ReplicatEdu: A Computer Science Course Platform

Alex Hortin
Georgia Tech OMSCS Student
USA
ahortin3@gatech.edu

ABSTRACT

In the following paper, I will discuss the motivations and creation of a platform called ReplicatEdu. This platform has been designed to offer a standardized low-cost way for instructors to run a computer science course. The platform has been designed with the primary motivation of bypassing lengthy configuration and setup steps usually required by other course platforms. In this paper I will discuss the motivations, the various components behind the platform, how the platform workflow would work, and some architecture details that outline the platforms capabilities.

Author Keywords
Docker; Github; knowledge management; automation; reproducibility

ACM Classification Keywords
knowledge management; automation; reproducibility

INTRODUCTION

Computer Science and related disciplines have often been at the forefront of educational technology. The instructors who prepare these courses often implement tooling to lower administrative overhead, make content more engaging, and provide a better overall experience for the student and facilitators of the course. There are several different learning platforms that have come out over the past few years to deliver lectures, provide student performance statistics, and allow for collaboration. However, few platforms have provided content creators the ability to create, share, and facilitate technical laboratory or project-based exercises in an easily testable and easily reproducible fashion.

Bringing the problem to a higher level, it is primarily about research project and lab reproducibility. The term *reproducible research* related to computer science was discussed with the intent that researchers should provide not only the academic paper, but also the data and computer code upon which the paper is based. This would allow readers to reach the same conclusions about the same data set [1]. Computer Science Lab and Project work should follow this type of model, as they are usually guiding students to derive a higher level understanding given the problem they are given or allow students to be evaluate on new insights they find into the problem.

Often to achieve these reproducible labs, students are expected to download a virtual machine specific to a lab or course to perform work, which needs to be upkept by course administrators to ensure the solutions are still working and dependencies are met to accomplish the project. Instructors have also taken the approach of providing students with lab environment setup instructions, which can atrophy over time and are difficult to maintain without offering heavy troubleshooting support. These environments do not often contain tools to provide student feedback or have automated testing upkeep features to ensure they still function correctly, and unless the instructor has built a custom solution like Bonnie at GA Tech, a framework for structuring this type of feedback is not readily available outside of commercial offerings like Codio. In this regard, Codio is also limited because it does not let a student easily iterate and capture their environment to share their modified environment back to the instructor.

This is a problem for instructors because it limits their ability to quickly iterate on course material and a problem for students, as troubleshooting lab and project environment setup can often detract from the material the student is trying to learn and forces stagnation of lab environments. I propose the development of a platform used by instructors and students to plan, build, test, execute, share, and iterate on reproducible project and lab materials. As part of this, I will also propose a working open schema for how those materials will be structured. A portion of this platform would then be used by students to directly execute the lab using the canonical environment built and tested by the instructor or could even be used by the student to develop reproducible research projects to be evaluated by the instructor.

MOTIVATIONS

On-campus students as well as online students may lack a common platform to accomplish their projects that require specialized environments. Universities often have labs set up that can offer these environments, but these can also be limited since the platform they are built on is often shared between many courses with varying dependencies. Often, virtual machines or setup instructions are the most common solution. This is not ideal because these virtual machines are often heavyweight and classes require at least one each, if not more.

Lowering the administrative burden of teaching and participating in a project and lab-driven course could be approached by augmenting current tools with an open project and lab environment platform. Such a framework could be heavily built around two best practices borrowed from the software engineering industry to make maintaining the labs, projects, and environments required to accomplish them easy to set up and iterate on. These two overarching best practices would be environment reproducibility using "infrastructure as code" and evaluation through the use of test-driven development tools for continuous integration of the labs and projects against that environment.

