SmartDBO: Smart Docker Benchmarking Orchestrator for Web-application

Devki Nandan Jha
Newcastle University
Newcastle Upon Tyne, UK
d.n.jha2@newcastle.ac.uk

Michael Nee
Newcastle University
Newcastle Upon Tyne, UK
info@michael-nee.co.uk

Zhenyu Wen
Newcastle University
Newcastle Upon Tyne, UK
zhenyu.wen@newcastle.ac.uk

Albert Zomaya
Sydney University
Sydney, Australia
albert.zomaya@sydney.edu.au

Rajiv Ranjan
Newcastle University
Newcastle Upon Tyne, UK
raj.ranjan@newcastle.ac.uk

ABSTRACT

Containerized web-applications have gained popularity recently due to the advantages provided by the containers including light-weight, packaged, fast start up and shut down and easy scalability. As there are more than 267 cloud providers, finding a flexible deployment option for containerized web-applications is very difficult as each cloud offers numerous deployment infrastructure. Benchmarking is one of the eminent options to evaluate the provisioned resources before product-level deployment. However, benchmarking the massive infrastructure resources provisioned by various cloud providers is a time consuming, tedious and costly process and is not practical to accomplish manually.

In this demonstration, we present **Smart D**ocker **B**enchmarking **O**rchestrator (**SmartDBO**), a general orchestration framework that automatically benchmarks (deploys and executes) users' containerized web-applications across different cloud providers while meeting the constraints of budget and deployment configurations. SmartDBO aims to answer two questions: (i) how to automate the benchmarking of containerized web-application across multi-cloud environments?, (ii) how to maximize the diversity in a benchmarking solution which covers maximum numbers of cloud providers and types of provisioned infrastructures without exceeding users' budgets? We evaluate all the features of SmartDBO using SimplCommerce and TPC-W executing across Amazon AWS and Microsoft Azure.

CCS CONCEPTS

• **Information systems → Web applications**; • **Computer systems organization → Cloud computing**; • **Software and its engineering → Cloud computing**; *Organizing principles for web applications.*

KEYWORDS

Web-application; Docker Container; Benchmark; Orchestrator; Cloud

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3314137

ACM Reference Format:
Devki Nandan Jha, Michael Nee, Zhenyu Wen, Albert Zomaya, and Rajiv Ranjan. 2019. SmartDBO: Smart Docker Benchmarking Orchestrator for Web-application. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3308558.3314137

1 INTRODUCTION

Web-application have become an integral part of our life providing various functions including social networking, banking, e-commerce, etc. With the growing demand, these applications are becoming complex and follow multi-tier architecture, which makes it hard to develop, debug, deploy and upgrade [4]. However, microservices architecture brings a new approach to overcome the difficulty of developing this large software by breaking it into a suite of small, independent and versioned modular services [2, 9]. Evolution of container technology with different features including self-contained, light weight, fast start up and shut down, easy scalability, etc. provides great support for developing microserviced web-applications [8]. It allows each modular service of the complex web-application to be deployed as an independent containerized web-application. The containerized web-applications can interact with each other through Restful APIs and be easily deployed, upgraded and scaled.

Additionally, cloud has become the main infrastructure for the deployment of containerized web-applications. There are about 267 cloud providers which provide various computing resources for the containerized application using 3 main *Deployment Infrastructures (DI)*: Bare Metal (BM), Virtual Machine (VM) and Container Service (CS) [5]. Each type of DI has the variety of hosts either predefined by cloud providers or user customized. For example, Amazon EC2 provides different types of pre-customized hosts (VMs) for their customers along with the self customized hosts. In the rest of the paper, we use host to represent an instance of the DI. The containerized web-application can be deployed and executed on any type of host.

Various cloud providers provision heterogeneous hosts that may significantly affect the performance of web-applications, which has a strong impact on user satisfaction [3]. Thus, we need to benchmark the hosts before deploying the real applications [6]. Most of the research in benchmarking is focused on developing a better benchmark application in terms of reality of emulation, efficiency and scalability [1, 10]. The challenge of deploying a benchmark

for containerized applications across multiple clouds has not been well explored. If a user manually performs the deployment, it will be tedious, error-prone and requires a lot of time and expertise. Due to the fact that different cloud environments have different accessing schemes and different ways to interact, an orchestrator is desired that can automatically run the benchmark applications across multiple clouds, and collect the required system parameters while providing the flexibility for users to customize their experiment settings.

