published as: Thomas Staubitz, Hauke Klement, Ralf Teusner, Jan Renz, Christoph Meinel: CodeOcean - A Versatile Platform for Practical Programming Excercises in Online Environments, In Proceedings of 7th Annual IEEE Global Engineering Education Conference (EDUCON2016), 10-13 April 2016, Abu Dhabi
2016 ANNUAL IEEE GLOBAL ENGINEERING EDUCATION CONFERENCE

CodeOcean - A Versatile Platform for Practical Programming Excercises in Online Environments

Thomas Staubitz, Hauke Klement, Ralf Teusner, Jan Renz, Christoph Meinel
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
{thomas.staubitz, ralf.teusner, jan.renz, christoph.meinel}@hpi.de
hauke.klement@student.hpi.de

Abstract—**The paper at hand introduces CodeOcean, a web-based platform to provide practical programming exercises. CodeOcean is designed to be used in Massive Open Online Courses (MOOCs) to teach programming to beginners. Its concept and implementation are discussed with regard to tools provided to students and teachers, sandboxed and scalable code execution, scalable assessment, and interoperability. MOOCs bear a tremendous potential for teaching programming to a large and diverse audience. Learning to program, however, is a hands-on effort; watching videos and solving multiple choice tests will not be sufficient. A platform, such as CodeOcean, to work on practical programming exercises and to solve actual programming tasks is required. Due to the massiveness of the courses, teaching teams cannot check, give feedback, or assess the submissions of the participants manually. CodeOcean provides the participants with proper automated feedback in a timely manner and is able to assess the given programming tasks in an automated way.**

Keywords—MOOC, Hands-on Experience; Online Assessment; Scalability; E-Learning; Automated Assessment; Programming.

I. Introduction

In today's society, technological innovation plays an ever-increasing role for a country's development and economic growth [1]. Technology touches virtually every part of our daily lives. As a consequence, programming abilities are required in many professional areas. Programming has become a key qualification of the 21st century. In recent years, Massive Open Online Courses (MOOCs) have become a phenomenon presenting the prospect of free high class education to everybody. They bear a tremendous potential for teaching programming to a large and diverse audience. The typical MOOC components, such as video lectures, reading material, and easily assessable quizzes, however, are not sufficient for proper programming education. To learn programming, participants need to work on practical programming exercises and to solve actual programming tasks. It is further crucial that the participants receive proper feedback on their work in a timely manner. Thus, without a tool for automated assessment of programming assignments, the teaching teams would be restricted to offer optional ungraded exercises only. In a previous paper [2], we have shown that similar tools have a long history in computer science and have dealt with the question how MOOCs can integrate practical programming assignments in a manner that meets the demands of novice programmers and satisfies the inherent scalability requirements of large-scale e-learning environments. In the paper at hand, we introduce CodeOcean, a web-based platform for practical programming exercises, which is designed to be used in programming MOOCs. CodeOcean allows teachers to create programming exercises, which can be automatically graded by employing unit tests as a measure of quality. Its design and implementation are discussed with regard to tools provided to students and teachers, sandboxed and scalable code execution, scalable assessment, and interoperability. CodeOcean aims at facilitating the entry into programming and at attracting a diverse audience of interested learners. While the application is designed to be novice-friendly, it is not specifically tailored to beginner-oriented programming paradigms. Rather, it is designed to support a wide range of programming languages in a fashion that encourages novices, yet is not limited to trivial programming tasks. CodeOcean has already been employed to a different extend in three courses on openHPI.

The remainder of this paper is structured as follows: Sections II, III, and IV present the requirements, design, and some details of our initial implementation of CodeOcean. Section V presents the results of an early load simulation test. In the following sections, we conclude our findings, and give an outlook on our upcoming plans.

II. Determining the Requirements

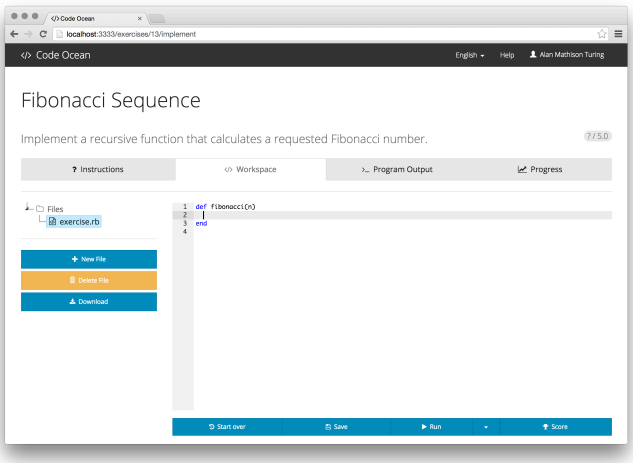
To start with, we defined five high-level requirements– versatility, novice-friendliness, scalability, security, and interoperability–that CodeOcean (and large-scale programming education solutions in general) needs to comply with. The following paragraphs will highlight these requirements one by one.

a) Versatility: Many of the programming tools that we examined are designed for a single use case; e.g. *Webpython*, one of the predecessors of CodeOcean at openHPI, only supported exercises in the Python programming language. One of our requirements is to support a wide variety of programming languages and application domains. A further requirement is to enable teachers to create practical assignments that can make use of third-party applications and libraries. As a consequence, CodeOcean's approaches towards program execution and code assessment had to be chosen with flexibility in mind. According to Pieterse [3], the quality of automated assessment is largely dependent on the quality of the test cases that are used. Therefore, we required that CodeOcean is able to promote teachers' creativity in assessment design by providing them the freedom to decide which program aspects to assess and which tools to use for this purpose. CodeOcean is required not to dictate a universal assessment approach but to permit the usage of any desired tool that fits the particular use case best, such as an established testing framework or a tailor-made solution.

b) Novice-Friendliness: The large diversity of MOOC participants implies that classmates lack a common knowledge base and educational background [4]. Therefore, learners' prior knowledge and digital literacy vary considerably. The first use cases on openHPI to employ CodeOcean in practice were courses that addressed complete beginners. We require that learners are provided with a homogeneous development environment that has a simple and appealing User Interface (UI), requires no prior knowledge, and supports them in many aspects of their endeavor to learn programming. We also require that CodeOcean minimizes the challenges that the usage of an automated assessment tool may entail (see also [3]). A very important aspect of the learning process is feedback. Feedback towards assignments allows students both to understand their mistakes and to revise their work [5]. Compared to a traditional learning setting, feedback quality is even more important in MOOCs because communication opportunities are limited [3]. We consider providing students with understandable and useful feedback as crucial for their long-term motivation and as an important requirement for CodeOcean, particularly, since both MOOCs and programming courses in general are affected by high dropout rates [6].

c) Scalability: As MOOCs are aimed at unlimited numbers of participants, they need to be inherently scalable [7]. Any tool to be employed in a MOOC, obviously needs to provide this inherent scalability as well. In our specific use case, many students must be enabled to write and execute code in parallel. Moreover, a certain level of responsiveness is required in order to achieve a satisfying User Experience (UX) [8]. Therefore, CodeOcean is required to follow a code execution approach that provides fast feedback and that scales for the number of users to be expected in a MOOC. The same scalability requirements also apply to assessment. Huge enrollment numbers in MOOCs make manual feedback and grading impossible. Instead, CodeOcean needs to provide a scalable assessment approach that fits the needs of large-scale education.

d) Security: Server-side execution of student-written programs implies that arbitrary code is executed within the boundaries of an e-learning system. This constitutes a considerable risk. Faulty student programs could excessively consume server resources; intendedly malicious programs could try to cause damage or obtain unauthorized access. In fact, automated assessment systems that are integrated into Learning Management Systems (LMS) are considered a tempting target for attackers [9]. Due to these risks, providing a secured execution environment for running students' programs is regarded to be an essential requirement for automated assessment systems that employ dynamic evaluation techniques [10]. CodeOcean is required to provide means for the sandboxed execution of learners' code that guarantee that untrusted code can neither harm the platform nor influence other learners' code submissions.



