, however, allow the attachment of *Additional Drives*. The element *Instance* stands for the deployed and running virtual machine. On an instance an operation system communicating with the virtual hardware and the software stack is needed to provide all the functionality necessary for the distributed system.

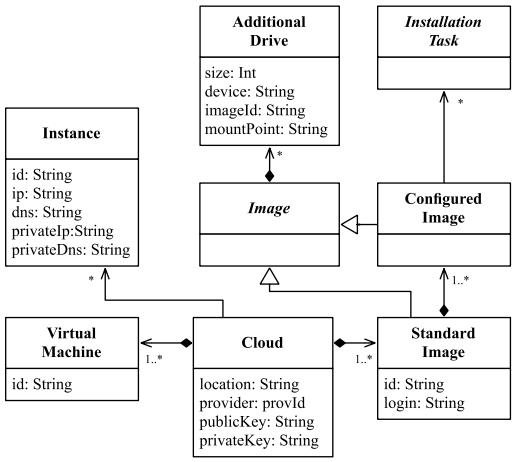

Fig. 4. MOF class diagramm of the infrastructure package

3) Federation

In general, Cloud providers are trying to retain their customers by lock-in effects [33]. Therefore, the interoperability between the providers is rather low. For disaster recovery it is, however, necessary to establish interoperability between different Cloud providers. Therefore, the distributed system is not directly assigned to provider-specific elements, but is complemented by an additional abstraction layer, the *Federation*. During deployment it will be decided which of the provider-specific elements of the federation elements will be selected and used. The meta-model is shown in Fig. 5.

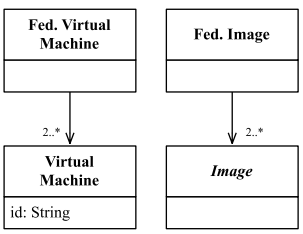

Fig. 5. MOF class diagram of the federation package

The *Federated Virtual Machine* is a group element for all the provider specific virtual machine elements in the model. The virtual machine elements hereby represent the non-functional attributes of the running instance. These attributes are not explicitly modeled, but the modeler is using his knowledge to group only elements with similar non-functional attributes. In order to support him, specific discussion support tools like presented by Wittern et al. [42] can be used. Also for grouping the *Federated Images* tools like the image crawler by Menzel et al. [43] exist. When grouping images, the modeler has to make sure that all the images within the group have the same functionality, meaning same software packages with the same configuration installed.

4) Distributed System

The distributed system package contains all of the elements that are necessary to describe the distributed system on the federated Cloud infrastructure. As described before, the model is designed to support elastic distributed systems. The meta-model is illustrated in Fig. 6.

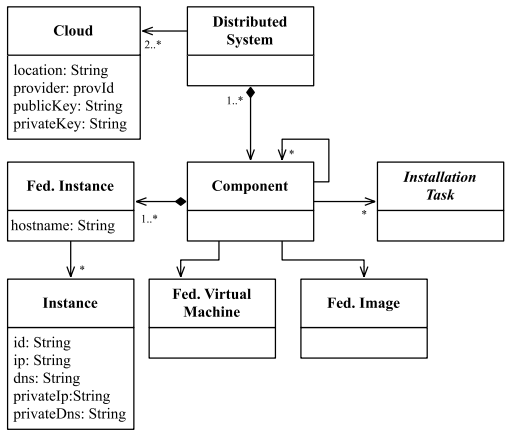

Fig. 6. MOF class diagram of the distributed system package

The element *Distributed System* represents the distributed system of the real world that has to be deployed provider independently. The distributed system has a reference to the *Cloud* element in the infrastructure package. In order to have a reliable and deterministic disaster recovery, it is necessary that in the case of a disaster always the same provider is used as the standby Cloud. The reference defines an ordered list of Clouds that has to include at least two elements: primary Cloud and standby Cloud. In the model more than two elements are allowed in this list, meaning that more than one standby Cloud is supported. However, when using this deployment method with Cloud Standby Disaster Recovery, only one standby Cloud is supported. More standby Clouds are subject of our future work. The Federated Instance is an abstract instance grouping the replicated instances of different Cloud providers. This element is automatically created and cannot be modeled. In order to allow vertical scaling and elastic applications the virtual machine and image are not attached to the instance but rather to the *Component*. This ensures that all the instances within a component have the same configuration and can therefore be vertically scaled. Between components, requirement relationships can be defined. This allows the modeler for example to ensure that during deployment, the application servers are started after the database servers and the load balancer. Nevertheless the modeler has to ensure that the requirement relationships are not cyclic. In order to reduce the complexity of certain models, installation tasks can be attached directly to the component. This allows, for example, using the same image for the Web server and database server component.

