Facilitating Agile Prototyping of Cloud Applications — A Model-Based Approach

Ta'id Holmes
Infrastructure Cloud, Deutsche Telekom Technik GmbH
Darmstadt, Germany
t.holmes@telekom.de

***Abstract*—Modern cloud applications, generally, demand complex service topologies, e.g., for meeting scalability, maintenance, or security requirements. Thus, often, it is desirable to increase the complexity of service topologies in cloud applications. The required changes, however, may constitute a burden for improving cloud applications. Changes, overall, are undertaken frequently in prototyping or when adopting agile development. Their costs correlate with the number of people involved. For facilitating agile prototyping of cloud applications this demonstration presents a model-based approach incorporating different roles. Using, integrating with, and building on top of open-source projects, it comprises domain-specific language editors and showcases their use and the realized automation fostering iterative development.**

***Index Terms*—automation, agile, application, cloud, DSL, model-based, prototyping, provisioning, service topology**

I. Introduction

For various reasons, modern cloud applications are composed of increasingly complex service topologies (cf. [1]). Factors that contribute to the complexity of service topologies are the ability to scale out services for meeting varying user loads, security requirements that demand the separation of application logic as well as of data, and maintenance considerations that foster distribution of self-contained services for decoupling their lifecycle.

For expressing and capturing service topologies, the OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) standard [1] provides a metamodel. When describing a cloud application in terms of a metamodel and when following a model-driven engineering (MDE) approach, provisioning can be automated (cf. [2]). Yet, development of a cloud application remains difficult as its service topology needs to be modeled and the various services implemented. In an agile practice, the architecture and implementation of cloud applications usually experiences early and frequent changes such as refactorings. For example, the topology may be changed in a way that requires alignment of the model, e.g., when services are split into distributed entities. At the same time, the respective services need to be adapted as well.

Multiple roles with particular responsibilities are involved in the engineering process, such as architects, test developers, and service engineers – each one having a different set of expertise. For realizing a new cloud application an enterprise architect performs a functional analysis and aligns architectural building blocks (ABBs) with software building blocks (SBBs) (cf. [3]). The service topology, that comprises latter building blocks, may be modeled by a team of experts. Some services are available off the shelf, while others need to be implemented by service engineers. Deployment services automate the provisioning of different stages such as for testing or pre-production. Finally, a continuous integration service executes various tests that have been implemented by test developers.

In case of iterative development all of these roles repeatably need to coordinate their activities. If a cloud architect would like a particular service to be decoupled from another because of scalability issues the alignment of building blocks may need adjustment, the model reflecting the service topology needs to be changed, tests may have to be adapted, and service implementations have to be modified.

For lowering the burden of improving the cloud application accordingly the overall turnaround time shall be kept as short as possible in such cases. With a multitude of people and roles involved, agile development of complex cloud applications is challenging: Most of all, coordination needs have to be addressed. Next, development of individual services needs to be facilitated in a sense that integration into the service topology is eased. Finally, overall automation is required.

Tackling these issues, the demonstration proposes a model-based approach and presents a toolset comprising domain-specific language (DSL) editors for facilitating agile prototyping of cloud applications. Showcasing an overall example, it combines two former contributions: Addressing coordination needs, a descriptive DSL [4] permits to define roles, artifacts, and services. The alignment of ABBs and SBBs can be undertaken by an enterprise architect using another DSL which is also used for describing the provisioning of the cloud application [5]. Using a running example, this demonstration describes the overall process, artifacts, and tools involved. Abstracting from infrastructure as a service (IaaS) providers, the approach automates various tasks and enables developers to incrementally deploy cloud services facilitating iterative prototyping of cloud applications.

The remainder is structured as follows: Section II introduces the running example of the demonstration by describing its context and by characterizing related problems as a further motivation. The approach using the running example is presented in Section III. The work is compared to some related work in Section IV and Section V concludes.

