Towards a Methodology to Form Microservices from Monolithic Ones

Gabor Kecskemeti[1(✉)], Attila Kertesz[2,3], and Attila Csaba Marosi[2]

[1] Liverpool John Moores University, Liverpool, UK
g.kecskemeti@ljmu.ac.uk
[2] Institute for Computer Science and Control,
Hungarian Academy of Sciences, Budapest, Hungary
{kertesz.attila,marosi.attila}@sztaki.mta.hu
[3] University of Szeged, Szeged, Hungary
keratt@inf.u-szeged.hu

Abstract. Cloud computing is the cornerstone for elastic and on-demand service provisioning to achieve more efficient resource utilisation and quicker responses to varying application loads. Virtual machines, one of the building blocks of clouds, can be created using provider specific templates stored in proprietary repositories, which may lead to provider lock-in and decreased portability. Despite these enabling technologies, large scale service oriented applications are still mostly inelastic. Such applications often use monolithic services that limit the elasticity (e.g., by obstructing the replicability of parts of a monolithic service). Decomposing these services to smaller, more targeted and more modular services would open towards elasticity, but the decomposition process is mostly manual. This paper introduces a methodology for decomposing monolithic services to several so called microservices. The proposed methodology applies several achievements of the ENTICE project: its image synthesis and optimisation tools. Finally, the paper provides insights on how these achievements help revitalise past monolithic services, and what techniques are applied to aid future microservice developers.

1 Introduction

Cloud computing enables elastic and on-demand service provisioning by building on the achievements of virtualisation technologies. Virtual machines, or in short VMs, are software constructs that mimic real-life hardware with the help of hypervisors, also known as virtual machine monitors. VMs open up possibilities like improving resource utilisation (e.g., by server consolidation) and adapting applications to varying application loads by scaling them up or down. VMs can be created using provider specific templates and virtual hard disk files (so called virtual machine images) stored in proprietary repositories. The creation process

This research work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644179 (ENTICE).

F. Desprez et al. (Eds.): Euro-Par 2016 Workshops, LNCS 10104, pp. 284–295, 2017.
DOI: 10.1007/978-3-319-58943-5_23

of VMs depends on the applied cloud and virtualisation technique in particular, but as well as on the application to be hosted in the VM.

These virtualised environments host a wide range of services, but are mostly delivered as a monolithic block composed of multitude of sometimes vaguely related functionalities. Unfortunately, because of the monolithic nature of these services, creating VMs hosting them costs significant amounts of time. Also, the user needs to instantiate a VM that host a complete monolithic service regardless of whether he/she needs only a subset or one of the offered functionalities. This results in large portions of the VM left unused, since the rest of the functionalities are not needed by users. The concept of microservices were proposed [13] to avoid these problems. This concept ensures that there is only a single, well defined functionality offered by a particular VM and its image is optimised just to host this functionality.

Namiot and Sneps-Sneppe [8] defined microservices as lightweight and independent services that perform single functions collaborating with other similar services through a well-defined interface. On the contrary, in monolithic architecture, services are deployed as united solution called a monolith. Its main drawback is the large code base and complexity, which erodes modularity and hinders productivity. The authors also argued that splitting up monoliths to microservices can result in a more manageable and scalable application.

Creating virtual machine images for such microservices is mostly done manually by skilled developers and it is a tedious task. Generally, the building process is done through the following distinct approaches: (*i*) developing a new system just for the necessary functionality, (*ii*) manually selecting parts of a previously created and widely used monolithic service (that is often integral part of a company's business process) until it mostly contains the desired functionality. In the first case, the past legacy service functionality is replaced with a new one, which might not fit well into the current business processes. In the second case, the manual code clean-up procedure often overlooks significant parts of the monolithic service thus the procedure does not necessary lead to the level of microservices (i.e., the resulting VM image might retain some unrelated features).

The goal of this research is to propose a methodology that can be used to split up a monolithic service to small microservices. These later can be used to increase the elasticity of large scale applications, or to allow more flexible compositions with other services. To achieve this, we incorporate several techniques to the microservice creation process: (*i*) we present a recipe based generic image creation service that is capable to create VM and container images crafted for particular cloud systems, (*ii*) we reveal how a dynamic, live-evaluation based image size optimisation technique could be utilised to create a family of images based on the previous monolithic service, and (*iii*) we show how this image family can be turned to a set of microservices within the ENTICE environment.