5) Data Restore

The data recovery backup package bundles all the elements that are necessary for restoring an existing backup (see Fig. 7). In the meta-model, this is only an abstract element, which must be extended by concrete elements in the implementation. These concrete elements are also part of the data recovery backup package.

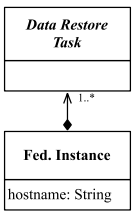

Fig. 7. Meta-model of the data restore package

The *Data Restore Task* is especially important for the Cloud Standby Disaster Recovery. It is used to replicate the business data between Cloud providers. Restoring the last data backup is the last task in the deployment process, described in the following section. The Restore Task is also an abstract element that has to be concretized when implementing the method.

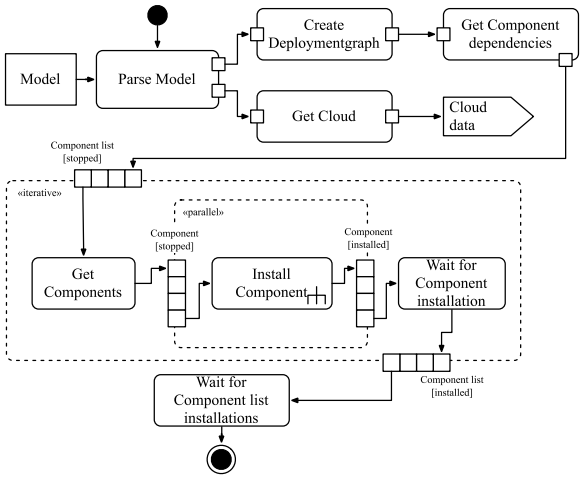

Fig. 8. The overall deployment process

C. Deployment Process

The Cloud standby description language is designed to deploy a distributed system on various Cloud providers. It only defines a data model for describing the distributed system but does not contain the processes necessary to run the system. Therefore, this process will be explained here. The deployment process is the core of the deployment method. It is closely linked to the description language, since it uses the modeled elements and their attributes to instantiate a running system from it. The overall process is illustrated in Fig. 8.

The process starts with parsing the model and determining the Cloud provider where the deployment has to be executed. From the requirement relationships between the components the deployment process can also identify parallelizable deployment tasks. This is a huge advantage of this deployment method. Even in changing deployment structures the deployment process. The deployment algorithm is based on a MPM

algorithm [44] allowing to identify parallel paths in a graph. The MPM algorithm is obtained from the critical path analysis in operations research[5]. After having determined the list of parallelizable components, these can, in the next step, be installed in parallel. The components within a list, however, have to be installed in sequence. When all components are installed the deployment process is done. The process of the installation of a single component is illustrated in Fig. 9.

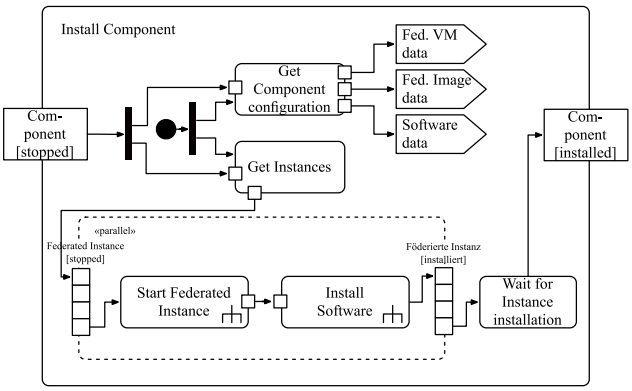

Fig. 9. Installation of a single component

The installation of a component starts, like the overall process, with the parsing of the model and determination of the federated virtual machine, the federated image, and the software. When all this information is gathered, all the instances of the components can be installed simultaneously. This means that the instance is started on the selected Cloud provider and the corresponding software has been installed. If the instance is started with software already installed (e.g. as part of the update cycle), the process is finished very quickly. However, if it is installed from scratch, the process can take several minutes. When all instances are started, the component is deployed. After presenting the deployment description language and the deployment process as part of the deployment method in this section, the following is dealing with the evaluation of the method in the context of disaster recovery in a given use case.

