Standardizing Students' Programming Environments with Docker Containers

Using Visual Studio Code in the Cloud with GitHub Codespaces

David J. Malan
Harvard University
malan@harvard.edu

CCS CONCEPTS

• **Social and professional topics** → **Information technology education**; **Computer science education**; **CS1**; • **Information systems** → **Collaborative and social computing systems and tools**.

KEYWORDS

code, code editor, container, containerization, Docker, editor, integrated development environment, IDE, programming, text editor

ACM Reference Format:
David J. Malan. 2022. Standardizing Students' Programming Environments with Docker Containers: Using Visual Studio Code in the Cloud with GitHub Codespaces. In *Proceedings of the 2022 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '22), July 8–13, 2022, Dublin, Ireland*. ACM, New York, NY, USA, 2 pages.

1 INTRODUCTION

For CS50 at Harvard University, we have long sought to provide students with a standardized programming environment at term's start so that students can dive right into programming in C (followed by Python), without having to install any software themselves or wrestle with any technical difficulties beyond those in their own code. Toward that end, we have used, over the years, the university's own on-campus cluster of Linux systems, our own re-creation thereof in the cloud, a client-side virtual machine (VM), and a browser-based integrated development environment (IDE). In the first of those models, we found ourselves pedagogically constrained by older packages of software and lack of root access. And in all subsequent models, we found ourselves all too technologically distracted from teaching itself, having become our own system administrators.

In 2021, though, GitHub launched Codespaces [2], a cloud-based version of Visual Studio Code (VS Code) backed by Docker containers (lighter-weight alternatives, in effect, to virtual machines). Using Codespaces, we realized, not only could we provide students with a standardized programming environment at term's start, we could also transition them at term's end to a nearly identical client-side installation thereof (so as to continue programming without the course's infrastructure thereafter), without having to administer the former ourselves. We thus developed a lightweight front end to Codespaces for students, a web app that automates, using the Codespaces API [1], the process of creating, within the course's own GitHub "organization," one "codespace" (i.e., container) per student, based on our own Docker image, using a `.devcontainer.json` file [3], and redirecting them thereto, per Figure 1, with the containers themselves hosted by GitHub, per Figure 2. We deploy mid-semester updates to that same image via a VS Code extension, which prompts students to rebuild their container, preserving their own files therein. By way of VS Code's Remote Development extension pack [4], students can even run VS Code locally but still connect via SSH to their container in the cloud. And, if comfortable installing Docker locally, they can run their own container as well, completely offline. Via a Visual Studio Live Share extension can students share their codespaces with the course's teaching fellows (TFs), to help them troubleshoot bugs in real time, or even with classmates, if collaborating on a project, as via pair programming.

Advantages. Not only do containers provide students with a standardized environment, reducing technical difficulties and frequently asked questions at term's start, they also provide instructors with full control over the software in use and versions thereof, additionally allowing instructors to deploy updates mid-semester. Particularly for large courses with hundreds or even thousands of students, containers allow staff to focus more of their time on teaching than on technical support. And, coupled with text editors that support extensions or plugins, containers allow instructors to optimize students' environment for learning, while still acquainting students with industry-standard tools.

Disadvantages. Among the few downsides to date is that VS Code's current API for extensions is less featureful than would be ideal pedagogically, and we have not been able to simplify VS Code's UI to the extent that we would like. We would prefer to hide icons and buttons that we do not expect students will use (yet), lest they distract early on.

In this session, we present our experience with Codespaces, along with a live demonstration thereof, and argue in favor of precisely this model: browser-based programming environments, backed by Docker containers, as alternatives to clusters of servers or virtual machines for students in introductory programming courses. Our own adaptation of Codespaces has already been used by thousands of students, both on campus and off, and is freely available to fellow teachers and students beyond.