[1] http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf

II. Running Example: Context and Motivation

Let us now consider an industrial machine-to-machine (M2M) context in which a new business idea arose. As part of the value proposition it envisages the correlation of sensor data with other data and the visualization of results in a dashboard. In this scenario M2M devices emit data from sensors to an M2M gateway. A backend server receives and processes aggregated sensor data from these gateways. For this, the data is normalized and correlated with user data and data from an external data source. Results are stored in a database and visualized in a web application. Figure 1 depicts a rudimentary server landscape for the M2M scenario.

Various roles participate in the engineering: among them are service and web engineers that implement different parts of a minimum viable product (MVP) (see below) as well as architects that describe the functional architecture.

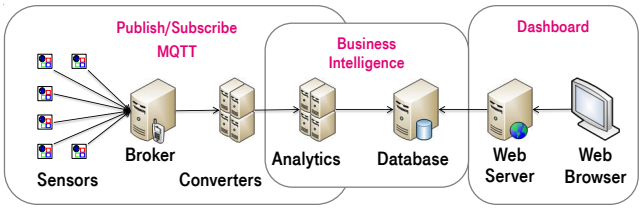

Figure 1. Server Landscape of a Machine-to-Machine Scenario

Following the idea of the lean startup philosophy [6] the value proposition of the business idea is to be tested with potential customers. For this, prototyping constitutes a suitable approach for the (rapid) development of a MVP (cf. [7]) and for gaining findings such as customer requirements.

In situations in which products depend on several cloud services a platform as a service (PaaS) may provide a suitable basis for the development and deployment of software as a service (SaaS). It needs to cover the requirements of the product in terms of deployed technologies and available cloud services. For the development of a MVP the decision to deploy a (certain) PaaS may be an early decision to take. Indeed, one of the principles of lean management is deferring decisions for being able to also consider new information.

As an alternative and as exercised in the demonstration, the entire cloud stack from IaaS to SaaS – realizing the MVP – needs to be described for cloud provisioning and deployed. This is particularly interesting when customized cloud stacks are preferred over a uniform cloud stack or a PaaS. At the same time the specification and deployment of a customized cloud stack, e.g., in TOSCA without further tool support, requires expert knowledge. This burden may prevent a project to opt for a tailored cloud setup and hinder lean development.

For realizing the MVP the functional architecture of the cloud application needs to become manifested in tangible cloud services and resources. The mapping from ABBs to SBBs – performed by architects (indicated with Role `A`) – is shown in the top three levels of Figure 2. Describing the cloud provisioning, the services are associated with server instances

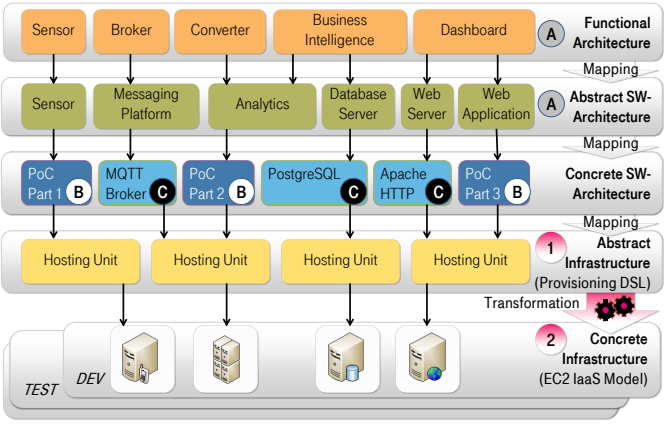

Figure 2. From Design to Deployment of the Cloud Application

via `hosting units`. Finally the development of the various parts of the MVP – as performed by software developers (Role `B`) and configuration management (CM) experts (Role `C`) – requires coordination between the roles. Finally their services have to be deployed (realized with Transformations `1` and `2`). As indicated at the bottom, this may occur for multiple stages of the engineering lifecycle (e.g., `DEV` or `TEST`).

A lack of automation renders the process – consisting of a multitude of steps – time-consuming and costly. As a result, iterative cycles involving the reprovisioning of (parts of) the service topology may be avoided. Decreasing agility, this may proof problematic at an early phase of product development such as when developing a MVP. Besides the loss of flexibility (e.g., as required when performing adjustments), the complexity of the overall process may render effective participation of architects problematic.

