Containerizing CS1 with Docker

Standardizing Students' Programming Environments with Visual Studio Code atop GitHub Codespaces

Author
Institution
City, State, Country
Email

ABSTRACT

We argue in favor of Docker containers as alternatives to clusters of servers or virtual machines for students in introductory programming courses. We present our experience with the same in CS1 at our university since 2015, as well as the pedagogical and operational motivations behind our selection. We present, too, the evolution of our environments for students over the years, from an on-campus cluster, to an off-campus cloud, to client-side virtual machines, to Docker containers, discussing the trade-offs of each. 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. Most recently implemented atop GitHub Codespaces, a cloud-based version of Visual Studio Code, our own container-based solutions have since been used by hundreds of thousands of students, both on campus and off, and are freely available to all teachers and students.

CCS CONCEPTS

• Social and professional topics → CS1; **• Applied computing → Computer-assisted instruction**.

KEYWORDS

code, code editor, container, containerization, Docker, editor, graphical user interface, GUI, integrated development environment, IDE, Kubernetes, programming, text editor, web app, web application

ACM Reference Format:

Author. 2024. Containerizing CS1 with Docker: Standardizing Students' Programming Environments with Visual Studio Code atop GitHub Codespaces. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE 2024), March 20–23, 2024, Portland, Oregon.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

CS1 is our university's introductory course in programming for majors and non-majors alike, taught primarily in C toward its start and primarily in Python toward its end. The course is among our university's largest, with approximately 600 students each fall, 100 students each spring, and 30 students in summer. Two thirds of the students have never taken a course in computer science before, and one half describe themselves as being among "those less comfortable" with computing. The course itself is intended to be accessible to students of all backgrounds, with or without prior programming experience, with different tracks within the course for those less comfortable and more comfortable alike, whereby students can attend different sections (i.e., recitations) and implement (somewhat) different programs each week based on their comfort level [2].

With so many students (and, in turn, so many versions of Windows, macOS, and Linux), it would be a challenge to have everyone install and configure the requisite software on their own for the course's assignments, including `clang`, `gdb`, `valgrind`, and `python3`. Even Python's installer, which is perhaps easiest, doesn't necessarily update students' `$PATH` accordingly. Even in smaller classes, start-of-term setup has not proved, in our experience, the best use of time. And, at worst, the least comfortable of students might conclude that programming simply isn't for them if they can't even get their computer set up.

We have thus long provided students with a standardized programming environment, so as to focus immediately at term's start on concepts and implementation thereof rather than technical difficulties. We first provided such using the university's own on-campus cluster of Linux servers on which students had shell accounts and NFS-mounted home directories, accessible via SSH. Without root access, though, we found it difficult to configure their accounts exactly as we wanted, and so we transitioned to a similar topology of our own in the cloud [3]. Managing that cluster ourselves proved more time-consuming than intended, and so we eventually migrated to client-side virtual machines (VMs), a la Stoker et al. [41] and Harvie et al. [27], with each student running their own Linux "appliance" [4]. While VMs did standardize students' environment, they tended to be slow, especially on lower-end laptops. And so we returned to the cloud, this time using Docker [19] containers, much like Valstar et al. [46] but with an addition of browser-based graphical user interfaces (GUIs), initially via AWS Cloud9 [12] and, most recently, Visual Studio Code [13] atop GitHub Codespaces [14]. Not only did containers enable us to standardize programming environments, both server-side and client-side, they proved far lighter-weight, faster for staff to develop

SIGCSE 2024, March 20–23, 2024, Portland, Oregon

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and for students to run. Coupled with web-based GUIs (complete with code editors, file explorers, and terminal windows), they have enabled us to provide students with full-featured programming environments in the cloud. And students can even install the same client-side at term's end, thereby continuing to write code locally even without the course's own infrastructure, and without having had to figure out how right at term's start.

