Flexible Deployment of Social Media Analysis Tools

Flexible, Policy-Oriented and Multi-Cloud deployment of Social Media Analysis Tools in the COLA Project

Gabriele Pierantoni, Tamas Kiss, Gregoire Gesmier, James DesLauriers, Gabor Terstyanszky
Department of Computer Science,
University of Westminster
London, United Kingdom
pierang@westminster.ac.uk

José Manuel Martín Rapún
R&D Department
INYCOM
Zaragoza, Spain
josemanuel.martin@inycom.es

***Abstract*— The relationship between companies and customers and among public authorities and citizens has changed dramatically with the widespread utilisation of the Internet and Social Networks. To help governments to keep abreast of these changes, Inycom has developed Eccobuzz and Magician, a set of web applications for Social Media data mining. The unpredictable load of these applications requires flexible user-defined policies and automated scalability during deployment and execution time. Even more importantly, privacy norms require that data is restricted to certain physical locations. This paper explains how such applications are described with Application Description Templates (ADTs). ADTs define complex topology descriptions and various deployment, scalability and security policies, and how these templates are used by a submitter that translates this generic information into executable format for submission to the reference framework of the COLA European project**

Keywords—COLA, TOSCA, Cloud Orchestration, Swarm

I. Introduction

The relationship of companies with their customers and of public authorities with their citizens has changed dramatically with the widespread utilisation of the Internet and Social Networks. Monitoring this information has become a critical aspect for private companies, but it is still scarcely used in the public sector. To overcome this disparity, the Aragon Regional Government in Spain has set the goal to develop communication channels with citizens to become aware of their opinion about the local government's services, and how these can be improved. The authorities also want to offer information to entrepreneurs and companies in the region that can be used to improve businesses or develop new ones.

The local government already collects large amounts of data resulting from interactions with its citizens (e.g. applying for public services such as grants, aids, subsidies, and licenses). This data can be combined and extended with information publicly available in social networks. The aim is to set up a business gateway that is supported by the intelligent analysis and utilisation of all available information.

To fulfil this target, Inycom [1], an ICT company headquartered in Aragon, developed Eccobuzz, a web application for Social Media data mining, competitive intelligence and brand and media management. It is offered as a Software as Service (SaaS) product hosted in Inycom's Data Centre. Some private customers, particularly larger organizations, use a different distribution of Eccobuzz with extended functionalities, called Magician. This distribution can be personalized and deployed in customers' premises. An interesting feature provided by Magician is that it not only collects and structures raw data from the Internet, but it also gathers information about the user posting this data, and his/her sentiment about it. Therefore, Magician is an ideal candidate for the Aragon Regional Government to collect and analyse data regarding its citizens, and their opinion and attitude towards local services.

When Social Media Analysis Tools are used correctly, they can offer precious insight that can be used to improve services, on the other hand, the very nature of these tools and the data they rely upon, pose difficult challenges in the protection of the privacy of the data of the users. These challenges are further complicated when databases and software processes may run on different Clouds requiring access to a heterogeneous set of resources in a dynamic way. Two further aspects of these Social Media Analysis Tools are the ever-increasing size of the amount of data available and, at the same time, the unpredictable fluctuation of computing load required due to the high level of uncertainty on how much information will be collected by the crawlers to be processed later.

Eccobuzz and Magician search the Internet for information and process it semantically producing valuable structured information made available through web interface, excel export and pdf reports. The main components and information flow in this system are displayed in Figure 1. This paper does not intend to describe in great detail the implementation of Eccobuzz and Magician, but rather to describe how, in order to meet support the above challenges, these tools are currently being prototyped for the submission to the reference framework developed by the COLA (Cloud Orchestration at the Level of Application) European project [2].

As part of this work, the applications' architectures are described in a TOSCA-based (Topology and Orchestration Specification for Cloud Applications) [3] description format, together with Policies that describe desired Quality of Service

(QoS) parameters related to performance, scalability, economic viability and other security and privacy-related aspects.

