Integrating Faban with Docker for Performance Benchmarking

[Demonstration Paper]

Vincenzo Ferme
Faculty of Informatics
University of Lugano (USI)
vincenzo.ferme@usi.ch

Cesare Pautasso
Faculty of Informatics
University of Lugano (USI)
cesare.pautasso@usi.ch

ABSTRACT

Reliability and repeatability are key requirements in performance benchmarking ensuring the trustworthiness of the obtained performance results. To apply a benchmark to multiple systems, the reusability of the load driver is essential. While Faban has been designed to ensure the reliability of the performance data obtained from a benchmark experiment, it lacks support for ensuring that the system under test is deployed in a known configuration. This is what Docker, a recently emerging containerization technology, excels at. In this demo paper we present how we integrated Faban with Docker as part of the BenchFlow framework to offer a complete and automated performance benchmarking framework that provides a reliable and reusable environment, ensuring the repeatability of the experiments.

Keywords

Faban, Docker, Performance Benchmarking, Repeatability

1. INTRODUCTION

Benchmarking is a well established practice for discovering application's performance pitfalls and bottlenecks [4]. A key requirement in performance testing is ensuring the reliability of the experiments [3], since important decisions will be taken based on the obtained results. Many tools for reliable performance testing exist. One of them is Faban[1], a free and open source performance workload creation and execution framework, largely used in industry standard benchmarks, such as the ones released by the Standard Performance Evaluation Corporation (SPEC)[2]. The growing complexity of modern distributed applications requires to automate their deployment with tools, such as the ones provided by the Docker ecosystem[3]. Most of the available performance testing tools, do not provide a well defined and standard mechanism to integrate the deployment of the System Under Test (SUT) into the performance testing and benchmarking process. This is key to reduce the cost of automating the benchmarking. Additionally, this integration is necessary to enhance the repeatability [3] of the performed experiments. Additionally, given the growing complexity of the targeted deployment environment, it has become important to provide lightweight and non-invasive means to define custom performance data to be collected on the SUT side.

With BenchFlow[4] [2] we aim at solving the aforementioned limitations, as well as enhancing the reusability of the load drivers. In this demo paper we discuss the functionality and the architecture of the framework, and present a walk-through use case to demonstrate its capabilities.

[1]http://www.faban.org
[2]https://www.spec.org
[3]https://www.docker.com

ICPE'16 March 12-18, 2016, Delft, Netherlands

ACM ISBN 978-1-4503-4080-9/16/03.

DOI: http://dx.doi.org/10.1145/2851553.2858676

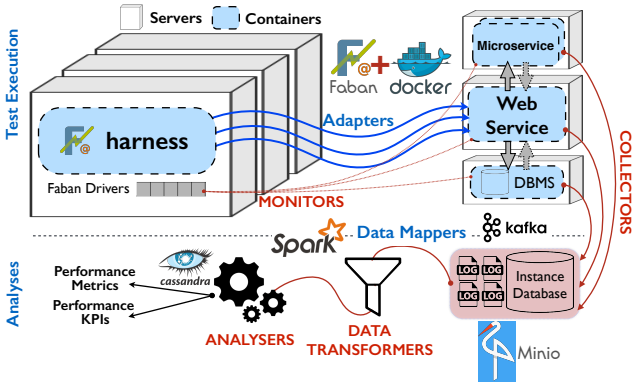

Figure 1: BenchFlow Framework

2. THE BENCHFLOW FRAMEWORK

The BenchFlow framework, presented in Fig.1, is deployed in Docker containers and builds on top of Faban, to provide reliable performance benchmarking, and Docker, to ensure repeatability and reusability. The load drivers, defined by exploiting the Faban framework, are executed by the Harness, and provide the infrastructure needed to define the simulated users and their interaction with the SUT. The BenchFlow framework defines adapters between the load drivers and the SUTs. That way the same load drivers can be reused to load different systems by defining the mapping between the abstract interaction defined in the generic load drivers, and the actual implementation for each SUT. The

[4]http://www.benchflow.inf.usi.ch

adapters guarantee the reusability of the same driver for different types of performance test [4].

