Capturing Computational Environments

20/01/2022

Georgios Fotakis

Institute of Bioinformatics
Medical University of Innsbruck

MEDIZINISCHE UNIVERSITÄT INNSBRUCK

A motivational example

Why should I use computational environments to begin with?

- You have been working on a project for weeks, using Tensorflow v1.3, and you finally got your code working! No errors, you get the expected results and life is beautiful.
- A few days later you start a new project that requires Tensorflow v2.4. You eagerly upgrade to the newest TF version and keep working (hard) on the project.
- Sometime later you decide to revisit the first project.
- You run the code and you get 10 errors, 5 dependency issues and 108 conflicts.

What happened there?
The newer version of TF no longer supports vital functions that were used extensively in your initial project!

Capturing Computational Environments

- A computational environment is a set of packages that can be used in one or multiple projects.
- Computational environments managers can be broadly split into two categories:

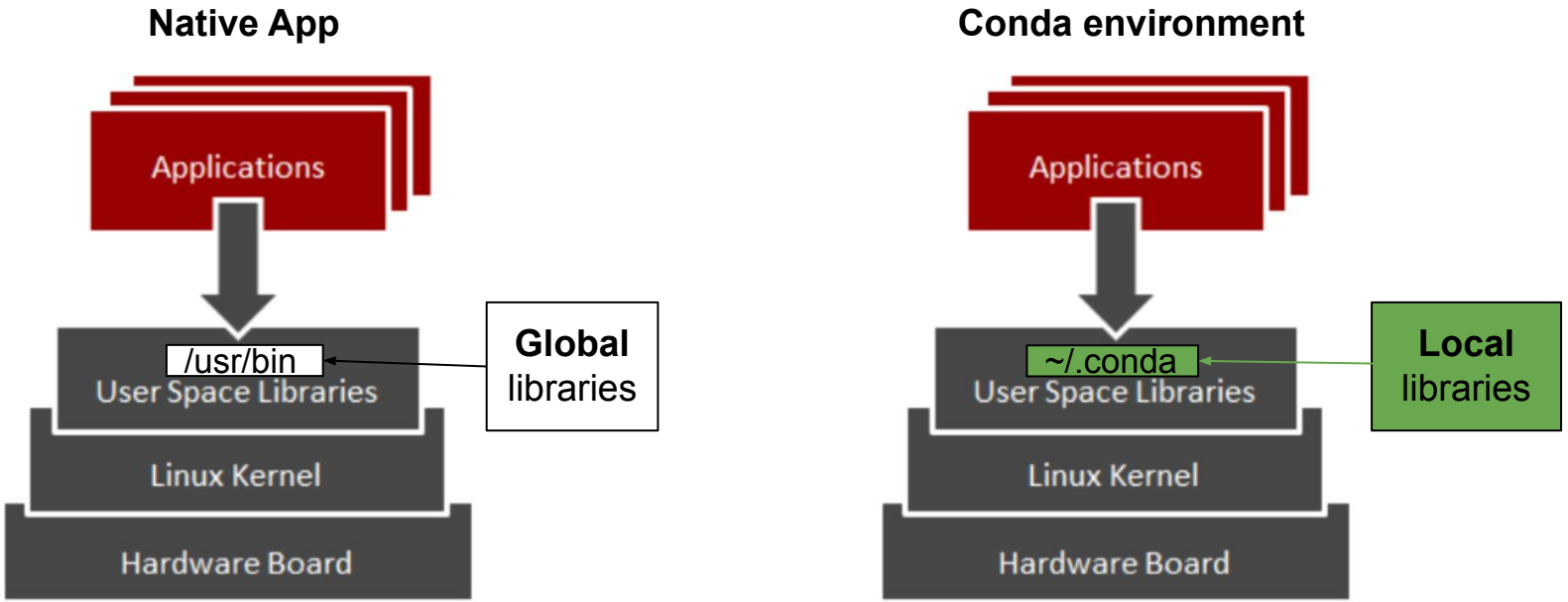
1. **Package Management Systems:**
   They capture only the software and its versions used in an environment (conda, miniconda, venv, mamba, micromamba).
2. **Containers:**
   They replicate an entire computational environment - including the operating system and customised settings (VM, Docker, Singularity, Podman).