In this work, we present our path to containers and web-based programming environments atop them, including the pedagogical and technological motivations therefor. In Section 2, we elaborate on the advantages and disadvantages of our prior approaches. In Section 3, we explain our motivation for Docker. In Section 4, we discuss implementation details and trade-offs among them. In Section 5, we present our results and next steps. In Section 6, we conclude.

2 PRIOR APPROACHES

Whether on-campus or off-campus, server-side or client-side, CS1 has long sought to standardize students' programming environments to minimize technical difficulties at term's start. We present in this section our prior approaches to motivate our recommended approach in Section 3 so that others need not reinvent wheels.

Were CS1 entirely focused on Python, or even a language like JavaScript, students could write and execute code with a browser alone [9, 34, 35, 40, 44], without any server-side infrastructure (beyond a static website). But we have not found browser-based compilers or interpreters for C to be nearly as robust, although technologies like WebAssembly [48] with Emscripten [22] are promising. And, pedagogically, we indeed prefer to start with C and end with Python, so that students ultimately understand the abstractions that higher-level languages provide. We'd ideally like students to acquire experience with a full-fledged Linux command line as well, though browser-based emulation of Linux tends to be slow and lacking in support for libraries and package management [31, 49]. In-browser solutions would also complicate CS1 students' development of web apps in Python with Flask [24] at course's end, as browsers' security models do not allow in-browser code to create and bind to TCP/IP sockets, preventing students from serving their app during development and testing.

We have thus found that the course's pedagogical aims are served best by providing students with a full-fledged Linux environment in some form, even in 2023.

2.1 On-Campus Cluster

As of 2007, students in CS1 (and other courses at our university) all had shell accounts and NFS-mounted home directories on an on-campus cluster of Linux servers to which they could connect via SSH, managed by the university itself. We ourselves did not have root access and were thus reliant on the university's own staff for any installations and updates, which were slow to happen, if only because others might be fixated on other versions.

Starting in 2007, then, we began to install our own versions of software within the course's own home directory, directing students to run a single command at term's start (e.g., `~cs1/setup.sh`), which would modify their `$PATH` accordingly by editing one of their dotfiles. Of course, we still didn't have root access, which meant installing all software locally from source, rather than more conveniently via package managers. And the servers invariably experienced technical difficulties outside of the university's business hours, which meant delays for fixes, if not extensions for students, even if we ourselves were awake and willing to fix.

2.2 Off-Campus Cloud

In 2008, we thus moved CS1 into the cloud, recreating the on-campus cluster virtually with Amazon Web Services (AWS) [39], using Elastic Compute Cloud (EC2) [21] for virtual machines (VMs) and Elastic Block Store (EBS) [20] for students' home directories [3], with Simple Storage Service (S3) [38] for backups. On this virtual Linux cluster, students had their own CS1-specific shell accounts, managed by the course's own LDAP server, also implemented within a VM. And we installed all software globally, so students' accounts automatically worked as intended.

Via educational grants from AWS [25] (available by application to all educators), this solution was free. But we underappreciated the time involved in administering our own cluster. And, by 2011, we daresay the novelty had worn off. By that time, too, the course had a growing OpenCourseWare audience, with the course's videos, slides, assignments, and more freely available to the public at large. But, with shell accounts restricted to students at our university, "taking" the course via OpenCourseWare was a fairly passive experience, as only the most comfortable of students online were inclined to figure out how to install `clang`, `gdb`, `valgrind`, and the like locally in order to engage actively with the course's assignments and final capstone project. We thus began to develop a client-side alternative to the course's virtual cluster that anyone on the internet could download and use, including our on-campus students.

2.3 Client-Side Virtual Machines

In 2011, we phased out the course's cloud-based cluster, instead providing students with a downloadable VM, an "CS1 Appliance," not unlike Griffin et al. [26] and Laadan et al. [33], preconfigured with all of the course's software that students could run locally on their PC or Mac simply by installing a hypervisor like VirtualBox [47]. The appliance enabled students to run Linux within a window on their own computer (initially Fedora, subsequently Ubuntu) and even a desktop environment (Xfce, which enabled graphical assignments), with `clang`, `gdb`, `valgrind`, and more already installed.