Fibonacci Sequence

Implement a recursive function that calculates a requested Fibonacci number.

```
def fibonacci(n)

end

```

Fig. 1. CodeOcean: In the assignment workplace the user is presented the problem at hand to be solved.

e) Interoperability: We believe that educational programming platforms are more widely adopted if they can be easily integrated into existent e-learning infrastructures. Therefore, CodeOcean is required to be interoperable with existing e-learning systems, such as LMSs and other MOOC platforms, next to openHPI. In order to extend their courses' contents with practical programming tasks, instructors should be able to prepare assignments on the CodeOcean platform and embed them into their courses. Learners, on the other hand, should be able to solve these assignments in a transparent manner, without the need for registration.

III. Design and Components

A. Development Environment and User Interface

Based on the gained insights that have been discussed in [2] and the requirements as defined in the previous section, we decided in favor of a web-based development environment composed of a client-side code editor and a server-side component for code execution. This approach entails a number of advantages. At first, it allows us to provide learners with a homogeneous and novice-friendly programming environment. Secondly, the approach supports a variety of programming languages and third-party libraries while providing a consistent workflow for both code execution and assessment. Thirdly, the approach enables insights into learners' problem-

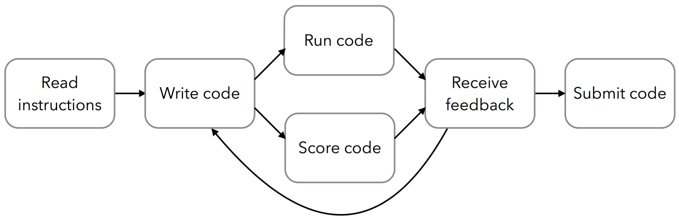
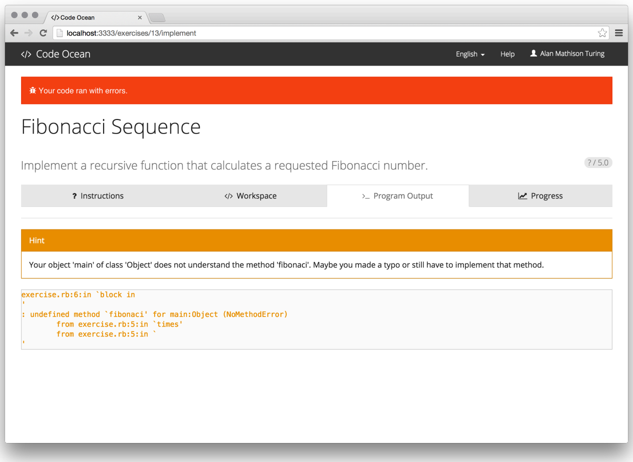


Fig. 2. Development Process

solving strategies by analyzing their code submissions. All web-based programming tools used in MOOCs that we compared[1] are restricted to a single unit of editable code. In contrast, we decided that CodeOcean should promote the concept of files. We believe that it is important to support multiple editable files and the creation of new files since this enables more profound programming exercises, fosters learners' creativity and flexibility, and empowers learners to practice program design [3]. Furthermore, we want to provide learners with the ability to explore the behavior of the code they wrote by running it and having it assessed as frequently as desired.

CodeOcean's development environment is based on widespread web standards that are natively supported by current web browsers. Non-native technologies, such as Java applets and third-party plugins, are avoided (see also [11]). Currently out of scope of this paper, are customization, debugging, and refactoring features, as these are negligible for our current purpose of teaching programming to novices. The development environment's UI is shown in the Figures 1, 3, and 4. The upper part of the view contains the active exercise's title and description, as well as the most recent score as determined by running the learner's solution against the exercise's tests. The navigation bar at the top of the window allows to control the UI's language and to access help. The UI is available in the locales English and German. Further translations can be added with little effort. Although a certain proficiency in English is usually required in the field of programming [12], we decided to internationalize CodeOcean's UI. This is required, particularly, with regard to the courses for novices and children. The major part of the development environment was split up into four tabbed areas, each of which is associated to a step in learners' iterative development workflow depicted in Figure 2.


Code Ocean
localhost:3333/exercises/13/implement
Code Ocean — English — Help — Alan Mathison Turing

Your code ran with errors.

Fibonacci Sequence

Implement a recursive function that calculates a requested Fibonacci number. ? / 5.0

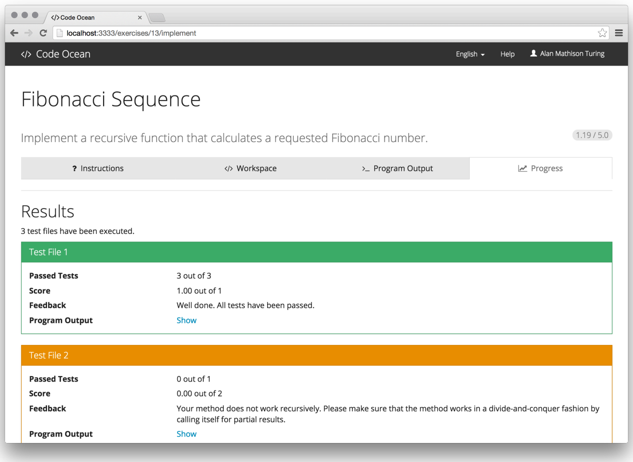
Instructions | Workspace | Program Output | Progress

Hint

Your object 'main' of class 'Object' does not understand the method 'fibonaci'. Maybe you made a typo or still have to implement that method.

```
exercise.rb:6:in `block in
'
: undefined method `fibonaci' for main:Object (NoMethodError)
	from exercise.rb:5:in `times'
	from exercise.rb:5:in `
'
```

Fig. 3. CodeOcean: Code has been run with errors. From the output we assume that the method name was misspelled (fibonaci vs. fibonacci).


Code Ocean
localhost:3333/exercises/13/implement
Code Ocean — English — Help — Alan Mathison Turing

Fibonacci Sequence

Implement a recursive function that calculates a requested Fibonacci number. 1.19 / 5.0

Instructions | Workspace | Program Output | Progress

Results

3 test files have been executed.

Test File 1

Passed Tests	3 out of 3
Score	1.00 out of 1
Feedback	Well done. All tests have been passed.
Program Output	Show

Test File 2

Passed Tests	0 out of 1
Score	0.00 out of 2
Feedback	Your method does not work recursively. Please make sure that the method works in a divide-and-conquer fashion by calling itself for partial results.
Program Output	Show

Fig. 4. CodeOcean: Code run against teacher defined test cases. Tests can be defined in separate files. Per file any number of tests is allowed, however points can only be rewarded on a per file basis.