IV. EVALUATION

For the evaluation of this method we apply a use case based experimental evaluation. As shown in the related work, no other deployment method is available for the use in the context of disaster recovery. Therefore, we evaluate how disaster recovery can benefit from the modeling of the distributed system on top of different Cloud providers. In particular, we analyze how the modeling of requirements between components can be used to speed up the deployment process, if the distributed system can be deployed on different Cloud providers, and how the automated deployment can reduce the recovery time of a distributed system when used with disaster recovery. For executing the experiments, we used our implementation of Cloud Standby Disaster Recovery, Cloud Standby Deployment, and the Cloud Standby Editor for creating the model[6].

A. Use Case

For our evaluation, we use the infrastructure of the barcoo application. Barcoo is a German startup with over 10 million app downloads. The barcoo infrastructure consists of four components: a load balancer, a set of application servers, a MySQL database, and a NoSQL database (for caching).

The load balancer is used as an access point for all clients (browser, smartphone app, etc.). Here, the requests are forwarded to an application server. Each application server uses data from the MySQL database or NoSQL-cache in order to make the appropriate information available to the user. Data in the NoSQL store comes mainly from external providers (price service provider, warnings about food scandals, etc.) and are cached only for performance and bandwidth reasons. Besides non-critical cache data, the production system of barcoo also includes mission-critical and privacy-related data. For the unlimited use of this data, special precautions and consents to third parties are required, which cannot be obtained in the evaluation. For this reason, not all data of barcoo is available for an evaluation and will be replaced by non-critical data in this paper. Thus, it is also possible to reproduce the findings of this work without access to the internal data of barcoo[7]. We assumed that the database is growing by 400 MB every week, and every two weeks there is a new release of the application. In our setup, having 16 experiments for the time span of 8 weeks, this means the time between two experiments is 3.5 days.

B. Experimentation and Results

The aim of the model-based deployment method presented herein is to enable a provider independent deployment. To prove this, the distributed system described in the use case was first deployed once to multiple Cloud datacenters operated by different providers. As shown in Fig. 10, the deployment on the providers AWS EC2, Google, HP, and Rackspace was successful in several different Cloud datacenters. It turned out that sometimes the deployment times clearly differ from each other at the different providers. Reasons for this could be the better performance of the cloud provider and its management processes, the currentness or structure of the images, or other unknown reasons. Since there is a big difference between the various providers, the reason for the differences should be examined more closely in future works.

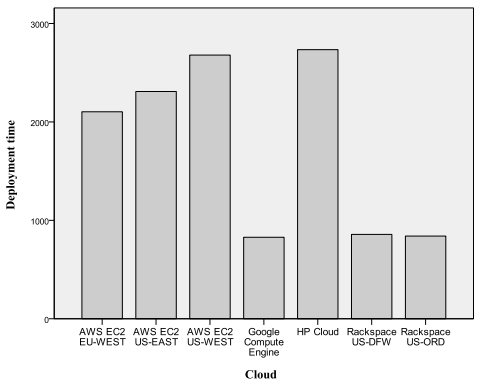

Fig. 10. Comparison of deployment times of the initial deployments of various providers

The next question was how the modelling itself can speed up the deployment process. As described above, different dependencies within a distributed system may exist. For example, the deployment of the application server might require that a database is already available. In the description language, these dependencies are made explicit by means of the association

[5] for a detailed description of the algorithm see [37]

[6] availabe online: https://github.com/alexlenk/CloudStandby

[7] for detailed information on the experiment setup see [7]

"component requires component" (see section III.B.4). This facilitates the prevention of errors in the deployment. Moreover, a deployment graph can be created from the above-mentioned association in order to automatically perform the deployment. The order of the various components to be deployed must then be derived from the deployment graph. Without deployment modelling of the components, a sequential deployment can ensure that no problems occur during deployment and no dependencies are violated. However, parallelizable deployment steps are neglected in this case. In this experiment we started the distributed system of the use case on AWS EC2[8] with our Cloud Standby implementation. For this reason a deployment algorithm was introduced as part of the deployment process in section 3 which also takes parallel paths into account and thus results in a reduction of the deployment time. A comparison of the deployment time for sequential deployment and parallel deployment based on our deployment process is shown in Fig. 11.