The remainder of this paper is as follows: Sect. 2 presents related work, then Sect. 3 introduces the ENTICE project. Section 4 introduces the proposed methodology, detailing the recipe-based image synthesis and image size optimizations. Finally, the contributions are summarised in Sect. 5.

2 Related Work

To foster a more efficient and scalable cloud application management, the approach of composing microservices can be used [13]. Microservice building can be done by different tools, such as Puppet [10], Chef [2], and Docker [3]. These tools can cover the development and operation aspects of system administration tasks, such as delivery, testing and maintenance to improve reliability, security and so on. For example, Tihfon et al. [12] used Docker to deploy applications based on microservices. Gabbrielli et al. [5] proposed an automatic and optimised deployment of microservices written in the Jolie language. Their tool can automatically generate a fully detailed Service-Oriented Architecture configuration starting from an abstract description of the target application. In this paper, we focus on microservice image synthesis and optimisations during the creation process instead of optimisations applied during the deployment of the services.

Existing methods for VM image creation do not provide size and functional optimisation features other than dependency management, which is based on predefined dependency trees produced by third-party software maintainers. If a complex software is not annotated with dependency information, it requires manual dependency analysis upon VM image creation based on worst case assumptions and consequently. The resulting VM images are far from optimal size in most cases. On the other hand, optimising the size of existing images by aiming at providing only particular functionalities can be addressed with two approaches:

1. *The pre-optimising approach* requires the VM image developer to provide the application and its known dependencies prepared as reusable VM image components. The image developers select from these components so that they can form the base of the user application. These approaches then form the VM image with the selected reusable components and the service itself. For example, the company SAS [11] applied this algorithm with an extension that supports creating custom VM images by building from the source code. Other pre-optimising approaches determine dependencies within the VM image by using its source code using Software clone and dependency detection techniques [1]. Once the dependencies are detected, these approaches leave only those components that are required for serving the key functionality of the VM image. Optimising a VM image with these techniques requires the source code of all the software encapsulated within the image and to analyse the underlying systems.
2. *The post-optimising approach* uses existing but unoptimised VM images or, in the extreme case, optimised VM images with known software. To support this approach, several OS and application vendors offer the minimalist versions of their products packaged together with their Just-enough Operating System [6] using the Virtual Appliance approach. However, this approach requires the image developer to manually install its application to a suitable optimised VM image. The advantage of these approaches is the fast creation of the images but at the price that the developer has to trust the optimisation attempt of the used VM image's vendor. If the image is not well optimised, or the vendor

offers a generic image for all uses then the descendant VM images cannot be optimal without further efforts.

Existing research mostly focuses on pre-optimising approaches, which are not applicable to already available VM images. In ENTICE we use an VM synthesiser to extend pre-optimising approaches so that image dependency descriptions are mostly automatically generated.

3 The ENTICE Project

The ENTICE project [4] is a multidisciplinary team of computer scientists, application developers, cloud providers and operators with the aim to research a ubiquitous repository-based technology for VM and container image management called ENTICE environment. This environment proves a universal backbone for IaaS VM/container image management operations, which accommodate the needs for different use cases with dynamic resource (e.g., requiring resources for minutes or just for a few seconds) and other QoS requirements. As the discussed concepts are not dependent on the applied virtualisation technology, the rest of the paper uses the terms VM image and container image interchangeably.

The technologies developed by the ENTICE project are completely decoupled from the particular applications and their runtimes. Despite the decoupling, ENTICE still provides constant support for applications via optimised VM image creation, assembly, migration and storage. ENTICE expects users to provide their original and functionally complete VM or container images. Then it transparently tailors and optimises the images for user targeted Cloud infrastructures with respect to their size, configuration, and geographical distribution. As a result of the optimisation, these images are dispatched to the clouds (even across Clouds), executed faster and they have a potential for QoS improvement. ENTICE stores metadata about the images and fragments in a distributed knowledge base to be used for interoperability, integration, reasoning and optimisation purposes (e.g., supporting decisions about replica locations for high demand images and also decisions about the time instances at which an image should be replicated).

