Reproducibility with git, Docker, Github actions

I'm assuming you are doing the job with another colleague (ie you're a group of 2 students)
We will revisit and reuse the work of previous lab session.
The idea is to concretely practice git, Docker, and Github actions (or Gitlab runner).

I'm not giving a complete course or doing a lab session on these technologies (assuming you knew them a bit)… However, I'm giving some concrete explanations and pointers to do it by yourself. And I'm obviously here to assist you!

Hint: take as much time as needed to install tools and be comfortable with the underlying technologies… the lab session is long and time-consuming, but there are several "bonus" questions. What is mandatory is that you're able to fully reproduce the associativity computation in a Github/Gitlab

Reproducible?

- Create a git repository with Github or Gitlab
- Push your code about associativity computation with floating point
- Refactor your code in such a way anyone can run your code and automatically produce an answer to the question "How often x+(y+z) == (x+y)+z)"
- Write as precise instructions as possible to reproduce your results in a README.md, including the answer someone is supposed to get
- Share your git with others (in the sheet https://lite.framacalc.org/xvmr5qw64a-aafk column "Git for the lab session")
- Try to reproduce another group's result by taking their git (take the group just after in the sheet… if you're the last group in the sheet, take the first group)
- Report on some difficulties through Github/Gitlab issues
- (consensus) Is the result you get equal to your result?

Moving to Docker

Getting back to *your* git…

Create a Dockerfile to make your application runnable

"It works on my machine" is sometimes a fallacy.

Docker https://docs.docker.com is a platform that enables developers to package, distribute, and run applications in isolated environments called containers, which can be easily deployed and run consistently on any system.

Docker holds the *promise* of providing an environment that is the same on every machine. With Docker, the hope is that you can easily move your application from one machine to another, be it your laptop, a server, or a cloud provider.
In your lab session, you'll use Docker to create an isolated environment where your program will compute how often (x + y) + z == x + (y + z) holds true, ensuring that it runs the same on every machine.

Some basic terminologies

A **Dockerfile** is a configuration file that contains a set of instructions for building a Docker image. It typically specifies:

- the base image,
- working directories,
- dependencies,
- the commands needed to set up and run your application.

It's like a recipe that tells Docker how to create an environment for your code to run.

A **Docker image** is a snapshot of an environment created based on the instructions in a Dockerfile. It contains everything needed to run an application: the operating system, the code, and the dependencies. An image is immutable, meaning it doesn't change after it is created.

A **Docker container** is a running *instance* of a Docker image. While the image is the blueprint, the container is the active version of that blueprint, where the application runs in an isolated environment. Containers are lightweight and fast because they share the host operating system's kernel, unlike traditional virtual machines.

Your goal is to create a Dockerfile in such a way a container can be run and compute the answer

1. Choose a Base Image

Select a base image that corresponds to the programming language you are using for the project.

- Examples:
    - For Python: `python:3.9-slim`
    - For Node.js: `node:16`
    - For C: `gcc:latest`

Example (Python):

```
FROM python:3.9-slim
```

2. Set the Working Directory

Create a working directory inside the Docker container where your code will reside.

Example:

```
WORKDIR /app
```

3. Copy Application Code

Copy your source code into the container's working directory. This will make your program available inside the container.

Example:

```
COPY . /app
```

4. Install Dependencies (if needed)

Install the necessary dependencies for your program. This could be Python libraries, Node.js packages, or system libraries.

- **For Python:** Ensure you have a `requirements.txt` file with the required packages.
- **For C:** Install any necessary libraries or compilers.

Example (Python):

```
RUN pip install -r requirements.txt
```

Example (C):

```
RUN apt-get update && apt-get install -y build-essential
```

5. Write and Run the Program

Configure Docker to run this program when the container starts.

Example (in Python):

```
CMD ["python", "associative_property.py"]
```

Example (in C):

```
RUN gcc -o associative_property associative_property.c
CMD ["./associative_property"]
```

Steps to Build and Run Your Docker Image

Build the Docker Image:

```
docker build -t associative-property .
```

Run the Docker Container:

```
docker run associative-property
```

Final step: Push and update the instructions to reproduce your result with Docker

Automate further

Use Github actions or Gitlab runner *with Docker*.

NEWS: all groups should move to Github and Github actions (there are too much issues with Gitlab runners at INSA right now)

Please intensively refer to documentation:
https://docs.github.com/en/actions/sharing-automations/creating-actions/creating-a-docker-container-action
~~https://docs.gitlab.com/runner/install/docker.html~~

in such a way the computed result is serialized/pushed in a file answer_associativity.txt of your git repo (hint: check that you have write permissions https://github.com/ad-m/github-push-action)
https://github.com/actions/checkout?tab=readme-ov-file#push-a-commit-using-the-built-in-token

Same for banking

Using the same git, redo the same steps (instructions, Dockerfile, etc.) for the banking problem (this time, store the answer in answer_banking.txt)

Reproducing and replicating slides…

Sylvie Boldo talk :
https://www.college-de-france.fr/fr/agenda/seminaire/semantiques-mecanisees-quand-la-machine-raisonne-sur-ses-langages/arithmetique-des-ordinateurs-et-sa-formalisation
(also look at the slides/PDF of the talk)

Reproduce a subset of everything (figures, tables, etc.) that has been said in the talk:

- Push the code into the repo
- Use Docker and Github actions/Gitlab runner to synthesize a answer_boldo.pdf

Replicate (change programming language, compiler flags or versions, etc.)

- Push the code into the repo
- Use Docker and Github actions/Gitlab runner to synthesize a answer_repl_boldo.pdf
- Is it consistent with the original claims of the talk?