III. Domain-Specific Languages for Facilitating the Engineering of Cloud Applications

For addressing the issues mentioned in the introduction and as further elaborated in the previous section, a model-based approach is proposed. That is, separation of concerns is realized between architecture, functionality, and technology platforms while automating provisioning and deployment in a role-based development process. After giving an overview, the use of the DSLs is illustrated by means of the running example. The demonstrator entirely relies on open source software: The Eclipse Modeling Framework (EMF) [2] was chosen with Xtext for defining the DSLs and for realizing respective editors and Xtend for implementing the model transformations. At runtime and besides the resulting DSL editors, Git [3] is used as version control system (VCS), and Puppet [4] for the CM. Finally, OpenStack [5] serves as an IaaS solution.

[2] http://eclipse.org/modeling/emf
[3] http://git-scm.com
[4] http://puppetlabs.com
[5] http://openstack.org

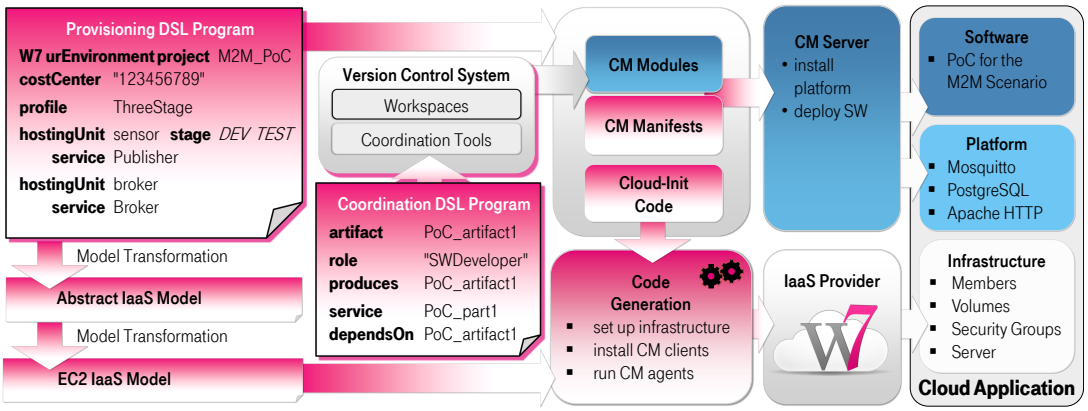

Figure 3. Overview of the Model-Based Approach for the Development and Continuous Deployment of Cloud Applications

A. Approach Overview

In addition to Figure 2, a technical overview of the model-based realization is depicted in Figure 3. The cloud provisioning is described using a DSL as shown in the upper left part of the figure. Through a sequence of model transformations an IaaS consumer is generated that realizes the provisioning of cloud infrastructure services. Platform and software services as referenced in the DSL program are provisioned using a CM system. CM modules are looked up for such services and a CM server is instructed accordingly while server instances are provisioned with CM clients.

A coordination DSL program declarativly describes roles, artifacts, services, and dependencies between these. Coordination tools are generated that interplay with a VCS and the workspaces of stakeholders. For developed services, CM modules are prepared in addition and built for further simplifying the overall engineering process while profiting from the functionality and automation as realized by the CM system.

B. Business Idea and Functional Analysis

Starting from the business idea from Section II, architects conduct a functional analysis. The resulting functional architecture comprises `sensors`, a `broker`, `converters`, a `business intelligence (BI)`, and a `dashboard` as shown in Figure 2. Resulting ABBs are `sensors`, a `messaging platform`, an `analytics` engine, a `database server`, a `web server`, and a `web application`. Using a DSL editor and profiting from syntax highlighting, code completion, and scoping as shown in Figure 4, the ABBs are associated with SBBs (i.e., Mosquitto as a MQ Telemetry Transport (MQTT) broker, a PostgreSQL server, an Apache HTTP server, and three different parts of a proof of concept (PoC) implementation). The abstract building blocks are so associated with concrete building blocks to be deployed and are used (i.e., referenced) for the provisioning as shown later.