As it turned out during the first two courses, the process in Figure 2 is flawed. Reading the instructions cannot be regarded as a separate step. It is a process that occurs constantly throughout the work on the task. This had to be reflected in the UI. The first tab used to display the exercise instructions including the problem description, expected program behavior in edge cases, exemplary code snippets, and more. As the assumed underlying process was flawed, the design turned out not to be successful. Participants complained that they could not see the instructions while working on the solution. Therefore, teaching teams often used the *short description*, which is visible throughout the process at the top of the page, to provide the detailed instructions. The place where the instructions originally were intended to be displayed, often remained unused and eventually the first tab was removed completely. The (originally) second tab (see Figure 1) hosts the development environment's core component, which is the programming workspace. This workspace consists of two elements: a file tree and a code editor. The teacher is enabled to decide for each file if it is to be shown in the file tree or if it should be hidden. The content of files that are shown in the file tree can be accessed by the participants, the teacher decides if the file is writeable or read-only. The file tree also allows users to create, delete, and download files themselves. The ability to create custom workspace files enables learners to practice

[1] The results of this comparison–we examined several courses on Coursera, iversity, edX, Udacity and openHPI and the coding environments they employed–will be published more detailed in a separate paper.

program design and modularization by splitting up their code into functional units of their own choice. For novices, particularly in the courses that address children and adolescents, this is not required and turned out to be confusing for the participants. Therefore, the option to hide the file tree (including the buttons to create and delete files) was added for the teachers.

CodeOcean's web-based development environment is based on Ace[2], an embeddable open-source code editor, written in JavaScript. We chose Ace from the set of available code editors due to its rich functionality, good reputation, and active maintenance. Ace offers source code editing capabilities that match the functionality and performance of native desktop editors. Its rich feature set includes syntax highlighting for a myriad of programming languages, UI theming, code folding, automatic indent, keyboard shortcuts, find-and-replace functionality, and more.

The remaining tabs contain the program output and the test results, which will be discussed in the following chapters. The tabbed design is, currently, subject of discussion and will be redesigned in a more usable way asking less clicks from the user to finally submit her solution.

See also Figure 1, 3, and 4 to get an impression of CodeOcean from the participant's perspective.

1) The participant has opened CodeOcean and is presented a programming problem to solve (Figure 1.)

2) The participant has tried to run her code but the execution failed due to a syntax error (Figure 3.)

3) The participant has fixed the error and evaluated the code against the teacher-defined unit tests (Figure 4.)

B. Code Execution and Security

The execution of student-written code demands security measures since learners may submit programs that excessively consume resources or even cause damage to the system. A number of possible attacks against systems using automatic code evaluation have been described by Forišek [13]. Although it is far more likely that student programs are rather erroneous than deliberately malicious, providing our system in a MOOC context to a large number of learners makes it plausible that individual users may attempt to gain unauthorized access or do harm [3]. Therefore, CodeOcean has to provide a secured environment for running student programs that restricts the amount of consumable resources and withstands the damage that faulty or malicious programs may cause. Isolation and resource control have traditionally been achieved through the use of Virtual Machines (VM). However, abstraction provided by VMs comes at the cost of reduced performance. In order to meet the scalability requirements of MOOCs, OS-level virtualization techniques present an interesting alternative to traditional VMs since they impose almost no overhead. Rather than running a full OS on virtual hardware, OS-level virtualization approaches leverage built-in OS capabilities that enable isolated environments without starting a VM. Unlike a VM, such an environment can comprise as little as a single process and only owns the resources that it actively consumes. Linux Containers (LXC) is one of several OS-level virtualization methods that provide multiple virtual environments on a single host system. LXC is based on control groups[3] and namespaces, which are OS features that allow limiting and isolating the resources used by groups of processes. Therefore, virtual environments, so-called containers, have their own process and network spaces and cannot see or access objects on the outside. Although LXC's underlying concepts are well known and mature, it has only recently been adopted and standardized in mainstream OSs. Due to its benefits, container-based virtualization seems to be predestined for our use case. It allows to run students' programs in isolated environments that are practically unrestricted in terms of executable software while involving hardly any virtualization overhead. Containers represent a very flexible platform for diverse code evaluation tasks that is open to any programming language and third-party library available for Linux. The isolation provided by LXC for now dispenses us from measures such as integrating a stand-alone sandboxing solution [9]. An element of risk remains, however, if a user manages to escape from the container. Therefore, further securing the containers is part of our ongoing research and will be discussed in a future paper.

Since every single code submission is executed in a clean, preconfigured container being in a known state, start-up or reset times are important. Low latencies for code execution are a precondition for providing results and feedback in a timely manner. Small feedback times grant learners a more interactive development process and facilitate iterative problem-solving strategies. In contrast to traditional VMs, LXC entails much less overhead for starting the virtualization platform, which is why the execution of a code submission usually involves an overhead of less than a second. Therefore, an interactive development workflow is enabled.

The open-source software Docker[4] provides an abstraction layer on top of LXC, including an image format and convenient tools for building, versioning, distributing, and deploying containers. Although several management tools for LXC exist, Docker has emerged as the de facto standard [14]. We decided to employ Docker as the execution platform for students' code submissions because it offers competitive performance [14] as well as user-friendly tools that allow instructors to define custom execution environments by themselves. Docker involves the concept of images, which are stateless templates for containers that are used to prepare applications for execution in a Docker container. Since most applications rely on third-party utilities, libraries, or services, images enable users to package such applications along with their dependencies [15]. Existing Docker images can be used as starting points for the definition of new ones. Therefore, common dependencies can be bundled in a general base image to be used by multiple other im-

[2] https://ace.c9.io/#nav=about

[3] https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[4] https://www.docker.com/

ages. Besides open-source software, Docker offers Docker Hub[5], a web-based repository for Docker images. Users can push their images to the repository and fetch them on another machine. By providing images publicly, they can be easily shared. In order to prepare an execution environment that is tailored to the needs of a particular course, teachers are expected to create a corresponding Docker image and publish it at Docker Hub. After that, CodeOcean can pull the image from the repository and utilize it for the execution of students' code submissions. Docker provides two means for the creation of images. Firstly, an image can be created manually by making changes to a container and committing the results to a new image. This approach allows to evolve existing images in a simple way, but it does not promote automation and collaboration. Secondly, Docker provides a tool for building images automatically from a list of instructions, as well as a DSL[6] for specifying the concrete steps to be taken. Using so-called *Dockerfiles*, images can be composed and adjusted in a textual fashion, which allows automated and reproducible creation. In addition to the capabilities provided by Docker Hub, *Dockerfiles* enable collaboration and version control using standard source code management tools, repositories, and practices.