Package Management Systems

Why is conda useful?

- With conda, you can create an **isolated** environment for your project.
- Conda will automatically try to **solve** dependencies and **resolve** conflicts.
- Heavily supported by **Nextflow** and has a Bioinformatics dedicated channel (**bioconda**).
- **Miniconda**: a minimal conda executable, with enough functionalities to bootstrap a fully functional conda environment.

Environment isolation



Managing Conda environments

There are two ways to create a conda environment:

1. **Using a YAML file:**
   - YAML stands for “**Y**AML **A**in’t **M**arkup **L**anguage” (do you even recurse bro?)
   - It is a human - readable, data - serialization language

2. **Manually specifying packages through the CLI:**

   $ *conda create -c <channel_name> -n <env_name> [specs]*

1. Creating environments with YAML files

Goal = create a Conda environment with the following specifications:

- Uses the conda-forge channel.
- Has Python 3.7.12 and pandas v1.3.1 as dependencies.
- The environment name should be [**test_env**].

Steps:

1. Create the YAML file (for this example the file name is test_env1.yml).
2. Create the conda environment.
   $ conda env create -f /path/to/test_env.yml
3. Activate the environment.
   $ conda activate test_env
4. Check if the dependencies are installed and working (from within the environment).
   (test_env)$ pip list | grep <lib_name>
5. Deactivate the environment
   (test_env)$ conda deactivate

2. Creating environments via CLI

Goal = create a Conda environment with the following specifications:

- Uses the conda-forge channel.
- Has Python 3.7.12 and pandas v1.3.1 as dependencies.
- The environment name should be [**test_env2**].

Steps:

1. Create the conda environment.
   $ conda create -c conda-forge -n test_env2 python=3.7.12 pandas=1.3.1
2. Activate the environment.
   $ conda activate test_env2
3. Check if the dependencies are installed and working.
   (**test_env2**)**$ pip list | grep <lib_name>**
4. Deactivate the environment
   (**test_env2**)**$ conda deactivate**

Useful commands

1. **Check for current active environments**

   $ conda env list

2. **Activate / Deactivate environments**

   $ conda activate <env_name>

   $ conda deactivate <env_name>

3. **Export / Delete environments**

   $ conda env export --file /path/to/my_env.yml --name <env_name>

   $ conda remove --name <env_name> --all

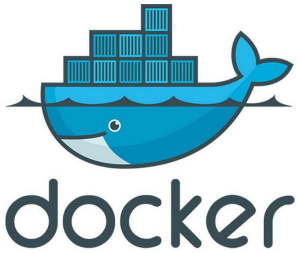
Mamba - A Conda alternative (kinda…)

- Mamba is a reimplementation of the conda package manager in **C++**.
- Parallel downloading of repository data and package files using multi-threading.
- **libsolv** for fast dependency solving.
- **Micromamba**: the miniconda alternative.

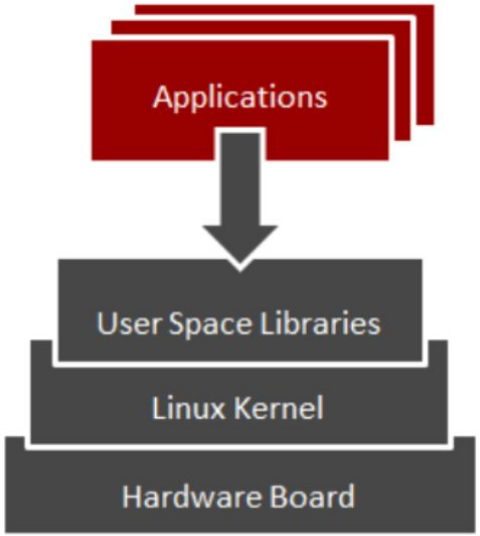
$ *mamba create -c <channel_name> -n <env_name> [list of pkgs]*