Via open-source software, this solution was free. But development of this appliance was incredibly time-consuming. Even with the build process automated via Kickstart [43] initially and our own packages subsequently, each update thereto might take us hours to export and test. The appliance's disk image, meanwhile, was nearly 2 GB (even with unneeded packages pruned) and slow for students to download, especially off campus. While the appliance's post-installation performance was fine on most students' laptops, lower-end netbooks (at the time) struggled under its weight. And the appliance was slow on most laptops to boot up.

Windows updates at the time, too, had a tendency to break VMs' virtual network adapters, causing headaches for web programming. Worst, though, at the time were bugs in VirtualBox itself: at one point, closing the lid of one's laptop with the VM still running could “brick” it entirely. We mitigated some disasters by encouraging students to back up their work, as via Dropbox within the appliance, but downloading a new image to replace a bricked one was still a slow process. (And Dropbox eventually deprecated their Linux client.) The course’s teaching staff therefore too often found themselves troubleshooting virtual disk images and virtual network adapters in the middle of office hours, when we preferred to focus on helping students with actual code. Those difficulties, and frequently asked questions, were only magnified at scale via OpenCourseWare. We eventually transitioned to VMware Player (for Windows) and VMware Fusion (for macOS) for on-campus students, both of which proved more stable than VirtualBox at the time, but the latter was not always available for free to the OpenCourseWare audience too.

Despite these shortcomings, the appliance proved necessary in 2012, when the course became even more broadly available as a massive open online course (MOOC) via edX [5]. Without a client-side solution already in place, the course could not have scaled during the heyday of MOOCs to so many thousands of learners all of a sudden.

<table>
<tr><td>Virtual Machine
Guest OS</td><td>Virtual Machine
Guest OS</td><td>Virtual Machine
Guest OS</td></tr>
<tr><td colspan="3">Hypervisor</td></tr>
<tr><td colspan="3">Host OS</td></tr>
<tr><td colspan="3">Bare Metal</td></tr>
</table>

Figure 1: Virtual machines (VMs) tend to be heavier-weight than containers, in part because each VM runs an entire operating system (OS), a guest OS that runs atop a (type-2) hypervisor, which runs atop a host OS, which runs atop bare metal. Whereas a server might have ample resources only to run, e.g., 3 VMs, that same server might have ample resources to run, e.g., 14 containers, as in Figure 2.

<table>
<tr><td>Container</td><td>Container</td><td>Container</td><td>Container</td><td>Container</td><td>Container</td><td>Container</td><td>Container</td><td>Container</td><td>Container</td><td>Container</td><td>Container</td><td>Container</td><td>Container</td></tr>
<tr><td colspan="14">Docker</td></tr>
<tr><td colspan="14">Host OS</td></tr>
<tr><td colspan="14">Bare Metal</td></tr>
</table>

Figure 2: Containers tend to be lighter-weight than virtual machines (VMs), in part because containers share OS kernels via Docker, which runs atop a host OS, which runs atop bare metal (or even a virtual machine). Whereas a server might have ample resources to run, e.g., 14 containers, that same server might only have ample resources to run, e.g., 3 VMs, as in Figure 1.

3 TOWARD CONTAINERIZATION

By 2015, we were eager to transition away from the client-side appliance, even though it had proved precisely the solution we needed up until then. By that time, too, a web-based alternative seemed an obvious direction, but most of the in-browser programming environments available struck us as oversimplified pedagogically, with file systems flattened and terminal commands like `clang` abstracted away entirely as buttons. User-friendly, perhaps, but we were reluctant to provide students with a “toy” environment, lest they not know how or where to write code after term’s end.