The benchmark life cycle starts with SUT's deployment by exploiting the Docker containerization technology. Thus it provides for the replicability of the experiments by ensuring the SUT is always deployed in the exact same initial state. While containerization technologies introduce some overhead on system's performance, a recent reliable performance analysis of Docker [1] indicated that, if carefully configured, Docker reaches near-zero overhead. During the performance test execution, the BenchFlow framework monitors the experiment's execution state to gather resource utilization (e.g., CPU, RAM, Network) data, using the lightweight monitors of the Docker stats API. When the performance test execution is complete, a set of collectors gather the raw performance data from the distributed infrastructure on which the performance test has been performed, and send them to a central storage service, e.g., Amazon S3, or Minio[5]. We rely on a storage service, since the data have to be efficiently accessed from the components computing the metrics. To abstract from the different data formats that different SUTs might have, the performance data are then transformed to a canonical meta-model. After the transformation, the performance data are stored in a Cassandra[6] database (DB), and the performance metrics and KPIs are computed. Cassandra is a powerful DB for storing and accessing performance data from the service that computes metrics on top of them. The computation is performed by relying on Apache Spark[7], a fast, general-purpose engine for large-scale data processing. Before the computation of the metrics is triggered, the BenchFlow framework checks the logs collected from the SUT to identify execution errors and validate the experiment.

The orchestration of the performance test execution, the data collection, and the performance data analysis, is delegated to Apache Kafka[8], a publish-subscribe messaging framework. We have introduced this state-of-the-art framework to decouple the benchmark execution managed by the Faban Harness, from the performance metrics computation, and thus pipeline the gathering of performance data with the corresponding analytics, which can be performed offline.

3. GOALS OF THE DEMONSTRATION

The BenchFlow framework is currently used for benchmarking Workflow Management Systems, and has been successfully applied in different experiments [2]. In the demo we will present a walk-through use case that shows BenchFlow framework's capabilities, both from the perspective of performance researchers and performance testers. During the demo, the framework will be pre-installed on multiple servers, in a dedicated, reliable and controlled environment, which should be accessible over VPN from the conference venue. We will go through the end-to-end performance test process, by defining, submitting and monitoring the performance test, describing the SUT deployment definition, and accessing the automatically calculated performance metrics.

Defining the performance test: the framework simplifies the definition of load drivers' behavior with Faban, through load driver definition descriptors. We will define a performance test, involving a distributed system of multiple microservices and their DBs, by means of a benchmark definition file, where we can define all the performance test parameters (e.g., a load test). The framework will take care of translating the performance test definition to the actual format used internally by Faban. **Describing the SUT deployment definition**: the deployment is performed by relying on the Docker Compose tool. This ensures that each SUT deployment configuration (e.g., different amounts of RAM) and its initial state can be precisely described and executed obtaining the exact same initial conditions for the experiment. Paired with Docker Swarm[9] it is possible to automate SUT's deployment on a distributed infrastructure. **Submitting and monitoring the performance test**: we will submit the performance test we have previously defined, and monitor its execution status. The test will last 1 minute and will be repeated 3 times. **Accessing the automatically calculated performance metrics**: when the benchmark execution is complete and the performance data are ready to be explored, we will visualize them at the end of the demo.

During the demo, we aim to demonstrate the flexibility and the simplicity of using the BenchFlow framework. Once defined by means of the BenchFlow benchmark definition file and the Docker Compose deployment definition file, the experiments can be replicated multiple times in a fully automated way to obtain reliable and verified results.

4. CONCLUSIONS AND FUTURE WORK

The BenchFlow framework greatly simplifies and accelerates the definition and execution of reliable, repeatable and reusable performance tests. It does so by integrating two powerful technologies: Faban and Docker, and building on top of their functionality to provide a complete framework for automated performance test execution over a distributed environment. The next planned steps concern enabling the simplified definition of additional performance test types (e.g., spike testing), as well as providing stronger performance test validation and means for performance test result analysis and exploration. Moreover we also plan to collaborate with the ICPE and the SPEC community, to drive the framework's development and strengthen its functionality.

5. ACKNOWLEDGMENTS

This work is partially funded by the Swiss National Science Foundation with the BenchFlow - A Benchmark for Workflow Management Systems (Grant Nr. 145062) project.

6. REFERENCES

[1] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. Technical report, IBM, July 2014.

[2] V. Ferme, A. Ivanchikj, and C. Pautasso. A framework for benchmarking BPMN 2.0 workflow management systems. Proc. of BPM '15, pages 251–259, 2015.

[3] K. Huppler. The art of building a good benchmark. TPCTC 2009, pages 18–30. Springer, 2009.

[4] I. Molyneaux. *The Art of Application Performance Testing: From Strategy to Tools*. O'Reilly, 2nd edition, 2014.

[5] https://www.minio.io

[6] http://cassandra.apache.org

[7] http://spark.apache.org

[8] http://kafka.apache.org

[9] https://www.docker.com/docker-swarm