ITiCSE '22, July 8–13, 2022, Dublin, Ireland

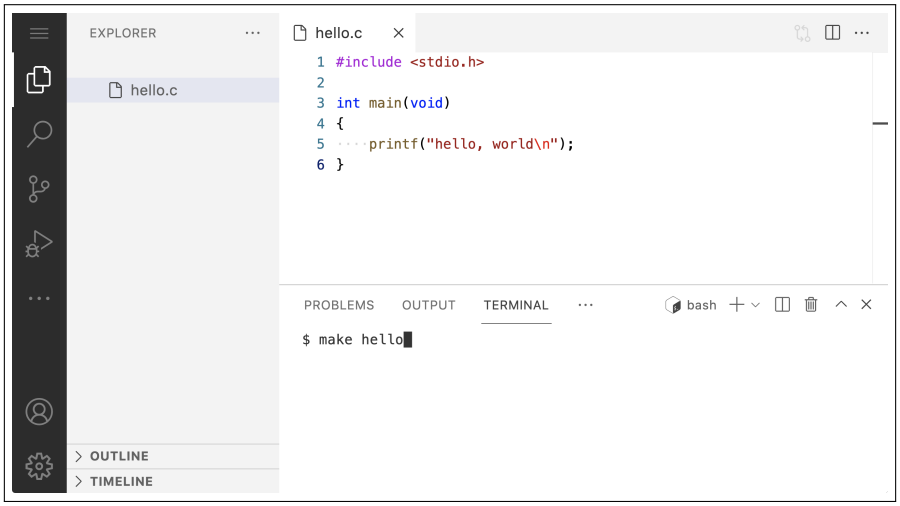

Figure 1: CS50's adaptation of Codespaces provides students with a web-based version of Visual Studio Code, connected to a Docker container running the course's own image, its default UI simplified and enhanced for teaching and learning via the course's own extensions.

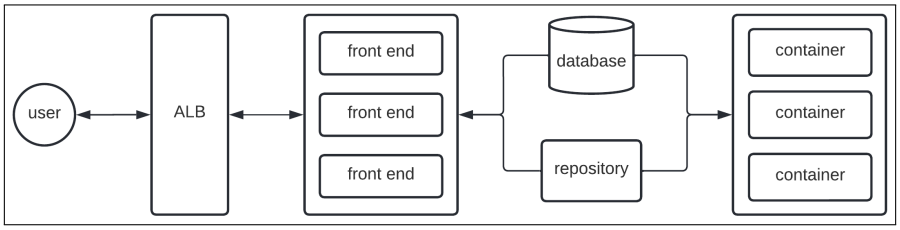

Figure 2: CS50's adaptation of Codespaces. Users are routed via an application load balancer (ALB) to a web-based front end. A database stores metadata like the IDs of students' codespaces, while a GitHub repository stores backups of their codespaces' files. GitHub hosts VS Code itself and provides users with codespaces, containers based on the course's own image.

2 RESULTS

Since its debut in 2021, CS50's adaptation of Codespaces has been used by more than 80,000 users so far. Not only has VS Code atop Codespaces supported our pedagogical goals of providing students with a standardized environment for C and Python at term's start, it has also eliminated the need for system administration on our end. Via our own `Dockerfile` and `.devcontainer.js` file can we still customize students' containers by pre-installing packages and extensions. Not only has this solution allowed us to focus more time on students, without nearly as much time spent on system administration, it has also enabled us to provide students with an experience that begins in the cloud but ends on their own PC or Mac.

We argue, ultimately, that teachers elsewhere should consider containerization as a compelling alternative to any cluster- or VM-based environments, at least for introductory courses. Courses requiring specialized architectures might still benefit from other solutions. But just as containers have commoditized how applications can be packaged and deployed for production in industry, so might containers standardize more easily than ever programming environments for students.

APPENDIX

The course's adaptation of Codespaces is freely available for fellow teachers and students at https://code.cs50.io/. And the `Dockerfile` and `.devcontainer.json` files that define students' containers are at https://github.com/cs50/codespace.

ACKNOWLEDGEMENTS

Many thanks to Brenda Anderson, Brian Yu, Carter Zenke, Doug Lloyd, Glenn Holloway, Kareem Zidane, and Rongxin Liu for their assistance with this work. And many thanks to Amazon Web Services, GitHub, and Microsoft for their support of this work.

At the time of writing, this work's author is consulting part-time for GitHub as a Professor in Residence.

REFERENCES

[1] Codespaces API. 2022. https://docs.github.com/en/rest/reference/codespaces
[2] GitHub Codespaces. 2022. https://github.com/features/codespaces
[3] devcontainer.json reference. 2022. https://code.visualstudio.com/docs/remote/devcontainerjson-reference
[4] VS Code Remote Development. 2022. https://code.visualstudio.com/docs/remote/remote-overview