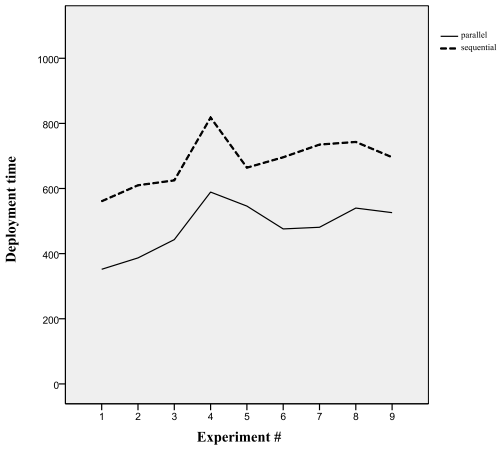

Fig. 11. Influence of parallel deployments on the deployment time

For each of the cases we made sure that all the necessary software packages were installed in experiment no. 0 and saved the system. In the next experiment runs we started the distributed system for each case and measured the deployment time. As illustrated in Fig. 11, all experiments result in a reduction of deployment time by about 200 seconds or 30% in the case of parallel deployment with our proposed deployment description and process as opposed to the sequential deployment without taking parallelizable tasks into account. The influence of a new application release is also observable in the figure: in experiment the deployment time raises clearly in no. 4 and slightly in no. 8 in both the parallel and the sequential deployment.

Since we now evaluated that the standby system can be deployed on different clouds and that the deployment modelling allows for an automated deployment that takes parallelizable deployment tasks into account, we now evaluated the new deployment method in the context of Cloud Standby Disaster Recovery where it is used as a basis for the whole approach. Without the Cloud Standby Deployment also Cloud Standby Recovery as published in our previous work[9] would not be possible. For the evaluation of the whole approach we prepared two systems: One with and the other without Cloud Standby. When deploying the system in the experiment series 46 times, we used the data backup restore mechanism in each step to update the system to the current status. Since the system without Cloud Standby had to restore more data over time, the deployment time increased. The system with enabled Cloud Standby, however, had almost the same deployment time in every experiment (see Fig. 12).

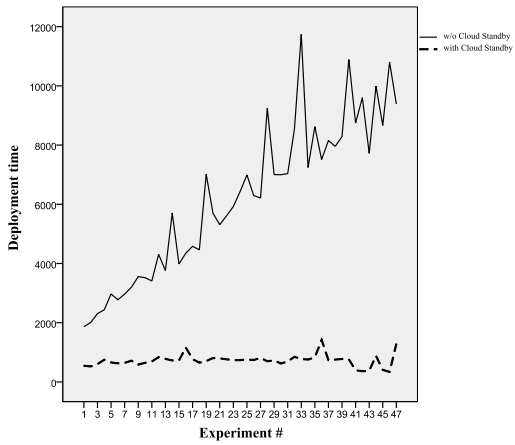

Fig. 12. Deployment times with Cloud Standby [7]

It can be seen in the figure it is possible to significantly reduce the deployment time of a standby system with Cloud Standby. Without Cloud Standby the deployment time in our experiments went up to about 15 times the deployment time of the system with enabled Cloud Standby. We can also see in the figure that, especially in the case without Cloud Standby in some cases, the deployment time has several spikes. These spikes are unpredictable and seem to be caused by the currently available performance of the VM when restoring the data in the database. In earlier studies, we revealed that the variance of performance in CPU intensive tasks in Cloud Computing can be big [34]. Also, when transferring huge amount of data, the network has a big influence on the deployment process. We would like to further investigate in our future work what the cause of these spikes in the context of disaster recovery is. In the deployment times of the system with Cloud Standby, only few spikes are noticeable in the figure. The spikes at 4, 16, and 36 are the new application server software versions, needing new Ruby versions, which are thus deployed on the system. The influence of the compute infrastructure on the deployment process observed for the case without Cloud Standby is furthermore reduced in the case of Cloud Standby as only few data had to be inserted in each step. Thereby, Cloud Standby also leads to much better predictability of deployment time.

V. Conclusion and Future Work

In this work we introduced a new deployment method that was designed for the use with Cloud Standby Disaster Recovery for warm standby in the Cloud. The deployment method consists of a description language based on a meta model and a deployment process. The description language can be used to model a distributed system independently from a specific provider and so the created model can be used in order to deploy the distributed system provider independently. The evaluation showed that it was possible to deploy a single distributed system on 7 different Clouds, operated by 4 different Cloud providers. The deployment process presented in this paper is based on the description language and determines parallel paths within the modeled deployment to reduce the deployment time. The evaluation showed that in the given use case the deployment time can be reduced by about 30%. When used with Cloud

[8] http://aws.amazon.com , US east datacenter, m1.small instance

[9] for more information on the conducted experiments and the results see the publication of the Cloud Standby Disaster Recovery approach [7]

Standby Disaster Recovery, the deployment method thus enables a significant reduction of the deployment time of a standby system in the case of a disaster with only little financial overhead [7]. In our future work we would like to incorporate the description language and deployment process with the emerging standard TOSCA in order to give the user of this standard more assistance when deploying a distributed system on different cloud providers. We also would like to investigate the fault-tolerance of the deployment method itself.