Any web-based alternative, we felt, should still have a terminal window, with the browser providing students not only with a GUI but a command-line interface (CLI) to Linux as well. We could not imagine backing each terminal window with a server-side VM, though, as via EC2. With hundreds of students on campus and thousands of students online, the cost of a VM-per-student model would likely exceed any educational grants. Even so, we preferred not to revert to a multiuser model like our on-campus cluster or off-campus cloud, as it had proved helpful for every student to have root access via `sudo` to their appliance, so that they could install packages at will, particularly for end-of-course capstone projects. And we appreciated that VMs sandboxed potentially malicious (or buggy) code more so than one multiuser system alone.

For an alternative back end, we thus turned to Docker, which implements OS-level virtualization instead, via which multiple “containers” can run in parallel on a host, all of them sharing the same OS kernel. Whereas a virtual machine might take minutes to boot (as was often the case with our own appliance), a container might only take seconds, in large part because the underlying OS is already running. By contrast, each VM on a host must boot its own OS. Containers are much lighter-weight. Per Figures 1 and 2, inspired by Docker’s own [30], whereas a host might have sufficient resources to run a few VMs in parallel, that same host might have sufficient resources to run more containers by an order of magnitude. Docker ultimately allows applications and their dependencies (or, in our case, programming environments) to be “containerized” (i.e., packaged) in an “image” that can be started quickly and portably in production.

On any server (or desktop or laptop) with Docker Engine [23] installed, which itself is free, you can start one such container running Ubuntu Linux, for instance, with an interactive terminal attached, with just:

```
docker run -it ubuntu
```

By default, very few packages are preinstalled, so you can alternatively prepare your own custom image by creating a text file called `Dockerfile` like:

```
FROM ubuntu
RUN apt update && apt install -y clang gdb valgrind
```

You can then “build” that image with

```
docker build .
```

and “push” it to the Docker Registry [36] with

```
docker push NAME
```

wherein `NAME` is a unique identifier for that image. Thereafter, anyone can “pull” and start a container running that same image with just

```
docker run -it NAME
```

with `clang`, `gdb`, and `valgrind` already installed for them.

4 IMPLEMENTATION DETAILS

For students, however, it wasn't a headless, client-side CLI that we wanted but a web-based GUI plus CLI instead, so that students would not need to install any software themselves. Containers running on servers only offered a potential back end. We just needed a front end to which to connect those containers.

4.1 CS1 IDE

For a front end, we initially used Cloud9 IDE [16], an open-source integrated development environment (IDE), complete with code editor, file explorer, and terminal window, backed by per-user Docker containers server-side, precisely the model we had in mind. At the time, it was hosted, along with the back-end containers, by an Amsterdam-based startup, with whom we collaborated to build our own "CS1 IDE" in the cloud, with each student's container based on our own Docker image, `cs1/ide`. (See Appendix for complete `Dockerfile`.) Using Cloud9's SDK to write plugins, we customized the IDE's GUI for teaching and learning, adding to it, for instance, a "presentation mode" (with enlarged text) for teaching assistants' sections and a graphical debugger for students, built atop GDB/MI [29]. We also implemented a virtual rubber duck that quacks pseudorandomly to encourage rubber-duck debugging [28].

In time, Cloud9 was acquired by AWS, and Cloud9 IDE evolved into AWS Cloud9. Unfortunately, AWS Cloud9 assumed a VM-per-user model, but, to reduce computational costs by an order of magnitude, we ultimately reconstructed a container-per-student model, using Kubernetes [32] to orchestrate a cluster of containers ourselves, each connected to the IDE's front end.

The end result was a full-fledged Linux environment per student in the cloud, per Figure 3, tailored to the course's instructional needs, requiring only that students create an account and log in at term's start, using only a browser. Once logged in, a container running the course's own image awaited, to which a persistent home directory was attached for the student. Educational grants from AWS (available, as before, by application to all educators) covered the cost.

