Towards Cloud Application Description Templates Supporting Quality of Service

Gabriele Pierantoni, Tamas Kiss, Gabor Terstyanszky
University of Westminster
London, UK
pierang@westminster.ac.uk, {t.kiss, g.z.terstyanszky}@westminster.ac.uk

***Abstract*—Typical scientific, industrial and public sector applications require resource scalability and efficient resource utilization in order to serve a variable number of customers. Cloud computing provides an ideal solution to support such applications. However, the dynamic and intelligent utilization of cloud infrastructure resources from the perspective of cloud applications is not trivial. Although there have been several efforts to support the intelligent and coordinated deployment, and to a smaller extent also the run-time orchestration of cloud applications, no comprehensive solution has emerged until now that successfully leverages large scale near operational levels and ease of use. COLA is a European research project to provide a reference implementation of a generic and pluggable framework that supports the optimal and secure deployment and run-time orchestration of cloud applications. Such applications can then be embedded into workflows or science gateway frameworks to support complex application scenarios from user-friendly interfaces. A specific aspect of the cloud orchestration framework developed by COLA is the ability to describe complex application architectures incorporating several services. Besides the description of service components, the framework will also support the definition of various Quality of Service (QoS) parameters related to performance, economic viability and security. This paper concentrates on this latter aspect analysing how such application description templates can be developed based on existing standards and technologies.**

Keywords — Cloud Orchestration, Cloud Application Topologies, TOSCA, COLA

I. INTRODUCTION

Cloud Computing has successfully and steadily addressed issues of increasing complexity of IT management[1][2] and control of its costs for the last decade. However, at each step, new challenges must be solved. Nowadays, the use of Infrastructure as a Service (IaaS) layers to externalize the management and decrease the relative costs of the infrastructure is quite common but its adoption still raises difficulties of both technical and economic nature. Although it could offer significant savings, the move to Cloud IaaS has been somehow slower and more cautious in certain fields due to limited application-level flexibility and security concerns.

Typical applications from the scientific, industrial and public sector require resource scalability and efficient resource utilization in order to serve a variable number of customers. However, the dynamic and intelligent utilization of cloud infrastructure resources from the perspective of cloud applications is not trivial. Although there have been several efforts to support the intelligent and coordinated deployment, and to a smaller extent also the run-time orchestration of cloud applications, no comprehensive solution has emerged until now that could be applied in large scale near operational level industry trials.

The migration to IaaS platform is also slowed down by the intrinsic complexity required to describe the correlated services that compose an application, the QoS that describe its execution and the procedures to deploy, undeploy and migrate applications in different IaaS platforms. When faced with such complicated solutions, users may decide to procrastinate or refuse to use IaaS solutions if they are not properly supported. This problem bears some conceptual similarities to those faced in large scale systems, included Science Gateways and Workflow systems in optimally exposing information to its different users at runtime[4][5]. However the current problem focuses on the description of "what is an application made of" (e.g. the graph of its services and how to execute them given a set of QoS parameters) rather "what is an application doing" as investigated in the Science Gateway and Workflow problem.

A new European funded research project, Cloud Orchestration at the Level of Application (COLA) [3] aims at addressing these difficulties to foster the adoption of cloud computing services. The COLA project will provide a reference implementation of a generic and pluggable framework that supports the optimal and secure deployment and run-time orchestration of cloud applications. Such applications can then be embedded into workflows or science gateway frameworks to support complex application scenarios from user-friendly interfaces. A specific aspect of the cloud orchestration framework developed by COLA is the ability to describe complex application architectures incorporating several services. Besides the description of service components, the framework will also support the definition of various Quality of Service (QoS) parameters related to performance, economic viability and security.

This work was funded by the COLA Cloud Orchestration at the level of Applications Project No. 731574 project.

In order to assess the validity of its solutions, COLA will test the applicability of the developed infrastructure in demonstrators and twenty further proof of concept case studies from four distinct application areas that include SMEs and the public sector. COLA use cases incorporate social media data analytics for local governments, simulation-based evacuation planning, data-intensive web applications, and simulation solutions for manufacturing and engineering [10].