ACKNOLEDGEMENTS

We would like to thank Tobias Bräuer for his help and the insights he gave us on the barcoo infrastructure.

REFERENCES

[1] Alexander Lenk, "Cloud Standby - Eine Methode zur Notfallwiederherstellung in der Cloud," PhD Thesis, Karlsruher Institut für Techologie (KIT), Karlsruhe, 2014.

[2] A. Hiles, *The definitive handbook of business continuity management*. Wiley, 2010.

[3] I. Menken, T. Malone, and G. Blokdijk, *Itil V3 Malc-Managing Across the Lifecycle Full Certification Online Learning and Study Book Course: The Itil V3 Intermediate Malc Complete Certification Kit.* Emereo Pty Limited, 2009.

[4] M. Whitman, H. Mattord, and A. Green, *Principles of incident response and disaster recovery*. Cengage Learning, 2013.

[5] M. Pokharel, Seulki Lee, and Jong Sou Park, "Disaster Recovery for System Architecture Using Cloud Computing," in *Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on*, 2010, pp. 304–307.

[6] Symantec, "2011 SMB Disaster Preparedness Survey - Global Results," 2011.

[7] A. Lenk and S. Tai, "Cloud Standby: Disaster Recovery of Distributed Systems in the Cloud," in *Service-Oriented and Cloud Computing*, vol. 8745, M. Villari, W. Zimmermann, and K.-K. Lau, Eds. Springer Berlin Heidelberg, 2014, pp. 32–46.

[8] P. Mell and T. Grance, "The NIST definition of cloud computing," *NIST Spec. Publ.*, vol. 800, p. 145, 2011.

[9] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the Cloud? An architectural map of the Cloud landscape," in *Software Engineering Challenges of Cloud Computing, 2009. CLOUD'09. ICSE Workshop on*, 2009, pp. 23–31.

[10] AWS Inc., "Amazon Web Services, Cloud Computing: Compute, Storage, Database," 2013. [Online]. Available: https://aws.amazon.com/. [Accessed: 30-May-2013].

[11] E. Brandtzæg, P. Mohagheghi, and S. Mosser, "Towards a domain-specific language to deploy applications in the clouds," in *CLOUD COMPUTING 2012, The Third International Conference on Cloud Computing, GRIDs, and Virtualization*, 2012, pp. 213–218.

[12] N. Ferry, F. Chauvel, A. Rossini, B. Morin, and A. Solberg, "Managing multi-cloud systems with CloudMF," in *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies*, 2013.

[13] M. H. Kalantar, F. Rosenberg, J. Doran, T. Eilam, M. D. Elder, F. Oliveira, E. C. Snible, and T. Roth, "Weaver: Language and runtime for software defined environments," *IBM J. Res. Dev.*, vol. 58, no. 2, pp. 1–12, 2014.

[14] A. V. Konstantinou, T. Eilam, M. Kalantar, A. A. Totok, W. Arnold, and E. Snible, "An architecture for virtual solution composition and deployment in infrastructure clouds," in *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, 2009, pp. 9–18.

[15] E. M. Maximilien, A. Ranabahu, R. Engehausen, and L. C. Anderson, "Toward cloud-agnostic middlewares," in *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications - OOPSLA '09*, Orlando, Florida, USA, 2009, p. 619.

[16] M. Catan, R. Di Cosmo, A. Eiche, T. Lascu, M. Lienhardt, J. Mauro, R. Treinen, S. Zacchiroli, G. Zavattaro, and J. Zwolakowski, "Aeolus: Mastering the Complexity of Cloud Application Deployment," in *Service-Oriented and Cloud Computing*, vol. 8135, K.-K. Lau, W. Lamersdorf, and E. Pimentel, Eds. Springer Berlin Heidelberg, 2013.

[17] A. Lenk, C. Danschel, M. Klems, D. Bermbach, and T. Kurze, "Requirements for an IaaS deployment language in federated Clouds," in *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, 2011, pp. 1–4.

[18] OASIS, "Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0." 18-Mar-2013.

