Eclipse Che for Fair and Easy Education

Leonhard Applis [1]

Abstract: Eclipse Che is a revolutionary cloud-based IDE, inheriting the approach of building cloud native software within a remote environment. Instead of developing on a local machine, a remote development-server is accessed via web-browser and the artifacts are compiled into docker-containers.

This paper shortly summarizes the required environment, introduces the concept and workflow of Eclipse Che and evaluates the benefits and downsides of this approach. Focus is set on a new point of view, enhancing computer studies and programming classes with this technology.

Keywords: Eclipse Che; Cloud Native; DevOps; Docker; Kubernetes; Higher Education; Software Development

1 Introduction

Children look into the clouds and count sheep. Developers look into the clouds and check their applications. Cloud technologies are already state of the art, enabling global players like Google to run applications on a world-scale. The market-leaders stocked up their portfolios with a variety of different cloud technologies, ranging from cloud computing, cloud storage, database-as-a-service or rented authentication. Modern software needs to be scalable, self-contained, and should run on any device, as well as in the cloud. This trend is also visible in Google searches 1.

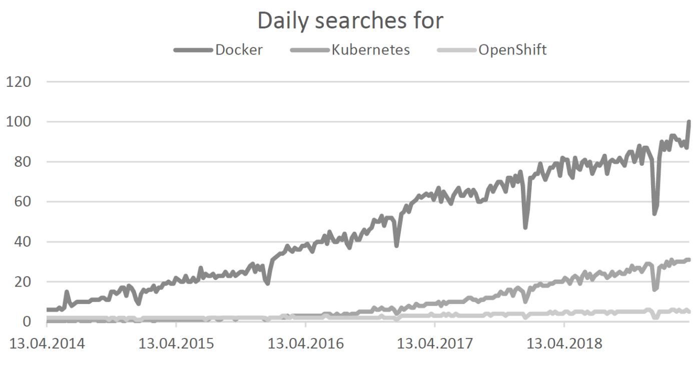

Abb. 1: Rising interest in Cloud Tec, Data provided by Google Trends

[1] Technische Hochschule Georg Simon Ohm, Leonhard.Applis@Protonmail.com

Figure 1 shows the steadily rising interest in the most popular cloud technologies. Docker is a container-engine enabling OS-Virtualization, Kubernetes is a Container-Platform manager and OpenShift is a software-suite based on Kubernetes. These technologies will be summarized in 2.1.

As cloud-technology such technologies are referred which achieve five attributes: On-demand self-service, broad network-access, resource pooling, rapid elasticity and measured service [Me11]. Cloud technology usually splits into two components: infrastructure and services, where Docker and Kubernetes are considered infrastructure and provide a platform to host services. Most of the attributes are (pseudo-) achievable by the infrastructure alone – such as restarting containers. Software running as a service can be put into two categories: *Cloud ready* – meaning that it can be run in common cloud environments, such as Kubernetes. The term cloud ready does usually not include live-scalability. Cloud-readiness is mostly achieved by removing dependencies and either running on bare-linux or being ported into a docker-container. Lately the term *Cloud native* came up [Bi17], referring to software which is built from the very first steps to run inside the cloud. This kind of software is usually running inside a docker-container and both resilient and elastic by design [To17]. Resilience refers to the ability of handling failure, both from external sources and restarting/configuring themselves. Elasticity is the ability to request more resources on demand, performing more computations when required, but also to free the resources if not needed. Cloud-ready software usually achieves only a certain degree of resilience. Cloud-Native applications are aware of their context and collect meaningful metrics for the platform that they cannot reach other services or need more capacities [Mi18]. The applications must also be able to use the gained resources in a productive and healthy way. Modern cloud-platforms can perform pseudo-scalability for cloud ready-services by starting multiple instances of the same service. The cloud-platform performs load-balancing between these instances. This naive approach can work out fine, but for example starting two services working on the same database will not yield any real scalability. Another common problem arises when two services are started on the same machine and need the same port, just to name two problems with cloud ready software in a cloud environment.