Previous benchmark frameworks from both academia [1, 7, 10] and industry[1, 2] mainly focus on VM based applications, which is not applicable for containerized web-applications. Also, the variety of cloud providers offer a massive configuration choices of host. For instance, Amazon EC2 provides 43 types of hosts for their customers[3] excluding self customized hosts. Thus, this poses another challenge for the proposed orchestrator, i.e., how to provide an optimized recommendation to help users in selecting the hosts from the provided clouds?

The aforementioned challenges discussed above can be resolved by answering the following research questions:

- *How to automatically orchestrate the benchmark applications across multi-cloud environments based on the user's specification?*
- *How to find an optimal set of hosts for benchmarking in a fixed budget that covers the diversity both within a cloud provider and among the cloud providers?*

To answer these questions, we developed SmartDBO that makes the following contributions:

- We developed a noble orchestrator, SmartDBO that automates the definition and execution of benchmarks for containerized web-applications. In particular, the orchestrator allows the user to choose the benchmark applications and hosts of DI across different cloud providers.
- SmartDBO also has a native feature of optimization that can maximize the utility of a user's budget by maximizing the number of cloud providers and the hosts of the DI.

To demonstrate the effectiveness of SmartDBO, we chose AWS and Azure as the cloud providers. TPC-W[4], a traditional web-application benchmark is used for testing and the workload for testing the benchmark application is generated by the Apache JMeter[5]. Moreover, our SmartDBO captures the live performance of the examined experiments and stores the results in the Database for further analysis and reporting.

2 SYSTEM OVERVIEW

This section discusses the architecture and the implementation details of SmartDBO.

[1]https://www.cloudorado.com/cloud_server_comparison.jsp
[2]https://cloudharmony.com/status
[3]https://aws.amazon.com/ec2/instance-types/
[4]https://cs.nyu.edu/ totok/professional/software/tpcw/tpcw.html
[5]https://jmeter.apache.org/

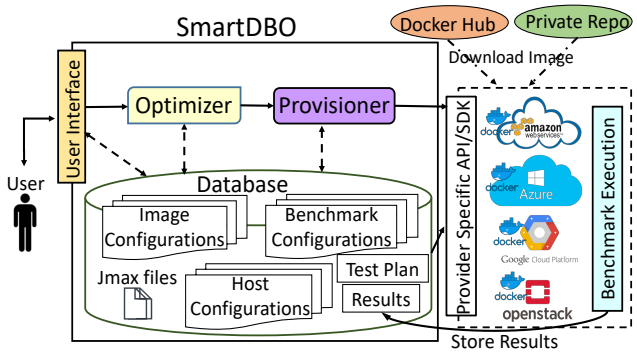

Figure 1: System Architecture of SmartDBO

2.1 SmartDBO Architecture

Figure 1 illustrates the architecture of SmartDBO and the dependencies of each component. SmartDBO is implemented as a web-application that provides a *User Interface* for users to interact, explore and manage their benchmarking experiments. The *User Interface* allows the user to choose an existing benchmark application or customize a new application. Moreover, users can select the available hosts from different cloud providers, define the benchmarking time for each selected host, and specify the total budget for running the experiments via the *User Interface*. Next, this configuration information is stored in a relational Database.

The *Optimizer* is designed to create an optimized host list based on the information provided by the user. It retrieves the necessary information (host configurations, time of benchmark and budget) from the Database and applies a heuristic algorithm to generate an optimized host list for running the benchmarking experiments. More details about the *Optimizer* are given in § 2.2. The generated host list is automatically stored in the Database. If the user chooses to execute the benchmark experiments, the *Provisioner* will be triggered to provision the resources, deploy the benchmark applications and execute the applications based on the user entered information and optimized host list. The benchmark is executed for the specified interval of time and the completion is notified to the *Provisioner*. The results are stored in the Database in real-time for further evaluation and analysis. Finally, the user is notified after completion of the experiment while releasing the cloud resources.

2.2 SmartDBO Implementation

The SmartDBO is mainly implemented by using *Microsoft ASP.NET Core 2.1 C#*[6] programming language. The *User Interface* is built using *CSS*, *JavaScript* and *HTML5* to ensure a good user experience. We aim to provide a unified interface for users to deploy their benchmark applications over multiple clouds, therefore hiding the difficulties of interacting with heterogeneous cloud APIs. Moreover, the SmartDBO also provides an optimized solution for users based on their budgets. These two main features are implemented in the modules of *Optimizer* and *Provisioner* and will be detailed as follows.

