Cloud Systems (H/M) – Lab 3 – Containerization

Last Changed: 25/01/2025

The primary goal of this lab is to gain practical experience with OS-level virtualization (containers), specifically using Docker (creating and using Docker files and using Docker compose). Same as the previous labs, the lab builds towards the first assessed exercise (in which you might want to benchmark containers).

As before, we recommend working in small groups (of 2-4 students), allowing you to split up tasks and explain what you did to each other.

1. Prerequisites

We suggest working on your personal computer using Linux (either natively or in a virtual machine) or using GCP, the public cloud platform of Google (see Lab 2).

If you have not installed docker yet, have a look at the official documentation, e.g. https://docs.docker.com/engine/install/ubuntu/, for instructions.

2. Creating and Running Containers

2.1 Containerizing a Microbenchmark

Containerize a SysBench using Docker. For this, write your own Dockerfile and create an image.

A Dockerfile for SysBench could look like:

```
FROM ubuntu:latest

ENV cpu_arch=amd64

RUN apt-get update -y && \
    apt-get install -y sysbench && \
    apt-get clean

CMD ["/usr/bin/sysbench"]
```

Build an image from this Dockerfile (using the docker build command, e.g. `docker build -t sysbench-ubuntu .`) and run the container (e.g. `docker run --rm sysbench-ubuntu`). This should show you the SysBench version, confirming SysBench is running in your container.

You can run the container with a command. For example, to test CPU performance, you could run: `docker run --rm sysbench-ubuntu sysbench cpu --cpu-max-prime=3000 --threads=4 run`

We can limit the CPU usage of this container using cgroups. See https://docs.docker.com/engine/containers/resource_constraints/#cpu for an overview of options available through the Docker command and try different CPU limits (e.g. `--cpus=2`, `--cpus=1`, `--cpus=0.5`).

Practice task: Run the container with different CPU limits as well as threads and look at the impact this has on the benchmark performance (i.e. events per second).

2.2 Running a Multi-Container Application on Docker Compose

Docker compose, https://docs.docker.com/compose/, is a tool to run multi-container applications on a single node. That is, it can be used to run an application consisting of multiple cooperating services, such as a web application that uses a server, a database, and perhaps also some proxies.

You might need to install docker compose. See https://docs.docker.com/compose/install/ for instructions.

A simple example for a multi-service, multi-container Web application is available at https://github.com/docker/awesome-compose/tree/master/nginx-nodejs-redis. The repository contains a compose file, docker files, and system configurations to serve a Node.js application with Nginx proxy and Redis database. The application counts and displays visits to the website, storing the count in the database.

Read through the files in the repository, then deploy the application, following the example given in the README.md (under the “Deploy with docker compose” heading, using `docker compose up -d`).

Note that there are also instructions on how to stop the entire multi-container application with one docker-compose command (`docker compose down`).

Practice task: Once the application is running, use curl, as suggested in the README, to test the application (from your host machine, without browsing the website).

Note how starting this application the first time takes longer than the subsequent starts, when the required images have already been pulled and built.

3 Open-Ended Task Towards AE1: Container Performance Benchmarking

There should not be any substantial overhead for running CPU-intensive or disk-intensive applications in containers on Linux. You can test this yourself, for example, using Sysbench and/or fio, comparing an ordinary process and a container side by side.

Other measurements around containers that might be interesting are how long it takes to build an image from a Dockerfile and how long it takes to start the container. The standard Linux command-line tool `time` might help to take some measurements.
Similarly, it might be interesting to look at the size of a docker image. You can use `ls -l` to get the file size of an image.
Of course, the build time depends on the Dockerfile and the size of the resulting image. Moreover, build, boot, and execution all depend on your particular host or virtual machine.

Starting a container can take slightly longer than starting an ordinary process, especially if there is considerable configuration, such as for networking.