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-ready software within a remote environment. Instead of developing on your local machine, a remote development-server is accessed via a webbrowser 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-Ready, Cloud-Native, DevOps, Docker, Kubernetes, OpenShift

1 Introduction

As children we, used to look into the clouds and count sheep. As developers, we look into the clouds and check our applications. Cloud technologies are already state of the art, enabling global players like Google to run applications on a world-scale. The market-leaders got their portfolio stocked up with a variety of different cloud technologies, ranging from cloud computing, cloud storage, database-as-a-service or rentable authentication. Modern Software needs to be scalable, self-contained, and should run on any device, aswell as in the cloud. This trend is also visible in Googles searches (TODO: Abbildung 1).

Figure 1

Figure 1(TODO LINK) 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 [15]. 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 cloudready does usually not include live-scalability. Cloud-readieness is mostly achieved by removing dependencies

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

and either running on bare-linux or being ported into a dockercontainer. Lately the term "Cloud-native" came up [17], 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 elasticby design [8]. Resilience refers to the ability of handling failure, both from external sources and restarting/configuring themselves. Elasticy 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 [16]. The applications must also be able to use the gained resources in a productive and healthy way. Modern cloud-platforms can perform pseudo-scalabilty 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 cloudplatforms and cloud-native development is sealed behind corporate doors and conference workshops. Meanwhile, the annual Stackoverflow developersurvey [22] showed, that DevOps-engineers are not only amongst the most wanted and most 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. So why there are no courses? There are many reasons, but the most prominent is the complexity of the topic: To build a real cloud native application, the developer 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 developersurvey [22] which states that most DevOps engineers have a decade of experience, usually in operations topics. Lucky for us, 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 amongst 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

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

in more detail, the second part of this pa- per 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 cloudnative. 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-plattform, 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 simplyfing 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 simple two-tier web-application, where the database and webserver 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 authentification with KeyCloack. For Openshift the standard-openshift authentification is used. Eclipse Che can be run on any base-technology on localhost – this is rather for demonstrations aswell 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.

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

2.2 Technology and Workflow

A normal IDE is software on your 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 autocompletion. It is important that an IDE is not required to develop – you can write .java-files in Vim, compile on the console and run a .Jar manually. With that in mind, we can see that the IDE is just a tool to make things (a lot) more comfortable – but you also have requirements such as the JDK, the .Jars you need in your classpath etc. as they are only utilized by your IDE. You can consider this stack as your 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 (TODO Figure 2).

FIGURE 2 CHES OVERVIEW

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

A workspace is like your 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 your software. Instead of building the .Jar in your 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 your machine – you compile an executable and run it 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 webpage performing the required REST-calls and utility-tasks for the developer. It is sent by the administrative server when an authorized user accesses his 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 your 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 the workspaces are orchestrated and monitored. The authorization is also done at the admin-server and not by every workspace. The workspaces can be split amongst multiple machines. The runtimes do not necessarily need to be on the same machines as their workspaces. Instead of the web-based IDE, you can connect a desktop IDE to your workspace. This is done by mounting the workspaces-filesystem in your desktop IDE. This will grant you access to autocompletion and other features, but you will miss Che specific helpers the web-IDE offers. My personal oppinion is, that mounting remote file systems for development sounds like a infinite source of problems. I expect this feature 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 subprojects. You can also compose applications inside the workspaces, meaning that the resulting container of a project can be the base-image of another project. A workspace can be shared amongst 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 workspaces. The code, and the workspace itself, can commonly be shared with any versioning tool like 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 Demonstrationen

Das Symbol für Potenzmengen ($\wp$) wird korrekt angezeigt. Es ist kein Weierstraß-p ($\wp$) mehr.

Spitze Klammen können direkt eingegeben werden: <test />

Hier eine kleine Demonstration von microtype: Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. $\sin^2(\alpha) + \cos^2(\beta) = 1$. Der Text gibt lediglich den Grauwert der Schrift an $E = mc^2$. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. $\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{ab}$. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. $\frac{\sqrt[n]{a}}{\sqrt[n]{b}} = \sqrt[n]{\frac{a}{b}}$. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. $a\sqrt[n]{b} = \sqrt[n]{a^n b}$. Er muss keinen

Sinn ergeben, sollte aber lesbar sein. $\mathrm{d}\Omega = \sin\vartheta \mathrm{d}\vartheta \mathrm{d}\varphi$. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