In the following we list the main ENTICE objectives:

1. The distributing Virtual Machine and Container Images (VMIs) with the ENTICE repository;
2. The analysis and synthesis of VMIs for already existing services and functionalities;
3. An image portal in association with a knowledge base, composing together the components of the projects' distributed, highly optimised repository.

Albeit there could be numerous stakeholders in the cloud computing context, the project aims at the following list of stakeholders specifically who should directly benefit from the distributed image repository built by ENTICE:

– End-customers, such as the users of the satellite image service of Deimos[1] should not be aware of the Deimos's use of the ENTICE repository environment. On the other hand, they should still benefit from the better Quality of Service (QoS) in the runtime of Deimos's applications as a result of the ENTICE applied optimisations.
– Cloud Application Providers and/or Software as a Service (SaaS) providers, such as the company Wellness Telecom[2], are offering SaaS applications utilizing the Cloud to serve many of their customers;
– Application Developers, such the above mentioned Deimos, who aim at deploy and run their applications with high efficiency. For example, Deimos is operating a satellite and is in great need for such deployment optimisations for its Earth observation platform and its customers (i.e. the previously discussed end-customers);
– Cloud Operators, such as the well-known company Flexiant[3] which has several offerings in the area of managing cloud applications across multiple clouds;
– Cloud Providers, such as Amazon EC2[4] could benefit through incorporating ENTICE technologies in their VMI storage and management solution, or even if just their customers are applying ENTICE optimizations on their images.

4 The Proposed Methodology

Our goal set out for this research was to identify a simple to follow methodology usable fragment a monolithic application alongside its sub-service boundaries. Allowing these sub-services to act as small micro-services that later can be composed to other services (without the need of the entire monolithic application). The original monolithic image can then act as an shared base for its derived micro-service family. To achieve this, we use image synthesis and image analysis methods which both have pivotal roles within the architecture of the ENTICE project.

Our VMI synthesis mechanism enables users to build new images with several approaches. First, it allows the use of generic user provided images or software recipes to act as the foundation before specialising them into micro services. Next, VMI synthesis cooperates with the ENTICE image portal (the main GUI for image creation and distribution procedures) to identify the functional requirements a newly created image must meet (this must be done on a per micro service basis - i.e., resulting a new image from every functional requirement specified). Then, our synthesis tool modifies the generic images (from the first step) either directly (by altering the image file(s)) or indirectly (through creating alternative recipes that lead to more compact images). These alterations aim at removing contents from the original images. Thus the alterations lead the generic images

[1] http://www.deimos-space.com/.
[2] http://www.wtelecom.es/.
[3] https://www.flexiant.com/.
[4] http://aws.amazon.com/ec2.

towards their single purpose: namely, the functional requirements listed against the image in the portal. For optimised images, VMI Synthesis offers image maintenance operations (e.g., allowing software updates to be done on the original image and transforming those updates to the optimised image).

Alongside synthesis, ENTICE also delivers VMI analysis allowing the discovery of equivalent pieces in apparently non-related VM images which were sometimes even received from different stakeholders or communities. Analysis operates independently of the cloud provider where the image is stored. Equivalence information then stored in the ENTICE knowledge base for later use. ENTICE also allows splitting VM images into smaller fragments allowing the storage of the frequently shared image components only once (e.g., a particular flavour of Linux used by two different images). Fragmenting fosters the VM image distribution and enables the optimization of overall storage space throughout the distributed repository.

As fragmented images would not be directly usable in clouds, the ENTICE environment offers virtual machine management templates (so called VMMTs) to be stored in the repositories of the connected cloud systems. These VMMTs allow the fragmented images to be reconstructed at runtime. For optimal VM instantiation performance, the templates are formulated as stand-alone VM/Container images solely having functionality to access and build fragments from the project's distributed repository. After a VM is instantiated using a VMMT, it will ensure fragments (needed for a particular functionality specified by the user) are placed and enabled for use in the instantiated VM. VMMTs even allow customisation of files/directories for specific VMs in accordance of the needs of various stakeholders. The functions of the VMMTs are underpinned by user-defined functional and non-functional descriptions about the application to be deployed with the help of the ENTICE knowledge base and its reasoning mechanisms.