[19] Trieu Chieu, A. Karve, A. Mohindra, and A. Segal, "Simplifying solution deployment on a Cloud through composite appliances," in *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, 2010, pp. 1–5.

[20] T. Eilam, M. H. Kalantar, A. V. Konstantinou, G. Pacifici, and J. Pershing, "Managing the configuration complexity of distributed applications in internet data centers," *IEEE Commun. Mag.*, 2006.

[21] T. Eilam, M. Kalantar, A. Konstantinou, and G. Pacifici, "Model-based automation of service deployment in a constrained environment," *Res. Rep. IBM*, 2004.

[22] R. Mietzner, T. Unger, and F. Leymann, "Cafe: A generic configurable customizable composite cloud application framework," *Move Meaningful Internet Syst. OTM 2009*, pp. 357–364, 2009.

[23] R. Di Cosmo, S. Zacchiroli, and G. Zavattaro, "Towards a formal component model for the cloud," in *Software Engineering and Formal Methods*, Springer, 2012, pp. 156–171.

[24] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards Model-Driven Provisioning, Deployment, Monitoring, and Adaptation of Multi-cloud Systems," 2013, pp. 887–894.

[25] A. Flissi, J. Dubus, N. Dolet, and P. Merle, "Deploying on the Grid with DeployWare," 2008, pp. 177–184.

[26] G. Katsaros, M. Menzel, A. Lenk, J. R. Revelant, R. Skipp, and J. Eberhardt, "Cloud Application Portability with TOSCA, Chef and Openstack," in *Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, 2014, pp. 295–302.

[27] Openstack, "Heat." [Online]. Available: https://wiki.openstack.org/wiki/Heat. [Accessed: 02-Aug-2014].

[28] IBM Corporation, "IBM UrbanCode Deploy," 2013. [Online]. Available: http://public.dhe.ibm.com/common/ssi/ecm/en/rad14132usen/RAD14132USEN.PDF. [Accessed: 16-Apr-2014].

[29] IBM Corporation, "IBM SmartCloud Provisioning," 2013. [Online]. Available: http://public.dhe.ibm.com/common/ssi/ecm/en/ibd03003usen/IBD03003USEN.PDF. [Accessed: 16-Apr-2014].

[30] T. Eilam, M. H. Kalantar, A. V. Konstantinou, G. Pacifici, J. Pershing, and A. Agrawal, "Managing the configuration complexity of distributed applications in internet data centers," *Commun. Mag. IEEE*, vol. 44, no. 3, pp. 166–177, 2006.

[31] M. Van Steen and A. Tanenbaum, "Distributed Systems: Principles and Paradigms," *Network*, vol. 1, p. 2, 2001.

[32] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and others, "Above the clouds: A berkeley view of cloud computing," *EECS Dep. Univ. Calif. Berkeley Tech Rep UCBEECS-2009-28*, 2009.

[33] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze, "Cloud federation," presented at the CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization, 2011, pp. 32–38.

[34] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann, "What Are You Paying For? Performance Benchmarking for Infrastructure-as-a-Service Offerings," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, 2011, pp. 484–491.

[35] Object Management Group, Inc. (OMG), "Meta Object Facility (MOF) Core Specification Version 2.4.1." 2011.

[36] D. Steinberg, *EMF: Eclipse Modeling Framework*, 2nd ed., Rev. and updated. Upper Saddle River, NJ: Addison-Wesley, 2009.

[37] A. Lenk and F. Pallas, "Cloud Standby System and Quality Model," *Int. J. Cloud Comput. IJCC*, vol. 1, no. 2, pp. 48–59, 2013.

[38] Opscode, "Chef." [Online]. Available: http://www.opscode.com/chef/. [Accessed: 30-May-2013].

[39] S. Nelson-Smith, *Test-driven infrastructure with Chef*, 1st ed. Sebastopol, CA: O'Reilly, 2011.

[40] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[41] W. Hummer, F. Rosenberg, F. Oliveira, and T. Eilam, "Testing Idempotence for Infrastructure as Code," in *Middleware Conference*, 2013.

[42] E. Wittern, A. Lenk, S. Bartenbach, and T. Braeuer, "Feature-based Configuration and Deployment of IaaS Execution Environments," presented at the ICSOC, 2013.

[43] M. Menzel, M. Klems, H. A. Le, and S. Tai, "A configuration crawler for virtual appliances in compute clouds," in *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, 2013, pp. 201–209.

[44] K. Neumann and M. Morlock, *Operations Research*, 2. Aufl. München: Hanser, 2002.