4.2 Visual Studio Code for CS1

Upon AWS's acquisition of Cloud9, we found ourselves to be system administrators again, with CS1 IDE's cluster of containers no longer managed by Cloud9 but by us, a distraction from teaching we had hoped to avoid. Unfulfilled, too, was a desire to offboard students from CS1 IDE to their own PCs and Macs at term's end so that they could continue to program client-side without the course's own infrastructure. We encouraged students to install a popular [42] (and free) editor like Visual Studio (VS) Code along with, for instance, a Python interpreter, but that transition from cloud to computer was not nearly as smooth as we would have liked, particularly with the user interfaces so different. Students could alternatively install Docker and run an offline version of CS1 IDE locally, but we preferred that they "graduate" instead from the course's own interface.

In 2021, though, GitHub launched Codespaces [14], a cloud-based version of VS Code backed by Docker containers. 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. (Alternatives like Replit [37] offer the former but not the latter. And other alternatives like Codio [15] only offer paid tiers.) We thus re-implemented most of CS1 IDE's plugins as VS Code extensions using the VS Code API [8]. And we developed a web application at `Anonymized URL` that, using GitHub's Codespaces API [6], automates the process of creating, within the course's own "organization" on GitHub, one "repository" per student, each with its own "codespace" (i.e., container) based on our own Docker image, `cs1/codespace`, and redirecting them thereto, with the containers themselves hosted by GitHub, per Figure 4. (See Appendix for complete `Dockerfile`.) Via a `.devcontainer.json` file [17] that we "commit" to each student's repository via GitHub's Repos API [7], the course's extensions are preinstalled in each codespace, and VS Code's interface is preconfigured with the course's recommended settings [45]. For instance, we are able to preinstall extensions for C and Python and configure students' terminal windows to use Bash by default via JSON like the below. (See Appendix for complete `.devcontainer.json`.)

```
{
  "extensions": [
    "ms-vscode.cpptools",
    "ms-python.python"
  ],
  "settings": {
    "terminal.integrated.defaultProfile.linux": "bash"
  }
}
```

If we happen to update the course's image mid-semester, we deploy the update to students via a VS Code extension, which prompts them to rebuild their container, preserving their own files therein.

Per Figure 5, the end result is, again, a full-fledged Linux environment per student with VS Code as its front end, this time implemented as software as a service (SaaS) managed by GitHub rather than infrastructure as a service (IaaS) orchestrated by us.

CS1's adaptation of VS Code is freely available to teachers and students, per the Appendix, and can be used to write, debug, and execute code in any language for which a compiler or interpreter is installed, either in advance in our image or manually thereafter, with (or without) any learning management system. We ourselves have students download assignments into their codespaces via `curl` and submit them via another command (or manual upload) to us. In addition to C and Python, our own image includes support for C++, Java, JavaScript (via Node.js), Ruby, and SQL (via SQLite). A preinstalled X server and VNC client provide support for graphics as well, as via Swing or Tkinter.

Offline Support

By way of VS Code's Remote Development extension pack [18], students can even run VS Code locally when offline but still connect via SSH to their codespace in the cloud when online. If comfortable installing Docker as well, students can even run their own

Figure 3: CS1's first web-based programming environment, CS1 IDE, was built atop AWS Cloud9, backed by a Kubernetes cluster of Docker containers. Upon login, students were allocated a newly created container to which their own home directory was attached. Students could then write, debug, and execute code within that container via a web-based UI, complete with a code editor, file explorer, and terminal window, with AWS Cloud9's default UI both simplified and enhanced for teaching and learning via the course's own plugins.

containers locally, completely offline. For those more-comfortable students and especially staff, the course also has a "headless" image, `cs1/cli`, that students can use offline without VS Code. (See Appendix for complete `Dockerfile`.) The course further provides a command-line tool written in Python, `cli`, otherwise known as CS1 CLI, that automates the creation of client-side containers using that image. While the CLI lacks, by design, VS Code's GUI, it enables students to start, within seconds, a Linux container on their PC or Mac, mounting within it their current working directory. CS1 CLI is also freely available to teachers and students, per the Appendix. Via a command-line argument, it supports teachers' own custom images, too.

