INSTITUT FÜR INFORMATIK

Employing Open Source Software for RBEE Regression Benchmarking Execution Environment

Armin Moebius and Wilhelm Hasselbring

Bericht Nr. 1609
December 2016
ISSN 2192-6247

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

ZU KIEL

Abstract

We introduce our Regression Benchmarking Execution Environment (RBEE) approach. Based on self-contained systems (SCS), RBEE is separated into several independent systems which can both, be scaled and replaced independently. RBEE requires only a running container infrastructure (Docker) including a cluster manager (Docker Swarm) for operating the corresponding SCSs. Prebuilt container images are provided for each SCS. For collecting monitoring data, we are employing the Kieker Monitoring Framework. Therefor we extended Kieker to store its monitoring data directly in RBEE's write-optimized storage (Apache Cassandra). Within RBEE itself we are using read-optimized storage (Elasticsearch). Using polyglot persistence as well as polyglot programming, each SCS of the RBEE approach uses data storage and programming languages which best fit to its requirements. In addition, there are transformation processes between read- and write-optimized storage. Communication among the different SCS takes place via a separate communication backend (RabbitMQ, REST) and well-defined communication APIs. Data stored within the SCS is not persistent. Hence, benchmark results are stored within a separate benchmark result repository in a traditional virtual machine detached from the container infrastructure. Each SCS contains an own user interface for system management, but for overall system monitoring and visualization RBEE provides a central control center. RBEE will be published as open source software (rbee.io).

1 Introduction

Our Regression Benchmarking Execution Environment (RBEE) [1] offers benchmarking capabilities for detecting performance regressions of Java-based software built within continuous integration environments. For gathering monitoring data, we employ the Kieker Monitoring Framework [2, 3], which hands over its generated monitoring output to RBEE's Monitoring Log [4]. In the remainder of this paper we introduce the RBEE's architecture and the underlying open source software.

2 Foundations

Our RBEE architecture is built on top of Docker [5], an open source software which automates the deployment of Linux-based applications inside software containers. Within each software container a complete file system including everything required to run is wrapped. For this purpose Docker utilizes an additional abstraction layer and automates operating-system-level virtualization on Linux systems. In more detail, Linux Containers (LXC) allow running multiple isolated Linux instances on one host. Containers enable us to isolate a group of processes from the others on a running Linux system. By using Linux kernel's resource management and resource isolation features (cgroups and name spaces), these processes have their own private view of the operating system with its own process ID (PID) space, file system and network interfaces. All containers on one host share the same kernel with anything else that is running on it. Each Container can be constrained to only use a defined amount of resources such as processor, memory or I/O.

Moreover, we are using polyglot programming and polyglot persistence. Each Docker container uses programming languages and data storage, which best fits to its requirements. In case of polyglot programming, code is written in different languages to capture additional functionality and efficiency not available in a single language. Similar to polyglot programming, polyglot persistence employs the best data storage solution for the use case. Each container employs its own data storage.

RBEE consists of several independent self-contained Systems (SCS) [6]. The SCS approach focuses on a separation of functionality into several independent systems. The whole system is a collaboration of many smaller systems, which avoids the problem of large monoliths that grow constantly and eventually become unmaintainable.

3 RBEE Architecture

RBEE consists of several independent self-contained systems (SCS). Figure 1 shows a conceptual view on RBEE's architecture. Each SCS is autonomous and have well defined interfaces. All data required for the SCS's domain and all application logic resides within the SCS. Everything necessary to fulfill its primary task is contained within the particular SCS. There is no interaction with any other SCS required. In the remainder of this section we will describe each SCS in more detail.