[6]https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.1

Optimizer. SmartDBO benchmarks containerized web-application in a multi-cloud environment. Let N represent the cloud providers $C_i | i \in \{1, N\}$ where each provider C_i has T type of hosts $v_{i,t} | t \in \{1, T\}$. In our evaluation, we assume a one-to-one mapping between host and container. Consider $C(v_{i,t})$ to be the unit cost of using $v_{i,t}$, $\tau_{i,t}$ is the time units for which $v_{i,t}$ is chosen to run and $\mathcal{B}$ is the user budget for the benchmark, finding an optimal set of hosts for the benchmarking is modeled as a Binary Integer Linear Programming problem (BILP). The defined objective function is shown below,

$$\textbf{maximize:} \sum_{\mathbf{i,t}} \mathbf{x_{i,t}} + \lambda\{\sum_{\mathbf{i}}\{\sum_{\mathbf{t}} \mathbf{x_{i,t}} - \mathbf{T}\}\}$$

where $x_{i,t} | x_{i,t} \in \{0, 1\}$ is a binary variable which represents whether $v_{i,t}$ is selected or not. The first factor of the optimization problem is to comprehend maximum selection of hosts and the second considers a penalizing factor to boost the spanning of the maximum number of cloud providers. λ is a tunable parameter which is incorporated to maintain a balance. There are certain constraints associated with the objective function as given below.

$$\sum_{i,t} C(v_{i,t}) \times \tau_{i,t} \leq \mathcal{B} \tag{1}$$

$$\forall i \forall t \; x_{i,t} \in \{0, 1\}, x_{i,t} = \begin{cases} 1, & \text{when VM, } V_{i,t} \text{ is selected} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$\forall i \forall t \; \tau_{i,t} \geq 0 \tag{3}$$

$$\forall i \; \sum_{t} x_{i,t} \geq 1, \; \forall t \; \sum_{i} x_{i,t} \geq 1 \tag{4}$$

Constraint (1) states that the total cost of benchmarking different containers running inside the host must be less than the defined budget. Also, the cost is calculated only if a host is selected as specified by the constraint (2). Constraint (3) represents variable stability constraints while constraint (4) enforces the selection of at least one cloud provider and at least one host configuration.

We developed and implemented a heuristic algorithm for *Optimizer* to solve the problem formalized above. The algorithm selects an optimized list of hosts while satisfying all the defined constraints. The source code and the implementation details of *Optimizer* is available as an open-source project on github[7]. The advantage of *Optimizer* for selecting hosts in both multi-cloud and single cloud is illustrated in Figure 2. The results show that the solution generated from the *Optimizer* covers more types of host in both single cloud and multi-cloud scenarios, compared to the case of random host selection with a given budget.

Provisioner. Once the *Optimizer* generates a benchmark plan, the user can decide whether he/she wants to submit the plan for execution via the user friendly web interface. If the user agrees to perform the experiment, the functions implemented in *Provisioner* will be triggered. First, the *Provisioner* will check the connection and the requirement of the resources on different clouds. Next, it uses a background process application, Hangfire[8], to create and launch the hosts on the selected cloud providers.

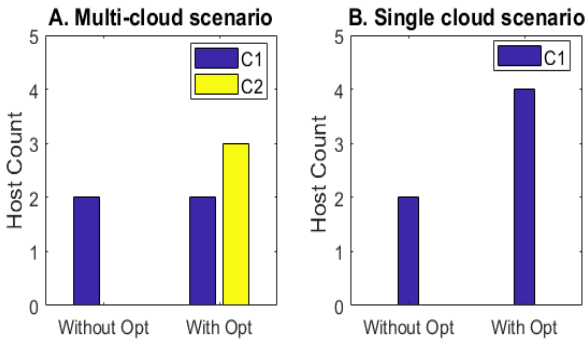

Figure 2: Showing advantage of Optimizer

3 DEMONSTRATION

In this section, we will walk through the execution workflow of SmartDBO by showing the benchmarking of a sample e-commerce web-application, SimplCommerce[9]. The SimplCommerce application is configured with 50 test products to replicate a small real-life e-commerce store.

3.1 Experiment setup

SmartDBO. The proposed orchestrator is implemented in C# language which can be executed in a variety of environments e.g. Linux, Windows, MacOS. In this demonstration, we run our orchestrator on a Lenovo PC with Intel(R) Core(TM) i5-6200U CPU @2.3GHz - 2.4GHz with a memory 16 GB and storage of 512 GB SSD. The system is installed with 64 bit Windows 10 operating system. We execute SmartDBO using Visual Studio 2017[10] IDE. It is open-sourced and the current version of code along with samples and installation details are available on github[7].