Link: https://mamba.readthedocs.io/en/latest/

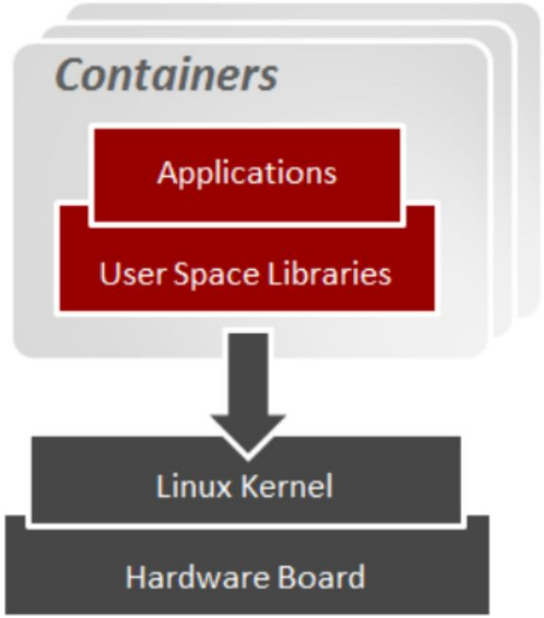
Containers



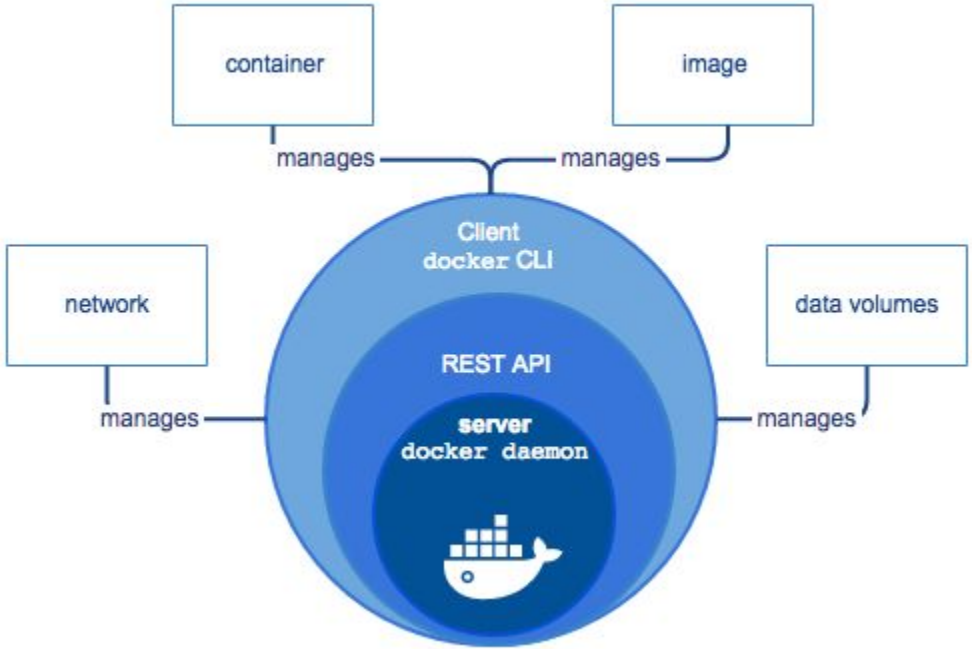
Native App