DIY Options

Each of Docker, VS Code, and Codespaces can be used (at no cost by teachers and students) without any dependencies on our own CS1. Per Section 3, any teacher can create their own `Dockerfile` and, in turn, image, based or not based on our own. Students can run that same image with `docker` alone or the teacher's own command-line wrapper, with or without VS Code. Each of CS1's extensions can be independently installed. And any teacher can sign up for GitHub Global Campus [10] to use Codespaces for free in their classes via GitHub Classroom [11], a web application via which students can "accept" a teacher's assignment, which itself is just a repository with its own codespace, preconfigured with the teacher's own `.devcontainer.json`, based or not based on our own, per GitHub's own documentation.

5 RESULTS

Since its debut in 2015, CS1 IDE was used by more than 500,000 users. And, since its debut in late 2021, CS1's adaptation of VS Code has been used by more than 761,000 users so far. Not only has VS Code atop Codespaces supported our pedagogical goals of providing students with a standardized environment for C, Python, and other languages, too, it has also eliminated the need for system administration on our end. Via our own `Dockerfile` and `devcontainer.js` file, we can still customize students' containers, preinstalling packages and extensions. That they are containers, too, and not virtual machines, means that students' codespaces start in just seconds, allowing students to focus on their work as well. In students' own words, meanwhile, CS1's adaptation was "accessible," "easy to use," and "helpful," and it "helped with providing 'training wheels.'"

Among the few downsides to date is that VS Code's API for extensions is less featureful in some ways than was Cloud9's own, and we have not been able to simplify VS Code's interface 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. VS Code's API allows for some customizations, though, that Cloud9 did not, and we anticipate integrating automated feedback for students into VS Code's UI beyond the graphical debugger alone. Despite our goal of offboarding students from the cloud to their own PCs and Macs toward term's end, we've realized that we ourselves might not have facilitated such sufficiently. Per one student, "Not really sure how to set up an IDE on my computer directly." We plan to remedy through additional documentation and guidance in future terms.

Since our transition to VS Code atop Codespaces in late 2021, we have already found ourselves with far more (human) cycles than we previously had, enabling us, finally, to focus all the more time on students themselves as well as on development of future extensions for teaching and learning. We have just begun to use GitHub Actions [1] as well, which supports Docker, too, allowing us to autograde students' work in containers identical to their own.

6 CONCLUSION

For CS1 at our university, we have long provided students with standardized programming environments to reduce technical difficulties at term's start, to enable students to focus on learning and, ideally,

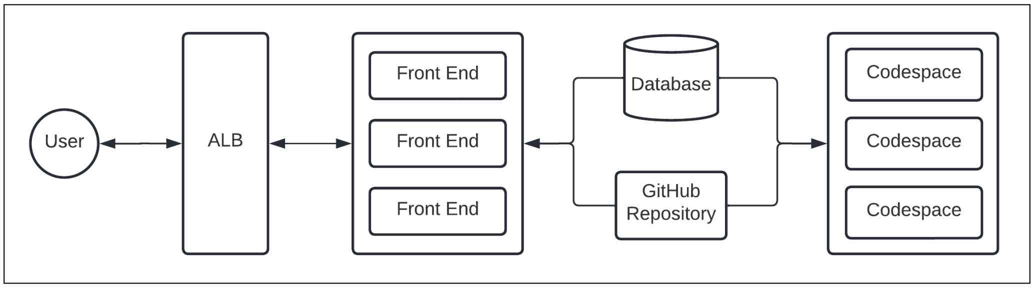

Figure 4: CS1's adaptation of VS Code is built atop GitHub Codespaces. When students visit `Anonymized URL` with their browser, they are routed via an application load balancer (ALB) to one of several front-end web servers. A database stores metadata like the IDs of students' codespaces, while a GitHub repository stores backups of their home directories. Codespaces itself provides students with containers.