C. Domain-Specific Language for Cloud Provisioning

The DSL program in the upper left part of Figure 3 shows an excerpt for defining the cloud provisioning. For

```
W7 urEnvironment catalogue
namespace M2M_PoC
serviceDef Publisher     realizedBy PoC_part1
    implies services MosquittoClient PyXB
serviceDef Broker
    implies services Mosquitto
serviceDef Analytics     realizedBy PoC_part2
    implies services MosquittoClient PyXB SQLAlchemy
serviceDef Database
    implies services PostgreSQL
serviceDef Report        realizedBy PoC_part3
    implies services SQLAlchemy ApacheWSGI
```

Figure 4. Mapping Architectural to Software Building Blocks

this, services are grouped in `hostingUnits`. Through model transformations and automation, the latter are mapped to server instances or clusters and appropriate IaaS clients are generated. Please note, that the services reference the abstract `serviceDefs` from Figure 4. This and further relationships (`implies`) are exploited for automating the provisioning. The DSL facilitates specification of custom cloud stacks as required for different stages of the engineering lifecycle such as development (`DEV`), testing (`TEST`), or production (`PROD`). Choosing a `profile` the provisioning is performed similarly for each of the defined `stages`. If desired this is done in different cloud regions (also defined in the respective `profile`). If not bound to certain `stages` (e.g., sensor data generators for development and test) server instances (e.g., broker) are provisioned for all stages (e.g., in all the related cloud regions).

D. Formalizing Dependencies of Services, Artifacts, and Roles

A declarative DSL allows to enumerate roles participating in the engineering process, artifacts, services, and their dependencies in a generic way. From these, coordination tools are generated (cf. [4]). Integrated into a VCS these may send out notifications when artifacts are available and synchronize workspaces accordingly. In addition to supporting the coordination, automation tools are derived from the DSL programs for preparing the deployment of developed services to the respective `hostingUnits` and for pushing and applying incremental changes through Puppet. Services that are devel-

oped such as the PoC parts are referenced by `serviceDefs` following the `realizedBy` keyword (see Figure 4).

E. Integration with Configuration Management Software

Higher-level cloud services depending on IaaS are provisioned through a CM system. For this, Puppet modules are looked up for services using name-based matching. Using the `realizedBy` keyword Puppet modules are automatically synchronized with required software artifacts using the VCS. Yet, module manifests need to be supplied by Puppet experts. At execution time, the approach integrates with Puppet by generating manifest files and cloud-init [6] configs (passed as user-data when launching server instances). This way the different engineers can work independently using their workspace focusing on their actual task while the packaging and deployment is automated.

F. Iterative Development and Continuous Deployment

When executing the generated IaaS consumer all the cloud services (see right box of Figure 3) – starting from IaaS (i.e., security groups, volumes, and instances) and PaaS (i.e., Apache HTTP server, Mosquitto, and PostgreSQL) to SaaS (i.e., PoC Parts 1–3) – are provisioned in an interplay with Puppet and thus the cloud application is deployed. If work is performed in the VCS, e.g., when modifying a particular service, changes can be applied incrementally using Puppet behind the scenes facilitating continuous deployment. Changes to the service topology are more critical: While in many cases they are respected by the demonstrator, a reprovisioning may become necessary, e.g., when renaming occurs. Please note that while the latter takes longer until all services are provisioned it is still fully automated and may be thus acceptable for prototyping.

IV. Related Work

Rapid application development (RAD) (cf. [8]) can help to obtain results in a timely and efficient manner. For instance, various RAD frameworks exist (cf. [9]) that are tailored towards web applications and that can be combined with the approach. For the development of cloud applications a development PaaS such as the Google App Engine [7] may provide ready to use (technology) stacks of services while easing the development of SaaS. In contrast this work permits to define and work with customized cloud stacks. Realizing separation of concerns it incorporates different roles in the engineering using tailored DSLs and coordinates workspaces. As it directly operates on IaaS deployments in an interplay with CM systems it is independent of a PaaS and certain (technology) stacks.