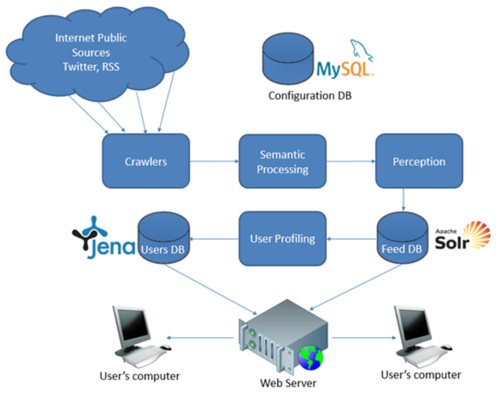


Figure 1, main component of the Magician Social Tool

These policies are the means through which COLA describes the requirements and constraints that are to be met during the application lifecycle in the cloud in order to meet the unique challenges of the Social Media Analysis Tools. This standards-based description is then translated to be used by components of the COLA reference framework in order to deploy the application topology and enforce various scalability and security policies. As these applications are the very first large-scale case studies to be deployed, the experiences learned during this process are constantly fed back and influence the development of the COLA reference framework.

This paper presents our experiences when developing application description templates including both topology and policy descriptions for Eccobuzz and Magician and demonstrates how such templates can be translated and utilized by container-based cloud technologies for their deployment. The rest of this paper is structured as follows: Section 2 introduces the COLA project and its objectives, Section 3 describes the concept of Application Description Templates (ADT) used to describe applications and their policies in COLA, Section 4 presents the architecture of the Application Submitter, Section 5 and Section 6 provide details of the current prototype. Section 7 describes related work, comparing ADTs to similar solutions offered by other cloud service providers including Microsoft ARM templates and Amazon CloudFormation templates. Finally, Section 8 concludes the paper with a summary and future work.

II. The COLA Project

SMEs and public-sector organizations increasingly investigate the possibilities of using cloud computing services in their everyday business. Accessing services and resources in the cloud on-demand and in a flexible and elastic way could result in significant cost savings due to more efficient and convenient resource utilization that also replaces large investment costs with long term operational costs. On the other hand, the take up of cloud computing by SMEs and the public sector is still relatively low due to limited application-level flexibility and security concerns as suggested in the Work Programme for Information and Communication Technology[1]

A European funded research project, Cloud Orchestration at the Level of Application (COLA) aims at addressing these difficulties to foster the adoption of cloud computing services. COLA is based on a conceptual architecture that describes several generic components which offer fundamental functionalities needed to support 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.

COLA does not dictate implementation details of its components and defines a reference implementation whereby the various components can be substituted in a pluggable fashion. However, COLA defines a few fundamental functionalities that have to be offered by its pluggable components: the management of virtual machines, the management of containers within these virtual machines and the enforcement of policies which describe various Quality of Service (QoS) parameters related to deployment, geographical location, performance, economic viability and security.

To describe the application architecture and the policies that govern their lifecycle, the COLA project developed the concept of Application Description Templates (ADT) which are based on the TOSCA Language Specification [4][5]. ADTs offer a description of applications that is as technology-agnostic as possible. As TOSCA is a generic specification of a meta-language, the COLA project has proposed a specific TOSCA compliant description format for the Application Description Templates.

To support the development and testing of such description language, COLA prototypes three industry case-studies and develops near production level demonstrators to showcase its results. These three large-scale application examples, besides the already described Eccobuzz and Magician, include the scalable deployment of complex evacuation simulation scenarios, and the analysis of data arising from ticket sales of various cultural institutions. While these three case-studies directly influence the early development of the COLA reference framework and the related ADTs, in the second phase of the project, twenty further use-cases will be prototyped as proof of concepts on the developed solution.

III. The Application Description Template (ADT)

ADTs act as information conduits that connect Application Developers to the various components of the COLA reference framework. Although the COLA reference framework defines

[1] http://ec.europa.eu/research/participants/data/ref/h2020/wp/2016_2017/main/h2020-wp1617-leit-ict_en.pdf

various components, the most relevant for the design of ADTs and for the scope of this paper are those represented in Figure 2.

The information stored in the ADT is parsed and dispatched to the relevant components. This information includes policies that define behavior of the applications (including security features), and information that is required for the deployment of virtual machines and containers.

