FAKULTÄT FÜR
INFORMATIK UND MATHEMATIK
Skalierbare Datenbanksysteme
Prof. Dr. Stefanie Scherzinger
Stefan Klessinger

UNIVERSITÄT PASSAU

This exercise sheet is based on material from the lecture "Reproducibility Engineering", joint work with Wolfgang Mauerer (OTH Regensburg).

Reproducibility Workshop at TU Dresden, June 2022

On-Hands Exercises

We prepared material for this session in a GitHub repository. To get started, clone the repository:

```
$ git clone https://github.com/sdbs-uni-p/reproducibility-workshop
```

1 Image Comparison

Inside the repository you will find the folder `image_comparison`, containing two images: `fox.jpg` and `fox_secret.jpg`. The latter has a secret message embedded. We will compare these images using different approaches:

1. Open the images and compare them visually.
2. Build and run the docker container in interactive mode and change directory to `res`:

   ```
   $ docker build -t img_comp .
   $ docker run -it img_comp
   $ cd res
   ```

3. Perform a bitwise comparison using shasum:

   ```
   $ sha256sum fox.jpg
   $ sha256sum fox_secret.jpg
   ```

4. Execute the R script `img-corr.r` that calculates the correlation coefficient between `fox.jpg` and `fox_secret.jpg`.

   ```
   $ Rscript img-corr.r
   ```

2 ReproTest

A central aspect in ensuring the reproducibility of builds is building in different environments. ReproTest is a tool that helps with testing builds, by building twice, in different environments and checking whether bitwise identical binaries are produced.

2.1 Extending Docker

We start by extending a provided Dockerfile to install ReproTest:

1. Change directory to `reprotest`.
2. Extend the `Dockerfile` so that ReproTest is automatically installed and necessary dependencies are added to the container. Those are:

   `diffoscope`, `disorderfs`, `faketime`, `reprotest`

   Note that there is a convention to list packages in alphabetical order.
3. Build a Docker image using the `Dockerfile`. While the image is building, check out the documentation on ReproTest.

   $ docker build -t reprotest .
4. Run a container with that image in privileged interactive mode.

   $ docker run -it --privileged reprotest

   It is necessary to run in privileged mode because ReproTest requires elevated permissions, e.g., for setarch.

2.2 Using ReproTest and Diffoscope

Next, we investigate the reproducibility of our code with ReproTest:

1. Test `hello.c` using ReproTest:

   $ reprotest 'gcc hello.c -o hello' hello
2. Which part of the source code has an impact on reproducibility?

   Change `hello.c`, so that the build is deterministic and repeat the test with ReproTest.
3. Compiler flags may impact the resulting binary. Build `hello.c` (with the previous modifications) with and without optimization and compare the results:

   $ gcc hello.c -o hello
   $ gcc -O hello.c -o hello_opt
   $ diffoscope hello hello_opt
4. Compiling with debug information also causes non-deterministic builds:

   $ reprotest 'gcc -g hello.c -o hello' hello

3 Setting up a Benchmark

In this exercise, we modify a reproduction package for running a benchmark on SQPolite, a modified, more polite version of SQLite. This reproduction package uses the TPC-H benchmark, which we replace with a new benchmark called `clessbench`.

3.1 Setup

First, we set up a docker container with the tools needed for this exercise. Build a Docker container from the Dockerfile provided in `benchmark` and run it interactively. While the image is building, familiarize yourself with the Dockerfile.

3.2 Adapting the Dispatcher

Now, we will modify the dispatcher script. This dispatcher runs the TPC-H benchmark which we will replace with an alternative and proprietary benchmark, clessbench.

1. Open the dispatch script, `dispatch.sh`.

   Hint: You can also edit the file at `benchmark/scripts/dispatch.sh` on the host system. Remeber to rebuild the container afterwards.

2. Which changes do we need to make to the parameters in order to use clessbench instead of TPC-H?

   Hint: clessbench has three parameters: scale factor (a double, e.g. 1.0), random seed (true/false), query-type (read/insert/mixed).

3. Which additional changes do we need to make to the script?

3.3 Benchmarking SQPolite

After setting up the Docker container and modifying the dispatcher script, we can now run benchmarks on SQPolite.

We start by running the benchmark locally using our dispatch script.

1. Run `dispatch.sh` with different parameters.

2. Verify that your results are stored on the paths we specified.

3.4 Processing the Results

After performing experiments, we need to visualise our results. We use knitr to create a PDF document with visualisations.

1. Modify `results/results.Rnw` so the correct results are loaded.

2. Convert `results.Rnw` to PDF:

   $ Rscript -e "require('knitr'); knit2pdf('results.Rnw')"

3. Copy the PDF from the container to the host file system by running the following command outside the container:

   $ docker cp <container_ID>:/home/repro/results.pdf .

   Hint: To find the ID fo your container, run `docker ps`

4. Inspect the PDF.