As Cloud is a central element of modern IT-development, it is strange to see that it is not yet widely teached across universities in Germany [2]. There are courses to teach either classic virtualization or docker-basics, but the knowledge of full-stack cloud-platforms and cloud native development is sealed behind corporate doors and conference workshops. Meanwhile, the annual StackOverflow developer-survey [St19] showed, that DevOps-engineers are not only among the most wanted and best paid jobs but also tend to be the happiest participants. Every student either wants to be happy or to be rich, making Cloud an interesting topic for their career.

There are many reasons why there are not many courses, but the most prominent is the complexity of the topic: To build a real cloud native application, the developer

[2] https://www.hochschulkompass.de/studium/studiengangsuche lists currently only 10 german universities with courses on cloud computing, about 30 courses on virtualization

must understand virtualization, system-administration, development, infrastructure and networking. Additionally, the developer needs to use specific tools, know best practices, tests and work in a team. All these factors make a high stake for entrance, also represented in the developer-survey [St19] which states that most DevOps engineers have a decade of experience, usually in operations topics.

Luckily a new open-source software, Eclipse Che, is on the rise to enable everyone for cloud native development. Che is unlike a normal Eclipse-distribution: It is hosted on a server and developers get access with their browser. Instead of installing dependencies, compilers etc. locally, a common workspace is set up, which is shared among the programmers. The software build is run inside a docker-container, making it cloud native by default. With Version 7 upcoming and prominent support from RedHat, Eclipse Che is not a prototype anymore. The latest distributions of RedHats OpenShift are shipped with Che in default. Therefore, it is worth looking at this possible game-changer. With the first part of this paper, the structure and ideas of Che are explained in more detail, the second part of this paper covers general arguments about this approach and gives a detailed overview of reachable benefits for education.

2 Eclipse Che

2.1 Environment and Requirements

Che is hosted as a server-application and is already cloud native. The only requirements are that either Docker, Kubernetes or OpenShift are available for the installation. The differences between these will shortly be summarized.

Docker[3] is a container-platform, where a container is a standardized unit of of software including the code, dependencies and core-functionalities. A container runs on an OS-Virtualization, originating from LXC, has declared interfaces and can be parameterized. One of the core features is to connect containers for bigger projects, e.g. one database container and one web-server-container, which are connected into a virtual network. Docker is mostly famous for this docker-engine, but enriches it with monitoring and logging, as well as the ability to move containers onto different machines.

Kubernetes[4] is a container-orchestration-system based on Docker. Kubernetes picked up the growing problem in managing multiple docker-containers by simplifying resource-management. Additional to deployment and monitoring of container-groups it provides auto-restart mechanisms and scaling based on (custom) metrics. When run on Kubernetes, Che is hosted as a single docker container in the existing platform. The term pod, also later used in this paper, refers to a suite of connected containers. An example pod would be a

[3] https://www.docker.com/
[4] https://kubernetes.io/

simple two-tier web-application, where the database and web-server are two different, but connected containers.

OpenShift[5] provides a software-suite around Kubernetes with the goal to automate cloud native development and delivery. Notable additions include Jenkins, Gitea, Sonarqube and lately Che. Regarding Che there are no notable changes in handling, as Che is simply run on the build in Kubernetes. When run on Kubernetes or Docker Che will require two additional containers for authentication with KeyCloack. For OpenShift the standard-openshift authentication is used.

Eclipse Che can be run on any base-technology on localhost – this is rather for demonstrations as well as for developers working on the Che-Code. When run (locally) on docker, the plugin CheDir enables a portable workspace, which can be used on any docker based Che.

2.2 Technology and Workflow

A normal IDE is software on a computer which makes programming easier than doing it in a plain text-editor and commandline. Common tasks for an IDE are dependency management, debugging, refactoring and auto-completion. It is important that an IDE is not *required* to develop - Vim can write Java files, the JDK can compile and the JRE can run everything manually from console. With that in mind,an IDE is just a tool-stack to make things (a lot) more comfortable – but it also has requirements such as the JDK, the .Jars needed in the classpath etc. as they are only utilized by the IDE. This stack can be considered a *workspace* as it contains everything required to work, with the IDE making the workspace comfortable to use. Eclipse Che does not work like a normal IDE. Che consists of three main components shown in Figure 2[6].