In Fig. 1, we reveal the use case diagram for ENTICE's image synthesis. The nodes (use cases) of the diagram were derived from the comprehensive requirement set (i.e., both originated from pilot cases and architectural ones) and the foundational principles of the project objectives. As a result, these use cases are expected to cover the requirements and project objectives where applicable, while they are strictly limited to image synthesis and analysis aspects. The Application Developer is expected to behave as the key actor who interacts with most use cases and can initiate most activities. Apart from the developer, we also expect Service Providers to use our image synthesis solution when they decide whether they should to adopt a particular service and image version. We also expect ENTICE's image distribution component to interact with the optimiser if it foresees potential for more optimal delivery by automatically continuing the optimization of not yet completely optimised images. In the coming subsections, we describe the most relevant use cases by considering and discussing their requirements (and some of their specific aspects), and then revealing our plans to fulfil them.

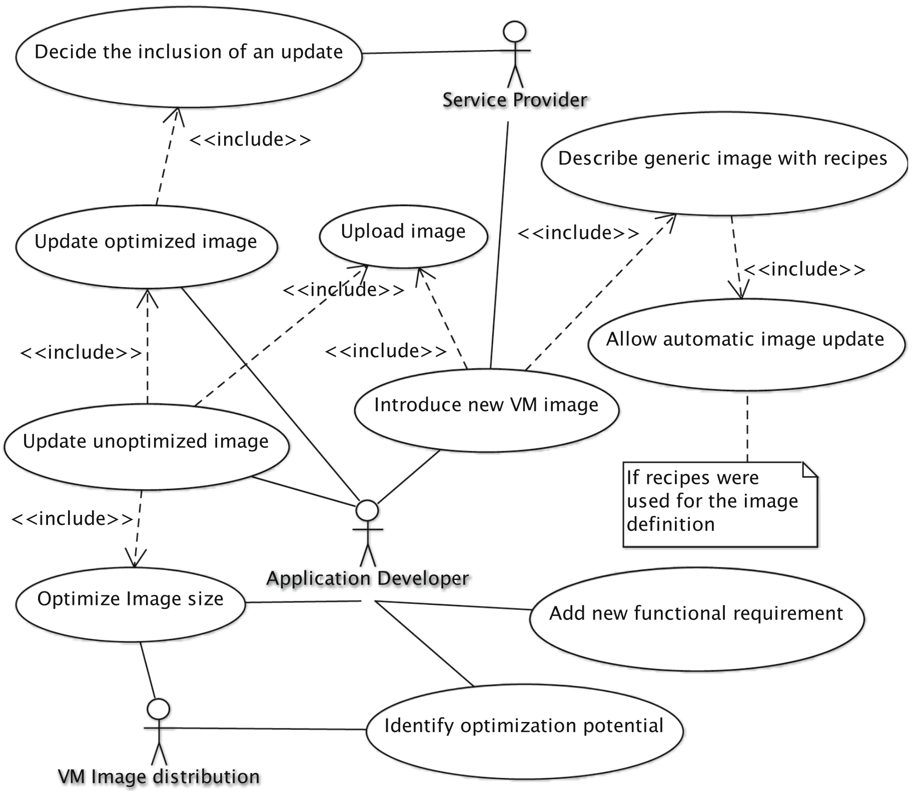

Fig. 1. Detailed use cases of image synthesis

4.1 Recipe Based Image Synthesis

This section mainly focuses on the use cases of “Describe generic image with recipes” and “Introduce new VM image” (see Fig. 1). These use cases focus on the application developer’s activities when he/she wishes to build a set of cloud provider specific VM/Container images. The use cases discuss the ways developer provided recipes used to create new images with the help of devops concepts. On this use case level, the recipes are expected to guide the creation of the developer’s original monolithic service on a generic way.