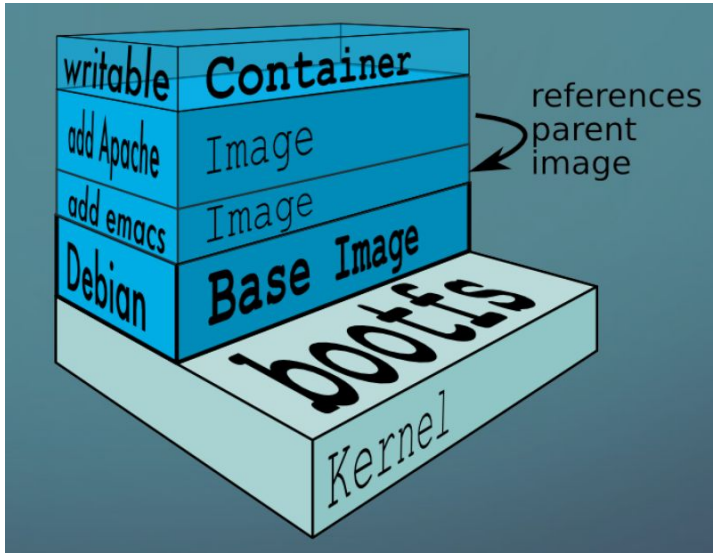
Containerized App

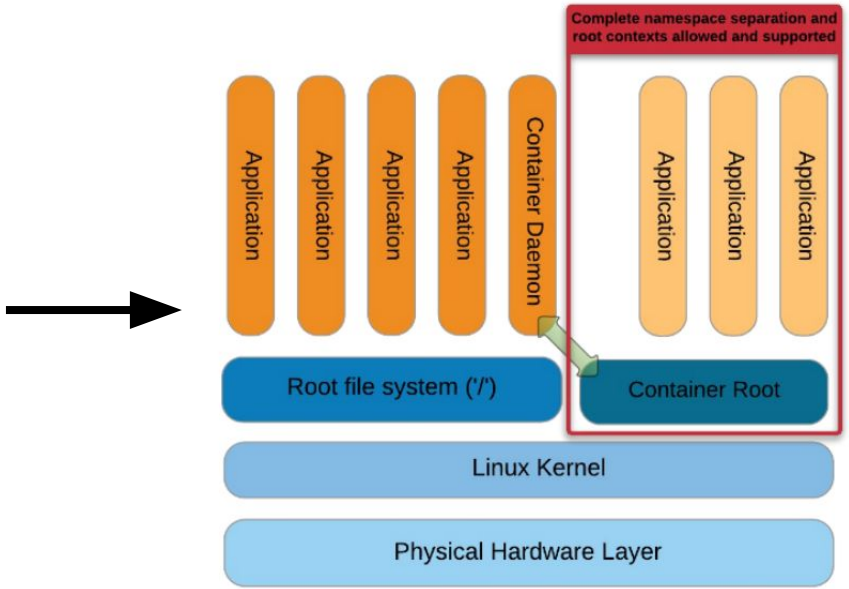


What is Docker?



What is a Docker image?