[5] https://www.openshift.com/

[6] source: `https://en.wikipedia.org/wiki/Eclipse_Che#/media/File:Eclipse_Che_-_Workflow.PNG`

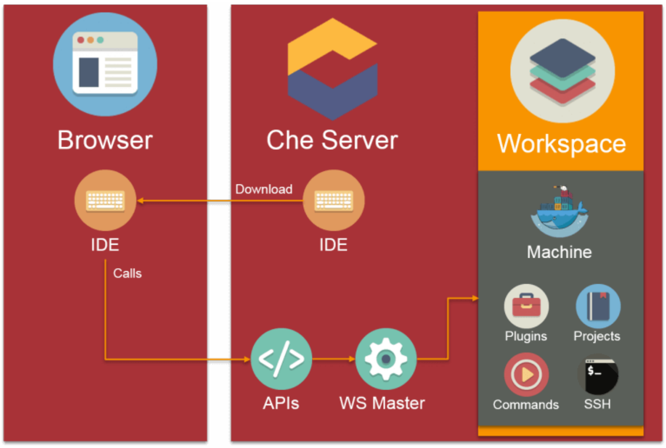

Abb. 2: Che Workflow

1. Workspaces, including Runtimes and IDEs
2. A browser-based IDE
3. An administrative server

A workspace is like the above mentioned common *workspace*. It is a single machine, as a container, containing the project files, compilers, package managers and an IDE-Interface. There are two big differences to a normal development-setup:

- The required items for the workspace are explicitly declared, making the workspace itself a docker image
- The IDE is not graphical – it is a REST-API performing actions like a normal IDE, such as build, writing to files and installing packages.

An additional distinction is about building software. Instead of building the .Jar in the remote-workspace, it is build inside a container in the remote-workspace. While this sounds a bit confusing, it makes sense to separate the actual runtime from the workspace. This is also what happens on a local machine – the executable is compiled and run separately. Having these runtimes inside a docker-container comes with great benefits, outlined later in this paper.

The second component is the web-based IDE, which is basically a web-page performing the required REST-calls and utility-tasks for the developer. It is sent by the administrative server when an authorized user accesses the workspace via browser. These two components can be enriched with plugins. Common plugins are language-extensions which support syntax-highlighting etc. or package managers with their regarding lifecycles. The difference to a normal IDE plugin is that it has two components: One in the workspace-API and one in the hosted web-IDE. Che offers a suite of common plugins dependent on the project. When starting a Maven-project, the workspace is initialized with Maven and Java.

The last component is the Che-administration server. In the administration-server workspaces are orchestrated and monitored. The authorization is also done at the admin-server and not by every workspace. The workspaces can be shipped onto multiple separate machines. The runtimes do not necessarily need to be on the same machines as their workspace. Instead of the web-based IDE, a desktop IDE can be connected to the workspace. This is done by mounting the workspaces-filesystem in the desktop IDE. This will grant access to auto-completion and other features, but will miss Che specific helpers the web-IDE offers. Mounting remote file systems for development sounds lie an infinite source of problems. This feature looks to be rather a proof of concept.

A single workspace can contain multiple projects, whereof every project will have its own runtime. Projects can either share a versioning control or be separated. You can build whole pods in a single workspace with the single containers being the sub-projects. You can also compose applications inside the workspace, meaning that the resulting container of a project can be the base-image of another project. A workspace can be shared among multiple users. Both will work on the same remote filesystem, altering files simultaneous. Due to changes being transmitted as HTTP-requests, the current implementation when working on the same file is a *last write wins*-policy. For version 7 a multi-cursor file-editing will be provided, like it‘s common in drive-documents. This is not to be mixed up with having multiple users work on the same project but on their own workspace. The code, and the workspace itself, can commonly be shared with any versioning tool such as Git. It’s mostly dependent on the team which kind of cooperation they prefer.

As an interesting fact, Eclipse Che is developed with Eclipse Che, proving that complex cloud development is possible. This so-called *dogfooding* usually leads to a product which puts requirements over any a-priori design or foreign interests. Existing components will be enhanced if working with them becomes troublesome, leaving a clean and valuable core product.

