The ENTICE Approach to Decompose Monolithic Services into Microservices

Gabor Kecskemeti, Attila Csaba Marosi and Attila Kertesz
Institute for Computer Science and Control, Hungarian Academy of Sciences
1111 Budapest, Kende u. 13-17, Hungary
Email: {kecskemeti.gabor, kertesz.attila, atisu}@sztaki.mta.hu

***Abstract*—Cloud computing has enabled elastic and on-demand service provisioning to achieve more efficient resource utilisation and quicker responses to varying application loads. Virtual machines, 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 (leading 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 outcomes of the ENTICE project (namely its image synthesis and optimisation tools). Finally, the paper provides insights on how these outcomes help revitalise past monolithic services, and what techniques are applied to aid future microservice developers.**

***Index Terms*—cloud computing; microservices; service oriented architectures; virtual machine images**

I. Introduction

Cloud computing builds on the advances of virtualisation technologies to enable elastic and on-demand service provisioning. Virtual machines (software constructs that mimic real life hardware with the help of virtual machine monitors), or in short VMs, open up possibilities that for example improve resource utilisation (e.g, by server consolidation) or adapt applications (by scaling them up/down) to varying application loads. VMs can be created using provider specific templates (so called virtual machine images) stored in proprietary repositories. The creation process of these VMs depends on the applied cloud and virtualisation technique, as well as on the application to be hosted in the virtual machine.

Services are widely hosted in these virtualised environments, but they are mostly delivered as a monolithic block 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, when the user of the service would need only some of the offered functionalities, he/she still needs to instantiate a VM that hosts the complete monolithic service. As the rest of the functionalities are not needed by users, large portions of the VM are left unused. To avoid these problems, the concept of microservices were proposed [12]. 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 et al. [11] defined microservices as lightweight and independent services that perform single functions collaborating with other similar services through a well-defined interface. The opposite approach is the so-called monolithic architecture, in which services are deployed as a united solution called a monolith. Its main drawback is the large code base, which slows the productivity and erodes modularity. They 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 a tedious task and it is mostly done manually by skilled developers. Generally, the creation process is done through the following distinct approaches: (i) developing a new system just for the necessary functionality, (ii) manually selecting parts of the code 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 cleanup 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 that 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/container images crafted for particular cloud systems, (ii) we reveal how a dynamic, live-evaluation based VM/container image size optimisation technique could be utilised to create a family of VM/container images based on the previous monolithic service, and (iii) we show how this VM image family can be turned to a set of microservices within the ENTICE environment.

The remainder of this paper is as follows: Section II presents related work, then Section III introduces the ENTICE project. Section IV introduces the proposed methodology, detailing the recipe-based image synthesis and image size optimizations.

978-1-5090-2088-1/16/$31.00 ©2016 IEEE

Finally, the contributions are summarised in Section V.

II. Related work

To foster a more efficient and scalable cloud application management, the approach of composing microservices can be used [12]. Microservice creation can be done by orchestration tools, such as Puppet [6], Chef [7], and Docker [8]. These tools 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. [13] used Docker to deploy applications based on microservices. Gabbrielli et al. [14] 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. The first one, 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 [2] 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 [3]. 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.

The second, 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 [4] 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.

III. The ENTICE project

The ENTICE project [1] 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 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 ENTICE technology is completely decoupled from the applications and their specific runtime environments, but continuously supports them through optimised VM image creation, assembly, migration and storage. It is designed to receive unmodified and functionally complete VM images from users, and transparently tailor and optimise them for specific Cloud infrastructures with respect to their size, configuration, and geographical distribution, such that they are loaded, delivered (across Cloud boundaries), and executed faster and with improved QoS compared to their current behaviour. ENTICE will gradually store information about the VMI and fragments in a knowledge base that will be used for interoperability, integration, reasoning and optimisation purposes (e.g. repositories should decide at which other repositories one needs replicas of a heavily requested image and at which time such an image is replicated).

ENTICE has the following relevant goals:

1) The distribution of Virtual Machine Images and Container Images (VMIs) in the ENTICE repository;
2) The VMI analysis and synthesis;
3) The VMI images portal and its associated knowledge base, acting as glue for the distributed, highly optimised repository.