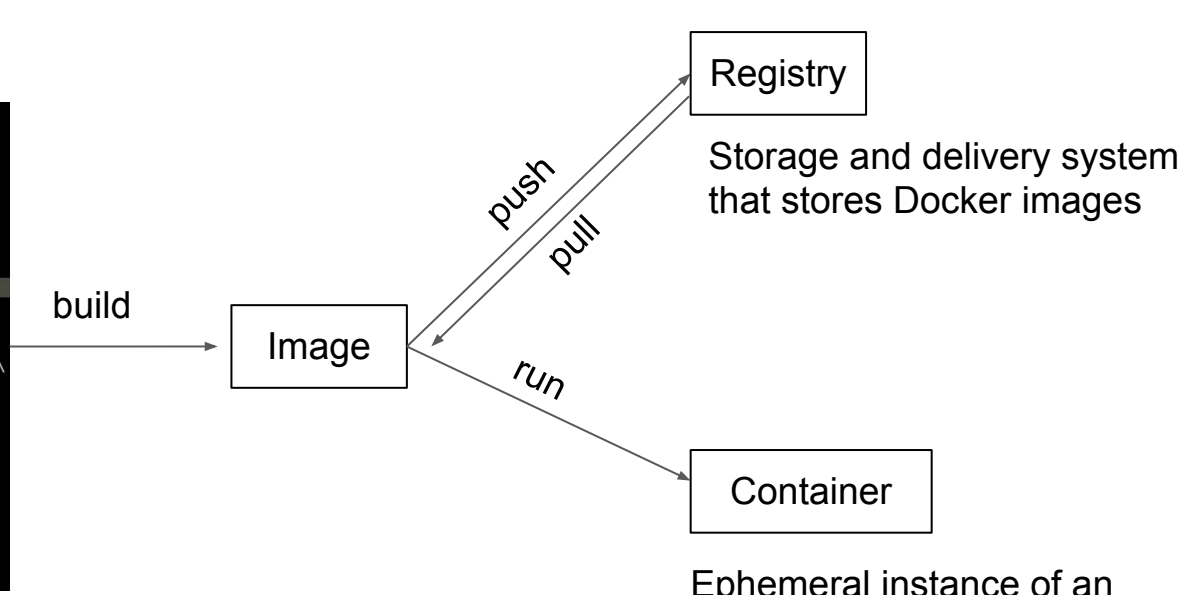


Managing Docker images/containers

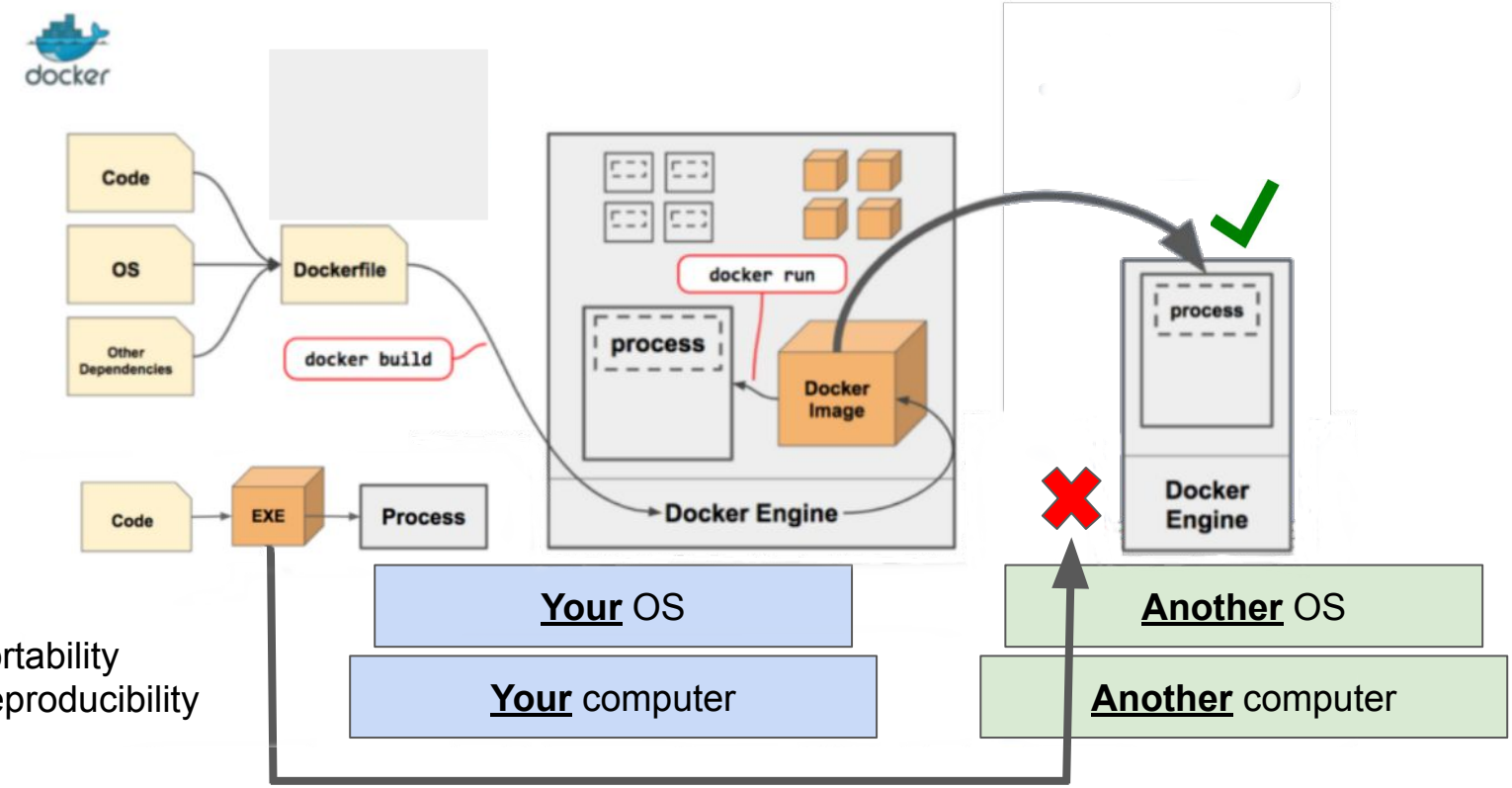
Dockerfile

```
# work from latest LTS ubuntu release
FROM ubuntu:18.04

# set the environment variables
ENV optitype_version 1.3.2
ENV samtools_version 1.2
ENV bcftools_version 1.2
ENV STAR_VERSION 2.7.1a
ENV PATH /home/bin:${PATH}

# run update and install necessary libs
RUN export DEBIAN_FRONTEND=noninteractive && \
    apt-get update -y && \
    apt-get install -y --no-install-recommends r-base \
    ca-certificates \
    libcurl4-openssl-dev \
    libxml2-dev && \
    build-essential \
    curl \
    unzip \
    python-minimal \
    bzip2 \
    zlib1g-dev \
    libncurses5-dev \
    libncursesw5-dev \
```