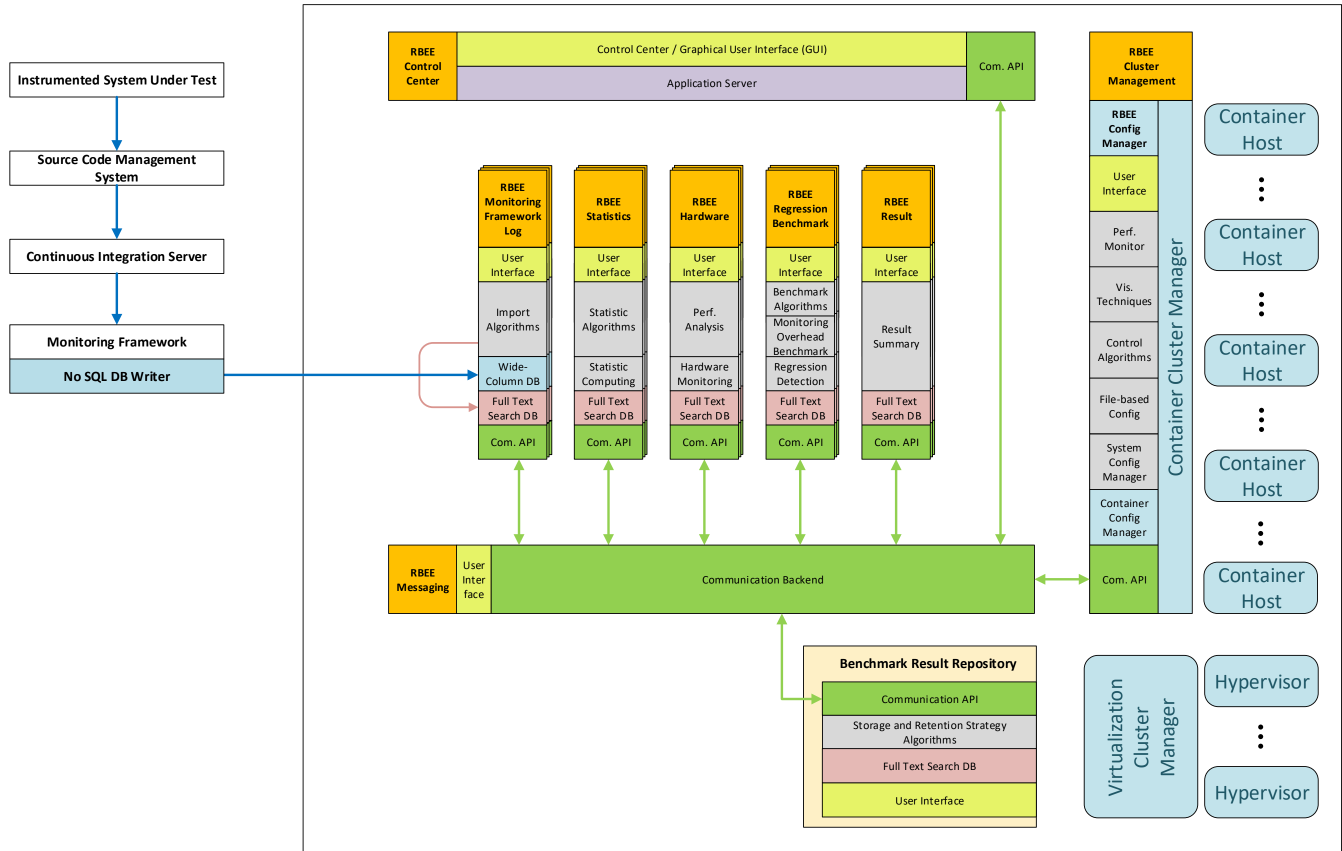

Fig. 1: RBEE Architecture Concept View

In addition to the conceptual view in Figure 1, we provide an implementation view in Figure 2.

Every interaction between the SCSs or third party systems is asynchronous. For this purpose we employ RabbitMQ [7] within our RBEE Messaging SCS [8]. It is an Erlang-based message broker, which distributes messages from one SCS to another, following well defined delivery rules. In addition to the message-based communication, this SCS implements a REST-proxy for data transfers between the different SCS. For this purpose we use HAPROXY [9]. So, we are able to decouple our SCSs from each other. There is no direct communication between the different SCS.

For RBEE's central management we provide the RBEE Control Center SCS [10]. It is the only SCS the user interacts with. It provides a graphical user interface. This is built as Java Server Faces web application, which is hosted on an Apache Tomcat [11] application server.