Infrastructure as Code (IaC) allows the efficient deployment of a complicated environment using automation scripts derived from flat file infrastructure definitions [2]. Using a platform that required each project or lab to have a standardized environment configuration as part of the lab would allow instructors to develop a canonical environment for students to perform the project or lab. When paired with modern containerization tools like Docker, this becomes much less of an administrative burden then maintaining instructions or a virtual machine. In addition, project environments could be updated easily via cluster management solutions built into some of the platforms. Reproducibility also allows others to build upon existing work and use it to test new ideas and develop methods [3]. Expansions or revisions on projects is much easier to complete without hand tailoring an environment from scratch every time a new dependency is needed or software update is required.

Continuous integration would allow for the tested verification of these environments as instructors create or build upon a project or lab. A continuous integration framework provides automated source repository change detection. When changes to the repository are detected the full environment is built [4] and environment tests execute to verify the environment is still suitable to accomplish the project. Using this methodology, thorough tests serve two purposes: the evaluation of student work and the verification of the lab environment. When evaluated for a grade, student code could be pulled from version control into a pristine environment that mirrored the development environment they built the project with. This portion of the testing could also perform plagiarism detection, provide instant student feedback, and allow instructors to watch the class progress with the problem set in real time against the tests.

RELATED RESEARCH

Over the course of this project, I did extensive research into other platforms and research that attempt to solve similar problems. The platforms mentioned below include Codecademy, Coderbyte, and Codio. A few of the main issues that supported my problem are described below.

- Many commercial products are not open source, and rapidly escalating cost of proprietary software leaves too little of an institution's budget available for creative exploration [5].
- Solutions often require coursework to be created in various proprietary formats.
- Current environments can place a heavy cognitive load put on programming students unrelated to the course [6].
- Platforms lack an open standardized ability to grade projects in an easy format so tests can be easily structured and modified that avoid common problems associated with online grading of computer science problems, namely: plagiarism and queue hogging [7].
- Complete coursework solutions (like Codio) require a large investment in hardware and support.
- Most solutions require a constant internet connection which is not ideal for developing nations.
- A lack of solutions are available that support non-trivial architectures (multi-host, clustering, hardware emulation, security exercises, etc) that can be easily reproduced without extensive student configuration.

This research led me to conclude that no secure solution for hosting a course platform that met these goals currently exists. Docker's ecosystem provided a reasonably secure environment to try and accomplish some of these goals [8]. I proposed building a platform using container-based technology as the backbone of a product that would mitigate these problems.

DEVELOPMENT TRACK

For this project I chose to develop a platform and undertook this in the development track. During the proposal, I outlined a project with several technical requirements. I have been in many courses that have issues that have influenced this platforms design. In addition, new technologies like Docker allowed me to approach this problem from a new standpoint. I chose Rust as the language to write all of the tools in because it offered excellent testing, secure coding practices, and produces easily distributable and extremely fast binaries.

TECHNICAL REQUIREMENTS

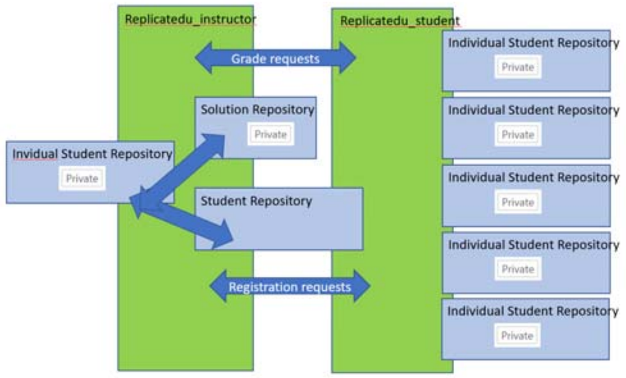
An infrastructure and course management console program was created to perform various upkeep tasks that will manage the overall infrastructure and build the reproducible images. This was be the bulk of the development portion and the tasks it will handle are described below:

- *Class setup and registration.* The platform handles both the instructor setup and student registration portion via Github issues as the stored database.
- *Building and testing of Project and Lab environments during development.* Reading the specification format and testing to ensure the project is ready for deployment. This is handled via a skeleton code parser and via the infrastructure modules which automatically builds the workflow.
- *Creation and publishing of canonical images related to each project for use by students.* These will be made available for each student participating in the course.
- *Evaluating student code.* Evaluations can be done with a mix of stdin/stdout, behavior heuristics, or file io validation.
- *State management.* Creation, storage, and retrieval of a student grade repository to persistent data stores in Github.
- *Management of a distributed test cluster.* Individuals, including students running the ReplicatEdu platform can opt in to have their local computational resources utilized to test other students' code. This could be incentivized by offering extra credit. This could potentially pose a cheating concern, so care will be taken to heavily isolate the test cluster from the underlying platform. In addition, professors can opt to not use this and locally run all grading tests or deploy a testing cluster to cloud resources on demand.
- *Environment access.* Handle student access to the lab environments they are working on.
- *Storage management.* Deconflict and manage local persistent student data.

The instructors will develop laboratories or projects based on a lightweight schema and folder structure. This structure will have the following elements:

- *Solution code.* Instructor solution to the problem with metadata tags to indicate code that should be removed if provided as skeleton code.
- *Private tests.* Test code that will not be provided to the students in local tests.
- *Public tests.* Local test cases the students can download and execute with the curator.
- *Infrastructure setup.* Dockerfile metadata with instructions for the curator on how to build the laboratory environment.

These components will allow for the platform to create, test, and deploy for student use and grade these projects or labs. It will automate the creation of the environments that students utilize and offer the ability to rapidly change or iterate projects between offerings to revise learning objectives.



Replicatedu_instructor
Replicatedu_student
Grade requests
Solution Repository
Private
Invidual Student Repository
Private
Student Repository
Registration requests
Individual Student Repository
Private
Individual Student Repository
Private
Individual Student Repository
Private
Individual Student Repository
Private
Individual Student Repository
Private

Figure 1 – *This is a high-level outline of the Github workflow.*

Github was used for storing persistent data related to a course and project accomplishments (see Figure 1).

- *Student code.* All student code will be kept in private repositories on student accounts. Instructors will have read-only credentials to these to pull and evaluate code. In addition, modifications to the lab environment can be captured here as well with a Dockerfile.
- *Student grades.* Each student will be granted read-only access to a student repository that will contain their grade. Student Grades pages will be available to query by the student for their performance.
- *Student registrations.* Students will register with a private key given by the instructor.
- *Laboratory/project specification.* Private Github repositories will be used to store this data accessible to the instructors.