Plain text file that contains all the commands to assemble the image layers



Why use Docker?



1. Portability
2. Reproducibility

Docker hands-on example

Goal = create a Docker image with the following specifications:

- Uses Ubuntu 18.04 as base image.
- Has all necessary utilities installed (make, gcc, g++, etc.).
- STAR v2.7.10a is installed.
- The image name should be [**test_star**].

Steps:

1. Create the Dockerfile.
2. Build the image via the Docker daemon.
   $ sudo docker build -t test_star /path/to/dir/Dockerfile
3. Launch the container (interactive mode) and mount a specific volume.
   $ sudo docker -it -v /path/to/dir:/path/to/mountpoint run test_star
4. Check if the dependencies are installed and working (from within the container).
   icbi_user@container$ STAR --version
5. Return to base environment.
   icbi_user@container$ exit
6. Push the image to a registry (you would first need to “tag” the image).

So, Docker appears to be the answer to life, the universe and everything!

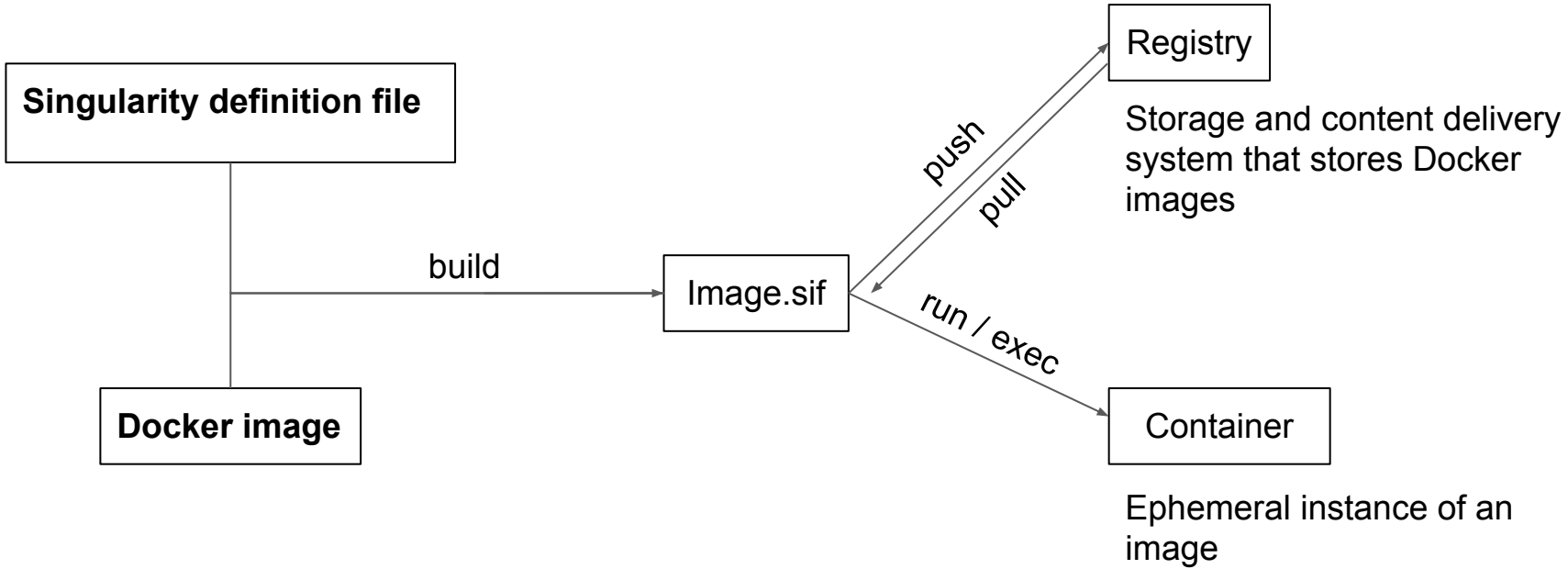
Well… 42

If you ever need to scale beyond your local resources

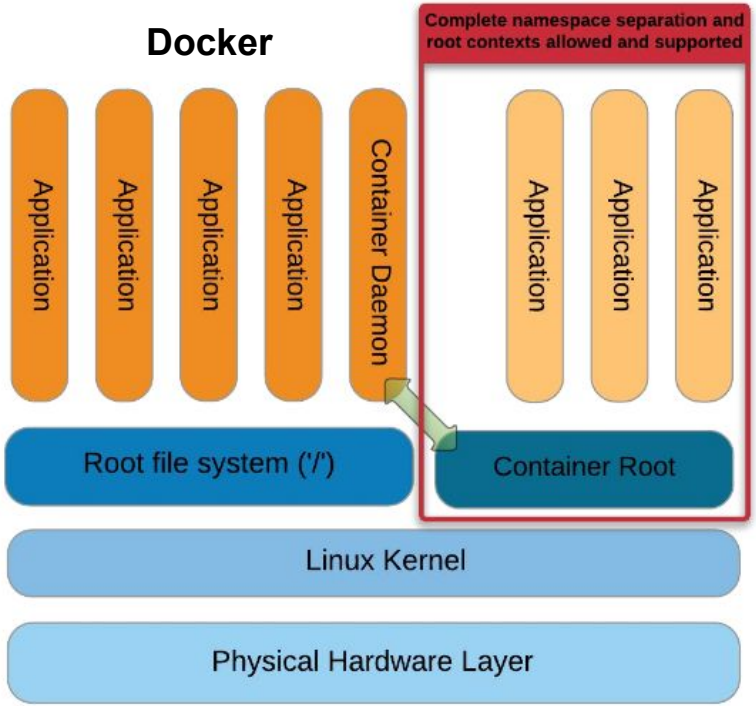
What then?!

