Introducing DevOps Techniques in a Software Construction Class

Mark Hills
East Carolina University, Greenville, NC, USA
mhills@cs.ecu.edu

***Abstract*—As more companies adopt techniques related to DevOps and continuous deployment, it is critical for students in software engineering courses to gain hands-on experience with these techniques. In this paper, we describe a collection of assignments given in a graduate software construction class that guides students through the process of creating Docker containers, configuring continuous integration services, constructing a build pipeline, and automating deployment of new versions of a software system when changes are committed to the code repository. We also briefly analyze the performance of students on these hands-on exercises, identifying areas where additional support is needed to ensure student success.**

I. Introduction

DevOps [1]–[3] refers to the integration of development, quality assurance, and operations teams in a company. Interest in DevOps has increased over the last decade as companies such as Amazon and Etsy have promoted DevOps techniques that are claimed to lead to reduced cost and faster time to market. Adopting DevOps practices often requires both technical and cultural changes within a company, as cross-functional teams replace the formerly separate teams mentioned above. DevOps relies heavily on tools and automation to support practices including test automation, build automation, continuous integration [4], continuous delivery [5], and continuous deployment [6].

Given the increased interest in DevOps among companies, it is critical for students in software engineering courses to gain hands-on experience with these practices and the technologies that enable them. In this paper, we describe a collection of assignments given in a graduate software construction class, part of our graduate program in software engineering, that guide students through the process of creating Docker containers, configuring continuous integration services, constructing a build pipeline, and automating deployment of new versions of a software system when changes are committed to the configuration management system.

The remainder of this paper is organized as follows. In Section II, we provide an overview of our software construction class, a class where students improve their programming and abstraction abilities while using professional tools and techniques. Section III then describes the hands-on assignments we have created to introduce some of the technical practices used in DevOps. Section IV reflects back on our experiences using this assignment, and a variant of this assignment, for the last two years. Related work is briefly mentioned in Section V, while Section VI concludes. All assignments described in this paper are available freely for other faculty to use, and are released under an open-source license.

II. Course Overview

The software construction course in our graduate software engineering program is a semester-long elective. Students entering the course are assumed to know Java and to have a programming background equivalent to what would normally be developed in an undergraduate introductory programming sequence. While they also have taken a course on software engineering foundations at that point, most have not created or modified a significant software system.

The course has two main purposes. The first is to develop the object-oriented design and development skills of our students, with a strong focus on using abstraction as a means to reason about software. This is done using a sequence of assignments where students need to use features such as interfaces, exceptions, inheritance, and generic types in the context of extending an existing software system. Java is the working language for the course, although the concepts are applicable to other object-oriented languages. The second is to accustom students to using professional tools and development techniques. Students use GitHub to keep track of the code for each homework assignment, use the IntelliJ IDEA[1] integrated development environment for programming assignments, use Apache Maven[2] as a build and dependency management tool, and use JUnit[3] and JaCoCo[4] for unit testing and code coverage. As part of later assignments in the course, students begin to use the Travis CI[5] continuous integration service to automatically run their unit tests when they push code changes to GitHub. They also use PMD[6] and FindBugs [7] to identify possible bugs and instances of poor Java usage in their code.

III. Assignment Details

In past offerings of our software construction course, some limited attempts were made to introduce DevOps concepts, but this mainly involved using standalone activities that leaned heavily on existing tutorials provided by vendors such as Google. In this section, we provide details of a newer collection

[1]https://www.jetbrains.com/idea/
[2]https://maven.apache.org/
[3]https://junit.org/
[4]https://www.eclemma.org/jacoco/
[5]https://travis-ci.com/
[6]https://pmd.github.io/