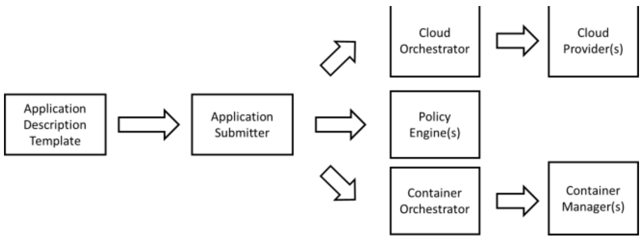


Figure 2, the application submitter and the other components of the COLA reference framework

The Application Submitter is responsible for the division of this information as follows: Information that describes the deployment of containers goes to a container orchestrator that is connected to container managers. Information specific to the selection and execution of virtual machines is passed on to a cloud orchestrator component which is connected to one or more cloud provider. Finally, information specific to policies, is dispatched to one or more policy engines that in turn enforce the various policies by connecting to the other components of the COLA reference framework.

To combine the flexibility offered by technology-oriented agnosticism of COLA with the expressiveness required to describe all the various facets of a large variety of applications, we have designed the ADT to describe two main aspects of applications: its topology and its policies. The topology describes the main components of the application whilst the policies describe the modalities that govern the various parts of the application lifecycle.

A. *The ADT General Structure*

Each ADT consists of one topology template that comprehends the components described in Figure 3. **The Input Section** groups together fields that Application Developers are likely to override their default value. **The Policies Section** defines the policies that are applied to the application. Each policy can be applied to one or more nodes at different stages of their lifecycle. **The Container Section** defines a set of nodes that specify the Container Images that contain the various components of the applications. **Finally, the Virtual Images Section** defines the characteristics of the Virtual Machines that will host the Container Images defined in the section above.

B. *Application Topology*

The Topology of the Application is contained in the Container and Virtual Images layers. The first deals with the deployment of the application components into containerized services, and the second describes how such containers are to be deployed in virtual machines. The various containers that compose the image are connected with edges that implement the TOSCA *ConnectsTo* relations. The relationship that connects a container to the virtual machine it should deployed in is described with the *HostedOn* relationship specified by TOSCA.

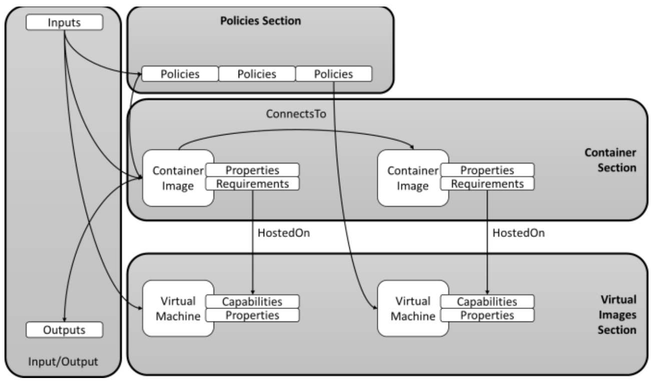


Figure 3, the COLA ADT Structure

C. *Application Policies*

The description and enforcement of policies in TOSCA is an additional dimension to the description of the application topologies. Policies define and enforce the modalities that regulate the application lifecycle.

TOSCA allows the definition of type hierarchies of arbitrary complexity. COLA defines a three-layered hierarchy of policies that all derive from the TOSCA Root Policy. To facilitate the application of policies at the various levels, each policy defines the nodes it has to be applied to. For each node, the overall policy is composed of sub-policies that describe the various features such as security, scalability, etc. Each policy is in turn divided into two main sections. The Description Section comprises meta-data which define the name, type and description of the policy, as well as a target (defined elsewhere in the topology) to which the policy should apply. The Properties Section contains data that falls under two kinds of parameters: common to all COLA policies types, and specific to each policy type.

Common Properties are Stage (which defines at which stage of the lifecycle of the element the policy is applied), and Priority (which is an arbitrary integer ranging from 0 to 100 used to define the priority with which the policy will be implemented). Specific Properties are specific to each Policy. These parameters vary depending on the nature of the policy itself. As an example: a scalability policy based on CPU consumption will define various parameters that specify scalability thresholds, a deployment policy will define minimum number of CPUs, minimum memory size for deployment, etc…

IV. The Application Submitter

The Application Submitter, a prototype of which is currently under development and testing at the University of Westminster, implements the submission of the ADT. The Application Submitter performs two main actions upon an ADT: it separates its various components (Container Orchestration, Virtual

Machine, and Policies) and then invokes adaptors which translate the information into a format understood by their respective components. To support the technology-oriented agnosticism of the ADT, the Application Submitter adopts the design described in Figure 4.