4 Demonstration der Einhaltung der Richtlinien

4.1 Literaturverzeichnis

Der letzte Abschnitt zeigt ein beispielhaftes Literaturverzeichnis für Bücher mit einem Autor [**Ez10**] und zwei AutorInnen [**AB00**], einem Beitrag in Proceedings mit drei AutorInnen [**ABC01**], einem Beitrag in einem LNI Band mit mehr als drei AutorInnen [**Az09**], zwei Bücher mit den jeweils selben vier AutorInnen im selben Erscheinungsjahr [**Wa14**] und [**Wa14b**], ein Journal [**Gl06**], eine Website [**GI19**] bzw. anderweitige Literatur ohne konkrete AutorInnenschaft [**XX14**]. Es wird biblatex verwendet, da es UTF8 sauber unterstützt und im Gegensatz zu lni.bst keine Fehler beim bibtexen auftreten.

Referenzen sollten nicht direkt als Subjekt eingebunden werden, sondern immer nur durch Authorenanganben: Beispiel: **AB00** geben ein Beispiel, aber auch **Az09**. Hinweis: Großes C bei `Citet`, wenn es am Satzanfang steht. Dies ist analog zu `Cref`.

Formatierung und Abkürzungen werden für die Referenzen `book`, `inbook`, `proceedings`, `inproceedings`, `article`, `online` und `misc` automatisch vorgenommen. Mögliche Felder für Referenzen können der Beispieldatei `lni-paper-example-de.bib` entnommen werden. Andere Referenzen sowie Felder müssen allenfalls nachträglich angepasst werden.

4.2 Abbildungen

Abb. 1 zeigt eine Abbildung.

4.3 Tabellen

Tab. 1 zeigt eine Tabelle.

Überschriftsebenen	Beispiel	Schriftgröße und -art
Titel (linksbündig)	Der Titel ...	14 pt, Fett
Überschrift 1	1 Einleitung	12 pt, Fett
Überschrift 2	2.1 Titel	10 pt, Fett

Tab. 1: Die Überschriftsarten

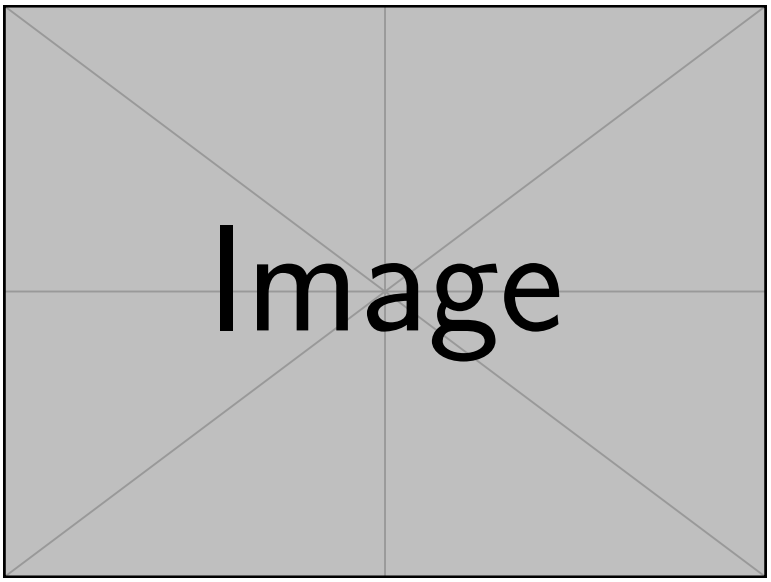

Abb. 1: Demographik

4.4 Programmcode

Die LNI-Formatvorlage verlangt die Einrückung von Listings vom linken Rand. In der `lni`-Dokumentenklasse ist dies für die `verbatim`-Umgebung realisiert.

```
public class Hello {
    public static void main (String[] args) {
        System.out.println("Hello World!");
    }
}
```

Alternativ kann auch die `lstlisting`-Umgebung verwendet werden.

List. 1 zeigt uns ein Beispiel, das mit Hilfe der `lstlisting`-Umgebung realisiert ist.

```
public class Hello {
    public static void main (String[] args) {
        System.out.println("Hello␣World!");
    }
}
```

List. 1: Beschreibung

4.5 Formeln und Gleichungen

Die korrekte Einrückung und Nummerierung für Formeln ist bei den Umgebungen `equation` und `eqnarray` gewährleistet.

$$1 = 4 - 3 \tag{1}$$

und

$$2 = 7 - 5 \tag{2}$$
$$3 = 2 - 1 \tag{3}$$