RBEE uses Kieker's monitoring output as input for its benchmarking process. Therefor, our RBEE Messaging Log SCS [12] acts as gateway between Kieker and RBEE. Monitoring output is transferred directly from a Kieker instrumented system under test (SUT) to the write-optimized noSQL storage within the RBEE Monitoring Log SCS. We employ Apache Cassandra [13] for this part. Additionally, we employ ElasticSearch [14] as read-optimized noSQL data storage. Read-optimized and write-optimized data storage are linked asynchronous via our rbee_cte command line tool [15]. When Kieker finishes writing monitoring output to Apache Cassandra, rbee_cte starts migrating data to ElasticSearch using its Java API.

When the data migration from Cassandra to ElasticSearch within the RBEE Monitoring Log is finished, RabbitMQ is used to inform the RBEE Control Center. The Control Center triggers the RBEE Config Manager SCS to provide an instance of the RBEE Statistics SCS. When the RBEE Statistics Docker container image is running, it notifies the RBEE Control Center, which triggers the RBEE Statistics SCS to pull the data using REST through the RBEE Messaging SCS.

Our RBEE Hardware SCS provides algorithms for hardware monitoring and hardware performance analysis. When the SUT is not instrumented with Kieker, RBEE supports manual instrumentation as well as instrumentation via aspect oriented programming [16, 17]. To make measurement results comparable, RBEE Hardware generates reference performance values for the used hardware before benchmarking the SUT.

The RBEE Benchmarking SCS collates all gathered monitoring data and sets it in relation to past benchmarks of the same SUT. Data about past benchmarks is stored in our Benchmark Result Repository virtual machine. Further, it generates hints for detecting performance regressions and can specify the caused monitoring overhead [18].

When benchmarking is completed, our RBEE Result SCS aggregates the gathered data and transmits it to our Benchmark Result Repository.

All data stored within the SCSs is not persistent. Once, the SCS shuts down, all data is lost. For storing