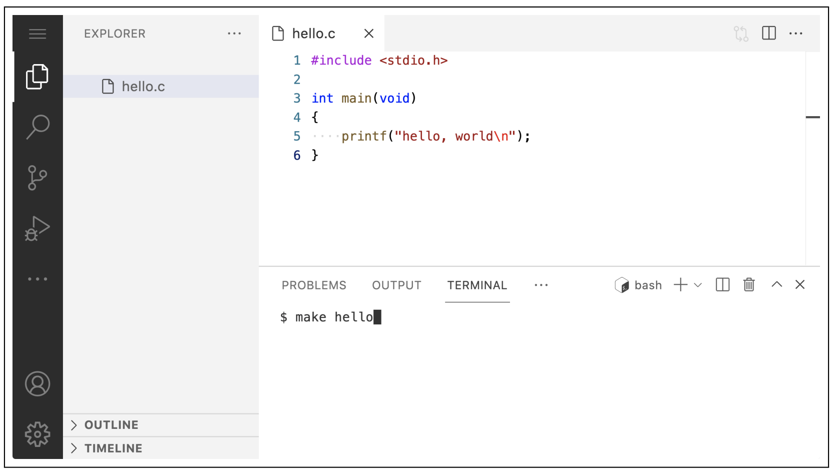

Figure 5: CS1's adaptation of VS Code provides students with a web-based version of Visual Studio Code, connected to a Docker container running the course's own image, its default UI simplified and enhanced for teaching and learning via the course's own extensions and settings.

teachers to focus on teaching. What began as an on-campus cluster of Linux servers evolved into a cloud-based implementation of the same, which itself evolved into client-side virtual machines, which most recently evolved into Docker containers with web-based GUIs back in the cloud. Both pedagogically and technologically, our current adaptation of VS Code atop Codespaces is already proving the most successful implementation to date. Not only has the SaaS-based 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.

Thanks to containerization, students' programming environments now start within seconds rather than minutes. And we can develop and deploy updates in far less time than before. All of our solutions are freely available to all teachers and students elsewhere, per the Appendix.

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 in production, so might containers standardize more easily than ever programming environments for students.

APPENDIX

The course's adaptation of VS Code atop Codespaces is freely available for teachers and students at `Anonymized URL`; its `Dockerfile` and `.devcontainer.json` are at `Anonymized URL`. CS1 CLI is freely available for teachers and students at `Anonymized URL`; its `Dockerfile` is at `Anonymized URL`. And the `Dockerfile` for CS1 IDE is at `Anonymized URL`.

ACKNOWLEDGEMENTS

Our thanks to Amazon Web Services, GitHub, and Microsoft for their support of this work. This work's author has also consulted part-time for GitHub.