3 General Benefits and Handicaps

The primary goals of Che are twofold: First it reduces failure when migrating software into the cloud and enhances the experience of working in a rich remote environment. Second is to set up people into existing projects faster and fail safe.

3.1 Reproduction and Maintenance

The University of Arizona showed [CP16], that about 50% of software inspected was not able to build. Common reasons are missing, changed or faulty dependencies, leading to the common term of *dependency hell* as well as unclear documentation. Experienced programmers are usually able to overcome these issues, given enough time. This continues to be an issue [Cl14], and without a different approach probably always will be.

This problem is solved, if the software is built inside a docker-container and follows the principles of self-containment and image-immutability [Bi17]. The self-containment principle dictates that at build-time every required dependency is given, and a fully functional image is created. A containerized maven-application would include java, maven and every required maven package to run the application in the current state. The image-immutability means to forbid alternations to the environment. No installations at runtime, no change in ports, no creation of database-schemas. These two principles, while being optional, lead to always reproducible and functional artifacts. The main challenge is to migrate existing software and fulfill these properties. Typically, after making the first iterations of the software on a local machine, containerization will yield to 100 problems at once, not only being frustrating but taking tons of time.

With Che, this kind of Big-Bang-Migration is not possible, as from the very first step it is working inside of a docker-container. The properties above can be hold with every change to the image and checked in a normal review-process. Problems will arise one at a time and be solved one at a time, whether they are about code or operations. Che therefore embraces the core-ideas of DevOps [Hu12][Ja16] and agile software development [Eb16]. That being said, of course it's possible to big-bang-migrate into Che, causing the normal problem mentioned above. However, once the errors are resolved, every further development will work more agile by design.

The artifacts (whether containers or pods) produced with Che are self-contained and immutable, therefore every result is reproducible. Any further development is incremental on the existing images and keeps old artifacts stable. As the code is fully build inside containers and are already run remotely, the famous "It works on my Machine!" is eliminated. This does not only enhance deployments, but also has a big impact on quality assurance and the motivation of developers. You can keep multiple projects in a single workspace and launch multiple pods at once, running the full environment. The behavior of this multi-pod is unified and testable, making a stable release of full environments possible.

This is one of the primary goals of the DevOps approach [Ja16]. The workspace can therefore form totally new ways of programming, such as IoT Applications [7] or a full setup of micro-services [Iv18], while being agile and reproducible.

[7] the Plugin Arktik for Eclipse offered IoT Development with Che. While Arktik is unsupported since February 2019, it proved that the concept worked.

3.2 Sharing Results

The topic *docker for reproducible research* has already been promoted [Bo15] and successfully applied to complex topics [St13] such as machine learning. As Clark et al. [Cl14] have used code of several repositories provided from their university, most of the projects can be expected scientific or experimental, 50% of the research results were not reproducible.

The idea to not only ship code, but ship the experiment as a whole makes reproduction much easier. With Che (and using best practices for containers) the experiments done will be containerized and therefore reproducible. Even more complex environments including multiple components can be made in the same paradigms forming a pod. Such a pod could test applications working on an Cassandra-cluster, fullstack web-applications or network-traffic-simulations.

Che also offers a whole new way of sharing the results: If peers wants to inspect the code, they can be granted accounts and get hands on the experiment without any setup. In an further attempt, a common guest-account can be established for anyone interested with read-only rights. This does not only help cooperation and understanding, but is also a good way to verify the results.

3.3 Faster Setup

Onboarding new teammembers is made easy with Che. A newly joining member does not need to install IDEs, languages, tools and packages, they only needs access to their Che-Account and a singular workspace with every dependency build for them. Depending on the complexity of the project this required at most an hour. While this may seem long, the setup time is dependent on the complexity of the workspace. Setting up a complex workplace will also take more time manually. The work for this is done automatic and remote, so the user is free to dive into the documentation or have some coffee. His workspace will be able to launch the latest version immediately. It is also notable, that the (technical) onboarding of developers and operators is streamlined into one single process.