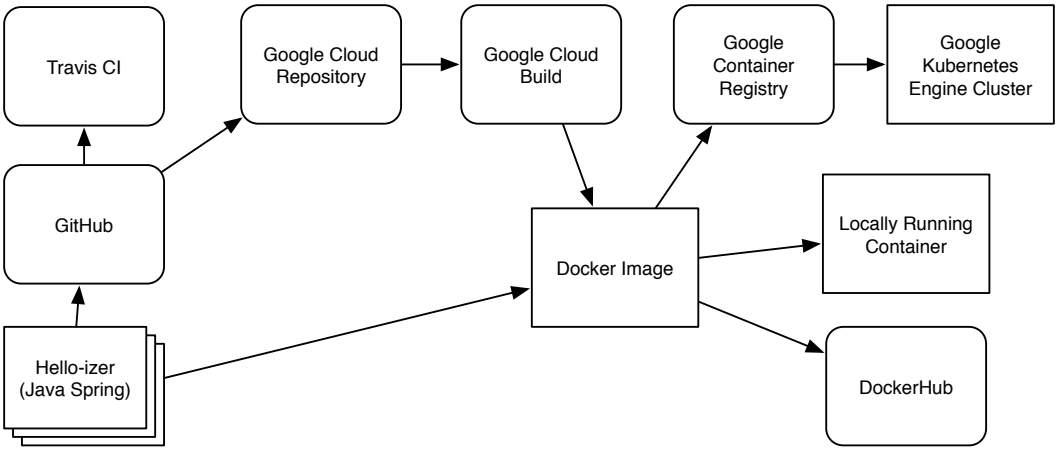

Fig. 1. Assignment Overview: Building a Continuous Deployment Pipeline.

of assignments, built as a sequence of steps, and carried out over the course of the semester, that are designed to teach students about DevOps techniques such as build automation and continuous deployment. These assignments are run in parallel to the main assignments in the class, ensuring that they do not block student progress on other assignments.

A. Assignment Overview

An overview of the different moving parts in this collection of assignments is given in Figure 1. In this figure, the different processes and systems used by students are shown in boxes with rounded corners, while different artifacts edited by, or produced by, the students (e.g., code and configuration files, images) are shown in regular boxes.

- In the first assignment, students enable continuous integration for a Java Spring[7] application named the Hello-izer. This uses the Hello-izer code, GitHub, and Travis CI.
- In the second assignment, students create a Docker[8] image for this same application and run it within a container. This uses the Hello-izer code, a created image, a locally-running container for the application, and DockerHub[9].
- In the third assignment, students use Google Cloud Platform[10] and Kubernetes[11] to deploy this image to a Kubernetes cluster. This uses the Hello-izer code, a created Docker image, the Google Container Registry[12], and a Google Kubernetes Engine cluster.
- In the fourth assignment, students link GitHub and Google Cloud Platform to enable automated build and deployment support, allowing new images to be created and deployed when changes are pushed to GitHub. This uses the parts used in the third assignment, as well as Google Cloud Source Repositories[13] and Google Cloud Build[14].

Details of each of these assignments, including how the various processes and artifacts mentioned above are used, follow.

B. Continuous Integration

Continuous integration [4] is a practice developed as part of Beck's Extreme Programming methodology [8]. In continuous integration, software teams integrate their changes frequently. Automated builds and tests are used to quickly find bugs introduced as part of this process. One popular tool for continuous integration is Travis CI. Travis CI is a software service that can monitor a source repository, listening for change events. When one is received, it can pull the current version of the code from the repository and run a configured build and test process, reporting the results back to the maintainer of the system.

In the first assignment, students start with a simple Java Spring application named the Hello-izer. Since this was not a class on web programming, and students may have limited background in that topic, the goal was to select a web technology that students could easily understand. Spring applications are written in Java—the language used in the course—and use Java annotations to indicate that specific methods can be used to respond to web requests. They also are self-contained for simple applications, avoiding the complexities of many web development frameworks. The Hello-izer is built as a RESTful [9] web service that provides a single feature: when it receives a request, it returns a response that includes the current date and time. The default response does not include a name, but instead refers to the requester as "Anonymous User". Optionally, it is possible for a name to be passed as a parameter, in which case it *should* include the name in its response. However, in the version of the code given to the students, this functionality is not working, so the response

[7]https://spring.io/
[8]https://www.docker.com/
[9]https://hub.docker.com/
[10]https://cloud.google.com/
[11]https://kubernetes.io/
[12]https://cloud.google.com/container-registry
[13]https://cloud.google.com/source-repositories
[14]https://cloud.google.com/cloud-build

always uses "Anonymous User" as the name. Two tests are included with the code, one of which passes, and one of which fails because of this missing feature.

