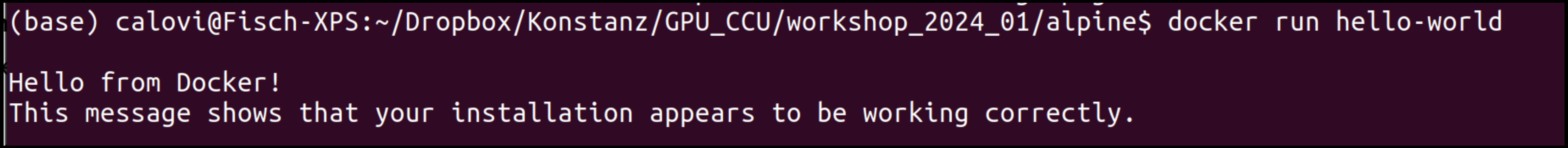
Docker

- Docker is a virtualization software
  - It creates a standalone VM (container) that is able to run cross platform with little efficiency loss (depends on the application, but fair approximation)
- Think of Conda environments, but completely separated from your system
  - Had you ever had to run a software that only runs in Ubuntu xx and uses an outdated package that conflicts with other libraries and softwares?
- Docker containers are versionable
  - Each new addition just gets added to the last version instead of having to redo it

Learning by doing it

- First thing to check is if docker is running properly in your machine

  docker run hello-world



```
(base) calovi@Fisch-XPS:~/Dropbox/Konstanz/GPU_CCU/workshop_2024_01/alpine$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

  After, type

  docker images

- To delete, type

  docker rmi hello-world

- Most likely it failed, try:

  docker rmi -f hello-world

How to create a docker image?

- Extract folder Alpine into an appropriate location
- Open the file Dockerfile
- Within a terminal, go inside that folder
- Type:

```
docker build -f Dockerfile -t alpine .
```

Running a container

- Type:

```
docker run alpine
```

- Open the file Dockerfile_2
  - What are the differences?
  - What do you imagine it will happen when we run it?
- Now build it and run it:

```
docker build -f Dockerfile_2 -t alpine2 .
docker run alpine2
```

- Was it what you expected?

Running a container

- Type:

```
docker run alpine
```

- Open the file Dockerfile_2
  - What are the differences?
  - What do you imagine it will happen when we run it?
- Now build it and run it:

```
docker build -f Dockerfile_2 -t alpine2 .
docker run alpine2
```

- Was it what you expected?
  - Aren’t containers stateless?

RUN/CMD commands

- Run commands are performed during the building of the image
  - The base state of our container already contained the 2 files
- Only the last CMD command will be run
  - The first one in the Dockerfile_2 was ignored

Stateless version

- Open file Dockerfile_3, and then build it and run it

  ```
  docker build -f Dockerfile_3 -t alpine3 .
  docker run alpine3
  ```

- Note the difference between the CMD syntax of the first container to this one
  - Exec form (CMD ["command", "param"]) Directly executes commands without a shell, enhancing signal responsiveness and process control
  - Shell form (CMD command) Executes commands via a shell, enabling complex scripting such as command chaining and variable expansion
- For complex commands, better to create a bash script and use CMD to run it, e.g.

  ```
  CMD ["/usr/local/bin/start-notebook.sh"]
  ```

Docker is versionable

- Open file Dockerfile_4, examine it and build it

  ```
  docker build -f Dockerfile_4 -t apine4 .
  ```

- Quite a few more packages were added, now uncomment line 7 (`RUN apk --no-cache add git`) and build the file again
  - Not everything was rebuilt, git was just appended to the image

Docker is versionable

- Open file Dockerfile_4, examine it and build it

  ```
  docker build -f Dockerfile_4 -t apine4 .
  ```

- Quite a few more packages were added, now uncomment line 7 (`RUN apk --no-cache add git`) and build the file again
  - Not everything was rebuilt, git was just appended to the image
- Now move line 7 before line 6 and build it again
  - It now had to rebuild package “feh” as well

Docker is versionable

- When building complex containers it is worth to using RUN many times in order to have more saved states
- If building crashes midway through, all completed iterations of RUN are already cached
  - imagine having to install opencv multiple times because something at the end of your container crashed?

Versioning your containers

- In Docker :latest is just the default version, not exactly the latest version
- Instead of creating multiple images (alpine, alpine2, alpine3), we can create different versions of them
- Type:

```
docker build -f Dockerfile -t alpine:1.0  -t alpine:latest .
docker build -f Dockerfile_3 -t alpine:2.0  .
docker run alpine
docker run alpine:2.0
```

- Tags can be anything and are case sensitive, latest and Latest would refer to different versions

Repositories

- When building an image you can use default packages like
  - FROM alpine:latest
  - FROM quay.io/jupyter/base-notebook
  - FROM nvcr.io/nvidia/tensorflow:21.02-tf2-py3
- But you can also download a pre-packaged container

This will download a 1GB container

```
docker run -p 8888:8888 jupyter/base-notebook
```

- Where the port syntax is: `-p localhost:container`

More complex example

- Open file Dockerfile_Jupyter
  - (I did some last minute trimming, might not work from scratch)

Logging in the container

- You have a container, and you want to get inside that VM, type:

  ```
  docker run -d --name running_alpine alpine tail -f /dev/null
  ```

  - -d is to keep the terminal free (detached)
  - --name is to ensure you give an specific name
- Type

  ```
  docker ps
  ```

  ```
  docker exec -it running_alpine /bin/sh
  ```
- Now you are actually inside the pod, and while it is active you can perform changes to it

Uploading your container

- To upload your container you need to be logged in a repository and have permission to push (upload) it
- Before pushing, one needs to update the image name to have the address of the repository, i.e.

```
docker tag ccu-workshop-jupyter
ccu-k8s.inf.uni-konstanz.de:32250/daniel.calovi/ccu-workshop-jupyter

docker push ccu-k8s.inf.uni-konstanz.de:32250/daniel.calovi/ccu-workshop-jupyter
```

- This will not run for you, just an example for later

Questions/Lunch Break?