This paper focuses on the proposed application description template concept of the COLA project. The rest of the paper is structured as follows: Section II details the objectives of the project, Section III describes the overall architecture of the COLA, Section IV describes the state of the art in the languages used to describe applications, Section V describes the criteria under which TOSCA was selected, Section VI, outlines how the TOSCA language will be used and extended in COLA, finally, Section VII covers conclusions and future work.

II. Objectives of the COLA project

On the resource provision side, Infrastructure as a Service (IaaS) clouds can scale up or down on demand; however, the dynamic and intelligent utilisation of such scalability from the perspective of cloud applications is not trivial. Many legacy applications have been migrated to cloud infrastructures that only consume and run on a predefined static set of resources. More cloud-aware applications have also been developed that offer dynamic scalability based on the demands of user numbers or application characteristics. However, these applications have typically been custom developed requiring significant time and low level cloud computing expertise to implement. On the other hand, typical application patterns can be relatively easily identified that support a large number of similar applications and can use rather similar underlying mechanisms from dynamic clouds.

The overall objective of the COLA project is to define a generic pluggable framework, called MiCADO (Microservices-based Cloud Application-level Dynamic Orchestrator) [4] that supports optimal and secure deployment and run-time orchestration of cloud applications. The MiCADO framework will be able connect to multiple cloud middleware (e.g. EC2, CloudSigma, OpenStack, OpenNebula, etc.) or generic cloud access layers (e.g. CloudBroker Platform) via well-defined standardised interfaces to avoid dependence on one particular cloud technology. MiCADO will also be expressed with a set of interfaces that support the integration of MiCADO enabled applications to workflow systems and science gateway frameworks to provide more convenient access for end-users. COLA will build on existing low level cloud container technologies (e.g. Docker[5], Swarm[6], Consul[7]), management and orchestration solutions (e.g. Chef, Puppet, Occopus[8]), and existing standards, such as, TOSCA [9]. COLA will provide the missing link between existing non cloud aware applications and the dynamic capabilities of IaaS Clouds in the form of a generic framework where multiple technology implementations can be plugged-in and applied on demand.

III. COLA Architecture

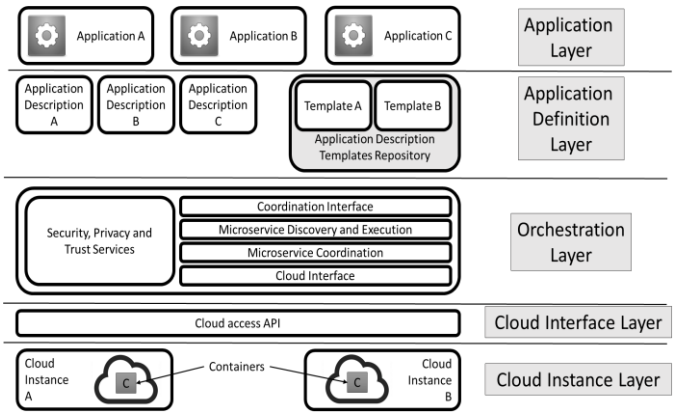

Figure 1, The COLA Project Architecture

In order to address the above objectives, COLA defined a generic architecture to cover all layers of application level orchestration. The overall framework (see Figure 1), called MiCADO, is generic in the sense that it does not dictate the actual implementation of its components. The identified layers and their desired functionality can be implemented in various ways using different technologies and services that in many occasions already exist. The layers of the generic MiCADO architecture (from top to bottom), are as follows:

Application Layer contains actual application code and data to make an incarnation of an application definition. For example, this layer could populate database with initial data, and configure HTTP server with look and feel and application logic. This layer of the architecture will be represented by the COLA SME and public sector demonstrators that will be implemented using the developed MiCADO tools.