As mentioned, TOSCA provides a metamodel for service topologies. As a standard, it constitutes a desirable basis for an implementation of the approach presented. Similar to the presented work TOSCA can be combined with CM systems (cf. [10]). The DSL building on top of Amazon Elastic Compute Cloud (EC2) [8] concepts and used in this work for describing the provisioning permits to define rudimentary service topologies. While they can be mapped to TOSCA the DSL aims particularly at giving stakeholders access to the modeling. In fact, a provisioning can be described in a couple of lines and can be changed easily. Instead of the DSL shown the approach can use any modeling as long as it does not slow down development and iteration cycles. User studies have to be carried out in this regard in order to answer the question which modeling tools for service topologies are best for the development of cloud applications as required in prototyping.

[6] http://launchpad.net/cloud-init

[7] http://cloud.google.com/appengine/

[8] http://awsdocs.s3.amazonaws.com/EC2/latest/ec2-api.pdf

V. Conclusion

Agile development of cloud applications can be facilitated using a model-based approach, e.g., using textual DSLs as presented. For this a high degree of automation is required beyond provisioning. Also, as many stakeholders are involved in the engineering, it is crucial to harmonize their collaboration and to provide them with means for participation. This is realized by exploiting formalized dependencies and by providing tailored DSL editors while abstracting from technologies such as CM and IaaS solutions.

For wide applicability, the approach directly builds on top of IaaS and does not rely on a development PaaS. Yet, in case the code generator is adapted, the presented work can build on different application programming interfaces (APIs) or other technologies both for the provisioning and the CM. The approach was successfully adopted for the rapid development of several demonstrators and a PoC in an industrial context.

References

[1] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to Adapt Applications for the Cloud Environment," in *Computing*. Springer, 2013, vol. 95, pp. 493–535.

[2] N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg, "CloudMF: Applying MDE to Tame the Complexity of Managing Multi-cloud Applications," in *IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*. IEEE, Dec 2014, pp. 269–277.

[3] M. Iacob, D. Jonkers, H. Quartel, H. Franken, and H. van den Berg, *Delivering Enterprise Architecture with TOGAF and ARCHIMATE*. Enschede: BIZZdesign, 2012.

[4] T. Holmes, "Facilitating Development and Provisioning of Service Topologies through Domain-Specific Languages," in *18th IEEE International Enterprise Distributed Object Computing Conference Workshops and Demonstrations*, G. Grossmann, S. Hallé, D. Karastoyanova, M. Reichert, and S. Rinderle-Ma, Eds. IEEE, Sep. 2014, pp. 422–425.

[5] ——, "Automated Provisioning of Customized Cloud Service Stacks using Domain-Specific Languages," in *2nd International Workshop on Model-Driven Engineering on and for the Cloud*, vol. 1242. CEUR-WS.org, Sep. 2014, pp. 46–55.

[6] E. Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011.

[7] M. Poppendieck and M. A. Cusumano, "Lean software development: A tutorial," *IEEE Software*, vol. 29, no. 5, pp. 26–32, 2012.

[8] P. Beynon-Davies, C. Carne, H. Mackay, and D. Tudhope, "Rapid application development (RAD): An empirical review," *European Journal of Information Systems*, no. 8, pp. 211–223, 1999.

[9] I. Vosloo and D. G. Kourie, "Server-centric web frameworks: An overview," *ACM Comput. Surv.*, vol. 40, no. 2, 2008.

[10] J. Wettinger, M. Behrendt, T. Binz, U. Breitenbücher, G. Breiter, F. Leymann, S. Moser, I. Schwertle, and T. Spatzier, "Integrating Configuration Management with Model-Driven Cloud Management based on TOSCA," in *3rd International Conference on Cloud Computing and Services Science*, F. Desprez, D. Ferguson, E. Hadar, F. Leymann, M. Jarke, and M. Helfert, Eds. SciTePress, 2013, pp. 437–446.