Docker Evolution Scenario

Benkler, Niko; Haßberg, Tobias; Heinrich, Robert

Institute for Program Structures and Data Organization

1 Docker

Docker is an open source software using container technology to isolate applications on a system.
The big advantage of using this technology is a higher efficiency regarding hardware consumption. This is achieved by running independent containers on a single Linux Kernel instead of using several virtual machines to separate the applications. Furthermore, there is no need to manually install software that is necessary to run the application, as the so-called Dockerfile specifies the environment in which the application is able to run.
Docker builds an *Image* by using a Dockerfile. Starting from an empty server environment, Docker executes the commands of the Dockerfile to set up the docker container, the application environment and the application itself.
Docker can be forced to download the newest software version, if available, when building a new *Image*. This is usually done in the Dockerfile specification, too. As a consequence, the application itself and the software it uses are always up to date.
The docker environment is suitable for efficiently running various containers at a time. It is therefore predestined for Microservice-based applications, as the various Services can be deployed independently in different containers.

2 Docker for CoCoME

As described in the CoCoME installation guide, CoCoME needs a bunch of software that has to be installed manually. This platform dependant software needs to be kept up to date, no matter on what kind of system it is deployed. The basic idea behind *Docker for CoCoME* is to create a platform independent version which does not require any manual installation by a person apart from Docker itself.

3 Evolution Scenario

Within several steps, CoCoME became more complex. This includes further software that has to be installed to run CoCoME.
The main objective in this evolution scenario is to provide a platform independent version of CoCoME that does not require any preconditions like installing

or updating software. In this case, Docker is a suitable alternative.
By using Docker, CoCoME can be instantiated on any device without installing additional software.

4 Implementation Overview

Implementing the *Dockerfile* turned out to be the main part of this evolution scenario. It defines the way in which the technology stack is extended. The installation is described in detail in section 5.
Briefly, we are using a Docker container that provides a full Linux distribution and deploys all needed software to accomplish the installation of CoCoME as described in the official CoCoME installation guide[1].

5 Description

As shown in Fig. 1, the changes are affecting the technology stack by adding additional layers. More detailed, the given CoCoME Stack is moved into the Docker Deamon, which runs a Linux distribution. As mentioned before, Glassfish and the JVM are still part of the CoCoME Stack.

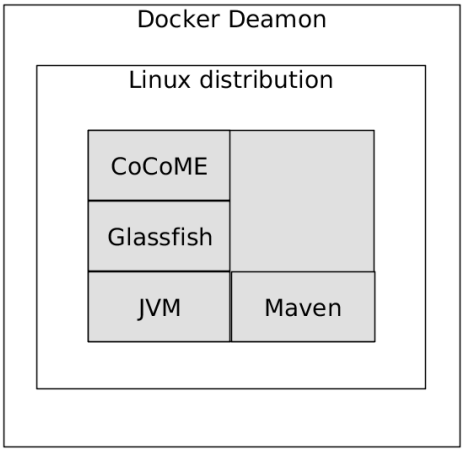

Fig. 1. Extended technology stack CoCoME

The Dockerfile defines an environment based on the latest version of Ubuntu 16:04. Maven, Git and Java are installed on top of Ubuntu using Ubuntu's default package manager.

[1] `https://github.com/cocome-community-case-study/cocome-cloud-jee-platform-migration/blob/master/cocome-maven-project/doc/Deployment%20Setup.md`

Git has two purposes: On the one hand it is used to download the most recent version of CoCoME. On the other hand, it is used to download a prefabricated version of Glassfish that already includes domains and other adjustments required for CoCoME. Java is required by Glassfish and CoCoME as they need the Java Virtual Machine. Maven is needed to deploy the latest version of CoCoME onto the provided Glassfish servers.

6 Deployment

During the development, it was decided to implement and to provide two different versions. The first version always pulls the most recent CoCoME source code from GitHub, downloads the entire dependencies with maven, compiles and builds the project and finally, deploys CoCoME on the Glassfish servers. As a consequence, creating and starting a Docker Container takes about one hour.
In contrast, the second version only pulls a prefabricated version of CoCoME from GitHub. Therefore, pulling the source code up to building the project is skipped. As a consequence, Maven does not have to be included in the technology stack. Solely, deploying CoCoME on the glassfish server is necessary.
This reduces the deployment time to a few minutes but has a disadvantage: The prefabricated version is updated manually. Therefore, it is sometime not the most recent version.
By providing both, a fast deploying version and a current version, the user can choose whats the best for its situation. As shown above in figure 2 the docker Container contains five different Glassfish servers. In particular they are called *WEB*, *ENTERPRISE*, *STORE*, *REGISTRY* and *ADAPTER* and correspond to the given by the CoCoME deplyoment setup. By default, Glassfish provides a Derby DB that is connected to the server Adapter using Java Database Conectivity (JDBS) interface.
As mentioned before, CoCoME is deployed inside the docker container on the same way it is usually deployed. This means the maven generated archive files *cloud-web-frontend*,*enterprise-logic-ear*,*store-logic-ear cloud-registry-sevice* and *service-adapter-ear* are deployed to the servers with the following assignment:

Server	Deployment file
WEB	cloud-web-frontend
ENTERPRISE	enterprise-logic-ear
STORE	store-logic-ear
REGISTRY	cloud-registry-service
ADAPTER	service-adapter-ear

Fig. 3. Assignment of archive files to Servers

Fig. 3 demonstrates the assignment between the archive files and the servers as it is implemented and also recommended by the CoCoME deployment guide.

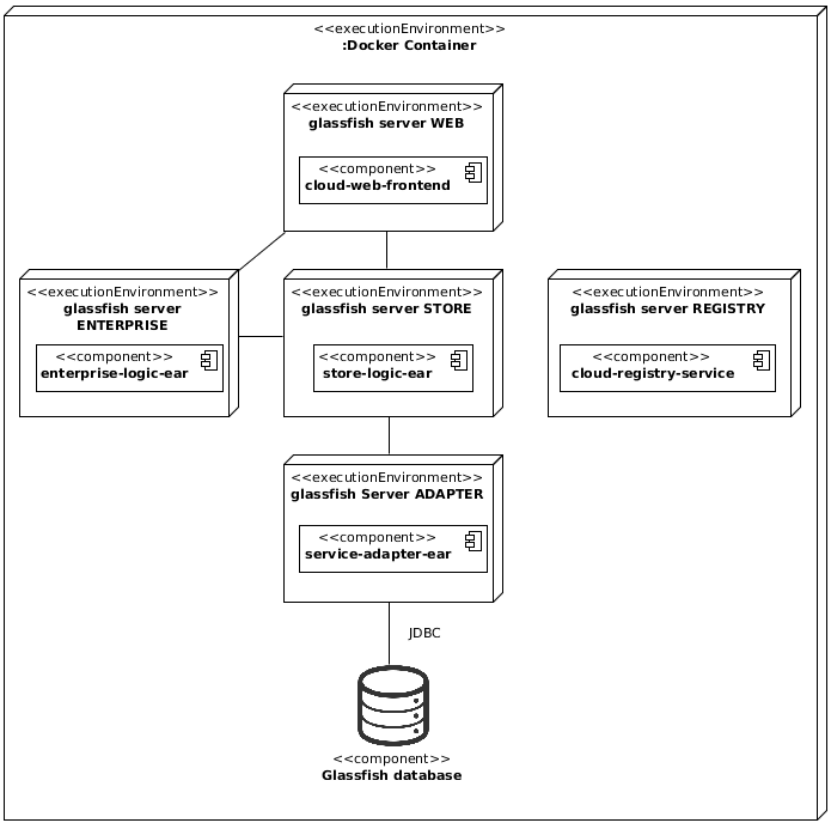

Fig. 2. Deployment diagram CoCoME

This information is also represented in Fig. 2. The two different Docker versions both use this assignment. In addition, the fast version can be extended by the Pickup Shop[2]. This extension runs inside a separate container which is shown in Fig. 4. It basically uses the same technology stack. As shown in Fig. 4, this container provides only one Glassfish server.

[2] `https://github.com/cocome-community-case-study/cocome-cloud-jee-web-shop`

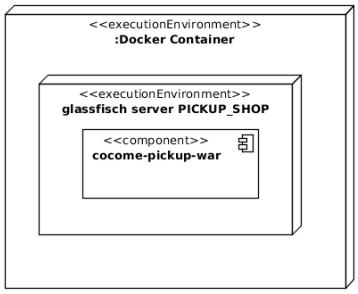

Fig. 4. Deployment diagram CoCoME Pickup Shop

Server	Deployment file
PICKUP_SHOP	cocome-pickup-war

Fig. 5. Assignment archive files to Servers

To control the start of both containers, precisely the CoCoME and the Pickup Shop, another specific file is needed: the Docker Compose file. It ensures that the CoCoME Container is active before the Pickup Shop container starts. This is necessary as the Pickup Shop requires a running instance of CoCoME to register itself.
Whereas CoCoME does not require the Pickup Shop, the inversion is not correct. Both containers need to communicate with each other. By default, docker prohibits any outgoing and ingoing communication from an in a container. This is solved by opening specific ports through which the communication is possible. Which ports the containers can use is specified in the Docker Compose file as well.