The Application Submitter relies on the OpenStack TOSCA parser[7] which is used to read a TOSCA file, check its syntactical correctness and create an in-memory dictionary which is then passed to the Mapper component. The Mapper uses a key list to isolate the information subset that is then passed along to the various adaptors that format them into the structure expected by the corresponding components of the COLA reference framework.

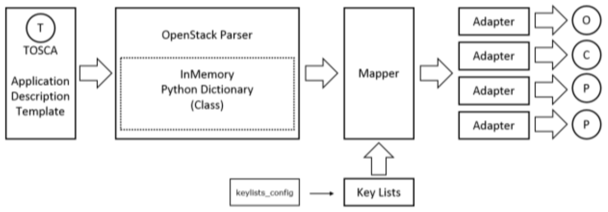


Figure 4, Design of the Application Submitter

V. The Description of Magician with the ADT

We have used an ADT to describe the deployment of the Motor Engine of Magician and its configuration database (based on MongoDB[8]) to the COLA reference framework. For this first prototype, it was decided to deploy the main components of Magician within a single container and to express four main aspects of the policies that govern its behaviour: scalability, resources and security connections. The two policies that deal with scalability and Location Deployment are particularly relevant to the deployment and execution of Social Media Analysis Tools as they address the main concerns of unexpected load fluctuations and they allow to maintain the data sets within a geographical area ruled by certain legal constraints and requirements (e.g. the European General Data Protection Regulation[2]). The policies are recapitulated in Table 1 and described following the structure introduced in Section III. C.

Policy Number	Policy Type	Policy Description
P1.1	Consumption Based Scalability	Defines minimum and maximum CPU consumptions thresholds above which a new container instance is deployed and below which a container instance is un-deployed
P1.2	Connection Deployment Policy	Defines the set of inbound connections that must be allowed.
P1.3	Resource Deployment Policy	Defines the minimum requirements of the Virtual Machine in terms of CPU, Memory Size and Disk size.
P1.4	Location Deployment Policy	Defines the physical location of the computation and storage resources.

Table 1, Policies Describing the deployment and execution of Magician

As an example of the ADT policy description, we describe in Table 2 the parameters that define the Application Scalability Policy which govern how containers are scaled up and down depending on the CPU consumption of the Magician Container.

Name	Value	Description	Type
Stage	Execution	The part of the application lifecycle to which the policy applies	TOSCA Node State
Priority	100	The priority with which the policy is enforced	Integer
Namespace	Prometheus	Defines the namespace of the service used for monitoring the CPU consumption.	String
Max	80	Defines the maximum CPU consumption (in percentage) above which a new instance is deployed.	Integer
Min	20	Defines the minimum CPU consumption (in percentage) under which the instance is un-deployed.	Integer
Time	600	Defines (in seconds) the amount of time above or below the Max/Min threshold before the action is triggered	Integer

Table 2, Example of Policy Implementation Parameters

VI. Prototype Implementation

To demonstrate the feasibility of the above-described concepts, a first version on the Application Submitter has been implemented that translates the ADT into a format understood by a Container Orchestrator. The current prototype of the COLA reference framework uses Docker Swarm[9] as Container Orchestrator and we have configured the Application Submitter to connect with it through an adapter that transforms the container-level application topology into a compose file[10]. The Policy Keeper is currently under development, so Policies are parsed by the Application Submitter but no adaptor is available as of now. For testing purposes, the same policies are hard-coded on the different components.

The ADT is first parsed and validated by the OpenStack parser. The mapper then separates out the container-relevant sections of TOSCA and passes them to a translator. This container-level portion of TOSCA (Figure 7) is translated into the format of a Docker-Compose file (Figure 8) so that it may be understood by the currently implemented container orchestrator, Docker Swarm. As TOSCA and Docker Compose both appropriate YAML as a language, translation from the former to the latter is straightforward. An adaptor for this specific container orchestrator considers three basic sets of information within the subset of data that is provided by the Mapper.

The first are the TOSCA properties defined within the description of containers. These properties should align with the runtime arguments that can be passed to Docker via the *docker run* command or via a Docker Compose file, and should follow the naming conventions of the Docker Compose format. One example of this is seen in Figure 7, where the property for ports is defined, but other options such as command, *entrypoint*, and environment should also be defined at this level. Translation of

[2] https://www.eugdpr.org/

these properties simply involves copying them across under a new key in the Compose file.

