Science of Computer Programming 166 (2018) 194–213

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Orchestrating incomplete TOSCA applications with Docker

Antonio Brogi, Davide Neri, Luca Rinaldi, Jacopo Soldani *

Department of Computer Science, University of Pisa, Italy

ARTICLE INFO

Article history:
Received 18 December 2017
Received in revised form 6 July 2018
Accepted 11 July 2018
Available online xxxx

Keywords:
Cloud applications
TOSCA
Docker
Container reuse

ABSTRACT

Cloud applications typically integrate multiple components, each needing a virtualised runtime environment that provides the required software support (e.g., operating system, libraries). This paper shows how TOSCA and Docker can effectively support the orchestration of multi-component applications, even when their runtime specification is incomplete. More precisely, we first introduce a TOSCA-based representation of multi-component applications, and we illustrate how such representation can be exploited to specify only the application-specific components. We then present TosKeriser, a tool for automatically completing TOSCA application specifications, which can automatically discover the Docker-based runtime environments that provide the software support needed by the application components. We also show how we fruitfully exploited TosKeriser in two concrete case studies. Finally, we discuss how the specifications completed by TosKeriser can be automatically orchestrated by already existing TOSCA engines.

1. Introduction

Cloud computing permits running on-demand distributed applications at a fraction of the cost which was necessary just a few years ago [2]. This has revolutionised the way applications are built in the IT industry, where monoliths are giving way to distributed, component-based architectures. Modern cloud applications typically consist of multiple interacting components, which (compared to monoliths) permit better capitalising the benefits of cloud computing [11].

At the same time, the need for orchestrating the management of multi-component applications across heterogeneous cloud platforms has emerged [4,17]. The deployment, configuration, enactment and termination of the components forming an application must be suitably orchestrated. This must be done by considering all the dependencies occurring among the components forming an application, as well as the fact that each application component must run in a virtualised environment providing the software support it needs [13].

Developers and operators are currently required to manually select and configure an appropriate runtime environment for each application component, and to explicitly describe how to orchestrate such components on top of the selected environments [19]. As we discuss in Sect. 2, such process must then be manually repeated whenever a developer wishes to modify the virtual environment actually used to run an application component, e.g., because the latter has been updated and it now needs additional software support.

The current support for developing cloud applications should be enhanced. In particular, developers should be required to describe only the components forming an application, the dependencies occurring among such components, and the software support needed by each component [3]. Such description should be fed to tools capable of automatically selecting

* Corresponding author.
E-mail address: soldani@di.unipi.it (J. Soldani).

https://doi.org/10.1016/j.scico.2018.07.005
0167-6423/

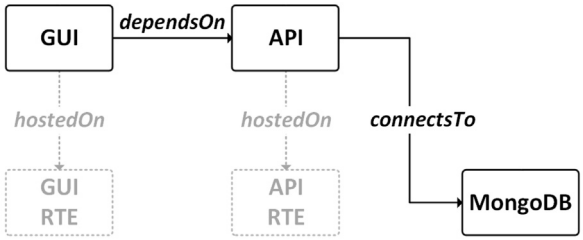

Fig. 1. Running example: The application *Thinking*.

and configuring an appropriate runtime environment for each application component, and of automatically orchestrating the application management on top of the selected runtime environments. Such tools should also allow developers to change the virtual environment running an application component whenever they wish (e.g., by automatically replacing a previously selected environment with another satisfying the new/updated requirements of an application component).

In this paper, we present a solution geared towards providing such an enhanced support. Our solution is based on TOSCA [22], the OASIS standard for orchestrating cloud applications, and on Docker, the de-facto standard for cloud container virtualisation [24]. The main contributions of this paper are indeed the following:

- We propose a TOSCA-based representation for multi-component applications, which can be used to specify the components forming an application, the dependencies occurring among them, and the software support that each component requires to effectively run.
- We present TosKeriser, a tool that automatically completes TOSCA application specifications, by discovering and including Docker-based runtime environments providing the software support needed by the application components. The tool also permits changing — when/if needed — the runtime environment used to host a component.

The obtained application specifications can then be processed by orchestration engines supporting TOSCA and Docker (such as TosKer [7], for instance). Such engines will automatically orchestrate the deployment and management of the corresponding applications on top of the given runtime environments.

This paper extends [5] by (a) extending the approach of [5] to permit hosting groups of software components on the same Docker container, by (b) providing a detailed description of the implementation of TosKeriser, and by (c) presenting two novel case studies comparing the orchestration of the management of applications with and without our solution (based on three KPIs) and illustrating the usefulness of groups.

The rest of the paper is organised as follows. Sect. 2 illustrates an example further motivating the need for an enhanced support for orchestrating the management of cloud applications. Sect. 3 provides some background on TOSCA and Docker. Sect. 4 shows how to specify application-specific components only, with TOSCA. Sect. 5 then presents our tool to automatically determine appropriate Docker-based environments for hosting the components of an application. Sect. 6 illustrates the two case studies, while Sects. 7 and 8 discuss related work and draw some concluding remarks, respectively.

2. Motivating scenario

Consider the open-source web-based application *Thinking*,[1] which allows its users to share their thoughts, so that all other users can read them. *Thinking* is composed by three interconnected components (Fig. 1), namely (i) a *MongoDB* storing the collection of thoughts shared by end-users, (ii) a Java-based REST *API* to remotely access the database of shared thoughts, and (iii) a web-based *GUI* visualising all shared thoughts and allowing to insert new thoughts into the database. As indicated in the documentation of the *Thinking* application:

(i) The *MongoDB* component can be obtained by directly instantiating a standalone Docker-based service, such as `mongo`,[2] for instance.
(ii) The *API* component must be hosted on a virtualised environment supporting maven (version 3), java (version 1.8) and git (any version). The *API* must also be connected to the *MongoDB*.
(iii) The *GUI* component must be hosted on a virtualised environment supporting nodejs (version 6), npm (version 3) and git (any version). The *GUI* also depends on the availability of the *API* to properly work (as it sends GET/POST requests to the *API* to retrieve/add shared thoughts).

Docker containers work as virtualised environments for running application components [24]. However, we currently have to manually look for the Docker containers offering the software support needed by *API* and *GUI* (or to manually

[1] https://github.com/di-unipi-socc/thinking.
[2] https://hub.docker.com/_/mongo/.