The combination of a quick setup and the usability of every enhancement makes Che especially interesting to recruit troubleshooters into projects, which solve some of the very specialized tasks. With the artifacts being reproducible, the expert can quickly inspect the subject and analyze problems. With a finished workspace they can quickly start working. With their changes being also in docker, they will be visible and functional for the whole team. Therefore, the hardest common problems regarding *quick help* are addressed and solved. As an additional minor benefit, it is easy to hibernate resources using Che with Kubernetes or Openshift. The workspaces collect metrics like any other container and are therefore scalable on demand. The ability of down-scaling the development resources has a big impact on the overall workload. As a further example, the workspace could be shut down out of normal business-time. This attribute is crucial for a rented infrastructure, which is paid usually on both workload and storage.

3.4 Always On

The very first handicap coming with Che is the requirement for a good internet connection. While a common office is already equipped with internet, one may want to work on a train or there is simply a problem with the WiFi in the hotel. With the possibility to setup Che on localhost, the basics and routines can be learned offline. As the main benefits come in terms of teamwork, this is not a real solution.

3.5 Knowledge Required

The primary problem with Che is the requirement for an already dense knowledge of cloud related technology. Setting up the Workspace, docker-images, build and CI scripts is a complex topic and can not be done by someone getting started. This requirement extends when working with a multi-pod-environment. While most of the tasks are by themselves rather simple, the bare amount of such tasks (especially when migrating an existing project to Che) requires a broad knowledge. This can be done by a single expert user, enabling the other developers to use the build-routines and environments provided. However, many problems which arise are related to the cloud-environment and will be hard to solve by a novice user, making him dependent on the expert. It's therefore necessary to get everyone on a certain base-level to solve their own problems, otherwise the expert will be overloaded and the project stuck.

3.6 Different Workflow

Additionally, the missing default "build and run" is a paradigm-shift for many developers, which will take a while to get used to. Clark et al. [Cl14] describe this rather as a change in the mindset, than in actual technology:

"The primary shift that's required is not one of new tooling, as most developers already have the basic tooling they need.
Rather, the needed shift is one of philosophy."

Intuitively this is true: Every developer needs to setup their own workspace – the skills to do this are already there. The only difference is to setup the workspace in a common, documented and descriptive way.

Another problem with Che is the uncommon debugging. As the code is running in a remote-runtime, the common breakpoints and value checks are missing. Instead, one either must be familiar with the languages commandline-debugging or write good logs and metrics. While this will slow down development, it also offers some benefits: The debugging and error-analysis inside the container is the same as it would be in the productive environment. Having meaningful logs with different loglevels and collecting valuable metrics is in general a good attribute for software in any circumstance. It just means additional effort and know-how.

3.7 Vulnerability

The last problem is the vulnerability to malevolent users. Someone who already is in possession of a valid account can drain the shared resources of a full cluster, therefore blocking any other user from work. This can also be achieved by human error. Dependent on the setup, the attacker could also auto-deploy hidden spyware into a production environment. While this is also a problem for normal developer teams, the fast pace of deployments with Che requires additional care. A rich CI/CD environment, such as a well-maintained Jenkins may lead to less-detailed code-reviews, as both unit- and end-to-end-tests succeed. It is therefore important to not only restrict access to project resources, but also the double-check and guard the releases.

Some security risks can also emerge from docker-images as well [Pe13] but are out of scope for this paper.

4 Benefits for Education

In addition to the previous mentioned benefits, there are more points about Che that are especially important for educational purposes, such as courses in the university, professional school or research departments. For simplicity only universities will be addressed.

4.1 New motivational and intuitive Courses

Eclipse Che provides the possibility to learn primary cloud-technologies hands on and in a simple and supportive manner. Unfortunately, these topics are currently not widely teached at university. Che makes it possible to teach students fundamentals of modern software development. Depending on the courses subject, one can either provide scalable software or a running environment. With fitting material provided, it's easy to learn cloud native development, DevOps or Cloud-Operations. Even complicated topics, such as micro-services and IoT-applications can easily be adapted inside Che [Iv18] and performed as for example as group-projects with a common goal. The student will need to understand cloud-components as well as team-work to succeed with Che.