C. *Code Assessment and Feedback*

In [2] we underlined the relevance of assessment for the learning process and presented multiple approaches towards scalable assessment of programming assignments. Due to its focus on large-scale e-learning environments, such as MOOCs, scalable assessment is a crucial requirement of CodeOcean. Since the purpose of CodeOcean is to provide an appropriate platform for teaching programming to everyone, including complete beginners, we decided, for now, to rely on automated assessment techniques rather than peer assessment. Furthermore, automated assessment provides highly available and objective evaluation which makes it a predestined approach to supply learners with means for step-by-step refinement of their solutions before they finally submit their work. An integration with openHPI's peer assessment mechanism is planned, however, in the future. This will enable us to support peers in assessing the submissions by giving them the information that the submission has fulfilled the basic requirements: the code compiles and produces the requested outcome. Thus, the peers can concentrate on code quality and providing qualitative feedback rather than having to deal with the basics. The application of additional manual assessment capabilities is not within the scope of this paper and will be subject of future research. In the context of automated assessment, we decided not to dictate a universally applicable assessment approach, such as I/O-based testing, but to grant instructors the freedom to select an assessment strategy that they consider appropriate for a specific use case. Docker provides a versatile platform for executing the tests for assessment. The employed mechanism is very similar to the one that is used to run the student's code. Instructors are very flexible in their choice of a particular assessment strategy. For instance, they are free to use an arbitrary testing framework, such as the one that is best practice for the language they teach, the one that fits the application domain best, or the one that they are most experienced with. Besides relying on industry-strength testing tools, instructors can also choose to utilize tailor-made scripts for their assessment workflows. However, we encourage instructors to favor well-known solutions over improvised ones since established testing frameworks usually supply greater functionality and robustness, and can provide learners with relevant experience in using them. Since the system's underlying assessment approach is visible to learners, they get in contact with the concept of software testing from the very beginning. Besides imparting an objective quality to the assessment, test-based evaluation provides learners with an understanding of the primary method for verifying industrial software [16]. Moreover, learners become accustomed to the idea of software testing as a means for controlling software quality and might be more willing to write their own tests later on [17]. Instructors are free to provide the tests they use for assessment as a visible part of the exercise skeleton, in this way permitting even deeper insights into the testing approaches used by professional developers. Furthermore, CodeOcean has the ability to be employed in courses on test-driven development by assessing if the tests that are submitted by the students are testing at least as much as the tests that have been provided by the teachers.

To take full advantage of the role of assessment as a feedback channel for learners, the results of teacher-provided test cases should convey learners a good understanding regarding the extent to which their code adheres to the exercise specification. The output generated by a testing framework can be confusing for beginners [6]. Inexperienced programmers might find it difficult to match the feedback supplied with a failing test to errors in their code [18]. In order to facilitate learners' troubleshooting, we supplement test frameworks' low-level output with instructor-provided feedback that is better understandable. Teachers are encouraged to provide understandable natural-language feedback for every test. Consequently, in the case of a failing test, the learner is supplied with a useful hint on how the program's behavior does not fulfill the specification and how it can be improved. By providing the student's with a clear understanding of their program's inadequate aspects, we intend to increase the students' motivation to revise their solution.

D. *Interoperability, User Management, and Scoring*

Instead of designing CodeOcean as a proprietary component for a single e-learning platform, such as openHPI, or adding course management features to its scope, we decided to build a lean stand-alone application that is interoperable with existent e-learning systems. The main issues in this approach are to enable the users of the e-learning platform to access CodeOcean without being required to have an additional account, and to transfer the points or grades that they received on CodeOcean back to the e-Learning platform where they started. The de-facto standard Learning Tools Interoperability (LTI) [19] interface provides exactly these functionalities. By imple-

[5] https://hub.docker.com/

[6] http://docs.docker.com/reference/builder/

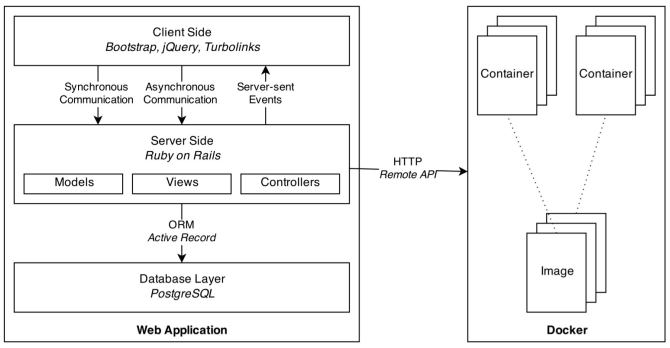


Fig. 5. CodeOcean: Three Tier Architecture plus Docker Server.

menting the LTI interface, CodeOcean is interoperable with a wide range of applications that are compliant to the same standard. These applications include the popular open-source LMSs Canvas[7], Moodle[8], and Sakai[9], commercial LMSs, such as Blackboard, and the MOOC platforms Coursera, edX, and openHPI. LTI is a specification developed by the IMS Global Learning Consortium[10]. It is aimed at establishing a standard for integrating remote content and third-party services into e-learning applications. In LTI lingo, these third-party services are called tools. Tools are hosted and supplied by so-called tool providers. E-learning applications that utilize tools are referred to as tool consumers. CodeOcean implements the LTI specification in version 1.1.1[11]. It covers the following mechanisms for interaction between tool providers and tool consumers:

- The provisioning and installation of external tools in e-learning applications.
- A tool launch protocol for sending a tool consumer's user to a tool provider while securely providing user identity, user role, and course context.
- Runtime web services that allow tool providers to create, retrieve, and delete results for users.

CodeOcean uses these capabilities in order to provide its services to trusted consumer applications, to receive user information regarding learners who start a programming session, and to send learners' results back to their consumer applications. The described mechanism has some implications in the context of scoring points for the assessments. According to the LTI 1.1 specification the score that is transferred from the tool provider to the tool consumer is always a value between 0 and 1. The tool consumer is in charge of the ultimate decision how much points will be rewarded for a certain assignment. While CodeOcean, generally, allows to write several (unit) tests for assessment in one file, it is only possible to reward points per test file. To provide a fine grained scoring model, the tests, therefore, need to be provided in separate files per test. A student receives at least one point per succeeding test. The teacher is free, however, to assign more than one point to a given test file, to emphasize the value of this certain test. Thus, it is pos-

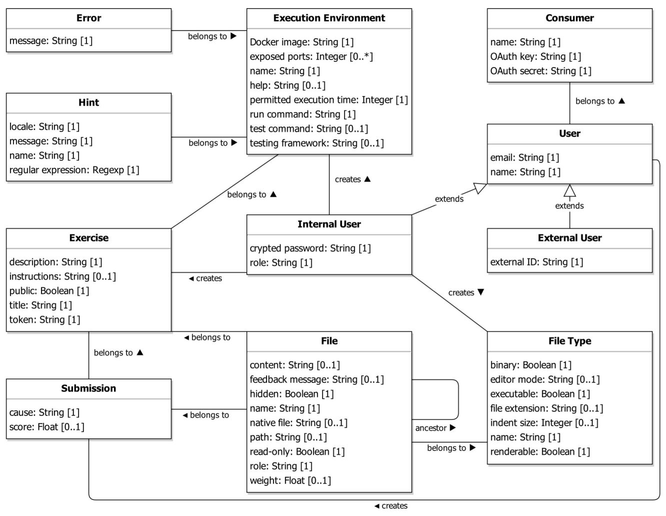


Fig. 6. CodeOcean: Domain Model.

sible to end up with an amount of points that might not be appropriate in the global context of the course. In order to map the results of a test run to a numerical grade, CodeOcean calculates a score that is based on the ratio of passed tests to failed tests. This results in a value that is within the range of 0 and 1, and thus, can be transferred to the tool consumer via LTI. The tool consumer then multiplies this value with the amount of points that are considered to be appropriate within the global course context. Thus, it happens that the points rewarded by the tool provider differ from the points rewarded by the tool consumer. CodeOcean's original UI displayed the absolute number of points (as rewarded by CodeOcean) on the exercise's result page in the form of (*x Points/max Points*). When the points that a user received on CodeOcean differed from the points that were rewarded for the same exercise on openHPI, particularly when the differences were small, users got confused and suspected a bug in the system. We, therefore, replaced the display of absolute points with a display of the achieved percentage.