There are various individuals and organisations that may be considered as stakeholders in the cloud computing domain, and may be highly interested in in a distributed image repository built ENTICE. The following stakeholders are participants of the ENTICE environment:

- End-customers, such as the users of the satellite image service of Deimos[1] will not notice the presence of the

[1] http://www.deimos-space.com/

ENTICE repository environment, but will experience a better Quality of Service (QoS) in the runtime of their applications due to the optimisations applied by ENTICE in the background.

- Cloud Application Providers and/or Software as a Service providers, such as the company Wellness Telecom[2], which offers SaaS applications deployed in the Cloud to many customers;
- Application Developers, such as the company Deimos, which is operating a satellite and is in great need to develop and deploy a highly efficient cloud application for Earth observation for its customers (i.e. end-customers);
- Cloud Operators, such as the well-known company Flexiant[3] that pioneered its solutions for the management of Cloud applications across multiple Clouds;
- Cloud Providers, such as Amazon EC2[4] will benefit by integrating ENTICE as their VMI storage and management solution, or if their customers are willing to use ENTICE.

IV. The Proposed Methodology

The goal of this research is to propose a methodology that can be used to split up a monolithic application to small microservices that later can be composed to other services. This monolithic application can then act as an ancestor of the family of those small microservices that are derived from it. To achieve this, we use image synthesis and image analysis techniques that play a central role in the ENTICE architecture.

The VMI synthesis tool allows the creation of new virtual machine images with several approaches. First, it allows the use of generic user provided images or software recipes to act as the foundation of specialisation. Next, the synthesis tool collaborates with the ENTICE image portal (the graphical user interface for the image creation and distribution process) to identify the functional requirements a newly created image must meet. Then the synthesis technique alters the user provided images either directly or indirectly (through recipes). These alterations target partial content removal from the original images allowing them offering only their single purpose, namely the functions identified in the image portal. Once the initial optimised image is ready, the VMI Synthesis will offer image maintenance operations (like managing software updates on the image).

Next to synthesis, ENTICE uses a VMI analysis functionality to allow discovery of identical portions in apparently unrelated VM images coming from even different stakeholders and communities, regardless of the cloud provider where they are physically stored. This information is automatically stored in the ENTICE knowledge base for later use. The environment also allows splitting of VM images into fragments for storing the frequently shared image components only once (e.g. a particular flavour of Linux used by two different images). This operation allows the VM image distribution component

[2]http://www.wtelecom.es/
[3]https://www.flexiant.com/
[4]http://aws.amazon.com/

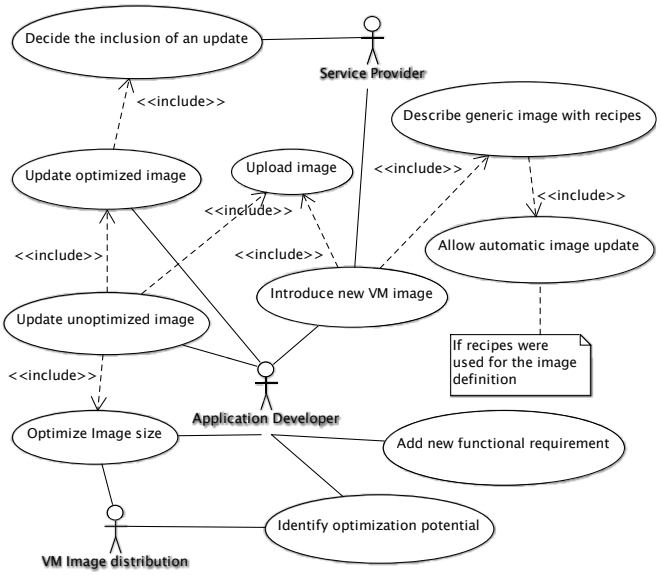

Fig. 1. Detailed use cases of image synthesis

to optimize the overall storage space throughout the distributed repository.

To enable the use of the fragmented VM images, the ENTICE environment distributes virtual machine management templates (so called VMMTs) to the various repositories of the connected cloud systems. These VMMTs allow VMs to be assembled at runtime from the previously identified fragments. The templates are stand-alone VM images containing the necessary components to access the project's distributed repository. After a VM is instantiated from a VMMT, it will customise the contents of the instantiated VM with the VM fragments required to match the user's functional requirements (even allowing new files/directories be placed in particular VMs to meet the demands of the various stakeholders).