Required open-source tools. In this demonstration, we utilized a popular benchmark application called TPC-W[11] to emulate the activities of a sample e-commerce web application. The load on the web-application is created by Apache JMeter[12] according to the test plan defined by the user. To emulate real traffic, JMeter is not configured on the same cloud where the benchmark applications are running. PostgreSQL Database[13] is associated with SmartDBO, and stores different host, benchmark and image configurations for setting up the experiments in real-time.

Cloud providers. We deployed our containerized benchmark application on AWS and Azure with different host configurations. In this demonstration, we use VM as the default host configuration. SmartDBO can provision any types of VMs which are available on the subscribed cloud providers.

[7] https://github.com/smartdockerbenchmarkingorchestrator/Smart-Docker-Benchmarking-Orchestrator

[8] https://www.hangfire.io/

[9] https://www.simplcommerce.com/

[10] https://visualstudio.microsoft.com/vs/

[11] https://cs.nyu.edu/ totok/professional/software/tpcw/tpcw.html

[12] https://jmeter.apache.org/

[13] https://www.postgresql.org/

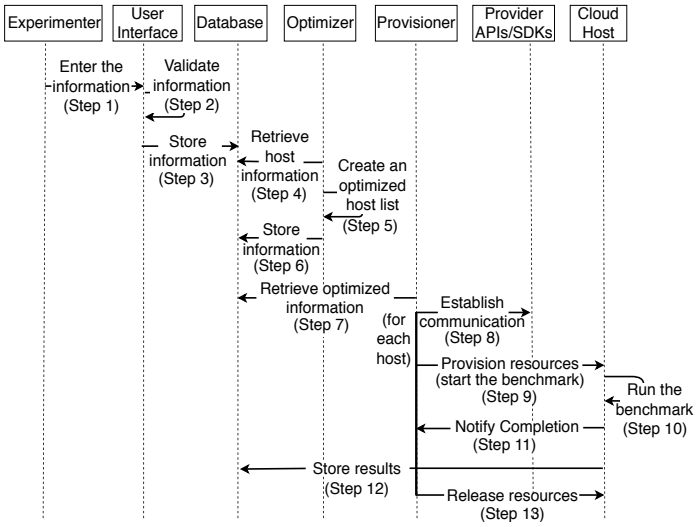

Figure 3: SmartDBO Execution Workflow

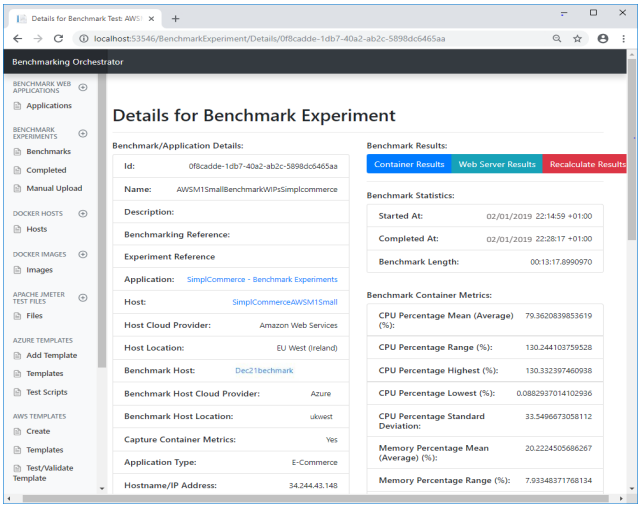

Figure 4: SmartDBO Execution Interface

3.2 The execution workflow

In this subsection, we use an example to demonstrate the process of orchestrating a benchmark application as shown in Figure 3.

Define the benchmark application. At the beginning, SmartDBO is designed for benchmarking different types of web-application. The application is defined in terms of a web-application image and a JMeter workload image using the interactive *User Interface* as depicted in Step 1 of Figure 3. The images are referred from Docker hub or other private repository or created using the provided template. For the demonstration, we create an application using SimplCommerce and a JMeter docker image.

Define the host configuration. When we decide the benchmark application, our SmartDBO allows us to define the host configuration to run the benchmark application. The *User Interface* allows us to choose a name, description, host type (application/benchmark) and cloud-specific (e.g. credentials, template) information as shown in Step 1. It also provides the HTTP and TLS authentication to guarantee secured communication.

Define the test plan. SmartDBO allows us to define the workload using a test plan where we can provide the JMX file for continuous workload generation. New test plans can be easily added by filling the template provided by the framework (Step 1). The plan includes all the parameters for JMeter to generate a continuous load for benchmarking the web-application.