IV. SOME IMPLEMENTATION DETAILS

A. Architecture

As depicted in Figure 5, our solution is composed of a three-tiered web application based on Ruby on Rails (RoR)[12] and a Docker server. The web application provides the development environment for learners as well as an administration back-end for teachers. Docker is used for code execution and assessment. The two core components can either be hosted on the same machine or distributed on different hosts. The web application communicates with the Docker server using its HTTP-based Remote API[13]. For this purpose, it utilizes an object-oriented interface to the API, which is provided by the docker-api[14] Ruby gem[15]. Client-server communication is

[7] https://www.canvas.net/
[8] https://moodle.org/
[9] https://sakaiproject.org/
[10] http://www.imsglobal.org/
[11] http://www.imsglobal.org/lti/v1p1p1/ltiIMGv1p1p1.html

[12] http://rubyonrails.org/
[13] https://docs.docker.com/reference/api/docker remote api/
[14] https://github.com/swipely/docker-api
[15] Ruby gems are packaged Ruby modules that provide a certain functionality. Often these gems are available for free under one of the open source licenses

heavily based on Asynchronous JavaScript and XML (AJAX). Asynchronous background requests enable a single-page development workflow for the web-based development environment. Moreover, Rails' Turbolinks[16] feature improves page load times when navigating through the application by partially reloading visible content instead of performing full page loads. In our scenario, PostgreSQL[17] is employed as the database as it fits best in our landscape. As RoR adds an additional abstraction layer in form of an object-relational mapping (ORM) the database can be exchanged easily for other scenarios.

B. Domain Model

Figure 6 depicts the application's domain model in the form of a Unified Modeling Language (UML) class diagram. The diagram is limited to the most relevant attributes. The most important concepts will be introduced now in short.

a) Consumer: The LTI tool consumer as introduced in Section III-D. Tool consumers have to be registered with CodeOcean. This is a manual process. Administrators of tool consumers that are interested in using our instance of CodeOcean have to request this access. We can then in turn figure out if we will be able to provide the resources for the requested use case in the given timeframe. Alternatively, as CodeOcean has been open sourced, interested parties can host an instance of CodeOcean in their own landscape. In this scenario as well, the intended tool consumer has to be registered with the CodeOcean instance.

b) User: For now, three types of users are implemented: *Administrators* manage tool consumer applications and platform users, *Teachers* create content and examine learners' performance, while *Learners* solve programming exercises. Whereas, teachers and administrators are internal users, who are registered directly at CodeOcean and also can log in without any detours, learners are mere visitors that are sent from a consumer application, complete their task, and return to that consumer application right away. A dedicated standalone login as it exists for teachers and admins, is not available for learners.

c) Execution Environment: An execution environment describes a software platform, which is used for the execution of code submissions. An execution environment's central element is its Docker image, which provides an operating system as well as third-party applications and libraries. Depending on the execution environment's needs, the permitted execution time for student-written code and a number of exposed ports can be specified. Furthermore, *run* and *test* commands need to be defined for each environment.

d) Exercise: An exercise belongs to an execution environment. The exercise's creator can decide whether it is public, and therefore visible to other teachers, or not. Each exercise has a uniquely generated token that is used for referencing the exercise when embedded by means of LTI.

[16] https://rubygems.org/gems/turbolinks
[17] http://www.postgresql.org/

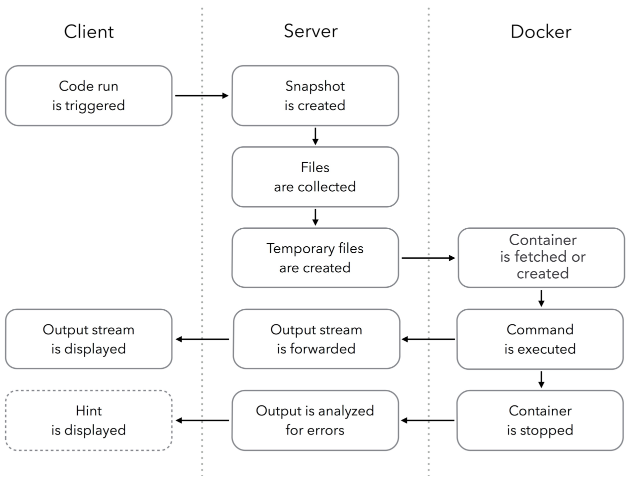

Client
Server
Docker
Code run is triggered
Snapshot is created
Files are collected
Temporary files are created
Container is fetched or created
Output stream is displayed
Output stream is forwarded
Command is executed
Hint is displayed
Output is analyzed for errors
Container is stopped

Fig. 7. Code execution workflow.

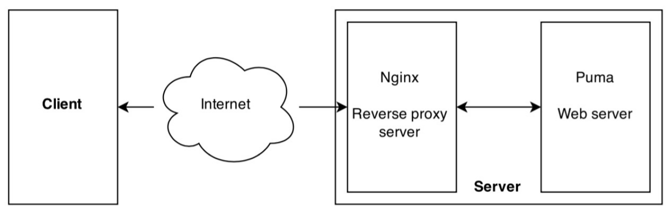
e) Submission: A submission is a snapshot of a user's ongoing implementation of an exercise. The complete snapshot history is stored with timestamps in the database.

f) File, Error: Files exist either as part of an exercise or as part of a submission. The content of a binary file is stored in the native file system, whereas a non-binary file's content is stored in the database. Finally, errors that occur during the execution of learners' code are stored and aggregated in order to provide teachers with a guideline towards their students' common misconceptions.

C. Code Execution

Whenever a learner triggers a code run from the development environment, her current implementation progress is sent to the server using AJAX. A snapshot is created and stored. To execute a code submission, a new or resetted Docker container, based on the Docker image of the exercise's execution environment, is provided. Every code execution starts with a blank slate, which prevents potential side effects of previous code executions. Based on the execution environment's configuration, a number of network ports can be exposed by the Docker container during its runtime in order to allow a student to send and receive data. To avoid port collisions among simultaneously active learners, a pool of available ports is maintained, which provides mutually exclusive access to ports. To supply the container with the necessary files, the exercise's skeleton files plus student-written files are collected from the database and are written to a submission-specific temporary directory. Depending on the physical location of the application server in relation to the Docker server, the files can be placed in a shared folder, explicitly transferred over a network, or made available through a network or Cloud file system. In the end, the submission-specific directory is mounted as a data volume into the container's file system.

More often than experienced programmers, novices write non-terminating code, which can block available server processes and waste resources [20]. In order to restrict the amount of wasted computing power, CodeOcean puts a limit on Docker containers' permitted execution time. If a running



container's execution time exceeds the permitted duration, for example due to an infinite loop or never-ending recursion, the container is stopped and the learner is notified that she has built a non-terminating program. Besides cutting down infinite loops, limiting execution time can also be used as a measure of quality assessment since an efficiency limit for code submissions can be enforced [3].