The recipe based image synthesis process of ENTICE can be seen in Fig. 2. It depicts 7 steps starting by creating an image, and ending with an optional cancel request. ENTICE provides APIs in a REST interface to use the services covered by these steps. There is also a backend part of this Synthesis service that uses other subcomponents to create the requested images. The images that can be managed in these processes may be of normal virtual machines (e.g. VMIs) or containers. The contents can also vary from microservices to complex ones. As they suggests, microservices in containers have smaller footprints, therefore they are easier to optimize. As shown in Fig. 2 the API enables the following processes:

- 1: submission of build requests;
- 5: retrieve build results;
- 6: query the status of the builds (optional);
- and 7: cancel ongoing builds (optional).

The image creation process at the backend consists of two parts. The first one is the building phase, while the second is the testing phase. First let's detail the building phase. It can be initiated with the create API call (step no. 1 in Fig. 2) by specifying the build target with its parameters, and the service description for the provisioning step, and the test cases for the testing phase.

The first part of the building phase is the bootstrapping step (no. 2). It is responsible to make a base image (in case of VMI's) or a container available for the provision step. It is possible to create them in the following ways:

- from scratch (with tools QEMU/QCOW2);
- targeting a container build (e.g., Docker);
- or using an existing one from a cloud image repository (e.g., Amazon Web-Services or OpenStack).

In case of QEMU/QCOW2, the build target Debian and Red Hat derived distributions are supported.

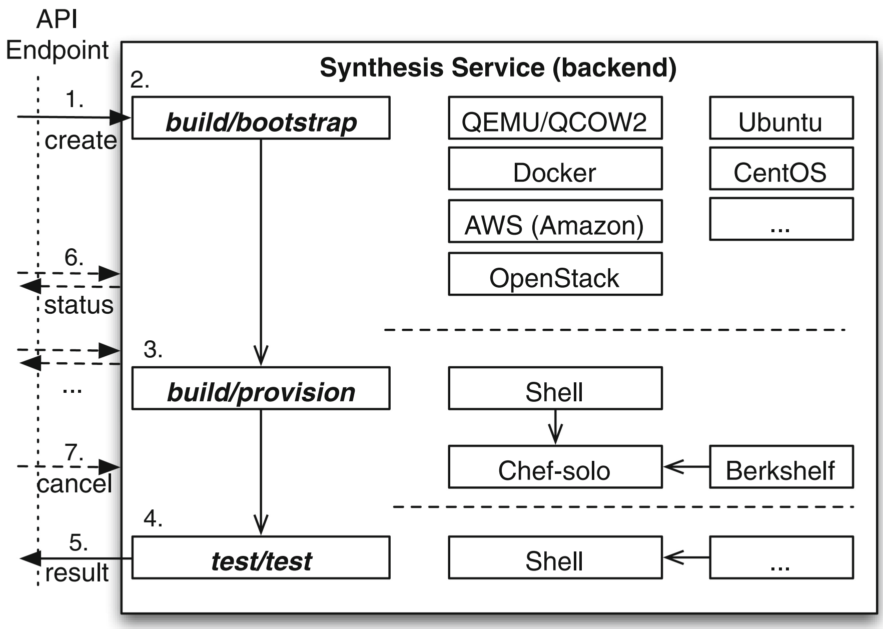

Fig. 2. Process of recipe based image synthesis

The first part of the building phase is the provisioning phase, which responsible for installing the requested microservice by the specified description. It can be done in two ways. First, a custom shell script can be provided containing sequential steps to be executed. Another option is to use Chef-solo, where Chef cookbooks must be provided (e.g. retrieved via Berkshelf) or a custom one. These targets can also be used together when needed, e.g., performing basic maintenance via Shell and deploying the requested microservice components to the image via Chef.

In the testing phase, the image is duplicated, and the supplied test script is executed in the copied image. The testing methods can be of any type, only the exit status is what matters: zero means everything went fine, non-zero denotes an error. The script can deploy any packages from the Linux distribution repository and beside the shell script a custom zip file can be supplied containing additional testing tools, but no other external access is allowed for security reasons. The methods to be used in the testing phase are very flexible, since different services require different methods or tools to be tested. The copied test image is discarded after the tests, and the original one will be available for download. Currently there is no option to link the image to another location or repository, this feature will be considered for future work. Our current implementation relies on ImageFactory [7] and Packer [9].