Given this as a starting point, the goal of the assignment is for the students to configure their projects to use Travis CI. To do this, students need to add a configuration file to their projects that includes information about the build environment needed for the Hello-izer. If students are using their own repositories in an organization they control themselves, they also need to grant Travis-CI permissions to their GitHub repository for the project. Since the project includes a Maven configuration file, additional configuration beyond this is not needed—Travis CI can use common configuration file formats such as Maven directly. Once students have configured the build environment, a change to the code followed by a push of this change to GitHub will trigger a Travis CI build, which will fail because of the failing test. Students will receive a notification of this failure on the Travis CI website and (if configured) in their email, and can then examine the build log on Travis to identify the source of the problem.

Once they have done this, they can fix the problem in the code, which is a straightforward exercise for students in the class. They can then commit their changes and push them back up to GitHub, which will trigger another build. In this case, assuming the change fixes the bug, the build will pass, showing them the expected behavior of the continuous integration tool when all the tests pass. Optionally, students can further experiment with the code, or add additional build environments (in this case, different versions of Java). Once introduced here, assignments in the main assignment sequence also use Travis CI.

C. Building and Using Containers

Installing many custom software applications can be tricky, since they may have dependencies on specific versions of the operating system and various libraries. These dependencies could also conflict with those of other applications, making it hard or impossible to install both at once, or to upgrade one once installed. Containers are one potential solution to this problem. A container includes the application and all dependencies. Since multiple containers can be run at once, different applications can all be installed in their own containers with their own dependencies.

Docker is the de facto standard for containers. Docker uses a configuration file, called a *Dockerfile*, to define a container *image* which is used to create a running container. These images are often shared on sites such as DockerHub, but can also be kept in other image repositories, such as the Google Cloud Platform Container Registry. New versions of an application are released by providing a new image, not by upgrading the software in place, making it easy to repeat the install process and to revert back to an earlier version of the system if needed.

In the second assignment, students start with the application from the first assignment, or from instructor-provided code if they were not able to complete the prior assignment. The goal of the assignment is to automate the creation of a Docker image for the Hello-izer as part of the build process. To do this, students start by creating a Dockerfile that configures the image. This file specifies a base image—in this case, a preconfigured Linux install that includes a Java development environment. Students also need to add the proper commands to install and start the Spring application. Once this is done, students can create the image. This image will include the current version of the Hello-izer, configured to run when a container is started. After this, students can test the image by creating a container and making sure they can connect to the Spring website running inside the container.

Once this is working, there are two final steps students need to take. First, they need to modify the Maven file to include the creation of a new image, with the current version of the Hello-izer application, as part of the build process. Second, students need to upload the created image to DockerHub. This allows the image to be shared with other users, including with the instructor for grading purposes.

D. Deploying Containers with Kubernetes

Quite often, a deployed system is actually a collection of different systems, potentially running on different machines, and working collaboratively to respond to application users. There are multiple reasons that this may be true, including: application logic may be divided between client, server, and database layers, as in a traditional three-tier client-server architecture; applications may be divided up into microservices [10], with each service running in its own container; or systems may use techniques such as load balancing to coordinate multiple running instances of the same application. One solution for defining and managing the various moving parts in such systems is Kubernetes, which allows the deployment and management of applications in a collection of nodes called a *cluster*. Kubernetes ensures the running system matches the defined configuration, e.g., by starting a new container if an existing container crashes.