These methods are supported by user-defined functional and non-functional descriptions about the application in the ENTICE knowledge base and the implemented reasoning mechanisms that feed in necessary information for the decision making process.

Figure 1 presents a use case diagram for image synthesis. The nodes (use cases) in this diagram were distilled from overall requirements (both pilot cases and architectural ones) and overall principles of the project objectives. In other words, these use cases must cover the requirements and the project objectives where applicable, with the focus strictly set to image synthesis and analysis aspects. We envision the Application Developer as the central actor that interacts with most of the use cases and can initiate most of the activities. Apart from the developer, we expect that Service Providers could also utilise the image synthesis solution of ENTICE by deciding when to adopt a particular service and image version. We also expect ENTICE's image distribution component to interact with the optimiser functionality when it recognises the chances of more optimal delivery by automatically continuing not completely

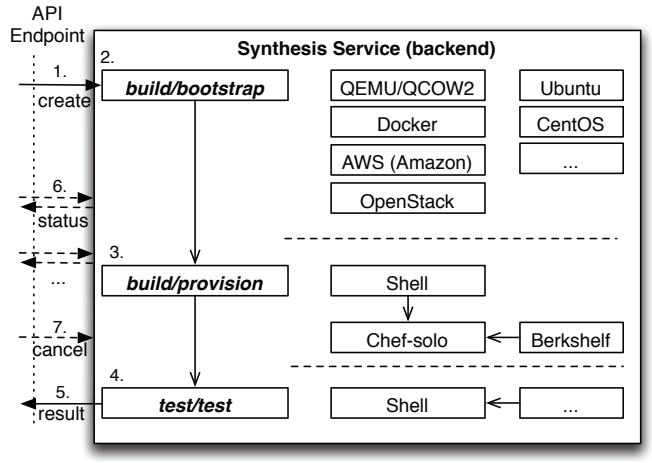

Fig. 2. Process of recipe based image synthesis

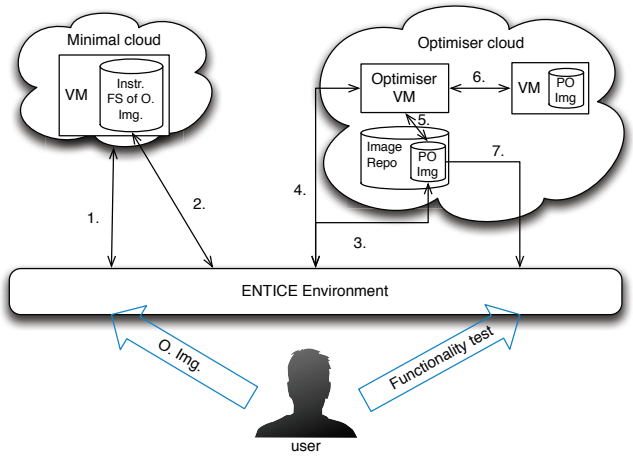

Fig. 3. Transfroming the original image to only host the intended microservice

optimised images. In the following, we list the descriptions of the most prominent use cases by paying particular attention to their requirements (or some of their specific aspects), and how we plan to achieve them.

A. Recipe based image synthesis

This section mainly focuses on the use cases of "Describe generic image with recipes" and "Introduce new VM image" (see Figure 1). In these use cases the application developer creates a set of VM images specific to the aimed cloud providers. The images are created through developer provided recipes which use devops concepts to define the creation of the monolithic service on a generic way.

Figure 2 depicts the process of recipe based image synthesis in ENTICE. Images can be traditional VMI's and containers as well. These images may contain complex services or microservices. However the benefit of containers (with microservices) is a smaller footprint and less complex services that are easier to optimize. The synthesis service consists of an API with a REST interface and a backend that creates the requested images.

The API allows the submission of build requests (see 1 in Figure 2); query the status of builds (see 6 in Figure 2); cancel ongoing builds (see 7 in Figure 2); and retrieve build results (see 5 in Figure 2). The creation request must contain the build target (e.g., QEMU/QCOW2) with its parameters (e.g., disk layout and Linux distribution for QEMU/QCOW2; or infrastructure dependant VMI identifier for the base image in case of OpenStack); the service description for the provision step; and the test cases for the test phase. These are going to be detailed next.

