DM848: Microservice Programming

Sandboxed Interpreter for Interactive Code Execution Using Docker

Anders Busch, anbus12@student.sdu.dk

June 24, 2016

Contents

1 Introduction

When learning a new programming language, it is often useful to have a environment where the learner can execute code, quick and easy without having to worry about breaking the execution environment or installing new software locally.

The goal of this project was to create a cloud server for executing JOLIE programs through the browser; and to embed the cloud server in a virtual environment provided by the virtualization library DOCKER.

To achieve this goal, we had to solve the following problems:

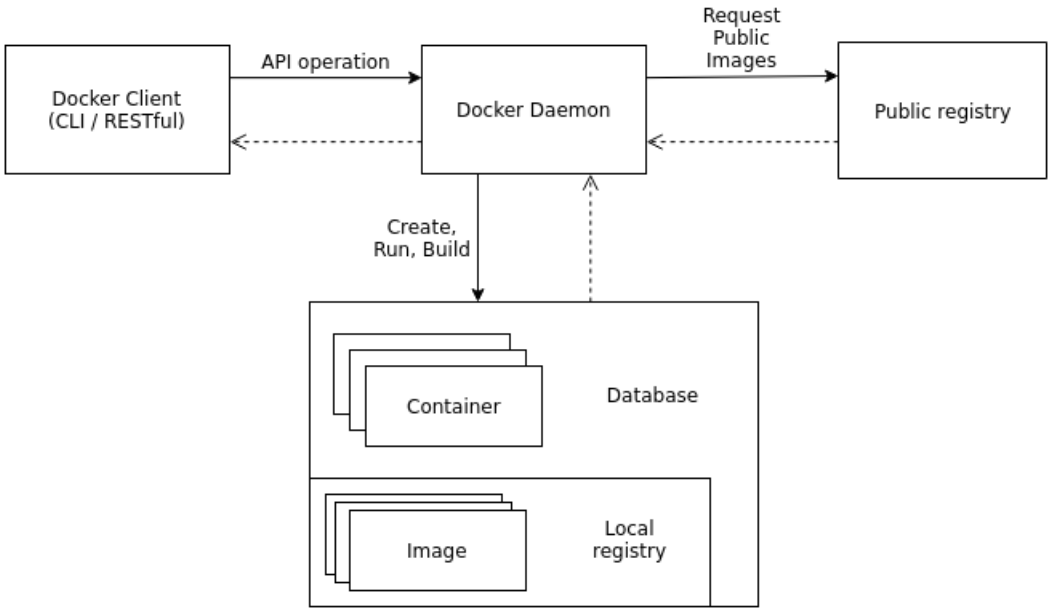
1. Create an DOCKER image for running JOLIE code in a DOCKER container. This image helps automating the process of creating new containers and is available on-line[1] for everyone interested in using DOCKER in conjunction with JOLIE.
2. Produce a service for handling the setup of the DOCKER container. As our execution environment uses JOLIE to communicate between services, we opted to create a service that handles the communication with the DOCKER CLI.
3. Make a service that can evaluate JOLIE within the DOCKER container. For our evaluator to work, we have decided to use a JOLIE service to evaluate sent JOLIE code.
4. Assemble a front-end service for presenting and handling of the user interface. Here we opted for a simple JOLIE-based HTTP server that serves our user interface.

2 Preliminaries

2.1 Docker

The DOCKER application is a lightweight virtualization technology that focuses on creating virtualization of applications in microservice manner.[1]

[1] https://hub.docker.com/r/ezbob/jolie/



DOCKER application architecture

Architecture DOCKER uses a server-client architecture to communicate between the client, and the underlying server, the DOCKER daemon. The client, in this case, is a front-end program that exposes some interface to the user (Command Line Interface or a RESTful API), and uses this to communicate with the daemon (the server). The server is the main workhorse program that builds, executes and maintains the different DOCKER containers.

Containers A DOCKER container is the virtualization environment that contains a running application, and as such can be thought of an microservice or a instance of a class (using Object Oriented terminology) that the DOCKER daemon maintains. Container provides isolation for the running code and uses a layered file system called the union file system inside the container.[1] This union file system keeps track of changes much in the same way version control systems, such as GIT, does; namely by keeping track of writes and only write to the file system once an action has been committed. Containers are build from images.

Images A DOCKER image, is an read-only minimal-size file that can be used to build new containers via the DOCKER daemon. These images make it possible to share different container-setup with different users, and can be created either using an existing DOCKER container or using a special file called a DOCKER file.

Registries Images can either be fetched from a local repository called the **local registry** or remotely from a repository called the **remote registry**. DOCKER provides a standard remote registry service[2] for fetching images such as an UBUNTU linux image and much more.

Volumes A volume is a way of creating a persistent data volume within a container. A volume is created separately from any containers, initialized with any container and can be mounted to one or more containers for sharing of persistent data.

Docker Engine The DOCKER engine is the part of the DOCKER daemon that handles the running containers. The engine communicates directly with the host operating system, on behave of the containers. This is different to most other virtual environments, where a hypervisor is employed support the virtual operating system running in the virtual environment.

3 Technical Description

4 Related Work and Discussion

Although we partly achieved what we set out to create; there are some limitation in the current implementation that can be expanded upon in future revision of this execution tool, namely the support for inter-service communications, better error explanation and dynamic linting[3] of the program code.

References

[1] C. Anderson, "Docker," *IEEE Software*, vol. 32, no. 3, p. 102, 2015.

[2] https://hub.docker.com/

[3] https://en.wikipedia.org/wiki/Lint_%28software%29