In the third assignment, students start with either the Docker image they built as part of the second assignment, or with an instructor-provided Docker image if they were not able to get that part of the assignment working. The goal of the assignment is to deploy this image to a Kubernetes cluster, with two load-balanced nodes running containers based on the Hello-izer image. To start, students need to claim Google Cloud Platform (GCP) education credits provided by Google for the class and set up a GCP *project*, which is used to house a collection of related cloud services. Once this is done, they install the Google Cloud SDK[15], which provides a command-line interface for managing these services. Using this SDK, they can then upload a Docker image to the GCP Container Registry. This serves a similar purpose as DockerHub, providing a service for managing Docker images. In this case the images are private, which would be a more typical scenario when the images represent software being developed within a company.

Once the image is uploaded to the Container Registry, students can then set up a Kubernetes cluster. As with the

[15]https://cloud.google.com/sdk

prior step, they do this by following instructions provided by the course instructor, as well as instructions provided as part of existing Google tutorials. After following these instructions, they have a cluster made up of two nodes, but with nothing running on either. Students then deploy the uploaded image from Container Registry to these nodes, creating a container on each node. As a last step, students expose this cluster to external traffic by creating a load balancer, which manages incoming traffic and directs it to one of the two running containers. Optionally, students can experiment with some Kubernetes features, e.g., by intentionally crashing a node and watching Kubernetes recreate it to maintain the specified two nodes.

E. Continuous Deployment

At this point, each student has a Kubernetes cluster running the Hello-izer application. When the application is updated, most of the process outlined in the prior assignment must be followed to update the running version of the application: a new image must be created, this image must be pushed to the Container Registry, and this image must then be deployed from the Container Registry to the running cluster.

In the fourth assignment, students automate this process by creating a continuous deployment pipeline. This starts with the same image definition as in the second assignment and the Kubernetes cluster set up in the third assignment. If students could not complete the third assignment, the instructor needs to work with them (individually or in a group) to walk through the required steps to get the initial Kubernetes cluster set up.

At the start of this assignment, students need to set up two services: Google Cloud Build, and Google Cloud Source Repositories. To start, students simply need to enable Google Cloud Build. Students then create a new Cloud Source Repository (CSR) as a mirror of their existing GitHub repository. This is configured so that any changes pushed to GitHub trigger an update of the CSR, keeping the two in sync. With this in place, students then follow two existing, Google-provided tutorials to actually set up the build instructions, using a process similar to what they used in assignment one for configuring Travis CI. This involves adding a new configuration file into the repository which contains information about how to build the application. A Cloud Build configuration specifies several different *builders*, each of which handles different types of build tasks. The first used here is to build the project according to its Maven file, which compiles the Java code into a running Spring application and then runs any provided tests. If these tests pass, the second builder then uses the provided Dockerfile to create a new Docker image, based on the current version of the application code. This also updates the version of the image in the Container Registry with a new image containing the newest application code.

At this point, a build can be triggered manually, but code changes will still not trigger an automated build. The instructions given to the students walk them through the process of adding a build trigger, which will cause Google Cloud Build to rebuild the Hello-izer when changes are made to the CSR for the project. Students verify this works by making a small change, committing the change, and pushing their changes to GitHub. When configured correctly, this will then synchronize the changes to the CSR and will automatically start a build, uploading a new image to the Container Registry. Students can then manually deploy the new image to their Kubernetes cluster, as was done in the prior assignment, to see that their changes are included in the current version of the image.

The final step is to actually link the build and deploy process. Using instructions from the course instructor, in tandem with tutorials provided by Google, students do this with some additional changes to the configuration file for Google Cloud Build. These changes introduce a new builder, for Kubernetes, that is configured to deploy the new image to the cluster configured in the third assignment, creating new containers based on this new image. Once this has been added to the configuration file, students then make another change to the code, modifying the output from the RESTful service. When this change is committed, the build process described above occurs. The newly added builder then takes the new image and deploys it, ensuring the version of the system running in the cluster is kept up to date with the current version of the code.