Application definition layer is where software components and their requirements (both infrastructure and security specifications) as well as their interconnectivity are defined using application descriptions. As the infrastructure is agnostic to the actual application using it, the application template can be shared with any application that requires such an environment. At this level the COLA project investigates and develops generic and widely applicable application templates that can be reused by application developers in multiple use-case scenarios, significantly speeding up the development process. The guidelines driving the design of the Application Description Template are to support re-usability, to foster the exchanging and sharing of descriptions among similar applications and, finally, to allow different profile of users to focus on the facet they are most interested in allowing the incremental definition of the entire application.

Orchestration layer is divided into four sub-layers. **Coordination interface API** provides access to orchestration

control and decouples the orchestration layer from the application definition. **Microservices discovery and execution layer** manages the execution of microservices and keeps track of services running. **Microservices coordination logic layer** gathers and analyses information on the current performances of the cloud execution environment. **Cloud interface API** offers an abstraction layer to cloud access.

Cloud interface layer provides means to launch and shut down cloud instances. Finally, **Cloud infrastructure layer** contains cloud instances as provided by IaaS cloud providers.

The rest of this paper concentrates on the Application definition layer of MiCADO.

IV. Application Description

A fundamental aspect of COLA and its success is to allow users to define applications in a simple, reusable and flexible fashion and, at the same time, to allow the definition of Quality of Service (QoS) terms with which the applications must be executed. The application description language should also be capable of supporting the description of security and other policies. Additionally, the applications description language should be supported by tools to be easily written and understood, to be shared and queried in repositories and marketplaces, and to be either directly understood by cloud orchestrator components or be translated easily into their own native language.

To support efficient orchestration of application execution on the Cloud COLA elaborates the concept of application template to support application description at three levels: architecture, service, and implementation. The architecture level manages architectures that can be used by different applications in business, industry and public sector. Application templates describe these architectures specifying their service types, relations, and requirements. The service types are high-level services, for example business logic, presentation logic, data service, etc. The service level identifies particular types of services specified in the application template, for example MongoDB, SQL or other database as data service. Service descriptions must be added to the application template to create a service template. The implementation level specifies the service version needed to run the service, for example MongoDB v3.1, v3.2 etc., and the required service signature. Adding this information to the service template developers create an implementation template. Each application template may have multiple service templates, and each of them may have multiple implementation templates, as shown in Figure 2.

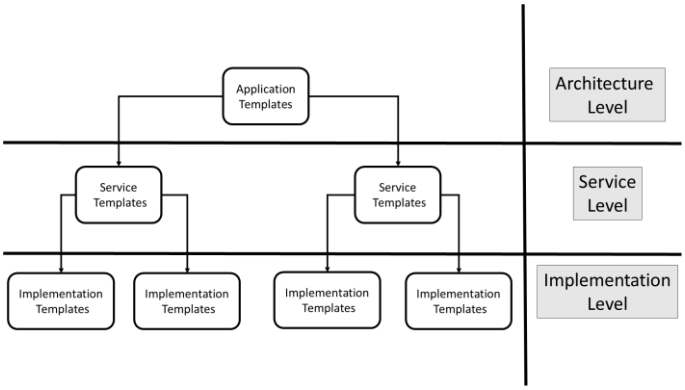

Figure 2, Application, Service and Implementation Templates

The problem of application description is fundamental in Cloud Computing and a large body of work has been produced both in the Academia and the Industry. One of the first tasks of the COLA project has been to assess various solutions and the following, either technology dependent or independent options have been considered.

A. Amazon Machine Image Template

Amazon uses Amazon Machine Image (AMI) template [11] to describe all information required to launch an Amazon EC2 instance. An AMI template includes: root volume for the instance, launch permissions that control which AWS accounts can use the AMI, and block device mapping that specifies the volumes to attach to the instance when it is launched. There are three AMI templates: base AMI template that contains only the OS image, foundational AMI template that includes elements of a stack that change infrequently, and, full stack AMI template that contains all elements of the stack.

Amazon CloudFormation[12] supports development, deployment and running of applications on the Amazon cloud. The applications are described by the AWS CloudFormation templates that combine AMI templates. The templates are stored as text files that comply with the JavaScript Object Notation (JSON) or YAML[13]. The templates can be created and edited in any text editor and can be managed in the source code IDE.