The image creation process at the backend consists of two main phases. First there is a build phase, followed by a test phase. The build phase itself consist of two steps. The first step is the bootstrap step (see 2 in Figure 2). It is responsible for providing a base image (in case of VMI's) or a container for the provision step. Supported methods are creating one from scratch (see QEMU/QCOW2 in Figure 2); using a prepared existing one from a cloud image repository (e.g., Amazon WebServices or OpenStack); or targeting a container build (e.g., Docker). In case of the QEMU/QCOW2 build target Debian and Red Hat (via kickstart) derived distributions are supported. The provision phase is responsible for installing the requested microservice using the provided description. Here currently two methods are available. First a custom shell script can be provided (see "Shell" in build/provision in Figure 2). This should contain sequential steps to be executed. Another option is to use Chef-solo (serverless Chef). Here a list of Chef cookbooks must be specified (cookbooks are retrieved via Berkshelf) or a custom cookbook. In the provision step Chef-solo and Shell targets can be used together when needed, e.g., perform basic maintenance via Shell and the deploy the requested microservice components to the image via Chef.

After the build phase finished successfully, the testing phase begins. The image is copied and the supplied test script is executed in the copy. The method of testing can be chosen freely as the backend only monitors the exit status of the test script: zero exit status is assumed everything went fine, non-zero denotes an error. The script can freely deploy packages from the Linux distribution repository (although for security reasons no other external access is allowed) and beside the shell script a custom tarball/zip file can be supplied that may contain additional testing tools. The testing phase provides as much freedom as possible since different services require different methods or tools for testing them. After the tests finish, the test image is discarded and the original one is made available for download. (A push option to desired locations/repositories is planned for the future).

Our initial version for the build/bootstrap step (see Figure 2) used the ImageFactory [9], but our current implementation relies on Packer [10].

B. Targeted Size Optimisation

This subsection has the use case of "Optimize Image size" from Figure 1 as its primary target. It assumes that the

recipe based synthesis technique already created several virtual machine or container images for the application developer with the intended monolithic service. We call these images the original images. Once there is an original image available that incorporates the functionality of the microservice, the user (i.e., the microservice developer) can instruct the ENTICE environment to create customised VM/container images that only focus on the intended microservice's functionality. This process is shown in Figure 3.

To do so, the user first prepares a functionality test for each microservice to be extracted from the original image (this first step is represented in the use case of "Add new functional requirement" in Figure 1). This test is expected to utilise all features of the intended microservice. The test could be derived from a previously developed unit or similar test set. Currently, the ENTICE environment expects this test to be specified in a self contained shell script (such scripts have no dependencies) that can be instructed to evaluate a directly accessible network host for the intended microservice's feature set. Note that although the script is a rudimentary technique to specify the evaluator for the intended microservice, this is already sufficient for proof of concept scenarios. In future works, we envision techniques that could be used for describing the functionality of the intended microservice on a automatically evaluable way.

Once the functionality test is pushed to the ENTICE Image portal, the system switches to the pre-evaluation phase. In this phase, a minimal cloud infrastructure is used to instantiate the original image (see step 1 in Figure 3, shortened as O. Img.). This minimal cloud infrastructure is part of the ENTICE Environment and it is specially prepared: when it instantiates a new VM/container (virtualised environment - VE) it ensures that its filesystem(s) are instrumented for read operations (called Instr. FS in the figure). Once the original image is instantiated in this minimal cloud, the ENTICE environment starts to collect the VE's read access operations to its disks (shown as step 2 in Figure 3). Alongside the data collection, the microservice's functionality test is also started by pointing its shell script to the VE's host. After the completion of the test the VE and the data collection is terminated. If the test fails, the collected data is discarded and the user is notified about the incorrect test script and/or original image. If the test is successful, then the collected data (which are usually just a list of blocks read throughout the lifetime of 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.