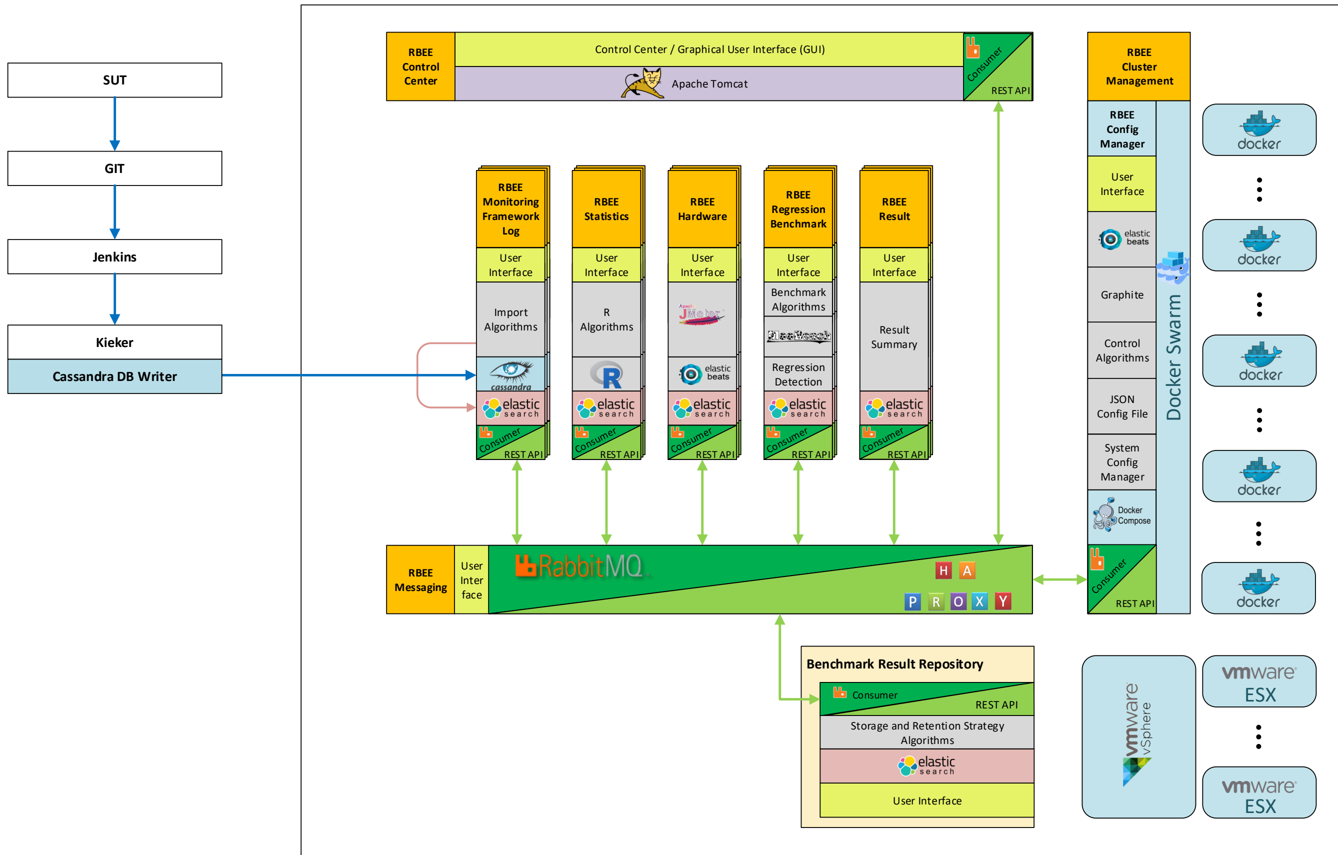

Fig. 2: RBEE Architecture Implementation View

persistent data, we employ our Benchmark Result Repository (BMR). Basically, this is a Linux-based virtual machine. It contains an ElasticSearch instance and algorithms for data management. All benchmark results are stored within this ElasticSearch instance. So, future benchmarks can be compared with the stored ones. The BMR can be accessed via the RBEE Control Center.

4 Conclusions and Future Work

We have shown, how the usage of open source software can ease our development process. By using well established open source software as foundation for RBEE, we can concentrate on our main objective. Publishing RBEE on GitHub and DockerHub enables us to ensure an easy deployment. Users can download the RBEE Docker images including the latest RBEE releases. RBEE's development is still in progress.

Bibliography

[1] *Regression Benchmarking Execution Environment*. http://www.rbee.io/.

[2] André van Hoorn, Jan Waller und Wilhelm Hasselbring. „Kieker: A Framework for Application Performance Monitoring and Dynamic Software Analysis“. In: *ICPE 2012*, S. 247–248.

[3] *Kieker Monitoring Framework*. http://www.kieker-monitoring.net/.

[4] Armin Moebius und Sven Ulrich. „Improving Kieker's Scalability by Employing Linked Read-Optimized and Write-Optimized NoSQL Storage“. In: *Symposium on Software Performance 2016 (SSP '16)*. 2016.

[5] *Docker*. http://www.docker.com/.

[6] *Self-Contained Systems*. http://scs-architecture.org/.

[7] *RabbitMQ*. https://www.rabbitmq.com/.

[8] *RBEE Messaging Docker Image*. https://hub.docker.com/r/rbee/mesg/.

[9] *HAPROXY*. http://www.haproxy.org.

[10] *RBEE Controlcenter Docker Image*. https://hub.docker.com/r/rbee/controlcenter/.

[11] *Apache Tomcat*. http://tomcat.apache.org/.

[12] *RBEE Monitoring Log Docker Image*. https://hub.docker.com/r/rbee/mlog/.

[13] *Apache Cassandra*. http://cassandra.apache.org/.

[14] *ElasticSearch*. http://www.elastic.co.

[15] *RBEE Monitoring Log Command Line Tool*. https://github.com/rbee-dev/mlog/.

[16] Gregor Kiczales u. a. „Aspect-Oriented Programming“. In: *ECOOP*. 1997, S. 220–242. URL: http://dx.doi.org/10.1007/BFb0053381.

[17] Thilo Focke u. a. „Instrumentierung zum Monitoring mittels Aspekt-orientierter Programmierung“. In: *Tagungsband Software Engineering 2007*. Lecture Notes in Informatics. 2007, S. 55–58. URL: http://eprints.uni-kiel.de/14525/.

[18] Jan Waller. „Performance Benchmarking of Application Monitoring Frameworks“. PhD Thesis. Faculty of Engineering, Kiel University, 2014. URL: http://eprints.uni-kiel.de/26979/.