B. Azure Resource Manager Templates

Microsoft Azure describes resources through Azure Resource Manager (ARM) [14]. ARM combines compute, storage and network resources and shows them as a single unit that can be created, managed and deleted together. ARM templates contain four entities: parameters, variables, resources to be deployed, and outputs to be produced. There are four template scopes. Capacity scope delivers a set of resources in a standard topology that is pre-configured to be in compliance with regulations and policies. Capability scope is focused on deploying and configuring a topology for a given technology. End-to-end solution scope is targeted beyond a single capability, and instead focused on delivering an end to end

solution comprised of multiple capabilities. Finally, solution scope manifests itself as a set of one or more capability scoped templates with solution specific resources, logic, and desired state.

C. Oracle Virtual Machine Templates

ORACLE enables quick configuration and provisioning of multi-tier application topologies onto virtualized and cloud environments by capturing the configuration and packaging of existing software components as self-contained building blocks known as appliances. These appliances can then be easily connected to form application blueprints, called as assemblies [15]. They are built on Oracle VM Templates that allow deploying a fully configured software stack by offering pre-installed and pre-configured software images. The Template contains the virtual machine configuration information, and virtual disks that contain the operating system and any application software. These components are packaged together as an Oracle VM Template file according to the industry-standard Open Virtualization Format (OVF). ORACLE offers developers the Oracle Virtual Assembly Builder (OVAB) [16] to support customisation and provisioning of complex enterprise applications with no manual intervention onto virtualized and cloud environments.

D. Chef recipes and cookbooks

Chef [17] [18] is open source cloud orchestration tool that supports integration with cloud-based platforms. It launches and maintains servers, and manages clients that run on nodes, which can be physical or virtual machines. This client performs the automation tasks the specific node requires. The nodes register at a server, which then provides recipes defining these automation tasks and assigns roles. Cookbooks are used to organize related recipes, which are basically Ruby scripts, and supporting resources. Roles contain lists of recipes, which are then executed by the Chef client upon retrieval from the server leading to the desired configuration.

Chef uses a pure-Ruby, domain-specific language (DSL) to describe system configuration and it explicitly describes how to deploy and connect cloud application components.

E. Heat Orschestration Templates

Heat [19] [20] is a pattern-based orchestration mechanism developed by OpenStack. It provides a template-based orchestration for describing a cloud application by executing appropriate OpenStack API calls that generate running cloud applications. The software integrates other core components of OpenStack into a one-file template system. The templates allow for the creation of most OpenStack resource types as well as more advanced functionality such as instance high availability, instance auto-scaling, and nested stacks. These templates, called Heat Orchestration Templates (HOT), are native to Heat and are expressed in YAML. These templates consist of: resources (mandatory fields) which are the OpenStack objects that must be created, like server, volume, object storage, and network resources. Each resource consists of *references* that are used to create nested stacks, *properties* that describe input values for the resource, *attributes* that describe output values for the resource, *parameters* (optional) that denote the properties of the resources, and *output* (optional) that denotes the output created after running the Heat template, such as the IP address of the server.

F. Juju Charms

Juju [21] [22] is an open source automatic service orchestration management tool that enables deploying, managing, and scaling software and services on a wide variety of cloud services and servers. It can significantly reduce efforts needed for deploying and configuring a product's services. Juju utilizes charms to simplify deployment and management tasks. A charm is a set of scripts that can be written in any language. After a service is deployed, Juju can define relationships between services and expose some services to the outside world. Charms encapsulate application configurations, define how services are deployed, how they connect to other services, and how they are scaled. Charms define how services integrate, and how their service units react to events in the distributed environment, as orchestrated by Juju. Charms usually include all the intelligence needed to scale a service horizontally by adding machines to the cluster, preserving relationships with all of the services that depend on that service. This enables developers to build and scale up and down the service on the cloud.