D. Assessment

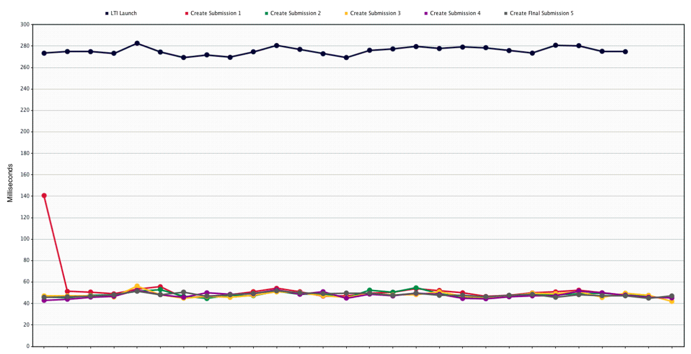
CodeOcean's assessment workflow is based on executing a learner's solution against a set of tests. Since these tests are invoked in the same manner as learners' main programs, the assessment execution workflow is very similar to the code execution workflow. Initially, an up-to-date code snapshot is

Fig. 8. Code execution workflow.

created. After that, the learner's work, exercise skeleton files, and, most important, the exercise's tests are written to a temporary directory, which is supplied to a Docker container. When the test runs are finished, test results are extracted from the testing framework's output. The final assignment score is calculated based on the single test files' weighted percentages of passed tests. In order to calculate a score and display it prominently in CodeOcean's UI, the application has to obtain key figures from the test run. Those are the number of executed test cases, the number of passed test cases, and the number of failed test cases. Since we decided to provide teachers with the flexibility to use an arbitrary testing framework of their choice, there is no single source of data but a wide range of testing frameworks, each of which providing a proprietary API and using a proprietary output format. In order to obtain the required key figures despite this inhomogeneity, CodeOcean utilizes framework-specific adapters that extract the required information from testing frameworks' textual output. Such an adapter has to be provided for every testing framework or family of related frameworks to be used with CodeOcean.

V. Evaluation

In order to evaluate the scalability of our application, we prepared a production environment that is appropriate for handling the number of concurrent users to be expected in a MOOC. We simulated a corresponding load using Apache JMeter[18].



A. Test Environment

The production server is equipped with two Intel Xeon E5-2680 v2[19] central processing units (CPUs), supplying 40 logical CPU cores in total, and 64 GB of memory. The server hosts all major components of CodeOcean, which are the web application, a web server, the database, and Docker. We use Ubuntu 14.04.1 LTS (64-bit), Docker 1.3.1, and PostgreSQL 9.3.5. As depicted in Figure 8, the web application is served by Puma[20] 2.10.1, a web server built for speed and concurrency, using Nginx21 1.6.2 as a reverse proxy. In order to make best use of the parallelism provided by the server's many-core CPU, we decided not to use Ruby's standard interpreter, which is Matz's Ruby Interpreter (MRI), but to rely on JRuby, its JVM-based equivalent. While MRI's Global Interpreter Lock (GIL) limits the multi-thread performance of a single

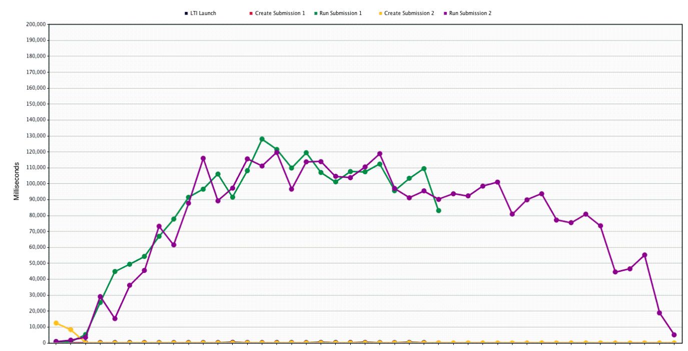
Fig. 9. Response Time Graph Depicting Requests for LTI Launch and Snapshot Creation.

interpreter process by allowing only one thread to execute at a time [21], JRuby offers thread-level parallelism by mapping Ruby threads to Java threads, which in turn are mapped to native OS threads by most JVMs.

In order to prepare the production environment for the planned load, we set the maximum numbers of database connections allowed by PostgreSQL and allocable by Active Record's connection pool to 1024. Moreover, Puma has been configured to use up to 64 threads.

B. Test Plan

The JMeter-based load test simulates 500 learners who use CodeOcean in parallel to solve a practical assignment. Each

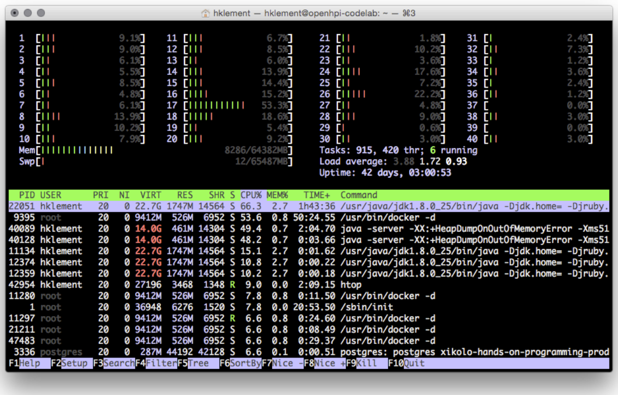


simulated student's programming session starts with launching

[18] http://jmeter.apache.org/

[19] http://ark.intel.com/products/75277

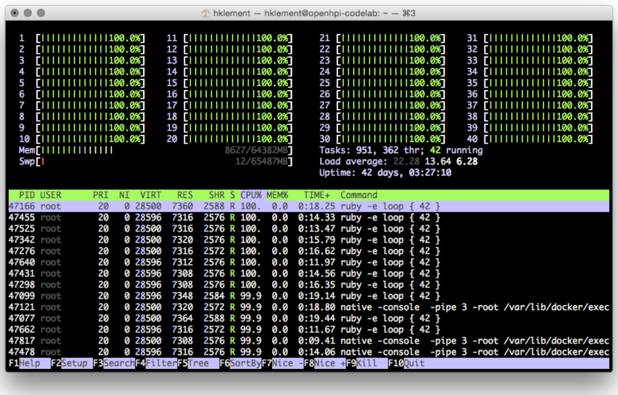
[20] http://puma.io/



```
hklement — hklement@openhpi-codelab: ~ — ⌘3

  1  [|||        9.1%]   11 [|||         6.7%]   21 [||          1.8%]   31 [|           2.4%]
  2  [|||        9.0%]   12 [|||         8.5%]   22 [|||        10.2%]   32 [||          7.3%]
  3  [||         6.1%]   13 [||          6.0%]   23 [||          3.6%]   33 [|           1.2%]
  4  [||         5.5%]   14 [|||        13.9%]   24 [||||       17.6%]   34 [||          3.6%]
  5  [|||        8.5%]   15 [|||        14.4%]   25 [||          7.2%]   35 [|           2.4%]
  6  [||         4.8%]   16 [||||       15.2%]   26 [|||||      22.2%]   36 [||          1.2%]
  7  [||         6.1%]   17 [||||||||||||53.3%]  27 [||          4.8%]   37 [            0.0%]
  8  [||||      13.9%]   18 [|||||      18.6%]   28 [|||         9.0%]   38 [||          3.0%]
  9  [||        10.2%]   19 [||          5.4%]   29 [|           0.6%]   39 [            0.0%]
  10 [|||        7.9%]   20 [|||         9.2%]   30 [||          3.0%]   40 [||          3.0%]
  Mem[||||||||||||||          8286/64382MB]   Tasks: 915, 420 thr; 6 running
  Swp[|                         12/65487MB]   Load average: 3.88 1.72 0.93
                                              Uptime: 42 days, 03:00:53

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
22051 hklement   20   0 22.7G 1747M 14564 S 66.3  2.7  1h43:36 /usr/java/jdk1.8.0_25/bin/java -Djdk.home= -Djruby.
 9395 root       20   0 9412M  526M  6952 S 53.6  0.8 50:24.55 /usr/bin/docker -d
40089 hklement   20   0 14.0G  461M 14304 S 49.4  0.7  2:04.70 java -server -XX:+HeapDumpOnOutOfMemoryError -Xms51
40128 hklement   20   0 14.0G  461M 14304 S 48.2  0.7  0:03.66 java -server -XX:+HeapDumpOnOutOfMemoryError -Xms51
11134 hklement   20   0 22.7G 1747M 14564 S 15.1  2.7  0:01.62 /usr/java/jdk1.8.0_25/bin/java -Djdk.home= -Djruby.
12374 hklement   20   0 22.7G 1747M 14564 S 10.8  2.7  0:00.22 /usr/java/jdk1.8.0_25/bin/java -Djdk.home= -Djruby.
12359 hklement   20   0 22.7G 1747M 14564 S 10.2  2.7  0:00.18 /usr/java/jdk1.8.0_25/bin/java -Djdk.home= -Djruby.
42954 hklement   20   0 27196  3468  1348 R  9.0  0.0  2:09.15 htop
11280 root       20   0 9412M  526M  6952 S  7.8  0.8  0:11.50 /usr/bin/docker -d
    1 root       20   0 36948  6276  1520 S  7.8  0.0 20:53.50 /sbin/init
11297 root       20   0 9412M  526M  6952 R  6.6  0.8  0:24.60 /usr/bin/docker -d
21211 root       20   0 9412M  526M  6952 S  6.6  0.8  0:08.49 /usr/bin/docker -d
47483 root       20   0 9412M  526M  6952 S  6.6  0.8  0:29.37 /usr/bin/docker -d
 3336 postgres   20   0  287M 44192 42128 S  6.6  0.1  0:00.51 postgres: postgres xikolo-hands-on-programming-prod
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```