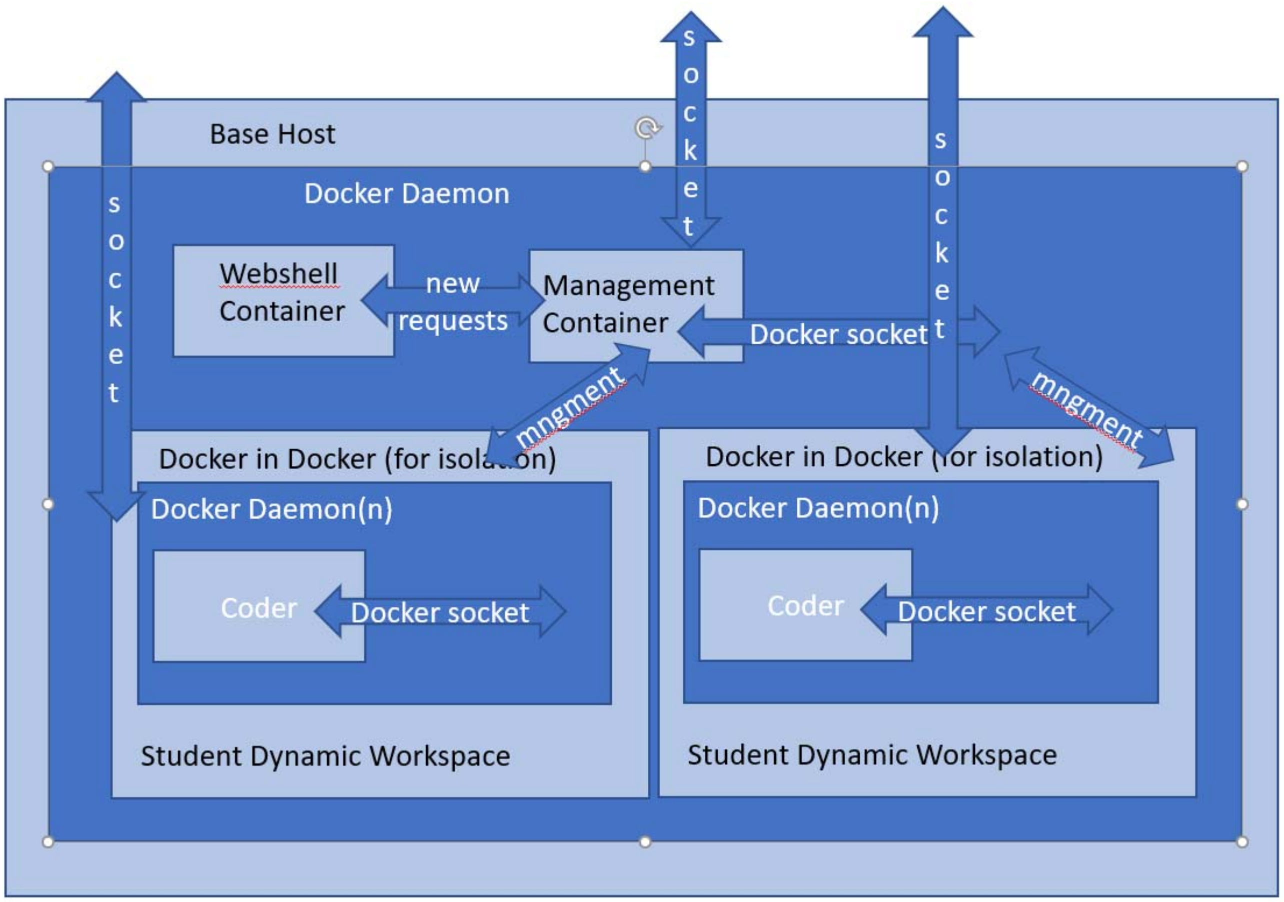


Figure 2 – *This is a high-level outline of the infrastructure model that runs the demonstration. Isolation is achieved by using isolated Docker daemons.*

ARCHITECTURE

The module architecture is outlined here to show how each module in the Github repo integrates into the final platform.

- *Demo_wrapper*. This is a wrapper that build and integrates the demonstration test server used for the final project submission.
- *Replicatededu_instructor*. The program that the instructor uses to build and test the courses, run the grading and registration daemons and create the git repos for the course iteration.
- *Replicated_student*. This is the program for a student to manage their repositories, request grades, and register for classes.
- *Test_runner*. This is a program that will run in a directory and output test scores based on the replicated.manifest defined tests.
- *test_class*. An example class that shows how to build one of these classes for the platform
- *Issue_database*. A wrapper for the commands the use github issues as a backing datastore primarily registration and grades.
- *Management_server*. This is the server that powers the dynamic demo environment that lets users dynamically spin up and shut down servers. It also offers a VS Code based ide that is hosted within the platform and provided to the student via web browser (see Figure 2).
- *Docker_wrapper*. A wrapper for all docker commands used in this project.
- *Class_crypto*. A custom crypto library used to secure student grades and registrations.
- *Git_wrapper*. A custom git library used to post repos on github.
- *Skeleton_parser*. This is a library to generate skeleton code for assignments using markup embedded in the assignment**.**

RESULTS

I have concluded building a platform that meets all of these goals. The three primary applications that were created for this project; *replicatedu_instructor*, *replicatedu_student*, and *test_runner* are built using the other supporting modules all contained within the replicatedu Github account.