4.2 Targeted Size Optimisation

In this subsection we detail and exemplify the "Optimize Image size" case of Fig. 1. This optimization process can be executed once the recipe based synthesis is finished resulting in several VMIs or container images composing a monolithic service. We refer to these composing images as original images of an application. Usually one of these images implements the functionality of a microservice, therefore the user can use the ENTICE environment to transform such original image to an optimized one that holds only the intended microservice functionality. The steps needed for this transformation are depicted in Fig. 3. Before the microservice can be extracted from the original image the user, who knows the application behavior, need to prepare a functionality test for the required microservice (in the form of self-contained shell scripts without any dependencies), as shown in Fig. 1 with the "Add new functional requirement" case. Such tests should utilise all features of the required microservice, and generally they can be constructed from unit tests of the original, composing application. Hence these scripts test the intended functionalities of a microservice, they needs manual preparation, but they are sufficient to be used in proof of concept scenarios. In our future works we will develop techniques to describe the functionality of an intended microservice, in order to enable automatic test script generation.

Once the functionality test is made available in the ENTICE Image portal, the pre-evaluation phase can be started, where the original image is instantiated in a minimal cloud infrastructure (which is part of the ENTICE environment), as depicted in step 1 of Fig. 3. To this end a virtualised environment (VE) is set up by instantiating a new VM or container with its filesystem instrumented for

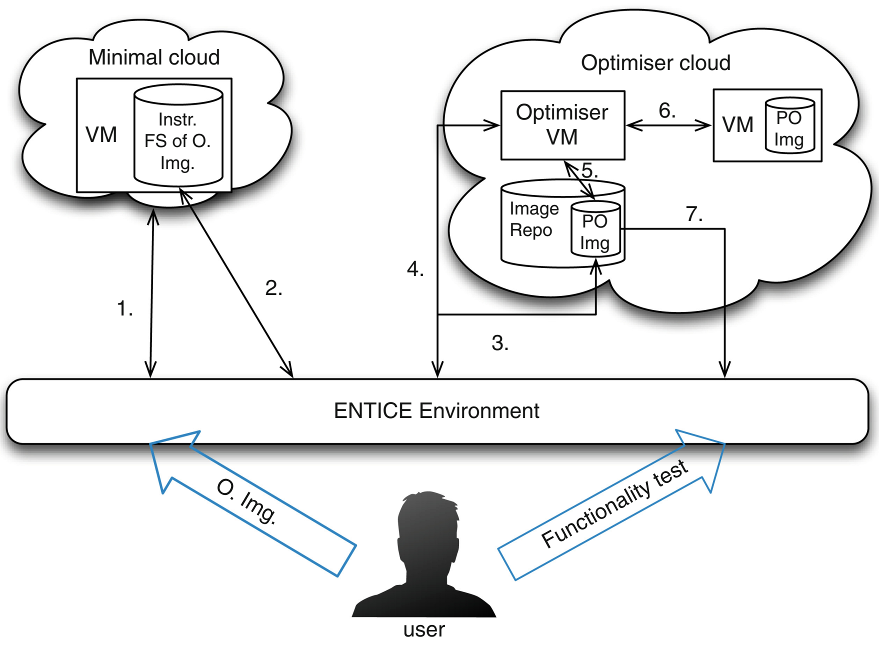

Fig. 3. Steps to transfrom an image to host only the intended microservice

read operations (called Instr. FS in the figure). Once the ENTICE environment starts, it collects the VE's read access operations to its disks (step 2 in Fig. 3).

Besides this data collection process, the microservice's functionality test is also executed by pointing its shell script to the VE's host. If the test fails after the execution, the collected data is discarded and the user is notified about the incorrect test result for the original image. If the test is successful, the collected data (representing the list of read blocks in the VE) is transformed to reflect individual files in the original image. The list of files acquired during this transformation is the so called restricted list.