Define the benchmark experiment. Once we have the benchmark application and hosts configured, we can set the whole experiment for performing the benchmarks. Then, we define the benchmark experiment which maps the load generator to the benchmarking host. To this end, we pair a load generator with the host running the benchmark web-application. Along with this, we can specify the execution time and the budget for executing the experiment. The entered data is validated (Step 2) after which it is stored in the Database (Step 3).

Perform optimization. The *Optimizer* retrieves the input host information from the Database (Step 4) to generate an optimized host list for the execution (Step 5). At the same time the optimized host list is stored in the Database (Step 6).

Execute the benchmark. *Provisioner* retrieves the optimized host list from the Database (Step 7), we can then start to execute the experiment. First the communication with the cloud host is established using the provider specific APIs/SDKs (Step 8) and then resources are provisioned for benchmarking (Step 9). The *Provisioner* is notified of the completion (Step 11) and the result is stored in the Database (Step 12) for visualization and further analysis. The experiment result provides different performance metrics including benchmark statistics (e.g. started at, benchmark length), benchmark container metrics (e.g. CPU percentage mean, CPU percentage range, memory percentage mean, network input total, network output total), benchmark web-server metrics (e.g. throughput, average response time, number of requests, number of errors, average latency) and Apdex rating (Apdex count, Apdex rating, Apdex satisfied count,). Finally, the resources are released (Step 13).

Figure 4 shows the overview of a successful execution of TPC-W benchmark using SimplCommerce web-application executing on AWS and Azure, with the obtained result.

Report the result. The benchmark execution results can be easily visualized using the *User Interface*. There are multiple options to report the results in different formats including on-screen, csv file, excel sheet, pdf for further analytics.

The complete demonstration explained here is available as a screencast video on YouTube [14].

ACKNOWLEDGEMENTS

This research was partly supported by Engineering and Physical Sciences Research Council, UK under grant EPSRC - EP/R033293/1.

[14] https://www.youtube.com/watch?v=c44FSCR3C-w&feature=youtu.be

REFERENCES

[1] Emmanuel Cecchet, Veena Udayabhanu, Timothy Wood, and Prashant Shenoy. 2011. Benchlab: an open testbed for realistic benchmarking of web applications. In *Proceedings of the 2nd USENIX conference on Web application development*. USENIX Association, 4–4.

[2] Tomas Cerny, Michael J. Donahoo, and Michal Trnka. 2018. Contextual Understanding of Microservice Architecture: Current and Future Directions. *SIGAPP Appl. Comput. Rev.* 17, 4 (Jan. 2018), 29–45. https://doi.org/10.1145/3183628.3183631

[3] Justin Cheng, Caroline Lo, and Jure Leskovec. 2017. Predicting intent using activity logs: How goal specificity and temporal range affect user behavior. In *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, 593–601.

[4] Mariela Curiel and Ana Pont. 2018. Workload Generators for Web-Based Systems: Characteristics, Current Status, and Challenges. *IEEE Communications Surveys & Tutorials* 20, 2 (2018), 1526–1546.

[5] Maria Fazio, Antonio Celesti, Rajiv Ranjan, Chang Liu, Lydia Chen, and Massimo Villari. 2016. Open issues in scheduling microservices in the cloud. *IEEE Cloud Computing* 3, 5 (2016), 81–88.

[6] Devki Nandan Jha, Saurabh Garg, Prem Prakash Jayaraman, Rajkumar Buyya, Zheng Li, and Rajiv Ranjan. 2018. A Holistic Evaluation of Docker Containers for Interfering Microservices. In *2018 IEEE International Conference on Services Computing (SCC)*. IEEE, 33–40.

[7] Diego Lugones, Jordi Arjona Aroca, Yue Jin, Alessandra Sala, and Volker Hilt. 2017. AidOps: A Data-driven Provisioning of High-availability Services in Cloud. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17)*. ACM, New York, NY, USA, 466–478. https://doi.org/10.1145/3127479.3129250

[8] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.

[9] Zhongshan Ren, Wei Wang, Guoquan Wu, Chushu Gao, Wei Chen, Jun Wei, and Tao Huang. 2018. Migrating Web Applications from Monolithic Structure to Microservices Architecture. In *Proceedings of the Tenth Asia-Pacific Symposium on Internetware (Internetware '18)*. ACM, New York, NY, USA, Article 7, 10 pages. https://doi.org/10.1145/3275219.3275230

[10] Joel Scheuner, Jürgen Cito, Philipp Leitner, and Harald Gall. 2015. Cloud workbench: Benchmarking iaas providers based on infrastructure-as-code. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 239–242.