IV. Retrospective

Student reactions to the assignments described in Section III were generally quite positive. In a standard survey given for the course, students commented that they found the assignments beneficial, with one stating that the assignments gave them a "tremendous feeling of accomplishment" and that they should be released publicly. Student generally had few problems with the first two assignments, but some struggled with the final two. Looking back over the assignments, we believe the following lessons can be taken for how to improve them going forward.

a) Students Need Help with the Command Line: Many students, even in a graduate software engineering program, are not very comfortable using the command line. They are instead accustomed to using GUIs and web-based interfaces. We are planning to add an additional introductory assignment on basic command line usage to help prepare them for the later activities, which make extensive use of command-line tools including the Docker command-line interface and the Google Cloud SDK. Existing assignments already focus on using Git.

b) Provided Documentation Must Be Supplemented: Most vendors, including Google, provide extensive documentation with their products. This includes basic tutorials, more detailed guides, and walk-throughs that illustrate how to use the products to solve specific problems. Unfortunately, the level of technical background assumed by these documents is often higher than that possessed by students. This led to challenges in using this documentation as part of the assignments described above, and in the prior year—students often got lost in the documentation, and had trouble understanding which steps they needed to follow verbatim, versus which needed to be adapted to reflect differences in their tasks. Although we already supplemented this documentation with our own explanations, even more is needed to ensure the required steps are clear to students. While it would also be possible to avoid this documentation

completely and provide our own, we feel it is important for students to learn to effectively use such documentation.

c) Smaller Steps Help: Along with the last point, we have found that breaking the assignments into smaller tasks helps students to complete them successfully. If there are too many steps, with too many moving parts, students that get lost part way through have a harder time getting back on track and completing the assignment. Based on this, we plan to revisit the assignments described above, possibly breaking the fourth down into two smaller activities, one for automating the build of the image, and another for automating the deployment of the newly-built image to Kubernetes.

V. Related Work

DevOps has been a topic of significant interest to researchers, but there has been less work on how these practices can be incorporated into the classroom. Krusche and Alperowitz [11] describe a project course where continuous integration and continuous delivery techniques are introduced and used in support of real-world projects, while Süß and Billingsley [12] instead focus on using continuous integration as a feedback mechanism for classes with a large number of students but limited teaching resources. Benni et al. [13] present a graduate-level course that includes both software architecture and DevOps principles in the context of a realistic software project, while Bobrov et al. [14], [15] discuss both a graduate-level DevOps course and industry training. Kuusinen and Albertsen [16] focus on both the technical and cultural aspects of DevOps and continuous delivery as part of a two week course taught with industry. One difference between this work and our own is that the courses in the cited work are all either DevOps courses or are traditional project-based courses, while the assignments presented above have been designed to integrate technical DevOps concepts into a course on software construction that was not traditionally focused on DevOps.

More similarly to our own setting, Eddy et al. [17] focus on how to integrate some DevOps techniques—in their case, continuous integration and continuous delivery—into a traditional software engineering course using a set of tools called the Continuous Delivery Educational Pipeline [18]. We instead use Google Cloud Platform, which gives students exposure to professional tools at the expense of additional complexity. Rong et al. [19] propose an educational support system, *DevOpsEnvy*, to give students practice with a curated set of tools. Finally, Christiansen [20] used DevOps techniques, especially around containers, for a cloud computing course, with students submitting extensions to an online game by uploading images to DockerHub.

VI. Conclusions

In this paper we presented a series of assignments designed to introduce students to DevOps techniques such as build automation, continuous integration, containers, and continuous deployment. We have reported an encouraging initial response from our students, who have found the assignments beneficial. We plan to keep improving these assignments while also conducting a more in-depth and rigorous study of their effectiveness in future classes. All assignments being developed are freely available for faculty to use in their own classrooms.

Acknowledgments

This research is supported in part by NSF grant EEC-1730568. Classroom activities have been supported in part through the Google Cloud Teaching Credits program.

References

[1] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," *IEEE Software*, vol. 33, no. 03, pp. 94–100, 2016.