```
hklement — hklement@openhpi-codelab: ~ — ⌘3

  1  [|||||||||||||100.0%]   11 [|||||||||||||100.0%]   21 [|||||||||||||100.0%]   31 [|||||||||||||100.0%]
  2  [|||||||||||||100.0%]   12 [|||||||||||||100.0%]   22 [|||||||||||||100.0%]   32 [|||||||||||||100.0%]
  3  [|||||||||||||100.0%]   13 [|||||||||||||100.0%]   23 [|||||||||||||100.0%]   33 [|||||||||||||100.0%]
  4  [|||||||||||||100.0%]   14 [|||||||||||||100.0%]   24 [|||||||||||||100.0%]   34 [|||||||||||||100.0%]
  5  [|||||||||||||100.0%]   15 [|||||||||||||100.0%]   25 [|||||||||||||100.0%]   35 [|||||||||||||100.0%]
  6  [|||||||||||||100.0%]   16 [|||||||||||||100.0%]   26 [|||||||||||||100.0%]   36 [|||||||||||||100.0%]
  7  [|||||||||||||100.0%]   17 [|||||||||||||100.0%]   27 [|||||||||||||100.0%]   37 [|||||||||||||100.0%]
  8  [|||||||||||||100.0%]   18 [|||||||||||||100.0%]   28 [|||||||||||||100.0%]   38 [|||||||||||||100.0%]
  9  [|||||||||||||100.0%]   19 [|||||||||||||100.0%]   29 [|||||||||||||100.0%]   39 [|||||||||||||100.0%]
  10 [|||||||||||||100.0%]   20 [|||||||||||||100.0%]   30 [|||||||||||||100.0%]   40 [|||||||||||||100.0%]
  Mem[||||||||||||||          8627/64382MB]   Tasks: 951, 362 thr; 42 running
  Swp[|                         12/65487MB]   Load average: 22.28 13.64 6.28
                                              Uptime: 42 days, 03:27:10

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
47166 root       20   0 28500  7360  2588 R 100.  0.0  0:18.25 ruby -e loop { 42 }
47455 root       20   0 28596  7316  2576 R 100.  0.0  0:14.33 ruby -e loop { 42 }
47525 root       20   0 28500  7316  2576 R 100.  0.0  0:13.47 ruby -e loop { 42 }
47342 root       20   0 28500  7320  2576 R 100.  0.0  0:15.79 ruby -e loop { 42 }
47276 root       20   0 28500  7316  2572 R 100.  0.0  0:16.62 ruby -e loop { 42 }
47640 root       20   0 28596  7312  2576 R 100.  0.0  0:11.97 ruby -e loop { 42 }
47431 root       20   0 28596  7308  2576 R 100.  0.0  0:14.56 ruby -e loop { 42 }
47298 root       20   0 28596  7308  2576 R 100.  0.0  0:16.35 ruby -e loop { 42 }
47099 root       20   0 28596  7348  2584 R 99.9  0.0  0:19.14 ruby -e loop { 42 }
47121 root       20   0 28500  7320  2572 R 99.9  0.0  0:18.80 native -console  -pipe 3 -root /var/lib/docker/exec
47077 root       20   0 28500  7364  2588 R 99.9  0.0  0:19.44 ruby -e loop { 42 }
47662 root       20   0 28596  7316  2572 R 99.9  0.0  0:11.67 ruby -e loop { 42 }
47817 root       20   0 28500  7308  2576 R 99.9  0.0  0:09.41 native -console  -pipe 3 -root /var/lib/docker/exec
47478 root       20   0 28596  7316  2576 R 99.9  0.0  0:14.06 native -console  -pipe 3 -root /var/lib/docker/exec
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

CodeOcean from within openHPI. After that, students perform an iterative development process. Each iteration comprises two requests between client and server. The first request sends the learner's code to the server for creating a code snapshot. The second request triggers either a code run or the execution of tests. The requests associated to a single simulated student are issued at intervals of five seconds, which mimic the student's thinking time.

C. *Test Results*

Figure 9 depicts the response times of requests for starting the programming session and sending code revisions to the server. An LTI launch request, which involves validating the request signature, redirecting the learner to the specified exercise, and rendering the development environment, takes about

Fig. 10. Response Time Graph Depicting Requests for LTI Launch and Snapshot Creation.

Fig. 11. Server Workload During the Load Test.

270ms on average. The simpler request for creating a code submission, which does not render a view but yields a JSON response, takes about 50ms on average. Short and hardly varying response times throughout the entire load test indicate that CodeOcean is able to handle the simulated load for the regarded requests without problems. Moreover, the application should be able to handle a higher number of concurrent users when given a proportionate amount of resources. Unfortunately, an entirely different picture emerges when taking code execution and assessment into account. Figure 10 depicts the response times of a subset of the requests regarded in Figure 9 as well as requests corresponding to code execution.

As the graph in Figure 10 shows, response times for code execution increase with the number of concurrent requests. In a massively parallel usage scenario, as simulated by the load test, response times rapidly reach a level at which students cannot be provided with feedback in a timely manner anymore.