Singularity: Containers for science

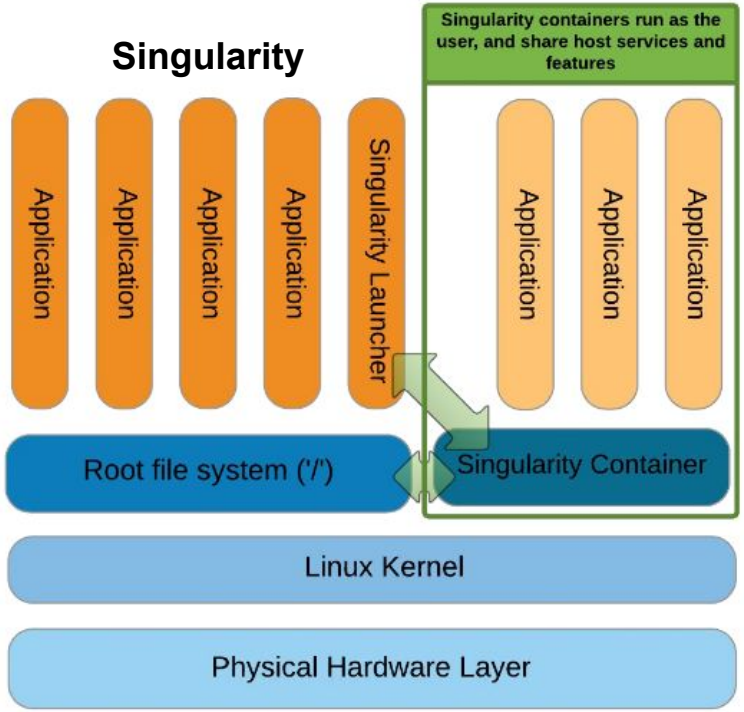
Managing Singularity images/containers



Singularity == Docker ?



There is a minor performance penalty as the kernel must now navigate namespaces.



Much lighter footprint, greater performance potential and easier integration than container platforms, like Docker, that are designed for full isolation.

Security and root escalation

Docker:

- Users running docker commands need to gain elevated system access (sudo).
- Containers start with privileged access (root).

Singularity:

- Users run images without special privileges.
- Any escalation pathways inside the container are blocked.

Podman - A Docker alternative

- Podman is a **daemonless, open source, Linux native tool**.
- Manages the entire container ecosystem using the libpod library.
- Containers start without privileged access (!sudo).
- Reduced overhead when compared to Docker.

Link: https://podman.io/getting-started/

Singularity hands-on example

Goal = create a Singularity image with the following specifications:

- Uses Ubuntu 18.04 as base image.
- Has all necessary utilities installed (git, make, gcc, g++, etc.).
- STAR v2.7.10a is installed.
- The image name should be [**test_STAR.sif**].

Steps:

1. Pull the Docker image from the ICBI registry and build the Singularity image.
   $ singularity build /path/to/test_STAR.sif docker://icbi/test_star
2. Launch the container (interactive mode) and mount a specific volume.
   $ singularity exec -B /path/to/dir:/path/to/mountpoint /path/to/test_STAR.sif bash
3. Check if the dependencies are installed and working (from within the container).
   Singularity> STAR --version
4. Return to the base environment.
   Singularity> exit

Useful Links

- **Conda**

  https://docs.conda.io/projects/conda/en/latest/user-guide/index.html

  https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf

- **Docker**

  https://docs.docker.com/get-started/

  https://dockerlabs.collabnix.com/docker/cheatsheet/

- **Singularity**

  https://sylabs.io/guides/3.5/user-guide.pdf

  https://cs-cheatsheet.readthedocs.io/en/latest/subjects/unix/singularity.html

Practice makes perfect

Goal = create a Docker/Singularity image and perform some basic data processing:

- Uses Ubuntu 18.04 as base image (this is optional, you can choose whatever base image suits your needs better).
- Conda is installed in the image (you can experiment on how to implement this step, but bear in mind that there are base images that come with conda pre-installed).
- Has all necessary utilities installed.
- STAR is installed via the bioconda channel, Pandas and Python are installed via the conda-forge channel (try to do it using the conda CLI and/or a YAML file).
- Launch the image interactively and mount the directory containing your data, the index files, as well as the output directory (you can mount more than one volumes to a container).
- Check if all requirements are met.
- Perform the alignment step using all required files (STAR index files, input FASTQ files, etc.).
- Return the results to the output directory.
- (Optional: Tag and push the image to a registry.)

Rember: Do not try to build the image in one go! Start by experimenting layer by layer until you have the complete image.

Capturing Computational Environments

20/01/2022

Georgios Fotakis

Institute of Bioinformatics
Medical University of Innsbruck

MEDIZINISCHE UNIVERSITÄT INNSBRUCK