The next step is the image optimisation phase. From now on, we assume that files that are not referenced by the restricted list are not relevant for the actual microservice. This means that in step 3 of Fig. 3 the system uploads a partially optimised image (PO image) that contains only the registered files (thus all unreferenced files are deleted from it). In step 4, an Optimizer VM is deployed and contectualized to use this image, and to perform the optimisation procedure by executing a test script in step 5. Here it analyses the remaining contents of the PO image and selects parts of the image that can still be removed. These newly selected parts should also be not relevant for the microservice's intended functionality, instead they are believed to be used by background activities of the original image (e.g., startup procedures and periodic activities unrelated to the core functionality). Since in this paper we present and describe the methodology (inner workings) of the ENTICE environment, we do not introduce specific selection techniques to applied on the PO image for further optimisations.

Once the PO image is modified and additional files are removed, the Optimiser VM uploads the new image to the cloud in step 6, and tests the image by instantiating it and evaluating its VE via the user-provided shell script. In case the evaluation is successful, the newly uploaded image will be taken as a new PO image instead of the previous one. It the evaluation fails, the selection technique is restarted with the previously examined PO image. These optimization processes are repeated until the user-defined cost limits are not achieved, or till no more selectable image parts are found. By the end of this step the final PO image will be ready (containing only the intended functionality of the microservice), which is given back to the ENTICE environment in step 7.

It may happen that the user wants to alter the interface of this optimized image after the optimization process. Generally the microservice offers a minimised feature set compared to the original image, so it is reasonable to reduce the interface, too. In this case the optimisation phase should be rerun with the altered interfaces on the original image, but the process will be much faster, since past selection errors are saved and reused by the system.

5 Conclusion

Virtual machine and container images are generally created by provider-specific templates stored in proprietary repositories, which may lead to provider lock-in and decreased portability. Despite these enabling technologies, large-scale service-oriented applications are still mostly inelastic due to the robust services they create. In this paper we introduced image repository management of multiple federated clouds in the frame of the ENTICE project, which tries to address this issue by transfroming monolithic services to microservices. Hence, we provided a methodology for microservice creation with an image synthesis approach, which can be used to create optimized images in a distributed repository.

In the future we will work on generalizing monolithic service fragmentation to support such monolithic services that cannot be decomposed without introducing alternative protocols in the communication between the fragmented microservices. We also envision further optimisations of microservice delivery by identifying common parts of microservice in the form of custom virtual machine management templates. Such templates would allow better image part selection and faster optimisations.

References

1. Belguidoum, M., Dagnat, F.: Dependency management in software component deployment. Electr. Notes Theor. Comput. Sci. **182**, 17–32 (2007)
2. Chef: http://www.getchef.com, May 2016
3. Docker: https://www.docker.io, May 2016

4. ENTICE consortium: Entice project website. http://www.entice-project.eu/, May 2016
5. Gabbrielli, M., Giallorenzo, S., Guidi, C., Mauro, J., Montesi, F.: Self-reconfiguring microservices. In: Ábrahám, E., Bonsangue, M., Johnsen, E.B. (eds.) Theory and Practice of Formal Methods, pp. 194–210. Springer, Heidelberg (2016)
6. Geer, D.: The OS faces a brave new world. Computer **42**, 15–17 (2009)
7. Image Factory: http://imgfac.org/, May 2016
8. Namiot, D., Sneps-Sneppe, M.: On micro-services architecture. Int. J. Open Inf. Technol. **2**(9) (2014)
9. Packer: https://www.packer.io/, May 2016
10. Puppet: http://puppetlabs.com, May 2016
11. SAS: rBuilder. http://www.sas.com/en_us/software/sas9.html, May 2016
12. Tihfon, G.M., Kim, J., Kim, K.J.: A new virtualized environment for application deployment based on Docker and AWS. In: Kim, K., Joukov, N. (eds.) ICISA 2016. LNEE, vol. 376, pp. 1339–1349. Springer, Heidelberg (2016)
13. Toffetti, G., Brunner, S., Blöchlinger, M., Dudouet, F., Edmonds, A.: An architecture for self-managing microservices. In: Proceedings of the 1st International Workshop on Automated Incident Management in Cloud, pp. 19–24. ACM (2015)