While concurrent requests increase the response times for code execution to tens of seconds, the response times of all other requests remain at similar levels as depicted in Figure 10 However, due to the increased range of response times, these requests are barely perceptible in Figure 10. The fact that only Docker-related requests' response times escalate, while other requests are handled with ease, suggests that there is no general resource shortage but rather a problem concerning Docker. Figure 11 shows the server's workload during the load test as determined using the process viewer *htop*.

In fact, the image does not indicate resource shortage, but it shows that the server's available CPUs and memory are hardly used. Therefore, we believe that the poor scalability of code execution is attributable to problems with parallelizing server-to-server requests between the web application and Docker's API endpoint. The production server's many-core CPU should easily support running a sizable number of Docker containers in parallel. In order to eliminate the possibility of a general parallelization problem, we conducted an experiment investigating the parallelizability of concurrent Docker executions. In contrast to short-running processes, which are usual

Fig. 12. Server Workload During the *Parallelizability* Experiment.

for students' code submissions and which are represented in the load test, we regarded the behavior of long-running processes executed in Docker containers. In order to minimize the differences between the load test and the experiment, containers were started from within CodeOcean's web application. Figure 12 depicts the server's workload during the experiment. The image shows that all logical CPU cores are fully occupied by running the Docker containers. Hence, parallelization of concurrently running Docker containers is possible on the regarded infrastructure. Based on our observations, we deduce that Docker can provide sandboxed execution of multiple learners' code submissions in parallel. While this should theoretically provide the scalability needed for a large-scale usage of CodeOcean, scalable code execution was not achieved in practice. We suspect that the increasing response times for concurrent code execution requests are caused by a problem with parallelizing the allocation of new Docker containers. In order to enable the usage of CodeOcean in a MOOC, this issue had to be eliminated or evaded. To tackle the issue, we employed a pooling concept, allowing us to start containers upfront and assign them to users on demand. Whenever possible, new containers are started in the background to cover the relatively long start-up times. Having run all these tests made us look forward optimistically to the course start. The actual production usage in a Java programming course on openHPI, however, turned out to reveal several problems, which are beyond the scope of this paper and will be discussed in a follow up paper in detail. Particularly, the decision to use JRuby, turned out to be problematic.

VI. Future Work

As already mentioned, our next step will be a more detailed evaluation of our first programming courses that have been using CodeOcean. The focus here is on CodeOcean's performance. Another aspect that we are focusing on, is to further strengthen the security of the code execution environments. We are working on an extension to allow the upload of locally coded exercises, preferably directly from within an IDE such as Eclipse, Netbeans, IntelliJ, or–on a beginners' level–BlueJ. Furthermore, we are examining how code quality control tools, such as CheckStyle, FindBugs, PMD, or RuboCop might be employed to extend the assessment spectrum. Finally, we aim to allow tutoring sessions via direct video chats, in order to increase interaction between participants and the teaching team.

VII. Conclusion

Having previously examined the history and state of the art of automated code assessment tools, we now took the next step and designed and implemented a modern version of such a tool, particularly tailored for the usage in MOOCs, but not restricted to that use case. The tool has been successfully employed in two different MOOCs on the openHPI platform and thus has demonstrated its ability to provide MOOCs with hands-on programming exercises. After a bit of a rough start for CodeOcean during the Java programming course, we have by now fixed the majority of the initial problems. During the pilots we have proved the tool's versatility and consider it to be production ready. CodeOcean has been open sourced and everybody is invited to use the tool with their e-learning platform and to contribute to the improvement of the system.

References

[1] S. AlHumoud, H. S. Al-Khalifa, M. Al-Razgan, and A. Alfaries, "Using App Inventor and LEGO mindstorm NXT in a Summer Camp to attract High School Girls to Computing Fields," in *Global Engineering Education Conference (EDUCON), 2014 IEEE*, 2014, pp. 173–177.

[2] T. Staubitz, H. Klement, J. Renz, R. Teusner, and C. Meinel, "Towards Practical Programming Exercises and Automated Assessment in Massive Open Online Courses," in *Proceedings of IEEE TALE Conference*, Zhuhai, 2015.

[3] V. Pieterse, "Automated Assessment of Programming Assignments," in *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, 2013, pp. 45–56.

[4] L. Pappano, "The Year of the MOOC," *N. Y. Times*, 2012.

[5] L. Malmi, A. Korhonen, and R. Saikkonen, "Experiences in Automatic Assessment on Mass Courses and Issues for Designing Virtual Courses," *ACM SIGCSE Bull.*, vol. 34, no. 3, pp. 55–59, 2002.

[6] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A Study of the Difficulties of Novice Programmers," *ACM SIGCSE Bull.*, vol. 37, no. 3, pp. 14–18, 2005.

[7] A. Vihavainen, M. Luukkainen, and J. Kurhila, "Multi-faceted Support for MOOC in Programming," in *Proceedings of the 13th Annual Conference on Information Technology Education*, 2012, pp. 171–176.

[8] M. M. Ben-Ari, "MOOCs on Introductory Programming: A Travelogue," *ACM Inroads*, vol. 4, no. 2, pp. 58–61, 2013.

[9] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of Recent Systems for Automatic Assessment of Programming Assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, 2010, pp. 86–93.

[10] K. Ala-Mutka, "A Survey of Automated Assessment Approaches for Programming Assignments," *Comput. Sci. Educ.*, vol. 15, no. 2, pp. 83–102, 2005.

[11] N. Truong, P. Bancroft, and P. Roe, "Learning to Program Through the Web," *ACM SIGCSE Bull.*, vol. 37, no. 3, pp. 9–13, 2005.

[12] T. Staubitz, J. Renz, C. Willems, J. Jasper, and C. Meinel, "Lightweight Ad Hoc Assessment of Practical Programming Skills at Scale," in *Global Engineering Education Conference (EDUCON), 2014 IEEE*, 2014, pp. 475–483.

[13] M. Forišek, "Security of Programming Contest Systems," *Inform. Second. Sch. Evol. Perspect.*, 2006.

[14] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An Updated Performance Comparison of Virtual Machines and Linux Containers," IBM, Austin, TX, USA, 2014.

[15] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux J.*, vol. 2014, no. 239, 2014.

[16] C. Douce, D. Livingstone, and J. Orwell, "Automatic Test-Based Assessment of Programming: A Review," *J. Educ. Resour. Comput. JERIC*, vol. 5, no. 3, p. 4, 2005.

[17] A. Vihavainen, T. Vikberg, M. Luukkainen, and M. Pärtel, "Scaffolding Students' Learning using Test My Code," in *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, 2013, pp. 117–122.

[18] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated Feedback Generation for Introductory Programming Assignments," in *ACM SIGPLAN Notices*, 2013, vol. 48, pp. 15–26.

[19] C. Severance, T. Hanss, and J. Hardin, "IMS Learning Tools Interoperability: Enabling a Mash-up Approach to Teaching and Learning Tools," *Technol. Instr. Cogn. Learn.*, vol. 7, no. 3–4, pp. 245–262, 2010.

[20] M. von Löwis, T. Staubitz, R. Teusner, J. Renz, S. Tannert, and C. Meinel, "Scaling Youth Development Training in IT Using an xMOOC Platform," in *Frontiers in Education Conference (FIE), 2015 IEEE*, 2015.

[21] R. Odaira, J. G. Castanos, and H. Tomari, "Eliminating Global Interpreter Locks in Ruby through Hardware Transactional Memory," in *Proceedings of the 19th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2014, pp. 131–142.