After the restricted list is prepared the system switches to the image optimisation phase. During this phase, it is assumed that files not referenced by the restricted list are not relevant for the microservice, so they are dropped immediately resulting in a new partially optimised image (PO. Img. in the figure). The rest of this phase will use such partially optimised images. As seen in step 3 in Figure 3, the partially optimised image is uploaded to the cloud service to be used for the optimisation procedure (note that this cloud could be different from the previously used one and it is selected by and paid for the user). Next, the ENTICE Environment instantiates an optimiser virtual machine in the same cloud (expressed in step 4 of Figure 3). This VM is contextualised to know the partially optimised image and the test script. Upon startup, the optimiser VM, analyses the remaining contents of the partially optimised image and selects parts of the image to be removed (see step 5 in Figure 3). These selected parts are expected not to play a direct role in the microservice's functionality – instead they are believed to be used by background activities of the original image (e.g., startup procedures and periodic activities orthogonal to the intended functionality). The selection technique applied here is out of scope of this paper (as this paper is intended to be user - methodology oriented). To analyse the selection the optimiser VM uploads a new image to the cloud (now without the selected contents) and tests the image by instantiating it and evaluating its VE via the user provided shell script (shown in step 6 of Figure 3). If the evaluation is successful, the newly uploaded image is going to become the new partially optimised image and the old image will be discarded. Otherwise, a new image part selection procedure starts. This phase completes if the user defined cost limits are achieved for the optimisation operation or when there are no further selectable parts to be removed from the partially optimised image.

The final version of the partially optimised image is then fetched by the ENTICE Environment as the one containing the intended functionality for the microservice (this is represented in step 7 in Figure 3). If the user intends to alter the service interface (e.g., because in the original image there was a much wider service interface available that needs to be refocused for the minimised feature set now available in the optimised image), then the optimisation phase could be re-run with the altered interfaces more rapidly because the optimisation phase learns its past selection errors and aims at minimising them in later optimisation operations issued by the user.

V. Conclusion

Virtual machine/container images can be created using provider specific templates stored in proprietary repositories, which may lead to provider lock-in and decreased portability. In this paper we addressed image repository management of multiple federated clouds in the frame of the ENTICE project. We provided a methodology for microservice creation by an image synthesis approach to create optimized images in a distributed repository.

As the microservices created through the proposed methodology are derived from a common ancestor (a previously available monolithic service), in our future works, we envision the further optimisation of microservice delivery by building on the common parts of the various microservices created. Using these common parts, we expect that the project will research custom virtual machine management templates that encapsulate the common parts and allow their rapid extension towards any of the previously defined microservice images.

Also, we intend to investigate further how the monolithic service's fragmentation into microservices could be generalised. With this generalisation we plan to support such monolithic services that would not be possible to decompose without introducing alternative protocols in the communication between the fragmented microservices.

ACKNOWLEDGMENT

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

REFERENCES

[1] ENTICE project website. Online: http://www.entice-project.eu/, accessed at March 2016.

[2] SAS – rBuilder. Online: http://www.sas.com/en_us/software/sas9.html, accessed at March 2016.

[3] M. Belguidoum and F. Dagnat. Dependency management in software component deployment. Electr. Notes Theor. Comput. Sci., 182:1732, 2007.

[4] D. Geer. The OS Faces a Brave New World. Computer, 42:1517, October 2009.

[5] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum. Virtual appliances for deploying and maintaining software. In Proceedings of the 17th USENIX Conference on System Administration, LISA'03, pages 181–94, Berkeley, CA, USA, 2003. USENIX Association.

[6] Puppet. Online: http://puppetlabs.com, accessed at March 2016.

[7] Chef. Online: http://www.getchef.com, accessed at March 2016.

[8] Docker. Online: https://www.docker.io, accessed at March 2016.

[9] Image Factory. Online: http://imgfac.org/, accessed at March 2016.

[10] Packer. Online: https://www.packer.io/, accessed at March 2016.

[11] D. Namiot, M. Sneps-Sneppe, On Micro-services Architecture. In International Journal of Open Information Technologies, vol. 2, no. 9, 2014.

[12] Giovanni Toffetti, Sandro Brunner, Martin Blchlinger, Florian Dudouet, and Andrew Edmonds, An architecture for self-managing microservices. In Proceedings of the 1st International Workshop on Automated Incident Management in Cloud (AIMC '15), Victor Ion Munteanu and Teodor-Florin Forti (Eds.). ACM, New York, NY, USA, 19-24, 2015.

[13] Tihfon, G.M., Kim, J. and Kim, K.J., A New Virtualized Environment for Application Deployment Based on Docker and AWS. In Information Science and Applications (ICISA), pp. 1339–1349. Springer Singapore, 2016.

[14] M. Gabbrielli, S. Giallorenzo, C. Guidi, J. Mauro, F. Montesi, Self-Reconfiguring Microservices. In Theory and Practice of Formal Methods, Volume 9660 of the series Lecture Notes in Computer Science pp 194-210, March 2016.