[2] G. Kim, P. Debois, J. Willis, and J. Humble, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[3] N. Mendes, "DevOps Maturity Model Report: Trends and Best Practices in 2017," 2017. [Online]. Available: https://www.atlassian.com/blog/devops/devops-culture-and-adoption-trends

[4] M. Fowler, "Continuous Integration," 2006. [Online]. Available: https://martinfowler.com/articles/continuousIntegration.html

[5] J. Humble, "Continuous Delivery," 2017. [Online]. Available: https://continuousdelivery.com/

[6] S. Pittet, "Continuous Deployment," 2020. [Online]. Available: https://www.atlassian.com/continuous-delivery/continuous-deployment

[7] D. Hovemeyer and W. Pugh, "Finding More Null Pointer Bugs, But Not Too Many," in *Proceedings of PASTE 2007*. ACM, 2007, pp. 9–14.

[8] K. L. Beck, "Embracing Change with Extreme Programming," *IEEE Computer*, vol. 32, no. 10, pp. 70–77, 1999.

[9] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[10] J. Lewis and M. Fowler, "Microservices," 2006. [Online]. Available: https://martinfowler.com/articles/microservices.html

[11] S. Krusche and L. Alperowitz, "Introduction of Continuous Delivery in Multi-Customer Project Courses," in *Proceedings of ICSE 2014 Companion*. ACM, 2014, pp. 335–343.

[12] J. G. Süß and W. Billingsley, "Using Continuous Integration of Code and Content to Teach Software Engineering With Limited Resources," in *Proceedings of ICSE 2012 Companion*. IEEE Computer Society, 2012, pp. 1175–1184.

[13] B. Benni, P. Collet, G. Molines, S. Mosser, and A. Pinna-Dery, "Teaching DevOps at the Graduate Level - A Report from Polytech Nice Sophia," in *Proceedings of DEVOPS 2018*, ser. LNCS, vol. 11350. Springer, 2018, pp. 60–72.

[14] E. Bobrov, A. Bucchiarone, A. Capozucca, N. Guelfi, M. Mazzara, and S. Masyagin, "Teaching DevOps in Academia and Industry: Reflections and Vision," in *Proceedings of DEVOPS 2019*, ser. LNCS, vol. 12055. Springer, 2019, pp. 1–14.

[15] E. Bobrov, A. Bucchiarone, A. Capozucca, N. Guelfi, M. Mazzara, A. Naumchev, and L. Safina, "Devops and its philosophy: Education matters!" in *Microservices, Science and Engineering*. Springer, 2020, pp. 349–361.

[16] K. Kuusinen and S. Albertsen, "Industry-Academy Collaboration in Teaching DevOps and Continuous Delivery to Software Engineering Students: Towards Improved Industrial Relevance in Higher Education," in *Proceedings of ICSE (SEET) 2019*. IEEE / ACM, 2019, pp. 23–27.

[17] B. P. Eddy, N. Wilde, N. A. Cooper, B. Mishra, V. S. Gamboa, K. M. Shah, A. M. Deleon, and N. A. Shields, "A Pilot Study on Introducing Continuous Integration and Delivery into Undergraduate Software Engineering Courses," in *Proceedings of CSEE&T 2017*. IEEE, 2017, pp. 47–56.

[18] B. P. Eddy, N. Wilde, N. A. Cooper, B. Mishra, V. S. Gamboa, K. N. Patel, and K. M. Shah, "CDEP: Continuous Delivery Educational Pipeline," in *Proceedings of the 2017 ACM Southeast Regional Conference*. ACM, 2017, pp. 55–62.

[19] G. Rong, S. Gu, H. Zhang, and D. Shao, "DevOpsEnvy: An Education Support System for DevOps," in *Proceedings of CSEE&T 2017*. IEEE, 2017, pp. 37–46.

[20] H. B. Christensen, "Teaching DevOps and Cloud Computing using a Cognitive Apprenticeship and Story-Telling Approach," in *Proceedings of ITiCSE 2016*. ACM, 2016, pp. 174–179.