G. TOSCA

An OASIS technical committee [23], containing industrial partners, service providers and research organizations has developed the TOSCA (Topology and Orchestration Specification for Cloud Applications) Language Specification [9], [24], [25] [26] as an interface interoperability standard [12]. Its main goal is to enable the creation of portable cloud applications and the automation of their deployment and management. In order to achieve this goal, TOSCA focuses on three goals: **Automated Application Deployment** and management: is achieved by requiring developers to define an abstract topology of a complex application and to create plans describing its deployment and management. **Portability of Application Descriptions** and their management (but not the actual portability of the applications themselves): TOSCA provides a standardized way to describe the topology of multi-component applications and it addresses management portability by relying on the portability of workflow languages used to describe deployment and management plans. Finally, **Interoperability and reusability of components**: TOSCA aims at describing the components of complex cloud applications in an interoperable and reusable way [27] [28] [29]. The TOSCA language specification is now based on YAML [30] [31] and it allows the description of topologies, nodes and relationships at three different levels of abstractions: Types: akin to an Abstract class in Object Oriented Languages, Templates: akin to a Concrete class in Object Oriented Languages and, finally, Instances: akin to an instance of a class in Object Oriented Languages.

The combination of these three levels of abstraction supports re-usability of descriptions and offers a flexible and expressive syntax for the definition of application templates at different level of granularity. Such an approach, also supports implementation where profiles can be automatically completed to ease the burden of complete specifications [26].

V. SELECTION OF APPLICATION DESCRIPTION APPROACH

The decision on which approach would be ideal to meet COLA's requirements was based on a comprehensive and multi-dimensional analysis to weight strengths and weaknesses of each solution. Such an analysis showed that TOSCA would be an ideal candidate for COLA for the following reasons:

Basic Properties that cover the support for generic functionalities such as portability, scalability and possible implementation characteristics and constraints such as the packaging of installation artefacts and the availability of examples and tutorials. TOSCA offers support for generic functionalities (portability, scalability, etc.) suitable for the project, it offers the packaging of installation artefacts and supports a large variety of installation methodologies that vary from simple scripts to complex workflows. Furthermore and quite importantly, TOSCA is an accepted standard, it is supported by a strong and growing community, and there is ample literature of several and successful attempts of its usage by the research community.

Entities and Storage that cover the capacity of the various solutions to describe the individual components and the whole application, their relationships and their overall design. This category also covers the support to publish, store query and share application description profiles. TOSCA's philosophy is very similar and highly compatible with COLA's three layered concept to describe applications, their components, their relationships and generic templates. TOSCA also supports the possibility to publish, discover and share application description templates.

QoS parameters that covers the possibility (if not explicitly the capacity) of expressing Quality of Service parameters such as elasticity, scalability, and security. TOSCA does not directly or explicitly support QoS parameters but it is flexible and generic enough to allow them to be described as policies that will be later interpreted to other components of the MICADO architecture.

Application execution that covers the support for the execution of the applications. TOSCA per se is a language specification so it does not directly provide runtime or container support but there are implementations that do so (as an example OPENTosca [32], [33] and [34]).

As a further argument to select TOSCA is its **interoperability**. Even today many cloud orchestration tools are able to manage TOSCA based application/service descriptions and their number is increasing every year. Selecting a TOSCA based approach to specify applications/services will improve the possibility to share description of COLA applications.

VI. EXTENDING TOSCA TO MEET COLA REQUIREMENTS

TOSCA is a highly flexible language specification and it is already successfully employed in various cloud-based initiatives. Its adoption in COLA is promising for various reasons.

First, COLA's overall architecture (see Figure 1) is divided into applications, services and implementations that can be intuitively mapped into the TOSCA specifications of templates, nodes and relationships (see Figure 2) as detailed in Figure 3. Each of the COLA layers can be defined as a TOSCA Topology Template (a graph of Nodes and Relationship Templates) enriched by level-specific policies. At the lowest level, implementation plans are detailed either as scripts and/or workflows. This approach allows for reusability of application descriptions as these descriptions (and parts thereof) that are common (or have significant overlap) of different applications can be shared and individually finalized. This process can also be replicated across the different layers by deriving and incrementally defining the service topologies at each level.

At the same time, TOSCA can be extended with the definitions of policies (e.g. to define scale-up and scale-down profiles at run time, redundancy approaches, security and trust) at the Application and Service layers that will be used to define the implementation parameters at the lowest levels.