The general course workflow would work as follows. Instructor designs and tests a course using test runner in one single consolidated repository. This repository will include a Dockerfile that will show the platform how to meet all software dependencies and build requirements. The platform also has the potential to support multiple machines per work instance, but this is not exclusively supported yet.

When ready to instruct, they parse the class using the provided tools which creates a solution repository and a student repository. The student repository will be what each student clones from and will also have all of the registration and grading metadata captured in issues. Class updates can also be posted there and students can pull from this repo. The instructor distributes the course coordination secret key from the *coord_keys* in solution and the student repo.

The students can now use this to register for the course. The registration provides each student's public keys to the instructors and registers them. The instructor can run a management daemon at any time to process any outstanding registration requests. When students register, the instructor's public SSH keys are automatically added to the new personal student repo so the instructor can pull and grade assignments.

When grading needs to be achieved, the student will submit their assignment and the instructor will now download and execute their code within an isolated Docker environment. Points are awarded via a custom test format. Grading servers can be started easily in any system with docker and access to the instructor's private solution repository and corresponding SSH private keys. In addition, students can test their code anytime against any tests in the student repository that were given with the course.

A guided tour of using the platform to accomplish these can be found in the following repository and deployed to any server or local machine for student usage. https://github.com/replicatedu/demo_wrapper/blob/master/walkthrough.md

Beyond the scope of my initial project proposal, I created a working hosted version that can run multiple isolated instances of the platform to support concurrent users on a shared resource. Instructors and students can build and deploy a full working version of the project using the instructions and platform deploy scripts contained here: https://github.com/replicatedu/demo_wrapper.

Finally, the biggest result is that I utilized many features of the platform to develop it, trying to bootstrap early on and see if it was a viable environment to develop non-trivial work. I found it very productive and will be using it for future projects and pieces of it in an upcoming class I am about to teach to several incoming interns at work.

FURTHER RESEARCH AND WORK

While planning building this platform, I have come up with several features that I would like to explore if this platform undergoes continued development. The biggest feature that would be useful is integrations into other services that would make instructor and student work flow even easier.

One of the features originally planned for was the integration of code submission in plagiarism detection. The platform is setup so this would be easy to accomplish now after some more platform polishing. This would allow an automated detection of student code against platforms like Stanford's MOSS system.

There are also opportunities to integrate into other services to improve the workflow. For example, students could receive notifications via slack that tell them when their registration is confirmed or assignments are graded. Introspection into the queue status could also be done.

Another opportunity would be allowing anyone to run the platform to share the burden and run automated grading servers. This would allow the instructors to use student resources to help run and evaluate student code. There are several approaches to make this viable in a secure way, but it would also allow a student to potentially reverse engineer the platform, gather other students code, and use it.

Container-based security is still lacking compared to hypervisor-based virtualization, but mitigations can be taken to help bring them closer to the same level. I tried to use several recommendations to help secure those containers [9] in this project. This platform tries to mitigate those via layered docker sockets but goes against a few security paradigms that could lead to platform vulnerabilities. These could be further mitigated through a properly configured project virtual machine, but this is outside the scope of this project.

The platform as designed has some serious issues right now that would need to be remediated prior to mass adoption. There are currently several edge cases when managing the classrooms state that are unhandled. This could lead to the loss of submission or grading data but should be trivial to fix.

The tools are not structured currently to offer robust error handling. Part of the reason I wrote the project in Rust is it checks to ensure errors are handled at compile time. In the interest of getting this project completed by the deadline, I used a few shortcuts to simplify the error handling model, but those are done in such a way it is obvious I need to go back and fix them.

In this same vein I would refactor each of the tools to support better presentation of the errors to each student. The tools should be commented and documented more thoroughly to show the commands that are possible.

The class state is also dependent on Github issues. In the future it may be worth looking at other solutions to alleviate this storage necessity. There are several distributed hash tables that could be used to remove the Github dependencies.