The produced artifacts can simply be shipped as containers, giving comfortable access to the professor. With both the product and the code inside the workspace easily accessible, the professor can grant faster and better feedback for the students, which can have massive impact on learning [HSM85][HLK18]. Additionally, the professor can provide automated end-to-end tests with tools such as Jenkins or TravisCI. Given such an environment, the student will immediately get objective feedback, whether their code fulfills the requirements. This immediate feedback is a big motivator and has a positive impact on the learning experience [Fi05]. With a clear reason on the usefulness of the task, the visible progress and the direct impact of personal effort should lead to intrinsic motivation, which studies show to have a strong effect on education [DR93].

As a requirement, it's necessary to provide a meaningful and fun task. Fortunately, such tasks are not that rare in development. Other studies have shown that only automated and unified testing is not beneficial to education in general [Cr88]. But the tools provided by Che and CI/CD slim down the overhead until both personal feedback and automated can be supplied. If the whole classroom utilizes this system, the educator can also monitor who is having trouble in an automated manner. With a single investment for the tests, feedback for every student is granted at every time. With the single investment of providing a running Che instance (and a short lecture how to use it), feedback on the artifacts and code can be immediately granted to every student, without having the overhead of setting up the workspace. These investments can be done before the course starts at any time and can be reused. It's not dependent on the individual students.

In general, this approach enables studies to be less about the subject, but instead to be about learning the subject. This can be considered one of the primary goals of university courses. Another benefit for education is about fairness in IT-Education.

4.2 Equality and Accessibility

With access to a university Che workspace, every person can develop the same software in the same environment. A well known problem is the necessity of an IOS-build-device for IOS-appdevelopment. Students which are limited in their access to the university-laboratory or cannot afford a mac are therefore handicapped. Another rising problem are GPU-accelerated machine learning algorithms. Depending on the task, training neural networks can be 20 times faster if a CUDA-compatible graphics card is used. The range for these graphic-cards start at 300$ and are therefore not commodity hardware,especially for students. While these technologies are great and important for many applications, the required technologies are exclusive due to their costs.

This hindrance for education is usually addressed with IOS-laboratories and/or cloud-resources paid for by university. The topic of socio-economic factors is beyond the scope of this work, but the impact has been analyzed by the PISA Studies [OE15b], which state that socio-economicly handicapped students fail three times more likely to achieve a basic level of performance. According to PISA, this rate has not significantly changed since the studies started in 2006. Germany scored better on the impact of socioeconomic-factors on education in 2015 than it did in 2006 but is worse than the global average [OE15a]. The Mac-Laboratories and rented cloud space are technically a good attempt at inclusion. But students which are required to get to university to do their assignments are less likely to do so. Making assignments mandatory, providing extrinsic motivation [DR93] has usually a negative impact on the subjective experience.

A different problem comes with machine-learning, which is usually a process of trial and error to some extend. Long waiting times disconnect the action from the result being not beneficial for the learning process [HSM85][DR93].

One may argue that having this kind of anticipation is extra rewarding and therefore highly motivational. From experience, working in this mode becomes exhausting after a period of weeks, especially when errors occur. While this experience is highly personal,providing the fastest infrastructure possible, enabling the fastest results,is the better way. Making progress slower is much easier than making it faster. In addition, the time spent waiting on a machine learning algorithm or moving to the mac-laboratory on weekends is simply dead time. It should therefore be minimized.

With studies showing basics of good learning experiences [Fi05][Cr88] and the impact of motivation [HSM85][DR93], it is mandatory for institutions to provide the best possible environment. While many socio-economic factors cannot be grasped at the university or by the educators, those originating in the students' economic background can be addressed. Most universities already provide free services, such as software-suites, e-libraries or cloud storage.

Simply providing a technical correct solution (like laboratories) is not enough to provide a good environment. One student with a better socio-economic background who can afford a 300$ GPU will have a better baseline for their education. This is simply put unfair. With Che, universities can provide a free, good and accessible development environment for every student.

5 Conclusion

Cloud technology will be dominant in the foreseeable future. Not *everything* will be best fit for cloud, but most of the everyday software benefits from the concepts. With more and more tools making it easier, it's just a matter of time until cloud-tec becomes standard instead of bonus.

With Eclipse Che many problems about teaching students these Cloud Skills and reducing the initial scope for learning, are addressed. While the impacts on motivation and the actual learning results are only hypothetical, most of the technical requirements are solved by Che.