The two latter characteristics combined allow to implement information hiding strategies that diversify the contribution of each user profile to what is necessary to be decided at each level, leaving all other information to be inferred by templates at the upper level of automatic definition driven by policies.

From a programmatic point of view, TOSCA also offers two points of great interest to COLA: the possibility to define the implementation plan both as scripts and as workflows thus offering solutions at the required level of complexity, and, the support to two different execution modes: imperative and declarative (one where the exact implementation steps are specified, the other where the result is defined and the implementation can be chosen among a certain range of solutions).

Finally, TOSCA is being actively used both in the academic and industrial worlds and therefore offers not only a vast experience from which the COLA participants can greatly benefit but also a variety of implementations that can be directly used to provide inspiration to COLA. Among these, the most promising are: OPENTosca, an open source implementation of TOSCA and TOSCAMart [27], a place to share Application Descriptions, Winery [35], a modelling tool to create TOSCA profiles, and Vinotek [36], a portal for the provision of Cloud applications on demand. Using a combination of these tools will allow to share Applications Templates and also to edit them through dedicated GUIs in the same fashion as an IDE.

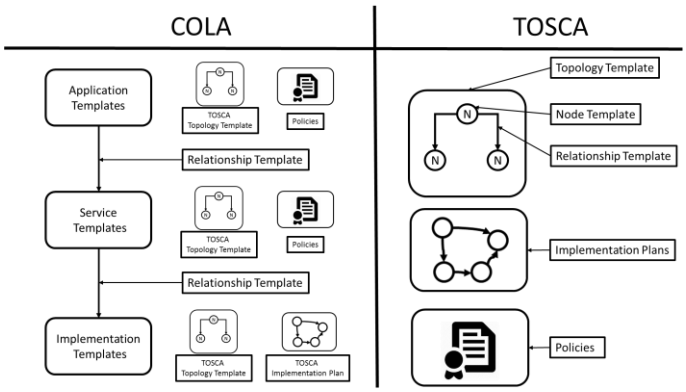

Figure 3, Extending TOSCA in the COLA project

VII. Conclusion and Future Work

COLA is in its infancy at the moment. The project started in January 2017, and a few preliminary results have already been achieved. A survey of application description languages has clearly highlighted TOSCA as the best approach for COLA and, on the implementation side, first prototypes of MiCADO have shown the possibility to rapidly scale up and down deployment of a test application (data-avenue [37], a data transfer suite). In the current prototype, the scaling policies have not been described in TOSCA but rather with the local native language of the Cloud orchestrator tool, Occopus, applied. The definition of such policies in TOSCA in such a way to be understandable by MiCADO is one of the next steps of the project.

References

[1] D. R. Avresky, M. Diaz, A. Bode, B. Ciciani, and E. Dekel, Eds., *Cloud Computing*, vol. 34. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[2] F. Leymann, *Cloud Computing*. 2011.

[3] "About – COLA Project – Cloud Orchestration at the Level of Application." [Online]. Available: http://www.project-cola.eu/cola-project/. [Accessed: 27-Mar-2017].

[4] H. Visti, T. Kiss, G. Terstyanszky, G. Gesmier, and S. Winter, "MiCADO – Towards a microservice-based cloud application-level dynamic orchestrator," Jan. 2016.

[5] "Docker - Build, Ship, and Run Any App, Anywhere." [Online]. Available: https://www.docker.com/. [Accessed: 30-Mar-2017].

[6] "Docker Swarm overview - Docker Documentation." [Online]. Available: https://docs.docker.com/swarm/overview/. [Accessed: 30-Mar-2017].

[7] "Consul Architecture - Consul by HashiCorp." [Online]. Available: https://www.consul.io/docs/internals/architecture.html. [Accessed: 30-Mar-2017].

[8] "Welcome - Occopus." [Online]. Available: http://occopus.lpds.sztaki.hu/en_GB/. [Accessed: 29-Mar-2017].

[9] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "TOSCA: Portable Automated Deployment and Management of Cloud Applications," in *Advanced Web Services*, 2014, pp. 527–549.