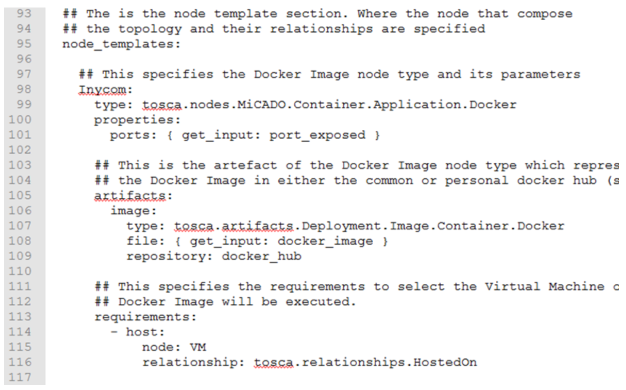
The second set of information is contained within TOSCA artefacts, which define external data that must be retrieved during orchestration. The image from which to build the container is described as an artefact with properties that define the image name, as well as the repository where it can be found.

The final set of information to translate comes in the form of TOSCA relationships. Relationships match TOSCA requirements with matching capabilities to describe how the various components within the ADT should interact with each other. TOSCA standards define the following: a *HostedOn* relationship between a container and a virtual machine, a *ConnectsTo* relationship between two containers, and an *AttachesTo* relationship for connecting a container to a block storage volume.

Translating an *AttachesTo* relationship requires defining a new volume and providing an appropriate reference to that volume, both inside the Compose file. Translating a *ConnectsTo* relationship involves defining a new network, and referencing that network under both of the connecting components, again inside the Compose file. Translation of the HostedOn relationship needs to define a constraint inside the Compose file, and then requires further cooperation from the cloud orchestrator to ensure an appropriate reference is made for that constraint.

Following the TOSCA standards, as well as the specific outline of the ADT as implemented by the COLA reference framework is important in ensuring an adaptor can understand and translate the information into an understandable format.

In order to finally launch the application, the resultant Docker-Compose file is passed to Swarm and executed by the *docker deploy* command on an instance of MiCADO V3[2] which is the current implementation choice for the COLA reference framework. In this implementation of MiCADO, cloud orchestration and default policies are hard-coded, making it ideal to test container-level orchestration in isolation.

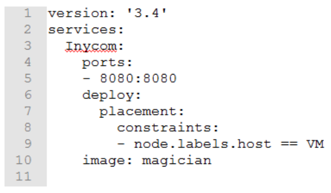


```
## The is the node template section. Where the node that compose
## the topology and their relationships are specified
node_templates:

  ## This specifies the Docker Image node type and its parameters
  Inycom:
    type: tosca.nodes.MiCADO.Container.Application.Docker
    properties:
      ports: { get_input: port_exposed }

    ## This is the artefact of the Docker Image node type which repres
    ## the Docker Image in either the common or personal docker hub (s
    artifacts:
      image:
        type: tosca.artifacts.Deployment.Image.Container.Docker
        file: { get_input: docker_image }
        repository: docker_hub

    ## This specifies the requirements to select the Virtual Machine c
    ## Docker Image will be executed.
    requirements:
      - host:
          node: VM
          relationship: tosca.relationships.HostedOn
```

Figure 5, ADT Container-Level Topology Description in TOSCA

VII. Related Work

Research described in this paper is closely linked to similar attempts at application description both in industry and academia. The OASIS [11] efforts to define standards for the application description have created TOSCA [12][13][14], a widely adopted standard both with several implementations [15] and related tools [16][17][18].



```
version: '3.4'
services:
  Inycom:
    ports:
    - 8080:8080
    deploy:
      placement:
        constraints:
        - node.labels.host == VM
    image: magician
```

Figure 6, Translation of the ADT Container-Level Topology Description into a Swarm COMPOSE file

Amazon uses Amazon Machine Images (AMI) [19] to describe all information required to launch Amazon EC2 instances. AMIs are templates which include a description of the root volume (i.e. an operating system, an application server, and applications), 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. The AMI template must contain at least the base operating system and it may be customised to include additional configuration and software code. AMIs are not written templates, but rather read-only, re-usable snapshots of EC2 instances. The COLA project currently implements Docker container orchestration to provide similar functionality to that which is offered by an AMI. Docker serves as a lightweight, more technology agnostic solution which sees wider use. Through the modularity of the COLA framework, any other container orchestrator, or even a virtual machine image solution could be substituted in place of Docker.