Finally, the server that provides the on-demand instances is not currently set to hold much traffic. It would need to be robust against denial of service attacks since the platform is currently not hardened against them. There is also a limitation if multiple people are trying to log on at once.

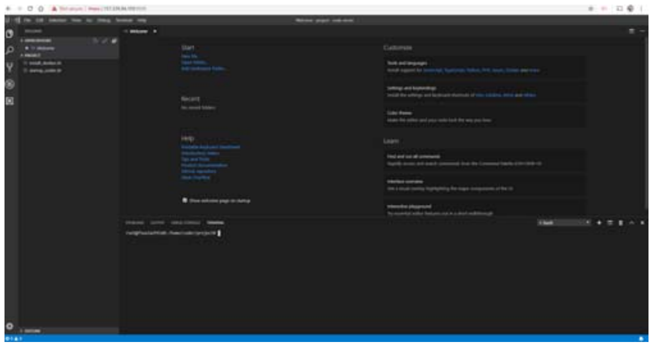


Figure 3 – *Screenshot of the platform running the online IDE.*

CONCLUSION

This platform can be used in its current state for testing this workflow. Certain portions will be used when I help instruct a class later this year and I will revise this paper with those results.

Using git for assignment submission and planning allows for student exposure to industry standard workflow, and git integration into other services like Travis CI allows for the easy and automated evaluation of code [10]. This workflow made it easy for me to build this platform and should apply well to most class workflow.

Bootstrapping early on and trying to automate the standup of this platform led me to various design decisions that I might not have found had I been developing in another workflow. I would highly recommend that anyone developing an online development ecosystem get the minimum viable portions together and start "eating your own dog food" as soon as possible to help find problematic areas and remediate them before you have actual users working on a platform (see Figure 3).

Overall, I think that a platform following this model will assist instructors and students as programming assignments continue to grow in complexity. This platform automates many of the traditional pain points and offers solutions that are completely open source and cheap to run.

ACKNOWLEDGMENTS

I would like to thank all the instructors and peers who gave feedback on my work, and I would like to thank the open source community as a whole for offering and supporting such great tools, enabling one student to build – in a little over 125 hours – a platform that several years ago would have required much more resources.

REFERENCES

1. Buckheit, J. B., & Donoho, D. L. (1995). Wavelab and reproducible research. In *Wavelets and statistics* (pp. 55-81). Springer, New York, NY.
2. Hummer, W., Rosenberg, F., Oliveira, F., & Eilam, T. (2013, December). Testing idempotence for infrastructure as code. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing* (pp. 368-388). Springer, Berlin, Heidelberg.
3. Ram, K. (2013). Git can facilitate greater reproducibility and increased transparency in science. *Source code for biology and medicine*, *8*(1), 7. Retrieved From

   https://scfbm.biomedcentral.com/articles/10.1186/1751-0473-8-7
4. Stolberg, S. (2009, August). Enabling agile testing through continuous integration. In *2009 Agile Conference* (pp. 369-374). IEEE.
5. Jhurree, V. (2005). Technology integration in education in developing countries: Guidelines to policy makers. *International Education Journal*, *6*(4), 467-483.
6. Pun, T. (2018, Summer) Creating Project Based programming tutorials from GitHub Repositories. Georgia Institute of Technology. Retrieved from

   https://github.gatech.edu/pages/kbrunson6/edtech_projects/#/project/ONFJRncO0D
7. Cheang, B., Kurnia, A., Lim, A., & Oon, W. C. (2003). On automated grading of programming assignments in an academic institution. *Computers & Education*, *41*(2), 121-131.
8. Martin, A., Raponi, S., Combe, T., & Di Pietro, R. (2018). Docker ecosystem–vulnerability analysis. *Computer Communications*, *122*, 30-43.
9. Bui, T. (2015). Analysis of Docker Security. arXiv preprint arXiv:1501.02967.
10. Kelleher, J. (2014, January). Employing Git in the Classroom. In Computer Applications and Information Systems (WCCAIS), 2014 World Congress on (pp. 1-4). IEEE.