[10] S. J. E. Taylor, T. Kiss, G. Terstyanszky, P. Kacsuk, and N. Fantini, "Cloud computing for simulation in manufacturing and engineering: introducing the CloudSME simulation platform," *Proceedings of the 2014 Annual Simulation Symposium*. Society for Computer Simulation International, p. 12, 2014.

[11] "Learn Template Basics - AWS CloudFormation." [Online]. Available: http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/gettingstarted.templatebasics.html. [Accessed: 20-Feb-2017].

[12] "AWS CloudFormation - Infrastructure as Code & AWS Resource Provisioning." [Online]. Available: https://aws.amazon.com/cloudformation/. [Accessed: 30-Mar-2017].

[13] "The Official YAML Web Site." .

[14] "Microsoft Azure Essentials Azure Web Apps for Developers | Microsoft Press Store." .

[15] Kai Yu, "Design and Implement a SelfService Enabled Private Cloud with Oracle Enterprise Manager 12c." .

[16] "Oracle Fusion Middleware."

[17] "Chef - Automate IT Infrastructure | Chef." [Online]. Available: https://www.chef.io/chef/. [Accessed: 29-Mar-2017].

[18] M. Pfeiffer, "Chef Server on the AWS Cloud: Quick Start Reference Deployment," 2015.

[19] R. Mateescu, "OpenStack Heat – Overview."

[20] "Heat - OpenStack." [Online]. Available: https://wiki.openstack.org/wiki/Heat. [Accessed: 29-Mar-2017].

[21] "Juju | Cloud | Ubuntu." [Online]. Available: https://www.ubuntu.com/cloud/juju. [Accessed: 29-Mar-2017].

[22] J. Baker and U. S. Team, "Service Orchestration for Cloud Environments with Juju," 2012.

[23] OASIS, "OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC," *Website*, 2014. [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca. [Accessed: 15-Feb-2017].

[24] "tosca-primer-v1.0."

[25] A. Brogi, J. Soldani, and P. Wang, "TOSCA in a nutshell: Promises and Perspectives."

[26] P. Hirmer, U. Breitenbücher, T. Binz, and F. Leymann, "Automatic Topology Completion of TOSCA-based Cloud Applications."

[27] J. Soldani, T. Binz, U. Breitenbücher, F. Leymann, and A. Brogi, "ToscaMart: A method for adapting and reusing cloud applications," *J. Syst. Softw.*, vol. 113, pp. 395–406, 2016.

[28] J. Soldani, T. Binz, U. Breitenbücher, F. Leymann, and A. Brogi, "TOSCA-MART: A Method for Adapting and Reusing Cloud Applications TOSCA-MART: A Method for Adapting and Reusing Cloud Applications *," 2015.

[29] A. Brogi and J. Soldani, "Reusing cloud-based services with TOSCA *."

[30] "TOSCA Simple Profile in YAML Version 1.0." [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csd03/TOSCA-Simple-Profile-YAML-v1.0-csd03.html. [Accessed: 15-Feb-2017].

[31] W. Draft, "TOSCA Simple Profile in YAML Version 1.0," 2014. [Online]. Available: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/csprd01/TOSCA-Simple-Profile-YAML-v1.0-csprd01.html. [Accessed: 14-Feb-2017].

[32] "OpenTOSCA Ecosystem." .

[33] "OpenTOSCA Container – Architecture." [Online]. Available: http://www.iaas.uni-stuttgart.de/OpenTOSCA/container_architecture.php. [Accessed: 20-Feb-2017].

[34] T. Binz *et al.*, "OpenTOSCA – A Runtime for TOSCA-based Cloud Applications."

[35] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann, and U. Breitenb, "Winery – A Modeling Tool for TOSCA-based Cloud Applications."

[36] U. Breitenbücher, T. Binz, O. Kopp, and F. Leymann, "Vinothek – A Self-Service Portal for TOSCA."

[37] "Welcome - Data Avenue." [Online]. Available: https://data-avenue.eu/. [Accessed: 03-Apr-2017].