As Eclipse Che is cutting edge it whatsoever lacks many quality of life features, has some bugs, and the design is undefined to some extend.

It somewhat is a chance: Instances such as a university can have massive impact on the design of Che and drive the project to some extend for their own needs. With faculties working on the open source project, either providing code and features, or just testing Che in educational environments, both quality issues and missing features can be solved. Now cloud development is only for corporations. There is hardly any chance to successfully learn it at university. With Che there is another chance to “get back into the game” – and provide corporation-free education.

While this sounds harsh towards companies, unbiased education for students is not a nice thing to have, but a core concept of education in general. Only if that is true, the students can make unbiased decisions and make unbiased research. Being dependent on Google for Universities is in a similar way disturbing, as if a biology faculty would be sponsored by Bayer. One can only imagine a grim future if the curriculum is influenced by profit.

Future Work

As this paper only summarizes benefits and outlines some ideas, the next step would be to put it to the test. In addition, a study could be elaborated on both the acceptance and effect of a CI/CD based development course with Che. Unfortunately, the study would lack any comparison as there are few courses on this topic. These studies could be done with universities and corporations, which may yield very different results.

Literaturverzeichnis

[Bi17] Principles of Container-Based Application Design.

[Bo15] Boettiger, Carl: An Introduction to Docker for Reproducible Research. SIGOPS Oper. Syst. Rev., 49(1):71–79, Januar 2015.

[Cl14] Clark, Dav; Culich, Aaron; Hamlin, Brian; Lovett, Ryan: BCE: Berkeley’s common scientific compute environment for research and education. In: Proceedings of the 13th Python in Science Conference (SciPy 2014). S. 1–8, 2014.

[CP16] Collberg, Christian; Proebsting, Todd A.: Repeatability in Computer Systems Research. Commun. ACM, 59(3):62–69, Februar 2016.

[Cr88] Crooks, Terence J.: The Impact of Classroom Evaluation Practices on Students. Review of Educational Research, 58(4):438–481, 1988.

[DR93] Deci, E.; Ryan, R.: Die Selbstbestimmungstheorie der Motivation und ihre Bedeutung für die Pädagogik. Zeitschrift für Pädagogik, 39(2):223–238, 1993.

[Eb16] Ebert, C.; Gallardo, G.; Hernantes, G.; Serrano, N.: DevOps. IEEE Software, 33(3):94–100, May 2016.

[Fi05] Fink, L: Creating Significant Learning Experiences : An Integrated Approach to Designing College Courses / L.D. Fink. 01 2005.

[HLK18] Hutfilter, A.; Lehmann, S.; Kim, E. J.: Improving skills and their use in Germany. (1516), 2018.

[HSM85] Hughes, Billie; Sullivan, Howard J.; Mosley, Mary Lou: External Evaluation, Task Difficulty, and Continuing Motivation. The Journal of Educational Research, 78(4):210–215, 1985.

[Hu12] Huettermann, M.: DevOps for developers. Apress, 2012.

[Iv18] Developing Multi-Pod Apps with Kubernetes and Che.

[Ja16] Jabbari, R.; bin Ali, N.; Petersen, K.; Tanveer, B.: What is devops?: A systematic mapping study on definitions and practices. In: Proceedings of the Scientific Workshop Proceedings of XP2016. ACM, S. 12, 2016.

[Me11] Mell, P.; Grance, T. et al.: The NIST definition of cloud computing. 2011.

[Mi18] Spring Boot Autoscaler.

[OE15a] OECD: Country Note Germany – Results from PISA. 2015.

[OE15b] OECD: PISA 2015 – Ergebnisse im Fokus. 2015.

[Pe13] Containers and Docker: How Secure Are They?

[St13] Stodden, V. et al: Reproducibility in computational and experimental mathematics - . Examining reproducibility in computer science. 2013.

[St19] Stack Overflow Developer Survey 2019.

[To17] Toffetti, Giovanni; Brunner, Sandro; Blöchlinger, Martin; Spillner, Josef; Bohnert, Thomas Michael: Self-managing cloud-native applications: Design, implementation, and experience. Future Generation Computer Systems, 72:165 – 179, 2017.