REFERENCES

[1] GitHub Actions. 2023. https://github.com/features/actions
[2] Anonymized. 2010.
[3] Anonymized. 2010.
[4] Anonymized. 2013.
[5] Anonymized. 2022. Anonymized
[6] Codespaces API. 2023. https://docs.github.com/en/rest/reference/codespaces
[7] Repo API. 2023. https://docs.github.com/en/rest/repos
[8] VS Code API. 2023. https://code.visualstudio.com/api/references/vscode-api
[9] Brython. 2023. https://brython.info/
[10] GitHub Global Campus. 2023. https://education.github.com/
[11] GitHub Classroom. 2023. https://classroom.github.com/
[12] AWS Cloud9. 2023. https://aws.amazon.com/cloud9/
[13] Visual Studio Code. 2023. https://code.visualstudio.com/
[14] GitHub Codespaces. 2023. https://github.com/features/codespaces
[15] Codio. 2023. https://www.codio.com/
[16] Cloud9 Core. 2023. https://github.com/c9/core
[17] devcontainer.json reference. 2023. https://code.visualstudio.com/docs/remote/devcontainerjson-reference
[18] VS Code Remote Development. 2023. https://code.visualstudio.com/docs/remote/remote-overview
[19] Docker. 2023. https://www.docker.com/
[20] Amazon Elastic Block Store (EBS). 2023. https://aws.amazon.com/ebs/
[21] Amazon EC2. 2023. https://aws.amazon.com/ec2/
[22] Emscripten. 2023. https://emscripten.org/
[23] Docker Engine. 2023. https://docs.docker.com/engine/
[24] Flask. 2023. https://flask.palletsprojects.com/en/2.1.x/
[25] Amazon Programs for Research and Education. 2022. https://aws.amazon.com/grants/
[26] Thomas F. Griffin and Zack Jourdan. 2012. Educational Use Cases for Virtual Machines. In *Proceedings of the 50th Annual Southeast Regional Conference* (Tuscaloosa, Alabama) *(ACM-SE '12)*. Association for Computing Machinery, New York, NY, USA, 365–366. https://doi.org/10.1145/2184512.2184607
[27] David P Harvie, Jason R Cody, Christopher Morrell, and Tanya T Estes. 2019. Using Virtual Machines to Enhance the Educational Experience in an Introductory Computing Course. In *Proceedings of the 20th Annual SIG Conference on Information Technology Education*. 28–32.
[28] Andrew Hunt and David Thomas. 2000. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Longman Publishing Co., Inc., USA.
[29] The GDB/MI Interface. 2023. https://sourceware.org/gdb/onlinedocs/gdb/GDB_002fMI.html
[30] What is a Container? 2023. https://www.docker.com/resources/what-container
[31] JSLinux. 2023. https://bellard.org/jslinux/
[32] Kubernetes. 2023. https://kubernetes.io/
[33] Oren Laadan, Jason Nieh, and Nicolas Viennot. 2010. Teaching Operating Systems Using Virtual Appliances and Distributed Version Control. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (Milwaukee, Wisconsin, USA) *(SIGCSE '10)*. Association for Computing Machinery, New York, NY, USA, 480–484. https://doi.org/10.1145/1734263.1734427
[34] Pyodide. 2023. https://github.com/pyodide/pyodide
[35] PyPy.js. 2023. https://pypyjs.org/
[36] Docker Registry. 2023. https://docs.docker.com/registry/
[37] Replit. 2023. https://replit.com/
[38] Amazon S3. 2023. https://aws.amazon.com/s3/
[39] Amazon Web Services. 2023. https://aws.amazon.com/
[40] Skulpt. 2023. https://skulpt.org/
[41] Geoff Stoker, Todd Arnold, and Paul Maxwell. 2013. Using Virtual Machines to Improve Learning and Save Resources in an Introductory IT Course. In *Proceedings of the 14th Annual ACM SIGITE Conference on Information Technology Education* (Orlando, Florida, USA) *(SIGITE '13)*. Association for Computing Machinery, New York, NY, USA, 91–96. https://doi.org/10.1145/2512276.2512287
[42] Stack Overflow Developer Survey. 2023. https://survey.stackoverflow.co/2023/
[43] Automating the Installation with Kickstart. 2023. https://docs.fedoraproject.org/en-US/fedora/rawhide/install-guide/advanced/Kickstart_Installations/
[44] Transcrypt. 2023. https://www.transcrypt.org/
[45] User and Workspace Settings. 2023. https://code.visualstudio.com/docs/getstarted/settings
[46] Sander Valstar, William G. Griswold, and Leo Porter. 2020. Using DevContainers to Standardize Student Development Environments: An Experience Report. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 377–383. https://doi.org/10.1145/3341525.3387424
[47] VirtualBox. 2023. https://www.virtualbox.org/
[48] WebAssembly. 2023. https://webassembly.org/
[49] Virtual x86. 2023. http://copy.sh/v86/