Amazon CloudFormation[20] supports development, deployment and running applications on the Amazon cloud. AWS CloudFormation templates describe the different instances (using AMIs) and resources that make up an application stack, as well as security rules and other customisations that should apply to the deployment. The templates are stored as text files that comply with JavaScript Object Notation (JSON) or YAML [21]. They can be created and edited in any text editor and can be managed in the source code IDE. CloudFormation templates are analogous to the ADTs used by COLA and can even describe container based deployments, though they follow an internal language specification instead of a global solution such as that offered by the TOSCA standard.

Microsoft Azure[22] describes resources through Azure Resource Manager (ARM) which 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 that can be entered during run-time with a set of values or default values pre-defined, variables that are static in the code and used for deploying resources, resources to be deployed, and outputs to be produced. ARM templates are written and stored as text files in the JSON format. The functionality offered by ARM templates is no different from that offered by AWS CloudFormation templates, or COLA's own ADTs. Again, these templates follow their own structure and wording instead of using the specification laid out by OASIS in TOSCA.

COLA approaches the description of an application stack in the cloud with more modularity and technology agnosticism than do Amazon CloudFormation templates or Microsoft ARM templates. By offering a modular approach to assembling and deploying images which make up the applications, COLA allows for and encourages reuse of those images across platforms.

Similarly, as the TOSCA standard becomes more widely accepted and appropriated, COLA ADT templates will become more familiar and could even be reused across platforms. Cloudify [23] and Alien4Cloud [24] are two open-source cloud orchestrators currently under development. While they do not support the same modularity of components offered by COLA, they do conform to the TOSCA standard. Comparing other aspects of these open-source solutions with COLA are outside the scope of this paper, but their adoption of TOSCA is encouraging and hopefully suggests an upward trend in adherence to TOSCA and the growth of its community.

VIII. Conclusion and Future Work

This first proof of concept prototype has demonstrated the viability of the technology-agnostic approach and how a TOSCA-based ADT can be used to describe abstract application topologies and delegate to a component (Application Submitter) the translation to formats that are technology-specific. This first prototype will now be extended to implement further functionalities such as flexible submission lifecycle management and support for more detailed policies through a Policy Keeper currently under development in COLA. The Application Submitter will be embedded into the MiCADO architecture and will connect the ADTs with the application-level orchestration features of MiCADO.

Acknowledgments

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

References

[1] "Inycom | Tecnoloía e Innovación para tu Negocio." [Online]. Available: https://www.inycom.es/. [Accessed: 18-Mar-2018].

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

[3] "TOSCA_overview."

[4] "TOSCA-spec-v1.0."

[5] 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.

[6] 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].

[7] "TOSCA-Parser - OpenStack." [Online]. Available: https://wiki.openstack.org/wiki/TOSCA-Parser. [Accessed: 29-Oct-2017].

[8] "MongoDB for GIANT Ideas | MongoDB." [Online]. Available: https://www.mongodb.com/. [Accessed: 30-Oct-2017].

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

[10] "Compose file version 3 reference | Docker Documentation." [Online]. Available: https://docs.docker.com/compose/compose-file/. [Accessed: 21-Mar-2018].

[11] "OASIS | Advancing open standards for the information society." [Online]. Available: https://www.oasis-open.org/. [Accessed: 18-Mar-2018].

[12] "Topology and Orchestration Specification for Cloud Applications," 2013.

[13] G. Katsaros, M. Menzel, A. Lenk, R. Skipp, and J. Eberhardt, "Cloud Service Orchestration with TOSCA, Chef and Openstack Jannis Rake-Revelant."

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

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

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

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

[18] 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.

[19] S. Pearce and S. Bryen, "Managing Your AWS Infrastructure at Scale," 2015.

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

[21] "The Official YAML Web Site." [Online]. Available: http://www.yaml.org/. [Accessed: 20-Feb-2017].

[22] Rick Rainey, *Microsoft Azure Essentials Azure Web Apps for Developers | Microsoft Press Store*. .

[23] "Cloudify - Cloud Orchestration" [Online]. Available: https://cloudify.co/product/. [Accessed: 22-Apr-2017].

[24] "Alien4Cloud" [Online]. Available: https://alien4cloud.github.io/index.html. [Accessed: 22-Apr-2017].